

---

# **STC15W4K32S4 series MCU**

## **Data Sheet**

# CONTENTS

Chapter 1 General Overview of STC15W4K32S4 series .....	12
1.1 Introduction of STC15W4K32S4 series MCU .....	12
1.2 Block diagram of STC15W4K32S4 series MCU .....	15
1.3 Pin Configurations of STC15W4K32S4 series MCU .....	16
1.4 STC15W4K32S4 series Selection and Price Table.....	21
1.5 Naming rules of STC15W4K32S4 series MCU .....	22
1.6 Application Circuit Diagram for ISP of STC15W4K series .....	23
1.6.1 Application Circuit Diagram for ISP using RS-232 Converter .....	23
1.6.2 Application Circuit Diagram for ISP using USB to convert Serial Port .....	24
1.6.3 Application Circuit Diagram for ISP directly using USB port.....	25
—P3.0/P3.1 of STC15W4K series and IAP15W4K58S4 connect directly with D-/D+ of USB .....	25
1.7 Pin Descriptions of STC15W4K32S4 series MCU .....	26
1.8 Package Dimension Drawings of STC15 series MCU .....	33
1.8.1 Dimension Drawings of DFN8.....	33
1.8.2 Dimension Drawings of SOP8 .....	34
1.8.3 Dimension Drawings of DIP8 .....	35
1.8.4 Dimension Drawings of SOP16 .....	36
1.8.5 Dimension Drawings of DIP16 .....	37
1.8.6 Dimension Drawings of SOP20 .....	38
1.8.7 Dimension Drawings of TSSOP20.....	39
1.8.8 Dimension Drawings of LSSOP20.....	40
1.8.9 Dimension Drawings of DIP20 .....	41
1.8.10 Dimension Drawings of SOP28 .....	42
1.8.11 Dimension Drawings of TSSOP28.....	43
1.8.12 Dimension Drawings of SKDIP28 .....	44
1.8.13 Dimension Drawings of QFN28.....	45
1.8.14 Dimension Drawings of LQFP32.....	46
1.8.15 Dimension Drawings of SOP32 .....	47
1.8.16 Dimension Drawings of QFN32.....	48
1.8.17 Dimension Drawings of PDIP40.....	49
1.8.18 Dimension Drawings of LQFP44.....	50
1.8.19 Dimension Drawings of PLCC44.....	51

1.8.20 Dimension Drawings of PQFP44 .....	52
1.8.21 Dimension Drawings of LQFP48 .....	53
1.8.22 Dimension Drawings of QFN48 .....	54
1.8.23 Dimension Drawings of LQFP64S .....	55
1.8.24 Dimension Drawings of LQFP64L .....	56
1.8.25 Dimension Drawings of QFN64 .....	57
<b>1.9 Special Peripheral Function(CCP/SPI,UART1/2/3/4) Switch .....</b>	<b>58</b>
1.9.1 Test Porgram that Switch CCP/PWM/PCA (C and ASM) .....	60
1.9.2 Test Porgram that Switch PWM2/3/4/5/PWMFLT (C and ASM) .....	62
1.9.3 Test Porgram that Switch PWM6/PWM7 (C and ASM) .....	64
1.9.4 Test Porgram that Switch SPI (C and ASM) .....	66
1.9.5 Test Porgram that Switch UART1 (C and ASM) .....	68
1.9.6 Test Porgram that Switch UART2 (C and ASM) .....	70
1.9.7 Test Porgram that Switch UART3 (C and ASM) .....	72
1.9.8 Test Porgram that Switch UART4 (C and ASM) .....	74
<b>1.10 Global Unique Identification Number (ID) .....</b>	<b>76</b>
<b>Chapter 2 Clock, Reset and Power Management.....</b>	<b>81</b>
<b>2.1 Clock .....</b>	<b>81</b>
2.1.1 On-Chip Configurable Clock .....	81
2.1.2 Divider for System Clock .....	82
2.1.3 Programmable Clock Output (or as Frequency Divider) .....	83
2.1.3.1 Special Function Registers Related to Programmable Clock Output .....	83
2.1.3.2 Master Clock Output and Demo Program(C and ASM) .....	88
2.1.3.3 Timer 0 Programmable Clock Output and Demo Program(C and ASM) .....	91
2.1.3.4 Timer 1 Programmable Clock Output and Demo Program(C and ASM) .....	95
2.1.3.5 Timer 2 Programmable Clock Output and Demo Program (C and ASM) .....	99
2.1.3.6 Timer 3 Programmable Clock Output and Demo Program (C and ASM) .....	103
2.1.3.7 Timer 4 Programmable Clock Output and Demo Program (C and ASM) .....	104
<b>2.2 RESET Sources .....</b>	<b>105</b>
2.2.1 External RST pin Reset .....	105
2.2.2 Software Reset and Demo Program (C and ASM) .....	106
2.2.3 Power-Off / Power-On Reset (POR) .....	109
2.2.4 MAX810 Speical Circuit Reset (Power-Off/ Power-On Reset Delay) .....	109
2.2.5 Internal Low Voltage Detection Reset .....	110
2.2.6 Watch-Dog-Timer Reset .....	113
2.2.7 Reset Caused by Program Accessing an Invalid Address .....	117
2.2.8 Warm Boot and Cold Boot Reset .....	118

2.3	Power Management Modes.....	119
2.3.1	Slow Down Mode and Demo Program (C and ASM).....	120
2.3.2	Idle Mode and Demo Program (C and ASM).....	123
2.2.3	Stop / Power Down (PD) Mode and Demo Program (C and ASM).....	125
2.3.3.1	Demo Program Using Power-Down Wake-Up Timer to Wake Up Stop/PD Mode....	127
2.3.3.2	Demo Program Using External Interrupt INT0 to Wake Up Stop/PD Mode.....	129
2.3.3.3	Demo Program Using External Interrupt INT1 to Wake Up Stop/PD Mode.....	131
2.3.3.4	Demo Program Using External Interrupt $\overline{\text{INT2}}$ to Wake Up Stop/PD Mode.....	133
2.3.3.5	Demo Program Using External Interrupt $\overline{\text{INT3}}$ to Wake Up Stop/PD Mode.....	135
2.3.3.6	Demo Program Using External Interrupt $\overline{\text{INT4}}$ to Wake Up Stop/PD Mode.....	137
2.3.3.7	Program Using External Interrupt Extended by CCP/PCA to Wake Up PD Mode..	139
2.3.3.8	Program Using the Level Change of RxD pin to Wake Up Stop/PD Mode.....	143
2.3.3.9	Program Using the Level Change of RxD2 pin to Wake Up Stop/PD Mode.....	147
<b>Chapter 3</b>	<b>Memory Organization and SFRs.....</b>	<b>151</b>
3.1	Program Memory .....	151
3.2	Data Memory (SRAM) .....	152
3.2.1	On-chip Scratch-Pad RAM .....	152
3.2.2	On-Chip Expanded RAM / XRAM /AUX-RAM.....	154
3.2.3	External Expandable 64KB RAM (Off-Chip RAM).....	160
3.3	Special Function Registers.....	163
3.3.1	Special Function Registers Address Map.....	163
3.3.2	Special Function Registers Bits Description .....	164
3.3.3	Dual Data Pointer Register (DPTR).....	170
<b>Chapter 4</b>	<b>Configurable I/O Ports of STC15 series MCU .....</b>	<b>171</b>
4.1	I/O Ports Configurations .....	171
4.2	Special Explanation of P1.7/XTAL1 and P1.6/XTAL2 pin.....	174
4.3	Special Explanation of RST pin.....	174
4.4	Special Explanation of RSTOUT_LOW pin .....	174
4.5	SFRs related to I/O ports and Its Address Statement.....	175
4.6	Demo Program of STC15 series P0/P1/P2/P3/P4/P5 .....	179
4.7	I/O ports Modes .....	185
4.7.1	Quasi-Bidirectional I/O .....	185
4.7.2	Push-Pull Output .....	185
4.7.3	Input-Only (High-Impedance)Mode .....	186
4.7.4	Open-Drain Output.....	186

4.8 I/O Port Application Notes.....	186
4.9 Typical Transistor Control Circuit .....	187
4.10 Typical Diode Control Circuit.....	187
4.11 How to Make I/O Port Low after MCU Reset.....	187
4.12 Keyboard Scanning Circuit using I/O ports.....	188
4.13 Pin Function and Logic Turth Table of 74HC595.....	189
4.14 Circuit Expanding I/O ports using 74HC595.....	190
4.15 Circuit Driving 8-segment Digitron using 74HC595.....	191
4.16 Demo Program of Driving 8-Segment Digitron .....	192
— Using common I/O ports to Control 74HC595.....	192
4.17 Application Circuit using A/D Conversion to Scan Key .....	199
4.18 Demo Program using I/O ports to Simulate I <sup>2</sup> C Interface .....	200
4.18.1 Master Mode using I/O ports to Simulate I <sup>2</sup> C Interface by Software .....	200
4.18.2 Slave Mode using I/O ports to Simulate I <sup>2</sup> C Interface by Software.....	203
<b>Chapter 5. Instruction System.....</b>	<b>206</b>
5.1 Addressing Modes.....	206
5.1.1 Immediate Addressing.....	206
5.1.2 Direct Addressing .....	206
5.1.3 Indirect Addressing.....	206
5.1.4 Register Addressing.....	207
5.1.5 Inherent Addressing.....	207
5.1.6 Index Addressing .....	207
5.1.7 Bit Addressing .....	207
5.2 Instruction Set Summary.....	208
5.3 Instruction Definitions of Traditional 8051 MCU .....	214
<b>Chapter 6 Interrupt System .....</b>	<b>251</b>
6.1 Interrupt Structure .....	252
6.2 Interrupt Vector Address/Priority/Request Flag Table .....	255
6.3 How to Declare Interrupt Function in Keil C .....	256
6.4 Interrupt Registers.....	257
6.5 Interrupt Priorities .....	266
6.6 Interrupt Handling.....	268
6.7 Interrupt Nesting .....	270

6.8	External Interrupts .....	270
6.9	Interrupt Demo Program (C and ASM) .....	271
6.9.1	External Interrupt 0 (INT0) Demo Program.....	271
6.9.1.1	External Interrupt INT0 (rising + falling edge) Demo Program (C and ASM)...	271
6.9.1.2	External Interrupt INT0 (falling edge) Demo Program (C and ASM).....	273
6.9.2	External Interrupt 1 (INT1) Demo Program.....	275
6.9.2.1	External Interrupt INT1 (rising + falling edge) Demo Program (C and ASM)...	275
6.9.2.2	External Interrupt INT1 (falling edge) Demo Program (C and ASM).....	277
6.9.3	External Interrupt 2 ( <u>INT2</u> ) (falling) Demo Program (C and ASM).....	279
6.9.4	External Interrupt 3 ( <u>INT3</u> ) (falling) Demo Program (C and ASM).....	281
6.9.5	External Interrupt 4 ( <u>INT4</u> ) (falling) Demo Program (C and ASM).....	283
6.9.6	Demo Program using T0 to expand External Interrupt (Falling) .....	285
	—— T0 as Counter (C and ASM).....	285
6.9.7	Demo Program using T1 to expand External Interrupt (Falling) .....	287
	—— T1 as Counter (C and ASM).....	287
6.9.8	Demo Program using T2 to expand External Interrupt (Falling) .....	289
	—— T2 as Counter (C and ASM).....	289
6.9.9	Demo Program using CCP/PCA to expand External Interrupt .....	292
<b>Chapter 7</b>	<b>Timer/Counter .....</b>	<b>296</b>
7.1	Special Function Registers about Timer/Counter .....	297
7.2	Timer/Counter 0 Modes .....	305
7.2.1	Mode 0 (16-Bit Auto-Reload Timer/Counter) and Demo Program .....	305
7.2.1.1	Demo Program of 16-bit Auto-Reload Timer/Counter 0 (C and ASM).....	306
7.2.1.2	Demo Program of T0 Programmable Clock Output (C and ASM).....	309
	—— T0 as 16-bit Auto-Reload Timer/Counter.....	309
7.2.1.3	Demo Program using 16-bit auto-reload Timer 0 to Simulate 10 or 16 bits PWM..	312
7.2.1.4	Demo Program using T0 to expand External Interrupt (Falling edge).....	315
	—— T0 as 16-bit Auto-Reload Counter (C and ASM).....	315
7.2.2	Mode 1 (16-bit Timer/Counter) and Demo Program (C and ASM).....	317
7.2.3	Mode 2 (8-bit Auto-Reload Timer/Counter) and Demo Program .....	321
7.2.4	Mode 3 (16-bit Auto-Reload Timer/Counter whose Interrupt can not be disabled)	324
7.3	Timer/Counter 1 Modes .....	325
7.3.1	Mode 0 (16-Bit Auto-Reload Timer/Counter) and Demo Program .....	325
7.3.1.1	Demo Program of 16-bit Auto-Reload Timer/Counter 1 (C and ASM).....	326
7.3.1.2	Demo Program of T1 Programmable Clock Output (C and ASM).....	329
	—— T1 as 16-bit Auto-Reload Timer/Counter.....	329
7.3.1.3	Demo Program using 16-bit auto-reload Timer 1 as UART1 baud-rate Generator..	332

7.3.1.4 Demo Program using T1 to expand External Interrupt (Falling edge).....	338
—— T1 as 16-bit Auto-Reload Counter (C and ASM).....	338
7.3.2 Mode 1 (16-bit Timer/Counter) and Demo Programs (C and ASM) .....	340
7.3.3 Mode 2 (8-bit Auto-Reload Timer/Counter) and Demo Program .....	344
7.3.3.1 Demo Program using 8-bit auto-reload Timer 1 as UART1 baud-rate Generator....	345
7.3.3.2 Demo Program using T1 to expand External Interrupt (Falling edge).....	350
—— T1 as 8-bit Auto-Reload Counter (C and ASM).....	350
<b>7.4 Timer/Counter 2 .....</b>	<b>352</b>
7.4.1 Special Function Registers about Timer/Counter 2.....	352
7.4.2 Timer/Counter 2 as 16-Bit Auto-Reload Timer/Counter.....	355
7.5.2.1 Demo Program of 16-bit Auto-Reload Timer/Counter 2 (C and ASM).....	356
7.5.2.2 Demo Program using T2 to expand External Interrupt (Falling edge).....	359
—— T2 as 16-bit Auto-Reload Counter (C and ASM).....	359
7.4.3 Timer/Counter 2 Programmable Clock Output and Demo Program.....	362
7.4.4 Timer/Counter 2 as Baud-Rate Generator of Serial Port (UART) .....	366
7.5.4.1 Demo Program using Timer/Counter 2 as UART1 Baud-Rate Generator .....	367
7.5.4.2 Demo Program using Timer/Counter 2 as UART2 Baud-Rate Generator .....	373
<b>7.5 Timer/Counter 3 and Timer/Counter 4.....</b>	<b>379</b>
7.5.1 Special Function Registers about Timer/Counter 3 and 4.....	379
7.5.2 Timer/Counter 3 .....	381
7.5.2.1 Timer/Counter 3 as 16-Bit Auto-Reload Timer/Counter.....	381
7.5.2.2 Timer/Counter 3 Programmable Clock Output .....	382
7.5.2.3 Timer/Counter 3 as Baud-Rate Generator of Serial Port 3 (UART3) .....	383
7.5.3 Timer/Counter 4 .....	384
7.5.3.1 Timer/Counter 4 as 16-Bit Auto-Reload Timer/Counter.....	384
7.5.3.2 Timer/Counter 4 Programmable Clock Output .....	385
7.5.3.3 Timer/Counter 4 as Baud-Rate Generator of Serial Port 4 (UART4) .....	386
<b>7.6 How to Increase T0/T1/T2/T3/T4 Speed by 12 times .....</b>	<b>387</b>
<b>7.7 Programmable Clock Output (or as Frequency Divider).....</b>	<b>389</b>
7.7.1 Special Function Registers Related to Programmable Clock Output.....	389
7.7.2 Master Clock Output and Demo Program(C and ASM) .....	394
7.7.3 Timer 0 Programmable Clock Output and Demo Program.....	397
7.7.4 Timer 1 Programmable Clock Output and Demo Program.....	401
7.7.5 Timer 2 Programmable Clock Output and Demo Program.....	405
7.7.6 Timer 3 Programmable Clock Output and Demo Program.....	409
7.7.7 Timer 4 Programmable Clock Output and Demo Program.....	410
<b>7.8 Power-Down Wake-Up Special Timer and Demo Program .....</b>	<b>411</b>
<b>7.9 Application Notes for Timer in practice.....</b>	<b>416</b>

Chapter 8 Serial Port (UART) Communication.....	417
8.1 Special Function Registers about Serial Port 1 (UART1) .....	418
8.2 UART1 Operation Modes .....	423
8.2.1 Mode 0 : 8-Bit Shift Register .....	423
8.2.2 Mode 1: 8-Bit UART with Variable Baud Rate.....	425
8.2.3 Mode 2: 9-Bit UART with Fixed Baud Rate.....	428
8.2.4 Mode 3: 9-Bit UART with Variable Baud Rate.....	430
8.3 Buad Rates Setting of UART1 and Demo Program.....	432
8.4 Demo Program of UART1 (C and ASM) .....	434
8.4.1 Demo Program using T2 as UART1 Baud-Rate Generator (C&ASM) ...	434
8.4.2 Demo Program using T1 as UART1 Baud-Rate Generator(C&ASM) ....	440
—— T1 in Mode 0 (16-bit Auto-Reload Timer/Counter) ...	440
8.4.3 Demo Program using T1 as UART1 Baud-Rate Generator(C&ASM) ....	446
—— T1 in Mode 2 (8-bit Auto-Reload Timer/Counter).....	446
8.5 Frame Error Detection .....	452
8.6 Multiprocessor Communications .....	452
8.7 Automatic Address Recognition of UART1 .....	453
8.7.1 Special Fuction Registers about Automatic Address Recognition .....	453
8.7.2 Instruction of Automatic Address Recognition .....	455
8.7.3 Demo Program of Automatic Address Recognition (C and ASM) .....	458
8.8 Special Function Registers about Serial Port 2 (UART2) .....	464
8.9 UART2 Operation Modes .....	467
8.9.1 Mode 0 : 8-bit UART2 with Variable Baud-Rate.....	467
8.9.2 Mode 3: 9-bit UART2 with Variable Baud-Rate.....	467
8.10 Demo Program of UART2 (C and ASM) .....	468
----- Using Timer 2 as UART2 Baud-Rate Generator.....	468
8.11 Special Function Registers about Serial Port 3 (UART3).....	474
8.12 UART3 Operation Modes .....	478
8.12.1 Mode 0 : 8-bit UART3 with Variable Baud-Rate.....	478
8.12.2 Mode 3: 9-bit UART3 with Variable Baud-Rate.....	479
8.13 Special Function Registers about Serial Port 4 (UART4) .....	480
8.14 UART4 Operation Modes .....	484
8.14.1 Mode 0 : 8-bit UART4 with Variable Baud-Rate.....	484
8.14.2 Mode 3: 9-bit UART4 with Variable Baud-Rate.....	485



<b>Chapter 9 IAP/EEPROM Function of STC15 Series .....</b>	<b>486</b>
9.1 IAP / EEPROM Special Function Registers .....	487
9.2 STC15W4K32S4 Series Internal EEPROM Allocation Table .....	491
9.3 IAP/EEPROM Assembly Program Introduction .....	494
9.4 EEPROM Demo Program (C and ASM) .....	497
9.4.1 EEPROM Demo Program (not Transmit data by UART).....	497
9.4.2 EEPROM Demo Program (Transmit data by UART) (C and ASM) .....	505
<b>Chapter 10 Analog to Digital Converter .....</b>	<b>515</b>
10.1 A/D Converter Structure .....	515
10.2 Registers for ADC.....	517
10.3 ADC Typical Application Circuit.....	520
10.4 Application Circuit using A/D Conversion to Scan Key .....	521
10.5 ADC Reference Voltage Source.....	522
10.6 ADC Demo Program (C and ASM) .....	523
10.6.1 Demo Program (Demonstrate in ADC Interrupt Mode).....	523
10.6.2 Demo Program (Demonstrate in Polling Mode) .....	529
10.7 Circuit Diagram using SPI to Extend 12-bit ADC(TLC2543).....	537
<b>Chapter 11 Application of CCP/PCA/PWM/DAC .....</b>	<b>538</b>
11.1 Special Function Registers related with CCP/PCA/PWM.....	538
11.2 CCP/PCA/PWM Structure .....	544
11.3 CCP/PCA Modules Operation Mode .....	546
11.3.1 CCP/PCA Capture Mode .....	547
11.3.2 16-bit Software Timer Mode .....	547
11.3.3 High Speed Output Mode .....	548
11.3.4 Pulse Width Modulator Mode (PWM mode) .....	549
11.3.4.1 8-bit Pulse Width Modulator (PWM mode).....	549
11.3.4.2 7-bit Pulse Width Modulator (PWM mode).....	550
11.3.4.3 6-bit Pulse Width Modulator (PWM mode).....	552
11.4 Program using CCP/PCA to Extend External Interrupt.....	553
11.5 Demo Program for CCP/PCA acted as 16-bit Timer .....	557
11.6 Demo Program using CCP/PCA to output High Speed Pulse .....	562
11.7 Demo Program for CCP/PCA Outputing PWM (6+7+8 bit).....	567
11.8 Program achieving 9~16 bit PWM Output by CCP/PCA.....	571

11.9 Demo Program of CCP/PCA 16-bit Capture Mode .....	575
11.10 Demo Program using T0 to Simulate 10 or 16 bits PWM .....	581
——T0 as 16-bit Auto-Reload Timer/Counter .....	581
11.11 Circuit Diagram using CCP/PCA to achieve 8~16 bit DAC .....	584
Chapter 12 New 6 Channels of PWM of STC15W4K series .....	585
——High-Precision PWM with Death Time Control.....	585
12.1 Special Function Registers of New PWM Generators.....	586
12.2 Interrupts of New Enhanced PWM Generators .....	594
Chapter 13 Comparator of STC15W series MCU .....	605
13.1 Comparator Demo Program using Interrupt(C and ASM).....	608
13.2 Comparator Demo Program using Polling(C and ASM) .....	612
Chapter 14 Capacitive Sensing Touch Key.....	616
—— Achieved by ADC of STC15W series.....	616
Chapter 15 Synchronous Serial Peripheral Interface .....	637
15.1 Special Function Registers related with SPI.....	637
15.2 SPI Structure .....	640
15.3 SPI Data Communication .....	641
15.3.1 SPI Data Communication Modes .....	642
15.3.2 SPI Configuration.....	644
15.3.3 Additional Considerations for a Slave .....	645
15.3.4 Additional Considerations for a Master .....	645
15.3.5 Mode Change on $\overline{SS}$ -pin .....	645
15.3.6 Write Collision .....	646
15.3.7 SPI Clock Rate Select .....	646
15.3.8 SPI Data Mode .....	647
15.4 SPI Function Demo Program(Single Master—Single Slave).....	649
15.4.1 SPI Function Demo Program using Interrupt(C and ASM) .....	649
15.4.2 SPI Function Demo Programs using Polling mode (C and ASM) .....	655
15.5 SPI Function Demo Program(Each other as Master-Slave).....	661
15.5.1 SPI Function Demo Programs using Interrupts (C and ASM) .....	661
15.5.2 SPI Function Demo Programs using Polling.....	667
15.6 SPI Demo (Single Master Multiple Slave) .....	673

Chapter 16 Compiler / ISP Programmer / Emulator .....	683
16.1 Compiler/Assembler and Head File.....	683
16.□ ISP Programmer / Burner.....	69□
16.□1 In-System-Programming (ISP) principle.....	69□
16.□2 Application Circuit Diagram for ISP of STC15W4K32S4 series .....	69□
16.□2.1 Application Circuit Diagram for ISP using RS-232 Converter.....	69□
16.□2.2 Application Circuit Diagram for ISP using USB to convert Serial Port .....	69□
16.□2.3 Application Circuit Diagram for ISP directly using USB port .....	□□□
—P3.0/P3.1 of STC15W4K series and IAP15W4K58S4 connect directly with D-/D+ of USB .....	□□□
16.□□ PC Side Control Software Usage .....	696
16.□□ How to Release Project .....	705
16.□□ How to Encrypt User Code by Software STC15-ISP-Ver6.82.....	709
16.□□ Self-Defined Download and Demo Program .....	710
16.□ Emulator of STC15 series MCU.....	713
Chapter 17 How to Program Slave Chip by Master Chip.....	718
——the Slave Chip is only for STC15 series MCU .....	718
Appendix A: Assembly Language Programming .....	729
Appendix B: 8051 C Programming .....	751
Appendix C: Indirect addressing inner 256B RAM .....	761
Appendix D: Using Serial port to Expand I/O Ports.....	762
Appendix E: LED Driven by an I/O port and Key Scan.....	764
Appendix F: Notes of STC15 replacing Standard 8051 .....	765
Appendix G: Instruction Speed Boost Summary.....	767
Appendix H: How to reduce the Code Length by Keil C .....	773

---

# Chapter 1. General Overview of STC15W4K32S4 series

## 1.1 Introduction of STC15W4K32S4 series MCU

STC15W4K32S4 series MCU is a single-chip microcontroller based on a high performance 1T architecture 8051 CPU, which is produced by STC MCU Limited. It is a new generation of 8051 MCU with high speed, high stability, wide voltage range, low power consumption and super strong anti-disturbance. With the enhanced kernel, STC15W4K32S4 series MCU is faster than the traditional 8051 one in executing instructions (about 8~12 times the rate of the traditional 8051 MCU), and has a fully compatible instruction set with traditional 8051 series microcontroller. External expensive crystal can be removed by being integrated internal high-precise R/C clock( $\pm 0.3\%$ ) with  $\pm 1\%$  temperature drift (-40 ~+85 ) while  $\pm 0.6\%$  in normal temperature (-20 ~+65 ). External reset circuit also can be removed by being integrated internal highly reliable one with 16 levels optional threshold voltage of reset. The STC15W4K32S4 series MCU retains all features of the traditional 8051 one. In addition, it has 8-channels and 10-bits PWM, 8-channels and 10-bits A/D Converter(300 thousand times per sec.), Comparator, large capacity of 4K bytes SRAM, four high-speed asynchronous serial ports----UARTs(UART1/UART2/UART3/UART4) and a high-speed synchronous serial peripheral interface----SPI.

In Keil C development environment, please choose the Intel 8052 to compiling and only contain < reg51.h > as header file.

STC15 family with super high-speed CPU core of STC-Y5 works 20% faster than STC early 1T series (such as STC12/STC11/STC10 series) in same clock frequency.

- Enhanced 8051 Central Processing Unit, 1T, single clock per machine cycle, faster 8~12 times than the rate of a traditional 8051.
- Operating voltage range : 5.5V ~ 2.5V.
- On-chip 16K/32K/40K/48K/56K/61K/63.5K FLASH program memory with flexible ISP/IAP capability, can be repeatedly erased more than 100 thousand times.
- Large capacity of on-chip 4096 bytes SRAM: 256 byte scratch-pad RAM and 3840 bytes of auxiliary RAM
- Be capable of addressing up to 64K byte of external RAM
- On-chip EEPROM with large capacity can be repeatedly erased more than 100 thousand times.
- Dual Data Pointer (DPTR) to speed up data movement
- ISP/IAP, In-System-Programming and In-Application-Programming , no need for programmer and emulator.
- 8 channels and 10 bits Analog-to-Digital Converter (ADC), the speed up to 300 thousand times per second, 3 channels PWM also can be used as 3 channels D/A Converter(DAC).
- 6 channels 15 bits high-precision PWM (with a dead-section controller) and 2 channels CCP (The high-speed pulse function of which can be utilized to realize 11 ~ 16 bits PWM)  
---- can be used as 8 channels D/A Converter or 2 Times or 2 external Interrupts (which can be generated on rising or falling edge).
- Internal highly reliable Reset with 16 levels optional threshold voltage of reset, so that external reset circuit can be completely removed.

- 
- Internal high- precise R/C clock( $\pm 0.3\%$ ) with  $\pm 1\%$  temperature drift ( $-40 \sim +85$  ) while  $\pm 0.6\%$  ( $-20 \sim +65$  ) in normal temperature and wide frequency adjustable between 5MHz and 35MHz (5.5296MHz / 11.0592MHz / 22.1184MHz / 33.1776MHz).
  - Operating frequency range: 5- 35MHz, is equivalent to traditional 8051:60~420MHz.
  - Four high-speed asynchronous serial ports----UARTs (UART1/UART2/UART3/UART4 can be used simultaneously and regarded as 9 serial ports by shifting among 9 groups of pins):
    - UART1(RxD/P3.0, TxD/P3.1) can be switched to (RxD\_2/P3.6, TxD\_2/P3.7),  
also can be switched to (RxD\_3/P1.6, TxD\_3/P1.7);
    - UART2(RxD2/P1.0, TxD2/P1.1) can be switched to (RxD2\_2/P4.6, TxD2\_2/P4.7);
    - UART3(RxD3/P0.0, TxD3/P0.1) can be switched to (RxD3\_2/P5.0, TxD3\_2/P5.1)
    - UART4(RxD4/P0.2, TxD4/P0.3) can be switched to (RxD4\_2/P5.2, TxD4\_2/P5.3)
  - A high-speed synchronous serial peripheral interface----SPI.
  - Support the function of Encryption Download (to protect your code from being intercepted).
  - Support the function of RS485 Control
  - Code protection for flash memory access, excellent noise immunity, very low power consumption
  - Power management mode: Slow-Down mode, Idle mode(all interrupt can wake up Idle mode), Stop/Power-Down mode.
  - Timers which can wake up stop/power-down mode: have internal low-power special wake-up Timer.
  - Resource which can wake up stop/power-down mode are: INT0/P3.2, INT1/P3.3 (INT0/INT1, may be generated on both rising and falling edges), INT2/P3.6, INT3/P3.7, INT4/P3.0 (INT2/INT3/INT4, only be generated on falling edge); pins CCP0/CCP1; pins RxD/RxD2/RxD3/RxD4; pins T0/T1/T2/T3/T4(their falling edge can wake up if T0/T1/T2/T3/T4 have been enabled before power-down mode, but no interrupts can be generated); internal low-power special wake-up Timer.
  - 7 Timers/Counters: five 16-bit reloadable Timers/Counters (T0/T1/T2/T3/T4, T0 and T1 are compatible with Timer0/Timer1 of traditional 8051) and 2 Timers which maybe realized by 2 channels CCP. T0/T1/T2/T3/T4 all can independently achieve external programmable clock output (5 channels) .
  - Programmable clock output function(output by dividing the frequency of the internal system clock or the input clock of external pin):
    - The Programmable clock output of T0 is on P3.5/T0CLKO (output by dividing the frequency of the internal system clock or the input clock of external pin T0/P3.4)
    - The Programmable clock output of T1 is on P3.4/T1CLKO (output by dividing the frequency of the internal system clock or the input clock of external pin T1/P3.5)
    - The Programmable clock output of T2 is on P3.0/T2CLKO (output by dividing the frequency of the internal system clock or the input clock of external pin T2/P3.1)
    - The Programmable clock output of T3 is on P0.4/T3CLKO (output by dividing the frequency of the internal system clock or the input clock of external pin T3/P0.5)
-

---

The Programmable clock output of T4 is on P0.6/T4CLKO (output by dividing the frequency of the internal system clock or the input clock of external pin T4/P0.7)

Five timers/counters in above all can be output by dividing the frequency from 1 to 65536.

The Programmable clock output of master clock is on P5.4/MCLKO, and its frequency can be divided into MCLK/1, MCLK/2, MCLK/4, MCLK/16.

The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator.

MCLK is the frequency of master clock. MCLKO is the output of master clock.

- **Comparator**, which can be used as 1 channel ADC or brownout detect function and support comparing by external pin CMP+ and CMP- or internal reference voltage and generating output signal (its polarity can be configured) on CMPO pin.
- One 15 bits Watch-Dog-Timer with 8-bit pre-scaler (one-time-enabled)
- advanced instruction set, which is fully compatible with traditional 8051 MCU, have hardware multiplication / division command.
- 62/46/42/38/30/26 common I/O ports are available, their mode is quasi\_bidirectional/weak pull-up (traditional 8051 I/O ports mode) after reset, and can be set to four modes: quasi\_bidirectional/weak pull-up, strong push-pull/ strong pull-up, input-only/high-impedance and open drain.

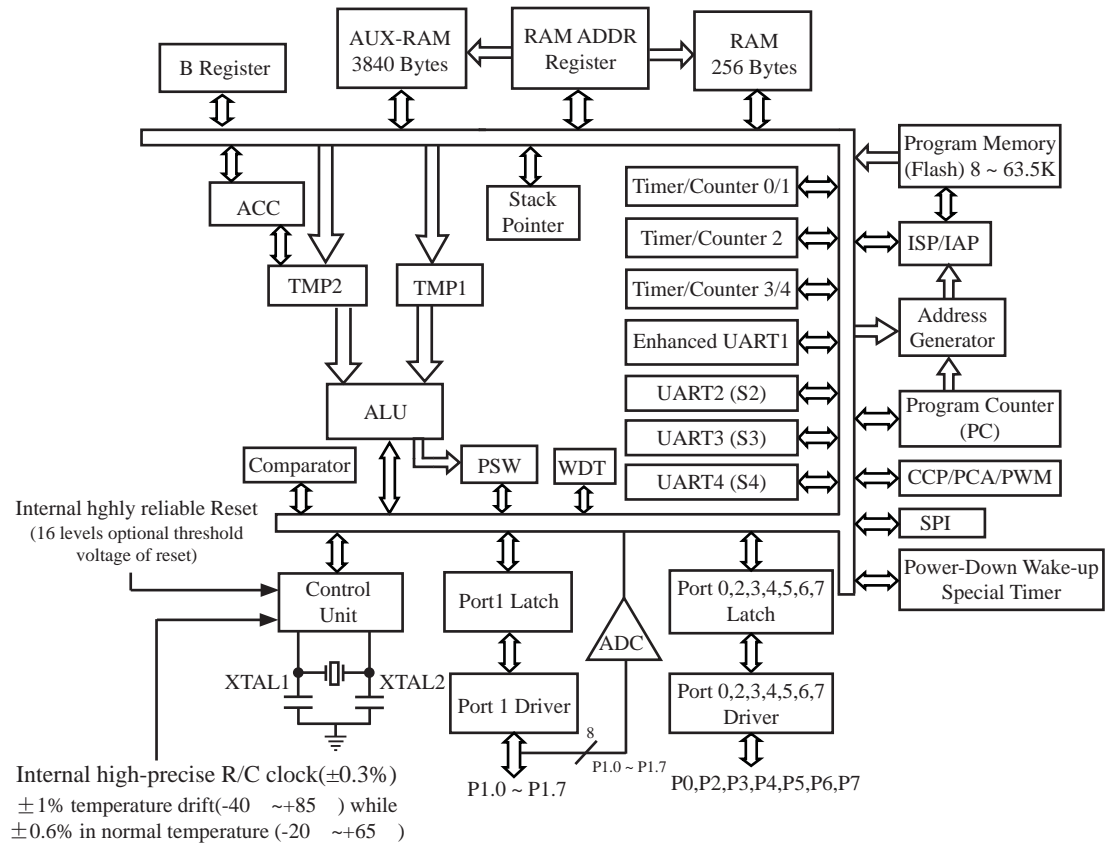
the driving ability of each I/O port can be up to 20mA, but it don't exceed this maximum 120mA that the current of the whole chip of 40-pin or more than 40-pin MCU, while 90mA that the current of the whole chip of 16-pin or more than 16-pin MCU or 32-pin or less than 32-pin MCU.

If I/O ports are not enough, it can be extended by connecting a 74HC595(reference price: RMB 0.21 yuan). Besides, cascading several chips also can extend to dozens of I/O ports.

- Package: LQFP64L(16mm x 16mm), LQFP64S(12mm x 12mm), LQFP48(9mm x 9mm), LQFP44(12mm x 12mm), LQFP32(9mm x 9mm), SOP28, SKDIP28, PDIP40.
- **All products are baked 8 hours in high-temperature 175 after be packaged, Manufacture guarantee good quality.**
- In Keil C development environment, select the Intel 8052 to compiling and only contain < reg51.h > as header file.

## 1.2 Block diagram of STC15W4K32S4 series MCU

The internal structure of STC15W4K32S4 series MCU is shown in the block diagram below. STC15W4K32S4 series MCU includes central processor unit(CPU), program memory (Flash), data memory(SRAM), Timers/ Counters, I/O ports, high-speed A/D converter(ADC), Comparator, Watchdog, high-speed asynchronous serial communication ports---UART(UART1/UART2/UART3/UART4), CCP/PWM/PCA, a group of high-speed synchronous serial peripheral interface (SPI), internal high- precise R/C clock, internal highly reliable Reset and so on. STC15W4K32S4 series MCU almost includes all of the modules required in data acquisition and control, so can be regarded as an on-chip system (SysTem Chip or SysTem on Chip, abbreviated as STC, this is the name origin of Hongjing technology STC Limited).

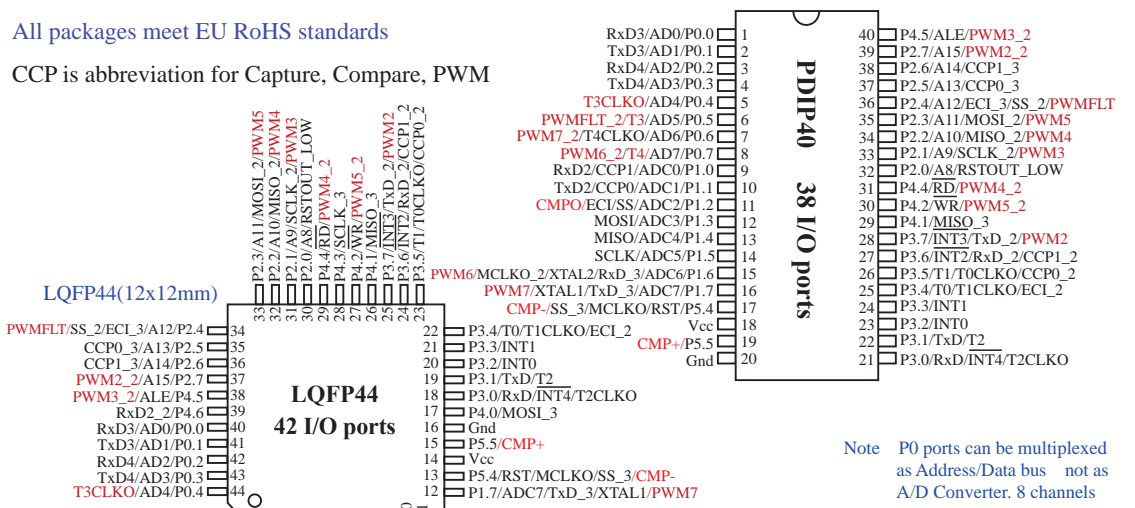


STC15W4K32S4 series Block Diagram

# 1.3 Pin Configurations of STC15W4K32S4 series MCU

All packages meet EU RoHS standards

CCP is abbreviation for Capture, Compare, PWM



Note P0 ports can be multiplexed as Address/Data bus not as A/D Converter. 8 channels of A/D Converter are on P1.

MCLKO is the output of master clock whose frequency can be divided into MCLK/1, MCLK/2, MCLK/4, MCLK/16. The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator. MCLK is the frequency of master clock.

Recommend UART1 on [P3.6/RxD\_2, P3.7/TxD\_2] or [P1.6/RxD\_3/XTAL2, P1.7/TxD\_3/XTAL1]

Consequently P0.x/ADx means that P0.x can be used as Address/Data bus, while P1.x/ADCx means P1.x can be used as A/D conversion channel in the pin map.

T0CLKO refers to the programmable clock output of Timer/Counter 0 (output by dividing the frequency of the internal system clock or the input clock of external pin T0/P3.4);

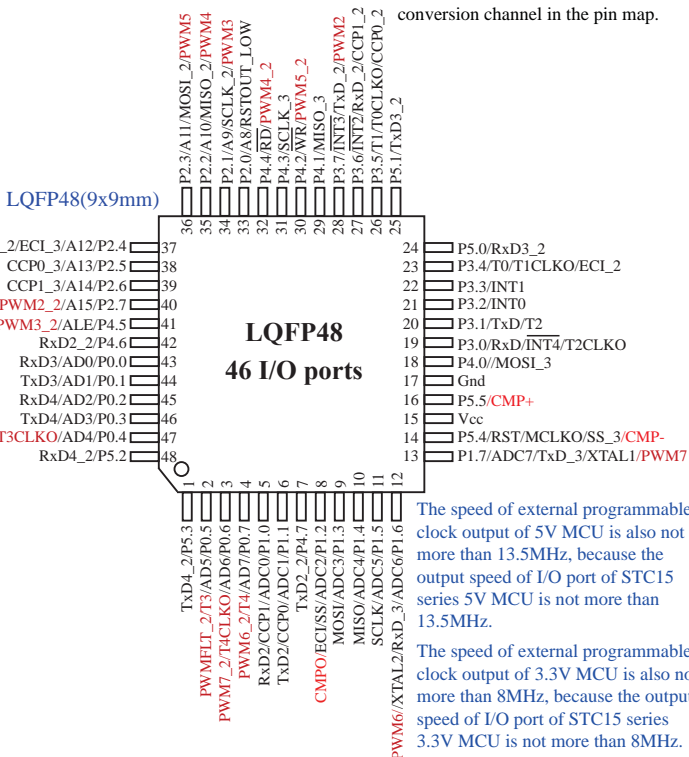
T1CLKO refers to the programmable clock output of Timer/Counter 1 (output by dividing the frequency of the internal system clock or the input clock of external pin T1/P3.5);

T2CLKO refers to the programmable clock output of Timer/Counter 2 (output by dividing the frequency of the internal system clock or the input clock of external pin T2/P3.1);

T3CLKO refers to the programmable clock output of Timer/Counter 3 (output by dividing the frequency of the internal system clock or the input clock of external pin T3/P0.5);

T4CLKO refers to the programmable clock output of Timer/Counter 4 (output by dividing the frequency of the internal system clock or the input clock of external pin T4/P0.7).

In addition to programmable output on the internal system clock, T0CLKO/T1CLKO/T2CLKO/T3CLKO/T4CLKO also can be used as divider by dividing the frequency of the internal system clock or the input clock of external pin T0/T1/T2/T3/T4.



The speed of external programmable clock output of 5V MCU is also not more than 13.5MHz, because the output speed of I/O port of STC15 series 5V MCU is not more than 13.5MHz.

The speed of external programmable clock output of 3.3V MCU is also not more than 8MHz, because the output speed of I/O port of STC15 series 3.3V MCU is not more than 8MHz.



T0CLKO refers to the programmable clock output of Timer/Counter 0 (output by dividing the frequency of the internal system clock or the input clock of external pin T0/P3.4);

T1CLKO refers to the programmable clock output of Timer/Counter 1 (output by dividing the frequency of the internal system clock or the input clock of external pin T1/P3.5);

T2CLKO refers to the programmable clock output of Timer/Counter 2 (output by dividing the frequency of the internal system clock or the input clock of external pin T2/P3.1);

T3CLKO refers to the programmable clock output of Timer/Counter 3 (output by dividing the frequency of the internal system clock or the input clock of external pin T3/P0.5);

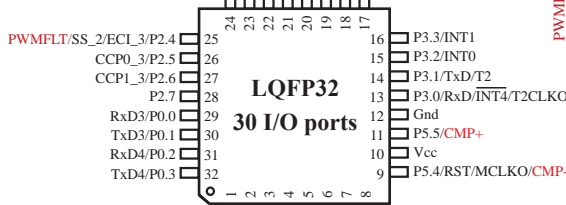
T4CLKO refers to the programmable clock output of Timer/Counter 4 (output by dividing the frequency of the internal system clock or the input clock of external pin T4/P0.7).

In addition to programmable output on the internal system clock, T0CLKO/T1CLKO/T2CLKO/T3CLKO/T4CLKO also can be used as divider by dividing the frequency of the internal system clock or the input clock of external pin T0/T1/T2/T3/T4.

8 channels of A/D Converter are on P1.

P1.x/ADCx means P1.x can be used as A/D conversion channel in the pin map.

LQFP32(9x9mm)

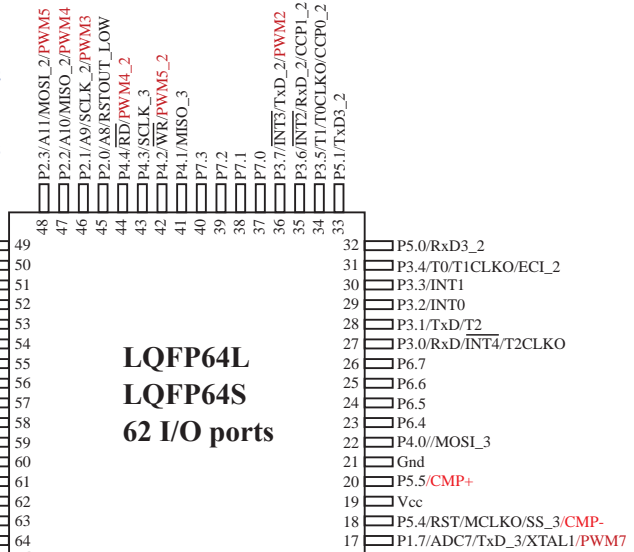


MCLKO is the output of master clock whose frequency can be divided into MCLK/1, MCLK/2, MCLK/4, MCLK/16. The master clock can either be internal R/C clock or the external crystal oscillator. MCLK is the frequency of master clock.

Recommend UART1 on [P3.6/RxD\_2, P3.7/TxD\_2] or [P1.6/RxD\_3/XTAL2, P1.7/TxD\_3/XTAL1]

All packages meet EU RoHS standards

CCP is abbreviation for Capture, Compare, PWM



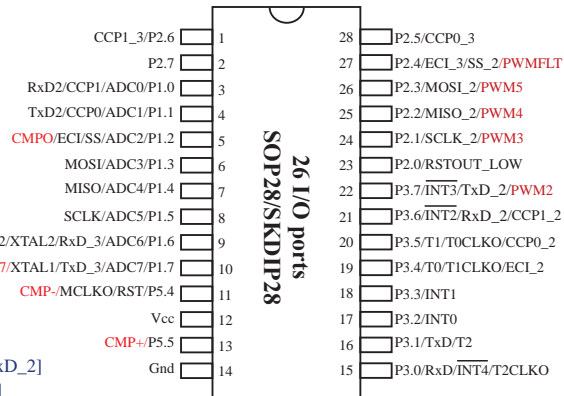
LQFP64L  
LQFP64S  
62 I/O ports

LQFP64L(16x16mm)

LQFP64S(12x12mm)

The speed of external programmable clock output of 5V MCU is also not more than 13.5MHz, because the output speed of I/O port of STC15 series 5V MCU is not more than 13.5MHz.

The speed of external programmable clock output of 3.3V MCU is also not more than 8MHz, because the output speed of I/O port of STC15 series 3.3V MCU is not more than 8MHz.



26 I/O ports  
SOP28/SKIDP28

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1_P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000 0000
P_SW2	BAH	Peripheral function switch			PWM67_S	PWM2345_S		S4_S	S3_S	S2_S	xxxx xxxx
CLK_DIV (PCON2)	97H	Clock Division register	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000
INT_CLKO (AUXR2)	8FH	External Interrupt enable and Clock output register	-	EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO	x000 0000

UART1/S1 can be switched in 3 groups of pins by selecting the control bits S1_S0 and S1_S1.		
S1_S1	S1_S0	UART1/S1 can be switched between P1 and P3
0	0	UART1/S1 on [P3.0/RxD,P3.1/TxD]
0	1	UART1/S1 on [P3.6/RxD_2,P3.7/TxD_2]
1	0	UART1/S1 on [P1.6/RxD_3/XTAL2,P1.7/TxD_3/XTAL1] when UART1 is on P1, please using internal R/C clock.
1	1	Invalid

Recommmed UART1 on [P3.6/RxD\_2,P3.7/TxD\_2] or [P1.6/RxD\_3/XTAL2,P1.7/TxD\_3/XTAL1].

UART2/S2 can be switched in 2 groups of pins by selecting the control bit S2_S.	
S2_S	UART2/S2 can be switched between P1 and P4
0	UART2/S2 on [P1.0/RxD2,P1.1/TxD2]
1	UART2/S2 on [P4.6/RxD2_2,P4.7/TxD2_2]

UART3/S3 can be switched in 2 groups of pins by selecting the control bit S3_S.	
S3_S	UART3/S3 can be switched between P0 and P5
0	UART3/S3 on [P0.0/RxD3,P0.1/TxD3]
1	UART3/S3 on [P5.0/RxD3_2,P5.1/TxD3_2]

UART4/S4 can be switched in 2 groups of pins by selecting the control bit S4_S.	
S4_S	UART4/S4 can be switched between P0 and P5
0	UART4/S4 on [P0.2/RxD4,P0.3/TxD4]
1	UART4/S4 on [P5.2/RxD4_2,P5.3/TxD4_2]

SPI can be switched in 3 groups of pins by selecting the control bits SPI_S1 and SPI_S0		
SPI_S1	SPI_S0	SPI can be switched in P1 and P2 and P4
0	0	SPI on [P1.2/SS,P1.3/MOSI,P1.4/MISO,P1.5/SCLK]
0	1	SPI on [P2.4/SS_2,P2.3/MOSI_2,P2.2/MISO_2,P2.1/SCLK_2]
1	0	SPI on [P5.4/SS_3,P4.0/MOSI_3,P4.1/MISO_3,P4.3/SCLK_3]
1	1	Invalid

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000 0000
P_SW2	BAH	Peripheral function switch			PWM67_S	PWM2345_S		S4_S	S3_S	S2_S	xxxx x000
CLK_DIV (PCON2)	97H	Clock Division register	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000

CCP can be switched in 3 groups of pins by selecting the control bits CCP\_S1 and CCP\_S0.

CCP_S1	CCP_S0	CCP can be switched in P1 and P2 and P3
0	0	CCP on [P1.2/ECI,P1.1/CCP0,P1.0/CCP1]
0	1	CCP on [P3.4/ECI_2,P3.5/CCP0_2,P3.6/CCP1_2]
1	0	CCP on [P2.4/ECI_3,P2.5/CCP0_3,P2.6/CCP1_3]
1	1	Invalid

PWM2/PWM3/PWM4/PWM5/PWMFLT can be switched in 2 groups of pins by selecting the control bit PWM2345\_S.

PWM2345_S	PWM2/PWM3/PWM4/PWM5/PWMFLT can be switched between P2, P3, and P4
0	PWM2/PWM3/PWM4/PWM5/PWMFLT on [P3.7/PWM2, P2.1/PWM3, P2.2/PWM4, P2.3/PWM5, P2.4/PWMFLT]
1	PWM2/PWM3/PWM4/PWM5/PWMFLT on [P2.7/PWM2_2, P4.5/PWM3_2, P4.4/PWM4_2, P4.2/PWM5_2, P0.5/PWMFLT_2]

PWM6/PWM7 can be switched in 2 groups of pins by selecting the control bit PWM67\_S.

PWM67_S	PWM2/PWM3/PWM4/PWM5/PWMFLT can be switched between P0 and P1
0	PWM6/PWM7 on [P1.6/PWM6,P1.7/PWM7]
1	PWM6/PWM7 on [P0.7/PWM6_2,P0.6/PWM7_2]

DPS DPTR registers select bit.

- 0 DPTR0 is selected
- 1 DPTR1 is selected

ADRJ the adjustment bit of ADC result

- 0 ADC\_RES[7:0] store high 8-bit ADC result ADC\_RESL[1:0] store low 2-bit ADC result
- 1 ADC\_RES[1:0] store high 2-bit ADC result ADC\_RESL[7:0] store low 8-bit ADC result

Tx\_Rx the set bit of relay and broadcast mode of UART1

- 0 UART1 works on normal mode
- 1 UART1 works on relay and broadcast mode that to say output the input level state of RxD port to the outside TxD pin in real time, namely the external output of TxD pin can reflect the input level state of RxD port.

the RxD and TxD of UART1 can be switched in 3 groups of pins: [RxD/P3.0, TxD/P3.1];

[RxD\_2/P3.6, TxD\_2/P3.7];

[RxD\_3/P1.6, TxD\_3/P1.7].

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
CLK_DIV (PCON2)	97H	Clock Division register	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCKO_2	CLKS2	CLKS1	CLKS0	0000 0000
INT_CLKO (AUXR2)	8FH	External Interrupt enable and Clock output register	-	EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO	x000 0000

MCKO_S2	MCKO_S1	MCKO_S0	the control bit of master clock output by dividing the frequency (The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator)
0	0	0	Master clock do not output external clock
0	0	1	Master clock output external clock but its frequency do not be divided and the output clock frequency = MCLK / 1
0	1	0	Master clock output external clock but its frequency is divided by 2 and the output clock frequency = MCLK / 2
0	1	1	Master clock output external clock but its frequency is divided by 4 and the output clock frequency = MCLK / 4
1	0	0	Master clock output external clock but its frequency is divided by 4 and the output clock frequency = MCLK / 16

The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator.  
MCLK is the frequency of master clock.

STC15W4K32S4 series MCU output master clock on MCLKO/P5.4

MCKO\_2 to select Master Clock output on where

0 Master Clock output on MCLKO/P5.4

1 Master Clock output on MCKO\_2/XTAL2/P1.6

The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator.

CLKS2	CLKS1	CLKS0	the control bit of system clock (System clock refers to the master clock that has been divided frequency, which is offered to CPU, UARTs, SPI, Timers, CCP/PWM/PCA and A/D Converter)
0	0	0	Master clock frequency/1, No division
0	0	1	Master clock frequency/2
0	1	0	Master clock frequency/4
0	1	1	Master clock frequency/8
1	0	0	Master clock frequency/16
1	0	1	Master clock frequency/32
1	1	0	Master clock frequency/64
1	1	1	Master clock frequency/128

The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator.

# 1.4 STC15W4K32S4 series Selection and Price Table

Type 1T 8051 MCU	Operating Voltage (V)	Flash (byte)	SRAM (byte)	UART	Common Timers T0-T4	8 channels PWM		Special Power- down Wake- up Timer	Standard External Interrupts	A/D 8-channel	C O M P A R A T O R	EEP ROM	Internal Low- Voltage Detection Interrupt	W D T	Internal High- reliable Reset (with optional threshold voltage)	Internal High- Precision Clock	Output clock and reset signal from MCU	Encryption Download (to protect your code from being intercepted)	RS485 Control	All Packages LQFP64/LQFP48/ LQFP44/PDIP40 LQFP32/SOP28/ SKDIP28					
						15-bit special PWM (with a dead- section controller)	10-bit CCP													Price of a part of packages (RMB ¥)					
STC15W4K32S4 series MCU Selection and Price Table																									
Note: 8 channels PWM can be used as 8 channels DAC, 2 channels CCP can be used as 2 Timers or 2 external interrupts.																									
STC15W4K16S4	5.5-2.5	16K	4K	4	Y	5	6-ch	2-ch	Y	5	10 bits	Y	2	45K	Y	Y	16-level	Y	Y	Y	Y	¥5.7	¥5.2	¥5.2	¥5.4
STC15W4K32S4	5.5-2.5	32K	4K	4	Y	5	6-ch	2-ch	Y	5	10 bits	Y	2	29K	Y	Y	16-level	Y	Y	Y	Y	¥5.9	¥5.5	¥5.5	¥5.7
STC15W4K40S4	5.5-2.5	40K	4K	4	Y	5	6-ch	2-ch	Y	5	10 bits	Y	2	21K	Y	Y	16-level	Y	Y	Y	Y	¥5.9	¥5.6	¥5.6	¥5.8
STC15W4K48S4	5.5-2.5	48K	4K	4	Y	5	6-ch	2-ch	Y	5	10 bits	Y	2	13K	Y	Y	16-level	Y	Y	Y	Y	¥5.9	¥5.6	¥5.6	¥5.8
STC15W4K56S4	5.5-2.5	56K	4K	4	Y	5	6-ch	2-ch	Y	5	10 bits	Y	2	5K	Y	Y	16-level	Y	Y	Y	Y	¥5.9	¥5.6	¥5.6	¥5.8
IAP15W4K58S4 (which itself is a emulator)	5.5-2.5	58K	4K	4	Y	5	6-ch	2-ch	Y	5	10 bits	Y	2	IAP	Y	Y	16-level	Y	Y	Y	Y	¥5.9	¥5.6	¥5.6	¥5.8
IAP15W4K61S4 (which itself is a emulator)	5.5-2.5	61K	4K	4	Y	5	6-ch	2-ch	Y	5	10 bits	Y	2	IAP	Y	Y	16-level	Y	Y	Y	Y	¥5.9	¥5.6	¥5.6	¥5.8
IRC15W4K63S4 (Using external crystal or internal 24MHz clock)	5.5-2.5	63.5K	4K	4	Y	5	6-ch	2-ch	Y	5	10 bits	Y	2	IAP	Y	Y	Fixed	Y	Y	N	N	¥5.9	¥5.6	¥5.6	¥5.8

**Encryption Download** : please burn source code with encryption key onto MCU in the factory. Then, you can make a simple update software just with one "update" button by firstly using the fuction "encyrtion download" and then "release project" to update yourself code unable to be intercepted when you need to upgrade your code.

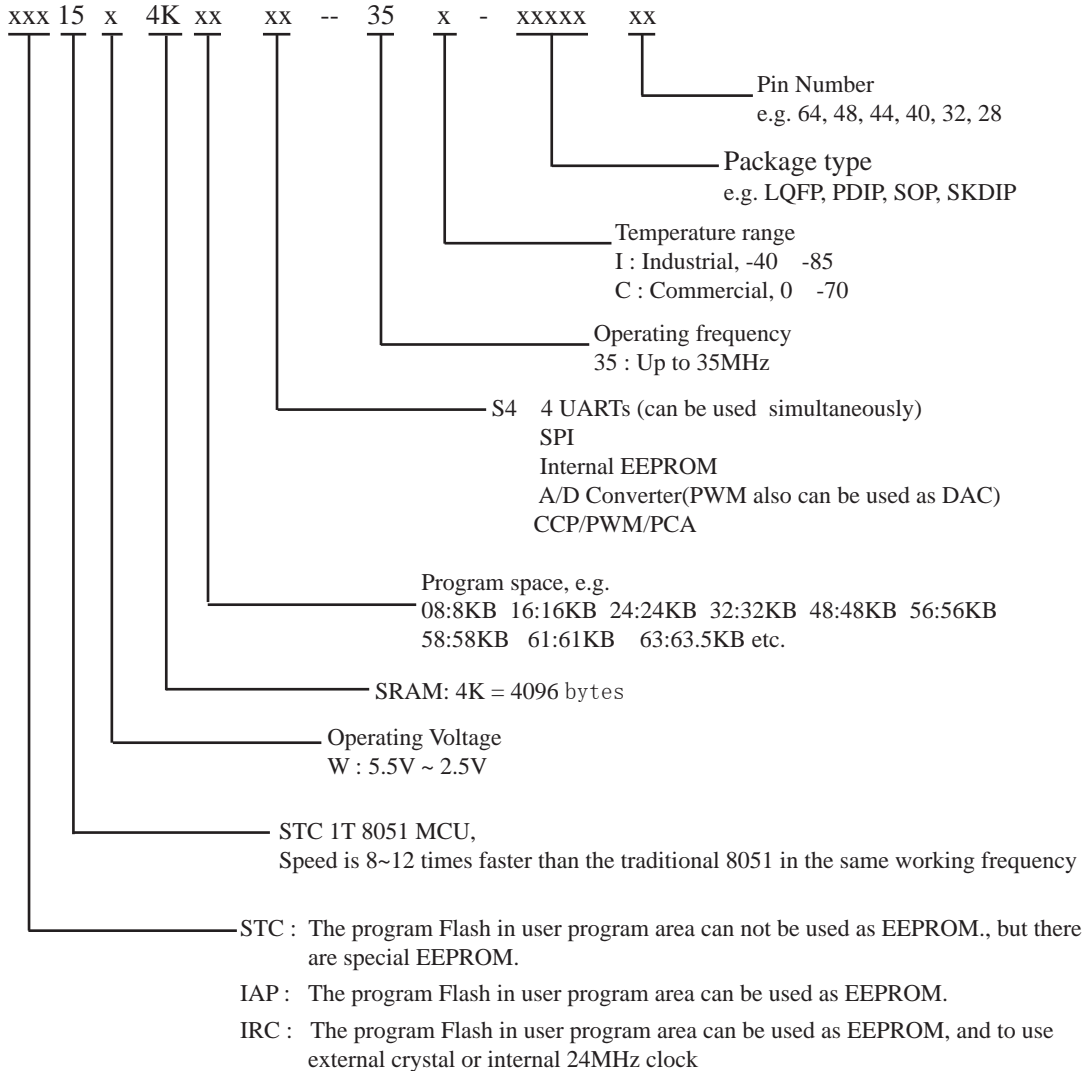
If user wants to use 40-pin and above MCU, LQFP-44 is suggested, while PDIP-40 is still supplied normal ; if user wants to use the 32-pin MCU, LQFP-32 is recommended; if user wants to use the 28-pin MCU, SOP-28 is recommended.

### To provide customized IC services

Because the last 7 bytes of the program area is stored mandatorily the contents of only global ID, the program space the user can actually use is 7 bytes smaller than the space shown in the selection table.

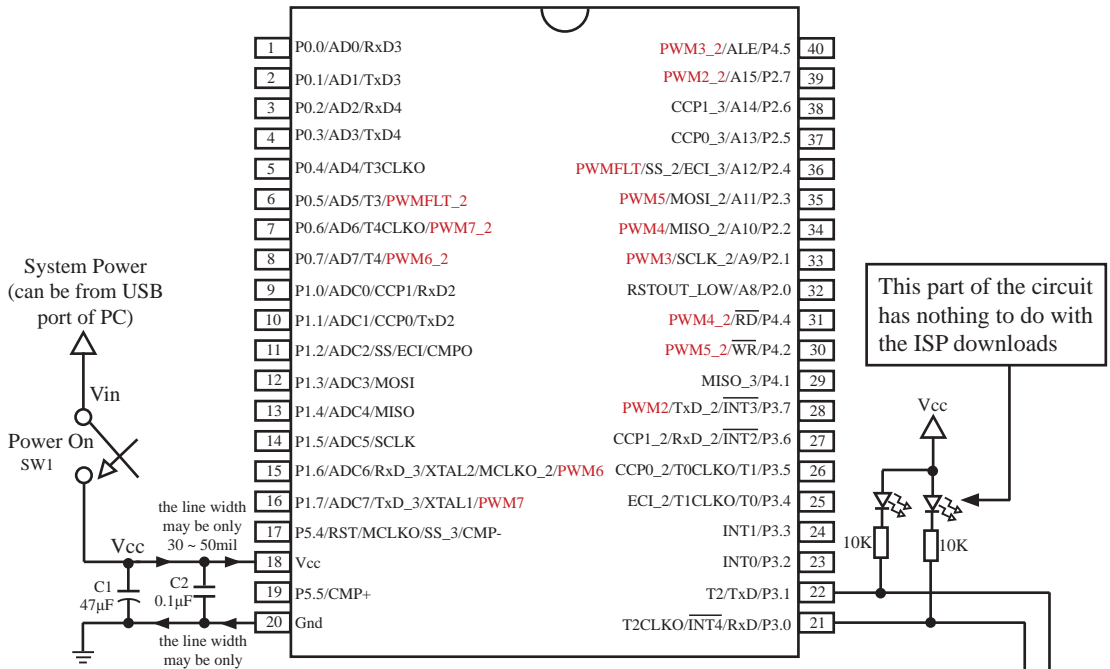
**Conclusion** : STC15W4K32S4 series MCU have: Five 16-bit reloadable Timers/Counters that are Timer/Counter 0, Timer/Counter 1, Timer/Counter 2, Timer/Counter 3 and Timer/Counter 4; 8 channels and 10 bits PWM (can achieve 8 D/A converters or 2 timers or 2 external interrupts again); special power-down wake-up timer; 5 external interrupts INT0/INT1/INT2/INT3/INT4; 4 high-speed asynchronous serial ports ---- UARTs (UART1/UART2/UART3/UART4 can be used simultaneously); a high-speed synchronous serial peripheral interface ---- SPI; 8 channels and 10 bits high-speed A/D converter; a group of Comparator, 2 data pointers ---- DPTR; external data bus and so on.

## 1.5 Naming rules of STC15W4K32S4 series MCU



# 1.6 Application Circuit Diagram for ISP of STC15W4K series

## 1.6.1 Application Circuit Diagram for ISP using RS-232 Converter

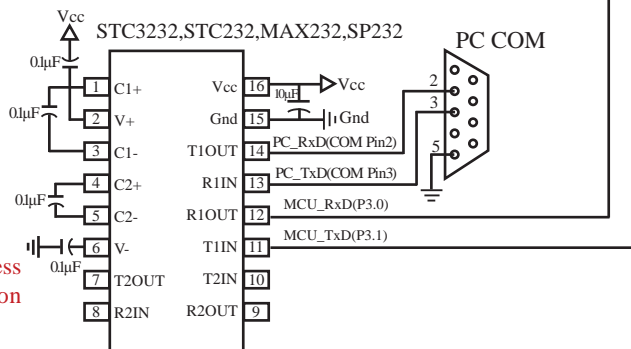


Circuit diagram for ISP of STC MCU, STC RS-232 Converter

Note P0 ports can be multiplexed as Address/Data bus not as A/D Converter. 8 channels of A/D Converter are on P1.

Consequently P0.x/ADx means that P0.x can be used as Address/Data bus, while P1.x/ADCx means P1.x can be used as A/D conversion channel in the pin map.

Please power on the target MCU after press down the button "Download/Program" on STC-ISP.exe when burning code to MCU.



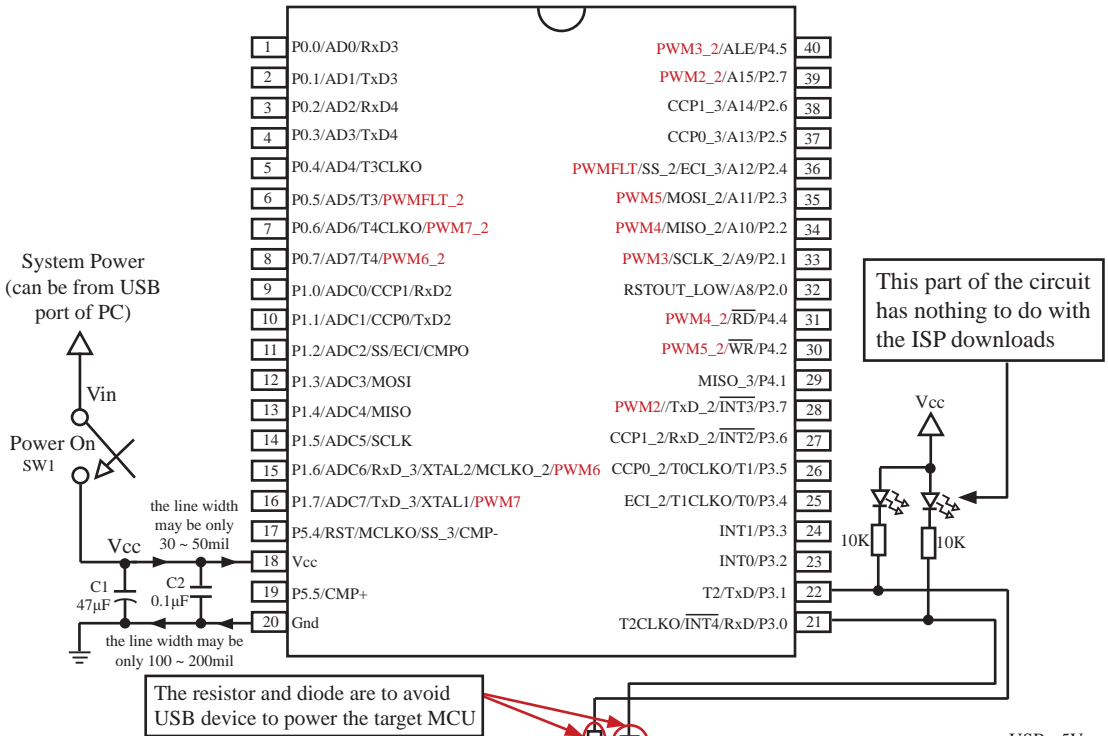
Internal highly reliable Reset, so external reset circuit can be completely removed.

P5.4/RST/MCLKO pin factory defaults to the I/O port, which can be set as RST reset pin(active high) through the STC-ISP programmer.

Internal high-precise R/C clock (±3%), ±1% temperature drift (-40 ~+85 ) while ±0.6% in normal temperature (-20 ~+65 ), so external expensive crystal can be completely removed.

Recommend to add decoupling capacitor C1(47µF) and C2(0.1µF) between Vcc and Gnd that can remove power noise and improve the anti-interference ability.

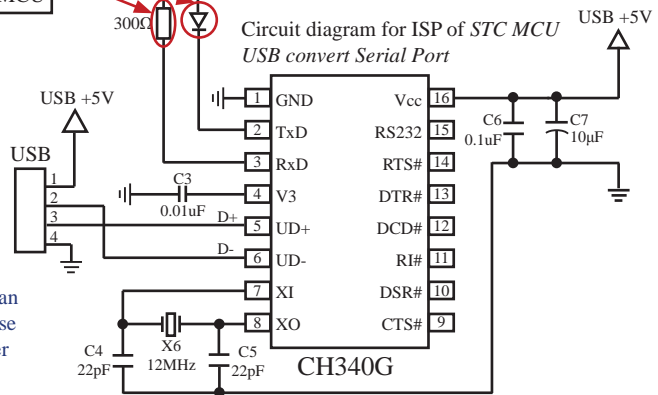
## 1.6.2 Application Circuit Diagram for ISP using USB to convert Serial



Note P0 ports can be multiplexed as Address/ Data bus not as A/D Converter. 8 channels of A/D Converter are on P1.

Consequently P0.x/ADx means that P0.x can be used as Address/Data bus, while P1.x/ADCx means P1.x can be used as A/D conversion channel in the pin map.

Recommend to choose CH340G ( Its pins are not compatible with CH341's, but whose price less than RMB 1.1 yuan is more cheap), also you can choose PL2303(its price is less than RMB 1.0 yuan), refer to [www.wch.cn](http://www.wch.cn) for more detail.



Internal highly reliable Reset, so external reset circuit can be completely removed.

P5.4/RST/MCLKO pin factory defaults to the I/O port, which can be set as RST reset pin(active high) through the STC-ISP programmer.

Internal high-precise R/C clock ( $\pm 3\%$ ),  $\pm 1\%$  temperature drift ( $-40 \sim +85$ ) while  $\pm 0.6\%$  in normal temperature ( $-20 \sim +65$ ), so external expensive crystal can be completely removed.

Recommend to add decoupling capacitor C1(47µF) and C2(0.1µF) between Vcc and Gnd that can remove power noise and improve the anti-interference ability.

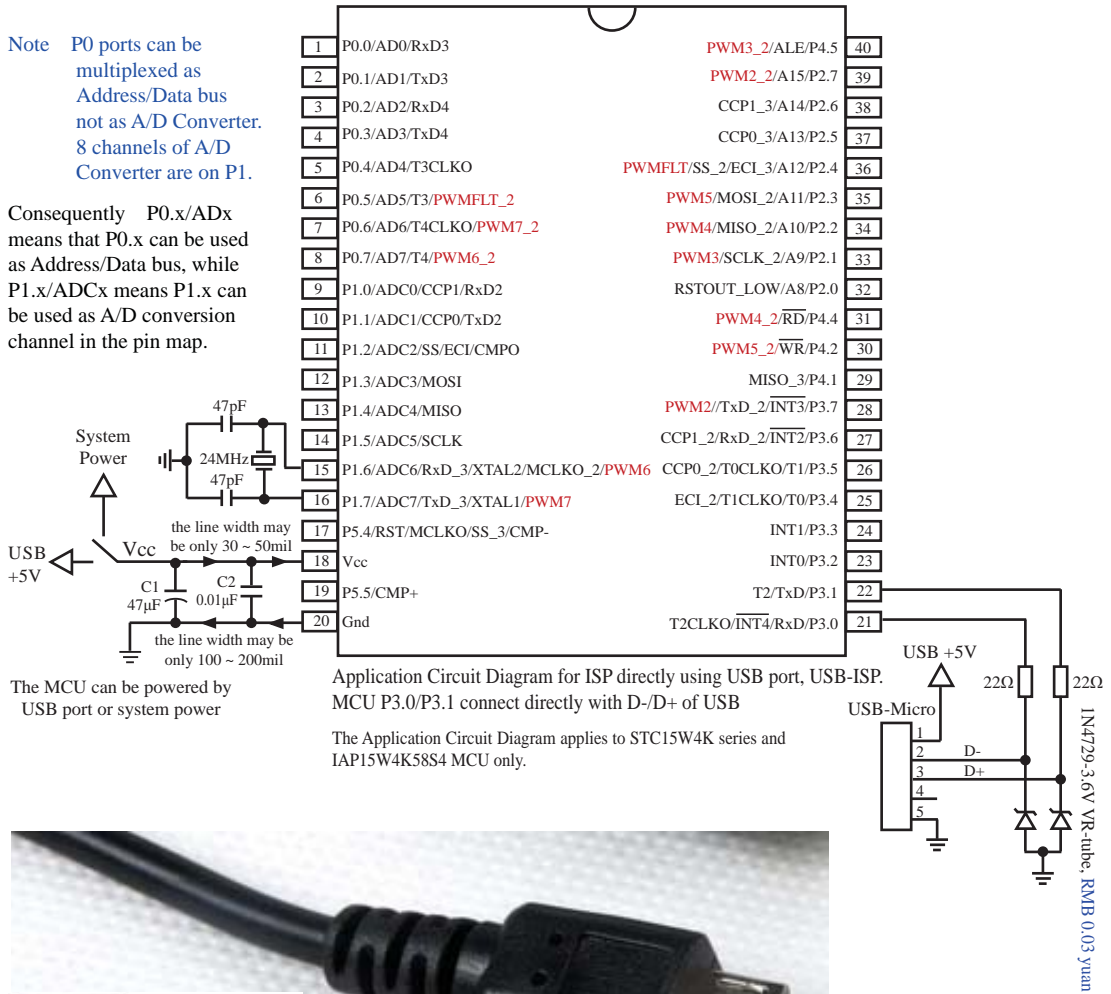


### 1.6.3 Application Circuit Diagram for ISP directly using USB port

—P3.0/P3.1 of STC15W4K series and IAP15W4K58S4 connect directly with D-/D+ of USB

Note P0 ports can be multiplexed as Address/Data bus not as A/D Converter. 8 channels of A/D Converter are on P1.

Consequently P0.x/ADx means that P0.x can be used as Address/Data bus, while P1.x/ADCx means P1.x can be used as A/D conversion channel in the pin map.



USB-Micro

## 1.7 Pin Descriptions of STC15W4K32S4 series MCU

MNEMONIC	Pin Number							DESCRIPTION	
	LQFP64	LQFP48	LQFP44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28		
P0.0/AD0/ RxD3	59	43	40	1	1	29	-	P0.0	common I/O port PORT0[0]
								AD0	Address/Data Bus
								RxD3	Receive Data Port of UART3
P0.1/AD1/ TxD3	60	44	41	2	2	30	-	P0.1	common I/O port PORT0[1]
								AD1	Address/Data Bus
								TxD3	Transit Data Port of UART3
P0.2/AD2/ RxD4	61	45	42	3	3	31	-	P0.2	common I/O port PORT0[2]
								AD2	Address/Data Bus
								RxD4	Receive Data Port of UART4
P0.3/AD3/ TxD4	62	46	43	4	4	32	-	P0.3	common I/O port PORT0[3]
								AD3	Address/Data Bus
								TxD4	Transit Data Port of UART4
P0.4/AD4/ T3CLKO	63	47	44	5	-	-	-	P0.4	common I/O port PORT0[4]
								AD4	Address/Data Bus
								T3CLKO	T3 Clock Output The pin can be configured for T3CLKO by setting T4T3M[0] bit /T3CLKO
P0.5/AD5/T3/ PWMFLT_2	2	2	1	6	-	-	-	P0.5	common I/O port PORT0[5]
								AD5	Address/Data Bus
								T3	External input of Timer/Counter 3
								PWMFLT_2	Control PWM to emergency stop
P0.6/AD6/ T4CLKO/ PWM7_2	3	3	2	7	-	-	-	P0.6	common I/O port PORT0[6]
								AD6	Address/Data Bus
								T4CLKO	T4 Clock Output The pin can be configured for T4CLKO by setting T4T3M[4] bit /T4CLKO
								PWM7_2	The seventh output channel of Pulse Width Modulation. The port mode defaults to input-only(high-impedance) mode after power-on or reset
P0.7/AD7/T4/ PWM6_2	4	4	3	8	-	-	-	P0.7	common I/O port PORT0[7]
								AD7	Address/Data Bus
								T4	External input of Timer/Counter 4
								PWM6_2	The sixth output channel of Pulse Width Modulation. The port mode defaults to input-only(high-impedance) mode after power-on or reset
P1.0/ADC0/ CCP1/RxD2	9	5	4	9	5	1	3	P1.0	common I/O port PORT1[0]
								ADC0	ADC input channel-0
								CCP1	Capture of external signal(measure frequency or be used as external interrupts) high-speed Pulse and Pulse-Width Modulation output channel-1
								RxD2	Receive Data Port of UART2

MNEMONIC	Pin Number							DESCRIPTION	
	LQFP64	LQFP48	LQFP44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28		
P1.1/ADC1/ CCP0/TxD2	10	6	5	10	6	2	4	P1.1	common I/O port PORT1[1]
								ADC1	ADC input channel-1
								CCP0	Capture of external signal(measure frequency or be used as external interrupts) high-speed Pulse and Pulse-Width Modulation output channel-0
								TxD2	Transit Data Port of UART2
P1.2/ADC2/ SS/ECI/ CMPO	12	8	7	11	7	3	5	P1.2	common I/O port PORT1[2]
								ADC2	ADC input channel-2
								SS	Slave selection signal of synchronous serial peripheral interface----SPI
								ECI	External pulse input pin of CCP/ PCA counter
CMPO	The output port of reslut compared by comparator								
P1.3/ADC3/ MOSI	13	9	8	12	8	4	6	P1.3	common I/O port PORT1[3]
								ADC3	ADC input channel-3
								MOSI	Master Output Slave Input of SPI
P1.4/ADC4/ MISO	14	10	9	13	9	5	7	P1.4	common I/O port PORT1[4]
								ADC4	ADC input channel-4
								MISO	Master Iutput Slave Onput of SPI
P1.5/ADC5/ SCLK	15	11	10	14	10	6	8	P1.5	common I/O port PORT1[5]
								ADC5	ADC input channel-5
								SCLK	Clock Signal of synchronous serial peripheral interface----SPI
P1.6/ADC6/ RxD_3/ XTAL2/ MCLKO_2/ PWM6	16	12	11	15	11	7	9	P1.6	common I/O port PORT1[6]
								ADC6	ADC input channel--6
								RxD_3	Receive Data Port of UART1
								MCLKO_2	Master clock output; the output frequency can be MCLK/1, MCLK/2 and MCLK/4. The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator.
								XTAL2	Output from the inverting amplifier of internal clock circuit. This pin should be floated when an external oscillator is used.
								PWM6	The sixth output channel of Pulse Width Modulation. The port mode defaults to input-only(high-impedance) mode after power-on or reset

MNEMONIC	Pin Number							DESCRIPTION	
	LQFP64	LQFP48	LQFP44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28		
P1.7/ADC7/ TxD_3/ XTAL1/ PWM7	17	13	12	16	12	8	10	P1.7	common I/O port PORT1[7]
								ADC7	ADC input channel--7
								TxD_3	Transit Data Port of UART1
								XTAL1	Input to the inverting oscillator amplifier of internal clock circuit. Receives the external oscillator signal when an external oscillator is used.
PWM7	The seventh output channel of Pulse Width Modulation. The port mode defaults to input-only(high-impedance) mode after power-on or reset								
P2.0/A8/ RSTOUT_LOW	45	33	30	32	25	21	23	P2.0	common I/O port PORT2[0]
								A8	The eighth bit of Address bus — A8
								RSTOUT_LOW	the pin output low after power-on and during reset, which can be set to output high by software
P2.1/A9/ SCLK_2/ PWM3	46	34	31	33	26	22	24	P2.1	common I/O port PORT2[1]
								A9	The ninth bit of Address bus — A9
								SCLK_2	Clock Signal of synchronous serial peripheral interface----SPI
								PWM3	The third output channel of Pulse Width Modulation. The port mode defaults to input-only(high-impedance) mode after power-on or reset
P2.2/A10/ MISO_2/ PWM4	47	35	32	34	27	23	25	P2.2	common I/O port PORT2[2]
								A10	The tenth bit of Address bus — A10
								MISO_2	Master Input Slave Output of SPI
								PWM4	The fourth output channel of Pulse Width Modulation. The port mode defaults to input-only(high-impedance) mode after power-on or reset
P2.3/A11/ MOSI_2/ PWM5	48	36	33	35	28	24	26	P2.3	common I/O port PORT2[3]
								A11	The eleventh bit of Address bus —A11
								MOSI_2	Master Output Slave Input of SPI
								PWM5	The fifth output channel of Pulse Width Modulation. The port mode defaults to input-only(high-impedance) mode after power-on or reset
P2.4/A12/ ECL_3/SS_2/ PWMFLT	49	37	34	36	29	25	27	P2.4	common I/O port PORT2[4]
								A12	The twelfth bit of Address bus — A12
								ECL_3	External pulse input pin of CCP/PCA counter
								SS_2	Slave selection signal of synchronous serial peripheral interface----SPI
PWMFLT	Control PWM to emergency stop								
P2.5/A13/ CCP0_3	50	38	35	37	30	26	28	P2.5	common I/O port PORT2[5]
								A13	The thirteenth bit of Address bus — A13
								CCP0_3	Capture of external signal(measure frequency or be used as external interrupts) high-speed Pulse and Pulse-Width Modulation output channel-0

MNEMONIC	Pin Number							DESCRIPTION	
	LQFP64	LQFP48	LQFP44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28		
P2.6/A14/ CCP1_3	51	39	36	38	31	27	1	P2.6	common I/O port PORT2[6]
								A14	The fourteenth bit of Address bus—A14
								CCP1_3	Capture of external signal(measure frequency or be used as external interrupts) high-speed Pulse and Pulse-Width Modulation output channel-1
P2.7/A15/ PWM2_2	52	40	37	39	32	28	2	P2.7	common I/O port PORT2[7]
								A15	The fifteenth bit of Address bus — A15
								PWM2_2	The second output channel of Pulse Width Modulation. The port mode defaults to input-only(high-impedance) mode after power-on or reset
P3.0/RxD/ INT4 /T2CLKO	27	19	18	21	17	13	15	P3.0	common I/O port PORT3[0]
								RxD	Receive Data Port of UART1
								INT4	External interrupt 4, which only can be generated on falling edge. /INT4 supports power-down waking-up
								T2CLKO	T2 Clock Output The pin can be configured for T2CLKO by setting INT_CLKO[2] bit /T2CLKO
P3.1/TxD/T2	28	20	19	22	18	14	16	P3.1	common I/O port PORT3[1]
								TxD	Transit Data Port of UART1
								T2	External input of Timer/Counter 2
P3.2/INT0	29	21	20	23	19	15	17	P3.2	common I/O port PORT3[2]
								INT0	External interrupt 0, which both can be generated on rising and falling edge. INT0 only can generate interrupt on falling edge if IT0 (TCON.0) is set to 1. And, INT0 both can generate interrupt on rising and falling edge if IT0 (TCON.0) is set to 0.
P3.3/INT1	30	22	21	24	20	16	18	P3.3	common I/O port PORT3[3]
								INT1	External interrupt 1, which both can be generated on rising and falling edge. INT1 only can generate interrupt on falling edge if IT1 (TCON.2) is set to 1. And, INT1 both can generate interrupt on rising and falling edge if IT1 (TCON.2) is set to 0. INT1 supports power-down waking-up
P3.4/T0/ T1CLKO/ ECL_2	31	23	22	25	21	17	19	P3.4	common I/O port PORT3[4]
								T0	External input of Timer/Counter 0
								T1CLKO	T1 Clock Output The pin can be configured for T1CLKO by setting INT_CLKO[1] bit /T1CLKO
								ECL_2	External pulse input pin of CCP/PCA counter

MNEMONIC	Pin Number							DESCRIPTION	
	LQFP64	LQFP48	LQFP44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28		
P3.5/T1/ T0CLKO/ CCP0_2	34	26	23	26	22	18	20	P3.5	common I/O port PORT3[5]
								T1	External input of Timer/Counter 1
								T0CLKO	T0 Clock Output The pin can be configured for T0CLKO by setting INT_CLKO[0] bit /T0CLKO
								CCP0_2	Capture of external signal(measure frequency or be used as external interrupts) high-speed Pulse and Pulse-Width Modulation output channel-0
P3.6/ $\overline{\text{INT2}}$ / RxD_2/ CCP1_2	35	27	24	27	23	19	21	P3.6	common I/O port PORT3[6]
								$\overline{\text{INT2}}$	External interrupt 2, which only can be generated on falling edge. /INT2 supports power-down waking-up
								RxD_2	Receive Data Port of UART1
								CCP1_2	Capture of external signal(measure frequency or be used as external interrupts) high-speed Pulse and Pulse-Width Modulation output channel-1
P3.7/ $\overline{\text{INT3}}$ /TxD_2/ PWM2	36	28	25	28	24	20	22	P3.7	common I/O port PORT3[7]
								$\overline{\text{INT3}}$	External interrupt 3, which only can be generated on falling edge. /INT3 supports power-down waking-up
								TxD_2	Transit Data Port of UART1
								PWM2	The second output channel of Pulse Width Modulation. The port mode defaults to input-only(high-impedance) mode after power-on or reset
P4.0/MOSI_3	22	18	17	-	-	-	-	P4.0	common I/O port PORT4[0]
								MISO_3	Master Input Slave Output of SPI
P4.1/MISO_3	41	29	26	29	-	-	-	P4.1	common I/O port PORT4[1]
								MOSI_3	Master Output Slave Input of SPI
P4.2/ $\overline{\text{WR}}$ /PWM5_2	42	30	27	30	-	-	-	P4.2	common I/O port PORT4[2]
								$\overline{\text{WR}}$	Write pulse of external data memory
								PWM5_2	The fifth output channel of Pulse Width Modulation. The port mode defaults to input-only(high-impedance) mode after power-on or reset
P4.3/SCLK_3	43	31	28	-	-	-	-	P4.3	PORT4[3]
								SCLK_3	Clock Signal of synchronous serial peripheral interface----SPI

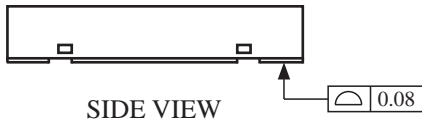
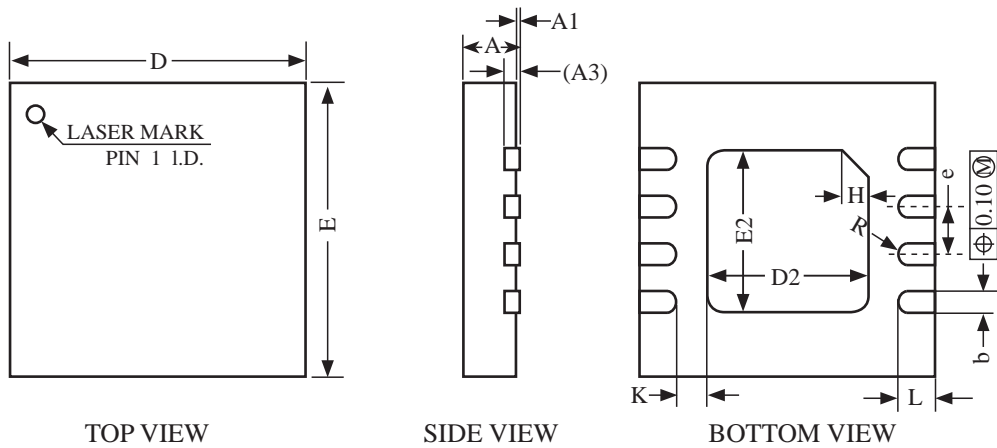
MNEMONIC	Pin Number							DESCRIPTION	
	LQFP64	LQFP48	LQFP44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28		
P4.4/ $\overline{\text{RD}}$ /PWM4_2	44	32	29	31	-	-	-	P4.4	common I/O port PORT4[4]
								$\overline{\text{RD}}$	Read pulse of external data memory
P4.5/ALE/ PWM3_2	57	41	38	40	-	-	-	PWM4_2	The fourth output channel of Pulse Width Modulation. The port mode defaults to input-only(high-impedance) mode after power-on or reset
								P4.5	common I/O port PORT4[5]
P4.6/ RxD2_2	58	42	39	-	-	-	-	ALE	Address Latch Enable. It is used for external data memory cycles (MOVX)
								PWM3_2	The third output channel of Pulse Width Modulation. The port mode defaults to input-only(high-impedance) mode after power-on or reset
P4.7/ TxD2_2	11	7	6	-	-	-	-	P4.6	common I/O port PORT4[6]
								RxD2_2	Receive Data Port of UART2
P5.0/ RxD3_2	32	24	-	-	-	-	-	P4.7	common I/O port PORT4[7]
								TxD2_2	Transit Data Port of UART2
P5.1/ TxD3_2	33	25	-	-	-	-	-	P5.0	common I/O port PORT5[0]
								RxD3_2	Receive Data Port of UART3
P5.2/ RxD4_2	64	48	-	-	-	-	-	P5.1	common I/O port PORT5[1]
								TxD3_2	Transit Data Port of UART3
P5.3/ TxD4_2	1	1	-	-	-	-	-	P5.2	common I/O port PORT5[2]
								RxD4_2	Receive Data Port of UART4
P5.4/RST/ MCLKO/ SS_3/CMP-	18	14	13	17	13	9	11	P5.3	common I/O port PORT5[3]
								TxD4_2	Transit Data Port of UART4
P5.5/CMP+	20	16	15	19	15	11	13	P5.4	common I/O port PORT5[4]
								RST	Reset pin. A high on this pin for at least two machine cycles will reset the device.
P5.5/CMP+	20	16	15	19	15	11	13	MCLKO	Master clock output; the output frequency can be MCLK/1, MCLK/2 and MCLK/4. The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator.
								SS_3	Slave selection signal of synchronous serial peripheral interface---SPI
P5.5/CMP+	20	16	15	19	15	11	13	CMP-	Comparator negative input
								P5.5	common I/O port PORT5[5]
								CMP+	Comparator positive input

MNEMONIC	Pin Number							DESCRIPTION
	LQFP64	LQFP48	LQFP44	PDIP40	SOP32	LQFP32	SOP28 SKDIP28	
P6.0	5							common I/O port PORT6[0]
P6.1	6							common I/O port PORT6[1]
P6.2	7							common I/O port PORT6[2]
P6.3	8							common I/O port PORT6[3]
P6.4	23							common I/O port PORT6[4]
P6.5	24							common I/O port PORT6[5]
P6.6	25							common I/O port PORT6[6]
P6.7	26							common I/O port PORT6[7]
P7.0	37							common I/O port PORT7[0]
P7.1	38							common I/O port PORT7[1]
P7.2	39							common I/O port PORT7[2]
P7.3	40							common I/O port PORT7[3]
P7.4	53							common I/O port PORT7[4]
P7.5	54							common I/O port PORT7[5]
P7.6	55							common I/O port PORT7[6]
P7.7	56							common I/O port PORT7[7]
Vcc	19	15	14	18	14	10	12	The positive pole of power
Gnd	21	17	16	20	16	12	14	The negative pole of power, Gound



## 1.8 Package Dimension Drawings of STC15 series MCU

### 1.8.1 Dimension Drawings of DFN8



COMMON DIMENSIONS			
UNITS OF MEASURE = mm (MILLIMETER)			
SYMBOL	MIN.	NOM.	MAX.
A	0.70	0.75	0.80
A1	0.00	0.02	0.05
A3	0.20REF		
b	0.25	0.30	0.35
D	3.90	4.00	4.10
E	3.90	4.00	4.10
D2	2.10	2.20	2.30
E2	2.10	2.20	2.30
e	0.55	0.65	0.75
H	0.35REF		
K	0.35REF		
L	0.45	0.55	0.65
R	0.13	-	-

Note:

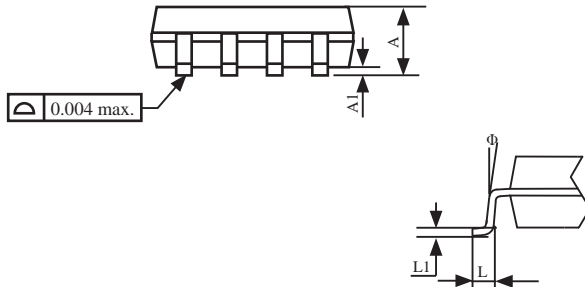
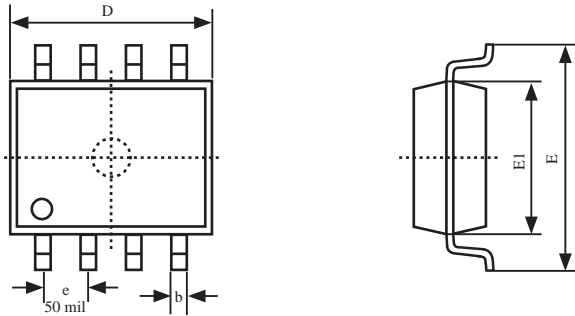
All dimensions do not include mold flash or protrusions

## 1.8.2 Dimension Drawings of SOP8

### Dimension Drawings of SOP8

8-PIN SMALL OUTLINE PACKAGE (SOP8)

Dimensions in Inches



COMMON DIMENSIONS			
(UNITS OF MEASURE = INCH)			
SYMBOL	MIN.	NOM.	MAX.
A	0.053	-	0.069
A1	0.004	-	0.010
b	-	0.016	-
D	0.189	-	0.196
E	0.228	-	0.244
E1	0.150	-	0.157
e	0.050		
L	0.016	-	0.050
L1	0.008		
Φ	0°	-	8°

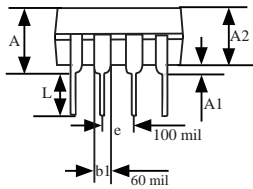
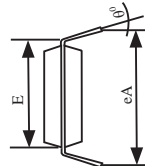
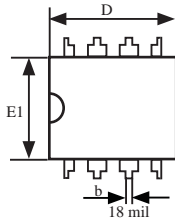
UNIT: INCH, 1 inch = 1000 mil

## 1.8.3 Dimension Drawings of DIP8

### Dimension Drawings of DIP8

#### 8-Pin Plastic Dual Inline Package (DIP8)

Dimensions in Inches



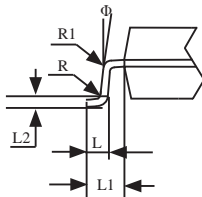
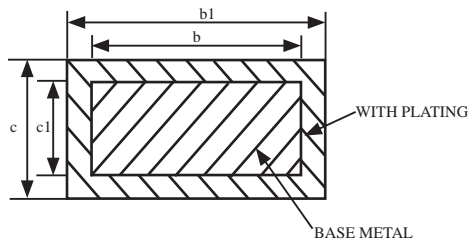
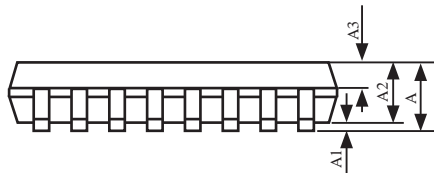
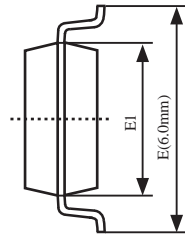
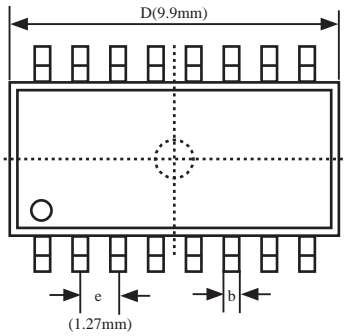
COMMON DIMENSIONS			
(UNITS OF MEASURE = INCH)			
SYMBOL	MIN.	NOM.	MAX.
A	-	-	0.210
A1	0.015	-	-
A2	0.125	0.130	0.135
b	-	0.018	-
b1	-	0.060	-
D	0.355	0.365	0.400
E	-	0.300	-
E1	0.245	0.250	0.255
e	-	0.100	-
L	0.115	0.130	0.150
θ°	0	7	15
eA	0.335	0.355	0.375

UNIT: INCH, 1 inch = 1000 mil

## 1.8.4 Dimension Drawings of SOP16

### Dimension Drawings of SOP16

16-PIN SMALL OUTLINE PACKAGE (SOP16)



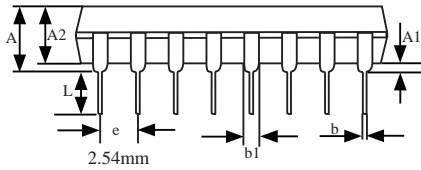
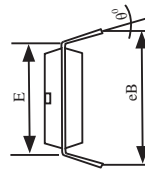
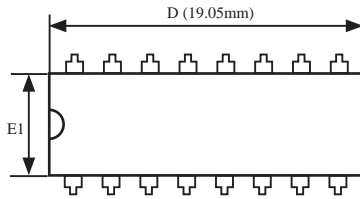
COMMON DIMENSIONS			
(UNITS OF MEASURE = MILLMETER)			
SYMBOL	MIN	NOM	MAX
A	1.35	1.60	1.75
A1	0.10	0.15	0.25
A2	1.25	1.45	1.65
A3	0.55	0.65	0.75
b1	0.36	-	0.49
b	0.35	0.40	0.45
c	0.16	-	0.25
c1	0.15	0.20	0.25
D	9.80	9.90	10.00
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e	1.27		
L	0.45	0.60	0.80
L1	1.04		
L2	0.25		
R	0.07	-	-
R1	0.07	-	-
Φ	6 <sup>0</sup>	8 <sup>0</sup>	10 <sup>0</sup>

## 1.8.5 Dimension Drawings of DIP16

### Dimension Drawings of DIP16

16-Pin Plastic Dual Inline Package (DIP16)

Dimensions in Inches and Millimeters



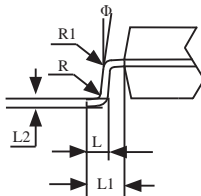
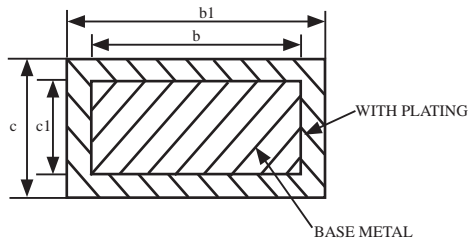
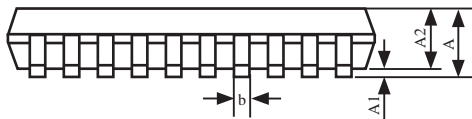
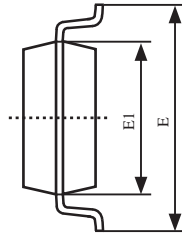
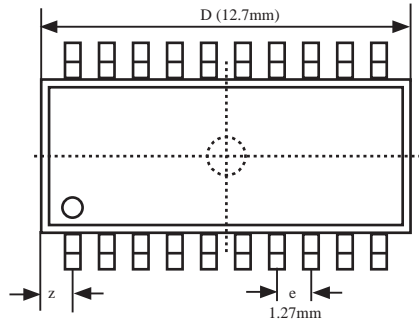
COMMON DIMENSIONS			
(UNITS OF MEASURE = MILLIMETER)			
SYMBOL	MIN	NOM	MAX
A	-	-	4.80
A1	0.50	-	-
A2	3.10	3.30	3.50
b	0.38	-	0.55
b1	0.38	0.46	0.51
D	18.95	19.05	19.15
E	7.62	7.87	8.25
E1	6.25	6.35	6.45
e	2.54		
eB	7.62	8.80	10.90
L	2.92	3.30	3.81
$\theta^{os}$	0	7	15

## 1.8.6 Dimension Drawings of SOP20

### Dimension Drawings of SOP20

#### 20-Pin Small Outline Package (SOP20)

Dimensions in Inches and (Millimeters)

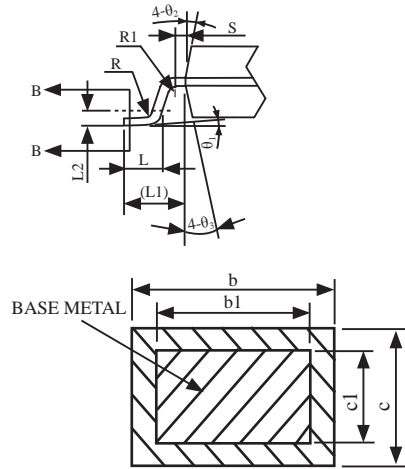
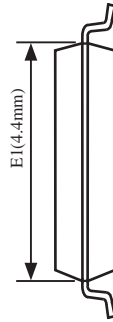
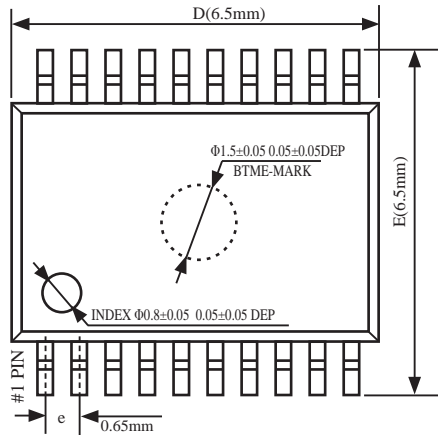


COMMON DIMENSIONS			
(UNITS OF MEASURE = MILLIMETER/ mm)			
SYMBOL	MIN.	NOM.	MAX.
A	2.465	2.515	2.565
A1	0.100	0.150	0.200
A2	2.100	2.300	2.500
b1	0.366	0.426	0.486
b	0.356	0.406	0.456
c	0.234	-	0.274
c1	-	0.254	-
D	12.500	12.700	12.900
E	10.206	10.306	10.406
E1	7.450	7.500	7.550
e	1.27		
L	0.800	0.864	0.900
L1	1.303	1.403	1.503
L2	-	0.274	-
R	-	0.300	-
R1	-	0.200	-
$\Phi$	0°	-	10°
z	-	0.660	-

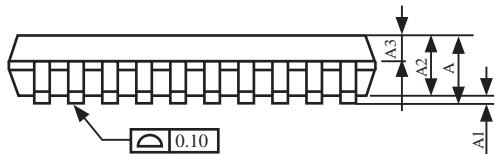
## 1.8.7 Dimension Drawings of TSSOP20

### 20-Pin Plastic Thin Shrink Small Outline Package (TSSOP20)

Dimensions in Millimeters



SECTION B-B



COMMON DIMENSIONS			
(UNITS OF MEASURE = MILLIMETER)			
SYMBOL	MIN	NOM	MAX
A	-	-	1.2
A1	0.05	-	0.15
A2	0.90	1.00	1.05
A3	0.34	0.44	0.54
b	0.20	-	0.28
b1	0.20	-	0.24
c	0.10	-	0.19
c1	0.10	0.13	0.15
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		
R	0.09	-	-
R1	0.09	-	-
S	0.20	-	-
$\theta_1$	0°	-	8°
$\theta_2$	10°	12°	14°
$\theta_3$	10°	12°	14°

NOTES:

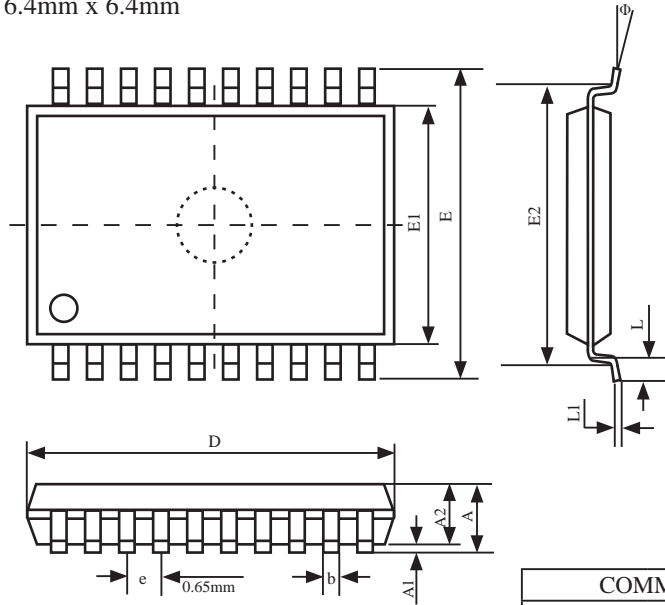
ALL DIMENSIONS REFER TO JEDEC STANDARD MO-153 AC  
DO NOT INCLUDE MOLD FLASH OR PROTRUSIONS.

## 1.8.8 Dimension Drawings of LSSOP20

### Dimension Drawings of LSSOP20

#### 20-Pin Plastic Shrink Small Outline Package (LSSOP20)

LSSOP-20, 6.4mm x 6.4mm



COMMON DIMENSIONS			
(UNITS OF MEASURE = MILLMETER)			
SYMBOL	MIN	NOM	MAX
A	-	-	1.85
A1	0.05	-	-
A2	1.40	1.50	1.60
b	0.17	0.22	0.32
D	6.40	6.50	6.60
E	6.20	6.40	6.60
E1	4.30	4.40	4.50
E2	-	5.72	-
e	0.57	0.65	0.73
L	0.30	0.50	0.70
L1	0.1	0.15	0.25
Φ	0°	-	8°

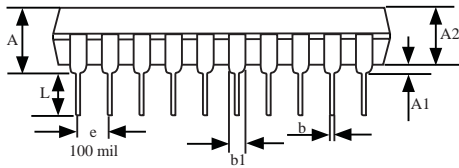
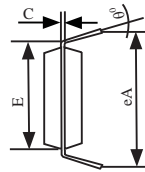
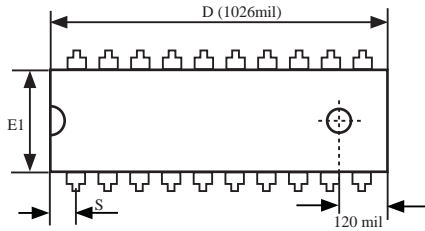


## 1.8.9 Dimension Drawings of DIP20

### Dimension Drawings of DIP20

#### 20-Pin Plastic Dual Inline Package (DIP20)

Dimensions in Inches



COMMON DIMENSIONS			
(UNITS OF MEASURE = INCH)			
SYMBOL	MIN.	NOM.	MAX.
A	-	-	0.175
A1	0.015	-	-
A2	0.125	0.13	0.135
b	0.016	0.018	0.020
b1	0.058	0.060	0.064
C	0.008	0.010	0.11
D	1.012	1.026	1.040
E	0.290	0.300	0.310
E1	0.245	0.250	0.255
e	0.090	0.100	0.110
L	0.120	0.130	0.140
$\theta^0$	0	-	15
eA	0.355	0.355	0.375
S	-	-	0.075

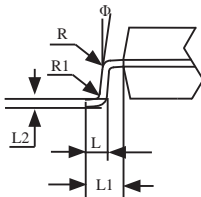
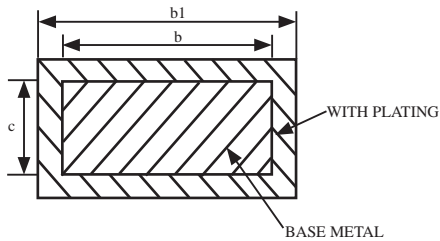
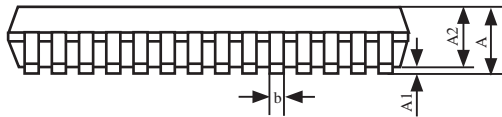
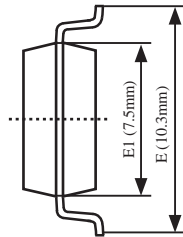
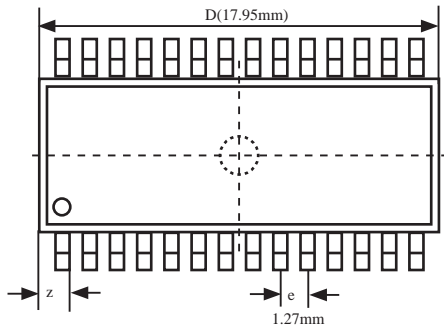
UNIT: INCH, 1 inch = 1000 mil

## 1.8.10 Dimension Drawings of SOP28

### Dimension Drawings of SOP28

#### 28-Pin Small Outline Package (SOP28)

Dimensions in Millimeters

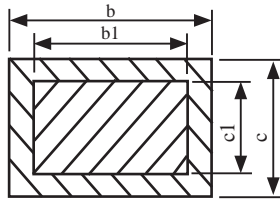
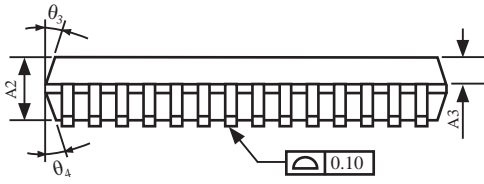
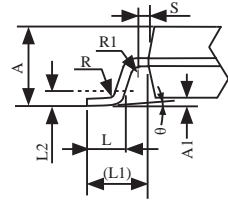
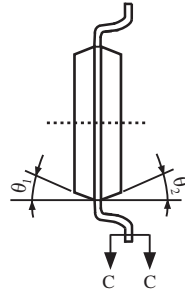
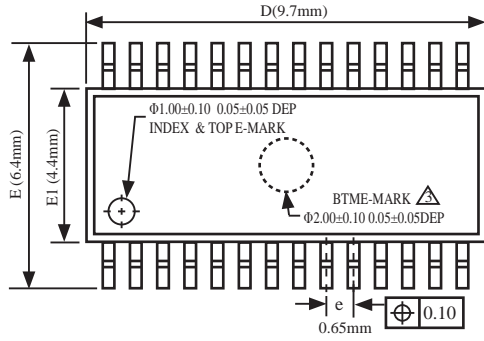


COMMON DIMENSIONS			
(UNITS OF MEASURE = MILLIMETER / mm)			
SYMBOL	MIN.	NOM.	MAX.
A	2.465	2.515	2.565
A1	0.100	0.150	0.200
A2	2.100	2.300	2.500
b	0.356	0.406	0.456
b1	0.366	0.426	0.486
c	-	0.254	-
D	17.750	17.950	18.150
E	10.100	10.300	10.500
E1	7.424	7.500	7.624
e	1.27		
L	0.764	0.864	0.964
L1	1.303	1.403	1.503
L2	-	0.274	-
R	-	0.200	-
R1	-	0.300	-
$\Phi$	0°	-	10°
z	-	0.745	-

## 1.8.11 Dimension Drawings of TSSOP28

### 28-Pin Plastic Thin Shrink Small Outline Package (TSSOP28)

Dimensions in Millimeters



SECTION C-C  
NORMAL PLATING

COMMON DIMENSIONS			
(UNITS OF MEASURE = MILLIMETER / mm)			
SYMBOL	MIN.	NOM.	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	-	0.29
b1	0.19	0.22	0.25
c	0.13	-	0.18
c1	0.12	0.13	0.14
D	9.60	9.70	9.80
E	6.20	6.40	6.60
E1	4.30	4.40	4.50
e	0.55	0.65	0.75
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R	0.09	-	-
R1	0.09	-	-
S	0.20	-	-
$\theta$	0°	-	8°
$\theta_1$	10°	12°	14°
$\theta_2$	10°	12°	14°
$\theta_3$	10°	12°	14°
$\theta_4$	10°	12°	14°

NOTES:

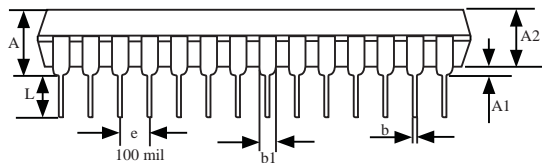
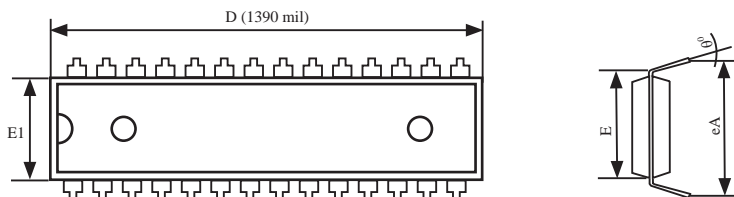
ALL DIMENSIONS REFER TO JEDEC STANDARD MO-153 AE  
DO NOT INCLUDE MOLD FLASH OR PROTRUSIONS.

## 1.8.12 Dimension Drawings of SKDIP28

### Dimension Drawings of SKDIP28

28-Pin Plastic Dual-In-line Package (SKDIP28)

Dimensions in Inches

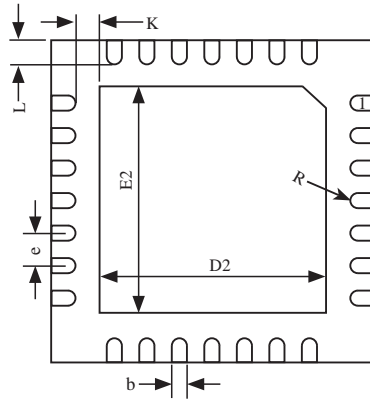
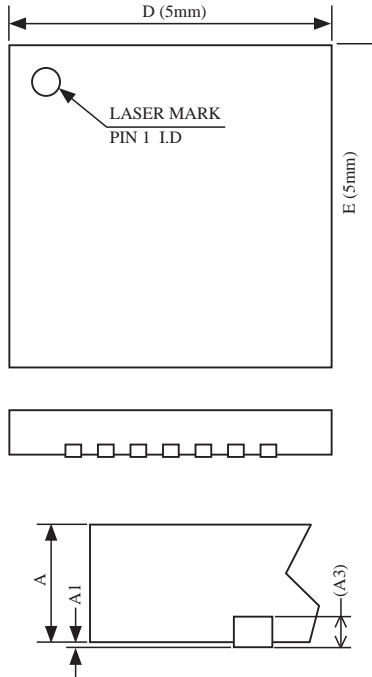


COMMON DIMENSIONS			
(UNITS OF MEASURE = INCH)			
SYMBOL	MIN.	NOM.	MAX.
A	-	-	0.210
A1	0.015	-	-
A2	0.125	0.13	0.135
b	-	0.018	-
b1	-	0.060	-
D	1.385	1.390	1.40
E	-	0.310	-
E1	0.283	0.288	0.293
e	-	0.100	-
L	0.115	0.130	0.150
$\theta^\circ$	0	7	15
eA	0.330	0.350	0.370

UNIT: INCH, 1 inch = 1000 mil

## 1.8.13 Dimension Drawings of QFN28

### QFN28 OUTLINE PACKAGE



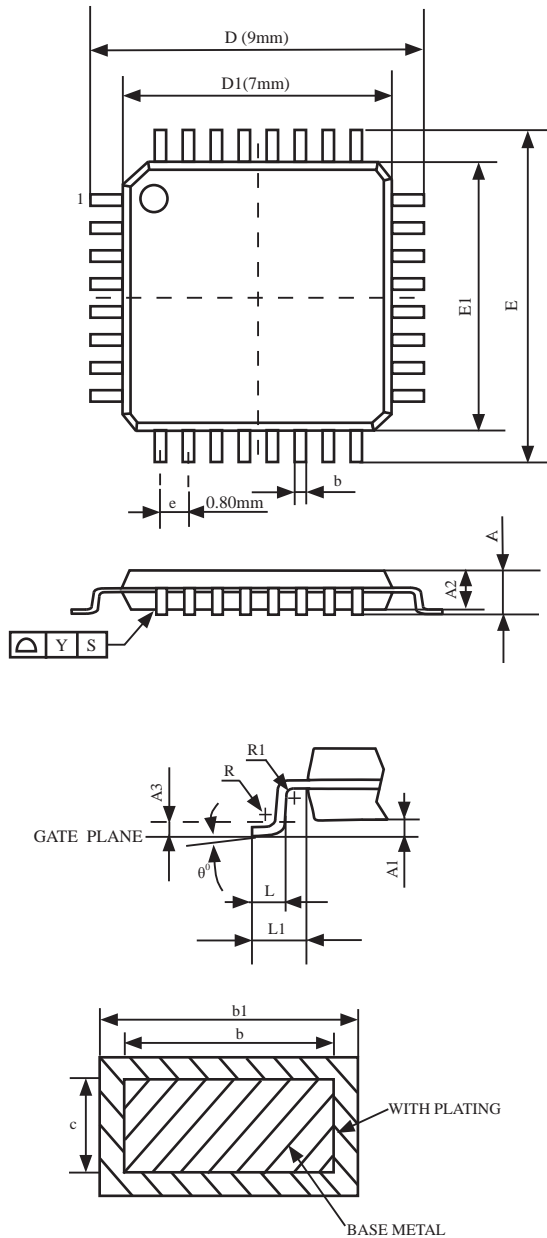
COMMON DIMENSIONS			
(UNITS OF MEASURE = MILLIMETER /mm)			
SYMBOL	MIN.	NOM.	MAX.
A	0.70	0.75	0.80
A1	0	0.02	0.05
A3	0.20REF		
b	0.20	0.25	0.30
D	4.90	5.00	5.10
E	4.90	5.00	5.10
D2	3.35	3.50	3.65
E2	3.35	3.50	3.65
e	0.40	0.50	0.60
K	0.20	-	-
L	0.30	0.40	0.50
R	0.09	-	-

#### NOTES:

ALL DIMENSIONS REFER TO JEDEC STANDARD MO-220 WHHD-3

## 1.8.14 Dimension Drawings of LQFP32

### LQFP32 OUTLINE PACKAGE



VARIATIONS (ALL DIMENSIONS SHOWN IN MM)

SYMBOLS	MIN.	NOM.	MAX.
A	1.45	1.55	1.65
A1	0.01	-	0.21
A2	1.35	1.40	1.45
A3	-	0.254	-
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	0.80		
b	0.3	0.35	0.4
b1	0.31	0.37	0.43
c	-	0.127	-
L	0.43	-	0.71
L1	0.90	1.00	1.10
R	0.1	-	0.25
R1	0.1	-	-
$\theta^0$	$0^0$	-	$10^0$

NOTES:

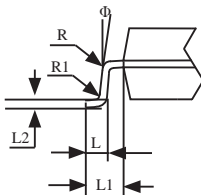
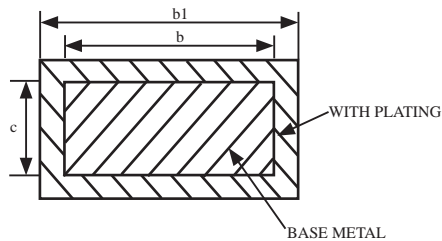
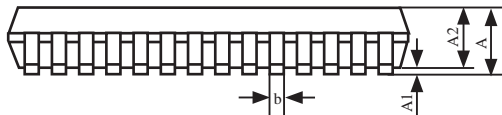
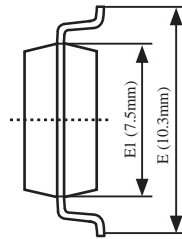
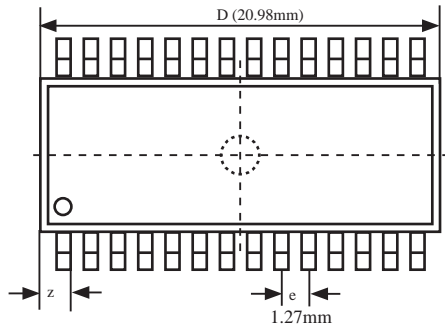
- All dimensions are in mm
- Dim D1 AND E1 does not include plastic flash.  
Flash: Plastic residual around body edge after de junk/singulation
- Dim b does not include dambar protrusion/ intrusion.
- Plating thickness 0.05~0.015 mm.

## 1.8.15 Dimension Drawings of SOP32

Dimension Drawings of SOP32(SOP32 is not produced now, LQFP-32 is recommended)

32-Pin Small Outline Package (SOP32)

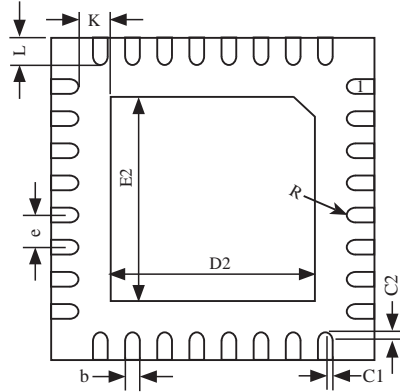
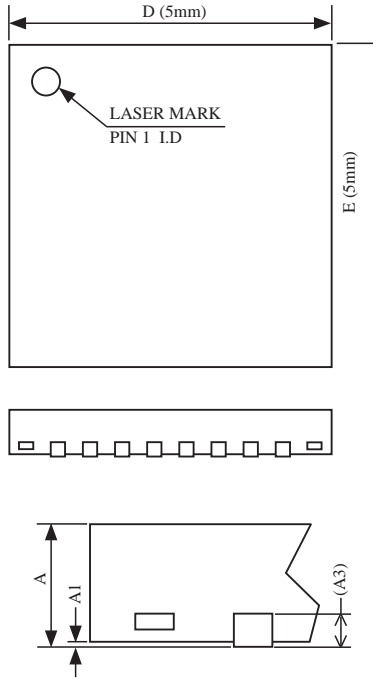
Dimensions in Millimeters



COMMON DIMENSIONS			
(UNITS OF MEASURE = MILLIMETER /mm)			
SYMBOL	MIN	NOM	MAX
A	2.465	2.515	2.565
A1	0.100	0.150	0.200
A2	2.100	2.300	2.500
b	0.356	0.406	0.456
b1	0.366	0.426	0.486
c	-	0.254	-
D	20.88	20.98	21.08
E	10.100	10.300	10.500
E1	7.424	7.500	7.624
e	1.27		
L	0.700	0.800	0.900
L1	1.303	1.403	1.503
L2	-	0.274	-
R	-	0.200	-
R1	-	0.300	-
$\Phi$	0°	-	10°
z	-	0.745	-

## 1.8.16 Dimension Drawings of QFN32

### QFN32 OUTLINE PACKAGE



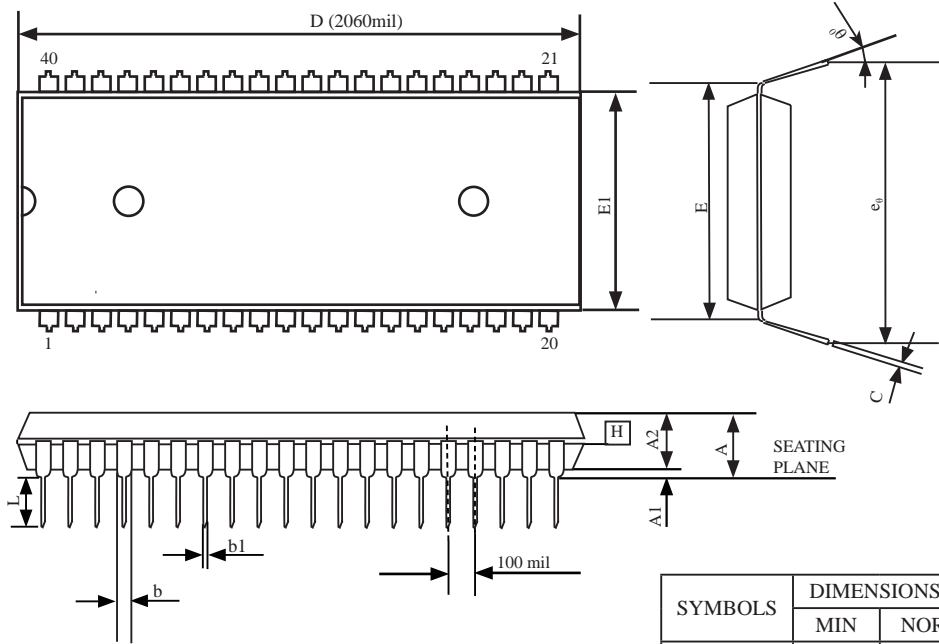
COMMON DIMENSIONS			
(UNITS OF MEASURE = MILLIMETER /mm)			
SYMBOL	MIN.	NOM.	MAX.
A	0.70	0.75	0.80
A1	0	0.02	0.05
A3	0.20REF		
b	0.18	0.25	0.30
D	4.90	5.00	5.10
E	4.90	5.00	5.10
D2	3.10	3.20	3.30
E2	3.10	3.20	3.30
e	0.40	0.50	0.60
K	0.20	-	-
L	0.35	0.40	0.45
R	0.09	-	-
C1	-	0.08	-
C2	-	0.08	-

NOTES:  
ALL DIMENSIONS REFER TO JEDEC STANDARD  
MO-220 WHHD-4



## 1.8.17 Dimension Drawings of PDIP40

### PDIP40 OUTLINE PACKAGE

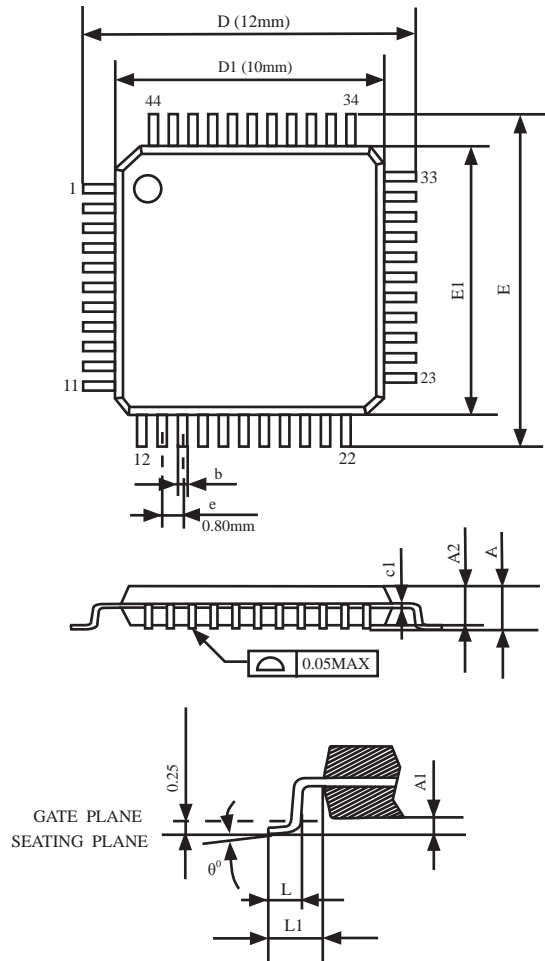


SYMBOLS	DIMENSIONS IN INCH		
	MIN	NOR	MAX
A	-	-	0.190
A1	0.015	-	0.020
A2	0.15	0.155	0.160
C	0.008	-	0.015
D	2.025	2.060	2.070
E	0.600 BSC		
E1	0.540	0.545	0.550
L	0.120	0.130	0.140
b1	0.015	-	0.021
b	0.045	-	0.067
$e_b$	0.630	0.650	0.690
0	0	7	15

UNIT: INCH 1 inch = 1000mil

## 1.8.18 Dimension Drawings of LQFP44

### LQFP-44 OUTLINE PACKAGE



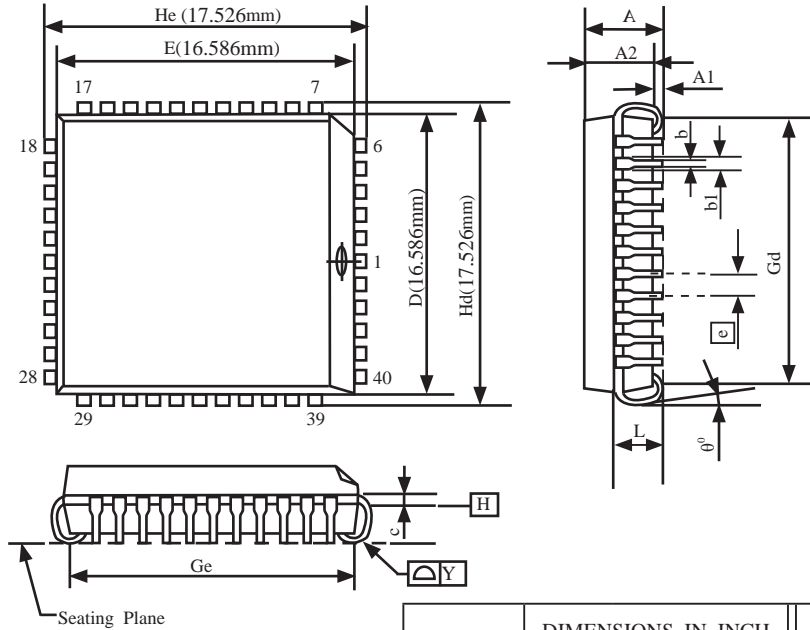
VARIATIONS (ALL DIMENSIONS SHOWN IN MM)

SYMBOLS	MIN.	NOM.	MAX.
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
c1	0.09	-	0.16
D	12.00		
D1	10.00		
E	12.00		
E1	10.00		
e	0.80		
$\Delta$ b(w/o plating)	0.25	0.30	0.35
L	0.45	0.60	0.75
L1	1.00REF		
$\theta^0$	$0^0$	$3.5^0$	$7^0$

## 1.8.19 Dimension Drawings of PLCC44

(PLCC44 is not produced now in STC15 series, LQFP44 is recommended)

### PLCC44 OUTLINE PACKAGE



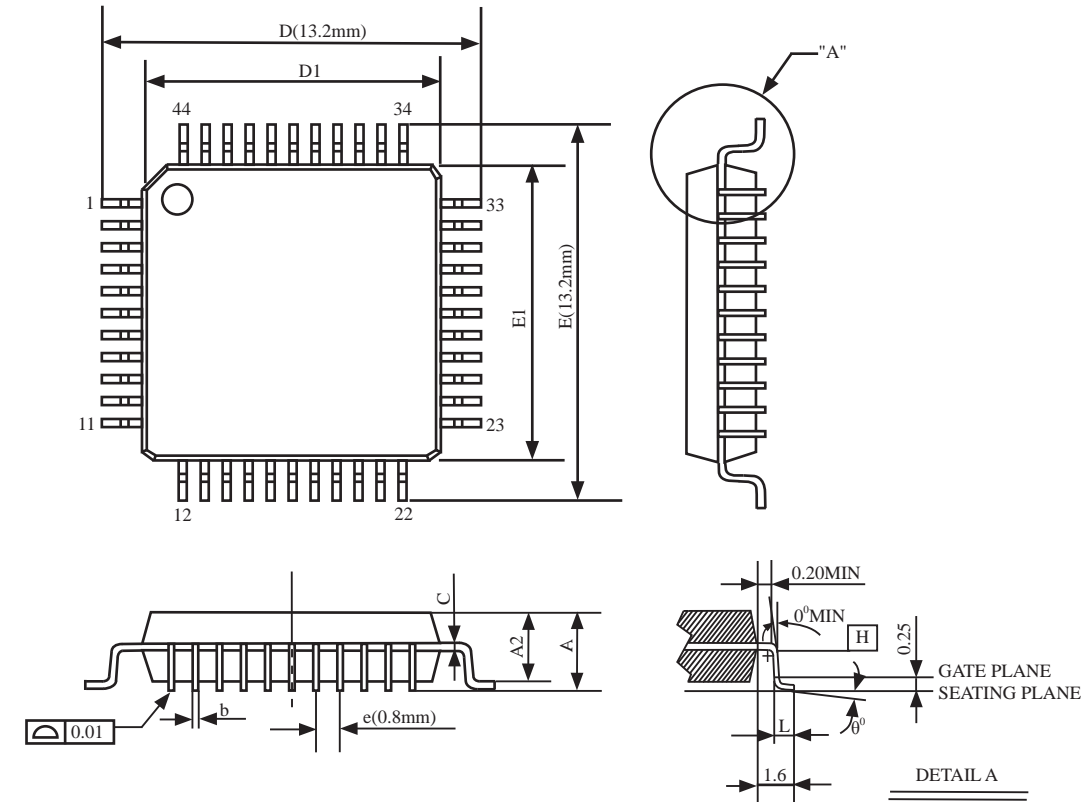
SYMBOLS	DIMENSIONS IN INCH			DIMENSIONS IN MILLIMETERS		
	MIN	NOM	MAX	MIN	NOM	MAX
A	0.165	-	0.180	4.191	-	4.572
A1	0.020	-	-	0.508	-	-
A2	0.147	-	0.158	3.734	-	4.013
b1	0.026	0.028	0.032	0.660	0.711	0.813
b	0.013	0.017	0.021	0.330	0.432	0.533
c	0.007	0.010	0.0013	0.178	0.254	0.330
D	0.650	0.653	0.656	16.510	16.586	16.662
E	0.650	0.653	0.656	16.510	16.586	16.662
<b>[e]</b>	0.050BSC			1.270BSC		
Gd	0.590	0.610	0.630	14.986	15.494	16.002
Ge	0.590	0.610	0.630	14.986	15.494	16.002
Hd	0.685	0.690	0.695	17.399	17.526	17.653
He	0.685	0.690	0.695	17.399	17.526	17.653
L	0.100	-	0.112	2.540	-	2.845
Y	-	-	0.004	-	-	0.102

1 inch = 1000 mil

## 1.8.20 Dimension Drawings of PQFP44

(PQFP44 is not produced now in STC15 series, LQFP44 is recommended)

### PQFP44 OUTLINE PACKAGE



SYMBOLS	MIN.	NOM	MAX.
A	-	-	2.70
A1	0.25	-	0.50
A2	1.80	2.00	2.20
b(w/o plating)	0.25	0.30	0.35
D	13.00	13.20	13.40
D1	9.9	10.00	10.10
E	13.00	13.20	13.40
E1	9.9	10.00	10.10
L	0.73	0.88	0.93
e	0.80 BSC.		
$\theta^\circ$	0	-	7
C	0.1	0.15	0.2

UNIT:mm

#### NOTES:

1.JEDEC OUTLINE:M0-108 AA-1

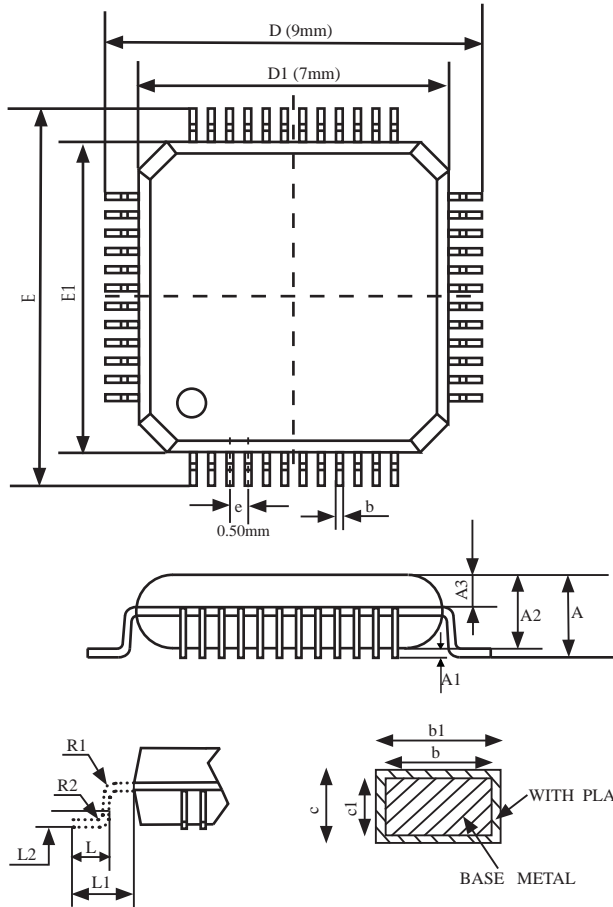
2.DATUM PLANE **[H]** IS LOCATED AT THE BOTTOM OF THE MOLD PARTING LINE COINCIDENT WITH WHERE THE LAED EXITS THE BODY.

3.DIMENSIONS D1 AND E1 D0 NOT INCLUDE MOLD PROTRUSION. ALLOWABLE PROTRUSION IS 0.25mm PER SIDE. DIMENSIONS D1 AND E1 D0 INCLUDE MOLD MISMATCH AND ARE DETERMINED AT DATUM PLANE **[H]**.

4.DIMENSION b DOES NOT INCLUDE DAMBAR PROTRUSION.

## 1.8.21 Dimension Drawings of LQFP48

### LQFP48 OUTLINE PACKAGE

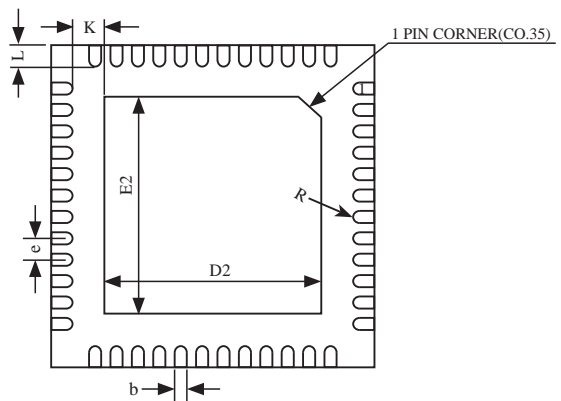
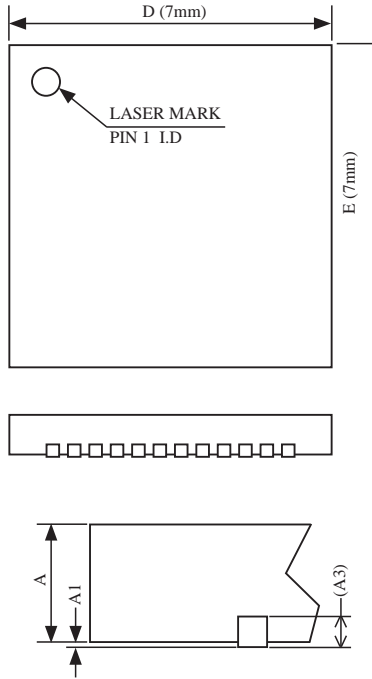


SYMBOL	MIN	NOM	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.18	-	0.27
b1	0.17	0.20	0.23
c	0.13	-	0.18
c1	0.12	0.127	0.134
D	8.80	9.00	9.20
D1	6.90	7.00	7.10
E	8.80	9.00	9.20
E1	6.90	7.00	7.10
e	0.50		
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-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

VARIATIONS (ALL DIMENSIONS SHOWN IN MM)

## 1.8.22 Dimension Drawings of QFN48

### QFN48 OUTLINE PACKAGE

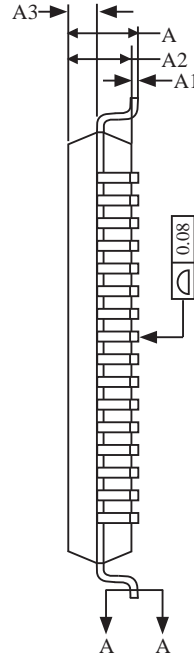
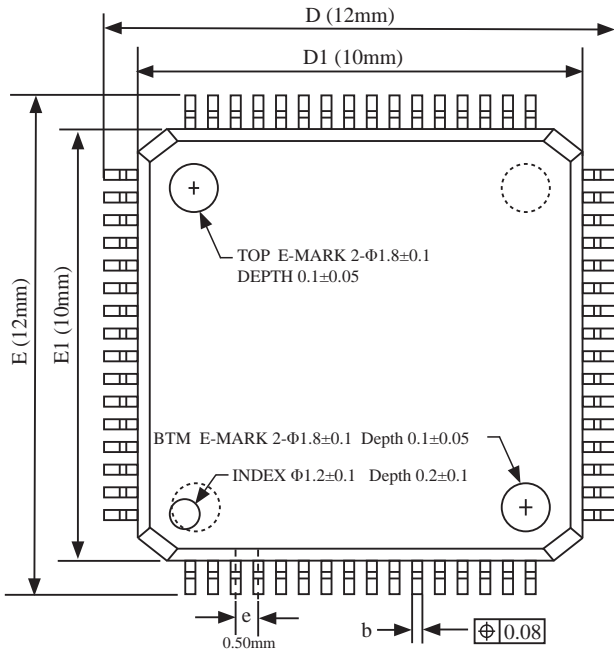


COMMON DIMENSIONS			
(UNITS OF MEASURE = MILLMETER /mm)			
SYMBOL	MIN.	NOM.	MAX.
A	0.70	0.75	0.80
A1	0	0.02	0.05
A3	0.20REF		
b	0.15	0.20	0.25
D	6.90	7.00	7.10
E	6.90	7.00	7.10
D2	3.95	4.05	4.15
E2	3.95	4.05	4.15
e	0.45	0.50	0.55
K	0.20	-	-
L	0.35	0.40	0.45
R	0.09	-	-

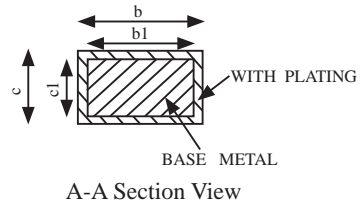
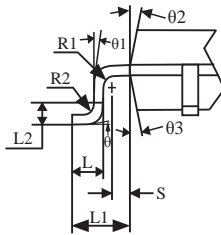
NOTES:  
ALL DIMENSIONS REFER TO JEDEC STANDARD  
MO-220 WJJE.

## 1.8.23 Dimension Drawings of LQFP64S

### LQFP64 SMALL OUTLINE PACKAGE (LQFP64S)



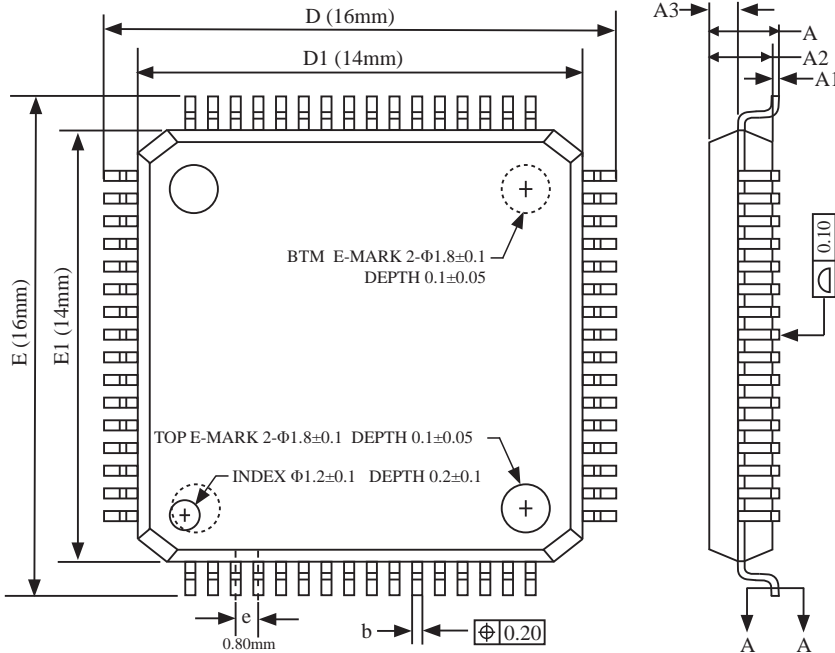
COMMON DIMENSIONS			
(UNITS OF MEASURE = MILLIMETER / mm)			
SYMBOL	MIN	NOM	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.18	-	0.27
b1	0.17	0.20	0.23
c	0.13	-	0.18
c1	0.12	0.127	0.134
D	11.80	12.00	12.20
D1	9.90	10.00	10.10
E	11.80	12.00	12.20
E1	9.90	10.00	10.10
e	0.50BSC		
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-
θ	0°	3.5°	7°
θ1	0°	-	-
θ2	11°	12°	13°
θ3	11°	12°	13°



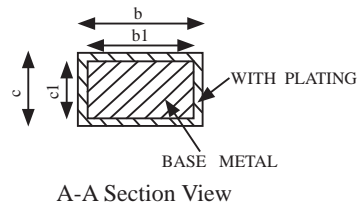
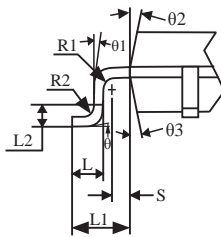
NOTES:  
 ALL DIMENSIONS MEET JEDEC STANDARD  
 MS-026 BEB DO NOT INCLUDE MOLD  
 FLASH OR PROTRUSIONS.

## 1.8.24 Dimension Drawings of LQFP64L

### LQFP64 LARGE OUTLINE PACKAGE (LQFP64L)



COMMON DIMENSIONS			
(UNITS OF MEASURE = MILLIMETER / mm)			
SYMBOL	MIN	NOM	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.31	-	0.44
b1	0.30	0.35	0.40
c	0.13	-	0.18
c1	0.12	0.127	0.134
D	15.80	16.00	16.20
D1	13.90	14.00	14.10
E	15.8	16.00	16.20
E1	13.90	14.00	14.10
e	0.70	0.80	0.90
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-
θ	0°	3.5°	7°
θ1	0°	-	-
θ2	11°	12°	13°
θ3	11°	12°	13°

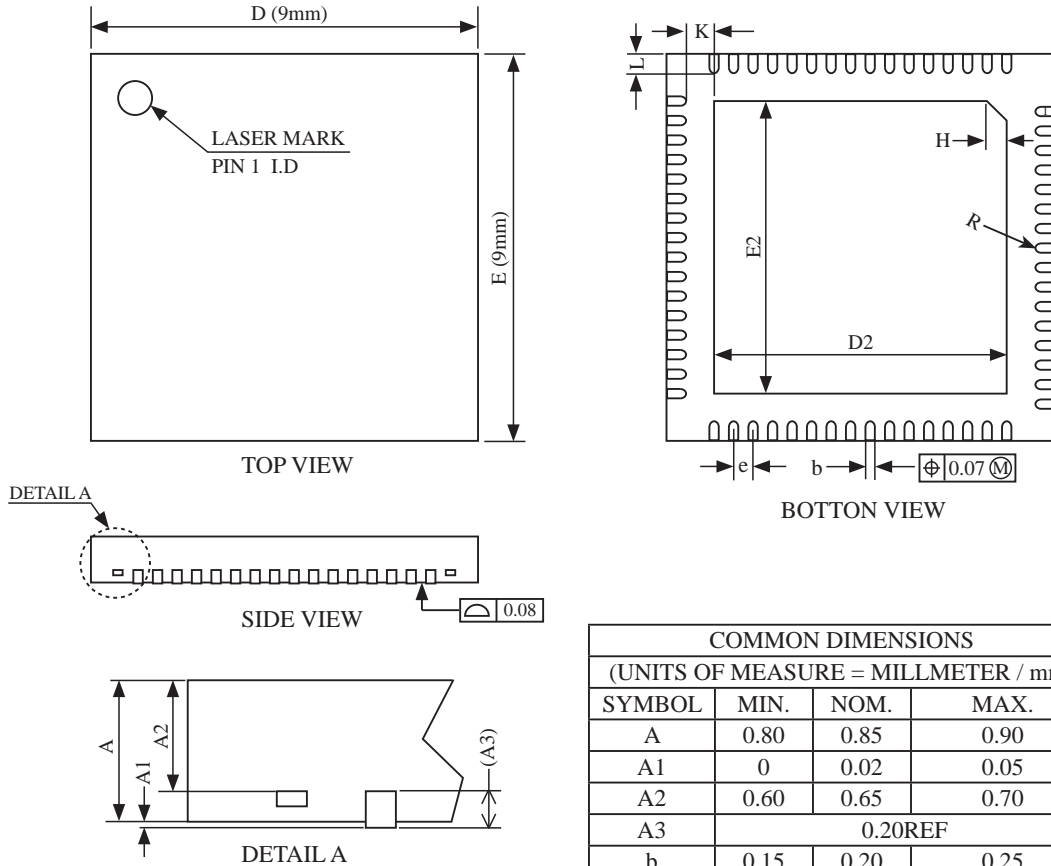


NOTES:  
 ALL DIMENSIONS MEET JEDEC STANDARD  
 MS-026 BEB DO NOT INCLUDE MOLD FLASH  
 OR PROTRUSIONS.



## 1.8.25 Dimension Drawings of QFN64

### QFN64 OUTLINE PACKAGE



COMMON DIMENSIONS			
(UNITS OF MEASURE = MILLMETER / mm)			
SYMBOL	MIN.	NOM.	MAX.
A	0.80	0.85	0.90
A1	0	0.02	0.05
A2	0.60	0.65	0.70
A3	0.20REF		
b	0.15	0.20	0.25
D	8.90	9.00	9.10
E	8.90	9.00	9.10
D2	5.90	6.00	6.10
E2	5.90	6.00	6.10
e	0.45	0.50	0.55
H	0.35REF		
K	0.40	-	-
L	0.30	0.40	0.50
R	0.09	-	-

NOTES:  
ALL DIMENSIONS DO NOT INCLUDE MOLD FLASH  
OR PROTRUSION

## 1.9 Special Peripheral Function(CCP/SPI,UART1/2/3/4) Switch

CCP is abbreviation for Capture, Compare, PWM

Special Peripheral function of STC154K60S2 series MCU, such as CCP/PWM SPI UART1 UART2 UART3 UART4 and so on, can be switched among several ports.

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000 0000
P_SW2	BAH	Peripheral function switch			PWM67_S	PWM2345_S		S4_S	S3_S	S2_S	xxxx x000

CCP can be switched in 3 groups of pins by selecting the control bits CCP_S1 and CCP_S0.		
CCP_S1	CCP_S0	CCP can be switched in P1 and P2 and P3
0	0	CCP on [P1.2/ECL,P1.1/CCP0,P1.0/CCP1]
0	1	CCP on [P3.4/ECL_2,P3.5/CCP0_2,P3.6/CCP1_2]
1	0	CCP on [P2.4/ECL_3,P2.5/CCP0_3,P2.6/CCP1_3]
1	1	Invalid

PWM2/PWM3/PWM4/PWM5/PWMFLT can be switched in 2 groups of pins by selecting the control bit PWM2345_S.	
PWM2345_S	PWM2/PWM3/PWM4/PWM5/PWMFLT can be switched between P2, P3, and P4
0	PWM2/PWM3/PWM4/PWM5/PWMFLT on [P3.7/PWM2, P2.1/PWM3, P2.2/PWM4, P2.3/PWM5, P2.4/PWMFLT]
1	PWM2/PWM3/PWM4/PWM5/PWMFLT on [P2.7/PWM2_2, P4.5/PWM3_2, P4.4/PWM4_2, P4.2/PWM5_2, P0.5/PWMFLT_2]

PWM6/PWM7 can be switched in 2 groups of pins by selecting the control bit PWM67_S.	
PWM67_S	PWM2/PWM3/PWM4/PWM5/PWMFLT can be switched between P0 and P1
0	PWM6/PWM7 on [P1.6/PWM6,P1.7/PWM7]
1	PWM6/PWM7 on [P0.7/PWM6_2,P0.6/PWM7_2]

SPI can be switched in 3 groups of pins by selecting the control bits SPI_S1 and SPI_S0		
SPI_S1	SPI_S0	SPI can be switched in P1 and P2 and P4
0	0	SPI on [P1.2/SS,P1.3/MOSI,P1.4/MISO,P1.5/SCLK]
0	1	SPI on [P2.4/SS_2,P2.3/MOSI_2,P2.2/MISO_2,P2.1/SCLK_2]
1	0	SPI on [P5.4/SS_3,P4.0/MOSI_3,P4.1/MISO_3,P4.3/SCLK_3]
1	1	Invalid

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0000 0000
P_SW2	BAH	Peripheral function switch			PWM67_S	PWM2345_S		S4_S	S3_S	S2_S	xxxx x000

UART1/S1 can be switched in 3 groups of pins by selecting the control bits S1_S0 and S1_S1.		
S1_S1	S1_S0	UART1/S1 can be switched between P1 and P3
0	0	UART1/S1 on [P3.0/RxD,P3.1/TxD]
0	1	UART1/S1 on [P3.6/RxD_2,P3.7/TxD_2]
1	0	UART1/S1 on [P1.6/RxD_3/XTAL2,P1.7/TxD_3/XTAL1] when UART1 is on P1, please using internal R/C clock.
1	1	Invalid

Recommended UART1 on [P3.6/RxD\_2,P3.7/TxD\_2] or [P1.6/RxD\_3/XTAL2,P1.7/TxD\_3/XTAL1].

UART2/S2 can be switched in 2 groups of pins by selecting the control bit S2_S.	
S2_S	UART2/S2 can be switched between P1 and P4
0	UART2/S2 on [P1.0/RxD2,P1.1/TxD2]
1	UART2/S2 on [P4.6/RxD2_2,P4.7/TxD2_2]

UART3/S3 can be switched in 2 groups of pins by selecting the control bit S3_S.	
S3_S	UART3/S3 can be switched between P0 and P5
0	UART3/S3 on [P0.0/RxD3,P0.1/TxD3]
1	UART3/S3 on [P5.0/RxD3_2,P5.1/TxD3_2]

UART4/S4 can be switched in 2 groups of pins by selecting the control bit S4_S.	
S4_S	UART4/S4 can be switched between P0 and P5
0	UART4/S4 on [P0.2/RxD4,P0.3/TxD4]
1	UART4/S4 on [P5.2/RxD4_2,P5.3/TxD4_2]

DPS : DPTR registers select bit.

0 : DPTR0 is selected

1 : DPTR1 is selected

---

## 1.9.1 Test Program that Switch CCP/PWM/PCA (C and ASM)

CCP is abbreviation for Capture, Compare and PWM.

