STC8G family of Microcontrollers Reference Manual

STC MCU

Technical support website: http://www.STCMCUDATA.com
Date of update: **2022/3/9**

Contents

1 Overview 1
2 Features, Price and Pins 3
2.1 STC8G1K08-38I-TSSOP20/QFN20/SOP16 family 3
2.1.1 Features and Price 3
2.1.2 Pinouts 5
2.1.3 Pin descriptions 8
2.2 STC8G1K08-36I-SOP8/DFN8 family 10
2.2.1 Features and Price 10
2.2.2 Pinouts 12
2.2.3 Pin descriptions 14
2.3 STC8G1K08A-36I-SOP8/DFN8/DIP8 family 15
2.3.1 Features and Price 15
2.3.2 Pinouts 17
2.3.3 Pin descriptions 18
2.4 STC8G2K64S4-36I-LQFP48/QFN48/LQFP32/QFN32 family 20
2.4.1 Features and Price 20
2.4.2 Pinouts 23
2.4.3 Pin descriptions 25
2.5 STC8G2K64S2-36I-LQFP48/QFN48 family 31
2.5.1 Features and Price 31
2.5.2 Pinouts 34
2.5.3 Pin descriptions 35
2.6 STC8G1K08T-36I-TSSOP20 touch key family 39
2.6.1 Features and Price 39
2.6.2 Pinouts 42
2.6.3 Pin descriptions 42
2.7 STC15H family (Traditional STC15 series to enhance the performance of special models) 46
2.7.1 Features and Price 46
2.7.2 Pinouts 47
3 Function pins switch 49
3.1 Register related to function pin switch 49
3.1.1 Peripheral port switch register 1 (P_SW1), for UART1, CCP and SPI switch 49
3.1.2 Peripheral port switch register 2 (P_SW2), for UART2/3/4, I2C and comparator switch 50
3.1.3 Clock selection register (MCLKOCR) 51
3.2 Example Routines 51
3.2.1 UART1 switch 51
3.2.2 UART2 switch 52
3.2.3 UART3 switch 54
3.2.4 UART4 switch 55
3.2.5 SPI switch 57

STC MCU

3.2.6 PCA/CCP/PWM switch 59
3.2.7 I2C switch 60
3.2.8 Comparator output switch 62
3.2.9 Main clock output switch 63
4 Package Dimensions 66
4.1 SOP8 Package mechanical data 66
4.2 DFN8 Package mechanical data (3mm*3mm) 67
4.3 SOP16 Package mechanical data 68
4.4 TSSOP20 Package mechanical data 69
4.5 QFN20 Package mechanical data (3mm*3mm) 70
4.6 LQFP32 Package mechanical data (9mm*9mm) 71
4.7 QFN32 Package mechanical data (4mm*4mm) 72
4.8 LQFP48 Package mechanical data (9mm*9mm) 73
4.9 QFN48 Package mechanical data (6mm*6mm) 74
4.10 LQFP64S Package mechanical data (12mm*12mm) 75
4.11 QFN64 Package mechanical data (8mm*8mm) 76
4.12 Naming rules of STC8G family 76
5 ISP Download and typical application circuit 77
5.1 STC8G seriesISP download application circuit 77
5.1.1 Download using RS-232 converter (no independent VREF pin) 77
5.1.2 Download using RS-232 converter (independent VREF pin, general accuracy ADC), supporting emulation 77
5.1.3 Download using RS-232 converter (independent VREF pin, high accuracy ADC), supporting emulation 78
5.1.4 Download using PL2303-GL, supporting emulation 78
5.1.5 Use general USB to UART tool to download, support ISP online download, and also support emulation 79
5.1.6 Use U8-Mini tool to download, support ISP online and offline download, and also support emulation 80
5.1.7 Download using U8W tool, support ISP online and offline download, and also support emulation 81
5.1.8 ISP Download using Simulate USB directly, only supports ISP download, does not support emulation 82
5.1.9 Power supply control reference circuit of MCU 83
6 Clock, Reset, Power Saving Mode and Power Management 84
6.1 System Clock Control 84
6.1.1 Clock selection register (CKSEL) 84
6.1.2 CLKDIV (Clock Division register) 84
6.1.3 Internal high-speed and high-precision IRC control register (HIRCCR) 85
6.1.4 External Oscillator control register (XOSCCR) 85
6.1.5 Internal 32KHz low speed IRC control register (IRC32KCR) 86
6.1.6 Master clock output control register (MCLKOCR) 86
6.2 STC8G series internal IRC frequency adjustment 87
6.2.1 IRC band selection register (IRCBAND) 87

6.2.2 IRC frequency adjustment register (IRTRIM) 87
6.2.3 IRC frequency trim register (LIRTIRM) 87
6.2.4 Clock Divide Register (CLKDIV) 88
6.2.5 Example of fine-tuning to get a user frequency of 3MHz 88
6.3 System reset 92
6.2.1 Watch-Dog-Timer reset (WDT_CONTR) 92
6.3.2 Software reset (IAP_CONTR) 93
6.3.3 Low-voltage detection reset (RSTCFG) 94
6.3.4 Low-voltage power-on reset reference cuicuit (No need in general) 94
6.3.5 Low-voltage manual button reset reference cuicuit 95
6.3.6 Power-on reset reference cuicuit of traditional 8051 95
6.4 External crystal oscillator and external clock circuit 96
6.4.1 External crystal oscillator input cuicuit 96
6.4.2 External crystal oscillator input cuicuit (P1.6 cannot be used as general I/O) 96
6.5 Clock Stop/Power Saving Mode and System Power Management 96
6.5.1 Power control register (PCON) 96
6.6 Power-down wake-up timer 98
6.6.1 Power-down wake-up timer counter register (WKTCL,WKTCH) 98
6.7 Example Routines 99
6.7.1 System Clock Soure Selection 99
6.7.2 Main Clock Output after being devided 101
6.7.3 Application of Watch-dog Timer 103
6.7.4 User Defined Downloading by Using Software Reset 105
6.7.5 Low Voltage Detection 107
6.7.6 Power Saving Mode 109
6.7.7 Wake up MCU from Power Saving Mode using INT0/INT1/INT2/INT3/INT4 interrupts 111
6.7.8 Wake up MCU from Power Saving Mode using T0/T1/T2/T3/T4 interrupts 114
6.7.9 Wake up MCU from Power Saving Mode using RxD/RxD2/RxD3/RxD4 interrupts 119
6.7.10 Wake up MCU from Power Saving Mode using I2C SDA pin 122
6.7.11 Wake up MCU from Power Saving Mode using power-down wake-up timer 125
6.7.12 Wake up MCU from Power Saving Mode using LVD interrupt 127
6.7.13 Wake up MCU from Power Saving Mode using CCP0/CCP1/CCP2 interrupts 130
6.7.14 Wake up MCU from Power Saving Mode using CMP interrupt 133
6.7.15 Detect the Operating Voltage (Battery Voltage) using LVD 135
7 Memory 140
7.1 Program Memory 140
7.2 Data Memory 142
7.2.1 Internal RAM 142
7.2.2 Program Status Word register (PSW) 143
7.2.3 On-chip extended RAM, XRAM, XDATA 143
7.2.4 Auxiliary register (AUXR) 144
7.2.5 External extended RAM, XRAM, XDATA 144
7.2.6 Bus speed control register (BUS_SPEED) 144
7.2.7 Bit Addressable Data Memory in 8051 145

7.3 Special parameters of memory 147
7.3.1 Read Internal 1.19V Referrence Voltage (from Flash) 149
~~7.3.2 Read Internal Referrence Voltage (from RAM)~~ 152
7.3.3 Read the Unique ID (from Flash) 155
~~7.3.4 Read the Unique ID (from RAM)~~ 158
7.3.5 Read the Frequency of 32K Power-down Wake-up Timer (from Flash) 161
~~7.3.6 Read the Frequency of 32K Power-down Wake-up Timer (from RAM)~~ 164
7.3.7 Read the User-defined IRC Frequency (from Flash) 167
~~7.3.8 Read the User-defined IRC Frequency (from RAM)~~ 172
8 Special Function Registers 175
8.1 STC8G1K08 family 175
8.2 STC8G1K08-8Pin family 176
8.3 STC8G1K08A family 177
8.4 STC8G2K64S4 family 178
8.5 STC8G2K64S2 family 180
8.6 STC8G1K08T touch key family 182
8.7 STC15H2K64S4 family 183
8.8 List of Special Function Registers 185
9 I/O Ports 194
9.1 Registers Related to I/O 194
9.1.1 Port Data Register (Px) 195
9.1.2 Ports Mode Registers (PxM0, PxM1) 195
9.1.3 Pull-up Resistor Control Registers (PxPU) 196
9.1.4 Schmitt Trigger Control Registers (PxNCS) 196
9.1.5 Level Togglling Speed Control Registers (PxSR) 196
9.1.6 Drive Current Control Registers (PxDR) 196
9.1.7 Digital Signal Input Enabling Control Registers (PxIE) 197
9.2 I/O Ports Configurations 198
9.3 I/O Ports Structure 199
9.3.1 Quasi-Bidirectional I/O (weak pull-up) 199
9.3.2 Push-Pull Output 199
9.3.3 High Impedance Input 200
9.3.4 Open-Drain Output 200
9.3.5 4.1K Pull-up Resistor 200
9.3.6 How to set the output speed of the IO port 201
9.3.7 How to set the drive current of the IO port 201
9.3.8 How to reduce the external radiation of the I/O port 201
9.4 Example Routines 202
9.4.1 Port Mode Setting 202
9.4.2 Reading and Writing Operation of Bidirection Port 203
9.5 A Typical Circuit Controlled by Triode 205
9.6 Typical Control Circuit of LED 205
9.7 Interconnection of 3V/5V Devices in Mixed Voltage Power Supply System 206
9.8 Make I/O Port Low When Power on Reset 206

9.9 Circuit Diagram of Driving 8 Digital LEDs using 74HC595 207
9.10 Digital LEDs Driven Directly by I/O Port Circuit 208
9.11 LCD Segment LCD Driven Directly by I/O Port Circuit 208
10 Instruction Set 229
11 Interrupt System 232
11.1 Interrupt sources of STC8G series 232
11.2 Structure of STC8G Interrupt 234
11.3 Interrupt List of STC8G Series of Microcontroller 235
11.4 Registers Related to Interrupt 238
11.4.1 Interrupt Enable Control Registers (Interrupt Enable bits) 239
11.4.2 Interrupt Request Registers (Interrupt flags) 244
11.4.3 Interrupt Priority Control Registers 247
11.5 Example Routines 250
11.5.1 INT0 Interrupt (Rising and Falling Edges) 250
11.5.2 INT0 Interrupt (Falling Edge) 252
11.5.3 INT1 Interrupt (Rising and Falling Edges) 253
11.5.4 INT1 Interrupt (Falling Edge) 255
11.5.5 INT2 Interrupt (Falling Edge) 257
11.5.6 INT3 Interrupt (Falling Edge) 259
11.5.7 INT4 Interrupt (Falling Edge) 261
11.5.8 Timer0 Interrupt 263
11.5.9 Timer1 Interrupt 264
11.5.10 Timer2 Interrupt 266
11.5.11 Timer3 Interrupt 268
11.5.12 Timer4 Interrupt 271
11.5.13 UART1 Interrupt 273
11.5.14 UART2 Interrupt 275
11.5.15 UART3 Interrupt 278
11.5.16 UART4 Interrupt 280
11.5.17 ADC Interrupt 283
11.5.18 LVD Interrupt 285
11.5.19 PCA Interrupt 287
11.5.20 SPI Interrupt 290
11.5.21 CMP Interrupt 292
11.5.22 PWM Interrupt 294
11.5.23 I2C Interrupt 296
12 Timers/Counters 300
12.1 Registers Related to Timers 300
12.2 Timer 0/1 302
12.2.1 Timer 0 and 1 Control Register 302
12.2.2 Timer 0/1 Mode Register 302
12.2.3 Timer0 mode 0 (16-bit auto-reloadable mode) 303
12.2.4 Timer0 mode 1 (16-bit non-autoreloadable mode) 304
12.2.5 Timer 0 mode 2 (8-bit auto-reloadable mode) 305

STCMCU

12.2.6 Timer 0 mode 3 (16-bit auto-realoadable mode with non-maskable interrupt, which can be used as real-time operating system metronome) 306
12.2.7 Timer 1 mode 0 (16-bit auto-reloadable mode)....306
12.2.8 Timer1 mode 1 (16-bit non-autoreloadable mode)....308
12.2.9 Timer 1 mode 2 (8-bit auto-reloadable mode)....308
12.2.10 Timer 0 Counting Register (TL0, TH0) 309
12.2.11 Timer 1 Counting Register (TL1, TH1) 309
12.2.12 Auxiliary Register 1 (AUXR) 309
12.2.13 External Interrupt and Clock Output Control Register (INTCLKO)....310
12.2.14 Timer 0 timing calculation formula....310
12.2.15 Timer 1 timing calculation formula....310
12.3 Timer 2 (24-bit timer, 8-bit prescaler + 16-bit timing)....311
12.3.1 Auxiliary Register 1 (AUXR) 311
12.3.2 External Interrupt and Clock Output Control Register (INTCLKO)....311
12.3.3 Timer 2 Counting Register (T2L, T2H) 311
12.3.4 Timer 2 8-bit Prescaler Register (TM2PS)....312
12.3.5 Timer 2 working mode 312
12.3.6 Timer 2 timing calculation formula....313
12.4 Timer 3/4 (24-bit timer, 8-bit prescaler + 16-bit timing)....313
12.4.1 Timer4 and Timer 3 Control Register (T4T3M)....313
12.4.2 Timer 3 Counting Register (T3L, T3H) 313
12.4.3 Timer 4 Counting Register (T4L, T4H) 314
12.4.4 Timer 3 8-bit Prescaler Register (TM3PS)....314
12.4.5 Timer 4 8-bit Prescaler Register (TM4PS)....314
12.4.6 Timer 3 working mode 314
12.4.7 Timer 4 working mode 315
12.4.8 Timer 3 timing calculation formula....316
12.4.9 Timer 4 timing calculation formula....316
12.5 Example Routines....317
12.5.1 Timer 0 (Mode 0 - 16-bit auto reload)....317
12.5.2 Timer 0 (Mode 1 - 16-bit non-auto reloadable)....318
12.5.3 Timer 0 (Mode 2 - 8-bit auto-reloadable)....320
12.5.4 Timer 0 (Mode 3 - 16-bit auto-reloadable with non-maskable interrupt)....322
12.5.5 Timer 0 (External count - T0 is extended for external falling edge interrupt)....324
12.5.6 Timer 0 (Pulse width measurement for high-level width of INT0)....326
12.5.7 Timer 0 (Divided clock output)....328
12.5.8 Timer 1 (Mode 0 - 16-bit auto-reloadable)....330
12.5.9 Timer 1 (Mode 1 - 16-bit non-auto reloadable)....332
12.5.10 Timer 1 (Mode 2 - 8-bit auto-reloadable)....334
12.5.11 Timer 1 (External count – T1 is extended for external falling edge interrupt)....336
12.5.12 Timer 1 (Pulse width measurement for high-level width of INT1)....338
12.5.13 Timer 1 (mode 0, Divided clock output) 340
12.5.14 Timer 1 (Mode 0) is used as baud rate generator of UART1....341
12.5.15 Timer 1 (Mode 2) is used as baud rate generator of UART1....345

12.5.16 Timer 2 (16-bit auto-reloadable) 349
12.5.17 Timer 2 (External count – T2 is extended for external falling edge interrupt) 351
12.5.18 Timer 2 (Divided clock output) 353
12.5.19 Timer 2 is used as baud rate generator of UART1 355
12.5.20 Timer 2 is used as baud rate generator of UART2 359
12.5.21 Timer 2 is used as baud rate generator of UART3 363
12.5.22 Timer 2 is used as baud rate generator of UART4 367
12.5.23 Timer 3 (16-bit auto-reloadable) 371
12.5.24 Timer 3 (External count – T3 is extended for external falling edge interrupt) 373
12.5.25 Timer 3 (Divided clock output) 376
12.5.26 Timer 3 is used as baud rate generator of UART3 378
12.5.27 Timer 4 (16-bit auto-reloadable) 382
12.5.28 Timer 4 (External count – T4 is extended for external falling edge interrupt) 384
12.5.29 Timer 4 (Divided clock output) 387
12.5.30 Timer 4 is used as baud rate generator of UART4 388
13 Serial Port (UART) Communication 393
13.1 Registers Related to UARTs 393
13.2 UART1 394
13.2.1 UART1 control register (SCON) 394
13.2.2 UART1 data register (SBUF) 395
13.2.3 Power control register (PCON) 395
13.2.4 Auxiliary register 1 (AUXR) 395
13.2.5 UART1 Mode 0 395
13.2.6 UART1 Mode 1 397
13.2.7 UART1 Mode 2 398
13.2.8 UART1 Mode 3 399
13.2.9 Automatic Address Recognition 400
13.3 UART2 401
13.3.1 UART2 control register (S2CON) 401
13.3.2 UART2 data register (S2BUF) 402
13.3.3 UART2 Mode 0 402
13.3.4 UART2 Mode 1 403
13.4 UART3 403
13.4.1 UART3 control register (S3CON) 403
13.4.2 UART3 data register (S3BUF) 404
13.4.3 UART3 Mode 0 404
13.4.4 UART3 Mode 1 405
13.5 UART4 406
13.5.1 UART4 control register (S4CON) 406
13.5.2 UART4 data register (S4BUF) 407
13.5.3 UART4 Mode 0 407
13.5.4 UART4 Mode 1 407
13.6 Precautions of UARTs 408
13.7 Example Routines 409

STCMCU

13.7.1 UART1 using T2 as baud rate generator 409
13.7.2 UART1 using T1 (Mode 0) as baud rate generator 413
13.7.3 UART1 using T1 (Mode 2) as baud rate generator 417
13.7.4 UART2 using T2 as baud rate generator 421
13.7.5 UART3 using T2 as baud rate generator 425
13.7.6 UART3 using T3 as baud rate generator 429
13.7.7 UART4 using T2 as baud rate generator 433
13.7.8 UART4 using T4 as baud rate generator 437
13.7.9 Serial port multi-machine communication 441
13.7.10 UART to LIN bus 441
14 Comparator, Power-down Detection, Internal Reference Voltage 451
14.1 Internal Structure of Comparator 451
14.2 Registers Related to Comparator 452
14.2.1 Comparator control register 1 (CMPCR1) 452
14.2.2 Comparator control register 2 (CMPCR2) 453
14.3 Example Routines 454
14.3.1 Using Comparator (Interrupt Mode) 454
14.3.2 Using Comparator (Polling Mode) 456
14.3.3 Multiplexing application of comparator (comparator + ADC input channel) 459
14.3.4 Comparator is Used for External Power-down Detection 461
14.3.5 Comparator is Used to Detect the Operation Voltage (Battery Voltage) 462
15 IAP/EEPROM 467
15.1 EEPROM operation time 467
15.2 Registers Related to EEPROM 467
15.2.1 EEPROM data register (IAP_DATA) 468
15.2.2 EEPROM address registers 468
15.2.3 EEPROM command register (IAP_CMD) 468
15.2.4 EEPROM trigger register (IAP_TRIG) 468
15.2.5 EEPROM control register (IAP_CONTR) 468
15.2.6 EEPROM wait time control register (IAP_TPS) 469
15.3 EEPROM Size and Address 470
15.4 Example Routines 473
15.4.1 EEPROM Basic Operation 473
15.4.2 Read EEPROM using MOVC 476
15.4.3 Send Out the Data in EEPROM Using UART 480
16 ADC, Internal 1.19V Reference Voltage 485
16.1 Registers Related to ADC 485
16.1.1 ADC control register (ADC_CONTR) 485
16.1.2 ADC configuration register (ADCCFG) 486
16.1.3 ADC conversion result register (ADC_RES, ADC_RESL) 487
16.1.4 ADC timing control register (ADCTIM) 487
16.2 Calculation formula related to ADC 488
16.2.1 Speed calculation formula of ADC 488
16.2.2 Conversion result calculation formula of ADC 489

STC MCU

16.2.3 Reverse calculation formula for ADC input voltage 489
16.2.4 Reverse operation voltage calculation formula 489
16.3 10-bit ADC Static Characteristics....490
16.4 12-bit ADC Static Characteristics....490
16.5 ADC application reference circuit diagram....491
16.5.1 Reference circuit diagram without independent VREF pin....491
16.5.2 Reference circuit diagram of general precision ADC with independent VREF pin 491
16.5.3 High-precision ADC reference circuit diagram with independent VREF pin 491
16.6 Example Routines....493
16.6.1 ADC Basic Operation (Polling Mode) 493
16.6.2 ADC Basic Operation (Interrupt Mode)....495
16.6.3 Format ADC Conversion Result....497
16.6.4 Detect External Voltage or Battery Voltage using ADC 15th Channel....500
16.6.5 Using ADC as Capacitive Sensing Touch Keys 502
16.6.6 Key-scan Application Circuit Diagram using ADC 515
16.6.7 Reference circuit diagram for detecting negative voltage 516
16.6.8 The Application of Common Adding Circuits in ADC 517
17 Application of PCA/CCP/PWM....518
17.1 Registers Related to PCA....518
17.1.1 PCA control register (CCON) 519
17.1.2 PCA mode register (CMOD)....519
17.1.3 PCA counter registers (CL, CH)....519
17.1.4 PCA mode control registers (CCAPMn) 519
17.1.5 PCA capture value/compare value registers (CCAPnL, CCAPnH) 520
17.1.6 PCA PWM mode control registers (PCA_PWMn) 520
17.2 PCA Operation Mode 520
17.2.1 Capture Mode....521
17.2.2 Software Timer Mode....521
17.2.3 High Speed Pulse Output Mode 522
17.2.4 Pulse Width Modulation Mode (PWM mode)....522
17.3 Reference Circuit for Implementing 8 ~ 16-bit DAC using CCP / PCA module 527
17.4 Example Routines....528
17.4.1 PCA Output PWM (6/7/8/10 bit)....528
17.4.2 PCA Capture and Measure Pulse Width....531
17.4.3 PCA Implements 16-bit Software Timing 535
17.4.4 PCA realizes 16-bit software timing (ECI external clock mode) 538
17.4.5 PCA Output High-speed Pulse 541
17.4.6 PCA Extends External Interrupt....544
18 Enhanced PWM with 15-bit Accuracy 548
18.1 Registers Related to PWM 549
18.1.1 Enhanced PWM global configuration register (PWMSET) 553
18.1.2 Enhanced PWM configuration registers (PWMCFGn)....554
18.1.3 PWM Interrupt Flag Registers (PWMnIF)....555
18.1.4 PWM Fault Detection Control Registers (PWMnFDCR) 555

18.1.5 PWM Counter Registers (PWMnCH, PWMnCL) 556
18.1.6 PWM Clock Selection Registers (PWMnCKS) 556
18.1.7 PWM Trigger ADC counter Registers (PWMnTADC) 557
18.1.8 PWM Level output setting count value Registers (PWMnT1， PWMnT2) 557
18.1.9 PWM Channel Control Registers (PWMnCR) 560
18.1.10 PWM Channel Level Holding Control Registers (PWMnHLD) 562
18.2 Example Routines 563
18.2.1 Output waveforms with arbitrary period and arbitrary duty 563
18.2.2 Two-channel PWMs realize complementary symmetrical waveform with dead-time control 566
18.2.3 PWM Implements Gradient Light (Breathing Light) 569
18.2.4 Use PWM to trigger ADC conversion 574
18.2.5 Generate 3 complementary PWM waveforms with dead-time with a phase difference of 120 degrees 578
18.2.6 Method of outputting a PWM waveform with a duty cycle of 100% (fixed output high) and 0% (fixed output low) (take PWM00 as an example) 581
18.2.7 Enhanced PWM-adjustable frequency-pulse counting 581
19 Sysnchronous Serial Peripheral Interface (SPI) 585
19.1 Registers Related to SPI 585
19.1.1 SPI Status register (SPSTAT) 585
19.1.2 SPI Control register (SPCTL) 585
19.1.3 SPI Data register (SPDAT) 586
19.2 SPI Communication Modes 587
19.2.1 Single Master and Single Slave Mode 587
19.2.2 Dual Devices Configuration Mode 587
19.2.3 Single Master and Multiple Slaves Mode 588
19.3 Configure SPI 589
19.4 Data Format 591
19.5 Example Routines 593
19.5.1 Master Routine of Single Master Single Slave Mode (Interrupt Mode) 593
19.5.2 Slave Routine of Single Master Single Slave Mode (Interrupt Mode) 595
19.5.3 Master Routine of Single Master Single Slave Mode (Polling Mode) 597
19.5.4 Slave Routine of Single Master Single Slave Mode (Polling Mode) 600
19.5.5 Routine of Mutual Master-Slave Mode (Interrupt Mode) 602
19.5.6 Routine of Mutual Master-Slave Mode (Polling Mode) 605
20 I^2C Bus 608
20.1 Registers Related to I^2C 608
20.2 I^2C Master Mode 609
20.2.1 I^2C Configuration Register (I2CCFG) 609
20.2.2 I^2C Master Control Register (I2CMSCR) 610
20.2.3 I^2C Master Auxiliary Control Register (I2CMSAUX) 611
20.2.4 I^2C Master Status Register (I2CMSST) 612
20.3 I^2C Slave Mode 613
20.3.1 I^2C Slave Control Register (I2CSLCR) 613

20.3.2 I²C Slave Status Register (I2CSLST) 613
20.3.3 I²C Slave Address Register (I2CSLADR) 614
20.3.4 I²C data Registers (I2CTXD, I2CRXD) 615
20.4 Example Routines 616
20.4.1 I²C is Used to Access AT24C256 in Master Mode (Interrupt Mode) 616
20.4.2 I²C is Used to Access AT24C256 in Master Mode AT24C256 (Polling Mode) 622
20.4.3 I²C is Used to Access PCF8563 in Master Mode 628
20.4.4 I²C Slave Mode (Polling Mode) 633
20.4.5 I²C Slave Mode (Polling Mode) 638
20.4.6 Master Codes for testing I²C Slave Mode 642
21 Touch Key Controller 649
21.1 Internal Structure Diagram of Touch Key Controller 651
21.2 Wake-up from Low Power Mode using Touch Key 651
21.3 Operation Steps When Touch Key Function is Used only 651
21.4 Operation Steps for Wake-up from Low Power Mode using Touch Key 652
21.5 Registers Related to Touch Key Controller 653
21.5.1 Touch Key Enable Registers (TSCHENn) 653
21.5.2 Touch Key Configuration Registers (TSCFGn) 654
21.5.3 Touch Key power-down mode wake-up time control register (TSWUTC) 654
21.5.4 Touch Key Control Register (TSCTRL) 655
21.5.5 Touch Key Status Register 1 (TSSTA1) 656
21.5.6 Touch Key Status Register 2 (TSSTA2) 656
21.5.7 Touch Key Time Control Register (TSRT) 657
21.5.8 Touch Key Data Registers (TSDAT) 657
21.5.9 Touch Key Threshold Registers (TSTH) 657
21.6 Basic Reference Circuit and Precautions 658
22 LED Driver 659
22.1 Internal Structure Diagram of LED Driver 660
22.2 Registers Related to LED Driver 660
22.2.1 COM Enable Register (COMEN) 661
22.2.2 SEG Enable Register (SEGEN) 661
22.2.3 LED Control Register (LEDCTRL) 661
22.2.4 LED Clock Divide Register (LEDCKS) 661
22.2.5 LED data registers of common anode mode (COMn_DA) 662
22.2.6 LED data registers of common cathod mode (COMn_DC) 662
22.3 LED Common Cathod Mode (LEDMODE = 00) 663
22.4 LED Common Anode Mode (LEDMODE = 01) 664
22.5 LED Common Cathode/ Common Anode Mode (LEDMODE = 10) 665
22.6 Touch Keys and LED Driver Share I/O 666
22.7 Reference Circuit of Common Cathode Mode 668
22.8 Reference Circuit of Common Anode Mode 668
22.9 Reference Circuit of Common Cathode/Common Anode Mode 669
22.10 Example Routines 669
22.10.1 Common cathode/common anode mode drives 16 7-segment digital LEDs 670

23 Enhanced Dual Data Pointer 673
23.1 Related special function registers 673
23.1.1 1st 16-bit Data Pointer Registers (DPTR0) 673
23.1.2 2nd 16-bit Data Pointer Registers (DPTR1) 673
23.1.3 DPTR control register (DPS) 673
23.1.4 DPTR control register (TA) 674
23.2 Example Routines 676
23.2.1 Example Routine 1 676
23.2.2 Example Routine 2 677
24 MDU16 Hardware 16-bit Multiplier and Divider 679
24.1 Registers Related to MDU16 679
24.1.1 Operand 1 Data Registers (MD0~MD3) 679
24.1.2 Operand 2 Data Registers (MD4~MD5) 679
24.1.3 MDU Mode Control Register (ARCON) 680
24.1.4 MDU Operation Control Register (OPCON) 680
24.2 Miscellaneous talks about the application of MDU16 by netizens (provide ideas for reference only) 681
24.3 Example Routines 682
Appendix A STC Emulator User Guide 685
Appendix B How to Make the Traditional 8051 MCU EVB Emulatable 692
Appendix C STC-USB Driver Installation Instructions 694
Appendix D Download Step Demo using USB 757
Appendix E Circuit of RS485 Automatic Control or I/O Control 761
Appendix F STC tool instruction manual 763
F.1 Overview 763
F.2 In-system programmable (ISP) process description 763
F.3 USB type online/offline download tool U8W/U8W-Mini 764
F.3.1 Install U8W/U8W-Mini driver 766
F.3.2 U8W function introduction 769
F.3.3 Instructions for online download of U8W 770
F.3.4 Instructions for offline download of U8W 771
F.3.5 U8W-Mini's function introduction 774
F.3.6 U8W-Mini's online download instructions 775
F.3.7 Instructions for offline download of U8W-Mini 776
F.3.8 Make/Update U8W/U8W-Mini 778
F.3.10 Set U8W/U8W-Mini to pass-through mode (can be used for simulation) 779
F.3.11 Reference circuit of U8W/U8W-Mini 779
F.4.1 Appearance of STC Universal USB to Serial Tool 781
F.4.2 Layout diagram of STC general USB to serial tool 781
F.4.3 STC Universal USB to Serial Tool Driver Installation 782
F.4.4 Use the STC universal USB to serial tool to download the program to the MCU 783
F.4.5 Use STC universal USB to serial port tool to simulate user code 783
F.5.2 STC Universal USB to Serial Tool Application Reference Circuit Diagram 786
Appendix G Partial Circuit of RS485 in U8W Download Tool 786

Appendix H ISP Download Starts Automatically After Receiving User Command While Running User Program (no Power-down) 788
Appendix I Use STC's IAP series MCU to develop your own ISP program 790
Appendix J The method of resetting the user program to the system area for ISP download (without power down) 800
Appendix K Example Routine of ISP download for STC8G series MCUs using third-party MCU..805
Appendix L Use a third-party application program to call the STC release project program to download to MCU using the ISP.....813
Appendix M Method for Creating Multi-file Projects in Keil 816
Appendix N Handling of Compilation Error in Keil with Interrupt Numbers Greater Than 31.....821
Appendix O Electrical Characteristics.....831
Appendix P Application note.....835
Appendix Q PCB design guidance for touch buttons 837
Appendix R QFN/DFN packaged components welding method.....838
Appendix S Precautions for replacing STC15 series with STC8G series MCU.....841
Appendix T Precautions for replacing STC8A/8F series with STC8G series MCU 842
Appendix U Precautions for replacing STC15F/L/W series with STC15H series MCU 843
Appendix V Update Records.....845
Appendix W STC8 series naming tidbits.....853
Appendix X Product Authorization Letter.....854

STC MCU

1 Overview

STC8G series of microcontrollers are microcontrollers that do not need an external crystal oscillator and external reset circuit. They are 8051 core microcontrollers with the goal of strong anti-interference, ultra low price, high speed and low power consumption. Under the same operating frequency, STC8G series of microcontrollers are about 12 times faster (11.2 ~ 13.2 times) than traditional 8051. To execute all 111 instructions in sequence, the STC8G series microcontroller only needs 147 clocks, while the traditional 8051 requires 1944 clocks. STC8G series of microcontrollers are single clock/machine cycle (1T) microcontrollers produced by STC. It is a new generation 8051 microcontrollers with wide voltage, high speed, high reliability, low power consumption, strong antistatic, strong anti-interference and super encrypted. The instruction codes are fully compatible with traditional 8051.

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

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

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

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

Rich digital peripherals and analog peripherals are provided in MCU, including UARTs, timers, PCAs, PWMs and I2C, SPI, ultra-high speed ADC and comparator, which can meet the needs of users when designing a product.

The enhanced dual data pointers are integrated in the STC8G series of microcontrollers. Using program control, the function of automatic increasing or decreasing of data pointer and automatic switching of two sets of data pointers can be realized.

Products	I/O	UART	Timers	ADC	Enhanced PWM	PCA	CMP	SPI	I2C	MDU16	LED	Touch Key
STC8G1K08 family	18	2	3	15_{CH}*10_{B}		●	●	●	●			
STC8G1K08-8Pin family	6	1	2					●	●	●		

STC8G1K08A family	**6**	**1**	**2**	**6_{CH}*10_{B}**		●		●	●	●		
STC8G2K64S4 family	**45**	**4**	**5**	**15_{CH}*10_{B}**	●	●	●	●	●	●		
STC8G2K64S2 family	**45**	**2**	**5**	**15_{CH}*10_{B}**	●	●	●	●	●	●		
STC8G1K08T family	**16**	**1**	**3**	**15_{CH}*10_{B}**		●	●	●	●		●	●
STC15H2K64S4 family	**42**	**4**	**5**	**15CH*10B**	●	●	●	●	●	●		

STC MCU

2 Features, Price and Pins

2.1 STC8G1K08-38I-TSSOP20/QFN20/SOP16 family

2.1.1 Features and Price

➢ **Selection and price (No external crystal and external reset required with 15 channels 10-bit ADC)**

MCU model	Operating voltage (V)	Flash Code Memory (100 thousand times) (Byte)	idata, internal DATA RAM (Byte)	xdata, internal extended SRAM (Byte)	Enhanced Dual DPTR increasing or decreasing	EEPROM 100 thousand times) (Byte)	Maximum I/O Lines	UARTs which may wake-up CPU	SPI	I²C	Timers/Counters (T0-T2 Pin can wake-up CPU)	16-bit advanced PWM timer with complementary symmetrical dead-time	15-bit enhanced PWM (with dead-time control)	PCA/CCP/PWM (can be used as external interrupt and can wake-up CPU)	Power-down Wake-up timer	15-bit high speed ADC (3 PCAs can be used as 3 DACs)	Comparator (May be used as ADC to detect external power-down)	Internal LVD interrupt (can wake-up CPU)	Watch-dog Timer	Internal high reliable reset circuit with 4 level optional reset threshold voltage	Internal high presision Clock (adjustbal under 36MHz)	Clock output and Reset	Program encrypted transmission (Anti-blocking)	Password can be set for next update	Support RS485 download	Support software USBdownload directly	Online debugging	Package						Main product supply information
																												TSSOP20	QFN20 (3mm*3mm)	SOP16	SOP20(Not recommended for use)	DIP20(Not recommended for use)	DIP16(Not recommended for use)	
STC8G1K04	1.9-5.5	4K	256B	1K	2	8K	18	2	Y	Y	3	-	-	3	Y	10bit	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y				√			Available
STC8G1K08	1.9-5.5	8K	256B	1K	2	4K	18	2	Y	Y	3	-	-	3	Y	10bit	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	√	√	√	√	√	√	
~~STC8G1K12~~	~~1.9-5.5~~	~~12K~~	~~256B~~	~~1K~~	~~2~~	~~IAP~~	~~18~~	~~2~~	~~Y~~	~~Y~~	~~3~~	~~-~~	~~-~~	~~3~~	~~Y~~	~~10bit~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	√	√	√				
STC8G1K17	1.9-5.5	17K	256B	1K	2	IAP	18	2	Y	Y	3	-	-	3	Y	10bit	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	-	√	√	√		√	√	

Note: The above unit prices are for orders of quantity of 10K and above. If the quantity is small, an additional RMB 0.1 per piece will be required. When the total amount of the order reaches or exceeds 3,000 yuan, it can be shipped free of charge, otherwise the customer will have to bear the freight. Retail sale starts at 10 pieces.

➢ **Core**

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

➢ **Operating voltage**

- ✓ 1.9V～5.5V
- ✓ Built-in LDO

➢ **Operating temperature**

- ✓ -40℃~85℃(For applications beyond the temperature range, please refer to the description of the electrical characteristics chapter)

➢ **Flash memory**

- ✓ Up to 17Kbytes of Flash memory to be used to store user code
- ✓ Configurable size EEPROM, 512bytes single page erased, can be repeatedly erased more than 100 thousand times.
- ✓ In-System-Programming, ISP in short, can be used to update the application code, no need for special programmer.
- ✓ Online debugging with single chip is supported, and no special emulator is needed. The number of breakpoints is unlimited theoratically.

➢ **SRAM**

- ✓ 128 bytes internal direct access RAM (DATA)
- ✓ 128 bytes internal indirect access RAM (IDATA)
- ✓ 1024 bytes internal extended RAM (internal XDATA)

➢ **Clock**

- ✓ Internal high precise R/C clock (IRC, range from 4MHz to 36MHz), adjustable while ISP and can be divided to lower frequency by user software, 100KHz for instance.
  - ⌖ Error:±0.3% (at the temperature 25℃)
  - ⌖ -1.38%～+1.42% temperature drift (at the temperature range of -40℃ to +85℃)
  - ⌖ -0.88%～+1.05% temperature drift (at the temperature range of -20℃ to 65℃)
- ✓ Internal 32KHz low speed IRC with large error
- ✓ External 4MHz~33MHz oscillator or external clock

  The three clock sources above can be selected freely by used code.

➢ **Reset**

- ✓ Hardware reset
  - ⌖ Power-on reset. Measured voltage value is 1.69V~1.82V. (**Effective when the chip does not enable the low voltage reset function**)

    The power-on reset voltage is a voltage range consisting of an upper limit voltage and a lower limit voltage. When the operating voltage drops from 5V / 3.3V to the lower limit threshold voltage of the power-on reset, the chip is in reset state. When the voltage rises from 0V to the upper threshold voltage of power-on reset, the chip is released from the reset state.
  - ⌖ Reset by reset pin. The default function of P5.4 is I/O port. P5.4 pin can be set as the reset pin while ISP download. (**Note: When the P5.4 pin is set as the reset pin, the reset level is low**.)
  - ⌖ Watch dog timer reset
  - ⌖ Low voltage detection reset. 4 low voltage detection levels are provided, 2.2V (Measured as 1.90V~2.04V), 2.4V (Measured as 2.30V~2.50V), V2.7 (Measured as 2.61V~2.82V), V3.0 (Measured as 2.90V~3.13V). Each level of low-voltage detection voltage is a voltage range consisting of an upper limit voltage and a lower limit voltage. When the operating voltage drops from 5V / 3.3V to the lower limit threshold voltage of low-voltage detection, the low-voltage detection takes effect. When the voltage rises from 0V to the upper threshold voltage, the low voltage detection becomes effective.
- ✓ Software reset
  - ⌖ Writing the reset trigger register using software

➢ **Interrupts**

- ✓ 16 interrupt sources: INT0(Supports rising edge and falling edge interrupt), INT1(Supports rising edge and falling edge interrupt), INT2(Supports falling edge interrupt only), INT3(Supports falling edge interrupt only), INT4(Supports falling edge interrupt only), timer0, timer1, timer2, UART1, UART2, ADC, LVD, SPI, I^2C, comparator, PCA/CCP/PWM
- ✓ 4 interrupt priority levels
- ✓ Interrupts that can awaken the CPU in clock stop mode: INT0 (P3.2), INT1 (P3.3), INT2 (P3.6), INT3 (P3.7), INT4 (P3.0), T0 (P3.4)), T1(P3.5), T2(P1.2), RXD(P3.0/P3.6/P1.6), RXD2(P1.4), CCP0(P1.1/P3.5), CCP1 (P1.0/P3.6), CCP2 (P3.7), I2C_SDA (P1.4/P3.3) and comparator interrupt, low-voltage detection interrupt, power-down wake-up timer.

➢ **Digital peripherals**

- ✓ 3 16-bit timers: timer0, timer1, timer2, where the mode 3 of timer 0 has the Non Maskable Interrupt (NMI in short) function. Mode 0 of timer 0 and timer 1 is 16-bit Auto-reload mode.
- ✓ 2 high speed UARTs: UART1, UART2, whose baudrate clock may be as fast as FOSC/4
- ✓ 3 groups of PCAs: CCP0, CCP1, CCP2, which can be used as capture, high speed output and 6-bits, 7-bits, 8-bits or 10-bits PWM.
- ✓ SPI: Master mode, slave mode or master/slave automatic switch mode are supported.
- ✓ I^2C: Master mode or slave mode are supported.

➢ **Analog peripherals**

- ✓ 15 channels (channel 0 to channel 14) ultra-high speed ADC which supports 10-bit precision analog-to-digital conversion, the speed can be as fast as 500K(500,000 conversions per second).
- ✓ ADC channel 15 is used to test the internal reference voltage. (The default internal reference voltage is 1.19V when the chip is shipped)
- ✓ Comparator. A set of comparators (the positive terminal of the comparator can select the CMP+ and all ADC input ports, so the comparator can be used as a multi-channel comparator for time division multiplexing).
- ✓ DAC. 3 groups of PCAs can be used as DAC.

➢ **GPIO**

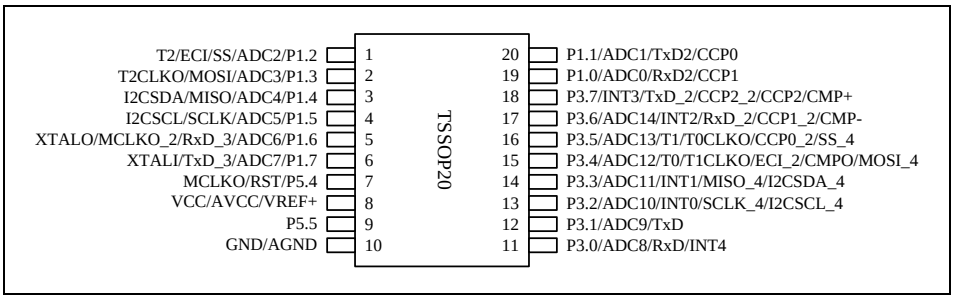
- ✓ Up to 18 GPIOs: P1.0~P1.7, P3.0~P3.7, P5.4~P5.5
- ✓ 4 modes for all GPIOs: quasi_bidirectional mode, push-pull outputmode, open drain mode, high-impedance input mode
- ✓ **Except for P3.0 and P3.1, all other I/O ports are in high-impedance state after power-on. User must set the I/O ports mode before using them. In addition, each I/O can independently enable the internal 4K pull-up resistor.**

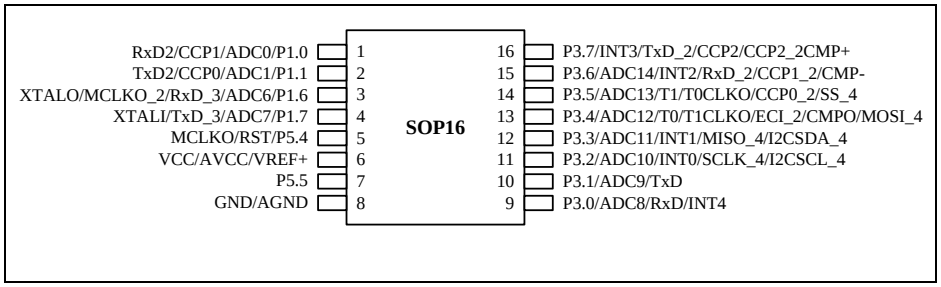
➢ **Package**

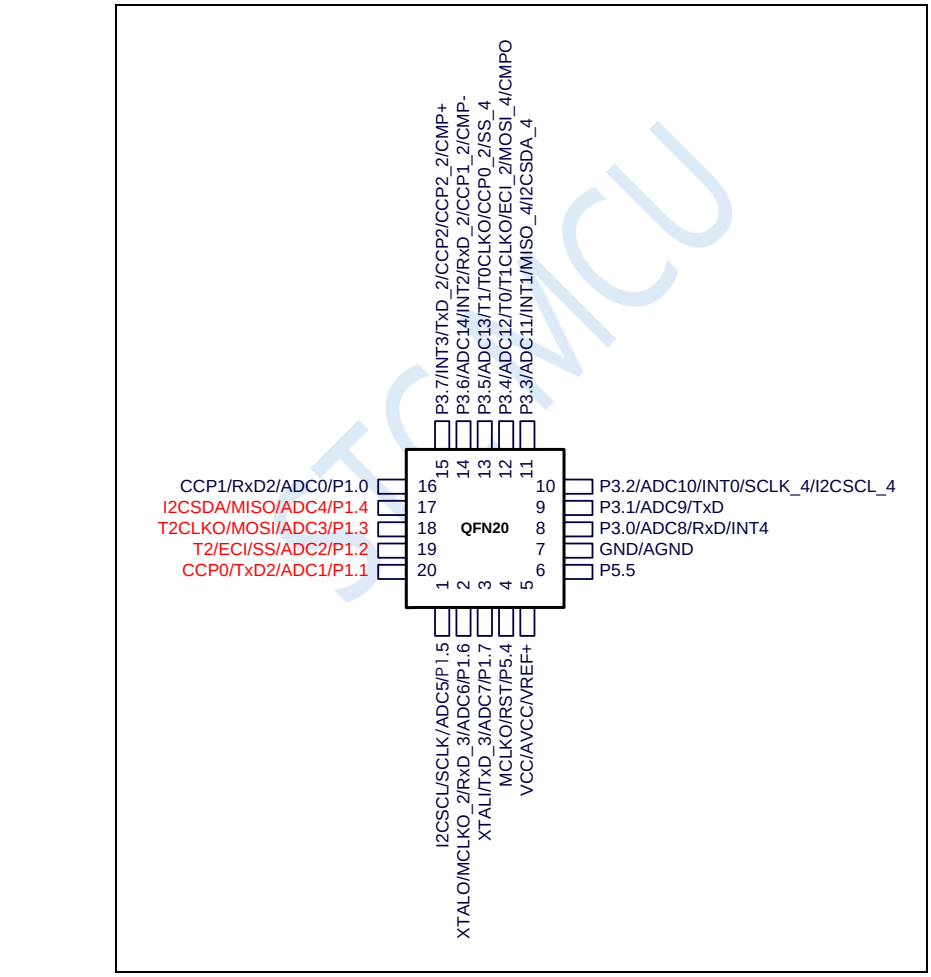
- ✓ TSSOP20, QFN20 (3mm*3mm), SOP20, DIP20, SOP16, DIP16

2.1.2 Pinouts

STC MCU









universal USB to UART tool

ISP download steps:

1. **Connect the universal USB to UART tool to the target chip according to the connection method shown in the figure above.**
2. **Press the power button to confirm that the target chip is in a power-off state (the power-on indicator is off).**

   Note: When the tool is powered on for the first time, there is no external power supply, so if it is the first time to use this tool, you can skip this step.
3. **Click the "Download/Program" button in the STC-ISP download software.**
4. **Press the power button again to power on the target chip (the power-on indicator is on).**
5. **Start ISP download.**

   Note: It has been found that when using the USB cable for ISP download, if the USB cable is too thin and the voltage drop on the USB cable is too large, this will result in insufficient power supply during the ISP download. Therefore, please be sure to use the booster USB cable for ISP download.

Note:

1. **If USB download is not required, P3.0/P3.1/P3.2 can not be low at the same time when the chip is reset.**
2. **Except for P3.0 and P3.1, all other I/O ports are in high-impedance input state after power-on. User must set the I/O port mode firstly when using I/O.**
3. **All I/O ports can be set to quasi-bidirectional port mode, strong push-pull output mode, open-drain output mode or high-impedance input mode. In addition, each I/O can independently enable the internal 4K pull-up resistor.**
4. **When P5.4 is enabled as the reset pin, the reset level is low.**
5. **For STC8G1K08-20Pin/16Pin series B version chip, when P5.4 is used as I/O port, the current should not exceed 50mA, and there should be no strong impact.**
6. **The USB download supported by the STC8G1K08-20Pin/16Pin series is software simulated USB communication by the I/O port. It is inevitably affected by various software and hardware factors, especially the different software and hardware versions on the computer side, resulting in a certain proportion of chips cannot be downloaded via USB (about 0.2% of chips cannot be downloaded via USB). It is recommended to use ordinary serial download or USB to serial module to download for batch production.**

2.1.3 Pin descriptions

Pin number				name	type	description
TSSOP20 DIP20	QFN20	SOP16 DIP16				
1	19			P1.2	I/O	Standard IO port
				ADC2	I	ADC analog input 2
				SS	I/O	Slave selection of SPI
				T2	I	Timer2 external input
				ECI	I	External pulse input of PCA
2	18			P1.3	I/O	Standard IO port
				ADC3	I	ADC analog input 3
				T2CLKO	O	Clock out of timer 2
				MOSI	I/O	Master Output/Slave Input of SPI
3	17			P1.4	I/O	Standard IO port
				ADC4	I	ADC analog input 4
				MISO	I/O	Master Iutput/Slave Onput of SPI
				SDA	I/O	Serial data line of I2C
4	1			P1.5	I/O	Standard IO port
				ADC5	I	ADC analog input 5
				SCLK	I/O	Serial Clock of SPI
				SCL	I/O	Serial Clock line of I2C
5	2	3		P1.6	I/O	Standard IO port
				ADC6	I	ADC analog input 6
				RxD_3	I	Serial input of UART1
				MCLKO_2	O	Master clock output
				XTALO	O	Connect to external oscillator
6	3	4		P1.7	I/O	Standard IO port
				ADC7	I	ADC analog input 7
				TxD_3	O	Serial output of UART 1
				XTALI	I	Connect to external oscillator
7	4	5		P5.4	I/O	Standard IO port
				RST	I	Reset pin
				MCLKO	O	Master clock output
8	5	6		VCC	VCC	Power Supply
				AVCC	VCC	Power Supply for ADC
				VREF+	I	Reference voltage pin of ADC
9	6	7		P5.5	I/O	Standard IO port
10	7	8		GND	GND	Ground
				AGND	GND	ADC Ground
11	8	9		P3.0	I/O	Standard IO port
				RxD	I	Serial input of UART1
				ADC8	I	ADC analog input 8
				INT4	I	External interrupt 4
12	9	10		P3.1	I/O	Standard IO port
				TxD	O	Serial output of UART 1
				ADC9	I	ADC analog input 9
13	10	11		P3.2	I/O	Standard IO port
				INT0	I	External interrupt 0
				ADC10	I	ADC analog input 10
				SCL_4	I/O	Clock of I2C
				SCLK_4	I/O	Clock of SPI

Pin number				name	type	description
TSSOP20 DIP20	QFN20	SOP16 DIP16				
14	11	12		P3.3	I/O	Standard IO port
				INT1	I	External interrupt 1
				ADC11	I	ADC analog input 11
				SDA_4	I/O	Data of I2C
				MISO_4	I/O	Master Iutput/Slave Onput of SPI
15	12	13		P3.4	I/O	Standard IO port
				T0	I	Timer0 external input
				T1CLKO	O	Clock out of timer 1
				ADC12	I	ADC analog input 12
				ECI_2	I	External pulse input of PCA
				CMPO	O	Comparator output
				MOSI_4	I/O	Master Output/Slave Input of SPI
16	13	14		P3.5	I/O	Standard IO port
				T1	I	Timer1 external input
				T0CLKO	O	Clock out of timer 0
				ADC13	I	ADC analog input 13
				CCP0_2	I/O	Capture of external signal/High-speed Pulse output of PCA
				SS_4	I	Slave selection of SPI (it is output with regard to master)
17	14	15		P3.6	I/O	Standard IO port
				INT2	I	External interrupt 2
				RxD_2	I	Serial input of UART1
				ADC14	I	ADC analog input 14
				CCP1_2	I/O	Capture of external signal/High-speed Pulse output of PCA
				CMP-	I	Comparator negative input
18	15	16		P3.7	I/O	Standard IO port
				INT3	I	External interrupt 3
				TxD_2	O	Serial output of UART 1
				CCP2	I/O	Capture of external signal/High-speed Pulse output of PCA
				CCP2_2	I/O	Capture of external signal/High-speed Pulse output of PCA
				CMP+	I	Comparator positive input
19	16	1		P1.0	I/O	Standard IO port
				RxD2	I	Serial input of UART2
				ADC0	I	ADC analog input 0
				CCP1	I/O	Capture of external signal/High-speed Pulse output of PCA
20	20	2		P1.1	I/O	Standard IO port
				TxD2	O	Serial output of UART 2
				ADC1	I	ADC analog input 1
				CCP0	I/O	Capture of external signal/High-speed Pulse output of PCA

2.2 STC8G1K08-36I-SOP8/DFN8 family

2.2.1 Features and Price

- **Selection and price (No external crystal and external reset circuit required)**

MCU model	Operating voltage（V）	Flash Code Memory (100 thousand times) (Byte)	idata, Internal DATA RAM (Byte)	xdata, Internal extended SRAM (Byte)	Enhanced Dual DPTR (increasing or decreasing)	EEPROM 100 thousand times) (Byte)	Maximum I/O Lines	UARTs which may wake-up CPU	MDU16 Hardware 16-bit Multiplier and Divider	SPI	I²C	Timers/Counters (T0-T2 Pin Can wake-up CPU)	16-bit advanced PWM timer Complementary symmetrical dead-time	15-bit enhanced PWM (with dead-time control)	PCA/CCP/PWM (can be used as external interrupt and can wake-up CPU)	Power-down Wake-up timer	15-channels high speed ADC (8 PWMs can be used as 8 DACs)	Comparator (May be used as ADC to detect external power-down)	Internal LVD interrupt (can wake-up CPU)	Watch-dog Timer	Internal high reliable reset circuit with 4-level optional reset threshold voltage	Internal high pression Clock (adjustbal under 36MHz)	Clock output and Reset	Program encrypted transmission (Anti-blocking)	Password can be set for next update	Support RS485 download	Support software USBdownload directly	Online debugging	Package		Main product supply information
																													SOP8	DFN8<3mm*3mm>	
STC8G1K08	1.9-5.5	8K	256	1K	2	4K	6	1	Y	Y	Y	2	-	-	-	Y	-		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	√	√	Available
~~STC8G1K12~~	~~1.9-5.5~~	~~12K~~	~~256~~	~~1K~~	~~2~~	~~IAP~~	~~6~~	~~1~~	~~Y~~	~~Y~~	~~Y~~	~~2~~	~~-~~	~~-~~	~~-~~	~~Y~~	~~-~~		~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~			
STC8G1K17	1.9-5.5	17K	256	1K	2	IAP	6	1	Y	Y	Y	2	-	-	-	Y	-		Y	Y	Y	Y	Y	Y	Y	Y	-	-	√	√	

Note: The above unit prices are for orders of quantity of 10K and above. If the quantity is small, an additional RMB 0.1 per piece will be required. When the total amount of the order reaches or exceeds 3,000 yuan, it can be shipped free of charge, otherwise the customer will have to bear the freight. Retail sale starts at 10 pieces.

- **Core**
  - ✓ Ultra-high speed 8051 core with single clock per machine cycle, which is called 1T and is about 12 times faster than traditional 8051
  - ✓ Fully compatible instruction set with traditional 8051
  - ✓ 11 interrupt sources and 4 interrupt priority levels
  - ✓ Online debugging is supported
- **Operating voltage**
  - ✓ 1.9V～5.5V
  - ✓ Built-in LDO
- **Operating temperature**
  - ✓ -40℃~85℃
- **Flash memory**
  - ✓ Up to 17K bytes of Flash memory to be used to store user code

- ✓ Configurable size EEPROM, 512 bytes single page erased, can be repeatedly erased more than 100 thousand times.
- ✓ In-System-Programming, ISP in short, can be used to update the application code, no need for special programmer.
- ✓ Online debugging with single chip is supported, and no special emulator is needed. The number of breakpoints is unlimited theoratically.

➢ **SRAM**

- ✓ 128 bytes internal direct access RAM (DATA)
- ✓ 128 bytes internal indirect access RAM (IDATA)
- ✓ 1024 bytes internal extended RAM (internal XDATA)

➢ **Clock**

- ✓ Internal high precise R/C clock (IRC, range from 4MHz to 36MHz), adjustable while ISP and can be divided to lower frequency by user software, 100KHz for instance.
  - ⌖ Error:±0.3% (at the temperature 25℃)
  - ⌖ -1.38%～+1.42% temperature drift (at the temperature range of -40℃ to +85℃)
  - ⌖ -0.88%～+1.05% temperature drift (at the temperature range of -20℃ to 65℃)
- ✓ Internal 32KHz low speed IRC with large error

➢ **Reset**

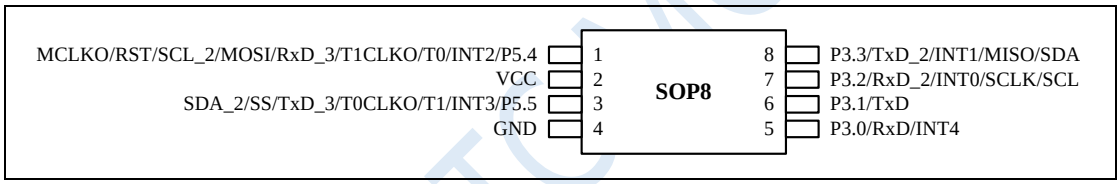
- ✓ Hardware reset
  - ⌖ Power-on reset. Measured voltage value is 1.69V~1.82V. (**Effective when the chip does not enable the low voltage reset function**)
    The power-on reset voltage is a voltage range consisting of an upper limit voltage and a lower limit voltage. When the operating voltage drops from 5V / 3.3V to the lower limit threshold voltage of the power-on reset, the chip is in a reset state; when the voltage rises from 0V to the upper threshold voltage of power-on reset, the chip is released from the reset state.
  - ⌖ Reset by reset pin. The default function of P5.4 is the I/O port. The P5.4 pin can be set as the reset pin while ISP download. (**Note: When the P5.4 pin is set as the reset pin, the reset level is low**.)
  - ⌖ Watch dog timer reset
  - ⌖ Low voltage detection reset. 4 low voltage detection levels are provided, 2.2V (Measured as 1.90V~2.04V), 2.4V (Measured as 2.30V~2.50V), V2.7 (Measured as 2.61V~2.82V), V3.0 (Measured as 2.90V~3.13V). Each level of low-voltage detection voltage is a voltage range consisting of an upper limit voltage and a lower limit voltage. When the operating voltage drops from 5V / 3.3V to the lower limit threshold voltage of low-voltage detection, the low-voltage detection takes effect. When the voltage rises from 0V to the upper threshold voltage, the low voltage detection becomes effective.
- ✓ Software reset
  - ⌖ Writing the reset trigger register using software

➢ **Interrupts**

- ✓ 11 interrupt sources: INT0(Supports rising edge and falling edge interrupt), INT1(Supports rising edge and falling edge interrupt), INT2(Supports falling edge interrupt only), INT3(Supports falling edge interrupt only), INT4(Supports falling edge interrupt only), timer0, timer1, UART1, LVD, SPI, I^2C
- ✓ 4 interrupt priority levels
- ✓ Interrupts that can awaken the CPU in clock stop mode: INT0 (P3.2), INT1 (P3.3), INT2 (P3.6), INT3 (P3.7), INT4 (P3.0), T0 (P3.4), T1(P3.5), RXD(P3.0/P3.2/P5.4), I2C_SDA (P3.3/P5.5) and low-voltage detection interrupt, power-down wake-up timer.

- **Digital peripherals**
  - ✓ 2 16-bit timers: timer0, timer1, where the mode 3 of timer0 has the Non Maskable Interrupt (NMI in short) function. Mode 0 of timer0 and timer1 is 16-bit Auto-reload mode.
  - ✓ 1 high speed UART: UART1, whose baudrate clock source may be fast as FOSC/4
  - ✓ SPI: Master mode, slave mode or master/slave automatic switch mode are supported.
  - ✓ I^2C: Master mode or slave mode are supported.
  - ✓ **MDU16: Hardware 16-bit Multiplier and Divider which supports 32-bit divided by 16-bit, 16-bit divided by 16-bit, 16-bit by 16-bit, data shift, and data normalization operations.**
- **GPIO**
  - ✓ Up to 6 GPIOs: P3.0~P3.3, P5.4~P5.5
  - ✓ 4 modes for all GPIOs: quasi_bidirectional mode, push-pull outputmode, open drain mode, high-impedance input mode
  - ✓ **Except for P3.0 and P3.1, all other I/O ports are in high-impedance state after power-on. User must set the I/O ports mode before using them. In addition, each I/O can independently enable the internal 4K pull-up resistor.**
- **Package**
  - ✓ SOP8, DFN8 (3mm*3mm)

2.2.2 Pinouts





universal USB to UART tool

ISP download steps:

1. **Connect the universal USB to UART tool to the target chip according to the connection method shown in the figure above.**
2. **Press the power button to confirm that the target chip is in a power-off state (the power-on indicator is off).**

   Note: When the tool is powered on for the first time, there is no external power supply, so if it is the first time to use this tool, you can skip this step.
3. **Click the "Download/Program" button in the STC-ISP download software.**
4. **Press the power button again to power on the target chip (the power-on indicator is on).**
5. **Start ISP download.**

Note: It has been found that when using the USB cable for ISP download, if the USB cable is too thin and the voltage drop on the USB cable is too large, this will result in insufficient power supply during

the ISP download. Therefore, please be sure to use the booster USB cable for ISP download.

STC MCU

2.2.3 Pin descriptions

Pin number SOP8		name	type	description
1		P5.4	I/O	Standard IO port
		RST	I	Reset pin
		MCLKO	O	Master clock output
		INT2	I	External interrupt 2
		T0	I	Timer0 external input
		T1CLKO	O	Clock out of timer 1
		RxD_3	I	Serial input of UART1
		MOSI	I/O	Master Output/Slave Input of SPI
		SCL_2	I/O	Serial Clock line of I2C
2		VCC	VCC	Power Supply
3		P5.5	I/O	Standard IO port
		INT3	I	External interrupt 3
		T1	I	Timer1 external input
		T0CLKO	O	Clock out of timer 0
		TxD_3	O	Serial output of UART 1
		SS	I	Slave selection of SPI (it is output with regard to master)
		SDA_2	I/O	Serial data line of I2C
4		GND	GND	Ground
5		P3.0	I/O	Standard IO port
		RxD	I	Serial input of UART1
		INT4	I	External interrupt 4
6		P3.1	I/O	Standard IO port
		TxD	O	Serial output of UART 1
7		P3.2	I/O	Standard IO port
		INT0	I	External interrupt 0
		SCLK	I/O	Serial Clock of SPI
		SCL	I/O	Serial Clock line of I2C
		RxD_2	I	Serial input of UART1
8		P3.3	I/O	Standard IO port
		INT1	I	External interrupt 1
		MISO	I/O	Master Iutput/Slave Onput of SPI
		SDA	I/O	Serial data line of I2C
		TxD_2	O	Serial output of UART 1

2.3 STC8G1K08A-36I-SOP8/DFN8/DIP8 family

2.3.1 Features and Price

- **Selection and price (No external crystal and external reset required with 6 channels 10-bit ADC)**

MCU model	Operating voltage（V）	Flash Code Memory (100 thousand times) (Byte)	idata, Internal DATA RAM (Byte)	xdata, Internal extended SRAM (Byte)	Enhanced Dual DPTR increasing or decreasing	EEPROM 100 thousand times) (Byte)	Maximum I/O Lines	UARTs which may wake-up CPU	MDU16 Hardware 16-bit Multiplier and Divider	SPI	I²C	Timers/Counters (T0-T2 Pin can wake-up CPU)	16-bit advanced PWM timer Complementary symmetrical dead-time	15-bit enhanced PWM (with dead-time control)	PCA/CCP/PWM (can be used as external interrupt and can wake-up CPU)	Power-down wake-up timer	6-channels high speed ADCs (8 PWMs can be used as 8 DACs)	Comparator (May be used as ADC to detect external power-down)	Internal LVD interrupt (can wake-up CPU)	Watch-dog Timer	Internal high reliable reset circuit with 4-level optional reset threshold voltage	Internal high presision Clock (adjustbal under 36MHz)	Clock output and Reset	Program encrypted transmission (Anti-blocking)	Password can be set for next update	Support RS485 download	Support software USBdownload directly	Online debugging	Package			Main product supply information
																													SOP8	DFN8<3mm*3mm>	DIP8(Not recommended for use)	
STC8G1K08A	1.9-5.5	8K	256	1K	2	4K	6	1	Y	Y	Y	2	-	-	3	Y	10bit		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	√	√	√	Available
~~STC8G1K12A~~	~~1.9-5.5~~	~~12K~~	~~256~~	~~1K~~	~~2~~	~~IAP~~	~~6~~	~~1~~	~~¥~~	~~¥~~	~~¥~~	~~2~~	~~-~~	~~-~~	~~3~~	~~¥~~	~~10bit~~		~~¥~~	~~¥~~	~~¥~~	~~¥~~	~~¥~~	~~¥~~	~~¥~~	~~¥~~	~~¥~~	~~¥~~	~~¥~~				
STC8G1K17A	1.9-5.5	17K	256	1K	2	IAP	6	1	Y	Y	Y	2	-	-	3	Y	10bit		Y	Y	Y	Y	Y	Y	Y	Y	-	-	√	√	√	

Note: The above unit prices are for orders of quantity of 10K and above. If the quantity is small, an additional RMB 0.1 per piece will be required. When the total amount of the order reaches or exceeds 3,000 yuan, it can be shipped free of charge, otherwise the customer will have to bear the freight. Retail sale starts at 10 pieces.

- **Core**
  - ✓ Ultra-high speed 8051 Core with single clock per machine cycle, which is called 1T and the speed is about 12 times faster than traditional 8051
  - ✓ Fully compatible instruction set with traditional 8051
  - ✓ 13 interrupt sources and 4 interrupt priority levels
  - ✓ Online debugging is supported
- **Operating voltage**
  - ✓ 1.9V～5.5V
  - ✓ Built-in LDO
- **Operating temperature**
  - ✓ -40℃~85℃
- **Flash memory**
  - ✓ Up to 17K bytes of Flash memory to be used to store user code

- ✓ Configurable size EEPROM, 512bytes single page erased, can be repeatedly erased more than 100 thousand times.
- ✓ In-System-Programming, ISP in short, can be used to update the application code, no need for special programmer.
- ✓ Online debugging with single chip is supported, and no special emulator is needed. The number of breakpoints is unlimited theoratically.

➢ **SRAM**

- ✓ 128 bytes internal direct access RAM (DATA)
- ✓ 128 bytes internal indirect access RAM (IDATA)
- ✓ 1024 bytes internal extended RAM (internal XDATA)

➢ **Clock**

- ✓ Internal high precise R/C clock (IRC, range from 4MHz to 36MHz), adjustable while ISP and can be divided to lower frequency by user software, 100KHz for instance.
  - ⌖ Error:±0.3% (at the temperature 25℃)
  - ⌖ -1.38%～+1.42% temperature drift (at the temperature range of -40℃ to +85℃)
  - ⌖ -0.88%～+1.05% temperature drift (at the temperature range of -20℃ to 65℃)
- ✓ Internal 32KHz low speed IRC with large error

➢ **Reset**

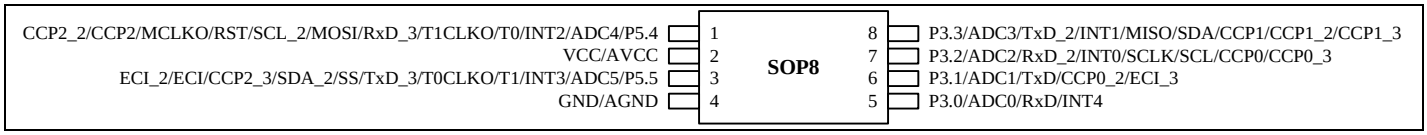
- ✓ Hardware reset
  - ⌖ Power-on reset. Measured voltage value is 1.69V~1.82V. (**Effective when the chip does not enable the low voltage reset function**)
    The power-on reset voltage is a voltage range consisting of an upper limit voltage and a lower limit voltage. When the operating voltage drops from 5V / 3.3V to the lower limit threshold voltage of the power-on reset, the chip is in a reset state; when the voltage rises from 0V to the upper threshold voltage of power-on reset, the chip is released from the reset state.
  - ⌖ Reset by reset pin. The default function of P5.4 is I/O port. The P5.4 pin can be set as the reset pin while ISP download. (**Note: When the P5.4 pin is set as the reset pin, the reset level is low**.)
  - ⌖ Watch dog timer reset
  - ⌖ Low voltage detection reset. 4 low voltage detection levels are provided, 2.2V (Measured as 1.90V~2.04V), 2.4V (Measured as 2.30V~2.50V), V2.7 (Measured as 2.61V~2.82V), V3.0 (Measured as 2.90V~3.13V). Each level of low-voltage detection voltage is a voltage range consisting of an upper limit voltage and a lower limit voltage. When the operating voltage drops from 5V / 3.3V to the lower limit threshold voltage of low-voltage detection, the low-voltage detection takes effect. When the voltage rises from 0V to the upper threshold voltage, the low voltage detection becomes effective.
- ✓ Software reset
  - ⌖ Writing the reset trigger register using software

➢ **Interrupts**

- ✓ 13 interrupt sources: INT0(Supports rising edge and falling edge interrupt), INT1(Supports rising edge and falling edge interrupt), INT2(Supports falling edge interrupt only), INT3(Supports falling edge interrupt only), INT4(Supports falling edge interrupt only), timer0, timer1, UART1, ADC, LVD, SPI, I^2C, PCA/CCP/PWM
- ✓ 4 interrupt priority levels
- ✓ Interrupts that can awaken the CPU in clock stop mode: INT0 (P3.2), INT1 (P3.3), INT2 (P3.6), INT3 (P3.7), INT4 (P3.0), T0 (P3.4), T1(P3.5), RXD(P3.0/P3.2/P1.6/P5.4), CCP0(P3.2/P3.1), CCP1 (P3.3), CCP2 (P5.4/P5.5), I2C_SDA (P3.3/P5.5) and low-voltage detection interrupt, power-down wake-up timer.

- **Digital peripherals**
  - ✓ 2 16-bit timers: timer0, timer1, where the mode 3 of timer0 has the Non Maskable Interrupt (NMI in short) function. Mode 0 of timer0 and timer1 is 16-bit Auto-reload mode.
  - ✓ 1 high speed UART: UART1, whose baudrate clock may be fast as FOSC/4
  - ✓ 3 groups of PCAs: CCP0, CCP1, CCP2, which can be used as capture, high speed output and 6-bits, 7-bits, 8-bits or 10-bits PWM.
  - ✓ SPI: Master mode, slave mode or master/slave automatic switch mode are supported.
  - ✓ I^2C: Master mode or slave mode are supported.
  - ✓ **MDU16: Hardware 16-bit Multiplier and Divider which supports 32-bit divided by 16-bit, 16-bit divided by 16-bit, 16-bit by 16-bit, data shift, and data normalization operations.**
- **Analog peripherals**
  - ✓ 6 channels (channel 0 to channel 5) ultra-high speed ADC which supports 10-bit precision analog-to-digital conversion, the speed can be as fast as 500K(500,000 conversions per second).
  - ✓ ADC channel 15 is used to test the internal reference voltage. (The default internal reference voltage is 1.19V when the chip is shipped)
  - ✓ DAC. 3 groups of PCAs can be used as DAC.
- **GPIO**
  - ✓ Up to 6 GPIOs: P3.0~P3.3, P5.4~P5.5
  - ✓ 4 modes for all GPIOs: quasi_bidirectional mode, push-pull outputmode, open drain mode, high-impedance input mode
  - ✓ **Except for P3.0 and P3.1, all other I/O ports are in a high-impedance state after power-on. User must set the I/O ports mode before using them. In addition, each I/O can independently enable the internal 4K pull-up resistor.**
- **Package**
  - ✓ SOP8, DFN8 (3mm*3mm), DIP8

2.3.2 Pinouts





universal USB to UART tool

ISP download steps:

1. **Connect the universal USB to UART tool to the target chip according to the connection method shown in the figure above.**
2. **Press the power button to confirm that the target chip is in a power-off state (the power-on indicator is off).**

Note: When the tool is powered on for the first time, there is no external power supply, so if it is the first time to use this tool, you can skip this step.

3. **Click the "Download/Program" button in the STC-ISP download software.**
4. **Press the power button again to power on the target chip (the power-on indicator is on).**
5. **Start ISP download.**

Note: It has been found that when using the USB cable for ISP download, if the USB cable is too thin and the voltage drop on the USB cable is too large, this will result in insufficient power supply during the ISP download. Therefore, please be sure to use the booster USB cable for ISP download.

2.3.3 Pin descriptions

Pin number SOP8 DFN8 DIP8		name	type	description
1		P5.4	I/O	Standard IO port
		RST	I	Reset pin
		MCLKO	O	Master clock output
		INT2	I	External interrupt 2
		T0	I	Timer0 external input
		T1CLKO	O	Clock out of timer 1
		RxD_3	I	Serial input of UART1
		MOSI	I/O	Master Output/Slave Input of SPI
		SCL_2	I/O	Serial Clock line of I2C
		ADC4	I	ADC analog input 4
		CCP2	I/O	Capture of external signal/High-speed Pulse output of PCA
		CCP2_2	I/O	Capture of external signal/High-speed Pulse output of PCA
2		VCC	VCC	Power Supply
		AVCC	VCC	ADC Power Supply
3		P5.5	I/O	Standard IO port
		INT3	I	External interrupt 3
		T1	I	Timer1 external input
		T0CLKO	O	Clock out of timer 0
		TxD_3	O	Serial output of UART 1
		SS	I	Slave selection of SPI (it is output with regard to master)
		SDA_2	I/O	Serial data line of I2C
		ADC5	I	ADC analog input 5
		ECI	I	External pulse input of PCA
		ECI_2	I	External pulse input of PCA
		CCP2_3	I/O	Capture of external signal/High-speed Pulse output of PCA
4		GND	GND	Ground
		AGND	GND	ADC Ground
5		P3.0	I/O	Standard IO port
		RxD	I	Serial input of UART1
		INT4	I	External interrupt 4
		ADC0	I	ADC analog input 0
6		P3.1	I/O	Standard IO port
		TxD	O	Serial output of UART 1
		ADC1	I	ADC analog input 1
		ECI_3	I	External pulse input of PCA
		CCP0_2	I/O	Capture of external signal/High-speed Pulse output of PCA

Pin number		name	type	description
SOP8 DFN8 DIP8				
7		P3.2	I/O	Standard IO port
		INT0	I	External interrupt 0
		SCLK	I/O	Serial Clock of SPI
		SCL	I/O	Serial Clock line of I2C
		RxD_2	I	Serial input of UART1
		ADC2	I	ADC analog input 2
		CCP0	I/O	Capture of external signal/High-speed Pulse output of PCA
		CCP0_3	I/O	Capture of external signal/High-speed Pulse output of PCA
8		P3.3	I/O	Standard IO port
		INT1	I	External interrupt 1
		MISO	I/O	Master Iutput/Slave Onput of SPI
		SDA	I/O	Serial data line of I2C
		TxD_2	O	Serial output of UART 1
		ADC3	I	ADC analog input 3
		CCP1	I/O	Capture of external signal/High-speed Pulse output of PCA
		CCP1_2	I/O	Capture of external signal/High-speed Pulse output of PCA
		CCP1_3	I/O	Capture of external signal/High-speed Pulse output of PCA

STC MCU

2.4 STC8G2K64S4-36I-LQFP48/QFN48/LQFP32/QFN32 family

2.4.1 Features and Price

➢ **Selection and price (No external crystal and external reset required with 15 channels 10-bit ADC)**

MCU model	Operating voltage（V）	Flash Code Memory (100 thousand times) (Byte)	idata, Internal DATA RAM(Byte)	xdata, Internal extended SRAM (Byte)	Enhanced Dual DPTR increasing or decreasing	EEPROM 100 thousand times) (Byte)	Maximum I/O Lines	UARTs which may wake-up CPU	MDU16 Hardware 16-bit Multiplier and Divider	SPI	I²C	Timers/Counters (T0-T4 Pin Can wake-up CPU)	16-bit advanced PWM timer Complementary symmetrical dead-time	15-bit enhanced PWM (with dead-time control)	PCA/CCP/PWM (can be used as external interrupt and can wake-up CPU)	Power-down wake-up timer	15 channels high speed ADC (All PWMs can be used as DACs)	Comparator (May be used as ADC to detect external power-down)	Internal LVD interrupt (can wake-up CPU)	Watch-dog Timer	Internal high reliable reset circuit with 4-level optional reset threshold voltage	Internal high presision Clock (adjustbal under 36MHz)	Clock output and Reset	Program encrypted transmission (Anti-blocking)	Password can be set for next update	Support RS485 download	Support software USBdownload directly	Online debugging	Package				Main product supply information
																													LQFP48	QFN48<6mm*6mm>	LQFP32	QFN32<4mm*4mm>	
STC8G2K32S4	1.9-5.5	32K	256	2K	2	32K	45	4	Y	Y	Y	5	-	45	3	Y	10bit	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	√	√	√		Available
STC8G2K48S4	1.9-5.5	48K	256	2K	2	32K	45	4	Y	Y	Y	5	-	45	3	Y	10bit	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	√	√	√		
STC8G2K60S4	1.9-5.5	60K	256	2K	2	4K	45	4	Y	Y	Y	5	-	45	3	Y	10bit	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	√	√	√		
STC8G2K64S4	1.9-5.5	64K	256	2K	2	IAP	45	4	Y	Y	Y	5	-	45	3	Y	10bit	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	√	√	√	√	

Note: The above unit prices are for orders of quantity of 10K and above. If the quantity is small, an additional RMB 0.1 per piece will be required. When the total amount of the order reaches or exceeds 3,000 yuan, it can be shipped free of charge, otherwise the customer will have to bear the freight. Retail sale starts at 10 pieces.

If you need to choose a 48-pin chip, it is recommended to choose LQFP48 package, QFN48 needs to be ordered in advance.

➢ **Core**

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

➢ **Operating voltage**

- ✓ 1.9V～5.5V
- ✓ Built-in LDO

➢ **Operating temperature**

- ✓ -40℃~85℃

- **Flash memory**
  - ✓ Up to 64K bytes of Flash memory to be used to store user code
  - ✓ Configurable size EEPROM, 512bytes single page erased, can be repeatedly erased more than 100 thousand times.
  - ✓ In-System-Programming, ISP in short, can be used to update the application code, no need for special programmer.
  - ✓ Online debugging with single chip is supported, and no special emulator is needed. The number of breakpoints is unlimited theoratically.

- **SRAM**
  - ✓ 128 bytes internal direct access RAM (DATA)
  - ✓ 128 bytes internal indirect access RAM (IDATA)
  - ✓ 2048 bytes internal extended RAM (internal XDATA)

- **Clock**
  - ✓ Internal high precise R/C clock (IRC, range from 4MHz to 36MHz), adjustable while ISP and can be divided to lower frequency by user software, 100KHz for instance.
    - ⌖ Error:±0.3% (at the temperature 25℃)
    - ⌖ -1.38%～+1.42% temperature drift (at the temperature range of -40℃ to +85℃)
    - ⌖ -0.88%～+1.05% temperature drift (at the temperature range of -20℃ to 65℃)
  - ✓ Internal 32KHz low speed IRC with large error
  - ✓ External 4MHz~33MHz oscillator or external clock

- **Reset**
  - ✓ Hardware reset
    - ⌖ Power-on reset. Measured voltage value is 1.69V~1.82V. (**Effective when the chip does not enable the low voltage reset function**)

      The power-on reset voltage is a voltage range consisting of an upper limit voltage and a lower limit voltage. When the operating voltage drops from 5V / 3.3V to the lower limit threshold voltage of the power-on reset, the chip is in a reset state; when the voltage rises from 0V to the upper threshold voltage of power-on reset, the chip is released from the reset state.
    - ⌖ Reset by reset pin. The default function of P5.4 is the I/O port. The P5.4 pin can be set as the reset pin while ISP download. (**Note: When the P5.4 pin is set as the reset pin, the reset level is low**.)
    - ⌖ Watch dog timer reset
    - ⌖ Low voltage detection reset. 4 low voltage detection levels are provided, 2.2V (Measured as 1.90V~2.04V), 2.4V (Measured as 2.30V~2.50V), V2.7 (Measured as 2.61V~2.82V), V3.0 (Measured as 2.90V~3.13V). Each level of low-voltage detection voltage is a voltage range consisting of an upper limit voltage and a lower limit voltage. When the operating voltage drops from 5V / 3.3V to the lower limit threshold voltage of low-voltage detection, the low-voltage detection takes effect. When the voltage rises from 0V to the upper threshold voltage, the low voltage detection becomes effective.
  - ✓ Software reset
    - ⌖ Writing the reset trigger register using software

- **Interrupts**
  - ✓ 29 interrupt sources: INT0(Supports rising edge and falling edge interrupt), INT1(Supports rising edge and falling edge interrupt), INT2(Supports falling edge interrupt only), INT3(Supports falling edge interrupt only), INT4(Supports falling edge interrupt only), timer0, timer1, timer2, timer3, timer4, UART1, UART2, UART3, UART4, ADC, LVD, SPI, I^2C, comparator, PCA/CCP/PWM, enhanced PWM0, enhanced PWM1, enhanced

PWM2, enhanced PWM3, enhanced PWM4, enhanced PWM5, enhanced PWM0 fault detection, enhanced PWM2 fault detection, enhanced PWM4 fault detection.

- ✓ 4 interrupt priority levels
- ✓ Interrupts that can awaken the CPU in clock stop mode: INT0 (P3.2), INT1 (P3.3), INT2 (P3.6), INT3 (P3.7), INT4 (P3.0), T0(P3.4), T1(P3.5), T2(P1.2), T3(P0.4), T4(P0.6), RXD(P3.0/P3.6/P1.6/P4.3), RXD2(P1.4/P4.6), RXD3(P0.0/P5.0), RXD4(P0.2/P5.2), CCP0(P1.1/P3.5/P2.5), CCP1(P1.0/P3.6/P2.6), CCP2(P3.7/P2.7), I2C_SDA (P1.4/P2.4/P3.3) and comparator interrupt, low-voltage detection interrupt, power-down wake-up timer.

➢ **Digital peripherals**

- ✓ 5 16-bit timers: timer0, timer1, timer2, timer3, timer4, where the mode 3 of timer0 has the Non Maskable Interrupt (NMI in short) function. Mode 0 of timer0 and timer1 is 16-bit Auto-reload mode.
- ✓ 4 high speed uarts: uart1, uart2, uart3, uart4, whose baudrate clock source may be fast as FOSC/4
- ✓ 3 groups of PCAs: CCP0, CCP1, CCP2, which can be used as capture, high speed output and 6-bits, 7-bits, 8-bits or 10-bits PWM.
- ✓ 45 groups of 15-bit enhanced PWMs, which can realize control signals with dead time, and support external fault detection function. In addition, there are 3 groups of traditional PCA/CCP/PWMs can be used as PWM.
- ✓ SPI: Master mode, slave mode or master/slave automatic switch mode are supported.
- ✓ I^2C: Master mode or slave mode are supported.
- ✓ **MDU16: Hardware 16-bit Multiplier and Divider which supports 32-bit divided by 16-bit, 16-bit divided by 16-bit, 16-bit by 16-bit, data shift, and data normalization operations.**

➢ **Analog peripherals**

- ✓ 15 channels (channel 0 to channel 14) ultra-high speed ADC which supports 10-bit precision analog-to-digital conversion, the speed can be as fast as 500K(500,000 conversions per second).
- ✓ ADC channel 15 is used to test the internal reference voltage. (The default internal reference voltage is 1.19V when the chip is shipped)
- ✓ Comparator. A set of comparators (the positive terminal of the comparator can select the CMP+ and all ADC input ports, so the comparator can be used as a multi-channel comparator for time division multiplexing).
- ✓ DAC. 3 groups of PCAs can be used as DACs. 45-channel enhanced PWMs can be used as 45-channel DACs.

➢ **GPIO**

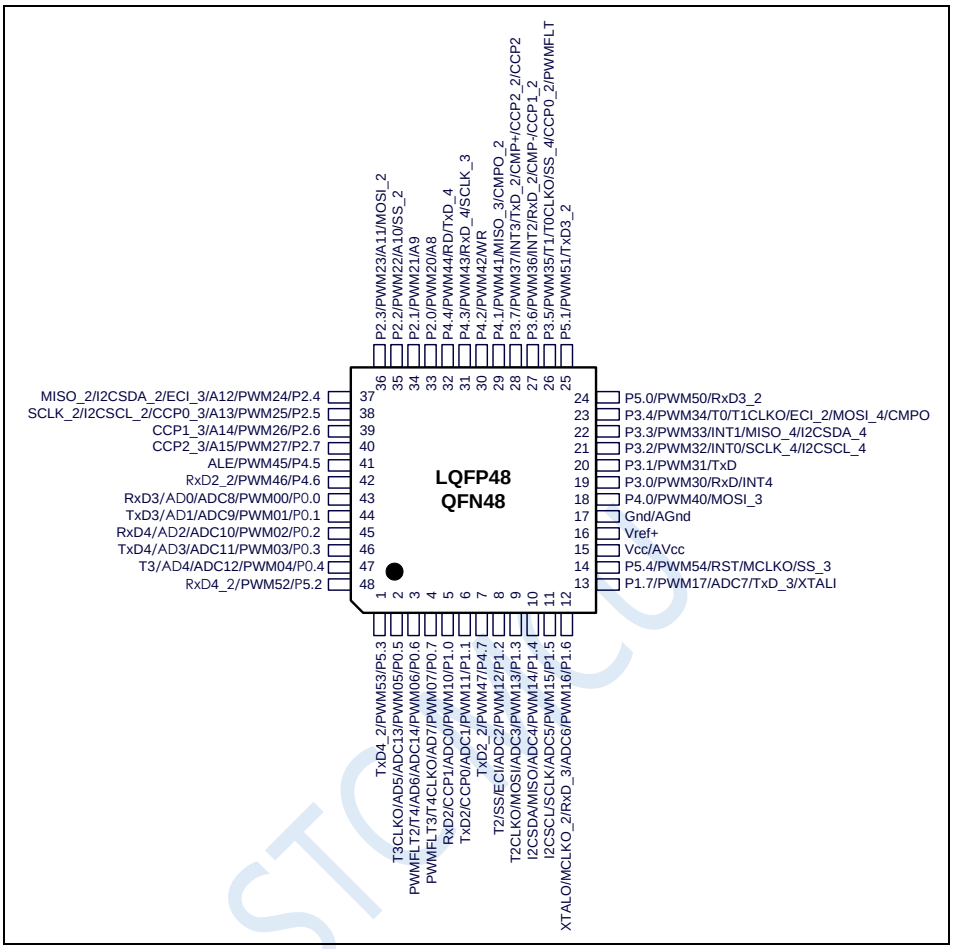
- ✓ Up to 45 GPIOs: P0.0~P0.7, P1.0~P1.7, P2.0~P2.7, P3.0~P3.7, P4.0~P4.7, P5.0~P5.4
- ✓ 4 modes for all GPIOs: quasi_bidirectional mode, push-pull outputmode, open drain mode, high-impedance input mode
- ✓ **Except for P3.0 and P3.1, all other I/O ports are in a high-impedance state after power-on. User must set the I/O ports mode before using them. In addition, each I/O can independently enable the internal 4K pull-up resistor.**

➢ **Package**

- ✓ LQFP48, QFN48, LQFP32(Not available at present)

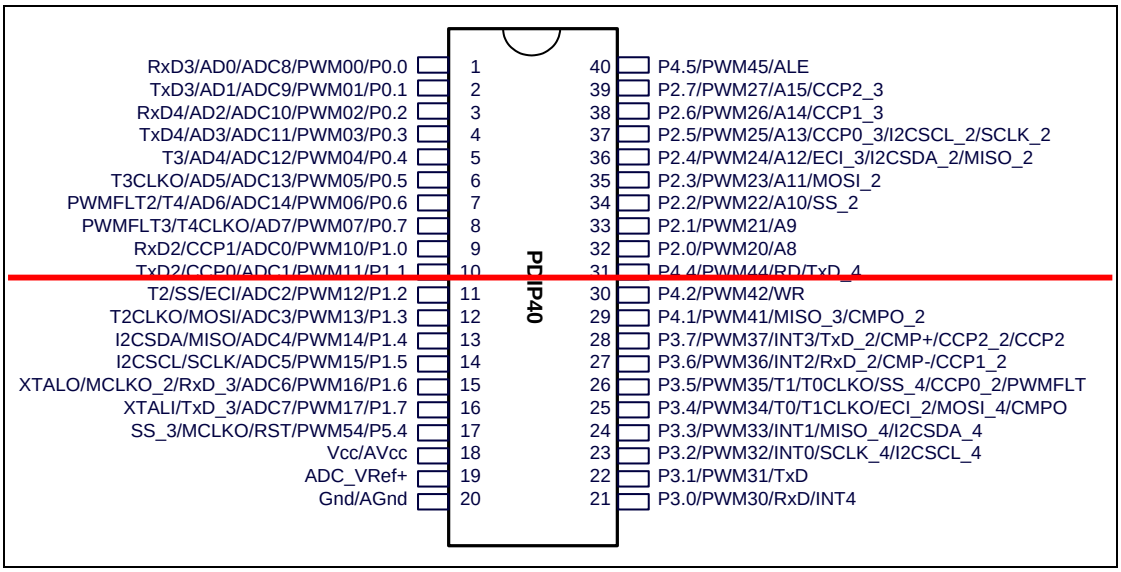
STCMCU

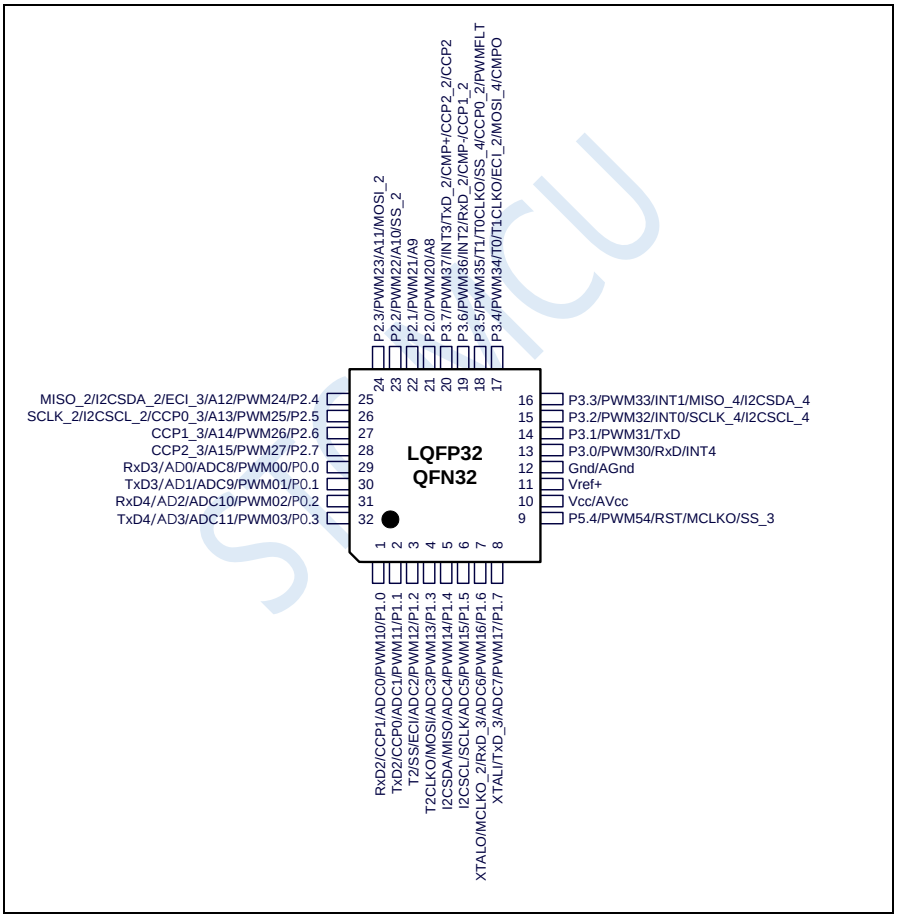
2.4.2 Pinouts





universal USB to UART tool





Note:

1. ADC's external reference power supply pin ADC_VRef+ must not be floating, it must be connected to an external reference power supply or directly connected to Vcc.

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

ISP download steps:

1. Connect the universal USB to UART tool to the target chip according to the connection method shown in the figure above.

2. Press the power button to confirm that the target chip is in a power-off state (the power-on

indicator is off).

Note: When the tool is powered on for the first time, there is no external power supply, so if it is the first time to use this tool, you can skip this step.

3. **Click the "Download/Program" button in the STC-ISP download software.**
4. **Press the power button again to power on the target chip (the power-on indicator is on).**
5. **Start ISP download.**

Note: It has been found that when using the USB cable for ISP download, if the USB cable is too thin and the voltage drop on the USB cable is too large, this will result in insufficient power supply during the ISP download. Therefore, please be sure to use the booster USB cable for ISP download.

2.4.3 Pin descriptions

Pin number		name	type	description
LQFP48				
1		P5.3	I/O	Standard IO port
		PWM53	O	Enhanced PWM output
		TxD4_2	O	Serial output of UART 4
2		P0.5	I/O	Standard IO port
		PWM05	O	Enhanced PWM output
		AD5	I	Address/data bus
		ADC13	I	ADC analog input 13
		T3CLKO	O	Clock out of timer 3
3		P0.6	I/O	Standard IO port
		PWM06	O	Enhanced PWM output
		AD6	I	Address/data bus
		ADC14	I	ADC analog input 14
		T4	I	Timer4 external input
		PWMFLT2	I	Enhanced PWM fault detection
4		P0.7	I/O	Standard IO port
		PWM07	O	Enhanced PWM output
		AD7	I	Address/data bus
		T4CLKO	O	Clock out of timer 4
		PWMFLT3	I	Enhanced PWM fault detection
5		P1.0	I/O	Standard IO port
		PWM10	O	Enhanced PWM output
		ADC0	I	ADC analog input 0
		CCP1	I/O	Capture of external signal/High-speed Pulse output of PCA
		RxD2	I	Serial input of UART2
6		P1.1	I/O	Standard IO port
		PWM11	O	Enhanced PWM output
		ADC1	I	ADC analog input 1
		CCP0	I/O	Capture of external signal/High-speed Pulse output of PCA
		TxD2	O	Serial output of UART 2
7		P4.7	I/O	Standard IO port
		PWM47	O	Enhanced PWM output
		TxD2_2	O	Serial output of UART 2
8		P1.2	I/O	Standard IO port
		PWM12	O	Enhanced PWM output
		ADC2	I	ADC analog input 2

		ECI	I	External pulse input of PCA
		SS	I	Slave selection of SPI (it is output with regard to master)
		T2	I	Timer2 external input
9		P1.3	I/O	Standard IO port
		PWM13	O	Enhanced PWM output
		ADC3	I	ADC analog input 3
		MOSI	I/O	Master Output/Slave Input of SPI
		T2CLKO	O	Clock out of timer 2
10		P1.4	I/O	Standard IO port
		PWM14	O	Enhanced PWM output
		ADC4	I	ADC analog input 4
		MISO	I/O	Master Iutput/Slave Onput of SPI
		SDA	I/O	Serial data line of I2C

STC MCU

Pin number		name	type	description
LQFP48				
11		P1.5	I/O	Standard IO port
		PWM15	O	Enhanced PWM output
		ADC5	I	ADC analog input 5
		SCLK	I/O	Serial Clock of SPI
		SCL	I/O	Serial Clock line of I2C
12		P1.6	I/O	Standard IO port
		PWM16	O	Enhanced PWM output
		ADC6	I	ADC analog input 6
		RxD_3	I	Serial input of UART1
		MCLKO_2	O	Master clock output
		XTALO	O	Connect to external oscillator
13		P1.7	I/O	Standard IO port
		PWM17	O	Enhanced PWM output
		ADC7	I	ADC analog input 7
		TxD_3	O	Serial output of UART 1
		XTALI	I	Connect to external oscillator
14		P5.4	I/O	Standard IO port
		PWM54	O	Enhanced PWM output
		RST	I	Reset pin
		MCLKO	O	Master clock output
		SS_3	I	Slave selection of SPI (it is output with regard to master)
15		Vcc	VCC	Power Supply
		AVcc	VCC	ADC Power Supply
16		Vref+	I	Reference voltage pin of ADC
17		Gnd	GND	Ground
		AGnd	GND	ADC Ground
18		P4.0	I/O	Standard IO port
		PWM40	O	Enhanced PWM output
		MOSI_3	I/O	Master Output/Slave Input of SPI
19		P3.0	I/O	Standard IO port
		PWM30	O	Enhanced PWM output
		RxD	I	Serial input of UART1
		INT4	I	External interrupt 4
20		P3.1	I/O	Standard IO port
		PWM31	O	Enhanced PWM output
		TxD	O	Serial output of UART 1
21		P3.2	I/O	Standard IO port
		PWM32	O	Enhanced PWM output
		INT0	I	External interrupt 0
		SCLK_4	I/O	Serial Clock of SPI
		SCL_4	I/O	Serial Clock line of I2C
22		P3.3	I/O	Standard IO port
		PWM33	O	Enhanced PWM output
		INT1	I	External interrupt 1
		MISO_4	I/O	Master Iutput/Slave Onput of SPI
		SDA_4	I/O	Serial data line of I2C
23		P3.4	I/O	Standard IO port
		PWM34	O	Enhanced PWM output
		T0	I	Timer0 external input
		T1CLKO	O	Clock out of timer 1
		ECI_2	I	External pulse input of PCA
		MOSI_4	I/O	Master Output/Slave Input of SPI
		CMPO	O	Comparator output

Pin number		name	type	description
LQFP48				
24		P5.0	I/O	Standard IO port
		PWM50	O	Enhanced PWM output
		RxD3_2	I	Serial input of UART3
25		P5.1	I/O	Standard IO port
		PWM51	O	Enhanced PWM output
		TxD3_2	O	Serial output of UART 3
26		P3.5	I/O	Standard IO port
		PWM35	O	Enhanced PWM output
		T1	I	Timer1 external input
		T0CLKO	O	Clock out of timer 0
		SS_4	I	Slave selection of SPI (it is output with regard to master)
		CCP0_2	I/O	Capture of external signal/High-speed Pulse output of PCA
		PWMFLT	I	Enhanced PWM fault detection
27		P3.6	I/O	Standard IO port
		PWM36	O	Enhanced PWM output
		INT2	I	External interrupt 2
		RxD_2	I	Serial input of UART1
		CMP-	I	Comparator negative input
		CCP1_2	I/O	Capture of external signal/High-speed Pulse output of PCA
28		P3.7	I/O	Standard IO port
		PWM37	O	Enhanced PWM output
		INT3	I	External interrupt 3
		TxD_2	O	Serial output of UART 1
		CMP+	I	Comparator positive input
		CCP2	I/O	Capture of external signal/High-speed Pulse output of PCA
		CCP2_2	I/O	Capture of external signal/High-speed Pulse output of PCA
29		P4.1	I/O	Standard IO port
		PWM41	O	Enhanced PWM output
		MISO_3	I/O	Master Iutput/Slave Onput of SPI
		CMPO_2	O	Comparator output
30		P4.2	I/O	Standard IO port
		PWM42	O	Enhanced PWM output
		WR	O	Write signal of external bus
31		P4.3	I/O	Standard IO port
		PWM43	O	Enhanced PWM output
		RxD_4	I	Serial input of UART1
		SCLK_3	I/O	Serial Clock of SPI
32		P4.4	I/O	Standard IO port
		PWM44	O	Enhanced PWM output
		RD	O	Read signal of external bus
		TxD_4	O	Serial output of UART 1
33		P2.0	I/O	Standard IO port
		PWM20	O	Enhanced PWM output
		A8	I	Address bus
34		P2.1	I/O	Standard IO port
		PWM21	O	Enhanced PWM output
		A9	I	Address bus
35		P2.2	I/O	Standard IO port
		PWM22	O	Enhanced PWM output
		A10	I	Address bus
		SS_2	I	Slave selection of SPI (it is output with regard to master)

Pin number LQFP48		name	type	description
36		P2.3	I/O	Standard IO port
		PWM23	O	Enhanced PWM output
		A11	I	Address bus
		MOSI_2	I/O	Master Output/Slave Input of SPI
		CCP0_2	I/O	Capture of external signal/High-speed Pulse output of PCA
37		P2.4	I/O	Standard IO port
		PWM24	O	Enhanced PWM output
		A12	I	Address bus
		ECI_3	I	External pulse input of PCA
		SDA_2	I/O	Serial data line of I2C
		MISO_2	I/O	Master Iutput/Slave Onput of SPI
38		P2.5	I/O	Standard IO port
		PWM25	O	Enhanced PWM output
		A13	I	Address bus
		CCP0_3	I/O	Capture of external signal/High-speed Pulse output of PCA
		SCL_2	I/O	Serial Clock line of I2C
		SCLK_2	I/O	Serial Clock of SPI
39		P2.6	I/O	Standard IO port
		PWM26	O	Enhanced PWM output
		A14	I	Address bus
		CCP1_3	I/O	Capture of external signal/High-speed Pulse output of PCA
40		P2.7	I/O	Standard IO port
		PWM27	O	Enhanced PWM output
		A15	I	Address bus
		CCP2_3	I/O	Capture of external signal/High-speed Pulse output of PCA
41		P4.5	I/O	Standard IO port
		PWM45	O	Enhanced PWM output
		ALE	O	Address Latch Enable signal
42		P4.6	I/O	Standard IO port
		PWM46	O	Enhanced PWM output
		RxD2_2	I	Serial input of UART2
43		P0.0	I/O	Standard IO port
		PWM00	O	Enhanced PWM output
		ADC8	I	ADC analog input 8
		AD0	I	Address/data bus
		RxD3	I	Serial input of UART3
44		P0.1	I/O	Standard IO port
		PWM01	O	Enhanced PWM output
		ADC9	I	ADC analog input 9
		AD1	I	Address/data bus
		TxD3	O	Serial output of UART 3
45		P0.2	I/O	Standard IO port
		PWM02	O	Enhanced PWM output
		ADC10	I	ADC analog input 10
		AD2	I	Address/data bus
		RxD4	I	Serial input of UART4
46		P0.3	I/O	Standard IO port
		PWM03	O	Enhanced PWM output
		ADC11	I	ADC analog input 11
		AD3	I	Address/data bus
		TxD4	O	Serial output of UART 4

Pin number		name	type	description
LQFP48				
47		P0.4	I/O	Standard IO port
		PWM04	O	Enhanced PWM output
		ADC12	I	ADC analog input 12
		AD4	I	Address/data bus
		T3	I	Timer3 external input
48		P5.2	I/O	Standard IO port
		PWM52	O	Enhanced PWM output
		RxD4_2	I	Serial input of UART4

STC MCU

2.5 STC8G2K64S2-36I-LQFP48/QFN48 family

Note: STC8G2K64S2 series only P2 port has enhanced PWM, other ports do not have.

2.5.1 Features and Price

- **Selection and price (No external crystal and external reset required with 15 channels 10-bit ADC)**

MCU model	Operating voltage（V）	Flash Code Memory (100 thousand times) (Byte)	idata, internal DATA RAM (Byte)	xdata, internal extended SRAM (Byte)	Enhanced Dual DPTR increasing or decreasing	EEPROM 100 thousand times) (Byte)	Maximum I/O Lines	UARTs which may wake-up CPU	MDU16 Hardware 16-bit Multiplier and Divider	SPI	I²C	Timers/Counters (T0-T4 Pin can wake-up CPU)	16-bit advanced PWM timer Complementary symmetrical dead-time	15-bit enhanced PWM (with dead-time control)	PCA/CCP/PWM (can be used as external interrupt and can wake-up CPU)	Power-down Wake-up timer	15-channels high speed ADC (All PWMs can be used as DACs)	Comparator (May be used as ADC to detect external power-down)	Internal LVD interrupt (can wake-up CPU)	Watch-dog Timer	Internal high reliable reset circuit with 4-level optional reset threshold voltage	Internal high presision Clock (adjustbal under 36MHz)	Clock output and Reset	Program encrypted transmission (Anti-blocking)	Support RS485 download	Password can be set for next update	Support software USBdownload directly	Online debugging	Package		Main product supply information
																													LQFP48	QFN48<6mm*6mm>	
STC8G2K16S2	1.9-5.5	16K	256	2K	2	48K	45	2	Y	Y	Y	5	-	8	3	Y	10bit	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	√	√	Available
STC8G2K32S2	1.9-5.5	32K	256	2K	2	32K	45	2	Y	Y	Y	5	-	8	3	Y	10bit	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	√	√	
STC8G2K48S2	1.9-5.5	48K	256	2K	2	16K	45	2	Y	Y	Y	5	-	8	3	Y	10bit	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	√	√	
~~STC8G2K60S2~~	~~1.9-5.5~~	~~60K~~	~~256~~	~~2K~~	~~2~~	~~4K~~	~~45~~	~~2~~	~~Y~~	~~Y~~	~~Y~~	~~5~~	~~-~~	~~8~~	~~3~~	~~Y~~	~~10bit~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~			
~~STC8G2K64S2~~	~~1.9-5.5~~	~~64K~~	~~256~~	~~2K~~	~~2~~	~~IAP~~	~~45~~	~~2~~	~~Y~~	~~Y~~	~~Y~~	~~5~~	~~-~~	~~8~~	~~3~~	~~Y~~	~~10bit~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~	~~Y~~			

Note: The above unit prices are for orders of quantity of 10K and above. If the quantity is small, an additional RMB 0.1 per piece will be required. When the total amount of the order reaches or exceeds 3,000 yuan, it can be shipped free of charge, otherwise the customer will have to bear the freight. Retail sale starts at 10 pieces.

- **Core**
  - ✓ Ultra-high speed 8051 Core with single clock per machine cycle, which is called 1T and the speed is about 12 times faster than traditional 8051
  - ✓ Fully compatible instruction set with traditional 8051
  - ✓ 27 interrupt sources and 4 interrupt priority levels
  - ✓ Online debugging is supported
- **Operating voltage**
  - ✓ 1.9V～5.5V

- ✓ Built-in LDO

➢ **Operating temperature**

- ✓ -40℃~85℃

➢ **Flash memory**

- ✓ Up to 64Kbytes of Flash memory to be used to store user code
- ✓ Configurable size EEPROM, 512bytes single page erased, can be repeatedly erased more than 100 thousand times.
- ✓ In-System-Programming, ISP in short, can be used to update the application code, no need for special programmer.
- ✓ Online debugging with single chip is supported, and no special emulator is needed. The number of breakpoints is unlimited theoratically.

➢ **SRAM**

- ✓ 128 bytes internal direct access RAM (DATA)
- ✓ 128 bytes internal indirect access RAM (IDATA)
- ✓ 2048 bytes internal extended RAM (internal XDATA)

➢ **Clock**

- ✓ Internal high precise R/C clock (IRC, range from 4MHz to 36MHz), adjustable while ISP and can be divided to lower frequency by user software, 100KHz for instance.
  - ⌖ Error:±0.3% (at the temperature 25℃)
  - ⌖ -1.38%～+1.42% temperature drift (at the temperature range of -40℃ to +85℃)
  - ⌖ -0.88%～+1.05% temperature drift (at the temperature range of -20℃ to 65℃)
- ✓ Internal 32KHz low speed IRC with large error
- ✓ External 4MHz~33MHz oscillator or external clock

➢ **Reset**

- ✓ Hardware reset
  - ⌖ Power-on reset. Measured voltage value is 1.69V~1.82V. (**Effective when the chip does not enable the low voltage reset function**)
    The power-on reset voltage is a voltage range consisting of an upper limit voltage and a lower limit voltage. When the operating voltage drops from 5V / 3.3V to the lower limit threshold voltage of the power-on reset, the chip is in a reset state; when the voltage rises from 0V to the upper threshold voltage of power-on reset, the chip is released from the reset state.
  - ⌖ Reset by reset pin. The default function of P5.4 is the I/O port. The P5.4 pin can be set as the reset pin while ISP download. (**Note: When the P5.4 pin is set as the reset pin, the reset level is low**.)
  - ⌖ Watch dog timer reset
  - ⌖ Low voltage detection reset. 4 low voltage detection levels are provided, 2.2V (Measured as 1.90V~2.04V), 2.4V (Measured as 2.30V~2.50V), V2.7 (Measured as 2.61V~2.82V), V3.0 (Measured as 2.90V~3.13V). Each level of low-voltage detection voltage is a voltage range consisting of an upper limit voltage and a lower limit voltage. When the operating voltage drops from 5V / 3.3V to the lower limit threshold voltage of low-voltage detection, the low-voltage detection takes effect. When the voltage rises from 0V to the upper threshold voltage, the low voltage detection becomes effective.
- ✓ Software reset
  - ⌖ Writing the reset trigger register using software

➢ **Interrupts**

- ✓ 27 interrupt sources: INT0(Supports rising edge and falling edge interrupt), INT1(Supports rising edge and falling edge interrupt), INT2(Supports falling edge interrupt only), INT3(Supports falling edge interrupt only), INT4(Supports falling edge interrupt only), timer0, timer1, timer2, timer3, timer4, UART1, UART2, ADC, LVD, SPI, I^2C, comparator, PCA/CCP/PWM, enhanced PWM2, enhanced PWM2 fault detection.
- ✓ 4 interrupt priority levels
- ✓ Interrupts that can awaken the CPU in clock stop mode: INT0 (P3.2), INT1 (P3.3), INT2 (P3.6), INT3 (P3.7), INT4 (P3.0), T0 (P3.4), T1(P3.5), T2(P1.2), T3(P0.4), T4(P0.6), RXD(P3.0/P3.6/P1.6/P4.3), RXD2(P1.4/P4.6), CCP0(P1.1/P3.5/P2.5), CCP1(P1.0/P3.6/P2.6), CCP2 (P3.7/P2.7), I2C_SDA (P1.4/P2.4/P3.3) and comparator interrupt, low-voltage detection interrupt, power-down wake-up timer.

➢ **Digital peripherals**

- ✓ 5 16-bit timers: timer0, timer1, timer2, timer3, timer4, where the mode 3 of timer0 has the Non Maskable Interrupt (NMI in short) function. Mode 0 of timer0 and timer1 is 16-bit Auto-reload mode.
- ✓ 2 high speed UARTs: UART1, UART2, whose baudrate clock source may be fast as FOSC/4
- ✓ 3 groups of 16-bit PCAs: CCP0, CCP1, CCP2, which can be used as capture, high speed output and 6-bits, 7-bits, 8-bits or 10-bits PWM.
- ✓ 8 groups of 15-bit enhanced PWMs, which can realize control signals with dead-time, and support external fault detection function. In addition, there are 3 groups of traditional PCA / CCP / PWM can be used as PWM.
- ✓ SPI: Master mode, slave mode or master/slave automatic switch mode are supported.
- ✓ I^2C: Master mode or slave mode are supported.
- ✓ **MDU16: Hardware 16-bit Multiplier and Divider which supports 32-bit divided by 16-bit, 16-bit divided by 16-bit, 16-bit by 16-bit, data shift, and data normalization operations.**

➢ **Analog peripherals**

- ✓ 15 channels (channel 0 to channel 14) ultra-high speed ADC which supports 10-bit precision analog-to-digital conversion.
- ✓ ADC channel 15 is used to test the internal reference voltage. (The default internal reference voltage is 1.19V when the chip is shipped)
- ✓ Comparator. A set of comparators (the positive terminal of the comparator can select the CMP+ and all ADC input ports, so the comparator can be used as a multi-channel comparator for time division multiplexing).
- ✓ DAC. 3 groups of PCAs can be used as DACs. 8 channels enhanced PWMs can be used as DACs.

➢ **GPIO**

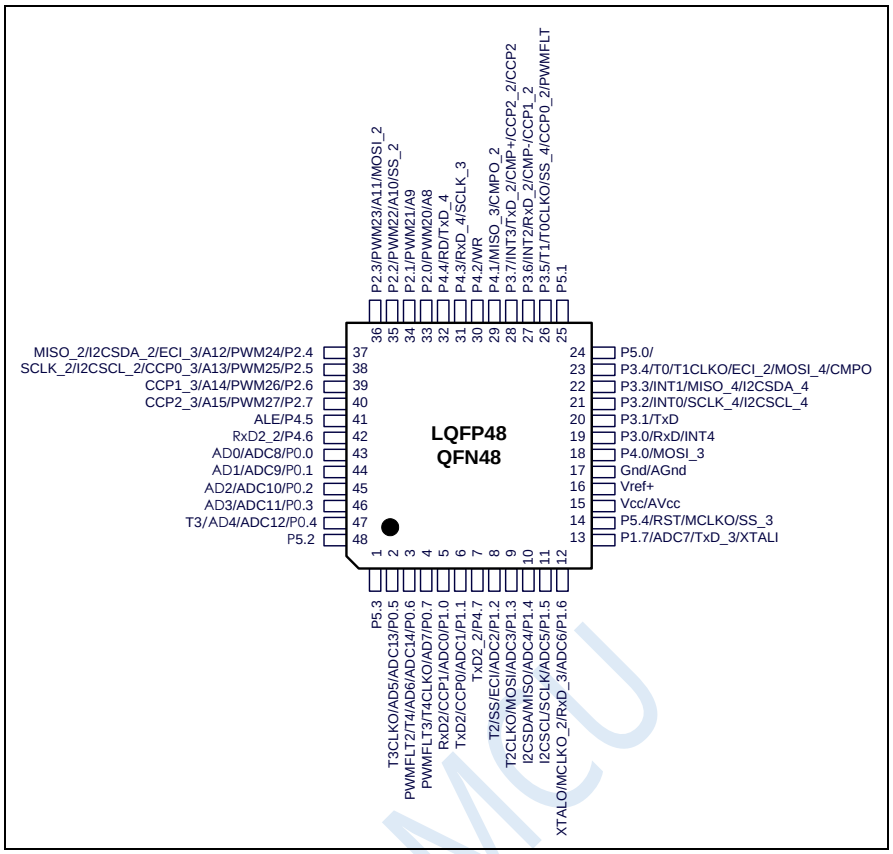
- ✓ Up to 45 GPIOs: P0.0~P0.7, P1.0~P1.7, P2.0~P2.7, P3.0~P3.7, P4.0~P4.7, P5.0~P5.4
- ✓ 4 modes for all GPIOs: quasi_bidirectional mode, push-pull outputmode, open drain mode, high-impedance input mode
- ✓ **Except for P3.0 and P3.1, all other I/O ports are in a high-impedance state after power-on. User must set the I/O ports mode before using them. In addition, each I/O can independently enable the internal 4K pull-up resistor.**

➢ **Package**

- ✓ LQFP48, QFN48

STC MCU

2.5.2 Pinouts



Note:

1. ADC's external reference power supply pin ADC_VRef+ must not be floating, it must be connected to an external reference power supply or directly connected to Vcc.
2. If USB download is not required, P3.0/P3.1/P3.2 cannot be at low level at the same time when the chip is reset.



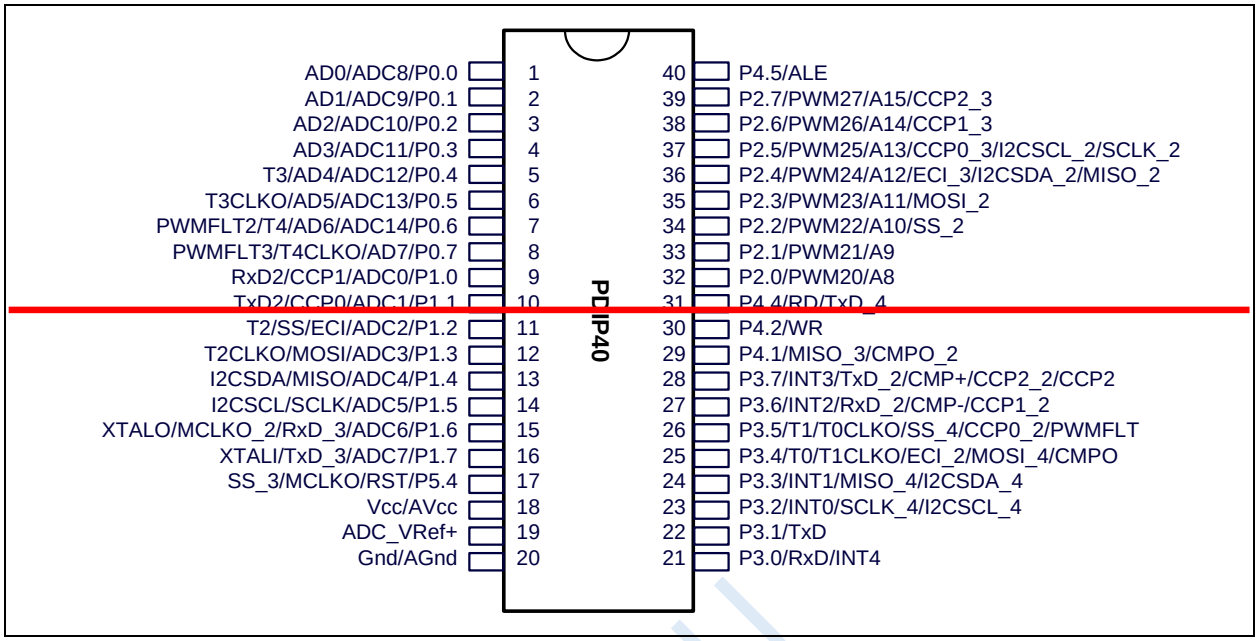
universal USB to UART tool

ISP download steps:

1. **Connect the universal USB to UART tool to the target chip according to the connection method shown in the figure above.**
2. **Press the power button to confirm that the target chip is in a power-off state (the power-on indicator is off).**

   Note: When the tool is powered on for the first time, there is no external power supply, so if it is the first time to use this tool, you can skip this step.
3. **Click the "Download/Program" button in the STC-ISP download software.**
4. **Press the power button again to power on the target chip (the power-on indicator is on).**
5. **Start ISP download.**

Note: It has been found that when using the USB cable for ISP download, if the USB cable is too thin and the voltage drop on the USB cable is too large, this will result in insufficient power supply during the ISP download. Therefore, please be sure to use the booster USB cable for ISP download.



2.5.3 Pin descriptions

Pin number LQFP48		name	type	description
1		P5.3	I/O	Standard IO port
2		P0.5	I/O	Standard IO port
		AD5	I	Address/data bus
		ADC13	I	ADC analog input 13
		T3CLKO	O	Clock out of timer 3
3		P0.6	I/O	Standard IO port
		AD6	I	Address/data bus
		ADC14	I	ADC analog input 14
		T4	I	Timer4 external input
		PWMFLT2	I	Enhanced PWM fault detection
4		P0.7	I/O	Standard IO port
		AD7	I	Address/data bus
		T4CLKO	O	Clock out of timer 4
		PWMFLT3	I	Enhanced PWM fault detection
5		P1.0	I/O	Standard IO port
		ADC0	I	ADC analog input 0
		CCP1	I/O	Capture of external signal/High-speed Pulse output of PCA
		RxD2	I	Serial input of UART2
6		P1.1	I/O	Standard IO port
		ADC1	I	ADC analog input 1
		CCP0	I/O	Capture of external signal/High-speed Pulse output of PCA
		TxD2	O	Serial output of UART 2
7		P4.7	I/O	Standard IO port
		TxD2_2	O	Serial output of UART 2
8		P1.2	I/O	Standard IO port
		ADC2	I	ADC analog input 2

		ECI	I	External pulse input of PCA
		SS	I	Slave selection of SPI (it is output with regard to master)
		T2	I	Timer2 external input
Pin number		**name**	**type**	**description**
LQFP48				
9		P1.3	I/O	Standard IO port
		ADC3	I	ADC analog input 3
		MOSI	I/O	Master Output/Slave Input of SPI
		T2CLKO	O	Clock out of timer 2
10		P1.4	I/O	Standard IO port
		ADC4	I	ADC analog input 4
		MISO	I/O	Master Iutput/Slave Onput of SPI
		SDA	I/O	Serial data line of I2C
11		P1.5	I/O	Standard IO port
		ADC5	I	ADC analog input 5
		SCLK	I/O	Serial Clock of SPI
		SCL	I/O	Serial Clock line of I2C
12		P1.6	I/O	Standard IO port
		ADC6	I	ADC analog input 6
		RxD_3	I	Serial input of UART1
		MCLKO_2	O	Master clock output
		XTALO	O	Connect to external oscillator
13		P1.7	I/O	Standard IO port
		ADC7	I	ADC analog input 7
		TxD_3	O	Serial output of UART 1
		XTALI	I	Connect to external oscillator
14		P5.4	I/O	Standard IO port
		RST	I	Reset pin
		MCLKO	O	Master clock output
		SS_3	I	Slave selection of SPI (it is output with regard to master)
15		Vcc	VCC	Power Supply
		AVcc	VCC	ADC Power Supply
16		Vref+	I	Reference voltage pin of ADC
17		Gnd	GND	Ground
		AGnd	GND	ADC Ground
18		P4.0	I/O	Standard IO port
		MOSI_3	I/O	Master Output/Slave Input of SPI
19		P3.0	I/O	Standard IO port
		RxD	I	Serial input of UART1
		INT4	I	External interrupt 4
20		P3.1	I/O	Standard IO port
		TxD	O	Serial output of UART 1
21		P3.2	I/O	Standard IO port
		INT0	I	External interrupt 0
		SCLK_4	I/O	Serial Clock of SPI
		SCL_4	I/O	Serial Clock line of I2C
22		P3.3	I/O	Standard IO port
		INT1	I	External interrupt 1
		MISO_4	I/O	Master Iutput/Slave Onput of SPI
		SDA_4	I/O	Serial data line of I2C
23		P3.4	I/O	Standard IO port
		T0	I	Timer0 external input
		T1CLKO	O	Clock out of timer 1
		ECI_2	I	External pulse input of PCA
		MOSI_4	I/O	Master Output/Slave Input of SPI
		CMPO	O	Comparator output
24		P5.0	I/O	Standard IO port

		RxD3_2	I	Serial input of UART3
25		P5.1	I/O	Standard IO port

Pin number		name	type	description
LQFP48				
26		P3.5	I/O	Standard IO port
		T1	I	Timer1 external input
		T0CLKO	O	Clock out of timer 0
		SS_4	I	Slave selection of SPI (it is output with regard to master)
		CCP0_2	I/O	Capture of external signal/High-speed Pulse output of PCA
		PWMFLT	I	Enhanced PWM fault detection
27		P3.6	I/O	Standard IO port
		INT2	I	External interrupt 2
		RxD_2	I	Serial input of UART1
		CMP-	I	Comparator negative input
		CCP1_2	I/O	Capture of external signal/High-speed Pulse output of PCA
28		P3.7	I/O	Standard IO port
		INT3	I	External interrupt 3
		TxD_2	O	Serial output of UART 1
		CMP+	I	Comparator positive input
		CCP2	I/O	Capture of external signal/High-speed Pulse output of PCA
		CCP2_2	I/O	Capture of external signal/High-speed Pulse output of PCA
29		P4.1	I/O	Standard IO port
		MISO_3	I/O	Master Iutput/Slave Onput of SPI
		CMPO_2	O	Comparator output
30		P4.2	I/O	Standard IO port
		WR	O	Write signal of external bus
31		P4.3	I/O	Standard IO port
		RxD_4	I	Serial input of UART1
		SCLK_3	I/O	Serial Clock of SPI
32		P4.4	I/O	Standard IO port
		RD	O	Read signal of external bus
		TxD_4	O	Serial output of UART 1
33		P2.0	I/O	Standard IO port
		PWM20	O	Enhanced PWM output
		A8	I	Address bus
34		P2.1	I/O	Standard IO port
		PWM21	O	Enhanced PWM output
		A9	I	Address bus
35		P2.2	I/O	Standard IO port
		PWM22	O	Enhanced PWM output
		A10	I	Address bus
		SS_2	I	Slave selection of SPI (it is output with regard to master)
36		P2.3	I/O	Standard IO port
		PWM23	O	Enhanced PWM output
		A11	I	Address bus
		MOSI_2	I/O	Master Output/Slave Input of SPI
		CCP0_2	I/O	Capture of external signal/High-speed Pulse output of PCA
37		P2.4	I/O	Standard IO port
		PWM24	O	Enhanced PWM output
		A12	I	Address bus
		ECI_3	I	External pulse input of PCA
		SDA_2	I/O	Serial data line of I2C
		MISO_2	I/O	Master Iutput/Slave Onput of SPI
38		P2.5	I/O	Standard IO port
		PWM25	O	Enhanced PWM output

		A13	I	Address bus
		CCP0_3	I/O	Capture of external signal/High-speed Pulse output of PCA
		SCL_2	I/O	Serial Clock line of I2C
		SCLK_2	I/O	Serial Clock of SPI
Pin number		**name**	**type**	**description**
LQFP48				
39		P2.6	I/O	Standard IO port
		PWM26	O	Enhanced PWM output
		A14	I	Address bus
		CCP1_3	I/O	Capture of external signal/High-speed Pulse output of PCA
40		P2.7	I/O	Standard IO port
		PWM27	O	Enhanced PWM output
		A15	I	Address bus
		CCP2_3	I/O	Capture of external signal/High-speed Pulse output of PCA
41		P4.5	I/O	Standard IO port
		ALE	O	Address Latch Enable signal
42		P4.6	I/O	Standard IO port
		RxD2_2	I	Serial input of UART2
43		P0.0	I/O	Standard IO port
		ADC8	I	ADC analog input 8
		AD0	I	Address/data bus
44		P0.1	I/O	Standard IO port
		ADC9	I	ADC analog input 9
		AD1	I	Address/data bus
45		P0.2	I/O	Standard IO port
		ADC10	I	ADC analog input 10
		AD2	I	Address/data bus
46		P0.3	I/O	Standard IO port
		ADC11	I	ADC analog input 11
		AD3	I	Address/data bus
47		P0.4	I/O	Standard IO port
		ADC12	I	ADC analog input 12
		AD4	I	Address/data bus
		T3	I	Timer3 external input
48		P5.2	I/O	Standard IO port

2.6 STC8G1K08T-36I-TSSOP20 touch key family

2.6.1 Features and Price

- **Selection and price (No external crystal and external reset required with 15 channels 10-bit ADC)**

MCU model	Operating voltage（V）	Flash Code Memory (100 thousand times) (Byte)	idata, Internal DATA RAM(Byte)	xdata, Internal extended SRAM (Byte)	Enhanced Dual DPTR increasing or decreasing	EEPROM 100 thousand times) (Byte)	Maximum I/O Lines	UARTs which may wake-up CPU	SPI	I²C	Touch key	LED driver	Timers/Counters (T0-T2 Pin Can wake-up CPU)	PCA/CCP/PWM (can be used as external interrupt and can wake-up CPU)	Power-down Wake-up timer	15-channels high speed ADC (3 PWMs can be used as 3 DACs)	Comparator (May be used as ADC to detect external power-down)	Internal LVD interrupt (can wake-up CPU)	Watch-dog Timer	Internal high reliable reset circuit with 4 level optional reset threshold voltage	Internal high presision Clock (adjustbal under 36MHz)	Clock output and Reset	Program encrypted transmission (Anti-blocking)	Password can be set for next update	Support RS485 download	Support software USBdownload directly	Online debugging	Package			2020 New product supply information
																												TSSOP20	QFN20 (3mm*3mm)	SOP16	
STC8G1K08T	1.9-5.5	8K	256	1K	2	4K	16	1	Y	Y	Y	Y	3	3	Y	10bit	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	√			Examples
STC8G1K17T	1.9-5.5	17K	256	1K	2	IAP	16	1	Y	Y	Y	Y	3	3	Y	10bie	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	-	√			

Note: The above unit prices are for orders of quantity of 10K and above. If the quantity is small, an additional RMB 0.1 per piece will be required. When the total amount of the order reaches or exceeds 3,000 yuan, it can be shipped free of charge, otherwise the customer will have to bear the freight. Retail sale starts at 10 pieces.

- **Core**
  - ✓ Ultra-high speed 8051 Core with single clock per machine cycle, which is called 1T and the speed is about 12 times faster than traditional 8051
  - ✓ Fully compatible instruction set with traditional 8051
  - ✓ 16 interrupt sources and 4 interrupt priority levels
  - ✓ Online debugging is supported
- **Operating voltage**
  - ✓ 1.9V～5.5V
  - ✓ Built-in LDO
- **Operating temperature**
  - ✓ -40℃~85℃
- **Flash memory**

- ✓ Up to 17Kbytes of Flash memory to be used to store user code
- ✓ Configurable size EEPROM, 512bytes single page erased, can be repeatedly erased more than 100 thousand times.
- ✓ In-System-Programming, ISP in short, can be used to update the application code, no need for special programmer.
- ✓ Online debugging with single chip is supported, and no special emulator is needed. The number of breakpoints is unlimited theoratically.

➢ **SRAM**

- ✓ 128 bytes internal direct access RAM (DATA)
- ✓ 128 bytes internal indirect access RAM (IDATA)
- ✓ 1024 bytes internal extended RAM (internal XDATA)

➢ **Clock**

- ✓ Internal high precise R/C clock (IRC, range from 4MHz to 36MHz), adjustable while ISP and can be divided to lower frequency by user software, 100KHz for instance.
  - ⌖ Error:±0.3% (at the temperature 25℃)
  - ⌖ -1.38%～+1.42% temperature drift (at the temperature range of -40℃ to +85℃)
  - ⌖ -0.88%～+1.05% temperature drift (at the temperature range of -20℃ to 65℃)
- ✓ Internal 32KHz low speed IRC with large error
- ✓ External 4MHz~33MHz oscillator or external clock

  The three clock sources above can be selected freely by used code.

➢ **Reset**

- ✓ Hardware reset
  - ⌖ Power-on reset. Measured voltage value is 1.69V~1.82V. (**Effective when the chip does not enable the low voltage reset function**)

    The power-on reset voltage is a voltage range consisting of an upper limit voltage and a lower limit voltage. When the operating voltage drops from 5V / 3.3V to the lower limit threshold voltage of the power-on reset, the chip is in a reset state; when the voltage rises from 0V to the upper threshold voltage of power-on reset, the chip is released from the reset state.
  - ⌖ Reset by reset pin. The default function of P5.4 is the I/O port. The P5.4 pin can be set as the reset pin while ISP download. (**Note: When the P5.4 pin is set as the reset pin, the reset level is low**.)
  - ⌖ Watch dog timer reset
  - ⌖ Low voltage detection reset. 4 low voltage detection levels are provided, 2.2V (Measured as 1.90V~2.04V), 2.4V (Measured as 2.30V~2.50V), V2.7 (Measured as 2.61V~2.82V), V3.0 (Measured as 2.90V~3.13V). Each level of low-voltage detection voltage is a voltage range consisting of an upper limit voltage and a lower limit voltage. When the operating voltage drops from 5V / 3.3V to the lower limit threshold voltage of low-voltage detection, the low-voltage detection takes effect. When the voltage rises from 0V to the upper threshold voltage, the low voltage detection becomes effective.
- ✓ Software reset
  - ⌖ Writing the reset trigger register using software

➢ **Interrupts**

- ✓ 16 interrupt sources: INT0(Supports rising edge and falling edge interrupt), INT1(Supports rising edge and falling edge interrupt), INT2(Supports falling edge interrupt only), INT3(Supports falling edge interrupt only), INT4(Supports falling edge interrupt only), timer0, timer1, timer2, UART1, ADC, LVD, SPI, I^2C, comparator, PCA/CCP/PWM, touch key

- ✓ 4 interrupt priority levels
- ✓ Interrupts that can awaken the CPU in clock stop mode: INT0 (P3.2), INT1 (P3.3), INT2 (P3.6), INT3 (P3.7), INT4 (P3.0), T0(P3.4), T1(P3.5), T2(P1.2), RXD(P3.0/P3.6/P1.6), RXD2(P1.0), CCP0(P1.1/P3.5), CCP1(P1.0/P3.6), CCP2(P3.7), I2C_SDA (P1.4/P3.3) and comparator interrupt, low-voltage detection interrupt, power-down wake-up timer.

➢ **Digital peripherals**

- ✓ 3 16-bit timers: timer0, timer1, timer2, where the mode 3 of timer0 has the Non Maskable Interrupt (NMI in short) function. Mode 0 of timer0 and timer1 is 16-bit Auto-reload mode.
- ✓ 1 high speed UART: UART1, whose baudrate clock source may be fast as FOSC/4
- ✓ 3 groups of 16-bit PCAs: CCP0, CCP1, CCP2, which can be used as capture, high speed output and 6-bits, 7-bits, 8-bits or 10-bits PWM.
- ✓ SPI: Master mode, slave mode or master/slave automatic switch mode are supported.
- ✓ I^2C: Master mode or slave mode are supported.

➢ **Analog peripherals**

- ✓ 15 channels (channel 0 to channel 14) ultra-high speed ADC which supports 10-bit precision analog-to-digital conversion.
- ✓ ADC channel 15 is used to test the internal reference voltage. (The default internal reference voltage is 1.19V when the chip is shipped)
- ✓ Comparator. A set of comparators (the positive terminal of the comparator can select the CMP+ and all ADC input ports, so the comparator can be used as a multi-channel comparator for time division multiplexing).
- ✓ Touch key: The microcontroller supports up to 16 touch keys. Each touch key can be independently enabled. The internal reference voltage is adjustable in 4 levels. Charge and discharge time settings and internal working frequency settings are flexible. The touch key supports wake-up from low-power mode.
- ✓ **LED driver: The microcontroller can drive up to 128 (8 * 8 * 2) LEDs, support common negative mode, common positive mode and common negative/common positive mode, and support 8 levels of gray adjustment (brightness adjustment).**
- ✓ DAC. 3 groups of PCAs can be used as DACs.

➢ **GPIO**

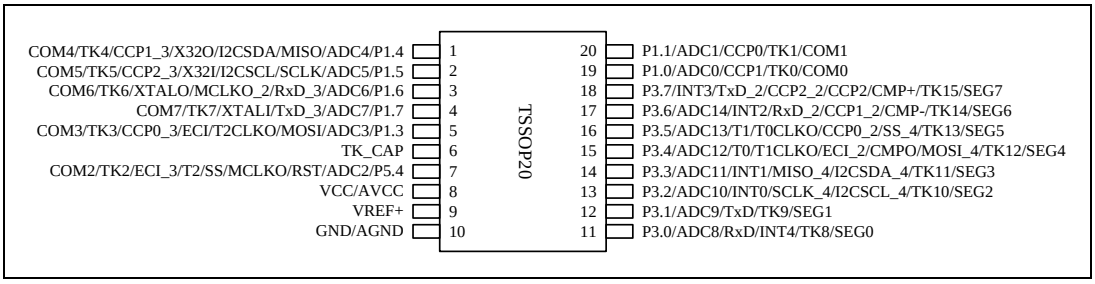
- ✓ Up to 16 GPIOs: P1.0~P1.1, P1.3~P1.7, P3.0~P3.7, P5.4
- ✓ 4 modes for all GPIOs: quasi_bidirectional mode, push-pull outputmode, open drain mode, high-impedance input mode
- ✓ **Except for P3.0 and P3.1, all other I/O ports are in a high-impedance state after power-on. User must set the I/O ports mode before using them. In addition, each I/O can independently enable the internal 4K pull-up resistor.**

➢ **Package**

- ✓ TSSOP20, QFN20 (3mm*3mm)

STC MCU

2.6.2 Pinouts



Note:

1. ADC's external reference power supply pin ADC_VRef+ must not be floating, it must be connected to an external reference power supply or directly connected to Vcc.
2. If USB download is not required, P3.0/P3.1/P3.2 cannot be at low level at the same time when the chip is reset.



universal USB to UART tool

ISP download steps:

1. **Connect the universal USB to UART tool to the target chip according to the connection method shown in the figure above.**
2. **Press the power button to confirm that the target chip is in a power-off state (the power-on indicator is off).**
   Note: When the tool is powered on for the first time, there is no external power supply, so if it is the first time to use this tool, you can skip this step.
3. **Click the "Download/Program" button in the STC-ISP download software.**
4. **Press the power button again to power on the target chip (the power-on indicator is on).**
5. **Start ISP download.**

Note: It has been found that when using the USB cable for ISP download, if the USB cable is too thin and the voltage drop on the USB cable is too large, this will result in insufficient power supply during the ISP download. Therefore, please be sure to use the booster USB cable for ISP download.

2.6.3 Pin descriptions

Pin number		name	type	description
TSSOP20	QFN20			
1		P1.4	I/O	Standard IO port
		ADC4	I	ADC analog input 4
		MISO	I/O	Master Iutput/Slave Onput of SPI
		SDA	I/O	Serial data line of I2C
		X32O	O	Connect to external oscillator
		CCP1_3	I/O	Capture of external signal/High-speed Pulse output of PCA
		TK4	I	Touch key
		COM4	O	LED dirver
2		P1.5	I/O	Standard IO port

		ADC5	I	ADC analog input 5
		SCLK	I/O	Serial Clock of SPI
		SCL	I/O	Serial Clock line of I2C
		X32I	I	Connect to external oscillator
		CCP2_3	I/O	Capture of external signal/High-speed Pulse output of PCA
		TK5	I	Touch key
		COM5	O	LED dirver
3		P1.6	I/O	Standard IO port
		ADC6	I	ADC analog input 6
		RxD_3	I	Serial input of UART1
		MCLKO_2	O	Master clock output
		XTALO	O	Connect to external oscillator
		TK6	I	Touch key
		COM6	O	LED dirver

Pin number		name	type	description
TSSOP20	QFN20			
4		P1.7	I/O	Standard IO port
		ADC7	I	ADC analog input 7
		TxD_3	O	Serial output of UART 1
		XTALI	I	Connect to external oscillator
		TK7	I	Touch key
		COM7	O	LED dirver
5		P1.3	I/O	Standard IO port
		ADC3	I	ADC analog input 3
		T2CLKO	O	Clock out of timer 2
		MOSI	I/O	Master Output/Slave Input of SPI
		ECI	I	External pulse input of PCA
		CCP0_3	I/O	Capture of external signal/High-speed Pulse output of PCA
		TK3	I	Touch key
		COM3	O	LED dirver
6		TK_CAP	I	External capacitor for touch key
7		P5.4	I/O	Standard IO port
		ADC2	I	ADC analog input 2
		RST	I	Reset pin
		MCLKO	O	Master clock output
		SS	I/O	SPI slave selection
		T2	I	Timer2 external input
		ECI_3	I	External pulse input of PCA
		TK2	I	Touch key
		COM2	O	LED dirver
8		VCC	VCC	Power Supply
		AVCC	VCC	Power Supply for ADC
9		VREF+	I	Reference voltage pin of ADC
10		GND	GND	Ground
		AGND	GND	ADC Ground
11		P3.0	I/O	Standard IO port
		RxD	I	Serial input of UART1
		ADC8	I	ADC analog input 8
		INT4	I	External interrupt 4
		TK8	I	Touch key
		SEG0	O	LED dirver
12		P3.1	I/O	Standard IO port
		TxD	O	Serial output of UART 1
		ADC9	I	ADC analog input 9
		TK9	I	Touch key

		SEG1	O	LED dirver
13		P3.2	I/O	Standard IO port
		INT0	I	External interrupt 0
		ADC10	I	ADC analog input 10
		SCLK_4	I/O	Serial Clock of SPI
		SCL_4	I/O	Serial Clock line of I2C
		TK10	I	Touch key
		SEG2	O	LED dirver
14		P3.3	I/O	Standard IO port
		INT1	I	External interrupt 1
		ADC11	I	ADC analog input 11
		MISO_4	I/O	Master Iutput/Slave Onput of SPI
		SDA_4	I/O	Serial data line of I2C
		TK11	I	Touch key
		SEG3	O	LED dirver
Pin number		**name**	**type**	**description**
TSSOP20	**QFN20**			
15		P3.4	I/O	Standard IO port
		T0	I	Timer0 external input
		T1CLKO	O	Clock out of timer 1
		ADC12	I	ADC analog input 12
		ECI_2	I	External pulse input of PCA
		CMPO	O	Comparator output
		MOSI_4	I/O	Master Output/Slave Input of SPI
		TK12	I	Touch key
		SEG4	O	LED dirver
16		P3.5	I/O	Standard IO port
		T1	I	Timer1 external input
		T0CLKO	O	Clock out of timer 0
		ADC13	I	ADC analog input 13
		CCP0_2	I/O	Capture of external signal/High-speed Pulse output of PCA
		SS_4	I	Slave selection of SPI (it is output with regard to master)
		TK13	I	Touch key
		SEG5	O	LED dirver
17		P3.6	I/O	Standard IO port
		INT2	I	External interrupt 2
		RxD_2	I	Serial input of UART1
		ADC14	I	ADC analog input 14
		CCP1_2	I/O	Capture of external signal/High-speed Pulse output of PCA
		CMP-	I	Comparator negative input
		TK14	I	Touch key
		SEG6	O	LED dirver
18		P3.7	I/O	Standard IO port
		INT3	I	External interrupt 3
		TxD_2	O	Serial output of UART 1
		CCP2	I/O	Capture of external signal/High-speed Pulse output of PCA
		CCP2_2	I/O	Capture of external signal/High-speed Pulse output of PCA
		CMP+	I	Comparator positive input
		TK15	I	Touch key
		SEG7	O	LED dirver
19		P1.0	I/O	Standard IO port
		ADC0	I	ADC analog input 0
		CCP1	I/O	Capture of external signal/High-speed Pulse output of PCA
		TK0	I	Touch key
		COM0	O	LED dirver
20		P1.1	I/O	Standard IO port
		ADC1	I	ADC analog input 1

		CCP0	I/O	Capture of external signal/High-speed Pulse output of PCA
		TK1	I	Touch key
		COM1	O	LED dirver

STC MCU

2.7 STC15H family (Traditional STC15 series to enhance the performance of special models)

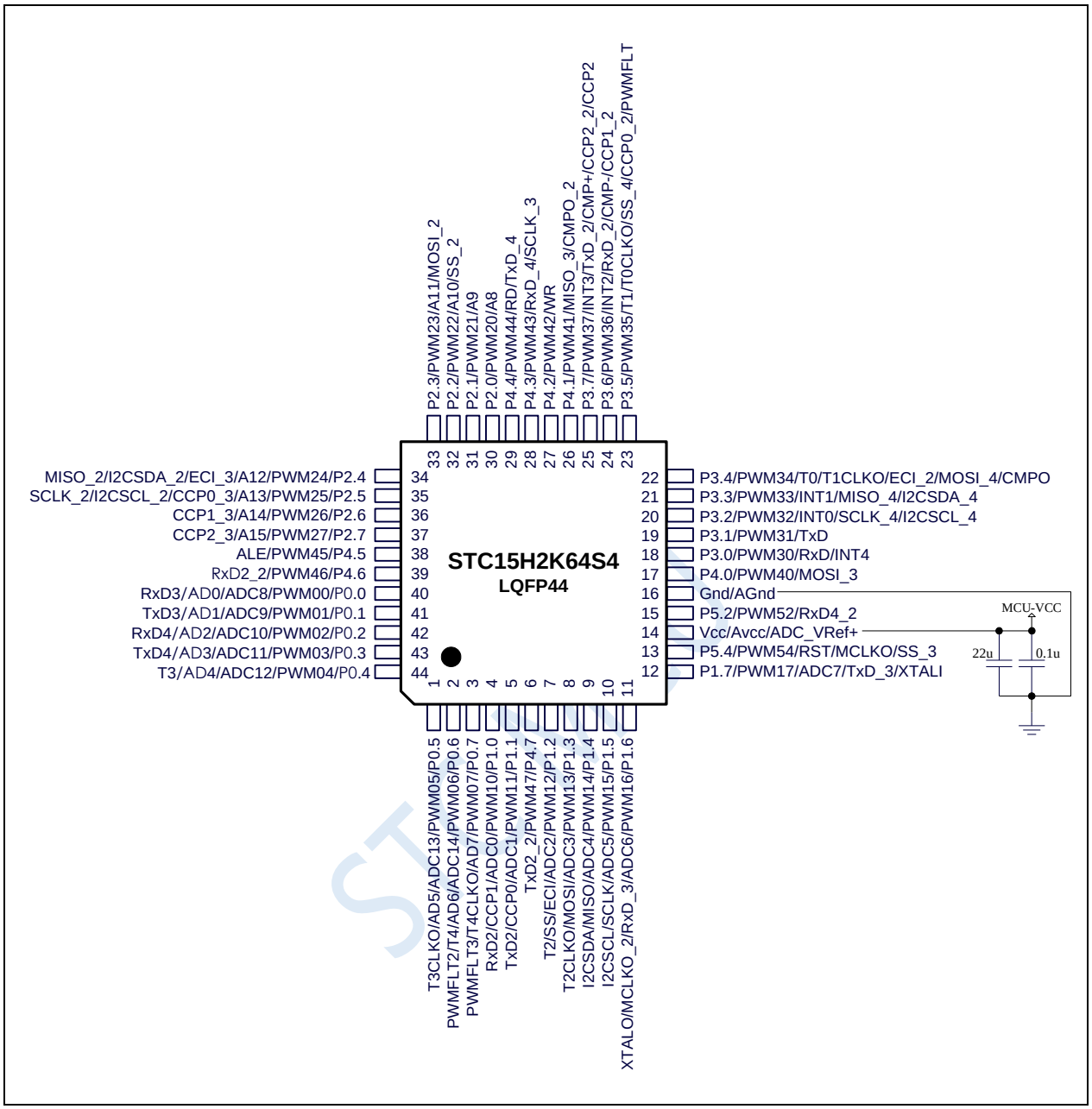
2.7.1 Features and Price

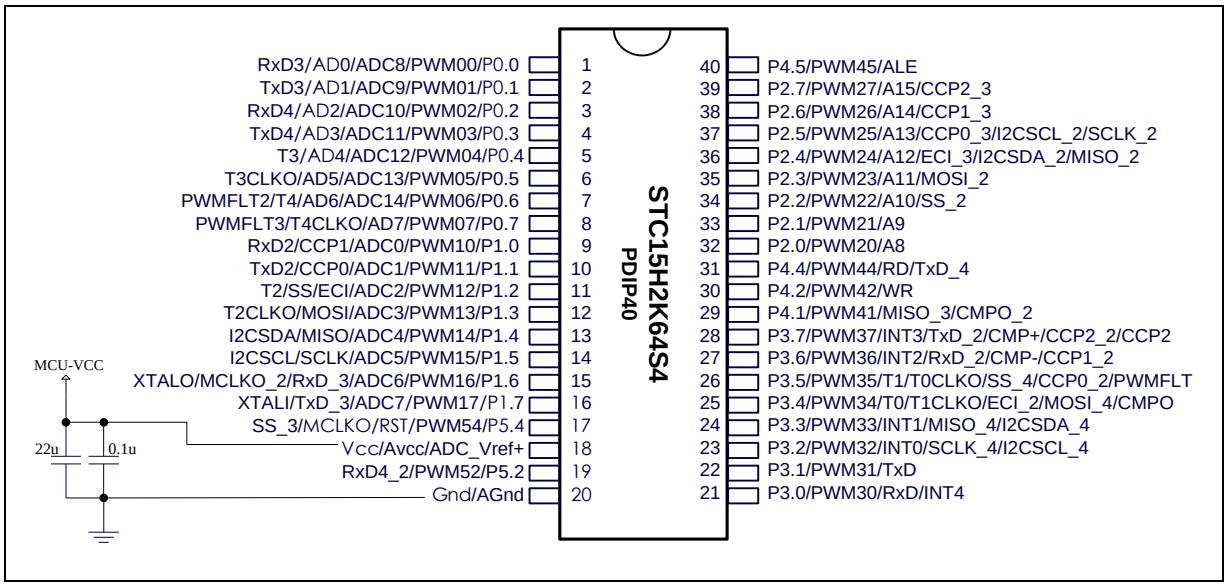
➢ **Selection and price**

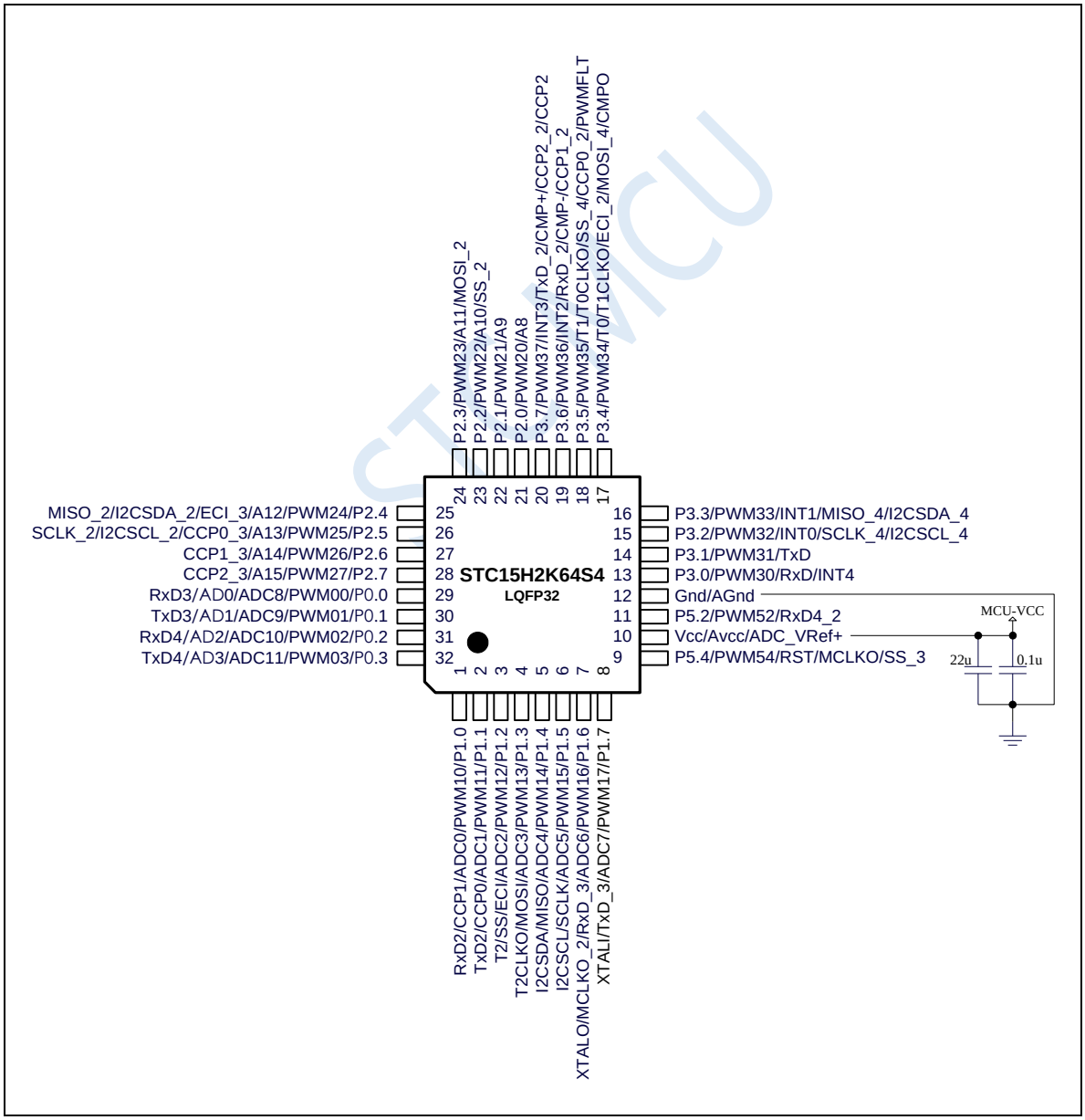
MCU model	Operating voltage（V）	Flash Code Memory (100 thousand times) (Byte)	idata, internal DATA RAM (Byte)	xdata, internal extended SRAM (Byte)	Enhanced Dual DPTR increasing or decreasing	EEPROM 100 thousand times) (Byte)	Maximum I/O Lines	UARTs which may wake-up CPU	MDU16 Hardware 16-bit Multiplier and Divider	SPI	I²C	Timers/Counters (T0-T4 Pin can wake-up CPU)	16-bit advanced PWM timer Complementary symmetrical dead-time	15-bit enhanced PWM (with dead-time control)	PCA/CCP/PWM (can be used as external interrupt and can wake-up CPU)	Power-down Wake-up timer	15-channels high speed ADC (All PWMs can be used as DACs)	Comparator (May be used as ADC to detect external power-down)	Internal LVD interrupt (can wake-up CPU)	Watch-dog Timer	Internal high reliable reset circuit with 4-level optional reset threshold voltage	Internal high presision Clock (adjustbal under 36MHz)	Clock output and Reset	Program encrypted transmission (Anti-blocking)	Password can be set for next update	Support RS485 download	Support software USBdownload directly	Online debugging	Package				Main product supply information
																													LQFP48	LQFP44	PDIP40	LQFP32	
STC15H2K48S4	1.9-5.5	48K	256	2K	2	16K	42	4	Y	Y	Y	5	-	42	3	Y	10bit	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		✓	✓	✓	Available
STC15H2K64S4	1.9-5.5	64K	256	2K	2	IAP	42	4	Y	Y	Y	5	-	42	3	Y	10bit	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		✓	✓	✓	

Note: Because the performance of the traditional STC15F2K series needs to be improved, and the current supply of wafers is tight, the STC8G2K64S4 series wafers are used to produce the above special models. If you need to purchase the above special models, please order in advance.

2.7.2 Pinouts


STC15H2K64S4
LQFP44
P2.3/PWM23/A11/MOSI_2
P2.2/PWM22/A10/SS_2
P2.1/PWM21/A9
P2.0/PWM20/A8
P4.4/PWM44/RD/TxD_4
P4.3/PWM43/RxD_4/SCLK_3
P4.2/PWM42/WR
P4.1/PWM41/MISO_3/CMPO_2
P3.7/PWM37/INT3/TxD_2/CMP+/CCP2_2/CCP2
P3.6/PWM36/INT2/RxD_2/CMP-/CCP1_2
P3.5/PWM35/T1/T0CLKO/SS_4/CCP0_2/PWMFLT
MISO_2/I2CSDA_2/ECI_3/A12/PWM24/P2.4
SCLK_2/I2CSCL_2/CCP0_3/A13/PWM25/P2.5
CCP1_3/A14/PWM26/P2.6
CCP2_3/A15/PWM27/P2.7
ALE/PWM45/P4.5
RxD2_2/PWM46/P4.6
RxD3/AD0/ADC8/PWM00/P0.0
TxD3/AD1/ADC9/PWM01/P0.1
RxD4/AD2/ADC10/PWM02/P0.2
TxD4/AD3/ADC11/PWM03/P0.3
T3/AD4/ADC12/PWM04/P0.4
P3.4/PWM34/T0/T1CLKO/ECI_2/MOSI_4/CMPO
P3.3/PWM33/INT1/MISO_4/I2CSDA_4
P3.2/PWM32/INT0/SCLK_4/I2CSCL_4
P3.1/PWM31/TxD
P3.0/PWM30/RxD/INT4
P4.0/PWM40/MOSI_3
Gnd/AGnd
P5.2/PWM52/RxD4_2
Vcc/Avcc/ADC_VRef+
P5.4/PWM54/RST/MCLKO/SS_3
P1.7/PWM17/ADC7/TxD_3/XTALI
MCU-VCC
22u
0.1u
T3CLKO/AD5/ADC13/PWM05/P0.5
PWMFLT2/T4/AD6/ADC14/PWM06/P0.6
PWMFLT3/T4CLKO/AD7/PWM07/P0.7
RxD2/CCP1/ADC0/PWM10/P1.0
TxD2/CCP0/ADC1/PWM11/P1.1
TxD2_2/PWM47/P4.7
T2/SS/ECI/ADC2/PWM12/P1.2
T2CLKO/MOSI/ADC3/PWM13/P1.3
I2CSDA/MISO/ADC4/PWM14/P1.4
I2CSCL/SCLK/ADC5/PWM15/P1.5
XTALO/MCLKO_2/RxD_3/ADC6/PWM16/P1.6





Note:

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

3 Function pins switch

Some special peripherals of STC8G series of microcontrollers can be switched among several I/O pins to realize one peripheral used as multiple device time-sharing, such as UART, SPI, PCA, I2C and bus control pins.

3.1 Register related to function pin switch

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

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
MCLKOCR	Master clock output control register	FE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000

3.1.1 Peripheral port switch register 1 (P_SW1), for UART1, CCP and SPI switch

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

S1_S[1:0]：UART1 pin selection bits

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

S1_S[1:0]：UART1 pin selection bits (**STC8G1K08-8Pin family, STC8G1K08A family**)

S1_S[1:0]	RxD	TxD
00	P3.0	P3.1
01	P3.2	P3.3
10	P5.4	P5.5
11	-	-

CCP_S[1:0]：PCA pin selection bits

CCP_S[1:0]	ECI	CCP0	CCP1	CCP2
00	P1.2	P1.1	P1.0	P3.7
01	P3.4	P3.5	P3.6	P3.7
10	P2.4	P2.5	P2.6	P2.7
11	-	-	-	-

CCP_S[1:0]：PCA pin selection bits (**STC8G1K08A family**)

CCP_S[1:0]	ECI	CCP0	CCP1	CCP2
00	P5.5	P3.2	P3.3	P5.4
01	P5.5	P3.1	P3.3	P5.4
10	P3.1	P3.2	P3.3	P5.5
11	-	-	-	-

CCP_S[1:0]：PCA pin selection bits (**STC8G1K08T family**)

CCP_S[1:0]	ECI	CCP0	CCP1	CCP2
00	P1.3	P1.1	P1.0	P3.7
01	P3.4	P3.5	P3.6	P3.7
10	P5.4	P1.3	P1.4	P1.5
11	-	-	-	-

SPI_S[1:0]：SPI pin selection bits

SPI_S[1:0]	SS	MOSI	MISO	SCLK
00	P1.2	P1.3	P1.4	P1.5

STC MCU

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

SPI_S[1:0]：SPI pin selection bits (**STC8G1K08-8Pin family, STC8G1K08A family**)

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

3.1.2 Peripheral port switch register 2 (P_SW2), for UART2/3/4, I2C and comparator switch

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

EAXFR: extended RAM area special function register (XFR) access control bit.

0: prohibit access to XFR

1: Enable access to XFR.

If you need to access XFR, you must first set EAXFR to 1, then XFR can be read and written normally.

I2C_S[1:0]：I^2C pin selection bits

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

I2C_S[1:0]：I^2C pin selection bits (**STC8G1K08-8Pin family, STC8G1K08A family)**

I2C_S[1:0]	SCL	SDA
00	P3.2	P3.3
01	P5.4	P5.5
10	-	-
11	-	-

CMPO_S：Comparator output pin selection bit

CMPO_S	CMPO
0	P3.4
1	P4.1

S4_S：UART4 pin selection bit

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

S3_S：UART3 pin selection bit

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

S2_S：UART2 pin selection bit

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

STC MCU

3.1.3 Clock selection register (MCLKOCR)

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

MCLKO_S：Main clock out pin selection bit

MCLKO_S	MCLKO
0	P5.4
1	P1.6

3.2 Example Routines

3.2.1 UART1 switch

C language code

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

#include "reg51.h"

sfr         P_SW1       =   0xa2;

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

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

    P_SW1 = 0x00;                   //RXD/P3.0, TXD/P3.1
//  P_SW1 = 0x40;                   //RXD_2/P3.6, TXD_2/P3.7
//  P_SW1 = 0x80;                   //RXD_3/P1.6, TXD_3/P1.7
//  P_SW1 = 0xc0;                   //RXD_4/P4.3, TXD_4/P4.4

    while (1);
}
```

STC MCU

Assembly code

```
; Operating frequency for test is 11.0592MHz

P_SW1       DATA    0A2H

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

            ORG     0000H
            LJMP    MAIN

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

            MOV     P_SW1,#00H          ;RXD/P3.0, TXD/P3.1
;           MOV     P_SW1,#40H          ;RXD_2/P3.6, TXD_2/P3.7
;           MOV     P_SW1,#80H          ;RXD_3/P1.6, TXD_3/P1.7
;           MOV     P_SW1,#0C0H         ;RXD_4/P4.3, TXD_4/P4.4

            SJMP    $

            END
```

3.2.2 UART2 switch

C language code

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

#include "reg51.h"

sfr     P_SW2   =   0xba;

sfr     P0M1    =   0x93;
```

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

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

    P_SW2 = 0x00;                   //RXD2/P1.0, TXD2/P1.1
//  P_SW2 = 0x01;                   //RXD2_2/P4.6, TXD2_2/P4.7

    while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

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

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
```

```
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#00H          ;RXD2/P1.0, TXD2/P1.1
;       MOV     P_SW2,#01H          ;RXD2_2/P4.0, TXD2_2/P4.2

        SJMP    $

        END
```

3.2.3 UART3 switch

C language code

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

#include "reg51.h"

sfr     P_SW2       =   0xba;

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

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

STC MCU

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

    while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

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

            ORG     0000H
            LJMP    MAIN

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

            MOV     P_SW2,#00H      ;RXD3/P0.0, TXD3/P0.1
;           MOV     P_SW2,#02H      ;RXD3_2/P5.0, TXD3_2/P5.1

            SJMP    $

            END
```

3.2.4 UART4 switch

C language code

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

#include "reg51.h"

sfr     P_SW2     =   0xba;

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

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

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

    while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

P_SW2     DATA    0BAH

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

```
                ORG     0000H
                LJMP    MAIN

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

                MOV     P_SW2,#00H          ;RXD4/P0.2, TXD4/P0.3
;               MOV     P_SW2,#04H          ;RXD4_2/P5.2, TXD4_2/P5.3

                SJMP    $

                END
```

3.2.5 SPI switch

C language code

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

#include "reg51.h"

sfr     P_SW1   =   0xa2;

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

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

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

    P_SW1 = 0x00;                   //SS/P1.2, MOSI/P1.3, MISO/P1.4, SCLK/P1.5
//  P_SW1 = 0x04;                   //SS_2/P2.2, MOSI_2/P2.3, MISO_2/P2.4, SCLK_2/P2.5
//  P_SW1 = 0x08;                   //SS_3/P5.4, MOSI_3/P4.0, MISO_3/P4.1, SCLK_3/P4.3
//  P_SW1 = 0x0c;                   //SS_4/P3.5, MOSI_4/P3.4, MISO_4/P3.3, SCLK_4/P3.2

    while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

P_SW1       DATA    0A2H

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

            ORG     0000H
            LJMP    MAIN

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

            MOV     P_SW1,#00H      ;SS/P1.2, MOSI/P1.3, MISO/P1.4, SCLK/P1.5
;           MOV     P_SW1,#04H      ;SS_2/P2.2, MOSI_2/P2.3, MISO_2/P2.4, SCLK_2/P2.5
;           MOV     P_SW1,#08H      ;SS_3/P5.4, MOSI_3/P4.0, MISO_3/P4.1, SCLK_3/P4.3
;           MOV     P_SW1,#0CH      ;SS_4/P3.5, MOSI_4/P3.4, MISO_4/P3.3, SCLK_4/P3.2
```

STC MCU

```
        SJMP    $

        END
```

3.2.6 PCA/CCP/PWM switch

C language code

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

#include "reg51.h"

sfr     P_SW1   =   0xa2;

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

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

    P_SW1 = 0x00;                   //ECI/P1.2, CCP0/P1.1, CCP1/P1.0, CCP2/P3.7
//  P_SW1 = 0x10;                   //ECI_2/P3.4, CCP0_2/P3.5, CCP1_2/P3.6, CCP2_2/P3.7
//  P_SW1 = 0x20;                   //ECI_3/P2.4, CCP0_3/P2.5, CCP1_3/P2.6, CCP2_3/P2.7

    while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

P_SW1       DATA    0A2H

P0M1        DATA    093H
```

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

            ORG     0000H
            LJMP    MAIN

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

            MOV     P_SW1,#00H      ;ECI/P1.2, CCP0/P1.1, CCP1/P1.0, CCP2/P3.7
;           MOV     P_SW1,#10H      ;ECI_2/P3.4, CCP0_2/P3.5, CCP1_2/P3.6, CCP2_2/P3.7
;           MOV     P_SW1,#20H      ;ECI_3/P2.4, CCP0_3/P2.5, CCP1_3/P2.6, CCP2_3/P2.7

            SJMP    $

            END
```

3.2.7 I2C switch

C language code

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

#include "reg51.h"

sfr     P_SW2   =   0xba;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
```

```
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

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

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

        while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

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

            ORG     0000H
            LJMP    MAIN

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

```
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#00H          ;SCL/P1.5, SDA/P1.4
;       MOV     P_SW2,#10H          ;SCL_2/P2.5, SDA_2/P2.4
;       MOV     P_SW2,#20H          ;SCL_3/P7.7, SDA_3/P7.6
;       MOV     P_SW2,#30H          ;SCL_4/P3.2, SDA_4/P3.3

        SJMP    $
        END
```

3.2.8 Comparator output switch

C language code

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

#include "reg51.h"

sfr     P_SW2   =   0xba;

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

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

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

STC MCU

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

Assembly code

```
; Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

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

            ORG     0000H
            LJMP    MAIN

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

            MOV     P_SW2,#00H          ;CMPO/P3.4
;           MOV     P_SW2,#08H          ;CMPO_2/P4.1

            SJMP    $

            END
```

3.2.9 Main clock output switch

C language code

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

#include "reg51.h"
```

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

sfr       P_SW2     =   0xba;

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

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

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

    while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

P_SW2     DATA    0BAH

CLKOCR    EQU     0FE05H

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

```
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

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

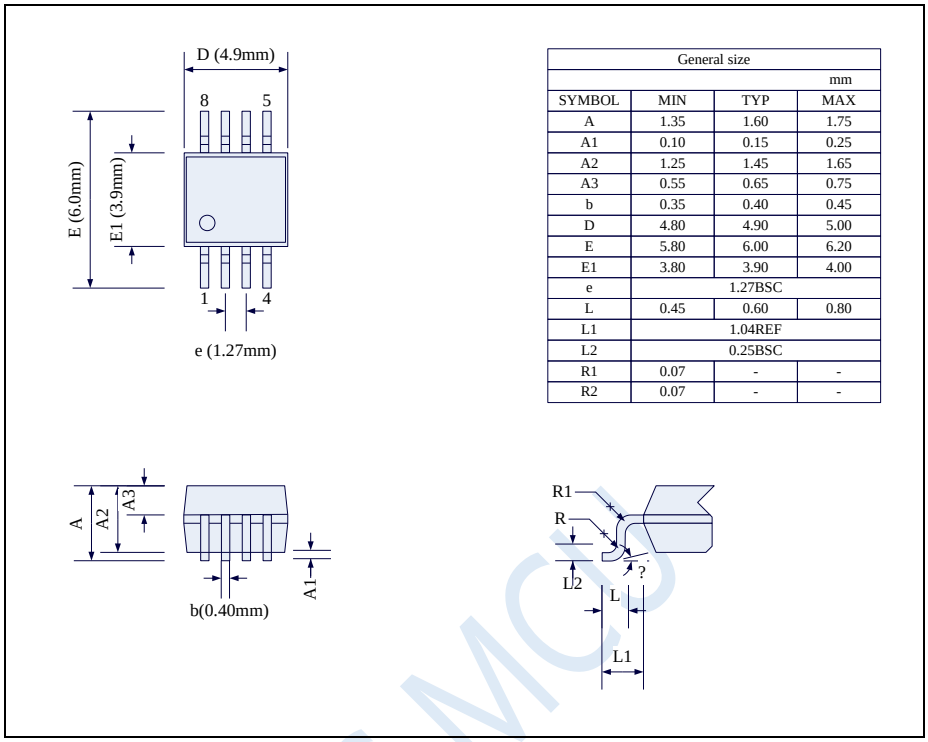
            MOV     P_SW2,#80H
            MOV     A,#04H          ;HIRC/4 output via MCLKO/P5.4
;           MOV     A,#84H          ;HIRC/4 output via MCLKO_2/P1.6
            MOV     DPTR,#CLKOCR
            MOVX    @DPTR,A
            MOV     P_SW2,#00H

            SJMP    $
            END
```

STC MCU

4 Package Dimensions

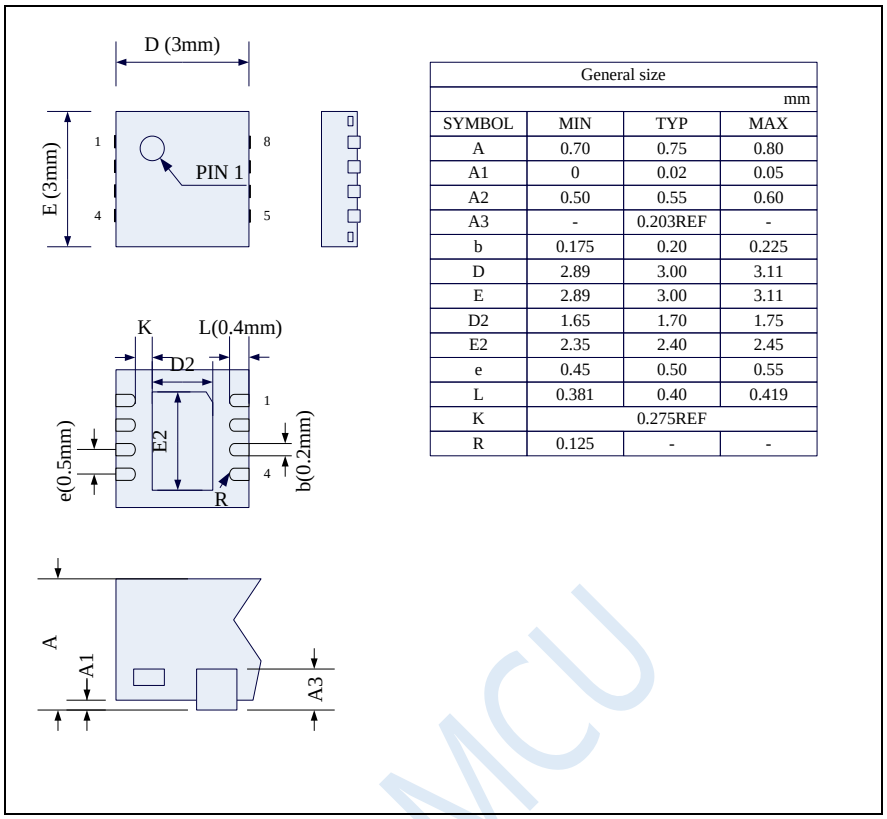
4.1 SOP8 Package mechanical data



General size			
			mm
SYMBOL	MIN	TYP	MAX
A	1.35	1.60	1.75
A1	0.10	0.15	0.25
A2	1.25	1.45	1.65
A3	0.55	0.65	0.75
b	0.35	0.40	0.45
D	4.80	4.90	5.00
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e		1.27BSC	
L	0.45	0.60	0.80
L1		1.04REF	
L2		0.25BSC	
R1	0.07	-	-
R2	0.07	-	-

STCMCU

4.2 DFN8 Package mechanical data (3mm*3mm)

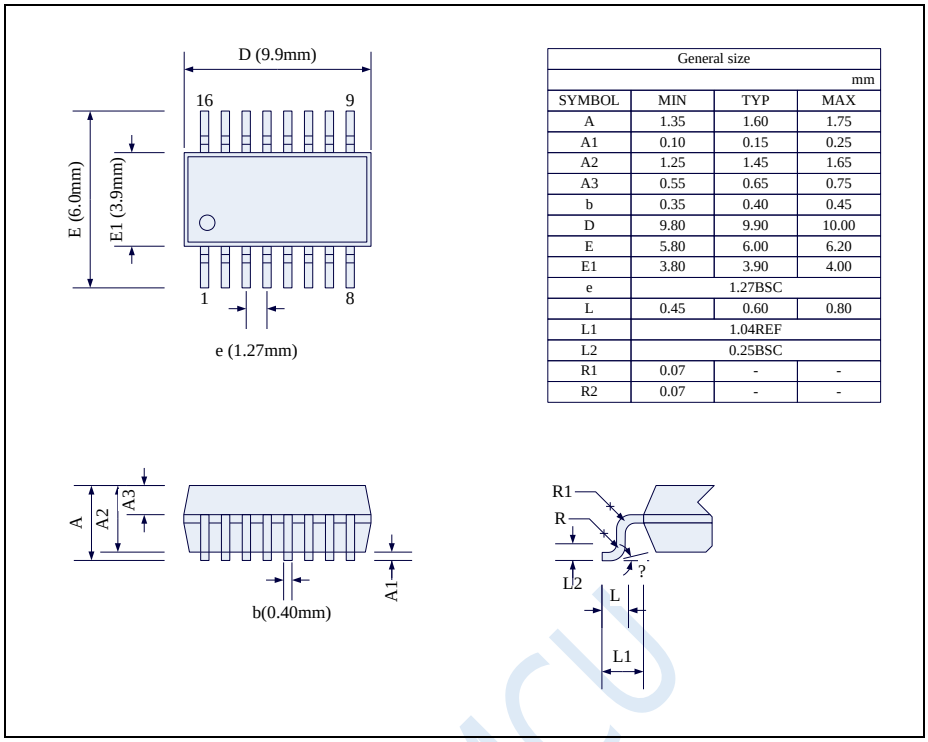


General size			
			mm
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0	0.02	0.05
A2	0.50	0.55	0.60
A3	-	0.203REF	-
b	0.175	0.20	0.225
D	2.89	3.00	3.11
E	2.89	3.00	3.11
D2	1.65	1.70	1.75
E2	2.35	2.40	2.45
e	0.45	0.50	0.55
L	0.381	0.40	0.419
K		0.275REF	
R	0.125	-	-

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

STCMCU

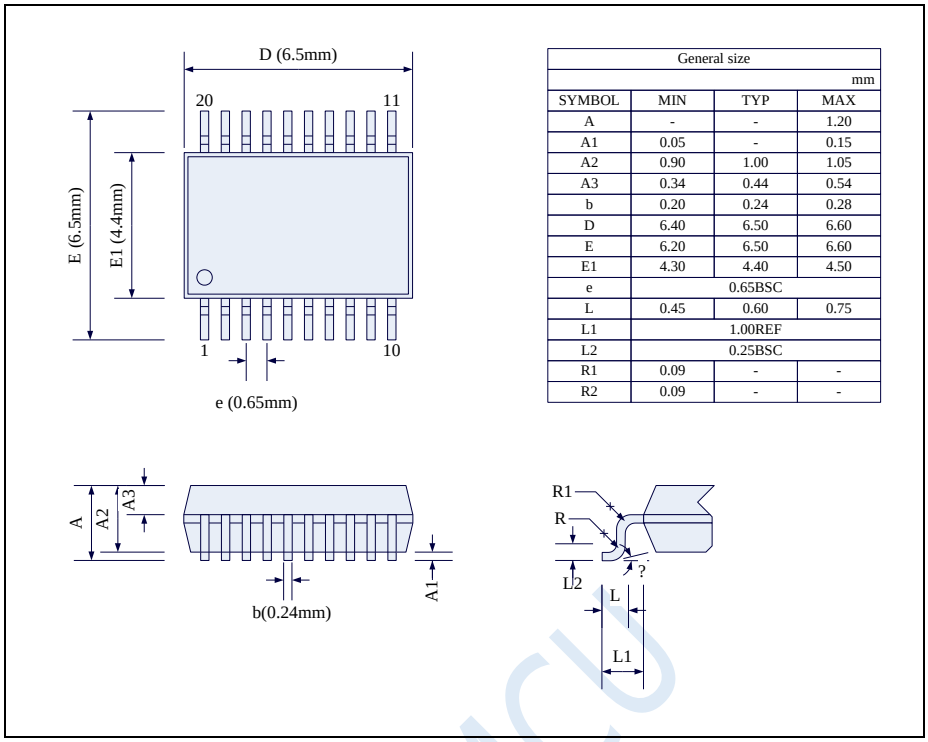
4.3 SOP16 Package mechanical data



General size			
			mm
SYMBOL	MIN	TYP	MAX
A	1.35	1.60	1.75
A1	0.10	0.15	0.25
A2	1.25	1.45	1.65
A3	0.55	0.65	0.75
b	0.35	0.40	0.45
D	9.80	9.90	10.00
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e		1.27BSC	
L	0.45	0.60	0.80
L1		1.04REF	
L2		0.25BSC	
R1	0.07	-	-
R2	0.07	-	-

STC MCU

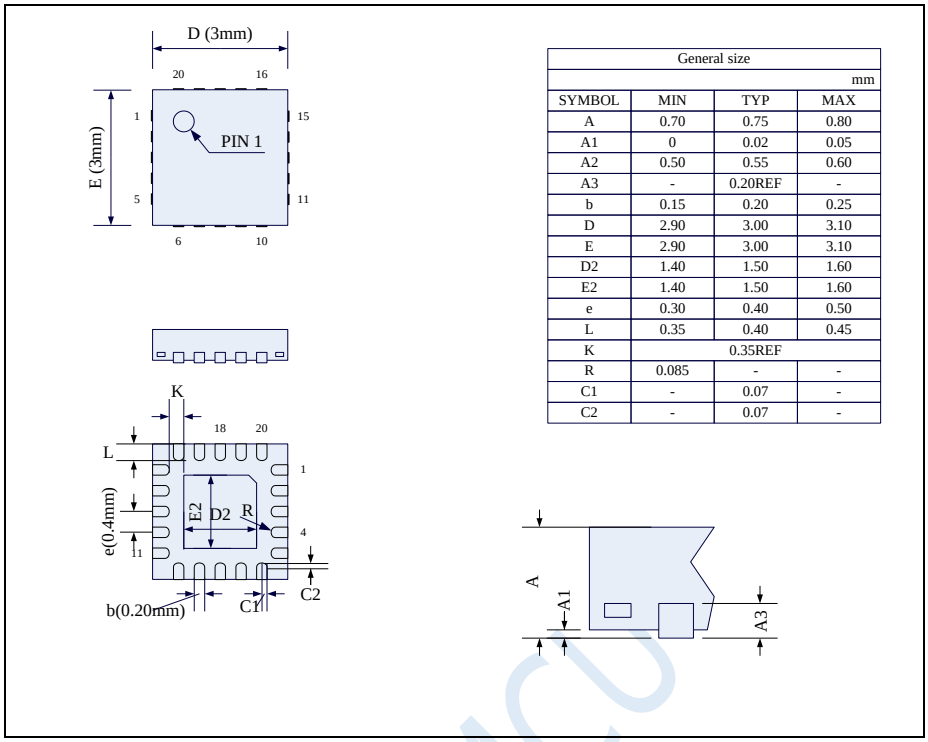
4.4 TSSOP20 Package mechanical data



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

STC MCU

4.5 QFN20 Package mechanical data (3mm*3mm)

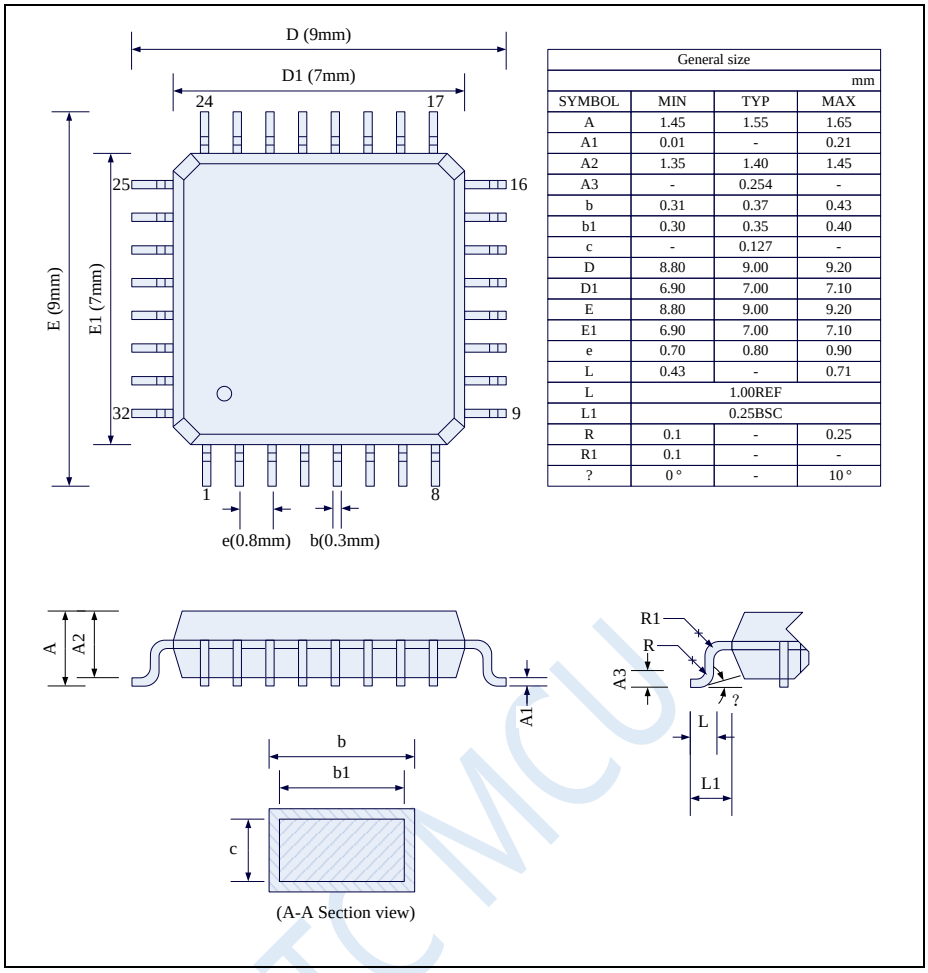


General size			
			mm
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0	0.02	0.05
A2	0.50	0.55	0.60
A3	-	0.20REF	-
b	0.15	0.20	0.25
D	2.90	3.00	3.10
E	2.90	3.00	3.10
D2	1.40	1.50	1.60
E2	1.40	1.50	1.60
e	0.30	0.40	0.50
L	0.35	0.40	0.45
K		0.35REF	
R	0.085	-	-
C1	-	0.07	-
C2	-	0.07	-

The back metal sheet (substrate) of STC's existing QFN20 packaged chip is not grounded inside the chip. It can be grounded or not grounded on the user's PCB board, which will not affect the performance of the chip.

STC MCU

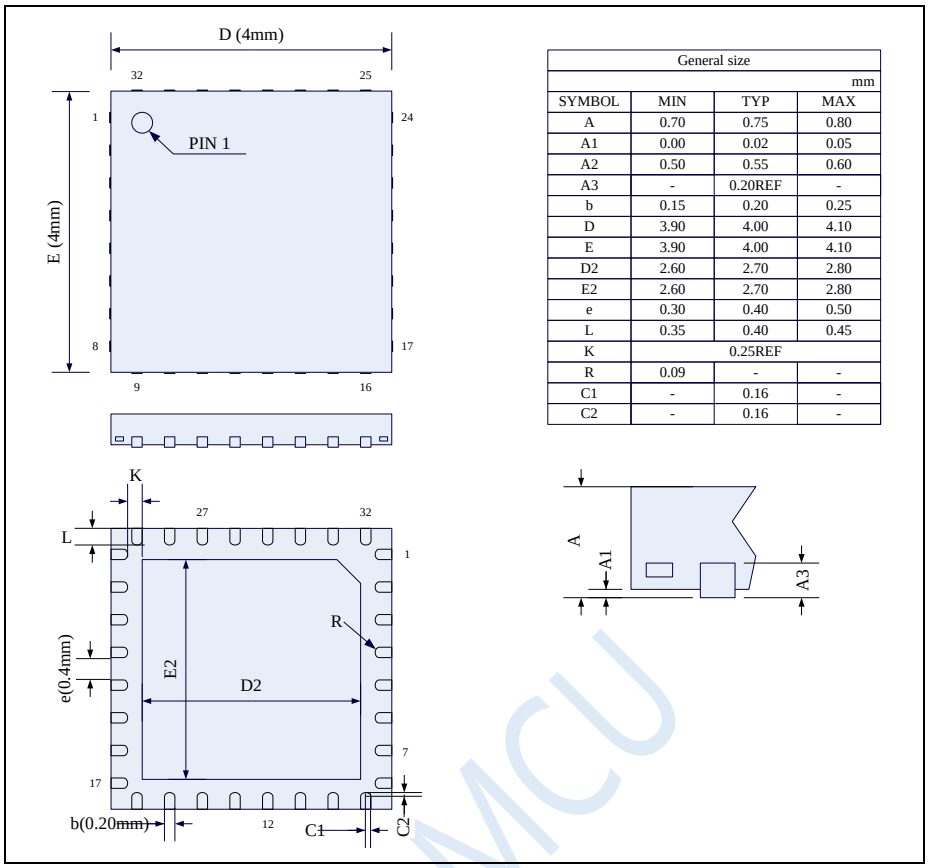
4.6 LQFP32 Package mechanical data (9mm*9mm)



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

STC MCU

4.7 QFN32 Package mechanical data (4mm*4mm)

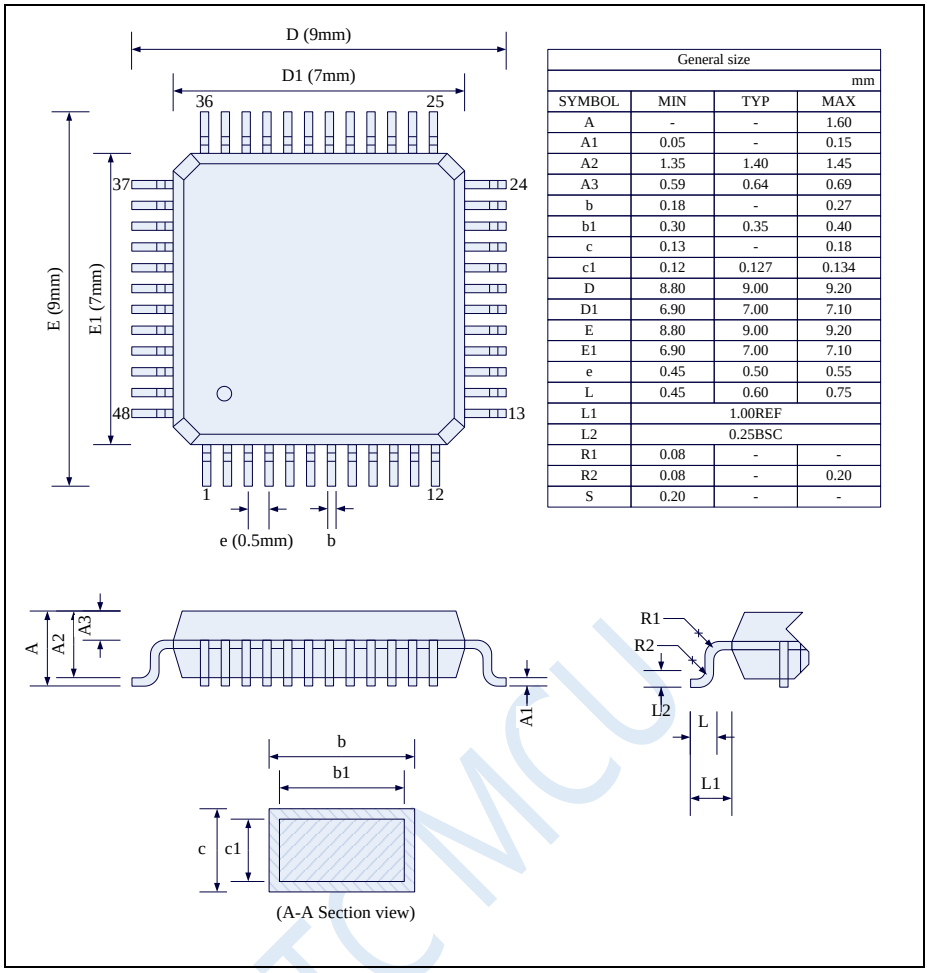


General size			
			mm
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0.00	0.02	0.05
A2	0.50	0.55	0.60
A3	-	0.20REF	-
b	0.15	0.20	0.25
D	3.90	4.00	4.10
E	3.90	4.00	4.10
D2	2.60	2.70	2.80
E2	2.60	2.70	2.80
e	0.30	0.40	0.50
L	0.35	0.40	0.45
K		0.25REF	
R	0.09	-	-
C1	-	0.16	-
C2	-	0.16	-

The back metal sheet (substrate) of STC's existing QFN32 packaged chip is not grounded inside the chip. It can be grounded or not grounded on the user's PCB board, which will not affect the performance of the chip.