### 1.C Program Listing

```
/*-----*/
/* --- Exam Program that switch STC15W4K32S4 series CCP/PCA/PWM in several ports--*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

#define FOSC 18432000L

//-----

sfr P_SW1 = 0xA2; //Peripheral function switch register

#define CCP_S0 0x10 //P_SW1.4
#define CCP_S1 0x20 //P_SW1.5

//-----

void main()
{
    ACC = P_SW1;
    ACC &= ~(CCP_S0 | CCP_S1); //CCP_S0=0 CCP_S1=0
    P_SW1 = ACC; // (P1.2/ECl, P1.1/CCP0, P1.0/CCP1)

// ACC = P_SW1;
// ACC &= ~(CCP_S0 | CCP_S1); //CCP_S0=1 CCP_S1=0
// ACC |= CCP_S0; // (P3.4/ECl_2, P3.5/CCP0_2, P3.6/CCP1_2)
// P_SW1 = ACC;
//
// ACC = P_SW1;
// ACC &= ~(CCP_S0 | CCP_S1); //CCP_S0=0 CCP_S1=1
// ACC |= CCP_S1; // (P2.4/ECl_3, P2.5/CCP0_3, P2.6/CCP1_3)
// P_SW1 = ACC;
    while (1); //program end
}
```

---

## 2. Assembler Listing

```
/*-----*/
/* --- Exam Program that switch STC15W4K32S4 series CCP/PCA/PWM in several ports--*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/
//suppose the frequency of test chip is 18.432MHz
#define FOSC 18432000L

//-----
P_SW1 EQU 0A2H //Peripheral function switch register

CCP_S0 EQU 10H //P_SW1.4
CCP_S1 EQU 20H //P_SW1.5

//-----
ORG 0000H
LJMP MAIN

//-----

ORG 0100H
MAIN:
MOV SP, #3FH

MOV A, P_SW1
ANL A, #0CFH //CCP_S0=0 CCP_S1=0
MOV P_SW1, A // (P1.2/ECI, P1.1/CCP0, P1.0/CCP1)

// MOV A, P_SW1
// ANL A, #0CFH //CCP_S0=1 CCP_S1=0
// ORL A, #CCP_S0 // (P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2)
// MOV P_SW1, A
//
// MOV A, P_SW1
// ANL A, #0CFH //CCP_S0=0 CCP_S1=1
// ORL A, #CCP_S1 // (P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3)
// MOV P_SW1, A
SJMP $ //program end

END
```

---

## 1.9.2 Test Program that Switch PWM2/3/4/5/PWMFLT (C and ASM)

### 1.C Program Listing

```
/*----- PWM(Pulse Width Modulation) -----*/
/*-----*/
/* --- Exam Program that switch STC15W4K32S4 series PWM2/3/4/5/PWMFLT in serveral ports--*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz
#include "reg51.h"

#define FOSC 18432000L

//-----

sfr P_SW2 = 0xBA; //Peripheral function switch register 2

#define PWM2345_S 0x10 //P_SW2.4

//-----

void main()
{
    P_SW2 &= ~PWM2345_S; //PWM2345_S=0 ( P3.7/PWM2, P2.1/PWM3,
                        //P2.2/PWM4, P2.3/PWM5, P2.4/PWMFLT )

    // P_SW2 |= PWM2345_S; //PWM2345_S=1 (P2.7/PWM2_2, P4.5/PWM3_2,
                        //P4.4/PWM4_2, P4.2/PWM5_2, P0.5/PWMFLT_2)

    while (1); //program end
}
```

---

## 2. Assembler Listing

```
/*----- PWM(Pulse Width Modulation) -----*/
/*-----*/
/* --- Exam Program that switch STC15W4K32S4 series PWM2/3/4/5/PWMFLT in serveral ports--*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#define FOSC 18432000L

//-----

P_SW2 EQU 0BAH //Peripheral function switch register 2
PWM2345_S EQU 10H //P_SW2.4

//-----

ORG 0000H
LJMP MAIN //Reset entrance
//-----

ORG 0100H

MAIN:
MOV SP, #3FH

ANL P_SW2, #NOT PWM2345_S //PWM2345_S=0 ( P3.7/PWM2, P2.1/PWM3,
//P2.2/PWM4, P2.3/PWM5, P2.4/PWMFLT )

// ORL P_SW2, #PWM2345_S //PWM2345_S=1 (P2.7/PWM2_2, P4.5/PWM3_2,
//P4.4/PWM4_2, P4.2/PWM5_2, P0.5/PWMFLT_2)

SJMP $ //program end

END
```

---

## 1.9.3 Test Program that Switch PWM6/PWM7 (C and ASM)

### 1.C Program Listing

```
/*----- PWM(Pulse Width Modulation) -----*/
/*-----*/
/* --- Exam Program that switch STC15W4K32S4 series PWM6/PWM7 in serveral ports-----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz
#include "reg51.h"
#define FOSC 18432000L

//-----

sfr P_SW2 = 0xBA; //Peripheral function switch register 2

#define PWM67_S 0x20 //P_SW2.5

//-----

void main()
{
    P_SW2 &= ~PWM67_S; //PWM67_S=0 ( P1.6/PWM6, P1.7/PWM7 )

    // P_SW2 |= PWM67_S; //PWM67_S=1 ( P0.7/PWM6_2, P0.6/PWM7_2 )

    while (1); //program end
}
```



---

## 2. Assembler Listing

```
/*----- PWM(Pulse Width Modulation) -----*/
/*-----*/
/* --- Exam Program that switch STC15W4K32S4 series PWM6/PWM7 in serveral ports-----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#define FOSC 18432000L

//-----

P_SW2 EQU 0BAH //Peripheral function switch register 2
PWM67_S EQU 20H //P_SW2.5

//-----

ORG 0000H
LJMP MAIN //Reset entrance

//-----

ORG 0100H

MAIN:
MOV SP, #3FH
ANL P_SW2, #NOT PWM67_S //PWM67_S=0 ( P1.6/PWM6, P1.7/PWM7 )
// ORL P_SW2, #PWM67_S //PWM67_S=1 ( P0.7/PWM6_2, P0.6/PWM7_2 )

SJMP $ //program end

END
```

---

## 1.9.4 Test Program that Switch SPI (C and ASM)

### 1.C Program Listing

```
/*-----*/
/* --- Exam Program that switch STC15W4K32S4 series SPI in several ports -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/
//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

#define FOSC 18432000L

//-----

sfr      P_SW1  =      0xA2;                //Peripheral function switch register

#define   SPI_S0 0x04                //P_SW1.2
#define   SPI_S1 0x08                //P_SW1.3

//-----

void main()
{
    ACC    =      P_SW1;
    ACC    &=    ~(SPI_S0 | SPI_S1);    //SPI_S0=0 SPI_S1=0
    P_SW1  =      ACC;                //(P1.2/SS, P1.3/MOSI, P1.4/MISO, P1.5/SCLK)

//    ACC    =      P_SW1;
//    ACC    &=    ~(SPI_S0 | SPI_S1);    //SPI_S0=1 SPI_S1=0
//    ACC    |=    SPI_S0;                //(P2.4/SS_2, P2.3/MOSI_2, P2.2/MISO_2, P2.1/SCLK_2)
//    P_SW1  =      ACC;
//
//    ACC    =      P_SW1;
//    ACC    &=    ~(SPI_S0 | SPI_S1);    //SPI_S0=0 SPI_S1=1
//    ACC    |=    SPI_S1;                //(P5.4/SS_3, P4.0/MOSI_3, P4.1/MISO_3, P4.3/SCLK_3)
//    P_SW1  =      ACC;

    while (1);                        //program end
}
```

---

## 2. Assembler Listing

```
/*-----*/
/* --- Exam Program that switch STC15W4K32S4 series SPI in several ports -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#define FOSC 18432000L

//-----

P_SW1 EQU 0A2H //Peripheral function switch register
SPI_S0 EQU 04H //P_SW1.2
SPI_S1 EQU 08H //P_SW1.3

//-----

ORG 0000H
LJMP MAIN

//-----

ORG 0100H

MAIN:
MOV SP, #3FH

MOV A, P_SW1
ANL A, #0F3H //SPI_S0=0 SPI_S1=0
MOV P_SW1, A //(P1.2/SS, P1.3/MOSI, P1.4/MISO, P1.5/SCLK)

// MOV A, P_SW1
// ANL A, #0F3H //SPI_S0=1 SPI_S1=0
// ORL A, #SPI_S0 //(P2.4/SS_2, P2.3/MOSI_2, P2.2/MISO_2, P2.1/SCLK_2)
// MOV P_SW1, A

// MOV A, P_SW1
// ANL A, #0F3H //SPI_S0=0 SPI_S1=1
// ORL A, #SPI_S1 //(P5.4/SS_3, P4.0/MOSI_3, P4.1/MISO_3, P4.3/SCLK_3)
// MOV P_SW1, A

SJMP $ //program end

END
```

---

## 1.9.5 Test Program that Switch UART1 (C and ASM)

### 1.C Program Listing

```
/*-----*/
/* --- Exam Program that switch STC15W4K32S4 series UART1 in serveral ports -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/
//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

#define FOSC 18432000L

//-----

sfr    P_SW1 =    0xA2;                //Peripheral function switch register
#define  S1_S0 0x40                    //P_SW1.6
#define  S1_S1 0x80                    //P_SW1.7

//-----

void main()
{
    ACC =    P_SW1;
    ACC  &=  ~(S1_S0 | S1_S1);        //S1_S0=0 S1_S1=0
    P_SW1 =    ACC;                    //(P3.0/RxD, P3.1/TxD)

//    ACC =    P_SW1;
//    ACC  &=  ~(S1_S0 | S1_S1);        //S1_S0=1 S1_S1=0
//    ACC  |=  S1_S0;                    //(P3.6/RxD_2, P3.7/TxD_2)
//    P_SW1 =    ACC;

//    ACC =    P_SW1;
//    ACC  &=  ~(S1_S0 | S1_S1);        //S1_S0=0 S1_S1=1
//    ACC  |=  S1_S1;                    //(P1.6/RxD_3, P1.7/TxD_3)
//    P_SW1 =    ACC;

    while (1);                        //program end
}
```

---

## 2. Assembler Listing

```
/*-----*/
/* --- Exam Program that switch STC15W4K32S4 series UART1 in serveral ports -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/
//suppose the frequency of test chip is 18.432MHz

#define FOSC 18432000L

//-----
P_SW1 EQU 0A2H //Peripheral function switch register

S1_S0 EQU 40H //P_SW1.6
S1_S1 EQU 80H //P_SW1.7

//-----

ORG 0000H
LJMP MAIN

//-----

ORG 0100H

MAIN:
MOV SP, #3FH
MOV A, P_SW1
ANL A, #03FH //S1_S0=0 S1_S1=0
MOV P_SW1, A // (P3.0/RxD, P3.1/TxD)

// MOV A, P_SW1
// ANL A, #03FH //S1_S0=1 S1_S1=0
// ORL A, #S1_S0 // (P3.6/RxD_2, P3.7/TxD_2)
// MOV P_SW1, A
//
// MOV A, P_SW1
// ANL A, #03FH //S1_S0=0 S1_S1=1
// ORL A, #S1_S1 // (P1.6/RxD_3, P1.7/TxD_3)
// MOV P_SW1, A

SJMP $ //program end

END
```

---

## 1.9.6 Test Program that Switch UART2 (C and ASM)

### 1.C Program Listing

```
/*-----*/
/* --- Exam Program that switch STC15W4K32S4 series UART2 in several ports -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

#define FOSC 18432000L

//-----

sfr P_SW2 = 0xBA; //Peripheral function switch register
#define S2_S 0x01 //P_SW2.0

//-----

void main()
{
    P_SW2 &= ~S2_S; //S2_S0=0 (P1.0/RxD2, P1.1/TxD2)

//    P_SW2 |= S2_S; //S2_S0=1 (P4.6/RxD2_2, P4.7/TxD2_2)

    while (1); //program end
}
```

---

## 2. Assembler Listing

```
/*-----*/
/* --- Exam Program that switch STC15W4K32S4 series UART2 in serveral ports -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/
//suppose the frequency of test chip is 18.432MHz

#define FOSC 18432000L

//-----

P_SW2 EQU 0BAH //Peripheral function switch register
S2_S EQU 01H //P_SW2.0

//-----

ORG 0000H
LJMP MAIN

//-----

ORG 0100H

MAIN:
MOV SP, #3FH
ANL P_SW2, #NOT S2_S //S2_S0=0 (P1.0/RxD2, P1.1/TxD2)
// ORL P_SW2, #S2_S //S2_S0=1 (P4.6/RxD2_2, P4.7/TxD2_2)
SJMP $ //program end

END
```

---

## 1.9.7 Test Program that Switch UART3 (C and ASM)

### 1.C Program Listing

```
/*-----*/
/* --- Exam Program that switch STC15W4K32S4 series UART3 in serveral ports -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

#define FOSC 18432000L

//-----

sfr P_SW2 = 0xBA; //Peripheral function switch register

#define S3_S 0x02 //P_SW2.1

//-----

void main()
{
    P_SW2 &= ~S3_S; //S3_S0=0 (P0.0/RxD3, P0.1/TxD3)

//    P_SW2 |= S3_S; //S3_S0=1 (P5.0/RxD3_2, P5.1/TxD3_2)

    while (1); //program end
}
```



---

## 2. Assembler Listing

```
/*-----*/
/* --- Exam Program that switch STC15W4K32S4 series UART3 in serveral ports -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#define FOSC 18432000L

//-----

P_SW2 EQU 0BAH //Peripheral function switch register
S3_S EQU 02H //P_SW2.1

//-----

ORG 0000H
LJMP MAIN

//-----

ORG 0100H

MAIN:
MOV SP, #3FH
ANL P_SW2, #NOT S3_S //S3_S0=0 (P0.0/RxD3, P0.1/TxD3)
// ORL P_SW2, #S3_S //S3_S0=1 (P5.0/RxD3_2, P5.1/TxD3_2)
SJMP $ //program end

END
```

---

## 1.9.8 Test Program that Switch UART4 (C and ASM)

### 1.C Program Listing

```
/*-----*/
/* --- Exam Program that switch STC15W4K32S4 series UART4 in several ports -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

#define FOSC 18432000L

//-----

sfr P_SW2 = 0xBA; //Peripheral function switch register

#define S4_S 0x04 //P_SW2.2

//-----

void main()
{
    P_SW2 &= ~S4_S; //S4_S0=0 (P0.2/RxD4, P0.3/TxD4)

//    P_SW2 |= S4_S; //S4_S0=1 (P5.2/RxD4_2, P5.3/TxD4_2)

    while (1); //program end
}
```

---

## 2. Assembler Listing

```
/*-----*/
/* --- Exam Program that switch STC15W4K32S4 series UART4 in serveral ports -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#define FOSC 18432000L

//-----

P_SW2 EQU 0BAH //Peripheral function switch register
S4_S0 EQU 04H //P_SW2.2

//-----

ORG 0000H
LJMP MAIN

//-----

ORG 0100H

MAIN:
MOV SP, #3FH
ANL P_SW2, #NOT S4_S //S4_S0=0 (P0.2/RxD4, P0.3/TxD4)
// ORL P_SW2, #S4_S //S4_S0=1 (P5.2/RxD4_2, P5.3/TxD4_2)
SJMP $ //program end

END
```

---

## 1.10 Global Unique Identification Number (ID)

The latest generation of STC MCU ---STC15 series MCU all have a global unique identification number (ID) when out of factory. The global unique ID number is located in the last 7 bytes units of program memory in the latest STC15 series MCU, which can not be modified. But the all program area of IAP15 series MCU, which is open to user, can be modified. That using STC15 series MCU and its EEPROM function which began to use from the starting address 0000H can effectively eliminate the attack to global unique ID when STC15 series MCU is protected by global unique ID.

In addition to the program memory of the last 7 bytes units store the only global ID, the content of internal RAM units F1H ~ F7H also is the global unique ID number. User can use “MOV @Ri” instruction read RAM unit F1~F7 to get the ID number after power on. If users need to the unique identification number to encrypt their procedures, detecting the procedures not be illegally modified should be done first. preventing the decryption to modification program, bypassing the judgment to global unique ID number .

Recommend to use the program memory of the last 7 bytes of global unique ID, instead of using the internal RAM units F1H - F7H global unique ID number. Because the program memory of the last 7 bytes of a global unique ID number is more than difficult to attack than the internal RAM units F1H - F7H.

*//The following example program written by C language is to read internal ID number from RAM or Program Memory.*

### 1.C Program Listing

```
/*-----*/
/* --- Exam Program that read internal ID number -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

typedef unsigned char    BYTE;
typedef unsigned int     WORD;

#define  URMD  0                //0: Timer 2 as Baud Rate Generator
                                //1:Timer1 in mode 0 (16-bit auto-reload mode) as Baud Rate Generator
                                //2:Timer1 in mode 2 (8-bit auto-reload mode) as Baud Rate Generator
```

---

```

sfr    T2H    =    0xd6;                //High 8 bit of Timer 2
sfr    T2L    =    0xd7;                //Low 8 bit of Timer 2

sfr    AUXR   =    0x8e;                //Auxiliary Register

#define  ID_ADDR_RAM  0xf1                //ID number be stored in RAM location 0F1H

//ID number be stored in the last 7 bytes of program memory

//#define ID_ADDR_ROM  0x3ff9            //16K MCU(eg. STC15W4K16S4)
//#define ID_ADDR_ROM  0x7ff9            //32K MCU(eg. STC15W4K32S4)
//#define ID_ADDR_ROM  0x9ff9            //40K MCU(eg. STC15W4K40S4)
//#define ID_ADDR_ROM  0xbff9            //48K MCU(eg. STC15W4K48S4)
#define  ID_ADDR_ROM  0xdff9            //56K MCU(eg. STC15W4K56S4)

//-----
void  InitUart();
void  SendUart(BYTE dat);
//-----
void  main()
{
    BYTE  idata    *iptr;
    BYTE  code     *cptr;
    BYTE  i;

    InitUart();                //initialize serial port

    iptr = ID_ADDR_RAM;        //read ID number from RAM
    for (i=0; i<7; i++)        //read 7 bytes
    {
        SendUart(*iptr++);    //send ID number to serial port
    }
    cptr = ID_ADDR_ROM;        //read ID number from program memory
    for (i=0; i<7; i++)        //read 7 bytes
    {
        SendUart(*cptr++);    //send ID number to serial port
    }

    while (1);                //program end
}

/*-----
Initialize serial port
-----*/
void InitUart()
{
    SCON    =    0x5a;                //UART1 in 8-bit variable baud rate mode

```

---

---

```

#if URMD == 0
    T2L = 0xd8; //Set the auto-reload parameter
    T2H = 0xff; //115200 bps(65536-18432000/4/115200)
    AUXR = 0x14; //T2 in 1T mode, strat up Timer 2
    AUXR |= 0x01; //Timer 2 as baud-rate Generator of UART1
#elif URMD == 1
    AUXR = 0x40; //T1 in 1T mode
    TMOD = 0x00; //Timer1 in mode 0(16-bit auto-reload mode)
    TL1 = 0xd8; //Set the auto-reload parameter
    TH1 = 0xff; //115200 bps(65536-18432000/4/115200)
    TR1 = 1; //strat up Timer 1
#else
    TMOD = 0x20; //Timer1 in mode 2 (8-bit auto-reload mode)
    AUXR = 0x40; //T1 in 1T mode
    TH1 = TL1 = 0xfb; //115200 bps(256 - 18432000/32/115200)
    TR1 = 1;
#endif
}
/*-----*/
Send serial port data
-----*/
void SendUart(BYTE dat)
{
    while (!TI); //wait to finish transmitting
    TI = 0;
    SBUF = dat; //Send serial port data
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- Exam Program that read internal ID number -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#define URMD 0 //0: Timer 2 as Baud Rate Generator
//1:Timer1 in mode 0 (16-bit auto-reload mode) as Baud Rate Generator
//2:Timer1 in mode 2 (8-bit auto-reload mode) as Baud Rate Generator

```

---

```

T2H   DATA   0D6H           //High 8 bit of Timer 2
T2L   DATA   0D7H           //Low 8 bit of Timer 2
AUXR  DATA   08EH           //Auxiliary Register
//-----

#define  ID_ADDR_RAM  0xf1           //ID number be stored in RAM location 0F1H
//ID number be stored in the last 7 bytes of program memory

//#define ID_ADDR_ROM  0x3ff9         //16K MCU(eg. STC15W4K16S4)
//#define ID_ADDR_ROM  0x7ff9         //32K MCU(eg. STC15W4K32S4)
//#define ID_ADDR_ROM  0x9ff9         //40K MCU(eg. STC15W4K40S4)
//#define ID_ADDR_ROM  0xbff9         //48K MCU(eg. STC15W4K48S4)
#define  ID_ADDR_ROM  0xdff9         //56K MCU(eg. STC15W4K56S4)

//-----

        ORG     0000H
        LJMP   MAIN

//-----

        ORG     0100H

MAIN:
        MOV    SP,    #3FH

        LCALL  INIT_UART             ///initialize serial port

        MOV    R0,    #ID_ADDR_RAM    //read ID number from RAM
        MOV    R1,    #7              //read 7 bytes

NEXT1:
        MOV    A,     @R0
        LCALL  SEND_UART             //send ID number to serial port
        INC    R0
        DJNZ  R1,     NEXT1

        MOV    DPTR,  #ID_ADDR_ROM    //read ID number from program memory
        MOV    R1,    #7              //read 7 bytes

NEXT2:
        CLR    A
        MOVC  A,     @A+DPTR
        LCALL  SEND_UART             //send ID number to serial port
        INC    DPTR
        DJNZ  R1,     NEXT2

        SJMP  $                      //program end

```

---

---

/\*-----

Initialize serial port

-----\*/

INIT\_UART:

```
    MOV    SCON,  #5AH           //UART1 in 8-bit variable baud rate mode
#if      URMD == 0
    MOV    T2L,   #0D8H         //Set the auto-reload value (65536-18432000/4/115200)
    MOV    T2H,   #0FFH
    MOV    AUXR,  #14H         //T2 in 1T mode, strat up Timer 2
    ORL    AUXR,  #01H         //Timer 2 as baud-rate Generator of UART1
#elif    URMD == 1
    MOV    AUXR,  #40H           //T1 in 1T mode
    MOV    TMOD,  #00H         //Timer1 in mode 0(16-bit auto-reload mode)
    MOV    TL1,   #0D8H         //Set the auto-reload value
                                           //(65536-18432000/4/115200)
    MOV    TH1,   #0FFH
    SETB   TR1                 //strat up Timer 1
#else
    MOV    TMOD,  #20H         //Timer1 in mode 2 (8-bit auto-reload mode)
    MOV    AUXR,  #40H         //T1 in 1T mode
    MOV    TL1,   #0FBH         //115200 bps(256 - 18432000/32/115200)
    MOV    TH1,   #0FBH
    SETB   TR1
#endif
    RET
```

/\*-----

Send serial port data

-----\*/

SEND\_UART:

```
    JNB    TI,    $             //wait to finish transmitting
    CLR    TI
    MOV    SBUF,  A             //Send serial port data
    RET

    END
```

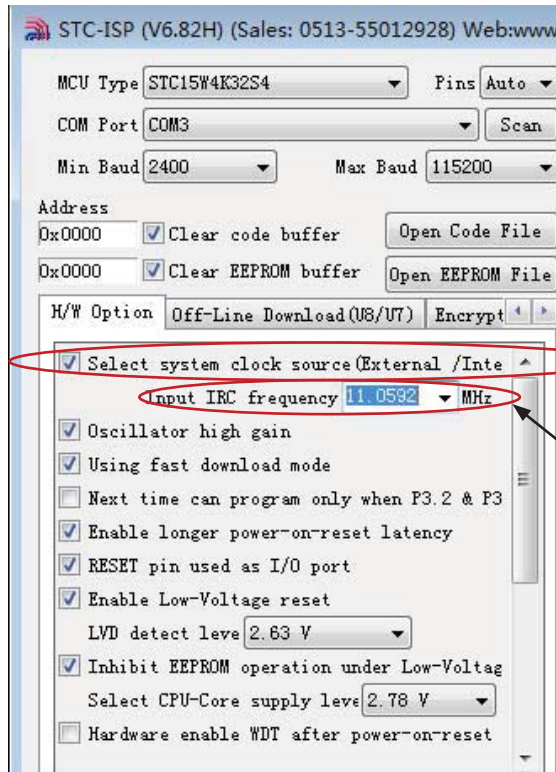


# Chapter 2 Clock, Reset and Power Management

## 2.1 Clock

The STC15W4K32S4 series MCU has two clock sources: internal high precise R/C clock and external clock (external input clock or external crystal oscillator). Internal high-precise R/C clock with  $\pm 0.3\%$  error has  $\pm 1\%$  temperature drift( $-40 \sim +85$  ) while  $\pm 0.6\%$  in normal temperature ( $-20 \sim +65$  ).

### 2.1.1 On-Chip Configurable Clock



STC-ISP (V6.82H) (Sales: 0513-55012928) Web:www:

MCU Type: STC15W4K32S4 Pins: Auto

COM Port: COM3 Scan

Min Baud: 2400 Max Baud: 115200

Address: 0x0000  Clear code buffer Open Code File

0x0000  Clear EEPROM buffer Open EEPROM File

H/W Option: Off-Line Download (US/UT) Encrypt

Select system clock source (External /Internal)   
 Choice: Select the internal R/C clock  
 No-Choice: Select the external clock

Input IRC frequency: 11.0592 MHz   
 Select the frequency of internal high precise R/C clock that the user program is running next time. The frequency also can be input directly. Input range: 5MHz ~ 35MHz

Oscillator high gain

Using fast download mode

Next time can program only when P3.2 & P3

Enable longer power-on-reset latency

RESET pin used as I/O port

Enable Low-Voltage reset

LVD detect level: 2.63 V

Inhibit EEPROM operation under Low-Voltage

Select CPU-Core supply level: 2.78 V

Hardware enable WDT after power-on-reset

## 2.1.2 Divider for System Clock

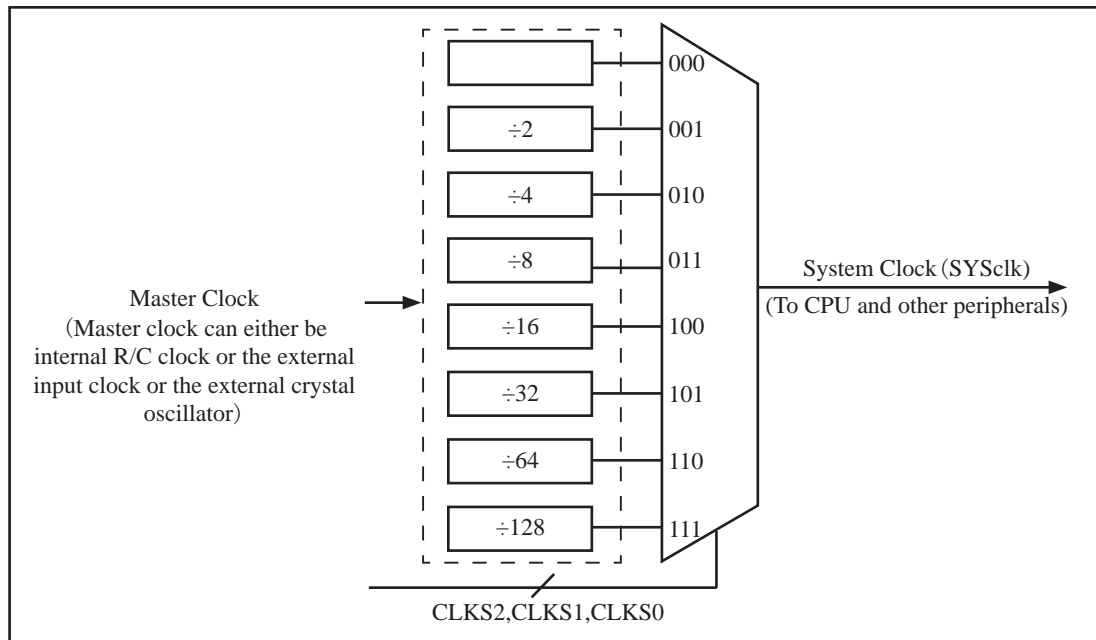
A clock divider(CLK\_DIV) is designed to slow down the operation speed of STC15W4K32S4 series MCU, to save the operating power dynamically. User can slow down the MCU by means of writing a non-zero value to the CLKS[2:0] bits in the CLK\_DIV register. This feature is especially useful to save power consumption in idle mode as long as the user changes the CLKS[2:0] to a non-zero value before entering the idle mode.

Clock Division Register CLK\_DIV (PCON2):

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CLK_DIV (PCON2)	97H	name	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCKO_2	CLKS2	CLKS1	CLKS0

CLKS2	CLKS1	CLKS0	the control bit of system clock (System clock refers to the master clock that has been divided frequency, which is offered to CPU, UARTs, SPI, Timers, CCP/PWM/PCA and A/D Converter)
0	0	0	Master clock frequency/1, No division
0	0	1	Master clock frequency/2
0	1	0	Master clock frequency/4
0	1	1	Master clock frequency/8
1	0	0	Master clock frequency/16
1	0	1	Master clock frequency/32
1	1	0	Master clock frequency/64
1	1	1	Master clock frequency/128

The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator.



**Clock Structure**

## 2.1.3 Programmable Clock Output (or as Frequency Divider)

STC15W4K32S4 series MCU has six channel programmable clock outputs. They are Master clock output MCLKO/P5.4, Timer 0 programmable clock output T0CLKO/P3.5, Timer 1 programmable clock output T1CLKO/P3.4, Timer 2 programmable clock output T2CLKO/P3.0, Timer 3 programmable clock output T3CLKO/P0.4, Timer 4 programmable clock output T4CLKO/P0.6. The speed of external programmable clock output is also not more than 13.5MHz, because the output speed of I/O port of STC15 series MCU is not more than 13.5MHz.

### 2.1.3.1 Special Function Registers Related to Programmable Clock Output

Symbol	Description	Address	Bit Address and Symbol								Value after Power-on or Reset
			MSB				LSB				
AUXR	Auxiliary register	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C $\bar{T}$	T2x12	EXTRAM	S1ST2	0000 0001B
INT_CLKO AUXR2	External Interrupt enable and Clock output register	8FH		EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO	x000 0000B
CLK_DIV (PCON2)	Clock Division register	97H	MCKO_S1	MCKO_S1	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000B
T4T3M	Timer 4 and Timer 3 Mode register	D1H	T4R	T4_C $\bar{T}$	T4x12	T4CLKO	T3R	T3_C $\bar{T}$	T3x12	T3CLKO	0000 0000B

The statement (used in C language) of Special function registers INT\_CLKO/AUXR/CLK\_DIV/T4T3M:

```
sfr INT_CLKO = 0x8F; //The address statement of special function register INT_CLKO
sfr AUXR = 0x8E; //The address statement of Special function register AUXR
sfr CLK_DIV = 0x97; //The address statement of Special function register CLK_DIV
sfr T4T3M = 0xD1; //The address statement of Special function register T4T3M
```

The statement (used in Assembly language) of Special function registers INT\_CLKO/AUXR/CLK\_DIV/T4T3M:

```
INT_CLKO EQU 8FH ;The address statement of special function register INT_CLKO
AUXR EQU 8EH ;The address statement of Special function register AUXR
CLK_DIV EQU 97H ;The address statement of Special function register CLK_DIV
T4T3M EQU D1H ;The address statement of Special function register T4T3M
```

#### 1. CLK\_DIV (PCON2) : Clock Division register(Non bit addressable)

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CLK_DIV (PCON2)	97H	name	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0

ADRJ the adjustment bit of ADC result

- 0 ADC\_RES[7:0] store high 8-bit ADC result ADC\_RESL[1:0] store low 2-bit ADC result
- 1 ADC\_RES[1:0] store high 2-bit ADC result ADC\_RESL[7:0] store low 8-bit ADC result

Tx\_Rx the set bit of relay and broadcast mode of UART1

- 0 UART1 works on normal mode
- 1 UART1 works on relay and broadcast mode that to say output the input level state of RxD port to the outside TxD pin in real time, namely the external output of TxD pin can reflect the input level state of RxD port.

the RxD and TxD of UART1 can be switched in 3 groups of pins: [RxD/P3.0, TxD/P3.1];

[RxD\_2/P3.6, TxD\_2/P3.7];

[RxD\_3/P1.6, TxD\_3/P1.7].

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
CLK_DIV (PCON2)	97H	Clock Division register	MCKO_S1	MCKO_S0	AD RJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000
INT_CLKO (AUXR2)	8FH	External Interrupt enable and Clock output register	-	EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO	x000 0000

MCKO_S2	MCKO_S1	MCKO_S0	the control bit of master clock output by dividing the frequency (The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator)
0	0	0	Master clock do not output external clock
0	0	1	Master clock output external clock but its frequency do not be divided and the output clock frequency = MCLK / 1
0	1	0	Master clock output external clock but its frequency is divided by 2 and the output clock frequency = MCLK / 2
0	1	1	Master clock output external clock but its frequency is divided by 4 and the output clock frequency = MCLK / 4
1	0	0	Master clock output external clock but its frequency is divided by 4 and the output clock frequency = MCLK / 16

The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator.  
MCLK is the frequency of master clock.

STC15W4K32S4 series MCU output master clock on MCLKO/P5.4

MCLKO\_2 to select Master Clock output on where

0 Master Clock output on MCLKO/P5.4

1 Master Clock output on MCLKO\_2/XTAL2/P1.6

The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator.

CLKS2	CLKS1	CLKS0	the control bit of system clock (System clock refers to the master clock that has been divided frequency, which is offered to CPU, UARTs, SPI, Timers, CCP/PWM/PCA and A/D Converter)
0	0	0	Master clock frequency/1, No division
0	0	1	Master clock frequency/2
0	1	0	Master clock frequency/4
0	1	1	Master clock frequency/8
1	0	0	Master clock frequency/16
1	0	1	Master clock frequency/32
1	1	0	Master clock frequency/64
1	1	1	Master clock frequency/128

The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator.

## 2. INT\_CLKO (AUXR2) : External Interrupt Enable and Clock Output register

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name		EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO

**B0 - T0CLKO** : Whether is P3.5/T1 configured for Timer 0(T0) programmable clock output T0CLKO or not.

1, P3.5/T1 is configured for Timer0 programmable clock output T0CLKO, the clock output frequency = T0 overflow/2

If Timer/Counter 0 in mode 0 (16 bit auto-reloadable mode),

and if  $C/\overline{T} = 0$ , namely Timer/Counter 0 count on the internal system clock,

When T0 in 1T mode (AUXR.7/T0x12=1), the output frequency = (SYSclk)/(65536-[RL\_TH0, RL\_TL0])/2

When T0 in 12T mode (AUXR.7/T0x12=0), the output frequency = (SYSclk) /12/ (65536-[RL\_TH0, RL\_TL0])/2

and if  $C/\overline{T} = 1$ , namely Timer/Counter 0 count on the external pulse input from P3.4/T0,

the output frequency = (T0\_Pin\_CLK) / (65536-[RL\_TH0, RL\_TL0])/2

If Timer/Counter 0 in mode 2 (8 bit auto-reloadable mode),

and if  $C/\overline{T} = 0$ , namely Timer/Counter 0 count on the internal system clock,

When T0 in 1T mode(AUXR.7/T0x12=1), the output frequency = (SYSclk) / (256-TH0) / 2

When T0 in 12T mode(AUXR.7/T0x12=0), the output frequency = (SYSclk) / 12 / (256-TH0) / 2

and if  $C/\overline{T} = 1$ , namely Timer/Counter 0 count on the external pulse input from P3.4/T0,

the output frequency = (T0\_Pin\_CLK) / (256-TH0) / 2

0, P3.5/T1 is not configure for Timer 0 programmable clock output T0CLKO

**B1 - T1CLKO** : Whether is P3.4/T0 configured for Timer 1(T1) programmable clock output T1CLKO or not.

1, P3.4/T0 is configured for Timer1 programmable clock output T1CLKO, the clock output frequency = T1 overflow/2

If Timer/Counter 1 in mode 1 (16 bit auto-reloadable mode),

and if  $C/\overline{T} = 0$ , namely Timer/Counter 1 count on the internal system clock,

When T1 in 1T mode (AUXR.6/T1x12=1), the output frequency = (SYSclk)/(65536-[RL\_TH1, RL\_TL1])/2

When T1 in 12T mode (AUXR.6/T1x12=0), the output frequency = (SYSclk) /12/ (65536-[RL\_TH1, RL\_TL1])/2

and if  $C/\overline{T} = 1$ , namely Timer/Counter 1 count on the external pulse input from P3.5/T1,

the output frequency = (T1\_Pin\_CLK) / (65536-[RL\_TH1, RL\_TL1])/2

If Timer/Counter 1 in mode 2 (8 bit auto-reloadable mode),

and if  $C/\overline{T} = 0$ , namely Timer/Counter 1 count on the internal system clock,

When T1 in 1T mode(AUXR.6/T1x12=1), the output frequency = (SYSclk) / (256-TH1) / 2

When T1 in 12T mode(AUXR.6/T1x12=0), the output frequency = (SYSclk) / 12 / (256-TH1) / 2

and if  $C/\overline{T} = 1$ , namely Timer/Counter 1 count on the external pulse input from P3.5/T1,

the output frequency = (T1\_Pin\_CLK) / (256-TH1) / 2

0, P3.4/T0 is not configure for Timer 1 programmable clock output T1CLKO

**B2 - T2CLKO** : Whether is P3.0 configured for Timer 2(T2) programmable clock output T2CLKO or not.

1, P3.0 is configured for Timer2 programmable clock output T2CLKO, the clock output frequency = T2 overflow/2

If T2\_  $C/\overline{T} = 0$ , namely Timer/Counter 2 count on the internal system clock,

When T2 in 1T mode (AUXR.2/T2x12=1), the output frequency = (SYSclk)/(65536-[RL\_TH2, RL\_TL2])/2

When T2 in 12T mode (AUXR.2/T2x12=0), the output frequency = (SYSclk) /12/ (65536-[RL\_TH2, RL\_TL2])/2

If T2\_  $C/\overline{T} = 1$ , namely Timer/Counter 2 count on the external pulse input from P3.1/T2,

the output frequency = (T2\_Pin\_CLK) / (65536-[RL\_TH2, RL\_TL2])/2

0, P3.0 is not configure for Timer 2 programmable clock output T2CLKO

## 2. INT\_CLKO (AUXR2) : External Interrupt Enable and Clock Output register

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name		EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO

B4 - EX2 : Enable bit of External Interrupt 2( $\overline{\text{INT2}}$ )

B5 - EX3 : Enable bit of External Interrupt 3( $\overline{\text{INT3}}$ )

B6 - EX4 : Enable bit of External Interrupt 4( $\overline{\text{INT4}}$ )

## 3. AUXR : Auxiliary register (Address:8EH, Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/ $\overline{\text{T}}$	T2x12	EXTRAM	S1ST2

B7 - T0x12 : Timer 0 clock source bit.

0 : The clock source of Timer 0 is SYSclk/12. It will compatible to the traditional 8051 MCU

1 : The clock source of Timer 0 is SYSclk/1. It will drive the T0 faster than a traditional 8051 MCU

B6 - T1x12 : Timer 1 clock source bit.

0 : The clock source of Timer 1 is SYSclk/12. It will compatible to the traditional 8051 MCU

1 : The clock source of Timer 1 is SYSclk/1. It will drive the T0 faster than a traditional 8051 MCU

If T1 is used as the baud-rate generator of UART1, T1x12 will decide whether UART1 is 1T or 12T.

B5 - UART\_M0x6 : Baud rate select bit of UART1 while it is working under 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.

B4 - T2R Timer 2 Run control bit

0 : not run Timer 2;

1 : run Timer 2.

B3 - T2\_C/ $\overline{\text{T}}$ : Counter or timer 2 selector

0 : as Timer (namely count on internal system clock)

1 : as Counter (namely count on the external pulse input from T2/P3.1)

B2 - T2x12 : Timer 2 clock source bit.

0 : The clock source of Timer 2 is SYSclk/12.

1 : The clock source of Timer 2 is SYSclk/1.

If T2 is used as the baud-rate generator of UART1 or UART2, T1x12 will decide whether UART1 or UART2 is 1T or 12T.

B1 - EXTRAM : Internal / external RAM access control bit.

0 : On-chip auxiliary RAM is enabled.

1 : On-chip auxiliary RAM is always disabled.

B0 - S1ST2 : the control bit that UART1 select Timer 2 as its baud-rate generator.

0 : Select Timer 1 as the baud-rate generator of UART1

1 : Select Timer 2 as the baud-rate generator of UART1. Timer 1 is released to use in other functions.

#### 4. T4T3M : Timer 4 and Timer 3 Mode register (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

B7 - T4R Timer 4 Run control bit

- 0 : not run Timer 4;
- 1 : run Timer 4.

B6 - T4\_C/T: Counter or timer 4 selector

- 0 : as Timer (namely count on internal system clock)
- 1 : as Counter (namely count on the external pulse input from T4/P0.7)

B5 - T4x12 : Timer 4 clock source bit.

- 0 : The clock source of Timer 4 is SYSclk/12.
- 1 : The clock source of Timer 4 is SYSclk/1.

B4 - T4CLKO : Whether is P0.6 configured for Timer 4(T4) programmable clock output T4CLKO or not.

- 1, P0.6 is configured for Timer 4 programmable clock output T4CLKO, the clock output frequency =  $T4 \text{ overflow} / 2$

If T4\_C/T = 0, namely Timer/Counter 4 count on the internal system clock,

When T4 in 1T mode (T4T3.5/T4x12=1), the output frequency =  $(\text{SYSclk}) / (65536 - [\text{RL\_TH4}, \text{RL\_TL4}]) / 2$

When T4 in 12T mode (T4T3.5/T4x12=0), the output frequency =  $(\text{SYSclk}) / 12 / (65536 - [\text{RL\_TH4}, \text{RL\_TL4}]) / 2$

If T4\_C/T = 1, namely Timer/Counter 4 count on the external pulse input from P0.7/T4,

the output frequency =  $(T4\_Pin\_CLK) / (65536 - [\text{RL\_TH4}, \text{RL\_TL4}]) / 2$

- 0, P0.6 is not configure for Timer 4 programmable clock output T4CLKO

B3 - T3R Timer 3 Run control bit

- 0 : not run Timer 3;
- 1 : run Timer 3.

B2 - T3\_C/T: Counter or timer 3 selector

- 0 : as Timer (namely count on internal system clock)
- 1 : as Counter (namely count on the external pulse input from T3/P0.5)

B1 - T3x12 : Timer 3 clock source bit.

- 0 : The clock source of Timer 3 is SYSclk/12.
- 1 : The clock source of Timer 3 is SYSclk/1.

B0 - T3CLKO : Whether is P0.4 configured for Timer 3(T3) programmable clock output T3CLKO or not.

- 1, P0.4 is configured for Timer 3 programmable clock output T3CLKO, the clock output frequency =  $T3 \text{ overflow} / 2$

If T3\_C/T = 0, namely Timer/Counter 3 count on the internal system clock,

When T3 in 1T mode (T4T3.1/T3x12=1), the output frequency =  $(\text{SYSclk}) / (65536 - [\text{RL\_TH3}, \text{RL\_TL3}]) / 2$

When T3 in 12T mode (T4T3.1/T3x12=0), the output frequency =  $(\text{SYSclk}) / 12 / (65536 - [\text{RL\_TH3}, \text{RL\_TL3}]) / 2$

If T3\_C/T = 1, namely Timer/Counter 3 count on the external pulse input from P0.5/T3,

the output frequency =  $(T3\_Pin\_CLK) / (65536 - [\text{RL\_TH3}, \text{RL\_TL3}]) / 2$

- 0, P0.4 is not configure for Timer 3 programmable clock output T3CLKO

### 2.1.3.2 Master Clock Output and Demo Program(C and ASM)

The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator. The speed of external programmable clock output of 5V MCU is also not more than 13.5MHz, because the output speed of I/O port of STC15 series 5V MCU is not more than 13.5MHz. The speed of external programmable clock output of 3.3V MCU is also not more than 8MHz, because the output speed of I/O port of STC15 series 3.3V MCU is not more than 8MHz.

CLK\_DIV (PCON2) : Clock Division Register (Non bit-addressable)

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CLK_DIV (PCON2)	97H	name	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0

INT\_CLKO (AUXR2) : External Interrupt Enable and Clock Output register

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name		EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO

How to output clock by using MCLKO/P5.4 or MCLKO\_2/XTAL2/P1.6.

The clock output of MCLKO/P5.4 or MCLKO\_2/XTAL2/P1.6 is controlled by the bits MCKO\_S2 and MCKO\_S1 and MCKO\_S0 of register CLK\_DIV. MCLKO/P5.4 or MCLKO\_2/XTAL2/P1.6 can be configured for master clock output whose frequency also can be choose by setting MCKO\_S2 (INT\_CLKO.3) and MCKO\_S1 (CLK\_DIV.7) and MCKO\_S0 (CLK\_DIV.6).

MCKO_S2	MCKO_S1	MCKO_S0	the control bit of master clock output by dividing the frequency (The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator)
0	0	0	Master clock do not output external clock
0	0	1	Master clock output external clock but its frequency do not be divided and the output clock frequency = MCLK / 1
0	1	0	Master clock output external clock but its frequency is divided by 2 and the output clock frequency = MCLK / 2
0	1	1	Master clock output external clock but its frequency is divided by 4 and the output clock frequency = MCLK / 4
1	0	0	Master clock output external clock but its frequency is divided by 4 and the output clock frequency = MCLK / 16

The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator.  
MCLK is the frequency of master clock.

STC15W4K32S4 series MCU output master clock on MCLKO/P5.4

**The speed of external programmable clock output of 5V MCU is also not more than 13.5MHz, because the output speed of I/O port of STC15 series 5V MCU is not more than 13.5MHz.**

**The speed of external programmable clock output of 3.3V MCU is also not more than 8MHz, because the output speed of I/O port of STC15 series 3.3V MCU is not more than 8MHz.**



---

the following is the demo program of Master clock output:

## 1. C Program Listing

```
/*-----*/
/* --- Exam Program of Master clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

typedef unsigned char    BYTE;
typedef unsigned int     WORD;

#define FOSC    18432000L
//-----
sfr    CLK_DIV    =    0x97;        //Clock divider register
sfr    INT_CLKO   =    0x8f;        //External Interrupt Enable and Clock Output register

//-----

void main()
{
    CLK_DIV    =    0x40;        //0100,0000 the output frequency of P5.4 is SYSclk
    INT_CLKO   =    0x00;

//    CLK_DIV    =    0x80;        //1000,0000 the output frequency of P5.4 is SYSclk/2
//    INT_CLKO   =    0x00;

//    CLK_DIV    =    0xC0;        //1100,0000 the output frequency of P5.4 is SYSclk/4
//    INT_CLKO   =    0x00;

//    CLK_DIV    =    0x00;        //0000,0000
//    INT_CLKO   =    0x08;        //0000,1000 the output frequency of P5.4 is SYSclk/16

    while (1);
}
```

---

## 2. Assembler Listing

```
/*-----*/
/* --- Exam Program of Master clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

CLK_DIV      DATA  97H                //Clock divider register
INT_CLKO     DATA  8FH;              //External Interrupt Enable and Clock Output register

;-----
;interrupt vector table

      ORG      0000H
      LJMP    MAIN
;-----

      ORG      0100H
MAIN:  MOV     SP,          #3FH        //initial SP

      MOV     CLK_DIV,    #40H        //0100,0000 the output frequency of P5.4 is SYSclk
      MOV     INT_CLKO,   #00H

//      MOV     CLK_DIV,    #80H        //1000,0000 the output frequency of P5.4 is SYSclk/2
//      MOV     INT_CLKO,   #00H

//      MOV     CLK_DIV,    #C0H        //1100,0000 the output frequency of P5.4 is SYSclk/4
//      MOV     INT_CLKO,   #00H

//      MOV     CLK_DIV,    #00H        //0000,0000
//      MOV     INT_CLKO,   #08H        //0000,1000 the output frequency of P5.4 is SYSclk/16

      SJMP   $

//-----
      END
```

### 2.1.3.3 Timer 0 Programmable Clock Output and Demo Program(C and ASM)

INT\_CLKO (AUXR2) : External Interrupt Enable and Clock Output register

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name		EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO

How to output clock by using T0CLKO/P3.5.

The clock output of T0CLKO/P3.5 is controlled by the bit T0CLKO of register INT\_CLKO (AUXR2).

INT\_CLKO .0 - T0CLKO : 1, enable T0 clock output  
0, disable T0 clock output

The output clock frequency of T0CLKO is controlled by Timer 0. When it is used as programmable clock output, Timer 0 must work in **mode 0 (16-bit auto-reload timer/counter)** or **mode 2 (8-bit auto-reload timer/counter)** and don't enable its interrupt to avoid CPU entering interrupt repeatedly unless special circumstances.

When T0CLKO/INT\_CLKO.0=1, P3.5/T1 is configured for Timer0 programmable clock output T0CLKO.

The clock output frequency =  $T0 \text{ overflow} / 2$

If Timer/Counter 0 in mode 0 (16 bit auto-reloadable mode),

and if  $C/\bar{T} = 0$ , namely Timer/Counter 0 count on the internal system clock,

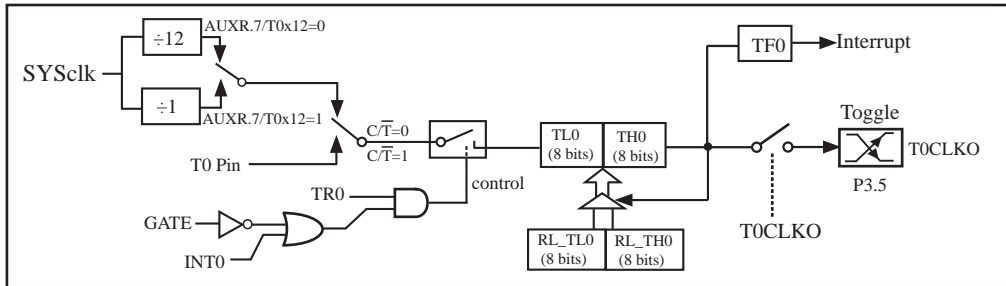
When T0 in 1T mode (AUXR.7/T0x12=1), the output frequency =  $(SYSclk) / (65536 - [RL\_TH0, RL\_TL0]) / 2$

When T0 in 12T mode (AUXR.7/T0x12=0), the output frequency =  $(SYSclk) / 12 / (65536 - [RL\_TH0, RL\_TL0]) / 2$

and if  $C/\bar{T} = 1$ , namely Timer/Counter 0 count on the external pulse input from P3.4/T0,

the output frequency =  $(T0\_Pin\_CLK) / (65536 - [RL\_TH0, RL\_TL0]) / 2$

RL\_TH0 is the reloaded register of TH0, RL\_TL0 is the reload register of TL0.



Timer/Counter 0 mode 0: 16 bit auto-reloadable mode

When T0CLKO/INT\_CLKO.0=1, P3.5/T1 is configured for Timer 0 programmable clock output T0CLKO.

The clock output frequency =  $T0 \text{ overflow} / 2$

If Timer/Counter 0 in mode 2 (8 bit auto-reloadable mode),

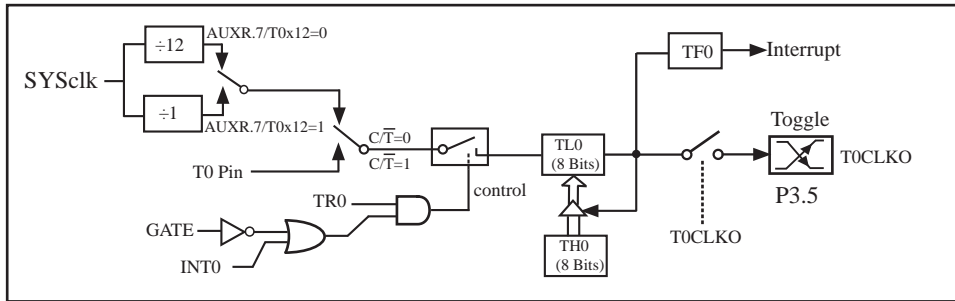
and if  $C/\bar{T} = 0$ , namely Timer/Counter 0 count on the internal system clock,

When T0 in 1T mode(AUXR.7/T0x12=1), the output frequency =  $(SYSclk) / (256 - TH0) / 2$

When T0 in 12T mode(AUXR.7/T0x12=0), the output frequency =  $(SYSclk) / 12 / (256 - TH0) / 2$

and if  $C/\bar{T} = 1$ , namely Timer/Counter 0 count on the external pulse input from P3.4/T0,

the output frequency =  $(T0\_Pin\_CLK) / (256 - TH0) / 2$



Timer/Counter 0 mode 2: 8 bit auto-reloadable mode

The following is the example program that Timer 0 output programmable clock by dividing the frequency of internal system clock or the clock input from external pin T0/P3.4 (C and assembly):

## 1. C Program Listing

```

/*-----*/
/* --- Exam Program of Timer 0 porgrammable clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

#define FOSC 18432000L

//-----
sfr    AUXR      =    0x8e;
sfr    INT_CLKO  =    0x8f;

sbit   T0CLKO   =    P3^5;

#define F38_4KHz (65536-FOSC/2/38400)           //1T Mode
//#define F38_4KHz (65536-FOSC/2/12/38400)      //12T Mode

```

---

```

//-----
void main()
{
    AUXR   |=   0x80;           //Timer 0 in 1T mode
//    AUXR   &=   ~0x80;       //Timer 0 in 12T mode

    TMOD   =   0x00;           //set Timer0 in mode 0(16 bit auto-reloadable mode)

    TMOD   &=   ~0x04;         //C/T0=0, count on internal system clock
//    TMOD   |=   0x04;         //C/T0=1, count on external pulse input from T0 pin

    TL0    =   F38_4KHz;       //Initial timing value
    TH0    =   F38_4KHz >> 8;
    TR0    =   1;
    INT_CLKO =   0x01;

    while (1);
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- Exam Program of Timer 0 porgrammable clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

AUXR      DATA  08EH
INT_CLKO  DATA  08FH

T0CLKO    BIT    P3.5

F38_4KHz  EQU    0FF10H      //38.4KHz(1T mode, 65536-18432000/2/38400)
//F38_4KHz EQU    0FFECH      //38.4KHz(12T mode,(65536-18432000/2/12/38400)
//-----

```

---

```

    ORG    0000H
    LJMP   MAIN

//-----
    ORG    0100H
MAIN:
    MOV    SP,    #3FH

    ORL    AUXR, #80H           //Timer 0 in 1T mode
//    ANL    AUXR, #7FH           //Timer 0 in 12T mode

    MOV    TMOD, #00H           //set Timer0 in mode 0(16 bit auto-reloadable mode)

    ANL    TMOD, #0FBH         //C/T0=0, count on internal system clock
//    ORL    TMOD, #04H         //C/T0=1, count on external pulse input from T0 pin

    MOV    TL0,    #LOW F38_4KHz //Initial timing value
    MOV    TH0,    #HIGH F38_4KHz
    SETB   TR0
    MOV    INT_CLKO,    #01H

    SJMP   $

;-----

    END

```

---

### 2.1.3.4 Timer 1 Programmable Clock Output and Demo Program(C and ASM)

INT\_CLKO (AUXR2) : External Interrupt Enable and Clock Output register

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name		EX4	EX3	EX2	MCKO_S2	T2CLKO	<b>T1CLKO</b>	T0CLKO

How to output clock by using T1CLKO/P3.4.

The clock output of T1CLKO/P3.4 is controlled by the bit T1CLKO of register INT\_CLKO (AUXR2).

INT\_CLKO.1 - T1CLKO                    1, enable T1 clock output  
  0, disable T1 clock output

The output clock frequency of T1CLKO is controlled by Timer 1. When it is used as programmable clock output, Timer 1 must work in mode 1 (16-bit auto-reload timer/counter) or mode 2 (8-bit auto-reload timer/counter) and don't enable its interrupt to avoid CPU entering interrupt repeatedly unless special circumstances.

When T1CLKO/INT\_CLKO.1=1, P3.4/T0 is configured for Timer 1 programmable clock output T1CLKO.

The clock output frequency = T1 overflow/2

If Timer/Counter 1 in mode 1 (16 bit auto-reloadable mode),

and if  $C/\bar{T} = 0$ , namely Timer/Counter 1 count on the internal system clock,

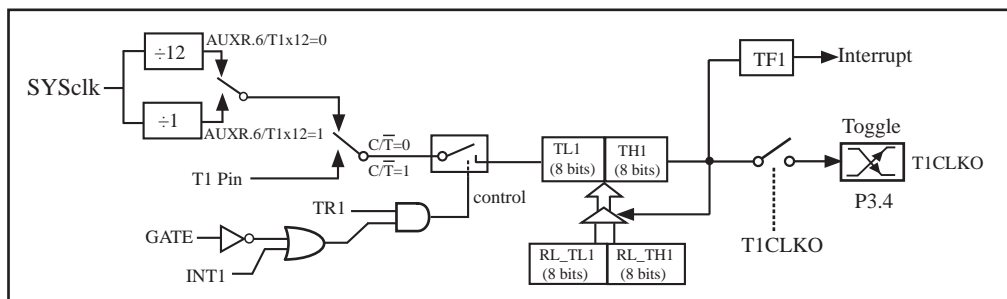
When T1 in 1T mode (AUXR.6/T1x12=1), the output frequency =  $(SYSclk)/(65536-[RL\_TH1, RL\_TL1])/2$

When T1 in 12T mode (AUXR.6/T1x12=0), the output frequency =  $(SYSclk) / 12 / (65536-[RL\_TH1, RL\_TL1])/2$

and if  $C/\bar{T} = 1$ , namely Timer/Counter 1 count on the external pulse input from P3.5/T1,

the output frequency =  $(T1\_Pin\_CLK) / (65536-[RL\_TH1, RL\_TL1])/2$

RL\_TH1 is the reloaded register of TH1, RL\_TL1 is the reload register of TL1.



Timer/Counter 1 mode 0: 16 bit auto-reloadable mode

When T1CLKO/INT\_CLKO.1=1, P3.4/T0 is configured for Timer 1 programmable clock output T1CLKO.

The clock output frequency = T1 overflow/2

If Timer/Counter 1 in mode 2 (8 bit auto-reloadable mode),

and if  $C/\bar{T} = 0$ , namely Timer/Counter 1 count on the internal system clock,

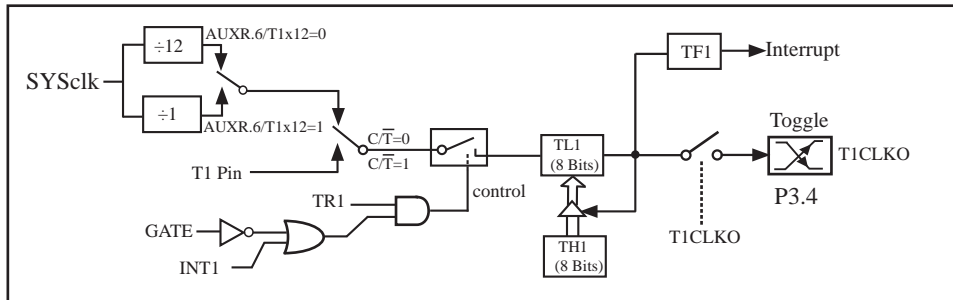
When T1 in 1T mode(AUXR.6/T1x12=1), the output frequency =  $(SYSclk) / (256-TH1) / 2$

When T1 in 12T mode(AUXR.6/T1x12=0), the output frequency =  $(SYSclk) / 12 / (256-TH1) / 2$

and if  $C/\bar{T} = 1$ , namely Timer/Counter 1 count on the external pulse input from P3.5/T1,

the output frequency =  $(T1\_Pin\_CLK) / (256-TH1) / 2$

RL\_TH1 is the reloaded register of TH1, RL\_TL1 is the reload register of TL1.



Timer/Counter 1 mode 2: 8 bit auto-reloadable mode

The following is the example program that Timer 1 output programmable clock by dividing the frequency of internal system clock or the clock input from external pin T1/P3.5 (C and assembly):

## 1. C Program Listing

```

/*-----*/
/* --- Exam Program of Timer 1 programmable clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

typedef unsigned char    BYTE;
typedef unsigned int     WORD;

#define FOSC 18432000L

//-----
sfr AUXR      = 0x8e;
sfr INT_CLKO  = 0x8f;

sbit T1CLKO   = P3^4;

#define F38_4KHz (65536-FOSC/2/38400)           //1T Mode
//#define F38_4KHz (65536-FOSC/2/12/38400)      //12T Mode

```



---

```

//-----
void main()
{
    AUXR  |=    0x40;           //Timer 1 in 1T mode
//    AUXR  &=   ~0x40;       //Timer 1 in 12T mode

    TMOD  =    0x00;           //set Timer 1 in mode 0(16 bit auto-reloadable mode)

    TMOD  &=   ~0x40;         //C/T1=0, count on internal system clock
//    TMOD  |=    0x40;       //C/T1=1, count on external pulse input from T1 pin

    TL1   =    F38_4KHz;       //Initial timing value
    TH1   =    F38_4KHz >> 8;
    TR1   =    1;
    INT_CLKO =    0x02;

    while (1);
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- Exam Program of Timer 1 programmable clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

AUXR      DATA  08EH
INT_CLKO  DATA  08FH

T1CLKO    BIT    P3.4
F38_4KHz  EQU    0FF10H      //38.4KHz(1T mode, 65536-18432000/2/38400)
//F38_4KHz EQU    0FFECH      //38.4KHz(12T mode, (65536-18432000/2/12/38400)

```

---

```

    ORG    0000H
    LJMP   MAIN

//-----
    ORG    0100H
MAIN:
    MOV    SP,    #3FH

    ORL    AUXR, #40H           //Timer 1 in 1T mode
//    ANL    AUXR, #0BFH       //Timer 1 in 12T mode

    MOV    TMOD, #00H         //set Timer 1 in mode 0(16 bit auto-reloadable mode)

//    ANL    TMOD, #0BFH       //C/T1=0, count on internal system clock
    ORL    TMOD, #40H         //C/T1=1, count on external pulse input from T1 pin

    MOV    TL1,    #LOW F38_4KHz //Initial timing value
    MOV    TH1,    #HIGH F38_4KHz
    SETB   TR1
    MOV    INT_CLKO, #02H

    SJMP   $

;-----

    END

```

### 2.1.3.5 Timer 2 Programmable Clock Output and Demo Program (C and ASM)

INT\_CLKO (AUXR2) : External Interrupt Enable and Clock Output register

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name		EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	TOCLKO

How to output clock by using T2CLKO/P3.0.

The clock output of T2CLKO/P3.0 is controlled by the bit T2CLKO of register INT\_CLKO (AUXR2).

INT\_CLKO.2 - T2CLKO :            1, enable T2 clock output  
  0, disable T2 clock output

The output clock frequency of T2CLKO is controlled by Timer 2. When it is used as programmable clock output, Timer 2 interrupt don't be enabled to avoid CPU entering interrupt repeatedly unless special circumstances.

When T2CLKO/INT\_CLKO.2=1, P3.0 is configured for Timer 2 programmable clock output T2CLKO.

The clock output frequency =  $T2 \text{ overflow} / 2$

If  $T2\_C\bar{T} = 0$ , namely Timer/Counter 2 count on the internal system clock,

When T2 in 1T mode (AUXR.2/T2x12=1), the output frequency =  $(SYSclk) / (65536 - [RL\_TH2, RL\_TL2]) / 2$

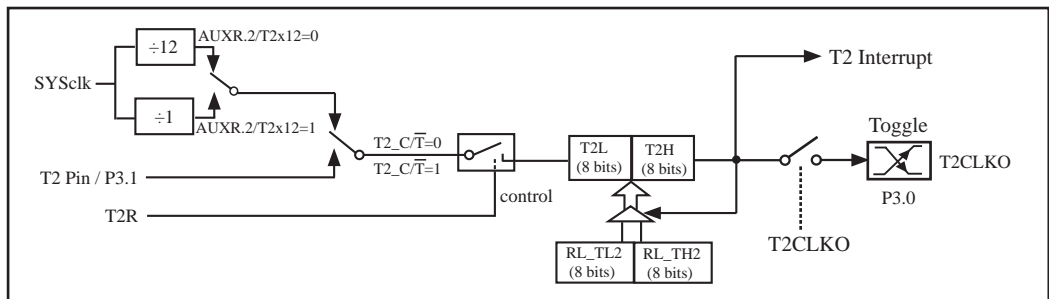
When T2 in 12T mode (AUXR.2/T2x12=0), the output frequency =  $(SYSclk) / 12 / (65536 - [RL\_TH2, RL\_TL2]) / 2$

If  $T2\_C\bar{T} = 1$ , namely Timer/Counter 2 count on the external pulse input from P3.1/T2,

the output frequency =  $(T2\_Pin\_CLK) / (65536 - [RL\_TH2, RL\_TL2]) / 2$

RL\_TH2 is the reloaded register of T2H, RL\_TL2 is the reload register of T2L.

Internal Structure Diagram of Timer 2 is shown below:



Timer / Counter 2 Operating Mode : 16 bit auto-reloadable Mode

---

The following is the example program that Timer 2 output programmable clock by dividing the frequency of internal system clock or the clock input from external pin T2/P3.1 (C and assembly):

## 1. C Program Listing

```
/*-----*/
/* --- Exam Program of Timer 2 porgrammable clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

typedef unsigned char    BYTE;
typedef unsigned int     WORD;

#define FOSC 18432000L

//-----

sfr    AUXR          = 0x8e;
sfr    INT_CLKO     = 0x8f;
sfr    T2H          = 0xD6;
sfr    T2L          = 0xD7;

sbit   T2CLKO       = P3^0;

#define F38_4KHz     (65536-FOSC/2/38400)           //1T mode
//#define F38_4KHz     (65536-FOSC/2/12/38400)      //12T mode

//-----

void main()
{
    AUXR  |= 0x04;           //Timer 2 in 1T mode
    //    AUXR  &= ~0x04;      //Timer 2 in 12T mode
```

---

```

    AUXR  &=    ~0x08;           //T2_C/T=0, count on internal system clock
//    AUXR  |=    0x08;           //T2_C/T=1, count on external pulse input from T2(P3.1) pin
    T2L   =    F38_4KHz;           //Initial timing value
    T2H   =    F38_4KHz >> 8;

    AUXR  |=    0x10;
    INT_CLKO =    0x04;

    while (1);
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- Exam Program of Timer 2 porgrammable clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

AUXR      DATA  08EH
INT_CLKO  DATA  08FH
T2H       DATA  0D6H
T2L       DATA  0D7H

T2CLKO    BIT    P3.0

F38_4KHz  EQU    0FF10H           //38.4KHz(1T mode, 65536-18432000/2/38400)
//F38_4KHz EQU    0FFECH           //38.4KHz(12T mode, (65536-18432000/2/12/38400)

//-----

```

---

```

    ORG    0000H
    LJMP   MAIN

//-----

    ORG    0100H
MAIN:
    MOV    SP,    #3FH

    ORL    AUXR,  #04H           //Timer 2 in 1T mode
//    ANL    AUXR,  #0FBH       //Timer 2 in 12T mode

    ANL    AUXR,  #0F7H         //T2_C/T=0, count on internal system clock
//    ORL    AUXR,  #08H         //T2_C/T=1, count on external pulse input from T2(P3.1) pin

    MOV    T2L,   #LOW F38_4KHz   //Initial timing value
    MOV    T2H,   #HIGH F38_4KHz
    ORL    AUXR,  #10H
    MOV    INT_CLKO, #04H

    SJMP   $

;-----

    END

```

### 2.1.3.6 Timer 3 Programmable Clock Output and Demo Program (C and ASM)

T4T3M : Timer 4 and Timer 3 Mode register (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

How to output clock by using T3CLKO/P0.4.

The clock output of T3CLKO/P0.4 is controlled by the bit T3CLKO of register T4T3M.

T4T3M.0 - T3CLKO :     1, enable T3 clock output  
                           0, disable T3 clock output

The output clock frequency of T3CLKO is controlled by Timer 3. When it is used as programmable clock output, Timer 3 interrupt don't be enabled to avoid CPU entering interrupt repeatedly unless special circumstances.

When T3CLKO/T4T3M.0=1, P0.4 is configured for Timer 3 programmable clock output T3CLKO.

The clock output frequency =  $T3 \text{ overflow} / 2$

If  $T3\_C/T = 0$ , namely Timer/Counter 3 count on the internal system clock,

When T3 in 1T mode ( $T4T3.1/T3x12=1$ ), the output frequency =  $(SYSclk)/(65536-[RL\_TH3, RL\_TL3])/2$

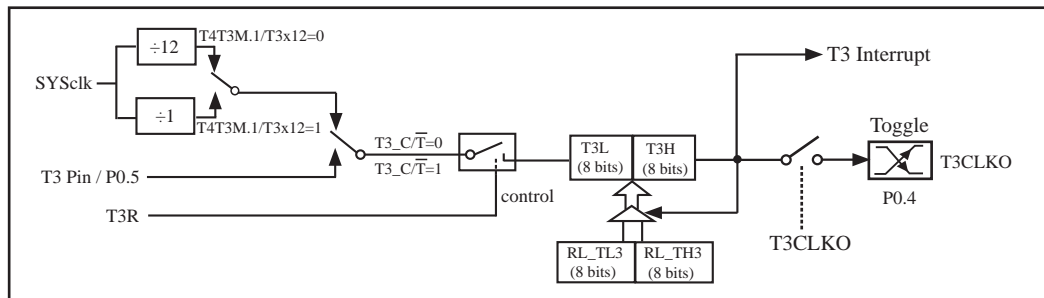
When T3 in 12T mode ( $T4T3.1/T3x12=0$ ), the output frequency =  $(SYSclk) / 12 / (65536-[RL\_TH3, RL\_TL3])/2$

If  $T3\_C/T = 1$ , namely Timer/Counter 3 count on the external pulse input from P0.5/T3,

the output frequency =  $(T3\_Pin\_CLK) / (65536-[RL\_TH3, RL\_TL3])/2$

RL\_TH3 is the reloaded register of T3H, RL\_TL3 is the reload register of T3L.

Internal Structure Diagram of Timer 3 is shown below:



Timer / Counter 3 Operating Mode : 16 bit auto-reloadable Mode

### 2.1.3.7 Timer 4 Programmable Clock Output and Demo Program (C and ASM)

T4T3M : Timer 4 and Timer 3 Mode register (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

How to output clock by using T4CLKO/P0.6.

The clock output of T4CLKO/P0.6 is controlled by the bit T4CLKO of register T4T3M.

T4T3M.4 - T4CLKO :     1, enable clock output  
                           0, disable clock output

The output clock frequency of T4CLKO is controlled by Timer 4. When it is used as programmable clock output, Timer 4 interrupt don't be enabled to avoid CPU entering interrupt repeatedly unless special circumstances.

When T4CLKO/T4T3M.4=1, P0.6 is configured for Timer 4 programmable clock output T4CLKO.

The clock output frequency =  $T4 \text{ overflow} / 2$

If T4\_C/T = 0, namely Timer/Counter 4 count on the internal system clock,

When T4 in 1T mode (T4T3.5/T4x12=1), the output frequency =  $(SYSclk) / (65536 - [RL\_TH4, RL\_TL4]) / 2$

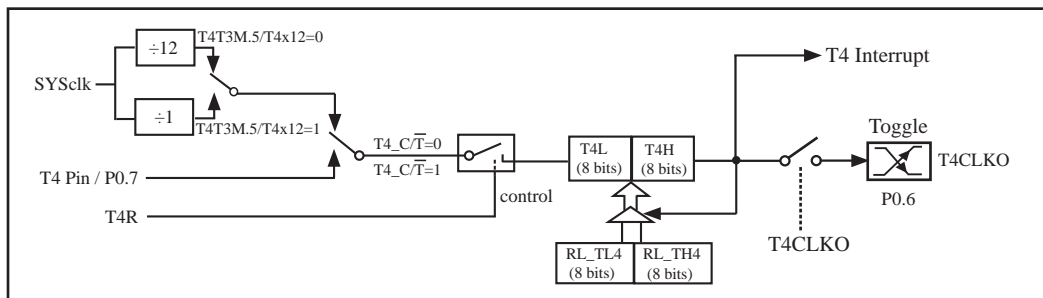
When T4 in 12T mode (T4T3.5/T4x12=0), the output frequency =  $(SYSclk) / 12 / (65536 - [RL\_TH4, RL\_TL4]) / 2$

If T4\_C/T = 1, namely Timer/Counter 4 count on the external pulse input from P0.7/T4,

the output frequency =  $(T4\_Pin\_CLK) / (65536 - [RL\_TH4, RL\_TL4]) / 2$

RL\_TH4 is the reloaded register of T4H, RL\_TL4 is the reload register of T4L.

Internal Structure Diagram of Timer 4 is shown below:



Timer / Counter 4 Operating Mode : 16 bit auto-reloadable Mode



---

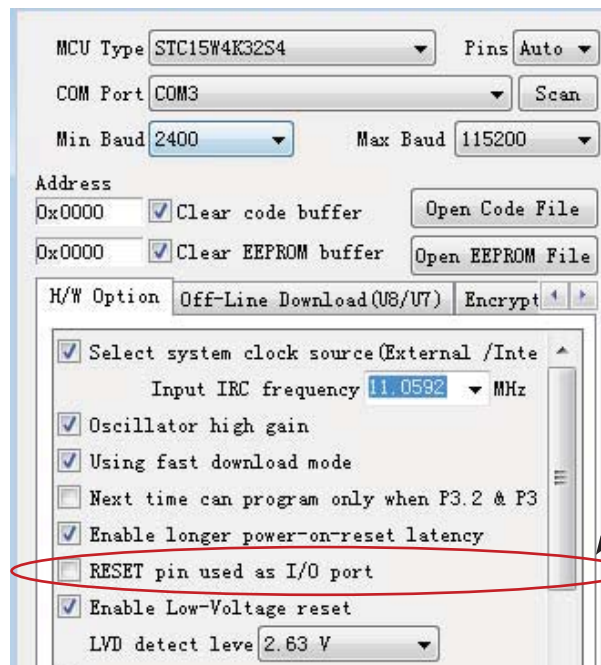
## 2.2 RESET Sources

There are 7 reset sources to generate a reset in STC15 series MCU. They are external RST pin reset, software reset, On-chip power-off / power-on reset(if delay 180mS after power-off / power-on reset, the reset mode is On-chip MAX810 special reset which actually add 180mS delay after power-off / power-on reset), internal low-voltage detection reset, MAX810 special circuit reset, Watch-Dog-Timer reset and the reset caused by illegal use of program address.

### 2.2.1 External RST pin Reset

The STC15W4K32S4 series MCU is on RST/P5.4. Now take RST/P5.4 for example to introducing the external RST pin reset.

External RST pin reset accomplishes the MCU reset by forcing a reset pulse to RST pin from external. The P5.4/RST pin at factory is as I/O port (default). If users need to configure it as reset function pin , they may enable the corresponding option in STC-ISP Writer/Programmer shown the following figure. If P5.4/RST pin has been configured as external reset pin, it will be as reset function pin which is the input to Schmitt Trigger and input pin for chip reset. Asserting an active-high signal and keeping at least 24 cycles plus 20us on the RST pin generates a reset. If the signal on RST pin changed active-low level, MCU will end the reset state and set the bit SWBS/IAP\_CONTR.6 and start to run from the system ISP monitor program area. External RST pin reset is hard reset of warm boot.



What part the RESET pin play  
Choice : RESET pin behaves as I/O pin  
No-Choice: RESET pin behaves as reset pin

## 2.2.2 Software Reset and Demo Program (C and ASM)

Users may need to achieve MCU system soft reset (one of the soft reset or warm boot reset) in the running process of user application program sometimes. Due to the hardware of traditional does not support this feature, the user must use software to realize with more trouble. Now to achieve the function, the register IAP\_CONTR is added according to the requirement of customer in STC new series. Users only need to control the two bits SWBS/SWRST in register IAP\_CONTR. Writing an “1” to SWRST bit in IAP\_CONTR register will generate an internal reset. SWBS bit decide where the program strat to run from after reset.

IAP\_CONTR: ISP/IAP Control Register

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	name	IAPEN	SWBS	SWRST	CMD_FAIL	-	WT2	WT1	WT0

IAPEN : ISP/IAP operation enable.

0 : Global disable all ISP/IAP program/erase/read function.

1 : Enable ISP/IAP program/erase/read function.

SWBS: software boot selection control bit

0 : Boot from main-memory after reset.

1 : Boot from ISP memory after reset.

SWRST: software reset trigger control.

0 : No operation

1 : Generate software system reset. It will be cleared by hardware automatically.

CMD\_FAIL: Command Fail indication for ISP/IAP operation.

0 : The last ISP/IAP command has finished successfully.

1 : The last ISP/IAP command fails. It could be caused since the access of flash memory was inhibited.

[;Software reset from user application program area \(AP area\) and switch to AP area to run program](#)

```
MOV IAP_CONTR, #00100000B ;SWBS = 0(Select AP area), SWRST = 1(Software reset)
```

[;Software reset from system ISP monitor program area \(ISP area\) and switch to AP area to run program](#)

```
MOV IAP_CONTR, #00100000B ;SWBS = 0(Select AP area), SWRST = 1(Software reset)
```

[;Software reset from user application program area \(AP area\) and switch to ISP area to run program](#)

```
MOV IAP_CONTR, #01100000B ;SWBS = 1(Select ISP area), SWRST = 1(Software reset)
```

[;Software reset from system ISP monitor program area \(ISP area\) and switch to ISP area to run program](#)

```
MOV IAP_CONTR, #01100000B ;SWBS = 1(Select ISP area), SWRST = 1(Software reset)
```

This reset is to reset the whole system, all special function registers and I/O prots will be reset to the initial value

---

## 1. C Program Listing

```
/*-----*/
/* --- Exam Program of software reset -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

//-----
sfr    IAP_CONTR = 0xc7;           //IAP Control register
sbit   P10      =    P1^0;
//-----

void delay()                       //software delay
{
    int i;
    for (i=0; i<10000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}
void main()
{
    P10 = !P10;
    delay();
    P10 = !P10;
    delay();

    IAP_CONTR = 0x20;           //softwate reset, strat to run from user application program area
//    IAP_CONTR = 0x60;           //softwate reset, strat to run from system ISP monitor program area

    while (1);
}
```

---

## 2. Assembler Listing

```
/*-----*/
/* --- Exam Program of software reset -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

IAP_CONTR    DATA    0C7H
//-----
        ORG    0000H
        LJMP   MAIN
//-----
        ORG    0100H
MAIN:
        MOV    SP,    #3FH

        CPL    P1.0
        LCALL  DELAY
        CPL    P1.0
        LCALL  DELAY

        MOV    IAP_CONTR,    #20H           //softwate reset,
                                           //strat to run from user appliction program area
//        MOV    IAP_CONTR,    #60H           //softwate reset,
                                           //strat to run from system ISP monitor program area

        JMP    $
;-----
DELAY:
        MOV    R0,    #0                   //software delay
        MOV    R1,    #0
WAIT:
        DJNZ   R0,    WAIT
        DJNZ   R1,    WAIT
        RET
;-----
        END
```

## 2.2.3 Power-Off / Power-On Reset (POR)

When VCC drops below the detection threshold of POR circuit, all of the logic circuits are reset.

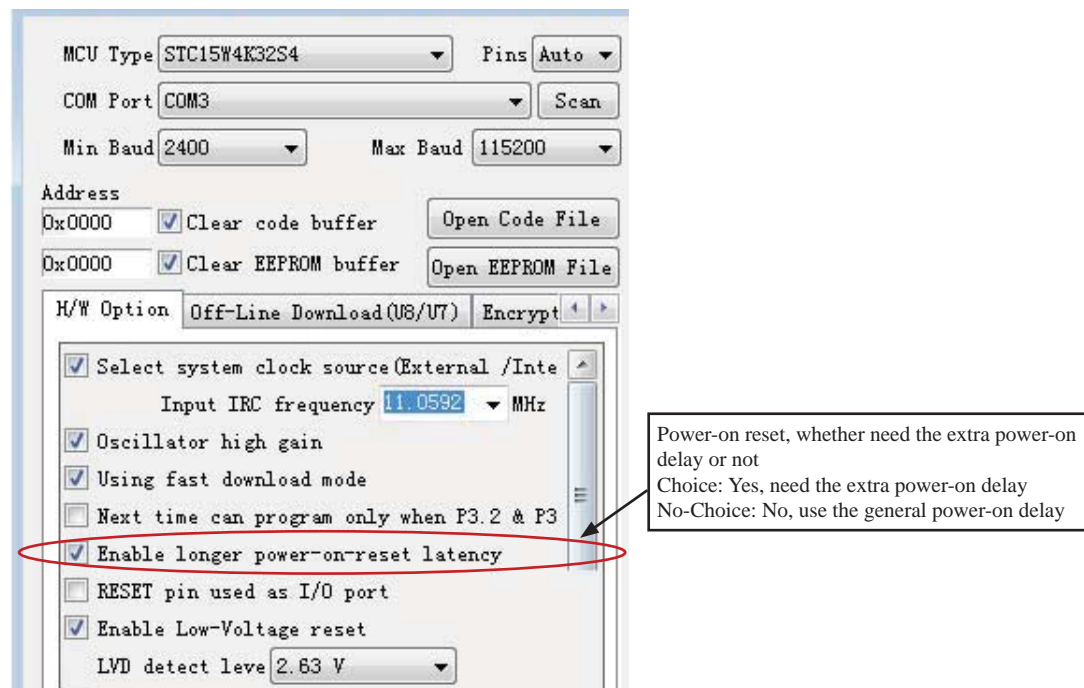
When VCC goes back up again, an internal reset is released automatically after a delay of 32768 clocks. After power-off / power-on reset, MCU will set the bit SWBS/IAP\_CONTR.6 and start to run from the system ISP monitor program area. power-off / power-on reset is one of cold boot reset.

The nominal POR detection threshold is around 1.8V for 3.3V device and 3.2V for 5V device.

The Power-Off / Power-On flag, POF/PCON.4, is set by hardware to denote the VCC power has ever been less than the POR voltage. And, it helps users to check if the start of running of the CPU is from power-on or from hardware reset (such as RST-pin reset), software reset or Watchdog Timer reset. The POF bit should be cleared by software.

## 2.2.4 MAX810 Speical Circuit Reset (Power-Off/ Power-On Reset Delay)

There is another on-chip POR delay circuit s integrated on STC15 series MCU. This circuit is MAX810—speical reset circuit and is controlled by configuring STC-ISP Writer/Programmer shown in the next figure. MAX810 special reset circuit just generate about 180mS extra reset-delay-time after power-off / power-on reset. So it is another power-off / power-on reset. After the reset is released, MCU will set the bit SWBS/IAP\_CONTR.6 and start to run from the system ISP monitor program area. MAX810 special circuit reset is one of cold boot reset.



## 2.2.5 Internal Low Voltage Detection Reset

Besides the POR voltage, there is a higher threshold voltage: the Low Voltage Detection (LVD) voltage for STC15W4K32S4 series MCU. If user have enabled low-voltage reset in STC-ISP Writer/Programmer, it will generate a reset when the VCC power drops down to the LVD voltage. And the Low voltage Flag, LVDF bit (PCON.5), will be set by hardware simultaneously. (Note that during power-on, this flag will also be set, and the user should clear it by software for the following Low Voltage detecting.) Internal low-voltage detection reset don't set the bit SWBS/IAP\_CONTR.6. If the bit SWBS/IAP\_CONTR.6 has been set as 0 before reset, MCU will start to run from the user application program area after reset. If the bit SWBS/IAP\_CONTR.6 has been set as 1 before reset, MCU will start to run from the system ISP monitor program area after reset on the contrary. Internal low-voltage detection reset is one of hard reset of warm boot.

The threshold voltage of STC15W4K32S4 series MCU built-in low voltage detection reset is optional in STC-ISP Writer/Programmer. see the following figure.

The low-voltage detector parameter of STC15W4K32S4 series MCU shown in following figure is optional :

Enable Low-Voltage Reset, controls reset or not while the Low-Voltage event  
Choice: Reset while detect a low-voltage  
No-Choice: Interrupt while detect a low-voltage

The low-voltage detector parameter adjust the thresh voltage level of the built-in low-voltage detector.

When the oscillator frequency is between 4M ~ 24MHz, low-voltage detection threshold voltage is recommended to choose more than 2.62V.

When the oscillator frequency is between 25M ~ 35MHz, low-voltage detection threshold voltage is recommended to choose more than 2.79V.

Optional reset threshold voltage of STC15W4K32S4 series MCU

Option	Value
Enable longer power-on-reset latency	<input checked="" type="checkbox"/>
RESET pin used as I/O port	<input type="checkbox"/>
Enable Low-Voltage reset	<input checked="" type="checkbox"/>
LVD detect level	2.63 V
Inhibit EEPROM	<input checked="" type="checkbox"/>
Select CPU-Core	2.23 V
Hardware enable	<input type="checkbox"/>
Watch-Dog-Timer	2.33 V
WDT stop count	2.50 V
Erase all EEPROM	<input type="checkbox"/>
P2.0 power-on	2.63 V
Switch UART1 [R	<input type="checkbox"/>
to [P3.6, P3.7],	2.78 V
	2.98 V
	3.06 V
	3.15 V
	3.25 V
	3.38 V
	3.50 V
	3.61 V
	3.76 V
	3.86 V
	3.96 V
	4.08 V

Download/Program Stop Re-Program

Check MCU Notice Delay 3 se

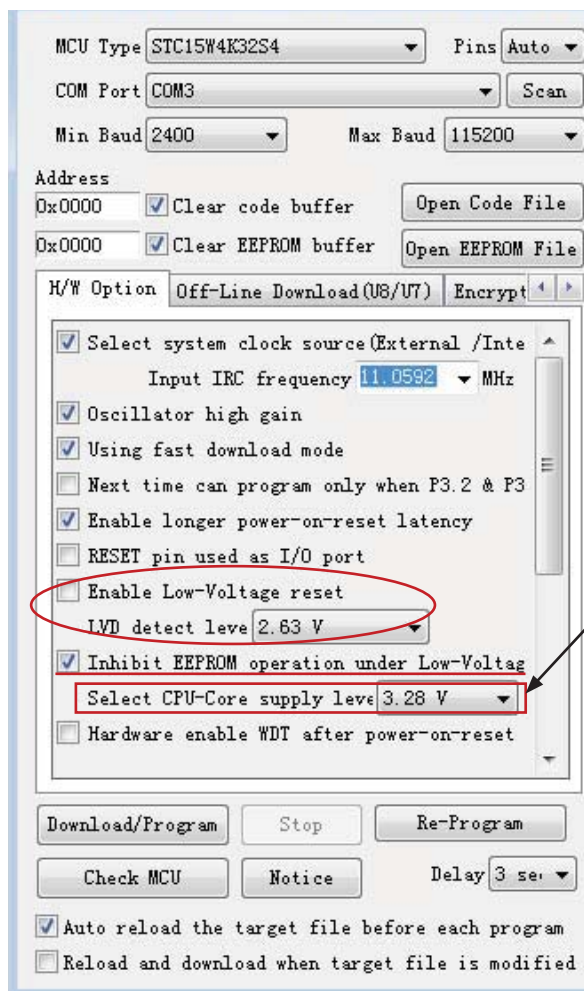
Auto reload the target file before each program  
 Reload and download when target file is modified

If low-voltage detection reset is not be enabled , in other words, low-voltage detection interrupt is enabled in STC-ISP Writer/Programmer, it will generate a interrupt when the VCC power drops down to the LVD voltage. And the Low voltage Flag, LVDF bit (PCON.5), will be set by hardware simultaneously.

The low voltage detection threshold voltage of STC15 series also is optional in STC-ISP Writer/Programmer. see the above figure too.

If internal low voltage detection interrupt function is needed to continue normal operation during stop/power-down mode, it can be used to wake up MCU from stop/power-down mode.

Don't enable EEPROM/IAP function when the operation voltage is too low. Namely, select the option "Inhibit EEPROM operation under Low-Voltage" in STC-ISP Writer/Programmer



Select CPU-Core supply level:  
1M~24M, recommend set to about 2.66V  
24M~28M, recommend set to about 3.32V  
28M~40M, recommend set to about 3.63V

---

Some SFRs related to Low voltage detection as shown below.

**PCON register** (Power Control Register)

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

LVDF : Pin Low-Voltage Flag. Once low voltage condition is detected (VCC power is lower than LVD voltage), it is set by hardware (and should be cleared by software).

**IE: Interrupt Enable Register**

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

Enable Bit = 1 enables the interrupt; Enable Bit = 0 disables it .

EA (IE.7): disables all interrupts. if EA = 0, no interrupt will be acknowledged. if EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

ELVD (IE.6): Low voltage detection interrupt enable bit.

**IP: Interrupt Priority Register**

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE	B8H	name	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0

PLVD : Low voltage detection interrupt priority control bits.

PLVD=0, Low voltage detection interrupt is assigned low priority.

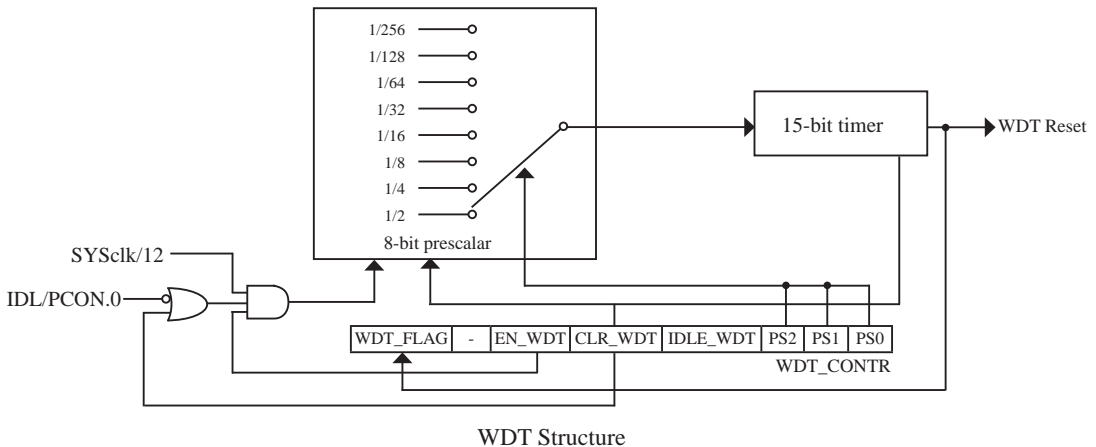
PLVD=1, Low voltage detection interrupt is assigned high priority.



## 2.2.6 Watch-Dog-Timer Reset

The watch dog timer in STC15 series MCU consists of an 8-bit pre-scaler timer and an 15-bit timer. The timer is one-time enabled by setting EN\_WDT(WDT\_CONTR.5). Clearing EN\_WDT can stop WDT counting. When the WDT is enabled, software should always reset the timer by writing 1 to CLR\_WDT bit before the WDT overflows. If STC15W4K32S4 series MCU is out of control by any disturbance, that means the CPU can not run the software normally, then WDT may miss the "writing 1 to CLR\_WDT" and overflow will come. An overflow of Watch-Dog-Timer will generate a internal reset.

Watch-Dog Timer (WDT) reset don't set the bit SWBS/IAP\_CONTR.6. If the bit SWBS/IAP\_CONTR.6 has been set as 0 before reset, MCU will start to run from the user application program area after reset. If the bit SWBS/IAP\_CONTR.6 has been set as 1 before reset, MCU will start to run from the system ISP monitor program area after reset on the contrary. WDT reset is one of soft reset of warm boot.



WDT\_CONTR: Watch-Dog-Timer Control Register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
WDT_CONTR	0C1H	name	WDT_FLAG	-	EN_WDT	CLR_WDT	IDLE_WDT	PS2	PS1	PS0

WDT\_FLAG : WDT reset flag.

0 : This bit should be cleared by software.

1 : When WDT overflows, this bit is set by hardware to indicate a WDT reset happened.

EN\_WDT : Enable WDT bit. When set, WDT is started.

CLR\_WDT : WDT clear bit. When set, WDT will recount. Hardware will automatically clear this bit.

IDLE\_WDT : WDT IDLE mode bit. When set, WDT is enabled in IDLE mode. When clear, WDT is disabled in IDLE.

PS2, PS1, PS0: WDT Pre-scale value set bit.

Pre-scale value of Watchdog timer is shown as the bellowed table :

PS2	PS1	PS0	Pre-scale	WDT overflow Time @20MHz
0	0	0	2	39.3 mS
0	0	1	4	78.6 mS
0	1	0	8	157.3 mS
0	1	1	16	314.6 mS
1	0	0	32	629.1 mS
1	0	1	64	1.25 S
1	1	0	128	2.5 S
1	1	1	256	5 S

The WDT overflow time is determined by the following equation:

$$\text{WDT overflow time} = (12 \times \text{Pre-scale} \times 32768) / \text{SYSclk}$$

The SYSclk is 20MHz in the table above.

If SYSclk is 12MHz, The WDT overflow time is :

$$\text{WDT overflow time} = (12 \times \text{Pre-scale} \times 32768) / 12000000 = \text{Pre-scale} \times 393216 / 12000000$$

WDT overflow time is shown as the bellowed table when SYSclk is 12MHz:

PS2	PS1	PS0	Pre-scale	WDT overflow Time @12MHz
0	0	0	2	65.5 mS
0	0	1	4	131.0 mS
0	1	0	8	262.1 mS
0	1	1	16	524.2 mS
1	0	0	32	1.0485 S
1	0	1	64	2.0971 S
1	1	0	128	4.1943 S
1	1	1	256	8.3886 S

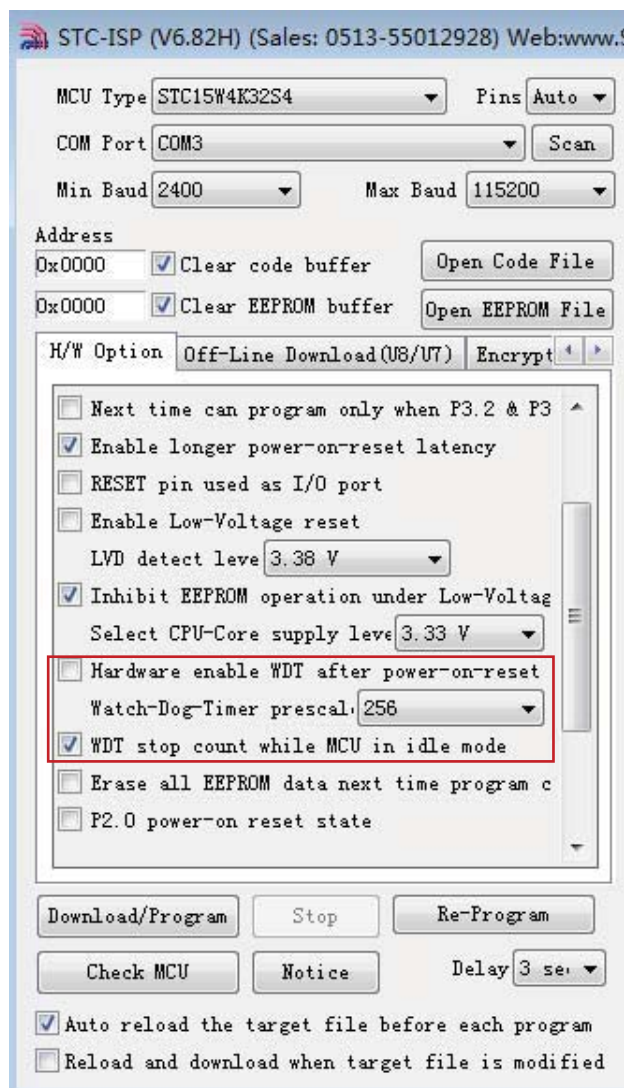
If SYSclk is 11.0592MHz, The WDT overflow time is :

$$\text{WDT overflow time} = (12 \times \text{Pre-scale} \times 32768) / 11059200 = \text{Pre-scale} \times 393216 / 11059200$$

WDT overflow time is shown as the bellowed table when SYSclk is 11.0592MHz:

PS2	PS1	PS0	Pre-scale	WDT overflow Time @11.0592MHz
0	0	0	2	71.1 mS
0	0	1	4	142.2 mS
0	1	0	8	284.4 mS
0	1	1	16	568.8 mS
1	0	0	32	1.1377 S
1	0	1	64	2.2755 S
1	1	0	128	4.5511 S
1	1	1	256	9.1022 S

Options related with WDT in STC-ISP Writer/Programmer is shown in the following figure



---

The following example is a assembly language program that demonstrates STC 1T Series MCU WDT.

```
;/*-----*/
;/* --- STC 1T Series MCU WDT Demo -----*/
;/* If you want to use the program or the program referenced in the -----*/
;/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
;/*-----*/

; WDT overflow time = (12 × Pre-scale × 32768) / SYSclk
WDT_CONTR      EQU    0C1H          ;WDT address
WDT_TIME_LED   EQU    P1.5          ;WDT overflow time LED on P1.5
                                   ;The WDT overflow time may be measured by the LED light time
WDT_FLAG_LED   EQU    P1.7          ;WDT overflow reset flag LED indicator on P1.7
Last_WDT_Time_LED_Status EQU    00H
                                   ;bit variable used to save the last stauts of WDT overflow time LED indicator

;WDT reset time , the SYSclk is 18.432MHz
;Pre_scale_Word EQU    00111100B    ;open WDT, Pre-scale value is 32, WDT overflow time=0.68S
;Pre_scale_Word EQU    00111101B    ;open WDT, Pre-scale value is 64, WDT overflow time=1.36S
;Pre_scale_Word EQU    00111110B    ;open WDT, Pre-scale value is 128, WDT overflow time=2.72S
;Pre_scale_Word EQU    00111111B    ;open WDT, Pre-scale value is 256, WDT overflow time=5.44S

                ORG    0000H
                AJMP   MAIN
                ORG    0100H

MAIN:
                MOV    A,      WDT_CONTR          ;detection if WDT reset
                ANL    A,      #10000000B
                JNZ    WDT_Reset
                ;WDT_CONTR.7=1, WDT reset, jump WDT reset subroutine
                ;WDT_CONTR.7=0, Power-On reset, cold start-up, the content of RAM is random
                SETB   Last_WDT_Time_LED_Status    ;Power-On reset
                CLR    WDT_TIME_LED                ;Power-On reset,open WDT overflow time LED
                MOV    WDT_CONTR, #Pre_scale_Word    ;open WDT
```

---

WAIT1:

```
        SJMP    WAIT1                ;wait WDT overflow reset
;WDT_CONTR.7=1, WDT reset, hot start-up, the content of RAM is constant and just like before reset
WDT_Reset:
```

```
    CLR    WDT_FLAG_LED
           ;WDT reset,open WDT overflow reset flag LED indicator
    JB     Last_WDT_Time_LED_Status,    Power_Off_WDT_TIME_LED
           ;when set Last_WDT_Time_LED_Status, close the corresponding LED indicator
           ;clear, open the corresponding LED indicator
           ;set WDT_TIME_LED according to the last status of WDT overflow time LED indicator
    CLR    WDT_TIME_LED                ;close the WDT overflow time LED indicator
    CPL    Last_WDT_Time_LED_Status
           ;reverse the last status of WDT overflow time LED indicator
```

WAIT2:

```
        SJMP    WAIT2                ;wait WDT overflow reset
Power_Off_WDT_TIME_LED:
    SETB   WDT_TIME_LED                ;close the WDT overflow time LED indicator
    CPL    Last_WDT_Time_LED_Status
           ;reverse the last status of WDT overflow time LED indicator
```

WAIT3:

```
        SJMP    WAIT3                ;wait WDT overflow reset
    END
```

## 2.2.7 Reset Caused by Program Accessing an Invalid Address

It will generate a reset if the address that program counter point to is invalid. That is a reset caused by program accessing an invalid address. this reset don't set the bit SWBS/IAP\_CONTR.6. If the bit SWBS/IAP\_CONTR.6 has been set as 0 before reset, MCU will start to run from the user application program area after reset. If the bit SWBS/IAP\_CONTR.6 has been set as 1 before reset, MCU will start to run from the system ISP monitor program area after reset on the contrary. Reset caused by illegal use of program address is one of soft reset of warm boot.

## 2.2.8 Warm Boot and Cold Boot Reset

Reset type		Reset source		Result	The value of SWBS/ IAP_CONTR.6 after reset
Warm boot	Soft reset	Software Reset	20H → IAP_CONTR	System will reset to AP address 0000H and begin running user application program	0
			60H → IAP_CONTR	System will reset to ISP address 0000H and begin running ISP monitor program, if not detected legitimate ISP command, system will software reset to the user program area automatically.	1
		Watch-Dog-Timer Reset	If the value of SWBS/ IAP_CONTR.6 is 0 before reset	System will reset to AP address 0000H and begin running user application program	0
			If the value of SWBS/ IAP_CONTR.6 is 1 before reset	System will reset to ISP address 0000H and begin running ISP monitor program, if not detected legitimate ISP command, system will software reset to the user program area automatically.	1
		Reset caused by illegal use of program address	If the value of SWBS/ IAP_CONTR.6 is 0 before reset	System will reset to AP address 0000H and begin running user application program	0
			If the value of SWBS/ IAP_CONTR.6 is 1 before reset	System will reset to ISP address 0000H and begin running ISP monitor program, if not detected legitimate ISP command, system will software reset to the user program area automatically.	1
	Hard reset	Internal Low-Voltage Detection Reset	If the value of SWBS/ IAP_CONTR.6 is 0 before reset	System will reset to AP address 0000H and begin running user application program	0
			If the value of SWBS/ IAP_CONTR.6 is 1 before reset	System will reset to ISP address 0000H and begin running ISP monitor program, if not detected legitimate ISP command, system will software reset to the user program area automatically.	1
		External RST Pin Reset		System will reset to ISP address 0000H and begin running ISP monitor program, if not detected legitimate ISP command, system will software reset to the user program area automatically.	1
	Cold boot	Cold boot reset namely Power-Off / Power-On Reset caused by the power of system be off or on		System will reset to ISP address 0000H and begin running ISP monitor program, if not detected legitimate ISP command, system will software reset to the user program area automatically.	1

### IAP\_CONTR: ISP/IAP Control Register

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	name	IAPEN	SWBS	SWRST	CMD_FAIL	-	WT2	WT1	WT0

SWBS: software boot selection control bit

0 : Boot from main-memory after reset.

1 : Boot from ISP memory after reset.

SWRST: software reset trigger control.

0 : No operation

1 : Generate software system reset. It will be cleared by hardware automatically.

## 2.3 Power Management Modes

The STC15 series core has three software programmable power management mode: slow-down, idle and stop/power-down mode. The power consumption of STC15W4K32S4 series is between 4mA~6mA in normal operation, while it is lower than 0.4uA in stop/power-down mode and 1mA in idle mode.

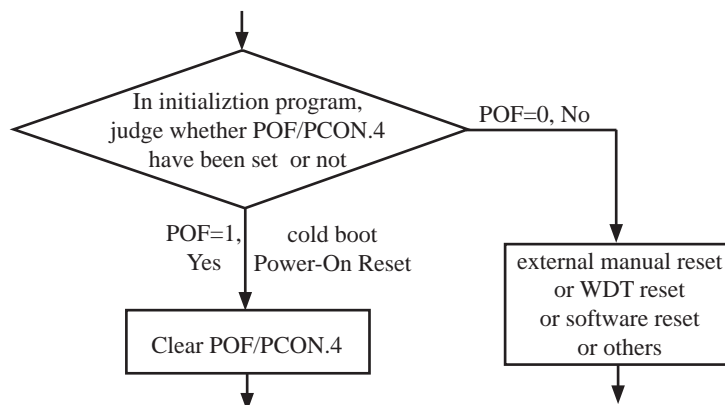
Slow-down mode is controlled by clock divider register CLK\_DIV (PCON2). Idle and stop/power-down is managed by the corresponding bit in Power control (PCON) register which is shown in below.

**PCON register** (Power Control Register)

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

**POF** : Power-On flag. It is set by power-off-on action and can only cleared by software.

Practical application: if it is wanted to know which reset the MCU is used, see the following figure.



GF1,GF0: General-purposed flag 1 and 0

**PD** : Stop Mode/Power-Down Select bit..

Setting this bit will place the STC15 series MCU in Stop/Power-Down mode. Stop/Power-Down mode can be waked up by external interrupt. Because the MCU' s internal oscillator stopped in Stop/Power-Down mode, CPU, Timers, UARTs and so on stop to run, only external interrupt go on to work. The following pins can wake up MCU from Stop/Power-Down mode: INT0/P3.2, INT1/P3.3,  $\overline{\text{INT2}}$ /P3.6,  $\overline{\text{INT3}}$ /P3.7,  $\overline{\text{INT4}}$ /P3.0; pins CCP0/CCP1/CCP2/CCP3/CCP4/CCP5; pins RxD/RxD2/RxD3/RxD4; pins T0/T1/T2/T3/T4; Internal power-down wake-up Timer.

**IDL** : Idle mode select bit.

Setting this bit will place the STC15 series in Idle mode. only CPU goes into Idle mode. (Shuts off clock to CPU, but clock to Timers, Interrupts, Serial Ports, and Analog Peripherals are still active.) External Interrupts, Timer interrupts, low-voltage detection interrupt and ADC interrupt all can wake up MCU from Idle mode.

### 2.3.1 Slow Down Mode and Demo Program (C and ASM)

A divider is designed to slow down the clock source prior to route to all logic circuit. The operating frequency of internal logic circuit can therefore be slowed down dynamically , and then save the power.

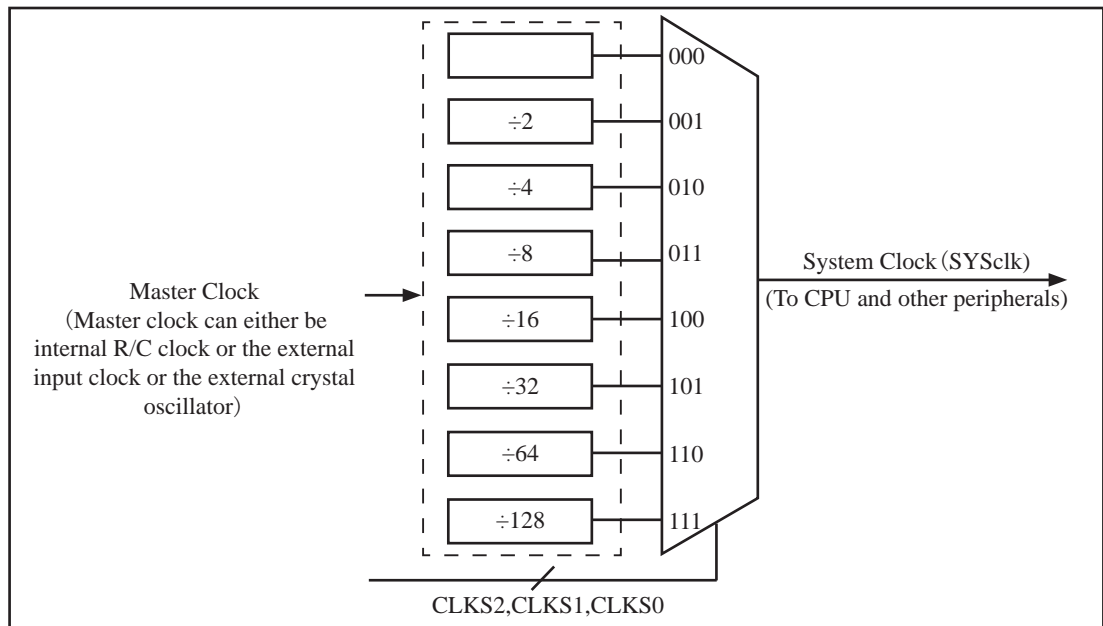
User can slow down the MCU by means of writing a non-zero value to the CLKS[2:0] bits in the CLK\_DIV register. This feature is especially useful to save power consumption in idle mode as long as the user changes the CLKS[2:0] to a non-zero value before entering the idle mode.

Clock Division Register CLK\_DIV (PCON2):

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CLK_DIV (PCON2)	97H	name	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCKO_2	CLKS2	CLKS1	CLKS0

CLKS2	CLKS1	CLKS0	the control bit of system clock (System clock refers to the master clock that has been divided frequency, which is offered to CPU, UARTs, SPI, Timers, CCP/PWM/PCA and A/D Converter)
0	0	0	Master clock frequency/1, No division
0	0	1	Master clock frequency/2
0	1	0	Master clock frequency/4
0	1	1	Master clock frequency/8
1	0	0	Master clock frequency/16
1	0	1	Master clock frequency/32
1	1	0	Master clock frequency/64
1	1	1	Master clock frequency/128

The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator.



**Clock Structure**



---

## 1. C Program Listing

```
/*-----*/
/* --- Exam Program of Slow-down mode -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

sfr      CLK_DIV   = 0x97;

//-----

void main()
{
    CLK_DIV = 0x00;      //System clock is MCLK (master clock)
//    CLK_DIV = 0x01;      //System clock is MCLK/2
//    CLK_DIV = 0x02;      //System clock is MCLK/4
//    CLK_DIV = 0x03;      //System clock is MCLK/8
//    CLK_DIV = 0x04;      //System clock is MCLK/16
//    CLK_DIV = 0x05;      //System clock is MCLK/32
//    CLK_DIV = 0x06;      //System clock is MCLK/64
//    CLK_DIV = 0x07;      //System clock is MCLK/128

    while (1);
}
```

---

## 2. Assembler Listing

```
/*-----*/
/* --- Exam Program of Slow-down mode -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

CLK_DIV  DATA      097H

//-----
        ORG    0000H
        LJMP   MAIN
//-----

        ORG    0100H
MAIN:
        MOV    SP,    #3FH

        MOV    CLK_DIV,    #0           //System clock is MCLK (master clock)
//      MOV    CLK_DIV,    #1           //System clock is MCLK/2
//      MOV    CLK_DIV,    #2           //System clock is MCLK/4
//      MOV    CLK_DIV,    #3           //System clock is MCLK/8
//      MOV    CLK_DIV,    #4           //System clock is MCLK/16
//      MOV    CLK_DIV,    #5           //System clock is MCLK/32
//      MOV    CLK_DIV,    #6           //System clock is MCLK/64
//      MOV    CLK_DIV,    #7           //System clock is MCLK/128

        SJMP   $

;-----

        END
```

---

## 2.3.2 Idle Mode and Demo Program (C and ASM)

An instruction that sets IDL/PCON.0 causes that to be the last instruction executed before going into the idle mode, the internal clock is gated off to the CPU but not to the interrupt, timer, CCP/PCA/PWM, SPI, ADC, WDT and serial port functions. The PCA can be programmed either to pause or continue operating during Idle. The CPU status is preserved in its entirety: the RAM, Stack Pointer, Program Counter, Program Status Word, Accumulator, and all other registers maintain their data during Idle. The port pins hold the logical states they had at the time Idle was activated. Idle mode leaves the peripherals running in order to allow them to wake up the CPU when an interrupt is generated. Timer 0, Timer 1, CCP/PCA/PWM timer and UARTs will continue to function during Idle mode.

There are two ways to terminate the idle. Activation of any enabled interrupt will cause IDL/PCON.0 to be cleared by hardware, terminating the idle mode. The interrupt will be serviced, and following RETI, the next instruction to be executed will be the one following the instruction that put the device into idle.

The flag bits (GFO and GF1) can be used to give an indication if an interrupt occurred during normal operation or during Idle. For example, an instruction that activates Idle can also set one or both flag bits. When Idle is terminated by an interrupt, the interrupt service routine can examine the flag bits.

The other way to wake-up from idle is to pull RESET high to generate internal hardware reset. Since the clock oscillator is still running, the hardware reset needs to be held active for at least 24 clocks plus 20us to complete the reset. After reset, MCU start to run from the system ISP monitor program area.

### 1. C Program Listing

```
/*-----*/
/* --- Exam Program of Idle mode -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

//-----
```

---

```

void main()
{
    while (1)
    {
        PCON |= 0x01;           //set IDL(PCON.0) as 1, MCU in Idle mode
        _nop_();
        _nop_();               //internal interrupts or external interrupts singnal can
        _nop_();               //wake up mcu from idle mode
        _nop_();
    }
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- Exam Program of Idle mode -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/
//suppose the frequency of test chip is 18.432MHz
//-----

        ORG    0000H
        LJMP   MAIN

//-----

        ORG    0100H
MAIN:    MOV    SP,    #3FH

LOOP:   MOV    PCON, #01H           //set IDL(PCON.0) as 1, MCU in Idle mode
        NOP                    //internal interrupts or external interrupts singnal can
        NOP                    //wake up mcu from idle mode
        NOP
        NOP
        JMP    LOOP

;-----

        END

```

---

---

## 2.2.3 Stop / Power Down (PD) Mode and Demo Program (C and ASM)

Setting the PD/PCON.1 bit enters Stop/Power-Down mode. In the Stop/Power-Down mode, the on-chip oscillator and the Flash memory are stopped in order to minimize power consumption. Only the power-on circuitry will continue to draw power during Stop/Power-Down. The contents of on-chip RAM and SFRs are maintained. The stop/power-down mode can be woken-up by RESET pin, external interrupt INT0/INT1/  $\overline{\text{INT2}}$ /  $\overline{\text{INT3}}$ /  $\overline{\text{INT4}}$ , RxD/RxD2/RxD3/RxD4 pins, T0/T1/T2/T3/T4 pins, CCP/PCA input pins — CCP0/CCP1 pins, low-voltage detection interrupt and internal power-down wake-up Timer.

When it is woken-up by RESET, the program will execute from the ISP monitor program area. Be carefully to keep RESET pin active for at least 10ms in order for a stable clock.

If it is woken-up from I/O, the CPU will rework through jumping to related interrupt service routine. Before the CPU rework, the clock is blocked and counted until 32768 in order for denouncing the unstable clock. To use I/O wake-up, interrupt-related registers have to be enabled and programmed accurately before power-down is entered. Pay attention to have at least one “NOP” instruction subsequent to the power-down instruction if I/O wake-up is used. When terminating Power-down by an interrupt, the wake up period is internally timed. At the negative edge on the interrupt pin, Power-Down is exited, the oscillator is restarted, and an internal timer begins counting. The internal clock will be allowed to propagate and the CPU will not resume execution until after the timer has reached internal counter full. After the timeout period, the interrupt service routine will begin. To prevent the interrupt from re-triggering, the interrupt service routine should disable the interrupt before returning. The interrupt pin should be held low until the device has timed out and begun executing. The user should not attempt to enter (or re-enter) the power-down mode for a minimum of 4 us until after one of the following conditions has occurred: Start of code execution(after any type of reset), or Exit from power-down mode.

### 1. C Program Listing

```
/*-----*/
/* --- Exam Program of Stop/Power-Down mode -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/
```

```
//suppose the frequency of test chip is 18.432MHz
```

```
#include "reg51.h"
#include "intrins.h"
//-----
void main()
{
```

---

```

while (1)
{
    PCON |= 0x02;           //Set STOP(PCON.1) as 1.
                           // After this instruction, MCU will be in power-down mode
                           //external clock stop

    _nop_();
    _nop_();
    _nop_();
}
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- Exam Program of Stop/Power-Down mode -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz
//-----
    ORG    0000H
    LJMP   MAIN
//-----
    ORG    0100H
MAIN:
    MOV    SP,    #3FH

LOOP:
    MOV    PCON, #02H           //Set STOP(PCON.1) as 1
                               // After this instruction, MCU will be in power-down mode
                               //external clock stop

    NOP
    NOP
    NOP
    NOP
    JMP    LOOP
;-----

    END

```

---

---

### 2.3.3.1 Demo Program Using Power-Down Wake-Up Timer to Wake Up Stop/PD Mode

/\*Demo program using internal power-down wake-up special Timer wake up Stop/Power-Down mode(C and ASM) \*/

#### 1. C Program Listing

```
/*-----*/
/* --- Exam Program using power-down wake-up Timer to wake up Stop/Power-Down mode */
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/
```

```
//suppose the frequency of test chip is 18.432MHz
```

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

```
//-----
```

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

```
sbit P10 = P1^0;
```

```
//-----
```

```
void main()
{
    WKTCL = 49; //wake-up cycle: 488us*(49+1) = 24.4ms
    WKTCH = 0x80;

    while (1)
    {
        PCON = 0x02; //Enter Stop/Power-Down Mode
        _nop_();
        _nop_();
        P10 = !P10;
    }
}
```

---

## 2. Assembler Listing

```
/*-----*/
/* --- Exam Program using power-down wake-up Timer wake up Stop/Power-Down mode -*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

WKTCL DATA 0AAH
WKTCH DATA 0ABH

//-----

        ORG    0000H
        LJMP   MAIN

//-----

MAIN:    ORG    0100H
        MOV    SP,    #3FH

        MOV    WKTCL, #49           //wake-up cycle: 488us*(49+1) = 24.4ms
        MOV    WKTCH, #80H

LOOP:   MOV    PCON, #02H           //Enter Stop/Power-Down Mode
        NOP
        NOP
        CPL    P1.0
        JMP    LOOP

        SJMP   $

;-----

        END
```



---

## 2.3.3.2 Demo Program Using External Interrupt INT0 to Wake Up Stop/PD Mode

### 1. C Program Listing

```
/*-----*/
/* --- Exam Program using external interrupt INT0 (rising +falling edge) to wake up Stop/Power-Down mode */
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

//-----
bit    FLAG;                //1:generate a interrupt on rising edge
                                //0:generate a interrupt on falling edge

sbit   P10    =    P1^0;

//-----
//Interrupt service routine
void exint0() interrupt 0
{
    P10    =    !P10;
    FLAG   =    INT0;        //save the sate of INT0, INT0=0(falling); INT0=1(rising)
}
//-----

void main()
{
    IT0 = 0;                //Both rising and falling edge of INT0 can wake up MCU
//    IT0 = 1;                //Only falling edge of INT0 can wake up MCU

    EX0 = 1;
    EA = 1;

    while (1)
    {
        PCON = 0x02;        //MCU enter Stop/Power-Down mode
        _nop_();            //Fisrt implement this statement and then enter interrupt service routine
                                //after be waked up from Stop/Power-Down mode

        _nop_();
    }
}
```

---

## 2. Assembler Listing

```
/*-----*/
/* --- Exam Program using external interrupt INT0 (rising +falling edge) to wake up Stop/Power-Down mode */
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

FLAG BIT 20H.0 //1:generate a interrupt on rising edge
//0:generate a interrupt on falling edge

//-----
ORG 0000H
LJMP MAIN

ORG 0003H
LJMP EXINT0

//-----
ORG 0100H
MAIN:
MOV SP, #3FH

CLR IT0 //Both rising and falling edge of INT0 can wake up MCU
// SETB IT0 //Only falling edge of INT0 can wake up MCU
SETB EX0
SETB EA

LOOP:
MOV PCON, #02H //MCU enter Stop/Power-Down mode
NOP //Fisrt implement this statement and then enter interrupt service routine
//after be waked up from Stop/Power-Down mode

NOP
SJMP LOOP

//-----
EXINT0: //Interrupt service routine
CPL P1.0
PUSH PSW
MOV C, INT0 //read the state of INT0
MOV FLAG, C //save the sate of INT0, INT0=0(falling); INT0=1(rising)
POP PSW
RETI

;-----
END
```

---

### 2.3.3.3 Demo Program Using External Interrupt INT1 to Wake Up Stop/PD Mode

#### 1. C Program Listing

```
/*-----*/
/* --- Exam Program using external interrupt INT1 (rising +falling edge) to wake up Stop/Power-Down mode */
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

//-----
bit    FLAG;                //1:generate a interrupt on rising edge
                        //0:generate a interrupt on falling edge

sbit   P10    =    P1^0;

//-----

void exint1() interrupt 2
{
    P10    =    !P10;
    FLAG   =    INT1;        //save the sate of INT1, INT1=0(falling); INT1=1(rising)
}

//-----

void main()                //Interrupt service routine
{
    IT1    =    0;        //Both rising and falling edge of INT1 can wake up MCU
//    IT1    =    1;        //Only falling edge of INT1 can wake up MCU

    EX1    =    1;
    EA     =    1;

    while (1)
    {
        PCON = 0x02;    //MCU enter Stop/Power-Down mode
        _nop_();        //Fisrt implement this statement and then enter interrupt service routine
                        //after be waked up from Stop/Power-Down mode
        _nop_();
    }
}
```

---

## 2. Assembler Listing

```
/*-----*/
/* --- Exam Program using external interrupt INT1 (rising +falling edge) to wake up Stop/Power-Down mode */
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

FLAG  BIT    20H.0                //1:generate a interrupt on rising edge
                                //0:generate a interrupt on falling edge

//-----
        ORG    0000H
        LJMP   MAIN

        ORG    0013H
        LJMP   EXINT1

//-----
        ORG    0100H
MAIN:
        MOV    SP,    #3FH

        CLR    IT1        //Both rising and falling edge of INT1 can wake up MCU
//        SETB  IT1        //Only falling edge of INT1 can wake up MCU

        SETB  EX1
        SETB  EA

LOOP:
        MOV    PCON,  #02H    //MCU enter Stop/Power-Down mode
        NOP                                //Fisrt implement this statement and then enter interrupt service routine
                                //after be waked up from Stop/Power-Down mode

        NOP
        SJMP   LOOP

;-----
EXINT1:
        CPL    P1.0
        PUSH  PSW
        MOV    C,    INT1        //read the state of INT1
        MOV    FLAG, C          //save the sate of INT1, INT1=0(falling); INT1=1(rising)
        POP   PSW
        RETI

;-----

        END
```

---

### 2.3.3.4 Demo Program Using External Interrupt $\overline{\text{INT2}}$ to Wake Up Stop/PD Mode

#### 1. C Program Listing

```
/*-----*/
/* --- Exam Program using external interrupt /INT2 (only falling edge) to wake up Stop/Power-Down mode ---*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"
#include "intrins.h"
//-----

sfr    INT_CLKO    =    0x8F;
sbit   INT2       =    P3^6;

sbit   P10        =    P1^0;
//-----
//Interrupt service routine
void exint2() interrupt 10
{
    P10    =    !P10;

//    INT_CLKO    &=  0xEF;
//    INT_CLKO    |=  0x10;
}
//-----
void main()
{
    INT_CLKO    |=  0x10;                //(EX2 = 1) enable the falling edge of INT2 interrupt
    EA = 1;

    while (1)
    {
        PCON = 0x02;                    //MCU enter Stop/Power-Down mode
        _nop_();                        //Fisrt implement this statement and then enter interrupt service routine
        _nop_();                        //after be waked up from Stop/Power-Down mode
    }
}
}
```

---

## 2. Assembler Listing

```
/*-----*/
/* --- Exam Program using external interrupt /INT2 (only falling edge) to wake up Stop/Power-Down mode ---*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

INT_CLKO    DATA    08FH
INT2        BIT      P3.6
//-----

        ORG    0000H
        LJMP   MAIN

        ORG    0053H
        LJMP   EXINT2
//-----

        ORG    0100H
MAIN:
        MOV    SP,    #3FH
        ORL    INT_CLKO,    #10H    //(EX2 = 1) enable the falling edge of INT2 interrupt
        SETB   EA

LOOP:
        MOV    PCON,    #02H        //MCU enter Stop/Power-Down mode
        NOP                                //Fisrt implement this statement and then enter interrupt service routine
                                           //after be waked up from Stop/Power-Down mode
        NOP
        SJMP   LOOP
//-----
//Interrupt service routine
EXINT2:
        CPL    P1.0
//        ANL    INT_CLKO,    #0EFH
//        ORL    INT_CLKO,    #10H

        RETI
;-----
        END
```

---

### 2.3.3.5 Demo Program Using External Interrupt $\overline{\text{INT3}}$ to Wake Up Stop/PD Mode

#### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using external interrupt /INT3 (only falling edge) to wake up Stop/Power-Down mode ---*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"
#include "intrins.h"
//-----

sfr    INT_CLKO    =    0x8F;
sbit   INT3       =    P3^7;
sbit   P10        =    P1^0;

//-----
//Interrupt service routine
void exint3() interrupt 11
{
    P10    =    !P10;

//    INT_CLKO    &=    0xDF;
//    INT_CLKO    |=    0x20;
}
//-----

void main()
{
    INT_CLKO    |=    0x20;        //(EX3 = 1) enable the falling edge of INT3 interrupt
    EA          =    1;

    while (1)
    {
        PCON = 0x02;                //MCU enter Stop/Power-Down mode
        _nop_();                    //Fisrt implement this statement and then enter interrupt service routine
        //after be waked up from Stop/Power-Down mode

        _nop_();
    }
}
}
```

---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using external interrupt /INT3 (only falling edge) to wake up Stop/Power-Down mode ---*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

INT_CLKO    DATA    08FH
INT3        BIT      P3.7
//-----

        ORG    0000H
        LJMP   MAIN

        ORG    005BH
        LJMP   EXINT3
//-----

MAIN:    ORG    0100H

        MOV    SP,    #3FH

        ORL    INT_CLKO,    #20H           //(EX3 = 1) enable the falling edge of INT3 interrupt

        SETB   EA

LOOP:

        MOV    PCON,    #02H           //MCU enter Stop/Power-Down mode
        NOP                                //Fisrt implement this statement and then enter interrupt service routine
                                           //after be waked up from Stop/Power-Down mode

        NOP
        SJMP   LOOP
//-----
//Interrupt service routine

EXINT3:

        CPL    P1.0

//        ANL    INT_CLKO,    #0DFH
//        ORL    INT_CLKO,    #20H

        RETI
;-----
        END
```



---

### 2.3.3.6 Demo Program Using External Interrupt INT4 to Wake Up Stop/PD Mode

#### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using external interrupt /INT4 (only falling edge) to wake up Stop/Power-Down mode ---*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

//-----
sfr INT_CLKO = 0x8F;
sbit INT4 = P3^0;
sbit P10 = P1^0;

//-----
//Interrupt service routine
void exint4() interrupt 16
{
    P10 = !P10;

    // INT_CLKO &= 0xBF;
    // INT_CLKO |= 0x40;
}

//-----
void main()
{
    INT_CLKO |= 0x40; // (EX4 = 1) enable the falling edge of INT4 interrupt
    EA = 1;

    while (1)
    {
        PCON = 0x02; // MCU enter Stop/Power-Down mode
        _nop_(); // First implement this statement and then enter interrupt service routine
        // after be waked up from Stop/Power-Down mode
        _nop_();
    }
}
```

---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using external interrupt /INT3 (only falling edge) to wake up Stop/Power-Down mode ---*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

INT_CLKO    DATA    08FH
INT4        BIT      P3.0
//-----
        ORG    0000H
        LJMP  MAIN

        ORG    0083H
        LJMP  EXINT4
//-----
        ORG    0100H
MAIN:
        MOV   SP,    #3FH
        ORL   INT_CLKO,    #40H           //(EX4 = 1) enable the falling edge of INT4 interrupt
        SETB  EA

LOOP:
        MOV   PCON,    #02H           //MCU enter Stop/Power-Down mode
        NOP                               //Fisrt implement this statement and then enter interrupt service routine
                                           //after be waked up from Stop/Power-Down mode
        NOP
        SJMP  LOOP

//-----
//Interrupt service routine
EXINT4:
        CPL   P1.0

//        ANL   INT_CLKO,    #0BFH
//        ORL   INT_CLKO,    #40H

        RETI
;-----
        END
```

---

### 2.3.3.7 Program Using External Interrupt Extended by CCP/PCA to Wake Up PD Mode

/\*Demo program using external interrupt (rising + falling edge) extended by CCP/PCA to wake up Stop/Power-Down mode(C and ASM) \*/

#### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using external interrupt extended by CCP/PCA to wake up Stop/Power-Down mode */
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/
```

```
//suppose the frequency of test chip is 18.432MHz
```

```
//This demo program take CCP/PCA module 0 for example. the use of CCP/PCA module 1 and CCP/PCA module //2 are same as CCP/PCA module 0
```

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

```
#define FOSC 18432000L
```

```
typedef unsigned char BYTE;
typedef unsigned int WORD;
typedef unsigned long DWORD;
```

```
sfr P_SW1 = 0xA2;
```

```
#define CCP_S0 0x10 //P_SW1.4
```

```
#define CCP_S1 0x20 //P_SW1.5
```

```
sfr CCON = 0xD8; //PCA Control register
```

```
sbit CCF0 = CCON^0;
```

```
sbit CCF1 = CCON^1;
```

```
sbit CR = CCON^6;
```

```
sbit CF = CCON^7;
```

```
sfr CMOD = 0xD9;
```

```
sfr CL = 0xE9;
```

```
sfr CH = 0xF9;
```

```
sfr CCAPM0 = 0xDA;
```

```
sfr CCAP0L = 0xEA;
```

```
sfr CCAP0H = 0xFA;
```

```
sfr CCAPM1 = 0xDB;
```

```
sfr CCAP1L = 0xEB;
```

```
sfr CCAP1H = 0xFB;
```

```
sbit P10 = P1^0;
```

---

```

void main()
{
    ACC    =    P_SW1;
    ACC    &=    ~(CCP_S0 | CCP_S1);    //CCP_S0=0 CCP_S1=0
    P_SW1  =    ACC;    //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)

//    ACC    =    P_SW1;
//    ACC    &=    ~(CCP_S0 | CCP_S1);    //CCP_S0=1 CCP_S1=0
//    ACC    |=    CCP_S0;    //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//    P_SW1  =    ACC;
//
//    ACC    =    P_SW1;
//    ACC    &=    ~(CCP_S0 | CCP_S1);    //CCP_S0=0 CCP_S1=1
//    ACC    |=    CCP_S1;    //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//    P_SW1  =    ACC;

    CCON  =    0;

    CL    =    0;
    CH    =    0;
    CCAP0L = 0;
    CCAP0H = 0;
    CMOD  =    0x08;    //Seting the PCA clock as system clock
    CCAPM0= 0x21;
//    CCAPM0 = 0x11;

//    CCAPM0 = 0x31;
    CR    =    1;
    EA    =    1;

    while (1)
    {
        PCON = 0x02;    //MCU enter Stop/Power-Down mode
        _nop_();    //Fisrt implement this statement and then enter interrupt service routine
        //after be waked up from Stop/Power-Down mode

        _nop_();
    }
}

void PCA_isr() interrupt 7 using 1
{
    if (CCF0)
    {
        CCF0    =    0;
        P10    =    !P10;
    }
}

```

---

---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using external interrupt extended by CCP/PCA to wake up Stop/Power-Down mode */
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/
```

```
//suppose the frequency of test chip is 18.432MHz
```

```
//This demo program take CCP/PCA module 0 for example. the use of CCP/PCA module 1 and CCP/PCA module
//2 are same as CCP/PCA module 0
```

```
P_SW1 EQU 0A2H

CCP_S0 EQU 10H //P_SW1.4
CCP_S1 EQU 20H //P_SW1.5
CCON EQU 0D8H //PCA Control register
CCF0 BIT CCON.0
CCF1 BIT CCON.1
CR BIT CCON.6
CF BIT CCON.7
CMOD EQU 0D9H
CL EQU 0E9H
CH EQU 0F9H
CCAPM0 EQU 0DAH
CCAP0L EQU 0EAH
CCAP0H EQU 0FAH
CCAPM1 EQU 0DBH
CCAP1L EQU 0EBH
CCAP1H EQU 0FBH
```

```
//-----
ORG 0000H
LJMP MAIN

ORG 003BH
PCA_ISR:
PUSH PSW
PUSH ACC
CKECK_CCF0:
JNB CCF0, PCA_ISR_EXIT
CLR CCF0
CPL P1.0
PCA_ISR_EXIT:
POP ACC
POP PSW
```

---

```

    RETI
//-----
    ORG    0100H
MAIN:
    MOV    SP,    #5FH

    MOV    A,     P_SW1
    ANL    A,     #0CFH    //CCP_S0=0 CCP_S1=0
    MOV    P_SW1, A        //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)

//    MOV    A,     P_SW1
//    ANL    A,     #0CFH    //CCP_S0=1 CCP_S1=0

//    ORL    A,     #CCP_S0    //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//    MOV    P_SW1, A

//
//    MOV    A,     P_SW1
//    ANL    A,     #0CFH    //CCP_S0=0 CCP_S1=1
//    ORL    A,     #CCP_S1    //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//    MOV    P_SW1, A
    MOV    CCON, #0
    CLR    A
    MOV    CL,    A
    MOV    CH,    A
    MOV    CCAP0L, A
    MOV    CCAP0H, A
    MOV    CMOD, #08H    //Seting the PCA clock as system clock
    MOV    CCAPM0, #21H
//    MOV    CCAPM0, #11H
//    MOV    CCAPM0, #31H

    SETB   CR
    SETB   EA

LOOP:
    MOV    PCON, #02H    //MCU enter Stop/Power-Down mode
    NOP                    //Fisrt implement this statement and then enter interrupt service routine
                        //after be waked up from Stop/Power-Down mode

    NOP
    SJMP  LOOP

//-----

    END

```

---

---

### 2.3.3.8 Program Using the Level Change of RxD pin to Wake Up Stop/PD Mode

/\*Demo program using the level change from high to low of RxD pin to wake up Stop/Power-Down mode(C and ASM) \*/

#### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using the level change from high to low of RxD pin to wake up Stop/Power-Down mode */
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

//-----

sfr    AUXR    =    0x8e;        //Auxiliary register
sfr    T2H     =    0xd6;
sfr    T2L     =    0xd7;

sfr    P_SW1   =    0xA2;

#define S1_S0  0x40            //P_SW1.6
#define S1_S1  0x80            //P_SW1.7

sbit   P10     =    P1^0;

//-----

void main()
{
    ACC     =    P_SW1;
    ACC    &=    ~(S1_S0 | S1_S1);    //S1_S0=0 S1_S1=0
    P_SW1   =    ACC;                //(P3.0/RxD, P3.1/TxD)

//    ACC     =    P_SW1;
//    ACC    &=    ~(S1_S0 | S1_S1);    //S1_S0=1 S1_S1=0
//    ACC    |=    S1_S0;                //(P3.6/RxD_2, P3.7/TxD_2)
//    P_SW1   =    ACC;
```

---

```

//
// ACC = P_SW1;
// ACC &= ~(S1_S0 | S1_S1); //S1_S0=0 S1_S1=1

// ACC |= S1_S1; // (P1.6/RxD_3, P1.7/TxD_3)
// P_SW1 = ACC;

SCON = 0x50; //8-bit variable baud rate
T2L = (65536 - (FOSC/4/BAUD)); //Setting the reload value of buad rate
T2H = (65536 - (FOSC/4/BAUD))>>8;
AUXR = 0x14; //T2 in 1T mode, and run Timer 2
AUXR |= 0x01; //Select Timer2 as the baud-rate generator of UART1

ES = 1;
EA = 1;

while (1)
{
    PCON = 0x02; //MCU enter Stop/Power-Down mode
    _nop_(); //Fisrt implement this statement and then enter interrupt service routine
    //after be waked up from Stop/Power-Down mode
    _nop_();
    P10 = !P10;
}
}

/*-----
UART interrupt service Routine
-----*/
void Uart() interrupt 4 using 1
{
    if (RI)
    {
        RI = 0; //clear RI
        P0 = SBUF;
    }
    if (TI)
    {
        TI = 0; //clear TI
    }
}

```



---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using the level change from high to low of RxD pin to wake up Stop/Power-Down mode */
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

//-----

AUXR EQU 08EH //Auxiliary register
T2H DATA 0D6H
T2L DATA 0D7H

P_SW1 EQU 0A2H

S1_S0 EQU 40H //P_SW1.6
S1_S1 EQU 80H //P_SW1.7

//-----

ORG 0000H
LJMP MAIN

ORG 0023H
LJMP UART_ISR

//-----

ORG 0100H
MAIN:
MOV SP, #3FH
MOV A, P_SW1
ANL A, #03FH //S1_S0=0 S1_S1=0
MOV P_SW1, A //(P3.0/RxD, P3.1/TxD)

// MOV A, P_SW1
// ANL A, #03FH //S1_S0=1 S1_S1=0
// ORL A, #S1_S0 //(P3.6/RxD_2, P3.7/TxD_2)
// MOV P_SW1, A
```

---

```

//      MOV    A,      P_SW1
//      ANL    A,      #03FH          //S1_S0=0 S1_S1=1
//      ORL    A,      #S1_S1        //(P1.6/RxD_3, P1.7/TxD_3)
//      MOV    P_SW1, A

      MOV    SCON,    #50H          //8-bit variable baud rate
      MOV    T2L,     #0D8H        //Setting the reload value of buad rate (65536-18432000/4/115200)
      MOV    T2H,     #0FFH
      MOV    AUXR,    #14H          //T2 in 1T mode, and run Timer 2
      ORL    AUXR,    #01H          //Select Timer2 as the baud-rate generator of UART1

      SETB   ES                //enable UART1 interrupt
      SETB   A

LOOP:
      MOV    PCON,    #02H          //MCU enter Stop/Power-Down mode
      NOP                                //Fisrt implement this statement and then enter interrupt service routine
                                      //after be waked up from Stop/Power-Down mode

      NOP
      CPL    P1.0
      SJMP   LOOP

;-----
;UART interrupt service Routine
;-----*/
UART_ISR:
      PUSH   ACC
      PUSH   PSW
      JNB    RI,      CHECKTI        //check RI
      CLR    RI                //clear RI
      MOV    P0,      SBUF

CHECKTI:
      JNB    TI,      ISR_EXIT        //check TI
      CLR    TI                //clear TI

ISR_EXIT:
      POP    PSW
      POP    ACC
      RETI

;-----

      END

```

---

---

### 2.3.3.9 Program Using the Level Change of RxD2 pin to Wake Up Stop/PD Mode

*/\*Demo program using the level change from high to low of RxD2 pin to wake up Stop/Power-Down mode(C and ASM) \*/*

#### 1. C Program Listing

```
/*-----*/  
/* --- STC MCU Limited. -----*/  
/* --- Exam Program using the level change from high to low of RxD2 pin to wake up Stop/Power-Down mode */  
/* If you want to use the program or the program referenced in the -----*/  
/* article, please specify in which data and procedures from STC -----*/  
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/  
/*---- And only contain < reg51.h > as header file -----*/  
/*-----*/
```

```
//suppose the frequency of test chip is 18.432MHz
```

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

```
#define FOSC 18432000L //System frequency  
#define BAUD 115200  
#define TM (65536 - (FOSC/4/BAUD))
```

```
//-----
```

```
sfr AUXR = 0x8e; //Auxiliary register  
sfr S2CON = 0x9a;  
sfr S2BUF = 0x9b;  
sfr T2H = 0xd6;  
sfr T2L = 0xd7;  
sfr IE2 = 0xaf;
```

```
#define S2RI 0x01 //S2CON.0  
#define S2TI 0x02 //S2CON.1  
#define S2RB8 0x04 //S2CON.2  
#define S2TB8 0x08 //S2CON.3
```

```
sfr P_SW2 = 0xBA;
```

```
#define S2_S 0x01 //P_SW2.0
```

```
sbit P20 = P2^0;
```

```
//-----
```

---

```

void main()
{
    P_SW2  &=   ~S2_S;           //S2_S=0 (P1.0/RxD2, P1.1/TxD2)
//    P_SW2  |=   S2_S;           //S2_S=1 (P4.6/RxD2_2, P4.7/TxD2_2)

    S2CON  =    0x50;           //8-bit variable baud rate
    T2L    =    TM;             //Setting the reload value of buad rate
    T2H    =    TM>>8;
    AUXR   =    0x14;           //T2 in 1T mode, and run Timer 2

    IE2    =    0x01;           //enable UART1 interrupt
    EA     =    1;

    while (1)
    {
        PCON  =    0x02;        //MCU enter Stop/Power-Down mode
        _nop_();                //Fisrt implement this statement and then enter interrupt service routine
                                //after be waked up from Stop/Power-Down mode
        _nop_();
        P20   =    !P20;
    }
}

/*-----
UART2 interrupt service Routine
-----*/
void Uart2() interrupt 8 using 1
{
    if (S2CON & S2RI)
    {
        S2CON  &=   ~S2RI;       //clear S2RI
        P0     =    S2BUF;
    }
    if (S2CON & S2TI)
    {
        S2CON  &=   ~S2TI;       //clear S2TI
    }
}

```

---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using the level change from high to low of RxD2 pin to wake up Stop/Power-Down mode */
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain <reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

AUXR EQU 08EH //Auxiliary register
S2CON EQU 09AH
S2BUF EQU 09BH
T2H DATA 0D6H
T2L DATA 0D7H
IE2 EQU 0AFH

P_SW2 EQU 0BAH

S2_S EQU 01H //P_SW2.0

S2RI EQU 01H //S2CON.0
S2TI EQU 02H //S2CON.1
S2RB8 EQU 04H //S2CON.2
S2TB8 EQU 08H //S2CON.3

//-----

ORG 0000H
LJMP MAIN

ORG 0043H
LJMP UART2_ISR

//-----

ORG 0100H
MAIN:
MOV SP, #3FH

ANL P_SW2, #NOT S2_S //S2_S=0 (P1.0/RxD2, P1.1/TxD2)
// ORL P_SW2, #S2_S //S2_S=1 (P4.6/RxD2_2, P4.7/TxD2_2)

MOV S2CON, #50H //8-bit variable baud rate
```

---

```

MOV    T2L,    #0D8H           //Setting the reload value of buad rate
                                   //(65536-18432000/4/115200)
MOV    T2H,    #0FFH
MOV    AUXR,   #14H           //T2 in 1T mode, and run Timer 2

ORL    IE2,    #01H           //enable UART1 interrupt
SETB   EA

LOOP:
MOV    PCON,   #02H           //MCU enter Stop/Power-Down mode
NOP                                   //Fisrt implement this statement and then enter interrupt service routine
                                   //after be waked up from Stop/Power-Down mode

NOP
CPL    P1.0
SJMP   LOOP

; /*-----
; UART2 interrupt service Routine
; -----*/
UART2_ISR:
    PUSH    ACC
    PUSH    PSW
    MOV     A,    S2CON
    JNB    ACC.0, CHECKTI       ;check S2RI
    ANL    S2CON, #NOT S2RI     ;clear S2RI
    MOV    P0,   S2BUF

CHECKTI:
    MOV    A,    S2CON
    JNB    ACC.1, ISR_EXIT      ;check S2TI
    ANL    S2CON, #NOT S2TI     ;clear S2TI

ISR_EXIT:
    POP     PSW
    POP     ACC
    RETI

;-----

END

```

---

---

## Chapter 3 Memory Organization and SFRs

The STC15 series MCU has separate address space for Program Memory and Data Memory. The logical separation of program and data memory allows the data memory to be accessed by 8-bit addresses, which can be quickly stored and manipulated by the CPU.

Program memory (ROM) can only be read, not written to. In the STC15 series, all the program memory are on-chip Flash memory, and without the capability of accessing external program memory because of no External Access Enable (/EA) and Program Store Enable (/PSEN) signals designed.

Data memory occupies a separate address space from program memory. There are large capacity of on-chip RAM in STC15 series MCU. For example, the STC15W4K32S4 series implements 4096 bytes of on-chip RAM which consists of 256 bytes of internal scratch-pad RAM and 3840 bytes of on-chip expanded RAM(XRAM). The upper 128 bytes occupy a parallel address space to the Special Function Registers. This means that the upper 128 bytes have the same addresses as the SFR space but are physically separate from SFR space. Besides 64K bytes external expanded RAM also can be accessed in STC15W4K32S4 series MCU.

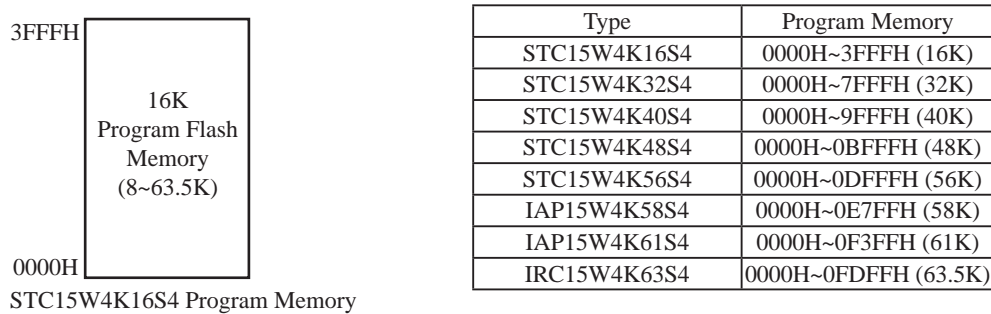
### 3.1 Program Memory

Program memory is the memory which stores the program codes for the CPU to execute. For STC15W4K32S4 series MCU example, there is 16K/32K/40K/48K/56K/58K/61K/63.5K bytes of flash memory embedded for program and data storage. The design allows users to configure it as like there are three individual partition banks inside. They are called AP(application program) region, IAP (In-Application-Program) region and ISP (In-System-Program) boot region. AP region is the space that user program is resided. IAP(In-Application-Program) region is the nonvolatile data storage space that may be used to save important parameters by AP program. In other words, the IAP capability of STC15 provides the user to read/write the user-defined on-chip data flash region to save the needing in use of external EEPROM device. ISP boot region is the space that allows a specific program we call “ISP program” is resided. Inside the ISP region, the user can also enable read/write access to a small memory space to store parameters for specific purposes. Generally, the purpose of ISP program is to fulfill AP program upgrade without the need to remove the device from system. STC15 hardware catches the configuration information since power-up duration and performs out-of-space hardware-protection depending on pre-determined criteria. The criteria is AP region can be accessed by ISP program only, IAP region can be accessed by ISP program and AP program, and ISP region is prohibited access from AP program and ISP program itself. But if the “ISP data flash is enabled”, ISP program can read/write this space. When wrong settings on ISP-IAP SFRs are done, The “out-of-space” happens and STC15 follows the criteria above, ignore the trigger command.

After reset, the CPU begins execution from the location 0000H of Program Memory, where should be the starting of the user’s application code. To service the interrupts, the interrupt service locations (called interrupt vectors) should be located in the program memory. Each interrupt is assigned a fixed location in the program memory. The interrupt causes the CPU to jump to that location, where it commences execution of the service routine. External Interrupt 0, for example, is assigned to location 0003H. If External Interrupt 0 is going to be used, its service routine must begin at location 0003H. If the interrupt is not going to be used, its service location is available as general purpose program memory.

The interrupt service locations are spaced at an interval of 8 bytes: 0003H for External Interrupt 0, 000BH for Timer 0, 0013H for External Interrupt 1, 001BH for Timer 1, etc. If an interrupt service routine is short enough (as is often the case in control applications), it can reside entirely within that 8-byte interval. Longer service routines can use a jump instruction to skip over subsequent interrupt locations, if other interrupts are in use.

Flash memory with flexibility can be repeatedly erased more than 100 thousand times.



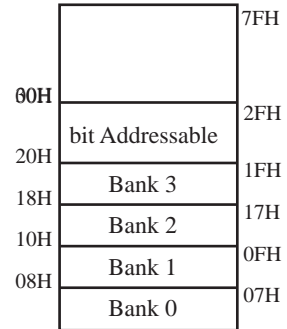
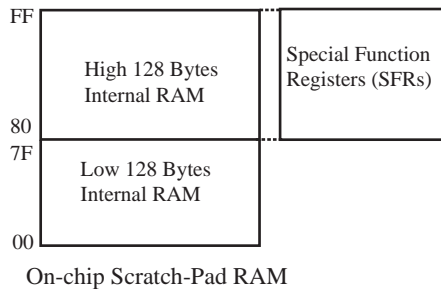
## 3.2 Data Memory (SRAM)

The STC15W4K32S4 series MCU implements 4096 bytes of on-chip RAM which consists of 256 bytes of internal scratch-pad RAM and 3840 bytes of on-chip expanded RAM(XRAM). Besides 64K bytes external expanded RAM also can be accessed in part of STC15 series MCU.

### 3.2.1 On-chip Scratch-Pad RAM

Just as same as the conventional 8051 micro-controller, there are 256 bytes of internal scratch-pad RAM data memory plus 128 bytes of SFR space available on the STC15 series. The lower 128 bytes of data memory may be accessed through both direct and indirect addressing. The upper 128 bytes of data memory and the 128 bytes of SFR space share the same address space. The upper 128 bytes of data memory may only be accessed using indirect addressing. The 128 bytes of SFR can only be accessed through direct addressing. The lowest 32 bytes of data memory are grouped into 4 banks of 8 registers each. Program instructions call out these registers as R0 through R7. The RS0 and RS1 bits in PSW register select which register bank is in use. Instructions using register addressing will only access the currently specified bank. This allows more efficient use of code space, since register instructions are shorter than instructions that use direct addressing. The next 16 bytes (20H~2FH) above the register banks form a block of bit-addressable memory space. The 8051 instruction set includes a wide selection of single-bit instructions, and the 128 bits in this area can be directly addressed by these instructions. The bit addresses in this area are 00H through 7FH.





All of the bytes in the Lower 128 can be accessed by either direct or indirect addressing while the Upper 128 can only be accessed by indirect addressing. SFRs include the Port latches, timers, peripheral controls, etc. These registers can only be accessed by direct addressing. Sixteen addresses in SFR space are both byte- and bit-addressable. The bit-addressable SFRs are those whose address ends in 0H or 8H.

### PSW register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PSW	D0H	name	CY	AC	F0	RS1	RS0	OV	F1	P

CY : Carry flag.

This bit is set when the last arithmetic operation resulted in a carry (addition) or a borrow (subtraction). It is cleared to logic 0 by all other arithmetic operations.

AC : Auxilliary Carry Flag.(For BCD operations)

This bit is set when the last arithmetic operation resulted in a carry into (addition) or a borrow from (subtraction) the high order nibble. It is cleared to logic 0 by all other arithmetic operations

F0 : Flag 0.(Available to the user for general purposes)

RS1: Register bank select control bit 1.

RS0: Register bank select control bit 0.

[RS1 RS0] select which register bank is used during register accesses

RS1	RS0	Working Register Bank(R0~R7) and Address
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.

This bit is set to 1 under the following circumstances:

- An ADD, ADDC, or SUBB instruction causes a sign-change overflow.
- A MUL instruction results in an overflow (result is greater than 255).
- A DIV instruction causes a divide-by-zero condition.

The OV bit is cleared to 0 by the ADD, ADDC, SUBB, MUL, and DIV instructions in all other cases.

---

## PSW register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PSW	D0H	name	CY	AC	F0	RS1	RS0	OV	F1	P

F1 : Flag 1. User-defined flag.

P : Parity flag.

This bit is set to logic 1 if the sum of the eight bits in the accumulator is odd and cleared if the sum is even.

### SP: Stack Pointer.

The Stack Pointer Register is 8 bits wide. It is incremented before data is stored during PUSH and CALL executions. The stack may reside anywhere in on-chip RAM. On reset, the Stack Pointer is initialized to 07H causing the stack to begin at location 08H, which is also the first register (R0) of register bank 1. Thus, if more than one register bank is to be used, the SP should be initialized to a location in the data memory not being used for data storage. The stack depth can extend up to 256 bytes.

## 3.2.2 On-Chip Expanded RAM / XRAM /AUX-RAM

There are 3840 bytes of additional data RAM available on STC15W4K32S4 series. They may be accessed by the instructions MOVX @Ri or MOVX @DPTR. A control bit – EXTRAM located in AUXR.1 register is to control access of auxiliary RAM. When set, disable the access of auxiliary RAM. When clear (EXTRAM=0), this auxiliary RAM is the default target for the address range from 0x0000 to 0x03FF and can be indirectly accessed by move external instruction, “MOVX @Ri” and “MOVX @DPTR”. If EXTRAM=0 and the target address is over 0x03FF, switches to access external RAM automatically. When EXTRAM=0, the content in DPH is ignored when the instruction MOVX @Ri is executed.

For KEIL-C51 compiler, to assign the variables to be located at Auxiliary RAM, the “pdata” or “xdata” definition should be used. After being compiled, the variables declared by “pdata” and “xdata” will become the memories accessed by “MOVX @Ri” and “MOVX @DPTR”, respectively. Thus the STC15W4K32S4 hardware can access them correctly.

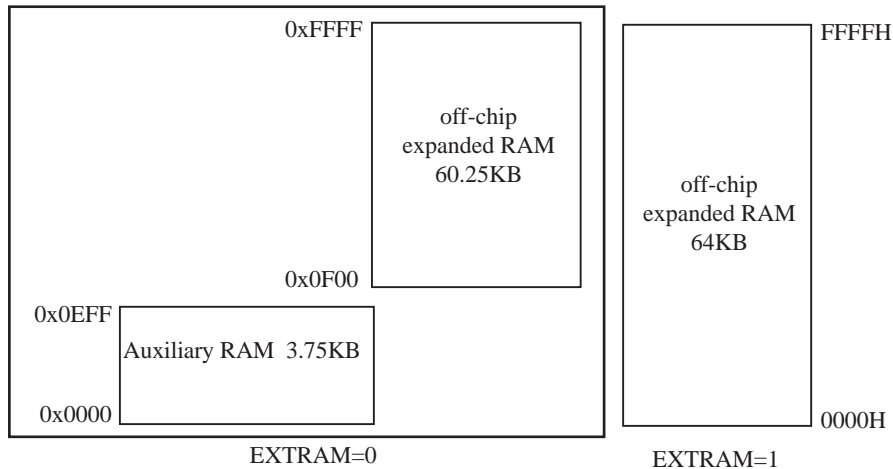


## AUXR register

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR	8EH	Auxiliary Register	T0x12	T1x12	UAR_M0x6	T2R	T2_C/T̄	T2x12	EXTRAM	S1ST2	0000,0001

EXTRAM : Internal / external RAM access control bit.

- 0 : On-chip auxiliary RAM is enabled and located at the address 0x0000 to 0x0EFF.  
For address over 0x0EFF, off-chip expanded RAM becomes the target automatically.
- 1 : On-chip auxiliary RAM is always disabled.



T0x12 : Timer 0 clock source bit.

- 0 : The clock source of Timer 0 is SYSclk/12. It will compatible to the traditional 8051 MCU
- 1 : The clock source of Timer 0 is SYSclk/1. It will drive the T0 faster than a traditional 8051 MCU

T1x12 : Timer 1 clock source bit.

- 0 : The clock source of Timer 1 is SYSclk/12. It will compatible to the traditional 8051 MCU
- 1 : The clock source of Timer 1 is SYSclk/1. It will drive the T0 faster than a traditional 8051 MCU

UART\_M0x6 : Baud rate select bit of UART1 while it is working under 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.

T2R Timer 2 Run control bit

- 0 : not run Timer 2;
- 1 : run Timer 2.

T2\_C/T̄: Counter or timer 2 selector

- 0 : as Timer (namely count on internal system clock)
- 1 : as Counter (namely count on the external pulse input from T2/P3.1)

T2x12 : Timer 2 clock source bit.

- 0 : The clock source of Timer 2 is SYSclk/12.
- 1 : The clock source of Timer 2 is SYSclk/1.

S1ST2 : the control bit that UART1 select Timer 2 as its baud-rate generator.

- 0 : Select Timer 1 as the baud-rate generator of UART1
- 1 : Select Timer 2 as the baud-rate generator of UART1. Timer 1 is released to use in other functions.

---

## An example program for internal expanded RAM demo of STC15 series:

```
;/*-----*/
;/* --- STC MCU International Limited -----*/
;/* --- STC 1T Series MCU internal expanded RAM Demo -----*/
;/* If you want to use the program or the program referenced in the */
;/* article, please specify in which data and procedures from STC */
;/*-----*/
#include<reg51.h>
#include<intrins.h>          /* use _nop_() function */

sfr      AUXR = 0x8e;

sbit     ERROM_LED = P1^5;
sbit     OK_LED = P1^7;

void main ()
{
    unsigned int array_point = 0;

    /*Test-array: Test_array_one[512], Test_array_two[512] */
    unsigned char xdata Test_array_one[512] =
    {
        0x00,  0x01,  0x02,  0x03,  0x04,  0x05,  0x06,  0x07,
        0x08,  0x09,  0x0a,  0x0b,  0x0c,  0x0d,  0x0e,  0x0f,
        0x10,  0x11,  0x12,  0x13,  0x14,  0x15,  0x16,  0x17,
        0x18,  0x19,  0x1a,  0x1b,  0x1c,  0x1d,  0x1e,  0x1f,
        0x20,  0x21,  0x22,  0x23,  0x24,  0x25,  0x26,  0x27,
        0x28,  0x29,  0x2a,  0x2b,  0x2c,  0x2d,  0x2e,  0x2f,
        0x30,  0x31,  0x32,  0x33,  0x34,  0x35,  0x36,  0x37,
        0x38,  0x39,  0x3a,  0x3b,  0x3c,  0x3d,  0x3e,  0x3f,
        0x40,  0x41,  0x42,  0x43,  0x44,  0x45,  0x46,  0x47,
        0x48,  0x49,  0x4a,  0x4b,  0x4c,  0x4d,  0x4e,  0x4f,
        0x50,  0x51,  0x52,  0x53,  0x54,  0x55,  0x56,  0x57,
        0x58,  0x59,  0x5a,  0x5b,  0x5c,  0x5d,  0x5e,  0x5f,
        0x60,  0x61,  0x62,  0x63,  0x64,  0x65,  0x66,  0x67,
        0x68,  0x69,  0x6a,  0x6b,  0x6c,  0x6d,  0x6e,  0x6f,
        0x70,  0x71,  0x72,  0x73,  0x74,  0x75,  0x76,  0x77,
        0x78,  0x79,  0x7a,  0x7b,  0x7c,  0x7d,  0x7e,  0x7f,
        0x80,  0x81,  0x82,  0x83,  0x84,  0x85,  0x86,  0x87,
        0x88,  0x89,  0x8a,  0x8b,  0x8c,  0x8d,  0x8e,  0x8f,
        0x90,  0x91,  0x92,  0x93,  0x94,  0x95,  0x96,  0x97,
        0x98,  0x99,  0x9a,  0x9b,  0x9c,  0x9d,  0x9e,  0x9f,
        0xa0,  0xa1,  0xa2,  0xa3,  0xa4,  0xa5,  0xa6,  0xa7,
        0xa8,  0xa9,  0xaa,  0xab,  0xac,  0xad,  0xae,  0xaf,
```

---

```

0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7,
0xb8, 0xb9, 0xba, 0xbb, 0xbc, 0xbd, 0xbe, 0xbf,
0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7,
0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf,
0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7,
0xd8, 0xd9, 0xda, 0xdb, 0xdc, 0xdd, 0xde, 0xdf,
0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7,
0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef,
0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff,
0xff, 0xfe, 0xfd, 0xfc, 0xfb, 0xfa, 0xf9, 0xf8,
0xf7, 0xf6, 0xf5, 0xf4, 0xf3, 0xf2, 0xf1, 0xf0,
0xef, 0xee, 0xed, 0xec, 0xeb, 0xea, 0xe9, 0xe8,
0xe7, 0xe6, 0xe5, 0xe4, 0xe3, 0xe2, 0xe1, 0xe0,
0xdf, 0xde, 0xdd, 0xdc, 0xdb, 0xda, 0xd9, 0xd8,
0xd7, 0xd6, 0xd5, 0xd4, 0xd3, 0xd2, 0xd1, 0xd0,
0xcf, 0xce, 0xcd, 0xcc, 0xcb, 0xca, 0xc9, 0xc8,
0xc7, 0xc6, 0xc5, 0xc4, 0xc3, 0xc2, 0xc1, 0xc0,
0xbf, 0xbe, 0xbd, 0xbc, 0xbb, 0xba, 0xb9, 0xb8,
0xb7, 0xb6, 0xb5, 0xb4, 0xb3, 0xb2, 0xb1, 0xb0,
0xaf, 0xae, 0xad, 0xac, 0xab, 0xaa, 0xa9, 0xa8,
0xa7, 0xa6, 0xa5, 0xa4, 0xa3, 0xa2, 0xa1, 0xa0,
0x9f, 0x9e, 0x9d, 0x9c, 0x9b, 0x9a, 0x99, 0x98,
0x97, 0x96, 0x95, 0x94, 0x93, 0x92, 0x91, 0x90,
0x8f, 0x8e, 0x8d, 0x8c, 0x8b, 0x8a, 0x89, 0x88,
0x87, 0x86, 0x85, 0x84, 0x83, 0x82, 0x81, 0x80,
0x7f, 0x7e, 0x7d, 0x7c, 0x7b, 0x7a, 0x79, 0x78,
0x77, 0x76, 0x75, 0x74, 0x73, 0x72, 0x71, 0x70,
0x6f, 0x6e, 0x6d, 0x6c, 0x6b, 0x6a, 0x69, 0x68,
0x67, 0x66, 0x65, 0x64, 0x63, 0x62, 0x61, 0x60,
0x5f, 0x5e, 0x5d, 0x5c, 0x5b, 0x5a, 0x59, 0x58,
0x57, 0x56, 0x55, 0x54, 0x53, 0x52, 0x51, 0x50,
0x4f, 0x4e, 0x4d, 0x4c, 0x4b, 0x4a, 0x49, 0x48,
0x47, 0x46, 0x45, 0x44, 0x43, 0x42, 0x41, 0x40,
0x3f, 0x3e, 0x3d, 0x3c, 0x3b, 0x3a, 0x39, 0x38,
0x37, 0x36, 0x35, 0x34, 0x33, 0x32, 0x31, 0x30,
0x2f, 0x2e, 0x2d, 0x2c, 0x2b, 0x2a, 0x29, 0x28,
0x27, 0x26, 0x25, 0x24, 0x23, 0x22, 0x21, 0x20,
0x1f, 0x1e, 0x1d, 0x1c, 0x1b, 0x1a, 0x19, 0x18,
0x17, 0x16, 0x15, 0x14, 0x13, 0x12, 0x11, 0x10,
0x0f, 0x0e, 0x0d, 0x0c, 0x0b, 0x0a, 0x09, 0x08,
0x07, 0x06, 0x05, 0x04, 0x03, 0x02, 0x01, 0x00
};

```

```

unsigned char xdata Test_array_two[512] =
{
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
    0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,

```

---

0x10,	0x11,	0x12,	0x13,	0x14,	0x15,	0x16,	0x17,
0x18,	0x19,	0x1a,	0x1b,	0x1c,	0x1d,	0x1e,	0x1f,
0x20,	0x21,	0x22,	0x23,	0x24,	0x25,	0x26,	0x27,
0x28,	0x29,	0x2a,	0x2b,	0x2c,	0x2d,	0x2e,	0x2f,
0x30,	0x31,	0x32,	0x33,	0x34,	0x35,	0x36,	0x37,
0x38,	0x39,	0x3a,	0x3b,	0x3c,	0x3d,	0x3e,	0x3f,
0x40,	0x41,	0x42,	0x43,	0x44,	0x45,	0x46,	0x47,
0x48,	0x49,	0x4a,	0x4b,	0x4c,	0x4d,	0x4e,	0x4f,
0x50,	0x51,	0x52,	0x53,	0x54,	0x55,	0x56,	0x57,
0x58,	0x59,	0x5a,	0x5b,	0x5c,	0x5d,	0x5e,	0x5f,
0x60,	0x61,	0x62,	0x63,	0x64,	0x65,	0x66,	0x67,
0x68,	0x69,	0x6a,	0x6b,	0x6c,	0x6d,	0x6e,	0x6f,
0x70,	0x71,	0x72,	0x73,	0x74,	0x75,	0x76,	0x77,
0x78,	0x79,	0x7a,	0x7b,	0x7c,	0x7d,	0x7e,	0x7f,
0x80,	0x81,	0x82,	0x83,	0x84,	0x85,	0x86,	0x87,
0x88,	0x89,	0x8a,	0x8b,	0x8c,	0x8d,	0x8e,	0x8f,
0x90,	0x91,	0x92,	0x93,	0x94,	0x95,	0x96,	0x97,
0x98,	0x99,	0x9a,	0x9b,	0x9c,	0x9d,	0x9e,	0x9f,
0xa0,	0xa1,	0xa2,	0xa3,	0xa4,	0xa5,	0xa6,	0xa7,
0xa8,	0xa9,	0xaa,	0xab,	0xac,	0xad,	0xae,	0xaf,
0xb0,	0xb1,	0xb2,	0xb3,	0xb4,	0xb5,	0xb6,	0xb7,
0xb8,	0xb9,	0xba,	0xbb,	0xbc,	0xbd,	0xbe,	0xbf,
0xc0,	0xc1,	0xc2,	0xc3,	0xc4,	0xc5,	0xc6,	0xc7,
0xc8,	0xc9,	0xca,	0xcb,	0xcc,	0xcd,	0xce,	0xcf,
0xd0,	0xd1,	0xd2,	0xd3,	0xd4,	0xd5,	0xd6,	0xd7,
0xd8,	0xd9,	0xda,	0xdb,	0xdc,	0xdd,	0xde,	0xdf,
0xe0,	0xe1,	0xe2,	0xe3,	0xe4,	0xe5,	0xe6,	0xe7,
0xe8,	0xe9,	0xea,	0xeb,	0xec,	0xed,	0xee,	0xef,
0xf0,	0xf1,	0xf2,	0xf3,	0xf4,	0xf5,	0xf6,	0xf7,
0xf8,	0xf9,	0xfa,	0xfb,	0xfc,	0xfd,	0xfe,	0xff,
0xff,	0xfe,	0xfd,	0xfc,	0xfb,	0xfa,	0xf9,	0xf8,
0xf7,	0xf6,	0xf5,	0xf4,	0xf3,	0xf2,	0xf1,	0xf0,
0xef,	0xee,	0xed,	0xec,	0xeb,	0xea,	0xe9,	0xe8,
0xe7,	0xe6,	0xe5,	0xe4,	0xe3,	0xe2,	0xe1,	0xe0,
0xdf,	0xde,	0xdd,	0xdc,	0xdb,	0xda,	0xd9,	0xd8,
0xd7,	0xd6,	0xd5,	0xd4,	0xd3,	0xd2,	0xd1,	0xd0,
0xcf,	0xce,	0xcd,	0xcc,	0xcb,	0xca,	0xc9,	0xc8,
0xc7,	0xc6,	0xc5,	0xc4,	0xc3,	0xc2,	0xc1,	0xc0,
0xbf,	0xbe,	0xbd,	0xbc,	0xbb,	0xba,	0xb9,	0xb8,
0xb7,	0xb6,	0xb5,	0xb4,	0xb3,	0xb2,	0xb1,	0xb0,
0xaf,	0xae,	0xad,	0xac,	0xab,	0xaa,	0xa9,	0xa8,
0xa7,	0xa6,	0xa5,	0xa4,	0xa3,	0xa2,	0xa1,	0xa0,
0x9f,	0x9e,	0x9d,	0x9c,	0x9b,	0x9a,	0x99,	0x98,
0x97,	0x96,	0x95,	0x94,	0x93,	0x92,	0x91,	0x90,
0x8f,	0x8e,	0x8d,	0x8c,	0x8b,	0x8a,	0x89,	0x88,
0x87,	0x86,	0x85,	0x84,	0x83,	0x82,	0x81,	0x80,
0x7f,	0x7e,	0x7d,	0x7c,	0x7b,	0x7a,	0x79,	0x78,
0x77,	0x76,	0x75,	0x74,	0x73,	0x72,	0x71,	0x70,

---

```

0x6f, 0x6e, 0x6d, 0x6c, 0x6b, 0x6a, 0x69, 0x68,
0x67, 0x66, 0x65, 0x64, 0x63, 0x62, 0x61, 0x60,
0x5f, 0x5e, 0x5d, 0x5c, 0x5b, 0x5a, 0x59, 0x58,
0x57, 0x56, 0x55, 0x54, 0x53, 0x52, 0x51, 0x50,
0x4f, 0x4e, 0x4d, 0x4c, 0x4b, 0x4a, 0x49, 0x48,
0x47, 0x46, 0x45, 0x44, 0x43, 0x42, 0x41, 0x40,
0x3f, 0x3e, 0x3d, 0x3c, 0x3b, 0x3a, 0x39, 0x38,
0x37, 0x36, 0x35, 0x34, 0x33, 0x32, 0x31, 0x30,
0x2f, 0x2e, 0x2d, 0x2c, 0x2b, 0x2a, 0x29, 0x28,
0x27, 0x26, 0x25, 0x24, 0x23, 0x22, 0x21, 0x20,
0x1f, 0x1e, 0x1d, 0x1c, 0x1b, 0x1a, 0x19, 0x18,
0x17, 0x16, 0x15, 0x14, 0x13, 0x12, 0x11, 0x10,
0x0f, 0x0e, 0x0d, 0x0c, 0x0b, 0x0a, 0x09, 0x08,
0x07, 0x06, 0x05, 0x04, 0x03, 0x02, 0x01, 0x00
};

ERROR_LED = 1;
OK_LED = 1;
for (array_point = 0; array_point<512; array_point++)
{
    if (Test_array_one[array_point] != Test_array_two [array_point])
    {
        ERROR_LED = 0;
        OK_LED = 1;
        break;
    }
    else{
        OK_LED = 0;
        ERROR_LED = 1;
    }
}
while (1);
}

```

### 3.2.3 External Expandable 64KB RAM (Off-Chip RAM)

There is 64K-byte addressing space available for STC15W4K32S4 to access external data RAM. Just the same as the design in the conventional 8051, the port – P2, P0, ALE/P4.5, P4.2/WR and P4.4/RD have alternative function for external data RAM access. In addition, a new register BUS\_SPEED (address: 0xA1) is design to control the access timing of "MOVX" instruction. By using BUS\_SPEED to change the instruction cycle time, STC15 series MCU can conformed to communicate with both of fast and slow peripheral devices without loss of communication efficiency.

#### BUS\_SPEED register

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
BUS_SPEED	A1H	Bus-Speed Control	-	-	-	-	-	-	EXRTS[1:0]		xxxx,xx10

#### EXRTS (Extend Ram Timing Selector)

0 0	: Setup / Hold / Read and Write Duty	1 clock cycle;	EXRAC	1
0 1	: Setup / Hold / Read and Write Duty	2 clock cycle;	EXRAC	2
1 0	: Setup / Hold / Read and Write Duty	4 clock cycle;	EXRAC	4
1 1	: Setup / Hold / Read and Write Duty	8 clock cycle;	EXRAC	8

When the target is on-chip auxiliary RAM, the setting on BUS\_SPEED register is discarded by hardware.

#### AUXR register

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR	8EH	Auxiliary Register	T0x12	T1x12	UAR_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1S2	0000,0001

EXTRAM : Internal / external RAM access control bit.

- 0 : On-chip auxiliary RAM is enabled and located at the address 0x0000 to 0x0EFF.  
For address over 0x0EFF, off-chip expanded RAM becomes the target automatically.
- 1 : On-chip auxiliary RAM is always disabled.

Mnemonic	Description	Execution Clocks	Condition (Take STC15W4K32S4 for example, namely on-chip expanded RAM is 3840 byte)
MOVX @DPTR, A	Move Acc to on-chip expanded RAM (16-bit addr). Write operation.	3	the content of DPTR is 0000H ~ 0EFFH (3840 bytes=[4096-256])
MOVX A, @DPTR	Move on-chip expanded RAM(16-bit addr) to Acc. Read operation.	2	the content of DPTR is 0000H ~ 0EFFH (3840 bytes=[4096-256])
MOVX @Ri, A	Move Acc to on-chip expanded RAM(8-bit addr). Write operation.	4	EXTRAM=0
MOVX A, @Ri	Move on-chip expanded RAM(8-bit addr) to Acc. Read operation	3	EXTRAM=0



Mnemonic	Description	Execution Clocks	Condition (Take STC15W4K32S4 for example, namely on-chip expanded RAM is 3840 byte)
MOVX @Ri, A	Move Acc to External RAM(8-bit addr). Write operation.	8	EXRTS[1:0] = [0,0], EXTRAM=1
MOVX A, @Ri	Move Acc to External RAM(8-bit addr). Read operation.	7	EXRTS[1:0] = [0,0], EXTRAM=1
MOVX @Ri, A	Move Acc to External RAM(8-bit addr). Write operation.	13	EXRTS[1:0] = [0,1], EXTRAM=1
MOVX A, @Ri	Move Acc to External RAM(8-bit addr). Read operation.	12	EXRTS[1:0] = [0,1], EXTRAM=1
MOVX @Ri, A	Move Acc to External RAM(8-bit addr). Write operation.	23	EXRTS[1:0] = [1,0], EXTRAM=1
MOVX A, @Ri	Move Acc to External RAM(8-bit addr). Read operation.	22	EXRTS[1:0] = [1,0], EXTRAM=1
MOVX @Ri, A	Move Acc to External RAM(8-bit addr). Write operation.	43	EXRTS[1:0] = [1,1], EXTRAM=1
MOVX A, @Ri	Move Acc to External RAM(8-bit addr). Read operation.	42	EXRTS[1:0] = [1,1], EXTRAM=1

Note: Ri means R1 and R0 in above table.

Mnemonic	Description	Execution Clocks	Condition (Take STC15W4K32S4 for example, namely on-chip expanded RAM is 3840 byte)
MOVX @DPTR, A	Move Acc to External RAM (16-bit addr). Write operation.	7	EXRTS[1:0] = [0,0], DPTR>=3840 namely (4096-256) or EXTRAM=1
MOVX A, @DPTR	Move External RAM(16-bit addr) to Acc. Read operation.	6	EXRTS[1:0] = [0,0], DPTR>=3840 namely (4096-256) or EXTRAM=1
MOVX @DPTR, A	Move Acc to External RAM (16-bit addr). Write operation.	12	EXRTS[1:0] = [0,1], DPTR>=3840 namely (4096-256) or EXTRAM=1
MOVX A, @DPTR	Move External RAM(16-bit addr) to Acc. Read operation.	11	EXRTS[1:0] = [0,1], DPTR>=3840 namely (4096-256) or EXTRAM=1
MOVX @DPTR, A	Move Acc to External RAM (16-bit addr). Write operation.	22	EXRTS[1:0] = [1,0], DPTR>=3840 namely (4096-256) or EXTRAM=1
MOVX A, @DPTR	Move External RAM(16-bit addr) to Acc. Read operation.	21	EXRTS[1:0] = [1,0], DPTR>=3840 namely (4096-256) or EXTRAM=1
MOVX @DPTR, A	Move Acc to External RAM (16-bit addr). Write operation.	42	EXRTS[1:0] = [1,1], DPTR>=3840 namely (4096-256) or EXTRAM=1

The execution clocks of accessing external RAM is computed as the following formula

MOVX @R0/R1

write :  $5 \times N + 3$

read :  $5 \times N + 2$

MOVX @DPTR

write :  $5 \times N + 2$

read :  $5 \times N + 1$

When EXRTS[1:0] = [0,0], N=1 in above formula;

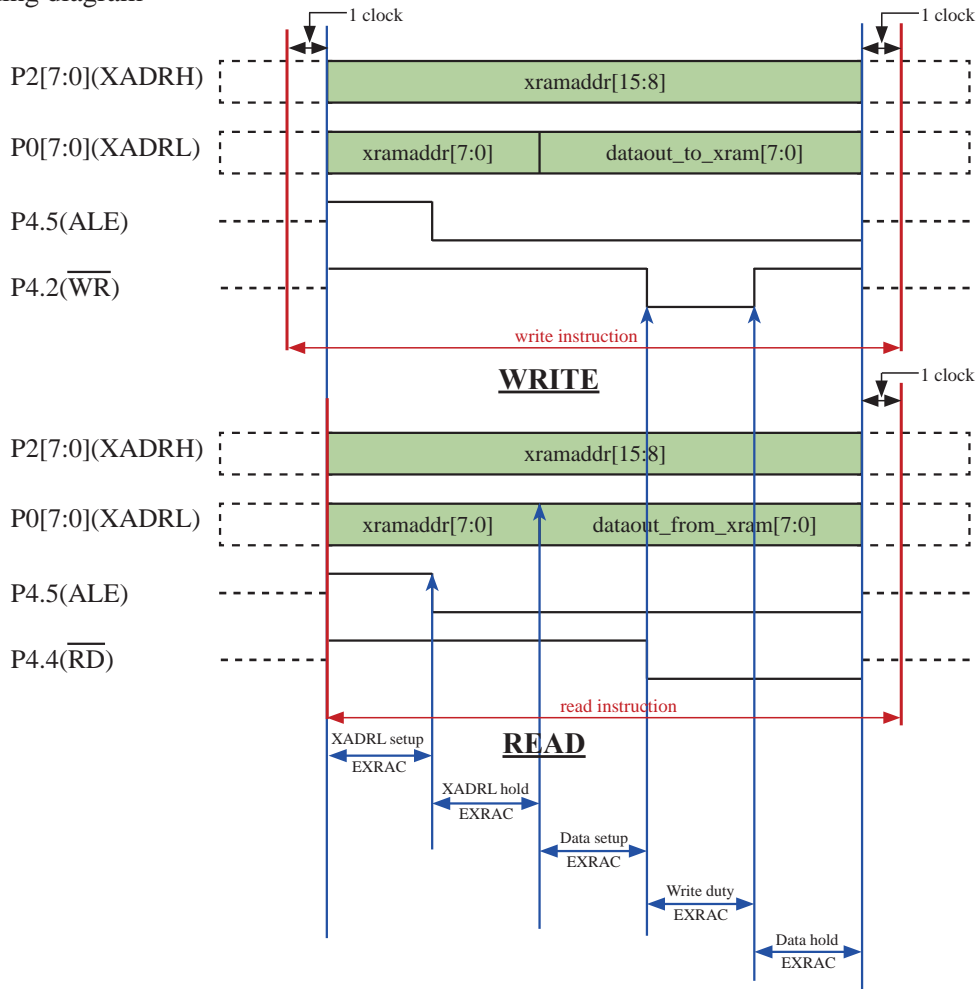
When EXRTS[1:0] = [0,1], N=2 in above formula;

When EXRTS[1:0] = [1,0], N=4 in above formula;

When EXRTS[1:0] = [1,1], N=8 in above formula;

Thus it can be seen that the speed of instruction accessing external RAM is adjustable for STC15 series MCU.

Timing diagram



## 3.3 Special Function Registers

### 3.3.1 Special Function Registers Address Map

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
0F8H	P7 1111,1111	CH 0000,0000	CCAP0H 0000,0000	CCAP1H 0000,0000	CCAP2H 0000,0000				0FFH
0F0H	B 0000,0000	PWMCFG 0000,0000	PCA_PWM0 00xx,xx00	PCA_PWM1 00xx,xx00	PCA_PWM2 00xx,xx00	PWMCR 0000,0000	PWMIF 0000,0000	PWMFDCR 0000,0000	0F7H
0E8H	P6 1111,1111	CL 0000,0000	CCAP0L 0000,0000	CCAP1L 0000,0000	CCAP2L 0000,0000				0EFH
0E0H	ACC 0000,0000	P7M1 0000,0000	P7M0 0000,0000				CMPCR1 0000,0000	CMPCR2 0000,1001	0E7H
0D8H	CCON 00xx,0000	CMOD 0xxx,x000	CCAPM0 x000,0000	CCAPM1 x000,0000	CCAPM2 x000,0000				0DFH
0D0H	PSW 0000,00x0	T4T3M 0000,0000	T4H RL_TH4 0000,0000	T4L RL_TL4 0000,0000	T3H RL_TH3 0000,0000	T3L RL_TL3 0000,0000	T2H RL_TH2 0000,0000	T2L RL_TL2 0000,0000	0D7H
0C8H	P5 xxxx,1111	P5M1 xxxx,0000	P5M0 xxxx,0000	P6M1 0000,0000	P6M0 0000,0000	SPSTAT 00xx,xxxx	SPCTL 0000,0100	SPDAT 0000,0000	0CFH
0C0H	P4 1111,1111	WDT_CONTR 0x00,0000	IAP_DATA 1111,1111	IAP_ADDRH 0000,0000	IAP_ADDRL 0000,0000	IAP_CMD xxxx,xx00	IAP_TRIG xxxx,xxxx	IAP_CONTR 0000,0000	0C7H
0B8H	IP x0x0,0000	SADEN	P_SW2 xxxx,x000		ADC_CONTR 0000,0000	ADC_RES 0000,0000	ADC_RESL 0000,0000		0BFH
0B0H	P3 1111,1111	P3M1 0000,0000	P3M0 0000,0000	P4M1 0000,0000	P4M0 0000,0000	IP2 xxx0,0000	IP2H 0000,0000	IPH 0000,0000	0B7H
0A8H	IE 0000,0000	SADDR	WKTCL WKTCL_CNT 0111 1111	WKTCH WKTCH_CNT 0111 1111	S3CON 0000,0000	S3BUF xxxx,xxxx		IE2 x000,0000	0AFH
0A0H	P2 1111,1111	BUS_SPEED xxxx,xx10	AUXR1 P_SW1 0100,0000	Don't use	Don't use	Don't use		Don't use	0A7H
098H	SCON 0000,0000	SBUF xxxx,xxxx	S2CON 0100,0000	S2BUF xxxx,xxxx	Don't use	P1ASF 0000,0000	Don't use	Don't use	09FH
090H	P1 1111,1111	P1M1 0000,0000	P1M0 0000,0000	P0M1 0000,0000	P0M0 0000,0000	P2M1 0000,0000	P2M0 0000,0000	CLK_DIV PCON2 0000,0000	097H
088H	TCON 0000,0000	TMOD 0000,0000	TL0 RL_TL0 0000,0000	TL1 RL_TL1 0000,0000	TH0 RL_TH0 0000,0000	TH1 RL_TH1 0000,0000	AUXR 0000,0001	INT_CLKO AUXR2 0000,0000	08FH
080H	P0 1111,1111	SP 0000,0111	DPL 0000,0000	DPH 0000,0000	S4CON 0000,0000	S4BUF xxxx,xxxx		PCON 0011,0000	087H

### 3.3.2 Special Function Registers Bits Description

Symbol	Description	Address	Bit Address and Symbol								Value after Power-on or Reset
			MSB				LSB				
P0	Port 0	80H	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	1111 1111B
SP	Stack Pointer	81H									0000 0111B
DPTR	DPL	Data Pointer Low	82H								0000 0000B
	DPH	Data Pointer High	83H								0000 0000B
S4CON	S4 Control	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000B
S4BUF	S4 Serial Buffer	85H									xxxx,xxxxB
PCON	Power Control	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011 0000B
TCON	Timer Control	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000 0000B
TMOD	Timer Mode	89H	GATE	C $\bar{T}$	M1	M0	GATE	C $\bar{T}$	M1	M0	0000 0000B
TL0	Timer Low 0	8AH									0000 0000B
TL1	Timer Low 1	8BH									0000 0000B
TH0	Timer High 0	8CH									0000 0000B
TH1	Timer High 1	8DH									0000 0000B
AUXR	Auxiliary register	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C $\bar{T}$	T2x12	EXTRAM	S1ST2	0000 0001B
INT_CLKO AUXR2	CLK_Output and External Interrupt enable register	8FH	-	EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO	x000 0000B
P1	Port 1	90H	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	1111 1111B
P1M1	P1 configuration 1	91H									0000 0000B
P1M0	P1 configuration 0	92H									0000 0000B
P0M1	P0 configuration 1	93H									0000 0000B
P0M0	P0 configuration 0	94H									0000 0000B
P2M1	P2 configuration 1	95H									0000 0000B
P2M0	P2 configuration 0	96H									0000 0000B
CLK_DIV PCON2	Clock Divder	97H	MCKO_S1	MCKO_S1	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000B
SCON	Serial Control	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000 0000B
SBUF	Serial Buffer	99H									xxxx xxxxB
S2CON	S2 Control	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100 0000B
S2SBUF	S2 Serial Buffer	9BH									xxxx xxxxB
P1ASF	P1 Analog Special Function	9DH	P17ASF	P16ASF	P15ASF	P14ASF	P13ASF	P12ASF	P11ASF	P10ASF	0000 0000B
P2	Port 2	A0H	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	1111 1111B
BUS_SPEED	Bus-Speed Control	A1H	-	-	-	-	-	-	EXRTS[1:0]		xxxx xx10B
AUXR1 P_SW1	Auxiliary register1	A2H	S1_S1	S1_S0	CCP_S1	CCP_S0	SPL_S1	SPL_S0	0	DPS	0100 0000B
IE	Interrupt Enable	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000 0000B

Symbol	Description	Address	Bit Address and Symbol								Value after Power-on or Reset
			MSB				LSB				
SADDR	Slave Address	A9H									0000 0000B
WKTCL WKTCL_CNT	Power-Down Wake-up Timer Control register low	AAH									1111 1111B
WKTCH WKTCH_CNT	Power-Down Wake-up Timer Control register high	ABH	WKTEN								0111 1111B
S3CON	S3 Control	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000B
S3BUF	S3 Serial Buffer	ADH									xxxx,xxxxB
IE2	Interrupt Enable 2	AFH		ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000 0000B
P3	Port 3	B0H	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0	1111 1111B
P3M1	P3 configuration 1	B1H									0000 0000B
P3M0	P3 configuration 0	B2H									0000 0000B
P4M1	P4 configuration 1	B3H									0000 0000B
P4M0	P4 configuration 0	B4H									0000 0000B
IP2	2rd Interrupt Priority Low register	B5H	-	-	-	PX4	PPWMFD	PPWM	PSPI	PS2	xxx0 0000B
IP	Interrupt Priority Low	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000 0000B
SADEN	Slave Address Mask	B9H									0000 0000B
P_SW2	Peripheral Function Switch register 2	BAH	EAXSFR	0	0	0	-	S4_S	S3_S	S2_S	0000 x000B
ADC_CONTR	ADC Control	BCH	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHIS0	0000 0000B
ADC_RES	ADC Result	BDH									0000 0000B
ADC_RESL	ADC Result Low	BEH									0000 0000B
P4	Port 4	C0H	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0	1111 1111B
WDT_CONTR	Watch-Dog-Timer Control Register	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDLE_WDT	PS2	PS1	PS0	xx00 0000B
IAP_DATA	ISP/IAP Flash Data Register	C2H									1111 1111B
IAP_ADDRH	ISP/IAP Flash Address High	C3H									0000 0000B
IAP_ADDRL	ISP/IAP Flash Address Low	C4H									0000 0000B
IAP_CMD	ISP/IAP Flash Command Register	C5H	-	-	-	-	-	-	MS1	MS0	xxxx x000B
IAP_TRIG	ISP/IAP Flash Command Trigger	C6H									xxxx xxxxB
IAP_CONTR	ISP/IAP Control Register	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	WT2	WT1	WT0	0000 x000B
P5	Port 5	C8H	-	-	P5.5	P5.4	P5.3	P5.2	P5.1	P5.0	xxxx 1111B
P5M1	P5 Configuration 1	C9H									0000 0000B
P5M0	P5 Configuration 0	CAH									0000 0000B

Symbol	Description	Address	Bit Address and Symbol								Value after Power-on or Reset
			MSB				LSB				
P6M1	P6 Configuration 1	CBH									
P6M0	P6 Configuration 0	CCH									
SPSTAT	SPI Status register	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx xxxxB
SPCTL	SPI control register	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	0000 0100B
SPDAT	SPI Data register	CFH	-	-	-	-	-	-	-	-	0000 0000B
PSW	Program Status Word	D0H	CY	AC	F0	RS1	RS0	OV	F1	P	0000 0000B
T4T3M	T4 and T3 mode register	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000 0000B
T4H	Timer 4 high 8-bit register	D2H									0000 0000B
T4L	Timer 4 low 8-bit register	D3H									0000 0000B
T3H	Timer 3 high 8-bit register	D4H									0000 0000B
T3L	Timer 3 low 8-bit register	D5H									0000 0000B
T2H	Timer 2 high 8-bit register	D6H									0000 0000B
T2L	Timer 2 low 8-bit register	D7H									0000 0000B
CCON	PCA Control Register	D8H	CF	CR	-	-	CCF3	CCF2	CCF1	CCF0	00xx 0000B
CMOD	PCA Mode Register	D9H	CIDL	-	-	-	CPS2	CPS1	CPS0	ECF	00xx 0000B
CCAPM0	PCA Module 0 Mode Register	DAH	-	ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0	x000 0000B
CCAPM1	PCA Module 1 Mode Register	DBH	-	ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1	x000 0000B
ACC	Accumulator	E0H									0000 0000B
P7M1	P7 configuration 1	E1H									
P7M0	P7 configuration 0	E2H									
P6	Port 6	E8H									
CL	PCA Base Timer Low	E9H									0000 0000B
CCAP0L	PCA module 0 capture register low	EAH									0000 0000B
CCAP1L	PCA module 1 capture register low	EBH									0000 0000B
B	B Register	F0H									0000 0000B
PCA_PWM0	PCA PWM Mode Auxiliary Register 0	F2H	EBS0_1	EBS0_0	-	-	-	-	EPC0H	EPC0L	xxxx xx00B
PCA_PWM1	PCA PWM Mode Auxiliary Register 1	F3H	EBS1_1	EBS1_0	-	-	-	-	EPC1H	EPC1L	xxxx xx00B

Symbol	Description	Address	Bit Address and Symbol								Value after Power-on or Reset
			MSB							LSB	
CH	PCA Base Timer High	F9H									0000 0000B
CCAP0H	PCA Module-0 Capture Register High	FAH									0000 0000B
CCAP1H	PCA Module-1 Capture Register High	FBH									0000 0000B

### Extended Special Function Registers

Symbol	Description	Add.	Bit Address and Symbol								Value after Power-on or Reset
			B7	B6	B5	B4	B3	B2	B1	B0	
PWMCFG	PWM Configure register	F1H	-	CBTADC	C7INI	C6INI	C5INI	C4INI	C3INI	C2INI	0000,0000
PWMCR	PWM Control register	F5H	ENPWM	ECBI	ENC7O	ENC6O	ENC5O	ENC4O	ENC3O	ENC2O	0000,0000
PWMIF	PWM Interrupt Flag register	F6H	-	CBIF	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	x000,0000
PWMFDCR	PWM F_ception Detection Control Register	F7H	-	-	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	xx00,0000
PWMCH	PWM Counter High	FFF0H	-	PWMCH[14:8]							x000,0000
PWMCL	PWM Counter low	FFF1H	PWMCL[7:0]								0000,0000
PWMCKS	PWM Clock Selection register	FFF2H	-	-	-	SELT2	PS[3:0]				xxx0,0000
PWM2T1H	Timer 1 of PWM2 High	FF00H	-	PWM2T1H[14:8]							x000,0000
PWM2T1L	Timer 1 of PWM2 Low	FF01H	PWM2T1L[7:0]								0000,0000
PWM2T2H	Timer 2 of PWM2 High	FF02H	-	PWM2T2H[14:8]							x000,0000
PWM2T2L	Timer 2 of PWM2 Low	FF03H	PWM2T2L[7:0]								0000,0000
PWM2CR	PWM2 Control register	FF04H	-	-	-	-	PWM2_PS	EPWM2I	EC2T2SI	EC2T1SI	xxxx,0000
PWM3T1H	Timer 1 of PWM3 High	FF10H	-	PWM3T1H[14:8]							x000,0000
PWM3T1L	Timer 1 of PWM3 Low	FF11H	PWM3T1L[7:0]								0000,0000
PWM3T2H	Timer 2 of PWM3 High	FF12H	-	PWM3T2H[14:8]							x000,0000
PWM3T2L	Timer 2 of PWM3 Low	FF13H	PWM3T2L[7:0]								0000,0000
PWM3CR	PWM3 Control register	FF14H	-	-	-	-	PWM3_PS	EPWM3I	EC3T2SI	EC3T1SI	xxxx,0000
PWM4T1H	Timer 1 of PWM4 High	FF20H	-	PWM4T1H[14:8]							x000,0000
PWM4T1L	Timer 1 of PWM4 Low	FF21H	PWM4T1L[7:0]								0000,0000
PWM4T2H	Timer 2 of PWM4 High	FF22H	-	PWM4T2H[14:8]							x000,0000
PWM4T2L	Timer 2 of PWM4 Low	FF23H	PWM4T2L[7:0]								0000,0000

Extended Special Function Registers (continued)

Symbol	Description	Add.	Bit Address and Symbol								Value after Power-on or Reset
			B7	B6	B5	B4	B3	B2	B1	B0	
PWM4CR	PWM4 Control register	FF24H	-	-	-	-	PWM4_PS	EPWM4I	EC4T2SI	EC4T1SI	xxxx,0000
PWM5T1H	Timer 1 of PWM5 High	FF30H	-	PWM5T1H[14:8]							x000,0000
PWM5T1L	Timer 1 of PWM5 Low	FF31H	PWM5T1L[7:0]							0000,0000	
PWM5T2H	Timer 2 of PWM5 High	FF32H	-	PWM5T2H[14:8]							x000,0000
PWM5T2L	Timer 2 of PWM5 Low	FF33H	PWM5T2L[7:0]							0000,0000	
PWM5CR	PWM5 Control register	FF34H	-	-	-	-	PWM5_PS	EPWM5I	EC5T2SI	EC5T1SI	xxxx,0000
PWM6T1H	Timer 1 of PWM6 High	FF40H	-	PWM6T1H[14:8]							x000,0000
PWM6T1L	Timer 1 of PWM6 Low	FF41H	PWM6T1L[7:0]							0000,0000	
PWM6T2H	Timer 2 of PWM6 High	FF42H	-	PWM6T2H[14:8]							x000,0000
PWM6T2L	Timer 2 of PWM6 Low	FF43H	PWM6T2L[7:0]							0000,0000	
PWM6CR	PWM6 Control register	FF44H	-	-	-	-	PWM6_PS	EPWM6I	EC6T2SI	EC6T1SI	xxxx,0000
PWM7T1H	Timer 1 of PWM7 High	FF50H	-	PWM7T1H[14:8]							x000,0000
PWM7T1L	Timer 1 of PWM7 Low	FF51H	PWM7T1L[7:0]							0000,0000	
PWM7T2H	Timer 2 of PWM7 High	FF52H	-	PWM7T2H[14:8]							x000,0000
PWM7T2L	Timer 2 of PWM7 Low	FF53H	PWM7T2L[7:0]							0000,0000	
PWM7CR	PWM7 Control register	FF54H	-	-	-	-	PWM7_PS	EPWM7I	EC7T2SI	EC7T1SI	xxxx,0000

Some common SFRs of traditional 8051 are shown as below.

#### Accumulator

ACC is the Accumulator register. The mnemonics for accumulator-specific instructions, however, refer to the accumulator simply as A.

#### B-Register

The B register is used during multiply and divide operations. For other instructions it can be treated as another scratch pad register.

#### Stack Pointer

The Stack Pointer register is 8 bits wide. It is incremented before data is stored during PUSH and CALL executions. While the stack may reside anywhere in on-chip RAM, the Stack Pointer is initialized to 07H after a reset. Therefore, the first value pushed on the stack is placed at location 0x08, which is also the first register (R0) of register bank 1. Thus, if more than one register bank is to be used, the SP should be initialized to a location in the data memory not being used for data storage. The stack depth can extend up to 256 bytes.



---

## Program Status Word(PSW)

The program status word(PSW) contains several status bits that reflect the current state of the CPU. The PSW, shown below, resides in the SFR space. It contains the Carry bit, the Auxiliary Carry(for BCD operation), the two register bank select bits, the Overflow flag, a Parity bit and two user-definable status flags.

The Carry bit, other than serving the function of a Carry bit in arithmetic operations, also serves as the “Accumulator” for a number of Boolean operations.

The bits RS0 and RS1 are used to select one of the four register banks shown in the previous page. A number of instructions refer to these RAM locations as R0 through R7.

The Parity bit reflects the number of 1s in the Accumulator. P=1 if the Accumulator contains an odd number of 1s and otherwise P=0.

### PSW register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PSW	D0H	name	CY	AC	F0	RS1	RS0	OV	F1	P

CY : Carry flag.

This bit is set when the last arithmetic operation resulted in a carry (addition) or a borrow (subtraction). It is cleared to logic 0 by all other arithmetic operations.

AC : Auxilliary Carry Flag.(For BCD operations)

This bit is set when the last arithmetic operation resulted in a carry into (addition) or a borrow from (subtraction) the high order nibble. It is cleared to logic 0 by all other arithmetic operations

F0 : Flag 0.(Available to the user for general purposes)

RS1: Register bank select control bit 1.

RS0: Register bank select control bit 0.

[RS1 RS0] select which register bank is used during register accesses

RS1	RS0	Working Register Bank(R0~R7) and Address
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.

This bit is set to 1 under the following circumstances:

- An ADD, ADDC, or SUBB instruction causes a sign-change overflow.
- A MUL instruction results in an overflow (result is greater than 255).
- A DIV instruction causes a divide-by-zero condition.

The OV bit is cleared to 0 by the ADD, ADDC, SUBB, MUL, and DIV instructions in all other cases.

F1 : Flag 1. User-defined flag.

P : Parity flag.

This bit is set to logic 1 if the sum of the eight bits in the accumulator is odd and cleared if the sum is even.

### 3.3.3 Dual Data Pointer Register (DPTR)

The Data Pointer (DPTR) consists of a high byte (DPH) and a low byte (DPL). Its intended function is to hold a 16-bit address. It may be manipulated as a 16-bit register or as two independent 8-bit registers.

For fast data movement, STC15W4K32S4 series MCU supports two data pointers. They share the same SFR address and are switched by the register bit – DPS/AUXR.0.

#### AUXR1 register

Mnemonic	Address	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary Register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0100,0000

DPS : DPTR registers select bit.

0 : Default. DPTR0 is selected as Data pointer.

1 : The secondary DPTR is switched to use.

**The following program is an assembly program that demonstrates how the dual data pointer be used.**

```

AUXR1      DATA    0A2H           ;Define special function register AUXR1
MOV        AUXR1, #0              ;DPS=0, select DPTR0

MOV        DPTR,  #1FFH           ;Set DPTR0 for 1FFH
MOV        A,      #55H
MOVX      @DPTR, A                ;load the value 55H in the 1FFH unit

MOV        DPTR,  #2FFH           ;Set DPTR0 for 2FFH
MOV        A,      #0AAH
MOVX      @DPTR, A                ;load the value 0AAH in the 2FFH unit

INC        AUXR1                  ;DPS=1, DPTR1 is selected
MOV        DPTR,  #1FFH           ;Set DPTR1 for 1FFH
MOVX      A,      @DPTR           ;Get the content of 1FFH unit
;which is pointed by DPTR1,
;the content of Accumulator has changed for 55H

INC        AUXR1                  ;DPS=0, DPTR0 is selected
MOVX      A,      @DPTR           ;Get the content of 2FFH unit
;which is pointed by DPTR0,
;the content of Accumulator has changed for 0AAH

INC        AUXR1                  ;DPS=1, DPTR1 is selected
MOVX      A,      @DPTR           ;Get the content of 1FFH unit
;which is pointed by DPTR1,
;the content of Accumulator has changed for 55H

INC        AUXR1                  ;DPS=0, DPTR0 is selected
MOVX      A,      @DPTR           ;Get the content of 2FFH unit
;which is pointed by DPTR0,
;the content of Accumulator has changed for 0AAH

```

---

# Chapter 4 Configurable I/O Ports of STC15 series MCU

## 4.1 I/O Ports Configurations

STC15 series MCU owns 62 I/O ports (such as 64-pin MCU), at most. The 62 I/O ports are P0.0~P0.7, P1.0~P1.7, P2.0~P2.7, P3.0~P3.7, P4.0~P4.7, P5.0~P5.5, P6.0~P6.7 and P7.0~P7.5. All I/O ports of STC15 series MCU may be independently configured to one of four modes by setting the corresponding bit in two mode registers PxMn (x= 0 ~ 7, n = 0, 1). The four modes are quasi-bidirectional (traditional 8051 port output), push-pull output, input-only and open-drain output. All port pins default to quasi-bidirectional after reset. Each one has a Schmitt-triggered input for improved input noise rejection. Any port can drive 20mA current, but it had better drive lower than 120mA current that the whole chip of 40-pin or more than 40-pin MCU, while 90mA that the whole chip of 16-pin or more than 16-pin MCU or 32-pin or less than 32-pin MCU .

### Configure I/O ports mode

P0 Configure <P0.7, P0.6, P0.5, P0.4, P0.3, P0.2, P0.1, P0.0 port> (P0 address 80H)

P0M1[7 : 0] P0M1 address is 93H	P0M0 [7 : 0] P0M0 address is 94H	I/O ports Mode
0	0	quasi_bidirectional (traditional 8051 I/O port output , Sink Current up to 20mA , pull-up Current is 270μA , Because of manufactured error, the actual pull-up current is 270uA ~ 150uA
0	1	push-pull output(strong pull-up output current can be up to 20mA, resistors need to be added to restrict current
1	0	input-only (high-impedance )
1	1	Open Drain internal pull-up resistors should be disabled and external pull-up resistors need to join.

Example: MOV P0M1, #10100000B

MOV P0M0, #11000000B

;P0.7 in Open Drain mode, P0.6 in strong push-pull output, P0.5 in high-impedance input, P0.4/P0.3/P0.2/P0.1/P0.0 in quasi\_bidirectional/weak pull-up

P1 Configure <P1.7, P1.6, P1.5, P1.4, P1.3, P1.2, P1.1, P1.0 port> (P1 address 90H)

P1M1[7 : 0] P1M1 address is 91H	P1M0 [7 : 0] P1M0 address is 92H	I/O ports Mode
0	0	quasi_bidirectional(traditional 8051 I/O port output , Sink Current up to 20mA , pull-up Current is 270μA , Because of manufactured error, the actual pull-up current is 270uA ~ 150uA
0	1	push-pull output(strong pull-up output current can be up to 20mA, resistors need to be added to restrict current
1	0	input-only (high-impedance )
1	1	Open Drain internal pull-up resistors should be disabled and external pull-up resistors need to join.

Example: MOV P1M1, #10100000B

MOV P1M0, #11000000B

;P1.7 in Open Drain mode, P1.6 in strong push-pull output, P1.5 in high-impedance input, P1.4/P1.3/P1.2/P1.1/P1.0 in quasi\_bidirectional/weak pull-up

P2 Configure <P2.7, P2.6, P2.5, P2.4, P2.3, P2.2, P2.1, P2.0 port> (P2 address A0H)

P2M1[7 : 0] P2M1 address is 95H	P2M0 [7 : 0] P2M0 address is 96H	I/O ports Mode
0	0	quasi_bidirectional(traditional 8051 I/O port output , Sink Current up to 20mA , pull-up Current is 270μA , Because of manufactured error, the actual pull-up current is 270uA ~ 150uA
0	1	push-pull output(strong pull-up output current can be up to 20mA, resistors need to be added to restrict current
1	0	input-only (high-impedance )
1	1	Open Drain internal pull-up resistors should be disabled and external pull-up resistors need to join.

Example: MOV P2M1, #10100000B

MOV P2M0, #11000000B

;P2.7 in Open Drain mode, P2.6 in strong push-pull output, P2.5 in high-impedance input, P2.4/P2.3/P2.2/P2.1/P2.0 in quasi\_bidirectional/weak pull-up

P3 Configure <P3.7, P3.6, P3.5, P3.4, P3.3, P3.2, P3.1, P3.0 port> (P3 address B0H)

P3M1[7 : 0] P3M1 address is B1H	P3M0 [7 : 0] P3M0 address is B2H	I/O ports Mode
0	0	quasi_bidirectional(traditional 8051 I/O port output , Sink Current up to 20mA , pull-up Current is 270μA , Because of manufactured error, the actual pull-up current is 270uA ~ 150uA
0	1	push-pull output(strong pull-up output current can be up to 20mA, resistors need to be added to restrict current
1	0	input-only (high-impedance )
1	1	Open Drain internal pull-up resistors should be disabled and external pull-up resistors need to join.

Example: MOV P3M1, #10100000B

MOV P3M0, #11000000B

;P3.7 in Open Drain mode, P3.6 in strong push-pull output, P3.5 in high-impedance input, P3.4/P3.3/P3.2/P3.1/P3.0 in quasi\_bidirectional/weak pull-up

P4 Configure <P4.7, P4.6, P4.5, P4.4, P4.3, P4.2, P4.1, P4.0 port> (P4 address C0H)

P4M1[7 : 0] P4M1 address is B3H	P4M0 [7 : 0] P4M0 address is B4H	I/O ports Mode
0	0	quasi_bidirectional(traditional 8051 I/O port output), Sink Current up to 20mA , pull-up Current is 270μA , Because of manufactured error, the actual pull-up current is 270uA ~ 150uA
0	1	push-pull output(strong pull-up output, current can be up to 20mA, resistors need to be added to restrict current
1	0	input-only (high-impedance)
1	1	Open Drain, internal pull-up resistors should be disabled and external pull-up resistors need to join.

Example: MOV P4M1, #10100000B

MOV P4M0, #11000000B

;P4.7 in Open Drain mode, P4.6 in strong push-pull output, P4.5 in high-impedance input, P4.4/P4.3/P4.2/P4.1/P4.0 in quasi\_bidirectional/weak pull-up

P5 Configure <x, x, P5.5, P5.4, P5.3, P5.2, P5.1, P5.0 port> (P5 address C8H)

P5M1[5 : 0] P5M1 address is C9H	P5M0 [5 : 0] P5M0 address is CAH	I/O ports Mode
0	0	quasi_bidirectional(traditional 8051 I/O port output), Sink Current up to 20mA , pull-up Current is 270μA , Because of manufactured error, the actual pull-up current is 270uA ~ 150uA
0	1	push-pull output(strong pull-up output, current can be up to 20mA, resistors need to be added to restrict current
1	0	input-only (high-impedance)
1	1	Open Drain, internal pull-up resistors should be disabled and external pull-up resistors need to join.

Example: MOV P5M1, #00101000B

MOV P5M0, #00110000B

;P5.5 in Open Drain mode, P5.4 in strong push-pull output, P5.3 in high-impedance input, P5.2/P5.1/P5.0 in quasi\_bidirectional/weak pull-up

P6 Configure <P6.7, P6.6, P6.5, P6.4, P6.3, P6.2, P6.1, P6.0 port> (P6 address E8H)

P6M1[7 : 0] P6M1 address is CBH	P6M0 [7 : 0] P6M0 address is CCH	I/O ports Mode
0	0	quasi_bidirectional (traditional 8051 I/O port output , Sink Current up to 20mA , pull-up Current is 270μA , Because of manufactured error, the actual pull-up current is 270uA ~ 150uA
0	1	push-pull output(strong pull-up output current can be up to 20mA, resistors need to be added to restrict current
1	0	input-only (high-impedance )
1	1	Open Drain internal pull-up resistors should be disabled and external pull-up resistors need to join.

Example: MOV P6M1, #10100000B

MOV P6M0, #11000000B

;P6.7 in Open Drain mode, P6.6 in strong push-pull output, P6.5 in high-impedance input, P6.4/P6.3/P6.2/P6.1/P6.0 in quasi\_bidirectional/weak pull-up

P7 Configure <P7.7, P7.6, P7.5, P7.4, P7.3, P7.2, P7.1, P7.0 port> (P7 address F8H)

P7M1[7 : 0] P7M1 address is E1H	P7M0 [7 : 0] P7M0 address is E2H	I/O ports Mode
0	0	quasi_bidirectional (traditional 8051 I/O port output , Sink Current up to 20mA , pull-up Current is 270μA , Because of manufactured error, the actual pull-up current is 270uA ~ 150uA
0	1	push-pull output(strong pull-up output current can be up to 20mA, resistors need to be added to restrict current
1	0	input-only (high-impedance )
1	1	Open Drain internal pull-up resistors should be disabled and external pull-up resistors need to join.

Example: MOV P7M1, #10100000B

MOV P7M0, #11000000B

;P7.7 in Open Drain mode, P7.6 in strong push-pull output, P7.5 in high-impedance input, P7.4/P7.3/P7.2/P7.1/P7.0 in quasi\_bidirectional/weak pull-up

---

## 4.2 Special Explanation of P1.7/XTAL1 and P1.6/XTAL2 pin

All I/O ports default to quasi-bidirectional / weak-pull after power-on reset. But P1.7/XTAL1 and P1.6/XTAL2 are not necessarily in quasi-two-dimensional / weak-pull mode after power-on reset due to P1.7 and P1.6 also can be used as external crystal or clock pins XTAL1 and XTAL2. When P1.7/XTAL1 and P1.6/XTAL2 are used as XTAL1 and XTAL2, they are in high impedance input mode after power-on reset

The mode of P1.7/XTAL1 and P1.6/XTAL2 is set according to the following steps after each power-on reset :

First, P1.7/XTAL1 and P1.6/XTAL2 will be set to high impedance input mode in a short time;

Then, MCU will automatically determine the setting of P1.7/XTAL1 and P1.6/XTAL2 what the user do in STC-ISP Writer / Programmer last time;

If P1.7/XTAL1 and P1.6/XTAL2 were set to the common I/O ports in STC-ISP Writer / Programmer last time, they would be in quasi-bidirectional / weak pull-up mode after power-on reset;

If P1.7/XTAL1 and P1.6/XTAL2 were set to XTAL1 and XTAL2 in STC-ISP Writer / Programmer last time, they would be in high impedance input mode after power-on reset.

## 4.3 Special Explanation of RST pin

The reset pin is on RST/P5.4 for STC15W4K32S4 series MCU. P5.4/RST pin factory defaults to the I/O port, which can be set as RST reset pin(active high) through the STC-ISP Writer / Programmer. If it is as I/O port, it will be in quasi-bidirectional / weak pull-up mode after power-on reset. MCU will automatically determine the setting of P5.4/RST what the user do in STC-ISP Writer / Programmer last time after each power-on reset. If P5.4/RST were set to the common I/O port in STC-ISP Writer / Programmer last time, it would be in quasi-bidirectional / weak pull-up mode after power-on reset. If P5.4/RST were set to Reset pin in STC-ISP Writer / Programmer last time, they would be still as reset pin after power-on reset.

## 4.4 Special Explanation of RSTOUT\_LOW pin

The output low after reset pin is on RSTOUT\_LOW/P2.0 for STC15W4K32S4 series MCU. P2.0/RSTOUT\_LOW pin can output low or high after power-on reset. When the operation voltage Vcc is higher than power-on reset threshold voltage (POR), users can set whether the P2.0/RSTOUT\_LOW pin output low or high in STC-ISP Writer/Programmer.

When the operation voltage Vcc is lower than power-on reset threshold voltage (POR), P2.0/RSTOUT\_LOW pin output low. When the operation voltage Vcc is higher than power-on reset threshold voltage (POR), MCU will automatically determine the setting in STC-ISP Writer / Programmer last time after each power-on reset. If P2.0/RSTOUT\_LOW pin was set to output low after each power-on reset in STC-ISP Writer / Programmer last time, P2.0/RSTOUT\_LOW pin will output low. If P2.0/RSTOUT\_LOW pin was set to output high after each power-on reset in STC-ISP Writer / Programmer last time, P2.0/RSTOUT\_LOW pin will output high.

---

## 4.5 SFRs related to I/O ports and Its Address Statement

Some SFRs related with I/O ports are listed below.

**P0 register** (bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P0	80H	name	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0

**P0M1 register** (non bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P0M1	93H	name	P0M1.7	P0M1.6	P0M1.5	P0M1.4	P0M1.3	P0M1.2	P0M1.1	P0M1.0

**P0M0 register** (non bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P0M0	94H	name	P0M0.7	P0M0.6	P0M0.5	P0M0.4	P0M0.3	P0M0.2	P0M0.1	P0M0.0

**P1 register** (bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P1	90H	name	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0

**P1M1 register** (non bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P1M1	91H	name	P1M1.7	P1M1.6	P1M1.5	P1M1.4	P1M1.3	P1M1.2	P1M1.1	P1M1.0

**P1M0 register** (non bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P1M0	92H	name	P1M0.7	P1M0.6	P1M0.5	P1M0.4	P1M0.3	P1M0.2	P1M0.1	P1M0.0

**P2 register** (bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P2	A0H	name	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0

**P2M1 register** (non bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P2M1	95H	name	P2M1.7	P2M1.6	P2M1.5	P2M1.4	P2M1.3	P2M1.2	P2M1.1	P2M1.0

**P2M0 register** (non bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P2M0	96H	name	P2M0.7	P2M0.6	P2M0.5	P2M0.4	P2M0.3	P2M0.2	P2M0.1	P2M0.0

---

**P3 register** (bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P3	B0H	name	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0

**P3M1 register** (non bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P3M1	B1H	name	P3M1.7	P3M1.6	P3M1.5	P3M1.4	P3M1.3	P3M1.2	P3M1.1	P3M1.0

**P3M0 register** (non bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P3M0	B2H	name	P3M0.7	P3M0.6	P3M0.5	P3M0.4	P3M0.3	P3M0.2	P3M0.1	P3M0.0

**P4 register** (bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P4	C0H	name	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0

**P4M1 register** (non bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P4M1	B3H	name	P4M1.7	P4M1.6	P4M1.5	P4M1.4	P4M1.3	P4M1.2	P4M1.1	P4M1.0

**P4M0 register** (non bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P4M0	B4H	name	P4M0.7	P4M0.6	P4M0.5	P4M0.4	P4M0.3	P4M0.2	P4M0.1	P4M0.0

**P5 register** (bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P5	C8H	name	-	-	P5.5	P5.4	P5.3	P5.2	P5.1	P5.0

**P5M1 register** (non bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P5M1	C9H	name	-	-	P5M1.5	P5M1.4	P5M1.3	P5M1.2	P5M1.1	P5M1.0

**P5M0 register** (non bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P5M0	CAH	name	-	-	P5M0.5	P5M0.4	P5M0.3	P5M0.2	P5M0.1	P5M0.0



---

**P6 register** (bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P6	E8H	name	P6.7	P6.6	P6.5	P6.4	P6.3	P6.2	P6.1	P6.0

**P6M1 register** (non bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P6M1	CBH	name	P6M1.7	P6M1.6	P6M1.5	P6M1.4	P6M1.3	P6M1.2	P6M1.1	P6M1.0

**P6M0 register** (non bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P6M0	CCH	name	P6M0.7	P6M0.6	P6M0.5	P6M0.4	P6M0.3	P6M0.2	P6M0.1	P6M0.0

**P7 register** (bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P7	F8H	name	P7.7	P7.6	P7.5	P7.4	P7.3	P7.2	P7.1	P7.0

**P7M1 register** (non bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P7M1	E1H	name	P7M1.7	P7M1.6	P7M1.5	P7M1.4	P7M1.3	P7M1.2	P7M1.1	P7M1.0

**P7M0 register** (non bit addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P7M0	E2H	name	P7M0.7	P7M0.6	P7M0.5	P7M0.4	P7M0.3	P7M0.2	P7M0.1	P7M0.0

Assembly

```
P7 EQU 0F8H ; or P7 DATA 0F8H
P7M1 EQU 0E1H ; or P7M1 DATA 0E1H
P7M0 EQU 0E2H
;P7 address statement is shown above
P6 EQU 0E8H ; or P6 DATA 0E8H
P6M1 EQU 0CBH ; or P6M1 DATA 0CBH
P6M0 EQU 0CCH
;P6 address statement is shown above
P5 EQU 0C8H ; or P5 DATA 0C8H
P5M1 EQU 0C9H ; or P5M1 DATA 0C9H
P5M0 EQU 0CAH
;P5 address statement is shown above
```

---

```

P4      EQU    0C0H      ; or P4      DATA  0C0H
P4M1   EQU    0B3H      ; or P4M1   DATA  0B3H
P4M0   EQU    0B4H
;P4 address statement is shown above
P3M1   EQU    0B1H      ; or P3M1   DATA  0B1H
P3M0   EQU    0B2H
;P3 address statement is shown above
P2M1   EQU    095H
P2M0   EQU    096H
;P2 address statement is shown above
P1M1   EQU    091H
P1M0   EQU    092H
;P1 address statement is shown above
P0M1   EQU    093H
P0M0   EQU    094H
;P0 address statement is shown above

```

#### C Language:

```

sfr     P7      = 0xf8;
sfr     P7M1   = 0xe1;
sfr     P7M0   = 0xe2;
/*P7 address statement is shown above*/
sfr     P6      = 0xe8;
sfr     P6M1   = 0xcb;
sfr     P6M0   = 0xcc;
/*P6 address statement is shown above*/
sfr     P5      = 0xc8;
sfr     P5M1   = 0xc9;
sfr     P5M0   = 0xca;
/*P5 address statement is shown above*/
sfr     P4      = 0xc0;
sfr     P4M1   = 0xb3;
sfr     P4M0   = 0xb4;
/*P4 address statement is shown above*/
sfr     P3M1   = 0xb1;
sfr     P3M0   = 0xb2;
/*P3 address statement is shown above*/
sfr     P2M1   = 0x95;
sfr     P2M0   = 0x96;
/*P2 address statement is shown above*/
sfr     P1M1   = 0x91;
sfr     P1M0   = 0x92;
/*P1 address statement is shown above*/
sfr     P0M1   = 0x93;
sfr     P0M0   = 0x94;
/*P0 address statement is shown above*/

```

---

## 4.6 Demo Program of STC15 series P0/P1/P2/P3/P4/P5

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program test P0/P1/P2/P3/P4/P5 -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

sfr P5 = 0xC8; //6 bit Port5 P5.7 P5.6 P5.5 P5.4 P5.3 P5.2 P5.1 P5.0 xxxx1,1111
sfr P5M0 = 0xC9; // 0000,0000
sfr P5M1 = 0xCA; // 0000,0000
// 7 6 5 4 3 2 1 0 Reset Value

sfr P4 = 0xC0; //8 bitPort4 P4.7 P4.6 P4.5 P4.4 P4.3 P4.2 P4.1 P4.0 1111,1111
sfr P4M0 = 0xB4; // 0000,0000
sfr P4M1 = 0xB3; // 0000,0000

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 P30 = P3^0;
sbit P31 = P3^1;
sbit P32 = P3^2;
sbit P33 = P3^3;
sbit P34 = P3^4;
sbit P35 = P3^5;
sbit P36 = P3^6;
sbit P37 = P3^7;
```

---

```
sbit 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 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 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;
```

```
void delay(void);
```

```
void main(void)
```

```
{
    P10 = 0;
    delay();
    P11 = 0;
    delay();
    P12 = 0;
    delay();
    P13 = 0;
    delay();
    P14 = 0;
    delay();
}
```

---

```
P15 = 0;
delay();
P16 = 0;
delay();
P17 = 0;
delay();

P1 = 0xff;

P30 = 0;
delay();
P31 = 0;
delay();
P32 = 0;
delay();
P33 = 0;
delay();
P34 = 0;
delay();
P35 = 0;
delay();
P36 = 0;
delay();
P37 = 0;
delay();

P3 = 0xff;

P20 = 0;
delay();
P21 = 0;
delay();
P22 = 0;
delay();
P23 = 0;
delay();
P24 = 0;
delay();
P25 = 0;
delay();
P26 = 0;
delay();
P27 = 0;
delay();

P2 = 0xff;

P07 = 0;
delay();
```

---

---

```
P06 = 0;
delay();
P05 = 0;
delay();
P04 = 0;
delay();
P03 = 0;
delay();
P02 = 0;
delay();
P01 = 0;
delay();
P00 = 0;
delay();

P0 = 0xff;

P40 = 0;
delay();
P41 = 0;
delay();
P42 = 0;
delay();
P43 = 0;
delay();
P44 = 0;
delay();
P45 = 0;
delay();
P46 = 0;
delay();
P47 = 0;
delay();

P4 = 0xff;

P50 = 0;
delay();
P51 = 0;
delay();
P52 = 0;
delay();
P53 = 0;
delay();
P54 = 0;
delay();
P55 = 0;
delay();

P5 = 0xff;
```

---

---

```

while(1)
{
    P1    =    0x00;
    delay();
    P1    =    0xff;

    P3    =    0x00;
    delay();
    P3    =    0xff;

    P2    =    0x00;
    delay();
    P2    =    0xff;

    P0    =    0x00;
    delay();
    P0    =    0xff;

    P4    =    0x00;
    delay();
    P4    =    0xff;

    P5    =    0x00;
    delay();
    P5    =    0xff;
}
}

```

```

void delay(void)
{
    unsigned int i = 0;
    for(i=60000;i>0;i--)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

```

---

```

        _nop_();
        _nop_();
        _nop_();
        _nop_();

        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        }
}

```



---

## 4.7 I/O ports Modes

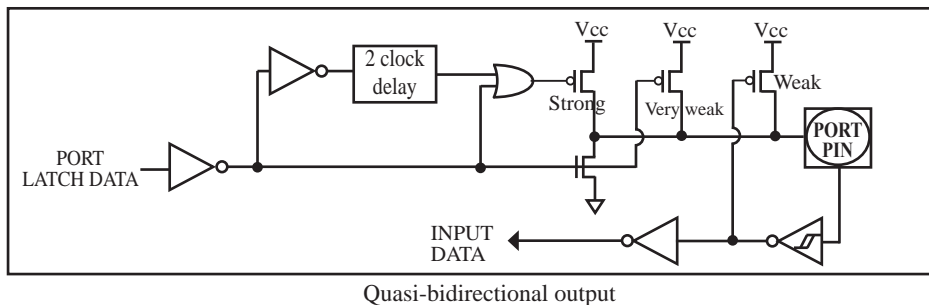
### 4.7.1 Quasi-Bidirectional I/O

Port pins in quasi-bidirectional output mode function similar to the traditional 8051 port pins. A quasi-bidirectional port can be used as an input and output without the need to reconfigure the port. This is possible because when the port outputs a logic high, it is weakly driven, allowing an external device to pull the pin low. When the pin outputs low, it is driven strongly and able to sink a large current. There are three pull-up transistors in the quasi-bidirectional output that serve different purposes.

One of these pull-ups, called the “very weak” pull-up, is turned on whenever the port register for the pin contains a logic “1”. This very weak pull-up sources a very small current that will pull the pin high if it is left floating.

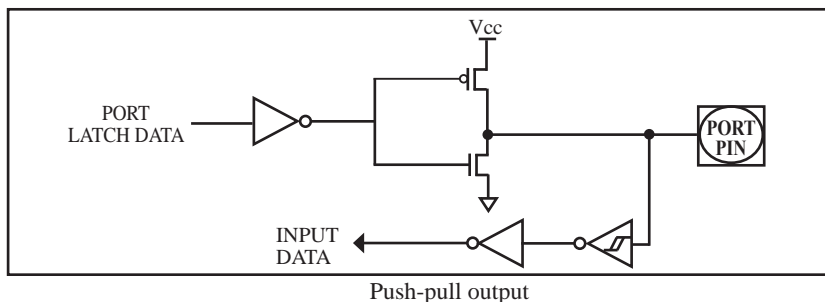
A second pull-up, called the “weak” pull-up, is turned on when the port register for the pin contains a logic “1” and the pin itself is also at a logic “1” level. This pull-up provides the primary source current for a quasi-bidirectional pin that is outputting a 1. If this pin is pulled low by the external device, this weak pull-up turns off, and only the very weak pull-up remains on. In order to pull the pin low under these conditions, the external device has to sink enough current to over-power the weak pull-up and pull the port pin below its input threshold voltage.

The third pull-up is referred to as the “strong” pull-up. This pull-up is used to speed up low-to-high transitions on a quasi-bidirectional port pin when the port register changes from a logic “0” to a logic “1”. When this occurs, the strong pull-up turns on for two CPU clocks, quickly pulling the port pin high.



### 4.7.2 Push-Pull Output

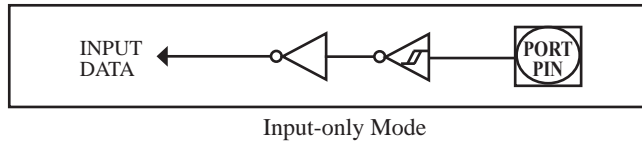
The push-pull output configuration has the same pull-down structure as both the open-drain and the quasi-bidirectional output modes, but provides a continuous strong pull-up when the port register contains a logic “1”. The push-pull mode may be used when more source current is needed from a port output. In addition, input path of the port pin in this configuration is also the same as quasi-bidirectional mode.



---

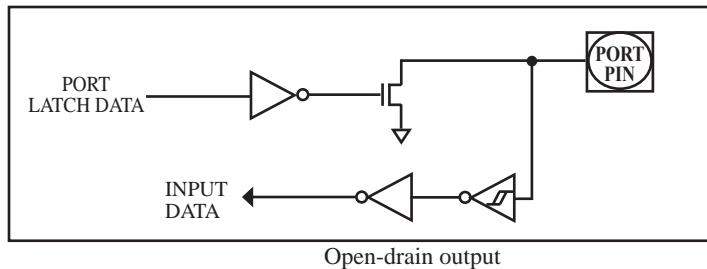
### 4.7.3 Input-Only (High-Impedance) Mode

The input-only configuration is a Schmitt-triggered input without any pull-up resistors on the pin.



### 4.7.4 Open-Drain Output

The open-drain output configuration turns off all pull-ups and only drives the pull-down transistor of the port pin when the port register contains a logic “0”. To use this configuration in application, a port pin must have an external pull-up, typically tied to VCC. The input path of the port pin in this configuration is the same as quasi-bidirection mode.



## 4.8 I/O Port Application Notes

Traditional 8051 access I/O (signal transition or read status) timing is 12 clocks, STC15 series MCU is 4 clocks. When you need to read an external signal, if internal output a rising edge signal, for the traditional 8051, this process is 12 clocks, you can read at once, but for STC15W4K32S4 series MCU, this process is 4 clocks, when internal instructions is complete but external signal is not ready, so you must delay 1~2 nop operation.

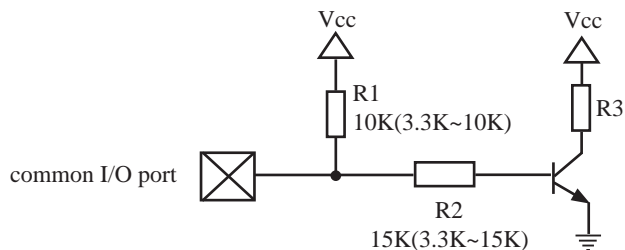
When MCU is connected to a SPI or I2C or other open-drain peripherals circuit, you need add a 10K pull-up resistor.

Some IO port connected to a PNP transistor, but no pul-up resistor. The correct access method is IO port pull-up resistor and transistor base resistor should be consistent, or IO port is set to a strongly push-pull output mode.

Using IO port drive LED directly or matrix key scan, needs add a 470ohm to 1Kohm resistor to limit current.

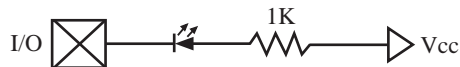
---

## 4.9 Typical Transistor Control Circuit



If I/O is configured as “weak” pull-up, you should add an external pull-up resistor R1(3.3K~10K ohm). If no pull-up resistor R1, proposal to add a 15K ohm series resistor R2 at least or config I/O as “push-pull” mode.

## 4.10 Typical Diode Control Circuit



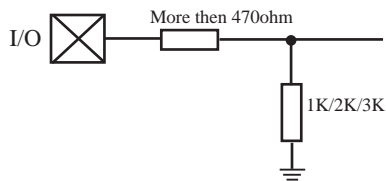
For weak pull-up / quasi-bidirectional I/O, use sink current drive LED, current limiting resistor as greater than 1K ohm, minimum not less than 470 ohm.



For push-pull / strong pull-up I/O, use drive current drive LED.

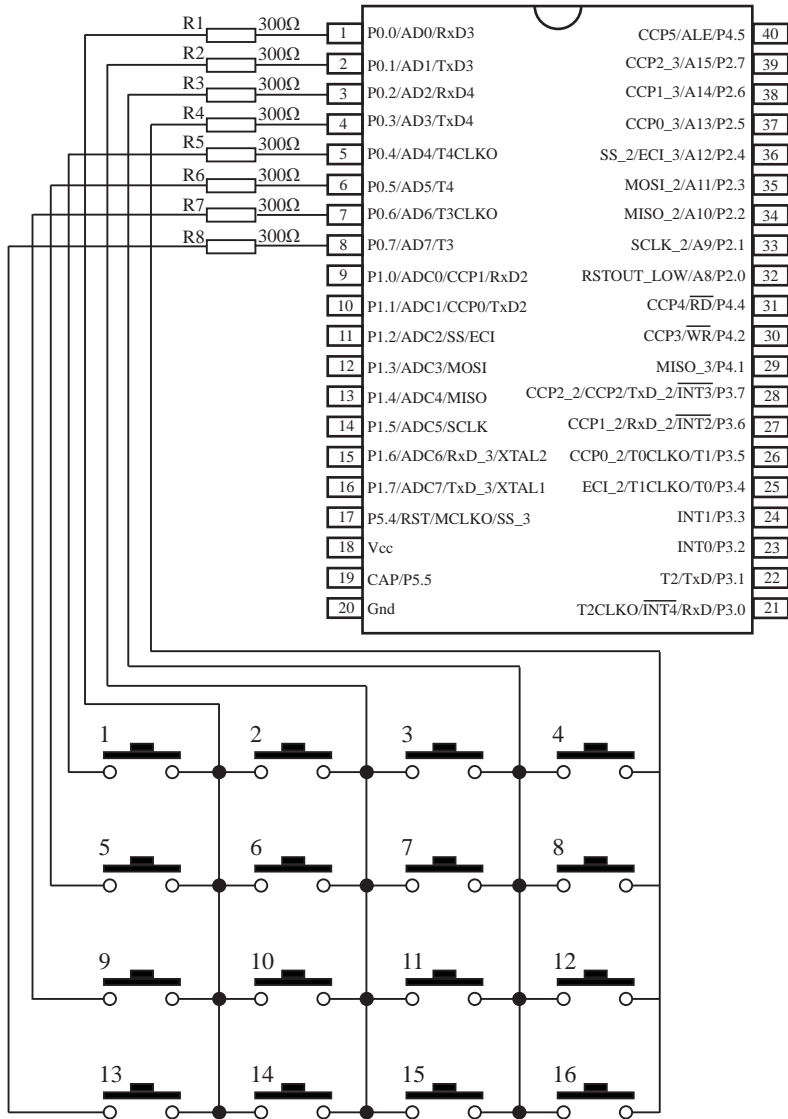
## 4.11 How to Make I/O Port Low after MCU Reset

Traditional 8051 MCU power-on reset, the general IO port are weak pull-high output, while many practical applications require IO port remain low level after power-on reset, otherwise the system malfunction would be generated. For STC15 series MCU, IO port can add a pull-down resistor (1K/2K/3K), so that when power-on reset, although a weak internal pull-up to make MCU output high, but because of the limited capacity of the internal pull-up, it can not pull-high the pad, so this IO port is low level after power-on reset. If the I/O port need to drive high, you can set the IO model as the push-pull output mode, while the push-pull mode the drive current can be up to 20mA, so it can drive this I/O high.

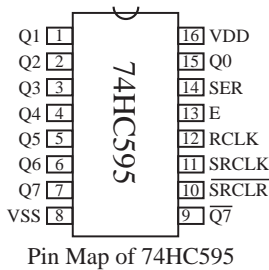


Note: Users can set whether the P2.0/RSTOUT\_LOW pin output low or high after power-on reset in STC-ISP Writer/Programmer. But other pins of STC15 series all output high after power-on reset.

## 4.12 Keyboard Scanning Circuit using I/O ports



## 4.13 Pin Function and Logic Turth Table of 74HC595



Pin Name	Pin Number	Pin Function
Q0 ~ Q7	15, 1~7	Noninverted, 3-state, latch outputs
$\overline{Q7}$	9	Serial data output
SRCLR	10	reset(active-low)
SRCLK	11	Shift Register Clock Input
RCLK	12	Storage Latch Clock Input
E	13	Active-low Output Enable
SER	14	Serial data input
VDD	16	Power
VSS	8	Gnd

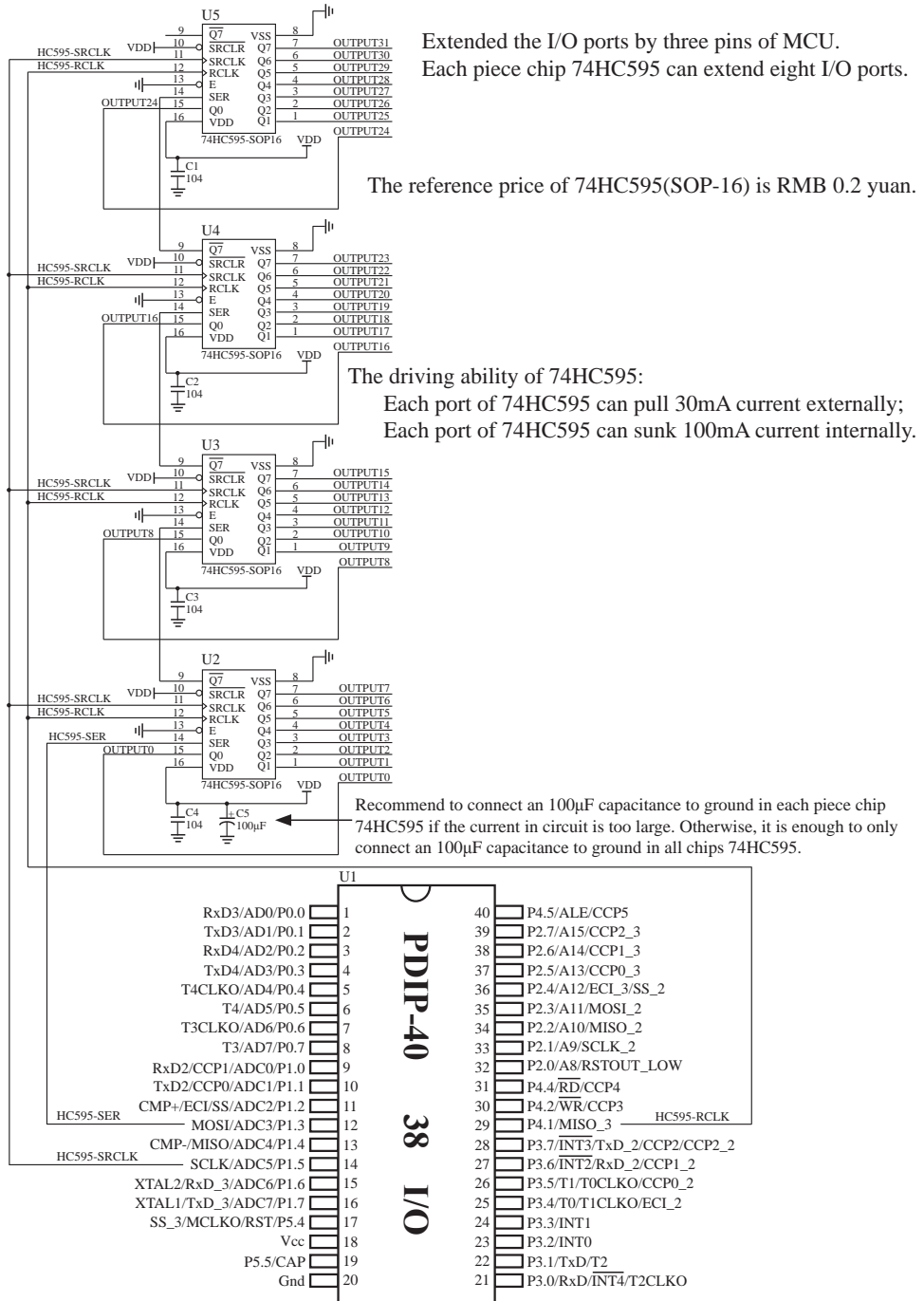
The 74HC595 consists of an 8-bit shift register and an 8-bit D-type latch with three-state parallel outputs. The shift register accepts serial data and provides a serial output. The shift register also provides parallel data to the 8-bit latch. The shift register and latch have independent clock inputs. This device also has an asynchronous reset for the shift register.

The HC595 directly interfaces with the SPI serial data port on CMOS MPUs and MCUs.

Serial data input pin SER, the data on this pin is shifted into the 8-bit serial shift register. Shift register clock input pin SRCLK, a low-to-high transition on this input causes the data at the Serial Input pin to be shifted into the 8-bit shift register. Reset pin SRCLR, active-low, asynchronous, Shift Register Reset Input. A low on this pin resets the shift register portion of this device only. The 8-bit latch is not affected. Storage Latch Clock Input pin RCLK, a low-to-high transition on this input latches the shift register data. Active-low Output Enable pin E, a low on this input allows the data from the latches to be presented at the outputs. A high on this input forces the outputs (Q0~Q7) into the high-impedance state. The serial output is not affected by this control unit. Noninverted, Serial Data Output pin  $\overline{Q7}$ , this is the output of the eighth stage of the 8-bit shift register. This output does not have three-state capability.

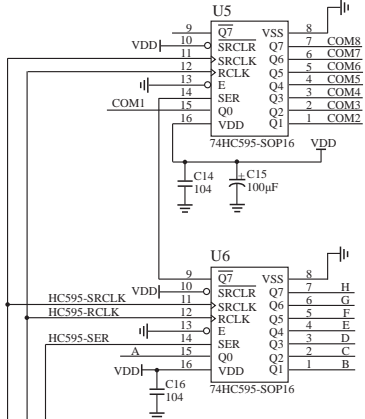
Inputs					Outputs
SER	SRCLK	SRCLR	RCLK	E	
X	X	X	X	H	Q0~Q7 force outputs into high impedance state
X	X	X	X	L	Enable parallel outputs Q0~Q7
X	X	L	X	X	Reset shift register
L	↑	H	X	X	Shift data "L" into shift register
H	↑	H	X	X	Shift data "H" into shift register
X	↓	H	X	X	Shift register remains unchanged
X	X	X	↑	X	Transfer shift register contents to latch register
X	X	X	↓	X	Latch register remains unchanged

## 4.14 Circuit Expanding I/O ports using 74HC595

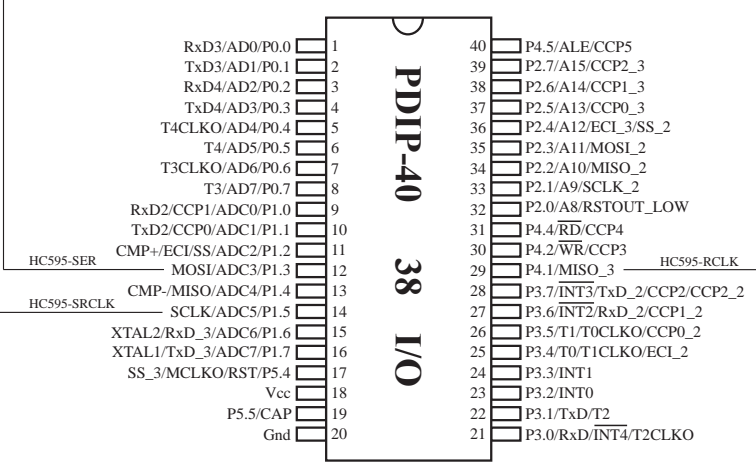
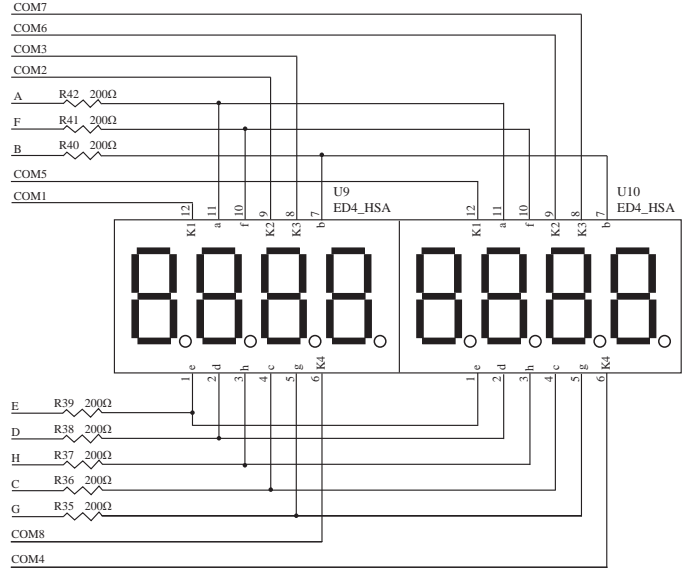


# 4.15 Circuit Driving 8-segment Digitron using 74HC595

Use two piece chips 74HC595 to drive 8-segment digitron  
 It would be better to select common cathode digitron.  
 (both Common cathode and Common anode)



The reference price of 74HC595(SOP-16) is RMB 0.2 yuan.



---

## 4.16 Demo Program of Driving 8-Segment Digitron

### —— Using common I/O ports to Control 74HC595

#### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program that driving 8-segment digitron -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

/***** the description of functions *****/

drive 8-bit digitron using common I/O ports to control 74HC595

users can choose the clock frequency by revised macros.

users can choose whether the digitron is common cathode or anode in display function.
recommend to choose common cathode

Display effect: cycle display 0,1,2...,A,B..F, black-out in 8 digital tube.

*****/

#include "reg52.h"

/***** define macros *****/

#define MAIN_Fosc      11059200UL           //define master clock
//#define MAIN_Fosc    22118400UL         //define clock

/*****

/***** generate macro automatically, can not be changed *****/

#define Timer0_Reload  (MAIN_Fosc / 12000)

/*****
```



---

```

/***** declare local constant *****/
unsigned char code t_display[]={
//      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  black-out
      0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71,0x00};
                                                    //block code

unsigned char code T_COM[]={0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};           //bit code

/***** declare local variable *****/
//sbit   P_HC595_SER   =      P3^2;           //pin 14  SER   data input
//sbit   P_HC595_RCLK  =      P3^4;           //pin 12  RCLK  store (latch) clock
//sbit   P_HC595_SRCLK =      P3^3;           //pin 11  SRCLK Shift data clock

sbit     P_HC595_SER   =      P1^3;           //pin 14  SER   data input
sbit     P_HC595_RCLK  =      P4^1;           //pin 12  RCLK  store (latch) clock
sbit     P_HC595_SRCLK =      P1^5;           //pin 11  SRCLK Shift data clock

unsigned char   LED8[8];           //display buffer
unsigned char   display_index;     //display bit index
bit             B_1ms;             //1ms flag

/*****
void main(void)
{
    unsigned char   i, k;
    unsigned int     j;

    TMOD  =      0x01;           //Timer 0 config as 16bit timer, 12T
    TH0   =      (65536 - Timer0_Reload) / 256;
    TL0   =      (65536 - Timer0_Reload) % 256;
    ET0   =      1;
    TR0   =      1;
    EA    =      1;

    for(i=0; i<8; i++) LED8[i] = 0x10;
    j = 0;
    k = 0;
//    for(i=0; i<8; i++) LED8[i] = i;

    while(1)
    {
        while(!B_1ms);           //wait for 1ms
        B_1ms = 0;

```

---

---

```

        if(++j >= 500)                                //500ms
        {
            j = 0;
            for(i=0; i<8; i++) LED8[i] = k;           //
            if(++k > 0x10) k = 0;
                                                    //cycle display 0,1,2...,A,B..F, black-out in 8 digital tube
        }
    }
}
/*****/

/*****/
void Send_595(unsigned char dat)                      //send one byte
{
    unsigned char i;
    for(i=0; i<8; i++)
    {
        if(dat & 0x80) P_HC595_SER = 1;
        else P_HC595_SER = 0;
        P_HC595_SRCLK = 1;
        P_HC595_SRCLK = 0;
        dat = dat << 1;
    }
}

/*****/
void DisplayScan(void)                               //display scan function
{
    // Send_595(~T_COM[display_index]);              //common cathode output bit code
    // Send_595(t_display[LED8[display_index]]);    //common cathode output block code
    Send_595(T_COM[display_index]);                 //common anode output bit code
    Send_595(~t_display[LED8[display_index]]);     //common anode output block code
    P_HC595_RCLK = 1;
    P_HC595_RCLK = 0;                              //latch output data
    if(++display_index >= 8) display_index = 0;    //8 bits return 0
}

/*****/
void timer0 (void) interrupt 1                       //Timer0 1ms interrupt function
{
    TH0 = (65536 - Timer0_Reload) / 256;           //reload timing value
    TL0 = (65536 - Timer0_Reload) % 256;

    DisplayScan();                                 //1ms scanning display
    B_1ms = 1;                                    //1ms flag
}

```

---

---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program that driving 8-segment digitron -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain <reg51.h > as header file -----*/
/*-----*/

;***** the description of functions *****

;drive 8-bit digitron using common I/O ports to control 74HC595

;users can choose the clock frequency by revised macros.

;users can choose whether the digitron is common cathode or anode in display function.
;recommend to choose common cathode

;Display effect: cycle display 0,1,2...,A,B..F, black-out in 8 digital tube.

;*****/

;declare the reload value of Timer0 1ms
D_Timer0_Reload EQU (0-921) ;1ms for 11.0592MHZ
//D_Timer0_Reload EQU (0-1832) ;1ms for 22.1184MHZ

;***** declare local variable *****/
;P_HC595_SER BIT P3.2 ;pin 14 SER data input
;P_HC595_RCLK BIT P3.4 ;pin 12 RCLK store (latch) clock
;P_HC595_SRCLK BIT P3.3 ;pin 11 SRCLK Shift data clock

P_HC595_SER BIT P1.3 ;pin 14 SER data input
P_HC595_RCLK BIT P4.1 ;pin 12 RCLK store (latch) clock
P_HC595_SRCLK BIT P1.5 ;pin 11 SRCLK Shift data clock

LED8 EQU 030H
display_index DATA 038H
FLAG0 DATA 20H
B_1ms BIT FLAG0.0

;*****
;*****
```

---

```

        ORG    00H                                ;reset
        LJMP   F_MAIN_FUNC

;
        ORG    03H                                ;INT0 interrupt
        LJMP   F_INT0_interrupt
        RETI

        ORG    0BH                                ;Timer0 interrupt
        LJMP   F_Timer0_interrupt
        RETI

;
        ORG    13H                                ;INT1 interrupt
        LJMP   F_INT1_interrupt

;
        ORG    1BH                                ;Timer1 interrupt
        LJMP   F_Timer1_interrupt
        RETI

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

;*****/
F_MAIN_FUNC:
        MOV    SP,    #50H

        MOV    TMOD,  #01H                        ;Timer 0 config as 16bit timer, 12T
        MOV    TH0,   #HIGH D_Timer0_Reload      ;1ms
        MOV    TL0,   #LOW  D_Timer0_Reload
        SETB   ET0
        SETB   TR0
        SETB   EA

        MOV    R0,    #LED8
L_InitLoop1:
        MOV    @R0,   #10H
        INC    R0
        MOV    A,R0
        CJNE   A,     #(LED8+8),    L_InitLoop1

        MOV    R2,    #HIGH 500                    ;500ms
        MOV    R3,    #LOW  500
        MOV    R4,    #0
L_MainLoop:
        JNB    B_1ms, $                            ;//wait for 1ms
        CLR    B_1ms

        MOV    A,     R3
        CLR    C
        SUBB   A,     #1

```

---

```

MOV R3, A
MOV A, R2
SUBB A, #0
MOV R2, A
ORL A, R3
JNZ L_MainLoop

MOV R2, #HIGH 500 ;500ms
MOV R3, #LOW 500

MOV R0, #LED8
L_OptionLoop1:
MOV A, R4
MOV @R0, A ;
INC R0
MOV A, R0
CJNE A, #(LED8+8), L_OptionLoop1
INC R4 ;
MOV A, R4
CJNE A, #11H, L_MainLoop ;cycle display 0,1,2...,A,B..F, black-out in 8 digital tube.

MOV R4, #0
SJMP L_MainLoop

;*****/

t_display:
;0 1 2 3 4 5 6 7 8 9 A B C D E F black-out
DB 03FH,006H,05BH,04FH,066H,06DH,07DH,007H,07FH,06FH,077H,07CH,039H,05EH,079H,071H,000H
;block code

T_COM:
DB 01H,02H,04H,08H,10H,20H,40H,80H ;bit code

;*****/
F_Send_595: ;send one byte
MOV R0, #8
L_Send595_Loop:
RLC A
MOV P_HC595_SER,C
SETB P_HC595_SRCLK
CLR P_HC595_SRCLK
DJNZ R0, L_Send595_Loop
RET

;*****/
F_DisplayScan: ;display scan function
MOV DPTR, #T_COM
MOV A, display_index

```

---

```

;      MOV C  A,      @A+DPTR
;      CPL          A                                ;common cathode
;                                                  ;comment this instruction if common anode
;      LCALL F_Send_595                            ;output bit code

;      MOV  DPTR,  #t_display
;      MOV  A,    #LED8
;      ADD  A,    display_index
;      MOV  R0,   A
;      MOV  A,    @R0
;      MOV C  A,    @A+DPTR
;      CPL  A                                ;common anode
;                                                  ;comment this instruction if common anode
;      LCALL F_Send_595                            ;output block code
;      SETB P_HC595_RCLK
;      CLR  P_HC595_RCLK                        ;latch output data
;      INC  display_index
;      MOV  A,    display_index
;      CJNE A,    #8,L_QuitDisplayScan
;      MOV  display_index, #0                    ;8 bits return 0
L_QuitDisplayScan:
;      RET

;*****
;*****
;      F_Timer0_interrupt:                        ;Timer0 1ms interrupt function
;      PUSH PSW                                ;scene protection
;      PUSH ACC
;      MOV  A,    R0
;      PUSH ACC
;      PUSH DPH
;      PUSH DPL

;      MOV  TH0,  #HIGH D_Timer0_Reload        ;1ms reload timing value
;      MOV  TL0,  #LOW  D_Timer0_Reload

;      LCALL F_DisplayScan                      ;1ms scanning display
;      SETB B_1ms                              ;1ms flag

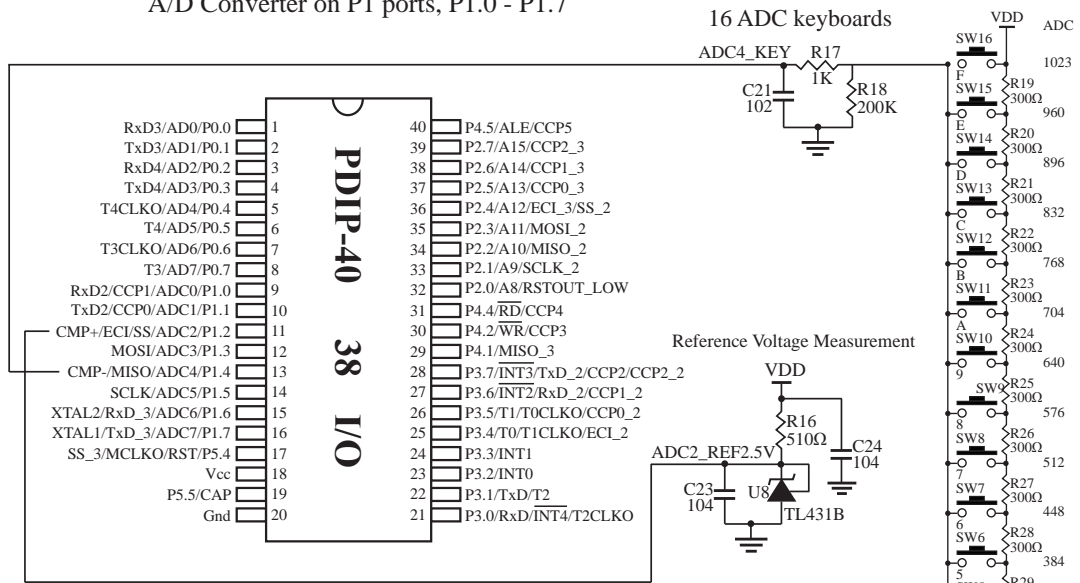
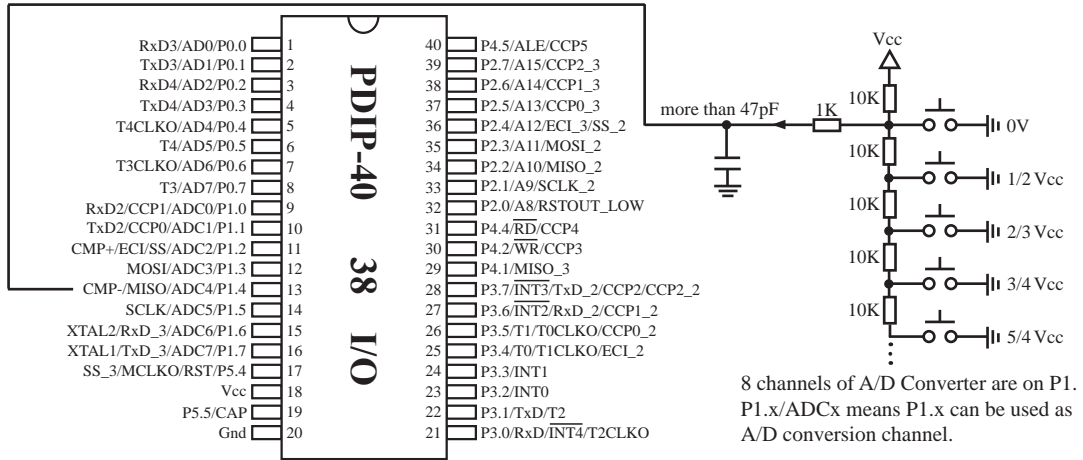
;      L_QuitT0Interrupt:
;      POP  DPL                                ;spot recovery
;      POP  DPH
;      POP  ACC
;      MOV  R0,A
;      POP  ACC
;      POP  PSW
;      RETI

;      END

```

---

# 4.17 Application Circuit using A/D Conversion to Scan Key



Method that read ADC key :

read the value of ADC key every 10ms, and save the last 3 times values. determine the key again if the variation is small. Allow some error when determining key, such as ±16 words error

---

## 4.18 Demo Program using I/O ports to Simulate I<sup>2</sup>C Interface

### 4.18.1 Master Mode using I/O ports to Simulate I<sup>2</sup>C Interface by Software

```
;/*-----*/
;/* --- STC MCU Limited. -----*/
;/* --- STC 1T Series MCU Simulate I2C Master Demo -----*/
;/* If you want to use the program or the program referenced in the */
;/* article, please specify in which data and procedures from STC */
;/*-----*/

SCL   BIT    P1.0
SDA   BIT    P1.1

;-----

        ORG    0000H

        MOV    TMOD, #20H           ;Initialize the serial port for (9600,n,8,1)
        MOV    SCON, #5AH
        MOV    A,    #-5           ;-18432000/12/32/9600
        MOV    TH1,  A
        MOV    TL1,  A
        SETB   TR1

MAIN:
        CALL   UART_RXDATA         ;receive next serial data
        MOV    R0,    A             ;save data to R0 temporarily
                                           ;read the data of I2C device IDATA 80H

        CALL   I2C_START           ;start to read
        MOV    A,    #01H
        CALL   I2C_TXBYTE          ;send address data and reading signal
        CALL   I2C_RXACK          ;receive ACK
        CALL   I2C_RXBYTE         ;receive data
        SETB   C
        CALL   I2C_TXACK          ;send NAK
        CALL   I2C_STOP           ;finish reading

        CALL   UART_TXDATA         ;send the data that have been read to UART
                                           ;push the data of R0 to I2C device IDATA 80H

        CALL   I2C_START           ;start to write
        MOV    A,    #00H
        CALL   I2C_TXBYTE          ;send address data and writing signal
        CALL   I2C_RXACK          ;receive ACK
        MOV    A,    R0
```



---

```

CALL I2C_TXBYTE      ;write data
CALL I2C_RXACK      ;receive ACK
CALL I2C_STOP       ;finish writing

JMP  MAIN

;-----
;wait for serial data
;-----
UART_RXDATA:
    JNB  RI,    $      ;wait to finish receiving
    CLR  RI
    MOV  A,    SBUF    ;save data
    RET

;-----
;send serial data
;-----
UART_TXDATA:
    JNB  TI,    $      ;wait to finish sending last a data
    CLR  TI
    MOV  SBUF,  A      ;send data
    RET

;-----
;send the first signal of I2C
;-----
I2C_START:
    CLR  SDA
    CALL I2C_DELAY    ;delay
    CLR  SCL          ;clock->low
    CALL I2C_DELAY    ;delay
    RET

;-----
;send the stop signal of I2C
;-----
I2C_STOP:
    CLR  SDA
    SETB SCL          ;clock->high
    CALL I2C_DELAY    ;delay
    SETB SDA
    CALL I2C_DELAY    ;delay
    RET

;-----
;send ACK/NAK signal
;-----

```

---

---

```

I2C_TXACK:
    MOV    SDA,    C                ;deliver ACK data
    SETB   SCL                ;clock->high
    CALL   I2C_DELAY            ;delay
    CLR    SCL                ;clock->low
    CALL   I2C_DELAY            ;delay
    SETB   SDA                ;finish sending
    RET

```

```

;-----
;receive ACK/NAK signal
;-----

```

```

I2C_RXACK:
    SETB   SDA                ;ready to read data
    SETB   SCL                ;clock->high
    CALL   I2C_DELAY            ;delay
    MOV    C,    SDA            ;read ACK signal
    CLR    SCL                ;clock->low
    CALL   I2C_DELAY            ;delay
    RET

```

```

;-----
;receive next byte of data
;-----

```

```

I2C_TXBYTE:
    MOV    R7,    #8
TXNEXT:
    RLC    A                ;shift out data bit
    MOV    SDA,    C
    SETB   SCL                ;clock->high
    CALL   I2C_DELAY            ;delay
    CLR    SCL                ;clock->low
    CALL   I2C_DELAY            ;delay
    DJNZ   R7,    TXNEXT        ;deliver next bit
    RET

```

```

;-----
;send a byte of data
;-----

```

```

I2C_RXBYTE:
    MOV    R7,    #8
RXNEXT:
    SETB   SCL                ;clock->high
    CALL   I2C_DELAY            ;delay
    MOV    C,    SDA
    RLC    A
    CLR    SCL                ;clock->low
    CALL   I2C_DELAY            ;delay

```

```

        DJNZ R7, RXNEXT ;receive next byte of data
        RET
;-----

I2C_DELAY:
        PUSH 0 ;6
        MOV R0, #1 ;4
        DJNZ R0, $ ;2 6(200K) 1(400K) [18'432'000/400'000=46]
        POP 0 ;4
        RET ;3
;-----

        END

```

### 4.18.2 Slave Mode using I/O ports to Simulate I<sup>2</sup>C Interface by Software

```

;-----*/
;*/ --- STC MCU Limited. -----*/
;*/ --- STC 1T Series MCU Simulate I2C Slave Demo -----*/
;*/ If you want to use the program or the program referenced in the */
;*/ article, please specify in which data and procedures from STC */
;*/-----*/

SCL BIT P1.0
SDA BIT P1.1
;-----

        ORG 0

RESET:
        SETB SCL
        SETB SDA

        CALL I2C_WAITSTART ;wait for first data
        CALL I2C_RXBYTE ;receive address data
        CLR C
        CALL I2C_TXACK ;respond to ACK
        SETB C ;read/write IDATA[80H - FFH]
        RRC A ;read/write bit ->C
        MOV R0, A ;push address to R0
        JC READDRESS ;C=1(read) C=0(write)

```

---

```

WRITEDATA:
    CALL I2C_RXBYTE           ;receive data
    MOV  @R0,  A              ;write in IDATA
    INC  R0                   ;address+1
    CLR  C
    CALL I2C_TXACK            ;respond to ACK
    CALL I2C_WAITSTOP         ;wait for stop signal
    JMP  RESET

READDATA:
    MOV  A,    @R0
    INC  R0
    CALL I2C_TXBYTE           ;send IDATA data
    CALL I2C_RXACK            ;receive ACK
    CALL I2C_WAITSTOP         ;wait for stop signal
    JMP  RESET

;-----
;wait for first signal
;-----
I2C_WAITSTART:
    JNB  SCL,  $              ;wait fo clock->high
    JB   SDA,  $
    JB   SCL,  $              ;wait for clock ->low
    RET

;-----
;wait for end signal
;-----
I2C_WAITSTOP:
    JNB  SCL,  $              ;wait for clock ->high
    JNB  SDA,  $
    RET

;-----
;send ACK/NAK signal
;-----
I2C_TXACK:
    MOV  SDA,  C              ;send ACK data
    JNB  SCL,  $              ;wait for clock ->high
    JB   SCL,  $              ;wait for clock ->low
    SETB SDA                  ;finish sending
    RET

;-----
;receive ACK/NAK signal
;-----
I2C_RXACK:
    SETB SDA

```

---

---

```

    JNB  SCL,  $           ;wait for clock ->high
    MOV  C,    SDA        ;read ACK signal
    JB   SCL,  $           ;wait for clock ->low
    RET

;-----
;receive a byte of data
;-----
I2C_RXBYTE:
    MOV  R7,  #8
RXNEXT:
    JNB  SCL,  $           ;wait for clock ->high
    MOV  C,    SDA        ;read data port
    RLC  A           ;save data
    JB   SCL,  $           ;wait for clock ->low
    DJNZ R7,  RXNEXT      ;receive next byte of data
    RET

;-----
;send a byte of data
;-----
I2C_TXBYTE:
    MOV  R7,  #8
TXNEXT:
    RLC  A           ;shift out data bit
    MOV  SDA,  C
    JNB  SCL,  $           ;wait for clock ->high
    JB   SCL,  $           ;wait for clock ->low
    DJNZ R7,  TXNEXT      ;deliver next byte of data
    RET

;-----

    END

```

---

# Chapter 5. Instruction System

## 5.1 Addressing Modes

Addressing modes are an integral part of each computer's instruction set. They allow specifying the source or destination of data in different ways, depending on the programming situation. There are five modes available:

- Immediate addressing
- Direct addressing
- Indirect addressing
- Register addressing
- Inherent addressing
- Indexed addressing
- Bit addressing

### 5.1.1 Immediate Addressing

This does not access any memory locations, but uses the constant number given after the instruction as the data value. The value of a constant can follow the opcode in the program memory. This operand is preceded by a # (hash) to indicate immediate mode. For example,

```
MOV  A, #70H
```

loads the Accumulator with the hex digits 70. The same number could be specified in decimal number as 112.

### 5.1.2 Direct Addressing

In direct addressing the operand is specified by an 8-bit address field in the instruction. Only 128 lowest bytes of internal data RAM and SFRs can be direct addressed. Direct addresses use the address values without the # sign. For example, to move the contents of address 4AH into address 12H the following is used:

```
MOV  12H, 4AH
```

### 5.1.3 Indirect Addressing

In indirect addressing the instruction specifies a register which contains the address of the operand. Both internal and external RAM can be indirectly addressed. Instead of giving an actual address as the operand of an instruction, a pointer to the address can be specified by indicating a register which contains the actual address.

The address register for 8-bit addresses can be R0 or R1 of the selected bank, or the Stack Pointer. The address register for 16-bit addresses can only be the 16-bit data pointer register – DPTR. Registers R0, R1 and DPTR may be used as indirection registers for this purpose, and are preceded by an @ sign to indicate the indirection. For example, to move the number 55H into the address whose value is stored in register R1 the following is used:

```
MOV  @R1, #55H
```

---

## 5.1.4 Register Addressing

The register banks, containing registers R0 through R7, can be accessed by certain instructions which carry a 3-bit register specification within the opcode of the instruction. Instructions that access the registers this way are code efficient because this mode eliminates the need of an extra address byte. When such instruction is executed, one of the eight registers in the selected bank is accessed. For example, to move the contents of register R6 to accumulator A the following is used:

```
MOV  A,  R6
```

## 5.1.5 Inherent Addressing

Some instructions do not require operands since they do not access memory. For these, the addressing is called inherent, and the main examples are the instructions for return from subroutines and interrupt service routines.

## 5.1.6 Index Addressing

Only program memory can be accessed with indexed addressing and it can only be read. This addressing mode is intended for reading look-up tables in program memory. A 16-bit base register(either DPTR or PC) points to the base of the table, and the accumulator is set up with the table entry number. Another type of indexed addressing is used in the conditional jump instruction.

In conditional jump, the destination address is computed as the sum of the base pointer and the accumulator.

## 5.1.7 Bit Addressing

Many of the instructions used by MCU are related to single bits of data. This implies that the operands can be individual bits. Examples of such instructions are:

```
SETB  45H    (same as SETB 28.5H)
CLR    P0.3
CPL    ACC.7
```

---

## 5.2 Instruction Set Summary

The STC MCU instructions are fully compatible with the traditional 8051's, which are divided among five functional groups:

- Arithmetic
- Logical
- Data transfer
- Boolean variable
- Program branching

Instruction execution speed boost summary :

There are 111 instructions in MCU. For new STC15 series MCU

24 times faster execution speed than the traditional 8051	2
12 times faster execution speed than the traditional 8051	28
8 times faster execution speed than the traditional 8051	19
6 times faster execution speed than the traditional 8051	40
4.8 times faster execution speed than the traditional 8051	8
4 times faster execution speed than the traditional 8051	14

Based on the analysis of frequency of use order statistics, STC15 series MCU instruction execution speed is faster than the traditional 8051 MCU 8 ~ 12 times in the same working environment.

Instruction execution clock count (for new STC15 series)

1 clock instruction	22
2 clock instruction	37
3 clock instruction	31
4 clock instruction	12
5 clock instruction	8
6 clock instruction	1

It needs 283 clocks to finish executing at one time all 111 instructions for STC15 series, while it needs 1944 clocks for the traditional 8051 MCU. Obviously, the speed of executing instruction for STC15 series MCU has been greatly enhanced. The average speed of STC15 series is 8~12 times faster than traditional 8051 MCU

The following tables provide a quick reference chart showing all the 8051 and STC15 series MCU instructions. Once you are familiar with the instruction set, this chart should prove a handy and quick source of reference.



STC15 series MCU with super high-speed CPU core of STC-Y5 works 20% faster than STC early 1T series (such as STC12/STC11/STC10 series) at same clock frequency.

### ARITHMETIC OPERATIONS

Mnemonic	Description	Byte	Execution clocks of traditional 8051	Execution clocks of STC15 series (super high-speed 1T 8051 CPU core of STC-Y5)	Efficiency Improved
ADD A Rn	Add register to Accumulator	1	12	1	12x
ADD A direct	Add direct byte to Accumulator	2	12	2	6x
ADD A @Ri	Add indirect RAM to Accumulator	1	12	2	6x
ADD A #data	Add immediate data to Accumulator	2	12	2	6x
ADDC A Rn	Add register to Accumulator with Carry	1	12	1	12x
ADDC A direct	Add direct byte to Accumulator with Carry	2	12	2	6x
ADDC A @Ri	Add indirect RAM to Accumulator with Carry	1	12	2	6x
ADDC A #data	Add immediate data to Acc with Carry	2	12	2	6x
SUBB A Rn	Subtract Register from Acc with borrow	1	12	1	6x
SUBB A direct	Subtract direct byte from Acc with borrow	2	12	2	6x
SUBB A @Ri	Subtract indirect RAM from ACC with borrow	1	12	2	6x
SUBB A #data	Subtract immediate data from ACC with borrow	2	12	2	6x
INC A	Increment Accumulator	1	12	1	12x
INC Rn	Increment register	1	12	2	6x
INC direct	Increment direct byte	2	12	3	4x
INC @Ri	Increment direct RAM	1	12	3	4x
DEC A	Decrement Accumulator	1	12	1	12x
DEC Rn	Decrement Register	1	12	2	6x
DEC direct	Decrement direct byte	2	12	3	4x
DEC @Ri	Decrement indirect RAM	1	12	3	4x
INC DPTR	Increment Data Pointer	1	24	1	24x
MUL AB	Multiply A & B	1	48	2	24x
DIV AB	Divide A by B	1	48	6	8x
DA A	Decimal Adjust Accumulator	1	12	3	4x

## LOGICAL OPERATIONS

Mnemonic	Description	Byte	Execution clocks of traditional 8051	Execution clocks of STC15 series (super high-speed 1T 8051 CPU core of STC-Y5)	Efficiency Improved
ANL A Rn	AND Register to Accumulator	1	12	1	12x
ANL A direct	AND direct byte to Accumulator	2	12	2	6x
ANL A @Ri	AND indirect RAM to Accumulator	1	12	2	6x
ANL A #data	AND immediate data to Accumulator	2	12	2	6x
ANL direct A	AND Accumulator to direct byte	2	12	3	4x
ANL direct #data	AND immediate data to direct byte	3	24	3	8x
ORL A Rn	OR register to Accumulator	1	12	1	12x
ORL A direct	OR direct byte to Accumulator	2	12	2	6x
ORL A, @Ri	OR indirect RAM to Accumulator	1	12	2	6x
ORL A # data	OR immediate data to Accumulator	2	12	2	6x
ORL direct A	OR Accumulator to direct byte	2	12	3	4x
ORL direct #data	OR immediate data to direct byte	3	24	3	8x
XRL A Rn	Exclusive-OR register to Accumulator	1	12	1	12x
XRL A direct	Exclusive-OR direct byte to Accumulator	2	12	2	6x
XRL A @Ri	Exclusive-OR indirect RAM to Accumulator	1	12	2	6x
XRL A # data	Exclusive-OR immediate data to Accumulator	2	12	2	6x
XRL direct A	Exclusive-OR Accumulator to direct byte	2	12	3	4x
XRL direct #data	Exclusive-OR immediate data to direct byte	3	24	3	8x
CLR A	Clear Accumulator	1	12	1	12x
CPL A	Complement Accumulator	1	12	1	12x
RL A	Rotate Accumulator Left	1	12	1	12x
RLC A	Rotate Accumulator Left through the Carry	1	12	1	12x
RR A	Rotate Accumulator Right	1	12	1	12x
RRC A	Rotate Accumulator Right through the Carry	1	12	1	12x
SWAP A	Swap nibbles within the Accumulator	1	12	1	12x

## DATA TRANSFER

Mnemonic	Description	Byte	Execution clocks of traditional 8051	Execution clocks of STC15 series (super high-speed 1T 8051 CPU core of STC-Y5)	Efficiency Improved
MOV A, Rn	Move register to Accumulator	1	12	1	12x
MOV A, direct	Move direct byte to Accumulator	2	12	2	6x
MOV A, @Ri	Move indirect RAM to Accumulator	1	12	2	6x
MOV A, #data	Move immediate data to Accumulator	2	12	2	6x
MOV Rn, A	Move Accumulator to register	1	12	1	12x
MOV Rn, direct	Move direct byte to register	2	24	3	8x
MOV Rn, #data	Move immediate data to register	2	12	2	6x
MOV direct, A	Move Accumulator to direct byte	2	12	2	6x
MOV direct, Rn	Move register to direct byte	2	24	2	12x
MOV direct, direct	Move direct byte to direct	3	24	3	8x
MOV direct, @Ri	Move indirect RAM to direct byte	2	24	3	8x
MOV direct, #data	Move immediate data to direct byte	3	24	3	8x
MOV @Ri, A	Move Accumulator to indirect RAM	1	12	2	6x
MOV @Ri, direct	Move direct byte to indirect RAM	2	24	3	8x
MOV @Ri, #data	Move immediate data to indirect RAM	2	12	2	6x
MOV DPTR, #data16	Move immediate data to indirect RAM	3	24	3	8x
MOVC A, @A+DPTR	Move Code byte relative to DPTR to Acc	1	24	5	4.8x
MOVC A, @A+PC	Move Code byte relative to PC to Acc	1	24	4	6x
MOVX A, @Ri	Move on-chip expanded RAM(8-bit addr) to Acc. Read operation	1	24	3	8x
MOVX @Ri, A	Move Acc to on-chip expanded RAM(8-bit addr). Write operation.	1	24	4	8x
MOVX A, @DPTR	Move on-chip expanded RAM(16-bit addr) to Acc. Read operation.	1	24	2	12x
MOVX @DPTR, A	Move Acc to on-chip expanded RAM (16-bit addr). Write operation.	1	24	3	8x
MOVX A, @Ri	Move Acc to External RAM(8-bit addr). Read operation.	1	24	5xN+2 see the following illustration about the value of N	*Note1
MOVX @Ri, A	Move Acc to External RAM(8-bit addr). Write operation.	1	24	5xN+3	*Note1
MOVX A, @DPTR	Move External RAM(16-bit addr) to Acc. Read operation.	1	24	5xN+1	*Note1
MOVX @DPTR, A	Move Acc to External RAM (16-bit addr). Write operation.	1	24	5xN+2	*Note1
PUSH direct	Push direct byte onto stack	2	24	3	8x
POP direct	POP direct byte from stack	2	24	2	12x
XCH A, Rn	Exchange register with Accumulator	1	12	2	6x
XCH A, direct	Exchange direct byte with Accumulator	2	12	3	4x
XCH A, @Ri	Exchange indirect RAM with Accumulator	1	12	3	4x
XCHD A, @Ri	Exchange low-order Digit indirect RAM with Acc	1	12	3	4x

When EXRTS[1:0] = [0,0], N=1 in above formula;

When EXRTS[1:0] = [0,1], N=2 in above formula;

When EXRTS[1:0] = [1,0], N=4 in above formula;

When EXRTS[1:0] = [1,1], N=8 in above formula;

EXRTS[1 0] are the bit of B0 and B1 BUS\_SPEED

**BOOLEAN VARIABLE MANIPULATION**

Mnemonic		Description	Byte	Execution clocks of traditional 8051	Execution clocks of STC15 series (super high-speed 1T 8051 CPU core of STC-Y5)	Efficiency Improved
CLR	C	Clear Carry	1	12	1	12x
CLR	bit	Clear direct bit	2	12	3	4x
SETB	C	Set Carry	1	12	1	12x
SETB	bit	Set direct bit	2	12	3	4x
CPL	C	Complement Carry	1	12	1	12x
CPL	bit	Complement direct bit	2	12	3	4x
ANL	C, bit	AND direct bit to Carry	2	24	2	12x
ANL	C, /bit	AND complement of direct bit to Carry	2	24	2	12x
ORL	C, bit	OR direct bit to Carry	2	24	2	12x
ORL	C, /bit	OR complement of direct bit to Carry	2	24	2	12x
MOV	C, bit	Move direct bit to Carry	2	12	2	12x
MOV	bit, C	Move Carry to direct bit	2	24	3	8x
JC	rel	Jump if Carry is set	2	24	3	8x
JNC	rel	Jump if Carry not set	2	24	3	8x
JB	bit, rel	Jump if direct bit is set	3	24	5	4.8x
JNB	bit, rel	Jump if direct bit is not set	3	24	5	4.8x
JBC	bit, rel	Jump if direct bit is set & clear bit	3	24	5	4.8x

**PROGRAM BRANCHING**

<b>Mnemonic</b>	<b>Description</b>	<b>Byte</b>	<b>Execution clocks of traditional 8051</b>	<b>Execution clocks of STC15 series (super high-speed 1T 8051 CPU core of STC-Y5)</b>	<b>Efficiency Improved</b>
ACALL   addr11	Absolute Subroutine Call	2	24	4	6x
LCALL   addr16	Long Subroutine Call	3	24	4	6x
RET	Return from Subroutine	1	24	4	6x
RETI	Return from interrupt	1	24	4	6x
AJMP    addr11	Absolute Jump	2	24	3	8x
LJMP    addr16	Long Jump	3	24	4	6x
SJMP    re1	Short Jump (relative addr)	2	24	3	8x
JMP     @A+DPTR	Jump indirect relative to the DPTR	1	24	5	4.8x
JZ      re1	Jump if Accumulator is Zero	2	24	4	6x
JNZ     re1	Jump if Accumulator is not Zero	2	24	4	6x
CJNE    A   direct   re1	Compare direct byte to Acc and jump if not equal	3	24	5	4.8x
CJNE    A   #data   re1	Compare immediate data to Acc and Jump if not equal	3	24	4	6x
CJNE    Rn   #data   re1	Compare immediate data to register and Jump if not equal	3	24	4	6x
CJNE    @Ri   #data   re1	Compare immediate data to indirect and jump if not equal	3	24	5	4.8x
DJNZ    Rn   re1	Decrement register and jump if not Zero	2	24	4	6x
DJNZ    direct   re1	Decrement direct byte and Jump if not Zero	3	24	5	4.8x
NOP	No Operation	1	12	1	12x

---

## 5.3 Instruction Definitions of Traditional 8051 MCU

### ACALL addr 11

---

**Function:** Absolute Call

**Description:** ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the Stack Pointer twice. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC opcode bits 7-5, and the second byte of the instruction. The subroutine called must therefore start within the same 2K block of the program memory as the first byte of the instruction following ACALL. No flags are affected.

**Example:** Initially SP equals 07H. The label "SUBRTN" is at program memory location 0345H. After executing the instruction,

ACALL SUBRTN

at location 0123H, SP will contain 09H, internal RAM locations 08H and 09H will contain 25H and 01H, respectively, and the PC will contain 0345H.

**Bytes:** 2

**Cycles:** 2

**Encoding:**

a10	a9	a8	1	0	0	1	0
-----	----	----	---	---	---	---	---

a7	a6	a5	a4	a3	a2	a1	a0
----	----	----	----	----	----	----	----

**Operation:** ACALL  
 $(PC) \leftarrow (PC) + 2$   
 $(SP) \leftarrow (SP) + 1$   
 $((SP)) \leftarrow (PC_{7-0})$   
 $(SP) \leftarrow (SP) + 1$   
 $((SP)) \leftarrow (PC_{15-8})$   
 $(PC_{10-0}) \leftarrow \text{page address}$

### ADD A,<src-byte>

---

**Function:** Add

**Description:** ADD adds the byte variable indicated to the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct register-indirect, or immediate.

**Example:** The Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B). The instruction,

ADD A,R0

will leave 6DH (01101101B) in the Accumulator with the AC flag cleared and both the carry flag and OV set to 1.

---

**ADD A,Rn****Bytes:** 1**Cycles:** 1**Encoding:**

0 0 1 0	1 r r r
---------	---------

**Operation:** ADD  
 $(A) \leftarrow (A) + (Rn)$ **ADD A,direct****Bytes:** 2**Cycles:** 1**Encoding:**

0 0 1 0	0 1 0 1	direct address
---------	---------	----------------

**Operation:** ADD  
 $(A) \leftarrow (A) + (\text{direct})$ **ADD A,@Ri****Bytes:** 1**Cycles:** 1**Encoding:**

0 0 1 0	0 1 1 i
---------	---------

**Operation:** ADD  
 $(A) \leftarrow (A) + ((Ri))$ **ADD A,#data****Bytes:** 2**Cycles:** 1**Encoding:**

0 0 1 0	0 1 0 0	immediate data
---------	---------	----------------

**Operation:** ADD  
 $(A) \leftarrow (A) + \#data$ **ADDC A,<src-byte>**

---

**Function:** Add with Carry**Description:** ADC simultaneously adds the byte variable indicated, the Carry flag and the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

**Example:** The Accumulator holds 0C3H(11000011B) and register 0 holds 0AAH (10101010B) with the Carry. The instruction,  
ADDC A,R0  
will leave 6EH (01101101B) in the Accumulator with the AC flag cleared and both the carry flag and OV set to 1.

---

**ADDC A,Rn****Bytes:** 1**Cycles:** 1**Encoding:**

0	0	1	1	1	r	r	r
---	---	---	---	---	---	---	---

**Operation:** ADDC  
 $(A) \leftarrow (A) + (C) + (Rn)$ **ADDC A,direct****Bytes:** 2**Cycles:** 1**Encoding:**

0	0	1	1	0	1	0	1
---	---	---	---	---	---	---	---

direct address
----------------

**Operation:** ADDC  
 $(A) \leftarrow (A) + (C) + (\text{direct})$ **ADDC A,@Ri****Bytes:** 1**Cycles:** 1**Encoding:**

0	0	1	1	0	1	1	i
---	---	---	---	---	---	---	---

**Operation:** ADDC  
 $(A) \leftarrow (A) + (C) + ((Ri))$ **ADDC A,#data****Bytes:** 2**Cycles:** 1**Encoding:**

0	0	1	1	0	1	0	0
---	---	---	---	---	---	---	---

immediate data
----------------

**Operation:** ADDC  
 $(A) \leftarrow (A) + (C) + \#data$ 

---

**AJMP addr 11****Function:** Absolute Jump**Description:** AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (after incrementing the PC twice), opcode bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2K block of program memory as the first byte of the instruction following AJMP.**Example:** The label "JMPADR" is at program memory location 0123H. The instruction, AJMP JMPADR is at location 0345H and will load the PC with 0123H.**Bytes:** 2**Cycles:** 2**Encoding:**

a10	a9	a8	0	0	0	0	1
-----	----	----	---	---	---	---	---

a7	a6	a5	a4	a3	a2	a1	a0
----	----	----	----	----	----	----	----

**Operation:** AJMP  
 $(PC) \leftarrow (PC) + 2$   
 $(PC_{10-0}) \leftarrow \text{page address}$



---

## ANL <dest-byte>, <src-byte>

---

**Function:** Logical-AND for byte variables

**Description:** ANL performs the bitwise logical-AND operation between the variables indicated and stores the results in the destination variable. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch not the input pins.

**Example:** If the Accumulator holds 0C3H (11000011B) and register 0 holds 55H (01010101B) then the instruction,

ANL A,R0

will leave 41H (0100001B) in the Accumulator.

When the destination is a directly addressed byte, this instruction will clear combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the Accumulator at run-time. The instruction,

ANL Pl, #01110011B

will clear bits 7, 3, and 2 of output port 1.

### ANL A,Rn

**Bytes:** 1

**Cycles:** 1

**Encoding:**

0	1	0	1	1	r	r	r
---	---	---	---	---	---	---	---

**Operation:** ANL  
(A)←(A) (Rn)

### ANL A,direct

**Bytes:** 2

**Cycles:** 1

**Encoding:**

0	1	0	1	0	1	0	1
---	---	---	---	---	---	---	---

direct address
----------------

**Operation:** ANL  
(A)←(A) (direct)

### ANL A,@Ri

**Bytes:** 1

**Cycles:** 1

**Encoding:**

0	1	0	1	0	1	i
---	---	---	---	---	---	---

**Operation:** ANL  
(A)←(A) ((Ri))

---

**ANL A,#data****Bytes:** 2**Cycles:** 1**Encoding:**

0 1 0 1	0 1 0 0	immediate data
---------	---------	----------------

**Operation:** ANL  
(A)←(A) #data**ANL direct,A****Bytes:** 2**Cycles:** 1**Encoding:**

0 1 0 1	0 0 1 0	direct address
---------	---------	----------------

**Operation:** ANL  
(direct)←(direct) (A)**ANL direct,#data****Bytes:** 3**Cycles:** 2**Encoding:**

0 1 0 1	0 0 1 1	direct address	immediate data
---------	---------	----------------	----------------

**Operation:** ANL  
(direct)←(direct) #data**ANL C, <src-bit>**

---

**Function:** Logical-AND for bit variables**Description:** If the Boolean value of the source bit is a logical 0 then clear the carry flag; otherwise leave the carry flag in its current state. A slash (“/”) preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, *but the source bit itself is not affected*. No other flgs are affected.

Only direct addressing is allowed for the source operand.

**Example:** Set the carry flag if, and only if, P1.0 = 1, ACC. 7 = 1, and OV = 0:

MOV C, P1.0 ;LOAD CARRY WITH INPUT PIN STATE

ANL C, ACC.7 ;AND CARRY WITH ACCUM. BIT.7

ANL C, /OV ;AND WITH INVERSE OF OVERFLOW FLAG

**ANL C,bit****Bytes:** 2**Cycles:** 2**Encoding:**

1 0 0 0	0 0 1 0	bit address
---------	---------	-------------

**Operation:** ANL  
(C) ← (C) (bit)

---

**ANL C, /bit****Bytes:** 2**Cycles:** 2**Encoding:**

1 0 1 1	0 0 0 0	bit address
---------	---------	-------------

**Operation:** ADD  
 $(C) \leftarrow (C) \ \overline{(\text{bit})}$ 

---

**CJNE <dest-byte>, <src-byte>, rel**

---

**Function:** Compare and Jump if Not Equal**Description:** CJNE compares the magnitudes of the first two operands, and branches if their values are not equal. The branch destination is computed by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.

The first two operands allow four addressing mode combinations: the Accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

**Example:** The Accumulator contains 34H. Register 7 contains 56H. The first instruction in the sequence

```
                CJNE   R7,#60H, NOT-EQ
;                ...           ; R7 = 60H.
NOT_EQ:        JC     REQ_LOW   ; IF R7 < 60H.
;                ...           ; R7 > 60H.
```

sets the carry flag and branches to the instruction at label NOT-EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 60H.

If the data being presented to Port 1 is also 34H, then the instruction,

```
WAIT:  CJNE  A,P1,WAIT
```

clears the carry flag and continues with the next instruction in sequence, since the Accumulator does equal the data read from P1. (If some other value was being input on P1, the program will loop at this point until the P1 data changes to 34H.)

**CJNE A,direct,rel****Bytes:** 3**Cycles:** 2**Encoding:**

1 0 1 1	0 1 0 1	direct address	rel. address
---------	---------	----------------	--------------

**Operation:**  $(PC) \leftarrow (PC) + 3$   
IF  $(A) <> (direct)$   
THEN  
 $(PC) \leftarrow (PC) + \text{relative offset}$   
IF  $(A) < (direct)$   
THEN  
 $(C) \leftarrow 1$   
ELSE  
 $(C) \leftarrow 0$

---

**CJNE A,#data,rel****Bytes:** 3**Cycles:** 2**Encoding:**

1 0 1 1	0 1 0 1
---------	---------

immediata data
----------------

rel. address
--------------

**Operation:**  $(PC) \leftarrow (PC) + 3$   
IF (A) <> (data)  
THEN  
     $(PC) \leftarrow (PC) + \text{relative offset}$   
IF (A) < (data)  
THEN  
    (C)  $\leftarrow$  1  
ELSE  
    (C)  $\leftarrow$  0

**CJNE Rn,#data,rel****Bytes:** 3**Cycles:** 2**Encoding:**

1 0 1 1	1 r r r
---------	---------

immediata data
----------------

rel. address
--------------

**Operation:**  $(PC) \leftarrow (PC) + 3$   
IF (Rn) <> (data)  
THEN  
     $(PC) \leftarrow (PC) + \text{relative offset}$   
IF (Rn) < (data)  
THEN  
    (C)  $\leftarrow$  1  
ELSE  
    (C)  $\leftarrow$  0

**CJNE @Ri,#data,rel****Bytes:** 3**Cycles:** 2**Encoding:**

1 0 1 1	0 1 1 i
---------	---------

immediate data
----------------

rel. address
--------------

**Operation:**  $(PC) \leftarrow (PC) + 3$   
IF ((Ri)) <> (data)  
THEN  
     $(PC) \leftarrow (PC) + \text{relative offset}$   
IF ((Ri)) < (data)  
THEN  
    (C)  $\leftarrow$  1  
ELSE  
    (C)  $\leftarrow$  0

---

## CLR A

---

**Function:** Clear Accumulator

**Description:** The Accumulator is cleared (all bits set on zero). No flags are affected.

**Example:** The Accumulator contains 5CH (01011100B). The instruction,  
CLR A  
will leave the Accumulator set to 00H (00000000B).

**Bytes:** 1

**Cycles:** 1

**Encoding:**

1	1	1	0	0	1	0	0
---	---	---	---	---	---	---	---

**Operation:** CLR  
(A) ← 0

---

## CLR bit

---

**Function:** Clear bit

**Description:** The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.

**Example:** Port 1 has previously been written with 5DH (01011101B). The instruction,  
CLR P1.2  
will leave the port set to 59H (01011001B).

## CLR C

**Bytes:** 1

**Cycles:** 1

**Encoding:**

1	1	0	0	0	0	1	1
---	---	---	---	---	---	---	---

**Operation:** CLR  
(C) ← 0

## CLR bit

**Bytes:** 2

**Cycles:** 1

**Encoding:**

1	1	0	0	0	0	1	0
---	---	---	---	---	---	---	---

bit address
-------------

**Operation:** CLR  
(bit) ← 0

---

## CPL A

---

**Function:** Complement Accumulator

**Description:** Each bit of the Accumulator is logically complemented (one's complement). Bits which previously contained a one are changed to a zero and vice-versa. No flags are affected.

**Example:** The Accumulator contains 5CH(01011100B). The instruction,

CPL A

will leave the Accumulator set to 0A3H (101000011B).

**Bytes:** 1

**Cycles:** 1

**Encoding:**

1	1	1	1	0	1	0	0
---	---	---	---	---	---	---	---

**Operation:** CPL  $\overline{\quad}$   
(A) ←  $\overline{\text{(A)}}$

---

## CPL bit

---

**Function:** Complement bit

**Description:** The bit variable specified is complemented. A bit which had been a one is changed to zero and vice-versa. No other flags are affected. CLR can operate on the carry or any directly addressable bit.

Note:When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, not the input pin.

**Example:** Port 1 has previously been written with 5DH (01011101B). The instruction,

CLR P1.1

CLR P1.2

will leave the port set to 59H (01011001B).

---

## CPL C

---

**Bytes:** 1

**Cycles:** 1

**Encoding:**

1	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---

**Operation:** CPL  $\overline{\quad}$   
(C) ←  $\overline{\text{(C)}}$

---

## CPL bit

---

**Bytes:** 2

**Cycles:** 1

**Encoding:**

1	0	1	1	0	0	1	0
---	---	---	---	---	---	---	---

bit address
-------------

**Operation:** CPL  $\overline{\quad}$   
(bit) ←  $\overline{\text{(bit)}}$

---

**DA A**

---

**Function:** Decimal-adjust Accumulator for Addition

**Description:** DA A adjusts the eight-bit value in the Accumulator resulting from the earlier addition of two variables (each in packed-BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition.

If Accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the Accumulator producing the proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise.

If the carry flag is now set or if the four high-order bits now exceed nine(1010xxxx-111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the Accumulator, depending on initial Accumulator and PSW conditions.

Note: DA A cannot simply convert a hexadecimal number in the Accumulator to BCD notation, nor does DA A apply to decimal subtraction.

**Example:** The Accumulator holds the value 56H(01010110B) representing the packed BCD digits of the decimal number 56. Register 3 contains the value 67H (01100111B) representing the packed BCD digits of the decimal number 67. The carry flag is set. The instruction sequence.

```
ADDC A,R3
DA A
```

will first perform a standard twos-complement binary addition, resulting in the value 0BEH (10111110) in the Accumulator. The carry and auxiliary carry flags will be cleared.

The Decimal Adjust instruction will then alter the Accumulator to the value 24H (00100100B), indicating the packed BCD digits of the decimal number 24, the low-order two digits of the decimal sum of 56,67, and the carry-in. The carry flag will be set by the Decimal Adjust instruction, indicating that a decimal overflow occurred. The true sum 56, 67, and 1 is 124.

BCD variables can be incremented or decremented by adding 01H or 99H. If the Accumulator initially holds 30H (representing the digits of 30 decimal), then the instruction sequence,

```
ADD A,#99H
DA A
```

will leave the carry set and 29H in the Accumulator, since 30+99=129. The low-order byte of the sum can be interpreted to mean 30 - 1 = 29.

---

**Bytes:** 1  
**Cycles:** 1  
**Encoding:**

1	1	0	1	0	1	0	0
---	---	---	---	---	---	---	---

  
**Operation:** DA  
 -contents of Accumulator are BCD  
 IF  $[[(\mathbf{A}_{3-0}) > 9] \vee [(\mathbf{AC}) = 1]]$   
 THEN  $(\mathbf{A}_{3-0}) \leftarrow (\mathbf{A}_{3-0}) + 6$   
 AND  
 IF  $[[(\mathbf{A}_{7-4}) > 9] \vee [(\mathbf{C}) = 1]]$   
 THEN  $(\mathbf{A}_{7-4}) \leftarrow (\mathbf{A}_{7-4}) + 6$

## DEC byte

---

**Function:** Decrement  
**Description:** The variable indicated is decremented by 1. An original value of 00H will underflow to 0FFH.  
 No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register-indirect.  
*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

**Example:** Register 0 contains 7FH (01111111B). Internal RAM locations 7EH and 7FH contain 00H and 40H, respectively. The instruction sequence,

```
DEC @R0
DEC R0
DEC @R0
```

will leave register 0 set to 7EH and internal RAM locations 7EH and 7FH set to 0FFH and 3FH.

## DEC A

**Bytes:** 1  
**Cycles:** 1  
**Encoding:**

0	0	0	1	0	1	0	0
---	---	---	---	---	---	---	---

  
**Operation:** DEC  
 $(\mathbf{A}) \leftarrow (\mathbf{A}) - 1$

## DEC Rn

**Bytes:** 1  
**Cycles:** 1  
**Encoding:**

0	0	0	1	1	r	r	r
---	---	---	---	---	---	---	---

  
**Operation:** DEC  
 $(\mathbf{Rn}) \leftarrow (\mathbf{Rn}) - 1$

---



---

**DEC direct****Bytes:** 2**Cycles:** 1**Encoding:**

0 0 0 1	0 1 0 1	direct address
---------	---------	----------------

**Operation:** DEC  
 $(\text{direct}) \leftarrow (\text{direct}) - 1$ **DEC @Ri****Bytes:** 1**Cycles:** 1**Encoding:**

0 0 0 1	0 1 1 i
---------	---------

**Operation:** DEC  
 $((Ri)) \leftarrow ((Ri)) - 1$ 

---

**DIV AB****Function:** Divide**Description:** DIV AB divides the unsigned eight-bit integer in the Accumulator by the unsigned eight-bit integer in register B. The Accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared.

*Exception:* if B had originally contained 00H, the values returned in the Accumulator and B-register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

**Example:** The Accumulator contains 251(0FBH or 11111011B) and B contains 18(12H or 00010010B). The instruction,

DIV AB

will leave 13 in the Accumulator (0DH or 00001101B) and the value 17 (11H or 00010010B) in B, since  $251 = (13 \times 18) + 17$ . Carry and OV will both be cleared.**Bytes:** 1**Cycles:** 4**Encoding:**

1 0 0 0	0 1 0 0
---------	---------

**Operation:** DIV  
 $(A)_{15-8} \leftarrow (A)/(B)$   
 $(B)_{7-0}$

---

**DJNZ <byte>, <rel-addr>**

---

**Function:** Decrement and Jump if Not Zero

**Description:** DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to 0FFH. No flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

The location decremented may be a register or directly addressed byte.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

**Example:** Internal RAM locations 40H, 50H, and 60H contain the values 01H, 70H, and 15H, respectively. The instruction sequence,

```
DJNZ 40H, LABEL_1
DJNZ 50H, LABEL_2
DJNZ 60H, LABEL_3
```

will cause a jump to the instruction at label LABEL\_2 with the values 00H, 6FH, and 15H in the three RAM locations. The first jump was not taken because the result was zero.

This instruction provides a simple way of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. The instruction sequence,

```
MOV R2,#8
TOOOLE: CPL P1.7
DJNZ R2, TOOGLE
```

will toggle P1.7 eight times, causing four output pulses to appear at bit 7 of output Port 1. Each pulse will last three machine cycles; two for DJNZ and one to alter the pin.

**DJNZ Rn,rel**

**Bytes:** 2

**Cycles:** 2

**Encoding:**

1	1	0	1
---	---	---	---

1	r	r	r
---	---	---	---

rel. address
--------------

**Operation:** DJNZ  
 $(PC) \leftarrow (PC) + 2$   
 $(Rn) \leftarrow (Rn) - 1$   
IF  $(Rn) > 0$  or  $(Rn) < 0$   
THEN  
 $(PC) \leftarrow (PC) + rel$

**DJNZ direct, rel**

**Bytes:** 3

**Cycles:** 2

**Encoding:**

1	1	0	1
---	---	---	---

0	1	0	1
---	---	---	---

direct address
----------------

rel. address
--------------

---

**Operation:** DJNZ  
 $(PC) \leftarrow (PC) + 2$   
 $(direct) \leftarrow (direct) - 1$   
IF  $(direct) > 0$  or  $(direct) < 0$   
THEN  
 $(PC) \leftarrow (PC) + rel$

### INC <byte>

---

**Function:** Increment

**Description:** INC increments the indicated variable by 1. An original value of 0FFH will overflow to 00H. No flags are affected. Three addressing modes are allowed: register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

**Example:** Register 0 contains 7EH (01111110B). Internal RAM locations 7EH and 7FH contain 0FFH and 40H, respectively. The instruction sequence,

```
INC @R0
INC R0
INC @R0
```

will leave register 0 set to 7FH and internal RAM locations 7EH and 7FH holding (respectively) 00H and 41H.

### INC A

**Bytes:** 1

**Cycles:** 1

**Encoding:**

0	0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---	---

**Operation:** INC  
 $(A) \leftarrow (A) + 1$

### INC Rn

**Bytes:** 1

**Cycles:** 1

**Encoding:**

0	0	0	0	0	1	r	r	r	r
---	---	---	---	---	---	---	---	---	---

**Operation:** INC  
 $(Rn) \leftarrow (Rn) + 1$

### INC direct

**Bytes:** 2

**Cycles:** 1

**Encoding:**

0	0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---	---

direct address
----------------

**Operation:** INC  
 $(direct) \leftarrow (direct) + 1$

---

**INC @Ri****Bytes:** 1**Cycles:** 1**Encoding:**

0 0 0 0	0 1 1 i
---------	---------

**Operation:** INC  
 $((Ri)) \leftarrow ((Ri)) + 1$ 

---

**INC DPTR**

---

**Function:** Increment Data Pointer**Description:** Increment the 16-bit data pointer by 1. A 16-bit increment (modulo  $2^{16}$ ) is performed; an overflow of the low-order byte of the data pointer (DPL) from 0FFH to 00H will increment the high-order-byte (DPH). No flags are affected. This is the only 16-bit register which can be incremented.**Example:** Register DPH and DPL contains 12H and 0FEH, respectively. The instruction sequence,  
INC DPTR  
INC DPTR  
INC DPTR  
will change DPH and DPL to 13H and 01H.**Bytes:** 1**Cycles:** 2**Encoding:**

1 0 1 0	0 0 1 1
---------	---------

**Operation:** INC  
 $(DPTR) \leftarrow (DPTR) + 1$ 

---

**JB bit, rel**

---

**Function:** Jump if Bit set**Description:** If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified. No flags are affected.***Example:** The data present at input port 1 is 11001010B. The Accumulator holds 56 (01010110B). The instruction sequence,  
JB P1.2, LABEL1  
JB ACC.2, LABEL2  
will cause program execution to branch to the instruction at label LABEL2.**Bytes:** 3**Cycles:** 2**Encoding:**

0 0 1 0	0 0 0 0	bit address	rel. address
---------	---------	-------------	--------------

**Operation:** JB  
 $(PC) \leftarrow (PC) + 3$   
IF (bit) = 1  
THEN  
 $(PC) \leftarrow (PC) + rel$

---

**JBC bit, rel**

---

**Function:** Jump if Bit is set and Clear bit

**Description:** If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. *The bit will not be cleared if it is already a zero.* The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.

Note: When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, not the input pin.

**Example:** The Accumulator holds 56H (01010110B). The instruction sequence,

```
JBC ACC.3, LABEL1
JBC ACC.2, LABEL2
```

will cause program execution to continue at the instruction identified by the label LABEL2, with the Accumulator modified to 52H (01010010B).

**Bytes:** 3

**Cycles:** 2

**Encoding:**

0 0 0 1	0 0 0 0	bit address	rel. address
---------	---------	-------------	--------------

**Operation:** JBC  
 $(PC) \leftarrow (PC) + 3$   
IF (bit) = 1  
THEN  
    (bit)  $\leftarrow$  0  
     $(PC) \leftarrow (PC) + \text{rel}$

---

**JC rel**

---

**Function:** Jump if Carry is set

**Description:** If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.

**Example:** The carry flag is cleared. The instruction sequence,

```
JC LABEL1
CPL C
JC LABEL2s
```

will set the carry and cause program execution to continue at the instruction identified by the label LABEL2.

**Bytes:** 2

**Cycles:** 2

**Encoding:**

0 1 0 0	0 0 0 0	rel. address
---------	---------	--------------

**Operation:** JC  
 $(PC) \leftarrow (PC) + 2$   
IF (C) = 1  
THEN  
     $(PC) \leftarrow (PC) + \text{rel}$

---

## JMP @A+DPTR

---

**Function:** Jump indirect

**Description:** Add the eight-bit unsigned contents of the Accumulator with the sixteen-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo  $2^{16}$ ): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the Accumulator nor the Data Pointer is altered. No flags are affected.

**Example:** An even number from 0 to 6 is in the Accumulator. The following sequence of instructions will branch to one of four AJMP instructions in a jump table starting at JMP\_TBL:

```
                MOV    DPTR, #JMP_TBL
                JMP    @A+DPTR
JMP-TBL:       AJMP   LABEL0
                AJMP   LABEL1
                AJMP   LABEL2
                AJMP   LABEL3
```

If the Accumulator equals 04H when starting this sequence, execution will jump to label LABEL2. Remember that AJMP is a two-byte instruction, so the jump instructions start at every other address.

**Bytes:** 1

**Cycles:** 2

**Encoding:**

0 1 1 1	0 0 1 1
---------	---------

**Operation:** JMP  
(PC)  $\leftarrow$  (A) + (DPTR)

---

## JNB bit, rel

---

**Function:** Jump if Bit is not set

**Description:** If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

**Example:** The data present at input port 1 is 11001010B. The Accumulator holds 56H (01010110B). The instruction sequence,

```
JNB    P1.3, LABEL1
JNB    ACC.3, LABEL2
```

will cause program execution to continue at the instruction at label LABEL2

**Bytes:** 3

**Cycles:** 2

**Encoding:**

0 0 1 1	0 0 0 0	bit address	rel. address
---------	---------	-------------	--------------

**Operation:** JNB  
(PC)  $\leftarrow$  (PC) + 3  
IF (bit) = 0  
THEN (PC)  $\leftarrow$  (PC) + rel

---

**JNC rel**

---

**Function:** Jump if Carry not set

**Description:** If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified

**Example:** The carry flag is set. The instruction sequence,

```
JNC LABEL1
CPL C
JNC LABEL2
```

will clear the carry and cause program execution to continue at the instruction identified by the label LABEL2.

**Bytes:** 2

**Cycles:** 2

**Encoding:**

0 1 0 1	0 0 0 0	rel. address
---------	---------	--------------

**Operation:** JNC  
 $(PC) \leftarrow (PC) + 2$   
IF  $(C) = 0$   
THEN  $(PC) \leftarrow (PC) + rel$

---

**JNZ rel**

---

**Function:** Jump if Accumulator Not Zero

**Description:** If any bit of the Accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

**Example:** The Accumulator originally holds 00H. The instruction sequence,

```
JNZ LABEL1
INC A
JNZ LAEEL2
```

will set the Accumulator to 01H and continue at label LABEL2.

**Bytes:** 2

**Cycles:** 2

**Encoding:**

0 1 1 1	0 0 0 0	rel. address
---------	---------	--------------

**Operation:** JNZ  
 $(PC) \leftarrow (PC) + 2$   
IF  $(A) \neq 0$   
THEN  $(PC) \leftarrow (PC) + rel$

---

**JZ rel**

---

**Function:** Jump if Accumulator Zero

**Description:** If all bits of the Accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

**Example:** The Accumulator originally contains 01H. The instruction sequence,

```
JZ LABEL1
DEC A
JZ LAEEL2
```

will change the Accumulator to 00H and cause program execution to continue at the instruction identified by the label LABEL2.

**Bytes:** 2

**Cycles:** 2

**Encoding:**

0 1 1 0	0 0 0 0	rel. address
---------	---------	--------------

**Operation:** JZ  
 $(PC) \leftarrow (PC) + 2$   
IF  $(A) = 0$   
THEN  $(PC) \leftarrow (PC) + rel$

---

**LCALL addr16**

---

**Function:** Long call

**Description:** LCALL calls a subroutine loaded at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the Stack Pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64K-byte program memory address space. No flags are affected.

**Example:** Initially the Stack Pointer equals 07H. The label "SUBRTN" is assigned to program memory location 1234H. After executing the instruction,

```
LCALL SUBRTN
```

at location 0123H, the Stack Pointer will contain 09H, internal RAM locations 08H and 09H will contain 26H and 01H, and the PC will contain 1234H.

**Bytes:** 3

**Cycles:** 2

**Encoding:**

0 0 0 1	0 0 1 0	addr15-addr8	addr7-addr0
---------	---------	--------------	-------------

**Operation:** LCALL  
 $(PC) \leftarrow (PC) + 3$   
 $(SP) \leftarrow (SP) + 1$   
 $((SP)) \leftarrow (PC_{7-0})$   
 $(SP) \leftarrow (SP) + 1$   
 $((SP)) \leftarrow (PC_{15-8})$   
 $(PC) \leftarrow addr_{15-0}$



---

## LJMP addr16

---

**Function:** Long Jump

**Description:** LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.

**Example:** The label "JMPADR" is assigned to the instruction at program memory location 1234H. The instruction,

```
LJMP  JMPADR
```

at location 0123H will load the program counter with 1234H.

**Bytes:** 3

**Cycles:** 2

**Encoding:**

0 0 0 0	0 0 1 0	addr15-addr8	addr7-addr0
---------	---------	--------------	-------------

**Operation:** LJMP  
(PC) ← addr<sub>15:0</sub>

---

## MOV <dest-byte> , <src-byte>

---

**Function:** Move byte variable

**Description:** The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.

This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.

**Example:** Internal RAM location 30H holds 40H. The value of RAM location 40H is 10H. The data present at input port 1 is 11001010B (0CAH).

```
MOV  R0, #30H  ;R0 <= 30H
MOV  A, @R0    ;A <= 40H
MOV  R1, A     ;R1 <= 40H
MOV  B, @R1    ;B <= 10H
MOV  @R1, P1   ;RAM (40H) <= 0CAH
MOV  P2, P1    ;P2 #0CAH
```

leaves the value 30H in register 0, 40H in both the Accumulator and register 1, 10H in register B, and 0CAH(11001010B) both in RAM location 40H and output on port 2.

## MOV A,Rn

**Bytes:** 1

**Cycles:** 1

**Encoding:**

1 1 1 0	1 r r r
---------	---------

**Operation:** MOV  
(A) ← (Rn)

---

**\*MOV A,direct**

Bytes: 2

Cycles: 1

Encoding: 

1 1 1 0	0 1 0 1
---------	---------

direct address
----------------

Operation: MOV  
(A) $\leftarrow$ (direct)

**\*MOV A,ACC is not a valid instruction**

**MOV A,@Ri**

Bytes: 1

Cycles: 1

Encoding: 

1 1 1 0	0 1 1 i
---------	---------

Operation: MOV  
(A)  $\leftarrow$  ((Ri))

**MOV A,#data**

Bytes: 2

Cycles: 1

Encoding: 

0 1 1 1	0 1 0 0
---------	---------

immediate data
----------------

Operation: MOV  
(A) $\leftarrow$  #data

**MOV Rn,A**

Bytes: 1

Cycles: 1

Encoding: 

1 1 1 1	1 r r r
---------	---------

Operation: MOV  
(Rn) $\leftarrow$ (A)

**MOV Rn,direct**

Bytes: 2

Cycles: 2

Encoding: 

1 0 1 0	1 r r r
---------	---------

direct addr.
--------------

Operation: MOV  
(Rn) $\leftarrow$ (direct)

**MOV Rn,#data**

Bytes: 2

Cycles: 1

Encoding: 

0 1 1 1	1 r r r
---------	---------

immediate data
----------------

Operation: MOV  
(Rn)  $\leftarrow$  #data

---

**MOV direct, A**

Bytes: 2

Cycles: 1

Encoding: 

1 1 1 1	0 1 0 1
---------	---------

direct address

Operation: MOV  
(direct) ← (A)

**MOV direct, Rn**

Bytes: 2

Cycles: 2

Encoding: 

1 0 0 0	1 r r r
---------	---------

direct address

Operation: MOV  
(direct) ← (Rn)

**MOV direct, direct**

Bytes: 3

Cycles: 2

Encoding: 

1 0 0 0	0 1 0 1
---------	---------

dir.addr. (src)

Operation: MOV  
(direct) ← (direct)

**MOV direct, @Ri**

Bytes: 2

Cycles: 2

Encoding: 

1 0 0 0	0 1 1 i
---------	---------

direct addr.

Operation: MOV  
(direct) ← ((Ri))

**MOV direct, #data**

Bytes: 3

Cycles: 2

Encoding: 

0 1 1 1	0 1 0 1
---------	---------

direct address

Operation: MOV  
(direct) ← #data

**MOV @Ri, A**

Bytes: 1

Cycles: 1

Encoding: 

1 1 1 1	0 1 1 i
---------	---------

Operation: MOV  
((Ri)) ← (A)

---

**MOV @Ri, direct****Bytes:** 2**Cycles:** 2**Encoding:**

1 0 1 0	0 1 1 i
---------	---------

direct addr.
--------------

**Operation:** MOV  
((Ri)) ← (direct)**MOV @Ri, #data****Bytes:** 2**Cycles:** 1**Encoding:**

0 1 1 1	0 1 1 i
---------	---------

immediate data
----------------

**Operation:** MOV  
((Ri)) ← #data

---

**MOV <dest-bit>, <src-bit>**

---

**Function:** Move bit data**Description:** The Boolean variable indicated by the second operand is copied into the location specified by the first operand. One of the operands must be the carry flag; the other may be any directly addressable bit. No other register or flag is affected.**Example:** The carry flag is originally set. The data present at input Port 3 is 11000101B. The data previously written to output Port 1 is 35H (00110101B).

```
MOV    P1.3, C
MOV    C, P3.3
MOV    P1.2, C
```

will leave the carry cleared and change Port 1 to 39H (00111001B).

**MOV C,bit****Bytes:** 2**Cycles:** 1**Encoding:**

1 0 1 0	0 0 1 1
---------	---------

bit address
-------------

**Operation:** MOV  
(C) ← (bit)**MOV bit,C****Bytes:** 2**Cycles:** 2**Encoding:**

1 0 0 1	0 0 1 0
---------	---------

bit address
-------------

**Operation:** MOV  
(bit) ← (C)

---

## MOV DPTR, #data 16

---

**Function:** Load Data Pointer with a 16-bit constant

**Description:** The Data Pointer is loaded with the 16-bit constant indicated. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected. This is the only instruction which moves 16 bits of data at once.

**Example:** The instruction,  
MOV DPTR, #1234H  
will load the value 1234H into the Data Pointer: DPH will hold 12H and DPL will hold 34H.

**Bytes:** 3

**Cycles:** 2

**Encoding:**

1	0	0	1
---	---	---	---

0	0	0	0
---	---	---	---

immediate data 15-8			
---------------------	--	--	--

**Operation:** MOV  
(DPTR) ← #data<sub>15,0</sub>  
DPH DPL ← #data<sub>15,8</sub> #data<sub>7,0</sub>

---

## MOVC A, @A+ <base-reg>

---

**Function:** Move Code byte

**Description:** The MOVC instructions load the Accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit Accumulator contents and the contents of a sixteen-bit base register, which may be either the Data Pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the Accumulator; otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

**Example:** A value between 0 and 3 is in the Accumulator. The following instructions will translate the value in the Accumulator to one of four values defined by the DB (define byte) directive.

```
REL-PC: INC    A
          MOVC  A, @A+PC
          RET
          DB    66H
          DB    77H
          DB    88H
          DB    99H
```

If the subroutine is called with the Accumulator equal to 01H, it will return with 77H in the Accumulator. The INC A before the MOVC instruction is needed to “get around” the RET instruction above the table. If several bytes of code separated the MOVC from the table, the corresponding number would be added to the Accumulator instead.

---

## MOVC A, @A+DPTR

**Bytes:** 1

**Cycles:** 2

**Encoding:**

1	0	0	1
---	---	---	---

0	0	1	1
---	---	---	---

**Operation:** MOVC  
(A) ← ((A)+(DPTR))

---

**MOVC A,@A+PC****Bytes:** 1**Cycles:** 2**Encoding:**

1 0 0 0	0 0 1 1
---------	---------

**Operation:** MOVC  
(PC) ← (PC)+1  
(A) ← ((A)+(PC))**MOVX <dest-byte> , <src-byte>**

---

**Function:** Move External**Description:** The MOVX instructions transfer data between the Accumulator and a byte of external data memory, hence the “X” appended to MOV. There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM.

In the first type, the contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or for a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

In the second type of MOVX instruction, the Data Pointer generates a sixteen-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large data arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.

It is possible in some situations to mix the two MOVX types. A large RAM array with its high-order address lines driven by P2 can be addressed via the Data Pointer, or with code to output high-order address bits to P2 followed by a MOVX instruction using R0 or R1.

**Example:** An external 256 byte RAM using multiplexed address/data lines (e.g., an Intel 8155 RAM/I/O/Timer) is connected to the 8051 Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Registers 0 and 1 contain 12H and 34H. Location 34H of the external RAM holds the value 56H. The instruction sequence,

```
MOVX    A, @R1
MOVX    @R0, A
```

copies the value 56H into both the Accumulator and external RAM location 12H.

**MOVX A,@Ri****Bytes:** 1**Cycles:** 2**Encoding:**

1 1 1 0	0 0 1 i
---------	---------

**Operation:** MOVX  
(A) ← ((Ri))

---

**MOVX A,@DPTR****Bytes:** 1**Cycles:** 2**Encoding:**

1 1 1 0	0 0 0 0
---------	---------

**Operation:** MOVX  
(A) ← ((DPTR))**MOVX @Ri, A****Bytes:** 1**Cycles:** 2**Encoding:**

1 1 1 1	0 0 1 i
---------	---------

**Operation:** MOVX  
((Ri)) ← (A)**MOVX @DPTR, A****Bytes:** 1**Cycles:** 2**Encoding:**

1 1 1 1	0 0 0 0
---------	---------

**Operation:** MOVX  
(DPTR) ← (A)

---

**MUL AB**

---

**Function:** Multiply**Description:** MUL AB multiplies the unsigned eight-bit integers in the Accumulator and register B. The low-order byte of the sixteen-bit product is left in the Accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared**Example:** Originally the Accumulator holds the value 80 (50H). Register B holds the value 160 (0A0H). The instruction,

MUL AB

will give the product 12,800 (3200H), so B is changed to 32H (00110010B) and the Accumulator is cleared. The overflow flag is set, carry is cleared.

**Bytes:** 1**Cycles:** 4**Encoding:**

1 0 1 0	0 1 0 0
---------	---------

**Operation:** MUL  
(A)<sub>7-0</sub> ← (A) × (B)  
(B)<sub>15-8</sub>

---

## NOP

---

**Function:** No Operation

**Description:** Execution continues at the following instruction. Other than the PC, no registers or flags are affected.

**Example:** It is desired to produce a low-going output pulse on bit 7 of Port 2 lasting exactly 5 cycles. A simple SETB/CLR sequence would generate a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming no interrupts are enabled) with the instruction sequence.

```
CLR    P2.7
NOP
NOP
NOP
NOP
SETB   P2.7
```

**Bytes:** 1

**Cycles:** 1

**Encoding:**

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

**Operation:** NOP  
(PC) ← (PC)+1

---

## ORL <dest-byte> , <src-byte>

---

**Function:** Logical-OR for byte variables

**Description:** ORL performs the bitwise logical-OR operation between the indicated variables, storing the results in the destination byte. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

**Example:** If the Accumulator holds 0C3H (11000011B) and R0 holds 55H (01010101B) then the instruction,

```
ORL    A, R0
```

will leave the Accumulator holding the value 0D7H (11010111B).

When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte, which may be either a constant data value in the instruction or a variable computed in the Accumulator at run-time. The instruction,

```
ORL    P1, #00110010B
```

will set bits 5, 4, and 1 of output Port 1.