STCMCU

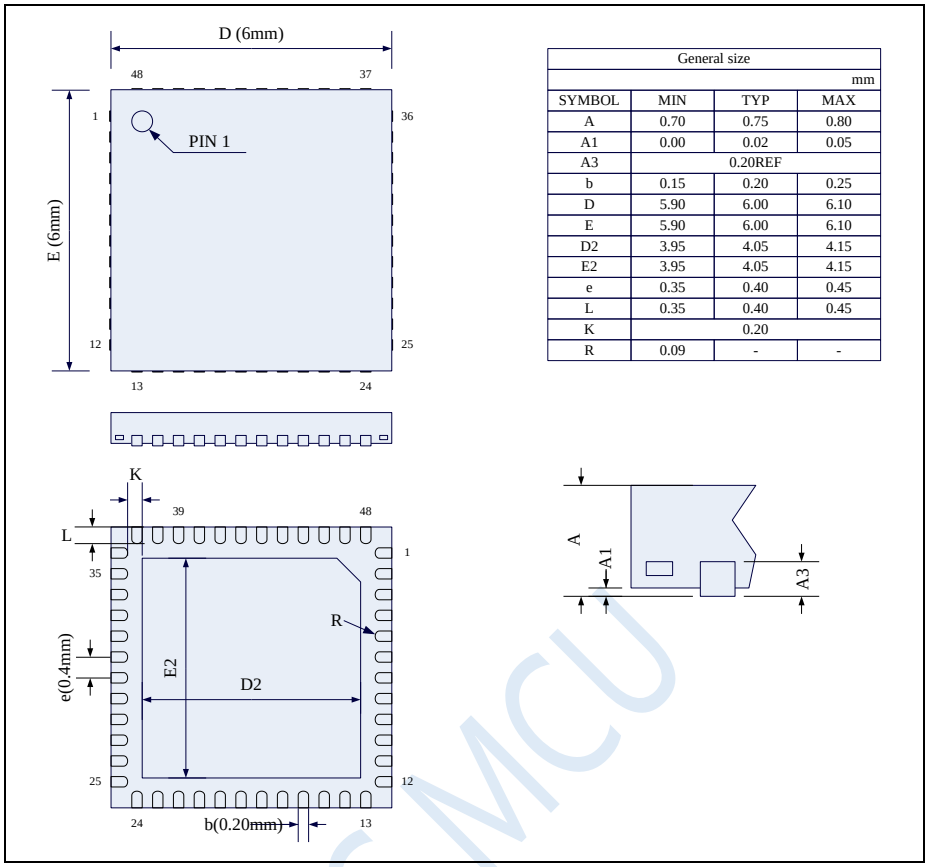
4.8 LQFP48 Package mechanical data (9mm*9mm)



General size			
			mm
SYMBOL	MIN	TYP	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.18	-	0.27
b1	0.30	0.35	0.40
c	0.13	-	0.18
c1	0.12	0.127	0.134
D	8.80	9.00	9.20
D1	6.90	7.00	7.10
E	8.80	9.00	9.20
E1	6.90	7.00	7.10
e	0.45	0.50	0.55
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-

STC MCU

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

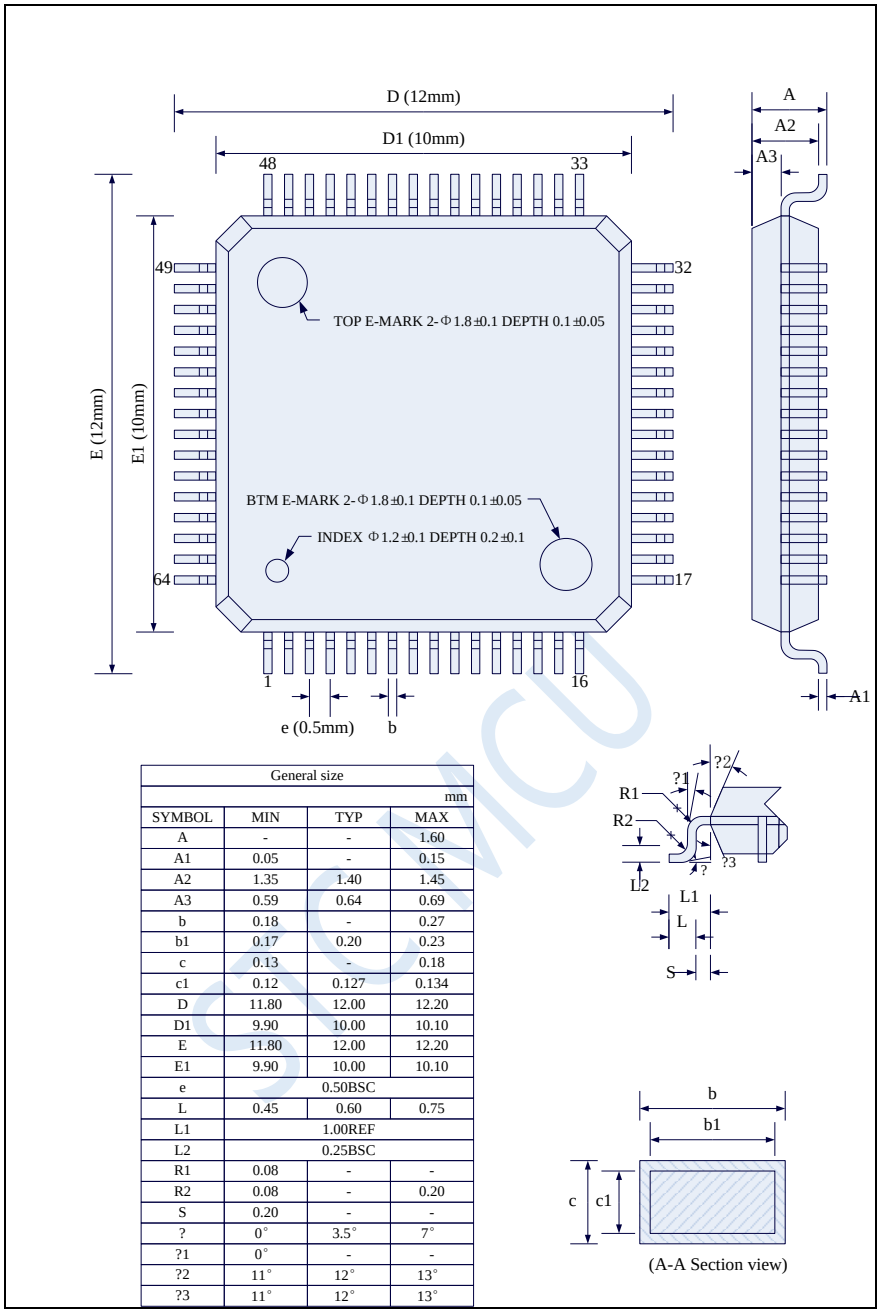


General size			
			mm
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0.00	0.02	0.05
A3		0.20REF	
b	0.15	0.20	0.25
D	5.90	6.00	6.10
E	5.90	6.00	6.10
D2	3.95	4.05	4.15
E2	3.95	4.05	4.15
e	0.35	0.40	0.45
L	0.35	0.40	0.45
K		0.20	
R	0.09	-	-

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

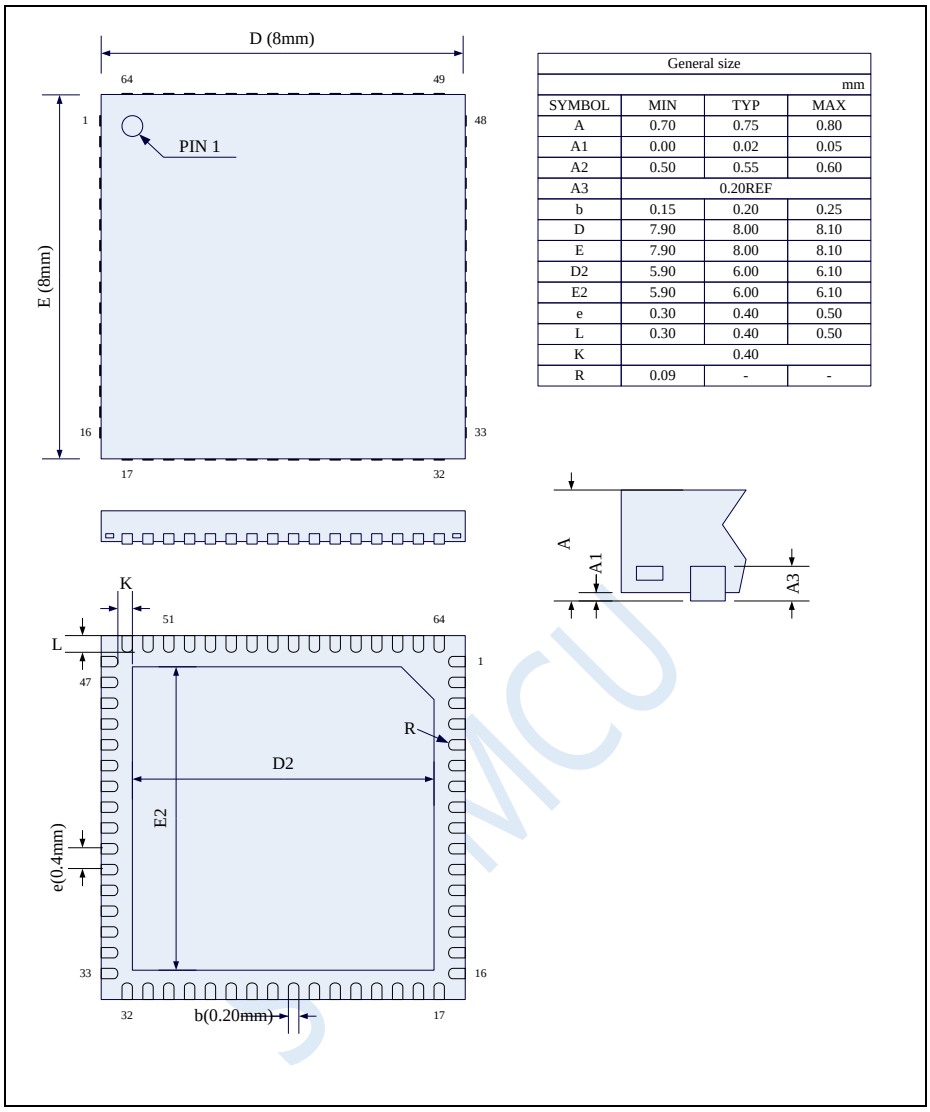
STC MCU

4.10 LQFP64S Package mechanical data (12mm*12mm)



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

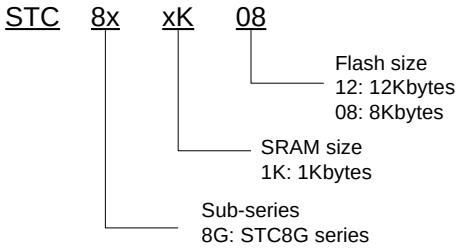
4.11 QFN64 Package mechanical data (8mm*8mm)



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

The back metal sheet (substrate) of STC's existing QFN64 packaged chip is not grounded inside the chip. It can be grounded or not grounded on the user's PCB board, which will not affect the performance of the chip.

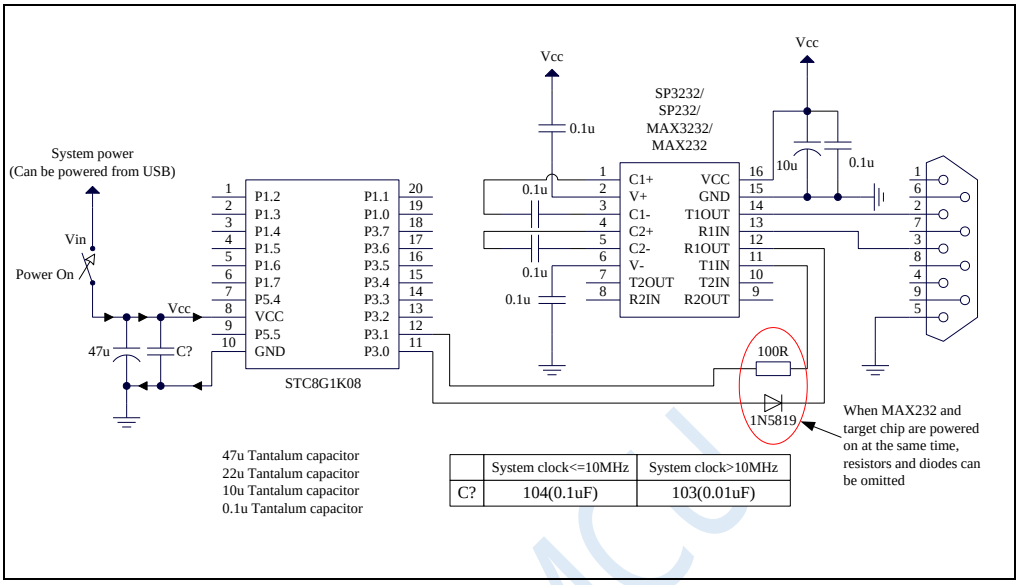
4.12 Naming rules of STC8G family



5 ISP Download and typical application circuit

5.1 STC8G seriesISP download application circuit

5.1.1 Download using RS-232 converter (no independent VREF pin)

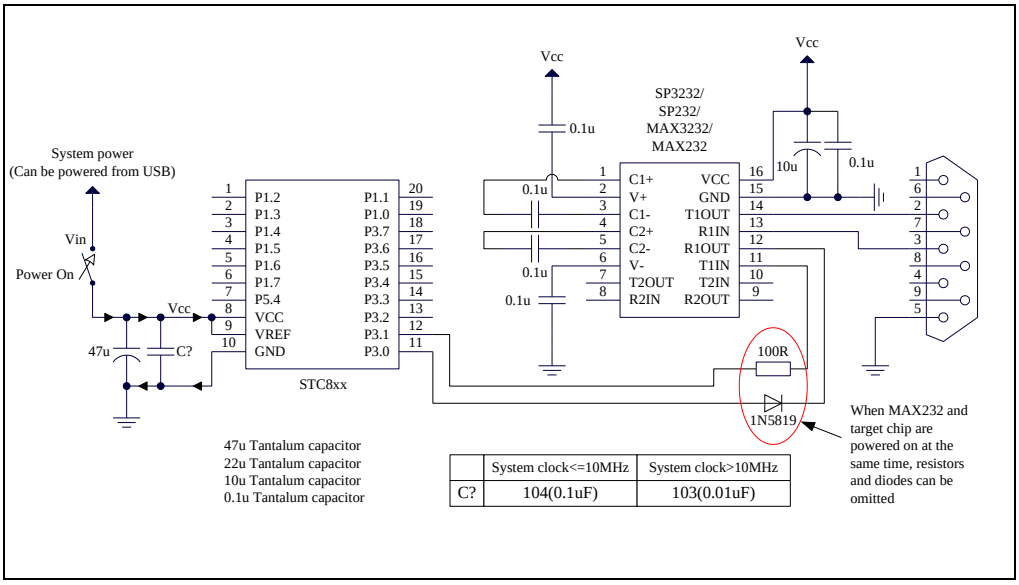


	System clock<=10MHz	System clock>10MHz
C?	104(0.1uF)	103(0.01uF)

ISP download steps:

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

5.1.2 Download using RS-232 converter (independent VREF pin, general accuracy ADC), supporting emulation

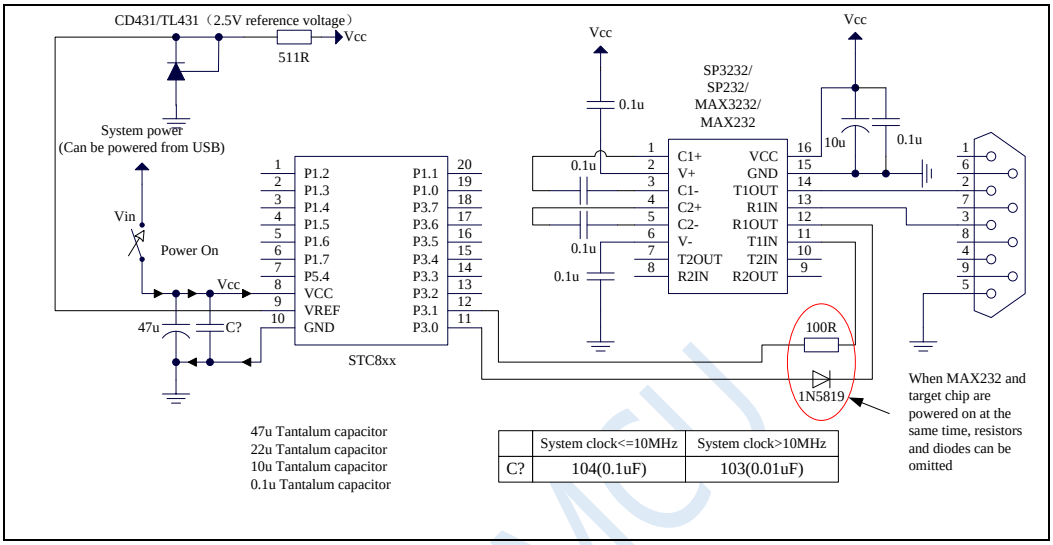


	System clock<=10MHz	System clock>10MHz
C?	104(0.1uF)	103(0.01uF)

ISP download steps:

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

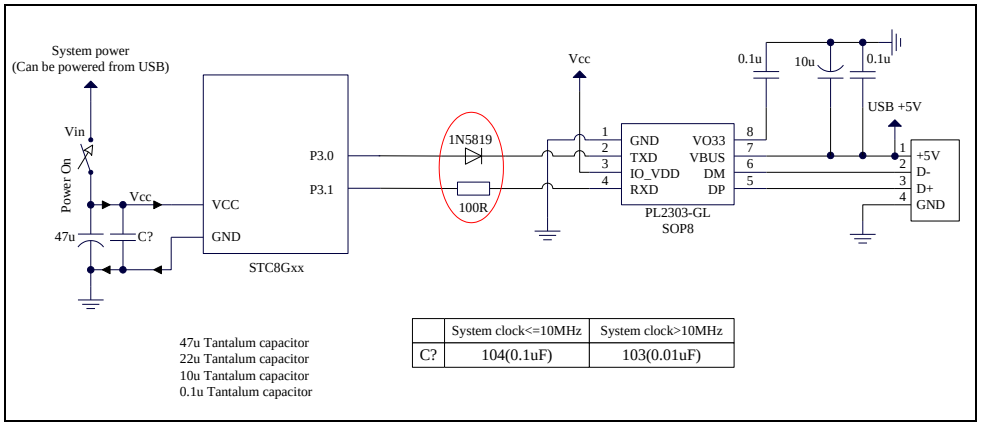
5.1.3 Download using RS-232 converter (independent VREF pin, high accuracy ADC), supporting emulation



ISP download steps:

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

5.1.4 Download using PL2303-GL, supporting emulation



ISP download steps:

1. Power off the target chip while keep the USB-to-UART chip power on.

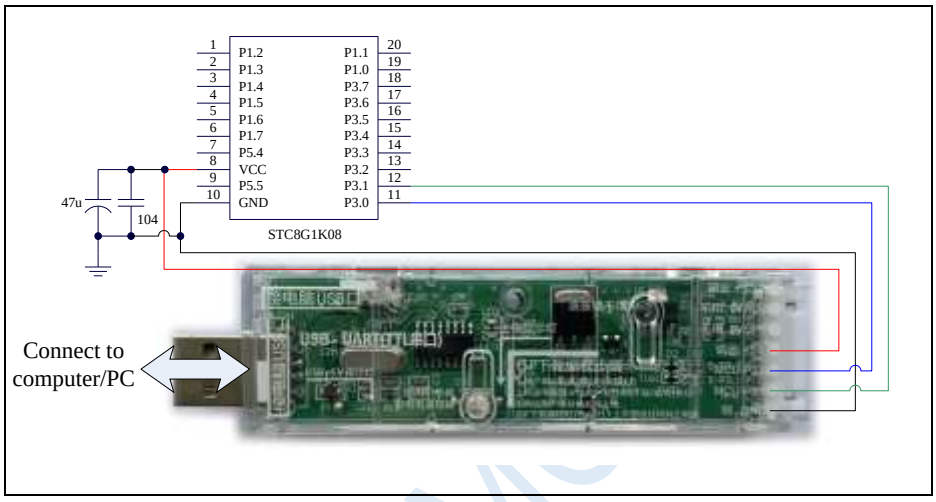
Note: Part of the PL2303-SA's baut rate has large error, PL2303-GL is recommended.

2. Since the sending pin of the USB-to-UART chip is generally a strong push-pull output, a diode must be connected in series between P3.0 of the target chip and the sending pin of the USB-to-UART chip, otherwise the target chip cannot be completely powered off.

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

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

5.1.5 Use general USB to UART tool to download, support ISP online download, and also support emulation

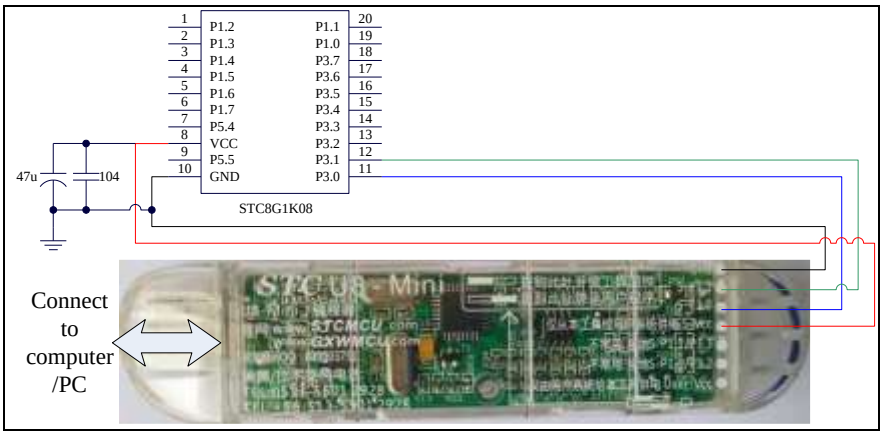


ISP download steps:

1. Connect the universal USB to UART 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 it is the first time to use this tool, you can skip this step.**
3. Click the "Download/Program" button in the STC-ISP download software.
4. Press the power button again to power on the target chip (the power-on indicator is on).
5. Start ISP download.

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

5.1.6 Use U8-Mini tool to download, support ISP online and offline download, and also support emulation



ISP download steps:

1. Connect the U8-Mini 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 ISP download.

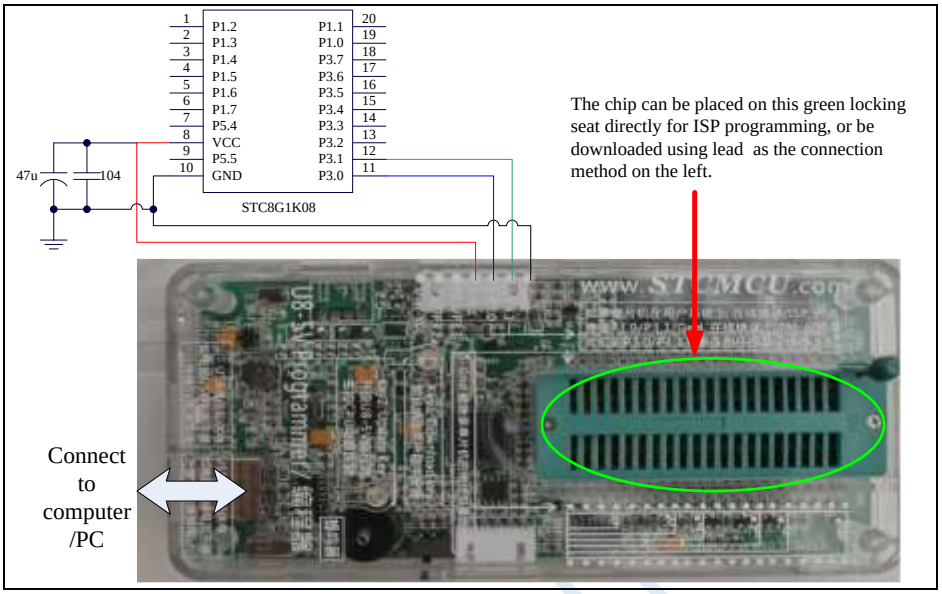
Note: If U8-Mini is used to power the target system, the total current of the target system cannot be greater than 200mA, otherwise the download will fail.

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

To use U8-Mini for emulation, you must set U8-Mini to pass-through mode firstly. The method for U8W/U8W-Mini to realize the USB to serial port pass-through mode is as follows:

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

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



ISP download steps (connection mode):

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 ISP download.

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

ISP download steps (on-board mode):

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

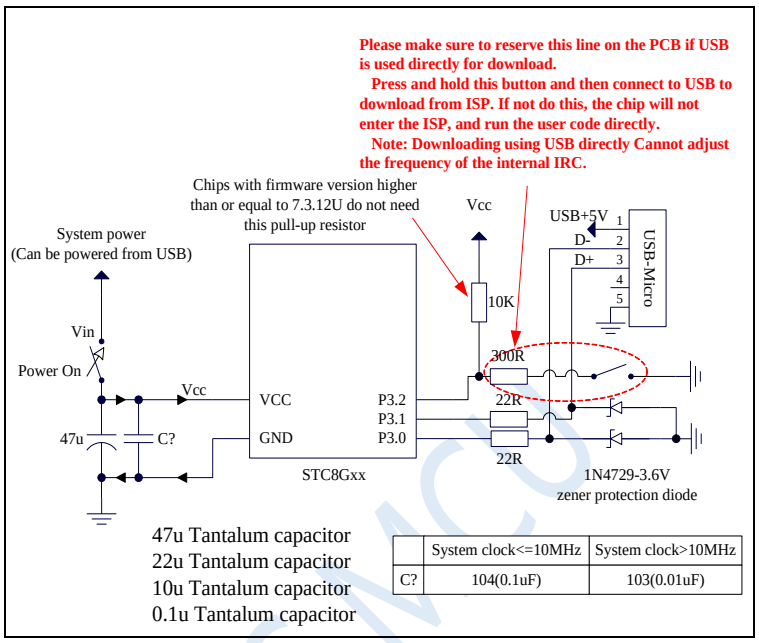
Note: At present, it has been found that when using the USB cable for ISP download, the USB cable is too thin and the voltage drop on the USB cable is too large, resulting in insufficient power supply during the ISP download. Therefore, please make sure to use the booster USB cable for ISP download.

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

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

5.1.8 ISP Download using Simulate USB directly, only supports ISP download, does not support emulation

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



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

ISP download steps:

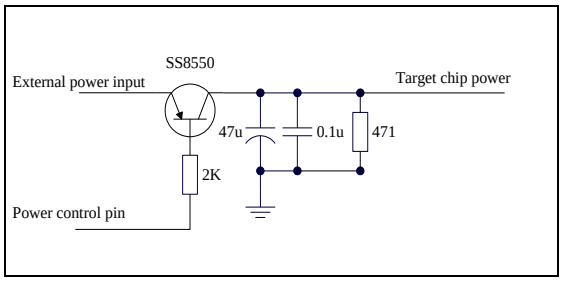
1. Power off the target chip.
2. Connect P3.0/P3.1 to the USB port according to the connection method shown in the figure.
3. Connect P3.2 to GND.
4. Power on the target chip, and wait for the "STC USB Writer (USB1)" to be automatically recognized in the STC-ISP download software.
5. Click the "Download/Program"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

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

Note: The USB download supported by the STC8G1K08 series is the USB communication simulated by the I/O port software, which is inevitably affected by various software and hardware factors, especially the influence of the different software and hardware versions of the computer. A certain percentage of chips cannot be downloaded via USB (about 0.2% of the actual test cannot be downloaded via USB). It is recommended to use ordinary serial port download or USB to serial port download for mass production.

For detailed USB download, please refer to the appendix of this document.

5.1.9 Power supply control reference circuit of MCU



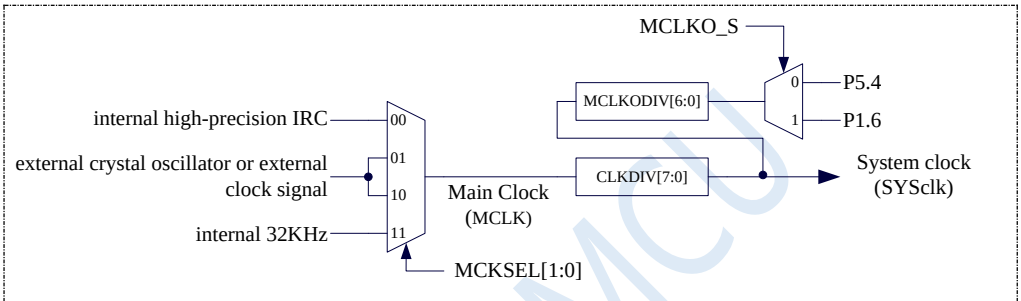
STC MCU

6 Clock, Reset, Power Saving Mode and Power Management

6.1 System Clock Control

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

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



System clock structure diagram

Related registers

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CKSEL	Clock selection register	FE00H	-						MCKSEL[1:0]		xxxx,xx00
CLKDIV	Clock Division Register	FE01H									0000,0100
HIRCCR	Internal high speed Oscillator control register	FE02H	ENIRC24M	-	-	-	-	-	-	IRC24MST	1xxx,xxx0
XOSCCR	External Oscillator control register	FE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST	00xx,xxx0
IRC32KCR	Internal 32KHz Oscillator control register	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0
MCLKOCR	Master clock output control register	FE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000

6.1.1 Clock selection register (CKSEL)

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

MCKSEL[1:0]: Master clock source selection

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

6.1.2 CLKDIV (Clock Division register)

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

CLKDIV: Main clock dividing coefficient. The system clock (SYSCLK) is the clock signal of master clock

(MLK) after being divided.

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

Note: After the user program is reset, the system will automatically set the initial value of this register according to the frequency division coefficient required for the operating frequency set during the last ISP download.

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

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

ENHIRC: internal high-speed and high-precision IRC enable bit

0: disable internal high-precision IRC

1: enable internal high-precision IRC

HIRCST: internal high-precision IRC frequency stability flag (read-only)

After the internal IRC is enabled from the stopped state, it must take some time for the frequency of the oscillator to become stable. The clock controller will set the HIRCST flag automatically after the internal oscillator frequency stabilizes. When the user program needs to switch the clock to internal IRC, ENHIRC must be set at first to enable the oscillator and then keep polling the oscillator stable flag HIRCST until the flag changes to 1.

6.1.4 External Oscillator control register (XOSCCR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
XOSCCR	FE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST

ENXOSC: external oscillator enable bit

0: disable external oscillator

1: enable external oscillator

XITYPE: external clock source type

0: The external clock source is the external clock signal (or active crystal). The signal source only needs to be connected to the XTALI(P1.7) of microcontroller. (Now, P1.6 is fixed in high-impedance input mode, which can be used to read external digital signals or used as ADC input, but it is generally not recommended to use it because the high-frequency oscillation signal of P1.7 will affect the signal of P1.6)

1: The external clock source is a crystal oscillator which is connected to XTALI (P1.7) and XTALO (P1.6) of microcontroller.

XOSCST: external crystal oscillator frequency stability flag (read-only)

After the external crystal oscillator is enabled from the stopped state, it must take some time for the frequency of the oscillator to become stable. The clock controller will set the XOSCSTflag automatically

after the oscillator frequency stabilizes. When the user program needs to switch the clock to external crystal oscillator, ENXOSC must be set at first to enable the oscillator and then keep polling the oscillator stable flag XOSCST until the flag changes to 1.

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

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

ENIRC32K: internal 32KHz low speed IRC enable bit

0: disable internal 32KHz low speed IRC

1: enable internal 32KHz low speed IRC

IRC32KST: internal 32KHz low speed IRC frequency stability flag (read-only)

After the internal 32KHz low speed IRC is enabled from the stopped state, it must take some time for the frequency of the oscillator to become stable. The clock controller will set the IRC32KST flag automatically after the internal oscillator frequency stabilizes. When the user program needs to switch the clock to the internal 32KHz low speed IRC, ENIRC32K must be set at first to enable the oscillator and then keep polling the oscillator stable flag IRC32KST until the flag changes to 1.

6.1.6 Master clock output control register (MCLKOCR)

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

MCLKODIV[6:0]: Master clock output division factor

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

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

MCLKO_S: Main clock output pin selection

0: Main clock output to P5.4

1: Main clock output to P1.6

6.2 STC8G series internal IRC frequency adjustment

All STC8G series of microcontrollers integrate a high-precision internal IRC oscillator. The ISP download software will automatically adjust the frequency according to the frequency selected / set by the user when users download user program using ISP. The general frequency value can be adjusted below ±0.3%. The temperature drift of the adjusted frequency can be -1.35% ~ 1.30% within the full temperature range (-40℃~85℃).

The internal IRC of the STC8G series has two frequency bands whose center frequencies are 20MHz and 33MHz respectively. The adjustment range of the 20M band is about 14.7MHz to 26MHz. The adjustment range of the 33M band is about 24.5MHz to 42.2MHz. Note: Different chips or different manufacturing batches may have manufacturing errors of about 5%. After actual testing, the maximum operating frequency of some chips can only be 39MHz. For safety reasons, it is recommended that the IRC frequency is set not higher than 35MHz during ISP download.

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

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

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

Related registers

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
IRCBAND	IRC band selection register	9DH	-	-	-	-	-	-	-	SEL	0000,00nn
LIRTRIM	IRC frequency trim register	9EH	-	-	-	-	-	-	LIRTRIM[1:0]		0000,00nn
IRTRIM	IRC frequency adjustment register	9FH	IRTRIM[7:0]								nnnn,nnnn
CLKDIV	Clock Divide Register	FE01H	CLKDIV[7:0]								0000,0100

6.2.1 IRC band selection register (IRCBAND)

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

SEL：band selection

0: Select 20MHz band

1: Select 33MHz band

6.2.2 IRC frequency adjustment register (IRTRIM)

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

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

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

6.2.3 IRC frequency trim register (LIRTIRM)

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

LIRTRIM[1:0]: Internal high-precision IRC frequency trim register

LIRTRIM can adjust the IRC frequency in 3 levels. The frequency range adjusted by the 3 levels is shown in the table below.

LIRTRIM[1:0]	Adjusted frequency range
00	Not fine-tuned
01	Adjusted about 0.10%
10	Adjusted about 0.04%
11	Adjusted about 0.10%

6.2.4 Clock Divide Register (CLKDIV)

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

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

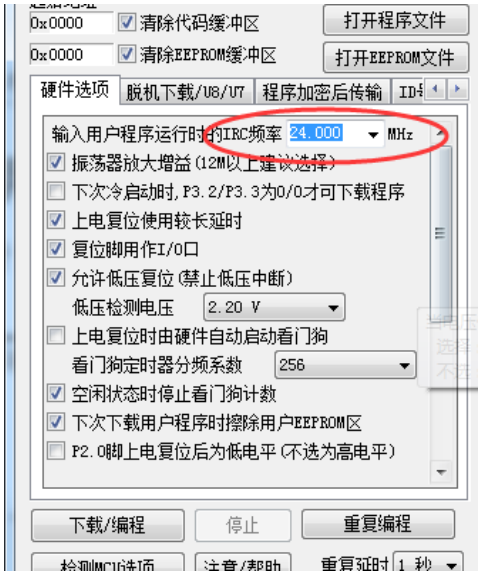
CLKDIV	System clock frequency
0	MCLK/1
1	MCLK/1
2	MCLK/2
3	MCLK/3
…	…
x	MCLK/x
…	…
255	MCLK/255

The adjustable ranges of the two frequency bands within the STC8G series of microcontrollers are 14.7MHz to 26MHz and 24.5MHz to 42.2MHz respectively. Although the upper limit of the 33MHz frequency band can be adjusted to more than 40MHz, the internal Flash program memory of the chip cannot run at a speed of more than 40MHz. Therefore, you should set the internal IRC frequency not higher than 40MHz when using ISP download. It is generally recommended that the frequency should be set below 35MHz. If you need a lower operating frequency, you can use the CLKDIV register to divide the adjusted frequency. For example, if you need a frequency of 11.0592MHz, and this frequency cannot be obtained using the internal IRC direct adjustment, but the internal IRC can be adjusted to 22.1184MHz, you can get 11.0592MHz by dividing by 2 with CLKDIV.

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

To get a frequency of 3MHz, you can use the method of 24MHz divided by 8.

Select the internal IRC operating frequency as 24MHz firstly when downloading the ISP, as shown in the figure below.



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

C language code

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

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

#define   CKSEL     (*(unsigned char volatile xdata *)0xfe00)
#define   CLKDIV    (*(unsigned char volatile xdata *)0xfe01)
#define   HIRCCR    (*(unsigned char volatile xdata *)0xfe02)
#define   XOSCCR    (*(unsigned char volatile xdata *)0xfe03)
#define   IRC32KCR  (*(unsigned char volatile xdata *)0xfe04)

sfr       P_SW2     =   0xba;
sfr       IRTRIM    =   0x9f;

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
```

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

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

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

    while (1);
}
```

Assembly code

```
; Operating frequency for test is 24MHz

P_SW2       DATA    0BAH
IRTRIM      DATA    09FH

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

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

            ORG     0000H
            LJMP    MAIN

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

STC MCU

```
        MOV     P5M1, #00H

        MOV     P_SW2,#80H
        MOV     A,#00H                  ; Select internal IRC
        MOV     DPTR,#CKSEL
        MOVX    @DPTR,A
        MOV     A,#08H                  ; Clock divided by 8
        MOV     DPTR,#CLKDIV
        MOVX    @DPTR,A
        MOV     P_SW2,#00H

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

        JMP     $

        END
```

STC MCU

6.3 System reset

There are two types of resets in STC8G series of microcontrollers, hardware reset and software reset.

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

- Power-on reset
- Low-voltage detection reset
- RST pin reset (Low-level reset)
- Watch-Dog-Timer reset

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

- Write SWRST bit in IAP_CONTR register to trigger reset

Related registers

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
WDT_CONTR	Watchdog control register	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]			0x00,0000
IAP_CONTR	IAP Control Register	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-				0000,xxxx
RSTCFG	Reset Configuration Registe	FFH	-	ENLVR	-	P54RST	-	-	LVDS[1:0]		0000,0000

6.2.1 Watch-Dog-Timer reset (WDT_CONTR)

In systems that require high reliability, such as industrial control/automotive electronics/aerospace, etc., in order to prevent "the system is interfered under abnormal conditions, the MCU/CPU program runs away, causing the system to work abnormally for a long time", usually the watch-dog-timer is introduced. If MCU/CPU does not access the watchdog as required within the specified time, it is considered that the MCU/CPU is in an abnormal state, and the watchdog will force the MCU/CPU to reset, so that the system restarts to execute the user program from the beginning.

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

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 : WDT reset flag.

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

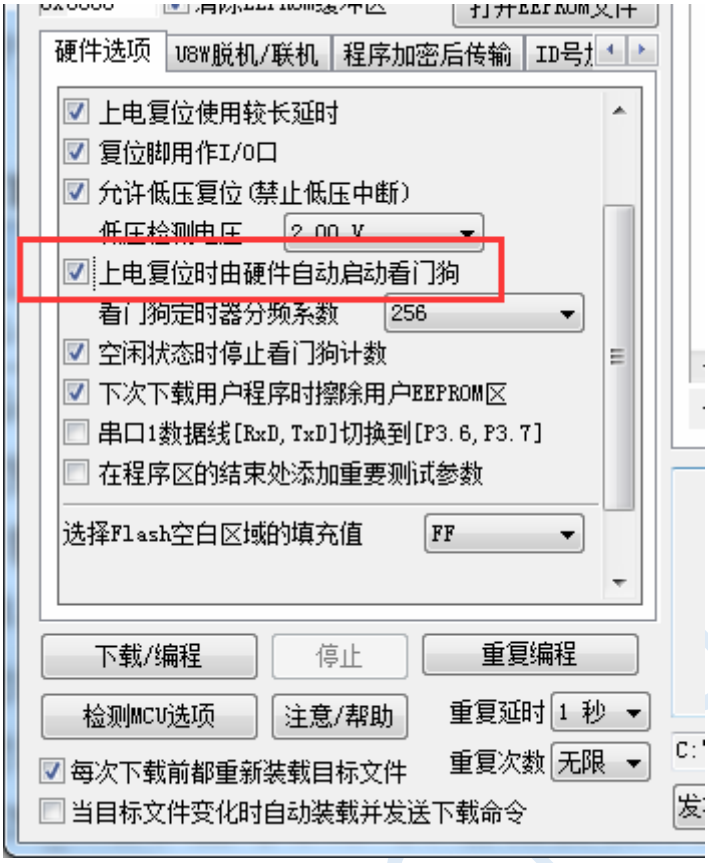
EN_WDT: WDT enable bit.

0: No operation

1: WDT is started.

Note: The watchdog timer can be started in software or automatically by hardware. Once the watchdog timer is started, the software cannot be turned off, and the microcontroller must be powered on

again to turn it off. The software only needs to write 1 to the EN_WDT bit to start the watchdog. If you need the hardware to start the watchdog, you need to perform the settings as shown in the figure below when downloading from the ISP:



CLR_WDT: WDT clear bit.

0: No operation

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

IDL_WDT: WDT control bit in IDLE mode.

0: WDT is disabled in IDLE mode.

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

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

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

The WDT overflow time is determined by the following equation:

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

6.3.2 Software reset (IAP_CONTR)

IAP Control Register (IAP_CONTR)

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

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-			

SWBS: Software boot selection bit

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

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

SWRST: Software reset trigger bit.

0: No operation

1: Trigger software reset.

6.3.3 Low-voltage detection reset (RSTCFG)

Reset Configuration Registe (RSTCFG)

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

ENLVR: Low voltage detection reset enable bit

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

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

P54RST: RST pin function selection bit

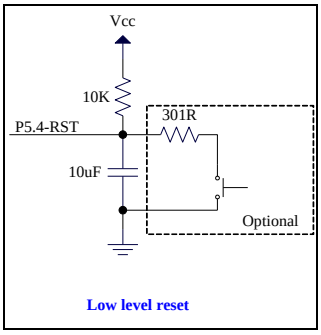
0: RST pin is used as common I/O (P5.4).

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

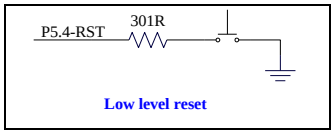
LVDS[1:0] : Low voltage detection threshold voltage setting bits

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

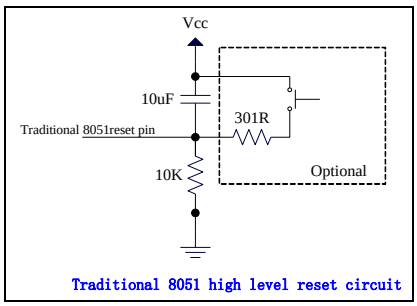
6.3.4 Low-voltage power-on reset reference cuicuit (No need in general)



6.3.5 Low-voltage manual button reset reference cuicuit



6.3.6 Power-on reset reference cuicuit of traditional 8051

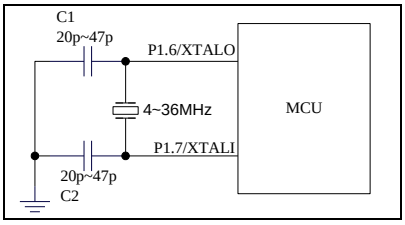


The picture above shows the high-level reset circuit of the traditional 8051. The reset of STC8G is low-level reset, which is different from the traditional reset circuit.

STC MCU

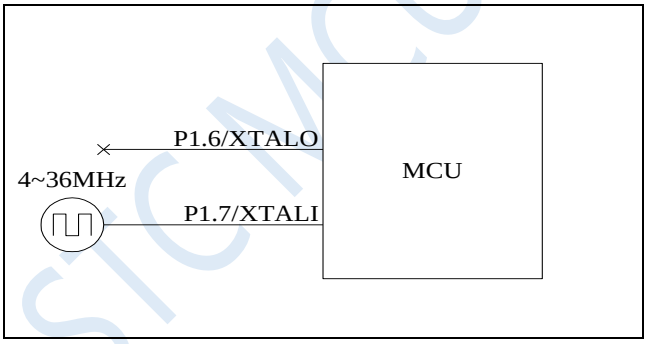
6.4 External crystal oscillator and external clock circuit

6.4.1 External crystal oscillator input cuicuit



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

6.4.2 External crystal oscillator input cuicuit (P1.6 cannot be used as general I/O)



6.5 Clock Stop/Power Saving Mode and System Power Management

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

6.5.1 Power control register (PCON)

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

LVDF: Low voltage detection flag. When the system detects a low-voltage event, it is set by the hardware automatically and an interrupt request to the CPU ocuurs. It should be cleared by user software.

POF: Power-On reset flag. It is set by the hardware when power-off-on action occurs everytime, and can only be cleared by software.

PD: Power-Down mode control bit

0: No operation.

1: Make the microcontroller entering power-down mode. CPU and all peripherals stop working. It is cleared by hardware automatically after the microcontroller wakes up.

IDL: IDLE mode control bit

0: No operation.

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

Note: Although both the LVD and the comparator can wake up the clock stop mode, it is not recommended to start the LVD and the comparator in the clock stop power saving mode, otherwise the hardware system will automatically start the internal 1.19V high-precision reference source. This high-precision reference source has a corresponding anti-temperature drift and adjustment circuit, which will increase the power consumption of about 300uA. After the MCU enters the clock stop mode, it only consumes about 0.4uA at the 3.3V operating voltage. So tt is not recommended to turn on the LVD and the comparator in the clock stop mode. If you really need to use it, it is recommended to turn on the power-down wake-up timer. The power-down wake-up timer will only increase the power consumption of about 1.4uA. This power consumption is generally acceptable for the system. Let the power-down wake-up timer wake up the MCU every 5 seconds. And after wake-up, the external battery voltage can be detected by LVD, comparator, and ADC. The detection will take about 1ms and then enter the clock stop/power saving mode again, so that the average current increase is less than 1uA, and the overall power consumption is about 2.8uA (0.4uA + 1.4uA + luA).

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

STC MCU

6.6 Power-down wake-up timer

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

6.6.1 Power-down wake-up timer counter register (WKTCL,WKTCH)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
WKTCL	AAH								
WKTCH	ABH	WKTEN							

WKTEN: Power-down wake-up timer enable bit

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

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

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

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

The calculation formula for the counting time of the dedicated timer for power-down wake-up is as follows: (where F_{wt} is the clock frequency of the dedicated timer for power-down wake-up we obtained from the RAM area F8H and F9H)

$$\text{time of the dedicated timer for power down wake up} = \frac{10^6 \times 16 \times \text{counting times}}{F_{wt}} \ (\text{us})$$

Asumme that F_{wt}=32KHz, then

{WKTCH[6:0], WKTCL[7:0]}	time of the dedicated timer for power-down wake-up
1	10^6*16*(1+1)/32K≈1ms
9	10^6*16*(1+9)/32K≈5ms
99	10^6*16*(1+99)/32K≈50ms
999	10^6*16*(1+999)/32K≈0.5s
4095	10^6*16*(1+4095)/32K≈2s
32766	10^6*16*(1+32766)/32K≈16s

6.7 Example Routines

6.7.1 System Clock Soure Selection

C language code

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

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

#define   CKSEL     (*(unsigned char volatile xdata *)0xfe00)
#define   CLKDIV    (*(unsigned char volatile xdata *)0xfe01)
#define   HIRCCR    (*(unsigned char volatile xdata *)0xfe02)
#define   XOSCCR    (*(unsigned char volatile xdata *)0xfe03)
#define   IRC32KCR  (*(unsigned char volatile xdata *)0xfe04)

sfr       P_SW2     =   0xba;

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

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

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

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

```
    P_SW2 = 0x00;
*/

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

Assembly code

```
; Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

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

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

            ORG     0000H
            LJMP    MAIN

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

            MOV     P_SW2,#80H
```

STC MCU

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

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

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

                JMP     $

                END
```

6.7.2 Main Clock Output after being devided

C language code

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

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

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

sfr     P_SW2     =   0xba;

sfr     P0M1      =   0x93;
sfr     P0M0      =   0x94;
sfr     P1M1      =   0x91;
sfr     P1M0      =   0x92;
sfr     P2M1      =   0x95;
sfr     P2M0      =   0x96;
```

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

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

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

    while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

MCLKOCR     EQU     0FE05H

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

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
```

```
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#80H
;       MOV     A,#01H          ;Main clock output to P5.4
;       MOV     A,#02H          ;Divide the main clock by 2 and output to P5.4
        MOV     A,#04H          ;Divide the main clock by 4 and output to P5.4
;       MOV     A,#84H          ;Divide the main clock by 4 and output to P1.6
        MOV     DPTR,#MCLKOCR
        MOVX    @DPTR,A
        MOV     P_SW2,#00H

        JMP     $

        END
```

6.7.3 Application of Watch-dog Timer

C language code

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

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

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

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

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

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

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

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

        Display();                      //Display module
        Scankey();                      //Key scan module
        MotorDriver();                  //Motor driving module

    }
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

WDT_CONTR   DATA    0C1H

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

            ORG     0000H
            LJMP    MAIN

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

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

;               MOV     WDT_CONTR,#23H      ;Watchdog enabled, overflow time is about 0.5s
                MOV     WDT_CONTR,#24H      ;Watchdog enabled, overflow time is about 1s
;               MOV     WDT_CONTR,#27H      ;Watchdog enabled, overflow time is about 8s
                CLR     P3.2                ;Test port
LOOP:
;               MOV     WDT_CONTR,#33H      ;Clear watchdog timer, otherwise system reset
                MOV     WDT_CONTR,#34H      ;Clear watchdog timer, otherwise system reset
;               MOV     WDT_CONTR,#37H      ;Clear watchdog timer, otherwise system reset
                LCALL   DISPLAY             ;Display module
                LCALL   SCANKEY             ;Key scan module
                LCALL   MOTORDRIVER         ;Motor driving module
                JMP     LOOP

                END
```

6.7.4 User Defined Downloading by Using Software Reset

C language code

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

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

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

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

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

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

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

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

Assembly code

```
; Operating frequency for test is 11.0592MHz

IAP_CONTR   DATA    0C7H

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

            ORG     0000H
            LJMP    MAIN

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

            SETB    P3.2
            SETB    P3.3
```

STC MCU

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

        END
```

6.7.5 Low Voltage Detection

C language code

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

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

sfr     RSTCFG  =   0xff;
#define ENLVR       0x40            //RSTCFG.6
#define LVD2V0      0x00            //LVD@2.0V
#define LVD2V4      0x01            //LVD@2.4V
#define LVD2V7      0x02            //LVD@2.7V
#define LVD3V0      0x03            //LVD@3.0V
sbit    ELVD    =   IE^6;
#define LVDF        0x20            //PCON.5
sbit    P32     =   P3^2;

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

void Lvd_Isr() interrupt 6
{
    PCON &= ~LVDF;                  //Clear interrupt flag
    P32 = ~P32;                     //Test port
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
```

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

    PCON &= ~LVDF;                  //Test port
//  RSTCFG = ENLVR | LVD3V0;        //Low voltage reset when 3.0V is enabled, no LVD interrupt is generated
    RSTCFG = LVD3V0;                //Low voltage interrupt when 3.0V is enabled
    ELVD = 1;                       //Enable LVD interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

RSTCFG      DATA    0FFH
ENLVR       EQU     40H         ;RSTCFG.6
LVD2V0      EQU     00H         ;LVD@2.0V
LVD2V4      EQU     01H         ;LVD@2.4V
LVD2V7      EQU     02H         ;LVD@2.7V
LVD3V0      EQU     03H         ;LVD@3.0V

ELVD        BIT     IE.6
LVDF        EQU     20H         ;PCON.5

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

            ORG     0000H
            LJMP    MAIN
            ORG     0033H
            LJMP    LVDISR

            ORG     0100H
LVDISR:
            ANL     PCON,#NOT LVDF  ;Clear interrupt flag
            CPL     P3.2            ;Test port
            RETI
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
```

```
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        ANL     PCON,#NOT LVDF          ;LVDF flag needs to be cleared after power on
;       MOV     RSTCFG,#ENLVR | LVD3V0  ;Low voltage reset when 3.0V is enabled, no LVD interrupt
is generated
        MOV     RSTCFG,#LVD3V0          ;Low voltage interrupt when 3.0V is enabled
        SETB    ELVD                    ;Enable LVD interrupt
        SETB    EA
        JMP     $

        END
```

6.7.6 Power Saving Mode

C language code

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

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

#define   IDL             0x01            //PCON.0
#define   PD              0x02            //PCON.1
sbit      P34       =     P3^4;
sbit      P35       =     P3^5;

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

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

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

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

    EX0 = 1;                        //Enable INT0 interrupt to wake up MCU
    EA = 1;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    PCON = IDL;                     //MCU enters IDLE mode
//  PCON = PD;                      //MCU enters power down mode
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    P35 = 0;

    while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     0003H
            LJMP    INT0ISR

            ORG     0100H
INT0ISR:
            CPL     P3.4            ;Test port
            RETI
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
```

```
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        SETB    EX0             ;Enable INT0 interrupt to wake up MCU
        SETB    EA
        NOP
        NOP
        NOP
        NOP
;       MOV     PCON,#IDL       ;MCU enters IDLE mode
        MOV     PCON,#PD        ;MCU enters power down mode
        NOP
        NOP
        NOP
        NOP
        CLR     P3.5            ;Test port
        JMP     $

        END
```

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

C language code

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

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

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

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
```

```
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void INT0_Isr() interrupt 0
{
    P10 = !P10;                                 //Test port
}

void INT1_Isr() interrupt 2
{
    P10 = !P10;                                 //Test port
}

void INT2_Isr() interrupt 10
{
    P10 = !P10;                                 //Test port
}

void INT3_Isr() interrupt 11
{
    P10 = !P10;                                 //Test port
}

void INT4_Isr() interrupt 16
{
    P10 = !P10;                                 //Test port
}

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

    IT0 = 0;                                    //Enable INT0 rising edge and falling edge interrupts
//  IT0 = 1;                                    //Enable INT0 falling edge interrupt
    EX0 = 1;                                    //Enable INT0 interrupt

    IT1 = 0;                                    //Enable INT1 rising edge and falling edge interrupts
//  IT1 = 1;                                    //Enable INT1 falling edge interrupt
    EX1 = 1;                                    //Enable INT1 interrupt

    INTCLKO = EX2;                              //Enable INT2 falling edge interrupt
    INTCLKO |= EX3;                             //Enable INT3 falling edge interrupt
    INTCLKO |= EX4;                             //Enable INT4 falling edge interrupt

    EA = 1;
```

STC MCU

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

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

Assembly code

```
; Operating frequency for test is 11.0592MHz

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

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

            ORG     0000H
            LJMP    MAIN

            ORG     0003H
            LJMP    INT0ISR
            ORG     0013H
            LJMP    INT1ISR
            ORG     0053H
            LJMP    INT2ISR
            ORG     005BH
            LJMP    INT3ISR
            ORG     0083H
            LJMP    INT4ISR

            ORG     0100H
INT0ISR:
            CPL     P1.0            ;Test port
            RETI
INT1ISR:
            CPL     P1.0            ;Test port
            RETI
INT2ISR:
            CPL     P1.0            ;Test port
            RETI
```

```
INT3ISR:
        CPL     P1.0            ;Test port
        RETI
INT4ISR:
        CPL     P1.0            ;Test port
        RETI

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

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

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

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

        SETB    EA

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

        END
```

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

C language code

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

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

sfr         T2L             =   0xd7;
sfr         T2H             =   0xd6;
sfr         T3L             =   0xd5;
sfr         T3H             =   0xd4;
sfr         T4L             =   0xd3;
sfr         T4H             =   0xd2;
sfr         T4T3M           =   0xd1;
sfr         AUXR            =   0x8e;
sfr         IE2             =   0xaf;
#define     ET2                 0x04
#define     ET3                 0x20
#define     ET4                 0x40
sfr         AUXINTIF        =   0xef;
#define     T2IF                0x01
#define     T3IF                0x02
#define     T4IF                0x04

sbit        P10             =   P1^0;
sbit        P11             =   P1^1;

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

void TM0_Isr() interrupt 1
{
    P10 = !P10;                                 //Test port
}

void TM1_Isr() interrupt 3
{
    P10 = !P10;                                 //Test port
}

void TM2_Isr() interrupt 12
{
    P10 = !P10;                                 //Test port
}

void TM3_Isr() interrupt 19
{
    P10 = !P10;                                 //Test port
}

void TM4_Isr() interrupt 20
{
```

STC MCU

```
    P10 = !P10;                                 //Test port
}

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

    TMOD = 0x00;
    TL0 = 0x66;                                 //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                    //Start timer
    ET0 = 1;                                    //Enable timer interrupt

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

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

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

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

    EA = 1;

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

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

Assembly code

```
; Operating frequency for test is 11.0592MHz

T2L     DATA    0D7H
T2H     DATA    0D6H
T3L     DATA    0D5H
T3H     DATA    0D4H
T4L     DATA    0D3H
T4H     DATA    0D2H
T4T3M   DATA    0D1H
AUXR    DATA    8EH

IE2     DATA    0AFH
ET2     EQU     04H
ET3     EQU     20H
ET4     EQU     40H

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

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

        ORG     0000H
        LJMP    MAIN
        ORG     000BH
        LJMP    TM0ISR
        ORG     001BH
        LJMP    TM1ISR
        ORG     0063H
        LJMP    TM2ISR
        ORG     009BH
        LJMP    TM3ISR
        ORG     00A3H
        LJMP    TM4ISR

        ORG     0100H
TM0ISR:
        CPL     P1.0            ;Test port
        RETI
TM1ISR:
        CPL     P1.0            ;Test port
        RETI
TM2ISR:
        CPL     P1.0            ;Test port
        RETI
TM3ISR:
```

```
            CPL     P1.0            ;Test port
            RETI
TM4ISR:
            CPL     P1.0            ;Test port
            RETI

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

            MOV     TMOD,#00H
            MOV     TL0,#66H        ;65536-11.0592M/12/1000
            MOV     TH0,#0FCH
            SETB    TR0             ;Start timer
            SETB    ET0             ;Enable timer interrupt

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

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

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

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

            SETB    EA

            MOV     PCON,#02H       ;MCU enters power down mode
            NOP     ;Does not enter the interrupt service routine immediately after wake-up from power down
mode
                    ;Instead, wait for the timer to overflow before entering the interrupt service routine.
            NOP
            NOP
            NOP
LOOP:
            CPL     P1.1
            JMP     LOOP
```

END

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

C language code

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

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

sfr     IE2     =   0xaf;
#define ES2         0x01
#define ES3         0x08
#define ES4         0x10

sfr     P_SW1   =   0xa2;
sfr     P_SW2   =   0xba;

sbit    P11     =   P1^1;

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

void UART1_Isr() interrupt 4
{
}

void UART2_Isr() interrupt 8
{
}

void UART3_Isr() interrupt 17
{
}

void UART4_Isr() interrupt 18
{
}

void main()
{
```

STC MCU

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

    P_SW1 = 0x00;                   //Wake up MCU on the falling edge of RXD / P3.0
//  P_SW1 = 0x40;                   //Wake up MCU on the falling edge of RXD_2/P3.6
//  P_SW1 = 0x80;                   //Wake up MCU on the falling edge of RXD_3/P1.6
//  P_SW1 = 0xc0;                   //Wake up MCU on the falling edge of RXD_4/P4.3

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

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

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

    ES = 1;                         //Enable UART1 interrupt
    IE2 = ES2;                      //Enable UART2 interrupt
    IE2 |= ES3;                     //Enable UART3 interrupt
    IE2| = ES4;                     //Enable UART4 interrupt
    EA = 1;

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

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

Assembly code

```
; Operating frequency for test is 11.0592MHz

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

P_SW1       DATA    0A2H
P_SW2       DATA    0BAH

P0M1        DATA    093H
P0M0        DATA    094H
```

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

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART1ISR
            ORG     0043H
            LJMP    UART2ISR
            ORG     008BH
            LJMP    UART3ISR
            ORG     0093H
            LJMP    UART4ISR

            ORG     0100H
UART1ISR:
            RETI
UART2ISR:
            RETI
UART3ISR:
            RETI
UART4ISR:
            RETI

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

            MOV     P_SW1,#00H      ;Wake up MCU on the falling edge of RXD / P3.0
;           MOV     P_SW1,#40H      ;Wake up MCU on the falling edge of RXD_2/P3.6
;           MOV     P_SW1,#80H      ;Wake up MCU on the falling edge of RXD_3/P1.6
;           MOV     P_SW1,#0C0H     ;Wake up MCU on the falling edge of RXD_4/P4.3

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

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

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

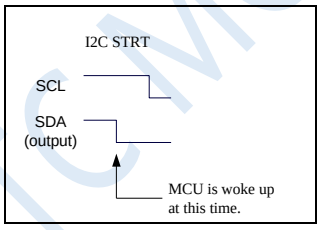
STC MCU

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

                SETB    ES                      ;Enable UART1 interrupt
                MOV     IE2,#ES2                ;Enable UART2 interrupt
                ORL     IE2,#ES3                ;Enable UART3 interrupt
                ORL     IE2,#ES4                ;Enable UART4 interrupt
                SETB    EA
                NOP     ; After the MCU is woken up by falling edges of RxD/RxD2/RxD3/RxD4 and the corresponding switchable sets of pins, it does not enter the interrupt service routine, and continue to execute the program. This is different from the INT0/INTI/INT2/INT3/INT4 power-down wake-up. It is recommended to add a few more NOP instructions, such as 3 or more.
                NOP
                MOV     PCON,#02H               ;MCU enters power down mode
                NOP     ;It will not enter the interrupt service routine after wake-up from power down mode.
                NOP
                NOP
                NOP
LOOP:
                CPL     P1.1
                JMP     LOOP

                END
```

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



STCMCU

C language code

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

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

sfr     P_SW2       =   0xba;

#define I2CCFG          (*(unsigned char volatile xdata *)0xfe80)
#define I2CSLCR         (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST         (*(unsigned char volatile xdata *)0xfe84)

sbit    P11         =   P1^1;

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
```

```
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void i2c_isr() interrupt 24
{
    P_SW2 |= 0x80;
    I2CSLST &= ~0x40;
}

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

    P_SW2 = 0x00;                   //Wake up MCU on the falling edge of SDA/P1.4
//  P_SW2 = 0x10;                   //Wake up MCU on the falling edge of SDA_2/P2.4
//  P_SW2 = 0x30;                   //Wake up MCU on the falling edge of SDA_4/P3.3
    P_SW2 |= 0x80;
    I2CCFG = 0x80;                  //Enable slave mode of I2C module
    I2CSLCR = 0x40;                 //Enable start signal interrupt
    EA = 1;

    PCON = 0x02;                    //MCU enters power down mode
    _nop_();
    _nop_();
    _nop_();
    _nop_();

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

Assembly code

```
; Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     00C3H
            LJMP    I2CISR

            ORG     0100H
I2CISR:
            PUSH    ACC
            PUSH    DPH
            PUSH    DPL
            ORL     PSW2,#80H
            MOV     DPTR,#I2CSLST
            MOVX    A,@DPTR
            ANL     A,#NOT 40H
            MOVX    @DPTR,A
            POP     DPL
            POP     DPH
            POP     ACC
            RETI

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

            MOV     P_SW2,#00H          ;Wake up MCU on the falling edge of SDA/P1.4
//          MOV     P_SW2,#10H          ;Wake up MCU on the falling edge of SDA_2/P2.4
//          MOV     P_SW2,#30H          ;Wake up MCU on the falling edge of SDA_4/P3.3
            ORL     P_SW2,#80H
            MOV     DPTR,#I2CCFG
            MOV     A,#80H
            MOVX    @DPTR,A             ;Enable slave mode of I2C module
            MOV     DPTR,# I2CSLCR
            MOV     A,#40H              ;Enable start signal interrupt
            SETB    EA

            MOV     PCON,#02H           ;MCU enters power down mode
            NOP     ;It will not enter the interrupt service routine after wake-up from power down mode.
            NOP
```

STC MCU

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

        END
```

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

C language code

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

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

sfr     WKTCL   =   0xaa;
sfr     WKTCH   =   0xab;

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

sbit    P11     =   P1^1;

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

    WKTCL = 0xff;                   //Set the power-down wake-up timer as 1s or so
    WKTCH= 0x87;
```

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

Assembly code

```
; Operating frequency for test is 11.0592MHz

WKTCL       DATA    0AAH
WKTCH       DATA    0ABH

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

            ORG     0000H
            LJMP    MAIN

            ORG     0100H

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

            MOV     WKTCL,#0FFH         ;Set power-down wake-up timer as 1s or so.
            MOV     WKTCH,#87H

LOOP:
            NOP
            NOP
```

STC MCU

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

        END
```

6.7.12 Wake up MCU from Power Saving Mode using LVD interrupt

It is not recommended to start the LVD and the comparator in the clock stop power saving mode, otherwise the hardware system will automatically start the internal 1.19V high-precision reference source. This high-precision reference source has a corresponding anti-temperature drift and adjustment circuit, which will increase the power consumption of about 300uA. After the MCU enters the clock stop mode, it only consumes about 0.4uA at the 3.3V operating voltage. So tt is not recommended to turn on the LVD and the comparator in the clock stop mode. If you really need to use it, it is recommended to turn on the power-down wake-up timer. The power-down wake-up timer will only increase the power consumption of about 1.4uA. This power consumption is generally acceptable for the system. Let the power-down wake-up timer wake up the MCU every 5 seconds. And after wake-up, the external battery voltage can be detected by LVD, comparator, and ADC. The detection will take about 1ms and then enter the clock stop/power saving mode again, so that the average current increase is less than 1uA, and the overall power consumption is about 2.8uA (0.4uA + 1.4uA + luA).

C language code

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

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

sfr     RSTCFG  =   0xff;
#define ENLVR       0x40            //RSTCFG.6
#define LVD2V0      0x00            //LVD@2.0V
#define LVD2V4      0x01            //LVD@2.4V
#define LVD2V7      0x02            //LVD@2.7V
#define LVD3V0      0x03            //LVD@3.0V
sbit    ELVD    =   IE^6;
#define LVDF        0x20            //PCON.5

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
```

```
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void LVD_Isr() interrupt 6
{
    PCON &= ~LVDF;                  //Clear interrupt flag
    P10 = !P10;                     //Test port
}

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

    PCON &= ~LVDF;                  //Interrupt flag needs to be cleared at power-on
    RSTCFG = LVD3V0;                //Set the LVD voltage to 3.0V
    ELVD = 1;                       //Enable LVD interrupt
    EA = 1;

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

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

Assembly code

```
; Operating frequency for test is 11.0592MHz

RSTCFG      DATA    0FFH
ENLVR       EQU     40H             ;RSTCFG.6
LVD2V0      EQU     00H             ;LVD@2.0V
LVD2V4      EQU     01H             ;LVD@2.4V
LVD2V7      EQU     02H             ;LVD@2.7V
LVD3V0      EQU     03H             ;LVD@3.0V

ELVD        BIT     IE.6
LVDF        EQU     20H             ;PCON.5

P0M1        DATA    093H
P0M0        DATA    094H
```

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

            ORG     0000H
            LJMP    MAIN
            ORG     0033H
            LJMP    LVDISR

            ORG     0100H
LVDISR:
            ANL     PCON,#NOT LVDF      ;Clear interrupt flag
            CPL     P1.0                ;Test port
            RETI

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

            ANL     PCON,#NOT LVDF      ;Interrupt flag needs to be cleared at power-on
            MOV     RSTCFG,# LVD3V0     ;Set the LVD voltage to 3.0V
            SETB    ELVD                ;Enable LVD interrupt
            SETB    EA

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

            END
```

STC MCU

6.7.13 Wake up MCU from Power Saving Mode using CCP0/CCP1/CCP2 interrupts

C language code

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

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

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

sfr     P_SW1   =   0xa2;

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;

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

void PCA_Isr() interrupt 7
{
    CCON &= ~0x8f;                  //Clear interrupt flag
    P10 = !P10;                     //Test port
}
```

STC MCU

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

    CCON = 0x00;
    CMOD = 0x08;                    //PCA clock is the system clock
    CCAPM0 = 0x31;                  //Enable CCP0 edge wake-up function
    CCAPM1 = 0x31;                  //Enable CCP1 edge wake-up function
    CCAPM2 = 0x31;                  //Enable CCP2 edge wake-up function

    CR = 1;                         //Start PCA timer
    EA = 1;

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

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

Assembly code

```
; Operating frequency for test is 11.0592MHz

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

```
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H

P_SW1       DATA    0A2H

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

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

            ORG     0100H
PCAISR:
            ANL     CCON,#NOT 8FH       ;Clear interrupt flag
            CPL     P1.0                ;Test port
            RETI

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

            MOV     CCON,#00H
            MOV     CMOD,#08H           ;PCA clock is the system clock
            MOV     CCAPM0,#31H         ;Enable CCP0 edge wake-up function
            MOV     CCAPM1,#31H         ;Enable CCP1 edge wake-up function
            MOV     CCAPM2,#31H         ;Enable CCP2 edge wake-up function
            SETB    CR                  ;Start PCA timer
            SETB    EA

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

LOOP:
```

STC MCU

```
        CPL     P1.1
        JMP     LOOP

        END
```

6.7.14 Wake up MCU from Power Saving Mode using CMP interrupt

It is not recommended to start the LVD and the comparator in the clock stop power saving mode, otherwise the hardware system will automatically start the internal 1.19V high-precision reference source. This high-precision reference source has a corresponding anti-temperature drift and adjustment circuit, which will increase the power consumption of about 300uA. After the MCU enters the clock stop mode, it only consumes about 0.4uA at the 3.3V operating voltage. So tt is not recommended to turn on the LVD and the comparator in the clock stop mode. If you really need to use it, it is recommended to turn on the power-down wake-up timer. The power-down wake-up timer will only increase the power consumption of about 1.4uA. This power consumption is generally acceptable for the system. Let the power-down wake-up timer wake up the MCU every 5 seconds. And after wake-up, the external battery voltage can be detected by LVD, comparator, and ADC. The detection will take about 1ms and then enter the clock stop/power saving mode again, so that the average current increase is less than 1uA, and the overall power consumption is about 2.8uA (0.4uA + 1.4uA + luA).

C language code

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

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

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

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;

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

void CMP_Isr() interrupt 21
{
    CMPCR1 &= ~0x40;                    //Clear interrupt flag
    P10 = !P10;                         //Test port
}
```

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

    CMPCR2 = 0x00;
    CMPCR1 = 0x80;                  //Enable comparator module
    CMPCR1 |= 0x30;                 //Enable edge interrupt of comparator
    CMPCR1 &= ~0x08;                //P3.6 is CMP + input pin
    CMPCR1 |= 0x04;                 //P3.7 is CMP- input pin
    CMPCR1 |= 0x02;                 //Enable Comparator output
    EA = 1;

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

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

Assembly code

```
; Operating frequency for test is 11.0592MHz

CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

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

            ORG     0000H
            LJMP    MAIN
            ORG     00ABH
            LJMP    CMPISR
```

```
            ORG     0100H
CMPISR:
            ANL     CMPCR1,#NOT 40H     ;Clear interrupt flag
            CPL     P1.0                ;Test port
            RETI

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

            MOV     CMPCR2,#00H
            MOV     CMPCR1,#80H         ;Enable comparator module
            ORL     CMPCR1,#30H         ;Enable edge interrupt of comparator
            ANL     CMPCR1,#NOT 08H     ;P3.6 is CMP + input pin
            ORL     CMPCR1,#04H         ;P3.7 is CMP- input pin
            ORL     CMPCR1,#02H         ;Enable Comparator output
            SETB    EA

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

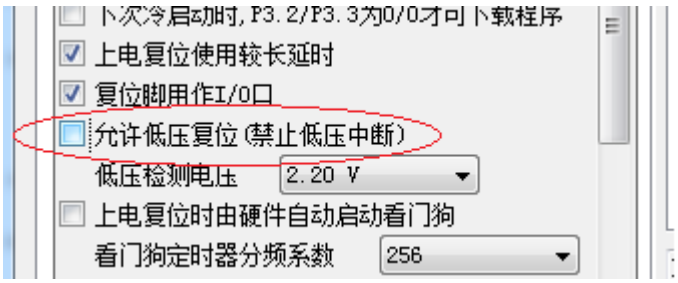
LOOP:
            CPL     P1.1
            JMP     LOOP

            END
```

6.7.15 Detect the Operating Voltage (Battery Voltage) using LVD

If you need to use LVD to detect the battery voltage, you need to remove the low-voltage reset function when downloading from the ISP, as shown in the following figure.

(It is recommended to use the 15th channel of ADC to detect battery voltage, see ADC chapter.)



C language code

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

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

#define FOSC            11059200UL
#define T1MS            (65536 - FOSC/4/100)

sfr     RSTCFG  =   0xff;
#define LVD2V0          0x00                    //LVD@2.0V
#define LVD2V4          0x01                    //LVD@2.4V
#define LVD2V7          0x02                    //LVD@2.7V
#define LVD3V0          0x03                    //LVD@3.0V

#define LVDF            0x20                    //PCON.5

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

void delay()
{
    int i;

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

void main()
{
    unsigned char power;

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

STC MCU

```
    PCON &= ~LVDF;
    RSTCFG = LVD3V0;

    while (1)
    {
        power = 0x0f;

        RSTCFG = LVD3V0;
        delay();
        PCON &= ~LVDF;
        delay();
        if (PCON & LVDF)
        {
            power >>= 1;
            RSTCFG = LVD2V7;
            delay();
            PCON &= ~LVDF;
            delay();
            if (PCON & LVDF)
            {
                power >>= 1;
                RSTCFG = LVD2V4;
                delay();
                PCON &= ~LVDF;
                delay();
                if (PCON & LVDF)
                {
                    power >>= 1;
                    RSTCFG = LVD2V2;
                    delay();
                    PCON &= ~LVDF;
                    delay();
                    if (PCON & LVDF)
                    {
                        power >>= 1;
                    }
                }
            }
        }
        RSTCFG = LVD3V0;
        P2 = ~power;     // P2.3 ~ P2.0 are used to display battery level
    }
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

RSTCFG      DATA    0FFH
LVD2V0      EQU     00H             ;LVD@2.0V
LVD2V4      EQU     01H             ;LVD@2.4V
LVD2V7      EQU     02H             ;LVD@2.7V
LVD3V0      EQU     03H             ;LVD@3.0V

LVDF        EQU     20H             ;PCON.5

P0M1        DATA    093H
```

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

            ORG     0000H
            JMP     MAIN

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

            ANL     PCON,#NOT LVDF
            MOV     RSTCFG,#LVD3V0
LOOP:
            MOV     B,#0FH

            MOV     RSTCFG,#LVD3V0
            CALL    DELAY
            ANL     PCON,#NOT LVDF
            CALL    DELAY
            MOV     A,PCON
            ANL     A,#LVDF
            JZ      SKIP
            MOV     A,B
            CLR     C
            RRC     A
            MOV     B,A

            MOV     RSTCFG,#LVD2V7
            CALL    DELAY
            ANL     PCON,#NOT LVDF
            CALL    DELAY
            MOV     A,PCON
            ANL     A,#LVDF
            JZ      SKIP
            MOV     A,B
            CLR     C
            RRC     A
            MOV     B,A
```

STC MCU

```
        MOV     RSTCFG,#LVD2V4
        CALL    DELAY
        ANL     PCON,#NOT LVDF
        CALL    DELAY
        MOV     A,PCON
        ANL     A,#LVDF
        JZ      SKIP
        MOV     A,B
        CLR     C
        RRC     A
        MOV     B,A

        MOV     RSTCFG,#LVD2V2
        CALL    DELAY
        ANL     PCON,#NOT LVDF
        CALL    DELAY
        MOV     A,PCON
        ANL     A,#LVDF
        JZ      SKIP
        MOV     A,B
        CLR     C
        RRC     A
        MOV     B,A

SKIP:
        MOV     A,B
        CPL     A
        MOV     P2,A                ; P2.3 ~ P2.0 are used to display battery level
        JMP     LOOP

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

        END
```

STC MCU

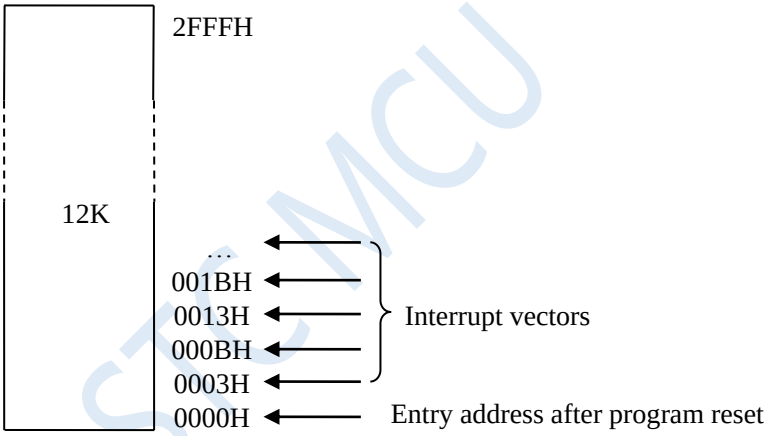
7 Memory

The STC8G series of microcontrollers have separate address spaces for Program Memory and Data Memory. Since no bus is provided for accessing external program memory, all program memory for all microcontrollers is on-chip Flash memory. The microcontrollers can not access external program memory.

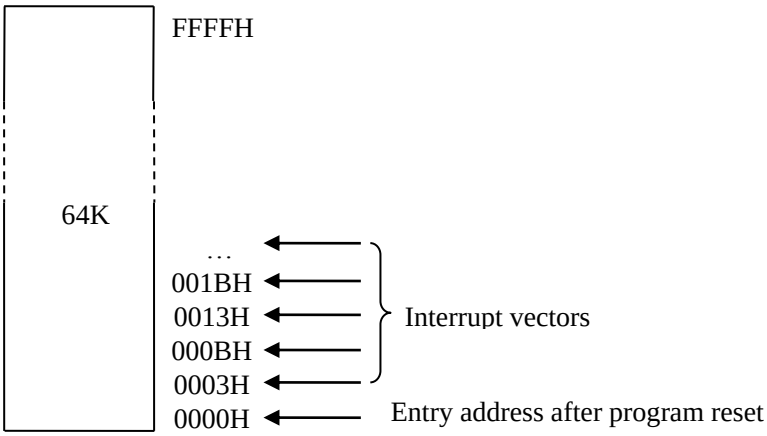
Large-capacity data memory is integrated in STC8G series of microcontrollers. The data memory inside the STC8G series of microcontrollers is physically and logically separated into two address spaces: 256 bytes of internal RAM and internal extended RAM. The addresses of the high 128 bytes of internal RAM and special function registers (SFRs) overlap. They can be accessed through different addressing modes in actual use.

7.1 Program Memory

Program memory is used to store user programs, data, tables and other information. 12K bytes of Flash program memory is integrated in STC8G1K08 family, STC8G1K08-8Pin family, STC8G1K08A family, STC8G1K08T family of microcontrollers.



64K bytes of Flash program memory is integrated in STC8G2K64S4 family, STC8G2K64S2 family, STC15H2K64S4 family of microcontrollers.



After the microcontroller resets, the content of the Program Counter (PC) is 0000H, and the CPU begins to execute program from 0000H of Program Memory. The entry addresses of interrupt service routines, which are also called interrupt vectors, are also located in the program memory. Each interrupt has a fixed entry address in Program Memory. When an interrupt occurs and gets response, the microcontroller will automatically jump to

its corresponding interrupt entry address to execute the service routine. The entry address of the interrupt service routine for the external interrupt 0 (INT0) is 0003H, the entry address for the timer / counter 0 (TIMER0) interrupt service routine is 000BH, and the entry address for the interrupt service routine for the external interrupt 1 (INT1) is 0013H. The counter/counter 1 (TIMER1) interrupt service routine's entry address is 001BH. More interrupt service routine entry address (interrupt vector), please refer to interrupt chapter.

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

All STC8G series of microcontrollers integrate Flash data memory (EEPROM). The EEPROM is read or written in byte, and is erased in page of 512-bytes. It can be repeatedly programmed and erased over 100,000 times, which improves the flexibility and convenience of use.

STC MCU

7.2 Data Memory

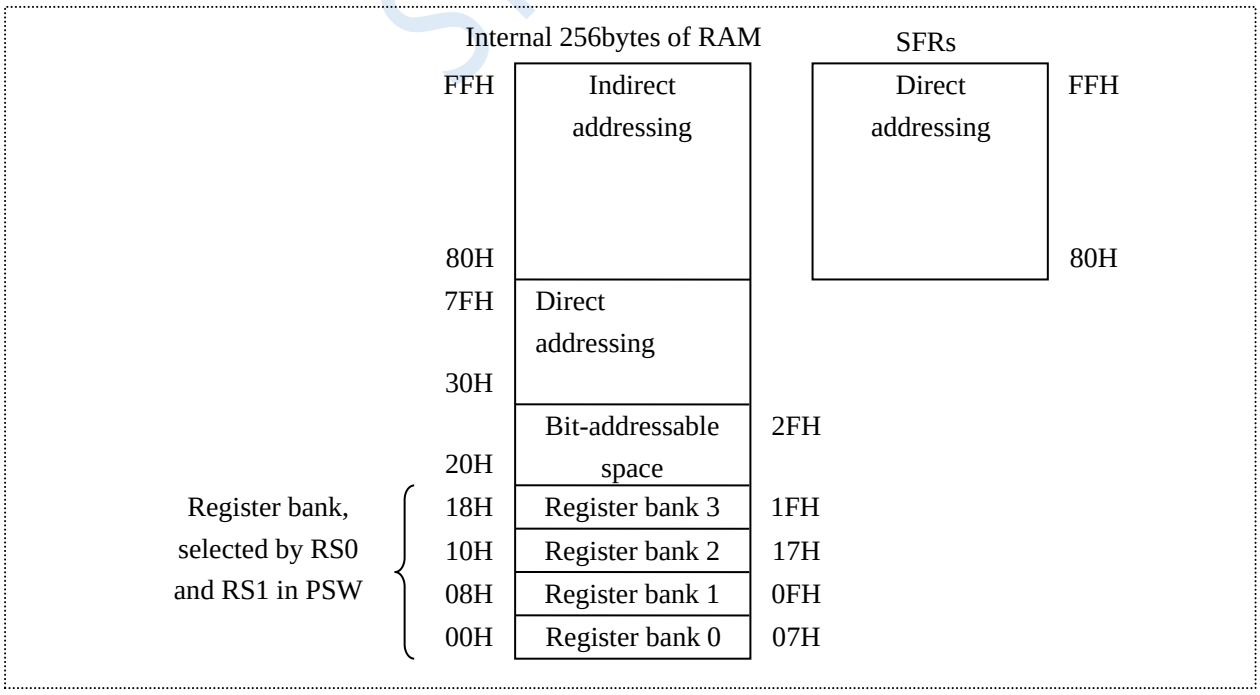
The RAM integrated in the STC8G series of microcontrollers can be used to store intermediate results and process data during program execution.

family of microcontrollers	Internal direct access RAM (DATA)	Internal indirect access RAM (IDATA)	On-chip extended RAM (XDATA)
STC8G1K08 family	128 bytes	128 bytes	1024 bytes
STC8G2K64S4 family	128 bytes	128 bytes	2048 bytes
STC8G2K64S2 family	128 bytes	128 bytes	2048 bytes
STC8G1K08-8Pin family	128 bytes	128 bytes	1024 bytes
STC8G1K08A family	128 bytes	128 bytes	1024 bytes
STC8G1K08T family	128 bytes	128 bytes	1024 bytes
STC15H2K64S4 family	128 bytes	128 bytes	2048 bytes

7.2.1 Internal RAM

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

Internal RAM is mapped in the following figure.



The Lower 128 bytes of RAM are also called as general-purpose RAM space. The general-purpose RAM space can be divided into working register banks space, bit addressable space, user RAM space and stack space.

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

7.2.2 Program Status Word register (PSW)

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

CY: Carry/borrow flag bit.

AC: Auxiliary carry/borrow flag.

F0: User flag bit 0.

RS1, RS0: Working register select bit

RS1	RS0	working register bank (R0~R7)
0	0	Bank 0 (00H~07H)
0	1	Bank 1 (08H~0FH)
1	0	Bank 2 (10H~17H)
1	1	Bank 3 (18H~1FH)

OV: Overflow flag.

F1: User flag bit 1.

P: Parity check flag.

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

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

SP is an 8-bit dedicated register. It indicates the top of the stack in the internal RAM. On reset, SP is initialized to 07H, which makes the stack space begin from 08H. The addresses 08H to 1FH are also the addresses of working register bank 1 through 3. It is better to change the SP value to a value of 80H or more if these spaces are used in user application. The stack of STC8G series of microcontrollers grows upward, which means that when a datum is pushed into the stack, the content of SP will increase.

7.2.3 On-chip extended RAM, XRAM, XDATA

In addition to 256 bytes of internal RAM, on-chip extended RAM is integrated in STC8G series of microcontrollers. The method of accessing the on-chip extended RAM is the same as that of the traditional 8051 MCU accessing the external extended RAM. However, the P0 port (data bus and low-order address bus), P2 port (high-order address bus), RD, WR and ALE are not affected.

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

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

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

```
unsigned char xdata i;
unsigned int pdata j;
```

Note that pdata is the lower 256 bytes of xdata. After declaring a variable as the pdata type in C program, the compiler will automatically allocate the variables in the 0000H to 00FFH of XDATA and use MOVX @ Ri, A and MOVX A @ Ri to access.

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

7.2.4 Auxiliary register (AUXR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	**EXTRAM**	S1ST2

EXTRAM: on-chip extended RAM access control bit

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

1: On-chip extended RAM is disabled.

7.2.5 External extended RAM, XRAM, XDATA

STC8G series microcomputers with 40 and above pins have the ability to expand 64KB of external data memory. During access to the external data memory, the WR/RD/ALE signal must be valid. The STC8G series MCUs have added a special function register BUS_SPEED to control the speed of the external 64K byte data bus. The description is as follows:

7.2.6 Bus speed control register (BUS_SPEED)

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

EXTRAM: on-chip extended RAM access control bit

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

1: On-chip extended RAM is disabled.

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

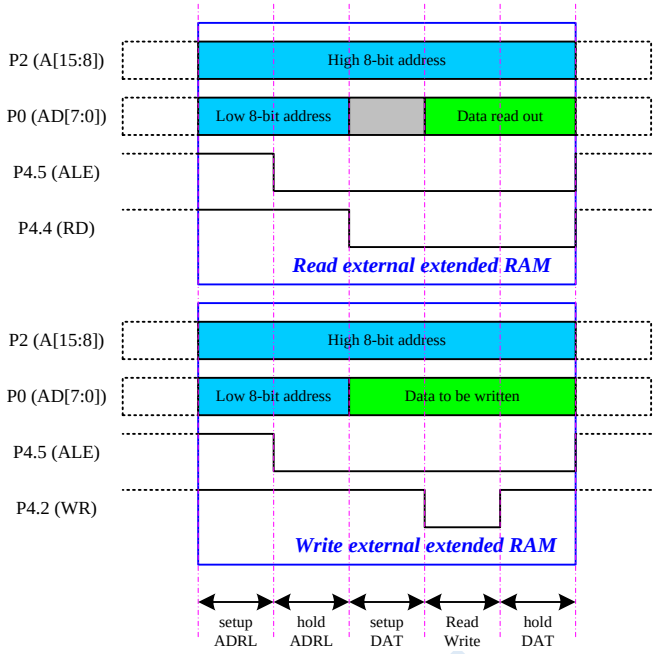
00: P4.4 is RD, P4.2 is WR.

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)

Instruction	Clocks	
	Access internal extended RAM	Access internal extended RAM
MOVX A,@Ri	3	3+5*(SPEED+1)
MOVX @Ri,A	3	3+5*(SPEED+1)
MOVX A,@DPTR	2	2+5*(SPEED+1)
MOVX @DPTR,A	2	2+5*(SPEED+1)

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



7.2.7 Bit Addressable Data Memory in 8051

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

Address of Data Memory	Bit-addressable address							
	B7	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0**
F8H (P7)	FFH	FEH	FDH	FCH	FBH	FAH	F9H	F8H
	F8H.7	F8H.6	F8H.5	F8H.4	F8H.3	F8H.2	F8H.1	F8H.0
F0H (B)	F7H	F6H	F5H	F4H	F3H	F2H	F1H	F0H
	F0H.7	F0H.6	F0H.5	F0H.4	F0H.3	F0H.2	F0H.1	F0H.0
E8H (P6)	EFH	EEH	EDH	ECH	EBH	EAH	E9H	E8H
	E8H.7	E8H.6	E8H.5	E8H.4	E8H.3	E8H.2	E8H.1	E8H.0
E0H (ACC)	E7H	E6H	E5H	E4H	E3H	E2H	E1H	E0H
	E0H.7	E0H.6	E0H.5	E0H.4	E0H.3	E0H.2	E0H.1	E0H.0
D8H (CCON)	DFH	DEH	DDH	DCH	DBH	DAH	D9H	D8H
	D8H.7	D8H.6	D8H.5	D8H.4	D8H.3	D8H.2	D8H.1	D8H.0
D0H (PSW)	D7H	D6H	D5H	D4H	D3H	D2H	D1H	D0H
	D0H.7	D0H.6	D0H.5	D0H.4	D0H.3	D0H.2	D0H.1	D0H.0
C8H (P5)	CFH	CEH	CDH	CCH	CBH	CAH	C9H	C8H
	C8H.7	C8H.6	C8H.5	C8H.4	C8H.3	C8H.2	C8H.1	C8H.0
C0H (P4)	C7H	C6H	C5H	C4H	C3H	C2H	C1H	C0H
	C0H.7	C0H.6	C0H.5	C0H.4	C0H.3	C0H.2	C0H.1	C0H.0
B8H (IP)	BFH	BEH	BDH	BCH	BBH	BAH	B9H	B8H
	B8H.7	B8H.6	B8H.5	B8H.4	B8H.3	B8H.2	B8H.1	B8H.0
B0H (P3)	B7H	B6H	B5H	B4H	B3H	B2H	B1H	B0H
	B0H.7	B0H.6	B0H.5	B0H.4	B0H.3	B0H.2	B0H.1	B0H.0
A8H (IE)	AFH	AEH	ADH	ACH	ABH	AAH	A9H	A8H
	A8H.7	A8H.6	A8H.5	A8H.4	A8H.3	A8H.2	A8H.1	A8H.0
A0H (P2)	A7H	A6H	A5H	A4H	A3H	A2H	A1H	A0H

98H (SCON)	A0H.7	A0H.6	A0H.5	A0H.4	A0H.3	A0H.2	A0H.1	A0H.0
	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H
	98H.7	98H.6	98H.5	98H.4	98H.3	98H.2	98H.1	98H.0
90H (P1)	97H	96H	95H	94H	93H	92H	91H	90H
	90H.7	90H.6	90H.5	90H.4	90H.3	90H.2	90H.1	90H.0
88H (TCON)	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H
	88H.7	88H.6	88H.5	88H.4	88H.3	88H.2	88H.1	88H.0
80H (P0)	87H	86H	85H	84H	83H	82H	81H	80H
	80H.7	80H.6	80H.5	80H.4	80H.3	80H.2	80H.1	80H.0
2FH	7FH	7EH	7DH	7CH	7BH	7AH	79H	78H
	2FH.7	2FH.6	2FH.5	2FH.4	2FH.3	2FH.2	2FH.1	2FH.0
2EH	77H	76H	75H	74H	73H	72H	71H	70H
	2EH.7	2EH.6	2EH.5	2EH.4	2EH.3	2EH.2	2EH.1	2EH.0
2DH	6FH	6EH	6DH	6CH	6BH	6AH	69H	68H
	2DH.7	2DH.6	2DH.5	2DH.4	2DH.3	2DH.2	2DH.1	2DH.0
2CH	67H	66H	65H	64H	63H	62H	61H	60H
	2CH.7	2CH.6	2CH.5	2CH.4	2CH.3	2CH.2	2CH.1	2CH.0
2BH	5FH	5EH	5DH	5CH	5BH	5AH	59H	58H
	2BH.7	2BH.6	2BH.5	2BH.4	2BH.3	2BH.2	2BH.1	2BH.0
2AH	57H	56H	55H	54H	53H	52H	51H	50H
	2AH.7	2AH.6	2AH.5	2AH.4	2AH.3	2AH.2	2AH.1	2AH.0
29H	4FH	4EH	4DH	4CH	4BH	4AH	49H	48H
	29H.7	29H.6	29H.5	29H.4	29H.3	29H.2	29H.1	29H.0
28H	47H	46H	45H	44H	43H	42H	41H	40H
	28H.7	28H.6	28H.5	28H.4	28H.3	28H.2	28H.1	28H.0
27H	3FH	3EH	3DH	3CH	3BH	3AH	39H	38H
	27H.7	27H.6	27H.5	27H.4	27H.3	27H.2	27H.1	27H.0
26H	37H	36H	35H	34H	33H	32H	31H	30H
	26H.7	26H.6	26H.5	26H.4	26H.3	26H.2	26H.1	26H.0
25H	2FH	2EH	2DH	2CH	2BH	2AH	29H	28H
	25H.7	25H.6	25H.5	25H.4	25H.3	25H.2	25H.1	25H.0
24H	27H	26H	25H	24H	23H	22H	21H	20H
	24H.7	24H.6	24H.5	24H.4	24H.3	24H.2	24H.1	24H.0
23H	1FH	1EH	1DH	1CH	1BH	1AH	19H	18H
	23H.7	23H.6	23H.5	23H.4	23H.3	23H.2	23H.1	23H.0
22H	17H	16H	15H	14H	13H	12H	11H	10H
	22H.7	22H.6	22H.5	22H.4	22H.3	22H.2	22H.1	22H.0
21H	0FH	0EH	0DH	0CH	0BH	0AH	09H	08H
	21H.7	21H.6	21H.5	21H.4	21H.3	21H.2	21H.1	21H.0
20H	07H	06H	05H	04H	03H	02H	01H	00H
	20H.7	20H.6	20H.5	20H.4	20H.3	20H.2	20H.1	20H.0

STC MCU

7.3 Special parameters of memory

The data memory and program memory of the STC8G series of microcontrollers store some special parameters related to the chip, including the global unique ID, the frequency of the 32K power-down wake-up timer, the internal 1.19V reference voltage value, and the IRC parameters.

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

Parameters	Addresses				Parameter Description
	STC8G1K04-8Pin STC8G1K04A STC8G1K04 STC8G1K04T	**STC8G1K08-8Pin STC8G1K08A STC8G1K08 STC8G1K08T**	**STC8G1K12-8Pin STC8G1K12A STC8G1K12 STC8G1K12T**	**STC8G1K17-8Pin STC8G1K17A STC8G1K17 STC8G1K17T**	
global unique ID	0FF9H~0FFFH	1FF9H~1FFFH	2FF9H~2FFFH	43F9H~43FFH	7 bytes
internal 1.19V reference voltage	0FF7H~0FF8H	1FF7H~1FF8H	2FF7H~2FF8H	43F7H~43F8H	mV (high byte first)
frequency of the 32K power-down wake-up timer	0FF5H~0FF6H	1FF5H~1FF6H	2FF5H~2FF6H	43F5H~43F6H	Hz (high byte first)
parameters of 22.1184MHz IRC	0FF4H	1FF4H	2FF4H	43F4H	-
parameters of 24MHz IRC	0FF3H	1FF3H	2FF3H	43F3H	-
parameters of 20MHz IRC	0FF2H	1FF3H	2FF2H	43F2H	Valid for the firmware version 7.3.12U or above
parameters of 27MHz IRC	0FF1H	1FF1H	2FF1H	43F1H	
parameters of 30MHz IRC	0FF0H	1FF0H	2FF0H	43F0H	
parameters of 33.1776MHz IRC	0FEFH	1FEFH	2FEFH	43EFH	
parameters of 35MHz IRC	0FEEH	1FEEH	2FEEH	43EEH	
parameters of 36.864MHz IRC	0FEDH	1FEDH	2FEDH	43EDH	
Reserved	0FECH	1FECH	2FECH	43ECH	
Reserved	0FEBH	1FEBH	2FEBH	43EBH	
VRTRIM parameters of 20MHz	0FEAH	1FEAH	2FEAH	43EAH	
VRTRIM parameters of 35MHz	0FE9H	1FE9H	2FE9H	43E9H	

Parameters	Addresses					Parameter Description
	STC8G2K16S2 STC8G2K16S4 STC15H2K16S4	**STC8G2K32S2 STC8G2K32S4 STC15H2K32S4**	**STC8G2K48S2 STC8G2K48S4 STC15H2K48S4**	**STC8G2K60S2 STC8G2K60S4 STC15H2K60S4**	**STC8G2K64S2 STC8G2K64S4 STC15H2K64S4**	
global unique ID	3FF9H~3FFFH	7FF9H~7FFFH	0BFF9H~0BFFFH	0EFF9H~0EFFFH	0FDF9H~0FDFFH	7 bytes
internal 1.19V reference voltage	3FF7H~3FF8H	7FF7H~7FF8H	0BFF7H~0BFF8H	0EFF7H~0EFF8H	0FDF7H~0FDF8H	mV (high byte first)
frequency of the 32K power-down wake-up timer	3FF5H~3FF6H	7FF5H~7FF6H	0BFF5H~0BFF6H	0EFF5H~0EFF6H	0FDF5H~0FDF6H	Hz (high byte first)
parameters of 22.1184MHz IRC	3FF4H	7FF4H	0BFF4H	0EFF4H	0FDF4H	-
parameters of 24MHz IRC	3FF3H	7FF3H	0BFF3H	0EFF3H	0FDF3H	-
parameters of 20MHz IRC	3FF2H	7FF2H	BFF2H	EFF2H	FDF2H	Valid for the firmware version 7.3.12U or above
parameters of 27MHz IRC	3FF1H	7FF1H	BFF1H	EFF1H	FDF1H	
parameters of 30MHz IRC	3FF0H	7FF0H	BFF0H	EFF0H	FDF0H	
parameters of 33.1776MHz IRC	3FEFH	7FEFH	BFEFH	EFEFH	FDEFH	
parameters of 35MHz IRC	3FEEH	7FEEH	BFEEH	EFEEH	FDEEH	
parameters of 36.864MHz IRC	3FEDH	7FEDH	BFEDH	EFEDH	FDEDH	
Reserved	3FECH	7FECH	BFECH	EFECH	FDECH	
Reserved	3FEBH	7FEBH	BFEBH	EFEBH	FDEBH	
VRTRIM parameters of 20MHz	3FEAH	7FEAH	BFEAH	EFEAH	FDEAH	
VRTRIM parameters of 35MHz	3FE9H	7FE9H	BFE9H	EFE9H	FDE9H	

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

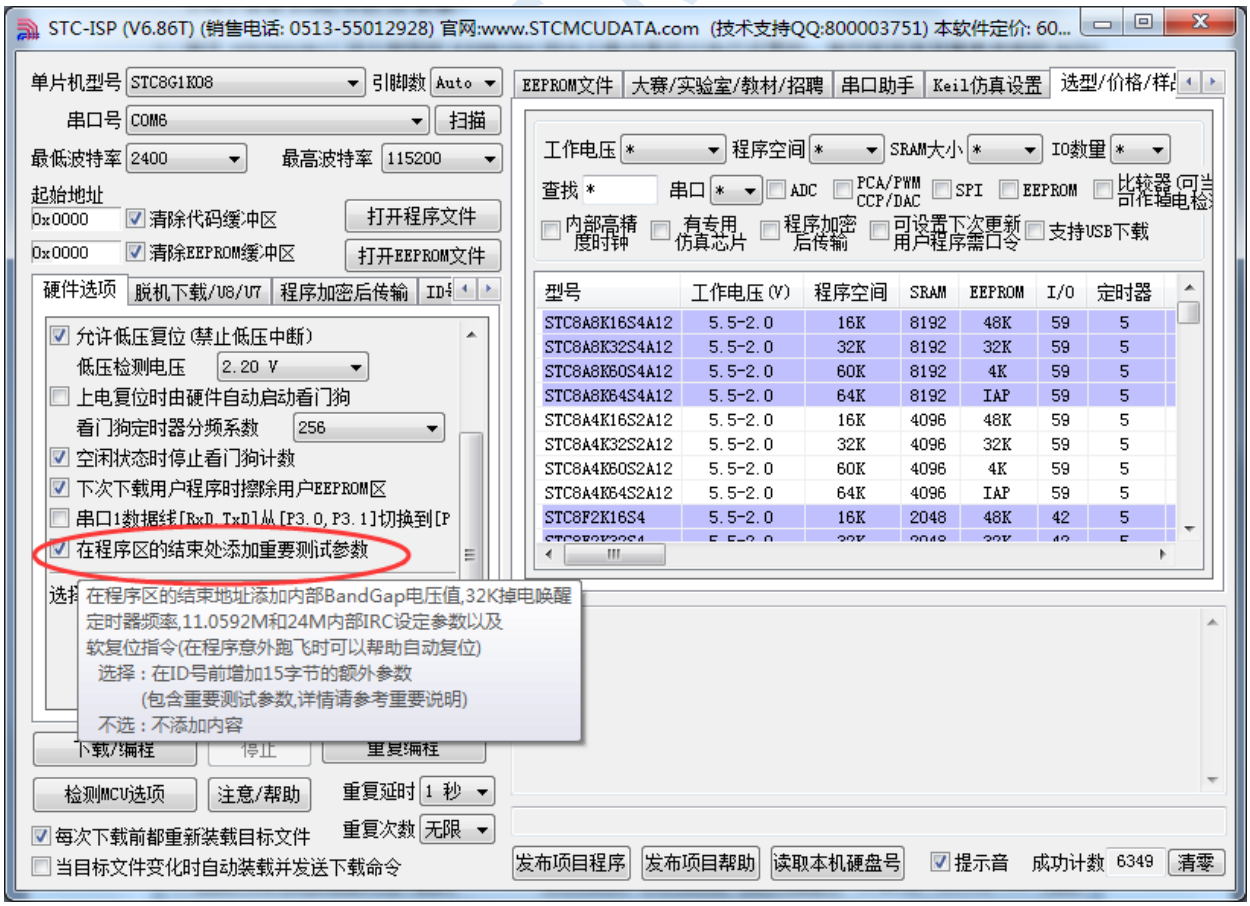
Parameters	addresses	**Parameter Description**
internal 1.19V reference voltage	idata：0EFH~0F0H	mV (high byte first)
global unique ID	idata：0F1H~0F7H	7 bytes
frequency of the 32K power-down wake-up timer	idata：0F8H~0F9H	Hz (high byte first)
parameters of 22.1184MHz IRC	idata：0FAH	-
parameters of 24MHz IRC	idata：0FBH	-

Special Note

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

2. Due to the size of EEPROM in STC8G1K12, STC8G1K12-8Pin, STC8G1K12A, STC8G1K12T-20Pin, STC8G1K17-20Pin/16Pin, STC8G1K17-8Pin, STC8G1K17A-8Pin, STC8G1K17T-20Pin, STC8G2K64S4, STC8G2K64S2 can be set by user, important parameters stored in the FLASH program memory (ROM) space may be erased or modified when the FLASH is used as EEPROM. So this issue needs to be considered when using these microcontrollers for ID number encryption.

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



7.3.1 Read Internal 1.19V Referrence Voltage (from Flash)

C language code

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

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

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

sfr       AUXR        =   0x8e;

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

bit       busy;
int       *BGV;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

void UartSend(char dat)
{
```

STC MCU

```
    while (busy);
    busy = 1;
    SBUF = dat;
}

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

    BGV = (int code *)0x1ff7;                   // STC8G1K08
    UartInit();
    ES = 1;
    EA = 1;
    UartSend(*BGV >> 8);                        //Read the high byte of the internal 1.19V reference voltage
    UartSend(*BGV);                             //Read the low byte of the internal 1.19V reference voltage

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH
BGV         EQU     01FF7H              ;STC8G1K08

BUSY        BIT     20H.0

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

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H
```

```
UART_ISR:
        JNB     TI,CHKRI
        CLR     TI
        CLR     BUSY
CHKRI:
        JNB     RI,UARTISR_EXIT
        CLR     RI
UARTISR_EXIT:
        RETI

UART_INIT:
        MOV     SCON,#50H
        MOV     TMOD,#00H
        MOV     TL1,#0E8H               ;65536-11059200/115200/4=0FFE8H
        MOV     TH1,#0FFH
        SETB    TR1
        MOV     AUXR,#40H
        CLR     BUSY
        RET

UART_SEND:
        JB      BUSY,$
        SETB    BUSY
        MOV     SBUF,A
        RET

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

        LCALL   UART_INIT
        SETB    ES
        SETB    EA

        MOV     DPTR,#BGV
        CLR     A
        MOVC    A,@A+DPTR               ;Read the high byte of the internal reference voltage
        LCALL   UART_SEND
        MOV     A,#1
        MOVC    A,@A+DPTR               ;Read the low byte of the internal reference voltage
        LCALL   UART_SEND

LOOP:
        JMP     LOOP

        END
```

STC MCU

~~7.3.2 Read Internal Referrence Voltage (from RAM)~~

C language code

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

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

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

sfr       AUXR        =   0x8e;

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

bit       busy;
int       *BGV;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

void UartSend(char dat)
{
```

STC MCU

```
    while (busy);
    busy = 1;
    SBUF = dat;
}

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

    P5M0 = 0x00;
    P5M1 = 0x00;

    BGV = (int idata *)0xef;
    UartInit();
    ES = 1;
    EA = 1;
    UartSend(*BGV >> 8);                //Read the high byte of the internal 1.19V reference voltage
    UartSend(*BGV);                     //Read the low byte of the internal 1.19V reference voltage

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH
BGV         DATA    0EFH

BUSY        BIT     20H.0

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

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
```

```
                JNB     TI,CHKRI
                CLR     TI
                CLR     BUSY
CHKRI:
                JNB     RI,UARTISR_EXIT
                CLR     RI
UARTISR_EXIT:
                RETI

UART_INIT:
                MOV     SCON,#50H
                MOV     TMOD,#00H
                MOV     TL1,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     TH1,#0FFH
                SETB    TR1
                MOV     AUXR,#40H
                CLR     BUSY
                RET

UART_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     SBUF,A
                RET

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

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     R0,#BGV
                MOV     A,@R0                   ;Read the high byte of the internal 1.19V reference voltage
                LCALL   UART_SEND
                INC     R0
                MOV     A,@R0                   ;Read the low byte of the internal 1.19V reference voltage
                LCALL   UART_SEND

LOOP:
                JMP     LOOP

                END
```

STC MCU

7.3.3 Read the Unique ID (from Flash)

C language code

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

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

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

sfr       AUXR        =   0x8e;

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

bit       busy;
char      *ID;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
```

STC MCU

```
    SBUF = dat;
}

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

    char i;

    ID = (char code *)0x1ff9;                    // STC8G1K08
    UartInit();
    ES = 1;
    EA = 1;

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

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH
ID          EQU     01FF9H              ; STC8G1K08

BUSY        BIT     20H.0

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

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR
```

```
                ORG     0100H

UART_ISR:
                JNB     TI,CHKRI
                CLR     TI
                CLR     BUSY
CHKRI:
                JNB     RI,UARTISR_EXIT
                CLR     RI
UARTISR_EXIT:
                RETI

UART_INIT:
                MOV     SCON,#50H
                MOV     TMOD,#00H
                MOV     TL1,#0E8H       ;65536-11059200/115200/4=0FFE8H
                MOV     TH1,#0FFH
                SETB    TR1
                MOV     AUXR,#40H
                CLR     BUSY
                RET

UART_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     SBUF,A
                RET

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

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#ID
                MOV     R1,#7
NEXT:           CLR     A
                MOVC    A,@A+DPTR
                LCALL   UART_SEND
                INC     DPTR
                DJNZ    R1,NEXT

LOOP:
                JMP     LOOP
```

STC MCU

END

7.3.4 Read the Unique ID (from RAM)

C language code

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

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

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

sfr       AUXR        =   0x8e;

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

bit       busy;
char      *ID;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}
```

STC MCU

```
void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

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

    char i;

    ID = (char idata *)0xf1;
    UartInit();
    ES = 1;
    EA = 1;

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

    while (1);
}
```

STC MCU

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA        8EH
ID          DATA        0F1H

BUSY        BIT         20H.0

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

```
                ORG     0000H
                LJMP    MAIN
                ORG     0023H
                LJMP    UART_ISR

                ORG     0100H

UART_ISR:
                JNB     TI,CHKRI
                CLR     TI
                CLR     BUSY
CHKRI:
                JNB     RI,UARTISR_EXIT
                CLR     RI
UARTISR_EXIT:
                RETI

UART_INIT:
                MOV     SCON,#50H
                MOV     TMOD,#00H
                MOV     TL1,#0E8H       ;65536-11059200/115200/4=0FFE8H
                MOV     TH1,#0FFH
                SETB    TR1
                MOV     AUXR,#40H
                CLR     BUSY
                RET

UART_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     SBUF,A
                RET

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

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     R0,#ID
                MOV     R1,#7
NEXT:           MOV     A,@R0
                LCALL   UART_SEND
                INC     R0
                DJNZ    R1,NEXT
```

STC MCU

```
LOOP:
        JMP     LOOP

        END
```

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

C language code

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

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

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

sfr     AUXR    =   0x8e;

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

bit     busy;
int     *F32K;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
```

STC MCU

```
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

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

    F32K = (int code *)0x1ff5;                  // STC8G1K08
    UartInit();
    ES = 1;
    EA = 1;

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

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH
F32K        EQU     01FF5H          ; STC8G1K08

BUSY        BIT     20H.0

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

```
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
UARTISR_EXIT:
            RETI

UART_INIT:
            MOV     SCON,#50H
            MOV     TMOD,#00H
            MOV     TL1,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     TH1,#0FFH
            SETB    TR1
            MOV     AUXR,#40H
            CLR     BUSY
            RET

UART_SEND:
            JB      BUSY,$
            SETB    BUSY
            MOV     SBUF,A
            RET

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

            LCALL   UART_INIT
            SETB    ES
            SETB    EA

            MOV     DPTR,#F32K
            CLR     A
            MOVC    A,@A+DPTR           ;Read high byte of 32K frequency
            LCALL   UART_SEND
            INC     DPTR
```

STC MCU

```
                CLR     A
                MOVC    A,@A+DPTR               ;Read low byte of 32K frequency
                LCALL   UART_SEND

LOOP:
                JMP     LOOP

                END
```

~~7.3.6 Read the Frequency of 32K Power-down Wake-up Timer (from RAM)~~

C language code

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

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

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

sfr AUXR = 0x8e;

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

bit busy;
int *F32K;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
```

```
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

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

    F32K = (int idata *)0xf8;
    UartInit();
    ES = 1;
    EA = 1;

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

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA        8EH
F32K        DATA        0F8H

BUSY        BIT         20H.0

P0M1        DATA        093H
P0M0        DATA        094H
P1M1        DATA        091H
P1M0        DATA        092H
P2M1        DATA        095H
P2M0        DATA        096H
P3M1        DATA        0B1H
```

```
P3M0            DATA        0B2H
P4M1            DATA        0B3H
P4M0            DATA        0B4H
P5M1            DATA        0C9H
P5M0            DATA        0CAH

                ORG         0000H
                LJMP        MAIN
                ORG         0023H
                LJMP        UART_ISR

                ORG         0100H

UART_ISR:
                JNB         TI,CHKRI
                CLR         TI
                CLR         BUSY
CHKRI:
                JNB         RI,UARTISR_EXIT
                CLR         RI
UARTISR_EXIT:
                RETI

UART_INIT:
                MOV         SCON,#50H
                MOV         TMOD,#00H
                MOV         TL1,#0E8H           ;65536-11059200/115200/4=0FFE8H
                MOV         TH1,#0FFH
                SETB        TR1
                MOV         AUXR,#40H
                CLR         BUSY
                RET

UART_SEND:
                JB          BUSY,$
                SETB        BUSY
                MOV         SBUF,A
                RET

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

                LCALL       UART_INIT
                SETB        ES
                SETB        EA

                MOV         R0,#F32K
```

STC MCU

```
                MOV     A,@R0                   ;Read high byte of 32K frequency
                LCALL   UART_SEND
                INC     R0
                MOV     A,@R0                   ;Read low byte of 32K frequency
                LCALL   UART_SEND

LOOP:
                JMP     LOOP

                END
```

7.3.7 Read the User-defined IRC Frequency (from Flash)

C language code

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

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

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

; The following table is the parameter list of STC8G1K08
#define   ID_ROMADDR          (*(unsigned char volatile xdata *)0x1ff9)
#define   VREF_ROMADDR        (*(unsigned char volatile xdata *)0x1ff7)
#define   F32K_ROMADDR        (*(unsigned char volatile xdata *)0x1ff5)
#define   T22M_ROMADDR        (*(unsigned char volatile xdata *)0x1ff4)//22.1184MHz
#define   T24M_ROMADDR        (*(unsigned char volatile xdata *)0x1ff3)//24MHz
#define   T20M_ROMADDR        (*(unsigned char volatile xdata *)0x1ff2)//20MHz
#define   T27M_ROMADDR        (*(unsigned char volatile xdata *)0x1ff1)//27MHz
#define   T30M_ROMADDR        (*(unsigned char volatile xdata *)0x1ff0)    //30MHz
#define   T33M_ROMADDR        (*(unsigned char volatile xdata *)0x1fef)    //33.1776MHz
#define   T35M_ROMADDR        (*(unsigned char volatile xdata *)0x1fee)    //35MHz
#define   T36M_ROMADDR        (*(unsigned char volatile xdata *)0x1fed)    //36.864MHz
#define   VRT20M_ROMADDR      (*(unsigned char volatile xdata *)0x1fea)    //VRTRIM_20M
#define   VRT35M_ROMADDR      (*(unsigned char volatile xdata *)0x1fe9)    //VRTRIM_35M


sfr     P_SW2       =   0xba;
sfr     IRCBAND     =   0x9d;
sfr     IRTRIM      =   0x9f;
sfr     VRTRIM      =   0xa6;

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

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

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

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

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

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

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

//  //Select 33.1776MHz
//  P_SW2=0x80;
//  CLKDIV=0x04;
```

STC MCU

```
//  IRTRIM= T33M_ROMADDR;
//  VRTRIM= VRT35M_ROMADDR;
//  IRCBAND=0x01;
//  CLKDIV=0x00;

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

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

; The following table is the parameter list of STC8G1K08
ID_ROMADDR          EQU   01FF9H
VREF_ROMADDR        EQU   01FF7H
F32K_ROMADDR        EQU   01FF5H
T22M_ROMADDR        EQU   01FF4H          ;22.1184MHz
T24M_ROMADDR        EQU   01FF3H          ;24MHz
T20M_ROMADDR        EQU   01FF2H          ;20MHz
T27M_ROMADDR        EQU   01FF1H          ;27MHz
T30M_ROMADDR        EQU   01FF0H          ;30MHz
T33M_ROMADDR        EQU   01FEFH          ;33.1776MHz
T35M_ROMADDR        EQU   01FEEH          ;35MHz
T36M_ROMADDR        EQU   01FEDH          ;36.864MHz
VRT20M_ROMADDR      EQU   01FEAH          ;VRTRIM_20M
VRT35M_ROMADDR      EQU   01FE9H          ;VRTRIM_35M

P_SW2       DATA    0BAH
CKSEL       EQU     0FE00H
CLKDIV      EQU     0FE01H

IRCBAND     DATA    09DH
IRCTRIM     DATA    09FH
VRTRIM      DATA    0A6H

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

            ORG     0000H
            LJMP    MAIN
```

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

;           ;Select 20MHz
;           MOV     P_SW2,#80H
;           MOV     A,#4
;           MOV     DPTR,#CLKDIV
;           MOV     DPTR,#T20M_ROMADDR
;           CLR     A
;           MOVC    A,@A+DPTR
;           MOV     IRTRIM,A
;           MOV     DPTR,#VRT20M_ROMADDR
;           CLR     A
;           MOVC    A,@A+DPTR
;           MOV     VRTRIM,A
;           MOV     IRCBAND,#00H
;           MOV     A,#0
;           MOV     DPTR,#CLKDIV
;           MOV     P_SW2,#00H

;           ;Select 22.1184MHz
;           MOV     P_SW2,#80H
;           MOV     A,#4
;           MOV     DPTR,#CLKDIV
;           MOV     DPTR,#T22M_ROMADDR
;           CLR     A
;           MOVC    A,@A+DPTR
;           MOV     IRTRIM,A
;           MOV     DPTR,#VRT20M_ROMADDR
;           CLR     A
;           MOVC    A,@A+DPTR
;           MOV     VRTRIM,A
;           MOV     IRCBAND,#00H
;           MOV     A,#0
;           MOV     DPTR,#CLKDIV
;           MOV     P_SW2,#00H

            ;Select 24MHz
            MOV     P_SW2,#80H
            MOV     A,#4
            MOV     DPTR,#CLKDIV
            MOV     DPTR,#T24M_ROMADDR
            CLR     A
            MOVC    A,@A+DPTR
            MOV     IRTRIM,A
            MOV     DPTR,#VRT20M_ROMADDR
```

STC MCU

```
                CLR     A
                MOVC    A,@A+DPTR
                MOV     VRTRIM,A
                MOV     IRCBAND,#00H
                MOV     A,#0
                MOV     DPTR,#CLKDIV
                MOV     P_SW2,#00H

;               ;Select 27MHz
;               MOV     P_SW2,#80H
;               MOV     A,#4
;               MOV     DPTR,#CLKDIV
;               MOV     DPTR,#T27M_ROMADDR
;               CLR     A
;               MOVC    A,@A+DPTR
;               MOV     IRTRIM,A
;               MOV     DPTR,#VRT35M_ROMADDR
;               CLR     A
;               MOVC    A,@A+DPTR
;               MOV     VRTRIM,A
;               MOV     IRCBAND,#01H
;               MOV     A,#0
;               MOV     DPTR,#CLKDIV
;               MOV     P_SW2,#00H

;               ;Select 30MHz
;               MOV     P_SW2,#80H
;               MOV     A,#4
;               MOV     DPTR,#CLKDIV
;               MOV     DPTR,#T30M_ROMADDR
;               CLR     A
;               MOVC    A,@A+DPTR
;               MOV     IRTRIM,A
;               MOV     DPTR,#VRT35M_ROMADDR
;               CLR     A
;               MOVC    A,@A+DPTR
;               MOV     VRTRIM,A
;               MOV     IRCBAND,#01H
;               MOV     A,#0
;               MOV     DPTR,#CLKDIV
;               MOV     P_SW2,#00H

;               ;Select 33.1776MHz
;               MOV     P_SW2,#80H
;               MOV     A,#4
;               MOV     DPTR,#CLKDIV
;               MOV     DPTR,#T33M_ROMADDR
;               CLR     A
;               MOVC    A,@A+DPTR
;               MOV     IRTRIM,A
;               MOV     DPTR,#VRT35M_ROMADDR
;               CLR     A
;               MOVC    A,@A+DPTR
;               MOV     VRTRIM,A
;               MOV     IRCBAND,#01H
;               MOV     A,#0
;               MOV     DPTR,#CLKDIV
;               MOV     P_SW2,#00H
```

STC MCU

```
;               ;Select 35MHz
;               MOV         P_SW2,#80H
;               MOV         A,#4
;               MOV         DPTR,#CLKDIV
;               MOV         DPTR,#T35M_ROMADDR
;               CLR         A
;               MOVC        A,@A+DPTR
;               MOV         IRTRIM,A
;               MOV         DPTR,#VRT35M_ROMADDR
;               CLR         A
;               MOVC        A,@A+DPTR
;               MOV         VRTRIM,A
;               MOV         IRCBAND,#01H
;               MOV         A,#0
;               MOV         DPTR,#CLKDIV
;               MOV         P_SW2,#00H

;               ;Select 36.864MHz
;               MOV         P_SW2,#80H
;               MOV         A,#4
;               MOV         DPTR,#CLKDIV
;               MOV         DPTR,#T36M_ROMADDR
;               CLR         A
;               MOVC        A,@A+DPTR
;               MOV         IRTRIM,A
;               MOV         DPTR,#VRT35M_ROMADDR
;               CLR         A
;               MOVC        A,@A+DPTR
;               MOV         VRTRIM,A
;               MOV         IRCBAND,#01H
;               MOV         A,#0
;               MOV         DPTR,#CLKDIV
;               MOV         P_SW2,#00H

                JMP         $

                END
```

7.3.8 Read the User-defined IRC Frequency (from RAM)

C language code

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

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

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

sfr         P_SW2       =   0xba;
sfr         IRTRIM      =   0x9f;

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

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

char    *IRC22M;
char    *IRC24M;

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

    IRC22M = (char idata *)0xfa;
    IRC24M = (char idata *) 0xfb;
//  IRTRIM = *IRC22M;                   //Load 22.1184MHz IRC parameters
    IRTRIM = *IRC24M;                   //Load 24MHz IRC parameters

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

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH
CLKDIV      EQU     0FE01H

IRTRIM      DATA    09FH

IRC22M      DATA    0FAH
IRC24M      DATA    0FBH

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
```

```
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

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

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

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

            JMP     $

            END
```

STC MCU

8 Special Function Registers

8.1 STC8G1K08 family

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H		CH	CCAP0H	CCAP1H	CCAP2H			RSTCFG
F0H	B		PCA_PWM0	PCA_PWM1	PCA_PWM2	IAP_TPS		
E8H		CL	CCAP0L	CCAP1L	CCAP2L			AUXINTIF
E0H	ACC			DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2		ADCCFG	
D0H	PSW						T2H	T2L
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H		WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2		ADC_CONTR	ADC_RES	ADC_RESL	
B0H	P3	P3M1	P3M0			IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH			TA	IE2
A0H			P_SW1					
98H	SCON	SBUF	S2CON	S2BUF		IRCBAND	LIRTRIM	IRTRIM
90H	P1	P1M1	P1M0					
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H		SP	DPL	DPH				PCON

Bit addressable

Non bit addressable

Note: Bit addressing can only be carried out if the register address is divisible by 8,while the bit addressing is not available if the register address is not divisible by 8.

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FEA8H	ADCTIM							
FEA0H			TM2PS					
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE30H		P1IE		P3IE				
FE28H		P1DR		P3DR		P5DR		
FE20H		P1SR		P3SR		P5SR		
FE18H		P1NCS		P3NCS		P5NCS		
FE10H		P1PU		P3PU		P5PU		
FE00H	CKSEL	CLKDIV	HIRCCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDB	

STCMCU

8.2 STC8G1K08-8Pin family

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H								RSTCFG
F0H	B					IAP_TPS		
E8H								AUXINTIF
E0H	ACC			DPS	DPL1	DPH1		
D8H								
D0H	PSW							
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H		WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2					
B0H	P3	P3M1	P3M0			IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH			TA	IE2
A0H			P_SW1					
98H	SCON	SBUF				IRCBAND	LIRTRIM	IRTRIM
90H								
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H		SP	DPL	DPH				PCON

↑ Bit addressable

Non bit addressable

Note: Bit addressing can only be carried out if the register address is divisible by 8,while the bit addressing is not available if the register address is not divisible by 8.

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FCF0H	MD3	MD2	MD1	MD0	MD5	MD4	ARCON	OPCON
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE28H				P3DR		P5DR		
FE20H				P3SR		P5SR		
FE18H				P3NCS		P5NCS		
FE10H				P3PU		P5PU		
FE00H	CKSEL	CLKDIV	HIRCCR		IRC32KCR	MCLKOCR	IRCDB	

8.3 STC8G1K08A family

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H		CH	CCAP0H	CCAP1H	CCAP2H			RSTCFG
F0H	B		PCA_PWM0	PCA_PWM1	PCA_PWM2	IAP_TPS		
E8H		CL	CCAP0L	CCAP1L	CCAP2L			AUXINTIF
E0H	ACC			DPS	DPL1	DPH1		
D8H	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2		ADCCFG	
D0H	PSW							
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H		WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2		ADC_CONTR	ADC_RES	ADC_RESL	
B0H	P3	P3M1	P3M0			IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH			TA	IE2
A0H			P_SW1					
98H	SCON	SBUF				IRCBAND	LIRTRIM	IRTRIM
90H								
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H		SP	DPL	DPH				PCON

Bit addressable

Non bit addressable

Note: Bit addressing can only be carried out if the register address is divisible by 8,while the bit addressing is not available if the register address is not divisible by 8.

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FCF0H	MD3	MD2	MD1	MD0	MD5	MD4	ARCON	OPCON
FEA8H	ADCTIM							
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE30H				P3IE		P5IE		
FE28H				P3DR		P5DR		
FE20H				P3SR		P5SR		
FE18H				P3NCS		P5NCS		
FE10H				P3PU		P5PU		
FE00H	CKSEL	CLKDIV	HIRCCR		IRC32KCR	MCLKOCR	IRCDB	

STCMCU

8.4 STC8G2K64S4 family

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H		CH	CCAP0H	CCAP1H	CCAP2H		PWMCFG45	RSTCFG
F0H	B	PWMSET	PCA_PWM0	PCA_PWM1	PCA_PWM2	IAP_TPS	PWMCFG01	PWMCFG23
E8H		CL	CCAP0L	CCAP1L	CCAP2L		IP3H	AUXINTIF
E0H	ACC			DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2		ADCCFG	IP3
D0H	PSW	T4T3M	TH4	TL4	TH3	TL3	T2H	T2L
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H	P4	WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2		ADC_CONTR	ADC_RES	ADC_RESL	
B0H	P3	P3M1	P3M0	P4M1	P4M0	IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH	S3CON	S3BUF	TA	IE2
A0H	P2		P_SW1					
98H	SCON	SBUF	S2CON	S2BUF		IRCBAND	LIRTRIM	IRTRIM
90H	P1	P1M1	P1M0	P0M1	P0M0	P2M1	P2M0	
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H	P0	SP	DPL	DPH	S4CON	S4BUF		PCON

Bit addressable

Non bit addressable

Note: Bit addressing can only be carried out if the register address is divisible by 8,while the bit addressing is not available if the register address is not divisible by 8.

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FCF0H	MD3	MD2	MD1	MD0	MD5	MD4	ARCON	OPCON
FFE8H	PWM27T1H	PWM27T1L	PWM27T2H	PWM27T2L	PWM27CR	PWM27HLD		
FFE0H	PWM26T1H	PWM26T1L	PWM26T2H	PWM26T2L	PWM26CR	PWM26HLD		
FFD8H	PWM25T1H	PWM25T1L	PWM25T2H	PWM25T2L	PWM25CR	PWM25HLD		
FFD0H	PWM24T1H	PWM24T1L	PWM24T2H	PWM24T2L	PWM24CR	PWM24HLD		
FFC8H	PWM23T1H	PWM23T1L	PWM23T2H	PWM23T2L	PWM31CR	PWM23HLD		
FFC0H	PWM22T1H	PWM22T1L	PWM22T2H	PWM22T2L	PWM22CR	PWM22HLD		
FFB8H	PWM21T1H	PWM21T1L	PWM21T2H	PWM21T2L	PWM21CR	PWM21HLD		
FFB0H	PWM20T1H	PWM20T1L	PWM20T2H	PWM20T2L	PWM20CR	PWM20HLD		
FFA0H	PWM2CH	PWM2CL	PWM2CKS	PWM2TADCH	PWM2TADCL	PWM2IF	PWM2FDCR	
FF98H	PWM17T1H	PWM17T1L	PWM17T2H	PWM17T2L	PWM17CR	PWM17HLD		
FF90H	PWM16T1H	PWM16T1L	PWM16T2H	PWM16T2L	PWM16CR	PWM16HLD		
FF88H	PWM15T1H	PWM15T1L	PWM15T2H	PWM15T2L	PWM15CR	PWM15HLD		
FF80H	PWM14T1H	PWM14T1L	PWM14T2H	PWM14T2L	PWM14CR	PWM14HLD		
FF78H	PWM13T1H	PWM13T1L	PWM13T2H	PWM13T2L	PWM31CR	PWM13HLD		
FF70H	PWM12T1H	PWM12T1L	PWM12T2H	PWM12T2L	PWM12CR	PWM12HLD		
FF68H	PWM11T1H	PWM11T1L	PWM11T2H	PWM11T2L	PWM11CR	PWM11HLD		
FF60H	PWM10T1H	PWM10T1L	PWM10T2H	PWM10T2L	PWM10CR	PWM10HLD		
FF50H	PWM1CH	PWM1CL	PWM1CKS			PWM1IF	PWM1FDCR	
FF48H	PWM07T1H	PWM07T1L	PWM07T2H	PWM07T2L	PWM07CR	PWM07HLD		
FF40H	PWM06T1H	PWM06T1L	PWM06T2H	PWM06T2L	PWM06CR	PWM06HLD		
FF38H	PWM05T1H	PWM05T1L	PWM05T2H	PWM05T2L	PWM05CR	PWM05HLD		
FF30H	PWM04T1H	PWM04T1L	PWM04T2H	PWM04T2L	PWM04CR	PWM04HLD		
FF28H	PWM03T1H	PWM03T1L	PWM03T2H	PWM03T2L	PWM31CR	PWM03HLD		
FF20H	PWM02T1H	PWM02T1L	PWM02T2H	PWM02T2L	PWM02CR	PWM02HLD		
FF18H	PWM01T1H	PWM01T1L	PWM01T2H	PWM01T2L	PWM01CR	PWM01HLD		
FF10H	PWM00T1H	PWM00T1L	PWM00T2H	PWM00T2L	PWM00CR	PWM00HLD		
FF00H	PWM0CH	PWM0CL	PWM0CKS	PWM0TADCH	PWM0TADCL	PWM0IF	PWM0FDCR	
FEA8H	ADCTIM							
FEA0H			TM2PS	TM3PS	TM4PS			
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE30H	P0IE	P1IE						
FE28H	P0DR	P1DR	P2DR	P3DR	P4DR	P5DR		
FE20H	P0SR	P1SR	P2SR	P3SR	P4SR	P5SR		
FE18H	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS		

FE10H	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU		
FE00H	CKSEL	CLKDIV	HIRCCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDB	
FCE8H	PWM57T1H	PWM57T1L	PWM57T2H	PWM57T2L	PWM57CR	PWM57HLD		
FCE0H	PWM56T1H	PWM56T1L	PWM56T2H	PWM56T2L	PWM56CR	PWM56HLD		
FCD8H	PWM55T1H	PWM55T1L	PWM55T2H	PWM55T2L	PWM55CR	PWM55HLD		
FCD0H	PWM54T1H	PWM54T1L	PWM54T2H	PWM54T2L	PWM54CR	PWM54HLD		
FCC8H	PWM53T1H	PWM53T1L	PWM53T2H	PWM53T2L	PWM31CR	PWM53HLD		
FCC0H	PWM52T1H	PWM52T1L	PWM52T2H	PWM52T2L	PWM52CR	PWM52HLD		
FCB8H	PWM51T1H	PWM51T1L	PWM51T2H	PWM51T2L	PWM51CR	PWM51HLD		
FCB0H	PWM50T1H	PWM50T1L	PWM50T2H	PWM50T2L	PWM50CR	PWM50HLD		
FCA0H	PWM5CH	PWM5CL	PWM5CKS			PWM5IF	PWM5FDCR	
FC98H	PWM47T1H	PWM47T1L	PWM47T2H	PWM47T2L	PWM47CR	PWM47HLD		
FC90H	PWM46T1H	PWM46T1L	PWM46T2H	PWM46T2L	PWM46CR	PWM46HLD		
FC88H	PWM45T1H	PWM45T1L	PWM45T2H	PWM45T2L	PWM45CR	PWM45HLD		
FC80H	PWM44T1H	PWM44T1L	PWM44T2H	PWM44T2L	PWM44CR	PWM44HLD		
FC78H	PWM43T1H	PWM43T1L	PWM43T2H	PWM43T2L	PWM31CR	PWM43HLD		
FC70H	PWM42T1H	PWM42T1L	PWM42T2H	PWM42T2L	PWM42CR	PWM42HLD		
FC68H	PWM41T1H	PWM41T1L	PWM41T2H	PWM41T2L	PWM41CR	PWM41HLD		
FC60H	PWM40T1H	PWM40T1L	PWM40T2H	PWM40T2L	PWM40CR	PWM40HLD		
FC50H	PWM4CH	PWM4CL	PWM4CKS	PWM4TADCH	PWM4TADCL	PWM4IF	PWM4FDCR	
FC48H	PWM37T1H	PWM37T1L	PWM37T2H	PWM37T2L	PWM37CR	PWM37HLD		
FC40H	PWM36T1H	PWM36T1L	PWM36T2H	PWM36T2L	PWM36CR	PWM36HLD		
FC38H	PWM35T1H	PWM35T1L	PWM35T2H	PWM35T2L	PWM35CR	PWM35HLD		
FC30H	PWM34T1H	PWM34T1L	PWM34T2H	PWM34T2L	PWM34CR	PWM34HLD		
FC28H	PWM33T1H	PWM33T1L	PWM33T2H	PWM33T2L	PWM31CR	PWM33HLD		
FC20H	PWM32T1H	PWM32T1L	PWM32T2H	PWM32T2L	PWM32CR	PWM32HLD		
FC18H	PWM31T1H	PWM31T1L	PWM31T2H	PWM31T2L	PWM31CR	PWM31HLD		
FC10H	PWM30T1H	PWM30T1L	PWM30T2H	PWM30T2L	PWM30CR	PWM30HLD		
FC00H	PWM3CH	PWM3CL	PWM3CKS			PWM3IF	PWM3FDCR	

STC MCU

8.5 STC8G2K64S2 family

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H		CH	CCAP0H	CCAP1H	CCAP2H		~~PWMCFG45~~	RSTCFG
F0H	B	~~PWMSET~~	PCA_PWM0	PCA_PWM1	PCA_PWM2	IAP_TPS	~~PWMCFG01~~	PWMCFG23
E8H		CL	CCAP0L	CCAP1L	CCAP2L		IP3H	AUXINTIF
E0H	ACC			DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2		ADCCFG	IP3
D0H	PSW	T4T3M	TH4	TL4	TH3	TL3	T2H	T2L
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H	P4	WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2		ADC_CONTR	ADC_RES	ADC_RESL	
B0H	P3	P3M1	P3M0	P4M1	P4M0	IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH			TA	IE2
A0H	P2		P_SW1					
98H	SCON	SBUF	S2CON	S2BUF		IRCBAND	LIRTRIM	IRTRIM
90H	P1	P1M1	P1M0	P0M1	P0M0	P2M1	P2M0	
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H	P0	SP	DPL	DPH				PCON

Bit addressable

Non bit addressable

Note: Bit addressing can only be carried out if the register address is divisible by 8, while the bit addressing is not available if the register address is not divisible by 8.

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FCF0H	MD3	MD2	MD1	MD0	MD5	MD4	ARCON	OPCON
FFE8H	PWM27T1H	PWM27T1L	PWM27T2H	PWM27T2L	PWM27CR	PWM27HLD		
FFE0H	PWM26T1H	PWM26T1L	PWM26T2H	PWM26T2L	PWM26CR	PWM26HLD		
FFD8H	PWM25T1H	PWM25T1L	PWM25T2H	PWM25T2L	PWM25CR	PWM25HLD		
FFD0H	PWM24T1H	PWM24T1L	PWM24T2H	PWM24T2L	PWM24CR	PWM24HLD		
FFC8H	PWM23T1H	PWM23T1L	PWM23T2H	PWM23T2L	PWM31CR	PWM23HLD		
FFC0H	PWM22T1H	PWM22T1L	PWM22T2H	PWM22T2L	PWM22CR	PWM22HLD		
FFB8H	PWM21T1H	PWM21T1L	PWM21T2H	PWM21T2L	PWM21CR	PWM21HLD		
FFB0H	PWM20T1H	PWM20T1L	PWM20T2H	PWM20T2L	PWM20CR	PWM20HLD		
FFA0H	PWM2CH	PWM2CL	PWM2CKS	PWM2TADCH	PWM2TADCL	PWM2IF	PWM2FDCR	
FF98H	~~PWM17T1H~~	~~PWM17T1L~~	~~PWM17T2H~~	~~PWM17T2L~~	~~PWM17CR~~	~~PWM17HLD~~		
FF90H	~~PWM16T1H~~	~~PWM16T1L~~	~~PWM16T2H~~	~~PWM16T2L~~	~~PWM16CR~~	~~PWM16HLD~~		
FF88H	~~PWM15T1H~~	~~PWM15T1L~~	~~PWM15T2H~~	~~PWM15T2L~~	~~PWM15CR~~	~~PWM15HLD~~		
FF80H	~~PWM14T1H~~	~~PWM14T1L~~	~~PWM14T2H~~	~~PWM14T2L~~	~~PWM14CR~~	~~PWM14HLD~~		
FF78H	~~PWM13T1H~~	~~PWM13T1L~~	~~PWM13T2H~~	~~PWM13T2L~~	~~PWM31CR~~	~~PWM13HLD~~		
FF70H	~~PWM12T1H~~	~~PWM12T1L~~	~~PWM12T2H~~	~~PWM12T2L~~	~~PWM12CR~~	~~PWM12HLD~~		
FF68H	~~PWM11T1H~~	~~PWM11T1L~~	~~PWM11T2H~~	~~PWM11T2L~~	~~PWM11CR~~	~~PWM11HLD~~		
FF60H	~~PWM10T1H~~	~~PWM10T1L~~	~~PWM10T2H~~	~~PWM10T2L~~	~~PWM10CR~~	~~PWM10HLD~~		
FF50H	~~PWM1CH~~	~~PWM1CL~~	~~PWM1CKS~~			~~PWM1IF~~	~~PWM1FDCR~~	
FF48H	~~PWM07T1H~~	~~PWM07T1L~~	~~PWM07T2H~~	~~PWM07T2L~~	~~PWM07CR~~	~~PWM07HLD~~		
FF40H	~~PWM06T1H~~	~~PWM06T1L~~	~~PWM06T2H~~	~~PWM06T2L~~	~~PWM06CR~~	~~PWM06HLD~~		
FF38H	~~PWM05T1H~~	~~PWM05T1L~~	~~PWM05T2H~~	~~PWM05T2L~~	~~PWM05CR~~	~~PWM05HLD~~		
FF30H	~~PWM04T1H~~	~~PWM04T1L~~	~~PWM04T2H~~	~~PWM04T2L~~	~~PWM04CR~~	~~PWM04HLD~~		
FF28H	~~PWM03T1H~~	~~PWM03T1L~~	~~PWM03T2H~~	~~PWM03T2L~~	~~PWM31CR~~	~~PWM03HLD~~		
FF20H	~~PWM02T1H~~	~~PWM02T1L~~	~~PWM02T2H~~	~~PWM02T2L~~	~~PWM02CR~~	~~PWM02HLD~~		
FF18H	~~PWM01T1H~~	~~PWM01T1L~~	~~PWM01T2H~~	~~PWM01T2L~~	~~PWM01CR~~	~~PWM01HLD~~		
FF10H	~~PWM00T1H~~	~~PWM00T1L~~	~~PWM00T2H~~	~~PWM00T2L~~	~~PWM00CR~~	~~PWM00HLD~~		
FF00H	~~PWM0CH~~	~~PWM0CL~~	~~PWM0CKS~~	~~PWM0TADCH~~	~~PWM0TADCL~~	~~PWM0IF~~	~~PWM0FDCR~~	
FEA8H	ADCTIM							
FEA0H			TM2PS	TM3PS	TM4PS			
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE30H	P0IE	P1IE						
FE28H	P0DR	P1DR	P2DR	P3DR	P4DR	P5DR		
FE20H	P0SR	P1SR	P2SR	P3SR	P4SR	P5SR		
FE18H	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS		

FE10H	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU		
FE00H	CKSEL	CLKDIV	IRC24MCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDB	
FCE8H	~~PWM57T1H~~	~~PWM57T1L~~	~~PWM57T2H~~	~~PWM57T2L~~	~~PWM57CR~~	~~PWM57HLD~~		
FCE0H	~~PWM56T1H~~	~~PWM56T1L~~	~~PWM56T2H~~	~~PWM56T2L~~	~~PWM56CR~~	~~PWM56HLD~~		
FCD8H	~~PWM55T1H~~	~~PWM55T1L~~	~~PWM55T2H~~	~~PWM55T2L~~	~~PWM55CR~~	~~PWM55HLD~~		
FCD0H	~~PWM54T1H~~	~~PWM54T1L~~	~~PWM54T2H~~	~~PWM54T2L~~	~~PWM54CR~~	~~PWM54HLD~~		
FCC8H	~~PWM53T1H~~	~~PWM53T1L~~	~~PWM53T2H~~	~~PWM53T2L~~	~~PWM31CR~~	~~PWM53HLD~~		
FCC0H	~~PWM52T1H~~	~~PWM52T1L~~	~~PWM52T2H~~	~~PWM52T2L~~	~~PWM52CR~~	~~PWM52HLD~~		
FCB8H	~~PWM51T1H~~	~~PWM51T1L~~	~~PWM51T2H~~	~~PWM51T2L~~	~~PWM51CR~~	~~PWM51HLD~~		
FCB0H	~~PWM50T1H~~	~~PWM50T1L~~	~~PWM50T2H~~	~~PWM50T2L~~	~~PWM50CR~~	~~PWM50HLD~~		
FCA0H	~~PWM5CH~~	~~PWM5CL~~	~~PWM5CKS~~			~~PWM5IF~~	~~PWM5FDCR~~	
FC98H	~~PWM47T1H~~	~~PWM47T1L~~	~~PWM47T2H~~	~~PWM47T2L~~	~~PWM47CR~~	~~PWM47HLD~~		
FC90H	~~PWM46T1H~~	~~PWM46T1L~~	~~PWM46T2H~~	~~PWM46T2L~~	~~PWM46CR~~	~~PWM46HLD~~		
FC88H	~~PWM45T1H~~	~~PWM45T1L~~	~~PWM45T2H~~	~~PWM45T2L~~	~~PWM45CR~~	~~PWM45HLD~~		
FC80H	~~PWM44T1H~~	~~PWM44T1L~~	~~PWM44T2H~~	~~PWM44T2L~~	~~PWM44CR~~	~~PWM44HLD~~		
FC78H	~~PWM43T1H~~	~~PWM43T1L~~	~~PWM43T2H~~	~~PWM43T2L~~	~~PWM31CR~~	~~PWM43HLD~~		
FC70H	~~PWM42T1H~~	~~PWM42T1L~~	~~PWM42T2H~~	~~PWM42T2L~~	~~PWM42CR~~	~~PWM42HLD~~		
FC68H	~~PWM41T1H~~	~~PWM41T1L~~	~~PWM41T2H~~	~~PWM41T2L~~	~~PWM41CR~~	~~PWM41HLD~~		
FC60H	~~PWM40T1H~~	~~PWM40T1L~~	~~PWM40T2H~~	~~PWM40T2L~~	~~PWM40CR~~	~~PWM40HLD~~		
FC50H	~~PWM4CH~~	~~PWM4CL~~	~~PWM4CKS~~	~~PWM4TADCH~~	~~PWM4TADCL~~	~~PWM4IF~~	~~PWM4FDCR~~	
FC48H	~~PWM37T1H~~	~~PWM37T1L~~	~~PWM37T2H~~	~~PWM37T2L~~	~~PWM37CR~~	~~PWM37HLD~~		
FC40H	~~PWM36T1H~~	~~PWM36T1L~~	~~PWM36T2H~~	~~PWM36T2L~~	~~PWM36CR~~	~~PWM36HLD~~		
FC38H	~~PWM35T1H~~	~~PWM35T1L~~	~~PWM35T2H~~	~~PWM35T2L~~	~~PWM35CR~~	~~PWM35HLD~~		
FC30H	~~PWM34T1H~~	~~PWM34T1L~~	~~PWM34T2H~~	~~PWM34T2L~~	~~PWM34CR~~	~~PWM34HLD~~		
FC28H	~~PWM33T1H~~	~~PWM33T1L~~	~~PWM33T2H~~	~~PWM33T2L~~	~~PWM31CR~~	~~PWM33HLD~~		
FC20H	~~PWM32T1H~~	~~PWM32T1L~~	~~PWM32T2H~~	~~PWM32T2L~~	~~PWM32CR~~	~~PWM32HLD~~		
FC18H	~~PWM31T1H~~	~~PWM31T1L~~	~~PWM31T2H~~	~~PWM31T2L~~	~~PWM31CR~~	~~PWM31HLD~~		
FC10H	~~PWM30T1H~~	~~PWM30T1L~~	~~PWM30T2H~~	~~PWM30T2L~~	~~PWM30CR~~	~~PWM30HLD~~		
FC00H	~~PWM3CH~~	~~PWM3CL~~	~~PWM3CKS~~			~~PWM3IF~~	~~PWM3FDCR~~	

STC MCU

8.6 STC8G1K08T touch key family

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H		CH	CCAP0H	CCAP1H	CCAP2H			RSTCFG
F0H	B		PCA_PWM0	PCA_PWM1	PCA_PWM2	IAP_TPS		
E8H		CL	CCAP0L	CCAP1L	CCAP2L			AUXINTIF
E0H	ACC			DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2		ADCCFG	
D0H	PSW						T2H	T2L
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H		WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2		ADC_CONTR	ADC_RES	ADC_RESL	
B0H	P3	P3M1	P3M0			IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH			TA	IE2
A0H			P_SW1					
98H	SCON	SBUF				IRCBAND	LIRTRIM	IRTRIM
90H	P1	P1M1	P1M0					
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H		SP	DPL	DPH				PCON

Bit addressable

Non bit addressable

Note: Bit addressing can only be carried out if the register address is divisible by 8, while the bit addressing is not available if the register address is not divisible by 8.

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FEA8H	ADCTIM							
FEA0H			TM2PS					
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE30H		P1IE		P3IE				
FE28H		P1DR		P3DR		P5DR		
FE20H		P1SR		P3SR		P5SR		
FE18H		P1NCS		P3NCS		P5NCS		
FE10H		P1PU		P3PU		P5PU		
FE00H	CKSEL	CLKDIV	HIRCCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDB	
FB68H	TSTH12H	TSTH12L	TSTH13H	TSTH13L	TSTH14H	TSTH14L	TSTH15H	TSTH15L
FB60H	TSTH08H	TSTH08L	TSTH09H	TSTH09L	TSTH10H	TSTH10L	TSTH11H	TSTH11L
FB58H	TSTH04H	TSTH04L	TSTH05H	TSTH05L	TSTH06H	TSTH06L	TSTH07H	TSTH07L
FB50H	TSTH00H	TSTH00L	TSTH01H	TSTH01L	TSTH02H	TSTH02L	TSTH03H	TSTH03L
FB48H	TSRT	TSDATH	TSDATL					
FB40H	TSCHEN1	TSCHEN2	TSCFG1	TSCFG2	TSWUTC	TSCTRL	TSSTA1	TSSTA2
FB28H	COM0_DC_H	COM1_DC_H	COM2_DC_H	COM3_DC_H	COM4_DC_H	COM5_DC_H	COM6_DC_H	COM7_DC_H
FB20H	COM0_DC_L	COM1_DC_L	COM2_DC_L	COM3_DC_L	COM4_DC_L	COM5_DC_L	COM6_DC_L	COM7_DC_L
FB18H	COM0_DA_H	COM1_DA_H	COM2_DA_H	COM3_DA_H	COM4_DA_H	COM5_DA_H	COM6_DA_H	COM7_DA_H
FB10H	COM0_DA_L	COM1_DA_L	COM2_DA_L	COM3_DA_L	COM4_DA_L	COM5_DA_L	COM6_DA_L	COM7_DA_L
FB00H	COMEN	SEGENL	SEGENH	LEDCTRL	LEDCKS			

8.7 STC15H2K64S4 family

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H		CH	CCAP0H	CCAP1H	CCAP2H		PWMCFG45	RSTCFG
F0H	B	PWMSET	PCA_PWM0	PCA_PWM1	PCA_PWM2	IAP_TPS	PWMCFG01	PWMCFG23
E8H		CL	CCAP0L	CCAP1L	CCAP2L		IP3H	AUXINTIF
E0H	ACC			DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2		ADCCFG	IP3
D0H	PSW	T4T3M	TH4	TL4	TH3	TL3	T2H	T2L
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H	P4	WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2		ADC_CONTR	ADC_RES	ADC_RESL	
B0H	P3	P3M1	P3M0	P4M1	P4M0	IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH	S3CON	S3BUF	TA	IE2
A0H	P2	BUS_SPEED	P_SW1					
98H	SCON	SBUF	S2CON	S2BUF		IRCBAND	LIRTRIM	IRTRIM
90H	P1	P1M1	P1M0	P0M1	P0M0	P2M1	P2M0	
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H	P0	SP	DPL	DPH	S4CON	S4BUF		PCON

Bit addressable Non bit addressable

Note: Bit addressing can only be carried out if the register address is divisible by 8, while the bit addressing is not available if the register address is not divisible by 8.

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FCF0H	MD3	MD2	MD1	MD0	MD5	MD4	ARCON	OPCON
FFE8H	PWM27T1H	PWM27T1L	PWM27T2H	PWM27T2L	PWM27CR	PWM27HLD		
FFE0H	PWM26T1H	PWM26T1L	PWM26T2H	PWM26T2L	PWM26CR	PWM26HLD		
FFD8H	PWM25T1H	PWM25T1L	PWM25T2H	PWM25T2L	PWM25CR	PWM25HLD		
FFD0H	PWM24T1H	PWM24T1L	PWM24T2H	PWM24T2L	PWM24CR	PWM24HLD		
FFC8H	PWM23T1H	PWM23T1L	PWM23T2H	PWM23T2L	PWM31CR	PWM23HLD		
FFC0H	PWM22T1H	PWM22T1L	PWM22T2H	PWM22T2L	PWM22CR	PWM22HLD		
FFB8H	PWM21T1H	PWM21T1L	PWM21T2H	PWM21T2L	PWM21CR	PWM21HLD		
FFB0H	PWM20T1H	PWM20T1L	PWM20T2H	PWM20T2L	PWM20CR	PWM20HLD		
FFA0H	PWM2CH	PWM2CL	PWM2CKS	PWM2TADCH	PWM2TADCL	PWM2IF	PWM2FDCR	
FF98H	PWM17T1H	PWM17T1L	PWM17T2H	PWM17T2L	PWM17CR	PWM17HLD		
FF90H	PWM16T1H	PWM16T1L	PWM16T2H	PWM16T2L	PWM16CR	PWM16HLD		
FF88H	PWM15T1H	PWM15T1L	PWM15T2H	PWM15T2L	PWM15CR	PWM15HLD		
FF80H	PWM14T1H	PWM14T1L	PWM14T2H	PWM14T2L	PWM14CR	PWM14HLD		
FF78H	PWM13T1H	PWM13T1L	PWM13T2H	PWM13T2L	PWM31CR	PWM13HLD		
FF70H	PWM12T1H	PWM12T1L	PWM12T2H	PWM12T2L	PWM12CR	PWM12HLD		
FF68H	PWM11T1H	PWM11T1L	PWM11T2H	PWM11T2L	PWM11CR	PWM11HLD		
FF60H	PWM10T1H	PWM10T1L	PWM10T2H	PWM10T2L	PWM10CR	PWM10HLD		
FF50H	PWM1CH	PWM1CL	PWM1CKS			PWM1IF	PWM1FDCR	
FF48H	PWM07T1H	PWM07T1L	PWM07T2H	PWM07T2L	PWM07CR	PWM07HLD		
FF40H	PWM06T1H	PWM06T1L	PWM06T2H	PWM06T2L	PWM06CR	PWM06HLD		
FF38H	PWM05T1H	PWM05T1L	PWM05T2H	PWM05T2L	PWM05CR	PWM05HLD		
FF30H	PWM04T1H	PWM04T1L	PWM04T2H	PWM04T2L	PWM04CR	PWM04HLD		
FF28H	PWM03T1H	PWM03T1L	PWM03T2H	PWM03T2L	PWM31CR	PWM03HLD		
FF20H	PWM02T1H	PWM02T1L	PWM02T2H	PWM02T2L	PWM02CR	PWM02HLD		
FF18H	PWM01T1H	PWM01T1L	PWM01T2H	PWM01T2L	PWM01CR	PWM01HLD		
FF10H	PWM00T1H	PWM00T1L	PWM00T2H	PWM00T2L	PWM00CR	PWM00HLD		
FF00H	PWM0CH	PWM0CL	PWM0CKS	PWM0TADCH	PWM0TADCL	PWM0IF	PWM0FDCR	
FEA8H	ADCTIM							
FEA0H			TM2PS	TM3PS	TM4PS			
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE30H	P0IE	P1IE						

FE28H	P0DR	P1DR	P2DR	P3DR	P4DR	P5DR		
FE20H	P0SR	P1SR	P2SR	P3SR	P4SR	P5SR		
FE18H	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS		
FE10H	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU		
FE00H	CKSEL	CLKDIV	HIRCCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDB	
FCE8H	PWM57T1H	PWM57T1L	PWM57T2H	PWM57T2L	PWM57CR	PWM57HLD		
FCE0H	PWM56T1H	PWM56T1L	PWM56T2H	PWM56T2L	PWM56CR	PWM56HLD		
FCD8H	PWM55T1H	PWM55T1L	PWM55T2H	PWM55T2L	PWM55CR	PWM55HLD		
FCD0H	PWM54T1H	PWM54T1L	PWM54T2H	PWM54T2L	PWM54CR	PWM54HLD		
FCC8H	PWM53T1H	PWM53T1L	PWM53T2H	PWM53T2L	PWM31CR	PWM53HLD		
FCC0H	PWM52T1H	PWM52T1L	PWM52T2H	PWM52T2L	PWM52CR	PWM52HLD		
FCB8H	PWM51T1H	PWM51T1L	PWM51T2H	PWM51T2L	PWM51CR	PWM51HLD		
FCB0H	PWM50T1H	PWM50T1L	PWM50T2H	PWM50T2L	PWM50CR	PWM50HLD		
FCA0H	PWM5CH	PWM5CL	PWM5CKS			PWM5IF	PWM5FDCR	
FC98H	PWM47T1H	PWM47T1L	PWM47T2H	PWM47T2L	PWM47CR	PWM47HLD		
FC90H	PWM46T1H	PWM46T1L	PWM46T2H	PWM46T2L	PWM46CR	PWM46HLD		
FC88H	PWM45T1H	PWM45T1L	PWM45T2H	PWM45T2L	PWM45CR	PWM45HLD		
FC80H	PWM44T1H	PWM44T1L	PWM44T2H	PWM44T2L	PWM44CR	PWM44HLD		
FC78H	PWM43T1H	PWM43T1L	PWM43T2H	PWM43T2L	PWM31CR	PWM43HLD		
FC70H	PWM42T1H	PWM42T1L	PWM42T2H	PWM42T2L	PWM42CR	PWM42HLD		
FC68H	PWM41T1H	PWM41T1L	PWM41T2H	PWM41T2L	PWM41CR	PWM41HLD		
FC60H	PWM40T1H	PWM40T1L	PWM40T2H	PWM40T2L	PWM40CR	PWM40HLD		
FC50H	PWM4CH	PWM4CL	PWM4CKS	PWM4TADCH	PWM4TADCL	PWM4IF	PWM4FDCR	
FC48H	PWM37T1H	PWM37T1L	PWM37T2H	PWM37T2L	PWM37CR	PWM37HLD		
FC40H	PWM36T1H	PWM36T1L	PWM36T2H	PWM36T2L	PWM36CR	PWM36HLD		
FC38H	PWM35T1H	PWM35T1L	PWM35T2H	PWM35T2L	PWM35CR	PWM35HLD		
FC30H	PWM34T1H	PWM34T1L	PWM34T2H	PWM34T2L	PWM34CR	PWM34HLD		
FC28H	PWM33T1H	PWM33T1L	PWM33T2H	PWM33T2L	PWM31CR	PWM33HLD		
FC20H	PWM32T1H	PWM32T1L	PWM32T2H	PWM32T2L	PWM32CR	PWM32HLD		
FC18H	PWM31T1H	PWM31T1L	PWM31T2H	PWM31T2L	PWM31CR	PWM31HLD		
FC10H	PWM30T1H	PWM30T1L	PWM30T2H	PWM30T2L	PWM30CR	PWM30HLD		
FC00H	PWM3CH	PWM3CL	PWM3CKS			PWM3IF	PWM3FDCR	

STC MCU

8.8 List of Special Function Registers

Note: Bit addressing can only be carried out if the register address is divisible by 8,while the bit addressing is not available if the register address is not divisible by 8.

Registers in STC8G series that can be bit-addressed are: P0 (80H), TCON (88H), P1 (90H), SCON (98H), P2 (A0H), IE (A8H), P3 (B0H), IP (B8H), P4(C0H), P5(C8H), PSW(D0H), CCON (D8H), ACC(E0H), P6 (E8H), B (F0H), P7(F8H).

Symbol	Description	Address	Bit Address and Symbol								Value after Reset
			B7	B6	B5	B4	B3	B2	B1	B0	
P0	Port 0	80H	P07	P06	P05	P04	P03	P02	P01	P00	1111,1111
SP	Stack Pointer	81H									0000,0111
DPL	Data pointer low byte register	82H									0000,0000
DPH	Data pointer high byte register	83H									0000,0000
S4CON	Serial port 4 control register	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
S4BUF	Serial port 4 data buffer register	85H									0000,0000
PCON	Power control register	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
TCON	Timer 0 and 1 control register	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
TMOD	Timer 0 and 1 mode register	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000,0000
TL0	Timer 0 low byte register	8AH									0000,0000
TL1	Timer 1 low byte register	8BH									0000,0000
TH0	Timer 0 high byte register	8CH									0000,0000
TH1	Timer 1 high byte register	8DH									0000,0000
AUXR	Auxiliary register 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
INTCLKO	External interrupt and clock output control register	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
P1	Port 1	90H	P17	P16	P15	P14	P13	P12	P11	P10	1111,1111
P1M1	Port 1 mode register 1	91H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1	1111,1111
P1M0	Port 1 mode register 0	92H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0	0000,0000
P0M1	Port 0 mode register 1	93H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1	1111,1111
P0M0	Port 0 mode register 0	94H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0	0000,0000
P2M1	Port 2 mode register 1	95H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1	1111,1111
P2M0	Port 2 mode register 0	96H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0	0000,0000
SCON	UART1 control register	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	UART1 data buffer register	99H									0000,0000
S2CON	UART2 control register	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0x00,0000
S2BUF	UART2 data buffer registe	9BH									0000,0000
IRCBAND	IRC band selection detection	9DH	-	-	-	-	-	-	-	SEL	xxxx,xxxn
LIRTRIM	IRC frequency trim register	9EH	-	-	-	-	-	-	LIRTRIM[1:0]		xxxx,xxnn
IRTRIM	IRC frequency adjustment register	9FH	IRTRIM[7:0]								nnnn,nnnn
P2	Port 2	A0H	P27	P26	P25	P24	P23	P22	P21	P20	1111,1111
BUS_SPEED	Bus speed control register	A1	RW_S[1:0]					SPEED[2:0]			
P_SW1	Peripheral port switch register 1	A2H	S1_S[1:0]		CCP_S[1:0]		SPI_S[1:0]		0	-	nn00,000x
IE	Interrupt enable register	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000,0000
SADDR	UART1 slave address register	A9H									0000,0000
WKTCL	Wake-up Timer Control Register low	AAH									1111,1111
WKTCH	Wake-up Timer Control Register High Byte	ABH	WKTEN								0111,1111
S3CON	UART3 control register	ACH	S3SM0	S3ST4	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S3BUF	UART3 data buffer register	ADH									0000,0000
TA	DPTR Timing control register	AEH									0000,0000
IE2	Interrupt enable register 2	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000,0000
P3	Port 3	B0H	P37	P36	P35	P34	P33	P32	P31	P30	1111,1111
P3M1	Port 3 mode register 1	B1H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1	1111,1100
P3M0	Port 3 mode register 0	B2H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0	0000,0000
P4M1	Port 4 mode register 1	B3H	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1	1111,1111
P4M0	Port 4 mode register 0	B4H	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0	0000,0000
IP2	2nd Interrupt Priority register low byte	B5H	PPWM2FD	PI2C	PCMP	PX4	PPWM0FD	PPWM0	PSPI	PS2	0000,0000
IP2H	2nd Interrupt Priority register high byte	B6H	PPWM2FDH	PI2CH	PCMPH	PX4H	PPWM0FDH	PPWM0H	PSPIH	PS2H	0000,0000
IPH	Interrupt Priority High Byte	B7H	PPCAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	0000,0000
IP	Interrupt Priority Low Byte	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000,0000
SADEN	UART1 slave address enable register	B9H									0000,0000
P_SW2	Peripheral port switch register 2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S	0x00,0000
ADC_CONTR	ADC control register	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]				000x,0000
ADC_RES	ADC Result High Byte	BDH									0000,0000
ADC_RESL	ADC Result Low Byte	BEH									0000,0000
P4	Port 4	C0H	P47	P46	P45	P44	P43	P42	P41	P40	1111,1111
WDT_CONTR	Watchdog control register	C1H	WDT_FL	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]			0xn0,nnnn

			AG							
IAP_DATA	IAP Flash Data Register	C2H	1111,1111							
IAP_ADDRH	IAP Flash Address High Byte	C3H	0000,0000							
IAP_ADDRL	IAP Flash Address Low Byte	C4H	0000,0000							
IAP_CMD	IAP Flash Command Register	C5H	-	-	-	-	-	-	CMD[1:0]	xxxx,xx00
IAP_TRIG	IAP Flash Trigger register	C6H	0000,0000							
IAP_CONTR	IAP Control Register	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-	0000,xxxx
P5	Port 5	C8H	-	-	P55	P54	P53	P52	P51	P50	xx11,1111
P5M1	Port 5 mode register 1	C9H	-	-	P55M1	P54M1	P53M1	P52M1	P51M1	P50M1	xx11,1111
P5M0	Port 5 mode register 0	CAH	-	-	P55M0	P54M0	P53M0	P52M0	P51M0	P50M0	xx00,0000
SPSTAT	SPI Status register	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
SPCTL	SPI Control Register	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]		0000,0100
SPDAT	SPI Data Register	CFH	0000,0000							
PSW	Program Status Word Register	D0H	CY	AC	F0	RS1	RS0	OV	-	P	0000,00x0
T4T3M	Timer4 and Timer 3 Control Register	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000,0000
T4H	Timer 4 high byte register	D2H	0000,0000							
T4L	Timer 4 low byte register	D3H	0000,0000							
T3H	Timer 3 high byte register	D4H	0000,0000							
T3L	Timer 3 low byte register	D5H	0000,0000							
T2H	Timer 2 high byte register	D6H	0000,0000							
T2L	Timer 2 low byte register	D7H	0000,0000							
CCON	PCA Control Register	D8H	CF	CR	-	-	-	CCF2	CCF1	CCF0	00xx,x000
CMOD	PCA Mode Register	D9H	CIDL	-	-	-	CPS[2:0]			ECF	0xxx,0000
CCAPM0	PCA 0 Mode Control Register	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAPM1	PCA 1 Mode Control Register	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CCAPM2	PCA 2 Mode Control Register	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
ADCCFG	ADC Configuration Register	DEH	-	-	RESFMT	-	SPEED[3:0]				xx0x,0000
IP3	3nd Interrupt Priority register low byte	DFH	PPWM4FD	PPWM5	PPWM4	PPWM3	PPWM2	PPWM1	PS4	PS3	0000,0000
ACC	Accumulator	E0H	0000,0000							
DPS	DPTR Selection Register	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL	0000,0xx0
DPL1	2nd Data pointer low byte	E4H	0000,0000							
DPH1	2nd Data pointer high byte	E5H	0000,0000							
CMPCR1	Comparator Control Register 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
CMPCR2	Comparator Control Register 2	E7H	INVCMPO	DISFLT	LCDTY[5:0]						0000,0000
CL	PCA Counter Low Byte	E9H	0000,0000							
CCAP0L	PCA 0 Capture Register Low Byte	EAH	0000,0000							
CCAP1L	PCA 1 Capture Register Low Byte	EBH	0000,0000							
CCAP2L	PCA 2 Capture Register Low Byte	ECH	0000,0000							
IP3H	3nd Interrupt Priority Register High Byte	EEH	PPWM4FDH	PPWM5H	PPWM4H	PPWM3H	PPWM2H	PPWM1H	PS4H	PS3H	0000,0000
AUXINTIF	Extended External Interrupt Flag Register	EFH	-	INT4IF	INT3IF	INT2IF	-	-	-	T2IF	x000,xxx0
B	B register	F0H	0000,0000							
PWMSET	PWM Set Register	F1H	ENGLBSET	PWMRST	ENPWM5	ENPWM4	ENPWM3	ENPWM2	ENPWM1	ENPWM0	0000,0000
PCA_PWM0	PCA0 PWM Mode Register	F2H	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L	0000,0000
PCA_PWM1	PCA1 PWM Mode Register	F3H	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L	0000,0000
PCA_PWM2	PCA2 PWM Mode Register	F4H	EBS2[1:0]		XCCAP2H[1:0]		XCCAP2L[1:0]		EPC2H	EPC2L	0000,0000
IAP_TPS	IAP Waiting Time Control Register	F5H	-	-	IAPTPS[5:0]						xx00,0000
PWMCFG01	PWM Configuration Register	F6H	PWM1CBIF	EPWM1CBI	FLTPS0	PWM1CEN	PWM0CBIF	EPWM0CBI	ENPWM0TA	PWM0CEN	0000,0000
PWMCFG23	PWM Configuration Register	F7H	PWM3CBIF	EPWM3CBI	FLTPS1	PWM3CEN	PWM2CBIF	EPWM2CBI	ENPWM2TA	PWM2CEN	0000,0000
CH	PCA Counter High Byte	F9H	0000,0000							
CCAP0H	PCA 0 Capture Register High Byte	FAH	0000,0000							
CCAP1H	PCA 1 Capture Register High Byte	FBH	0000,0000							
CCAP2H	PCA 2 Capture Register High Byte	FCH	0000,0000							
PWMCFG45	PWM Configuration Register	FEH	PWM5CBIF	EPWM5CBI	FLTPS2	PWM5CEN	PWM4CBIF	EPWM4CBI	ENPWM4TA	PWM4CEN	0000,0000
RSTCFG	Reset Configuration Register	FFH	-	ENLVR	-	P54RST	-	-	LVDS[1:0]		xnxn,xxnn

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

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CKSEL	Clock Select Register	FE00H	-	-	-	-	-	-	MCKSEL[1:0]		xxxx,xx00
CLKDIV	Clock Divide Register	FE01H	0000,0100								
HIRCCR	Internal high speed Oscillator Control Register	FE02H	ENHIRC	-	-	-	-	-	-	HIRCST	1xxx,xxx0
XOSCCR	External Oscillator Control Register	FE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST	00xx,xxx0

IRC32KCR	Internal 32K Oscillator Control Register	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0
MCLKOCR	Main Clock Output Control Register	FE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000
IRCDB	Internal high-speed oscillator debounce control	FE06H	IRCDB_PAR[7:0]								1000,0000
P0PU	Port0 Pull-up Resistor Control Register	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU	0000,0000
P1PU	Port1 Pull-up Resistor Control Register	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU	0000,0000
P2PU	Port2 Pull-up Resistor Control Register	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU	0000,0000
P3PU	Port3 Pull-up Resistor Control Register	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU	0000,0000
P4PU	Port4 Pull-up Resistor Control Register	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU	0000,0000
P5PU	Port5 Pull-up Resistor Control Register	FE15H	P57PU	P56PU	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU	xx00,0000
P0NCS	Port0 Schmitt Trigger Control Register	FE18H	P07NCS	P06NCS	P05NCS	P04NCS	P03NCS	P02NCS	P01NCS	P00NCS	0000,0000
P1NCS	Port1 Schmitt Trigger Control Register	FE19H	P17NCS	P16NCS	P15NCS	P14NCS	P13NCS	P12NCS	P11NCS	P10NCS	0000,0000
P2NCS	Port2 Schmitt Trigger Control Register	FE1AH	P27NCS	P26NCS	P25NCS	P24NCS	P23NCS	P22NCS	P21NCS	P20NCS	0000,0000
P3NCS	Port3 Schmitt Trigger Control Register	FE1BH	P37NCS	P36NCS	P35NCS	P34NCS	P33NCS	P32NCS	P31NCS	P30NCS	0000,0000
P4NCS	Port4 Schmitt Trigger Control Register	FE1CH	P47NCS	P46NCS	P45NCS	P44NCS	P43NCS	P42NCS	P41NCS	P40NCS	0000,0000
P5NCS	Port5 Schmitt Trigger Control Register	FE1DH	P57NCS	P56NCS	P55NCS	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS	xx00,0000
P0SR	Port0 Level Shift Rate Register	FE20H	P07SR	P06SR	P05SR	P04SR	P03SR	P02SR	P01SR	P00SR	1111,1111
P1SR	Port1 Level Shift Rate Register	FE21H	P17SR	P16SR	P15SR	P14SR	P13SR	P12SR	P11SR	P10SR	1111,1111
P2SR	Port2 Level Shift Rate Register	FE22H	P27SR	P26SR	P25SR	P24SR	P23SR	P22SR	P21SR	P20SR	1111,1111
P3SR	Port3 Level Shift Rate Register	FE23H	P37SR	P36SR	P35SR	P34SR	P33SR	P32SR	P31SR	P30SR	1111,1111
P4SR	Port4 Level Shift Rate Register	FE24H	P47SR	P46SR	P45SR	P44SR	P43SR	P42SR	P41SR	P40SR	1111,1111
P5SR	Port5 Level Shift Rate Register	FE25H	P57SR	P56SR	P55SR	P54SR	P53SR	P52SR	P51SR	P50SR	xx11,1111
P0DR	Port0 Drive Current Control Register	FE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR	1111,1111
P1DR	Port1 Drive Current Control Register	FE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR	1111,1111
P2DR	Port2 Drive Current Control Register	FE2AH	P27DR	P26DR	P25DR	P24DR	P23DR	P22DR	P21DR	P20DR	1111,1111
P3DR	Port3 Drive Current Control Register	FE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR	1111,1111
P4DR	Port4 Drive Current Control Register	FE2CH	P47DR	P46DR	P45DR	P44DR	P43DR	P42DR	P41DR	P40DR	1111,1111
P5DR	Port5 Drive Current Control Register	FE2DH	P57DR	P56DR	P55DR	P54DR	P53DR	P52DR	P51DR	P50DR	xx11,0000
P0IE	Port0 Input Enable Control Register	FE30H	P07IE	P06IE	P05IE	P04IE	P03IE	P02IE	P11IE	P00IE	1111,1111
P1IE	Port1 Input Enable Control Register	FE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE	1111,1111
P3IE	Port3 Input Enable Control Register r	FE33H	P37IE	P36IE	P35IE	P34IE	P33IE	P32IE	P31IE	P30IE	1111,1111
I2CCFG	I²C Configuration Register	FE80H	ENI2C	MSSL	MSSPEED[6:1]						0000,0000
I2CMSCR	I²C Master Control Register	FE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I²C Master Status Register	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I²C Slave Control Register	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I²C Slave Status Register	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I²C Slave Address Register	FE85H	SLADR[6:0]							MA	0000,0000
I2CTXD	I²C Data Transmission Register	FE86H									0000,0000
I2CRXD	I²C Data Receive Register	FE87H									0000,0000
I2CMSAUX	I²C Master Auxiliary Control Register	FE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0
TM2PS	Timer2 Clock Prescaler Register	FEA2H									0000,0000
TM3PS	Timer3 Clock Prescaler Register	FEA3H									0000,0000
TM4PS	Timer4 Clock Prescaler Register	FEA4H									0000,0000
ADCTIM	ADC Timing Control Register	FEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]					0010,1010
PWM0CH	PWM0 Counter High Byte	FF00H	-								x000,0000
PWM0CL	PWM0 Counter Low Byte	FF01H									0000,0000
PWM0CKS	PWM0 Clock Select Register	FF02H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000
PWM0TADCH	PWM0 Trigger ADC Counter High Byte	FF03H	-								x000,0000
PWM0TADCL	PWM0 Trigger ADC Counter Low Byte	FF04H									0000,0000
PWM0IF	PWM0 Interrupt Flag Register	FF05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM0FDCR	PWM0 Fault Detection Control Register	FF06H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000
PWM00T1H	PWM00T1 High Byte	FF10H	-								x000,0000
PWM00T1L	PWM00T1 Low Byte	FF11H									0000,0000
PWM00T2H	PWM00T2 High Byte	FF12H	-								x000,0000
PWM00T2L	PWM00T2 Low Byte	FF13H									0000,0000
PWM00CR	PWM00 Control Register	FF14H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM00HLD	PWM00 Level Holding Control Register	FF15H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00

PWM01T1H	PWM01T1 High Byte	FF18H	-								x000,0000
PWM01T1L	PWM01T1 Low Byte	FF19H									0000,0000
PWM01T2H	PWM01T2 High Byte	FF1AH	-								x000,0000
PWM01T2L	PWM01T2 Low Byte	FF1BH									0000,0000
PWM01CR	PWM01 Control Register	FF1CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM01HLD	PWM01 Level Holding Control Register	FF1DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM02T1H	PWM02T1 High Byte	FF20H	-								x000,0000
PWM02T1L	PWM02T1 Low Byte	FF21H									0000,0000
PWM02T2H	PWM02T2 High Byte	FF22H	-								x000,0000
PWM02T2L	PWM02T2 Low Byte	FF23H									0000,0000
PWM02CR	PWM02 Control Register	FF24H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM02HLD	PWM02 Level Holding Control Register	FF25H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM03T1H	PWM03T1 High Byte	FF28H	-								x000,0000
PWM03T1L	PWM03T1 Low Byte	FF29H									0000,0000
PWM03T2H	PWM03T2 High Byte	FF2AH	-								x000,0000
PWM03T2L	PWM03T2 Low Byte	FF2BH									0000,0000
PWM03CR	PWM03 Control Register	FF2CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM03HLD	PWM03 Level Holding Control Register	FF2DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM04T1H	PWM04T1 High Byte	FF30H	-								x000,0000
PWM04T1L	PWM04T1 Low Byte	FF31H									0000,0000
PWM04T2H	PWM04T2 High Byte	FF32H	-								x000,0000
PWM04T2L	PWM04T2 Low Byte	FF33H									0000,0000
PWM04CR	PWM04 Control Register	FF34H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM04HLD	PWM04 Level Holding Control Register	FF35H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM05T1H	PWM05T1 High Byte	FF38H	-								x000,0000
PWM05T1L	PWM05T1 Low Byte	FF39H									0000,0000
PWM05T2H	PWM05T2 High Byte	FF3AH	-								x000,0000
PWM05T2L	PWM05T2 Low Byte	FF3BH									0000,0000
PWM05CR	PWM05 Control Register	FF3CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM05HLD	PWM05 Level Holding Control Register	FF3DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM06T1H	PWM06T1 High Byte	FF40H	-								x000,0000
PWM06T1L	PWM06T1 Low Byte	FF41H									0000,0000
PWM06T2H	PWM06T2 High Byte	FF42H	-								x000,0000
PWM06T2L	PWM06T2 Low Byte	FF43H									0000,0000
PWM06CR	PWM06 Control Register	FF44H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM06HLD	PWM06 Level Holding Control Register	FF45H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM07T1H	PWM07T1 High Byte	FF48H	-								x000,0000
PWM07T1L	PWM07T1 Low Byte	FF49H									0000,0000
PWM07T2H	PWM07T2 High Byte	FF4AH	-								x000,0000
PWM07T2L	PWM07T2 Low Byte	FF4BH									0000,0000
PWM07CR	PWM07 Control Register	FF4CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM07HLD	PWM07 Level Holding Control Register	FF4DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM1CH	PWM1 Counter High Byte	FF50H	-								x000,0000
PWM1CL	PWM1 Counter Low Byte	FF51H									0000,0000
PWM1CKS	PWM1 Clock Select Register	FF52H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000
PWM1IF	PWM1 Interrupt Flag Register	FF55H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM1FDCR	PWM1 Fault Detection Control Register	FF56H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF	0000,0000
PWM10T1H	PWM10T1 High Byte	FF60H	-								x000,0000
PWM10T1L	PWM10T1 Low Byte	FF61H									0000,0000
PWM10T2H	PWM10T2 High Byte	FF62H	-								x000,0000
PWM10T2L	PWM10T2 Low Byte	FF63H									0000,0000
PWM10CR	PWM10 Control Register	FF64H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM10HLD	PWM10 Level Holding Control Register	FF65H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM11T1H	PWM11T1 High Byte	FF68H	-								x000,0000
PWM11T1L	PWM11T1 Low Byte	FF69H									0000,0000
PWM11T2H	PWM11T2 High Byte	FF6AH	-								x000,0000
PWM11T2L	PWM11T2 Low Byte	FF6BH									0000,0000
PWM11CR	PWM11 Control Register	FF6CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM11HLD	PWM11 Level Holding Control Register	FF6DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM12T1H	PWM12T1 High Byte	FF70H	-								x000,0000
PWM12T1L	PWM12T1 Low Byte	FF71H									0000,0000
PWM12T2H	PWM12T2 High Byte	FF72H	-								x000,0000
PWM12T2L	PWM12T2 Low Byte	FF73H									0000,0000
PWM12CR	PWM12 Control Register	FF74H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM12HLD	PWM12 Level Holding Control Register	FF75H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM13T1H	PWM13T1 High Byte	FF78H	-								x000,0000
PWM13T1L	PWM13T1 Low Byte	FF79H									0000,0000
PWM13T2H	PWM13T2 High Byte	FF7AH	-								x000,0000
PWM13T2L	PWM13T2 Low Byte	FF7BH									0000,0000
PWM13CR	PWM13 Control Register	FF7CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM13HLD	PWM13 Level Holding Control Register	FF7DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM14T1H	PWM14T1 High Byte	FF80H	-								x000,0000

PWM14T1L	PWM14T1 Low Byte	FF81H									0000,0000
PWM14T2H	PWM14T2 High Byte	FF82H	-								x000,0000
PWM14T2L	PWM14T2 Low Byte	FF83H									0000,0000
PWM14CR	PWM14 Control Register	FF84H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM14HLD	PWM14 Level Holding Control Register	FF85H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM15T1H	PWM15T1 High Byte	FF88H	-								x000,0000
PWM15T1L	PWM15T1 Low Byte	FF89H									0000,0000
PWM15T2H	PWM15T2 High Byte	FF8AH	-								x000,0000
PWM15T2L	PWM15T2 Low Byte	FF8BH									0000,0000
PWM15CR	PWM15 Control Register	FF8CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM15HLD	PWM15 Level Holding Control Register	FF8DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM16T1H	PWM16T1 High Byte	FF90H	-								x000,0000
PWM16T1L	PWM16T1 Low Byte	FF91H									0000,0000
PWM16T2H	PWM16T2 High Byte	FF92H	-								x000,0000
PWM16T2L	PWM16T2 Low Byte	FF93H									0000,0000
PWM16CR	PWM16 Control Register	FF94H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM16HLD	PWM16 Level Holding Control Register	FF95H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM17T1H	PWM17T1 High Byte	FF98H	-								x000,0000
PWM17T1L	PWM17T1 Low Byte	FF99H									0000,0000
PWM17T2H	PWM17T2 High Byte	FF9AH	-								x000,0000
PWM17T2L	PWM17T2 Low Byte	FF9BH									0000,0000
PWM17CR	PWM17 Control Register	FF9CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM17HLD	PWM17 Level Holding Control Register	FF9DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM2CH	PWM2 Counter High Byte	FFA0H	-								x000,0000
PWM2CL	PWM2 Counter Low Byte	FFA1H									0000,0000
PWM2CKS	PWM2 Clock Select Register	FFA2H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000
PWM2TADCH	PWM2 Trigger ADC Counter High Byte	FFA3H	-								x000,0000
PWM2TADCL	PWM2 Trigger ADC Counter Low Byte	FFA4H									0000,0000
PWM2IF	PWM2 Interrupt Flag Register	FFA5H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM2FDCR	PWM2 Fault Detection Control Register	FFA6H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000
PWM20T1H	PWM20T1 High Byte	FFB0H	-								x000,0000
PWM20T1L	PWM20T1 Low Byte	FFB1H									0000,0000
PWM20T2H	PWM20T2 High Byte	FFB2H	-								x000,0000
PWM20T2L	PWM20T2 Low Byte	FFB3H									0000,0000
PWM20CR	PWM20 Control Register	FFB4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM20HLD	PWM20 Level Holding Control Register	FFB5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM21T1H	PWM21T1 High Byte	FFB8H	-								x000,0000
PWM21T1L	PWM21T1 Low Byte	FFB9H									0000,0000
PWM21T2H	PWM21T2 High Byte	FFBAH	-								x000,0000
PWM21T2L	PWM21T2 Low Byte	FFBBH									0000,0000
PWM21CR	PWM21 Control Register	FFBCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM21HLD	PWM21 Level Holding Control Register	FFBDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM22T1H	PWM22T1 High Byte	FFC0H	-								x000,0000
PWM22T1L	PWM22T1 Low Byte	FFC1H									0000,0000
PWM22T2H	PWM22T2 High Byte	FFC2H	-								x000,0000
PWM22T2L	PWM22T2 Low Byte	FFC3H									0000,0000
PWM22CR	PWM22 Control Register	FFC4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM22HLD	PWM22 Level Holding Control Register	FFC5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM23T1H	PWM23T1 High Byte	FFC8H	-								x000,0000
PWM23T1L	PWM23T1 Low Byte	FFC9H									0000,0000
PWM23T2H	PWM23T2 High Byte	FFCAH	-								x000,0000
PWM23T2L	PWM23T2 Low Byte	FFCBH									0000,0000
PWM23CR	PWM23 Control Register	FFCCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM23HLD	PWM23 Level Holding Control Register	FFCDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM24T1H	PWM24T1 High Byte	FFD0H	-								x000,0000
PWM24T1L	PWM24T1 Low Byte	FFD1H									0000,0000
PWM24T2H	PWM24T2 High Byte	FFD2H	-								x000,0000
PWM24T2L	PWM24T2 Low Byte	FFD3H									0000,0000
PWM24CR	PWM24 Control Register	FFD4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM24HLD	PWM24 Level Holding Control Register	FFD5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM25T1H	PWM25T1 High Byte	FFD8H	-								x000,0000
PWM25T1L	PWM25T1 Low Byte	FFD9H									0000,0000
PWM25T2H	PWM25T2 High Byte	FFDAH	-								x000,0000
PWM25T2L	PWM25T2 Low Byte	FFDBH									0000,0000
PWM25CR	PWM25 Control Register	FFDCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM25HLD	PWM25 Level Holding Control Register	FFDDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM26T1H	PWM26T1 High Byte	FFE0H	-								x000,0000
PWM26T1L	PWM26T1 Low Byte	FFE1H									0000,0000
PWM26T2H	PWM26T2 High Byte	FFE2H	-								x000,0000
PWM26T2L	PWM26T2 Low Byte	FFE3H									0000,0000
PWM26CR	PWM26 Control Register	FFE4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000

PWM26HLD	PWM26 Level Holding Control Register	FFE5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM27T1H	PWM27T1 High Byte	FFE8H	-								x000,0000
PWM27T1L	PWM27T1 Low Byte	FFE9H									0000,0000
PWM27T2H	PWM27T2 High Byte	FFEAH	-								x000,0000
PWM27T2L	PWM27T2 Low Byte	FFEBH									0000,0000
PWM27CR	PWM27 Control Register	FFECH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM27HLD	PWM27 Level Holding Control Register	FFEDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM3CH	PWM3 Counter High Byte	FC00H	-								x000,0000
PWM3CL	PWM3 Counter Low Byte	FC01H									0000,0000
PWM3CKS	PWM3 Clock Select Register	FC02H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000
PWM3IF	PWM3 Interrupt Flag Register	FC05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM3FDCR	PWM3 Fault Detection Control Register	FC06H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF	0000,0000
PWM30T1H	PWM30T1 High Byte	FC10H	-								x000,0000
PWM30T1L	PWM30T1 Low Byte	FC11H									0000,0000
PWM30T2H	PWM30T2 High Byte	FC12H	-								x000,0000
PWM30T2L	PWM30T2 Low Byte	FC13H									0000,0000
PWM30CR	PWM30 Control Register	FC14H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM30HLD	PWM30 Level Holding Control Register	FC15H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM31T1H	PWM31T1 High Byte	FC18H	-								x000,0000
PWM31T1L	PWM31T1 Low Byte	FC19H									0000,0000
PWM31T2H	PWM31T2 High Byte	FC1AH	-								x000,0000
PWM31T2L	PWM31T2 Low Byte	FC1BH									0000,0000
PWM31CR	PWM31 Control Register	FC1CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM31HLD	PWM31 Level Holding Control Register	FC1DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM32T1H	PWM32T1 High Byte	FC20H	-								x000,0000
PWM32T1L	PWM32T1 Low Byte	FC21H									0000,0000
PWM32T2H	PWM32T2 High Byte	FC22H	-								x000,0000
PWM32T2L	PWM32T2 Low Byte	FC23H									0000,0000
PWM32CR	PWM32 Control Register	FC24H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM32HLD	PWM32 Level Holding Control Register	FC25H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM33T1H	PWM33T1 High Byte	FC28H	-								x000,0000
PWM33T1L	PWM33T1 Low Byte	FC29H									0000,0000
PWM33T2H	PWM33T2 High Byte	FC2AH	-								x000,0000
PWM33T2L	PWM33T2 Low Byte	FC2BH									0000,0000
PWM33CR	PWM33 Control Register	FC2CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM33HLD	PWM33 Level Holding Control Register	FC2DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM34T1H	PWM34T1 High Byte	FC30H	-								x000,0000
PWM34T1L	PWM34T1 Low Byte	FC31H									0000,0000
PWM34T2H	PWM34T2 High Byte	FC32H	-								x000,0000
PWM34T2L	PWM34T2 Low Byte	FC33H									0000,0000
PWM34CR	PWM34 Control Register	FC34H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM34HLD	PWM34 Level Holding Control Register	FC35H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM35T1H	PWM35T1 High Byte	FC38H	-								x000,0000
PWM35T1L	PWM35T1 Low Byte	FC39H									0000,0000
PWM35T2H	PWM35T2 High Byte	FC3AH	-								x000,0000
PWM35T2L	PWM35T2 Low Byte	FC3BH									0000,0000
PWM35CR	PWM35 Control Register	FC3CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM35HLD	PWM35 Level Holding Control Register	FC3DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM36T1H	PWM36T1 High Byte	FC40H	-								x000,0000
PWM36T1L	PWM36T1 Low Byte	FC41H									0000,0000
PWM36T2H	PWM36T2 High Byte	FC42H	-								x000,0000
PWM36T2L	PWM36T2 Low Byte	FC43H									0000,0000
PWM36CR	PWM36 Control Register	FC44H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM36HLD	PWM36 Level Holding Control Register	FC45H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM37T1H	PWM37T1 High Byte	FC48H	-								x000,0000
PWM37T1L	PWM37T1 Low Byte	FC49H									0000,0000
PWM37T2H	PWM37T2 High Byte	FC4AH	-								x000,0000
PWM37T2L	PWM37T2 Low Byte	FC4BH									0000,0000
PWM37CR	PWM37 Control Register	FC4CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM37HLD	PWM37 Level Holding Control Register	FC4DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM4CH	PWM4 Counter High Byte	FC50H	-								x000,0000
PWM4CL	PWM4 Counter Low Byte	FC51H									0000,0000
PWM4CKS	PWM4 Clock Select Register	FC52H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000
PWM4TADCH	PWM4 Trigger ADC Counter High Byte	FC53H	-								x000,0000
PWM4TADCL	PWM4 Trigger ADC Counter Low Byte	FC54H									0000,0000
PWM4IF	PWM4 Interrupt Flag Register	FC55H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM4FDCR	PWM4 Fault Detection Control Register	FC56H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000
PWM40T1H	PWM40T1 High Byte	FC60H	-								x000,0000
PWM40T1L	PWM40T1 Low Byte	FC61H									0000,0000
PWM40T2H	PWM40T2 High Byte	FC62H	-								x000,0000

PWM40T2L	PWM40T2 Low Byte	FC63H									0000,0000
PWM40CR	PWM40 Control Register	FC64H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM40HLD	PWM40 Level Holding Control Register	FC65H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM41T1H	PWM41T1 High Byte	FC68H	-								x000,0000
PWM41T1L	PWM41T1 Low Byte	FC69H									0000,0000
PWM41T2H	PWM41T2 High Byte	FC6AH	-								x000,0000
PWM41T2L	PWM41T2 Low Byte	FC6BH									0000,0000
PWM41CR	PWM41 Control Register	FC6CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM41HLD	PWM41 Level Holding Control Register	FC6DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM42T1H	PWM42T1 High Byte	FC70H	-								x000,0000
PWM42T1L	PWM42T1 Low Byte	FC71H									0000,0000
PWM42T2H	PWM42T2 High Byte	FC72H	-								x000,0000
PWM42T2L	PWM42T2 Low Byte	FC73H									0000,0000
PWM42CR	PWM42 Control Register	FC74H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM42HLD	PWM42 Level Holding Control Register	FC75H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM43T1H	PWM43T1 High Byte	FC78H	-								x000,0000
PWM43T1L	PWM43T1 Low Byte	FC79H									0000,0000
PWM43T2H	PWM43T2 High Byte	FC7AH	-								x000,0000
PWM43T2L	PWM43T2 Low Byte	FC7BH									0000,0000
PWM43CR	PWM43 Control Register	FC7CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM43HLD	PWM43 Level Holding Control Register	FC7DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM44T1H	PWM44T1 High Byte	FC80H	-								x000,0000
PWM44T1L	PWM44T1 Low Byte	FC81H									0000,0000
PWM44T2H	PWM44T2 High Byte	FC82H	-								x000,0000
PWM44T2L	PWM44T2 Low Byte	FC83H									0000,0000
PWM44CR	PWM44 Control Register	FC84H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM44HLD	PWM44 Level Holding Control Register	FC85H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM45T1H	PWM45T1 High Byte	FC88H	-								x000,0000
PWM45T1L	PWM45T1 Low Byte	FC89H									0000,0000
PWM45T2H	PWM45T2 High Byte	FC8AH	-								x000,0000
PWM45T2L	PWM45T2 Low Byte	FC8BH									0000,0000
PWM45CR	PWM45 Control Register	FC8CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM45HLD	PWM45 Level Holding Control Register	FC8DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM46T1H	PWM46T1 High Byte	FC90H	-								x000,0000
PWM46T1L	PWM46T1 Low Byte	FC91H									0000,0000
PWM46T2H	PWM46T2 High Byte	FC92H	-								x000,0000
PWM46T2L	PWM46T2 Low Byte	FC93H									0000,0000
PWM46CR	PWM46 Control Register	FC94H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM46HLD	PWM46 Level Holding Control Register	FC95H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM47T1H	PWM47T1 High Byte	FC98H	-								x000,0000
PWM47T1L	PWM47T1 Low Byte	FC99H									0000,0000
PWM47T2H	PWM47T2 High Byte	FC9AH	-								x000,0000
PWM47T2L	PWM47T2 Low Byte	FC9BH									0000,0000
PWM47CR	PWM47 Control Register	FC9CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM47HLD	PWM47 Level Holding Control Register	FC9DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM5CH	PWM5 Counter High Byte	FCA0H	-								x000,0000
PWM5CL	PWM5 Counter Low Byte	FCA1H									0000,0000
PWM5CKS	PWM5 Clock Select Register	FCA2H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000
PWM5IF	PWM5 Interrupt Flag Register	FCA5H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM5FDCR	PWM5 Fault Detection Control Register	FCA6H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF	0000,0000
PWM50T1H	PWM50T1 High Byte	FCB0H	-								x000,0000
PWM50T1L	PWM50T1 Low Byte	FCB1H									0000,0000
PWM50T2H	PWM50T2 High Byte	FCB2H	-								x000,0000
PWM50T2L	PWM50T2 Low Byte	FCB3H									0000,0000
PWM50CR	PWM50 Control Register	FCB4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM50HLD	PWM50 Level Holding Control Register	FCB5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM51T1H	PWM51T1 High Byte	FCB8H	-								x000,0000
PWM51T1L	PWM51T1 Low Byte	FCB9H									0000,0000
PWM51T2H	PWM51T2 High Byte	FCBAH	-								x000,0000
PWM51T2L	PWM51T2 Low Byte	FCBBH									0000,0000
PWM51CR	PWM51 Control Register	FCBCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM51HLD	PWM51 Level Holding Control Register	FCBDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM52T1H	PWM52T1 High Byte	FCC0H	-								x000,0000
PWM52T1L	PWM52T1 Low Byte	FCC1H									0000,0000
PWM52T2H	PWM52T2 High Byte	FCC2H	-								x000,0000
PWM52T2L	PWM52T2 Low Byte	FCC3H									0000,0000
PWM52CR	PWM52 Control Register	FCC4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM52HLD	PWM52 Level Holding Control Register	FCC5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM53T1H	PWM53T1 High Byte	FCC8H	-								x000,0000
PWM53T1L	PWM53T1 Low Byte	FCC9H									0000,0000
PWM53T2H	PWM53T2 High Byte	FCCAH	-								x000,0000
PWM53T2L	PWM53T2 Low Byte	FCCBH									0000,0000

PWM53CR	PWM53 Control Register	FCCCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM53HLD	PWM53 Level Holding Control Register	FCCDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM54T1H	PWM54T1 High Byte	FCD0H	-								x000,0000
PWM54T1L	PWM54T1 Low Byte	FCD1H									0000,0000
PWM54T2H	PWM54T2 High Byte	FCD2H	-								x000,0000
PWM54T2L	PWM54T2 Low Byte	FCD3H									0000,0000
PWM54CR	PWM54 Control Register	FCD4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM54HLD	PWM54 Level Holding Control Register	FCD5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM55T1H	PWM55T1 High Byte	FCD8H	-								x000,0000
PWM55T1L	PWM55T1 Low Byte	FCD9H									0000,0000
PWM55T2H	PWM55T2 High Byte	FCDAH	-								x000,0000
PWM55T2L	PWM55T2 Low Byte	FCDBH									0000,0000
PWM55CR	PWM55 Control Register	FCDCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM55HLD	PWM55 Level Holding Control Register	FCDDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM56T1H	PWM56T1 High Byte	FCE0H	-								x000,0000
PWM56T1L	PWM56T1 Low Byte	FCE1H									0000,0000
PWM56T2H	PWM56T2 High Byte	FCE2H	-								x000,0000
PWM56T2L	PWM56T2 Low Byte	FCE3H									0000,0000
PWM56CR	PWM56 Control Register	FCE4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM56HLD	PWM56 Level Holding Control Register	FCE5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM57T1H	PWM57T1 High Byte	FCE8H	-								x000,0000
PWM57T1L	PWM57T1 Low Byte	FCE9H									0000,0000
PWM57T2H	PWM57T2 High Byte	FCEAH	-								x000,0000
PWM57T2L	PWM57T2 Low Byte	FCEBH									0000,0000
PWM57CR	PWM57 Control Register	FCECH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM57HLD	PWM57 Level Holding Control Register	FCEDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
MD3	MDU Data Register	FCF0H	MD3[7:0]								0000,0000
MD2	MDU Data Register	FCF1H	MD2[7:0]								0000,0000
MD1	MDU Data Register	FCF2H	MD1[7:0]								0000,0000
MD0	MDU Data Register	FCF3H	MD0[7:0]								0000,0000
MD5	MDU Data Register	FCF4H	MD5[7:0]								0000,0000
MD4	MDU Data Register	FCF5H	MD4[7:0]								0000,0000
ARCON	MDU Mode Control Register	FCF6H	MODE[2:0]			SC[4:0]					0000,0000
OPCON	MDU Operation Control Register	FCF7H	-	MDOV	-	-	-	-	RST	ENOP	0000,0000
COMEN	COM Enable Register	FB00H	C7EN	C6EN	C5EN	C4EN	C3EN	C2EN	C1EN	C0EN	0000,0000
SEGENL	SEG Enable Register	FB01H	S7EN	S6EN	S5EN	S4EN	S3EN	S2EN	S1EN	S0EN	0000,0000
SEGENH	SEG Enable Register	FB02H	S15EN	S14EN	S13EN	S12EN	S11EN	S10EN	S9EN	S8EN	0000,0000
LEDCTRL	LED Control Register	FB03H	LEDON	-	LEDMODE[1:0]		-	LEDDUTY[2:0]			0000,0000
LEDCKS	LED Clock Divide Register	FB04H									0000,0001
COM0_DA_L	Common Anode Mode Disply	FB10H									0000,0000
COM1_DA_L	Common Anode Mode Disply	FB11H									0000,0000
COM2_DA_L	Common Anode Mode Disply	FB12H									0000,0000
COM3_DA_L	Common Anode Mode Disply	FB13H									0000,0000
COM4_DA_L	Common Anode Mode Disply	FB14H									0000,0000
COM5_DA_L	Common Anode Mode Disply	FB15H									0000,0000
COM6_DA_L	Common Anode Mode Disply	FB16H									0000,0000
COM7_DA_L	Common Anode Mode Disply	FB17H									0000,0000
COM0_DA_H	Common Cathode Mode Disply	FB18H									0000,0000
COM1_DA_H	Common Cathode Mode Disply	FB19H									0000,0000
COM2_DA_H	Common Cathode Mode Disply	FB1AH									0000,0000
COM3_DA_H	Common Cathode Mode Disply	FB1BH									0000,0000
COM4_DA_H	Common Cathode Mode Disply	FB1CH									0000,0000
COM5_DA_H	Common Cathode Mode Disply	FB1DH									0000,0000
COM6_DA_H	Common Cathode Mode Disply	FB1EH									0000,0000
COM7_DA_H	Common Cathode Mode Disply	FB1FH									0000,0000
COM0_DC_L	Common Anode Mode Disply	FB20H									0000,0000
COM1_DC_L	Common Anode Mode Disply	FB21H									0000,0000
COM2_DC_L	Common Anode Mode Disply	FB22H									0000,0000
COM3_DC_L	Common Anode Mode Disply	FB23H									0000,0000
COM4_DC_L	Common Anode Mode Disply	FB24H									0000,0000
COM5_DC_L	Common Anode Mode Disply	FB25H									0000,0000
COM6_DC_L	Common Anode Mode Disply	FB26H									0000,0000
COM7_DC_L	Common Anode Mode Disply	FB27H									0000,0000
COM0_DC_H	Common Cathode Mode Disply	FB28H									0000,0000
COM1_DC_H	Common Cathode Mode Disply	FB29H									0000,0000
COM2_DC_H	Common Cathode Mode Disply	FB2AH									0000,0000
COM3_DC_H	Common Cathode Mode Disply	FB2BH									0000,0000
COM4_DC_H	Common Cathode Mode Disply	FB2CH									0000,0000
COM5_DC_H	Common Cathode Mode Disply	FB2DH									0000,0000
COM6_DC_H	Common Cathode Mode Disply	FB2EH									0000,0000
COM7_DC_H	Common Cathode Mode Disply	FB2FH									0000,0000
TSCHEN1	Touch Key Enable Register 1	FB40H	TKEN7	TKEN6	TKEN5	TKEN4	TKEN3	TKEN2	TKEN1	TKEN0	0000,0000
TSCHEN2	Touch Key Enable Register 2	FB41H	TKEN15	TKEN14	TKEN13	TKEN12	TKEN11	TKEN10	TKEN9	TKEN8	0000,0000
TSCFG1	Touch Key Configuration Register 1	FB42H	-	SCR[2:0]			-	DT[2:0]			0000,0000
TSCFG2	Touch Key Configuration Register 2	FB43H	-	-	-	-	-	-	TSVR[1:0]		0000,0000
TSWUTC	Touch Key Wakeup Control Register	FB44H									0000,0000

TSCTRL	Touch Key Control Register	FB45H	TSGO	SINGLE	TSWAIT	TSWUCS	TSDCEN	TSWUEN	TSSAMP[1:0]	0000,0000
TSSTA1	Touch Key Status Register 1	FB46H	LEDWK	-	-	-	TSWKCHN[3:0]			0000,0000
TSSTA2	Touch Key Status Register 2	FB47H	TSIF	TSDOV	-	-	TSDNCHN[3:0]			0000,0000
TSRT	Touch Key Time Control Register	FB48H								0000,0000
TSDATH	Touch Key Data High Byte	FB49H								0000,0000
TSDATL	Touch Key Data Low Byte	FB4AH								0000,0000
TSTH00H	Touch Key0 Threshold High Byte	FB50H								0000,0000
TSTH00L	Touch Key0 Threshold Low Byte	FB51H								0000,0000
TSTH01H	Touch Key1 Threshold High Byte	FB52H								0000,0000
TSTH01L	Touch Key1 Threshold Low Byte	FB53H								0000,0000
TSTH02H	Touch Key2 Threshold High Byte	FB54H								0000,0000
TSTH02L	Touch Key2 Threshold Low Byte	FB55H								0000,0000
TSTH03H	Touch Key3 Threshold High Byte	FB56H								0000,0000
TSTH03L	Touch Key3 Threshold Low Byte	FB57H								0000,0000
TSTH04H	Touch Key4 Threshold High Byte	FB58H								0000,0000
TSTH04L	Touch Key4 Threshold Low Byte	FB59H								0000,0000
TSTH05H	Touch Key5 Threshold High Byte	FB5AH								0000,0000
TSTH05L	Touch Key5 Threshold Low Byte	FB5BH								0000,0000
TSTH06H	Touch Key6 Threshold High Byte	FB5CH								0000,0000
TSTH06L	Touch Key6 Threshold Low Byte	FB5DH								0000,0000
TSTH07H	Touch Key7 Threshold High Byte	FB5EH								0000,0000
TSTH07L	Touch Key7 Threshold Low Byte	FB5FH								0000,0000
TSTH08H	Touch Key8 Threshold High Byte	FB60H								0000,0000
TSTH08L	Touch Key8 Threshold Low Byte	FB61H								0000,0000
TSTH09H	Touch Key9 Threshold High Byte	FB62H								0000,0000
TSTH09L	Touch Key9 Threshold Low Byte	FB63H								0000,0000
TSTH10H	Touch Key10 Threshold High Byte	FB64H								0000,0000
TSTH10L	Touch Key10 Threshold Low Byte	FB65H								0000,0000
TSTH11H	Touch Key11 Threshold High Byte	FB66H								0000,0000
TSTH11L	Touch Key11 Threshold Low Byte	FB67H								0000,0000
TSTH12H	Touch Key12 Threshold High Byte	FB68H								0000,0000
TSTH12L	Touch Key12 Threshold Low Byte	FB69H								0000,0000
TSTH13H	Touch Key13 Threshold High Byte	FB6AH								0000,0000
TSTH13L	Touch Key13 Threshold Low Byte	FB6BH								0000,0000
TSTH14H	Touch Key14 Threshold High Byte	FB6CH								0000,0000
TSTH14L	Touch Key14 Threshold Low Byte	FB6DH								0000,0000
TSTH15H	Touch Key15 Threshold High Byte	FB6EH								0000,0000
TSTH15L	Touch Key15 Threshold Low Byte	FB6FH								0000,0000

Note: The meaning of the initial value of the special function register

0: The initial value is 0.

1: The initial value is l.

n: The initial value is related to the hardware options during ISP download.

x: This bit does not exist, and the initial value is uncertain.

STC MCU

9 I/O Ports

Product line	Max I/O lines
STC8G1K08family	**18**
STC8G1K08-8Pin family	**6**
STC8G1K08A family	**6**
STC8G2K64S4 family	**45**
STC8G2K64S2 family	**45**
STC8G1K08T family	**16**
STC15H2K64S4 family	**42**

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

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

9.1 Registers Related to I/O

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
P0	Port 0	80H	P07	P06	P05	P04	P03	P02	P01	P00	1111,1111
P1	Port 1	90H	P17	P16	P15	P14	P13	P12	P11	P10	1111,1111
P2	Port 2	A0H	P27	P26	P25	P24	P23	P22	P21	P20	1111,1111
P3	Port 3	B0H	P37	P36	P35	P34	P33	P32	P31	P30	1111,1111
P4	Port 4	C0H	P47	P46	P45	P44	P43	P42	P41	P40	1111,1111
P5	Port 5	C8H	-	-	P55	P54	P53	P52	P51	P50	xx11,1111
P0M1	Port 0 mode register 1	93H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1	1111,1111
P0M0	Port 0 mode register 0	94H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0	0000,0000
P1M1	Port 1 mode register 1	91H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1	1111,1111
P1M0	Port 1 mode register 0	92H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0	0000,0000
P2M1	Port 2 mode register 1	95H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1	1111,1111
P2M0	Port 2 mode register 0	96H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0	0000,0000
P3M1	Port 3 mode register 1	B1H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1	n111,1100
P3M0	Port 3 mode register 0	B2H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0	n000,0000
P4M1	Port 4 mode register 1	B3H	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1	1111,1111
P4M0	Port 4 mode register 0	B4H	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0	0000,0000
P5M1	Port 5 mode register 1	C9H	-	-	P55M1	P54M1	P53M1	P52M1	P51M1	P50M1	xx11,1111
P5M0	Port 5 mode register 0	CAH	-	-	P55M0	P54M0	P53M0	P52M0	P51M0	P50M0	xx00,0000

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
P0PU	Port0 Pull-up resistor control register	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU	0000,0000
P1PU	Port1 Pull-up resistor control register	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU	0000,0000
P2PU	Port2 Pull-up resistor control register	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU	0000,0000
P3PU	Port3 Pull-up resistor control register	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU	0000,0000
P4PU	Port4 Pull-up resistor control register	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU	0000,0000
P5PU	Port5 Pull-up Resistor Control Register	FE15H	-	-	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU	xx00,0000
P0NCS	Port0 Schmitt Trigger Control Register	FE18H	P07NCS	P06NCS	P05NCS	P04NCS	P03NCS	P02NCS	P01NCS	P00NCS	0000,0000
P1NCS	Port1 Schmitt Trigger Control Register	FE19H	P17NCS	P16NCS	P15NCS	P14NCS	P13NCS	P12NCS	P11NCS	P10NCS	0000,0000
P2NCS	Port2 Schmitt Trigger Control Register	FE1AH	P27NCS	P26NCS	P25NCS	P24NCS	P23NCS	P22NCS	P21NCS	P20NCS	0000,0000
P3NCS	Port3 Schmitt Trigger Control Register	FE1BH	P37NCS	P36NCS	P35NCS	P34NCS	P33NCS	P32NCS	P31NCS	P30NCS	0000,0000

P4NCS	Port4 Schmitt Trigger Control Register	FE1CH	P47NCS	P46NCS	P45NCS	P44NCS	P43NCS	P42NCS	P41NCS	P40NCS	0000,0000
P5NCS	Port5 Schmitt Trigger Control Register	FE1DH	-	-	P55NCS	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS	xx00,0000
P0SR	Port0 Level Shift Rate Register	FE20H	P07SR	P06SR	P05SR	P04SR	P03SR	P02SR	P01SR	P00SR	1111,1111
P1SR	Port1 Level Shift Rate Register	FE21H	P17SR	P16SR	P15SR	P14SR	P13SR	P12SR	P11SR	P10SR	1111,1111
P2SR	Port2 Level Shift Rate Register	FE22H	P27SR	P26SR	P25SR	P24SR	P23SR	P22SR	P21SR	P20SR	1111,1111
P3SR	Port3 Level Shift Rate Register	FE23H	P37SR	P36SR	P35SR	P34SR	P33SR	P32SR	P31SR	P30SR	1111,1111
P4SR	Port4 Level Shift Rate Register	FE24H	P47SR	P46SR	P45SR	P44SR	P43SR	P42SR	P41SR	P40SR	1111,1111
P5SR	Port5 Level Shift Rate Register	FE25H	-	-	P55SR	P54SR	P53SR	P52SR	P51SR	P50SR	xx11,1111
P0DR	Port0 Drive Current Control Register	FE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR	1111,1111
P1DR	Port1 Drive Current Control Register	FE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR	1111,1111
P2DR	Port2 Drive Current Control Register	FE2AH	P27DR	P26DR	P25DR	P24DR	P23DR	P22DR	P21DR	P20DR	1111,1111
P3DR	Port3 Drive Current Control Register	FE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR	1111,1111
P4DR	Port4 Drive Current Control Register	FE2CH	P47DR	P46DR	P45DR	P44DR	P43DR	P42DR	P41DR	P40DR	1111,1111
P5DR	P5 口驱动电流控制寄存器	FE2DH	-	-	P55DR	P54DR	P53DR	P52DR	P51DR	P50DR	xx11,1111
P0IE	Port0 Input Enable Control Register	FE30H	P07IE	P06IE	P05IE	P04IE	P03IE	P02IE	P01IE	P00IE	1111,1111
P1IE	Port1 Input Enable Control Register	FE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE	1111,1111
P3IE	Port3 Input Enable Control Register	FE33H	P37IE	P36IE	P35IE	P34IE	P33IE	P32IE	P31IE	P30IE	1111,1111

9.1.1 Port Data Register (Px)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
P0	80H	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
P1	90H	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
P2	A0H	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
P3	B0H	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
P4	C0H	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
P5	C8H	-	-	P5.5	P5.4	P5.3	P5.2	P5.1	P5.0

Read and write port status

Write 0: Output low to port buffer.

Write 1: Output high to port buffer.

Read: Read the level on the port pin directly.

9.1.2 Ports Mode Registers (PxM0, PxM1)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
P0M0	94H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1
P0M1	93H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0
P1M0	92H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1
P1M1	91H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0
P2M0	96H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1
P2M1	95H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0
P3M0	B2H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1
P3M1	B1H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0
P4M0	B4H	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1
P4M1	B3H	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0
P5M0	CAH	-	-	P55M1	P54M1	P53M1	P52M1	P51M1	P50M1
P5M1	C9H	-	-	P55M0	P54M0	P53M0	P52M0	P51M0	P50M0

Configure the mode of the ports as shown below.

PnM1.x	PnM0.x	Pn.x operating mode
0	0	quasi bidirectional mode
0	1	push-pull output mode
1	0	high-impedance input mode

1	1	open drain mode

Note: When an I/0 port is selected as the ADC input channel, the PxMO/PxM1 register must be set to set the I/0 port mode to input mode. In addition, if the MCU still needs to enable the ADC channel after entering the power-down mode/clock stop mode, you need to set the PxIE register to turn off the digital input to ensure that there will be no additional power consumption

9.1.3 Pull-up Resistor Control Registers (PxPU)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
P0PU	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU
P1PU	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU
P2PU	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU
P3PU	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU
P4PU	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU
P5PU	FE15H	-	-	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU

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

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

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

9.1.4 Schmitt Trigger Control Registers (PxNCS)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
P0NCS	FE18H	P07NCS	P06NCS	P05NCS	P04NCS	P03NCS	P02NCS	P01NCS	P00NCS
P1NCS	FE19H	P17NCS	P16NCS	P15NCS	P14NCS	P13NCS	P12NCS	P11NCS	P10NCS
P2NCS	FE1AH	P27NCS	P26NCS	P25NCS	P24NCS	P23NCS	P22NCS	P21NCS	P20NCS
P3NCS	FE1BH	P37NCS	P36NCS	P35NCS	P34NCS	P33NCS	P32NCS	P31NCS	P30NCS
P4NCS	FE1CH	P47NCS	P46NCS	P45NCS	P44NCS	P43NCS	P42NCS	P41NCS	P40NCS
P5NCS	FE1DH	-	-	P55NCS	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS

Schmitt trigger control bit:

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

1: Disable schmitt trigger function on the port.

9.1.5 Level Togglling Speed Control Registers (PxSR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
P0SR	FE20H	P07SR	P06SR	P05SR	P04SR	P03SR	P02SR	P01SR	P00SR
P1SR	FE21H	P17SR	P16SR	P15SR	P14SR	P13SR	P12SR	P11SR	P10SR
P2SR	FE22H	P27SR	P26SR	P25SR	P24SR	P23SR	P22SR	P21SR	P20SR
P3SR	FE23H	P37SR	P36SR	P35SR	P34SR	P33SR	P32SR	P31SR	P30SR
P4SR	FE24H	P47SR	P46SR	P45SR	P44SR	P43SR	P42SR	P41SR	P40SR
P5SR	FE25H	-	-	P55SR	P54SR	P53SR	P52SR	P51SR	P50SR

Level toggling speed control bits:

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

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

9.1.6 Drive Current Control Registers (PxDR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0

P0DR	FE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR
P1DR	FE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR
P2DR	FE2AH	P27DR	P26DR	P25DR	P24DR	P23DR	P22DR	P21DR	P20DR
P3DR	FE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR
P4DR	FE2CH	P47DR	P46DR	P45DR	P44DR	P43DR	P42DR	P41DR	P40DR
P5DR	FE2DH	-	-	P55DR	P54DR	P53DR	P52DR	P51DR	P50DR

Drive capability control bit:

0: Enhanced drive ability

1: General drive ability

9.1.7 Digital Signal Input Enabling Control Registers (PxIE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
P0IE	FE30H	P07IE	P06IE	P05IE	P04IE	P03IE	P02IE	P01IE	P00IE
P1IE	FE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE
P3IE	FE33H	P37IE	P36IE	P35IE	P34IE	P33IE	P32IE	P31IE	P30IE
P5IE	FE35H	-	-	P55IE	P54IE	P53IE	P52IE	P51IE	P50IE

Digital signal input enabling control bit:

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

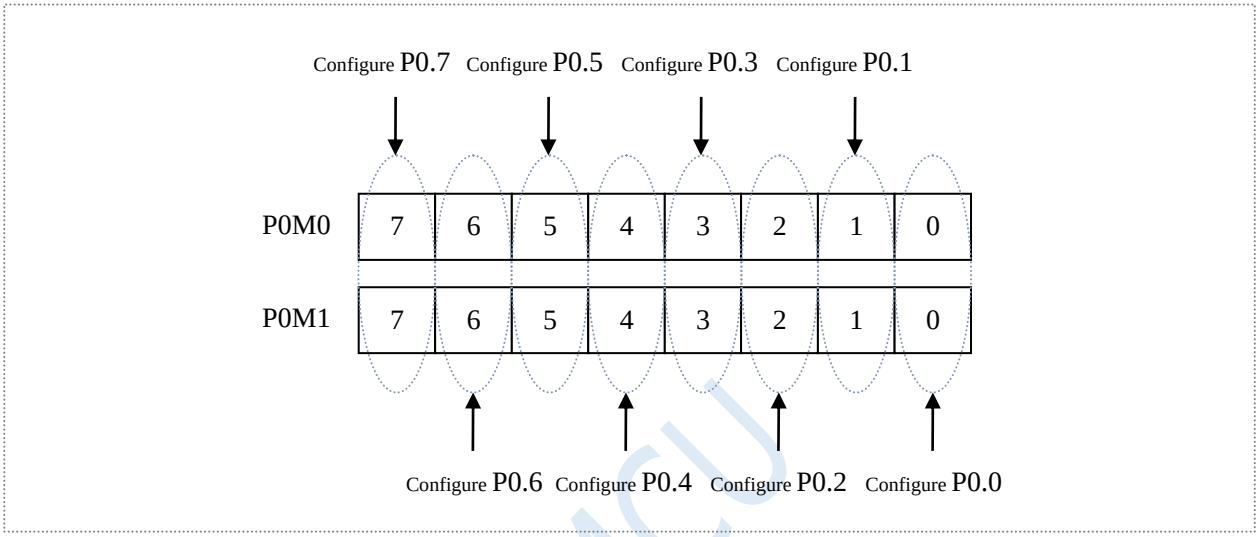
1: Enable digital signal input. If the I/O is used as a digital port, it must be set to 1, otherwise the MCU cannot read the level of the external port.

STC MCU

9.2 I/O Ports Configurations

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

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



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

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

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

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

Note:

Although each I/0 pin can withstand a 20mA sink current in weak pull-up (quasi-bidirectional)/strong push-pull output/open-drain mode (current-limiting resistance is still required, such as 1K, 560Ω, 472Ω, etc.), it can output a current of 20mA in the strong push-pull output mode, (current limiting resistor should also be added), the working current of the whole chip is recommended not to exceed 70mA, that is, the current flowing in from Vcc is recommended not to exceed 70mA, and the current flowing from Gnd is recommended not to exceed 70mA, the overall inflow/outflow current is recommended not to exceed 70mA. (For STC8G1K08A series and STC8GIK08-8Pin series, the working current of the entire chip is recommended not to exceed 35mA, that is, the current flowing in from Vcc is recommended not to exceed 35mA, the current flowing from Gnd is recommended not to exceed 35mA,

and the overall inflow/outflow current is recommended not to exceed 35mA.)

9.3 I/O Ports Structure

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

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

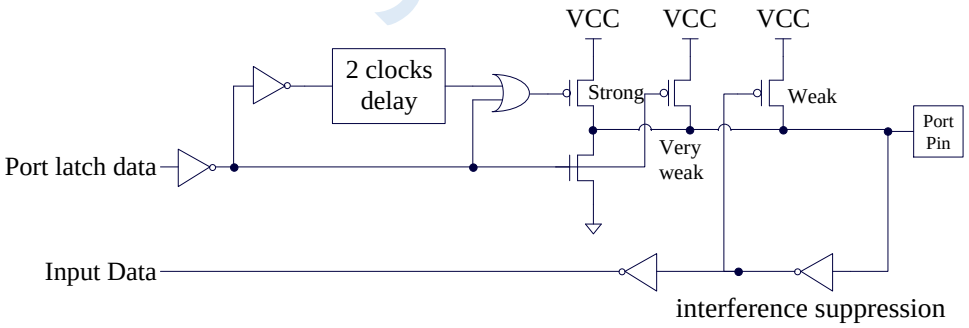
One of the three pull-up transistors, called "weak pull-up", is turned on when the port register is logic “1” and the pin itself is logic “1”. This pull-up transistor provides the basic drive current to make the quasi-bidirectional port output logic “1”. If one of the pin outputs logic “1” and the external device pulls it low, the weak pull-up transistor is off and the "very weak pull-up" maintains on. To pull the pin low, the external device must have sufficient sink capability to make the voltage on the pin drop below the threshold voltage. For a 5V microcontroller, the current of "weak pull-up" transistor is about 250uA; for a 3.3V microcontroller, the current of "weak pull-up" transistor is about 150uA.

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

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

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

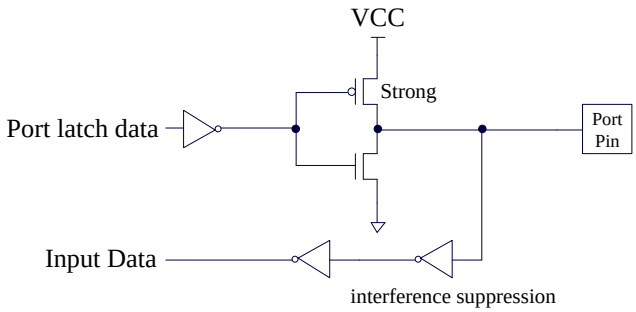
The structure of quasi-bidirectional port (weak pull-up) output is shown below:



9.3.2 Push-Pull Output

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

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

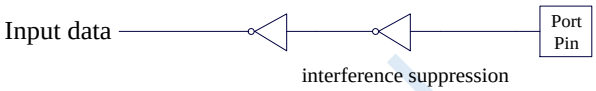


9.3.3 High Impedance Input

The current can neither flow in nor flow out.

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

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



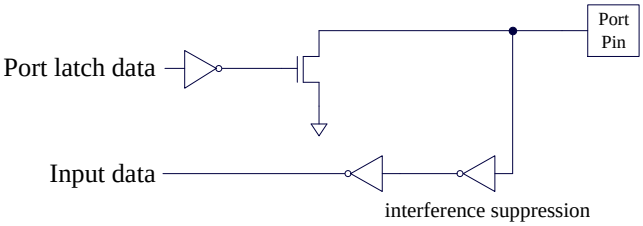
9.3.4 Open-Drain Output

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

All pull-up transistors are turned off in the open-drain output configuration when the port latch is logic “0”. There must be an external pull-up resistor in this configuration when the port outputs a logic high, typically the port pin is externally connected to VCC through a resistor. An open-drain I/O port pin can read the external state if the external pull-up resistor is connected, and the open-drain mode I/O port pin can be used as input mode. The pull-down structure in this way is the same as quasi-bidirectional mode.

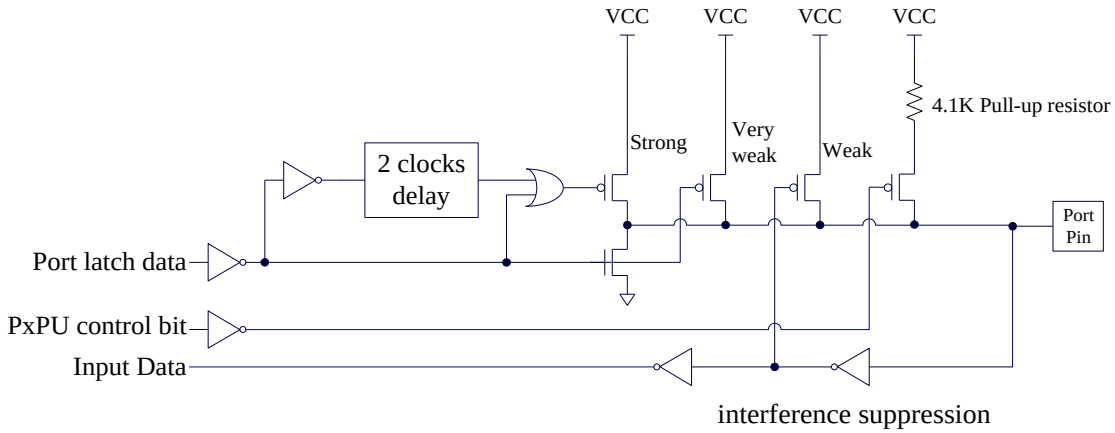
The open drain port has a Schmitt trigger input and an interference suppression circuit.

The structure of open drain port configuration is shown below:



9.3.5 4.1K Pull-up Resistor

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



Pull-up Resistor Control Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
P0PU	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU
P1PU	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU
P2PU	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU
P3PU	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU
P4PU	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU
P5PU	FE15H	-	-	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU

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

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

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

9.3.6 How to set the output speed of the IO port

When user needs the I/O port to output at a faster frequency, he can increase the I/O port drive current and increase the I/O port level conversion speed to increase the I/O port external output speed

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

9.3.7 How to set the drive current of the IO port

Setting the PxDR registers can be used to control the drive current of the I/O port. When it is set to 1, the I/O outputs a general drive current, and when it is set to 0, it outputs a strong drive current.

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

Setting PxSR registers can be used to control the I/O port level conversion speed, and setting PxDR registers can be used to control the IO port drive current.

When it is necessary to reduce the radiation of the I/O port to the outside, the PxSR registers need to be set to 1 to reduce the I/O port level conversion speed, and the PxDR registers need to be set to 1 to reduce the I/O drive current, then to reduce the radiation of the I/O port finally.

9.4 Example Routines

9.4.1 Port Mode Setting

C language code

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

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

sfr     P0M0    =   0x94;
sfr     P0M1    =   0x93;
sfr     P1M0    =   0x92;
sfr     P1M1    =   0x91;
sfr     P2M0    =   0x96;
sfr     P2M1    =   0x95;
sfr     P3M0    =   0xb2;
sfr     P3M1    =   0xb1;
sfr     P4M0    =   0xb4;
sfr     P4M1    =   0xb3;
sfr     P5M0    =   0xca;
sfr     P5M1    =   0xc9;
sfr     P6M0    =   0xcc;
sfr     P6M1    =   0xcb;
sfr     P7M0    =   0xe2;
sfr     P7M1    =   0xe1;

void main()
{
    P0M0 = 0x00;                    //Set P0.0 ~ P0.7 as bidirectional port mode
    P0M1 = 0x00;
    P1M0 = 0xff;                    //Set P1.0 ~ P1.7 as push-pull output mode
    P1M1 = 0x00;
    P2M0 = 0x00;                    //Set P2.0 ~ P2.7 as high impedance input mode
    P2M1 = 0xff;
    P3M0 = 0xff;                    //Set P3.0 ~ P3.7 as open-drain mode
    P3M1 = 0xff;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

```
P5M0        DATA    0CAH
P5M1        DATA    0C9H
P6M0        DATA    0CCH
P6M1        DATA    0CBH
P7M0        DATA    0E2H
P7M1        DATA    0E1H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH

            MOV     P0M0,#00H       ;Set P0.0 ~ P0.7 as bidirectional port mode
            MOV     P0M1,#00H
            MOV     P1M0,#0FFH      ;Set P1.0 ~ P1.7 as push-pull output mode
            MOV     P1M1,#00H
            MOV     P2M0,#00H       ;Set P2.0 ~ P2.7 as high impedance input mode
            MOV     P2M1,#0FFH
            MOV     P3M0,#0FFH      ;Set P3.0 ~ P3.7 as open-drain mode
            MOV     P3M1,#0FFH

            JMP     $

            END
```

9.4.2 Reading and Writing Operation of Bidirection Port

C language code

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

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

sfr     P0M0    =   0x94;
sfr     P0M1    =   0x93;
sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;
sbit    P00     =   P0^0;

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

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

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

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

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

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN

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

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

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

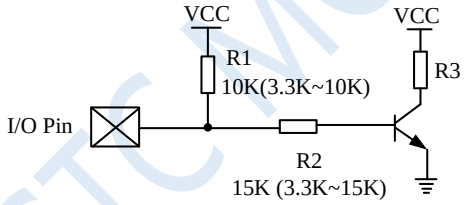
        SETB    P0.0                ;P0.0 output high level
        CLR     P0.0                ;P0.0 output low level

        SETB    P0.0                ;Enable the internal weak pull-up resistor before reading the port
        NOP                         ;Wait for two clocks
        NOP
        MOV     C,P0.0              ;Read port status

        JMP     $

        END
```

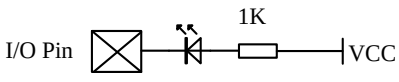
9.5 A Typical Circuit Controlled by Triode



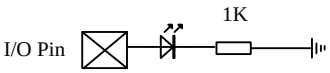
For pull-up control, it is recommended to add a pull-up resistor R1 (3.3K ～ 10K). If pull-up resistor R1 (3.3K ～ 10K) is not connected, it is recommended that the value of R2 be above 15K, or use strong push-pull output mode.