---

**ORL A,Rn****Bytes:** 1**Cycles:** 1**Encoding:**

0	1	0	0	1	r	r	r
---	---	---	---	---	---	---	---

**Operation:** ORL $(A) \leftarrow (A) \ (Rn)$ **ORL A,direct****Bytes:** 2**Cycles:** 1**Encoding:**

0	1	0	0	0	1	0	1
---	---	---	---	---	---	---	---

direct address
----------------

**Operation:** ORL $(A) \leftarrow (A) \ (\text{direct})$ **ORL A,@Ri****Bytes:** 1**Cycles:** 1**Encoding:**

0	1	0	0	0	1	1	i
---	---	---	---	---	---	---	---

**Operation:** ORL $(A) \leftarrow (A) \ ((Ri))$ **ORL A,#data****Bytes:** 2**Cycles:** 1**Encoding:**

0	1	0	0	0	1	0	0
---	---	---	---	---	---	---	---

immediate data
----------------

**Operation:** ORL $(A) \leftarrow (A) \ \#data$ **ORL direct, A****Bytes:** 2**Cycles:** 1**Encoding:**

0	1	0	0	0	0	1	0
---	---	---	---	---	---	---	---

direct address
----------------

**Operation:** ORL $(\text{direct}) \leftarrow (\text{direct}) \ (A)$ **ORL direct, #data****Bytes:** 3**Cycles:** 2**Encoding:**

0	1	0	0	0	0	1	1
---	---	---	---	---	---	---	---

direct address
----------------

immediate data
----------------

**Operation:** ORL $(\text{direct}) \leftarrow (\text{direct}) \ \#data$

---

**ORL C, <src-bit>**

---

**Function:** Logical-OR for bit variables

**Description:** Set the carry flag if the Boolean value is a logical 1; leave the carry in its current state otherwise. A slash (“/”) preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

**Example:** Set the carry flag if and only if P1.0 = 1, ACC. 7 = 1, or OV = 0:  
MOV C, P1.0 ;LOAD CARRY WITH INPUT PIN P10  
ORL C, ACC.7 ;OR CARRY WITH THE ACC.BIT 7  
ORL C, /OV ;OR CARRY WITH THE INVERSE OF OV

**ORL C, bit**

**Bytes:** 2

**Cycles:** 2

**Encoding:**

0	1	1	1	0	0	1	0
---	---	---	---	---	---	---	---

 bit address

**Operation:** ORL  
(C) ← (C) (bit)

**ORL C, /bit**

**Bytes:** 2

**Cycles:** 2

**Encoding:**

1	0	1	0	0	0	0	0
---	---	---	---	---	---	---	---

 bit address

**Operation:** ORL  
(C) ← (C)  $\overline{\text{(bit)}}$

---

**POP direct**

---

**Function:** Pop from stack

**Description:** The contents of the internal RAM location addressed by the Stack Pointer is read, and the Stack Pointer is decremented by one. The value read is then transferred to the directly addressed byte indicated. No flags are affected.

**Example:** The Stack Pointer originally contains the value 32H, and internal RAM locations 30H through 32H contain the values 20H, 23H, and 01H, respectively. The instruction sequence,  
POP DPH  
POP DPL  
will leave the Stack Pointer equal to the value 30H and the Data Pointer set to 0123H. At this point the instruction,  
POP SP  
will leave the Stack Pointer set to 20H. Note that in this special case the Stack Pointer was decremented to 2FH before being loaded with the value popped (20H).

**Bytes:** 2

**Cycles:** 2

**Encoding:**

1	1	0	1	0	0	0	0
---	---	---	---	---	---	---	---

 direct address

**Operation:** POP  
(direct) ← ((SP))  
(SP) ← (SP) - 1

---

## PUSH direct

---

**Function:** Push onto stack

**Description:** The Stack Pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the Stack Pointer. Otherwise no flags are affected.

**Example:** On entering interrupt routine the Stack Pointer contains 09H. The Data Pointer holds the value 0123H. The instruction sequence,

```
PUSH DPL
PUSH DPH
```

will leave the Stack Pointer set to 0BH and store 23H and 01H in internal RAM locations 0AH and 0BH, respectively.

**Bytes:** 2

**Cycles:** 2

**Encoding:**

1	1	0	0
---	---	---	---

0	0	0	0
---	---	---	---

direct address
----------------

**Operation:** PUSH  
 $(SP) \leftarrow (SP) + 1$   
 $((SP)) \leftarrow (\text{direct})$

---

## RET

---

**Function:** Return from subroutine

**Description:** RET pops the high-and low-order bytes of the PC successively from the stack, decrementing the Stack Pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.

**Example:** The Stack Pointer originally contains the value 0BH. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

```
RET
```

will leave the Stack Pointer equal to the value 09H. Program execution will continue at location 0123H.

**Bytes:** 1

**Cycles:** 2

**Encoding:**

0	0	1	0
---	---	---	---

0	0	1	0
---	---	---	---

**Operation:** RET  
 $(PC_{15-8}) \leftarrow ((SP))$   
 $(SP) \leftarrow (SP) - 1$   
 $(PC_{7-0}) \leftarrow ((SP))$   
 $(SP) \leftarrow (SP) - 1$

---

## RETI

---

**Function:** Return from interrupt

**Description:** RETI pops the high- and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The Stack Pointer is left decremented by two. No other registers are affected; the PSW is not automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower- or same-level interrupt had been pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.

**Example:** The Stack Pointer originally contains the value 0BH. An interrupt was detected during the instruction ending at location 0122H. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

RETI

will leave the Stack Pointer equal to 09H and return program execution to location 0123H.

**Bytes:** 1

**Cycles:** 2

**Encoding:**

0	0	1	1	0	0	1	0
---	---	---	---	---	---	---	---

**Operation:** RETI  
 $(PC_{15-8}) \leftarrow ((SP))$   
 $(SP) \leftarrow (SP) - 1$   
 $(PC_{7-0}) \leftarrow ((SP))$   
 $(SP) \leftarrow (SP) - 1$

---

## RL A

---

**Function:** Rotate Accumulator Left

**Description:** The eight bits in the Accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.

**Example:** The Accumulator holds the value 0C5H (11000101B). The instruction,

RL A

leaves the Accumulator holding the value 8BH (10001011B) with the carry unaffected.

**Bytes:** 1

**Cycles:** 1

**Encoding:**

0	0	1	0	0	0	1	1
---	---	---	---	---	---	---	---

**Operation:** RL  
 $(A_{n+1}) \leftarrow (A_n) \quad n = 0-6$   
 $(A_0) \leftarrow (A_7)$

---

**RLC A**

---

**Function:** Rotate Accumulator Left through the Carry flag

**Description:** The eight bits in the Accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.

**Example:** The Accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction, RLC A leaves the Accumulator holding the value 8BH (10001011B) with the carry set.

**Bytes:** 1

**Cycles:** 1

**Encoding:**

0	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---

**Operation:** RLC  
 $(A_{n+1}) \leftarrow (A_n) \quad n = 0-6$   
 $(A_0) \leftarrow (C)$   
 $(C) \leftarrow (A_7)$

---

**RR A**

---

**Function:** Rotate Accumulator Right

**Description:** The eight bits in the Accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.

**Example:** The Accumulator holds the value 0C5H (11000101B). The instruction, RR A leaves the Accumulator holding the value 0E2H (11100010B) with the carry unaffected.

**Bytes:** 1

**Cycles:** 1

**Encoding:**

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

**Operation:** RR  
 $(A_n) \leftarrow (A_{n+1}) \quad n = 0 - 6$   
 $(A_7) \leftarrow (A_0)$

---

**RRC A**

---

**Function:** Rotate Accumulator Right through the Carry flag

**Description:** The eight bits in the Accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position. No other flags are affected.

**Example:** The Accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction, RRC A leaves the Accumulator holding the value 62H (01100010B) with the carry set.

**Bytes:** 1

**Cycles:** 1

**Encoding:**

0	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---

**Operation:** RRC  
 $(A_{n+1}) \leftarrow (A_n) \quad n = 0-6$   
 $(A_7) \leftarrow (C)$   
 $(C) \leftarrow (A_0)$

---

**SETB <bit>**

---

**Function:** Set bit

**Description:** SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected

**Example:** The carry flag is cleared. Output Port 1 has been written with the value 34H (00110100B). The instructions,  
SETB C  
SETB P1.0  
will leave the carry flag set to 1 and change the data output on Port 1 to 35H (00110101B).

**SETB C**

**Bytes:** 1

**Cycles:** 1

**Encoding:**

1	1	0	1	0	0	1	1
---	---	---	---	---	---	---	---

**Operation:** SETB  
(C) ← 1

**SETB bit**

**Bytes:** 2

**Cycles:** 1

**Encoding:**

1	1	0	1	0	0	1	0
---	---	---	---	---	---	---	---

bit address
-------------

**Operation:** SETB  
(bit) ← 1

---

**SJMP rel**

---

**Function:** Short Jump

**Description:** Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128bytes preceding this instruction to 127 bytes following it.

**Example:** The label "RELADR" is assigned to an instruction at program memory location 0123H. The instruction,  
SJMP RELADR  
will assemble into location 0100H. After the instruction is executed, the PC will contain the value 0123H.  
(Note: Under the above conditions the instruction following SJMP will be at 102H. Therefore, the displacement byte of the instruction will be the relative offset (0123H - 0102H) = 21H. Put another way, an SJMP with a displacement of 0FEH would be an one-instruction infinite loop).

**Bytes:** 2

**Cycles:** 2

**Encoding:**

1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

rel. address
--------------

**Operation:** SJMP  
(PC) ← (PC)+2  
(PC) ← (PC)+rel

---

**SUBB A, <src-byte>**

---

**Function:** Subtract with borrow

**Description:** SUBB subtracts the indicated variable and the carry flag together from the Accumulator, leaving the result in the Accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7, and clears C otherwise. (If C was set before executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the Accumulator along with the source operand). AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

**Example:** The Accumulator holds 0C9H (11001001B), register 2 holds 54H (01010100B), and the carry flag is set. The instruction,

SUBB A, R2

will leave the value 74H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.

Notice that 0C9H minus 54H is 75H. The difference between this and the above result is due to the carry (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should be explicitly cleared by a CLR C instruction.

**SUBB A, Rn**

**Bytes:** 1

**Cycles:** 1

**Encoding:**

1	0	0	1	1	r	r	r
---	---	---	---	---	---	---	---

**Operation:** SUBB  
 $(A) \leftarrow (A) - (C) - (Rn)$

**SUBB A, direct**

**Bytes:** 2

**Cycles:** 1

**Encoding:**

1	0	0	1	0	1	0	1
---	---	---	---	---	---	---	---

direct address
----------------

**Operation:** SUBB  
 $(A) \leftarrow (A) - (C) - (\text{direct})$

**SUBB A, @Ri**

**Bytes:** 1

**Cycles:** 1

**Encoding:**

1	0	0	1	0	1	1	i
---	---	---	---	---	---	---	---

**Operation:** SUBB  
 $(A) \leftarrow (A) - (C) - ((Ri))$

---

**SUBB A, #data****Bytes:** 2**Cycles:** 1**Encoding:**

1	0	0	1	0	1	0	0
---	---	---	---	---	---	---	---

immediate data
----------------

**Operation:** SUBB  
 $(A) \leftarrow (A) - (C) - \#data$ 

---

**SWAP A****Function:** Swap nibbles within the Accumulator**Description:** SWAP A interchanges the low- and high-order nibbles (four-bit fields) of the Accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction. No flags are affected.**Example:** The Accumulator holds the value 0C5H (11000101B). The instruction,  
SWAP A  
leaves the Accumulator holding the value 5CH (01011100B).**Bytes:** 1**Cycles:** 1**Encoding:**

1	1	0	0	0	1	0	0
---	---	---	---	---	---	---	---

**Operation:** SWAP  
 $(A_{3-0}) \leftrightarrow (A_{7-4})$ 

---

**XCH A, <byte>****Function:** Exchange Accumulator with byte variable**Description:** XCH loads the Accumulator with the contents of the indicated variable, at the same time writing the original Accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.**Example:** R0 contains the address 20H. The Accumulator holds the value 3FH (00111111B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCH A, @R0

will leave RAM location 20H holding the values 3FH (00111111B) and 75H (01110101B) in the accumulator.

---

**XCH A, Rn****Bytes:** 1**Cycles:** 1**Encoding:**

1	1	0	0	1	r	r	r
---	---	---	---	---	---	---	---

**Operation:** XCH  
 $(A) \leftrightarrow (Rn)$ 

---

**XCH A, direct****Bytes:** 2**Cycles:** 1**Encoding:**

1	1	0	0	0	1	0	1
---	---	---	---	---	---	---	---

direct address
----------------

**Operation:** XCH  
 $(A) \leftrightarrow (\text{direct})$



---

**XCH A, @Ri****Bytes:** 1**Cycles:** 1**Encoding:**

1	1	0	0	0	1	1	i
---	---	---	---	---	---	---	---

**Operation:** XCH  
(A)  $\longleftrightarrow$  ((Ri))

---

**XCHD A, @Ri**

---

**Function:** Exchange Digit**Description:** XCHD exchanges the low-order nibble of the Accumulator (bits 3-0), generally representing a hexadecimal or BCD digit, with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.**Example:** R0 contains the address 20H. The Accumulator holds the value 36H (00110110B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCHD A, @R0

will leave RAM location 20H holding the value 76H (01110110B) and 35H (00110101B) in the accumulator.

**Bytes:** 1**Cycles:** 1**Encoding:**

1	1	0	1	0	1	1	i
---	---	---	---	---	---	---	---

**Operation:** XCHD  
(A<sub>3:0</sub>)  $\longleftrightarrow$  (Ri<sub>3:0</sub>)

---

**XRL <dest-byte>, <src-byte>**

---

**Function:** Logical Exclusive-OR for byte variables**Description:** XRL performs the bitwise logical Exclusive-OR operation between the indicated variables, storing the results in the destination. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

*(Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.)***Example:** If the Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

XRL A, R0

will leave the Accumulator holding the value 69H (01101001B).

When the destination is a directly addressed byte, this instruction can complement combination of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the Accumulator at run-time. The instruction,

XRL P1, #00110001B

will complement bits 5, 4 and 0 of output Port 1.

---

**XRL A, Rn****Bytes:** 1**Cycles:** 1**Encoding:**

0 1 1 0	1 r r r
---------	---------

**Operation:** XRL $(A) \leftarrow (A) \hat{\wedge} (Rn)$ **XRL A, direct****Bytes:** 2**Cycles:** 1**Encoding:**

0 1 1 0	0 1 0 1	direct address
---------	---------	----------------

**Operation:** XRL $(A) \leftarrow (A) \hat{\wedge} (\text{direct})$ **XRL A, @Ri****Bytes:** 1**Cycles:** 1**Encoding:**

0 1 1 0	0 1 1 i
---------	---------

**Operation:** XRL $(A) \leftarrow (A) \hat{\wedge} ((Ri))$ **XRL A, #data****Bytes:** 2**Cycles:** 1**Encoding:**

0 1 1 0	0 1 0 0	immediate data
---------	---------	----------------

**Operation:** XRL $(A) \leftarrow (A) \hat{\wedge} \#data$ **XRL direct, A****Bytes:** 2**Cycles:** 1**Encoding:**

0 1 1 0	0 0 1 0	direct address
---------	---------	----------------

**Operation:** XRL $(\text{direct}) \leftarrow (\text{direct}) \hat{\wedge} (A)$ **XRL direct, #data****Bytes:** 3**Cycles:** 2**Encoding:**

0 1 1 0	0 0 1 1	direct address	immediate data
---------	---------	----------------	----------------

**Operation:** XRL $(\text{direct}) \leftarrow (\text{direct}) \hat{\wedge} \# data$

---

## Chapter 6 Interrupt System

Microcontrollers are normally found in situations where the flow of a program will be subject to external events. These will come from hardware either outside the microcontroller or within the chip itself. Therefore an important feature of these devices is their ability to respond to signals known as interrupts which are received by the microcontroller.

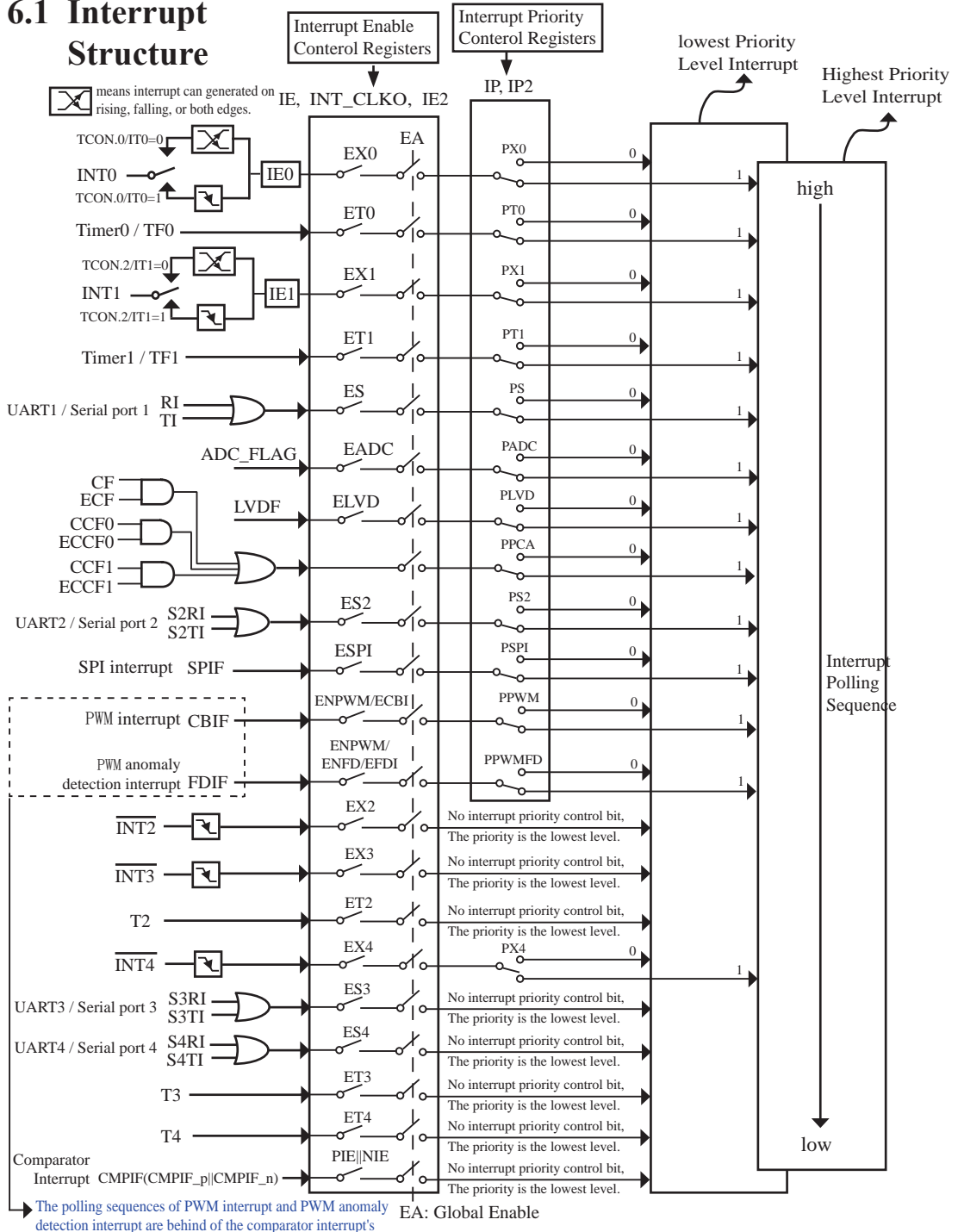
STC15W4K32S4 series MCU supports 21 interrupt sources. The 21 interrupt sources are external interrupt 0 (INT0), Timer 0 interrupt, external interrupt 1 (INT1), Timer 1 interrupt, serial port 1 (UART1) interrupt, ADC interrupt, low voltage detection (LVD) interrupt, CCP/PCA/PWM interrupt, serial port 2 (UART2) interrupt, SPI interrupt, external interrupt 2 (INT2), external interrupt 3 (INT3), Timer 2 interrupt, external interrupt 4 (INT4), serial port 3 (UART3) interrupt, serial port 4 (UART4) interrupt, Timer 3 interrupt, Timer 4 interrupt, comparator interrupt, PWM interrupt and PWM anomaly detection interrupt. Except external interrupt 2 (INT2), external interrupt 3 (INT3), Timer 2 interrupt, serial port 3 (UART3) interrupt, serial port 4 (UART4) interrupt, Timer 3 interrupt, Timer 4 interrupt and comparator interrupt are fixed with the lowest priority, the other interrupts all have two priority levels.

Each interrupt source has one or more associated interrupt-request flag(s) in SFRs. Associating with each interrupt vector, the interrupt sources can be individually enabled or disabled by setting or clearing a bit (interrupt enable control bit) in the SFRs IE, IE2, INT\_CLKO(AUXR2) and CCON. However, interrupts must first be globally enabled by setting the EA bit (IE.7) to logic 1 before the individual interrupt enables are recognized. Setting the EA bit to logic 0 disables all interrupt sources regardless of the individual interrupt-enable settings.

If interrupts are enabled for the source, an interrupt request is generated when the interrupt-request flag is set. As soon as execution of the current instruction is complete, the CPU generates an LCALL to a predetermined address to begin execution of an interrupt service routine (ISR). Each ISR must end with a RETI instruction, which returns program execution to the next instruction that would have been executed if the interrupt request had not occurred. If interrupts are not enabled, the interrupt-pending flag is ignored by the hardware and program execution continues as normal. (The interrupt-pending flag is set to logic 1 regardless of the interrupt's enable/disable state.)

Except external interrupt 2 (INT2), external interrupt 3 (INT3), Timer 2 interrupt, serial port 3 (UART3) interrupt, serial port 4 (UART4) interrupt, Timer 3 interrupt, Timer 4 interrupt and comparator interrupt, each interrupt source has one corresponding bit to represent its priority, which is located in SFR named IP and IP2 register. Higher-priority interrupt will be not interrupted by lower-priority interrupt request. If two interrupt requests of different priority levels are received simultaneously, the request of higher priority is serviced. If interrupt requests of the same priority level are received simultaneously, an internal polling sequence determine which request is serviced. The following table shows the internal polling sequence in the same priority level and the interrupt vector address.

# 6.1 Interrupt Structure



---

The External Interrupts INT0 and INT1 can be generated on rising, falling or both edges, depending on bits IT0/TCON.0 and IT1/TCON.2 in Register TCON. The flags that actually request these interrupts are bits IE0/TCON.1 and IE1/TCON.3 in register TCON, which would be automatically cleared when the external interrupts service routine is vectored to. The External Interrupts INT0 and INT1 can be generated on both rising and falling edge if the bits ITx = 0 (x = 0,1). The External Interrupts INT0 and INT1 only can be generated on falling edge if the bits ITx = 1 (x = 0,1). External interrupts also can be used to wake up MCU from Stop/Power-Down mode.

The request flags of Timer 0 and Timer1 Interrupts are bits TF0 and TF1, which are set by a rollover in their respective Timer/Counter registers in most cases. When a timer interrupt are generated, the responding flags are cleared by the on-chip hardware when the service routine is vectored to.

The External Interrupts  $\overline{\text{INT2}}$ ,  $\overline{\text{INT3}}$  and  $\overline{\text{INT4}}$  only can be falling-activated. The request flags of external interrupt 2~4 are invisible to users. When an external interrupt is generated, the interrupt request flag would be cleared by the hardware if the service routine is vectored to or EXn = 0 (n = 2,3,4).

The request flag of Timer 2 interrupt is invisible to users. When Timer 2 interrupt is generated, the interrupt request flag would be cleared by the hardware if the service routine is vectored to or ET2 = 0.

The request flags of Timer 3 interrupt and Timer 4 interrupt are invisible to users. When Timer 3 or Timer 4 interrupt is generated, the responding request flag would be cleared by the hardware if the service routine is vectored to or ET3 / ET4 = 0.

The Serial Port Interrupt is generated by the logical OR of RI and TI. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine will normally have to determine whether it was RI and TI that generated the interrupt, and the bit will have to be cleared by software.

The secondary serial port interrupt is generated by the logical OR of S2RI and S2TI. Neither of these flags is cleared by hardware when the service routine is vectored to. The service routine should poll S2RI and S2TI to determine which one to request service and it will be cleared by software.

The UART3 interrupt is generated by the logical OR of S3RI and S3TI. Neither of these flags is cleared by hardware when the service routine is vectored to. The service routine should poll S3RI and S3TI to determine which one to request service and it will be cleared by software.

The UART4 interrupt is generated by the logical OR of S4RI and S4TI. Neither of these flags is cleared by hardware when the service routine is vectored to. The service routine should poll S4RI and S4TI to determine which one to request service and it will be cleared by software.

The ADC interrupt is generated by the flag – ADC\_FLAG. It should be cleared by software.

The Low Voltage Detect interrupt is generated by the flag – LVDF(PCON.5) in PCON register. It should be cleared by software.

The CCP/PCA/PWM interrupt is generated by the logical OR of CF, CCF0 ~ CCF1. The service routine should poll CF and CCF0 ~ CCF1 to determine which one to request service and it will be cleared by software.

The SPI interrupt is generated by the flag SPIF. It can only be cleared by writing a “1” to SPIF bit in software.

All of the bits that generate interrupts can be set or cleared by software, with the same result as though it had been set or cleared by hardware. In other words, interrupts can be generated or pending interrupts can be canceled in software.

Interrupt Trigger Table

Interrupt Source	Trigger Behaviour
INT0 (External interrupt 0)	(IT0 = 1): falling edge (IT0 = 0): both rising and falling edges
Timer 0	Timer 0 overflow
INT1 (External interrupt 1)	(IT1 = 1): falling edge (IT1 = 0): both rising and falling edges
Timer1	Timer 1 overflow
UART1	finish sending or receiving of UART1
ADC	finish A/D converting
LVD	the operation voltage drops to less than LVD voltage.
UART2	finish sending or receiving of UART2
SPI	SPI data transmission is completed
$\overline{\text{INT2}}$ (External interrupt 2)	falling edge
$\overline{\text{INT3}}$ (External interrupt 3)	falling edge
Timer2	Timer 2 overflow
$\overline{\text{INT4}}$ (External interrupt 4)	falling edge
UART3	finish sending or receiving of UART3
UART4	finish sending or receiving of UART4
Timer3	Timer 3 overflow
Timer4	Timer 4 overflow
Comparator	The result after comparing by comparator have changed from low to high or from high to low

## 6.2 Interrupt Vector Address/Priority/Request Flag Table

Interrupt Sources, vector address, priority and polling sequence Table

Interrupt Sources	Interrupt Vector address	Priority within level	Interrupt Priority setting (IP, IP2)	Priority 0 (lowest)	Priority 1 (highest)	Interrupt Request	Interrupt Enable Control Bit
INT0 (External Interrupt 0)	0003H	0(highest)	PX0	0	1	IE0	EX0/EA
Timer 0	000BH	1	PT0	0	1	TF0	ET0/EA
INT1 (External Interrupt 1)	0013H	2	PX1	0	1	IE1	EX1/EA
Timer1	001BH	3	PT1	0	1	TF1	ET1/EA
Serial port 1(UART1)	0023B	4	PS	0	1	RI+TI	ES/EA
ADC	002BH	5	PADC	0	1	ADC_FLAG	EADC/EA
LVD	0033H	6	PLVD	0	1	LVDF	ELVD/EA
CCP/PCA	003BH	7	PPCA	0	1	CF+CCF0+CCF1 +CCF2	(ECF+ECCF0+ECCF1 +ECCF2)/EA
Serial port 2(UART2)	0043H	8	PS2	0	1	S2RI+S2TI	ES2/EA
SPI	004BH	9	PSPI	0	1	SPIF	ESPI/EA
$\overline{\text{INT}}2$ (External Interrupt 2)	0053H	10	0	0			EX2/EA
$\overline{\text{INT}}3$ (External Interrupt 3)	005BH	11	0	0			EX3/EA
Timer 2	0063H	12	0	0			ET2/EA
-	006BH	13					
System Reserved	0073H	14					
System Reserved	007BH	15					
$\overline{\text{INT}}4$ (External Interrupt 4)	0083H	16	PX4	0	1		EX4/EA
Serial port 3(UART3) interrupt	008BH	17	0	0		S3RI+S3TI	ES3/EA
Serial port 4(UART4) interrupt	0093H	18	0	0		S4RI+S4TI	ES4/EA
Timer 3 interrupt	009BH	19	0	0			ET3/EA
Timer 4 interrupt	00A3H	20	0	0			ET4/EA
Comparator interrupt	00ABH	21(lowest)	0	0		CMPIF_p	PIE/EA (Positive-edge)
						CMPIF_n	NIE/EA (Negative-edge)
PWM interrupt	00B3H	22	PPWM	0	1	CBIF	ENPWM/ECBI/EA
						C2IF	ENPWM / EPWM2I / EC2T2SI    EC2T1SI / EA
						C3IF	ENPWM / EPWM3I / EC3T2SI    EC3T1SI / EA
						C4IF	ENPWM / EPWM4I / EC4T2SI    EC4T1SI / EA
						C5IF	ENPWM / EPWM5I / EC5T2SI    EC5T1SI / EA
						C6IF	ENPWM / EPWM6I / EC6T2SI    EC6T1SI / EA
C7IF	ENPWM / EPWM7I / EC7T2SI    EC7T1SI / EA						
PWM anomaly detection interrupt	00BBH	23(lowest)	PPWMFD	0	1	FDIF	ENPWM/ENFD/EFDI/ EA

---

## 6.3 How to Declare Interrupt Function in Keil C

In C language program. the interrupt polling sequence number is equal to interrupt number, for example,

```
void    Int0_Routine(void)        interrupt 0;
void    Timer0_Routine(void)     interrupt 1;
void    Int1_Routine(void)       interrupt 2;
void    Timer1_Routine(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    Timer2_Routine(void)     interrupt 12;
void    Int4_Routine(void)       interrupt 16;
void    S3_Routine(void)         interrupt 17;
void    S4_Routine(void)         interrupt 18;
void    Timer3_Routine(void)     interrupt 19;
void    Timer4_Routine(void)     interrupt 20;
void    Comparator_Routine(void) interrupt 21;
void    PWM_Routine(void)        interrupt 22;
void    PWMFD_Routine(void)      interrupt 23;
```



## 6.4 Interrupt Registers

Symbol	Description	Address	Bit Address and Symbol								Value after Power-on or Reset
			MSB				LSB				
IE	Interrupt Enable	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000 0000B
IE2	Interrupt Enable 2	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000 0000B
INT_CLKO AUXR2	External Interrupt enable and Clock Output register	8FH	-	EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO	x000 0000B
IP	Interrupt Priority Low	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000 0000B
IP2	2rd Interrupt Priority register	B5H	-	-	-	PX4	PPWMFD	PPWM	PSPI	PS2	xx00 0000B
TCON	Timer Control register	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000 0000B
SCON	Serial Control	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000 0000B
S2CON	Serial 2/ UART2 Control	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0000 0000B
S3CON	UART3 Control Register	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S4CON	UART4 Control Register	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
T4T3M	T4 and T3 Control and Mode register	D1H	T4R	T4_C $\bar{T}$	T4x12	T4CLKO	T3R	T3_C $\bar{T}$	T3x12	T3CLKO	0000 0000B
PCON	Power Control register	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011 0000B
ADC_CONTR	ADC control register	BCH	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHS0	0000 0000B
SPSTAT	SPI Status register	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx xxxxB
CCON	PCA Control Register	D8H	CF	CR	-	-	-	-	CCF1	CCF0	00xx x000B
CMOD	PCA Mode Register	D9H	CIDL	-	-	-	CPS2	CPS1	CPS0	ECF	0xxx 0000B
CCAPM0	PCA Module 0 Mode Register	DAH	-	ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0	x000 0000B
CCAPM1	PCA Module 1 Mode Register	DBH	-	ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1	x000 0000B
AUXR	Auxiliary register	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C $\bar{T}$	T2x12	EXTRAM	S1ST2	0000 0001B
CMPCR1	Compartor control Register 1	E6H	CMPEM	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000 0000B
PWMCR	PWM Control Register	F5H	ENPWM	ECBI	ENC70	ENC60	ENC50	ENC40	ENC30	ENC20	0000 0000B
PWMIF	PWM Interrupt Flag Register	F6H	-	CBIF	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	x000 0000B
PWMFDCR	PWM F_ception Detection Control Register	F7H	-	-	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	xx00 0000B

SFRs of STC15W4K32S4 series MCU (continued)

Symbol	Description	Address	Bit Address and Symbol								Value after Power-on or Reset
			B7	B6	B5	B4	B3	B2	B1	B0	
PWM2CR	PWM2 Control Register	FF04H	-	-	-	-	PWM2_PS	EPWM2I	EC2T2SI	EC2T1SI	xxxx,0000B
PWM3CR	PWM3 Control Register	FF14H	-	-	-	-	PWM3_PS	EPWM3I	EC3T2SI	EC3T1SI	xxxx,0000B
PWM4CR	PWM4 Control Register	FF24H	-	-	-	-	PWM4_PS	EPWM4I	EC4T2SI	EC4T1SI	xxxx,0000B
PWM5CR	PWM5 Control Register	FF34H	-	-	-	-	PWM5_PS	EPWM5I	EC5T2SI	EC5T1SI	xxxx,0000B
PWM6CR	PWM6 Control Register	FF44H	-	-	-	-	PWM6_PS	EPWM6I	EC6T2SI	EC6T1SI	xxxx,0000B
PWM7CR	PWM7 Control Register	FF54H	-	-	-	-	PWM7_PS	EPWM7I	EC7T2SI	EC7T1SI	xxxx,0000B

### 1. Interrupt Enable control Registers IE, IE2 and INT\_CLKO (AUXR2)

#### IE: Interrupt Enable Register (Bit-addressable)

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

Enable Bit = 1 enables the interrupt .

Enable Bit = 0 disables it .

EA (IE.7): disables all interrupts.

If EA = 0, no interrupt would be acknowledged.

If EA = 1, each interrupt source would be individually enabled or disabled by setting or clearing its enable bit.

ELVD (IE.6): Low voltage detection interrupt enable bit.

If ELVD = 0, Low voltage detection interrupt would be disabled.

If ELVD = 1, Low voltage detection interrupt would be enabled.

EADC (IE.5): ADC interrupt enable bit.

If EADC = 0, ADC interrupt would be disabled.

If EADC = 1, ADC interrupt would be enabled.

ES (IE.4): Serial Port 1 (UART1) interrupt enable bit.

If ES = 0, UART1 interrupt would be disabled.

If ES = 1, UART1 interrupt would be enabled.

ET1 (IE.3): Timer 1 interrupt enable bit.

If ET1 = 0, Timer 1 interrupt would be disabled.

If ET1 = 1, Timer 1 interrupt would be enabled.

EX1 (IE.2): External interrupt 1 enable bit.

If EX1 = 0, external interrupt 1 would be disabled.

If EX1 = 1, external interrupt 1 would be enabled.

ET0 (IE.1): Timer 0 interrupt enable bit.

If ET0 = 0, Timer 0 interrupt would be disabled.

If ET0 = 1, Timer 0 interrupt would be enabled.

EX0 (IE.0): External interrupt 0 enable bit.

If EX0 = 0, external interrupt 0 would be disabled.

If EX0 = 1, external interrupt 0 would be enabled.

---

**IE2: Interrupt Enable 2 Register** (Non bit-addressable)

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

ET4 (IE.6): Timer 4 interrupt enable bit.  
If ET4 = 0, Timer 4 interrupt would be disabled.  
If ET4 = 1, Timer 4 interrupt would be enabled.

ET3 (IE.5): Timer 3 interrupt enable bit.  
If ET3 = 0, Timer 3 interrupt would be disabled.  
If ET3 = 1, Timer 3 interrupt would be enabled.

ES4 (IE2.4): Serial Port 4 (UART4) interrupt enable bit.  
If ES4 = 0, UART4 interrupt would be disabled.  
If ES4 = 1, UART4 interrupt would be enabled.

ES3 (IE2.3): Serial Port 3 (UART3) interrupt enable bit.  
If ES3 = 0, UART3 interrupt would be disabled.  
If ES3 = 1, UART3 interrupt would be enabled.

ET2 (IE2.2) Timer 2 interrupt enable bit.  
If ET2 = 0, Timer 2 interrupt would be disabled.  
If ET2 = 1, Timer 2 interrupt would be enabled.

ESPI (IE2.1): SPI interrupt enable bit.  
If ESPI = 0, SPI interrupt would be disabled.  
If ESPI = 1, SPI interrupt would be enabled.

ES2 (IE2.0): Serial Port 2 (UART2) interrupt enable bit.  
If ES2 = 0, UART2 interrupt would be disabled.  
If ES2 = 1, UART2 interrupt would be enabled.

**INT\_CLKO (AUXR2) : External Interrupt Enable and Clock Output register**

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name	-	EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO

EX4 (IE.6): Enable bit of External Interrupt 4 ( $\overline{\text{INT4}}$ )  
If EX4 = 0, External Interrupt 4 ( $\overline{\text{INT4}}$ ) would be disabled.  
If EX4 = 1, External Interrupt 4 ( $\overline{\text{INT4}}$ ) would be enabled.

EX3 (IE.5): Enable bit of External Interrupt 3 ( $\overline{\text{INT3}}$ )  
If EX3 = 0, External Interrupt 3 ( $\overline{\text{INT3}}$ ) would be disabled.  
If EX3 = 1, External Interrupt 3 ( $\overline{\text{INT3}}$ ) would be enabled.

EX2 (IE.4): Enable bit of External Interrupt 2 ( $\overline{\text{INT2}}$ )  
If EX2 = 0, External Interrupt 2 ( $\overline{\text{INT2}}$ ) would be disabled.  
If EX2 = 1, External Interrupt 2 ( $\overline{\text{INT2}}$ ) would be enabled.

T2CLKO, T1CLKO, T0CLKO bits are not introduced here because they are not related with interrupts.

## 2. Interrupt Priority control Registers IP and IP2

Except external interrupt 2( $\overline{\text{INT2}}$ ), external interrupt 3( $\overline{\text{INT3}}$ ), Timer 2 interrupt, external interrupt 4( $\overline{\text{INT4}}$ ), serial port 3(UART3) interrupt, serial port 4(UART4) interrupt, Timer 3 interrupt and Timer 4 interrupt, each interrupt source of STC15 all can be individually programmed to one of two priority levels by setting or clearing the bit in Special Function Registers IP or IP2. A low-priority interrupt can itself be interrupted by a high-priority interrupt, but not by another low-priority interrupt. A high-priority interrupt can't be interrupted by any other interrupt source.

### IP: Interrupt Priority Register (Bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	name	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0

PPCA: PCA interrupt priority control bit.

if PPCA=0, PCA interrupt is assigned lowest priority (priority 0).

if PPCA=1, PCA interrupt is assigned highest priority (priority 1).

PLVD: Low voltage detection interrupt priority control bit.

if PLVD=0, Low voltage detection interrupt is assigned lowest priority(priority 0).

if PLVD=1, Low voltage detection interrupt is assigned highest priority(priority 1).

PADC: ADC interrupt priority control bit.

if PADC=0, ADC interrupt is assigned lowest priority (priority 0).

if PADC=1, ADC interrupt is assigned highest priority (priority 1).

PS : Serial Port 1 (UART1) interrupt priority control bit.

if PS=0, UART1 interrupt is assigned lowest priority (priority 0).

if PS=1, UART1 interrupt is assigned highest priority (priority 1).

PT1 : Timer 1 interrupt priority control bit.

if PT1=0, Timer 1 interrupt is assigned lowest priority (priority 0).

if PT1=1, Timer 1 interrupt is assigned highest priority (priority 1).

PX1 : External interrupt 1 priority control bit.

if PX1=0, External interrupt 1 is assigned lowest priority (priority 0).

if PX1=1, External interrupt 1 is assigned highest priority (priority 1).

PT0 : Timer 0 interrupt priority control bit.

if PT0=0, Timer 0 interrupt is assigned lowest priority (priority 0).

if PT0=1, Timer 0 interrupt is assigned highest priority (priority 1).

PX0 : External interrupt 0 priority control bit.

if PX0=0, External interrupt 0 is assigned lowest priority (priority 0).

if PX0=1, External interrupt 0 is assigned highest priority (priority 1).

### IP2: Interrupt Priority Register (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP2	B5H	name	-	-	-	PX4	PPWMFD	PPWM	PSPI	PS2

PX4 : External interrupt 4 priority control bit.

if PX4=0, External interrupt 4 is assigned lowest priority (priority 0).

if PX4=1, External interrupt 4 is assigned highest priority (priority 1).

---

**IP2: Interrupt Priority Register** (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP2	B5H	name	-	-	-	PX4	PPWMFD	PPWM	PSPI	PS2

PSPI : SPI interrupt priority control bit.

if PSPI=0, SPI interrupt is assigned lowest priority (priority 0).

if PSPI=1, SPI interrupt is assigned highest priority (priority 1).

PS2 : Serial Port 2 (UART2) interrupt priority control bit.

if PS2=0, UART2 interrupt is assigned lowest priority (priority 0).

if PS2=1, UART2 interrupt is assigned highest priority (priority 1).

**3. TCON register: Timer/Counter Control Register** (Bit-Addressable)

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

TF1: Timer/Counter 1 Overflow Flag. Set by hardware on Timer/Counter 1 overflow. The flag can be cleared by software but is automatically cleared by hardware when processor vectors to the Timer 1 interrupt routine.

If TF1 = 0, No Timer 1 overflow detected.

If TF1 = 1, Timer 1 has overflowed.

TR1: Timer/Counter 1 Run Control bit. Set/cleared by software to turn Timer/Counter on/off.

If TR1 = 0, Timer 1 disabled.

If TR1 = 1, Timer 1 enabled.

TF0: Timer/Counter 0 Overflow Flag. Set by hardware on Timer/Counter 0 overflow. The flag can be cleared by software but is automatically cleared by hardware when processor vectors to the Timer 0 interrupt routine.

If TF0 = 0, No Timer 0 overflow detected.

If TF0 = 1, Timer 0 has overflowed.

TR0: Timer/Counter 0 Run Control bit. Set/cleared by software to turn Timer/Counter on/off.

If TR0 = 0, Timer 0 disabled.

If TR0 = 1, Timer 0 enabled.

IE1: External Interrupt 1 request flag. Set by hardware when external interrupt rising or falling edge defined by IT1 is detected. The flag can be cleared by software but is automatically cleared when the external interrupt 1 service routine has been processed.

IT1 : External Interrupt 1 Type Select bit. Set/cleared by software to specify rising / falling edges triggered external interrupt 1.

If IT1 = 0, INT1 is both rising and falling edges triggered.

If IT1 = 1, INT1 is only falling edge triggered.

IE0 : External Interrupt 0 request flag. Set by hardware when external interrupt rising or falling edge defined by IT0 is detected. The flag can be cleared by software but is automatically cleared when the external interrupt 1 service routine has been processed.

IT0 : External Interrupt 0 Type Select bit. Set/cleared by software to specify rising / falling edges triggered external interrupt 0.

If IT0 = 0, INT0 is both rising and falling edges triggered.

If IT0 = 1, INT0 is only falling edge triggered.

---

#### 4. SCON register: Serial Port 1 (UART1) Control Register (Bit-Addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	name	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

TI : Transmit interrupt flag. Set by hardware when a byte of data has been transmitted by UART1 (after the 8th bit in 8-bit UART Mode, or at the beginning of the STOP bit in 9-bit UART Mode). When the UART1 interrupt is enabled, setting this bit causes the CPU to vector to the UART1 interrupt service routine. This bit must be cleared manually by software.

RI : Receive interrupt flag. Set to '1' by hardware when a byte of data has been received by UART1 (set at the STOP bit sam-pling time). When the UART1 interrupt is enabled, setting this bit to '1' causes the CPU to vector to the UART1 interrupt service routine. This bit must be cleared manually by software.

The other bits of SCON register without relation to the UART1 interrupt is not be introduced here.

#### 5. S2CON register: Serial Port 2 (UART2) Control Register (No bit-Addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S2CON	9AH	name	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI

S2TI : Transmit interrupt flag. Set by hardware when a byte of data has been transmitted by UART2 (after the 8th bit in 8-bit UART Mode, or at the beginning of the STOP bit in 9-bit UART Mode). When the UART2 interrupt is enabled, setting this bit causes the CPU to vector to the UART2 interrupt service routine. This bit must be cleared manually by software.

S2RI : Receive interrupt flag. Set to '1' by hardware when a byte of data has been received by UART2 (set at the STOP bit sam-pling time). When the UART2 interrupt is enabled, setting this bit to '1' causes the CPU to vector to the UART2 interrupt service routine. This bit must be cleared manually by software.

The other bits of S2CON register without relation to the UART2 interrupt is not be introduced here.

#### 6. S3CON register: Serial Port 3 (UART3) Control Register (No bit-Addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S3CON	ACH	name	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI

S3TI : Transmit interrupt flag. Set by hardware when a byte of data has been transmitted by UART3 (after the 8th bit in 8-bit UART Mode, or at the beginning of the STOP bit in 9-bit UART Mode). When the UART3 interrupt is enabled, setting this bit causes the CPU to vector to the UART3 interrupt service routine. This bit must be cleared manually by software.

S3RI : Receive interrupt flag. Set to '1' by hardware when a byte of data has been received by UART3 (set at the STOP bit sam-pling time). When the UART3 interrupt is enabled, setting this bit to '1' causes the CPU to vector to the UART3 interrupt service routine. This bit must be cleared manually by software.

The other bits of S3CON register without relation to the UART3 interrupt is not be introduced here.

## 7. S4CON register: Serial Port 4 (UART4) Control Register (No bit-Addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S4CON	84H	name	S4SM0	S4ST3	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

**S4TI** : Transmit interrupt flag. Set by hardware when a byte of data has been transmitted by UART4 (after the 8th bit in 8-bit UART Mode, or at the beginning of the STOP bit in 9-bit UART Mode). When the UART4 interrupt is enabled, setting this bit causes the CPU to vector to the UART4 interrupt service routine. This bit must be cleared manually by software.

**S4RI** : Receive interrupt flag. Set to '1' by hardware when a byte of data has been received by UART4 (set at the STOP bit sam-pling time). When the UART4 interrupt is enabled, setting this bit to '1' causes the CPU to vector to the UART4 interrupt service routine. This bit must be cleared manually by software.

The other bits of S4CON register without relation to the UART4 interrupt is not be introduced here.

## 8. Register related with LVD interrupt: Power Control register PCON (Non bit-Addressable)

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

SMOD: double Baud rate control bit.

0 : Disable double Baud rate of the UART.

1 : Enable double Baud rate of the UART in mode 1,2,or 3.

SMOD0: Frame Error select.

0 : SCON.7 is SM0 function.

1 : SCON.7 is FE function. Note that FE will be set after a frame error regardless of the state of SMOD0.

**LVDF** : Pin Low-Voltage Flag. Once low voltage condition is detected (VCC power is lower than LVD voltage), it is set by hardware (and should be cleared by software).

**POF** : Power-On flag. It is set by power-off-on action and can only cleared by software.

**GF1** : General-purposed flag 1

**GF0** : General-purposed flag 0

**PD** : Power-Down bit.

**IDL** : Idle mode bit.

IE: Interrupt Enable Rsgister (Bit-addressable)

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

**EA** : disables all interrupts.

If EA = 0,no interrupt will be acknowledged.

If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

**ELVD**: Low volatge detection interrupt enable bit.

If ELVD = 0, Low voltage detection interrupt would be disabled.

If ELVD = 1, Low voltage detection interrupt would be enabled.

---

## 9. ADC\_CONTR: AD Control register (Non bit-Addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
ADC_CONTR	BCH	name	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHS0

ADC\_POWER : When clear, shut down the power of ADC bolck. When set, turn on the power of ADC block.

ADC\_FLAG : ADC interrupt flag.It will be set by the device after the device has finished a conversion, and should be cleared by the user's software.

ADC\_STRAT : ADC start bit, which enable ADC conversion.It will automatically cleared by the device after the device has finished the conversion

IE: Interrupt Enable Rsgister (Bit-addressable)

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

EA : disables all interrupts.

If EA = 0,no interrupt will be acknowledged.

If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

EADC: ADC interrupt enable bit.

If EADC = 0, ADC interrupt would be disabled.

If EADC = 1, ADC interrupt would be enabled.

## 10. Register related with PCA interrupt

CCON: PCA Control Register (bit-Addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CCON	D8H	name	CF	CR	-	-	-	-	CCF1	CCF0

CF : PCA Counter Overflow flag. Set by hardware when the counter rolls over. CF flags an interrupt if bit ECF in CMOD is set. CF may be set by either hardware or software but can only be cleared by software.

CR : PCA Counter Run control bit. Set by software to turn the PCA counter on. Must be cleared by software to turn the PCA counter off.

CCF1: PCA Module 1 interrupt flag. Set by hardware when a match or capture occurs. Must be cleared by software.

CCF0: PCA Module 0 interrupt flag. Set by hardware when a match or capture occurs. Must be cleared by software.



## 10. Register related with PCA interrupt (continued)

### CMOD: PCA Mode Register (Non bit-Addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	D9H	name	CIDL	-	-	-	CPS2	CPS1	CPS0	ECF

CIDL : Counter Idle control bit.

CIDL=0 programs the PCA Counter to continue functioning during idle mode.

CIDL=1 programs it to be gated off during idle.

CPS2, CPS1, CPS0 : PCA Counter Pulse Select bits, as shown below.

CPS2	CPS1	CPS0	PCA Counter Pulse Select bits.
0	0	0	0, System clock, SYSclk/12
0	0	1	1, System clock, SYSclk/2
0	1	0	2, Timer 0 overflow pulse. the frequency of PWM output can be adjusted by changing Timer 0 overflow.
0	1	1	3, External clock at ECI/P1.2 pin ( the maximum frequency = SYSclk/2)
1	0	0	4, System clock, SYSclk
1	0	1	5, System clock/4, SYSclk/4
1	1	0	6, System clock/6, SYSclk/6
1	1	1	7, System clock/8, SYSclk/8

ECF : PCA Enable Counter Overflow interrupt. ECF=1 enables CF bit in CCON to generate an interrupt.

### CCAPMn register (Non bit-Addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CCAPM0	DAH	name	-	ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0
CCAPM1	DBH	name	-	ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1

ECOMn : Enable Comparator. ECOMn=1 enables the comparator function.

CAPPn : Capture Positive, CAPPn=1 enables positive edge capture.

CAPNn : Capture Negative, CAPNn=1 enables negative edge capture.

MATn : Match. When MATn=1, a match of the PCA counter with this module's compare/capture register causes the CCFn bit in CCON to be set.

TOGn : Toggle. When TOGn=1, a match of the PCA counter with this module's compare/capture register causes the CEXn pin to toggle.

PWMn : Pulse Width Modulation. PWMn=1 enables the CEXn pin to be used as a pulse width modulated output.

ECCFn : Enable CCF interrupt. Enables compare/capture flag CCFn in the CCON register to generate

---

## 11. Register related with SPI interrupt

### SPSTAT: SPI Status Control Register (Non bit-Addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	CDH	name	SPIF	WCOL	-	-	-	-	-	-

SPIF : SPI transfer completion flag. When a serial transfer finishes, the SPIF bit is set and an interrupt is generated if both the ESPI(IE.6) bit and the EA(IE.7) bit are set. If SS is an input and is driven low when SPI is in master mode with SSIG = 0, SPIF will also be set to signal the “mode change”. The SPIF is cleared in software by “writing 1 to this bit”.

WCOL: SPI write collision flag. The WCOL bit is set if the SPI data register, SPDAT, is written during a data transfer. The WCOL flag is cleared in software by “writing 1 to this bit”

### IE: Interrupt Enable Register (Bit-addressable)

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

EA : disables all interrupts.

If EA = 0, no interrupt will be acknowledged.

If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

### IE2: Interrupt Enable 2 Register (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	name	-	-	-	-	-	-	ESPI	ES2

ESPI: SPI interrupt enable bit.

If ESPI = 0, SPI interrupt would be disabled.

If ESPI = 1, SPI interrupt would be enabled.

## 6.5 Interrupt Priorities

Except external interrupt 2( $\overline{INT2}$ ), external interrupt 3( $\overline{INT3}$ ), Timer 2 interrupt, serial port 3(UART3) interrupt, serial port 4(UART4) interrupt, Timer 3 interrupt, Timer 4 interrupt and comparator interrupt, each interrupt source of STC15W4K32S4 series MCU can be individually programmed to one of two priority levels by setting or clearing the bit in Special Function Registers IP or IP2. A low-priority interrupt can itself be interrupted by a high-priority interrupt, but not by another low-priority interrupt. A high-priority interrupt can't be interrupted by any other interrupt source.

If two requests of different priority levels are received simultaneously, the request of higher priority level is serviced. If requests of the same priority level are received simultaneously, an internal polling sequence determines which request is serviced. Thus within each priority level there is a second priority structure determined by the polling sequence, as follows:

	Interrupt Source	Priority Within Level
0.	INT0	(highest)
1.	Timer 0	↓
2.	INT1	
3.	Timer 1	
4.	UART1	
5.	ADC interrupt	
6.	LVD	
7.	PCA	
8.	UART2	
9.	SPI	
10.	INT2	
11.	INT3	
12.	Timer 2	
13.		
14.		
15.	_____	
16.	INT4	
17.	UART3	
18.	UART4	
19.	Timer 3	
20.	Timer 4	
21.	Comparator	
22.	PWM	
23.	PWMFD	

Note that the “priority within level” structure is only used to resolve *simultaneous requests of the same priority level*.

In C language program, the interrupt polling sequence number is equal to interrupt number, for example,

```

void Int0_Routine(void)      interrupt 0;
void Timer0_Routine(void)   interrupt 1;
void Int1_Routine(void)     interrupt 2;
void Timer1_Routine(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 Timer2_Routine(void)   interrupt 12;
void Int4_Routine(void)     interrupt 16;
void S3_Routine(void)       interrupt 17;
void S4_Routine(void)       interrupt 18;
void Timer3_Routine(void)   interrupt 19;
void Timer4_Routine(void)   interrupt 20;
void Comparator_Routine(void) interrupt 21;
void PWM_Routine(void)      interrupt 22;
void PWMFD_Routine(void)    interrupt 23;

```

---

## 6.6 Interrupt Handling

The CPU usually has several lines connected to it which can receive interrupts in the form of voltage changes. When an interrupt is received, the following actions are carried out by the MCU:

1. The current instruction in the main program is allowed to complete execution.
2. The address of the next instruction is pushed to the stack.
3. Control jumps to the start of a subprogram, known as an Interrupt Service Routine (ISR).
4. The ISR code is executed.
5. When the instruction RETI (Return from Interrupt) is encountered in the ISR, the return address is popped from the stack into the PC.
6. Control is returned to the original location in the main program.

An Interrupt Service Routine (ISR) (sometimes called interrupt handler) is similar in form to a subroutine. However, the great difference between the two is that the subroutine is called by an instruction within the program, while the ISR is activated by a hardware voltage change into the CPU.

External interrupt pins and other interrupt sources are sampled at the rising edge of each instruction *OPcode fetch cycle*. The samples are polled during the next instruction *OPcode fetch cycle*. If one of the flags was in a set condition of the first cycle, the second cycle of polling cycles will find it and the interrupt system will generate an hardware LCALL to the appropriate service routine as long as it is not blocked by any of the following conditions.

### Block conditions :

- An interrupt of equal or higher priority level is already in progress.
- The current cycle (polling cycle) is not the final cycle in the execution of the instruction in progress.
- The instruction in progress is RETI or any write to the IE, IE2, IP and IP2 registers.
- The ISP/IAP activity is in progress.

Any of these four conditions will block the generation of the hardware LCALL to the interrupt service routine. Condition 2 ensures that the instruction in progress will be completed before vectoring into any service routine. Condition 3 ensures that if the instruction in progress is RETI or any access to IE, IE2, IP and IP2, then at least one or more instruction will be executed before any interrupt is vectored to.

The polling cycle is repeated with the last clock cycle of each instruction cycle. Note that if an interrupt flag is active but not being responded to for one of the above conditions, if the flag is not still active when the blocking condition is removed, the denied interrupt will not be serviced. In other words, the fact that the interrupt flag was once active but not being responded to for one of the above conditions, if the flag is not still active when the blocking condition is removed, the denied interrupt will not be serviced. The interrupt flag was once active but not serviced is not kept in memory. Every polling cycle is new.

---

Note that if an interrupt of higher priority level goes active prior to the rising edge of the third machine cycle, then in accordance with the above rules it will be vectored to during fifth and sixth machine cycle, without any instruction of the lower priority routine having been executed.

Thus the processor acknowledges an interrupt request by executing a hardware-generated LCALL to the appropriate servicing routine. In some cases it also clears the flag that generated the interrupt, and in other cases it doesn't. It never clears the Serial Port flags. This has to be done in the user's software. It clears an external interrupt flag (IE0 or IE1) only if it was transition-activated. The hardware-generated LCALL pushes the contents of the Program Counter onto the stack (but it does not save the PSW) and reloads the PC with an address that depends on the source of the interrupt being vectored to, as shown below.

<b>Source</b>	<b>Vector Address</b>
External Interrupt 0	0003H
Timer 0	000BH
External Interrupt 1	0013H
Timer 1	001BH
S1(UART1)	0023H
ADC interrupt	002BH
LVD	0033H
PCA	003BH
S2(UART2)	0043H
SPI	004BH
External Interrupt 2	0053H
External Interrupt 3	005BH
Timer 2	0063H
/	006BH
/	0073H
/	007BH
External Interrupt 4	0083H
S3(UART3)	008BH
S4(UART4)	0093H
Timer 3	009BH
Timer 4	00A3H
Comparator	00ABH
PWM	00B3H
PWMD	00BBH

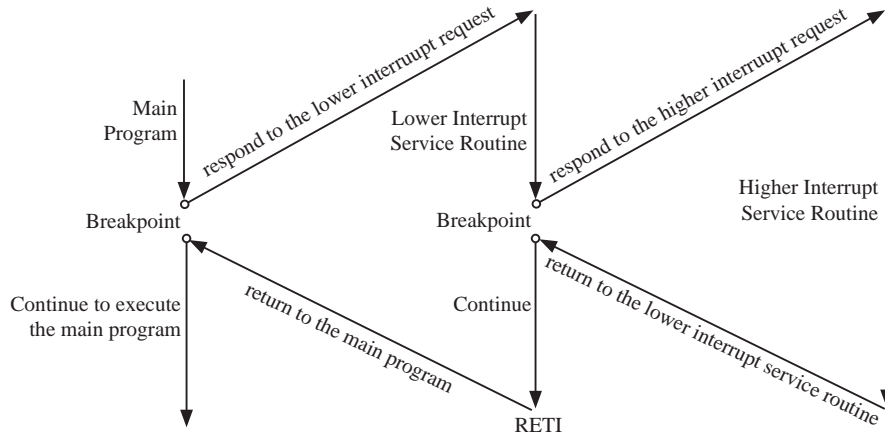
Execution proceeds from that location until the RETI instruction is encountered. The RETI instruction informs the processor that this interrupt routine is no longer in progress, then pops the top two bytes from the stack and reloads the Program Counter. Execution of the interrupted program continues from where it left off.

Note that a simple RET instruction would also have returned execution to the interrupted program, but it would have left the interrupt control system thinking an interrupt was still in progress.

---

## 6.7 Interrupt Nesting

The interrupt requests of a higher priority can preempt the interrupt requests and service routine of a lower priority. Only the interrupt service routine of the higher priority has been accomplished, should the service of routine of the lower priority be continue to execute. This is called interrupt nesting. The schematic diagram of interrupt nesting is shown below.



## 6.8 External Interrupts

The External Interrupts INT0 and INT1 can be generated on rising, falling or both edges, depending on bits IT0/TCON.0 and IT1/TCON.2 in Register TCON. The flags that actually request these interrupts are bits IE0/TCON.1 and IE1/TCON.3 in register TCON, which would be automatically cleared when the external interrupts service routine is vectored to. The External Interrupts INT0 and INT1 can be generated on both rising and falling edge if the bits ITx = 0 (x = 0,1). The External Interrupts INT0 and INT1 only can be generated on falling edge if the bits ITx = 1 (x = 0,1). External interrupts also can be used to wake up MCU from Stop/Power-Down mode.

The External Interrupts  $\overline{\text{INT2}}$ ,  $\overline{\text{INT3}}$  and  $\overline{\text{INT4}}$  only can be falling-activated. The request flags of external interrupt 2~4 are invisible to users. When an external interrupt is generated, the interrupt request flag would be cleared by the hardware if the service routine is vectored to or EXn = 0 (n = 2,3,4).

If the external interrupt is falling or rising edges-activated, the external source has to hold the request active until the requested interrupt is actually generated. Then it has to deactivate the request before the interrupt service routine is completed, or else another interrupt will be generated. Since the external interrupt pins are sampled once each machine cycle, an input high or low should hold for at least one system clocks to ensure sampling.

---

## 6.9 Interrupt Demo Program (C and ASM)

### 6.9.1 External Interrupt 0 (INT0) Demo Program

#### 6.9.1.1 External Interrupt INT0 (rising + falling edge) Demo Program (C and ASM)

##### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of INT0 -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

//-----
bit    FLAG;                                //1: interrupt can be generated on rising edge
                                           //0: interrupt can be generated on falling edge

sbit   P10    =    P1^0;

//-----
//External Interrupt Service Routine
void exint0() interrupt 0                    //INT0, interrupt 0 (location at 0003H)
{
    P10    =    !P10;
    FLAG   =    INT0;                       //save the state of INT0 pin, INT0=0(falling); INT0=1(rising)
}

//-----
void main()
{
    INT0    =    1;
    IT0     =    0;                          //Setting INT0 interrupt type
                                           //(1:only falling 0:both falling and rising edges)
    EX0     =    1;                          //enable INT0 interrupt
    EA      =    1;

    while (1);
}
```

---

## 2.Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of INT0 -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

FLAG BIT 20H.0 //1: interrupt can be generated on rising edge
//0: interrupt can be generated on falling edge

//-----

ORG 0000H
LJMP MAIN

ORG 0003H //INT0, interrupt 0 (location at 0003H)
LJMP EXINT0

//-----

MAIN:
ORG 0100H
MOV SP, #3FH

CLR IT0 //Setting INT0 interrupt type
//1:only falling 0:both falling and rising edges

SETB EX0 //enable INT0 interrupt
SETB EA
SJMP $

//-----
//External Interrupt Service Routine

EXINT0:
CPL P1.0
PUSH PSW
MOV C, INT0 //read the status of INT0 pin
MOV FLAG, C //save, INT0=0(falling edge); INT0=1(rising edge)
POP PSW
RETI

;-----
END
```



---

## 6.9.1.2 External Interrupt INT0 (falling edge) Demo Program (C and ASM)

### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of INT0 -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

//-----

sbit    P10    =    P1^0;

//-----
//External interrupt0 service routine
void exint0() interrupt 0                //INT0, interrupt 0 (location at 0003H)
{
    P10    =    !P10;
}

//-----

void main()
{
    INT0    =    1;
    IT0     =    1;                //Setting INT0 interrupt type
                                        //(1:only falling 0:both falling and rising edges)
    EX0     =    1;                //enable INT0 interrupt
    EA      =    1;

    while (1);
}
```

---

## 2.Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of INT0 -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

    ORG    0000H
    LJMP   MAIN

    ORG    0003H           //INT0, interrupt 0 (location at 0003H)
    LJMP   EXINT0

//-----

    ORG    0100H
MAIN:
    MOV    SP,    #3FH

    SETB   IT0           //Setting INT0 interrupt type
                        //(1:only falling 0:both falling and rising edges)
    SETB   EX0           //enable INT0 interrupt
    SETB   EA
    SJMP   $

//-----
//External Interrupt Service Routine

EXINT0:
    CPL    P1.0
    RETI

;-----

    END
```

---

## 6.9.2 External Interrupt 1(INT1) Demo Program

### 6.9.2.1 External Interrupt INT1 (rising + falling edge) Demo Program (C and ASM)

#### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of INT1 -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

//-----

bit    FLAG;                //1: interrupt can be generated on rising edge
                                //0: interrupt can be generated on falling edge

sbit   P10    =    P1^0;

//-----
//External Interrupt Service Routine
void exint1() interrupt 2    //INT1, interrupt 0 (location at 0013H)
{
    P10    =    !P10;
    FLAG   =    INT1;      //Save the status of INT1 pin, INT1=0(falling); INT1=1(rising)
}

//-----

void main()
{
    INT1    =    1;
    IT1     =    0;        //Setting INT1 interrupt type
                                //(1:only falling 0:both falling and rising edges)
    EX1     =    1;        //enable INT1 interrupt
    EA      =    1;

    while (1);
}
```

---

## 2.Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of INT1 -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

FLAG  BIT    20H.0                //1: interrupt can be generated on rising edge
                                     //0: interrupt can be generated on falling edge

//-----
        ORG    0000H
        LJMP   MAIN

        ORG    0013H                //INT1, interrupt 0 (location at 0013H)
        LJMP   EXINT1

//-----
MAIN:   ORG    0100H
        MOV    SP,    #3FH

        CLR    IT1                //Setting INT1 interrupt type
                                     //(1:only falling 0:both falling and rising edges)
        SETB   EX1                //enable INT1 interrupt
        SETB   EA
        SJMP   $

//-----
//External Interrupt Service Routine

EXINT1:
        CPL    P1.0
        PUSH   PSW
        MOV    C,    INT1        //read the status of INT1 pin
        MOV    FLAG, C          //save, INT1=0(falling); INTO=1(rising)
        POP    PSW
        RETI

;-----
        END
```

---

## 6.9.2.2 External Interrupt INT1 (falling edge) Demo Program (C and ASM)

### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of INT1 -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

//-----

sbit    P10    =    P1^0;

//-----
//External Interrupt Service Routine
void exint1() interrupt 2                //INT1, interrupt 0 (location at 0013H)
{
    P10    =    !P10;
}

//-----

void main()
{
    INT1    =    1;
    IT1     =    1;                //Setting INT1 interrupt type
                                        //(1:only falling 0:both falling and rising edges)
    EX1     =    1;                //Enable INT1 interrupt
    EA      =    1;

    while (1);
}
```

---

## 2.Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of INT1 -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz
    ORG    0000H
    LJMP   MAIN

    ORG    0013H           //INT1, interrupt 0 (location at 0013H)
    LJMP   EXINT1

//-----

    ORG    0100H
MAIN:
    MOV    SP,    #3FH

    SETB   IT1           //Setting INT1 interrupt type
                        //(1:only falling 0:both falling and rising edges)
    SETB   EX1           //enable INT1 interrupt
    SETB   EA
    SJMP   $

//-----
//External Interrupt Service Routine

EXINT1:
    CPL    P1.0
    RETI

;-----

    END
```

---

## 6.9.3 External Interrupt 2 (INT2) (falling) Demo Program (C and ASM)

### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of (INT2) (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

//-----

sfr    INT_CLKO    =    0x8f;           //External interrupt control register
sbit   P10        =    P1^0;

//-----
//External Interrupt Service Routine
void exint2() interrupt 10             //INT2, interrupt 2 (location at 0053H)
{
    P10    =    !P10;

//    INT_CLKO    &=    0xEF;
//    INT_CLKO    |=    0x10;
}

void main()
{
    INT_CLKO    |=    0x10;           //(EX2 = 1), enable INT2 interrupt
    EA    =    1;

    while (1);
}
```

---

## 2.Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of (INT2) (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

INT_CLKO DATA 08FH //External interrupt control register
//-----

    ORG    0000H
    LJMP   MAIN

    ORG    0053H //INT2, interrupt 2 (location at 0053H)
    LJMP   EXINT2
//-----

    ORG    0100H
MAIN:   MOV    SP,    #3FH

    ORL    INT_CLKO,    #10H // (EX2 = 1), enable INT2 interrupt

    SETB   EA

    SJMP   $

//-----
//External Interrupt Service Routine

EXINT2:
    CPL    P1.0

//    ANL    INT_CLKO,    #0EFH

//    ORL    INT_CLKO,    #10H

    RETI

;-----
    END
```



---

## 6.9.4 External Interrupt 3 ( $\overline{\text{INT3}}$ )(falling) Demo Program (C and ASM)

### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of ( $\overline{\text{INT3}}$ ) (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

//-----

sfr    INT_CLKO    =    0x8f;           //External interrupt control register
sbit   P10        =    P1^0;

//-----
//External Interrupt Service Routine
void exint3() interrupt 11             //INT3, interrupt 3 (location at 005BH)
{
    P10    =    !P10;

//    INT_CLKO    &=    0xDF;

//    INT_CLKO    |=    0x20;
}

void main()
{
    INT_CLKO    |=    0x20;           //(EX3 = 1), enable INT3 interrupt
    EA    =    1;

    while (1);
}
```

---

## 2.Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of (INT3) (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

INT_CLKO DATA 08FH //External Interrupt control
//-----

    ORG    0000H
    LJMP  MAIN

    ORG    005BH //INT3, interrupt 3 (location at 005BH)
    LJMP  EXINT3
//-----

    ORG    0100H
MAIN:
    MOV   SP,    #3FH

    ORL   INT_CLKO,    #20H // (EX3 = 1), enable INT3 interrupt

    SETB EA
    SJMP $

//-----
//External Interrupt Service Routine

EXINT3:
    CPL   P1.0

//    ANL   INT_CLKO,    #0DFH
//    ORL   INT_CLKO,    #20H

    RETI
;-----

    END
```

---

## 6.9.5 External Interrupt 4 (INT4) (falling) Demo Program (C and ASM)

### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of (INT4) (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

//-----

sfr    INT_CLKO    =    0x8f;           //External interrupt control register
sbit   P10        =    P1^0;

//-----
//External Interrupt Service Routine
void exint4() interrupt 16             //INT4, interrupt 4 (location at 0083H)
{
    P10    =    !P10;

//    INT_CLKO    &=    0xBF;

//    INT_CLKO    |=    0x40;
}

void main()
{
    INT_CLKO    |=    0x40;           //(EX4 = 1), enable INT4 interrupt
    EA    =    1;

    while (1);
}
```

---

## 2.Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of (INT4) (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

INT_CLKO DATA 08FH //External interrupt control register

//-----
ORG 0000H
LJMP MAIN

ORG 0083H //INT4, interrupt 4 (location at 0083H)
LJMP EXINT4
//-----

ORG 0100H
MAIN:
MOV SP, #3FH

ORL INT_CLKO, #40H //(EX4 = 1), enable INT4 interrupt

SETB EA

SJMP $

//-----
//External Interrupt Service Routine
EXINT4:
CPL P1.0

// ANL INT_CLKO, #0BFH
// ORL INT_CLKO, #40H

RETI
;-----
END
```

---

## 6.9.6 Demo Program using T0 to expand External Interrupt (Falling)

### —— T0 as Counter (C and ASM)

#### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using T0 to extend external interrupt (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

//-----

sfr    AUXR  =    0x8e;           //Auxiliary register
sbit   P10   =    P1^0;

//-----
//Timer 0 Interrupt Service Routine
void t0int() interrupt 1         //Timer 0 interrupt, location at 000BH
{
    P10    =    !P10;
}

void main()
{
    AUXR   =    0x80;           //T0 in 1T mode
    TMOD   =    0x04;         //T0 as external counter
                                //and T0 in 16-bit auto-reload mode
    TH0    =    TL0    =    0xff; //Set the initial value of T0
    TR0    =    1;           //start up T0
    ETO    =    1;           //Enable T0 interrupt

    EA     =    1;

    while (1);
}
```

---

## 2.Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using T0 to extend external interrupt (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

AUXR    DATA 08EH                //Auxiliary register
//-----

        ORG    0000H
        LJMP   MAIN

        ORG    000BH                //Timer 0 interrupt, location at 000BH
        LJMP   TOINT
//-----

        ORG    0100H
MAIN:
        MOV    SP,    #3FH

        MOV    AUXR, #80H           //T0 in 1T mode
        MOV    TMOD, #04H          //T0 as external counter
                                        //and T0 in 16-bit auto-reload mode
        MOV    A,     #0FFH        //Set the initial value of T0
        MOV    TL0,   A
        MOV    TH0,   A
        SETB  TR0                //start up T0
        SETB  ET0                //Enable T0 interrupt

        SETB  EA

        SJMP  $

//-----
//Timer 0 interrupt service routine

TOINT:
        CPL    P1.0
        RETI
;-----
        END
```

---

## 6.9.7 Demo Program using T1 to expand External Interrupt (Falling) —— T1 as Counter (C and ASM)

### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using T1 to extend external interrupt (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

//-----

sfr    AUXR  =    0x8e;           //Auxiliary register
sbit   P10   =    P1^0;

//-----
//Timer 1 Interrupt Service Routine
void t1int() interrupt 3         //Timer 1 interrupt, location at 001BH
{
    P10    =    !P10;
}

void main()
{
    AUXR   =    0x40;           //T1 in 1T mode
    TMOD   =    0x40;           //T1 as external counter
                                   //and T1 in 16-bit auto-reload mode
    TH1 = TL1 =    0xff;       //Set the initial value of T1
    TR1    =    1;             //start up T1
    ET1    =    1;             //Enable T1 interrupt

    EA     =    1;

    while (1);
}
```

---

## 2.Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using T1 to extend external interrupt (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

AUXR    DATA 08EH                //Auxiliary register
//-----

        ORG    0000H
        LJMP   MAIN

        ORG    001BH                //Timer 1 interrupt, location at 001BH
        LJMP   T1INT
//-----

MAIN:   ORG    0100H
        MOV    SP,    #3FH

        MOV    AUXR, #40H           //T1 in 1T mode
        MOV    TMOD, #40H           //T1 as external counter
                                           //and T1 in 16-bit auto-reload mode
        MOV    A,     #0FFH         //Set the initial value of T1
        MOV    TL1,   A
        MOV    TH1,   A
        SETB   TR1                //start up T1
        SETB   ET1                //Enable T1 interrupt

        SETB   EA

        SJMP   $

//-----
//Timer 1 Interrupt Service Routine
T1INT:  CPL    P1.0
RETI
;-----
        END
```



---

## 6.9.8 Demo Program using T2 to expand External Interrupt (Falling) —— T2 as Counter (C and ASM)

### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using T2 to extend external interrupt (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

//-----
sfr    IE2    =    0xaf;           //Interrupt enable register 2
sfr    AUXR   =    0x8e;           //Auxiliary register
sfr    T2H    =    0xD6;
sfr    T2L    =    0xD7;

sbit   P10    =    P1^0;

//-----
//Timer 2 Interrupt Service Routine
void t2int() interrupt 12           //Timer 2 interrupt, location at 0063H
{
    P10    =    !P10;

//    IE2    &=    ~0x04;

//    IE2    |=    0x04;
}

void main()
{
    AUXR   |=    0x04;           //T2 in 1T mode
```

---

```

AUXR      |=      0x08;          //T2_C/T=1, T2(P3.1) as Clock Source
T2H  = T2L  =      0xff;        //Set the initial value of T2
AUXR      |=      0x10;        //start up T2

IE2   |=      0x04;           //Enable T2 interrupt

EA   =      1;

while (1);
}

```

## 2.Assembler Listing

```

/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using T2 to extend external interrupt (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

IE2   DATA  0AFH           //Interrupt enable register 2
AUXR  DATA  08EH           //Auxiliary register
T2H   DATA  0D6H
T2L   DATA  0D7H

//-----

        ORG    0000H
        LJMP   MAIN

        ORG    0063H           //Timer 2 interrupt, location at 0063H
        LJMP   T2INT

//-----

        ORG    0100H

```

---

MAIN:

```
MOV    SP,    #3FH

ORL    AUXR,  #04H           //T2 in 1T mode
ORL    AUXR,  #08H           //T2_C/T=1, T2(P3.1) as Clock Source

MOV    A,     #0FFH         //Set the initial value of T2
MOV    T2L,   A
MOV    T2H,   A

ORL    AUXR,  #10H         //start up T2

ORL    IE2,   #04H         //Enable T2 interrupt

SETB   EA

SJMP   $
```

```
//-----
//Timer 2 Interrupt Service Routine
```

T2INT:

```
CPL    P1.0

//     ANL    IE2,  #0FBH

//     ORL    IE2,  #04H

RETI
```

```
;-----
```

END

---

## 6.9.9 Demo Program using CCP/PCA to expand External Interrupt

### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using CCP/PCA to extend external interrupt (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

//This demo program take CCP/PCA module 0 for example. the use of CCP/PCA module 1 and CCP/PCA module
//2 are same as CCP/PCA module 0

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

#define FOSC 18432000L

typedef unsigned char BYTE;
typedef unsigned int WORD;
typedef unsigned long DWORD;

sfr P_SW1 = 0xA2; //Peripheral Function Switch register 1

#define CCP_S0 0x10 //P_SW1.4
#define CCP_S1 0x20 //P_SW1.5

sfr CCON = 0xD8; //PCA Control Register
sbit CCF0 = CCON^0; //the interrupt request flag of PCA module 0
sbit CCF1 = CCON^1; //the interrupt request flag of PCA module 1
sbit CR = CCON^6; //the run bit of PCA timer
sbit CF = CCON^7; //the overflow flag of PCA timer
sfr CMOD = 0xD9; //PCA Mode register
sfr CL = 0xE9;
sfr CH = 0xF9;
sfr CCAPM0 = 0xDA;
sfr CCAP0L = 0xEA;
sfr CCAP0H = 0xFA;
sfr CCAPM1 = 0xDB;
sfr CCAP1L = 0xEB;
sfr CCAP1H = 0xFB;
sfr CCAPM2 = 0xDC;
sfr CCAP2L = 0xEC;
```

---

```

sfr    CCAP2H      =    0xFC;

sfr    PCAPWM0    =    0xF2;
sfr    PCAPWM1    =    0xF3;
sfr    PCA_  PWM2 =    0xF4;

sbit   PCA_LED    =    P1^0;           //PCA test LED

void PCA_isr() interrupt 7 using 1
{
    CCF0 = 0;                          //clear the interrupt request flag
    PCA_LED = !PCA_LED;
}

void main()
{
    ACC    =    P_SW1;
    ACC    &=    ~(CCP_S0 | CCP_S1); //CCP_S0=0 CCP_S1=0
    P_SW1  =    ACC;                   //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)

//    ACC    =    P_SW1;
//    ACC    &=    ~(CCP_S0 | CCP_S1); //CCP_S0=1 CCP_S1=0
//    ACC    |=    CCP_S0;           //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//    P_SW1  =    ACC;
//
//    ACC    =    P_SW1;
//    ACC    &=    ~(CCP_S0 | CCP_S1); //CCP_S0=0 CCP_S1=1
//    ACC    |=    CCP_S1;           //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//    P_SW1  =    ACC;

    CCON   =    0;                     //Initialize the PCA control register
                                           //disable PCA timer
                                           //clear CF bit
                                           //clear the interrupt request flag

    CL     =    0;                     //reset PCA timer
    CH     =    0;
    CMOD   =    0x00;

    CCAPM0 =    0x11;                 //PCA module 0 can be activated on falling edge
//    CCAPM0 =    0x21;                 //PCA module 0 can be activated on rising edge
//    CCAPM0 =    0x31;                 //PCA module 0 can be activated
                                           //both on falling and rising edge

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

    while (1);
}

```

---

---

## 2.Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using CCP/PCA to extend external interrupt (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

//This demo program take CCP/PCA module 0 for example. the use of CCP/PCA module 1 and CCP/PCA module
//2 are same as CCP/PCA module 0

P_SW1 EQU 0A2H //Peripheral Function Switch register 1
CCP_S0 EQU 10H //P_SW1.4
CCP_S1 EQU 20H //P_SW1.5

CCON EQU 0D8H ;PCA Control Register
CCF0 BIT CCON.0 ;the interrupt request flag of PCA module 0
CCF1 BIT CCON.1 ;the interrupt request flag of PCA module 1
CR BIT CCON.6 ;the run bit of PCA timer
CF BIT CCON.7 ;the overflow flag of PCA timer
CMOD EQU 0D9H ;PCA Mode register
CL EQU 0E9H
CH EQU 0F9H
CCAPM0 EQU 0DAH
CCAP0L EQU 0EAH
CCAP0H EQU 0FAH
CCAPM1 EQU 0DBH
CCAP1L EQU 0EBH
CCAP1H EQU 0FBH
CCAPM2 EQU 0DCH
CCAP2L EQU 0ECH
CCAP2H EQU 0FCH
PCA_PWM0 EQU 0F2H
PCA_PWM1 EQU 0F3H
PCA_PWM2 EQU 0F4H

PCA_LED BIT P1.0 ;PCA test LED
;-----
ORG 0000H
LJMP MAIN

ORG 003BH
```

```

PCA_ISR:
    PUSH    PSW
    PUSH    ACC
CKECK_CCF0:
    JNB     CCF0,    PCA_ISR_EXIT
    CLR     CCF0                ;clear the interrupt request flag
    CPL     PCA_LED
PCA_ISR_EXIT:
    POP     ACC
    POP     PSW
    RETI

;-----
    ORG     0100H
MAIN:
    MOV     SP,     #5FH

    MOV     A,     P_SW1
    ANL     A,     #0CFH        //CCP_S0=0 CCP_S1=0
    MOV     P_SW1, A            //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, P3.7/CCP2)

//     MOV     A,     P_SW1
//     ANL     A,     #0CFH        //CCP_S0=1 CCP_S1=0
//     ORL     A,     #CCP_S0      //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2, P3.7/CCP2_2)
//     MOV     P_SW1, A
//
//     MOV     A,     P_SW1
//     ANL     A,     #0CFH        //CCP_S0=0 CCP_S1=1
//     ORL     A,     #CCP_S1      //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3, P2.7/CCP2_3)
//     MOV     P_SW1, A

    MOV     CCON,  #0                ;Initialize the PCA control register
                                        ;disable PCA timer
                                        ;clear CF bit
                                        ;clear the interrupt request flag

    CLR     A                        ;
    MOV     CL,   A                ;reset PCA timer
    MOV     CH,   A                ;
    MOV     CMOD, #00H

    MOV     CCAPM0, #11H           ;PCA module 0 capture the falling edge of CCP0(P1.3) pin
;     MOV     CCAPM0, #21H           ;PCA module 0 capture the rising edge of CCP0(P1.3) pin
;     MOV     CCAPM0, #31H           ;PCA module 0 capture falling as well as
                                        ;rising edge of CCP0(P1.3) pin

;-----
    SETB    CR                    ;run PCA timer
    SETB    EA

    SJMP    $

;-----
    END

```

---

## Chapter 7 Timer/Counter

There are five 16-bit Timer/Counter: T0, T1, T2, T3 and T4, which all can be as Timer or Counter. For T0 and T1 which are compatible with conventional 8051, the “Timer” or “Counter” function is selected by control bits  $C/\overline{T}$  in the Special Function Register TMOD. For T2, the “Timer” or “Counter” function is selected by control bits  $T2\_C/\overline{T}$  in the Special Function Register AUXR. For T3, the “Timer” or “Counter” function is selected by control bits  $T3\_C/\overline{T}$  in the Special Function Register T4T3M. For T4, the “Timer” or “Counter” function is selected by control bits  $T4\_C/\overline{T}$  in the Special Function Register T4T3M. Timer counts internal system clock, and Counter counts external pulses from pins T0 or T1 or T2 or T3 or T4.

For T0, T1 and T2, the timer register (TH and TL) is incremented every 12 system clocks or every system clock depending on AUXR.7(T0x12) and AUXR.6(T1x12) and AUXR.2(T2x12) bits in the “Timer” function. In the default state, it is fully the same as the conventional 8051. In the x12 mode, the count rate equals to the system clock. For T3 and T4, the timer register (TH and TL) is incremented every 12 system clocks or every system clock depending on T4T3M.1(T3x12) and T4T3M.5(T4x12) bits in the “Timer” function.

In the “Counter” function, the register (TH and TL) is incremented in response to a 1-to-0 transition at its corresponding external input pin, T0 or T1 or T2 or T3 or T4. In this function, the external input is sampled once at the positive edge of every clock cycle. When the samples show a high in one cycle and a low in the next cycle, the count is incremented. The new count value appears in the register during at the end of the cycle following the one in which the transition was detected. Since it takes 2 machine cycles (24 system clocks) to recognize a 1-to-0 transition, the maximum count rate is 1/24 of the system clock. There are no restrictions on the duty cycle of the external input signal, but to ensure that a given level is sampled at least once before it changes, it should be held for at least one full machine cycle.

In addition to the “Timer” or “Counter” selection, Timer/Counter 0 has four operating modes which are selected by bit-pairs (M1, M0) in TMOD. These four modes are mode 0 (16-bit auto-reload timer/counter), mode 1 (16-bit timer/counter), mode 2 (8-bit auto-reload timer/counter) and mode 3 (16-bit auto-reload timer/counter whose interrupt can not be disabled). And for Timer/Counter 1, Modes 0, 1, and 2 are the same as Timer/Counter 0. Mode 3 is different. the mode 3 of Timer/Counter 1 is invalid. The four operating modes are described in the following text. For T2, T3 and T4, they only have one mode : 16-bit auto-reload timer/counter. Besides as Timer/Counter, T2, T3 and T4 also can be as the baud-rate generator and programmable clock output.



## 7.1 Special Function Registers about Timer/Counter

Symbol	Description	Address	Bit Address and Symbol								Value after Power-on or Reset
			MSB				LSB				
TCON	Timer Control	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000 0000B
TMOD	Timer Mode	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000 0000B
TL0	Timer Low 0	8AH									0000 0000B
TL1	Timer Low 1	8BH									0000 0000B
TH0	Timer High 0	8CH									0000 0000B
TH1	Timer High 1	8DH									0000 0000B
IE	Interrupt Enable	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000 0000B
IP	Interrupt Enable 2	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000 0000B
T2H	The high 8-bit of Timer 2 register	D6H									0000 0000B
T2L	The low 8-bit of Timer 2 register	D7H									0000 0000B
AUXR	Auxiliary register	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000 0001B
INT_CLKO AUXR2	External Interrupt enable and Clock Output register	8FH	-	EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO	x000 0000B
T4T3M	T4 and T3 Control and Mode register	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000 0000B
T4H	The high 8-bit of Timer 4 register	D2H									0000 0000B
T4L	The low 8-bit of Timer 4 register	D3H									0000 0000B
T3H	The high 8-bit of Timer 3 register	D4H									0000 0000B
T3L	The low 8-bit of Timer 3 register	D5H									0000 0000B
IE2	Interrupt Enable register	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000 0000B

---

## 1. TCON register: Timer/Counter Control Register (Bit-Addressable)

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

**TF1:** Timer/Counter 1 Overflow Flag. Set by hardware on Timer/Counter 1 overflow. The flag can be cleared by software but is automatically cleared by hardware when processor vectors to the Timer 1 interrupt routine.  
If TF1 = 0, No Timer 1 overflow detected.  
If TF1 = 1, Timer 1 has overflowed.

**TR1:** Timer/Counter 1 Run Control bit. Set/cleared by software to turn Timer/Counter on/off.  
If TR1 = 0, Timer 1 disabled.  
If TR1 = 1, Timer 1 enabled.

**TF0:** Timer/Counter 0 Overflow Flag. Set by hardware on Timer/Counter 0 overflow. The flag can be cleared by software but is automatically cleared by hardware when processor vectors to the Timer 0 interrupt routine.  
If TF0 = 0, No Timer 0 overflow detected.  
If TF0 = 1, Timer 0 has overflowed.

**TR0:** Timer/Counter 0 Run Control bit. Set/cleared by software to turn Timer/Counter on/off.  
If TR0 = 0, Timer 0 disabled.  
If TR0 = 1, Timer 0 enabled.

**IE1:** External Interrupt 1 request flag. Set by hardware when external interrupt rising or falling edge defined by IT1 is detected. The flag can be cleared by software but is automatically cleared when the external interrupt 1 service routine has been processed.

**IT1 :** External Interrupt 1 Type Select bit. Set/cleared by software to specify rising / falling edges triggered external interrupt 1.  
If IT1 = 0, INT1 is both rising and falling edges triggered.  
If IT1 = 1, INT1 is only falling edge triggered.

**IE0 :** External Interrupt 0 request flag. Set by hardware when external interrupt rising or falling edge defined by IT0 is detected. The flag can be cleared by software but is automatically cleared when the external interrupt 1 service routine has been processed.

**IT0 :** External Interrupt 0 Type Select bit. Set/cleared by software to specify rising / falling edges triggered external interrupt 0.  
If IT0 = 0, INT0 is both rising and falling edges triggered.  
If IT0 = 1, INT0 is only falling edge triggered.

## 2. TMOD register: Timer/Counter Mode Register

TMOD address: 89H (Non bit-addressable)



GATR / TMOD.7 : Timer/Counter Gate Control.

- If GATE / TMOD.7 = 0, Timer/Counter 1 enabled when TR1 is set irrespective INT1 of logic level;
- If GATE / TMOD.7 = 1, Timer/Counter 1 enabled only when TR1 is set AND INT1 pin is high.

C/T / TMOD.6 : Timer/Counter 1 Select bit.

- If C/T / TMOD.6 = 0, Timer/Counter 1 is set for Timer operation (input from internal system clock);
- If C/T / TMOD.6 = 1, Timer/Counter 1 is set for Counter operation (input from external T1 pin).

M1 / TMOD.5 ~ M0 / TMOD.4 : Timer 1 Mode Select bits.

M1	M0	Operating Mode
0	0	Mode 0: 16-bit auto-reload Timer/Counter for T1
0	1	Mode 1: 16-bit Timer/Counter. TH1 and TL1 are cascaded; there is no prescaler.
1	0	Mode 2: 8-bit auto-reload Timer/Counter. TH1 holds a value which is to be reloaded into TL1 each time it overflows.
1	1	Timer/Counter 1 is stopped

GATR / TMOD.3 : Timer/Counter Gate Control.

- If GATE / TMOD.3 = 0, Timer/Counter 0 enabled when TR0 is set irrespective of INT0 logic level;
- If GATE / TMOD.3 = 1, Timer/Counter 0 enabled only when TR0 is set AND INT0 pin is high.

C/T / TMOD.2 : Timer/Counter 0 Select bit.

- If C/T / TMOD.2 = 0, Timer/Counter 0 is set for Timer operation (input from internal system clock);
- If C/T / TMOD.2 = 1, Timer/Counter 0 is set for Counter operation (input from external T0 pin).

M1 / TMOD.1 ~ M0 / TMOD.0 : Timer 0 Mode Select bits.

M1	M0	Operating Mode
0	0	Mode 0: 16-bit auto-reload Timer/Counter for T0
0	1	Mode 1: 16-bit Timer/Counter. TH0 and TL0 are cascaded; there is no prescaler.
1	0	Mode 2: 8-bit auto-reload Timer/Counter. TH0 holds a value which is to be reloaded into TL0 each time it overflows.
1	1	Mode 3: 16-bit auto-reload Timer/Counter whose interrupt can not be disabled for T0.

---

### 3. AUXR: Auxiliary register (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C $\bar{T}$	T2x12	EXTRAM	S1ST2

B7 - T0x12 : Timer 0 clock source bit.

- 0 : The clock source of Timer 0 is SYSclk/12. It will compatible to the traditional 8051 MCU
- 1 : The clock source of Timer 0 is SYSclk/1. It will drive the T0 faster than a traditional 8051 MCU

B6 - T1x12 : Timer 1 clock source bit.

- 0 : The clock source of Timer 1 is SYSclk/12. It will compatible to the traditional 8051 MCU
- 1 : The clock source of Timer 1 is SYSclk/1. It will drive the T0 faster than a traditional 8051 MCU

If T1 is used as the baud-rate generator of UART1, T1x12 will decide whether UART1 is 1T or 12T.

B5 - UART\_M0x6 : Baud rate select bit of UART1 while it is working under 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.

B4 - T2R Timer 2 Run control bit

- 0 : not run Timer 2;
- 1 : run Timer 2.

B3 - T2\_C $\bar{T}$ : Counter or timer 2 selector

- 0 : as Timer (namely count on internal system clock)
- 1 : as Counter (namely count on the external pulse input from T2/P3.1)

B2 - T2x12 : Timer 2 clock source bit.

- 0 : The clock source of Timer 2 is SYSclk/12.
- 1 : The clock source of Timer 2 is SYSclk/1.

If T2 is used as the baud-rate generator of UART1 or UART2, T1x12 will decide whether UART1 or UART2 is 1T or 12T.

B1 - EXTRAM : Internal / external RAM access control bit.

- 0 : On-chip auxiliary RAM is enabled.
- 1 : On-chip auxiliary RAM is always disabled.

B0 - S1ST2 : the control bit that UART1 select Timer 2 as its baud-rate generator.

- 0 : Select Timer 1 as the baud-rate generator of UART1
- 1 : Select Timer 2 as the baud-rate generator of UART1. Timer 1 is released to use in other functions.

#### 4. T0, T1 and T2 Clock Output and External Interrupt Enable register : INT\_CLKO (AUXR2)

The output clock frequency of T0CLKO is controlled by Timer 0. The output clock frequency of T1CLKO is controlled by Timer 1. When they are used as programmable clock output, Timer 0 and Timer 1 must work in mode 0 (16-bit auto-reload timer/counter) or mode 2 (8-bit auto-reload timer/counter) and don't enable their interrupt to avoid CPU entering interrupt repeatedly unless special circumstances. The output clock frequency of T2CLKO is controlled by Timer 2 which only has one mode (16-bit auto-reload timer/counter). Similarly, when T2 is used as programmable clock output, it also don't enable their interrupt to avoid CPU entering interrupt repeatedly unless special circumstances.

INT\_CLKO (AUXR2) : Clock Output and External Interrupt Enable register (Non bit-Addressable)

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name	-	EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO

B0 - T0CLKO : Whether is P3.5/T1 configured for Timer 0(T0) programmable clock output T0CLKO or not.

1, P3.5/T1 is configured for Timer0 programmable clock output T0CLKO, the clock output frequency =  $T0 \text{ overflow} / 2$

If Timer/Counter 0 in mode 0 (16 bit auto-reloadable mode),

and if  $C/\bar{T} = 0$ , namely Timer/Counter 0 count on the internal system clock,

When T0 in 1T mode (AUXR.7/T0x12=1), the output frequency =  $(SYSclk) / (65536 - [RL\_TH0, RL\_TL0]) / 2$

When T0 in 12T mode (AUXR.7/T0x12=0), the output frequency =  $(SYSclk) / 12 / (65536 - [RL\_TH0, RL\_TL0]) / 2$

and if  $C/\bar{T} = 1$ , namely Timer/Counter 0 count on the external pulse input from P3.4/T0,

the output frequency =  $(T0\_Pin\_CLK) / (65536 - [RL\_TH0, RL\_TL0]) / 2$

If Timer/Counter 0 in mode 2 (8 bit auto-reloadable mode),

and if  $C/\bar{T} = 0$ , namely Timer/Counter 0 count on the internal system clock,

When T0 in 1T mode (AUXR.7/T0x12=1), the output frequency =  $(SYSclk) / (256 - TH0) / 2$

When T0 in 12T mode (AUXR.7/T0x12=0), the output frequency =  $(SYSclk) / 12 / (256 - TH0) / 2$

and if  $C/\bar{T} = 1$ , namely Timer/Counter 0 count on the external pulse input from P3.4/T0,

the output frequency =  $(T0\_Pin\_CLK) / (256 - TH0) / 2$

0, P3.5/T1 is not configure for Timer 0 programmable clock output T0CLKO

B1 - T1CLKO : Whether is P3.4/T0 configured for Timer 1(T1) programmable clock output T1CLKO or not.

1, P3.4/T0 is configured for Timer1 programmable clock output T1CLKO, the clock output frequency =  $T1 \text{ overflow} / 2$

If Timer/Counter 1 in mode 1 (16 bit auto-reloadable mode),

and if  $C/\bar{T} = 0$ , namely Timer/Counter 1 count on the internal system clock,

When T1 in 1T mode (AUXR.6/T1x12=1), the output frequency =  $(SYSclk) / (65536 - [RL\_TH1, RL\_TL1]) / 2$

When T1 in 12T mode (AUXR.6/T1x12=0), the output frequency =  $(SYSclk) / 12 / (65536 - [RL\_TH1, RL\_TL1]) / 2$

and if  $C/\bar{T} = 1$ , namely Timer/Counter 1 count on the external pulse input from P3.5/T1,

the output frequency =  $(T1\_Pin\_CLK) / (65536 - [RL\_TH1, RL\_TL1]) / 2$

If Timer/Counter 1 in mode 2 (8 bit auto-reloadable mode),

and if  $C/\bar{T} = 0$ , namely Timer/Counter 1 count on the internal system clock,

When T1 in 1T mode (AUXR.6/T1x12=1), the output frequency =  $(SYSclk) / (256 - TH1) / 2$

When T1 in 12T mode (AUXR.6/T1x12=0), the output frequency =  $(SYSclk) / 12 / (256 - TH1) / 2$

and if  $C/\bar{T} = 1$ , namely Timer/Counter 1 count on the external pulse input from P3.5/T1,

the output frequency =  $(T1\_Pin\_CLK) / (256 - TH1) / 2$

0, P3.4/T0 is not configure for Timer 1 programmable clock output T1CLKO

B2 - T2CLKO : Whether is P3.0 configured for Timer 2(T2) programmable clock output T2CLKO or not.

1, P3.0 is configured for Timer2 programmable clock output T2CLKO, the clock output frequency =  $T2 \text{ overflow} / 2$

If  $T2\_C/\bar{T} = 0$ , namely Timer/Counter 2 count on the internal system clock,

When T2 in 1T mode ( $AUXR.2/T2x12=1$ ), the output frequency =  $(SYSclk)/(65536-[RL\_TH2, RL\_TL2])/2$

When T2 in 12T mode ( $AUXR.2/T2x12=0$ ), the output frequency =  $(SYSclk) / 12 / (65536-[RL\_TH2, RL\_TL2])/2$

If  $T2\_C/\bar{T} = 1$ , namely Timer/Counter 2 count on the external pulse input from P3.1/T2,

the output frequency =  $(T2\_Pin\_CLK) / (65536-[RL\_TH2, RL\_TL2])/2$

0, P3.0 is not configure for Timer 2 programmable clock output T2CLKO

## 5. Register related to T0 and T1 interrupt: IE and IP

IE: Interrupt Enable Register (Bit-addressable)

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

EA : disables all interrupts.

If EA = 0, no interrupt will be acknowledged.

If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

ET1: Timer 1 interrupt enable bit.

If ET1 = 0, Timer 1 interrupt would be disabled.

If ET1 = 1, Timer 1 interrupt would be enabled.

ET0: Timer 0 interrupt enable bit.

If ET0 = 0, Timer 0 interrupt would be disabled.

If ET0 = 1, Timer 0 interrupt would be enabled.

IP: Interrupt Priority Register (Bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	name	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0

PT1 : Timer 1 interrupt priority control bit.

if PT1=0, Timer 1 interrupt is assigned lowest priority (priority 0).

if PT1=1, Timer 1 interrupt is assigned highest priority (priority 1).

PT0 : Timer 0 interrupt priority control bit.

if PT0=0, Timer 0 interrupt is assigned lowest priority (priority 0).

if PT0=1, Timer 0 interrupt is assigned highest priority (priority 1).

## 6. T4T3M : Timer 4 and Timer 3 Mode register (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/ $\bar{T}$	T4x12	T4CLKO	T3R	T3_C/ $\bar{T}$	T3x12	T3CLKO

B7 - T4R Timer 4 Run control bit

- 0 : not run Timer 4;
- 1 : run Timer 4.

B6 - T4\_C/ $\bar{T}$ : Counter or timer 4 selector

- 0 : as Timer (namely count on internal system clock)
- 1 : as Counter (namely count on the external pulse input from T4/P0.7)

B5 - T4x12 : Timer 4 clock source bit.

- 0 : The clock source of Timer 4 is SYSclk/12.
- 1 : The clock source of Timer 4 is SYSclk/1.

B4 - T4CLKO : Whether is P0.6 configured for Timer 4(T4) programmable clock output T4CLKO or not.

- 1, P0.6 is configured for Timer 4 programmable clock output T4CLKO, the clock output frequency = T4 overflow/2  
 If T4\_C/ $\bar{T}$  = 0, namely Timer/Counter 4 count on the internal system clock,  
     When T4 in 1T mode (T4T3.5/T4x12=1), the output frequency = (SYSclk)/(65536-[RL\_TH4, RL\_TL4])/2  
     When T4 in 12T mode (T4T3.5/T4x12=0), the output frequency = (SYSclk) /12/ (65536-[RL\_TH4, RL\_TL4])/2  
 If T4\_C/ $\bar{T}$  = 1, namely Timer/Counter 4 count on the external pulse input from P0.7/T4,  
     the output frequency = (T4\_Pin\_CLK) / (65536-[RL\_TH4, RL\_TL4])/2

0, P0.6 is not configure for Timer 4 programmable clock output T4CLKO

B3 - T3R Timer 3 Run control bit

- 0 : not run Timer 3;
- 1 : run Timer 3.

B2 - T3\_C/ $\bar{T}$ : Counter or timer 3 selector

- 0 : as Timer (namely count on internal system clock)
- 1 : as Counter (namely count on the external pulse input from T3/P0.5)

B1 - T3x12 : Timer 3 clock source bit.

- 0 : The clock source of Timer 3 is SYSclk/12.
- 1 : The clock source of Timer 3 is SYSclk/1.

B0 - T3CLKO : Whether is P0.4 configured for Timer 3(T3) programmable clock output T3CLKO or not.

- 1, P0.4 is configured for Timer 3 programmable clock output T3CLKO, the clock output frequency = T3 overflow / 2  
 If T3\_C/ $\bar{T}$  = 0, namely Timer/Counter 3 count on the internal system clock,  
     When T3 in 1T mode (T4T3.1/T3x12=1), the output frequency = (SYSclk)/(65536-[RL\_TH3, RL\_TL3])/2  
     When T3 in 12T mode (T4T3.1/T3x12=0), the output frequency = (SYSclk) /12/ (65536-[RL\_TH3, RL\_TL3])/2  
 If T3\_C/ $\bar{T}$  = 1, namely Timer/Counter 3 count on the external pulse input from P0.5/T3,  
     the output frequency = (T3\_Pin\_CLK) / (65536-[RL\_TH3, RL\_TL3])/2

0, P0.4 is not configure for Timer 3 programmable clock output T3CLKO

---

## 7. T2, T3 and T4 Interrupt Enable Register : IE2

IE2: Interrupt Enable 2 Register (Non bit-addressable)

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

ET4 : Timer 4 interrupt enable bit.

If ET4 = 0, Timer 4 interrupt would be disabled.

If ET4 = 1, Timer 4 interrupt would be enabled.

ET3 : Timer 3 interrupt enable bit.

If ET3 = 0, Timer 3 interrupt would be disabled.

If ET3 = 1, Timer 3 interrupt would be enabled.

ES4 : Serial Port 4 (UART4) interrupt enable bit.

If ES4 = 0, UART4 interrupt would be disabled.

If ES4 = 1, UART4 interrupt would be enabled.

ES3 : Serial Port 3 (UART3) interrupt enable bit.

If ES3 = 0, UART3 interrupt would be disabled.

If ES3 = 1, UART3 interrupt would be enabled.

ET2 : Timer 2 interrupt enable bit.

If ET2 = 0, Timer 2 interrupt would be disabled.

If ET2 = 1, Timer 2 interrupt would be enabled.

ESPI: SPI interrupt enable bit.

If ESPI = 0, SPI interrupt would be disabled.

If ESPI = 1, SPI interrupt would be enabled.

ES2 : Serial Port 2 (UART2) interrupt enable bit.

If ES2 = 0, UART2 interrupt would be disabled.

If ES2 = 1, UART2 interrupt would be enabled.

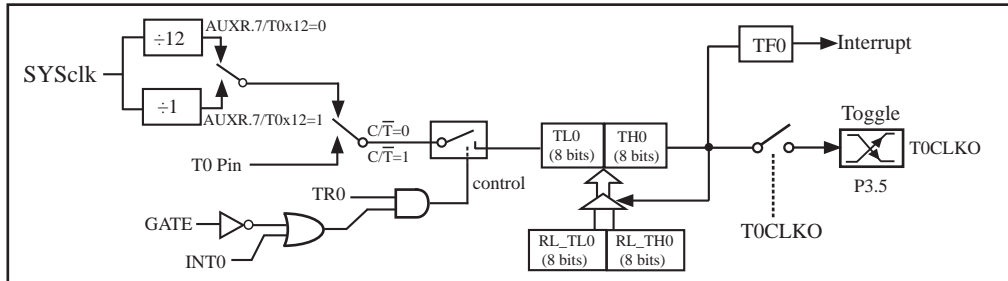


## 7.2 Timer/Counter 0 Modes

Timer/Counter 0 can be configured for four modes by setting M1(TMOD.1) and M0(TMOD.0) in special function register TMOD.

### 7.2.1 Mode 0 (16-Bit Auto-Reload Timer/Counter) and Demo Program

In this mode, the timer/counter 0 is configured as a 16-bit auto-reload timer/counter, which is shown below.



Timer/Counter 0 Mode 0: 16-Bit Auto-Reload Timer/Counter

The counted input is enabled to the timer when TR0 = 1 and either GATE = 0 or INTO = 1. (Setting GATE = 1 allows the Timer to be controlled by external input INTO, to facilitate pulse width measurements.) TR0 is a control bit in the Special Function Register TCON. GATE is in TMOD. There are two different GATE bits, one for Timer 1 (TMOD.7) and one for Timer 0 (TMOD.3).

If  $C/\bar{T} / \text{TMOD}.2 = 0$ , Timer/Counter 0 would be set for Timer operation (input from internal system clock). However, if  $C/\bar{T} / \text{TMOD}.2 = 1$ , Timer/Counter 0 would be set for Counter operation (input from external T0/P3.4 pin).

In the “Timer” function, the timer register [TL0, TH0] is incremented every 12 system clocks or every system clock depending on AUXR.7(T0x12) bit. If T0x12 = 0, the register [TL0, TH0] will be incremented every 12 system clocks. If T0x12 = 1, the register [TL0, TH0] will be incremented every system clock.

There are two hidden registers RL\_TH0 and RL\_TL0 for Timer/Counter 0. The address of RL\_TH0 is the same as TH0's. And, RL\_TL0 and TL0 share in the same address. When TR0 = 0 disable Timer/Counter 0, the content written into register [TL0, TH0] will be written into [RL\_TL0, RL\_TH0] too. When TR0 = 1 enable Timer/Counter 0, the content written into register [TL0, TH0] actually don't be written into [TL0, TH0], but into [RL\_TL0, RL\_TH0]. When users read the content of [TL0, TH0], it is the content of [TL0, TH0] to read instead of [RL\_TL0, RL\_TH0].

When Timer/Counter 0 work in mode 0 (TMOD[1:0] / [M1, M0]=00B), overflow from [TL0, TH0] will not only set TF0, but also reload [TL0, TH0] with the content of [RL\_TL0, RL\_TH0], which is preset by software. The reload leaves [RL\_TL0, RL\_TH0] unchanged.

---

When T0CLKO/INT\_CLKO.0=1, P3.5/T1 is configured for Timer0 programmable clock output T0CLKO.

The clock output frequency =  $T0 \text{ overflow} / 2$

If Timer/Counter 0 in mode 0 (16 bit auto-reloadable mode),

and if  $C/\overline{T} = 0$ , namely Timer/Counter 0 count on the internal system clock,

When T0 in 1T mode (AUXR.7/T0x12=1), the output frequency =  $(SYSclk) / (65536 - [RL\_TH0, RL\_TL0]) / 2$

When T0 in 12T mode (AUXR.7/T0x12=0), the output frequency =  $(SYSclk) / 12 / (65536 - [RL\_TH0, RL\_TL0]) / 2$

and if  $C/\overline{T} = 1$ , namely Timer/Counter 0 count on the external pulse input from P3.4/T0,

the output frequency =  $(T0\_Pin\_CLK) / (65536 - [RL\_TH0, RL\_TL0]) / 2$

RL\_TH0 is the reloaded register of TH0, RL\_TL0 is the reload register of TL0.

## 7.2.1.1 Demo Program of 16-bit Auto-Reload Timer/Counter 0 (C and ASM)

### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of 16-bit auto-reload timer/counter 0 -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

typedef unsigned char    BYTE;
typedef unsigned int     WORD;

//-----

#define FOSC 18432000L

#define T1MS (65536-FOSC/1000)           //1T mode, 18.432KHz
//#define T1MS (65536-FOSC/12/1000)      //12T mode, 18.432KHz

sfr    AUXR  =    0x8e;                //Auxiliary register
sbit   P10   =    P1^0;

//-----
```

---

```

/* Timer0 interrupt routine */
void tm0_isr() interrupt 1 using 1
{
    P10    =    ! P10;
}

//-----

/* main program */
void main()
{
    AUXR   |=    0x80;           //T0 in 1T mode
    //     AUXR   &=    0x7f;       //T0 in 12T mode

    TMOD   =    0x00;           //set T0 as 16-bit auto-reload timer/counter
    TL0    =    T1MS;           //initialize the timing value
    TH0    =    T1MS >> 8;
    TR0    =    1;              //run T0
    ET0    =    1;              //Enable T0 interrupt
    EA     =    1;

    while (1);
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of 16-bit auto-reload timer/counter 0 -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

AUXR   DATA   08EH           //Auxiliary register

;-----

T1MS   EQU     0B800H        //1T mode, the timing value of 1ms is (65536-18432000/1000)
//T1MS EQU     0FA00H        //12Tmode, the timing value of 1ms is (65536-18432000/1000/12)

```

---

```

;-----
    ORG    0000H
    LJMP   MAIN

    ORG    000BH           //interrupt entrance
    LJMP   TOINT

;-----

MAIN:    ORG    0100H
        MOV    SP,    #3FH

        ORL    AUXR, #80H           //T0 in 1T mode
//      ANL    AUXR, #7FH           //T0 in 12T mode

        MOV    TMOD, #00H           //set T0 as 16-bit auto-reload timer/counter

        MOV    TL0,  #LOW T1MS      //initialize the timing value
        MOV    TH0,  #HIGH T1MS
        SETB   TR0
        SETB   ET0                 //Enable T0 interrupt

        SETB   EA

        SJMP   $

//-----
//Timer0 interrupt routine

TOINT:   CPL    P1.0
        RETI

;-----

        END

```

---

---

## 7.2.1.2 Demo Program of T0 Programmable Clock Output (C and ASM)

### —— T0 as 16-bit Auto-Reload Timer/Counter

The following is the example program that Timer 0 output programmable clock by dividing the frequency of internal system clock or the clock input from external pin T0/P3.4 (C and assembly):

#### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of Timer 0 porgrammable clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

#define FOSC 18432000L

//-----
sfr    AUXR      =    0x8e;
sfr    INT_CLKO  =    0x8f;

sbit   T0CLKO   =    P3^5;

#define F38_4KHz (65536-FOSC/2/38400)           //1T Mode
//define F38_4KHz (65536-FOSC/2/12/38400)      //12T Mode
//-----
void main()
{
    AUXR  |=    0x80;           //Timer 0 in 1T mode
//    AUXR  &= ~0x80;         //Timer 0 in 12T mode

    TMOD  =    0x00;           //set Timer0 in mode 0(16 bit auto-reloadable mode)
```

---

```

    TMOD  &=    ~0x04;           //C/T0=0, count on internal system clock
//    TMOD  |=    0x04;           //C/T0=1, count on external pulse input from T0 pin

    TL0   =    F38_4KHz;        //Initial timing value
    TH0   =    F38_4KHz >> 8;
    TR0   =    1;
    INT_CLKO =    0x01;

    while (1);
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of Timer 0 programmable clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

AUXR      DATA    08EH
INT_CLKO  DATA    08FH

T0CLKO    BIT      P3.5

F38_4KHz  EQU      0FF10H      //38.4KHz(1T mode, 65536-18432000/2/38400)
//F38_4KHz EQU      0FFFECH    //38.4KHz(12T mode,(65536-18432000/2/12/38400)
//-----

        ORG      0000H
        LJMP    MAIN

//-----

```

---

```

    ORG    0100H
MAIN:  MOV    SP,    #3FH

    ORL    AUXR,  #80H           //Timer 0 in 1T mode
//     ANL    AUXR,  #7FH           //Timer 0 in 12T mode

    MOV    TMOD,  #00H           //set Timer0 in mode 0(16 bit auto-reloadable mode)

    ANL    TMOD,  #0FBH         //C/T0=0, count on internal system clock
//     ORL    TMOD,  #04H         //C/T0=1, count on external pulse input from T0 pin

    MOV    TL0,   #LOW F38_4KHz  //Initial timing value
    MOV    TH0,   #HIGH F38_4KHz
    SETB   TR0
    MOV    INT_CLKO,  #01H

    SJMP   $

;-----
    END

```

---

### 7.2.1.3 Demo Program using 16-bit auto-reload Timer 0 to Simulate 10 or 16 bits PWM

#### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using 16-bit auto-reload timer/counter to simulate 10 or 16 bits PWM ---*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

#define PWM6BIT      64           //6-bit PWM periodicity
#define PWM8BIT      256          //8-bit PWM periodicity
#define PWM10BIT     1024         //10-bit PWM periodicity
#define PWM16BIT     65536        //16-bit PWM periodicity

#define HIGHDUTY     64           // high duty (duty ratio 64/256=25%)
#define LOWDUTY      (PWM8BIT-HIGHDUTY) //low duty

sfr  AUXR           = 0x8e;      //Auxiliary register
sfr  INT_CLKO       = 0x8f;      //Clock Output register
sbit T0CLKO        = P3^5;      //T0 Clock Output

bit   flag;

// Timer 0 interrupt service routine
void tm0() interrupt 1
{
    flag = !flag;
    if (flag)
    {
        TL0 = (65536-HIGHDUTY);
        TH0 = (65536-HIGHDUTY) >> 8;
    }
    else
    {
        TL0 = (65536-LOWDUTY);
        TH0 = (65536-LOWDUTY) >> 8;
    }
}
```



---

```

void main()
{
    AUXR    =    0x80;           //T0 in 1T mode
    INT_CLKO =    0x01;         //enable the function of Timer 0 Clock Output
    TMOD    &= 0xf0;           //T0 in mode 0(16-bit auto-reload timer/counter)
    TL0     =    (65536-LOWDUTY); //initialize the reload value
    TH0     =    (65536-LOWDUTY) >> 8;
    T0CLKO  =    1;             //initialize the pin of clock output (soft PWM port)
    flag    =    0;
    TR0     =    1;             //run Timer 0
    ET0     =    1;             //enable Timer 0 interrupt
    EA      =    1;
    while (1);
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using 16-bit auto-reload timer/counter to simulate 10 or 16 bits PWM ---*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

;PWM6BIT    EQU    64           ;6-bit PWM periodicity
;PWM8BIT    EQU    256          ;8-bit PWM periodicity
;PWM10BIT   EQU    1024         ;10-bit PWM periodicity
;PWM16BIT   EQU    65536        ;16-bit PWM periodicity

HIGHDUTY    EQU    64           ;high duty (duty ratio 64/256=25%)
LOWDUTY     EQU    (PWM8BIT-HIGHDUTY) ;low duty

AUXR        DATA    08EH       ;Auxiliary register
INT_CLKO    DATA    08FH       ;Clock Output register
T0CLKO      BIT     P3.5         ;T0 Clock Output

FLAG        BIT     20H.0

;-----

```

---

```

    ORG    0000H
    LJMP   MAIN

    ORG    000BH
    LJMP   TM0_ISR

;-----

MAIN:
    MOV    AUXR, #80H           ;T0 in 1T mode
    MOV    INT_CLKO, #01H      ;enable the function of Timer 0 clock output
    ANL    TMOD, #0F0H        ;T0 in mode 0(16-bit auto-reload timer/counter)
    MOV    TL0, #LOW (65536-LOWDUTY) ;initialize the reload value
    MOV    TH0, #HIGH (65536-LOWDUTY)
    SETB   T0CLKO             ;initialize the pin of clock output (soft PWM port)
    CLR    FLAG
    SETB   TR0                ;run Timer 0
    SETB   ET0                ;enable Timer 0 interrupt
    SETB   EA

    SJMP   $

;-----
;Timer 0 interrupt service routine
TM0_ISR:
    CPL    FLAG
    JNB    FLAG, READYLOW
READYHIGH:
    MOV    TL0, #LOW (65536-HIGHDUTY)
    MOV    TH0, #HIGH (65536-HIGHDUTY)
    JMP    TM0ISR_EXIT
READYLOW:
    MOV    TL0, #LOW (65536-LOWDUTY)
    MOV    TH0, #HIGH (65536-LOWDUTY)
TM0ISR_EXIT:
    RETI

;-----

    END

```

---

---

## 7.2.1.4 Demo Program using T0 to expand External Interrupt (Falling edge) —— T0 as 16-bit Auto-Reload Counter (C and ASM)

### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using T0 to extend external interrupt (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

//-----

sfr    AUXR    =    0x8e;           //Auxiliary register
sbit   P10     =    P1^0;

//-----

//Timer 0 Interrupt Service Routine
void t0int() interrupt 1           //Timer 0 interrupt, location at 000BH
{
    P10     =    !P10;
}

void main()
{
    AUXR    =    0x80;           //T0 in 1T mode
    TMOD    =    0x04;           //T0 as external counter
                                   //and T0 in 16-bit auto-reload mode
    TH0     =    TL0     =    0xff; //Set the initial value of T0
    TR0     =    1;           //start up T0
    ET0     =    1;           //Enable T0 interrupt

    EA     =    1;

    while (1);
}
```

---

## 2.Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using T0 to extend external interrupt (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

AUXR    DATA 08EH                //Auxiliary register
//-----

        ORG    0000H
        LJMP   MAIN

        ORG    000BH                //Timer 0 interrupt, location at 000BH
        LJMP   TOINT
//-----

        ORG    0100H
MAIN:    MOV    SP,    #3FH

        MOV    AUXR, #80H           //T0 in 1T mode
        MOV    TMOD, #04H           //T0 as external counter
                                           //and T0 in 16-bit auto-reload mode
        MOV    A,     #0FFH         //Set the initial value of T0
        MOV    TL0,   A
        MOV    TH0,   A
        SETB   TR0                //start up T0
        SETB   ET0                //Enable T0 interrupt

        SETB   EA

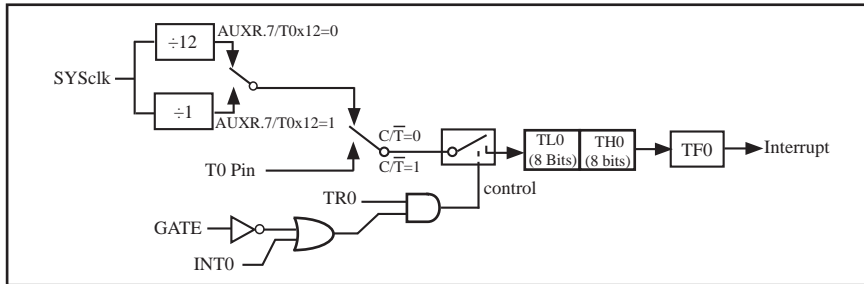
        SJMP   $

//-----
//Timer 0 interrupt service routine

TOINT:   CPL    P1.0
        RETI
;-----
        END
```

## 7.2.2 Mode 1 (16-bit Timer/Counter) and Demo Program (C and ASM)

In this mode, the timer/counter 0 is configured as a 16-bit timer/counter, which is shown below.



Timer/Counter 0 Mode 1 : 16-Bit Timer/Counter

In this mode, the timer register is configured as a 16-bit register. The 16-Bit register consists of all 8 bits of TH0 and the lower 8 bits of TL0. Setting the run flag (TR0) does not clear the registers. As the count rolls over from all 1s to all 0s, it sets the timer interrupt flag TF0.

The counted input is enabled to the timer when TR0 = 1 and either GATE = 0 or INT0 = 1. (Setting GATE = 1 allows the Timer to be controlled by external input INT0, to facilitate pulse width measurements.) TR0 is a control bit in the Special Function Register TCON. GATE is in TMOD. There are two different GATE bits. one for Timer 1 (TMOD.7) and one for Timer 0 (TMOD.3).

If  $C\bar{T} / \text{TMOD}.2 = 0$ , Timer/Counter 0 would be set for Timer operation (input from internal system clock). However, if  $C\bar{T} / \text{TMOD}.2 = 1$ , Timer/Counter 0 would be set for Counter operation (input from external T0/P3.4 pin).

In the “Timer” function, the timer register [TL0, TH0] is incremented every 12 system clocks or every system clock depending on AUXR.7(T0x12) bit. If T0x12 = 0, the register [TL0, TH0] will be incremented every 12 system clocks. If T0x12 = 1, the register [TL0, TH0] will be incremented every system clock.

There are two simple programs that demonstrates Timer 0 as 16-bit Timer/Counter, one written in C language while other in Assembly language.

### 1. C Program:

```

/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 1T Series 16-bit Timer Demo -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

```

---

```

#include "reg51.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;
//-----
/* define constants */
#define FOSC 18432000L
#define MODE1T           //Timer clock mode, comment this line is 12T mode, uncomment is 1T mode

#ifndef MODE1T
#define T1MS (65536-FOSC/1000) //1ms timer calculation method in 1T mode
#else
#define T1MS (65536-FOSC/12/1000) //1ms timer calculation method in 12T mode
#endif

/* define SFR */
sfr  AUXR  = 0x8e; //Auxiliary register
sbit TEST_LED = P0^0; //work LED, flash once per second

/* define variables */
WORD count; //1000 times counter
//-----
/* Timer0 interrupt routine */
void tm0_isr() interrupt 1 using 1
{
    TL0 = T1MS; //reload timer0 low byte
    TH0 = T1MS >> 8; //reload timer0 high byte
    if (count-- == 0) //1ms * 1000 -> 1s
    {
        count = 1000; //reset counter
        TEST_LED = !TEST_LED; //work LED flash
    }
}
//-----
/* main program */
void main()
{
#ifndef MODE1T
    AUXR = 0x80; //timer0 work in 1T mode
#endif
    TMOD = 0x01; //set timer0 as mode1 (16-bit)
    TL0 = T1MS; //initial timer0 low byte
    TH0 = T1MS >> 8; //initial timer0 high byte
    TR0 = 1; //timer0 start running
    ET0 = 1; //enable timer0 interrupt
    EA = 1; //open global interrupt switch
    count = 0; //initial counter

    while (1); //loop
}

```

---

## 2. Assembly Program:

```
;/*-----*/
;/* --- STC MCU International Limited -----*/
;/* --- STC 1T Series 16-bit Timer Demo -----*/
;/* If you want to use the program or the program referenced in the -----*/
;/* article, please specify in which data and procedures from STC -----*/
;/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
;/*---- And only contain < reg51.h > as header file -----*/
;/*-----*/

;/* define constants */
#define MODE1T                ;Timer clock mode, comment this line is 12T mode, uncomment is 1T mode

#ifndef MODE1T
T1MS    EQU 0B800H            ;1ms timer calculation method in 1T mode is (65536-18432000/1000)
#else
T1MS    EQU 0FA00H            ;1ms timer calculation method in 12T mode is (65536-18432000/12/1000)
#endif

;/* define SFR */
    AUXR    DATA    8EH        ;Auxiliary register
    TEST_LED BIT    P1.0        ;work LED, flash once per second

;/* define variables */
    COUNT   DATA    20H        ;1000 times counter (2 bytes)

;-----
    ORG     0000H
    LJMP    MAIN
    ORG     000BH
    LJMP    TM0_ISR

;-----
;/* main program */
MAIN:
#ifndef MODE1T
    MOV     AUXR, #80H          ;timer0 work in 1T mode
#endif
    MOV     TMOD, #01H          ;set timer0 as mode1 (16-bit)
    MOV     TL0, #LOW T1MS      ;initial timer0 low byte
    MOV     TH0, #HIGH T1MS     ;initial timer0 high byte
    SETB    TR0                 ;timer0 start running
    SETB    ET0                 ;enable timer0 interrupt
    SETB    EA                  ;open global interrupt switch
    CLR     A
```

---

```

MOV    COUNT,  A
MOV    COUNT+1, A           ;initial counter
SJMP   $

;-----
;/* Timer0 interrupt routine */
TM0_ISR:
    PUSH    ACC
    PUSH    PSW
    MOV     TL0,    #LOW T1MS           ;reload timer0 low byte
    MOV     TH0,    #HIGH T1MS        ;reload timer0 high byte
    MOV     A,      COUNT
    ORL     A,      COUNT+1           ;check whether count(2byte) is equal to 0
    JNZ     SKIP
    MOV     COUNT, #LOW 1000           ;1ms * 1000 -> 1s
    MOV     COUNT+1,#HIGH 1000
    CPL     TEST_LED                 ;work LED flash
SKIP:
    CLR     C
    MOV     A,      COUNT             ;count--
    SUBB   A,      #1
    MOV     COUNT,  A
    MOV     A,      COUNT+1
    SUBB   A,      #0
    MOV     COUNT+1,A
    POP     PSW
    POP     ACC
    RETI

;-----

END

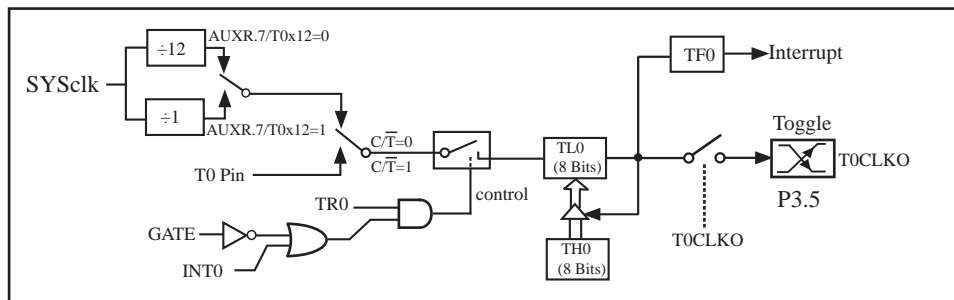
```

---



## 7.2.3 Mode 2 (8-bit Auto-Reload Timer/Counter) and Demo Program

Mode 2 configures the timer register as an 8-bit Timer/Counter(TL0) with automatic reload. Overflow from TL0 not only set TF0, but also reload TL0 with the content of TH0, which is preset by software. The reload leaves TH0 unchanged.



Timer/Counter 0 Mode 2: 8-Bit Auto-Reload

When T0CLKO/INT\_CLKO.0=1, P3.5/T1 is configured for Timer 0 programmable clock output T0CLKO.

The clock output frequency =  $\text{T0 overflow} / 2$

If Timer/Counter 0 in mode 2 (8 bit auto-reloadable mode),

and if  $C/\bar{T} = 0$ , namely Timer/Counter 0 count on the internal system clock,

When T0 in 1T mode(AUSR.7/T0x12=1), the output frequency =  $(\text{SYSclk}) / (256-\text{TH0}) / 2$

When T0 in 12T mode(AUSR.7/T0x12=0), the output frequency =  $(\text{SYSclk}) / 12 / (256-\text{TH0}) / 2$

and if  $C/\bar{T} = 1$ , namely Timer/Counter 0 count on the external pulse input from P3.4/T0,

the output frequency =  $(\text{T0\_Pin\_CLK}) / (256-\text{TH0}) / 2$

---

;T0 Interrupt (falling edge) Demo programs, where T0 operated in Mode 2 (8-bit auto-reload mode)  
; The Timer Interrupt can not wake up MCU from Power-Down mode in the following programs

### 1. C program

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 1T Series MCU T0 (Falling edge) Demo -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

#include "reg51.h"

sfr    AUXR  =    0x8e;                //Auxiliary register

//T0 interrupt service routine
void t0int( ) interrupt 1              //T0 interrupt (location at 000BH)
{
}

void main()
{
    AUXR  =    0x80;                //timer0 work in 1T mode
    TMOD  =    0x06;                //set timer0 as counter mode2 (8-bit auto-reload)
    TL0   =  TH0   =  0xff;         //fill with 0xff to count one time
    TR0   =    1;                  //timer0 start run
    ET0   =    1;                  //enable T0 interrupt
    EA    =    1;                  //open global interrupt switch

    while (1);
}
```

---

## 2. Assembly program

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 1T Series MCU T0 (Falling edge) Demo -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

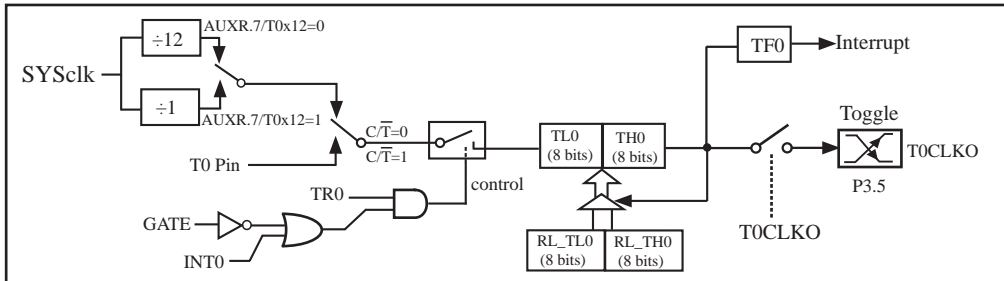
AUXR   DATA   08EH                               ;Auxiliary register
;-----
;interrupt vector table
        ORG    0000H
        LJMP   MAIN
        ORG    000BH                               ;T0 interrupt (location at 000BH)
        LJMP   T0INT
;-----
MAIN:   ORG    0100H
        MOV    SP,    #7FH                         ;initial SP
        MOV    AUXR,  #80H                         ;timer0 work in 1T mode
        MOV    TMOD, #06H                         ;set timer0 as counter mode2 (8-bit auto-reload)
        MOV    A,    #0FFH
        MOV    TL0,  A                             ;fill with 0xff to count one time
        MOV    TH0,  A
        SETB  TR0
        SETB  ET0
        SETB  EA
        SJMP  $
;-----
;T0 interrupt service routine
T0INT:  RETI
;-----
        END
```

## 7.2.4 Mode 3 (16-bit Auto-Reload Timer/Counter whose Interrupt can not be disabled)

Timer/Counter 1 in Mode 3 simply holds its count, the effect is the same as setting TR1 = 0.

For Timer/Counter 0, mode 3 is the same as Mode 0, except that the timer interrupt in mode 3 can not be disabled by EA or ET0 bits. The principle diagram of mode 3 is shown below.

When T0 in mode 3, only can ET0/IE.1=1 enable its interrupt irrespective of EA/IE.7. Once the T0 interrupt is enabled by ET0/IE.1, it will not be disabled by any bit including ET0 and EA bits and will be in the highest priority, which will not be interrupted by any interrupt.



Timer/Counter 0 Mode 3: 16-bit auto-reload Timer/Counter whose interrupt can not be disabled

If Timer/Counter 0 works in mode 3, how is the T0 interrupt enabled.

Setting using C Language:

```

TMOD = 0x11;           //set Timer/Counter 0 in mode 3
TR0 = 1;               //run Timer/Counter 0
//EA = 1;              //Comment EA=1,
                       //the interrupt of T0 in mode 3 is irrespective of EA

ET0 = 1;               //Enable T0 interrupt
    
```

Setting using assembly:

```

MOV    TMOD, #00H     //set Timer/Counter 0 in mode 3
SETB   TR0            //run Timer/Counter 0
//SETB EA             //Comment EA=1,
                       //the interrupt of T0 in mode 3 is irrespective of EA

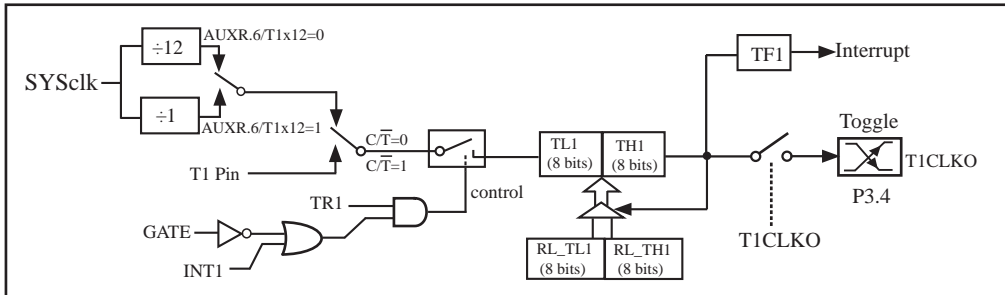
SETB   ET0            //Enable T0 interrupt
    
```

## 7.3 Timer/Counter 1 Modes

Timer/Counter 1 can be configured for three modes by setting M1(TMOD.5) and M0(TMOD.4) in special function register TMOD.

### 7.3.1 Mode 0 (16-Bit Auto-Reload Timer/Counter) and Demo Program

In this mode, the timer/counter 1 is configured as a 16-bit auto-reload timer/counter, which is shown below.



Timer/Counter 1 Mode 0: 16-Bit Auto-Reload Timer/Counter

The counted input is enabled to the timer when TR1 = 1 and either GATE = 0 or INT1 = 1. (Setting GATE = 1 allows the Timer to be controlled by external input INT1, to facilitate pulse width measurements.) TR1 is a control bit in the Special Function Register TCON. GATE is in TMOD. There are two different GATE bits, one for Timer 1 (TMOD.7) and one for Timer 0 (TMOD.3).

If  $C/\bar{T} / \text{TMOD}.6 = 0$ , Timer/Counter 1 would be set for Timer operation (input from internal system clock). However, if  $C/\bar{T} / \text{TMOD}.6 = 1$ , Timer/Counter 1 would be set for Counter operation (input from external T1/P3.5 pin).

In the “Timer” function, the timer register [TL1, TH1] is incremented every 12 system clocks or every system clock depending on AUXR.6(T1x12) bit. If T1x12 = 0, the register [TL1, TH1] will be incremented every 12 system clocks. If T1x12 = 1, the register [TL1, TH1] will be incremented every system clock.

There are two hidden registers RL\_TH1 and RL\_TL1 for Timer/Counter 1. the address of RL\_TH1 is the same as TH1's. And, RL\_TL1 and TL1 share in the same address. When TR1 = 0 disable Timer/Counter 1, the content written into register [TL1, TH1] will be written into [RL\_TL1, RL\_TH1] too. When TR1 = 1 enable Timer/Counter 1, the content written into register [TL1, TH1] actually don't be written into [TL1, TH1], but into [RL\_TL1, RL\_TH1]. When users read the content of [TL1, TH1], it is the content of [TL1, TH1] to read instead of [RL\_TL1, RL\_TH1].

When Timer/Counter 1 work in mode 0(TMOD[5:4]/[M1,M0]=00B), overflow from [TL1, TH1] will not only set TF1, but also reload [TL1, TH1] with the content of [RL\_TL1, RL\_TH1], which is preset by software. The reload leaves [RL\_TL1, RL\_TH1] unchanged.

---

When T1CLKO/INT\_CLKO.1=1, P3.4/T0 is configured for Timer 1 programmable clock output T1CLKO.

The clock output frequency =  $T1 \text{ overflow} / 2$

If Timer/Counter 1 in mode 2 (8 bit auto-reloadable mode),

and if  $C\bar{T} = 0$ , namely Timer/Counter 1 count on the internal system clock,

When T1 in 1T mode(AUXR.6/T1x12=1), the output frequency =  $(SYSclk) / (256-TH1) / 2$

When T1 in 12T mode(AUXR.6/T1x12=0), the output frequency =  $(SYSclk) / 12 / (256-TH1) / 2$

and if  $C\bar{T} = 1$ , namely Timer/Counter 1 count on the external pulse input from P3.5/T1,

the output frequency =  $(T1\_Pin\_CLK) / (256-TH1) / 2$

RL\_TH1 is the reloaded register of TH1, RL\_TL1 is the reload register of TL1.

### 7.3.1.1 Demo Program of 16-bit Auto-Reload Timer/Counter 1 (C and ASM)

#### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of 16-bit auto-reload timer/counter 1 -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

//-----

#define FOSC 18432000L

#define T1MS (65536-FOSC/1000) //1T mode, 18.432KHz
//#define T1MS (65536-FOSC/12/1000) //12T mode, 18.432KHz

sfr AUXR = 0x8e; //Auxiliary register
sbit P10 = P1^0;

//-----
```

---

```

/* Timer1 interrupt routine */
void tm1_isr() interrupt 3 using 1
{
    P10    =    ! P10;
}

//-----

/* main program */
void main()
{
    AUXR  |=    0x40;           //T1 in 1T mode
    //    AUXR  &= 0xdf;       //T1 in 12T mode

    TMOD  =    0x00;           //set T1 as 16-bit auto-reload timer/counter
    TL1   =    T1MS;           //initialize the timing value
    TH1   =    T1MS    >> 8;
    TR1   =    1;              //run T1
    ET1   =    1;              //Enable T1 interrupt
    EA    =    1;

    while (1);
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of 16-bit auto-reload timer/counter 1 -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

AUXR  DATA  08EH           //Auxiliary register

;-----
T1MS  EQU    0B800H        //1T mode, the timing value of 1ms is (65536-18432000/1000)
//T1MS EQU    0FA00H        //12Tmode, the timing value of 1ms is (65536-18432000/1000/12)

```

---

```

;-----
    ORG    0000H
    LJMP   MAIN

    ORG    001BH
    LJMP   T1INT

;-----

MAIN:    ORG    0100H
        MOV    SP,    #3FH

        ORL    AUXR, #40H           //T1 in 1T mode
//      ANL    AUXR, #0DFH         //T1 in 12T mode

        MOV    TMOD, #00H           //set T1 as 16-bit auto-reload timer/counter

        MOV    TL1,  #LOW T1MS      //initialize the timing value
        MOV    TH1,  #HIGH T1MS
        SETB   TR1
        SETB   ET1                   //run T1

        SETB   EA

        SJMP  $

//-----
//Timer1 interrupt routine

T1INT:   CPL    P1.0
        RETI

;-----

        END

```



---

### 7.3.1.2 Demo Program of T1 Programmable Clock Output (C and ASM)

#### —— T1 as 16-bit Auto-Reload Timer/Counter

The following is the example program that Timer 1 output programmable clock by dividing the frequency of internal system clock or the clock input from external pin T1/P3.5 (C and assembly):

#### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of Timer 1 programmable clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

typedef unsigned char    BYTE;
typedef unsigned int     WORD;

#define FOSC    18432000L

//-----
sfr  AUXR      =    0x8e;
sfr  INT_CLKO  =    0x8f;

sbit  T1CLKO   =    P3^4;

#define F38_4KHz    (65536-FOSC/2/38400)           //1T Mode
//#define F38_4KHz    (65536-FOSC/2/12/38400)       //12T Mode

//-----
void main()
{
    AUXR  |=    0x40;           //Timer 1 in 1T mode
//    AUXR  &=    ~0x40;       //Timer 1 in 12T mode
```

---

```

    TMOD = 0x00; //set Timer 1 in mode 0(16 bit auto-reloadable mode)

    TMOD &= ~0x40; //C/T1=0, count on internal system clock
//    TMOD |= 0x40; //C/T1=1, count on external pulse input from T1 pin

    TL1 = F38_4KHz; //Initial timing value
    TH1 = F38_4KHz >> 8;
    TR1 = 1;
    INT_CLKO = 0x02;

    while (1);
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of Timer 1 porgrammable clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

AUXR      DATA 08EH
INT_CLKO  DATA 08FH

T1CLKO    BIT    P3.4
F38_4KHz  EQU    0FF10H //38.4KHz(1T mode, 65536-18432000/2/38400)
//F38_4KHz EQU    0FFECH //38.4KHz(12T mode, (65536-18432000/2/12/38400)

    ORG 0000H
    LJMP MAIN

//-----

    ORG 0100H

```

---

---

MAIN:

```
MOV    SP,    #3FH

ORL    AUXR,  #40H           //Timer 1 in 1T mode
// ANL    AUXR,  #0BFH       //Timer 1 in 12T mode

MOV    TMOD,  #00H           //set Timer 1 in mode 0(16 bit auto-reloadable mode)

ANL    TMOD,  #0BFH         //C/T1=0, count on internal system clock
// ORL    TMOD,  #40H         //C/T1=1, count on external pulse input from T1 pin

MOV    TL1,   #LOW F38_4KHz  //Initial timing value
MOV    TH1,   #HIGH F38_4KHz
SETB   TR1
MOV    INT_CLKO,    #02H

SJMP   $
```

-----

END

---

### 7.3.1.3 Demo Program using 16-bit auto-reload Timer 1 as UART1 baud-rate Generator

#### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using 16-bit auto-reload timer/counter 1 as UART1 baud-rate generator */
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

typedef unsigned char    BYTE;
typedef unsigned int     WORD;

#define FOSC    18432000L           //system frequency
#define BAUD    115200             //baud-rate

#define NONE_PARITY      0         //none parity
#define ODD_PARITY       1         //odd parity
#define EVEN_PARITY      2         //even parity
#define MARK_PARITY      3         //mark parity
#define SPACE_PARITY     4         //space parity

#define PARITYBIT EVEN_PARITY      //define the parity bit

sfr    AUXR    =    0x8e;         //Auxiliary register

sbit   P22     =    P2^2;

bit    busy;

void SendData(BYTE dat);
void SendString(char *s);
```

---

```

void main()
{
#if (PARITYBIT == NONE_PARITY)
    SCON = 0x50; //8-bit variable baud-rate
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    SCON = 0xda; //9-bit variable baud-rate
                //the parity bit is initialized for 1
#elif (PARITYBIT == SPACE_PARITY)
    SCON = 0xd2; //9-bit variable baud-rate
                //the parity bit is initialized for 0
#endif

    AUXR = 0x40; //T1 in 1T mode
    TMOD = 0x00; //T1 in mode 0 (16-bit auto-reload timer/counter)
    TL1 = (65536 - (FOSC/32/BAUD)); //set the preload value
    TH1 = (65536 - (FOSC/32/BAUD))>>8;
    TR1 = 1; //run T1
    ES = 1; //enable UART1 interrupt
    EA = 1;

    SendString("STC15W4K32S4\r\nUart Test !\r\n");
    while(1);
}

/*-----
UART Interrupt Service Routine
-----*/
void Uart() interrupt 4 using 1
{
    if (RI)
    {
        RI = 0; //clear RI
        P0 = SBUF; //serial data is shown in P0
        P22 = RB8; //P2.2 display parity bit
    }
    if (TI)
    {
        TI = 0; //clear TI
        busy = 0; //clear busy flag
    }
}

```

---

---

```

/*-----
Send UART data
-----*/
void SendData(BYTE dat)
{
    while (busy);                //wait to finish sending the previous data
    ACC = dat;                    // access to the parity bit ---- P (PSW.0)
    if (P)
    {
        #if (PARITYBIT == ODD_PARITY)
            TB8 = 0;              //the parity bit is set for 0
        #elif (PARITYBIT == EVEN_PARITY)
            TB8 = 1;              //the parity bit is set for 1
        #endif
    }
    else
    {
        #if (PARITYBIT == ODD_PARITY)
            TB8 = 1;              //the parity bit is set for 1
        #elif (PARITYBIT == EVEN_PARITY)
            TB8 = 0;              //the parity bit is set for 0
        #endif
    }
    busy = 1;
    SBUF = ACC;                   //write the data into SBUF of UART
}

/*-----
Send string
-----*/
void SendString(char *s)
{
    while (*s)
    {
        SendData(*s++);          //send the current char
    }
}

```

---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using 16-bit auto-reload timer/counter 1 as UART1 baud-rate generator */
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

#define NONE_PARITY 0 //none parity
#define ODD_PARITY 1 //odd parity
#define EVEN_PARITY 2 //even parity
#define MARK_PARITY 3 //mark parity
#define SPACE_PARITY 4 //space parity

#define PARITYBIT EVEN_PARITY //define the parity bit

//-----

AUXR EQU 08EH //Auxiliary register
BUSY BIT 20H.0

//-----

ORG 0000H
LJMP MAIN

ORG 0023H
LJMP UART_ISR

//-----

ORG 0100H
MAIN:
CLR BUSY
CLR EA
MOV SP, #3FH

#if (PARITYBIT == NONE_PARITY)
MOV SCON, #50H //8-bit variable baud-rate
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
MOV SCON, #0DAH //9-bit variable baud-rate, the parity bit is initialized for 1
#elif (PARITYBIT == SPACE_PARITY)
MOV SCON, #0D2H //9-bit variable baud-rate, the parity bit is initialized for 0
#endif
```

---

```

//-----
MOV    AUXR, #40H           //T1 in 1T mode
MOV    TMOD, #00H          //T1 in mode 0 (16-bit auto-reload timer/counter)
MOV    TL1,  #0FBH         //set the preload value (65536-18432000/32/115200)
MOV    TH1,  #0FFH
SETB   TR1                  //run T1
SETB   ES                   //enable UART1 interrupt
SETB   EA

MOV    DPTR, #TESTSTR
LCALL SENDSTRING

SJMP  $

;-----
TESTSTR:
    DB "STC15W4K32S4 Uart1 Test !",0DH,0AH,0

;-----
;UART Interrupt Service Routine
;-----*/
UART_ISR:
    PUSH ACC
    PUSH PSW
    JNB RI, CHECKTI
    CLR RI                   //clear RI
    MOV P0, SBUF             //serial data is shown in P0
    MOV C, RB8
    MOV P2.2, C              //P2.2 display parity bit

CHECKTI:
    JNB TI, ISR_EXIT
    CLR TI                   //clear TI
    CLR BUSY                 //clear busy flag

ISR_EXIT:
    POP PSW
    POP ACC
    RETI

;-----
;Send UART data
;-----*/
SENDDATA:
    JB BUSY, $               //wait to finish sending the previous data
    MOV ACC, A               //access to the parity bit ---- P (PSW.0)
    JNB P, EVEN1INACC

```

---



---

```

ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
    CLR    TB8                                //the parity bit is set for 0
#elif (PARITYBIT == EVEN_PARITY)
    SETB   TB8                                //the parity bit is set for 1
#endif
    SJMP   PARITYBITOK
EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
    SETB   TB8                                //the parity bit is set for 1
#elif (PARITYBIT == EVEN_PARITY)
    CLR    TB8                                //the parity bit is set for 0
#endif
PARITYBITOK:
    SETB   BUSY
    MOV    SBUF, A                            //write the data into SBUF of UART
    RET

; /*-----
; Send string
; /*-----*/
SENDSTRING:
    CLR    A
    MOVC   A, @A+DPTR
    JZ     STRINGEND
    INC    DPTR
    LCALL  SENDDATA
    SJMP   SENDSTRING
STRINGEND:
    RET
; /*-----
    END

```

---

### 7.3.1.4 Demo Program using T1 to expand External Interrupt (Falling edge) —— T1 as 16-bit Auto-Reload Counter (C and ASM)

#### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using T1 to extend external interrupt (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

//-----

sfr    AUXR  =    0x8e;           //Auxiliary register
sbit   P10   =    P1^0;

//-----

//Timer 1 Interrupt Service Routine
void t1int() interrupt 3         //Timer 1 interrupt, location at 001BH
{
    P10    =    !P10;
}

void main()
{
    AUXR   =    0x40;           //T1 in 1T mode
    TMOD   =    0x40;           //T1 as external counter
                                           //and T1 in 16-bit auto-reload mode
    TH1    =    TL1    =    0xff; //Set the initial value of T1
    TR1    =    1;             //start up T1
    ET1    =    1;             //Enable T1 interrupt

    EA     =    1;

    while (1);
}
```

---

## 2.Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using T1 to extend external interrupt (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

AUXR    DATA 08EH                //Auxiliary register
//-----

        ORG    0000H
        LJMP   MAIN

        ORG    001BH                //Timer 1 interrupt, location at 001BH
        LJMP   T1INT
//-----

MAIN:   ORG    0100H

        MOV    SP,    #3FH

        MOV    AUXR, #40H           //T1 in 1T mode
        MOV    TMOD, #40H           //T1 as external counter
                                           //and T1 in 16-bit auto-reload mode
        MOV    A,     #0FFH         //Set the initial value of T1
        MOV    TL1,   A
        MOV    TH1,   A
        SETB   TR1                //start up T1
        SETB   ET1                //Enable T1 interrupt

        SETB   EA

        SJMP   $

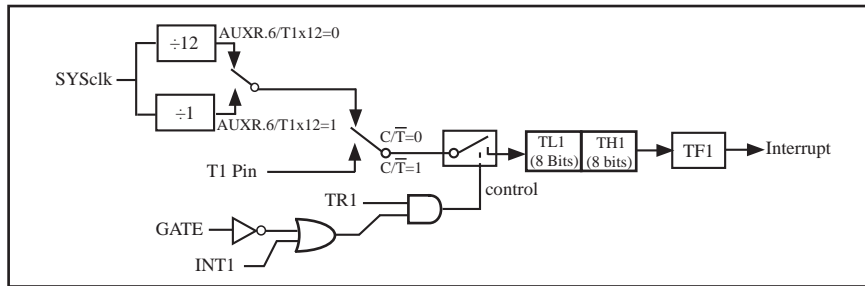
//-----
//Timer 1 Interrupt Service Routine

T1INT:  CPL    P1.0

RETI
;-----
        END
```

## 7.3.2 Mode 1 (16-bit Timer/Counter) and Demo Programs (C and ASM)

In this mode, the timer/counter 1 is configured as a 16-bit timer/counter, which is shown below.



Timer/Counter 1 Mode 1 : 16-Bit Timer/Counter

In this mode, the timer register is configured as a 16-bit register. The 16-Bit register consists of all 8 bits of TH1 and the lower 8 bits of TL1. Setting the run flag (TR1) does not clear the registers. As the count rolls over from all 1s to all 0s, it sets the timer interrupt flag TF1.

The counted input is enabled to the timer when TR1 = 1 and either GATE = 0 or INT1 = 1. (Setting GATE = 1 allows the Timer to be controlled by external input INT1, to facilitate pulse width measurements.) TR1 is a control bit in the Special Function Register TCON. GATE is in TMOD. There are two different GATE bits, one for Timer 1 (TMOD.7) and one for Timer 0 (TMOD.3).

If  $C/\overline{T} / \text{TMOD}.6 = 0$ , Timer/Counter 1 would be set for Timer operation (input from internal system clock). However, if  $C/\overline{T} / \text{TMOD}.6 = 1$ , Timer/Counter 1 would be set for Counter operation (input from external T1/P3.5 pin).

In the “Timer” function, the timer register [TL1, TH1] is incremented every 12 system clocks or every system clock depending on AUXR.6(T1x12) bit. If T1x12 = 0, the register [TL1, TH1] will be incremented every 12 system clocks. If T1x12 = 1, the register [TL1, TH1] will be incremented every system clock.

There are another two simple programs that demonstrates Timer 1 as 16-bit Timer/Counter, one written in C language while other in Assembly language.