9.6 Typical Control Circuit of LED

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

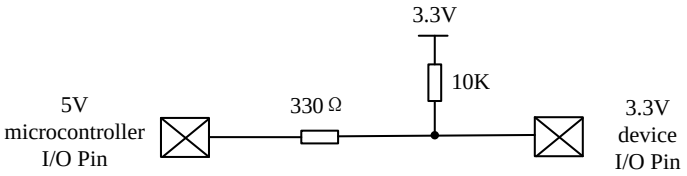


For push-pull (strong pull-up) I/O, you can drive the light-emitting diode with pull current mode.

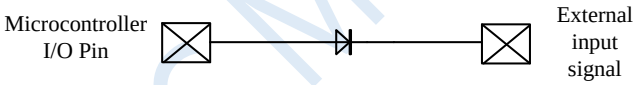


9.7 Interconnection of 3V/5V Devices in Mixed Voltage Power Supply System

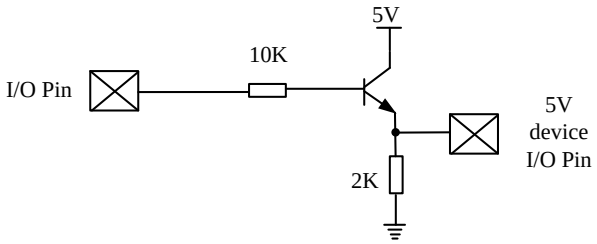
When STC's 5V microcontroller is connected to a 3.3V device, the corresponding I/O port of the 5V microcontroller can be connected with a 330Ω current limiting resistor to the 3.3V device I/O port in order to prevent the 3.3V device from withstanding 5V. The I/O port of the microcontroller is set to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



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



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

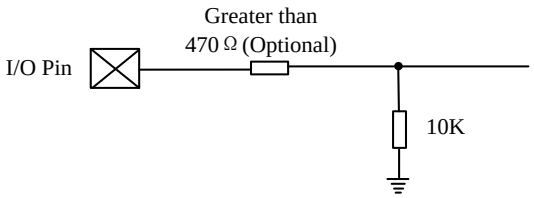


9.8 Make I/O Port Low When Power on Reset

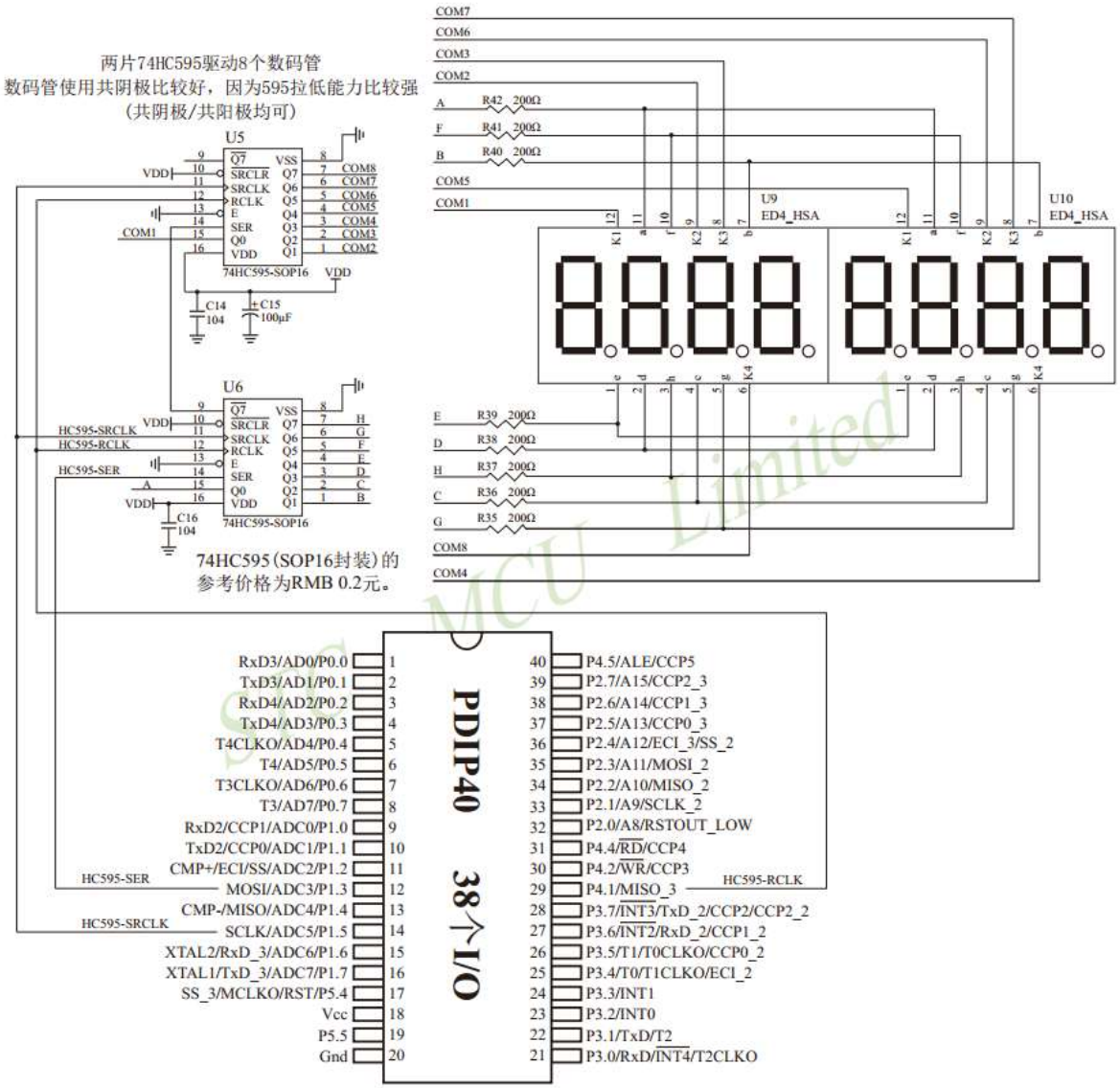
The general I/O port which is a weak pull-up (quasi-bidirectional) port will output high level when the traditional 8051 microcontroller is powered on and reset. Many practical applications require that certain I/O ports be low level output when powered on, otherwise the system (such as the motor) controlled by the microcontroller will malfunction. The STC microcontroller's I/O ports have weak pull-up output mode and strong push-pull output mode, which can easily solve this problem.

Now you can connect a pull-down resistor (1K, 2K or 3K) to the I/O port of STC microcontroller. The I/O port

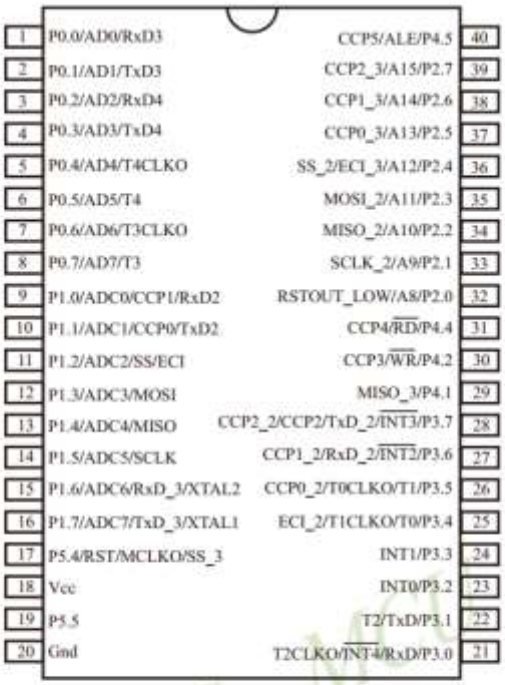
of the microcontroller is internal weak pull-up (quasi-bidirectional)/high-level output when the microcontroller is powered on reset. It cannot be pulled high because the internal pull-up capability is limited and the external pull-down resistor is small. Therefore the I/O port is externally low when power is reset. If you want to drive this I/O port to a high level, you can set this I/O port in strong push-pull output mode, under this circumstance, the I/O port drive current can reach 20mA, so the port can be definitely driven high.

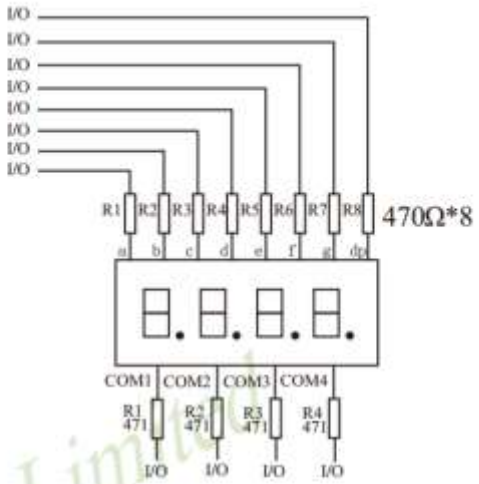


9.9 Circuit Diagram of Driving 8 Digital LEDs using 74HC595



9.10 Digital LEDs Driven Directly by I/O Port Circuit

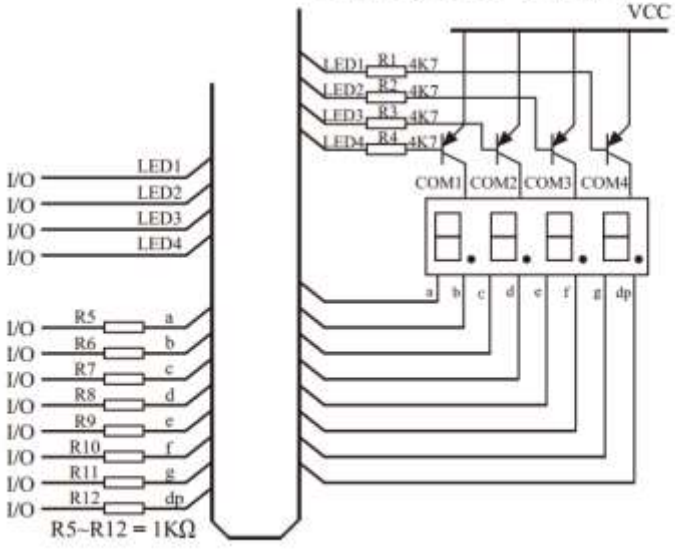




I/O口动态扫描驱动4个共阴极数码管参考电路图

I/O 口动态扫描驱动数码时，可以一次点亮一个数码管中的8段，但为降低功耗，建议可以一次只点亮其中的4段或者2段

I/O 口动态扫描驱动4个共阳极数码管参考电路图



9.11 LCD Segment LCD Driven Directly by I/O Port Circuit

An external LCD driver IC is needed when you design a product having segment LCD display requirement using MCU without LCD driver is used, which will increase cost. In fact, many small projects, such as a large number of small appliances, do not need to display a lot of segment codes. There are usually four 8 colons with

a decimal point or clock: ":", so if you use the IO port to scan and display directly, it will reduce costs and work more reliably.

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

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

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

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

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

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

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

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

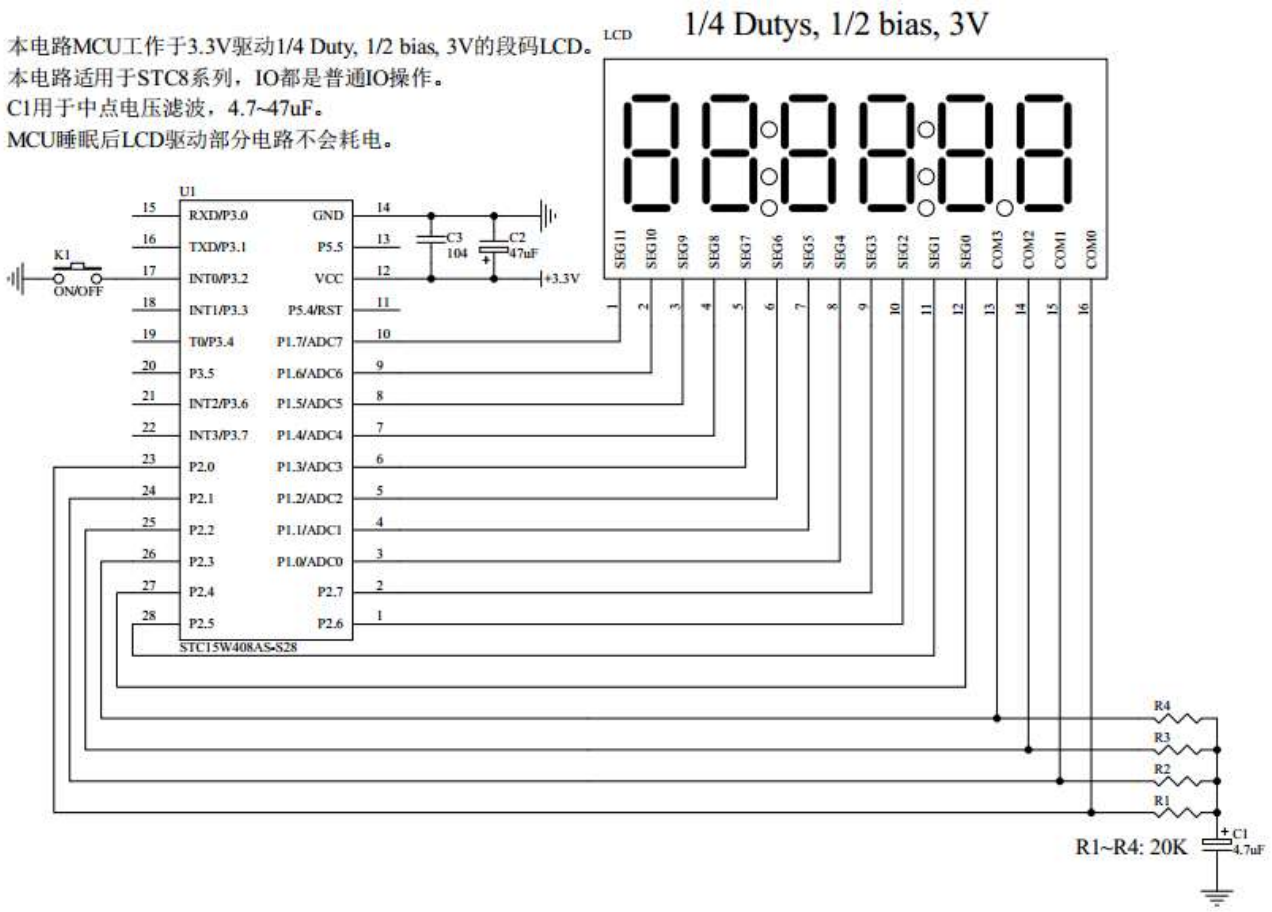


Figure 2 Segment name

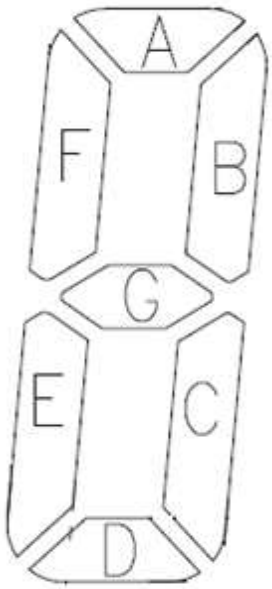


Figure 3 Principle of '1/4Duty, 1/2BIAS' scanning

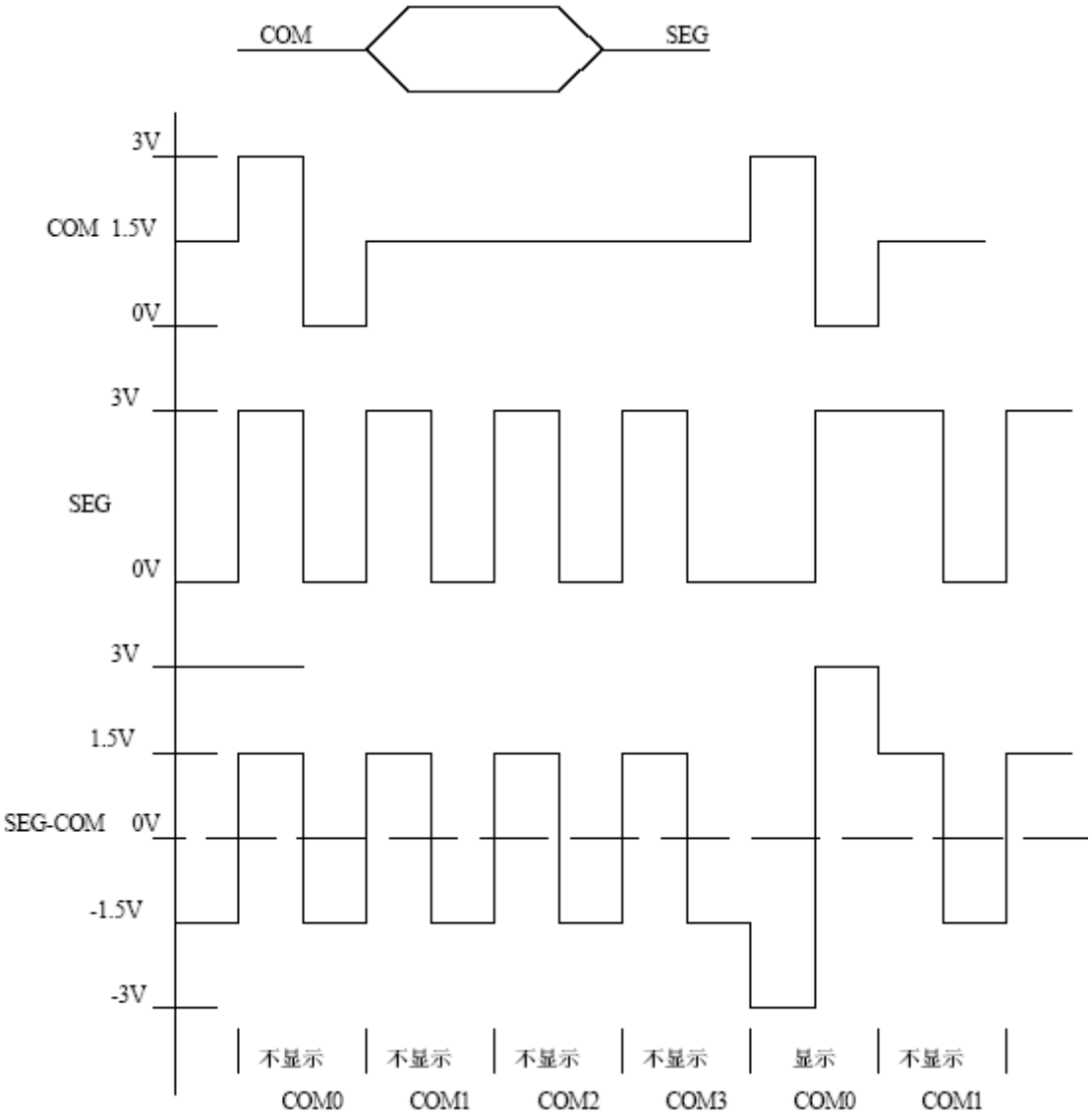


Figure 4 Driving circuit of '1/4Duty, 1/3BIAS, 3V' LCD

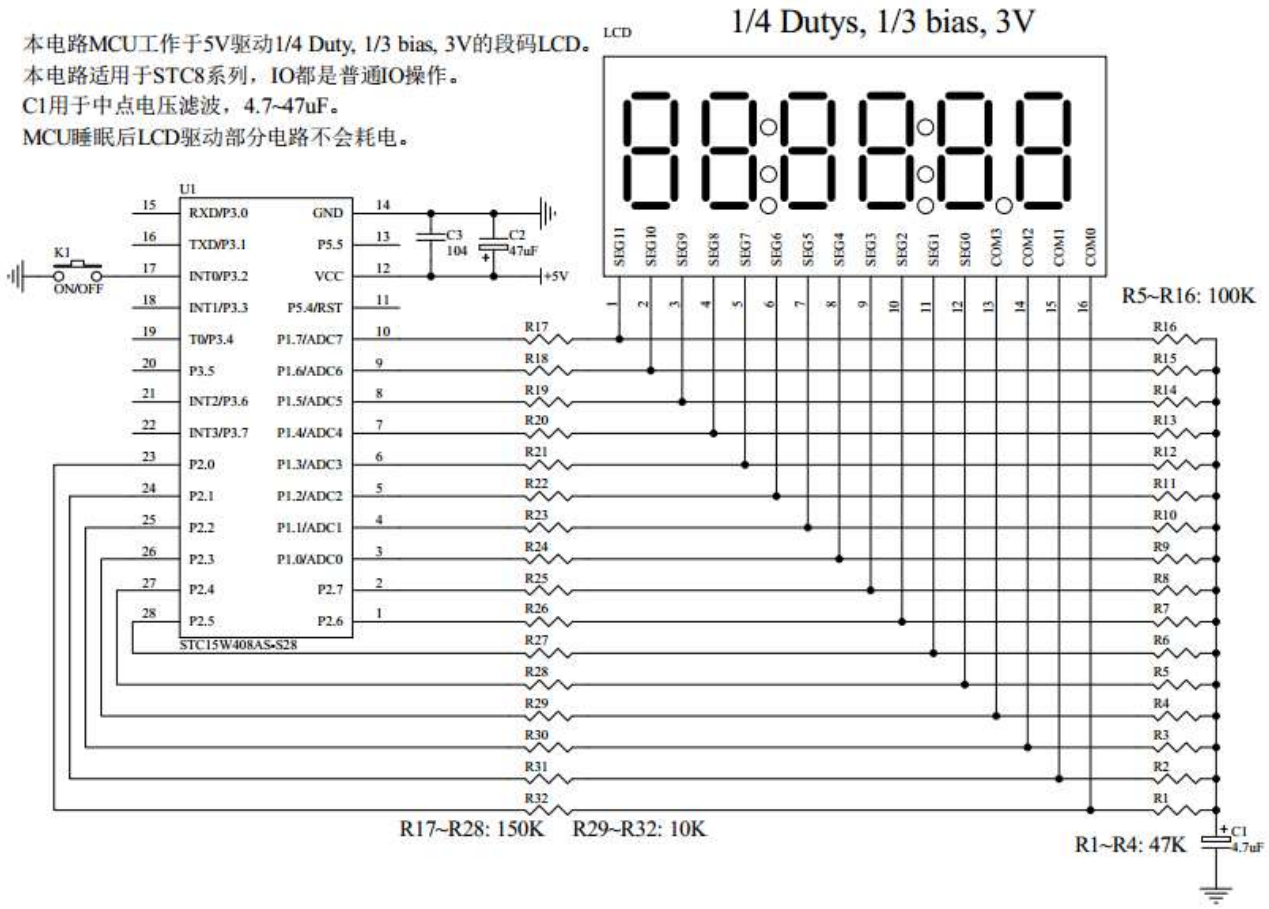
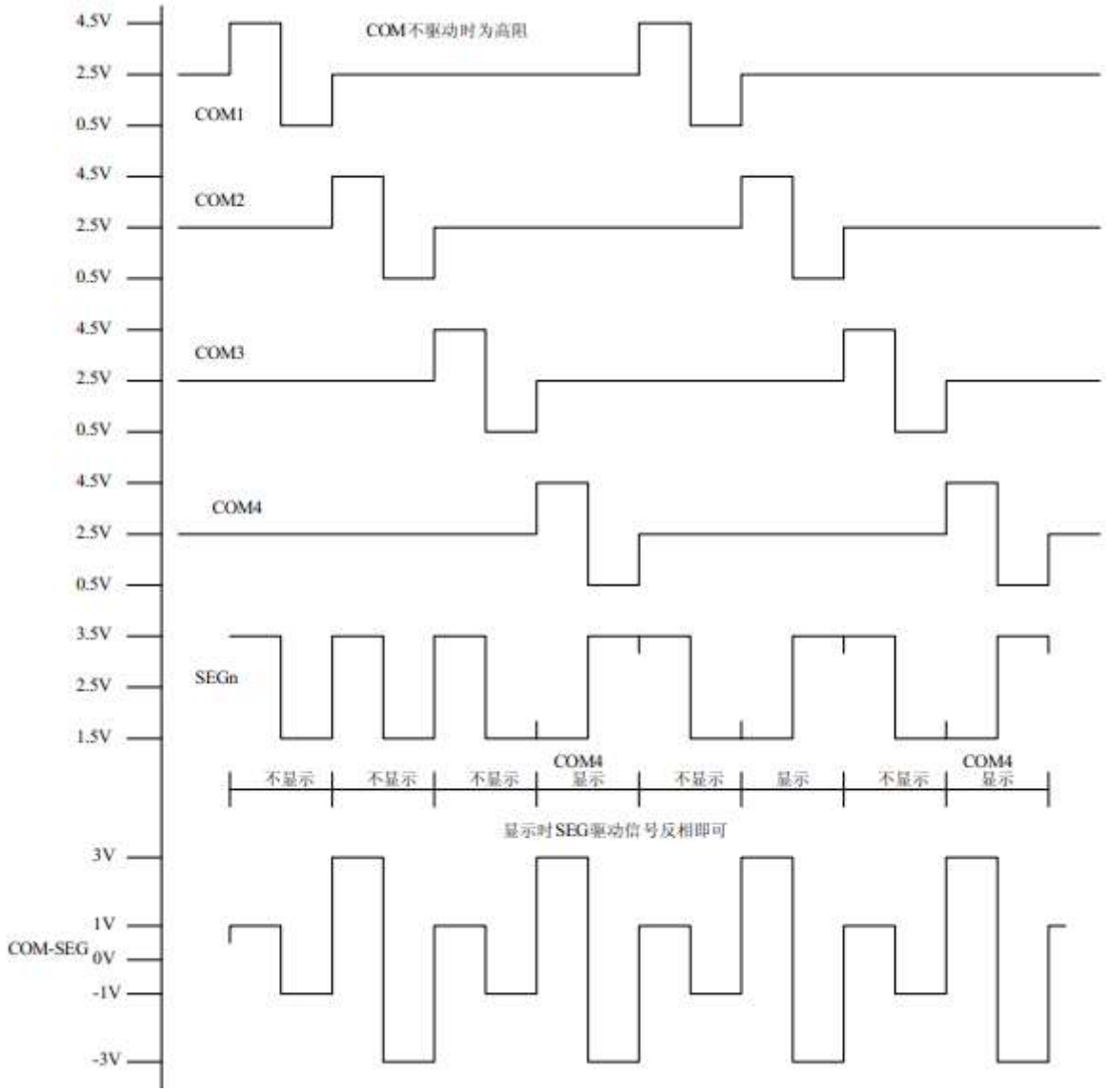


Figure 5 Principle of '1/4Duty, 1/3BIAS' scanning



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

Figure 6 LCD truth table and memory mapping table

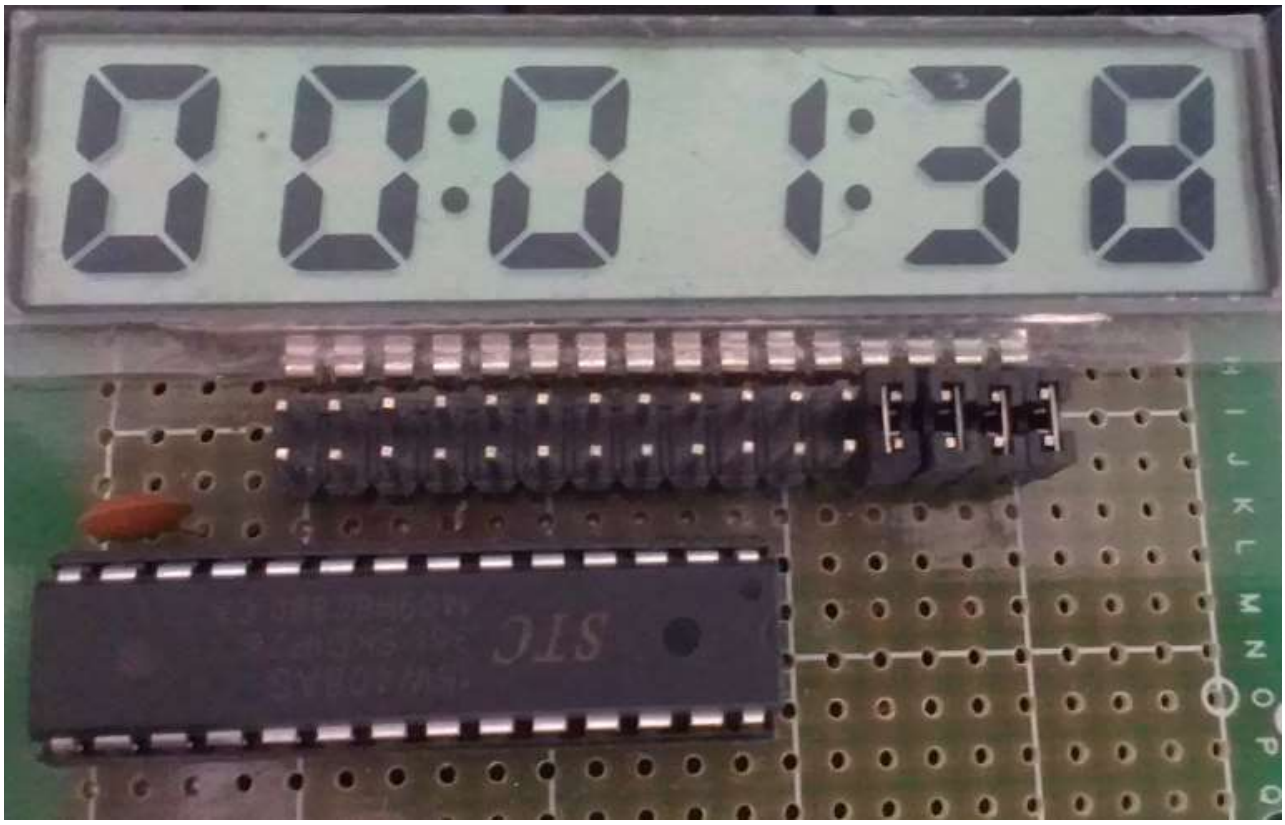
LCD真值表：

MCU PIN	P17	P16	P15	P14	P13	P12	P11	P10	P27	P26	P25	P24	P23	P22	P21	P20
LCD PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
LCD PIN name	SEG11	SEG10	SEG9	SEG8	SEG7	SEG6	SEG5	SEG4	SEG3	SEG2	SEG1	SEG0	COM3	COM2	COM1	COM0
	--	1D	2:	2D	2.	3D	4:	4D	4.	5D	5.	6D	COM3			
	1E	1C	2E	2C	3E	3C	4E	4C	5E	5C	6E	6C		COM2		
	1G	1B	2G	2B	3G	3B	4G	4B	5G	5B	6G	6B			COM1	
	1F	1A	2F	2A	3F	3A	4F	4A	5F	5A	6F	6A				COM0

显存影射表：

	B7	B6	B5	B4	B3	B2	B1	B0
buff[0]:	--	1D	2:	2D	2.	3D	4:	4D
buff[1]:	1E	1C	2E	2C	3E	3C	4E	4C
buff[2]:	1G	1B	2G	2B	3G	3B	4G	4B
buff[3]:	1F	1A	2F	2A	3F	3A	4F	4A
buff[4]:	4.	5D	5.	6D	--	--	--	--
buff[5]:	5E	5C	6E	6C	--	--	--	--
buff[6]:	5G	5B	6G	6B	--	--	--	--
buff[7]:	5F	5A	6F	6A	--	--	--	--

Figure 7: Driving effect photo



Only two functions are required in the LCD scanning program:

1. LCD segment code scan function

void LCD_scan(void)

The program calls this function at a certain interval, and it will display the contents of the LCD display buffer on the LCD. It takes 8 calling cycles to scan all of them. The calling interval is generally 1~ 2ms. The scanning cycle is 8ms if 1ms is used. The refresh rate is 125Hz.

2. LCD segment code display buffer loading function

void LCD_load(u8 n,u8 dat)

This function is used to put the displayed numbers or characters in the LCD display buffer. For example, LCD_load (1,6) is to display the number 6 at the first digit position. It supports the display of 0 ~ 9, A ~ F. You can add them by yourself if you need other characters.

In addition, macros are used to display, extinguish, or flash colons or decimal points.

Assembly code

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

;*************************************************************
P0M1        DATA    0x93
P0M0        DATA    0x94
P1M1        DATA    0x91
P1M0        DATA    0x92
P2M1        DATA    0x95
P2M0        DATA    0x96
P3M1        DATA    0xB1
P3M0        DATA    0xB2
P4M1        DATA    0xB3
P4M0        DATA    0xB4
P5M1        DATA    0xC9
P5M0        DATA    0xC
P6M1        DATA    0xCB
P6M0        DATA    0xCC
P7M1        DATA    0xE1
P7M0        DATA    0xE2
AUXR        DATA    0x8E
INT_CLKO    DATA    0x8F
IE2         DATA    0xAF
P4          DATA    0xC0
T2H         DATA    0xD6
T2L         DATA    0xD7

;*************************************************************
DIS_BLACK   EQU     010H
DIS_        EQU     011H
DIS_A       EQU     00AH
DIS_B       EQU     00BH
DIS_C       EQU     00CH
DIS_D       EQU     00DH
DIS_E       EQU     00EH
DIS_F       EQU     00FH

B_2ms       BIT     20H.0       ;2ms signal
B_Second    BIT     20H.1       ;second signal
cnt_500ms   DATA    30H
second      DATA    31H
minute      DATA    32H
hour        DATA    33H
scan_index  DATA    34H

LCD_buff    DATA    40H         ;40H~47H

;*************************************************************
            ORG     0000H
            LJMP    F_Main

            ORG     000BH
            LJMP    F_Timer0_Interrupt
```

STC MCU

```
;*************************************************************
                ORG     0100H
F_Main:
                CLR     A
                MOV     P3M1, A                 ;Set as a quasi-bidirectional port
                MOV     P3M0, A
                MOV     P5M1, A                 ;Set as a quasi-bidirectional port
                MOV     P5M0, A

                MOV     P1M1, #0                ; segments are set as push-pull output mode
                MOV     P1M0, #0ffh
                ANL     P2M1, #NOT 0f0h         ; segments are set as push-pull output mode
                ORL     P2M0, #0f0h
                ORL     P2M1, #0fH  ; All COM outputs are high impedance, and COM's voltage is the midpoint
                ANL     P2M0, #0f0H
                MOV     SP, #0D0H
                MOV     PSW, #0
                USING   0                       ;Select bank0 R0 ~ R7

;*************************************************************
                MOV     R2, #8
                MOV     R0, #LCD_buff
L_ClearLcdRam:
                MOV     @R0, #0
                INC     R0
                DJNZ    R2, L_ClearLcdRam

                LCALL   F_Timer0_init
                SETB    EA

;               ORL     LCD_buff, #020H         ;Display hour-minute division interval:
;               ORL     LCD_buff, #002H         ;Display minute-second division interval:

                MOV     hour, #12
                MOV     minute, #00
                MOV     second, #00
                LCALL   F_LoadRTC               ;Display time

;*************************************************************
L_Main_Loop:
                JNB     B_2ms, L_Main_Loop      ;2ms beat
                CLR     B_2ms

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

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

                CPL     B_Second
                JNB     B_Second, L_Main_Loop

                INC     second
                MOV     A, second
                CJNE    A, #60, L_Main_Load
                MOV     second, #0              ; reach to 1 minute
```

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

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

F_Timer0_init:
                CLR         TR0                     ; Stop counting
                ANL         TMOD, #0f0H
                SETB        ET0                     ; Enable interrupt
                ORL         TMOD, #0                ; Working mode 0: 16-bit auto-reload
                ANL         INT_CLKO, #NOT 0x01     ; Does not output clock
                ORL         AUXR, #0x80             ; 1T mode
                MOV         TH0, #HIGH (-22118)     ; 2ms
                MOV         TL0, #LOW  (-22118)     ;
                SETB        TR0                     ; Start operation
                RET

;*************************************************************
F_Timer0_Interrupt:         ;Timer0 1ms interrupt function
                PUSH        PSW                     ;push PSW into stack
                PUSH        ACC                     ;push ACC into stack
                PUSH        AR0
                PUSH        AR7
                PUSH        DPH
                PUSH        DPL

                LCALL       F_LCD_scan
                SETB        B_2ms

                POP         DPL
                POP         DPH
                POP         AR7
                POP         AR0
                POP         ACC                     ;pop ACC from stack
                POP         PSW                     ;pop PSW from stack
                RETI

;********************* Display time **************************
F_LoadRTC:
                MOV         R6, #1                  ;LCD_load(1,hour/10);
                MOV         A, hour
                MOV         B, #10
                DIV         AB
                MOV         R7, A
                LCALL       F_LCD_load              ;R6 is the number, which is 1~6, R7 is the number to be
displayed

                MOV         R6, #2                  ;LCD_load(2,hour%10);
                MOV         A, hour
                MOV         B, #10
```

```
                DIV     AB
                MOV     R7, B
                LCALL   F_LCD_load      ;R6 is the number, which is 1~6, R7 is the number to be
displayed

                MOV     R6, #3          ;LCD_load(3,minute/10);
                MOV     A, minute
                MOV     B, #10
                DIV     AB
                MOV     R7, A
                LCALL   F_LCD_load      ;R6 is the number, which is 1~6, R7 is the number to be
displayed

                MOV     R6, #4          ;LCD_load(4,minute%10);
                MOV     A, minute
                MOV     B, #10
                DIV     AB
                MOV     R7, B
                LCALL   F_LCD_load      ;R6 is the number, which is 1~6, R7 is the number to be
displayed

                MOV     R6, #5          ;LCD_load(5,second/10);
                MOV     A, second
                MOV     B, #10
                DIV     AB
                MOV     R7, A
                LCALL   F_LCD_load      ;R6 is the number, which is 1~6, R7 is the number to be
displayed

                MOV     R6, #6          ;LCD_load(6,second%10);
                MOV     A, second
                MOV     B, #10
                DIV     AB
                MOV     R7, B
                LCALL   F_LCD_load      ;R6 is the number, which is 1~6, R7 is the number to be
displayed

                RET

;*************************************************************
T_COM:
                DB      008H, 004H, 002H, 001H

F_LCD_scan:
                MOV     A, scan_index   ;j = scan_index >> 1;
                CLR     C
                RRC     A
                MOV     R7, A           ;R7 = j
                ADD     A, #LCD_buff
                MOV     R0, A           ;R0 = LCD_buff[j]
                ORL     P2M1, #00fH     ;All COM outputs are high impedance, and COM’s
voltage is the midpoint
                ANL     P2M0, #0f0H

                MOV     A, scan_index
                JNB     ACC.0, L_LCD_Scan2      ;if(scan_index & 1)    // Reverse scan
                MOV     A, @R0          ;P1 = ~LCD_buff[j];
                CPL     A
                MOV     P1, A
```

```
                MOV     A, R0                   ;P2 = ~(LCD_buff[j|4] & 0xf0);
                ADD     A, #4
                MOV     R0, A
                MOV     A, @R0
                ANL     A, #0f0H
                CPL     A
                MOV     P2, A
                SJMP    L_LCD_Scan3

L_LCD_Scan2:                                    ; Normal phase scan
                MOV     A, @R0                  ;P1 = LCD_buff[j];
                MOV     P1, A
                MOV     A, R0                   ;P2 = (LCD_buff[j|4] & 0xf0);
                ADD     A, #4
                MOV     R0, A
                MOV     A, @R0
                ANL     A, #0f0H
                MOV     P2, A

L_LCD_Scan3:
                MOV     DPTR, #T_COM            ;A COM is set as push-pull output mode
                MOV     A, R7
                MOVC    A, @A+DPTR
                ORL     P2M0, A
                CPL     A
                ANL     P2M1, A

                INC     scan_index              ;if(++scan_index == 8)      scan_index = 0;
                MOV     A, scan_index
                CJNE    A, #8, L_QuitLcdScan
                MOV     scan_index, #0

L_QuitLcdScan:
                RET

;******************** Standard font**************************
T_Display:
;               0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
DB              03FH,006H,05BH,04FH,066H,06DH,07DH,007H,07FH,06FH,077H,07CH,039H,05EH,079H,071H
;               black  -
DB              000H,040H

;******************** Load display functions for numbers 1 to 6, and the algorithm is simple ****************************
F_LCD_load:                                     ;R6 is the number, which is 1~6, R7 is the number to be
displayed
                MOV     DPTR, #T_Display        ;i = t_display[dat];
                MOV     A, R7
                MOVC    A, @A+DPTR
                MOV     B, A                    ;the number to be displayed

                MOV     A, R6
                CJNE    A, #1, L_NotLoadChar1
                MOV     R0,                     #LCD_buff
                MOV     A, @R0
                MOV     C, B.3                  ;D
                MOV     ACC.6, C
                MOV     @R0, A

                INC     R0
```

```
        MOV     A, @R0
        MOV     C, B.2          ;C
        MOV     ACC.6, C
        MOV     C, B.4          ;E
        MOV     ACC.7, C
        MOV     @R0, A

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

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

L_NotLoadChar1:
        CJNE    A, #2, L_NotLoadChar2
        MOV     R0,#LCD_buff
        MOV     A, @R0
        MOV     C, B.3          ;D
        MOV     ACC.4, C
        MOV     @R0, A

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

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

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

L_NotLoadChar2:
        CJNE    A, #3, L_NotLoadChar3
```

```
        MOV     R0,#LCD_buff
        MOV     A, @R0
        MOV     C, B.3          ;D
        MOV     ACC.2, C
        MOV     @R0, A

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

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

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

L_NotLoadChar3:
        CJNE    A, #4, L_NotLoadChar4
        MOV     R0,#LCD_buff
        MOV     A, @R0
        MOV     C, B.3          ;D
        MOV     ACC.0, C
        MOV     @R0, A

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

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

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

STC MCU

```
        MOV     C, B.5          ;F
        MOV     ACC.1, C
        MOV     @R0, A
        RET

L_NotLoadChar4:
        CJNE    A, #5, L_NotLoadChar5
        MOV     R0,#LCD_buff+4
        MOV     A, @R0
        MOV     C, B.3          ;D
        MOV     ACC.6, C
        MOV     @R0, A

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

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

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

L_NotLoadChar5:
        CJNE    A, #6, L_NotLoadChar6
        MOV     R0,#LCD_buff+4
        MOV     A, @R0
        MOV     C, B.3          ;D
        MOV     ACC.4, C
        MOV     @R0, A

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

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

```
                MOV     ACC.5, C
                MOV     @R0, A

                INC     R0
                MOV     A, @R0
                MOV     C, B.0          ;A
                MOV     ACC.4, C
                MOV     C, B.5          ;F
                MOV     ACC.5, C
                MOV     @R0, A
                RET
L_NotLoadChar6:
                RET

                END
```

C language code

```
/*****************Function description*****************
STC15 series of microcontrollers are used to test segment LCD driven by I/O directly (6 8-word LCDs, 1/4 Dutys, 1/3 bias).
Time (hours, minutes and seconds) is displayed after power-on.
P3.2 is connected to ground via a switch to enter sleep or wake up.
*********************************************/

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

typedef     unsigned char       u8;
typedef     unsigned int        u16;
typedef     unsigned long       u32;

sfr AUXR = 0x8e;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;

/*********************Local constant declaration*****************************/
#define MAIN_Fosc           11059200L           //Define the main clock

#define DIS_BLACK           0x10
#define DIS_                0x11
#define DIS_A               0x0A
#define DIS_B               0x0B
#define DIS_C               0x0C
#define DIS_D               0x0D
#define DIS_E               0x0E
#define DIS_F               0x0F

#define LCD_SET_DP2         LCD_buff[0] |=  0x08
#define LCD_CLR_DP2         LCD_buff[0] &= ~0x08
#define LCD_FLASH_DP2       LCD_buff[0] ^=  0x08

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

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

STC MCU

```
#define LCD_FLASH_2M        LCD_buff[0] ^= 0x20

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

#define LCD_SET_DP5         LCD_buff[4] |= 0x20
#define LCD_CLR_DP5         LCD_buff[4] &= ~0x20
#define LCD_FLASH_DP5       LCD_buff[4] ^= 0x20

#define P1n_standard(bitn)      P1M1 &= ~(bitn), P1M0 &= ~(bitn)
#define P1n_push_pull(bitn)     P1M1 &= ~(bitn), P1M0 |= (bitn)
#define P1n_pure_input(bitn)    P1M1 |= (bitn), P1M0 &= ~(bitn)
#define P1n_open_drain(bitn)    P1M1 |= (bitn), P1M0 |= (bitn)

#define P2n_standard(bitn)      P2M1 &= ~(bitn), P2M0 &= ~(bitn)
#define P2n_push_pull(bitn)     P2M1 &= ~(bitn), P2M0 |= (bitn)
#define P2n_pure_input(bitn)    P2M1 |= (bitn), P2M0 &= ~(bitn)
#define P2n_open_drain(bitn)    P2M1 |= (bitn), P2M0 |= (bitn)

/*********************Local variable declaration*****************************/
u8 cnt_500ms;
u8 second,minute,hour;
bit B_Second;
bit B_2ms;
u8 LCD_buff[8];
u8 scan_index;

/*********************Local function declaration*****************************/
void LCD_load(u8 n,u8 dat);
void LCD_scan(void);
void LoadRTC(void);
void delay_ms(u8 ms);

/*********************main function*****************************/
void main(void)
{
    u8    i;

    AUXR = 0x80;
    TMOD = 0x00;
    TL0 = (65536 - (MAIN_Fosc / 500));
    TH0 = (65536 - (MAIN_Fosc / 500)) >> 8;
    TR0 = 1;
    ET0 = 1;
    EA = 1;

                                                //Initialize LCD memory
    for(i=0; i<8; i++) LCD_buff[i] = 0;
    P2n_push_pull(0xf0);
    P1n_push_pull(0xff);                        //segment is set as push-pull output mode

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

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

```
_nop_();
_nop_();
_nop_();

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

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

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

    if(!INT0)                               //key is pressed, ready to sleep
    {
        LCD_CLR_2M;                         //Display hour-minute interval:
        LCD_CLR_4M;                         //Display minute-second interval:
        LCD_load(1,DIS_BLACK);
        LCD_load(2,DIS_BLACK);
        LCD_load(3,0);
        LCD_load(4,0x0F);
        LCD_load(5,0x0F);
        LCD_load(6,DIS_BLACK);

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

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

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

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

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

/*********************delay function**************************/
void delay_ms(u8 ms)
{
    unsigned int i;
    do{
        i = MAIN_Fosc / 13000;
        while(--i);                                     //14T per loop
    }while(--ms);
}

/********************** Timer0 interrupt function************************/
void timer0_int (void) interrupt 1
{
    LCD_scan();
    B_2ms = 1;
}

/********************** INT0 interrupt function *************************/
void INT0_int (void) interrupt 0
{
    EX0 = 0;
    IE0  = 0;
}

/******************* LCD Segment code scan function ***************************/
void  LCD_scan(void)                                    //5us @22.1184MHZ
{
    u8 code T_COM[4]={0x08,0x04,0x02,0x01};
    u8 j;

    j = scan_index >> 1;
    P2n_pure_input(0x0f);     //All COM outputs are high impedance, and COM's voltage is the midpoint
    if(scan_index & 1)                                  // Reverse scan
    {
        P1 = ~LCD_buff[j];
        P2 = ~(LCD_buff[j|4] & 0xf0);
    }
    else                                                // Normal phase scan
    {
        P1 = LCD_buff[j];
        P2 = LCD_buff[j|4] & 0xf0;
    }
    P2n_push_pull(T_COM[j]);                            //A COM is set as push-pull output mode
    if(++scan_index >= 8) scan_index = 0;
}
```

STC MCU

```
/****************** Load display functions for numbers 1 to 6 ***************************/
void  LCD_load(u8 n, u8 dat)                    // n is the number，dat is the number to be displayed
{
    u8 code t_display[]={                       // Standard font
        // 0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
        0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71,
        //black  -
        0x00,0x40
    };
    u8 code T_LCD_mask[4]  = {~0xc0,~0x30,~0x0c,~0x03};
    u8 code T_LCD_mask4[4] = {~0x40,~0x10,~0x04,~0x01};
    u8 i,k;
    u8 *p;

    if((n == 0) || (n > 6)) return;
    i = t_display[dat];

    if(n <= 4)                                  //1~4
    {
        n--;
        p = LCD_buff;
    }
    else
    {
        n = n - 5;
        p = &LCD_buff[4];
    }

    k = 0;
    if(i & 0x08) k |= 0x40;                     //D
    *p = (*p & T_LCD_mask4[n]) | (k>>2*n);
    p++;

    k = 0;
    if(i & 0x04) k |= 0x40;                     //C
    if(i & 0x10) k |= 0x80;                     //E
    *p = (*p & T_LCD_mask[n]) | (k>>2*n);
    p++;

    k = 0;
    if(i & 0x02) k |= 0x40;                     //B
    if(i & 0x40) k |= 0x80;                     //G
    *p = (*p & T_LCD_mask[n]) | (k>>2*n);
    p++;

    k = 0;
    if(i & 0x01) k |= 0x40;                     //A
    if(i & 0x20) k |= 0x80;                     //F
    *p = (*p & T_LCD_mask[n]) | (k>>2*n);
}

/*********************Display time ***************************/
void LoadRTC(void)
{
    LCD_load(1,hour/10);
    LCD_load(2,hour%10);
    LCD_load(3,minute/10);
    LCD_load(4,minute%10);
    LCD_load(5,second/10);
```

STC MCU

```
    LCD_load(6,second%10);
}
```

STC MCU

10 Instruction Set

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

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

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

[1]: When accessing external extended RAM, the instruction execution cycle is related to the SPEED [2: 0] bits in the BUS_SPEED register.

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

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

STC MCU

11 Interrupt System

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

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

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

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

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

11.1 Interrupt sources of STC8G series

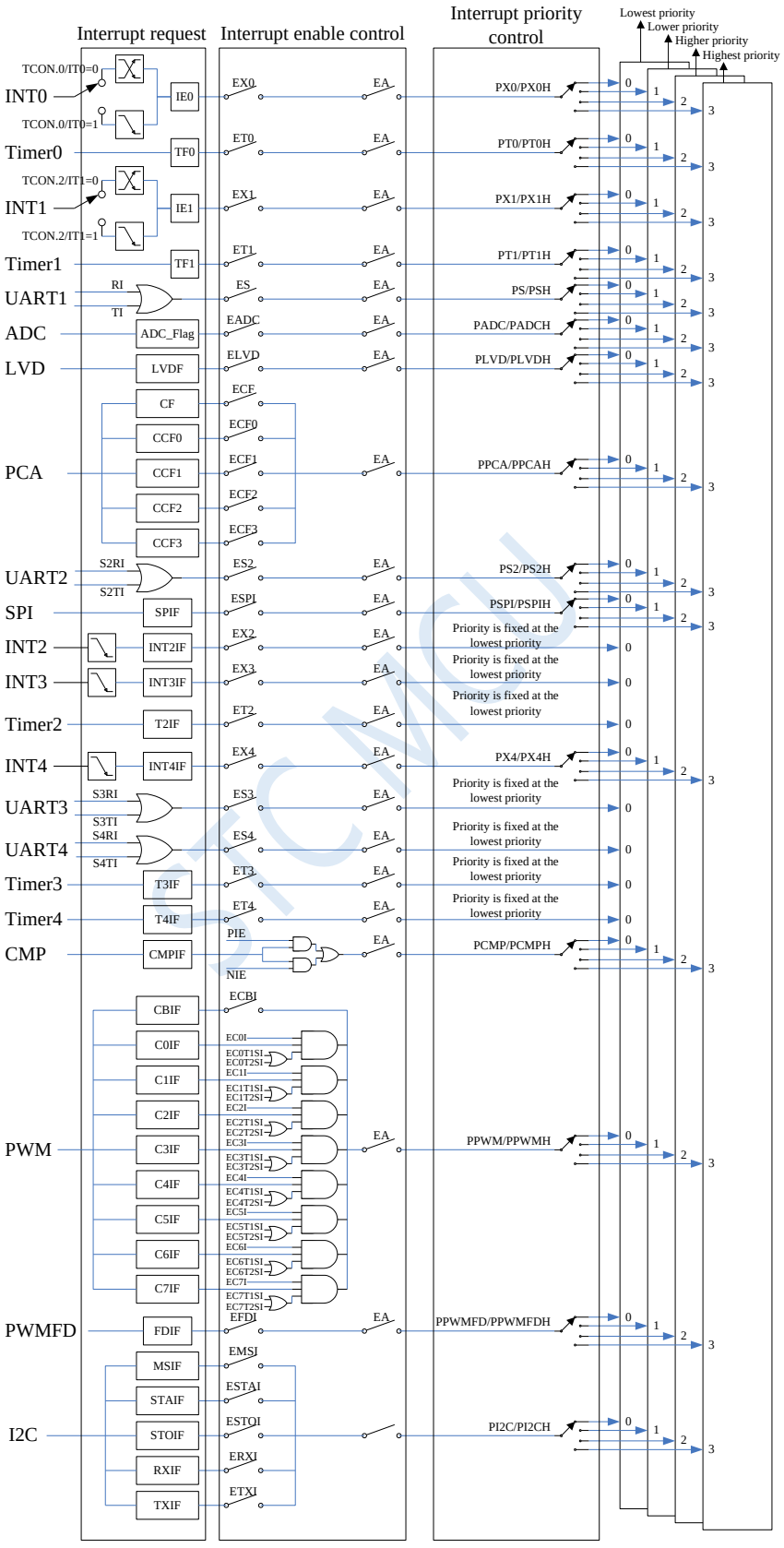
The √ in the following table indicates that the corresponding series have the corresponding interrupt source.

Interrupt sources	STC8G1K08 family	STC8G1K08 8PIN family	STC8G1K08A family	STC8G2K64S4 family	STC15H2K64S4 family	STC8G2K64S2 family	STC8G1K08T family
External interrupt 0 (INT0)	√	√	√	√	√	√	√
Timer 0 interrrupt (Timer0)	√	√	√	√	√	√	√
External interrupt 1 (INT1)	√	√	√	√	√	√	√
Timer 1 interrrupt (Timer1)	√	√	√	√	√	√	√
UART1 interrupt (UART1)	√	√	√	√	√	√	√
ADC interrupt (ADC)	√		√	√	√	√	√
Low voltage detection interrupt (LVD)	√	√	√	√	√	√	√

Capture interrupt (CCP/PCA/PWM)	√		√	√	√	√	√
UART2 interrupt (UART2)	√			√	√	√	
SPI interrupt (SPI)	√	√	√	√	√	√	√
External interrupt 2 (INT2)	√	√	√	√	√	√	√
External interrupt 3 (INT3)	√	√	√	√	√	√	√
Timer 2 interrrupt (Timer2)	√			√	√	√	√
External interrupt 4 (INT4)	√	√	√	√	√	√	√
UART3 interrupt (UART3)				√	√		
UART4 interrupt (UART4)				√	√		
Timer 3 interrrupt (Timer3)				√	√	√	
Timer 4 interrrupt (Timer4)				√	√	√	
Comparator interrupt (CMP)	√			√	√	√	√
Enhenced PWM0 interrupt				√	√	√	
PWM0 fault detection interrupt (PWM0FD)				√	√	√	
I2C interrupt	√	√	√	√	√	√	√
Touch Key interrupt							√
Enhenced PWM1 interrupt				√	√	~~√~~	
Enhenced PWM2 interrupt				√	√	√	
Enhenced PWM3 interrupt				√	√	~~√~~	
Enhenced PWM4 interrupt				√	√	~~√~~	
Enhenced PWM5 interrupt				√	√	~~√~~	
PWM2 fault detection interrupt (PWM2FD)				√	√	√	
PWM4 fault detection interrupt (PWM4FD)				√	√	~~√~~	

STC MCU

11.2 Structure of STC8G Interrupt



11.3 Interrupt List of STC8G Series of Microcontroller

Interrupt source	Interrupt vector	Order	Priority level setup bit	Priority level	Interrupt request flag	Interrupt enable bit
INT0	0003H	0	PX0PX0H	0/1/2/3	IE0	EX0
Timer0	000BH	1	PT0,PT0H	0/1/2/3	TF0	ET0
INT1	0013H	2	PX1,PX1H	0/1/2/3	IE1	EX1
Timer1	001BH	3	PT1,PT1H	0/1/2/3	TF1	ET1
UART1	0023H	4	PS,PSH	0/1/2/3	RI || TI	ES
ADC	002BH	5	PADC,PADCH	0/1/2/3	ADC_FLAG	EADC
LVD	0033H	6	PLVD,PLVDH	0/1/2/3	LVDF	ELVD
PCA	003BH	7	PPCA,PPCAH	0/1/2/3	CF	ECF
					CCF0	ECCF0
					CCF1	ECCF1
					CCF2	ECCF2
					CCF3	ECCF3
UART2	0043H	8	PS2,PS2H	0/1/2/3	S2RI || S2TI	ES2
SPI	004BH	9	PSPI,PSPIH	0/1/2/3	SPIF	ESPI
INT2	0053H	10		0	INT2IF	EX2
INT3	005BH	11		0	INT3IF	EX3
Timer2	0063H	12		0	T2IF	ET2
INT4	0083H	16	PX4,PX4H	0/1/2/3	INT4IF	EX4
UART3	008BH	17	PS3,PS3H	0/1/2/3	S3RI || S3TI	ES3
UART4	0093H	18	PS4,PS4H	0/1/2/3	S4RI || S4TI	ES4
Timer3	009BH	19		0	T3IF	ET3
Timer4	00A3H	20		0	T4IF	ET4
CMP	00ABH	21	PCMP,PCMPH	0/1/2/3	CMPIF	PIE|NIE

STC MCU

Interrupt source	Interrupt vector	Order	Priority level setup bit	Priority level	Interrupt request flag	Interrupt enable bit
PWM0	00B3H	22	PPWM0,PPWM0H	0/1/2/3	CBIF	ECBI
					CnIF	ECnI && ECnT1SI
						ECnI && ECnT2SI
PWM0FD	00BBH	23	PPWM0FD,PPWM0FDH	0/1/2/3	FDIF	EFDI
I2C	00C3H	24	PI2C,PI2CH	0/1/2/3	MSIF	EMSI
					STAIF	ESTAI
					RXIF	ERXI
					TXIF	ETXI
					STOIF	ESTOI
PWM1	00E3H	28	PPWM1,PPWM1H	0/1/2/3	CBIF	ECBI
					CnIF	ECnI && ECnT1SI
						ECnI && ECnT2SI
PWM2	00EBH	29	PPWM2,PPWM2H	0/1/2/3	CBIF	ECBI
					CnIF	ECnI && ECnT1SI
						ECnI && ECnT2SI
PWM3	00F3H	30	PPWM3,PPWM3H	0/1/2/3	CBIF	ECBI
					CnIF	ECnI && ECnT1SI
						ECnI && ECnT2SI
PWM4	00FBH	31	PPWM4,PPWM4H	0/1/2/3	CBIF	ECBI
					CnIF	ECnI && ECnT1SI
						ECnI && ECnT2SI
PWM5	0103H	32	PPWM5,PPWM5H	0/1/2/3	CBIF	ECBI
					CnIF	ECnI && ECnT1SI
						ECnI && ECnT2SI
PWM2FD	010BH	33	PPWM2FD,PPWM2FDH	0/1/2/3	FDIF	EFDI
PWM4FD	0113H	34	PPWM4FD,PPWM4FDH	0/1/2/3	FDIF	EFDI
TKSU	011BH	35	PTKSU,PTKSUH	0/1/2/3	TKIF	ETKSUI

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

```
void    INT0_Routine(void)      interrupt  0;
void    TM0_Rountine(void)      interrupt  1;
void    INT1_Routine(void)      interrupt  2;
void    TM1_Rountine(void)      interrupt  3;
void    UART1_Routine(void)     interrupt  4;
void    ADC_Routine(void)       interrupt  5;
void    LVD_Routine(void)       interrupt  6;
void    PCA_Routine(void)       interrupt  7;
void    UART2_Routine(void)     interrupt  8;
void    SPI_Routine(void)       interrupt  9;
void    INT2_Routine(void)      interrupt  10;
void    INT3_Routine(void)      interrupt  11;
void    TM2_Routine(void)       interrupt  12;
```

```
void    INT4_Routine(void)      interrupt 16;
void    UART3_Routine(void)     interrupt 17;
void    UART4_Routine(void)     interrupt 18;
void    TM3_Routine(void)       interrupt 19;
void    TM4_Routine(void)       interrupt 20;
void    CMP_Routine(void)       interrupt 21;
void    PWM0_Routine(void)      interrupt 22;
void    PWM0FD_Routine(void)    interrupt 23;
void    I2C_Routine(void)       interrupt 24;
void    PWM1_Routine(void)      interrupt 28;
void    PWM2_Routine(void)      interrupt 29;
void    PWM3_Routine(void)      interrupt 30;
void    PWM4_Routine(void)      interrupt 31;

//void  PWM5_Routine(void)      interrupt 32;
//void  PWM2FD_Routine(void)    interrupt 33;
//void  PWM4FD_Routine(void)    interrupt 34;
//void  TKSU_Routine(void)      interrupt 35;
```

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

STC MCU

11.4 Registers Related to Interrupt

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
IE	Interrupt enable register	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000,0000
IE2	Interrupt enable register2	AFH	ETKSUI	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000,0000
INTCLKO	External interrupt and clock output control register	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
IP	Interrupt Priority Low Byte	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000,0000
IPH	Interrupt Priority High Byte	B7H	PPCAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	0000,0000
IP2	2nd Interrupt Priority register low byte	B5H	PPWM2FD PTKSU	PI2C	PCMP	PX4	PPWM0FD	PPWM0	PSPI	PS2	0000,0000
IP2H	2nd Interrupt Priority register high byte	B6H	PPWM2FDH PTKSUH	PI2CH	PCMPH	PX4H	PPWM0FDH	PPWM0H	PSPIH	PS2H	0000,0000
IP3	3nd Interrupt Priority register low byte	DFH	PPWM4FD	PPWM5	PPWM4	PPWM3	PPWM2	PPWM1	PS4	PS3	0000,0000
IP3H	3nd Interrupt Priority Register High Byte	EEH	PPWM4FDH	PPWM5H	PPWM4H	PPWM3H	PPWM2H	PPWM1H	PS4H	PS3H	0000,0000
TCON	Timer 0 and 1 control register	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
AUXINTIF	Extended External Interrupt Flag Register	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF	x000,x000
SCON	UART1 control register	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
S2CON	UART2 control register	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0000,0000
S3CON	UART3 control register	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S4CON	Serial port 4 control register	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
PCON	Power control register	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
ADC_CONTR	ADC control register	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]				000x,0000
SPSTAT	SPI Status register	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
CCON	PCA Control Register	D8H	CF	CR	-	-	-	CCF2	CCF1	CCF0	00xx,x000
CMOD	PCA Mode Register	D9H	CIDL	-	-	-	CPS[2:0]			ECF	0xxx,0000
CCAPM0	PCA 0 Mode Control Register	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAPM1	PCA 1 Mode Control Register	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CCAPM2	PCA 2 Mode Control Register	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
CMPCR1	Comparator Control Register 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
PWMCFG01	PWM Configuration Register	F6H	PWM1CBIF	EPWM1CBI	FLTPS0	PWM1CEN	PWM0CBIF	EPWM0CBI	ENPWM0TA	PWM0CEN	0000,0000
PWMCFG23	PWM Configuration Register	F7H	PWM3CBIF	EPWM3CBI	FLTPS1	PWM3CEN	PWM2CBIF	EPWM2CBI	ENPWM2TA	PWM2CEN	0000,0000
PWMCFG45	PWM Configuration Register	FEH	PWM5CBIF	EPWM5CBI	FLTPS2	PWM5CEN	PWM4CBIF	EPWM4CBI	ENPWM4TA	PWM4CEN	0000,0000

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
I2CMSCR	I²C Master Control Register	FE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I²C Master Status Register	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I²C Slave Control Register	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I²C Slave Status Register	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
PWM0IF	PWM0 Interrupt Flag Register	FF05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM0FDCR	PWM0 Fault Detection Control Register	FF06H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000
PWM00CR	PWM00 Control Register	FF14H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM01CR	PWM01 Control Register	FF1CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM02CR	PWM02 Control Register	FF24H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM03CR	PWM03 Control Register	FF2CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM04CR	PWM04 Control Register	FF34H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM05CR	PWM05 Control Register	FF3CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM06CR	PWM06 Control Register	FF44H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM07CR	PWM07 Control Register	FF4CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM1IF	PWM1 Interrupt Flag Register	FF55H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM1FDCR	PWM1 Fault Detection Control Register	FF56H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF	0000,0000
PWM10CR	PWM10 Control Register	FF64H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM11CR	PWM11 Control Register	FF6CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM12CR	PWM12 Control Register	FF74H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM13CR	PWM13 Control Register	FF7CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM14CR	PWM14 Control Register	FF84H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM15CR	PWM15 Control Register	FF8CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM16CR	PWM16 Control Register	FF94H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM17CR	PWM17 Control Register	FF9CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM2IF	PWM2 Interrupt Flag Register	FFA5H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM2FDCR	PWM2 Fault Detection Control Register	FFA6H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000
PWM20CR	PWM20 Control Register	FFB4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM21CR	PWM21 Control Register	FFBCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000

PWM22CR	PWM22 Control Register	FFC4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM23CR	PWM23 Control Register	FFCCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM24CR	PWM24 Control Register	FFD4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM25CR	PWM25 Control Register	FFDCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM26CR	PWM26 Control Register	FFE4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM27CR	PWM27 Control Register	FFECH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM3IF	PWM3 Interrupt Flag Register	FC05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM3FDCR	PWM3 Fault Detection Control Register	FC06H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF	0000,0000
PWM30CR	PWM30 Control Register	FC14H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM31CR	PWM31 Control Register	FC1CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM32CR	PWM32 Control Register	FC24H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM33CR	PWM33 Control Register	FC2CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM34CR	PWM34 Control Register	FC34H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM35CR	PWM35 Control Register	FC3CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM36CR	PWM36 Control Register	FC44H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM37CR	PWM37 Control Register	FC4CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM4IF	PWM4 Interrupt Flag Register	FC55H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM4FDCR	PWM4 Fault Detection Control Register	FC56H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000
PWM40CR	PWM40 Control Register	FC64H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM41CR	PWM41 Control Register	FC6CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM42CR	PWM42 Control Register	FC74H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM43CR	PWM43 Control Register	FC7CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM44CR	PWM44 Control Register	FC84H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM45CR	PWM45 Control Register	FC8CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM46CR	PWM46 Control Register	FC94H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM47CR	PWM47 Control Register	FC9CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM5IF	PWM5 Interrupt Flag Register	FCA5H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM5FDCR	PWM5 Fault Detection Control Register	FCA6H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF	0000,0000
PWM50CR	PWM50 Control Register	FCB4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM51CR	PWM51 Control Register	FCBCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM52CR	PWM52 Control Register	FCC4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM53CR	PWM53 Control Register	FCCCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM54CR	PWM54 Control Register	FCD4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM55CR	PWM55 Control Register	FCDCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM56CR	PWM56 Control Register	FCE4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM57CR	PWM57 Control Register	FCECH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
TSSTA2	Touch Key Status Register 2	FB47H	TSIF	TSDOV	-	-	TSDNCHN[3:0]				00xx,0000

11.4.1 Interrupt Enable Control Registers (Interrupt Enable bits)

IE (Interrupt Enable Rsgister)

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

EA: The general or global interrupt enable control bit. The function of EA is to allow interrupts to be multi-level controlled. That is, every interrupt source is controlled by EA firstly and then by its own interrupt enable control bit.

0: All interrupts are masked.

1: Enable the CPU interrupt, every interrupt source would be individually enabled or disabled by setting or clearing its enable bit.

ELVD: Low volatge detection interrupt enable bit.

0: disable low voltage detection interrupt.

1: enable Low voltage detection interrupt.

EADC: ADC interrupt enable bit.

0: disable ADC interrupt.

1: enable ADC interrupt.

ES: UART1 interrupt enable bit.

0: disable UART1 interrupt.

1: enable UART1 interrupt.

ET1: Timer 1 interrupt enable bit.

0: disable Timer 1 interrupt.

1: enable Timer 1 interrupt.

EX1: External interrupt 1 enable bit.

0: disable external interrupt 1.

1: enable external interrupt 1.

ET0: Timer 0 interrupt enable bit.

0: disable Timer 0 interrupt.

1: enable Timer 0 interrupt.

EX0: External interrupt 0 enable bit.

0: disable external interrupt 0.

1: enable external interrupt 0.

IE2 (Interrupt Enable Rsgister 2)

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

ETKSUI: Touch key interrupt enable bit.

0: disable touch key interrupt.

1: enable touch key interrupt.

ET4: Timer 4 interrupt enable bit.

0: disable Timer 4 interrupt.

1: enable Timer 4 interrupt.

ET3: Timer 3 interrupt enable bit.

0: disable Timer 3 interrupt.

1: enable Timer 3 interrupt.

ES4: UART4 interrupt enable bit.

0: disable UART4 interrupt.

1: enable UART4 interrupt.

ES3: UART3 interrupt enable bit.

0: disable UART3 interrupt.

1: enable UART3 interrupt.

ET2: Timer 2 interrupt enable bit.

0: disable Timer 2 interrupt.

1: enable Timer 2 interrupt.

ESPI: SPI interrupt enalbe bit.

0: disable SPI interrupt.

1: enable SPI interrupt.

ES2: UART2 interrupt enable bit.

0: disable UART2 interrupt.

1: enable UART2 interrupt.

INTCLKO (External interrupt and clock output control register)

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

EX4: External Interrupt 4 enabel bit.

0: disable External Interrupt 4.

1: enable External Interrupt 4.

EX3: External Interrupt 3 enabel bit.

STC MCU

0: disable External Interrupt 3.

1: enable External Interrupt 3.

EX2: External Interrupt 2 enabel bit.

0: disable External Interrupt 2.

1: enable External Interrupt 2.

PCA/CCP/PWM Interrupt Control Registers

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

ECF: PCA counter interrupt enable bit.

0: disable PCA counter interrupt.

1: enable PCA counter interrupt.

ECCF0: PCA 0 interrupt enable bit.

0: disable PCA 0 interrupt.

1: enable PCA 0 interrupt.

ECCF1: PCA 1 interrupt enable bit.

0: disable PCA 1 interrupt.

1: enable PCA 1 interrupt.

ECCF2: PCA 2 interrupt enable bit.

0: disable PCA 2 interrupt.

1: enable PCA 2 interrupt.

CMPCR1（Comparator Control Register 1）

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

PIE: Comparator rising-edge interrupt enable bit.

0: disable comparator rising-edge interrupt.

1: enable comparator rising-edge interrupt.

NIE: Comparator falling-edge interrupt enable bit.

0: disable comparator falling-edge interrupt.

1: enable comparator falling-edge interrupt.

I2C Control Register

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

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

0: disable I^2C master mode interrupt.

1: enable I^2C master mode interrupt.

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

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

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

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

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

1: enable I²C slave completes receiving data event interrupt.

ETXI: I²C slave completes transmitting data event interrupt enable bit.

0: disable I²C slave completes transmitting data event interrupt.

1: enable I²C slave completes transmitting data event interrupt.

ESTOI: I²C slave receives STOP event interrupt enable bit.

0: disable I²C slave receives STOP event interrupt.

1: enable I²C slave receives STOP event interrupt.

Enhanced PWM Configuration Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWMCFG01	F6H	PWM1CBIF	EPWM1CBI	FLTPS0	PWM1CEN	PWM0CBIF	EPWM0CBI	ENPWM0TA	PWM0CEN
PWMCFG23	F7H	PWM3CBIF	EPWM3CBI	FLTPS1	PWM3CEN	PWM2CBIF	EPWM2CBI	ENPWM2TA	PWM2CEN
PWMCFG45	FEH	PWM5CBIF	EPWM5CBI	FLTPS2	PWM5CEN	PWM4CBIF	EPWM4CBI	ENPWM4TA	PWM4CEN

EPWM0CBI: Enhanced PWM0 counter interrupt enable bit.

0: disable PWM0 counter interrupt enable bit

1: enable PWM0 counter interrupt enable bit

EPWM1CBI: Enhanced PWM1 counter interrupt enable bit.

0: disable PWM1 counter interrupt enable bit

1: enable PWM1 counter interrupt enable bit

EPWM2CBI: Enhanced PWM2 counter interrupt enable bit.

0: disable PWM2 counter interrupt enable bit

1: enable PWM2 counter interrupt enable bit

EPWM3CBI: Enhanced PWM3 counter interrupt enable bit.

0: disable PWM3 counter interrupt enable bit

1: enable PWM3 counter interrupt enable bit

EPWM4CBI: Enhanced PWM4 counter interrupt enable bit.

0: disable PWM4 counter interrupt enable bit

1: enable PWM4 counter interrupt enable bit

EPWM5CBI: Enhanced PWM5 counter interrupt enable bit.

0: disable PWM5 counter interrupt enable bit

1: enable PWM5 counter interrupt enable bit

Enhanced PWM Fault Detection Control Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWM0FDCR	FF06H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF
PWM1FDCR	FF56H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF
PWM2FDCR	FFA6H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF
PWM3FDCR	FC06H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF
PWM4FDCR	FC56H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF
PWM5FDCR	FCA6H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF

EFDI: PWM external fault event interrupt enable bit.

0: disable PWM external fault event interrupt.

1: enable PWM external fault event interrupt.

Enhanced PWM Control Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0

PWM00CR	FF14H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM01CR	FF1CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM02CR	FF24H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM03CR	FF2CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM04CR	FF34H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM05CR	FF3CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM06CR	FF44H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM07CR	FF4CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM10CR	FF64H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM11CR	FF6CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM12CR	FF74H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM13CR	FF7CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM14CR	FF84H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM15CR	FF8CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM16CR	FF94H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM17CR	FF9CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM20CR	FFB4H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM21CR	FFBCH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM22CR	FFC4H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM23CR	FFCCH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM24CR	FFD4H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM25CR	FFDCH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM26CR	FFE4H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM27CR	FFECH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM30CR	FC14H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM31CR	FC1CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM32CR	FC24H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM33CR	FC2CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM34CR	FC34H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM35CR	FC3CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM36CR	FC44H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM37CR	FC4CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM40CR	FC64H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM41CR	FC6CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM42CR	FC74H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM43CR	FC7CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM44CR	FC84H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM45CR	FC8CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM46CR	FC94H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM47CR	FC9CH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM50CR	FCB4H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM51CR	FCBCH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM52CR	FCC4H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM53CR	FCCCH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM54CR	FCD4H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM55CR	FCDCH	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM56CR	FCE4H	ENO	INI	-	-		ENI	ENT2I	ENT1I
PWM57CR	FCECH	ENO	INI	-	-		ENI	ENT2I	ENT1I

ECI: PWM interrupt enable bit.

0: disable PWMn interrupt.

1: enable PWMn interrupt.

ET2SI: PWM second trigger point interrupt enable bit.

0: disable PWM the second trigger point interrupt.

1: enable PWM the second trigger point interrupt.

ET1SI: PWM first trigger point interrupt enable bit.

0: disable PWM the first trigger point interrupt.

1: enable PWM the first trigger point interrupt.

11.4.2 Interrupt Request Registers (Interrupt flags)

Timer 0 and 1 Control Register

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

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

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

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

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

Auxiliary Interrupt Flag Register

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

INT4IF: external interrupt 4 request flag, which must be cleared by software.

INT3IF: external interrupt 3 request flag, which must be cleared by software.

INT2IF: external interrupt 2 request flag, which must be cleared by software.

T4IF: timer 4 overflow interrupt flag, which must be cleared by software.

T3IF: timer 3 overflow interrupt flag, which must be cleared by software.

T2IF: timer 2 overflow interrupt flag, which must be cleared by software.

Notice:

STC15 series added 16-bit reload timer in the early adopting 0.35um process 1T 8051, which is the important change in the 8051 world. Due to the high producing cost, the interrupt request flag registers of 16-bit reload timer 2/3/4 are not designed for users to access. There are only internally hidden flag bits. The method provided to the user software to clear the internal hidden flag bits is: when the user software disables the timer 2/3/4 interrupt, the hardware automatically clears the timer 2/3/4 internal hidden the interrupt request flag bit.

For product consistency, the STC8A/STC8F and subsequent STC8G/STC8H/STC8C/STC12H series who use 0.18um technology have added interrupt request flag registers that can be accessed by users of timer 2/3/4, but the function of the hidden interrupt request flags being cleared by hardware automatically in timer 2/3/4 when their interrupts are prohibited is still reserved. So do not arbitrarily disable the timer 2/3/4 interrupt when the timer 2/3/4 does not stop counting, otherwise, the hidden interrupt request flags that actually work will be cleared. It is possible that after the counter overflows, and the hidden interrupt request flag is set to 1, it is

cleared by the user while waiting to request an interrupt.

UARTs Control Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI
S2CON	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI
S3CON	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI
S4CON	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

TI: Transmit interrupt flag of UART1, which must be cleared by software.

RI: Receive interrupt flag of UART1, which must be cleared by software.

S2TI: Transmit interrupt flag of UART2, which must be cleared by software.

S2RI: Receive interrupt flag of UART2, which must be cleared by software.

S3TI: Transmit interrupt flag of UART3, which must be cleared by software.

S3RI: Receive interrupt flag of UART3, which must be cleared by software.

S4TI: Transmit interrupt flag of UART4, which must be cleared by software.

S4RI: Receive interrupt flag of UART4, which must be cleared by software.

Power Control Register

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

LVDF: Low voltage detection interrupt flag, which must be cleared by software.

ADC Control Register

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

ADC_FLAG: ADC completes conversion interrupt request flag, which must be cleared by software.

SPI Status Register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	CDH	SPIF	WCOL	-	-	-	-	-	-

SPIF: SPI transmission completion interrupt request flag, which must be cleared by software.

PCA Control Register

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

CF: PCA counter interrupt request flag, which must be cleared by software.

CCF3: PCA3 interrupt request flag, which must be cleared by software.

CCF2: PCA2 interrupt request flag, which must be cleared by software.

CCF1: PCA1 interrupt request flag, which must be cleared by software.

CCF0: PCA0 interrupt request flag, which must be cleared by software.

Comparator Control Register 1

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

CMPIF: Comparator interrupt request flag, which must be cleared by software.

I2C Status Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSST	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO
I2CSLST	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO

MSIF: I²C master mode interrupt request flag, which must be cleared by software.

ESTAI: I²C slave receives the START event interrupt request flag, which must be cleared by software.

ERXI: I²C slave completes receiving data event interrupt request flag, which must be cleared by software.

ETXI: I²C slave completes transmitting data event interrupt request flag, which must be cleared by software.

ESTOI: I²C slave receives the STOP event interrupt request flag, which should be must by software.

Enhanced PWM Configuration Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWMCFG01	F6H	PWM1CBIF	EPWM1CBI	FLTPS0	PWM1CEN	PWM0CBIF	EPWM0CBI	ENPWM0TA	PWM0CEN
PWMCFG23	F7H	PWM3CBIF	EPWM3CBI	FLTPS1	PWM3CEN	PWM2CBIF	EPWM2CBI	ENPWM2TA	PWM2CEN
PWMCFG45	FEH	PWM5CBIF	EPWM5CBI	FLTPS2	PWM5CEN	PWM4CBIF	EPWM4CBI	ENPWM4TA	PWM4CEN

PWM0CBIF: Enhanced PWM0 counter interrupt request flag, which must be cleared by software.

PWM1CBIF: Enhanced PWM1 counter interrupt request flag, which must be cleared by software.

PWM2CBIF: Enhanced PWM2 counter interrupt request flag, which must be cleared by software.

PWM3CBIF: Enhanced PWM3 counter interrupt request flag, which must be cleared by software.

PWM4CBIF: Enhanced PWM4 counter interrupt request flag, which must be cleared by software.

PWM5CBIF: Enhanced PWM5 counter interrupt request flag, which must be cleared by software.

Enhanced PWM Interrupt Flag Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWM0IF	FF05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF
PWM1IF	FF55H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF
PWM2IF	FFA5H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF
PWM3IF	FC05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF
PWM4IF	FC55H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF
PWM5IF	FCA5H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF

C7IF: Enhanced PWM7 interrupt request flag, which must be cleared by software.

C6IF: Enhanced PWM6 interrupt request flag, which must be cleared by software.

C5IF: Enhanced PWM5 interrupt request flag, which must be cleared by software.

C4IF: Enhanced PWM4 interrupt request flag, which must be cleared by software.

C3IF: Enhanced PWM3 interrupt request flag, which must be cleared by software.

C2IF: Enhanced PWM2 interrupt request flag, which must be cleared by software.

C1IF: Enhanced PWM 1 interrupt request flag, which must be cleared by software.

C0IF: Enhanced PWM0 interrupt request flag, which must be cleared by software.

Enhanced PWM Fault Detection Control Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWM0FDCR	FF06H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF
PWM1FDCR	FF56H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF
PWM2FDCR	FFA6H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF
PWM3FDCR	FC06H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF
PWM4FDCR	FC56H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF
PWM5FDCR	FCA6H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF

FDIF: PWM fault detection interrupt request flag, which must be cleared by software.

Touch Key Status Register 2

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TSSTA2	FB47H	TSIF	TSDOV	-	-	TSDNCHN[3:0]			

TSIF: Touch key interrupt flag, which must be cleared by writing 1.

11.4.3 Interrupt Priority Control Registers

All other interrupts except INT2, INT3, Timer 2, Timer 3, and Timer 4 have 4-level interrupt priority levels that can be set.

Interrupt Priority Control Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0
IPH	B7H	PPCAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H
IP2	B5H	PPWM2FD PTKSU	PI2C	PCMP	PX4	PPWM0FD	PPWM0	PSPI	PS2
IP2H	B6H	PPWM2FDH PTKSUH	PI2CH	PCMPH	PX4H	PPWM0FDH	PPWM0H	PSPIH	PS2H
IP3	DFH	PPWM4FD	PPWM5	PPWM4	PPWM3	PPWM2	PPWM1	PS4	PS3
IP3H	EEH	PPWM4FDH	PPWM5H	PPWM4H	PPWM3H	PPWM2H	PPWM1H	PS4H	PS3H

PX0H,PX0: External interrupt 0 interrupt priority control bit.

00: INT0 interrupt priority level is 0 (lowest)
01: INT0 interrupt priority level is 1 (lower)
10: INT0 interrupt priority level is 2 (higher)
11: INT0 interrupt priority level is 3 (highest)

PT0H,PT0: Timer 0 interrupt priority control bit.

00: Timer 0 interrupt priority level is 0 (lowest)
01: Timer 0 interrupt priority level is 1 (lower)
10: Timer 0 interrupt priority level is 2 (higher)
11: Timer 0 interrupt priority level is 3 (highest)

PX1H,PX1: External interrupt 1 interrupt priority control bit.

00：INT1 interrupt priority level is 0 (lowest)
01：INT1 interrupt priority level is 1 (lower)
10：INT1 interrupt priority level is 2 (higher)
11：INT1 interrupt priority level is 3 (highest)

PT1H,PT1: Timer 1 interrupt priority control bit.

00: Timer 1 interrupt priority level is 0 (lowest)
01: Timer 1 interrupt priority level is 1 (lower)
10: Timer 1 interrupt priority level is 2 (higher)
11: Timer 1 interrupt priority level is 3 (highest)

PSH,PS: UART1 interrupt priority control bit.

00: UART1 interrupt priority level is 0 (lowest)
01: UART1 interrupt priority level is 1 (lower)
10: UART1 interrupt priority level is 2 (higher)
11: UART1 interrupt priority level is 3 (highest)

PADCH,PADC: ADC interrupt priority control bit.

00: ADC interrupt priority level is 0 (lowest)

01: ADC interrupt priority level is 1 (lower)

10: ADC interrupt priority level is 2 (higher)

11: ADC interrupt priority level is 3 (highest)

PLVDH,PLVD: Low voltage detection interrupt priority control bit.

00: LVD interrupt priority level is 0 (lowest)

01: LVD interrupt priority level is 1 (lower)

10: LVD interrupt priority level is 2 (higher)

11: LVD interrupt priority level is 3 (highest)

PPCAH,PPCA: CCP/PCA interrupt priority control bit.

00: CCP/PCA interrupt priority level is 0 (lowest)

01: CCP/PCA interrupt priority level is 1 (lower)

10: CCP/PCA interrupt priority level is 2 (higher)

11: CCP/PCA interrupt priority level is 3 (highest)

PS2H,PS2: UART2 interrupt priority control bit.

00: UART2 interrupt priority level is 0 (lowest)

01: UART2 interrupt priority level is 1 (lower)

10: UART2 interrupt priority level is 2 (higher)

11: UART2 interrupt priority level is 3 (highest)

PSPIH,PSPI: SPI interrupt priority control bit.

00: SPI interrupt priority level is 0 (lowest)

01: SPI interrupt priority level is 1 (lower)

10: SPI interrupt priority level is 2 (higher)

11: SPI interrupt priority level is 3 (highest)

PX4H,PX4: External interrupt 4 interrupt priority control bit.

00: INT4 interrupt priority level is 0 (lowest)

01: INT4 interrupt priority level is 1 (lower)

10: INT4 interrupt priority level is 2 (higher)

11: INT4 interrupt priority level is 3 (highest)

PCMPH,PCMP: Comparator interrupt priority control bit.

00: CMP interrupt priority level is 0 (lowest)

01: CMP interrupt priority level is 1 (lower)

10: CMP interrupt priority level is 2 (higher)

11: CMP interrupt priority level is 3 (highest)

PI2CH,PI2C: I2C interrupt priority control bit.

00: I2C interrupt priority level is 0 (lowest)

01: I2C interrupt priority level is 1 (lower)

10: I2C interrupt priority level is 2 (higher)

11: I2C interrupt priority level is 3 (highest)

PPWM0H,PPWM0: Enhanced PWM0 interrupt priority control bit.

00: Enhanced PWM0 interrupt priority level is 0 (lowest)

01: Enhanced PWM0 interrupt priority level is 1 (lower)

10: Enhanced PWM0 interrupt priority level is 2 (higher)

STC MCU

11: Enhanced PWM0 interrupt priority level is 3 (highest)

PPWM1H,PPWM1: Enhanced PWM1 interrupt priority control bit.

00: Enhanced PWM1 interrupt priority level is 0 (lowest)
01: Enhanced PWM1 interrupt priority level is 1 (lower)
10: Enhanced PWM1 interrupt priority level is 2 (higher)
11: Enhanced PWM1 interrupt priority level is 3 (highest)

PPWM2H,PPWM2: Enhanced PWM2 interrupt priority control bit.

00: Enhanced PWM2 interrupt priority level is 0 (lowest)
01: Enhanced PWM2 interrupt priority level is 1 (lower)
10: Enhanced PWM2 interrupt priority level is 2 (higher)
11: Enhanced PWM2 interrupt priority level is 3 (highest)

PPWM3H,PPWM3: Enhanced PWM3 interrupt priority control bit.

00: Enhanced PWM3 interrupt priority level is 0 (lowest)
01: Enhanced PWM3 interrupt priority level is 1 (lower)
10: Enhanced PWM3 interrupt priority level is 2 (higher)
11: Enhanced PWM3 interrupt priority level is 3 (highest)

PPWM4H,PPWM4: Enhanced PWM4 interrupt priority control bit.

00: Enhanced PWM4 interrupt priority level is 0 (lowest)
01: Enhanced PWM4 interrupt priority level is 1 (lower)
10: Enhanced PWM4 interrupt priority level is 2 (higher)
11: Enhanced PWM4 interrupt priority level is 3 (highest)

PPWM5H,PPWM5: Enhanced PWM5 interrupt priority control bit.

00: Enhanced PWM5 interrupt priority level is 0 (lowest)
01: Enhanced PWM5 interrupt priority level is 1 (lower)
10: Enhanced PWM5 interrupt priority level is 2 (higher)
11: Enhanced PWM5 interrupt priority level is 3 (highest)

PPWM0FDH,PPWM0FD: Enhanced PWM0 fault detection interrupt priority control bit.

00: Enhanced PWM0 fault detection interrupt priority level is 0 (lowest)
01: Enhanced PWM0 fault detection interrupt priority level is 1 (lower)
10: Enhanced PWM0 fault detection interrupt priority level is 2 (higher)
11: Enhanced PWM0 fault detection interrupt priority level is 3 (highest)

PPWM2FDH,PPWM2FD: Enhanced PWM2 fault detection interrupt priority control bit.

00: Enhanced PWM2 fault detection interrupt priority level is 0 (lowest)
01: Enhanced PWM2 fault detection interrupt priority level is 1 (lower)
10: Enhanced PWM2 fault detection interrupt priority level is 2 (higher)
11: Enhanced PWM2 fault detection interrupt priority level is 3 (highest)

PPWM4FDH,PPWM4FD: Enhanced PWM4 fault detection interrupt priority control bit.

00: Enhanced PWM4 fault detection interrupt priority level is 0 (lowest)
01: Enhanced PWM4 fault detection interrupt priority level is 1 (lower)
10: Enhanced PWM4 fault detection interrupt priority level is 2 (higher)
11: Enhanced PWM4 fault detection interrupt priority level is 3 (highest)

PTKSUH,PTKSU: Touch key interrupt priority control bit.

00: Touch key interrupt priority level is 0 (loweset)
01: Touch key interrupt priority level is 1 (lower)

10: Touch key interrupt priority level is 2 (higher)

11: Touch key interrupt priority level is 3 (highest)

11.5 Example Routines

11.5.1 INT0 Interrupt (Rising and Falling Edges)

C language code

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

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

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

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;

void INT0_Isr() interrupt 0
{
    if (INT0)                       //Judging rising and falling edges
    {
        P10 = !P10;                 //Test port
    }
    else
    {
        P11 = !P11;                 //Test port
    }
}

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

    IT0 = 0;                        //Enable INT0 rising edge and falling edge interrupts
```

```
    EX0 = 1;                                    //Enable INT0 interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            ORG     0003H
            LJMP    INT0ISR

            ORG     0100H
INT0ISR:
            JB      INT0,RISING         ;Judging rising and falling edges
            CPL     P1.0                ;Test port
            RETI
RISING:
            CPL     P1.1                ;Test port
            RETI

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

            CLR     IT0                 ;Enable INT0 rising edge and falling edge interrupts
            SETB    EX0                 ;Enable INT0 interrupt
            SETB    EA
            JMP     $
            END
```

STC MCU

11.5.2 INT0 Interrupt (Falling Edge)

C language code

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

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

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

sbit    P10     =   P1^0;

void INT0_Isr() interrupt 0
{
    P10 = !P10;                                 //Test port
}

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

    IT0 = 1;                                    //Enable INT0 falling edge interrupt
    EX0 = 1;                                    //Enable INT0 interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P0M1        DATA        093H
```

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

            ORG     0000H
            LJMP    MAIN
            ORG     0003H
            LJMP    INT0ISR

            ORG     0100H
INT0ISR:
            CPL     P1.0                ;Test port
            RETI

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

            SETB    IT0                 ;Enable INT0 falling edge interrupt
            SETB    EX0                 ;Enable INT0 interrupt
            SETB    EA
            JMP     $

            END
```

11.5.3 INT1 Interrupt (Rising and Falling Edges)

C language code

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

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

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

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

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;

void INT1_Isr() interrupt 2
{
    if (INT1)                       //Judging rising and falling edges
    {
        P10 = !P10;                 //Test port
    }
    else
    {
        P11 = !P11;                 //Test port
    }
}

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

    IT1 = 0;                        //Enable INT1 rising edge and falling edge interrupts
    EX1 = 1;                        //Enable INT1 interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
```

```
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0013H
            LJMP    INT1ISR

            ORG     0100H
INT1ISR:
            JB      INT1,RISING     ;Judging rising and falling edges
            CPL     P1.0            ;Test port
            RETI
RISING:
            CPL     P1.1            ;Test port
            RETI

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

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

            END
```

11.5.4 INT1 Interrupt (Falling Edge)

C language code

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

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

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
```

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

sbit    P10     =   P1^0;

void INT1_Isr() interrupt 2
{
    P10 = !P10;                                 //Test port
}

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

    IT1 = 1;                                    //Enable INT1 falling edge interrupt
    EX1 = 1;                                    //Enable INT1 interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            ORG     0013H
            LJMP    INT1ISR

            ORG     0100H
```

```
INT1ISR:
        CPL     P1.0            ;Test port
        RETI

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

        SETB    IT1             ;Enable INT1 falling edge interrupt
        SETB    EX1             ;Enable INT1 interrupt
        SETB    EA
        JMP     $

        END
```

11.5.5 INT2 Interrupt (Falling Edge)

C language code

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

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

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

sfr     INTCLKO =   0x8f;
#define EX2         0x10
#define EX3         0x20
#define EX4         0x40
sbit    P10     =   P1^0;

void INT2_Isr() interrupt 10
{
```

```
    P10 = !P10;                                 //Test port
}

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

    INTCLKO = EX2;                              //Enable INT2 interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     0053H
            LJMP    INT2ISR

            ORG     0100H
INT2ISR:
            CPL     P1.0                    ;Test port
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
```

```
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     INTCLKO,#EX2        ;Enable INT2 interrupt
        SETB    EA
        JMP     $

        END
```

11.5.6 INT3 Interrupt (Falling Edge)

C language code

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

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

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

sfr     INTCLKO =   0x8f;
#define EX2         0x10
#define EX3         0x20
#define EX4         0x40
sbit    P10     =   P1^0;

void INT3_Isr() interrupt 11
{
    P10 = !P10;                         //Test port
}

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

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

    INTCLKO = EX3;                          //Enable INT3 interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     005BH
            LJMP    INT3ISR

            ORG     0100H
INT3ISR:
            CPL     P1.0                ;Test port
            RETI

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

```
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     INTCLKO,#EX3        ;Enable INT3 interrupt
        SETB    EA
        JMP     $

        END
```

11.5.7 INT4 Interrupt (Falling Edge)

C language code

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

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

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

sfr     INTCLKO =   0x8f;
#define EX2         0x10
#define EX3         0x20
#define EX4         0x40
sbit    P10     =   P1^0;

void INT4_Isr() interrupt 16
{
    P10 = !P10;                     //Test port
}

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

```
    P5M1 = 0x00;

    INTCLKO = EX4;                          //Enable INT4 interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     0083H
            LJMP    INT4ISR

            ORG     0100H
INT4ISR:
            CPL     P1.0                ;Test port
            RETI

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

            MOV     INTCLKO,#EX4        ;Enable INT4 interrupt
            SETB    EA
            JMP     $
```

STC MCU

END

11.5.8 Timer0 Interrupt

C language code

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

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

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

sbit    P10     =   P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;                                 //Test port
}

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

    TMOD = 0x00;
    TL0 = 0x66;                                 //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                    //Start timer
    ET0 = 1;                                    //Enable timer0 interrupt
    EA = 1;

    while (1);
}
```

STC MCU

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            ORG     000BH
            LJMP    TM0ISR

            ORG     0100H
TM0ISR:
            CPL     P1.0            ;Test port
            RETI

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

            MOV     TMOD,#00H
            MOV     TL0,#66H        ;65536-11.0592M/12/1000
            MOV     TH0,#0FCH
            SETB    TR0             ;Start timer
            SETB    ET0             ;Enable timer0 interrupt
            SETB    EA

            JMP     $

            END
```

11.5.9 Timer1 Interrupt

C language code

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

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

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

sbit    P10     =   P1^0;

void TM1_Isr() interrupt 3
{
    P10 = !P10;                                 //Test port
}

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

    TMOD = 0x00;
    TL1 = 0x66;                                 //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                                    //Start timer
    ET1 = 1;                                    //Enable timer1 interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     001BH
            LJMP    TM1ISR

            ORG     0100H
TM1ISR:
            CPL     P1.0                ;Test port
            RETI

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

            MOV     TMOD,#00H
            MOV     TL1,#66H            ;65536-11.0592M/12/1000
            MOV     TH1,#0FCH
            SETB    TR1                 ;Start timer
            SETB    ET1                 ;Enable timer1 interrupt
            SETB    EA

            JMP     $

            END
```

11.5.10 Timer2 Interrupt

C language code

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

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

sfr     T2L     =   0xd7;
```

```
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2         0x04
sfr     AUXINTIF=   0xef;
#define T2IF        0x01

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

sbit    P10     =   P1^0;

void TM2_Isr() interrupt 12
{
    P10 = !P10;                             //Test port
}

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

    T2L = 0x66;                             //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;                            //Start timer
    IE2 = ET2;                              //Enable timer2 interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T2L     DATA    0D7H
T2H     DATA    0D6H
AUXR    DATA    8EH
IE2     DATA    0AFH
```

```
ET2         EQU     04H
AUXINTIF    DATA    0EFH
T2IF        EQU     01H

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

            ORG     0000H
            LJMP    MAIN
            ORG     0063H
            LJMP    TM2ISR

            ORG     0100H
TM2ISR:
            CPL     P1.0                ;Test port
            RETI

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

            MOV     T2L,#66H            ;65536-11.0592M/12/1000
            MOV     T2H,#0FCH
            MOV     AUXR,#10H           ;Start timer
            MOV     IE2,#ET2            ;Enable timer2 interrupt
            SETB    EA

            JMP     $

            END
```

11.5.11 Timer3 Interrupt

C language code

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

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

sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T4T3M   =   0xd1;
sfr     IE2     =   0xaf;
#define ET3             0x20
sfr     AUXINTIF=   0xef;
#define T3IF            0x02

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

sbit    P10     =   P1^0;

void TM3_Isr() interrupt 19
{
    P10 = !P10;                 //Test port
}

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

    T3L = 0x66;                 //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x08;               //Start timer
    IE2 = ET3;                  //Enable timer3 interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T3L     DATA    0D5H
T3H     DATA    0D4H
T4T3M   DATA    0D1H
IE2     DATA    0AFH
ET3     EQU     20H
AUXINTIF DATA   0EFH
T3IF    EQU     02H

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

        ORG     0000H
        LJMP    MAIN
        ORG     009BH
        LJMP    TM3ISR

        ORG     0100H
TM3ISR:
        CPL     P1.0            ;Test port
        RETI

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

        MOV     T3L,#66H        ;65536-11.0592M/12/1000
        MOV     T3H,#0FCH
        MOV     T4T3M,#08H      ;Start timer
        MOV     IE2,#ET3        ;Enable timer3 interrupt
        SETB    EA

        JMP     $

        END
```

STC MCU

11.5.12 Timer4 Interrupt

C language code

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

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

sfr         T3L             =   0xd5;
sfr         T3H             =   0xd4;
sfr         T4L             =   0xd3;
sfr         T4H             =   0xd2;
sfr         T4T3M           =   0xd1;
sfr         IE2             =   0xaf;
#define     ET3                 0x20
#define     ET4                 0x40
sfr         AUXINTIF        =   0xef;
#define     T3IF                0x02
#define     T4IF                0x04

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

sbit        P10             =   P1^0;

void TM4_Isr() interrupt 20
{
    P10 = !P10;                         //Test port
}

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

STC MCU

```
    T4L = 0x66;                 //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0x80;               //Start timer
    IE2 = ET4;                  //Enable timer4 interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T3L         DATA    0D5H
T3H         DATA    0D4H
T4L         DATA    0D3H
T4H         DATA    0D2H
T4T3M       DATA    0D1H
IE2         DATA    0AFH
ET3         EQU     20H
ET4         EQU     40H
AUXINTIF    DATA    0EFH
T3IF        EQU     02H
T4IF        EQU     04H

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

            ORG     0000H
            LJMP    MAIN
            ORG     00A3H
            LJMP    TM4ISR

            ORG     0100H
TM4ISR:
            CPL     P1.0            ;Test port
            RETI

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

STC MCU

```
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     T4L,#66H                ;65536-11.0592M/12/1000
        MOV     T4H,#0FCH
        MOV     T4T3M,#80H              ;Start timer
        MOV     IE2,#ET4                ;Enable timer4 interrupt
        SETB    EA

        JMP     $

        END
```

11.5.13 UART1 Interrupt

C language code

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

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

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;

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

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;

void UART1_Isr() interrupt 4
{
    if (TI)
    {
        TI = 0;                         //Clear interrupt flag
        P10 = !P10;                     //Test port
    }
    if (RI)
    {
        RI = 0;                         //Clear interrupt flag
        P11 = !P11;                     //Test port
    }
}
```

STC MCU

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

    SCON = 0x50;
    T2L = 0xe8;                     //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x15;                    //Start timer
    ES = 1;                         //Enable UART1 interrupt
    EA = 1;
    SBUF = 0x5a;                    // Send test data

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T2L     DATA    0D7H
T2H     DATA    0D6H
AUXR    DATA    8EH

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

        ORG     0000H
        LJMP    MAIN
        ORG     0023H
        LJMP    UART1ISR

        ORG     0100H
UART1ISR:
        JNB     TI,CHECKRI
        CLR     TI              ;Clear interrupt flag
        CPL     P1.0            ;Test port
CHECKRI:
```

```
            JNB     RI,ISREXIT
            CLR     RI                  ;Clear interrupt flag
            CPL     P1.1                ;Test port
ISREXIT:
            RETI

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

            MOV     SCON,#50H
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#15H           ;Start timer
            SETB    ES                  ;Enable UART1 interrupt
            SETB    EA
            MOV     SBUF,#5AH           ; Send test data

            JMP     $

            END
```

11.5.14 UART2 Interrupt

C language code

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

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

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     S2CON   =   0x9a;
sfr     S2BUF   =   0x9b;
sfr     IE2     =   0xaf;
#define ES2         0x01

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
```

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

sbit    P12     =   P1^2;
sbit    P13     =   P1^3;

void UART2_Isr() interrupt 8
{
    if (S2CON & 0x02)
    {
        S2CON &= ~0x02;                 //Clear interrupt flag
        P12 = !P12;                     //Test port
    }
    if (S2CON & 0x01)
    {
        S2CON &= ~0x01;                 //Clear interrupt flag
        P13 = !P13;                     //Test port
    }
}

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

    S2CON = 0x10;
    T2L = 0xe8;                         //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x14;                        //Start timer
    IE2 = ES2;                          //Enable UART2 interrupt
    EA = 1;
    S2BUF = 0x5a;                       // Send test data

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T2L     DATA    0D7H
T2H     DATA    0D6H
AUXR    DATA    8EH
S2CON   DATA    9AH
S2BUF   DATA    9BH
```

```
IE2         DATA    0AFH
ES2         EQU     01H

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

            ORG     0000H
            LJMP    MAIN
            ORG     0043H
            LJMP    UART2ISR

            ORG     0100H
UART2ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     A,S2CON
            JNB     ACC.1,CHECKRI
            ANL     S2CON,#NOT 02H      ;Clear interrupt flag
            CPL     P1.2                ;Test port
CHECKRI:
            MOV     A,S2CON
            JNB     ACC.0,ISREXIT
            ANL     S2CON,#NOT 01H      ;Clear interrupt flag
            CPL     P1.3                ;Test port
ISREXIT:
            POP     PSW
            POP     ACC
            RETI

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

            MOV     S2CON,#10H
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H           ;Start timer
            MOV     IE2,#ES2            ;Enable UART2 interrupt
```

STC MCU

```
        SETB    EA
        MOV     S2BUF,#5AH          ; Send test data

        JMP     $

        END
```

11.5.15 UART3 Interrupt

C language code

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

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

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     S3CON   =   0xac;
sfr     S3BUF   =   0xad;
sfr     IE2     =   0xaf;
#define ES3         0x08

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

sbit    P12     =   P1^2;
sbit    P13     =   P1^3;

void UART3_Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;                 //Clear interrupt flag
        P12 = !P12;                     //Test port
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;                 //Clear interrupt flag
        P13 = !P13;                     //Test port
    }
}

void main()
{
```

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

    S3CON = 0x10;
    T2L = 0xe8;                         //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x14;                        //Start timer
    IE2 = ES3;                          //Enable UART3 interrupt
    EA = 1;
    S3BUF = 0x5a;                       //Send test data

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
S3CON       DATA    0ACH
S3BUF       DATA    0ADH
IE2         DATA    0AFH
ES3         EQU     08H

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

            ORG     0000H
            LJMP    MAIN
            ORG     008BH
            LJMP    UART3ISR

            ORG     0100H
UART3ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     A,S3CON
```

```
            JNB     ACC.1,CHECKRI
            ANL     S3CON,#NOT 02H      ;Clear interrupt flag
            CPL     P1.2                ;Test port
CHECKRI:
            MOV     A,S3CON
            JNB     ACC.0,ISREXIT
            ANL     S3CON,#NOT 01H      ;Clear interrupt flag
            CPL     P1.3                ;Test port
ISREXIT:
            POP     PSW
            POP     ACC
            RETI

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

            MOV     S3CON,#10H
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H           ;Start timer
            MOV     IE2,#ES3            ;Enable UART3 interrupt
            SETB    EA
            MOV     S3BUF,#5AH          ;Send test data

            JMP     $

            END
```

11.5.16 UART4 Interrupt

C language code

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

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

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     S4CON   =   0x84;
sfr     S4BUF   =   0x85;
sfr     IE2     =   0xaf;
#define ES4         0x10
```

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

sbit    P12     =   P1^2;
sbit    P13     =   P1^3;

void UART4_Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;                 //Clear interrupt flag
        P12 = !P12;                     //Test port
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;                 //Clear interrupt flag
        P13 = !P13;                     //Test port
    }
}

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

    S4CON = 0x10;
    T2L = 0xe8;                         //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x14;                        //Start timer
    IE2 = ES4;                          //Enable UART4 interrupt
    EA = 1;
    S4BUF = 0x5a;                       //Send test data

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T2L     DATA    0D7H
T2H     DATA    0D6H
AUXR    DATA    8EH
S4CON   DATA    84H
S4BUF   DATA    85H
IE2     DATA    0AFH
ES4     EQU     10H

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

        ORG     0000H
        LJMP    MAIN
        ORG     0093H
        LJMP    UART4ISR

        ORG     0100H
UART4ISR:
        PUSH    ACC
        PUSH    PSW
        MOV     A,S4CON
        JNB     ACC.1,CHECKRI
        ANL     S4CON,#NOT 02H      ;Clear interrupt flag
        CPL     P1.2                ;Test port
CHECKRI:
        MOV     A,S4CON
        JNB     ACC.0,ISREXIT
        ANL     S4CON,#NOT 01H      ;Clear interrupt flag
        CPL     P1.3                ;Test port
ISREXIT:
        POP     PSW
        POP     ACC
        RETI

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

STC MCU

```
        MOV     P5M1, #00H

        MOV     S4CON,#10H
        MOV     T2L,#0E8H       ;65536-11059200/115200/4=0FFE8H
        MOV     T2H,#0FFH
        MOV     AUXR,#14H       ;Start timer
        MOV     IE2,#ES4        ;Enable UART4 interrupt
        SETB    EA
        MOV     S4BUF,#5AH      ;Send test data

        JMP     $

        END
```

11.5.17 ADC Interrupt

C language code

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

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

sfr     ADC_CONTR   =   0xbc;
sfr     ADC_RES     =   0xbd;
sfr     ADC_RESL    =   0xbe;
sfr     ADCCFG      =   0xde;
sbit    EADC        =   IE^5;

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

void ADC_Isr() interrupt 5
{
    ADC_CONTR &= ~0x20;             //Clear interrupt flag
    P0 = ADC_RES;                   //Test port
    P2 = ADC_RESL;                  //Test port
}

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

STC MCU

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

    ADCCFG = 0x00;
    ADC_CONTR = 0xc0;                   //Enable and start the ADC module
    EADC = 1;                           //Enable ADC interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

ADC_CONTR   DATA    0BCH
ADC_RES     DATA    0BDH
ADC_RESL    DATA    0BEH
ADCCFG      DATA    0DEH
EADC        BIT     IE.5

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

            ORG     0000H
            LJMP    MAIN
            ORG     002BH
            LJMP    ADCISR

            ORG     0100H
ADCISR:
            ANL     ADC_CONTR,#NOT 20H  ;Clear interrupt flag
            MOV     P0,ADC_RES          ;Test port
            MOV     P2,ADC_RESL         ;Test port
            RETI

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

STC MCU

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

        MOV     ADCCFG,#00H
        MOV     ADC_CONTR,#0C0H     ;Enable and start the ADC module
        SETB    EADC                ;Enable ADC interrupt
        SETB    EA

        JMP     $

        END
```

11.5.18 LVD Interrupt

C language code

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

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

sfr     RSTCFG  =   0xff;
#define ENLVR       0x40        //RSTCFG.6
#define LVD2V2      0x00        //LVD@2.2V
#define LVD2V4      0x01        //LVD@2.4V
#define LVD2V7      0x02        //LVD@2.7V
#define LVD3V0      0x03        //LVD@3.0V
sbit    ELVD    =   IE^6;
#define LVDF        0x20        //PCON.5

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;
sbit    P10     =   P1^0;

void LVD_Isr() interrupt 6
{
    PCON &= ~LVDF;              //Clear interrupt flag
    P10 = !P10;                 //Test port
}

void main()
{
```

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

    PCON &= ~LVDF;                  //Interrupt flag needs to be cleared at power-on
    RSTCFG = LVD3V0;                //Set the LVD voltage to 3.0V
    ELVD = 1;                       //Enable LVD interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

RSTCFG      DATA    0FFH
ENLVR       EQU     40H             ;RSTCFG.6
LVD2V2      EQU     00H             ;LVD@2.2V
LVD2V4      EQU     01H             ;LVD@2.4V
LVD2V7      EQU     02H             ;LVD@2.7V
LVD3V0      EQU     03H             ;LVD@3.0V
ELVD        BIT     IE.6
LVDF        EQU     20H             ;PCON.5

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

            ORG     0000H
            LJMP    MAIN
            ORG     0033H
            LJMP    LVDISR

            ORG     0100H
LVDISR:
            ANL     PCON,#NOT LVDF  ;Clear interrupt flag
            CPL     P1.0            ;Test port
            RETI

MAIN:
```

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

        ANL     PCON,#NOT LVDF      ;Interrupt flag needs to be cleared at power-on
        MOV     RSTCFG,# LVD3V0     ;Set the LVD voltage to 3.0V
        SETB    ELVD                ;Enable LVD interrupt
        SETB    EA
        JMP     $

        END
```

11.5.19 PCA Interrupt

C language code

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

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

sfr     CCON    =   0xd8;
sbit    CF      =   CCON^7;
sbit    CR      =   CCON^6;
sbit    CCF3    =   CCON^3;
sbit    CCF2    =   CCON^2;
sbit    CCF1    =   CCON^1;
sbit    CCF0    =   CCON^0;
sfr     CMOD    =   0xd9;
sfr     CL      =   0xe9;
sfr     CH      =   0xf9;
sfr     CCAPM0  =   0xda;
sfr     CCAP0L  =   0xea;
sfr     CCAP0H  =   0xfa;
sfr     PCA_PWM0=   0xf2;
sfr     CCAPM1  =   0xdb;
sfr     CCAP1L  =   0xeb;
sfr     CCAP1H  =   0xfb;
sfr     PCA_PWM1=   0xf3;
sfr     CCAPM2  =   0xdc;
sfr     CCAP2L  =   0xec;
sfr     CCAP2H  =   0xfc;
sfr     PCA_PWM2=   0xf4;

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

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

sbit    P10     =   P1^0;

void PCA_Isr() interrupt 7
{
    if (CF)
    {
        CF = 0;                 //Clear interrupt flag
        P10 = !P10;             //Test port
    }
}

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

    CCON = 0x00;
    CMOD = 0x09;                //PCA clock is the system clock, enable PCA timing interrupt
    CR = 1;                     //Start PCA timer
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CCON    DATA    0D8H
CF      BIT     CCON.7
CR      BIT     CCON.6
CCF3    BIT     CCON.3
CCF2    BIT     CCON.2
CCF1    BIT     CCON.1
CCF0    BIT     CCON.0
CMOD    DATA    0D9H
CL      DATA    0E9H
```

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

            ORG     0100H
PCAISR:
            JNB     CF,ISREXIT
            CLR     CF              ;Clear interrupt flag
            CPL     P1.0            ;Test port
ISREXIT:
            RETI

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

            MOV     CCON,#00H
            MOV     CMOD,#09H       ;PCA clock is the system clock, enable PCA timing
interrupt
            SETB    CR              ;Start PCA timer
```

STC MCU

```
        SETB    EA

        JMP     $

        END
```

11.5.20 SPI Interrupt

C language code

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

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

sfr     SPSTAT  =   0xcd;
sfr     SPCTL   =   0xce;
sfr     SPDAT   =   0xcf;
sfr     IE2     =   0xaf;
#define ESPI        0x02

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

sbit    P10     =   P1^0;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                  //Clear interrupt flag
    P10 = !P10;                     //Test port
}

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

STC MCU

```
    SPCTL = 0x50;                           //Enable SPI master mode
    SPSTAT = 0xc0;                          //Clear interrupt flag
    IE2 = ESPI;                             //Enable SPI interrupt
    EA = 1;
    SPDAT = 0x5a;                           //Send test data

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            MOV     SPSTAT,#0C0H        ;Clear interrupt flag
            CPL     P1.0                ;Test port
            RETI

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

STC MCU

```
        MOV     SPCTL,#50H          ;Enable SPI master mode
        MOV     SPSTAT,#0C0H        ;Clear interrupt flag
        MOV     IE2,#ESPI           ;Enable SPI interrupt
        SETB    EA
        MOV     SPDAT,#5AH          ;Send test data

        JMP     $

        END
```

11.5.21 CMP Interrupt

C language code

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

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

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

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

sbit    P10     =   P1^0;

void CMP_Isr() interrupt 21
{
    CMPCR1 &= ~0x40;                //Clear interrupt flag
    P10 = !P10;                     //Test port
}

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

STC MCU

```
    CMPCR2 = 0x00;
    CMPCR1 = 0x80;                  //Enable comparator module
    CMPCR1 |= 0x30;                 //Enable edge interrupt of comparator
    CMPCR1 &= ~0x08;                //P3.6 is CMP + input pin
    CMPCR1 |= 0x04;                 //P3.7 is CMP- input pin
    CMPCR1 |= 0x02;                 //Enable Comparator output
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

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

            ORG     0000H
            LJMP    MAIN
            ORG     00ABH
            LJMP    CMPISR

            ORG     0100H
CMPISR:
            ANL     CMPCR1,#NOT 40H     ;Clear interrupt flag
            CPL     P1.0                ;Test port
            RETI

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

            MOV     CMPCR2,#00H
```

STC MCU

```
        MOV     CMPCR1,#80H             ;Enable comparator module
        ORL     CMPCR1,#30H             ;Enable edge interrupt of comparator
        ANL     CMPCR1,#NOT 08H         ;P3.6 is CMP + input pin
        ORL     CMPCR1,#04H             ;P3.7 is CMP- input pin
        ORL     CMPCR1,#02H             ;Enable Comparator output
        SETB    EA

        JMP     $

        END
```

11.5.22 PWM Interrupt

C language code

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

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

sfr     P_SW2       =   0xba;

sfr     PWMSET      =   0xF1;
sfr     PWMCFG01    =   0xF6;

#define PWM0CH          (*(unsigned char volatile xdata *)0xFF00)
#define PWM0CL          (*(unsigned char volatile xdata *)0xFF01)
#define PWM0CKS         (*(unsigned char volatile xdata *)0xFF02)

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

sbit    P10         =   P1^0;
sbit    P11         =   P1^1;

void PWM0_Isr() interrupt 22
{
    if (PWMCFG01 & 0x08)
    {
        PWMCFG01 &= ~0x08;                  //Clear interrupt flag
        P10 = !P10;                         //Test port
    }
}

void main()
{
```

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

    PWMSET = 0x01;                  //Enable PWM0 module (The configuration is effective only after the module is enabled.)

    P_SW2 = 0x80;
    PWM0CKS = 0x0f;                 //PWM0 clock is system clock/16.
    PWM0CH = 0x01;                  //Set the PWM0 period to 256 PWM0 clocks
    PWM0CL = 0x00;
    P_SW2 = 0x00;

    PWMCFG01 = 0x05;                //Start PWM0 module and enable PWM0 counter interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

PWMSET      DATA    0F1H
PWMCFG01    DATA    0F6H

PWM0CH      EQU     0FF00H
PWM0CL      EQU     0FF01H
PWM0CKS     EQU     0FF02H

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

            ORG     0000H
            LJMP    MAIN
            ORG     00B3H
            LJMP    PWM0ISR
```

```
            ORG     0100H
PWM0ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     A,PWMCFG01
            JNB     ACC.3,ISREXIT
            ANL     PWMCFG01,#NOT 08H       ;Clear interrupt flag
            CPL     P1.0                    ;Test port
ISREXIT:
            POP     PSW
            POP     ACC
            RETI

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

            MOV     PWMSET,#01H             ;Enable PWM0 module (The configuration is effective only after the module is enabled.)

            MOV     P_SW2,#80H
            MOV     A,#0FH
            MOV     DPTR,#PWM0CKS
            MOVX    @DPTR,A                 ;PWM0 clock is system clock/16.
            MOV     A,#01H
            MOV     DPTR,#PWM0CH            ;Set the PWM0 period to 256 PWM0 clocks
            MOVX    @DPTR,A
            MOV     A,#00H
            MOV     DPTR,#PWM0CL
            MOVX    @DPTR,A
            MOV     P_SW2,#00H

            MOV     PWMCFG01,#05H           ;Start PWM0 module and enable PWM0 counter interrupt
            SETB    EA

            JMP     $

            END
```

11.5.23 I2C Interrupt

C language code

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

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

sfr         P_SW2           =   0xba;

#define     I2CCFG              (*(unsigned char volatile xdata *)0xfe80)
#define     I2CMSCR             (*(unsigned char volatile xdata *)0xfe81)
#define     I2CMSST             (*(unsigned char volatile xdata *)0xfe82)
#define     I2CSLCR             (*(unsigned char volatile xdata *)0xfe83)
#define     I2CSLST             (*(unsigned char volatile xdata *)0xfe84)
#define     I2CSLADR            (*(unsigned char volatile xdata *)0xfe85)
#define     I2CTXD              (*(unsigned char volatile xdata *)0xfe86)
#define     I2CRXD              (*(unsigned char volatile xdata *)0xfe87)

sfr         P0M1            =   0x93;
sfr         P0M0            =   0x94;
sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P2M1            =   0x95;
sfr         P2M0            =   0x96;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P4M1            =   0xb3;
sfr         P4M0            =   0xb4;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;
sbit        P10             =   P1^0;

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;
    if (I2CMSST & 0x40)
    {
        I2CMSST &= ~0x40;                   //Clear interrupt flag
        P10 = !P10;                         //Test port
    }
    _pop_(P_SW2);
}

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

    P_SW2 = 0x80;
    I2CCFG = 0xc0;                          //Enable I2C master mode
    I2CMSCR = 0x80;                         //Enable I2C interrupt;
    P_SW2 = 0x00;
```

STC MCU

```
    EA = 1;

    P_SW2 = 0x80;
    I2CMSCR = 0x81;                         //Send start command
    P_SW2 = 0x00;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

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

            ORG     0000H
            LJMP    MAIN
            ORG     00C3H
            LJMP    I2CISR

            ORG     0100H
I2CISR:
            PUSH    ACC
            PUSH    DPL
            PUSH    DPH
            PUSH    P_SW2
            MOV     P_SW2,#80H
            MOV     DPTR,#I2CMSST
            MOVX    A,@DPTR
            ANL     A,#NOT 40H          ;Clear interrupt flag
            MOVX    @DPTR,A
            CPL     P1.0                ;Test port
            POP     P_SW2
            POP     DPH
            POP     DPL
            POP     ACC
            RETI
```

STC MCU

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

        MOV     P_SW2,#80H
        MOV     A,#0C0H             ;Enable I2C master mode
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#80H              ;Enable I2C interrupt
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        MOV     P_SW2,#00H
        SETB    EA

        MOV     P_SW2,#80H
        MOV     A,#081H             ;Send start command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        MOV     P_SW2,#00H

        JMP     $

        END
```

STC MCU

12 Timers/Counters

Product line	Number of timers
STC8G1K08 family	**3**
STC8G1K08-8Pin family	**2**
STC8G1K08A family	**2**
STC8G2K64S4 family	**5**
STC8G2K64S2 family	**5**
STC8G1K08T family	**3**
STC15H2K64S4 family	**5**

Five 16-bit Timers/Counters are integrated in STC8G series of microcontrollers: T0, T1, T2, T3 and T4. All of them can be used as timer or counter. For T0 and T1, the 'timer' or 'counter' function is selected by the control bits C/T in the special function register TMOD. For T2, the 'timer' or 'counter' function is selected by the control bit T2_C/T in the special function register AUXR. For T3, the 'timer' or 'counter' function is selected by the control bit T3_C/T in the special function register T4T3M. For T4, the 'timer' or 'counter' function is selected by the control bit T4_C/T in the special function register T4T3M. The core of the timer/counter is a up counter, the essence of which is counting pulses. The only difference of 'timer' mode and 'counter' mode is the different counting pulses sources. If the counting pulse is from the system clock, the timer/counter runs in the timing mode, it counts once every 12 clocks or one clock. If the counting pulse is from the microcontroller external pins, the timer/counter runs in counting mode, it counts once every pulse.

When T0, T1 and T2 are operating in 'timer' mode, T0x12, T1x12 and T2x12 in AUXR register are used to determine the clocks of T0, T1 and T2 are system clock/12 or system clock/1. When T3 and T4 are operating in 'timer' mode, T3x12 and T4x12 in the T4T3M register determine the clocks of T3 and T4 are system clock/12 or system clock/1. When the timer/counters are operating in 'counter' mode, the frequency of the external pulse is not divided.

T0 has four operating modes which are selected by bit-pairs (M1, M0) in TMOD. The four modes are mode 0 (16-bit auto-reload mode), mode 1 (16-bit non-auto-reload mode), mode 2 (8-bit auto-reload mode) and mode 3 (16-bit auto-reload mode whose interrupt can not be disabled). And for T1, all modes except mode 3 are the same as T0. The mode 3 of T1 is invalid and stops counting. For T2, T3 and T4, there is only one mode, 16-bit auto-reload mode. Besides being used as timer/counters, T2, T3 and T4 can also be as the baud-rate generators of UARTs and programmable clock outputs.

12.1 Registers Related to Timers

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
TCON	Timer 0 and 1 control register	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
TMOD	Timer 0 and 1 mode register	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000,0000
TL0	Timer 0 low byte register	8AH									0000,0000
TL1	Timer 1 low byte register	8BH									0000,0000
TH0	Timer 0 high byte register	8CH									0000,0000
TH1	Timer 1 high byte register	8DH									0000,0000
AUXR	Auxiliary register 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
INTCLKO	External interrupt and clock output control register	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
WKTCL	Wake-up Timer Control Register low	AAH									1111,1111
WKTCH	Wake-up Timer Control Register high byte	ABH	WKTEN								0111,1111
T4T3M	Timer4 and Timer 3 Control Register	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000,0000
T4H	Timer 4 high byte register	D2H									0000,0000
T4L	Timer 4 low byte register	D3H									0000,0000

T3H	Timer 3 high byte register	D4H		0000,0000
T3L	Timer 3 low byte register	D5H		0000,0000
T2H	Timer 2 high byte register	D6H		0000,0000
T2L	Timer 2 low byte register	D7H		0000,0000

Symbol	Description	Address	Bit Address and Symbol								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
TM2PS	Clock prescaler register of timer2	FEA2H									0000,0000
TM3PS	Clock prescaler register of timer3	FEA3H									0000,0000
TM4PS	Clock prescaler register of timer4	FEA4H									0000,0000

STC MCU

12.2 Timer 0/1

12.2.1 Timer 0 and 1 Control Register

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

TF1: T1 overflow flag. After T1 is enabled to count, it performs adding 1 count from the initial value. TF1 is set by hardware on T1 overflow and requests interrupt to CPU. It will keep the status until CPU responds the interrupt and is cleared by hardware automatically. It also can be cleared by software.

TR1: T1 run control bit. It is set or cleared by software to turn the timer on/off. If GATE (TMOD.7) = 0, T1 will start counting as soon as TR1=1 and stop counting when TR1=0. If GATE (TMOD.7) = 1, T1 is enabled to count only if TR1 = 1 and INT1 is high.

TF0: T0 overflow flag. After T0 is enabled to count, it performs adding 1 count from the initial value. TF0 is set by hardware on T0 overflow and requests interrupt to CPU. It will keep the status until CPU responds the interrupt and is cleared by hardware automatically. It also can be cleared by software.

TR0: T0 run control bit. It is set or cleared by software to turn the timer on/off. If GATE (TMOD.3) = 0, T0 will start counting as soon as TR0=1 and stop counting when TR0=0. If GATE (TMOD.0) = 1, T0 is enabled to count only if TR0 = 1 and INT0 is high.

IE1: External Interrupt 1 (INT1/P3.3) request flag. IE1=1 means external interrupt requests interrupt to CPU.It is cleared by hardware automatically when the CPU responds to the interrupt.

IT1: External Intenupt 1 trigger edge type control bit. If IT1 = 0, INT1 can be triggered by both rising and falling edges. If IT1 = 1, INT1 can be triggered only by falling edge.

IE0: External Interrupt 0 (INT0/P3.2) request flag. IE0=1 means external interrupt requests interrupt to CPU.It is cleared by hardware automatically when the CPU responds to the interrupt.

IT0: External Intenupt 0 trigger edge type control bit. If IT0 = 0, INT0 can be triggered by both rising and falling edges. If IT0 = 1, INT0 can be triggered only by falling edge.

12.2.2 Timer 0/1 Mode Register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TMOD	89H	T1_GATE	T1_C/T	T1_M1	T1_M0	T0_GATE	T0_C/T	T0_M1	T0_M0

T1_GATE: T1 gate control. If GATE/TMOD.7=1, T1 starts only when TR1 is set AND INT1 pin is high.

T0_GATE: T0 gate control. If GATE/TMOD.3 =1, T0 starts only when TR0 is set AND INT0 pin is high.

T1_C/T: T1 mode select bit. If it is reset, T1 is used as a timer (input pulse is from internal system clock). If it is set, T1 is used as a counter (input pulse is from external T1/P3.5 pin).

T0_C/T: T0 mode select bit. If it is reset, T0 is used as a timer (input pulse is from internal system clock). If it is set, T0 is used as a counter (input pulse is from external T0/P3.4 pin).

T1_M1/T1_M0: T1 mode select bits.

T1_M1	T1_M0	T1 operating mode
0	0	16-bit auto-reload mode. When the 16-bit counter [TH1, TL1] overflows, the system loads the reload value

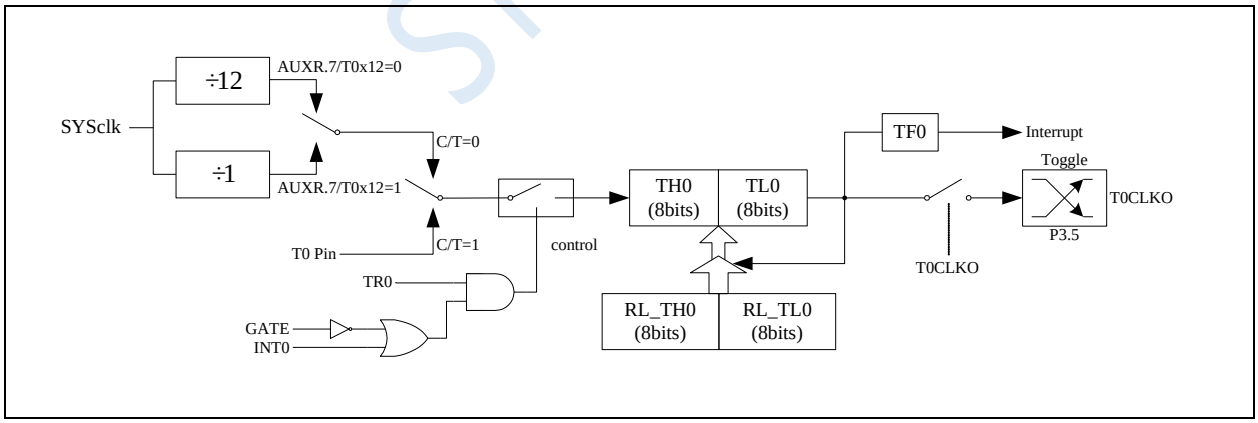
		in the internal 16-bit reload register into [TH1, TL1] automatically.
0	1	16-bit non-auto-reload mode. When the 16-bit counter [TH1, TL1] overflows, timer 1 will count from 0.
1	0	8-bit auto-reload mode. When the 8-bit counter TL1 overflows, the system loads the reload value in TH1 into TL1 automatically.
1	1	T1 stops working.

T0_M1/T0_M0: T0 mode select bits.

T0_M1	T0_M0	T0 operating mode
0	0	16-bit auto-reload mode. When the 16-bit counter [TH0, TL0] overflows, the system loads the reload value in the internal 16-bit reload register into [TH0, TL0] automatically.
0	1	16-bit non-auto-reload mode. When the 16-bit counter [TH1, TL1] overflows, timer 1 will count from 0.
1	0	8-bit auto-reload mode. When the 8-bit counter TL0 overflows, the system loads the reload value in TH0 into TL0 automatically.
1	1	Same as mode 0,16-bit automatic reload mode with non-maskable interrupt. The non-maskable interrupt with highest interrupt priority, that is whose priority is higher than the priority of all other interrupts, can not be closed, which can be used as the system tick timer of the operating system, or system monitoring timer.

12.2.3 Timer0 mode 0 (16-bit auto-reloadable mode)

In this mode, Timer/Counter 0 is used as a 16-bit counter that can be automatically reloaded, as shown in the figure below.



Timer/Counter0 mode 0: 16-bit auto-reload mode

When GATE=0 (TMOD.3), the timer will count if TR0=1. When GATE=1, it is allowed to control timer0 by external input INT0, so that pulse width measurement can be realized. TR0 is the control bit in the TCON register. For the specific function description of each bit of the TCON register, see the introduction of the TCON register in the previous section.

When C/T=0, the multiplexer is connected to the frequency division output of the system clock. T0 counts the internal system clock, and works in timing mode. When C/T=1, the multiplexer is connected to the external pulse input P3.4/T0, and T0 works in counting mode.

Timer0 of STC microcontroller has two counting rates: one is 12T mode, which is increased by 1 for every 12 clocks, which is the same as traditional 8051 microcontroller, the other is 1T mode, which is increased by 1 for each clock, and the speed is 12 times of traditional 8051. The rate of T0 is determined by T0x12 in the special function register AUXR. If T0x12=0, T0 works in 12T mode, and if T0x12=1, T0 works in 1T mode.

Timer0 has two hidden registers RL_TH0 and RL_TL0. RL_TH0 and TH0 share the same address, and RL_TL0 and TL0 share the same address. When TR0=0, that is, when Timer/Counter0 is disabled, the content written to TL0 will be written to RL_TL0 at the same time, and the content written to TH0 will also be written to RL_TH0 at the same time. When TR0=1, that is, when Timer/Counter0 is allowed to work, writing content to TL0 is not actually written to the current register TL0, but written to the hidden register RL_TL0, and writing content to TH0 is actually also it is not written into the current register TH0, but into the hidden register RL_TH0, which can cleverly realize the 16-bit reload timer. When reading the contents of TH0 and TL0, the contents be read are the contents of TH0 and TL0, not the contents of RL_TH0 and RL_TL0.

When Timer0 is working in mode 0 (TMOD[1:0]/[M1,M0]=00B), the overflow of [TH0,TL0] not only sets TF0, but also automatically reloads the contents of [RL_TH0,RL_TL0] to [TH0,TL0].

If T0CLKO/INT_CLKO.0=1, the P3.5/T1 pin is configured as timer 0's clock output T0CLKO. The output clock frequency is T0 overflow rate/2.

If C/T=0, the timer/counter 0 counts the internal system clock, then:

if T0 works in 1T mode (AUXR.7/T0x12=1), the output clock frequency = (SYSclk)/(65536-[RL_TH0, RL_TL0])/2

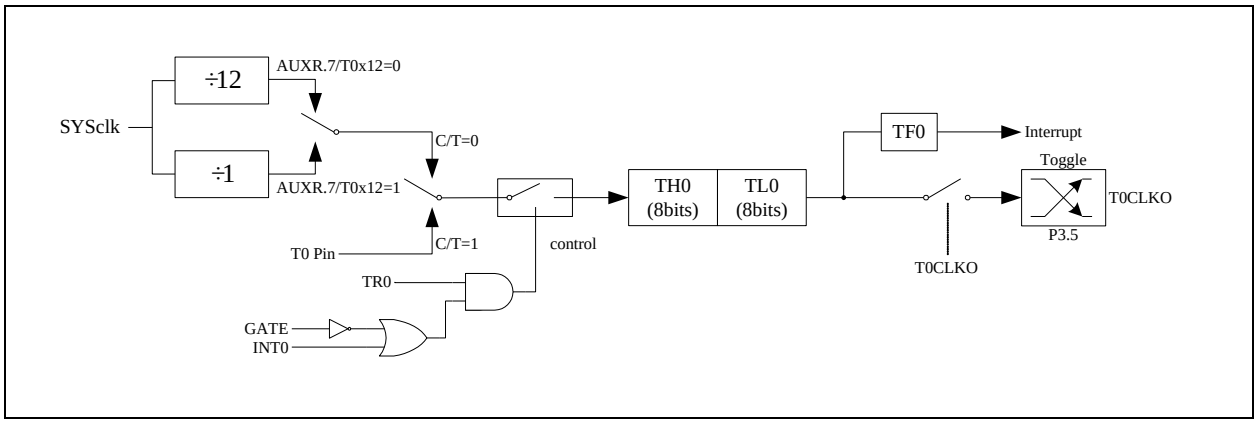
if T0 works in 12T mode (AUXR.7/T0x12=0), the output clock frequency when = (SYSclk)/12/(65536-[RL_TH0, RL_TL0])/2

If C/T=1, the timer/counter 0 counts the external pulse input (P3.4/T0), then:

the output clock frequency = (T0_Pin_CLK) / (65536-[RL_TH0, RL_TL0])/2

12.2.4 Timer0 mode 1 (16-bit non-autoreloadable mode)

In this mode, Timer/Counter 0 works in 16-bit non-reloadable mode, as shown in the figure below.



Timer/counter 0 mode 1: 16-bit non-reloadable mode

In this mode, Timer/Counter 0 is configured as a 16-bit non-reloadable mode, which is composed of 8 bits of TL0 and 8 bits of TH0. The 8-bit overflow of TL0 carries over to TH0, and the overflow of TH0 counts the overflow flag TF0 in TCON.

When GATE=0 (TMOD.3), the timer will count if TR0=1. When GATE=1, it is allowed to control timer 0

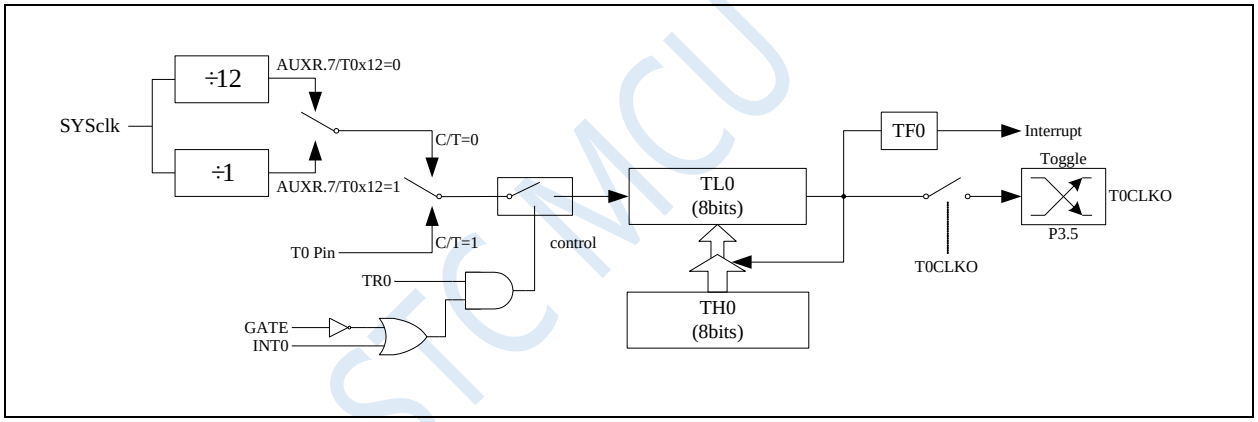
by external input INT0, so that pulse width measurement can be realized. TR0 is the control bit in the TCON register. For the specific function description of each bit of the TCON register, see the introduction of the TCON register in the previous section.

When C/T=0, the multiplexer is connected to the frequency division output of the system clock, T0 counts the internal system clock, and works in timing mode. When C/T=1, the multiplexer is connected to the external pulse input P3.4/T0, and T0 works in counting mode.

Timer0 of STC microcontroller has two counting rates: one is 12T mode, which is increased by 1 for every 12 clocks, which is the same as traditional 8051 microcontroller, the other is 1T mode, which is increased by 1 for each clock, and the speed is 12 times of traditional 8051. The rate of T0 is determined by T0x12 in the special function register AUXR. If T0x12=0, T0 works in 12T mode, and if T0x12=1, T0 works in 1T mode.

12.2.5 Timer 0 mode 2 (8-bit auto-reloadable mode)

In this mode, Timer/Counter 0 is an 8-bit counter that can be automatically reloaded, as shown in the figure below.



Timer/counter 0 mode 2: 8-bit auto-reloadable mode

The overflow of TL0 not only sets TF0, but also reloads the content of TH0 into TL0. The content of TH0 is preset by software, and its content remains unchanged during reloading.

When T0CLKO/INT_CLKO.0=1, the P3.5/T1 pin is configured as timer 0's clock output T0CLKO. The output clock frequency is T0 overflow rate/2.

If C/T=0, the timer/counter 0 counts the internal system clock, then:

if T0 works in 1T mode (AUXR.7/T0x12=1), the output clock frequency = (SYSclk)/(256-TH0)/2

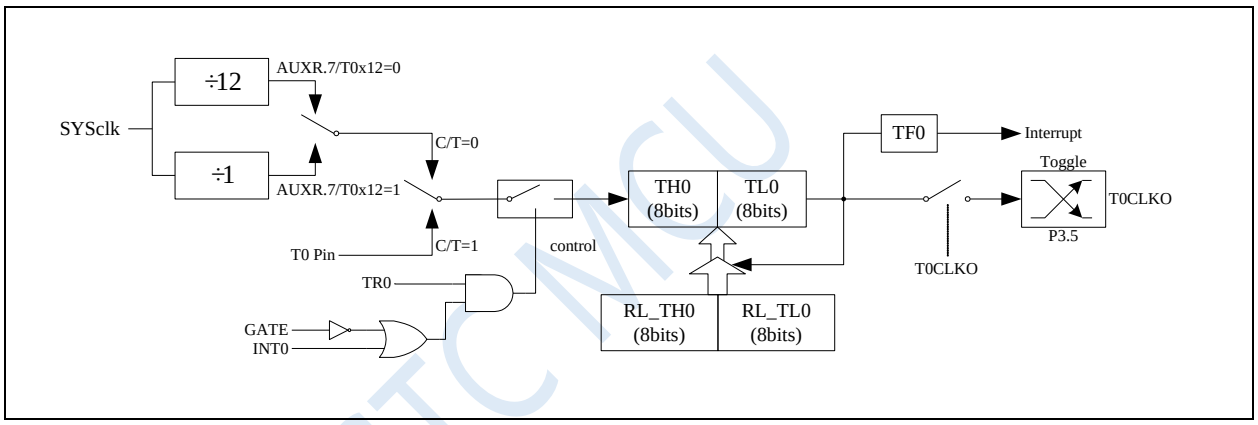
if T0 works in 12T mode (AUXR.7/T0x12=0), the output clock frequency = (SYSclk)/12/(256-TH0)/2

If C/T=1, the timer/counter T0 counts the external pulse input (P3.4/T0), then:

Output clock frequency = (T0_Pin_CLK) / (256-TH0)/2

12.2.6 Timer 0 mode 3 (16-bit auto-realoadable mode with non-maskable interrupt, which can be used as real-time operating system metronome)

For timer/counter 0, its working mode 3 is the same as working mode 0 (the schematic diagram of timer mode 3 in the figure below is the same as working mode 0). The only difference is: when timer/counter 0 is working in mode 3, its interrupt can be enabled just setting ET0/IE.1 (timer/counter 0 interrupt enable bit), and EA/IE.7 (total interrupt enable bit) is not required. The timer/counter 0 interrupt in this mode has nothing to do with the total interrupt enable bit EA. Once the timer/counter 0 interrupt working in mode 3 is enabled (ET0=1), then the interrupt is non-maskable, and the priority of the interrupt is the highest, that is, the interrupt cannot be interrupted by any interrupt, and the interrupt is neither controlled by EA/IE.7 nor controlled by ET0 after it is enabled, When EA=0 or ET0=0, this interrupt cannot be disabled. This mode is so called the 16-bit automatic reload mode with non-maskable interrupt.

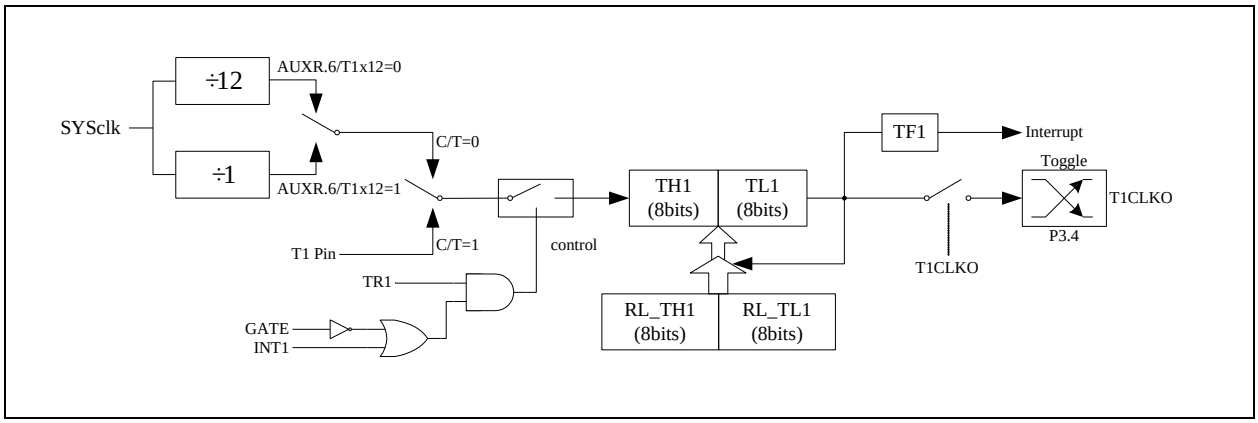


Timer/counter 0 mode 3: 16-bit auto-reload mode with non-maskable interrupt

Note: When Timer/Counter 0 works in mode 3 (16-bit auto-reload mode with non-maskable interrupt), it is not necessary to enable EA/IE.7 (total interrupt enable bit), only ET0/IE.1 is required. (Timer/counter 0 interrupt enable bit) can turn on the timer/counter 0 interrupt. The timer/counter 0 interrupt in this mode has nothing to do with the total interrupt enable bit EA. Once the timer/counter 0 interrupt in this mode is enabled, the timer/counter 0 interrupt priority is the highest, and it cannot be interrupted by any other interrupt (no matter it is lower than the timer/counter 0 interrupt priority). After the interrupt in this mode is enabled, it is neither controlled by EA/IE.7 nor controlled by ET0. Clearing EA nor ET0 can not disable this interrupt.

12.2.7 Timer 1 mode 0 (16-bit auto-reloadable mode)

In this mode, Timer/Counter 1 is used as a 16-bit counter that can be automatically reloaded, as shown in the figure below.



Timer/Counter 1 mode 0: 16-bit auto-reload mode

When GATE=0 (TMOD.7), the timer will count if TR1=1. When GATE=1, it is allowed to control timer1 by external input INT1, so that pulse width measurement can be realized. TR1 is the control bit in the TCON register. For the specific function description of each bit of the TCON register, see the introduction of the TCON register in the previous section.

When C/T=0, the multiplexer is connected to the frequency division output of the system clock. T1 counts the internal system clock, and works in timing mode. When C/T=1, the multiplexer is connected to the external pulse input P3.5/T1, and T1 works in counting mode.

Timer1 of STC microcontroller has two counting rates: one is 12T mode, which is increased by 1 for every 12 clocks, which is the same as traditional 8051 microcontroller, the other is 1T mode, which is increased by 1 for each clock, and the speed is 12 times of traditional 8051. The rate of T1 is determined by T1x12 in the special function register AUXR. If T1x12=0, T1 works in 12T mode, and if T1x12=1, T1 works in 1T mode.

Timer1 has two hidden registers RL_TH1 and RL_TL1. RL_TH1 and TH1 share the same address, and RL_TL1 and TL1 share the same address. When TR1=0, that is, when Timer/Counter1 is disabled, the content written to TL1 will be written to RL_TL1 at the same time, and the content written to TH1 will also be written to RL_TH1 at the same time. When TR1=1, that is, when Timer/Counter1 starts to work, writing content to TL1 is not actually written to the current register TL1, but written to the hidden register RL_TL1, and writing content to TH1 is actually also it is not written into the current register TH1, but into the hidden register RL_TH1, which can cleverly realize the 16-bit reload timer. When reading the contents of TH1 and TL1, the contents be read are the contents of TH1 and TL1, not the contents of RL_TH1 and RL_TL1.

When Timer1 is working in mode 0 (TMOD[5:4]/[M1,M0]=00B), the overflow of [TH1,TL1] not only sets TF1, but also automatically reloads the contents of [RL_TH1,RL_TL1] to [TH1,TL1].

If T1CLKO/INT_CLKO.1=1, the P3.4/T1 pin is configured as timer 1's clock output T1CLKO. The output clock frequency is T1 overflow rate/2.

If C/T=0, the timer/counter 1 counts the internal system clock, then:

if T1 works in 1T mode (AUXR.6/T1x12=1), the output clock frequency = (SYSclk)/(65536-[RL_TH1, RL_TL1])/2

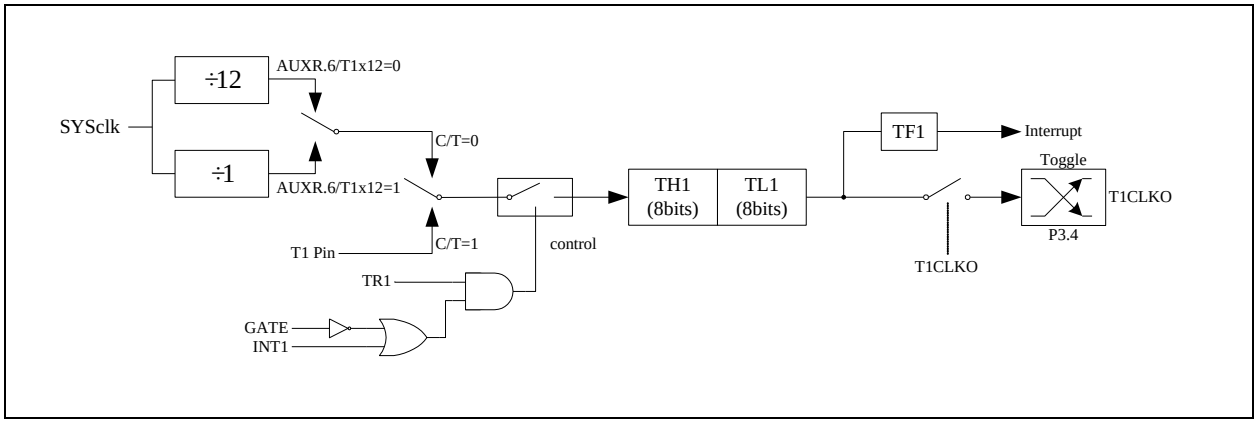
if T1 works in 12T mode (AUXR.6/T1x12=0), the output clock frequency when = (SYSclk)/12/(65536-[RL_TH1, RL_TL1])/2

If C/T=1, the timer/counter 1 counts the external pulse input (P3.5/T1), then:

the output clock frequency = (T1_Pin_CLK) / (65536-[RL_TH1, RL_TL1])/2

12.2.8 Timer1 mode 1 (16-bit non-autoreloadable mode)

In this mode, Timer/Counter 1 works in 16-bit non-reloadable mode, as shown in the figure below.



Timer/counter 1 mode 1: 16-bit non-reloadable mode

In this mode, Timer/Counter 1 is configured as a 16-bit non-reloadable mode, which is composed of 8 bits of TL1 and 8 bits of TH1. The 8-bit overflow of TL1 carries over to TH1, and the overflow of TH1 counts the overflow flag TF1 in TCON.

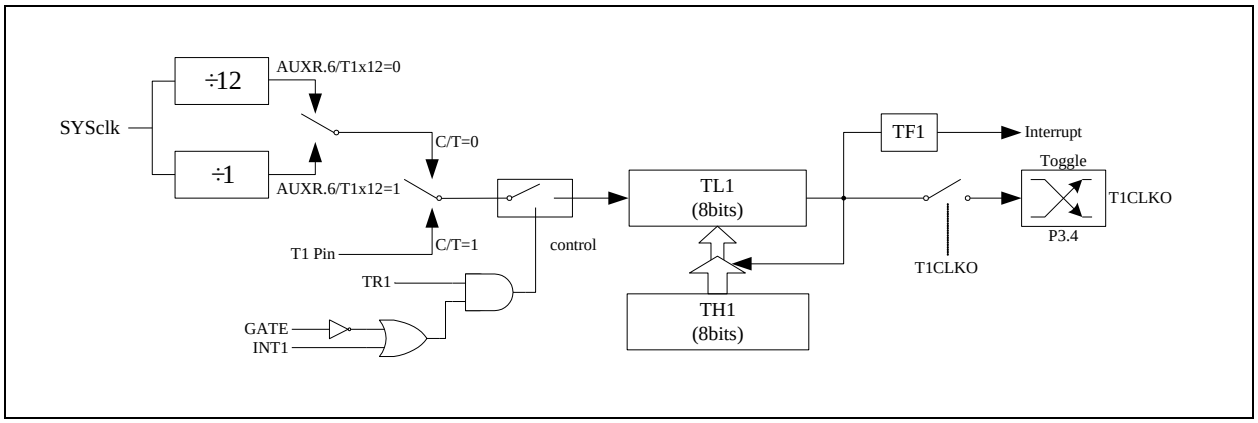
When GATE=0 (TMOD.7), the timer will count if TR1=1. When GATE=1, it is allowed to control timer 1 by external input INT1, so that pulse width measurement can be realized. TR1 is the control bit in the TCON register. For the specific function description of each bit of the TCON register, see the introduction of the TCON register in the previous section.

When C/T=0, the multiplexer is connected to the frequency division output of the system clock, T1 counts the internal system clock, and works in timing mode. When C/T=1, the multiplexer is connected to the external pulse input P3.5/T1, and T1 works in counting mode.

Timer1 of STC microcontroller has two counting rates: one is 12T mode, which is increased by 1 for every 12 clocks, which is the same as traditional 8051 microcontroller, the other is 1T mode, which is increased by 1 for each clock, and the speed is 12 times of traditional 8051. The rate of T1 is determined by T1x12 in the special function register AUXR. If T1x12=0, T1 works in 12T mode, and if T1x12=1, T1 works in 1T mode.

12.2.9 Timer 1 mode 2 (8-bit auto-reloadable mode)

In this mode, Timer/Counter 1 is an 8-bit counter that can be automatically reloaded, as shown in the figure below.



Timer/counter 1 mode 2: 8-bit auto-reloadable mode

The overflow of TL1 not only sets TF1, but also reloads the content of TH1 into TL1. The content of TH1 is preset by software, and its content remains unchanged during reloading.

When T1CLKO/INT_CLKO.1=1, the P3.4/T0 pin is configured as timer 1's clock output T1CLKO. The output clock frequency is T1 overflow rate/2.

If C/T=0, the timer/counter 1 counts the internal system clock, then:

if T1 works in 1T mode (AUXR.6/T1x12=1), the output clock frequency = (SYSclk)/(256-TH1)/2

if T1 works in 12T mode (AUXR.6/T1x12=0), the output clock frequency = (SYSclk)/12/(256-TH1)/2

If C/T=1, the timer/counter T1 counts the external pulse input (P3.5/T1), then:

Output clock frequency = (T1_Pin_CLK) / (256-TH1)/2

12.2.10 Timer 0 Counting Register (TL0, TH0)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TL0	8AH								
TH0	8CH								

When T0 is operating in 16-bit mode (Mode 0, Mode 1, Mode 3), TL0 and TH0 combine into a 16-bit register with TL0 as the low byte and TH0 as the high byte. For 8-bit mode (mode 2), TL0 and TH0 are two independent 8-bit registers.

12.2.11 Timer 1 Counting Register (TL1, TH1)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TL1	8BH								
TH1	8DH								

When T1 is operating in 16-bit mode (Mode 0, Mode 1, Mode 3), TL1 and TH1 combine into a 16-bit register with TL1 as the low byte and TH1 as the high byte. For 8-bit mode (mode 2), TL1 and TH1 are two independent 8-bit registers.

12.2.12 Auxiliary Register 1 (AUXR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

T0x12: T0 speed control bit.

0: The clock source of T0 is SYSclk/12.

1: The clock source of T0 is SYSclk/1.

T1x12: T1 speed control bit.

0: The clock source of T1 is SYSclk/12.

1: The clock source of T1 is SYSclk/1.

12.2.13 External Interrupt and Clock Output Control Register (INTCLKO)

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

T0CLKO: T0 clock out control bit.

0: Turn off the clock output.

1: P3.5 is configured for T0 clock output pin. When T0 overflows, P3.5 will flip automatically.

T1CLKO: T0 clock out control bit.

0: Turn off the clock output.

1: P3.4 is configured for T1 clock output pin. When T1 overflows, P3.4 will flip automatically.

12.2.14 Timer 0 timing calculation formula

mode	speed	period calculation formula
Mode 0/3 (16-bit auto-reloadable mode)	1T	$\text{Period} = \frac{65536 - [TH0, TL0]}{SYSclk}$ (auto-reloadable)
	12T	$\text{Period} = \frac{65536 - [TH0, TL0]}{SYSclk} \times 12$ (auto-reloadable)
Mode 1 (16-bit non-autoreloadable mode)	1T	$\text{Period} = \frac{65536 - [TH0, TL0]}{SYSclk}$ (Software loading required)
	12T	$\text{Period} = \frac{65536 - [TH0, TL0]}{SYSclk} \times 12$ (Software loading required)
Mode 2 (8-bit auto-reloadable mode)	1T	$\text{Period} = \frac{256 - TH0}{SYSclk}$ (auto-reloadable)
	12T	$\text{Period} = \frac{256 - TH0}{SYSclk} \times 12$ (auto-reloadable)

12.2.15 Timer 1 timing calculation formula

mode	speed	period calculation formula
Mode 0 (16-bit auto-reloadable mode)	1T	$\text{Period} = \frac{65536 - [TH1, TL1]}{SYSclk}$ (auto-reloadable)
	12T	$\text{Period} = \frac{65536 - [TH1, TL1]}{SYSclk} \times 12$ (auto-reloadable)

Mode 1 (16-bit non-autoreloadable mode)	1T	$\text{Period} = \frac{65536 - [TH1, TL1]}{SYSclk}$ (Software loading required)
	12T	$\text{Period} = \frac{65536 - [TH1, TL1]}{SYSclk} \times 12$ (Software loading required)
Mode 2 (8-bit auto-reloadable mode)	1T	$\text{Period} = \frac{256 - TH1}{SYSclk}$ (auto-reloadable)
	12T	$\text{Period} = \frac{256 - TH1}{SYSclk} \times 12$ (auto-reloadable)

12.3 Timer 2 (24-bit timer, 8-bit prescaler + 16-bit timing)

12.3.1 Auxiliary Register 1 (AUXR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

T2R: T2 run control bit.

0: T2 stops counting.

1: T2 start counting.

T2_C/T: T2 mode select bit.

0: T2 is used as a timer (input pulse is from internal system clock);

1: T2 is used as a counter (input pulse is from external T2/P1.2 pin).

T2x12: T2 speed control bit.

0: The clock source of T2 is SYSclk/12.

1: The clock source of T2 is SYSclk/1.

12.3.2 External Interrupt and Clock Output Control Register (INTCLKO)

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

T2CLKO: T2 clock out control bit.

0: Turn off the clock output.

1: P1.3 is configured for T2 clock output pin. When T2 overflows, P1.3 will flip automatically.

12.3.3 Timer 2 Counting Register (T2L, T2H)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
T2L	D7H								
T2H	D6H								

T2 operates in 16-bit auto-reload mode. T2L and T2H combine into a 16-bit register with T2L as the low byte and T2H as the high byte. When the 16-bit counter [T2H, T2L] overflows, the system loads the reload value in the internal 16-bit reload register into [T2H, T2L] automatically.

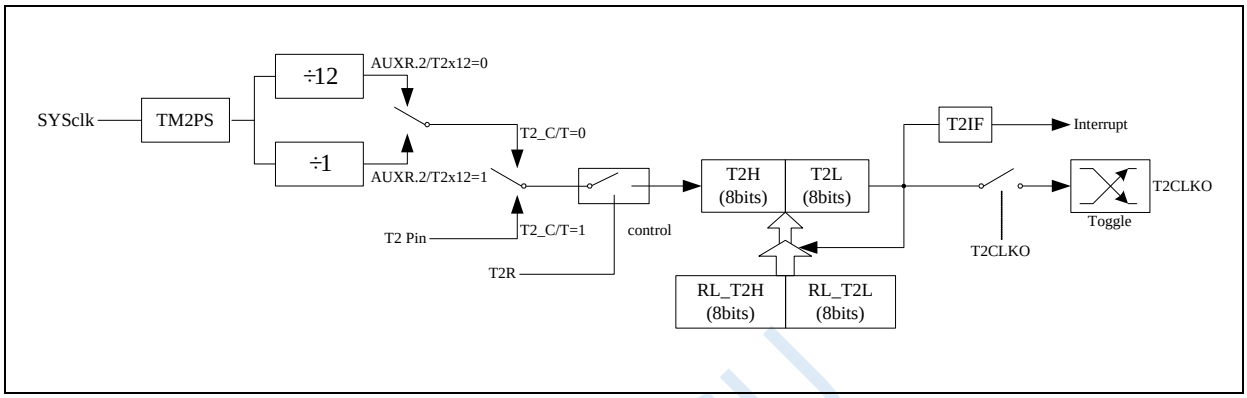
12.3.4 Timer 2 8-bit Prescaler Register (TM2PS)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TM2PS	FEA2H								

Timer 2 clock = SYSclk ÷ (TM2PS + 1)

12.3.5 Timer 2 working mode

The functional block diagram of Timer/Counter 2 is as follows.



Timer/counter 2 working mode: 16-bit auto-reload mode

T2R/AUXR.4 is the control bit in the AUXR register. For the specific function description of each bit of the AUXR register, see the introduction of the AUXR register in the previous section.

When T2_C/T=0, the multiplexer is connected to the frequency division output of the system clock, T2 counts the internal system clock, and works in timing mode. When T2_C/T=1, the multiplexer is connected to the external pulse input T2, and T2 works in counting mode.

Timer2 of STC microcontroller has two counting rates: one is 12T mode, which is increased by 1 for every 12 clocks, which is the same as traditional 8051 microcontroller, the other is 1T mode, which is increased by 1 for each clock, and the speed is 12 times of traditional 8051. The rate of T2 is determined by T2x12 in the special function register AUXR. If T2x12=0, T2 works in 12T mode, and if T2x12=1, T1 works in 1T mode.

Timer2 has two hidden registers RL_T2H and RL_T2L. RL_T2H and T2H share the same address, and RL_T2L and T2L share the same address. When T2R=0, that is, when Timer/Counter2 is disabled, the content written to T2L will be written to RL_T2L at the same time, and the content written to T2H will also be written to RL_T2H at the same time. When T2R=1, that is, when Timer/Counter2 starts to work, writing content to T2L is not actually written to the current register T2L, but written to the hidden register RL_T2L, and writing content to T2H is actually also it is not written into the current register T2H, but into the hidden register RL_T2H, which can cleverly realize the 16-bit reload timer. When reading the contents of T2H and T2L, the contents be read are the contents of T2H and T2L, not the contents of RL_T2H and RL_T2L.

The overflow of [T2H, T2L] not only sets the interrupt request flag (T2IF), which causes the CPU to switch to the timer 2 interrupt routine, but also automatically reloads the contents of [RL_T2H, RL_T2L] into [T2H, T2L].

12.3.6 Timer 2 timing calculation formula

speed	period calculation formula
1T	$\text{Period} = \frac{65536 - [T2H, T2L]}{SYSclk/TM2PS}$ (auto-reloadable)
12T	$\text{Period} = \frac{65536 - [T2H, T2L]}{SYSclk/TM2PS} \times 12$ (auto-reloadable)

12.4 Timer 3/4 (24-bit timer, 8-bit prescaler + 16-bit timing)

12.4.1 Timer4 and Timer 3 Control Register (T4T3M)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

T4R: T4 run control bit.

0: T4 stops counting.

1: T4 start counting.

T4_C/T: T4 mode select bit.

0: T4 is used as a timer (input pulse is from internal system clock);

1: T4 is used as a counter (input pulse is from external T4/P0.6 pin).

T4x12: T4 speed control bit.

0: The clock source of T4 is SYSclk/12.

1: The clock source of T4 is SYSclk/1.

T4CLKO: T4 clock out control bit.

0: Turn off the clock output.

1: P0.7 is configured for T4 clock output pin. When T4 overflows, P0.7 will flip automatically.

T3R: T3 run control bit.

0: T3 stops counting.

1: T3 start counting.

T3_C/T: T3 mode select bit.

0: T3 is used as a timer (input pulse is from internal system clock);

1: T3 is used as a counter (input pulse is from external T3/P0.4 pin).

T3x12: T3 speed control bit.

0: The clock source of T3 is SYSclk/12.

1: The clock source of T3 is SYSclk/1.

T3CLKO: T3 clock out control bit.

0: Turn off the clock output.

1: P0.5 is configured for T3 clock output pin. When T3 overflows, P0.5 will flip automatically.

12.4.2 Timer 3 Counting Register (T3L, T3H)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
T3L	D5H								
T3H	D4H								

T3 operates in 16-bit auto-reload mode. T3L and T3H combine into a 16-bit register with T3L as the low byte and T3H as the high byte. When the 16-bit counter [T3H, T3L] overflows, the system loads the reload value in the internal 16-bit reload register into [T3H, T3L] automatically.

12.4.3 Timer 4 Counting Register (T4L, T4H)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
T4L	D3H								
T4H	D2H								

T4 operates in 16-bit auto-reload mode. T4L and T4H combine into a 16-bit register with T4L as the low byte and T4H as the high byte. When the 16-bit counter [T4H, T4L] overflows, the system loads the reload value in the internal 16-bit reload register into [T4H, T4L] automatically.

12.4.4 Timer 3 8-bit Prescaler Register (TM3PS)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TM3PS	FEA3H								

Timer 3 clock = SYSclk ÷ (TM3PS + 1)

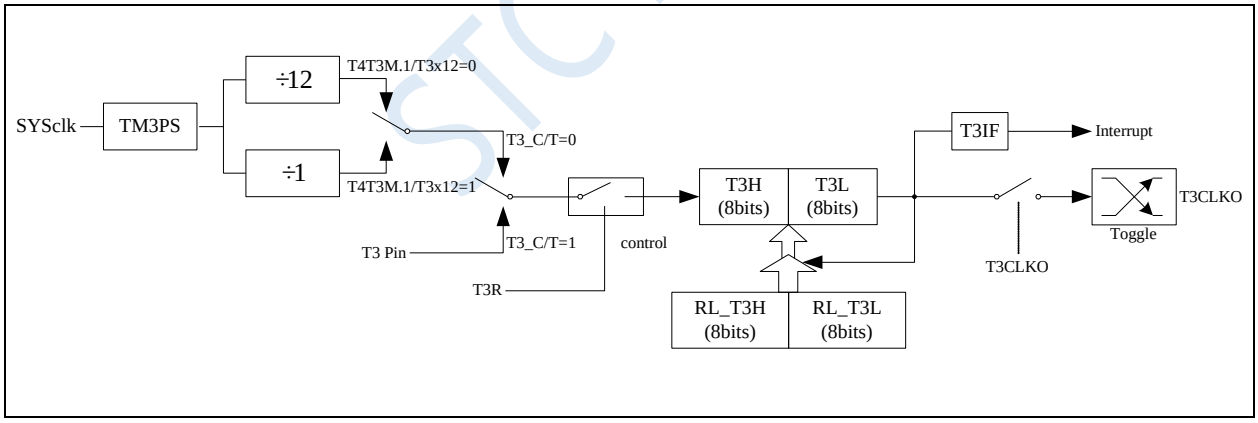
12.4.5 Timer 4 8-bit Prescaler Register (TM4PS)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TM4PS	FEA4H								

Timer 4 clock = SYSclk ÷ (TM4PS + 1)

12.4.6 Timer 3 working mode

The functional block diagram of Timer/Counter 3 is as follows.



Timer/counter 3 working mode: 16-bit auto-reload mode

T3R/T4T3M.3 is the control bit in the T4T3M register. For the specific function description of each bit of the T4T3M register, see the introduction of the T4T3M register in the previous section.

When T3_C/T=0, the multiplexer is connected to the frequency division output of the system clock, T3 counts the internal system clock, and works in timing mode. When T3_C/T=1, the multiplexer is connected to the external pulse input T3, and T3 works in counting mode.

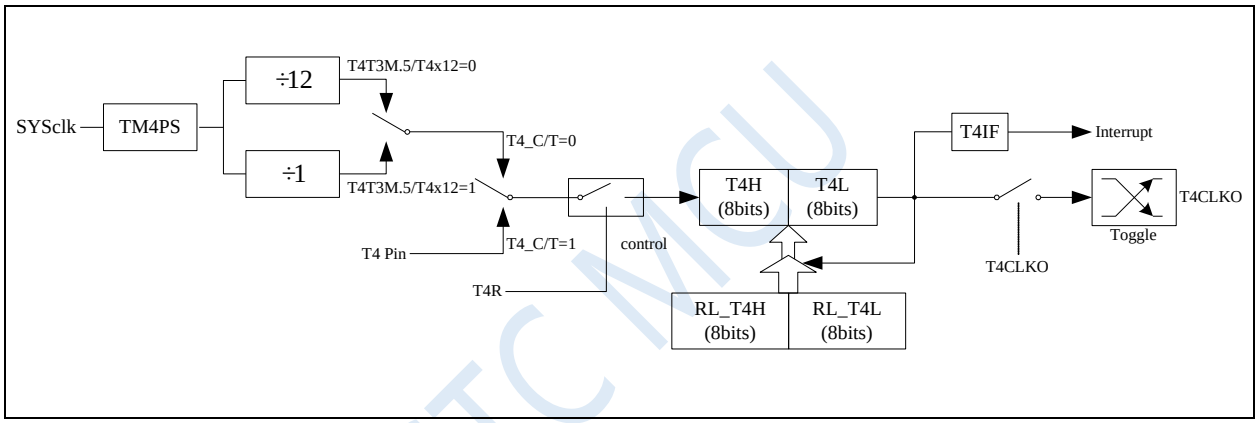
Timer3 of STC microcontroller has two counting rates: one is 12T mode, which is increased by 1 for every 12 clocks, which is the same as traditional 8051 microcontroller, the other is 1T mode, which is increased by 1 for each clock, and the speed is 12 times of traditional 8051. The rate of T3 is determined by T3x12 in the special function register T4T3M. If T3x12=0, T3 works in 12T mode, and if T3x12=1, T3 works in 1T mode.

Timer3 has two hidden registers RL_T3H and RL_T3L. RL_T3H and T3H share the same address, and RL_T3L and T3L share the same address. When T3R=0, that is, when Timer/Counter3 is disabled, the content written to T3L will be written to RL_T3L at the same time, and the content written to T3H will also be written to RL_T3H at the same time. When T3R=1, that is, when Timer/Counter3 starts to work, writing content to T3L is not actually written to the current register T3L, but written to the hidden register RL_T3L, and writing content to T3H is actually also it is not written into the current register T3H, but into the hidden register RL_T3H, which can cleverly realize the 16-bit reload timer. When reading the contents of T3H and T3L, the contents be read are the contents of T3H and T3L, not the contents of RL_T3H and RL_T3L.

The overflow of [T3H, T3L] not only sets the interrupt request flag (T3IF), which causes the CPU to switch to the timer 3 interrupt routine, but also automatically reloads the contents of [RL_T3H, RL_T3L] into [T3H, T3L].

12.4.7 Timer 4 working mode

The functional block diagram of Timer/Counter 4 is as follows.



Timer/counter 4 working mode: 16-bit auto-reload mode

T4R/T4T3M.7 is the control bit in the T4T3M register. For the specific function description of each bit of the T4T3M register, see the introduction of the T4T3M register in the previous section.

When T4_C/T=0, the multiplexer is connected to the frequency division output of the system clock, T4 counts the internal system clock, and works in timing mode. When T4_C/T=1, the multiplexer is connected to the external pulse input T4, and T4 works in counting mode.

Timer4 of STC microcontroller has two counting rates: one is 12T mode, which is increased by 1 for every 12 clocks, which is the same as traditional 8051 microcontroller, the other is 1T mode, which is increased by 1 for each clock, and the speed is 12 times of traditional 8051. The rate of T4 is determined by T4x12 in the special function register T4T3M. If T4x12=0, T4 works in 12T mode, and if T4x12=1, T4 works in 1T mode.

Timer4 has two hidden registers RL_T4H and RL_T4L. RL_T4H and T4H share the same address, and RL_T4L and T4L share the same address. When T4R=0, that is, when Timer/Counter4 is disabled, the content written to T4L will be written to RL_T4L at the same time, and the content written to T4H will also be written to RL_T4H at the same time. When T4R=1, that is, when Timer/Counter4 starts to work, writing content to T4L is not actually written to the current register T4L, but written to the hidden register RL_T4L, and writing content to T4H is actually also it is not written into the current register T4H, but into the hidden register RL_T4H, which can cleverly realize the 16-bit reload timer. When reading the contents of T4H and T4L, the

contents be read are the contents of T4H and T4L, not the contents of RL_T4H and RL_T4L.

The overflow of [T4H, T4L] not only sets the interrupt request flag (T4IF), which causes the CPU to switch to the timer 4 interrupt routine, but also automatically reloads the contents of [RL_T4H, RL_T4L] into [T4H, T4L].

12.4.8 Timer 3 timing calculation formula

speed	period calculation formula
1T	$\text{Period} = \frac{65536 - [T3H, T3L]}{SYSclk/TM3PS}$ (auto-reloadable)
12T	$\text{Period} = \frac{65536 - [T3H, T3L]}{SYSclk/TM3PS} \times 12$ (auto-reloadable)

12.4.9 Timer 4 timing calculation formula

speed	period calculation formula
1T	$\text{Period} = \frac{65536 - [T4H, T4L]}{SYSclk/TM4PS}$ (auto-reloadable)
12T	$\text{Period} = \frac{65536 - [T4H, T4L]}{SYSclk/TM4PS} \times 12$ (auto-reloadable)

STC MCU

12.5 Example Routines

12.5.1 Timer 0 (Mode 0 - 16-bit auto reload)

C language code

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

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

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

sbit    P10     =   P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;                             //Test port
}

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

    TMOD = 0x00;                            //Mode 0
    TL0 = 0x66;                             //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                //Start timer
    ET0 = 1;                                //Enable timer interrupt
    EA = 1;

    while (1);
}
```

STC MCU

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            ORG     000BH
            LJMP    TM0ISR

            ORG     0100H
TM0ISR:
            CPL     P1.0            ;Test port
            RETI

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

            MOV     TMOD,#00H       ;Mode 0
            MOV     TL0,#66H        ;65536-11.0592M/12/1000
            MOV     TH0,#0FCH
            SETB    TR0             ;Start timer
            SETB    ET0             ;Enable timer interrupt
            SETB    EA

            JMP     $

            END
```

12.5.2 Timer 0 (Mode 1 - 16-bit non-auto reloadable)

C language code

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

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

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

sbit    P10     =   P1^0;

void TM0_Isr() interrupt 1
{
    TL0 = 0x66;                 //Reset parameters
    TH0 = 0xfc;
    P10 = !P10;                 //Test port
}

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

    TMOD = 0x01;                //Mode 1
    TL0 = 0x66;                 //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                    //Start timer
    ET0 = 1;                    //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P0M1        DATA        093H
```

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

            ORG     0000H
            LJMP    MAIN
            ORG     000BH
            LJMP    TM0ISR

            ORG     0100H
TM0ISR:
            MOV     TL0,#66H            ;Reset parameters
            MOV     TH0,#0FCH
            CPL     P1.0                ;Test port
            RETI

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

            MOV     TMOD,#01H           ;Mode 1
            MOV     TL0,#66H            ;65536-11.0592M/12/1000
            MOV     TH0,#0FCH
            SETB    TR0                 ;Start timer
            SETB    ET0                 ;Enable timer interrupt
            SETB    EA

            JMP     $

            END
```

12.5.3 Timer 0 (Mode 2 - 8-bit auto-reloadable)

C language code

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

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

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

sbit    P10     =   P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;                                 //Test port
}

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

    TMOD = 0x02;                                //Mode 2
    TL0 = 0xf4;                                 //256-11.0592M/12/76K
    TH0 = 0xf4;
    TR0 = 1;                                    //Start timer
    ET0 = 1;                                    //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
```

```
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     000BH
            LJMP    TM0ISR

            ORG     0100H
TM0ISR:
            CPL     P1.0                ;Test port
            RETI

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

            MOV     TMOD,#02H           ;Mode 2
            MOV     TL0,#0F4H           ;256-11.0592M/12/76K
            MOV     TH0,#0F4H
            SETB    TR0                 ;Start timer
            SETB    ET0                 ;Enable timer interrupt
            SETB    EA

            JMP     $

            END
```

12.5.4 Timer 0 (Mode 3 - 16-bit auto-reloadable with non-maskable interrupt)

C language code

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

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

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

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

sbit    P10     =   P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;                             //Test port
}

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

    TMOD = 0x03;                            //Mode 3
    TL0 = 0x66;                             //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                //Start timer
    ET0 = 1;                                //Enable timer interrupt
//  EA = 1;                                 // Not under EA control

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

STC MCU

```
                ORG     0000H
                LJMP    MAIN
                ORG     000BH
                LJMP    TM0ISR

                ORG     0100H
TM0ISR:
                CPL     P1.0                ;Test port
                RETI

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

                MOV     TMOD,#03H           ;Mode 3
                MOV     TL0,#66H            ;65536-11.0592M/12/1000
                MOV     TH0,#0FCH
                SETB    TR0                 ;Start timer
                SETB    ET0                 ;Enable timer interrupt
;               SETB    EA                  ; Not under EA control

                JMP     $

                END
```

12.5.5 Timer 0 (External count - T0 is extended for external falling edge interrupt)

C language code

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

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

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
```

```
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;                                 //Test port
}

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

    TMOD = 0x04;                                //External counting mode
    TL0 = 0xff;
    TH0 = 0xff;
    TR0 = 1;                                    //Start timer
    ET0 = 1;                                    //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

        ORG     0000H
        LJMP    MAIN
        ORG     000BH
        LJMP    TM0ISR
```

```
            ORG     0100H
TM0ISR:
            CPL     P1.0            ;Test port
            RETI

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

            MOV     TMOD,#04H       ;External counting mode
            MOV     TL0,#0FFH
            MOV     TH0,#0FFH
            SETB    TR0             ;Start timer
            SETB    ET0             ;Enable timer interrupt
            SETB    EA

            JMP     $

            END
```

12.5.6 Timer 0 (Pulse width measurement for high-level width of INT0)

C language code

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

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

sfr     AUXR    =   0x8e;

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

```
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void INT0_Isr() interrupt 0
{
    P0 = TL0;  //TL0 is the low byte of the measured value
    P1 = TH0; //TH0 is the high byte of the measured value
}

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

    AUXR = 0x80;                    //1T mode
    TMOD = 0x08;                    //Enable GATE, and enable timing when INT0 is 1
    TL0 = 0x00;
    TH0 = 0x00;
    while (INT0);                   //Wait for INT0 to be low
    TR0 = 1;                        //Start timer
    IT0 = 1;                        //Enable INT0 falling edge interrupt
    EX0 = 1;
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR    DATA    8EH
P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
        ORG     0003H
        LJMP    INT0ISR
```

```
            ORG     0100H
INT0ISR:
            MOV     P0,TL0          ;TL0 is the low byte of the measured value
            MOV     P1,TH0          ;TH0 is the high byte of the measured value
            RETI

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

            MOV     AUXR,#80H       ;1T mode
            MOV     TMOD,#08H       ;Enable GATE, and enable timing when INT0 is 1
            MOV     TL0,#00H
            MOV     TH0,#00H
            JB      INT0,$          ;Wait for INT0 to be low
            SETB    TR0             ;Start timer
            SETB    IT0             ;Enable INT0 falling edge interrupt
            SETB    EX0
            SETB    EA

            JMP     $

            END
```

12.5.7 Timer 0 (Divided clock output)

C language code

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

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

sfr     INTCLKO     =   0x8f;

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

```
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

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

    TMOD = 0x00;                    //Mode 0
    TL0 = 0x66;                     //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                        //Start timer
    INTCLKO = 0x01;                 //Enable clock output

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

INTCLKO     DATA    8FH
P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
```

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

        MOV     TMOD,#00H               ;Mode 0
        MOV     TL0,#66H                ;65536-11.0592M/12/1000
        MOV     TH0,#0FCH
        SETB    TR0                     ;Start timer
        MOV     INTCLKO,#01H            ;Enable clock output

        JMP     $

        END
```

12.5.8 Timer 1 (Mode 0 - 16-bit auto-reloadable)

C language code

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

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

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

sbit    P10     =   P1^0;

void TM1_Isr() interrupt 3
{
    P10 = !P10;                         //Test port
}

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

STC MCU

```
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;                    //Mode 0
    TL1 = 0x66;                     //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                        //Start timer
    ET1 = 1;                        //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            ORG     001BH
            LJMP    TM1ISR

            ORG     0100H
TM1ISR:
            CPL     P1.0            ;Test port
            RETI

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

            MOV     TMOD,#00H       ;Mode 0
            MOV     TL1,#66H        ;65536-11.0592M/12/1000
            MOV     TH1,#0FCH
```

STC MCU

```
        SETB    TR1                     ;Start timer
        SETB    ET1                     ;Enable timer interrupt
        SETB    EA

        JMP     $

        END
```

12.5.9 Timer 1 (Mode 1 - 16-bit non-auto reloadable)

C language code

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

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

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

sbit    P10     =   P1^0;

void TM1_Isr() interrupt 3
{
    TL1 = 0x66;                         //Reset parameters
    TH1 = 0xfc;
    P10 = !P10;                         //Test port
}

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

    TMOD = 0x10;                        //Mode 1
    TL1 = 0x66;                         //65536-11.0592M/12/1000
```

```
    TH1 = 0xfc;
    TR1 = 1;                                //Start timer
    ET1 = 1;                                //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            ORG     001BH
            LJMP    TM1ISR

            ORG     0100H
TM1ISR:
            MOV     TL1,#66H            ;Reset parameters
            MOV     TH1,#0FCH
            CPL     P1.0                ;Test port
            RETI

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

            MOV     TMOD,#10H           ;Mode 1
            MOV     TL1,#66H            ;65536-11.0592M/12/1000
            MOV     TH1,#0FCH
            SETB    TR1                 ;Start timer
            SETB    ET1                 ;Enable timer interrupt
            SETB    EA
```

STC MCU

```
        JMP     $

        END
```

12.5.10 Timer 1 (Mode 2 - 8-bit auto-reloadable)

C language code

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

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

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

sbit    P10     =   P1^0;

void TM1_Isr() interrupt 3
{
    P10 = !P10;                         //Test port
}

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

    TMOD = 0x20;                        //Mode 2
    TL1 = 0xf4;                         //256-11.0592M/12/76K
    TH1 = 0xf4;
    TR1 = 1;                            //Start timer
    ET1 = 1;                            //Enable timer interrupt
    EA = 1;

    while (1);
```

STC MCU

```
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            ORG     001BH
            LJMP    TM1ISR

            ORG     0100H
TM1ISR:
            CPL     P1.0            ;Test port
            RETI

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

            MOV     TMOD,#20H       ;Mode 2
            MOV     TL1,#0F4H       ;256-11.0592M/12/76K
            MOV     TH1,#0F4H
            SETB    TR1             ;Start timer
            SETB    ET1             ;Enable timer interrupt
            SETB    EA

            JMP     $

            END
```

STC MCU

12.5.11 Timer 1 (External count – T1 is extended for external falling edge interrupt)

C language code

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

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

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

sbit    P10     =   P1^0;

void TM1_Isr() interrupt 3
{
    P10 = !P10;                             //Test port
}

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

    TMOD = 0x40;                            //External counting mode
    TL1 = 0xff;
    TH1 = 0xff;
    TR1 = 1;                                //Start timer
    ET1 = 1;                                //Enable timer interrupt
    EA = 1;

    while (1);
}
```

STC MCU

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            ORG     001BH
            LJMP    TM1ISR

            ORG     0100H
TM1ISR:
            CPL     P1.0            ;Test port
            RETI

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

            MOV     TMOD,#40H       ;External counting mode
            MOV     TL1,#0FFH
            MOV     TH1,#0FFH
            SETB    TR1             ;Start timer
            SETB    ET1             ;Enable timer interrupt
            SETB    EA

            JMP     $

            END
```

STC MCU

12.5.12 Timer 1 (Pulse width measurement for high-level width of INT1)

C language code

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

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

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

sfr     AUXR    =   0x8e;

void INT1_Isr() interrupt 2
{
    P0 = TL1;  //TL1 is the low byte of the measured value
    P1 = TH1;  //TH1 is the high byte of the measured value
}

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

    AUXR = 0x40;                    //1T mode
    TMOD = 0x80;                    //Enable GATE, and enable timing when INT1 is 1
    TL1 = 0x00;
    TH1 = 0x00;
    while (INT1);                   //Wait for INT1 to be low
    TR1 = 1;                        //Start timer
    IT1 = 1;                        //Enable INT1 falling edge interrupt
    EX1 = 1;
    EA = 1;
```

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

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR    DATA    8EH
P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
        ORG     0013H
        LJMP    INT1ISR

        ORG     0100H
INT1ISR:
        MOV     P0,TL1          ;TL1 is the low byte of the measured value
        MOV     P1,TH1          ;TH1 is the high byte of the measured value
        RETI

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

        MOV     AUXR,#40H       ;1T mode
        MOV     TMOD,#80H       ;Enable GATE, and enable timing when INT1 is 1
        MOV     TL1,#00H
        MOV     TH1,#00H
        JB      INT1,$          ;Wait for INT1 to be low
        SETB    TR1             ;Start timer
        SETB    IT1             ;Enable INT1 falling edge interrupt
        SETB    EX1
        SETB    EA

        JMP     $
```

STC MCU

```
END
```

12.5.13 Timer 1 (mode 0, Divided clock output)

C language code

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

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

sfr     INTCLKO     =   0x8f;

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

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

    TMOD = 0x00;                    //Mode 0
    TL1 = 0x66;                     //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                        //Start timer
    INTCLKO = 0x02;                 //Enable clock output

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

INTCLKO     DATA    8FH
P0M1        DATA    093H
```

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

            ORG     0000H
            LJMP    MAIN

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

            MOV     TMOD,#00H       ;Mode 0
            MOV     TL1,#66H        ;65536-11.0592M/12/1000
            MOV     TH1,#0FCH
            SETB    TR1             ;Start timer
            MOV     INTCLKO,#02H    ;Enable clock output

            JMP     $

            END
```

12.5.14 Timer 1 (Mode 0) is used as baud rate generator of UART1

C language code

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

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

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

sfr      AUXR       =   0x8e;

sfr      P0M1       =   0x93;
```

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

bit     busy;
char    wptr;
char    rptr;
char    buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSEND(*p++);
    }
}
```

STC MCU

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

    UartInit();
    ES = 1;
    EA = 1;
    UartSENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

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

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR
```

```
                ORG     0100H

UART_ISR:
                PUSH    ACC
                PUSH    PSW
                MOV     PSW,#08H

                JNB     TI,CHKRI
                CLR     TI
                CLR     BUSY
CHKRI:
                JNB     RI,UARTISR_EXIT
                CLR     RI
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,SBUF
                INC     WPTR
UARTISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

UART_INIT:
                MOV     SCON,#50H
                MOV     TMOD,#00H
                MOV     TL1,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     TH1,#0FFH
                SETB    TR1
                MOV     AUXR,#40H
                CLR     BUSY
                MOV     WPTR,#00H
                MOV     RPTR,#00H
                RET

UART_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     SBUF,A
                RET

UART_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SENDEND
                LCALL   UART_SEND
                INC     DPTR
                JMP     UART_SENDSTR
SENDEND:
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
```

STC MCU

```
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART_SEND
                INC     RPTR
                JMP     LOOP

STRING:         DB      'Uart Test !',0DH,0AH,00H

                END
```

12.5.15 Timer 1 (Mode 2) is used as baud rate generator of UART1

C language code

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

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

#define   FOSC        11059200UL
#define   BRT         (256 - FOSC / 115200 / 32)

sfr       AUXR        =   0x8e;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
```

```
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

bit     busy;
char    wptr;
char    rptr;
char    buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x20;
    TL1 = BRT;
    TH1 = BRT;
    TR1 = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSEND(*p++);
    }
}

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

STC MCU

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

    UartInit();
    ES = 1;
    EA = 1;
    UartSENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H             ;16 bytes

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

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H
```

```
            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,SBUF
            INC     WPTR
UARTISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART_INIT:
            MOV     SCON,#50H
            MOV     TMOD,#20H
            MOV     TL1,#0FDH           ;256-11059200/115200/32=0FDH
            MOV     TH1,#0FDH
            SETB    TR1
            MOV     AUXR,#40H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART_SEND:
            JB      BUSY,$
            SETB    BUSY
            MOV     SBUF,A
            RET

UART_SENDSTR:
            CLR     A
            MOVC    A,@A+DPTR
            JZ      SENDEND
            LCALL   UART_SEND
            INC     DPTR
            JMP     UART_SENDSTR
SENDEND:
            RET

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

STC MCU

```
                MOV         P5M1, #00H

                LCALL       UART_INIT
                SETB        ES
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

12.5.16 Timer 2 (16-bit auto-reloadable)

C language code

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

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

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2         0x04
sfr     AUXINTIF    =   0xef;
#define T2IF        0x01

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

```
sbit    P10     =   P1^0;

void TM2_Isr() interrupt 12
{
    P10 = !P10;                                 //Test port
}

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

    T2L = 0x66;                                 //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;                                //Start timer
    IE2 = ET2;                                  //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
IE2         DATA    0AFH
ET2         EQU     04H
AUXINTIF    DATA    0EFH
T2IF        EQU     01H

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

            ORG     0000H
            LJMP    MAIN
```

```
            ORG     0063H
            LJMP    TM2ISR

            ORG     0100H
TM2ISR:
            CPL     P1.0                ;Test port
            RETI

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

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

            JMP     $

            END
```

12.5.17 Timer 2 (External count – T2 is extended for external falling edge interrupt)

C language code

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

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

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2         0x04
sfr     AUXINTIF=   0xef;
#define T2IF        0x01

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

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

sbit    P10     =   P1^0;

void TM2_Isr() interrupt 12
{
    P10 = !P10;                                 //Test port
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T2L = 0xff;
    T2H = 0xff;
    AUXR = 0x18;                                //Set external counting mode and start timer
    IE2 = ET2;                                  //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
IE2         DATA    0AFH
ET2         EQU     04H
AUXINTIF    DATA    0EFH
T2IF        EQU     01H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
```

```
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0063H
            LJMP    TM2ISR

            ORG     0100H
TM2ISR:
            CPL     P1.0                ;Test port
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     T2L,#0FFH
            MOV     T2H,#0FFH
            MOV     AUXR,#18H           ;Set external counting mode and start timer
            MOV     IE2,#ET2            ;Enable timer interrupt
            SETB    EA

            JMP     $

            END
```

12.5.18 Timer 2 (Divided clock output)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     INTCLKO =   0x8f;
```

```
sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T2L = 0x66;                 //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;                //Start timer
    INTCLKO = 0x04;             //Enable clock output

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
INTCLKO     DATA    8FH

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
```

```
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     T2L,#66H            ;65536-11.0592M/12/1000
            MOV     T2H,#0FCH
            MOV     AUXR,#10H           ;Start timer
            MOV     INTCLKO,#04H        ;Enable clock output

            JMP     $

            END
```

12.5.19 Timer 2 is used as baud rate generator of UART1

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC      11059200UL
#define   BRT       (65536 - FOSC / 115200 / 4)

sfr       AUXR      =   0x8e;
sfr       T2H       =   0xd6;
sfr       T2L       =   0xd7;

sfr       P0M1      =   0x93;
sfr       P0M0      =   0x94;
sfr       P1M1      =   0x91;
sfr       P1M0      =   0x92;
sfr       P2M1      =   0x95;
sfr       P2M0      =   0x96;
sfr       P3M1      =   0xb1;
sfr       P3M0      =   0xb2;
sfr       P4M1      =   0xb3;
sfr       P4M0      =   0xb4;
sfr       P5M1      =   0xc9;
sfr       P5M0      =   0xca;
```

```
bit		busy;
char		wptr;
char		rptr;
char		buffer[16];

void UartIsr() interrupt 4
{
	if (TI)
	{
		TI = 0;
		busy = 0;
	}
	if (RI)
	{
		RI = 0;
		buffer[wptr++] = SBUF;
		wptr &= 0x0f;
	}
}

void UartInit()
{
	SCON = 0x50;
	T2L = BRT;
	T2H = BRT >> 8;
	AUXR = 0x15;
	wptr = 0x00;
	rptr = 0x00;
	busy = 0;
}

void UartSend(char dat)
{
	while (busy);
	busy = 1;
	SBUF = dat;
}

void UartSendStr(char *p)
{
	while (*p)
	{
		UartSEND(*p++);
	}
}

void main()
{
	P0M0 = 0x00;
	P0M1 = 0x00;
	P1M0 = 0x00;
	P1M1 = 0x00;
	P2M0 = 0x00;
	P2M1 = 0x00;
	P3M0 = 0x00;
	P3M1 = 0x00;
	P4M0 = 0x00;
	P4M1 = 0x00;
	P5M0 = 0x00;
```

STC MCU

```
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
```

```
                JNB     RI,UARTISR_EXIT
                CLR     RI
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,SBUF
                INC     WPTR
UARTISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

UART_INIT:
                MOV     SCON,#50H
                MOV     T2L,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     T2H,#0FFH
                MOV     AUXR,#15H
                CLR     BUSY
                MOV     WPTR,#00H
                MOV     RPTR,#00H
                RET

UART_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     SBUF,A
                RET

UART_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SENDEND
                LCALL   UART_SEND
                INC     DPTR
                JMP     UART_SENDSTR
SENDEND:
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#STRING
```

STC MCU

```
            LCALL   UART_SENDSTR

LOOP:
            MOV     A,RPTR
            XRL     A,WPTR
            ANL     A,#0FH
            JZ      LOOP
            MOV     A,RPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     A,@R0
            LCALL   UART_SEND
            INC     RPTR
            JMP     LOOP

STRING:     DB      'Uart Test !',0DH,0AH,00H

            END
```

12.5.20 Timer 2 is used as baud rate generator of UART2

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC      11059200UL
#define   BRT       (65536 - FOSC / 115200 / 4)

sfr       AUXR      =   0x8e;
sfr       T2H       =   0xd6;
sfr       T2L       =   0xd7;
sfr       S2CON     =   0x9a;
sfr       S2BUF     =   0x9b;
sfr       IE2       =   0xaf;

sfr       P0M1      =   0x93;
sfr       P0M0      =   0x94;
sfr       P1M1      =   0x91;
sfr       P1M0      =   0x92;
sfr       P2M1      =   0x95;
sfr       P2M0      =   0x96;
sfr       P3M1      =   0xb1;
sfr       P3M0      =   0xb2;
sfr       P4M1      =   0xb3;
sfr       P4M0      =   0xb4;
sfr       P5M1      =   0xc9;
sfr       P5M0      =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];
```

```
void Uart2Isr() interrupt 8
{
    if (S2CON & 0x02)
    {
        S2CON &= ~0x02;
        busy = 0;
    }
    if (S2CON & 0x01)
    {
        S2CON &= ~0x01;
        buffer[wptr++] = S2BUF;
        wptr &= 0x0f;
    }
}

void Uart2Init()
{
    S2CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart2Send(char dat)
{
    while (busy);
    busy = 1;
    S2BUF = dat;
}

void Uart2SendStr(char *p)
{
    while (*p)
    {
        Uart2SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart2Init();
    IE2 = 0x01;
```

STC MCU

```
    EA = 1;
    Uart2SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart2SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H
S2CON       DATA    9AH
S2BUF       DATA    9BH
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0043H
            LJMP    UART2_ISR

            ORG     0100H

UART2_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S2CON
            JNB     ACC.1,CHKRI
            ANL     S2CON,#NOT 02H
            CLR     BUSY
CHKRI:
```

STC MCU

```
                JNB     ACC.0,UART2ISR_EXIT
                ANL     S2CON,#NOT 01H
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,S2BUF
                INC     WPTR
UART2ISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

UART2_INIT:
                MOV     S2CON,#10H
                MOV     T2L,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     T2H,#0FFH
                MOV     AUXR,#14H
                CLR     BUSY
                MOV     WPTR,#00H
                MOV     RPTR,#00H
                RET

UART2_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     S2BUF,A
                RET

UART2_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SEND2END
                LCALL   UART2_SEND
                INC     DPTR
                JMP     UART2_SENDSTR
SEND2END:
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART2_INIT
                MOV     IE2,#01H
                SETB    EA

                MOV     DPTR,#STRING
```

STC MCU

```
                LCALL   UART2_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART2_SEND
                INC     RPTR
                JMP     LOOP

STRING:         DB      'Uart Test !',0DH,0AH,00H

                END
```

12.5.21 Timer 2 is used as baud rate generator of UART3

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       AUXR        =   0x8e;
sfr       T2H         =   0xd6;
sfr       T2L         =   0xd7;
sfr       S3CON       =   0xac;
sfr       S3BUF       =   0xad;
sfr       IE2         =   0xaf;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];
```

```
void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}

void Uart3Init()
{
    S3CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart3Init();
    IE2 = 0x08;
```

STC MCU

```
    EA = 1;
    Uart3SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H
S3CON       DATA    0ACH
S3BUF       DATA    0ADH
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     008BH
            LJMP    UART3_ISR

            ORG     0100H

UART3_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S3CON
            JNB     ACC.1,CHKRI
            ANL     S3CON,#NOT 02H
            CLR     BUSY
CHKRI:
```

STC MCU

```
                JNB     ACC.0,UART3ISR_EXIT
                ANL     S3CON,#NOT 01H
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,S3BUF
                INC     WPTR
UART3ISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

UART3_INIT:
                MOV     S3CON,#10H
                MOV     T2L,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     T2H,#0FFH
                MOV     AUXR,#14H
                CLR     BUSY
                MOV     WPTR,#00H
                MOV     RPTR,#00H
                RET

UART3_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     S3BUF,A
                RET

UART3_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SEND3END
                LCALL   UART3_SEND
                INC     DPTR
                JMP     UART3_SENDSTR
SEND3END:
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART3_INIT
                MOV     IE2,#08H
                SETB    EA

                MOV     DPTR,#STRING
```

STC MCU

```
            LCALL   UART3_SENDSTR

LOOP:
            MOV     A,RPTR
            XRL     A,WPTR
            ANL     A,#0FH
            JZ      LOOP
            MOV     A,RPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     A,@R0
            LCALL   UART3_SEND
            INC     RPTR
            JMP     LOOP

STRING:     DB      'Uart Test !',0DH,0AH,00H

            END
```

12.5.22 Timer 2 is used as baud rate generator of UART4

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC      11059200UL
#define   BRT       (65536 - FOSC / 115200 / 4)

sfr       AUXR      =   0x8e;
sfr       T2H       =   0xd6;
sfr       T2L       =   0xd7;
sfr       S4CON     =   0x84;
sfr       S4BUF     =   0x85;
sfr       IE2       =   0xaf;

sfr       P0M1      =   0x93;
sfr       P0M0      =   0x94;
sfr       P1M1      =   0x91;
sfr       P1M0      =   0x92;
sfr       P2M1      =   0x95;
sfr       P2M0      =   0x96;
sfr       P3M1      =   0xb1;
sfr       P3M0      =   0xb2;
sfr       P4M1      =   0xb3;
sfr       P4M0      =   0xb4;
sfr       P5M1      =   0xc9;
sfr       P5M0      =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];
```

```
void Uart4Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}

void Uart4Init()
{
    S4CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart4Send(char dat)
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart4Init();
    IE2 = 0x10;
```

STC MCU

```
    EA = 1;
    Uart4SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H
S4CON       DATA    84H
S4BUF       DATA    85H
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0093H
            LJMP    UART4_ISR

            ORG     0100H

UART4_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S4CON
            JNB     ACC.1,CHKRI
            ANL     S4CON,#NOT 02H
            CLR     BUSY
CHKRI:
```

STC MCU

```
            JNB     ACC.0,UART4ISR_EXIT
            ANL     S4CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S4BUF
            INC     WPTR
UART4ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART4_INIT:
            MOV     S4CON,#10H
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART4_SEND:
            JB      BUSY,$
            SETB    BUSY
            MOV     S4BUF,A
            RET

UART4_SENDSTR:
            CLR     A
            MOVC    A,@A+DPTR
            JZ      SEND4END
            LCALL   UART4_SEND
            INC     DPTR
            JMP     UART4_SENDSTR
SEND4END:
            RET

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            LCALL   UART4_INIT
            MOV     IE2,#10H
            SETB    EA

            MOV     DPTR,#STRING
```

```
            LCALL   UART4_SENDSTR

LOOP:
            MOV     A,RPTR
            XRL     A,WPTR
            ANL     A,#0FH
            JZ      LOOP
            MOV     A,RPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     A,@R0
            LCALL   UART4_SEND
            INC     RPTR
            JMP     LOOP

STRING:     DB      'Uart Test !',0DH,0AH,00H

            END
```

12.5.23 Timer 3 (16-bit auto-reloadable)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T4T3M   =   0xd1;
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2     0x04
#define ET3     0x20
#define ET4     0x40
sfr     AUXINTIF=   0xef;
#define T2IF    0x01
#define T3IF    0x02
#define T4IF    0x04

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
```

```
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM3_Isr() interrupt 19
{
    P10 = !P10;                 //Test port
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T3L = 0x66;                 //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x08;               //Start timer
    IE2 = ET3;                  //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
IE2         DATA    0AFH
ET2         EQU     04H
ET3         EQU     20H
ET4         EQU     40H
AUXINTIF    DATA    0EFH
T2IF        EQU     01H
T3IF        EQU     02H
T4IF        EQU     04H

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
```

```
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     009BH
            LJMP    TM3ISR

            ORG     0100H
TM3ISR:
            CPL     P1.0                ;Test port
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     T3L,#66H            ;65536-11.0592M/12/1000
            MOV     T3H,#0FCH
            MOV     T4T3M,#08H          ;Start timer
            MOV     IE2,#ET3            ;Enable timer interrupt
            SETB    EA

            JMP     $

            END
```

12.5.24 Timer 3 (External count – T3 is extended for external falling edge interrupt)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"
```

```
sfr         T4T3M       =   0xd1;
sfr         T4L         =   0xd3;
sfr         T4H         =   0xd2;
sfr         T3L         =   0xd5;
sfr         T3H         =   0xd4;
sfr         T2L         =   0xd7;
sfr         T2H         =   0xd6;
sfr         AUXR        =   0x8e;
sfr         IE2         =   0xaf;
#define     ET2             0x04
#define     ET3             0x20
#define     ET4             0x40
sfr         AUXINTIF    =   0xef;
#define     T2IF            0x01
#define     T3IF            0x02
#define     T4IF            0x04

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        P10         =   P1^0;

void TM3_Isr() interrupt 19
{
    P10 = !P10;                 //Test port
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T3L = 0x66;                 //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x0c;               //Set external counting mode and start timer
    IE2 = ET3;                  //Enable timer interrupt
    EA = 1;

    while (1);
```

STC MCU

```
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz
T4T3M     DATA    0D1H
T4L       DATA    0D3H
T4H       DATA    0D2H
T3L       DATA    0D5H
T3H       DATA    0D4H
T2L       DATA    0D7H
T2H       DATA    0D6H
AUXR      DATA    8EH
IE2       DATA    0AFH
ET2       EQU     04H
ET3       EQU     20H
ET4       EQU     40H
AUXINTIF  DATA    0EFH
T2IF      EQU     01H
T3IF      EQU     02H
T4IF      EQU     04H

P1M1      DATA    091H
P1M0      DATA    092H
P0M1      DATA    093H
P0M0      DATA    094H
P2M1      DATA    095H
P2M0      DATA    096H
P3M1      DATA    0B1H
P3M0      DATA    0B2H
P4M1      DATA    0B3H
P4M0      DATA    0B4H
P5M1      DATA    0C9H
P5M0      DATA    0CAH

          ORG     0000H
          LJMP    MAIN
          ORG     009BH
          LJMP    TM3ISR

          ORG     0100H
TM3ISR:
          CPL     P1.0            ;Test port
          RETI

MAIN:
          MOV     SP, #5FH
          MOV     P0M0, #00H
          MOV     P0M1, #00H
          MOV     P1M0, #00H
          MOV     P1M1, #00H
          MOV     P2M0, #00H
          MOV     P2M1, #00H
          MOV     P3M0, #00H
          MOV     P3M1, #00H
          MOV     P4M0, #00H
          MOV     P4M1, #00H
          MOV     P5M0, #00H
          MOV     P5M1, #00H
```

STC MCU

```
        MOV     T3L,#66H                ;65536-11.0592M/12/1000
        MOV     T3H,#0FCH
        MOV     T4T3M,#0CH              ;Set external counting mode and start timer
        MOV     IE2,#ET3                ;Enable timer interrupt
        SETB    EA

        JMP     $

        END
```

12.5.25 Timer 3 (Divided clock output)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T4T3M   =   0xd1;
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;
```

STC MCU

```
    T3L = 0x66;                                 //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x09;                               //Enable clock output and start timer

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     T3L,#66H                ;65536-11.0592M/12/1000
            MOV     T3H,#0FCH
            MOV     T4T3M,#09H              ;Enable clock output and start timer

            JMP     $

            END
```

STC MCU

12.5.26 Timer 3 is used as baud rate generator of UART3

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       T4T3M       =   0xd1;
sfr       T4L         =   0xd3;
sfr       T4H         =   0xd2;
sfr       T3L         =   0xd5;
sfr       T3H         =   0xd4;
sfr       T2L         =   0xd7;
sfr       T2H         =   0xd6;
sfr       S3CON       =   0xac;
sfr       S3BUF       =   0xad;
sfr       IE2         =   0xaf;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}
```

STC MCU

```
void Uart3Init()
{
    S3CON = 0x50;
    T3L = BRT;
    T3H = BRT >> 8;
    T4T3M = 0x0a;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz
```

```
T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H
S3CON       DATA    0ACH
S3BUF       DATA    0ADH
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H             ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     008BH
            LJMP    UART3_ISR

            ORG     0100H

UART3_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S3CON
            JNB     ACC.1,CHKRI
            ANL     S3CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART3ISR_EXIT
            ANL     S3CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S3BUF
            INC     WPTR
UART3ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI
```

STC MCU

```
UART3_INIT:
        MOV     S3CON,#50H
        MOV     T3L,#0E8H               ;65536-11059200/115200/4=0FFE8H
        MOV     T3H,#0FFH
        MOV     T4T3M,#0AH
        CLR     BUSY
        MOV     WPTR,#00H
        MOV     RPTR,#00H
        RET

UART3_SEND:
        JB      BUSY,$
        SETB    BUSY
        MOV     S3BUF,A
        RET

UART3_SENDSTR:
        CLR     A
        MOVC    A,@A+DPTR
        JZ      SEND3END
        LCALL   UART3_SEND
        INC     DPTR
        JMP     UART3_SENDSTR
SEND3END:
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        LCALL   UART3_INIT
        MOV     IE2,#08H
        SETB    EA

        MOV     DPTR,#STRING
        LCALL   UART3_SENDSTR

LOOP:
        MOV     A,RPTR
        XRL     A,WPTR
        ANL     A,#0FH
        JZ      LOOP
        MOV     A,RPTR
        ANL     A,#0FH
        ADD     A,#BUFFER
        MOV     R0,A
        MOV     A,@R0
```

STC MCU

```
                LCALL   UART3_SEND
                INC     RPTR
                JMP     LOOP

STRING:         DB      'Uart Test !',0DH,0AH,00H

                END
```

12.5.27 Timer 4 (16-bit auto-reloadable)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T4T3M   =   0xd1;
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2         0x04
#define ET3         0x20
#define ET4         0x40
sfr     AUXINTIF    =   0xef;
#define T2IF        0x01
#define T3IF        0x02
#define T4IF        0x04

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM4_Isr() interrupt 20
{
    P10 = !P10;                 //Test port
}

void main()
{
```

```
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T4L = 0x66;                 //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0x80;               //Start timer
    IE2 = ET4;                  //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
IE2         DATA    0AFH
ET2         EQU     04H
ET3         EQU     20H
ET4         EQU     40H
AUXINTIF    DATA    0EFH
T2IF        EQU     01H
T3IF        EQU     02H
T4IF        EQU     04H

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00A3H
```

STC MCU

```
        LJMP    TM4ISR

        ORG     0100H
TM4ISR:
        CPL     P1.0                ;Test port
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     T4L,#66H            ;65536-11.0592M/12/1000
        MOV     T4H,#0FCH
        MOV     T4T3M,#80H          ;Start timer
        MOV     IE2,#ET4            ;Enable timer interrupt
        SETB    EA

        JMP     $

        END
```

12.5.28 Timer 4 (External count – T4 is extended for external falling edge interrupt)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T4T3M   =   0xd1;
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2     0x04
#define ET3     0x20
#define ET4     0x40
```

```
sfr     AUXINTIF    =   0xef;
#define T2IF            0x01
#define T3IF            0x02
#define T4IF            0x04

sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

sbit    P10         =   P1^0;

void TM4_Isr() interrupt 20
{
    P10 = !P10;                 //Test port
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T4L = 0x66;                 //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0xc0;               //Set external counting mode and start timer
    IE2 = ET4;                  //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T4T3M   DATA    0D1H
T4L     DATA    0D3H
T4H     DATA    0D2H
T3L     DATA    0D5H
T3H     DATA    0D4H
T2L     DATA    0D7H
```

```
T2H         DATA    0D6H
AUXR        DATA    8EH
IE2         DATA    0AFH
ET2         EQU     04H
ET3         EQU     20H
ET4         EQU     40H
AUXINTIF    DATA    0EFH
T2IF        EQU     01H
T3IF        EQU     02H
T4IF        EQU     04H

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00A3H
            LJMP    TM4ISR

            ORG     0100H
TM4ISR:
            CPL     P1.0            ;Test port
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     T4L,#66H        ;65536-11.0592M/12/1000
            MOV     T4H,#0FCH
            MOV     T4T3M,#0C0H     ;Set external counting mode and start timer
            MOV     IE2,#ET4        ;Enable timer interrupt
            SETB    EA

            JMP     $

            END
```

STC MCU

12.5.29 Timer 4 (Divided clock output)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T4T3M   =   0xd1;
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P3M1 = 0x00;

    T4L = 0x66;                     //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0x90;                   //Enable clock output and start timer

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz
```

STC MCU

```
T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     T4L,#66H            ;65536-11.0592M/12/1000
            MOV     T4H,#0FCH
            MOV     T4T3M,#90H          ;Enable clock output and start timer

            JMP     $

            END
```

12.5.30 Timer 4 is used as baud rate generator of UART4

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"
```

```
#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       T4T3M       =   0xd1;
sfr       T4L         =   0xd3;
sfr       T4H         =   0xd2;
sfr       T3L         =   0xd5;
sfr       T3H         =   0xd4;
sfr       T2L         =   0xd7;
sfr       T2H         =   0xd6;
sfr       S4CON       =   0x84;
sfr       S4BUF       =   0x85;
sfr       IE2         =   0xaf;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart4Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}

void Uart4Init()
{
    S4CON = 0x50;
    T4L = BRT;
    T4H = BRT >> 8;
    T4T3M = 0xa0;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}
```

STC MCU

```
void Uart4Send(char dat)
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T4T3M   DATA    0D1H
T4L     DATA    0D3H
T4H     DATA    0D2H
T3L     DATA    0D5H
T3H     DATA    0D4H
T2L     DATA    0D7H
T2H     DATA    0D6H
S4CON   DATA    84H
S4BUF   DATA    85H
IE2     DATA    0AFH
```

STC MCU

```
BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0093H
            LJMP    UART4_ISR

            ORG     0100H

UART4_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S4CON
            JNB     ACC.1,CHKRI
            ANL     S4CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART4ISR_EXIT
            ANL     S4CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S4BUF
            INC     WPTR
UART4ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART4_INIT:
            MOV     S4CON,#50H
            MOV     T4L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T4H,#0FFH
            MOV     T4T3M,#0A0H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET
```

STC MCU

```
UART4_SEND:
        JB      BUSY,$
        SETB    BUSY
        MOV     S4BUF,A
        RET

UART4_SENDSTR:
        CLR     A
        MOVC    A,@A+DPTR
        JZ      SEND4END
        LCALL   UART4_SEND
        INC     DPTR
        JMP     UART4_SENDSTR
SEND4END:
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        LCALL   UART4_INIT
        MOV     IE2,#10H
        SETB    EA

        MOV     DPTR,#STRING
        LCALL   UART4_SENDSTR

LOOP:
        MOV     A,RPTR
        XRL     A,WPTR
        ANL     A,#0FH
        JZ      LOOP
        MOV     A,RPTR
        ANL     A,#0FH
        ADD     A,#BUFFER
        MOV     R0,A
        MOV     A,@R0
        LCALL   UART4_SEND
        INC     RPTR
        JMP     LOOP

STRING: DB      'Uart Test !',0DH,0AH,00H

        END
```

STC MCU

13 Serial Port (UART) Communication

Product line	Number of UART
STC8G1K08 family	**2**
STC8G1K08-8Pin family	**1**
STC8G1K08A family	**1**
STC8G2K64S4 family	**4**
STC8G2K64S2 family	**2**
STC8G1K08T family	**1**
STC15H2K64S4 family	**4**

There are 4 full duplex asynchronous serial communication ports (UART in short) in STC8G series of microcontrollers. Each UART consists of two data buffers, a shift register, a serial control register and a baud rate generator. Each UART data buffer consists of two independent receive and transmit buffers, which can transmit and receive data simultaneously.

There are 4 modes for UART1 of STC8G series of microcontrollers, the baud rates of two modes of them are variable, and the baud rates of the other two modes are fixed. They can be chosen for different applications. There are only two modes in UART2, UART3 and UART4, and their baud rates are variable. Different baud rates and different modes can be set by software. It is flexible for the host to query the receiving or sending process, or use the interrupt method.

All the pins of UART1, UART2, UART3 and UART4 can be switched amoung multiple groups of ports using the pin switching function, so that a serial port can be multiplexed into severial serial ports in a time-sharing manner.

13.1 Registers Related to UARTs

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
SCON	UART1 control register	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	UART1 data buffer register	99H									0000,0000
S2CON	UART2 control register	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
S2BUF	UART2 data buffer registe	9BH									0000,0000
S3CON	UART3 control register	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S3BUF	UART3 data buffer register	ADH									0000,0000
S4CON	UART4 control register	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
S4BUF	UART4 data buffer register	85H									0000,0000
PCON	Power control register	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
AUXR	Auxiliary register 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
SADDR	UART1 slave address register	A9H									0000,0000
SADEN	UART1 slave address enable register	B9H									0000,0000

13.2 UART1

SM0/FE: If the SMOD0 bit in the PCON register is 1, this bit is the frame error detection flag. When the UART detects an invalid stop bit during reception, it is set by the UART receiver and must be cleared by software. If SMOD0 bit

13.2.1 UART1 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: If the SMOD0 bit in the PCON register is 1, this bit is the frame error detection flag. When the UART detects an invalid stop bit during reception, it is set by the UART receiver and must be cleared by software. If SMOD0 bit in PCON register is 0, this bit and SM1 specify the communication mode of UART1 as shown in the following table:

SM0	SM1	Mode of UART1	Function description
0	0	Mode 0	synchronous shift serial mode
0	1	Mode 1	8-bit UART, whose baud-rate is variable
1	0	Mode 2	9-bit UART, whose baud-rate is fixed
1	1	Mode 3	9-bit UART, whose baud-rate is variable

SM2: Mode 2 or mode 3 multi-machine communication enable control bit. When UART1 adopts mode 2 or mode 3, if the SM2 bit is 1 and the REN bit is 1, the receiver is in the Address Frame Filter state. In this case, the received 9th bit (RB8) can be used to filter the address frame. If RB8 = 1, it indicates that the frame is an address frame, the address information can enter SBUF and set RI bit. The address information is compared in the interrupt service routine. If RB8 = 0, it indicates that the frame is not an address frame, which should be discarded and keep RI = 0. In mode 2 or mode 3, if the SM2 bit is 0 and the REN bit is 1, the receiver is in a state where the address frame filtering is disabled. The received message can enter SBUF regardless of whether RB8 is 0 or 1, and make RI = 1. Here, RB8 is usually used as a check bit. Mode 1 and mode 0 are non-multi-machine communication modes. In these two modes, SM2 should be set to 0.

REN: Receive enable control bit.

0: disable UART1 receive data.

1: enable UART1 receive data.

TB8: The 9th bit be transmitted for UART1 in mode 2 and 3. It can be set or cleared by software. It is not used in mode 0 and mode 1.

RB8: The 9th bit received for UART1 in mode 2 and 3 which is usually used as a check bit or address frame/data frame flag. It is not used in mode 0 and mode 1.

TI: Transmit interrupt request flag of UART1. In mode 0, when the ransmission of the 8th bit completes, TI is set by the hardware automatically and requests the interrupt to the CPU. After the CPU responds the interrupt, TI must be cleared by software. In other modes, TI is set by the hardware automatically at the start of the stop bit transmittion and requests interrupts to the CPU. TI must be cleared by software after the interrupt is responded.

RI: Receive interrupt request flag of UART1. In mode 0, when the serial port receives the 8th bit of datum, RI is set by the hardware automatically and requests interrupt to the CPU. After the interrupt is responded, RI must be cleared by software. In other modes, RI is set by hardware automatically at the middle of stop bit the serial port received, and requests the interrupt to the CPU. After the interrupt is responded, RI

must be cleared by software.

13.2.2 UART1 data register (SBUF)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
SBUF	99H								

SBUF: It is used as the buffer in transmission and receiving of UART1. SBUF is actually two buffers, reading buffer and writing buffer. Two operations correspond to two different registers, one is write-only register (writing buffer), the other is read-only register (reading buffer). In fact, the CPU reads serial receive buffer when reads SBUF. When CPU writes to the SBUF will trigger the serial port to start sending data.

13.2.3 Power control register (PCON)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

SMOD: double Baud rate of UART1 control bit.

0: disable double baud rate of the UART1.

1: enable double baud rate of the UART1.

SMOD0: Frame error detection control bit.

0: No frame error detection function, SCON.7 is SM0 function.

1: enable frame error detection function. The function of SM0/FE is FE.

13.2.4 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	S1ST2

UART_M0x6: Baud rate select bit of UART1 while it works in mode 0.

0: The baud-rate of UART in mode 0 is SYSclk/12.

1: The baud-rate of UART in mode 0 is SYSclk/2.

S1ST2: UART1 baud rate generator select bit.

0: Select Timer 1 as the baud-rate generator of UART1.

1: Select Timer 2 as the baud-rate generator of UART1.

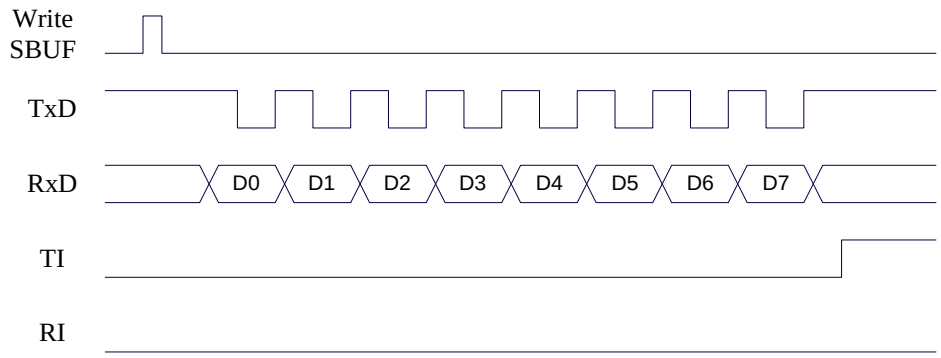
13.2.5 UART1 Mode 0

When mode 0 is selected for UART1, UART1 operates in synchronous shift register mode. When the serial port mode 0 communication speed setting bit UART_M0x6 is 0, the baud rate is fixed to SYSclk/12. When UART_M0x6 is 1, the baud rate is fixed to SYSclk/2. RxD is used as serial communication data pin, TxD is used as synchronous shift pulse output pin. 8-bit data are transmitted and received, LSB first.

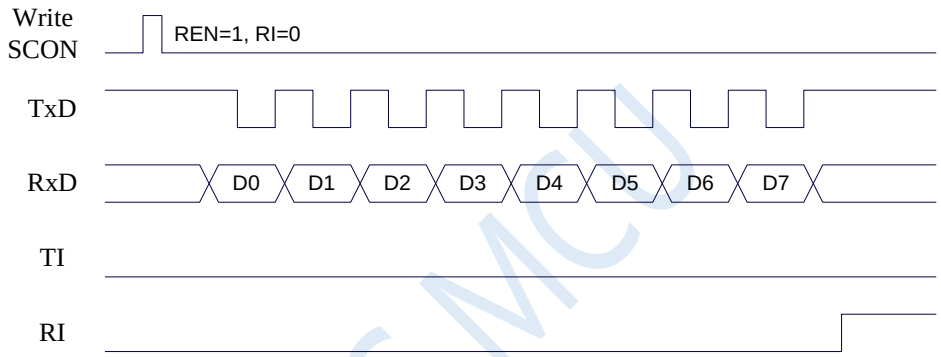
Transmission process of mode 0: Transmission is initiated by any instruction that write data to SBUF. The 8-bit datum is output from the RxD pin at the baud rate of SYSclk/12 or SYSclk/2 (determined by the UART_M0x6 divided by 12 or 2), from LSB to MSB. The TxD pin outputs the synchronous shift pulse signal. The interrupt flag TI will be set when transmittion is completed. When the write signal is valid, the transmit control signal SEND is active (high) one clock apart, allowing RxD to send data while allowing the TxD output the synchronous shift pulse. When a frame (8 bits) of datum is sent, all control signals are reset to the original status, and only TI keeps high level and keeps the interrupt request status. TI must be cleared by software before sending data again.

Receiving process of mode 0: Receiving is initiated by setting REN and the receive interrupt request flag RI=0. After starting the receive process, RxD is the serial data input pin and TxD is the synchronous pulse output pin. The

serial receiving baud rate is SYSclk/12 or SYSclk/2 (determined by UART_M0x6 is 12 or 2). After receiving a frame of datum (8 bits), the control signal is reset and the interrupt flag RI is set to 1, the interrupt requet status appears. RI must be cleared by software for the next receiving data.



Transmitting data (UART1 mode 0)



Receiving data (UART1 mode 0)

In mode 0, SM2 must be cleared so that TB8 and RB8 bits are not affected. Since the baud rate is fixed at SYSclk/12 or SYSclk/2, no timer is required and the clock of the microcontroller is used as the synchronous shift pulse directly.

The baud rates of UART1 mode 0 are shown in the following table, where SYSclk is the system operating frequency:

UART_M0x6	Baud rate calculation formula
0	$\text{Baud rate} = \frac{\text{SYSclk}}{12}$
1	$\text{Baud rate} = \frac{\text{SYSclk}}{2}$

13.2.6 UART1 Mode 1

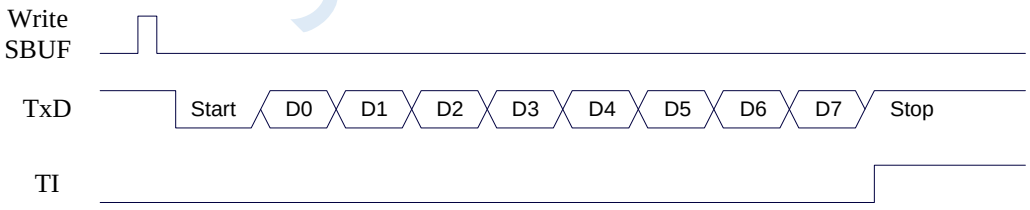
If SM0 and SM1 of SCON are set to '01' by the software, UART1 will work in mode 1, which is a 8-bit UART mode. In mode 1, a frame of information consists 10 bits: 1 start bit, 8 data bits (LSB first) and 1 stop bit. The baud rate is variable, which can be set by the software as needed. TxD is the data transmitting pin, and RxD is the data receiving pin, the UART is a full duplex receiver/transmitter.

Transmission process of mode 1: TxD is used as data output pin when transmitting a datum. Transmission is initiated by writing SBUF. "1" is also written into the 9th bit of transmission shift register by the writing "SBUF" signal, and the TX control unit is notified to start sending. The shift register shifts the data right to TxD to send, and shifts "0" in the left to supplement. When the highest bit of data is shifted to the output of the shift register, it is followed by the 9th bit "1", and all bits to the left of it are "0". This state causes the TX control unit to make the last shift output, and then disables the transmission signal "SEND" to complete the transmission of a frame and sets TI, and requests interrupt processing to CPU.

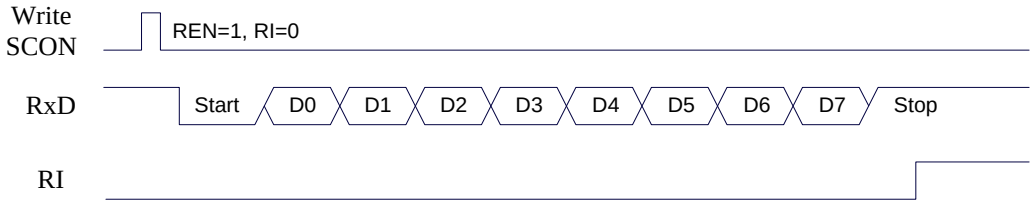
Receiving process of mode 1: After the software sets the reception enable flag REN, i.e. REN = 1, the receiver will detect the RxD pin signal. The receiver is ready to receive data when a "1" → "0" falling edge is detected at RxD pin, and resets the receiving counter of the baud rate generator immediately, loads 1FFH into the shift register. The received datum is shifted in from the right of the receiving shift register, the loaded 1FFH is shifted out to the left. When the start bit "0" is shifted to the far left of the shift register, the RX controller shifts for the last time and completes a frame receiving. The received datum is valid only if the following two conditions are met:

- RI=0;
- SM2=0 or the stop bit received is 1.

The datum received is loaded into SBUF, the stop bit is loaded into RB8, RI flag is set to request interrupt to CPU. If the two conditions can not be met at the same time, the received data is invalid and is discarded. Regardless of the conditions are met or not, the receiver will re-test RxD pin of the "1" → "0" edge, and continue to receive the next frame. If the received datum is valid, the RI flag must be cleared by software in the interrupt service routine. Usually, SM2 is set to "0" when serial port is operating in mode 1.



Transmitting data (UART1 mode 1)



Receiving data (UART1 mode 1)

The baud rate of UART1 is variable. It can be generated by T1 or T2. If the timer is in 1T mode (12x speed), the corresponding baud rate is increased by 12 times.

The baud rate of UART1 mode 1 is calculated as follows, where SYSclk is the system operating frequency.

Timer selected	Speed of timer	Baud rate calculation formula
T2	1T	$\text{reload value of T2} = 65536 - \frac{\text{SYSclk}}{4 \times \text{baud rate}}$
	12T	$\text{reload value of Timer 2} = 65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{baud rate}}$
T1 mode 0	1T	$\text{reload value of T1} = 65536 - \frac{\text{SYSclk}}{4 \times \text{baud rate}}$
	12T	$\text{reload value of T1} = 65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{baud rate}}$
T1 mode 2	1T	$\text{reload value of T1} = 256 - \frac{2^{SMOD} \times \text{SYSclk}}{32 \times \text{baud rate}}$
	12T	$\text{reload value of T1} = 256 - \frac{2^{SMOD} \times \text{SYSclk}}{12 \times 32 \times \text{baud rate}}$

The reload value of the timers corresponding to the common frequency and the common baud rate are as following.

Frequency (MHz)	Baud rate	T2		T1 mode 0		T1 mode 2			
		1T mode	12T mode	1T mode	12T mode	SMOD=1		1T mode	
						1T mode	12T mode	1T mode	12T mode
11.0592	115200	FFE8H	FFFEH	FFE8H	FFFEH	FAH	-	FDH	-
	57600	FFD0H	FFFCH	FFD0H	FFFCH	F4H	FFH	FAH	-
	38400	FFB8H	FFFAH	FFB8H	FFFAH	EEH	-	F7H	-
	19200	FF70H	FFF4H	FF70H	FFF4H	DCH	FDH	EEH	-
	9600	FEE0H	FFE8H	FEE0H	FFE8H	B8H	FAH	DCH	FDH
18.432	115200	FFD8H	-	FFD8H	-	F6H	-	FBH	-
	57600	FFB0H	-	FFB0H	-	ECH	-	F6H	-
	38400	FF88H	FFF6H	FF88H	FFF6H	E2H	-	F1H	-
	19200	FF10H	FFECH	FF10H	FFECH	C4H	FBH	E2H	-
	9600	FE20H	FFD8H	FE20H	FFD8H	88H	F6H	C4H	FBH
22.1184	115200	FFD0H	FFFCH	FFD0H	FFFCH	F4H	FFH	FAH	-
	57600	FFA0H	FFF8H	FFA0H	FFF8H	E8H	FEH	F4H	FFH
	38400	FF70H	FFF4H	FF70H	FFF4H	DCH	FDH	EEH	-
	19200	FEE0H	FFE8H	FEE0H	FFE8H	B8H	FAH	DCH	FDH
	9600	FDC0H	FFD0H	FDC0H	FFD0H	70H	F4H	B8H	FAH

13.2.7 UART1 Mode 2

If the two bits of SM0 and SM1 are '10', UART1 operates in mode 2. UART1 operating in mode 2 is a 9-bit data asynchronous communication UART. One frame of data consists of 11 bits: 1 start bit, 8 data bits (LSB first), 1 programmable bit (9th bit) and 1 stop bit. The transmitted programmable bit (9th bit) is supplied by TB8 in SCON, which can be confugred as either 1 or 0 by software. Or, the odd/even parity bit P in the PSW can be loaded into TB8. Not only can TB8 be used as a multi-machine communication address/data flag, but also it can be used as datum parity check bit. The 9th bit is received into RB8 of SCON. TxD is the transmitting pin, and RxD is the receiving pin, the serial port is a full duplex receiver/transmitter.

The baud rate of mode 2 is fixed to the system clock divided by 64 or 32 depending on the value of SMOD in PCON.

The baud rate of UART1 mode 2 is shown in the following table, where SYSclk is the system operating

frequency.

SMOD	Baud rate calculation formula
0	$\text{baud rate} = \frac{\text{SYSclk}}{64}$
1	$\text{baud rate} = \frac{\text{SYSclk}}{32}$

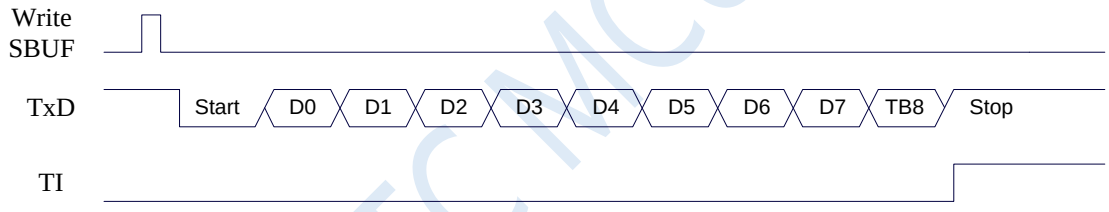
Except that the source of the baud rate is slightly different, and the 9th bit of the shift register supplied by TB8 while being sent is different, the functional and structure of mode 2 and mode 1 are basically the same, the receiving / sending operation and timing of mode 2 and mode 1are also basically the same.

After the receiver receives a frame of information, the following conditions must be met at the same time.

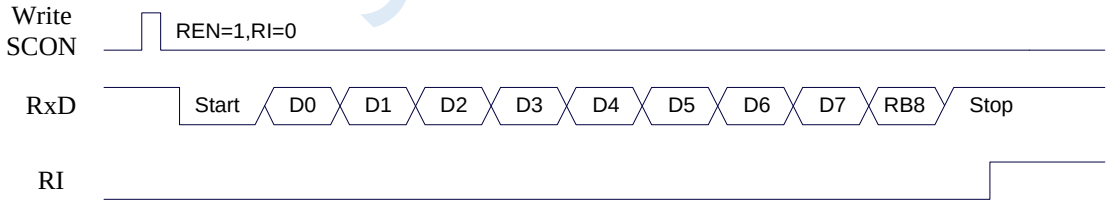
- RI=0
- SM2=0 or SM2=1 and the 9th bit received RB8=1.

Only when the two conditions above are met at the same time, the data received in shift register is loaded into SBUF and RB8. The RI flag is set to 1, and the interrupt is requested to CPU. If one of the above conditions is not met, the data received in the shift register is invalid and is discarded, and RI is not set. Regardless of the above conditions are met or not, the receiver begins to detect the RxD pin hopping information again to receive the next frame of information. In mode 2, the received stop bit is not related to SBUF, RB8 and RI.

It provides for the convenience of multi-machine communication by setting SM2, TB8 of SCON and communication protocol using the software.



Transmitting data (UART1 mode 2)



Receiving data (UART1 mode 2)

13.2.8 UART1 Mode 3

If the two bits of SM0 and SM1 are '11', UART1 operates in mode 3. UART1 operating in mode 3 is a 9-bit data asynchronous communication UART. One frame of data consists of 11 bits: 1 start bit, 8 data bits (LSB first), 1 programmable bit (9th bit) and 1 stop bit. The transmitted programmable bit (9th bit) is supplied by TB8 in SCON, which can be confugred as either 1 or 0 by software. Or, the odd/even parity bit P in the PSW can be loaded into TB8. Not only can TB8 be used as a multi-machine communication address/data flag, but also it can be used as datum parity check bit. The 9th bit is received into RB8 of SCON. TxD is the transmitting pin, and RxD is the receiving pin, the serial port is a full duplex receiver/transmitter.

Except that the 9th bit of the shift register supplied by TB8 while being sent is different, the functional and

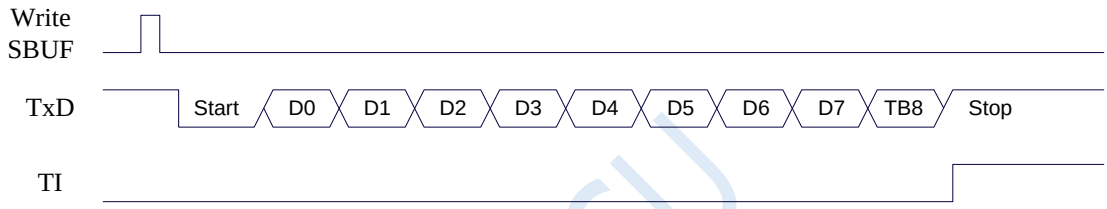
structure of mode 3 and mode 1 are basically the same, the receiving / sending operation and timing of mode 3 and mode 1 are also basically the same.

After the receiver receives a frame of information, the following conditions must be met at the same time.

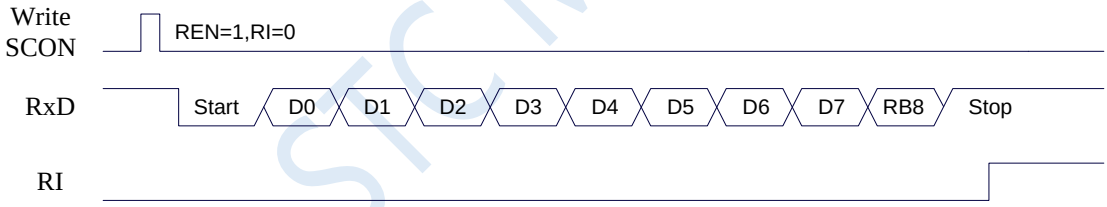
- RI=0
- SM2=0 or SM2=1 and the 9th bit received RB8=1.

Only when the two conditions above are met at the same time, the data received in shift register is loaded into SBUF and RB8. The RI flag is set to 1, and the interrupt is requested to CPU. If one of the above conditions is not met, the data received in the shift register is invalid and is discarded, and RI is not set. Regardless of the above conditions are met or not, the receiver begins to detect the RxD pin hopping information again to receive the next frame of information. In mode 3, the received stop bit is not related to SBUF, RB8 and RI.

It provides for the convenience of multi-machine communication by setting SM2, TB8 of SCON and communication protocol using the software.



Transmitting data (UART1 mode 3)



Receiving data (UART1 mode 3)

The baud rate calculation formula of UART1 mode 3 is exactly the same as that of mode 1. Please refer to the mode 1 baud rate calculation formula.

13.2.9 Automatic Address Recognition

UART1 slave address control registers (SADDR, SADEN)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
SADDR	A9H								
SADEN	B9H								

SADDR: Slave address register

SADEN: Slave address mask register

The automatic address recognition function is typically used in the field of multi-machine communications. Its main principle is that the slave system identifies the address information from the master serial port data stream through the hardware comparison function. The address of the slave is set by the registers SADDR and

SADEN. The hardware filters the slave address automatically. The hardware will generate a serial port interrupt when the slave address information from the master matches the slave address set by the slave. Otherwise, the hardware will discard the serial port data automatically without any interruption. When a number of slaves in Idle mode are connected together, only the slave that matches the slave address will wake up from Idle mode. Then the power consumption of the slave MCU reduces greatly. Constantly entering the serial port interrupt which reduces the system execution efficiency can be avoided even if the slave is in normal operation.

To use the automatic address recognition feature of the serial port, mode 2 or mode 3 of the serial port of the MCU that participates in communication is selected. Usually, the mode 3 with variable baud rate is selected because the baud rate of mode 2 is fixed, and it is inconvenient to adjust. SM2 bit of slave in SCON is set to 1. The 9th bit which is stored in RB8 is the address/data flag in mode 2 or 3. When the 9th bit is 1, it indicates the previous 8-bit datum stored in SBUF is the address information. If SM2 is set to 1, the slave MCU will filter out non-address data whose 9th bit is 0 automatically while the address data whose 9th bit is 1 in SBUF will automatically be matched with the address set in SADDR and SADEN. If the address matches, RI will be set to "1" and an interrupt will occur. Otherwise, the received data is discarded.

The slave address is set by two registers, SADDR and SADEN. SADDR is the slave address register, where the slave address is stored. SADEN is the slave address mask register, which is used to set the ignore bit in the address information. The setting method is as follows.

For example

SADDR = 11001010

SADEN = 10000001

Then the matched address is 1xxxxxx0

That is, as long as bit 0 is 0 and bit 7 is 1 in the address data sent by the master, the address can be matched with the local address.

Another example

SADDR = 11001010

SADEN = 00001111

Then the matched address is xxxx1010

That is, as long as the low 4 bits are 1010 in the address data sent by the master, the address can be matched with the local address. The high 4 bits can be any value and are ignored.

The Broadcast Address (FFH) can be used by the master to select all the slaves simultaneously for communication.

13.3 UART2

13.3.1 UART2 control register (S2CON)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
S2CON	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI

S2SM0: Serial port 2 mode select bit.

S2SM0	UART2 mode	Function description
0	Mode 0	8-bit UART, whose baud-rate is variable
1	Mode 1	9-bit UART, whose baud-rate is variable

S2SM2: UART2 multi-machine communication control enable bit. In mode 1, if the S2SM2 bit is 1 and the S2REN bit is 1, the receiver is in the address frame filter state. In this case, the received 9th bit (S2RB8)

can be used to filter the address frame. If S2RB8 = 1, the frame is the address frame, address information can enter S2BUF, S2RI becomes 1, and then address can be compared in the interrupt service routine. If S2RB8 = 0, it indicates that the frame is not an address frame and should be discarded and keep S2RI = 0. In mode 1, if the S2SM2 bit is 0 and the S2REN bit is 1, the receiver is in the address frame filter disabled state. Regardless of the received S2RB8 is 0 or 1, the information received can enter into the S2BUF, and make S2RI = 1. Here, S2RB8 is usually used as check bit. Mode 0 is non-multi-machine communication mode, where S2SM2 should be 0.

S2REN: Receive enable control bit.

0: disable UART2 receive data.

1: enable UART2 receive data.

S2TB8: S2TB8 is the 9th bit of datum to be sent when UART2 is in mode 1, which is usually used as a parity check bit or an address frame / data frame flag. It can be set or cleared by software as required. In mode 0, this bit is not used.

S2RB8: S2RB8 is the 9th bit of datum recieved when UART2 is in mode 1, which is usually used as a parity check bit or an address frame / data frame flag. It can be set or cleared by software as required. In mode 0, this bit is not used.

S2TI: Transmit interrupt request flag of UART2. S2TI is set by the hardware automatically at the beginning of the stop bit transmittion and requests interrupts to the CPU. S2TI must be cleared by software after the interrupt is responded.

S2RI: Receive interrupt request flag of UART2. S2RI is set by hardware automatically at the middle of stop bit received, and requests the interrupt to the CPU. After the interrupt is responded, S2RI must be cleared by software.

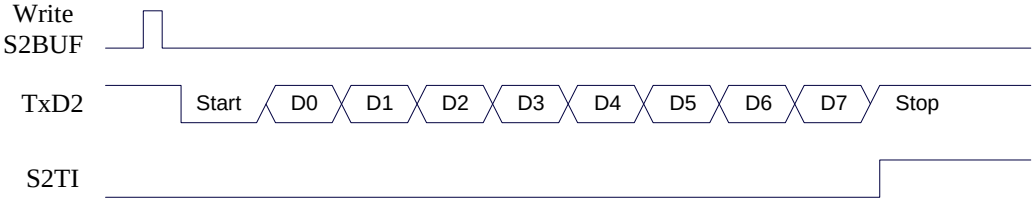
13.3.2 UART2 data register (S2BUF)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
S2BUF	9BH								

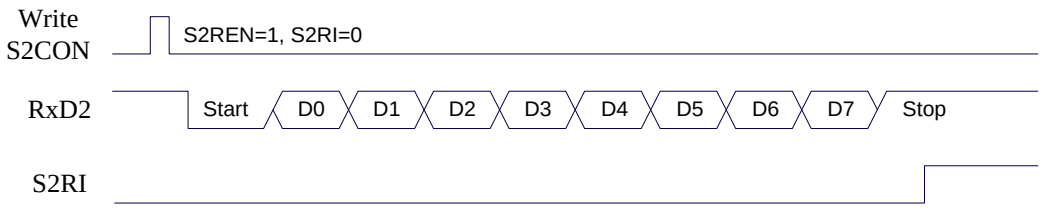
S2BUF: It is used as the buffer in transmission and receiving for UART2. S2BUF is actually two buffers, reading buffer and writing buffer. Two operations correspond to two different registers, one is write-only register (writing buffer), the other is read-only register (reading buffer). The CPU reads serial receiving buffer when reads S2BUF, and writes to the S2BUF will trigger the serial port to start sending data.

13.3.3 UART2 Mode 0

Serial port 2 mode 0 is 8-bit UART mode with ariable baud rate. In this mode, a frame of data consists 10 bits: 1 start bit, 8 data bits (LSB first) and 1 stop bit. The baud rate is variable, which can be set by the software as needed. TxD2 is the data transmitting pin, and RxD2 is the data receiving pin, the serial port is a full duplex receiver/transmitter.



Transmitting data (UART2 mode 0)



Receiving data (UART2 mode 0)

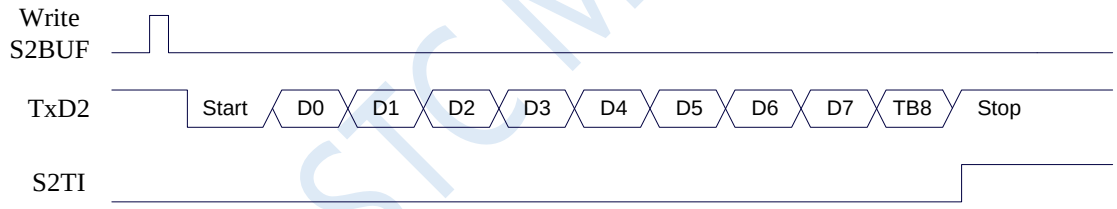
The baud rate of UART2 is variable. It is generated by T2. If the timer is in 1T mode (12x speed), the corresponding baud rate is increased by 12 times.

The baud rate of UART2 mode 0 is calculated as follows, where SYSclk is the system operating frequency.

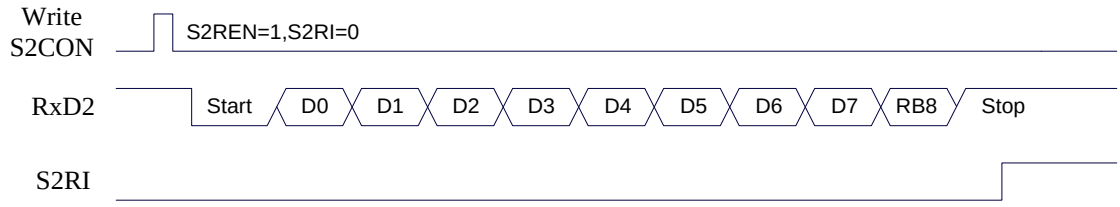
Timer selected	Speed of timer	Baud rate calculation formula
T2	1T	$\text{reload value of timer 2} = 65536 - \frac{SYSclk}{4 \times \text{baud rate}}$
	12T	$\text{reload value of timer 2} = 65536 - \frac{SYSclk}{12 \times 4 \times \text{baud rate}}$

13.3.4 UART2 Mode 1

UART2 operating in mode 1 is a 9-bit data UART mode with variable baud rate. One frame data consists of 11 bits: 1 start bit, 8 data bits (LSB first), 1 programmable bit (9th bit) and 1 stop bit. The baud rate is variable, which can be set by the software as needed. TxD2 is the data transmitting pin, and RxD2 is the data receiving pin, the serial port is a full duplex receiver/transmitter.



Transmitting data (UART2 mode 1)



Receiving data (UART2 mode 1)

The baud rate calculation formula of UART2 mode 1 is exactly the same as that of mode 0. Please refer to the mode 0 baud rate calculation formula.

13.4 UART3

13.4.1 UART3 control register (S3CON)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
S3CON	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI

S3SM0: UART3 mode select bit.

S3SM0	UART3 mode	Function description
0	Mode 0	8-bit UART, whose baud-rate is variable
1	Mode 1	9-bit UART, whose baud-rate is variable

S3ST3: UART3 baud rate generator select bit.

0: Select T2 as the baud-rate generator of UART3.

1: Select T3 as the baud-rate generator of UART3.

S3SM2: UART3 multi-machine communication control bit. In mode 1, if the S3SM2 bit is 1 and the S3REN bit is 1, the receiver is in the address frame filter state. In this case, the received 9th bit (S3RB8) can be used to filter the address frame. If S3RB8 = 1, the frame is the address frame, address information can enter S3BUF, S3RI becomes 1, and then the address is compared with the slave address in the interrupt service routine. If S3RB8 = 0, it indicates that the frame is not an address frame and should be discarded and keep S3RI = 0. In mode 1, if the S3SM2 bit is 0 and the S3REN bit is 1, the receiver is in the address frame filter disabled state. Regardless of the received S3RB8 is 0 or 1, the information received can enter into the S3BUF, and make S3RI = 1. Here, S3RB8 is usually used as parity check bit. Mode 0 is non-multi-machine communication mode, where S3SM2 should be 0.

S3REN: Receive enable control bit.

0: disable UART3 receive data.

1: enable UART3 receive data.

S3TB8: S3TB8 is the 9th bit of datum to be sent when UART3 is in mode 1, which is usually used as a parity check bit or an address frame / data frame flag. It can be set or cleared by software as required. In mode 0, this bit is not used.

S3RB8: S3RB8 is the 9th bit of datum recieved when UART3 is in mode 1, which is usually used as a parity check bit or an address frame / data frame flag. It can be set or cleared by software as required. In mode 0, this bit is not used.

S3TI: Transmit interrupt request flag of UART3. S3TI is set by the hardware automatically at the beginning of the stop bit transmittion and requests interrupt to the CPU. S3TI must be cleared by software after the interrupt is responded.

S3RI: Receive interrupt request flag of UART3. S3RI is set by hardware automatically at the middle of stop bit the serial port received, and requests interrupt to the CPU. After the interrupt is responded, S3RI must be cleared by software.

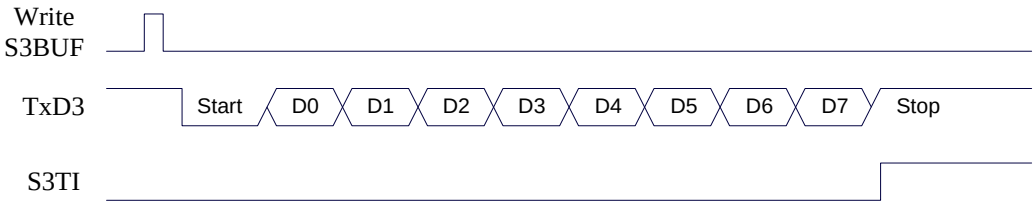
13.4.2 UART3 data register (S3BUF)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
S3BUF	ADH								

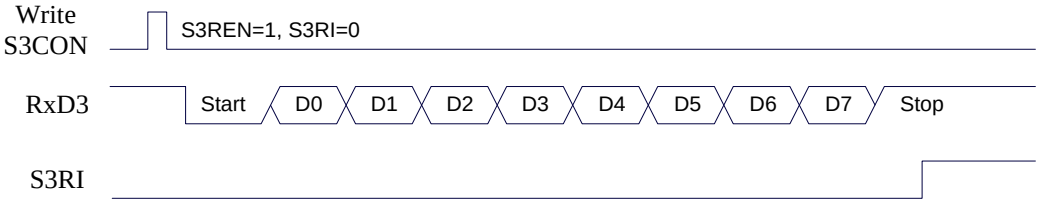
S3BUF: It is used as the buffer in transmission and receiving for UART3. S3BUF is actually two buffers, reading buffer and writing buffer. Two operations correspond to two different registers, one is write-only register (writing buffer), the other is read-only register (reading buffer). The CPU reads serial receive buffer when reads S3BUF, and writes to the S3BUF will trigger the serial port to start sending data.

13.4.3 UART3 Mode 0

UART3 mode 0 is 8-bit UART mode with variable baud rate, where a frame of data consists 10 bits: 1 start bit, 8 data bits (LSB first) and 1 stop bit. The baud rate is variable, which can be set by the software as needed. TxD3 is the data transmitting pin, and RxD3 is the data receiving pin, the serial port is a full duplex receiver/transmitter.



Transmitting data (UART3 mode 0)



Receiving data (UART3 mode 0)

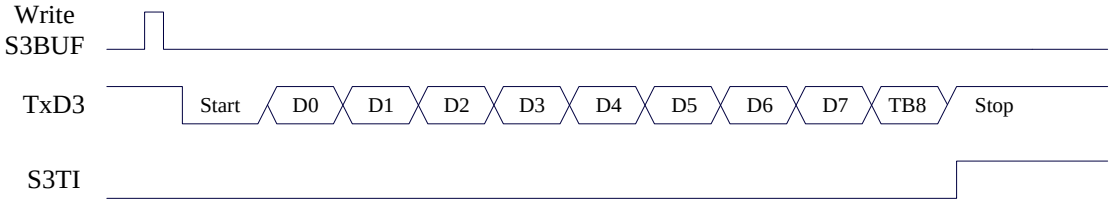
The baud rate of UART3 is variable. It is generated by T2 or T3. If the timer is in 1T mode (12x speed), the corresponding baud rate is increased by 12 times.

The baud rate of UART3 mode 0 is calculated as follows, where SYSclk is the system operating frequency.

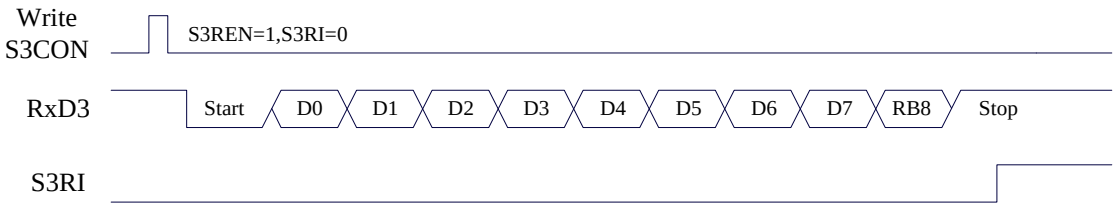
Timer selected	Speed of timer	Baud rate calculation formula
T2	1T	$\text{reload value of T2} = 65536 - \frac{\text{SYSclk}}{4 \times \text{baud rate}}$
	12T	$\text{reload value of T2} = 65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{baud rate}}$
T3	1T	$\text{reload value of T3} = 65536 - \frac{\text{SYSclk}}{4 \times \text{baud rate}}$
	12T	$\text{reload value of T3} = 65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{baud rate}}$

13.4.4 UART3 Mode 1

UART3 operating in mode 1 is 9-bit data UART mode with variable baud rate. One frame data consists of 11 bits: 1 start bit, 8 data bits (LSB first), 1 programmable bit (9^{th} bit) and 1 stop bit. The baud rate is variable, which can be set by the software as needed. TxD3 is the data transmitting pin, and RxD3 is the data receiving pin, the serial port is a full duplex receiver/transmitter.



Transmitting data (UART3 mode 1)



Receiving data (UART3 mode 1)

The baud rate calculation formula of UART3 mode 1 is exactly the same as that of mode 0. Please refer to the mode 0 baud rate calculation formula.

13.5 UART4

13.5.1 UART4 control register (S4CON)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
S4CON	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

S4SM0: UART4 mode select bit.

S4SM0	UART4 mode	Function description
0	Mode 0	8-bit UART, whose baud-rate is variable
1	Mode 1	9-bit UART, whose baud-rate is variable

S4ST4: UART4 baud rate generator select bit.

0: Select T2 as the baud-rate generator of UART4.

1: Select T4 as the baud-rate generator of UART4.

S4SM2: UART4 multi-machine communication control bit. In mode 1, if the S4SM2 bit is 1 and the S4REN bit is 1, the receiver is in the address frame filter state. In this case, the received 9th bit (S4RB8) can be used to filter the address frame. If S4RB8 = 1, the frame is the address frame, address information can enter S4BUF, S4RI becomes 1, and then address can be compared with the slave address in the interrupt service routine. If S4RB8 = 0, it indicates that the frame is not an address frame and should be discarded and keep S4RI = 0. In mode 1, if the S4SM2 bit is 0 and the S4REN bit is 1, the receiver is in the address frame filter disabled state. Regardless of the received S4RB8 is 0 or 1, the information received can enter into the S4BUF, and make S4RI = 1. Here, S4RB8 is usually used as parity check bit. Mode 0 is non-multi-machine communication mode, where S4SM2 should be 0.

S4REN: Receive enable control bit.

0: disable UART4 receive data.

1: enable UART4 receive data.

S4TB8: S4TB8 is the 9th bit of datum to be sent when UART4 is in mode 1, which is usually used as a parity check bit or an address frame / data frame flag. It can be set or cleared by software as required. In mode 0, this bit is not used.

S4RB8: S4RB8 is the 9th bit of datum recieved when UART4 is in mode 1, which is usually used as a parity check bit or an address frame / data frame flag. It can be set or cleared by software as required. In mode 0, this bit is not used.

S4TI: Transmit interrupt request flag of UART4. S4TI is set by the hardware automatically at the beginning of the stop bit transmittion and requests interrupt to the CPU. S4TI must be cleared by software after the interrupt is responded.

S4RI: Receive interrupt request flag of UART4. S4RI is set by hardware automatically at the middle of stop bit the serial port received, and requests interrupt to the CPU. After the interrupt is responded, S4RI must be

cleared by software.

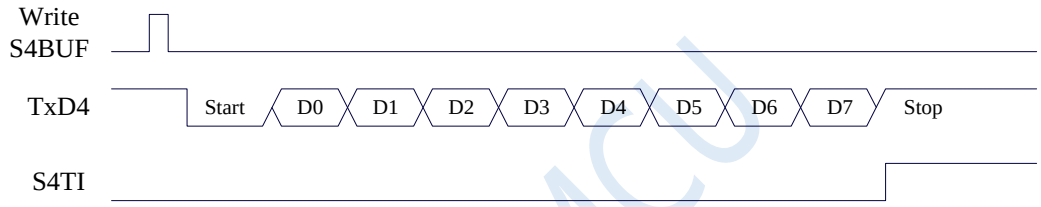
13.5.2 UART4 data register (S4BUF)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
S4BUF	85H								

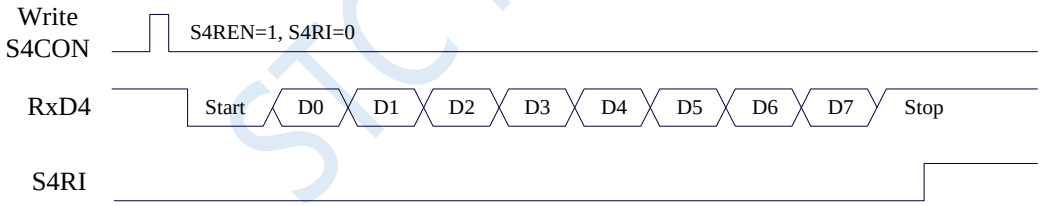
S4BUF: It is used as the buffer in transmission and receiving for UART4. S4BUF is actually two buffers, reading buffer and writing buffer. Two operations correspond to two different registers, one is write-only register (writing buffer), the other is read-only register (reading buffer). The CPU reads serial receive buffer when reads S4BUF, and writes to the S4BUF will trigger the serial port to start sending data.

13.5.3 UART4 Mode 0

UART4 mode 0 is 8-bit UART mode with variable baud rate, where a frame of data consists 10 bits: 1 start bit, 8 data bits (LSB first) and 1 stop bit. The baud rate is variable, which can be set by the software as needed. TxD4 is the data transmitting pin, and RxD4 is the data receiving pin, the serial port is a full duplex receiver/transmitter.



Transmitting data (UART4 mode 0)



Receiving data (UART4 mode 0)

The baud rate of UART4 is variable. It is generated by T2 or T4. If the timer is in 1T mode (12x speed), the corresponding baud rate is increased by 12 times.

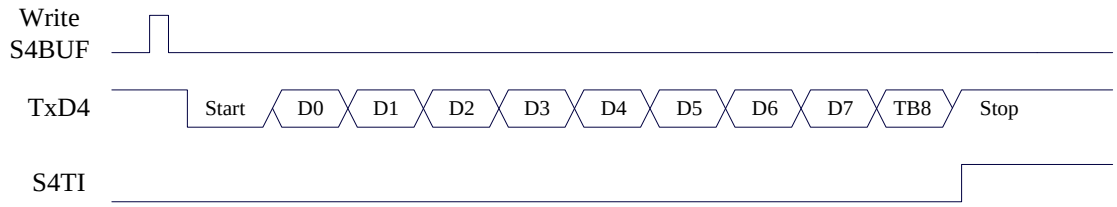
The baud rate of UART4 mode 0 is calculated as follows, where SYSclk is the system operating frequency.

Timer selected	Speed of timer	Baud rate calculation formula
T2	1T	$\text{reload value of T2} = 65536 - \frac{SYSclk}{4 \times \text{baud rate}}$
	12T	$\text{reload value of T2} = 65536 - \frac{SYSclk}{12 \times 4 \times \text{baud rate}}$
T4	1T	$\text{reload value of T4} = 65536 - \frac{SYSclk}{4 \times \text{baud rate}}$
	12T	$\text{reload value of T4} = 65536 - \frac{SYSclk}{12 \times 4 \times \text{baud rate}}$

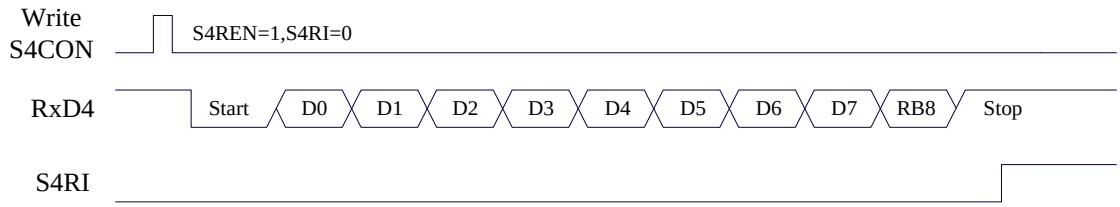
13.5.4 UART4 Mode 1

UART4 operating in mode 1 is 9-bit data UART mode with variable baud rate. One frame data consists of

11 bits: 1 start bit, 8 data bits (LSB first), 1 programmable bit (9th bit) and 1 stop bit. The baud rate is variable, which can be set by the software as needed. TxD4 is the data transmitting pin, and RxD4 is the data receiving pin, the serial port is a full duplex receiver/transmitter.



Transmitting data (UART4 mode 1)



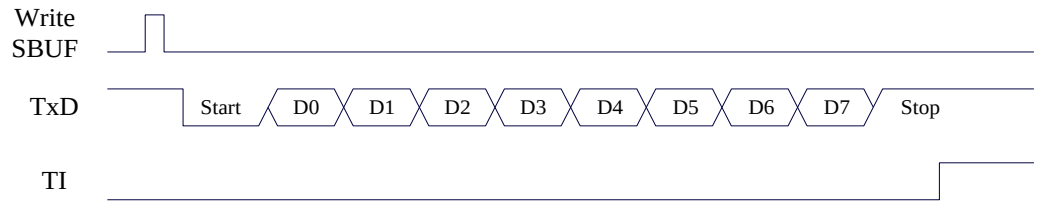
Receiving data (UART4 mode 1)

The baud rate calculation formula of UART4 mode 1 is exactly the same as that of mode 0. Please refer to the mode 0 baud rate calculation formula.

13.6 Precautions of UARTs

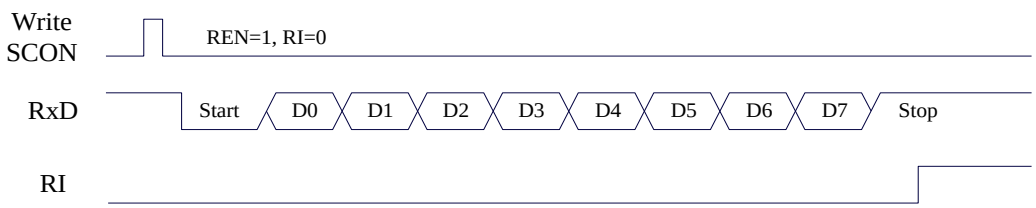
Regarding the UART interrupts requests, the following issues need to be noted. UART1, UART2, UART3, and UART4 are all similar, and serial port 1 is used as an example below.

In 8-bit data mode, TI interrupt request is generated after the entire stop bit is transmitted, as shown in the following figure:



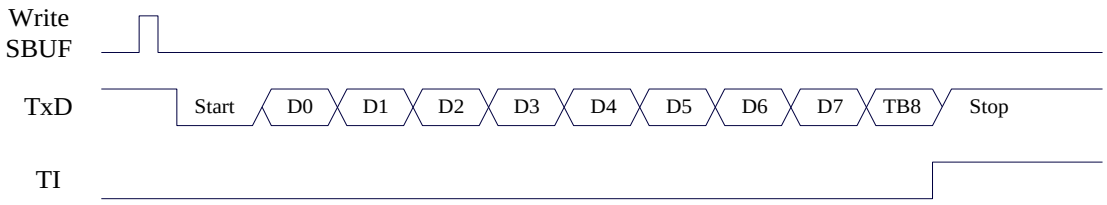
Transmission data (8-bit data)

In 8-bit data mode, RI interrupt request is generated after half of the stop bit is received, as shown in the following figure:



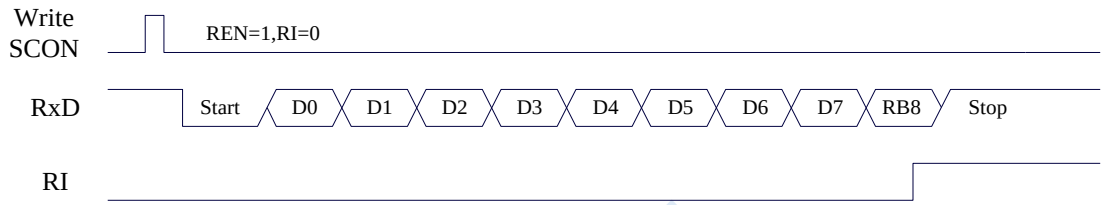
Receiving data (8-bit data)

In 9-bit data mode, TI interrupt request is generated after the entire 9th data bit is transmitted, as shown in the following figure:



Transmission data (9-bit data)

In 9-bit data mode, RI interrupt request is generated after receiving half of the 9th bit, as shown in the following figure:



Receiving data (9-bit data)

13.7 Example Routines

13.7.1 UART1 using T2 as baud rate generator

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       AUXR    =   0x8e;
sfr       T2H     =   0xd6;
sfr       T2L     =   0xd7;

sfr       P0M1    =   0x93;
sfr       P0M0    =   0x94;
sfr       P1M1    =   0x91;
sfr       P1M0    =   0x92;
sfr       P2M1    =   0x95;
sfr       P2M0    =   0x96;
sfr       P3M1    =   0xb1;
sfr       P3M0    =   0xb2;
sfr       P4M1    =   0xb3;
sfr       P4M0    =   0xb4;
sfr       P5M1    =   0xc9;
sfr       P5M0    =   0xca;
```

```
bit		busy;
char		wptr;
char		rptr;
char		buffer[16];

void UartIsr() interrupt 4
{
	if (TI)
	{
		TI = 0;
		busy = 0;
	}
	if (RI)
	{
		RI = 0;
		buffer[wptr++] = SBUF;
		wptr &= 0x0f;
	}
}

void UartInit()
{
	SCON = 0x50;
	T2L = BRT;
	T2H = BRT >> 8;
	AUXR = 0x15;
	wptr = 0x00;
	rptr = 0x00;
	busy = 0;
}

void UartSend(char dat)
{
	while (busy);
	busy = 1;
	SBUF = dat;
}

void UartSendStr(char *p)
{
	while (*p)
	{
		UartSEND(*p++);
	}
}

void main()
{
	P0M0 = 0x00;
	P0M1 = 0x00;
	P1M0 = 0x00;
	P1M1 = 0x00;
	P2M0 = 0x00;
	P2M1 = 0x00;
	P3M0 = 0x00;
	P3M1 = 0x00;
	P4M0 = 0x00;
	P4M1 = 0x00;
```

STC MCU

```
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
```

```
CHKRI:
        JNB     RI,UARTISR_EXIT
        CLR     RI
        MOV     A,WPTR
        ANL     A,#0FH
        ADD     A,#BUFFER
        MOV     R0,A
        MOV     @R0,SBUF
        INC     WPTR
UARTISR_EXIT:
        POP     PSW
        POP     ACC
        RETI

UART_INIT:
        MOV     SCON,#50H
        MOV     T2L,#0E8H               ;65536-11059200/115200/4=0FFE8H
        MOV     T2H,#0FFH
        MOV     AUXR,#15H
        CLR     BUSY
        MOV     WPTR,#00H
        MOV     RPTR,#00H
        RET

UART_SEND:
        JB      BUSY,$
        SETB    BUSY
        MOV     SBUF,A
        RET

UART_SENDSTR:
        CLR     A
        MOVC    A,@A+DPTR
        JZ      SENDEND
        LCALL   UART_SEND
        INC     DPTR
        JMP     UART_SENDSTR
SENDEND:
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        LCALL   UART_INIT
        SETB    ES
        SETB    EA
```

STC MCU

```
                MOV         DPTR,#STRING
                LCALL       UART_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

13.7.2 UART1 using T1 (Mode 0) as baud rate generator

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       AUXR        =   0x8e;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
```

```
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSENDStr("Uart Test !\r\n");
```

STC MCU

```
    while (1)
    {
        if (rptr != wptr)
        {
            UartSEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,SBUF
            INC     WPTR
```

```
UARTISR_EXIT:
        POP     PSW
        POP     ACC
        RETI

UART_INIT:
        MOV     SCON,#50H
        MOV     TMOD,#00H
        MOV     TL1,#0E8H           ;65536-11059200/115200/4=0FFE8H
        MOV     TH1,#0FFH
        SETB    TR1
        MOV     AUXR,#40H
        CLR     BUSY
        MOV     WPTR,#00H
        MOV     RPTR,#00H
        RET

UART_SEND:
        JB      BUSY,$
        SETB    BUSY
        MOV     SBUF,A
        RET

UART_SENDSTR:
        CLR     A
        MOVC    A,@A+DPTR
        JZ      SENDEND
        LCALL   UART_SEND
        INC     DPTR
        JMP     UART_SENDSTR
SENDEND:
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        LCALL   UART_INIT
        SETB    ES
        SETB    EA

        MOV     DPTR,#STRING
        LCALL   UART_SENDSTR

LOOP:
        MOV     A,RPTR
        XRL     A,WPTR
        ANL     A,#0FH
```

STC MCU

```
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

13.7.3 UART1 using T1 (Mode 2) as baud rate generator

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (256 - FOSC / 115200 / 32)

sfr       AUXR        =   0x8e;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
```

STC MCU

```
            buffer[wptr++] = SBUF;
            wptr &= 0x0f;
        }
}

void UartInit()
{
        SCON = 0x50;
        TMOD = 0x20;
        TL1 = BRT;
        TH1 = BRT;
        TR1 = 1;
        AUXR = 0x40;
        wptr = 0x00;
        rptr = 0x00;
        busy = 0;
}

void UartSend(char dat)
{
        while (busy);
        busy = 1;
        SBUF = dat;
}

void UartSendStr(char *p)
{
        while (*p)
        {
            UartSEND(*p++);
        }
}

void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        UartInit();
        ES = 1;
        EA = 1;
        UartSENDStr("Uart Test !\r\n");

        while (1)
        {
            if (rptr != wptr)
            {
                UartSEND(buffer[rptr++]);
                rptr &= 0x0f;
```

STC MCU

```
        }
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H             ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,SBUF
            INC     WPTR
UARTISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART_INIT:
            MOV     SCON,#50H
```

STC MCU

```
        MOV     TMOD,#20H
        MOV     TL1,#0FDH       ;256-11059200/115200/32=0FDH
        MOV     TH1,#0FDH
        SETB    TR1
        MOV     AUXR,#40H
        CLR     BUSY
        MOV     WPTR,#00H
        MOV     RPTR,#00H
        RET

UART_SEND:
        JB      BUSY,$
        SETB    BUSY
        MOV     SBUF,A
        RET

UART_SENDSTR:
        CLR     A
        MOVC    A,@A+DPTR
        JZ      SENDEND
        LCALL   UART_SEND
        INC     DPTR
        JMP     UART_SENDSTR
SENDEND:
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        LCALL   UART_INIT
        SETB    ES
        SETB    EA

        MOV     DPTR,#STRING
        LCALL   UART_SENDSTR

LOOP:
        MOV     A,RPTR
        XRL     A,WPTR
        ANL     A,#0FH
        JZ      LOOP
        MOV     A,RPTR
        ANL     A,#0FH
        ADD     A,#BUFFER
        MOV     R0,A
        MOV     A,@R0
        LCALL   UART_SEND
```

STC MCU

```
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

13.7.4 UART2 using T2 as baud rate generator

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       AUXR        =   0x8e;
sfr       T2H         =   0xd6;
sfr       T2L         =   0xd7;
sfr       S2CON       =   0x9a;
sfr       S2BUF       =   0x9b;
sfr       IE2         =   0xaf;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart2Isr() interrupt 8
{
    if (S2CON & 0x02)
    {
        S2CON &= ~0x02;
        busy = 0;
    }
    if (S2CON & 0x01)
    {
        S2CON &= ~0x01;
        buffer[wptr++] = S2BUF;
        wptr &= 0x0f;
```

STC MCU

```
    }
}

void Uart2Init()
{
    S2CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart2Send(char dat)
{
    while (busy);
    busy = 1;
    S2BUF = dat;
}

void Uart2SendStr(char *p)
{
    while (*p)
    {
        Uart2SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart2Init();
    IE2 = 0x01;
    EA = 1;
    Uart2SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart2SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

STC MCU

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H
S2CON       DATA    9AH
S2BUF       DATA    9BH
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0043H
            LJMP    UART2_ISR

            ORG     0100H

UART2_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S2CON
            JNB     ACC.1,CHKRI
            ANL     S2CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART2ISR_EXIT
            ANL     S2CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S2BUF
            INC     WPTR
UART2ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART2_INIT:
```

```
        MOV     S2CON,#10H
        MOV     T2L,#0E8H       ;65536-11059200/115200/4=0FFE8H
        MOV     T2H,#0FFH
        MOV     AUXR,#14H
        CLR     BUSY
        MOV     WPTR,#00H
        MOV     RPTR,#00H
        RET

UART2_SEND:
        JB      BUSY,$
        SETB    BUSY
        MOV     S2BUF,A
        RET

UART2_SENDSTR:
        CLR     A
        MOVC    A,@A+DPTR
        JZ      SEND2END
        LCALL   UART2_SEND
        INC     DPTR
        JMP     UART2_SENDSTR
SEND2END:
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        LCALL   UART2_INIT
        MOV     IE2,#01H
        SETB    EA

        MOV     DPTR,#STRING
        LCALL   UART2_SENDSTR

LOOP:
        MOV     A,RPTR
        XRL     A,WPTR
        ANL     A,#0FH
        JZ      LOOP
        MOV     A,RPTR
        ANL     A,#0FH
        ADD     A,#BUFFER
        MOV     R0,A
        MOV     A,@R0
        LCALL   UART2_SEND
        INC     RPTR
```

STC MCU

```
        JMP     LOOP

STRING: DB      'Uart Test !',0DH,0AH,00H

        END
```

13.7.5 UART3 using T2 as baud rate generator

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define FOSC    11059200UL
#define BRT     (65536 - FOSC / 115200 / 4)

sfr     AUXR    =   0x8e;
sfr     T2H     =   0xd6;
sfr     T2L     =   0xd7;
sfr     S3CON   =   0xac;
sfr     S3BUF   =   0xad;
sfr     IE2     =   0xaf;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

bit     busy;
char    wptr;
char    rptr;
char    buffer[16];

void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
```

STC MCU

```
}

void Uart3Init()
{
    S3CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

STC MCU

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H
S3CON       DATA    0ACH
S3BUF       DATA    0ADH
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     008BH
            LJMP    UART3_ISR

            ORG     0100H

UART3_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S3CON
            JNB     ACC.1,CHKRI
            ANL     S3CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART3ISR_EXIT
            ANL     S3CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S3BUF
            INC     WPTR
UART3ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART3_INIT:
```

```
        MOV     S3CON,#10H
        MOV     T2L,#0E8H       ;65536-11059200/115200/4=0FFE8H
        MOV     T2H,#0FFH
        MOV     AUXR,#14H
        CLR     BUSY
        MOV     WPTR,#00H
        MOV     RPTR,#00H
        RET

UART3_SEND:
        JB      BUSY,$
        SETB    BUSY
        MOV     S3BUF,A
        RET

UART3_SENDSTR:
        CLR     A
        MOVC    A,@A+DPTR
        JZ      SEND3END
        LCALL   UART3_SEND
        INC     DPTR
        JMP     UART3_SENDSTR
SEND3END:
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        LCALL   UART3_INIT
        MOV     IE2,#08H
        SETB    EA

        MOV     DPTR,#STRING
        LCALL   UART3_SENDSTR

LOOP:
        MOV     A,RPTR
        XRL     A,WPTR
        ANL     A,#0FH
        JZ      LOOP
        MOV     A,RPTR
        ANL     A,#0FH
        ADD     A,#BUFFER
        MOV     R0,A
        MOV     A,@R0
        LCALL   UART3_SEND
        INC     RPTR
```

STC MCU

```
        JMP     LOOP

STRING: DB      'Uart Test !',0DH,0AH,00H

        END
```

13.7.6 UART3 using T3 as baud rate generator

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define FOSC    11059200UL
#define BRT     (65536 - FOSC / 115200 / 4)

sfr     T4T3M   =   0xd1;
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     S3CON   =   0xac;
sfr     S3BUF   =   0xad;
sfr     IE2     =   0xaf;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

bit     busy;
char    wptr;
char    rptr;
char    buffer[16];

void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
```

STC MCU

```
            S3CON &= ~0x01;
            buffer[wptr++] = S3BUF;
            wptr &= 0x0f;
        }
}

void Uart3Init()
{
        S3CON = 0x50;
        T3L = BRT;
        T3H = BRT >> 8;
        T4T3M = 0x0a;
        wptr = 0x00;
        rptr = 0x00;
        busy = 0;
}

void Uart3Send(char dat)
{
        while (busy);
        busy = 1;
        S3BUF = dat;
}

void Uart3SendStr(char *p)
{
        while (*p)
        {
            Uart3SEND(*p++);
        }
}

void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        Uart3Init();
        IE2 = 0x08;
        EA = 1;
        Uart3SENDStr("Uart Test !\r\n");

        while (1)
        {
            if (rptr != wptr)
            {
                Uart3SEND(buffer[rptr++]);
                rptr &= 0x0f;
            }
```

STC MCU

```
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H
S3CON       DATA    0ACH
S3BUF       DATA    0ADH
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     008BH
            LJMP    UART3_ISR

            ORG     0100H

UART3_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S3CON
            JNB     ACC.1,CHKRI
            ANL     S3CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART3ISR_EXIT
            ANL     S3CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
```

STC MCU

```
                MOV     @R0,S3BUF
                INC     WPTR
UART3ISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

UART3_INIT:
                MOV     S3CON,#50H
                MOV     T3L,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     T3H,#0FFH
                MOV     T4T3M,#0AH
                CLR     BUSY
                MOV     WPTR,#00H
                MOV     RPTR,#00H
                RET

UART3_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     S3BUF,A
                RET

UART3_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SEND3END
                LCALL   UART3_SEND
                INC     DPTR
                JMP     UART3_SENDSTR
SEND3END:
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART3_INIT
                MOV     IE2,#08H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART3_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
```

STC MCU

```
        JZ      LOOP
        MOV     A,RPTR
        ANL     A,#0FH
        ADD     A,#BUFFER
        MOV     R0,A
        MOV     A,@R0
        LCALL   UART3_SEND
        INC     RPTR
        JMP     LOOP

STRING: DB      'Uart Test !',0DH,0AH,00H

        END
```

13.7.7 UART4 using T2 as baud rate generator

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define FOSC 11059200UL
#define BRT (65536 - FOSC / 115200 / 4)

sfr AUXR = 0x8e;
sfr T2H = 0xd6;
sfr T2L = 0xd7;
sfr S4CON = 0x84;
sfr S4BUF = 0x85;
sfr IE2 = 0xaf;

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

bit busy;
char wptr;
char rptr;
char buffer[16];

void Uart4Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
```

STC MCU

```
            busy = 0;
        }
        if (S4CON & 0x01)
        {
            S4CON &= ~0x01;
            buffer[wptr++] = S4BUF;
            wptr &= 0x0f;
        }
}

void Uart4Init()
{
        S4CON = 0x10;
        T2L = BRT;
        T2H = BRT >> 8;
        AUXR = 0x14;
        wptr = 0x00;
        rptr = 0x00;
        busy = 0;
}

void Uart4Send(char dat)
{
        while (busy);
        busy = 1;
        S4BUF = dat;
}

void Uart4SendStr(char *p)
{
        while (*p)
        {
            Uart4SEND(*p++);
        }
}

void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        Uart4Init();
        IE2 = 0x10;
        EA = 1;
        Uart4SENDStr("Uart Test !\r\n");

        while (1)
        {
            if (rptr != wptr)
```

STC MCU

```
        {
            Uart4SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR    DATA    8EH
T2H     DATA    0D6H
T2L     DATA    0D7H
S4CON   DATA    84H
S4BUF   DATA    85H
IE2     DATA    0AFH

BUSY    BIT     20H.0
WPTR    DATA    21H
RPTR    DATA    22H
BUFFER  DATA    23H                 ;16 bytes

P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
        ORG     0093H
        LJMP    UART4_ISR

        ORG     0100H

UART4_ISR:
        PUSH    ACC
        PUSH    PSW
        MOV     PSW,#08H

        MOV     A,S4CON
        JNB     ACC.1,CHKRI
        ANL     S4CON,#NOT 02H
        CLR     BUSY
CHKRI:
        JNB     ACC.0,UART4ISR_EXIT
        ANL     S4CON,#NOT 01H
        MOV     A,WPTR
        ANL     A,#0FH
        ADD     A,#BUFFER
        MOV     R0,A
```

STC MCU

```
                MOV     @R0,S4BUF
                INC     WPTR
UART4ISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

UART4_INIT:
                MOV     S4CON,#10H
                MOV     T2L,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     T2H,#0FFH
                MOV     AUXR,#14H
                CLR     BUSY
                MOV     WPTR,#00H
                MOV     RPTR,#00H
                RET

UART4_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     S4BUF,A
                RET

UART4_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SEND4END
                LCALL   UART4_SEND
                INC     DPTR
                JMP     UART4_SENDSTR
SEND4END:
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART4_INIT
                MOV     IE2,#10H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART4_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
```

STC MCU

```
            JZ          LOOP
            MOV         A,RPTR
            ANL         A,#0FH
            ADD         A,#BUFFER
            MOV         R0,A
            MOV         A,@R0
            LCALL       UART4_SEND
            INC         RPTR
            JMP         LOOP

STRING:     DB          'Uart Test !',0DH,0AH,00H

            END
```

13.7.8 UART4 using T4 as baud rate generator

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       T4T3M   =   0xd1;
sfr       T4L     =   0xd3;
sfr       T4H     =   0xd2;
sfr       T3L     =   0xd5;
sfr       T3H     =   0xd4;
sfr       T2L     =   0xd7;
sfr       T2H     =   0xd6;
sfr       S4CON   =   0x84;
sfr       S4BUF   =   0x85;
sfr       IE2     =   0xaf;

sfr       P0M1    =   0x93;
sfr       P0M0    =   0x94;
sfr       P1M1    =   0x91;
sfr       P1M0    =   0x92;
sfr       P2M1    =   0x95;
sfr       P2M0    =   0x96;
sfr       P3M1    =   0xb1;
sfr       P3M0    =   0xb2;
sfr       P4M1    =   0xb3;
sfr       P4M0    =   0xb4;
sfr       P5M1    =   0xc9;
sfr       P5M0    =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart4Isr() interrupt 18
```

STC MCU

```
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}

void Uart4Init()
{
    S4CON = 0x50;
    T4L = BRT;
    T4H = BRT >> 8;
    T4T3M = 0xa0;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart4Send(char dat)
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SENDStr("Uart Test !\r\n");
```

STC MCU

```
    while (1)
    {
        if (rptr != wptr)
        {
            Uart4SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H
S4CON       DATA    84H
S4BUF       DATA    85H
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0093H
            LJMP    UART4_ISR

            ORG     0100H

UART4_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S4CON
            JNB     ACC.1,CHKRI
            ANL     S4CON,#NOT 02H
```

```
                CLR     BUSY
CHKRI:
                JNB     ACC.0,UART4ISR_EXIT
                ANL     S4CON,#NOT 01H
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,S4BUF
                INC     WPTR
UART4ISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

UART4_INIT:
                MOV     S4CON,#50H
                MOV     T4L,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     T4H,#0FFH
                MOV     T4T3M,#0A0H
                CLR     BUSY
                MOV     WPTR,#00H
                MOV     RPTR,#00H
                RET

UART4_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     S4BUF,A
                RET

UART4_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SEND4END
                LCALL   UART4_SEND
                INC     DPTR
                JMP     UART4_SENDSTR
SEND4END:
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART4_INIT
                MOV     IE2,#10H
                SETB    EA
```

STC MCU

```
                MOV         DPTR,#STRING
                LCALL       UART4_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART4_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

13.7.9 Serial port multi-machine communication

Now refer to STC15 series data manual, follow-up supplement.

13.7.10 UART to LIN bus

C language code

```
//Operating frequency for test is 22.1184MHz

/*************  Function Description   ***************
This routine is based on STC8H8K64U as the main control chip in experiment box 8 for coding and testing. It can be used as general reference for STC8G and STC8H series chips.
Connect the LIN transceiver through the UART interface to realize the LIN bus signal transceiver test routine.
UART1 is connected to the computer through a serial port tool.
UART2 is connected to the LIN transceiver (TJA1020/1) and then is connected to the LIN bus.
The data sent from the computer serial port is forwarded to the LIN bus, and the data received from the LIN bus is forwarded to the computer serial port.
The default transmission rate is 9600 bps. The routine switches the baud rate before sending LIN data, and sends 13 dominant interval signals.
Please select the clock 22.1184MHz when downloading user program (users can modify the frequency by themselves).
*****************************************/

#include "reg51.h"
#include "intrins.h"

#define    MAIN_Fosc        22118400L

typedef    unsigned char    u8;
typedef    unsigned int     u16;
typedef    unsigned long    u32;
```

```
sfr     AUXR    =   0x8E;
sfr     S2CON   =   0x9A;
sfr     S2BUF   =   0x9B;
sfr     TH2     =   0xD6;
sfr     TL2     =   0xD7;
sfr     IE2     =   0xAF;
sfr     INT_CLKO =  0x8F;
sfr     P_SW1   =   0xA2;
sfr     P_SW2   =   0xBA;

sfr     P4      =   0xC0;
sfr     P5      =   0xC8;
sfr     P6      =   0xE8;
sfr     P7      =   0xF8;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xB1;
sfr     P3M0    =   0xB2;
sfr     P4M1    =   0xB3;
sfr     P4M0    =   0xB4;
sfr     P5M1    =   0xC9;
sfr     P5M0    =   0xCA;
sfr     P6M1    =   0xCB;
sfr     P6M0    =   0xCC;
sfr     P7M1    =   0xE1;
sfr     P7M0    =   0xE2;

sbit    P00     =   P0^0;
sbit    P01     =   P0^1;
sbit    P02     =   P0^2;
sbit    P03     =   P0^3;
sbit    P04     =   P0^4;
sbit    P05     =   P0^5;
sbit    P06     =   P0^6;
sbit    P07     =   P0^7;
sbit    P10     =   P1^0;
sbit    P11     =   P1^1;
sbit    P12     =   P1^2;
sbit    P13     =   P1^3;
sbit    P14     =   P1^4;
sbit    P15     =   P1^5;
sbit    P16     =   P1^6;
sbit    P17     =   P1^7;
sbit    P20     =   P2^0;
sbit    P21     =   P2^1;
sbit    P22     =   P2^2;
sbit    P23     =   P2^3;
sbit    P24     =   P2^4;
sbit    P25     =   P2^5;
sbit    P26     =   P2^6;
sbit    P27     =   P2^7;
sbit    P30     =   P3^0;
sbit    P31     =   P3^1;
sbit    P32     =   P3^2;
```

STC MCU

```
sbit    P33     =   P3^3;
sbit    P34     =   P3^4;
sbit    P35     =   P3^5;
sbit    P36     =   P3^6;
sbit    P37     =   P3^7;
sbit    P40     =   P4^0;
sbit    P41     =   P4^1;
sbit    P42     =   P4^2;
sbit    P43     =   P4^3;
sbit    P44     =   P4^4;
sbit    P45     =   P4^5;
sbit    P46     =   P4^6;
sbit    P47     =   P4^7;
sbit    P50     =   P5^0;
sbit    P51     =   P5^1;
sbit    P52     =   P5^2;
sbit    P53     =   P5^3;
sbit    P54     =   P5^4;
sbit    P55     =   P5^5;
sbit    P56     =   P5^6;
sbit    P57     =   P5^7;

sbit    SLP_N   =   P2^4;                   //0: Sleep

/******************************** User-defined macro ***********************************/

#define Baudrate1           (65536UL - (MAIN_Fosc /   4) / 9600UL)
#define Baudrate2           (65536UL - (MAIN_Fosc /   4) / 9600UL)

#define Baudrate_Break      (65536UL - (MAIN_Fosc /   4) / 6647UL)    // Baud rate for sending dominant interval signal

#define UART1_BUF_LENGTH    32
#define UART2_BUF_LENGTH    32

#define LIN_ID          0x31

u8 TX1_Cnt;                                 // Sending count
u8 RX1_Cnt;                                 // Receiving count
u8 TX2_Cnt;                                 // Sending count
u8 RX2_Cnt;                                 // Receiving count
bit B_TX1_Busy;                             // Busy flag of sending
bit B_TX2_Busy;                             // Busy flag of sending
u8 RX1_TimeOut;
u8 RX2_TimeOut;

u8 xdata RX1_Buffer[UART1_BUF_LENGTH];      // Buffer of Receiving
u8 xdata RX2_Buffer[UART2_BUF_LENGTH];      // Buffer of Receiving

void UART1_config(u8 brt);
void UART2_config(u8 brt);
void PrintString1(u8 *puts);
void delay_ms(u8 ms);
void UART1_TxByte(u8 dat);
void UART2_TxByte(u8 dat);
void Lin_Send(u8 *puts);
void SetTimer2Baudraye(u16 dat);

//=========================================================================
```

```
// Function: void main(void)
// Description: main function
// Parameters: none
// Return: none.
// Version: VER1.0
// Date: 2014-11-28
// Note:
//=========================================================================
void main(void)
{
    u8 i;

    P0M1 = 0; P0M0 = 0;                     // Set as a quasi-bidirectional port
    P1M1 = 0; P1M0 = 0;                     // Set as a quasi-bidirectional port
    P2M1 = 0; P2M0 = 0;                     // Set as a quasi-bidirectional port
    P3M1 = 0; P3M0 = 0;                     // Set as a quasi-bidirectional port
    P4M1 = 0; P4M0 = 0;                     // Set as a quasi-bidirectional port
    P5M1 = 0; P5M0 = 0;                     // Set as a quasi-bidirectional port
    P6M1 = 0; P6M0 = 0;                     // Set as a quasi-bidirectional port
    P7M1 = 0; P7M0 = 0;                     // Set as a quasi-bidirectional port

    UART1_config(1);
    UART2_config(2);
    EA = 1;                                 //Enable global interrupt
    SLP_N =  1;

    PrintString1("STC8H8K64U UART1 Test Programme!\r\n"); //UART1 sends a string

    while (1)
    {
        delay_ms(1);
        if(RX1_TimeOut > 0)
        {
            if(--RX1_TimeOut == 0)          //UART ends receiving if it is time out
            {
                if(RX1_Cnt > 0)
                {
                    Lin_Send(RX1_Buffer);   // Send the data received by UART1 to the LIN bus
                }
                RX1_Cnt = 0;
            }
        }

        if(RX2_TimeOut > 0)
        {
            if(--RX2_TimeOut == 0)          // UART ends receiving if it is time out
            {
                if(RX2_Cnt > 0)
                {
                    for (i=0; I < RX2_Cnt; i++)     // End with stop character 0
                    {
                        UART1_TxByte(RX2_Buffer[i]); // The data received from the LIN bus is sent to UART1
                    }
                }
                RX2_Cnt = 0;
            }
        }
    }
}
```

```
//========================================================================
// Function: void delay_ms(unsigned     char ms)
// Description: delay function
// Parameters: ms, number of msto delay, 1~255ms. Automatically adapt to the master clock.
// Return: none.
// Version: VER1.0
// Date: 2013-4-1
// Note:
//========================================================================
void delay_ms(u8 ms)
{
    u16 i;
    do{
        i = MAIN_Fosc / 13000;
        while(--i);                                     //14T per loop
    }while(--ms);
}

//========================================================================
// Function: u8                 Lin_CheckPID(u8      id)
// Description: ID code plus check code, converted into PID code
// Parameters: ID code
// Return: PID code
// Version: VER1.0
// Date: 2020-12-2
// Note:
//========================================================================
u8 Lin_CheckPID(u8 id)
{
    u8 returnpid ;
    u8 P0 ;
    u8 P1 ;

    P0 = (((id)^(id>>1)^(id>>2)^(id>>4))&0x01)<<6;
    P1 = ((~((id>>1)^(id>>3)^(id>>4)^(id>>5)))&0x01)<<7;

    returnpid = id|P0|P1 ;

    return returnpid ;
}

//========================================================================
// Function: u8 LINCalcChecksum(u8 *dat)
// Description: calculate check code
// Parameters: data transmitted in data field
// Return: check code.
// Version: VER1.0
// Date: 2020-12-2
// Note:
//========================================================================
static u8 LINCalcChecksum(u8 *dat)
{
    u16 sum = 0;
    u8 i;

    for(I = 0; i < 8; i++)
    {
    sum += dat[i];
```

```
    if(sum & 0xFF00)
    {
        sum = (sum & 0x00FF) + 1;
    }
    }
    sum ^= 0x00FF;
    return (u8)sum;
}

//==========================================================================
// Function: void Lin_SendBreak(void)
// Description: Send a dominant interval signal
// Parameters: none.
// Return: none.
// Version: VER1.0
// Date: 2020-12-2
// Note:
//==========================================================================
void Lin_SendBreak(void)
{
    SetTimer2Baudraye(Baudrate_Break);
    UART2_TxByte(0);
    SetTimer2Baudraye(Baudrate2);
}

//==========================================================================
// Function: void Lin_Send(u8 *puts)
// Description: Send LIN bus message
// Parameters: The content of the data field to be sent
// Return: none.
// Version: VER1.0
// Date: 2020-12-2
// Note:
//==========================================================================
void Lin_Send(u8 *puts)
{
    u8 i;

    Lin_SendBreak();                                //Break
    UART2_TxByte(0x55);                             //SYNC
    UART2_TxByte(Lin_CheckPID(LIN_ID));             //LIN ID
    for(i=0;i<8;i++)
    {
        UART2_TxByte(puts[i]);
    }
    UART2_TxByte(LINCalcChecksum(puts));
}

//==========================================================================
// Function: void UART1_TxByte(u8 dat)
// Description: Send a byte
// Parameters: none
// Return: none
// Version: V1.0, 2014-6-30
//==========================================================================
void UART1_TxByte(u8 dat)
{
    SBUF = dat;
    B_TX1_Busy = 1;
```

STC MCU

```
    while(B_TX1_Busy);
}

//========================================================================
// Function: void UART2_TxByte(u8 dat)
// Description: Send a byte
// Parameters: none
// Return: none
// Version: V1.0, 2014-6-30
//========================================================================
void UART2_TxByte(u8 dat)
{
    S2BUF = dat;
    B_TX2_Busy = 1;
    while(B_TX2_Busy);
}

//========================================================================
// Function: void PrintString1(u8 *puts)
// Description: Send string using UART1
// Parameters: puts: pointer of string
// Return: none.
// Version: VER1.0
// Date: 2014-11-28
// Note:
//========================================================================
void PrintString1(u8 *puts)
{
    for (; *puts != 0; puts++)                  // End with stop character 0
    {
        SBUF = *puts;
        B_TX1_Busy = 1;
        while(B_TX1_Busy);
    }
}

//========================================================================
// Function: void PrintString1(u8 *puts)
// Description: Send string using UART2
// Parameters: puts: pointer of string
// Return: none.
// Version: VER1.0
// Date: 2014-11-28
// Note:
//========================================================================
//void PrintString2(u8 *puts)
//{
//    for (; *puts != 0;  puts++)                   // End with stop character 0
//    {
//        S2BUF = *puts;
//        B_TX2_Busy = 1;
//        while(B_TX2_Busy);
//    }
//}

//========================================================================
// Function: SetTimer2Baudraye(u16 dat)
// Description: Set Timer2 as a baud rate generator.
// Parameters: dat: Reload value of Timer2
```

STC MCU

```
// Return: none.
// Version: VER1.0
// Date: 2014-11-28
// Note:
//==========================================================================
void SetTimer2Baudraye(u16 dat)
{
	AUXR &= ~(1<<4);		//Timer stop
	AUXR &= ~(1<<3);		//Timer2 set As Timer
	AUXR |= (1<<2);		//Timer2 set as 1T mode
	TH2  = dat / 256;
	TL2 = dat % 256;
	IE2 &= ~(1<<2);		//Disable interrupt
	AUXR |= (1<<4);		//Timer run enable
}

//==========================================================================
// Function: void UART1_config(u8 brt)
// Description: UART1 initial function
// Parameters: brt: Select baud rate, 2: Use Timer2 as baud rate generator, Other values: Use Timer1 as the baud rate generator.
// Return: none.
// Version: VER1.0
// Date: 2014-11-28
// Note:
//==========================================================================
void UART1_config(u8 brt)
{
	/********** Use Timer2 as baud rate generator *****************/
	if(brt == 2)
	{
		AUXR |= 0x01;		//S1 BRT Use Timer2;
		SetTimer2Baudraye(Baudrate1);
	}

	/********** Use Timer1 as the baud rate generator *****************/
	else
	{
		TR1  = 0;
		AUXR &= ~0x01;		//S1 BRT Use Timer1;
		AUXR |= (1<<6);		//Timer1 set as	1T mode
		TMOD &= ~(1<<6);		//Timer1 set As	Timer
		TMOD &= ~0x30;		//Timer1_16bitAutoReload;
		TH1 = (u8)(Baudrate1 / 256);
		TL1 = (u8)(Baudrate1 % 256);
		ET1 = 0;		//Disable interrupt
		INT_CLKO &= ~0x02;		// No clock output
		TR1 = 1;
	}
	/*************************************************/

	SCON = (SCON & 0x3f) | 0x40;		//UART1 mode:  0x00: Synchronous shift output,
	//			0x40: 8 bits data, Variable baud rate,
	//			0x80: 9 bits data, Fixed baud rate,
	//			0xc0: 9 bits data, Variable baud rate
//	PS    = 1;		// High priority interrupt
	ES    = 1;		//Enable interrupt
	REN = 1;		//Enable recieving
	P_SW1 &= 0x3f;
//	P_SW1 |= 0x80;		//UART1switch to:	0x00: P3.0 P3.1,
```

```
//                      0x40: P3.6 P3.7,
//                      0x80: P1.6 P1.7,
//                      0xC0: P4.3 P4.4

    B_TX1_Busy = 0;
    TX1_Cnt = 0;
    RX1_Cnt = 0;
}

//========================================================================
// Function: void UART2_config(u8 brt)
// Description: UART2 initial function
// Parameters: brt: Select baud rate, 2: Use Timer2 as baud rate generator, Other values: invalid
// Return: none.
// Version: VER1.0
// Date: 2014-11-28
// Note:
//========================================================================
void UART2_config(u8 brt)
{
    if(brt == 2)
    {
        SetTimer2Baudraye(Baudrate2);

        S2CON &= ~(1<<7);                   //8 bits data, 1 start bit, 1 stop bit,  no check bit
        IE2 |= 1;                           //Enable interrupt
        S2CON |= (1<<4);                    //Enable recieving
        P_SW2 &= ~0x01;
//      P_SW2 |= 1;                         //UART2 switch to: 0: P1.0/P1.1, 1: P4.6/P4.7

        B_TX2_Busy = 0;
        TX2_Cnt = 0;
        RX2_Cnt = 0;
    }
}

//========================================================================
// Function: void UART1_int (void) interrupt UART1_VECTOR
// Description: UART1 interrupt function
// Parameters: none.
// Return: none.
// Version: VER1.0
// Date: 2014-11-28
// Note:
//========================================================================
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;
```

STC MCU

```
            B_TX1_Busy = 0;
        }
}

//========================================================================
// Function: void UART2_int        (void) interrupt UART2_VECTOR
// Description: UART2 interrupt function
// Parameters: none.
// Return: none.
// Version: VER1.0
// Date: 2014-11-28
// Note:
//========================================================================
void UART2_int (void) interrupt   8
{
        if((S2CON & 1) != 0)
        {
            S2CON &= ~1;                                        //Clear Rx flag
            if(RX2_Cnt >= UART2_BUF_LENGTH) RX2_Cnt = 0;
            RX2_Buffer[RX2_Cnt] = S2BUF;
            RX2_Cnt++;
            RX2_TimeOut = 5;
        }

        if((S2CON &     2) != 0)
        {
            S2CON &= ~2;                                        //Clear Tx flag
            B_TX2_Busy = 0;
        }
}
```

STC MCU

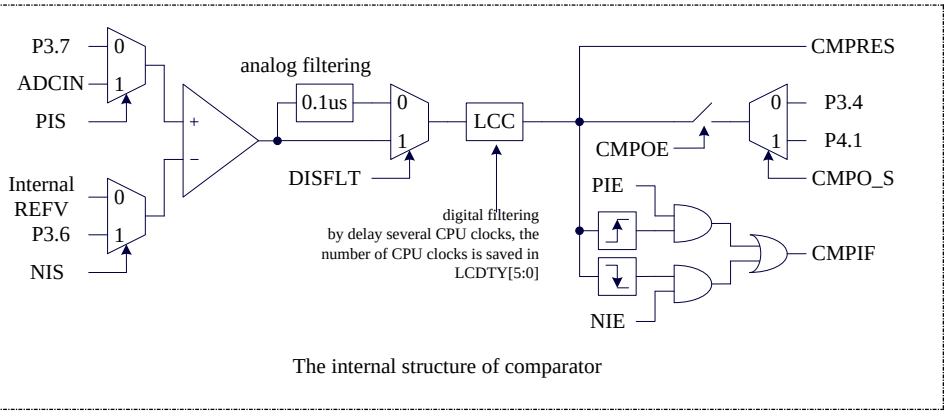
14 Comparator, Power-down Detection, Internal Reference Voltage

Production line	Comparator
STC8G1K08 family	●
STC8G1K08-8Pin family	
STC8G1K08A family	
STC8G2K64S4 family	●
STC8G2K64S2 family	●
STC8G1K08T family	●
STC15H2K64S4 family	●

A comparator is integrated in STC8G series of microcontrollers. The positive terminal of the comparator can be P3.7 or ADC analog input and the negative can be P3.6 or the REFV voltage of the internal BandGap after amplified. The application of multiple comparators can be realized through multiplexer and time division multiplexing.

There are two stages programmable filterings inside the comparator: analog filtering and digital filtering. Analog filtering can filter out glitches in the input signal, and digital filtering can wait for the input signal to stabilize before making a comparison. The result of the comparison can be obtained directly by reading the internal register bits or output the result of the comparator forward or reverse to the external port. Outputting the comparison result to the external port can be used as the trigger signal of external events and the feedback signal to expand the scope of application.

14.1 Internal Structure of Comparator



The internal structure of comparator

ADCIN includes ADC0, ADC1, ADC2, ADC3, ADC4, ADC5, ADC6, ADC7, ADC8, ADC9, ADC10, ADC11, ADC12, ADC13, ADC14 and ADC15.

Note: When the positive pole of the comparator selects the ADC input channel, make sure to turn on the ADC power control bit ADC_POWER and ADC channel selection bit ADC_CHS in the ADC_CONTR register.

14.2 Registers Related to Comparator

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
CMPCR1	Comparator Control Register 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
CMPCR2	Comparator Control Register 2	E7H	INVCMPO	DISFLT	LCDTY[5:0]						0000,0000

14.2.1 Comparator control register 1 (CMPCR1)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES

CMPEN: Comparator enable bit

0: disable comparator

1: enable comparator

CMPIF: Comparator interrupt flag. When PIE or NIE is enabled, if the corresponding interrupt signal is generated, the hardware will automatically set CMPIF and request interrupt to CPU. This flag must be cleared by software. **(Note: When the comparator interrupt is not enabled, this interrupt flag will not be set by the hardware, that is, this interrupt flag cannot be queried when the comparator is accessed in query mode.)**

PIE: Comparator rising edge interrupt enable bit

0: disable comparator rising edge interrupt

1: enable comparator rising edge interrupt. Enable the interrupt request when the compare result of the comparator changes from 0 to 1.

NIE: Comparator falling edge interrupt enable bit

0: disable comparator falling edge interrupt

1: enable comparator falling edge interrupt. Enable the interrupt request when the compare result of the comparator changes from 1 to 0.

PIS: Positive pole selection bit of comparator

0: Select P3.7 as the comparator positive input source.

1: The analog input of the ADC selected by the ADC_CHS bits in ADC_CONTR is selected as the comparator positive input source.

(Note 1: When the positive pole of the comparator selects the ADC input channel, make sure to turn on the ADC power control bit ADC_POWER and ADC channel selection bit ADC_CHS in the ADC_CONTR register.)

(Note 2: When it is necessary to use the comparator interrupt to wake up the power-down mode/clock stop mode, the positive pole of the comparator must select P3.7, and the ADC input channel cannot be used.)

NIS: Negative pole selection bit of comparator

0: The REFV voltage of the internal BandGap after amplified is selected as the comparator negative input source. (When the chip is shipped, the internal reference voltage is adjusted to 1.19V)

1: Select P3.6 as the comparator negative input source.

CMPOE: Comparator result output control bit

0: disable comparator result output.

1: enable comparator result output. The comparator result is output to P3.4 or P4.1, which is selected by CMPO_S in P_SW2.

CMPRES: Flag bit of comparator result. (Read-only)

0: the level of CMP+ is lower than CMP-.

1: the level of CMP+ is higher than CMP-.

CMPRES is the digitally filtered output signal, not the comparator's direct output.

14.2.2 Comparator control register 2 (CMPCR2)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR2	E7H	INVCMPO	DISFLT	LCDTY[5:0]					

INVCMPO: Inverse comparator output control bit

0: Normal output the result of comparator. If CMPRES is 0, P3.4 / P4.1 output low, and vice versa output high.

1: Output the result of comparator after it is inversed. If CMPRES is 0, P3.4 / P4.1 output high, and vice versa output low.

DISFLT: Analog filtering function control bit

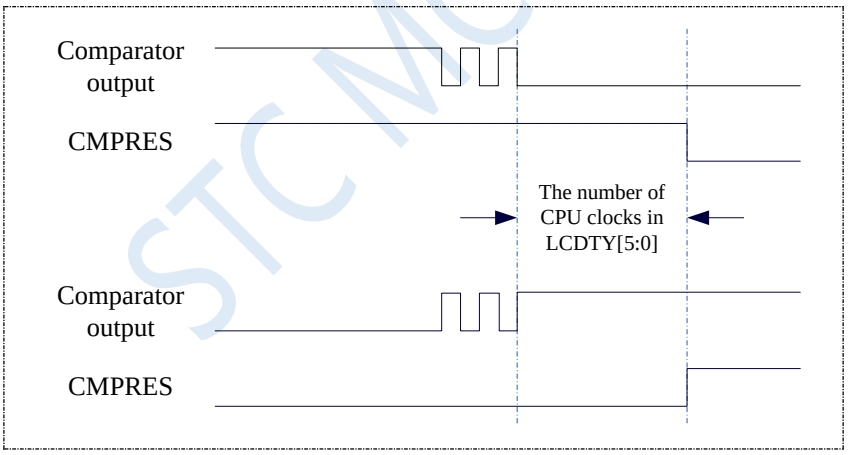
0: enable 0.1us analog filtering function

1: disable 0.1us analog filtering function, which can speed up the comparator slightly.

LCDTY[5:0]: Digital filtering function control bit

Digital filtering is the debouncing function of the digital signal. When the comparison result changes at the rising edge or falling edge, the data changing is considered be valid only if the signal the comparator detected does not change and maintains the number of CPU clocks set in LCDTY. Otherwise, the signal will be treated as no change.

Note: When the digital filter function is enabled, the actual wait clock inside the chip needs to add two additional state machine switches time, that is, if LCDTY is set to 0, the digital filter function is turned off. And if LCDTY is set to a non-zero value n (n =1~63), the actual digital filtering time is (n+2) system clocks.



STC MCU

14.3 Example Routines

14.3.1 Using Comparator (Interrupt Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     CMPCR1  =   0xe6;
sfr     CMPCR2  =   0xe7;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;

void CMP_Isr() interrupt 21
{
    CMPCR1 &= ~0x40;                    //Clear interrupt flag
    if (CMPCR1 & 0x01)
    {
        P10 = !P10;                     //Falling edge interrupt test port
    }
    else
    {
        P11 = !P11;                     //Rising edge interrupt test port
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
```

STC MCU

```
    P5M1 = 0x00;

    CMPCR2 = 0x00;
    CMPCR2 &= ~0x80;                    //Comparator forward output
//  CMPCR2 |= 0x80;                     //Comparator inverted output
    CMPCR2 &= ~0x40;                    //Disable 0.1us filtering
//  CMPCR2 |= 0x40;                     //Enable 0.1us filtering
//  CMPCR2 &= ~0x3f;                    //Output comparator result directly
    CMPCR2 |= 0x10;                     //Output comparator result after 16 debounce clocks
    CMPCR1 = 0x00;
    CMPCR1 |= 0x30;                     //Enable edge interrupt of comparator
//  CMPCR1 &= ~0x20;                    //Disable comparator rising edge interrupt
//  CMPCR1 |= 0x20;                     //Enable comparator rising edge interrupt
//  CMPCR1 &= ~0x10;                    //Disable comparator falling edge interrupt
//  CMPCR1 |= 0x10;                     //Enable comparator falling edge interrupt
    CMPCR1 &= ~0x08;                    //P3.7 is CMP+ input pin
//  CMPCR1 |= 0x08;                     //ADC input pin is CMP+ input pin
//  CMPCR1 &= ~0x04;                    //Internal reference voltage is CMP- input pin
    CMPCR1 |= 0x04;                     //P3.6 is CMP- input pin
//  CMPCR1 &= ~0x02;                    //Disable comparator output
    CMPCR1 |= 0x02;                     //Enable Comparator output
    CMPCR1 |= 0x80;                     //Enable comparator module

    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00ABH
            LJMP    CMPISR

            ORG     0100H
CMPISR:
            PUSH    ACC
            ANL     CMPCR1,#NOT 40H     ;Clear interrupt flag
            MOV     A,CMPCR1
            JB      ACC.0,RSING
```

```
FALLING:
        CPL     P1.0            ;Falling edge interrupt test port
        POP     ACC
        RETI
RSING:
        CPL     P1.1            ;Rising edge interrupt test port
        POP     ACC
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     CMPCR2,#00H
        ANL     CMPCR2,#NOT 80H ;Comparator forward output
;       ORL     CMPCR2,#80H     ;Comparator inverted output
        ANL     CMPCR2,#NOT 40H ;Disable 0.1us filtering
;       ORL     CMPCR2,#40H     ;Enable 0.1us filtering
;       ANL     CMPCR2,#NOT 3FH ;Output comparator result directly
        ORL     CMPCR2,#10H     ;Output comparator result after 16 debounce clocks
        MOV     CMPCR1,#00H
        ORL     CMPCR1,#30H     ;Enable edge interrupt of comparator
;       ANL     CMPCR1,#NOT 20H ;Disable comparator rising edge interrupt
;       ORL     CMPCR1,#20H     ;Enable comparator rising edge interrupt
;       ANL     CMPCR1,#NOT 10H ;Disable comparator falling edge interrupt
;       ORL     CMPCR1,#10H     ;Enable comparator falling edge interrupt
        ANL     CMPCR1,#NOT 08H ;P3.7 is CMP+ input pin
;       ORL     CMPCR1,#08H     ;ADC input pin is CMP+ input pin
;       ANL     CMPCR1,#NOT 04H ;Internal reference voltage is CMP- input pin
        ORL     CMPCR1,#04H     ;P3.6 is CMP- input pin
;       ANL     CMPCR1,#NOT 02H ;Disable comparator output
        ORL     CMPCR1,#02H     ;Enable Comparator output
        ORL     CMPCR1,#80H     ;Enable comparator module
        SETB    EA

        JMP     $

        END
```

14.3.2 Using Comparator (Polling Mode)

C language code

```
//Operating frequency for test is 11.0592MHz
```

```
#include "reg51.h"
#include "intrins.h"

sfr     CMPCR1      =   0xe6;
sfr     CMPCR2      =   0xe7;

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

sbit    P10         =   P1^0;
sbit    P11         =   P1^1;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CMPCR2 = 0x00;
    CMPCR2 &= ~0x80;                    //Comparator forward output
//  CMPCR2 |= 0x80;                     //Comparator inverted output
    CMPCR2 &= ~0x40;                    //Disable 0.1us filtering
//  CMPCR2 |= 0x40;                     //Enable 0.1us filtering
//  CMPCR2 &= ~0x3f;                    //Output comparator result directly
    CMPCR2 |= 0x10;                     //Output comparator result after 16 debounce clocks
    CMPCR1 = 0x00;
    CMPCR1 |= 0x30;                     //Enable edge interrupt of comparator
//  CMPCR1 &= ~0x20;                    //Disable comparator rising edge interrupt
//  CMPCR1 |= 0x20;                     //Enable comparator rising edge interrupt
//  CMPCR1 &= ~0x10;                    //Disable comparator falling edge interrupt
//  CMPCR1 |= 0x10;                     //Enable comparator falling edge interrupt
    CMPCR1 &= ~0x08;                    //P3.7 is CMP+ input pin
//  CMPCR1 |= 0x08;                     //ADC input pin is CMP+ input pin
//  CMPCR1 &= ~0x04;                    //Internal reference voltage is CMP- input pin
    CMPCR1 |= 0x04;                     //P3.6 is CMP- input pin
//  CMPCR1 &= ~0x02;                    //Disable comparator output
    CMPCR1 |= 0x02;                     //Enable Comparator output
    CMPCR1 |= 0x80;                     //Enable comparator module

    while (1)
```

STC MCU

```
    {
        P10 = CMPCR1 & 0x01;                    //Read comparator comparison result
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     CMPCR2,#00H
            ANL     CMPCR2,#NOT 80H     ;Comparator forward output
;           ORL     CMPCR2,#80H         ;Comparator inverted output
            ANL     CMPCR2,#NOT 40H     ;Disable 0.1us filtering
;           ORL     CMPCR2,#40H         ;Enable 0.1us filtering
;           ANL     CMPCR2,#NOT 3FH     ;Output comparator result directly
            ORL     CMPCR2,#10H         ;Output comparator result after 16 debounce clocks
            MOV     CMPCR1,#00H
            ORL     CMPCR1,#30H         ;Enable edge interrupt of comparator
;           ANL     CMPCR1,#NOT 20H     ;Disable comparator rising edge interrupt
;           ORL     CMPCR1,#20H         ;Enable comparator rising edge interrupt
;           ANL     CMPCR1,#NOT 10H     ;Disable comparator falling edge interrupt
;           ORL     CMPCR1,#10H         ;Enable comparator falling edge interrupt
            ANL     CMPCR1,#NOT 08H     ;P3.7 is CMP+ input pin
;           ORL     CMPCR1,#08H         ;ADC input pin is CMP+ input pin
```

```
;               ANL     CMPCR1,#NOT 04H     ;Internal reference voltage is CMP- input pin
                ORL     CMPCR1,#04H         ;P3.6 is CMP- input pin
;               ANL     CMPCR1,#NOT 02H     ;Disable comparator output
                ORL     CMPCR1,#02H         ;Enable Comparator output
                ORL     CMPCR1,#80H         ;Enable comparator module

LOOP:
                MOV     A,CMPCR1
                MOV     C,ACC.0
                MOV     P1.0,C              ;Read comparator comparison result
                JMP     LOOP

                END
```

14.3.3 Multiplexing application of comparator (comparator + ADC input channel)

Since the positive pole of the comparator can select the analog input channel of the ADC, the application of multiple comparators can be realized through the multiplexer and time division multiplexing.

Note: When the positive pole of the comparator selects the ADC input channel, make sure to turn on the ADC power control bit ADC_POWER and ADC channel selection bit ADC_CHS in the ADC_CONTR register

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     CMPCR1  =   0xe6;
sfr     CMPCR2  =   0xe7;

sfr     ADC_CONTR   =   0xbc;

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
```

```
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P1M0 &= 0xfe;                   // Set P1.0 as input port
    P1M1 |= 0x01;
    ADC_CONTR = 0x80;               // Enable ADC module and select P1.0 as ADC input pin

    CMPCR2 = 0x00;
    CMPCR1 = 0x00;

    CMPCR1 |= 0x08;                 // Set ADC input pin as CMP+ input pin
    CMPCR1 |= 0x04;                 //Set P3.6 as CMP- input pin
    CMPCR1 |= 0x02;                 // Enable comparator output
    CMPCR1 |= 0x80;                 // Enable the comparator module

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H
ADC_CONTR   DATA    0BCH

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
```

STC MCU

```
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        ANL     P1M0,#0FEH          ; Set P1.0 as input port
        ORL     P1M1,#01H
        MOV     ADC_CONTR,#80H      ; Enable ADC module and select P1.0 as ADC input pin

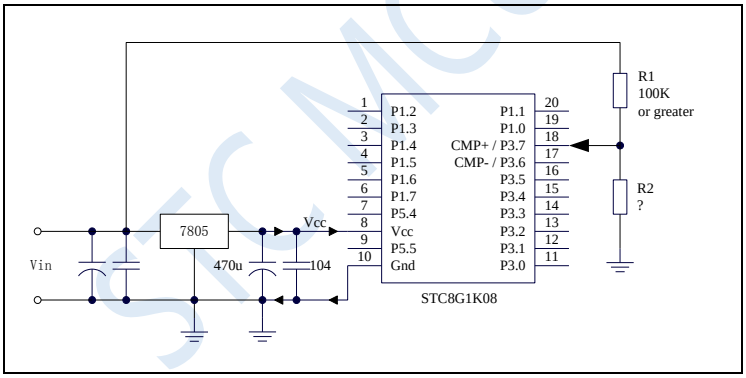
        MOV     CMPCR2,#00H
        MOV     CMPCR1,#00H

        ORL     CMPCR1,#08H         ; Set ADC input pin as CMP+ input pin
        ORL     CMPCR1,#04H         ; Set P3.6 as CMP- input pin
        ORL     CMPCR1,#02H         ; Enable comparator output
        ORL     CMPCR1,#80H         ; Enable the comparator module

LOOP:
        JMP     LOOP

        END
```

14.3.4 Comparator is Used for External Power-down Detection



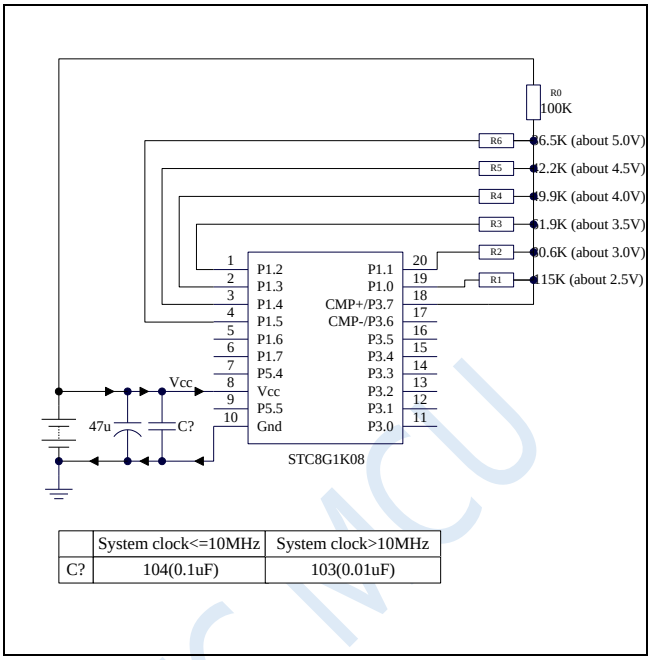
In the figure above, the resistor R1 and R2 divide the front-end voltage of the voltage regulator 7805. The divided voltage is used as the external input of the comparator CMP+ to compare with the internal reference voltage.

When the AC power is at 220V, the DC voltage at the front end of the voltage regulator block 7805 is 11V, and when the AC voltage drops to 160V, the DC voltage at the front end of the voltage regulator 7805 is 8.5V. When the dc voltage at the front end of the voltage regulator 7805 is lower than or equal to 8.5V, the dc voltage at the front end is divided by the resistors R1 and R2, and added to the comparator positive input terminal CMP+. The input voltage at the CMP+ terminal is lower than the internal reference voltage. A comparator interrupt can be generated at this time, so that there is sufficient time to save the data to the EEPROM during power-down detection. When the DC voltage of the front end of the voltage regulator 7805 is higher than 8.5V, the DC voltage input by the front end is divided by the resistors R1 and R2, and connected to the comparator positive input terminal CMP+. The input voltage of the CMP+ terminal is higher than the internal reference voltage. At this time, the CPU Can continue to work normally.

The internal reference voltage is the REPV of the internal BandGap after the amplified (the internal reference voltage is adjusted to 1.19V when the chip is shipped from the factory). The specific value should be

obtained by reading the value occupied by the internal reference voltage in the internal RAM area or the Flash program memory (ROM) area. For STC8 series, the storage address of the internal reference voltage value in RAM and Flash program memory (ROM), please refer to "Chapter 7.3 Special Parameters in Memory".

14.3.5 Comparator is Used to Detect the Operation Voltage (Battery Voltage)



	System clock<=10MHz	System clock>10MHz
C?	104(0.1uF)	103(0.01uF)

In the figure above, the working voltage of the MCU can be approximately measured using the principle of resistance voltage division. The I/O port of selected channel outputs low level, which is close to GND, and the I/O port of unselected channel working in open-drain mode outputs high. Other channels are not affected.

The negative terminal of the comparator selects the internal reference voltage, and the positive terminal selects the voltage value got from the voltage divided by a resistor as the input to the CMP+ pin.

In initialization, P1.5 ~ P1.0 ports are set to open-drain mode and output high. Firstly, P1.0 outputs a low level. At this time, if the VCC voltage is lower than 2.5V, the comparison value of the comparator is 0. Otherwise, if the VCC voltage is higher than 2.5V, the comparison value of the comparator is 1.

If you make sure that VCC is higher than 2.5V, then make the output of P1.0 high and the output of P1.1 low. At this time, if the VCC voltage is lower than 3.0V, the comparison value of the comparator is 0. Otherwise, if the VCC voltage is higher than 3.0V, the comparison value of the comparator is 1.

If you make sure that VCC is higher than 3.0V, then make the output of P1.1 high and the output of P1.2 low. At this time, if the VCC voltage is lower than 3.5V, the comparison value of the comparator is 0. Otherwise, if the VCC voltage is higher than 3.5V, the comparison value of the comparator is 1.

If you make sure that VCC is higher than 3.5V, then make the output of P1.2 high and the output of P1.3 low. At this time, if the VCC voltage is lower than 4.0V, the comparison value of the comparator is 0. Otherwise, if the VCC voltage is higher than 4.0V, the comparison value of the comparator is 1.

If you make sure that VCC is higher than 4.0V, then make the output of P1.3 high and the output of P1.4 low. At this time, if the VCC voltage is lower than 4.5V, the comparison value of the comparator is 0. Otherwise, if the VCC voltage is higher than 4.5V, the comparison value of the comparator is 1.

If you make sure that VCC is higher than 4.5V, then make the output of P1.4 high and the output of P1.5 low. At this time, if the VCC voltage is lower than 5.0V, the comparison value of the comparator is 0. Otherwise, if the VCC voltage is higher than 5.0V, the comparison value of the comparator is 1.

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     CMPCR1    =    0xe6;
sfr     CMPCR2    =    0xe7;

sfr     P0M1      =    0x93;
sfr     P0M0      =    0x94;
sfr     P1M1      =    0x91;
sfr     P1M0      =    0x92;
sfr     P2M1      =    0x95;
sfr     P2M0      =    0x96;
sfr     P3M1      =    0xb1;
sfr     P3M0      =    0xb2;
sfr     P4M1      =    0xb3;
sfr     P4M0      =    0xb4;
sfr     P5M1      =    0xc9;
sfr     P5M0      =    0xca;

void delay ()
{
    char i;

    for (i=0; i<20; i++);
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    unsigned char v;

    P1M0 = 0x3f;                    //P1.5 ~ P1.0 are initialized to open-drain mode
    P1M1 = 0x3f;
    P1 = 0xff;

    CMPCR2 = 0x10;                  //Output comparator result after 16 debounce clocks
    CMPCR1 = 0x00;
    CMPCR1 &= ~0x08;                //P3.6 is CMP + input pin
```

```
    CMPCR1 &= ~0x04;                    //Internal reference voltage is CMP- input pin
    CMPCR1 &= ~0x02;                    //Disable comparator output
    CMPCR1 |= 0x80;                     //Enable comparator module

    while (1)
    {
        v = 0x00;                       //Voltage <2.5V
        P1 = 0xfe;                      //P1.0 outputs 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x01;                       //Voltage>2.5V
        P1 = 0xfd;                      //P1.1 outputs 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x03;                       //Voltage>3.0V
        P1 = 0xfb;                      //P1.2 outputs 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x07;                       //Voltage>3.5V
        P1 = 0xf7;                      //P1.3 outputs 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x0f;                       //Voltage>4.0V
        P1 = 0xef;                      //P1.4 outputs 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x1f;                       //Voltage>4.5V
        P1 = 0xdf;                      //P1.5 outputs 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x3f;                       //Voltage>5.0V
ShowVol:
        P1 = 0xff;
        P0 = ~v;
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

P2M0        DATA    096H
P2M1        DATA    095H
P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH
```

```
                ORG     0000H
                LJMP    MAIN

                ORG     0100H
MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                MOV     P1M0,#00111111B     ;P1.5 ~ P1.0 are initialized to open-drain mode
                MOV     P1M1,#00111111B
                MOV     P1,#0FFH
                MOV     CMPCR2,#10H         ;Output comparator result after 16 debounce clocks
                MOV     CMPCR1,#00H
                ANL     CMPCR1,#NOT 08H     ;P3.7 is CMP+ input pin
                ANL     CMPCR1,#NOT 04H     ;Internal reference voltage is CMP- input pin
                ANL     CMPCR1,#NOT 02H     ;Disable comparator output
                ORL     CMPCR1,#80H         ;Enable comparator module
LOOP:
                MOV     R0,#00000000B       ;Voltage <2.5V
                MOV     P1,#11111110B       ;P1.0 outputs 0
                CALL    DELAY
                MOV     A,CMPCR1
                JNB     ACC.0,SKIP
                MOV     R0,#00000001B       ;Voltage>2.5V
                MOV     P1,#11111101B       ;P1.1 outputs 0
                CALL    DELAY
                MOV     A,CMPCR1
                JNB     ACC.0,SKIP
                MOV     R0,#00000011B       ;Voltage>3.0V
                MOV     P1,#11111011B       ;P1.2 outputs 0
                CALL    DELAY
                MOV     A,CMPCR1
                JNB     ACC.0,SKIP
                MOV     R0,#00000111B       ;Voltage>3.5V
                MOV     P1,#11110111B       ;P1.3 outputs 0
                CALL    DELAY
                MOV     A,CMPCR1
                JNB     ACC.0,SKIP
                MOV     R0,#00001111B       ;Voltage>4.0V
                MOV     P1,#11101111B       ;P1.4 outputs 0
                CALL    DELAY
                MOV     A,CMPCR1
                JNB     ACC.0,SKIP
                MOV     R0,#00011111B       ;Voltage>4.5V
                MOV     P1,#11011111B       ;P1.5 outputs 0
                CALL    DELAY
                MOV     A,CMPCR1
                JNB     ACC.0,SKIP
```

```
        MOV     R0,#00111111B       ;Voltage>5.0V
SKIP:
        MOV     P1,#11111111B
        MOV     A,R0
        CPL     A
        MOV     P0,A                ;P0.5 ~ P0.0 display voltage
        JMP     LOOP

DELAY:
        MOV     R0,#20
        DJNZ    R0,$
        RET

        END
```

STC MCU

15 IAP/EEPROM

Large capacity of internal EEPROM is integrated in STC8G series of microcontrollers. The internal Data Flash can be used as EEPROM by using ISP / IAP technology. And it can be repeatedly erased more than 100,000 times. EEPROM can be divided into several sectors, each sector contains 512 bytes.

Note: The EEPROM write operation can only write 1 in the byte as 0. When the 0 in the byte needs to be written as 1, the sector erase operation must be performed. The read/write operation of EEPROM is carried out in units of 1 byte, while the erasing operation of EEPROM is carried out in units of 1 sector (512 bytes). During the erasing operation, if there are data that need to be reserved in the target sector, these data must be read into RAM for temporary storage in advance, and then the saved data and the data that need to be updated will be written back to EEPROM/DATA-FLASH after erasing is completed.

When EEPROM is used, it is recommended that the data modified at the same time be stored in the same sector, and data modified at different time be stored in different sectors, and not necessarily full. Data memory is erased sector by sector.

EEPROM can be used to save some parameters which need to be modified in the application and need be kept when power down takes place. In the user program, byte read / byte programming / sector erase can be performed to the EEPROM. When the operating voltage is low, it is recommended not to carry out EEPROM operation to avoid data loss.

15.1 EEPROM operation time

- Read 1 byte: 4 system clocks (use MOVC instruction to read more convenient and fast)
- Programming 1 byte: about 30~40us (the actual programming time is 6~7.5us, but the state conversion time and the SETUP and HOLD time of various control signals need to be added)
- Erase 1 sector (512 bytes): about 4~6ms

The time required for EEPROM operation is automatically controlled by the hardware, and the user only needs to set the IAP_TPS register correctly.

IAP_TPS = system operating frequency/1000000 (the decimal part is rounded to the nearest whole number)

For example, the operating frequency of the system is 12MHz, then IAP_TPS is set to 12.

Another example, the system operating frequency is 22.1184MHz, then IAP_TPS is set to 22.

Another example, the system operating frequency is 5.5296MHz, then IAP_TPS is set to 6.

15.2 Registers Related to EEPROM

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
IAP_DATA	IAP Flash Data Register	C2H									1111,1111
IAP_ADDRH	IAP Flash Address High Byte	C3H									0000,0000
IAP_ADDRL	IAP Flash Address Low Byte	C4H									0000,0000
IAP_CMD	IAP Flash Command Register	C5H	-	-	-	-	-	-	CMD[1:0]		xxxx,xx00
IAP_TRIG	IAP Flash Trigger register	C6H									0000,0000
IAP_CONTR	IAP Control Register	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-	0000,xxxx
IAP_TPS	IAP Waiting Time Control	F5H	-	-	IAPTPS[5:0]						xx00,0000

	Register					

15.2.1 EEPROM data register (IAP_DATA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
IAP_DATA	C2H								

During EEPROM being read operation, the EEPROM data be read after the command execution is completed is stored in the IAP_DATA register. When writing the EEPROM, the data to be written must be stored in the IAP_DATA register before the write command is sent. The erase EEPROM command is not related to the IAP_DATA register.

15.2.2 EEPROM address registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
IAP_ADDRH	C3H								
IAP_ADDRL	C4H								

The target address register of EEPROM for reading, writing and erasing operation. IAP_ADDRH is the high byte address, and IAP_ADDRL is the low byte of the address.

15.2.3 EEPROM command register (IAP_CMD)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CMD	C5H	-	-	-	-	-	-	CMD[1:0]	

CMD[1:0]: EEPROM operation command to be sent.

00: No operation.

01: EEPROM reading command. Read one byte from the destination address.

10: EEPROM writing command. Write one byte from the destination address. Note: Write operation can only write 1 in the target byte as 0, but cannot write 0 as 1. Generally, when the target byte is not FFH, it must be erased first.

11: EEPROM erasing command. Write one sector from the destination address. Note: The erasing operation will erase 1 sector (512 bytes) at a time, and the contents of the entire sector will all become FFH.

15.2.4 EEPROM trigger register (IAP_TRIG)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TRIG	C6H								

After setting the command register, address register, data register and control register of EEPROM for reading, writing and erasing operation, 5AH and A5H are written to the trigger register IAP_TRIG sequentially to trigger the corresponding operation. The order of 5AH and A5H can not be changed. After the operation is completed, the contents of the EEPROM address registers IAP_ADDRH, IAP_ADDRL and the EEPROM command register IAP_CMD do not change. The value of the IAP_ADDRH and IAP_ADDRL registers must be updated manually if the datum of the next address needs to be operated.

Note: For every EEPROM operation, 5AH should be written to IAP_TRIG firstly and then A5H to take effect the corresponding command. After the trigger command has been written, the CPU is in IDLE state until the corresponding IAP operation completes. And then the CPU will return to the normal state from the IDLE and resume executing the CPU instructions.

15.2.5 EEPROM control register (IAP_CONTR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-

IAPEN: EEPROM operation enable control bit.

0: disable EEPROM operation.

1: Enable EEPROM operation.

SWBS: Software reset selection control bit, which should be used with SWRST.

0: Execute the program from the user code area after the software reset.

1: Execute the program from the ISP memory area after the software reset.

SWRST: Software reset control bit.

0: No operation.

1: Generate software reset.

CMD_FAIL: Command fail status bit for EEPROM operation which should be cleared by software.

0: EEPROM operation is right.

1: EEPROMoperation fails.

15.2.6 EEPROM wait time control register (IAP_TPS)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TPS	F5H	-	-	IAPTPS[5:0]					

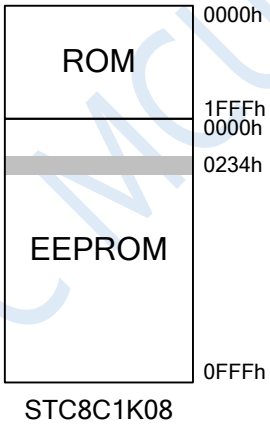
Need to be set according to the operating frequency. If the working frequency is 12MHz, IAP_TPS needs to be set to 12; if the working frequency is 24MHz, IAP_TPS needs to be set to 24, and so on for other frequencies.

STC MCU

15.3 EEPROM Size and Address

There is EEPROM for saving user data in all STC8G series of microcontrollers. There are three operation modes for the internal EEPROM: reading, writing, and erasing. The erasing operation is performed in sectors. Each sector has 512 bytes. That is, each time an erasing command will erase a sector when it executes. The reading and writing operations are in bytes, that is, only one byte can be read or written each time when a reading or writing command is executed.

There are two ways to access the internal EEPROM of STC8G series microcontrollers: IAP mode and MOVC mode. The IAP mode can perform reading, writing and erasing operations on the EEPROM. MOVC can only perform reading operations on the EEPROM, cannot perform writing and erasing operations. Regardless of whether IAP or MOVC is used to access the EEPROM, the correct target address must be set firstly. In IAP mode, the target address is the same as the actual physical address of the EEPROM. Both of them are accessed from address 0000H. However, when using MOVC instruction to read EEPROM data, the target address must be the actual physical address of the EEPROM plus a program size offset. STC8G1K08 is used as an example to describe the target address in detail as following.



STC8C1K08

The program space of STC8G1K08 is 8K bytes (0000h ~ 1FFFh), and the EEPROM space is 4K bytes (0000h ~ 0FFFh). When you need to read, write, and erase the unit with EEPROM physical address 0234h, if you use IAP to access, set the target address to 0234h, that is, IAP_ADDRH is set to 02h, IAP_ADDRL is set to 34h, and then the corresponding trigger command can be set and the 0234h can be operated correctly. However, if the 0234h unit of the EEPROM is read by MOVC, the flash program memory (ROM) space must be added in addition to 0234h. That is, the DPTR must be set to 2234h before the MOVC instruction can be used for reading.

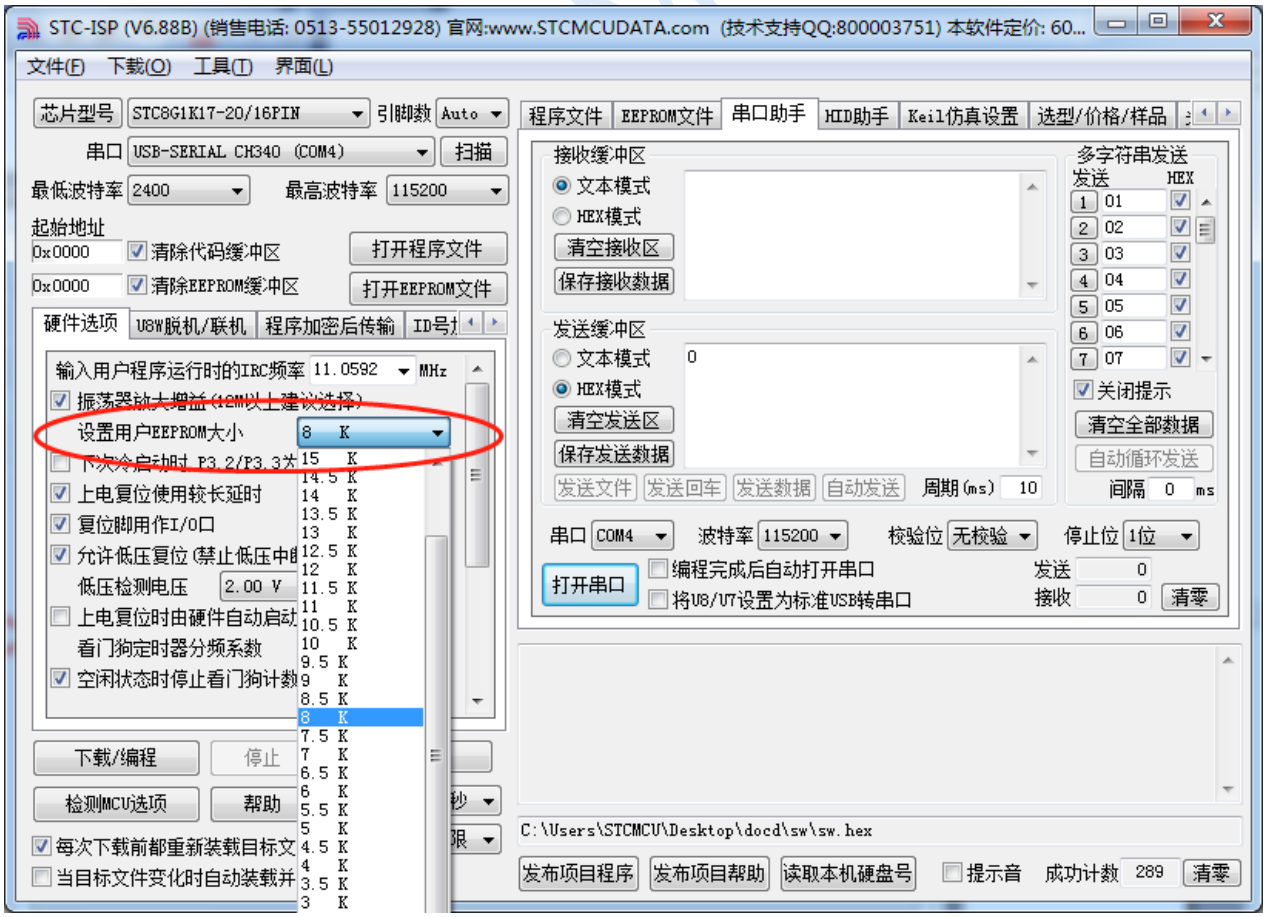
Note: Because the erasing is performed in 512-byte units, the lower 9 bits of the target address set when performing the erasing operation are meaningless. For example, if the target address is set to 0234H, 0200H, 0300H or 03FFH when executing the erasing command, the final erasing operation is the same, and the 512 bytes of 0200H ~ 03FFH are erased.

The size and access address of the internal EEPROM are different for different models. The size and address of EEPROM of each model are listed in the table below.

Model	Size	Sectors	Reading, writing, erasing in IAP mode		Reding using MOVC	
			Beginning adress	End address	Beginning adress	End address

STC8G1K04	8K	16	0000h	1FFFh	1000h	2FFFh
STC8G1K08	4K	8	0000h	0FFFh	2000h	2FFFh
STC8G1K12	User defined[1]					
STC8G1K17	User defined[1]					
STC8G1K08-8PIN	4K	8	0000h	0FFFh	2000h	2FFFh
STC8G1K12-8PIN	User defined[1]					
STC8G1K17-8PIN	User defined[1]					
STC8G1K08A	4K	8	0000h	0FFFh	2000h	2FFFh
STC8G1K12A	User defined[1]					
STC8G1K17A	User defined[1]					
STC8G1K08T	4K	8	0000h	0FFFh	2000h	2FFFh
STC8G1K12T	User defined[1]					
STC8G1K17T	User defined[1]					
STC8G2K32S4	32K	64	0000h	7FFFh	8000h	FFFFh
STC8G2K48S4	16K	32	0000h	3FFFh	C000h	FFFFh
STC8G2K60S4	4K	8	0000h	0FFFh	F000h	FFFFh
STC8G2K64S4	User defined[1]					
STC8G2K32S2	32K	64	0000h	7FFFh	8000h	FFFFh
STC8G2K48S2	16K	32	0000h	3FFFh	C000h	FFFFh
STC8G2K60S2	4K	8	0000h	0FFFh	F000h	FFFFh
STC8G2K64S2	User defined[1]					
STC15H2K32S4	32K	64	0000h	7FFFh	8000h	FFFFh
STC15H2K48S4	16K	32	0000h	3FFFh	C000h	FFFFh
STC15H2K60S4	4K	8	0000h	0FFFh	F000h	FFFFh
STC15H2K64S4	User defined[1]					

[1]: This is a special model. The EEPROM size of this model can be set by the user when downloading by the ISP, as shown below:



Users can plan any EEPROM space in the entire FLASH space provided that the size does not exceed the

FLASH size according to their own needs. It should be noted that the EEPROM is always planned from the back to the front.

For example, the size of FLASH in STC8G1K12 is 12K. If user wants to allocate 4K of it as EEPROM, the physical address of EEPROM is the last 4K of 12K, and the physical address is 2000h ~ 2FFFh. Of course, if the user uses IAP to access, the target address still starts from 0000h and ends at 0FFFh. When using MOVC to read, the target address is in the range from 2000h to 2FFFh.

STC MCU

15.4 Example Routines

15.4.1 EEPROM Basic Operation

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sfr         IAP_DATA    =   0xC2;
sfr         IAP_ADDRH   =   0xC3;
sfr         IAP_ADDRL   =   0xC4;
sfr         IAP_CMD     =   0xC5;
sfr         IAP_TRIG    =   0xC6;
sfr         IAP_CONTR   =   0xC7;
sfr         IAP_TPS     =   0xF5;

void IapIdle()
{
    IAP_CONTR = 0;                  //Disable IAP function
    IAP_CMD = 0;                    //Clear command register
    IAP_TRIG = 0;                   //Clear trigger register
    IAP_ADDRH = 0x80;               //Set the address to a non-IAP area
    IAP_ADDRL = 0;
}

char IapRead(int addr)
{
    char dat;

    IAP_CONTR = 0x80;               //Enable IAP
    IAP_TPS = 12;                   //Set the erasing wait parameter of 12MHz
    IAP_CMD = 1;                    //Set IAP read command
    IAP_ADDRL = addr;               //Set IAP low address
    IAP_ADDRH = addr >> 8;          //Set IAP high address
    IAP_TRIG = 0x5a;                //Write trigger command (0x5a)
    IAP_TRIG = 0xa5;                //Write trigger command (0xa5)
    _nop_();
    dat = IAP_DATA;                 //Read IAP data
    IapIdle();                      //Disable IAP function
```

STC MCU

```
    return dat;
}

void IapProgram(int addr, char dat)
{
    IAP_CONTR = 0x80;                   //Enable IAP
    IAP_TPS = 12;                       //Set the erasing wait parameter of 12MHz
    IAP_CMD = 2;                        //Set IAP writing command
    IAP_ADDRL = addr;                   //Set IAP low address
    IAP_ADDRH = addr >> 8;              //Set IAP high address
    IAP_DATA = dat;                     //Write IAP data
    IAP_TRIG = 0x5a;                    //Write trigger command (0x5a)
    IAP_TRIG = 0xa5;                    //Write trigger command (0xa5)
    _nop_();
    IapIdle();                          //Disable IAP function
}

void IapErase(int addr)
{
    IAP_CONTR = 0x80;                   //Enable IAP
    IAP_TPS = 12;                       //Set the erasing wait parameter of 12MHz
    IAP_CMD = 3;                        //Set IAP erasing command
    IAP_ADDRL = addr;                   //Set IAP low address
    IAP_ADDRH = addr >> 8;              //Set IAP high address
    IAP_TRIG = 0x5a;                    //Write trigger command (0x5a)
    IAP_TRIG = 0xa5;                    //Write trigger command (0xa5)
    _nop_();                            //
    IapIdle();                          //Disable IAP function
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IapErase(0x0400);
    P0 = IapRead(0x0400);               //P0=0xff
    IapProgram(0x0400, 0x12);
    P1 = IapRead(0x0400);               //P1=0x12

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

IAP_DATA    DATA    0C2H
IAP_ADDRH   DATA    0C3H
```

```
IAP_ADDRL   DATA    0C4H
IAP_CMD     DATA    0C5H
IAP_TRIG    DATA    0C6H
IAP_CONTR   DATA    0C7H
IAP_TPS     DATA    0F5H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H

IAP_IDLE:
            MOV     IAP_CONTR,#0        ;Disable IAP function
            MOV     IAP_CMD,#0          ;Clear command register
            MOV     IAP_TRIG,#0         ;Clear trigger register
            MOV     IAP_ADDRH,#80H      ;Set the address to a non-IAP area
            MOV     IAP_ADDRL,#0
            RET

IAP_READ:
            MOV     IAP_CONTR,#80H      ;Enable IAP
            MOV     IAP_TPS,#12         ;Set the erasing wait parameter of 12MHz
            MOV     IAP_CMD,#1          ;Set IAP read command
            MOV     IAP_ADDRL,DPL       ;Set IAP low address
            MOV     IAP_ADDRH,DPH       ;Set IAP high address
            MOV     IAP_TRIG,#5AH       ;Write trigger command (0x5a)
            MOV     IAP_TRIG,#0A5H      ;Write trigger command (0xa5)
            NOP
            MOV     A,IAP_DATA          ;Read IAP data
            LCALL   IAP_IDLE            ;Disable IAP function
            RET

IAP_PROGRAM:
            MOV     IAP_CONTR,#80H      ;Enable IAP
            MOV     IAP_TPS,#12         ;Set the erasing wait parameter of 12MHz
            MOV     IAP_CMD,#2          ;Set IAP writing command
            MOV     IAP_ADDRL,DPL       ;Set IAP low address
            MOV     IAP_ADDRH,DPH       ;Set IAP high address
            MOV     IAP_DATA,A          ;Write IAP data
            MOV     IAP_TRIG,#5AH       ;Write trigger command (0x5a)
            MOV     IAP_TRIG,#0A5H      ;Write trigger command (0xa5)
            NOP
            LCALL   IAP_IDLE            ;Disable IAP function
            RET

IAP_ERASE:
```

```
        MOV     IAP_CONTR,#80H          ;Enable IAP
        MOV     IAP_TPS,#12             ;Set the erasing wait parameter of 12MHz
        MOV     IAP_CMD,#3              ;Set IAP erasing command
        MOV     IAP_ADDRL,DPL           ;Set IAP low address
        MOV     IAP_ADDRH,DPH           ;Set IAP high address
        MOV     IAP_TRIG,#5AH           ;Write trigger command (0x5a)
        MOV     IAP_TRIG,#0A5H          ;Write trigger command (0xa5)
        NOP
        LCALL   IAP_IDLE                ;Disable IAP function
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     DPTR,#0400H
        LCALL   IAP_ERASE
        MOV     DPTR,#0400H
        LCALL   IAP_READ
        MOV     P0,A                    ;P0=0FFH
        MOV     DPTR,#0400H
        MOV     A,#12H
        LCALL   IAP_PROGRAM
        MOV     DPTR,#0400H
        LCALL   IAP_READ
        MOV     P1,A                    ;P1=12H

        SJMP    $

        END
```

15.4.2 Read EEPROM using MOVC

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
```

```
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

sfr     IAP_DATA    =   0xC2;
sfr     IAP_ADDRH   =   0xC3;
sfr     IAP_ADDRL   =   0xC4;
sfr     IAP_CMD     =   0xC5;
sfr     IAP_TRIG    =   0xC6;
sfr     IAP_CONTR   =   0xC7;
sfr     IAP_TPS     =   0xF5;

#define IAP_OFFSET      0x2000H             //STC8G1K08

void IapIdle()
{
    IAP_CONTR = 0;                          //Disable IAP function
    IAP_CMD = 0;                            //Clear command register
    IAP_TRIG = 0;                           //Clear trigger register
    IAP_ADDRH = 0x80;                       //Set the address to a non-IAP area
    IAP_ADDRL = 0;
}

char IapRead(int addr)
{
    addr += IAP_OFFSET;                     //Using MOVC to read the EEPROM needs to add the
corresponding offset
    return *(char code *)(addr);            //Read data using MOVC
}

void IapProgram(int addr, char dat)
{
    IAP_CONTR = 0x80;                       //Enable IAP
    IAP_TPS = 12;                           //Set the erasing wait parameter of 12MHz
    IAP_CMD = 2;                            //Set IAP writing command
    IAP_ADDRL = addr;                       //Set IAP low address
    IAP_ADDRH = addr >> 8;                  //Set IAP high address
    IAP_DATA = dat;                         //Write IAP data
    IAP_TRIG = 0x5a;                        //Write trigger command (0x5a)
    IAP_TRIG = 0xa5;                        //Write trigger command (0xa5)
    _nop_();
    IapIdle();                              //Disable IAP function
}

void IapErase(int addr)
{
    IAP_CONTR = 0x80;                       //Enable IAP
    IAP_TPS = 12;  //Set the erasing wait parameter of 12MHz
    IAP_CMD = 3;  //Set IAP erasing command
    IAP_ADDRL = addr;                       //Set IAP low address
    IAP_ADDRH = addr >> 8;                  //Set IAP high address
    IAP_TRIG = 0x5a;                        //Write trigger command (0x5a)
    IAP_TRIG = 0xa5;                        //Write trigger command (0xa5)
    _nop_();                                //
    IapIdle();                              //Disable IAP function
```

```
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IapErase(0x0400);
    P0 = IapRead(0x0400);                   //P0=0xff
    IapProgram(0x0400, 0x12);
    P1 = IapRead(0x0400);                   //P1=0x12

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

IAP_DATA    DATA    0C2H
IAP_ADDRH   DATA    0C3H
IAP_ADDRL   DATA    0C4H
IAP_CMD     DATA    0C5H
IAP_TRIG    DATA    0C6H
IAP_CONTR   DATA    0C7H
IAP_TPS     DATA    0F5H

IAP_OFFSET  EQU     2000H               ;STC8G1K08

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H

IAP_IDLE:
            MOV     IAP_CONTR,#0        ;Disable IAP function
```

```
        MOV     IAP_CMD,#0              ;Clear command register
        MOV     IAP_TRIG,#0             ;Clear trigger register
        MOV     IAP_ADDRH,#80H          ;Set the address to a non-IAP area
        MOV     IAP_ADDRL,#0
        RET

IAP_READ:
        MOV     A,#LOW IAP_OFFSET       ;Using MOVC to read the EEPROM needs to add the corresponding offset
        ADD     A,DPL
        MOV     DPL,A
        MOV     A,@HIGH IAP_OFFSET
        ADDC    A,DPH
        MOV     DPH,A
        CLR     A
        MOVC    A,@A+DPTR               ;Read data using MOVC
        RET

IAP_PROGRAM:
        MOV     IAP_CONTR,#80H          ;Enable IAP
        MOV     IAP_TPS,#12             ;Set the erasing wait parameter of 12MHz
        MOV     IAP_CMD,#2              ;Set IAP writing command
        MOV     IAP_ADDRL,DPL           ;Set IAP low address
        MOV     IAP_ADDRH,DPH           ;Set IAP high address
        MOV     IAP_DATA,A              ;Write IAP data
        MOV     IAP_TRIG,#5AH           ;Write trigger command (0x5a)
        MOV     IAP_TRIG,#0A5H          ;Write trigger command (0xa5)
        NOP
        LCALL   IAP_IDLE                ;Disable IAP function
        RET

IAP_ERASE:
        MOV     IAP_CONTR,#80H          ;Enable IAP
        MOV     IAP_TPS,#12             ;Set the erasing wait parameter of 12MHz
        MOV     IAP_CMD,#3              ;Set IAP erasing command
        MOV     IAP_ADDRL,DPL           ;Set IAP low address
        MOV     IAP_ADDRH,DPH           ;Set IAP high address
        MOV     IAP_TRIG,#5AH           ;Write trigger command (0x5a)
        MOV     IAP_TRIG,#0A5H          ;Write trigger command (0xa5)
        NOP
        LCALL   IAP_IDLE                ;Disable IAP function
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     DPTR,#0400H
```

```
        LCALL   IAP_ERASE
        MOV     DPTR,#0400H
        LCALL   IAP_READ
        MOV     P0,A                    ;P0=0FFH
        MOV     DPTR,#0400H
        MOV     A,#12H
        LCALL   IAP_PROGRAM
        MOV     DPTR,#0400H
        LCALL   IAP_READ
        MOV     P1,A                    ;P1=12H

        SJMP    $

        END
```

15.4.3 Send Out the Data in EEPROM Using UART

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define FOSC        11059200UL
#define BRT         (65536 - FOSC / 115200 / 4)

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sfr     AUXR    =   0x8e;
sfr     T2H     =   0xd6;
sfr     T2L     =   0xd7;

sfr     IAP_DATA    =   0xC2;
sfr     IAP_ADDRH   =   0xC3;
sfr     IAP_ADDRL   =   0xC4;
sfr     IAP_CMD     =   0xC5;
sfr     IAP_TRIG    =   0xC6;
sfr     IAP_CONTR   =   0xC7;
sfr     IAP_TPS     =   0xF5;

void UartInit()
{
    SCON = 0x5a;
    T2L = BRT;
```

```
    T2H = BRT >> 8;
    AUXR = 0x15;
}

void UartSend(char dat)
{
    while (!TI);
    TI = 0;
    SBUF = dat;
}

void IapIdle()
{
    IAP_CONTR = 0;                  //Disable IAP function
    IAP_CMD = 0;                    //Clear command register
    IAP_TRIG = 0;                   //Clear trigger register
    IAP_ADDRH = 0x80;               //Set the address to a non-IAP area
    IAP_ADDRL = 0;
}

char IapRead(int addr)
{
    char dat;

    IAP_CONTR = 0x80;               //Enable IAP
    IAP_TPS = 12;                   //Set the erasing wait parameter of 12MHz
    IAP_CMD = 1;                    //Set IAP read command
    IAP_ADDRL = addr;               //Set IAP low address
    IAP_ADDRH = addr >> 8;          //Set IAP high address
    IAP_TRIG = 0x5a;                //Write trigger command (0x5a)
    IAP_TRIG = 0xa5;                //Write trigger command (0xa5)
    _nop_();
    dat = IAP_DATA;                 //Read IAP data
    IapIdle();                      //Disable IAP function

    return dat;
}

void IapProgram(int addr, char dat)
{
    IAP_CONTR = 0x80;               //Enable IAP
    IAP_TPS = 12;                   //Set the erasing wait parameter of 12MHz
    IAP_CMD = 2;                    //Set IAP writing command
    IAP_ADDRL = addr;               //Set IAP low address
    IAP_ADDRH = addr >> 8;          //Set IAP high address
    IAP_DATA = dat;                 //Write IAP data
    IAP_TRIG = 0x5a;                //Write trigger command (0x5a)
    IAP_TRIG = 0xa5;                //Write trigger command (0xa5)
    _nop_();
    IapIdle();                      //Disable IAP function
}

void IapErase(int addr)
{
    IAP_CONTR = 0x80;               //Enable IAP
    IAP_TPS = 12;                   //Set the erasing wait parameter of 12MHz
    IAP_CMD = 3;                    //Set IAP erasing command
    IAP_ADDRL = addr;               //Set IAP low address
    IAP_ADDRH = addr >> 8;          //Set IAP high address
```

```
    IAP_TRIG = 0x5a;                    //Write trigger command (0x5a)
    IAP_TRIG = 0xa5;                    //Write trigger command (0xa5)
    _nop_();                            //
    IapIdle();                          //Disable IAP function
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    IapErase(0x0400);
    UartSEND(IapRead(0x0400));
    IapProgram(0x0400, 0x12);
    UartSEND(IapRead(0x0400));

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H

IAP_DATA    DATA    0C2H
IAP_ADDRH   DATA    0C3H
IAP_ADDRL   DATA    0C4H
IAP_CMD     DATA    0C5H
IAP_TRIG    DATA    0C6H
IAP_CONTR   DATA    0C7H
IAP_TPS     DATA    0F5H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH
```

STCMCU

```
                ORG     0000H
                LJMP    MAIN

                ORG     0100H

UART_INIT:
                MOV     SCON,#5AH
                MOV     T2L,#0E8H       ;65536-11059200/115200/4=0FFE8H
                MOV     T2H,#0FFH
                MOV     AUXR,#15H
                RET

UART_SEND:
                JNB     TI,$
                CLR     TI
                MOV     SBUF,A
                RET

IAP_IDLE:
                MOV     IAP_CONTR,#0    ;Disable IAP function
                MOV     IAP_CMD,#0      ;Clear command register
                MOV     IAP_TRIG,#0     ;Clear trigger register
                MOV     IAP_ADDRH,#80H  ;Set the address to a non-IAP area
                MOV     IAP_ADDRL,#0
                RET

IAP_READ:
                MOV     IAP_CONTR,#80H  ;Enable IAP
                MOV     IAP_TPS,#12     ;Set the erasing wait parameter of 12MHz
                MOV     IAP_CMD,#1      ;Set IAP read command
                MOV     IAP_ADDRL,DPL   ;Set IAP low address
                MOV     IAP_ADDRH,DPH   ;Set IAP high address
                MOV     IAP_TRIG,#5AH   ;Write trigger command (0x5a)
                MOV     IAP_TRIG,#0A5H  ;Write trigger command (0xa5)
                NOP
                MOV     A,IAP_DATA      ;Read IAP data
                LCALL   IAP_IDLE        ;Disable IAP function
                RET

IAP_PROGRAM:
                MOV     IAP_CONTR,#80H  ;Enable IAP
                MOV     IAP_TPS,#12     ;Set the erasing wait parameter of 12MHz
                MOV     IAP_CMD,#2      ;Set IAP writing command
                MOV     IAP_ADDRL,DPL   ;Set IAP low address
                MOV     IAP_ADDRH,DPH   ;Set IAP high address
                MOV     IAP_DATA,A      ;Write IAP data
                MOV     IAP_TRIG,#5AH   ;Write trigger command (0x5a)
                MOV     IAP_TRIG,#0A5H  ;Write trigger command (0xa5)
                NOP
                LCALL   IAP_IDLE        ;Disable IAP function
                RET

IAP_ERASE:
                MOV     IAP_CONTR,#80H  ;Enable IAP
                MOV     IAP_TPS,#12     ;Set the erasing wait parameter of 12MHz
                MOV     IAP_CMD,#3      ;Set IAP erasing command
                MOV     IAP_ADDRL,DPL   ;Set IAP low address
                MOV     IAP_ADDRH,DPH   ;Set IAP high address
                MOV     IAP_TRIG,#5AH   ;Write trigger command (0x5a)
```

```
        MOV     IAP_TRIG,#0A5H      ;Write trigger command (0xa5)
        NOP
        LCALL   IAP_IDLE            ;Disable IAP function
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        LCALL   UART_INIT
        MOV     DPTR,#0400H
        LCALL   IAP_ERASE
        MOV     DPTR,#0400H
        LCALL   IAP_READ
        LCALL   UART_SEND
        MOV     DPTR,#0400H
        MOV     A,#12H
        LCALL   IAP_PROGRAM
        MOV     DPTR,#0400H
        LCALL   IAP_READ
        LCALL   UART_SEND

        SJMP    $

        END
```

STC MCU

16 ADC, Internal 1.19V Reference Voltage

Product line	ADC Resolution	Number ADC Channels
STC8G1K08 family	**10 bit**	**15**
STC8G1K08-8Pin family		
STC8G1K08A family	**10 bit**	**6**
STC8G2K64S4 family	**10 bit**	**15**
STC8G2K64S2 family	**10 bit**	**15**
STC8G1K08T family	**10 bit**	**15**
STC15H2K64S4 family	**10 bit**	**15**

A 10-bit high-speed Analog to Digital Converter is integrated in STC8G series of microcontrollers. The system frequency is divided by 2 and then divided again by the user-set division ratio as the clock frequency of the ADC. The range of ADC clock frequency is SYSclk/2/1 ~ SYSclk/2/16.

The maximum speed of the STC8G series ADC is 800K for 12-bit ADC (800,000 ADC conversions per second), and 500K for 10-bit ADC (500,000 ADC conversions per second).

There are two data formats for ADC conversion results: Align left and align right. It is convenient for user program to read and reference.

Note: The 15th channel of the ADC can only be used to detect the internal reference voltage. The reference voltage value is calibrated to 1.19V at the factory. Due to the manufacturing errors and measurement errors, the actual internal reference voltage has about ±1% error compared to 1.19V. If you want to know the exact internal reference voltage of each chip, you can connect the accurate reference voltage and then use the 15th channel of the ADC to measure the calibration.

If the chip has ADC external reference power supply pin ADC_VRef+, it must not be floating, it must be connected to an external reference power supply or directly connected to VCC.

16.1 Registers Related to ADC

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
ADC_CONTR	ADC control register	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]				000x,0000
ADC_RES	ADC Result High Byte	BDH									0000,0000
ADC_RESL	ADC Result Low Byte	BEH									0000,0000
ADCCFG	ADC Configuration Register	DEH	-	-	RESFMT	-	SPEED[3:0]				xx0x,0000
ADCTIM	ADC Timing Control Register	FEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]					0010,1010

16.1.1 ADC control register (ADC_CONTR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
ADC_CONTR	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]			

ADC_POWER: ADC power supply control bit.

0: turn off the power supply of ADC.

1: turn on the power supply of ADC.

It is recommended to turn off the ADC before entering Idle mode and Power-down mode to reduce the power consumption.

Attention:

1. After turning on the power of the internal ADC module of the MCU, wait for about 1ms, and let the ADC work after the internal ADC power supply of the MCU is stable.

2. Properly lengthen the sampling time for external signals, that is, the charging or discharging time of the

ADC's internal sampling and holding capacitor. The internal potential can be equal to the external potential if the time is sufficient.

ADC_START: ADC start bit. ADC conversion will start after write 1 to this bit. It is cleared automatically by the hardware after A/D conversion completes.

0: no effect. Writing 0 to this bit will not stop the A/D conversion if the ADC has already started.

1: start the A/D conversion. It is cleared automatically by the hardware after A/D conversion completes.

ADC_FLAG: ADC conversion completement flag. It is set by the hardware after the ADC conversion hasfinished, and requests interrupt to CPU. It must be cleared by software.

ADC_EPWMT: enable PWM synchronous trigger ADC function.

ADC_CHS[3:0]: ADC anolog channel selection bits.

(Note: PxM0/PxM1 registers of the I/O port selected as the ADC input channel must be set to set the I/O port mode to high-impedance input mode. In addition, if the MCU enters the power-down mode/clock stop mode, the ADC channel is still needed to be enabled, and the PxIE register is needed to set to close the digital input channel to prevent the external analog input signal from fluctuating high and low which will cause additional power consumption.)

(Note: The channels in red font in the table below represent that different series may be on different ports, and red just means highlighting.)