### 1. C Program

```

/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 1T Series 16-bit Timer Demo -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

```

---

```

#include "reg51.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;
//-----
/* define constants */
#define FOSC 1843200L
#define MODE1T           //Timer clock mode, comment this line is 12T mode, uncomment is 1T mode

#ifndef MODE1T
#define T1MS (65536-FOSC/1000)           //1ms timer calculation method in 1T mode
#else
#define T1MS (65536-FOSC/12/1000)       //1ms timer calculation method in 12T mode
#endif

/* define SFR */
sfr  AUXR    = 0x8e;           //Auxiliary register
sbit  TEST_LED = P0^0;        //work LED, flash once per second

/* define variables */
WORD count;                   //1000 times counter
//-----
/* Timer0 interrupt routine */
void tm1_isr() interrupt 3 using 1
{
    TL1 = T1MS;                //reload timer1 low byte
    TH1 = T1MS >> 8;          //reload timer1 high byte
    if (count-- == 0)          //1ms * 1000 -> 1s
    {
        count = 1000;         //reset counter
        TEST_LED = !TEST_LED; //work LED flash
    }
}
//-----
/* main program */
void main()
{
#ifndef MODE1T
    AUXR = 0x40;               //timer1 work in 1T mode
#endif
    TMOD = 0x10;               //set timer1 as mode1 (16-bit)
    TL1 = T1MS;                //initial timer1 low byte
    TH1 = T1MS >> 8;          //initial timer1 high byte
    TR1 = 1;                   //timer1 start running
    ET1 = 1;                   //enable timer1 interrupt
    EA = 1;                     //open global interrupt switch
    count = 0;                  //initial counter

    while (1);                 //loop
}

```

---

---

## 2. Assembly Program

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 1T Series 16-bit Timer Demo -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

/* define constants */
#define MODE1T           ;Timer clock mode, comment this line is 12T mode, uncomment is 1T mode

#ifndef MODE1T
TIMS EQU 0B800H        ;1ms timer calculation method in 1T mode is (65536-18432000/1000)
#else
TIMS EQU 0FA00H        ;1ms timer calculation method in 12T mode is (65536-18432000/12/1000)
#endif

/* define SFR */
AUXR DATA 8EH        ;Auxiliary register
TEST_LED BIT P1.0     ;work LED, flash once per second

/* define variables */
COUNT DATA 20H      ;1000 times counter (2 bytes)

;-----
ORG 0000H
LJMP MAIN
ORG 001BH
LJMP TM1_ISR

;-----

/* main program */
MAIN:
#ifndef MODE1T
MOV AUXR, #40H        ;timer1 work in 1T mode
#endif

MOV TMOD, #10H        ;set timer1 as mode1 (16-bit)
MOV TL1, #LOW TIMS    ;initial timer1 low byte
MOV TH1, #HIGH TIMS   ;initial timer1 high byte
SETB TR1              ;timer1 start running
SETB ET1              ;enable timer1 interrupt
SETB EA               ;open global interrupt switch
CLR A
```

---

```

MOV    COUNT, A
MOV    COUNT+1,A           ;initial counter
SJMP   $

;-----

;*/ Timer1 interrupt routine */
TM1_ISR:
    PUSH    ACC
    PUSH    PSW
    MOV     TL1,    #LOW T1MS           ;reload timer1 low byte
    MOV     TH1,    #HIGH T1MS         ;reload timer1 high byte
    MOV     A,      COUNT
    ORL    A,      COUNT+1             ;check whether count(2byte) is equal to 0
    JNZ    SKIP
    MOV     COUNT, #LOW 1000            ;1ms * 1000 -> 1s
    MOV     COUNT+1,#HIGH 1000
    CPL    TEST_LED                    ;work LED flash
SKIP:
    CLR    C
    MOV     A,      COUNT               ;count--
    SUBB   A,      #1
    MOV     COUNT, A
    MOV     A,      COUNT+1
    SUBB   A,      #0
    MOV     COUNT+1,A
    POP    PSW
    POP    ACC
    RETI

;-----

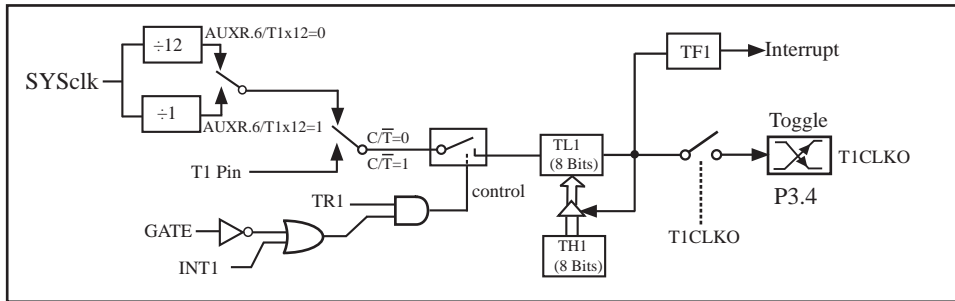
END

```

---

### 7.3.3 Mode 2 (8-bit Auto-Reload Timer/Counter) and Demo Program

Mode 2 configures the timer register as an 8-bit t Timer/Counter (TL1) with automatic reload. Overflow from TL1 not only set TF1, but also reload TL1 with the content of TH1, which is preset by software. The reload leaves TH1 unchanged.



Timer/Counter 1 Mode 2: 8-Bit Auto-Reload

When T1CLKO/INT\_CLKO.1=1, P3.4/T0 is configured for Timer 1 programmable clock output T1CLKO. The clock output frequency =  $T1 \text{ overflow} / 2$

If Timer/Counter 1 in mode 2 (8 bit auto-reloadable mode),

and if  $C/\bar{T} = 0$ , namely Timer/Counter 1 count on the internal system clock,

When T1 in 1T mode(AUXR.6/T1x12=1), the output frequency =  $(SYSclk) / (256-TH1) / 2$

When T1 in 12T mode(AUXR.6/T1x12=0), the output frequency =  $(SYSclk) / 12 / (256-TH1) / 2$

and if  $C/\bar{T} = 1$ , namely Timer/Counter 1 count on the external pulse input from P3.5/T1,

the output frequency =  $(T1\_Pin\_CLK) / (256-TH1) / 2$

RL\_TH1 is the reloaded register of TH1, RL\_TL1 is the reload register of TL1.



---

### 7.3.3.1 Demo Program using 8-bit auto-reload Timer 1 as UART1 baud-rate Generator

#### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using 8-bit auto-reload timer/counter 1 as UART1 baud-rate generator ---*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

#include "intrins.h"

typedef unsigned char      BYTE;
typedef unsigned int       WORD;

#define FOSC 18432000L      //system frequency
#define BAUD 115200        //baud-rate

#define NONE_PARITY        0    //none parity
#define ODD_PARITY         1    //odd parity
#define EVEN_PARITY        2    //even parity
#define MARK_PARITY        3    //mark parity
#define SPACE_PARITY       4    //space parity

#define PARITYBIT EVEN_PARITY //define the parity bit

sfr  AUXR  = 0x8e;        //Auxiliary register
sbit P22   = P2^2;
bit  busy;

void SendData(BYTE dat);
void SendString(char *s);

void main()
{
#if (PARITYBIT == NONE_PARITY)
    SCON = 0x50;          //8-bit variable baud-rate
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    SCON = 0xda;          //9-bit variable baud-rate, the parity bit is initialized for 1
```

---

```

#elif (PARITYBIT == SPACE_PARITY)
    SCON = 0xd2;           //9-bit variable baud-rate, the parity bit is initialized for 0
#endif

    AUXR = 0x40;           //T1 in 1T mode
    TMOD = 0x20;           //T1 in mode2 (8-bit auto-reload timer/counter)
    TL1 = (256 - (FOSC/32/BAUD)); //set the preload value
    TH1 = (256 - (FOSC/32/BAUD));
    TR1 = 1;              //run T1
    ES = 1;               //enable UART1 interrupt
    EA = 1;

    SendString("STC15W4K32S4\r\nUart Test !\r\n");
    while(1);
}

/*-----
UART Interrupt Service Routine
-----*/
void Uart() interrupt 4 using 1
{
    if (RI)
    {
        RI = 0;           //clear RI
        P0 = SBUF;        //serial data is shown in P0
        P22 = RB8;        //P2.2 display parity bit
    }
    if (TI)
    {
        TI = 0;           //clear TI
        busy = 0;         //clear busy flag
    }
}

/*-----
Send UART data
-----*/
void SendData(BYTE dat)
{
    while (busy);        //wait to finish sending the previous data
    ACC = dat;           //access to the parity bit ---- P (PSW.0)
    if (P)
    {
        #if (PARITYBIT == ODD_PARITY)
            TB8 = 0;      //the parity bit is set for 0
        #elif (PARITYBIT == EVEN_PARITY)
            TB8 = 1;      //the parity bit is set for 1
        #endif
    }
}

```

---

---

```

    else
    {
    #if (PARITYBIT == ODD_PARITY)
        TB8 = 1;           //the parity bit is set for 1
    #elif (PARITYBIT == EVEN_PARITY)
        TB8 = 0;          //the parity bit is set for 0
    #endif
    }
    busy = 1;
    SBUF = ACC;           //write the data into SBUF of UART
}

/*-----
Send string
-----*/
void SendString(char *s)
{
    while (*s)
    {
        SendData(*s++);
    }
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using 8-bit auto-reload timer/counter 1 as UART1 baud-rate generator -*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#define NONE_PARITY 0           //none parity
#define ODD_PARITY 1           //odd parity
#define EVEN_PARITY 2          //even parity
#define MARK_PARITY 3          //mark parity
#define SPACE_PARITY 4         //space parity

#define PARITYBIT EVEN_PARITY  //define the parity bit

//-----

```

```

AUXR EQU 08EH //Auxiliary register
BUSY BIT 20H.0

//-----
ORG 0000H
LJMP MAIN

ORG 0023H
LJMP UART_ISR
//-----

MAIN:
ORG 0100H
CLR BUSY
CLR EA
MOV SP, #3FH

#if (PARITYBIT == NONE_PARITY)
MOV SCON, #50H //8-bit variable baud-rate
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
MOV SCON, #0DAH //9-bit variable baud-rate, the parity bit is initialized for 1
#elif (PARITYBIT == SPACE_PARITY)
MOV SCON, #0D2H //9-bit variable baud-rate, the parity bit is initialized for 0
#endif

//-----
MOV AUXR, #40H //T1 in 1T mode
MOV TMOD, #20H //T1 in mode2 (8-bit auto-reload timer/counter)
MOV TL1, #0FBH //set the preload value (256-18432000/32/115200)
MOV TH1, #0FBH
SETB TR1 //run T1
SETB ES //enable UART1 interrupt
SETB EA

MOV DPTR, #TESTSTR
LCALL SENDSTRING

SJMP $

;-----
TESTSTR:
DB "STC15W4K32S4 Uart1 Test !",0DH,0AH,0

; /*-----
; UART Interrupt Service Routine
; -----*/
UART_ISR:
PUSH ACC
PUSH PSW
JNB RI, CHECKTI
CLR RI //clear RI
MOV P0, SBUF //serial data is shown in P0
MOV C, RB8

```

---

```

        MOV     P2.2,  C           //P2.2 display parity bit
CHECKTI:
        JNB    TI,      ISR_EXIT
        CLR    TI           //clear TI
        CLR    BUSY      //clear busy flag
ISR_EXIT:
        POP    PSW
        POP    ACC
        RETI

;/*-----
;Send UART data
;-----*/
SENDDATA:
        JB     BUSY, $           //wait to finish sending the previous data
        MOV    ACC,  A           //access to the parity bit ---- P (PSW.0)
        JNB   P,      EVEN1INACC
ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
        CLR    TB8           //the parity bit is set for 0
#elif (PARITYBIT == EVEN_PARITY)
        SETB   TB8           //the parity bit is set for 1
#endif
        SJMP   PARITYBITOK
EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
        SETB   TB8           //the parity bit is set for 1
#elif (PARITYBIT == EVEN_PARITY)
        CLR    TB8           //the parity bit is set for 0
#endif
PARITYBITOK:
        SETB   BUSY
        MOV    SBUF,  A           //write the data into SBUF of UART
        RET

;/*-----
;Send string
//-----*/
SENDSTRING:
        CLR    A
        MOVC  A,      @A+DPTR
        JZ    STRINGEND
        INC   DPTR
        LCALL SENDDATA
        SJMP SENDSTRING
STRINGEND:
        RET

//-----
        END

```

---

---

### 7.3.3.2 Demo Program using T1 to expand External Interrupt (Falling edge) —— T1 as 8-bit Auto-Reload Counter (C and ASM)

;T1 Interrupt (falling edge) Demo programs, where T1 operated in Mode 2 (8-bit auto-reload mode)  
; The Timer Interrupt can not wake up MCU from Power-Down mode in the following programs

#### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using T1 to extend external interrupt (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

sfr AUXR = 0x8e; //Auxiliary register

//T1 interrupt service routine
void t1int() interrupt 3 //T1 interrupt (location at 001BH)
{
}

void main()
{
    AUXR = 0x40; //timer1 work in 1T mode
    TMOD = 0x60; //set timer1 as counter mode2 (8-bit auto-reload)
    TL1 = TH1 = 0xff; //fill with 0xff to count one time
    TR1 = 1; //timer1 start run
    ET1 = 1; //enable T1 interrupt
    EA = 1; //open global interrupt switch

    while (1);
}
```

---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using T1 to extend external interrupt (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

AUXR  DATA  08EH                ;Auxiliary register
;-----
;interrupt vector table

        ORG    0000H
        LJMP   MAIN

        ORG    001BH                ;T1 interrupt (location at 001BH)
        LJMP   T1INT
;-----

        ORG    0100H
MAIN:   MOV     SP,    #7FH            ;initial SP
        MOV     AUXR, #40H            ;timer1 work in 1T mode
        MOV     TMOD, #60H            ;set timer1 as counter mode2 (8-bit auto-reload)
        MOV     A,    #0FFH
        MOV     TL1,  A                ;fill with 0xff to count one time
        MOV     TH1,  A
        SETB   TR1                    ;timer1 start run
        SETB   ET1                    ;enable T1 interrupt
        SETB   EA                    ;open global interrupt switch
        SJMP   $

;-----
;T1 interrupt service routine
T1INT:  RETI
;-----

        END
```

## 7.4 Timer/Counter 2

Timer/Counter 2 only have one mode : 16-bit auto-reload timer/counter. Besides as Timer/Counter, T2 also can be as the baud-rate generator and programmable clock output.

### 7.4.1 Special Function Registers about Timer/Counter 2

Symbol	Description	Address	Bit Address and Symbol								Value after Power-on or Reset
			MSB				LSB				
T2H	The high 8-bit of Timer 2 register	D6H									0000 0000B
T2L	The low 8-bit of Timer 2 register	D7H									0000 0000B
AUXR	Auxiliary register	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C $\bar{T}$	T2x12	EXTRAM	S1ST2	0000 0001B
INT_CLKO AUXR2	External Interrupt enable and Clock Output register	8FH	-	EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	TOCLKO	x000 0000B
IE2	Interrupt Enable register	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000 0000B

#### 1. AUXR: Auxiliary register (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C $\bar{T}$	T2x12	EXTRAM	S1ST2

B4 - T2R Timer 2 Run control bit

- 0 : not run Timer 2;
- 1 : run Timer 2.

B3 - T2\_C $\bar{T}$ : Counter or timer 2 selector

- 0 : as Timer (namely count on internal system clock)
- 1 : as Counter (namely count on the external pulse input from T2/P3.1)

B2 - T2x12 : Timer 2 clock source bit.

- 0 : The clock source of Timer 2 is SYSclk/12.
- 1 : The clock source of Timer 2 is SYSclk/1.

If T2 is used as the baud-rate generator of UART1 or UART2, T1x12 will decide whether UART1 or UART2 is 1T or 12T.

B0 - S1ST2 : the control bit that UART1 select Timer 2 as its baud-rate generator.

- 0 : Select Timer 1 as the baud-rate generator of UART1
- 1 : Select Timer 2 as the baud-rate generator of UART1. Timer 1 is released to use in other functions.

B7 - T0x12 : Timer 0 clock source bit.

- 0 : The clock source of Timer 0 is SYSclk/12. It will compatible to the traditional 8051 MCU
- 1 : The clock source of Timer 0 is SYSclk/1. It will drive the T0 faster than a traditional 8051 MCU



---

B6 - T1x12 : Timer 1 clock source bit.

0 : The clock source of Timer 1 is SYSclk/12. It will compatible to the traditional 8051 MCU

1 : The clock source of Timer 1 is SYSclk/1. It will drive the T0 faster than a traditional 8051 MCU

If T1 is used as the baud-rate generator of UART1, T1x12 will decide whether UART1 is 1T or 12T.

B5 - UART\_M0x6 : Baud rate select bit of UART1 while it is working under 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.

B1 - EXTRAM : Internal / external RAM access control bit.

0 : On-chip auxiliary RAM is enabled.

1 : On-chip auxiliary RAM is always disabled.

## 2. T2 Clock Output control bit : T2CLKO

The output clock frequency of T2CLKO is controlled by Timer 2 which only has one mode (16-bit auto-reload timer/counter). Similarly, when T2 is used as programmable clock output, it also don't enable thier interrupt to avoid CPU entering interrupt repeatedly unless special circumstances.

INT\_CLKO (AUXR2) : Clock Output and External Interrupt Enable register (Non bit-Addressable)

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name	-	EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO

B2 - T2CLKO : Whether is P3.0 configured for Timer 2(T2) programmable clock output T2CLKO or not.

1, P3.0 is configured for Timer2 programmable clock output T2CLKO, the clock output frequency = T2 overflow/2

If  $T2\_C\bar{T} = 0$ , namely Timer/Counter 2 count on the internal system clock,

When T2 in 1T mode (AUXR.2/T2x12=1), the output frequency = (SYSclk)/(65536-[RL\_TH2, RL\_TL2])/2

When T2 in 12T mode (AUXR.2/T2x12=0), the output frequency = (SYSclk) /12/ (65536-[RL\_TH2, RL\_TL2])/2

If  $T2\_C\bar{T} = 1$ , namely Timer/Counter 2 count on the external pulse input from P3.1/T2,

the output frequency = (T2\_Pin\_CLK) / (65536-[RL\_TH2, RL\_TL2])/2

0, P3.0 is not configure for Timer 2 programmable clock output T2CLKO

B0 - T0CLKO : Whether is P3.5/T1 configured for Timer 0(T0) programmable clock output T0CLKO or not.

1, P3.5/T1 is configured for Timer0 programmable clock output T0CLKO, the clock output frequency = T0 overflow/2

If Timer/Counter 0 in mode 0 (16 bit auto-reloadable mode),

and if  $C\bar{T} = 0$ , namely Timer/Counter 0 count on the internal system clock,

When T0 in 1T mode (AUXR.7/T0x12=1), the output frequency = (SYSclk)/(65536-[RL\_TH0, RL\_TL0])/2

When T0 in 12T mode (AUXR.7/T0x12=0), the output frequency = (SYSclk) /12/ (65536-[RL\_TH0, RL\_TL0])/2

and if  $C\bar{T} = 1$ , namely Timer/Counter 0 count on the external pulse input from P3.4/T0,

the output frequency = (T0\_Pin\_CLK) / (65536-[RL\_TH0, RL\_TL0])/2

If Timer/Counter 0 in mode 2 (8 bit auto-reloadable mode),

and if  $C\bar{T} = 0$ , namely Timer/Counter 0 count on the internal system clock,

When T0 in 1T mode(AUXR.7/T0x12=1), the output frequency = (SYSclk) / (256-TH0) / 2

When T0 in 12T mode(AUXR.7/T0x12=0), the output frequency = (SYSclk) / 12 / (256-TH0) / 2

and if  $C\bar{T} = 1$ , namely Timer/Counter 0 count on the external pulse input from P3.4/T0,

the output frequency = (T0\_Pin\_CLK) / (256-TH0) / 2

0, P3.5/T1 is not configure for Timer 0 programmable clock output T0CLKO

---

B1 - T1CLKO : Whether is P3.4/T0 configured for Timer 1(T1) programmable clock output T1CLKO or not.

1, P3.4/T0 is configured for Timer1 programmable clock output T1CLKO, the clock output frequency = T1 overflow/2

If Timer/Counter 1 in mode 1 (16 bit auto-reloadable mode),

and if  $C/\overline{T} = 0$ , namely Timer/Counter 1 count on the internal system clock,

When T1 in 1T mode (AUXR.6/T1x12=1), the output frequency = (SYSclk)/(65536-[RL\_TH1, RL\_TL1])/2

When T1 in 12T mode (AUXR.6/T1x12=0), the output frequency = (SYSclk) / 12 / (65536-[RL\_TH1, RL\_TL1])/2

and if  $C/\overline{T} = 1$ , namely Timer/Counter 1 count on the external pulse input from P3.5/T1,

the output frequency = (T1\_Pin\_CLK) / (65536-[RL\_TH1, RL\_TL1])/2

If Timer/Counter 1 in mode 2 (8 bit auto-reloadable mode),

and if  $C/\overline{T} = 0$ , namely Timer/Counter 1 count on the internal system clock,

When T1 in 1T mode(AUXR.6/T1x12=1), the output frequency = (SYSclk) / (256-TH1) / 2

When T1 in 12T mode(AUXR.6/T1x12=0), the output frequency = (SYSclk) / 12 / (256-TH1) / 2

and if  $C/\overline{T} = 1$ , namely Timer/Counter 1 count on the external pulse input from P3.5/T1,

the output frequency = (T1\_Pin\_CLK) / (256-TH1) / 2

0, P3.4/T0 is not configure for Timer 1 programmable clock output T1CLKO

### 3. T2 Interrupt Enable bit : ET2

IE2: Interrupt Enable 2 Register (Non bit-addressable)

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

ET4 : Timer 4 interrupt enable bit.

If ET4 = 0, Timer 4 interrupt would be disabled.

If ET4 = 1, Timer 4 interrupt would be enabled.

ET3 : Timer 3 interrupt enable bit.

If ET3 = 0, Timer 3 interrupt would be disabled.

If ET3 = 1, Timer 3 interrupt would be enabled.

ES4 : Serial Port 4 (UART4) interrupt enable bit.

If ES4 = 0, UART4 interrupt would be disabled.

If ES4 = 1, UART4 interrupt would be enabled.

ES3 : Serial Port 3 (UART3) interrupt enable bit.

If ES3 = 0, UART3 interrupt would be disabled.

If ES3 = 1, UART3 interrupt would be enabled.

ET2 : Timer 2 interrupt enable bit.

If ET2 = 0, Timer 2 interrupt would be disabled.

If ET2 = 1, Timer 2 interrupt would be enabled.

ESPI: SPI interrupt enable bit.

If ESPI = 0, SPI interrupt would be disabled.

If ESPI = 1, SPI interrupt would be enabled.

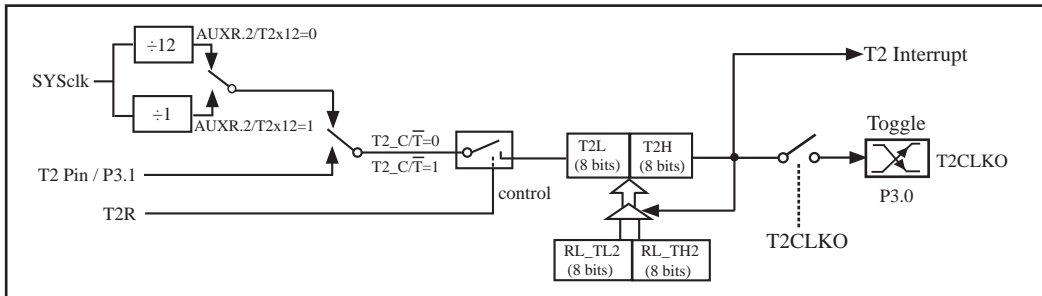
ES2 : Serial Port 2 (UART2) interrupt enable bit.

If ES2 = 0, UART2 interrupt would be disabled.

If ES2 = 1, UART2 interrupt would be enabled.

## 7.4.2 Timer/Counter 2 as 16-Bit Auto-Reload Timer/Counter

The schematic of Timer/Counter 2 is shown below :



Timer/Counter 2 mode : 16-bit auto-reload timer/counter

The counted input is enabled to the timer when  $T2R = 1$ .  $T2R/AUXR.4$  is a control bit in the Special Function Register AUXR.

If  $T2\_C\bar{T} / AUXR.3 = 0$ , Timer/Counter 2 would be set for Timer operation (input from internal system clock). However, if  $T2\_C\bar{T} / AUXR.3 = 1$ , Timer/Counter 2 would be set for Counter operation (input from external T2/P3.1 pin).

In the “Timer” function, the timer register [T2L, T2H] is incremented every 12 system clocks or every system clock depending on  $AUXR.2(T2x12)$  bit. If  $T2x12 = 0$ , the register [T2L, T2H] will be incremented every 12 system clocks. If  $T2x12 = 1$ , the register [T2L, T2H] will be incremented every system clock.

There are two hidden registers  $RL\_TH2$  and  $RL\_TL2$  for Timer/Counter 2. the address of  $RL\_TH2$  is the same as T2H's. And,  $RL\_TL2$  and T2L share in the same address. When  $T2R = 0$  disable Timer/Counter 2, the content written into register [T2L, T2H] will be written into [ $RL\_TL2$ ,  $RL\_TH2$ ] too. When  $T2R = 1$  enable Timer/Counter 2, the content written into register [T2L, T2H] actually don't be written into [T2L, T2H], but into [ $RL\_TL2$ ,  $RL\_TH2$ ]. When users read the content of [T2L, T2H], it is the content of [T2L, T2H] to read instead of [ $RL\_TL2$ ,  $RL\_TH2$ ].

The overflow from [T2L, T2H] will not only set the T2 interrupt request flag (which is invisible for users), but also reload [T2L, T2H] with the content of [ $RL\_TL2$ ,  $RL\_TH2$ ], which is preset by software. The reload leaves [ $RL\_TL2$ ,  $RL\_TH2$ ] unchanged.

---

## 7.4.2.1 Demo Program of 16-bit Auto-Reload Timer/Counter 2 (C and ASM)

### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of 16-bit auto-reload timer/counter 2 -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

//-----

/* define constants */
#define FOSC 18432000L

#define T38_4KHz (256-18432000/12/38400/2) //38.4KHz

/* define SFR */

sfr IE2 = 0xAF; // (IE2.2) timer2 interrupt control bit
sfr AUXR = 0x8E;
sfr T2H = 0xD6;
sfr T2L = 0xD7;

sbit TEST_PIN = P0^0; //test pin

//-----

/* Timer2 interrupt routine */
void t2_isr() interrupt 12 using 1
{
    TEST_PIN = !TEST_PIN;
}

//-----
```

---

```

/* main program */
void main()
{
    T2L    =    T38_4KHz;           //set timer2 reload value
    T2H    =    T38_4KH  >> 8;
    AUXR   |=    0x10;             //timer2 start run
    IE2    |=    0x04;             //enable timer2 interrupt
    EA     =    1;                 //open global interrupt switch

    while (1);                     //loop
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of 16-bit auto-reload timer/counter 2 -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

IE2    DATA    0AFH                //(IE2.2)timer2 interrupt control bit
AUXR   DATA    08EH                //Auxiliary register
T2H    DATA    0D6H
T2L    DATA    0D7H

F38_4KHz    EQU    0FF10H            //38.4KHz(1T mode, 65536-18432000/2/38400)

//-----

    ORG    0000H
    LJMP   MAIN

    ORG    0063H
    LJMP   T2INT

//-----

```

---

```

    ORG    0100H
MAIN:  MOV    SP,    #3FH

    ORL    AUXR,  #04H           //T2 in 1T mode

    MOV    T2L,   #LOW F38_4KHz  //set timer2 reload value
    MOV    T2H,   #HIGH F38_4KHz

    ORL    AUXR,  #10H           //T2 start to run

    ORL    IE2,   #04H           //enable T2 interrupt

    SETB   EA

    SJMP   $

//-----
//Timer2 interrupt routine

T2INT: CPL  P1.0

//    ANL  IE2,   #0FBH
//    ORL  IE2,   #04H

    RETI

;-----

    END

```

---

## 7.4.2.2 Demo Program using T2 to expand External Interrupt (Falling edge) —— T2 as 16-bit Auto-Reload Counter (C and ASM)

### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using T2 to extend external interrupt (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

//-----
sfr IE2 = 0xaf; //Interrupt enable register 2
sfr AUXR = 0x8e; //Auxiliary register
sfr T2H = 0xD6;
sfr T2L = 0xD7;

sbit P10 = P1^0;

//-----
//Timer 2 Interrupt Service Routine
void t2int() interrupt 12 //Timer 2 interrupt, location at 0063H
{
    P10 = !P10;

//    IE2 &= ~0x04;

//    IE2 |= 0x04;
}

void main()
{
    AUXR |= 0x04; //T2 in 1T mode
```

---

```

AUXR      |=      0x08;          //T2_C/T=1, T2(P3.1) as Clock Source
T2H  = T2L  =      0xff;        //Set the initial value of T2
AUXR      |=      0x10;        //start up T2

IE2   |=      0x04;            //Enable T2 interrupt

EA   =      1;

while (1);
}

```

## 2.Assembler Listing

```

/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using T2 to extend external interrupt (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

IE2   DATA  0AFH           //Interrupt enable register 2
AUXR  DATA  08EH           //Auxiliary register
T2H   DATA  0D6H
T2L   DATA  0D7H

//-----

        ORG    0000H
        LJMP   MAIN

        ORG    0063H           //Timer 2 interrupt, location at 0063H
        LJMP   T2INT

//-----

        ORG    0100H

```



---

MAIN:

```
MOV    SP,    #3FH

ORL    AUXR,  #04H           //T2 in 1T mode
ORL    AUXR,  #08H           //T2_C/T=1, T2(P3.1) as Clock Source

MOV    A,     #0FFH         //Set the initial value of T2
MOV    T2L,   A
MOV    T2H,   A

ORL    AUXR,  #10H         //start up T2

ORL    IE2,   #04H         //Enable T2 interrupt

SETB   EA

SJMP   $
```

```
//-----
//Timer 2 Interrupt Service Routine
```

T2INT:

```
CPL    P1.0

//     ANL    IE2,   #0FBH

//     ORL    IE2,   #04H

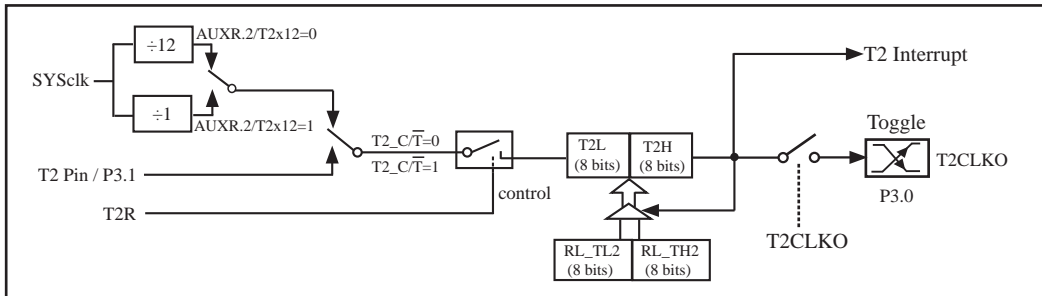
RETI
```

```
;-----
```

END

## 7.4.3 Timer/Counter 2 Programmable Clock Output and Demo Program

The schematic of Timer/Counter 2 is shown below :



Timer/Counter 2 mode : 16-bit auto-reload timer/counter

Besides as Timer/Counter, T2 also can be as the programmable clock output. The output clock frequency of T2CLKO is controlled by Timer 2. When it is used as programmable clock output, Timer 2 interrupt don't be enabled to avoid CPU entering interrupt repeatedly unless special circumstances.

The clock output of T2CLKO/P3.0 is controlled by the bit T2CLKO of register INT\_CLKO (AUXR2).

AUXR2.2 - T2CLKO :       1, enable clock output  
                              0, disable clock output

INT\_CLKO (AUXR2) (Address:8FH)

When T2CLKO/INT\_CLKO.2=1, P3.0 is configured for Timer 2 programmable clock output T2CLKO.

The clock output frequency =  $T2 \text{ overflow} / 2$

If  $T2\_C/\bar{T} = 0$ , namely Timer/Counter 2 count on the internal system clock,

When T2 in 1T mode (AUXR.2/T2x12=1), the output frequency =  $(SYSclk) / (65536 - [RL\_TH2, RL\_TL2]) / 2$

When T2 in 12T mode (AUXR.2/T2x12=0), the output frequency =  $(SYSclk) / 12 / (65536 - [RL\_TH2, RL\_TL2]) / 2$

If  $T2\_C/\bar{T} = 1$ , namely Timer/Counter 2 count on the external pulse input from P3.1/T2,

the output frequency =  $(T2\_Pin\_CLK) / (65536 - [RL\_TH2, RL\_TL2]) / 2$

RL\_TH2 is the reloaded register of T2H, RL\_TL2 is the reload register of T2L.

---

The following is the example program that Timer 2 output programmable clock by dividing the frequency of internal system clock or the clock input from external pin T2/P3.1 (C and assembly):

## 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of Timer 2 porgrammable clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

typedef unsigned char    BYTE;
typedef unsigned int    WORD;

#define FOSC 18432000L

//-----

sfr    AUXR        = 0x8e;
sfr    INT_CLKO    = 0x8f;
sfr    T2H         = 0xD6;
sfr    T2L         = 0xD7;

sbit   T2CLKO      = P3^0;

#define F38_4KHz    (65536-FOSC/2/38400)           //1T mode
//#define F38_4KHz    (65536-FOSC/2/12/38400)      //12T mode

//-----

void main()
{
    AUXR    |=    0x04;           //Timer 2 in 1T mode
//    AUXR    &=    ~0x04;       //Timer 2 in 12T mode
```

---

```

    AUXR  &=    ~0x08;           //T2_C/T=0, count on internal system clock
//  AUXR  |=    0x08;           //T2_C/T=1, count on external pulse input from T2(P3.1) pin
    T2L   =    F38_4KHz;           //Initial timing value
    T2H   =    F38_4KHz >> 8;

    AUXR  |=    0x10;
    INT_CLKO =    0x04;

    while (1);
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of Timer 2 porgrammable clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

```

```

//suppose the frequency of test chip is 18.432MHz

```

```

AUXR      DATA  08EH
INT_CLKO  DATA  08FH
T2H       DATA  0D6H
T2L       DATA  0D7H

```

```

T2CLKO    BIT    P3.0

```

```

F38_4KHz  EQU    0FF10H           //38.4KHz(1T mode, 65536-18432000/2/38400)
//F38_4KHz EQU    0FFECH           //38.4KHz(12T mode, (65536-18432000/2/12/38400)

```

```

//-----

```

---

```

    ORG    0000H
    LJMP   MAIN

//-----

    ORG    0100H
MAIN:   MOV    SP,    #3FH

    ORL    AUXR, #04H           //Timer 2 in 1T mode
//      ANL    AUXR, #0FBH       //Timer 2 in 12T mode

    ANL    AUXR, #0F7H         //T2_C/T=0, count on internal system clock
//      ORL    AUXR, #08H         //T2_C/T=1, count on external pulse input from T2(P3.1) pin

    MOV    T2L, #LOW F38_4KHz   //Initial timing value
    MOV    T2H, #HIGH F38_4KHz
    ORL    AUXR, #10H
    MOV    INT_CLKO, #04H

    SJMP   $

;-----

    END

```

---

---

## 7.4.4 Timer/Counter 2 as Baud-Rate Generator of Serial Port (UART)

Besides as Timer/Counter and programmable clock output, T2 also can be as the UART baud-rate generator. UART1 prefer to select Timer 2 as its baud-rate generator. UART2 only can choose Timer 2 as its its baud-rate generator. UART3 and UART4 default to selecting Timer 2 as their baud-rate generator.

When UART1 works in mode 1 (8-bit UART with variable baud-rate) and mode 3 (9-bit UART variable with baud-rate), its baud rate can be generated by T2. The Calculating Formula of buad-rate when UART1 select T2 as its baud-rate generator is shown below :

Baud-Rate of UART1 = (T2 overflow)/4. **Note: the bau-rate is independent of SMOD bit.**

If T2 works in 1T mode (AUXR.2/T2x12=1), the T2 overflow =  $\text{SYSclk} / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

So, Baud-Rate of UART1 =  $\text{SYSclk} / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$

If T2 works in 12T mode (AUXR.2/T2x12=0), the T2 overflow =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

So, Baud-Rate of UART1 =  $\text{SYSclk} / 12 / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$

UART2 only has two modes : mode 0 (8-bit UART variable with baud-rate) and mode 1 (9-bit UART variable with baud-rate). UART2 only can select Timer 2 as its baud-rate generator. The Calculating Formula of UART2 buad-rate is shown below :

Baud-Rate of UART2 = (T2 overflow)/4.

If T2 works in 1T mode (AUXR.2/T2x12=1), the T2 overflow =  $\text{SYSclk} / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

So, Baud-Rate of UART2 =  $\text{SYSclk} / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$

If T2 works in 12T mode (AUXR.2/T2x12=0), the T2 overflow =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

So, Baud-Rate of UART2 =  $\text{SYSclk} / 12 / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$

UART3 only has two modes : mode 0 (8-bit UART variable with baud-rate) and mode 1 (9-bit UART variable with baud-rate). UART3 either can select Timer 2 or Timer 3 as its baud-rate generator. It default to choosing Timer 2 as its baud-rate generator. The Calculating Formula of the buad-rate that UART3 select Timer 2 as its baud-rate generator is shown below :

Baud-Rate of UART3 = (T2 overflow)/4.

If T2 works in 1T mode (AUXR.2/T2x12=1), the T2 overflow =  $\text{SYSclk} / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

So, Baud-Rate of UART3 =  $\text{SYSclk} / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$

If T2 works in 12T mode (AUXR.2/T2x12=0), the T2 overflow =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

So, Baud-Rate of UART3 =  $\text{SYSclk} / 12 / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$

UART4 only has two modes : mode 0 (8-bit UART variable with baud-rate) and mode 1 (9-bit UART variable with baud-rate). UART4 either can select Timer 2 or Timer 4 as its baud-rate generator. It default to choosing Timer 2 as its baud-rate generator. The Calculating Formula of the buad-rate that UART4 select Timer 2 as its baud-rate generator is shown below :

Baud-Rate of UART4 = (T2 overflow)/4.

If T2 works in 1T mode (AUXR.2/T2x12=1), the T2 overflow =  $\text{SYSclk} / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

So, Baud-Rate of UART4 =  $\text{SYSclk} / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$

If T2 works in 12T mode (AUXR.2/T2x12=0), the T2 overflow =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

So, Baud-Rate of UART4 =  $\text{SYSclk} / 12 / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$

RL\_TH2 is the reloaded register of T2H, and RL\_TL2 is the reload register of T2L in above formula.

---

## 7.4.4.1 Demo Program using Timer/Counter 2 as UART1 Baud-Rate Generator

### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using Timer 2 as UART1 baud-rate generator -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

typedef unsigned char      BYTE;
typedef unsigned int      WORD;

#define FOSC  18432000L           //System frequency
#define BAUD  115200             //UART1 baud-rate

#define NONE_PARITY          0   //none parity
#define ODD_PARITY           1   //odd parity
#define EVEN_PARITY          2   //even parity
#define MARK_PARITY          3   //mark parity
#define SPACE_PARITY         4   //space parity

#define PARITYBIT EVEN_PARITY    //define the parity bit

sfr  AUXR  =  0x8e;             //Auxiliary register
sfr  T2H   =  0xd6;
sfr  T2L   =  0xd7;

sbit  P22  =  P2^2;

bit busy;

void SendData(BYTE dat);
void SendString(char *s);
void main()
{
#if (PARITYBIT == NONE_PARITY)
```

---

```

        SCON = 0x50; //8-bit variable baud-rate
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
        SCON = 0xda; //9-bit variable baud-rate,
                    //the parity bit is initialized for 1
#elif (PARITYBIT == SPACE_PARITY)
        SCON = 0xd2; //9-bit variable baud-rate,
                    //the parity bit is initialized for 0
#endif

        T2L = (65536 - (FOSC/4/BAUD)); //Set the preload value
        T2H = (65536 - (FOSC/4/BAUD))>>8;
        AUXR = 0x14; //T2 in 1T mode, and run T2
        AUXR |= 0x01; //select T2 as UART1 baud-rate generator
        ES = 1; //enable UART1 interrupt
        EA = 1;

        SendString("STC15W4K32S4\r\nUart Test !\r\n");
        while(1);
}

/*-----
UART Interrupt Service Routine
-----*/
void Uart() interrupt 4 using 1
{
    if (RI)
    {
        RI = 0; //clear RI
        P0 = SBUF; //serial data is shown in P0
        P22 = RB8; //P2.2 display the parity bit
    }
    if (TI)
    {
        TI = 0; //clear TI
        busy = 0; //clear busy flag
    }
}

/*-----
Send UART data
-----*/
void SendData(BYTE dat)
{
    while (busy); //wait to finish sending the previous data
    ACC = dat; //access to the parity bit ---- P (PSW.0)
    if (P)
    {
        #if (PARITYBIT == ODD_PARITY)

```

---



---

```

        TB8 = 0;                //the parity bit is set for 0
    #elif (PARITYBIT == EVEN_PARITY)
        TB8 = 1;                //the parity bit is set for 1
    #endif
    }
    else
    {
    #if (PARITYBIT == ODD_PARITY)
        TB8 = 1;                //the parity bit is set for 1
    #elif (PARITYBIT == EVEN_PARITY)
        TB8 = 0;                //the parity bit is set for 0
    #endif
    }
    busy = 1;
    SBUF = ACC;
}

/*-----
Send string
-----*/
void SendString(char *s)
{
    while (*s)
    {
        SendData(*s++);
    }
}

```

---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using Timer 2 as UART1 baud-rate generator -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#define NONE_PARITY 0 //none parity
#define ODD_PARITY 1 //odd parity
#define EVEN_PARITY 2 //even parity
#define MARK_PARITY 3 //mark parity
#define SPACE_PARITY 4 //space parity

#define PARITYBIT EVEN_PARITY //define the parity bit

//-----

AUXR EQU 08EH //Auxiliary register
T2H DATA 0D6H
T2L DATA 0D7H

//-----
BUSY BIT 20H.0
//-----
ORG 0000H
LJMP MAIN

ORG 0023H
LJMP UART_ISR
//-----
ORG 0100H
MAIN:
CLR BUSY
CLR EA
MOV SP, #3FH

#if (PARITYBIT == NONE_PARITY)
MOV SCON, #50H //8-bit variable baud-rate
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
```

```

MOV     SCON, #0DAH                                //9-bit variable baud-rate
                                                //the parity bit is initialized for 1
#elif (PARITYBIT == SPACE_PARITY)
MOV     SCON, #0D2H                                //9-bit variable baud-rate
                                                //the parity bit is initialized for 0
#endif
//-----
MOV     T2L,  #0D8H                                //Set the preload value (65536-18432000/4/115200)
MOV     T2H,  #0FFH
MOV     AUXR, #14H                                //T2 in 1T mode, and run T2
ORL     AUXR, #01H                                //select T2 as UART1 baud-rate generator
SETB    ES                                         //enable UART1 interrupt
SETB    EA

MOV     DPTR, #TESTSTR
LCALL   SENDSTRING

SJMP   $

;-----
TESTSTR:
DB "STC15W4K32S4 Uart1 Test !",0DH,0AH,0

;/*-----
;UART Interrupt Service Routine
;-----*/
UART_ISR:
PUSH   ACC
PUSH   PSW
JNB    RI,    CHECKTI
CLR    RI                                           //clear RI
MOV    P0,    SBUF                                  //serial data is shown in P0
MOV    C,     RB8
MOV    P2.2,  C                                    //P2.2 display the parity bit
CHECKTI:
JNB    TI,    ISR_EXIT
CLR    TI                                           //clear TI
CLR    BUSY                                        //clear busy flag
ISR_EXIT:
POP    PSW
POP    ACC
RETI

;/*-----
;Send UART data
;-----*/
SENDDATA:
JB     BUSY,  $                                    //wait to finish sending the previous data
MOV    ACC,  A                                    //access to the parity bit ---- P (PSW.0)
JNB    P,     EVEN1INACC

```

---

```

ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
    CLR    TB8                //the parity bit is set for 0
#elseif (PARITYBIT == EVEN_PARITY)
    SETB   TB8                //the parity bit is set for 1
#endif
    SJMP   PARITYBITOK
EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
    SETB   TB8                //the parity bit is set for 1
#elseif (PARITYBIT == EVEN_PARITY)
    CLR    TB8                //the parity bit is set for 0
#endif
PARITYBITOK:
    SETB   BUSY
    MOV    SBUF, A
    RET

; /*-----
; Send string
; -----*/
SENDSTRING:
    CLR    A
    MOVC   A, @A+DPTR
    JZ     STRINGEND
    INC    DPTR
    LCALL  SENDDATA
    SJMP   SENDSTRING
STRINGEND:
    RET
//-----
    END

```

---

## 7.4.4.2 Demo Program using Timer/Counter 2 as UART2 Baud-Rate Generator

### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using Timer 2 as UART2 baud-rate generator -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

typedef unsigned char    BYTE;
typedef unsigned int     WORD;

#define FOSC    18432000L           //System frequency
#define BAUD    115200             //UART2 baud-rate
#define TM      (65536 - (FOSC/4/BAUD))

#define NONE_PARITY    0           //none parity
#define ODD_PARITY     1           //odd parity
#define EVEN_PARITY    2           //even parity
#define MARK_PARITY    3           //mark parity
#define SPACE_PARITY   4           //space parity

#define PARITYBIT EVEN_PARITY      //define the parity bit

sfr    AUXR    =    0x8e;          //Auxiliary register
sfr    S2CON   =    0x9a;          //UART2 Control register
sfr    S2BUF   =    0x9b;          //UART2 data register
sfr    T2H     =    0xd6;
sfr    T2L     =    0xd7;
sfr    IE2     =    0xaf;          //Interrupt Enable register 2

#define S2RI    0x01              //S2CON.0
#define S2TI    0x02              //S2CON.1
```

---

```

#define S2RB8 0x04 //S2CON.2
#define S2TB8 0x08 //S2CON.3

bit busy;

void SendData(BYTE dat);
void SendString(char *s);

void main()
{
#if (PARITYBIT == NONE_PARITY)
    S2CON = 0x50; //8-bit variable baud-rate
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    S2CON = 0xda; //9-bit variable baud-rate
//the parity bit is initialized for 1
#elif (PARITYBIT == SPACE_PARITY)
    S2CON = 0xd2; //9-bit variable baud-rate
//the parity bit is initialized for 0
#endif

    T2L = TM; //Set the preload value
    T2H = TM>>8;
    AUXR = 0x14; //T2 in 1T mode, and run T2
    IE2 = 0x01; //enable UART2 interrupt
    EA = 1;

    SendString("STC15W4K32S4\r\nUart2 Test !\r\n");
    while(1);
}

/*-----
UART2 Interrupt Service Routine
-----*/
void Uart2() interrupt 8 using 1
{
    if (S2CON & S2RI)
    {
        S2CON &= ~S2RI; //clear S2RI
        P0 = S2BUF; //serial data is shown in P0
        P2 = (S2CON & S2RB8); //P2.2 display the parity bit
    }
    if (S2CON & S2TI)
    {
        S2CON &= ~S2TI; //clear S2TI
        busy = 0; //clear busy flag
    }
}

```

---

---

```

/*-----
Send UART data
-----*/
void SendData(BYTE dat)
{
    while (busy);           //wait to finish sending the previous data
    ACC = dat;              //access to the parity bit ---- P (PSW.0)
    if (P)
    {
        #if (PARITYBIT == ODD_PARITY)
            S2CON &= ~S2TB8;           //the parity bit is set for 0
        #elif (PARITYBIT == EVEN_PARITY)
            S2CON |= S2TB8;           //the parity bit is set for 1
        #endif
    }
    else
    {
        #if (PARITYBIT == ODD_PARITY)
            S2CON |= S2TB8;           //the parity bit is set for 1
        #elif (PARITYBIT == EVEN_PARITY)
            S2CON &= ~S2TB8;         //the parity bit is set for 0
        #endif
    }
    busy = 1;
    S2BUF = ACC;
}

/*-----
Send sting
-----*/
void SendString(char *s)
{
    while (*s)
    {
        SendData(*s++);
    }
}

```

---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using Timer 2 as UART2 baud-rate generator -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz
#define NONE_PARITY      0           //none parity
#define ODD_PARITY      1           //odd parity
#define EVEN_PARITY     2           //even parity
#define MARK_PARITY     3           //mark parity
#define SPACE_PARITY    4           //space parity

#define PARITYBIT EVEN_PARITY      //define the parity bit

//-----

AUXR EQU 08EH           //Auxiliary register
S2CON EQU 09AH          //UART2 Control register
S2BUF EQU 09BH          //UART2 data register
T2H DATA 0D6H
T2L DATA 0D7H
IE2 EQU 0AFH           //Interrupt Enable register 2

S2RI EQU 01H           //S2CON.0
S2TI EQU 02H           //S2CON.1
S2RB8 EQU 04H          //S2CON.2
S2TB8 EQU 08H          //S2CON.3

//-----
BUSY BIT 20H.0
//-----

ORG 0000H
LJMP MAIN

ORG 0043H
LJMP UART2_ISR
```



```

//-----
        ORG     0100H
MAIN:
        CLR     BUSY
        CLR     EA
        MOV     SP,     #3FH
#if (PARITYBIT == NONE_PARITY)
        MOV     S2CON, #50H                //8-bit variable baud-rate
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
        MOV     S2CON, #0DAH                //9-bit variable baud-rate
                                           //the parity bit is initialized for 1
#elif (PARITYBIT == SPACE_PARITY)
        MOV     S2CON, #0D2H                //9-bit variable baud-rate
                                           //the parity bit is initialized for 0
#endif

//-----
        MOV     T2L,   #0D8H                //Set the preload value (65536-18432000/4/115200)
        MOV     T2H,   #0FFH
        MOV     AUXR,  #14H                //T2 in 1T mode, and run T2
        ORL     IE2,   #01H                //enable UART2 interrupt
        SETB    EA

        MOV     DPTR,  #TESTSTR
        LCALL   SENDSTRING

        SJMP    $
;-----
TESTSTR:
        DB     "STC15W4K32S4 Uart2 Test !",0DH,0AH,0

; /*-----
;UART2 Interrupt Service Routine
;-----*/
UART2_ISR:
        PUSH    ACC
        PUSH    PSW
        MOV     A,     S2CON                ;read the content of S2CON
        JNB    ACC.0,  CHECKTI
        ANL    S2CON, #NOT S2RI            ;clear S2RI
        MOV     P0,    S2BUF                ;serial data is shown in P0
        ANL    A,     #S2RB8                ;
        MOV     P2,    A                    ;P2.2 display the parity bit
CHECKTI:
        MOV     A,     S2CON                ;read the content of S2CON
        JNB    ACC.1,  ISR_EXIT
        ANL    S2CON, #NOT S2TI            ;clear S2RI
        CLR     BUSY                        ;clear busy flag

```

---

```

ISR_EXIT:
    POP    PSW
    POP    ACC
    RETI

;/*-----
;Send UART data
;-----*/
SENDDATA:
    JB     BUSY, $           //wait to finish sending the previous data
    MOV    ACC, A           //access to the parity bit ---- P (PSW.0)
    JNB    P, EVEN1INACC
ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
    ANL    S2CON, #NOT S2TB8 //the parity bit is set for 0
#elif (PARITYBIT == EVEN_PARITY)
    ORL    S2CON, #S2TB8    //the parity bit is set for 1
#endif
    SJMP   PARITYBITOK
EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
    ORL    S2CON, #S2TB8    //the parity bit is set for 1
#elif (PARITYBIT == EVEN_PARITY)
    ANL    S2CON, #NOT S2TB8 //the parity bit is set for 0
#endif
PARITYBITOK:
    SETB   BUSY
    MOV    S2BUF, A
    RET

;/*-----
;Send sting
//-----*/
SENDSTRING:
    CLR    A
    MOVC   A, @A+DPTR
    JZ     STRINGEND
    INC    DPTR
    LCALL  SENDDATA
    SJMP   SENDSTRING
STRINGEND:
    RET
//-----
    END

```

---

## 7.5 Timer/Counter 3 and Timer/Counter 4

Another two 16-bit timers/counters also are added to STC15W4K32S4 series MCU : Timer/Counter 3 and Timer/Counter 4. Just like T2, T3 and T4 all only have one mode : 16-bit auto-reload timer/counter. Besides as Timer/Counter, T3 and T4 also can be as the baud-rate generator and programmable clock output.

### 7.5.1 Special Function Registers about Timer/Counter 3 and 4

Symbol	Description	Address	Bit Address and Symbol								Value after Power-on or Reset
			MSB				LSB				
T4T3M	T4 and T3 Control and Mode register	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000 0000B
T4H	The high 8-bit of Timer 4 register	D2H									0000 0000B
T4L	The low 8-bit of Timer 4 register	D3H									0000 0000B
T3H	The high 8-bit of Timer 3 register	D4H									0000 0000B
T3L	The low 8-bit of Timer 3 register	D5H									0000 0000B
IE2	Interrupt Enable register	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000 0000B

#### 1. T4T3M : Timer 4 and Timer 3 Mode register (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

B7 - T4R Timer 4 Run control bit

- 0 : not run Timer 4;
- 1 : run Timer 4.

B6 - T4\_C/T: Counter or timer 4 selector

- 0 : as Timer (namely count on internal system clock)
- 1 : as Counter (namely count on the external pulse input from T4/P0.7)

B5 - T4x12 : Timer 4 clock source bit.

- 0 : The clock source of Timer 4 is SYSclk/12.
- 1 : The clock source of Timer 4 is SYSclk/1.

B4 - T4CLKO : Whether is P0.6 configured for Timer 4(T4) programmable clock output T4CLKO or not.

1, P0.6 is configured for Timer 4 programmable clock output T4CLKO, the clock output frequency = T4 overflow/2

If T4\_C/T = 0, namely Timer/Counter 4 count on the internal system clock,

When T4 in 1T mode (T4T3.5/T4x12=1), the output frequency = (SYSclk)/(65536-[RL\_TH4, RL\_TL4])/2

When T4 in 12T mode (T4T3.5/T4x12=0), the output frequency = (SYSclk) /12/ (65536-[RL\_TH4, RL\_TL4])/2

If T4\_C/T = 1, namely Timer/Counter 4 count on the external pulse input from P0.7/T4,

the output frequency = (T4\_Pin\_CLK) / (65536-[RL\_TH4, RL\_TL4])/2

0, P0.6 is not configure for Timer 4 programmable clock output T4CLKO

---

B3 - T3R Timer 3 Run control bit

- 0 : not run Timer 3;
- 1 : run Timer 3.

B2 - T3\_C/ $\bar{T}$ : Counter or timer 3 selector

- 0 : as Timer (namely count on internal system clock)
- 1 : as Counter (namely count on the external pulse input from T3/P0.5)

B1 - T3x12 : Timer 3 clock source bit.

- 0 : The clock source of Timer 3 is SYSclk/12.
- 1 : The clock source of Timer 3 is SYSclk/1.

B0 - T3CLKO : Whether is P0.4 configured for Timer 3(T3) programmable clock output T3CLKO or not.

1, P0.4 is configured for Timer 3 programmable clock output T3CLKO, the clock output frequency = T3 overflow / 2

If T3\_C/ $\bar{T}$  = 0, namely Timer/Counter 3 count on the internal system clock,

When T3 in 1T mode (T4T3.1/T3x12=1), the output frequency = (SYSclk)/(65536-[RL\_TH3, RL\_TL3])/2

When T3 in 12T mode (T4T3.1/T3x12=0), the output frequency = (SYSclk) /12/ (65536-[RL\_TH3, RL\_TL3])/2

If T3\_C/ $\bar{T}$  = 1, namely Timer/Counter 3 count on the external pulse input from P0.5/T3,

the output frequency = (T3\_Pin\_CLK) / (65536-[RL\_TH3, RL\_TL3])/2

0, P0.4 is not configure for Timer 3 programmable clock output T3CLKO

## 2. T3 and T4 Interrupt Enable Register : IE2

IE2: Interrupt Enable 2 Register (Non bit-addressable)

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

ET4 : Timer 4 interrupt enable bit.

If ET4 = 0, Timer 4 interrupt would be disabled.

If ET4 = 1, Timer 4 interrupt would be enabled.

ET3 : Timer 3 interrupt enable bit.

If ET3 = 0, Timer 3 interrupt would be disabled.

If ET3 = 1, Timer 3 interrupt would be enabled.

ES4 : Serial Port 4 (UART4) interrupt enable bit.

If ES4 = 0, UART4 interrupt would be disabled.

If ES4 = 1, UART4 interrupt would be enabled.

ES3 : Serial Port 3 (UART3) interrupt enable bit.

If ES3 = 0, UART3 interrupt would be disabled.

If ES3 = 1, UART3 interrupt would be enabled.

ET2 : Timer 2 interrupt enable bit.

If ET2 = 0, Timer 2 interrupt would be disabled.

If ET2 = 1, Timer 2 interrupt would be enabled.

ESPI: SPI interrupt enable bit.

If ESPI = 0, SPI interrupt would be disabled.

If ESPI = 1, SPI interrupt would be enabled.

ES2 : Serial Port 2 (UART2) interrupt enable bit.

If ES2 = 0, UART2 interrupt would be disabled.

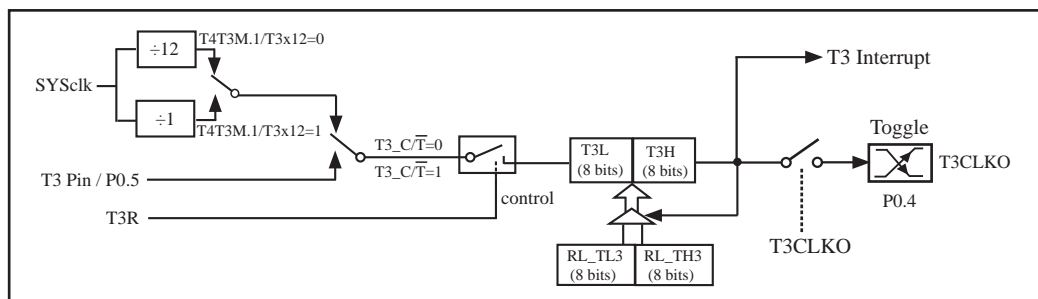
If ES2 = 1, UART2 interrupt would be enabled.

## 7.5.2 Timer/Counter 3

T3 only has one mode : 16-bit auto-reload timer/counter. T3 either can be as Timer/Counter or as the baud-rate generator or programmable clock output.

### 7.5.2.1 Timer/Counter 3 as 16-Bit Auto-Reload Timer/Counter

The schematic of Timer/Counter 3 is shown below :



Timer/Counter 3 mode : 16-bit auto-reload timer/counter

The counted input is enabled to the timer when  $T3R = 1$ .  $T3R/T4T3M.3$  is a control bit in the Special Function Register  $T4T3M$ .

If  $T3\_C/\overline{T} / T4T3M.2 = 0$ , Timer/Counter 3 would be set for Timer operation (input from internal system clock). However, if  $T3\_C/\overline{T} / T4T3M.2 = 1$ , Timer/Counter 3 would be set for Counter operation (input from external T3/P0.5 pin).

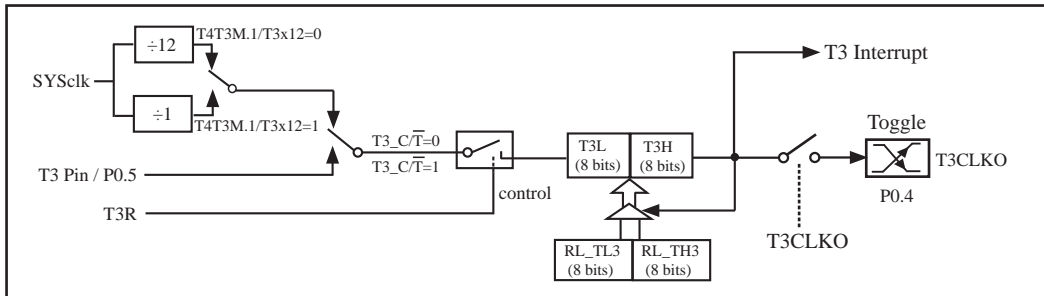
In the “Timer” function, the timer register [T3L, T3H] is incremented every 12 system clocks or every system clock depending on  $T4T3M.1(T3x12)$  bit. If  $T3x12 = 0$ , the register [T3L, T3H] will be incremented every 12 system clocks. If  $T3x12 = 1$ , the register [T3L, T3H] will be incremented every system clock.

There are two hidden registers  $RL\_TH3$  and  $RL\_TL3$  for Timer/Counter 3. the address of  $RL\_TH3$  is the same as  $T3H$ 's. And,  $RL\_TL3$  and  $T3L$  share in the same address. When  $T3R = 0$  disable Timer/Counter 3, the content written into register [T3L, T3H] will be written into [RL\_TL3, RL\_TH3] too. When  $T3R = 1$  enable Timer/Counter 3, the content written into register [T3L, T3H] actually don not be written into [T3L, T3H], but into [RL\_TL3, RL\_TH3]. When users read the content of [T3L, T3H], it is the content of [T3L, T3H] to read instead of [RL\_TL3, RL\_TH3].

The overflow from [T3L, T3H] will not only set the T3 interrupt request flag (which is invisible for users), but also reload [T3L, T3H] with the content of [RL\_TL3, RL\_TH3], which is preset by software. The reload leaves [RL\_TL3, RL\_TH3] unchanged.

## 7.5.2.2 Timer/Counter 3 Programmable Clock Output

The schematic of Timer/Counter 3 is shown below :



Timer/Counter 3 mode : 16-bit auto-reload timer/counter

Besides as Timer/Counter, T3 also can be as the programmable clock output. The output clock frequency of T3CLKO is controlled by Timer 3. When it is used as programmable clock output, Timer 3 interrupt don't be enabled to avoid CPU entering interrupt repeatedly unless special circumstances.

The clock output of T3CLKO/P0.4 is controlled by the bit T3CLKO of register T4T3M.

T4T3M.0 - T3CLKO :    1, enable clock output  
                          0, disable clock output

T4T3M(Address:D1H)

When T3CLKO/T4T3M.0=1, P0.4 is configured for Timer 3 programmable clock output T3CLKO.

The clock output frequency =  $T3 \text{ overflow} / 2$

If  $T3\_C/\bar{T} = 0$ , namely Timer/Counter 3 count on the internal system clock,

When T3 in 1T mode ( $T4T3.1/T3x12=1$ ), the output frequency =  $(SYSclk)/(65536-[RL\_TH3, RL\_TL3])/2$

When T3 in 12T mode ( $T4T3.1/T3x12=0$ ), the output frequency =  $(SYSclk) / 12 / (65536-[RL\_TH3, RL\_TL3])/2$

If  $T3\_C/\bar{T} = 1$ , namely Timer/Counter 3 count on the external pulse input from P0.5/T3,

the output frequency =  $(T3\_Pin\_CLK) / (65536-[RL\_TH3, RL\_TL3])/2$

RL\_TH3 is the reloaded register of T3H, RL\_TL3 is the reload register of T3L.

---

### 7.5.2.3 Timer/Counter 3 as Baud-Rate Generator of Serial Port 3 (UART3)

Besides as Timer/Counter and programmable clock output, T3 also can be as the UART3 baud-rate generator. UART3 defaults to selecting Timer 2 as their baud-rate generator. But it also can select Timer 3 as its baud-rate generator by setting S3ST3/S3CON.6.

S3CON : Serial Port 3 Control Register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S3CON	ACH	name	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI

S3ST3 : the control bit whether UART3 choose T3 as its baud-rate generator or not.

0, Choose T2 as UART3 baud-rate generator

1, Choose T3 as UART3 baud-rate generator

UART3 only has two modes : mode 0 (8-bit UART variable with baud-rate) and mode 1 (9-bit UART variable with baud-rate). UART3 either can select Timer 2 or Timer 3 as its baud-rate generator. When UART3 select Timer 3 as its baud-rate generator, the **Calculating Formula is shown below** :

Baud-Rate of UART3 = (T3 overflow)/4.

If T3 works in 1T mode (T4T3M.1/T3x12=1), the T3 overflow =  $\text{SYSclk} / (65536 - [\text{RL\_TH3}, \text{RL\_TL3}])$  ;

So, Baud-Rate of UART3 =  $\text{SYSclk} / (65536 - [[\text{RL\_TH3}, \text{RL\_TL3}]] / 4$

If T3 works in 12T mode (T4T3M.1/T3x12=0), the T3 overflow =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH3}, \text{RL\_TL3}])$  ;

So, Baud-Rate of UART3 =  $\text{SYSclk} / 12 / (65536 - [[\text{RL\_TH3}, \text{RL\_TL3}]] / 4$

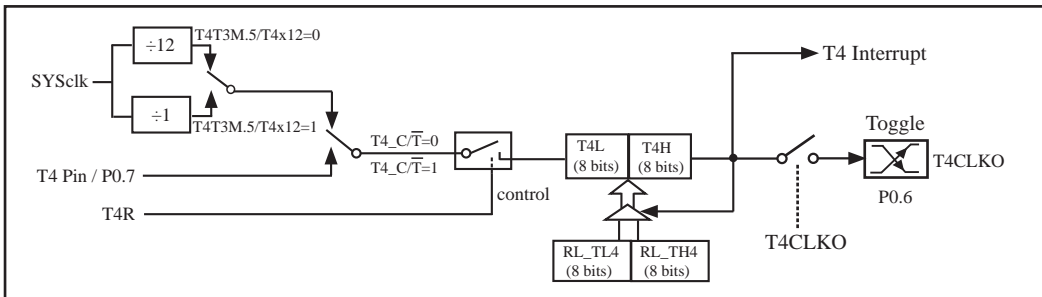
RL\_TH3 is the reloaded register of T3H, and RL\_TL3 is the reload register of T3L in above formula.

## 7.5.3 Timer/Counter 4

T4 only has one mode : 16-bit auto-reload timer/counter. T4 either can be as Timer/Counter or as the baud-rate generator or programmable clock output.

### 7.5.3.1 Timer/Counter 4 as 16-Bit Auto-Reload Timer/Counter

The schematic of Timer/Counter 4 is shown below :



Timer/Counter 4 mode : 16-bit auto-reload timer/counter

The counted input is enabled to the timer when  $T4R = 1$ .  $T4R/T4T3M.7$  is a control bit in the Special Function Register  $T4T3M$ .

If  $T4\_C/\overline{T} / T4T3M.6 = 0$ , Timer/Counter 4 would be set for Timer operation (input from internal system clock). However, if  $T4\_C/\overline{T} / T4T3M.6 = 1$ , Timer/Counter 4 would be set for Counter operation (input from external  $T4/P0.7$  pin).

In the “Timer” function, the timer register  $[T4L, T4H]$  is incremented every 12 system clocks or every system clock depending on  $T4T3M.5$  ( $T4x12$ ) bit. If  $T4x12 = 0$ , the register  $[T4L, T4H]$  will be incremented every 12 system clocks. If  $T4x12 = 1$ , the register  $[T4L, T4H]$  will be incremented every system clock.

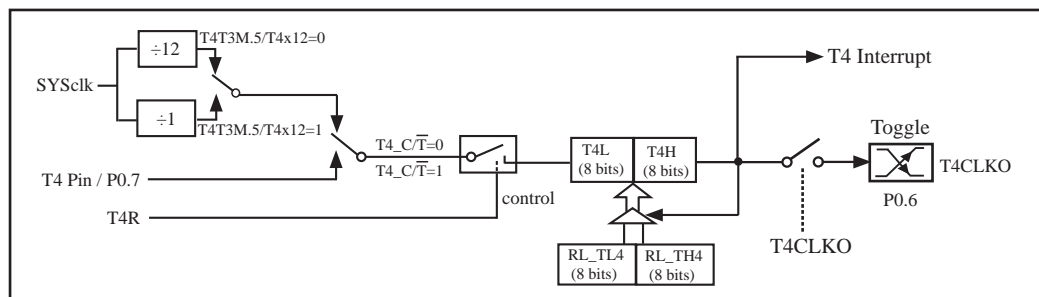
There are two hidden registers  $RL\_TH4$  and  $RL\_TL4$  for Timer/Counter 4. the address of  $RL\_TH4$  is the same as  $T4H$ 's. And,  $RL\_TL4$  and  $T4L$  share in the same address. When  $T4R = 0$  disable Timer/Counter 3, the content written into register  $[T4L, T4H]$  will be written into  $[RL\_TL4, RL\_TH4]$  too. When  $T4R = 1$  enable Timer/Counter 4, the content written into register  $[T4L, T4H]$  actually do not be written into  $[T4L, T4H]$ , but into  $[RL\_TL4, RL\_TH4]$ . When users read the content of  $[T4L, T4H]$ , it is the content of  $[T4L, T4H]$  to read instead of  $[RL\_TL4, RL\_TH4]$ .

The overflow from  $[T4L, T4H]$  will not only set the T4 interrupt request flag (which is invisible for users), but also reload  $[T4L, T4H]$  with the content of  $[RL\_TL4, RL\_TH4]$ , which is preset by software. The reload leaves  $[RL\_TL4, RL\_TH4]$  unchanged.



### 7.5.3.2 Timer/Counter 4 Programmable Clock Output

The schematic of Timer/Counter 4 is shown below :



Timer/Counter 4 mode : 16-bit auto-reload timer/counter

Besides as Timer/Counter, T4 also can be as the programmable clock output. The output clock frequency of T4CLKO is controlled by Timer 4. When it is used as programmable clock output, Timer 4 interrupt don't be enabled to avoid CPU entering interrupt repeatedly unless special circumstances.

The clock output of T4CLKO/P0.6 is controlled by the bit T4CLKO of register T4T3M.

T4T3M.4 - T4CLKO :     1, enable clock output  
                          0, disable clock output

T4T3M(Address:D1H)

When T4CLKO/T4T3M.4=1, P0.6 is configured for Timer 4 programmable clock output T4CLKO.

The clock output frequency =  $T4 \text{ overflow} / 2$

If  $T4\_C/\bar{T} = 0$ , namely Timer/Counter 4 count on the internal system clock,

When T4 in 1T mode ( $T4T3.5/T4x12=1$ ), the output frequency =  $(SYSclk)/(65536-[RL\_TH4, RL\_TL4])/2$

When T4 in 12T mode ( $T4T3.5/T4x12=0$ ), the output frequency =  $(SYSclk) / 12 / (65536-[RL\_TH4, RL\_TL4])/2$

If  $T4\_C/\bar{T} = 1$ , namely Timer/Counter 4 count on the external pulse input from P0.7/T4,

the output frequency =  $(T4\_Pin\_CLK) / (65536-[RL\_TH4, RL\_TL4])/2$

RL\_TH4 is the reloaded register of T4H, RL\_TL4 is the reload register of T4L.

---

### 7.5.3.3 Timer/Counter 4 as Baud-Rate Generator of Serial Port 4 (UART4)

Besides as Timer/Counter and programmable clock output, T4 also can be as the UART4 baud-rate generator. UART4 defaults to selecting Timer 2 as their baud-rate generator. But it also can select Timer 4 as its baud-rate generator by setting S4ST4/S4CON.6.

S4CON : Serial Port 4 Control Register

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S4CON	84H	name	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

S4ST4 : the control bit whether UART4 choose T4 as its baud-rate generator or not.

0, Choose T2 as UART4 baud-rate generator

1, Choose T4 as UART4 baud-rate generator

UART4 only has two modes : mode 0 (8-bit UART variable with baud-rate) and mode 1 (9-bit UART variable with baud-rate). UART4 either can select Timer 2 or Timer 4 as its baud-rate generator. When UART4 select Timer 4 as its baud-rate generator, the **Calculating Formula is shown below** :

Baud-Rate of UART4 = (T4 overflow)/4.

If T4 works in 1T mode (T4T3M.5/T4x12=1), the T4 overflow =  $\text{SYSclk} / (65536 - [\text{RL\_TH4}, \text{RL\_TL4}])$  ;

So, Baud-Rate of UART4 =  $\text{SYSclk} / (65536 - [[\text{RL\_TH4}, \text{RL\_TL4}]] / 4$

If T4 works in 12T mode (T4T3M.5/T4x12=0), the T4 overflow =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH4}, \text{RL\_TL4}])$  ;

So, Baud-Rate of UART4 =  $\text{SYSclk} / 12 / (65536 - [[\text{RL\_TH4}, \text{RL\_TL4}]] / 4$

RL\_TH4 is the reloaded register of T4H, and RL\_TL4 is the reload register of T4L in above formula.

---

## 7.6 How to Increase T0/T1/T2/T3/T4 Speed by 12 times

### 1. The speed control bits of T0/T1/T2 : T0x12 / T1x12 / T2x12

AUXR: Auxiliary register (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C $\bar{T}$	T2x12	EXTRAM	S1ST2

B7 - T0x12 : Timer 0 clock source bit.

- 0 : The clock source of Timer 0 is SYSclk/12. It will compatible to the traditional 8051 MCU
- 1 : The clock source of Timer 0 is SYSclk/1. It will drive the T0 faster than a traditional 8051 MCU

B6 - T1x12 : Timer 1 clock source bit.

- 0 : The clock source of Timer 1 is SYSclk/12. It will compatible to the traditional 8051 MCU
- 1 : The clock source of Timer 1 is SYSclk/1. It will drive the T0 faster than a traditional 8051 MCU

If T1 is used as the baud-rate generator of UART1, T1x12 will decide whether UART1 is 1T or 12T.

B2 - T2x12 : Timer 2 clock source bit.

- 0 : The clock source of Timer 2 is SYSclk/12.
- 1 : The clock source of Timer 2 is SYSclk/1.

If T2 is used as the baud-rate generator of UART1 or UART2, T1x12 will decide whether UART1 or UART2 is 1T or 12T.

B5 - UART\_M0x6 : Baud rate select bit of UART1 while it is working under 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.

B4 - T2R Timer 2 Run control bit

- 0 : not run Timer 2;
- 1 : run Timer 2.

B3 - T2\_C $\bar{T}$ : Counter or timer 2 selector

- 0 : as Timer (namely count on internal system clock)
- 1 : as Counter (namely count on the external pulse input from T2/P3.1)

B1 - EXTRAM : Internal / external RAM access control bit.

- 0 : On-chip auxiliary RAM is enabled.
- 1 : On-chip auxiliary RAM is always disabled.

B0 - S1ST2 : the control bit that UART1 select Timer 2 as its baud-rate generator.

- 0 : Select Timer 1 as the baud-rate generator of UART1
- 1 : Select Timer 2 as the baud-rate generator of UART1. Timer 1 is released to use in other functions.

## 2. The speed control bits of T4/T3 : T4x12 / T3x12

**T4T3M : Timer 4 and Timer 3 Mode register** (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

**B5 - T4x12 : Timer 4 clock source bit.**

- 0 : The clock source of Timer 4 is SYSclk/12.
- 1 : The clock source of Timer 4 is SYSclk/1.

**B1 - T3x12 : Timer 3 clock source bit.**

- 0 : The clock source of Timer 3 is SYSclk/12.
- 1 : The clock source of Timer 3 is SYSclk/1.

**B7 - T4R** Timer 4 Run control bit

- 0 : not run Timer 4;
- 1 : run Timer 4.

**B6 - T4\_C/T**: Counter or timer 4 selector

- 0 : as Timer (namely count on internal system clock)
- 1 : as Counter (namely count on the external pulse input from T4/P0.7)

**B4 - T4CLKO** : Whether is P0.6 configured for Timer 4(T4) programmable clock output T4CLKO or not.

1, P0.6 is configured for Timer 4 programmable clock output T4CLKO, the clock output frequency = T4 overflow/2

If T4\_C/T = 0, namely Timer/Counter 4 count on the internal system clock,

When T4 in 1T mode (T4T3.5/T4x12=1), the output frequency = (SYSclk)/(65536-[RL\_TH4, RL\_TL4])/2

When T4 in 12T mode (T4T3.5/T4x12=0), the output frequency = (SYSclk) /12/ (65536-[RL\_TH4, RL\_TL4])/2

If T4\_C/T = 1, namely Timer/Counter 4 count on the external pulse input from P0.7/T4,

the output frequency = (T4\_Pin\_CLK) / (65536-[RL\_TH4, RL\_TL4])/2

0, P0.6 is not configure for Timer 4 programmable clock output T4CLKO

**B3 - T3R** Timer 3 Run control bit

- 0 : not run Timer 3;
- 1 : run Timer 3.

**B2 - T3\_C/T**: Counter or timer 3 selector

- 0 : as Timer (namely count on internal system clock)
- 1 : as Counter (namely count on the external pulse input from T3/P0.5)

**B0 - T3CLKO** : Whether is P0.4 configured for Timer 3(T3) programmable clock output T3CLKO or not.

1, P0.4 is configured for Timer 3 programmable clock output T3CLKO, the clock output frequency = T3 overflow / 2

If T3\_C/T = 0, namely Timer/Counter 3 count on the internal system clock,

When T3 in 1T mode (T4T3.1/T3x12=1), the output frequency = (SYSclk)/(65536-[RL\_TH3, RL\_TL3])/2

When T3 in 12T mode (T4T3.1/T3x12=0), the output frequency = (SYSclk) /12/ (65536-[RL\_TH3, RL\_TL3])/2

If T3\_C/T = 1, namely Timer/Counter 3 count on the external pulse input from P0.5/T3,

the output frequency = (T3\_Pin\_CLK) / (65536-[RL\_TH3, RL\_TL3])/2

0, P0.4 is not configure for Timer 3 programmable clock output T3CLKO

## 7.7 Programmable Clock Output (or as Frequency Divider)

STC15W4K32S4 series MCU has six channel programmable clock outputs. They are Master clock output MCLKO/P5.4, Timer 0 programmable clock output T0CLKO/P3.5, Timer 1 programmable clock output T1CLKO/P3.4, Timer 2 programmable clock output T2CLKO/P3.0, Timer 3 programmable clock output T3CLKO/P0.4, Timer 4 programmable clock output T4CLKO/P0.6. The speed of external programmable clock output is also not more than 13.5MHz, because the output speed of I/O port of STC15 series MCU is not more than 13.5MHz.

### 7.7.1 Special Function Registers Related to Programmable Clock Output

Symbol	Description	Address	Bit Address and Symbol								Value after Power-on or Reset
			MSB				LSB				
AUXR	Auxiliary register	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C $\bar{T}$	T2x12	EXTRAM	S1ST2	0000 0001B
INT_CLKO AUXR2	External Interrupt enable and Clock output register	8FH	-	EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO	x000 0000B
CLK_DIV (PCON2)	Clock Division register	97H	MCKO_S1	MCKO_S1	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000B
T4T3M	Timer 4 and Timer 3 Mode register	D1H	T4R	T4_C $\bar{T}$	T4x12	T4CLKO	T3R	T3_C $\bar{T}$	T3x12	T3CLKO	0000 0000B

The statement (used in C language) of Special function registers INT\_CLKO/AUXR/CLK\_DIV/T4T3M:

```
sfr INT_CLKO = 0x8F; //The address statement of special function register INT_CLKO
sfr AUXR = 0x8E; //The address statement of Special function register AUXR
sfr CLK_DIV = 0x97; //The address statement of Special function register CLK_DIV
sfr T4T3M = 0xD1; //The address statement of Special function register T4T3M
```

The statement (used in Assembly language) of Special function registers INT\_CLKO/AUXR/CLK\_DIV/T4T3M:

```
INT_CLKO EQU 8FH ;The address statement of special function register INT_CLKO
AUXR EQU 8EH ;The address statement of Special function register AUXR
CLK_DIV EQU 97H ;The address statement of Special function register CLK_DIV
T4T3M EQU D1H ;The address statement of Special function register T4T3M
```

#### 1. CLK\_DIV (PCON2) : Clock Division register(Non bit addressable)

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CLK_DIV (PCON2)	97H	name	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0

ADRJ the adjustment bit of ADC result

- 0 ADC\_RES[7:0] store high 8-bit ADC result ADC\_RESL[1:0] store low 2-bit ADC result
- 1 ADC\_RES[1:0] store high 2-bit ADC result ADC\_RESL[7:0] store low 8-bit ADC result

Tx\_Rx the set bit of relay and broadcast mode of UART1

- 0 UART1 works on normal mode
- 1 UART1 works on relay and broadcast mode that to say output the input level state of RxD port to the outside TxD pin in real time, namely the external output of TxD pin can reflect the input level state of RxD port.

the RxD and TxD of UART1 can be switched in 3 groups of pins: [RxD/P3.0, TxD/P3.1];

[RxD\_2/P3.6, TxD\_2/P3.7];

[RxD\_3/P1.6, TxD\_3/P1.7].

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
CLK_DIV (PCON2)	97H	Clock Division register	MCKO_S1	MCKO_S0	ADJR	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000
INT_CLKO (AUXR2)	8FH	External Interrupt enable and Clock output register	-	EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO	x000 0000

MCKO_S2	MCKO_S1	MCKO_S0	the control bit of master clock output by dividing the frequency (The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator)
0	0	0	Master clock do not output external clock
0	0	1	Master clock output external clock but its frequency do not be divided and the output clock frequency = MCLK / 1
0	1	0	Master clock output external clock but its frequency is divided by 2 and the output clock frequency = MCLK / 2
0	1	1	Master clock output external clock but its frequency is divided by 4 and the output clock frequency = MCLK / 4
1	0	0	Master clock output external clock but its frequency is divided by 4 and the output clock frequency = MCLK / 16

The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator.  
MCLK is the frequency of master clock.

STC15W4K32S4 series MCU output master clock on MCLKO/P5.4

MCLKO\_2 to select Master Clock output on where

0 Master Clock output on MCLKO/P5.4

1 Master Clock output on MCLKO\_2/XTAL2/P1.6

The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator.

CLKS2	CLKS1	CLKS0	the control bit of system clock (System clock refers to the master clock that has been divided frequency, which is offered to CPU, UARTs, SPI, Timers, CCP/PWM/PCA and A/D Converter)
0	0	0	Master clock frequency/1, No division
0	0	1	Master clock frequency/2
0	1	0	Master clock frequency/4
0	1	1	Master clock frequency/8
1	0	0	Master clock frequency/16
1	0	1	Master clock frequency/32
1	1	0	Master clock frequency/64
1	1	1	Master clock frequency/128

The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator.

## 2. INT\_CLKO (AUXR2) : External Interrupt Enable and Clock Output register

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name		EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO

**B0 - T0CLKO** : Whether is P3.5/T1 configured for Timer 0(T0) programmable clock output T0CLKO or not.

1, P3.5/T1 is configured for Timer0 programmable clock output T0CLKO, the clock output frequency = T0 overflow/2

If Timer/Counter 0 in mode 0 (16 bit auto-reloadable mode),

and if  $C/\overline{T} = 0$ , namely Timer/Counter 0 count on the internal system clock,

When T0 in 1T mode (AUXR.7/T0x12=1), the output frequency = (SYSclk)/(65536-[RL\_TH0, RL\_TL0])/2

When T0 in 12T mode (AUXR.7/T0x12=0), the output frequency = (SYSclk) /12/ (65536-[RL\_TH0, RL\_TL0])/2

and if  $C/\overline{T} = 1$ , namely Timer/Counter 0 count on the external pulse input from P3.4/T0,

the output frequency = (T0\_Pin\_CLK) / (65536-[RL\_TH0, RL\_TL0])/2

If Timer/Counter 0 in mode 2 (8 bit auto-reloadable mode),

and if  $C/\overline{T} = 0$ , namely Timer/Counter 0 count on the internal system clock,

When T0 in 1T mode(AUXR.7/T0x12=1), the output frequency = (SYSclk) / (256-TH0) / 2

When T0 in 12T mode(AUXR.7/T0x12=0), the output frequency = (SYSclk) / 12 / (256-TH0) / 2

and if  $C/\overline{T} = 1$ , namely Timer/Counter 0 count on the external pulse input from P3.4/T0,

the output frequency = (T0\_Pin\_CLK) / (256-TH0) / 2

0, P3.5/T1 is not configure for Timer 0 programmable clock output T0CLKO

**B1 - T1CLKO** : Whether is P3.4/T0 configured for Timer 1(T1) programmable clock output T1CLKO or not.

1, P3.4/T0 is configured for Timer1 programmable clock output T1CLKO, the clock output frequency = T1 overflow/2

If Timer/Counter 1 in mode 1 (16 bit auto-reloadable mode),

and if  $C/\overline{T} = 0$ , namely Timer/Counter 1 count on the internal system clock,

When T1 in 1T mode (AUXR.6/T1x12=1), the output frequency = (SYSclk)/(65536-[RL\_TH1, RL\_TL1])/2

When T1 in 12T mode (AUXR.6/T1x12=0), the output frequency = (SYSclk) /12/ (65536-[RL\_TH1, RL\_TL1])/2

and if  $C/\overline{T} = 1$ , namely Timer/Counter 1 count on the external pulse input from P3.5/T1,

the output frequency = (T1\_Pin\_CLK) / (65536-[RL\_TH1, RL\_TL1])/2

If Timer/Counter 1 in mode 2 (8 bit auto-reloadable mode),

and if  $C/\overline{T} = 0$ , namely Timer/Counter 1 count on the internal system clock,

When T1 in 1T mode(AUXR.6/T1x12=1), the output frequency = (SYSclk) / (256-TH1) / 2

When T1 in 12T mode(AUXR.6/T1x12=0), the output frequency = (SYSclk) / 12 / (256-TH1) / 2

and if  $C/\overline{T} = 1$ , namely Timer/Counter 1 count on the external pulse input from P3.5/T1,

the output frequency = (T1\_Pin\_CLK) / (256-TH1) / 2

0, P3.4/T0 is not configure for Timer 1 programmable clock output T1CLKO

**B2 - T2CLKO** : Whether is P3.0 configured for Timer 2(T2) programmable clock output T2CLKO or not.

1, P3.0 is configured for Timer2 programmable clock output T2CLKO, the clock output frequency = T2 overflow/2

If T2\_  $C/\overline{T} = 0$ , namely Timer/Counter 2 count on the internal system clock,

When T2 in 1T mode (AUXR.2/T2x12=1), the output frequency = (SYSclk)/(65536-[RL\_TH2, RL\_TL2])/2

When T2 in 12T mode (AUXR.2/T2x12=0), the output frequency = (SYSclk) /12/ (65536-[RL\_TH2, RL\_TL2])/2

If T2\_  $C/\overline{T} = 1$ , namely Timer/Counter 2 count on the external pulse input from P3.1/T2,

the output frequency = (T2\_Pin\_CLK) / (65536-[RL\_TH2, RL\_TL2])/2

0, P3.0 is not configure for Timer 2 programmable clock output T2CLKO

## 2. INT\_CLKO (AUXR2) : External Interrupt Enable and Clock Output register

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name		EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO

B4 - EX2 : Enable bit of External Interrupt 2( $\overline{\text{INT2}}$ )

B5 - EX3 : Enable bit of External Interrupt 3( $\overline{\text{INT3}}$ )

B6 - EX4 : Enable bit of External Interrupt 4( $\overline{\text{INT4}}$ )

## 3. AUXR : Auxiliary register (Address:8EH, Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/ $\overline{\text{T}}$	T2x12	EXTRAM	S1ST2

B7 - T0x12 : Timer 0 clock source bit.

0 : The clock source of Timer 0 is SYSclk/12. It will compatible to the traditional 8051 MCU

1 : The clock source of Timer 0 is SYSclk/1. It will drive the T0 faster than a traditional 8051 MCU

B6 - T1x12 : Timer 1 clock source bit.

0 : The clock source of Timer 1 is SYSclk/12. It will compatible to the traditional 8051 MCU

1 : The clock source of Timer 1 is SYSclk/1. It will drive the T0 faster than a traditional 8051 MCU

If T1 is used as the baud-rate generator of UART1, T1x12 will decide whether UART1 is 1T or 12T.

B5 - UART\_M0x6 : Baud rate select bit of UART1 while it is working under 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.

B4 - T2R Timer 2 Run control bit

0 : not run Timer 2;

1 : run Timer 2.

B3 - T2\_C/ $\overline{\text{T}}$ : Counter or timer 2 selector

0 : as Timer (namely count on internal system clock)

1 : as Counter (namely count on the external pulse input from T2/P3.1)

B2 - T2x12 : Timer 2 clock source bit.

0 : The clock source of Timer 2 is SYSclk/12.

1 : The clock source of Timer 2 is SYSclk/1.

If T2 is used as the baud-rate generator of UART1 or UART2, T1x12 will decide whether UART1 or UART2 is 1T or 12T.

B1 - EXTRAM : Internal / external RAM access control bit.

0 : On-chip auxiliary RAM is enabled.

1 : On-chip auxiliary RAM is always disabled.

B0 - S1ST2 : the control bit that UART1 select Timer 2 as its baud-rate generator.

0 : Select Timer 1 as the baud-rate generator of UART1

1 : Select Timer 2 as the baud-rate generator of UART1. Timer 1 is released to use in other functions.



#### 4. T4T3M : Timer 4 and Timer 3 Mode register (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/ $\bar{T}$	T4x12	T4CLKO	T3R	T3_C/ $\bar{T}$	T3x12	T3CLKO

B7 - T4R Timer 4 Run control bit

- 0 : not run Timer 4;
- 1 : run Timer 4.

B6 - T4\_C/ $\bar{T}$ : Counter or timer 4 selector

- 0 : as Timer (namely count on internal system clock)
- 1 : as Counter (namely count on the external pulse input from T4/P0.7)

B5 - T4x12 : Timer 4 clock source bit.

- 0 : The clock source of Timer 4 is SYSclk/12.
- 1 : The clock source of Timer 4 is SYSclk/1.

B4 - T4CLKO : Whether is P0.6 configured for Timer 4(T4) programmable clock output T4CLKO or not.

- 1, P0.6 is configured for Timer 4 programmable clock output T4CLKO, the clock output frequency =  $T4 \text{ overflow} / 2$

If T4\_C/ $\bar{T}$  = 0, namely Timer/Counter 4 count on the internal system clock,

When T4 in 1T mode (T4T3.5/T4x12=1), the output frequency =  $(\text{SYSclk}) / (65536 - [\text{RL\_TH4}, \text{RL\_TL4}]) / 2$

When T4 in 12T mode (T4T3.5/T4x12=0), the output frequency =  $(\text{SYSclk}) / 12 / (65536 - [\text{RL\_TH4}, \text{RL\_TL4}]) / 2$

If T4\_C/ $\bar{T}$  = 1, namely Timer/Counter 4 count on the external pulse input from P0.7/T4,

the output frequency =  $(T4\_Pin\_CLK) / (65536 - [\text{RL\_TH4}, \text{RL\_TL4}]) / 2$

- 0, P0.6 is not configure for Timer 4 programmable clock output T4CLKO

B3 - T3R Timer 3 Run control bit

- 0 : not run Timer 3;
- 1 : run Timer 3.

B2 - T3\_C/ $\bar{T}$ : Counter or timer 3 selector

- 0 : as Timer (namely count on internal system clock)
- 1 : as Counter (namely count on the external pulse input from T3/P0.5)

B1 - T3x12 : Timer 3 clock source bit.

- 0 : The clock source of Timer 3 is SYSclk/12.
- 1 : The clock source of Timer 3 is SYSclk/1.

B0 - T3CLKO : Whether is P0.4 configured for Timer 3(T3) programmable clock output T3CLKO or not.

- 1, P0.4 is configured for Timer 3 programmable clock output T3CLKO, the clock output frequency =  $T3 \text{ overflow} / 2$

If T3\_C/ $\bar{T}$  = 0, namely Timer/Counter 3 count on the internal system clock,

When T3 in 1T mode (T4T3.1/T3x12=1), the output frequency =  $(\text{SYSclk}) / (65536 - [\text{RL\_TH3}, \text{RL\_TL3}]) / 2$

When T3 in 12T mode (T4T3.1/T3x12=0), the output frequency =  $(\text{SYSclk}) / 12 / (65536 - [\text{RL\_TH3}, \text{RL\_TL3}]) / 2$

If T3\_C/ $\bar{T}$  = 1, namely Timer/Counter 3 count on the external pulse input from P0.5/T3,

the output frequency =  $(T3\_Pin\_CLK) / (65536 - [\text{RL\_TH3}, \text{RL\_TL3}]) / 2$

- 0, P0.4 is not configure for Timer 3 programmable clock output T3CLKO

## 7.7.2 Master Clock Output and Demo Program(C and ASM)

The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator. The speed of external programmable clock output of 5V MCU is also not more than 13.5MHz, because the output speed of I/O port of STC15 series 5V MCU is not more than 13.5MHz. The speed of external programmable clock output of 3.3V MCU is also not more than 8MHz, because the output speed of I/O port of STC15 series 3.3V MCU is not more than 8MHz.

CLK\_DIV (PCON2) : Clock Division Register (Non bit-addressable)

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CLK_DIV (PCON2)	97H	name	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0

INT\_CLKO (AUXR2) : External Interrupt Enable and Clock Output register

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name		EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO

How to output clock by using MCLKO/P5.4 or MCLKO\_2/XTAL2/P1.6.

The clock output of MCLKO/P5.4 or MCLKO\_2/XTAL2/P1.6 is controlled by the bits MCKO\_S2 and MCKO\_S1 and MCKO\_S0 of register CLK\_DIV. MCLKO/P5.4 or MCLKO\_2/XTAL2/P1.6 can be configured for master clock output whose frequency also can be choose by setting MCKO\_S2 (INT\_CLKO.3) and MCKO\_S1 (CLK\_DIV.7) and MCKO\_S0 (CLK\_DIV.6).

MCKO_S2	MCKO_S1	MCKO_S0	the control bit of master clock output by dividing the frequency (The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator)
0	0	0	Master clock do not output external clock
0	0	1	Master clock output external clock but its frequency do not be divided and the output clock frequency = MCLK / 1
0	1	0	Master clock output external clock but its frequency is divided by 2 and the output clock frequency = MCLK / 2
0	1	1	Master clock output external clock but its frequency is divided by 4 and the output clock frequency = MCLK / 4
1	0	0	Master clock output external clock but its frequency is divided by 4 and the output clock frequency = MCLK / 16

The master clock can either be internal R/C clock or the external input clock or the external crystal oscillator.  
MCLK is the frequency of master clock.

STC15W4K32S4 series MCU output master clock on MCLKO/P5.4

**The speed of external programmable clock output of 5V MCU is also not more than 13.5MHz, because the output speed of I/O port of STC15 series 5V MCU is not more than 13.5MHz.**

**The speed of external programmable clock output of 3.3V MCU is also not more than 8MHz, because the output speed of I/O port of STC15 series 3.3V MCU is not more than 8MHz.**

---

the following is the demo program of Master clock output:

### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of Master clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

#define FOSC 18432000L
//-----
sfr CLK_DIV = 0x97; //Clock divider register
sfr INT_CLKO = 0x8f; //External Interrupt Enable and Clock Output register

//-----

void main()
{
    CLK_DIV = 0x40; //0100,0000 the output frequency of P5.4 is SYSclk
    INT_CLKO = 0x00;

//    CLK_DIV = 0x80; //1000,0000 the output frequency of P5.4 is SYSclk/2
//    INT_CLKO = 0x00;

//    CLK_DIV = 0xC0; //1100,0000 the output frequency of P5.4 is SYSclk/4
//    INT_CLKO = 0x00;

//    CLK_DIV = 0x00; //0000,0000
//    INT_CLKO = 0x08; //0000,1000 the output frequency of P5.4 is SYSclk/16

    while (1);
}
```

---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of Master clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

CLK_DIV      DATA    97H          //Clock divider register
INT_CLKO     DATA    8FH;        //External Interrupt Enable and Clock Output register

;-----
;interrupt vector table

        ORG    0000H
        LJMP   MAIN

;-----

MAIN:    ORG    0100H
        MOV    SP,          #3FH          //initial SP
        MOV    CLK_DIV,    #40H          //0100,0000 the output frequency of P5.4 is SYSclk
        MOV    INT_CLKO   #00H

//      MOV    CLK_DIV,    #80H          //1000,0000 the output frequency of P5.4 is SYSclk/2
//      MOV    INT_CLKO   #00H

//      MOV    CLK_DIV,    #C0H          //1100,0000 the output frequency of P5.4 is SYSclk/4
//      MOV    INT_CLKO   #00H

//      MOV    CLK_DIV,    #00H          //0000,0000
//      MOV    INT_CLKO   #08H          //0000,1000 the output frequency of P5.4 is SYSclk/16

        SJMP   $

//-----
        END
```

## 7.7.3 Timer 0 Programmable Clock Output and Demo Program

INT\_CLKO (AUXR2) : External Interrupt Enable and Clock Output register

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name		EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	T0CLKO

How to output clock by using T0CLKO/P3.5.

The clock output of T0CLKO/P3.5 is controlled by the bit T0CLKO of register INT\_CLKO (AUXR2).

INT\_CLKO .0 - T0CLKO : 1, enable T0 clock output  
0, disable T0 clock output

The output clock frequency of T0CLKO is controlled by Timer 0. When it is used as programmable clock output, Timer 0 must work in **mode 0 (16-bit auto-reload timer/counter)** or **mode 2 (8-bit auto-reload timer/counter)** and don't enable its interrupt to avoid CPU entering interrupt repeatedly unless special circumstances.

When T0CLKO/INT\_CLKO.0=1, P3.5/T1 is configured for Timer0 programmable clock output T0CLKO.

The clock output frequency =  $T0 \text{ overflow} / 2$

If Timer/Counter 0 in mode 0 (16 bit auto-reloadable mode),

and if  $C/\bar{T} = 0$ , namely Timer/Counter 0 count on the internal system clock,

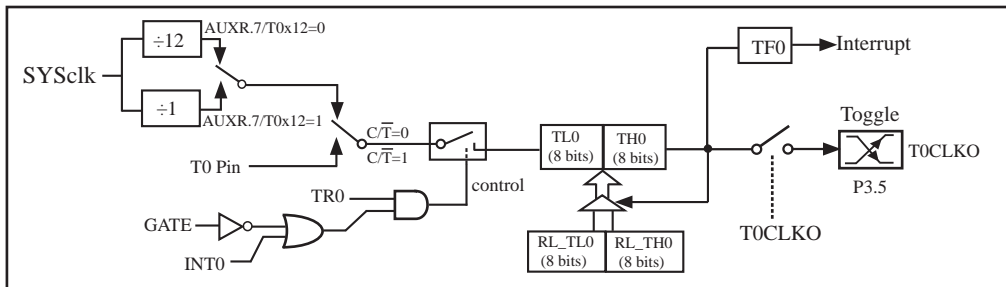
When T0 in 1T mode (AUXR.7/T0x12=1), the output frequency =  $(SYSclk) / (65536 - [RL\_TH0, RL\_TL0]) / 2$

When T0 in 12T mode (AUXR.7/T0x12=0), the output frequency =  $(SYSclk) / 12 / (65536 - [RL\_TH0, RL\_TL0]) / 2$

and if  $C/\bar{T} = 1$ , namely Timer/Counter 0 count on the external pulse input from P3.4/T0,

the output frequency =  $(T0\_Pin\_CLK) / (65536 - [RL\_TH0, RL\_TL0]) / 2$

RL\_TH0 is the reloaded register of TH0, RL\_TL0 is the reload register of TL0.



Timer/Counter 0 mode 0: 16 bit auto-reloadable mode

When T0CLKO/INT\_CLKO.0=1, P3.5/T1 is configured for Timer 0 programmable clock output T0CLKO.

The clock output frequency =  $T0 \text{ overflow} / 2$

If Timer/Counter 0 in mode 2 (8 bit auto-reloadable mode),

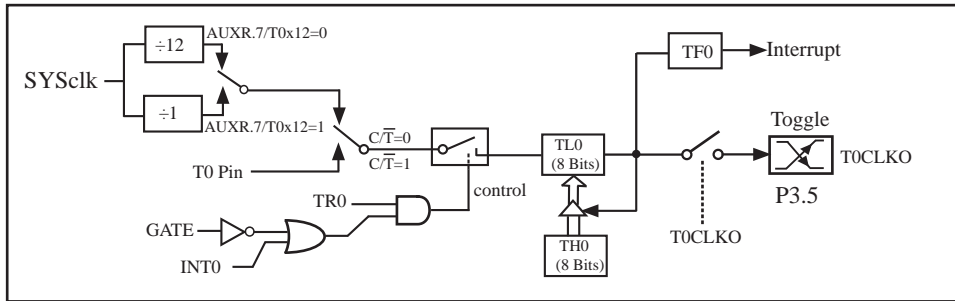
and if  $C/\bar{T} = 0$ , namely Timer/Counter 0 count on the internal system clock,

When T0 in 1T mode (AUXR.7/T0x12=1), the output frequency =  $(SYSclk) / (256 - TH0) / 2$

When T0 in 12T mode (AUXR.7/T0x12=0), the output frequency =  $(SYSclk) / 12 / (256 - TH0) / 2$

and if  $C/\bar{T} = 1$ , namely Timer/Counter 0 count on the external pulse input from P3.4/T0,

the output frequency =  $(T0\_Pin\_CLK) / (256 - TH0) / 2$



Timer/Counter 0 mode 2: 8 bit auto-reloadable mode

The following is the example program that Timer 0 output programmable clock by dividing the frequency of internal system clock or the clock input from external pin T0/P3.4 (C and assembly):

## 1. C Program Listing

```

/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of Timer 0 porgrammable clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

typedef unsigned char BYTE;
typedef unsigned int WORD;

#define FOSC 18432000L

//-----
sfr    AUXR      =    0x8e;
sfr    INT_CLKO  =    0x8f;

sbit   TOCLKO   =    P3^5;

#define F38_4KHz (65536-FOSC/2/38400)           //1T Mode
//#define F38_4KHz (65536-FOSC/2/12/38400)      //12T Mode

```

---

```

//-----
void main()
{
    AUXR   |=   0x80;           //Timer 0 in 1T mode
//    AUXR   &=   ~0x80;       //Timer 0 in 12T mode

    TMOD   =   0x00;           //set Timer0 in mode 0(16 bit auto-reloadable mode)

    TMOD   &=   ~0x04;         //C/T0=0, count on internal system clock
//    TMOD   |=   0x04;         //C/T0=1, count on external pulse input from T0 pin

    TL0    =   F38_4KHz;       //Initial timing value
    TH0    =   F38_4KHz >> 8;
    TR0    =   1;
    INT_CLKO =   0x01;

    while (1);
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of Timer 0 porgrammable clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

AUXR      DATA  08EH
INT_CLKO  DATA  08FH

T0CLKO    BIT    P3.5

F38_4KHz  EQU    0FF10H      //38.4KHz(1T mode, 65536-18432000/2/38400)
//F38_4KHz EQU    0FFECH      //38.4KHz(12T mode,(65536-18432000/2/12/38400)
//-----

```

---

```

    ORG    0000H
    LJMP   MAIN

//-----
    ORG    0100H
MAIN:
    MOV    SP,    #3FH

    ORL    AUXR, #80H           //Timer 0 in 1T mode
//    ANL    AUXR, #7FH           //Timer 0 in 12T mode

    MOV    TMOD, #00H           //set Timer0 in mode 0(16 bit auto-reloadable mode)

    ANL    TMOD, #0FBH         //C/T0=0, count on internal system clock
//    ORL    TMOD, #04H         //C/T0=1, count on external pulse input from T0 pin

    MOV    TL0,    #LOW F38_4KHz //Initial timing value
    MOV    TH0,    #HIGH F38_4KHz
    SETB   TR0
    MOV    INT_CLKO,    #01H

    SJMP   $

;-----

    END

```

---



## 7.7.4 Timer 1 Programmable Clock Output and Demo Program

INT\_CLKO (AUXR2) : External Interrupt Enable and Clock Output register

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name		EX4	EX3	EX2	MCKO_S2	T2CLKO	<b>T1CLKO</b>	T0CLKO

How to output clock by using T1CLKO/P3.4.

The clock output of T1CLKO/P3.4 is controlled by the bit T1CLKO of register INT\_CLKO (AUXR2).

INT\_CLKO.1 - T1CLKO                            1, enable T1 clock output  
  0, disable T1 clock output

The output clock frequency of T1CLKO is controlled by Timer 1. When it is used as programmable clock output, Timer 1 must work in **mode 1 (16-bit auto-reload timer/counter)** or **mode 2(8-bit auto-reload timer/counter)** and don't enable its interrupt to avoid CPU entering interrupt repeatedly unless special circumstances.

When T1CLKO/INT\_CLKO.1=1, P3.4/T0 is configured for Timer 1 programmable clock output T1CLKO.

The clock output frequency = **T1 overflow**/2

If Timer/Counter 1 in mode 1 (16 bit auto-reloadable mode),

and if  $C/\overline{T} = 0$ , namely Timer/Counter 1 count on the internal system clock,

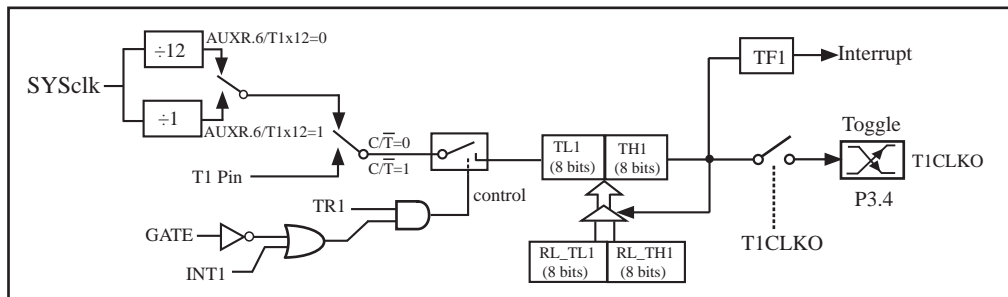
When T1 in 1T mode (AUXR.6/T1x12=1), the output frequency =  $(SYSclk)/(65536-[RL\_TH1, RL\_TL1])/2$

When T1 in 12T mode (AUXR.6/T1x12=0), the output frequency =  $(SYSclk) / 12 / (65536-[RL\_TH1, RL\_TL1])/2$

and if  $C/\overline{T} = 1$ , namely Timer/Counter 1 count on the external pulse input from P3.5/T1,

the output frequency =  $(T1\_Pin\_CLK) / (65536-[RL\_TH1, RL\_TL1])/2$

$RL\_TH1$  is the reloaded register of **TH1**,  $RL\_TL1$  is the reload register of **TL1**.



Timer/Counter 1 mode 0: 16 bit auto-reloadable mode

When T1CLKO/INT\_CLKO.1=1, P3.4/T0 is configured for Timer 1 programmable clock output T1CLKO.

The clock output frequency = **T1 overflow**/2

If Timer/Counter 1 in mode 2 (8 bit auto-reloadable mode),

and if  $C/\overline{T} = 0$ , namely Timer/Counter 1 count on the internal system clock,

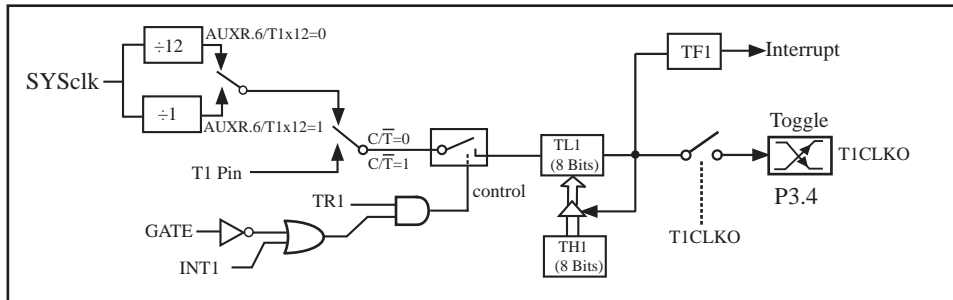
When T1 in 1T mode(AUXR.6/T1x12=1), the output frequency =  $(SYSclk) / (256-TH1) / 2$

When T1 in 12T mode(AUXR.6/T1x12=0), the output frequency =  $(SYSclk) / 12 / (256-TH1) / 2$

and if  $C/\overline{T} = 1$ , namely Timer/Counter 1 count on the external pulse input from P3.5/T1,

the output frequency =  $(T1\_Pin\_CLK) / (256-TH1) / 2$

$RL\_TH1$  is the reloaded register of **TH1**,  $RL\_TL1$  is the reload register of **TL1**.



Timer/Counter 1 mode 2: 8 bit auto-reloadable mode

The following is the example program that Timer 1 output programmable clock by dividing the frequency of internal system clock or the clock input from external pin T1/P3.5 (C and assembly):

## 1. C Program Listing

```

/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of Timer 1 programmable clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

typedef unsigned char    BYTE;
typedef unsigned int     WORD;

#define FOSC  18432000L

//-----
sfr  AUXR      =  0x8e;
sfr  INT_CLKO  =  0x8f;

sbit  T1CLKO   =  P3^4;

#define F38_4KHz  (65536-FOSC/2/38400)           //1T Mode
//#define F38_4KHz  (65536-FOSC/2/12/38400)      //12T Mode

```

---

```

//-----
void main()
{
    AUXR  |=    0x40;           //Timer 1 in 1T mode
//    AUXR  &=   ~0x40;        //Timer 1 in 12T mode

    TMOD  =    0x00;           //set Timer 1 in mode 0(16 bit auto-reloadable mode)

    TMOD  &=   ~0x40;         //C/T1=0, count on internal system clock
//    TMOD  |=    0x40;        //C/T1=1, count on external pulse input from T1 pin

    TL1   =    F38_4KHz;       //Initial timing value
    TH1   =    F38_4KHz >> 8;
    TR1   =    1;
    INT_CLKO =    0x02;

    while (1);
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of Timer 1 programmable clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

AUXR      DATA  08EH
INT_CLKO  DATA  08FH

T1CLKO    BIT    P3.4
F38_4KHz  EQU    0FF10H      //38.4KHz(1T mode, 65536-18432000/2/38400)
//F38_4KHz EQU    0FFECH      //38.4KHz(12T mode, (65536-18432000/2/12/38400)

```

---

```

    ORG    0000H
    LJMP   MAIN

//-----
    ORG    0100H
MAIN:
    MOV    SP,    #3FH

    ORL    AUXR, #40H           //Timer 1 in 1T mode
//    ANL    AUXR, #0BFH       //Timer 1 in 12T mode

    MOV    TMOD, #00H           //set Timer 1 in mode 0(16 bit auto-reloadable mode)

//    ANL    TMOD, #0BFH       //C/T1=0, count on internal system clock
    ORL    TMOD, #40H           //C/T1=1, count on external pulse input from T1 pin

    MOV    TL1,    #LOW F38_4KHz //Initial timing value
    MOV    TH1,    #HIGH F38_4KHz
    SETB   TR1
    MOV    INT_CLKO, #02H

    SJMP   $

;-----

    END

```

## 7.7.5 Timer 2 Programmable Clock Output and Demo Program

INT\_CLKO (AUXR2) : External Interrupt Enable and Clock Output register

SFR Name	SFR Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
INT_CLKO AUXR2	8FH	name		EX4	EX3	EX2	MCKO_S2	T2CLKO	T1CLKO	TOCLKO

How to output clock by using T2CLKO/P3.0.

The clock output of T2CLKO/P3.0 is controlled by the bit T2CLKO of register INT\_CLKO (AUXR2).

INT\_CLKO.2 - T2CLKO :            1, enable T2 clock output  
                                      0, disable T2 clock output

The output clock frequency of T2CLKO is controlled by Timer 2. When it is used as programmable clock output, Timer 2 interrupt don't be enabled to avoid CPU entering interrupt repeatedly unless special circumstances.

When T2CLKO/INT\_CLKO.2=1, P3.0 is configured for Timer 2 programmable clock output T2CLKO.

The clock output frequency =  $T2 \text{ overflow} / 2$

If  $T2\_C\bar{T} = 0$ , namely Timer/Counter 2 count on the internal system clock,

When T2 in 1T mode (AUXR.2/T2x12=1), the output frequency =  $(SYSclk) / (65536 - [RL\_TH2, RL\_TL2]) / 2$

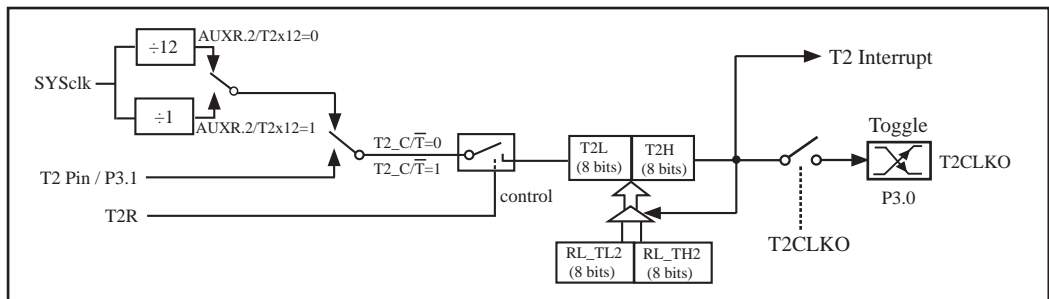
When T2 in 12T mode (AUXR.2/T2x12=0), the output frequency =  $(SYSclk) / 12 / (65536 - [RL\_TH2, RL\_TL2]) / 2$

If  $T2\_C\bar{T} = 1$ , namely Timer/Counter 2 count on the external pulse input from P3.1/T2,

the output frequency =  $(T2\_Pin\_CLK) / (65536 - [RL\_TH2, RL\_TL2]) / 2$

RL\_TH2 is the reloaded register of T2H, RL\_TL2 is the reload register of T2L.

Internal Structure Diagram of Timer 2 is shown below:



Timer / Counter 2 Operating Mode : 16 bit auto-reloadable Mode

---

The following is the example program that Timer 2 output programmable clock by dividing the frequency of internal system clock or the clock input from external pin T2/P3.1 (C and assembly):

## 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of Timer 2 porgrammable clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

typedef unsigned char    BYTE;
typedef unsigned int    WORD;

#define FOSC 18432000L

//-----

sfr    AUXR        = 0x8e;
sfr    INT_CLKO    = 0x8f;
sfr    T2H         = 0xD6;
sfr    T2L         = 0xD7;

sbit   T2CLKO     = P3^0;

#define F38_4KHz    (65536-FOSC/2/38400)        //1T mode
//#define F38_4KHz    (65536-FOSC/2/12/38400)    //12T mode

//-----

void main()
{
    AUXR    |=    0x04;                //Timer 2 in 1T mode
    //    AUXR    &=    ~0x04;        //Timer 2 in 12T mode
```

---

```

    AUXR  &=    ~0x08;           //T2_C/T=0, count on internal system clock
//  AUXR  |=    0x08;           //T2_C/T=1, count on external pulse input from T2(P3.1) pin
    T2L   =    F38_4KHz;           //Initial timing value
    T2H   =    F38_4KHz >> 8;

    AUXR  |=    0x10;
    INT_CLKO =    0x04;

    while (1);
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of Timer 2 porgrammable clock output -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

AUXR      DATA  08EH
INT_CLKO  DATA  08FH
T2H       DATA  0D6H
T2L       DATA  0D7H

T2CLKO    BIT    P3.0

F38_4KHz  EQU    0FF10H           //38.4KHz(1T mode, 65536-18432000/2/38400)
//F38_4KHz EQU    0FFECH           //38.4KHz(12T mode, (65536-18432000/2/12/38400)

//-----

```

---

```

    ORG    0000H
    LJMP   MAIN

//-----

    ORG    0100H
MAIN:
    MOV    SP,    #3FH

    ORL    AUXR, #04H           //Timer 2 in 1T mode
//    ANL    AUXR, #0FBH       //Timer 2 in 12T mode

    ANL    AUXR, #0F7H        //T2_C/T=0, count on internal system clock
//    ORL    AUXR, #08H        //T2_C/T=1, count on external pulse input from T2(P3.1) pin

    MOV    T2L,   #LOW F38_4KHz           //Initial timing value
    MOV    T2H,   #HIGH F38_4KHz
    ORL    AUXR, #10H
    MOV    INT_CLKO, #04H

    SJMP   $

;-----

    END

```



## 7.7.6 Timer 3 Programmable Clock Output and Demo Program

T4T3M : Timer 4 and Timer 3 Mode register (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

How to output clock by using T3CLKO/P0.4.

The clock output of T3CLKO/P0.4 is controlled by the bit T3CLKO of register T4T3M.

T4T3M.0 - T3CLKO :     1, enable T3 clock output  
                           0, disable T3 clock output

The output clock frequency of T3CLKO is controlled by Timer 3. When it is used as programmable clock output, Timer 3 interrupt don't be enabled to avoid CPU entering interrupt repeatedly unless special circumstances.

When T3CLKO/T4T3M.0=1, P0.4 is configured for Timer 3 programmable clock output T3CLKO.

The clock output frequency =  $T3 \text{ overflow} / 2$

If  $T3\_C/T = 0$ , namely Timer/Counter 3 count on the internal system clock,

When T3 in 1T mode (T4T3.1/T3x12=1), the output frequency =  $(SYSclk) / (65536 - [RL\_TH3, RL\_TL3]) / 2$

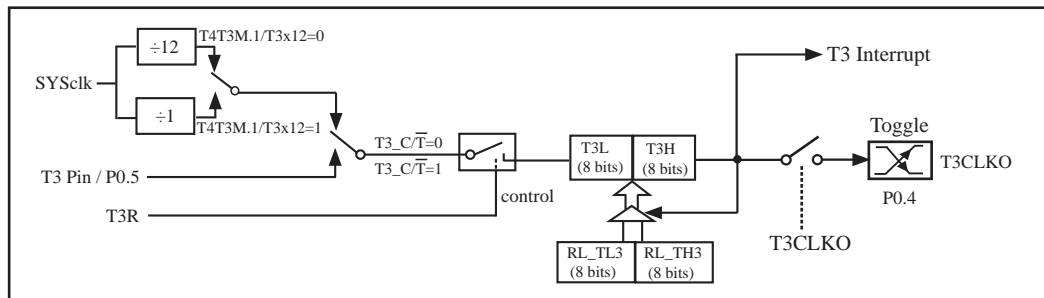
When T3 in 12T mode (T4T3.1/T3x12=0), the output frequency =  $(SYSclk) / 12 / (65536 - [RL\_TH3, RL\_TL3]) / 2$

If  $T3\_C/T = 1$ , namely Timer/Counter 3 count on the external pulse input from P0.5/T3,

the output frequency =  $(T3\_Pin\_CLK) / (65536 - [RL\_TH3, RL\_TL3]) / 2$

RL\_TH3 is the reloaded register of T3H, RL\_TL3 is the reload register of T3L.

Internal Structure Diagram of Timer 3 is shown below:



Timer / Counter 3 Operating Mode : 16 bit auto-reloadable Mode

## 7.7.7 Timer 4 Programmable Clock Output and Demo Program

T4T3M : Timer 4 and Timer 3 Mode register (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

How to output clock by using T4CLKO/P0.6.

The clock output of T4CLKO/P0.6 is controlled by the bit T4CLKO of register T4T3M.

T4T3M.4 - T4CLKO :     1, enable clock output  
                           0, disable clock output

The output clock frequency of T4CLKO is controlled by Timer 4. When it is used as programmable clock output, Timer 4 interrupt don't be enabled to avoid CPU entering interrupt repeatedly unless special circumstances.

When T4CLKO/T4T3M.4=1, P0.6 is configured for Timer 4 programmable clock output T4CLKO.

The clock output frequency = T4 overflow/2

If T4\_C/T = 0, namely Timer/Counter 4 count on the internal system clock,

When T4 in 1T mode (T4T3.5/T4x12=1), the output frequency = (SYSclk)/(65536-[RL\_TH4, RL\_TL4])/2

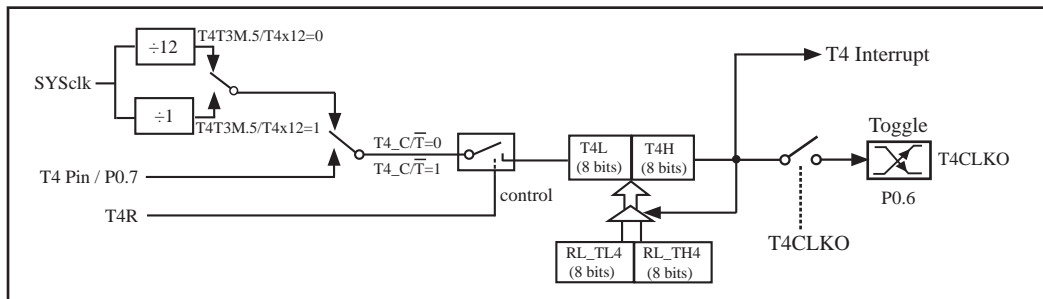
When T4 in 12T mode (T4T3.5/T4x12=0), the output frequency = (SYSclk) /12/ (65536-[RL\_TH4, RL\_TL4])/2

If T4\_C/T = 1, namely Timer/Counter 4 count on the external pulse input from P0.7/T4,

the output frequency = (T4\_Pin\_CLK) / (65536-[RL\_TH4, RL\_TL4])/2

RL\_TH4 is the reloaded register of T4H, RL\_TL4 is the reload register of T4L.

Internal Structure Diagram of Timer 4 is shown below:



Timer / Counter 4 Operating Mode : 16 bit auto-reloadable Mode

## 7.8 Power-Down Wake-Up Special Timer and Demo Program

Power-down wake-up special Timer is added to parts of STC15W4K32S4 series MCU. Besides external interrupts, power-down wake-up timer also can wake up MCU from Stop/PD mode after MCU go into Stop/Power-Down (PD) mode.

The power consumption of power-down wake-up special Timer : 3uA (for 3V chip) and 5uA (for 5V chip).

Power-down wake-up special Timer is controlled and managed by registers [WKTCH](#) and [WKTCL](#).

**WKTCL** : Power-Down Wake-up Timer Control register low (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
WKTCL	AAH	name									1111 1110B

**WKTCH** : Power-Down Wake-up Timer Control register high (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
WKTCH	ABH	name	WKTEN								0111 1111B

Internal power-down wake-up special Timer consists of a 15-bit timer {WKTCH[6:0],WKTCL[7:0]}. The maximum count value of the 15-bit timer {WKTCH[6:0],WKTCL[7:0]} is 32768, while the minimum is 0.

**WKTEN** The enable bit of internal power-down wake-up special Timer

WKTEN=1 enable internal power-down wake-up special Timer

WKTEN=0 disable internal power-down wake-up special Timer.

There are two hidden registers WKTCL\_CNT and WKTCH\_CNT designed for internal power-down wake-up special Timer. The address of WKTCL\_CNT is the same as WKTCL's, and WKTCH\_CNT and WKTCH share in the same address. In fact, WKTCL\_CNT and WKTCH\_CNT are used as counter, while WKTCL and WKTCH are used as comparator. The writing on registers [WKTCH, WKTCL] only can be written into registers [WKTCH, WKTCL], but not into registers [WKTCH\_CNT, WKTCL\_CNT]. However, it is actually not to read the content of registers [WKTCH, WKTCL] but the registers [WKTCH\_CNT, WKTCL\_CNT] that reads the content of registers [WKTCH, WKTCL].

Special Function Registers WKTCL\_CNT and WKTCH\_CNT are shown below:

**WKTCL\_CNT**

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
WKTCL_CNT	AAH	name									1111 1111B

**WKTCH\_CNT**

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
WKTCH_CNT	ABH	name	-								x111 1111B

---

That can enable the internal power-down wake-up timer by setting the bit WKTEN(Power Down Wakeup Timer Enable) for 1. Once MCU go into Stop/Power-Down mode, the register [WKTCH\_CNT,WKTCL\_CNT] would be incremented from 7FFFH to the preload value of register{WKTCH[6:0],WKTCL[7:0]}. If the value of register [WKTCH\_CNT,WKTCL\_CNT] has been incremented to equal to the register{WKTCH[6:0],WKTCL[7:0]}, the system clock would start to oscillate. If the internal system clock is used as the master clock (selected by STC-ISP Writer/Programmer), MCU would be waked up from Stop/Power-Down mode after 64 clocks. If the external crystal or clock is used as the master clock (selected by STC-ISP Writer/Programmer), MCU would be waked up from Stop/Power-Down mode after 1024 clocks. The content of register [WKTCH\_CNT,WKTCL\_CNT] leaves unchanged after MCU is waked up from Stop/Power-Down mode. The waiting time of MCU in Stop/Power-Down mode can be required by reading the register [WKTCH,WKTCL] (actually read the register [WKTCH\_CNT,WKTCL\_CNT]).

Note: The preload value of register {WKTCH[6:0], WKTCL[7:0]} equals to subtract 1 from the count value that users want to. For example, if users want to count 10 times, the preload value of register {WKTCH[6:0], WKTCL[7:0]} would be 9. And 7FFFH (that is 32767) would be written into the register {WKTCH[6:0], WKTCL[7:0]} if the count value is 32768.

Internal power-down wake-up Timer has its own internal clock which decide the time taken by counting a time. The clock frequency of internal power-down wake-up Timer is about 32768Hz. The frequency in normal temperature can be accessed by reading the content of F8 and F9 units in RAM area for STC15 series MCU (except STC15F101W series). For STC15F101W series, it can be obtained by reading the content of 78 and 79 units in RAM area. Take F8 and F9 units in RAM area for example to introduce the frequency of internal power-down wake-up Timer.

If [WIRC\_H,WIRC\_L] represent the clock frequency of internal power-down wake-up Timer in normal temperature accessed from the units F8 and F9 in RAM area, the counting time of internal power-down wake-up Timer is calculated by following equation:

$$\text{Counting time of internal power-down wake-up Timer} = \frac{10^6 \text{uS}}{[\text{WIRC\_H}, \text{WIRC\_L}]} \times 16 \times \text{times}$$

If the content of F8 unit is 80H and F9 is 00H, that is to say [WIRC\_H,WIRC\_L] (the frequency of internal power-down wake-up Timer) is 32768Hz, the counting time of internal power-down wake-up Timer would be :

488.28uS x 1	= 488.28uS	when {WKTCH[6:0],WKTCL[7:0]} = 0
488.28uS x 10	= 4.8828mS	when {WKTCH[6:0],WKTCL[7:0]} = 9
488.28uS x 100	= 48.828mS	when {WKTCH[6:0],WKTCL[7:0]} = 99
488.28uS x 1000	= 488.28mS	when {WKTCH[6:0],WKTCL[7:0]} = 999
488.28uS x 4096	= 2.0S	when {WKTCH[6:0],WKTCL[7:0]} = 4095
488.28uS x 32768	=16S	when {WKTCH[6:0],WKTCL[7:0]} = 32767

---

If the content of F8 unit is 79H and F9 is 18H, that is to say [WIRC\_H,WIRC\_L] (the frequency of internal power-down wake-up Timer) is 31000Hz, the counting time of internal power-down wake-up Timer would be :

516.13uS x 1	≈ 516.13uS	when {WKTCH[6:0],WKTCL[7:0]} = 0
516.13uS x 10	≈ 5.1613mS	when {WKTCH[6:0],WKTCL[7:0]} = 9
516.13uS x 100	≈ 51.613mS	when {WKTCH[6:0],WKTCL[7:0]} = 99
516.13uS x 1000	≈ 516.13mS	when {WKTCH[6:0],WKTCL[7:0]} = 999
516.13uS x 4096	≈ 2.1S	when {WKTCH[6:0],WKTCL[7:0]} = 4095
516.13uS x 32768	≈16.9S	when {WKTCH[6:0],WKTCL[7:0]} = 32767

If the content of F8 unit is 80H and F9 is E8H, that is to say [WIRC\_H,WIRC\_L] (the frequency of internal power-down wake-up Timer) is 31000Hz, the counting time of internal power-down wake-up Timer would be :

484. 85uS x 1	≈ 484. 85uS	when {WKTCH[6:0],WKTCL[7:0]} = 0
484. 85uS x 10	≈ 4.8485mS	when {WKTCH[6:0],WKTCL[7:0]} = 9
484. 85uS x 100	≈ 48.485mS	when {WKTCH[6:0],WKTCL[7:0]} = 99
484. 85uS x 1000	≈ 484. 85mS	when {WKTCH[6:0],WKTCL[7:0]} = 999
484. 85uS x 4096	≈ 1.986S	when {WKTCH[6:0],WKTCL[7:0]} = 4095
484. 85uS x 32768	≈15.89S	when {WKTCH[6:0],WKTCL[7:0]} = 32767

*/\*Demo program using internal power-down wake-up special Timer wake up Stop/Power-Down mode(C and ASM) \*/*

### **1. C Program Listing**

```
/*-----*/  
/* --- STC MCU Limited. -----*/  
/* --- Exam Program using power-down wake-up Timer to wake up Stop/Power-Down mode */  
/* If you want to use the program or the program referenced in the -----*/  
/* article, please specify in which data and procedures from STC -----*/  
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/  
/*---- And only contain < reg51.h > as header file -----*/  
/*-----*/
```

```
//suppose the frequency of test chip is 18.432MHz
```

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

---

```

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

sbit   P10 = P1^0;

//-----
void main()
{
    WKTCL = 49;                //wake-up cycle: 488us*(49+1) = 24.4ms
    WKTCH = 0x80;

    while (1)
    {
        PCON = 0x02;          //Enter Stop/Power-Down Mode
        _nop_();
        _nop_();
        P10 = !P10;
    }
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using power-down wake-up Timer wake up Stop/Power-Down mode -*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

WKTCL DATA 0AAH
WKTCH DATA 0ABH

```

---

//-----

```
    ORG    0000H
    LJMP   MAIN
```

//-----

```
MAIN:    ORG    0100H
         MOV    SP,    #3FH

         MOV    WKTCL, #49           //wake-up cycle: 488us*(49+1) = 24.4ms
         MOV    WKTCH, #80H

LOOP:    MOV    PCON,  #02H         //Enter Stop/Power-Down Mode
         NOP
         NOP
         CPL    P1.0
         JMP    LOOP

         SJMP   $
```

;-----

```
    END
```

---

## 7.9 Application Notes for Timer in practice

### (1) Real-time Timer

Timer/Counter start running, When the Timer/Counter is overflow, the interrupt request generated, this action handle by the hardware automatically, however, the process which from propose interrupt request to respond interrupt request requires a certain amount of time, and that the delay interrupt request on-site with the environment varies, it normally takes three machine cycles of delay, which will bring real-time processing bias. In most occasions, this error can be ignored, but for some real-time processing applications, which require compensation.

Such as the interrupt response delay, for timer mode 0 and mode 1, there are two meanings: the first, because of the interrupt response time delay of real-time processing error; the second, if you require multiple consecutive timing, due to interruption response delay, resulting in the interrupt service program once again sets the count value is delayed by several count cycle.

If you choose to use Timer/Counter mode 1 to set the system clock, these reasons will produce real-time error for this situation, you should use dynamic compensation approach to reducing error in the system clock, compensation method can refer to the following example program.

```
...
CLR    EA                ;disable interrupt
MOV    A,    TLx        ;read TLx
ADD    A,    #LOW       ;LOW is low byte of compensation value
MOV    TLx,  A          ;update TLx
MOV    A,    THx        ;read THx
ADDC   A,    #HIGH      ;HIGH is high byte of compensation value
MOV    THx,  A          ;update THx
SETB   EA                ;enable interrupt
...
```

### (2) Dynamic read counts

When dynamic read running timer count value, if you do not pay attention to could be wrong, this is because it is not possible at the same time read the value of the TLx and THx. For example the first reading TLx then THx, because the timer is running, after reading TLx, TLx carry on the THx produced, resulting in error; Similarly, after the first reading of THx then TLx, also have the same problems.

A kind of way avoid reading wrong is first reading THx then TLx and read THx once more, if the THx twice to read the same value, then the read value is correct, otherwise repeat the above process. Realization method reference to the following example code.

```
...
RDTM: MOV    A,    THx        ;save THx to ACC
      MOV    R0,   TLx        ;save TLx to R0
      CJNE   A,    THx,    RDTM ;read THx again and compare with the previous value
      MOV    R1,   A          ;save THx to R1
...
```



---

## Chapter 8 Serial Port (UART) Communication

STC15W4K32S4 series MCU has integrated four serial data communication ports, known as UARTs (Universal Asynchronous Receivers/Transmitters). All the UARTs support full duplex, meaning they can transmit and receive simultaneously. They are also receive-buffered, meaning they can commence reception of a second byte before a previously received byte has been read from the receive register. (However, if the first byte still hasn't been read by the time reception of the second byte is complete, one of the bytes will be lost). UART1 uses register SBUF (address:99H) to hold both the received and transmitted data passing through pins RxD and TxD. Actually, there is two SBUF in the chip, one is for transmit and the other is for receive. Similarly, UART2 uses register S2BUF (address:9BH) to hold both the received and transmitted data passing through pins RxD2 and TxD2. UART3 uses register S3BUF (address:ADH) to hold both the received and transmitted data passing through pins RxD3 and TxD3. UART4 uses register S4BUF (address:85H) to hold both the received and transmitted data passing through pins RxD4 and TxD4. Actually, S2BUF and S3BUF and S4BUF all have two in the chip, one for transmit and the other for receive.

Serial communication for UART1 can take 4 different modes: Mode 0 provides synchronous communication while Modes 1, 2, and 3 provide asynchronous communication. The asynchronous communication operates as a full-duplex Universal Asynchronous Receiver and Transmitter (UART), which can transmit and receive simultaneously and at different baud rates. But there are only two different modes for UART2 and UART3 and UART4. The baud rate of the two modes are all variable.

Serial communication involves the transmission of bits of data through only one communication line. The data are transmitted bit by bit in either synchronous or asynchronous format. Synchronous serial communication transmits out whole block of characters in synchronization with a reference clock while asynchronous serial communication randomly transmits one character at any time, independent of any clock.

UART1 receive and transmit data through pins RxD and TxD which can be switched in three different groups of pins by setting the bits S1\_S1/AUXR1.7 and S1\_S0/P\_SW1.6 in register AUXR1/P\_SW1. the RxD and TxD of UART1 can be switched from [RxD/P3.0,TxD/P3.1] to [RxD\_2/P3.6,TxD\_2/P3.7] or to [RxD\_3/P1.6/XTAL2,TxD\_3/P1.7/XTAL1].

UART2 receive and transmit data through pins RxD2 and TxD2 which can be switched in two different groups of pins by setting the bit S2\_S/P\_SW2.0 in register P\_SW2. the RxD2 and TxD2 of UART2 can be switched from [RxD2/P1.0,TxD2/P1.1] to [RxD2\_2/P4.6,TxD2\_2/P4.7].

UART3 receive and transmit data through pins RxD3 and TxD3 which can be switched in two different groups of pins by setting the bit S3\_S/P\_SW2.1 in register P\_SW2. the RxD3 and TxD3 of UART3 can be switched from [RxD3/P0.0,TxD3/P0.1] to [RxD3\_2/P5.0,TxD3\_2/P5.1].

UART4 receive and transmit data through pins RxD4 and TxD4 which can be switched in two different groups of pins by setting the bit S4\_S/P\_SW2.2 in register P\_SW2. the RxD4 and TxD4 of UART4 can be switched from [RxD4/P0.2,TxD4/P0.3] to [RxD4\_2/P5.2,TxD4\_2/P5.3].

## 8.1 Special Function Registers about Serial Port 1 (UART1)

Symbol	Description	Address	Bit Address and Symbol								Value after Power-on or Reset
			MSB							LSB	
T2H	The high 8-bit of Timer 2 register	D6H									0000 0000B
T2L	The low 8-bit of Timer 2 register	D7H									0000 0000B
AUXR	Auxiliary register	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C $\bar{T}$	T2x12	EXTRAM	S1ST2	0000 0001B
SCON	Serial Control	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000 0000B
SBUF	Serial Buffer	99H									xxxx xxxxB
PCON	Power Control	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011 0000B
IE	Interrupt Enable	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000 0000B
IP	Interrupt Priority Low	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000 0000B
SADEN	Slave Address Mask	B9H									0000 0000B
SADDR	Slave Address	A9H									0000 0000B
AUXR1 P_SW1	Auxiliary register 1	A2H	S1_S1	S1_S0	CCP_S1	CCP_S0	SPL_S1	SPL_S0	0	DPS	0100 0000B
CLK_DIV PCON2	Clock Division register	97H	MCKO_S1	MCKO_S1	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000B

### 1. Serial Port 1 (UART1) Control Register: SCON and PCON

Serial port 1 of STC15 series has two control registers: Serial port control register (SCON) and PCON which used to select Baud-Rate

#### SCON: Serial port Control Register (Bit-Addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	name	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

FE : Framing Error bit. The SMOD0 bit must be set to enable access to the FE bit

0 : The FE bit is not cleared by valid frames but should be cleared by software.

1 : This bit set by the receiver when an invalid stop bit id detected.

REN : When set enables serial reception.

TB8 : The 9th data bit which will be transmitted in mode 2 and 3.

RB8 : In mode 2 and 3, the received 9th data bit will go into this bit.

SM0, SM1 : Serial Port Mode Bit 0/1.

SM0	SM1	Mode	Description	Baud Rate
0	0	Mode 0	synchronous shift serial mode: 8-bit shift register	If UART_M0x6 = 0, baud rate = SYSclk/12, If UART_M0x6 = 1, baud rate = SYSclk / 2
0	1	Mode 1	8-bit UART, baud-rate variable	If UART1 select Timer 2 or Timer 1 (as 16-bit auto-reload timer), baud rate= (T1 or T2 overflow )/4. If UART1 select Timer 1 (as 8-bit auto-reload timer), baud rate = ( 2 <sup>SMOD</sup> /32 )×(T1 overflow)
1	0	Mode 2	9-bit UART	( 2 <sup>SMOD</sup> / 64) x SYSclk SYSclk is system clock frequency
1	1	Mode 3	9-bit UART, baud-rate variable	If UART1 select Timer 2 or Timer 1 (as 16-bit auto-reload timer), baud rate= (T1 or T2 overflow )/4. If UART1 select Timer 1 (as 8-bit auto-reload timer), baud rate = ( 2 <sup>SMOD</sup> /32 )×(T1 overflow)
<p>If T1 in mode 0 (16-bit auto-reload timer/counter) and AUXR.6/T1x12 = 0 , T1 overflow = SYSclk/12/( 65536 - [RL_TH1,RL_TL1]) ;</p> <p>If T1 in mode 0 (16-bit auto-reload timer/counter) and AUXR.6/T1x12 = 1, T1 overflow = SYSclk / (65536 - [RL_TH1,RL_TL1])</p> <p>RL_TH1 is the reloaded register of TH1, and RL_TL1 is the reload register of TL1 in above formula.</p> <p>If T1 in mode 2 (8-bit auto-reload timer/counter) and T1x12 = 0, T1 overflow = SYSclk/12/( 256 - TH1) ;</p> <p>If T1 in mode 2 (8-bit auto-reload timer/counter) and T1x12 = 1, T1 overflow = SYSclk / ( 256 - TH1)</p> <p>If AUXR.2/T2x12 = 0, T2 overflow = SYSclk / 12/ ( 65536 - [RL_TH2,RL_TL2] ) ;</p> <p>If AUXR.2/T2x12 = 1, T2 overflow = SYSclk / ( 65536 - [RL_TH2,RL_TL2] ) .</p> <p>RL_TH2 is the reloaded register of T2H, and RL_TL2 is the reload register of T2L in above formula.</p>				

SM2 : Enable the automatic address recognition feature in mode 2 and 3. If SM2=1, RI will not be set unless the received 9th data bit is 1, indicating an address, and the received byte is a Given or Broadcast address. In mode1, if SM2=1 then RI will not be set unless a valid stop Bit was received, and the received byte is a Given or Broadcast address. In mode 0, SM2 should be 0.

TI : Transmit interrupt flag. Set by hardware when a byte of data has been transmitted by UART0 (after the 8th bit in 8-bit UART Mode, or at the beginning of the STOP bit in 9-bit UART Mode). When the UART0 interrupt is enabled, setting this bit causes the CPU to vector to the UART0 interrupt service routine. This bit must be cleared manually by software.

RI : Receive interrupt flag. Set to '1' by hardware when a byte of data has been received by UART0 (set at the STOP bit sam-pling time). When the UART0 interrupt is enabled, setting this bit to '1' causes the CPU to vector to the UART0 interrupt service routine. This bit must be cleared manually by software.

SMOD/PCON.7 in PCON register can be used to set whether the baud rates of mode 1, mode2 and mode 3 are doubled or not.

**PCON: Power Control register** (Non bit-addressable)

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

SMOD: double Baud rate control bit.

0 : Disable double Baud rate of the UART.

1 : Enable double Baud rate of the UART in mode 1,2,or 3.

SMOD0: Frame Error select.

0 : SCON.7 is SM0 function.

1 : SCON.7 is FE function. Note that FE will be set after a frame error regardless of the state of SMOD0.

## 2. SBUF: Serial port 1 Data Buffer register (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SBUF	99H	name								

It is used as the buffer register in transmission and reception. The serial port buffer register (SBUF) is really two 8-bit registers. Writing to SBUF loads data to be transmitted, and reading SBUF accesses received data. These are two separate and distinct registers, the transmit write-only register, and the receive read-only register.

## 3. Slave Address Control registers SADEN and SADDR

SADEN: Slave Address Mask register

SADDR: Slave Address register

SADDR register is combined with SADEN register to form Given/Broadcast Address for automatic address recognition. In fact, SADEN function as the "mask" register for SADDR register. The following is the example for it.

$$\begin{array}{rcl}
 \text{SADDR} & = & 1100\ 0000 \\
 \text{SADEN} & = & 1111\ 1101 \\
 \hline
 \text{Given} & = & 1100\ 00x0 \longrightarrow \text{The Given slave address will be checked except bit 1 is treated as "don't care".}
 \end{array}$$

The Broadcast Address for each slave is created by taking the logical OR of SADDR and SADEN. Zero in this result is considered as "don't care" and a Broadcast Address of all " don't care". This disables the automatic address detection feature.

## 4. Register bits related to UART1 interrupt: ES and PS

IE: Interrupt Enable Register (Bit-addressable)

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

EA : disables all interrupts.

If EA = 0, no interrupt will be acknowledged.

If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

ES : Serial port 1(UART1) interrupt enable bit.

If ES = 0, Serial port 1(UART1) interrupt would be disabled.

If ES = 1, Serial port 1(UART1) interrupt would be enabled.

IP: Interrupt Priority Register (Bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	name	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0

PS : Serial Port 1 (UART1) interrupt priority control bit.

if PS = 0, Serial Port 1 (UART1) interrupt is assigned lowest priority (priority 0).

if PS = 1, Serial Port 1 (UART1) interrupt is assigned highest priority (priority 1).

## 5. AUXR: Auxiliary register (Address:8EH, Non bit-addressable)

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

B7 - T0x12 : Timer 0 clock source bit.

0 : The clock source of Timer 0 is SYSclk/12. It will compatible to the traditional 8051 MCU

1 : The clock source of Timer 0 is SYSclk/1. It will drive the T0 faster than a traditional 8051 MCU

B6 - T1x12 : Timer 1 clock source bit.

0 : The clock source of Timer 1 is SYSclk/12. It will compatible to the traditional 8051 MCU

1 : The clock source of Timer 1 is SYSclk/1. It will drive the T0 faster than a traditional 8051 MCU

If T1 is used as the baud-rate generator of UART1, T1x12 will decide whether UART1 is 1T or 12T.

**B5 - UART\_M0x6 : Baud rate select bit of UART1 while it is working under 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.

B4 - T2R Timer 2 Run control bit

0 : not run Timer 2;

1 : run Timer 2.

B3 - T2\_C/T: Counter or timer 2 selector

0 : as Timer (namely count on internal system clock)

1 : as Counter (namely count on the external pulse input from T2/P3.1)

B2 - T2x12 : Timer 2 clock source bit.

0 : The clock source of Timer 2 is SYSclk/12.

1 : The clock source of Timer 2 is SYSclk/1.

If T2 is used as the baud-rate generator of UART1 or UART2, T1x12 will decide whether UART1 or UART2 is 1T or 12T.

B1 - EXTRAM : Internal / external RAM access control bit.

0 : On-chip auxiliary RAM is enabled.

1 : On-chip auxiliary RAM is always disabled.

**B0 - S1ST2 : the control bit that UART1 select Timer 2 as its baud-rate generator.**

0 : Select Timer 1 as the baud-rate generator of UART1

1 : Select Timer 2 as the baud-rate generator of UART1. Timer 1 is released to use in other functions.

Serial port 1(UART1) can select Timer 1, also can select Timer 2 as its baud-rate generator. When S1ST2/AUXR.0 is set, Serial port 1(UART1) will select Timer 2 as its baud-rate generator, and Timer 1 can be released for other functions such as timer, counter and programmable clock output.

UART2 only can choose Timer 2 as its its baud-rate generator. UART1 prefer to select Timer 2 as its baud-rate generator, also can choose Timer 1 set by software. UART3 and UART4 default to selecting Timer 2 as their baud-rate generator. UART3 also can choose Timer 3 and UART4 can choose Timer 4 as their baud-rate generator.

## 6. UART1 Switch Register : AUXR1 (P\_SW1)

AUXR1 (P\_SW1): Auxiliary register 1 (Non bit-addressable)

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0100,0000

UART1/S1 can be switched in 3 groups of pins by selecting the control bits S1\_S0 and S1\_S1.

S1_S1	S1_S0	UART1/S1 can be switched between P1 and P3
0	0	UART1/S1 on [P3.0/RxD,P3.1/TxD]
0	1	UART1/S1 on [P3.6/RxD_2,P3.7/TxD_2]
1	0	UART1/S1 on [P1.6/RxD_3/XTAL2,P1.7/TxD_3/XTAL1] when UART1 is on P1, please using internal R/C clock.
1	1	Invalid

Recommnd UART1 on [P3.6/RxD\_2,P3.7/TxD\_2] or [P1.6/RxD\_3/XTAL2,P1.7/TxD\_3/XTAL1].

CCP can be switched in 3 groups of pins by selecting the control bits CCP\_S1 and CCP\_S0.

CCP_S1	CCP_S0	CCP can be switched in P1 and P2 and P3
0	0	CCP on [P1.2/ECI,P1.1/CCP0,P1.0/CCP1]
0	1	CCP on [P3.4/ECI_2,P3.5/CCP0_2,P3.6/CCP1_2]
1	0	CCP on [P2.4/ECI_3,P2.5/CCP0_3,P2.6/CCP1_3]
1	1	Invalid

SPI can be switched in 3 groups of pins by selecting the control bits SPI\_S1 and SPI\_S0

SPI_S1	SPI_S0	SPI can be switched in P1 and P2 and P4
0	0	SPI on [P1.2/SS,P1.3/MOSI,P1.4/MISO,P1.5/SCLK]
0	1	SPI on [P2.4/SS_2,P2.3/MOSI_2,P2.2/MISO_2,P2.1/SCLK_2]
1	0	SPI on [P5.4/SS_3,P4.0/MOSI_3,P4.1/MISO_3,P4.3/SCLK_3]
1	1	Invalid

DPS : DPTR registers select bit.

0 : DPTR0 is selected

1 : DPTR1 is selected

## 8. Set bit of UART1 Relay and Broadcast mode : Tx\_Rx / CLK\_DIV4

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
CLK_DIV (PCON2)	97H	Clock Division register	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000,x000

Tx\_Rx the set bit of relay and broadcast mode of UART1

0 UART1 works on normal mode

1 UART1 works on relay and broadcast mode that to say output the input level state of RxD port to the outside TxD pin in real time, namely the external output of TxD pin can reflect the input level state of RxD port.

the RxD and TxD of UART1 can be switched in 3 groups of pins: [RxD/P3.0, TxD/P3.1];

[RxD\_2/P3.6, TxD\_2/P3.7];

[RxD\_3/P1.6, TxD\_3/P1.7].

---

## 8.2 UART1 Operation Modes

The serial port 1 (UART1) can be operated in 4 different modes which are configured by setting SM0 and SM1 in SFR SCON. Mode 1, Mode 2 and Mode 3 are asynchronous communication. In Mode 0, UART1 is used as a simple shift register.

### 8.2.1 Mode 0 : 8-Bit Shift Register

Mode 0, selected by writing 0s into bits SM1 and SM0 of SCON, puts the serial port into 8-bit shift register mode. Serial data enters and exits through RxD. TxD outputs the shift clock. Eight data bits are transmitted/received with the least-significant (LSB) first. The baud rate is fixed at 1/12 the System clock cycle in the default state. If AUXR.5 (UART\_M0x6) is set, the baud rate is 1/2 System clock cycle.

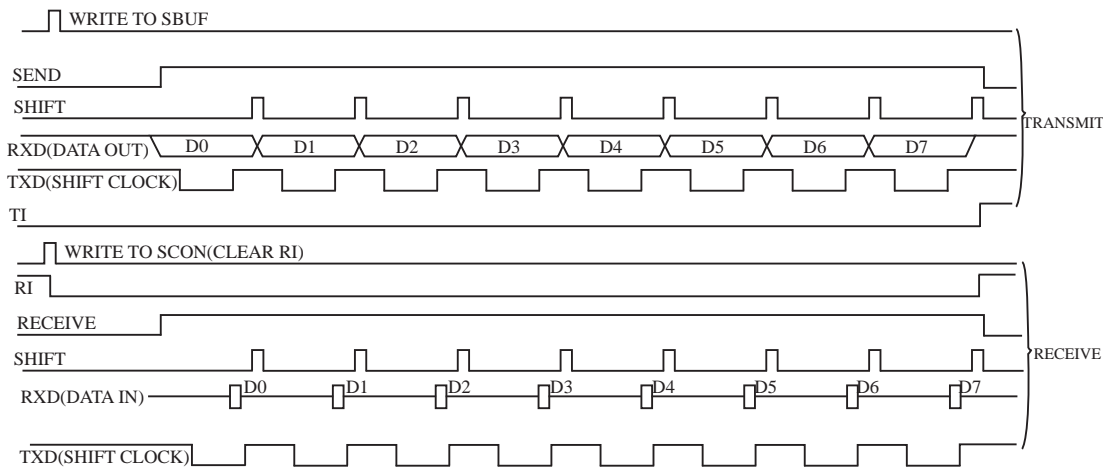
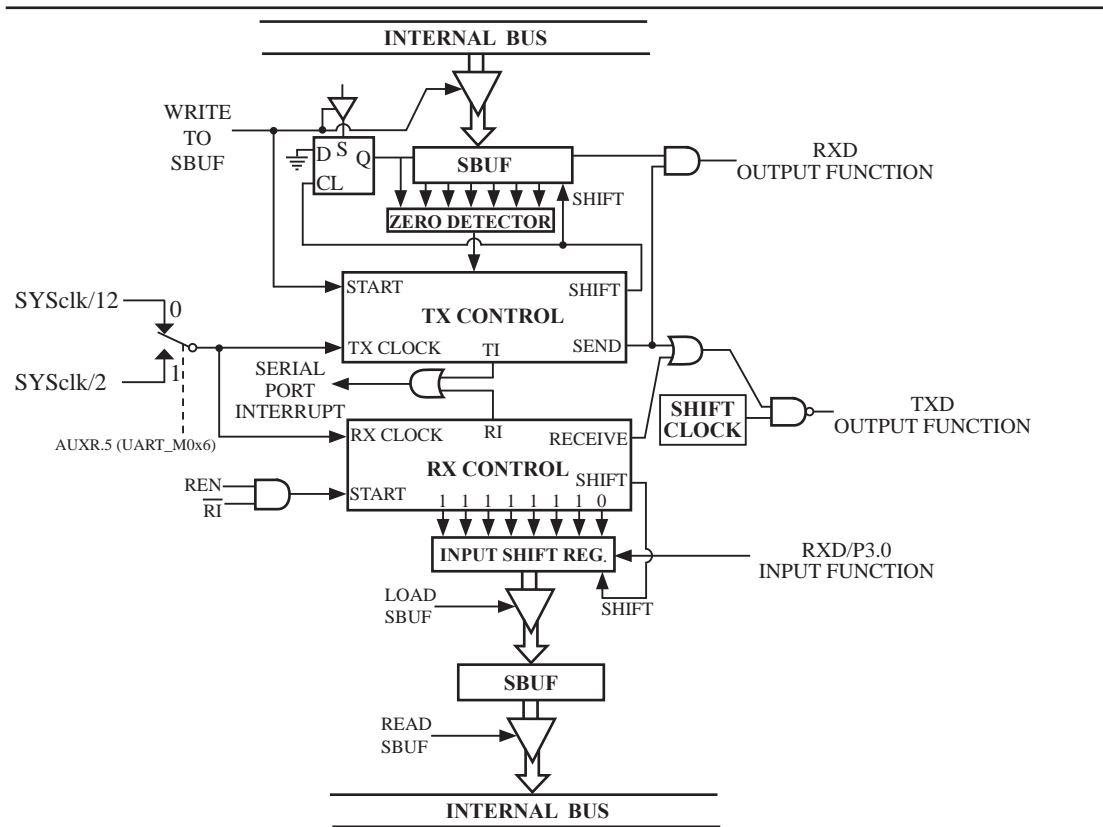
Transmission is initiated by any instruction that uses SBUF as a destination register. The “write to SBUF” signal also loads a “1” into the 9<sup>th</sup> position of the transmit shift register and tells the TX Control block to commence a transmission. The internal timing is such that one full system clock cycle will elapse between "write to SBUF," and activation of SEND.

SEND transfers the output of the shift register to the alternate output function line of P3.0, and also transfers Shift Clock to the alternate output function line of P3.1. At the falling edge of the Shift Clock, the contents of the shift register are shifted one position to the right.

As data bits shift out to the right, “0” come in from the left. When the MSB of the data byte is at the output position of the shift register, then the “1” that was initially loaded into the 9<sup>th</sup> position is just to the left of the MSB, and all positions to the left of that contains zeroes. This condition flags the TX Control block to do one last shift and then deactivate SEND and set TI. Both of these actions occur after "write to SBUF".

Reception is initiated by the condition REN=1 and RI=0. After that, the RX Control unit writes the bits 11111110 to the receive shift register, and in the next clock phase activates RECEIVE. RECEIVE enables SHIFT CLOCK to the alternate output function line of P3.1. At RECEIVE is active, the contents of the receive shift register are shifted to the left one position. The value that comes in from the right is the value that was sampled at the P3.0 pin the rising edge of Shift clock.

As data bits come in from the right, “1”s shift out to the left. When the “0” that was initially loaded into the right-most position arrives at the left-most position in the shift register, it flags the RX Control block to do one last shift and load SBUF. Then RECEIVE is cleared and RI is set.



**Serial Port Mode 0**



---

## 8.2.2 Mode 1: 8-Bit UART with Variable Baud Rate

10 bits are transmitted through TxD or received through RxD. The frame data includes a start bit (0), 8 data bits and a stop bit (1). One receive, the stop bit goes into RB8 in SFR – SCON.

Transmission is initiated by any instruction that uses SBUF as a destination register. The “write to SBUF” signal also loads a “1” into the 9<sup>th</sup> bit position of the transmit shift register and flags the TX Control unit that a transmission is requested. Transmission actually happens at the next rollover of divided-by-16 counter. Thus the bit times are synchronized to the divided-by-16 counter, not to the “write to SBUF” signal.

The transmission begins with activation of  $\overline{\text{SEND}}$ , which puts the start bit at TxD. One bit time later, DATA is activated, which enables the output bit of the transmit shift register to TxD. The first shift pulse occurs one bit time after that.

As data bits shift out to the right, zeroes are clocked in from the left. When the MSB of the data byte is at the output position of the shift register, then the 1 that was initially loaded into the 9<sup>th</sup> position is just to the left of the MSB, and all positions to the left of that contain zeroes. This condition flags the TX Control unit to do one last shift and then deactivate  $\overline{\text{SEND}}$  and set TI. This occurs at the 10<sup>th</sup> divide-by-16 rollover after “write to SBUF.”

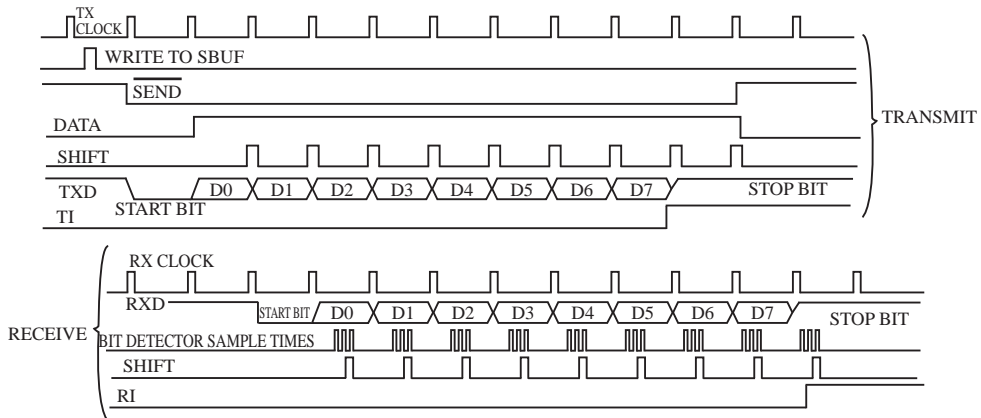
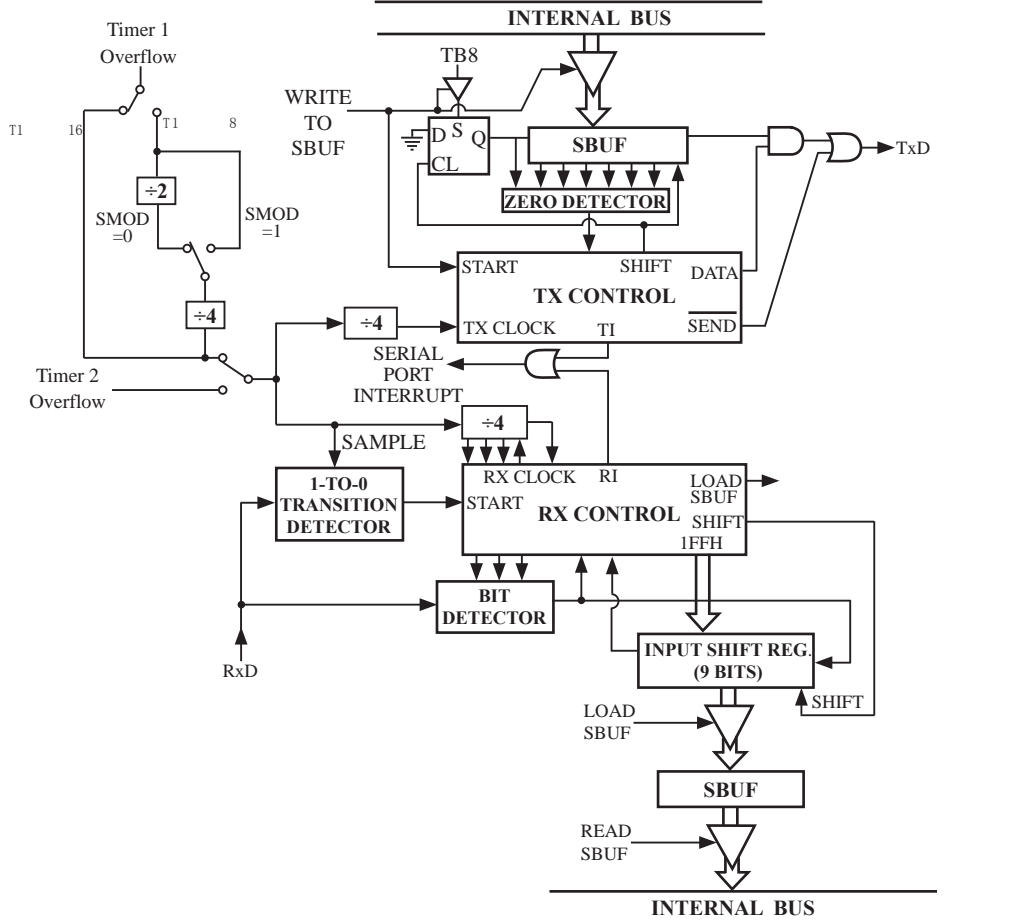
Reception is initiated by a 1-to-0 transition detected at RxD. For this purpose, RxD is sampled at a rate of 16 times the established baud rate. When a transition is detected, the divided-by-16 counter is immediately reset, and 1FFH is written into the input shift register. Resetting the divided-by-16 counter aligns its roll-overs with the boundaries of the incoming bit times.

The 16 states of the counter divide each bit time into 16ths. At the 7<sup>th</sup>, 8<sup>th</sup> and 9<sup>th</sup> counter states of each bit time, the bit detector samples the value of RxD. The value accepted is the value that was seen in at least 2 of the 3 samples. This is done to reject noise. In order to reject false bits, if the value accepted during the first bit time is not a 0, the receive circuits are reset and the unit continues looking for another 1-to-0 transition. This is to provide rejection of false start bits. If the start bit is valid, it is shifted into the input shift register, and reception of the rest of the frame proceeds.

As data bits come in from the right, “1”s shift out to the left. When the start bit arrives at the left most position in the shift register,(which is a 9-bit register in Mode 1), it flags the RX Control block to do one last shift, load SBUF and RB8, and set RI. The signal to load SBUF and RB8 and to set RI is generated if, and only if, the following conditions are met at the time the final shift pulse is generated.

- 1) RI=0 and
- 2) Either SM2=0, or the received stop bit = 1

If either of these two conditions is not met, the received frame is irretrievably lost. If both conditions are met, the stop bit goes into RB8, the 8 data bits go into SBUF, and RI is activated. At this time, whether or not the above conditions are met, the unit continues looking for a 1-to-0 transition in RxD.



Serial Port Mode 1

---

When UART1 work in mode 1, its baud rate is variable. UART1 prefer to select Timer 2 as its baud-rate generator, also can choose Timer 1 set by software. So, its baud rate is determined by the T2 or T1 overflow rate.

The Calculating Formula of buad-rate when UART1 select T2 as its baud-rate generator is shown below :

$$\text{Baud-Rate of UART1} = (\text{T2 overflow})/4. \quad \text{Note: the bau-rate is independent of SMOD bit.}$$

If T2 works in 1T mode (AUXR.2/T2x12=1), the T2 overflow =  $\text{SYSclk} / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

$$\text{So, Baud-Rate of UART1} = \text{SYSclk} / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$$

If T2 works in 12T mode (AUXR.2/T2x12=0), the T2 overflow =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

$$\text{So, Baud-Rate of UART1} = \text{SYSclk} / 12 / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$$

RL\_TH2 is the reloaded register of T2H, and RL\_TL2 is the reload register of T2L in above formula.