(STC8G1K08/STC8G1K08T family)

ADC_CHS[3:0]	ADC channel	ADC_CHS[3:0]	ADC channel
0000	P1.0/ADC0	1000	P3.0/ADC8
0001	P1.1/ADC1	1001	P3.1/ADC9
0010	P1.2/ADC2	1010	P3.2/ADC10
0011	P1.3/ADC3	1011	P3.3/ADC11
0100	P1.4/ADC4	1100	P3.4/ADC12
0101	P1.5/ADC5	1101	P3.5/ADC13
0110	P1.6/ADC6	1110	P3.6/ADC14
0111	P1.7/ADC7	1111	Test internal 1.19V

(STC8G2K64S4/STC8G2K64S2/STC15H2K64S4 family)

ADC_CHS[3:0]	ADC channel	ADC_CHS[3:0]	ADC channel
0000	P1.0/ADC0	1000	P0.0/ADC8
0001	P1.1/ADC1	1001	P0.1/ADC9
0010	P1.2/ADC2	1010	P0.2/ADC10
0011	P1.3/ADC3	1011	P0.3/ADC11
0100	P1.4/ADC4	1100	P0.4/ADC12
0101	P1.5/ADC5	1101	P0.5/ADC13
0110	P1.6/ADC6	1110	P0.6/ADC14
0111	P1.7/ADC7	1111	Test internal 1.19V

(STC8G1K08A family)

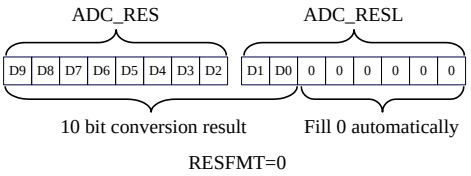
ADC_CHS[3:0]	ADC channel	ADC_CHS[3:0]	ADC channel
0000	P3.0/ADC0	1000	No such channel
0001	P3.1/ADC1	1001	No such channel
0010	P3.2/ADC2	1010	No such channel
0011	P3.3/ADC3	1011	No such channel
0100	P5.4/ADC4	1100	No such channel
0101	P5.5/ADC5	1101	No such channel
0110	No such channel	1110	No such channel
0111	No such channel	1111	Test internal 1.19V

16.1.2 ADC configuration register (ADCCFG)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
ADCCFG	DEH	-	-	RESFMT	-	SPEED[3:0]			

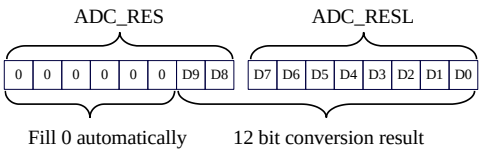
RESFMT: ADC conversion result format control bit

0: The conversion result aligns left. ADC_RES is used to save the upper 8 bits of the result and ADC_RESL is used to save the lower 2 bits of the result. The format is as follows:



RESFMT=0

1: The conversion result aligns right. ADC_RES is used to save the upper 2 bits of the result and ADC_RESL is used to save the lower 8 bits of the result. The format is as follows:



RESFMT=1

SPEED[3:0]: ADC clock control bits {F_{ADC}＝SYSclk/2/(SPEED+1)}

SPEED[3:0]	ADC clock frequency
0000	SYSclk/2/1
0001	SYSclk/2/2
0010	SYSclk/2/3
...	. . .
1101	SYSclk/2/14
1110	SYSclk/2/15
1111	SYSclk/2/16

16.1.3 ADC conversion result register (ADC_RES, ADC_RESL)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	BDH								
ADC_RESL	BEH								

After the A/D conversion is completed, the 10-bit conversion result is automatically saved to ADC_RES and ADC_RESL. Please refer to the RESFMT setting in the ADC_CFG register to see the result's data format.

16.1.4 ADC timing control register (ADCTIM)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
ADCTIM	FEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]				

CSSETUP: ADC channel selection time control T_{setup}

CSSETUP	ADC number of clocks
0	**1 (default)**
1	2

CSHOLD[1:0]: ADC Channel selection hold time control T_{hold}

CSHOLD[1:0]	ADC number of clocks
00	1
01	**2 (default)**
10	3
11	4

SMPDUTY[4:0]: ADC analog signal sampling time control T_{duty} (Note: SMPDUTY must not be less than 01010B)

SMPDUTY[4:0]	ADC number of clocks
00000	1

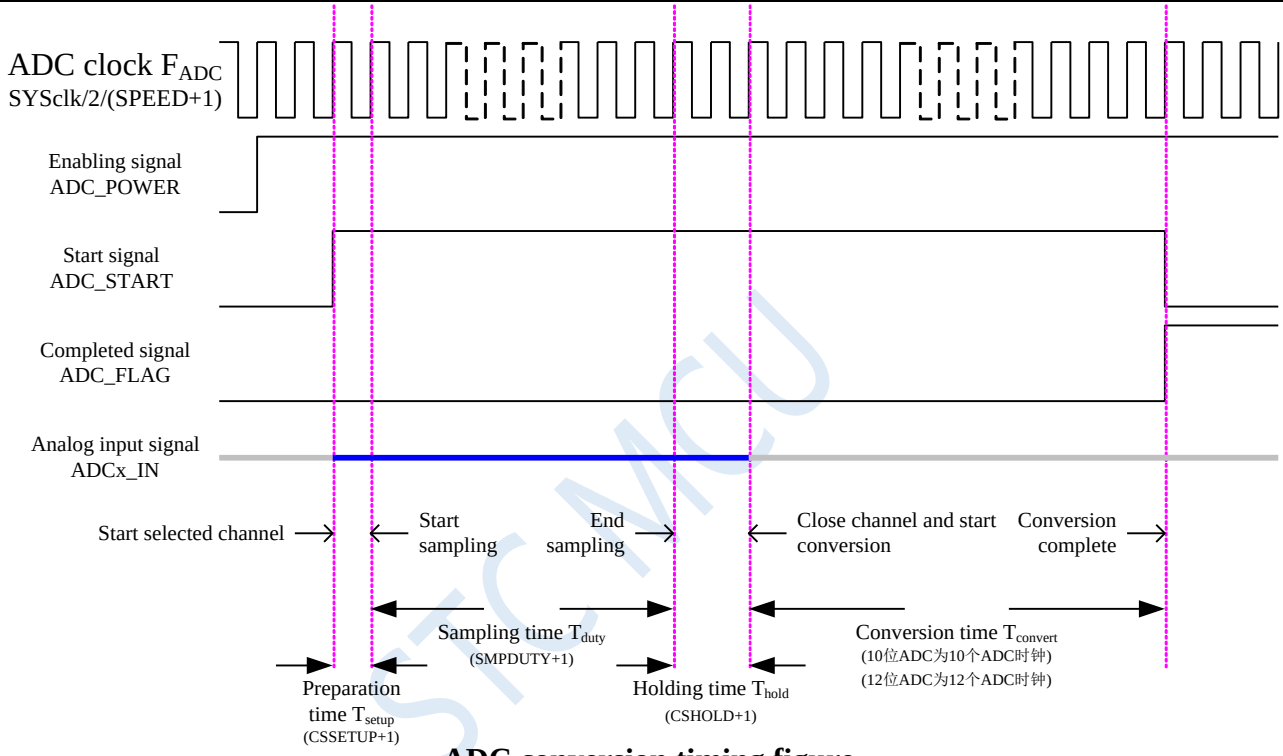
00001	2
...	...
01010	**11 (default)**
...	...
11110	31
11111	32

ADC digital-to-analog conversion time: Tconvert

The conversion time of 10-bit ADC is fixed at 10 ADC working clocks.

The conversion time of 12-bit ADC is fixed to 12 ADC working clocks.

A complete ADC conversion time is: Tsetup + Tduty + Thold + Tconvert, as shown in the figure below.



ADC conversion timing figure

16.2 Calculation formula related to ADC

16.2.1 Speed calculation formula of ADC

The conversion speed of the ADC is jointly controlled by the SPEED and ADCTIM registers in the ADCCFG register. The calculation formula of the conversion speed is as follows.

$$\text{Conversion speed of 10-bit ADC} = \frac{\text{MCU operation frequency SYSclk}}{2\times(\text{SPEED}[3:0]+1)\times[(\text{CSSETUP}+1)+(\text{CSHOLD}+1)+(\text{SMPDUTY}+1)+10]}$$

$$\text{Conversion speed of 12-bit ADC} = \frac{\text{MCU operation frequency SYSclk}}{2\times(\text{SPEED}[3:0]+1)\times[(\text{CSSETUP}+1)+(\text{CSHOLD}+1)+(\text{SMPDUTY}+1)+12]}$$

Notice:

- The speed of 10-bit ADC cannot be higher than 500KHz.
- The speed of 12-bit ADC cannot be higher than 800KHz.
- The value of SMPDUTY cannot be less than 10, it is recommended to be set to 15.
- CSSETUP can use the power-on default value 0.
- CHOLD can use the power-on default value 1 (ADCTIM is recommended to be set to 3FH).

16.2.2 Conversion result calculation formula of ADC

$$\text{Conversion result of 10-bit ADC} = 1024 \times \frac{\text{Input voltage of ADC channel converted Vin}}{\text{MCU operation voltage Vcc}} \quad \text{(No independent ADC_Vref+ pin)}$$

$$\text{Conversion result of 10-bit ADC} = 1024 \times \frac{\text{Input voltage of ADC channel converted Vin}}{\text{ADC external reference voltage}} \quad \text{(with independent ADC_Vref+ pin)}$$

$$\text{Conversion result of of 12-bit ADC} = 4096 \times \frac{\text{Input voltage of ADC channel converted Vin}}{\text{MCU operation voltage Vcc}} \quad \text{(No independent ADC_Vref+ pin)}$$

$$\text{Conversion result of of 12-bit ADC} = 4096 \times \frac{\text{Input voltage of ADC channel converted Vin}}{\text{ADC external reference voltage}} \quad \text{(with independent ADC_Vref+ pin)}$$

16.2.3 Reverse calculation formula for ADC input voltage

$$\text{Input voltage of ADC channel converted Vin} = \text{MCU operation voltage Vcc} \times \frac{\text{10-bit ADC conversion result}}{1024} \quad \text{(No independent ADC_Vref+ pin)}$$

$$\text{Input voltage of ADC channel converted Vin} = \text{ADC external reference voltage Vcc} \times \frac{\text{10-bit ADC conversion result}}{1024} \quad \text{(with independent ADC_Vref+ pin)}$$

$$\text{Input voltage of ADC channel converted Vin} = \text{MCU operation voltage Vcc} \times \frac{\text{12-bit ADC conversion result}}{4096} \quad \text{(No independent ADC_Vref+ pin)}$$

$$\text{Input voltage of ADC channel converted Vin} = \text{ADC external reference voltage Vcc} \times \frac{\text{12-bit ADC conversion result}}{4096} \quad \text{(with independent ADC_Vref+ pin)}$$

16.2.4 Reverse operation voltage calculation formula

When you need to use the ADC input voltage and ADC conversion results to reverse the operation voltage, if the target chip does not have an independent ADC_Vref+ pin, you can directly measure and use the following formula. If the target chip has an independent ADC_Vref+ pin, you must connect the ADC_Vref+ pin to the Vcc pin.

$$\text{MCU operation voltage Vcc} = 1024 \times \frac{\text{Input voltage of ADC channel converted Vin}}{\text{10-bit ADC conversion result}}$$

$$\text{MCU operation voltage Vcc} = 4096 \times \frac{\text{Input voltage of ADC channel converted Vin}}{\text{12-bit ADC conversion result}}$$

16.3 10-bit ADC Static Characteristics

Symbol	Description	Minimum	Typical	Max	Unit
RES	Resolution	-	10	-	Bits
E_T	Overall error	-	1.3	3	LSB
E_O	Offset error	-	0.3	1	LSB
E_G	Gain error	-	0	1	LSB
E_D	Differential nonlinear error	-	0.7	1.5	LSB
E_I	Integral nonlinear error	-	1	2	LSB
R_{AIN}	Channel equivalent resistance	-	∞	-	Ohm
R_{ESD}	Antistatic resistance connected in series before the sample and hold capacitor		700		
C_{ADC}	Internal sample and hold capacitor	-	16.5	-	pF

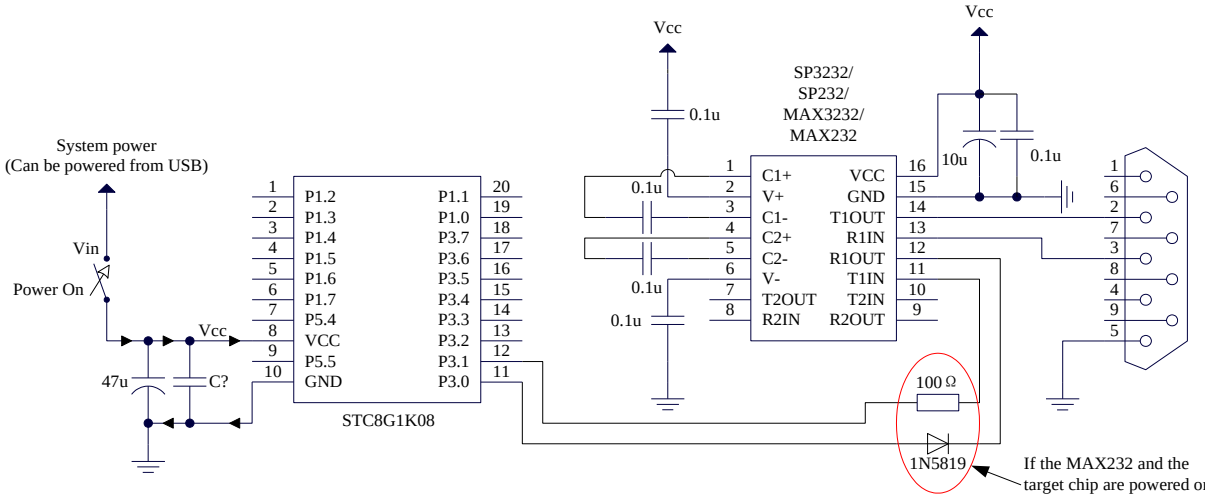
16.4 12-bit ADC Static Characteristics

Symbol	Description	Minimum	Typical	Max	Unit
RES	Resolution	-	12	-	Bits
E_T	Overall error	-	0.5	1	LSB
E_O	Offset error	-	-0.1	1	LSB
E_G	Gain error	-	0	1	LSB
E_D	Differential nonlinear error	-	0.7	1.5	LSB
E_I	Integral nonlinear error	-	1	2	LSB
R_{AIN}	Channel equivalent resistance	-	∞	-	Ohm
R_{ESD}	Antistatic resistance connected in series before the sample and hold capacitor		700		
C_{ADC}	Internal sample and hold capacitor	-	16.5	-	pF

STC MCU

16.5 ADC application reference circuit diagram

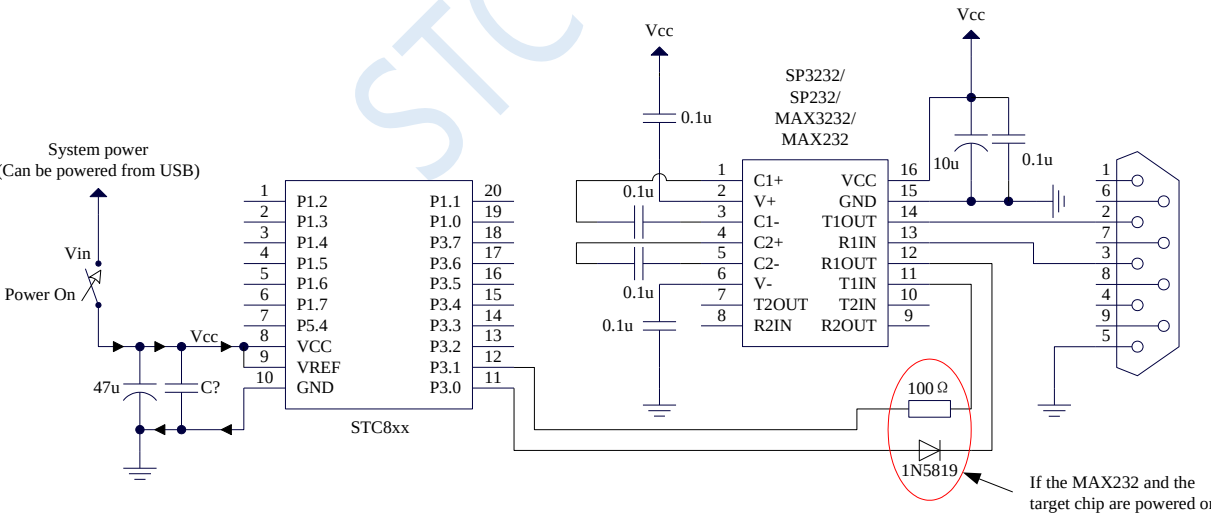
16.5.1 Reference circuit diagram without independent VREF pin



47u tantalum capacitor (package 3528) reference price<RMB ￥0.16
22u monolithic capacitor (package 0603) reference price<RMB ￥0.038
10u monolithic capacitor (package 0603) reference price<RMB ￥0.028
0.1u monolithic capacitor (package 0603) reference price<RMB ￥0.005

	System clock<=10MHz	System clock>10MHz
C?	104(0.1uF)	103(0.01uF)

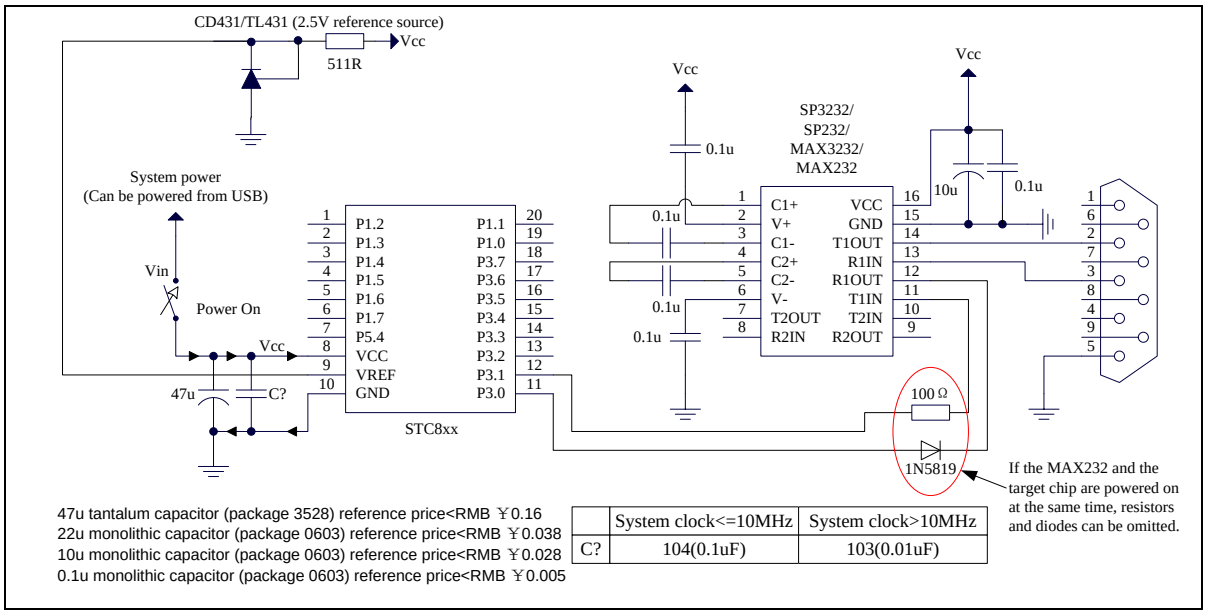
16.5.2 Reference circuit diagram of general precision ADC with independent VREF pin



47u tantalum capacitor (package 3528) reference price<RMB ￥0.16
22u monolithic capacitor (package 0603) reference price<RMB ￥0.038
10u monolithic capacitor (package 0603) reference price<RMB ￥0.028
0.1u monolithic capacitor (package 0603) reference price<RMB ￥0.005

	System clock<=10MHz	System clock>10MHz
C?	104(0.1uF)	103(0.01uF)

16.5.3 High-precision ADC reference circuit diagram with independent VREF pin



47u tantalum capacitor (package 3528) reference price<RMB ￥0.16
22u monolithic capacitor (package 0603) reference price<RMB ￥0.038
10u monolithic capacitor (package 0603) reference price<RMB ￥0.028
0.1u monolithic capacitor (package 0603) reference price<RMB ￥0.005

	System clock<=10MHz	System clock>10MHz
C?	104(0.1uF)	103(0.01uF)

STC MCU

16.6 Example Routines

16.6.1 ADC Basic Operation (Polling Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         ADC_CONTR   =       0xbc;
sfr         ADC_RES     =       0xbd;
sfr         ADC_RESL    =       0xbe;
sfr         ADCCFG      =       0xde;

sfr         P_SW2       =       0xba;
#define     ADCTIM      (*(unsigned char volatile xdata *)0xfea8)

sfr         P0M1        =       0x93;
sfr         P0M0        =       0x94;
sfr         P1M1        =       0x91;
sfr         P1M0        =       0x92;
sfr         P2M1        =       0x95;
sfr         P2M0        =       0x96;
sfr         P3M1        =       0xb1;
sfr         P3M0        =       0xb2;
sfr         P4M1        =       0xb3;
sfr         P4M0        =       0xb4;
sfr         P5M1        =       0xc9;
sfr         P5M0        =       0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P1M0 = 0x00;                    //Set P1.0 as ADC input
    P1M1 = 0x01;
    P_SW2 |= 0x80;
    ADCTIM = 0x3f;                  // Set ADC internal timing
    P_SW2 &= 0x7f;
    ADCCFG = 0x0f;                  //Set the ADC clock to the system clock/2/16
    ADC_CONTR = 0x80;               //Enable ADC module

    while (1)
```

STC MCU

```
    {
        ADC_CONTR |= 0x40;                  //Start AD conversion
        _nop_();
        _nop_();
        while (!(ADC_CONTR & 0x20));        //Query ADC completion flag
        ADC_CONTR &= ~0x20;                 //Clear completion flag
        P2 = ADC_RES;                       //Read ADC results
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

ADC_CONTR   DATA    0BCH
ADC_RES     DATA    0BDH
ADC_RESL    DATA    0BEH
ADCCFG      DATA    0DEH

P_SW2       DATA    0BAH
ADCTIM      XDATA   0FEA8H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P1M0,#00H           ;Set P1.0 as ADC input
            MOV     P1M1,#01H
            MOV     P_SW2,#80H
            MOV     DPTR,#ADCTIM        ; Set ADC internal timing
            MOV     A,#3FH
```

STC MCU

```
            MOVX    @DPTR,A
            MOV     P_SW2,#00H
            MOV     ADCCFG,#0FH         ;Set the ADC clock to the system clock/2/16/16
            MOV     ADC_CONTR,#80H      ;Enable ADC module
LOOP:
            ORL     ADC_CONTR,#40H      ;Start AD conversion
            NOP
            NOP
            MOV     A,ADC_CONTR         ;Query ADC completion flag
            JNB     ACC.5,$-2
            ANL     ADC_CONTR,#NOT 20H  ;Clear completion flag
            MOV     P2,ADC_RES          ;Read ADC results

            SJMP    LOOP

            END
```

16.6.2 ADC Basic Operation (Interrupt Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     ADC_CONTR   =   0xbc;
sfr     ADC_RES     =   0xbd;
sfr     ADC_RESL    =   0xbe;
sfr     ADCCFG      =   0xde;

sfr     P_SW2       =   0xba;
#define ADCTIM      (*(unsigned char volatile xdata *)0xfea8)

sbit    EADC        =   IE^5;

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

void ADC_Isr() interrupt 5
{
    ADC_CONTR &= ~0x20;                 //Clear interrupt flag
    P2 = ADC_RES;                       //Read ADC results
    ADC_CONTR |= 0x40;                  //Continue AD conversion
}
```

```
void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P1M0 = 0x00;                    //Set P1.0 as ADC input
    P1M1 = 0x01;
    P_SW2 |= 0x80;
    ADCTIM = 0x3f;                  // Set ADC internal timing
    P_SW2 &= 0x7f;
    ADCCFG = 0x0f;                  //Set the ADC clock to the system clock/2/16/16
    ADC_CONTR = 0x80;               //Enable ADC module
    EADC = 1;                       //Enable ADC interrupt
    EA = 1;
    ADC_CONTR |= 0x40;              //Start AD conversion

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

ADC_CONTR   DATA    0BCH
ADC_RES     DATA    0BDH
ADC_RESL    DATA    0BEH
ADCCFG      DATA    0DEH

P_SW2       DATA    0BAH
ADCTIM      XDATA   0FEA8H

EADC        BIT     IE.5

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     002BH
```

```
            LJMP    ADCISR

            ORG     0100H
ADCISR:
            ANL     ADC_CONTR,#NOT 20H      ;Clear completion flag
            MOV     P2,ADC_RES              ;Read ADC results
            ORL     ADC_CONTR,#40H          ;Continue AD conversion
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P1M0,#00H               ;Set P1.0 as ADC input
            MOV     P1M1,#01H
            MOV     P_SW2,#80H
            MOV     DPTR,#ADCTIM            ; Set ADC internal timing
            MOV     A,#3FH
            MOVX    @DPTR,A
            MOV     P_SW2,#00H
            MOV     ADCCFG,#0FH             ;Set the ADC clock to the system clock/2/16/16
            MOV     ADC_CONTR,#80H          ;Enable ADC module
            SETB    EADC                    ;Enable ADC interrupt
            SETB    EA
            ORL     ADC_CONTR,#40H          ;Start AD conversion

            SJMP    $

            END
```

16.6.3 Format ADC Conversion Result

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     ADC_CONTR   =   0xbc;
sfr     ADC_RES     =   0xbd;
sfr     ADC_RESL    =   0xbe;
sfr     ADCCFG      =   0xde;

sfr     P_SW2       =   0xba;
```

```
#define   ADCTIM      (*(unsigned char volatile xdata *)0xfea8)

sfr       P0M1    =   0x93;
sfr       P0M0    =   0x94;
sfr       P1M1    =   0x91;
sfr       P1M0    =   0x92;
sfr       P2M1    =   0x95;
sfr       P2M0    =   0x96;
sfr       P3M1    =   0xb1;
sfr       P3M0    =   0xb2;
sfr       P4M1    =   0xb3;
sfr       P4M0    =   0xb4;
sfr       P5M1    =   0xc9;
sfr       P5M0    =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P1M0 = 0x00;                                    //Set P1.0 as ADC input
    P1M1 = 0x01;
    P_SW2 |= 0x80;
    ADCTIM = 0x3f;                                  // Set ADC internal timing
    P_SW2 &= 0x7f;
    ADCCFG = 0x0f;                                  //Set the ADC clock to the system clock/2/16/16
    ADC_CONTR = 0x80;                               //Enable ADC module
    ADC_CONTR |= 0x40;                              //Start AD conversion
    _nop_();
    _nop_();
    while (!(ADC_CONTR & 0x20));                    //Query ADC completion flag
    ADC_CONTR &= ~0x20;                             //Clear completion flag

    ADCCFG = 0x00;                                  //Set result to align left
    ACC = ADC_RES;                                  //A stores the upper 8 bits of the ADC’ s 10-bit result
    B = ADC_RESL;           //B[7: 6] stores the lower 2 bits of the 10-bit ADC result, B [5: 0] is 0

//  ADCCFG = 0x20;                                  //Set result to align right
//  ACC = ADC_RES;      // A [1: 0] stores the upper 2 bits of the 10-bit result of the ADC, and A [7: 2] is 0
//  B = ADC_RESL;                                   //B stores the lower 8 bits of the ADC's 10-bit result

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

ADC_CONTR   DATA        0BCH
```

```
ADC_RES     DATA    0BDH
ADC_RESL    DATA    0BEH
ADCCFG      DATA    0DEH

P_SW2       DATA    0BAH
ADCTIM      XDATA   0FEA8H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P1M0,#00H           ;Set P1.0 as ADC input
            MOV     P1M1,#01H
            MOV     P_SW2,#80H
            MOV     DPTR,#ADCTIM        ; Set ADC internal timing
            MOV     A,#3FH
            MOVX    @DPTR,A
            MOV     P_SW2,#00H
            MOV     ADCCFG,#0FH         ;Set the ADC clock to the system clock/2/16/16
            MOV     ADC_CONTR,#80H      ;Enable ADC module

            ORL     ADC_CONTR,#40H      ;Start AD conversion
            NOP
            NOP
            MOV     A,ADC_CONTR         ;Query ADC completion flag
            JNB     ACC.5,$-2
            ANL     ADC_CONTR,#NOT 20H  ;Clear completion flag

            MOV     ADCCFG,#00H         ;Set result to align left
            MOV     A,ADC_RES           ;A stores the upper 8 bits of the ADC's 10-bit result
            MOV     B,ADC_RESL  ;B[7: 6] stores the lower 2 bits of the 10-bit ADC result, B [5: 0] is 0
```

STC MCU

```
;               MOV         ADCCFG,#20H             ;Set result to align right
;               MOV         A,ADC_RES ;A [3: 0] stores the upper 2 bits of the 10-bit result of the ADC, and A [7: 2] is 0
;               MOV         B,ADC_RESL              ;B stores the lower 8 bits of the ADC's 10-bit result

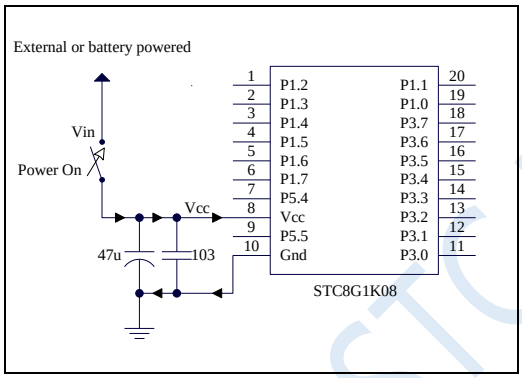
                SJMP        $

                END
```

16.6.4 Detect External Voltage or Battery Voltage using ADC 15th Channel

The 15th channel of ADC in the STC8G series of microcontrollers is used to measure the internal reference voltage. The internal reference voltage is stable, about 1.19V, and does not change with the chip's working voltage. So you can measure the internal reference voltage and use it to deduce the external voltage or external battery voltage through the value of the ADC.

The following figure is a reference circuit diagram.



C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC          11059200UL
#define   BRT           (65536 - FOSC / 115200 / 4)

sfr       AUXR          =   0x8e;

sfr       ADC_CONTR     =   0xbc;
sfr       ADC_RES       =   0xbd;
sfr       ADC_RESL      =   0xbe;
sfr       ADCCFG        =   0xde;

sfr       P_SW2         =   0xba;
#define   ADCTIM        (*(unsigned char volatile xdata *)0xfea8)

int       *BGV;                             //The internal reference voltage value is stored in idata
                                            //The high byte is stored in idata's EFH address
```

```
                                                    //The low byte is stored in idata's F0H address
                                                    //Voltage unit is millivolt (mV)
bit         busy;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void ADCInit()
{
    P_SW2 |= 0x80;
    ADCTIM = 0x3f;                                  // Set ADC internal timing
    P_SW2 &= 0x7f;

    ADCCFG = 0x2f;                                  //Set the ADC clock to the system clock/2/16/16
    ADC_CONTR = 0x8f;                               //Enable ADC module, and select channel 15
}

int    ADCRead()
{
    int res;

    ADC_CONTR |= 0x40;                              //Start AD conversion
    _nop_();
    _nop_();
    while (!(ADC_CONTR & 0x20));                    //Query ADC completion flag
    ADC_CONTR &= ~0x20;                             //Clear completion flag
    res = (ADC_RES << 8) | ADC_RESL;                //Read ADC results

    return res;
}
```

STC MCU

```
void main()
{
	int res;
	int vcc;
	int i;

	P0M0 = 0x00;
	P0M1 = 0x00;
	P1M0 = 0x00;
	P1M1 = 0x00;
	P2M0 = 0x00;
	P2M1 = 0x00;
	P3M0 = 0x00;
	P3M1 = 0x00;
	P4M0 = 0x00;
	P4M1 = 0x00;
	P5M0 = 0x00;
	P5M1 = 0x00;

	BGV = (int idata *)0xef;
	ADCInit();                                  //ADC initialization
	UartInit();                                 //UART initialization

	ES = 1;
	EA = 1;

//	ADCRead();
//	ADCRead();                                  //Discard the first two data
	res = 0;
	for (i=0; i<8; i++)
	{
		res += ADCRead();                       //Read data 8 times
	}
	res >>= 3;                                  //take the average

	vcc = (int)(4095L * *BGV / res);            //Calculate VREF pin voltage, i.e. battery voltage
	                                            //Note that this voltage is in millivolts (mV)
	UartSend(vcc >> 8);                         //Output voltage value to UART
	UartSend(vcc);

	while (1);
}
```

The above method uses the 15th channel of the ADC to reverse the external battery voltage. In the ADC measurement range, the external measurement voltage of the ADC is directly proportional to the measurement value of the ADC. Therefore, the 15th channel of the ADC can also be used to reverse the input voltage of the external channel. Assuming that the internal reference signal source voltage has been obtained as BGV, The ADC measurement value of the internal reference signal source is res_{bg}, and the ADC measurement value of the external channel input voltage is res_x, then the external channel input voltage $V_x = BGV / res_{bg} * res_x$;

16.6.5 Using ADC as Capacitive Sensing Touch Keys

Key is one of the most commonly used parts in the circuit, and it is an important input method for the human-machine interface. We are most familiar with mechanical keys. The mechanical keys have a disadvantage of limited contact life especially for the cheap keys. And they are easy to appear poor contact and failure. Non-contact keys have no mechanical contacts, long life and easy to use.

There are various solutions for non-contact keys. Capacitive-sensing keys are low-cost solutions. Specialized ICs were used to implement capacitive-sensing keys many years ago. With the enhancement of MCU functions and the practical experience of users, MCUs were used to implement capacitive-sensing keys directly. The technology of capacitive sensing keys is mature. The most typical and reliable one is the solution using ADC.

The solution of using STC series MCUs with ADC is described in detail in this document. Any MCU with ADC function can be used to implement the scheme. The first three diagrams below are the most commonly used methods. The principles are the same. The second diagram is used.

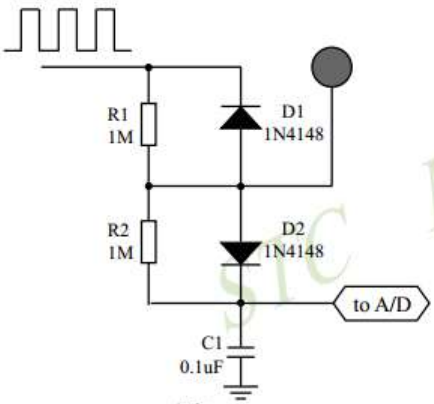


图1

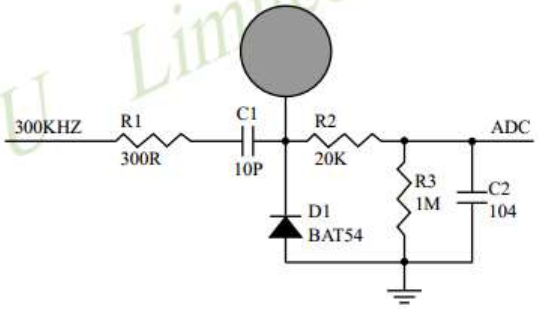


图2

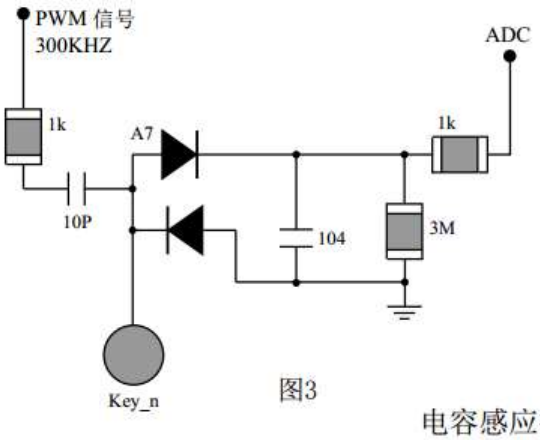


图3

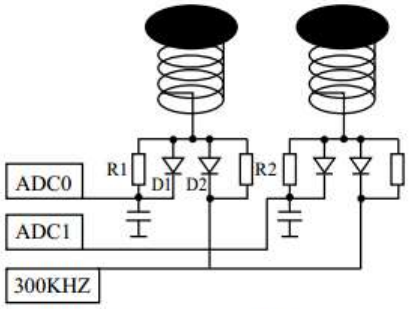
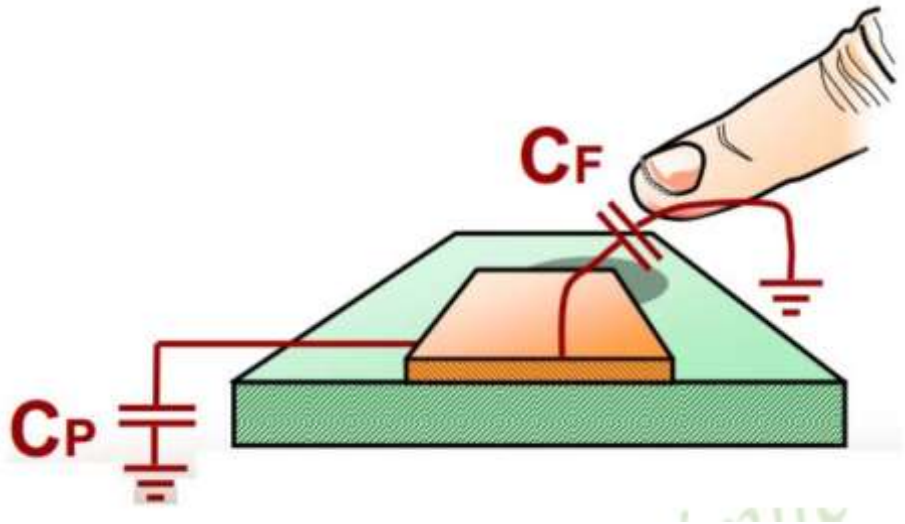


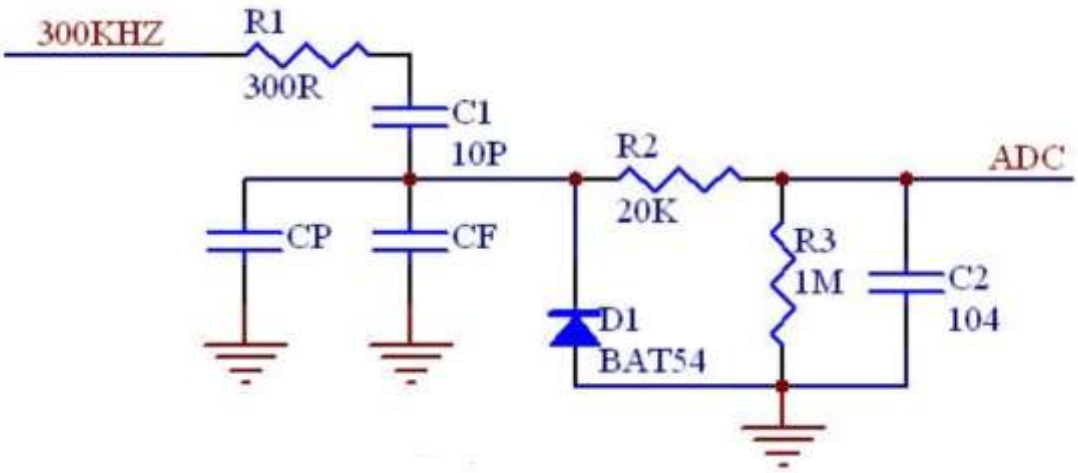
图4 加了感应弹簧

电容感应按键取样电路

In general applications, the induction spring shown in Figure 4 is used to increase the area pressed by a finger. The induction spring is equivalent to a metal plate to the ground. There is a capacitor CP to the ground. After pressing the finger, a capacitor CF is connected in parallel to the ground, as shown in the figure below.



The following is the description of the circuit diagram. CP is the distributed capacitance of metal plate and ground, CF is the finger capacitance, they are connected in parallel and connected with C1 to divide the input 300KHZ square wave. After being rectified by D1 and filtered by R2 and C2, the wave is sent to ADC. After pressing the finger, the voltage sent to the ADC decreases, and the program can detect the key action.



C language code

```
//Operating frequency for testing is 24MHz

#include "reg51.h"
#include "intrins.h"

#define   MAIN_Fosc     24000000UL                  //Define the main clock
#define   Timer0_Reload (65536UL -(MAIN_Fosc / 600000))  //Timer 0 reload value corresponds to 300KHz

typedef   unsigned char  u8;
typedef   unsigned int   u16;
typedef   unsigned long  u32;

sfr       P0M1         =    0x93;
sfr       P0M0         =    0x94;
sfr       P1M1         =    0x91;
sfr       P1M0         =    0x92;
sfr       P2M1         =    0x95;
```

```
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sfr     ADC_CONTR   =   0xBC;       //Series of microcontroller with ADC
sfr     ADC_RES     =   0xBD;       //Series of microcontroller with ADC
sfr     ADC_RESL    =   0xBE;       //Series of microcontroller with ADC
sfr     AUXR    =   0x8E;
sfr     AUXR2   =   0x8F;

#define CHANNEL     8               //ADC channel numbers
#define ADC_90T     (3<<5)          //ADC conversion time 90T
#define ADC_180T    (2<<5)          //ADC conversion time 180T
#define ADC_360T    (1<<5)          //ADC conversion time 360T
#define ADC_540T    0               //ADC conversion time 540T
#define ADC_FLAG    (1<<4)          //Cleared by software
#define ADC_START   (1<<3)          //Cleared automatically

sbit    P_LED7  =   P2^7;
sbit    P_LED6  =   P2^6;
sbit    P_LED5  =   P2^5;
sbit    P_LED4  =   P2^4;
sbit    P_LED3  =   P2^3;
sbit    P_LED2  =   P2^2;
sbit    P_LED1  =   P2^1;
sbit    P_LED0  =   P2^0;

u16 idata adc[TOUCH_CHANNEL];           //Current ADC value
u16 idata adc_prev[TOUCH_CHANNEL];      //Previous ADC value
u16 idata TouchZero[TOUCH_CHANNEL];     //ADC value of 0-point
u8 idata TouchZeroCnt[TOUCH_CHANNEL];   //Automatic tracking count for 0-point
u8 cnt_250ms;

void delay_ms(u8 ms);
void ADC_init(void);
u16 Get_ADC10bitResult(u8 channel);
void AutoZero(void);
u8 check_adc(u8 index);
void ShowLED(void);

void main(void)
{
    u8 i;

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
```

STC MCU

```
    P5M1 = 0x00;

    delay_ms(50);
    ET0 = 0;                                    //Initialize Timer0 to output a 300KHz clock
    TR0 = 0;
    AUXR |= 0x80;                               //Timer0 set as 1T mode
    AUXR2 |= 0x01;                              //Enable clock output
    TMOD = 0;                                   //Timer0 set as Timer, 16 bits Auto Reload.
    TH0 = (u8)(Timer0_Reload >> 8);
    TL0 = (u8)Timer0_Reload;
    TR0 = 1;
    ADC_init();                                 //ADC initialization
    delay_ms(50);                               //Delay 50ms
    for (i=0; i<TOUCH_CHANNEL; i++)  // Initialize the 0-point, the previous value and the 0-point auto-tracking count
    {
        adc_prev[i] = 1023;
        TouchZero[i] = 1023;
        TouchZeroCnt[i] = 0;
    }
    cnt_250ms = 0;
    while (1)
    {
        delay_ms(50);                           //Process key once every 50ms
        ShowLED();
        if (++cnt_250ms >= 5)
        {
            cnt_250ms = 0;
            AutoZero();                         //Process 0-point auto-tracking every 250ms
        }
    }
}

void delay_ms(u8 ms)
{
    unsigned int i;

    do
    {
        i = MAIN_Fosc / 13000;
        while(--i) ;
    } while(--ms);
}

void ADC_init(void)
{
    P1M0 = 0x00;                                //8 channels ADC
    P1M1 = 0xff;
    ADC_CONTR = 0x80;                           //Enable ADC
}

u16 Get_ADC10bitResult(u8 channel)
{
    ADC_RES = 0;
    ADC_RESL = 0;
    ADC_CONTR = 0x80 | ADC_90T | ADC_START | channel;   //Trigger ADC
    _nop_();
    _nop_();
    _nop_();
    _nop_();
```

STC MCU

```
    while((ADC_CONTR & ADC_FLAG) == 0) ;                    //Wait for ADC conversion complement
    ADC_CONTR = 0x80;                                       //Clear flag
    return(((u16)ADC_RES << 2) | ((u16)ADC_RESL & 3));      //Return ADC result
}

void AutoZero(void)                                         //Call once every 250ms
                        // This is detected using the sum of the absolute values of the differences between two adjacent samples.
{
    u8 i;
    u16 j,k;

    for(i=0; i<TOUCH_CHANNEL; i++)                          //Process 8 channels
    {
        j = adc[i];
        k = j - adc_prev[i];                                // Subtract previous reading
        F0 = 0;                                             //Pressed
        if(k & 0x8000) F0 = 1, k = 0 - k;                   //Release, get the difference between two samples
        if(k >= 20)                                         // Big change
        {
            TouchZeroCnt[i] = 0;                            // If the change is large, clear the counter
            if(F0) TouchZero[i] = j;    // If it is released, and the change is relatively large, then directly replace
        }
        else                        // If the change is relatively small, then creep, track 0-point automatically
        {
            if(++TouchZeroCnt[i] >= 20)                     // Continuously detect small changes 20 times/4 = 5 seconds.
            {
                TouchZeroCnt[i] = 0;
                TouchZero[i] = adc_prev[i];                 // Use slowly changing values as 0 points                }
        }
        adc_prev[i] = j;                                    // Save this time's sample value
    }
}

u8 check_adc(u8 index)                                      // Get touch information function, called every 50ms
                                                            // Judge key is pressed or released with hysteresis control
{
    u16 delta;

    adc[index] = 1023 - Get_ADC10bitResult(index);    // Get ADC value, convert to press the key, ADC value increases
    if(adc[index] < TouchZero[index]) return 0;             // A value smaller than 0-point is considered a key release
    delta = adc[index] - TouchZero[index];
    if(delta >= 40) return 1;                               //Key pressed
    if(delta <= 20) return 0;                               //Key released
    return 2;                                               // Keep the original state
}

void ShowLED(void)
{
    u8 i;

    i = check_adc(0);
    if(i == 0) P_LED0 = 1;                                  //Light off
    if(i == 1) P_LED0 = 0;                                  //Light on
    i = check_adc(1);
    if(i == 0) P_LED1 = 1;                                  //Light off
    if(i == 1) P_LED1 = 0;                                  //Light on
    i = check_adc(2);
```

STC MCU

```
    if(i == 0) P_LED2 = 1;                  //Light off
    if(i == 1) P_LED2 = 0;                  //Light on
    i = check_adc(3);
    if(i == 0) P_LED3 = 1;                  //Light off
    if(i == 1) P_LED3 = 0;                  //Light on
    i = check_adc(4);
    if(i == 0) P_LED4 = 1;                  //Light off
    if(i == 1) P_LED4 = 0;                  //Light on
    i = check_adc(5);
    if(i == 0) P_LED5 = 1;                  //Light off
    if(i == 1) P_LED5 = 0;                  //Light on
    i = check_adc(6);
    if(i == 0) P_LED6 = 1;                  //Light off
    if(i == 1) P_LED6 = 0;                  //Light on
    i = check_adc(7);
    if(i == 0) P_LED7 = 1;                  //Light off
    if(i == 1) P_LED7 = 0;                  //Light on
}
```

Assembly code

```
;Operating frequency for testing is 24MHz

Fosc_KHZ    EQU     24000               ;Define the main clock KHZ
Reload      EQU     (65536 - Fosc_KHZ/600)  ;Timer 0 reload value, corresponding to 300KHz

ADC_CONTR   DATA    0xBC                ;Series of microcontroller with ADC
ADC_RES     DATA    0xBD                ;Series of microcontroller with ADC
ADC_RESL    DATA    0xBE                ;Series of microcontroller with ADC
AUXR        DATA    0x8E
AUXR2       DATA    0x8F

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

CHANNEL     EQU     8                   ;ADC channel numbers
ADC_90T     EQU     (3 SHL 5)           ;ADC conversion time 90T
ADC_180T    EQU     (2 SHL 5)           ;ADC conversion time 180T
ADC_360T    EQU     (1 SHL 5)           ;ADC conversion time 360T
ADC_540T    EQU     0                   ;ADC conversion time 540T
ADC_FLAG    EQU     (1 SHL 4)           ;Cleared by software
ADC_START   EQU     (1 SHL 3)           ;Cleared automatically

P_LED7      BIT     P2.7;
P_LED6      BIT     P2.6;
P_LED5      BIT     P2.5;
P_LED4      BIT     P2.4;
P_LED3      BIT     P2.3;
```

```
P_LED2      BIT     P2.2;
P_LED1      BIT     P2.1;
P_LED0      BIT     P2.0;
adc         EQU     30H     ; Current ADC value in 30H ~ 3FH, two bytes constitute one value

adc_prev    EQU     40H     ; Previous ADC value in 40H ~ 4FH, two bytes constitute a value
TouchZero   EQU     50H                 ; ADC 0 value in 50H~5FH, two bytes constitute a value
TouchZeroCnt EQU    60H                 ; 0-point automatic tracking count in 60H~67H
cnt_250ms   DATA    68H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#0D0H
            MOV     P0M0,#00H
            MOV     P0M1,#00H
            MOV     P1M0,#00H
            MOV     P1M1,#00H
            MOV     P2M0,#00H
            MOV     P2M1,#00H
            MOV     P3M0,#00H
            MOV     P3M1,#00H
            MOV     P4M0,#00H
            MOV     P4M1,#00H
            MOV     P5M0,#00H
            MOV     P5M1,#00H

            MOV     R7,#50
            LCALL   F_delay_ms
            CLR     ET0             ;Initialize Timer0 to output a 300KHz clock
            CLR     TR0
            ORL     AUXR,#080H      ;Timer0 set as 1T mode
            ORL     AUXR2,#01H      ;Enable clock output
            MOV     TMOD,#0         ;Timer0 set as Timer,16 bits Auto Reload.
            MOV     TH0,#HIGH Reload
            MOV     TL0,#LOW Reload
            SETB    TR0
            LCALL   F_ADC_init
            MOV     R7,#50
            LCALL   F_delay_ms
            MOV     R0,#adc_prev    ;Initialize the previous ADC value
L_Init_Loop1:
            MOV     @R0,#03H
            INC     R0
            MOV     @R0,#0FFH
            INC     R0
            MOV     A,R0
            CJNE    A,#(adc_prev + CHANNEL * 2),L_Init_Loop1
            MOV     R0,#TouchZero   ;Initialize the ADC 0-point value
L_Init_Loop2:
            MOV     @R0,#03H
            INC     R0
            MOV     @R0,#0FFH
            INC     R0
            MOV     A,R0
            CJNE    A,#(TouchZero+CHANNEL * 2),L_Init_Loop2
            MOV     R0,#TouchZeroCnt    ;Initialize the automatic tracking count value
```

STC MCU

```
L_Init_Loop3:
        MOV     @R0,#0
        INC     R0
        MOV     A,R0
        CJNE    A,#(TouchZeroCnt + CHANNEL),L_Init_Loop3
        MOV     cnt_250ms,#5
L_MainLoop:
        MOV     R7,#50                  ;Delay 50ms
        LCALL   F_delay_ms
        LCALL   F_ShowLED               ;Handle key value once
        DJNZ    cnt_250ms,L_MainLoop
        MOV     cnt_250ms,#5    ;Processing once 0-point automatic tracking value every 250ms
        LCALL   F_AutoZero              ; Zero tracking
        SJMP    L_MainLoop

F_ADC_init:
        MOV     P1M0,#00H               ;8 channels ADC
        MOV     P1M1,#0FFH
        MOV     ADC_CONTR,#080H         ;Enable ADC
        RET

F_Get_ADC10bitResult:
        MOV     ADC_RES,#0
        MOV     ADC_RESL,#0
        MOV     A,R7
        ORL     A,#0E8H                 ;Trigger ADC
        MOV     ADC_CONTR,A
        NOP
        NOP
        NOP
        NOP
L_10bitADC_Loop1:
        MOV     A,ADC_CONTR
        JNB     ACC.4,L_10bitADC_Loop1  ;Wait for the ADC conversion complement
        MOV     ADC_CONTR,#080H         ;Clear flag
        MOV     A,ADC_RES
        MOV     B,#04H
        MUL     AB
        MOV     R7,A
        MOV     R6,B
        MOV     A,ADC_RESL
        ANL     A,#03H
        ORL     A,R7
        MOV     R7,A
        RET

F_AutoZero:                             ; Call once every 250ms
        ; This is detected using the sum of the absolute values of the differences between two adjacent samples.
        CLR     A
        MOV     R5,A
L_AutoZero_Loop:
        MOV     A,R5
        ADD     A,ACC
        ADD     A,#LOW (adc)
        MOV     R0,A
        MOV     A,@R0
        MOV     R6,A
        INC     R0
        MOV     A,@R0
```

```
                MOV     R7,A
                MOV     A,R5
                ADD     A,ACC
                ADD     A,#LOW (adc_prev+01H)
                MOV     R0,A
                CLR     C
                MOV     A,R7
                SUBB    A,@R0
                MOV     R3,A
                MOV     A,R6
                DEC     R0
                SUBB    A,@R0
                MOV     R2,A
                CLR     F0 ;按下
                JNB     ACC.7,L_AutoZero_1
                SETB    F0
                CLR     C
                CLR     A
                SUBB    A,R3
                MOV     R3,A
                MOV     A,R3
                CLR     A
                SUBB    A,R2
                MOV     R2,A
L_AutoZero_1:
                CLR     C                       ;Calculate [R2 R3] -   #20,if(k >= 20)
                MOV     A,R3
                SUBB    A,#20
                MOV     A,R2
                SUBB    A,#00H
                JC      L_AutoZero_2            ;[R2 R3] ,20, Jump
                MOV     A,#LOW (TouchZeroCnt)   ; If the change is large, clear the counter TouchZeroCnt[i] =
0;
                ADD     A,R5
                MOV     R0,A
                MOV     @R0,#0
                JNB     F0,L_AutoZero_3
                MOV     A,R5
                ADD     A,ACC
                ADD     A,#LOW (TouchZero)
                MOV     R0,A
                MOV     @R0,6
                INC     R0
                MOV     @R0,7
                SJMP    L_AutoZero_3
L_AutoZero_2:           ; If the change is relatively small, then creep, track 0-point automatically
                                                ; Continuously detect small changes 20 times/4 = 5 seconds.
                MOV     A,#LOW (TouchZeroCnt)
                ADD     A,R5
                MOV     R0,A
                INC     @R0
                MOV     A,@R0
                CLR     C
                SUBB    A,#20
                JC      L_AutoZero_3            ;if(TouchZeroCnt[i] < 20), jump
                MOV     @R0,#0                  ;TouchZeroCnt[i]= 0;
                MOV     A,R5                    ; Use slowly changing values as 0 points
                ADD     A,ACC
                ADD     A,#LOW (adc_prev)
```

```
                MOV     R0,A
                MOV     A,@R0
                MOV     R2,A
                INC     R0
                MOV     A,@R0
                MOV     R3,A
                MOV     A,R5
                ADD     A,ACC
                ADD     A,#LOW (TouchZero)
                MOV     R0,A
                MOV     @R0,2
                INC     R0
                MOV     @R0,3
L_AutoZero_3:                           ; Save the sampled value adc_prev[i] = j;
                MOV     A,R5
                ADD     A,ACC
                ADD     A,#LOW (adc_prev)
                MOV     R0,A
                MOV     @R0,6
                INC     R0
                MOV     @R0,7
                INC     R5
                MOV     A,R5
                XRL     A,#08H
                JZ      $ + 5H
                LJMP    L_AutoZero_Loop
                RET

F_check_adc:                            ; Judge key is pressed or released, with hysteresis control
                MOV R4,7
                LCALL   F_Get_ADC10bitResult    ; The ADC value returned is [R6 R7]
                CLR     C
                MOV     A,#0FFH
                SUBB    A,R7
                MOV     R7,A
                MOV     A,#03H
                SUBB    A,R6
                MOV     R6,A
                MOV     A,R4            ;Save adc[index]
                ADD     A,ACC
                ADD     A,#LOW (adc)
                MOV     R0,A
                MOV     @R0,6
                INC     R0
                MOV     @R0,7
                MOV     A,R4
                ADD     A,ACC
                ADD     A,#LOW (TouchZero+01H)
                MOV     R1,A
                MOV     A,R4
                ADD     A,ACC
                ADD     A,#LOW (adc)
                MOV     R0,A
                MOV     A,@R0
                MOV     R6,A
                INC     R0
                MOV     A,@R0
                CLR     C
                SUBB    A,@R1           ;Calculate adc[index] - TouchZero[index]
```

```
                MOV     A,R6
                DEC     R1
                SUBB    A,@R1
                JNC     L_check_adc_1
                MOV     R7,#00H
                RET
L_check_adc_1:
                MOV     A,R4
                ADD     A,ACC
                ADD     A,#LOW (TouchZero+01H)
                MOV     R1,A
                MOV     A,R4
                ADD     A,ACC
                ADD     A,#LOW (adc+01H)
                MOV     R0,A
                CLR     C
                MOV     A,@R0
                SUBB    A,@R1
                MOV     R7,A
                DEC     R0
                MOV     A,@R0
                DEC     R1
                SUBB    A,@R1
                MOV     R6,A
                CLR     C
                MOV     A,R7
                SUBB    A,#40
                MOV     A,R6
                SUBB    A,#00H
                JC      L_check_adc_2       ;if(delta < 40), jump
                MOV     R7,#1               ;if(delta >= 40) return 1; //Key pressed, return 1
                RET
L_check_adc_2:
                SETB    C
                MOV     A,R7
                SUBB    A,#20
                MOV     A,R6
                SUBB    A,#00H
                JNC     L_check_adc_3
                MOV     R7,#0
                RET
L_check_adc_3:
                MOV     R7,#2
                RET

F_ShowLED:
                MOV     R7,#0
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck0
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED0,C
L_QuitCheck0:
                MOV     R7,#1
                LCALL   F_check_adc
                MOV     A,R7
```

```
                ANL     A,#0FEH
                JNZ     L_QuitCheck1
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED1,C
L_QuitCheck1:
                MOV     R7,#2
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck2
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED2,C
L_QuitCheck2:
                MOV     R7,#3
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck3
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED3,C
L_QuitCheck3:
                MOV     R7,#4
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck4
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED4,C
L_QuitCheck4:
                MOV     R7,#5
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck5
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED5,C
L_QuitCheck5:
                MOV     R7,#6
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck6
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED6,C
L_QuitCheck6:
                MOV     R7,#7
                LCALL   F_check_adc
```

STC MCU

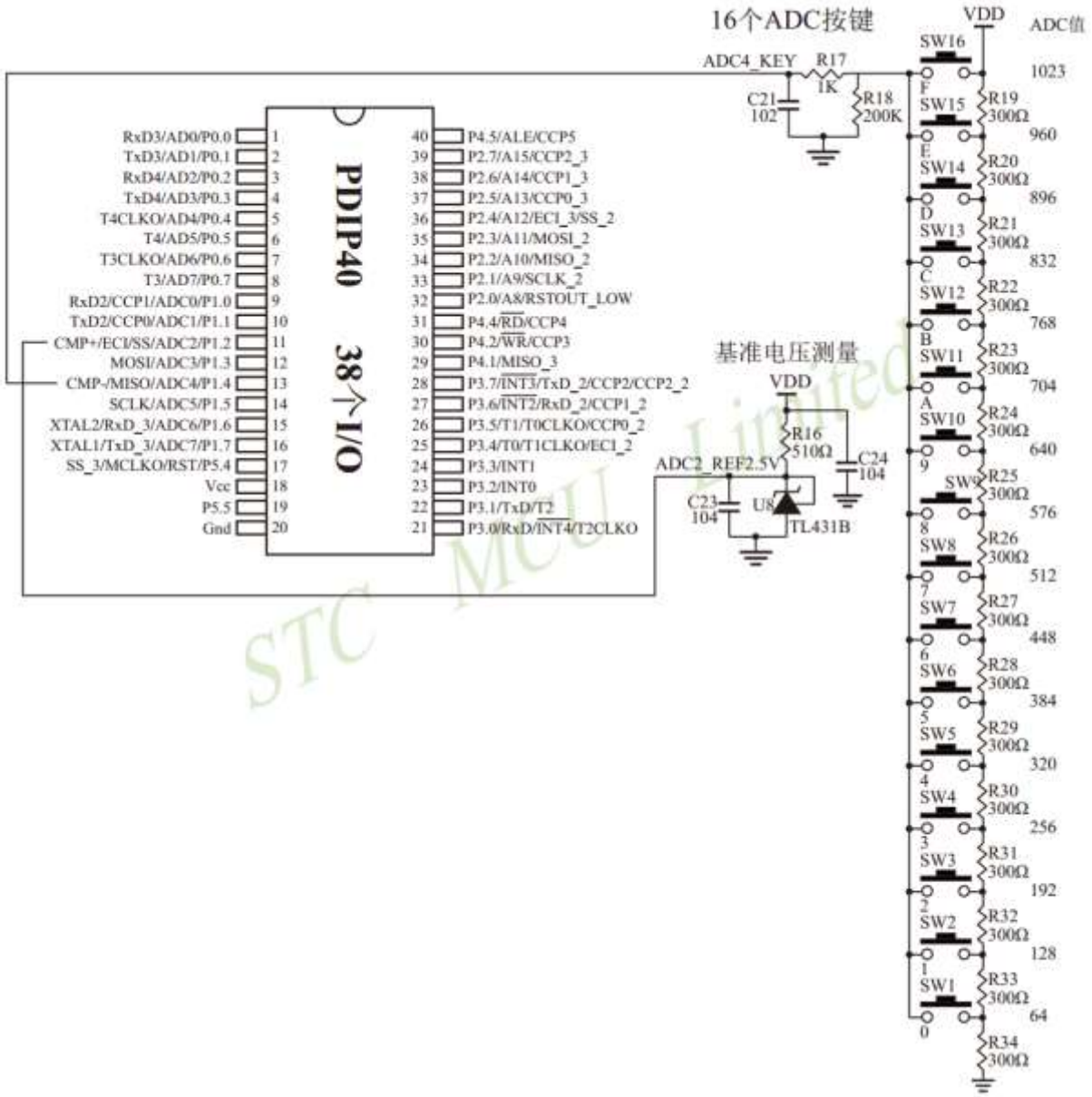
```
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck7
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED7,C
L_QuitCheck7:
                RET

F_delay_ms:
                PUSH    3
                PUSH    4
L_delay_ms_1:
                MOV     R3,#HIGH (Fosc_KHZ / 13)
                MOV     R4,#LOW (Fosc_KHZ / 13)
L_delay_ms_2:
                MOV     A,R4
                DEC     R4
                JNZ     L_delay_ms_3
                DEC     R3
L_delay_ms_3:
                DEC     A
                ORL     A,R3
                JNZ     L_delay_ms_2
                DJNZ    R7,L_delay_ms_1
                POP     4
                POP     3
                RET

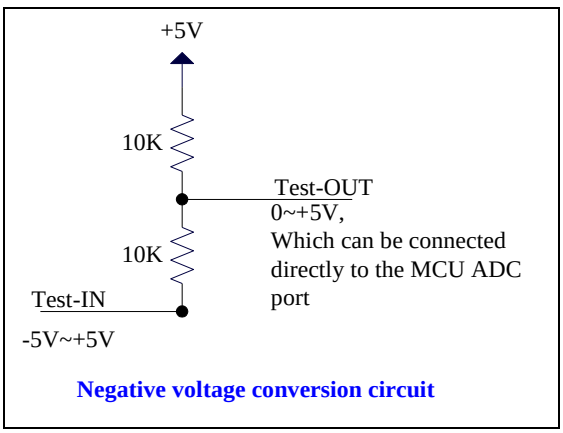
                END
```

16.6.6 Key-scan Application Circuit Diagram using ADC

Method for reading the ADC key: Read the ADC value every 10ms or so and save the last 3 readings. If the change is relatively small, judge the key. When the key is judged be valid, a certain deviation is allowed, such as a deviation of ±16 words.

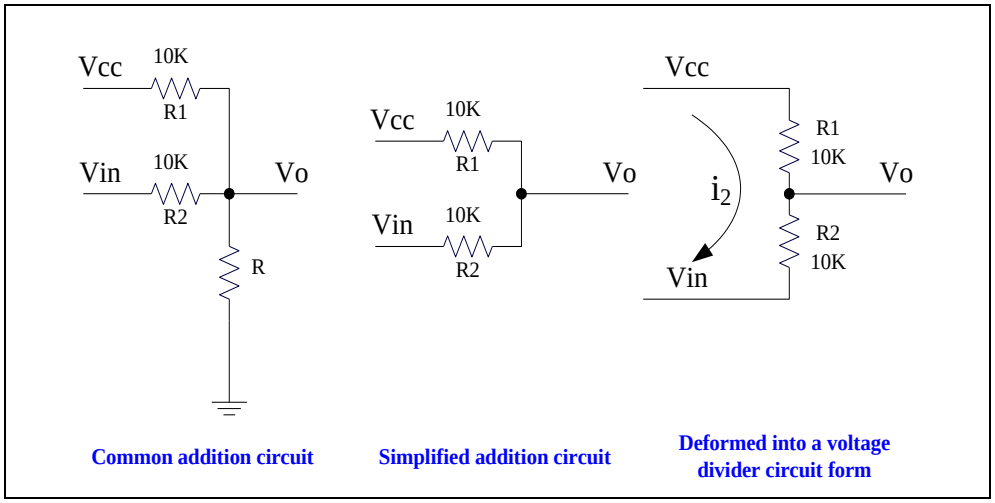


16.6.7 Reference circuit diagram for detecting negative voltage



Negative voltage conversion circuit

16.6.8 The Application of Common Adding Circuits in ADC



Refer to the voltage divider circuit to get formula 1

Formula 1: Vo = Vin + i2 * R2

Formula 2: i2 = (Vcc-Vin) / (R1 + R2) {Condition: current flowing to Vo ≈0}

Substituting R1=R2 into formula 2 gives formula 3

Formula 3: i2 = (Vcc-Vin) / 2R2

Substituting formula 3 into formula 1 gives formula 4

Formula 4: Vo = (Vcc + Vin) / 2

According to formula 4, the above circuit can be regarded as an addition circuit.

In the analog-to-digital conversion measurement of the microcontroller, the measured voltage is required to be greater than 0 and less than VCC. If the measured voltage is less than 0V, an addition circuit can be used to increase the measured voltage to above 0V. At this time, there are certain requirements for the variation range of the measured voltage:

Substituting the above conditions into formula 4, the following formula 2 can be obtained

(Vcc + Vin) / 2> 0 means Vin> -Vcc

(Vcc + Vin) / 2 <Vcc i.e. Vin <Vcc

The above 2 formulas can be combined: -Vcc <Vin <Vcc

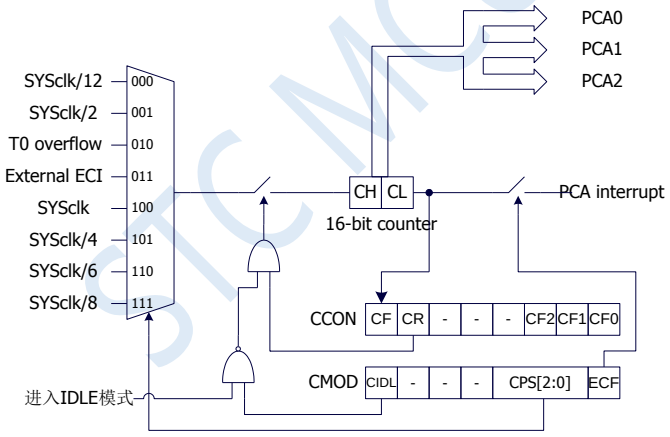
STC MCU

17 Application of PCA/CCP/PWM

Product line	PCA
STC8G1K08 family	●
STC8G1K08-8Pin family	
STC8G1K08A family	●
STC8G2K64S4 family	●
STC8G2K64S2 family	●
STC8G1K08T family	●
STC15H2K64S4 family	●

Three groups of programmable counter array (PCA/CCP/PWM) modules are integrated in STC8G series of microcontrollers, which can be used for software timer, external pulse capture, high-speed pulse output and pulse width modulation (PWM) output.

PCA contains a special 16-bit counter, with which three groups of PCA modules are connected. The structure of PCA counter is as follows:



Structure of PCA counter

17.1 Registers Related to PCA

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
CCON	PCA Control Register	D8H	CF	CR	-	-	-	CCF2	CCF1	CCF0	00xx,x000
CMOD	PCA Mode Register	D9H	CIDL	-	-	-	CPS[2:0]			ECF	0xxx,0000
CCAPM0	PCA 0 Mode Control Register	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAPM1	PCA 1 Mode Control Register	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CCAPM2	PCA 2 Mode Control Register	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
CL	PCA Counter Low Byte	E9H									0000,0000
CCAP0L	PCA 0 Capture Register Low Byte	EAH									0000,0000
CCAP1L	PCA 1 Capture Register Low Byte	EBH									0000,0000
CCAP2L	PCA 2 Capture Register Low Byte	ECH									0000,0000
PCA_PWM0	PCA0 PWM Mode Register	F2H	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L	0000,0000
PCA_PWM1	PCA1 PWM Mode Register	F3H	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L	0000,0000
PCA_PWM2	PCA2 PWM Mode Register	F4H	EBS2[1:0]		XCCAP2H[1:0]		XCCAP2L[1:0]		EPC2H	EPC2L	0000,0000
CH	PCA Counter High Byte	F9H									0000,0000
CCAP0H	PCA 0 Capture Register High Byte	FAH									0000,0000
CCAP1H	PCA 1 Capture Register High Byte	FBH									0000,0000
CCAP2H	PCA 1 Capture Register High Byte	FCH									0000,0000

17.1.1 PCA control register (CCON)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
CCON	D8H	CF	CR	-	-	-	CCF2	CCF1	CCF0

CF: PCA Counter overflow flag. It is set by hardware when the 16-bit counter of PCA overflows, and requests interrupt to CPU. It must be cleared by software.

CR: PCA counter enable bit.

0: Stop PCA counting.

1: Start PCA counting.

CCFn(n=0,1,2): PCA interrupt flag. When a match or a capture occurs on the PCA module, the corresponding flag bit is set by the hardware automatically and requests an interrupt to CPU. These flags should be cleared by software.

17.1.2 PCA mode register (CMOD)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	D9H	CIDL	-	-	-	CPS[2:0]			ECF

CIDL: PCA Counter control bit in Idle mode.

0: the PCA counter will continue counting in idle mode.

1: the PCA counter will stop counting in idle mode.

CPS[2:0]: PCA Counter pulse source select bits.

CPS[2:0]	Input clock source of PCA	Precautions
000	System clock/12	
001	System clock/2	
010	Overflow pulse of timer 0	
011	External clock input from ECI pin	The external clock frequency cannot be higher than 1/2 of the system frequency.
100	System clock	
101	System clock/4	
110	System clock/6	
111	System clock/8	

ECF: PCA counter overflow interrupt enable bit

0: disable PCA counter overflow interrupt

1: enable PCA counter overflow interrupt

17.1.3 PCA counter registers (CL, CH)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
CL	E9H								
CH	F9H								

The 16-bit counter is the combination of CL and CH, where CL is the low 8-bit counter and CH is the high 8-bit counter. The 16-bit counter of PCA increments automatically every one PCA clock.

17.1.4 PCA mode control registers (CCAPMn)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
CCAPM0	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0
CCAPM1	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1
CCAPM2	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2

ECOMn: PCAn Comparator enable bit

CCAPPn: PCAn Capture on rising edge enable bit

CCAPNn: PCAn Capture on falling edge enable bit

MATn: PCAn match function enable bit

TOGn: PCAn high speed pulse output function enable bit

PWMn: PCAn PWM output function enable bit

ECCFn: PCAn match/capture interrupt enable bit

17.1.5 PCA capture value/compare value registers (CCAPnL, CCAPnH)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
CCAP0L	EAH								
CCAP1L	EBH								
CCAP2L	ECH								
CCAP0H	FAH								
CCAP1H	FBH								
CCAP2H	FCH								

When the PCA capture function is enabled, CCAPnL and CCAPnH are used to save the count value (CL and CH) of the PCA at the time of capture. When the PCA comparison function is enabled, the PCA controller compares the current value in [CH,CL] and the value in [CCAPnH, CCAPnL], and the comparison result is given. When the PCA match function is enabled, the PCA controller compares the current value in [CH, CL] with the value stored in [CCAPnH, CCAPnL], and checks if they match (equal), then gives a match result.

17.1.6 PCA PWM mode control registers (PCA_PWMn)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PCA_PWM0	F2H	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L
PCA_PWM1	F3H	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L
PCA_PWM2	F4H	EBS2[1:0]		XCCAP2H[1:0]		XCCAP2L[1:0]		EPC2H	EPC2L

EBSn[1:0]: PCAn PWM number of bits control

EBSn[1:0]	PWM bits	Reload value	Comparison value
00	8-bits PWM	{EPCnH, CCAPnH[7:0]}	{EPCnL, CCAPnL[7:0]}
01	7-bits PWM	{EPCnH, CCAPnH[6:0]}	{EPCnL, CCAPnL[6:0]}
10	6-bits PWM	{EPCnH, CCAPnH[5:0]}	{EPCnL, CCAPnL[5:0]}
11	10-bits PWM	{EPCnH, XCCAPnH[1:0], CCAPnH[7:0]}	{EPCnL, XCCAPnL[1:0], CCAPnL[7:0]}

XCCAPnH[1:0]: The 9th bit and 10th bit of reload value of 10-bit PWM

XCCAPnL[1:0]: The 9th bit and 10th bit of comparison value of 10-bit PWM

EPCnH: The MSB of reload vaule in PWM mode (i.e. the 9th bit of 8-bit PWM, the 8th bit of 7-bit PWM, the 7th bit of 6-bit PWM, the 11th bit of 10-bit PWM)

EPCnL: The MSB of comparison vaule in PWM mode (i.e. the 9th bit of 8-bit PWM, the 8th bit of 7-bit PWM, the 7th bit of 6-bit PWM, the 11th bit of 10-bit PWM)

Note: When updating the reload value of 10-bit PWM, write the upper two bits of XCCAPnH [1: 0] firstly and then the lower 8 bits of CCAPnH [7: 0].

17.2 PCA Operation Mode

There are 3 groups of PCA modules in STC8G series of microcontrollers, and operation mode of each module can be set independently. The mode settings are as follows:

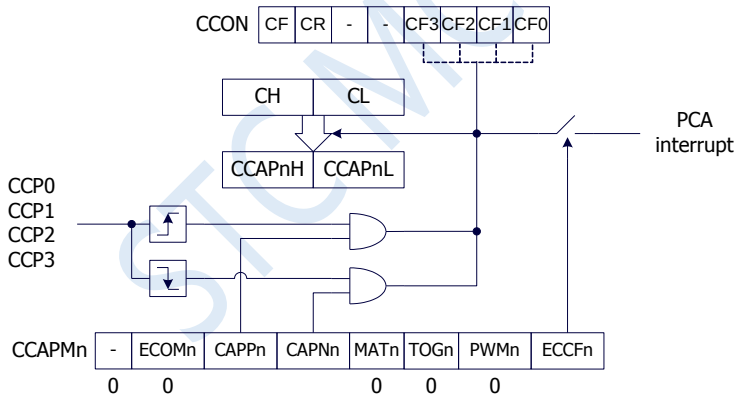
CCAPMn								Function of module
-	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	
-	0	0	0	0	0	0	0	No operation
-	1	0	0	0	0	1	0	6/7/8/10 bit PWM mode, no interrupt
-	1	1	0	0	0	1	1	6/7/8/10 bit PWM mode, rising edge interrupt
-	1	0	1	0	0	1	1	6/7/8/10 bit PWM mode, falling edge

								interrupt
-	1	1	1	0	0	1	1	6/7/8/10 bit PWM mode, rising and falling edge interrup
-	0	1	0	0	0	0	x	16 bit rising edge capture mode
-	0	0	1	0	0	0	x	16 bit falling edge capture mode
-	0	1	1	0	0	0	x	16 bit rising and falling edge capture mode
-	1	0	0	1	0	0	x	16 bit software timer mode
-	1	0	0	1	1	0	x	16 bit high speed pulse output mode

17.2.1 Capture Mode

At least one of CAPNn and CAPPn bits in CCAPMn must be set (or all of them are set) for a PCA module to operate in capture mode. When a PCA module is operating in capture mode, the input hoppings on the external CCP0/CCP1/CCP2 pins of the module are sampled. When a valid hopping is sampled, the PCA controller immediately loads the counter values in the PCA counters, CH and CL, into the module's capture registers, CCAPnL and CCAPnH, and sets the corresponding CCFn in the CCON register. If the ECCFn bit in CCAPMn is set to 1, an interrupt will be generated. Since all PCA modules' interrupt entry addresses are shared, it is necessary to determine which module generated an interrupt in the interrupt service routine and note that the interrupt flag bit must be cleared by software.

The structure of the PCA module working in capture mode is shown in the following figure.

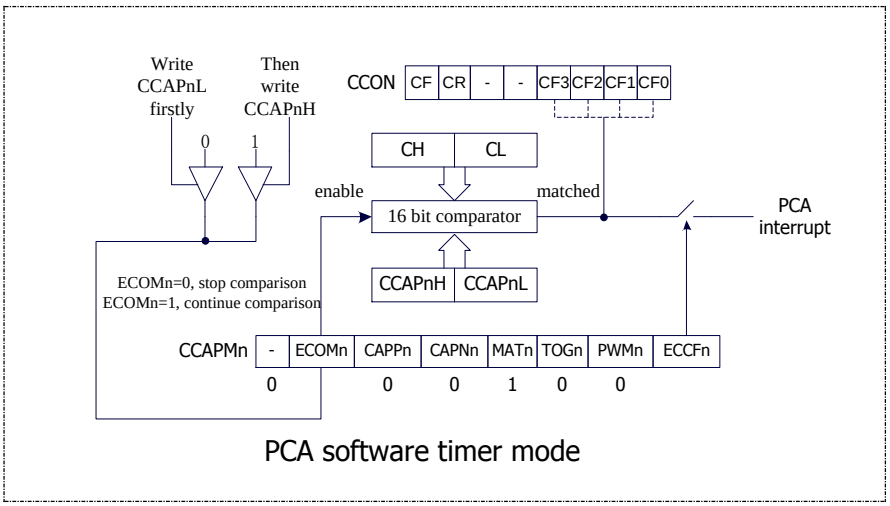


PCA capture mode

17.2.2 Software Timer Mode

The PCA module can be used as a software timer by setting the ECOM and MAT bits in the CCAPMn register. The PCA counter value in CL and CH is compared with the capture registers value in CCAPnL and CCAPnH. When they are equal, CCFn in CCON is set and an interrupt is generated if ECCFn in CCAPMn is set to 1. CCFn flag bit should be cleared by software.

The structure of PCA module working in software timer mode is shown in the following figure.

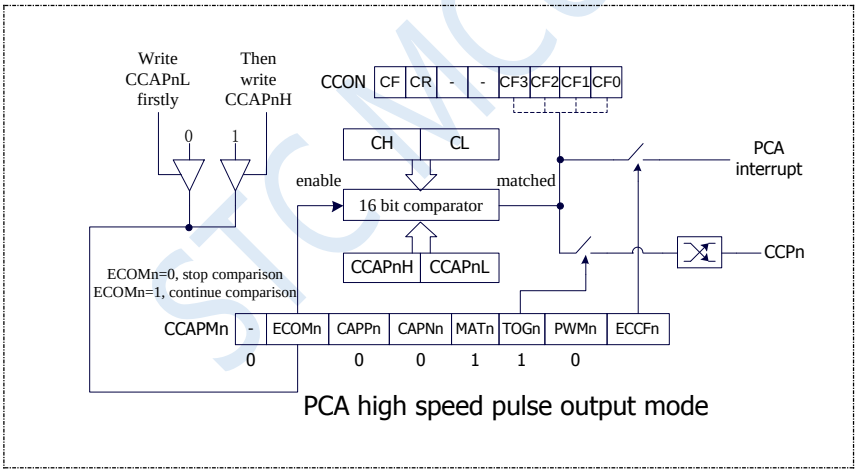


PCA software timer mode

17.2.3 High Speed Pulse Output Mode

When the count value of the PCA counter matches the value of the capture register, the CCPn output of the PCA module will hop. To activate the high speed pulse output mode, the TOGn, MATn, and ECOMn bits of the CCAPMn register must be set.

The structure of PCA module working in high-speed pulse output mode is shown below.



PCA high speed pulse output mode

17.2.4 Pulse Width Modulation Mode (PWM mode)

17.2.4.1 8-bit PWM Mode

Pulse width modulation is a technique that uses a program to control the duty ratio, cycle and phase of a waveform. It is widely used in applications such as three-phase motor driving and D/A conversion. The PCA modules of the STC8G series of microcontrollers can be configured to operate in 8-bit, 7-bit, 6-bit or 10-bit PWM mode by setting corresponding PCA_PWMn registers. To enable the PWM function of the PCA module, the PWMn and ECOMn bits of the module register CCAPMn must be set.

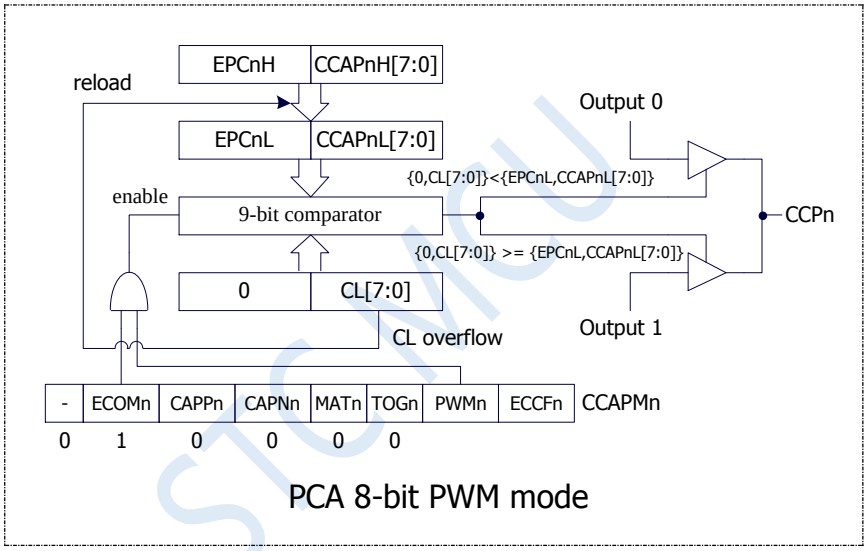
When EBSn [1:0] in the PCA_PWMn register is set to 00, PCAn operates in 8-bit PWM mode, where {0, CL [7: 0]} is compared with the capture registers {EPCnL, CCAPnL [7: 0]}. When PCA modules are operating

in 8-bit PWM mode, the output frequencies of them are the same because all the modules share a single PCA counter. The output duty ratio of each module is set using the registers {EPCnL, CCAPnL [7: 0]}. The output is low when the value of {0, CL [7: 0]} is less than {EPCnL, CCAPnL [7: 0]}, and the output is high when the value of {0, CL [7: 0]} is equal to or greater than {EPCnL , CCAPnL [7: 0]}. When CL [7: 0] overflows from FF to 00, the contents of {EPCnH, CCAPnH [7: 0]} are reloaded into {EPCnL, CCAPnL [7: 0]}. This makes it possible to update the PWM without interference.

$$\text{PWM frequency in 8-bit mode} = \frac{\text{PCA input clock source frequency}}{256}$$

When EPCnH=0 and CCAPnH=00H, PWM fixed output high
When EPCnH=1 and CCAPnH=FFH, PWM fixed output low

The structure of PCA module working in 8-bit PWM mode is shown below.



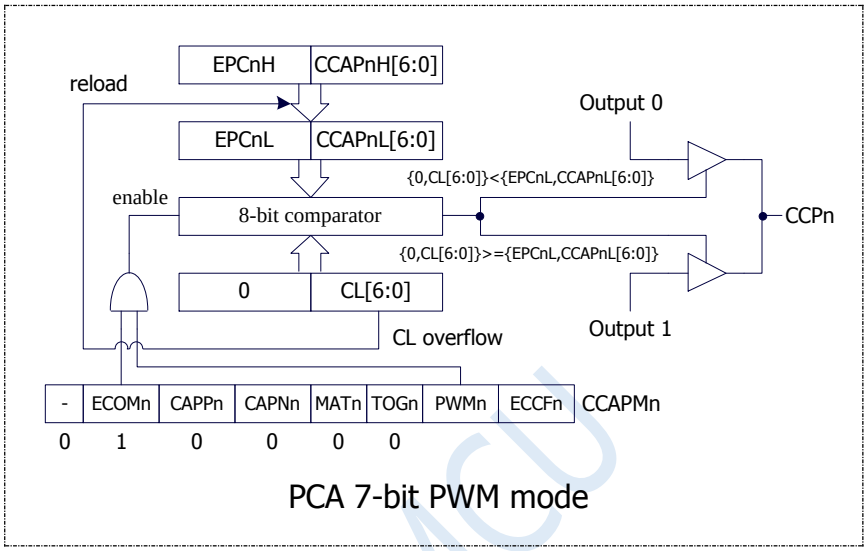
17.2.4.2 7-bit PWM Mode

When EBSn [1:0] in the PCA_PWMn register is set to 01, the PCAn operates in 7-bit PWM mode, where {0, CL [6: 0]} is compared with the capture registers {EPCnL, CCAPnL [6: 0]}. When PCA modules are operating in 7-bit PWM mode, the output frequencies of them are the same because all the modules share a single PCA counter. The output duty ratio of each module is set using the registers {EPCnL, CCAPnL [6: 0]}. The output is low when the value of {0, CL [6: 0]} is less than {EPCnL, CCAPnL [6: 0]}, and the output is high when the value of {0, CL [6: 0]} is equal to or greater than {EPCnL , CCAPnL [6: 0]}. When CL [6: 0] overflows from 7F to 00, the contents of {EPCnH, CCAPnH [6: 0]} are reloaded into {EPCnL, CCAPnL [6: 0]}. This makes it possible to update the PWM without interference.

$$\text{PWM frequency in 7-bit mode} = \frac{\text{PCA input clock source frequency}}{128}$$

When EPCnH=0 and CCAPnH=00H, PWM fixed output high
When EPCnH=1 and CCAPnH=FFH, PWM fixed output low

The structure of PCA module working in 7-bit PWM mode is shown below.



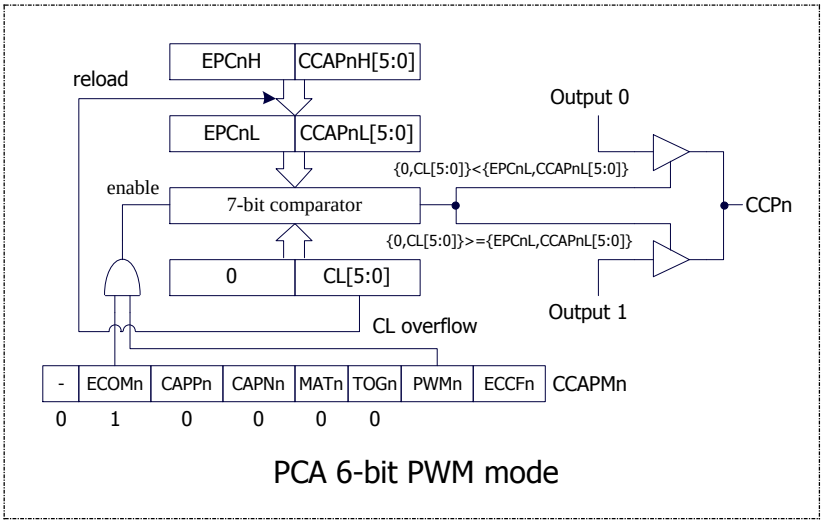
17.2.4.3 6-bit PWM Mode

When EBSn [1: 0] in the PCA_PWMn register is set to 10, PCAn operates in 6-bit PWM mode, where {0, CL [5: 0]} is compared with the capture registers {EPCnL, CCAPnL [5: 0] }. When PCA modules are operating in 6-bit PWM mode, the output frequencies of them are the same because all the modules share a single PCA counter. The output duty ratio of each module is set using the registers {EPCnL, CCAPnL [5: 0]}. The output is low when the value of {0, CL [5: 0]} is less than {EPCnL, CCAPnL [5: 0]}, and the output is high when the value of {0, CL [5: 0]} is equal to or greater than {EPCnL , CCAPnL [5: 0]}. When CL [5: 0] overflows from 3F to 00, the contents of {EPCnH, CCAPnH [5: 0]} are reloaded into {EPCnL, CCAPnL [5: 0]}. This makes it possible to update the PWM without interference.

$$\text{PWM frequency in 6-bit mode} = \frac{\text{PCA input clock source frequency}}{64}$$

When EPCnH=0 and CCAPnH=00H, PWM fixed output high
When EPCnH=1 and CCAPnH=FFH, PWM fixed output low

The structure of PCA module working in 6-bit PWM mode is shown below.



PCA 6-bit PWM mode

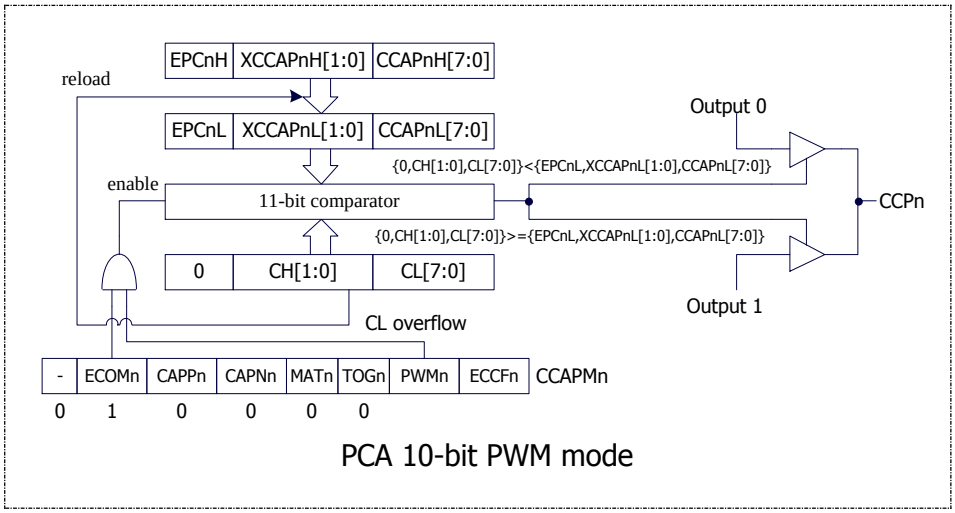
17.2.4.4 10-bit PWM Mode

When EBSn [1: 0] in the PCA_PWMn register is set to 11, PCAn operates in 10-bit PWM mode, where {CH[1:0],CL[7:0]} is compared with the capture registers{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}. When PCA modules are operating in 10-bit PWM mode, the output frequencies of them are the same because all the modules share a single PCA counter. The output duty ratio of each module is set using the registers {EPCnL, XCCAPnL[1:0],CCAPnL[7:0]}. The output is low when the value of {CH[1:0],CL[7:0]} is less than {EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}, and the output is high when the value of {CH[1:0],CL[7:0]} is equal to or greater than {EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}. When {CH[1:0],CL[7:0]} overflows from 3FF to 00, the contents of {EPCnH,XCCAPnH[1:0],CCAPnH[7:0]} are reloaded into {EPCnL,XCCAPnL[1:0], CCAPnL[7:0]}. This makes it possible to update the PWM without interference.

$$\text{PWM frequency in 10-bit mode} = \frac{\text{PCA input clock source frequency}}{1024}$$

When EPCnH=0 and CCAPnH=00H, PWM fixed output high
When EPCnH=1 and CCAPnH=FFH, PWM fixed output low

The structure of PCA module working in 10-bit PWM mode is shown below.

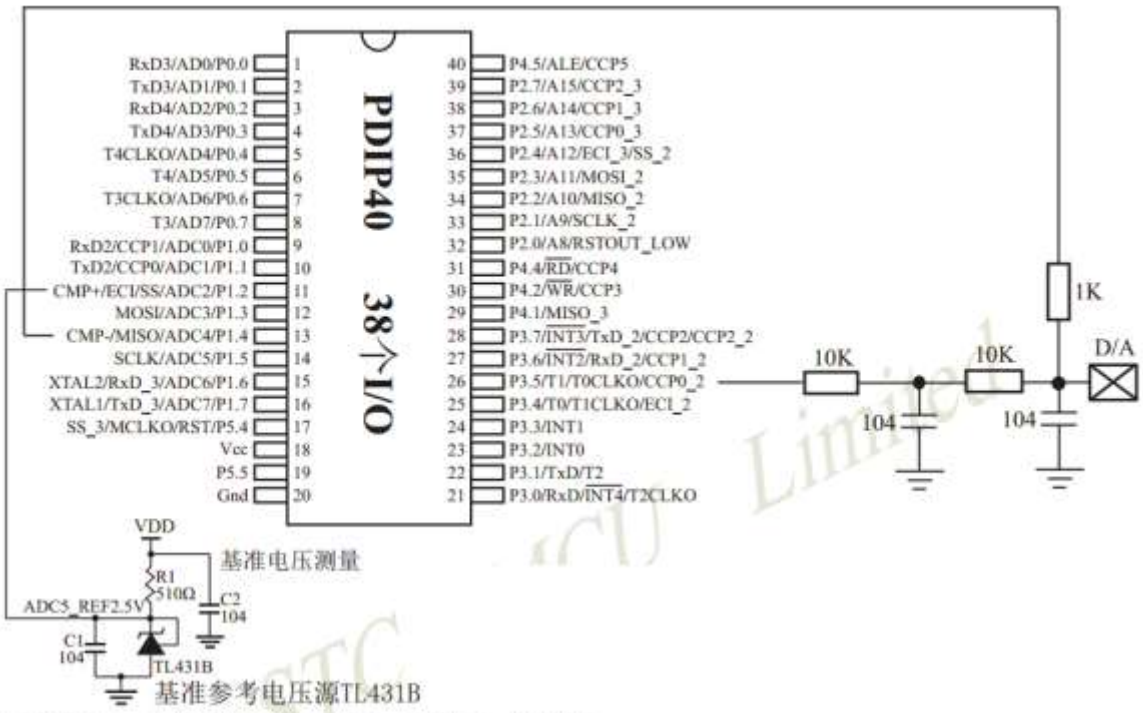


17.2.4.5 How to control PWM fixed output high / low

If PCA_PWMn &= 0xC0 and CCAPnH = 0x00, PWM fixed output high.
If PCA_PWMn |= 0x3F and CCAPnH = 0xFF, PWM fixed output low.

STC MCU

17.3 Reference Circuit for Implementing 8 ~ 16-bit DAC using CCP / PCA module



如应用简单，可无需基准参考电压源，直接与Vcc比较即可。

利用CCP/PCA模块的高速脉冲输出功能实现9~16位PWM来实现9~16位DAC，或用本身的硬件8位PWM来实现8位DAC，单片机本身也有10位ADC。

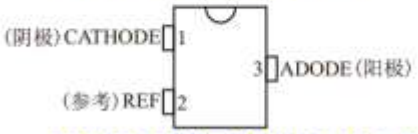
提示：

（1）PWM频率越高，输出波形越平滑。

（2）如果工作电压为5V，需输出1V电压，则设置高电平为1/5，低电平为4/5，则PWM输出电压就为1V。

（3）如果要输出高精准电压，建议用A/D检测输出的电压值，然后根据A/D检测的电压值逐步调整到所需要的电压。

基准参考电压源TL431B



SOT23-3封装，RMB￥0.15~0.3

基准参考电压源TL431B的符号



如应用简单，可无需基准参考电压源，直接与Vcc比较即可。

17.4 Example Routines

17.4.1 PCA Output PWM (6/7/8/10 bit)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     CCON    =   0xd8;
sbit    CF      =   CCON^7;
sbit    CR      =   CCON^6;
sbit    CCF2    =   CCON^2;
sbit    CCF1    =   CCON^1;
sbit    CCF0    =   CCON^0;
sfr     CMOD    =   0xd9;
sfr     CL      =   0xe9;
sfr     CH      =   0xf9;
sfr     CCAPM0  =   0xda;
sfr     CCAP0L  =   0xea;
sfr     CCAP0H  =   0xfa;
sfr     PCA_PWM0=   0xf2;
sfr     CCAPM1  =   0xdb;
sfr     CCAP1L  =   0xeb;
sfr     CCAP1H  =   0xfb;
sfr     PCA_PWM1=   0xf3;
sfr     CCAPM2  =   0xdc;
sfr     CCAP2L  =   0xec;
sfr     CCAP2H  =   0xfc;
sfr     PCA_PWM2=   0xf4;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
```

STC MCU

```
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x08;                    //PCA clock is the system clock
    CL = 0x00;
    CH = 0x00;
//——6 bit PWM——
    CCAPM0 = 0x42;                  //PCA 0 is in PWM mode
    PCA_PWM0 = 0x80;                //PCA 0 outputs 6-bit PWM
    CCAP0L = 0x20;                  //PWM duty cycle is 50%[(40H-20H)/40H]
    CCAP0H = 0x20;
//——7 bit PWM——
    CCAPM1 = 0x42;                  //PCA 1 is in PWM mode
    PCA_PWM1 = 0x40;                //PCA 1 outputs 7-bit PWM
    CCAP1L = 0x20;                  //PWM duty cycle is 75%[(80H-20H)/80H]
    CCAP1H = 0x20;
//——8 bit PWM——
//  CCAPM2 = 0x42;                  //PCA 2 is in PWM mode
//  PCA_PWM2 = 0x00;                //PCA 2 outputs 8-bit PWM
//  CCAP2L = 0x20;                  //PWM duty cycle is 87.5%[(100H-20H)/100H]
//  CCAP2H = 0x20;
//——10 bit PWM——
    CCAPM2 = 0x42;                  //PCA 2 is in PWM mode
    PCA_PWM2 = 0x00;                //PCA 2 outputs 10-bit PWM
    CCAP2L = 0x20;                  // PWM duty cycle is 96.875%[(400H-20H)/400H]
    CCAP2H = 0x20;
    CR = 1;                         //Start PCA timer

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CCON        DATA    0D8H
CF          BIT     CCON.7
CR          BIT     CCON.6
CCF2        BIT     CCON.2
CCF1        BIT     CCON.1
CCF0        BIT     CCON.0
CMOD        DATA    0D9H
CL          DATA    0E9H
CH          DATA    0F9H
CCAPM0      DATA    0DAH
CCAP0L      DATA    0EAH
CCAP0H      DATA    0FAH
PCA_PWM0    DATA    0F2H
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H
```

```
P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     CCON,#00H
            MOV     CMOD,#08H       ;PCA clock is the system clock
            MOV     CL,#00H
            MOV     CH,#0H
;――6 bit PWM――
            MOV     CCAPM0,#42H     ;PCA 0 is in PWM mode
            MOV     PCA_PWM0,#80H   ;PCA 0 outputs 6-bit PWM
            MOV     CCAP0L,#20H     ;PWM duty cycle is 50%[(40H-20H)/40H]
            MOV     CCAP0H,#20H
;――7 bit PWM――
            MOV     CCAPM1,#42H     ;PCA 1 is in PWM mode
            MOV     PCA_PWM1,#40H   ;PCA 1 outputs 7-bit PWM
            MOV     CCAP1L,#20H     ;PWM duty cycle is 75%[(80H-20H)/80H]
            MOV     CCAP1H,#20H
;――8 bit PWM――
;           MOV     CCAPM2,#42H     ;PCA 2 is in PWM mode
;           MOV     PCA_PWM2,#00H   ;PCA 2 outputs 8-bit PWM
;           MOV     CCAP2L,#20H     ;PWM duty cycle is 87.5%[(100H-20H)/100H]
;           MOV     CCAP2H,#20H
;――10 bit PWM――
            MOV     CCAPM2,#42H     ;PCA 2 is in PWM mode
            MOV     PCA_PWM2,#0C0H  ;PCA 2 outputs 10-bit PWM
            MOV     CCAP2L,#20H     ;PWM duty cycle is 96.875%[(400H-20H)/400H]
            MOV     CCAP2H,#20H
            SETB    CR              ;Start PCA timer
```

STC MCU

```
        JMP     $

        END
```

17.4.2 PCA Capture and Measure Pulse Width

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     CCON    =   0xd8;
sbit    CF      =   CCON^7;
sbit    CR      =   CCON^6;
sbit    CCF2    =   CCON^2;
sbit    CCF1    =   CCON^1;
sbit    CCF0    =   CCON^0;
sfr     CMOD    =   0xd9;
sfr     CL      =   0xe9;
sfr     CH      =   0xf9;
sfr     CCAPM0  =   0xda;
sfr     CCAP0L  =   0xea;
sfr     CCAP0H  =   0xfa;
sfr     PCA_PWM0=   0xf2;
sfr     CCAPM1  =   0xdb;
sfr     CCAP1L  =   0xeb;
sfr     CCAP1H  =   0xfb;
sfr     PCA_PWM1=   0xf3;
sfr     CCAPM2  =   0xdc;
sfr     CCAP2L  =   0xec;
sfr     CCAP2H  =   0xfc;
sfr     PCA_PWM2=   0xf4;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

unsigned char       cnt;                //Store PCA timing overflow times
unsigned long       count0;             //Record the last captured value
unsigned long       count1;             //Record the current captured value
unsigned long       length;             //Store the time length of signal

void PCA_Isr() interrupt 7
{
    if (CF)
```

```
    {
        CF = 0;
        cnt++;                                  //PCA timing overflow times+1
    }
    if (CCF0)
    {
        CCF0 = 0;
        count0 = count1;                        //Back up the last captured value
        ((unsigned char *)&count1)[3] = CCAP0L;
        ((unsigned char *)&count1)[2] = CCAP0H;
        ((unsigned char *)&count1)[1] = cnt;
        ((unsigned char *)&count1)[0] = 0;
        length = count1 - count0;               //length saved is the captured pulse width
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    cnt = 0;                                    //User variable initialization
    count0 = 0;
    count1 = 0;
    length = 0;
    CCON = 0x00;
    CMOD = 0x09;                                //PCA clock is the system clock, enable PCA timing interrupt
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x11;                              //PCA module 0 is 16-bit capture mode (falling edge capture)
    CCAPM0 = 0x21;                              //PCA module 0 is 16-bit capture mode (falling edge capture)
    CCAPM0 = 0x31;                              //PCA module 0 is 16-bit capture mode (falling edge capture)
    CCAP0L = 0x00;
    CCAP0H = 0x00;
    CR = 1;                                     //Start PCA timer
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CCON        DATA        0D8H
```

```
CF          BIT     CCON.7
CR          BIT     CCON.6
CCF2        BIT     CCON.2
CCF1        BIT     CCON.1
CCF0        BIT     CCON.0
CMOD        DATA    0D9H
CL          DATA    0E9H
CH          DATA    0F9H
CCAPM0      DATA    0DAH
CCAP0L      DATA    0EAH
CCAP0H      DATA    0FAH
PCA_PWM0    DATA    0F2H
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H

CNT         DATA    20H
COUNT0      DATA    21H         ;3 bytes
COUNT1      DATA    24H         ;3 bytes
LENGTH      DATA    27H         ;3 bytes, (COUNT1-COUNT0)

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

            ORG     0100H
PCAISR:
            PUSH    ACC
            PUSH    PSW
            JNB     CF,CHECKCCF0
            CLR     CF          ;Clear interrupt flag
            INC     CNT         ; PCA Timing overflow times+1
CHECKCCF0:
            JNB     CCF0,ISREXIT
            CLR     CCF0
            MOV     COUNT0,COUNT1   ;Back up the last captured value
            MOV     COUNT0+1,COUNT1+1
            MOV     COUNT0+2,COUNT1+2
            MOV     COUNT1,CNT      ;Save the current captured value
            MOV     COUNT1+1,CCAP0H
```

STC MCU

```
                MOV     COUNT1+2,CCAP0L
                CLR     C                       ;Calculate two captures' differences
                MOV     A,COUNT1+2
                SUBB    A,COUNT0+2
                MOV     LENGTH+2,A
                MOV     A,COUNT1+1
                SUBB    A,COUNT0+1
                MOV     LENGTH+1,A
                MOV     A,COUNT1
                SUBB    A,COUNT0
                MOV     LENGTH,A                ;LENGTH saved is the captured pulse width
ISREXIT:
                POP     PSW
                POP     ACC
                RETI

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                CLR     A
                MOV     CNT,A                   ;User variable initialization
                MOV     COUNT0,A
                MOV     COUNT0+1,A
                MOV     COUNT0+2,A
                MOV     COUNT1,A
                MOV     COUNT1+1,A
                MOV     COUNT1+2,A
                MOV     LENGTH,A
                MOV     LENGTH+1,A
                MOV     LENGTH+2,A

                MOV     CCON,#00H
                MOV     CMOD,#09H               ;PCA clock is the system clock, enable PCA timing
interrupt
                MOV     CL,#00H
                MOV     CH,#0H
                MOV     CCAPM0,#11H             ;PCA module 0 is 16-bit capture mode (falling edge capture)
;               MOV     CCAPM0,#21H             ;PCA module 0 is 16-bit capture mode (rising edge capture)
;               MOV     CCAPM0,#31H     ;PCA module 0 is 16-bit capture mode (falling and rising edge capture)
                MOV     CCAP0L,#00H
                MOV     CCAP0H,#00H
                SETB    CR                      ;Start PCA timer
                SETB    EA

                JMP     $

                END
```

17.4.3 PCA Implements 16-bit Software Timing

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   T50HZ           (11059200L / 12 / 2 / 50)

sfr       CCON      =     0xd8;
sbit      CF        =     CCON^7;
sbit      CR        =     CCON^6;
sbit      CCF2      =     CCON^2;
sbit      CCF1      =     CCON^1;
sbit      CCF0      =     CCON^0;
sfr       CMOD      =     0xd9;
sfr       CL        =     0xe9;
sfr       CH        =     0xf9;
sfr       CCAPM0    =     0xda;
sfr       CCAP0L    =     0xea;
sfr       CCAP0H    =     0xfa;
sfr       PCA_PWM0  =     0xf2;
sfr       CCAPM1    =     0xdb;
sfr       CCAP1L    =     0xeb;
sfr       CCAP1H    =     0xfb;
sfr       PCA_PWM1  =     0xf3;
sfr       CCAPM2    =     0xdc;
sfr       CCAP2L    =     0xec;
sfr       CCAP2H    =     0xfc;
sfr       PCA_PWM2  =     0xf4;

sfr       P0M1      =     0x93;
sfr       P0M0      =     0x94;
sfr       P1M1      =     0x91;
sfr       P1M0      =     0x92;
sfr       P2M1      =     0x95;
sfr       P2M0      =     0x96;
sfr       P3M1      =     0xb1;
sfr       P3M0      =     0xb2;
sfr       P4M1      =     0xb3;
sfr       P4M0      =     0xb4;
sfr       P5M1      =     0xc9;
sfr       P5M0      =     0xca;

sbit      P10       =     P1^0;

unsigned int              value;

void PCA_Isr() interrupt 7
{
    CCF0 = 0;
    CCAP0L = value;
    CCAP0H = value >> 8;
```

```
    value += T50HZ;

    P10 = !P10;                                 //Test port
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x00;                                //PCA clock is the system clock/12
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x49;                              //PCA module 0 is 16-bit timer mode
    value = T50HZ;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T50HZ;
    CR = 1;                                     //Start PCA timer
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CCON        DATA    0D8H
CF          BIT     CCON.7
CR          BIT     CCON.6
CCF2        BIT     CCON.2
CCF1        BIT     CCON.1
CCF0        BIT     CCON.0
CMOD        DATA    0D9H
CL          DATA    0E9H
CH          DATA    0F9H
CCAPM0      DATA    0DAH
CCAP0L      DATA    0EAH
CCAP0H      DATA    0FAH
PCA_PWM0    DATA    0F2H
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
```

```
PCA_PWM2    DATA    0F4H

T50HZ       EQU     2400H           ;11059200/12/2/50

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

            ORG     0100H
PCAISR:
            PUSH    ACC
            PUSH    PSW
            CLR     CCF0
            MOV     A,CCAP0L
            ADD     A,#LOW T50HZ
            MOV     CCAP0L,A
            MOV     A,CCAP0H
            ADDC    A,#HIGH T50HZ
            MOV     CCAP0H,A
            CPL     P1.0            ;Test port, flashing frequency is 50Hz
            POP     PSW
            POP     ACC
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     CCON,#00H
            MOV     CMOD,#00H       ;PCA clock is the system clock/12
            MOV     CL,#00H
            MOV     CH,#0H
            MOV     CCAPM0,#49H     ;PCA module 0 is 16-bit timer mode
            MOV     CCAP0L,#LOW T50HZ
```

STC MCU

```
        MOV     CCAP0H,#HIGH T50HZ
        SETB    CR                      ;Start PCA timer
        SETB    EA

        JMP     $

        END
```

17.4.4 PCA realizes 16-bit software timing (ECI external clock mode)

Note: The external clock frequency cannot be higher than 1/2 of the system frequency.

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define T50HZ (11059200L / 12 / 2 / 50)

sfr CCON = 0xd8;
sbit CF = CCON^7;
sbit CR = CCON^6;
sbit CCF2 = CCON^2;
sbit CCF1 = CCON^1;
sbit CCF0 = CCON^0;
sfr CMOD = 0xd9;
sfr CL = 0xe9;
sfr CH = 0xf9;
sfr CCAPM0 = 0xda;
sfr CCAP0L = 0xea;
sfr CCAP0H = 0xfa;
sfr PCA_PWM0 = 0xf2;
sfr CCAPM1 = 0xdb;
sfr CCAP1L = 0xeb;
sfr CCAP1H = 0xfb;
sfr PCA_PWM1 = 0xf3;
sfr CCAPM2 = 0xdc;
sfr CCAP2L = 0xec;
sfr CCAP2H = 0xfc;
sfr PCA_PWM2 = 0xf4;

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

sbit P10 = P1^0;

unsigned int value;
```

STC MCU

```
void PCA_Isr() interrupt 7
{
    CCF0 = 0;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T50HZ;

    P10 = !P10;                                 //Test port
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x06;                                // The PCA clock is an external clock input from the ECI
port
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x49;                              // PCA module 0 is 16-bit timer mode
    value = T50HZ;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T50HZ;
    CR = 1;                                     // Start the PCA timer
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CCON        DATA        0D8H
CF          BIT         CCON.7
CR          BIT         CCON.6
CCF2        BIT         CCON.2
CCF1        BIT         CCON.1
CCF0        BIT         CCON.0
CMOD        DATA        0D9H
CL          DATA        0E9H
CH          DATA        0F9H
CCAPM0      DATA        0DAH
CCAP0L      DATA        0EAH
CCAP0H      DATA        0FAH
PCA_PWM0    DATA        0F2H
```

```
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H

T50HZ       EQU     2400H               ;11059200/12/2/50

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

            ORG     0100H
PCAISR:
            PUSH    ACC
            PUSH    PSW
            CLR     CCF0
            MOV     A,CCAP0L
            ADD     A,#LOW T50HZ
            MOV     CCAP0L,A
            MOV     A,CCAP0H
            ADDC    A,#HIGH T50HZ
            MOV     CCAP0H,A
            CPL     P1.0                ; Test port, flashing frequency is 50Hz
            POP     PSW
            POP     ACC
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H
```

STC MCU

```
        MOV     CCON,#00H
        MOV     CMOD,#06H       ; The PCA clock is an external clock input from the ECI
port
        MOV     CL,#00H
        MOV     CH,#0H
        MOV     CCAPM0,#49H     ; PCA module 0 is 16-bit timer mode
        MOV     CCAP0L,#LOW T50HZ
        MOV     CCAP0H,#HIGH T50HZ
        SETB    CR              ; Start the PCA timer
        SETB    EA

        JMP     $

        END
```

17.4.5 PCA Output High-speed Pulse

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   T38K4HZ       (11059200L / 2 / 38400)

sfr     CCON    =   0xd8;
sbit    CF      =   CCON^7;
sbit    CR      =   CCON^6;
sbit    CCF2    =   CCON^2;
sbit    CCF1    =   CCON^1;
sbit    CCF0    =   CCON^0;
sfr     CMOD    =   0xd9;
sfr     CL      =   0xe9;
sfr     CH      =   0xf9;
sfr     CCAPM0  =   0xda;
sfr     CCAP0L  =   0xea;
sfr     CCAP0H  =   0xfa;
sfr     PCA_PWM0=   0xf2;
sfr     CCAPM1  =   0xdb;
sfr     CCAP1L  =   0xeb;
sfr     CCAP1H  =   0xfb;
sfr     PCA_PWM1=   0xf3;
sfr     CCAPM2  =   0xdc;
sfr     CCAP2L  =   0xec;
sfr     CCAP2H  =   0xfc;
sfr     PCA_PWM2=   0xf4;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
```

```
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

unsigned int        value;

void PCA_Isr() interrupt 7
{
    CCF0 = 0;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T38K4HZ;
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x08;                    //PCA clock is the system clock
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x4d;                  //PCA module 0 is 16-bit timer mode, and enable pulse output
    value = T38K4HZ;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T38K4HZ;
    CR = 1;                         //Start PCA timer
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CCON    DATA    0D8H
CF      BIT     CCON.7
CR      BIT     CCON.6
CCF2    BIT     CCON.2
CCF1    BIT     CCON.1
CCF0    BIT     CCON.0
CMOD    DATA    0D9H
CL      DATA    0E9H
CH      DATA    0F9H
```

```
CCAPM0      DATA    0DAH
CCAP0L      DATA    0EAH
CCAP0H      DATA    0FAH
PCA_PWM0    DATA    0F2H
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H

T38K4HZ     EQU     90H             ;11059200/2/38400

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

            ORG     0100H
PCAISR:
            PUSH    ACC
            PUSH    PSW
            CLR     CCF0
            MOV     A,CCAP0L
            ADD     A,#LOW T38K4HZ
            MOV     CCAP0L,A
            MOV     A,CCAP0H
            ADDC    A,#HIGH T38K4HZ
            MOV     CCAP0H,A
            POP     PSW
            POP     ACC
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
```

STC MCU

```
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     CCON,#00H
        MOV     CMOD,#08H               ;PCA clock is the system clock
        MOV     CL,#00H
        MOV     CH,#0H
        MOV     CCAPM0,#4DH             ;PCA module 0 is 16-bit timer mode, and enable pulse
output
        MOV     CCAP0L,#LOW T38K4HZ
        MOV     CCAP0H,#HIGH T38K4HZ
        SETB    CR                      ;Start PCA timer
        SETB    EA

        JMP     $

        END
```

17.4.6 PCA Extends External Interrupt

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     CCON    =   0xd8;
sbit    CF      =   CCON^7;
sbit    CR      =   CCON^6;
sbit    CCF2    =   CCON^2;
sbit    CCF1    =   CCON^1;
sbit    CCF0    =   CCON^0;
sfr     CMOD    =   0xd9;
sfr     CL      =   0xe9;
sfr     CH      =   0xf9;
sfr     CCAPM0  =   0xda;
sfr     CCAP0L  =   0xea;
sfr     CCAP0H  =   0xfa;
sfr     PCA_PWM0=   0xf2;
sfr     CCAPM1  =   0xdb;
sfr     CCAP1L  =   0xeb;
sfr     CCAP1H  =   0xfb;
sfr     PCA_PWM1=   0xf3;
sfr     CCAPM2  =   0xdc;
sfr     CCAP2L  =   0xec;
sfr     CCAP2H  =   0xfc;
sfr     PCA_PWM2=   0xf4;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
```

```
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void PCA_Isr() interrupt 7
{
    CCF0 = 0;
    P10 = !P10;
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x08;                    //PCA clock is the system clock
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x11;                  //Extend external port CCP0 as a falling edge interrupt port
//  CCAPM0 = 0x21;                  //Extend external port CCP0 as a rising edge interrupt port
//  CCAPM0 = 0x31;          //Extend external port CCP0 as a rising and falling edge interrupt port
    CCAP0L = 0;
    CCAP0H = 0;
    CR = 1;                         //Start PCA timer
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CCON    DATA    0D8H
CF      BIT     CCON.7
CR      BIT     CCON.6
CCF2    BIT     CCON.2
CCF1    BIT     CCON.1
CCF0    BIT     CCON.0
CMOD    DATA    0D9H
CL      DATA    0E9H
CH      DATA    0F9H
```

```
CCAPM0      DATA    0DAH
CCAP0L      DATA    0EAH
CCAP0H      DATA    0FAH
PCA_PWM0    DATA    0F2H
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

            ORG     0100H
PCAISR:
            CLR     CCF0
            CPL     P1.0
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     CCON,#00H
            MOV     CMOD,#08H       ;PCA clock is the system clock
            MOV     CL,#00H
            MOV     CH,#0H
            MOV     CCAPM0,#11H     ;Extend external port CCP0 as a falling edge interrupt port
;           MOV     CCAPM0,#21H     ;Extend external port CCP0 as a rising edge interrupt port
;           MOV     CCAPM0,#31H     ;Extend external port CCP0 as a rising and falling edge interrupt port
```

STC MCU

```
MOV     CCAP0L,#0
MOV     CCAP0H,#0
SETB    CR                  ;Start PCA timer
SETB    EA

JMP     $

END
```

STC MCU

18 Enhanced PWM with 15-bit Accuracy

Product line	Enhanced PWM
STC8G1K08 family	
STC8G1K08-8Pin family	
STC8G1K08A family	
STC8G2K64S4 family	●
STC8G2K64S2 family	●
STC8G1K08T family	
STC15H2K64S4 family	●

(When an interrupt with an interrupt number greater than 31 is used in a C program, an error will be reported when compiled in Keil. For the solution, please refer to Appendix.)

6 groups of individually 8-channel enhanced PWM waveform generators are integrated in some STC8G series microcontrollers. Each group can set the period separately, and has 8 independent channels. Because the P5 port lacks the 3 pins P5.5/P5.6/P5.7, the STC8G2K64S4 series can output up to 45 channels of PWM.

Group 0 has 8 channels, PWM00~PWM07, which use PWM0CH/PWM0CL to set the period value;

Group 1 has 8 channels, PWM10~PWM17, which use PWM1CH/PWM1CL to set the period value;

Group 2 has 8 channels, PWM20~PWM27, which use PWM2CH/PWM2CL to set the period value;

Group 3 has 8 channels, PWM30~PWM37, which use PWM3CH/PWM3CL to set the period value;

Group 4 has 8 channels, PWM40~PWM47, which use PWM4CH/PWM4CL to set the period value;

Group 5 has 5 channels, PWM50~PWM54, which use PWM5CH/PWM5CL to set the period value.

The STC8G2K64S2 series microcontrollers integrate a group (P2 port) enhanced PWM waveform generator. The P2 group can generate 8 independent PWMs, so the STC8G2K64S2 series can output up to 8 PWMs. PWM20~PWM27, using PWM2CH/PWM2CL to set the period value.

The clock source of PWM can be selected. There is a 15-bit PWM counter in the PWM waveform generator which is used for 8 channel PWMs. The initial level of each PWM can be set. In addition, two counters T1 and T2 are designed in the PWM waveform generator to control the waveform hopping for each PWM. The width of high and low level of each PWM can be set very flexibly, so that the PWM duty ratio and PWM output delay can be controlled. Because the 8 PWMs are independent and the initial state of each PWM can be set, any two of them can be used together to achieve special applications such as complementary symmetrical output and dead band control. Note: Enhanced PWM only has output function. If you need to measure pulse width, please use the PCA/CCP/PWM function of this series microcontroller. The enhanced PWM waveform generators also feature the ability to monitor external abnormal events, such as external port P3.5/0.6/0.7 abnormal level and the abnormal comparator results. It can be used to shutdown PWM outputs immediately. The PWM waveform generator can also be associated with ADC. Any point in the PWM cycle can be set to trigger the ADC conversion event.

Comparison of STC three kinds of hardware PWM:

Compatible with traditional 8051 PCA/CCP/PWM: It can output PWM waveforms, capture external input signals and output high-speed pulses. It can output 6-bit/7-bit/8-bit/10-bit PWM waveform externally. The frequency of the 6-bit PWM waveform is the PCA module clock source frequency/64. The frequency of the 7-bit PWM waveform is the PCA module clock source frequency/128. The frequency of the 8-bit PWM waveform is the PCA module clock source frequency/256. The frequency of the 10-bit PWM waveform is the PCA module clock source frequency/1024. For the external input signal capture, you can capture the rising edge, the falling edge or the rising edge and the falling edge at the same time.

15-bit enhanced PWM of STC8G series: It can only output PWM waveform externally, without input capture function. The frequency and duty cycle of PWM output can be set arbitrarily. Through software intervention, multiple complementary/symmetrical with dead-time PWM waveforms can be realized. There are external abnormality detection function and real-time trigger ADC conversion function.

The 16-bit advanced PWM timer of the STC8H series: It is the PWM with the strongest function in STC products at present, which can output PWM waveforms of any frequency and any duty. It can output complementary/symmetrical with dead-time PWM waveform without software intervention. It can capture the external signal for the rising edge, the falling edge or the rising edge and the falling edge at the same time. When measuring the external waveform, the period value and the duty ratio value of the waveform can be measured at the same time. There are quadrature encoding function, external anomaly detection function and real-time trigger ADC conversion function.

18.1 Registers Related to PWM

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
PWMSET	PWM Set Register	F1H	ENGLBSET	PWMRST	ENPWM5	ENPWM4	ENPWM3	ENPWM2	ENPWM1	ENPWM0	0000,0000
PWMCFG01	PWM Configuration Register	F6H	PWM1CBIF	EPWM1CBI	FLTPS0	PWM1CEN	PWM0CBIF	EPWM0CBI	ENPWM0TA	PWM0CEN	0000,0000
PWMCFG23	PWM Configuration Register	F7H	PWM3CBIF	EPWM3CBI	FLTPS1	PWM3CEN	PWM2CBIF	EPWM2CBI	ENPWM2TA	PWM2CEN	0000,0000
PWMCFG45	PWM Configuration Register	FEH	PWM5CBIF	EPWM5CBI	FLTPS2	PWM5CEN	PWM4CBIF	EPWM4CBI	ENPWM4TA	PWM4CEN	0000,0000

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
PWM0CH	PWM0 Counter High Byte	FF00H	-								x000,0000
PWM0CL	PWM0 Counter Low Byte	FF01H									0000,0000
PWM0CKS	PWM0 Clock Select Register	FF02H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000
PWM0TADCH	PWM0 Trigger ADC Counter High Byte	FF03H	-								x000,0000
PWM0TADCL	PWM0 Trigger ADC Counter Low Byte	FF04H									0000,0000
PWM0IF	PWM0 Interrupt Flag Register	FF05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM0FDCR	PWM0 Fault Detection Control Register	FF06H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000
PWM00T1H	PWM00 T1 High Byte	FF10H	-								x000,0000
PWM00T1L	PWM00 T1 Low Byte	FF11H									0000,0000
PWM00T2H	PWM00 T2 High Byte	FF12H	-								x000,0000
PWM00T2L	PWM00 T2 Low Byte	FF13H									0000,0000
PWM00CR	PWM00 Control Register	FF14H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM00HLD	PWM00 Level Holding Control Register	FF15H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM01T1H	PWM01 T1 High Byte	FF18H	-								x000,0000
PWM01T1L	PWM01 T1 Low Byte	FF19H									0000,0000
PWM01T2H	PWM01 T2 High Byte	FF1AH	-								x000,0000
PWM01T2L	PWM01 T2 Low Byte	FF1BH									0000,0000
PWM01CR	PWM01 Control Register	FF1CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM01HLD	PWM01 Level Holding Control Register	FF1DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM02T1H	PWM02 T1 High Byte	FF20H	-								x000,0000
PWM02T1L	PWM02 T1 Low Byte	FF21H									0000,0000
PWM02T2H	PWM02 T2 High Byte	FF22H	-								x000,0000
PWM02T2L	PWM02 T2 Low Byte	FF23H									0000,0000
PWM02CR	PWM02 Control Register	FF24H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM02HLD	PWM02 Level Holding Control Register	FF25H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM03T1H	PWM03 T1 High Byte	FF28H	-								x000,0000
PWM03T1L	PWM03 T1 Low Byte	FF29H									0000,0000
PWM03T2H	PWM03 T2 High Byte	FF2AH	-								x000,0000
PWM03T2L	PWM03 T2 Low Byte	FF2BH									0000,0000
PWM03CR	PWM03 Control Register	FF2CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM03HLD	PWM03 Level Holding Control Register	FF2DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM04T1H	PWM04 T1 High Byte	FF30H	-								x000,0000
PWM04T1L	PWM04 T1 Low Byte	FF31H									0000,0000
PWM04T2H	PWM04 T2 High Byte	FF32H	-								x000,0000

PWM04T2L	PWM04 T2 Low Byte	FF33H									0000,0000
PWM04CR	PWM04 Control Register	FF34H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM04HLD	PWM04 Level Holding Control Register	FF35H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM05T1H	PWM05 T1 High Byte	FF38H	-								x000,0000
PWM05T1L	PWM05 T1 Low Byte	FF39H									0000,0000
PWM05T2H	PWM05 T2 High Byte	FF3AH	-								x000,0000
PWM05T2L	PWM05 T2 Low Byte	FF3BH									0000,0000
PWM05CR	PWM05 Control Register	FF3CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM05HLD	PWM05 Level Holding Control Register	FF3DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM06T1H	PWM06 T1 High Byte	FF40H	-								x000,0000
PWM06T1L	PWM06 T1 Low Byte	FF41H									0000,0000
PWM06T2H	PWM06 T2 High Byte	FF42H	-								x000,0000
PWM06T2L	PWM06 T2 Low Byte	FF43H									0000,0000
PWM06CR	PWM06 Control Register	FF44H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM06HLD	PWM06 Level Holding Control Register	FF45H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM07T1H	PWM07 T1 High Byte	FF48H	-								x000,0000
PWM07T1L	PWM07 T1 Low Byte	FF49H									0000,0000
PWM07T2H	PWM07 T2 High Byte	FF4AH	-								x000,0000
PWM07T2L	PWM07 T2 Low Byte	FF4BH									0000,0000
PWM07CR	PWM07 Control Register	FF4CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM07HLD	PWM07 Level Holding Control Register	FF4DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM1CH	PWM1 Counter High Byte	FF50H	-								x000,0000
PWM1CL	PWM1 Counter Low Byte	FF51H									0000,0000
PWM1CKS	PWM1 Clock Select Register	FF52H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000
PWM1IF	PWM1 Interrupt Flag Register	FF55H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM1FDCR	PWM1 Fault Detection Control Register	FF56H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF	0000,0000
PWM10T1H	PWM10 T1 High Byte	FF60H	-								x000,0000
PWM10T1L	PWM10 T1 Low Byte	FF61H									0000,0000
PWM10T2H	PWM10 T2 High Byte	FF62H	-								x000,0000
PWM10T2L	PWM10 T2 Low Byte	FF63H									0000,0000
PWM10CR	PWM10 Control Register	FF64H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM10HLD	PWM10 Level Holding Control Register	FF65H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM11T1H	PWM11 T1 High Byte	FF68H	-								x000,0000
PWM11T1L	PWM11 T1 Low Byte	FF69H									0000,0000
PWM11T2H	PWM11 T2 High Byte	FF6AH	-								x000,0000
PWM11T2L	PWM11 T2 Low Byte	FF6BH									0000,0000
PWM11CR	PWM11 Control Register	FF6CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM11HLD	PWM11 Level Holding Control Register	FF6DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM12T1H	PWM12 T1 High Byte	FF70H	-								x000,0000
PWM12T1L	PWM12 T1 Low Byte	FF71H									0000,0000
PWM12T2H	PWM12 T2 High Byte	FF72H	-								x000,0000
PWM12T2L	PWM12 T2 Low Byte	FF73H									0000,0000
PWM12CR	PWM12 Control Register	FF74H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM12HLD	PWM12 Level Holding Control Register	FF75H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM13T1H	PWM13 T1 High Byte	FF78H	-								x000,0000
PWM13T1L	PWM13 T1 Low Byte	FF79H									0000,0000
PWM13T2H	PWM13 T2 High Byte	FF7AH	-								x000,0000
PWM13T2L	PWM13 T2 Low Byte	FF7BH									0000,0000
PWM13CR	PWM13 Control Register	FF7CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM13HLD	PWM13 Level Holding Control Register	FF7DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM14T1H	PWM14 T1 High Byte	FF80H	-								x000,0000
PWM14T1L	PWM14 T1 Low Byte	FF81H									0000,0000
PWM14T2H	PWM14 T2 High Byte	FF82H	-								x000,0000
PWM14T2L	PWM14 T2 Low Byte	FF83H									0000,0000
PWM14CR	PWM14 Control Register	FF84H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM14HLD	PWM14 Level Holding Control Register	FF85H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM15T1H	PWM15 T1 High Byte	FF88H	-								x000,0000
PWM15T1L	PWM15 T1 Low Byte	FF89H									0000,0000
PWM15T2H	PWM15 T2 High Byte	FF8AH	-								x000,0000
PWM15T2L	PWM15 T2 Low Byte	FF8BH									0000,0000
PWM15CR	PWM15 Control Register	FF8CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM15HLD	PWM15 Level Holding Control Register	FF8DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM16T1H	PWM16 T1 High Byte	FF90H	-								x000,0000
PWM16T1L	PWM16 T1 Low Byte	FF91H									0000,0000
PWM16T2H	PWM16 T2 High Byte	FF92H	-								x000,0000
PWM16T2L	PWM16 T2 Low Byte	FF93H									0000,0000
PWM16CR	PWM16 Control Register	FF94H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM16HLD	PWM16 Level Holding Control Register	FF95H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM17T1H	PWM17 T1 High Byte	FF98H	-								x000,0000
PWM17T1L	PWM17 T1 Low Byte	FF99H									0000,0000
PWM17T2H	PWM17 T2 High Byte	FF9AH	-								x000,0000
PWM17T2L	PWM17 T2 Low Byte	FF9BH									0000,0000
PWM17CR	PWM17 Control Register	FF9CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM17HLD	PWM17 Level Holding Control Register	FF9DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM2CH	PWM2 Counter High Byte	FFA0H	-								x000,0000
PWM2CL	PWM2 Counter Low Byte	FFA1H									0000,0000
PWM2CKS	PWM2 Clock Select Register	FFA2H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000
PWM2TADCH	PWM2 Trigger ADC Counter High Byte	FFA3H	-								x000,0000
PWM2TADCL	PWM2 Trigger ADC Counter Low Byte	FFA4H									0000,0000
PWM2IF	PWM2 Interrupt Flag Register	FFA5H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM2FDCR	PWM2 Fault Detection Control Register	FFA6H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000
PWM20T1H	PWM20 T1 High Byte	FFB0H	-								x000,0000
PWM20T1L	PWM20 T1 Low Byte	FFB1H									0000,0000

PWM20T2H	PWM20 T2 High Byte	FFB2H	-								x000,0000
PWM20T2L	PWM20 T2 Low Byte	FFB3H									0000,0000
PWM20CR	PWM20 Control Register	FFB4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM20HLD	PWM20 Level Holding Control Register	FFB5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM21T1H	PWM21 T1 High Byte	FFB8H	-								x000,0000
PWM21T1L	PWM21 T1 Low Byte	FFB9H									0000,0000
PWM21T2H	PWM21 T2 High Byte	FFBAH	-								x000,0000
PWM21T2L	PWM21 T2 Low Byte	FFBBH									0000,0000
PWM21CR	PWM21 Control Register	FFBCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM21HLD	PWM21 Level Holding Control Register	FFBDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM22T1H	PWM22 T1 High Byte	FFC0H	-								x000,0000
PWM22T1L	PWM22 T1 Low Byte	FFC1H									0000,0000
PWM22T2H	PWM22 T2 High Byte	FFC2H	-								x000,0000
PWM22T2L	PWM22 T2 Low Byte	FFC3H									0000,0000
PWM22CR	PWM22 Control Register	FFC4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM22HLD	PWM22 Level Holding Control Register	FFC5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM23T1H	PWM23 T1 High Byte	FFC8H	-								x000,0000
PWM23T1L	PWM23 T1 Low Byte	FFC9H									0000,0000
PWM23T2H	PWM23 T2 High Byte	FFCAH	-								x000,0000
PWM23T2L	PWM23 T2 Low Byte	FFCBH									0000,0000
PWM23CR	PWM23 Control Register	FFCCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM23HLD	PWM23 Level Holding Control Register	FFCDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM24T1H	PWM24 T1 High Byte	FFD0H	-								x000,0000
PWM24T1L	PWM24 T1 Low Byte	FFD1H									0000,0000
PWM24T2H	PWM24 T2 High Byte	FFD2H	-								x000,0000
PWM24T2L	PWM24 T2 Low Byte	FFD3H									0000,0000
PWM24CR	PWM24 Control Register	FFD4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM24HLD	PWM24 Level Holding Control Register	FFD5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM25T1H	PWM25 T1 High Byte	FFD8H	-								x000,0000
PWM25T1L	PWM25 T1 Low Byte	FFD9H									0000,0000
PWM25T2H	PWM25 T2 High Byte	FFDAH	-								x000,0000
PWM25T2L	PWM25 T2 Low Byte	FFDBH									0000,0000
PWM25CR	PWM25 Control Register	FFDCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM25HLD	PWM25 Level Holding Control Register	FFDDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM26T1H	PWM26 T1 High Byte	FFE0H	-								x000,0000
PWM26T1L	PWM26 T1 Low Byte	FFE1H									0000,0000
PWM26T2H	PWM26 T2 High Byte	FFE2H	-								x000,0000
PWM26T2L	PWM26 T2 Low Byte	FFE3H									0000,0000
PWM26CR	PWM26 Control Register	FFE4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM26HLD	PWM26 Level Holding Control Register	FFE5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM27T1H	PWM27 T1 High Byte	FFE8H	-								x000,0000
PWM27T1L	PWM27 T1 Low Byte	FFE9H									0000,0000
PWM27T2H	PWM27 T2 High Byte	FFEAH	-								x000,0000
PWM27T2L	PWM27 T2 Low Byte	FFEBH									0000,0000
PWM27CR	PWM27 Control Register	FFECH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM27HLD	PWM27 Level Holding Control Register	FFEDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM3CH	PWM3 Counter High Byte	FC00H	-								x000,0000
PWM3CL	PWM3 Counter Low Byte	FC01H									0000,0000
PWM3CKS	PWM3 Clock Select Register	FC02H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000
PWM3IF	PWM3 Interrupt Flag Register	FC05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM3FDCR	PWM3 Fault Detection Control Register	FC06H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF	0000,0000
PWM30T1H	PWM30 T1 High Byte	FC10H	-								x000,0000
PWM30T1L	PWM30 T1 Low Byte	FC11H									0000,0000
PWM30T2H	PWM30 T2 High Byte	FC12H	-								x000,0000
PWM30T2L	PWM30 T2 Low Byte	FC13H									0000,0000
PWM30CR	PWM30 Control Register	FC14H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM30HLD	PWM30 Level Holding Control Register	FC15H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM31T1H	PWM31 T1 High Byte	FC18H	-								x000,0000
PWM31T1L	PWM31 T1 Low Byte	FC19H									0000,0000
PWM31T2H	PWM31 T2 High Byte	FC1AH	-								x000,0000
PWM31T2L	PWM31 T2 Low Byte	FC1BH									0000,0000
PWM31CR	PWM31 Control Register	FC1CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM31HLD	PWM31 Level Holding Control Register	FC1DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM32T1H	PWM32 T1 High Byte	FC20H	-								x000,0000
PWM32T1L	PWM32 T1 Low Byte	FC21H									0000,0000
PWM32T2H	PWM32 T2 High Byte	FC22H	-								x000,0000
PWM32T2L	PWM32 T2 Low Byte	FC23H									0000,0000
PWM32CR	PWM32 Control Register	FC24H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM32HLD	PWM32 Level Holding Control Register	FC25H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM33T1H	PWM33 T1 High Byte	FC28H	-								x000,0000
PWM33T1L	PWM33 T1 Low Byte	FC29H									0000,0000
PWM33T2H	PWM33 T2 High Byte	FC2AH	-								x000,0000
PWM33T2L	PWM33 T2 Low Byte	FC2BH									0000,0000
PWM33CR	PWM33 Control Register	FC2CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM33HLD	PWM33 Level Holding Control Register	FC2DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM34T1H	PWM34 T1 High Byte	FC30H	-								x000,0000
PWM34T1L	PWM34 T1 Low Byte	FC31H									0000,0000
PWM34T2H	PWM34 T2 High Byte	FC32H	-								x000,0000
PWM34T2L	PWM34 T2 Low Byte	FC33H									0000,0000
PWM34CR	PWM34 Control Register	FC34H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM34HLD	PWM34 Level Holding Control Register	FC35H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM35T1H	PWM35 T1 High Byte	FC38H	-								x000,0000
PWM35T1L	PWM35 T1 Low Byte	FC39H									0000,0000

PWM35T2H	PWM35 T2 High Byte	FC3AH	-								x000,0000
PWM35T2L	PWM35 T2 Low Byte	FC3BH									0000,0000
PWM35CR	PWM35 Control Register	FC3CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM35HLD	PWM35 Level Holding Control Register	FC3DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM36T1H	PWM36 T1 High Byte	FC40H	-								x000,0000
PWM36T1L	PWM36 T1 Low Byte	FC41H									0000,0000
PWM36T2H	PWM36 T2 High Byte	FC42H	-								x000,0000
PWM36T2L	PWM36 T2 Low Byte	FC43H									0000,0000
PWM36CR	PWM36 Control Register	FC44H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM36HLD	PWM36 Level Holding Control Register	FC45H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM37T1H	PWM37 T1 High Byte	FC48H	-								x000,0000
PWM37T1L	PWM37 T1 Low Byte	FC49H									0000,0000
PWM37T2H	PWM37 T2 High Byte	FC4AH	-								x000,0000
PWM37T2L	PWM37 T2 Low Byte	FC4BH									0000,0000
PWM37CR	PWM37 Control Register	FC4CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM37HLD	PWM37 Level Holding Control Register	FC4DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM4CH	PWM4 Counter High Byte	FC50H	-								x000,0000
PWM4CL	PWM4 Counter Low Byte	FC51H									0000,0000
PWM4CKS	PWM4 Clock Select Register	FC52H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000
PWM4TADCH	PWM4 Trigger ADC Counter High Byte	FC53H	-								x000,0000
PWM4TADCL	PWM4 Trigger ADC Counter Low Byte	FC54H									0000,0000
PWM4IF	PWM4 Interrupt Flag Register	FC55H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM4FDCR	PWM4 Fault Detection Control Register	FC56H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000
PWM40T1H	PWM40 T1 High Byte	FC60H	-								x000,0000
PWM40T1L	PWM40 T1 Low Byte	FC61H									0000,0000
PWM40T2H	PWM40 T2 High Byte	FC62H	-								x000,0000
PWM40T2L	PWM40 T2 Low Byte	FC63H									0000,0000
PWM40CR	PWM40 Control Register	FC64H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM40HLD	PWM40 Level Holding Control Register	FC65H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM41T1H	PWM41 T1 High Byte	FC68H	-								x000,0000
PWM41T1L	PWM41 T1 Low Byte	FC69H									0000,0000
PWM41T2H	PWM41 T2 High Byte	FC6AH	-								x000,0000
PWM41T2L	PWM41 T2 Low Byte	FC6BH									0000,0000
PWM41CR	PWM41 Control Register	FC6CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM41HLD	PWM41 Level Holding Control Register	FC6DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM42T1H	PWM42 T1 High Byte	FC70H	-								x000,0000
PWM42T1L	PWM42 T1 Low Byte	FC71H									0000,0000
PWM42T2H	PWM42 T2 High Byte	FC72H	-								x000,0000
PWM42T2L	PWM42 T2 Low Byte	FC73H									0000,0000
PWM42CR	PWM42 Control Register	FC74H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM42HLD	PWM42 Level Holding Control Register	FC75H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM43T1H	PWM43 T1 High Byte	FC78H	-								x000,0000
PWM43T1L	PWM43 T1 Low Byte	FC79H									0000,0000
PWM43T2H	PWM43 T2 High Byte	FC7AH	-								x000,0000
PWM43T2L	PWM43 T2 Low Byte	FC7BH									0000,0000
PWM43CR	PWM43 Control Register	FC7CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM43HLD	PWM43 Level Holding Control Register	FC7DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM44T1H	PWM44 T1 High Byte	FC80H	-								x000,0000
PWM44T1L	PWM44 T1 Low Byte	FC81H									0000,0000
PWM44T2H	PWM44 T2 High Byte	FC82H	-								x000,0000
PWM44T2L	PWM44 T2 Low Byte	FC83H									0000,0000
PWM44CR	PWM44 Control Register	FC84H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM44HLD	PWM44 Level Holding Control Register	FC85H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM45T1H	PWM45 T1 High Byte	FC88H	-								x000,0000
PWM45T1L	PWM45 T1 Low Byte	FC89H									0000,0000
PWM45T2H	PWM45 T2 High Byte	FC8AH	-								x000,0000
PWM45T2L	PWM45 T2 Low Byte	FC8BH									0000,0000
PWM45CR	PWM45 Control Register	FC8CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM45HLD	PWM45 Level Holding Control Register	FC8DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM46T1H	PWM46 T1 High Byte	FC90H	-								x000,0000
PWM46T1L	PWM46 T1 Low Byte	FC91H									0000,0000
PWM46T2H	PWM46 T2 High Byte	FC92H	-								x000,0000
PWM46T2L	PWM46 T2 Low Byte	FC93H									0000,0000
PWM46CR	PWM46 Control Register	FC94H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM46HLD	PWM46 Level Holding Control Register	FC95H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM47T1H	PWM47 T1 High Byte	FC98H	-								x000,0000
PWM47T1L	PWM47 T1 Low Byte	FC99H									0000,0000
PWM47T2H	PWM47 T2 High Byte	FC9AH	-								x000,0000
PWM47T2L	PWM47 T2 Low Byte	FC9BH									0000,0000
PWM47CR	PWM47 Control Register	FC9CH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM47HLD	PWM47 Level Holding Control Register	FC9DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM5CH	PWM5 Counter High Byte	FCA0H	-								x000,0000
PWM5CL	PWM5 Counter Low Byte	FCA1H									0000,0000
PWM5CKS	PWM5 Clock Select Register	FCA2H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000
PWM5IF	PWM5 Interrupt Flag Register	FCA5H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWM5FDCR	PWM5 Fault Detection Control Register	FCA6H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF	0000,0000
PWM50T1H	PWM50 T1 High Byte	FCB0H	-								x000,0000
PWM50T1L	PWM50 T1 Low Byte	FCB1H									0000,0000
PWM50T2H	PWM50 T2 High Byte	FCB2H	-								x000,0000
PWM50T2L	PWM50 T2 Low Byte	FCB3H									0000,0000
PWM50CR	PWM50 Control Register	FCB4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM50HLD	PWM50 Level Holding Control Register	FCB5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM51T1H	PWM51 T1 High Byte	FCB8H	-								x000,0000

PWM51T1L	PWM51 T1 Low Byte	FCB9H								0000,0000
PWM51T2H	PWM51 T2 High Byte	FCBAH	-							x000,0000
PWM51T2L	PWM51 T2 Low Byte	FCBBH								0000,0000
PWM51CR	PWM51 Control Register	FCBCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM51HLD	PWM51 Level Holding Control Register	FCBDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM52T1H	PWM52 T1 High Byte	FCC0H	-								x000,0000
PWM52T1L	PWM52 T1 Low Byte	FCC1H									0000,0000
PWM52T2H	PWM52 T2 High Byte	FCC2H	-								x000,0000
PWM52T2L	PWM52 T2 Low Byte	FCC3H									0000,0000
PWM52CR	PWM52 Control Register	FCC4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM52HLD	PWM52 Level Holding Control Register	FCC5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM53T1H	PWM53 T1 High Byte	FCC8H	-								x000,0000
PWM53T1L	PWM53 T1 Low Byte	FCC9H									0000,0000
PWM53T2H	PWM53 T2 High Byte	FCCAH	-								x000,0000
PWM53T2L	PWM53 T2 Low Byte	FCCBH									0000,0000
PWM53CR	PWM53 Control Register	FCCCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM53HLD	PWM53 Level Holding Control Register	FCCDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM54T1H	PWM54 T1 High Byte	FCD0H	-								x000,0000
PWM54T1L	PWM54 T1 Low Byte	FCD1H									0000,0000
PWM54T2H	PWM54 T2 High Byte	FCD2H	-								x000,0000
PWM54T2L	PWM54 T2 Low Byte	FCD3H									0000,0000
PWM54CR	PWM54 Control Register	FCD4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM54HLD	PWM54 Level Holding Control Register	FCD5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM55T1H	PWM55 T1 High Byte	FCD8H	-								x000,0000
PWM55T1L	PWM55 T1 Low Byte	FCD9H									0000,0000
PWM55T2H	PWM55 T2 High Byte	FCDAH	-								x000,0000
PWM55T2L	PWM55 T2 Low Byte	FCDBH									0000,0000
PWM55CR	PWM55 Control Register	FCDCH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM55HLD	PWM55 Level Holding Control Register	FCDDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM56T1H	PWM56 T1 High Byte	FCE0H	-								x000,0000
PWM56T1L	PWM56 T1 Low Byte	FCE1H									0000,0000
PWM56T2H	PWM56 T2 High Byte	FCE2H	-								x000,0000
PWM56T2L	PWM56 T2 Low Byte	FCE3H									0000,0000
PWM56CR	PWM56 Control Register	FCE4H	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM56HLD	PWM56 Level Holding Control Register	FCE5H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM57T1H	PWM57 T1 High Byte	FCE8H	-								x000,0000
PWM57T1L	PWM57 T1 Low Byte	FCE9H									0000,0000
PWM57T2H	PWM57 T2 High Byte	FCEAH	-								x000,0000
PWM57T2L	PWM57 T2 Low Byte	FCEBH									0000,0000
PWM57CR	PWM57 Control Register	FCECH	ENO	INI	-	-		ENI	ENT2I	ENT1I	00xx,x000
PWM57HLD	PWM57 Level Holding Control Register	FCEDH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00

18.1.1 Enhanced PWM global configuration register (PWMSET)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWMSET	F1H	ENGLBSET	PWMRST	ENPWM5	ENPWM4	ENPWM3	ENPWM2	ENPWM1	ENPWM0

ENGLBSET: Global setting function control bit.

0: 6 groups of PWM adopt independent setting methods. Each group of PWM is independently configured using the corresponding control bits of PWMCFG01/PWMCFG23/PWMCFG45.

1: 6 groups of PWM adopt the unified setting method. Each group of PWM is configured using the setting of PWM0 in PWMCFG01.

PWMRST: Software resets 6 groups of PWM.

0: No effect.

1: Reset the XFR registers for all PWMs, but do not reset SFRs which are required software reset.

ENPWM5: PWM5 enable bit (including PWM50 ~ PWM54).

0: disable PWM5

1: enable PWM5

ENPWM4: PWM4 enable bit (including PWM40~PWM47).

0: disable PWM4

1: enable PWM4

ENPWM3: PWM3 enable bit (including PWM30~PWM37).

0: disable PWM3

1: enable PWM3

ENPWM2: PWM2 enable bit (including PWM20~PWM27).

0: disable PWM2

1: enable PWM2

ENPWM1: PWM1 enable bit (including PWM10~PWM17).

0: disable PWM1

1: enable PWM1

ENPWM0: PWM0 enable bit (including PWM00~PWM07).

0: disable PWM0

1: enable PWM0

18.1.2 Enhanced PWM configuration registers (PWMCFGn)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWMCFG01	F6H	PWM1CBIF	EPWM1CBI	FLTPS0	PWM1CEN	PWM0CBIF	EPWM0CBI	ENPWM0TA	PWM0CEN
PWMCFG23	F7H	PWM3CBIF	EPWM3CBI	FLTPS1	PWM3CEN	PWM2CBIF	EPWM2CBI	ENPWM2TA	PWM2CEN
PWMCFG45	FEH	PWM5CBIF	EPWM5CBI	FLTPS2	PWM5CEN	PWM4CBIF	EPWM4CBI	ENPWM4TA	PWM4CEN

PWMnCBIF: The interrupt flag bit of PWMn when the PWMn counter returns to zero (n=0~5).

When the 15-bit PWMn counter overflows and returns to zero, this bit is set by hardware automatically and an interrupt is requested to CPU. This flag bit needs to be cleared by software.

EPWMnCBI: PWMn counter returns to zero interrupt enable bit. (n= 0 ~5)

0: disable PWMn counter returns to zero interrupt (PWMnCBIF is still set by hardware)

1: enable PWMn counter returns to zero interrupt

PWMnCEN: PWMn waveform generator starts counting control bit. (n= 0~5)

0: PWMn stops counting

1: PWMn starts counting

Important note about the PWMnCEN control bit:

- Once PWMnCEN is enabled, the internal PWMn counter will start counting immediately and compare the counting value with the value of T1/T2. So PWMnCEN must be enabled after all other PWM settings (including T1 / T2 settings, initial level setting, PWM fault detection setting and PWM interrupt setting) are completed.
- During the PWMn counter counting, when the PWMnCEN control bit is turned off, the PWMn counting will stop immediately. When the PWMnCEN control bit is enabled again, the PWMn counting will start counting from 0 again without remembering the count value before the PWMn stopped counting.

EPWMnTA: Whether the PWMn is associated with the ADC or not. (n=0,2,4)

0: PWMn is not associated with the ADC.

1: PWMn is associated with the ADC. A/D conversion is enabled to be triggered at a certain point in the PWMn cycle. PWMnTADCHand PWMnTADCL are used to set the counter value..

(**Note: The ADC_POWER and ADC_EPWMT bits in the ADC_CONTR register need to be set at the same time to enable the PWM triger function. PWM only sets ADC_START to 1 automatically. Only PWM0, PWM2 and PWM4 can trigger the ADC**.)

FLTPS0、FLTPS1、FLTPS2: External fault detection pin selection control bit

FLTPS2	**FLTPS1**	**FLTPS0**	**PWM0/PWM1/PWM3/PWM5** external fault detection pin	**PWM2** external fault detection pin	**PWM4** external fault detection pin
0	0	0	PWMFLT（P3.5）	PWMFLT（P3.5）	PWMFLT（P3.5）
0	0	1	PWMFLT（P3.5）	PWMFLT2（P0.6）	PWMFLT3（P0.7）
0	1	0	PWMFLT（P3.5）	PWMFLT3（P0.7）	PWMFLT2（P0.6）
0	1	1	PWMFLT2（P0.6）	PWMFLT2（P0.6）	PWMFLT2（P0.6）

1	0	0	PWMFLT2（P0.6）	PWMFLT（P3.5）	PWMFLT3（P0.7）
1	0	1	PWMFLT2（P0.6）	PWMFLT3（P0.7）	PWMFLT（P3.5）
1	1	0	PWMFLT3（P0.7）	PWMFLT（P3.5）	PWMFLT2（P0.6）
1	1	1	PWMFLT3（P0.7）	PWMFLT2（P0.6）	PWMFLT（P3.5）

18.1.3 PWM Interrupt Flag Registers (PWMnIF)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWM0IF	FF05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF
PWM1IF	FF55H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF
PWM2IF	FFA5H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF
PWM3IF	FC05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF
PWM4IF	FC55H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF
PWM5IF	FCA5H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF

CiIF: The interrupt flag bit of i-channel of PWMn. (n＝0~5, i＝0~7)

T1 and T2 of each PWM can be set. When a match event occurs at the set point, this bit is set by hardware automatically and an interrupt is requested to the CPU. This flag bit must be cleared by software.

18.1.4 PWM Fault Detection Control Registers (PWMnFDCR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWM0FDCR	FF06H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF
PWM1FDCR	FF56H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF
PWM2FDCR	FFA6H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF
PWM3FDCR	FC06H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF
PWM4FDCR	FC56H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF
PWM5FDCR	FCA6H	INVCMP	INVIO	ENFD	FLTFLIO	-	FDCMP	FDIO	FDIF

INVCMP: Fault signal of comparator result selection bit

0: the fault signal is the comparator result changing from low to high.

1: the fault signal is the comparator result changing from high to low.

INVIO: Fault signal of external port PWMFLT selection bit

0: the fault signal is the external port PWMFLT signal changing from low to high.

1: the fault signal is the external port PWMFLT signal changing from high to low.

(Note: The external fault detection port PWMFLT of each group of PWM is selected by FLTPS0, FLTPS1, FLTPS.)

ENFD: PWMn external fault detection enable bit. (n=0~5)

0: disable the PWMn external fault detection.

1: enable the PWMn external fault detection.

FLTFLIO: PWMn output port control bit when external PWMn fault occurs. (n=0~5)

0: the PWMn output port does not change when external PWMn fault occurs.

1: the PWMn outport port is set as high impedance input mode when external PWMn default occurs.

Note: Only the port whose corresponding ENO = 1 is forcibly in high impedance state.

EFDI: PWMn fault detection interrupt enable bit. (n=0,2,4)

0: disable PWMn fault detection interrupt (FDIF will still be set by hardware.)

1: enable PWMn fault detection interrupt

FDCMP: fault detection of comparator output enable bit. (n=0~5)

0: the comparator is not associated with PWM.

1: the source of PWMn fault detection is comparator output. (The fault type is set by INVCMP.)

FDIO: PWMFLT level fault detection enable bit. (n=0~5)

0: PWMFLT level is not associated with PWMn

1: the source of PWMn fault detection is PWMFLT. (The fault type is set by INVIO.)

FDIF: the interrupt flag bit of PWMn fault detection. (n=0~5)

It is set automatically by the hardware when a PWMn fault occurs. If EFDI=1, the program will jump to the corresponding interrupt entry to execute interrupt service routine. It should be cleared by software.

(Note: Only PWM0, PWM2 and PWM4 will be interrupted. PWM1, PWM3, PWM5 have fault detection function, but do not enter the interrupt service routine.)

18.1.5 PWM Counter Registers (PWMnCH, PWMnCL)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWM0CH	FF00H	-							
PWM0CL	FF01H								
PWM1CH	FF50H	-							
PWM1CL	FF51H								
PWM2CH	FFA0H	-							
PWM2CL	FFA1H								
PWM3CH	FC00H	-							
PWM3CL	FC01H								
PWM4CH	FC50H	-							
PWM4CL	FC51H								
PWM5CH	FCA0H	-							
PWM5CL	FCA1H								

PWMnCH: upper 7 bits of PWMn counter period value. (n=0~5)

PWMnCL: lower 8 bits of PWMn counter period value. (n=0~5)

The PWMn counter is a 15-bit register that can be set to any value between 1 and 32767 as the PWMn cycle. The counter inside the PWMn waveform generator counts from 0 and increments by 1 every PWMn clock cycle. When the internal counter reaches the PWMn cycle set by [PWMCH, PWMCL], the internal counter of the PWMn waveform generator will count from 0 again, and the PWMn return-to-zero interrupt flag bit PWMnCBIF will be set by hardware automatically. If ECBI = 1, the program will jump to the corresponding interrupt entry address to execute the interrupt service routine.

18.1.6 PWM Clock Selection Registers (PWMnCKS)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWM0CKS	FF02H	-	-	-	SELT2	PWM_PS[3:0]			
PWM1CKS	FF52H	-	-	-	SELT2	PWM_PS[3:0]			
PWM2CKS	FFA2H	-	-	-	SELT2	PWM_PS[3:0]			
PWM3CKS	FC02H	-	-	-	SELT2	PWM_PS[3:0]			
PWM4CKS	FC52H	-	-	-	SELT2	PWM_PS[3:0]			
PWM5CKS	FCA2H	-	-	-	SELT2	PWM_PS[3:0]			

SELT2: PWMn clock source selection bit. (n=0~5)

0: The PWMn clock source is the clock generated by the system clock being divided by the frequency divider.

1: The PWMn clock source is the overflow pulse of timer 2.

PWM_PS[3:0]: System clock prescaler parameter select bits

SELT2	PWM_PS[3:0]	PWMn input clock frequency
1	xxxx	Overflow pulse of Timer 2
0	0000	SYSclk/1
0	0001	SYSclk/2
0	0010	SYSclk/3
…	…	…
0	x	SYSclk/(x+1)
…	…	…
0	1111	SYSclk/16

PWM output frequency calculation formula

The output frequency calculation formula of the 6 groups of PWM is the same, and each group can be set with a

different frequency.

Clock source selection (SELT2)	PWM output frequency calculation formula
SELT2＝0 (System clock)	PWM output frequency ＝ System operating frequency SYSclk / (PWM_PS + 1)×（[PWMnCH, PWMnCL] + 1）
SELT2＝1 (Timer 2 overflow pulse)	PWM output frequency ＝ Timer 2 overflow pulse frequency / ([PWMnCH, PWMnCL] + 1)

18.1.7 PWM Trigger ADC counter Registers (PWMnTADC)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWM0TADCH	FF03H	-							
PWM0TADCL	FF04H								
PWM2TADCH	FFA3H	-							
PWM2TADCL	FFA4H								
PWM4TADCH	FC53H	-							
PWM4TADCL	FC54H								

PWMnTADCH: the upper 7 bits of the PWMn triggers ADC time. (n=0,2,4)

PWMnTADCL: the lower 8 bits of the PWMn triggers ADC time. (n=0,2,4)

If EPWMnTA =1 and ADC_POWER=1, {PWMnTADCH, PWMnTADCL} forms a 15-bit register. In the PWMn counting cycle, the hardware will trigger A/D conversion automatically when the internal PWMn counting value equals to the value of {PWMnTADCH, PWMnTADCL}.

18.1.8 PWM Level output setting count value Registers (PWMnT1，PWMnT2)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWM00T1H	FF10H	-							
PWM00T1L	FF11H								
PWM00T2H	FF12H	-							
PWM00T2L	FF13H								
PWM01T1H	FF18H	-							
PWM01T1L	FF19H								
PWM01T2H	FF1AH	-							
PWM01T2L	FF1BH								
PWM02T1H	FF20H	-							
PWM02T1L	FF21H								
PWM02T2H	FF22H	-							
PWM02T2L	FF23H								
PWM03T1H	FF28H	-							
PWM03T1L	FF29H								
PWM03T2H	FF2AH	-							
PWM03T2L	FF2BH								
PWM04T1H	FF30H	-							
PWM04T1L	FF31H								
PWM04T2H	FF32H	-							
PWM04T2L	FF33H								
PWM05T1H	FF38H	-							
PWM05T1L	FF39H								
PWM05T2H	FF3AH	-							
PWM05T2L	FF3BH								
PWM06T1H	FF40H	-							
PWM06T1L	FF41H								
PWM06T2H	FF42H	-							

PWM06T2L	FF43H		
PWM07T1H	FF48H	-	
PWM07T1L	FF49H		
PWM07T2H	FF4AH	-	
PWM07T2L	FF4BH		
PWM10T1H	FF60H	-	
PWM10T1L	FF61H		
PWM10T2H	FF62H	-	
PWM10T2L	FF63H		
PWM11T1H	FF68H	-	
PWM11T1L	FF69H		
PWM11T2H	FF6AH	-	
PWM11T2L	FF6BH		
PWM12T1H	FF70H	-	
PWM12T1L	FF71H		
PWM12T2H	FF72H	-	
PWM12T2L	FF73H		
PWM13T1H	FF78H	-	
PWM13T1L	FF79H		
PWM13T2H	FF7AH	-	
PWM13T2L	FF7BH		
PWM14T1H	FF80H	-	
PWM14T1L	FF81H		
PWM14T2H	FF82H	-	
PWM14T2L	FF83H		
PWM15T1H	FF88H	-	
PWM15T1L	FF89H		
PWM15T2H	FF8AH	-	
PWM15T2L	FF8BH		
PWM16T1H	FF90H	-	
PWM16T1L	FF91H		
PWM16T2H	FF92H	-	
PWM16T2L	FF93H		
PWM17T1H	FF98H	-	
PWM17T1L	FF99H		
PWM17T2H	FF9AH	-	
PWM17T2L	FF9BH		
PWM20T1H	FFB0H	-	
PWM20T1L	FFB1H		
PWM20T2H	FFB2H	-	
PWM20T2L	FFB3H		
PWM21T1H	FFB8H	-	
PWM21T1L	FFB9H		
PWM21T2H	FFBAH	-	
PWM21T2L	FFBBH		
PWM22T1H	FFC0H	-	
PWM22T1L	FFC1H		
PWM22T2H	FFC2H	-	
PWM22T2L	FFC3H		
PWM23T1H	FFC8H	-	
PWM23T1L	FFC9H		
PWM23T2H	FFCAH	-	
PWM23T2L	FFCBH		
PWM24T1H	FFD0H	-	
PWM24T1L	FFD1H		
PWM24T2H	FFD2H	-	
PWM24T2L	FFD3H		
PWM25T1H	FFD8H	-	
PWM25T1L	FFD9H		
PWM25T2H	FFDAH	-	
PWM25T2L	FFDBH		
PWM26T1H	FFE0H	-	
PWM26T1L	FFE1H		
PWM26T2H	FFE2H	-	
PWM26T2L	FFE3H		

STC MCU

PWM27T1H	FFE8H	-	
PWM27T1L	FFE9H		
PWM27T2H	FFEAH	-	
PWM27T2L	FFEBH		
PWM30T1H	FC10H	-	
PWM30T1L	FC11H		
PWM30T2H	FC12H	-	
PWM30T2L	FC13H		
PWM31T1H	FC18H	-	
PWM31T1L	FC19H		
PWM31T2H	FC1AH	-	
PWM31T2L	FC1BH		
PWM32T1H	FC20H	-	
PWM32T1L	FC21H		
PWM32T2H	FC22H	-	
PWM32T2L	FC23H		
PWM33T1H	FC28H	-	
PWM33T1L	FC29H		
PWM33T2H	FC2AH	-	
PWM33T2L	FC2BH		
PWM34T1H	FC30H	-	
PWM34T1L	FC31H		
PWM34T2H	FC32H	-	
PWM34T2L	FC33H		
PWM35T1H	FC38H	-	
PWM35T1L	FC39H		
PWM35T2H	FC3AH	-	
PWM35T2L	FC3BH		
PWM36T1H	FC40H	-	
PWM36T1L	FC41H		
PWM36T2H	FC42H	-	
PWM36T2L	FC43H		
PWM37T1H	FC48H	-	
PWM37T1L	FC49H		
PWM37T2H	FC4AH	-	
PWM37T2L	FC4BH		
PWM40T1H	FC60H	-	
PWM40T1L	FC61H		
PWM40T2H	FC62H	-	
PWM40T2L	FC63H		
PWM41T1H	FC68H	-	
PWM41T1L	FC69H		
PWM41T2H	FC6AH	-	
PWM41T2L	FC6BH		
PWM42T1H	FC70H	-	
PWM42T1L	FC71H		
PWM42T2H	FC72H	-	
PWM42T2L	FC73H		
PWM43T1H	FC78H	-	
PWM43T1L	FC79H		
PWM43T2H	FC7AH	-	
PWM43T2L	FC7BH		
PWM44T1H	FC80H	-	
PWM44T1L	FC81H		
PWM44T2H	FC82H	-	
PWM44T2L	FC83H		
PWM45T1H	FC88H	-	
PWM45T1L	FC89H		
PWM45T2H	FC8AH	-	
PWM45T2L	FC8BH		
PWM46T1H	FC90H	-	
PWM46T1L	FC91H		
PWM46T2H	FC92H	-	
PWM46T2L	FC93H		
PWM47T1H	FC98H	-	

STC MCU

PWM47T1L	FC99H		
PWM47T2H	FC9AH	-	
PWM47T2L	FC9BH		
PWM50T1H	FCB0H	-	
PWM50T1L	FCB1H		
PWM50T2H	FCB2H	-	
PWM50T2L	FCB3H		
PWM51T1H	FCB8H	-	
PWM51T1L	FCB9H		
PWM51T2H	FCBAH	-	
PWM51T2L	FCBBH		
PWM52T1H	FCC0H	-	
PWM52T1L	FCC1H		
PWM52T2H	FCC2H	-	
PWM52T2L	FCC3H		
PWM53T1H	FCC8H	-	
PWM53T1L	FCC9H		
PWM53T2H	FCCAH	-	
PWM53T2L	FCCBH		
PWM54T1H	FCD0H	-	
PWM54T1L	FCD1H		
PWM54T2H	FCD2H	-	
PWM54T2L	FCD3H		
PWM55T1H	FCD8H	-	
PWM55T1L	FCD9H		
PWM55T2H	FCDAH	-	
PWM55T2L	FCDBH		
PWM56T1H	FCE0H	-	
PWM56T1L	FCE1H		
PWM56T2H	FCE2H	-	
PWM56T2L	FCE3H		
PWM57T1H	FCE8H	-	
PWM57T1L	FCE9H		
PWM57T2H	FCEAH	-	
PWM57T2L	FCEBH		

PWMniT1H: The upper 7 bits of T1 counter value of channel i of PWMn. (n= 0~5 and i= 0~7)

PWMniT1L: The lower 8 bits of T1 counter value of channel i of PWMn. (n= 0~5 and i= 0~7)

PWMniT2H: The upper 7 bits of T2 counter value of channel i of PWMn. (n= 0~5 and i= 0~7)

PWMniT2L: The lower 8 bits of T2counter value of channel i of PWMn. (n= 0~5 and i= 0~7)

{PWMniT1H, PWMniT1L} and {PWMniT2H, PWMniT2L} of every channel of every PWMn are combined into two 15-bit registers, which are used to control the two flip points of the PWM output waveform in every PWM cycle of each PWM. During the counting cycle of PWMn, the output of PWM will be low level when the internal counting value of PWMn is equal to the value of T1 set by{PWMniT1H, PWMniT1L}. And the output of the PWM will be high level when the internal counting value of PWM is queal to the value of the T2 set by {PWMnT2H, PWMnT2L}.

Note: When the values of {PWMniT1H, PWMniT1L} and {PWMniT2H, PWMniT2L} are set equal, and if the internal count value of PWM is equal to the set value of T1 / T2, it will output low level.

18.1.9 PWM Channel Control Registers (PWMnCR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWM00CR	FF14H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM01CR	FF1CH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM02CR	FF24H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM03CR	FF2CH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM04CR	FF34H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM05CR	FF3CH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM06CR	FF44H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM07CR	FF4CH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM10CR	FF64H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I

PWM11CR	FF6CH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM12CR	FF74H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM13CR	FF7CH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM14CR	FF84H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM15CR	FF8CH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM16CR	FF94H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM17CR	FF9CH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM20CR	FFB4H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM21CR	FFBCH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM22CR	FFC4H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM23CR	FFCCH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM24CR	FFD4H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM25CR	FFDCH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM26CR	FFE4H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM27CR	FFECH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM30CR	FC14H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM31CR	FC1CH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM32CR	FC24H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM33CR	FC2CH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM34CR	FC34H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM35CR	FC3CH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM36CR	FC44H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM37CR	FC4CH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM40CR	FC64H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM41CR	FC6CH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM42CR	FC74H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM43CR	FC7CH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM44CR	FC84H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM45CR	FC8CH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM46CR	FC94H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM47CR	FC9CH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM50CR	FCB4H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM51CR	FCBCH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM52CR	FCC4H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM53CR	FCCCH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM54CR	FCD4H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM55CR	FCDCH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM56CR	FCE4H	ENO	INI	-	-	-	ENI	ENT2I	ENT1I
PWM57CR	FCECH	ENO	INI	-	-	-	ENI	ENT2I	ENT1I

ENO: PWMni output enable bit. (n= 0~5 and i= 0~7)

0: the corresponding port of PWMn channel i is GPIO.

1: the corresponding port of PWMn channel i is PWM output port, which is controlled by the PWMn waveform generator.

INI: the initial level of PWMni output. (n= 0~5 and i= 0~7)

0: the initial level of PWMn channel i is low.

1: the initial level of PWMn channel i is high.

ENI: interrupt enable bit of PWMn channel i. (n= 0~5 and i= 0~7)

0: disable PWMn channel i interrupt.

1: enable PWMn channel i interrupt.

ENT2I: interrupt enable bit of the second flip point of PWMn channel i. (n= 0~5 and i= 0~7)

0: disable the interrupt of the second flip point of PWMn channel i.

1: enable the interrupt of the second flip point of PWMn channel i.

ENT1I: interrupt enable bit of the first flip point of PWMn channel i. (n= 0~5 and i= 0~7)

0: disable the interrupt of the first flip point of PWMn channel i.

1: enable the interrupt of the first flip point of PWMn channel i.

18.1.10 PWM Channel Level Holding Control Registers (PWMnHLD)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWM00HLD	FF15H	-	-	-	-	-	-	HLDH	HLDL
PWM01HLD	FF1DH	-	-	-	-	-	-	HLDH	HLDL
PWM02HLD	FF25H	-	-	-	-	-	-	HLDH	HLDL
PWM03HLD	FF2DH	-	-	-	-	-	-	HLDH	HLDL
PWM04HLD	FF35H	-	-	-	-	-	-	HLDH	HLDL
PWM05HLD	FF3DH	-	-	-	-	-	-	HLDH	HLDL
PWM06HLD	FF45H	-	-	-	-	-	-	HLDH	HLDL
PWM07HLD	FF4DH	-	-	-	-	-	-	HLDH	HLDL
PWM10HLD	FF65H	-	-	-	-	-	-	HLDH	HLDL
PWM11HLD	FF6DH	-	-	-	-	-	-	HLDH	HLDL
PWM12HLD	FF75H	-	-	-	-	-	-	HLDH	HLDL
PWM13HLD	FF7DH	-	-	-	-	-	-	HLDH	HLDL
PWM14HLD	FF85H	-	-	-	-	-	-	HLDH	HLDL
PWM15HLD	FF8DH	-	-	-	-	-	-	HLDH	HLDL
PWM16HLD	FF95H	-	-	-	-	-	-	HLDH	HLDL
PWM17HLD	FF9DH	-	-	-	-	-	-	HLDH	HLDL
PWM20HLD	FFB5H	-	-	-	-	-	-	HLDH	HLDL
PWM21HLD	FFBDH	-	-	-	-	-	-	HLDH	HLDL
PWM22HLD	FFC5H	-	-	-	-	-	-	HLDH	HLDL
PWM23HLD	FFCDH	-	-	-	-	-	-	HLDH	HLDL
PWM24HLD	FFD5H	-	-	-	-	-	-	HLDH	HLDL
PWM25HLD	FFDDH	-	-	-	-	-	-	HLDH	HLDL
PWM26HLD	FFE5H	-	-	-	-	-	-	HLDH	HLDL
PWM27HLD	FFEDH	-	-	-	-	-	-	HLDH	HLDL
PWM30HLD	FC15H	-	-	-	-	-	-	HLDH	HLDL
PWM31HLD	FC1DH	-	-	-	-	-	-	HLDH	HLDL
PWM32HLD	FC25H	-	-	-	-	-	-	HLDH	HLDL
PWM33HLD	FC2DH	-	-	-	-	-	-	HLDH	HLDL
PWM34HLD	FC35H	-	-	-	-	-	-	HLDH	HLDL
PWM35HLD	FC3DH	-	-	-	-	-	-	HLDH	HLDL
PWM36HLD	FC45H	-	-	-	-	-	-	HLDH	HLDL
PWM37HLD	FC4DH	-	-	-	-	-	-	HLDH	HLDL
PWM40HLD	FC65H	-	-	-	-	-	-	HLDH	HLDL
PWM41HLD	FC6DH	-	-	-	-	-	-	HLDH	HLDL
PWM42HLD	FC75H	-	-	-	-	-	-	HLDH	HLDL
PWM43HLD	FC7DH	-	-	-	-	-	-	HLDH	HLDL
PWM44HLD	FC85H	-	-	-	-	-	-	HLDH	HLDL
PWM45HLD	FC8DH	-	-	-	-	-	-	HLDH	HLDL
PWM46HLD	FC95H	-	-	-	-	-	-	HLDH	HLDL
PWM47HLD	FC9DH	-	-	-	-	-	-	HLDH	HLDL
PWM50HLD	FCB5H	-	-	-	-	-	-	HLDH	HLDL
PWM51HLD	FCBDH	-	-	-	-	-	-	HLDH	HLDL
PWM52HLD	FCC5H	-	-	-	-	-	-	HLDH	HLDL
PWM53HLD	FCCDH	-	-	-	-	-	-	HLDH	HLDL
PWM54HLD	FCD5H	-	-	-	-	-	-	HLDH	HLDL
PWM55HLD	FCDDH	-	-	-	-	-	-	HLDH	HLDL
PWM56HLD	FCE5H	-	-	-	-	-	-	HLDH	HLDL
PWM57HLD	FCEDH	-	-	-	-	-	-	HLDH	HLDL

HLDH: PWMn channel i outputs high compulsively control bit. (n= 0~5 and i= 0~7)

0: PWMn channel i output normally.

1: PWMn channel i outputs high compulsively.

HLDL: PWMn channel i outputs low compulsively control bit. (n= 0~5 and i= 0~7)

0: PWMn channel i output normally.

1: PWMn channel i outputs low compulsively.

STC MCU

18.2 Example Routines

18.2.1 Output waveforms with arbitrary period and arbitrary duty

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P_SW2           =   0xba;

sfr         PWMSET          =   0xF1;
sfr         PWMCFG01        =   0xF6;
sfr         PWMCFG23        =   0xF7;
sfr         PWMCFG45        =   0xFE;

#define     PWM0C               (*(unsigned int  volatile xdata *)0xFF00)
#define     PWM0CH              (*(unsigned char volatile xdata *)0xFF00)
#define     PWM0CL              (*(unsigned char volatile xdata *)0xFF01)
#define     PWM0CKS             (*(unsigned char volatile xdata *)0xFF02)
#define     PWM0TADC            (*(unsigned int  volatile xdata *)0xFF03)
#define     PWM0TADCH           (*(unsigned char volatile xdata *)0xFF03)
#define     PWM0TADCL           (*(unsigned char volatile xdata *)0xFF04)
#define     PWM0IF              (*(unsigned char volatile xdata *)0xFF05)
#define     PWM0FDCR            (*(unsigned char volatile xdata *)0xFF06)
#define     PWM00T1             (*(unsigned int  volatile xdata *)0xFF10)
#define     PWM00T1H            (*(unsigned char volatile xdata *)0xFF10)
#define     PWM00T1L            (*(unsigned char volatile xdata *)0xFF11)
#define     PWM00T2H            (*(unsigned char volatile xdata *)0xFF12)
#define     PWM00T2             (*(unsigned int  volatile xdata *)0xFF12)
#define     PWM00T2L            (*(unsigned char volatile xdata *)0xFF13)
#define     PWM00CR             (*(unsigned char volatile xdata *)0xFF14)
#define     PWM00HLD            (*(unsigned char volatile xdata *)0xFF15)

sfr         P0M1            =   0x93;
sfr         P0M0            =   0x94;
sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P2M1            =   0x95;
sfr         P2M0            =   0x96;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P4M1            =   0xb3;
sfr         P4M0            =   0xb4;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
```

```
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    PWMSET = 0x01;    //Enable PWM0 module (The configuration is effective only after the module is enabled.)

    P_SW2 = 0x80;
    PWM0CKS = 0x00;                         //The clock of PWM0 is the system clock
    PWM0C = 0x1000;                         //Set the PWM0 cycle  to 1000H PWM clocks
    PWM00T1= 0x0100;          //At the count value of 100H, the PWM00 channel outputs low level.
    PWM00T2= 0x0500;          //At the count value of 500H, the PWM00 channel outputs high level.
    PWM00CR= 0x80;                          //enable PWM00 output
    P_SW2 = 0x00;

    PWMCFG01 = 0x01;                        //Start PWM0 module

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

PWMSET      DATA    0F1H
PWMCFG01    DATA    0F6H
PWMCFG23    DATA    0F7H
PWMCFG45    DATA    0FEH

PWM0CH      EQU     0FF00H
PWM0CL      EQU     0FF01H
PWM0CKS     EQU     0FF02H
PWM0TADCH   EQU     0FF03H
PWM0TADCL   EQU     0FF04H
PWM0IF      EQU     0FF05H
PWM0FDCR    EQU     0FF06H
PWM00T1H    EQU     0FF10H
PWM00T1L    EQU     0FF11H
PWM00T2H    EQU     0FF12H
PWM00T2L    EQU     0FF13H
PWM00CR     EQU     0FF14H
PWM00HLD    EQU     0FF15H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
```

STC MCU

```
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     PWMSET,#01H     ;Enable PWM0 module (The configuration is effective only after the module is enabled.)

            MOV     P_SW2,#80H
            CLR     A
            MOV     DPTR,#PWM0CKS
            MOVX    @DPTR,A                 ;The clock of PWM0 is the system clock
            MOV     A,#10H
            MOV     DPTR,#PWM0CH            ;Set the PWM0 cycle  to 1000H PWM clocks
            MOVX    @DPTR,A
            MOV     A,#00H
            MOV     DPTR,#PWM0CL
            MOVX    @DPTR,A
            MOV     A,#01H
            MOV     DPTR,#PWM00T1H  ;At the count value of 100H, the PWM00 channel outputs low level.
            MOVX    @DPTR,A
            MOV     A,#00H
            MOV     DPTR,#PWM00T1L
            MOVX    @DPTR,A
            MOV     A,#05H
            MOV     DPTR,#PWM00T2H ;At the count value of 500H, the PWM00 channel outputs high level.
            MOVX    @DPTR,A
            MOV     A,#00H
            MOV     DPTR,#PWM00T2L
            MOVX    @DPTR,A
            MOV     A,#80H
            MOV     DPTR,#PWM00CR           ;enable PWM00 output
            MOVX    @DPTR,A
            MOV     P_SW2,#00H

            MOV     PWMCFG01,#01H           ;Start PWM0 module

            JMP     $

            END
```

18.2.2 Two-channel PWMs realize complementary symmetrical waveform with dead-time control

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;

sfr     PWMSET      =   0xF1;
sfr     PWMCFG01    =   0xF6;
sfr     PWMCFG23    =   0xF7;
sfr     PWMCFG45    =   0xFE;

#define PWM0C           (*(unsigned int  volatile xdata *)0xFF00)
#define PWM0CH          (*(unsigned char volatile xdata *)0xFF00)
#define PWM0CL          (*(unsigned char volatile xdata *)0xFF01)
#define PWM0CKS         (*(unsigned char volatile xdata *)0xFF02)
#define PWM0TADC        (*(unsigned int  volatile xdata *)0xFF03)
#define PWM0TADCH       (*(unsigned char volatile xdata *)0xFF03)
#define PWM0TADCL       (*(unsigned char volatile xdata *)0xFF04)
#define PWM0IF          (*(unsigned char volatile xdata *)0xFF05)
#define PWM0FDCR        (*(unsigned char volatile xdata *)0xFF06)
#define PWM00T1         (*(unsigned int  volatile xdata *)0xFF10)
#define PWM00T1H        (*(unsigned char volatile xdata *)0xFF10)
#define PWM00T1L        (*(unsigned char volatile xdata *)0xFF11)
#define PWM00T2H        (*(unsigned char volatile xdata *)0xFF12)
#define PWM00T2         (*(unsigned int  volatile xdata *)0xFF12)
#define PWM00T2L        (*(unsigned char volatile xdata *)0xFF13)
#define PWM00CR         (*(unsigned char volatile xdata *)0xFF14)
#define PWM00HLD        (*(unsigned char volatile xdata *)0xFF15)
#define PWM01T1         (*(unsigned int  volatile xdata *)0xFF18)
#define PWM01T1H        (*(unsigned char volatile xdata *)0xFF18)
#define PWM01T1L        (*(unsigned char volatile xdata *)0xFF19)
#define PWM01T2         (*(unsigned int  volatile xdata *)0xFF1A)
#define PWM01T2H        (*(unsigned char volatile xdata *)0xFF1A)
#define PWM01T2L        (*(unsigned char volatile xdata *)0xFF1B)
#define PWM01CR         (*(unsigned char volatile xdata *)0xFF1C)
#define PWM01HLD        (*(unsigned char volatile xdata *)0xFF1D)

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;
```

```
void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    PWMSET = 0x01;                  //Enable PWM0 module (The configuration is effective only after the module is enabled.)

    P_SW2 = 0x80;
    PWM0CKS = 0x00;                 //The clock of PWM0 is the system clock
    PWM0C = 0x0800;                 //Set the PWM0 cycle to 0800H PWM clocks
    PWM00T1= 0x0100;                //At the count value of 100H, PWM00 outputs low level.
    PWM00T2= 0x0700;                //At the count value of 700H, PWM00 outputs high level.
    PWM01T2= 0x0080;                //At the count value of 0080H, PWM01 outputs high level.
    PWM01T1= 0x0780;                //At the count value of 0780H, PWM01 outputs low level.
    PWM00CR= 0x80;                  //enable PWM00 output
    PWM01CR= 0x80;                  //enable PWM01 output
    P_SW2 = 0x00;

    PWMCFG01 = 0x01;                //Start PWM0 module

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

PWMSET      DATA    0F1H
PWMCFG01    DATA    0F6H
PWMCFG23    DATA    0F7H
PWMCFG45    DATA    0FEH

PWM0CH      EQU     0FF00H
PWM0CL      EQU     0FF01H
PWM0CKS     EQU     0FF02H
PWM0TADCH   EQU     0FF03H
PWM0TADCL   EQU     0FF04H
PWM0IF      EQU     0FF05H
PWM0FDCR    EQU     0FF06H
PWM00T1H    EQU     0FF10H
PWM00T1L    EQU     0FF11H
PWM00T2H    EQU     0FF12H
PWM00T2L    EQU     0FF13H
PWM00CR     EQU     0FF14H
PWM00HLD    EQU     0FF15H
PWM01T1H    EQU     0FF18H
```

```
PWM01T1L    EQU     0FF19H
PWM01T2H    EQU     0FF1AH
PWM01T2L    EQU     0FF1BH
PWM01CR     EQU     0FF1CH
PWM01HLD    EQU     0FF1DH

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     PWMSET,#01H         ;Enable PWM0 module (The configuration is effective only
after the module is enabled.)

            MOV     P_SW2,#80H
            CLR     A
            MOV     DPTR,#PWM0CKS
            MOVX    @DPTR,A             ;The clock of PWM0 is the system clock
            MOV     A,#08H
            MOV     DPTR,#PWM0CH        ;Set the PWM0 cycle to 0800H PWM clocks
            MOVX    @DPTR,A
            MOV     A,#00H
            MOV     DPTR,#PWM0CL
            MOVX    @DPTR,A
            MOV     A,#01H
            MOV     DPTR,#PWM00T1H      ;At the count value of 0100H, PWM00 outputs low level.
            MOVX    @DPTR,A
            MOV     A,#00H
            MOV     DPTR,#PWM00T1L
            MOVX    @DPTR,A
            MOV     A,#07H
            MOV     DPTR,#PWM00T2H      ;At the count value of 0700H, PWM00 outputs high level.
```

```
        MOVX    @DPTR,A
        MOV     A,#00H
        MOV     DPTR,#PWM00T2L
        MOVX    @DPTR,A
        MOV     A,#00H
        MOV     DPTR,#PWM01T2H      ;At the count value of 0080H, PWM01 outputs high level.
        MOVX    @DPTR,A
        MOV     A,#80H
        MOV     DPTR,#PWM01T2L
        MOVX    @DPTR,A
        MOV     A,#07H
        MOV     DPTR,#PWM01T1H      ;At the count value of 0780H, PWM01 outputs low level.
        MOVX    @DPTR,A
        MOV     A,#80H
        MOV     DPTR,#PWM01T1L
        MOVX    @DPTR,A
        MOV     A,#080H
        MOV     DPTR,#PWM00CR       ;enable PWM00 output
        MOVX    @DPTR,A
        MOV     A,#80H
        MOV     DPTR,#PWM01CR       ;enable PWM01 output
        MOVX    @DPTR,A
        MOV     P_SW2,#00H

        MOV     PWMCFG01,#01H       ;Start PWM0 module

        JMP     $

        END
```

18.2.3 PWM Implements Gradient Light (Breathing Light)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   CYCLE             0x1000

sfr       P_SW2       =     0xba;

sfr       PWMSET      =     0xF1;
sfr       PWMCFG01    =     0xF6;
sfr       PWMCFG23    =     0xF7;
sfr       PWMCFG45    =     0xFE;

#define   PWM0C             (*(unsigned int  volatile xdata *)0xFF00)
#define   PWM0CH            (*(unsigned char volatile xdata *)0xFF00)
#define   PWM0CL            (*(unsigned char volatile xdata *)0xFF01)
#define   PWM0CKS           (*(unsigned char volatile xdata *)0xFF02)
#define   PWM0TADC          (*(unsigned int  volatile xdata *)0xFF03)
#define   PWM0TADCH         (*(unsigned char volatile xdata *)0xFF03)
#define   PWM0TADCL         (*(unsigned char volatile xdata *)0xFF04)
#define   PWM0IF            (*(unsigned char volatile xdata *)0xFF05)
```

```
#define    PWM0FDCR        (*(unsigned char volatile xdata *)0xFF06)
#define    PWM00T1         (*(unsigned int  volatile xdata *)0xFF10)
#define    PWM00T1H        (*(unsigned char volatile xdata *)0xFF10)
#define    PWM00T1L        (*(unsigned char volatile xdata *)0xFF11)
#define    PWM00T2H        (*(unsigned char volatile xdata *)0xFF12)
#define    PWM00T2         (*(unsigned int  volatile xdata *)0xFF12)
#define    PWM00T2L        (*(unsigned char volatile xdata *)0xFF13)
#define    PWM00CR         (*(unsigned char volatile xdata *)0xFF14)
#define    PWM00HLD        (*(unsigned char volatile xdata *)0xFF15)

sfr        P0M1        =   0x93;
sfr        P0M0        =   0x94;
sfr        P1M1        =   0x91;
sfr        P1M0        =   0x92;
sfr        P2M1        =   0x95;
sfr        P2M0        =   0x96;
sfr        P3M1        =   0xb1;
sfr        P3M0        =   0xb2;
sfr        P4M1        =   0xb3;
sfr        P4M0        =   0xb4;
sfr        P5M1        =   0xc9;
sfr        P5M0        =   0xca;

void PWM0_Isr() interrupt 22
{
    static bit dir = 1;
    static int val = 0;

    if (PWMCFG01 & 0x08)
    {
        PWMCFG01 &= ~0x08;                  //Clear interrupt flag
        if (dir)
        {
            val++;
            if (val >= CYCLE) dir = 0;
        }
        else
        {
            val--;
            if (val <= 1) dir = 1;
        }
        _push_(P_SW2);
        P_SW2 |= 0x80;
        PWM00T2 = val;
        _pop_(P_SW2);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
```

STC MCU

```
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    PWMSET = 0x01;                  //Enable PWM0 module (The configuration is effective only after the module is enabled.)

    P_SW2 = 0x80;
    PWM0CKS = 0x00;                 // The clock of PWM0 is the system clock
    PWM0C = CYCLE;                  //Set PWM0 period
    PWM00T1= 0x0000;
    PWM00T2= 0x0001;
    PWM00CR= 0x80;                  //enable PWM00 output
    P_SW2 = 0x00;

    PWMCFG01 = 0x05;                //Start PWM0 module and enable PWM0 interrput
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CYCLE       EQU     1000H

P_SW2       DATA    0BAH

PWMSET      DATA    0F1H
PWMCFG01    DATA    0F6H
PWMCFG23    DATA    0F7H
PWMCFG45    DATA    0FEH

PWM0CH      EQU     0FF00H
PWM0CL      EQU     0FF01H
PWM0CKS     EQU     0FF02H
PWM0TADCH   EQU     0FF03H
PWM0TADCL   EQU     0FF04H
PWM0IF      EQU     0FF05H
PWM0FDCR    EQU     0FF06H
PWM00T1H    EQU     0FF10H
PWM00T1L    EQU     0FF11H
PWM00T2H    EQU     0FF12H
PWM00T2L    EQU     0FF13H
PWM00CR     EQU     0FF14H
PWM00HLD    EQU     0FF15H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
```

STC MCU

```
P5M0        DATA    0CAH

DIR         BIT     20H.0
VALL        DATA    21H
VALH        DATA    22H

            ORG     0000H
            LJMP    MAIN
            ORG     00B3H
            LJMP    PWM0ISR

            ORG     0100H
PWM0ISR:
            PUSH    ACC
            PUSH    PSW
            PUSH    DPL
            PUSH    DPH
            PUSH    P_SW2

            MOV     P_SW2,#80H
            MOV     A,PWMCFG01
            JNB     ACC.3,ISREXIT
            ANL     PWMCFG01,#NOT 08H       ;Clear interrupt flag
            JNB     DIR,PWMDN
PWMUP:
            MOV     A,VALL
            ADD     A,#1
            MOV     VALL,A
            MOV     A,VALH
            ADDC    A,#0
            MOV     VALH,A
            CJNE    A,#HIGH CYCLE,SETPWM
            MOV     A,VALL
            CJNE    A,#LOW CYCLE,SETPWM
            CLR     DIR
            JMP     SETPWM
PWMDN:
            MOV     A,VALL
            ADD     A,#0FFH
            MOV     VALL,A
            MOV     A,VALH
            ADDC    A,#0FFH
            MOV     VALH,A
            JNZ     SETPWM
            MOV     A,VALL
            CJNE    A,#1,SETPWM
            SETB    DIR
SETPWM:
            MOV     A,VALH
            MOV     DPTR,#PWM00T2H
            MOVX    @DPTR,A
            MOV     A,VALL
            MOV     DPTR,#PWM00T2L
            MOVX    @DPTR,A
ISREXIT:
            POP     P_SW2
            POP     DPH
            POP     DPL
            POP     PSW
```

STC MCU

```
        POP     ACC
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        SETB    DIR
        MOV     VALH,#00H
        MOV     VALL,#01H

        MOV     PWMSET,#01H             ;Enable PWM0 module (The configuration is effective only
after the module is enabled.)

        MOV     P_SW2,#80H
        CLR     A
        MOV     DPTR,#PWM0CKS
        MOVX    @DPTR,A                 ;The clock of PWM0 is the system clock
        MOV     A,#HIGH CYCLE
        MOV     DPTR,#PWM0CH            ;Set PWM0 period
        MOVX    @DPTR,A
        MOV     A,#LOW CYCLE
        MOV     DPTR,#PWM0CL
        MOVX    @DPTR,A
        MOV     A,#00H
        MOV     DPTR,#PWM00T1H
        MOVX    @DPTR,A
        MOV     A,#00H
        MOV     DPTR,#PWM00T1L
        MOVX    @DPTR,A
        MOV     A,VALH
        MOV     DPTR,#PWM00T2H
        MOVX    @DPTR,A
        MOV     A,VALL
        MOV     DPTR,#PWM00T2L
        MOVX    @DPTR,A
        MOV     A,#80H
        MOV     DPTR,#PWM00CR           ;enable PWM00 output
        MOVX    @DPTR,A
        MOV     P_SW2,#00H

        MOV     PWMCFG01,#05H           ;Start PWM0 module and enable PWM0 interrupt
        SETB    EA
        JMP     $

        END
```

STC MCU

18.2.4 Use PWM to trigger ADC conversion

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;

sfr     PWMSET      =   0xF1;
sfr     PWMCFG01    =   0xF6;
sfr     PWMCFG23    =   0xF7;
sfr     PWMCFG45    =   0xFE;

#define PWM0C           (*(unsigned int  volatile xdata *)0xFF00)
#define PWM0CH          (*(unsigned char volatile xdata *)0xFF00)
#define PWM0CL          (*(unsigned char volatile xdata *)0xFF01)
#define PWM0CKS         (*(unsigned char volatile xdata *)0xFF02)
#define PWM0TADC        (*(unsigned int  volatile xdata *)0xFF03)
#define PWM0TADCH       (*(unsigned char volatile xdata *)0xFF03)
#define PWM0TADCL       (*(unsigned char volatile xdata *)0xFF04)
#define PWM0IF          (*(unsigned char volatile xdata *)0xFF05)
#define PWM0FDCR        (*(unsigned char volatile xdata *)0xFF06)
#define PWM00T1         (*(unsigned int  volatile xdata *)0xFF10)
#define PWM00T1H        (*(unsigned char volatile xdata *)0xFF10)
#define PWM00T1L        (*(unsigned char volatile xdata *)0xFF11)
#define PWM00T2H        (*(unsigned char volatile xdata *)0xFF12)
#define PWM00T2         (*(unsigned int  volatile xdata *)0xFF12)
#define PWM00T2L        (*(unsigned char volatile xdata *)0xFF13)
#define PWM00CR         (*(unsigned char volatile xdata *)0xFF14)
#define PWM00HLD        (*(unsigned char volatile xdata *)0xFF15)

sfr     ADC_CONTR   =   0xbc;
#define ADC_POWER       0x80
#define ADC_START       0x40
#define ADC_FLAG        0x20
#define ADC_EPWMT       0x10
sfr     ADC_RES     =   0xbd;
sfr     ADC_RESL    =   0xbe;

sbit    EADC        =   IE^5;

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;
```

```
void delay()
{
    int i;
    for (i=0; i<100; i++);
}

void main()
{
     P0M0 = 0x00;
     P0M1 = 0x00;
     P1M0 = 0x00;
     P1M1 = 0x00;
     P2M0 = 0x00;
     P2M1 = 0x00;
     P3M0 = 0x00;
     P3M1 = 0x00;
     P4M0 = 0x00;
     P4M1 = 0x00;
     P5M0 = 0x00;
     P5M1 = 0x00;

    ADC_CONTR = ADC_POWER | ADC_EPWMT | 0;       // Select P1.0 as the ADC input channel
    delay();                    // Wait for the ADC power supply to stabilize
    EADC = 1;

     PWMSET = 0x01;                                     //Enable PWM0 module (The configuration is effective only after the module is enabled.)

     P_SW2 = 0x80;
     PWM0CKS = 0x00;                                    // The clock of PWM0 is the system clock
      PWM0C = 0x1000;    // Set the period of PWM0 to 1000H PWM clocks
    PWM00T1= 0x0100;    // When the count value is 100H, the PWM00 channel outputs low level
    PWM00T2= 0x0500;    // When the count value is 500H, the PWM00 channel outputs high level
    PWM0TADC = 0x0200;       // Set ADC trigger point
     PWM00CR= 0x80;                                     //Enable PWM00 output
     P_SW2 = 0x00;

     PWMCFG01 = 0x07;       // Start the PWM0 module and enable the PWM0 interrupt and ADC trigger
     EA = 1;

     while (1);
}

void pwm0_isr() interrupt 22
{
     if (PWMCFG01 & 0x08)
     {
          PWMCFG01 &= ~0x08;
     }
}

void ADC_ISR() interrupt 5
{
     ADC_CONTR &= ~ADC_FLAG;
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

PWMSET      DATA    0F1H
PWMCFG01    DATA    0F6H
PWMCFG23    DATA    0F7H
PWMCFG45    DATA    0FEH

PWM0CH      EQU     0FF00H
PWM0CL      EQU     0FF01H
PWM0CKS     EQU     0FF02H
PWM0TADCH   EQU     0FF03H
PWM0TADCL   EQU     0FF04H
PWM0IF      EQU     0FF05H
PWM0FDCR    EQU     0FF06H
PWM00T1H    EQU     0FF10H
PWM00T1L    EQU     0FF11H
PWM00T2H    EQU     0FF12H
PWM00T2L    EQU     0FF13H
PWM00CR     EQU     0FF14H
PWM00HLD    EQU     0FF15H

ADC_CONTR   DATA    0BCH
ADC_POWER   EQU     080H
ADC_START   EQU     040H
ADC_FLAG    EQU     020H
ADC_EPWMT   EQU     010H
ADC_RES     DATA    0BDH
ADC_RESL    DATA    0BEH

EADC        BIT     IE.5

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     002BH
            LJMP    ADCISR
            ORG     00B3H
            LJMP    PWM0ISR

            ORG     0100H
ADCISR:
            ANL     ADC_CONTR,#NOT ADC_FLAG
            RETI

PWM0ISR:
```

```
                PUSH    ACC
                MOV     A,PWMCFG01
                JNB     ACC.3,ISREXIT
                ANL     PWMCFG01,#NOT 08H
ISREXIT:
                POP     ACC
                RETI

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                MOV     ADC_CONTR,# ADC_POWER | ADC_EPWMT
                SETB    EADC

                MOV     PWMSET,#01H             ;Enable PWM0 module (The configuration is effective only after the module is enabled.)

                MOV     P_SW2,#80H
                CLR     A
                MOV     DPTR,#PWM0CKS
                MOVX    @DPTR,A                 ; The clock of PWM0 is the system clock
                MOV     A,#10H
                MOV     DPTR,#PWM0CH            ; Set the period of PWM0 to 1000H PWM clocks
                MOVX    @DPTR,A
                MOV     A,#00H
                MOV     DPTR,#PWM0CL
                MOVX    @DPTR,A
                MOV     A,#01H
                MOV     DPTR,#PWM00T1H          ; When the count value is 100H, the PWM00 channel outputs low level
                MOVX    @DPTR,A
                MOV     A,#00H
                MOV     DPTR,#PWM00T1L
                MOVX    @DPTR,A
                MOV     A,#05H
                MOV     DPTR,#PWM00T2H          ; When the count value is 500H, the PWM00 channel outputs high level
                MOVX    @DPTR,A
                MOV     A,#00H
                MOV     DPTR,#PWM00T2L
                MOVX    @DPTR,A
                MOV     A,#02H
                MOV     DPTR,#PWM0TADCH         ; Set ADC trigger point
                MOVX    @DPTR,A
                MOV     A,#00H
                MOV     DPTR,#PWM0TADCL
                MOVX    @DPTR,A
```

```
                MOV         A,#80H
                MOV         DPTR,#PWM00CR           ; Enable PWM00 output
                MOVX        @DPTR,A
                MOV         P_SW2,#00H

                MOV         PWMCFG01,#07H           ; Start the PWM0 module and enable the PWM0 interrupt
and ADC trigger
                SETB        EA
                JMP         $

                END
```

18.2.5 Generate 3 complementary PWM waveforms with dead-time with a phase difference of 120 degrees

C language code

```
//Operating frequency for test is 24MHz


#include "reg51.h"

#define   MAIN_Fosc       24000000L               // Define the master clock

sbit      P20       =     P2^0;
sbit      P21       =     P2^1;
sbit      P22       =     P2^2;
sbit      P23       =     P2^3;
sbit      P24       =     P2^4;
sbit      P25       =     P2^5;

sfr       P2M1      =     0x95;
sfr       P2M0      =     0x96;
sfr       P_SW2     =     0xBA;                   // Peripheral port switch register 2
sfr       PWMSET    =     0xF1;                   // Enhanced PWM global configuration register
sfr       PWMCFG01  =     0xF6;                   // Enhanced PWM configuration register
sfr       PWMCFG23  =     0xF7;                   // Enhanced PWM configuration register

#define   PWM0C             (*(unsigned int  volatile xdata *)0xFF00)

#define   PWM2CKS           (*(unsigned char volatile xdata *)0xffa2)
#define   PWM2C             (*(unsigned int  volatile xdata *)0xffa0)
#define   PWM20T1           (*(unsigned int  volatile xdata *)0xffb0)
#define   PWM20T2           (*(unsigned int  volatile xdata *)0xffb2)
#define   PWM20CR           (*(unsigned char volatile xdata *)0xffb4)
#define   PWM20HLD          (*(unsigned char volatile xdata *)0xffb5)
#define   PWM21T1           (*(unsigned int  volatile xdata *)0xffb8)
#define   PWM21T2           (*(unsigned int  volatile xdata *)0xffba)
#define   PWM21CR           (*(unsigned char volatile xdata *)0xffbc)
#define   PWM21HLD          (*(unsigned char volatile xdata *)0xffbd)
#define   PWM22T1           (*(unsigned int  volatile xdata *)0xffc0)
#define   PWM22T2           (*(unsigned int  volatile xdata *)0xffc2)
#define   PWM22CR           (*(unsigned char volatile xdata *)0xffc4)
#define   PWM22HLD          (*(unsigned char volatile xdata *)0xffc5)
```

```
#define   PWM23T1          (*(unsigned int  volatile xdata *)0xffc8)
#define   PWM23T2          (*(unsigned int  volatile xdata *)0xffca)
#define   PWM23CR          (*(unsigned char volatile xdata *)0xffcc)
#define   PWM23HLD         (*(unsigned char volatile xdata *)0xffcd)
#define   PWM24T1          (*(unsigned int  volatile xdata *)0xffd0)
#define   PWM24T2          (*(unsigned int  volatile xdata *)0xffd2)
#define   PWM24CR          (*(unsigned char volatile xdata *)0xffd4)
#define   PWM24HLD         (*(unsigned char volatile xdata *)0xffd5)
#define   PWM25T1          (*(unsigned int  volatile xdata *)0xffd8)
#define   PWM25T2          (*(unsigned int  volatile xdata *)0xffda)
#define   PWM25CR          (*(unsigned char volatile xdata *)0xffdc)
#define   PWM25HLD         (*(unsigned char volatile xdata *)0xffdd)

#define   P2n_push_pull(bitn)  P2M1 &= ~(bitn), P2M0 |=  (bitn)

#define   ENO              0x80          /* 1: the corresponding pin is PWM, 0: the corresponding pin is GPIO */
#define   CKS_1T           0
#define   PWM_Normal       0x00          /* PWM output normally */
#define   ENPWM2           0x04          /* 1: Enable PWM2(PWM20~PWM27), 0: Disable PWM2 */
#define   PWM2CEN          0x01          /* 1: PWM2 starts coutning, 0:  stops counting */
#define   EAXSFR()         P_SW2 |=  0x80
#define   EAXRAM()         P_SW2 &= ~0x80

/*************Function Description   ***************
This program is suitable for STC8G2K64S4 series STC8G2K64S2 series.
P2.0+P2.1 P2.2+P2.3 P2.4+P2.5 output 3 complementary PWMs with dead time with a phase difference of 120 degrees, the PWM frequency is 50KHz, and the dead zone time is 0.5us.
*****************************************/

void PWM2_config(void);

void main(void)
{
    PWM2_config();
    while (1);
}

//========================================================================
// Function: void PWM2_config(void)
// Description: PWM configure function
// Parameters: none.
// Return: none.
// Version: VER1.0
// Date: 2020-5-17
// Note:
//========================================================================
void PWM2_config(void)
{
    EAXSFR();                       // Access the macros in the XFR. header file.
    PWMCFG23 &= 0xf0;
    PWMSET |= ENPWM2;               //Enable P2(P2.0~P2.7) as PWM
    PWM2CKS = CKS_1T;               //Select PWM2 clock, CKS_TIMER2, CKS_1T ~ CKS_16T
    PWM2C   = 480;                  //Set PWM2 period  = PWM2C + 1

    PWM20T2  = 12;                  // T2 output high level moment
    PWM20T1  = 160;//800;           //T1 output low level moment
    PWM20HLD = PWM_Normal;          //PWM output normally
```

```
                                        //(PWM_KeepHigh: PWM outputs high level forcibly,
                                        //PWM_KeepLow: PWM outputs low level forcibly)
    PWM20CR  = ENO;                     //ENO: enable PWM output
    P20 = 0;
    P2n_push_pull(1<<0);                // PWM output port is set to push-pull output

    PWM21T2  = 172;//812;               //T2 output high level moment
    PWM21T1  = 0;                       //T1 output low level moment
    PWM21HLD = PWM_Normal;              //PWM output normally
                                        //(PWM_KeepHigh: PWM outputs high level forcibly,
                                        //PWM_KeepLow: PWM outputs low level forcibly)
    PWM21CR  = ENO;                     //ENO: enable PWM output
    P21 = 0;
    P2n_push_pull(1<<1);                // PWM output port is set to push-pull output

    PWM22T2  = 172;//812;               // T2 output high level moment
    PWM22T1  = 320;//1600;              // T1 output low level moment
    PWM22HLD = PWM_Normal;              // PWM output normally,
                                        //(PWM_KeepHigh: PWM outputs high level forcibly,
                                        //PWM_KeepLow: PWM outputs low level forcibly)
    PWM22CR  = ENO;                     //ENO: enable PWM output
    P22 = 0;
    P2n_push_pull(1<<2);                // PWM output port is set to push-pull output

    PWM23T2  = 332;//1612;              //T2 output high level moment
    PWM23T1  = 160;//800;               //T1 output low level moment
    PWM23HLD = PWM_Normal;              //PWM output normally
                                        //(PWM_KeepHigh:PWM outputs high level forcibly,
                                        //PWM_KeepLow: PWM outputs low level forcibly)
    PWM23CR  = ENO;                     //ENO: enable PWM output
    P23 = 0;
    P2n_push_pull(1<<3);                //PWM output port is set to push-pull output

    PWM24T2  = 332;//1612;              //T2 output high level moment
    PWM24T1  = 480;//2400;              //T1 output low level moment
    PWM24HLD = PWM_Normal;              //PWM output normally,
                                        //(PWM_KeepHigh:PWM outputs high level forcibly,
                                        //PWM_KeepLow: PWM outputs low level forcibly)
    PWM24CR  = ENO;                     //ENO: enable PWM output
    P24 = 0;
    P2n_push_pull(1<<4);                //PWM output port is set to push-pull output

    PWM25T2  = 12;                      //T2 output high level moment
    PWM25T1  = 320;//1600;              //T1 output low level moment
    PWM25HLD = PWM_Normal;              //PWM output normally,
                                        //(PWM_KeepHigh:PWM outputs high level forcibly,
                                        //PWM_KeepLow: PWM outputs low level forcibly)
    PWM25CR  = ENO;                     //ENO: enable PWM output
    P25 = 0;
    P2n_push_pull(1<<5);                //PWM output port is set to push-pull output

    PWMCFG23 |= PWM2CEN;                //Starts counter, PWM output begins, initialize the last
executed statement
}

/********************** PWM fail interrupt function ************************/
void PWMFD_int (void) interrupt 23
{
```

```
}
```

18.2.6 Method of outputting a PWM waveform with a duty cycle of 100% (fixed output high) and 0% (fixed output low) (take PWM00 as an example)

Method 1: Disable PWM output

After disabling the PWM output, the corresponding IO is called ordinary IO, and it needs to be set by itself if the IO output is high or low.

```
PWM00CR |= 0x80;                //ENO=1: Enable PWM output
delay_ms(5);                    //Output 5ms
P00 = 1;                        // Output low level continuously
PWM00CR &= ~0x80;               //ENO=0: Disable PWM output
delay_ms(5);                    //Close PWM, P0.0 outputs high for 5ms

PWM00CR |= 0x80;                //ENO=1: Enable PWM output
delay_ms(5);                    // Output 5ms
P00 = 0;                        // Output low level continuously
PWM00CR &= ~0x80;               //ENO=0: Disable PWM output
delay_ms(5);                    // Close PWM, P0.0 outputs high for 5ms
```

Method 2: PWM00T2 sets the output high level moment (generally sets it to 0)

PWM00T1 sets the output low level time, and PWM00T1-PWM00T2 is the output high level time.
If the value of PWM00T1 is set larger than the period value, the low level will not be output, and 100% duty cycle will be output.

Method 3: Direct use of PWMnHLD register (emphasis recommended)

Use the PWM channel level holding control register PWMnHLD to directly set the output high or low. This register is specifically used to set the continuous output high or low.

```
PWM00HLD = 0x00;                // PWM normal output
PWM00HLD = 0x01;                // PWM output low level continuously
PWM00HLD = 0x02;                // PWM output high level continuously
```

18.2.7 Enhanced PWM-adjustable frequency-pulse counting

C language code

```
//Operating frequency for test is 24MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;
sfr     P4          =   0xC0;
sfr     P5          =   0xC8;
```

```
sfr     P6          =   0xE8;
sfr     P7          =   0xF8;

sfr     PWMSET      =   0xF1;
sfr     PWMCFG01    =   0xF6;
sfr     PWMCFG23    =   0xF7;
sfr     PWMCFG45    =   0xFE;

#define PWM0C       (*(unsigned int volatile xdata *)0xFF00)
#define PWM0CH      (*(unsigned char volatile xdata *)0xFF00)
#define PWM0CL      (*(unsigned char volatile xdata *)0xFF01)
#define PWM0CKS     (*(unsigned char volatile xdata *)0xFF02)
#define PWM0TADC    (*(unsigned int volatile xdata *)0xFF03)
#define PWM0TADCH   (*(unsigned char volatile xdata *)0xFF03)
#define PWM0TADCL   (*(unsigned char volatile xdata *)0xFF04)
#define PWM0IF      (*(unsigned char volatile xdata *)0xFF05)
#define PWM0FDCR    (*(unsigned char volatile xdata *)0xFF06)
#define PWM00T1     (*(unsigned int volatile xdata *)0xFF10)
#define PWM00T1H    (*(unsigned char volatile xdata *)0xFF10)
#define PWM00T1L    (*(unsigned char volatile xdata *)0xFF11)
#define PWM00T2H    (*(unsigned char volatile xdata *)0xFF12)
#define PWM00T2     (*(unsigned int volatile xdata *)0xFF12)
#define PWM00T2L    (*(unsigned char volatile xdata *)0xFF13)
#define PWM00CR     (*(unsigned char volatile xdata *)0xFF14)
#define PWM00HLD    (*(unsigned char volatile xdata *)0xFF15)
#define PWM01CR     (*(unsigned char volatile xdata *)0xFF1C)
#define PWM02CR     (*(unsigned char volatile xdata *)0xFF24)
#define PWM01T1     (*(unsigned int volatile xdata *)0xFF18)
#define PWM01T2     (*(unsigned int volatile xdata *)0xFF1A)
#define PWM02T1     (*(unsigned int volatile xdata *)0xFF20)
#define PWM02T2     (*(unsigned int volatile xdata *)0xFF22)

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

#define MAIN_Fosc       11059200UL

unsigned char Counter;
unsigned int Period;
bit DirFlag;

void delay_ms(unsigned char ms);
void PeriodSet(unsigned int period);

void main()
{
    P0M0 = 0x00;
```

```
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Period = 0x1000;
    DirFlag = 0;

    PWMSET = 0x01;                  // Enable the PWM0 module (must enable the following
settings of the module to be effective)
    P_SW2 = 0x80;                   // Enable XFR access
    PWM0CKS = 0x00;                 // PWM0 clock is the system clock

    PWM00T1= 0x0000;                // When the count value is 00H, the PWM00 channel
outputs low level
    PWM00CR= 0x80;                  // Enable PWM00 output

    EA = 1;

    while (1)
    {
        delay_ms(10);
        PeriodSet(Period);          // Set period and duty
        PWMCFG01 = 0x05;            // Start the PWM0 module, enable the counter zero
interrupt

        if(DirFlag)
        {
            Period++;               // Cycle increment
            if(Period >= 0x1000)
            {
                DirFlag = 0;
            }
        }
        else
        {
            Period--;               // Cycle decreasement
            if(Period <= 0x0100)
            {
                DirFlag = 1;
            }
        }
    }
}

void pwm0_isr(void) interrupt 22
{
    if(PWMCFG01 & 0x08)             // Judgment counter overflow flag
    {
        Counter++;
        if(Counter >= 10)           // Turn off the PWM counter after counting 10 pulses
        {
```

```
                Counter = 0;
                PWMCFG01 = 0x00;
            }
            else
            {
                PWMCFG01 &= ~0x08;                  // Clear flag
            }
        }
}

//========================================================================
// Function: void PeriodSet(unsigned int period)
// Description: PWM period setting function.
// Parameters: period, The number of periods to set.
// Return: none.
// Version: VER1.0
// Date: 2021-08-23
// Note:
//========================================================================
void PeriodSet(unsigned int period)
{
    PWM0C = period;                                 // Set the period of PWM0 to period PWM clocks
    PWM00T2= (period>>1);                           // When the count value is Period/2, the PWM00 channel outputs high level
}

//========================================================================
// Function: void delay_ms(unsigned char ms)
// Description: delay function
// Parameters: ms, number of ms to delay, Here only support 1~255ms. Automatically adapt to the main clock.
// Return: none.
// Version: VER1.0
// Date: 2021-01-05
// Note:
//========================================================================
void delay_ms(unsigned char ms)
{
    unsigned int i;
    do{
        i = MAIN_Fosc / 10000;
        while(--i);
    }while(--ms);
}
```

STC MCU

19 Sysnchronous Serial Peripheral Interface (SPI)

Product line	SPI
STC8G1K08 family	●
STC8G1K08-8Pin family	●
STC8G1K08A family	●
STC8G2K64S4 family	●
STC8G2K64S2 family	●
STC8G1K08T family	●
STC15H2K64S4 family	●

A high-speed serial communication interface, SPI, is integrated in STC8G series of microcontrollers. SPI is a full-duplex high-speed synchronous communication bus. SPI interface integrated in the STC8G series of microcontrollers offers two operation modes: master mode and slave mode.

19.1 Registers Related to SPI

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
SPSTAT	SPI Status register	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
SPCTL	SPI Control Register	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]		0000,0100
SPDAT	SPI Data Register	CFH									0000,0000

19.1.1 SPI Status register (SPSTAT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	CDH	SPIF	WCOL	-	-	-	-	-	-

SPIF: SPI transfer completion flag.

When SPI completes sending / receiving 1 byte of data, the hardware will automatically set this bit and request interrupt to CPU. If the SSIG bit is set to 0, this flag will also be automatically set by hardware to indicate a mode change of device when the master / slave mode of the device changes due to changes in the SS pin level.

Note: This bit must be cleared using software writing 1 to it.

WCOL: SPI write collision flag bit.

This bit is set by hardware when the SPI is writing to the SPDAT register during data transfer.

Note: This bit must be cleared using software by writing 1 to it.

19.1.2 SPI Control register (SPCTL)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
SPCTL	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]	

SSIG: Control bit of whether SS pin is ignored or not.

0: the SS pin decides whether the device is a master or slave.

1: the function of SS pin is ignored. MSTR decides whether the device is a master or slave.

SPEN: SPI enable bit.

0: the SPI is disabled.

1: the SPI is enabled.

DORD: Set the transmitted or received SPI data order.

0: The MSB of the data is transmitted firstly.

1: The LSB of the data is transmitted firstly.

MSTR: Master/Slave mode select bit.

To set the mastert mode:

If SSIG = 0, the SS pin must be high and set MSTR to 1.

If SSIG = 1, it only needs to set MSTR to 1 (ignoring the SS pin level).

To set the slave mode:

If SSIG = 0, the SS pin must be low (regardless of the MSTR bit).

If SSIG = 1, it only needs to set MSTR to 0 (ignoring the SS pin level).

CPOL: SPI clock polarity select bit.

0：SCLK is low when idle. The leading edge of SCLK is the rising edge and the trailing edge is the falling edge.

1：SCLK is high when idle. The leading edge of SCLK is the falling edge and the trailing edge is the rising edge.

CPHA: SPI clock phase select bit.

0: The first bit of datum is driven when SS pin is low. The datum changes on the trailing edge of SCLK and is sampled on the leading edge of SCLK. (SSIG must be 0.)

1: The datum is driven on the leading edge of SCLK, and is sampled on the trailing edge.

SPR[1:0]: SPI clock frequency select bits

SPR[1:0]	SCLK frequency
00	SYSclk/4
01	SYSclk/8
10	SYSclk/16
11	SYSclk/32

19.1.3 SPI Data register (SPDAT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
SPDAT	CFH								

The SPDAT holds the data to be transmitted or the data received.

STCMCU

19.2 SPI Communication Modes

There are three SPI communication modes: single master and single slave mode, dual devices configuration mode (any one of them can be a master or slave), single master and multiple slaves mode.

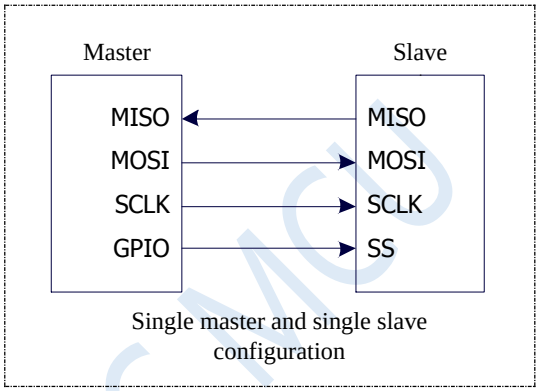
19.2.1 Single Master and Single Slave Mode

Two devices are connected, one of which is fixed as a master and the other as a slave.

Master settings: SSIG set to 1, MSTR set to 1, fixed to be master mode. The master can use any port to connect the slave SS pin, pull down the slave SS pin to enable the slave.

Slave settings: SSIG is set to 0, SS pin as the chip select signal of the slave.

Single master single slave connection configuration diagram is shown as follows:



Single master and single slave configuration

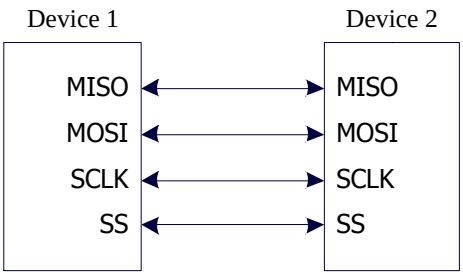
19.2.2 Dual Devices Configuration Mode

Two devices are connected, the master and the slave are not fixed.

Setting Method 1: Both devices are initialized with SSIG set to 0, MSTR set to 1, and SS pin set to bi-directional mode and output high. Now the both devices are in master mode with not ignoring SS. When one of the devices needs to initiate a transfer, set its own SS pin to output mode and output low to pull down the other device's SS pin so that the other device is forcibly set to slave mode.

Set Method 2: Both devices are initialized as slave mode with ignoring SS, where SIG is set to 1 and MSTR is set to 0. When one of the devices needs to initiate a transfer, detect the SS pin's level firstly. If SS is high, the device sets itself to master mode with ignoring SS, then starts the data transfer.

The connection configuration of dual devices configuration mode is shown as follows:



The connection configuration of dual devices configuration mode

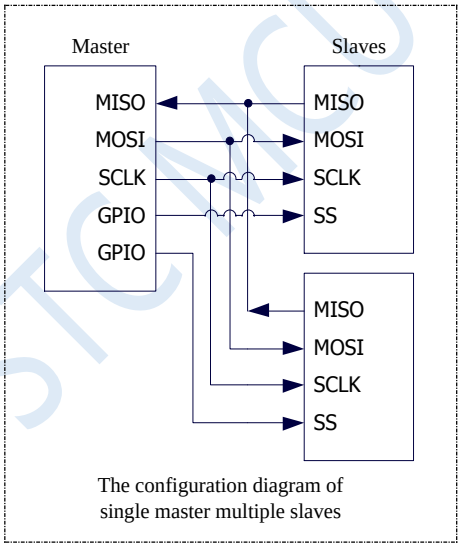
19.2.3 Single Master and Multiple Slaves Mode

Multiple devices are connected, one of which is fixed as a master and others are fixed as slaves.

Master settings: SSIG set to 1, MSTR set to 1, fixed to master mode. The master can use any port to connect with the SS pins of each slave respectively, and pull down the SS pin of one slave to enable the corresponding slave device.

Slave settings: SSIG is set to 0, SS pin is used as the chip select signal of the slave.

The configuration diagram of single master multiple slaves is as follows:



The configuration diagram of single master multiple slaves

STCMCU

19.3 Configure SPI

Control bits			Communication port pins				Descriptions
SPEN	SSIG	MSTR	SS	MISO	MOSI	SCLK	
0	x	x	x	input	input	input	SPI is disabled, SS/MOSI/MISO/SCLK are used as general I/O ports
1	0	0	0	output	input	input	**Selected as slave**
1	0	0	1	high impedance	input	input	**Selected as slave,** not selected.
1	0	1→0	0	output	input	input	**Slavce mode**, master mode with notignoring SS and MSTR is 1. When SS pin is pulled low, MSTR will be automatically cleared by hardware and the operating mode will be passively set to slave mode.
1	0	1	1	input	high impedance	high impedance	**Master mode,** idle state
					output	output	**Master mode, ative state**
1	1	0	x	output	input	input	**Slave mode**
1	1	1	x	input	output	output	**Master mode**

Additional Considerations for a Slave

When CPHA = 0, SSIG must be 0 (i.e. SS pin can not be ignored). The SS pin must be pulled low before each serial byte begins transfer and must be reset to high after the transfer completes. The SPDAT register can not be written while the SS pin is low, otherwise a write collision error will occur. Operation with CPHA = 0 and SSIG = 1 is undefined.

When CPHA = 1, SSIG may be set to 1 (i.e. the SS pin can be ignored). If SSIG = 0, the SS pin may remain active low (i.e., stay low all the way) for consecutive transfers. This method is suitable for fixed single master single slave system.

Additional Considerations for a Master

In SPI, transfers are always initiated by the master. If the SPI is enabled (SPEN = 1) and selected as the master, the master will initiate SPI clock generator and data transfer by writing to SPI data register, SPDAT. The data will appear on the MOSI pin a half to one SPI bit-time later after the data is written to SPDAT. The data written to the SPDAT register of the master is shifted out from the MOSI pin and sent to the MOSI pin of the slave. And, at the same time the data in SPDAT register of the selected slave is shifted out on MISO pin to the MISO pin of the master.

After one byte has been transmitted, the SPI clock generator is stopped, the transfer completion flag (SPIF) is set, and an SPI interrupt is generated if the SPI interrupt is enabled. The two shift registers for the master and slave CPUs can be considered as a 16-bit cyclic shift register. As data is shifted from the master to the slave, data is also shifted in the opposite direction simultaneously. This means that the data of the master and the slave are exchanged with each other in one shift period.

Mode is Changed by SS pin

If SPEN = 1, SSIG = 0 and MSTR = 1, SPI is enabled in master mode and the SS pin can be configured for input mode or quasi-bidirectional port mode. In this case, another master can drive this pin low to select the

device as an SPI slave and send data to it. To avoid bus contention, the SPI system clears the slave's MSTR, forces MOSI and SCLK to be input mode, and MISO changes to output mode. The SPIF flag in SPSTAT is set, and if the SPI interrupt is enabled, an SPI interrupt will occur.

The user software must always detect the MSTR bit. If this bit is cleared by a slave selection action and the user wants to continue using the SPI as a master, the MSTR bit must be set again, otherwise it will remain in slave mode.

Write Collision

The SPI is single buffered in the transmition process and double buffered in receiving process. New data for transmission can not be written to the shift register until the previous transmission is complete. The WCOL bit will be set to indicate that a data write collision error has occurred when the data register SPDAT is written during transmission. In this case, the data currently being transmitted will continue to be transmitted, and the newly written data will be lost.

A write collision condition on the master is rare when write collision detection is performed on a master or slave because the master has full control of the data transfer. However, a write collision may occur on the slave because the slave can not control it when the master initiates the transfer.

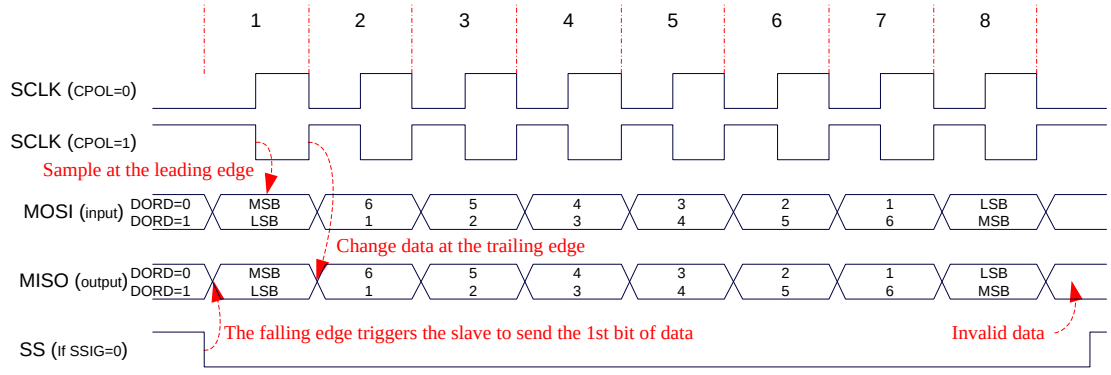
When receiving data, the received data is transferred to a parallel read data buffer, which will release the shift register for the next data reception. However, the received data must be read from the data register before the next character is completely shifted in. Otherwise, the previous received data will be lost.

WCOL can be cleared by software by writing "1" to it.

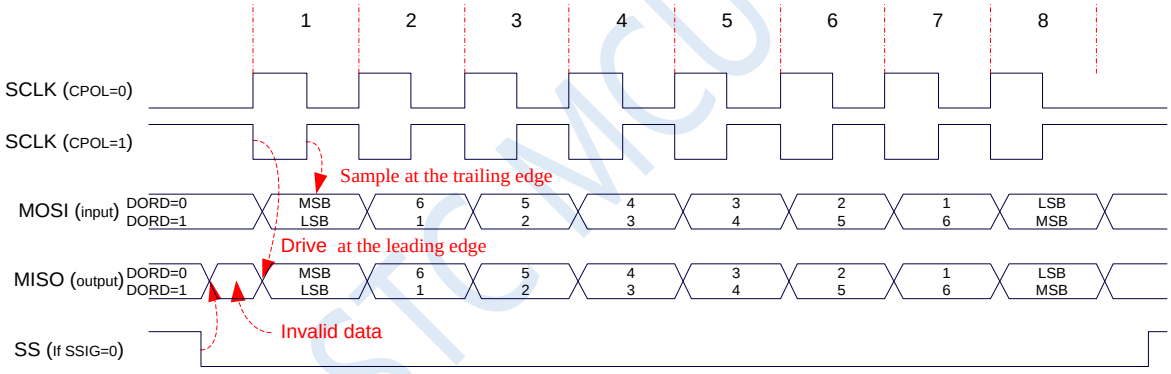
STC MCU

19.4 Data Format

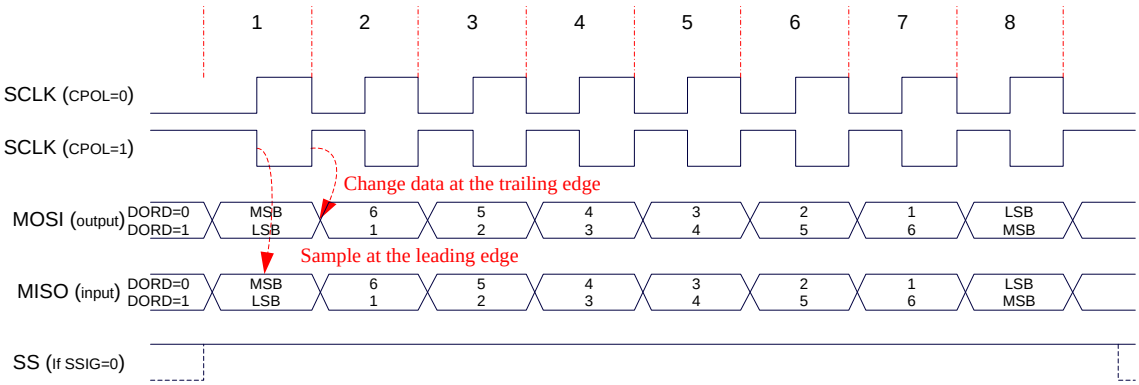
The clock phase control bit, CPHA, of the SPI allows the user to set the clock edge when the data is sampled and changed. The clock polarity bit, CPOL, allows the user to set the clock polarity. The following illustrations show the SPI communication timing under different clock phases and polarity settings.



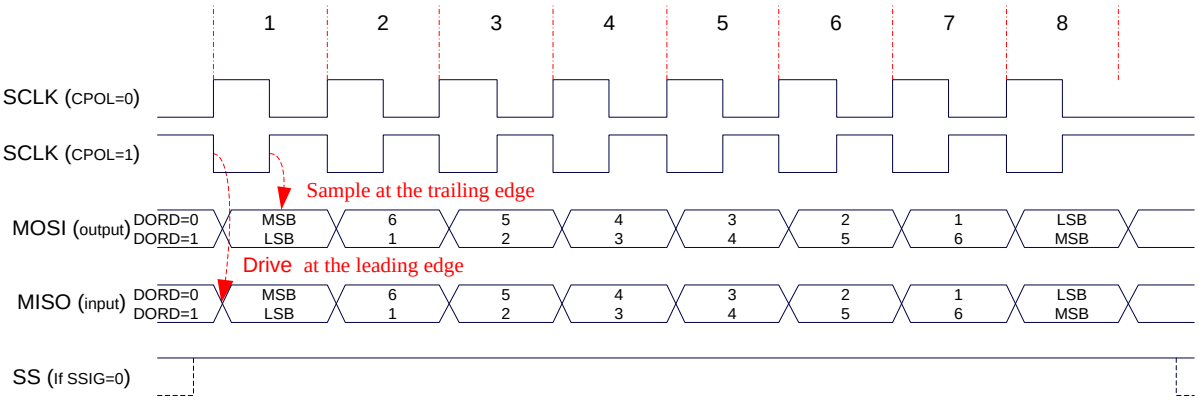
SPI slave transfer format with CPHA=0



SPI slave transfer format with CPHA=1



SPI master transfer format with CPHA=0



SPI master transfer format with CPHA=1

STC MCU

19.5 Example Routines

19.5.1 Master Routine of Single Master Single Slave Mode (Interrupt Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         SPSTAT      =   0xcd;
sfr         SPCTL       =   0xce;
sfr         SPDAT       =   0xcf;
sfr         IE2         =   0xaf;
#define     ESPI            0x02

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        SS          =   P1^0;
sbit        LED         =   P1^1;

bit         busy;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                  //Clear interrupt flag
    SS = 1;                         //Pull up the SS pin of the slave
    busy = 0;
    LED = !LED;                     //Test port
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
```

STC MCU

```
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    SS = 1;
    busy = 0;

    SPCTL = 0x50;                   //Enable SPI master mode
    SPSTAT = 0xc0;                  //Clear interrupt flag
    IE2 = ESPI;                     //Enable SPI interrupt
    EA = 1;

    while (1)
    {
        while (busy);
        busy = 1;
        SS = 0;                     //Pull down the slave SS pin
        SPDAT = 0x5a;               //Send test data
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

BUSY        BIT     20H.0
SS          BIT     P1.0
LED         BIT     P1.1

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            MOV     SPSTAT,#0C0H    ;Clear interrupt flag
            SETB    SS              ;Pull up the SS pin of the slave
```

```
                CLR     BUSY
                CPL     LED
                RETI

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                SETB    LED
                SETB    SS
                CLR     BUSY

                MOV     SPCTL,#50H      ;Enable SPI master mode
                MOV     SPSTAT,#0C0H    ;Clear interrupt flag
                MOV     IE2,#ESPI       ;Enable SPI interrupt
                SETB    EA

LOOP:
                JB      BUSY,$
                SETB    BUSY
                CLR     SS              ;Pull down the slave SS pin
                MOV     SPDAT,#5AH      ;Send test data
                JMP     LOOP

                END
```

19.5.2 Slave Routine of Single Master Single Slave Mode (Interrupt Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     SPSTAT  =   0xcd;
sfr     SPCTL   =   0xce;
sfr     SPDAT   =   0xcf;
sfr     IE2     =   0xaf;
#define ESPI            0x02

sfr     P0M1    =   0x93;
```

```
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    LED     =   P1^1;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                  //Clear interrupt flag
    SPDAT = SPDAT;                  //Transmit the received data back to the master
    LED = !LED;                     //Test port
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    SPCTL = 0x40;                   //Enable SPI slave mode
    SPSTAT = 0xc0;                  //Clear interrupt flag
    IE2 = ESPI;                     //Enable SPI interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

LED         BIT     P1.1

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
```

```
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            MOV     SPSTAT,#0C0H        ;Clear interrupt flag
            MOV     SPDAT,SPDAT         ;Transmit the received data back to the master
            CPL     LED
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     SPCTL,#40H          ;Enable SPI slave mode
            MOV     SPSTAT,#0C0H        ;Clear interrupt flag
            MOV     IE2,#ESPI           ;Enable SPI interrupt
            SETB    EA

            JMP     $

            END
```

19.5.3 Master Routine of Single Master Single Slave Mode (Polling Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
```

```
#include "intrins.h"

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sfr     SPSTAT  =   0xcd;
sfr     SPCTL   =   0xce;
sfr     SPDAT   =   0xcf;
sfr     IE2     =   0xaf;
#define ESPI        0x02

sbit    SS      =   P1^0;
sbit    LED     =   P1^1;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    SS = 1;

    SPCTL = 0x50;                   //Enable SPI master mode
    SPSTAT = 0xc0;                  //Clear interrupt flag

    while (1)
    {
        SS = 0;                     //Pull down the slave SS pin
        SPDAT = 0x5a;               //Send test data
        while (!(SPSTAT & 0x80));   //Query completion flag
        SPSTAT = 0xc0;              //Clear interrupt flag
        SS = 1;                     //Pull up the SS pin of the slave
        LED = !LED;                 //Test port
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz
```

```
SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

SS          BIT     P1.0
LED         BIT     P1.1

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            SETB    LED
            SETB    SS

            MOV     SPCTL,#50H          ;Enable SPI master mode
            MOV     SPSTAT,#0C0H        ;Clear interrupt flag

LOOP:
            CLR     SS                  ;Pull down the slave SS pin
            MOV     SPDAT,#5AH          ;Send test data
            MOV     A,SPSTAT            ;Query completion flag
            JNB     ACC.7,$-2
            MOV     SPSTAT,#0C0H        ;Clear interrupt flag
            SETB    SS
            CPL     LED
            JMP     LOOP

            END
```

STC MCU

19.5.4 Slave Routine of Single Master Single Slave Mode (Polling Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         SPSTAT      =   0xcd;
sfr         SPCTL       =   0xce;
sfr         SPDAT       =   0xcf;
sfr         IE2         =   0xaf;
#define     ESPI            0x02

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        LED         =   P1^1;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                          //Clear interrupt flag
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    SPCTL = 0x40;                           //Enable SPI slave mode
    SPSTAT = 0xc0;                          //Clear interrupt flag
```

STC MCU

```
    while (1)
    {
        while (!(SPSTAT & 0x80));                  //Query completion flag
        SPSTAT = 0xc0;                             //Clear interrupt flag
        SPDAT = SPDAT;                             //Transmit the received data back to the master
        LED = !LED;                                //Test port
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

LED         BIT     P1.1

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     SPCTL,#40H          ;Enable SPI slave mode
            MOV     SPSTAT,#0C0H        ;Clear interrupt flag

LOOP:
            MOV     A,SPSTAT            ;Query completion flag
            JNB     ACC.7,$-2
```

STC MCU

```
        MOV     SPSTAT,#0C0H        ;Clear interrupt flag
        MOV     SPDAT,SPDAT         ;Transmit the received data back to the master
        CPL     LED
        JMP     LOOP

        END
```

19.5.5 Routine of Mutual Master-Slave Mode (Interrupt Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     SPSTAT  =   0xcd;
sfr     SPCTL   =   0xce;
sfr     SPDAT   =   0xcf;
sfr     IE2     =   0xaf;
#define ESPI            0x02

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    SS      =   P1^0;
sbit    LED     =   P1^1;
sbit    KEY     =   P0^0;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                  //Clear interrupt flag
    if (SPCTL & 0x10)
    {                               //Master mode
        SS = 1;                     //Pull up the SS pin of the slave
        SPCTL = 0x40;               //Reset to slave and standby
    }
    else
    {                               //Slave mode
        SPDAT = SPDAT;              //Transmit the received data back to the master
    }
    LED = !LED;                     //Test port
}

void main()
{
```

```
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    KEY = 1;
    SS = 1;

    SPCTL = 0x40;                   //Enable SPI slave modeand standby
    SPSTAT = 0xc0;                  //Clear interrupt flag
    IE2 = ESPI;                     //Enable SPI interrupt
    EA = 1;

    while (1)
    {
        if (!KEY)                   //Wait for the key to trigger
        {
            SPCTL = 0x50;           //Enable SPI master mode
            SS = 0;                 //Pull down the slave SS pin
            SPDAT = 0x5a;           //Send test data
            while (!KEY);           //Wait for the keys to be released
        }
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

SS          BIT     P1.0
LED         BIT     P1.1
KEY         BIT     P0.0

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
```

```
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            PUSH    ACC
            MOV     SPSTAT,#0C0H        ;Clear interrupt flag
            MOV     A,SPCTL
            JB      ACC.4,MASTER
SLAVE:
            MOV     SPDAT,SPDAT         ;Transmit the received data back to the master
            JMP     ISREXIT
MASTER:
            SETB    SS                  ;Pull up the SS pin of the slave
            MOV     SPCTL,#40H          ;Reset to slave and standby
ISREXIT:
            CPL     LED
            POP     ACC
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            SETB    SS
            SETB    LED
            SETB    KEY

            MOV     SPCTL,#40H          ;Enable SPI slave mode and standby
            MOV     SPSTAT,#0C0H        ;Clear interrupt flag
            MOV     IE2,#ESPI           ;Enable SPI interrupt
            SETB    EA

LOOP:
            JB      KEY,LOOP            ;Wait for the key to trigger
            MOV     SPCTL,#50H          ;Enable SPI master mode
            CLR     SS                  ;Pull down the slave SS pin
            MOV     SPDAT,#5AH          ;Send test data
            JNB     KEY,$               ;Wait for the keys to be released
            JMP     LOOP

            END
```

STC MCU

19.5.6 Routine of Mutual Master-Slave Mode (Polling Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     SPSTAT  =   0xcd;
sfr     SPCTL   =   0xce;
sfr     SPDAT   =   0xcf;
sfr     IE2     =   0xaf;
#define ESPI            0x02

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    SS      =   P1^0;
sbit    LED     =   P1^1;
sbit    KEY     =   P0^0;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    KEY = 1;
    SS = 1;

    SPCTL = 0x40;                   //Enable SPI slave mode and standby
    SPSTAT = 0xc0;                  //Clear interrupt flag

    while (1)
    {
```

STC MCU

```
        if (!KEY)                                   //Wait for the key to trigger
        {
            SPCTL = 0x50;                           //Enable SPI master mode
            SS = 0;                                 //Pull down the slave SS pin
            SPDAT = 0x5a;                           //Send test data
            while (!KEY);                           //Wait for the keys to be released
        }
        if (SPSTAT & 0x80)
        {
            SPSTAT = 0xc0;                          //Clear interrupt flag
            if (SPCTL & 0x10)
            {                                       //Master mode
                SS = 1;                             //Pull up the SS pin of the slave
                SPCTL = 0x40;                       //Reset to slave and standby
            }
            else
            {                                       //Slave mode
                SPDAT = SPDAT;                      //Transmit the received data back to the master
            }
            LED = !LED;                             //Test port
        }
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

SS          BIT     P1.0
LED         BIT     P1.1
KEY         BIT     P0.0

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
```

```
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        SETB    SS
        SETB    LED
        SETB    KEY

        MOV     SPCTL,#40H          ;Enable SPI slave mode and standby
        MOV     SPSTAT,#0C0H        ;Clear interrupt flag

LOOP:
        JB      KEY,SKIP            ;Wait for the key to trigger
        MOV     SPCTL,#50H          ;Enable SPI master mode
        CLR     SS                  ;Pull down the slave SS pin
        MOV     SPDAT,#5AH          ;Send test data
        JNB     KEY,$               ;Wait for the keys to be released
SKIP:
        MOV     A,SPSTAT
        JNB     ACC.7,LOOP
        MOV     SPSTAT,#0C0H        ;Clear interrupt flag
        MOV     A,SPCTL
        JB      ACC.4,MASTER
SLAVE:
        MOV     SPDAT,SPDAT         ;Transmit the received data back to the master
        CPL     LED
        JMP     LOOP
MASTER:
        SETB    SS                  ;Pull up the SS pin of the slave
        MOV     SPCTL,#40H          ;Reset to slave and standby
        CPL     LED
        JMP     LOOP

        END
```

STC MCU

20 I²C Bus

Product line	I2C
STC8G1K08 family	●
STC8G1K08-8Pin family	●
STC8G1K08A family	●
STC8G2K64S4 family	●
STC8G2K64S2 family	●
STC8G1K08T family	●
STC15H2K64S4 family	●

An I²C serial bus controller is integrated in the STC8G series of microcontrollers. I²C is a high-speed synchronous communication bus, which uses SCL (clock line) and SDA (data line) to carry out two-wire synchronous communication. For the port allocation of SCL and SDA, STC8G series of microcontrollers provide pin switch mode that can switch SCL and SDA to different I/O ports. Therefor, it is convenience to use a set of I²C as multiple sets of I²C buses through time sharing.

Compared with the standard I²C protocol, the following two mechanisms are ignored:

- No arbitration will be performed after the start signal (START) is sent.
- No timeout detection when the clock signal (SCL) stays at low level.

The I²C bus of the STC8G series of microcontrollers offer two modes of operation: master mode (SCL is the output port, which is used to transmit synchronous clock signal) and slave mode (SCL is the input port, which is used to receive the synchronous clock signal).

STC innovation: When the I²C serial bus controller of STC works in slave mode, the falling edge signal of SDA pin can wake up the MCU which is in power-down mode. (Note: Due to the fast I²C transmission speed, the first packet of data after the MCU wakes up is generally incorrect.)

20.1 Registers Related to I²C

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
I2CCFG	I2C Configuration Register	FE80H	ENI2C	MSSL	MSSPEED[6:1]						0000,0000
I2CMSCR	I²C Master Control Register	FE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I²C Master Status Register	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I²C Slave Control Register	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I²C Slave Status Register	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I²C Slave Address Register	FE85H	SLADR[6:0]							MA	0000,0000
I2CTXD	I²C Data Transmission Register	FE86H									0000,0000
I2CRXD	I²C Data Receive Register	FE87H									0000,0000
I2CMSAUX	I²C Master Auxiliary Control Register	FE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0

20.2 I²C Master Mode

20.2.1 I²C Configuration Register (I2CCFG)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CCFG	FE80H	ENI2C	MSSL	MSSPEED[5:0]					

ENI2C: I²C function enable bit

0: disable I²C function

1: enable I²Cfunction

MSSL: I²C mode selection bit

0: Salve mode

1: Master mode

MSSPEED[5:0]: I²C bus speed control bits (clocks to wait), **I²C bus speed＝F_{OSC} / 2 / (MSSPEED * 2 + 4)**

MSSPEED[5:0]	Corresponding clocks
0	1
1	3
2	5
…	…
x	2x+1
…	…
62	125
63	127

The waiting parameter set by the MSSPEED is valid only when the I²C module is operating in the master mode. The waiting parameter is mainly used for the following signals in master mode.

T_{SSTA}: Setup Time of START

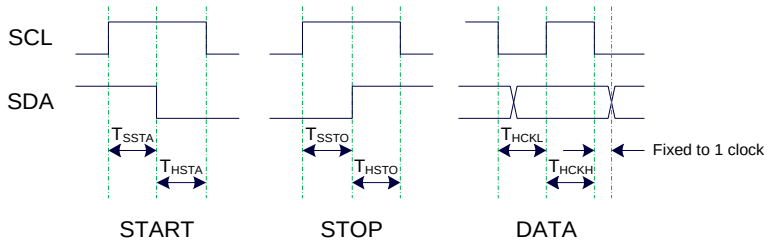
T_{HSTA}: Hold Time of START

T_{SSTO}: Setup Time of STOP

T_{HSTO}: Hold Time of STOP

$T_{HCKL:}$ Hold Time of SCL Low

$T_{HCKH:}$ Hold Time of SCL High



Example 1: When MSSPEED=10, TSSTA=THSTA=TSSTO=THSTO=THCKL=24/FOSC

Example 2: When an I2C bus speed of 400K is required at a working frequency of 24MHz,

MSSPEED = (24M / 400K / 2-4) / 2 = 13

Note:

Due to the need to cooperate with the clock synchronization mechanism, the high-level hold time (THCKH) of the clock signal should be at least twice as long as the low-level hold time (THCKL) of the clock signal, and the exact length of THCKH depends on the pull-up speed of the SCL port.

The data hold time of SDA is fixed as 1 clock after the falling edge of SCL.

20.2.2 I²C Master Control Register (I2CMSCR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	FE81H	EMSI	-	-	-	MSCMD[3:0]			

EMSI: Master mode interrupt enable control bit

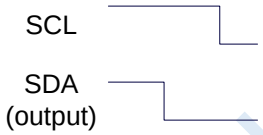
0: disable master mode interrupt

1: enable master mode interrupt

MSCMD[3:0]: master command bits

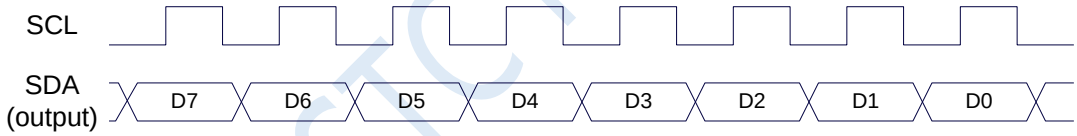
0000: Standby, no action

0001: START command. Send a START signal. If the I²C controller is in idle state currently, i.e. MSBUSY (I2CMSST.7) is 0, writing this command will make the controller enter the busy status, and the hardware will set the MSBUSY status bit automatically and start sending START signal. If the I²C controller is busy currently, writing this command will triger to send the START signal. Sending the START signal waveform is shown below:



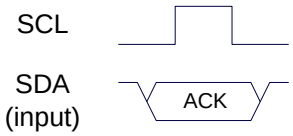
0010: Send data command.

After writing this command, the I²C bus controller will generate 8 clocks on the SCL pin and send the data in the I2CTXD register bit by bit to the SDA pin (send MSB firstly). The waveform of the transmitting data is shown in the following figure:



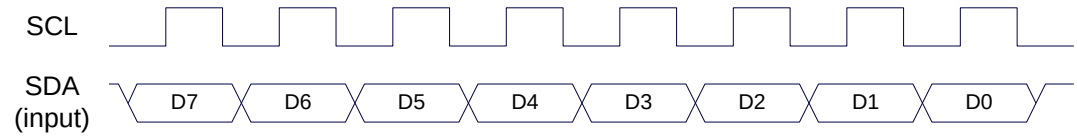
0011: Receive ACK command.

After writing this command, the I²C bus controller will generate a clock on the SCL pin and save the data bit read from SDA to MSACKI (I2CMSST.1). The waveform of the receiving ACK is shown below:



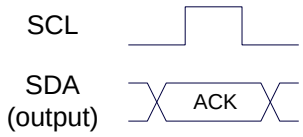
0100: Receive data command.

After writing this command, the I²C bus controller will generate 8 clocks on the SCL pin, and sequentially shift the data bit read from SDA to the I2CRXD register (receiving MSB firstly). The waveform of the receiving data is as shown in the figure below:



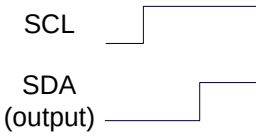
0101: Send ACK command.

After writing this command, the I²C bus controller will generate a clock on the SCL pin and send the data bit in MSACKO (I2CMSST.0) to SDA. The waveform of sending ACK is shown below:



0110: Send STOP signal.

After writing this command, the I²C bus controller starts sending STOP signal. After the signal is sent, the hardware clears the MSBUSY status bit automatically. The waveform of STOP signal is shown below:



0111: Reserved.

1000: Reserved.

1001: Start command + send data command + receive ACK command.

This command is a combination of command 0001, command 0010 and command 0011. After wrting this command, the controller will execute these three commands in sequence.

1010: Send data command + receive ACK command.

This command is a combination of command 0010 and command 0011. After writing this command, the controller will execute these two commands in sequence.

1011: Receive data command + send ACK (0) command.

This command is a combination of command 0100 and command 0101. After writing this command, the controller will execute these two commands in sequence.

Note: The response signal returned by this command is fixed as ACK (0) and is not affected by the MSACKO bit.

1100: Receive data command + send NAK (1) command.

This command is a combination of command 0100 and command 0101. After writing this command, the controller will execute these two commands in sequence.

Note: The response signal returned by this command is fixed to NAK (1), and is not affected by the MSACKO bit.

20.2.3 I²C Master Auxiliary Control Register (I2CMSAUX)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSAUX	FE88H	-	-	-	-	-	-	-	WDTA

WDTA: I2C data automatic transmission enable bit in master mode.

0: disable automatic transmission

1: enable automatic transmission

If the automatic transmission function is enabled, when the MCU finishes writing to the I2CTXD data register, the I²C controller will trigger the "1010" command automatically, that is, it will send data automatically and receive the ACK signal.

20.2.4 I²C Master Status Register (I2CMSST)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSST	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO

MSBUSY: status bit of I²C controller in master mode. (Read-only)

0: the controller is in idle state.

1: the controller is in busy state.

When the I²C controller is in master mode, the controller will enter the busy state after sending the START signal in the idle state. The busy state will be maintained until the STOP signal is successfully transmitted, and the state will return to the idle.

MSIF: master mode interrupt request bit (interrupt flag bit). When the I²C controller in the master mode executes the MSCMD command in the completion register I2CMSCR, it generates an interrupt signal. This bit is set to 1 by hardware automatically to request an interrupt to CPU. The MSIF bit must be cleared by software after responding to the interrupt.

MSACKI: In master mode, it is the ACK datum received after sending the "011" command to the MSCMD bit in I2CMSCR.

MSACKO: In master mode, it is the ACK signal ready to be transmitted. When the "101" command is sent to the MSCMD bit of I2CMSCR, the controller will read the datum of this bit automatically and send it as ACK to SDA.

STC MCU

20.3 I²C Slave Mode

20.3.1 I²C Slave Control Register (I2CSLCR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLCR	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST

ESTAI: interrupt enable bit when receiving START signal in slave mode.

0: disable interrupt when receiving START signal in slave mode.

1: enable interrupt when receiving START signal in slave mode.

ERXI: interrupt enable bit after 1 byte datum is received in slave mode

0: disable interrupt after a datum is received in slave mode.

1: enable interrupt after 1 byte datum is reveived in slave mode.

ETXI: interrupt enable bit after 1 byte datum is sent in slave mode

0: disable interrupt after a datum is sent in slave mode.

1: enable interrupt after 1 byte datum is sent in slave mode.

ESTOI: interrupt enable bit after STOP signal is received in slave mode.

0: disable interrupt after STOP signal is received in slave mode.

1: enable interrupt after STOP signal is received in slave mode.

SLRST: reset slave mode

20.3.2 I²C Slave Status Register (I2CSLST)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLST	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	-	SLACKI	SLACKO

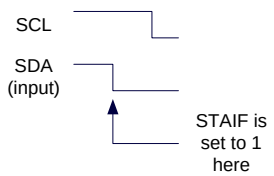
SLBUSY: status bit of I²C controller in slave mode. (Read-only)

0: the controller is in idle state.

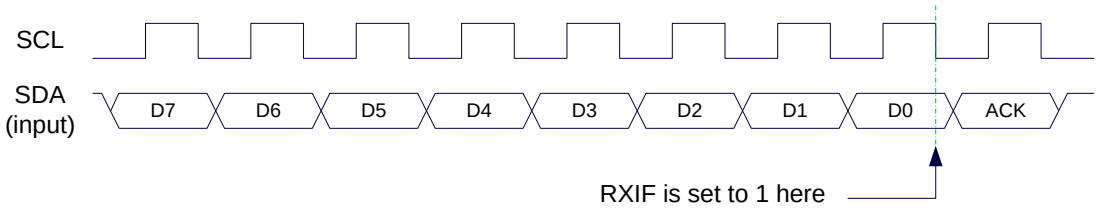
1: the controller is in busy state.

When the I²C controller is in slave mode, the controller will continue to detect the subsequent device address data when it receives the START signal from the master in idle state. If the device address matches the slave address set in the current I2CSLADR register, the controller will enter the busy state. And the busy state will be maintained until receives a STOP signal sent by the master successfully, and then the state will return to idle.

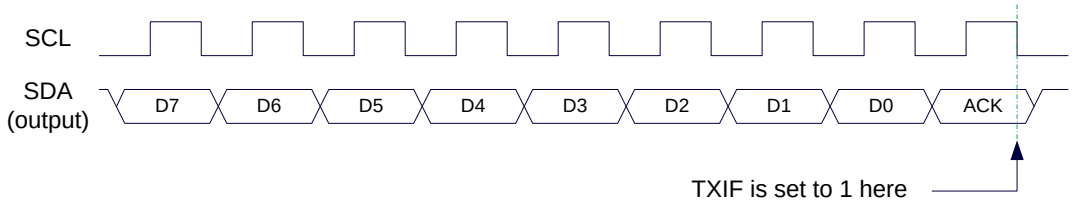
STAIF: interrupt request bit after START signal is received in slave mode. After the I²C controller in slave mode receives the START signal, this bit is set by hardware automatically and requests interrupt to CPU. The STAIF bit must be cleared by software after the interrupt is responded. The time point of STAIF being set is shown below:



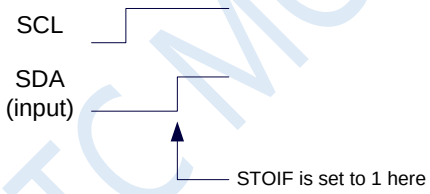
RXIF: interrupt request bit after 1-byte datum is received in slave mode. After the I²C controller in slave mode receives a 1-byte datum, this bit is set by hardware automatically at the falling edge of the 8th clock and will request interrupt to CPU. The RXIF bit must be cleared by software after the interrupt is responded. The time point of RXIF being set is shown in the figure below:



TXIF: interrupt request bit after 1-byte datum transmission is completed in slave mode. After the I²C controller in slave mode completes sending 1 byte of datum and receives a 1-bit ACK signal successfully, this bit is set by hardware automatically at the falling edge of the 9th clock and requests an interrupt to CPU. TXIF bit must be cleared by software after the interrupt is responded. The time point of TXIF being set is shown below:

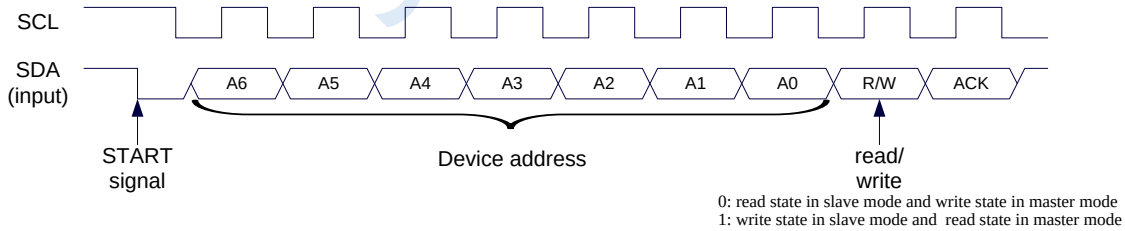


STOIF: interrupt request bit after STOP signal is received in slave mode. After the I²C controller in slave mode receives the STOP signal, this bit is set by hardware automatically and requests interrupt to CPU. The STOIF bit must be cleared by software after the interrupt is serviced. The time point of STOIF being set is shown below:



SLACKI: ACK data received in slave mode

SLACKO: the ACK signal ready to send out in slave mode.



20.3.3 I²C Slave Address Register (I2CSLADR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLADR	FE85H	SLADR[7:1]							MA

SLADR[7:1]: the slave device address

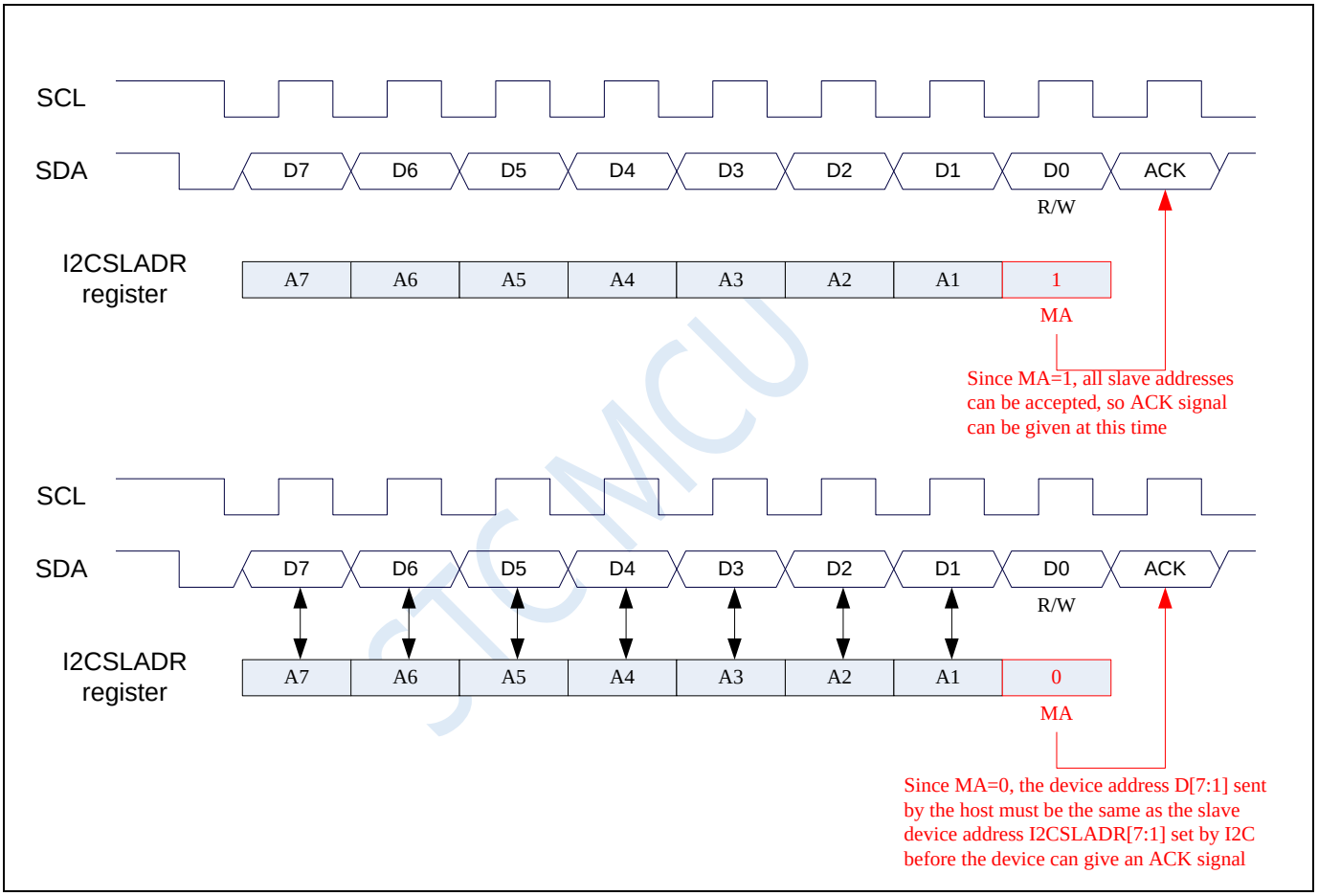
When the I²C controller is in slave mode, the controller will continue to detect the device address and read / write signals sent by the master after it receives the START signal. If the device address sent by the master matches the slave device address set in SLADR[7: 1], the controller will request an interrupt to CPU to process the I²C event. Otherwise, if the device address does not match, the I²C controller continues to monitor, wait for the next START signal, and match the next device address.

MA: Slave device address matching control bit

0: The device address must continue to match SLADR [7: 0].

1: Ignore the settings in SLADR and match all device addresses.

Note: The I2C bus protocol stipulates that a maximum of 128 I2C devices (theoretical value) can be mounted on the I2C bus, and different I2C devices are identified by different I2C slave device addresses. After the I2C master sends the start signal, the upper 7 bits of the first data (DATA0) sent are the slave device address (DATA0[7:1] is the I2C device address), and the lowest bit is the read and write signal. When the I2C device slave address register MA (I2CSLADR.0) is 1, it means that the I2C slave can accept all device addresses. At this time, any device address sent by the host, that is, DATA0[7:1] is any value, the slave can respond. When the I2C device slave address register MA (I2CSLADR.0) is 0, the device address DATA0[7:1] sent by the host must be the same as the device address I2CSLADR[7:1] of the slave to access the slave device.



20.3.4 I²C data Registers (I2CTXD, I2CRXD)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CTXD	FE86H								
I2CRXD	FE87H								

I2CTXD is the I²C transmit data register that holds the I²C data to be transmitted.

I2CRXD is the I²C receive data register that holds the I²C data received.

20.4 Example Routines

20.4.1 I²C is Used to Access AT24C256 in Master Mode (Interrupt Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P_SW2           =   0xba;

#define     I2CCFG              (*(unsigned char volatile xdata *)0xfe80)
#define     I2CMSCR             (*(unsigned char volatile xdata *)0xfe81)
#define     I2CMSST             (*(unsigned char volatile xdata *)0xfe82)
#define     I2CSLCR             (*(unsigned char volatile xdata *)0xfe83)
#define     I2CSLST             (*(unsigned char volatile xdata *)0xfe84)
#define     I2CSLADR            (*(unsigned char volatile xdata *)0xfe85)
#define     I2CTXD              (*(unsigned char volatile xdata *)0xfe86)
#define     I2CRXD              (*(unsigned char volatile xdata *)0xfe87)

sfr         P0M1            =   0x93;
sfr         P0M0            =   0x94;
sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P2M1            =   0x95;
sfr         P2M0            =   0x96;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P4M1            =   0xb3;
sfr         P4M0            =   0xb4;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        SDA             =   P1^4;
sbit        SCL             =   P1^5;

bit         busy;

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;
    if (I2CMSST & 0x40)
    {
        I2CMSST &= ~0x40;                   //Clear interrupt flag
        busy = 0;
    }
    _pop_(P_SW2);
}

void Start()
```

```
{
    busy = 1;
    I2CMSCR = 0x81;                         //Send START command
    while (busy);
}

void SendData(char dat)
{
    I2CTXD = dat;                           //Write data to the data buffer
    busy = 1;
    I2CMSCR = 0x82;                         //Send a SEND command
    while (busy);
}

void RecvACK()
{
    busy = 1;
    I2CMSCR = 0x83;                         //Send read ACK command
    while (busy);
}

char RecvData()
{
    busy = 1;
    I2CMSCR = 0x84;                         //Send RECV command
    while (busy);
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                         //Setup the ACK signal
    busy = 1;
    I2CMSCR = 0x85;                         //Send ACK command
    while (busy);
}

void SendNAK()
{
    I2CMSST = 0x01;                         //Setup the NAK signal
    busy = 1;
    I2CMSCR = 0x85;                         //Send ACK command
    while (busy);
}

void Stop()
{
    busy = 1;
    I2CMSCR = 0x86;                         //Send STOP command
    while (busy);
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
```

STC MCU

```
            _nop_();
            _nop_();
            _nop_();
        }
}

void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        P_SW2 = 0x80;

        I2CCFG = 0xe0;                          //Enable I2C master mode
        I2CMSST = 0x00;
        EA = 1;

        Start();      //Send start command
        SendData(0xa0);                         //Send device address + write command
        RecvACK();
        SendData(0x00);                         //Send storage address high byte
        RecvACK();
        SendData(0x00);                         //Send storage address low byte
        RecvACK();
        SendData(0x12);                         //Write test data 1
        RecvACK();
        SendData(0x78);                         //Write test data 2
        RecvACK();
        Stop();                                 //Send stop command

        Delay();                                //Waiting for the device to write data

        Start();                                //Send start command
        SendData(0xa0);                         //Send device address + write command
        RecvACK();
        SendData(0x00);                         //Send storage address high byte
        RecvACK();
        SendData(0x00);                         //Send storage address low byte
        RecvACK();
        Start();                                //Send start command
        SendData(0xa1);                         //Send device address + read command
        RecvACK();
        P0 = RecvData();                        //Read data 1
        SendACK();
        P2 = RecvData();                        //Read data 2
        SendNAK();
        Stop();                                 //Send stop command

        P_SW2 = 0x00;
```

STC MCU

```
    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5

BUSY        BIT     20H.0

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00C3H
            LJMP    I2CISR

            ORG     0100H
I2CISR:
            PUSH    ACC
            PUSH    DPL
            PUSH    DPH

            MOV     DPTR,#I2CMSST       ;Clear interrupt flag
            MOVX    A,@DPTR
            ANL     A,#NOT 40H
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            CLR     BUSY                ;Reset busy flag

            POP     DPH
            POP     DPL
            POP     ACC
            RETI
```

STC MCU

```
START:
        SETB    BUSY
        MOV     A,#10000001B        ;Send START command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
SENDDATA:
        MOV     DPTR,#I2CTXD        ;Write data to the data buffer
        MOVX    @DPTR,A
        SETB    BUSY
        MOV     A,#10000010B        ;Send a SEND command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
RECVACK:
        SETB    BUSY
        MOV     A,#10000011B        ;Send read ACK command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
RECVDATA:
        SETB    BUSY
        MOV     A,#10000100B        ;Send RECV command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        CALL    WAIT
        MOV     DPTR,#I2CRXD        ;Read data from the data buffer
        MOVX    A,@DPTR
        RET
SENDACK:
        MOV     A,#00000000B        ;Setup the ACK signal
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        SETB    BUSY
        MOV     A,#10000101B        ;Send ACK command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
SENDNAK:
        MOV     A,#00000001B        ;Setup the NAK signal
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        SETB    BUSY
        MOV     A,#10000101B        ;Send ACK command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
STOP:
        SETB    BUSY
        MOV     A,#10000110B        ;Send STOP command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
WAIT:
        JB      BUSY,$              ;Wait for the command to be sent
        RET

DELAY:
```

```
        MOV     R0,#0
        MOV     R1,#0
DELAY1:
        NOP
        NOP
        NOP
        NOP
        DJNZ    R1,DELAY1
        DJNZ    R0,DELAY1
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#80H

        MOV     A,#11100000B        ;Set the I2C module as master mode
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        SETB    EA

        CALL    START               ;Send start command
        MOV     A,#0A0H
        CALL    SENDDATA            ;Send device address + write command
        CALL    RECVACK
        MOV     A,#000H             ;Send storage address high byte
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H             ;Send storage address low byte
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#12H              ;Write test data 1
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#78H              ;Write test data 2
        CALL    SENDDATA
        CALL    RECVACK
        CALL    STOP                ;Send stop command

        CALL    DELAY               ;Waiting for the device to write data

        CALL    START               ;Send start command
        MOV     A,#0A0H             ;Send device address + write command
        CALL    SENDDATA
```

STC MCU

```
        CALL    RECVACK
        MOV     A,#000H             ;Send storage address high byte
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H             ;Send storage address low byte
        CALL    SENDDATA
        CALL    RECVACK
        CALL    START               ;Send start command
        MOV     A,#0A1H             ;Send device address + read command
        CALL    SENDDATA
        CALL    RECVACK
        CALL    RECVDATA            ;Read data 1
        MOV     P0,A
        CALL    SENDACK
        CALL    RECVDATA            ;Read data 2
        MOV     P2,A
        CALL    SENDNAK
        CALL    STOP                ;Send stop command

        JMP     $

        END
```

20.4.2 I²C is Used to Access AT24C256 in Master Mode AT24C256 (Polling Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;

#define I2CCFG          (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR         (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST         (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR         (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST         (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR        (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD          (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD          (*(unsigned char volatile xdata *)0xfe87)

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
```

```
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    SDA     =   P1^4;
sbit    SCL     =   P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                 //Send START command
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                   //Write data to the data buffer
    I2CMSCR = 0x02;                 //Send a SEND command
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                 //Send read ACK command
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                 //Send RECV command
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                 //Setup the ACK signal
    I2CMSCR = 0x05;                 //Send ACK command
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                 //Setup the NAK signal
    I2CMSCR = 0x05;                 //Send ACK command
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                 //Send STOP command
    Wait();
}

void Delay()
```

STC MCU

```
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0xe0;                  //Enable I2C master mode
    I2CMSST = 0x00;

    Start();                        //Send start command
    SendData(0xa0);                 //Send device address + write command
    RecvACK();
    SendData(0x00);                 //Send storage address high byte
    RecvACK();
    SendData(0x00);                 //Send storage address low byte
    RecvACK();
    SendData(0x12);                 //Write test data 1
    RecvACK();
    SendData(0x78);                 //Write test data 2
    RecvACK();
    Stop();                         //Send stop command

    Delay();                        //Waiting for the device to write data

    Start();                        //Send start command
    SendData(0xa0);                 //Send device address + write command
    RecvACK();
    SendData(0x00);                 //Send storage address high byte
    RecvACK();
    SendData(0x00);                 //Send storage address low byte
    RecvACK();
    Start();                        //Send start command
    SendData(0xa1);                 //Send device address + read command
    RecvACK();
    P0 = RecvData();                //Read data 1
    SendACK();
```

STC MCU

```
    P2 = RecvData();                    //Read data 2
    SendNAK();
    Stop();                             //Send stop command

    P_SW2 = 0x00;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
START:
            MOV     A,#00000001B    ;Send START command
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDDATA:
            MOV     DPTR,#I2CTXD    ;Write data to the data buffer
            MOVX    @DPTR,A
            MOV     A,#00000010B    ;Send a SEND command
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVACK:
            MOV     A,#00000011B    ;Send read ACK command
            MOV     DPTR,#I2CMSCR
```

STC MCU

```
            MOVX    @DPTR,A
            JMP     WAIT
RECVDATA:
            MOV     A,#00000100B        ;Send RECV command
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            CALL    WAIT
            MOV     DPTR,#I2CRXD        ;Read data from the data buffer
            MOVX    A,@DPTR
            RET
SENDACK:
            MOV     A,#00000000B        ;Setup the ACK signal
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            MOV     A,#00000101B        ;Send ACK command
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDNAK:
            MOV     A,#00000001B        ;Setup the NAK signal
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            MOV     A,#00000101B        ;Send ACK command
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
STOP:
            MOV     A,#00000110B        ;Send STOP command
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
WAIT:
            MOV     DPTR,#I2CMSST       ;Clear interrupt flag
            MOVX    A,@DPTR
            JNB     ACC.6,WAIT
            ANL     A,#NOT 40H
            MOVX    @DPTR,A
            RET

DELAY:
            MOV     R0,#0
            MOV     R1,#0
DELAY1:
            NOP
            NOP
            NOP
            NOP
            DJNZ    R1,DELAY1
            DJNZ    R0,DELAY1
            RET

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
```

```
MOV     P3M0, #00H
MOV     P3M1, #00H
MOV     P4M0, #00H
MOV     P4M1, #00H
MOV     P5M0, #00H
MOV     P5M1, #00H

MOV     P_SW2,#80H

MOV     A,#11100000B        ;Set the I2C module as master mode
MOV     DPTR,#I2CCFG
MOVX    @DPTR,A
MOV     A,#00000000B
MOV     DPTR,#I2CMSST
MOVX    @DPTR,A

CALL    START               ;Send start command
MOV     A,#0A0H
CALL    SENDDATA            ;Send device address + write command
CALL    RECVACK
MOV     A,#000H             ;Send storage address high byte
CALL    SENDDATA
CALL    RECVACK
MOV     A,#000H             ;Send storage address low byte
CALL    SENDDATA
CALL    RECVACK
MOV     A,#12H              ;Write test data 1
CALL    SENDDATA
CALL    RECVACK
MOV     A,#78H              ;Write test data 2
CALL    SENDDATA
CALL    RECVACK
CALL    STOP                ;Send stop command

CALL    DELAY               ;Waiting for the device to write data

CALL    START               ;Send start command
MOV     A,#0A0H             ;Send device address + write command
CALL    SENDDATA
CALL    RECVACK
MOV     A,#000H             ;Send storage address high byte
CALL    SENDDATA
CALL    RECVACK
MOV     A,#000H             ;Send storage address low byte
CALL    SENDDATA
CALL    RECVACK
CALL    START               ;Send start command
MOV     A,#0A1H             ;Send device address + read command
CALL    SENDDATA
CALL    RECVACK
CALL    RECVDATA            ;Read data 1
MOV     P0,A
CALL    SENDACK
CALL    RECVDATA            ;Read data 2
MOV     P2,A
CALL    SENDNAK
CALL    STOP                ;Send stop command

JMP     $
```

STC MCU

```
END
```

20.4.3 I²C is Used to Access PCF8563 in Master Mode

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P_SW2           =   0xba;

#define     I2CCFG              (*(unsigned char volatile xdata *)0xfe80)
#define     I2CMSCR             (*(unsigned char volatile xdata *)0xfe81)
#define     I2CMSST             (*(unsigned char volatile xdata *)0xfe82)
#define     I2CSLCR             (*(unsigned char volatile xdata *)0xfe83)
#define     I2CSLST             (*(unsigned char volatile xdata *)0xfe84)
#define     I2CSLADR            (*(unsigned char volatile xdata *)0xfe85)
#define     I2CTXD              (*(unsigned char volatile xdata *)0xfe86)
#define     I2CRXD              (*(unsigned char volatile xdata *)0xfe87)

sfr         P0M1            =   0x93;
sfr         P0M0            =   0x94;
sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P2M1            =   0x95;
sfr         P2M0            =   0x96;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P4M1            =   0xb3;
sfr         P4M0            =   0xb4;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        SDA             =   P1^4;
sbit        SCL             =   P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                             //Send START command
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                               //Write data to the data buffer
    I2CMSCR = 0x02;                             //Send a SEND command
    Wait();
```

```
}

void RecvACK()
{
    I2CMSCR = 0x03;                 //Send read ACK command
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                 //Send RECV command
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                 //Setup the ACK signal
    I2CMSCR = 0x05;                 //Send ACK command
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                 //Setup the NAK signal
    I2CMSCR = 0x05;                 //Send ACK command
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                 //Send STOP command
    Wait();
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
```

STC MCU

```
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0xe0;                          //Enable I2C master mode
    I2CMSST = 0x00;

    Start();                                //Send start command
    SendData(0xa2);                         //Send device address + write command
    RecvACK();
    SendData(0x02);                         //Send storage address
    RecvACK();
    SendData(0x00);                         //Set second value
    RecvACK();
    SendData(0x00);                         //Set minute value
    RecvACK();
    SendData(0x12);                         //Set hour value
    RecvACK();
    Stop();                                 //Send stop command

    while (1)
    {
        Start();                            //Send start command
        SendData(0xa2);                     //Send device address + write command
        RecvACK();
        SendData(0x02);                     //Send storage address
        RecvACK();
        Start();                            //Send start command
        SendData(0xa3);                     //Send device address + read command
        RecvACK();
        P0 = RecvData();                    //Read second value
        SendACK();
        P2 = RecvData();                    //Read minute value
        SendACK();
        P3 = RecvData();                    //Read hour value
        SendNAK();
        Stop();                             //Send stop command

        Delay();
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H
```

```
SDA             BIT         P1.4
SCL             BIT         P1.5

P0M1            DATA        093H
P0M0            DATA        094H
P1M1            DATA        091H
P1M0            DATA        092H
P2M1            DATA        095H
P2M0            DATA        096H
P3M1            DATA        0B1H
P3M0            DATA        0B2H
P4M1            DATA        0B3H
P4M0            DATA        0B4H
P5M1            DATA        0C9H
P5M0            DATA        0CAH

                ORG         0000H
                LJMP        MAIN

                ORG         0100H
START:
                MOV         A,#00000001B        ;Send START command
                MOV         DPTR,#I2CMSCR
                MOVX        @DPTR,A
                JMP         WAIT
SENDDATA:
                MOV         DPTR,#I2CTXD        ;Write data to the data buffer
                MOVX        @DPTR,A
                MOV         A,#00000010B        ;Send a SEND command
                MOV         DPTR,#I2CMSCR
                MOVX        @DPTR,A
                JMP         WAIT
RECVACK:
                MOV         A,#00000011B        ;Send read ACK command
                MOV         DPTR,#I2CMSCR
                MOVX        @DPTR,A
                JMP         WAIT
RECVDATA:
                MOV         A,#00000100B        ;Send RECV command
                MOV         DPTR,#I2CMSCR
                MOVX        @DPTR,A
                CALL        WAIT
                MOV         DPTR,#I2CRXD        ;Read data from the data buffer
                MOVX        A,@DPTR
                RET
SENDACK:
                MOV         A,#00000000B        ;Setup the ACK signal
                MOV         DPTR,#I2CMSST
                MOVX        @DPTR,A
                MOV         A,#00000101B        ;Send ACK command
                MOV         DPTR,#I2CMSCR
                MOVX        @DPTR,A
                JMP         WAIT
SENDNAK:
                MOV         A,#00000001B        ;Setup the NAK signal
                MOV         DPTR,#I2CMSST
                MOVX        @DPTR,A
                MOV         A,#00000101B        ;Send ACK command
                MOV         DPTR,#I2CMSCR
```

```
            MOVX    @DPTR,A
            JMP     WAIT
STOP:
            MOV     A,#00000110B        ;Send STOP command
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
WAIT:
            MOV     DPTR,#I2CMSST       ;Clear interrupt flag
            MOVX    A,@DPTR
            JNB     ACC.6,WAIT
            ANL     A,#NOT 40H
            MOVX    @DPTR,A
            RET

DELAY:
            MOV     R0,#0
            MOV     R1,#0
DELAY1:
            NOP
            NOP
            NOP
            NOP
            DJNZ    R1,DELAY1
            DJNZ    R0,DELAY1
            RET

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P_SW2,#80H

            MOV     A,#11100000B        ;Set the I2C module as master mode
            MOV     DPTR,#I2CCFG
            MOVX    @DPTR,A
            MOV     A,#00000000B
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A

            CALL    START               ;Send start command
            MOV     A,#0A2H
            CALL    SENDDATA            ;Send device address + write command
            CALL    RECVACK
            MOV     A,#002H             ;Send storage address
            CALL    SENDDATA
            CALL    RECVACK
            MOV     A,#00H              ;Set second value
```

STC MCU

```
                CALL    SENDDATA
                CALL    RECVACK
                MOV     A,#00H          ;Set minute value
                CALL    SENDDATA
                CALL    RECVACK
                MOV     A,#12H          ;Set hour value
                CALL    SENDDATA
                CALL    RECVACK
                CALL    STOP            ;Send stop command
LOOP:
                CALL    START           ;Send start command
                MOV     A,#0A2H         ;Send device address + write command
                CALL    SENDDATA
                CALL    RECVACK
                MOV     A,#002H         ;Send storage address
                CALL    SENDDATA
                CALL    RECVACK
                CALL    START           ;Send start command
                MOV     A,#0A3H         ;Send device address + read command
                CALL    SENDDATA
                CALL    RECVACK
                CALL    RECVDATA        ;Read second value
                MOV     P0,A
                CALL    SENDACK
                CALL    RECVDATA        ;Read minute value
                MOV     P2,A
                CALL    SENDACK
                CALL    RECVDATA        ;Read hour value
                MOV     P3,A
                CALL    SENDNAK
                CALL    STOP            ;Send stop command

                CALL    DELAY

                JMP     LOOP

                END
```

20.4.4 I²C Slave Mode (Polling Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
```

```
#define   I2CRXD          (*(unsigned char volatile xdata *)0xfe87)

sfr       P0M1      =     0x93;
sfr       P0M0      =     0x94;
sfr       P1M1      =     0x91;
sfr       P1M0      =     0x92;
sfr       P2M1      =     0x95;
sfr       P2M0      =     0x96;
sfr       P3M1      =     0xb1;
sfr       P3M0      =     0xb2;
sfr       P4M1      =     0xb3;
sfr       P4M0      =     0xb4;
sfr       P5M1      =     0xc9;
sfr       P5M0      =     0xca;

sbit      SDA       =     P1^4;
sbit      SCL       =     P1^5;

bit       isda;                                 //Device address flag
bit       isma;                                 //Storage address flag
unsigned char           addr;
unsigned char pdata     buffer[256];

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;

    if (I2CSLST & 0x40)
    {
        I2CSLST &= ~0x40;                       //Handle the START event
    }
    else if (I2CSLST & 0x20)
    {
        I2CSLST &= ~0x20;                       //Handle the RECV event
        if (isda)
        {
            isda = 0;                           //Handle the RECV event（RECV DEVICE ADDR）
        }
        else if (isma)
        {
            isma = 0;                           //Handle the RECV event（RECV MEMORY ADDR）
            addr = I2CRXD;
            I2CTXD = buffer[addr];
        }
        else
        {
            buffer[addr++] = I2CRXD;            //Handle the RECV event（RECV DATA）
        }
    }
    else if (I2CSLST & 0x10)
    {
        I2CSLST &= ~0x10;                       //Handle the SEND event
        if (I2CSLST & 0x02)
        {
            I2CTXD = 0xff;                      //Stop receiving data when receiving NAK
        }
        else
        {
```

STC MCU

```
            I2CTXD = buffer[++addr];                    //Continue reading data when receiving ACK
        }
    }
    else if (I2CSLST & 0x08)
    {
        I2CSLST &= ~0x08;                               //Handle the STOP event
        isda = 1;
        isma = 1;
    }

    _pop_(P_SW2);
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0x81;                                      //Enable I2C slave mode
    I2CSLADR = 0x5a;       // Set the slave device address register I2CSLADR=0101_1010B
                    //That is, I2CSLADR[7:1]=010_1101B,MA=0B.
                //Since MA is 0, the device address sent by the host must be the same as
                //I2CSLADR[7:1] to access this I2C slave device.
                //If the host needs to write data, it must send 5AH (0101_1010B)
                //If the host needs to read data, it must send 5BH (0101_1011B)
    I2CSLST = 0x00;
    I2CSLCR = 0x78;                                     //Enable interrupt of slave mode
    EA = 1;

    isda = 1;                                           //User variable initialization
    isma = 1;
    addr = 0;
    I2CTXD = buffer[addr];

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
```

```
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5
ISDA        BIT     20H.0       ;Device address flag
ISMA        BIT     20H.1       ;Storage address flag

ADDR        DATA    21H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00C3H
            LJMP    I2CISR

            ORG     0100H
I2CISR:
            PUSH    ACC
            PUSH    PSW
            PUSH    DPL
            PUSH    DPH
            MOV     DPTR,#I2CSLST   ;Detect slave status
            MOVX    A,@DPTR
            JB      ACC.6,STARTIF
            JB      ACC.5,RXIF
            JB      ACC.4,TXIF
            JB      ACC.3,STOPIF
ISREXIT:
            POP     DPH
            POP     DPL
            POP     PSW
            POP     ACC
            RETI
STARTIF:
            ANL     A,#NOT 40H      ;Handle the START event
            MOVX    @DPTR,A
            JMP     ISREXIT
RXIF:
            ANL     A,#NOT 20H      ;Handle the RECV event
            MOVX    @DPTR,A
            MOV     DPTR,#I2CRXD
            MOVX    A,@DPTR
            JBC     ISDA,RXDA
            JBC     ISMA,RXMA
```

STC MCU

```
            MOV     R0,ADDR             ;Handle the RECV event（RECV DATA）
            MOVX    @R0,A
            INC     ADDR
            JMP     ISREXIT
RXDA:
            JMP     ISREXIT             ;Handle the RECV event（RECV DEVICE ADDR）
RXMA:
            MOV     ADDR,A              ;Handle the RECV event（RECV MEMORY ADDR）
            MOV     R0,A
            MOVX    A,@R0
            MOV     DPTR,#I2CTXD
            MOVX    @DPTR,A
            JMP     ISREXIT
TXIF:
            ANL     A,#NOT 10H          ;Handle the SEND event
            MOVX    @DPTR,A
            JB      ACC.1,RXNAK
            INC     ADDR
            MOV     R0,ADDR
            MOVX    A,@R0
            MOV     DPTR,#I2CTXD
            MOVX    @DPTR,A
            JMP     ISREXIT
RXNAK:
            MOVX    A,#0FFH
            MOV     DPTR,#I2CTXD
            MOVX    @DPTR,A
            JMP     ISREXIT
STOPIF:
            ANL     A,#NOT 08H          ;Handle the STOP event
            MOVX    @DPTR,A
            SETB    ISDA
            SETB    ISMA
            JMP     ISREXIT

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P_SW2,#80H

            MOV     A,#10000001B        ;Enable I2C slave mode
            MOV     DPTR,#I2CCFG
            MOVX    @DPTR,A
            MOV     A,#01011010B    ; Set the slave device address register I2CSLADR=0101_1010B
                ;That is, I2CSLADR[7:1]=010_1101B,MA=0B.
            ;Since MA is 0, the device address sent by the host must be the same as
            ;I2CSLADR[7:1] to access this I2C slave device.
```

STC MCU

```
;If the host needs to write data, it must send 5AH (0101_1010B)
;If the host needs to read data, it must send 5BH (0101_1011B)
MOV     DPTR,#I2CSLADR
MOVX    @DPTR,A
MOV     A,#00000000B
MOV     DPTR,#I2CSLST
MOVX    @DPTR,A
MOV     A,#01111000B            ;Enable interrupt of slave mode
MOV     DPTR,#I2CSLCR
MOVX    @DPTR,A

SETB    ISDA                    ;User variable initialization
SETB    ISMA
CLR     A
MOV     ADDR,A
MOV     R0,A
MOVX    A,@R0
MOV     DPTR,#I2CTXD
MOVX    @DPTR,A

SETB    EA

SJMP    $

END
```

20.4.5 I²C Slave Mode (Polling Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;

#define I2CCFG          (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR         (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST         (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR         (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST         (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR        (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD          (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD          (*(unsigned char volatile xdata *)0xfe87)

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
```

```
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        SDA         =   P1^4;
sbit        SCL         =   P1^5;

bit         isda;                               //Device address flag
bit         isma;                               //Storage address flag
unsigned char           addr;
unsigned char pdata     buffer[256];

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0x81;//Enable I2C slave mode
    I2CSLADR = 0x5a;                            //Set the slave device address register I2CSLADR=0101_1010B
            //That is, I2CSLADR[7:1]=010_1101B,MA=0B.
            //Since MA is 0, the device address sent by the host must be the same as
            //I2CSLADR[7:1] to access this I2C slave device.
            //If the host needs to write data, it must send 5AH (0101_1010B)
            //If the host needs to read data, it must send 5BH (0101_1011B)
    I2CSLST = 0x00;
    I2CSLCR = 0x00;                             //Disable interrupt of slave mode

    isda = 1;                                   //User variable initialization
    isma = 1;
    addr = 0;
    I2CTXD = buffer[addr];

    while (1)
    {
        if (I2CSLST & 0x40)
        {
            I2CSLST &= ~0x40;                   //Handle the START event
        }
        else if (I2CSLST & 0x20)
        {
            I2CSLST &= ~0x20;                   //Handle the RECV event
            if (isda)
            {
                isda = 0;                       //Handle the RECV event（RECV DEVICE ADDR）
            }
            else if (isma)
            {
                isma = 0;                       //Handle the RECV event（RECV MEMORY ADDR）
```

```
                addr = I2CRXD;
                I2CTXD = buffer[addr];
            }
            else
            {
                buffer[addr++] = I2CRXD;                //Handle the RECV event（RECV DATA）
            }
        }
        else if (I2CSLST & 0x10)
        {
            I2CSLST &= ~0x10;                           //Handle the SEND event
            if (I2CSLST & 0x02)
            {
                I2CTXD = 0xff;                          //Stop receiving data when receiving NAK
            }
            else
            {
                I2CTXD = buffer[++addr];                //Continue reading data when receiving ACK
            }
        }
        else if (I2CSLST & 0x08)
        {
            I2CSLST &= ~0x08;                           //Handle the STOP event
            isda = 1;
            isma = 1;
        }
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5
ISDA        BIT     20H.0       ;Device address flag
ISMA        BIT     20H.1       ;Storage address flag

ADDR        DATA    21H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
```

```
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P_SW2,#80H

            MOV     A,#10000001B            ;Enable I2C slave mode
            MOV     DPTR,#I2CCFG
            MOVX    @DPTR,A
            MOV     A,#01011010B    ; Set the slave device address register I2CSLADR=0101_1010B
                ;That is, I2CSLADR[7:1]=010_1101B,MA=0B.
            ;Since MA is 0, the device address sent by the host must be the same as
            ;I2CSLADR[7:1] to access this I2C slave device.
            ;If the host needs to write data, it must send 5AH (0101_1010B)
            ;If the host needs to read data, it must send 5BH (0101_1011B)
            MOV     DPTR,#I2CSLADR
            MOVX    @DPTR,A
            MOV     A,#00000000B
            MOV     DPTR,#I2CSLST
            MOVX    @DPTR,A
            MOV     A,#00000000B            ;Disable interrupt of slave mode
            MOV     DPTR,#I2CSLCR
            MOVX    @DPTR,A

            SETB    ISDA                    ;User variable initialization
            SETB    ISMA
            CLR     A
            MOV     ADDR,A
            MOV     R0,A
            MOVX    A,@R0
            MOV     DPTR,#I2CTXD
            MOVX    @DPTR,A

LOOP:
            MOV     DPTR,#I2CSLST           ;Detect slave status
            MOVX    A,@DPTR
            JB      ACC.6,STARTIF
            JB      ACC.5,RXIF
            JB      ACC.4,TXIF
```

```
            JB      ACC.3,STOPIF
            JMP     LOOP
STARTIF:
            ANL     A,#NOT 40H          ;Handle the START event
            MOVX    @DPTR,A
            JMP     LOOP
RXIF:
            ANL     A,#NOT 20H          ;Handle the RECV event
            MOVX    @DPTR,A
            MOV     DPTR,#I2CRXD
            MOVX    A,@DPTR
            JBC     ISDA,RXDA
            JBC     ISMA,RXMA
            MOV     R0,ADDR             ;Handle the RECV event（RECV DATA）
            MOVX    @R0,A
            INC     ADDR
            JMP     LOOP
RXDA:
            JMP     LOOP                ;Handle the RECV event（RECV DEVICE ADDR）
RXMA:
            MOV     ADDR,A              ;Handle the RECV event（RECV MEMORY ADDR）
            MOV     R0,A
            MOVX    A,@R0
            MOV     DPTR,#I2CTXD
            MOVX    @DPTR,A
            JMP     LOOP
TXIF:
            ANL     A,#NOT 10H          ;Handle the SEND event
            MOVX    @DPTR,A
            JB      ACC.1,RXNAK
            INC     ADDR
            MOV     R0,ADDR
            MOVX    A,@R0
            MOV     DPTR,#I2CTXD
            MOVX    @DPTR,A
            JMP     LOOP
RXNAK:
            MOVX    A,#0FFH
            MOV     DPTR,#I2CTXD
            MOVX    @DPTR,A
            JMP     LOOP
STOPIF:
            ANL     A,#NOT 08H          ;Handle the STOP event
            MOVX    @DPTR,A
            SETB    ISDA
            SETB    ISMA
            JMP     LOOP

            END
```

20.4.6 Master Codes for testing I²C Slave Mode

C language code

```
//Operating frequency for test is 11.0592MHz
```

```
#include "reg51.h"
#include "intrins.h"

sfr         P_SW2           =   0xba;

#define     I2CCFG              (*(unsigned char volatile xdata *)0xfe80)
#define     I2CMSCR             (*(unsigned char volatile xdata *)0xfe81)
#define     I2CMSST             (*(unsigned char volatile xdata *)0xfe82)
#define     I2CSLCR             (*(unsigned char volatile xdata *)0xfe83)
#define     I2CSLST             (*(unsigned char volatile xdata *)0xfe84)
#define     I2CSLADR            (*(unsigned char volatile xdata *)0xfe85)
#define     I2CTXD              (*(unsigned char volatile xdata *)0xfe86)
#define     I2CRXD              (*(unsigned char volatile xdata *)0xfe87)

sfr         P0M1            =   0x93;
sfr         P0M0            =   0x94;
sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P2M1            =   0x95;
sfr         P2M0            =   0x96;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P4M1            =   0xb3;
sfr         P4M0            =   0xb4;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        SDA             =   P1^4;
sbit        SCL             =   P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                         //Send START command
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                           //Write data to the data buffer
    I2CMSCR = 0x02;                         //Send a SEND command
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                         //Send read ACK command
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                         //Send RECV command
    Wait();
```

STC MCU

```
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                         //Setup the ACK signal
    I2CMSCR = 0x05;                         //Send ACK command
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                         //Setup the NAK signal
    I2CMSCR = 0x05;                         //Send ACK command
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                         //Send STOP command
    Wait();
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0xe0;                          //Enable I2C master mode
    I2CMSST = 0x00;

    Start();                                //Send start command
    SendData(0x5a);                         //Send device address (010_1101B) + write command (0B)
    RecvACK();
```

STC MCU

```
    SendData(0x00);                         //Send storage address
    RecvACK();
    SendData(0x12);                         //Write test data 1
    RecvACK();
    SendData(0x78);                         //Write test data 2
    RecvACK();
    Stop();                                 //Send stop command

    Start();                                //Send start command
    SendData(0x5a);                         //Send device address (010_1101B) + write command (0B)
    RecvACK();
    SendData(0x00);                         //Send storage address high byte
    RecvACK();
    Start();                                //Send start command
    SendData(0x5b);                         //Send device address (010_1101B)+ read command (1B)
    RecvACK();
    P0 = RecvData();                        //Read data 1
    SendACK();
    P2 = RecvData();                        //Read data 2
    SendNAK();
    Stop();                                 //Send stop command

    P_SW2 = 0x00;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
```

```
        LJMP    MAIN

        ORG     0100H
START:
        MOV     A,#00000001B        ;Send START command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
SENDDATA:
        MOV     DPTR,#I2CTXD        ;Write data to the data buffer
        MOVX    @DPTR,A
        MOV     A,#00000010B        ;Send a SEND command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
RECVACK:
        MOV     A,#00000011B        ;Send read ACK command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
RECVDATA:
        MOV     A,#00000100B        ;Send RECV command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        CALL    WAIT
        MOV     DPTR,#I2CRXD        ;Read data from the data buffer
        MOVX    A,@DPTR
        RET
SENDACK:
        MOV     A,#00000000B        ;Setup the ACK signal
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        MOV     A,#00000101B        ;Send ACK command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
SENDNAK:
        MOV     A,#00000001B        ;Setup the NAK signal
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        MOV     A,#00000101B        ;Send ACK command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
STOP:
        MOV     A,#00000110B        ;Send STOP command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
WAIT:
        MOV     DPTR,#I2CMSST       ;Clear interrupt flag
        MOVX    A,@DPTR
        JNB     ACC.6,WAIT
        ANL     A,#NOT 40H
        MOVX    @DPTR,A
        RET

DELAY:
        MOV     R0,#0
```

STC MCU

```
        MOV     R1,#0
DELAY1:
        NOP
        NOP
        NOP
        NOP
        DJNZ    R1,DELAY1
        DJNZ    R0,DELAY1
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#80H

        MOV     A,#11100000B        ;Set the I2C module as master mode
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A

        CALL    START               ;Send start command
        MOV     A,#5AH              ;Slave address is 5A
        CALL    SENDDATA            ;Send device address (010_1101B) + write command (0B)
        CALL    RECVACK
        MOV     A,#000H             ;Send storage address
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#12H              ;Write test data 1
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#78H              ;Write test data 2
        CALL    SENDDATA
        CALL    RECVACK
        CALL    STOP                ;Send stop command

        CALL    DELAY               ;Waiting for the device to write data

        CALL    START               ;Send start command
        MOV     A,#5AH              ;Send device address (010_1101B)+ write command (0B)
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H             ;Send storage address
        CALL    SENDDATA
        CALL    RECVACK
        CALL    START               ;Send start command
```

```
        MOV     A,#5BH          ;Send device address (010_1101B) + read command (1B)
        CALL    SENDDATA
        CALL    RECVACK
        CALL    RECVDATA        ;Read data 1
        MOV     P0,A
        CALL    SENDACK
        CALL    RECVDATA        ;Read data 2
        MOV     P2,A
        CALL    SENDNAK
        CALL    STOP            ;Send stop command

        JMP     $

        END
```

STC MCU

21 Touch Key Controller

Product	Touch key
STC8G1K08 family	
STC8G1K08-8Pin family	
STC8G1K08A family	
STC8G2K64S4 family	
STC8G2K64S2 family	
STC8G1K08T family	●
STC15H2K64S4 family	

A touch key controller or touch sensor unit (TSU in short) is integrated in the STC8G series of microcontrollers, which can connect up to 16 keys. It can detect the small capacitance changes caused by a finger touching the key electrode and quantify the capacitance as a 16-bit number. In principle, the TSU module is similar to a 16-bit ADC. The only difference is that the ADC detects and quantifies the simulated voltage or current, while the TSU detects and quantifies the capacitor. For touch-key sensing, a fixed capacitor of about 10nF to 47nF needs to be added for application as a reference. If the temperature of the environment changes rapidly and intensely, a low-temperature drift capacitor should be used to maintain the TSU output data no too much change, so as to avoid software misjudgment.

A description of continuous sensing of a certain key is shown in the following figure. The counter output value is near 2572 if there is no finger touch. The counter output becomes near 2443 if there is a finger touch. The difference of counter output value with or without finger touch is about 5%.

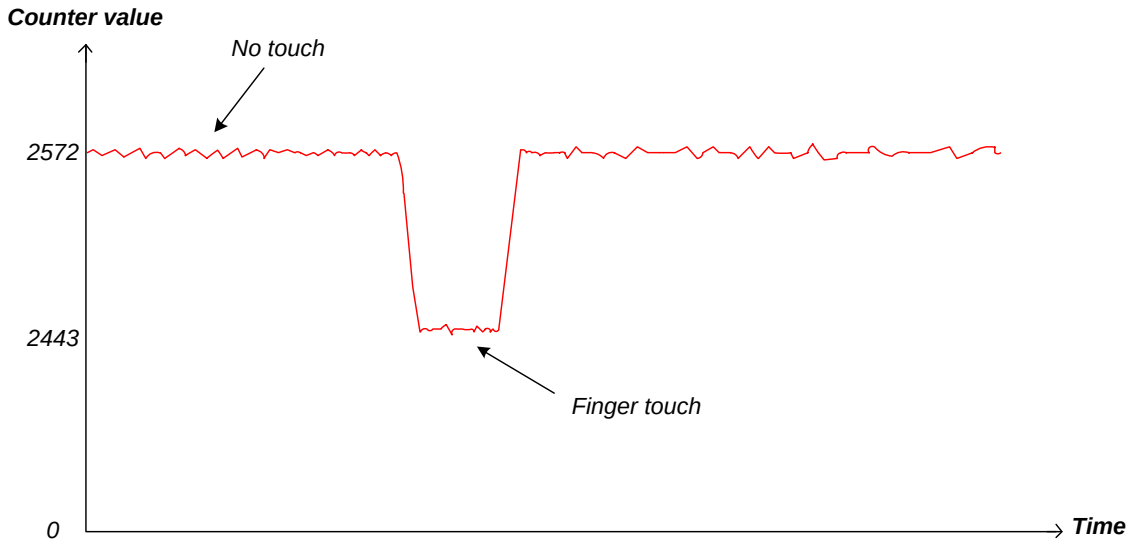


Fig. counter value for touch-sensing

The TSU module can connect up to 16 keys. Each touch key port can be independently enabled through the two 8-bit registers TSCHEN1 and TSCHEN2. The I/O port not used as touch key function can still maintain its

original GPIO or special functions such as LED driver. The frequency of the switched capacitor circuit is selected through SCR [2: 0]. It is recommended not to exceed 12.5MHz to avoid large errors. There are four reference voltage segments of the internal comparator of the TSU module for choosing, which is selected through TSVR [1:0]. Changing the reference voltage will change the touch sensing time and sensitivity.

The touch key scan can be configured to scan continuously or to stop after one round of the enabled key. This function is controlled by the SINGLE bit. The configuration register TSSAMP [1: 0] allows the TSU module to scan four times continuously to one channel to sample and calculate the average value of the data. To do this has the effect of hardware filtering. The WAIT bit allows the TSU module to enter the wait mode when the TSIF flag is 1. TSU will continue to perform key scans until the software clears the WAIT bit to 0, which helps the heavy CPU to have enough time to process other things. The value of the external capacitor Cref is recommended to be in the range of 10nF to 47nF. The time from the initial value of Cref discharging to zero must be sufficient. This can be adjusted flexibly through DT [2: 0].

When a key scan is completed, the output value of the 16-bit counter will be written to TSDATAH and TSDATAL. The flag bit TSIF is set to 1 by hardware, and the scanned touch channel number will be written to TSDNCHN [3: 0]. If the module's external interrupt controller is enabled, TSIF can request interrupt to the CPU. The software may read the TSDNCHN [3: 0] register content to determine which touch channel requested the TSIF interrupt. You can get the status of the TSU module being scanned and the number of keys being scanned in real time by reading TSWKCHN [3: 0] and TSGO using software. If the 16-bit counter overflows, the TSDOV flag will be set to 1 by hardware.

The TSU module can use I/O port in time-sharing multiplexing with the LED driver circuit. It is necessary to enable the LED driving circuit together and time-multiplex the I/O port when the content of the TSRT register is not zero, which means that TSGO enables TSU.Therefore, the relevant registers of the TSU module and the LED driving circuit must be configured in order firstly before TSGO is enabled. In TSU / LED time-sharing multi-tasking mode, in order to maintain the fixed frame rate of the LED, if the time allocated to the key scan has arrived but the key being scanned has not been completed, the incomplete key will be rescaned to start a new key scan.

21.1 Internal Structure Diagram of Touch Key Controller

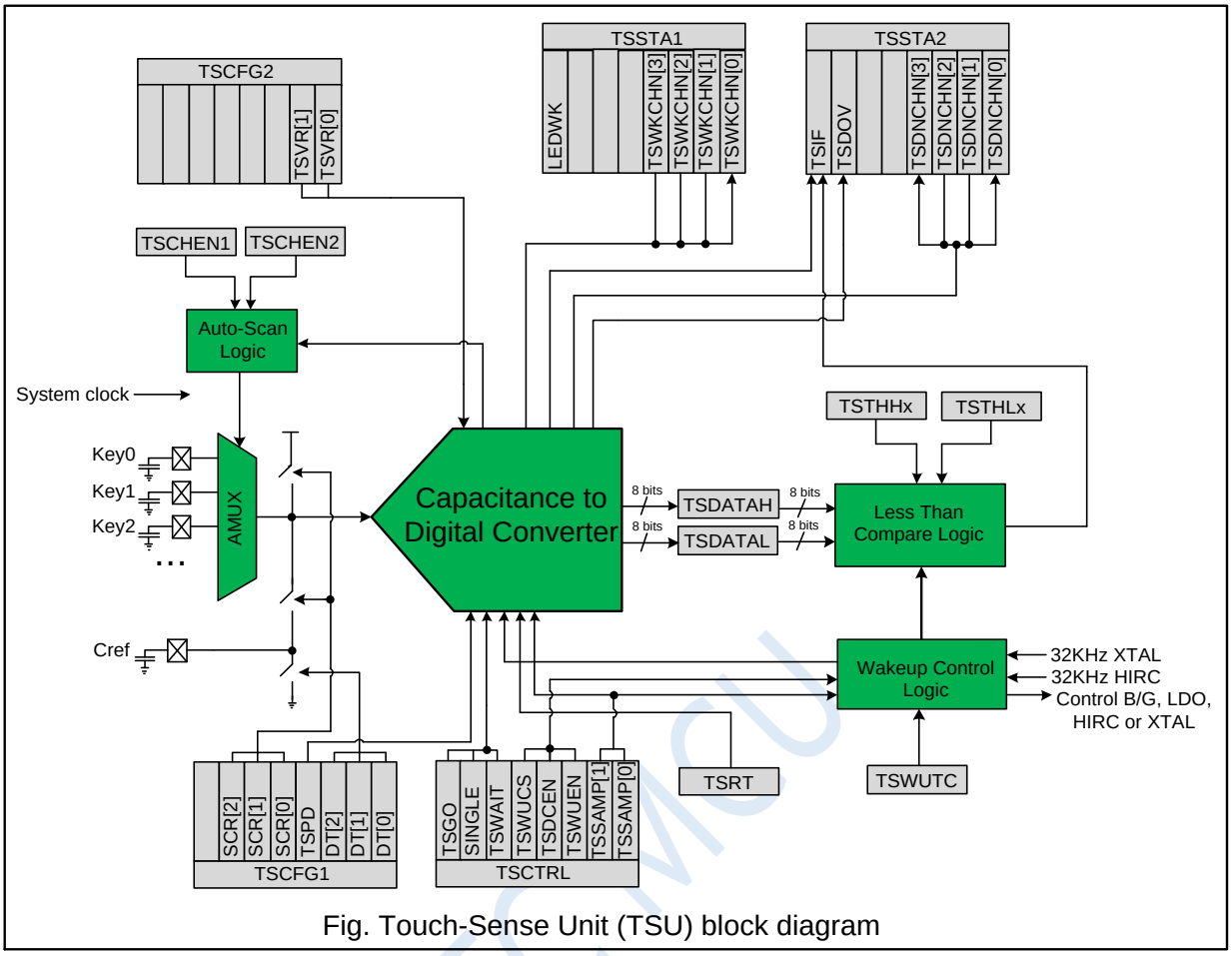


Fig. Touch-Sense Unit (TSU) block diagram

21.2 Wake-up from Low Power Mode using Touch Key

The TSU module has a dedicated timing and control circuit, which can accept an external 32KHz crystal or internal 32KHz RC oscillation circuit as a clock source, and wake up the TSU module at regular intervals to perform key scans, and realize the low-power touch wake-up function by duty control. There is a dedicated 16-bit threshold register {TSTHHx, TSTHLx} in each touch channel. If the wake-up enable bit TSWUEN and the digital comparator enable bit TSDCEN are set, MCU will enter the power-down state, and the entire chip enters a low-power state. In touch wake-up mode, the TSU module can repeatedly and regularly self-wake for key scans to wake up the CPU. If the data (or average result) of the key scan is less than the set threshold, the hardware will set TSIF to 1 and wake up the CPU out of power-down state. There is a hardware averaging circuit inside the TSU module, which can average the maximum of four consecutive scans of the same channel. The TSSAMP [1: 0] register is used to configure the number of samples. {TSDATAH, TSDATAL} stores average result.

21.3 Operation Steps When Touch Key Function is Used only

1. Turn on the TSU power switch TSPD = 0, select the channel to be scanned, and the registers are TSCHEN1 and TSCHEN2.

2. Set the TSRT content to 0x00, which means that the LED driver time-sharing multi-tasking function is not enabled.
3. Configure the switching frequency SCR [2: 0] and discharge time DT [2: 0] according to the value of Cref and the touch key capacitor, and select the internal comparator reference voltage TSVR [1: 0] according to the required scan time and sensitivity.
4. Configure the SINGLE bit to determine whether the scan is stopped automatically or continuously. Configure TSSAMP [1: 0] to allow a channel to be resampled up to four times. If the CPU task is heavy, configure TSWAIT to use the TSIF status to delay scanning of the next channel.
5. Configure TSDCEN to enable internal digital comparison function if necessary.
6. Set TSGO=1 to start the touch key scanning. You can read TSWKCHN [3: 0] to know which channel is currently being scanned using software. After finish scanning a channel, the hardware will set TSIF to 1, and the completed channel number will be written into TSDNCHN [3: 0]. If an overflow occurs, TSOV will also be set to 1. You should read these registers to decide what to do next in your software. TSIF and TSOV can only be set by hardware and cleared by software.
7. If SINGLE = 1, the hardware will clear TSGO automatically and end scanning after one round of scanning, otherwise TSGO will remain at 1 and continue the new scanning.
8. If you want to stop the touch key scanning, you can set TSGO to 0 at any time. If you want to reduce power consumption, you must set TSPD to 1.

21.4 Operation Steps for Wake-up from Low Power Mode using Touch Key

1. Select the channel to be scanned, the registers are TSCHEN1 and TSCHEN2.
2. Be sure to set the TSRT content to 0x00. At this time, you cannot turn on the LED driver time-sharing multitasking function.
3. Configure switching frequency SCR [2: 0], discharge time DT [2: 0] and select internal comparator reference voltage TSVR [1: 0].
4. Set the SINGLE bit to 0 for continuous scanning and set TSWAIT to 0.
5. Configure TSWUCS to determine which clock source is used to wake up the controller, external 32KHz crystal or internal 32KHz IRC.
6. Configure TSWUTC to determine how often TSU needs to work and enter the power saving mode automatically after work.
7. Configure TSSAMP [1: 0] to determine the number of scan samples for each channel, configure TSDCEN = 1 to enable the internal digital comparison function.
8. Configure the wake-up threshold {TSTHHx, TSTHLx} for each channel, which will be compared with the average of the scan results.
9. Enable TSWUEN = 1, set TSPD = 1 to turn off the analog power of the TSU module, enable TSIF to wake up the CPU, then let the MCU enter the power-down state. Once the MCU enters the power-down state, the wake-up controller inside the TSU starts to work, and controls the power switch, key scan, data comparison, and so on of the TSU module periodically.
10. If the data result is lower than the set threshold, the hardware will set TSIF to 1, write the key number in TSDNCHN [3: 0], and the CPU will be woken up, the low power wake-up process ends.
11. After the CPU is awakened, in addition to directly reading TSDNCHN [3: 0] to determine which key is touched, you can also perform a key scan in the normal working mode to confirm whether the wake-up is caused by noise interference.

21.5 Registers Related to Touch Key Controller

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
TSCHEN1	Touch Key Enable Register 1	FB40H	TKEN7	TKEN6	TKEN5	TKEN4	TKEN3	TKEN2	TKEN1	TKEN0	0000,0000
TSCHEN2	Touch Key Enable Register 2	FB41H	TKEN15	TKEN14	TKEN13	TKEN12	TKEN11	TKEN10	TKEN9	TKEN8	0000,0000
TSCFG1	Touch Key Configuration Register 1	FB42H	-	SCR[2:0]			-	DT[2:0]			0000,0000
TSCFG2	Touch Key Configuration Register 2	FB43H	-	-	-	-	-	-	TSVR[1:0]		0000,0000
TSWUTC	Touch key power-down mode wake-up time control register	FB44H									0000,0000
TSCTRL	Touch Key Control Register	FB45H	TSGO	SINGLE	TSWAIT	TSWUCS	TSDCEN	TSWUEN	TSSAMP[1:0]		0000,0000
TSSTA1	Touch Key Status Register 1	FB46H	LEDWK	-	-	-	TSWKCHN[3:0]				0000,0000
TSSTA2	Touch Key Status Register 2	FB47H	TSIF	TSDOV	-	-	TSDNCHN[3:0]				0000,0000
TSRT	Touch Key Time Control Register	FB48H									0000,0000
TSDATH	Touch Key Data High Byte	FB49H									0000,0000
TSDATL	Touch Key Data Low Byte	FB4AH									0000,0000
TSTH00H	Touch Key0 Threshold High Byte	FB50H									0000,0000
TSTH00L	Touch Key0 Threshold Low Byte	FB51H									0000,0000
TSTH01H	Touch Key1 Threshold High Byte	FB52H									0000,0000
TSTH01L	Touch Key1 Threshold Low Byte	FB53H									0000,0000
TSTH02H	Touch Key2 Threshold High Byte	FB54H									0000,0000
TSTH02L	Touch Key2 Threshold Low Byte	FB55H									0000,0000
TSTH03H	Touch Key3 Threshold High Byte	FB56H									0000,0000
TSTH03L	Touch Key3 Threshold Low Byte	FB57H									0000,0000
TSTH04H	Touch Key4 Threshold High Byte	FB58H									0000,0000
TSTH04L	Touch Key4 Threshold Low Byte	FB59H									0000,0000
TSTH05H	Touch Key5 Threshold High Byte	FB5AH									0000,0000
TSTH05L	Touch Key5 Threshold Low Byte	FB5BH									0000,0000
TSTH06H	Touch Key6 Threshold High Byte	FB5CH									0000,0000
TSTH06L	Touch Key6 Threshold Low Byte	FB5DH									0000,0000
TSTH07H	Touch Key7 Threshold High Byte	FB5EH									0000,0000
TSTH07L	Touch Key7 Threshold Low Byte	FB5FH									0000,0000
TSTH08H	Touch Key8 Threshold High Byte	FB60H									0000,0000
TSTH08L	Touch Key8 Threshold Low Byte	FB61H									0000,0000
TSTH09H	Touch Key9 Threshold High Byte	FB62H									0000,0000
TSTH09L	Touch Key9 Threshold Low Byte	FB63H									0000,0000
TSTH10H	Touch Key10 Threshold High Byte	FB64H									0000,0000
TSTH10L	Touch Key10 Threshold Low Byte	FB65H									0000,0000
TSTH11H	Touch Key11 Threshold High Byte	FB66H									0000,0000
TSTH11L	Touch Key11 Threshold Low Byte	FB67H									0000,0000
TSTH12H	Touch Key12 Threshold High Byte	FB68H									0000,0000
TSTH12L	Touch Key12 Threshold Low Byte	FB69H									0000,0000
TSTH13H	Touch Key13 Threshold High Byte	FB6AH									0000,0000
TSTH13L	Touch Key13 Threshold Low Byte	FB6BH									0000,0000
TSTH14H	Touch Key14 Threshold High Byte	FB6CH									0000,0000
TSTH14L	Touch Key14 Threshold Low Byte	FB6DH									0000,0000
TSTH15H	Touch Key15 Threshold High Byte	FB6EH									0000,0000
TSTH15L	Touch Key15 Threshold Low Byte	FB6FH									0000,0000

21.5.1 Touch Key Enable Registers (TSCHENn)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TSCHEN1	FB40H	TKEN7	TKEN6	TKEN5	TKEN4	TKEN3	TKEN2	TKEN1	TKEN0
TSCHEN2	FB41H	TKEN15	TKEN14	TKEN13	TKEN12	TKEN11	TKEN10	TKEN9	TKEN8

TKENn: Touch key enable bit (n＝0~15)

0: Corresponding TKn pin is GPIO

1: Corresponding TKn pin is a touch key

21.5.2 Touch Key Configuration Registers (TSCFGn)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TSCFG1	FB42H	-	SCR[2:0]			-	DT[2:0]		
TSCFG2	FB43H	-	-	-	-	-	-	TSVR[1:0]	

SCR: Configure the switching capacitor working frequency inside the touch key controller (the higher the frequency, the shorter the charging time)

$$\text{Working frequency of the switching capacitor} = \frac{\text{System working frequency}}{2 * (\text{SCR[2:0]} + 1)}$$

DT[2:0]: Configure the initial discharge time of Cref inside the touch key controller to ground

DT[2:0]	discharge time
000	125 system clocks
001	250 system clocks
010	500 system clocks
011	1000 system clocks
100	2000 system clocks
101	2500 system clocks
110	5000 system clocks
111	7500 system clocks

TSCV[1:0]: Configure the reference voltage inside the touch key controller

TSCV[1:0]	reference voltage
00	1/4 AVCC
01	1/2 AVCC
10	5/8 AVCC
11	3/4 AVCC

21.5.3 Touch Key power-down mode wake-up time control register (TSWUTC)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TSWUTC	FB44H								

TSWUTC register is used to configure how often the touch key controller is woken up.

$$\text{Wake-up frequency} = \frac{F_{32K}}{32 * 8 * \text{TSWUTC[7:0]}}$$

For example: if a external 32.768KHz crystal is used and TSWUTC = 0x80,
Then the wake-up frequency of the touch key controller is 32768 / (32 * 8 * 0x80) = 1Hz, that is to wake up once every 1 second.

Note: If the wake-up frequency is set too fast, the wake-up time is not enough to complete a round of key scan, and the touch key controller will scan continuously and cannot enter the power saving mode.

21.5.4 Touch Key Control Register (TSCTRL)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TSCTRL	FB45H	TSGO	SINGLE	TSWAIT	TSWUCS	TSDCEN	TSWUEN	TSSAMP[1:0]	

TSGO: Touch key controller start control bit

0: Touch key controller does not work.

1: Touch key controller start to work.

SINGLE: Single scan mode control bit

0: Repeat scan mode

1: Single scan mode. If TSGO=1, the hardware will clear TSGO to 0 automatically to terminate the scan after completing a round of key scan.

Note: IF TSGO = 1 and TSRT is not 0, it means that the touch button controller and LED driver share GPIO and time-sharing and multiplexing. At this time, the SINGLE control bit is invalid.

TSWAIT: Touch key controller waiting for control

0: touch key controller repeats scanning automatically

1: After completing one round of scanning, the TSIF is set to 1 by the hardware. At this time, the touch key controller will pause scanning until the TSIF flag is cleared to 0 and start the next round of scanning.

TSWUCS: clock source selection of touch key controller in low power mode

0: The clock source of the touch key controller in the low power mode is the internal 32K IRC.

1: The clock source of touch key controller in low power mode is external 32K crystal.

TSDCEN: 16-bit digital comparator inside the touch key controller control bit

0: disable 16-bit digital comparator inside the touch key controller

1: enable 16-bit digital comparator inside the touch key controller

Note: If the internal digital comparator of TSU is enabled, the TSIF will be set to 1 only when the touch sensing data result {TSDATAH, TSDATAL} is less than the threshold {TSTHHx, TSTHLx} set for the corresponding channel. This function is used to wake up the CPU by touch in low power mode.

TSWUEN: touch key controller low power wake up enable bit

0: disable touch key controller low power wake up function

1: enable touch key controller low power wake up function. After enabled, when the MCU enters the power-down state, it enters the touch-key low-power wake-up MCU mode immediately. In this mode, the low-power timing control circuit inside the touch key controller will enable the TSU periodically to perform key touch scanning. The duty control is used to maintain extremely low average current. TSWUEN is only effective when the MCU enters power-down mode and TSIF is 0.

Note: There are two 32K oscillators in this chip, one is an external 32K crystal, and the other is an internal IRC32K oscillator. In the case of non-STOP-mode, the internal IRC32K enable mechanism is simply using XFR: IRC32KCR [7], the external X32K enable mechanism is simply using XFR: X2KCR [7]. In the case of STOP-mode, the internal IRC32K enable mechanism is that in addition to XFR: IRC32KCR [7] must be set to 1, SFR: ENWKT or XFR: TSWUEN must also be set to 1, the external X32K enable mechanism is that in addition to XFR : X32KCR [7] must be set to 1, XFR: TSWUEN must also be set to 1. The key point to emphasize is that TSWUEN also plays the role of enabling or disabling the 32K oscillator in the case of STOP-mode.

TSSAMP[1:0]: single touch channel repeats scan times setting bits

TSSAMP [1:0]	Repeat scan times
00	once
01	2 times
10	3 times
11	4 times

Note: The interrupt flag TSIF will only be set when the number of scans of the same key reaches the configuration of TSSAMP. At this time, the average value of the results is written in {TSDATAH, TSDATAL}. However, if any overflow occurs, the hardware will set TSDOV to 1. With TSWKEN enabled, the average value must be less than the threshold to cause the interrupt flag TSIF to be set to 1 to wake the CPU.

21.5.5 Touch Key Status Register 1 (TSSTA1)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TSSTA1	FB46H	LEDWK	-	-	-	TSWKCHN[3:0]			

LEDWK: working state of touch key controller and LED driver in time-sharing operation

0: The LED driver is in waiting state and the touch key controller is in working state.

1: The LED driver is in working state and the touch key controller is in waiting state.

TSWKCHN [3:0]: touch channel scan status

TSWKCHN [3:0]	Touch channel scan status
0000	Touch channel 0 is being scanned
0001	Touch channel 1 is being scanned
0010	Touch channel 2 is being scanned
0011	Touch channel 3 is being scanned
0100	Touch channel 4 is being scanned
0101	Touch channel 5 is being scanned
0110	Touch channel 6 is being scanned
0111	Touch channel 7 is being scanned
1000	Touch channel 8 is being scanned
1001	Touch channel 9 is being scanned
1010	Touch channel 10 is being scanned
1011	Touch channel 11 is being scanned
1100	Touch channel 12 is being scanned
1101	Touch channel 13 is being scanned
1110	Touch channel 14 is being scanned
1111	Touch channel 15 is being scanned

21.5.6 Touch Key Status Register 2 (TSSTA2)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TSSTA2	FB47H	TSIF	TSDOV	-	-	TSDNCHN[3:0]			

TSIF: touch key channel scanning completion flag (cleared by writing 1 using software)

0: Scan has not completed.

1: When the number of scans set by TSSAMP is completed, TSIF is set by the hardware, and TSIF can request an interrupt to the CPU. If it is in the low-power wake-up mode, the scan data result value is lower than the set threshold at the same time, then TSIF will be set to 1.

Note: TSIF can only be set by hardware. Software cannot set TSIF to 1. It is important to note that

writing 1 to TSIF in software clears TSIF to 0, and writing 0 to TSIF in software has no effect. If TSWAIT = 1 and TSIF is 1, the TSU is in the pause and wait state. The next key scan will be continued after the CPU is busy finished and clears the TSIF to 0.

TSDOV: Key scan data overflow flag (cleared by writing 1 using software)

0: The key scan data does not overflow, the scan data is less than or equal to 0xFFFF.

1: The key scan data overflows, and the scan data is greater than 0xFFFF. At this time, extreme software configurations (such as TSVR) or system hardware must be adjusted to avoid overflow. TSDOV can only be set to 1 by hardware. It is cleared by software writing 1 to it. There is no effect on it if software writing 0 to it.

TSDNCHN [3:0]: Touch channel completion status

TSDNCHN [3:0]	Touch channel completion status
0000	Scanning of touch channel 0 is completed
0001	Scanning of touch channel 1 is completed
0010	Scanning of touch channel 2 is completed
0011	Scanning of touch channel 3 is completed
0100	Scanning of touch channel 4 is completed
0101	Scanning of touch channel 5 is completed
0110	Scanning of touch channel 6 is completed
0111	Scanning of touch channel 7 is completed
1000	Scanning of touch channel 8 is completed
1001	Scanning of touch channel 9 is completed
1010	Scanning of touch channel 10 is completed
1011	Scanning of touch channel 11 is completed
1100	Scanning of touch channel 12 is completed
1101	Scanning of touch channel 13 is completed
1110	Scanning of touch channel 14 is completed
1111	Scanning of touch channel 15 is completed

21.5.7 Touch Key Time Control Register (TSRT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TSRT	FB48H								

The TSRT register is used to configure the touch key controller and the LED driver to work in time-sharing.

If TSRT is not 00, the touch keycontroller and LED driver are in the time-sharing mode. The length of working time for the touch key controller is TSRT * T_{LED}. (Please refer to the LED driver description section for T_{LED})

21.5.8 Touch Key Data Registers (TSDAT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TSDATH	FB49H	TSDAT[15:8]							
TSDATL	FB4AH	TSDAT[7:0]							

TSDAT[15:0]: Data scanned by touch key

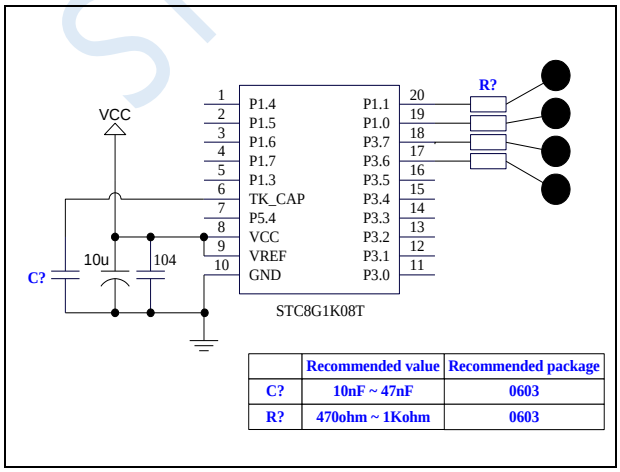
21.5.9 Touch Key Threshold Registers (TSTH)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TSTH00H	FB50H	TSTH00[15:8]							
TSTH00L	FB51H	TSTH00[7:0]							
TSTH01H	FB52H	TSTH01[15:8]							
TSTH01L	FB53H	TSTH01[7:0]							
TSTH02H	FB54H	TSTH02[15:8]							

TSTH02L	FB55H	TSTH02[7:0]
TSTH03H	FB56H	TSTH03[15:8]
TSTH03L	FB57H	TSTH03[7:0]
TSTH04H	FB58H	TSTH04[15:8]
TSTH04L	FB59H	TSTH04[7:0]
TSTH05H	FB5AH	TSTH05[15:8]
TSTH05L	FB5BH	TSTH05[7:0]
TSTH06H	FB5CH	TSTH06[15:8]
TSTH06L	FB5DH	TSTH06[7:0]
TSTH07H	FB5EH	TSTH07[15:8]
TSTH07L	FB5FH	TSTH07[7:0]
TSTH08H	FB60H	TSTH08[15:8]
TSTH08L	FB61H	TSTH08[7:0]
TSTH09H	FB62H	TSTH09[15:8]
TSTH09L	FB63H	TSTH09[7:0]
TSTH10H	FB64H	TSTH10[15:8]
TSTH10L	FB65H	TSTH10[7:0]
TSTH11H	FB66H	TSTH11[15:8]
TSTH11L	FB67H	TSTH11[7:0]
TSTH12H	FB68H	TSTH12[15:8]
TSTH12L	FB69H	TSTH12[7:0]
TSTH13H	FB6AH	TSTH13[15:8]
TSTH13L	FB6BH	TSTH13[7:0]
TSTH14H	FB6CH	TSTH14[15:8]
TSTH14L	FB6DH	TSTH14[7:0]
TSTH15H	FB6EH	TSTH15[15:8]
TSTH15L	FB6FH	TSTH15[7:0]

TSTHn[15:0]: Touch key scan data threshold. After the digital comparator is enabled, TSIF will be set to 1 by hardware only when the scan data is below this threshold.

21.6 Basic Reference Circuit and Precautions



	Recommended value	Recommended package
C?	10nF ~ 47nF	0603
R?	470ohm ~ 1Kohm	0603

Note: In the reference circuit diagram, C? is the sensitivity adjustment capacitor for touch keys, and R? is the ESD protection resistor. In PCB layout, C? and R? must be as close as possible to the IC pins.

22 LED Driver

Product	LED
STC8G1K08 family	
STC8G1K08-8Pin family	
STC8G1K08A family	
STC8G2K64S4 family	
STC8G2K64S2 family	
STC8G1K08T family	●
STC15H2K64S4 family	

An LED driver is integrated in STC8G series of microcontrollers.

The LED driver circuit includes a timing controller, 8 COM output pins and 8 SEGMENT output pins. Each pin has a corresponding register enable bit, which can independently control whether the pin is enabled or not. Pins that are not enabled can be used as pins for GPIO or other functions.

The LED driver supports three modes: common cathode, common anode, common cathode / common anode. At the same time, it can select 1/8 ~ 8/8 duty/cycle to adjust the gray scale. So only software is needed to adjust the LED and digital LED brightness.

After power-on reset, the enable bit LEDON is 0 and the LED driver is turned off. Set LEDON to 1 to enable the LED driver. If LEDMODE= 00, the driver works in the common cathode mode. At this time, the selected COM outputs a low level, the selected SEGMENTs to light LED output high level. Therefore, the forward bias of the LED between the two points of SEGMENT and COM turns on and lights up. Similarly, if LEDMODE = 01, the driver works in the common anode mode. At this time, the selected COM outputs a high level, and the selected SEGMENTs to light the LED output low level. The forward bias of the LED lights up. If LEDMODE = 10, the driver works in the common cathode / common anode time-sharing drive mode. The COM level is low and high-level time-sharing. The principle of LED lighting is the same as common cathod and common anode.

In the common cathode mode and the common anode mode, the display RAM address is independent. The location of display RAM address in the common cathode / common anode time-sharing mode is also read independently.

22.1 Internal Structure Diagram of LED Driver

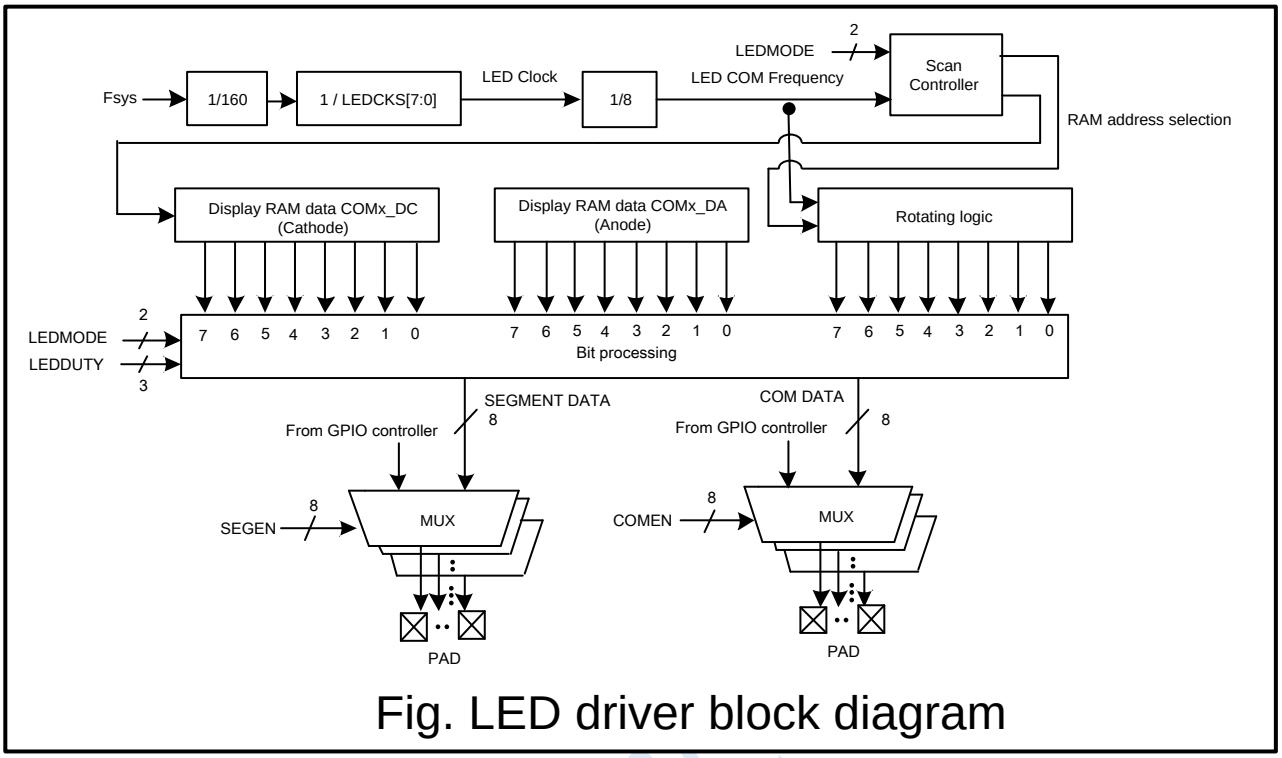


Fig. LED driver block diagram

22.2 Registers Related to LED Driver

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
COMEN	COM Enable Register	FB00H	C7EN	C6EN	C5EN	C4EN	C3EN	C2EN	C1EN	C0EN	0000,0000
SEGEN	SEG Enable Register	FB01H	S7EN	S6EN	S5EN	S4EN	S3EN	S2EN	S1EN	S0EN	0000,0000
LEDCTRL	LED Control Register	FB02H	LEDON	-	LEDMODE[1:0]		-	LEDDUTY[2:0]			0000,0000
LEDCKS	LED Clock Divide Register	FB03H									0000,0001
COM0_DA_L	Common Anode Mode Disply	FB10H									0000,0000
COM1_DA_L	Common Anode Mode Disply	FB11H									0000,0000
COM2_DA_L	Common Anode Mode Disply	FB12H									0000,0000
COM3_DA_L	Common Anode Mode Disply	FB13H									0000,0000
COM4_DA_L	Common Anode Mode Disply	FB14H									0000,0000
COM5_DA_L	Common Anode Mode Disply	FB15H									0000,0000
COM6_DA_L	Common Anode Mode Disply	FB16H									0000,0000
COM7_DA_L	Common Anode Mode Disply	FB17H									0000,0000
COM0_DA_H	Common Anode Mode Disply	FB18H									0000,0000
COM1_DA_H	Common Anode Mode Disply	FB19H									0000,0000
COM2_DA_H	Common Anode Mode Disply	FB1AH									0000,0000
COM3_DA_H	Common Anode Mode Disply	FB1BH									0000,0000
COM4_DA_H	Common Anode Mode Disply	FB1CH									0000,0000
COM5_DA_H	Common Anode Mode Disply	FB1DH									0000,0000
COM6_DA_H	Common Anode Mode Disply	FB1EH									0000,0000
COM7_DA_H	Common Anode Mode Disply	FB1FH									0000,0000
COM0_DC_L	Common Cathode Mode Disply	FB20H									0000,0000
COM1_DC_L	Common Cathode Mode Disply	FB21H									0000,0000
COM2_DC_L	Common Cathode Mode Disply	FB22H									0000,0000
COM3_DC_L	Common Cathode Mode Disply	FB23H									0000,0000
COM4_DC_L	Common Cathode Mode Disply	FB24H									0000,0000
COM5_DC_L	Common Cathode Mode Disply	FB25H									0000,0000
COM6_DC_L	Common Cathode Mode Disply	FB26H									0000,0000
COM7_DC_L	Common Cathode Mode Disply	FB27H									0000,0000
COM0_DC_H	Common Cathode Mode Disply	FB28H									0000,0000
COM1_DC_H	Common Cathode Mode Disply	FB29H									0000,0000
COM2_DC_H	Common Cathode Mode Disply	FB2AH									0000,0000
COM3_DC_H	Common Cathode Mode Disply	FB2BH									0000,0000
COM4_DC_H	Common Cathode Mode Disply	FB2CH									0000,0000
COM5_DC_H	Common Cathode Mode Disply	FB2DH									0000,0000

COM6_DC_H	Common Cathode Mode Disply	FB2EH		0000,0000
COM7_DC_H	Common Cathode Mode Disply	FB2FH		0000,0000

22.2.1 COM Enable Register (COMEN)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
COMEN	FB00H	C7EN	C6EN	C5EN	C4EN	C3EN	C2EN	C1EN	C0EN

CnEN: COMn enable control bit (n＝0~7)

0: disable COMn, keep GPIO function

1: enable COMn, the corresponding I/O outputs the driving waveform of COM when LEDON = 1.

22.2.2 SEG Enable Register (SEGEN)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
SEGEN	FB01H	S7EN	S6EN	S5EN	S4EN	S3EN	S2EN	S1EN	S0EN

SnEN: SEGn enable control bit (n＝0~7)

0: disable SEGn, keep GPIO function

1: enable SEGn, the corresponding I/O outputs the driving waveform of SEG when LEDON = 1.

22.2.3 LED Control Register (LEDCTRL)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
LEDCTRL	FB02H	LEDON	-	LEDMODE[1:0]		-	LEDDUTY[2:0]		

LEDON: LED driver enable control bit

0: disable LED driver

1: enable LED driver.

LEDMODE[1:0]: LED drive mode

LEDMODE[1:0]	Drive mode
00	Common cathod mode
01	Common anode mode
10	Common cathod / common anode mode
11	Reserved

LEDDUTY[2:0]: LED grayscale adjustment

LEDDUTY[2:0]	LED Duty/cycle	LED brightness
000	8/8	100%
001	7/8	87.5%
010	6/8	75%
011	5/8	62.5%
100	4/8	50%
101	3/8	37.5%
110	2/8	25%
111	1/8	12.5%

22.2.4 LED Clock Divide Register (LEDCKS)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
LEDCKS	FB03H								

LEDCKS: LED clock division control

$$\text{LED working frequency} = \frac{\text{SYSclk}}{160 * \text{LEDCKS[7:0]}}$$

If the value of the register LEDCKS is set too large, it will cause the LED to flicker. Generally, if the LED refresh frequency is greater than or equal to 75Hz, there will be no obvious flicker.

$$\text{suggested value: } \frac{\text{SYSclk}}{160 * 8 * N_{COM} * \text{LEDCKS[7:0]}} \geqslant 75\text{Hz}$$

N_{COM} is the number of COMs enabled, if it is a common cathode/common anode mode, it is twice the number of COMs.

For example, if the operating frequency of the microcontroller is 11.0592MHz, the working mode of the LED is common cathode/common anode mode, COMEN is set to 0FFH, even if there are 8 COMs, 11059200/160/8/16 / LEDCKS ⩾ 75, LEDCKS ⩽ 7.2, so LEDCKS is recommended to be set to 7.

$$\text{LED wOrking period } (T_{LED}) = \frac{1}{\text{LED working frequency}}$$

22.2.5 LED data registers of common anode mode (COMn_DA)

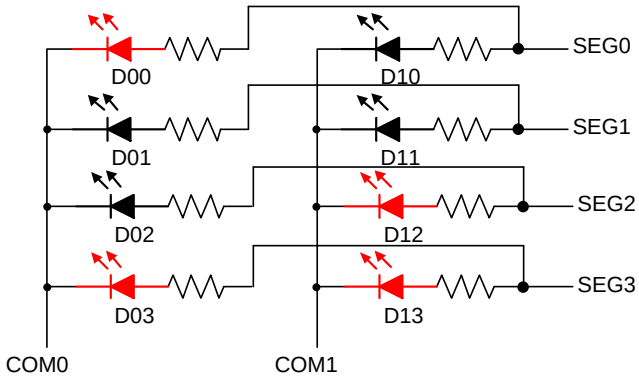
Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
COM0_DA_L	FB10H								
COM1_DA_L	FB11H								
COM2_DA_L	FB12H								
COM3_DA_L	FB13H								
COM4_DA_L	FB14H								
COM5_DA_L	FB15H								
COM6_DA_L	FB16H								
COM7_DA_L	FB17H								
COM0_DA_H	FB18H								
COM1_DA_H	FB19H								
COM2_DA_H	FB1AH								
COM3_DA_H	FB1BH								
COM4_DA_H	FB1CH								
COM5_DA_H	FB1DH								
COM6_DA_H	FB1EH								
COM7_DA_H	FB1FH								

22.2.6 LED data registers of common cathod mode (COMn_DC)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
COM0_DC_L	FB20H								
COM1_DC_L	FB21H								
COM2_DC_L	FB22H								
COM3_DC_L	FB23H								
COM4_DC_L	FB24H								
COM5_DC_L	FB25H								
COM6_DC_L	FB26H								
COM7_DC_L	FB27H								
COM0_DC_H	FB28H								
COM1_DC_H	FB29H								
COM2_DC_H	FB2AH								
COM3_DC_H	FB2BH								
COM4_DC_H	FB2CH								
COM5_DC_H	FB2DH								
COM6_DC_H	FB2EH								
COM7_DC_H	FB2FH								

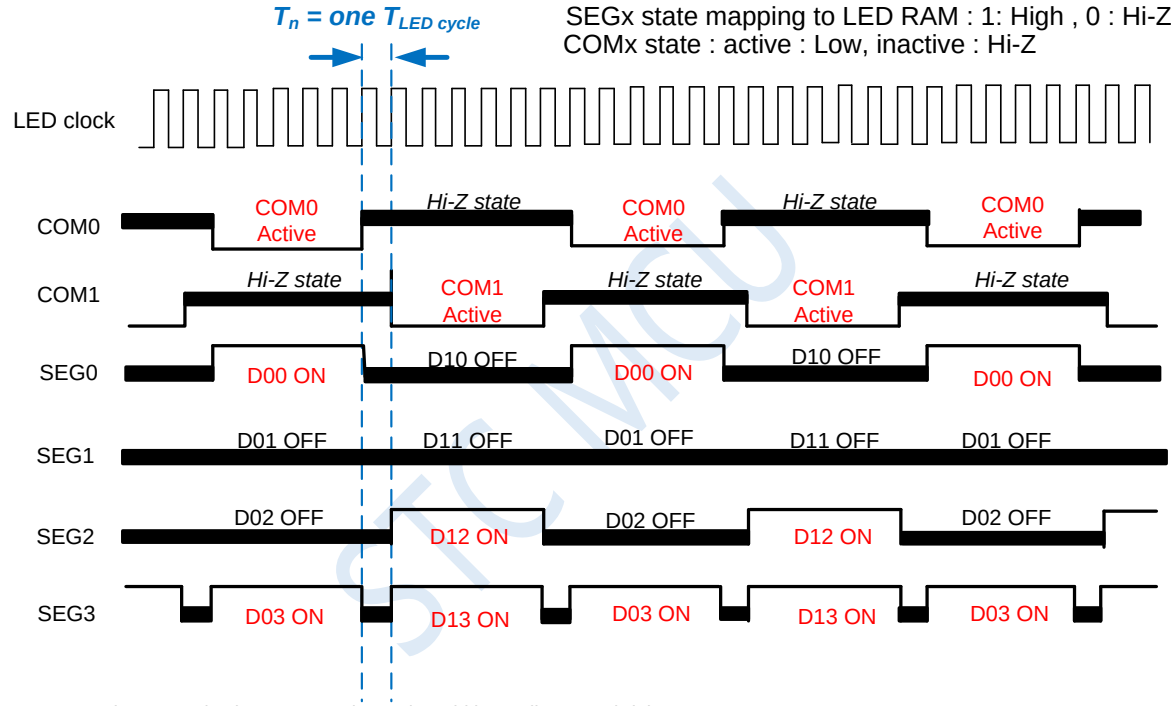
STC MCU

22.3 LED Common Cathod Mode (LEDMODE = 00)



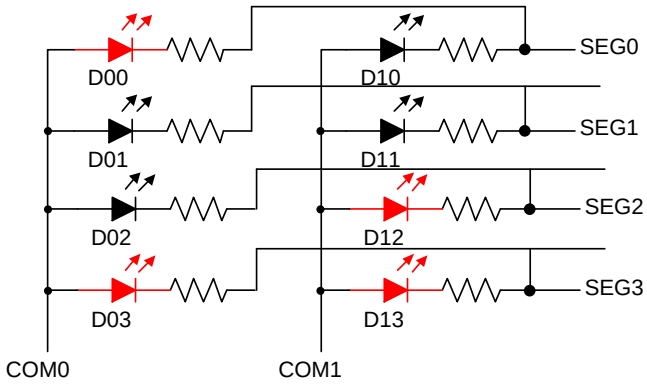
LED RAM 1 : ON 0: OFF

	D00	D01	D02	D03
COM0	1	0	0	1
	D10	D11	D12	D13
COM1	0	0	1	1



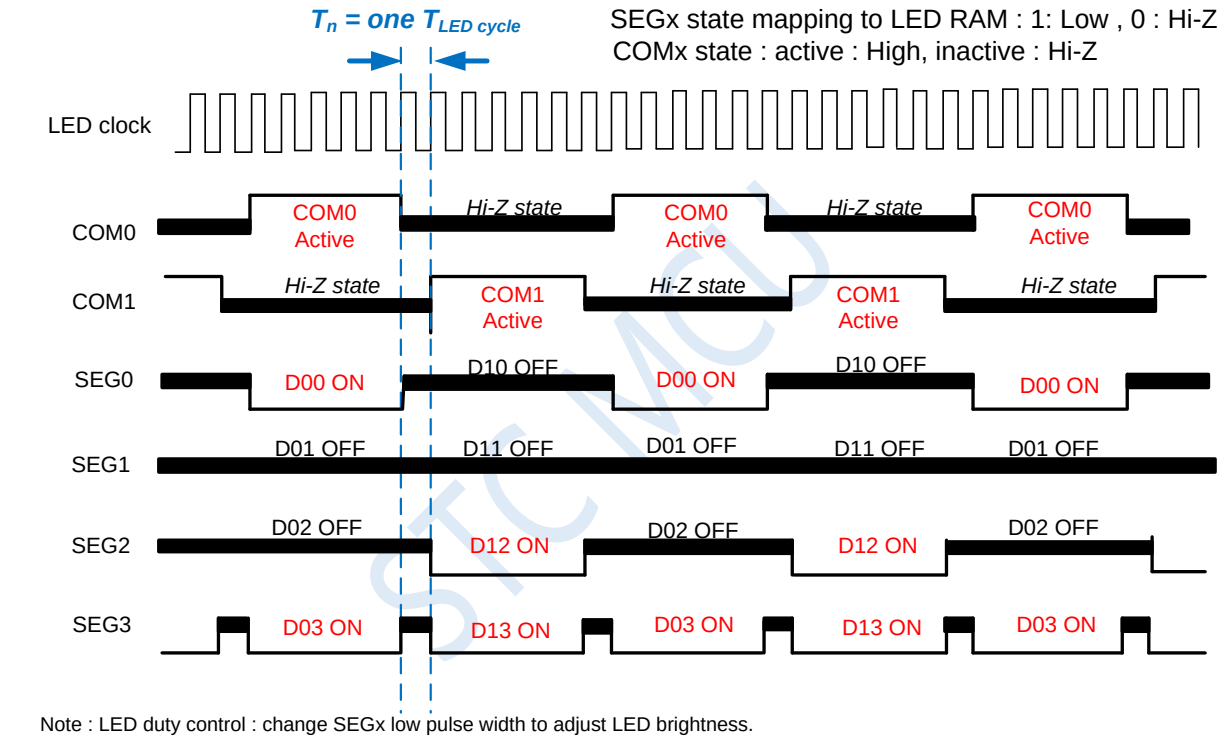
Note : LED duty control : change SEGx low pulse width to adjust LED brightness.

22.4 LED Common Anode Mode (LEDMODE = 01)



LED RAM 1 : ON 0: OFF

	D00	D01	D02	D03
COM0	1	0	0	1
	D10	D11	D12	D13
COM1	0	0	1	1

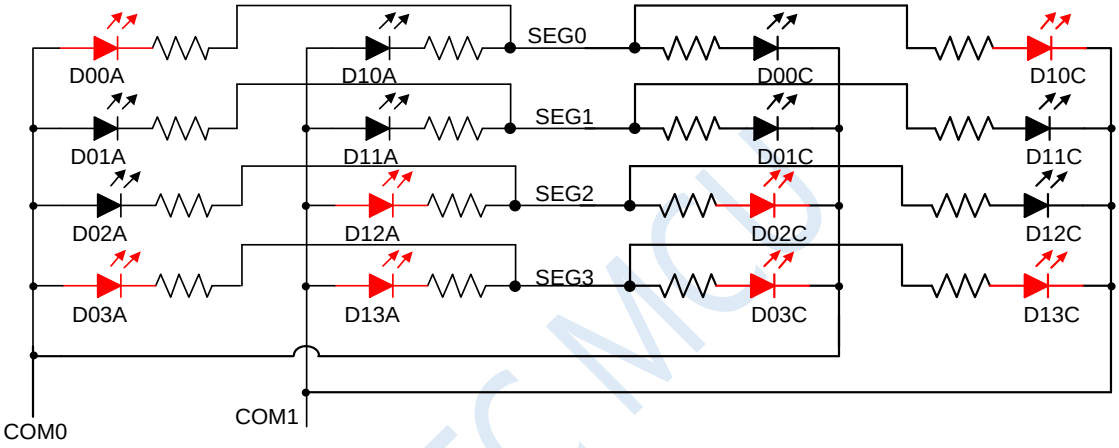


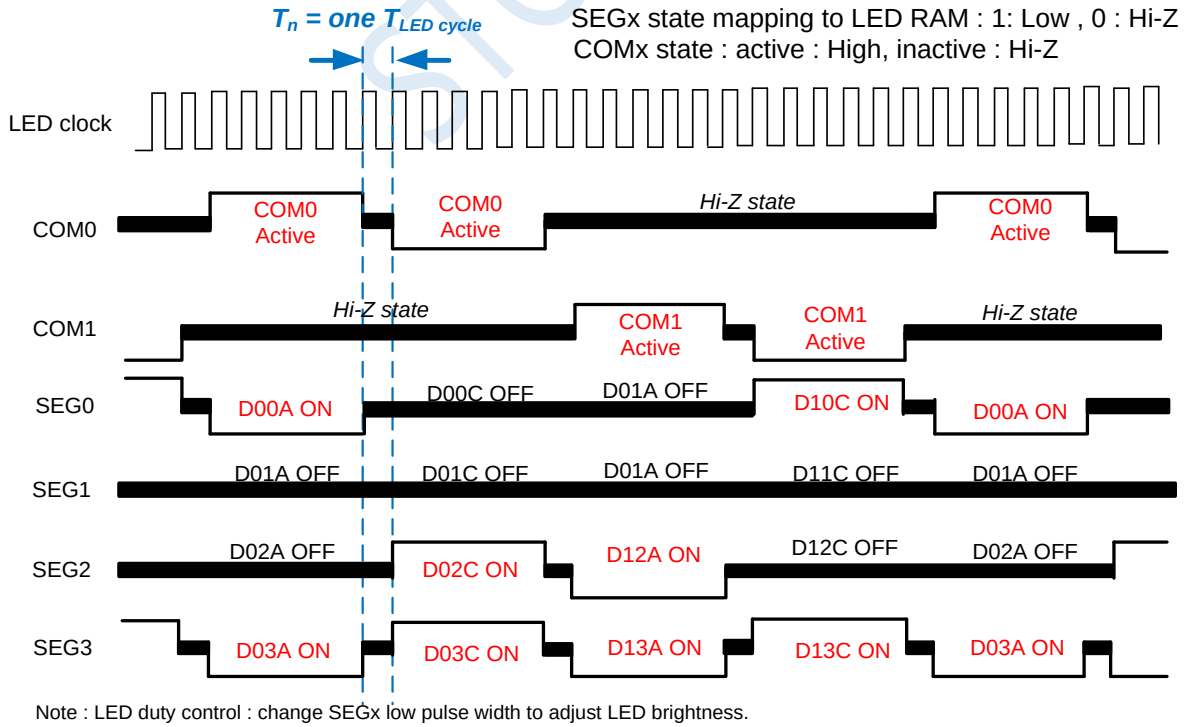
Note : LED duty control : change SEGx low pulse width to adjust LED brightness.

22.5 LED Common Cathode/ Common Anode Mode (LEDMODE = 10)

LED RAM 1 : ON 0: OFF

	D00A	D01A	D02A	D03A
COM0	1	0	0	1
	D00C	D01C	D02C	D03C
COM0	0	0	1	1
	D10A	D11A	D12A	D13A
COM1	0	0	1	1
	D10C	D11C	D12C	D13C
COM1	1	0	0	1





Note : LED duty control : change SEGx low pulse width to adjust LED brightness.

22.6 Touch Keys and LED Driver Share I/O

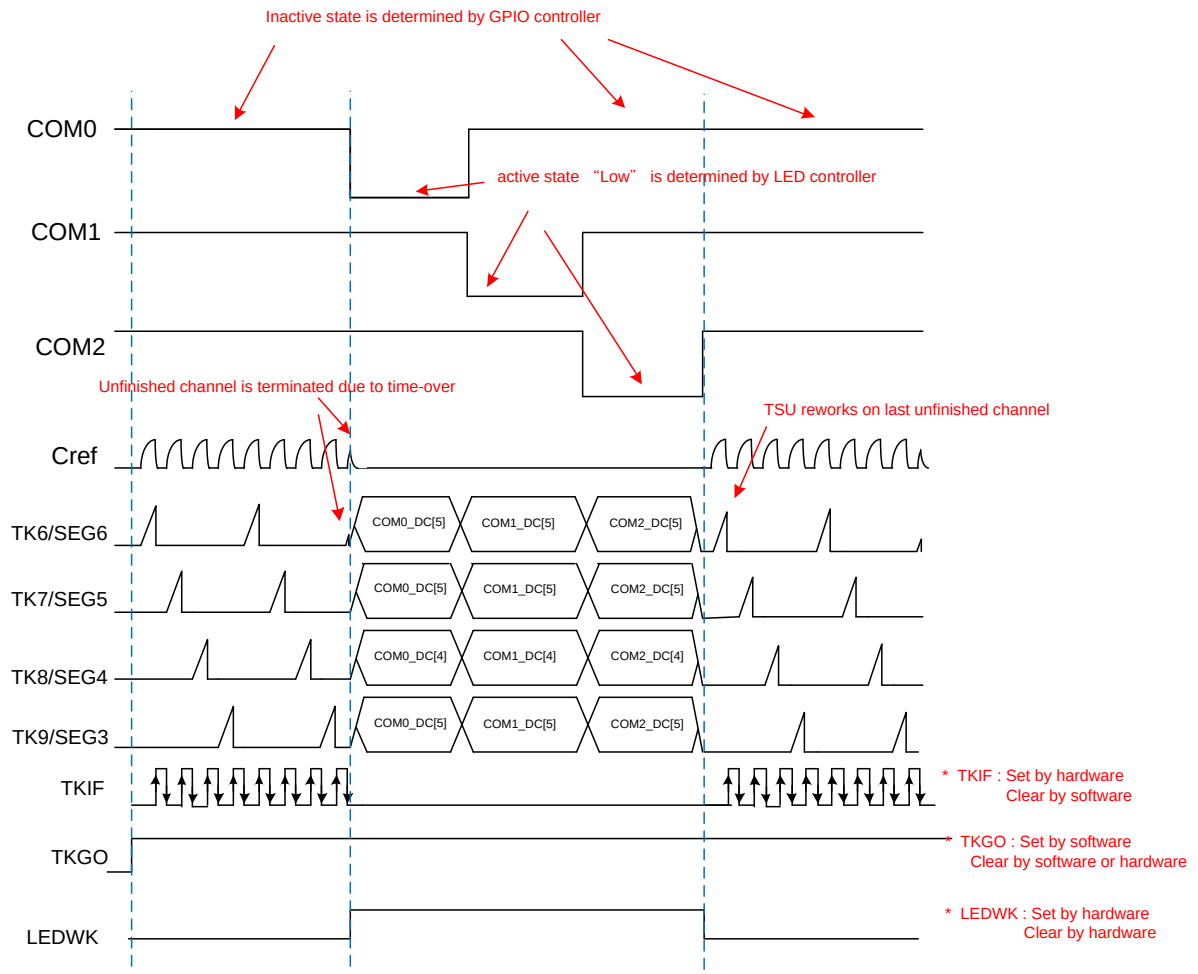


Fig. Touch/LED share timing diagram

(Common Cathode, only COM 0~2 and TK6~TK9 are enabled)

Steps:

1. Select the touch key channel to be scanned. The registers are TSCHEN1 and TSCHEN2.
2. Configure switching frequency SCR [2: 0], discharge time DT [2: 0] and select internal comparator reference voltage TSVR [1: 0].
3. Configure TSSAMP [1: 0] to determine the number of repeated scans of the same channel. If the CPU task is heavy, configure TSWAIT to use the TSIF state to delay the next channel scan.
4. If necessary, configure TSDCEN to enable the internal digital comparison function.
5. Set the TSRT content. If the TSRT content is not 0x00, the LED driver time sharing multitasking function is not enabled.
6. Configure the SEGEN and COMEN registers.
7. Configure LEDCKS to determine the time length of each COM action. This needs to be considered with the TKRT register to calculate the LED refresh rate.

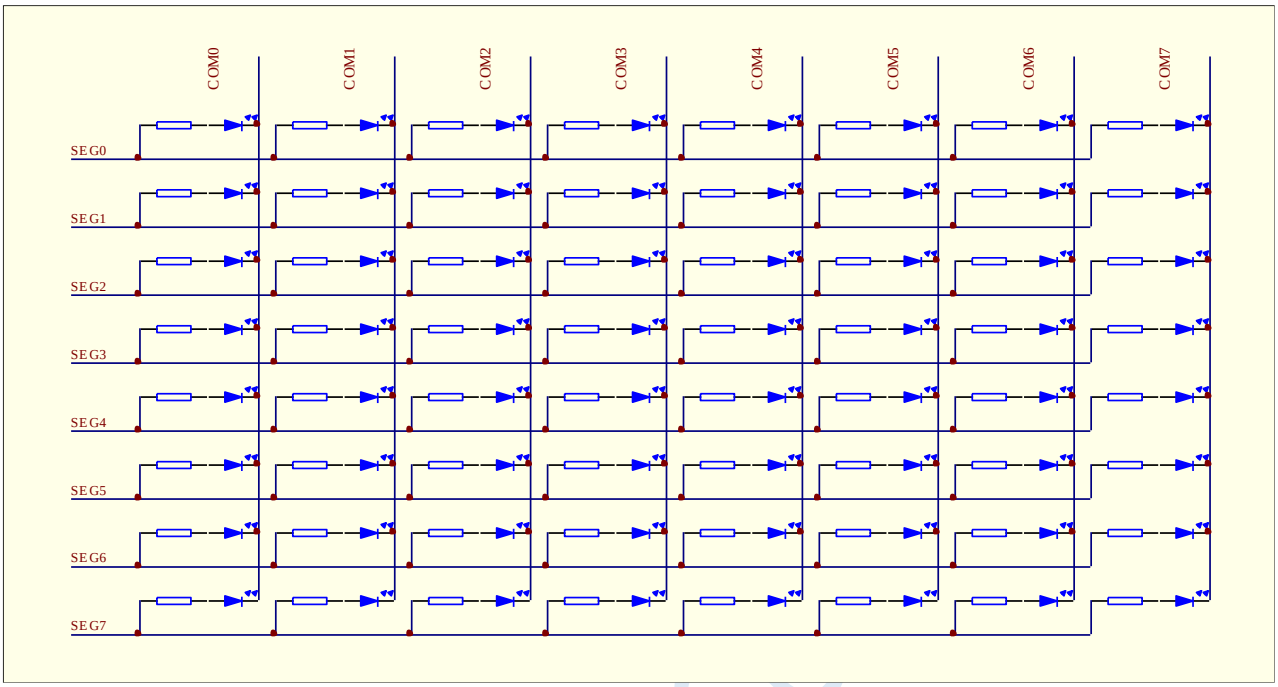
8. Configure LEDMODE [1: 0] and LEDDUTY according to the operating mode and brightness of the LED required.
9. Write data to the data registers COMx_DC and COM_DA of the LED.
10. Set TSGO=1, the touch key controller starts scanning.
    a) You can read TSWKCHN [3: 0] using software to know which channel is currently being scanned. After a channel is scanned, the hardware will set TSIF to 1 and the completed channel number will be written into TSDNCHN [3: 0]. If an overflow occurs, TSOV will also be set to 1. Software should read these registers to decide what to do next. TSIF and TSOV can only be set by hardware and cleared by software.
    b) When switching to the LED working time, the LEDWK bit is read as 1, which is used to determine whether the touch key controller is working or the LED driver is working.
    c) Software continuously updates LED data register according to actual needs.
11. If you want to terminate the touch key and LED time-sharing multi-tasking mode, you need to write TSGO = 0, then the multi-task mode is terminated, the touch key and LED are not working. And the control right of the I/O port returns to the GPIO controller.

Note:

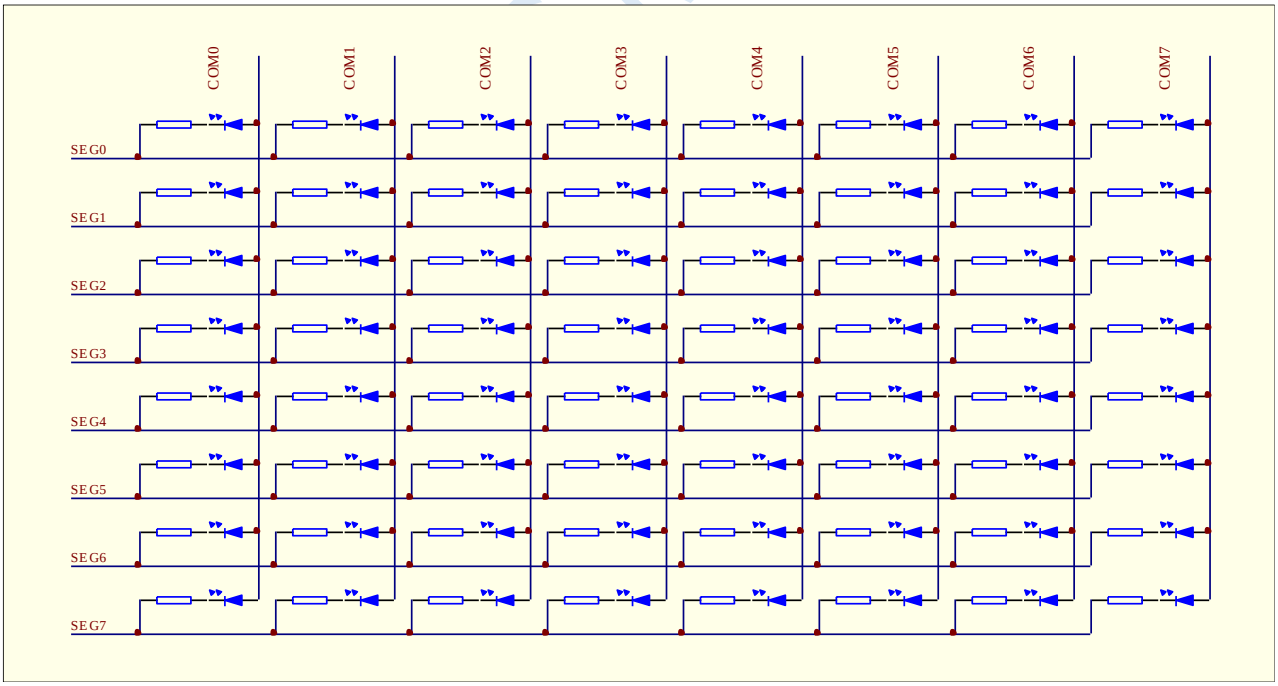
1. The SINGLE control bit is invalid in touch key and LED time-sharing multi-tasking mode. Only software writing TSGO can control the module on and off.
2. When the touch key scan time is terminated and turned to LED action time, the last key is almost incomplete. At this time, the hardware processing will not produce TSIF, and not update the registers related to the touch data, but the hardware will remember this channel number. The incomplete channel will be re-scanned and start a new round of scanning after the LED action period ends and turn to the touch key scanning time.
3. The circuit diagram of LED and touch multiplexing is as above. It should be noted that the LED's light-emitting color is different, and the equivalent capacitance of the LED represented will be different. The larger the capacitance, the more unfavorable the touch button, and the sensitivity will decrease. Generally speaking, the capacitance value of the red LED may be 35pF, but the yellow light will be as high as 100pF. At this time, if you want to increase the sensitivity of the touch button, you can connect a 1N4148 diode in series. The capacitance of 1N4148 is only 4pF, and 1N4148 string a 100pF yellow LED, the parasitic capacitance of the key will be slightly smaller than 4pF.

STC MCU

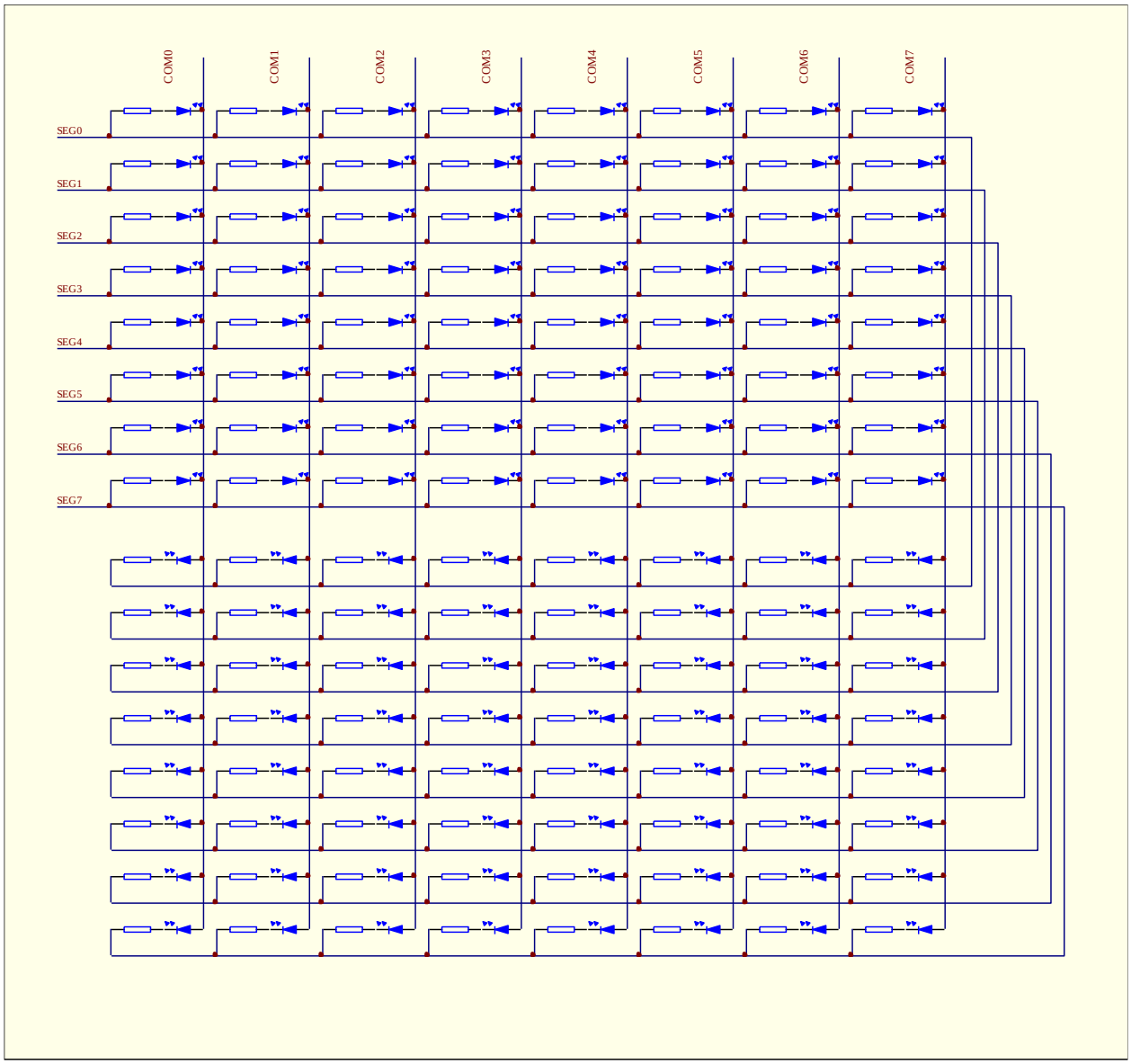
22.7 Reference Circuit of Common Cathode Mode



22.8 Reference Circuit of Common Anode Mode

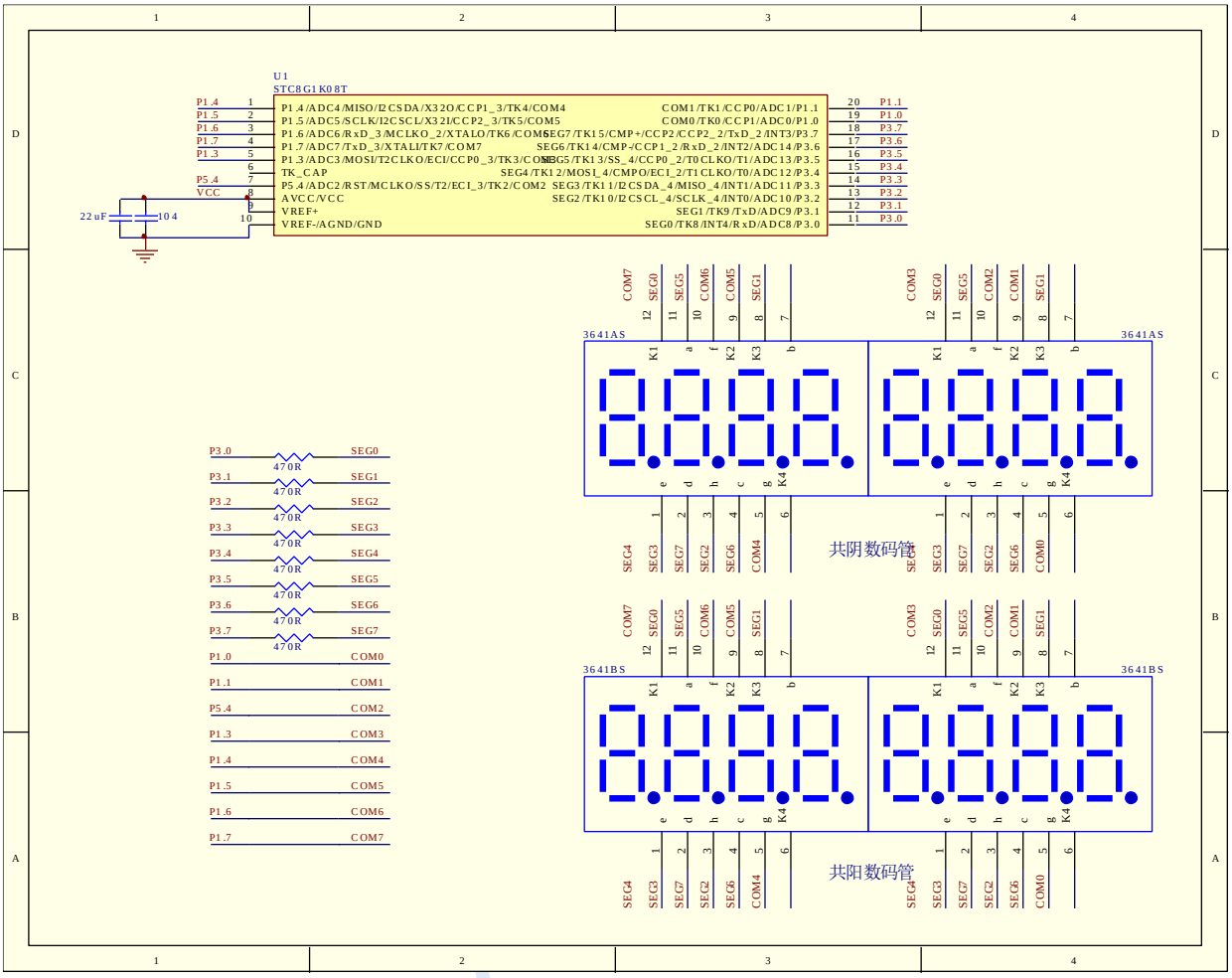


22.9 Reference Circuit of Common Cathode/Common Anode Mode



22.10 Example Routines

22.10.1 Common cathode/common anode mode drives 16 7-segment digital LEDs



C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;
```

```
#define COMEN       (*(unsigned char volatile xdata *)0xfb00)
#define SEGENL      (*(unsigned char volatile xdata *)0xfb01)
#define LEDCTR      (*(unsigned char volatile xdata *)0xfb03)
#define LEDCKS      (*(unsigned char volatile xdata *)0xfb04)
#define COM0_DA     (*(unsigned char volatile xdata *)0xfb10)
#define COM1_DA     (*(unsigned char volatile xdata *)0xfb11)
#define COM2_DA     (*(unsigned char volatile xdata *)0xfb12)
#define COM3_DA     (*(unsigned char volatile xdata *)0xfb13)
#define COM4_DA     (*(unsigned char volatile xdata *)0xfb14)
#define COM5_DA     (*(unsigned char volatile xdata *)0xfb15)
#define COM6_DA     (*(unsigned char volatile xdata *)0xfb16)
#define COM7_DA     (*(unsigned char volatile xdata *)0xfb17)
#define COM0_DC     (*(unsigned char volatile xdata *)0xfb20)
#define COM1_DC     (*(unsigned char volatile xdata *)0xfb21)
#define COM2_DC     (*(unsigned char volatile xdata *)0xfb22)
#define COM3_DC     (*(unsigned char volatile xdata *)0xfb23)
#define COM4_DC     (*(unsigned char volatile xdata *)0xfb24)
#define COM5_DC     (*(unsigned char volatile xdata *)0xfb25)
#define COM6_DC     (*(unsigned char volatile xdata *)0xfb26)
#define COM7_DC     (*(unsigned char volatile xdata *)0xfb27)

char code PATTERN[16] =
{
    0x3f,       //0
    0x06,       //1
    0x5b,       //2
    0x4f,       //3
    0x66,       //4
    0x6d,       //5
    0x7d,       //6
    0x27,       //7
    0x7f,       //8
    0x6f,       //9
    0x77,       //A
    0x7c,       //b
    0x39,       //C
    0x5E,       //d
    0x79,       //E
    0x71,       //F
};

void main()
{
    P1M0 = 0xff;
    P1M1 = 0x00;
    P3M0 = 0xff;
    P3M1 = 0x00;
    P5M0 = 0x10;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    COMEN = 0xff;                   //Enable COM0~COM7
    SEGENL = 0xff;                  //Enable SEG0~SEG7
    LEDCTRL = 0x20;                 //LED driver common cathode/common anode mode
    LEDCKS = 7;                     // Set LED refresh rate

    COM0_DA = PATTERN[0];           // Set LED display content
    COM1_DA = PATTERN[1];
```

STC MCU

```
    COM2_DA = PATTERN[2];
    COM3_DA = PATTERN[3];
    COM4_DA = PATTERN[4];
    COM5_DA = PATTERN[5];
    COM6_DA = PATTERN[6];
    COM7_DA = PATTERN[7];

    COM0_DC = PATTERN[8];
    COM1_DC = PATTERN[9];
    COM2_DC = PATTERN[10];
    COM3_DC = PATTERN[11];
    COM4_DC = PATTERN[12];
    COM5_DC = PATTERN[13];
    COM6_DC = PATTERN[14];
    COM7_DC = PATTERN[15];

    LEDCTRL |= 0x80;                    // Start the LED driver

    P_SW2 &= ~0x80;

    while (1);
}
```

STC MCU

23 Enhanced Dual Data Pointer

Two 16-bit data pointers are integrated in STC8G series of microcontrollers. The data pointers can be increased or decreased automatically by the program control, and they can be switched automatically.

23.1 Related special function registers

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
DPL	Data pointer low byte register	82H									0000,0000
DPH	Data pointer high byte register	83H									0000,0000
DPL1	2nd Data pointer low byte	E4H									0000,0000
DPH1	2nd Data pointer high byte	E5H									0000,0000
DPS	DPTR Selection Register	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL	0000,0xx0
TA	DPTR Timing control register	AEH									0000,0000

23.1.1 1st 16-bit Data Pointer Registers (DPTR0)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DPL	82H								
DPH	83H								

DPL is Data pointer 0 low byte.

DPH is Data pointer 0 high byte.

The combination of DPL and DPH is the first 16-bit data pointer register DPTR0.

23.1.2 2nd 16-bit Data Pointer Registers (DPTR1)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DPL1	E4H								
DPH1	E5H								

DPL1 is Data pointer 1 low byte.

DPH1 is Data pointer 1 low byte.

The combination of DPL1 and DPH1 is the second 16-bit data pointer register DPTR1.

23.1.3 DPTR control register (DPS)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DPS	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL

ID1: DPTR1 auto-increment or auto-decrement mode control bit

0: DPTR1 auto-increment mode

1: DPTR1 auto-decrement mode

ID0: DPTR0 auto-increment or auto-decrement mode control bit

0: DPTR0 auto-increment mode

1: DPTR0 auto-decrement mode

TSL: DPTR0/DPTR1 auto-switch control bit (invert SEL automatically)

0: DPTR0/DPTR1 auto switch is disabled.

1: DPTR0/DPTR1 auto switch is enabled.

If the TSL bit is set, the SEL bit will be inverted automatically after the relevant instruction is

executed.

Instructions related to TSL include:

```
MOV     DPTR,#data16
INC     DPTR
MOVC    A,@A+DPTR
MOVX    A,@DPTR
MOVX    @DPTR,A
```

AU1/AU0: Enable DPTR1 / DPTR0 Automatic increment / decrement control bit

0: disable Automatic increment / decrement function

1: enable Automatic increment / decrement function

Note: In write-protect mode, AU0 and AU1 can not be enabled individually. AU0 will be enabled automatically if AU1 is enabled. If AU0 is enabled alone, there is no effect to AU1. To enable AU1 or AU0 independently, the TA register must be used to trigger the DPS protection mechanism. For more detail, please refer to the description of the TA register. In addition, DPTR0 / DPTR1 will be incremented / decremented automatically only after executing the following three instructions.

```
MOVC    A,@A+DPTR
MOVX    A,@DPTR
MOVX    @DPTR,A
```

SEL: DPTR register select bit.

0: Default. DPTR0 is selected as current Data pointer.

1: DPTR1 is selected as current Data pointer.

The selection of current DPTR using SEL is valid for the following instructions,

```
MOV     DPTR,#data16
INC     DPTR
MOVC    A,@A+DPTR
MOVX    A,@DPTR
MOVX    @DPTR,A
JMP     @A+DPTR
```

23.1.4 DPTR control register (TA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TA	AEH								

AU1 and AU0 in the DPS register is write-protected by TA register. Since the program can not write AU1 and AU0 separately, TA register must be used to trigger enabling AU1 or AU0 independently. TA is a write-only register.

The following steps must be executed if you need to enable AU1 or AU0 separately.

```
CLR     EA          ; disable interrupt (it is necessary to disable interrupt)
MOV     TA,#0AAH    ; write the trigger command sequence 1
                    ; any other instructions can not be here
MOV     TA,#55H     ; write the trigger command sequence 2
                    ; any other instructions can not be here
MOV     DPS,#xxH    ; Write-protection is temporarily disabled,
                    ; and any value can be written to the DPS
                    ; DSP enters the write-protected mode again
```

```
SETB    EA              ; enable interrupt if necessary
```

STC MCU

23.2 Example Routines

23.2.1 Example Routine 1

Copy 4 bytes of data stored in program space 1000H to 1003H in reverse to 0100H to 0103H of the extended RAM, that is,

C:1000H -> X:0103H
C:1001H -> X:0102H
C:1002H -> X:0101H
C:1003H -> X:0100H

Assembly code

```
;Operating frequency for test is 11.0592MHz

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     DPS,#00100000B          ; enable TSL and select DPTR0
            MOV     DPTR,#1000H     ; write 1000H to DPTR0, and then select DPTR1 as current DPTR
            MOV     DPTR,#0103H             ; write 0103H to DPTR1
            MOV     DPS,#10111000B          ;set DPTR1 in auto decreasing mode,
                                    ;DPTR0 in auto increasing mode,enable TSL, AU0 and AU1,
                                            ;select DPTR0 as the current DPTR
            MOV     R7,#4                   ; set the counter of copies
COPY_NEXT:
            CLR     A                       ;
```

```
            MOVC    A,@A+DPTR           ; Read data from the program space indicated by DPTR0,
                                        ; When done, DPTR0 increments automatically and select DPTR1 as the current DPTR
            MOVX    @DPTR,A             ; write the content of ACC to XDARA indicated by DPTR1,
                                         ; When done, DPTR1 decrements automatically and select DPTR0 as the current DPTR
            DJNZ    R7,COPY_NEXT        ;

            SJMP    $

            END
```

23.2.2 Example Routine 2

Send the data stored in the extended RAM 0100H to 0103H to P0 port successively.

Assembly code

```
;Operating frequency for test is 11.0592MHz

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            CLR     EA              ; disable interrupt
            MOV     TA,#0AAH        ; Write DPS write-protection trigger command 1
            MOV     TA,#55H         ; Write DPS write-protection trigger command 2
            MOV     DPS,#00001000B  ; set DPTR0 in increasing mode,enable AU0 independently,
                                    ;and select DPTR0

            SETB    EA              ; enable interrupt
            MOV     DPTR,#0100H     ; write 0100H to DPTR0
            MOVX    A,@DPTR         ; Read data from XRAM indicated by DPTR0,
```

```
                                ;and then DPTR0 increments automatically
MOV         P0,A                ; output the datum to Port0
MOVX        A,@DPTR             ; Read data from XRAM indicated by DPTR0,
                                ;and then DPTR0 increments automatically
MOV         P0,A                ; output the datum to Port0
MOVX        A,@DPTR             ; Read data from XRAM indicated by DPTR0,
                                ;and then DPTR0 increments automatically
MOV         P0,A                ; output the datum to Port0
MOVX        A,@DPTR             ; Read data from XRAM indicated by DPTR0,
                                ;and then DPTR0 increments automatically
MOV         P0,A                ; output the datum to Port0

SJMP        $

END
```

STC MCU

24 MDU16 Hardware 16-bit Multiplier and Divider

Product	MDU16
STC8G1K08 family	
STC8G1K08-8Pin family	●
STC8G1K08A family	●
STC8G2K64S4 family	●
STC8G2K64S2 family	●
STC8G1K08T family	
STC15H2K64S4 family	●

A hardware multiply / divide unit MDU16 is integrated in the STC8G series of microcontrollers. It supports the following operations,

data normalization (It takes 3 to 20 clocks of operation time)

logical left shift (It takes 3 to 18 clocks of operation time)

logical right shift (It takes 3 to 18 clocks of operation time)

16-bit multiply by 16-bit (It takes 10 clocks of operation time)

16-bit divide by 16-bit (It takes 9 clocks of operation time)

32-bit divide by 16-bit (It takes 17 clocks of operation time)

All operations are based on unsigned integer data types.

24.1 Registers Related to MDU16

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
MD3	MDU Data Register	FCF0H	MD3[7:0]								0000,0000
MD2	MDU Data Register	FCF1H	MD2[7:0]								0000,0000
MD1	MDU Data Register	FCF2H	MD1[7:0]								0000,0000
MD0	MDU Data Register	FCF3H	MD0[7:0]								0000,0000
MD5	MDU Data Register	FCF4H	MD5[7:0]								0000,0000
MD4	MDU Data Register	FCF5H	MD4[7:0]								0000,0000
ARCON	MDU Mode Control Register	FCF6H	MODE[2:0]			SC[4:0]					0000,0000
OPCON	MDU Operation Control Register	FCF7H	-	MDOV	-	-	-	-	RST	ENOP	0000,0000

24.1.1 Operand 1 Data Registers (MD0~MD3)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
MD3	FCF0H	MD3[7:0]							
MD2	FCF1H	MD2[7:0]							
MD1	FCF2H	MD1[7:0]							
MD0	FCF3H	MD0[7:0]							

24.1.2 Operand 2 Data Registers (MD4~MD5)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
MD5	FCF4H	MD5[7:0]							
MD4	FCF5H	MD4[7:0]							

32-bit divided by 16-bit division:

Dividend: {MD3,MD2,MD1,MD0}

Divisor: {MD5,MD4}

Quotient: {MD3,MD2,MD1,MD0}

Remainder: {MD5,MD4}

16-bit divided by 16-bit division:

Dividend: {MD1,MD0}

Divisor: {MD5,MD4}

Quotient: {MD1,MD0}

Remainder: {MD5,MD4}

16-bit multiplicated by 16-bit multiplication:

Multiplicand: {MD1,MD0}

Multiplier: {MD5,MD4}

Product: {MD3,MD2,MD1,MD0}

32-bit logical shift left / logical shift right

Operand: {MD3,MD2,MD1,MD0}

32-bit data normalization:

Operand: {MD3,MD2,MD1,MD0}

24.1.3 MDU Mode Control Register (ARCON)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
ARCON	FCF6H	MODE[2:0]			SC[4:0]				

MODE[2:0]: MDU mode selection

MODE[2:0]	Mode	Clocks	Desciption
1	Logical right shift	3～18	Shift the data in {MD3, MD2, MD1, MD0} right by SC [4: 0] bits, MD3 high-order complement 0
2	Logical left shift	3～18	Shift the data in {MD3, MD2, MD1, MD0} SC [4: 0] bits to the left, low-order complement of MD0
3	Data normalization	3～20	Perform logical left shift on the data in {MD3, MD2, MD1, MD0}, shift out all the high-order 0s of the data, make the highest bit of MD3 be 1, and the number of logical left shifts is recorded in SC [4: 0].
4	16-bit multiplication	10	{MD1,MD0}×{MD5,MD4}={MD3,MD2,MD1,MD0}
5	16-bit division	9	{MD1,MD0}÷{MD5,MD4}= {MD1,MD0}…{MD5,MD4}
6	32-bit division	17	{MD3,MD2,MD1,MD0}÷{MD5,MD4}= {MD3,MD2,MD1,MD0}…{MD5,MD4}
Others	Invalid		

SC[4:0]: Data shift bits

When MDU is in shift mode, SC is used to set the number of bits for left/right shift.

When MDU is in the data normalization mode, SC is the actual number of bits moved by the data after the data is normalized.

24.1.4 MDU Operation Control Register (OPCON)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
OPCON	FCF7H	-	MDOV	-	-	-	-	RST	ENOP

MDOV: MDU Overflow flag (read-only flag)

MDOV is set by hardware automatically in the following situations:

1. When the divisor is 0;

2. When the product of multiplication is greater than 0FFFFH;

When software writes OPCON.0 (EN) or writes ARCON, MDOV is cleared by the hardware automatically.

RST: Software resets the MDU multiplication and division unit. Writing 1 to it will trigger a software reset. It is cleared by the hardware automatically after the MDU reset is complete.

Note: The value of the ARCON register is cleared when software resets the MDU multiply and divide unit.

ENOP: MDU enable bit.

Writing 1 to this bit will trigger the MDU module to start calculation. When the MDU calculation is completed, ENOP is cleared to 0 by the hardware automatically. After setting ENOP to 1, the software can query ENOP cyclically. When ENOP changes from 1 to 0, the calculation is completed.

24.2 Miscellaneous talks about the application of MDU16 by netizens (provide ideas for reference only)

Netizen 1: "Data normalization is illustrated by the following simple example"

1. There is a 7-digit decimal precision datum: 0.0000123. Due to the limited data width, if you need to effectively use the bit width, you need to shift the previous datum to the left. For example, the datum is 0.123e-4 after the left shift, where the exponent -4 is stored in another register, the number of times the left shift recorded is the size of the recorded exponent. The original register datum is converted to 0.123. This frees up the bit width on the right side of the datum to ensure the accuracy of subsequent calculations. The above is just a simple description of the principle of normalization in decimal, and the principle of binary is the same. Among them, floating-point and fixed-point (integer) conversion must use the principle of normalization. If the exponents of the addition and subtraction of two floating-point numbers are not the same, they also need to be normalized (this process is called pairing). If the exponents of two floating-point numbers are very different, the problem of large numbers eating decimals will occur when adding and subtracting. For example, 0.123e+4-0.12e-4 = 0.123e+4-0.0000000012e+4 = 0.123e+4. The result is the subtracted number, because the exponents of the two floating-point numbers need to be exactly the same before the subtraction operation (For the order), you need to shift the floating-point number with the small exponent to make the exponent +4. But the data width is limited to 7 decimal places, the data on the right side of the number 0.0000000012e+4 will be truncated to 0.0000000e +4 = 0.

Netizen 2: "With regard to the MDU function of STC8G, I would like to share a little bit of my own experience. If there is something wrong, please criticize and advise and improve together."

1. Functions 1 and 2 are effective for reducing and expanding integer data. First, when performing double-operand operations, if the lengths of the two numbers are not the same, they need to be converted to the same length before the operation is performed. For example, if a 32-bit integer is multiplied by an 8-bit integer, 8 bits must be converted to 32 bits. Secondly, the result of AD sampling needs to be shifted when it is converted to the specified digit precision. Finally, for example, for network communication, certain bits of data need to be extracted for command analysis or data decomposition and synthesis, and displacement is very important. Since the 8051 only has an instruction to move 1 bit, multi-bit movement requires additional loop code and requires a lot of instruction cycles. Therefore, using MDU will be several times faster than 51 assembly instructions.

2. Function 3 is a necessary function for converting integers to floating-point numbers. For a full-precision 32-bit integer, it generally takes more than 100 instruction cycles to implement this function, so the MDU has a

relatively large increase in the speed of rotation. Since the output of AD equipment and various three-axis acceleration outputs are generally integers (such as 16-bit ones), real number operations and trigonometric functions must be performed. The output of integers must be converted to floating-point numbers, and every time This data type conversion must be performed to collect data, and the number of conversions required is a lot. For high-speed data acquisition and applications such as drone control, the overall performance improvement is considerable if DMU is used.

3. Function 6 is the necessary division function for fixed-point real number operations. Function 4 is the 16-bit x 16-bit multiplication operation corresponding to function 6 with 32-bit result. The most common application of function 6 is scale conversion in data processing. For example, for the integer conversion of a 10-digit AD collected with a reference voltage of 5 volts, the calculation formula for displaying the 2-digit fixed decimal point of a 3-digit digital tube is: N32=ADN *500/1023. At this time, as long as (1) send AD sample value AND to MX (DM1MD0), (2) send 500 to NX (MD5MD4), (3) execute function 4, the result is 32 bits, (4) send 1023 to NX (MD5MD4), (3) execute function 6, the 16-bit result is in MX, just get it back. Another common application is to draw dots and lines on dot matrix screens such as TFT, such as digital oscilloscopes, which require multiplication and division of coordinate transformation-first multiply to a 32-bit integer, and then divide by a 16-bit integer to get 16 bits result.