When UART1 select T1 as its baud-rate generator and T1 is working in mode 0 (16-bit auto-reload timer/counter), The calculating formula of buad-rate is shown below :

$$\text{Baud-Rate of UART1} = (\text{T1 overflow})/4. \quad \text{Note: the bau-rate is independent of SMOD bit.}$$

If T1 works in 1T mode (AUXR.6/T1x12=1), the T1 overflow =  $\text{SYSclk} / (65536 - [\text{RL\_TH1}, \text{RL\_TL1}])$  ;

$$\text{So, Baud-Rate of UART1} = \text{SYSclk} / (65536 - [[\text{RL\_TH1}, \text{RL\_TL1}]] / 4$$

If T1 works in 12T mode (AUXR.6/T1x12=0), the T1 overflow =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH1}, \text{RL\_TL1}])$  ;

$$\text{So, Baud-Rate of UART1} = \text{SYSclk} / 12 / (65536 - [[\text{RL\_TH1}, \text{RL\_TL1}]] / 4$$

RL\_TH1 is the reloaded register of TH1, and RL\_TL1 is the reload register of TL1 in above formula.

When UART1 select T1 as its baud-rate generator and T1 is working in mode 3 (8-bit auto-reload timer/counter), The calculating formula of buad-rate is shown below :

$$\text{Baud-Rate of UART1} = (2^{\text{SMOD}}/32) \times (\text{T1 overflow}).$$

If T1 works in 1T mode (AUXR.6/T1x12=1), the T1 overflow =  $\text{SYSclk} / (256 - \text{TH1})$  ;

$$\text{So, Baud-Rate of UART1} = (2^{\text{SMOD}}/32) \times \text{SYSclk} / (256 - \text{TH1})$$

If T1 works in 12T mode (AUXR.6/T1x12=0), the T1 overflow =  $\text{SYSclk} / 12 / (256 - \text{TH1})$  ;

$$\text{So, Baud-Rate of UART1} = (2^{\text{SMOD}}/32) \times \text{SYSclk} / 12 / (256 - \text{TH1})$$

---

## 8.2.3 Mode 2: 9-Bit UART with Fixed Baud Rate

11 bits are transmitted through TxD or received through RxD. The frame data includes a start bit(0), 8 data bits, a programmable 9th data bit and a stop bit(1). On transmit, the 9th data bit comes from TB8 in SCON. On receive, the 9th data bit goes into RB8 in SCON. The baud rate is programmable to either 1/32 or 1/64 the System clock cycle.

$$\text{Baud rate in mode 2} = (2^{\text{SMOD}}/64) \times \text{SYSclk}$$

Transmission is initiated by any instruction that uses SBUF as a destination register. The “write to SBUF” signal also loads TB8 into the 9<sup>th</sup> bit position of the transmit shift register and flags the TX Control unit that a transmission is requested. Transmission actually happens at the next rollover of divided-by-16 counter. Thus the bit times are synchronized to the divided-by-16 counter, not to the “write to SBUF” signal.

The transmission begins when /SEND is activated, which puts the start bit at TxD. One bit time later, DATA is activated, which enables the output bit of the transmit shift register to TxD. The first shift pulse occurs one bit time after that. The first shift clocks a “1”(the stop bit) into the 9<sup>th</sup> bit position on the shift register. Thereafter, only “0”s are clocked in. As data bits shift out to the right, “0”s are clocked in from the left. When TB8 of the data byte is at the output position of the shift register, then the stop bit is just to the left of TB8, and all positions to the left of that contains “0”s. This condition flags the TX Control unit to do one last shift, then deactivate /SEND and set TI. This occurs at the 11<sup>th</sup> divided-by-16 rollover after “write to SBUF”.

Reception is initiated by a 1-to-0 transition detected at RxD. For this purpose, RxD is sampled at a rate of 16 times whatever baud rate has been established. When a transition is detected, the divided-by-16 counter is immediately reset, and IFFH is written into the input shift register.

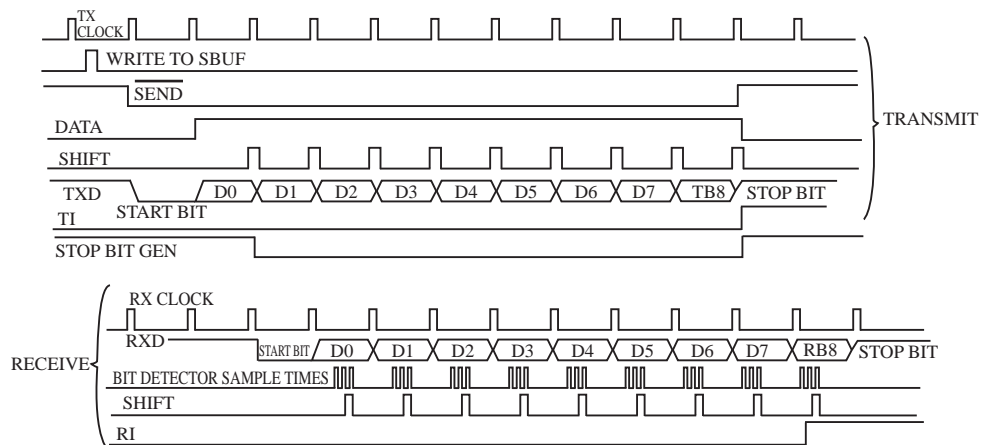
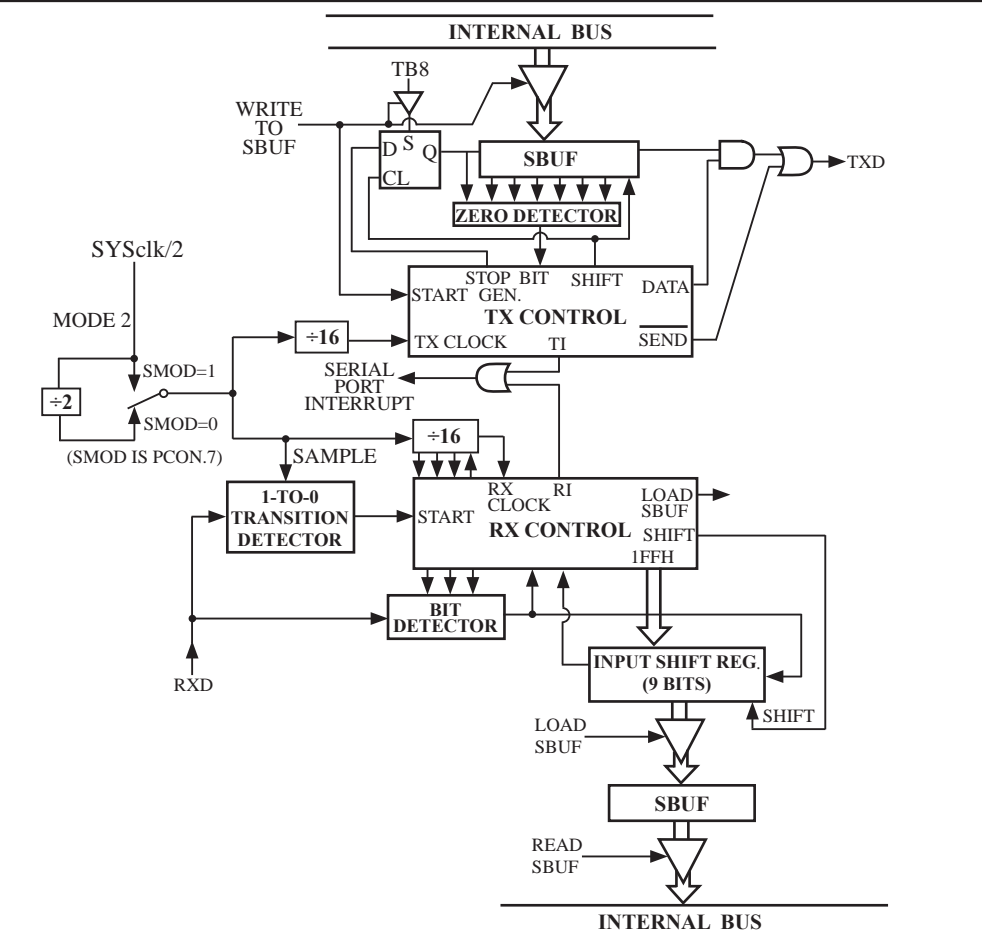
At the 7<sup>th</sup>, 8<sup>th</sup> and 9<sup>th</sup> counter states of each bit time, the bit detector samples the value of RxD. The value accepted is the value that was seen in at least 2 of the 3 samples. This is done to reject noise. In order to reject false bits, if the value accepted during the first bit time is not a 0, the receive circuits are reset and the unit continues looking for another 1-to-0 transition. If the start bit is valid, it is shifted into the input shift register, and reception of the rest of the frame proceeds.

As data bits come in from the right, “1”s shift out to the left. When the start bit arrives at the leftmost position in the shift register,(which is a 9-bit register in Mode-2 and 3), it flags the RX Control block to do one last shift, load SBUF and RB8, and set RI. The signal to load SBUF and RB8 and to set RI is generated if, and only if, the following conditions are met at the time the final shift pulse is generated.:

- 1) RI=0 and
- 2) Either SM2=0, or the received 9<sup>th</sup> data bit = 1

If either of these two conditions is not met, the received frame is irretrievably lost. If both conditions are met, the stop bit goes into RB8, the first 8 data bits go into SBUF, and RI is activated. At this time, whether or not the above conditions are met, the unit continues looking for a 1-to-0 transition at the RxD input.

Note that the value of received stop bit is irrelevant to SBUF, RB8 or RI.



**Serial Port Mode 2**

---

## 8.2.4 Mode 3: 9-Bit UART with Variable Baud Rate

Mode 3 is the same as mode 2 except the baud rate is variable.

When UART1 work in mode 3, it prefer to select Timer 2 as its baud-rate generator, also can choose Timer 1 set by software. So, its baud rate is determined by the T2 or T1 overflow rate.

The Calculating Formula of buad-rate when UART1 select T2 as its baud-rate generator is shown below :

Baud-Rate of UART1 = (T2 overflow)/4. **Note: the bau-rate is independent of SMOD bit.**

If T2 works in 1T mode (AUXR.2/T2x12=1), the T2 overflow =  $\text{SYSclk} / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

So, Baud-Rate of UART1 =  $\text{SYSclk} / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$

If T2 works in 12T mode (AUXR.2/T2x12=0), the T2 overflow =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

So, Baud-Rate of UART1 =  $\text{SYSclk} / 12 / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$

RL\_TH2 is the reloaded register of T2H, and RL\_TL2 is the reload register of T2L in above formula.

When UART1 select T1 as its baud-rate generator and T1 is working in mode 0 (16-bit auto-reload timer/counter), The calculating formula of buad-rate is shown below :

Baud-Rate of UART1 = (T1 overflow)/4. **Note: the bau-rate is independent of SMOD bit.**

If T1 works in 1T mode (AUXR.6/T1x12=1), the T1 overflow =  $\text{SYSclk} / (65536 - [\text{RL\_TH1}, \text{RL\_TL1}])$  ;

So, Baud-Rate of UART1 =  $\text{SYSclk} / (65536 - [[\text{RL\_TH1}, \text{RL\_TL1}]] / 4$

If T1 works in 12T mode (AUXR.6/T1x12=0), the T1 overflow =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH1}, \text{RL\_TL1}])$  ;

So, Baud-Rate of UART1 =  $\text{SYSclk} / 12 / (65536 - [[\text{RL\_TH1}, \text{RL\_TL1}]] / 4$

RL\_TH1 is the reloaded register of TH1, and RL\_TL1 is the reload register of TL1 in above formula.

When UART1 select T1 as its baud-rate generator and T1 is working in mode 3 (8-bit auto-reload timer/counter), The calculating formula of buad-rate is shown below :

Baud-Rate of UART1 =  $(2^{\text{SMOD}}/32) \times (\text{T1 overflow})$ .

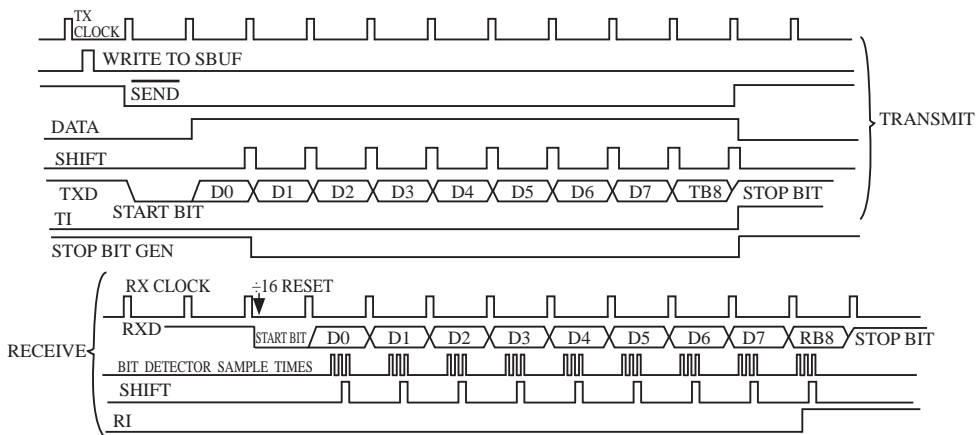
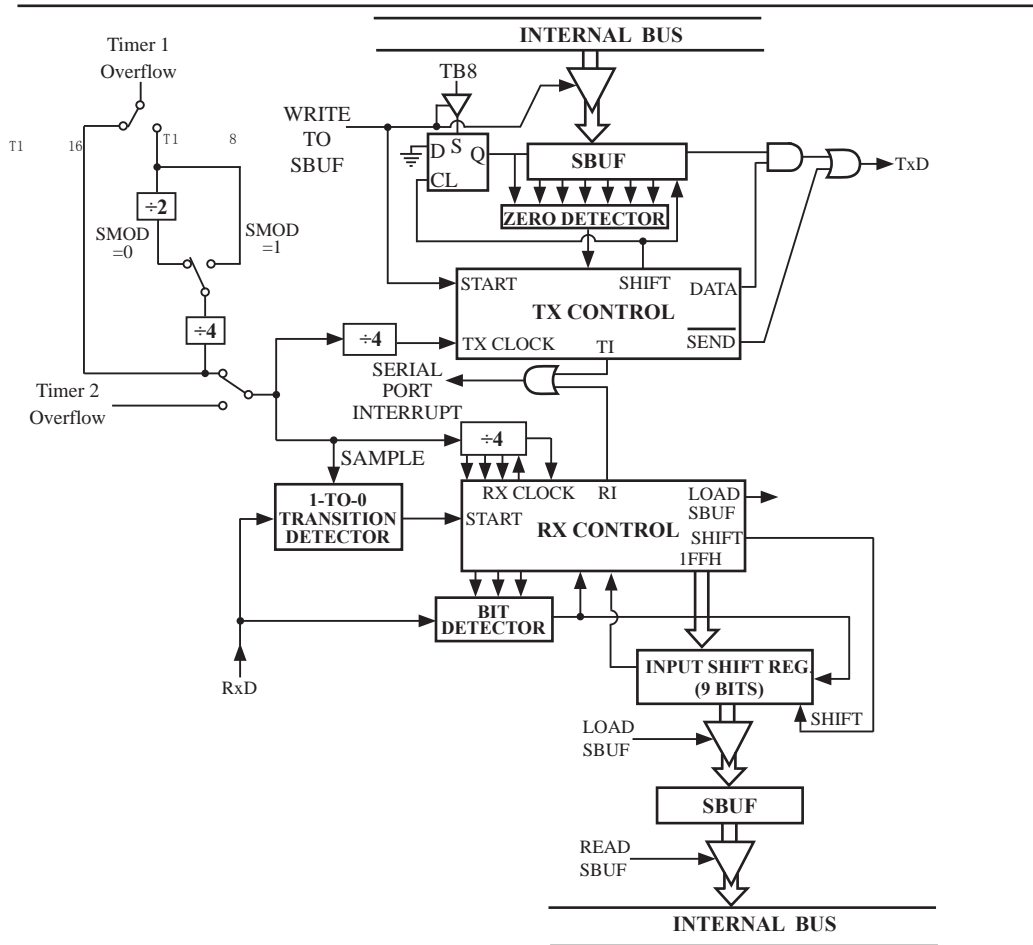
If T1 works in 1T mode (AUXR.6/T1x12=1), the T1 overflow =  $\text{SYSclk} / (256 - \text{TH1})$  ;

So, Baud-Rate of UART1 =  $(2^{\text{SMOD}}/32) \times \text{SYSclk} / (256 - \text{TH1})$

If T1 works in 12T mode (AUXR.6/T1x12=0), the T1 overflow =  $\text{SYSclk} / 12 / (256 - \text{TH1})$  ;

So, Baud-Rate of UART1 =  $(2^{\text{SMOD}}/32) \times \text{SYSclk} / 12 / (256 - \text{TH1})$

In all four modes, transmission is initiated by any instruction that use SBUF as a destination register. Reception is initiated in mode 0 by the condition RI = 0 and REN = 1. Reception is initiated in the other modes by the incoming start bit with 1-to-0 transition if REN=1.



Serial Port Mode 3

---

## 8.3 Baud Rates Setting of UART1 and Demo Program

The baud rate in Mode 0 is fixed:

$$\begin{aligned} \text{Mode 0 Baud Rate} &= \frac{\text{SYSclk}}{12} && \text{when AUXR.5/UART\_M0x6 = 0} \\ \text{or} &= \frac{\text{SYSclk}}{2} && \text{when AUXR.5/UART\_M0x6 = 1} \end{aligned}$$

The baud rate in Mode 2 depends on the value of bit SMOD in Special Function Register PCON. If SMOD = 0 (which is the value on reset), the baud rate is  $\frac{1}{64}$  the System clock cycle. If SMOD = 1, the baud rate is  $\frac{1}{32}$  the System clock cycle .

$$\text{Mode 2 Baud Rate} = \frac{2^{\text{SMOD}}}{64} \times (\text{SYSclk})$$

In the STC15 series MCU, the baud rates in Modes 1 and 3 are determined by Timer 1 or Timer 2 overflow rate. The baud rate in Mode 1 and 3 are variable:

The calculating formula of baud-rate when UART1 select T2 as its baud-rate generator is shown below :

$$\text{Baud-Rate of UART1} = (\text{T2 overflow})/4. \quad \text{Note: the bau-rate is independent of SMOD bit.}$$

If T2 works in 1T mode (AUXR.2/T2x12=1), the T2 overflow = SYSclk / ( 65536 - [RL\_TH2, RL\_TL2] ) ;

$$\text{So, Baud-Rate of UART1} = \text{SYSclk} / ( 65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$$

If T2 works in 12T mode (AUXR.2/T2x12=0), the T2 overflow = SYSclk / 12 / ( 65536 - [RL\_TH2, RL\_TL2] ) ;

$$\text{So, Baud-Rate of UART1} = \text{SYSclk} / 12 / ( 65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$$

RL\_TH2 is the reloaded register of T2H, and RL\_TL2 is the reload register of T2L in above formula.

When UART1 select T1 as its baud-rate generator and T1 is working in mode 0 (16-bit auto-reload timer/counter), The calculating formula of baud-rate is shown below :

$$\text{Baud-Rate of UART1} = (\text{T1 overflow})/4. \quad \text{Note: the bau-rate is independent of SMOD bit.}$$

If T1 works in 1T mode (AUXR.6/T1x12=1), the T1 overflow = SYSclk / ( 65536 - [RL\_TH1, RL\_TL1] ) ;

$$\text{So, Baud-Rate of UART1} = \text{SYSclk} / ( 65536 - [[\text{RL\_TH1}, \text{RL\_TL1}]] / 4$$

If T1 works in 12T mode (AUXR.6/T1x12=0), the T1 overflow = SYSclk / 12 / ( 65536 - [RL\_TH1, RL\_TL1] ) ;

$$\text{So, Baud-Rate of UART1} = \text{SYSclk} / 12 / ( 65536 - [[\text{RL\_TH1}, \text{RL\_TL1}]] / 4$$

RL\_TH1 is the reloaded register of TH1, and RL\_TL1 is the reload register of TL1 in above formula.

When UART1 select T1 as its baud-rate generator and T1 is working in mode 3 (8-bit auto-reload timer/counter), The calculating formula of baud-rate is shown below :

$$\text{Baud-Rate of UART1} = ( 2^{\text{SMOD}}/32 ) \times (\text{T1 overflow}).$$

If T1 works in 1T mode (AUXR.6/T1x12=1), the T1 overflow = SYSclk / ( 256 - TH1 ) ;

$$\text{So, Baud-Rate of UART1} = ( 2^{\text{SMOD}}/32 ) \times \text{SYSclk} / ( 256 - \text{TH1} )$$

If T1 works in 12T mode (AUXR.6/T1x12=0), the T1 overflow = SYSclk / 12 / ( 256 - TH1 ) ;

$$\text{So, Baud-Rate of UART1} = ( 2^{\text{SMOD}}/32 ) \times \text{SYSclk} / 12 / ( 256 - \text{TH1} )$$



Now take UART1 selecting T1 as its baud-rate generator for example.

When T1 is used as the baud rate generator, the T1 interrupt should be disabled in this application. The T1 itself can be configured for either “timer” or “counter” operation, and in any of its 3 running modes. In the most typical applications, it is configured for “timer” operation, in the auto-reload mode (high nibble of TMOD = 0010B).

One can achieve very low baud rate with Timer 1 by leaving the Timer 1 interrupt enabled, and configuring the Timer to run as a 16-bit timer (high nibble of TMOD = 0001B), and using the Timer 1 interrupt to do a 16-bit software reload.

The following figure lists various commonly used baud rates and how they can be obtained from Timer 1.

Baud Rate	System clock Frequency SYSclk	SMOD	Timer 1		
			C/T	Mode	Reload Value
Mode 0 MAX:1MHZ	12MHZ	X	X	X	X
Mode 2 MAX:375K	12MHZ	1	X	X	X
Mode 1,3:62.5K	12MHZ	1	0	2	FFH
19.2K	11.059MHZ	1	0	2	FDH
9.6K	11.059MHZ	0	0	2	FDH
4.8K	11.059MHZ	0	0	2	FAH
2.4K	11.059MHZ	0	0	2	F4H
1.2K	11.059MHZ	0	0	2	E8H
137.5	11.986MHZ	0	0	2	1DH
110	6MHZ	0	0	2	72H
110	12MHZ	0	0	1	FEEDH

**Timer 1 Generated Commonly Used Baud Rates**

Initialize the baud rate :

```

:
MOV    TMOD, #20H           ;0010,0000 set T1 for 8-bit auto-reload timer/counter
MOV    TH1,  #xxH          ;set T1 preload value
MOV    TL1,  #xxH
SETB   TR1                 ;Start to run T1
MOV    PCON, #80H          ;SMOD=1
MOV    SCON, #50H          ;UART1 in mode 1, 8-bit UART with variable baud-rate
:

```

The above program segment can achieve the set of T1 and UART operation mode.

---

## 8.4 Demo Program of UART1 (C and ASM)

### 8.4.1 Demo Program using T2 as UART1 Baud-Rate Generator (C&ASM)

#### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using Timer/Counter 2 as UART1 baud-rate generator -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

typedef unsigned char      BYTE;
typedef unsigned int      WORD;

#define FOSC  18432000L           //System frequency
#define BAUD  115200             //UART1 baud-rate

#define NONE_PARITY        0     //none parity
#define ODD_PARITY        1     //odd parity
#define EVEN_PARITY       2     //even parity
#define MARK_PARITY       3     //mark parity
#define SPACE_PARITY      4     //space parity

#define PARITYBIT EVEN_PARITY    //define the parity bit

sfr  AUXR  =  0x8e;             //Auxiliary register
sfr  T2H   =  0xd6;
sfr  T2L   =  0xd7;

sbit  P22  =  P2^2;

bit busy;

void SendData(BYTE dat);
void SendString(char *s);
```

---

```

void main()
{
#if (PARITYBIT == NONE_PARITY)

        SCON = 0x50; //8-bit variable baud-rate
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
        SCON = 0xda; //9-bit variable baud-rate,
        //the parity bit is initialized for 1
#elif (PARITYBIT == SPACE_PARITY)
        SCON = 0xd2; //9-bit variable baud-rate,
        //the parity bit is initialized for 0
#endif

        T2L = (65536 - (FOSC/4/BAUD)); //Set the preload value
        T2H = (65536 - (FOSC/4/BAUD))>>8;
        AUXR = 0x14; //T2 in 1T mode, and run T2
        AUXR |= 0x01; //select T2 as UART1 baud-rate generator
        ES = 1; //enable UART1 interrupt
        EA = 1;

        SendString("STC15W4K32S4\r\nUart Test !\r\n");
        while(1);
}

/*-----
UART Interrupt Service Routine
-----*/
void Uart() interrupt 4 using 1
{
    if (RI)
    {
        RI = 0; //clear RI
        P0 = SBUF; //serial data is shown in P0
        P22 = RB8; //P2.2 display the parity bit
    }
    if (TI)
    {
        TI = 0; //clear TI
        busy = 0; //clear busy flag
    }
}

/*-----
Send UART data
-----*/
void SendData(BYTE dat)
{
    while (busy); //wait to finish sending the previous data
    ACC = dat; //access to the parity bit ---- P (PSW.0)
}

```

---

---

```

    if (P)
    {
        #if (PARITYBIT == ODD_PARITY)

            TB8 = 0; //the parity bit is set for 0
        #elif (PARITYBIT == EVEN_PARITY)
            TB8 = 1; //the parity bit is set for 1
        #endif
    }
    else
    {
        #if (PARITYBIT == ODD_PARITY)
            TB8 = 1; //the parity bit is set for 1
        #elif (PARITYBIT == EVEN_PARITY)
            TB8 = 0; //the parity bit is set for 0
        #endif
    }
    busy = 1;
    SBUF = ACC;
}

/*-----
Send string
-----*/
void SendString(char *s)
{
    while (*s)
    {
        SendData(*s++);
    }
}

```

---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using Timer/Counter 2 as UART1 baud-rate generator -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#define NONE_PARITY 0 //none parity
#define ODD_PARITY 1 //odd parity
#define EVEN_PARITY 2 //even parity
#define MARK_PARITY 3 //mark parity
#define SPACE_PARITY 4 //space parity

#define PARITYBIT EVEN_PARITY //define the parity bit

//-----

AUXR EQU 08EH //Auxiliary register
T2H DATA 0D6H
T2L DATA 0D7H

//-----
BUSY BIT 20H.0
//-----
ORG 0000H
LJMP MAIN

ORG 0023H
LJMP UART_ISR

//-----
ORG 0100H
MAIN:
CLR BUSY
CLR EA
MOV SP, #3FH

#if (PARITYBIT == NONE_PARITY)
MOV SCON, #50H //8-bit variable baud-rate
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
```

---

```

        MOV     SCON, #0DAH                //9-bit variable baud-rate
                                           //the parity bit is initialized for 1
#elif (PARITYBIT == SPACE_PARITY)
        MOV     SCON, #0D2H                //9-bit variable baud-rate
                                           //the parity bit is initialized for 0
#endif
//-----
        MOV     T2L,  #0D8H                //Set the preload value (65536-18432000/4/115200)
        MOV     T2H,  #0FFH
        MOV     AUXR, #14H                //T2 in 1T mode, and run T2
        ORL    AUXR, #01H                //select T2 as UART1 baud-rate generator
        SETB   ES                          //enable UART1 interrupt
        SETB   EA

        MOV     DPTR, #TESTSTR
        LCALL  SENDSTRING

        SJMP   $
;-----
TESTSTR:
        DB "STC15W4K32S4 Uart1 Test !",0DH,0AH,0

;-----
;*/-----
;UART Interrupt Service Routine
;-----*/
UART_ISR:
        PUSH   ACC
        PUSH   PSW
        JNB    RI,    CHECKTI
        CLR    RI                //clear RI
        MOV    P0,    SBUF        //serial data is shown in P0
        MOV    C,     RB8
        MOV    P2.2,  C          //P2.2 display the parity bit
CHECKTI:
        JNB    TI,    ISR_EXIT
        CLR    TI                //clear TI
        CLR    BUSY            //clear busy flag
ISR_EXIT:
        POP    PSW
        POP    ACC
        RETI

;-----
;*/-----
;Send UART data
;-----*/
SENDDATA:
        JB     BUSY,  $          //wait to finish sending the previous data
        MOV    ACC,  A          //access to the parity bit ---- P (PSW.0)
        JNB    P,     EVEN1INACC

```

---

---

```

ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
    CLR    TB8                //the parity bit is set for 0
#elif (PARITYBIT == EVEN_PARITY)
    SETB   TB8                //the parity bit is set for 1
#endif
    SJMP   PARITYBITOK
EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
    SETB   TB8                //the parity bit is set for 1
#elif (PARITYBIT == EVEN_PARITY)
    CLR    TB8                //the parity bit is set for 0
#endif
PARITYBITOK:
    SETB   BUSY
    MOV    SBUF, A
    RET

; /*-----
; Send string
; /*-----*/
SENDSTRING:
    CLR    A
    MOVC   A, @A+DPTR
    JZ     STRINGEND
    INC    DPTR
    LCALL  SENDDATA
    SJMP   SENDSTRING
STRINGEND:
    RET
; /*-----
    END

```

---

## 8.4.2 Demo Program using T1 as UART1 Baud-Rate Generator(C&ASM) —— T1 in Mode 0 (16-bit Auto-Reload Timer/Counter)

### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using Timer/Counter 1 as UART1 baud-rate generator -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

typedef unsigned char    BYTE;
typedef unsigned int     WORD;

#define FOSC    18432000L           //System frequency
#define BAUD    115200             //UART1 baud-rate

#define NONE_PARITY      0         //none parity
#define ODD_PARITY       1         //odd parity
#define EVEN_PARITY      2         //even parity
#define MARK_PARITY      3         //mark parity
#define SPACE_PARITY     4         //space parity

#define PARITYBIT EVEN_PARITY     //define the parity bit

sfr    AUXR    =    0x8e;         //Auxiliary register
sbit   P22     =    P2^2;
bit    busy;

void SendData(BYTE dat);
void SendString(char *s);
```



---

```

void main()
{
#if (PARITYBIT == NONE_PARITY)
    SCON = 0x50;                                //8-bit variable baud-rate
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    SCON = 0xda;                                //9-bit variable baud-rate
                                                //the parity bit is initialized for 1
#elif (PARITYBIT == SPACE_PARITY)
    SCON = 0xd2;                                //9-bit variable baud-rate
                                                //the parity bit is initialized for 0
#endif

    AUXR = 0x40;                                //T1 in 1T mode
    TMOD = 0x00;                                //T1 in mode 0 (16-bit auto-reload timer/counter)
    TL1 = (65536 - (FOSC/4/BAUD));              //Set the preload value
    TH1 = (65536 - (FOSC/4/BAUD))>>8;
    TR1 = 1;                                    //start to run T1
    ES = 1;                                     //Enable UART1 interrupt
    EA = 1;

    SendString("STC15W4K32S4\r\nUart Test !\r\n");
    while(1);
}

/*-----
UART Interrupt Service Routine
-----*/
void Uart() interrupt 4 using 1
{
    if (RI)
    {
        RI = 0;                                //clear RI
        P0 = SBUF;                              //serial data is shown in P0
        P22 = RB8;                              //P2.2 display the parity bit
    }
    if (TI)
    {
        TI = 0;                                //clear TI
        busy = 0;                              //clear busy flag
    }
}

/*-----
Send UART data
-----*/

```

---

---

```

void SendData(BYTE dat)
{
    while (busy); //wait to finish sending the previous data
    ACC = dat; //access to the parity bit ---- P (PSW.0)
    if (P)
    {
        #if (PARITYBIT == ODD_PARITY)
            TB8 = 0; //the parity bit is set for 0
        #elif (PARITYBIT == EVEN_PARITY)
            TB8 = 1; //the parity bit is set for 1
        #endif
    }
    else
    {
        #if (PARITYBIT == ODD_PARITY)
            TB8 = 1; //the parity bit is set for 1
        #elif (PARITYBIT == EVEN_PARITY)
            TB8 = 0; //the parity bit is set for 0
        #endif
    }
    busy = 1;
    SBUF = ACC;
}

/*-----
Send string
-----*/
void SendString(char *s)
{
    while (*s)
    {
        SendData(*s++);
    }
}

```

---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using Timer/Counter 1 as UART1 baud-rate generator -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#define NONE_PARITY 0 //none parity
#define ODD_PARITY 1 //odd parity
#define EVEN_PARITY 2 //even parity
#define MARK_PARITY 3 //mark parity
#define SPACE_PARITY 4 //space parity

#define PARITYBIT EVEN_PARITY //define the parity bit

//-----

AUXR EQU 08EH //Auxiliary register
BUSY BIT 20H.0
//-----

ORG 0000H
LJMP MAIN

ORG 0023H
LJMP UART_ISR
//-----

ORG 0100H
MAIN:
CLR BUSY
CLR EA
MOV SP, #3FH

#if (PARITYBIT == NONE_PARITY)
MOV SCON, #50H //8-bit variable baud-rate
```

---

```

#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    MOV    SCON, #0DAH           //9-bit variable baud-rate, the parity bit is initialized for 1
#elif (PARITYBIT == SPACE_PARITY)
    MOV    SCON, #0D2H           //9-bit variable baud-rate, the parity bit is initialized for 0
#endif

//-----
    MOV    AUXR, #40H           //T1 in 1T mode
    MOV    TMOD, #00H           //T1 in mode 0 (16-bit auto-reload timer/counter)
    MOV    TL1, #0D8H           //Set the preload value (65536-18432000/4/115200)
    MOV    TH1, #0FFH
    SETB   TR1                   //start to run T1
    SETB   ES                     //Enable UART1 interrupt
    SETB   EA

    MOV    DPTR, #TESTSTR
    LCALL  SENDSTRING

    SJMP   $

;-----
TESTSTR:
    DB "STC15W4K32S4 Uart1 Test !",0DH,0AH,0

; /*-----
; UART Interrupt Service Routine
; ----- */
UART_ISR:
    PUSH   ACC
    PUSH   PSW
    JNB    RI,    CHECKTI
    CLR    RI           //clear RI
    MOV    P0,    SBUF   //serial data is shown in P0
    MOV    C,     RB8
    MOV    P2.2,  C       //P2.2 display the parity bit

CHECKTI:
    JNB    TI,    ISR_EXIT
    CLR    TI           //clear TI
    CLR    BUSY       //clear busy flag

ISR_EXIT:
    POP    PSW
    POP    ACC
    RETI

```

---

---

```

; /*-----
; Send serial data
; -----*/
SENDATA:
    JB     BUSY, $           //wait to finish sending the previous data
    MOV    ACC,  A           //access to the parity bit ---- P (PSW.0)
    JNB    P,    EVEN1INACC
ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
    CLR    TB8               //the parity bit is set for 0
#elif (PARITYBIT == EVEN_PARITY)
    SETB   TB8               //the parity bit is set for 1
#endif
    SJMP   PARITYBITOK
EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
    SETB   TB8               //the parity bit is set for 1
#elif (PARITYBIT == EVEN_PARITY)
    CLR    TB8               //the parity bit is set for 0
#endif
PARITYBITOK:
    SETB   BUSY
    MOV    SBUF, A
    RET

; /*-----
; Send string
; -----*/
SENDSTRING:
    CLR    A
    MOVC   A,    @A+DPTR
    JZ     STRINGEND
    INC    DPTR
    LCALL  SENDATA
    SJMP   SENDSTRING
STRINGEND:
    RET

//-----

    END

```

---

---

## 8.4.3 Demo Program using T1 as UART1 Baud-Rate Generator(C&ASM) —— T1 in Mode 2 (8-bit Auto-Reload Timer/Counter)

### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using 8-bit auto-reload timer/counter 1 as UART1 baud-rate generator ---*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

#include "intrins.h"

typedef unsigned char    BYTE;
typedef unsigned int     WORD;

#define FOSC    18432000L           //system frequency
#define BAUD    115200             //baud-rate

#define NONE_PARITY      0        //none parity
#define ODD_PARITY       1        //odd parity
#define EVEN_PARITY      2        //even parity
#define MARK_PARITY      3        //mark parity
#define SPACE_PARITY     4        //space parity

#define PARITYBIT EVEN_PARITY      //define the parity bit

sfr    AUXR    =    0x8e;          //Auxiliary register
sbit   P22     =    P2^2;
bit    busy;

void SendData(BYTE dat);
void SendString(char *s);
```

---

```

void main()
{
#if (PARITYBIT == NONE_PARITY)
    SCON = 0x50;                //8-bit variable baud-rate
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    SCON = 0xda;                //9-bit variable baud-rate, the parity bit is initialized for 1
#elif (PARITYBIT == SPACE_PARITY)
    SCON = 0xd2;                //9-bit variable baud-rate, the parity bit is initialized for 0
#endif

    AUXR = 0x40;                //T1 in 1T mode
    TMOD = 0x20;                //T1 in mode2 (8-bit auto-reload timer/counter)
    TL1 = (256 - (FOSC/32/BAUD)); //set the preload value
    TH1 = (256 - (FOSC/32/BAUD));
    TR1 = 1;                    //run T1
    ES = 1;                      //enable UART1 interrupt
    EA = 1;

    SendString("STC15W4K32S4\r\nUart Test !\r\n");
    while(1);
}

/*-----
UART Interrupt Service Routine
-----*/
void Uart() interrupt 4 using 1
{
    if (RI)
    {
        RI = 0;                //clear RI
        P0 = SBUF;              //serial data is shown in P0
        P22 = RB8;              //P2.2 display parity bit
    }
    if (TI)
    {
        TI = 0;                //clear TI
        busy = 0;               //clear busy flag
    }
}

/*-----
Send UART data
-----*/

```

---

---

```

void SendData(BYTE dat)
{
    while (busy);           //wait to finish sending the previous data
    ACC = dat;              //access to the parity bit ---- P (PSW.0)

    if (P)
    {
        #if (PARITYBIT == ODD_PARITY)
            TB8 = 0;        //the parity bit is set for 0
        #elif (PARITYBIT == EVEN_PARITY)
            TB8 = 1;        //the parity bit is set for 1
        #endif
    }

    else
    {
        #if (PARITYBIT == ODD_PARITY)
            TB8 = 1;        //the parity bit is set for 1
        #elif (PARITYBIT == EVEN_PARITY)
            TB8 = 0;        //the parity bit is set for 0
        #endif
    }
    busy = 1;
    SBUF = ACC;             //write the data into SBUF of UART
}

/*-----
Send string
-----*/
void SendString(char *s)
{
    while (*s)
    {
        SendData(*s++);
    }
}

```



---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using 8-bit auto-reload timer/counter 1 as UART1 baud-rate generator -*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#define NONE_PARITY 0 //none parity
#define ODD_PARITY 1 //odd parity
#define EVEN_PARITY 2 //even parity
#define MARK_PARITY 3 //mark parity
#define SPACE_PARITY 4 //space parity

#define PARITYBIT EVEN_PARITY //define the parity bit

//-----
AUXR EQU 08EH //Auxiliary register
BUSY BIT 20H.0
//-----
ORG 0000H
LJMP MAIN

ORG 0023H
LJMP UART_ISR
//-----
ORG 0100H
MAIN:
CLR BUSY
CLR EA
MOV SP, #3FH

#if (PARITYBIT == NONE_PARITY)
MOV SCON, #50H //8-bit variable baud-rate
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
```

---

```

MOV    SCON, #0DAH                //9-bit variable baud-rate, the parity bit is initialized for 1
#elif (PARITYBIT == SPACE_PARITY)
    MOV    SCON, #0D2H            //9-bit variable baud-rate, the parity bit is initialized for 0
#endif
//-----
    MOV    AUXR, #40H             //T1 in 1T mode
    MOV    TMOD, #20H            //T1 in mode2 (8-bit auto-reload timer/counter)
    MOV    TL1, #0FBH            //set the preload value (256-18432000/32/115200)
    MOV    TH1, #0FBH
    SETB   TR1                    //run T1
    SETB   ES                     //enable UART1 interrupt
    SETB   EA

    MOV    DPTR, #TESTSTR
    LCALL  SENDSTRING

    SJMP   $

;-----
TESTSTR:
    DB "STC15W4K32S4 Uart1 Test !",0DH,0AH,0

;/*-----
;UART Interrupt Service Routine
;-----*/
UART_ISR:
    PUSH   ACC
    PUSH   PSW
    JNB    RI,    CHECKTI
    CLR    RI                //clear RI
    MOV    P0,    SBUF        //serial data is shown in P0
    MOV    C,     RB8
    MOV    P2.2,  C           //P2.2 display parity bit
CHECKTI:
    JNB    TI,    ISR_EXIT
    CLR    TI                //clear TI
    CLR    BUSY        //clear busy flag
ISR_EXIT:
    POP    PSW
    POP    ACC
    RETI

;/*-----
;Send UART data
;-----*/

```

---

---

```

SENDATA:
    JB     BUSY, $           //wait to finish sending the previous data
    MOV    ACC, A           //access to the parity bit ---- P (PSW.0)
    JNB    P,     EVEN1INACC
ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
    CLR    TB8             //the parity bit is set for 0
#elif (PARITYBIT == EVEN_PARITY)
    SETB   TB8             //the parity bit is set for 1
#endif
    SJMP   PARITYBITOK
EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
    SETB   TB8             //the parity bit is set for 1
#elif (PARITYBIT == EVEN_PARITY)
    CLR    TB8             //the parity bit is set for 0
#endif
PARITYBITOK:
    SETB   BUSY
    MOV    SBUF, A         //write the data into SBUF of UART
    RET

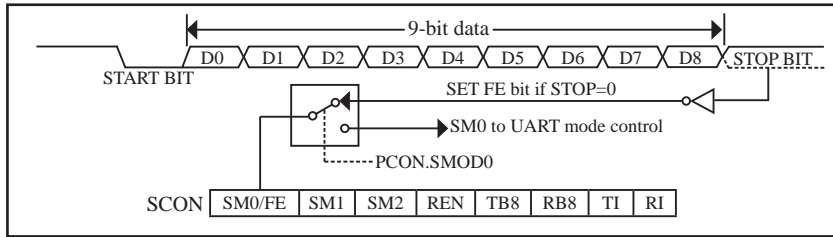
; /*-----
; Send string
; /*-----*/
SENDSTRING:
    CLR    A
    MOVC   A,     @A+DPTR
    JZ     STRINGEND
    INC    DPTR
    LCALL  SENDATA
    SJMP   SENDSTRING
STRINGEND:
    RET
; /*-----
    END

```

---

## 8.5 Frame Error Detection

When used for frame error detect, the UART looks for missing stop bits in the communication. A missing bit will set the FE bit in the SCON register. The FE bit shares the SCON.7 bit with SM0 and the function of SCON.7 is determined by PCON.6 (SMOD0). If SMOD0 is set then SCON.7 functions as FE. SCON.7 functions as SM0 when SMOD0 is cleared. When used as FE, SCON.7 can only be cleared by software. Refer to the following figure.



UART Frame Error Detection

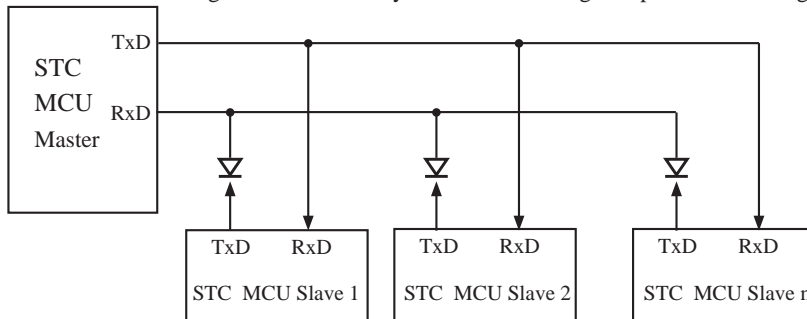
## 8.6 Multiprocessor Communications

Modes 2 and 3 have a special provision for multiprocessor communications. In these modes, 9 data bits are received. The 9th one goes into RB8. Then comes a stop bit. The port can be programmed such that when the stop bit is received, the serial port interrupt will be activated only if RB8 = 1. This feature is enabled by setting bit SM2 in SCON. A way to use this feature in multiprocessor systems is as follows.

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the 9th bit is 1 in an address byte and 0 in a data byte. With SM2 = 1, no slave will be interrupted by a data byte. An address byte, however, will interrupt all slaves, so that each slave can examine the received byte and see if it is being addressed. The addressed slave will clear its SM2 bit and prepare to receive the data bytes that will be coming. The slaves that weren't being addressed leave their SM2s set and go on about their business, ignoring the coming data bytes.

SM2 has no effect in Mode 0, and in Mode 1 can be used to check the validity of the stop bit. In a Mode 1 reception, if SM2 = 1, the receive interrupt will not be activated unless a valid stop bit is received.

The following figure shows a master MCU on the network, which can instruct individual slave devices to set or clear their SM2 bits to alter the configuration so that they either receive or ignore particular messages.



## 8.7 Automatic Address Recognition of UART1

### 8.7.1 Special Function Registers about Automatic Address Recognition

Symbol	Description	Address	Bit Address and Symbol								Value after Power-on or Reset
			MSB							LSB	
SCON	Serial Control	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000 0000B
SBUF	Serial Buffer	99H									xxxx xxxxB
SADEN	Slave Address Mask	B9H									0000 0000B
SADDR	Slave Address	A9H									0000 0000B

#### 1. Serial Port 1 (UART1) Control Register: SCON

SCON: Serial port Control Register (Bit-Addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	name	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

FE : Framing Error bit. The SMOD0 bit must be set to enable access to the FE bit

0 : The FE bit is not cleared by valid frames but should be cleared by software.

1 : This bit set by the receiver when an invalid stop bit id detected.

SM0,SM1 : Serial Port Mode Bit 0/1.

SM0	SM1	Mode	Description	Baud Rate
0	0	Mode 0	synchronous shift serial mode: 8-bit shift register	If UART_M0x6 = 0, baud rate = SYSclk/12, If UART_M0x6 = 1, baud rate = SYSclk / 2
0	1	Mode 1	8-bit UART, baud-rate variable	If UART1 select Timer 2 or Timer 1 (as 16-bit auto-reload timer), baud rate= (T1 or T2 overflow )/4. If UART1 select Timer 1 (as 8-bit auto-reload timer), baud rate = ( 2 <sup>SMOD</sup> /32 )×(T1 overflow)
1	0	Mode 2	9-bit UART	( 2 <sup>SMOD</sup> / 64 ) x SYSclk SYSclk is system clock frequency
1	1	Mode 3	9-bit UART, baud-rate variable	If UART1 select Timer 2 or Timer 1 (as 16-bit auto-reload timer), baud rate= (T1 or T2 overflow )/4. If UART1 select Timer 1 (as 8-bit auto-reload timer), baud rate = ( 2 <sup>SMOD</sup> /32 )×(T1 overflow)

SM2 : Enable the automatic address recognition feature in mode 2 and 3. If SM2=1, RI will not be set unless the received 9th data bit is 1, indicating an address, and the received byte is a Given or Broadcast address. In mode1, if SM2=1 then RI will not be set unless a valid stop Bit was received, and the received byte is a Given or Broadcast address. In mode 0, SM2 should be 0.

REN : When set enables serial reception.

TB8 : The 9th data bit which will be transmitted in mode 2 and 3.

RB8 : In mode 2 and 3, the received 9th data bit will go into this bit.

TI : Transmit interrupt flag. Set by hardware when a byte of data has been transmitted by UART0 (after the 8th bit in 8-bit UART Mode, or at the beginning of the STOP bit in 9-bit UART Mode). When the UART0 interrupt is enabled, setting this bit causes the CPU to vector to the UART0 interrupt service routine. This bit must be cleared manually by software.

RI : Receive interrupt flag. Set to '1' by hardware when a byte of data has been received by UART0 (set at the STOP bit sampling time). When the UART0 interrupt is enabled, setting this bit to '1' causes the CPU to vector to the UART0 interrupt service routine. This bit must be cleared manually by software.

## 2. SBUF: Serial port 1 Data Buffer register (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SBUF	99H	name								

It is used as the buffer register in transmission and reception. The serial port buffer register (SBUF) is really two 8-bit registers. Writing to SBUF loads data to be transmitted, and reading SBUF accesses received data. These are two separate and distinct registers, the transmit write-only register, and the receive read-only register.

## 3. Slave Address Control registers SADEN and SADDR

SADEN: Slave Address Mask register

SADDR: Slave Address register

SADDR register is combined with SADEN register to form Given/Broadcast Address for automatic address recognition. In fact, SADEN function as the "mask" register for SADDR register. The following is the example for it.

$$\begin{array}{rcl}
 \text{SADDR} & = & 1100\ 0000 \\
 \text{SADEN} & = & \underline{1111\ 1101} \\
 \text{Given} & = & 1100\ 00x0 \longrightarrow \text{The Given slave address will be checked except bit 1 is} \\
 & & \text{treated as "don't care"}.
 \end{array}$$

The Broadcast Address for each slave is created by taking the logical OR of SADDR and SADEN. Zero in this result is considered as "don't care" and a Broadcast Address of all " don't care". This disables the automatic address detection feature.

---

## 8.7.2 Instruction of Automatic Address Recognition

Automatic Address Recognition is a feature which allows the UART to recognize certain addresses in the serial bit stream by using hardware to make the comparisons. This feature saves a great deal of software overhead by eliminating the need for the software to examine every serial address which passes by the serial port. This feature is enabled by setting the SM2 bit in SCON. In the 9-bit UART modes, Mode 2 and Mode 3, the Receive interrupt flag (RI) will be automatically set when the received byte contains either the "Given" address or the "Broadcast" address. The 9-bit mode requires that the 9<sup>th</sup> information bit is a "1" to indicate that the received information is an address and not data.

The 8-bit mode is called Mode 1. In this mode the RI flag will be set if SM2 is enabled and the information received has a valid stop bit following the 8 address bits and the information is either a Given or Broadcast address.

Mode 0 is the Shift Register mode and SM2 is ignored.

Using the Automatic Address Recognition feature allows a master to selectively communicate with one or more slaves by invoking the given slave address or addresses. All of the slaves may be contacted by using the broadcast address. Two special function registers are used to define the slave's address, SADDR, and the address mask, SADEN. SADEN is used to define which bits in the SADDR are to be used and which bits are "don't care". The SADEN mask can be logically ANDed with the SADDR to create the "Given" address which the master will use for addressing each of the slaves. Use of the Given address allows multiple slaves to be recognized which excluding others. The following examples will help to show the versatility of this scheme :

Slave 0            SADDR = 1100 0000  
                  SADEN = 1111 1101  
                  GIVEN = 1100 00x0

Slave 1            SADDR = 1100 0000  
                  SADEN = 1111 1110  
                  GIVEN = 1100 000x

In the previous example SADDR is the same and the SADEN data is used to differentiate between the two slaves. Slave 0 requires a "0" in bit 0 and it ignores bit 1. Slave 1 requires a "0" in bit 1 and bit 0 is ignored. A unique address for slave 0 would be 11000010 since slave 1 requires a "0" in bit 1. A unique address for slave 1 would be 11000001 since a "1" in bit 0 will exclude slave 0. Both slaves can be selected at the same time by an address which has bit 0=0 (for slave 0) and bit 1 =0 (for slave 1). Thus, both could be addressed with 11000000.

In a more complex system the following could be used to select slaves 1 and 2 while excluding slave 0:

Slave 0            SADDR = 1100 0000  
                  SADEN = 1111 1001  
                  GIVEN = 1100 0xx0

---

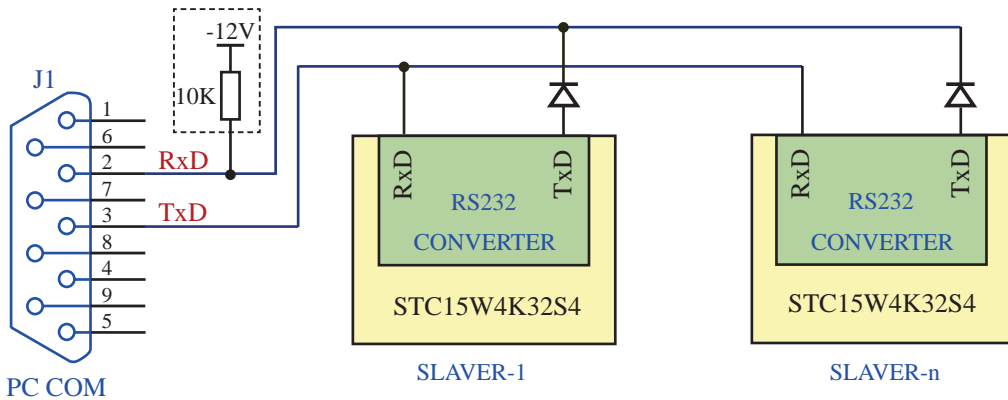
Slave 1	SADDR = 1110 0000 SADEN = 1111 1010 GIVEN = 1110 0x0x
Slave 2	SADDR = 1110 0000 SADEN = 1111 1100 GIVEN = 1110 00xx

In the above example the differentiation among the 3 slaves is in the lower 3 address bits. Slave 0 requires that bit0 = 0 and it can be uniquely addressed by 11100110. Slave 1 requires that bit 1=0 and it can be uniquely addressed by 11100101. Slave 2 requires that bit 2=0 and its unique address is 11100011. To select Slave 0 and 1 and exclude Slave 2, use address 11100100, since it is necessary to make bit2=1 to exclude Slave 2.

The Broadcast Address for each slave is created by taking the logic OR of SADDR and SADEN. Zeros in this result are treated as don't cares. In most cases, interpreting the don't cares as ones, the broadcast address will be FF hexadecimal.

Upon reset SADDR and SADEN are loaded with "0"s. This produces a given address of all "don't cares as well as a Broadcast address of all "don't cares". This effectively disables the Automatic Addressing mode and allows the microcontroller to use traditional 8051-type UART drivers which do not make use of this feature.

The test method of demo program is shown below.



The test method of demo program is shown below.

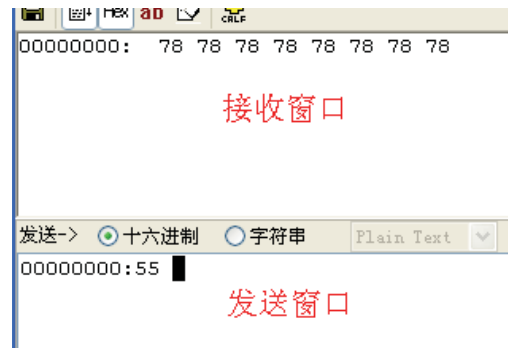
- 1, Firstly, connect two MCU to PC COM according to the above figure.
- 2, Burn the code in which have defined the slave as 0 ("#define SLAVER 0") onto the SLAVER-1 MCU. And burn the code in which have defined the slave as 1 ("#define SLAVER 1") onto the SLAVER-2 MCU



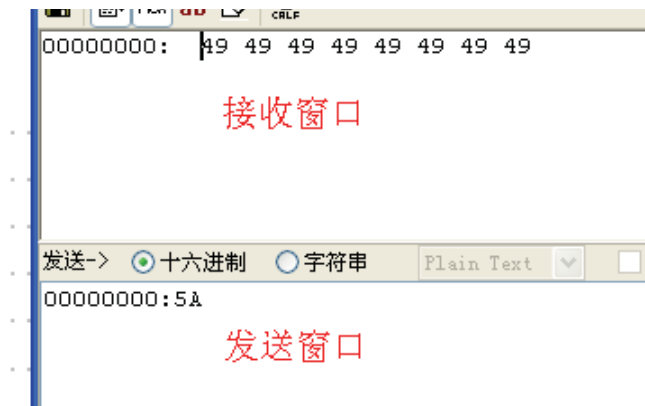
3, Open the COM Helper in PC, set the serial port according to the figure. Note the parity bit.



4, If users send the data 0x55 by COM Helper, Salve 1 would be enabled and answer eight 0x78. See the following figure.



5, If users send the data 0x5a by COM Helper again, Salve 2 would be enabled and answer eight 0x49. See the following figure.



---

## 8.7.3 Demo Program of Automatic Address Recognition (C and ASM)

### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of automatic address recognition -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

typedef unsigned char  BYTE;
typedef unsigned int   WORD;

//-----

#define SLAVER          0           //define the number of slave, 0 is Slave 1 and 1 is Slave 2
#if SLAVER == 0
#define SAMASK          0x33       //address mask bit of Slave 1

#define SERADR          0x55       //The address of Slave 1 is xx01,xx01.
#define ACKTST          0x78
#else
#define SAMASK          0x3C       //address mask bit of Slave 2
#define SERADR          0x5A       //The address of Slave 2 is xx01,10xx
#define ACKTST          0x49
#endif

#define URMD 0           //0: select T2 as UART1 baud-rate generator
                        //1: select T1 as UART1 baud-rate generator(T1 as 16-bit auto-relaod timer/counter)
                        //2: select T1 as UART1 baud-rate generator (T1 as 8-bit auto-relaod timer/counter)

sfr T2H = 0xd6;
sfr T2L = 0xd7;
```

---

```

sfr    AUXR    =    0x8e;        //Auxiliary register

sfr    SADDR   =    0xA9;        //Slave Address register
sfr    SADEN   =    0xB9;        //Slave Address Mask register

void InitUart();

char count;

void main()
{
    InitUart();                //Initialize the serial port
    ES = 1;
    EA = 1;
    while (1);
}

/*-----
UART Interrupt Service Routine
-----*/
void Uart() interrupt 4 using 1
{
    if (TI)
    {
        TI = 0;                //clear TI (transmit flag)
        if (count != 0)
        {
            count--;
            SBUF = ACKTST;
        }
        else
        {
            SM2 = 1;
        }
    }
    if (RI)
    {
        RI    =    0;          //Clear RI (receive flag)
        SM2   =    0;
        count =    7;
        SBUF  =    ACKTST;
    }
}

```

---

---

```

/*-----
Initialize the serial port
-----*/
void InitUart()
{
    SADDR =    SERADR;
    SADEN =    SAMASK;
    SCON  =    0xf8;           //set UART1 as 9-bit UART with variable baud-rate
                                //(set TB8 for 1, that easy to communicate with PC directly)

    #if      URMD ==    0
        T2L  =    0xd8;       //Set the proload value of baud-rate
        T2H  =    0xff;       //115200 bps(65536-18432000/4/115200)
        AUXR =    0x14;       //T2 in 1T mode, and run T2
        AUXR |= 0x01;       //select T2 as UART1 baud rate generator
    #elif    URMD ==    1
        AUXR =    0x40;       //T1 in 1T mode
        TMOD =    0x00;       //T1 in mode 0 (16-bit auto-reload timer/counter)
        TL1  =    0xd8;       //Set the proload value of baud-rate
        TH1  =    0xff;       //115200 bps(65536-18432000/4/115200)
        TR1  =    1;         //run T1
    #else
        TMOD =    0x20;       //T1 in mode 2 (8-bit auto-reload timer/counter)
        AUXR =    0x40;       //T1 in 1T mode
        TH1 = TL1 = 0xfb;     //115200 bps(256 - 18432000/32/115200)
        TR1  =    1;
    #endif
}

```

---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of automatic address recognition -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#define SLAVER      0           //define the number of slave, 0 is Slave 1 and 1 is Slave 2

#if SLAVER == 0
#define SAMASK     0x33       //the address mask bit of Slave 1
#define SERADR     0x55       //The address of Slave 1 is xx01,xx01
#define ACKTST     0x78
#else
#define SAMASK     0x3C       //the address mask bit of Slave 2
#define SERADR     0x5A       //The address of Slave 2 is xx01,10xx
#define ACKTST     0x49
#endif

#define URMD 0           //0: select T2 as UART1 baud-rate generator
                        //1: select T1 as UART1 baud-rate generator(T1 as 16-bit auto-relaod timer/counter)
                        //2: select T1 as UART1 baud-rate generator (T1 as 8-bit auto-relaod timer/counter)

T2H   DATA  0D6H
T2L   DATA  0D7H
AUXR  DATA  08EH           //Auxiliary register

SADDR DATA  0A9H           //Slave Address register
SADEN DATA  0B9H           //Slave Address Mask register

COUNT DATA  20H
//-----
        ORG    0000H
        LJMP  MAIN
```

---

```

        ORG    0023H
        LJMP   UART_ISR
//-----
        ORG    0100H
MAIN:
        MOV    SP,    #3FH
        LCALL  INIT_UART           //Initialize the serial port
        SETB   ES
        SETB   EA
        SJMP   $

//-----
//UART Interrupt Service Routine
UART_ISR:
        PUSH  PSW
        PUSH  ACC
        JNB   TI,    CHK_RX
        CLR   TI           //clear TI (transmit flag)
        MOV   A,    COUNT
        JZ    RESTART
        DEC   COUNT
        MOV   SBUF, #ACKTST
        JMP   UREXIT
RESTART:
        SETB  SM2
        JMP   UREXIT
CHK_RX:
        JNB   RI,    UREXIT
        CLR   RI           //Clear RI (receive flag)
        CLR   SM2
        MOV   SBUF, #ACKTST
        MOV   COUNT, #7
UREXIT:
        POP   ACC
        POP   PSW
        RETI

/*-----
Initialize serial port
-----*/
INIT_UART:
        MOV   SADDR, #SERADR
        MOV   SADEN, #SAMASK
        MOV   SCON,  #0F8H           //set UART1 as 9-bit UART with variable baud-rate,
                                     //(set TB8 for 1, that easy to communicate with PC directly)

```

---

---

```

#if    URMD    ==    0
      MOV     T2L,    #0D8H           //Set the proload value of baud-rate
                                           //(65536-18432000/4/115200)

      MOV     T2H,    #0FFH
      MOV     AUXR,   #14H           //T2 in 1T mode, and run T2
      ORL     AUXR,   #01H           //select T2 as UART1 baud rate generator
#elif  URMD    ==    1
      MOV     AUXR,   #40H           //T1 in 1T mode
      MOV     TMOD,   #00H           //T1 in mode 0 (16-bit auto-reload timer/counter)
      MOV     TL1,    #0D8H           //Set the proload value of baud-rate
                                           //(65536-18432000/4/115200)

      MOV     TH1,    #0FFH
      SETB    TR1                    ///run T1
#else
      MOV     TMOD,   #20H           //T1 in mode 2 (8-bit auto-reload timer/counter)
      MOV     AUXR,   #40H           //T1 in 1T mode
      MOV     TL1,    #0FBH           //115200 bps(256 - 18432000/32/115200)
      MOV     TH1,    #0FBH
      SETB    TR1
#endif
      RET

//-----

      END

```

---

## 8.8 Special Function Registers about Serial Port 2 (UART2)

Symbol	Description	Address	Bit Address and Symbol								Value after Power-on or Reset
			MSB							LSB	
S2CON	Serial 2 Control register	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100 0000B
S2BUF	Serial 2 Buffer	9BH									xxxx xxxxB
T2H	The high 8-bit of Timer 2 register	D6H									0000 0000B
T2L	The low 8-bit of Timer 2 register	D7H									0000 0000B
AUXR	Auxiliary register	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C $\bar{T}$	T2x12	EXTRAM	S1ST2	0000 0001B
IE	Interrupt Enable	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000 0000B
IE2	Interrupt Enable 2	AFH		ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000 0000B
IP2	Interrupt Priority 2 Low	B5H	-	-	-	PX4	PPWMFD	PPWM	PSPI	PS2	xxx0 0000B
P_SW2	Peripheral function switch register	BAH	EAXSFR	0	0	0	-	S4_S	S3_S	S2_S	0000 x000B

There are several special function registers which should be understood by users before using the secondary UART.

### 1. Serial port 2 Control register: S2CON (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S2CON	9AH	name	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI

S2SM0 : Serial Port 2 Mode Select Bit.

S2SM0	Operation Modes	Description	Baud Rate
0	Mode 0	8-bit UART, baud-rate variable	(T2 overflow rate) / 4
1	Mode 1	9-bit UART, baud-rate variable	(T2 overflow rate) / 4

If AUXR.2/T2x12 = 0, T2 overflow rate =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

If AUXR.2/T2x12 = 1, T2 overflow rate =  $\text{SYSclk} / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  .

RL\_TH2 is the reloaded register of T2H, and RL\_TL2 is the reload register of T2L in above formula.

B6 : Reserved

S2SM2 : Enable the automatic address recognition feature. In mode 1, if S2SM2=1, S2RI will not be set unless the received 9th data bit is 1, indicating an address, and the received byte is a Given or Broadcast address. In mode 0, if S2SM2=1 then S2RI will not be set unless a valid stop bit was received, and the received byte is a Given or Broadcast address.

S2REN : Enable the serial port reception.

When set, enable serial reception.

When clear, disable the secondary serial port reception.



---

S2TB8 : The 9th data bit which will be transmitted in mode 1.

S2RB8 : In mode 1, the received 9th data bit will go into this bit.

S2TI : Transmit interrupt flag. After a transmitting has been finished, the hardware will set this bit.

S2RI : Receive interrupt flag. After reception has been finished, the hardware will set this bit.

## 2. Serial port 2 Data Buffer register: S2BUF

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S2BUF	9BH	name								

It is used as the buffer register in transmission and reception. This SFR accesses two registers; a transmit shift register and a receive latch register. When data is written to S2BUF, it goes to the transmit shift register and is held for serial transmission. Writing a byte to S2BUF initiates the transmission. A read of S2BUF returns the contents of the receive latch.

## 3. UART2 only can select T2 as its Baud-Rate Generator ----- T2 register: T2H and T2L

The Timer 2 register T2H (address:D6H) and T2L (address:D7H) are used to load the time value.

**Note:** UART2 only can choose Timer 2 as its its baud-rate generator. UART1 prefer to select Timer 2 as its baud-rate generator, also can choose Timer 1 set by software. UART3 and UART4 default to selecting Timer 2 as their baud-rate generator. UART3 and UART4 also can choose Timer 3 and Timer 4 as their baud-rate generator respectively.

## 4. Timer 2 Control Bit ---- T2R, T2\_C/T, T2x12

AUXR: Auxiliary register (Non bit-addressable)

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

B4 - T2R Timer 2 Run control bit

- 0 : not run Timer 2;
- 1 : run Timer 2.

B3 - T2\_C/T: Counter or timer 2 selector

- 0 : as Timer (namely count on internal system clock)
- 1 : as Counter (namely count on the external pulse input from T2/P3.1)

B2 - T2x12 : Timer 2 clock source bit.

- 0 : The clock source of Timer 2 is SYSclk/12.
- 1 : The clock source of Timer 2 is SYSclk/1.

If T2 is used as the baud-rate generator of UART1 or UART2, T1x12 will decide whether UART1 or UART2 is 1T or 12T.

For STC15 series, Secondary UART (S2) only can select Timer 2 as its baud-rate generator. While UART1 not only can Timer 2, but also can select Timer 1 as its baud-rate generator.

## 5. Registers bits related with UART2 (S2) Interrupt : EA, ES2 and PS2

IE2: Interrupt Enable 2 Rsgister (Non bit-addressable)

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

ES2 : Serial port 2 (UART2) interrupt enable bit.

If ES2 = 0, UART2 interrupt would be disabled.

If ES2 = 1, UART2 interrupt would be enabled.

IE: Interrupt Enable Rsgister (Bit-addressable)

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

EA : disables all interrupts.

If EA = 0, no interrupt will be acknowledged.

If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

IP2: Interrupt Priority Register (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP2	B5H	name	-	-	-	PX4	PPWMFD	PPWM	PSPI	PS2

PS2 : Serial Port 2 (UART2) interrupt priority control bit.

if PS2=0, UART2 interrupt is assigned lowest priority (priority 0).

if PS2=1, UART2 interrupt is assigned highest priority (priority 1).

## 6. UART2 Switch Control bit: S2\_S / P\_SW2.0

P\_SW2 : Peripheral function switch register (Non bit-addressable)

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
P_SW2	BAH	Peripheral function switch register	EAXSFR	-	-	-	-	S4_S	S3_S	S2_S	0000 x000B

UART2/S2 can be switched in 2 groups of pins by selecting the control bit S2_S.	
S2_S	UART2/S2 can be switched between P1 and P4
0	UART2/S2 on [P1.0/RxD2,P1.1/TxD2]
1	UART2/S2 on [P4.6/RxD2_2,P4.7/TxD2_2]

UART3/S3 can be switched in 2 groups of pins by selecting the control bit S3_S.	
S3_S	UART3/S3 can be switched between P0 and P5
0	UART3/S3 on [P0.0/RxD3,P0.1/TxD3]
1	UART3/S3 on [P5.0/RxD3_2,P5.1/TxD3_2]

UART4/S4 can be switched in 2 groups of pins by selecting the control bit S4_S.	
S4_S	UART4/S4 can be switched between P0 and P5
0	UART4/S4 on [P0.2/RxD4,P0.3/TxD4]
1	UART4/S4 on [P5.2/RxD4_2,P5.3/TxD4_2]

---

## 8.9 UART2 Operation Modes

The serial port 2 (UART2) can be operated in two different modes which are configured by setting S2SM0 in SFR S2CON. Mode 0 and Mode 1 are both asynchronous communication.

### 8.9.1 Mode 0 : 8-bit UART2 with Variable Baud-Rate

10 bits are transmitted through Tx/D2/P1.1(TxD2\_2/P4.7) or received through Rx/D2/P1.0(RxD2\_2/P4.6). The frame data includes a start bit(0), 8 data bits and a stop bit(1). One receive, the stop bit goes into S2RB8 in SFR – S2CON. The baud rate is determined by the T2 overflow rate.

UART2 only can select T2 as its **baud-rate generator**. The calculating formula of UART2 baud-rate is shown below :

$$\text{Baud-Rate of UART2} = (\text{T2 overflow})/4.$$

If T2 works in 1T mode (AUXR.2/T2x12=1), the T2 overflow =  $\text{SYSclk} / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

$$\text{So, Baud-Rate of UART2} = \text{SYSclk} / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$$

If T2 works in 12T mode (AUXR.2/T2x12=0), the T2 overflow =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

$$\text{So, Baud-Rate of UART2} = \text{SYSclk} / 12 / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$$

RL\_TH2 is the reloaded register of T2H, and RL\_TL2 is the reload register of T2L in above formula.

### 8.9.2 Mode 3: 9-bit UART2 with Variable Baud-Rate

11 bits are transmitted through Tx/D2/P1.1(TxD2\_2/P4.7) or received through Rx/D2/P1.0(RxD2\_2/P4.6). The frame data includes a start bit(0), 8 data bits, a programmable 9th bit and a stop bit(1). On transmit, the 9th data bit comes from S2TB8 in S2CON. On receive, the 9th data bit goes into S2RB8 in S2CON. The baud rate is determined by the T2 overflow rate.

UART2 only can select T2 as its **baud-rate generator**. The calculating formula of UART2 baud-rate is shown below :

$$\text{Baud-Rate of UART2} = (\text{T2 overflow})/4.$$

If T2 works in 1T mode (AUXR.2/T2x12=1), the T2 overflow =  $\text{SYSclk} / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

$$\text{So, Baud-Rate of UART2} = \text{SYSclk} / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$$

If T2 works in 12T mode (AUXR.2/T2x12=0), the T2 overflow =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

$$\text{So, Baud-Rate of UART2} = \text{SYSclk} / 12 / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$$

RL\_TH2 is the reloaded register of T2H, and RL\_TL2 is the reload register of T2L in above formula.

*\* When S2\_S bit in P\_SW2 register is set, the function of UART2 is redirected to P4.6 for RXD2 and P4.7 for TXD2.*

---

## 8.10 Demo Program of UART2 (C and ASM)

### ----- Using Timer 2 as UART2 Baud-Rate Generator

#### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using Timer 2 as UART2 baud-rate generator -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/
```

```
//suppose the frequency of test chip is 18.432MHz
```

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

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

```
typedef unsigned char    BYTE;
typedef unsigned int     WORD;
```

```
#define FOSC 18432000L           //System frequency
```

```
#define BAUD 115200             //UART2 baud-rate
```

```
#define TM (65536 - (FOSC/4/BAUD))
```

```
#define NONE_PARITY 0           //none parity
```

```
#define ODD_PARITY 1            //odd parity
```

```
#define EVEN_PARITY 2           //even parity
```

```
#define MARK_PARITY 3           //mark parity
```

```
#define SPACE_PARITY 4          //space parity
```

```
#define PARITYBIT EVEN_PARITY   //define the parity bit
```

```
sfr AUXR = 0x8e;               //Auxiliary register
```

```
sfr S2CON = 0x9a;              //UART2 Control register
```

```
sfr S2BUF = 0x9b;              //UART2 data register
```

```
sfr T2H = 0xd6;
```

```
sfr T2L = 0xd7;
```

```
sfr IE2 = 0xaf;                //Interrupt Enable register 2
```

---

```

#define S2RI    0x01           //S2CON.0
#define S2TI    0x02           //S2CON.1

#define S2RB8   0x04           //S2CON.2
#define S2TB8   0x08           //S2CON.3

bit    busy;

void SendData(BYTE dat);
void SendString(char *s);

void main()
{
#if (PARITYBIT == NONE_PARITY)
    S2CON = 0x50;                //8-bit variable baud-rate
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    S2CON = 0xda;                //9-bit variable baud-rate
                                //the parity bit is initialized for 1
#elif (PARITYBIT == SPACE_PARITY)
    S2CON = 0xd2;                //9-bit variable baud-rate
                                //the parity bit is initialized for 0
#endif

    T2L = TM;                    //Set the preload value
    T2H = TM>>8;
    AUXR = 0x14;                 //T2 in 1T mode, and run T2
    IE2 = 0x01;                 //enable UART2 interrupt
    EA = 1;

    SendString("STC15W4K32S4\r\nUart2 Test !\r\n");
    while(1);
}

/*-----
UART2 Interrupt Service Routine
-----*/
void Uart2() interrupt 8 using 1
{
    if (S2CON & S2RI)
    {
        S2CON &= ~S2RI;          //clear S2RI
        P0 = S2BUF;              //serial data is shown in P0
        P2 = (S2CON & S2RB8);    //P2.2 display the parity bit
    }
}

```

---

---

```

        if (S2CON & S2TI)
        {
            S2CON &= ~S2TI;           //clear S2TI
            busy = 0;                 //clear busy flag
        }
    }

/*-----
Send UART data
-----*/
void SendData(BYTE dat)
{
    while (busy);                   //wait to finish sending the previous data
    ACC = dat;                       //access to the parity bit ---- P (PSW.0)
    if (P)
    {
        #if (PARITYBIT == ODD_PARITY)
            S2CON &= ~S2TB8;         //the parity bit is set for 0
        #elif (PARITYBIT == EVEN_PARITY)
            S2CON |= S2TB8;          //the parity bit is set for 1
        #endif
    }
    else
    {
        #if (PARITYBIT == ODD_PARITY)
            S2CON |= S2TB8;          //the parity bit is set for 1
        #elif (PARITYBIT == EVEN_PARITY)
            S2CON &= ~S2TB8;        //the parity bit is set for 0
        #endif
    }
    busy = 1;
    S2BUF = ACC;
}

/*-----
Send sting
-----*/
void SendString(char *s)
{
    while (*s)
    {
        SendData(*s++);
    }
}

```

---

---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using Timer 2 as UART2 baud-rate generator -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz
#define NONE_PARITY 0 //none parity
#define ODD_PARITY 1 //odd parity
#define EVEN_PARITY 2 //even parity
#define MARK_PARITY 3 //mark parity
#define SPACE_PARITY 4 //space parity

#define PARITYBIT EVEN_PARITY //define the parity bit

//-----

AUXR EQU 08EH //Auxiliary register
S2CON EQU 09AH //UART2 Control register
S2BUF EQU 09BH //UART2 data register
T2H DATA 0D6H
T2L DATA 0D7H
IE2 EQU 0AFH //Interrupt Enable register 2

S2RI EQU 01H //S2CON.0
S2TI EQU 02H //S2CON.1
S2RB8 EQU 04H //S2CON.2
S2TB8 EQU 08H //S2CON.3

//-----
BUSY BIT 20H.0
//-----

ORG 0000H
LJMP MAIN

ORG 0043H
LJMP UART2_ISR
```

---

```

//-----
    ORG    0100H
MAIN:
    CLR    BUSY
    CLR    EA
    MOV    SP,    #3FH
#if (PARITYBIT == NONE_PARITY)
    MOV    S2CON, #50H                //8-bit variable baud-rate
#elif (PARITYBIT == ODD_PARITY) || (PARITYBIT == EVEN_PARITY) || (PARITYBIT == MARK_PARITY)
    MOV    S2CON, #0DAH                //9-bit variable baud-rate
                                        //the parity bit is initialized for 1
#elif (PARITYBIT == SPACE_PARITY)
    MOV    S2CON, #0D2H                //9-bit variable baud-rate
                                        //the parity bit is initialized for 0
#endif

//-----
    MOV    T2L,    #0D8H                //Set the preload value (65536-18432000/4/115200)
    MOV    T2H,    #0FFH
    MOV    AUXR,   #14H                //T2 in 1T mode, and run T2
    ORL    IE2,    #01H                //enable UART2 interrupt
    SETB   EA

    MOV    DPTR,   #TESTSTR
    LCALL  SENDSTRING

    SJMP   $
;-----
TESTSTR:
    DB    "STC15W4K32S4 Uart2 Test !",0DH,0AH,0

;/*-----
;UART2 Interrupt Service Routine
;-----*/
UART2_ISR:
    PUSH   ACC
    PUSH   PSW
    MOV    A,    S2CON                ;read the content of S2CON
    JNB   ACC.0, CHECKTI
    ANL   S2CON, #NOT S2RI            ;clear S2RI
    MOV   P0,    S2BUF                ;serial data is shown in P0
    ANL   A,    #S2RB8                ;
    MOV   P2,    A                    ;P2.2 display the parity bit
CHECKTI:
    MOV   A,    S2CON                ;read the content of S2CON
    JNB   ACC.1, ISR_EXIT
    ANL   S2CON, #NOT S2TI            ;clear S2RI
    CLR   BUSY                        ;clear busy flag

```

---



---

```

ISR_EXIT:
    POP    PSW
    POP    ACC
    RETI

;/*-----
;Send UART data
;-----*/
SENDDATA:
    JB     BUSY, $           //wait to finish sending the previous data
    MOV    ACC, A           //access to the parity bit ---- P (PSW.0)
    JNB   P, EVEN1INACC
ODD1INACC:
#if (PARITYBIT == ODD_PARITY)
    ANL   S2CON, #NOT S2TB8 //the parity bit is set for 0
#elif (PARITYBIT == EVEN_PARITY)
    ORL   S2CON, #S2TB8    //the parity bit is set for 1
#endif
    SJMP  PARITYBITOK
EVEN1INACC:
#if (PARITYBIT == ODD_PARITY)
    ORL   S2CON, #S2TB8    //the parity bit is set for 1
#elif (PARITYBIT == EVEN_PARITY)
    ANL   S2CON, #NOT S2TB8 //the parity bit is set for 0
#endif
PARITYBITOK:
    SETB  BUSY
    MOV   S2BUF, A
    RET

;/*-----
;Send sting
//-----*/
SENDSTRING:
    CLR   A
    MOVC  A, @A+DPTR
    JZ    STRINGEND
    INC   DPTR
    LCALL SENDDATA
    SJMP  SENDSTRING
STRINGEND:
    RET
//-----
    END

```

---

## 8.11 Special Function Registers about Serial Port 3 (UART3)

Symbol	Description	Address	Bit Address and Symbol								Value after Power-on or Reset
			MSB				LSB				
S3CON	Serial 3 Control register	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000 0000B
S3BUF	Serial 3 Buffer	ADH									xxxx xxxxB
T2H	The high 8-bit of Timer 2 register	D6H									0000 0000B
T2L	The low 8-bit of Timer 2 register	D7H									0000 0000B
AUXR	Auxiliary register	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C $\bar{T}$	T2x12	EXTRAM	S1ST2	0000 0001B
T3H	The high 8-bit of Timer 3 register	D4H									0000 0000B
T3L	The low 8-bit of Timer 3 register	D5H									0000 0000B
T4T3M	T4 and T3 Mode control register	D1H	T4R	T4_C $\bar{T}$	T4x12	T4CLKO	T3R	T3_C $\bar{T}$	T3x12	T3CLKO	0000 0000B
IE2	Interrupt Enable 2	AFH		ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000 0000B
P_SW2	Peripheral function switch register	BAH	EAXSFR	0	0	0	-	S4_S	S3_S	S2_S	0000 x000B

There are several special function registers which should be understood by users before using the UART3.

### 1. Serial port 3 Control register: S3CON (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S3CON	ACH	name	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI

S3SM0 : Serial Port 3 Mode Select Bit.

S3SM0	Operation Modes	Description	Baud Rate
0	Mode 0	8-bit UART, baud-rate variable	(T2 overflow rate) / 4 or (T3 overflow rate) / 4
1	Mode 1	9-bit UART, baud-rate variable	(T2 overflow rate) / 4 or (T3 overflow rate) / 4

If AUXR.2/T2x12 = 0, T2 overflow rate =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;  
 If AUXR.2/T2x12 = 1, T2 overflow rate =  $\text{SYSclk} / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  .  
 RL\_TH2 is the reloaded register of T2H, and RL\_TL2 is the reload register of T2L in above formula.

If T4T3M.1/T3x12 = 0, T3 overflow rate =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH3}, \text{RL\_TL3}])$  ;  
 If T4T3M.1/T3x12 = 1, T3 overflow rate =  $\text{SYSclk} / (65536 - [\text{RL\_TH3}, \text{RL\_TL3}])$  .  
 RL\_TH3 is the reloaded register of T3H, and RL\_TL3 is the reload register of T3L in above formula.

S3ST3 : the control bit that UART3 select Timer 3 as its baud-rate generator.

- 0 : Select Timer 2 as the baud-rate generator of UART3
- 1 : Select Timer 3 as the baud-rate generator of UART3.

S3SM2 : Enable the automatic address recognition feature. In mode 1, if S3SM2=1, S3RI will not be set unless the received 9th data bit is 1, indicating an address, and the received byte is a Given or Broadcast address. In mode 0, if S3SM2=1 then S3RI will not be set unless a valid stop bit was received, and the received byte is a Given or Broadcast address.

S3REN : Enable the serial port reception.  
 When set, enable serial reception.  
 When clear, disable the secondary serial port reception.

S3TB8 : The 9th data bit which will be transmitted in mode 1.

S3RB8 : In mode 1, the received 9th data bit will go into this bit.

S3TI : Transmit interrupt flag. After a transmitting has been finished, the hardware will set this bit.

S3RI : Receive interrupt flag. After reception has been finished, the hardware will set this bit.

## 2. Serial port 3 Data Buffer register: S3BUF

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S3BUF	ADH	name								

It is used as the buffer register in transmission and reception. This SFR accesses two registers; a transmit shift register and a receive latch register. When data is written to S3BUF, it goes to the transmit shift register and is held for serial transmission. Writing a byte to S3BUF initiates the transmission. A read of S3BUF returns the contents of the receive latch.

## 3. UART3 either can select Timer 2 or Timer 3 as its Baud-Rate Generator ----- T2 register: T2H, T2L and T3 register: T3H, T3L

The Timer 2 register T2H (address:D6H) and T2L (address:D7H) are used to load the time value.

The Timer 3 register T3H (address:D4H) and T3L (address:D5H) are used to load the time value.

**Note:** UART2 only can choose Timer 2 as its its baud-rate generator. UART1 prefer to select Timer 2 as its baud-rate generator, also can choose Timer 1 set by software. UART3 and UART4 default to selecting Timer 2 as their baud-rate generator. UART3 and UART4 also can choose Timer 3 and Timer 4 as their baud-rate generator respectively.

## 4. Timer 2 Control Bit ---- T2R, T2\_C/ $\bar{T}$ , T2x12

AUXR: Auxiliary register (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/ $\bar{T}$	T2x12	EXTRAM	S1ST2

B4 - T2R Timer 2 Run control bit

0 : not run Timer 2;

1 : run Timer 2.

---

B3 - T2\_C/T: Counter or timer 2 selector

- 0 : as Timer (namely count on internal system clock)
- 1 : as Counter (namely count on the external pulse input from T2/P3.1)

B2 - T2x12 : Timer 2 clock source bit.

- 0 : The clock source of Timer 2 is SYSclk/12.
- 1 : The clock source of Timer 2 is SYSclk/1.

If T2 is used as the baud-rate generator of UART1 or UART2, T1x12 will decide whether UART1 or UART2 is 1T or 12T.

For STC15 series, Secondary UART (S2) only can select Timer 2 as its baud-rate generator. While UART3 not only can Timer 2, but also can select Timer 3 as its baud-rate generator.

### 5. Timer 3 Control Bit ---- T3R, T3\_C/T, T3x12

T4T3M: T4 and T3 mode control bit (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

B3 - T3R Timer 3 Run control bit

- 0 : not run Timer 3;
- 1 : run Timer 3.

B2 - T3\_C/T: Counter or timer 3 selector

- 0 : as Timer (namely count on internal system clock)
- 1 : as Counter (namely count on the external pulse input from T3/P0.5)

B1 - T3x12 : Timer 3 clock source bit.

- 0 : The clock source of Timer 3 is SYSclk/12.
- 1 : The clock source of Timer 3 is SYSclk/1.

If T3 is used as the baud-rate generator of UART3, T3x12 will decide whether UART3 is 1T or 12T.

### 6. Registers bits related with UART3 (S3) Interrupt : EA, ES3

IE2: Interrupt Enable 2 Rsgister (Non bit-addressable)

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

ES3 : Serial port 3 (UART3) interrupt enable bit.

If ES3 = 0, UART3 interrupt would be disabled.

If ES3 = 1, UART3 interrupt would be enabled.

IE: Interrupt Enable Rsgister (Bit-addressable)

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

EA : disables all interrupts.

If EA = 0, no interrupt will be acknowledged.

If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

## 7. UART3 Switch Control bit: S3\_S / P\_SW2.1

P\_SW2 : Peripheral function switch register (Non bit-addressable)

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
P_SW2	BAH	Peripheral function switch register	EAXSFR	0	0	0	-	S4_S	S3_S	S2_S	0000 x000B

UART3/S3 can be switched in 2 groups of pins by selecting the control bit S3_S.	
S3_S	UART3/S3 can be switched between P0 and P5
0	UART3/S3 on [P0.0/RxD3,P0.1/TxD3]
1	UART3/S3 on [P5.0/RxD3_2,P5.1/TxD3_2]

UART2/S2 can be switched in 2 groups of pins by selecting the control bit S2_S.	
S2_S	UART2/S2 can be switched between P1 and P4
0	UART2/S2 on [P1.0/RxD2,P1.1/TxD2]
1	UART2/S2 on [P4.6/RxD2_2,P4.7/TxD2_2]

UART4/S4 can be switched in 2 groups of pins by selecting the control bit S4_S.	
S4_S	UART4/S4 can be switched between P0 and P5
0	UART4/S4 on [P0.2/RxD4,P0.3/TxD4]
1	UART4/S4 on [P5.2/RxD4_2,P5.3/TxD4_2]

---

## 8.12 UART3 Operation Modes

The serial port 3 (UART3) can be operated in two different modes which are configured by setting S3SM0 in SFR S3CON. Mode 0 and Mode 1 are both asynchronous communication.

### 8.12.1 Mode 0 : 8-bit UART3 with Variable Baud-Rate

10 bits are transmitted through TxD3/P0.1(TxD3/P5.1) or received through RxD3/P0.0(RxD3/P5.0). The frame data includes a start bit(0), 8 data bits and a stop bit(1). One receive, the stop bit goes into S3RB8 in SFR – S3CON. The baud rate is determined by the T2 overflow rate or T3 overflow rate.

UART3 either can select T2 or T3 as its **baud-rate generator**. When UART3 select T2 as its **baud-rate generator** (that is to say S3ST3 / S3SCON.0 = 0), the **calculating formula of UART3 buad-rate is shown below :**

$$\text{Baud-Rate of UART3} = (\text{T2 overflow})/4.$$

If T2 works in 1T mode (AUXR.2/T2x12=1), the T2 overflow =  $\text{SYSclk} / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

$$\text{So, Baud-Rate of UART3} = \text{SYSclk} / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$$

If T2 works in 12T mode (AUXR.2/T2x12=0), the T2 overflow =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

$$\text{So, Baud-Rate of UART3} = \text{SYSclk} / 12 / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$$

RL\_TH2 is the reloaded register of T2H, and RL\_TL2 is the reload register of T2L in above formula.

When UART3 select T3 as its **baud-rate generator** (that is to say S3ST3 / S3SCON.0 = 1), the **calculating formula of UART3 buad-rate is shown below :**

$$\text{Baud-Rate of UART3} = (\text{T3 overflow})/4.$$

If T3 works in 1T mode (T4T3M.1/T3x12=1), the T3 overflow =  $\text{SYSclk} / (65536 - [\text{RL\_TH3}, \text{RL\_TL3}])$  ;

$$\text{So, Baud-Rate of UART3} = \text{SYSclk} / (65536 - [[\text{RL\_TH3}, \text{RL\_TL3}]] / 4$$

If T3 works in 12T mode (T4T3M.1/T3x12=0), the T3 overflow =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH3}, \text{RL\_TL3}])$  ;

$$\text{So, Baud-Rate of UART3} = \text{SYSclk} / 12 / (65536 - [[\text{RL\_TH3}, \text{RL\_TL3}]] / 4$$

RL\_TH3 is the reloaded register of T3H, and RL\_TL3 is the reload register of T3L in above formula.

---

## 8.12.2 Mode 3: 9-bit UART3 with Variable Baud-Rate

11 bits are transmitted through TxD3/P0.1(TxD3/P5.1) or received through RxD3/P0.0(RxD3/P5.0). The frame data includes a start bit(0), 8 data bits, a programmable 9th bit and a stop bit(1). On transmit, the 9th data bit comes from S3TB8 in S3CON. On receive, the 9th data bit goes into S3RB8 in S3CON. The baud rate is determined by the T2 overflow rate or T3 overflow rate.

UART3 either can select T2 or T3 as its baud-rate generator. When UART3 select T2 as its baud-rate generator (that is to say S3ST3 / S3SCON.0 = 0), the calculating formula of UART3 baud-rate is shown below :

$$\text{Baud-Rate of UART3} = (\text{T2 overflow})/4.$$

If T2 works in 1T mode (AUXR.2/T2x12=1), the T2 overflow =  $\text{SYSclk} / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

$$\text{So, Baud-Rate of UART3} = \text{SYSclk} / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$$

If T2 works in 12T mode (AUXR.2/T2x12=0), the T2 overflow =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

$$\text{So, Baud-Rate of UART3} = \text{SYSclk} / 12 / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$$

RL\_TH2 is the reloaded register of T2H, and RL\_TL2 is the reload register of T2L in above formula.

When UART3 select T3 as its baud-rate generator (that is to say S3ST3 / S3SCON.0 = 1), the calculating formula of UART3 baud-rate is shown below :

$$\text{Baud-Rate of UART3} = (\text{T3 overflow})/4.$$

If T3 works in 1T mode (T4T3M.1/T3x12=1), the T3 overflow =  $\text{SYSclk} / (65536 - [\text{RL\_TH3}, \text{RL\_TL3}])$  ;

$$\text{So, Baud-Rate of UART3} = \text{SYSclk} / (65536 - [[\text{RL\_TH3}, \text{RL\_TL3}]] / 4$$

If T3 works in 12T mode (T4T3M.1/T3x12=0), the T3 overflow =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH3}, \text{RL\_TL3}])$  ;

$$\text{So, Baud-Rate of UART3} = \text{SYSclk} / 12 / (65536 - [[\text{RL\_TH3}, \text{RL\_TL3}]] / 4$$

RL\_TH3 is the reloaded register of T3H, and RL\_TL3 is the reload register of T3L in above formula.

*\* When S3\_S bit in P\_SW2 register is set, the function of UART3 is redirected to P4.6 for RXD3 and P4.7 for TXD3.*

## 8.13 Special Function Registers about Serial Port 4 (UART4)

Symbol	Description	Address	Bit Address and Symbol								Value after Power-on or Reset
			MSB				LSB				
S4CON	Serial 4 Control register	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000 0000B
S4BUF	Serial 4 Buffer	85H									xxxx xxxxB
T2H	The high 8-bit of Timer 2 register	D6H									0000 0000B
T2L	The low 8-bit of Timer 2 register	D7H									0000 0000B
AUXR	Auxiliary register	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C $\bar{T}$	T2x12	EXTRAM	S1ST2	0000 0001B
T4H	The high 8-bit of Timer 4 register	D2H									0000 0000B
T4L	The low 8-bit of Timer 4 register	D3H									0000 0000B
T4T3M	T4 and T3 Mode control register	D1H	T4R	T4_C $\bar{T}$	T4x12	T4CLKO	T3R	T3_C $\bar{T}$	T3x12	T3CLKO	0000 0000B
IE2	Interrupt Enable 2	AFH		ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000 0000B
P_SW2	Peripheral function switch register	BAH	EAXSFR	0	0	0	-	S4_S	S3_S	S2_S	0000 x000B

There are several special function registers which should be understood by users before using the UART4.

### 1. Serial port 4 Control register: S4CON (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S4CON	84H	name	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

S4SM0 : Serial Port 4 Mode Select Bit.

S4SM0	Operation Modes	Description	Baud Rate
0	Mode 0	8-bit UART, baud-rate variable	(T2 overflow rate) / 4 or (T4 overflow rate) / 4
1	Mode 1	9-bit UART, baud-rate variable	(T2 overflow rate) / 4 or (T4 overflow rate) / 4

If AUXR.2/T2x12 = 0, T2 overflow rate =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;  
 If AUXR.2/T2x12 = 1, T2 overflow rate =  $\text{SYSclk} / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  .  
 RL\_TH2 is the reloaded register of T2H, and RL\_TL2 is the reload register of T2L in above formula.

If T4T3M.5/T4x12 = 0, T4 overflow rate =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH4}, \text{RL\_TL4}])$  ;  
 If T4T3M.5/T4x12 = 1, T4 overflow rate =  $\text{SYSclk} / (65536 - [\text{RL\_TH4}, \text{RL\_TL4}])$  .  
 RL\_TH4 is the reloaded register of T4H, and RL\_TL4 is the reload register of T4L in above formula.

S4ST4 : the control bit that UART4 select Timer 4 as its baud-rate generator.

- 0 : Select Timer 2 as the baud-rate generator of UART4
- 1 : Select Timer 4 as the baud-rate generator of UART4.



S4SM2 : Enable the automatic address recognition feature. In mode 1, if S4SM2=1, S4RI will not be set unless the received 9th data bit is 1, indicating an address, and the received byte is a Given or Broadcast address. In mode 0, if S4SM2=1 then S4RI will not be set unless a valid stop bit was received, and the received byte is a Given or Broadcast address.

S4REN : Enable the serial port reception.  
 When set, enable serial reception.  
 When clear, disable the secondary serial port reception.

S4TB8 : The 9th data bit which will be transmitted in mode 1.

S4RB8 : In mode 1, the received 9th data bit will go into this bit.

S4TI : Transmit interrupt flag. After a transmitting has been finished, the hardware will set this bit.

S4RI : Receive interrupt flag. After reception has been finished, the hardware will set this bit.

## 2. Serial port 4 Data Buffer register: S4BUF

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
S4BUF	85H	name								

It is used as the buffer register in transmission and reception. This SFR accesses two registers; a transmit shift register and a receive latch register. When data is written to S4BUF, it goes to the transmit shift register and is held for serial transmission. Writing a byte to S4BUF initiates the transmission. A read of S4BUF returns the contents of the receive latch.

## 3. UART4 either can select Timer 2 or Timer 4 as its Baud-Rate Generator ----- T2 register: T2H, T2L and T4 register: T4H, T4L

The Timer 2 register T2H (address:D6H) and T2L (address:D7H) are used to load the time value.

The Timer 4 register T4H (address:D2H) and T4L (address:D3H) are used to load the time value.

**Note:** UART2 only can choose Timer 2 as its its baud-rate generator. UART1 prefer to select Timer 2 as its baud-rate generator, also can choose Timer 1 set by software. UART3 and UART4 default to selecting Timer 2 as their baud-rate generator. UART3 and UART4 also can choose Timer 3 and Timer 4 as their baud-rate generator respectively.

## 4. Timer 2 Control Bit ---- T2R, T2\_C/ $\bar{T}$ , T2x12

AUXR: Auxiliary register (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/ $\bar{T}$	T2x12	EXTRAM	S1ST2

B4 - T2R Timer 2 Run control bit

0 : not run Timer 2;

1 : run Timer 2.

---

B3 - T2\_C/T: Counter or timer 2 selector

0 : as Timer (namely count on internal system clock)

1 : as Counter (namely count on the external pulse input from T2/P3.1)

B2 - T2x12 : Timer 2 clock source bit.

0 : The clock source of Timer 2 is SYSclk/12.

1 : The clock source of Timer 2 is SYSclk/1.

If T2 is used as the baud-rate generator of UART1 or UART2, T1x12 will decide whether UART1 or UART2 is 1T or 12T.

For STC15 series, Secondary UART (S2) only can select Timer 2 as its baud-rate generator. While UART3 not only can Timer 2, but also can select Timer 3 as its baud-rate generator.

## 5. Timer 4 Control Bit ---- T4R, T4\_C/T, T4x12

T4T3M: T4 and T3 mode control bit (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	name	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

B7 - T4R Timer 4 Run control bit

0 : not run Timer 4;

1 : run Timer 4.

B6 - T4\_C/T: Counter or timer 4 selector

0 : as Timer (namely count on internal system clock)

1 : as Counter (namely count on the external pulse input from T4/P0.7)

B5 - T4x12 : Timer 4 clock source bit.

0 : The clock source of Timer 4 is SYSclk/12.

1 : The clock source of Timer 4 is SYSclk/1.

If T4 is used as the baud-rate generator of UART4, T4x12 will decide whether UART4 is 1T or 12T.

## 6. Registers bits related with UART4 (S4) Interrupt : EA, ES4

IE2: Interrupt Enable 2 Rsgister (Non bit-addressable)

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

ES4 : Serial port 4 (UART4) interrupt enable bit.

If ES4 = 0, UART4 interrupt would be disabled.

If ES4 = 1, UART4 interrupt would be enabled.

IE: Interrupt Enable Rsgister (Bit-addressable)

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

EA : disables all interrupts.

If EA = 0, no interrupt will be acknowledged.

If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

## 7. UART3 Switch Control bit: S3\_S / P\_SW2.1

P\_SW2 : Peripheral function switch register (Non bit-addressable)

Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
P_SW2	BAH	Peripheral function switch register	EAXSFR	0	0	0	-	S4_S	S3_S	S2_S	0000 x000B

UART4/S4 can be switched in 2 groups of pins by selecting the control bit S4_S.	
S4_S	UART4/S4 can be switched between P0 and P5
0	UART4/S4 on [P0.2/RxD4,P0.3/TxD4]
1	UART4/S4 on [P5.2/RxD4_2,P5.3/TxD4_2]

UART3/S3 can be switched in 2 groups of pins by selecting the control bit S3_S.	
S3_S	UART3/S3 can be switched between P0 and P5
0	UART3/S3 on [P0.0/RxD3,P0.1/TxD3]
1	UART3/S3 on [P5.0/RxD3_2,P5.1/TxD3_2]

UART2/S2 can be switched in 2 groups of pins by selecting the control bit S2_S.	
S2_S	UART2/S2 can be switched between P1 and P4
0	UART2/S2 on [P1.0/RxD2,P1.1/TxD2]
1	UART2/S2 on [P4.6/RxD2_2,P4.7/TxD2_2]

---

## 8.14 UART4 Operation Modes

The serial port 4 (UART4) can be operated in two different modes which are configured by setting S4SM0 in SFR S4CON. Mode 0 and Mode 1 are both asynchronous communication.

### 8.14.1 Mode 0 : 8-bit UART4 with Variable Baud-Rate

10 bits are transmitted through TxD4/P0.3(TxD4\_2/P5.3) or received through RxD4/P0.2(RxD4\_2/P5.2). The frame data includes a start bit(0), 8 data bits and a stop bit(1). One receive, the stop bit goes into S4RB8 in SFR – S4CON. The baud rate is determined by the T2 overflow rate or T3 overflow rate.

UART4 either can select T2 or T4 as its baud-rate generator. When UART4 select T2 as its baud-rate generator (that is to say S4ST4 / S4SCON.1 = 0), the calculating formula of UART4 baud-rate is shown below :

$$\text{Baud-Rate of UART4} = (\text{T2 overflow})/4.$$

If T2 works in 1T mode (AUXR.2/T2x12=1), the T2 overflow =  $\text{SYSclk} / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

$$\text{So, Baud-Rate of UART4} = \text{SYSclk} / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$$

If T2 works in 12T mode (AUXR.2/T2x12=0), the T2 overflow =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

$$\text{So, Baud-Rate of UART4} = \text{SYSclk} / 12 / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$$

RL\_TH2 is the reloaded register of T2H, and RL\_TL2 is the reload register of T2L in above formula.

When UART4 select T4 as its baud-rate generator (that is to say S4ST4 / S4SCON.1 = 1), the calculating formula of UART4 baud-rate is shown below :

$$\text{Baud-Rate of UART4} = (\text{T4 overflow})/4.$$

If T4 works in 1T mode (T4T3M.5/T4x12=1), the T4 overflow =  $\text{SYSclk} / (65536 - [\text{RL\_TH4}, \text{RL\_TL4}])$  ;

$$\text{So, Baud-Rate of UART4} = \text{SYSclk} / (65536 - [[\text{RL\_TH4}, \text{RL\_TL4}]] / 4$$

If T4 works in 12T mode (T4T3M.5/T4x12=0), the T4 overflow =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH4}, \text{RL\_TL4}])$  ;

$$\text{So, Baud-Rate of UART4} = \text{SYSclk} / 12 / (65536 - [[\text{RL\_TH4}, \text{RL\_TL4}]] / 4$$

RL\_TH4 is the reloaded register of T4H, and RL\_TL4 is the reload register of T4L in above formula.

---

## 8.14.2 Mode 3: 9-bit UART4 with Variable Baud-Rate

11 bits are transmitted through TxD4/P0.3(TxD4\_2/P5.3) or received through RxD4/P0.2(RxD4\_2/P5.2). The frame data includes a start bit(0), 8 data bits, a programmable 9th bit and a stop bit(1). On transmit, the 9th data bit comes from S4TB8 in S4CON. On receive, the 9th data bit goes into S4RB8 in S4CON. The baud rate is determined by the T2 overflow rate or T3 overflow rate.

UART4 either can select T2 or T4 as its baud-rate generator. When UART4 select T2 as its baud-rate generator (that is to say S4ST4 / S4SCON.1 = 0), the calculating formula of UART4 baud-rate is shown below :

$$\text{Baud-Rate of UART4} = (\text{T2 overflow})/4.$$

If T2 works in 1T mode (AUXR.2/T2x12=1), the T2 overflow =  $\text{SYSclk} / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

$$\text{So, Baud-Rate of UART4} = \text{SYSclk} / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$$

If T2 works in 12T mode (AUXR.2/T2x12=0), the T2 overflow =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH2}, \text{RL\_TL2}])$  ;

$$\text{So, Baud-Rate of UART4} = \text{SYSclk} / 12 / (65536 - [[\text{RL\_TH2}, \text{RL\_TL2}]] / 4$$

RL\_TH2 is the reloaded register of T2H, and RL\_TL2 is the reload register of T2L in above formula.

When UART4 select T4 as its baud-rate generator (that is to say S4ST4 / S4SCON.1 = 1), the calculating formula of UART4 baud-rate is shown below :

$$\text{Baud-Rate of UART4} = (\text{T4 overflow})/4.$$

If T4 works in 1T mode (T4T3M.5/T4x12=1), the T4 overflow =  $\text{SYSclk} / (65536 - [\text{RL\_TH4}, \text{RL\_TL4}])$  ;

$$\text{So, Baud-Rate of UART4} = \text{SYSclk} / (65536 - [[\text{RL\_TH4}, \text{RL\_TL4}]] / 4$$

If T4 works in 12T mode (T4T3M.5/T4x12=0), the T4 overflow =  $\text{SYSclk} / 12 / (65536 - [\text{RL\_TH4}, \text{RL\_TL4}])$  ;

$$\text{So, Baud-Rate of UART4} = \text{SYSclk} / 12 / (65536 - [[\text{RL\_TH4}, \text{RL\_TL4}]] / 4$$

RL\_TH4 is the reloaded register of T4H, and RL\_TL4 is the reload register of T4L in above formula.

*\* When S4\_S bit in P\_SW2 register is set, the function of UART4 is redirected to P5.2 for RXD4 and P5.3 for TXD4.*

---

## Chapter 9 IAP/EEPROM Function of STC15 Series

STC15W4K32S4 series MCU has integrated a large capacity of internal EEPROM which is separated from program space. Internal EEPROM, which could be repeatedly erased more than 100 thousand times, can be used as Data Flash by ISP/IAP technology.

The In-System Programmable (ISP) in STC15 series makes it possible to update the user's application program and non-volatile application data (in IAP-memory) without removing the MCU chip from the actual end product. This useful capability makes a wide range of field-update applications possible. (Note ISP needs the loader program pre-programmed in the ISP-memory.) In general, the user needn't know how ISP operates because STC has provided the standard ISP tool and embedded ISP code in STC shipped samples. But, to develop a good program for ISP function, the user has to understand the architecture of the embedded flash.

The embedded EEPROM consists of several pages. Each page contains 512 bytes. Dealing with flash, the user must erase it in page unit before writing (programming) data into it. Erasing flash means setting the content of that flash as FFh. Two erase modes are available in this chip. One is mass mode and the other is page mode. The mass mode gets more performance, but it erases the entire flash. The page mode is something performance less, but it is flexible since it erases flash in page unit. Unlike RAM's real-time operation, to erase flash or to write (program) flash often takes long time so to wait finish.

Furthermore, it is a quite complex timing procedure to erase/program flash. Fortunately, the STC15Fseries carried with convenient mechanism to help the user read/change the flash content. Just filling the target address and data into several SFR, and triggering the built-in ISP automation, the user can easily erase, read, and program the embedded flash.

The In-Application Program feature is designed for user to Read/Write nonvolatile data flash. It may bring great help to store parameters those should be independent of power-up and power-done action. In other words, the user can store data in data flash memory, and after he shutting down the MCU and rebooting the MCU, he can get the original value, which he had stored in.

The user can program the data flash according to the same way as ISP program, so he should get deeper understanding related to SFR IAP\_DATA, IAP\_ADDRL, IAP\_ADDRH, IAP\_CMD, IAP\_TRIG, and IAP\_CONTR.

## 9.1 IAP / EEPROM Special Function Registers

The following special function registers are related to the IAP/ISP/EEPROM operation. All these registers can be accessed by software in the user's application program.

Symbol	Description	Address	Bit Address and Symbol								Value after Power-on or Reset
			MSB				LSB				
IAP_DATA	ISP/IAP Flash Data Register	C2H									1111 1111B
IAP_ADDRH	ISP/IAP Flash Address High	C3H									0000 0000B
IAP_ADDRL	ISP/IAP Flash Address Low	C4H									0000 0000B
IAP_CMD	ISP/IAP Flash Command Register	C5H	-	-	-	-	-	-	MS1	MS0	xxxx x000B
IAP_TRIG	ISP/IAP Flash Command Trigger	C6H									xxxx xxxxB
IAP_CONTR	ISP/IAP Control Register	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	WT2	WT1	WT0	0000 x000B
PCON	Power Control	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011 0000B

### 1. ISP/IAP Flash Data Register : IAP\_DATA (Address: C2H, Non bit-addressable)

IAP\_DATA is the data port register for ISP/IAP operation. The data in IAP\_DATA will be written into the desired address in operating ISP/IAP write and it is the data window of readout in operating ISP/IAP read.

### 2. ISP/IAP Flash Address Registers : IAP\_ADDRH and IAP\_ADDRL

IAP\_ADDRH is the high-byte address port for all ISP/IAP modes.

IAP\_ADDRH[7:5] must be cleared to 000, if one bit of IAP\_ADDRH[7:5] is set, the IAP/ISP write function must fail.

IAP\_ADDRL is the low port for all ISP/IAP modes. In page erase operation, it is ignored.

### 3. ISP/IAP Flash Command Register : IAP\_CMD (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CMD	C5H	name	-	-	-	-	-	-	MS1	MS0

B7~B2: Reserved.

MS1, MS0 : ISP/IAP operating mode selection. IAP\_CMD is used to select the flash mode for performing numerous ISP/IAP function or used to access protected SFRs.

0, 0 : Standby

0, 1 : Data Flash/EEPROM read.

1, 0 : Data Flash/EEPROM program.

1, 1 : Data Flash/EEPROM page erase.

Except IAP15 series MCU, STC15 series only can data flash/EEPROM byte-read / byte-program / page erase. The user program can directly modify the user program area in the user program area for IAP15 series.

**Special Statement :** EEPROM also can be read by instruction MOVC (which is used to read program memory), but whose start address is the next of end address in program memory instead of 0000H.

**4. ISP/IAP Flash Command Trigger Register : IAP\_TRIG** (Address: C6H, Non bit -addressable)

IAP\_TRIG is the command port for triggering ISP/IAP activity and protected SFRs access. If IAP\_TRIG is filled with sequential 0x5Ah, 0xA5h and if IAPEN(IAP\_CONTR.7) = 1, ISP/IAP activity or protected SFRs access will triggered.

**5. ISP/IAP Control Register : IAP\_CONTR** (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	name	IAPEN	SWBS	SWRST	CMD_FAIL	-	WT2	WT2	WT0

IAPEN : ISP/IAP operation enable.

- 0 : Global disable all ISP/IAP program/erase/read function.
- 1 : Enable ISP/IAP program/erase/read function.

SWBS: software boot selection control bit

- 0 : Boot from main-memory after reset.
- 1 : Boot from ISP memory after reset.

SWRST: software reset trigger control.

- 0 : No operation
- 1 : Generate software system reset. It will be cleared by hardware automatically.

CMD\_FAIL: Command Fail indication for ISP/IAP operation.

- 0 : The last ISP/IAP command has finished successfully.
- 1 : The last ISP/IAP command fails. It could be caused since the access of flash memory was inhibited.

B3: Reserved. Software must write “0” on this bit when IAP\_CONTR is written.

```

;Software reset from user application program area (AP area) and switch to AP area to run program
MOV IAP_CONTR, #00100000B ;SWBS = 0(Select AP area), SWRST = 1(Software reset)
;Software reset from system ISP monitor program area (ISP area) and switch to AP area to run program
MOV IAP_CONTR, #00100000B ;SWBS = 0(Select AP area), SWRST = 1(Software reset)
;Software reset from user application program area (AP area) and switch to ISP area to run program
MOV IAP_CONTR, #01100000B ;SWBS = 1(Select ISP area), SWRST = 1(Software reset)
;Software reset from system ISP monitor program area (ISP area) and switch to ISP area to run program
MOV IAP_CONTR, #01100000B ;SWBS = 1(Select ISP area), SWRST = 1(Software reset)

```

This reset is to reset the whole system, all special function registers and I/O ports will be reset to the initial value



WT2~WT0 : Waiting time selection while flash is busy.

Setting wait times			CPU wait times			
WT2	WT1	WT0	Read (2 System clocks)	Program (=55uS)	Sector Erase (=21mS)	Recommended System Clock Frequency (MHz)
1	1	1	2 SYSclks	55 SYSclks	21012 SYSclks	1MHz
1	1	0	2 SYSclks	110 SYSclks	42024 SYSclks	2MHz
1	0	1	2 SYSclks	165 SYSclks	63036 SYSclks	3MHz
1	0	0	2 SYSclks	330 SYSclks	126072 SYSclks	6MHz
0	1	1	2 SYSclks	660 SYSclks	252144 SYSclks	12MHz
0	1	0	2 SYSclks	1100 SYSclks	420240 SYSclks	20MHz
0	0	1	2 SYSclks	1320 SYSclks	504288 SYSclks	24MHz
0	0	0	2 SYSclks	1760 SYSclks	672384 SYSclks	30MHz

Note: Software reset actions could reset other SFR, but it never influences bits IAPEN and SWBS. The IAPEN and SWBS. The IAPEN and SWBS only will be reset by power-up action, while not software reset.

## 6. When the operation voltage is too low, EEPROM / IAP function should be disabled

PCON register (Power Control Register)

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

LVDF : Pin Low-Voltage Flag. Once low voltage condition is detected (VCC power is lower than LVD voltage), it is set by hardware (and should be cleared by software).

The low-voltage detector parameter of STC15W4K32S4 series MCU shown in following figure is optional :

Enabel Low-Voltage Reset, controls reset or not while the Low-Voltage event  
Choice: Reset while detect a low-voltage  
No-Choice: Interrupt while detect a low-voltage

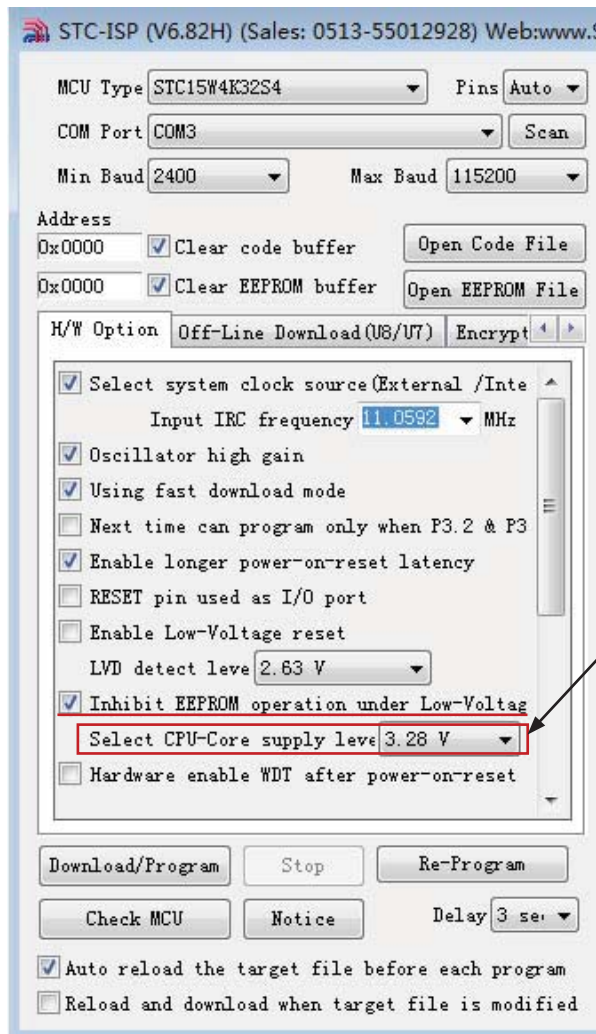
Optional reset threshold voltage of STC15W4K32S4 series MCU

The low-voltage detector parameter adjust the thresh voltage level of the built-in low-voltage detector.

When the oscillator frequency is between 4M ~ 24MHz, low-voltage detection threshold voltage is recommended to choose more than 2.62V.

When the oscillator frequency is between 25M ~ 35MHz, low-voltage detection threshold voltage is recommended to choose more than 2.79V.

Don't enable EEPROM/IAP function when the operation voltage is too low. Namely, enable the option "Inhibit EEPROM operation under Low-Voltage" in STC-ISP Writer/Programmer.



Select CPU-Core supply level:

1M~24M, recommend set to about 2.66V  
24M~28M, recommend set to about 3.32V  
28M~40M, recommend set to about 3.63V

## 9.2 STC15W4K32S4 Series Internal EEPROM Allocation Table

STC15 series microcontroller's Data Flash (internal available EEPROM) address (and program space is separate) : if the application area of IAP write Data/erase sector of the action, the statements will be ignore and continue to the next one. Program in user application area (AP area), only operate IAP/ISP on Data Flash (EEPROM )

STC15W4K32S4 series MCU internal EEPROM Selection Table							For STC15W4K32S4 series MCU, EEPROM also can be read by instruction MOVC (which is used to read program memory), but whose start address is the next of end address in program memory instead of 0000H.  Each sector 512 byte
Type	EEPROM (Byte)	Sector Numbers	If read by IAP byte, EPROM Begin_Sector Begin_Address	If read by IAP byte, EPROM End_Sector End_Address	If read by MOVC instruction, EPROM Begin_Sector Begin_Address	If read by MOVC instruction, EPROM End_Sector End_Address	
STC15W4K16S4	42K	84	0000h	A7FFh	4C00h	F3FFh	
STC15W4K32S4	26K	52	0000h	67FFh	8C00h	F3FFh	
STC15W4K40S4	18K	36	0000h	47FFh	AC00h	F3FFh	
STC15W4K48S4	10K	20	0000h	27FFh	CC00h	F3FFh	
STC15W4K56S4	2K	4	0000h	07FFh	EC00h	F3FFh	
The following series are special. User can directly modify the application program in the application area, all flash area could be used as EEPROM							
IAP15W4K58S4	-	116	0000h	E7FFh			No particular EEPROM, But the user program can directly modify the user program area in the user program area.
IAP15W4K61S4	-	122	0000h	F3FFh			No particular EEPROM, But the user program can directly modify the user program area in the user program area.
IRC15W4K63S4	-	127	0000h	FDFHh			No particular EEPROM, But the user program can directly modify the user program area in the user program area.

STC15 series MCU address reference table in detail (512 bytes per sector)							
Sector 1		Sector 2		Sector 3		Sector 4	
Start	End	Start	End	Start	End	Start	End
0000H	01FFH	0200H	03FFH	0400H	05FFH	0600H	07FFH
Sector 5		Sector 6		Sector 7		Sector 8	
Start	End	Start	End	Start	End	Start	End
0800H	09FFH	0A00H	0BFFH	0C00H	0DFFH	0E00H	0FFFH
Sector 9		Sector 10		Sector 11		Sector 12	
Start	End	Start	End	Start	End	Start	End
1000H	11FFH	1200H	13FFH	1400H	15FFH	1600H	17FFH
Sector 13		Sector 14		Sector 15		Sector 16	
Start	End	Start	End	Start	End	Start	End
1800H	19FFH	1A00H	1BFFH	1C00H	1DFFH	1E00H	1FFFH
Sector 17		Sector 18		Sector 19		Sector 20	
Start	End	Start	End	Start	End	Start	End
2000H	21FFH	2200H	23FFH	2400H	25FFH	2600H	27FFH
Sector 21		Sector 22		Sector 23		Sector 24	
Start	End	Start	End	Start	End	Start	End
2800H	29FFH	2A00H	2BFFH	2C00H	2DFFH	2E00H	2FFFH
Sector 25		Sector 26		Sector 27		Sector 28	
Start	End	Start	End	Start	End	Start	End
3000H	31FFH	3200H	33FFH	3400H	35FFH	3600H	37FFH
Sector 29		Sector 30		Sector 31		Sector 32	
Start	End	Start	End	Start	End	Start	End
3800H	39FFH	3A00H	3BFFH	3C00H	3DFFH	3E00H	3FFFH
Sector 33		Sector 34		Sector 35		Sector 36	
Start	End	Start	End	Start	End	Start	End
4000H	41FFH	4200H	43FFH	4400H	45FFH	4600H	47FFH
Sector 37		Sector 38		Sector 39		Sector 40	
Start	End	Start	End	Start	End	Start	End
4800H	49FFH	4A00H	4BFFH	4C00H	4DFFH	4E00H	4FFFH
Sector 41		Sector 42		Sector 43		Sector 44	
Start	End	Start	End	Start	End	Start	End
5000H	51FFH	5200H	53FFH	5400H	55FFH	5600H	57FFH
Sector 45		Sector 46		Sector 47		Sector 48	
Start	End	Start	End	Start	End	Start	End
5800H	59FFH	5A00H	5BFFH	5C00H	5DFFH	5E00H	5FFFH
Sector 49		Sector 50		Sector 51		Sector 52	
Start	End	Start	End	Start	End	Start	End
6000H	61FFH	6200H	63FFH	6400H	65FFH	6600H	67FFH
Sector 53		Sector 54		Sector 55		Sector 56	
Start	End	Start	End	Start	End	Start	End
6800H	69FFH	6A00H	6BFFH	6C00H	6DFFH	6E00H	6FFFH
Sector 57		Sector 58		Sector 59		Sector 60	
Start	End	Start	End	Start	End	Start	End
7000H	71FFH	7200H	73FFH	7400H	75FFH	7600H	77FFH
Sector 61		Sector 62		Sector 63		Sector 64	
Start	End	Start	End	Start	End	Start	End
7800h	79FFh	7A00h	7BFFh	7C00h	7DFFh	7E00h	7FFFh

Each sector 512 byte

Suggest the same times modified data in the same sector, each times modified data in different sectors, don't have to use full, of course, it was all to use

STC15 series MCU address reference table in detail (512 bytes per sector)							
Sector 65		Sector 66		Sector 67		Sector 68	
Start	End	Start	End	Start	End	Start	End
8000H	81FFH	8200H	83FFH	8400H	85FFH	8600H	87FFH
Sector 69		Sector 70		Sector 71		Sector 72	
Start	End	Start	End	Start	End	Start	End
8800H	89FFH	8A00H	8BFFH	8C00H	8DFFH	8E00H	8FFFH
Sector 73		Sector 74		Sector 75		Sector 76	
Start	End	Start	End	Start	End	Start	End
9000H	91FFH	9200H	93FFH	9400H	95FFH	9600H	97FFH
Sector 77		Sector 78		Sector 79		Sector 80	
Start	End	Start	End	Start	End	Start	End
9800H	99FFH	9A00H	9BFFH	9C000H	9DFFH	9E00H	9FFFH
Sector 81		Sector 82		Sector 83		Sector 84	
Start	End	Start	End	Start	End	Start	End
A000H	A1FFH	A200H	A3FFH	A400H	A5FFH	A600H	A7FFH
Sector 85		Sector 86		Sector 87		Sector 88	
Start	End	Start	End	Start	End	Start	End
A800H	A9FFH	AA00H	ABFFH	AC00H	ADFFH	AE00H	AFFFH
Sector 89		Sector 90		Sector 91		Sector 92	
Start	End	Start	End	Start	End	Start	End
B000H	B1FFH	B200H	B3FFH	B400H	B5FFH	B600H	B7FFH
Sector 93		Sector 94		Sector 95		Sector 96	
Start	End	Start	End	Start	End	Start	End
B800H	B9FFH	BA00H	BBFFH	BC000H	BDFFH	BE00H	BFFFH
Sector 97		Sector 98		Sector 99		Sector 100	
Start	End	Start	End	Start	End	Start	End
C000H	C1FFH	C200H	C3FFH	C400H	C5FFH	C600H	C7FFH
Sector 101		Sector 102		Sector 103		Sector 104	
Start	End	Start	End	Start	End	Start	End
C800H	C9FFH	CA00H	CBFFH	CC00H	CDFFH	CE00H	CFFFH
Sector 105		Sector 106		Sector 107		Sector 108	
Start	End	Start	End	Start	End	Start	End
D000H	D1FFH	D200H	D3FFH	D400H	D5FFH	D600H	D7FFH
Sector 109		Sector 110		Sector 111		Sector 112	
Start	End	Start	End	Start	End	Start	End
D800H	D9FFH	DA00H	DBFFH	DC00H	DDFFH	DE00H	DFFFH
Sector 113		Sector 114		Sector 115		Sector 116	
Start	End	Start	End	Start	End	Start	End
E000H	E1FFH	E200H	E3FFH	E400H	E5FFH	E600H	E7FFH
Sector 117		Sector 118		Sector 119		Sector 120	
Start	End	Start	End	Start	End	Start	End
E800H	E9FFH	EA00H	EBFFH	EC00H	EDFFH	EE00H	EFFFH
Sector 121		Sector 122		Sector 123		Sector 124	
Start	End	Start	End	Start	End	Start	End
F000H	F1FFH	F200H	F3FFH	F400H	F5FFH	F600H	F7FFH
Sector 125		Sector 126		Sector 127			
Start	End	Start	End	Start	End		
F800H	F9FFH	FA00H	FBFFH	FC00H	FDFFH		

Each sector 512 byte

Suggest the same times modified data in the same sector, each times modified data in different sectors, don't have to use full, of course, it was all to use

---

## 9.3 IAP/EEPROM Assembly Program Introduction

;/\*It is decided by the assembler/compiler used by users that whether the SFRs addresses are declared by the DATA or the EQU directive\*/

IAP_DATA	DATA	0C2H	or	IAP_DATA	EQU	0C2H
IAP_ADDRH	DATA	0C3H	or	IAP_ADDRH	EQU	0C3H
IAP_ADDRL	DATA	0C4H	or	IAP_ADDRL	EQU	0C4H
IAP_CMD	DATA	0C5H	or	IAP_CMD	EQU	0C5H
IAP_TRIG	DATA	0C6H	or	IAP_TRIG	EQU	0C6H
IAP_CONTR	DATA	0C7H	or	IAP_CONTR	EQU	0C7H

;/\*Define ISP/IAP/EEPROM command and wait time\*/

ISP_IAP_BYTE_READ	EQU	1	;Byte-Read
ISP_IAP_BYTE_PROGRAM	EQU	2	;Byte-Program
ISP_IAP_SECTOR_ERASE	EQU	3	;Sector-Erase
WAIT_TIME	EQU	0	;Set wait time

;/\*Byte-Read\*/

MOV	IAP_ADDRH,	#BYTE_ADDR_HIGH	;Set ISP/IAP/EEPROM address high
MOV	IAP_ADDRL,	#BYTE_ADDR_LOW	;Set ISP/IAP/EEPROM address low
MOV	IAP_CONTR,	#WAIT_TIME	;Set wait time
ORL	IAP_CONTR,	#1000000B	;Open ISP/IAP function
MOV	IAP_CMD,	#ISP_IAP_BYTE_READ	;Set ISP/IAP Byte-Read command
MOV	IAP_TRIG,	#5AH	;Send trigger command1 (0x5a)
MOV	IAP_TRIG,	#0A5H	;Send trigger command2 (0xa5)
NOP			;CPU will hold here until ISP/IAP/EEPROM operation complete
MOV	A,	IAP_DATA	;Read ISP/IAP/EEPROM data

;/\*Disable ISP/IAP/EEPROM function, make MCU in a safe state\*/

MOV	IAP_CONTR,	#0000000B	;Close ISP/IAP/EEPROM function
MOV	IAP_CMD,	#0000000B	;Clear ISP/IAP/EEPROM command
;MOV	IAP_TRIG,	#0000000B	;Clear trigger register to prevent mistrigger
;MOV	IAP_ADDRH,	#0FFH	;Move FFH into address high-byte unit, ;Data ptr point to non-EEPROM area
;MOV	IAP_ADDRL,	#0FFH	;Move FFH into address low-byte unit, ;prevent misuse

;/\*Byte-Program, if the byte is null(0FFH), it can be programmed; else, MCU must operate Sector-Erase firstly, and then can operate Byte-Program.\*/

MOV	IAP_DATA,	#ONE_DATA	;Write ISP/IAP/EEPROM data
MOV	IAP_ADDRH,	#BYTE_ADDR_HIGH	;Set ISP/IAP/EEPROM address high
MOV	IAP_ADDRL,	#BYTE_ADDR_LOW	;Set ISP/IAP/EEPROM address low
MOV	IAP_CONTR,	#WAIT_TIME	;Set wait time

---

```

ORL    IAP_CONTR,    #10000000B           ;Open ISP/IAP function
MOV    IAP_CMD,     #ISP_IAP_BYTE_READ  ;Set ISP/IAP Byte-Read command
MOV    IAP_TRIG,    #5AH                 ;Send trigger command1 (0x5a)
MOV    IAP_TRIG,    #0A5H                ;Send trigger command2 (0xa5)
NOP                                         ;CPU will hold here until ISP/IAP/EEPROM operation complete

; /*Disable ISP/IAP/EEPROM function, make MCU in a safe state*/
MOV    IAP_CONTR,    #00000000B         ;Close ISP/IAP/EEPROM function
MOV    IAP_CMD,     #00000000B         ;Clear ISP/IAP/EEPROM command
;MOV   IAP_TRIG,    #00000000B         ;Clear trigger register to prevent mistrigger
;MOV   IAP_ADDRH,    #FFH              ;Move FFH into address high-byte unit,
;                                           ;Data ptr point to non-EEPROM area
;MOV   IAP_ADDRL,    #0FFH            ;Move FFH into address low-byte unit,
;                                           ;prevent misuse

; /*Erase one sector area, there is only Sector-Erase instead of Byte-Erase, every sector area account for 512
bytes*/
MOV    IAP_ADDRH,    #SECTOT_FIRST_BYTE_ADDR_HIGH
;                                           ;Set the sector area starting address high
MOV    IAP_ADDRL,    #SECTOT_FIRST_BYTE_ADDR_LOW
;                                           ;Set the sector area starting address low
MOV    IAP_CONTR,    #WAIT_TIME         ;Set wait time
ORL    IAP_CONTR,    #10000000B         ;Open ISP/IAP function
MOV    IAP_CMD,     #ISP_IAP_SECTOR_ERASE ;Set Sectot-Erase command
MOV    IAP_TRIG,    #5AH                 ;Send trigger command1 (0x5a)
MOV    IAP_TRIG,    #0A5H                ;Send trigger command2 (0xa5)
NOP                                         ;CPU will hold here until ISP/IAP/EEPROM operation complete

; /*Disable ISP/IAP/EEPROM function, make MCU in a safe state*/
MOV    IAP_CONTR,    #00000000B         ;Close ISP/IAP/EEPROM function
MOV    IAP_CMD,     #00000000B         ;Clear ISP/IAP/EEPROM command
;MOV   IAP_TRIG,    #00000000B         ;Clear trigger register to prevent mistrigger
;MOV   IAP_ADDRH,    #0FFH            ;Move FFH into address high-byte unit,
;                                           ;Data ptr point to non-EEPROM area
;MOV   IAP_ADDRL,    #0FFH            ;Move FFH into address low-byte unit,
;                                           ;prevent misuse

```

---

---

**Little common sense:** (STC MCU Data Flash use as EEPROM function)

Three basic commands -- bytes read, byte programming, the sector erased

Byte programming: "1" write "1" or "0", will "0" write "0". Just FFH can byte programming. If the byte not FFH, you must erase the sector, because only the "sectors erased" to put "0" into "1".

Sector erased: only "sector erased" will also be a "0" erased for "1".

### Big proposal:

1. The same times modified data in the same sector, not the same times modified data in other sectors, won't have to read protection.
2. If a sector with only one byte, that's real EEPROM, STC MCU Data Flash faster than external EEPROM, read a byte/many one byte programming is about 2 clock / 55uS.
3. If in a sector of storing a large amounts of data, a only need to modify one part of a byte, or when the other byte don't need to modify data must first read on STC MCU, then erased RAM the whole sector, again will need to keep data and need to amend data in bytes written back to this sector section literally only bytes written orders (without continuous bytes, write command). Then each sector use bytes are using the less the convenient (not need read a lot of maintained data).

### Frequently asked questions:

1. IAP instructions after finishing, address is automatically "add 1" or "minus 1"?

Answer: not

2. Send 5A and A5 after IAP ordered the trigger whether to have sent 5A and A5 trigger?

Answer: yes



---

## 9.4 EEPROM Demo Program (C and ASM)

### 9.4.1 EEPROM Demo Program (not Transmit data by UART)

#### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program EEPROM/IAP -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

void IapIdle();
BYTE IapReadByte(WORD addr);

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

typedef unsigned char BYTE;
typedef unsigned int WORD;

//-----

sfr    IAP_DATA      = 0xC2;      //IAP data register
sfr    IAP_ADDRH     = 0xC3;      //IAP address HIGH
sfr    IAP_ADDRL     = 0xC4;      //IAP address LOW
sfr    IAP_CMD       = 0xC5;      //IAP command register
sfr    IAP_TRIG      = 0xC6;      //IAP command trigger register
sfr    IAP_CONTR     = 0xC7;      //IAP control register

#define  CMD_IDLE     0           //Stand-By
#define  CMD_READ     1           //IAP Byte-Read
#define  CMD_PROGRAM  2           //IAP Byte-Program
#define  CMD_ERASE    3           //IAP Sector-Erase

##define ENABLE_IAP  0x80        //if SYSCLK<30MHz
##define ENABLE_IAP  0x81        //if SYSCLK<24MHz
#define  ENABLE_IAP  0x82        //if SYSCLK<20MHz
##define ENABLE_IAP  0x83        //if SYSCLK<12MHz
##define ENABLE_IAP  0x84        //if SYSCLK<6MHz
```

---

```

#define ENABLE_IAP    0x85           //if SYSCLK<3MHz
#define ENABLE_IAP    0x86           //if SYSCLK<2MHz
#define ENABLE_IAP    0x87           //if SYSCLK<1MHz

//Start address for STC15 series MCU EEPROM
#define IAP_ADDRESS   0x0400

void Delay(BYTE n);
void IapIdle();
BYTE IapReadByte(WORD addr);
void IapProgramByte(WORD addr, BYTE dat);
void IapEraseSector(WORD addr);

void main()
{
    WORD i;

    P1 = 0xfe;           //1111,1110 System Reset OK
    Delay(10);          //Delay
    IapEraseSector(IAP_ADDRESS); //Erase current sector
    for (i=0; i<512; i++) //Check whether all sector data is FF
    {
        if (IapReadByte(IAP_ADDRESS+i) != 0xff)
            goto Error; //If error, break
    }
    P1 = 0xfc;           //1111,1100 Erase successful
    Delay(10);          //Delay
    for (i=0; i<512; i++) //Program 512 bytes data into data flash
    {
        IapProgramByte(IAP_ADDRESS+i, (BYTE)i);
    }
    P1 = 0xf8;           //1111,1000 Program successful
    Delay(10);          //Delay
    for (i=0; i<512; i++) //Verify 512 bytes data
    {
        if (IapReadByte(IAP_ADDRESS+i) != (BYTE)i)
            goto Error; //If error, break
    }
    P1 = 0xf0;           //1111,0000 Verify successful
    while (1);
Error:
    P1 &= 0x7f;         //0xxx,xxxx IAP operation fail
    while (1);
}

```

---

---

```

/*-----
Software delay function
-----*/
void Delay(BYTE n)
{
    WORD x;

    while (n--)
    {
        x = 0;
        while (++x);
    }
}

/*-----
Disable ISP/IAP/EEPROM function
Make MCU in a safe state
-----*/
void IapIdle()
{
    IAP_CONTR    = 0;           //Close IAP function
    IAP_CMD      = 0;           //Clear command to standby
    IAP_TRIG     = 0;           //Clear trigger register
    IAP_ADDRH    = 0x80;       //Data ptr point to non-EEPROM area
    IAP_ADDRL    = 0;           //Clear IAP address to prevent misuse
}

/*-----
Read one byte from ISP/IAP/EEPROM area
Input: addr (ISP/IAP/EEPROM address)
Output:Flash data
-----*/
BYTE IapReadByte(WORD addr)
{
    BYTE dat;                  //Data buffer

    IAP_CONTR = ENABLE_IAP;    //Open IAP function, and set wait time
    IAP_CMD = CMD_READ;        //Set ISP/IAP/EEPROM READ command
    IAP_ADDRL = addr;          //Set ISP/IAP/EEPROM address low
    IAP_ADDRH = addr >> 8;    //Set ISP/IAP/EEPROM address high
    IAP_TRIG = 0x5a;           //Send trigger command1 (0x5a)
    IAP_TRIG = 0xa5;           //Send trigger command2 (0xa5)
    _nop_();                   //MCU will hold here until ISP/IAP/EEPROM
                                //operation complete
    dat = IAP_DATA;            //Read ISP/IAP/EEPROM data
    IapIdle();                 //Close ISP/IAP/EEPROM function

    return dat;                //Return Flash data
}

```

---

---

```
/*-----*/
```

Program one byte to ISP/IAP/EEPROM area

Input: addr (ISP/IAP/EEPROM address)

dat (ISP/IAP/EEPROM data)

Output:-

```
-----*/
```

```
void IapProgramByte(WORD addr, BYTE dat)
```

```
{
    IAP_CONTR = ENABLE_IAP;           //Open IAP function, and set wait time
    IAP_CMD = CMD_PROGRAM;           //Set ISP/IAP/EEPROM PROGRAM command
    IAP_ADDRH = addr;                 //Set ISP/IAP/EEPROM address low
    IAP_ADDRH = addr >> 8;           //Set ISP/IAP/EEPROM address high
    IAP_DATA = dat;                   //Write ISP/IAP/EEPROM data
    IAP_TRIG = 0x5a;                  //Send trigger command1 (0x5a)
    IAP_TRIG = 0xa5;                  //Send trigger command2 (0xa5)
    _nop_();                           //MCU will hold here until ISP/IAP/EEPROM
                                        //operation complete

    IapIdle();
}
```

```
/*-----*/
```

Erase one sector area

Input: addr (ISP/IAP/EEPROM address)

Output:-

```
-----*/
```

```
void IapEraseSector(WORD addr)
```

```
{
    IAP_CONTR = ENABLE_IAP;           //Open IAP function, and set wait time
    IAP_CMD = CMD_ERASE;              //Set ISP/IAP/EEPROM ERASE command
    IAP_ADDRH = addr;                 //Set ISP/IAP/EEPROM address low
    IAP_ADDRH = addr >> 8;           //Set ISP/IAP/EEPROM address high
    IAP_TRIG = 0x5a;                  //Send trigger command1 (0x5a)
    IAP_TRIG = 0xa5;                  //Send trigger command2 (0xa5)
    _nop_();                           //MCU will hold here until ISP/IAP/EEPROM
                                        //operation complete

    IapIdle();
}
```

---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program EEPROM/IAP -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

;*/Declare SFRs associated with the IAP */
IAP_DATA EQU 0C2H ;Flash data register
IAP_ADDRH EQU 0C3H ;Flash address HIGH
IAP_ADDRL EQU 0C4H ;Flash address LOW
IAP_CMD EQU 0C5H ;Flash command register
IAP_TRIG EQU 0C6H ;Flash command trigger
IAP_CONTR EQU 0C7H ;Flash control register

;*/Define ISP/IAP/EEPROM command*/
CMD_IDLE EQU 0 ;Stand-By
CMD_READ EQU 1 ;Byte-Read
CMD_PROGRAM EQU 2 ;Byte-Program
CMD_ERASE EQU 3 ;Sector-Erase

;ENABLE_IAP EQU 80H //if SYSCLK<30MHz
;ENABLE_IAP EQU 81H //if SYSCLK<24MHz
ENABLE_IAP EQU 82H //if SYSCLK<20MHz
;ENABLE_IAP EQU 83H //if SYSCLK<12MHz
;ENABLE_IAP EQU 84H //if SYSCLK<6MHz
;ENABLE_IAP EQU 85H //if SYSCLK<3MHz
;ENABLE_IAP EQU 86H //if SYSCLK<2MHz
;ENABLE_IAP EQU 87H //if SYSCLK<1MHz

//Start address for STC15 series MCU EEPROM
IAP_ADDRESS EQU 0400H
//-----
ORG 0000H
LJMP MAIN
;-----
ORG 0100H
MAIN:
MOV P1, #0FEH //1111,1110 System Reset OK
LCALL DELAY //Delay
```

---

```

;-----
MOV      DPTR, #IAP_ADDRESS      ;Set ISP/IAP/EEPROM address
LCALL   IAP_ERASE                ;Erase current sector
;-----
MOV      DPTR, #IAP_ADDRESS      ;Set ISP/IAP/EEPROM address
MOV      R0, #0                  ;Set counter (512)
MOV      R1, #2
CHECK1: ;Check whether all sector data is FF
LCALL   IAP_READ                 ;Read Flash
CJNE    A, #0FFH, ERROR         ;If error, break
INC     DPTR                     ;Inc Flash address
DJNZ    R0, CHECK1              ;Check next
DJNZ    R1, CHECK1              ;Check next
;-----
MOV      P1, #0FCH               ;1111,1100 Erase successful
LCALL   DELAY                   ;Delay
;-----
MOV      DPTR, #IAP_ADDRESS      ;Set ISP/IAP/EEPROM address
MOV      R0, #0                  ;Set counter (512)
MOV      R1, #2
MOV      R2, #0                  ;Initial test data
NEXT:   ;Program 512 bytes data into data flash
MOV     A, R2                    ;Ready IAP data
LCALL   IAP_PROGRAM              ;Program flash
INC     DPTR                     ;Inc Flash address
INC     R2                       ;Modify test data
DJNZ    R0, NEXT                 ;Program next
DJNZ    R1, NEXT                 ;Program next
;-----
MOV      P1, #0F8H               ;1111,1000 Program successful
LCALL   DELAY                   ;Delay
;-----
MOV      DPTR, #IAP_ADDRESS      ;Set ISP/IAP/EEPROM address
MOV      R0, #0                  ;Set counter (512)
MOV      R1, #2
MOV      R2, #0
CHECK2: ;Verify 512 bytes data
LCALL   IAP_READ                 ;Read Flash
CJNE    A, 2, ERROR              ;If error, break
INC     DPTR                     ;Inc Flash address
INC     R2                       ;Modify verify data
DJNZ    R0, CHECK2               ;Check next
DJNZ    R1, CHECK2               ;Check next
;-----
MOV      P1, #0F0H               ;1111,0000 Verify successful
SJMP    $

```

---

```

;-----
ERROR:
    MOV    P0,    R0
    MOV    P2,    R1
    MOV    P3,    R2
    CLR    P1.7
    SJMP   $
;0xxx,xxxx IAP operation fail

;-----
;Software delay function
;-----*/
DELAY:
    CLR    A
    MOV    R0,    A
    MOV    R1,    A
    MOV    R2,    #20H

DELAY1:
    DJNZ   R0,    DELAY1
    DJNZ   R1,    DELAY1
    DJNZ   R2,    DELAY1
    RET

;-----
;Disable ISP/IAP/EEPROM function
;Make MCU in a safe state
;-----*/
IAP_IDLE:
    MOV    IAP_CONTR,    #0
    MOV    IAP_CMD,      #0
    MOV    IAP_TRIG,     #0
    MOV    IAP_ADDRH,    #80H
    MOV    IAP_ADDRL,    #0
    RET

;-----
;Read one byte from ISP/IAP/EEPROM area
;Input: DPTR(ISP/IAP/EEPROM address)
;Output:ACC (Flash data)
;-----*/
IAP_READ:
    MOV    IAP_CONTR,    #ENABLE_IAP
    MOV    IAP_CMD,      #CMD_READ
    MOV    IAP_ADDRL,    DPL
    MOV    IAP_ADDRH,    DPH
    MOV    IAP_TRIG,     #5AH
    MOV    IAP_TRIG,     #0A5H
    NOP
    MOV    A,            IAP_DATA
    LCALL  IAP_IDLE
    RET
;MCU will hold here until ISP/IAP/EEPROM operation complete
;Open IAP function, and set wait time
;Set ISP/IAP/EEPROM READ command
;Set ISP/IAP/EEPROM address low
;Set ISP/IAP/EEPROM address high
;Send trigger command1 (0x5a)
;Send trigger command2 (0xa5)
;Read ISP/IAP/EEPROM data
;Close ISP/IAP/EEPROM function

```

---

```

; /*-----
;Program one byte to ISP/IAP/EEPROM area
;Input: DPAT(ISP/IAP/EEPROM address)
;ACC (ISP/IAP/EEPROM data)
;Output:-
;-----*/
IAP_PROGRAM:
    MOV    IAP_CONTR,    #ENABLE_IAP    ;Open IAP function, and set wait time
    MOV    IAP_CMD,      #CMD_PROGRAM   ;Set ISP/IAP/EEPROM PROGRAM command
    MOV    IAP_ADDRLL,   DPL            ;Set ISP/IAP/EEPROM address low
    MOV    IAP_ADDRHH,   DPH            ;Set ISP/IAP/EEPROM address high
    MOV    IAP_DATA,     A              ;Write ISP/IAP/EEPROM data
    MOV    IAP_TRIG,     #5AH           ;Send trigger command1 (0x5a)
    MOV    IAP_TRIG,     #0A5H         ;Send trigger command2 (0xa5)
    NOP                                ;MCU will hold here until ISP/IAP/EEPROM operation complete
    LCALL  IAP_IDLE      ;Close ISP/IAP/EEPROM function
    RET

; /*-----
;Erase one sector area
;Input: DPTR(ISP/IAP/EEPROM address)
;Output:-
;-----*/
IAP_ERASE:
    MOV    IAP_CONTR,    #ENABLE_IAP    ;Open IAP function, and set wait time
    MOV    IAP_CMD,      #CMD_ERASE     ;Set ISP/IAP/EEPROM ERASE command
    MOV    IAP_ADDRLL,   DPL            ;Set ISP/IAP/EEPROM address low
    MOV    IAP_ADDRHH,   DPH            ;Set ISP/IAP/EEPROM address high
    MOV    IAP_TRIG,     #5AH           ;Send trigger command1 (0x5a)
    MOV    IAP_TRIG,     #0A5H         ;Send trigger command2 (0xa5)
    NOP                                ;MCU will hold here until ISP/IAP/EEPROM operation complete
    LCALL  IAP_IDLE      ;Close ISP/IAP/EEPROM function
    RET

    END

```



---

## 9.4.2 EEPROM Demo Program (Transmit data by UART) (C and ASM)

### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program EEPROM/IAP -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

typedef unsigned char BYTE;
typedef unsigned int WORD;

//-----
sfr    IAP_DATA    = 0xC2;    //IAP data register
sfr    IAP_ADDRH   = 0xC3;    //IAP address HIGH
sfr    IAP_ADDRL   = 0xC4;    //IAP address LOW
sfr    IAP_CMD     = 0xC5;    //IAP command register
sfr    IAP_TRIG    = 0xC6;    //IAP command trigger register
sfr    IAP_CONTR   = 0xC7;    //IAP control register

#define  CMD_IDLE   0          //Stand-By
#define  CMD_READ   1          //IAP Byte-Read
#define  CMD_PROGRAM 2        //IAP Byte-Program
#define  CMD_ERASE  3          //IAP Sector-Erase

#define  URMD  0          //0: select T2 as UART1 baud-rate generator
                          //1: select T1 as UART1 baud-rate generator(T1 as 16-bit auto-reload timer/counter)
                          //2: select T1 as UART1 baud-rate generator (T1 as 8-bit auto-reload timer/counter)

sfr    T2H      = 0xD6;
sfr    T2L      = 0xD7;

sfr    AUXR     = 0x8E;    //Auxiliary register
```

---

```

#define ENABLE_IAP    0x80           //if SYSCLK<30MHz
#define ENABLE_IAP    0x81           //if SYSCLK<24MHz
#define ENABLE_IAP    0x82           //if SYSCLK<20MHz
#define ENABLE_IAP    0x83           //if SYSCLK<12MHz
#define ENABLE_IAP    0x84           //if SYSCLK<6MHz
#define ENABLE_IAP    0x85           //if SYSCLK<3MHz
#define ENABLE_IAP    0x86           //if SYSCLK<2MHz
#define ENABLE_IAP    0x87           //if SYSCLK<1MHz

//Start address for STC15 series MCU EEPROM
#define IAP_ADDRESS 0x0400

void Delay(BYTE n);
void IapIdle();
BYTE IapReadByte(WORD addr);
void IapProgramByte(WORD addr, BYTE dat);
void IapEraseSector(WORD addr);
void InitUart();
BYTE SendData(BYTE dat);

void main()
{
    WORD i;

    P1 = 0xfe;                //1111,1110 System Reset OK
    InitUart();               //Initialize UART
    Delay(10);                //Delay
    IapEraseSector(IAP_ADDRESS); //Erase current sector
    for (i=0; i<512; i++)      //Check whether all sector data is FF
    {
        if (SendData(IapReadByte(IAP_ADDRESS+i)) != 0xff)
            goto Error;        //If error, break
    }
    P1 = 0xfc;                //1111,1100 Erase successful
    Delay(10);                //Delay
    for (i=0; i<512; i++)      //Program 512 bytes data into data flash
    {
        IapProgramByte(IAP_ADDRESS+i, (BYTE)i);
    }
    P1 = 0xf8;                //1111,1000 Program successful
    Delay(10);                //Delay
    for (i=0; i<512; i++)      //Verify 512 bytes data
    {
        if (SendData(IapReadByte(IAP_ADDRESS+i)) != (BYTE)i)
            goto Error;        //If error, break
    }
    P1 = 0xf0;                //1111,0000 Verify successful
    while (1);
}

```

---

---

```

Error:
    P1 &= 0x7f;                                //0xxx,xxxx IAP operation fail
    while (1);
}

/*-----
software delay
-----*/
void Delay(BYTE n)
{
    WORD x;

    while (n--)
    {
        x = 0;
        while (++x);
    }
}

/*-----
Disable ISP/IAP/EEPROM function
Make MCU in a safe state
-----*/
void IapIdle()
{
    IAP_CONTR    = 0;                //Close IAP function
    IAP_CMD      = 0;                //Clear command to standby
    IAP_TRIG     = 0;                //Clear trigger register
    IAP_ADDRH    = 0x80;            //Data ptr point to non-EEPROM area
    IAP_ADDRL    = 0;                //Clear IAP address to prevent misuse
}

/*-----
Read one byte from ISP/IAP/EEPROM area
Input: addr (ISP/IAP/EEPROM address)
Output:Flash data
-----*/
BYTE IapReadByte(WORD addr)
{
    BYTE dat;                            //Data buffer

    IAP_CONTR = ENABLE_IAP;           //Open IAP function, and set wait time
    IAP_CMD = CMD_READ;                //Set ISP/IAP/EEPROM READ command
    IAP_ADDRL = addr;                  //Set ISP/IAP/EEPROM address low
    IAP_ADDRH = addr >> 8;            //Set ISP/IAP/EEPROM address high
    IAP_TRIG = 0x5a;                  //Send trigger command1 (0x5a)
    IAP_TRIG = 0xa5;                  //Send trigger command2 (0xa5)
}

```

---

```

    _nop_();                //MCU will hold here until ISP/IAP/EEPROM
                           //operation complete
    dat = IAP_DATA;        //Read ISP/IAP/EEPROM data
    IapIdle();             //Close ISP/IAP/EEPROM function

    return dat;            //Return Flash data
}

```

```

/*-----
Program one byte to ISP/IAP/EEPROM area
Input: addr (ISP/IAP/EEPROM address)
       dat (ISP/IAP/EEPROM data)
Output:-
-----*/

```

```

void IapProgramByte(WORD addr, BYTE dat)
{
    IAP_CONTR = ENABLE_IAP;    //Open IAP function, and set wait time
    IAP_CMD = CMD_PROGRAM;     //Set ISP/IAP/EEPROM PROGRAM command
    IAP_ADDRLL = addr;         //Set ISP/IAP/EEPROM address low
    IAP_ADDRHH = addr >> 8;    //Set ISP/IAP/EEPROM address high
    IAP_DATA = dat;            //Write ISP/IAP/EEPROM data
    IAP_TRIG = 0x5a;           //Send trigger command1 (0x5a)
    IAP_TRIG = 0xa5;           //Send trigger command2 (0xa5)
    _nop_();                   //MCU will hold here until ISP/IAP/EEPROM
                               //operation complete

    IapIdle();
}

```

```

/*-----
Erase one sector area
Input: addr (ISP/IAP/EEPROM address)
Output:-
-----*/

```

```

void IapEraseSector(WORD addr)
{
    IAP_CONTR = ENABLE_IAP;    //Open IAP function, and set wait time
    IAP_CMD = CMD_ERASE;       //Set ISP/IAP/EEPROM ERASE command
    IAP_ADDRLL = addr;         //Set ISP/IAP/EEPROM address low
    IAP_ADDRHH = addr >> 8;    //Set ISP/IAP/EEPROM address high
    IAP_TRIG = 0x5a;           //Send trigger command1 (0x5a)
    IAP_TRIG = 0xa5;           //Send trigger command2 (0xa5)
    _nop_();                   //MCU will hold here until ISP/IAP/EEPROM
                               //operation complete

    IapIdle();
}

```

---

```

/*-----
Initialize UART
-----*/
void InitUart()
{
    SCON = 0x5a;           //set UART1 as 8-bit UART with variable baud-rate
#if    URMD == 0
        T2L = 0xd8;       //Set the preload value
        T2H = 0xff;       //115200 bps(65536-18432000/4/115200)
        AUXR = 0x14;      //T2 in 1T mode, and run T2
        AUXR |= 0x01;     //select T2 as UART1 baud rate generator
#elif  URMD == 1
        AUXR = 0x40;      //T1 in 1T mode
        TMOD = 0x00;      //T1 in mode 0 (16-bit auto-reload timer/counter)
        TL1 = 0xd8;      //Set the preload value
        TH1 = 0xff;      //115200 bps(65536-18432000/4/115200)
        TR1 = 1;         //run T1
#else
        TMOD = 0x20;      //T1 in mode 2 (8-bit auto-reload timer/counter)
        AUXR = 0x40;      //T1 in 1T mode
        TH1 = TL1 = 0xfb; //115200 bps(256 - 18432000/32/115200)
        TR1 = 1;
#endif
}

/*-----
Send data
-----*/
BYTE SendData(BYTE dat)
{
    while (!TI);
    TI = 0;           //Clear TI
    SBUF = dat;

    return dat;
}

```

---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program EEPROM/IAP -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#define URMD 0 //0: select T2 as UART1 baud-rate generator
//1: select T1 as UART1 baud-rate generator(T1 as 16-bit auto-relaod timer/counter)
//2: select T1 as UART1 baud-rate generator (T1 as 8-bit auto-relaod timer/counter)

T2H DATA 0D6H
T2L DATA 0D7H
AUXR DATA 08EH //Auxiliary register

;/*Declare SFRs associated with the IAP */
IAP_DATA EQU 0C2H ;Flash data register
IAP_ADDRH EQU 0C3H ;Flash address HIGH
IAP_ADDRL EQU 0C4H ;Flash address LOW
IAP_CMD EQU 0C5H ;Flash command register
IAP_TRIG EQU 0C6H ;Flash command trigger
IAP_CONTR EQU 0C7H ;Flash control register

;/*Define ISP/IAP/EEPROM command*/
CMD_IDLE EQU 0 ;Stand-By
CMD_READ EQU 1 ;Byte-Read
CMD_PROGRAM EQU 2 ;Byte-Program
CMD_ERASE EQU 3 ;Sector-Erase

;ENABLE_IAP EQU 80H //if SYSCLK<30MHz
;ENABLE_IAP EQU 81H //if SYSCLK<24MHz
ENABLE_IAP EQU 82H //if SYSCLK<20MHz
;ENABLE_IAP EQU 83H //if SYSCLK<12MHz
;ENABLE_IAP EQU 84H //if SYSCLK<6MHz
;ENABLE_IAP EQU 85H //if SYSCLK<3MHz
;ENABLE_IAP EQU 86H //if SYSCLK<2MHz
;ENABLE_IAP EQU 87H //if SYSCLK<1MHz
```

---

//Start address for STC15 series MCU EEPROM

IAP\_ADDRESS EQU 0400H

//-----

ORG 0000H  
LJMP MAIN

-----

ORG 0100H

MAIN:

LCALL INIT\_UART ;Initialize UART  
MOV P1, #0FEH ;1111,1110 System Reset OK  
LCALL DELAY ;Delay

-----

MOV DPTR, #IAP\_ADDRESS ;Set ISP/IAP/EEPROM address  
LCALL IAP\_ERASE ;Sector erase

-----

MOV DPTR, #IAP\_ADDRESS ;Set ISP/IAP/EEPROM address  
MOV R0, #0 ;Set counter (512)  
MOV R1, #2

CHECK1:

LCALL IAP\_READ ;Check whether all sector data is FF  
CJNE A, #0FFH, ERROR ;Read Flash  
INC DPTR ;If error, break  
DJNZ R0, CHECK1 ;Inc Flash address  
DJNZ R1, CHECK1 ;Check next  
DJNZ R1, CHECK1 ;Check next

-----

MOV P1, #0FCH ;1111,1100 Erase successful  
LCALL DELAY ;Delay

-----

MOV DPTR, #IAP\_ADDRESS ;Set ISP/IAP/EEPROM address  
MOV R0, #0 ;Set counter (512)  
MOV R1, #2  
MOV R2, #0 ;Initial test data

NEXT:

MOV A, R2 ;Program 512 bytes data into data flash  
LCALL IAP\_PROGRAM ;Ready IAP data  
INC DPTR ;Program flash  
INC R2 ;Inc Flash address  
DJNZ R0, NEXT ;Modify test data  
DJNZ R1, NEXT ;Program next  
DJNZ R1, NEXT ;Program next

-----

MOV P1, #0F8H ;1111,1000 Program successful  
LCALL DELAY ;Delay

-----

MOV DPTR, #IAP\_ADDRESS ;Set ISP/IAP/EEPROM address  
MOV R0, #0 ;Set counter (512)  
MOV R1, #2  
MOV R2, #0

---

---

```

CHECK2:                                     ;Verify 512 bytes data
        LCALL      IAP_READ                 ;Read Flash
        CJNE       A, 2, ERROR              ;If error, break
        INC        DPTR                    ;Inc Flash address
        INC        R2                      ;Modify verify data
        DJNZ       R0, CHECK2              ;Check next
        DJNZ       R1, CHECK2              ;Check next
;-----
        MOV        P1, #0F0H                ;1111,0000 Verify successful
        SJMP       $
;-----
ERROR:
        MOV        P0, R0
        MOV        P2, R1
        MOV        P3, R2
        CLR        P1.7                    ;0xxx,xxxx IAP operation fail
        SJMP       $
;-----
;*/-----
;Software delay function
;-----*/
DELAY:
        CLR        A
        MOV        R0, A
        MOV        R1, A
        MOV        R2, #20H
DELAY1:
        DJNZ       R0, DELAY1
        DJNZ       R1, DELAY1
        DJNZ       R2, DELAY1
        RET
;*/-----
;Disable ISP/IAP/EEPROM function
;Make MCU in a safe state
;-----*/
IAP_IDLE:
        MOV        IAP_CONTR, #0           ;Close IAP function
        MOV        IAP_CMD, #0            ;Clear command to standby
        MOV        IAP_TRIG, #0          ;Clear trigger register
        MOV        IAP_ADDRH, #80H       ;Data ptr point to non-EEPROM area
        MOV        IAP_ADDRL, #0        ;Clear IAP address to prevent misuse
        RET
;*/-----
;Read one byte from ISP/IAP/EEPROM area
;Input: DPTR(ISP/IAP/EEPROM address)
;Output:ACC (Flash data)
;-----*/

```

---



---

```

IAP_READ:
    MOV    IAP_CONTR,    #ENABLE_IAP    ;Open IAP function, and set wait time
    MOV    IAP_CMD,      #CMD_READ      ;Set ISP/IAP/EEPROM READ command
    MOV    IAP_ADDRL,    DPL            ;Set ISP/IAP/EEPROM address low
    MOV    IAP_ADDRH,    DPH            ;Set ISP/IAP/EEPROM address high
    MOV    IAP_TRIG,     #5AH           ;Send trigger command1 (0x5a)
    MOV    IAP_TRIG,     #0A5H          ;Send trigger command2 (0xa5)
    NOP                                ;MCU will hold here until ISP/IAP/EEPROM operation complete
    MOV    A,            IAP_DATA        ;Read ISP/IAP/EEPROM data
    LCALL  IAP_IDLE      ;Close ISP/IAP/EEPROM function
    RET

```

```

;/*-----
;Program one byte to ISP/IAP/EEPROM area
;Input: DPAT(ISP/IAP/EEPROM address)
;ACC (ISP/IAP/EEPROM data)
;Output:-
;-----*/

```

```

IAP_PROGRAM:
    MOV    IAP_CONTR,    #ENABLE_IAP    ;Open IAP function, and set wait time
    MOV    IAP_CMD,      #CMD_PROGRAM    ;Set ISP/IAP/EEPROM PROGRAM command
    MOV    IAP_ADDRL,    DPL            ;Set ISP/IAP/EEPROM address low
    MOV    IAP_ADDRH,    DPH            ;Set ISP/IAP/EEPROM address high
    MOV    IAP_DATA,     A              ;Write ISP/IAP/EEPROM data
    MOV    IAP_TRIG,     #5AH           ;Send trigger command1 (0x5a)
    MOV    IAP_TRIG,     #0A5H          ;Send trigger command2 (0xa5)
    NOP                                ;MCU will hold here until ISP/IAP/EEPROM operation complete
    LCALL  IAP_IDLE      ;Close ISP/IAP/EEPROM function
    RET

```

```

;/*-----
;Erase one sector area
;Input: DPTR(ISP/IAP/EEPROM address)
;Output:-
;-----*/

```

```

IAP_ERASE:
    MOV    IAP_CONTR,    #ENABLE_IAP    ;Open IAP function, and set wait time
    MOV    IAP_CMD,      #CMD_ERASE     ;Set ISP/IAP/EEPROM ERASE command
    MOV    IAP_ADDRL,    DPL            ;Set ISP/IAP/EEPROM address low
    MOV    IAP_ADDRH,    DPH            ;Set ISP/IAP/EEPROM address high
    MOV    IAP_TRIG,     #5AH           ;Send trigger command1 (0x5a)
    MOV    IAP_TRIG,     #0A5H          ;Send trigger command2 (0xa5)
    NOP                                ;MCU will hold here until ISP/IAP/EEPROM operation complete
    LCALL  IAP_IDLE      ;Close ISP/IAP/EEPROM function
    RET

```

---

```

; /*-----
; Initialize UART
; -----*/
INIT_UART:
    MOV     SCON, #5AH                ;set UART1 as 8-bit UART with variable baud-rate
#if URMD == 0
    MOV     T2L, #0D8H                ;Set the preload value (65536-18432000/4/115200)
    MOV     T2H, #0FFH
    MOV     AUXR, #14H                ;T2 in 1T mode, and run T2
    ORL     AUXR, #01H                ;select T2 as UART1 baud rate generator
#elif URMD == 1
    MOV     AUXR, #40H                ;T1 in 1T mode
    MOV     TMOD, #00H                ;T1 in mode 0 (16-bit auto-reload timer/counter)
    MOV     TL1, #0D8H                ;Set the preload value(65536-18432000/4/115200)
    MOV     TH1, #0FFH
    SETB    TR1                        ;run T1
#else
    MOV     TMOD, #20H                ;T1 in mode 2 (8-bit auto-reload timer/counter)
    MOV     AUXR, #40H                ;T1 in 1T mode
    MOV     TL1, #0FBH                ;115200 bps(256 - 18432000/32/115200)
    MOV     TH1, #0FBH
    SETB    TR1
#endif
    RET

; /*-----
; Send data
; -----*/
SEND_DATA:
    JNB     TI, $
    CLR     TI                        ;Clear TI
    MOV     SBUF, A
    RET

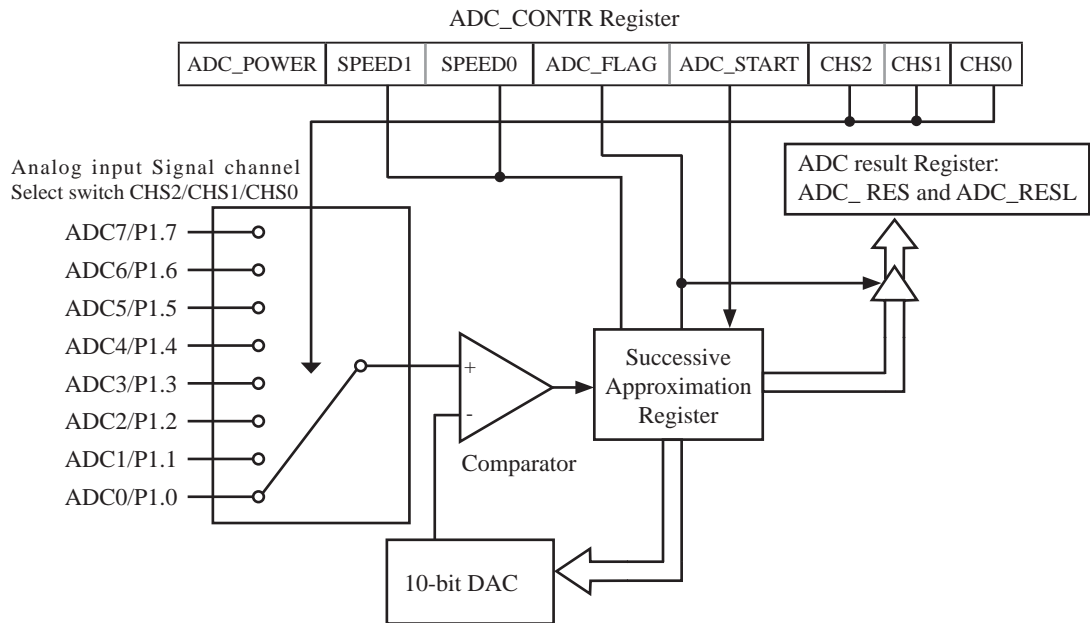
    END

```

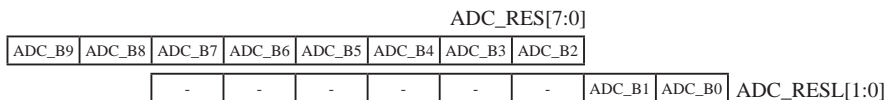
---

# Chapter 10 Analog to Digital Converter

## 10.1 A/D Converter Structure



If  $CLK\_DIV.5(PCON2.5)/ADRJ = 0$ , ADC result Register format is shown as below:



If  $CLK\_DIV.5(PCON2.5)/ADRJ = 1$ , ADC result Register format is shown as below:



STC15 series MCU with A/D conversion function have 8-channel and 10-bit high-speed A/D converters whose speed is up to 300KHz (300 thousand times per second). the 8-channel ADC, which are on P1 port (P1.0-P1.7), can be used as temperature detection, battery voltage detection, key scan, spectrum detection, etc. After power on reset, P1 ports are in weak pull-up mode. Users can set any one of 8 channels as A/D conversion through software. And those I/O ports not as ADC function can continue to be used as I/O ports.