4. The combination of function 4 and function 6 is the hardware basis for implementing discrete convolution. If floating-point acceleration hardware is not used, the realization of the four arithmetic operations of floating-point numbers is an order of magnitude slower than the realization of the four arithmetic operations of integers. Therefore, the predecessors invented the method of using integer variables to achieve convolution. First of all, for example, when we commonly convert JPG image data to RGB image data or vice versa, Fourier transform is required. Since the length of image data is fixed (8 bits or 16 bits), discrete Fourier transform can be used To achieve, basically only 8-bit or 16-bit integer multiplication and a very small amount of 32-bit multiplication and division are used. In this way, our early digital cameras could be realized. Secondly, the common template processing in PS image processing also uses the two-dimensional matrix convolution method, which also requires a huge number of integers (8-bit image visual image size requires 16-bit and 32-bit intermediate calculation results). Adding calculations and using discrete convolution will greatly increase the computing speed. Therefore, the STC8 microcontroller with MDU can be used not only to collect and display images in real time, but also to process images in real time. Finally, artificial intelligence also involves a large number of vector and matrix operations, such as neural network convolution, which can be implemented with a combination of function 4 and function 6. MDU should be able to be applied in small intelligent scenes. Only to realize these functions, the cooperation of STC8's enhanced dual data pointer is needed, and a special knowledge structure is required, and a function library is specially compiled to provide users with use, in order to give full play to the huge advantages of STC8's MDU.

24.3 Example Routines

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"
```

```
#define   MD3U32          (*(unsigned long volatile xdata *)0xfcf0)
#define   MD3U16          (*(unsigned int volatile xdata *)0xfcf0)
#define   MD1U16          (*(unsigned int volatile xdata *)0xfcf2)
#define   MD5U16          (*(unsigned int volatile xdata *)0xfcf4)

#define   MD3             (*(unsigned char volatile xdata *)0xfcf0)
#define   MD2             (*(unsigned char volatile xdata *)0xfcf1)
#define   MD1             (*(unsigned char volatile xdata *)0xfcf2)
#define   MD0             (*(unsigned char volatile xdata *)0xfcf3)
#define   MD5             (*(unsigned char volatile xdata *)0xfcf4)
#define   MD4             (*(unsigned char volatile xdata *)0xfcf5)
#define   ARCON           (*(unsigned char volatile xdata *)0xfcf6)
#define   OPCON           (*(unsigned char volatile xdata *)0xfcf7)

sfr       P_SW2       =   0xBA;

//////////////////////////////////////////////////////////////////////////////
//16 bits by 16 bits
//////////////////////////////////////////////////////////////////////////////
unsigned long res;
unsigned int dat1, dat2;

P_SW2 |= 0x80;                  // Access to the extended register xsfr
MD1U16 = dat1;                  //dat1 is given by user
MD5U16 = dat2;                  //dat2 is given by user
ARCON = 4 << 5;                 //16 bit*16 bit, multiply mode
OPCON = 1;                      //start calculation
while((OPCON & 1) != 0);        //wait for the calculation complement
res = MD3U32;                   //32 bit result

//////////////////////////////////////////////////////////////////////////////
//32 bit divided by 16 bit
//////////////////////////////////////////////////////////////////////////////
unsigned long res;
unsigned long dat1;
unsigned int dat2;

P_SW2 |= 0x80;                  // Access to the extended register xsfr
MD3U32 = dat1;                  //dat1 is given by user
MD5U16 = data2;                 //dat2 is given by user
ARCON = 6 << 5;                 //32 bit/16bit, division mode
OPCON = 1;                      //start calculation
while((OPCON & 1) != 0);        // wait for the calculation complement
res = MD3U32;                   //32-bit quotient, 16-bit remainder in MD5U16

//////////////////////////////////////////////////////////////////////////////
// Shift left or right:
//////////////////////////////////////////////////////////////////////////////
unsigned long res;
unsigned long dat1;
unsigned char num;              // The number of bits to shift, given by the user

MD3U32 = dat1;                  //dat1 is given by the user
ARCON = (2 << 5) + num;         //32-bit left shift mode
//ARCON = (1 << 5) + num;       //32-bit right shift mode
OPCON = 1;                      // start calculation
while((OPCON & 1) != 0);        // wait for the calculation complement
res = MD3U32;                   //32 bit result
```

STC MCU

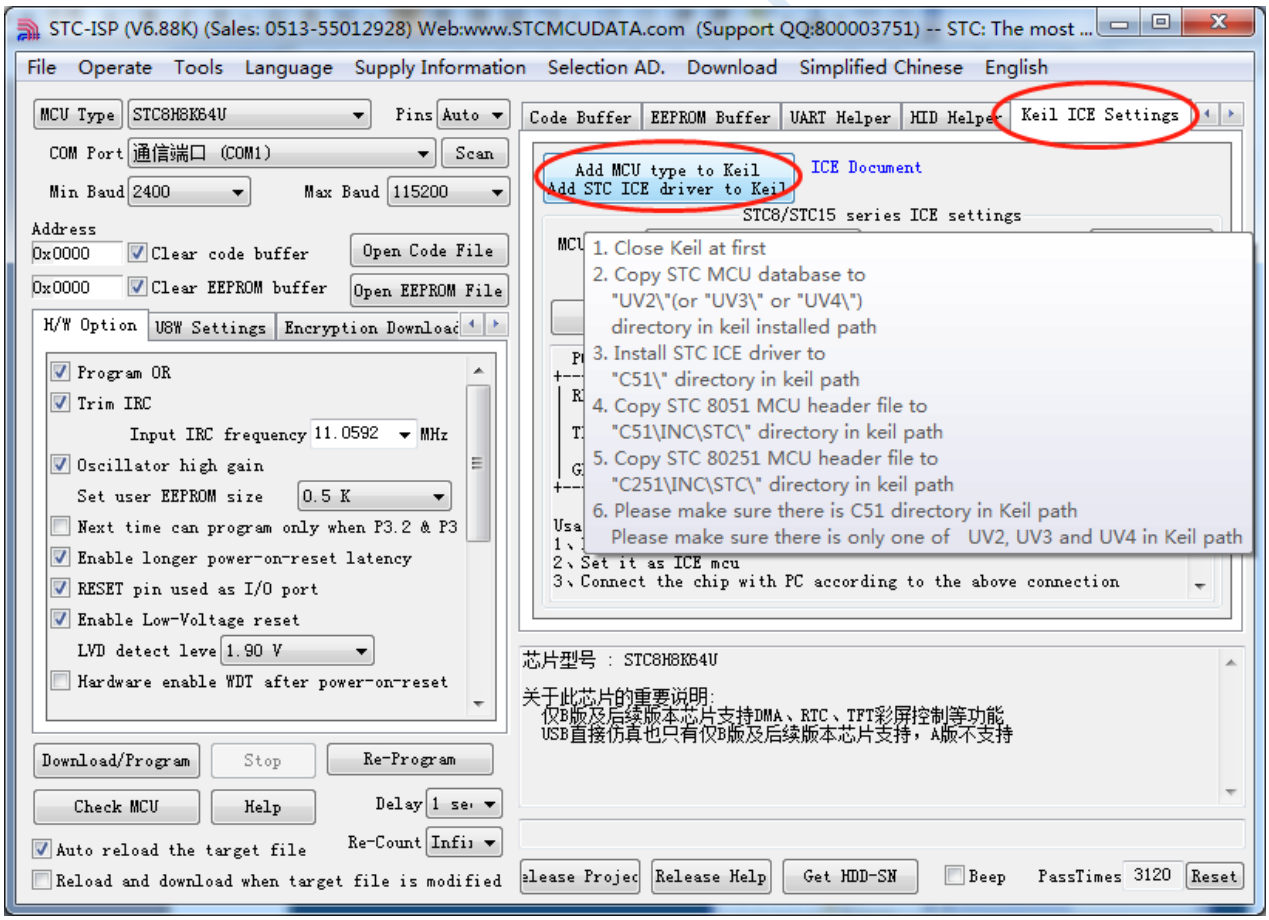
STC MCU

Appendix A STC Emulator User Guide

A: What kind of compiler/assembler should be used for STC MCU?
Q: Any old 8051 compiler/assembler can support it, and Keil C51 is popular now.

A: How to include header files in Keil environment
Q: After installing the driver and header files according to the steps shown below, select the corresponding STC MCU model when creating a new project, and directly use "#include <stc8g.h>" in the source file to complete the inclusion of the header file. If you choose Intel's 8052/87C52/87C54/87C58 or Philips' P87C52/P87C54/P87C58 to compile when creating a new project, the header file contains <reg51.h>, but the new STC special function registers need to be declared by the user.

1.Install Keil version of emulation driver.



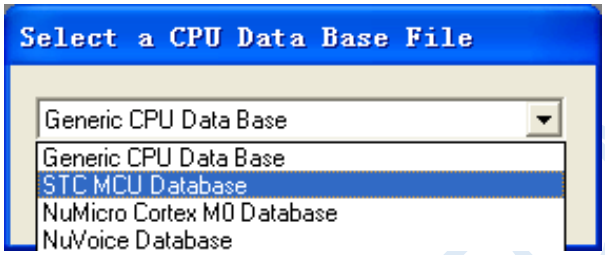
As shown above, select the "Keil Simulation Settings" page firstly, click "Add MCU model to Keil", and in the following directory selection window that appears, navigate to the installation directory of Keil (usually "C: \ Keil \") , After press "OK" button, the prompt message shown on the right in the following figure appears, indicating that the installation was successful. The STC Monitor51 emulation driver STCMON51.DLL will also be installed when adding the header file. The installation directory of the driver and header file is shown above.



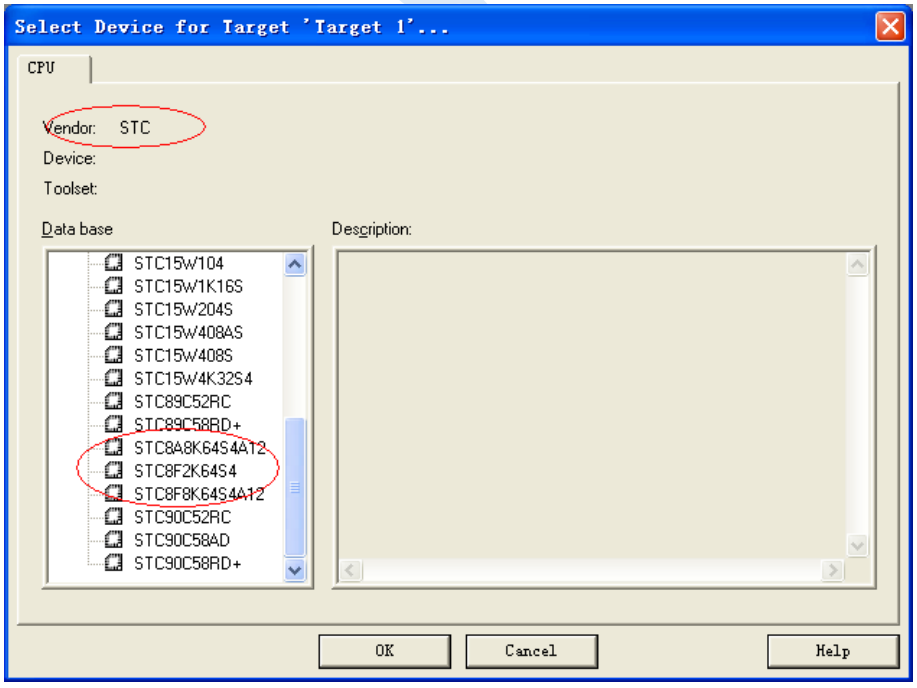


2. Create a project in Keil

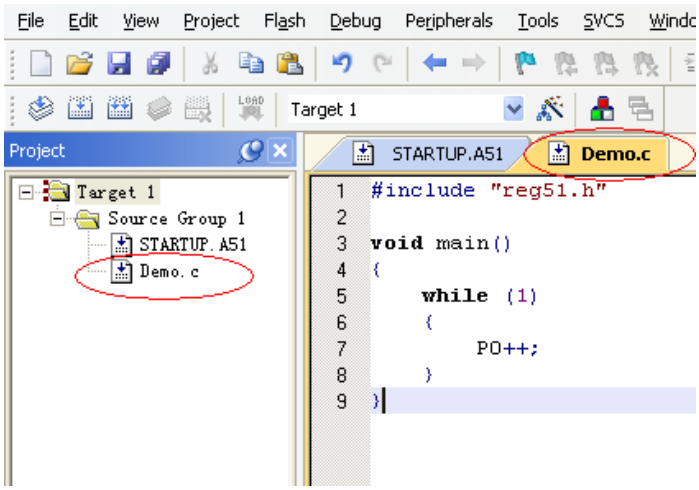
If the driver installation is successful in the first step, there will be an option of "STC MCU Database" in selecting the chip model when creating a new project in Keil as shown below.



Then select the responding MCU model from the list. Here we select the model of "STC8A8K64S4A12" and click "OK" to complete the selection.



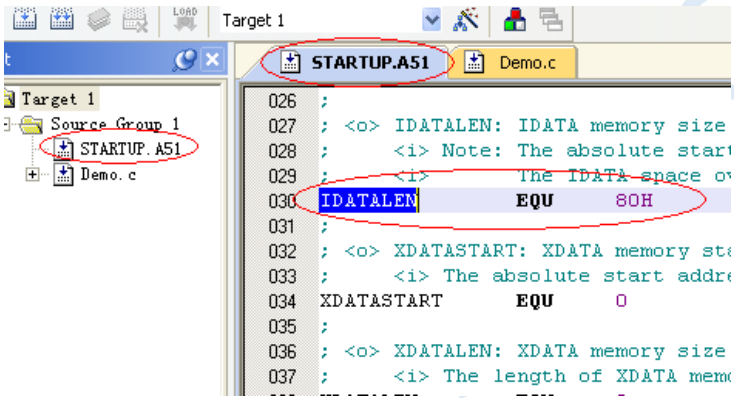
Add source code files to the project, as shown below:



Save the project. If there is no error while compiling the project, you can set the following projects.

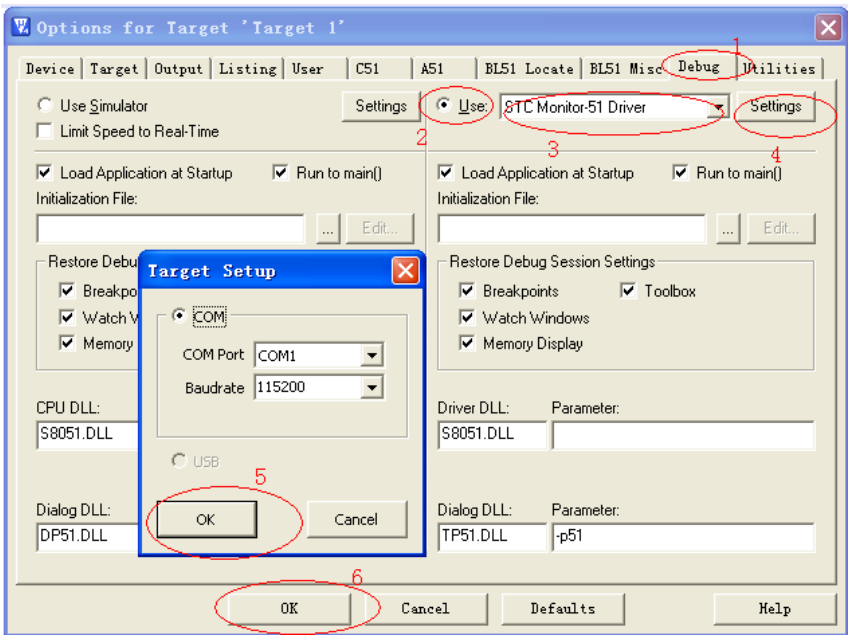
One additional note:

When a C language project is created and a startup file "STARTUP.A51" is added to the project, there is a macro definition named "IDATALEN", which is a macro used to define the size of the IDATA. The default value is 128, which is 80H in hexadecimal, and it is also the size of IDATA in the startup file that needs to be initialized to 0. Therefor if IDATA is defined as 80H, the code in STARTUP.A51 will initialize the RAM of 00-7F of IDATA to 0. Similarly, if IDATA is defined as 0FFH, the RAM of 00-FF of IDATA will be initialized to 0.



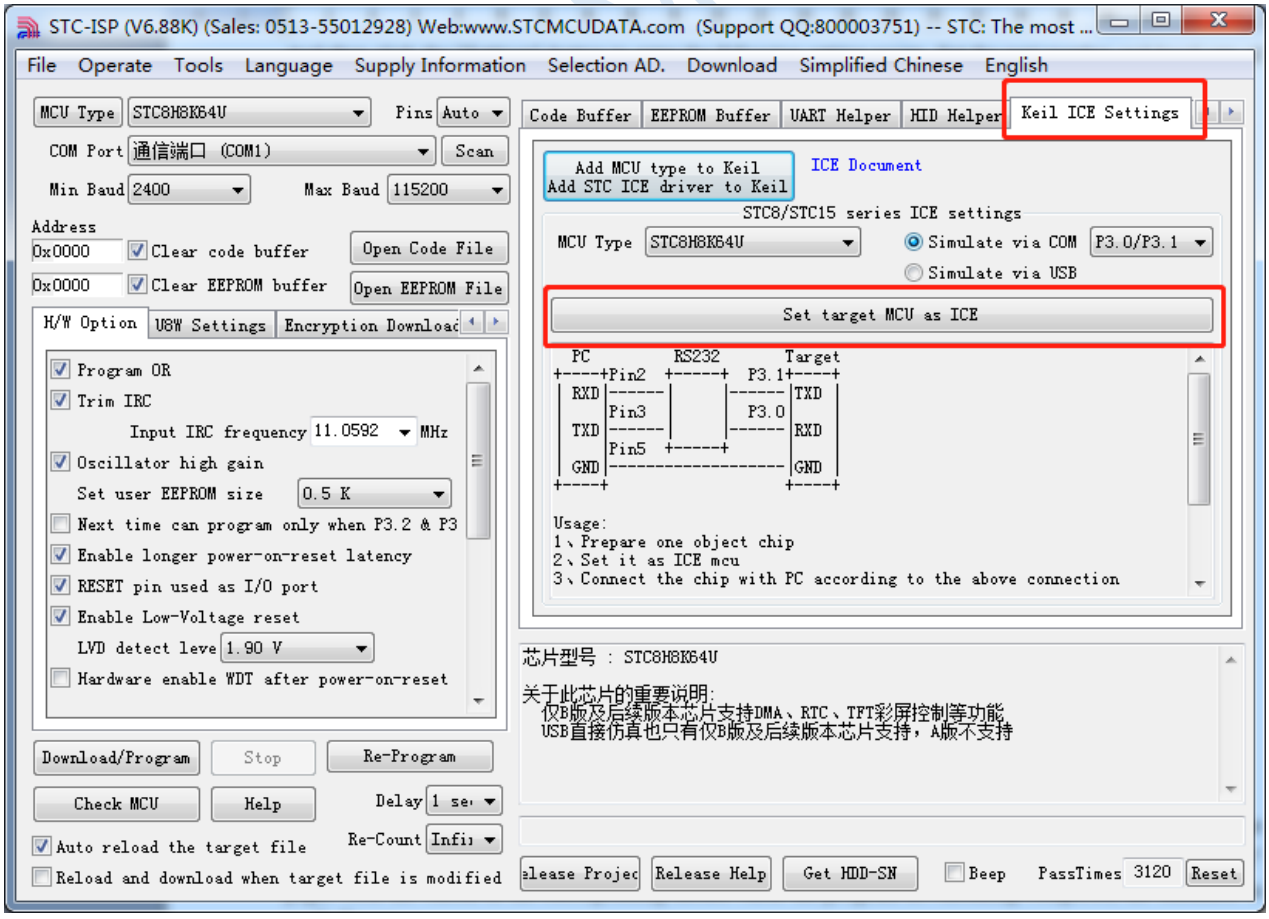
The IDATA size of the STC8 series of microcontrollers is 256 bytes (DATA of 00-7F and IDATA of 80H-FFH). Because the last 17 bytes of RAM have the ID number and related test parameters, if you need to use this part of the data in the program, you must not define IDATALEN as 256.

3. Project settings, select STC simulation driver.



As shown above, enter the project setting page firstly, select the "Debug" setting page, select the hardware emulation "Use…" on the right, and select "STC Monitor-51 Driver" in the emulation driver drop-down list. And then click the "Settings" button to enter the following setting screen. Set the port number and baud rate of the serial port. The baud rate is generally 115200. Then complete the setup.

4.Create simulation chip



Prepare a STC8A series or STC8F series chip, and connect it to the serial port of the computer through the download board. Then select the correct chip model as shown above, and enter the "Keil simulation settings"

page, click the button of the corresponding model. After the program downloading completes, the simulator is ready for use.

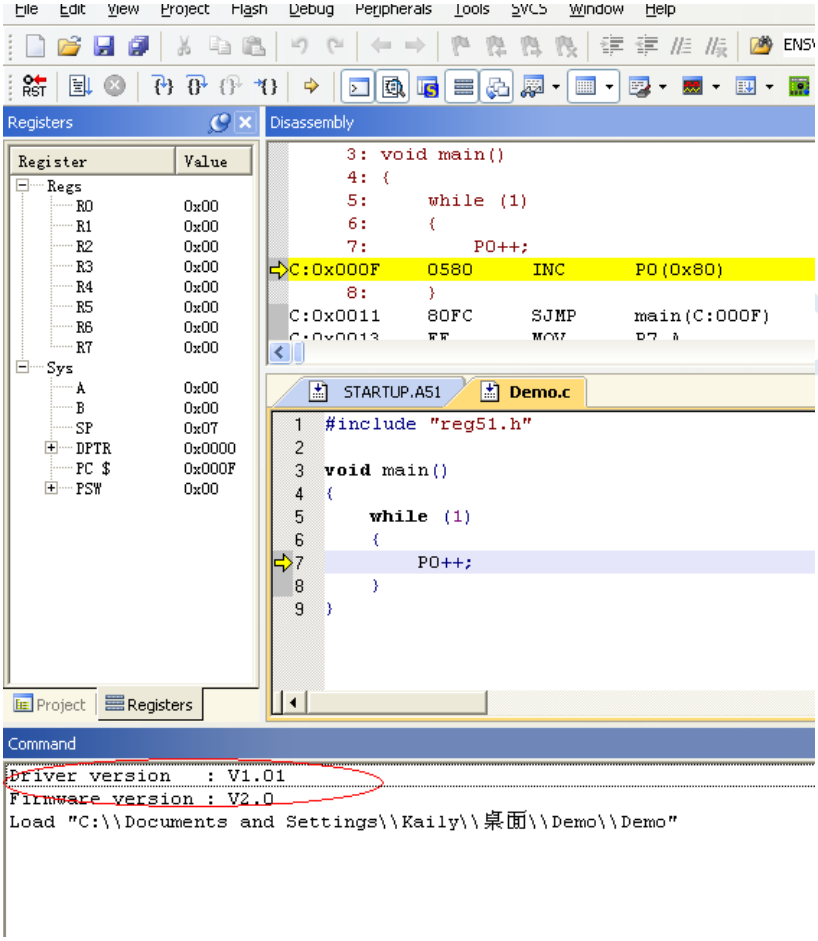
5.Start simulation

Connect the completed simulation chip to the computer through the serial port.

After compiling the project we created before without errors, press "Ctrl + F5" to start debugging.

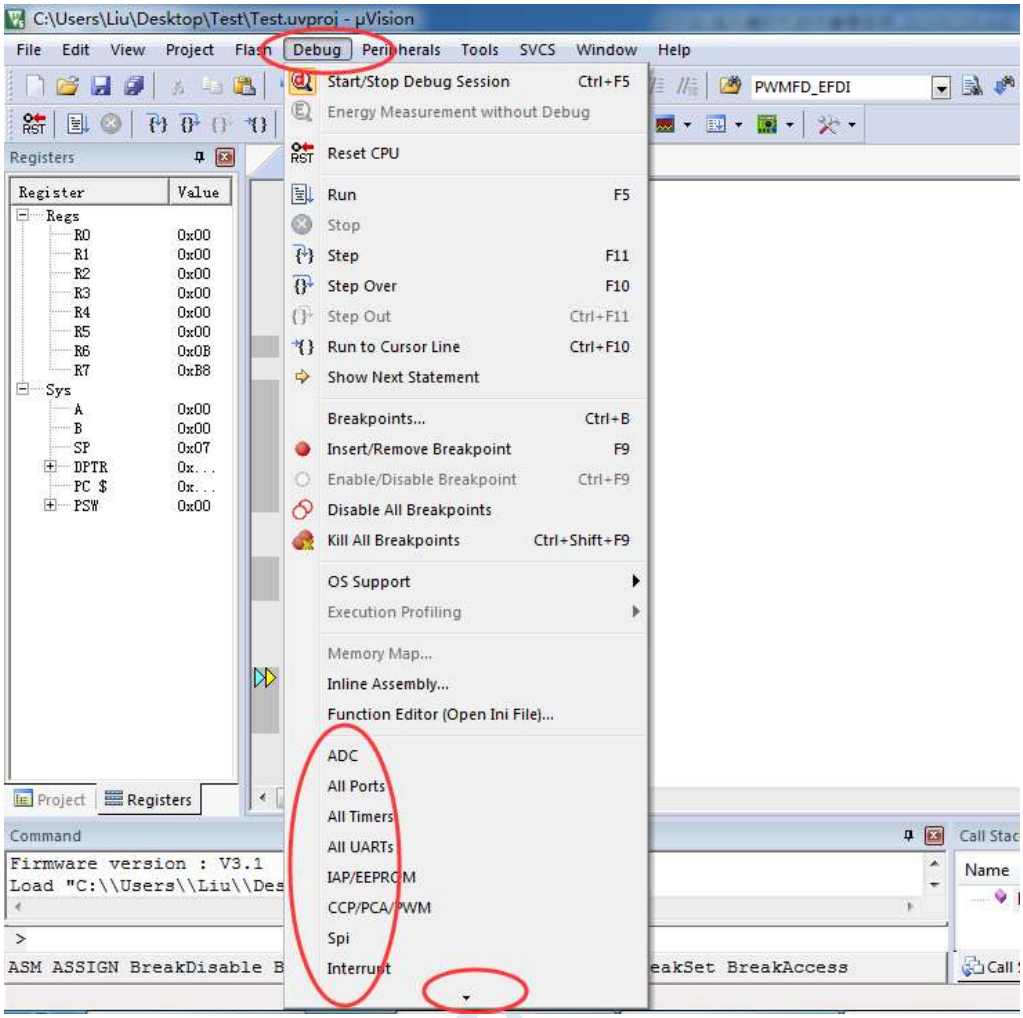
If the hardware connection is correct, you will enter a debugging interface similar to the following, and display the current simulation driver version number and the current simulation monitoring code firmware version number in the command output window.

The current maximum number of breakpoints is 20 (in theory, any number of breakpoints can be set, but too many breakpoints will affect the speed of debugging).



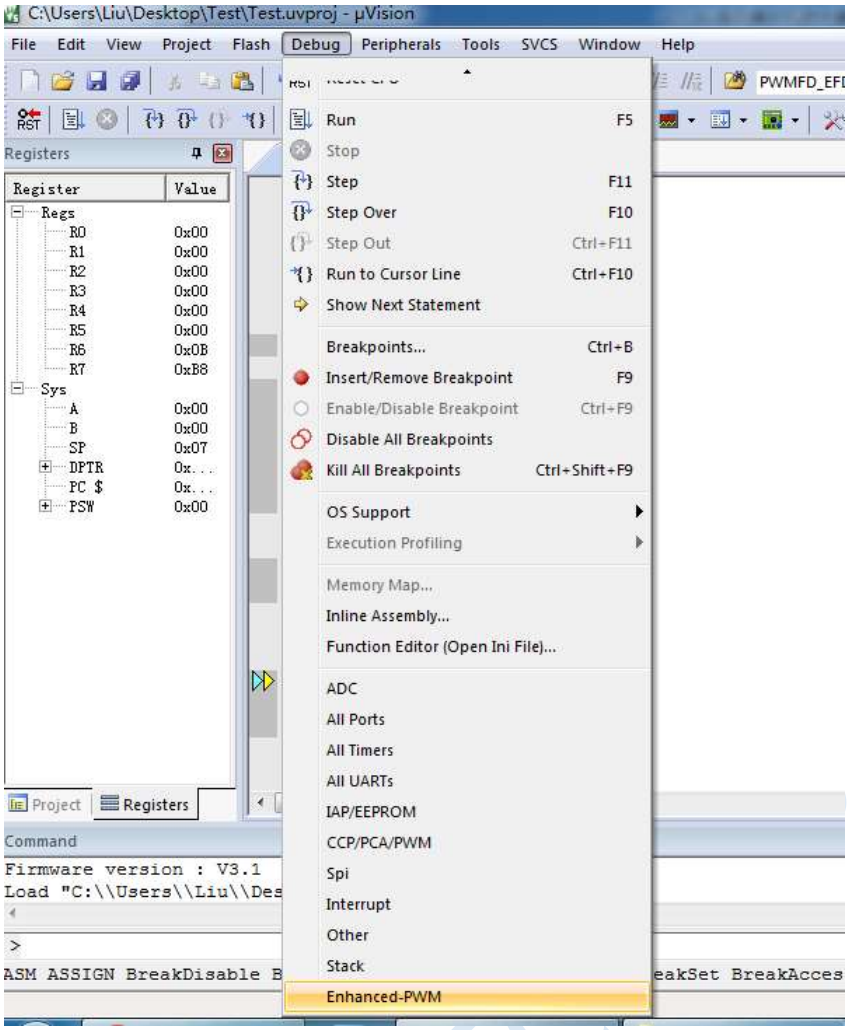
6. View of registers during simulation

During the simulation, you can view the related registers of MCU. A list of all registers is at the bottom of the "Debug" menu. As shown below.



At the bottom of the "Debug" menu in the figure above, there is a small black triangle, which means that there are hidden items (mainly due to the size of the display layout).

Mouse over the small triangle to drag out all items automatically, as shown below:



Simulation considerations:

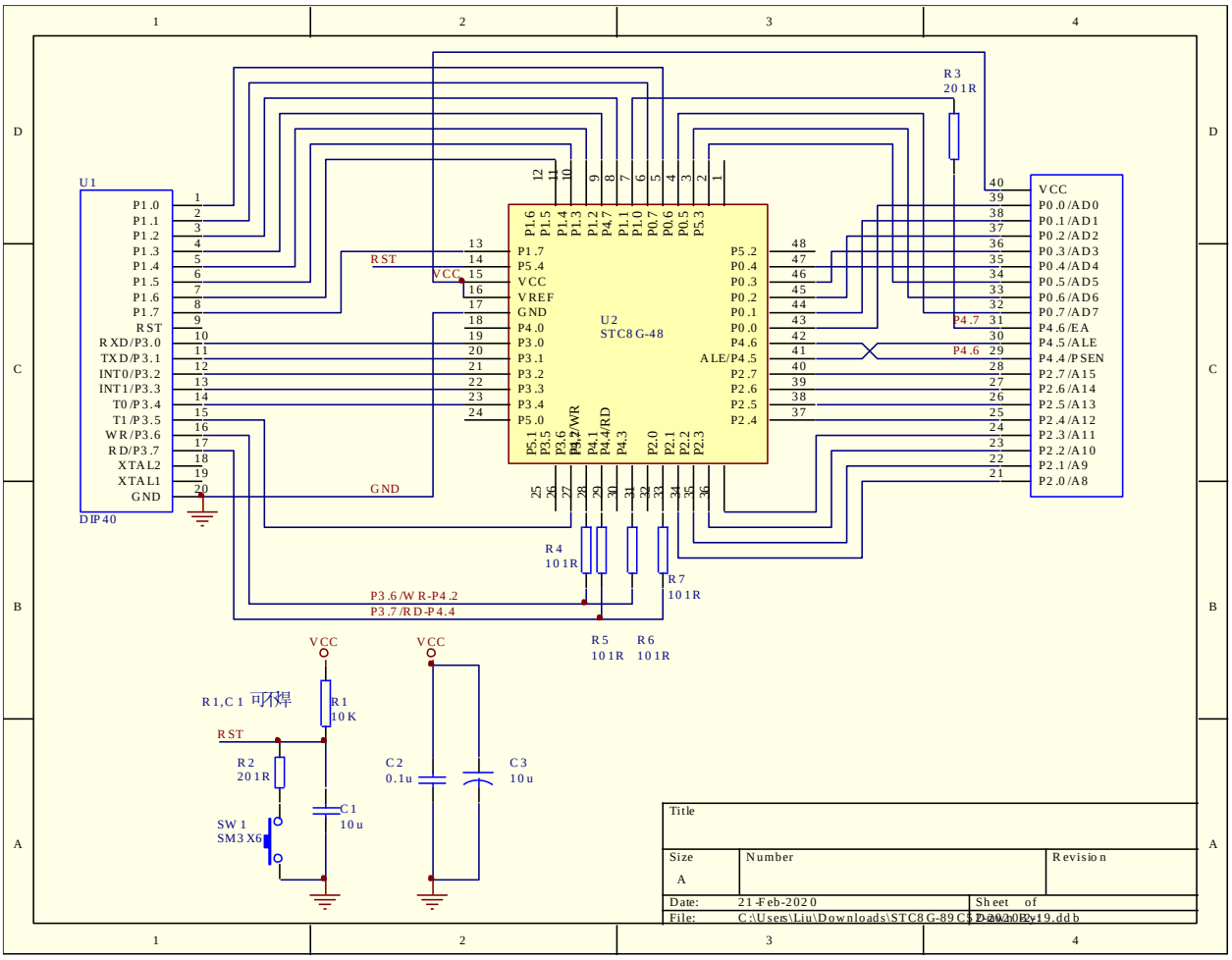
1. The simulation monitoring program occupies the two ports P3.0/P3.1, but does not occupy the serial port 1. The user can switch the serial port 1 to P3.6/P3.7 or P1.6/P1.7 to use.

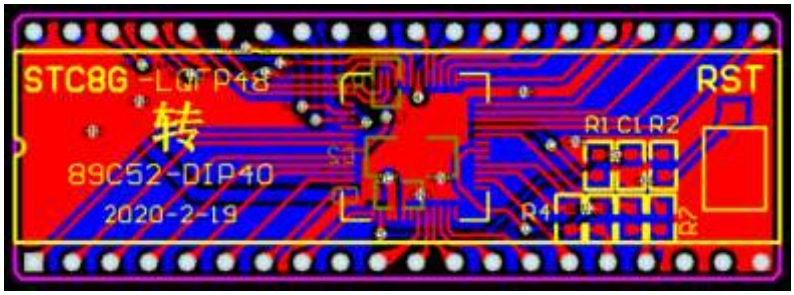
2. The simulation monitoring program occupies the last 768 bytes of the internal extended RAM (XDATA), and the user cannot write to XDATA in this area.

Appendix B How to Make the Traditional 8051 MCU EVB Emulatable

The traditional 8051 microcontroller EVB does not have simulation function. To enable the traditional 8051 microcontroller EVB to be simulated, a conversion board is needed. The physical picture of the conversion board is shown below. The converted pin arrangement is basically the same as that of the traditional 8051. Therefore, the simulation function of the standard 8051 learning board can be realized.

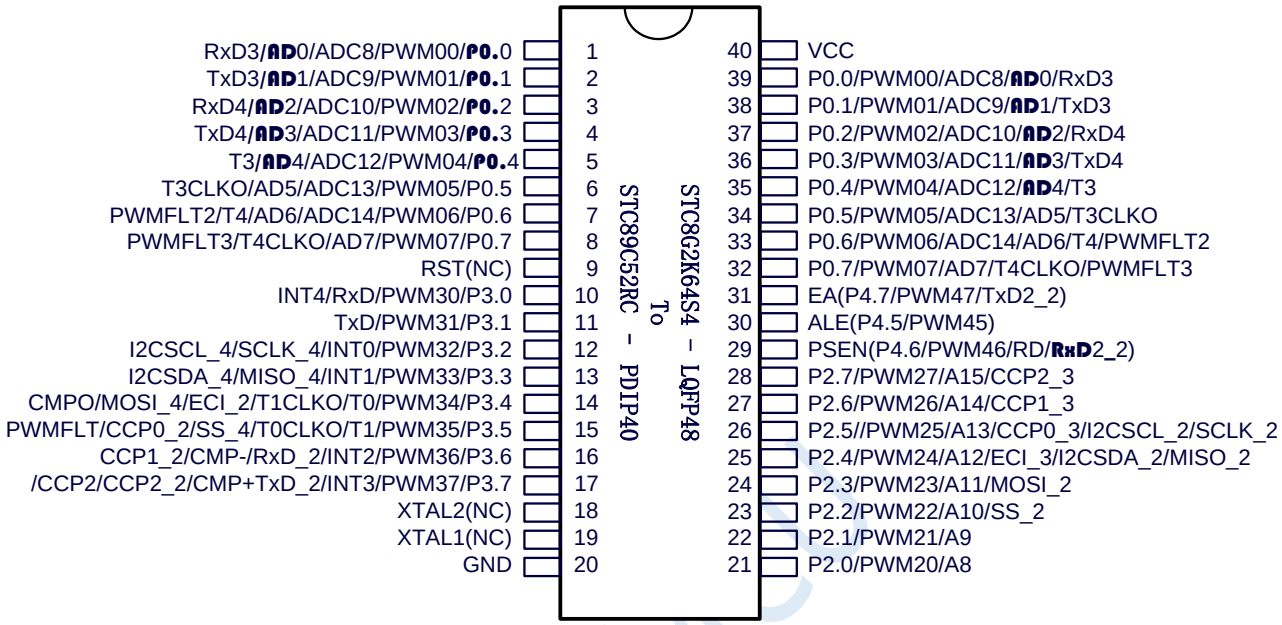
The figures below are the schematic and PCB layout of the converter board.





This conversion board can be used for STC8G series LQFP48 to STC89C52RC / STC89C58RD + series simulation.

The following figure is a functional diagram of the conversion board.



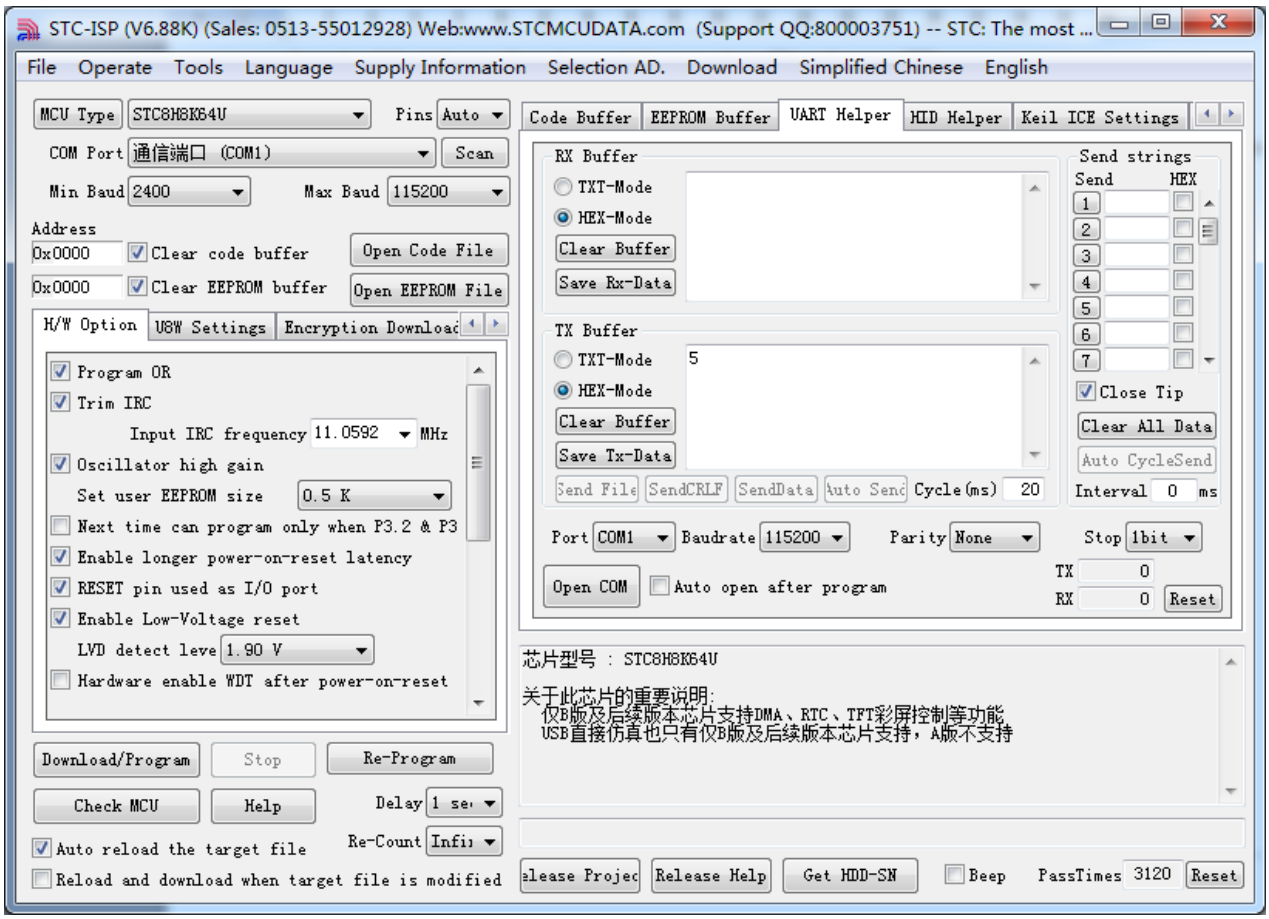
Note:

- ✓ Due to the built-in high-precision R/C clock, no external crystal is needed, XTAL1 and XTAL2 can be empty.
- ✓ WR and RD are P4.2/ WR and P4.4/ RD respectively, not traditional WR/P3.6 and RD/P3.7. **(In the conversion board, P4.2 and P3.6 are connected together, and P4.4 and P3.7 are connected together. When this conversion board is used to access the external bus, P3.6 and P3.7 should be set to high-impedance input mode, so that P4.2 and P4.4 can normally output the bus read and write signals. If the external bus is not needed to be accessed, P4.2 and P4.4 should be set to high-impedance input mode, and P3.6 and P3.7 are ordinary I/O.)**
- ✓ The STC8G series MCUs are low-level reset, it is not compatible with the high-level reset of the traditional 8051, so the RST pin is left floating, and replaced by the reset button and reset circuit on the conversion board.

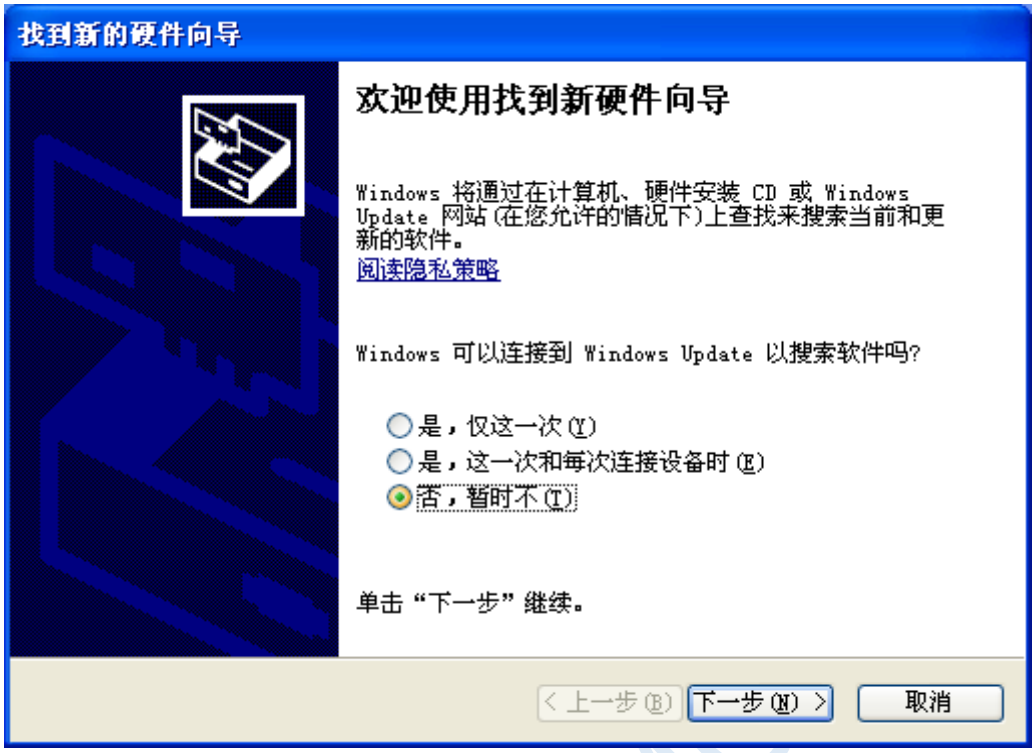
Appendix C STC-USB Driver Installation Instructions

Installation Instructions in Windows XP

Open the STC-ISP download software of V6.79 (or later). The download software will copy the driver files to the relevant system directory automatically.



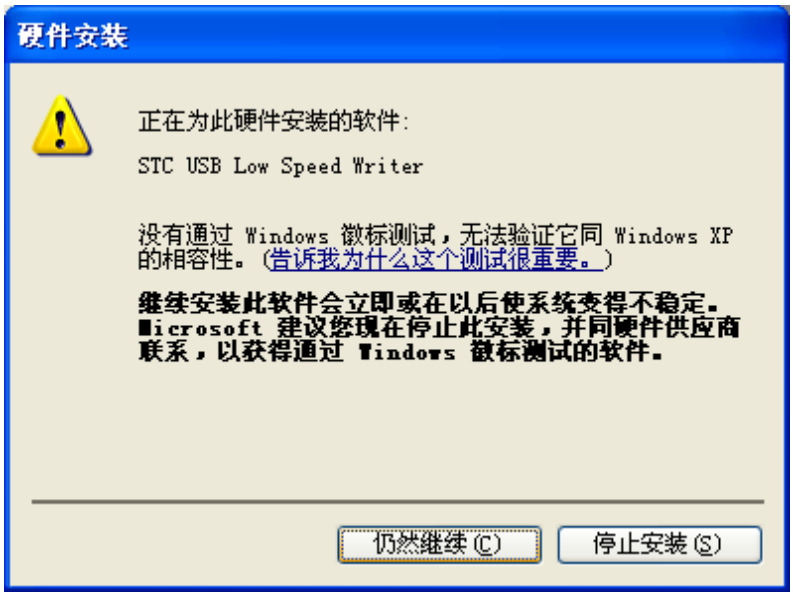
Plug in the USB device, the system will pop up the following dialog box automatically after finding the device, select "No, not this time".



Select "Install software automatically (recommended)" in the dialog below.



In the following dialog box that pops up, select the "Continue Anyway" button.



STC MCU

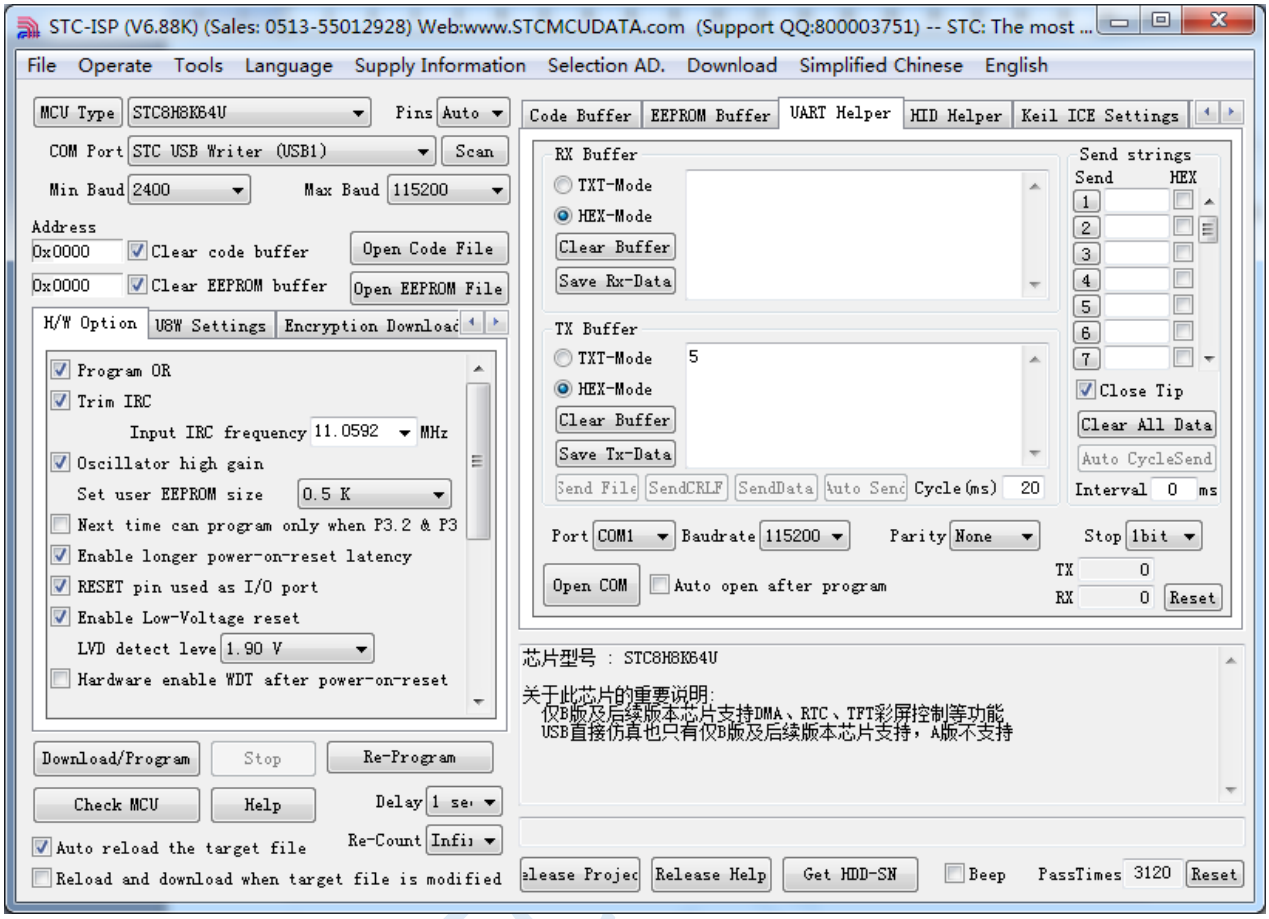
The system will automatically install the driver when connected, as shown below.



The following dialog box appears to indicate that the driver installation is complete.

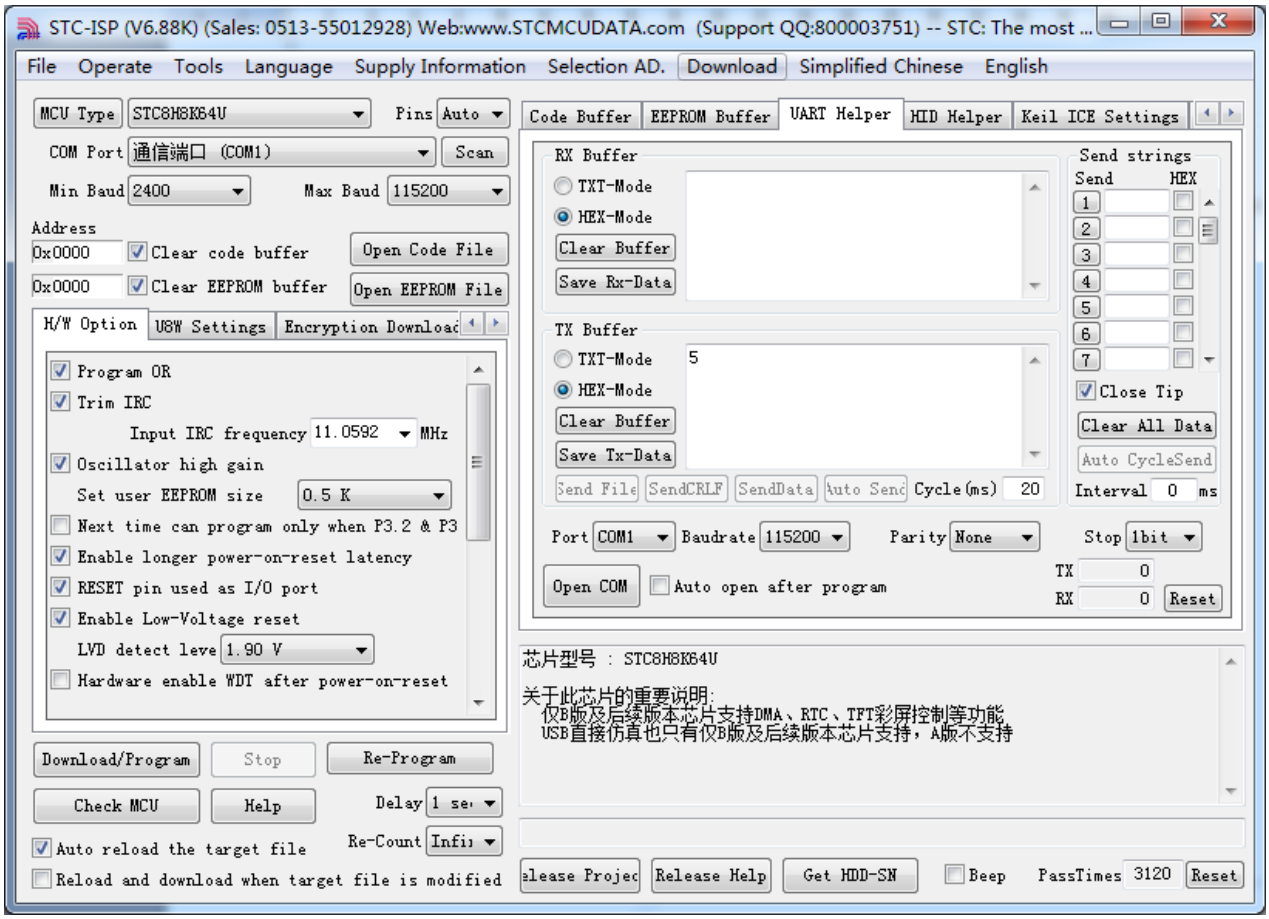


Now, the serial number list in the previously opened STC-ISP download software will select the inserted USB device automatically and display the device name as "STC USB Writer (USB1)", as shown below.



Installation Instructions in Windows 7 (32-bit)

Open the STC-ISP download software of V6.79 (or later). The download software will copy the driver files to the relevant system directory automatically.

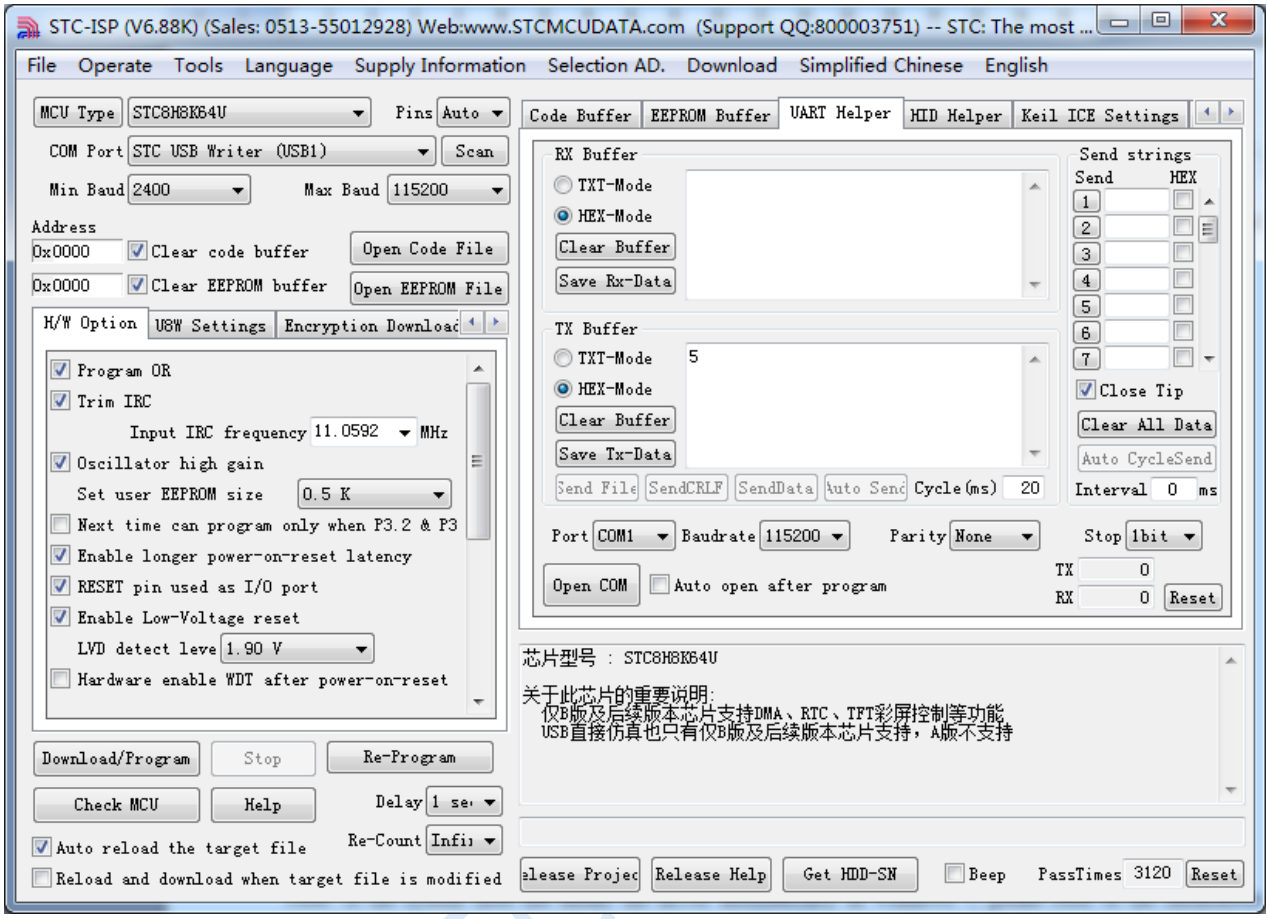


Plug in the USB device, and the system will install the driver automatically when it finds the device. After the installation is complete, the following prompt box will appear.



STC MCU

Now, the serial port number list in the previously opened STC-ISP download software will select the inserted USB device automatically and display the device name as "STC USB Writer (USB1)", as shown below.

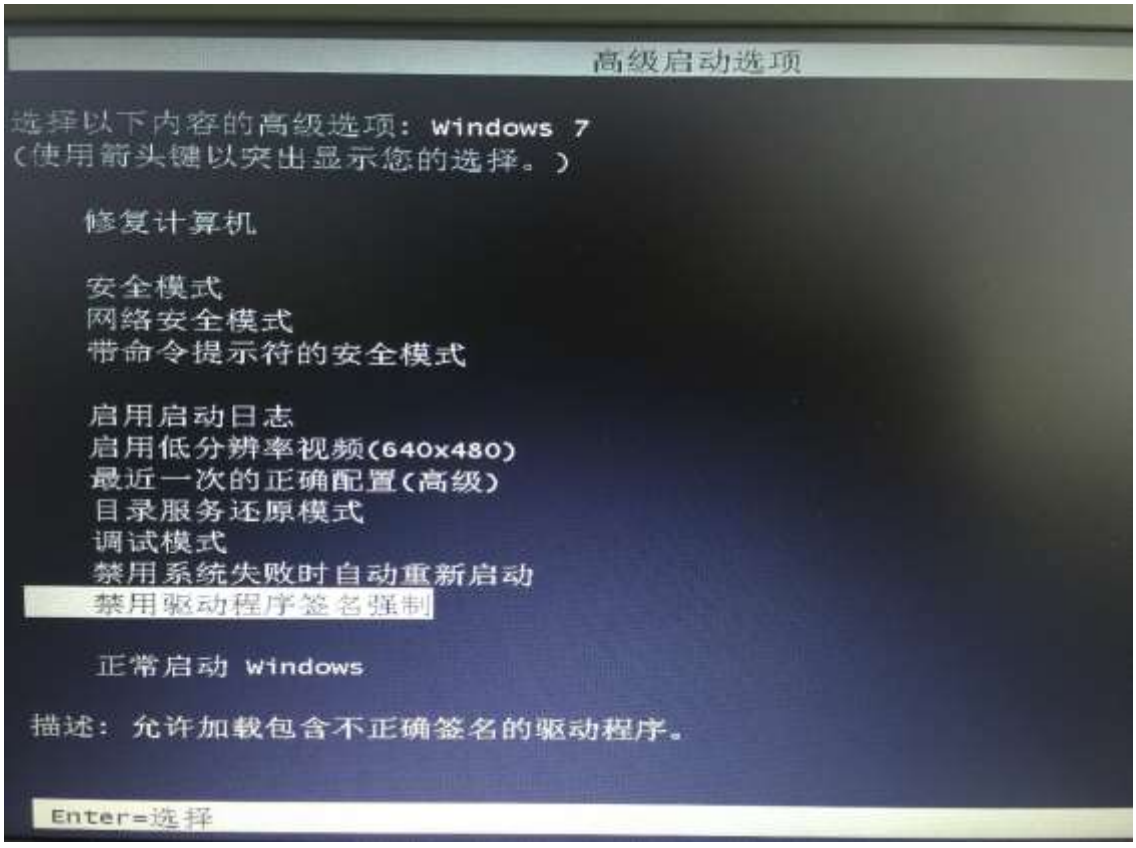


Note: If the system does not install the driver automatically in Windows 7, please refer to the installation method of Windows 8 (32-bit) for the driver installation method.

Installation Instructions in Windows 7 (64-bit)

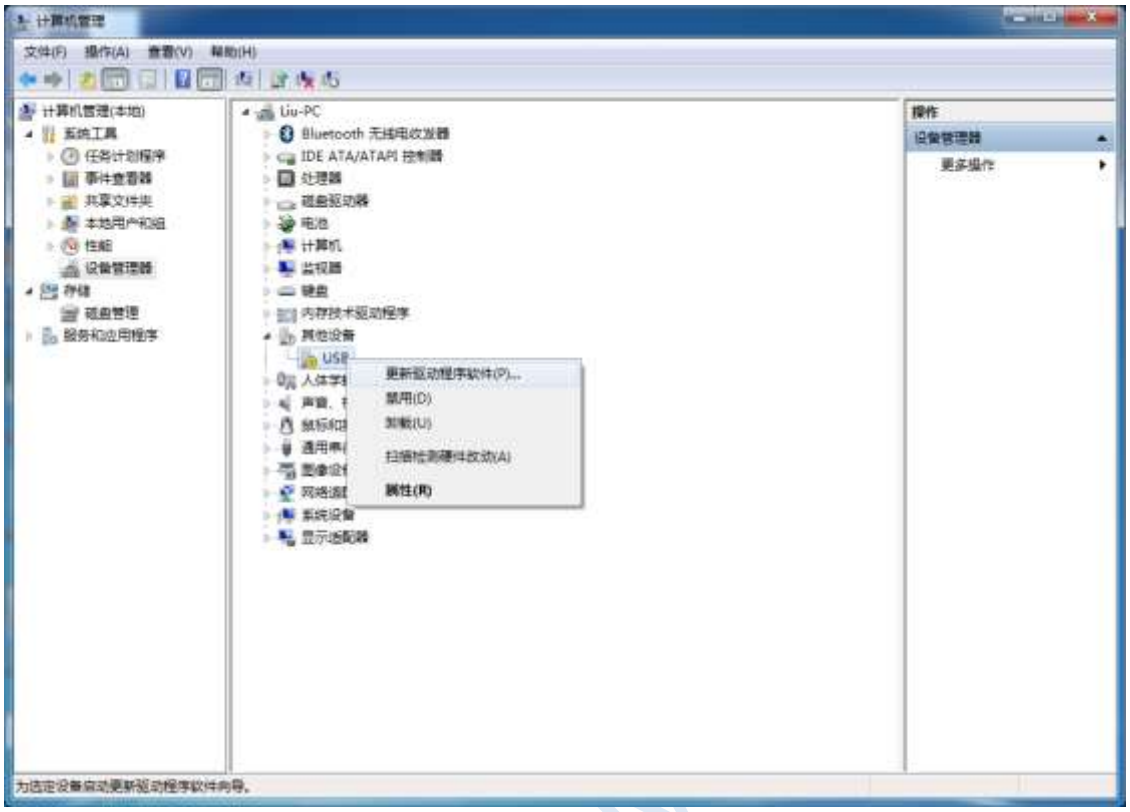
By default, the driver without digital signature cannot be successfully installed in Windows 7 64-bit operating system. So, you need to follow the steps below before installing the STC-USB driver, skip the digital signature temporarily, and the installation will be successful.

Restart the computer firstly and keep pressing F8 until the following startup screen appears.



Select ‘Disable Driver Signature Enforcement’. The digital signature verification function is temporarily turn off after startup.

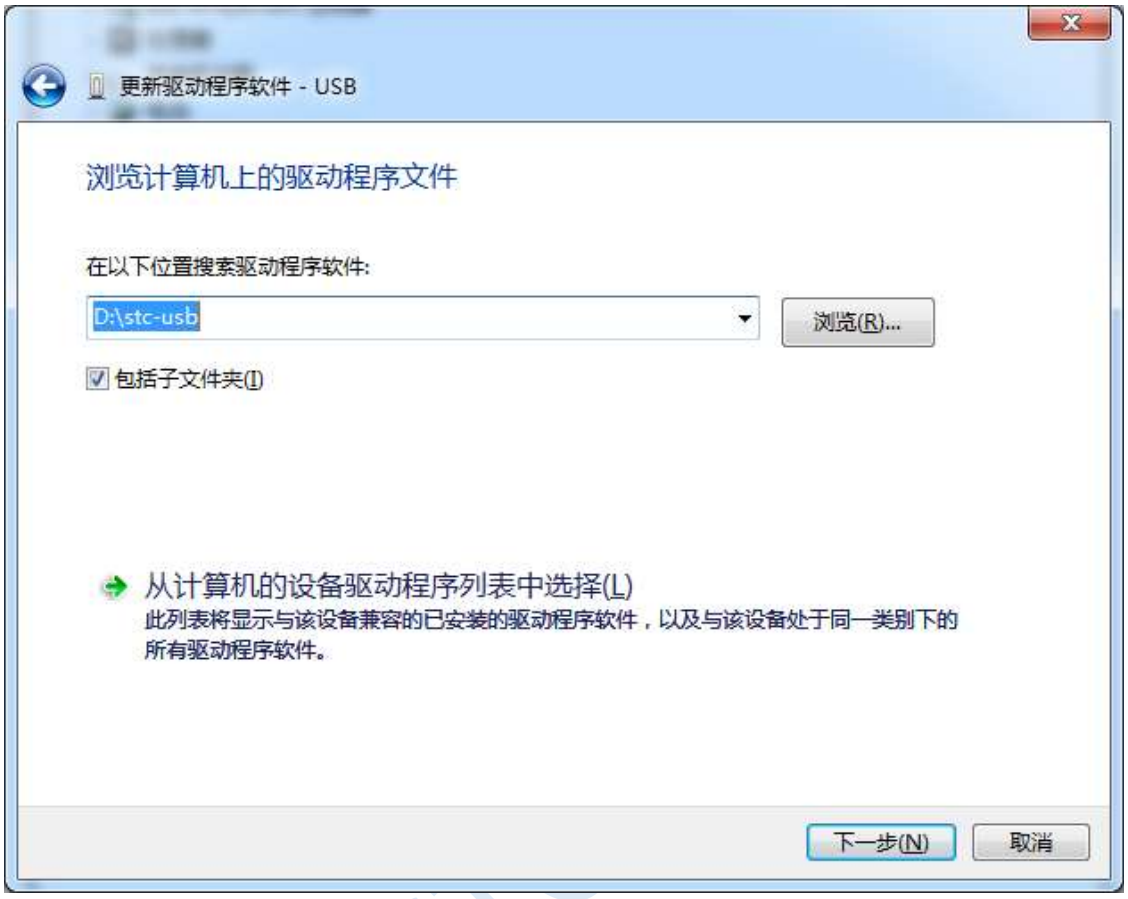
Plug in the USB device and open the Device Manager. Find the USB device with a yellow exclamation mark in the device list, in the device's right-click menu, select "Update Driver Software".



Select "Browse my computer for driver software" in the dialog below.



Click the "Browse" button in the dialog below to find the directory of the previous STC-USB driver stored (for example: the previous example directory is "D:\STC-USB", locate the path to the actual decompression directory).

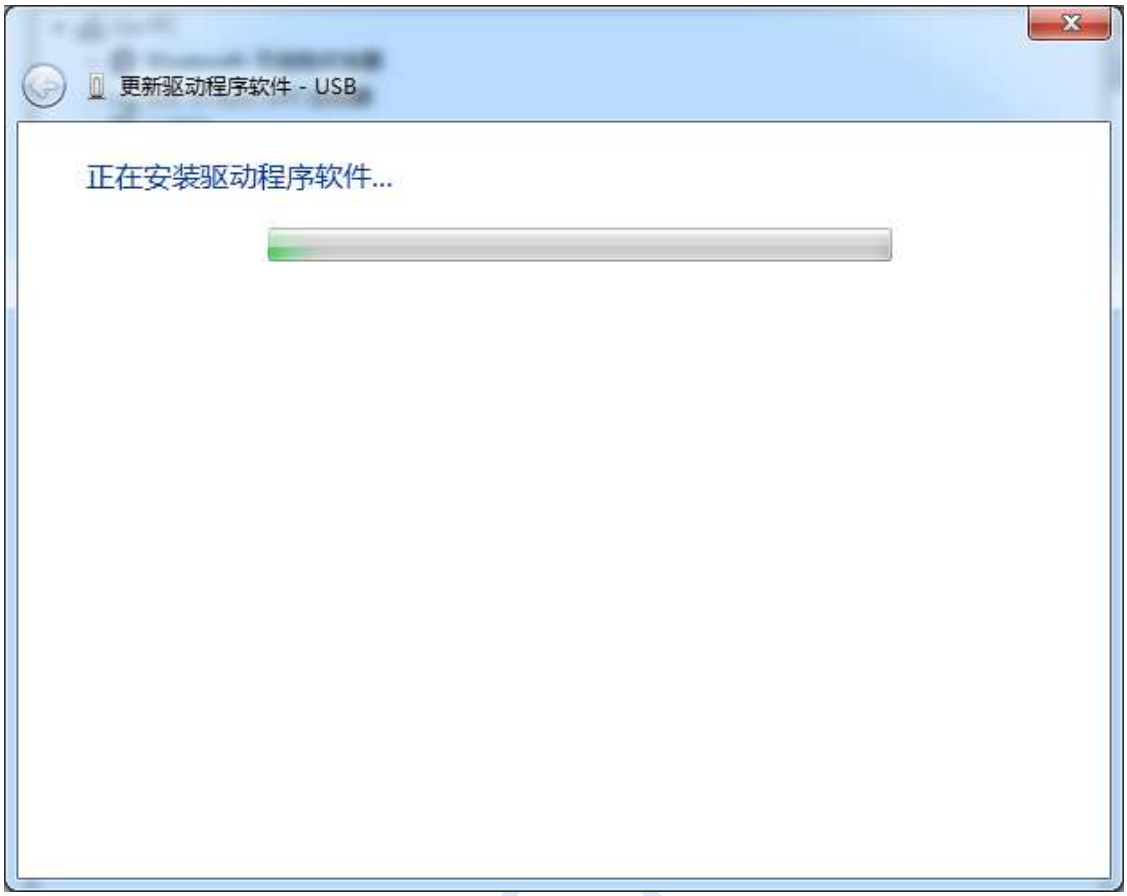


When the driver installation starts, the following dialog box will pop up, select "Always install this driver software".

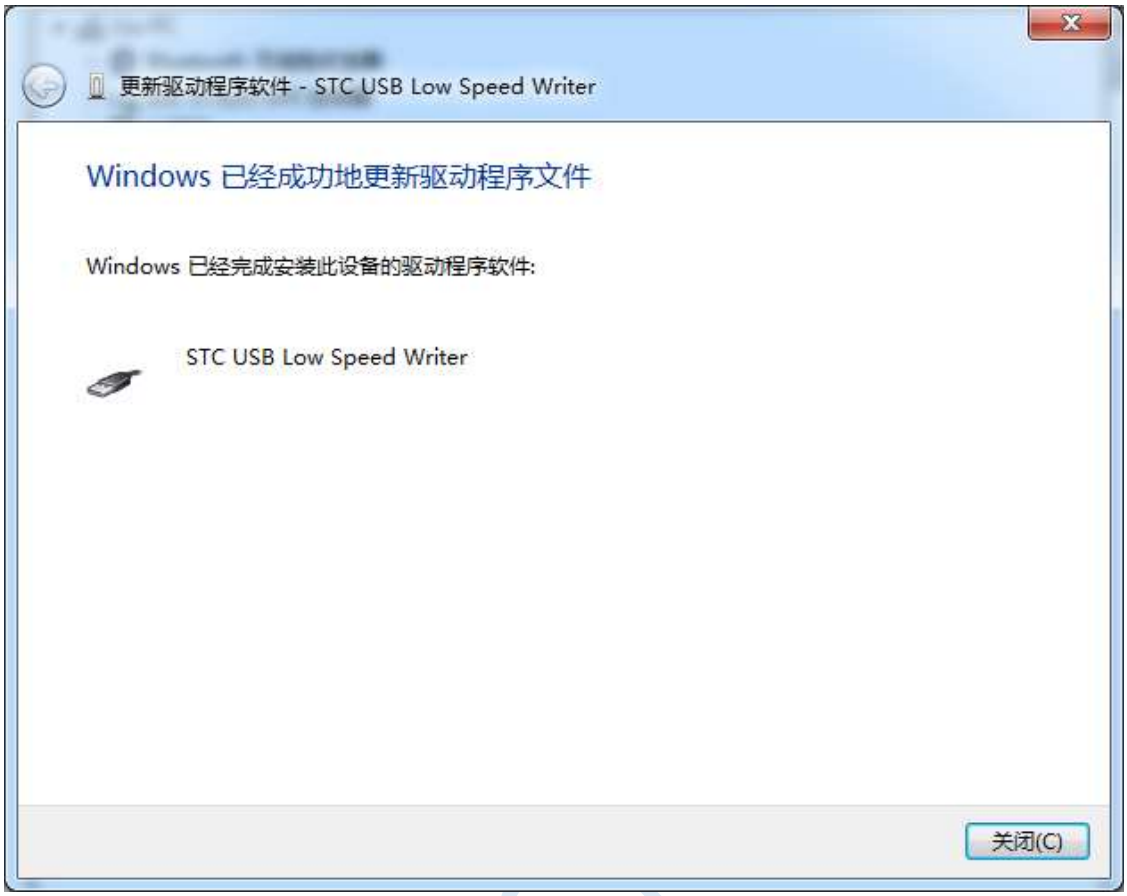


STC MCU

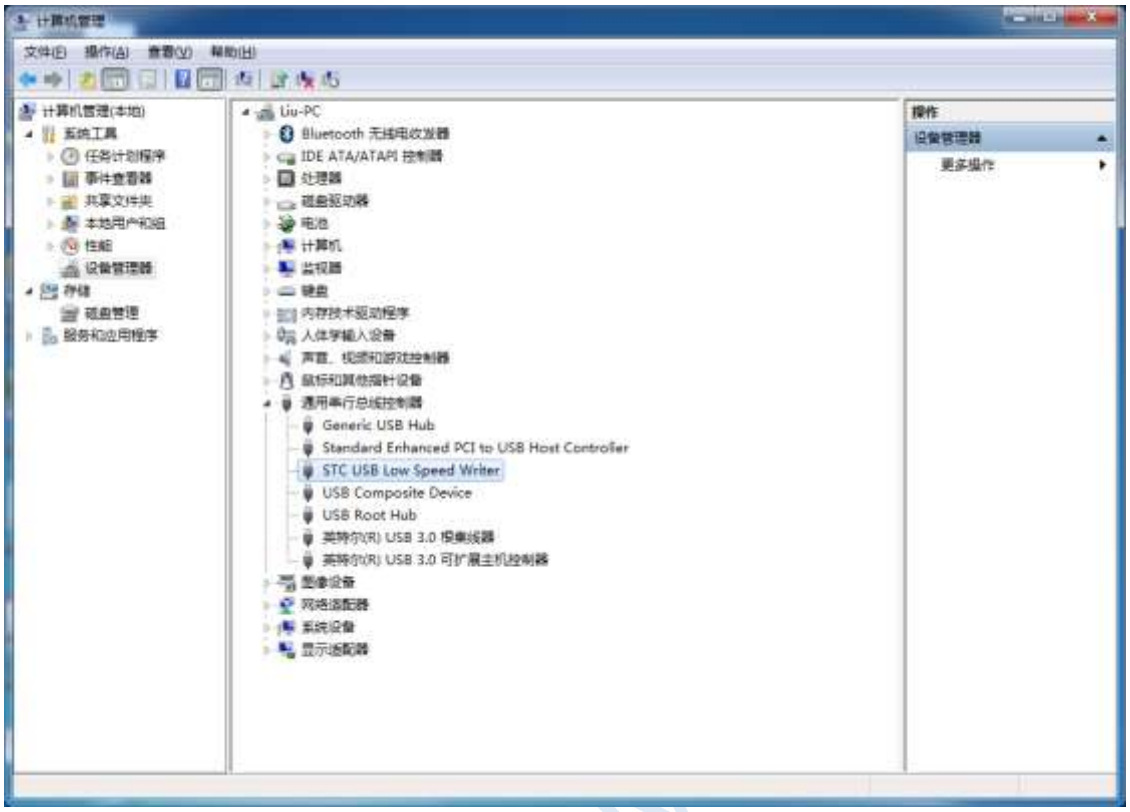
Next, the system will install the driver automatically, as shown below.



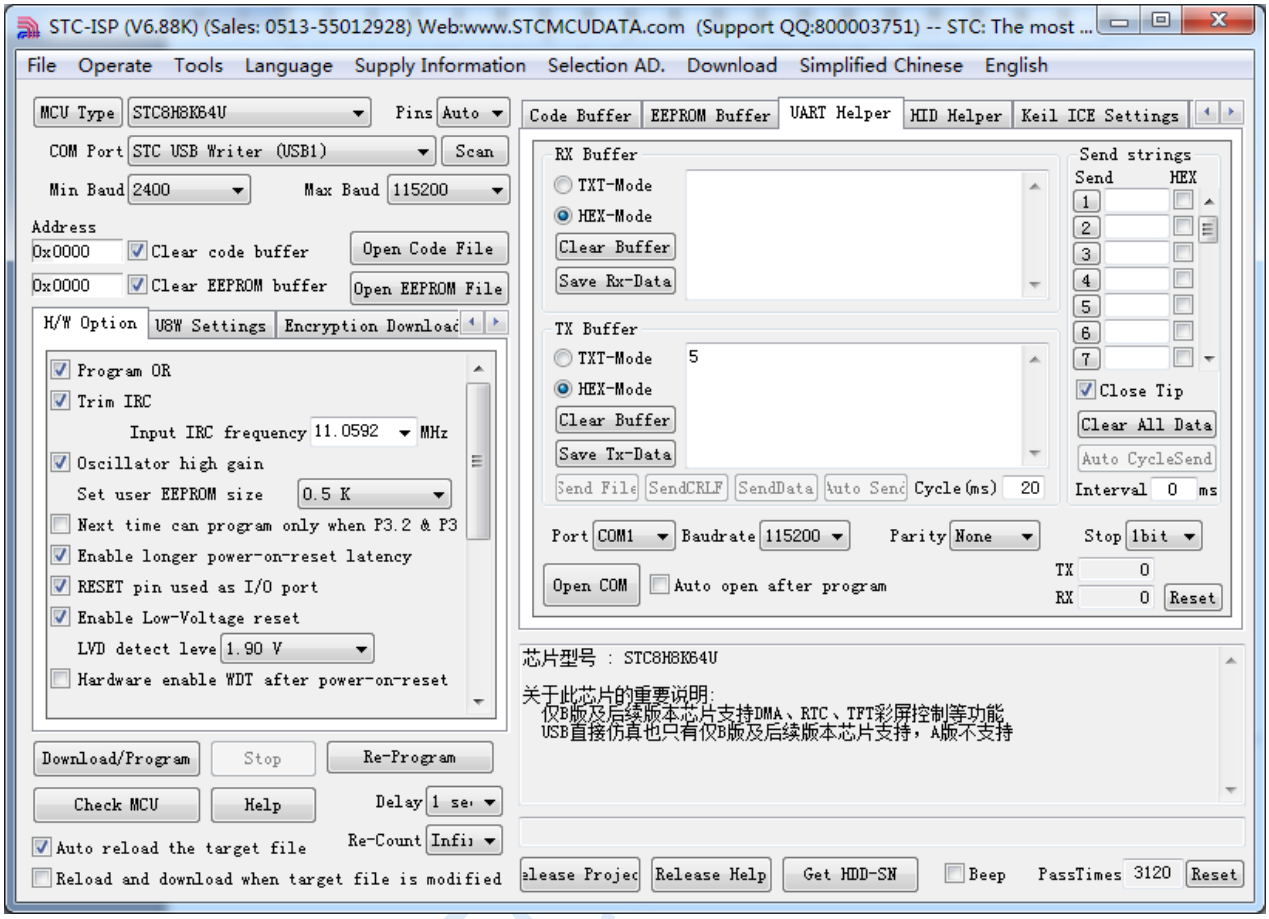
The following dialog box appears to indicate that the driver installation is complete.



Now in the device manager, the device with the yellow exclamation mark before will be displayed as the device name of "STC USB Low Speed Writer".

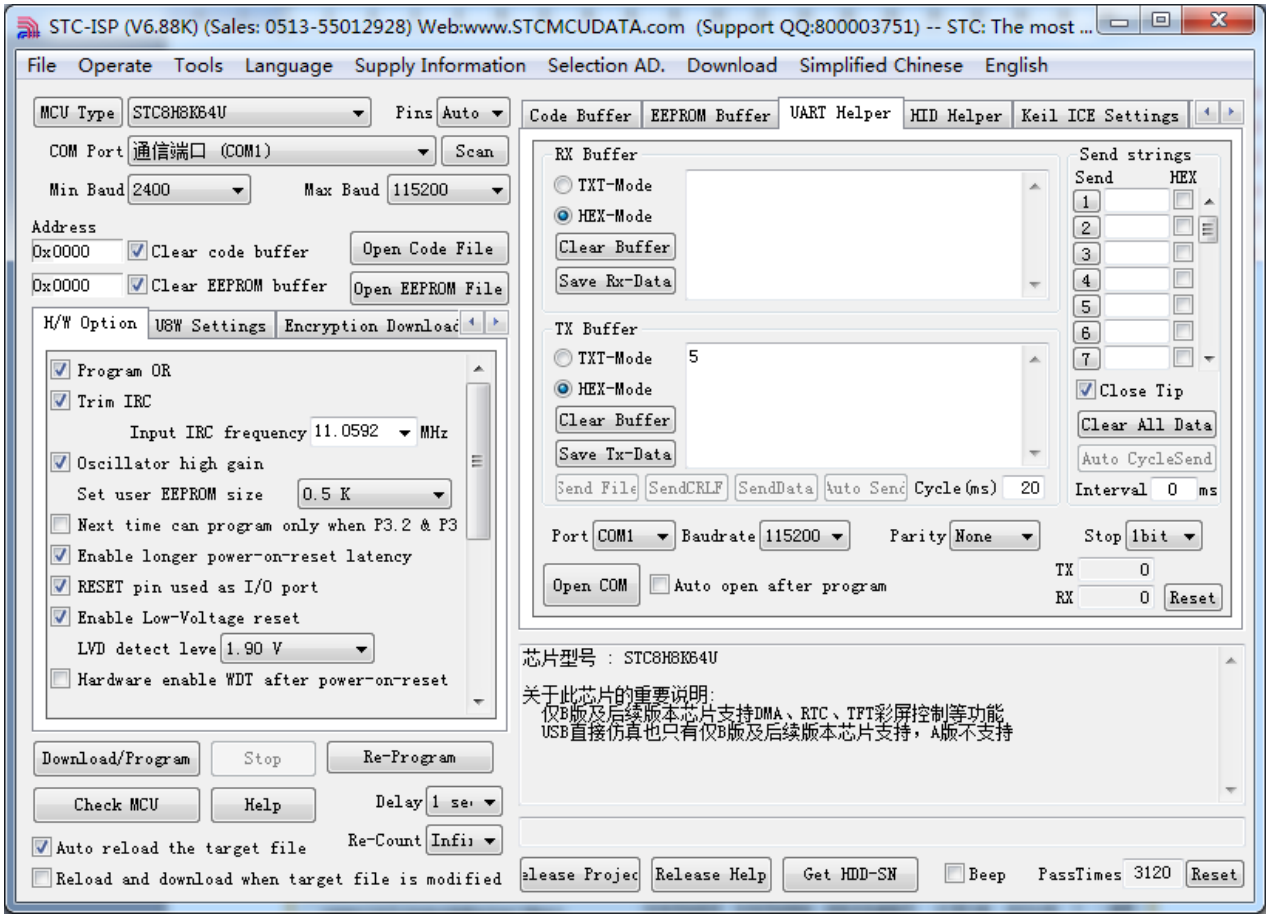


The serial number list in the previously downloaded STC-ISP download software will select the inserted USB device automatically and display the device name as "STC USB Writer (USB1)", as shown below.

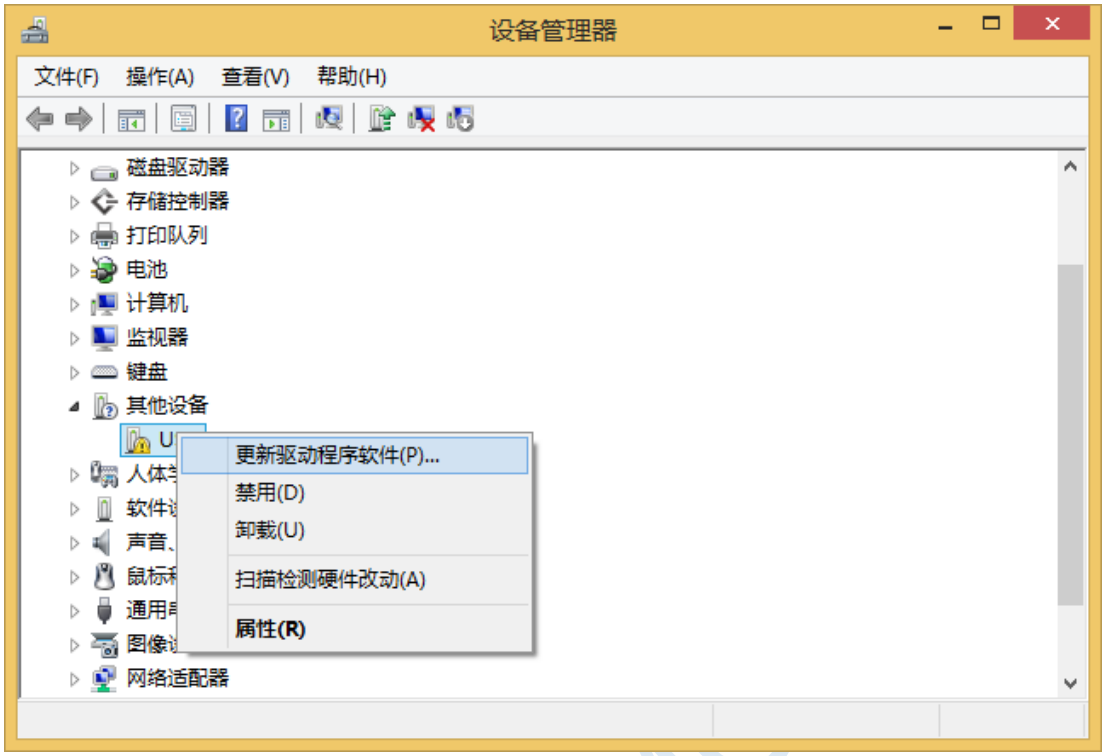


Installation Instructions in Windows 8 (32-bit)

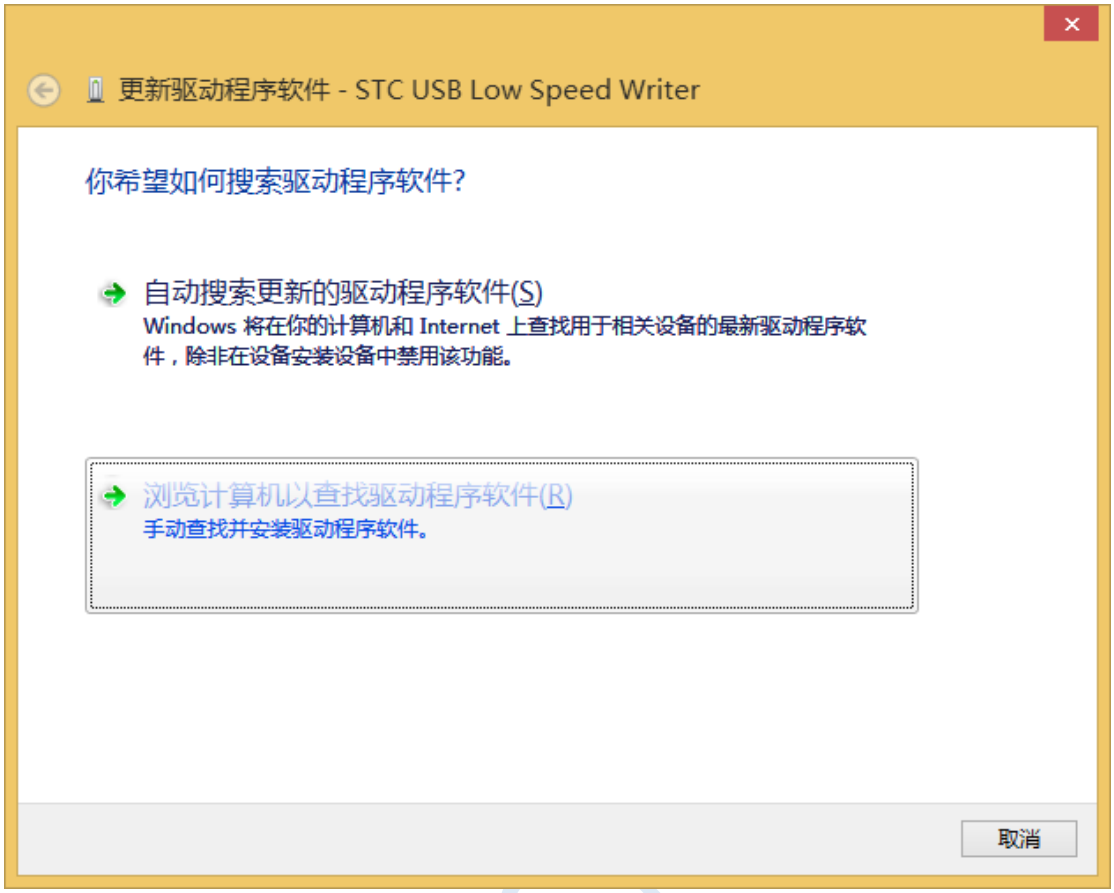
Open the STC-ISP download software of V6.79 (or newer version) (due to permission reasons, downloading the software in Windows 8 will not copy the driver files to the relevant system directory. It requires manual installation by the user. Firstly, download "stc-isp-15xx-v6.79.zip" (or newer version) from the STC official website, and decompress it to the local disk after downloading, then the STC-USB driver file will also be decompressed to the "STC-USB Driver" folder of the current folder. (For example, decompress the downloaded compressed file" stc-isp-15xx-v6.79.zip "to" F:\", then the STC-USB driver is in the" F:\ STC-USB Driver "directory))



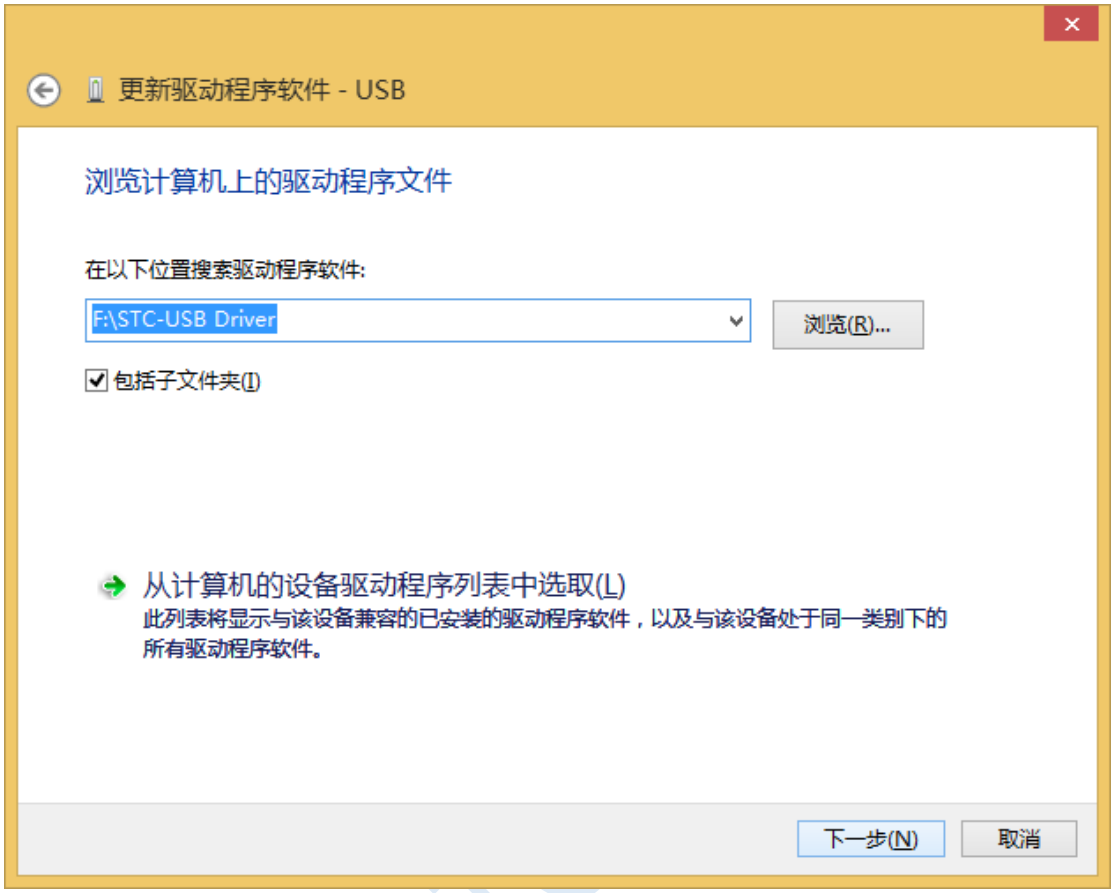
Plug in the USB device and open the Device Manager. Find the USB device with a yellow exclamation mark in the device list, in the device's right-click menu, select "Update Driver Software".



Select "Browse my computer for driver software" in the dialog below.



Click the "Browse" button in the dialog below to find the directory where the STC-USB driver was stored (for example: the previous example directory is "F: \ STC-USB Driver", locate the path to the actual decompression directory).

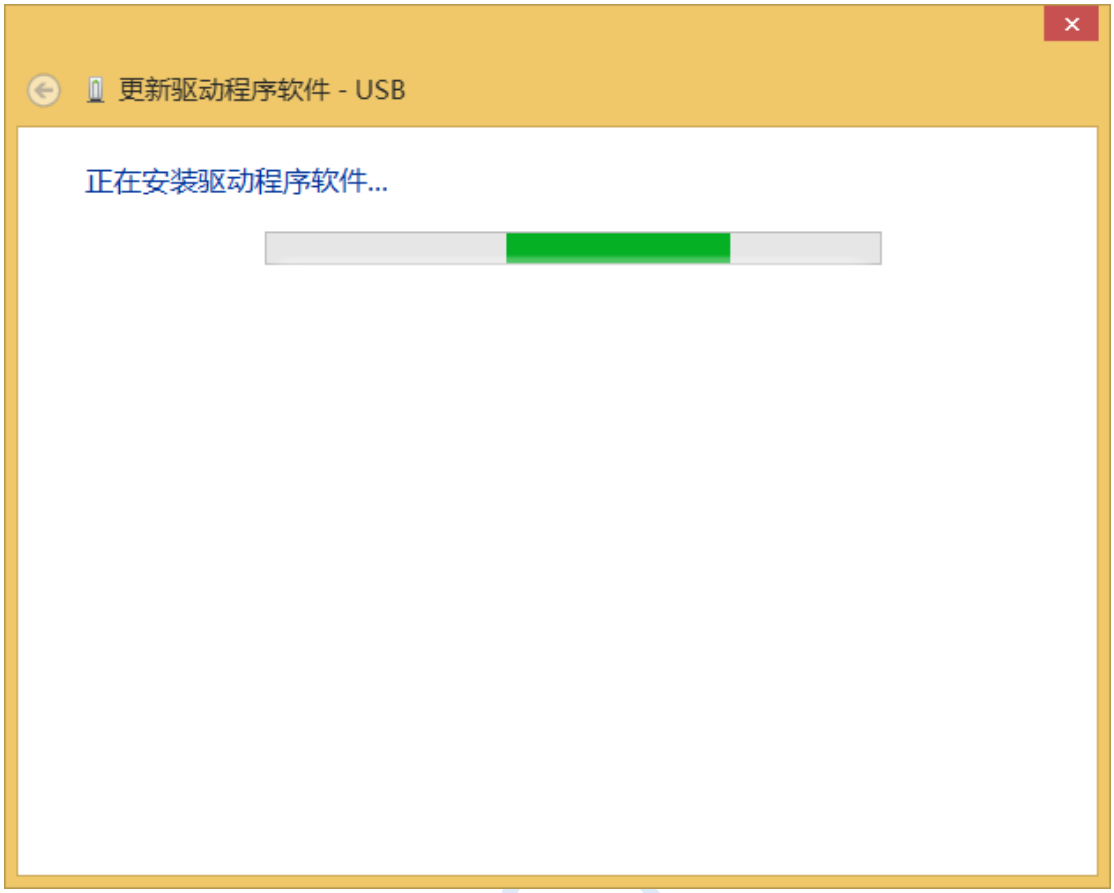


When the driver installation begins, the following dialog box will pop up, select "Always install this driver software".

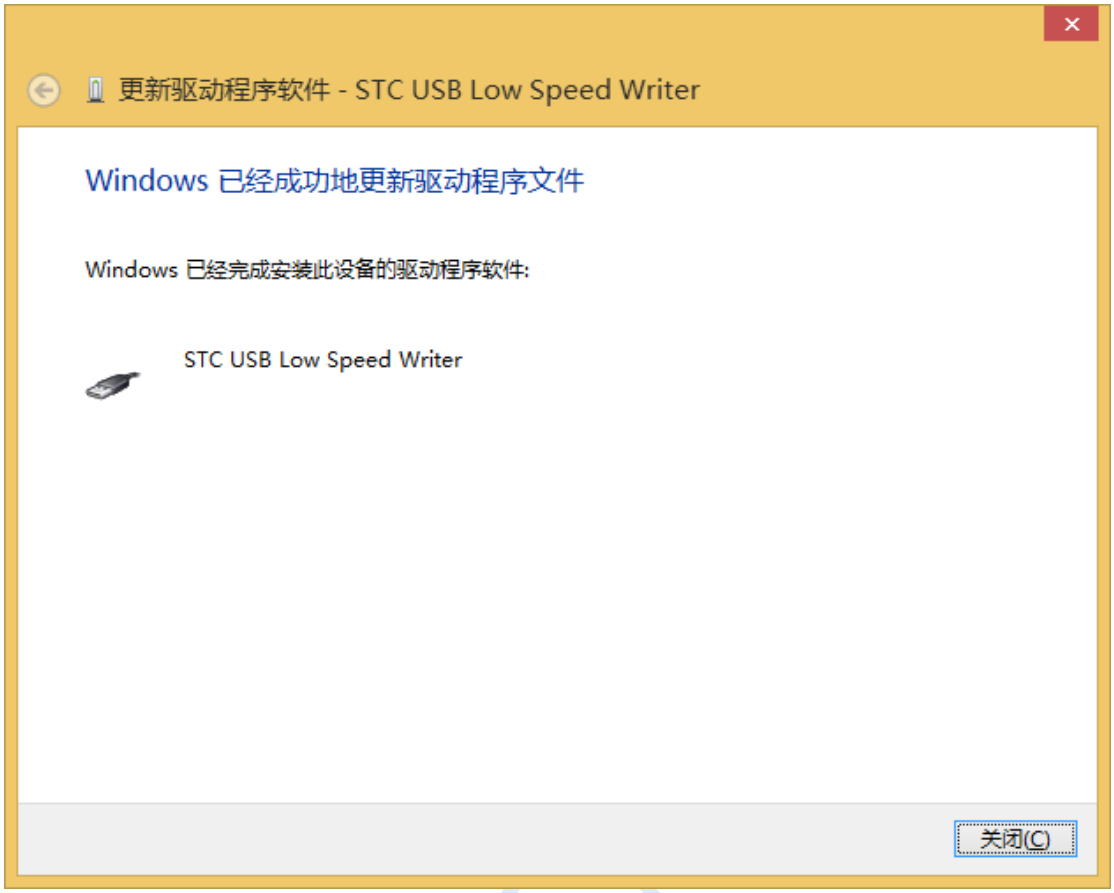


STC MCU

Next, the system will install the driver automatically, as shown below.



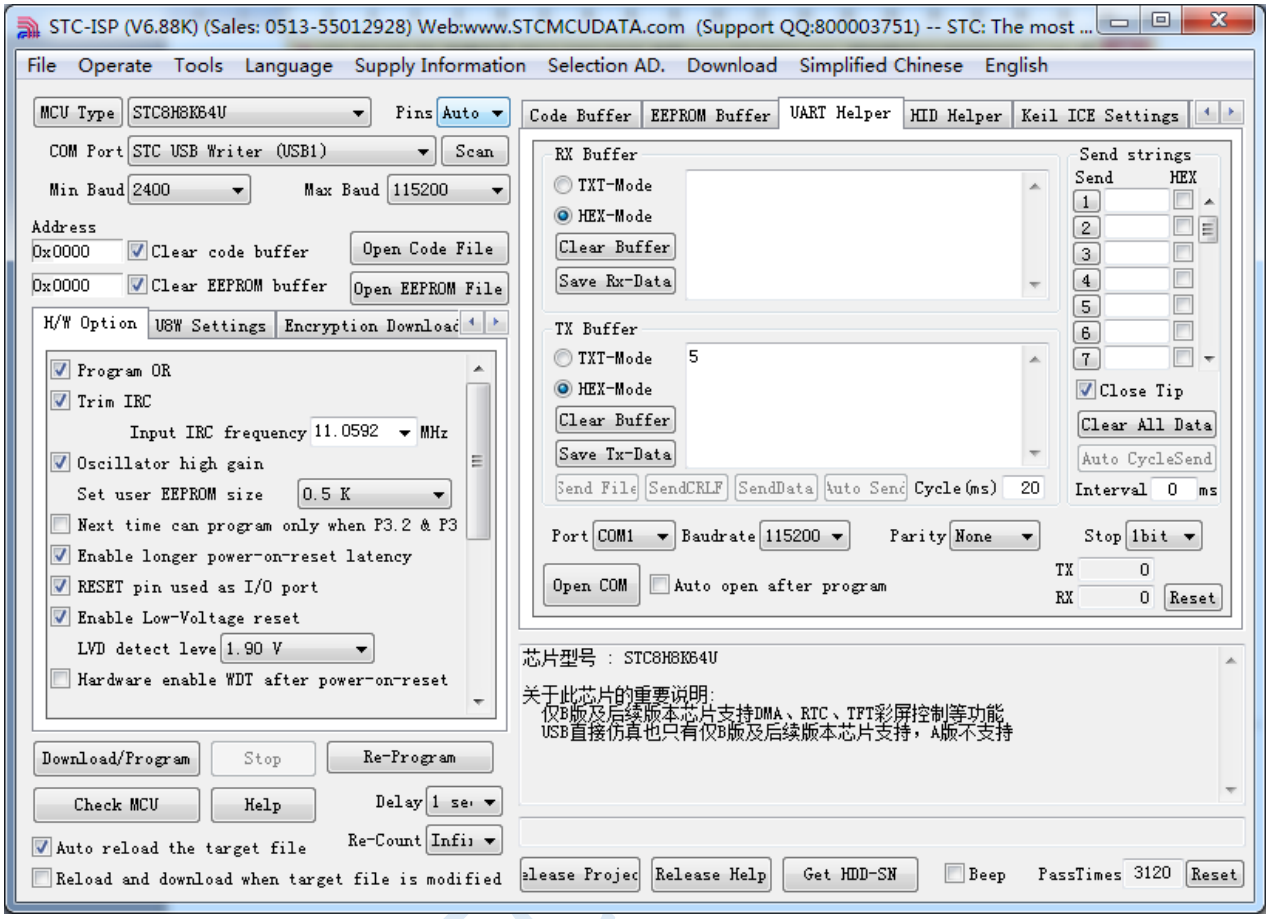
The following dialog box appears to indicate that the driver installation is complete.



Now in the device manager, the device with the yellow exclamation mark before will be displayed as the device name of "STC USB Low Speed Writer".



The serial number list in the previously downloaded STC-ISP download software will select the inserted USB device automatically and display the device name as "STC USB Writer (USB1)", as shown below:



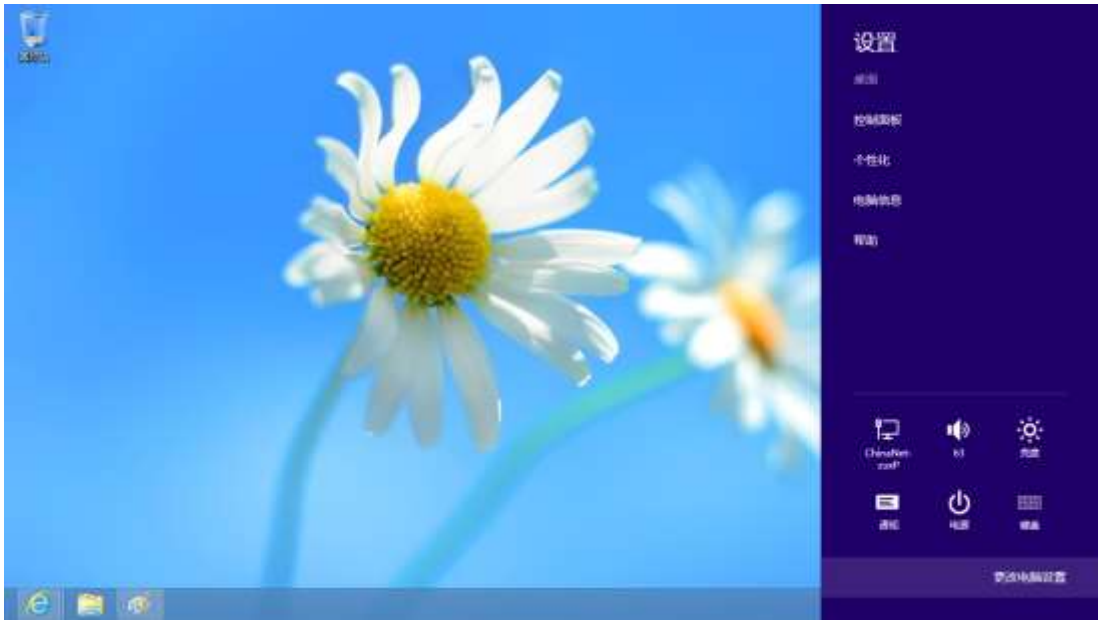
Installation Instructions in Windows 8 (64-bit)

By default, the driver without digital signature cannot be successfully installed in Windows 8 64-bit operating system. So, you need to follow the steps below before installing the STC-USB driver, skip the digital signature temporarily, and the installation will be successful.

Firstly, move the mouse to the lower right corner of the screen and select the "Settings" button.



Then select the "Change PC settings" item in the settings window.



STC MCU

In the computer settings, select the "Start Now" button under the "Advanced Startup" item in the "General" property page.



STC MCU

In the window below, select the "Troubleshooting" item.



Then select "Advanced Options" in "Troubleshooting".



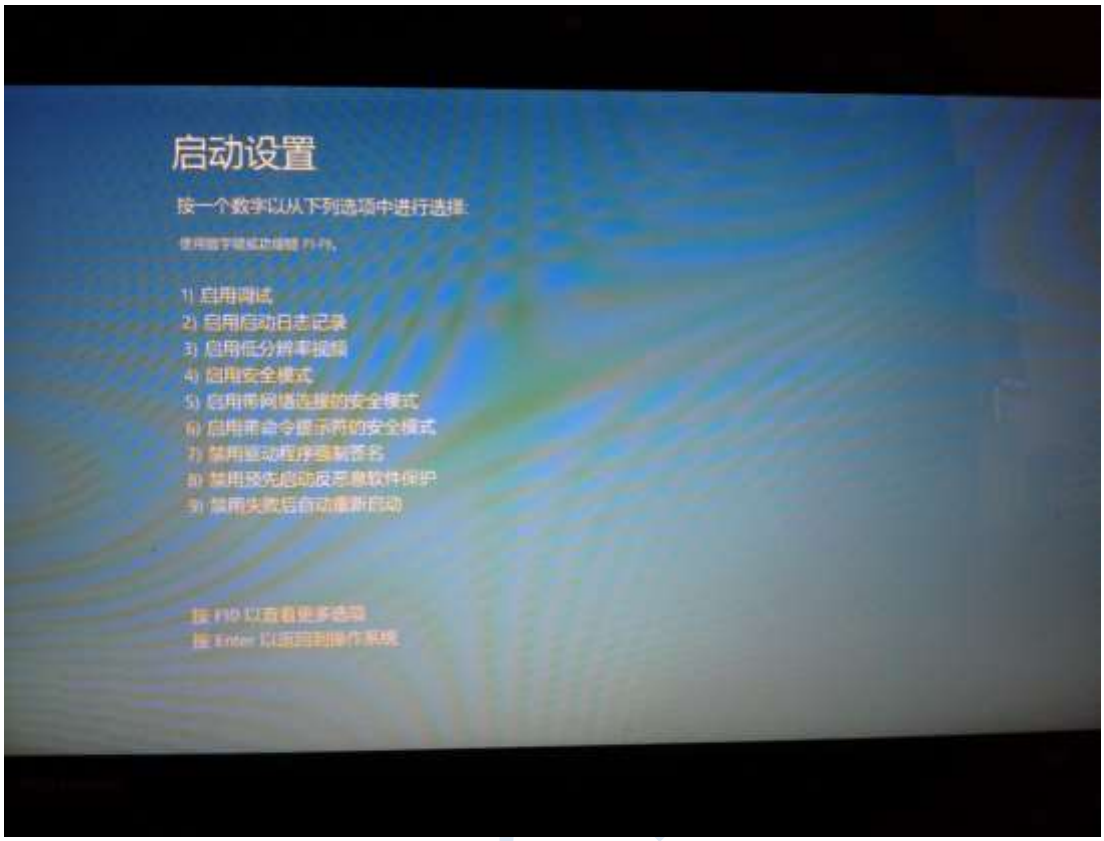
In the "Advanced Options" window below, select "Startup Settings".



In the "Startup Settings" window below, click the "Restart" button to restart the computer.



After the computer restarts, it will enter the "Startup Settings" window automatically as shown in the figure below. Press the number key "7" or press the function key "F7" to select "Disable driver forcing signature" to start.



After booting to Windows 8, follow the Windows 8 (32-bit) installation method to complete the driver installation.

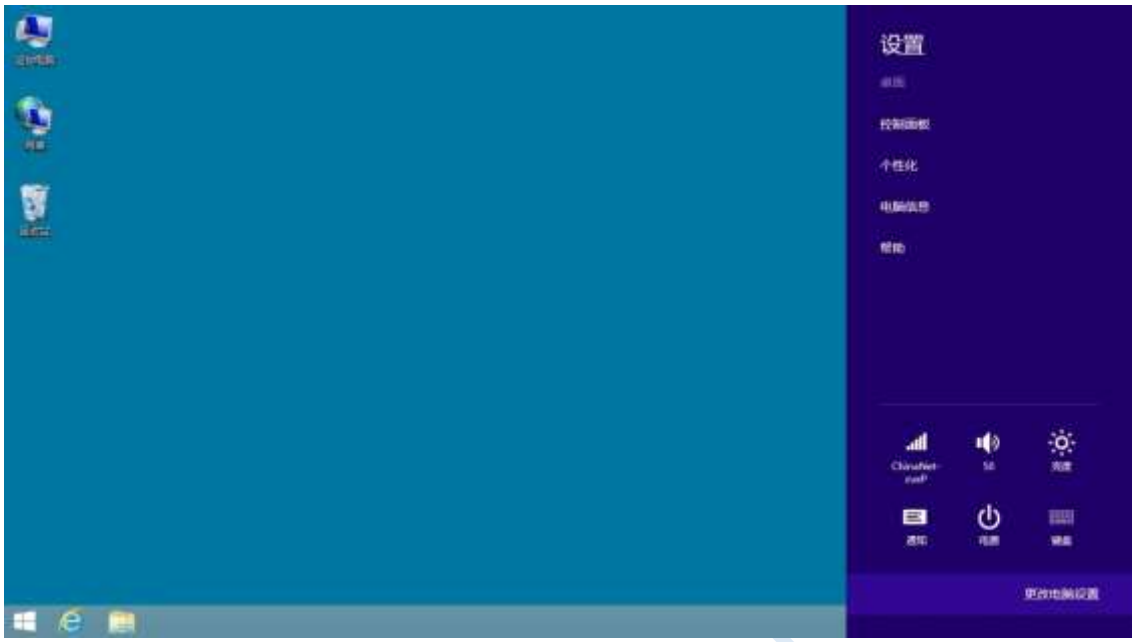
Installation Instructions in Windows 8.1 (64-bit)

Windows 8.1 has different method for entering the advanced boot menu with respect to Windows 8, which is specifically explained here.

Firstly, move the mouse to the lower right corner of the screen and select the "Settings" button.



Then select the "Change PC settings" item in the settings window.



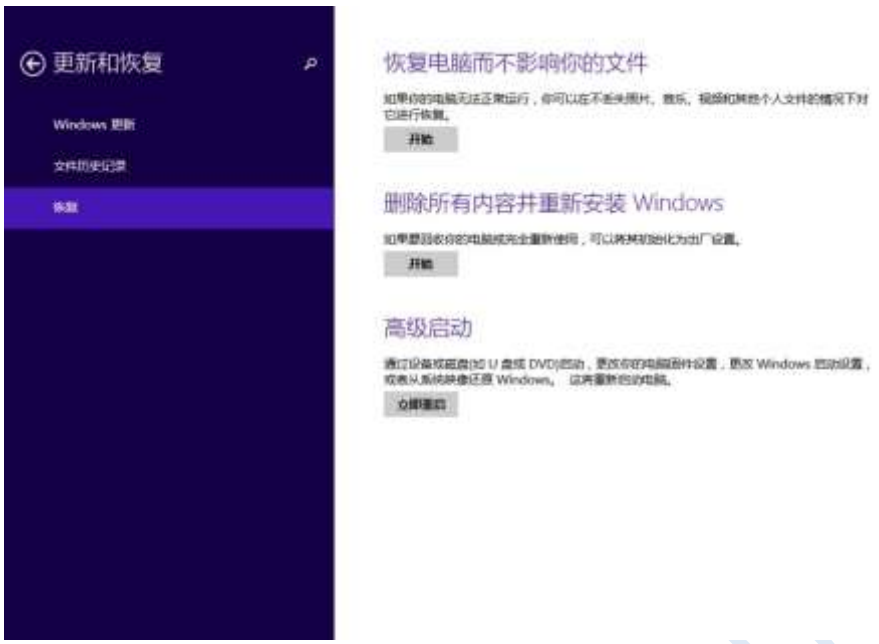
STC MCU

In the computer settings, select "Update and Recovery" (this is not the same as Windows 8, which is "General").



STC MCU

Select the "Restore" property page in the update and recovery page, and click the "Start Now" button under the "Advanced Startup" item.



STC MCU

The following steps are the same as those of Window 8.

In the window below, select the "Troubleshooting" item.



Then select "Advanced Options" in "Troubleshooting".



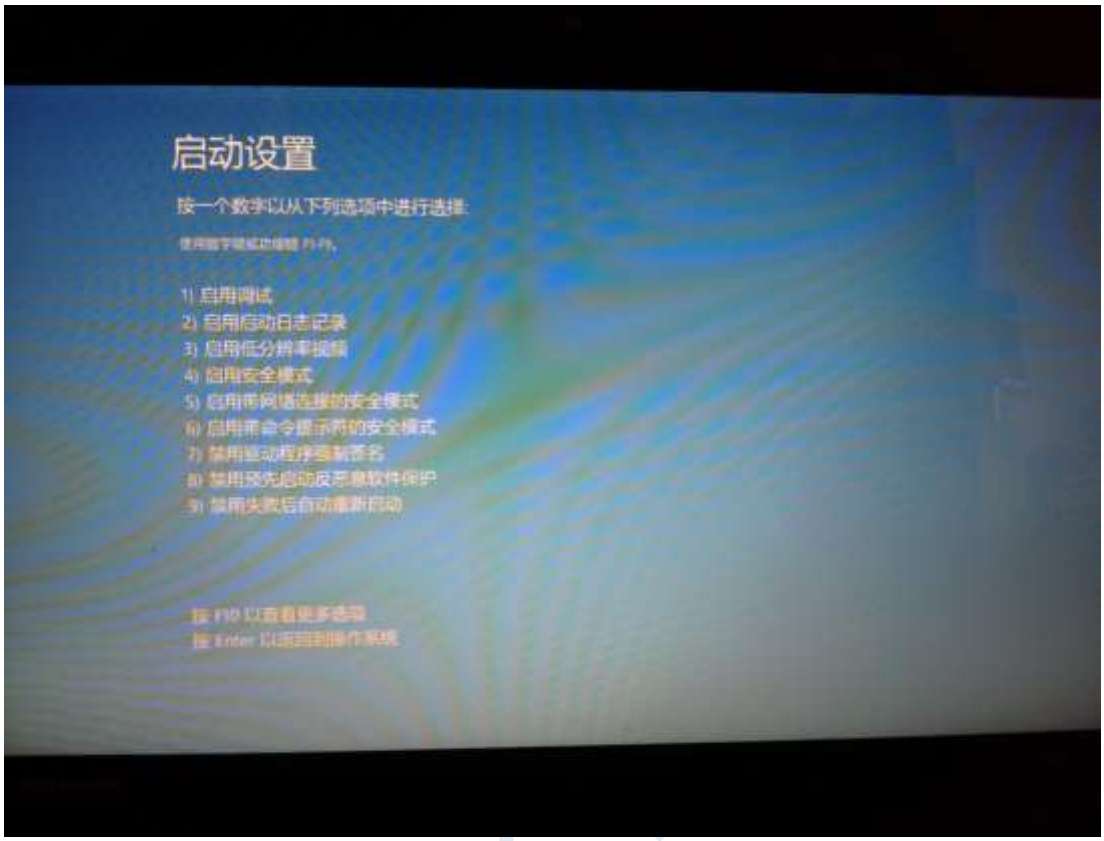
In the "Advanced Options" window below, select "Startup Settings".



In the "Startup Settings" window below, click the "Restart" button to restart the computer.



After the computer restarts, it will enter the "Startup Settings" window automatically as shown in the figure below. Press the number key "7" or press the function key "F7" to select "Disable driver forcing signature" to start.



After booting to Windows 8, follow the Windows 8 (32-bit) installation method to complete the driver installation.

Installation Instructions in Windows 10 (64-bit)

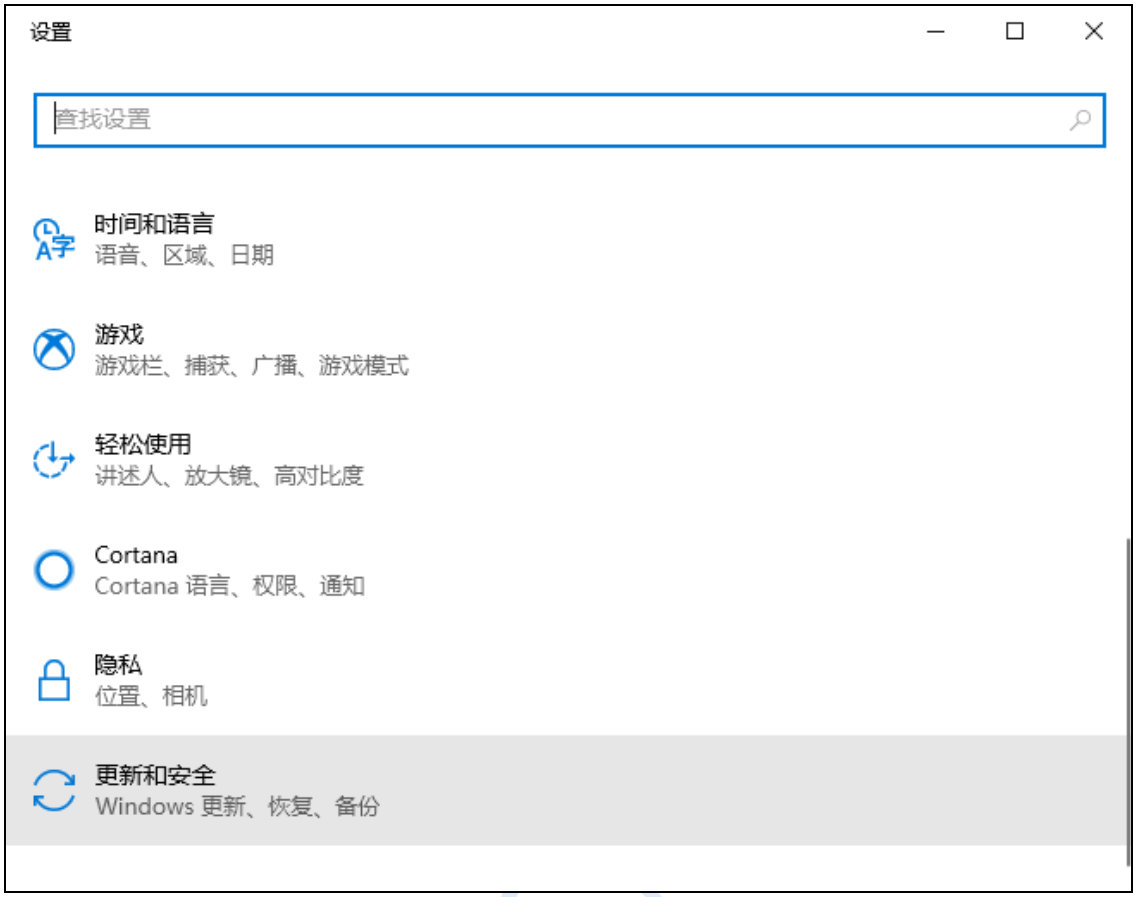
By default, the driver without digital signature cannot be successfully installed in Windows 10 64-bit operating system. So, you need to follow the steps below before installing the STC-USB driver, skip the digital signature temporarily, and the installation will be successful.

Before installing the driver, you need to extract the "STC-USB Driver" folder to the hard disk from the STC-ISP download software package downloaded from the STC official website. Prepare the chip with USB download function, but don't connect the computer firstly.

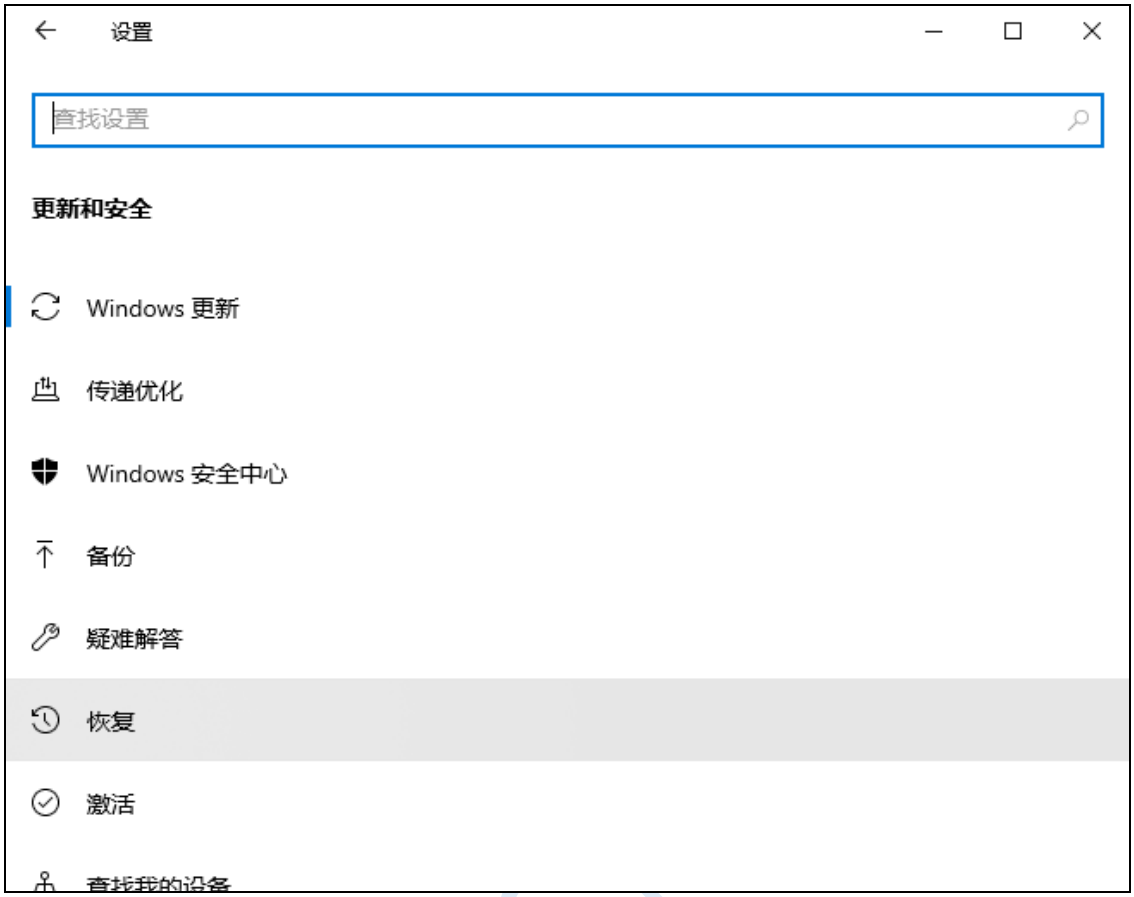
Right-click on the "Start" menu and select the "Settings" option.



Then select the "Update and Security" item in the settings window.



Then select the "Restore" item in the settings window.



In the recovery window, click the "Restart Now" button in the "Advanced Startup" item.



Before the computer restarts, the system will enter the following boot menu firstly and select the "Troubleshooting" item.



Select "Advanced Options" in the troubleshooting window.



Then select "View more recovery options".



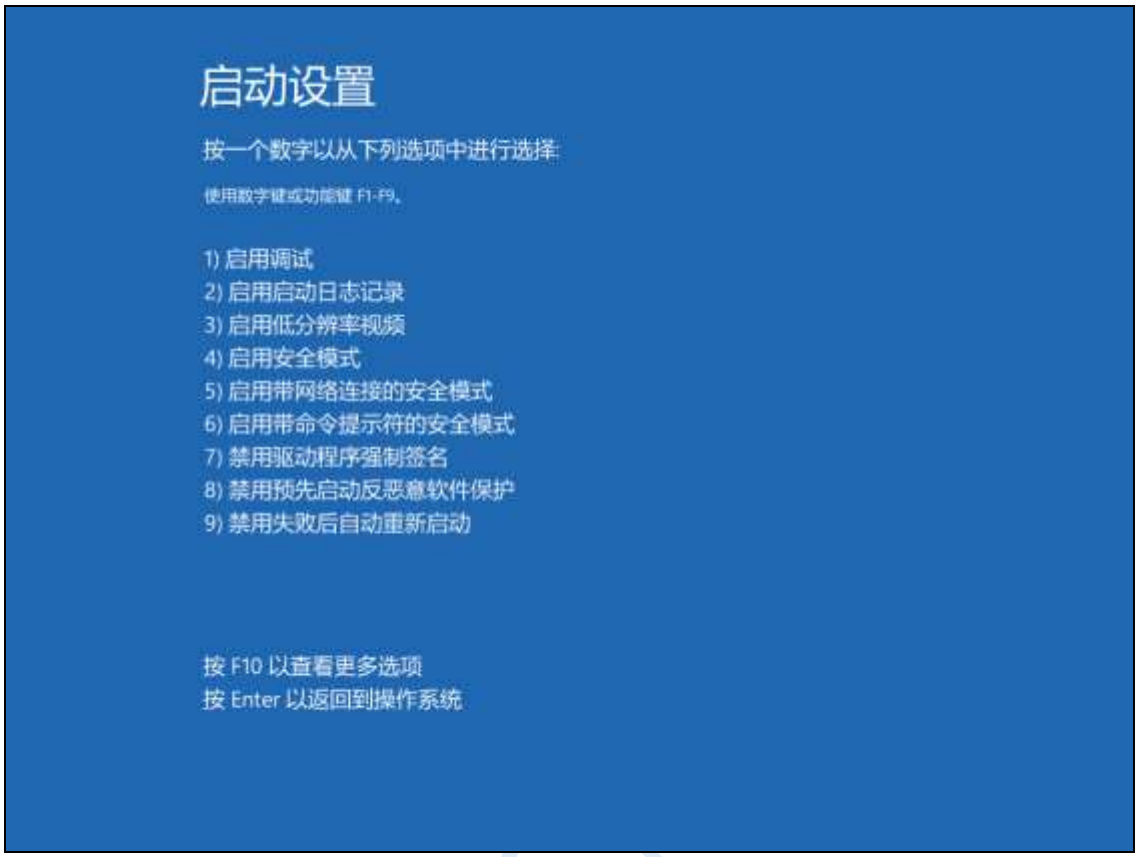
Select the "Startup Settings" item.



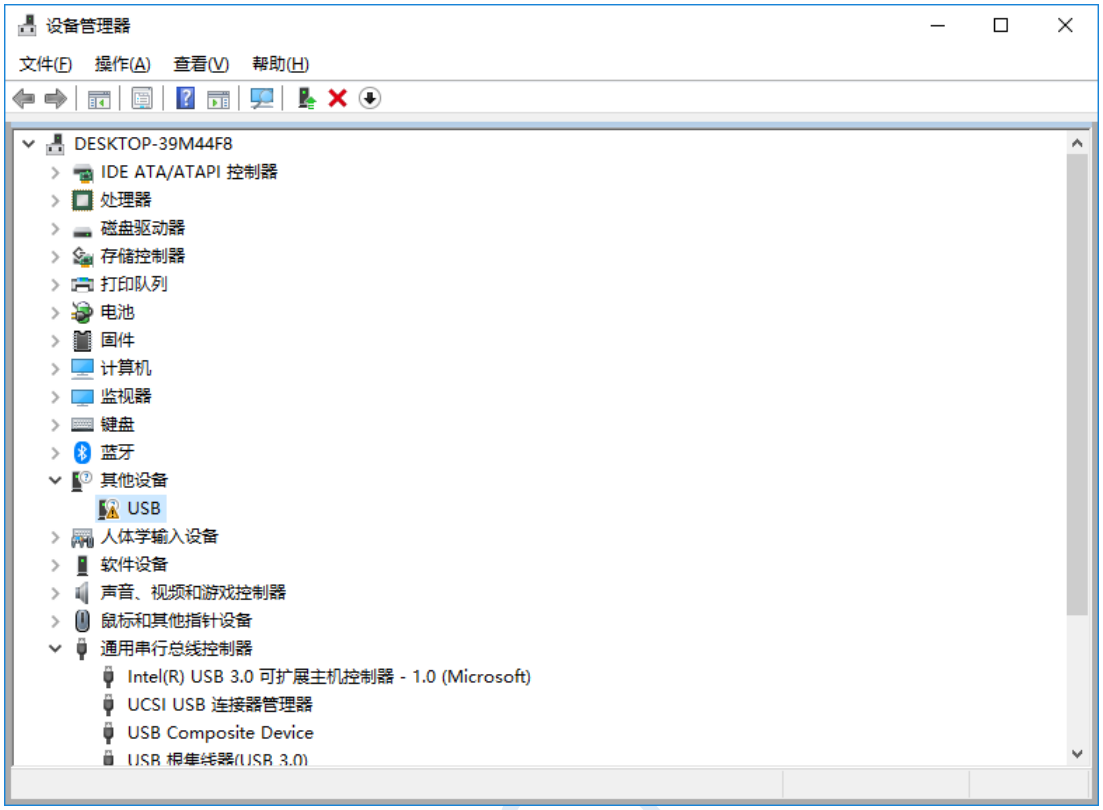
When the following screen appears, click the "Restart" button to restart the computer.



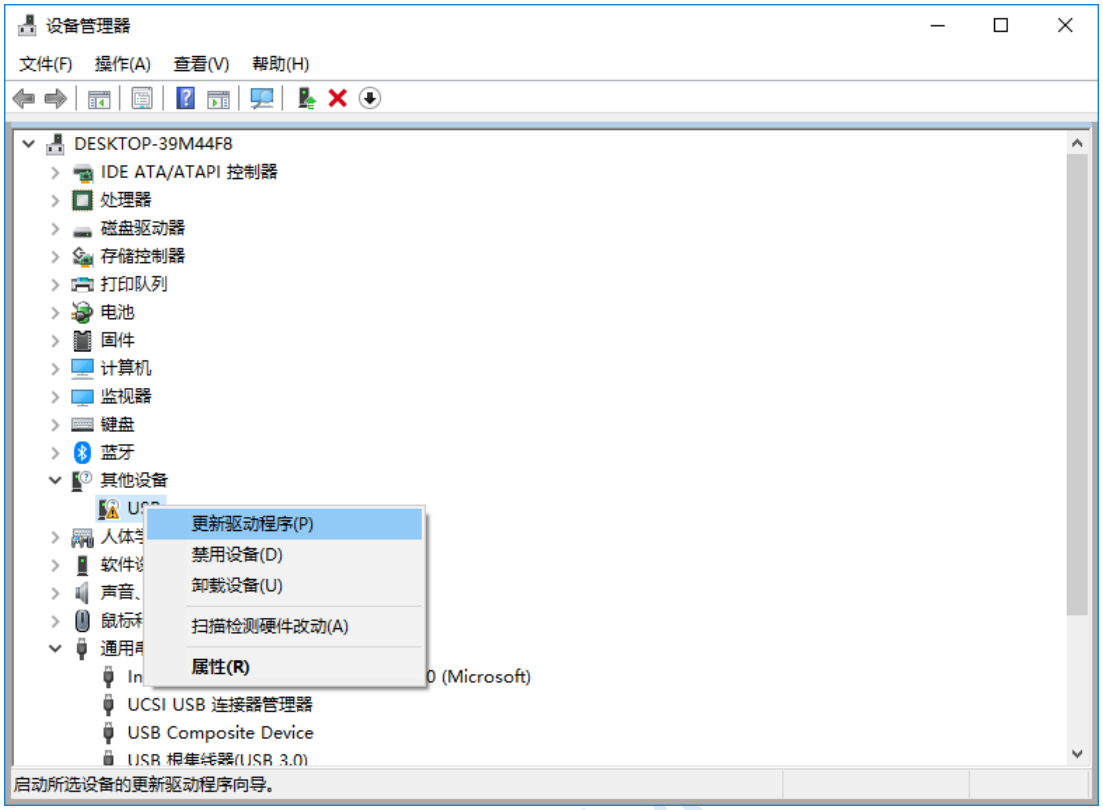
After the computer restarts, the "Startup Settings" window will pop up. Press the "F7" button to select the "Prohibit driver forcing signature" item.



After the computer is started, connect the prepared chip to the computer with a USB cable and open the "Device Manager". Now, the driver has not yet been installed, so it will appear as an unknown device with an exclamation point in the Device Manager.



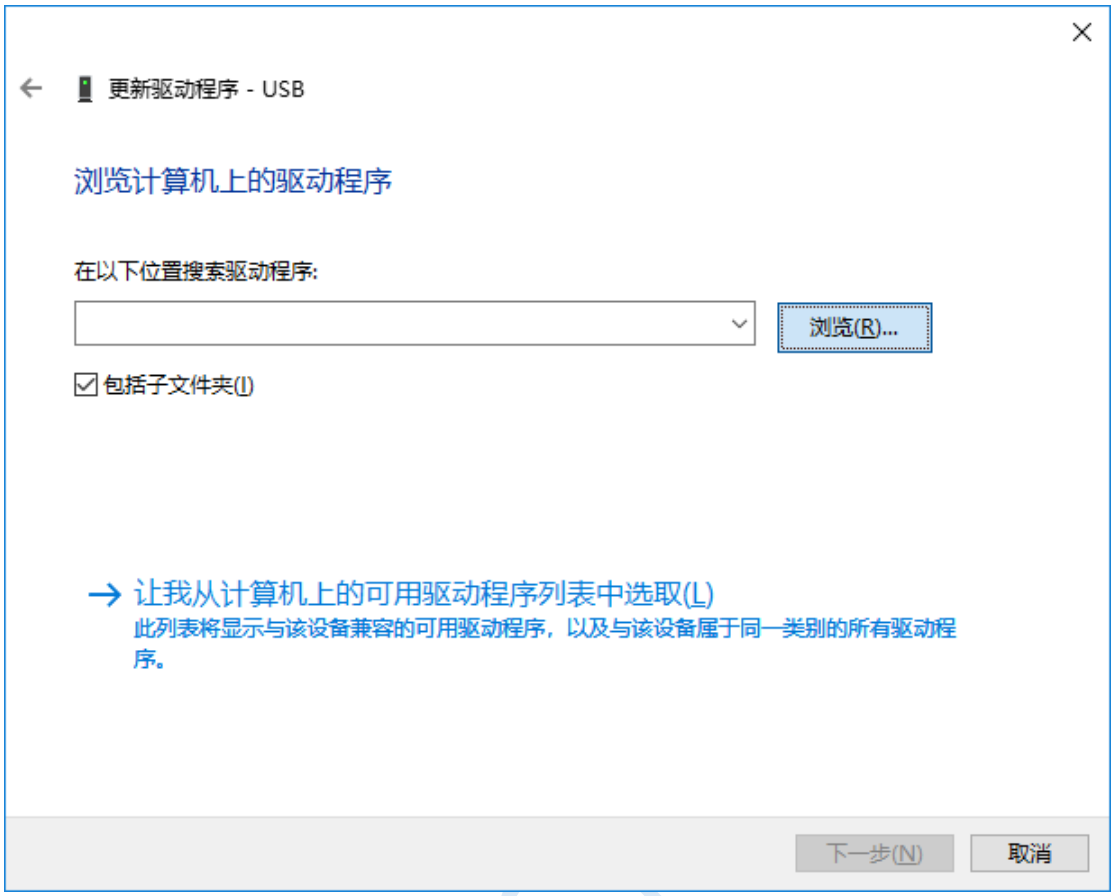
Right-click the unknown device and select "Update Driver" from the right-click menu.



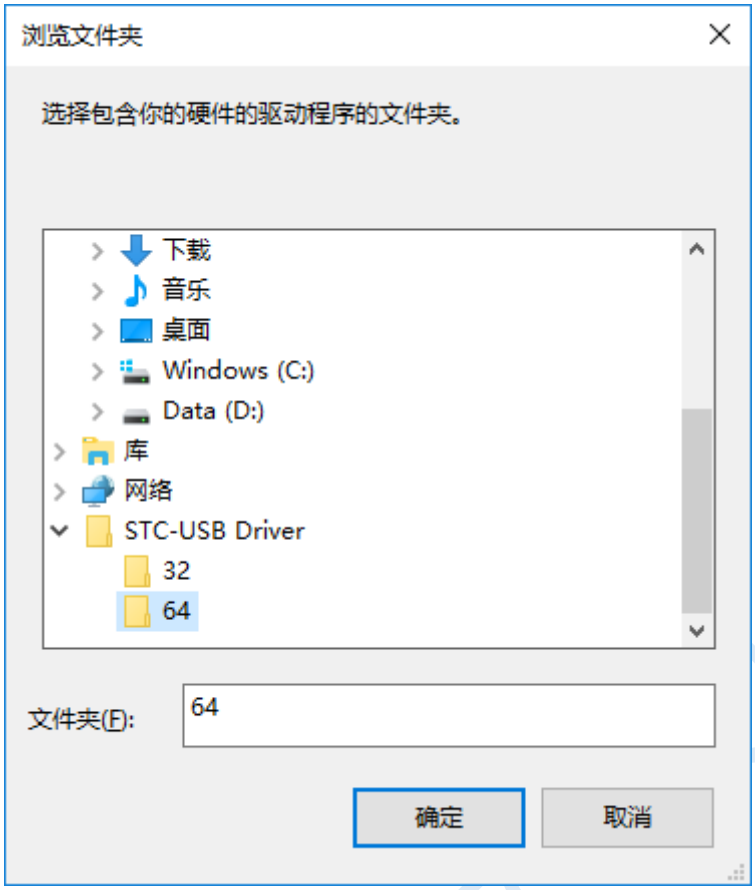
In the pop-up driver installer selection screen, select the "Browse my computer for driver software" item.



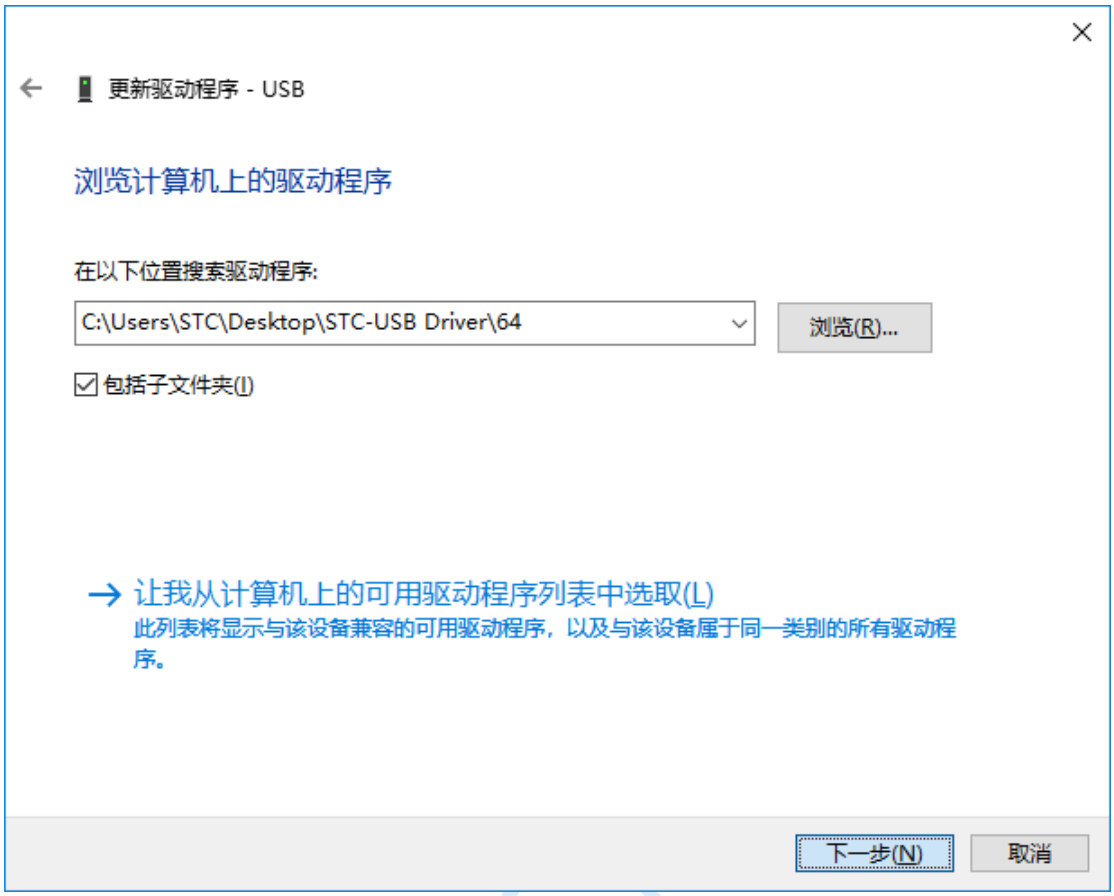
In the following window, click the "Browse" button.



Find the "STC-USB Driver" directory that was previously extracted to your hard disk, select the "64" directory in the directory, and press "OK" button.



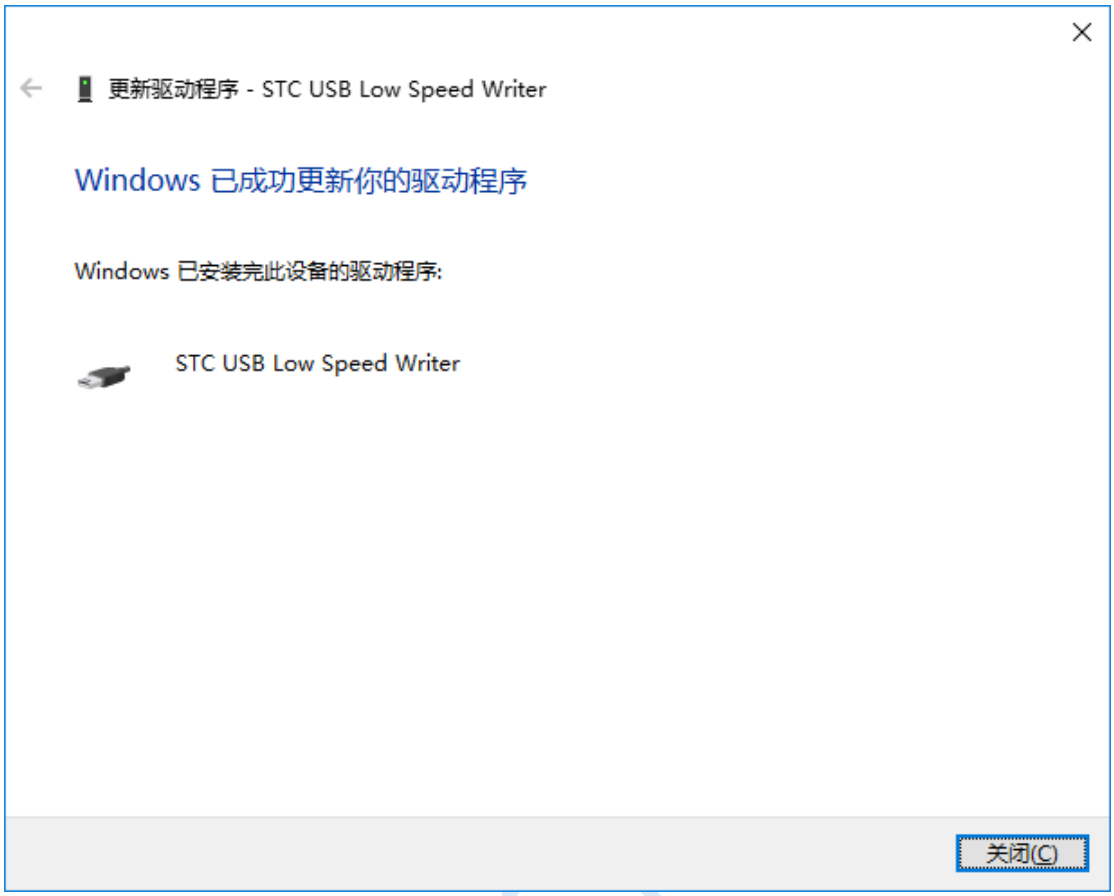
Click "Next" to start the driver installation.



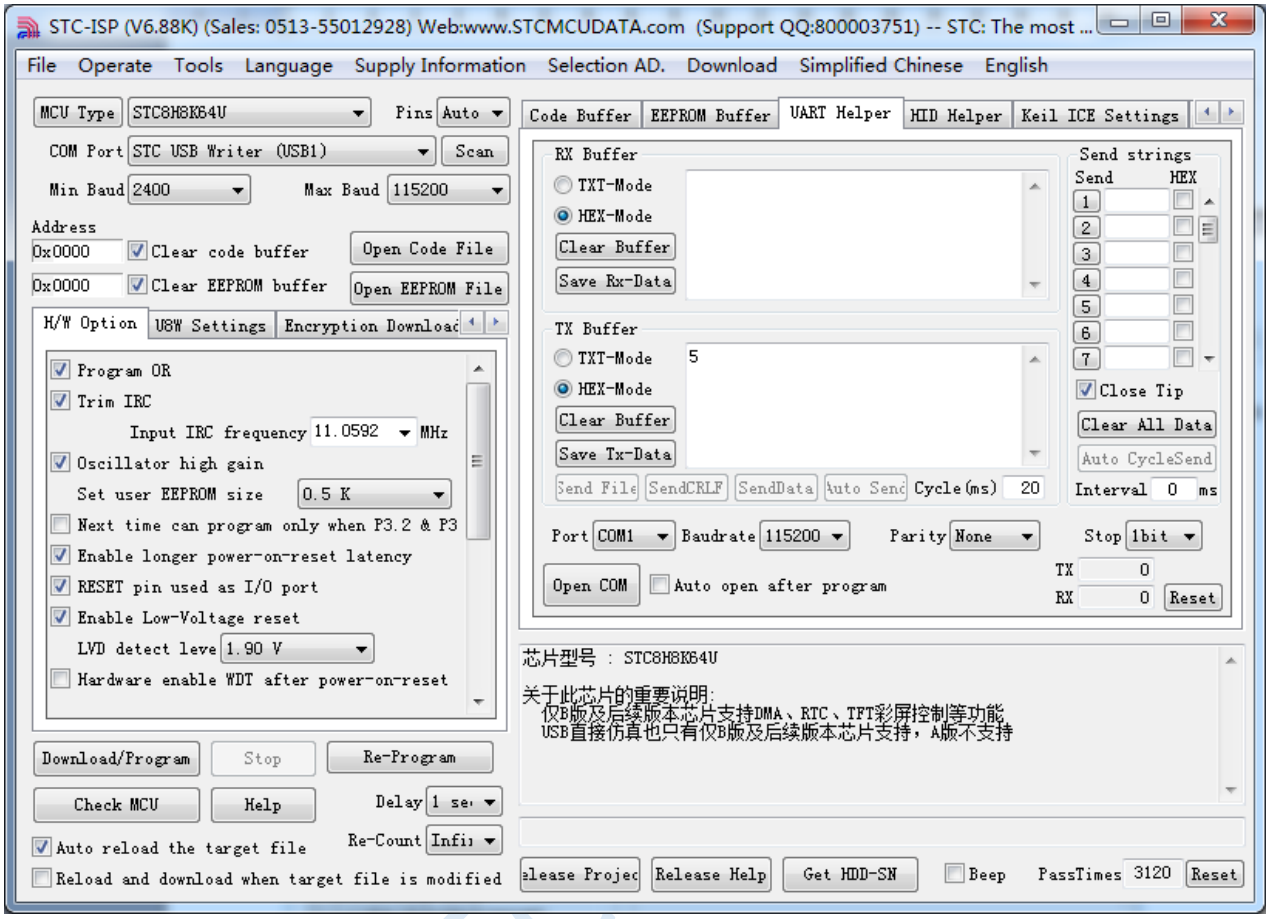
During the driver installation process, the following warning screen will pop up, select "Always install this driver software".



When the following screen appears, the driver is successfully installed.

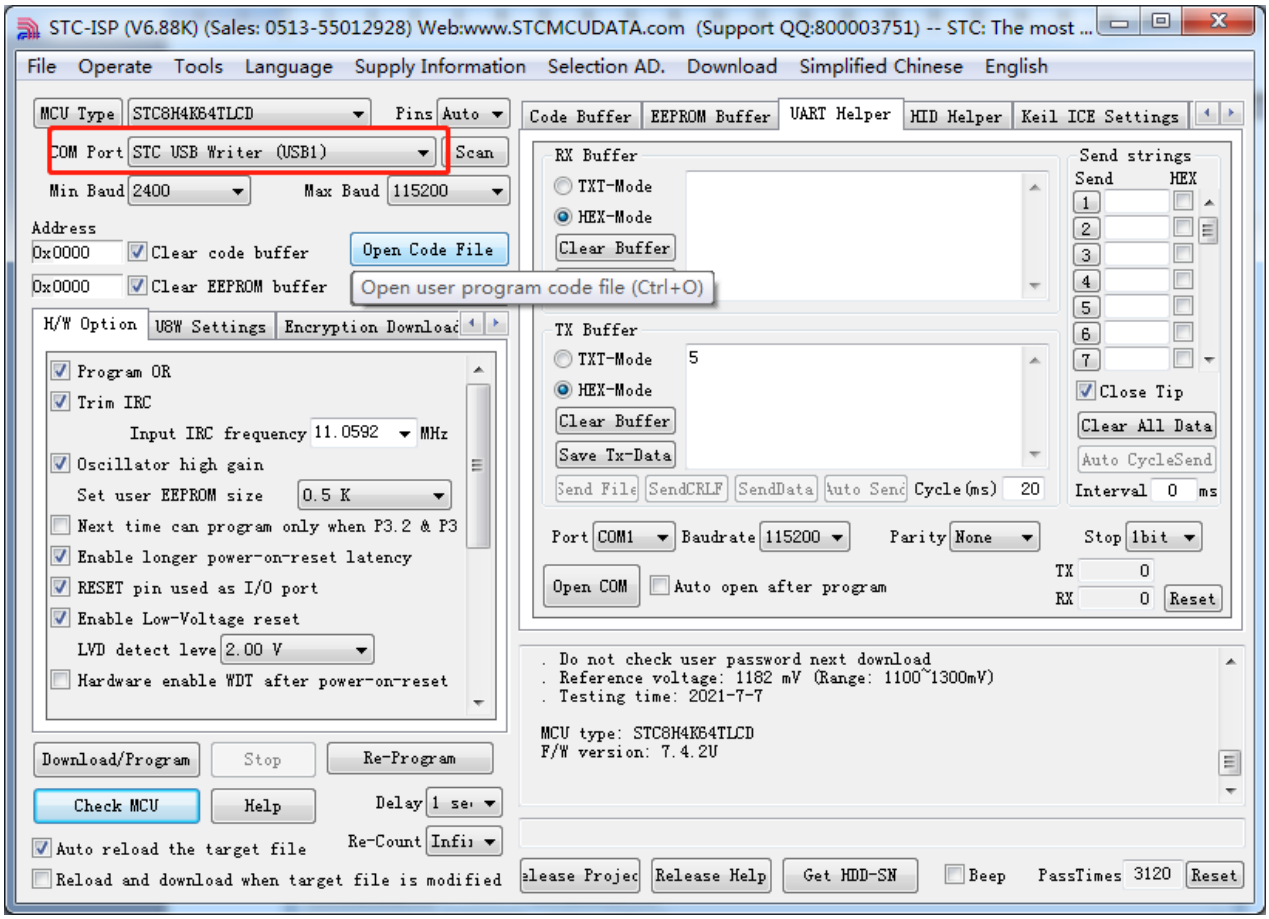


Back to the download software of STC-ISP, "STC USB Writer (USB1)" in the "Serial Port Number" drop-down list is selected automatically at this time, you can use USB for ISP download.

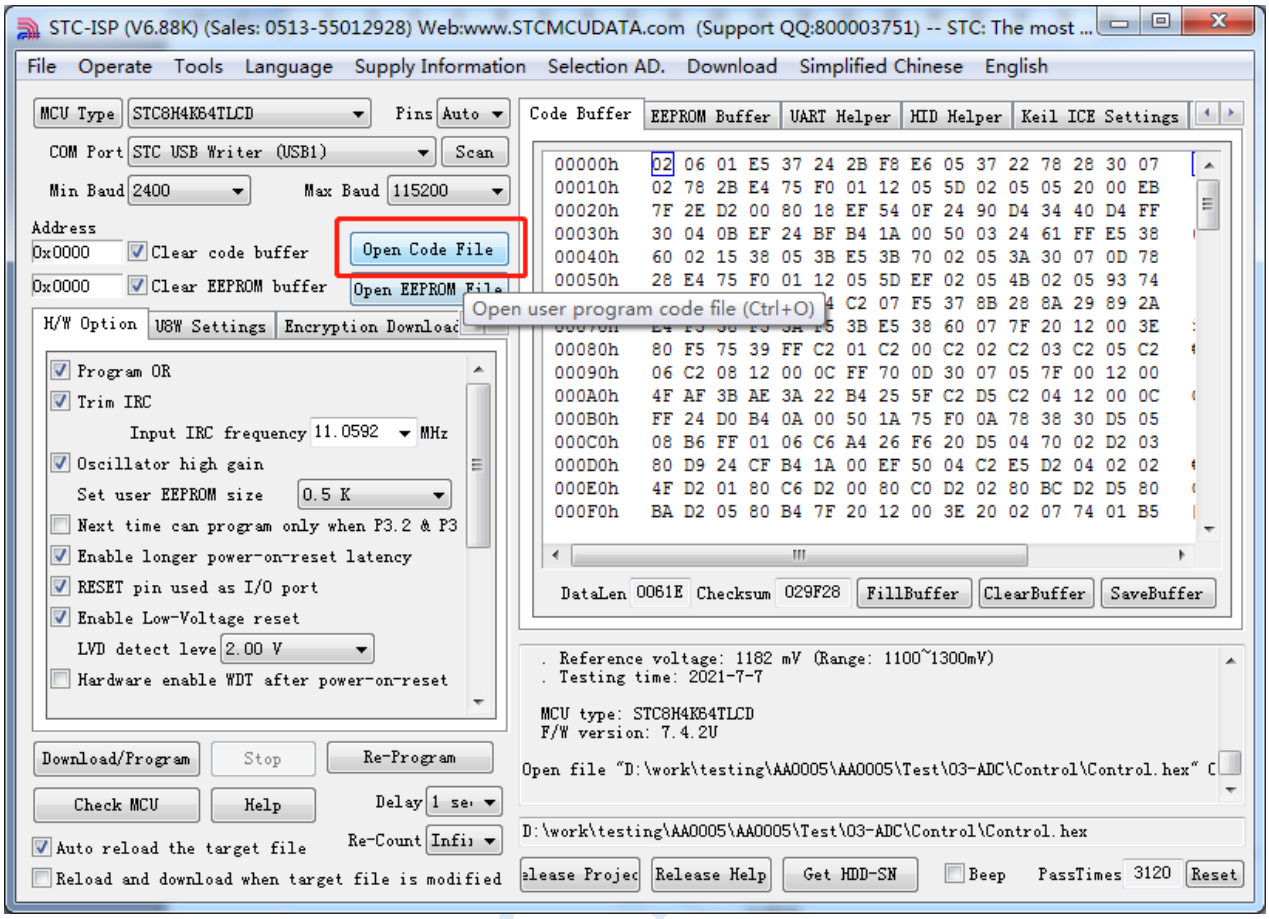


Appendix D Download Step Demo using USB

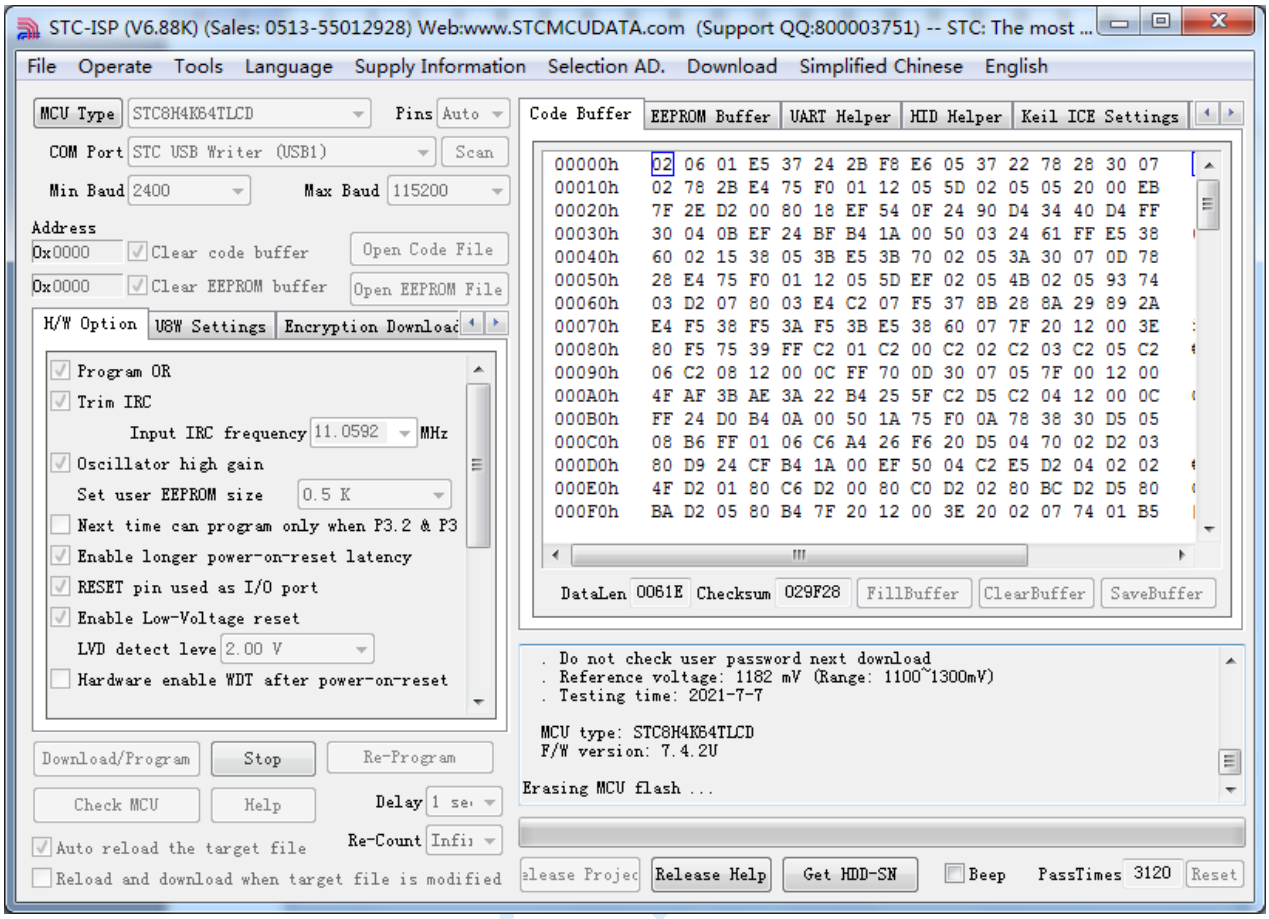
1. Refer to the application circuit diagram in P5.1.5 to connect the microcontroller firstly, and connect the P3.2 port of the target chip to GND, and then connect the system to the USB port on the PC side. Open the ISP to download the software, and the serial number of the downloaded software will search for the "STC USB Writer (USB1)" USB device automatically.



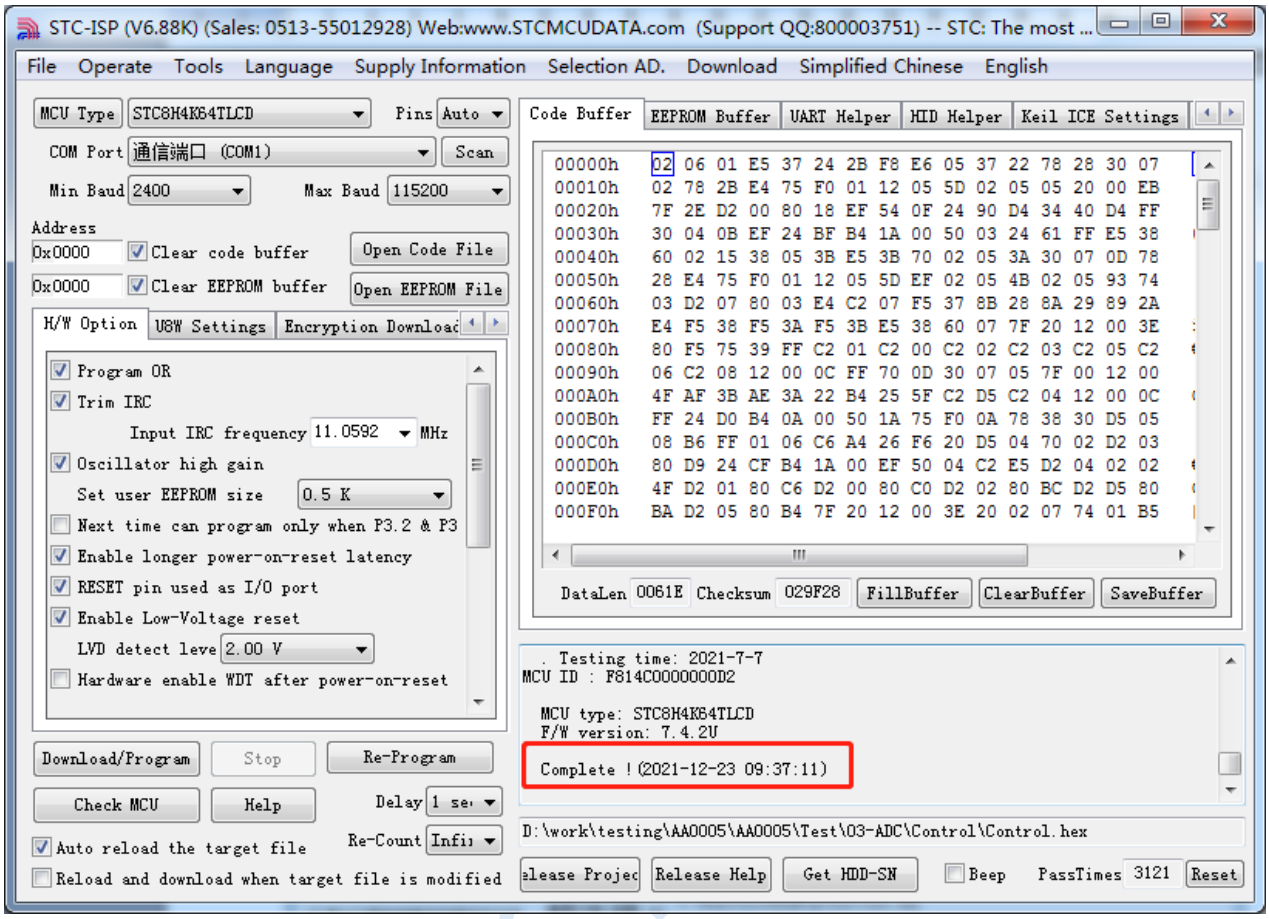
2. Open the user code program.



3. Click the "Download / Program" button to start downloading the user code.

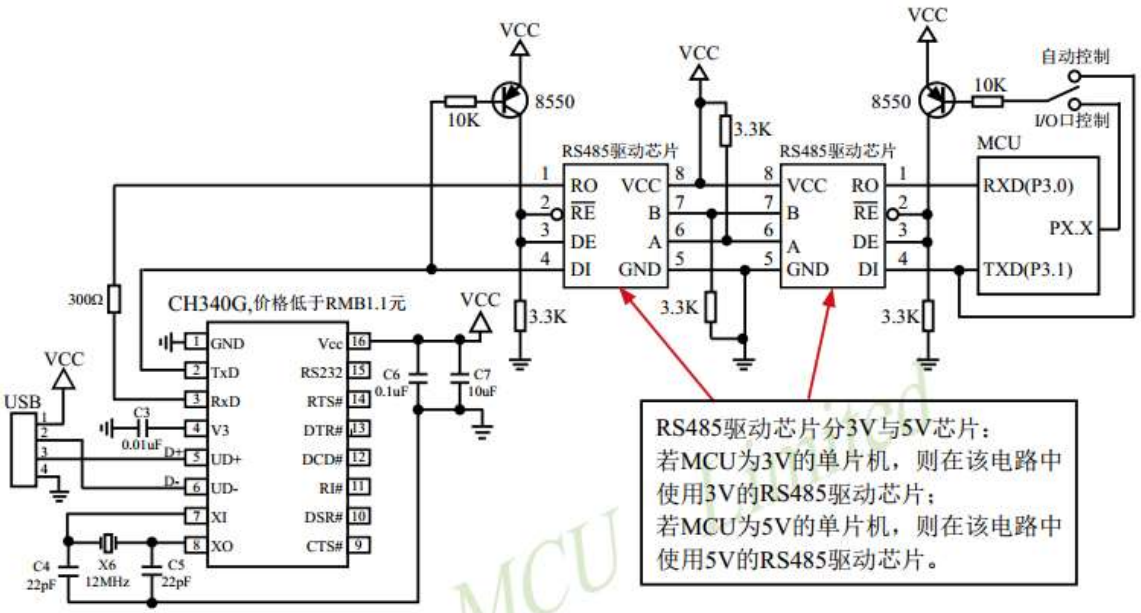


4. Until the prompt "Operation succeeded", it means that the program code download is complete.

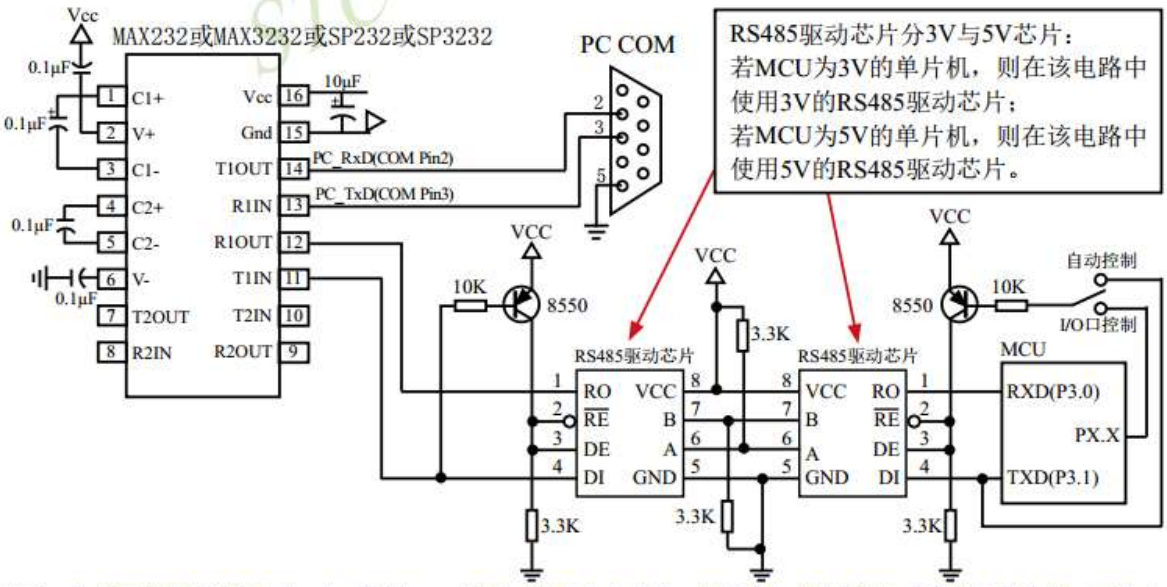


Appendix E Circuit of RS485 Automatic Control or I/O Control

1. RS485 control downloading circuit diagram using USB to serial connection computer (automatic control or I/O port control).



2. The circuit diagram of using RS232 to serial port conver to connect the computer to the RS485 control to download (automatic control or I / O port control).



注意：如果要设置单片机某个I/O口控制RS485发送或接收命令有效，则必须将单片机焊入电路板之前先用U8下载工具结合电脑ISP软件对该单片机进行“RS485控制”设置并烧录一下（如上节所述），否则将单片机实现不了RS485控制功能。

建议用户将本节所述“RS485控制下载线路图(自动控制或I/O口控制)”设计到您的用户板上

STC MCU

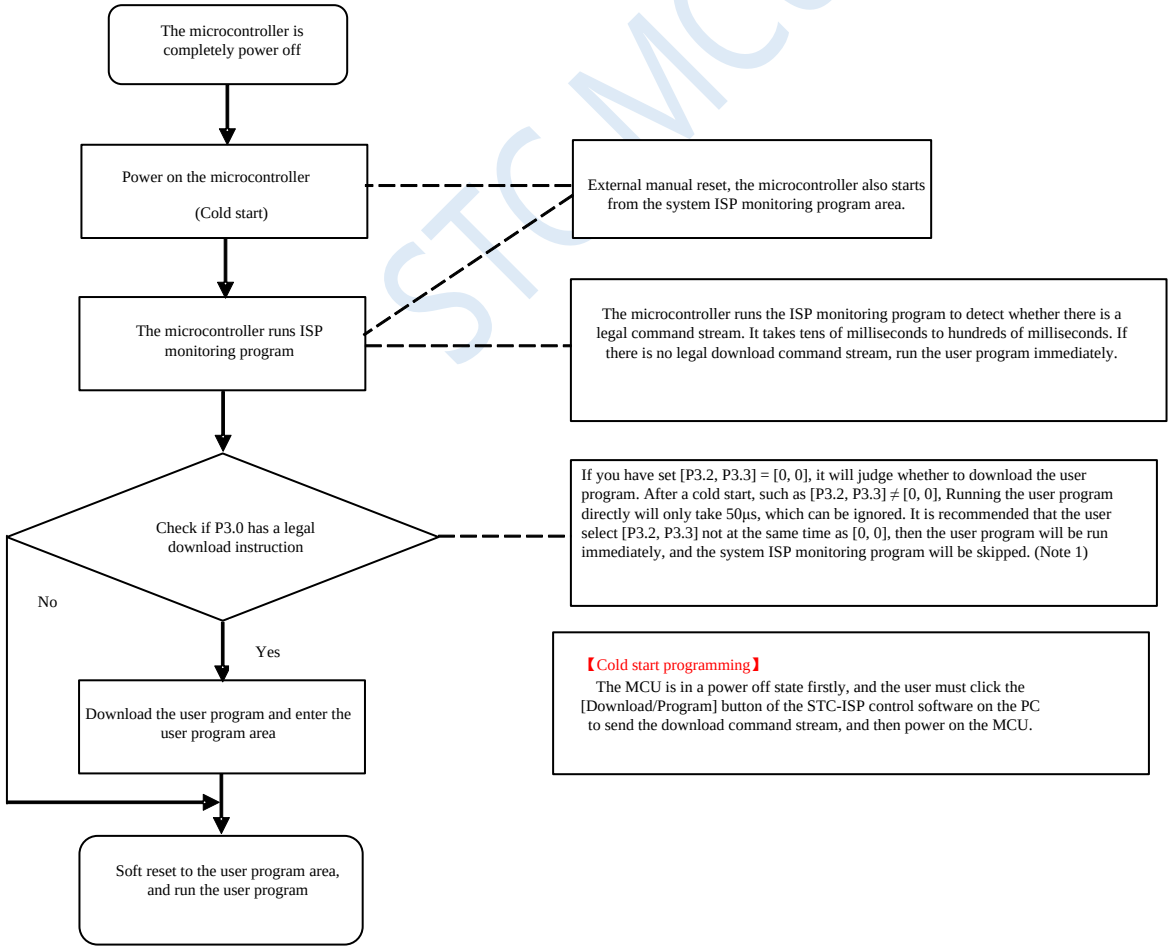
Appendix F STC tool instruction manual

F.1 Overview

U8W/U8W-Mini is a series of programming tools that integrates online download and offline download. STC Universal USB to Serial Port Tool is a programming tool that supports online download and online simulation.

Tool type	Online Download	Offline download	Burner download	Online simulation	Price (RMB)
U8W	support	support	support	Need to set pass-through mode	100
U8W-Mini	support	support	not support	Need to set pass-through mode	50
Universal USB to serial port	support	not support	not support	support	30

F.2 In-system programmable (ISP) process description



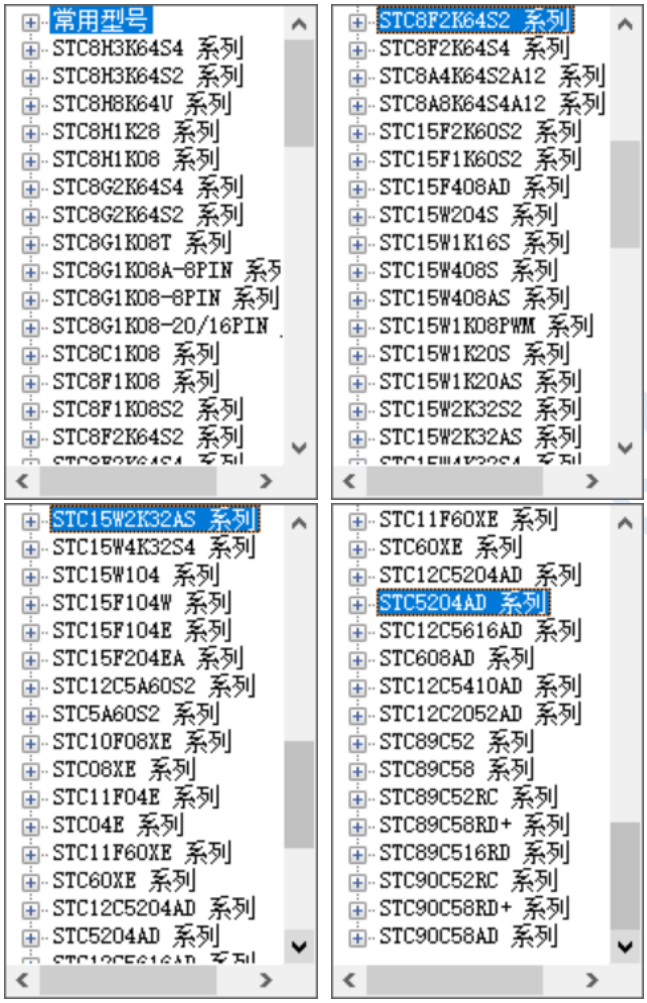
Note: Because [P3.0, P3.1] is used for download/simulation (download/simulation interface is only available [P3.0, P3.1]), it is recommended that users put serial port 1 on P3.6/P3.7 Or P1.6/P1.7, if the user does not want to switch and insists on using P3.0/P3.1 to work or communicate as the serial port 1, be sure to check the "Next

time cold restart" on the software when downloading the program The program can be downloaded only when P3.2/P3.3 is 0/0". 【Note 1】

[Note 1]: The programming protection pins of STC15, STC8 series and later new chips are P3.2/P3.3, and the programming protection pins of earlier chips are P1.0/P1.1.

F.3 USB type online/offline download tool U8W/U8W-Mini

The application range of U8W/U8W-Min can support all current MCU series of STC, and the Flash program space and EEPROM data space are not restricted. It can support the following and the upcoming STC full range of chips.



The offline download tool can be used for downloading without the computer, and can be used for mass production and remote upgrades. The offline download board can support multiple functions such as automatic increment, download limit, and encrypted transmission of user programs.

The following picture shows the front and back views of U8W tools and the front and back views of U8W-Mini.



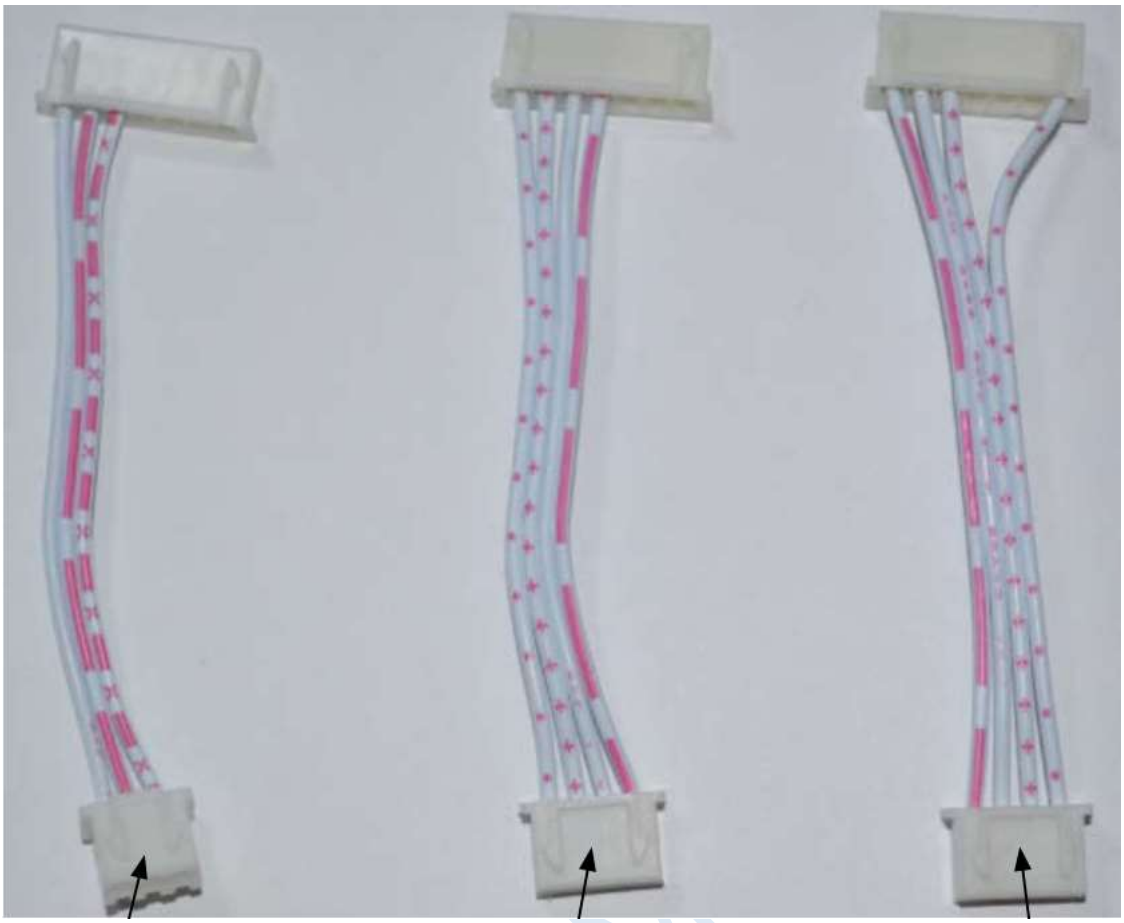
In addition, some of the following wires and tools are used together, such as:

(1) Two male USB cables (shown on the left in the figure below) and USB-Micro cables (shown on the right in the figure below):



Note: This USB cable is a USB enhanced cable specially customized by our company, which can ensure that the download can be successful when directly powered by USB. On the market, some relatively low-quality two-end male USB cables have much larger internal resistance, which leads to a large voltage drop (for example, the voltage of the USB is about 5.0V when it is empty. When using a low-quality USB cable to connect U8W/U8W-Mini/U8 /U8-Mini, the voltage to our download board may drop to 4.2V or lower, causing the chip to be in a reset state and fail to download successfully).

(2) The download cable connecting U8W/U8W-Mini and the user system (that is, the connecting cable between U8W/U8W-Mini and the target MCU on the user board), as shown in the figure below:



connection line for U8W/U8W-Mini and the user system power supply independently

U8W/U8W-Mini to the user system power supply connection line

User system to U8W/U8W-Mini Power supply connection line

F.3.1 Install U8W/U8W-Mini driver

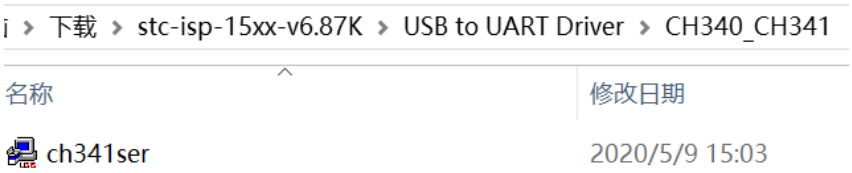
The U8W/U8W-Mini download board uses a CH340 USB-to-serial universal chip. This saves the trouble that some computers without a serial port must buy an additional USB-to-serial tool to download. However, CH340 is the same as other USB-to-serial tools, the driver must be installed before use.

Obtain the driver by downloading the STC-ISP software package.

The following is the download location of the STC-ISP software package provided on the STC official website (www.STCMCUDATA.com):



After downloading, decompress, the path of the CH340 driver installation package is stc-isp-15xx-v6.87K\USB to UART Driver\CH340_CH341:



Download the driver manually through STC's official website or in the latest STC-ISP download software.

Download the driver manually on the official website of STC or in the latest STC-ISP download software. The download link of the driver is: U8 programmer USB to serial port driver (http://www.stcmcu.com/STCISP/CH341SER.exe). The driver address on the website and STC-ISP download software is shown in the figure below.

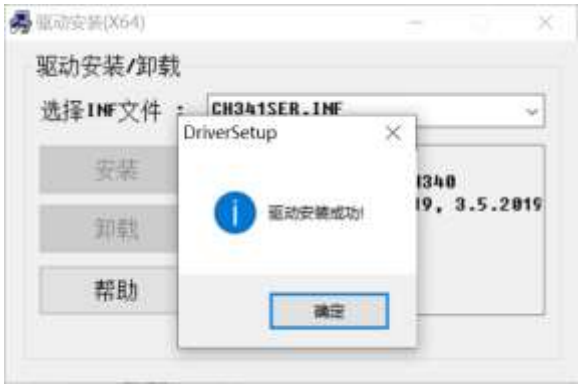


Install U8W/U8W-Mini driver

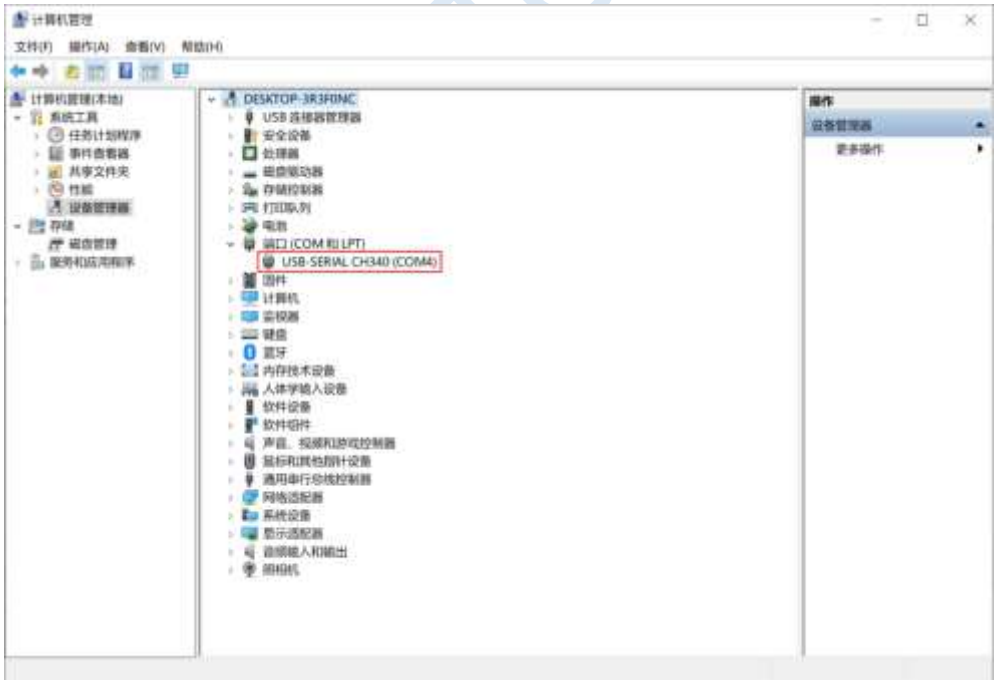
After the driver is downloaded to the machine, double-click the executable program and run it. The interface shown in the figure below appears, click the "Install" button to start the automatic driver installation:



Then the driver installation successful dialog box pops up, click the "OK" button to complete the installation:



Then use the USB cable provided by STC to connect the U8W/U8W-Mini download board to the computer, open the device manager of the computer, and under the port device class, if there is a device similar to "USB-SERIAL CH340 (COMx)", it means U8W/U8W-Mini can be used normally. As shown in the figure below (different computers may have different serial port numbers):

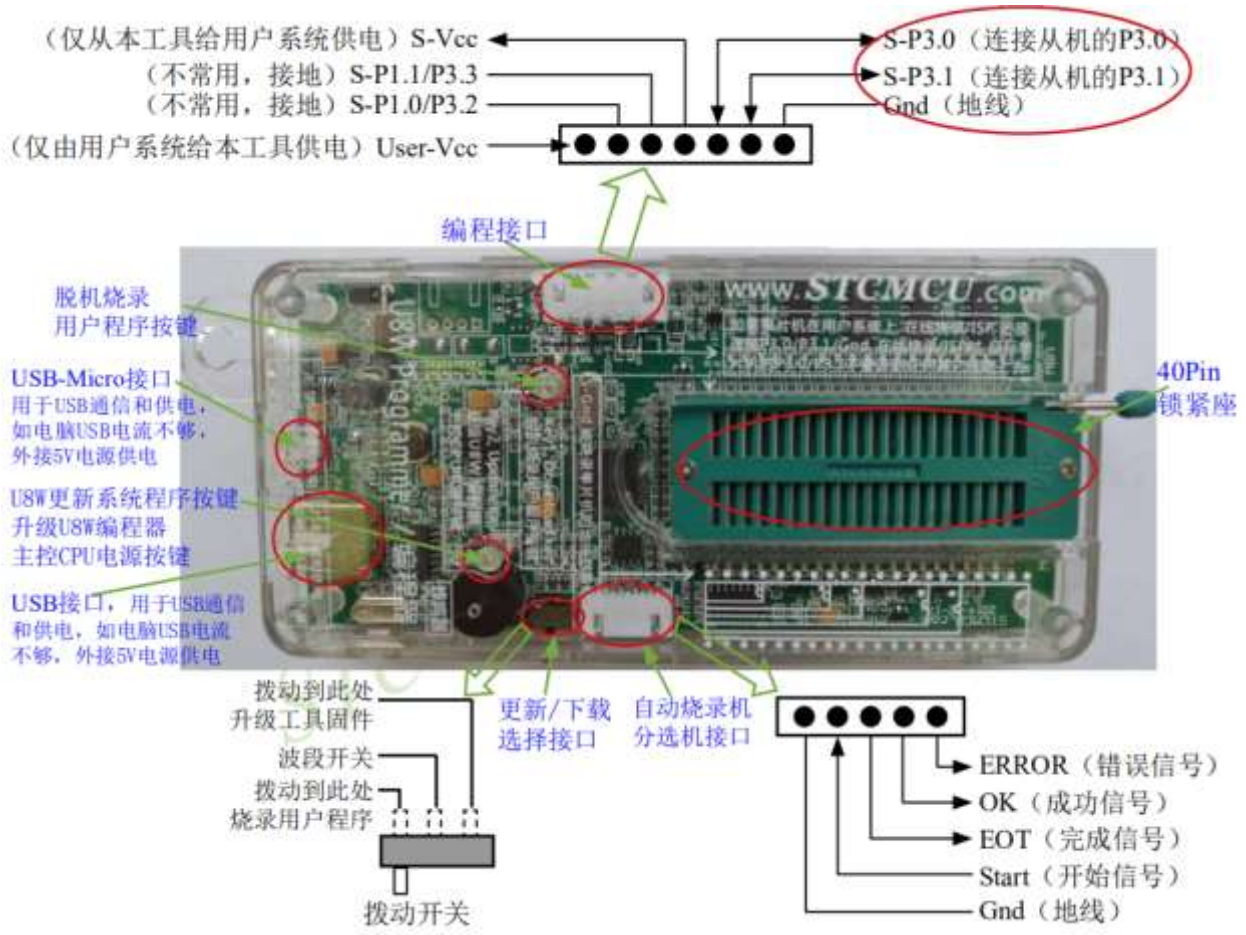


Note: When using STC-ISP to download software later, the selected serial port number must be the corresponding serial port number, as shown in the figure below:

F.3.2 U8W function introduction

The main interfaces and functions of the U8W tool are described in detail below:

If the MCU is on the user system, P3.0/P3.1/Gnd must be connected during online programming/ISP, and P3.0/P3.1 of the target MCU must not be connected to any other lines during online programming/ISP.



Programming interface: According to different power supply methods, use different download cables to connect the U8W download board and the user system.

U8W update system program button: It is used to update U8W tools, when there is a new version of U8W firmware, you need to press this button to update the U8W main control chip (note: you must set the toggle switch firstly on the update/download selection interface Toggle to upgrade tool firmware).

Offline download user program button: Start offline download button. Firstly, the PC downloads the offline code to the U8W board, and then uses the download cable to connect the user system to the U8W, and then press this button to start the offline download (the user code will also start downloading immediately every time the power is turned on).

Update/download selection interface: When you need to upgrade the underlying firmware of U8W, you need to turn this toggle switch to the firmware of the upgrade tool. When you need to program the target chip through U8W, you need to turn the toggle switch to Burn the user program.

(Please refer to the figure above for the connection method of the toggle switch)

Automatic burner/sorter interface: It is a control interface used to control the automatic burner/sorter for

automatic production.

F.3.3 Instructions for online download of U8W

The target chip is installed on the U8W locking base and the U8W connects to the computer for online download.

Firstly, use the USB cable provided by STC to connect the U8W to the computer, and then install the target microcontroller on the U8W in the direction shown in the figure below:

Then use STC-ISP to download the software to download the program, the steps are as follows:

1 Select the MCU model;
2 Select the number of pins. When the chip is directly installed on U8W to download, be sure to select the correct number of pins, otherwise the download will fail;
3 Select the serial port number corresponding to U8W;
4 Open the target file (HEX format or BIN format);
5 Set the hardware options;
6 Click the "Download/Program" button to start burning;
7 The step information of the burning process is displayed, and the message "Operation successful!" is displayed after the burning is completed.
When there is the version number information of the output download board and the corresponding information of the plug-in Flash in the information box, it means that the U8W download tool has been correctly detected. During the downloading process, the 4 LEDs on the U8W download tool will be displayed in a marquee mode. After the download is complete, if the download is successful, the 4 LEDs will be on and off at the same time; if the download fails, all the 4 LEDs will be off.
It is recommended that users use the latest version of STC-ISP to download the software (please always pay attention to the update of the STC-ISP download software on the STC official website http://www.STCMCUDATA.com. It is strongly recommended that users download the latest version of the software on the official website http://www.STCMCUDATA. com).

The target chip is connected to U8W through the user system leads and U8W is connected to the computer for online online download.

Firstly, use the USB cable provided by STC to connect the U8W to the computer, and then connect the U8W to the target microcontroller of the user system through the download line. The connection method is shown in the following figure:

Then use STC-ISP to download the software to download the program, the steps are as follows:
1. Select the MCU model;
2. Select the serial port number corresponding to U8W;
3. Open the target file (HEX format or BIN format);
4. Set the hardware options;
5. Click the "Download/Program" button to start burning;

6. The step information of the burning process is displayed, and the message "Operation successful!" is displayed when the burning is completed.

When there is the version number information of the output download board and the corresponding information of the plug-in Flash in the information box, it means that the U8W download tool has been correctly detected.
During the downloading process, the 4 LEDs on the U8W download tool will be displayed in a marquee mode. After the download is complete, if the download is successful, the 4 LEDs will be on and off at the same time; if the download fails, all the 4 LEDs will be off.
It is recommended that users use the latest version of STC-ISP to download the software (please always pay attention to the update of the STC-ISP download software on the STC official website http://www.STCMCUDATA.com. It is strongly recommended that users download the latest version of the software on the official website http://www.STCMCUDATA. com).

F.3.4 Instructions for offline download of U8W

The target chip is installed on the U8W seat and locked and connected to the computer via USB to supply power to the U8W for offline download.

The steps to use USB to power U8W for offline download are as follows:
(1) Use the USB cable provided by STC to connect the U8W download board to the computer, as shown below:

(2) Set up in the STC-ISP download software according to the steps shown in the figure below:

1. Select the MCU model;
2. Select the number of pins. When the chip is directly installed on U8W to download, be sure to select the correct number of pins, otherwise the download will fail;
3. Select the serial port number corresponding to U8W;
4. Open the target file (HEX format or BIN format);
5. Set the hardware options;
6. Select the "U8W Offline/Online" tab, set the offline programming options, and pay attention to the S-VCC output voltage matching the target chip operating voltage;
Click the "Download user program to U8/U7 programmer for offline download" button;
7. The step information of the setting process is displayed, and the prompt "Operation successful!" is displayed after the setting is completed.

Follow the steps in the above figure, after the operation is completed, if the download is successful, it means that the user code and related setting options have been downloaded to the U8W download tool.
It is recommended that users use the latest version of STC-ISP to download the software (please always pay attention to the update of the STC-ISP download software on the STC official website http://www.STCMCUDATA.com. It is strongly recommended that users download the latest version of the software on the official website http://www.STCMCUDATA. com).
(3) Place the target MCU in the U8W download tool in the direction shown in the figure below, as shown in the figure below:

(4) Then press the button as shown in the figure below and release it to start offline downloading:

During the downloading process, the 4 LEDs on the U8W download tool will be displayed in a marquee mode. After the download is complete, if the download is successful, the 4 LEDs will be on and off at the same time; if the download fails, all the 4 LEDs will be off.

Offline download plug and play burning function introduction:
1. After completing the above steps (1) and (2), U8W is in the plug-and-play programming state by default when it is connected to the computer and powered on;
2. Put the chip into the programming socket according to the instructions in step (3). When the socket wrench is tightened, U8W will automatically start programming;
3. Display the burning process and burning result through the indicator light;
4. After the programming is completed, loosen the wrench and take out the chip;
5. Repeat steps 2, 3 and 4 for continuous programming, eliminating the need to press the button to trigger the programming action.

The target chip is connected to U8W by the user system lead and connected to the computer via USB to supply power to U8W for offline download.

The steps to use USB to power U8W for offline download are as follows:
(1) Use the USB cable provided by STC to connect the U8W download board to the computer, as shown below:

(2) Set up in the STC-ISP download software according to the steps shown in the figure below:
It is recommended that users use the latest version of STC-ISP to download the software (please always pay attention to the update of the STC-ISP download software on the STC official website http://www.STCMCUDATA.com. It is strongly recommended that users download the latest version of the software on the official website http://www.STCMCUDATA. com).

1. Select the MCU model;
2. Select the number of pins. When the chip is directly installed on U8W to download, be sure to select the correct number of pins, otherwise the download will fail;
3. Select the serial port number corresponding to U8W;
4. Open the target file (HEX format or BIN format);
5. Set the hardware options;
6. Select the "U8W Offline/Online" tab, set the offline programming options, and pay attention to the S-VCC output voltage matching the target chip operating voltage;
Click the "Download user program to U8/U7 programmer for offline download" button;
7. The step information of the setting process is displayed, and the prompt "Operation successful!" is displayed after the setting is completed.
Follow the steps in the above figure, after the operation is completed, if the download is successful, it means that the user code and related setting options have been downloaded to the U8W download tool.
(3) Then use the cable to connect the computer, connect the U8W download tool and the user system (target microcontroller) as shown in the following figure, and press the button shown in the figure and release it to start offline downloading:

During the downloading process, the 4 LEDs on the U8W download tool will be displayed in a marquee mode.

After the download is complete, if the download is successful, the 4 LEDs will be on and off at the same time; if the download fails, all the 4 LEDs will be off.

The target chip is connected to U8W by the user system lead, and the user system supplies power to U8W for offline download.

(1) First use the USB cable provided by STC to connect the U8W download board to the computer, as shown in the figure below:

(2) Set up in the STC-ISP download software according to the steps shown in the figure below:
It is recommended that users use the latest version of STC-ISP to download the software (please always pay attention to the update of the STC-ISP download software on the STC official website http://www.STCMCUDATA.com. It is strongly recommended that users download the latest version of the software on the official website http://www.STCMCUDATA. com).

1. Select the MCU model;
2. Select the number of pins. When the chip is directly installed on U8W to download, be sure to select the correct number of pins, otherwise the download will fail;
3. Select the serial port number corresponding to U8W;
4. Open the target file (HEX format or BIN format);
5. Set the hardware options;
6. Select the "U8W Offline/Online" tab, set the offline programming options, and pay attention to the S-VCC output voltage matching the target chip operating voltage;
Click the "Download user program to U8/U7 programmer for offline download" button;
7. The step information of the setting process is displayed, and the prompt "Operation successful!" is displayed after the setting is completed.

Follow the steps in the above figure, after the operation is completed, if the download is successful, it means that the user code and related setting options have been downloaded to the U8W download tool.
(3) Then connect U8W to the user system as shown in the figure below, supply power to the user system, and then start offline downloading:

During the downloading process, the 4 LEDs on the U8W download tool will be displayed in a marquee mode. After the download is complete, if the download is successful, the 4 LEDs will be on and off at the same time; if the download fails, all the 4 LEDs will be off.

The target chip is connected to U8W by the user system lead, and U8W and the user system are independently powered for offline download.

(1) Firstly, use the USB cable provided by STC to connect the U8W download board to the computer, as shown in the figure below:

(2) Set up in the STC-ISP download software according to the steps shown in the figure below:
It is recommended that users use the latest version of STC-ISP to download the software (please always pay attention to the update of the STC-ISP download software on the STC official website http://www.STCMCUDATA.com. It is strongly recommended that users download the latest version of the software on the official website http://www.STCMCUDATA. com).

1. Select the MCU model;
2. Select the number of pins. When the chip is directly installed on U8W to download, be sure to select the correct number of pins, otherwise the download will fail;
3. Select the serial port number corresponding to U8W;
4. Open the target file (HEX format or BIN format);
5. Set the hardware options;
6. Select the "U8W Offline/Online" tab, set the offline programming options, and pay attention to the S-VCC output voltage matching the target chip operating voltage;
Click the "Download user program to U8/U7 programmer for offline download" button;
7. The step information of the setting process is displayed, and the prompt "Operation successful!" is displayed after the setting is completed.

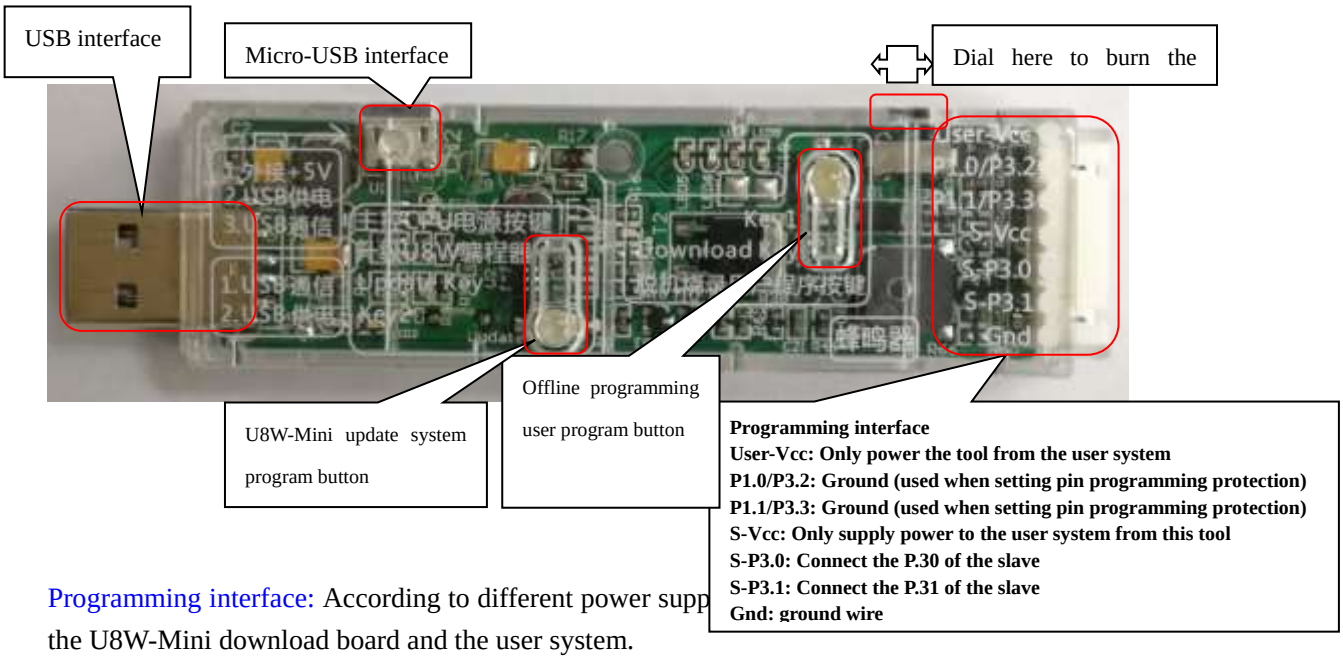
Follow the steps in the above figure, after the operation is completed, if the download is successful, it means that the user code and related setting options have been downloaded to the U8W download tool.

(3) Then connect U8W to the user system as shown in the figure, press and release the button shown in the figure, ready to start offline download, and finally power on/on the user system and download the user program begin:

During the downloading process, the 4 LEDs on the U8W download tool will be displayed in a marquee mode. After the download is complete, if the download is successful, the 4 LEDs will be on and off at the same time; if the download fails, all the 4 LEDs will be off.

F.3.5 U8W-Mini's function introduction

The main interfaces and functions of the U8W-Mini tool are described in detail below:



Programming interface: According to different power supp
the U8W-Mini download board and the user system.

U8W-Mini update system program button: It is used to update U8W-Mini tool, when there is a new version of

U8W firmware, you need to press this button to update the U8W-Mini main control chip (note: you must toggle the update/download switch on the interface to 'the upgrade tool firmware' firstly).

Offline download user program button: Start offline download button. Firstly, the PC downloads the offline code to the U8W-Mini, and then uses the download cable to connect the user system to the U8W-Mini, and then press this button to start the offline download (the user will also start downloading immediately every time the power is turned on).

Update/download selection interface: When you need to upgrade the underlying firmware of U8W-Mini, you need to move the toggle switch to the firmware of the upgrade tool. When you need to program the target chip through U8W-Mini, you need to toggle the switch to burn the user program. (Please refer to the figure above for the connection method of the toggle switch)

USB interface: The USB interface has the same function as the Micro-USB interface, and the user can connect one of the interfaces to the computer as needed.

F.3.6 U8W-Mini's online download instructions

The target chip is connected to the U8W-Mini through the user system lead, and the U8W-Mini is connected to the computer for online online download.

Firstly, use the USB cable provided by STC to connect the U8W-Mini to the computer, and then connect the U8W-Mini to the target MCU of the user system through the download cable. The connection method is shown in the following figure:

Then use STC-ISP to download the software to download the program, the steps are as follows:

1. Select the MCU model;
2. Select the number of pins. When the chip is directly installed on the U8W-Mini to download, be sure to select the correct number of pins, otherwise the download will fail;
3. Select the serial port number corresponding to U8W-Mini;
4. Open the target file (HEX format or BIN format);
5. Set the hardware options;
6. Click the "Download/Program" button to start burning;
7. The step information of the burning process is displayed, and the message "Operation successful!" is displayed when the burning is completed.

When there is the version number information of the output download board and the corresponding information of the external Flash in the information box, it means that the U8W-Mini download tool has been correctly detected.

During the downloading process, the 4 LEDs on the U8W-Mini download tool will be displayed in marquee mode. After the download is complete, if the download is successful, the 4 LEDs will be on and off at the same time; if the download fails, all the 4 LEDs will be off.

It is recommended that users use the latest version of STC-ISP to download the software (please always pay attention to the update of the STC-ISP download software on the STC official website http://www.STCMCUDATA.com. It is strongly recommended that users download the latest version of the software on the official website http://www.STCMCUDATA. com).

F.3.7 Instructions for offline download of U8W-Mini

The target chip is connected to the U8W-Mini by the user system lead and connected to the computer via USB to supply power to the U8W-Mini for offline download.

The steps to use USB to power U8W-Mini for offline download are as follows:
(1) Use the USB cable provided by STC to connect the U8W-Mini download board to the computer, as shown in the figure below:

(2) Set up in the STC-ISP download software according to the steps shown in the figure below:

1. Select the MCU model;
2. Select the number of pins. When the chip is directly installed on the U8W-Mini to download, be sure to select the correct number of pins, otherwise the download will fail;
3. Select the serial port number corresponding to U8W-Mini;
4. Open the target file (HEX format or BIN format);
5. Set the hardware options;
6. Select the "U8W Offline/Online" tab, set the offline programming options, and pay attention to the S-VCC output voltage matching the target chip operating voltage;
Click the "Download user program to U8/U7 programmer for offline download" button;
7. The step information of the setting process is displayed, and the prompt "Operation successful!" is displayed after the setting is completed.

Follow the steps in the above figure, after the operation is completed, if the download is successful, it means that the user code and related setting options have been downloaded to the U8W-Mini download tool.
It is recommended that users use the latest version of STC-ISP to download the software (please always pay attention to the update of the STC-ISP download software on the STC official website http://www.STCMCUDATA.com. It is strongly recommended that users download the latest version of the software on the official website http://www.STCMCUDATA. com).
(3) Then use the cable to connect the computer, connect the U8W-Mini download tool and the user system (target microcontroller) as shown in the figure below, and press the button shown in the figure and release it to start offline downloading:

During the downloading process, the 4 LEDs on the U8W-Mini download tool will be displayed in marquee mode. After the download is complete, if the download is successful, the 4 LEDs will be on and off at the same time; if the download fails, all the 4 LEDs will be off.

The target chip is connected to the U8W-Mini by the user system lead, and the U8W-Mini is powered by the user system for offline download.

(1) Firstly, use the USB cable provided by STC to connect the U8W-Mini download board to the computer, as shown below:

(2) Set up in the STC-ISP download software according to the steps shown in the figure below:

1. Select the MCU model;
2. Select the number of pins. When the chip is directly installed on the U8W-Mini to download, be sure to select the correct number of pins, otherwise the download will fail;
3. Select the serial port number corresponding to U8W-Mini;
4. Open the target file (HEX format or BIN format);
5. Set the hardware options;
6. Select the "U8W Offline/Online" tab, set the offline programming options, and pay attention to the S-VCC output voltage matching the target chip operating voltage;
Click the "Download user program to U8/U7 programmer for offline download" button;
7. The step information of the setting process is displayed, and the prompt "Operation successful!" is displayed after the setting is completed.

Follow the steps in the above figure, after the operation is completed, if the download is successful, it means that the user code and related setting options have been downloaded to the U8W-Mini download tool.

It is recommended that users use the latest version of STC-ISP to download the software (please always pay attention to the update of the STC-ISP download software on the STC official website http://www.STCMCUDATA.com. It is strongly recommended that users download the latest version of the software on the official website http://www.STCMCUDATA. com).

(3) Then use the cable to connect the computer, connect the U8W-Mini download tool and the user system (target microcontroller) as shown in the figure below, and press the button shown in the figure and release it to start offline downloading:

During the downloading process, the 4 LEDs on the U8W-Mini download tool will be displayed in marquee mode. After the download is complete, if the download is successful, the 4 LEDs will be on and off at the same time; if the download fails, all the 4 LEDs will be off.

The target chip is connected to the U8W-Mini by the user system lead, and the U8W-Mini and the user system are independently powered for offline download.

(1) Firstly, use the USB cable provided by STC to connect the U8W-Mini download board to the computer, as shown below:

(2) Set up in the STC-ISP download software according to the steps shown in the figure below:

1. Select the MCU model;
2. Select the number of pins. When the chip is directly installed on the U8W-Mini to download, be sure to select the correct number of pins, otherwise the download will fail;
3. Select the serial port number corresponding to U8W-Mini;
4. Open the target file (HEX format or BIN format);
5. Set the hardware options;
6. Select the "U8W Offline/Online" tab, set the offline programming options, and pay attention to the S-VCC output voltage matching the target chip operating voltage;
Click the "Download user program to U8/U7 programmer for offline download" button;
7. The step information of the setting process is displayed, and the prompt "Operation successful!" is displayed after the setting is completed.

Follow the steps in the above figure, after the operation is completed, if the download is successful, it means that the user code and related setting options have been downloaded to the U8W-Mini download tool.
It is recommended that users use the latest version of STC-ISP to download the software (please always pay attention to the update of the STC-ISP download software on the STC official website http://www.STCMCUDATA.com. It is strongly recommended that users download the latest version of the software on the official website http://www.STCMCUDATA. com).
(3) Then connect the U8W-Mini to the user system as shown in the figure below, and press the button shown in the figure first and then release it, ready to start offline download, and finally power on/on the user system to start starting download the user program:

During the downloading process, the 4 LEDs on the U8W-Mini download tool will be displayed in marquee mode. After the download is complete, if the download is successful, the 4 LEDs will be on and off at the same time; if the download fails, all the 4 LEDs will be off.

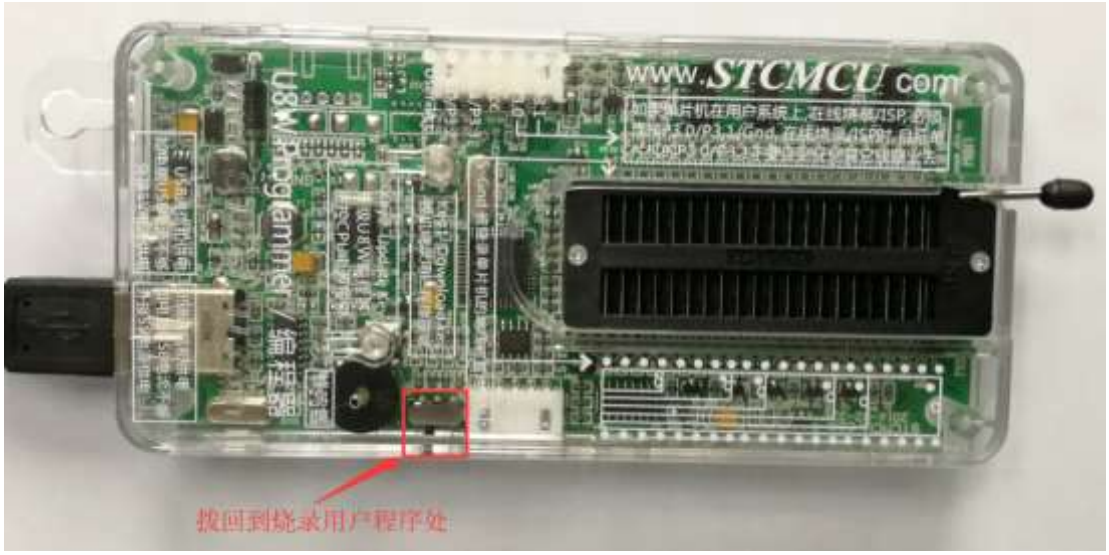
F.3.8 Make/Update U8W/U8W-Mini

The process of making a U8W/U8W-Mini download master is similar. To save space, the following uses U8W as an example to detail how to make a U8W download master.
Before making the U8W download master, you need to dial the "Update/Download Selection Interface" of the U8W download board to "Upgrade Tool Firmware", as shown in the figure below:

Then click the "Set U8W/U8-5V/U8-3V as the offline download master chip" button on the "U8W Offline/Online" page in the STC-ISP download program, as shown in the figure below: (Note: Be sure to select the serial port corresponding to U8W)

When the following screen appears, it indicates that the U8W control chip is completed:

After the production is completed, do not forget to dial the U8W "Update/Download Selection Interface" back to the "Burn User Program" mode, and power on the U8W download tool again, as shown in the figure below: (Otherwise, the programming will not be performed normally)



F.3.10 Set U8W/U8W-Mini to pass-through mode (can be used for simulation)

To use U8W/U8-Mini for simulation, you must set U8W/U8-Mini to pass-through mode firstly. The method for U8W/U8W-Mini to realize the USB-to-serial pass-through mode is as follows:

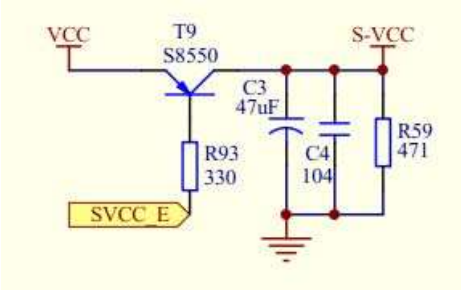
1. Firstly, the U8W/U8W-Mini firmware must be upgraded to v1.37 and above;
2. After the U8W/U8W-Mini is powered on, it is in normal download mode. At this time, press and hold the Key1 (download) button on the tool and do not release it. Press the Key2 (power) button again, and then release the Key2 (power) button. Release the Key1 (download) button again, and the U8W/U8W-Mini will enter the USB to serial port pass-through mode. (Press Key1→Press Key2→ Release Key2 → Release Key1);
3. The U8W/U8W-Mini tool that enters the pass-through mode is just a simple USB to serial port and does not have the offline download function. If you need to restore the original function of the U8W/U8W-Mini, you only need to press the Key2 (power) button separately again.

F.3.11 Reference circuit of U8W/U8W-Mini

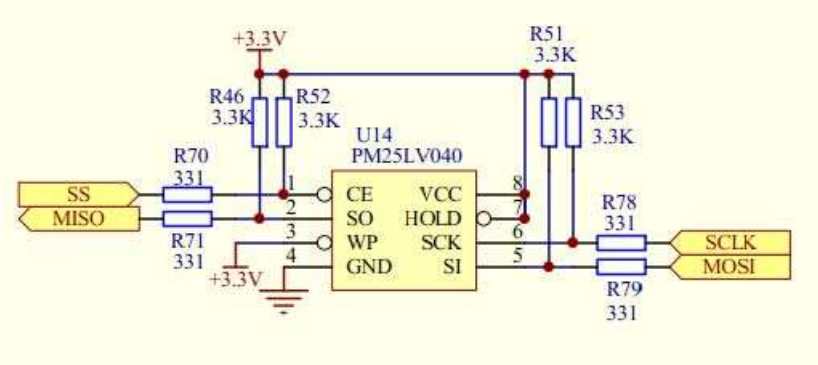
USB-type online/offline download board U8W/U8W-Mini provides users with the following common control interfaces:

Function	Port	Function description
Power control pin	P2.6	Active low
Download communication pin	P1.0	Serial port RXD, connect to the TXD of the target chip (P3.1)
	P1.1	Serial TXD, connect to the RXD of the target chip (P3.0)
Programming button	P3.6	Active low
Display	P3.2	LED1
	P3.3	LED2
	P3.4	LED3
	P5.5	LED4
External serial Flash control pin	P2.4	CE pin of Flash
	P2.2	SO pin of Flash
	P2.3	SI pin of Flash
	P2.1	SCLK pin of Flash
Automatic programming tool sorting machine signal	P3.6	Start signal
	P1.5	Completion signal
	P5.4	OK signal (Good product signal)
	P3.7	ERROR signal (Bad product signal)
BEEP control	P2.5	Active high (High level sounds)

Reference circuit diagram for power control part:

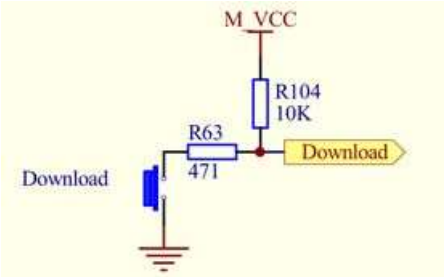


Reference circuit diagram of Flash control part:

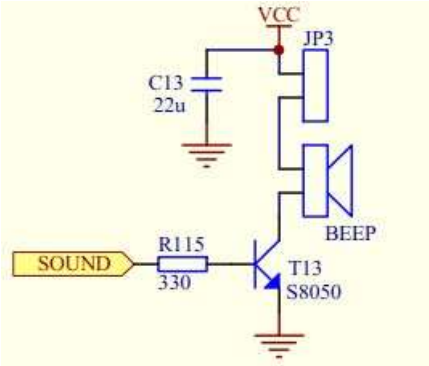


This Flash memory is required when the user program is larger than 41K.

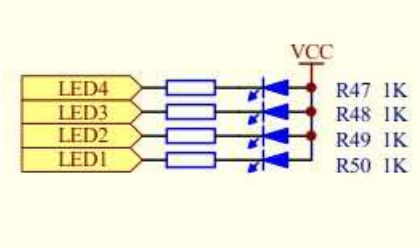
The reference circuit diagram of the button part:



Reference circuit diagram for the buzzer part:



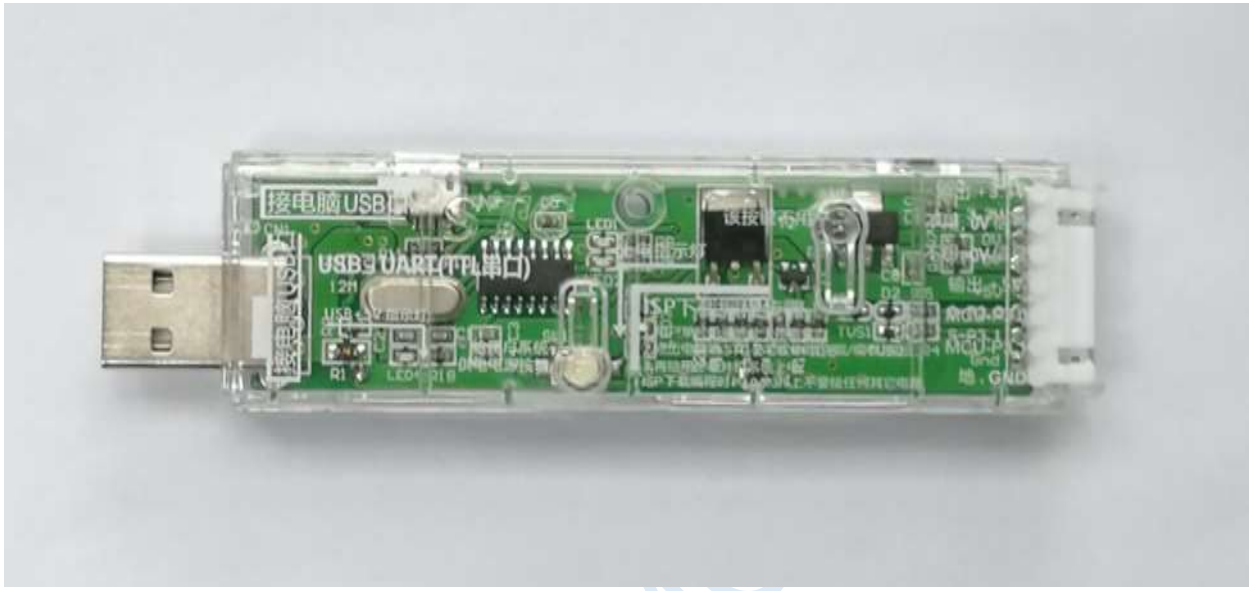
LED display part reference circuit diagram:



Reference circuit diagram of serial port communication pin connection part:

F.4.1 Appearance of STC Universal USB to Serial Tool

front:



back:



F.4.2 Layout diagram of STC general USB to serial tool

Here, the "power switch" needs to be explained:

The function of this button is the same as that of a self-locking switch. When the switch button is pressed for the first time, the switch is turned on and held, that is, self-locking. When the switch button is pressed for the second time, the switch turns off the power. In view of the characteristics of self-locking switches that are easily damaged during use, we designed a set of circuits that use light touch switches to replace self-locking switches to improve the service life of the tools.

For STC microcontrollers, if you want to perform ISP download, you must receive the serial port command at power-on reset to start executing the ISP program, so the correct steps to use the STC universal USB to serial tool to download the program to the MCU are:

1. Use STC universal USB to serial port tool to connect the MCU to be burned with the computer;
2. Open STC's ISP download software;
3. Select the MCU model;
4. Select the serial port corresponding to the STC universal USB to serial port tool;
5. Open the target file (HEX format or BIN format);
6. Click the "download/program" button in the ISP download software;
7. Press the "power switch" on the STC Universal USB to Serial Tool to supply power to the MCU and start downloading.

【Cold start burning】

In addition, the USB interface has the same function as the Micro-USB interface, and the user can connect one of the interfaces to the computer as needed.

The 0V signal pin of the programming interface has a 470ohm resistor grounded. It can be downloaded only when P1.0/P1.1=0/0 or P3.2/P3.3=0/0 is set. You can set P1.0, P1 .1 or P3.2, P3.3 are connected to the 0V signal pin.

STC MCU

F.4.3 STC Universal USB to Serial Tool Driver Installation

STC universal USB to serial port tool uses CH340 USB to serial chip (can plug in crystal oscillator, more accurate), just download the general CH340 serial driver and install it. The following is the CH341SER serial driver provided by STC official website (www.STCMCUDATA.com) download location:

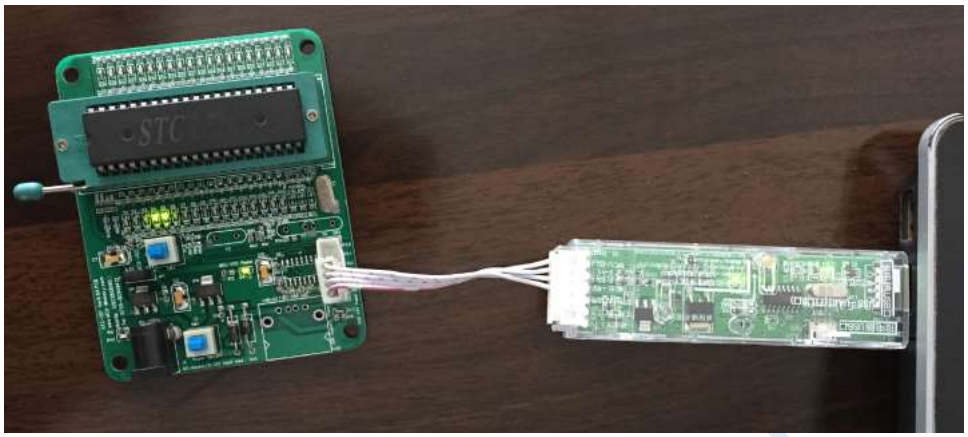
After downloading, decompress, the path of the CH340 driver installation package is stc-isp-15xx-v6.87K\USB to UART Driver\CH340_CH341:

Take the CH341SER serial port driver provided by STC official website as an example, double-click the "CH341SER.exe" installation package, and click the "Install" button on the pop-up main interface to start installing the driver:

Then the driver installation successful dialog box pops up, click the "OK" button to complete the installation:

F.4.4 Use the STC universal USB to serial tool to download the program to the MCU

1. Use STC universal USB to serial port tool to connect the MCU to be burned with the computer:



2. Open the STC-ISP software;
3. Select the model corresponding to the burning chip;
4. Select the serial port number recognized by the STC universal USB-to-Serial Tool (when the STC Universal USB-to-Serial Tool is correctly connected to the computer, the software will automatically scan and identify the serial port named "USB-SERIAL CH340 (COMx)", the specific COM The number will vary from computer to computer). When multiple USB-to-serial cables are connected to the computer, they must be selected manually;
5. Load the burning program;
6. Set burning options;
7. Click the "Download/Program" button;

8. When the prompt box in the lower right corner displays "Detecting target MCU...", press the "power switch" on the STC universal USB to serial port tool to supply power to the MCU, and then start downloading [cold start programming];

9. Wait for the download to end. If the download is successful, the prompt box in the lower right corner will display "Operation successful!".

F.4.5 Use STC universal USB to serial port tool to simulate user code

The current STC simulation is based on the Keil environment, so if you need to use the STC universal USB to serial port tool to simulate user code, you must install the Keil software.

After the Keil software is installed, you also need to install the STC simulation driver. The installation steps of STC's simulation driver are as follows:

Firstly, open STC-ISP to download the software;
Then click the "Add model and header file to Keil, add STC emulator driver to Keil" button in the "Keil simulation settings" page of the functional area on the right side of the software:

After pressing, the following screen will appear:

Locate the directory to the installation directory of the Keil software, and then confirm.
After the installation is successful, the following prompt box will pop up:

You can see the following files in the relevant directory of Keil, which means that the driver is installed correctly.

By default, the main control chip of STC is not a simulation chip and does not have a simulation function, so, if simulation is required, the main control chip of STC needs to be set as a simulation chip.
The steps of making a simulation chip are as follows:

Firstly, use the STC universal USB to serial port tool to connect the MCU to the computer;
Then open STC's ISP download software, and select the serial port number corresponding to the serial port tool in the serial port number drop-down list;
Select the MCU model;
Select the IRC frequency of the user program to run, and the frequency selected when making the simulation chip is consistent with the frequency set by the simulated user program, in order to achieve the real running effect.

Then click the "Set the selected target MCU as an emulation chip" button on the "Keil Simulation Settings" page in the right functional area of the software. After pressing, the following screen will appear:

Next, you need to press the "power switch" on the STC universal USB to serial port tool to power the MCU [cold start], and then you can start to make the simulation chip.
If the setting is successful, the following screen will appear:

Now, the simulation chip has been made successfully.
Next we open a project for simulation:

Then make the following project settings:

An additional note:
When a C language project is created and the startup file "STARTUP.A51" is added to the project, there is a macro definition named "IDATALEN", which is a macro used to define the size of IDATA. The default value is 128, which is 80H in hexadecimal, and it is also the size of IDATA that needs to be initialized to 0 in the startup file. When IDATA is defined as 80H, then the code in STARTUP.A51 will initialize the RAM of IDATA 00-7F to 0; similarly, if IDATA is defined as 0FFH, it will initialize the RAM of IDATA 00-FF to 0.

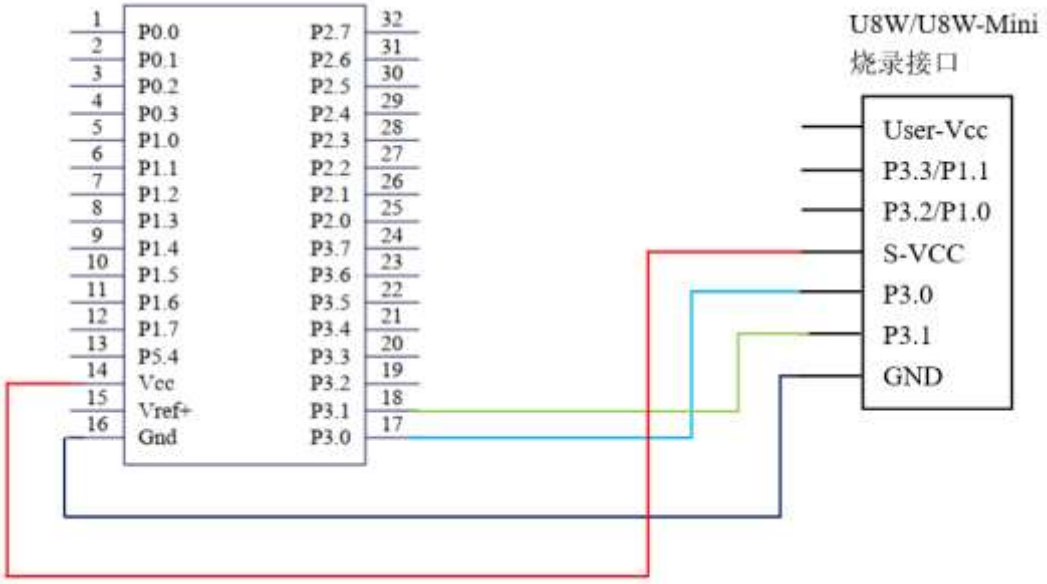
The IDATA size of the STC8H series microcontroller we selected is 256 bytes (00-7F DATA and 80H-FFH IDATA), but because the last 17 bytes of RAM have written ID numbers and related test parameters, If users need to use this part of data in the program, they must not define IDATALEN as 256.

Press the shortcut key "Alt+F7" or select "Option for Target ‘Target1’" in the menu "Project"
Configure the project in the "Option for Target ‘Target1’" dialog box:
Step 1. Enter the project setting page and select the "Debug" setting page;
Step 2. Select the hardware emulation "Use …" on the right;
Step 3. Select "STC Monitor-51 Driver" item in the simulation driver drop-down list;
Step 4. Click the "Settings" button to enter the serial port settings screen;
Step 5: Set the port number and baud rate of the serial port. The serial port number should be the serial port corresponding to the STC universal USB to serial port tool. The baud rate is generally 115200 or 57600.
Make sure to complete the simulation settings.
The detailed steps are shown in the figure below:

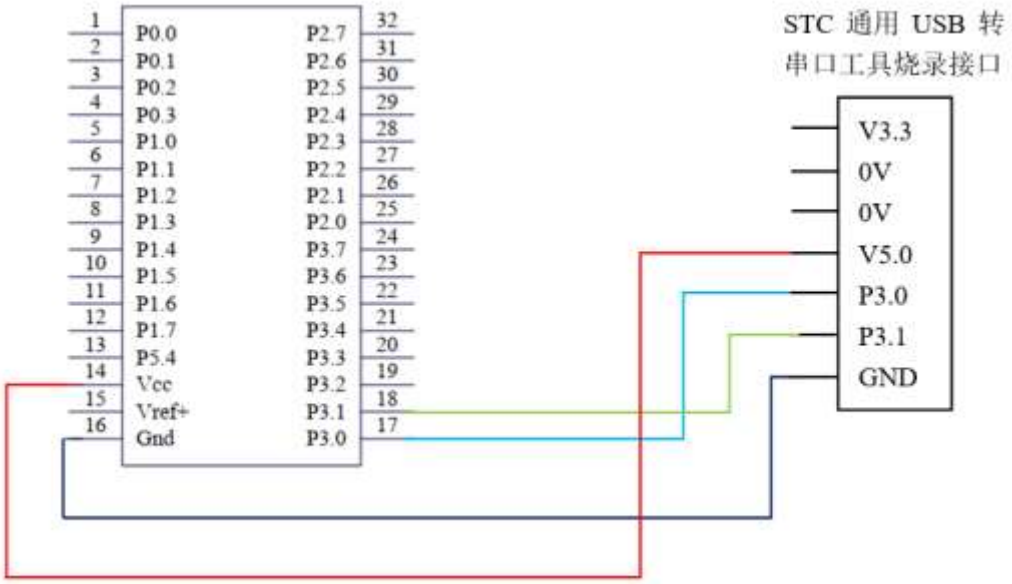
After finishing all the work above, you can press "Ctrl+F5" in Keil software to start simulation debugging.
If the hardware connection is correct, you will enter a debugging interface similar to the following, and display the current simulation driver version number and the current simulation monitoring code firmware version number in the command output window, as shown in the following figure:

During the simulation debugging process, you can perform multiple operations such as resetting, running at full speed, single stepping, and setting breakpoints.

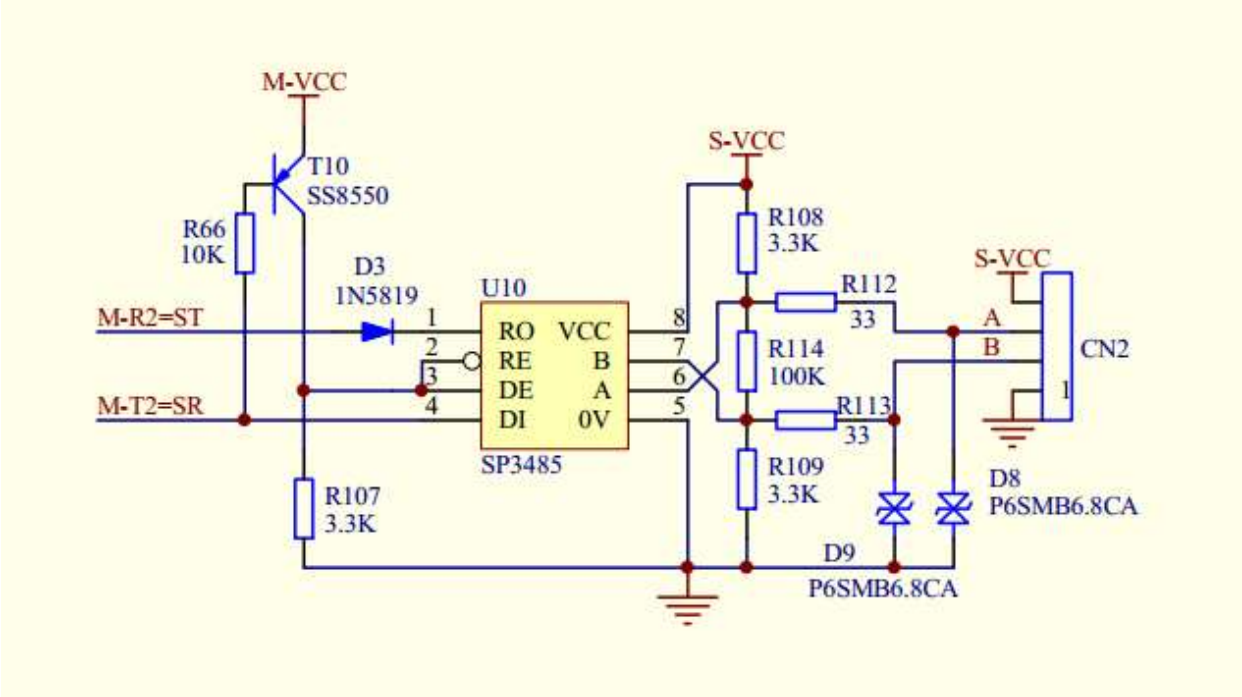
As shown in the figure above, multiple breakpoints can be set in the program, and the maximum number of breakpoint settings currently allowed is 20 (in theory, any number can be set, but setting too many breakpoints will affect the speed of debugging).



F.5.2 STC Universal USB to Serial Tool Application Reference Circuit Diagram



Appendix G Partial Circuit of RS485 in U8W Download Tool



BOM list:

Label	Model	Package	Note
U10	SP3485EN	SOP8	RS485 chip
R66	10K	0603	Resistor
R107	3.3K	0603	Resistor
R108	3.3K	0603	Resistor
R109	3.3K	0603	Resistor
R112	33R	0603	Resistor
R113	33R	0603	Resistor
R114	100K	0603	Resistor
T10	SS8550	SOT-23	PNP Triode
D3	1N5819	0603	Schottky diode
D8	P6SMB6.8CA	DO-214AA	TVS diode
D9	P6SMB6.8CA	DO-214AA	TVS diode
CN2		SIP4	Communication Interface

STC MCU

Appendix H ISP Download Starts Automatically After Receiving User Command While Running User Program (no Power-down)

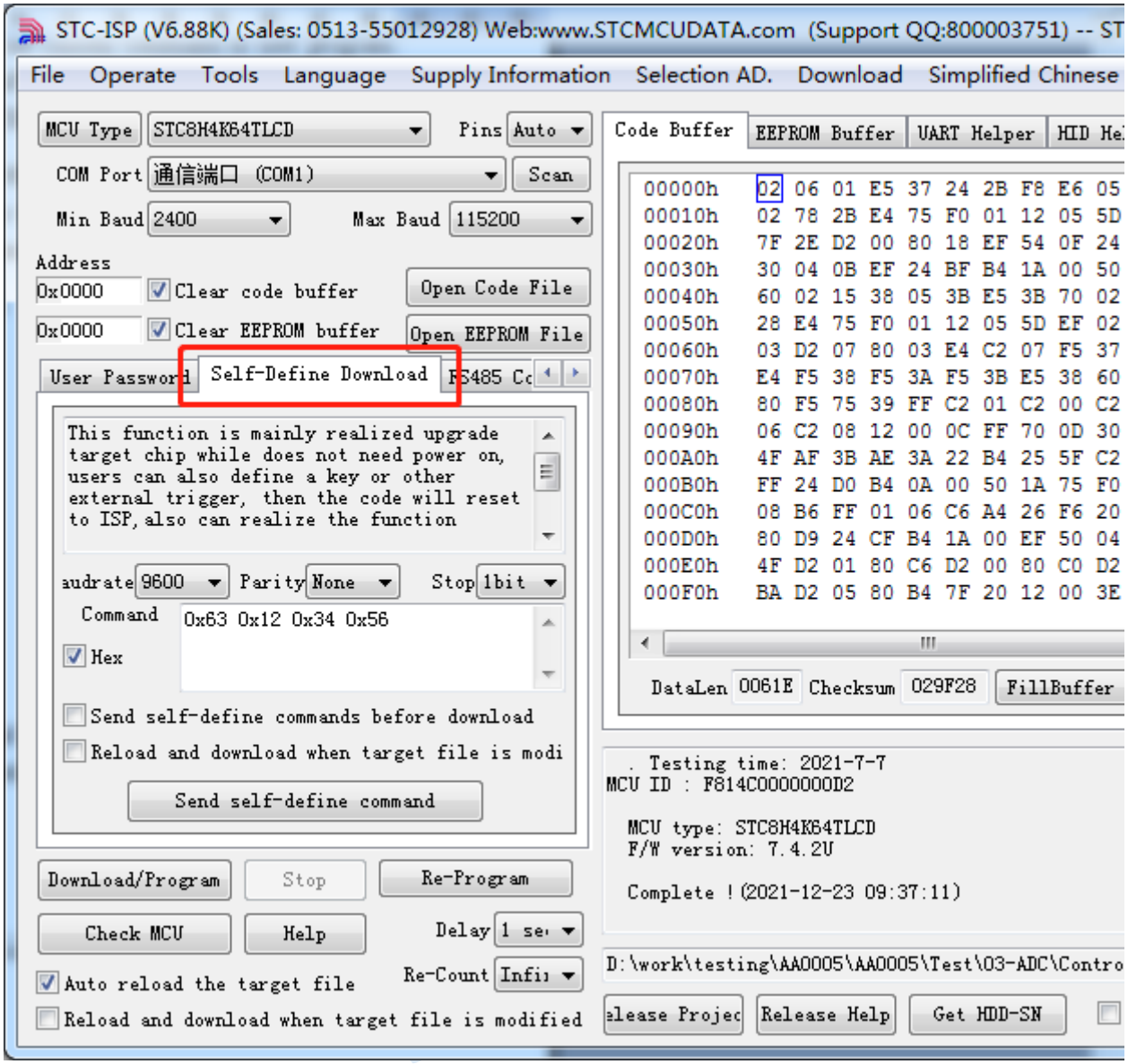
"User-defined download" and "user-defined encrypted download" are two completely different functions. Compared with the function of user-defined encrypted download, the function of user-defined download is simpler.

The specific functions is: Before the computer or offline download board starts to send the real ISP download programming handshake command, it first sends a user-defined string of commands (for this string of serial commands, user can set the baud rate, parity, and stop bits), and then immediately sends the ISP download programming handshake command.

The function of "user-defined download" is mainly used in the early development stage of the project, which can download user code without power-off (without re-power-on to the target chip). The specific implementation method is: User needs to add a piece of code to detect the custom command in user program. When the command is detected, execute the assembly code of "MOV IAP_CONTR, # 60H" or the C language code of "IAP_CONTR = 0x60;" , MCU will reset to ISP area to execute ISP code automatically.

As shown in the figure below, set the custom command sequence with a baud rate of 115200, no parity bit, and one stop bit: 0x12, 0x34, 0x56, 0xAB, 0xCD, 0xEF, 0x12. When the option "Send custom commands before each download" is checked, the user-defined download function can be implemented.

STCMCU



Click "Send user-defined download command" or click the "Download / Program" button in the lower left corner of the window, the application will send the serial data as shown below.

Appendix I Use STC's IAP series MCU to develop your own ISP program

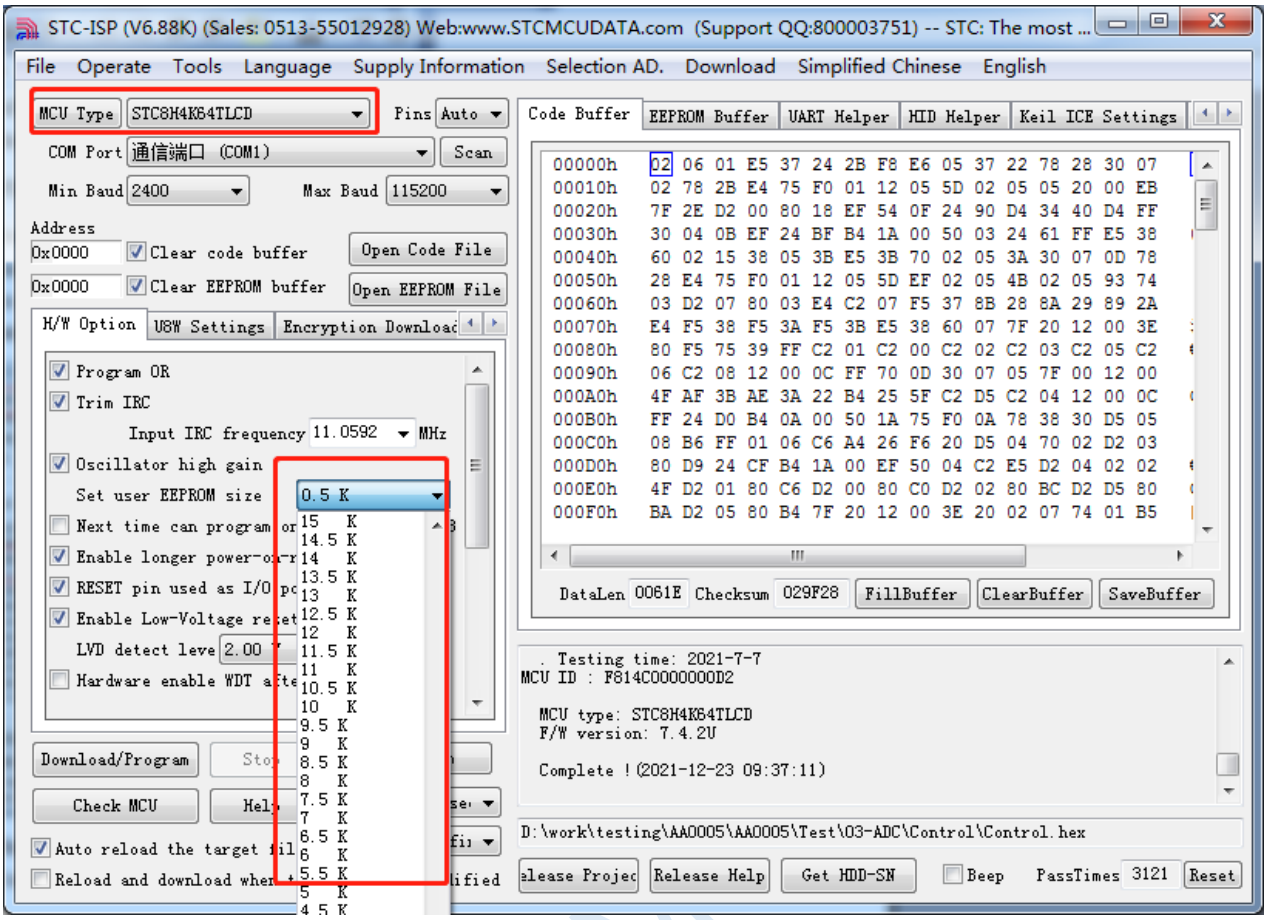
With the continuous development of IAP (In-Application-Programming) technology in the field of microcontrollers, it has brought great convenience to the application system program code upgrade. STC's serial ISP (In-System-Programming) program uses the IAP function to upgrade the user's program online, but for the sake of user code safety, neither the underlying code nor the upper-level application is open source. For this reason, STC launched with the IAP series microcontrollers, that is, users can rewrite the Flash space of the entire MCU in their own programs, so that the idea that users need to develop their own ISP programs can be realized.

All STC8G series microcontrollers that can customize the EEPROM size during ISP download are IAP series microcontrollers. At present, the STC8H series have the following types of microcontrollers as the IAP series: STC8G1K12, STC8G1K17, STC8G1K12A, STC8G1K17A, STC8G1K12-8Pin, STC8G1K17-8Pin, STC8G1K12T, STC8G1K17G2, STC8G8K64. STC8G2K64S4 is taken as an example to explain in detail the method of using STC's IAP microcontroller to develop the user's own ISP program, and gives the assembly and C source code based on the Keil environment.

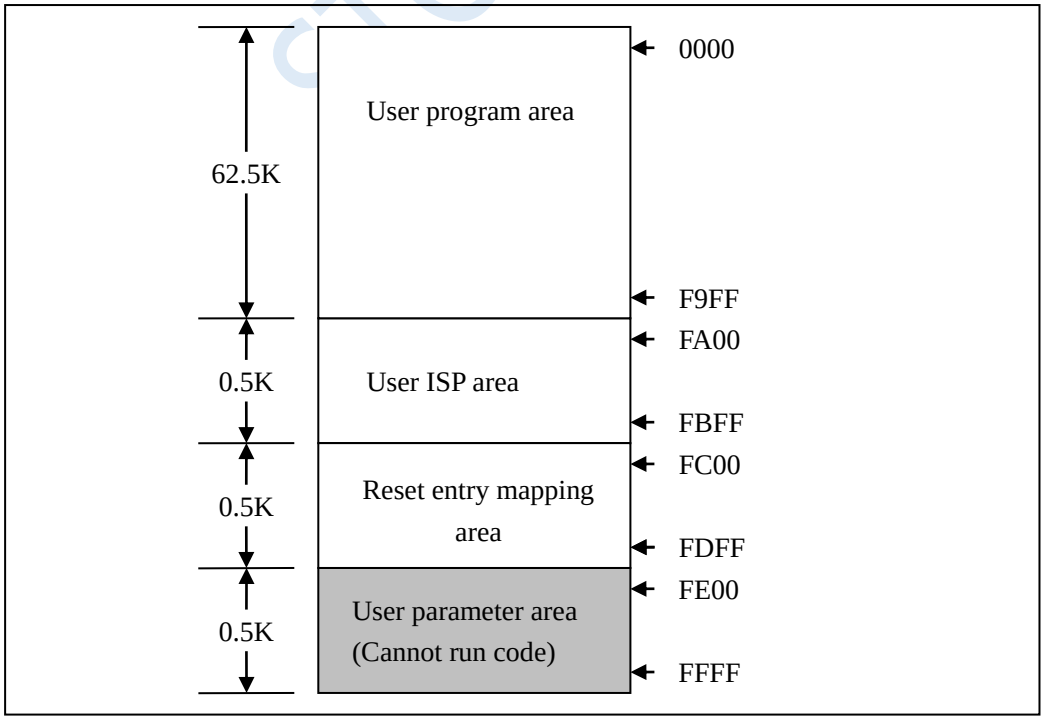
The first step: internal FLASH planning

Since the EEPROM of the IAP microcontroller of the STC8G series is set by the user during ISP download, if the user needs to implement his own ISP when downloading the user's own ISP program, the user needs to follow the figure below to set all 64K Set to EEPROM, so that the user program space and EEPROM space are completely overlapped, and users can modify and update their own program space.

STC MCU



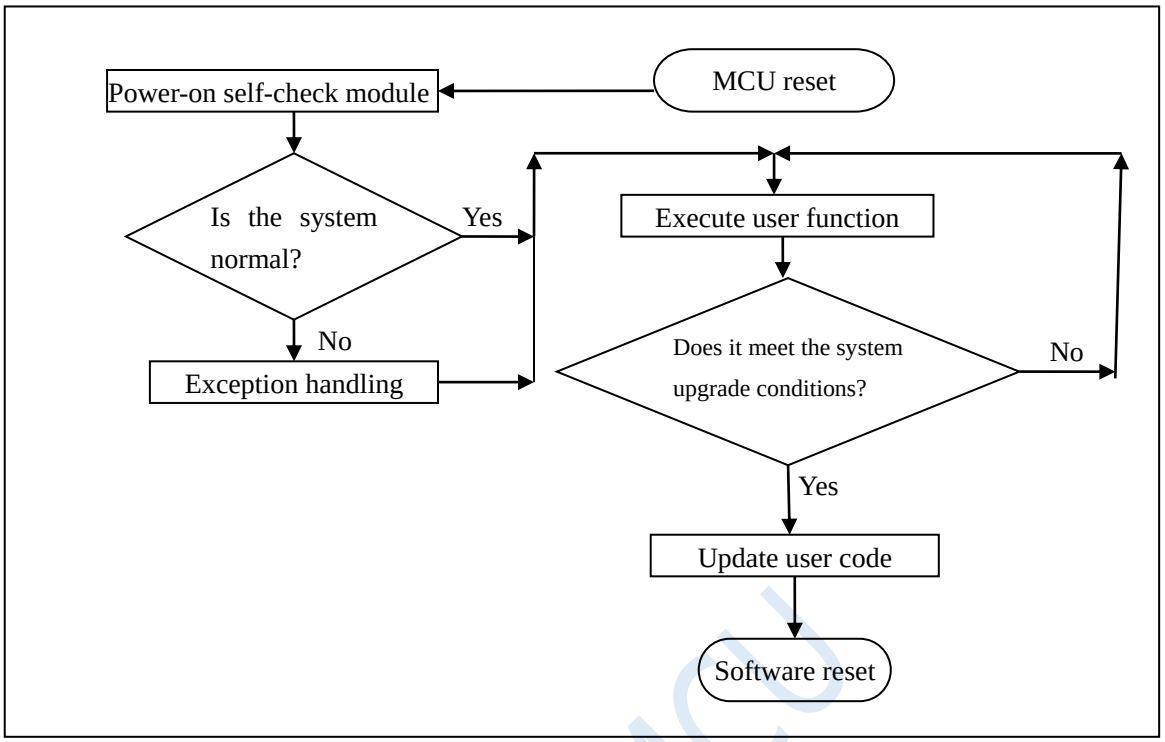
The following assumes that the user has set the entire 64K program space as EEPROM. Now the entire 64K program space is divided as follows:



In the FLASH space, the continuous 62.5K bytes of space starting from address 0000H is the user program area. When the specific download conditions are met, the user is required to jump the PC to the user ISP program area. At this time, the user program area can be erased and rewritten to achieve the purpose of updating the user

program.

The second step, the basic framework of the program



The third step, the firmware program description of the lower computer

The firmware program of the lower computer includes two parts: ISP (ISP code) and AP (user code)

ISP code (Assembly code)

```
; The test operating frequency is 11.0592MHz

UARTBAUD    EQU     0FFE8H      ; Define the serial port baud rate (65536-11059200/4/115200)

AUXR        DATA    08EH        ; Additional function control register
WDT_CONTR   DATA    0C1H        ; Watchdog control register
IAP_DATA    DATA    0C2H        ;IAP Data register
IAP_ADDRH   DATA    0C3H        ;IAP High address register
IAP_ADDRL   DATA    0C4H        ;IAP Low address register
IAP_CMD     DATA    0C5H        ;IAP Command register
IAP_TRIG    DATA    0C6H        ;IAP Command trigger register
IAP_CONTR   DATA    0C7H        ;IAP Control register
IAP_TPS     DATA    0F5H        ;IAP Waiting time control register

ISPCODE     EQU     0FA00H      ; ISP module entry address (page 1), which is also the address of the external interface
APENTRY     EQU     0FC00H      ; Application entry address (1 page)

            ORG     0000H

            LJMP    ISP_ENTRY   ; System reset entry

RESET:
            MOV     SCON,#50H   ; Set the serial port mode (8 data bits, no parity bit)
            MOV     AUXR,#40H   ; Timer 1 is 1T mode
```

```
                MOV     TMOD,#00H           ; Timer 1 works in mode 0 (16-bit reload)
                MOV     TH1,#HIGH UARTBAUD  ; Set reload value
                MOV     TL1,#LOW UARTBAUD
                SETB    TR1                 ; Start timer 1
NEXT1:
                MOV     R0,#16
NEXT2:
                JNB     RI,$                ; Waiting for serial port data
                CLR     RI
                MOV     A,SBUF
                CJNE    A,#7FH,NEXT1        ; Determine whether it is 7F
                DJNZ    R0,NEXT2
                LJMP    ISP_DOWNLOAD        ; Jump to the download interface

                ORG     ISPCODE

ISP_DOWNLOAD:
                CLR     A
                MOV     PSW,A               ; ISP module uses the 0th group of registers
                MOV     IE,A                ; Close all interrupts
                CLR     RI                  ; Clear serial port receiving flag
                SETB    TI                  ; Set serial port sending flag
                CLR     TR0
                MOV     SP,#5FH             ; Set the stack pointer

                MOV     A,#5AH              ; Return 5A 55 to the PC, indicating that the ISP erase module is ready
                LCALL   ISP_SENDUART
                MOV     A,#055H
                LCALL   ISP_SENDUART
                LCALL   ISP_RECVACK         ; Receive response data

                MOV     IAP_ADDRL,#0        ; First write "LJMP ISP_ENTRY" instruction at the start address of page 2
                MOV     IAP_ADDRH,#02H
                LCALL   ISP_ERASEIAP
                MOV     A,#02H
                LCALL   ISP_PROGRAMIAP      ; Programming user code reset vector code
                MOV     A,#HIGH  ISP_ENTRY
                LCALL   ISP_PROGRAMIAP      ; Programming user code reset vector code
                MOV     A,#LOW ISP_ENTRY
                LCALL   ISP_PROGRAMIAP      ; Programming user code reset vector code

                MOV     IAP_ADDRL,#0        ; User code address starts from 0
                MOV     IAP_ADDRH,#0
                LCALL   ISP_ERASEIAP
                MOV     A,#02H
                LCALL   ISP_PROGRAMIAP      ; Programming user code reset vector code
                MOV     A,#HIGH   ISP_ENTRY
                LCALL   ISP_PROGRAMIAP      ; Programming user code reset vector code
                MOV     A,#LOW ISP_ENTRY
                LCALL   ISP_PROGRAMIAP      ; Programming user code reset vector code

                MOV     IAP_ADDRL,#0        ; New code buffer address
                MOV     IAP_ADDRH,#02H
                MOV     R7,#124             ; Erase 62.5K bytes
ISP_ERASEAP:
                LCALL   ISP_ERASEIAP
                INC     IAP_ADDRH           ; Destination address +512
```

```
            INC     IAP_ADDRH
            DJNZ    R7,ISP_ERASEAP      ; Determine whether the erasure is complete

            MOV     IAP_ADDRL,#LOW APENTRY
            MOV     IAP_ADDRH,#HIGH APENTRY
            LCALL   ISP_ERASEIAP

            MOV     A,#5AH              ; Return 5A A5 to the PC, indicating that the ISP programming module is ready
            LCALL   ISP_SENDUART
            MOV     A,#0A5H
            LCALL   ISP_SENDUART
            LCALL   ISP_RECVACK         ; Receive response data

            LCALL   ISP_RECVUART        ; Receive length high byte
            MOV     R0,A
            LCALL   ISP_RECVUART        ; Receive length low byte
            MOV     R1,A
            CLR     C                   ; The total length-3
            MOV     A,#03H
            SUBB    A,R1
            MOV     DPL,A
            CLR     A
            SUBB    A,R0
            MOV     DPH,A               ; The complement of total length is stored in DPTR

            LCALL   ISP_RECVUART        ; Mapping user code resets the entry code to the mapping area
            LCALL   ISP_PROGRAMIAP      ;0000
            LCALL   ISP_RECVUART
            LCALL   ISP_PROGRAMIAP      ;0001
            LCALL   ISP_RECVUART
            LCALL   ISP_PROGRAMIAP      ;0002

            MOV     IAP_ADDRL,#03H      ; User code start address
            MOV     IAP_ADDRH,#00H
ISP_PROGRAMNEXT:
            LCALL   ISP_RECVUART        ; Receive code data
            LCALL   ISP_PROGRAMIAP      ; Program to user code area
            INC     DPTR
            MOV     A,DPL
            ORL     A,DPH
            JNZ     ISP_PROGRAMNEXT     ; Length detection

ISP_SOFTRESET:
            MOV     IAP_CONTR,#20H      ; Software reset system
            SJMP    $


ISP_ENTRY:
            MOV     WDT_CONTR,#17H      ; Clear watchdog
            MOV     IAP_CONTR,#80H      ; Enable IAP function
            MOV     IAP_TPS,#11         ; Set IAP waiting time parameters
            MOV     IAP_ADDRL,#LOW ISP_DOWNLOAD
            MOV     IAP_ADDRH,#HIGH ISP_DOWNLOAD
            MOV     IAP_DATA,#00H       ; Test Data  1
            MOV     IAP_CMD,#1          ; Read command
            MOV     IAP_TRIG,#5AH       ; Trigger ISP command
            MOV     IAP_TRIG,#0A5H
```

```
            MOV     A,IAP_DATA
            CJNE    A,#0E4H,ISP_ENTRY       ; If you cannot read the data, you need to wait for the
voltage to stabilize
            INC     IAP_ADDRL               ; Test address FC01H
            MOV     IAP_DATA,#45H           ; Test Data 2
            MOV     IAP_CMD,#1              ; Read command
            MOV     IAP_TRIG,#5AH           ; Trigger ISP command
            MOV     IAP_TRIG,#0A5H
            MOV     A,IAP_DATA
            CJNE    A,#0F5H,ISP_ENTRY       ; If you cannot read the data, you need to wait for the
voltage to stabilize

            MOV     SCON,#50H               ; Set the serial port mode (8 data bits, no parity bit)
            MOV     AUXR,#40H               ; Timer 1 is 1T mode
            MOV     TMOD,#00H               ; Timer 1 works in mode 0 (16-bit reload)
            MOV     TH1,#HIGH UARTBAUD      ; Set reload value
            MOV     TL1,#LOW UARTBAUD
            SETB    TR1                     ; Start timer 1
            SETB    TR0

            LCALL   ISP_RECVUART            ; Check if there is serial port data
            JC      GOTOAP
            MOV     R0,#16
ISP_CHECKNEXT:
            LCALL   ISP_RECVUART            ; Receive synchronized data
            JC      GOTOAP
            CJNE    A,#7FH,GOTOAP           ; Determine whether it is 7F
            DJNZ    R0,ISP_CHECKNEXT
            MOV     A,#5AH                  ; Return 5A 69 to the PC, indicating that the ISP module is
ready
            LCALL   ISP_SENDUART
            MOV     A,#69H
            LCALL   ISP_SENDUART
            LCALL   ISP_RECVACK             ; Receive response data
            LJMP    ISP_DOWNLOAD            ; Jump to the download interface
GOTOAP:
            CLR     A                       ; Restore SFR to reset value
            MOV     TCON,A
            MOV     TMOD,A
            MOV     TL0,A
            MOV     TH0,A
            MOV     TL1,A
            MOV     TH1,A
            MOV     SCON,A
            MOV     AUXR,A
            LJMP    APENTRY                 ; Run the user program normally

ISP_RECVACK:
            LCALL   ISP_RECVUART
            JC      GOTOAP
            XRL     A,#7FH
            JZ      ISP_RECVACK             ; Skip synchronized data
            CJNE    A,#25H,GOTOAP           ; Response data 1 detection
            LCALL   ISP_RECVUART
            JC      GOTOAP
            CJNE    A,#69H,GOTOAP           ; Response data 2 detection
            RET

ISP_RECVUART:
```

STC MCU

```
        CLR     A
        MOV     TL0,A           ; Initialize the timeout timer
        MOV     TH0,A
        CLR     TF0
        MOV     WDT_CONTR,#17H  ; Clear watchdog
ISP_RECVWAIT:
        JBC     TF0,ISP_RECVTIMEOUT ; Timeout detection
        JNB     RI,ISP_RECVWAIT ; Wait for the reception to complete
        MOV     A,SBUF          ; Read serial port data
        CLR     RI              ; Clear flag
        CLR     C               ; Receive serial data correctly
        RET
ISP_RECVTIMEOUT:
        SETB    C               ; Timeout exit
        RET

ISP_SENDUART:
        MOV     WDT_CONTR,#17H  ; Clear watchdog
        JNB     TI,ISP_SENDUART ; Waiting for the completion of the previous data transmission
        CLR     TI              ; Clear flag
        MOV     SBUF,A          ; Send current data
        RET

ISP_ERASEIAP:
        MOV     WDT_CONTR,#17H  ; Clear watchdog
        MOV     IAP_CMD,#3      ; Erase command
        MOV     IAP_TRIG,#5AH   ; Trigger ISP command
        MOV     IAP_TRIG,#0A5H
        NOP
        NOP
        NOP
        NOP
        RET

ISP_PROGRAMIAP:
        MOV     WDT_CONTR,#17H  ; Clear watchdog
        MOV     IAP_CMD,#2      ; Programming command
        MOV     IAP_DATA,A      ; Send current data to IAP data register
        MOV     IAP_TRIG,#5AH   ; Trigger ISP command
        MOV     IAP_TRIG,#0A5H
        NOP
        NOP
        NOP
        NOP
        MOV     A,IAP_ADDRL     ;IAP address +1
        ADD     A,#01H
        MOV     IAP_ADDRL,A
        MOV     A,IAP_ADDRH
        ADDC    A,#00H
        MOV     IAP_ADDRH,A
        RET


        ORG     APENTRY
        LJMP    RESET


        END
```

STC MCU

The ISP code includes the following external interface modules:

ISP_DOWNLOAD: program download entry address, absolute address FA00H

ISP_ENTRY: Power-on system self-check program (automatically called by the system)

For the user program, the user only needs to jump the PC value to ISPPROGRAM (that is, the absolute address of FA00H) when the download conditions are met, and then the code can be updated.

User code (C language code)

```
// The test operating frequency is 11.0592MHz

#include "reg51.h"

#define   FOSC              11059200L           // System clock frequency
#define   BAUD              (65536 - FOSC/4/115200)  // Define the serial port baud rate
#define   ISPPROGRAM        0xfa00              // ISP download program entry address

sfr       AUXR      =       0x8e;               // Baud rate generator control register
sfr       P1M0      =       0x92;
sfr       P1M1      =       0x91;

void (*IspProgram)() = ISPPROGRAM;              // Define pointer function
char cnt7f;                                     //Isp_Check Variables used internally

void uart() interrupt 4                         // Serial port interrupt service routine
{
    if (TI) TI = 0;                             // Send complete interrupt
    if (RI)                                     // Receive complete interrupt
    {
        if (SBUF == 0x7f)
        {
            cnt7f++;
            if (cnt7f >= 16)
            {
                IspProgram();                   // Call the download module (****Important Statement****)
            }
        }
        else
        {
            cnt7f = 0;
        }
        RI = 0;                                 // Clear receiving completion flag
    }
}

void main()
{
    SCON = 0x50;                                // Define the serial port mode as 8bit variable, no parity bit
    AUXR = 0x40;
    TH1  = BAUD >> 8;
    TL1  = BAUD;
    TR1  = 1;
    ES = 1;                                     // Enable serial port interrupt
    EA = 1;                                     // Enable  the global interrupt

    P1M0 = 0;
    P1M1 = 0;
```

```
    while (1)
    {
        P1++;
    }
}
```

User code (assembly code)

```
; The test operating frequency is 11.0592MHz

UARTBAUD    EQU     0FFE8H          ; Define the serial port baud rate  (65536-11059200/4/115200)
ISPPROGRAM  EQU     0FA00H          ; ISP download program entry address

AUXR        DATA    08EH            ; Additional function control register

CNT7F       DATA    60H             ; Receive 7F counter

            ORG     0000H
            LJMP    START           ; System reset entry

            ORG     0023H
            LJMP    UART_ISR        ; Serial port interrupt entry

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            JNB     TI,CHECKRI      ; Detect transmission interruption
            CLR     TI              ; Clear flag
CHECKRI:
            JNB     RI,UARTISR_EXIT ; Detect receive interrupt
            CLR     RI              ; Clear flag
            MOV     A,SBUF
            CJNE    A,#7FH,ISNOT7F
            INC     CNT7F
            MOV     A,CNT7F
            CJNE    A,#16,UARTISR_EXIT
            LJMP    ISPPROGRAM      ; Call the download module (****Important Statement****)
ISNOT7F:
            MOV     CNT7F,#0
UARTISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

START:
            MOV     R0,#7FH         ;Clear RAM
            CLR     A
            MOV     @R0,A
            DJNZ    R0,$-1
            MOV     SP,#7FH         ; Initialize SP

            MOV     SCON,#50H       ; Set the serial port mode (8-bit variable, no parity bit)
            MOV     AUXR,#15H       ; BRT works in 1T mode, and start BRT
            MOV     TMOD,#00H       ; Timer 1 works in mode 0 (16-bit reload)
            MOV     TH1,#HIGH UARTBAUD  ; Set reload value
            MOV     TL1,#LOW UARTBAUD
            SETB    TR1             ; Start timer 1
```

```
        SETB    ES              ; Enable serial port interrupt
        SETB    EA              ; Enable  the global interrupt

MAIN:
        INC     P0
        SJMP    MAIN

        END
```

User code can be written in C or assembly language, but one thing needs to be noted for assembly code is that the instruction at the reset entry address of 0000H must be a long jump statement (similar to LJMP START). In the user code, the serial port needs to be set up, and when the download conditions are met, the PC value is jumped to ISPPROGRAM (that is, the absolute address of FA00H) to achieve code update. For assembly code, we can use the "LJMP 0FA00H" instruction to call, as shown below.

In the C code, you must define a function pointer variable, and assign this variable to 0xFA00, and then call it, as shown in the figure below.

The fourth step, the host computer application program description

Open the host computer interface, as shown below.

Select the serial port number and set the same serial port baud rate as the lower computer.
Open the source data file to be downloaded, either in Bin or Intel hex format.
Click the "Download Data" button to start downloading data.

The sixth step, how to use the firmware of the lower computer

The lower computer has two target files "IAPISP.hex" and "AP.hex". For a new microcontroller, the "IAPISP.hex" must be written into the chip using the ISP download tool of STC Technology for the first time as shown below. After the update, there is no need to write the "IAPISP.hex" file. The "AP.hex" in the attachment is just a template of the user program. When the download conditions are met, the user only needs to jump the PC value to the address of FA00H, and then achieve code update.

Appendix J The method of resetting the user program to the system area for ISP download (without power down)

When the project is in the development stage, it is necessary to download the user code to the target chip repeatedly for code verification, and the normal ISP download of the STC MCU requires the target chip to be re-powered, which will make the development stage of the project more cumbersome. For this reason, STC MCU has added a special function register IAP_CONTR. When the user writes 0x60 to the register, the software can be reset to the system area, and then ISP download can be performed without power failure.

But how do users judge whether ISP download is in progress? When to write 0x60 to register IAP_CONTR to trigger a soft reset? Regarding these two issues, four methods of judgment are introduced below:

Use P3.0 port to detect the serial port start signal

The serial port ISP of STC microcontroller uses P3.0 and P3.1 fixedly. When the ISP download software starts to download, it will send a handshake command to the P3.0 port of the microcontroller. If the user's P3.0 and P3.1 are only used for ISP download, you can use the P3.0 port to detect the start signal of the serial port to judge the ISP download.

C laguage code

```
// The test operating frequency is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     IAP_CONTR   =   0xc7;
sfr     P3M0        =   0xb2;
sfr     P3M1        =   0xb1;

sbit    P30         =   P3^0;

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;
    P30 = 1;

    while (1)
    {
        if (!P30) IAP_CONTR = 0x60;          // The low level of P3.0 is the serial port start signal
                                             // Reset to the system area by software
```

```
        ...                                     //User code
    }
}
```

Use the falling edge interrupt of P3.0/INT4 port to detect the serial port start signal.

Method B is similar to method A, except that method A uses query mode, and method B uses interrupt mode. Because the P3.0 port of the STC microcontroller is the interrupt port of INT4.

C laguage code

```
// The test operating frequency is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     IAP_CONTR   =   0xc7;
sfr     INTCLKO     =   0x8f;
sfr     P3M0        =   0xb2;
sfr     P3M1        =   0xb1;

void Int4Isr() interrupt 16                     //INT4 Interrupt service routine
{
    IAP_CONTR = 0x60;                           // Serial start signal triggers INT4 interrupt
                                                // Reset to the system area by software
}

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    INTCLKO |= 0x40;                            // Enable INT4 interrupt
    EA = 1;

    while (1)
    {
        ...                                     // User code
    }
}
```

Use the serial port of P3.0/RxD port to receive and check the 7F sent by the ISP download software.

Method A and Method B are very simple, but easy to be interfered. If there is any interference signal on the P3.0 port, it will trigger a software reset, so method C is to verify the serial port data.

When the STC ISP download software performs ISP download, it will use the lowest baud rate (usually 2400) + even parity 9+1 stop bit to continuously send the handshake command 7F firstly, so the user can set the serial port in the program to 9 bit data bit + 2400 baud rate, and then continue to detect 7F, for example, continuous detection of 8 7Fs means that it is determined that ISP download is required, and then the software reset is triggered.

C laguage code

```
// The test operating frequency is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC          11059200UL
#define   BR2400        (65536 - FOSC / 4 / 2400)

sfr       IAP_CONTR   =   0xc7;
sfr       AUXR        =   0x8e;
sfr       P3M0        =   0xb2;
sfr       P3M1        =   0xb1;

char cnt7f;

void UartIsr() interrupt 4                  // Serial port interrupt service routine
{
    if (TI)
    {
        TI = 0;
    }

    if (RI)
    {
        RI = 0;
        if ((SBUF == 0x7f) && (RB8 == 1))   // Handshake command 7F sent by ISP download software
                                            //The even parity bit of 7F is 1
        {
            if (++cnt7f == 8)               // After 8 consecutive 7Fs are detected
                IAP_CONTR = 0x60;           // Reset to system area
        }
        else
        {
            cnt7f = 0;
        }
    }
}

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    SCON = 0xd0;                            // Set the serial port to 9 bits data
    TMOD = 0x00;
    AUXR = 0x40;
    TH1 = BR2400 >> 8;                      // Set the serial port baud rate to 2400
    TL1 = BR2400;
    TR1 = 1;
    ES = 1;
    EA = 1;

    cnt7f = 0;

    while (1)
    {
        ...                                 //User code
    }
}
```

STC MCU

Use P3.0/RxD serial port to receive and detect user download commands sent by ISP download software.

If the user code needs to use the serial port for communication, the above 3 methods may not be applicable. At this time, you can use the interface provided by STC's ISP download software to customize a set of dedicated user download commands (you can specify the baud rate, check bit and stop bit). If this function is enabled, the ISP download software will use the user-specified baud rate, check bit and stop bit to send the user download command before ISP download, and then send the handshake command. The user only needs to monitor the serial port command sequence in his own code. When the correct user download command is detected, the software is reset to the system area to realize the ISP function without power failure.

The following assumes that the user download command is the string "STCISP$", the serial port is set to 115200 baudrate, no parity bit and 1 stop bit. The settings in the ISP download software are as follows:

The user sample code is as follows:

C laguage code

```
// The test operating frequency is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define    FOSC          11059200UL
#define    BR115200      (65536 - FOSC / 4 / 115200)

sfr        IAP_CONTR  =  0xc7;
sfr        AUXR       =  0x8e;
sfr        P3M0       =  0xb2;
sfr        P3M1       =  0xb1;

char stage;

void UartIsr() interrupt 4                    // Serial port interrupt service routine
{
    char dat;

    if (TI)
    {
        TI = 0;
    }

    if (RI)
    {
        RI = 0;

        dat = SBUF;
        switch (stage)
        {
        case 0:
        default:
L_Check1st:
            if (dat == 'S') stage = 1;
            else stage = 0;
            break;
        case 1:
```

```
                if (dat == 'T') stage = 2;
                else goto L_Check1st;
                break;
            case 2:
                if (dat == 'C') stage = 3;
                else goto L_Check1st;
                break;
            case 3:
                if (dat == 'I') stage = 4;
                else goto L_Check1st;
                break;
            case 4:
                if (dat == 'S') stage = 5;
                else goto L_Check1st;
                break;
            case 5:
                if (dat == 'P') stage = 6;
                else goto L_Check1st;
                break;
            case 6:
                if (dat == '$')                         // When the correct user download command is detected
                    IAP_CONTR = 0x60;                   // Reset to system area
                else goto L_Check1st;
            break;
            }
        }
}

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    SCON = 0x50;                                        // Set the user serial port mode to 8 bits data
    TMOD = 0x00;
    AUXR = 0x40;
    TH1 = BR2400 >> 8;                                  // Set the serial port baud rate to 115200
    TL1 = BR2400;
    TR1 = 1;
    ES = 1;
    EA = 1;

    stage = 0;

    while (1)
    {
        ...                                             //User code
    }
}
```

STC MCU

Appendix K Example Routine of ISP download for STC8G series MCUs using third-party MCU

C language code

```
/*Note: When using this code to download the STC8G series of microcontrollers, you must execute the Download code before powering on the target chip, otherwise the target chip will not download correctly.*/

#include "reg51.h"

typedef    bit              BOOL;
typedef    unsigned char    BYTE;
typedef    unsigned short   WORD;

//Macro and constant definition
#define    FALSE            0
#define    TRUE             1
#define    LOBYTE(w)        ((BYTE)(WORD)(w))
#define    HIBYTE(w)        ((BYTE)((WORD)(w) >> 8))

#define    MINBAUD          2400L
#define    MAXBAUD          115200L

#define    FOSC             11059200L          //Main chip working frequency
#define    BR(n)            (65536 - FOSC/4/(n))   // Calculation formula of serial port baud rate of main chip
#define    T1MS             (65536 - FOSC/1000)    // 1ms timing initial value of main chip

#define    FUSER            24000000L          //STC8G Series target chip operating frequency
#define    RL(n)            (65536 - FUSER/4/(n))  //STC8G Serial target chip baud rate calculation formula

sfr        AUXR = 0x8e;
sfr        P3M1 = 0xB1;
sfr        P3M0 = 0xB2;

// Variable definitions
BOOL f1ms;                          //1ms flag
BOOL UartBusy;                      // Serial transmit busy flag
BOOL UartReceived;                  // Serial data receiving completion flag
BYTE UartRecvStep;                  // Serial data receiving control
BYTE TimeOut;                       // Serial communication timeout counter
BYTE xdata TxBuffer[256];           // Serial data transmission buffer
BYTE xdata RxBuffer[256];           // Serial data receiving buffer
char code DEMO[256];                // Demo code data

// Functions declarations
void Initial(void);
void DelayXms(WORD x);
BYTE UartSend(BYTE dat);
void CommInit(void);
void CommSend(BYTE size);
```

```
BOOL Download(BYTE *pdat, long size);

// Main function entry
void main(void)
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    Initial();
    if (Download(DEMO, 256))
    {
        // download successfully
        P3 = 0xff;
        DelayXms(500);
        P3 = 0x00;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0x00;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0x00;
        DelayXms(500);
        P3 = 0xff;
    }
    else
    {
        // download failed
        P3 = 0xff;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
    }

    while (1);
}

//1ms Timer interrupt service routine
void tm0(void) interrupt 1
{
    static BYTE Counter100;

    f1ms = TRUE;
    if (Counter100-- == 0)
    {
        Counter100 = 100;
        if (TimeOut) TimeOut--;
    }
}
```

STC MCU

```
// Serial port interrupt service routine
void uart(void) interrupt 4
{
    static WORD RecvSum;
    static BYTE RecvIndex;
    static BYTE RecvCount;
    BYTE dat;

    if (TI)
    {
        TI = 0;
        UartBusy = FALSE;
    }

    if (RI)
    {
        RI = 0;
        dat  = SBUF;
        switch (UartRecvStep)
        {
        case 1:
            if (dat != 0xb9) goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 2:
            if (dat != 0x68) goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 3:
            if (dat != 0x00) goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 4:
            RecvSum = 0x68 + dat;
            RecvCount = dat - 6;
            RecvIndex = 0;
            UartRecvStep++;
            break;
        case 5:
            RecvSum += dat;
            RxBuffer[RecvIndex++] = dat;
            if (RecvIndex == RecvCount)     UartRecvStep++;
            break;
        case 6:
            if (dat != HIBYTE(RecvSum))     goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 7:
            if (dat != LOBYTE(RecvSum))     goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 8:
            if (dat != 0x16) goto L_CheckFirst;
            UartReceived = TRUE;
            UartRecvStep++;
            break;
L_CheckFirst:
        case 0:
```

```
            default:
                CommInit();
                UartRecvStep = (dat == 0x46       ? 1 : 0);
                break;
            }
        }
}

// system initialization
void Initial(void)
{
        UartBusy = FALSE;

        SCON = 0xd0;                        // Serial data format must be 8-bit data + 1-bit even check
        AUXR = 0xc0;
        TMOD = 0x00;
        TH0 = HIBYTE(T1MS);
        TL0 = LOBYTE(T1MS);
        TR0 = 1;
        TH1 = HIBYTE(BR(MINBAUD));
        TL1 = LOBYTE(BR(MINBAUD));
        TR1 = 1;
        ET0 = 1;
        ES = 1;
        EA = 1;
}

//Xms Delay program
void DelayXms(WORD x)
{
        do
        {
            f1ms = FALSE;
            while (!f1ms);
        } while (x--);
}

// Serial data sending program
BYTE UartSend(BYTE dat)
{
        while (UartBusy);

        UartBusy = TRUE;
        ACC = dat;
        TB8 = P;
        SBUF = ACC;

        return dat;
}

// Serial communication initialization
void CommInit(void)
{
        UartRecvStep = 0;
        TimeOut = 20;
        UartReceived = FALSE;
}

// Send serial communication packets
```

STC MCU

```
void CommSend(BYTE size)
{
    WORD sum;
    BYTE i;

    UartSend(0x46);
    UartSend(0xb9);
    UartSend(0x6a);
    UartSend(0x00);
    sum = size + 6 + 0x6a;
    UartSend(size + 6);
    for (i=0; i<size; i++)
    {
        sum  += UartSend(TxBuffer[i]);
    }
    UartSend(HIBYTE(sum));
    UartSend(LOBYTE(sum));
    UartSend(0x16);
    while (UartBusy);

    CommInit();
}

//对 STC15H Series of chips for ISP download
BOOL Download(BYTE *pdat, long size)
{
    BYTE arg;
    BYTE offset;
    BYTE cnt;
    WORD addr;

    // Shake hands
    CommInit();
    while (1)
    {
        if (UartRecvStep == 0)
        {
            UartSend(0x7f);
            DelayXms(10);
        }
        if (UartReceived)
        {
            arg = RxBuffer[4];
            if (RxBuffer[0] == 0x50) break;
            return FALSE;
        }
    }

    // Set parameters (set the parameters such as the highest baud rate used by the slave chip and erase wait time)
    TxBuffer[0] = 0x01;
    TxBuffer[1] = arg;
    TxBuffer[2] = 0x40;
    TxBuffer[3] = HIBYTE(RL(MAXBAUD));
    TxBuffer[4] = LOBYTE(RL(MAXBAUD));
    TxBuffer[5] = 0x00;
    TxBuffer[6] = 0x00;
    TxBuffer[7] = 0x97;
    CommSend(8);
    while (1)
```

STC MCU

```
{
    if (TimeOut == 0) return FALSE;
    if (UartReceived)
    {
        if (RxBuffer[0] == 0x01) break;
        return FALSE;
    }
}

//prepare
TH1 = HIBYTE(BR(MAXBAUD));
TL1 = LOBYTE(BR(MAXBAUD));
DelayXms(10);
TxBuffer[0] = 0x05;
TxBuffer[1] = 0x00;
TxBuffer[2] = 0x00;
TxBuffer[3] = 0x5a;
TxBuffer[4] = 0xa5;
CommSend(5);
while (1)
{
    if (TimeOut == 0) return FALSE;
    if (UartReceived)
    {
        if (RxBuffer[0] == 0x05) break;
        return FALSE;
    }
}

// Erase
DelayXms(10);
TxBuffer[0] = 0x03;
TxBuffer[1] = 0x00;
TxBuffer[2] = 0x00;
TxBuffer[3] = 0x5a;
TxBuffer[4] = 0xa5;
CommSend(5);
TimeOut = 100;
while (1)
{
    if (TimeOut == 0) return FALSE;
    if (UartReceived)
    {
        if (RxBuffer[0] == 0x03) break;
        return FALSE;
    }
}

// Write user code
DelayXms(10);
addr = 0;
TxBuffer[0] = 0x22;
TxBuffer[3] = 0x5a;
TxBuffer[4] = 0xa5;
offset = 5;
while (addr < size)
{
    TxBuffer[1] = HIBYTE(addr);
    TxBuffer[2] = LOBYTE(addr);
```

STC MCU

```
        cnt    = 0;
        while (addr < size)
        {
            TxBuffer[cnt+offset] = pdat[addr];
            addr++;
            cnt++;
            if (cnt >= 128) break;
        }
        CommSend(cnt + offset);
        while (1)
        {
            if (TimeOut == 0) return FALSE;
            if (UartReceived)
            {
                if ((RxBuffer[0] == 0x02) && (RxBuffer[1] == 'T')) break;
                return FALSE;
            }
        }
        TxBuffer[0] = 0x02;
    }

    //// Write hardware options
    //// If you do not need to modify the hardware options, this step can be skipped directly.At this time, all the hardware options
    //// remain unchanged, the frequency of the MCU is the last adjusted frequency
    //// If you write the hardware option, the MCU's internal IRC frequency will be fixed to 24MHz,
   //// and other options will be restored to the factory settings.
    ////Suggestion: Set the hardware options of the slave chip when you use STC-ISP download software the first time.

    //// Do not write hardware options when downloading programs from the master chip to the slave chip.
    //DelayXms(10);
    //for (cnt=0; cnt<128; cnt++)
    //{
    //    TxBuffer[cnt] = 0xff;
    //}
    //TxBuffer[0] = 0x04;
    //TxBuffer[1] = 0x00;
    //TxBuffer[2] = 0x00;
    //TxBuffer[3] = 0x5a;
    //TxBuffer[4] = 0xa5;
    //TxBuffer[33] = arg;
    //TxBuffer[34] = 0x00;
    //TxBuffer[35] = 0x01;
    //TxBuffer[41] = 0xbf;
    //TxBuffer[42] = 0xbd;                  //P5.4 is I/O port
    ////TxBuffer[42] = 0xad;                //P5.4 is reset pin
    //TxBuffer[43] = 0xf7;
    //TxBuffer[44] = 0xff;
    //CommSend(45);
    //while (1)
    //{
    //    if (TimeOut == 0) return FALSE;
    //    if (UartReceived)
    //    {
    //        if ((RxBuffer[0] == 0x04) && (RxBuffer[1] == 'T')) break;
    //        return FALSE;
    //    }
    //}
```

STCMCU

```
    // Download completed
    return TRUE;
}

char code DEMO[256] =
{
    0x80,0x00,0x75,0xB2,0xFF,0x75,0xB1,0x00,0x05,0xB0,0x11,0x0E,0x80,0xFA,0xD8,0xFE,
    0xD9,0xFC,0x22,
};
```

Note: If user needs to set different working frequencies, please refer to the example codes in chapters 7.3.7 and 7.3.8.

STC MCU

Appendix L Use a third-party application program to call the STC release project program to download to MCU using the ISP

The release project program generated by STC's ISP download software is an executable EXE format file. The user can directly double-click the released project program to run it for ISP download, or call the release project program in a third-party application for ISP download. Two methods of calling are described below.

Simple call

In the third-party application, it is only a simple process of creating and publishing the project program. All other download operations are carried out in the publishing project program. The third-party application only needs to wait for the completion of the publishing project program and clean up the site.

VC code

```
BOOL IspProcess()
{
	// Define related variables
	STARTUPINFO si;
	PROCESS_INFORMATION pi;
	CString path;

	// The full path of the release project program
	path = _T("D:\\Work\\Upgrade.exe");

	// Variables initialization
	memset(&si, 0, sizeof(STARTUPINFO));
	memset(&pi, 0, sizeof(PROCESS_INFORMATION));

	// Set startup variables
	si.cb = sizeof(STARTUPINFO);
	GetStartupInfo(&si);
	si.wShowWindow = SW_SHOWNORMAL;
	si.dwFlags = STARTF_USESHOWWINDOW;

	// Create a release project program process
	if (CreateProcess(NULL, (LPTSTR)(LPCTSTR)path, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
	{
		// Waiting for the completion of the release project program operation
		// Since the main process will be blocked here, it is recommended to create a new work process and wait in the work process
		WaitForSingleObject(pi.hProcess,INFINITE);

		// Clean up
		CloseHandle(pi.hThread);
		CloseHandle(pi.hProcess);
```

```
            return TRUE;
        }
        else
        {
            AfxMessageBox(_T("Failed to create process !"));

            return FALSE;
        }
}
```

Advanced call

To create and publish the project program in the third-party application, and perform all ISP download operations in the third-party application, including selecting the serial port, starting ISP programming, stopping ISP programming, and closing the publishing project program, it is not necessary to publish the project program to interact with the interface.

VC 代码

```
// Define the data structure of the callback function parameters
struct CALLBACK_PARAM
{
    DWORD dwProcessId;                  // Main process ID
    HWND hMainWnd;                      // Main window handle
};

// Callback function for enumerating windows, used to get the handle of the main window
BOOL CALLBACK EnumWindowCallBack(HWND hWnd, LPARAM lParam)
{
    CALLBACK_PARAM *pcp = (CALLBACK_PARAM *)lParam;
    DWORD id;

    GetWindowThreadProcessId(hWnd, &id);
    if ((pcp->dwProcessId == id) && (GetParent(hWnd) == NULL))
    {
        pcp->hMainWnd = hWnd;

        return FALSE;
    }

    return TRUE;
}

BOOL IspProcess()
{
    // Define related variables
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    CALLBACK_PARAM cp;
    CString path;

    // Release the IDs of some controls in the project program
    const UINT ID_PROGRAM     = 1046;
    const UINT ID_STOP        = 1044;
    const UINT ID_COMPORT     = 1009;
    const UINT ID_PROGRESS    = 1044;
```

```
    // The full path of the release project program
    path = _T("D:\\Work\\Upgrade.exe");

    // Variables initialization
    memset(&si, 0, sizeof(STARTUPINFO));
    memset(&pi, 0, sizeof(PROCESS_INFORMATION));
    memset(&cp, 0, sizeof(CALLBACK_PARAM));

    // Set startup variables
    si.cb = sizeof(STARTUPINFO);
    GetStartupInfo(&si);
    si.wShowWindow = SW_SHOWNORMAL;                    // If set to SW_HIDE here, the interface of the release project program will not be displayed
                                                       // All ISP operations can be performed in the background
    si.dwFlags = STARTF_USESHOWWINDOW;

    // Create a release project program process
    if (CreateProcess(NULL, (LPTSTR)(LPCTSTR)path, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
    {
        // Waiting for the completion of the initialization of the publishing project program process
        WaitForInputIdle(pi.hProcess, 5000);

        // Get the handle of the main window of the published project program
        cp.dwProcessId = pi.dwProcessId;
        cp.hMainWnd = NULL;
        EnumWindows(EnumWindowCallBack, (LPARAM)&cp);

        if (cp.hMainWnd != NULL)
        {
            HWND hProgram;
            HWND hStop;
            HWND hPort;

            // Get the handle of part of the control in the main window of the publishing project
            hProgram = ::GetDlgItem(cp.hMainWnd, ID_PROGRAM);
            hStop = ::GetDlgItem(cp.hMainWnd, ID_STOP);
            hPort = ::GetDlgItem(cp.hMainWnd, ID_COMPORT);

            // Set the serial port number in the release project program, the third parameter is 0: COM1, 1: COM2, 2: COM3, ...
            ::SendMessage(hPort, CB_SETCURSEL, 0, 0);

            // Trigger the programming button to start ISP programming
            ::SendMessage(hProgram, BM_CLICK, 0, 0);

            // Wait for the programming to complete,
            // Since the main process will be blocked here, it is recommended to create a new work process and wait in the work process
            while (!::IsWindowEnabled(hProgram));

            //编程完成后关闭发布项目程序
            ::SendMessage(cp.hMainWnd, WM_CLOSE, 0, 0);
        }

        // Wait for the process to end
        WaitForSingleObject(pi.hProcess,INFINITE);

        // Clean up
```

```
            CloseHandle(pi.hThread);
            CloseHandle(pi.hProcess);

            return TRUE;
        }
        else
        {
            AfxMessageBox(_T("创建进程失败！"));

            return FALSE;
        }
}
```

Appendix M Method for Creating Multi-file Projects in Keil

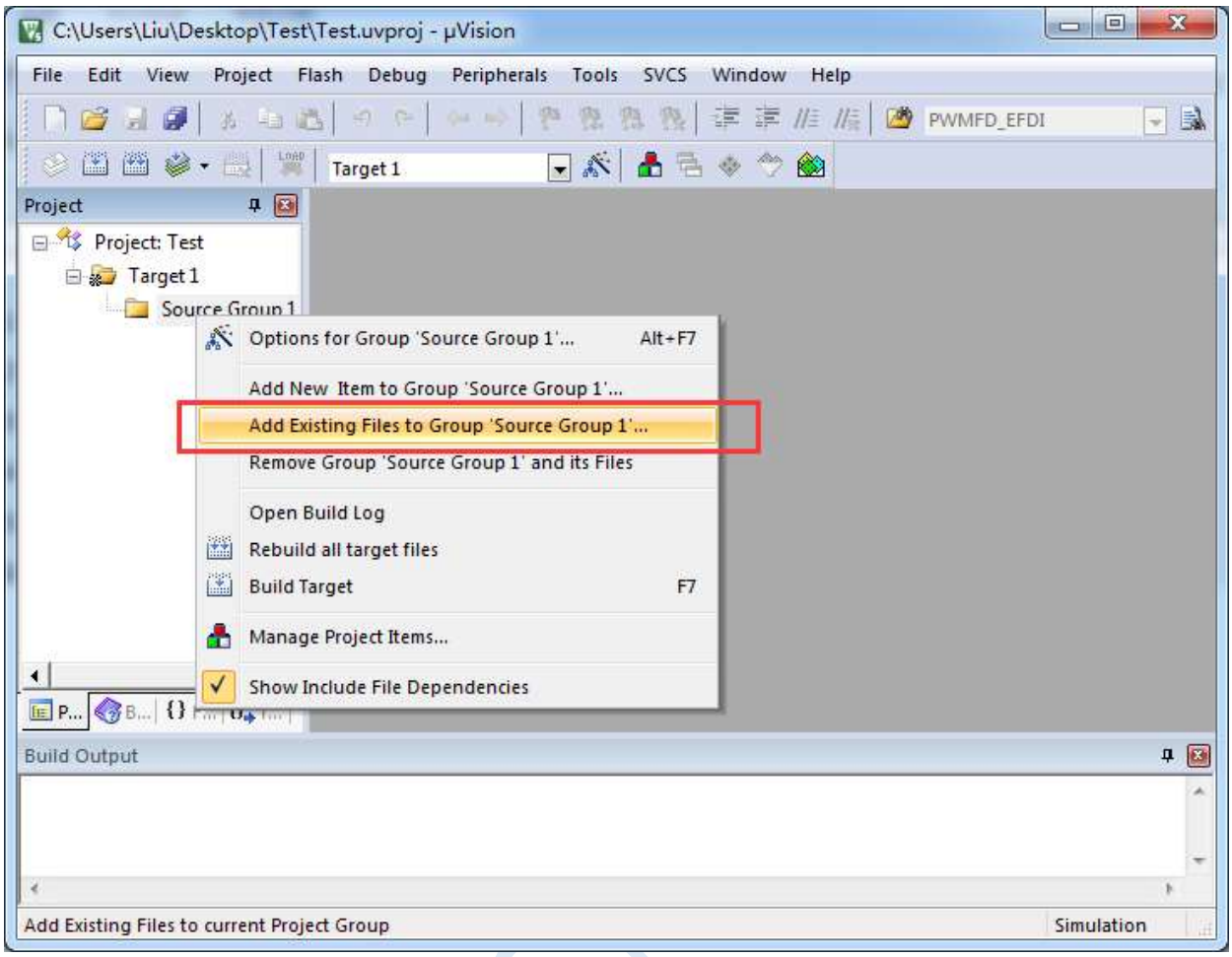
In Keil, relatively small projects generally have only one source file, but for some slightly more complex projects, multiple source files are often required. Here's how to set up a multi-file project:

1. Open Keil firstly and select "New uVision Project ..." from the "Project" menu to complete the creation of an empty project.

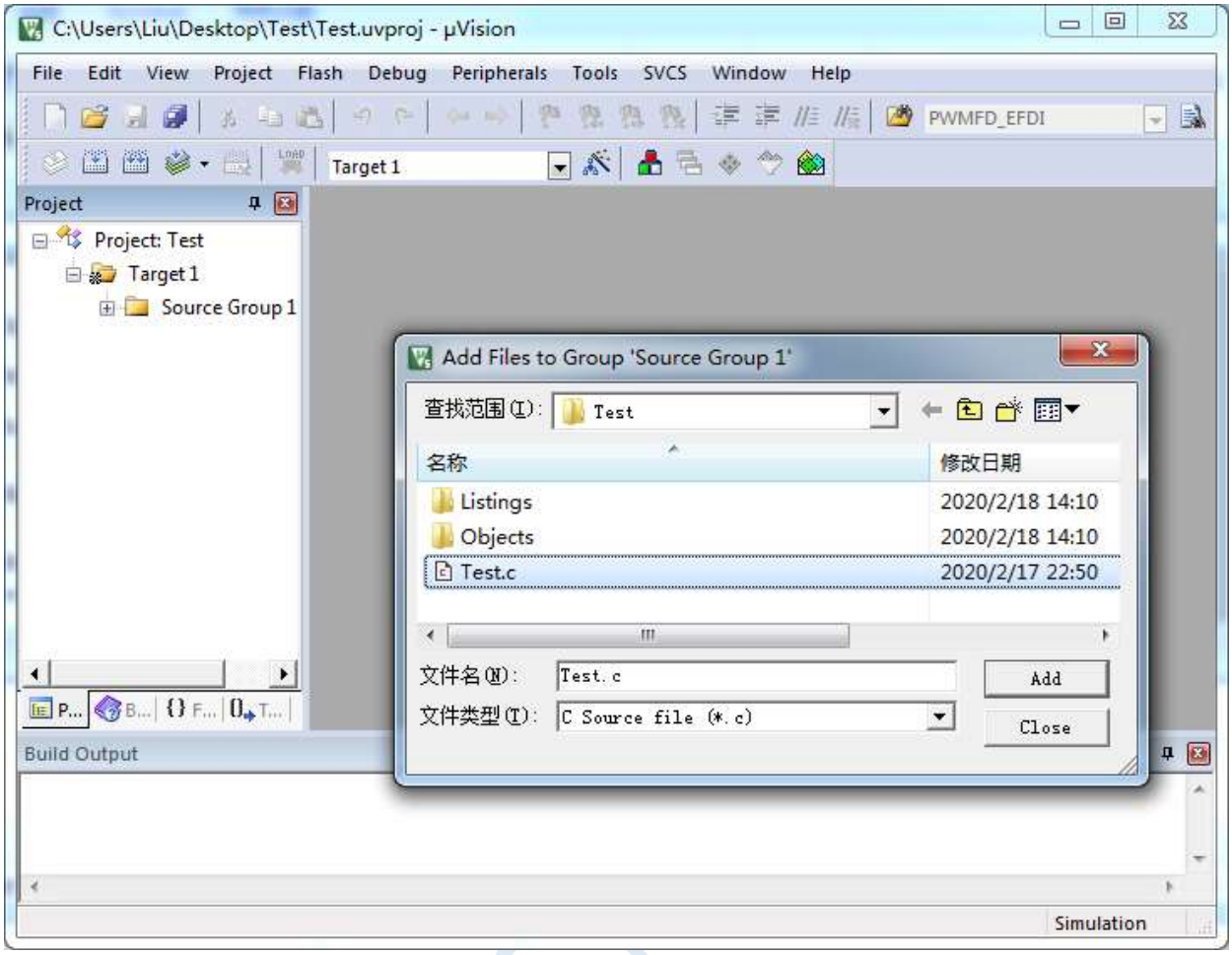
STC MCU



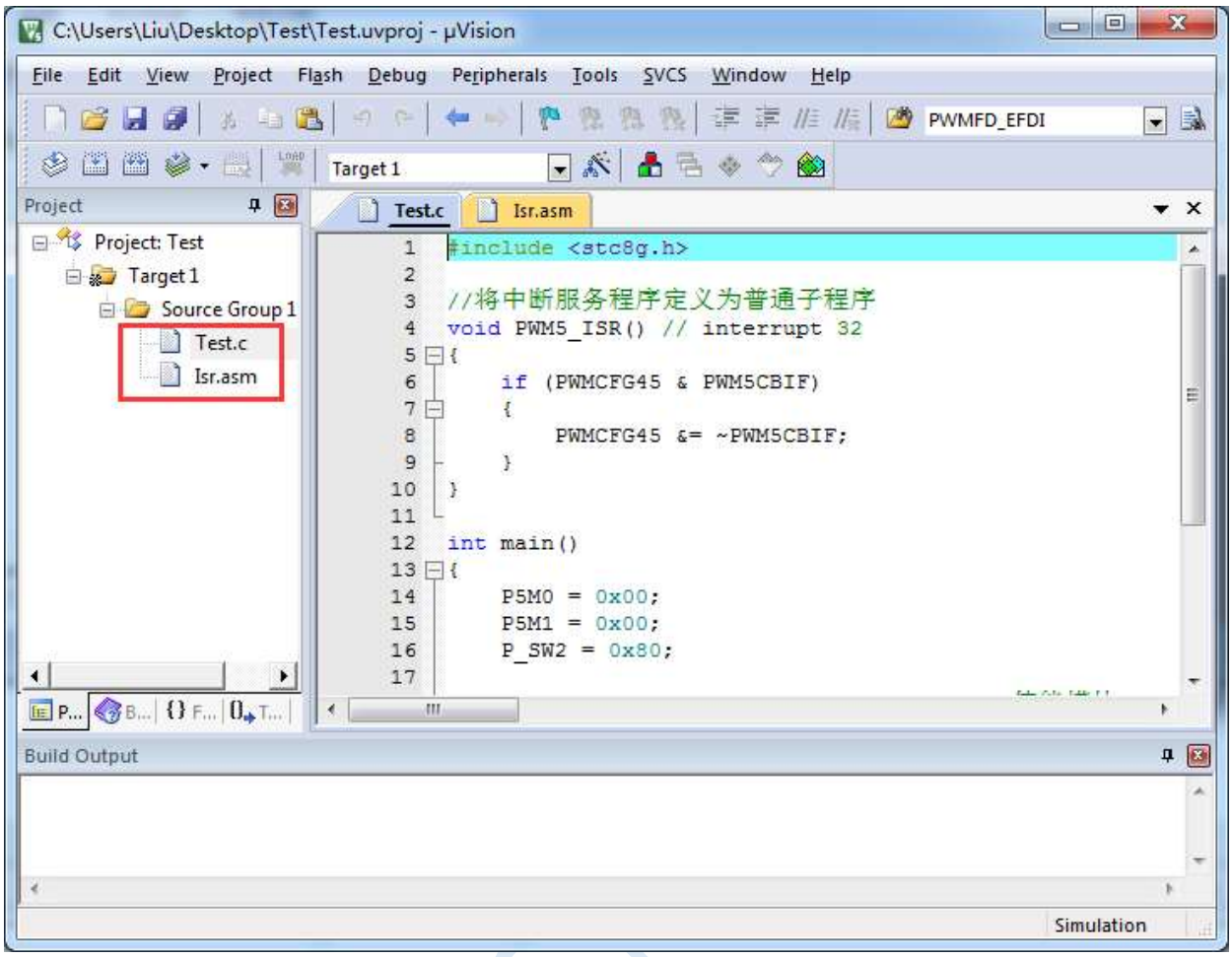
2. In the project tree of the empty project, right-click "Source Group 1" and select "Add Existing Files to Group" Source Group 1 "..." from the right-click menu.



3. In the file dialog that pops up, add the source file multiple times.

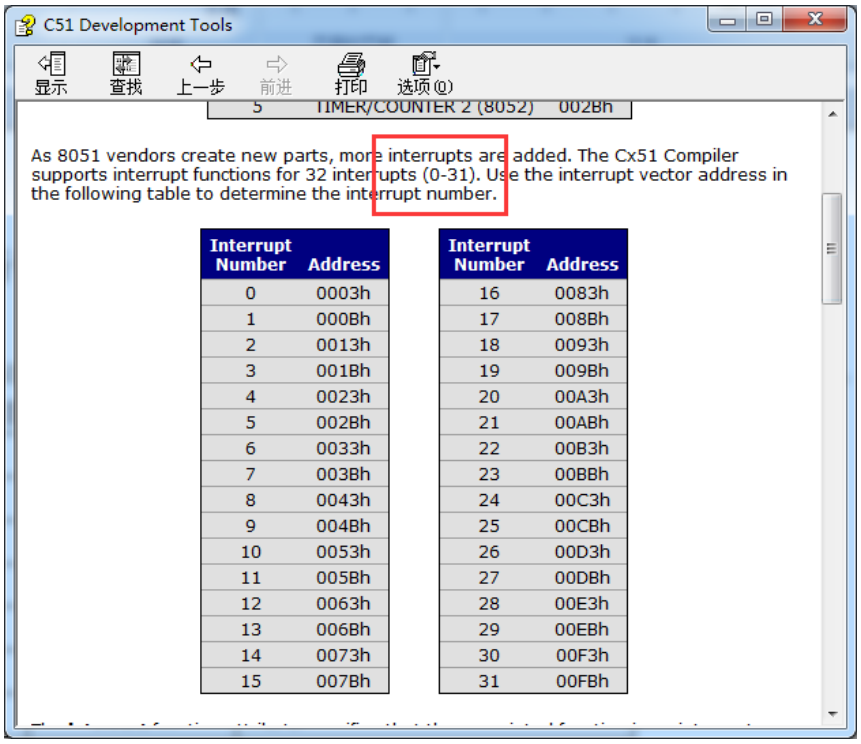


Complete the creation of the multi-file project as shown in the figure below.


C:\Users\Liu\Desktop\Test\Test.uvproj - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
PWMFD_EFDI
Target 1
Project
Project: Test
Target 1
Source Group 1
Test.c
Isr.asm
Test.c
Isr.asm
1 #include <stc8g.h>
2
3 //将中断服务程序定义为普通子程序
4 void PWM5_ISR() // interrupt 32
5 {
6 if (PWMCFG45 & PWM5CBIF)
7 {
8 PWMCFG45 &= ~PWM5CBIF;
9 }
10 }
11
12 int main()
13 {
14 P5M0 = 0x00;
15 P5M1 = 0x00;
16 P_SW2 = 0x80;
17
Build Output
Simulation

Appendix N Handling of Compilation Error in Keil with Interrupt Numbers Greater Than 31

In Keil's C51 compilation environment, only 0 ~ 31 of the interrupt number are supported, that is, the interrupt vector must be less than 0100H.



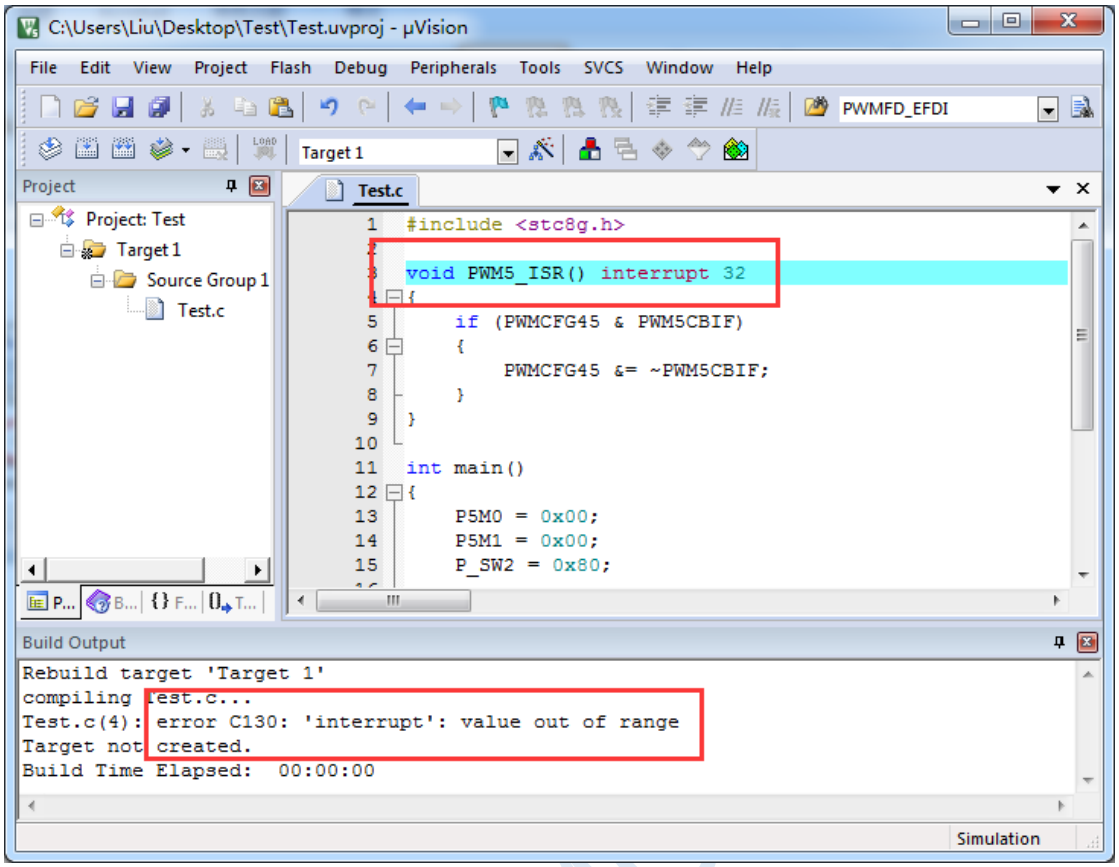
Interrupt Number	Address	Interrupt Number	Address
0	0003h	16	0083h
1	000Bh	17	008Bh
2	0013h	18	0093h
3	001Bh	19	009Bh
4	0023h	20	00A3h
5	002Bh	21	00ABh
6	0033h	22	00B3h
7	003Bh	23	00BBh
8	0043h	24	00C3h
9	004Bh	25	00CBh
10	0053h	26	00D3h
11	005Bh	27	00DBh
12	0063h	28	00E3h
13	006Bh	29	00EBh
14	0073h	30	00F3h
15	007Bh	31	00FBh

The following table is a list of interrupts for all current STC series:

Interrupt number	Interrupt vector	Interrupt type
0	0003 H	INT0
1	000B H	Timer 0
2	0013 H	INT1
3	001B H	Timer 1
4	0023 H	UART 1
5	002B H	ADC
6	0033 H	LVD
7	003B H	PCA
8	0043 H	UART 2
9	004B H	SPI
10	0053 H	INT2
11	005B H	INT3
12	0063 H	Timer 2
13	**006B H**	

14	0073 H	Simulation A5
15	007B H	Simulation PIN
16	0083 H	INT4
17	008B H	UART 3
18	0093 H	UART 4
19	009B H	Timer 3
20	00A3 H	Timer 4
21	00AB H	Comparator
22	00B3 H	Waveform generator 0
23	00BB H	Waveform generator fault 0
24	00C3 H	I2C
25	00CB H	USB
26	00D3 H	PWM1
27	00DB H	PWM2
28	00E3 H	Waveform generator 1
29	00EB H	Waveform generator 2
30	00F3 H	Waveform generator 3
31	00FB H	Waveform generator 4
32	**0103 H**	**Waveform generator 5**
33	**010B H**	**Waveform generator fault 2**
34	**0113 H**	**Waveform generator fault 4**
35	**011B H**	**Touch Key**
36	**0123 H**	**RTC**
37	**012B H**	**P0 interrupt**
38	**0133 H**	**P1 interrupt**
39	**013B H**	**P2 interrupt**
40	**0143 H**	**P3 interrupt**
41	**014B H**	**P4 interrupt**
42	**0153 H**	**P5 interrupt**
43	**015B H**	**P6 interrupt**
44	**0163 H**	**P7 interrupt**
45	**016B H**	**P8 interrupt**
46	**0173 H**	**P9 interrupt**

It is not difficult to find that starting from the interrupt of the waveform generator 5, there will be errors when all subsequent interrupt service routines be compiled in keil, as shown in the following figure:



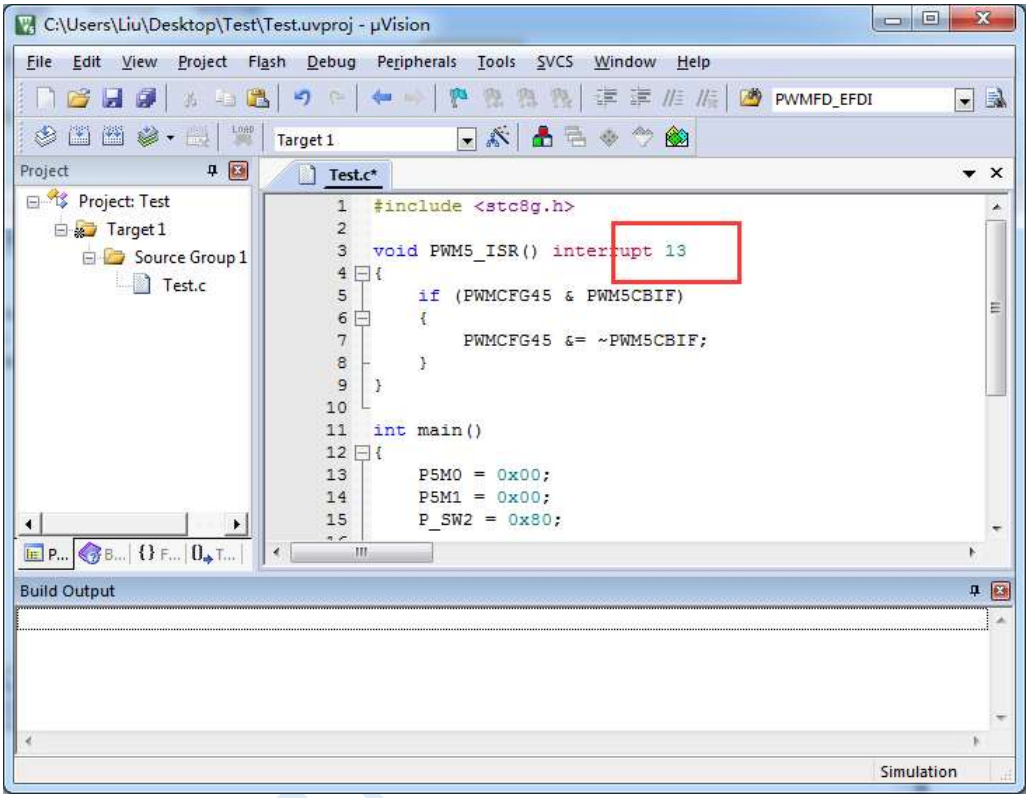
There are three ways to deal with this kind of error: (All of them need the help of assembly code, the first method is recommended)

Method 1: Borrow Interrupt Vector 13

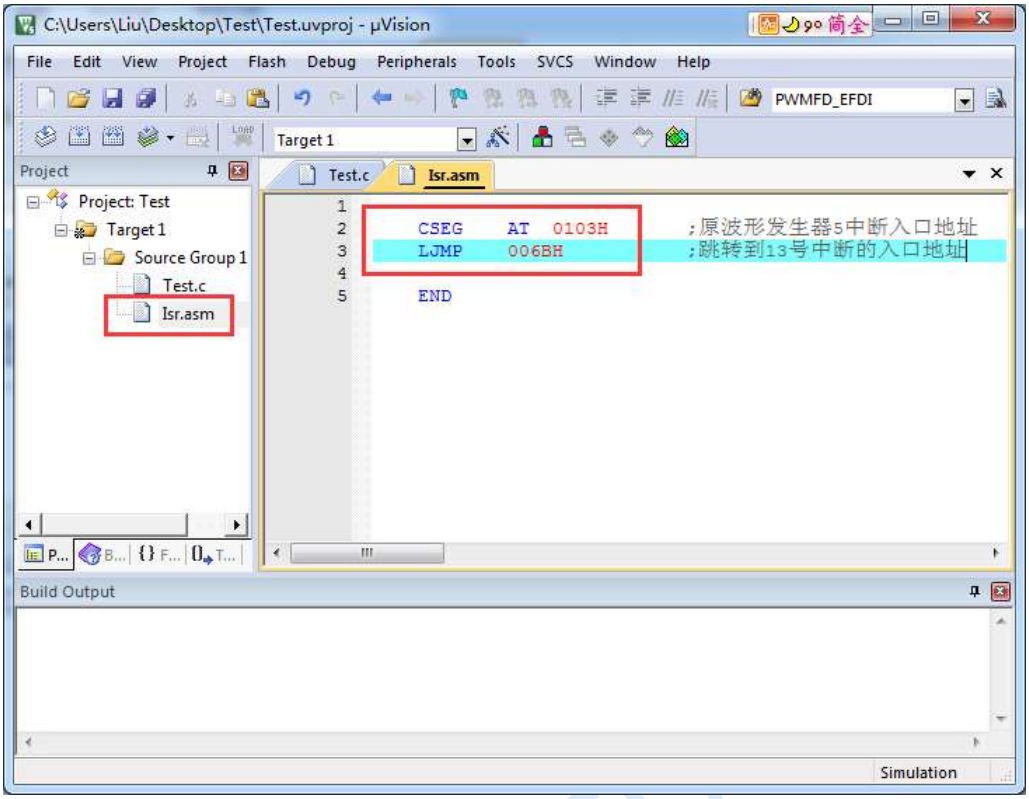
Among interrupts 0 ~ 31, the 13th is a reserved interrupt number, we can borrow this interrupt number.

The steps are as follows:

1. Change the interrupt number the compilation reported error to "13", as shown below:

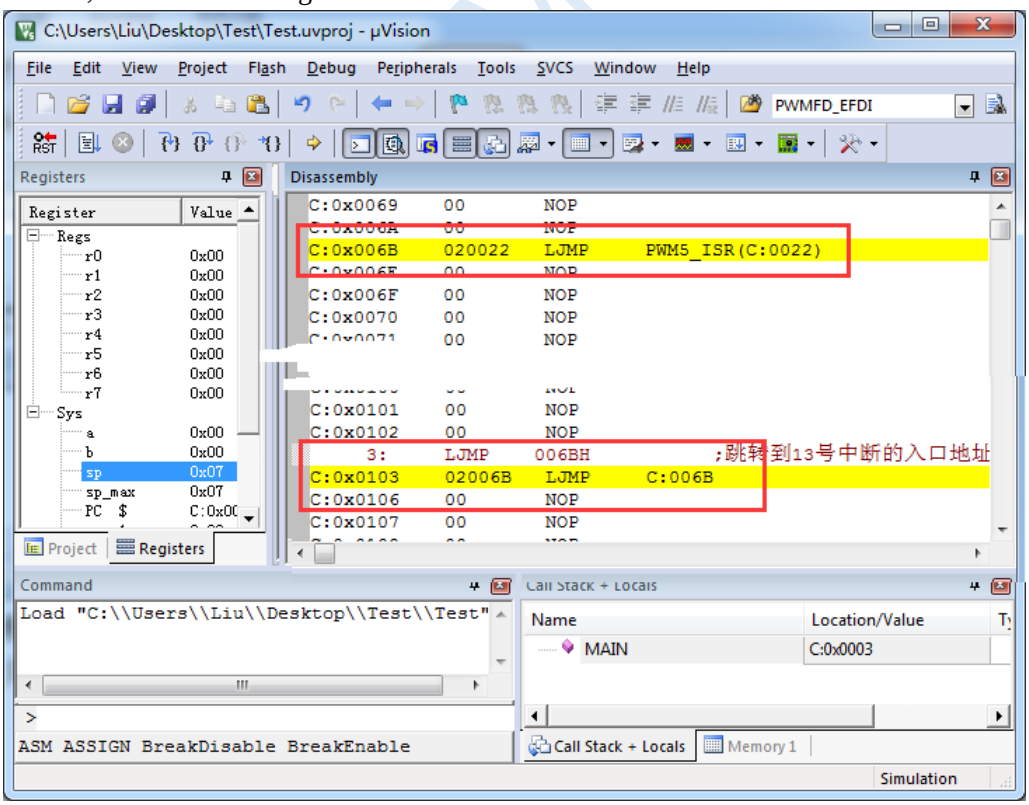


2. Create a new assembly language file, such as "isr.asm", add it to the project, and add "LJMP 006BH" at the address "0103H", as shown below:

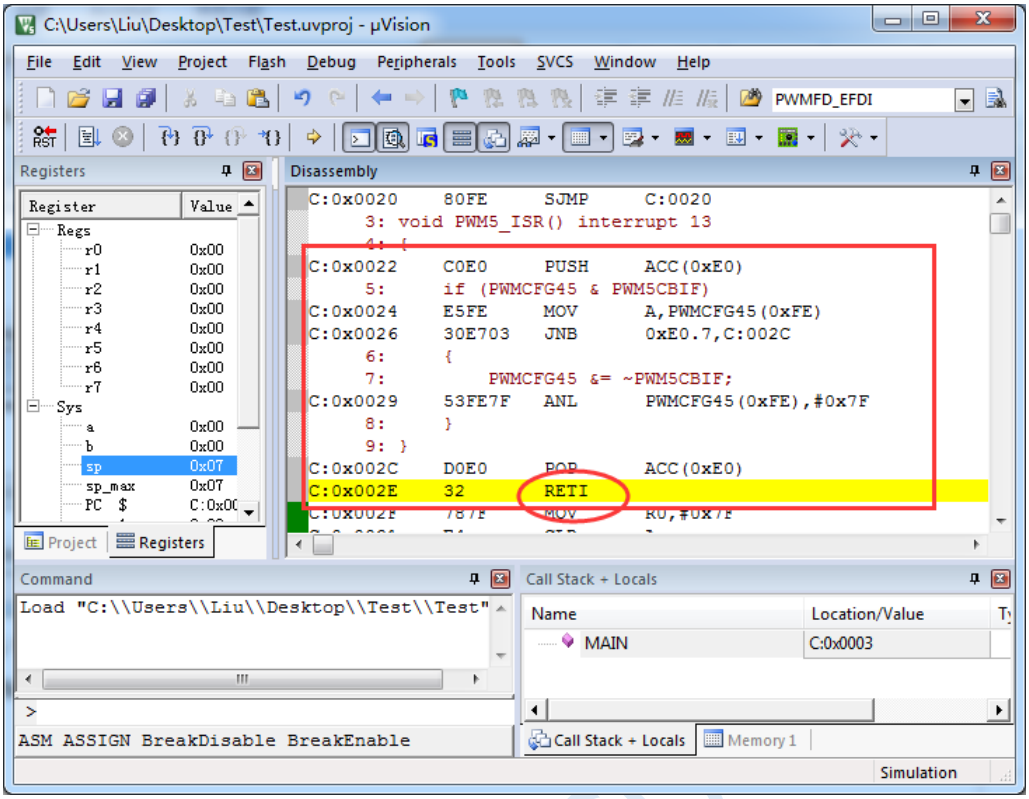


3. Compile successfully.

Now, after being compiled by Keil's C51 compiler, there is an "LJMP PWM5_ISR" at 006BH and an "LJMP 006BH" at 0103H, as shown in the figure below:



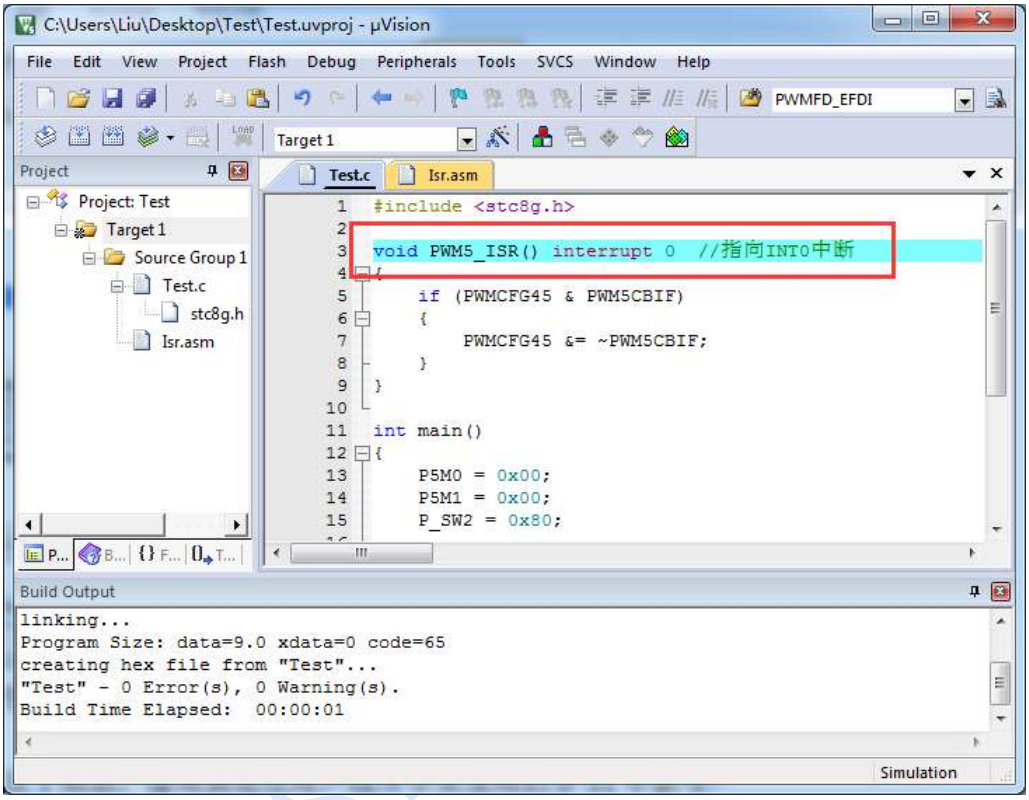
When the PWM5 interrupt occurs, the hardware will jump to the 0103H address automatically to execute "LJMP 006BH", and then execute "LJMP PWM5_ISR" at 006BH to jump to the real interrupt service routine, as shown in the figure below:

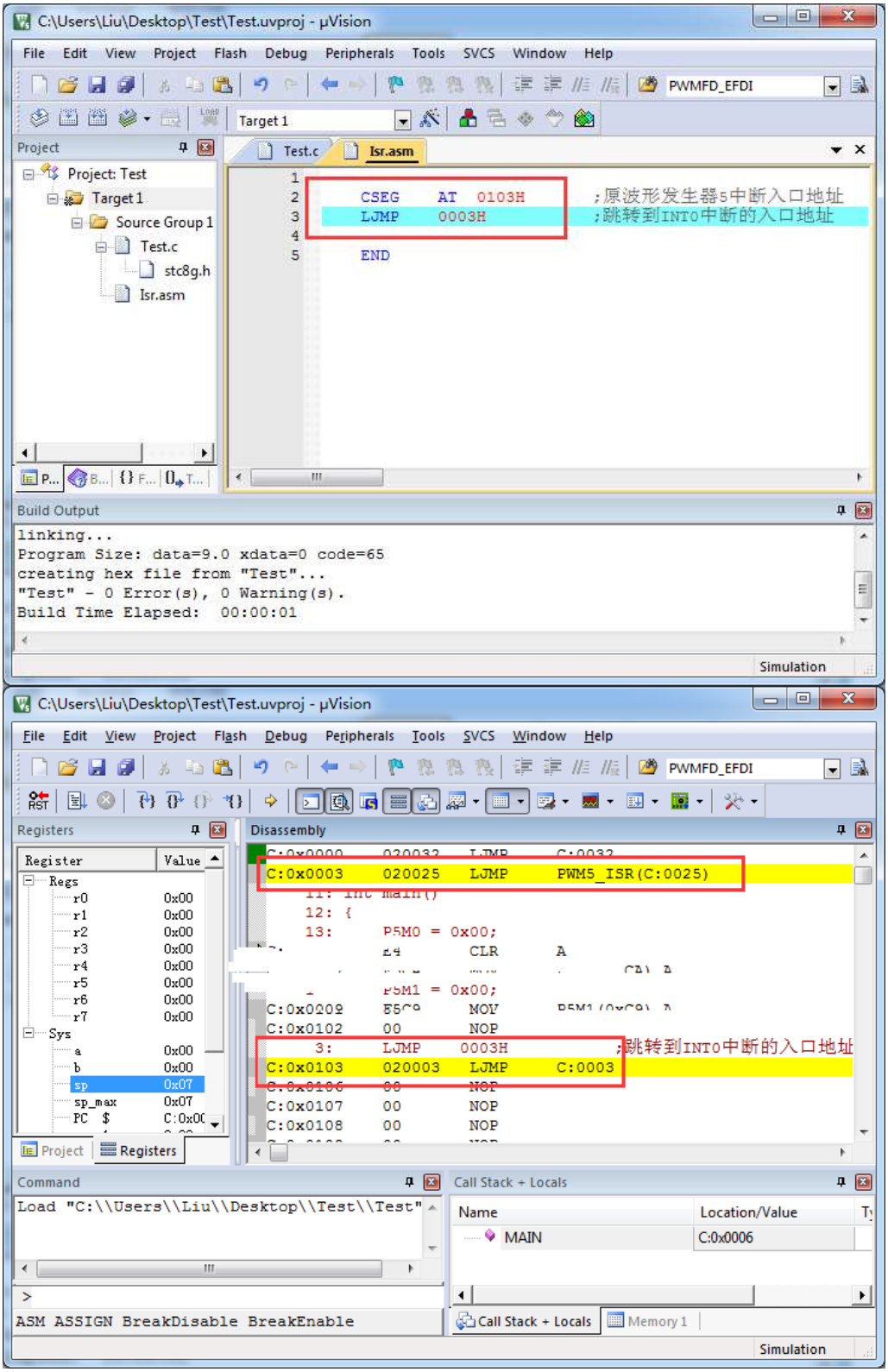


After the execution of the interrupt service routine is completed, it returns through the RETI instruction. The entire interrupt response process just executed an additional LJMP statement.

Method 2: Similar to method 1, borrow unused interrupt numbers from 0 to 31 in user program.

For example, in the user's code, if the INT0 interrupt is not used, the above code can be modified similarly to method 1:



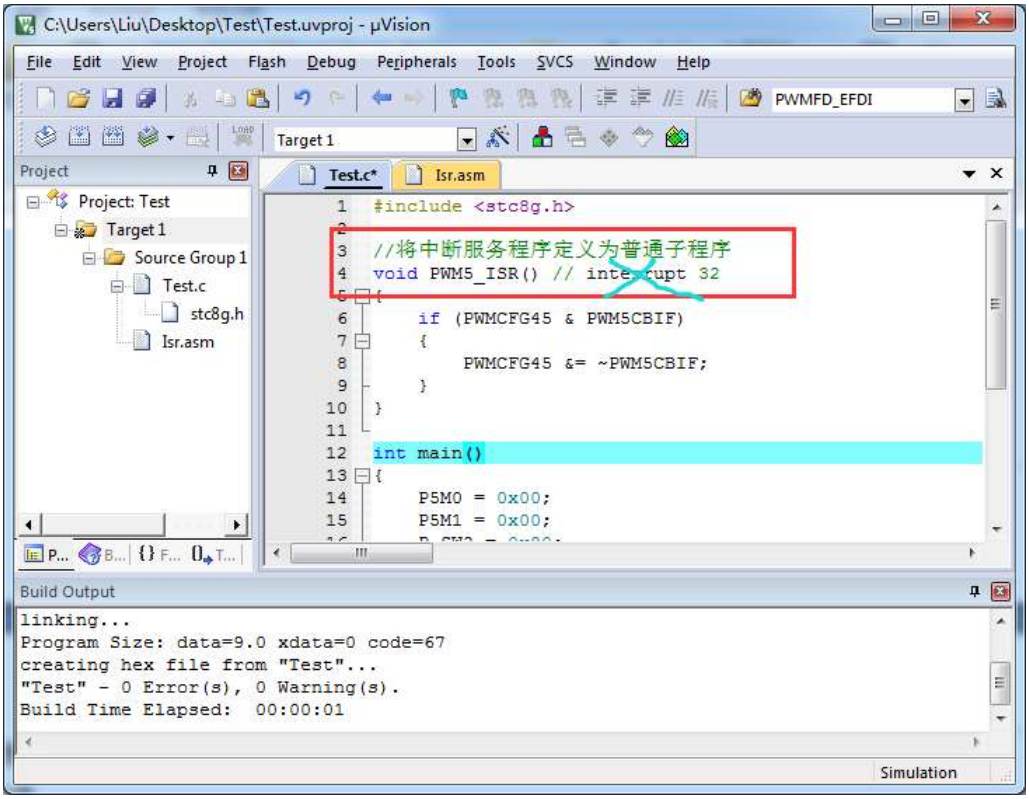


The execution effect is the same as Method 1. This method is applicable to the situation where multiple interrupt numbers greater than 31 need to be remapped.

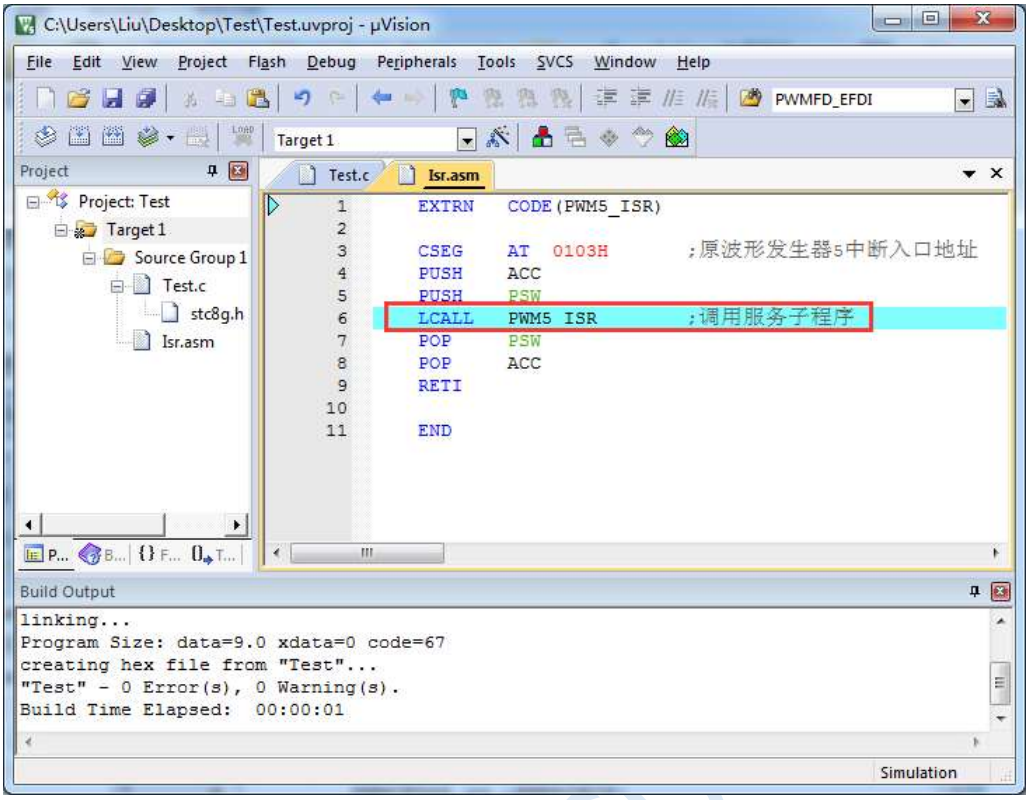
Method 3: Define the interrupt service routine as a subroutine, and then use the LCALL instruction in the interrupt entry address in the assembly code to execute the service routine.

The steps are as follows:

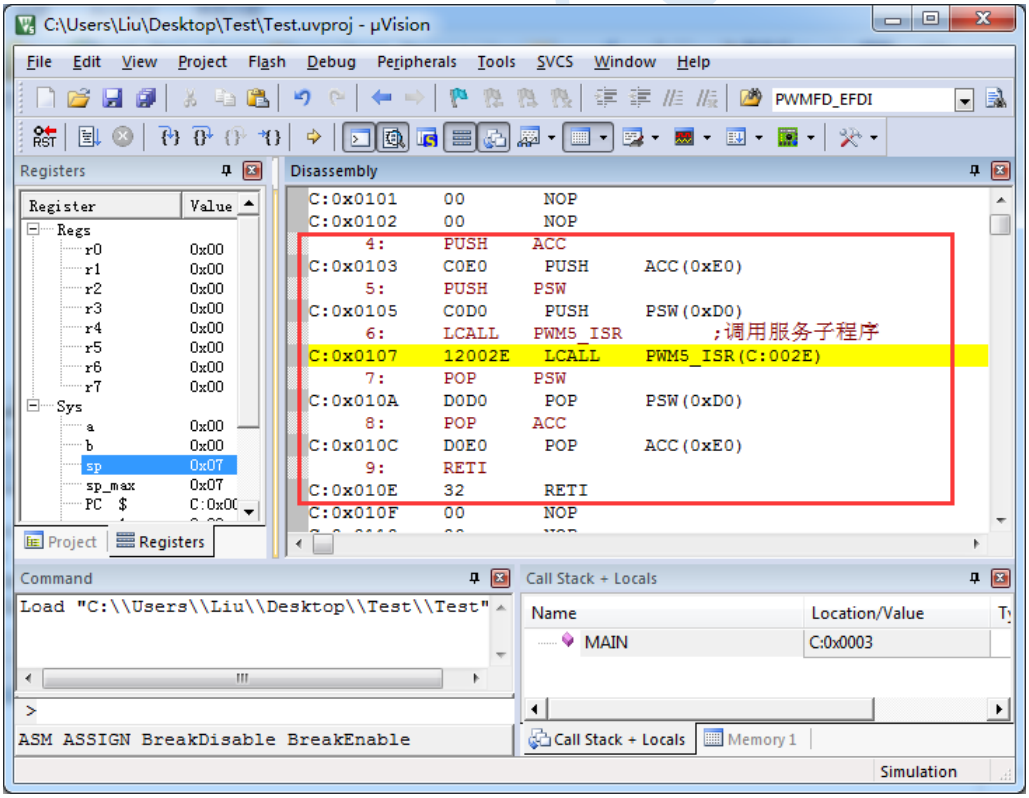
1. Remove the "interrupt" attribute from the interrupt service routine firstly and define it as an ordinary subroutine.



2. Then enter the code shown in the figure below at address 0103H of the assembly file.



3. After compiling, you can find the interrupt service routine at the address of 0103H.



This method does not need to remap interrupt entries. But there is a problem with this method. It requires the user to check the disassembly code of the C program to determine which registers need to be pushed onto the stack in the assembly file. PSW, ACC, B, DPL, DPH and R0 ~ R7 are included generally. In addition to the PSW must be pushed onto the stack, registers which are used in the user subroutine must be pushed onto the stack.

Appendix O Electrical Characteristics

● Absolute Maximum Ratings

Parameters	Minimum	Maximum	UNIT	Explanation
Storage temperature	-55	+125	℃	
Operating temperature	-40	+85	℃	Note 1
Operating Voltage	1.9	5.5	V	
VDD to ground voltage	-0.3	+5.5	V	
I/O port to ground voltage	-0.3	VDD+0.3	V	

Note 1

If the operating temperature is higher than 85℃ (such as around 125℃), because the frequency of the internal IRC clock has a large temperature drift at high temperatures, it is recommended to use an external high temperature clock or crystal oscillator. In addition, when the temperature is high, and the frequency does not run fast, it is recommended to use a working frequency below 24M. If the system must run at a higher temperature, please use an external high-reliability low-frequency active clock.

If the operating temperature is around -55℃, the operating voltage should not be too low. It is strongly recommended that the MCU-VCC voltage should not be lower than 3.0V. In addition, the power supply must also rise as fast as possible, preferably within milliseconds.

● DC ELECTRICAL CHARACTERISTICS (3.3V)

(VSS=0V, VDD=3.3V, test temperature =25℃)

Symbol	Parameter	Limits				Test Conditions
		MIN	TYP	MAX	UNIT	
I_{PD}	Power-down mode current	-	0.4	-	uA	
I_{WKT}	Power-down Wake-up timer	-	1.4	-	uA	
I_{LVD}	Low-voltage detection module	-	10	-	uA	
I_{CMP}	Comparator power consumption	-	90	-	uA	
I_{IDL}	Idle mode current (internal 32KHz)	-	0.48	-	mA	equivalent to traditional 8051 0.5M
	Idle mode current (6MHz)	-	0.88	-	mA	equivalent to traditional 8051 79M
	Idle mode current (12MHz)	-	1.00	-	mA	equivalent to traditional 8051 158M
	Idle mode current (24MHz)	-	1.16	-	mA	equivalent to traditional 8051 317M
I_{NOR}	Normal mode current (internal 32KHz)	-	0.48	-	mA	equivalent to traditional 8051 0.5M
	Normal mode current (500KHz)	-	0.88	-	mA	equivalent to traditional 8051 7M
	Normal mode current (600KHz）	-	0.88	-	mA	equivalent to traditional 8051 8M

	Normal mode current (700KHz）	-	0.90	-	mA	equivalent to traditional 8051 9M
	Normal mode current (800KHz）	-	0.91	-	mA	equivalent to traditional 8051 11M
	Normal mode current (900KHz）	-	0.91	-	mA	equivalent to traditional 8051 12M
	Normal mode current (1MHz）	-	0.94	-	mA	equivalent to traditional 8051 13M
	Normal mode current (2MHz）	-	1.05	-	mA	equivalent to traditional 8051 26M
	Normal mode current (3MHz）	-	1.17	-	mA	equivalent to traditional 8051 40M
	Normal mode current (4MHz）	-	1.26	-	mA	equivalent to traditional 8051 53M
	Normal mode current (5MHz）	-	1.40	-	mA	equivalent to traditional 8051 66M
	Normal mode current (6MHz）	-	1.49	-	mA	equivalent to traditional 8051 79M
	Normal mode current (12MHz）	-	2.09	-	mA	equivalent to traditional 8051 158M
	Normal mode current (24MHz）	-	3.16	-	mA	equivalent to traditional 8051 317M
V_{IL1}	Input low voltage	-	-	0.99	V	Turn on Schmitt trigger
		-	-	1.07	V	Turn off Schmitt trigger
V_{IH1}	Input high voltage1(general I/O)	1.18	-	-	V	Turn on Schmitt trigger
		1.09	-	-	V	Turn off Schmitt trigger
V_{IH2}	Input high voltage2(RST pin)	1.18	-	0.99	V	
I_{OL1}	Output low-level sink current	-	20	-	mA	Port voltage 0.45V
I_{OH1}	Output high level current (bi-direction mode)	200	270	-	uA	
I_{OH2}	Output high level current (Push-pull mode)	-	20	-	mA	Port voltage 2.4V
I_{IL}	Logic 0 input current	-	-	50	uA	Port voltage 0V
I_{TL}	Logical 1 to 0 transition current	100	270	600	uA	Port voltage 2.0V
R_{PU}	I/O port pull-up resistor	5.8	5.9	6.0	KΩ	
I/O speed	I/O high current drive, I/O fast conversion		25		MHz	PxDR=0, PxSR=0
	I/O low current drive, I/O fast conversion		22		MHz	PxDR=1, PxSR=0
	I/O high current drive, I/O slow conversion		16		MHz	PxDR=0, PxSR=1
	I/O low current drive, I/O slow conversion		12		MHz	PxDR=1, PxSR=1
Comparator	the fastest speed		10		MHz	Turn off all analog and digital filtering
	Analog filter time		0.1		us	
	Digital filter time		0		System clock	LCDTY＝0
			n+2			LCDTY=n (n=1~63)
I_{PD2}	Power-down mode power consumption when the comparator is enabled	-	400	-	uA	
I_{PD3}	Power-down mode power	-	470	-	uA	

	consumption when LVD is enabled					

● DC ELECTRICAL CHARACTERISTICS (5.0V)

(VSS=0V, VDD=5.0V, test temperature =25℃)

Symbol	Parameter	Limits				Test Conditions
		MIN	TYP	MAX	UNIT	
I_{PD}	Power-down mode current	-	0.6	-	uA	
I_{WKT}	Power-down Wake-up timer	-	3.6	-	uA	
I_{LVD}	Low-voltage detection module	-	30	-	uA	
I_{CMP}	Comparator power consumption	-	90	-	uA	
I_{IDL}	Idle mode current (internal 32KHz)	-	0.58	-	mA	equivalent to traditional 8051 0.5M
	Idle mode current (6MHz)	-	0.98	-	mA	equivalent to traditional 8051 79M
	Idle mode current (12MHz)	-	1.10	-	mA	equivalent to traditional 8051 158M
	Idle mode current (24MHz)	-	1.25	-	mA	equivalent to traditional 8051 317M
I_{NOR}	Normal mode current (internal 32KHz)	-	0.58	-	mA	equivalent to traditional 8051 0.5M
	Normal mode current (500KHz)		0.97		mA	equivalent to traditional 8051 7M
	Normal mode current (600KHz）		0.97		mA	equivalent to traditional 8051 8M
	Normal mode current (700KHz）		1.00		mA	equivalent to traditional 8051 9M
	Normal mode current (800KHz）		1.01		mA	equivalent to traditional 8051 11M
	Normal mode current (900KHz）		1.01		mA	equivalent to traditional 8051 12M
	Normal mode current (1MHz）		1.03		mA	equivalent to traditional 8051 13M
	Normal mode current (2MHz）		1.15		mA	equivalent to traditional 8051 26M
	Normal mode current (3MHz）		1.27		mA	equivalent to traditional 8051 40M
	Normal mode current (4MHz）		1.35		mA	equivalent to traditional 8051 53M
	Normal mode current (5MHz）		1.49		mA	equivalent to traditional 8051 66M
	Normal mode current (6MHz）	-	1.59	-	mA	equivalent to traditional 8051 79M
	Normal mode current (12MHz）	-	2.19	-	mA	equivalent to traditional 8051 158M
	Normal mode current (24MHz）	-	3.27	-	mA	equivalent to traditional 8051 317M
V_{IL1}	Input low voltage	-	-	1.32	V	Turn on Schmitt trigger
		-	-	1.48	V	Turn off Schmitt trigger
V_{IH1}	Input high voltage1(general I/O)	1.60	-	-	V	Turn on Schmitt trigger
		1.54	-	-	V	Turn off Schmitt

						trigger
V_{IH2}	Input high voltage2(RST pin)	1.60	-	1.32	V	
I_{OL1}	Output low-level sink current	-	20	-	mA	Port voltage 0.45V
I_{OH1}	Output high level current (bi-direction mode)	200	270	-	uA	
I_{OH2}	Output high level current (Push-pull mode)	-	20	-	mA	Port voltage 2.4V
I_{IL}	Logic 0 input current	-	-	50	uA	Port voltage 0V
I_{TL}	Logical 1 to 0 transition current	100	270	600	uA	Port voltage 2.0V
R_{PU}	I/O port pull-up resistor	4.1	4.2	4.4	KΩ	
I/O speed	I/O high current drive, I/O fast conversion		36		MHz	PxDR=0, PxSR=0
	I/O low 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 low current drive, I/O slow conversion		22		MHz	PxDR=1, PxSR=1
Comparator	the fastest speed		10		MHz	Turn off all analog and digital filtering
	Analog filter time		0.1		us	
	Digital filter time		0		System clock	LCDTY＝0
			n+2			LCDTY=n (n=1~63)
I_{PD2}	Power-down mode power consumption when the comparator is enabled	-	460	-	uA	
I_{PD3}	Power-down mode power consumption when LVD is enabled	-	520	-	uA	

● Internal IRC temperature drift characteristic (reference temperature 25℃)

Temperature	Range		
	MIN	TYP	MAX
-40℃～85℃		−1.38%～+1.42%	
-20℃～65℃		−0.88%～+1.05%	

● Low voltage reset threshold voltage (test temperature 25 ℃)

Level	Voltage		
	MIN	TYP	MAX
POR		（1.69V~1.82V）	
LVR0		2.0V（1.88V~1.99V）	
LVR1		2.4V（2.28V~2.45V）	
LVR2		2.7V（2.58V~2.76V）	
LVR3		3.0V（2.86V~3.06V）	

Appendix P Application note

● STC8G1K08A series

1. The STC8G1K08A series is currently mass-produced with version B chips. There is no problem, please feel free to use it.
2. The PCA interrupt shutdown instruction of STC8G1K08A series A version chips cannot be completed within one clock. The user must add an additional NOP instruction after the shutdown instruction. (Because enabling or disabling the EA total interrupt can take effect within one clock, if the user needs to mask the interrupt immediately, the method of turning off the EA can be used). This problem does not affect the normal use of the chip.
3. Special attention: Since all I/O of STC8G series (except ISP download port P3.0/P3.1) are in high-impedance input mode after power-on, the external level of I/O is not fixed. If the MCU directly enters the power-down mode/stop mode, which will cause additional power consumption for I/O. Before the MCU enters the power-down mode/stop mode, all I/O ports must be set according to the actual situation. In this mode, all unused external I/Os that are floating need to be set to quasi-bidirectional ports, and the output high level is fixed. Especially for chips with some pins, because there are some I/O ports inside the chip that are not wired to external pins, these I/Os are also in a floating state. This part of the I/O also needs to be set as a quasi-bidirectional port. And is fixed output high level.

● STC8G2K64S4/S2 series

1. For the C version of the STC8G2K64S4/S2 series currently mass-produced chips, except for the problem that the disable interrupt command of the PCA interrupt cannot be completed within one clock, the other known issues of the A and B versions have all been corrected.
2. STC8G2K64S4/S2 series currently mass-produced version B chips, PCA pulse outputing problem and P2.0/P2.1 port outputing PWM waveform problem have been corrected.
3. For STC8G2K64S4/S2 series currently mass-produced version B chips, PCA interrupt disable instruction can not be completed within one clock, the user must add an additional NOP instruction after the disable interrupt instruction. (Because enabling or disabling the EA total interrupt can take effect within one clock, if the user needs to mask the interrupt immediately, the method of turning off the EA can be used). This problem does not affect the normal use of the chip.
4. For STC8G2K64S4/S2 series currently mass-produced version B chips, when the P0.5 port is enabled to output PWM waveform, the output of P0.5 port will be immediately terminated when an external abnormality occurs, but the hardware has not set the P0.5 port to high-impedance input state, but switch to pull-up bidirectional port mode. Therefore, if there is a need to enable the PWM output function of port P0.5 in the project, please note that port P0.5 can still output a current of 20 ~ 30uA when an abnormality occurs.
5. For STC8G2K64S4/S2 series currently mass-produced version B chips, the initial value of the level conversion speed control register of all ports after power-on is 00H, that is, the default is fast flip speed after power-on, which is different from other series. The initial value of the level conversion speed control register of other series is FFH, that is, it defaults to the slow reversal speed after power-on.
6. For STC8G2K64S4/S2 series currently mass-produced version B chips, when the enhanced PWM output waveform is required, the CPU cannot enter the power saving mode, and neither IDLE mode/standby mode nor STOP mode/stop mode will work.

7. The PCA high-speed pulse output function of the STC8G2K64S4/S2 series A version chip will be affected by the same set of I/O port flips. For details, please refer to the reference code of the STC8G1K08 series in this section

8. The enhanced PWM function of STC8G2K64S4/S2 series A version of the chip has bugs in P2.0 and P2.1 ports. The other 43 I/O ports can output PWM waveforms correctly. It is recommended not to use P2.0 and P2. 1 to output PWM waveform.

9. The PCA interrupt disable instruction of STC8G2K64S4/S2 series A version chips cannot be completed within one clock. The user must add an additional NOP instruction after the disable interrupt instruction. (Because enabling or disabling the EA total interrupt can take effect within one clock, if the user needs to mask the interrupt immediately, the method of turning off the EA can be used). This problem does not affect the normal use of the chip.

10. Special attention: Since all I/O of STC8G series (except ISP download port P3.0/P3.1) are in high-impedance input mode after power-on, the external level of I/O is not fixed. If the MCU directly enters the power-down mode/stop mode at this time, it will cause additional power consumption for I/O. Before the MCU enters the power-down mode/stop mode, all I/O ports must be set according to the actual situation. In this mode, all unused I/Os that are external floating need to be set to quasi-bidirectional ports, and output high level. Especially for chips with some I/O ports inside the chip that are not wired to external pins, these I/Os are also in a floating state. This part of the I/O also needs to be set as a quasi-bidirectional port, and output high level.

● STC8G1K08 series

1. The interrupts of LVD, Timer 2, INT2, INT3 and INT4 of STC8G1K08 series C version chip and D version chip can not be disabled within one clock, the user must add 1 more NOP after the disable interrupt instruction instruction. (On the basis of the C version chip, the D version chip has modified the problem that the high-speed pulse output will be affected by the same set of I/O port flips)

2. The PCA high-speed pulse output function of the STC8G1K08 series C version chips will be affected by the same set of I/O port flips. It is recommended not to use the high-speed pulse output function (the D version of the chip does not have this problem).

3. Special attention: Since all I/O of STC8G series (except ISP download port P3.0/P3.1) are in high-impedance input mode after power-on, the external level of I/O is not fixed. If the MCU directly enters the power-down mode/stop mode at this time, it will cause additional power consumption for I/O. Before the MCU enters the power-down mode/stop mode, all I/O ports must be set according to the actual situation. In this mode, all unused I/Os that are external floating need to be set to quasi-bidirectional ports, and output high level. Especially for chips with some I/O ports inside the chip that are not wired to external pins, these I/Os are also in a floating state. This part of the I/O also needs to be set as a quasi-bidirectional port. And fixed output high level.

● STC8G1K08T series

1. STC8G1K08T series B version chip has been available for samples, please stop using A version chip and use B version chip for testing

2. Special attention: Since all I/O of STC8G series (except ISP download port P3.0/P3.1) are in high-impedance input mode after power-on, the external level of I/O is not fixed. If the MCU directly enters the power-down mode/stop mode at this time, it will cause additional power consumption for I/O. Before the MCU enters the power-down mode/stop mode, all I/O ports must be set according to the actual situation. In this mode, all unused I/Os that are external floating need to be set to quasi-bidirectional ports, and output high level. Especially for chips with some I/O ports inside the chip that are not wired to external pins, these I/Os are also in a floating state. This part of the I/O also needs to be set as a quasi-bidirectional port, and output high level.

Appendix Q PCB design guidance for touch buttons

The touch button has strict requirements for PCB design, otherwise its effect will be greatly reduced or even fail. It is recommended that users follow the following principles when designing PCB:

1. Follow the basic principles of common digital-analog hybrid circuit design.

The capacitive touch button module integrates an analog circuit for precision capacitance measurement, so it should be treated as an independent analog circuit when designing the PCB. Follow the basic principles of common digital-analog hybrid circuit design.

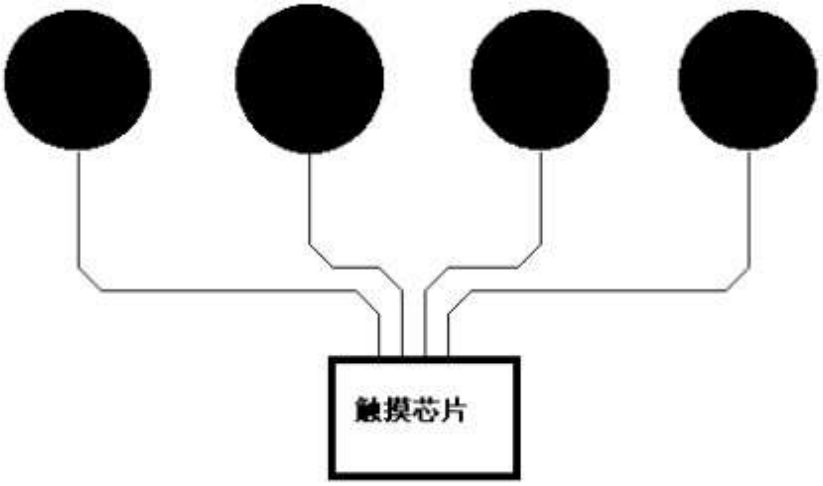
2. Use star grounding

The ground wire of the touch chip should not be shared with other circuits. It should be connected to the ground point of the power input of the board separately, which is usually referred to as "star grounding".

3. The impact of noise generated on the power supply on the touch chip

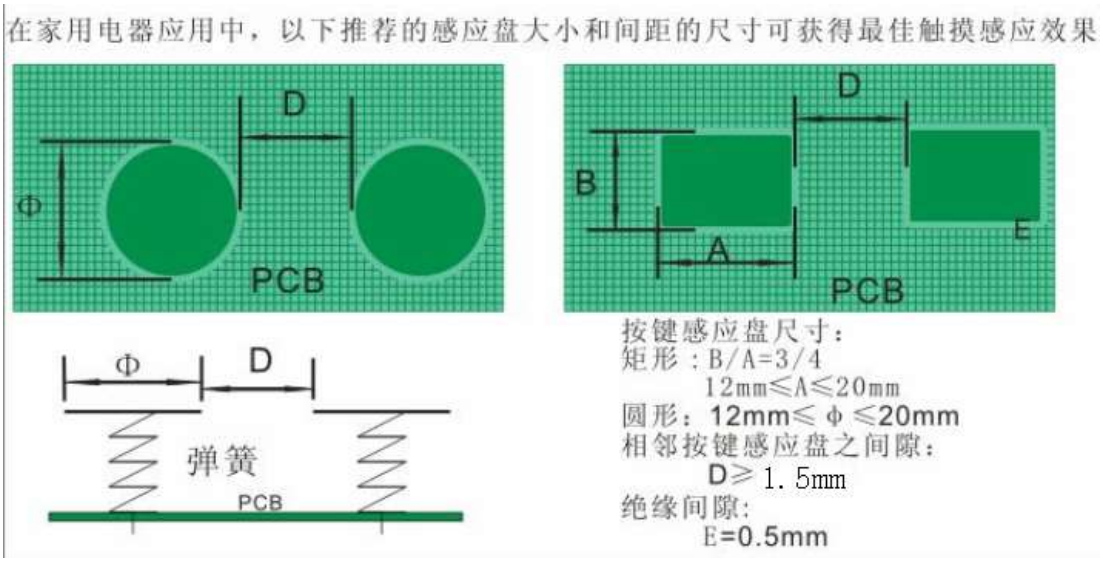
The power ripple and noise should be as small as possible. It is best to use an independent trace to take power from the power supply point of the board and add filtering measures. Do not share the power circuit with other circuits.

4. The connection between the IC and the induction plate is as long as possible, so that it has an approximate distributed capacitance, as shown in the figure below.



5. The size and gap of the key induction plate (capacitive sensor)

In the case of meeting the aesthetic design requirements of the panel, the optimal touch sensing effect must be obtained through a reasonable arrangement of the sensing plate size and the interval size. The induction plate is placed on the bottom layer, and the IC is also placed on the bottom layer. There should be no vias in the connection between the induction plate and the IC. The distance between the edges of adjacent sensing plates is preferably above 1.5mm (dimension D in the figure below). If the PCB area allows, try to use a larger distance. The distance between the copper paving and the induction plate is 0.5mm (dimension E in the figure below).



6. Copper plating

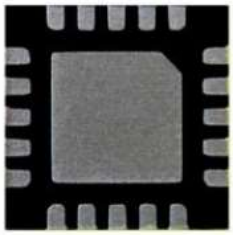
The bottom layer can be covered with grid copper or solid copper. Note that the distance between the copper and the induction plate is 0.5mm. The silk screen information of the top layer is printed on the button, and the frame shape of the silk screen is the same as that of the bottom sensor disk. The top layer corresponding to the bottom sensor disk must not be coated with copper, otherwise the touch action will be shielded. The copper on the top layer is the same as the copper on the bottom layer.

7. Wiring processing

It is better to use a smaller line width for the connection between the induction plate and the IC, such as between 10 and 15 mils. The connection between the induction pad and the touch chip should not cross the lines with strong interference, high frequency, and high current. Do not run other signal lines within 1.5mm of the connection between the sensor plate and the touch chip, the farther away the better. The top layer corresponds to the bottom layer of the induction plate and the connection line, it is best not to put any line.

Appendix R QFN/DFN packaged components welding method

In the packaging form of STC products, the more popular QFN and DFN packages have been added. Since the pins of the in this packaged chip are at the bottom of the chip, it is difficult to solder by hand. There are small companies on the market that specialize in welding engineering samples, which can undertake engineering sample proofing. If users need to weld by themselves, please refer to the following welding method.



1. Firstly, you need to prepare the following tools, soldering iron, hot air gun, tweezers, fixing frame and other tools.

2. The PCB boards and chips that need to be soldered are as follows:



3. Tin the pads of the chip on the board:



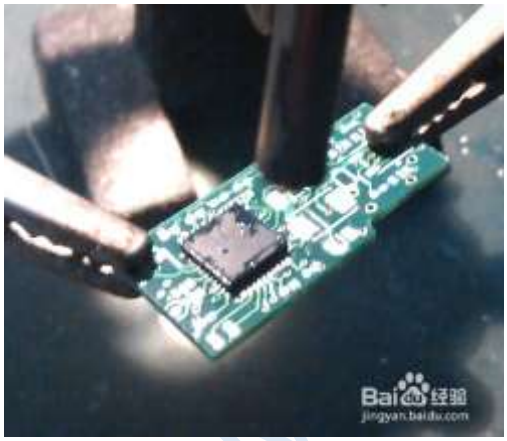
4. Then tin the bottom of the chip. After the tin is applied, the tin should be flattened, and the tin should be minimized, but it cannot be eliminated.



5. Adjust the temperature of the hot air gun, the actual air output is about 240 degrees, because the quality of the air gun is different, adjust it according to the actual situation.



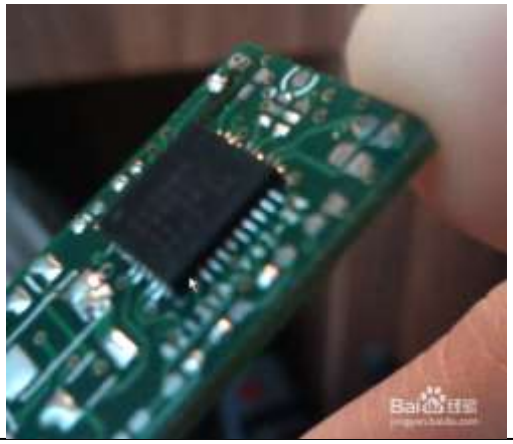
6. Put the chip on the pad, be sure to place it straight, and then blow it with a hot air gun at an even speed until the tin melts, usually within 20 seconds.

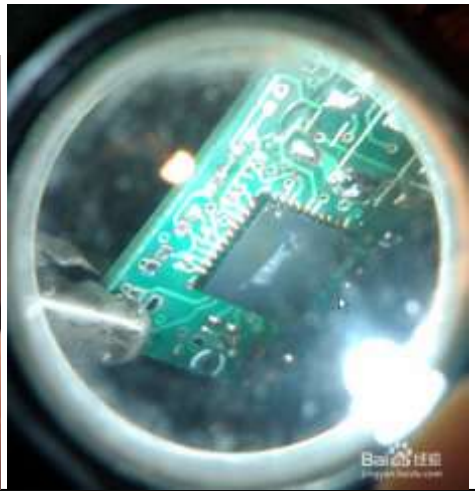


7. Use a soldering iron to tin the pins on the chip side.



8. The effect after the welding is completed





Appendix S Precautions for replacing STC15 series with STC8G series MCU

MCU instructions

The instruction code of the STC8G series is completely consistent with the STC15 series, so the code of the STC15 series is transplanted to the STC8G, and the operation is still correct, but the instruction speed of the STC8G series is faster than the STC15 series, and the instruction system of the STC15 series belongs to the STC-Y5 series of instructions, and the instructions of the STC8G series belong to the STC-Y6 series of instructions, most of the instructions of the STC-Y6 series only need one CPU clock to execute. If there is a command delay code in the user code, it needs to be adjusted. For the comparison of each instruction, please refer to the instruction table of the STC download software, as shown in the figure below:

I/O ports

After the STC8G series microcontrollers are powered on, the I/O ports mode is different from that of the STC15 series. All I/O ports of STC15 series microcontrollers are in 8051 quasi-bidirectional port mode after power-on. For STC8G series microcontrollers, except for ISP download pins P3.0/P3.1 which are quasi-bidirectional port modes, all the other I/O ports are in high-impedance input mode after power-on. The traditional 8051 and STC 15 series microcontrollers are in quasi-bidirectional port mode and output high level after power on. Often there will be moments that the motor moves or the LED flashes in the systems where I/Os are used to drive motors or LED lights when the microcontroller is powered on. The I/O of the STC8G series is in high-impedance input mode after power-on, which can avoid this kind of malfunction of the motor and LED.

Because in STC8G series microcontrollers, all the other I/O ports except ISP download pin P3.0/P3.1 which is quasi-bidirectional port mode are in high-impedance input mode after power-on, before the I/O ports of the STC8G series output signals, the two registers PxM0 and PxM1 must be used to set the I/O working mode.

Reset pin

The P5.4 port of the STC8G series and STC15 series is generally used as a normal I/O port. When the user sets P5.4 as the reset pin function during ISP download, the P5.4 port is the reset pin of the microcontroller. For the STC15H series, when the reset pin is high, the microcontroller is in the reset state, and when the reset pin is low, the microcontroller is released from the reset state. The reset levels of STC8 series and STC15H series are reversed, that is, for STC8G series, when the reset pin is low, the microcontroller is in the reset state, and when the reset is high, the microcontroller is released from the reset state.

Therefore, when the user enables the reset pin function of port P5.4, it is necessary to pay attention to the reset level.

ADC

The ADC_CONTR, ADC_RES, ADC_RESL registers addresses of STC8G series and STC15 series are the same. But the STC8G series adds two new registers: ADCCFG and ADCTIM.

STC15 series start ADC conversion bit ADC_START is located at BIT3 of register ADC_CONTR, while STC8G series is located at BIT6 of ADC_CONTR

The STC15 series ADC conversion complete flag ADC_FLAG is located at BIT4 of the register ADC_CONTR, while the STC8G series is located at BIT5 of ADC_CONTR

STC15 series ADC speed control bit ADC_SPEED is located in BIT6-BIT5 of register ADC_CONTR, and STC8G series is located in BIT3-BIT0 of ADCCFG

The alignment control bit ADRJ of the STC15 series ADC conversion result is located at BIT5 of the register CLK_DIV, while the alignment control bit RESFMT of the STC8G series is located at BIT5 of ADCCFG

The STC8G series adds a more precise ADC conversion timing control mechanism, which can be set through the register ADCTIM

EEPROM

The waiting time for EEPROM erasing and programming of STC15 series is set by Bit2-Bit0 of the register IAP_CONTR. The setting is only an approximate frequency range value. The STC8G series adds a new register IAP_TPS (SFR address: 0F5H), dedicated to setting EEPROM erasing and programming waiting time, and the user does not need to calculate, just fill in IAP_TPS directly according to the current CPU working frequency, and the hardware will automatically calculate the waiting time. (For example: the current CPU operating frequency is 24MHz, you only need to fill in 24 to IAP_TPS)

Appendix T Precautions for replacing STC8A/8F series with STC8G series MCU

I/O ports

After the STC8G series MCU is powered on, the I/O mode is different from that of the STC8A/8F series. All I/O ports of STC8A/8F series microcontollers are in the quasi-bidirectional mode of 8051 after power-on. All of the I/O ports of STC8G series microcontrollers except for the ISP download pin P3.0/P3.1 which are the quasi-bidirectional mode are in high-impedance input mode after power-on. The traditional 8051 and STC 15/8A/8F series microcontrollers are in quasi-bidirectional mode and output high level after power-on. Often there will be moments that the motor moves or the LED flashes in the systems where I/Os are used to drive motors or LED lights when the microcontroller is powered on. The I/O of the STC8G series is in high-impedance input mode after power-on, which can avoid this kind of malfunction of the motor and LED.

Because in STC8G series microcontrollers, all the other I/O ports except ISP download pin P3.0/P3.1 which is quasi-bidirectional port mode are in high-impedance input mode after power-on, before the I/O ports of the STC8G series output signals, the two registers PxM0 and PxM1 must be used to set the I/O working mode.

Reset pin

The P5.4 port of the STC8G series and STC8A/8F series is generally used as a normal I/O port. When the user sets P5.4 as the reset pin function during ISP download, the P5.4 port is the reset pin of the microcontroller. For the STC8A/8F series, when the reset pin is high, the microcontroller is in the reset state, and when the reset pin is low, the microcontroller is released from the reset state. The reset levels of STC8 series and STC15H series are reversed, that is, for STC8G series, when the reset pin is low, the microcontroller is in the reset state, and when the reset is high, the microcontroller is released from the reset state.

Therefore, when the user enables the reset pin function of port P5.4, it is necessary to pay attention to the reset level.

EEPROM

The waiting time for EEPROM erasing and programming of STC8A/8F series is set by Bit2-Bit0 of the register IAP_CONTR. The setting is only an approximate frequency range value. The STC8G series adds a new register IAP_TPS (SFR address: 0F5H), dedicated to setting EEPROM erasing and programming waiting time, and the user does not need to calculate, just fill in IAP_TPS directly according to the current CPU working frequency, and the hardware will automatically calculate the waiting time. (For example: the current CPU operating frequency is 24MHz, you only need to fill in 24 to IAP_TPS)

Appendix U Precautions for replacing STC15F/L/W series with STC15H series MCU

MCU instructions

The instruction code of the STC15H series is completely consistent with the STC15F/L/W series, so the code of the STC15F/L/W series is transplanted to the STC15H, and the operation is still correct, but the instruction speed of the STC15H series is faster than the STC15F/L/W series, and the instruction system of the STC15F/L/W series belongs to the STC-Y5 series of instructions, and the instructions of the STC15H series belong to the STC-Y6 series of instructions, most of the instructions of the STC-Y6 series only need one CPU clock to execute. If there is a command delay code in the user code, it needs to be adjusted. For the comparison of each instruction, please refer to the instruction table of the STC download software, as shown in the figure below:

I/O ports

After the STC15H series microcontrollers are powered on, the I/O ports mode is different from that of the STC15F/L/W series. All I/O ports of STC15F/L/W series microcontrollers are in 8051 quasi-bidirectional port mode after power-on. For STC15H series microcontrollers, except for ISP download pins P3.0/P3.1 which are quasi-bidirectional port modes, all the other I/O ports are in high-impedance input mode after power-on. The traditional 8051 and STC 15 series microcontrollers are in quasi-bidirectional port mode and output high level after power on. Often there will be moments that the motor moves or the LED flashes in the systems where I/Os are used to drive motors or LED lights when the microcontroller is powered on. The I/O of the STC15H series is in high-impedance input mode after power-on, which can avoid this kind of malfunction of the motor and LED.

Because in STC15H series microcontrollers, all the other I/O ports except ISP download pin P3.0/P3.1 which is quasi-bidirectional port mode are in high-impedance input mode after power-on, before the I/O ports of the STC8G series output signals, the two registers PxM0 and PxM1 must be used to set the I/O working mode.

Reset pin

The P5.4 port of the STC15H series and STC15F/L/W series is generally used as a normal I/O port. When the

user sets P5.4 as the reset pin function during ISP download, the P5.4 port is the reset pin of the microcontroller. For the STC15H series, when the reset pin is high, the microcontroller is in the reset state, and when the reset pin is low, the microcontroller is released from the reset state. The reset levels of STC8 series and STC15H series are reversed, that is, for STC15H series, when the reset pin is low, the microcontroller is in the reset state, and when the reset is high, the microcontroller is released from the reset state.

Therefore, when the user enables the reset pin function of port P5.4, it is necessary to pay attention to the reset level.

ADC

The ADC_CONTR, ADC_RES, ADC_RESL registers addresses of STC15H series and STC15F/L/W series are the same. But the STC15H series adds two new registers: ADCCFG and ADCTIM.

The STC15F/L/W series start ADC conversion bit ADC_START is located at BIT3 of register ADC_CONTR, while STC15H series is located at BIT6 of ADC_CONTR

The STC15F/L/W series ADC conversion complete flag ADC_FLAG is located at BIT4 of the register ADC_CONTR, while the STC15H series is located at BIT5 of ADC_CONTR

STC15F/L/W series ADC speed control bit ADC_SPEED is located in BIT6-BIT5 of register ADC_CONTR, and STC15H series is located in BIT3-BIT0 of ADCCFG

The alignment control bit ADRJ of the STC15F/L/W series ADC conversion result is located at BIT5 of the register CLK_DIV, while the alignment control bit RESFMT of the STC15H series is located at BIT5 of ADCCFG

The STC15H series adds a more precise ADC conversion timing control mechanism, which can be set through the register ADCTIM

EEPROM

The waiting time for EEPROM erasing and programming of STC15F/L/W series is set by Bit2-Bit0 of the register IAP_CONTR. The setting is only an approximate frequency range value. The STC15H series adds a new register IAP_TPS (SFR address: 0F5H), dedicated to setting EEPROM erasing and programming waiting time, and the user does not need to calculate, just fill in IAP_TPS directly according to the current CPU working frequency, and the hardware will automatically calculate the waiting time. (For example: the current CPU operating frequency is 24MHz, you only need to fill in 24 to IAP_TPS)

Comparators

The positive pole of the comparator in STC15W series is P5.5 and the negative pole is P5.4. The positive pole of the comparator in STC15H series is P3.7 and the negative pole is P3.6.

STC MCU

Appendix V Update Records

● 2022/3/9

1. Update data sheet

● 2022/1/25

1. Update the reference circuit diagram of TL431/CD431 in the document

● 2021/8/27

1. Correct the comment error in the sample program in the ADC chapter
2. Add sample programs to the enhanced PWM chapter

● 2021/7/19

1. Add STC15H series special models
2. The precautions for adding STC15H series to replace STC15F/L/W series

● 2021/7/7

1. Add DIP8 pin description of STC8G1K08A series
2. Add the pin description of DIP16 and DIP20 of STC8G1K08 series
3. In the enhanced PWM chapter, a sample program of "generating three complementary PWM waveforms with a dead zone with a phase difference of 120 degrees" is added.
4. In the Enhanced PWM chapter, a sample program of "PWM waveform with output duty ratio of 100% and 0%" is added.

● 2021/6/26

1. Modified the calculation formula of enhanced PWM output frequency
2. Add STC8G2K48S4 model
3. Add STC8G2K48S2 model

● 2021/5/10

1. Add the description of ADC power switch delay
2. Added the principle description and calculation formula of using the 15th channel of the ADC to reverse the input voltage of the external channel
3. Modify the wrong description of the maximum available FLASH size of some series

4. Added the description of the timer 2/3/4 interrupt flag bit

2021/3/8

1. Using STC8G1K17 model as an example how users plan their own EEPROM
2. Added STC8 series naming highlights in the appendix

2021/2/26

1. Add description about USB simulation download
2. Added descriptions of 8-bit clock prescaler registers for Timer 2, Timer 3, and Timer 4
3. The touch button sensitivity adjustment capacitor description of the touch button chip (it is recommended to use a monolithic capacitor)

2021/2/4

1. Correct the initial value of the CLKDIV register after reset
2. Update the reference circuit diagram of driving common cathode/common anode LED
3. Update the price of STC8G2K64S4-QFN32
4. Add description of initial value of special function register
5. Modify the description error in the 3V/5V device hybrid system application in the I/O port chapter
6. Add application reference circuit diagram under the pin diagram

2020/12/4

1. Correct some errors in the sample program
2. Add serial port to LIN bus example program

2020/11/25

1. Correct some errors in the sample program
2. Update interrupt structure diagram
3. Update the application notes of STC8G2K64S4/STC8G2K64S2 series
4. Add sample program for PCA module to use ECI external clock mode

2020/10/16

1. Update the price of LQFP32 package of STC8G2K64S4
2. Correct the error in the pin description of the STC8G1K08 series
3. Add the load capacitance description of the external crystal oscillator circuit

2020/9/23

1. Update application notes
2. Modify some description errors in the PCA/CCP/PWM chapter

STC MCU

2020/9/4

1. Modify some typos in the document
2. Update STC8G2K64S4/S2 application notes
3. Completes the internal hardware block diagram of timer 0/1/2/3/4 in the timer chapter
4. Correct the stop description of Mode 3 of Timer 0 (Mode 3 of Timer 0 is a non-maskable interrupt. Once it is started, it cannot be stopped by software, and the chip must be reset to stop it)
5. Organize document chapter order
6. Add STC8G2K64S4-LQFP32 pin diagram
7. "Microcontroller power supply control reference circuit" is added to the chapter of typical application circuit diagrams
8. Add 20M, 27M, 30M, 33.1776M, 35M and 36.864MHz IRC parameters to the special parameter list in Chapter 7.3 "Special Parameters in Memory"
9. Update the example code of Chapter 7.3.7 "User-defined internal IRC frequency"

2020/8/26

1. Add the chapter of timer calculation formula
2. Add the chapter of serial port baud rate calculation formula
3. Add 15-bit enhanced PWM output frequency calculation formula chapter
4. Add ADC related calculation formula chapters
5. Add 12-bit ADC static parameter reference data
6. Add the parameter of the number of clocks required for MDU16 operation
7. Add the time parameter required for EEPROM operation
8. The special function registers in all chapters are listed separately as directory subsections for easy searching
9. The precautions for adding STC8G series MCU to replace STC8A/8F series

2020/8/21

1. Modify some errors in the description of the document
2. Add 15-bit enhanced PWM feature description

2020/8/10

1. Add watchdog timer chapter
2. Organize the chapter on wake-up timer after power failure
3. Update application notes
4. Add the appendix chapter about STC download tool usage instructions

2020/8/6

1. Explain the working temperature
2. Add the application downloading circuit diagram using the universal USB to serial tool
3. Update application notes

2020/7/16

1. Add description of BUS_SPEED register
2. Add welding instructions for QFN/DFN packaged chips
3. Add EEPROM programming instructions
4. Add the method of setting U8W/U8-Mini to pass-through mode in the chapter of downloading application circuit diagram

2020/7/3

1. Add the appendix chapter, "How to reset the user program to the system area for ISP download without power failure"
2. Add an appendix chapter, "Use STC's IAP series MCU to develop your own ISP program"
3. Add the appendix chapter, "Precautions for STC8G series MCU to replace STC15 series"
4. Add appendix chapter, "Official website description"
5. In the ADC chapter, add ADC conversion timing diagram
6. Delete the enhanced PWM on ports P0/P1/P3/P4/P5 in the STC8G2K64S2 series

2020/6/15

1. Add ADC_VRef+ pin description
2. Add instructions for using diodes and resistors in the USB-to-serial reference circuit
3. Modify the description of the I/O port drive current control register PxDR (1: normal drive current; 0: strong drive current)
4. Add description of I2C slave device address

2020/6/8

1. Add the description of the fastest conversion speed of ADC
2. Detailed description of I2C bus speed setting
3. Update the reference circuit diagram of ISP download in simulated USB mode
4. Add DFN8, QFN20, QFN32, QFN48, QFN64 substrate descriptions in the package drawing
5. Add PCA outputing 10-bit PWM sample program
6. Add sample program for comparator multiplexing (comparator + ADC input) application

2020/5/29

1. Addition circuit application is added in ADC chapter
2. Add the description of the register EAXFR
3. Add the method of using a third-party application to call the release project program

2020/5/25

1. Add negative pressure detection circuit in ADC chapter
2. Fix garbled characters in some pictures

STC MCU

2020/5/20

1. Update the power consumption parameters of the clock stop mode when the low-voltage detection wake-up function is enabled in the electrical characteristics
2. Update the power consumption parameters of the clock stop mode when the comparator power-down wake-up function is enabled in the electrical characteristics
3. The ADCTIM register is added to the ADC sample program to control the internal timing of the ADC
4. Correct some typos in the document
5. Add an interrupt that can be used to wake up from clock stop mode in the features of each microcontroller series
6. Add the PWM frequency calculation formula and the method of outputting full high level and full low level in the PCA chapter
7. Added ISP download step guide in the ISP download application circuit diagram
8. Add power-down wake-up timer to wake up the power-saving mode sample program

2020/5/14

1. Add the description of comparator multiplexing
2. Add PWM trigger ADC sample program
3. Add ADC working clock frequency description in ADC chapter
4. Add ADC reference circuit diagram in ADC chapter
5. Update the power consumption parameters of low voltage detection, comparator, etc. in electrical characteristics
6. Add reference circuit diagram for power-on reset and button reset

2020/4/29

1. Change the serial port download reference circuit diagram, the series resistance on the TxD pin of the MCU is changed from 300 ohms to 100 ohms
2. Fix the error in the power supply part of the reference circuit diagram using PL2303GL for ISP download

2020/4/26

1. Modified the package size drawing of DFN8 (3mm*3mm)
2. Update I/O speed parameters in electrical characteristics
3. Update the speed parameter of the comparator in the electrical characteristics
4. Add LED driver example program
5. Add the reference pin diagram of PDIP40 of STC8G2K64S4 series and STC8G2K64S2 series
6. Correct the time point of setting TI and RI in the serial port note in chapter 13.6
7. Add the description of analog filtering and digital filtering in the chapter of Comparator
8. In Appendix E, the connection error between MAX232 and RS485 is corrected

2020/4/8

1. Add application precautions for STC8G1K08T series chips

2. Modify the parameters of STC8G2K64S4 model in the chapter "Special parameters in memory"
3. Correct the formula for calculating voltage in the chapter "Using ADC channel 15 to measure external voltage or battery voltage"
4. Amend the special function register related to LED driver of STC8G1K08T series
5. Add instructions for using the power-down wake-up timer register
6. Update the content about the overall drive current in the I/O port chapter

● 2020/3/26

1. IRC24MCR register is renamed HIRCCR
2. Add the reference circuit diagram of STC8G2K64S4-LQFP48 model which uses PL2303GL to download
3. Add STC8G2K64S4-LQFP48 model direct soft simulation USB download reference circuit diagram
4. Update the power consumption of the chip in the DC characteristics at different operating frequencies
5. Update application notes
6. Add a description at the beginning of the enhanced PWM chapter
7. Add "Touch Button PCB Design Guide" appendix chapter

● 2020/3/6

1. Rearranged the structure of the pin diagram chapter.
2. Correct the chip characteristics of STC8G1K17, STC8G1K17A models.

● 2020/3/5

1. Fixed touch key interrupt vector entry address
2. Correct touch key interrupt enable bit and special function register bit of interrupt priority

● 2020/3/4

1. Correct the part of the document about the description of the internal reference voltage.
2. Corrected some sample code errors in the PWM chapter.
3. Add the application circuit diagram of general precision ADC and high precision ADC.
4. Add the static parameters of the ADC module.
5. Add STC8G1K08T-20PIN touch key series.
6. Add touch key controller description section.
7. Add touch key reference circuit diagram.
8. Add LED driver description section.

● 2020/2/26

1. Add DFN8 package drawing.

● 2020/2/24

1. Add "Handling of compilation errors in Keil with interrupt numbers greater than 31" section

2. Add "Methods for Creating Multi-File Projects in Keil" section
3. Add "View all registers during simulation" section
4. Add application notes for STC8G2K64S4 series A chip
5. Add "Using Serial data line of I2CSDA to wake up MCU power saving mode" example code
6. Add "How to make traditional 8051 MCU learning board emulation" chapter
7. Update the content of “Directly Driving Segment LCD with I/O Port of STC Series MCU” section
8. Update model and price in "STC8G1K08-20PIN Series Features and Price"
9. Add "Added 4.1K Pull-up Resistor" chapter
10. Add "Bit-Addressable Data Memory in 8051" section to the "Memory" section

2020/1/20

1. Add "Using PCA module to implement DAC" reference circuit diagram
2. Add "a typical triode control" circuit
3. Add "Typical LED Control" circuit
4. Add the reference circuit of “Interconnecting I/O Ports of 3V/5V Devices in Mixed Voltage Power Supply System”
5. Add "How to make I / O port to low level when power on reset" reference circuit
6. Add the reference circuit of “Drive 8 digital tubes (serial extension, 3 wires) with 74HC595”
7. Add "I / O port direct drive digital LED" reference circuit
8. Add "Automatically launch ISP download after receiving user commands when running user programs" description

2020/1/17

1. Add the number clocks of MDU16 operation
2. Correct the descriptions in important notes

2020/1/15

1. Add "ADC as capacitive sensing touch key" chapter
2. Add "ADC as Key Scan Application Circuit Diagram" section
3. Add appendix "RS485 automatic control or I/O port control circuit diagram"
4. Add appendix "Part of RS485 circuit diagrams in U8W download tool"

2019/12/30

1. Update chip selection and price list
2. Add example code to generate user-specified frequency

2019/12/24

1. Modify the main control chip in the reference circuit diagram of the comparator section to STC8G series of microcontrollers
2. Modify the main control chip in the reference circuit diagram of the ADC chapter to STC8G series of microcontrollers

3. Name the channel used to measure the internal voltage in the ADC module as the 15th channel
4. Renamed the original STC8G1K08 series to STC8G1K08-20PIN series
5. Renamed the original STC8G2K64S4 series to STC8G2K64S4-48PIN series
6. Add STC8G1K08-8PIN series
7. Add STC8G1K08A-8PIN series
8. Add STC8G2K64S2-48PIN series
9. Correct the parameters of LVR voltage and electrical characteristics

2019/11/27

1. Add STC8G2K64S4 series
2. Add important instructions of STC8G1K08 series

2019/11/11

1. Add product authorization letter
2. Unified the name of Flash program memory and ROM in the document
3. Add USB download step demo (Appendix C)

2019/10/22

1. Add QFN20 pin out
2. Add QFN20 package diagram
3. Update the example program

2019/10/15

1. Correct four levels voltage of LVR
2. Correct the temperature drift range of the internal high-precision IRC
3. Correct the internal reference voltage
4. Update DC characteristics table data

2019/10/09

1. Remove the power control register (VOCTRL) part, STC8G series does not have this function
2. Modify the four-level voltage of the LVR
3. Fixed two frequency ranges of IRC

2019/8/13

1. Create STC8G Series MCU Technical Reference Manual Document

STC MCU

Appendix W STC8 series naming tidbits

STC8A: The letter "A" stands for ADC, which is the starting product of STC 12-bit ADC

STC8F: Without ADC, PWM and PCA functions, the current STC8F revised chip is fully compatible with the original STC8F pin, but the internal design has been optimized and updated, and the user needs to modify the program, so it is named STC8C

STC8C: The letter "C" represents the revised version, which is the revised chip of STC8F

STC8G: The letter "G" was originally a typo when the chip was produced, and then it was wrong. The G series is defined as the "GOOD" series, and the STC8G series is easy to learn.

STC8H: The letter "H" is taken from the first letter of the English word "High" of "高", "High" means "16-bit advanced PWM"

STC MCU

Appendix X Product Authorization Letter

Authorization Letter

To: Jiangsu Guoxin Technology Co., Ltd.

The intellectual property rights of the STC8G1K08 series products belong to Shenzhen Guoxin Artificial Intelligence Co., Ltd. Jiangsu Guoxin Technology Co., Ltd. is authorized to engage in the promotion and sales of STC8G1K08 series products in China.

Authorized by: Shenzhen Guoxin Artificial Intelligence Co., Ltd.

Authorization period: October 24, 2019-December 31, 2024

STC MCU

Independent property rights, production controllable

Shenzhen Guoxin Artificial Intelligence Co., Ltd. is a wholly-owned enterprise in the Mainland of the People's Republic of China, and is independently operated in accordance with Chinese laws and regulations. Its registered address is in Room 201, Building A, No. 1, Qianwan 1st Road, Qianhai Shenzhen-Hong Kong Cooperation Zone, Shenzhen.

The devices described in this manual are independently developed in China and have independent intellectual property rights.

The core product R&D is in China, with all design capabilities such as chip design, packaging design, structural design, reliability design, device simulation, process simulation, etc. The core product R&D team members and leaders are all composed of domestic personnel, with the R&D team taking the lead People who have been engaged in research and development for more than ten years, have long-term and stable follow-up support capabilities, and have patent certificates and software copyrights applied for in my country.

Wafer manufacturing: After the design of this device is completed, the wafer manufacturing process is completed in a wafer factory in the mainland of the People's Republic of China. It is subject to the

STC MCU

management, supervision and control of the laws and regulations of the People's Republic of China, and is completely controllable.

Packaging and manufacturing: The packaging and manufacturing of this device after the design is completed and processed in a packaging plant in the mainland of the People's Republic of China. It is subject to the management, supervision and control of the laws and regulations of the People's Republic of China, and is completely controllable.

Test: The test after the design of this device is completed in the mainland of the People's Republic of China, is subject to the management, supervision and control of the laws and regulations of the People's Republic of China, and is completely controllable.

All the key processes of this device are completed on our own production line, which can be supplied for a long time without the trouble of being cut off.

Hereby explain.