STC15 series MCU ADC (A/D converter) structure is shown above.

---

The ADC on STC15 series is an 10-bit resolution, successive-approximation approach, medium-speed A/D converter.  $V_{REFP}/V_{REFM}$  is the positive/negative reference voltage input for internal voltage-scaling DAC use, the typical sink current on it is 600uA ~ 1mA. For STC15 series, these two references are internally tied to VCC and GND separately.

Conversion is invoked since ADC\_STRAT(ADC\_CONTR.3) bit is set. Before invoking conversion, ADC\_POWER/ADC\_CONTR.7 bit should be set first in order to turn on the power of analog front-end in ADC circuitry. Prior to ADC conversion, the desired I/O ports for analog inputs should be configured as input-only or open-drain mode first. The converter takes around a fourth cycles to sample analog input data and other three fourths cycles in successive-approximation steps. Total conversion time is controlled by two register bits – SPEED1 and SPEED0. Eight analog channels are available on P1 and only one of them is connected to the comparator depending on the selection bits {CHS2,CHS1,CHS0}. When conversion is completed, the result will be saved onto {ADC\_RES,ADC\_RESL[1:0]} register if AUXR1.2(ADRJ) =0 or saved onto {ADC\_RES[1:0],ADC\_RESL} if ADRJ=1 . After the result are completed and saved, ADC\_FLAG is also set. ADC\_FLAG associated with its enable register IE.5(EADC). ADC\_FLAG should be cleared in software. The ADC interrupt service routine vectors to 2Bh . When the chip enters idle mode or power-down mode, the power of ADC is gated off by hardware.

When ADRJ = 0, if user need 10-bit conversion result, calculating the result according to the following formula:

$$\text{10-bit A/D Conversion Result:}(\text{ADC\_RES}[7:0], \text{ADC\_RESL}[1:0]) = 1024 \times \frac{V_{in}}{V_{cc}}$$

When ADRJ = 0, if user need 8-bit conversion result, calculating the result according to the following formula:

$$\text{8-bit A/D Conversion Result:}(\text{ADC\_RES}[7:0]) = 256 \times \frac{V_{in}}{V_{cc}}$$

When ADRJ = 1, if user need 10-bit conversion result, calculating the result according to the following formula:

$$\text{10-bit A/D Conversion Result:}(\text{ADC\_RES}[1:0], \text{ADC\_RESL}[7:0]) = 1024 \times \frac{V_{in}}{V_{cc}}$$

In the above formulas,  $V_{in}$  stand for analog input channel voltage,  $V_{cc}$  stand for actual operation voltage.

## 10.2 Registers for ADC

Mnemonic	Description	Address	bit address and Symbol								Reset value
			MSB				LSB				
P1ASF	P1 Analog Function Configure register	9DH	P17ASF	P16ASF	P15ASF	P14ASF	P13ASF	P12ASF	P11ASF	P10ASF	0000 0000B
ADC_CONTR	ADC Control Register	BCH	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHS0	0000 0000B
ADC_RES	ADC Result high	BDH									0000 0000B
ADC_RESL	ADC Result low	BEH									0000 0000B
CLK_DIV PCON2	Clock Division Register	97H	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000B
IE	Interrupt Enable	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000 0000B
IP	Interrupt Priority Low	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000 0000B

### 1. P1 Analog Function Configure register: P1ASF (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
P1ASF	9DH	name	P17ASF	P16ASF	P15ASF	P14ASF	P13ASF	P12ASF	P11ASF	P10ASF

P1xASF

0 := Keep P1.x as general-purpose I/O function.

1 := Set P1.x as ADC input channel-x

### 2. ADC control register: ADC\_CONTR (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
ADC_CONTR	BCH	name	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHS0

When operating to ADC\_CONTR register, "MOV" should be used, while "AND" and "OR" do not be recommended to use

ADC\_POWER : When clear shut down the power of ADC block. When set turn on the power of ADC block.

SPEED1, SPEED0 : Conversion speed selection.

SPEED1	SPEED0	Times needed by an A/D Conversion
0	0	540 clock cycles are needed for a conversion.
0	1	360 clock cycles are needed for a conversion.
1	0	180 clock cycles are needed for a conversion.
1	1	90 clock cycles are needed for a conversion. When the CPU operation frequency is 27MHz, the speed of ADC is about 300KHz (=27MHz / 90).

The clock source used by ADC block of STC15 series MCU is On-chip R/C clock which is not divided by Clock divider register CLK\_DIV.

ADC\_FLAG : ADC interrupt flag. It will be set by the device after the device has finished a conversion, and should be cleared by the user's software.

---

ADC\_STRAT : ADC start bit, which enable ADC conversion. It will automatically cleared by the device after the device has finished the conversion.

CHS2 ~ CHS0 : Used to select one analog input source from 8 channels.

CHS2	CHS1	CHS0	Source
0	0	0	P1.0 (default) as the A/D channel input
0	0	1	P1.1 as the A/D channel input
0	1	0	P1.2 as the A/D channel input
0	1	1	P1.3 as the A/D channel input
1	0	0	P1.4 as the A/D channel input
1	0	1	P1.5 as the A/D channel input
1	1	0	P1.6 as the A/D channel input
1	1	1	P1.7 as the A/D channel input

*Note : The corresponding bits in PIASF should be configured correctly before starting A/D conversion. The specific PIASF bits should be set corresponding with the desired channels.*

Because it will be delayed 4 CPU clocks after the instruction which set ADC\_CONTR register has been executed, Four "NOP" instructions should be added after setting ADC\_CONTR register. See the following code:

```
MOV    ADC_CONTR, #DATA
NOP
NOP
NOP
NOP
MOV    A,        ADC_CONTR
```

;Only delayed 4 clocks, can the ADC\_CONTR be read correctly.

### 3. ADC Result Arrangement Register Bit ----ADRJ

CLK\_DIV : Clock Division Register (Non bit-addressable)

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
CLK_DIV (PCON2)	97H	Clock Division Register	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 x000B

ADRJ : ADC result adjust bit

- 0 : The 10-bit conversion result of ADC is arranged as {ADC\_RES[7:0], ADC\_RESL[1:0]}.
- 1 : The 10-bit conversion result is right-justified, {ADC\_RES[1:0], ADC\_RESL[7:0]}.

#### 4. ADC result register: ADC\_RES and ADC\_RESL

ADC\_RES and ADC\_RESL are used to save the ADC result, their format as shown below:

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	BDH	ADC result register high								
ADC_RESL	BEH	ADC result register low								
CLK_DIV (PCON2)	97H	Clock Division Register	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0

The ADC\_RES and ADC\_RESL are the final result from the ADC. ADRJ/CLK\_DIV.5 is the control bit of ADC result arrangement in ADC result registers (ADC\_RES, ADC\_RESL).

If ADRJ=0, The higher 8 bits of 10 bits ADC result are arranged in ADC\_RES, and the lower 2 bits are in ADC\_RESL. See the following table.

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	BDH	ADC result register high	ADC_RES9	ADC_RES8	ADC_RES7	ADC_RES6	ADC_RES5	ADC_RES4	ADC_RES3	ADC_RES2
ADC_RESL	BEH	ADC result register low	-	-	-	-	-	-	ADC_RES0	ADC_RES1
CLK_DIV (PCON2)	97H	Clock Division Register			ADRJ=0					

If user need the full 10-bit conversion result, calculating the result according to the following formula:

$$10\text{-bit A/D Conversion Result:}(\text{ADC\_RES}[7:0], \text{ADC\_RESL}[1:0]) = 1024 \times \frac{V_{in}}{V_{cc}}$$

If user only need 8-bit conversion result, calculating the result according to the following formula:

$$8\text{-bit A/D Conversion Result:}(\text{ADC\_RES}[7:0]) = 256 \times \frac{V_{in}}{V_{cc}}$$

In the above formulas,  $V_{in}$  stand for analog input channel voltage,  $V_{cc}$  stand for actual operation voltage.

If ADRJ=1, The higher 2 bits of 10 bits ADC result are arranged in ADC\_RES, and the lower 8 bits are in ADC\_RESL. See the following table.

Mnemonic	Add	Name	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	BDH	ADC result register high							ADC_RES9	ADC_RES8
ADC_RESL	BEH	ADC result register low	ADC_RES7	ADC_RES6	ADC_RES5	ADC_RES4	ADC_RES3	ADC_RES2	ADC_RES0	ADC_RES1
CLK_DIV (PCON2)	97H	Clock Division Register						ADRJ=1		

Calculating the full 10-bit conversion result according to the following formula:

$$10\text{-bit A/D Conversion Result:}(\text{ADC\_RES}[1:0], \text{ADC\_RESL}[7:0]) = 1024 \times \frac{V_{in}}{V_{cc}}$$

In the above formulas,  $V_{in}$  stand for analog input channel voltage,  $V_{cc}$  stand for actual operation voltage.

## 5. Registers bits related with ADC Interrupt : EA, EADC and PADC

IE: Interrupt Enable Register (Bit-addressable)

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

EA : disables all interrupts.

If EA = 0, no interrupt will be acknowledged.

If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

EADC: ADC interrupt enable bit.

If EADC = 0, ADC interrupt would be disabled.

If EADC = 1, ADC interrupt would be enabled.

IP : Interrupt Priority Register (Non bit-addressable)

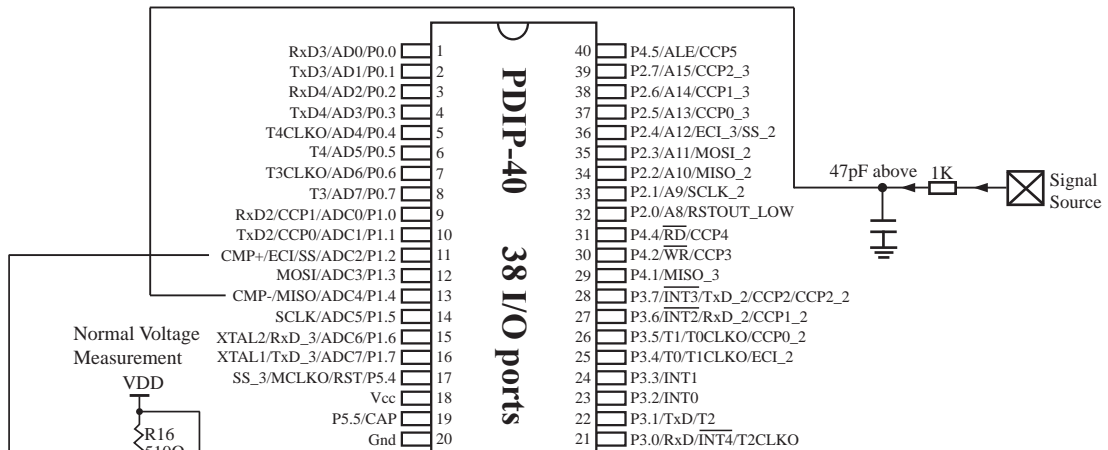
SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	name	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0

PADC : ADC interrupt priority control bit.

if PADC=0, ADC interrupt is assigned lowest priority (priority 0).

if PADC=1, ADC interrupt is assigned highest priority (priority 1).

## 10.3 ADC Typical Application Circuit



A/D Converter on P1 ports, P1.0 - P1.7

8 channels of A/D Converter are on P1.  
P1.x/ADCx means P1.x can be used as  
A/D conversion channel.

Normal Reference Voltage Source TL431B



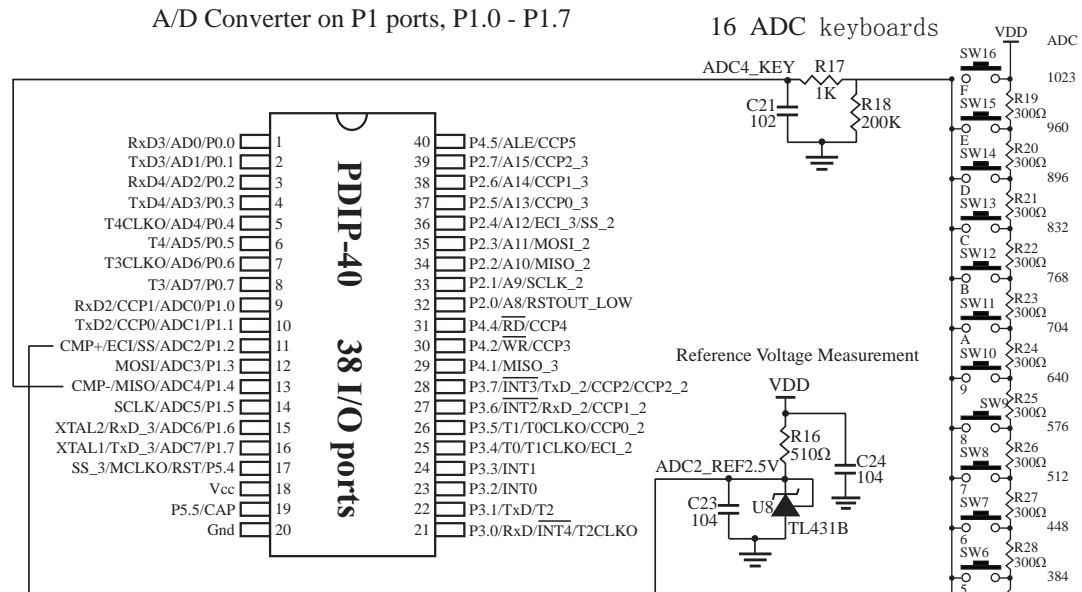
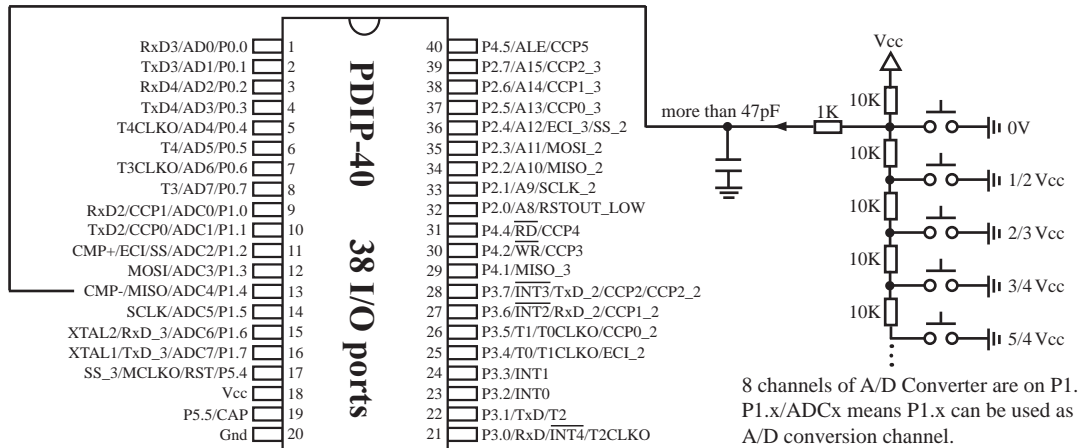
Package: SOT23-3; Price: RMB 0.15~0.3

The symbol of normal reference voltage source TL431B

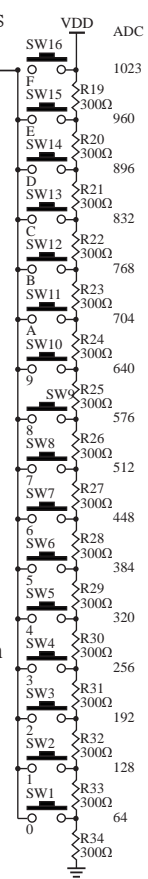




## 10.4 Application Circuit using A/D Conversion to Scan Key

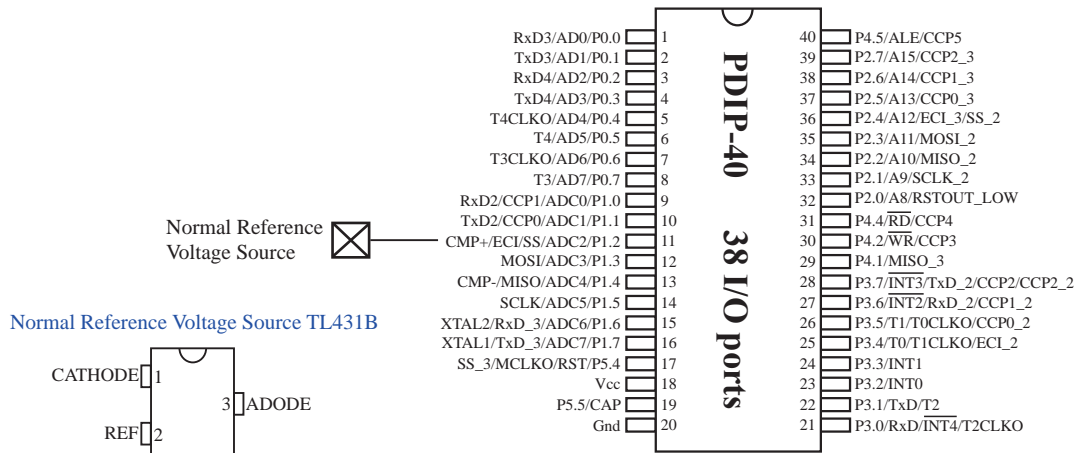


read the value of ADC key every 10ms, and save the last 3 times values. determine the key again if the variation is small. Allow some error when determining key, such as  $\pm 16$  words error

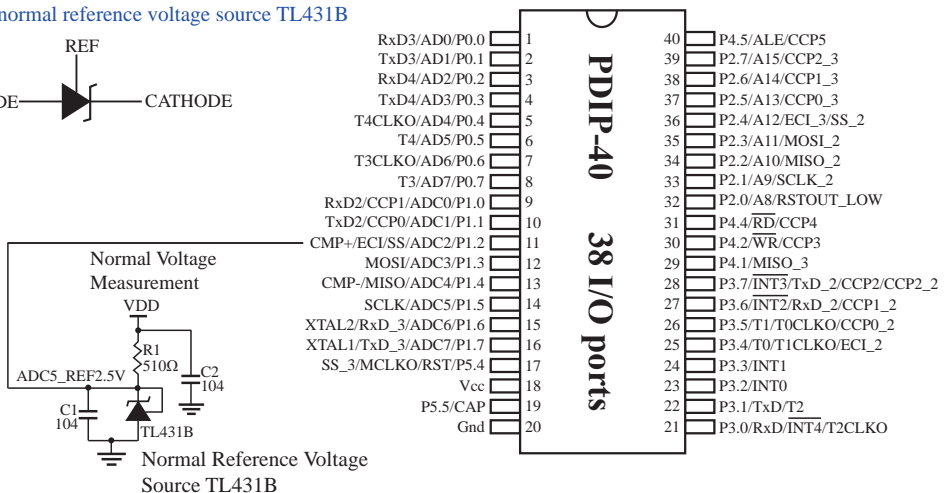
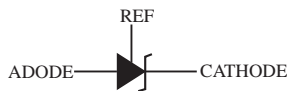


## 10.5 ADC Reference Voltage Source

STC15 series ADC reference voltage is from MCU power supply voltage directly, so it can work without an external reference voltage source. If the required precision is relatively high, then you maybe using a stable reference voltage source, in order to calculate the operating voltage VCC, then calculate the ADC exact value. For example, you can connect a 1.25V(or 1.00V, ect. ...) reference voltage source to ADC channel 2, according to the conversion result, you can get the actual VCC voltage, thus you can calculate other 7 channels ADC results. (Vcc is constant in short time)



The symbol of normal reference voltage source TL431B



---

## 10.6 ADC Demo Program (C and ASM)

### 10.6.1 Demo Program (Demonstrate in ADC Interrupt Mode)

There are two example procedures using interrupts to demonstrate A/D conversion, one written in C language and the other in assembly language.

#### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of ADC -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

#define FOSC 18432000L
#define BAUD 9600

typedef unsigned char BYTE;
typedef unsigned int WORD;

#define URMD 0 //0: select T2 as UART1 baud-rate generator
//1: select T1 as UART1 baud-rate generator(T1 as 16-bit auto-reload timer/counter)
//2: select T1 as UART1 baud-rate generator (T1 as 8-bit auto-reload timer/counter)

sfr T2H = 0xd6;
sfr T2L = 0xd7;

sfr AUXR = 0x8e; //Auxiliary register

/*Declare SFR associated with the ADC */
sfr ADC_CONTR = 0xBC; //ADC control register
sfr ADC_RES = 0xBD; //ADC high 8-bit result register
sfr ADC_LOW2 = 0xBE; //ADC low 2-bit result register
sfr P1ASF = 0x9D; //P1 secondary function control register
```

---

```

/*Define ADC operation const for ADC_CONTR*/
#define ADC_POWER    0x80           //ADC power control bit
#define ADC_FLAG     0x10           //ADC complete flag
#define ADC_START    0x08           //ADC start control bit
#define ADC_SPEEDLL  0x00           //540 clocks
#define ADC_SPEEDL   0x20           //360 clocks
#define ADC_SPEEDH   0x40           //180 clocks
#define ADC_SPEEDHH  0x60           //90 clocks

void    InitUart();
void    SendData(BYTE dat);
void    Delay(WORD n);
void    InitADC();

BYTE    ch = 0;                       //ADC channel NO.

void main()
{
    InitUart();                       //Init UART, use to show ADC result
    InitADC();                         //Init ADC sfr
    IE = 0xa0;                        //Enable ADC interrupt and Open master interrupt switch
                                        //Start A/D conversion

    while (1);
}
/*-----
ADC interrupt service routine
-----*/
void adc_isr() interrupt 5 using 1
{
    ADC_CONTR &= !ADC_FLAG;          //Clear ADC interrupt flag

    SendData(ch);                    //Show Channel NO.
    SendData(ADC_RES);                //Get ADC high 8-bit result and Send to UART

    //if you want show 10-bit result, uncomment next line
    // SendData(ADC_LOW2);            //Show ADC low 2-bit result

    if (++ch > 7) ch = 0;            //switch to next channel
    ADC_CONTR = ADC_POWER | ADC_SPEEDLL | ADC_START | ch;
}
/*-----
Initial ADC sfr
-----*/
void InitADC()
{
    P1ASF = 0xff;                    //Set all P1 as analog input port
    ADC_RES = 0;                     //Clear previous result
    ADC_CONTR = ADC_POWER | ADC_SPEEDLL | ADC_START | ch;
    Delay(2);                        //ADC power-on delay and Start A/D conversion
}

```

---

---

```

/*-----
Initial UART
-----*/
void InitUart()
{
    SCON = 0x5a; //set UART1 as 8-bit UART with variable baud-rate
#ifdef URMD == 0
    T2L = 0xd8; //Set the preload value
    T2H = 0xff; //115200 bps(65536-18432000/4/115200)
    AUXR = 0x14; //T2 in 1T mode, and run T2
    AUXR |= 0x01; //select T2 as UART1 baud rate generator
#elif URMD == 1
    AUXR = 0x40; //T1 in 1T mode
    TMOD = 0x00; //T1 in mode 0 (16-bit auto-reload timer/counter)
    TL1 = 0xd8; //Set the preload value
    TH1 = 0xff; //115200 bps(65536-18432000/4/115200)
    TR1 = 1; //run T1
#else
    TMOD = 0x20; //T1 in mode 2 (8-bit auto-reload timer/counter)
    AUXR = 0x40; //T1 in 1T mode
    TH1 = TL1 = 0xfb; //115200 bps(256 - 18432000/32/115200)
    TR1 = 1;
#endif
}
/*-----
Send one byte data to PC
Input: dat (UART data)
Output:-
-----*/
void SendData(BYTE dat)
{
    while (!TI); //Wait for the previous data is sent
    TI = 0; //Clear TI flag
    SBUF = dat; //Send current data
}
/*-----
Software delay function
-----*/
void Delay(WORD n)
{
    WORD x;

    while (n--)
    {
        x = 5000;
        while (x--);
    }
}

```

---

---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program EEPROM/IAP -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#define URMD 0 //0: select T2 as UART1 baud-rate generator
//1: select T1 as UART1 baud-rate generator(T1 as 16-bit auto-relaod timer/counter)
//2: select T1 as UART1 baud-rate generator (T1 as 8-bit auto-relaod timer/counter)

T2H DATA 0D6H
T2L DATA 0D7H
AUXR DATA 08EH //Auxiliary register

;*/Declare SFR associated with the ADC */
ADC_CONTR EQU 0BCH ;ADC control registe
ADC_RES EQU 0BDH ;ADC high 8-bit result register
ADC_LOW2 EQU 0BEH ;ADC low 2-bit result register
PIASF EQU 09DH ;P1 secondary function control register

;*/Define ADC operation const for ADC_CONTR*/
ADC_POWER EQU 80H ;ADC power control bit
ADC_FLAG EQU 10H ;ADC complete flag
ADC_START EQU 08H ;ADC start control bit
ADC_SPEEDLL EQU 00H ;540 clocks
ADC_SPEEDL EQU 20H ;360 clocks
ADC_SPEEDH EQU 40H ;180 clocks
ADC_SPEEDHH EQU 60H ;90 clocks

ADCCH DATA 20H ;ADC channel NO.

;-----
ORG 0000H
LJMP MAIN

ORG 002BH
LJMP ADC_ISR
;-----
```

---

```

    ORG    0100H
MAIN:
    MOV    SP,    #3FH
    MOV    ADCCH,    #0
    LCALL  INIT_UART           ;Init UART, use to show ADC result
    LCALL  INIT_ADC           ;Init ADC sfr
    MOV    IE,    #0A0H       ;Enable ADC interrupt
                                ;and Open master interrupt switch

    SJMP   $

;-----*/
;ADC interrupt service routine
;-----*/
ADC_ISR:
    PUSH  ACC
    PUSH  PSW

    ANL   ADC_CONTR,    #NOT ADC_FLAG       ;Clear ADC interrupt flag
    MOV   A,            ADCCH
    LCALL SEND_DATA           ;Send channel NO.
    MOV   A,            ADC_  RES          ;Get ADC high 8-bit result
    LCALL SEND_DATA           ;Send to UART

;    MOV   A,            ADC_LOW2          ;Get ADC low 2-bit result
;    LCALL SEND_DATA           ;Send to UART

    INC   ADCCH
    MOV   A,            ADCCH
    ANL   A,            #07H
    MOV   ADCCH, A
    ORL   A,            #ADC_POWER | ADC_SPEEDLL | ADC_START
    MOV   ADC_CONTR,    A                ;ADC power-on delay
                                            ;and re-start A/D conversion

    POP   PSW
    POP   ACC
    RETI

;-----*/
;Initial ADC sfr
;-----*/
INIT_ADC:
    MOV   P1ASF,        #0FFH           ;Set all P1 as analog input port
    MOV   ADC_RES,      #0              ;Clear previous result
    MOV   A,            ADCCH
    ORL   A,            #ADC_POWER | ADC_SPEEDLL | ADC_START
    MOV   ADC_CONTR,    A                ;ADC power-on delay
                                            ;and Start A/D conversion

    MOV   A,            #2
    LCALL DELAY
    RET

```

---

---

```

; /*-----
;Initial UART
;-----*/
INIT_UART:
    MOV     SCON, #5AH           ;set UART1 as 8-bit UART with variable baud-rate
#if      URMD == 0
    MOV     T2L, #0D8H         ;Set the preload value (65536-18432000/4/115200)
    MOV     T2H, #0FFH
    MOV     AUXR, #14H        ;T2 in 1T mode, and run T2
    ORL     AUXR, #01H        ;select T2 as UART1 baud rate generator
#elif URMD == 1
    MOV     AUXR, #40H        ;T1 in 1T mode
    MOV     TMOD, #00H        ;T1 in mode 0 (16-bit auto-reload timer/counter)
    MOV     TL1, #0D8H        ;Set the preload value(65536-18432000/4/115200)
    MOV     TH1, #0FFH
    SETB    TR1               ;run T1
#else
    MOV     TMOD, #20H        ;T1 in mode 2 (8-bit auto-reload timer/counter)
    MOV     AUXR, #40H        ;T1 in 1T mode
    MOV     TL1, #0FBH        ;115200 bps(256 - 18432000/32/115200)
    MOV     TH1, #0FBH
    SETB    TR1
#endif
    RET

; /*-----
;Send one byte data to PC
;Input: ACC (UART data)
;Output:-
;-----*/
SEND_DATA:
    JNB     TI, $              ;Wait for the previous data is sent
    CLR     TI                 ;Clear TI flag
    MOV     SBUF, A            ;Send current data
    RET

; /*-----
;Software delay function
;-----*/
DELAY:
    MOV     R2, A
    CLR     A
    MOV     R0, A
    MOV     R1, A
DELAY1:
    DJNZ   R0, DELAY1
    DJNZ   R1, DELAY1
    DJNZ   R2, DELAY1
    RET
END

```

---



---

## 10.6.2 Demo Program (Demonstrate in Polling Mode)

There are two example procedures using polling mode to demonstrate A/D conversion, one written in C language and the other in assembly language.

### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program of ADC -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

#define FOSC 18432000L
#define BAUD 9600

typedef unsigned char    BYTE;
typedef unsigned int     WORD;

#define URMD 0           //0: select T2 as UART1 baud-rate generator
                        //1: select T1 as UART1 baud-rate generator(T1 as 16-bit auto-relaod timer/counter)
                        //2: select T1 as UART1 baud-rate generator (T1 as 8-bit auto-relaod timer/counter)

sfr    T2H    =    0xd6;
sfr    T2L    =    0xd7;

sfr    AUXR   =    0x8e;           //Auxiliary register

/*Declare SFR associated with the ADC */
sfr    ADC_CONTR = 0xBC;           //ADC control register
sfr    ADC_RES   = 0xBD;           //ADC high 8-bit result register
sfr    ADC_LOW2  = 0xBE;           //ADC low 2-bit result register
sfr    P1ASF    = 0x9D;           //P1 secondary function control register
```

---

```

/*Define ADC operation const for ADC_CONTR*/
#define ADC_POWER    0x80           //ADC power control bit
#define ADC_FLAG     0x10           //ADC complete flag
#define ADC_START    0x08           //ADC start control bit
#define ADC_SPEEDLL  0x00           //540 clocks
#define ADC_SPEEDL   0x20           //360 clocks
#define ADC_SPEEDH   0x40           //180 clocks
#define ADC_SPEEDHH  0x60           //90 clocks

void InitUart();
void InitADC();
void SendData(BYTE dat);
BYTE GetADCResult(BYTE ch);
void Delay(WORD n);
void ShowResult(BYTE ch);

void main()
{
    InitUart();                //Init UART, use to show ADC result
    InitADC();                 //Init ADC sfr
    while (1)
    {
        ShowResult(0);        //Show Channel0
        ShowResult(1);        //Show Channel1
        ShowResult(2);        //Show Channel2
        ShowResult(3);        //Show Channel3
        ShowResult(4);        //Show Channel4
        ShowResult(5);        //Show Channel5
        ShowResult(6);        //Show Channel6
        ShowResult(7);        //Show Channel7
    }
}

/*-----
Send ADC result to UART
-----*/
void ShowResult(BYTE ch)
{
    SendData(ch);              //Show Channel NO.
    SendData(GetADCResult(ch)); //Show ADC high 8-bit result

    //if you want show 10-bit result, uncomment next line
    // SendData(ADC_LOW2);      //Show ADC low 2-bit result
}

```

---

---

```

/*-----
Get ADC result
-----*/
BYTE GetADCResult(BYTE ch)
{
    ADC_CONTR = ADC_POWER | ADC_SPEEDLL | ch | ADC_START;
    _nop_(); //Must wait before inquiry
    _nop_();
    _nop_();
    _nop_();
    while (!(ADC_CONTR & ADC_FLAG)); //Wait complete flag
    ADC_CONTR &= ~ADC_FLAG; //Close ADC

    return ADC_RES; //Return ADC result
}

/*-----
Initial UART
-----*/
void InitUart()
{
    SCON = 0x5a; //set UART1 as 8-bit UART with variable baud-rate
#if URMD == 0
    T2L = 0xd8; //Set the preload value
    T2H = 0xff; //115200 bps(65536-18432000/4/115200)
    AUXR = 0x14; //T2 in 1T mode, and run T2
    AUXR |= 0x01; //select T2 as UART1 baud rate generator
#elif URMD == 1
    AUXR = 0x40; //T1 in 1T mode
    TMOD = 0x00; //T1 in mode 0 (16-bit auto-reload timer/counter)
    TL1 = 0xd8; //Set the preload value
    TH1 = 0xff; //115200 bps(65536-18432000/4/115200)
    TR1 = 1; //run T1
#else
    TMOD = 0x20; //T1 in mode 2 (8-bit auto-reload timer/counter)
    AUXR = 0x40; //T1 in 1T mode
    TH1 = TL1 = 0xfb; //115200 bps(256 - 18432000/32/115200)
    TR1 = 1;
#endif
}

/*-----
Initial ADC sfr
-----*/

```

---

---

```
void InitADC()
{
    P1ASF = 0xff;                //Open 8 channels ADC function
    ADC_RES = 0;                 //Clear previous result
    ADC_CONTR = ADC_POWER | ADC_SPEEDLL;
    Delay(2);                    //ADC power-on and delay
}
```

```
/*-----
Send one byte data to PC
Input: dat (UART data)
Output:-
-----*/
```

```
void SendData(BYTE dat)
{
    while (!TI);                //Wait for the previous data is sent
    TI = 0;                     //Clear TI flag
    SBUF = dat;                 //Send current data
}
```

```
/*-----
Software delay function
-----*/
```

```
void Delay(WORD n)
{
    WORD x;
    while (n--)
    {
        x = 5000;
        while (x--);
    }
}
```

---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program EEPROM/IAP -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#define URMD 0 //0: select T2 as UART1 baud-rate generator
//1: select T1 as UART1 baud-rate generator(T1 as 16-bit auto-relaod timer/counter)
//2: select T1 as UART1 baud-rate generator (T1 as 8-bit auto-relaod timer/counter)

T2H DATA 0D6H
T2L DATA 0D7H
AUXR DATA 08EH //Auxiliary register

/*Declare SFR associated with the ADC */
ADC_CONTR EQU 0BCH ;ADC control registe
ADC_RES EQU 0BDH ;ADC high 8-bit result register
ADC_LOW2 EQU 0BEH ;ADC low 2-bit result register
PIASF EQU 09DH ;P1 secondary function control register

/*Define ADC operation const for ADC_CONTR*/
ADC_POWER EQU 80H ;ADC power control bit
ADC_FLAG EQU 10H ;ADC complete flag
ADC_START EQU 08H ;ADC start control bit
ADC_SPEEDLL EQU 00H ;540 clocks
ADC_SPEEDL EQU 20H ;360 clocks
ADC_SPEEDH EQU 40H ;180 clocks
ADC_SPEEDHH EQU 60H ;90 clocks

;-----
ORG 0000H
LJMP MAIN

;-----
ORG 0100H
MAIN:
LCALL INIT_UART ;Init UART, use to show ADC result
LCALL INIT_ADC ;Init ADC sfr
```

---

```

;-----
NEXT:
    MOV    A,    #0
    LCALL  SHOW_RESULT      ;Show channel 0 result
    MOV    A,    #1
    LCALL  SHOW_RESULT      ;Show channel 1 result
    MOV    A,    #2
    LCALL  SHOW_RESULT      ;Show channel 2 result
    MOV    A,    #3
    MOV    A,    #3
    LCALL  SHOW_RESULT      ;Show channel3 result
    MOV    A,    #4
    LCALL  SHOW_RESULT      ;Show channel4 result
    MOV    A,    #5
    LCALL  SHOW_RESULT      ;Show channel5 result
    MOV    A,    #6
    LCALL  SHOW_RESULT      ;Show channel6 result
    MOV    A,    #7
    LCALL  SHOW_RESULT      ;Show channel7 result

    SJMP   NEXT

;-----
;Send ADC result to UART
;Input: ACC (ADC channel NO.)
;Output:-
;-----*/
SHOW_RESULT:
    LCALL  SEND_DATA        ;Show Channel NO.
    LCALL  GET_ADC_RESULT   ;Get high 8-bit ADC result
    LCALL  SEND_DATA        ;Show result
; //if you want show 10-bit result, uncomment next 2 lines
;    MOV    A,    ADC_LOW2    ;Get low 2-bit ADC result
;    LCALL  SEND_DATA        ;Show result
    RET

;-----
;Read ADC conversion result
;Input: ACC (ADC channel NO.)
;Output:ACC (ADC result)
;-----*/
GET_ADC_RESULT:
    ORL    A,    #ADC_POWER | ADC_SPEEDLL | ADC_START
    MOV    ADC_CONTR,A      ;Start A/D conversion
    NOP                    ;Must wait before inquiry

```

---

---

```

        NOP
        NOP
        NOP
WAIT:   MOV     A,ADC_CONTR                ;Wait complete flag
        JNB   ACC.4, WAIT                ;ADC_FLAG(ADC_CONTR.4)
        ANL   ADC_CONTR, #NOT ADC_FLAG   ;Clear ADC_FLAG
        MOV   A,    ADC_RES              ;Return ADC result
        RET

```

```

; /*-----
;Initial ADC sfr
;-----*/

```

```

INIT_ADC:
        MOV   P1ASF, #0FFH                ;Open 8 channels ADC function
        MOV   ADC_RES,    #0                ;Clear previous result
        MOV   ADC_CONTR, #ADC_POWER | ADC_SPEEDLL
        MOV   A,    #2                    ;ADC power-on and delay
        LCALL DELAY
        RET

```

```

; /*-----
;Initial UART
;-----*/

```

```

INIT_UART:
        MOV   SCON, #5AH                ;set UART1 as 8-bit UART with variable baud-rate
#if      URMD == 0
        MOV   T2L, #0D8H                ;Set the preload value (65536-18432000/4/115200)
        MOV   T2H, #0FFH
        MOV   AUXR, #14H                ;T2 in 1T mode, and run T2
        ORL   AUXR, #01H                ;select T2 as UART1 baud rate generator
#elif   URMD == 1
        MOV   AUXR, #40H                ;T1 in 1T mode
        MOV   TMOD, #00H                ;T1 in mode 0 (16-bit auto-reload timer/counter)
        MOV   TL1, #0D8H                ;Set the preload value(65536-18432000/4/115200)
        MOV   TH1, #0FFH
        SETB  TR1                        ;run T1
#else
        MOV   TMOD, #20H                ;T1 in mode 2 (8-bit auto-reload timer/counter)
        MOV   AUXR, #40H                ;T1 in 1T mode
        MOV   TL1, #0FBH                ;115200 bps(256 - 18432000/32/115200)
        MOV   TH1, #0FBH
        SETB  TR1
#endif
        RET

```

---

```

; /*-----
;Send one byte data to PC
;Input: ACC (UART data)
;Output:-
;-----*/
SEND_DATA:
    NB    TI,    $           ;Wait for the previous data is sent
    CLR   TI           ;Clear TI flag
    MOV   SBUF,  A         ;Send current data
    RET

; /*-----
;Software delay function
;-----*/
DELAY:
    MOV   R2,    A
    CLR   A
    MOV   R0,    A
    MOV   R1,    A
DELAY1:
    DJNZ  R0,    DELAY1
    DJNZ  R1,    DELAY1
    DJNZ  R2,    DELAY1
    RET

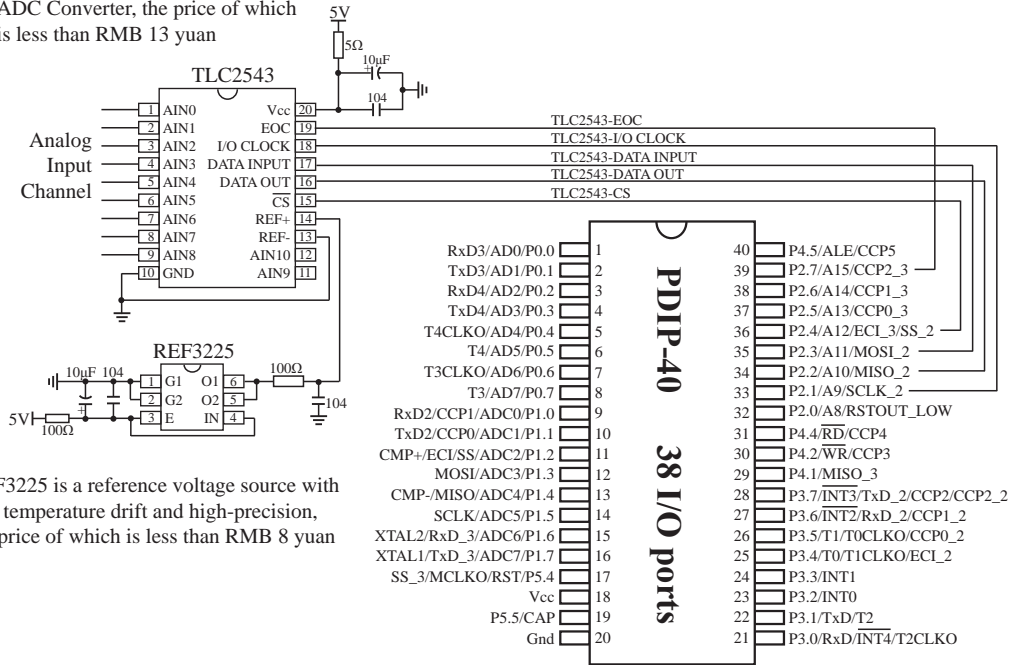
    END

```



## 10.7 Circuit Diagram using SPI to Extend 12-bit ADC(TLC2543)

TLC2543 is a high-precision 12-bit ADC Converter, the price of which is less than RMB 13 yuan



# Chapter 11 Application of CCP/PCA/PWM/DAC

STC15W4K32S4 series MCU has two 16-bit capture/compare modules associated with CCP/PCA/PWM. PCA stands for the Programmable Counter Array. Each of the modules can be programmed to operate in one of four modes: rising and/or falling edge capture(calculator of duty length for high/low pulse), software timer, high-speed pulse output, or pulse width modulator.

For STC15W4K32S4 series MCU, thier CCP/PWM/PCA all can be switched in 3 groups of pins :

- [CCP0/P1.1, CCP1/P1.0];
- [CCP0\_2/P3.5, CCP1\_2/P3.6];
- [CCP0\_3/P2.5, CCP1\_3/P2.6].

## 11.1 Special Function Registers related with CCP/PCA/PWM

CCP/PCA/PWM SFRs table

Mnemonic	Description	Add	Bit address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
CCON	PCA Control Register	D8H	CF	CR	-	-	-	-	CCF1	CCF0	00xx,xx00
CMOD	PCA Mode Register	D9H	CIDL	-	-	-	CPS2	CPS1	CPS0	ECF	0xxx,0000
CCAPM0	PCA Module 0 Mode Register	DAH	-	ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAPM1	PCA Module 1 Mode Register	DBH	-	ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CL	PCA Base Timer Low	E9H									0000,0000
CH	PCA Base Timer High	F9H									0000,0000
CCAP0L	PCA Module-0 Capture Register Low	EAH									0000,0000
CCAP0H	PCA Module-0 Capture Register High	FAH									0000,0000
CCAP1L	PCA Module-1 Capture Register Low	EBH									0000,0000
CCAP1H	PCA Module-1 Capture Register High	FBH									0000,0000
PCA_PWM0	PCA PWM Mode Auxiliary Register 0	F2H	EBS0_1	EBS0_0	-	-	-	-	EPC0H	EPC0L	00xx,xx00
PCA_PWM1	PCA PWM Mode Auxiliary Register 1	F3H	EBS1_1	EBS1_0	-	-	-	-	EPC1H	EPC1L	00xx,xx00
AUXR1 P_SW1	Auxiliary Register 1	A2H	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	-	DPS	0100,0000

## 1. PCA Operation Mode register: CMOD (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	D9H	name	CIDL	-	-	-	CPS2	CPS1	CPS0	ECF

CIDL : PCA Counter control bit in Idle mode.

If CIDL=0, the PCA counter will continue functioning during idle mode.

If CIDL=1, the PCA counter will be gated off during idle mode.

CPS2, CPS1, CPS0 : PCA Counter Pulse source Select bits.

CPS2	CPS1	CPS0	Select PCA/PWM clock source
0	0	0	0, System clock/12, SYSclk/12
0	0	1	1, System clock/2, SYSclk/2
0	1	0	2, Timer 0 overflow. PCA/PWM clock can up to SYSclk because Timer 0 can operate in 1T mode. Frequency-adjustable PWM output can be achieved by changing the Timer 0 overflow.
0	1	1	3, Exrenal clock from ECI/P1.2 (or P3.4 or P2.4) pin (max speed = SYSclk/2)
1	0	0	4, System clock, SYSclk
1	0	1	5, System clock/4, SYSclk/4
1	1	0	6, System clock/6, SYSclk/6
1	1	1	7, System clock/8, SYSclk/8

For example, If CPS2/CPS1/CPS0=1/0/0, PCA/PWM clock source is SYSclk.

If users need to select SYSclk/3 as PCA clock source, Timer 0 should be set to operate in 1T mode and generate an overflow every 3 counting pulse.

erflow interrupt Enable bit.

ECF=0 disables CF bit in CCON to generate an interrupt.

ECF=1 enables CF bit in CCON to generate an interrupt.

## 2. PCA Control register : CCON (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CCON	D8H	name	CF	CR	-	-	-	-	CCF1	CCF0

CF : PCA Counter overflow flag. Set by hardware when the counter rolls over. CF flags an interrupt if bit ECF in CMOD is set. CF may be set by either hardware or software but can only be cleared by software.

CR : PCA Counter Run control bit. Set by software to turn the PCA counter on. Must be cleared by software to turn the PCA counter off.

CCF1 : PCA Module 1 interrupt flag. Set by hardware when a match or capture from module 1 occurs. Must be cleared by software. A match means the value of the PCA counter equals the value of the Capture/Compare register in module 1. A capture means a specific edge from CCP1 happens, so the Capture/Compare register latches the value of the PCA counter, and the CCF1 is set.

CCF0 : PCA Module 0 interrupt flag. Set by hardware when a match or capture from module 0 occurs. Must be cleared by software. A match means the value of the PCA counter equals the value of the Capture/Compare register in module 0. A capture means a specific edge from CCP0 happens, so the Capture/Compare register latches the value of the PCA counter, and the CCF0 is set.

### 3. PCA Capture/Compare register CCAPM0 and CCAPM1

Each module in the PCA has a special function register associated with it. These registers are CCAPMn, n=0 ~1. CCAPM0 for module 0 and CCAPM1 for module 1. The register contains the bits that control the mode in which each module will operate. The ECCFn bit enables the CCFn flag in the CCON SFR to generate an interrupt when a match or compare occurs in the associated module. PWMn enables the pulse width modulation mode. The TOGn bit when set causes the CCPn output associated with the module to toggle when there is a match between the PCA counter and the module's capture/compare register. The match bit(MATn) when set will cause the CCFn bit in the CCON register to be set when there is a match between the PCA counter and the module's capture/compare register.

The next two bits CAPNn and CAPPn determine the edge that a capture input will be active on. The CAPNn bit enables the negative edge, and the CAPPn bit enables the positive edge. If both bits are set, both edges will be enabled and a capture will occur for either transition. The bit ECOMn when set enables the comparator function.

#### Capture/Compare register of PCA module 0 : CCAPM0 (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CCAPM0	DAH	name	-	ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0

B7 : Reserved.

ECOM0 : Comparator Enable bit.

ECOM0=0 disables the comparator function;

ECOM0=1 enables the comparator function.

CAPP0 : Capture Positive control bit.

CAPP0=1 enables positive edge capture.

CAPN0 : Capture Negative control bit.

CAPN0=1 enables negative edge capture.

MAT0 : Match control bit.

When MAT0 = 1, a match of the PCA counter with this module's compare/capture register causes the CCF0 bit in CCON to be set.

TOG0 : Toggle control bit.

When TOG0=1, a match of the PCA counter with this module's compare/capture register causes the CCP0 pin to toggle.

(CCP0/PCA0/PWM0/P1.1 or CCP0\_2/PCA0/PWM0/P3.5 or CCP0\_3/PCA0/PWM0/P2.5)

PWM0 : Pulse Width Modulation.

PWM0=1 enables the CCP0 pin to be used as a pulse width modulated output.

(CCP0/PCA0/PWM0/P1.1 or CCP0\_2/PCA0/PWM0/P3.5 or CCP0\_3/PCA0/PWM0/P2.5)

ECCF0 : Enable CCF0 interrupt.

Enables compare/capture flag CCF0 in the CCON register to generate an interrupt.

---

**Capture/Compare register of PCA module 1 : CCAPM1 (Non bit-addressable)**

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CCAPM1	DBH	name	-	ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1

ECOM1 : Comparator Enable bit.

ECOM1=0 disables the comparator function;

ECOM1=1 enables the comparator function.

CAPP1 : Capture Positive control bit.

CAPP1=1 enables positive edge capture.

CAPN1 : Capture Negative control bit.

CAPN1=1 enables negative edge capture.

MAT1 : Match control bit.

When MAT1 = 1, a match of the PCA counter with this module's compare/capture register causes the CCF1 bit in CCON to be set.

TOG1 : Toggle control bit.

When TOG1=1, a match of the PCA counter with this module's compare/capture register causes the CCP1 pin to toggle.

(CCP1/PCA1/PWM1/P1.0 or CCP1\_2/PCA1/PWM1/P3.6 or CCP1\_3/PCA1/PWM1/P2.6)

PWM1 : Pulse Width Modulation.

PWM1=1 enables the CCP1 pin to be used as a pulse width modulated output.

(CCP1/PCA1/PWM1/P1.0 or CCP1\_2/PCA1/PWM1/P3.6 or CCP1\_3/PCA1/PWM1/P2.6)

ECCF1 : Enable CCF1 interrupt.

Enables compare/capture flag CCF1 in the CCON register to generate an interrupt.

#### 4. PCA 16-bit Counter — low 8-bit CL and high 8-bit CH

The addresses of CL and CH respectively are E9H and F9H, and their reset value both are 00H. CL and CH are used to save the PCA load value.

#### 5. PCA Capture/Compare register — CCAPnL and CCAPnH

When PCA is used to capture/compare, CCAPnL and CCAPnH are used to save the 16-bit capture value in corresponding block. When PCA is operated in PWM mode, CCAPnL and CCAPnH are used to control the duty cycle of PWM output signal. "n=0 or 1" respectively stand for module 0 and 1. Reset value of registers CCAPnL and CCAPnH are both 00H. Their addresses respectively are:

CCAP0L — EAH, CCAP0H — FAH : Capture / Compare register of module 0

CCAP1L — EBH, CCAP1H — FBH : Capture / Compare register of module 1

## 6. PWM registers of PCA modules : PCA\_PWM0 and PCA\_PWM1

PCA\_PWM0 : PWM register of PCA module 0

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PCA_PWM0	F2H	name	EBS0_1	EBS0_0	-	-	-	-	EPC0H	EPC0L

EBS0\_1 , EBS0\_0 : Function Select bit when PCA module 0 work as Pulse Width Modulator (PWM)

- 0 , 0 : PCA module 0 is used as 8-bit PWM;
- 0 , 1 : PCA module 0 is used as 7-bit PWM;
- 1 , 0 : PCA module 0 is used as 6-bit PWM;
- 1 , 1 : Invalid, PCA module 0 is still used as 8-bit PWM.

B5 ~ B2 : Reserved

EPC0H : Associated with CCAP0H, it is used in PCA PWM mode.

EPC0L : Associated with CCAP0L, it is used in PCA PWM mode.

PCA\_PWM1 : PWM register of PCA module 1

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
PCA_PWM1	F3H	name	EBS1_1	EBS1_0	-	-	-	-	EPC1H	EPC1L

EBS1\_1 , EBS1\_0 : Function Select bit when PCA module 1 work as Pulse Width Modulator (PWM)

- 0 , 0 : PCA module 1 is used as 8-bit PWM;
- 0 , 1 : PCA module 1 is used as 7-bit PWM;
- 1 , 0 : PCA module 1 is used as 6-bit PWM;
- 1 , 1 : Invalid, PCA module 1 is still used as 8-bit PWM.

B5 ~ B2 : Reserved

EPC1H : Associated with CCAP1H, it is used in PCA PWM mode.

EPC1L : Associated with CCAP1L, it is used in PCA PWM mode.

The operation mode of PCA modules set as shown in the below table.

Setting the operation mode of PCA modules CCAPMn register n = 0,1

EBSn_1	EBSn_0	-	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	Function of PCA modules
X	X		0	0	0	0	0	0	0	No operation
0	0		1	0	0	0	0	1	0	8-bit PWM, no interrupt
0	1		1	0	0	0	0	1	0	7-bit PWM, no interrupt
1	0		1	0	0	0	0	1	0	6-bit PWM, no interrupt
1	1		1	0	0	0	0	1	0	8-bit PWM, no interrupt
0	0		1	1	0	0	0	1	1	8-bit PWM output, interrupt can be generated on rising edge.
0	1		1	1	0	0	0	1	1	7-bit PWM output, interrupt can be generated on rising edge.
1	0		1	1	0	0	0	1	1	6-bit PWM output, interrupt can be generated on rising edge.

Setting the operation mode of PCA modules CCAPMn register n = 0,1 (Continued)

EBSn_1	EBSn_0	-	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	Function of PCA modules
1	1		1	1	0	0	0	1	1	8-bit PWM output, interrupt can be generated on rising edge.
0	0		1	0	1	0	0	1	1	8-bit PWM output, interrupt can be generated on falling edge.
0	1		1	0	1	0	0	1	1	7-bit PWM output, interrupt can be generated on falling edge.
1	0		1	0	1	0	0	1	1	6-bit PWM output, interrupt can be generated on falling edge.
1	1		1	0	1	0	0	1	1	8-bit PWM output, interrupt can be generated on falling edge.
0	0		1	1	1	0	0	1	1	8-bit PWM output, interrupt can be generated on both rising and falling edges.
0	1		1	1	1	0	0	1	1	7-bit PWM output, interrupt can be generated on both rising and falling edges.
1	0		1	1	1	0	0	1	1	6-bit PWM output, interrupt can be generated on both rising and falling edges.
1	1		1	1	1	0	0	1	1	8-bit PWM output, interrupt can be generated on both rising and falling edges.
X	X		X	1	0	0	0	0	X	16-bit Capture Mode, capture triggered by the rising edge on CCPn/PCAn pin
X	X		X	0	1	0	0	0	X	16-bit Capture Mode, capture triggered by the falling edge on CCPn/PCAn pin
X	X		X	1	1	0	0	0	X	16-bit Capture Mode, capture triggered by the transition on CCPn/PCAn pin
X	X		1	0	0	1	0	0	X	16-bit software timer
X	X		1	0	0	1	1	0	X	16-bit high-speed output

## 7. CCP/PCA/PWM Switch Control bits: CCP\_S1 / P\_SW1.5 and CCP\_S0 / P\_SW1.4

AUXR1 / P\_SW1 : Peripheral function switch register (Non bit-addressable)

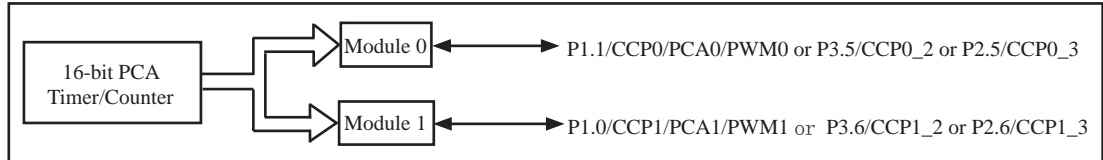
Mnemonic	Add	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0100,0000

CCP can be switched in 3 groups of pins by selecting the control bits CCP_S1 and CCP_S0.		
CCP_S1	CCP_S0	CCP can be switched in P1 and P2 and P3
0	0	CCP on [P1.2/ECL1,P1.1/CCP0,P1.0/CCP1]
0	1	CCP on [P3.4/ECL2,P3.5/CCP0_2,P3.6/CCP1_2]
1	0	CCP on [P2.4/ECL3,P2.5/CCP0_3,P2.6/CCP1_3]
1	1	Invalid

## 11.2 CCP/PCA/PWM Structure

There are 2 channels CCP/PWM/PCA (Programmable Counter Array) in STC15 series MCU. (CCP/PCA/PWM function can be switched from P1 port to P2 port or to P3 port by setting AUXR1/P\_SW1 register).

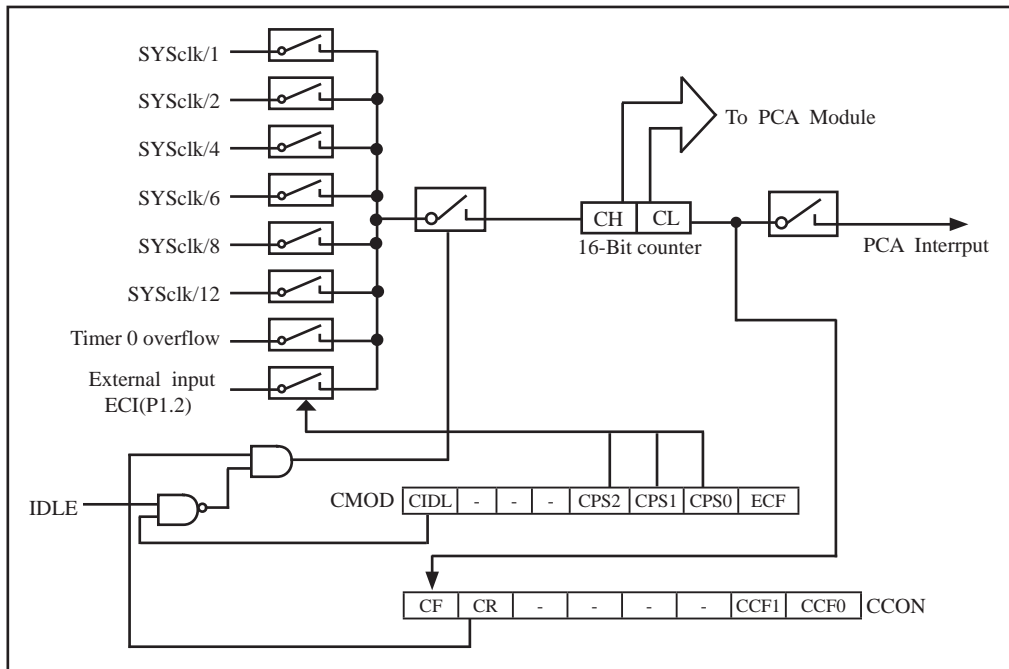
The Programmable Counter Array (PCA) is a special 16-bit Timer that has three 16-bit capture/compare modules associated with it. See the following figure.



Programmable Counter Array Structure

Each PCA/PWM module can be operated in 4 modes : rising/falling capture mode, software timer, high-speed output mode and adjustable pulse output mode.

STC15W4K32S4 series : module 0 connect to P1.1/CCP0 (which can be switched to P3.5/CCP0\_2 or to P2.5/CCP0\_3);  
module 1 connect to P1.0/CCP1 (which can be switched to P3.6/CCP1\_2 or to P2.6/CCP1\_3);



PCA Timer/Counter



---

The contents of registers CH and CL are the count value of 16-bit PCA timer. The PCA timer is a common time base for all three modules and can be programmed to run at 1/12 system clock, 1/8 system clock, 1/6 system clock, 1/2 system clock, system clock, the Timer 0 overflow or the input on ECI pin ( in P1.2 or P2.4 or P3.4). The timer count source is determined from CPS2 and CPS1 and CPS0 bits in the CMOD SFR.

In the CMOD SFR, there are two additional bits associated with the PCA. They are CIDL which allows the PCA to stop during idle mode, and ECF which when set causes an interrupt and the PCA overflow flag CF (in the CCON SFR) to be set when the PCA timer overflows.

The CCON SFR contains the run control bit (CR) for PCA and the flags for the PCA timer (CF) and each module (CCF1/CCF0). To run the PCA the CR bit(CCON.6) must be set by software; oppositely clearing bit CR will shut off PCA. The CF bit(CCON.7) is set when the PCA counter overflows and an interrupt will be generated if the ECF (CMOD.0) bit in the CMOD register is set. The CF bit can only be cleared by software. There are three bits named CCF0 and CCF1 in SFR CCON. The CCF0 and CCF1 are the flags for module 0 and module 1 respectively. They are set by hardware when either a match or a capture occurs. These flags also can only be cleared by software.

Each module in the PCA has a special function register associated with it, CCAPM0 for module-0 and CCAPM1 for module-1 . The register contains the bits that control the mode in which each module will operate.

The ECCFn (n=0,1) bit controls if to pass the interrupt from CCFn flag in the CCON SFR to the MCU when a match or compare occurs in the associated module.

PWMn enables the pulse width modulation mode.

The TOGn bit when set causes the pin CCPn output associated with the module to toggle when there is a match between the PCA counter and the module's Capture/Compare register.

The match bit(MATn) when set will cause the CCFn bit in the CCON register to be set when there is a match between the PCA counter and the module's Capture/Compare register.

The next two bits CAPNn and CAPPn determine the edge type that a capture input will be active on. The CAPNn bit enables the negative edge, and the CAPPn bit enables the positive edge. If both bits are set, both edges will be enabled and a capture will occur for either transition.

The bit ECOMn when set enables the comparator function.

## 11.3 CCP/PCA Modules Operation Mode

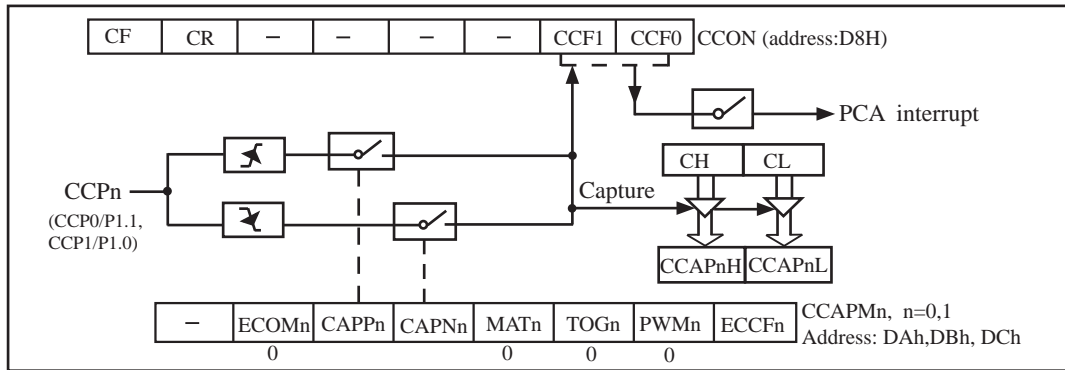
The operation mode of PCA modules set as shown in the below table.

Setting the operation mode of PCA modules CCAPMn register n = 0,1

EBSn_1	EBSn_0	-	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	Function of PCA modules
X	X		0	0	0	0	0	0	0	No operation
0	0		1	0	0	0	0	1	0	8-bit PWM, no interrupt
0	1		1	0	0	0	0	1	0	7-bit PWM, no interrupt
1	0		1	0	0	0	0	1	0	6-bit PWM, no interrupt
1	1		1	0	0	0	0	1	0	8-bit PWM, no interrupt
0	0		1	1	0	0	0	1	1	8-bit PWM output, interrupt can be generated on rising edge.
0	1		1	1	0	0	0	1	1	7-bit PWM output, interrupt can be generated on rising edge.
1	0		1	1	0	0	0	1	1	6-bit PWM output, interrupt can be generated on rising edge.
1	1		1	1	0	0	0	1	1	8-bit PWM output, interrupt can be generated on rising edge.
0	0		1	0	1	0	0	1	1	8-bit PWM output, interrupt can be generated on falling edge.
0	1		1	0	1	0	0	1	1	7-bit PWM output, interrupt can be generated on falling edge.
1	0		1	0	1	0	0	1	1	6-bit PWM output, interrupt can be generated on falling edge.
1	1		1	0	1	0	0	1	1	8-bit PWM output, interrupt can be generated on falling edge.
0	0		1	1	1	0	0	1	1	8-bit PWM output, interrupt can be generated on both rising and falling edges.
0	1		1	1	1	0	0	1	1	7-bit PWM output, interrupt can be generated on both rising and falling edges.
1	0		1	1	1	0	0	1	1	6-bit PWM output, interrupt can be generated on both rising and falling edges.
1	1		1	1	1	0	0	1	1	8-bit PWM output, interrupt can be generated on both rising and falling edges.
X	X		X	1	0	0	0	0	X	16-bit Capture Mode, capture triggered by the rising edge on CCPn/PCAn pin
X	X		X	0	1	0	0	0	X	16-bit Capture Mode, capture triggered by the falling edge on CCPn/PCAn pin
X	X		X	1	1	0	0	0	X	16-bit Capture Mode, capture triggered by the transition on CCPn/PCAn pin
X	X		1	0	0	1	0	0	X	16-bit software timer
X	X		1	0	0	1	1	0	X	16-bit high-speed output

### 11.3.1 CCP/PCA Capture Mode

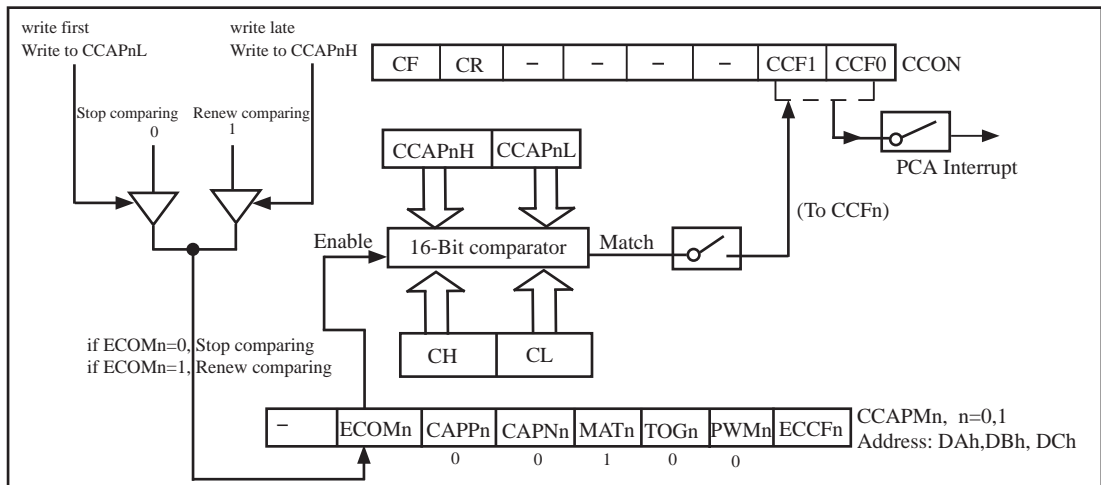
To use one of the PCA modules in the capture mode either one or both of the CCAPM bits – CAPPn and CAPNn, for the module must be set. The external CCPn input (CCP0/P1.1, CCP1/P1.0) for the module is sampled for a transition. When a valid transition occurs, the PCA hardware loads the value of the PCA counter register (CH and CL) into the module’s capture registers (CCAPnH and CCAPnL). If the CCFn bit for the module in the CCON SFR and the ECCFn bit in the CCAPMn SFR are set then an interrupt will be generated.



PCA Capture Mode (PCA Capture mode)

### 11.3.2 16-bit Software Timer Mode

The internal structure diagram of 16-bit software timer mode is shown below.



PCA Software Timer Mode / 16-bit software timer mode / PCA compare mode

The PCA modules can be used as software timers by setting both the ECOMn and MATn bits in the modules CCAPMn register. The PCA timer will be compared to the module’s capture registers and when a match occurs an interrupt will be generated if the CCFn and ECCFn bits for the module are both set.

[CH,CL] is automatically incremented at a certain time which depends on the selected clock source. For example,[CH,CL] is incremented every 12 clock when the clock source is SYSclk/12. When [CH,CL] have been increased to equal the value of register [CCAPnH, CCAPnL], a interrupt request would be generated and CCFn=1 (n=0, 1). The 16-bit software timer intervals depend on the selection of clock source and settings of PCA counter. The following example shows the calculation method of PCA count value.

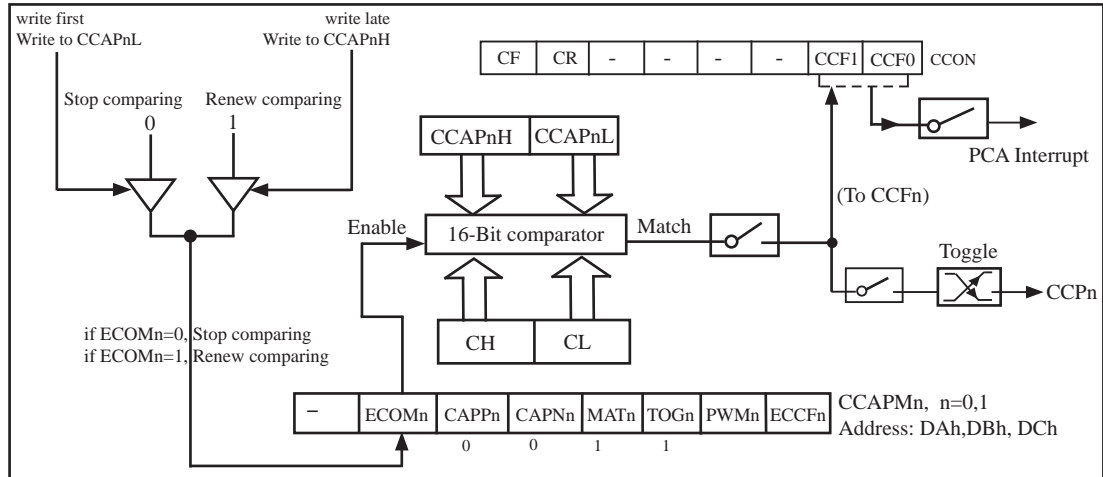
IF the system frequency SYSclk = 18.432MHz and the clock source SYSclk/12 is choosed and the timer intervals T = 5ms, the count value of PCA timer would be computed by the following formula:

$$\begin{aligned} \text{PCA count value} &= T / ((1 / \text{SYSclk}) \times 12) = 0.005 / ((1 / 18432000) \times 12) = 7680 \text{ (decimal)} \\ &= 1E00H \text{ (hexadecimal)} \end{aligned}$$

In other words, when [CH,CL] is incremented to equal 1E00H, the 5ms timer is time out.

### 11.3.3 High Speed Output Mode

In this mode the CCPn output (port latch) associated with the PCA module will toggle each time a match occurs between the PCA counter and the module's capture registers. To activate this mode the TOGn,MATn,and ECOMn bits in the module's CCAPMn SFR must be set.



PCA High-Speed Output Mode

The frequency of output pulse is determined by the value of CCAPn for PCA module n. When the PCA clock source is SYSclk/2, the output pulse frequency F is calculated by:

$$f = \text{SYSclk} / (4 \times \text{CCAPnL})$$

SYSclk stands for system clock frequency in above formula. Consequently  $\text{CCAPnL} = \text{SYSclk} / (4 \times f)$ .

If the computing result is not integer, CCPAPnL should be rounded to the nearest integer:

$$\text{CCAPnL} = \text{INT}(\text{SYSclk} / (4 \times f) + 0.5)$$

For example, if SYSclk = 20MHz, and PCA output 125kHz square wave, CCAPnL would be:

$$\text{CCAPnL} = \text{INT}(20000000 / (4 \times 125000) + 0.5) = \text{INT}(40 + 0.5) = 40 = 28H$$

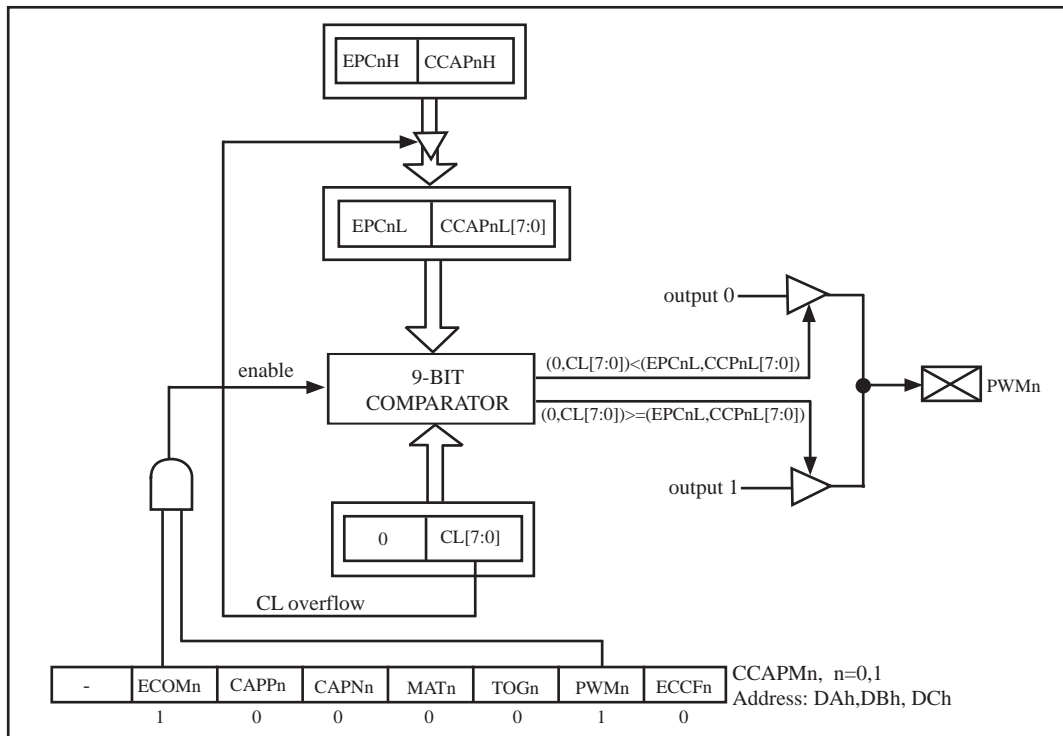
### 11.3.4 Pulse Width Modulator Mode (PWM mode)

Pulse Width Modulator (PWM) is to control waveform duty ratio, cycle and phase wave by software.

PCA module n (n=0,1, the same below) can work in 8-bit PWM mode or 7-bit PWM mode or 6-bit PWM mode by setting the corresponding bits EBSn\_1/PCA\_PWMn.7 and EBSn\_0/PCA\_PWMn.6 in register PCA\_PWMn.

#### 11.3.4.1 8-bit Pulse Width Modulator (PWM mode)

PCA module n (n=0,1) would be used as 8-bit pulse width modulator if [EBSn\_1,EBSn\_0]=[0,0] or [1,1]. And {0,CL[7:0]} would be compared with [EPCnL,CCAPnL[7:0]]. The internal structure diagram of 8-bit PWM mode is shown below.



All of the PCA modules can be used as PWM outputs. The frequency of the output depends on the source for the PCA timer. All of the modules will have the same frequency of output because they all share the same PCA timer. The duty cycle of each module is independently variable using the module's capture register {EPCnL, CCAPnL[7:0]}. When the value of {0,CL[7:0]} is less than the value in the module's {EPCnL,CCAPnL[7:0]} SFR, the output will be low. When it is equal to or greater than , the output will be high. When {0,CL[7:0]} overflows from FFH to 00H, {EPCnL,CCAPnL[7:0]} is reloaded with the value in {EPCnH,CCAPnH[7:0]}. That allows updating the PWM without glitches. The PWMn and ECOMn bits in the module's CCAPMn register must be set to enable the PWM mode.

8-bit PWM: 
$$\text{PWM Frequency} = \frac{\text{Frequency of PCA Clock input source}}{256}$$

PCA clock source may be from : SYSclk, SYSclk/2, SYSclk/4, SYSclk/6, SYSclk/8, SYSclk/12, Timer 0 overflow and input on ECI/P1.2 pin.

Question: find out the value of SYSclk if PCA module work in 8-bit PWM mode and output frequency is 38KHz and SYSclk is used as PCA/PWM clock input source.

Possible solution:  $38000 = \text{SYSclk}/256$  according to the above calculating formula. So the frequency of external clock  $\text{SYSclk} = 38000 \times 256 \times 1 = 9,728,000$

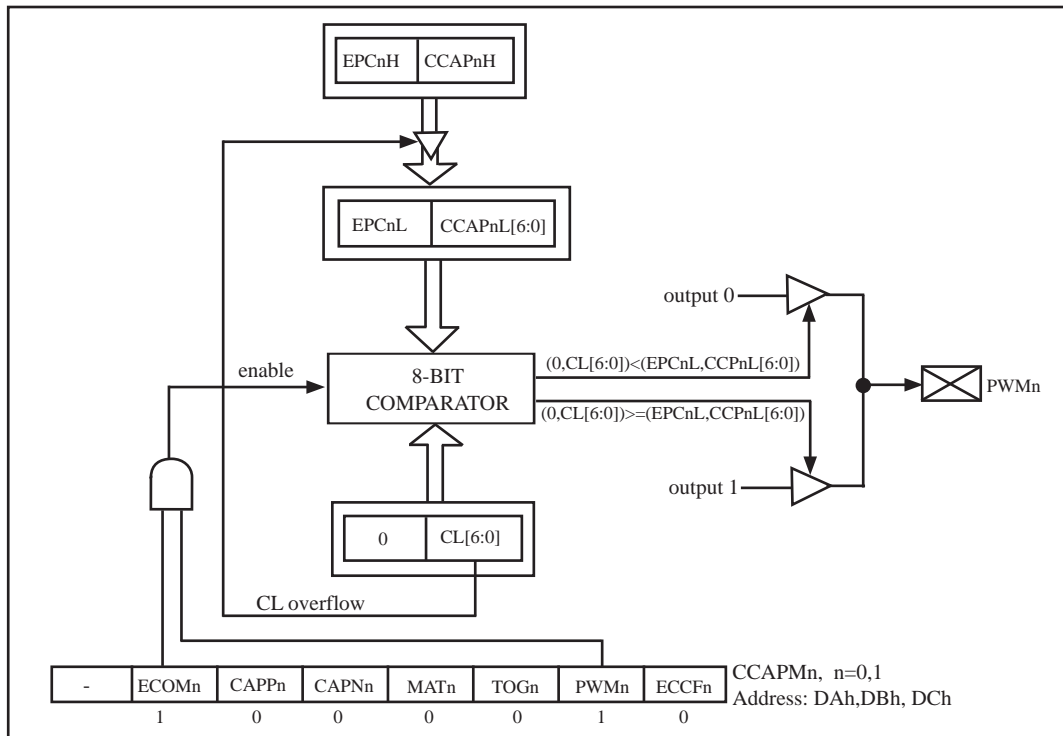
Frequency-adjustable can be achieved by selecting Timer 1 overflow or input from pin ECI as PCA/PWM clock source.

If EPCnL = 0 and CCAPnL = 00H, PWM output high.

If EPCnL = 1 and CCAPnL = FFH, PWM output low.

### 11.3.4.2 7-bit Pulse Width Modulator (PWM mode)

PCA module n (n=0,1) would be used as 7-bit pulse width modulator if [EBSn\_1,EBSn\_0]=[0,1]. And {0,CL[6:0]} would be compared with [EPCnL,CCAPnL[6:0]]. The internal structure diagram of 7-bit PWM mode is shown below.



---

All of the PCA modules can be used as PWM outputs. The frequency of the output depends on the source for the PCA timer. All of the modules will have the same frequency of output because they all share the same PCA timer. The duty cycle of each module is independently variable using the module's capture register {EPCnL, CCAPnL[6:0]}. When the value of {0,CL[6:0]} is less than the value in the module's {EPCnL,CCAPnL[6:0]} SFR, the output will be low. When it is equal to or greater than , the output will be high. When {0,CL[6:0]} overflows from 7FH to 00H, {EPCnL,CCAPnL[6:0]} is reloaded with the value in {EPCnH,CCAPnH[6:0]}. That allows updating the PWM without glitches. The PWMn and ECOMn bits in the module's CCAPMn register must be set to enable the PWM mode.

$$\text{7-bit PWM:} \quad \text{PWM Frequency} = \frac{\text{Frequency of PCA Clock input source}}{128}$$

PCA clock source may be from : SYSclk, SYSclk/2, SYSclk/4, SYSclk/6, SYSclk/8, SYSclk/12, Timer 0 overflow and input on ECI/P1.2 pin.

Question: find out the value of SYSclk if PCA module work in 7-bit PWM mode and output frequency is 38KHz and SYSclk is used as PCA/PWM clock input source.

Possible solution:  $38000 = \text{SYSclk}/128$  according to the above calculating formula. So the frequency of external clock  $\text{SYSclk} = 38000 \times 128 \times 1 = 4,864,000$

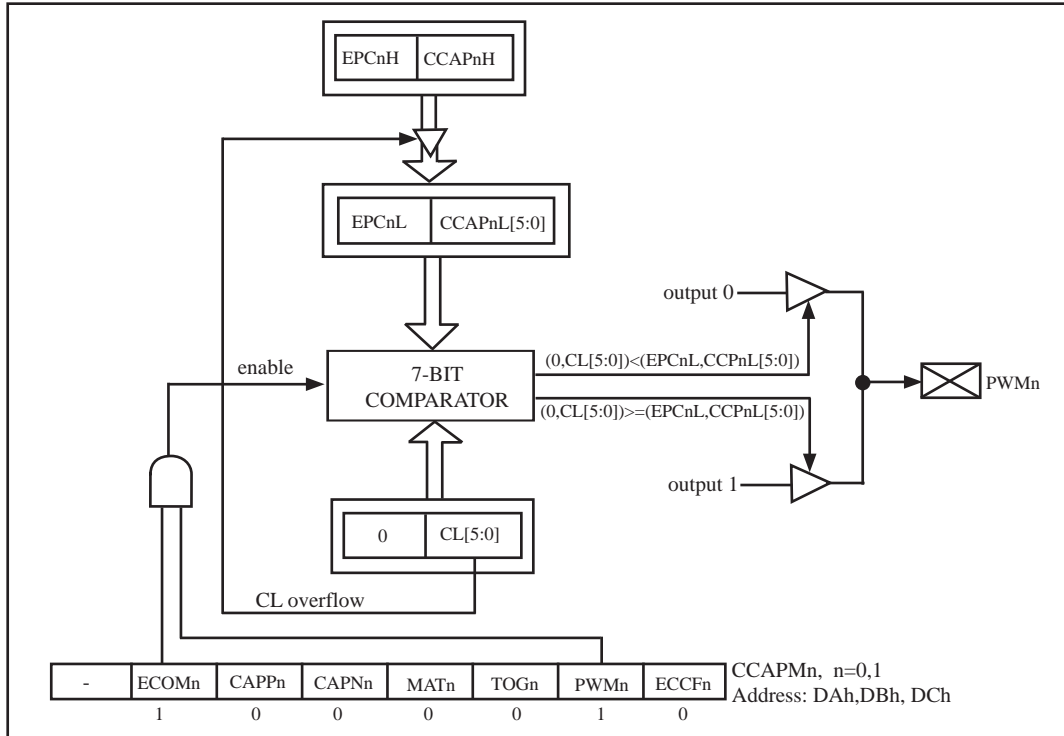
Frequency-adjustable can be achieved by selecting Timer 1 overflow or input from pin ECI as PCA/PWM clock source.

If EPCnL = 0 and CCAPnL = 80H, PWM output high.

If EPCnL = 1 and CCAPnL = FFH, PWM output low.

### 11.3.4.3 6-bit Pulse Width Modulator (PWM mode)

PCA module n (n=0,1) would be used as 6-bit pulse width modulator if [EBSn\_1,EBSn\_0]=[1,0]. And {0,CL[5:0]} would be compared with [EPCnL,CCAPnL[5:0]]. The internal structure diagram of 6-bit PWM mode is shown below.



PCA PWM mode (PCA as 6-bit Pulse Width Modulator)

All of the PCA modules can be used as PWM outputs. The frequency of the output depends on the source for the PCA timer. All of the modules will have the same frequency of output because they all share the same PCA timer. The duty cycle of each module is independently variable using the module's capture register {EPCnL, CCAPnL[5:0]}. When the value of {0,CL[5:0]} is less than the value in the module's {EPCnL, CCAPnL[5:0]} SFR, the output will be low. When it is equal to or greater than , the output will be high. When {0,CL[5:0]} overflows from 3FH to 00H, {EPCnL,CCAPnL[5:0]} is reloaded with the value in {EPCnH,CCAPnH[5:0]}. That allows updating the PWM without glitches. The PWMn and ECOMn bits in the module's CCAPMn register must be set to enable the PWM mode.

$$\text{6-bit PWM:} \quad \text{PWM Frequency} = \frac{\text{Frequency of PCA Clock input source}}{64}$$

PCA clock source may be from : SYSclk, SYSclk/2, SYSclk/4, SYSclk/6, SYSclk/8, SYSclk/12, Timer 0 overflow and input on ECI/P1.2 pin.



---

Question: find out the value of SYSclk if PCA module work in 6-bit PWM mode and output frequency is 38KHz and SYSclk is used as PCA/PWM clock input source.

Possible solution:  $38000 = \text{SYSclk}/64$  according to the above calculating formula. So the frequency of external clock  $\text{SYSclk} = 38000 \times 64 \times 1 = 2,432,000$

Frequency-adjustable can be achieved by selecting Timer 1 overflow or input from pin ECI as PCA/PWM clock source.

If EPCnL = 0 and CCAPnL = COH, PWM output high.

If EPCnL = 1 and CCAPnL = FFH, PWM output low.

C language and

## 11.4 Program using CCP/PCA to Extend External Interrupt

There are two demo programs for CCP/PCA module extended external interrupt, one written in the other in assembly language.

### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using CCP/PCA to extend external interrupt (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

//This demo program take CCP/PCA module 0 for example. the use of CCP/PCA module 1 and CCP/PCA module
//2 are same as CCP/PCA module 0

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

#define FOSC 18432000L

typedef unsigned char BYTE;
typedef unsigned int WORD;
typedef unsigned long DWORD;

sfr P_SW1 = 0xA2; //Peripheral Function Switch register 1

#define CCP_S0 0x10 //P_SW1.4
#define CCP_S1 0x20 //P_SW1.5

sfr CCON = 0xD8; //PCA Control Register
sbit CCF0 = CCON^0; //the interrupt request flag of PCA module 0
sbit CCF1 = CCON^1; //the interrupt request flag of PCA module 1
sbit CR = CCON^6; //the run bit of PCA timer
```

---

```

sbit  CF      =      CCON^7;           //the overflow flag of PCA timer
sfr   CMOD    =      0xD9;           //PCA Mode register
sfr   CL      =      0xE9;
sfr   CH      =      0xF9;
sfr   CCAPM0  =      0xDA;
sfr   CCAP0L  =      0xEA;
sfr   CCAP0H  =      0xFA;
sfr   CCAPM1  =      0xDB;
sfr   CCAP1L  =      0xEB;
sfr   CCAP1H  =      0xFB;
sfr   PCAPWM0 =      0xF2;
sfr   PCAPWM1 =      0xF3;
sbit  PCA_LED =      P1^0;           //PCA test LED

void PCA_isr() interrupt 7 using 1
{
    CCF0 = 0;                          //clear the interrupt request flag
    PCA_LED = !PCA_LED;
}

void main()
{
    ACC    =      P_SW1;
    ACC    &=      ~(CCP_S0 | CCP_S1); //CCP_S0=0 CCP_S1=0
    P_SW1  =      ACC;                  //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1 )

//    ACC    =      P_SW1;
//    ACC    &=      ~(CCP_S0 | CCP_S1); //CCP_S0=1 CCP_S1=0
//    ACC    |=      CCP_S0;            //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2 )
//    P_SW1  =      ACC;
//
//    ACC    =      P_SW1;
//    ACC    &=      ~(CCP_S0 | CCP_S1); //CCP_S0=0 CCP_S1=1
//    ACC    |=      CCP_S1;            //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3 )
//    P_SW1  =      ACC;

    CCON   =      0;                    //Initialize the PCA control register
                                           //disable PCA timer
                                           //clear CF bit
                                           //clear the interrupt request flag

    CL     =      0;                    //reset PCA timer
    CH     =      0;
    CMOD   =      0x00;

    CCAPM0 =      0x11;                 //PCA module 0 can be activated on falling edge
//    CCAPM0 =      0x21;                 //PCA module 0 can be activated on rising edge
//    CCAPM0 =      0x31;                 //PCA module 0 can be activated
                                           //both on falling and rising edge

    CR     =      1;                    //run PCA timer
    EA     =      1;
    while (1);
}

```

---

---

## 2.Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using CCP/PCA to extend external interrupt (falling edge) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

//This demo program take CCP/PCA module 0 for example. the use of CCP/PCA module 1 and CCP/PCA module
//2 are same as CCP/PCA module 0

P_SW1 EQU 0A2H //Peripheral Function Switch register 1
CCP_S0 EQU 10H //P_SW1.4
CCP_S1 EQU 20H //P_SW1.5

CCON EQU 0D8H ;PCA Control Register
CCF0 BIT CCON.0 ;the interrupt request flag of PCA module 0
CCF1 BIT CCON.1 ;the interrupt request flag of PCA module 1
CR BIT CCON.6 ;the run bit of PCA timer
CF BIT CCON.7 ;the overflow flag of PCA timer
CMOD EQU 0D9H ;PCA Mode register
CL EQU 0E9H
CH EQU 0F9H

CCAPM0 EQU 0DAH
CCAP0L EQU 0EAH
CCAP0H EQU 0FAH
CCAPM1 EQU 0DBH
CCAP1L EQU 0EBH
CCAP1H EQU 0FBH

PCA_PWM0 EQU 0F2H
PCA_PWM1 EQU 0F3H

PCA_LED BIT P1.0 ;PCA test LED
;-----
ORG 0000H
LJMP MAIN

ORG 003BH
PCA_ISR:
PUSH PSW
```

```

PUSH ACC
CKECK_CCF0:
JNB CCF0, PCA_ISR_EXIT
CLR CCF0 ;clear the interrupt request flag
CPL PCA_LED
PCA_ISR_EXIT:
POP ACC
POP PSW
RETI
;-----
ORG 0100H
MAIN:
MOV SP, #5FH

MOV A, P_SW1
ANL A, #0CFH //CCP_S0=0 CCP_S1=0
MOV P_SW1, A //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1)

// MOV A, P_SW1
// ANL A, #0CFH //CCP_S0=1 CCP_S1=0
// ORL A, #CCP_S0 //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2)
// MOV P_SW1, A
//
// MOV A, P_SW1
// ANL A, #0CFH //CCP_S0=0 CCP_S1=1
// ORL A, #CCP_S1 //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3)
// MOV P_SW1, A

MOV CCON, #0 ;Initialize the PCA control register
;disable PCA timer
;clear CF bit
;clear the interrupt request flag

CLR A ;
MOV CL, A ;reset PCA timer
MOV CH, A ;
MOV CMOD, #00H

MOV CCAPM0, #11H ;PCA module 0 capture the falling edge of CCP0(P1.3) pin
; MOV CCAPM0, #21H ;PCA module 0 capture the rising edge of CCP0(P1.3) pin
; MOV CCAPM0, #31H ;PCA module 0 capture falling as well as
;rising edge of CCP0(P1.3) pin
;-----
SETB CR ;run PCA timer
SETB EA

SJMP $
;-----
END

```

---

## 11.5 Demo Program for CCP/PCA acted as 16-bit Timer

There are two programs for PCA module acted as 16-bit software Timer demo, one written in C language and the other in assembly language.

### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using CCP/PCA as 16-bit software Timer -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

#define FOSC 18432000L
#define T100Hz (FOSC / 12 / 100)

typedef unsigned char BYTE;
typedef unsigned int WORD;

sfr P_SW1 = 0xA2; //Peripheral function switch register 1

#define CCP_S0 0x10 //P_SW1.4
#define CCP_S1 0x20 //P_SW1.5

/*Declare SFR associated with the PCA */
sfr CCON = 0xD8; //PCA control register
sbit CCF0 = CCON^0; //PCA module-0 interrupt flag
sbit CCF1 = CCON^1; //PCA module-1 interrupt flag
sbit CR = CCON^6; //PCA timer run control bit
sbit CF = CCON^7; //PCA timer overflow flag
sfr CMOD = 0xD9; //PCA mode register
sfr CL = 0xE9; //PCA base timer LOW
sfr CH = 0xF9; //PCA base timer HIGH
```

---

```

sfr    CCAPM0    =    0xDA;           //PCA module-0 mode register
sfr    CCAP0L    =    0xEA;           //PCA module-0 capture register LOW
sfr    CCAP0H    =    0xFA;           //PCA module-0 capture register HIGH
sfr    CCAPM1    =    0xDB;           //PCA module-1 mode register
sfr    CCAP1L    =    0xEB;           //PCA module-1 capture register LOW
sfr    CCAP1H    =    0xFB;           //PCA module-1 capture register HIGH

sfr    PCAPWM0   =    0xf2;
sfr    PCAPWM1   =    0xf3;

sbit   PCA_LED   =    P1^0;           //PCA test LED

```

```

BYTE   cnt;
WORD   value;

```

```

void PCA_isr() interrupt 7 using 1

```

```

{
    CCF0 = 0;           //Clear interrupt flag
    CCAP0L = value;
    CCAP0H = value >> 8; //Update compare value
    value += T100Hz;
    if (cnt-- == 0)
    {
        cnt = 100;     //Count 100 times
        PCA_LED = !PCA_LED; //Flash once per second
    }
}

```

```

void main()

```

```

{
    ACC    =    P_SW1;
    ACC    &=    ~(CCP_S0 | CCP_S1); //CCP_S0=0 CCP_S1=0
    P_SW1  =    ACC; // (P1.2/ECI, P1.1/CCP0, P1.0/CCP1)

// ACC    =    P_SW1;
// ACC    &=    ~(CCP_S0 | CCP_S1); //CCP_S0=1 CCP_S1=0
// ACC    |=    CCP_S0; // (P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2)
// P_SW1  =    ACC;
//
// ACC    =    P_SW1;
// ACC    &=    ~(CCP_S0 | CCP_S1); //CCP_S0=0 CCP_S1=1
// ACC    |=    CCP_S1; // (P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3)
// P_SW1  =    ACC;

```

---

```

CCON = 0; //Initial PCA control register
//PCA timer stop running
//Clear CF flag
//Clear all module interrupt flag
CL = 0; //Reset PCA base timer
CH = 0;
CMOD = 0x00; //Set PCA timer clock source as Fosc/12
//Disable PCA timer overflow interrupt

value = T100Hz;
CCAP0L = value;
CCAP0H = value >> 8; //Initial PCA module-0
value += T100Hz;
CCAPM0 = 0x49; //PCA module-0 work in 16-bit timer mode
//and enable PCA interrupt

CR = 1; //PCA timer start run
EA = 1;
cnt = 0;

while (1);
}

```

## 2.Assembler Listing

```

/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using CCP/PCA as 16-bit software Timer -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

T100Hz EQU 3C00H ;(18432000 / 12 / 100)
P_SW1 EQU 0A2H ;Peripheral function switch register1

CCP_S0 EQU 10H ;P_SW1.4
CCP_S1 EQU 20H ;P_SW1.5

```

---

```

; /*Declare SFR associated with the PCA */
CCON      EQU    0D8H      ;PCA control register
CCF0      BIT    CCON.0    ;PCA module-0 interrupt flag
CCF1      BIT    CCON.1    ;PCA module-1 interrupt flag
CR        BIT    CCON.6    ;PCA timer run control bit
CF        BIT    CCON.7    ;PCA timer overflow flag
CMOD      EQU    0D9H      ;PCA mode register
CL        EQU    0E9H      ;PCA base timer LOW
CH        EQU    0F9H      ;PCA base timer HIGH
CCAPM0    EQU    0DAH      ;PCA module-0 mode register
CCAP0L    EQU    0EAH      ;PCA module-0 capture register LOW
CCAP0H    EQU    0FAH      ;PCA module-0 capture register HIGH
CCAPM1    EQU    0DBH      ;PCA module-1 mode register
CCAP1L    EQU    0EBH      ;PCA module-1 capture register LOW
CCAP1H    EQU    0FBH      ;PCA module-1 capture register HIGH

PCA_LED   BIT    P1.0      ;PCA test LED
CNT       EQU    20H

;-----
        ORG    0000H
        LJMP   MAIN

        ORG    003BH
        LJMP   PCA_ISR

;-----
        ORG    0100H
MAIN:
        MOV    SP,    #3FH      ;Initial stack point
        MOV    A,    P_SW1
        ANL   A,    #0CFH      //CCP_S0=0 CCP_S1=0
        MOV   P_SW1, A        //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1)
//        MOV   A,    P_SW1
//        ANL   A,    #0CFH      //CCP_S0=1 CCP_S1=0
//        ORL   A,    #CCP_S0    //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2)
//        MOV   P_SW1, A
//
//        MOV   A,    P_SW1
//        ANL   A,    #0CFH      //CCP_S0=0 CCP_S1=1
//        ORL   A,    #CCP_S1    //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3)
//        MOV   P_SW1, A

        MOV    CCON, #0        ;Initial PCA control register
                                ;PCA timer stop running
                                ;Clear CF flag
                                ;Clear all module interrupt flag

```

---



---

```

    CLR    A                                ;
    MOV    CL,    A                          ;Reset PCA base timer
    MOV    CH,    A                          ;
    MOV    CMOD, #00H                       ;Set PCA timer clock source as Fosc/12
                                           ;Disable PCA timer overflow interrupt
;-----
    MOV    CCAP0L,    #LOW T100Hz          ;
    MOV    CCAP0H,    #HIGH T100Hz        ;Initial PCA module-0
    MOV    CCAPM0,    #49H                 ;PCA module-0 work in 16-bit timer mode
                                           ;and enable PCA interrupt
;-----
    SETB   CR                                ;PCA timer start run
    SETB   EA
    MOV    CNT,    #100

    SJMP   $
;-----
PCA_ISR:
    PUSH   PSW
    PUSH   ACC
    CLR    CCF0                             ;Clear interrupt flag
    MOV    A,    CCAP0L
    ADD   A,    #LOW T100Hz                 ;Update compare value
    MOV   CCAP0L,A
    MOV   A,    CCAP0H
    ADDC  A,    #HIGH T100Hz
    MOV   CCAP0H,    A
    DJNZ  CNT,   PCA_ISR_EXIT               ;count 100 times
    MOV   CNT,   #100
    CPL   PCA_LED                           ;Flash once per second
    POP   ACC
    POP   PSW
    RETI
;-----
    END

```

---

---

## 11.6 Demo Program using CCP/PCA to output High Speed Pulse

There are two programs using CCP/PCA to output high speed pulse, one written in C language and the other in assembly language.

### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using CCP/PCA to output 16-bit High Speed Pulse -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

#define FOSC 18432000L
#define T100KHz (FOSC / 4 / 100000)

typedef unsigned char BYTE;
typedef unsigned int WORD;
sfr P_SW1 = 0xA2; //Peripheral function switch register 1

#define CCP_S0 0x10 //P_SW1.4
#define CCP_S1 0x20 //P_SW1.5

/*Declare SFR associated with the PCA */
sfr CCON = 0xD8; //PCA control register
sbit CCF0 = CCON^0; //PCA module-0 interrupt flag
sbit CCF1 = CCON^1; //PCA module-1 interrupt flag

sbit CR = CCON^6; //PCA timer run control bit
sbit CF = CCON^7; //PCA timer overflow flag
sfr CMOD = 0xD9; //PCA mode register
sfr CL = 0xE9; //PCA base timer LOW
sfr CH = 0xF9; //PCA base timer HIGH
sfr CCAPM0 = 0xDA; //PCA module-0 mode register
sfr CCAP0L = 0xEA; //PCA module-0 capture register LOW
sfr CCAP0H = 0xFA; //PCA module-0 capture register HIGH
```

---

```

sfr    CCAPM1    = 0xDB;           //PCA module-1 mode register
sfr    CCAP1L    = 0xEB;           //PCA module-1 capture register LOW
sfr    CCAP1H    = 0xFB;           //PCA module-1 capture register HIGH

sfr    PCAPWM0   = 0xf2;
sfr    PCAPWM1   = 0xf3;

sbit   PCA_LED   = P1^0;           //PCA test LED

```

```

BYTE   cnt;
WORD   value;

```

```

void PCA_isr() interrupt 7 using 1

```

```

{
    CCF0 = 0;           //Clear interrupt flag
    CCAP0L = value;
    CCAP0H = value >> 8; //Update compare value
    value += T100KHz;
}

```

```

void main()

```

```

{
    ACC    =    P_SW1;
    ACC    &=    ~(CCP_S0 | CCP_S1); //CCP_S0=0 CCP_S1=0
    P_SW1  =    ACC; // (P1.2/ECI, P1.1/CCP0, P1.0/CCP1 )

// ACC    =    P_SW1;
// ACC    &=    ~(CCP_S0 | CCP_S1); //CCP_S0=1 CCP_S1=0
// ACC    |=    CCP_S0; // (P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2 )
// P_SW1  =    ACC;
//
// ACC    =    P_SW1;
// ACC    &=    ~(CCP_S0 | CCP_S1); //CCP_S0=0 CCP_S1=1
// ACC    |=    CCP_S1; // (P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3 )
// P_SW1  =    ACC;
    CCON = 0; //Initial PCA control register
                //PCA timer stop running
                //Clear CF flag
                //Clear all module interrupt flag

    CL = 0; //Reset PCA base timer
    CH = 0;

    CMOD = 0x02; //Set PCA timer clock source as Fosc/2
                //Disable PCA timer overflow interrupt

```

---

```

value = T100KHz;
CCAP0L = value;           //P1.3 output 100KHz square wave
CCAP0H = value >> 8;     //Initial PCA module-0
value += T100KHz;
CCAPM0 = 0x4d;           //PCA module-0 work in 16-bit timer mode
                           //and enable PCA interrupt, toggle the output pin CCP0(P1.3)

CR = 1;                  //PCA timer start run
EA = 1;
cnt = 0;

while (1);
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using CCP/PCA to output 16-bit High Speed Pulse -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

T100KHz      EQU    2EH                ;(18432000 / 4 / 100000)
P_SW1        EQU    0A2H              ;Peripheral function switch register1

CCP_S0        EQU    10H                ;P_SW1.4
CCP_S1        EQU    20H                ;P_SW1.5

/*Declare SFR associated with the PCA */
CCON          EQU    0D8H              ;PCA control register
CCF0          BIT    CCON.0            ;PCA module-0 interrupt flag
CCF1          BIT    CCON.1            ;PCA module-1 interrupt flag

```

---

CR	BIT	CCON.6	:PCA timer run control bit
CF	BIT	CCON.7	:PCA timer overflow flag
CMOD	EQU	0D9H	:PCA mode register
CL	EQU	0E9H	:PCA base timer LOW
CH	EQU	0F9H	:PCA base timer HIGH
CCAPM0	EQU	0DAH	:PCA module-0 mode register
CCAP0L	EQU	0EAH	:PCA module-0 capture register LOW
CCAP0H	EQU	0FAH	:PCA module-0 capture register HIGH
CCAPM1	EQU	0DBH	:PCA module-1 mode register
CCAP1L	EQU	0EBH	:PCA module-1 capture register LOW
CCAP1H	EQU	0FBH	:PCA module-1 capture register HIGH

-----

```

ORG 0000H
LJMP MAIN

ORG 003BH
PCA_ISR:
PUSH PSW
PUSH ACC
CLR CCF0 ;Clear interrupt flag
MOV A, CCAP0L
ADD A, #T100KHz ;Update compare value
MOV CCAP0L, A
CLR A
ADDC A, CCAP0H
MOV CCAP0H, A

```

PCA\_ISR\_EXIT:

```

POP ACC
POP PSW
RETI

```

-----

```

ORG 0100H
MOV A, P_SW1
ANL A, #0CFH ;//CCP_S0=0 CCP_S1=0
MOV P_SW1, A ;//(P1.2/ECI, P1.1/CCP0, P1.0/CCP1 )
// MOV A, P_SW1
// ANL A, #0CFH ;//CCP_S0=1 CCP_S1=0
// ORL A, #CCP_S0 ;//(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2 )
// MOV P_SW1, A

```

---

```

//
//  MOV  A,      P_SW1
//  ANL  A,      #0CFH           //CCP_S0=0 CCP_S1=1
//  ORL  A,      #CCP_S1       //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3 )
//  MOV  P_SW1,  A

      MOV  CCON,  #0           ;Initial PCA control register
                                ;PCA timer stop running
                                ;Clear CF flag
                                ;Clear all module interrupt flag

      CLR  A                   ;
      MOV  CL,   A           ;Reset PCA base timer
      MOV  CH,   A           ;

      MOV  CMOD, #02H        ;Set PCA timer clock source as Fosc/2
                                ;Disable PCA timer overflow interrupt
;-----
      MOV  CCAP0L, #T100KHz   ;P1.3 output 100KHz square wave
      MOV  CCAP0H, #0         ;Initial PCA module-0
      MOV  CCAPM0, #4dH      ;PCA module-0 work in 16-bit timer mode and enable
                                ;PCA interrupt, toggle the output pin CEX0(P1.3)
;-----
      SETB CR                 ;PCA timer start run
      SETB EA

      SJMP $
;-----
      END

```

---

## 11.7 Demo Program for CCP/PCA Outputting PWM (6+7+8 bit)

### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using CCP/PCA to output PWM (6-bit / 7-bit / 8-bit) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

#define FOSC 18432000L

typedef unsigned char BYTE;
typedef unsigned int WORD;

sfr P_SW1 = 0xA2; //Peripheral function switch register 1

#define CCP_S0 0x10 //P_SW1.4
#define CCP_S1 0x20 //P_SW1.5

/*Declare SFR associated with the PCA */
sfr CCON = 0xD8; //PCA control register
sbit CCF0 = CCON^0; //PCA module-0 interrupt flag
sbit CCF1 = CCON^1; //PCA module-1 interrupt flag

sbit CR = CCON^6; //PCA timer run control bit
sbit CF = CCON^7; //PCA timer overflow flag
sfr CMOD = 0xD9; //PCA mode register
sfr CL = 0xE9; //PCA base timer LOW
sfr CH = 0xF9; //PCA base timer HIGH
sfr CCAPM0 = 0xDA; //PCA module-0 mode register
sfr CCAP0L = 0xEA; //PCA module-0 capture register LOW
sfr CCAP0H = 0xFA; //PCA module-0 capture register HIGH
```

---

```

sfr    CCAPM1    =    0xDB;    //PCA module-1 mode register
sfr    CCAP1L    =    0xEB;    //PCA module-1 capture register LOW
sfr    CCAP1H    =    0xFB;    //PCA module-1 capture register HIGH
sfr    PCAPWM0    =    0xF2;
sfr    PCAPWM1    =    0xF3;

void main()
{
    ACC    =    P_SW1;
    ACC    &=    ~(CCP_S0 | CCP_S1);    //CCP_S0=0 CCP_S1=0
    P_SW1    =    ACC;    //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1 )

//    ACC    =    P_SW1;
//    ACC    &=    ~(CCP_S0 | CCP_S1);    //CCP_S0=1 CCP_S1=0
//    ACC    |=    CCP_S0;    //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2 )
//    P_SW1    =    ACC;
//
//    ACC    =    P_SW1;
//    ACC    &=    ~(CCP_S0 | CCP_S1);    //CCP_S0=0 CCP_S1=1
//    ACC    |=    CCP_S1;    //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3 )
//    P_SW1    =    ACC;

    CCON    =    0;    //Initial PCA control register
                //PCA timer stop running
                //Clear CF flag
                //Clear all module interrupt flag
                //Reset PCA base timer

    CL = 0;
    CH    =    0;
    CMOD = 0x02;    //Set PCA timer clock source as Fosc/2
                //Disable PCA timer overflow interrupt

    PCA_PWM0    =    0x00;    //PCA module 0 work in 8-bit PWM
    CCAP0H    =    CCAP0L    =    0x20;    //PWM0 port output 87.5% ((100H-20H)/100H)
                //duty cycle square wave

    CCAPM0    =    0x42;    //PCA module 0 work in 8-bit PWM
                //and no PCA interrupt

    PCA_PWM1 = 0x40;    //PCA module 1 work in 7-bit PWM
    CCAP1H = CCAP1L = 0x20;    //PWM1 port output 75% ((80H-20H)/80H)
                //duty cycle square wave

    CCAPM1 = 0x42;    //PCA module 1 work in 7-bit PWM
                //and no PCA interrupt

    CR = 1;    //PCA timer start run

    while (1);
}

```

---



---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using CCP/PCA to output PWM (6-bit / 7-bit / 8-bit) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

P_SW1      EQU    0A2H                ;Peripheral function switch register1

CCP_S0     EQU    10H                ;P_SW1.4
CCP_S1     EQU    20H                ;P_SW1.5

;/*Declare SFR associated with the PCA */
CCON       EQU    0D8H                ;PCA control register
CCF0       BIT    CCON.0              ;PCA module-0 interrupt flag
CCF1       BIT    CCON.1              ;PCA module-1 interrupt flag
CR         BIT    CCON.6              ;PCA timer run control bit
CF         BIT    CCON.7              ;PCA timer overflow flag
CMOD       EQU    0D9H                ;PCA mode register
CL         EQU    0E9H                ;PCA base timer LOW
CH         EQU    0F9H                ;PCA base timer HIGH
CCAPM0     EQU    0DAH                ;PCA module-0 mode register
CCAP0L     EQU    0EAH                ;PCA module-0 capture register LOW
CCAP0H     EQU    0FAH                ;PCA module-0 capture register HIGH
CCAPM1     EQU    0DBH                ;PCA module-1 mode register
CCAP1L     EQU    0EBH                ;PCA module-1 capture register LOW
CCAP1H     EQU    0FBH                ;PCA module-1 capture register HIGH
PCA_PWM0   EQU    0F2H
PCA_PWM1   EQU    0F3H

;-----
      ORG    0000H
      LJMP  MAIN

;-----
      ORG    0100H
```

---

MAIN:

```
MOV    A,      P_SW1
ANL    A,      #0CFH      //CCP_S0=0 CCP_S1=0
MOV    P_SW1, A          //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1, )

// MOV    A,      P_SW1
// ANL    A,      #0CFH      //CCP_S0=1 CCP_S1=0
// ORL    A,      #CCP_S0    //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2 )
// MOV    P_SW1, A
//
// MOV    A,      P_SW1
// ANL    A,      #0CFH      //CCP_S0=0 CCP_S1=1
// ORL    A,      #CCP_S1    //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3 )
// MOV    P_SW1, A

MOV    CCON, #0          ;Initial PCA control register
                          ;PCA timer stop running
                          ;Clear CF flag
                          ;Clear all module interrupt flag

CLR    A                ;Reset PCA base timer
MOV    CL,   A           ;
MOV    CH,   A           ;
MOV    CMOD, #02H       ;Set PCA timer clock source as Fosc/2
                          ;Disable PCA timer overflow interrupt

;-----
MOV    PCA_PWM0, #00H    ;PCA module 0 work in 8-bit PWM
MOV    A,      #020H    ;
MOV    CCAP0H, A        ;PWM0 port output 87.5% ((100H-20H)/100H)
                          ;duty cycle square wave

MOV    CCAP0L, A        ;
MOV    CCAPM0, #42H     ;PCA module-0 work in 8-bit PWM mode
                          ;and no PCA interrupt

;-----
MOV    PCA_PWM1, #40H    ;PCA module 1 work in 7-bit PWM
MOV    A,      #020H    ;
MOV    CCAP1H, A        ;PWM1 port output 75% ((80H-20H)/80H)
                          ;duty cycle square wave

MOV    CCAP1L, A        ;
MOV    CCAPM1, #42H     ;PCA module-1 work in 7-bit PWM mode
                          ;and no PCA interrupt

;-----
SETB   CR              ;PCA timer start run
SJMP  $

;-----
END
```

---

## 11.8 Program achieving 9~16 bit PWM Output by CCP/PCA

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program that utilize CCP/PCA to achieve 9~16 bit PWM by software plus hardware -*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

/***** the description of functions *****/
realize n-bit PWM by PCA
*****/

#include "PWMn.h"

//=====
// file: PWMn_main.c
// function: test PWM
// edition: VER1.0
// data: 2011-4-11
//
//=====

/***** function declaration *****/

Simulate 9~16bit PWM by PCA
Fosc=24.576MHZ. output 6000HZ 12bit PWM

*****/

/***** construction document
PWMn.c
PWMn_main.c
*****/

/***** declare local constant *****/
/***** declare local variable *****/

unsigned int pwm;
/***** declare local Function *****/
void delay_ms(unsigned char ms);
/***** declare external Function and external variable *****/
extern unsigned int PWM_high;
```

---

```

void PWMn_SetHighReg(unsigned int high);
void PWMn_init(unsigned int high);

/***** main function *****/
//=====
// function: void main(void)
// description: keep on updating the value of PWM
// parameter: none
// return: none
// edition: VER1.0
// data: 2011-4-11
//=====
void main(void)
{
    pwm = 1000;                //pwm initial value
    pwm = PWM_HIGH_MIN;       //pwm initial value
    PWMn_init(pwm);           //Initialize pwm

    while (1)
    {
        delay_ms(10);         //delay
        pwm += 10;
        if(pwm >= PWM_HIGH_MAX)    pwm = PWM_HIGH_MIN;
        PWMn_SetHighReg(pwm);     //update PWM duty cycle
    }
}

/*****
//=====
// function: void delay_ms(unsigned char ms)
// description: delay function
// parameter: ms
// return: none
// edition: VER1.0
// data: 2011-4-11
//=====
void delay_ms(unsigned char ms)
{
    unsigned int i;
    do
    {
        i = MAIN_Fosc / 14000L;    //1T
        while(--i);
    }while(--ms);
}

```

---

---

```

unsigned int    PWM_high;    //define PWM duty cycle
unsigned int    PWM_low;
unsigned int    CCAPO_tmp;

//=====================================================
// function: void PWMn_SetHighReg(unsigned int high)
// description: write the duty ratio data
// parameter: high: duty ratio data
// return: none
// edition: VER1.0
// data: 2009-12-30
//=====================================================

void PWMn_SetHighReg(unsigned int high)
{
    if(high > PWM_HIGH_MAX)
        high = PWM_HIGH_MAX;
    if(high < PWM_HIGH_MIN)
        high = PWM_HIGH_MIN;
    CR = 0;                //disable PCA
    PWM_high = high;
    PWM_low = PWM_DUTY - high;
    CR = 1;                //run PCA
}

//=====================================================
// function: void PWMn_init(unsigned int high)
// description: initialize
// parameter: high: initialize the duty ratio data
// return: none
// edition: VER1.0
// data: 2009-12-30
//=====================================================

void PWMn_init(unsigned int high)
{
    #ifdef    STC15W4K32S4
        P3M1 &= ~0x80, P3M0 |= 0x80;        //CCAPO in PUSH-PULL output mode

    #else
        P1M1 &= ~0x08, P1M0 |= 0x08;        //CCAPO in PUSH-PULL output mode

    #endif
    CCON = 0;                //clear CF CR CCF0 CCF1
}

```

---

---

```

    IPH |= 0x80;                                     //PCA interrupt in the highest priority
    PPCA = 1;

    CMOD = (PCA_IDLE_DISABLE << 7) | (PCA_SOURCE_SELECT << 1);
    CCAPM0 = 0x4D;                                   //high-speed output mode,enable interrupt(ECCF0=1)
    CL = 0;                                          //clear PCA registers
    CH = 0;
    CCAPO_tmp = 0;
    PWMn_SetHighReg(high);                           //initialize duty ratio data
    CR = 1;                                          //run PCA
    EA = 1;                                          //enable global interrupt
}

```

```

//=====
// Function: void PCA_interrupt (void) interrupt 7
// description: PCA interrupt service routine
// parameter: none
// return: none
// edition: VER1.0
// data: 2009-12-30
//=====

```

```

void PCA_interrupt (void) interrupt 7
{
    if(CCF0 == 1)                                    //PCA module 0 interrupt
    {
        CCF0 = 0;                                    //clear PCA module 0 interrupt flag

        if(CCP0 == 1)    CCAPO_tmp += PWM_high;
        else    CCAPO_tmp += PWM_low;

        CCAPOL = (unsigned char)CCAP0_tmp;
        CCAPOH = (unsigned char)(CCAP0_tmp >> 8);
    }
/*
    else if(CCF1 == 1)                                //PCA module 1 interrupt
    {
        CCF1 = 0;                                    //Clear PCA module 1 interrupt flag
    }
    else if(CF == 1)                                  //PCA overflow interrupt
    {
        CF = 0;                                       //clear PCA overflow interrupt flag
    }
*/
}

```

---

## 11.9 Demo Program of CCP/PCA 16-bit Capture Mode

There are two programs utilizing CCP/PCA 16-bit capture mode to measure pulse width, one written in C language and the other in assembly language.

### 1.C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program utilizing 16-bit capture mode of CCP/PCA to measure pulse width -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

#define FOSC 18432000L

typedef unsigned char BYTE;
typedef unsigned int WORD;
typedef unsigned long DWORD;

sfr P_SW1 = 0xA2; //Peripheral function switch register 1

#define CCP_S0 0x10 //P_SW1.4
#define CCP_S1 0x20 //P_SW1.5

/*Declare SFR associated with the PCA */
sfr CCON = 0xD8; //PCA control register
sbit CCF0 = CCON^0; //PCA module-0 interrupt flag
sbit CCF1 = CCON^1; //PCA module-1 interrupt flag

sbit CR = CCON^6; //PCA timer run control bit
sbit CF = CCON^7; //PCA timer overflow flag
sfr CMOD = 0xD9; //PCA mode register
sfr CL = 0xE9; //PCA base timer LOW
sfr CH = 0xF9; //PCA base timer HIGH
sfr CCAPM0 = 0xDA; //PCA module-0 mode register
sfr CCAP0L = 0xEA; //PCA module-0 capture register LOW
sfr CCAP0H = 0xFA; //PCA module-0 capture register HIGH
```

---

```

sfr    CCAPM1    = 0xDB;           //PCA module-1 mode register
sfr    CCAP1L    = 0xEB;           //PCA module-1 capture register LOW
sfr    CCAP1H    = 0xFB;           //PCA module-1 capture register HIGH

```

```

sfr    PCAPWM0   = 0xf2;
sfr    PCAPWM1   = 0xf3;

```

```

BYTE   cnt;
DWORD  count0;
DWORD  count1;
DWORD  length;

```

```

void main()
{
    ACC    =    P_SW1;
    ACC    &=    ~(CCP_S0 | CCP_S1);    //CCP_S0=0 CCP_S1=0
    P_SW1  =    ACC;                    //(P1.2/ECI, P1.1/CCP0, P1.0/CCP1 )

//    ACC    =    P_SW1;
//    ACC    &=    ~(CCP_S0 | CCP_S1);    //CCP_S0=1 CCP_S1=0
//    ACC    |=    CCP_S0;                //(P3.4/ECI_2, P3.5/CCP0_2, P3.6/CCP1_2 )
//    P_SW1  =    ACC;
//
//    ACC    =    P_SW1;
//    ACC    &=    ~(CCP_S0 | CCP_S1);    //CCP_S0=0 CCP_S1=1
//    ACC    |=    CCP_S1;                //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3 )
//    P_SW1  =    ACC;

    CCON = 0;                            //Initial PCA control register
                                           //PCA timer stop running
                                           //Clear CF flag
                                           //Clear all module interrupt flag

    CL = 0;                               //Reset PCA base timer

    CH    =    0;
    CCAP0L = 0;
    CCAP0H = 0;
    CMOD  =    0x09;                       //Set SYSclk as the PCA clock source,
                                           //and enable PCA overflow interrupt

    CCAPM0=    0x21;                       //PCA module 0 work in 16-bit capture mode
                                           //(capture triggered by the rising edge on CCPn/PCAn pin)
                                           // and enable capture interrupt

```



---

```

//      CCAPM0=      0x11; //PCAModule 0 work in 16-bit capture mode
//                          //(capture triggered by the falling edge on CCPn/PCAn pin)
//                          // and enable capture interrupt

//      CCAPM0=      0x31; //PCAModule 0 work in 16-bit capture mode
//                          //(capture triggered by the transition on CCPn/PCAn pin)
//                          // and enable capture interrupt

CR      =      1;          //PCA timer start run

EA      =      1;

cnt      =      0;
count0   =      0;
count1   =      0;

while (1);
}

void PCA_isr() interrupt 7 using 1
{
    if (CF)
    {
        CF = 0;
        cnt++;                //PCA overflow times +1
    }
    if (CCF0)
    {
        CCF0 = 0;
        count0 = count1;
        ((BYTE *)&count1)[3] = CCAP0L;
        ((BYTE *)&count1)[2] = CCAP0H;
        ((BYTE *)&count1)[1] = cnt;
        ((BYTE *)&count1)[0] = 0;
        length = count1 - count0;
    }
}

```

---

## 2. Assembler Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program utilizing 16-bit capture mode of CCP/PCA to measure pulse width -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

P_SW1      EQU    0A2H                ;Peripheral function switch register1

CCP_S0     EQU    10H                ;P_SW1.4
CCP_S1     EQU    20H                ;P_SW1.5

;/*Declare SFR associated with the PCA */
CCON       EQU    0D8H                ;PCA control register
CCF0       BIT    CCON.0              ;PCA module-0 interrupt flag
CCF1       BIT    CCON.1              ;PCA module-1 interrupt flag
CR         BIT    CCON.6              ;PCA timer run control bit
CF         BIT    CCON.7              ;PCA timer overflow flag
CMOD       EQU    0D9H                ;PCA mode register
CL         EQU    0E9H                ;PCA base timer LOW
CH         EQU    0F9H                ;PCA base timer HIGH
CCAPM0     EQU    0DAH                ;PCA module-0 mode register
CCAP0L     EQU    0EAH                ;PCA module-0 capture register LOW
CCAP0H     EQU    0FAH                ;PCA module-0 capture register HIGH
CCAPM1     EQU    0DBH                ;PCA module-1 mode register
CCAP1L     EQU    0EBH                ;PCA module-1 capture register LOW
CCAP1H     EQU    0FBH                ;PCA module-1 capture register HIGH
PCA_PWM0   EQU    0F2H
PCA_PWM1   EQU    0F3H

CNT        EQU    30H
COUNT0    EQU    31H
COUNT1    EQU    34H
LENGTH     EQU    37H
```

```

;-----
        ORG    0000H
        LJMP   MAIN
        ORG    003BH
PCA_ISR:
        PUSH  PSW
        PUSH  ACC
        JNB   CF,    CKECK_CCF0
        CLR   CF
        INC   CNT
CKECK_CCF0:
        JNB   CCF0,  PCA_ISR_EXIT
        CLR   CCF0
        MOV   COUNT0,    COUNT1
        MOV   COUNT0+1,  COUNT1+1
        MOV   COUNT0+2,  COUNT1+2
        MOV   COUNT1,    CNT
        MOV   COUNT1+1,  CCAP0H
        MOV   COUNT1+2,  CCAP0L
        CLR   C
        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
PCA_ISR_EXIT:
        POP   ACC
        POP   PSW
        RETI

;-----

        ORG    0100H
MAIN:
        MOV   SP,    #5FH

        MOV   A,    P_SW1
        ANL   A,    #0CFH           //CCP_S0=0 CCP_S1=0
        MOV   P_SW1, A             //(P1.2/ECl, P1.1/CCP0, P1.0/CCP1 )

//      MOV   A,    P_SW1
//      ANL   A,    #0CFH           //CCP_S0=1 CCP_S1=0
//      ORL   A,    #CCP_S0        //(P3.4/ECl_2, P3.5/CCP0_2, P3.6/CCP1_2 )
//      MOV   P_SW1, A
//

```

---

```

//      MOV    A,      P_SW1
//      ANL    A,      #0CFH                //CCP_S0=0 CCP_S1=1
//      ORL    A,      #CCP_S1             //(P2.4/ECI_3, P2.5/CCP0_3, P2.6/CCP1_3 )
//      MOV    P_SW1, A

      MOV    CCON, #0                      ;Initial PCA control register
                                           ;PCA timer stop running
                                           ;Clear CF flag
                                           ;Clear all module interrupt flag

      CLR    A                            ;
      MOV    CL,  A                        ;Reset PCA base timer
      MOV    CH,  A                        ;
      MOV    CCAP0L, A
      MOV    CCAP0H, A
      MOV    CMOD, #09H                   ;Set SYSclk as the PCA clock source,
                                           ;and enable PCA overflow interrupt

      MOV    CCAPM0, #21H                 ;PCA module 0 work in 16-bit capture mode
                                           ;(capture triggered by the rising edge on CCPn/PCAn pin)
                                           ; and enable capture interrupt

      MOV    CCAPM0, #11H                 ;PCA module 0 work in 16-bit capture mode
                                           ;(capture triggered by the falling edge on CCPn/PCAn pin)
                                           ; and enable capture interrupt

      MOV    CCAPM0, #31H                 ;PCA module 0 work in 16-bit capture mode
                                           ;(capture triggered by the transition on CCPn/PCAn pin)
                                           ;and enable capture interrupt

      SETB   CR                            ;PCA timer start run
      SETB   EA

      CLR    A                            ;Initialize variables
      MOV    CNT,  A
      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

      SJMP   $
;-----
      END

```

---

---

## 11.10 Demo Program using T0 to Simulate 10 or 16 bits PWM ——T0 as 16-bit Auto-Reload Timer/Counter

### 1. C Program Listing

```
/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using 16-bit auto-reload timer/counter to simulate 10 or 16 bits PWM ---*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

#define PWM6BIT      64          //6-bit PWM periodicity
#define PWM8BIT      256        //8-bit PWM periodicity
#define PWM10BIT     1024       //10-bit PWM periodicity
#define PWM16BIT     65536      //16-bit PWM periodicity

#define HIGHDUTY     64          // high duty (duty ratio 64/256=25%)
#define LOWDUTY      (PWM8BIT-HIGHDUTY) //low duty

sfr  AUXR          = 0x8e;      //Auxiliary register
sfr  INT_CLKO      = 0x8f;      //Clock Output register
sbit TOCLKO        = P3^5;      //T0 Clock Output

bit  flag;

// Timer 0 interrupt service routine
void tm0() interrupt 1
{
    flag = !flag;
    if (flag)
    {
        TL0 = (65536-HIGHDUTY);
        TH0 = (65536-HIGHDUTY) >> 8;
    }
    else
    {
        TL0 = (65536-LOWDUTY);
        TH0 = (65536-LOWDUTY) >> 8;
    }
}
}
```

---

```

void main()
{
    AUXR = 0x80;           //T0 in 1T mode
    INT_CLKO = 0x01;      //enable the function of Timer 0 Clock Output
    TMOD &= 0xf0;         //T0 in mode 0(16-bit auto-reload timer/counter)
    TL0 = (65536-LOWDUTY); //initialize the reload value
    TH0 = (65536-LOWDUTY) >> 8;
    T0CLKO = 1;           //initialize the pin of clock output (soft PWM port)
    flag = 0;
    TR0 = 1;              //run Timer 0
    ET0 = 1;              //enable Timer 0 interrupt
    EA = 1;
    while (1);
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- STC MCU Limited. -----*/
/* --- Exam Program using 16-bit auto-reload timer/counter to simulate 10 or 16 bits PWM ---*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

;PWM6BIT EQU 64 ;6-bit PWM periodicity
;PWM8BIT EQU 256 ;8-bit PWM periodicity
;PWM10BIT EQU 1024 ;10-bit PWM periodicity
;PWM16BIT EQU 65536 ;16-bit PWM periodicity

HIGHDUTY EQU 64 ;high duty (duty ratio 64/256=25%)
LOWDUTY EQU (PWM8BIT-HIGHDUTY) ;low duty

AUXR DATA 08EH ;Auxiliary register
INT_CLKO DATA 08FH ;Clock Output register
T0CLKO BIT P3.5 ;T0 Clock Output

FLAG BIT 20H.0

;-----

```

---

```

    ORG    0000H
    LJMP   MAIN

    ORG    000BH
    LJMP   TM0_ISR

;-----

MAIN:
    MOV    AUXR, #80H           ;T0 in 1T mode
    MOV    INT_CLKO, #01H      ;enable the function of Timer 0 clock output
    ANL    TMOD, #0F0H        ;T0 in mode 0(16-bit auto-reload timer/counter)
    MOV    TL0, #LOW (65536-LOWDUTY) ;initialize the reload value
    MOV    TH0, #HIGH (65536-LOWDUTY)
    SETB   T0CLKO             ;initialize the pin of clock output (soft PWM port)
    CLR    FLAG
    SETB   TR0                 ;run Timer 0
    SETB   ET0                 ;enable Timer 0 interrupt
    SETB   EA

    SJMP   $

;-----
;Timer 0 interrupt service routine
TM0_ISR:
    CPL    FLAG
    JNB    FLAG, READYLOW
READYHIGH:
    MOV    TL0, #LOW (65536-HIGHDUTY)
    MOV    TH0, #HIGH (65536-HIGHDUTY)
    JMP    TM0ISR_EXIT
READYLOW:
    MOV    TL0, #LOW (65536-LOWDUTY)
    MOV    TH0, #HIGH (65536-LOWDUTY)
TM0ISR_EXIT:
    RETI

;-----

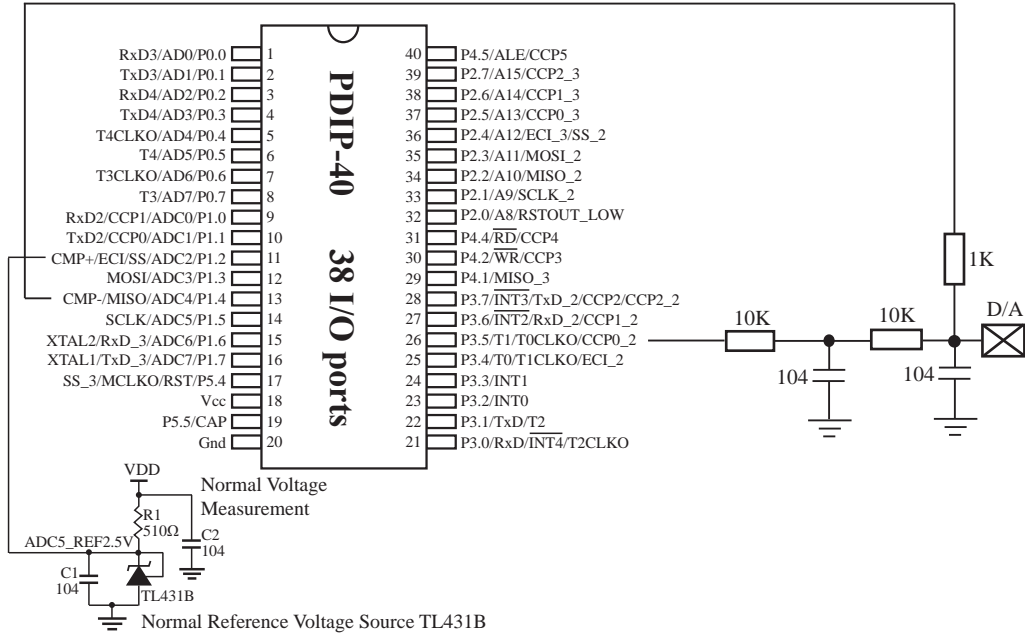
    END

```

---

# 11.11 Circuit Diagram using CCP/PCA to achieve 8~16 bit DAC

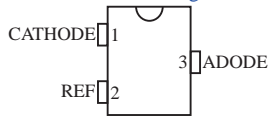
CCP is abbreviation for Capture, Compare, PWM



Note:

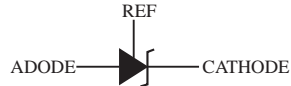
- 1 the higher the PWM frequency is, the smoother the output wave is.
- 2 Suppose the operating voltage is 5V and 1V need to be output, if high level is set to 1/5 and low level to 4/5, PWM output voltage would be 1V.

Normal Reference Voltage Source TL431B



Package: SOT23-3; Price: RMB 0.15~0.3

The symbol of normal reference voltage source TL431B





# Chapter 12 New 6 Channels of PWM of STC15W4K series

## ——High-Precision PWM with Death Time Control

There are a group of Pulse Width Modulation generators (six channels independently) intergated in STC-15W4K32S4 series MCU, each of one owns two counter T1 and T2 to control the level to change. Besides, a 15-bits counter also is availabe for all PWM generators.

The six channels of PWM also can monitor the external exception cases such as unusal level of P2.4 port or abnormal comparing result of comparator so as to emergency shutdown the PWM output.

The output ports related with the new six channels of PWM of STC15W4K32S4 series MCU are defined as below

[PWM2:P3.7, PWM3:P2.1, PWM4:P2.2, PWM5:P2.3, PWM6:P1.6, PWM7:P1.7]

Each of PWM output ports can be switched to the second group of pins by setting the SFRs bit CnPINSEL:

[PWM2\_2:P2.7, PWM3\_2:P4.5, PWM4\_2:P4.4, PWM5\_2:P4.2, PWM6\_2:P0.7, PWM7\_2:P0.6]

SFRs about port modes

Symbol	Description	Address	Bit Address and Symbol								Value after Power-on or Reset
			B7	B6	B5	B4	B3	B2	B1	B0	
P1M1	P1 configuration 1	91H									0000,0000
P1M0	P1 configuration 0	92H									0000,0000
P0M1	P0 configuration 1	93H									0000,0000
P0M0	P0 configuration 0	94H									0000,0000
P2M1	P2 configuration 1	95H									0000,0000
P2M0	P2 configuration 0	96H									0000,0000
P3M1	P3 configuration 1	B1H									0000,0000
P3M0	P3 configuration 0	B2H									0000,0000
P4M1	P4 configuration 1	B3H									0000,0000
P4M0	P4 configuration 0	B4H									0000,0000

Configue the modes of I/O ports

PxM1	PxM0	I/O ports Mode
0	0	quasi_bidirectional (traditional 8051 I/O port output
0	1	push-pull output(strong pull-up output)
1	0	input-only (high-impedance )
1	1	Open Drain

The output ports related with the new six channels of PWM must be set as quasi\_bidirectional or push-pull output(strong pull-up output) mode. Only then can the PWM output ports be used correctly.

For example, set all ports as quasi\_bidirectional mode in assembly code as bellow:

```

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

```

## 12.1 Special Function Registers of New PWM Generators

Symbol	Description	Add.	Bit Address and Symbol								Value after Power-on or Reset
			B7	B6	B5	B4	B3	B2	B1	B0	
P_SW2	Peripheral Function Switch register 2	BAH	EAXSFR	DBLPWR	P31PU	P30PU	-	S4_S	S3_S	S2_S	0000,0000
PWMCFG	PWM Configure register	F1H	-	CBTADC	C7INI	C6INI	C5INI	C4INI	C3INI	C2INI	0000,0000
PWMCRCR	PWM Control register	F5H	ENPWM	ECBI	ENC7O	ENC6O	ENC5O	ENC4O	ENC3O	ENC2O	0000,0000
PWMIF	PWM Interrupt Flag register	F6H	-	CBIF	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	x000,0000
PWMFDCR	PWM F_ception Dection Control Register	F7H	-	-	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	xx00,0000
PWMCH	PWM Counter High	FFF0H	-	PWMCH[14:8]							x000,0000
PWMCL	PWM Counter low	FFF1H	PWMCL[7:0]							0000,0000	
PWMCKS	PWM Clock Selection register	FFF2H	-	-	-	SELT2	PS[3:0]			xxx0,0000	
PWM2T1H	Timer 1 of PWM2 High	FF00H	-	PWM2T1H[14:8]							x000,0000
PWM2T1L	Timer 1 of PWM2 Low	FF01H	PWM2T1L[7:0]							0000,0000	
PWM2T2H	Timer 2 of PWM2 High	FF02H	-	PWM2T2H[14:8]							x000,0000
PWM2T2L	Timer 2 of PWM2 Low	FF03H	PWM2T2L[7:0]							0000,0000	
PWM2CR	PWM2 Control register	FF04H	-	-	-	-	PWM2_PS	EPWM2I	EC2T2SI	EC2T1SI	xxxx,0000
PWM3T1H	Timer 1 of PWM3 High	FF10H	-	PWM3T1H[14:8]							x000,0000
PWM3T1L	Timer 1 of PWM3 Low	FF11H	PWM3T1L[7:0]							0000,0000	

Symbol	Description	Add.	Bit Address and Symbol								Value after Power-on or Reset
			B7	B6	B5	B4	B3	B2	B1	B0	
PWM3T2H	Timer 2 of PWM3 High	FF12H	-	PWM3T2H[14:8]							x000,0000
PWM3T2L	Timer 2 of PWM3 Low	FF13H	PWM3T2L[7:0]								0000,0000
PWM3CR	PWM3 Control register	FF14H	-	-	-	-	PWM3_PS	EPWM3I	EC3T2SI	EC3T1SI	xxxx,0000
PWM4T1H	Timer 1 of PWM4 High	FF20H	-	PWM4T1H[14:8]							x000,0000
PWM4T1L	Timer 1 of PWM4 Low	FF21H	PWM4T1L[7:0]								0000,0000
PWM4T2H	Timer 2 of PWM4 High	FF22H	-	PWM4T2H[14:8]							x000,0000
PWM4T2L	Timer 2 of PWM4 Low	FF23H	PWM4T2L[7:0]								0000,0000
PWM4CR	PWM4 Control register	FF24H	-	-	-	-	PWM4_PS	EPWM4I	EC4T2SI	EC4T1SI	xxxx,0000
PWM5T1H	Timer 1 of PWM5 High	FF30H	-	PWM5T1H[14:8]							x000,0000
PWM5T1L	Timer 1 of PWM5 Low	FF31H	PWM5T1L[7:0]								0000,0000
PWM5T2H	Timer 2 of PWM5 High	FF32H	-	PWM5T2H[14:8]							x000,0000
PWM5T2L	Timer 2 of PWM5 Low	FF33H	PWM5T2L[7:0]								0000,0000
PWM5CR	PWM5 Control register	FF34H	-	-	-	-	PWM5_PS	EPWM5I	EC5T2SI	EC5T1SI	xxxx,0000
PWM6T1H	Timer 1 of PWM6 High	FF40H	-	PWM6T1H[14:8]							x000,0000
PWM6T1L	Timer 1 of PWM6 Low	FF41H	PWM6T1L[7:0]								0000,0000
PWM6T2H	Timer 2 of PWM6 High	FF42H	-	PWM6T2H[14:8]							x000,0000
PWM6T2L	Timer 2 of PWM6 Low	FF43H	PWM6T2L[7:0]								0000,0000
PWM6CR	PWM6 Control register	FF44H	-	-	-	-	PWM6_PS	EPWM6I	EC6T2SI	EC6T1SI	xxxx,0000
PWM7T1H	Timer 1 of PWM7 High	FF50H	-	PWM7T1H[14:8]							x000,0000
PWM7T1L	Timer 1 of PWM7 Low	FF51H	PWM7T1L[7:0]								0000,0000
PWM7T2H	Timer 2 of PWM7 High	FF52H	-	PWM7T2H[14:8]							x000,0000
PWM7T2L	Timer 2 of PWM7 Low	FF53H	PWM7T2L[7:0]								0000,0000
PWM7CR	PWM7 Control register	FF54H	-	-	-	-	PWM7_PS	EPWM7I	EC7T2SI	EC7T1SI	xxxx,0000

## 1. Peripheral Function Switch register 2: P\_SW2

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
P_SW2	BAH	name	EAXSFR	DBLPWR	P31PU	P30PU	-	S4_S	S3_S	S2_S	0000,0000B

EAXSFR Enable the Extended SFRs

0 MOVX A,@DPTR/MOVX @DPTR,A will access the extended RAM XRAM

1 MOVX A,@DPTR/MOVX @DPTR,A will access the extended SFR XSFR

Attention if needed to use the SFRs in the extended RAM, this bit EAXSFR must be enabled.

## 2. PWM Configure registe : PWMCFG

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMCFG	F1H	name	-	CBTADC	C7INI	C6INI	C5INI	C4INI	C3INI	C2INI	0000,0000

CBTADC if trigger A/D Convert when the counter of PWM return to zero CBIF==1 or not

0 do not trigger A/D Convert when the counter of PWM return to zero

1 trigger A/D Convert when the counter of PWM return to zero CBIF==1 It would be happen only has ENPWM==1 and ADCON==1 been set before

C7INI Set the initial level of PWM7 output ports

0 the initial level of PWM7 output ports is low level

1 the initial level of PWM7 output ports is high level

C6INI Set the initial level of PWM6 output ports

0 the initial level of PWM6 output ports is low level

1 the initial level of PWM6 output ports is high level

C5INI Set the initial level of PWM5 output ports

0 the initial level of PWM5 output ports is low level

1 the initial level of PWM5 output ports is high level

C4INI Set the initial level of PWM4 output ports

0 the initial level of PWM4 output ports is low level

1 the initial level of PWM4 output ports is high level

C3INI Set the initial level of PWM3 output ports

0 the initial level of PWM3 output ports is low level

1 the initial level of PWM3 output ports is high level

C2INI Set the initial level of PWM2 output ports

0 the initial level of PWM2 output ports is low level

1 the initial level of PWM2 output ports is high level

### 3. PWM Control register: PWMCR

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMCR	F5H	name	ENPWM	ECBI	ENC70	ENC60	ENC50	ENC40	ENC30	ENC20	0000,0000B

ENPWM if enable the new enhanced PWM generators or not

0 dsable the new enhanced PWM generators

1 enable the new enhanced PWM generators

ECBI if enable the PWM interrupt as the PWM counter return to zero or not

0 disable the PWM interrupt as the PWM counter return to zero

1 enable the PWM interrupt as the PWM counter return to zero

ENC70 set the ports of PWM7

0 the ports of PWM7 are just as GPIO

1 the ports of PWM7 are the output ports of PWM7 which would be controlled by PWM generator

ENC60 set the ports of PWM6

0 the ports of PWM6 are just as GPIO

1 the ports of PWM6 are the output ports of PWM6 which would be controlled by PWM generator

ENC50 set the ports of PWM5

0 the ports of PWM5 are just as GPIO

1 the ports of PWM5 are the output ports of PWM5 which would be controlled by PWM generator

ENC40 set the ports of PWM4

0 the ports of PWM4 are just as GPIO

1 the ports of PWM4 are the output ports of PWM4 which would be controlled by PWM generator

ENC30 set the ports of PWM3

0 the ports of PWM3 are just as GPIO

1 the ports of PWM3 are the output ports of PWM3 which would be controlled by PWM generator

ENC20 set the ports of PWM2

0 the ports of PWM2 are just as GPIO

1 the ports of PWM2 are the output ports of PWM2 which would be controlled by PWM generator

### 4. PWM Interrupt Flag register: PWMIF

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMIF	F6H	name	-	CBIF	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	x000,0000B

CBIF The flag bit of PWM interrupt happened as the PWM counter return to zero

The bit will be set to 1 by hardware as the PWM counter return to zero.If ECBI==1 the corresponding interrupt routine would be run. This bit may be cleared by software.

C7IF The flag bit of PWM7 interrupt

This bit will be set to 1 by hardware as the PWM has turned. If EPWM7I==1 the corresponding interrupt routine would be run. This bit may be cleared by software.

- 
- C6IF The flag bit of PWM6 interrupt  
This bit will be set to 1 by hardware as the PWM has turned. If  $EPWM6I==1$  the corresponding interrupt routine would be run. This bit may be cleared by software.
- C5IF The flag bit of PWM5 interrupt  
This bit will be set to 1 by hardware as the PWM has turned. If  $EPWM5I==1$  the corresponding interrupt routine would be run. This bit may be cleared by software.
- C4IF The flag bit of PWM4 interrupt  
This bit will be set to 1 by hardware as the PWM has turned. If  $EPWM4I==1$  the corresponding interrupt routine would be run. This bit may be cleared by software.
- C3IF The flag bit of PWM3 interrupt  
This bit will be set to 1 by hardware as the PWM has turned. If  $EPWM3I==1$  the corresponding interrupt routine would be run. This bit may be cleared by software.
- C2IF The flag bit of PWM2 interrupt  
This bit will be set to 1 by hardware as the PWM has turned. If  $EPWM2I==1$  the corresponding interrupt routine would be run. This bit may be cleared by software.

### 5. PWM F\_ception Detection Control Register: PWMFDCR

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMFDCR	F7H	name	-	-	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	xx00,0000B

ENFD if enable the function of PWM f\_ception detection or not

- 0 disable the function of PWM f\_ception detection
- 1 enable the function of PWM f\_ception detection

FLTFLIO set the mode of PWM output ports as the external exception cases happened

- 0 the mode of PWM output ports will stay the same as the external exception cases happened
- 1 the mode of PWM output ports will be set as input-only (high-impedance mode as the external exception cases happened)

EFDI if enable the PWM f\_ception detection interrupt or not

- 0 disable the PWM f\_ception detection interrupt
- 1 enable the PWM f\_ception detection interrupt

FDCMP Set the comparator output as the external exception source

- 0 the comparator is irrelevant to PWM
- 1 the external exception case will happen as the level of P5.5/CMP+ is higher than the one of P5.4/CMP- or internal bandGap voltage 1.28V

FDIO Set the level of P2.4 output as the external exception source

- 0 P2.4 is irrelevant to PWM
- 1 the external exception case will happen as the level of P2.4 is high

FDIF The flag bit of the PWM f\_ception detection interrupt

This bit will be set to 1 by hardware as the PWM external exception case has happened. If  $EFDI==1$  the corresponding interrupt routine would be run. This bit may be cleared by software.

## 6. PWM2 Control register: PWM2CR

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM2CR	FF04H (XSFR)	name	-	-	-	-	PWM2_PS	EPWM2I	EC2T2SI	EC2T1SI	xxxx,0000B

PWM2\_PS select the output ports of PWM2 on where

- 0 the output ports of PWM2 on P3.7
- 1 the output ports of PWM2 on P2.7

EPWM2I if enable the PWM2 interrupt or not

- 0 disable the PWM2 interrupt
- 1 enable the PWM2 interrupt. If **C2IF==1** the corresponding interrupt routine would be run.

EC2T2SI make the PWM2 interrupt to occur when the T2 of PWM2 has turned

- 0 make the PWM2 interrupt to do nothing with the T2 of PWM2
- 1 make the PWM2 interrupt to occur when the T2 of PWM2 has turned. And the T2 of PWM2 would turn and the bit C2IF would be set as 1 as the internal counting value of PWM2 equals the preset value of T2. And then if **EPWM2I==1** the corresponding interrupt routine would be run.

EC2T1SI make the PWM2 interrupt to occur when the T1 of PWM2 has turned

- 0 make the PWM2 interrupt to do nothing with the T1 of PWM2
- 1 make the PWM2 interrupt to occur when the T1 of PWM2 has turned. And the T1 of PWM2 would turn and the bit C2IF would be set as 1 as the internal counting value of PWM2 equals the preset value of T1. And then if **EPWM2I==1** the corresponding interrupt routine would be run.

## 7. PWM3 Control register: PWM3CR

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM3CR	FF14H (XSFR)	name	-	-	-	-	PWM3_PS	EPWM3I	EC3T2SI	EC3T1SI	xxxx,0000B

PWM3\_PS select the output ports of PWM3 on where

- 0 the output ports of PWM3 on P2.1
- 1 the output ports of PWM3 on P4.5

EPWM3I if enable the PWM3 interrupt or not

- 0 disable the PWM3 interrupt
- 1 enable the PWM3 interrupt. If **C3IF==1** the corresponding interrupt routine would be run.

EC3T2SI make the PWM3 interrupt to occur when the T2 of PWM3 has turned

- 0 make the PWM3 interrupt to do nothing with the T2 of PWM3
- 1 make the PWM3 interrupt to occur when the T2 of PWM3 has turned. And the T2 of PWM3 would turn and the bit C3IF would be set as 1 as the internal counting value of PWM3 equals the preset value of T2. And then if **EPWM3I==1** the corresponding interrupt routine would be run.

EC3T1SI make the PWM3 interrupt to occur when the T1 of PWM3 has turned

- 0 make the PWM3 interrupt to do nothing with the T1 of PWM3
- 1 make the PWM3 interrupt to occur when the T1 of PWM3 has turned. And the T1 of PWM3 would turn and the bit C3IF would be set as 1 as the internal counting value of PWM3 equals the preset value of T1. And then if **EPWM3I==1** the corresponding interrupt routine would be run.

## 8. PWM4 Control register: PWM4CR

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM4CR	FF24H (XSFR)	name	-	-	-	-	PWM4_PS	EPWM4I	EC4T2SI	EC4T1SI	xxxx,0000B

PWM4\_PS select the output ports of PWM4 on where  
 0 the output ports of PWM4 on P2.2  
 1 the output ports of PWM4 on P4.4

EPWM4I if enable the PWM4 interrupt or not  
 0 disable the PWM4 interrupt  
 1 enable the PWM4 interrupt. If **C4IF==1** the corresponding interrupt routine would be run.

EC4T2SI make the PWM4 interrupt to occur when the T2 of PWM4 has turned  
 0 make the PWM4 interrupt to do nothing with the T2 of PWM4  
 1 make the PWM4 interrupt to occur when the T2 of PWM4 has turned. And the T2 of PWM4 would turn and the bit C4IF would be set as 1 as the internal counting value of PWM4 equals the preset value of T2. And then if **EPWM4I==1** the corresponding interrupt routine would be run.

EC4T1SI make the PWM4 interrupt to occur when the T1 of PWM4 has turned  
 0 make the PWM4 interrupt to do nothing with the T1 of PWM4  
 1 make the PWM4 interrupt to occur when the T1 of PWM4 has turned. And the T1 of PWM4 would turn and the bit C4IF would be set as 1 as the internal counting value of PWM4 equals the preset value of T1. And then if **EPWM4I==1** the corresponding interrupt routine would be run.

## 9. PWM5 Control register: PWM5CR

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM5CR	FF34H (XSFR)	name	-	-	-	-	PWM5_PS	EPWM5I	EC5T2SI	EC5T1SI	xxxx,0000B

PWM5\_PS select the output ports of PWM5 on where  
 0 the output ports of PWM5 on P2.3  
 1 the output ports of PWM5 on P4.2

EPWM5I if enable the PWM5 interrupt or not  
 0 disable the PWM5 interrupt  
 1 enable the PWM5 interrupt. If **C5IF==1** the corresponding interrupt routine would be run.

EC5T2SI make the PWM5 interrupt to occur when the T2 of PWM5 has turned  
 0 make the PWM5 interrupt to do nothing with the T2 of PWM5  
 1 make the PWM5 interrupt to occur when the T2 of PWM5 has turned. And the T2 of PWM5 would turn and the bit C5IF would be set as 1 as the internal counting value of PWM5 equals the preset value of T2. And then if **EPWM5I==1** the corresponding interrupt routine would be run.

EC5T1SI make the PWM5 interrupt to occur when the T1 of PWM5 has turned  
 0 make the PWM5 interrupt to do nothing with the T1 of PWM5  
 1 make the PWM5 interrupt to occur when the T1 of PWM5 has turned. And the T1 of PWM5 would turn and the bit C5IF would be set as 1 as the internal counting value of PWM5 equals the preset value of T1. And then if **EPWM5I==1** the corresponding interrupt routine would be run.



## 10. PWM6 Control register: PWM6CR

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM6CR	FF44H (XSFR)	name	-	-	-	-	PWM6_PS	EPWM6I	EC6T2SI	EC6T1SI	xxxx,0000B

PWM6\_PS select the output ports of PWM6 on where

- 0 the output ports of PWM6 on P1.6
- 1 the output ports of PWM6 on P0.7

EPWM6I if enable the PWM6 interrupt or not

- 0 disable the PWM6 interrupt
- 1 enable the PWM6 interrupt. If **C6IF==1** the corresponding interrupt routine would be run.

EC6T2SI make the PWM6 interrupt to occur when the T2 of PWM6 has turned

- 0 make the PWM6 interrupt to do nothing with the T2 of PWM6
- 1 make the PWM6 interrupt to occur when the T2 of PWM6 has turned. And the T2 of PWM6 would turn and the bit C6IF would be set as 1 as the internal counting value of PWM6 equals the preset value of T2. And then if **EPWM6I==1** the corresponding interrupt routine would be run.

EC6T1SI make the PWM6 interrupt to occur when the T1 of PWM6 has turned

- 0 make the PWM6 interrupt to do nothing with the T1 of PWM6
- 1 make the PWM6 interrupt to occur when the T1 of PWM6 has turned. And the T1 of PWM6 would turn and the bit C6IF would be set as 1 as the internal counting value of PWM6 equals the preset value of T1. And then if **EPWM6I==1** the corresponding interrupt routine would be run.

## 11. PWM7 Control register: PWM7CR

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM7CR	FF54H (XSFR)	name	-	-	-	-	PWM7_PS	EPWM7I	EC7T2SI	EC7T1SI	xxxx,0000B

PWM7\_PS select the output ports of PWM7 on where

- 0 the output ports of PWM7 on P1.7
- 1 the output ports of PWM7 on P0.6

EPWM7I if enable the PWM7 interrupt or not

- 0 disable the PWM7 interrupt
- 1 enable the PWM7 interrupt. If **C7IF==1** the corresponding interrupt routine would be run.

EC7T2SI make the PWM7 interrupt to occur when the T2 of PWM7 has turned

- 0 make the PWM7 interrupt to do nothing with the T2 of PWM7
- 1 make the PWM7 interrupt to occur when the T2 of PWM7 has turned. And the T2 of PWM7 would turn and the bit C7IF would be set as 1 as the internal counting value of PWM7 equals the preset value of T2. And then if **EPWM7I==1** the corresponding interrupt routine would be run.

EC7T1SI make the PWM7 interrupt to occur when the T1 of PWM7 has turned

- 0 make the PWM7 interrupt to do nothing with the T1 of PWM7
- 1 make the PWM7 interrupt to occur when the T1 of PWM7 has turned. And the T1 of PWM7 would turn and the bit C7IF would be set as 1 as the internal counting value of PWM7 equals the preset value of T1. And then if **EPWM7I==1** the corresponding interrupt routine would be run.

## 12.2 Interrupts of New Enhanced PWM Generators

SFRs related with the interrupts of new enhanced PWM

Symbol	Description	Add.	Bit Address and Symbol								Value after Power-on or Reset
			B7	B6	B5	B4	B3	B2	B1	B0	
IP2	2rd Interrupt Priority register	B5H	-	-	-	PX4	PPWMFD	PPWM	PSPI	PS2	xxx0,0000
PWMCR	PWM Control register	F5H	ENPWM	ECBI	ENC70	ENC60	ENC50	ENC40	ENC30	ENC20	0000,0000
PWMIF	PWM Interrupt Flag register	F6H	-	CBIF	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	x000,0000
PWMFDCR	PWM F_ception Detection Control Register	F7H	-	-	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	xx00,0000
PWM2CR	PWM2 Control register	FF04H	-	-	-	-	PWM2_PS	EPWM2I	EC2T2SI	EC2T1SI	xxxx,0000
PWM3CR	PWM3 Control register	FF14H	-	-	-	-	PWM3_PS	EPWM3I	EC3T2SI	EC3T1SI	xxxx,0000
PWM4CR	PWM4 Control register	FF24H	-	-	-	-	PWM4_PS	EPWM4I	EC4T2SI	EC4T1SI	xxxx,0000
PWM5CR	PWM5 Control register	FF34H	-	-	-	-	PWM5_PS	EPWM5I	EC5T2SI	EC5T1SI	xxxx,0000
PWM6CR	PWM6 Control register	FF44H	-	-	-	-	PWM6_PS	EPWM6I	EC6T2SI	EC6T1SI	xxxx,0000
PWM7CR	PWM7 Control register	FF54H	-	-	-	-	PWM7_PS	EPWM7I	EC7T2SI	EC7T1SI	xxxx,0000

### 1. PWM Interrupt Priority register: IP2

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
IP2	B5H	name	-	-	-	PX4	PPWMFD	PPWM	PSPI	PS2	0000,0000B

PPWMFD: PWM f\_ception detection interrupt priority control bit.

if PPWMFD=0, PWM f\_ception detection interrupt is assigned lowest priority (priority 0).

if PPWMFD=1, PWM f\_ception detection interrupt is assigned highest priority (priority 1).

PPWM: PWM interrupt priority control bit.

if PPWM=0, PWM interrupt is assigned lowest priority (priority 0).

if PPWM=1, PWM interrupt is assigned highest priority (priority 1).

### 2. PWM Control register: PWMCR

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMCR	F5H	name	ENPWM	ECBI	ENC70	ENC60	ENC50	ENC40	ENC30	ENC20	0000,0000B

ECBI if enable the PWM interrupt as the PWM counter return to zero or not

0 disable the PWM interrupt as the PWM counter return to zero

1 enable the PWM interrupt as the PWM counter return to zero

### 3. PWM Interrupt Flag register: PWMIF

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMIF	F6H	name	-	CBIF	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	x000,0000B

- CBIF** The flag bit of PWM interrupt happened as the PWM counter return to zero  
The bit will be set to 1 by hardware as the PWM counter return to zero.If **ECBI==1** the corresponding interrupt routine would be run. This bit may be cleared by software.
- C7IF** The flag bit of PWM7 interrupt  
This bit will be set to 1 by hardware as the PWM has turned. If **EPWM7I==1** the corresponding interrupt routine would be run. This bit may be cleared by software.
- C6IF** The flag bit of PWM6 interrupt  
This bit will be set to 1 by hardware as the PWM has turned.If **EPWM6I==1** the corresponding interrupt routine would be run. This bit may be cleared by software.
- C5IF** The flag bit of PWM5 interrupt  
This bit will be set to 1 by hardware as the PWM has turned.If **EPWM5I==1** the corresponding interrupt routine would be run. This bit may be cleared by software.
- C4IF** The flag bit of PWM4 interrupt  
This bit will be set to 1 by hardware as the PWM has turned. If **EPWM4I==1** the corresponding interrupt routine would be run. This bit may be cleared by software.
- C3IF** The flag bit of PWM3 interrupt  
This bit will be set to 1 by hardware as the PWM has turned.If **EPWM3I==1** the corresponding interrupt routine would be run. This bit may be cleared by software.
- C2IF** The flag bit of PWM2 interrupt  
This bit will be set to 1 by hardware as the PWM has turned.If **EPWM2I==1** the corresponding interrupt routine would be run. This bit may be cleared by software.

### 4. PWM F\_ception Detection Control Register: PWMFDCR

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWMFDCR	F7H	name	-	-	ENFD	FLTFLIO	<b>EFDI</b>	FDCMP	FDIO	<b>FDIF</b>	xx00,0000B

- EFDI** if enable the PWM f\_ception detection interrupt or not  
0 disable the PWM f\_ception detection interrupt  
1 enable the PWM f\_ception detection interrupt

- FDIF** The flag bit of the PWM f\_ception detection interrupt  
This bit will be set to 1 by hardware as the PWM external exception case has happened. If **EFDI==1** the corresponding interrupt routine would be run. This bit may be cleared by software.

## 5. PWM2 Control register: PWM2CR

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM2CR	FF04H (XSFR)	name	-	-	-	-	PWM2_PS	EPWM2I	EC2T2SI	EC2T1SI	xxxx,0000B

EPWM2I if enable the PWM2 interrupt or not

0 disable the PWM2 interrupt

1 enable the PWM2 interrupt. If **C2IF==1** the corresponding interrupt routine would be run.

EC2T2SI make the PWM2 interrupt to occur when the T2 of PWM2 has turned

0 make the PWM2 interrupt to do nothing with the T2 of PWM2

1 make the PWM2 interrupt to occur when the T2 of PWM2 has turned. And the T2 of PWM2 would turn and the bit C2IF would be set as 1 as the internal counting value of PWM2 equals the preset value of T2. And then if **EPWM2I==1** the corresponding interrupt routine would be run.

EC2T1SI make the PWM2 interrupt to occur when the T1 of PWM2 has turned

0 make the PWM2 interrupt to do nothing with the T1 of PWM2

1 make the PWM2 interrupt to occur when the T1 of PWM2 has turned. And the T1 of PWM2 would turn and the bit C2IF would be set as 1 as the internal counting value of PWM2 equals the preset value of T1. And then if **EPWM2I==1** the corresponding interrupt routine would be run.

## 6. PWM3 Control register: PWM3CR

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM3CR	FF14H (XSFR)	name	-	-	-	-	PWM3_PS	EPWM3I	EC3T2SI	EC3T1SI	xxxx,0000B

EPWM3I if enable the PWM3 interrupt or not

0 disable the PWM3 interrupt

1 enable the PWM3 interrupt. If **C3IF==1** the corresponding interrupt routine would be run.

EC3T2SI make the PWM3 interrupt to occur when the T2 of PWM3 has turned

0 make the PWM3 interrupt to do nothing with the T2 of PWM3

1 make the PWM3 interrupt to occur when the T2 of PWM3 has turned. And the T2 of PWM3 would turn and the bit C3IF would be set as 1 as the internal counting value of PWM3 equals the preset value of T2. And then if **EPWM3I==1** the corresponding interrupt routine would be run.

EC3T1SI make the PWM3 interrupt to occur when the T1 of PWM3 has turned

0 make the PWM3 interrupt to do nothing with the T1 of PWM3

1 make the PWM3 interrupt to occur when the T1 of PWM3 has turned. And the T1 of PWM3 would turn and the bit C3IF would be set as 1 as the internal counting value of PWM3 equals the preset value of T1. And then if **EPWM3I==1** the corresponding interrupt routine would be run.

## 7. PWM4 Control register: PWM4CR

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM4CR	FF24H (XSFR)	name	-	-	-	-	PWM4_PS	EPWM4I	EC4T2SI	EC4T1SI	xxxx,0000B

EPWM4I if enable the PWM4 interrupt or not

0 disable the PWM4 interrupt

1 enable the PWM4 interrupt. If **C4IF==1** the corresponding interrupt routine would be run.

EC4T2SI make the PWM4 interrupt to occur when the T2 of PWM4 has turned

0 make the PWM4 interrupt to do nothing with the T2 of PWM4

1 make the PWM4 interrupt to occur when the T2 of PWM4 has turned. And the T2 of PWM4 would turn and the bit C4IF would be set as 1 as the internal counting value of PWM4 equals the preset value of T2. And then if **EPWM4I==1** the corresponding interrupt routine would be run.

EC4T1SI make the PWM4 interrupt to occur when the T1 of PWM4 has turned

0 make the PWM4 interrupt to do nothing with the T1 of PWM4

1 make the PWM4 interrupt to occur when the T1 of PWM4 has turned. And the T1 of PWM4 would turn and the bit C4IF would be set as 1 as the internal counting value of PWM4 equals the preset value of T1. And then if **EPWM4I==1** the corresponding interrupt routine would be run.

## 7. PWM5 Control register: PWM5CR

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM5CR	FF34H (XSFR)	name	-	-	-	-	PWM5_PS	EPWM5I	EC5T2SI	EC5T1SI	xxxx,0000B

EPWM5I if enable the PWM5 interrupt or not

0 disable the PWM5 interrupt

1 enable the PWM5 interrupt. If **C5IF==1** the corresponding interrupt routine would be run.

EC5T2SI make the PWM5 interrupt to occur when the T2 of PWM5 has turned

0 make the PWM5 interrupt to do nothing with the T2 of PWM5

1 make the PWM5 interrupt to occur when the T2 of PWM5 has turned. And the T2 of PWM5 would turn and the bit C5IF would be set as 1 as the internal counting value of PWM5 equals the preset value of T2. And then if **EPWM5I==1** the corresponding interrupt routine would be run.

EC5T1SI make the PWM5 interrupt to occur when the T1 of PWM5 has turned

0 make the PWM5 interrupt to do nothing with the T1 of PWM5

1 make the PWM5 interrupt to occur when the T1 of PWM5 has turned. And the T1 of PWM5 would turn and the bit C5IF would be set as 1 as the internal counting value of PWM5 equals the preset value of T1. And then if **EPWM5I==1** the corresponding interrupt routine would be run.

## 9. PWM6 Control register: PWM6CR

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM6CR	FF44H (XSFR)	name	-	-	-	-	PWM6_PS	EPWM6I	EC6T2SI	EC6T1SI	xxxx,0000B

EPWM6I if enable the PWM6 interrupt or not

0 disable the PWM6 interrupt

1 enable the PWM6 interrupt. If C6IF==1 the corresponding interrupt routine would be run.

EC6T2SI make the PWM6 interrupt to occur when the T2 of PWM6 has turned

0 make the PWM6 interrupt to do nothing with the T2 of PWM6

1 make the PWM6 interrupt to occur when the T2 of PWM6 has turned. And the T2 of PWM6 would turn and the bit C6IF would be set as 1 as the internal counting value of PWM6 equals the preset value of T2. And then if EPWM6I==1 the corresponding interrupt routine would be run.

EC6T1SI make the PWM6 interrupt to occur when the T1 of PWM6 has turned

0 make the PWM6 interrupt to do nothing with the T1 of PWM6

1 make the PWM6 interrupt to occur when the T1 of PWM6 has turned. And the T1 of PWM6 would turn and the bit C6IF would be set as 1 as the internal counting value of PWM6 equals the preset value of T1. And then if EPWM6I==1 the corresponding interrupt routine would be run.

## 10. PWM7 Control register: PWM7CR

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0	Reset Value
PWM7CR	FF54H (XSFR)	name	-	-	-	-	PWM7_PS	EPWM7I	EC7T2SI	EC7T1SI	xxxx,0000B

EPWM7I if enable the PWM7 interrupt or not

0 disable the PWM7 interrupt

1 enable the PWM7 interrupt. If C7IF==1 the corresponding interrupt routine would be run.

EC7T2SI make the PWM7 interrupt to occur when the T2 of PWM7 has turned

0 make the PWM7 interrupt to do nothing with the T2 of PWM7

1 make the PWM7 interrupt to occur when the T2 of PWM7 has turned. And the T2 of PWM7 would turn and the bit C7IF would be set as 1 as the internal counting value of PWM7 equals the preset value of T2. And then if EPWM7I==1 the corresponding interrupt routine would be run.

EC7T1SI make the PWM7 interrupt to occur when the T1 of PWM7 has turned

0 make the PWM7 interrupt to do nothing with the T1 of PWM7

1 make the PWM7 interrupt to occur when the T1 of PWM7 has turned. And the T1 of PWM7 would turn and the bit C7IF would be set as 1 as the internal counting value of PWM7 equals the preset value of T1. And then if EPWM7I==1 the corresponding interrupt routine would be run.

Interrupt Sources and Vector address Table

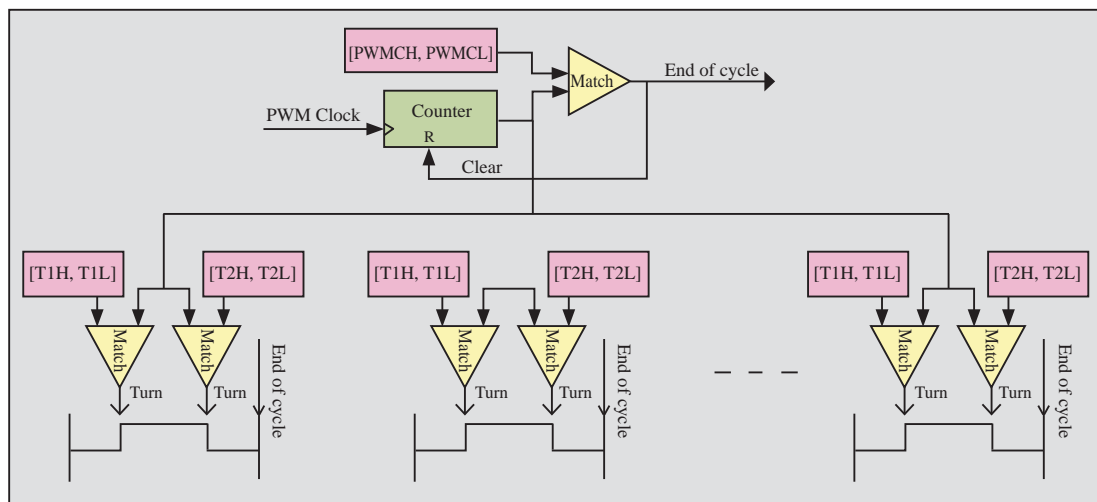
Interrupt Sources	Interrupt Vector address	Interrupt Priority setting (IP2)	Interrupt Request	Interrupt Enable Control Bit	How to clear the interrupt request bit
PWM interrupt	00B3H (22)	PPWM	CBIF	ENPWM/ECBI/EA	Cleared by software
			C2IF	ENPWM / EPWM2I / EC2T2SI    EC2T1SI / EA	Cleared by software
			C3IF	ENPWM / EPWM3I / EC3T2SI    EC3T1SI / EA	Cleared by software
			C4IF	ENPWM / EPWM4I / EC4T2SI    EC4T1SI / EA	Cleared by software
			C5IF	ENPWM / EPWM5I / EC5T2SI    EC5T1SI / EA	Cleared by software
			C6IF	ENPWM / EPWM6I / EC6T2SI    EC6T1SI / EA	Cleared by software
			C7IF	ENPWM / EPWM7I / EC7T2SI    EC7T1SI / EA	Cleared by software
PWM f_ception detection interrupt	00BBH (23)	PPWMFD	FDIF	ENPWM / ENFD / EFDI / EA	Cleared by software

The declaration of PWM interrupt functions are shown below in C language program

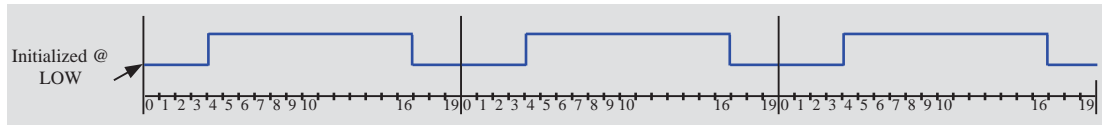
```
void PWM_Routine(void) interrupt 22;
```

```
void PWMFD_Routine(void) interrupt 23;
```

Structure of PWM generators



The following code is to generate a repetitive waveform shown in following figure :



```

Period == (16+1)   Channel-4: toggle-point-1==3(0003H)   toggle-point-2==16(0010H)

;; +-----+
;; | Global Configuration |
;; +-----+

; Set EAXSFR to enable xSFR writing against XRAM writing
MOV  A,    P_SW2
ORL  A,    #10000000B
MOV  P_SW2, A
;

; Set channel-4 output register start at LOW
MOV  A,    PWMCFG
ANL  A,    #11111011B           ; channel-4 start at LOW
MOV  PWMCFG, A
;

; Set a clock of the waveform generator consists of 4 Fosc
MOV  DPTR, #PWMCKS           ; FFF2H
MOV  A,    #00000011B
MOVX @DPTR, A
;

; Set period as 20
; {PWMCH,PWMCL} <= 19
MOV  DPTR, #PWMCH           ; FFF0H
MOV  A,    #00H             ; PWMCH should be changed first
MOVX @DPTR, A
MOV  DPTR, #PWMCL           ; FFF1H
MOV  A,    #13H            ; Write PWMCL simultaneous update PWMCH
MOVX @DPTR, A
;

;; +-----+
;; | Channel-4 Configuration |
;; +-----+

; Set toggle point 1 of Channel-4 as 3
MOV  DPTR, #PWM4T1H         ; FF20H

```



---

```

MOV  A,    #00H
MOVX @DPTR, A
;
MOV  DPTR, #PWM4T1L          ; FF21H
MOV  A,    #03H
MOVX @DPTR, A
;

; Set toggle point 2 of Channel-4 as 16
MOV  DPTR, #PWM4T2H          ; FF22H
MOV  A,    #00H
MOVX @DPTR, A
;
MOV  DPTR, #PWM4T2L          ; FF23H
MOV  A,    #10H
MOVX @DPTR, A
;

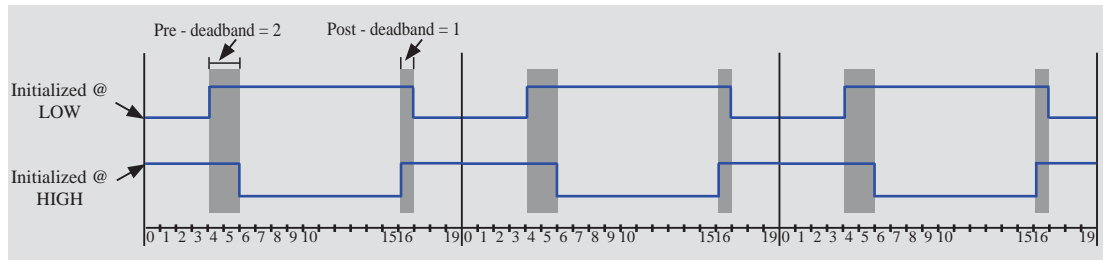
; Set Channel-4 output pin as default, and disable interrupting
MOV  DPTR, #PWM4CR           ; FF24H
MOV  A,    #00H
MOVX @DPTR, A
;

; Clear EAXSFR to disable xSFR, return MOVX-DPTR to normal XRAM access
MOV  A,    P_SW2
ANL  A,    #01111111B
MOV  P_SW2, A
;

;; +-----+
;; | Operate PWM output   |
;; +-----+
; Enable counter counting, and enable Channel-4 output
MOV  A,    PWMCR
ORL  A,    #10000100B
MOV  PWMCR, A
;

```

The following code is to generate a waveform shown in following figure :



Period == (16+1)    Channel-4: toggle-point-1==3(0003H)    toggle-point-2==16(0010H)  
 Channel-5: toggle-point-1==5(0005H)    toggle-point-2==15(000FH)

```
;; +-----+
;; | Global Configuration |
;; +-----+
```

```
;;;
;;; Set EAXSFR to enable xSFR writing against XRAM writing
;;;
```

```
MOV A, P_SW2
ORL A, #1000000B
MOV P_SW2, A
;
```

```
; Set channel-4 output register start at LOW, channel-5 at HIGH
```

```
MOV A, PWMCFG
ANL A, #11111011B                   ; channel-4 start at LOW
ORL A, #00001000B                   ; channel-5 start at HIGH
MOV PWMCFG, A
;
```

```
; Set a clock of the waveform generator consists of 4 Fosc
```

```
MOV DPTR, #PWMCKS                   ; FFF2H
MOV A, #00000011B
MOVX @DPTR, A
;
```

```
; Set period as 20
```

```
; {PWMCH,PWMCL} <= 19
```

```
MOV DPTR, #PWMCH                   ; FFF0H
MOV A, #00H                         ; PWMCH should be changed first
MOVX @DPTR, A
MOV DPTR, #PWMCL                   ; FFF1H
MOV A, #13H                         ; Write PWMCL simultaneous update PWMCH
```

---

```

MOVX  @DPTR, A
;

;; +-----+
;; | Channel-4 Configuration |
;; +-----+

; Set toggle point 1 of Channel-4 as 3
MOV   DPTR, #PWM4T1H           ; FF20H
MOV   A,    #00H
MOVX  @DPTR, A
;
MOV   DPTR, #PWM4T1L           ; FF21H
MOV   A,    #03H
MOVX  @DPTR, A
;
; Set toggle point 2 of Channel-4 as 16
MOV   DPTR, #PWM4T2H           ; FF22H
MOV   A,    #00H
MOVX  @DPTR, A
;
MOV   DPTR, #PWM4T2L           ; FF23H
MOV   A,    #10H
MOVX  @DPTR, A
;

; Set Channel-4 output pin as default, and disable interrupting
MOV   DPTR, #PWM4CR            ; FF24H
MOV   A,    #00H
MOVX  @DPTR, A
;

;; +-----+
;; | Channel-5 Configuration |
;; +-----+

; Set toggle point 1 of Channel-5 as 5
MOV   DPTR, #PWM5T1H           ; FF30H
MOV   A,    #00H
MOVX  @DPTR, A
;
MOV   DPTR, #PWM5T1L           ; FF31H
MOV   A,    #03H
MOVX  @DPTR, A

```

---

---

```

;
; Set toggle point 3 of Channel-5 as 15
    MOV    DPTR, #PWM5T2H           ; FF32H
    MOV    A,    #00H
    MOVX   @DPTR, A
;
    MOV    DPTR, #PWM5T2L           ; FF33H
    MOV    A,    #0FH
    MOVX   @DPTR, A
;

; Set Channel-5 output pin as default, and disable interrupting
    MOV    DPTR, #PWM5CR            ; FF34H
    MOV    A,    #00H
    MOVX   @DPTR, A
;

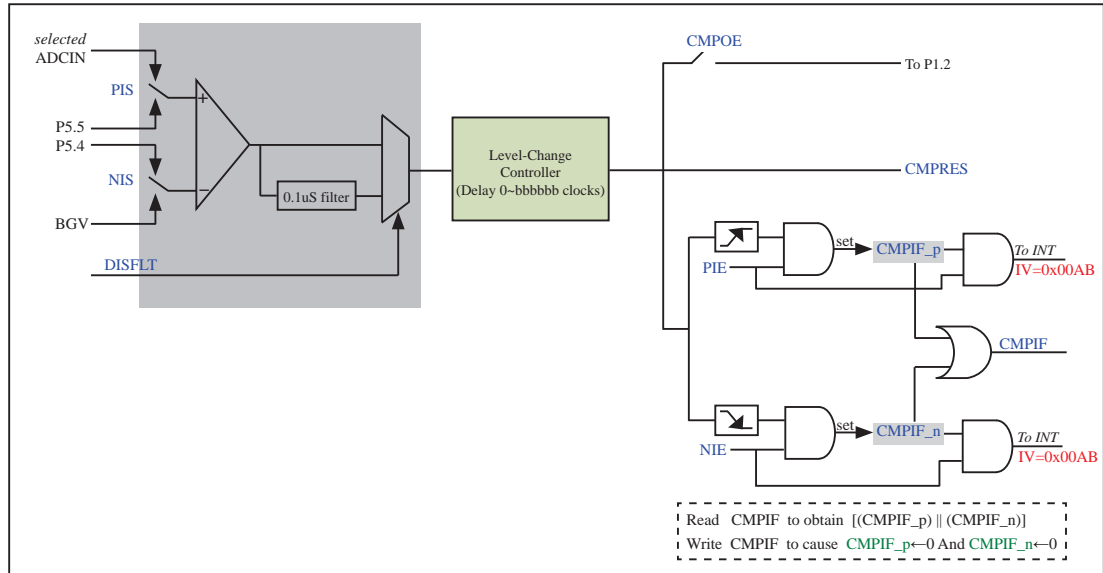
;;; Clear EAXSFR to disable xSFR, return MOVX-DPTR to normal XRAM access
    MOV    A,    P_SW2
    ANL   A,    #01111111B
    MOV    P_SW2, A
;

;; +-----+
;; | Operate PWM output |
;; +-----+
; Enable counter counting, and enable Channel-4 and Channel-5 output
    MOV    A,    PWMCR
    ORL   A,    #10001100B
    MOV    PWMCR,    A
;

```

# Chapter 13 Comparator of STC15W series MCU

There are the function of comparator for STC15W4K32S4 series MCU. Thereinto, the internal structure of comparator of STC15W4K32S4 series MCU is shown below:



STC15W SFRs associated with comparator

Mnemonic	Description	Address	Bit address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
CMPCR1	Comparator Control Register 1	E6H	COMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
CMPCR2	Comparator Control Register 2	E7H	INVCMP0	DISFLT	LCDTY[5:0]					0000,1001	

## 1. Comparator Control Register 1: CMPCR1

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	name	COMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES

COMPEN Enable bit of Comparator

COMPEN=1 Enable comparator

COMPEN=0 Disable comparator, powering off the comparator.

---

**CMPIF** Interrupt Flag bit of Comparator  
 When  $CMPEN = 1$   
     if the comparing result has changed from low to high and if  $PIE$  has been set to 1, a built-in register bit named  $CMPIF\_p$  would be set to 1  
     else if the comparing result has changed from high to low and if  $NIE$  has been set to 1, another built-in register bit named  $CMPIF\_n$  would be set to 1  
 If CPU go to read the value of  $CMPIF$ , the  $(CMPIF\_p \parallel CMPIF\_n)$  would be read;  
 If CPU write 0 into  $CMPIF$ , the value of  $CMPIF\_p$  and  $CMPIF\_n$  would all be cleared .  
 And the conditions of generating comparator interrupt are  
     [  $(EA==1) \ \&\& \ (((PIE==1)\&\&(CMPIF\_p==1)) \parallel ((NIE==1)\&\&(CMPIF\_n==1)))$  ]  
 $CMPIF$  must be cleared manually by software after that CPU has responded to the interrupt.

**PIE** Pos-edge Interrupt Enabling bit  
 $PIE = 1$  enable the comparator interrupt responding to the comparing result changed from low to high  
 $PIE = 0$  disable the comparator interrupt responding to the comparing result changed from low to high.

**NIE** Neg-edge Interrupt Enabling bit  
 $NIE = 1$  enable the comparator interrupt responding to the comparing result changed from high to low  
 $NIE = 0$  disable the comparator interrupt responding to the comparing result changed from high to low.

**PIS** bit to choose the postive pole of comparator  
 $PIS = 1$  choose **ADCIN determined by ADCIS[2:0] as the postive pole of comparator**  
 $PIS = 0$  choose external pin **P5.5 as the postive pole of comparator.**

**NIS** bit to choose the negative pole of comparator  
 $NIS = 1$  choose external pin **P5.4 as the negative pole of comparator**  
 $NIS = 0$  choose internal BandGap Votage **BGV as the negative pole of comparator.**

**CMPOE** Control bit of outputing comparing result  
 $CMPOE = 1$  Make the comparing result of comparator outputting on **P1.2**  
 $CMPOE = 0$  Forbid the comparing result of comparator outputting.

**COMPRES** Flag bit of Comparator Result  
 $COMPRES = 1$  the level of  $CMP+$  is higher than  $CMP-$ ( or the reference voltage of internal BandGap)  
 $COMPRES = 0$  the level of  $CMP+$  is lower than  $CMP-$ ( or the reference voltage of internal BandGap).  
 the bit  $COMPRES$  is a read-only one, so it doesn't make sense to write some value into it by software.

## 2. Comparator Control Register 2: **CMPCR2**

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR2	E7H	name	INVCMPO	DISFLT	LCDTY[5:0]					

**INVCMPO** Inverse Comparator Output  
 $INVCMPO = 1$  Output the comparing result of comparator on P1.2 after inverting them  
 $INVCMPO = 0$  Normal output the comparing result of comparator on P1.2.

---

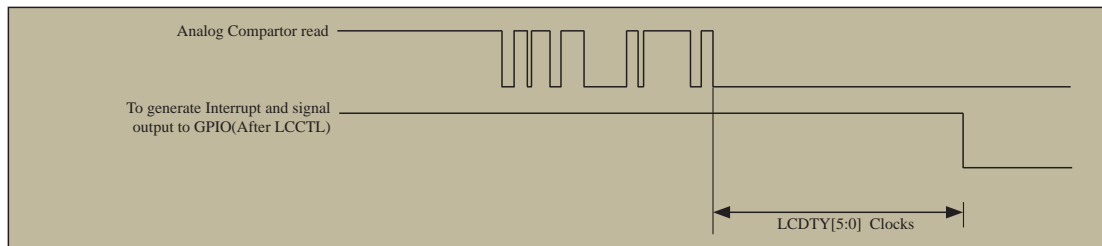
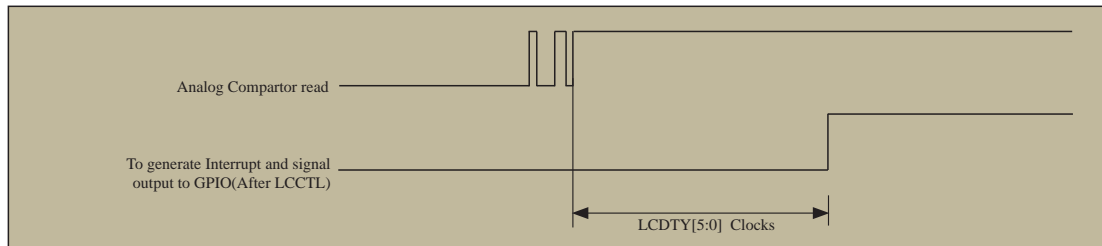
DISFLT Disable the 0.1uS Filter output by comparator  
DISFLT = 1 disable 0.1uS Filter **output by comparator**  
DISFLT = 0 enable the 0.1uS Filter **output by comparator**

LCDTY[5:0] set the Duty of Level-Change control filter in the output terminal of comparator  
bbbbbb =

If the comparing result had changed from low to high, only when the high state has been held on at least bbbbbb clocks would the comparing result of comparator be affirmed to have changed from low to high  
Else the CPU would believe nothing happened

If the comparing result had changed from high to low, only when the low state has been held on at least bbbbbb clocks would the comparing result of comparator be affirmed to have changed from high to low  
Else the CPU would believe nothing happened.

It means no Level-Change Control if LCDTY[5:0] be set to 000000.



---

## 13.1 Comparator Demo Program using Interrupt(C and ASM)

### 1. C Program Listing

```
/*-----*/
/* --- Comparator Demo Program using interrupt -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

sfr    CMPCR1  =  0xE6;           //Comparator control register 1
#define  CMPEN  0x80             //CMPCR1.7 : Enable bit of comparator
#define  CMPIF  0x40             //CMPCR1.6 : Interrupt flag bit of comparator
#define  PIE    0x20             //CMPCR1.5 : Pos-edge Interrupt Enabling bit
#define  NIE    0x10             //CMPCR1.4 : Neg-edge Interrupt Enabling bit
#define  PIS    0x08             //CMPCR1.3 : bit to choose the postive pole of comparator
#define  NIS    0x04             //CMPCR1.2 : bit to choose the negative pole of comparator
#define  CMPOE  0x02             //CMPCR1.1 : Control bit of outputing comparing result
#define  CMPRES 0x01             //CMPCR1.0 : Flag bit of Comparator Result

sfr    CMPCR2  =  0xE7;           //Comparator control register 2
#define  INVCMPO 0x80             //CMPCR2.7 : Inverse Comparator Output
#define  DISFLT  0x40             //CMPCR2.6 : Disable the 0.1uS Filter output by comparator
#define  LCDTY   0x3F             //CMPCR2.[5:0] : set the Duty of Level-Change control filter in
//the output terminal of comparator

sbit   LED      =  P1^1;         //Test pin

void cmp_isr() interrupt 21 using 1 //Comparator interrupt vector
{
    CMPCR1 &= ~CMPIF;           //Clear the finishing flag
    LED = !(CMPCR1 & CMPRES);    //Output the result CMPRES to test pin to display
}

```



---

```

void main()
{
    CMPCR1 = 0;           //Initilize the Comparator control register 1
    CMPCR2 = 0;           //Initilize the Comparator control register 2

    CMPCR1 &= ~PIS;       //choose external pin P5.5(CMP+) as the postive pole of comparator
//    CMPCR1 |= PIS;       //choose ADCIN determined by ADCIS[2:0]
//                           //as the postive pole of comparator

    CMPCR1 &= ~NIS;       //choose internal BandGap Votage BGV
//                           //as the negative pole of comparator
//    CMPCR1 |= NIS;       //choose external pin P5.4(CMP-)as the negative pole of comparator

    CMPCR1 &= ~CMPOE;     //Forbid the comparing result of comparator outputting
//    CMPCR1 |= CMPOE;     //Make the comparing result of comparator outputting on P1.2

    CMPCR2 &= ~INVCMPO;   //Normal output the comparing result of comparator on P1.2
//    CMPCR2 |= INVCMPO;   //Output the comparing result of comparator on P1.2
//                           //after inverting them

    CMPCR2 &= ~DISFLT;    //enable the 0.1uS Filter output by comparator
//    CMPCR2 |= DISFLT;    //disbale 0.1uS Filter output by comparator

    CMPCR2 &= ~LCDTY;
//    CMPCR2 |= (DISFLT & 0x10);

    CMPCR1 |= PIE;        //Enable Pos-edge Interrupt
//    CMPCR1 |= NIE;        //Enable Neg-edge Interrupt

    CMPCR1 |= CMPEN;      //Enable Comparator

    EA = 1;

    while (1);
}

```

## 2. Assembler Listing

```
/*-----*/
```

---

```

/* --- Comparator Demo Program using interrupt -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

CMPCR1      DATA   0E6H      ;Comparator control register 1
CMPEN       EQU     080H      ;CMPCR1.7 : Enable bit of comparator
CMPIF       EQU     040H      ;CMPCR1.6 : Interrupt flag bit of comparator
PIE         EQU     020H      ;CMPCR1.5 : Pos-edge Interrupt Enabling bit
NIE         EQU     010H      ;CMPCR1.4 : Neg-edge Interrupt Enabling bit
PIS         EQU     008H      ;CMPCR1.3 : bit to choose the postive pole of comparator
NIS         EQU     004H      ;CMPCR1.2 : bit to choose the negative pole of comparator
CMPOE       EQU     002H      ;CMPCR1.1 : Control bit of outputing comparing result
CMPRES      EQU     001H      ;CMPCR1.0 : Flag bit of Comparator Result

CMPCR2      DATA   0E7H      ;Comparator control register 2
INVCMPPO    EQU     080H      ;CMPCR2.7 : Inverse Comparator Output
DISFLT      EQU     040H      ;CMPCR2.6 : Disable the 0.1uS Filter output by comparator
LCDTY       EQU     03FH      ;CMPCR2.[5:0] : set the Duty of Level-Change control filter in
                                     ;the output terminal of comparator

LED         BIT     P1.1      ;Test pin
;-----
        ORG     0000H
        LJMP    MAIN

        ORG     00ABH
        LJMP    CMP_ISR      ;Comparator interrupt vector

;-----
        ORG     0100H
MAIN:
        MOV     CMPCR1,    #0      ;Initilize the Comparator control register 1
        MOV     CMPCR2,    #0      ;Initilize the Comparator control register 2

        ANL     CMPCR1,    #NOT PIS
                                     ;choose external pin P5.5(CMP+) as the postive pole of comparator

//        ORL     CMPCR1,    #PIS
                                     ;choose ADCIN determined by ADCIS[2:0] as the postive pole of comparator

```

---

---

```

        ANL    CMPCR1,    #NOT NIS
                ;choose internal BandGap Votage BGV as the negative pole of comparator
//      ORL    CMPCR1,    #NIS
                ;choose external pin P5.4(CMP-)as the negative pole of comparator

        ANL    CMPCR1,    #NOT CMPOE
                ;Forbid the comparing result of comparator outputting
//      ORL    CMPCR1,    #CMPOE
                ;Make the comparing result of comparator outputting on P1.2

        ANL    CMPCR2,    #NOT INVCMPO
                ;Normal output the comparing result of comparator on P1.2
//      ORL    CMPCR2,    #INVCMPO
                ;Output the comparing result of comparator on P1.2 after inverting them
        ANL    CMPCR2,    #NOT DISFLT
                ;enable the 0.1uS Filter output by comparator
//      ORL    CMPCR2,    #DISFLT
                ;disbale 0.1uS Filter output by comparator

        ANL    CMPCR2,    #NOT LCDTY
//      ORL    CMPCR2,    #(DISFLT AND 0x10)
        ORL    CMPCR1,    #PIE            ;Enable Pos-edge Interrupt
//      ORL    CMPCR1,    #NIE            ;Enable Neg-edge Interrupt
        ORL    CMPCR1,    #CMPEN        ;Enable Comparator

        SETB   EA

        SJMP   $

;-----
CMP_ISR:
        PUSH   PSW
        PUSH   ACC

        ANL    CMPCR1,    #NOT CMPIF    ;Clear the finishing flag
        MOV    A,    CMPCR1
        MOV    C,    ACC.0              ;Output the result CMPRES to test pin to display
        MOV    LED,    C

        POP    ACC
        POP    PSW
        RETI

;-----
        END

```

---

---

## 13.2 Comparator Demo Program using Polling(C and ASM)

### 1. C Program Listing

```
/*-----*/
/* --- Comparator Demo Program using polling -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

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

sfr    CMPCR1=    0xE6;    //Comparator control register 1
#define  CMPEN    0x80    //CMPCR1.7 : Enable bit of comparator
#define  CMPIF    0x40    //CMPCR1.6 : Interrupt flag bit of comparator
#define  PIE      0x20    //CMPCR1.5 : Pos-edge Interrupt Enabling bit
#define  NIE      0x10    //CMPCR1.4 : Neg-edge Interrupt Enabling bit
#define  PIS      0x08    //CMPCR1.3 : bit to choose the postive pole of comparator
#define  NIS      0x04    //CMPCR1.2 : bit to choose the negative pole of comparator
#define  CMPOE    0x02    //CMPCR1.1 : Control bit of outputing comparing result
#define  CMPRES    0x01    //CMPCR1.0 : Flag bit of Comparator Result

sfr    CMPCR2    =    0xE7;    //Comparator control register 2
#define  INVCMP0    0x80    //CMPCR2.7 : Inverse Comparator Output
#define  DISFLT    0x40    //CMPCR2.6 : Disable the 0.1uS Filter output by comparator
#define  LCDTY     0x3F    //CMPCR2.[5:0] : set the Duty of Level-Change control filter in
                        //the output terminal of comparator

sbit    LED      =    P1^1;    //Test pin

void main()
{
    CMPCR1 = 0;    //Initalize the Comparator control register 1
    CMPCR2 = 0;    //Initalize the Comparator control register 2
```

---

```

CMPCR1 &= ~PIS;           //choose external pin P5.5(CMP+) as the postive pole of comparator
// CMPCR1 |= PIS;           //choose ADCIN  determined by ADCIS[2:0]
                           //as the postive pole of comparator
CMPCR1 &= ~NIS;           //choose internal BandGap Votage BGV
                           //as the negative pole of comparator
// CMPCR1 |= NIS;           //choose external pin P5.4(CMP-)as the negative pole of comparator

CMPCR1 &= ~CMPOE;         //Forbid the comparing result of comparator outputting
// CMPCR1 |= CMPOE;         //Make the comparing result of comparator outputting on P1.2

CMPCR2 &= ~INVCMPPO;     //Normal output the comparing result of comparator on P1.2
// CMPCR2 |= INVCMPPO;     //Output the comparing result of comparator on P1.2
                           //after inversing them

CMPCR2 &= ~DISFLT;       //enable the 0.1uS Filter output by comparator
// CMPCR2 |= DISFLT;       //disbale 0.1uS Filter output by comparator

CMPCR2 &= ~LCDTY;
// CMPCR2 |= (DISFLT & 0x10);

CMPCR1 |= CMPEN;         //Enable Comparator
while (!(CMPCR1 & CMPIF)); //Query the finishing flag
CMPCR1 &= ~CMPIF;         //Clear the finishing flag
LED = !(CMPCR1 & CMPRES); //Output the result CMPRES to test pin to display

while (1);
}

```

## 2. Assembler Listing

```

/*-----*/
/* --- Comparator Demo Program using polling -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

```

```

//suppose the frequency of test chip is 18.432MHz

```

---

```

CMPCR1      DATA  0E6H      ;Comparator control register 1
CMPEN       EQU    080H      ;CMPCR1.7 : Enable bit of comparator
CMPIF       EQU    040H      ;CMPCR1.6 : Interrupt flag bit of comparator
PIE         EQU    020H      ;CMPCR1.5 : Pos-edge Interrupt Enabling bit
NIE         EQU    010H      ;CMPCR1.4 : Neg-edge Interrupt Enabling bit
PIS         EQU    008H      ;CMPCR1.3 : bit to choose the postive pole of comparator
NIS         EQU    004H      ;CMPCR1.2 : bit to choose the negative pole of comparator
CMPOE       EQU    002H      ;CMPCR1.1 : Control bit of outputing comparing result
CMPRES      EQU    001H      ;CMPCR1.0 : Flag bit of Comparator Result

CMPCR2      DATA  0E7H      ;Comparator control register 2
INVCMPO     EQU    080H      ;CMPCR2.7 : Inverse Comparator Output
DISFLT      EQU    040H      ;CMPCR2.6 : Disable the 0.1uS Filter output by comparator
LCDTY       EQU    03FH      ;CMPCR2.[5:0] : set the Duty of Level-Change control filter in
                                ;the output terminal of comparator

LED         BIT     P1.1      ;Test pin
;-----
          ORG     0000H
          LJMP    MAIN
;-----
          ORG     0100H
MAIN:
          MOV     CMPCR1,    #0          ;Initalize the Comparator control register 1
          MOV     CMPCR2,    #0          ;Initalize the Comparator control register 2

          ANL     CMPCR1,    #NOT PIS
                                ;choose external pin P5.5(CMP+) as the postive pole of comparator
//          ORL     CMPCR1,    #PIS
                                ;choose ADCIN determined by ADCIS[2:0] as the postive pole of comparator

          ANL     CMPCR1,    #NOT NIS
                                ;choose internal BandGap Votage BGV as the negative pole of comparator
//          ORL     CMPCR1,    #NIS
                                ;choose external pin P5.4(CMP-)as the negative pole of comparator

          ANL     CMPCR1,    #NOT CMPOE
                                ;Forbid the comparing result of comparator outputting
//          ORL     CMPCR1,    #CMPOE
                                ;Make the comparing result of comparator outputting on P1.2

          ANL     CMPCR2,    #NOT INVCMPO
                                ;Normal output the comparing result of comparator on P1.2

```

---

---

```

//      ORL    CMPCR2,    #INVCMP0
                                ;Output the comparing result of comparator on P1.2 after inverting them
      ANL    CMPCR2,    #NOT DISFLT
                                ;enable the 0.1uS Filter output by comparator
//      ORL    CMPCR2,    #DISFLT
                                ;disbale 0.1uS Filter output by comparator

      ANL    CMPCR2,    #NOT LCDTY
//      ORL    CMPCR2,    #(DISFLT AND 0x10)

      ORL    CMPCR1,    #CMPEN      ;Enable Comparator
WAIT:   MOV    A,        CMPCR1      ;Query the finishing flag

      ANL    A,        #CMPIF
      JZ    WAIT
      ANL    CMPCR1,    #NOT CMPIF  ;Clear the finishing flag
      MOV    A,        CMPCR1
      MOV    C,        ACC.0        ;Output the result CMPRES to test pin to display
      MOV    LED,      C

      SJMP   $

      END

```

# Chapter 14 Capacitive Sensing Touch Key

## — Achieved by ADC of STC15W series

Touch key as the most important way of human interaction is one of the most common circuit modules. Key may be include engine inducing key-press and non engine inducing key-press. For engine inducing key-press, especially cheap ones, they have a disadvantage which is easily destroyed. However, for the non engine ones, they have longer serving life and are more convenient to use because of without mechanical contact.

Capacitive sensing touch key is one of the cheapest non engine keys. Now let us to learn about how to achieve capacitive sensing touch key by ADC of STC15W4K32S4 series MCU.

The next three circuit figures with the same principle are the most often used. Take the fig.2 for example in this article.

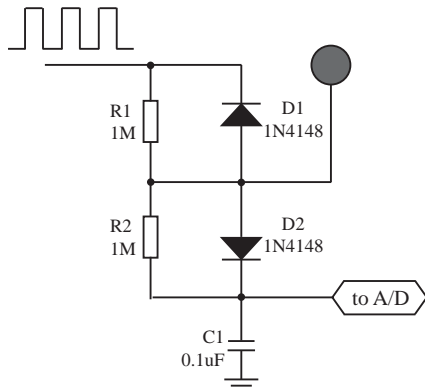


Fig. 1

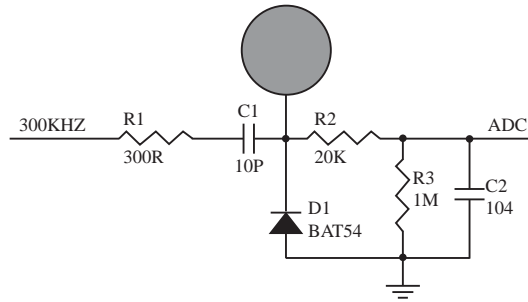


Fig. 2

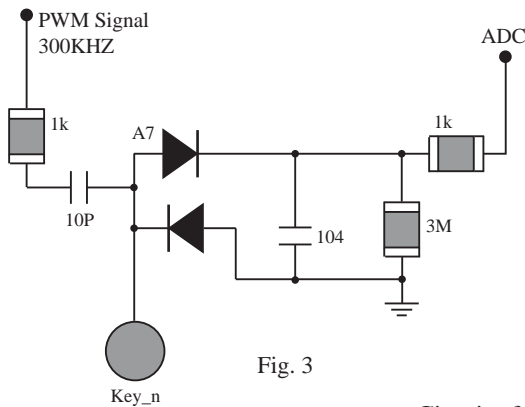


Fig. 3

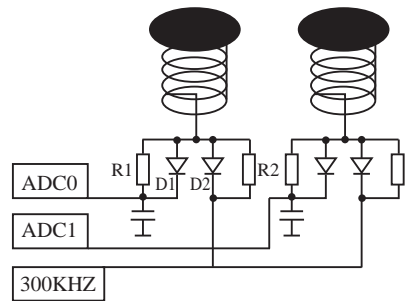
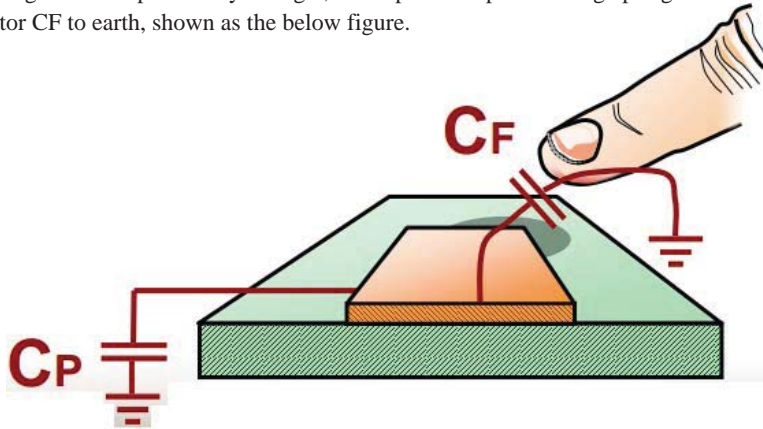


Fig. 4 touch key with sensing spring

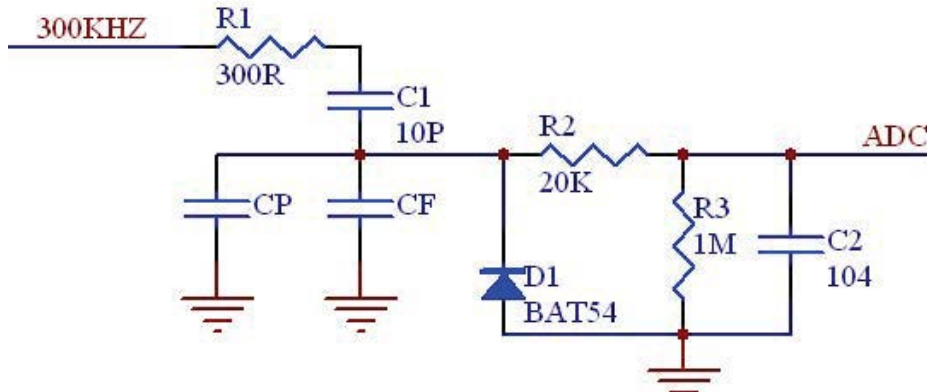
Circuit of capacitive sensing touch key



Circuit capacitive sensing touch key of Fig.4 may be used in practice to expand the area pressed by finger with sensing spring. The sensing spring has a capacitor  $C_p$  to earth, which is equivalent to a metal plate to earth. When the sensing spring has been pressed by a finger, the capacitor  $C_p$  of sensing spring to earth will in parallel with another capacitor  $C_f$  to earth, shown as the below figure.



Now let us explain the circuit : the 300KHz square waves input voltage is divided by the capacitor  $C_p$  of sensing spring to earth in parallel with the finger capacitor  $C_f$  and in series with the capacitor  $C_1$ , and then rectified by the diode  $D_1$ , and sent to ADC after filtered by  $R_2$  and  $C_2$ . If a finger goes to press the touch key, the voltage sent to the ADC will be decreased result in the touch gesture can be detected by the program



The following text is the detail program of utilizing ADC of STC15 series to achieve the capacitive sensing touch key.

### 1. C Program Listing

```

/*-----*/
/* --- Demo Program utilizing ADC of STC15 series to achieve the capacitive sensing touch key -----*/
/* If you want to use the program or the program referenced in the -----*/

```

---

```

/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

/*****      function declaration      *****/

Take ADC of STC15W408AS MCU for example
.
//suppose the frequency of test chip is 24MHz
*****/

#include <reg51.h>
#include <intrins.h>

#define MAIN_Fosc          24000000UL      //Define the master clock

typedef unsigned char     u8;
typedef unsigned int      u16;
typedef unsigned long     u32;

#define Timer0_Reload     (65536UL -(MAIN_Fosc / 600000))
                          //the reload value of Timer 0, correspond to 300KHZ

sfr    P1ASF              = 0x9D;         //Only write  select analog input
sfr    ADC_CONTR          = 0xBC;
sfr    ADC_RES            = 0xBD;
sfr    ADC_RESL          = 0xBE;
sfr    AUXR               = 0x8E;
sfr    AUXR2              = 0x8F;

/*****      Define Constant      *****/

#define TOUCH_CHANNEL     8              //number of ADC channels

#define ADC_90T (3<<5)                  //ADC time 90T
#define ADC_180T (2<<5)                  //ADC time 180T
#define ADC_360T (1<<5)                  //ADC time 360T
#define ADC_540T0                    //ADC time 540T
#define ADC_FLAG (1<<4)                  //clear by software
#define ADC_START (1<<3)                  //clear Automatically

/*****      Define Variable      *****/
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];           //ADC value at present
u16     idata adc_prev[TOUCH_CHANNEL];     //last ADC value
u16     idata TouchZero[TOUCH_CHANNEL];    //ADC value of 0
u8      idata TouchZeroCnt[TOUCH_CHANNEL]; //track for and count from 0 automatically

u8      cnt_250ms;

/***** Define Function *****/
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);

/***** Main Funicton *****/
void main(void)
{
    u8     i;

    delay_ms(50);

    ET0 = 0;
    TR0 = 0;
    AUXR |= 0x80;           //Timer0 set as 1T mode
    AUXR2 |= 0x01;        //Enable output clock
    TMOD = 0;             //Timer0 set as Timer, 16 bits Auto Reload.
    TH0 = (u8)(Timer0_Reload >> 8);
    TL0 = (u8)Timer0_Reload;
    TR0 = 1;

    ADC_init();           //Initialize ADC
    delay_ms(50);        //Delay 50ms

    for(i=0; i<TOUCH_CHANNEL; i++)
    {

```

---

---

```

        adc_prev[i] = 1023;
        TouchZero[i] = 1023;
        TouchZeroCnt[i] = 0;
    }
    cnt_250ms = 0;

    while (1)
    {
        delay_ms(50);           //Dispose the touch key every 50ms
        ShowLED();
        if(++cnt_250ms >= 5)
        {
            cnt_250ms = 0;
            AutoZero();         //Dispose the function AutoZero() every 250ms
        }
    }
}
/*****/

//=====
// Function:      void delay_ms(unsigned char ms)
// Description:   Delay function
// Parameter:     ms, time to delay, only among 1~255ms.
// Return:        none.
// Version:       VER1.0
// Date:          2013-4-1
// Remark:
//=====
void delay_ms(u8 ms)
{
    unsigned int i;
    do {
        i = MAIN_Fosc / 13000;
        while(--i);
    }while(--ms);
}
/*****/ Initialize ADC *****/
void ADC_init(void)
{
    P1ASF = 0xff;           //8 channels ADC
    ADC_CONTR = 0x80;      //Enable ADC
}

```

---

```

=====
// Function:      u16      Get_ADC10bitResult(u8 channel)
// Description:   Read ADC reslut by querying.
// Parameter:    channel: choose ADC to convert.
// Return:       10 bits ADC result.
// Version:      V1.0, 2012-10-22
=====
u16      Get_ADC10bitResult(u8 channel)      //channel = 0~7
{
    ADC_RES = 0;
    ADC_RESL = 0;
    ADC_CONTR = 0x80 | ADC_90T | ADC_START | channel;      // trigger ADC
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    while((ADC_CONTR & ADC_FLAG) == 0)      ;      //Waiting for finishing converting of ADC
    ADC_CONTR = 0x80;      //Clear flag
    return(((u16)ADC_RES << 2) | ((u16)ADC_RESL & 3));      //Return the ADC result
}

/***** Track for the 0 automatically *****/
void      AutoZero(void)      //Call the function every 250ms
{
    u8      i;
    u16     j,k;

    for(i=0; i<TOUCH_CHANNEL; i++)      //Deal with 8 channels
    {
        j = adc[i];
        k = j - adc_prev[i];      //read one value before decrease
        F0 = 0;      //press
        if(k & 0x8000)      F0 = 1,      k = 0 - k; //release to get the difference value of two sample
        if(k >= 20)      //the difference is large
        {
            TouchZeroCnt[i] = 0;      //If the difference is large clear the counter
            if(F0)      TouchZero[i] = j; //If release and the difference is large,replace directly
        }
        else      //the difference is samll, wriggle, track for 0 automatically
        {
            if(++TouchZeroCnt[i] >= 20)
            {
                TouchZeroCnt[i] = 0;
                TouchZero[i] = adc_prev[i];
            }
        }
        adc_prev[i] = j;      //Save the sample value of this time
    }
}

```

---

```

/*****Get the touch action , this function is called every 50ms*****/
u8 check_adc(u8 index)                                     //Judge press or release
{
    u16    delta;
    adc[index] = 1023 - Get_ADC10bitResult(index);        //Get the value of ADC to translate the
                                                         //press action, increase the value of ADC
    if(adc[index] < TouchZero[index])    return 0;        //If the ADC value is smaller than 0,
                                                         //release action will be regarded as happening

    delta = adc[index] - TouchZero[index];
    if(delta >= 40)    return 1;                          //Press
    if(delta <= 20)    return 0;                          //Release
    return 2;                                               //Hold on
}

/***** Deal with the touch action, this function is called every 50ms *****/
void ShowLED(void)
{
    u8    i;

    i = check_adc(0);
    if(i == 0)    P_LED0 = 1;    //LED indicator is off
    if(i == 1)    P_LED0 = 0;    //LED indicator is on

    i = check_adc(1);
    if(i == 0)    P_LED1 = 1;    //LED indicator is off
    if(i == 1)    P_LED1 = 0;    //LED indicator is on

    i = check_adc(2);
    if(i == 0)    P_LED2 = 1;    //LED indicator is off
    if(i == 1)    P_LED2 = 0;    //LED indicator is on

    i = check_adc(3);
    if(i == 0)    P_LED3 = 1;    //LED indicator is off
    if(i == 1)    P_LED3 = 0;    //LED indicator is on

    i = check_adc(4);
    if(i == 0)    P_LED4 = 1;    //LED indicator is off
    if(i == 1)    P_LED4 = 0;    //LED indicator is on

    i = check_adc(5);
    if(i == 0)    P_LED5 = 1;    //LED indicator is off
    if(i == 1)    P_LED5 = 0;    //LED indicator is on

    i = check_adc(6);
    if(i == 0)    P_LED6 = 1;    //LED indicator is off
    if(i == 1)    P_LED6 = 0;    //LED indicator is on

    i = check_adc(7);
    if(i == 0)    P_LED7 = 1;    //LED indicator is off
    if(i == 1)    P_LED7 = 0;    //LED indicator is on

}

```

---

---

## 2. Assembler Listing

```
/*-----*/
/* --- Demo Program utilizing ADC of STC15 series to achieve the capacitive sensing touch key -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

/***** function declaration *****/

Take ADC of STC15W408AS MCU for example
.
//suppose the frequency of test chip is 24MHz
*****/

;***** Define Macro *****/

Fosc_KHZ      EQU      24000                ;Define the master clock KHZ

STACK_POIRTER EQU      0D0H                ;Start Address of stack

Timer0_Reload EQU      (65536 - Fosc_KHZ/600) ;the reload value of Timer 0, correspond to 300KHZ

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

PIASF         DATA    0x9D;                ;Only write select analog input
ADC_CONTR     DATA    0xBC;
ADC_RES       DATA    0xBD;
ADC_RESL      DATA    0xBE;
AUXR          DATA    0x8E;
AUXR2         DATA    0x8F;

/***** Define Constant *****/
TOUCH_CHANNEL EQU      8                    ;number of ADC channels
ADC_90T       EQU      (3 SHL 5)           ;ADC time 90T
ADC_180T      EQU      (2 SHL 5)           ;ADC time 180T
```

---

```

ADC_360T    EQU (1 SHL 5)                ;ADC time 360T
ADC_540T    EQU 0                       ;ADC time 540T
ADC_FLAG    EQU (1 SHL 4)               ;clear by software
ADC_START   EQU (1 SHL 3)               ;clear Automatically

```

```

/***** Define Variable *****/

```

```

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    ;    ADC value at present    30H~3FH, a value with two bytes
adc_prev    EQU    40H    ;    last ADC value    40H~4FH, a value with two bytes
TouchZero   EQU    50H    ;    ADC value of 0    50H~5FH, a value with two bytes
TouchZeroCnt EQU    60H    ;    rack for and count from 0 automatically    60H~67H

cnt_250ms   DATA    68H    ;

```

```

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

```

```

    ORG    00H                ;reset
    LJMP   F_Main

    ORG    03H                ;0 INT0 interrupt
    RETI
    LJMP   F_INT0_Interrupt

    ORG    0BH                ;1 Timer0 interrupt
    LJMP   F_Timer0_Interrupt

    ORG    13H                ;2 INT1 interrupt
    LJMP   F_INT1_Interrupt

    ORG    1BH                ;3 Timer1 interrupt
    LJMP   F_Timer1_Interrupt

    ORG    23H                ;4 UART1 interrupt
    LJMP   F_UART1_Interrupt

    ORG    2BH                ;5 ADC and SPI interrupt

```



---

```

LJMP  F_ADC_Interrupt

ORG   33H                               ;6 Low Voltage Detect interrupt
LJMP  F_LVD_Interrupt

ORG   3BH                               ;7 PCA interrupt
LJMP  F_PCA_Interrupt

ORG   43H                               ;8 UART2 interrupt
LJMP  F_UART2_Interrupt

ORG   4BH                               ;9 SPI interrupt
LJMP  F_SPI_Interrupt

ORG   53H                               ;10 INT2 interrupt
LJMP  F_INT2_Interrupt

ORG   5BH                               ;11 INT3 interrupt
LJMP  F_INT3_Interrupt

ORG   63H                               ;12 Timer2 interrupt
LJMP  F_Timer2_Interrupt

ORG   83H                               ;16 INT4 interrupt
LJMP  F_INT4_Interrupt

```

```

;***** Main Procedure *****/

```

```

F_Main:

```

```

        MOV    R0, #1
L_ClearRamLoop:                ;ClearAM
        MOV    @R0, #0
        INC    R0
        MOV    A, R0
        CJNE  A, #0FFH, L_ClearRamLoop

        MOV    SP, #STACK_POIRTER
        MOV    PSW, #0
        USING 0                  ;Choose Nun.0 R0~R7

```

```

;===== Initialize Procedure =====

```

```

        MOV    R7, #50

```

---

```

    LCALL F_delay_ms

    CLR    ET0                ;
    CLR    TR0                ;
    ORL    AUXR, #080H       ; Timer0 set as 1T mode
    ORL    AUXR2, #01H      ; Enable output clock
    MOV    TMOD, #0          ; Timer0 set as Timer, 16 bits Auto Reload.
    MOV    TH0, #HIGH Timer0_Reload
    MOV    TL0, #LOW  Timer0_Reload ;
    SETB   TR0

    LCALL F_ADC_init
    MOV    R7, #50
    LCALL F_delay_ms

    MOV    R0, #adc_prev          ; Initialize the last value of ADC
L_Init_Loop1:
    MOV    @R0, #03H
    INC    R0
    MOV    @R0, #0FFH
    INC    R0
    MOV    A, R0
    CJNE  A, #(adc_prev + TOUCH_CHANNEL * 2), L_Init_Loop1

    MOV    R0, #TouchZero        ; Initialize the ADC value of 0
L_Init_Loop2:
    MOV    @R0, #03H
    INC    R0
    MOV    @R0, #0FFH
    INC    R0
    MOV    A, R0
    CJNE  A, #(TouchZero + TOUCH_CHANNEL * 2), L_Init_Loop2

    MOV    R0, #TouchZeroCnt;
L_Init_Loop3:
    MOV    @R0, #0
    INC    R0
    MOV    A, R0
    CJNE  A, #(TouchZeroCnt + TOUCH_CHANNEL), L_Init_Loop3

    MOV    cnt_250ms, #5

;===== Main Loop =====
L_MainLoop:

```

---

---

```

MOV    R7, #50                ;Delay 50ms
LCALL  F_delay_ms
LCALL  F_ShowLED              ; Deal with the value indicating the touch action a time
DJNZ   cnt_250ms, L_MainLoop

MOV    cnt_250ms, #5          ;Dispose the function AutoZero() every 250m
LCALL  F_AutoZero

SJMP   L_MainLoop

;===== End Main Procedure =====

; /***** Initialize ADC *****/
F_ADC_init:
MOV    P1ASF, #0FFH          ;8 channels ADC
MOV    ADC_CONTR, #080H      ;Enable ADC
RET
; END OF ADC_init

; //=====
; //Function: F_Get_ADC10bitResult
; // Description: Read ADC result by querying..
; // Parameter: R7: choose ADC to convert.
; // Return: R6 R7 == 10 bits ADC result.
; // Version: V1.0, 2014-3-25
; //=====

F_Get_ADC10bitResult:
USING 0                      ;Choose Num.0 R0~R7

MOV    ADC_RES, #0
MOV    ADC_RESL, #0
MOV    A, R7
ORL    A, #0E8H              ;(0x80 OR ADC_90T OR ADC_START)      ;trigger ADC
MOV    ADC_CONTR, A
NOP
NOP
NOP
NOP

L_10bitADC_Loop1:
MOV    A, ADC_CONTR

JNB    ACC.4, L_10bitADC_Loop1 ;Waiting for finishing converting of ADC

```

---

---

```

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
; END OF _Get_ADC10bitResult

; /***** Track for the 0 automatically *****/
F_AutoZero:                       ;Call the function every 250ms
    USING 0

    CLR   A
    MOV   R5,A
L_AutoZero_Loop:
    ;[R6 R7] = adc[i], (j = adc[i])
    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

    ; to get the abs of difference value [R2 R3] = adc[i] - adc_prev[i], (k = j - adc_prev[i];)
    //read one value before decrease
    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

; to get the abs of difference value [R2 R3], if(k & 0x8000)F0 = 1, k = 0 - k;
//release to get the difference value of two sample
CLR    F0                ;Press
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                ;Count [R2 R3] - #20, if(k >= 20) //the difference is large
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 difference is large clear the counter
                                ;TouchZeroCnt[i] = 0;

ADD    A,R5
MOV    R0,A
MOV    @R0, #0

; if(F0) TouchZero[i] = j; //If release and the difference is large, replace directly
JNB    F0,L_AutoZero_3
MOV    A,R5
ADD    A,ACC
ADD    A,#LOW (TouchZero)
MOV    R0,A
MOV    @R0,AR6
INC    R0
MOV    @R0,AR7
SJMP   L_AutoZero_3

L_AutoZero_2:
; if(++TouchZeroCnt[i] >= 20)
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             ;TouchZero[i] = adc_prev[i];
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,AR2
INC    R0
MOV    @R0,AR3

L_AutoZero_3:
                ;          Save the sample value          adc_prev[i] = j;
MOV    A,R5
ADD    A,ACC
ADD    A,#LOW (adc_prev)
MOV    R0,A
MOV    @R0,AR6
INC    R0
MOV    @R0,AR7

INC    R5
MOV    A,R5
XRL   A,#08H
JZ    $ + 5H
LJMP  L_AutoZero_Loop

RET

; END OF AutoZero

```

---

---

; /\*\*\*\*\* Get the touch action , this function is called every 50ms \*\*\*\*\*/

```
F_check_adc:                                ;Get the value of ADC to translate the
                                             ;press action, increase the value of ADC

    USING 0

    MOV    R4, AR7
;   adc[index] = 1023 - Get_ADC10bitResult(index);    ;Get the value of ADC to translate the
                                             ;press action, increase the value of ADC
    LCALL  F_Get_ADC10bitResult    ;return the ADC value to [R6 R7]
    CLR    C
    MOV    A,#0FFH                ;1023 - [R6 R7]
    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,AR6
    INC    R0
    MOV    @R0,AR7

;   if(adc[index] < TouchZero[index])    return    0;    //If the ADC value is smaller than 0,
                                             //release action will be regarded as happening

    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                    ;Count adc[index] - TouchZero[index]
    MOV    A,R6
    DEC    R1
    SUBB   A,@R1
```

---

```

JNC    L_check_adc_1    ;if(adc[index] >= TouchZero[index]), Jump
MOV    R7,#00H          ;if(adc[index] < TouchZero[index]), the ADC value is smaller than 0,
                        ;release action will be regarded as happening, return 0

RET

L_check_adc_1:
                                ; to get the difference value
                                ;[R6 R7] = delta = adc[index] - TouchZero[index];

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

                                ;---- Variable 'delta' assigned to Register 'R6/R7' ----

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;           //Press, 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            ;if(delta <= 20)   return 0;           //Release, return 0

```

---



---

```

    RET
L_check_adc_3:
    MOV    R7,#2          ;if((delta > 20) && (delta < 40))      Hold on , return 2
    RET
; END OF _check_adc

/***** Deal with the touch action, this function is called every 50ms *****/
F_ShowLED:
    USING  0

    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      ;if(i == 0), LED indicator is off , if(i == 1), LED indicator is on
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      ;if(i == 0), LED indicator is off , if(i == 1), LED indicator is on
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      ;if(i == 0), LED indicator is off , if(i == 1), LED indicator is on
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      ;;if(i == 0), LED indicator is off , if(i == 1), LED indicator is on
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      ;;if(i == 0), LED indicator is off , if(i == 1), LED indicator is on
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      ;;if(i == 0), LED indicator is off , if(i == 1), LED indicator is on
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      ;;if(i == 0), LED indicator is off , if(i == 1), LED indicator is on
L_QuitCheck6:

```

```

MOV    R7, #7
LCALL  F_check_adc

```

---

```

        MOV    A,R7
        ANL    A, #0FEH
        JNZ    L_QuitCheck7
        MOV    A, R7
        MOV    C, ACC.0
        CPL    C
        MOV    P_LED7, C      ;if(i == 0), LED indicator is off , if(i == 1), LED indicator is on
L_QuitCheck7:

        RET
; END OF ShowLED

;=====
; Function: F_delay_ms
; Description: Delay function
; Parameter: R7: delay time.
; Return: none.
; Version: VER1.0
; Date: 2013-4-1
; Remark: Except ACCC and PSW, all common registers must be pushed
;=====
F_delay_ms:
        PUSH  AR3              ;Push R3
        PUSH  AR4              ;Push R4

L_delay_ms_1:
        MOV   R3, #HIGH (Fosc_KHZ / 13)
        MOV   R4, #LOW  (Fosc_KHZ / 13)

L_delay_ms_2:
        MOV   A, R4              ;1T          Total 13T/loop
        DEC   R4                  ;2T
        JNZ   L_delay_ms_3        ;4T
        DEC   R3

L_delay_ms_3:
        DEC   A                  ;1T
        ORL   A, R3              ;1T
        JNZ   L_delay_ms_2        ;4T

        DJNZ  R7, L_delay_ms_1

        POP   AR4                ;Pop R2
        POP   AR3                ;Pop R3
        RET

```

---

---

```
*****
;*****
;***** Interrupt Function *****
F_Timer0_Interrupt:
    RETI

F_Timer1_Interrupt:
    RETI

F_Timer2_Interrupt:
    RETI

F_INT0_Interrupt:
    RETI

F_INT1_Interrupt:
    RETI

F_INT2_Interrupt:
    RETI

F_INT3_Interrupt:
    RETI

F_INT4_Interrupt:
    RETI

F_UART1_Interrupt:
    RETI

F_UART2_Interrupt:
    RETI

F_ADC_Interrupt:
    RETI

F_LVD_Interrupt:
    RETI

F_PCA_Interrupt:
    RETI

F_SPI_Interrupt:
    RETI

    END
```

---

# Chapter 15 Synchronous Serial Peripheral Interface

STC15W4K32S4 series MCU also provides another high-speed serial communication interface, the SPI interface. SPI is a full-duplex, high-speed, synchronous communication bus with two operation modes: Master mode and Slave mode.

Up to 3Mbit/s can be supported in either Master or Slave mode under the SYSclk=12MHz. Two status flags are provided to signal the transfer completion and write-collision occurrence.

For STC15W4K32S4 series MCU, thier SPI all can be switched in 3 groups of pins :

[SS/P1.2, MOSI/P1.3, MISO/P1.4, SCLK/P1.5];

[SS\_2/P2.4, MOSI\_2/P2.3, MISO\_2/P2.2, SCLK\_2/P2.1];

[SS\_3/P5.4, MOSI\_3/P4.0, MISO\_3/P4.1, SCLK\_3/P4.3]

## 15.1 Special Function Registers related with SPI

SPI Management SFRs

Mnemonic	Description	Address	Bit address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
SPCTL	SPI Control Register	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	0000,0100
SPSTAT	SPI Status Register	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
SPDAT	SPI Data Register	CFH									0000,0000
AUXR1 P_SW1	Auxiliary Register 1	A2H	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	-	DPS	0100,0000

### 1. SPI Control register: SPCTL (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SPCTL	CEH	name	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

SSIG : Control whether SS pin is ignored or not.

If SSIG=1, MSTR(SPCTL.4) decides whether the device is a master or slave.

If SSIG=0, the SS pin decides whether the device is a master or slave. SS pin can be used as I/O port.

SPEN : SPI enable bit.

If SPEN=0, the SPI interface is disabled and all SPI pins will be general-purpose I/O ports.

If SPEN=1, the SPI is enabled.

DORD : Set the transmitted or received SPI data order.

If DORD=1, The LSB of the data word is transmitted first.

If DORD=0, The MSB of the data word is transmitted first.

MSTR : Master/Slave mode select bit.

If MSTR=0, set the SPI to play as Slave part.

If MSTR=1, set the SPI to play as Master part.

---

CPOL : SPI clock polarity select bit.

If CPOL=1, SPICLK is high level when in idle mode. The leading edge of SPICLK is the falling edge and the trailing edge is the rising edge.

If CPOL=0, SPICLK is low when idle. The leading edge of SPICLK is the rising edge and the trailing edge is the falling edge.

CPHA : SPI clock phase select bit.

If CPHA=1, Data is driven on the leading edge of SPICLK, and is sampled on the trailing edge.

If CPHA=0, Data is driven when SS pin is low (SSIG=0) and changes on the trailing edge of SPICLK. Data is sampled on the leading edge of SPICLK. (Note : If SSIG=1, CPHA must not be 0, otherwise the operation is undefined)

SPR1-SPR0 : SPI clock rate select bit (when in master mode)

SPI clock frequency select bit

SPR1	SPR0	SPI clock (SCLK )
0	0	CPU_CLK/4
0	1	CPU_CLK/8
1	0	CPU_CLK/32
1	1	CPU_CLK/64

CPU\_CLK is CPU clock.

When CPHA equals 0, SSIG must be 0 and SS pin must be negated and reasserted between each successive serial byte transfer. If the SPDAT register is written while SS is active(0), a write collision error results and WCOL is set.

When CPHA equals 1, SSIG may be 0 or 1. If SSIG=0, the SS pin may remain active low between successive transfers(can be tied low at any times). This format is sometimes preferred for use in systems having a single fixed master and a single slave configuration.

## 2. SPI State register: SPSTAT (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	CDH	name	SPIF	WCOL	-	-	-	-	-	-

SPIF : SPI transfer completion flag.

When a serial transfer finishes, the SPIF bit is set and an interrupt is generated if both the ESPI (IE.6) bit and the EA (IE.7) bit are set. If SS is an input and is driven low when SPI is in master mode with SSIG = 0, SPIF will also be set to signal the “mode change”.The SPIF is cleared in software by “writing 1 to this bit”.

WCOL : SPI write collision flag.

The WCOL bit is set if the SPI data register, SPDAT, is written during a data transfer. The WCOL flag is cleared in software by “writing 1 to this bit”.

### 3. SPI Data register : SPDAT (Non bit-addressable)

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
SPDAT	CFH	name								

The SFR SPDAT holds the data to be transmitted or the data received.

### 4. SPI Switch Control bits: SPI\_S1 / P\_SW1.7 and SPI\_S0 / P\_SW1.6

AUXR1 / P\_SW1 : Peripheral function switch register (Non bit-addressable)

Mnemonic	Address	Name	7	6	5	4	3	2	1	0	Reset Value
AUXR1 P_SW1	A2H	Auxiliary register 1	S1_S1	S1_S0	CCP_S1	CCP_S0	SPI_S1	SPI_S0	0	DPS	0100,0000

SPI can be switched in 3 groups of pins by selecting the control bits SPI_S1 and SPI_S0		
SPI_S1	SPI_S0	SPI can be switched in P1 and P2 and P4
0	0	SPI on [P1.2/SS_2,P1.3/MOSI_1,P1.4/MISO_1,P1.5/SCLK_1]
0	1	SPI on [P2.4/SS_2,P2.3/MOSI_2,P2.2/MISO_2,P2.1/SCLK_2]
1	0	SPI on [P5.4/SS_3,P4.0/MOSI_3,P4.1/MISO_3,P4.3/SCLK_3]
1	1	Invalid

### 5. Registers bits related with SPI Interrupt : EA, ESPI and PSPI

IE2: Interrupt Enable 2 Register (Non bit-addressable)

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

ESPI : SPI interrupt enable bit.

If ESPI = 0, SPI interrupt would be disabled.

If ESPI = 1, SPI interrupt would be enabled.

IE: Interrupt Enable Register (Bit-addressable)

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

EA : disables all interrupts.

If EA = 0, no interrupt will be acknowledged.

If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

IP2: Interrupt Priority Register (Non bit-addressable)

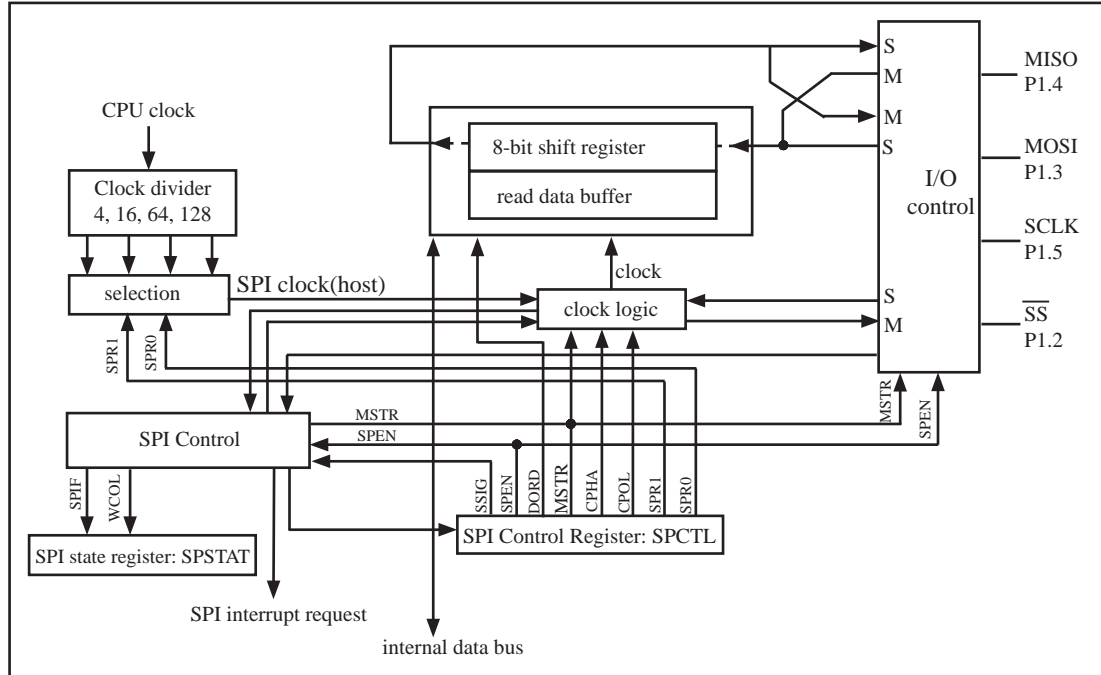
SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
IP2	B5H	name	-	-	-	-	-	-	PSPI	PS2

PSPI : SPI interrupt priority control bit.

if PSPI=0, SPI interrupt is assigned lowest priority (priority 0).

if PSPI=1, SPI interrupt is assigned highest priority (priority 1).

## 15.2 SPI Structure



SPI block diagram

The SPI interface has three pins implementing the SPI functionality: SCLK(P1.5), MISO(P1.4), MOSI(P1.3). An extra pin SS(P1.2) is designed to configure the SPI to run under Master or Slave mode. SCLK, MOSI and MISO are typically tied together between two or more SPI devices. Data flows from master to slave on MOSI(Master Out Slave In) pin and flows from slave to master on MISO(Master In Slave Out) pin. The SCLK signal is output in the master mode and is input in the slave mode. If the SPI system is disabled, i.e., SPEN(SPCTL.6)=0, these pins are configured as general-purpose I/O port(P1.2 ~ P1.5).

SS is the slave select pin. In a typical configuration, an SPI master asserts one of its port pins to select one SPI device as the current slave. An SPI slave device uses its SS pin to determine whether it is selected. But if SPEN=0 or SSIG(SPCTL.7) bit is 1, the SS pin is ignored. Note that even if the SPI is configured as a master(MSTR/SPCTL.4=1), it can still be converted to a slave by driving the SS pin low. When the conversion happened, the SPIF bit(SPSTAT.7) will be set.

Two devices with SPI interface communicate with each other via one synchronous clock signal, one input data signal, and one output data signal. There are two concerns the user should take care, one of them is latching data on the negative edge or positive edge of the clock signal which named polarity, the other is keeping the clock signal low or high while the device idle which named phase. Permuting those states from polarity and phase, there could be four modes formed, they are SPI-MODE-0, SPI-MODE-1, SPI-MODE-2, SPI-MODE-3. Many device declares that they meet SPI mechanism, but few of them are adaptive to all four modes. The STC15W4K32S4 series are flexible to be configured to communicate to another device with MODE-0, MODE-1, MODE-2 or MODE-3 SPI, and play part of Master and Slave.



---

## 15.3 SPI Data Communication

There are four SPI pins: **SCLK**, **MISO**, **MOSI** and  $\overline{\text{SS}}$  which can be switched in 3 groups of pins: [SCLK/P1.5, MISO/P1.4, MOSI/P1.3,  $\overline{\text{SS}}$ /P1.2]; [SCLK\_2/P2.1, MISO\_2/P2.2, MOSI\_2/P2.3,  $\overline{\text{SS}}$ \_2/P2.4]; [SCLK\_3/P4.3, MISO\_3/P4.1, MOSI\_3/P4.0,  $\overline{\text{SS}}$ \_3/P5.4].

**MOSI** ( Master Out Slave In) is directly connected between the Master Device and a Slave Device. The MOSI line is used to transfer data in series from the Master to the Slave. Therefore, it is an output signal from the Master, and an input signal to a Slave. A Byte (8-bit word) is transmitted most significant bit (MSB) first, least significant bit (LSB) last.

**MISO** (Master In Slave Out) is also directly connected between the Slave Device and a Master Device. The MISO line is used to transfer data in series from the Slave to the Master. Therefore, it is an output signal from the Slave, and an input signal to the Master. A Byte (8-bit word) is transmitted most significant bit (MSB) first, least significant bit (LSB) last.

**SCLK** ( SPI Serial Clock) is used to synchronize the data transmission both in and out of the devices through their MOSI and MISO lines. It is driven by the Master for eight clock cycles which allows to exchange one Byte on the serial lines.

SCLK, MOSI and MISO are typically tied together between two or more SPI devices. Data flows from master to slave on the MOSI pin (Master Out / Slave In) and flows from slave to master on the MISO pin (Master In / Slave Out). The SPICLK signal is output in the master mode and is input in the slave mode. If the SPI system is disabled, i.e., SPEN (SPCTL.6) = 0, these pins function as normal I/O pins.

$\overline{\text{SS}}$  is the optional slave select pin. This signal must stay low for any message for a Slave. It is obvious that only one Master ( $\overline{\text{SS}}$  high level) can drive the network. In a typical configuration, an SPI master asserts one of its port pins to select one SPI device as the current slave.

An SPI slave device uses its /SS pin to determine whether it is selected. The  $\overline{\text{SS}}$  is ignored if any of the following conditions are true:

- If the SPI system is disabled, i.e. SPEN (SPCTL.6) = 0 (reset value).
- If the SPI is configured as a master, i.e., MSTR (SPCTL.4) = 1, and P1.2 ( $\overline{\text{SS}}$ ) is configured as an output.
- If the /SS pin is ignored, i.e. SSIG (SPCTL.7) bit = 1, this pin is configured for port functions.

Note that even if the SPI is configured as a master (MSTR=1), it can still be converted to a slave by driving the  $\overline{\text{SS}}$  pin low (if SSIG=0 and P1.2/ $\overline{\text{SS}}$  is set to input). Should this happen, the SPIF bit (SPSTAT.7) will be set.

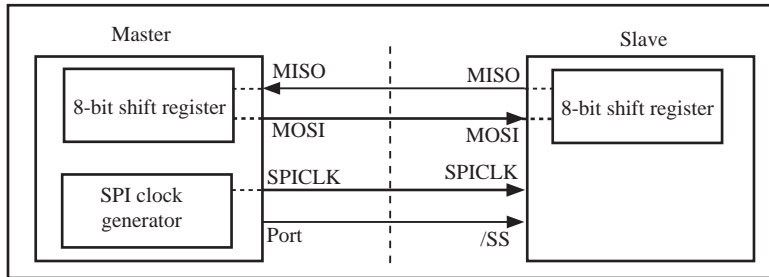
---

### 15.3.1 SPI Data Communication Modes

There are three modes of SPI data communication : single master — single slave, dual devices configuration (both can be a master or slave) and single master — multiple slaves.

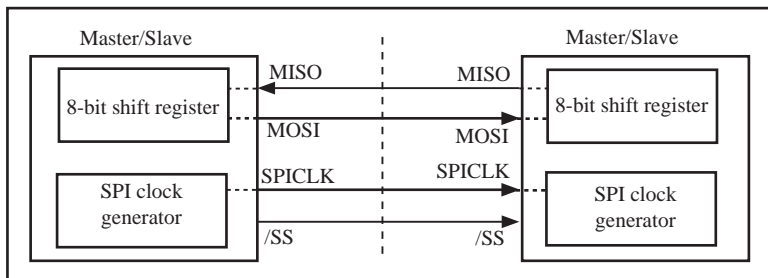
For the master: any port pin, including P1.2 ( $\overline{SS}$ ), can be used to drive the  $\overline{SS}$  pin of the slave.

For the slave: SSIG is '0', and  $\overline{SS}$  pin is used to determine whether it is selected



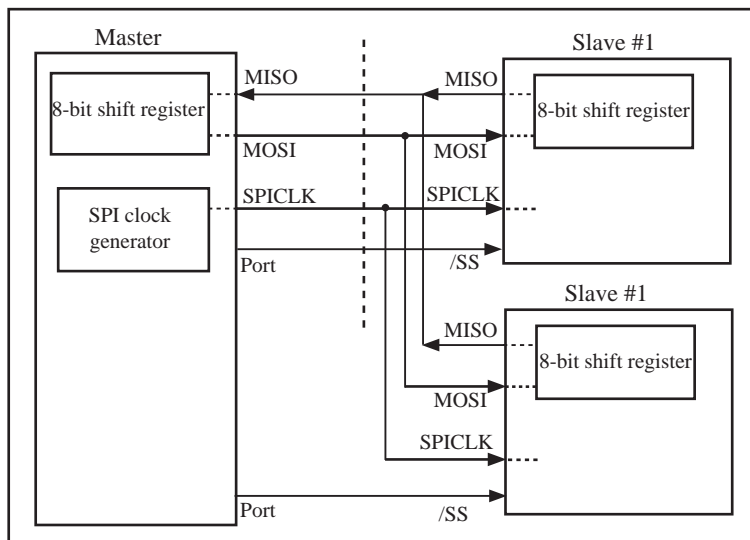
SPI single master — single slave configuration

Two devices are connected to each other and either device can be a master or a slave. When no SPI operation is occurring, both can be configured as masters with MSTR=1, SSIG=0 and P1.2 ( $\overline{SS}$ ) configured in quasi-bidirectional mode. When any device initiates a transfer, it can configure P1.2 as an output and drive it low to force a “mode change to slave” in the other device.



SPI dual device configuration, both can be a master or slave

For the master: any port pin, including P1.2 ( $\overline{SS}$ ), can be used to drive the  $\overline{SS}$  pins of the slaves.  
 For all the slaves: SSIG is '0', and  $\overline{SS}$  pin are used to determine whether it is selected



SPI single master multiple slaves configuration

In SPI, transfers are always initiated by the master. If the SPI is enabled (SPEN=1) and selected as master, any instruction that use SPI data register SPDAT as the destination will start the SPI clock generator and a data transfer. The data will start to appear on MOSI about one half SPI bit-time to one SPI bit-time after it. Before starting the transfer, the master may select a slave by driving the SS pin of the corresponding device low. Data written to the SPDAT register of the master shifted out of MOSI pin of the master to the MOSI pin of the slave. And at the same time the data in SPDAT register of the selected slave is shifted out of MISO pin to the MISO pin of the master. During one byte transfer, data in the master and in the slave is interchanged. After shifting one byte, the transfer completion flag(SPIF) is set and an interrupt will be created if the SPI interrupt is enabled.

If SPEN=1, SSIG=0, SS pin=1 and MSTR=1, the SPI is enabled in master mode. Before the instruction that use SPDAT as the destination register, the master is in idle state and can be selected as slave device by any other master drives the idle master SS pin low. Once this happened, MSTR bit of the idle master is cleared by hardware and changes its state a selected slave. User software should always check the MSTR bit. If this bit is cleared by the mode change of SS pin and the user wants to continue to use the SPI as a master later, the user must set the MSTR bit again, otherwise it will always stay in slave mode.

The SPI is single buffered in transmit direction and double buffered in receive direction. New data for transmission can not be written to the shift register until the previous transaction is complete. The WCOL bit is set to signal data collision when the data register is written during transaction. In this case, the data currently being transmitted will continue to be transmitted, but the new data which causing the collision will be lost. For receiving data, received data is transferred into a internal parallel read data buffer so that the shift register is free to accept a second byte. However, the received byte must be read from the data register(SPDAT) before the next byte has been completely transferred. Otherwise the previous byte is lost. WCOL can be cleared in software by "writing 1 to the bit".

## 15.3.2 SPI Configuration

When SPI data communication, SPEN, SSIG,  $\overline{SS}$ (P1.2) and MSTR jointly control the selection of master and slave.

SPEN	SSIG	$\overline{SS}$ pin P1.2	MSTR	Mode	MISO P1.4	MOSI P1.3	SCLK P1.5	Remark
0	X	P1.2/ $\overline{SS}$	X	SPI disabled	P1.4/ MISO	P1.3/ MOSI	P1.5/ SCLK	SPI is disabled, P1.2/ $\overline{SS}$ , P1.3/ MOSI, P1.4/MISO and P1.5/SCLK are used as general I/O ports
1	0	0	0	Selected slave	output	input	input	Selected as slave
1	0	1	0	Unselected slave	Hi-Z	input	input	Not selected.
1	0	0	1→0	slave (by mode change)	output	input	input	Mode change to slave if P1.2/ $\overline{SS}$ pin is driven low, and MSTR will be cleared to '0' by H/W automatically.
1	0	1	1	Master (idle)	input	Hi-Z	Hi-Z	MOSI and SCLK are at high impedance to avoid bus contention when the Master is idle.
				Master (active)		output	output	MOSI and SCLK are push-pull when the Master is active.
1	1	P1.2/ $\overline{SS}$	0	Slave	output	input	input	
1	1	P1.2/ $\overline{SS}$	1	Master	input	output	output	

"X" means "don't care"

---

### 15.3.3 Additional Considerations for a Slave

When CPHA is 0, SSIG must be 0 and  $\overline{SS}$  pin must be negated and reasserted between each successive serial byte transfer. Note the SPDAT register cannot be written while  $\overline{SS}$  pin is active (low), and the operation is undefined if CPHA is 0 and SSIG is 1.

When CPHA is 1, SSIG may be 0 or 1. If SSIG=0, the  $\overline{SS}$  pin may remain active low between successive transfers (can be tied low at all times). This format is sometimes preferred for use in systems having a single fixed master and a single slave configuration.

### 15.3.4 Additional Considerations for a Master

In SPI, transfers are always initiated by the master. If the SPI is enabled (SPEN=1) and selected as master, writing to the SPI data register (SPDAT) by the master starts the SPI clock generator and data transfer. The data will start to appear on MOSI about one half SPI bit-time to one SPI bit-time after data is written to SPDAT.

Before starting the transfer, the master may select a slave by driving the  $\overline{SS}$  pin of the corresponding device low. Data written to the SPDAT register of the master is shifted out of MOSI pin of the master 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 shifting one byte, the SPI clock generator stops, setting the transfer completion flag (SPIF) and an interrupt will be created if the SPI interrupt is enabled. The two shift registers in the master CPU and slave CPU can be considered as one distributed 16-bit circular shift register. When data is shifted from the master to the slave, data is also shifted in the opposite direction simultaneously. This means that during one shift cycle, data in the master and the slave are interchanged.

### 15.3.5 Mode Change on $\overline{SS}$ -pin

If SPEN=1, SSIG=0, MSTR=1 and  $\overline{SS}$  pin=1, the SPI is enabled in master mode. In this case, another master can drive this pin low to select this device as an SPI slave and start sending data to it. To avoid bus contention, the SPI becomes a slave. As a result of the SPI becoming a slave, the MOSI and SCLK pins are forced to be an input and MISO becomes an output. The SPIF flag in SPSTAT is set, and if the SPI interrupt is enabled, an SPI interrupt will occur. User software should always check the MSTR bit. If this bit is cleared by a slave select and the user wants to continue to use the SPI as a master, the user must set the MSTR bit again, otherwise it will stay in slave mode.

---

## 15.3.6 Write Collision

The SPI is single buffered in the transmit direction and double buffered in the receive direction. New data for transmission can not be written to the shift register until the previous transaction is complete. The WCOL (SPSTAT.6) bit is set to indicate data collision when the data register is written during transmission. In this case, the data currently being transmitted will continue to be transmitted, but the new data, i.e., the one causing the collision, will be lost.

While write collision is detected for both a master or a slave, it is uncommon for a master because the master has full control of the transfer in progress. The slave, however, has no control over when the master will initiate a transfer and therefore collision can occur.

For receiving data, received data is transferred into a parallel read data buffer so that the shift register is free to accept a second character. However, the received character must be read from the Data Register (SPDAT) before the next character has been completely shifted in. Otherwise, the previous data is lost.

WCOL can be cleared in software by writing '1' to the bit.

## 15.3.7 SPI Clock Rate Select

The SPI clock rate selection (in master mode) uses the SPR1 and SPR0 bits in the SPCTL register, as shown in following Table.

SPI Serial Clock Rates

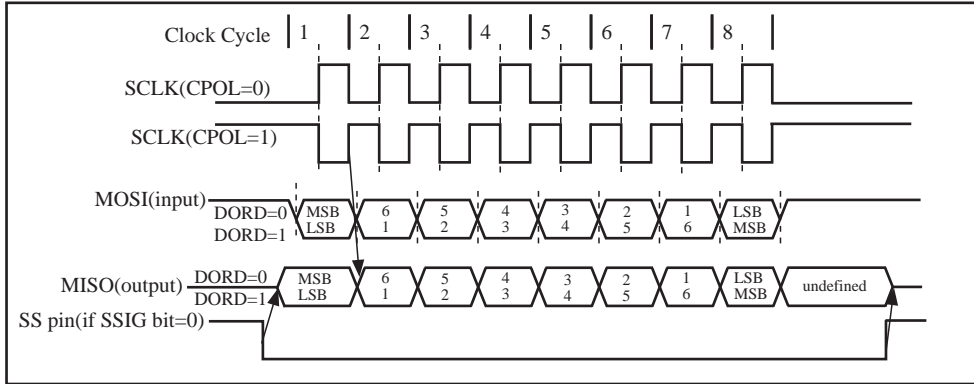
SPR1	SPR0	SPI Clock Rate @ SYSclk = 12MHz	SYSclk divided by
0	0	3 MHz	4
0	1	750 KHz	16
1	0	187.5 KHz	64
1	1	93.75 KHz	128

Where, SYSclk is the system clock

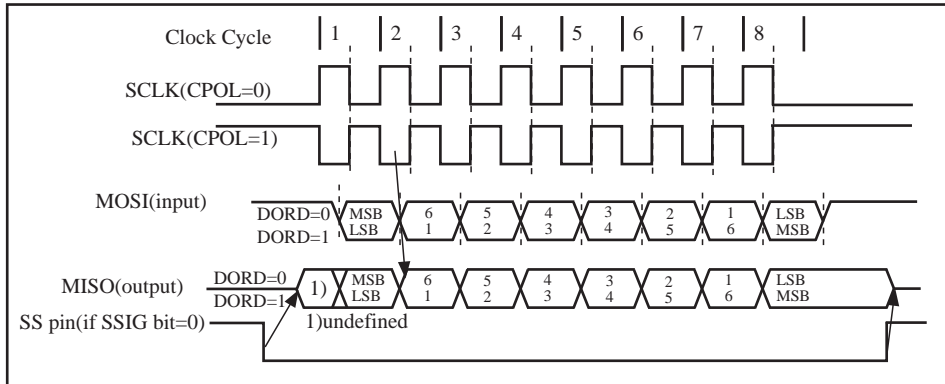
### 15.3.8 SPI Data Mode

CPHA/SPCTL.2 is SPI clock phase select bit which is used to setting the clock edge of Data sample and change. CPOL/SPCTL.3 is used to select SPI clock polarity.

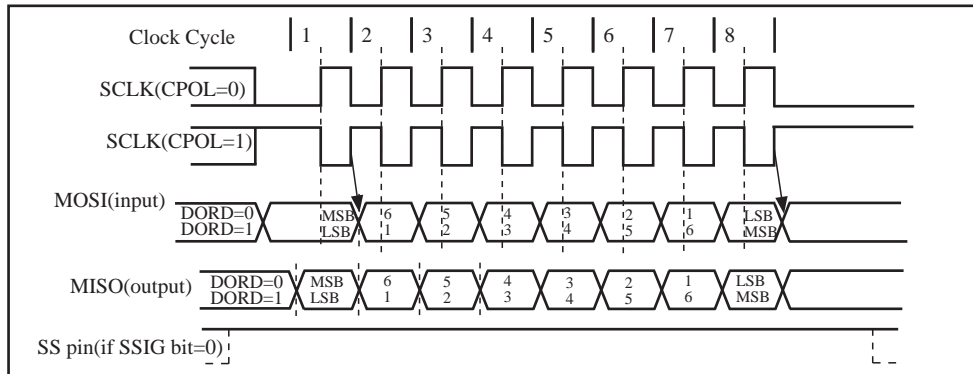
The following are some typical timing diagrams which depend on the value of CPHA/SPCTL.2



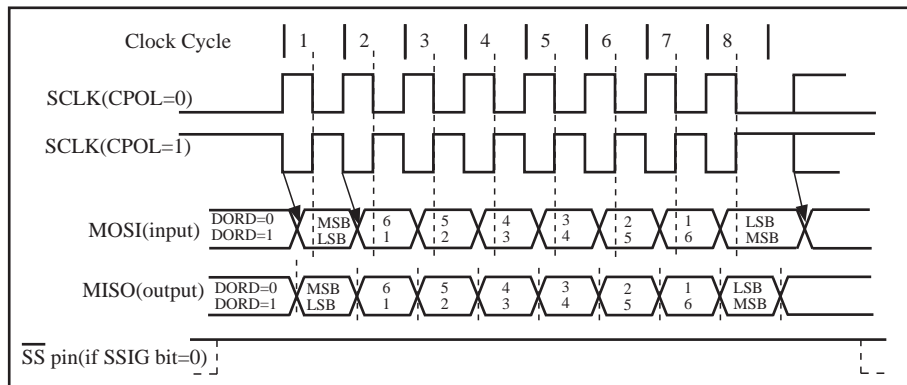
SPI slave transfer format with CPHA=0



SPI slave transfer format with CPHA=1



SPI master transfer format with CPHA=0



SPI master transfer format with CPHA=1

\* The function of SPI can be redirected from P1[2:5] to P2[1:4] pin by setting SPI\_S1 and SPI\_S0 bits in AUXR1/P\_SW1 register.



---

## 15.4 SPI Function Demo Program(Single Master—Single Slave)

### 15.4.1 SPI Function Demo Program using Interrupt(C and ASM)

*The following program, written in C language and assembly language, tests SPI function and applies to SPI single master single slave configuration.*

#### 1. C code listing:

```
;/*-----*/
;/* --- STC 1T Series MCU SPI Demo (1 master and 1 slave) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

#define MASTER //define:master undefine:slave
#define FOSC 18432000L
#define BAUD (256 - FOSC / 32 / 115200)

typedef unsigned char BYTE;
typedef unsigned int WORD;
typedef unsigned long DWORD;

sfr AUXR = 0x8e; //Auxiliary register

sfr SPSTAT = 0xcd; //SPI status register
#define SPIF 0x80 //SPSTAT.7
#define WCOL 0x40 //SPSTAT.6
sfr SPCTL = 0xce; //SPI control register
#define SSIG 0x80 //SPCTL.7
#define SPEN 0x40 //SPCTL.6
#define DORD 0x20 //SPCTL.5
#define MSTR 0x10 //SPCTL.4
#define CPOL 0x08 //SPCTL.3
#define CPHA 0x04 //SPCTL.2
#define SPDHH 0x00 //CPU_CLK/4
#define SPDH 0x01 //CPU_CLK/16
```

---

```

#define SPDL 0x02 //CPU_CLK/64
#define SPDLL 0x03 //CPU_CLK/128
sfr SPDAT = 0xcf; //SPI data register
sbit SPISS = P1^3; //SPI slave select, connect to slave' SS(P1.2) pin
sfr IE2 = 0xAF; //interrupt enable register 2

#define ESPI 0x02 //IE2.1

void InitUart();
void InitSPI();
void SendUart(BYTE dat); //send data to PC
BYTE RecvUart(); //receive data from PC

////////////////////////////////////
void main()
{
    InitUart(); //initial UART
    InitSPI(); //initial SPI
    IE2 |= ESPI;
    EA = 1;

    while (1)
    {
        #ifdef MASTER //for master (receive UART data from PC and send it to slave,
                        //in the meantime receive SPI data from slave and send it to PC)

            ACC = RecvUart();
            SPISS = 0; //pull low slave SS
            SPDAT = ACC; //trigger SPI send
        #endif
    }
}
////////////////////////////////////
void spi_isr() interrupt 9 using 1 //SPI interrupt routine 9 (004BH)
{
    SPSTAT = SPIF | WCOL; //clear SPI status
#ifdef MASTER
    SPISS = 1; //push high slave SS
    SendUart(SPDAT); //return received SPI data
#else
    SPDAT = SPDAT; //for slave (receive SPI data from master and
                  // send previous SPI data to master)
#endif
}
////////////////////////////////////

```

---

---

```

void InitUart()
{
    SCON = 0x5a;           //set UART mode as 8-bit variable baudrate
    TMOD = 0x20;          //timer1 as 8-bit auto reload mode
    AUXR = 0x40;          //timer1 work at 1T mode
    TH1 = TL1 = BAUD;     //115200 bps
    TR1 = 1;
}

```

```

////////////////////////////////////

```

```

void InitSPI()
{
    SPDAT = 0;            //initial SPI data
    SPSTAT = SPIF | WCOL; //clear SPI status
#ifdef MASTER
    SPCTL = SPEN | MSTR;  //master mode
#else
    SPCTL = SPEN;         //slave mode
#endif
}

```

```

////////////////////////////////////

```

```

void SendUart(BYTE dat)
{
    while (!TI);         //wait pre-data sent
    TI = 0;              //clear TI flag
    SBUF = dat;          //send current data
}

```

```

////////////////////////////////////

```

```

BYTE RecvUart()
{
    while (!RI);         //wait receive complete
    RI = 0;              //clear RI flag
    return SBUF;         //return receive data
}

```

---

## 2. Assembly code listing:

```
;/*-----*/
;/* --- STC 1T Series MCU SPI Demo (1 master and 1 slave) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

//#define MASTER //define:master undefine:slave

AUXR DATA 08EH ;Auxiliary register
SPSTAT DATA 0CDH ;SPI status register
SPIF EQU 080H ;SPSTAT.7
WCOL EQU 040H ;SPSTAT.6
SPCTL DATA 0CEH ;SPI control register
SSIG EQU 080H ;SPCTL.7
SPEN EQU 040H ;SPCTL.6
DORD EQU 020H ;SPCTL.5
MSTR EQU 010H ;SPCTL.4
CPOL EQU 008H ;SPCTL.3
CPHA EQU 004H ;SPCTL.2
SPDHH EQU 000H ;CPU_CLK/4
SPDH EQU 001H ;CPU_CLK/16
SPDL EQU 002H ;CPU_CLK/64
SPDLL EQU 003H ;CPU_CLK/128
SPDAT DATA 0CFH ;SPI data register
SPISS BIT P1.3 ;SPI slave select, connect to slave' SS(P1.2) pin

IE2 EQU 0AFH ;interrupt enable rgister 2
ESPI EQU 02H ;IE2.1

;////////////////////////////////////

ORG 0000H
LJMP RESET
```

---

```

        ORG    004BH                                ;SPI interrupt routine
SPI_ISR:
        PUSH  ACC
        PUSH  PSW
        MOV   SPSTAT, #SPIF | WCOL                ;clear SPI status
#ifdef MASTER
        SETB  SPISS                                ;push high slave SS
        MOV   A,    SPDAT                          ;return received SPI data
        LCALL SEND_UART
#else
                                                ;for salve (receive SPI data from master and
        MOV   SPDAT, SPDAT                          ;send previous SPI data to master)
#endif
        POP   PSW
        POP   ACC
        RETI

;////////////////////////////////////

        ORG    0100H
RESET:
        LCALL INIT_UART                          ;initial UART
        LCALL INIT_SPI                           ;initial SPI
        ORL   IE2,    #ESPI
        SETB  EA
MAIN:
#ifdef MASTER
                                                //for master (receive UART data from PC and send it to slave,
        LCALL RECV_UART                          ; in the meantime receive SPI data from slave and send it to PC)
        CLR  SPISS                                ;pull low slave SS
        MOV  SPDAT, A                             ;trigger SPI send
#endif
        SJMP MAIN

;////////////////////////////////////

INIT_UART:
        MOV  SCON, #5AH                          ;set UART mode as 8-bit variable baudrate
        MOV  TMOD, #20H                          ;timer1 as 8-bit auto reload mode
        MOV  AUXR, #40H                          ;timer1 work at 1T mode
        MOV  TL1, #0FBH                          ;115200 bps(256 - 18432000 / 32 / 115200)
        MOV  TH1, #0FBH
        SETB TR1
        RET

;////////////////////////////////////

```

---

---

```

INIT_SPI:
    MOV    SPDAT,  #0                ;initial SPI data
    MOV    SPSTAT, #SPIF | WCOL      ;clear SPI status
#ifdef MASTER
    MOV    SPCTL,  #SPEN | MSTR      ;master mode
#else
    MOV    SPCTL,  #SPEN              ;slave mode
#endif
    RET

;////////////////////////////////////

SEND_UART:
    JNB    TI,      $                ;wait pre-data sent
    CLR    TI
    MOV    SBUF,   A                ;send current data
    RET

;////////////////////////////////////

RCV_UART:
    JNB    RI,$                ;wait receive complete
    CLR    RI                ;clear RI flag
    MOV    A,      SBUF        ;return receive data
    RET
    RET

;////////////////////////////////////

    END

```

---

## 15.4.2 SPI Function Demo Programs using Polling mode (C and ASM)

### 1. C code listing:

```
/*-----*/
/* --- STC 1T Series MCU SPI Demo (1 master and 1 slave) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

#define MASTER //define:master undefine:slave
#define FOSC 18432000L
#define BAUD (256 - FOSC / 32 / 115200)

typedef unsigned char BYTE;
typedef unsigned int WORD;
typedef unsigned long DWORD;

sfr AUXR = 0x8e; //Auxiliary register
sfr SPSTAT = 0xcd; //SPI status register
#define SPIF 0x80 //SPSTAT.7
#define WCOL 0x40 //SPSTAT.6
sfr SPCTL = 0xce; //SPI control register
#define SSIG 0x80 //SPCTL.7
#define SPEN 0x40 //SPCTL.6
#define DORD 0x20 //SPCTL.5
#define MSTR 0x10 //SPCTL.4
#define CPOL 0x08 //SPCTL.3
#define CPHA 0x04 //SPCTL.2
#define SPDHH 0x00 //CPU_CLK/4
#define SPDH 0x01 //CPU_CLK/16
#define SPDL 0x02 //CPU_CLK/64
#define SPDLL 0x03 //CPU_CLK/128
sfr SPDAT = 0xcf; //SPI data register
sbit SPISS = P1^3; //SPI slave select, connect to slave' SS(P1.2) pin

void InitUart();
void InitSPI();
```

---

```

void    SendUart(BYTE dat);           //send data to PC
BYTE    RecvUart();                  //receive data from PC
BYTE    SPISwap(BYTE dat);          //swap SPI data between master
////////////////////////////////////

void main()
{
    InitUart();                      //initial UART
    InitSPI();                       //initial SPI

    while (1)
    {
#ifdef MASTER           //for master (receive UART data from PC and send it to slave,
                        // in the meantime receive SPI data from slave and send it to PC)
        SendUart(SPISwap(RecvUart()));
#else
        ACC = SPISwap(ACC);          //for slave (receive SPI data from master and
                                    // send previous SPI data to master)
#endif
    }
}

////////////////////////////////////

void InitUart()
{
    SCON = 0x5a;                    //set UART mode as 8-bit variable baudrate
    TMOD = 0x20;                    //timer1 as 8-bit auto reload mode
    AUXR = 0x40;                    //timer1 work at 1T mode
    TH1 = TL1 = BAUD;              //115200 bps
    TR1 = 1;
}

////////////////////////////////////

void InitSPI()
{
    SPDAT = 0;                      //initial SPI data
    SPSTAT = SPIF | WCOL;          //clear SPI status
#ifdef MASTER
    SPCTL = SPEN | MSTR;          //master mode
#else
    SPCTL = SPEN;                //slave mode
#endif
}

```

---



---

```
////////////////////////////////////
```

```
void SendUart(BYTE dat)
```

```
{
    while (!TI);           //wait pre-data sent
    TI = 0;                //clear TI flag
    SBUF = dat;           //send current data
}
```

```
////////////////////////////////////
```

```
BYTE RecvUart()
```

```
{
    while (!RI);          //wait receive complete
    RI = 0;               //clear RI flag
    return SBUF;         //return receive data
}
```

```
////////////////////////////////////
```

```
BYTE SPISwap(BYTE dat)
```

```
{
#ifdef MASTER
    SPISS = 0;           //pull low slave SS
#endif
    SPDAT = dat;        //trigger SPI send
    while (!(SPSTAT & SPIF)); //wait send complete
    SPSTAT = SPIF | WCOL; //clear SPI status
#ifdef MASTER
    SPISS = 1;           //push high slave SS
#endif
    return SPDAT;       //return received SPI data
}
```

---

## 2. Assembly code listing:

```
;/*-----*/
;/* --- STC 1T Series MCU SPI Demo (1 master and 1 slave) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#define MASTER //define:master undefine:slave

AUXR DATA 08EH ;Auxiliary register
SPSTAT DATA 0CDH ;SPI status register
SPIF EQU 080H ;SPSTAT.7
WCOL EQU 040H ;SPSTAT.6
SPCTL DATA 0CEH ;SPI control register
SSIG EQU 080H ;SPCTL.7
SPEN EQU 040H ;SPCTL.6
DORD EQU 020H ;SPCTL.5
MSTR EQU 010H ;SPCTL.4
CPOL EQU 008H ;SPCTL.3
CPHA EQU 004H ;SPCTL.2
SPDHH EQU 000H ;CPU_CLK/4
SPDH EQU 001H ;CPU_CLK/16
SPDL EQU 002H ;CPU_CLK/64
SPDLL EQU 003H ;CPU_CLK/128
SPDAT DATA 0CFH ;SPI data register
SPISS BIT P1.3 ;SPI slave select, connect to slave' SS(P1.2) pin

;////////////////////////////////////

ORG 0000H
LJMP RESET
ORG 0100H
RESET:
LCALL INIT_UART ;initial UART
LCALL INIT_SPI ;initial SPI
```

---

```

MAIN:
#ifdef MASTE //for master (receive UART data from PC and send it to slave, in the meantime
    LCALL RECV_UART ; receive SPI data from slave and send it to PC)
    LCALL SPI_SWAP
    LCALL SEND_UART
#else //for salve (receive SPI data from master and
    LCALL SPI_SWAP ; send previous SPI data to master)
#endif
    SJMP MAIN
;////////////////////////////////////

INIT_UART:
    MOV SCON, #5AH ;set UART mode as 8-bit variable baudrate
    MOV TMOD, #20H ;timer1 as 8-bit auto reload mode
    MOV AUXR, #40H ;timer1 work at 1T mode
    MOV TL1, #0FBH ;115200 bps(256 - 18432000 / 32 / 115200)
    MOV TH1, #0FBH
    SETB TR1
    RET
;////////////////////////////////////

INIT_SPI:
    MOV SPDAT, #0 ;initial SPI data
    MOV SPSTAT, #SPIF | WCOL ;clear SPI status
#ifdef MASTER
    MOV SPCTL, #SPEN | MSTR ;master mode
#else
    MOV SPCTL, #SPEN ;slave mode
#endif
    RET
;////////////////////////////////////

SEND_UART:
    JNB TI, $ ;wait pre-data sent
    CLR TI ;clear TI flag
    MOV SBUF, A ;send current data
    RET
;////////////////////////////////////

```

---

---

```

RECV_UART:
    JNB    RI,    $                ;wait receive complete
    CLR    RI                ;clear RI flag
    MOV    A,    SBUF            ;return receive data
    RET
    RET

;////////////////////////////////////

SPI_SWAP:
#ifdef MASTER
    CLR    SPISS                ;pull low slave SS
#endif
    MOV    SPDAT, A                ;trigger SPI send
WAIT:
    MOV    A,    SPSTAT
    JNB    ACC.7, WAIT            ;wait send complete
    MOV    SPSTAT, #SPIF | WCOL    ;clear SPI status
#ifdef MASTER
    SETB   SPISS                ;push high slave SS
#endif
    MOV    A,    SPDAT            ;return received SPI data
    RET

;////////////////////////////////////

    END

```

---

## 15.5 SPI Function Demo Program(Each other as Master-Slave)

### 15.5.1 SPI Function Demo Programs using Interrupts (C and ASM)

#### 1. C code listing:

```
;/*-----*/
;/* --- STC 1T Series MCU SPI Demo (Each other as the master-slave) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

#define FOSC      18432000L
#define BAUD      (256 - FOSC / 32 / 115200)

typedef unsigned char    BYTE;
typedef unsigned int     WORD;
typedef unsigned long    DWORD;

sfr    AUXR    = 0x8e;                //Auxiliary register

sfr    SPSTAT = 0xcd;                //SPI status register
#define SPIF    0x80                //SPSTAT.7
#define WCOL    0x40                //SPSTAT.6
sfr    SPCTL   = 0xce;                //SPI control register
#define SSIG    0x80                //SPCTL.7
#define SPEN    0x40                //SPCTL.6
#define DORD    0x20                //SPCTL.5
#define MSTR    0x10                //SPCTL.4
#define CPOL    0x08                //SPCTL.3
#define CPHA    0x04                //SPCTL.2
#define SPDHH   0x00                //CPU_CLK/4
#define SPDH    0x01                //CPU_CLK/16
#define SPDL    0x02                //CPU_CLK/64
#define SPDLL   0x03                //CPU_CLK/128
sfr    SPDAT   = 0xcf;                //SPI data register
sbit   SPISS   = P1^3;                //SPI slave select, connect to other MCU's SS(P1.2) pin
```

---

```

sfr      IE2      =      0xAF;                //interrupt enable register 2
#define   ESPI     0x02                        //IE2.1
void InitUart();
void InitSPI();
void SendUart(BYTE dat);                     //send data to PC
BYTE RecvUart();                             //receive data from PC

bit MSSEL;                                   //1: master 0:slave

////////////////////////////////////
void main()
{
    InitUart();          //initial UART
    InitSPI();          //initial SPI
    IE2 |= ESPI;
    EA = 1;

    while (1)
    {
        if (RI)
        {
            SPCTL = SPEN | MSTR;           //set as master
            MSSEL = 1;
            ACC = RecvUart();
            SPISS = 0;                     //pull low slave SS
            SPDAT = ACC;                   //trigger SPI send
        }
    }
}

////////////////////////////////////
void spi_isr() interrupt 9 using 1          //SPI interrupt routine 9 (004BH)
{
    SPSTAT = SPIF | WCOL;                  //clear SPI status
    if (MSSEL)
    {
        SPCTL = SPEN;                     //reset as slave
        MSSEL = 0;
        SPISS = 1;                         //push high slave SS
        SendUart(SPDAT);                   //return received SPI data
    }
    else
    {
        SPDAT = SPDAT;                     //for slave (receive SPI data from master and
                                           //      send previous SPI data to master)
    }
}

////////////////////////////////////

```

---

```
void InitUart()
{
    SCON = 0x5a;           //set UART mode as 8-bit variable baudrate
    TMOD = 0x20;          //timer1 as 8-bit auto reload mode
    AUXR = 0x40;          //timer1 work at 1T mode
    TH1 = TL1 = BAUD;     //115200 bps
    TR1 = 1;
}
```

```
////////////////////////////////////
```

```
void InitSPI()
{
    SPDAT = 0;            //initial SPI data
    SPSTAT = SPIF | WCOL; //clear SPI status
    SPCTL = SPEN;        //slave mode
}
```

```
////////////////////////////////////
```

```
void SendUart(BYTE dat)
{
    while (!TI);         //wait pre-data sent
    TI = 0;              //clear TI flag
    SBUF = dat;          //send current data
}
```

```
////////////////////////////////////
```

```
BYTE RecvUart()
{
    while (!RI);        //wait receive complete
    RI = 0;             //clear RI flag
    return SBUF;        //return receive data
}
```

---

## 2. Assembly code listing:

```
;/*-----*/
;/* --- STC 1T Series MCU SPI Demo (Each other as the master-slave) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
;/*-----*/

//suppose the frequency of test chip is 18.432MHz

AUXR DATA 08EH ;Auxiliary register
SPSTAT DATA 0CDH ;SPI status register
SPIF EQU 080H ;SPSTAT.7
WCOL EQU 040H ;SPSTAT.6
SPCTL DATA 0CEH ;SPI control register
SSIG EQU 080H ;SPCTL.7
SPEN EQU 040H ;SPCTL.6
DORD EQU 020H ;SPCTL.5
MSTR EQU 010H ;SPCTL.4
CPOL EQU 008H ;SPCTL.3
CPHA EQU 004H ;SPCTL.2
SPDHH EQU 000H ;CPU_CLK/4
SPDH EQU 001H ;CPU_CLK/16
SPDL EQU 002H ;CPU_CLK/64
SPDLL EQU 003H ;CPU_CLK/128
SPDAT DATA 0CFH ;SPI data register
SPISS BIT P1.3 ;SPI slave select, connect to other MCU's SS(P1.2) pin

IE2 EQU 0AFH ;interrupt enable register 2
ESPI EQU 02H ;IE2.1

MSSEL BIT 20H.0 ;1: master 0:slave

;////////////////////////////////////

ORG 0000H
LJMP RESET

ORG 004BH ;SPI interrupt routine
SPI_ISR:
PUSH ACC
PUSH PSW
```



---

```

        MOV    SPSTAT, #SPIF | WCOL           ;clear SPI status
        JBC    MSSEL, MASTER_SEND
SLAVE_RECV:
                                           ;for salve (receive SPI data from master and
                                           ;      send previous SPI data to master)
        MOV    SPDAT, SPDAT
        JMP    SPI_EXIT
MASTER_SEND:
        SETB   SPISS                         ;push high slave SS
        MOV    SPCTL, #SPEN                  ;      ;reset as slave
        MOV    A,    SPDAT                   ;return received SPI data
        LCALL  SEND_UART
SPI_EXIT:
        POP    PSW
        POP    ACC
        RETI

```

```

;////////////////////////////////////

```

```

        ORG    0100H
RESET:
        MOV    SP,#3FH
        LCALL  INIT_UART                     ;initial UART
        LCALL  INIT_SPI                      ;initial SPI
        ORL    IE2,#ESPI
        SETB   EA
MAIN:
        JNB    RI,    $                      ;wait UART data
        MOV    SPCTL, #SPEN | MSTR          ; ;set as master
        SETB   MSSEL
        LCALL  RECV_UART                     ;receive UART data from PC
        CLR    SPISS                        ;pull low slave SS
        MOV    SPDAT,A                      ;trigger SPI send
        SJMP   MAIN

```

```

;////////////////////////////////////

```

```

INIT_UART:
        MOV    SCON, #5AH                   ;set UART mode as 8-bit variable baudrate
        MOV    TMOD, #20H                   ;timer1 as 8-bit auto reload mode
        MOV    AUXR ,#40H                   ;timer1 work at 1T mode
        MOV    TL1,  #0FBH                  ;115200 bps(256 - 18432000 / 32 / 115200)
        MOV    TH1,  #0FBH
        SETB   TR1
        RET

```

---

;/;;

INIT\_SPI:

```
MOV    SPDAT, #0           ;initial SPI data
MOV    SPSTAT, #SPIF | WCOL ;clear SPI status
MOV    SPCTL, #SPEN       ;slave mode
RET
```

;/;;

SEND\_UART:

```
JNB    TI,    $           ;wait pre-data sent
CLR    TI
MOV    SBUF,  A           ;send current data
RET
```

;/;;

RECV\_UART:

```
JNB    RI,    $           ;wait receive complete
CLR    RI
MOV    A,     SBUF        ;return receive data
RET
RET
```

;/;;

END

---

## 15.5.2 SPI Function Demo Programs using Polling

### 1. C code listing:

```
/*-----*/
/* --- STC12C5Axx Series MCU SPI Demo(Each other as the master-slave) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

#include "reg51.h"

#define FOSC          18432000L
#define BAUD          (256 - FOSC / 32 / 115200)

typedef unsigned char  BYTE;
typedef unsigned int   WORD;
typedef unsigned long  DWORD;

sfr    AUXR    =    0x8e;           //Auxiliary register

sfr    SPSTAT  =    0xcd;           //SPI status register
#define SPIF    0x80               //SPSTAT.7
#define WCOL    0x40               //SPSTAT.6
sfr    SPCTL   =    0xce;           //SPI control register
#define SSIG    0x80               //SPCTL.7
#define SPEN    0x40               //SPCTL.6
#define DORD    0x20               //SPCTL.5
#define MSTR    0x10               //SPCTL.4
#define CPOL    0x08               //SPCTL.3
#define CPHA    0x04               //SPCTL.2
#define SPDHH   0x00               //CPU_CLK/4
#define SPDH    0x01               //CPU_CLK/16
#define SPDL    0x02               //CPU_CLK/64
#define SPDLL   0x03               //CPU_CLK/128
sfr    SPDAT   =    0xcf;           //SPI data register
sbit   SPISS   =    P1^3;           //SPI slave select, connect to slave' SS(P1.2) pin

void    InitUart();
void    InitSPI();
```

---

```

void    SendUart(BYTE dat);                //send data to PC
BYTE    RecvUart();                        //receive data from PC
BYTE    SPISwap(BYTE dat);                //swap SPI data between master

////////////////////////////////////

void main()
{
    InitUart();                            //initial UART
    InitSPI();                              //initial SPI

    while (1)
    {
        if (RI)
        {
            SPCTL = SPEN | MSTR;            //set as master
            SendUart(SPISwap(RecvUart()));
            SPCTL = SPEN;                  //reset as slave
        }
        if (SPSTAT & SPIF)
        {
            SPSTAT = SPIF | WCOL; //clear SPI status
            SPDAT = SPDAT;            //mov data from receive buffer to send buffer
        }
    }
}

////////////////////////////////////

void InitUart()
{
    SCON = 0x5a;                            //set UART mode as 8-bit variable baudrate
    TMOD = 0x20;                            //timer1 as 8-bit auto reload mode
    AUXR = 0x40;                            //timer1 work at 1T mode
    TH1 = TL1 = BAUD;                        //115200 bps
    TR1 = 1;
}

////////////////////////////////////

void InitSPI()
{
    SPDAT = 0;                              //initial SPI data
    SPSTAT = SPIF | WCOL;                  //clear SPI status
    SPCTL = SPEN;                          //slave mode
}

```

---

---

////////////////////////////////////

```
void SendUart(BYTE dat)
{
    while (!TI);           //wait pre-data sent
    TI = 0;                //clear TI flag
    SBUF = dat;           //send current data
}
```

////////////////////////////////////

```
BYTE RecvUart()
{
    while (!RI);          //wait receive complete
    RI = 0;               //clear RI flag
    return SBUF;         //return receive data
}
```

////////////////////////////////////

```
BYTE SPISwap(BYTE dat)
{
    SPISS = 0;           //pull low slave SS
    SPDAT = dat;        //trigger SPI send
    while (!(SPSTAT & SPIF)); //wait send complete
    SPSTAT = SPIF | WCOL; //clear SPI status
    SPISS = 1;         //push high slave SS
    return SPDAT;      //return received SPI data
}
```

---

## 2. Assembly code listing:

```
/*-----*/
/* --- STC12C5Axx Series MCU SPI Demo(Each other as the master-slave) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*--- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*--- And only contain < reg51.h > as header file -----*/
/*-----*/

//suppose the frequency of test chip is 18.432MHz

AUXR DATA 08EH ;Auxiliary register
SPSTAT DATA 0CDH ;SPI status register
SPIF EQU 080H ;SPSTAT.7
WCOL EQU 040H ;SPSTAT.6
SPCTL DATA 0CEH ;SPI control register
SSIG EQU 080H ;SPCTL.7
SPEN EQU 040H ;SPCTL.6
DORD EQU 020H ;SPCTL.5
MSTR EQU 010H ;SPCTL.4
CPOL EQU 008H ;SPCTL.3
CPHA EQU 004H ;SPCTL.2
SPDHH EQU 000H ;CPU_CLK/4
SPDH EQU 001H ;CPU_CLK/16
SPDL EQU 002H ;CPU_CLK/64
SPDLL EQU 003H ;CPU_CLK/128
SPDAT DATA 0CFH ;SPI data register
SPISS BIT P1.3 ;SPI slave select, connect to slave' SS(P1.4) pin

;////////////////////////////////////

ORG 0000H
LJMP RESET
ORG 0100H
RESET:
LCALL INIT_UART ;initial UART
LCALL INIT_SPI ;initial SPI

MAIN:
JB RI, MASTER_MODE
```

---

SLAVE\_MODE:

```
MOV A, SPSTAT
JNB ACC.7, MAIN
MOV SPSTAT, #SPIF | WCOL ;clear SPI status
MOV SPDAT, SPDAT ;return received SPI data
SJMP MAIN
```

MASTER\_MODE:

```
MOV SPCTL, #SPEN | MSTR ;set as master
LCALL RECV_UART ;receive UART data from PC
LCALL SPI_SWAP ;send it to slave, in the meantime, receive SPI data from slave
LCALL SEND_UART ;send SPI data to PC
MOV SPCTL, #SPEN ; ;reset as slave
SJMP MAIN
```

;;

INIT\_UART:

```
MOV SCON, #5AH ;set UART mode as 8-bit variable baudrate
MOV TMOD, #20H ;timer1 as 8-bit auto reload mode
MOV AUXR, #40H ;timer1 work at 1T mode
MOV TL1, #0FBH ;115200 bps(256 - 18432000 / 32 / 115200)
MOV TH1, #0FBH
SETB TR1
RET
```

;;

INIT\_SPI:

```
MOV SPDAT, #0 ;initial SPI data
MOV SPSTAT, #SPIF | WCOL ;clear SPI status
MOV SPCTL, #SPEN ;slave mode
RET
```

;;

SEND\_UART:

```
JNB TI, $ ;wait pre-data sent
CLR TI ;clear TI flag
MOV SBUF, A ;send current data
RET
```

---

;/;;

RECV\_UART:

JNB RI, \$ ;wait receive complete  
CLR RI ;clear RI flag  
MOV A, SBUF ;return receive data  
RET  
RET

;/;;

SPI\_SWAP:

CLR SPISS ;pull low slave SS  
MOV SPDAT, A ;trigger SPI send

WAIT:

MOV A, SPSTAT  
JNB ACC.7, WAIT ;wait send complete  
MOV SPSTAT, #SPIF | WCOL ;clear SPI status  
SETB SPISS ;push high slave SS  
MOV A, SPDAT ;return received SPI data  
RET

;/;;

END





---

```

;Define const
MASTER_SLAVE EQU 0 ;Master MCU
;MASTER_SLAVE EQU 1 ;Slave #1 MCU
;MASTER_SLAVE EQU 2 ;Slave #2 MCU

;RELOAD_8BIT_DATA EQU 0FFH ;56700@22.1184MHz
RELOAD_8BIT_DATA EQU 0FBH ;9600@18.432MHz
;RELOAD_8BIT_DATA EQU 0F6H ;4800@18.432MHz
;RELOAD_8BIT_DATA EQU 0FFH ;28800@11.0592MHz
;
;Define SFR
AUXR EQU 8EH ;Auxiliary register
SPCTL EQU 85H ;SPI control register
SPSTAT EQU 84H ;SPI status register
SPDAT EQU 86H ;SPI data register
EADC_SPI EQU IE.5 ;SPI interrupt enable bit

;Define SPI function pin
SCLK EQU P1.7 ;SPI clock pin
MISO EQU P1.6 ;SPI master input/slave output pin
MOSI EQU P1.5 ;SPI master output/slave input pin
SS EQU P1.4 ;SPI slave select pin
Slave1_SS EQU P1.2 ;slave #1 MCU select pin
Slave2_SS EQU P1.3 ;slave #2 MCU select pin

LED_MCU_START EQU P3.4 ;MCU work LED

;Define user variable
Flags EQU 20H ;user flag
SPI_Receive EQU Falgs.0 ;SPI receive flag
T0_10mS_count EQU 30H ;10ms counter
SPI_buffer EQU 31H ;SPI revecie buffer
;-----
ORG 0000H
LJMP MAIN
ORG 000BH
LJMP timer0_Routine ;timer0 interrupt routine
ORG 002BH
LJMP ADC_SPI_Interrupt_Routine ;SPI interrupt routine
;-----
ORG 0080H
MAIN:
CLR LED_MCU_START ;work led on
MOV SP,#7FH ;initial SP
ACALL Initial_System ;system initial
if MASTER_SLAVE == 0
CLR Slave1_SS ;select slave #1 MCU

```

---

---

```

Check_RS232:
    JNB     RI,Master_Check_SPI      ;check UART receive
    ACALL  Get_Byte_From_RS232      ;load UART data to ACC
;
;
;
;
    ACALL  SPI_Send_Byte      ;send data in ACC to SPI slave
    SJMP   Check_RS232

Master_Check_SPI:
    JNB     SPI_Receive,Check_RS232 ;check SPI receive
    MOV     A,SPI_buffer      ;load SPI data to ACC
    CLR     SPI_Recevie      ;clear SPI receive flag
    ACALL  SPI_Send_Byte      ; send data in ACC to SPI slave
    SJMP   Check_RS232

else
Slave_Check_SPI:
    JNB     SPI_Receive,Slave_Check_SPI ;check SPI receive
    MOV     A,SPI_buffer      ;load SPI data to ACC
    CLR     SPI_Receive      ;clear SPI receive flag
if MASTER_SLAVE == 2
    ADD     A,#1      ;value +1 on slave #2 MCU
endif
    MOV     SPDAT,A;save data into SPDAT
    SJMP   Slave_Check_SPI

endif

;-----
if MASTER_SLAVE == 0
timer0_Routine:
    PUSH   PSW
    PUSH   ACC
    MOV    TH0,#0C4H      ;reload timer0 10ms value
    INC    T0_10mS_count ;10ms counter
    MOV    A,#200      ;count 200 times
    CLR    C
    SUBB   A,T0_10mS_count
    JNC    timer0_Exit
    CPL    SLAVE1_SS      ;switch slave
    CPL    SLAVE2_SS
    MOV    T0_10mS_count,#0;reset counter
timer0_Exit:
    POP    ACC
    POP    PSW
    RETI

else
timer0_Routine:
    RETI
endif
;-----

```

---

---

ADC\_SPI\_Interrupt\_Routine:

```
MOV    SPDAT,#0C0H    ;clear SPIF and WCOL flag
MOV    A,SPDAT        ;save SPI received data
MOV    SPI_buffer,A
SETB   SPI_Receive    ;set SPI receive flag
RETI
```

-----

Initial\_System:

```
ACALL  Initial_Uart    ;initial UART sfr
ACALL  Initial_SPI     ;initial SPI sfr
SETB   TR0             ;start timer0
SETB   ET0             ;enable timer0 interrupt
MOV    Flags,#0       ;initial flag
SETB   EA              ;enable global interrupt flag
RET
```

-----

Initial\_Uart:

```
MOV    SCON,#50H      ;set UART as 8-bit variable mode
MOV    TMOD,#21H      ;set timer mode
MOV    TH1,#RELOAD_8BIT_DATA ;set UART baudrate
MOV    TL1,#RELOAD_8BIT_DATA
MOV    PCON,#80H      ;baudrate * 2
ORL    AUXR,#40H      ;1T mode
SETB   TR1            ;timer1 start
RET
```

-----

Initial\_SPI:

if MASTER\_SLAVE == 0

```
MOV    SPCTL,#11111100B ;master mode
```

else

```
MOV    SPCTL,#01101100B ;slave mode
```

endif

```
MOV    SPSTAT,#11000000B ;clear SPI flag
ORL    AUXR,#08H         ;AUXR.3(ESPI) = 1
SETB   EADC_SPI         ;enable SPI interrupt
RET
```

-----

RS232\_Send\_Byte:

```
CLR    TI              ;ready send
MOV    SBUF,A          ;write data to TX buffer
JNB    TI,$            ;wait send completed
CLR    TI              ;clear TI flag
RET
```

-----

SPI\_Send\_Byte:

```
CLR    EADC_SPI        ;disable SPI interrupt
MOV    SPDAT,A;write data to SPI data register
```

---

```

SPI_Send_Byte_Wait:
    MOV    A,SPSTAT      ;check SPI status
    ANL    A,#80H
    JZ     SPI_Send_Byte_Wait    ;wait SPI send complete
    SETB   EADC_SPI     ;enable SPI interrupt
    RET

;-----
Get_Byte_From_RS232:
    MOV    A,SBUF      ;load data to ACC
    CLR    RI          ;clear UART receive flag
    RET

;-----
    END

```

## 2. C listing code:

```

/*-----*/
/* --- STC 1T Series MCU SPI ASM Demo -----*/
/* If you want to use the program or the program referenced in the */
/* article, please specify in which data and procedures from STC */
/*-----*/

typedef unsigned char INT8U;
typedef unsigned int INT16U;
typedef unsigned long INT32U;

#include "new_8051.h"

//Define const
#define SPI_INTERRUPT_VECTOR 9
#define TRUE 1
#define FALSE 0
#define MASTER
#define CONFIG_MASTER 0xd0    //master mode
#define CONFIG_SLAVE 0xc0    //slave mode
#define SPIF_WCOL_MASK 0xc0  //SPIF & WCOL mask bit
#define FOSC 1843200
#define BAUD 9600
#define BUF_SIZE 0x20

```

---

```

//Define SFR
sfr SPCTL = 0x44;
sbit LED_MCU_START = P3^4;    //work LED
bit SPI_Receive;             //SPI received flag
bit SPI_status;             //SPI status
INT8U SPI_buffer;           //SPI receive data buffer
INT8U RS232_point;
INT8U ISP_point;
INT8U buffer[BUF_SIZE];
//-----
void Initial_SPI();
void Init_System();
INT8U Get_Byte_From_RS232();
void RS232_Send_Byte(INT8U ch);
void SPI_Send_Byte(INT8U);
void send_buffer_to_PC();
void clear_buffer();
void delay(INT16U d);
void SPI_read_from_slave(INT8U n);
//-----
void main()
{
    INT32U i=0;
    LED_MCU_START = 0;           //work LED on
    Init_System();              //system initial
    SPI_Receive = 0;           //initial user flag
    RS232_point = 0;
    ISP_point = 0;
    clear_buffer();            //empty buffer
#ifdef MASTER
    while (1)
    {
        if (RI)                //check UART RI
        {
            RI = 0;
            if (RS232_point < BUF_SIZE)
            {
                buffer[RS232_point++] = SBUF    //save UART RX data
            }
            i = 65000;          //wait another data
        }
        if (i > 0)
        {
            i--;                //check wait
            if (i == 0)          //send all data at wait end
            {
                if (RS232_point > 0)
                {
                    ISP_point = 0;

```

---

---

```

        SPI_status = 1;    //1:SPI send
        SPDAT = buffer[ISP_point++];    //trigger SPI send action
        while (ISP_point < RS232_point);    //other send in interrupt
    }
    delay(300);
    SPI_read_from_slave(RS232_point);    //read slave data
    send_buffer_to_PC();    //send back to PC
    clear_buffer();
    SPI_Recevie = 0;
    RS232_point = 0;
    ISP_point = 0;
    RI = 0;
    }
}
}
#else
SPI_Recevie = 0;
SPI_status = 0;    //0:SPI receive
RS232_point = 0;
ISP_point = 0;
while (1)
{
    if (SPI_Recevie)
    {
        SPI_Recevie = 0;
        i = 10000;    //wait another data
    }
    if (i > 0)
    {
        i--;
        if (i == 0)
        {
            if (!SPI_status)    //SPI receive
            {
                RS232_point = ISP_point;
                ISP_point = 0;
                send_buffer_to_PC();    //send buffer data to PC
            }
            ISP_point = 0;
            SPI_status = 1;    //1:SPI send
            SPI_Recevie = 0;
            while (!SPI_Recevie);    //wait send the 1st data
            delay(50);    //set timeout
            clear_buffer();
            RS232_point = 0;
            ISP_point = 0;
            SPI_status = 0;    //0:SPI receive
            SPI_Recevie = 0;
        }
    }
}

```

---

---

```

    }
}
#endif
//-----
void SPI_Interrupt_Routine() interrupt SPI_INTERRUPT_VECTOR
{
    SPI_buffer = SPDAT;                //save SPI data
    SPSTAT = SPIF_WCOL_MASK;          //clear SPI flag
    SPI_Receive = 1;                  //set SPI received flag
    if (SPI_status)                    //1:SPI send
    {
        if (ISP_point < RS232_point)
        {
            SPDAT = buffer[ISP_point];
            ISP_point++;
        }
    }
    else                                //0:SPI receive
    {
        if (ISP_point < BUF_SIZE)
        {
            buffer[ISP_point] = SPI_buffer;
            ISP_point++;
        }
    }
}
//-----
void Initial_RS232()
{
    ES = 0;
    SCON = 0x50;                       //UART mode(8-bit variable)
    TMOD &= 0x0f;                       //timer0 mode(8-bit auto-reload)
    TMOD |= 0x20;
    TH1 = TL1 = 256 - FOSC/384/BAUD; //UART baudrate
    TR1 = 1
    AUXR |= 0x40;                       //1T mode
}
//-----
void Initial_SPI()
{
#ifdef MASTER
    SPCTL = CONFIG_MASTER;              //master mode
#else
    SPCTL = CONFIG_SLAVE;              //slave mode
#endif
    SPSTAT = SPIF_WCOL_MASK;          //clear SPI flag
    IE2 |= 0x02;                       //enable SPI interrupt
}

```

---



---

```

//-----
void Init_System()
{
    Initial_RS232();           //initial UART
    Initial_SPI();           //initial SPI
    EA = 1;
}
//-----
void RS232_Send_Byte(INT8U ch)
{
    TI = 0;                   //ready send
    SBUF = ch;                //write UART data
    while (TI = 0);          //wait data sent
    TI = 0;                   //clear TX flag
}
//-----
void send_buffer_to_PC()      //send all data in buffer to PC
{
    INT8U i;
    if (RS232_point == 0) return;
    RS232_Send_Byte(RS232_point);
    if (i=0; i<RS232_point; i++)
    {
        RS232_Send_Byte(buffer[i]);
    }
}
//-----
void clear_buffer()          //empty data buffer
{
    INT8U i;
    for (i=0; i<BUF_SIZE; i++)
    {
        buffer[i] = 0;
    }
}
//-----
void delay(INT16U d)
{
    INT16U i;
    while (d--)
    {
        i = 1000;
        while (i--);
    }
}

```

---

```
//-----  
#ifndef MASRER  
void SPI_read_from_slave(INT8U n)           //receive slave data  
{  
    INT8U j;  
  
    clear_buffer()  
    SPI_status = 0;                          //0:SPI receive  
    ISP_point = 0;  
    SPI_Receive = 0;  
    SPDAT = 0x00;                            //trigger SPI clock  
    while (!SPI_Receive);  
    SPI_Receive = 0;  
    ISP_point = 0;                            //discard the 1st data  
    for (j=0; j<n; j++)  
    {  
        SPDAT = 0x00;                        //trigger SPI clock  
        while (!SPI_Receive);  
        SPI_Receive = 0;  
    }  
}  
#endif
```

---

# Chapter 16 Compiler / ISP Programmer / Emulator

## 16.1 Compiler/Assembler and Head File

About STC MCU Compiler/Assembler :

1. Any traditional compiler / assembler and the popular Keil C51 are suitable for STC MCU.
2. For selection MCU body, the traditional compiler / assembler, you can choose Intel's 8052 / 87C52 / 87C52 / 87C58 or Philips's P87C52 / P87C54/P87C58 in the traditional environment, in Keil environment, you can choose the types in front of the proposed or download the STC chips database file (STC.CDB) from the STC official website
3. For STC15 series MCU, in Keil C development environment, select the Intel 8052 to compiling, And only contain < reg51.h > as header file. New special function registers could be declared by sfr and new register bits declared by sbit. Take new special function registers and bits about P4 port for example:

Address statement by C language:

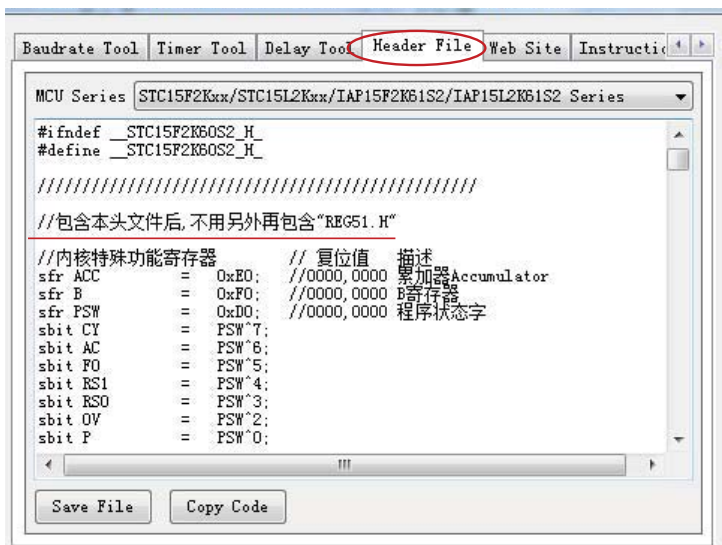
```
sfr P4      = 0xC0;      //8 bit   Port4  P4.7 P4.6 P4.5 P4.4 P4.3 P4.2 P4.1 P4.0  111,1111
sfr P4M0 = 0xB4;      //                               0000,0000
sfr P4M1 = 0xB3;      //                               0000,0000

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;
```

Address statement by Assembly:

```
P4      EQU 0C0H      ; or P4      DATA 0C0H
P4M1    EQU 0B3H      ; or P4M1    DATA 0B3H
P4M0    EQU 0B4H
```

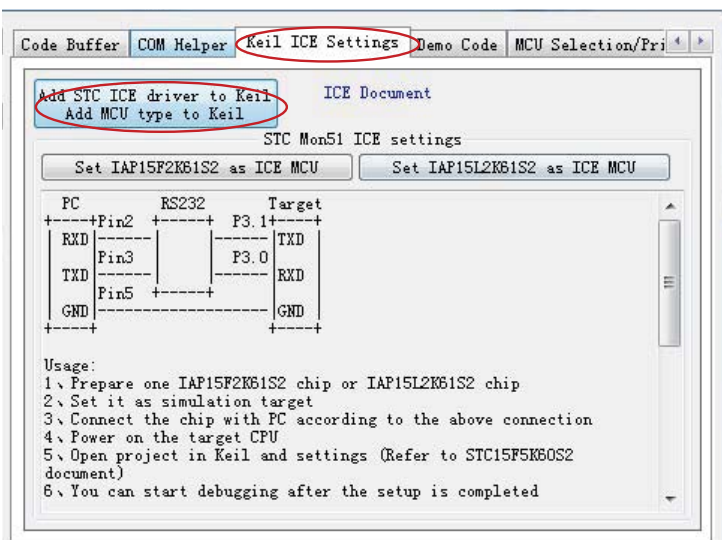
For parts of STC MCU, users can download their head files from STC official website . In addition, the latest STC ISP tool STC-ISP-15xx-V6.82 also could generate head files for STC15 series. See the following figure. These head files would replace "reg51.h" if need be.



There are many versions of Keil C51 development environment. But Keil  $\mu$ Vision2 and Keil  $\mu$ Vision3 and Keil  $\mu$ Vision4 are the most common ones for 8051 MCU. Now let us introduce how to develop, compile and debug uer program by Keil  $\mu$ Vision2 and Keil  $\mu$ Vision3 and Keil  $\mu$ Vision4

If need to add STC MCU into database of Keil  $\mu$ Vision2 or Keil  $\mu$ Vision3 or Keil  $\mu$ Vision4, you may be do as below:

(1) Open the newest version of STC-ISP Programmer/Writer — STC-ISP-V6.82, and choose the page "Keil ICD Settings", and then click the button "Add MCU type to Keil".

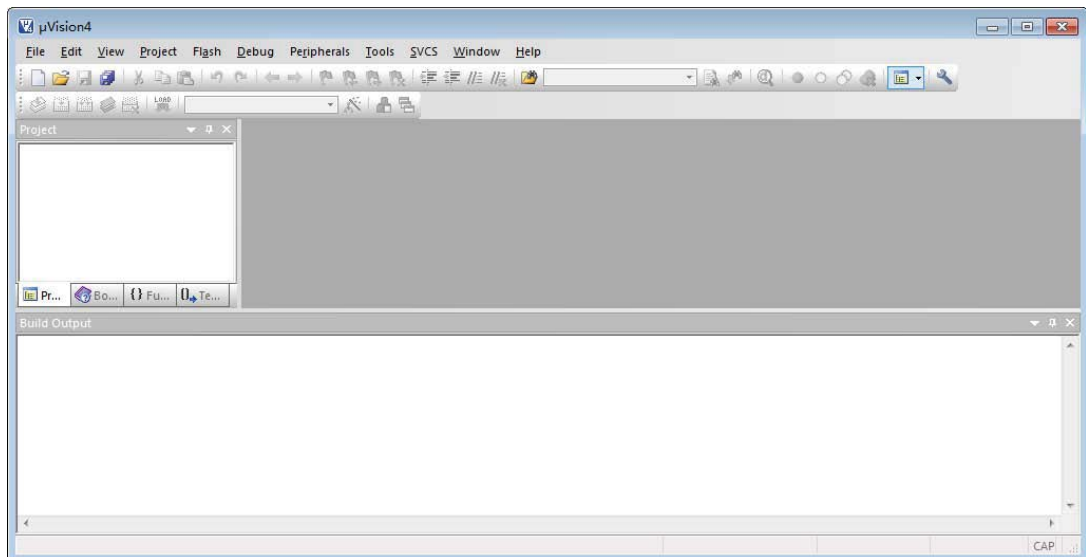


(2) Next location at the Keil setup directory(eg.“C:\Keil”), press “OK”.

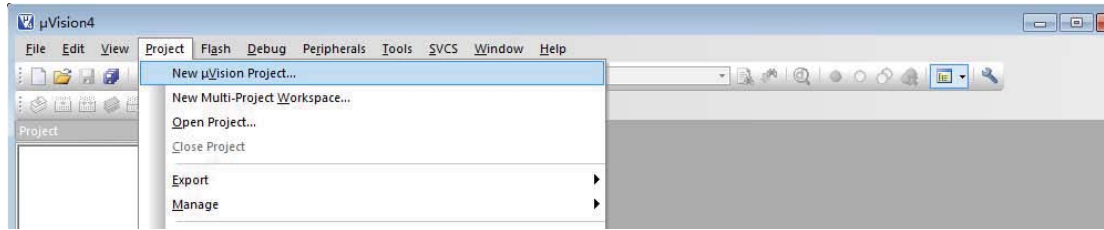


Now let us take Keil  $\mu$ Vision4 for example to introduce how to develop, compile and debug uer program

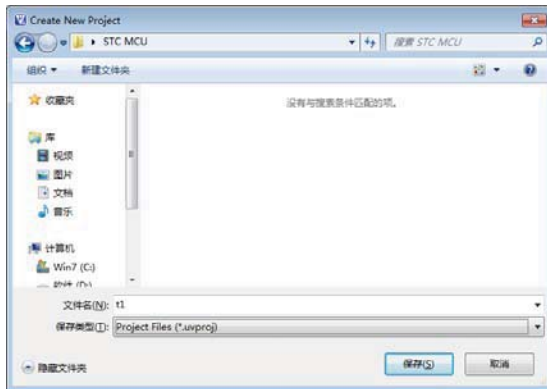
(1) Start up Keil  $\mu$ Vision4, the edit interface of Keil  $\mu$ Vision4 is shown below.



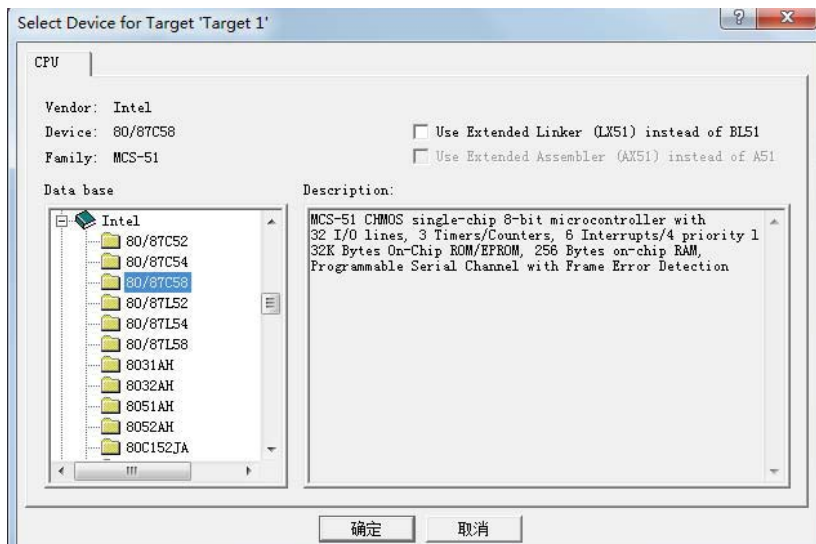
(2) Create a new project: click the menu **Project** , choose "New Project" in the drop-down boxes.



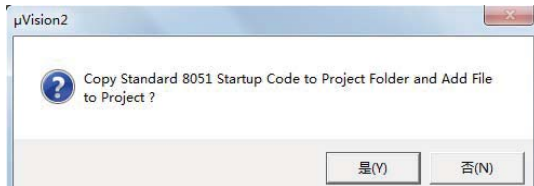
(3) Save the new project. For example, save the project into C:\Users\THINK\Documents\STC MCU, project name is "t1". The default extension name for a Keil μVision2 project file is **.uv2**



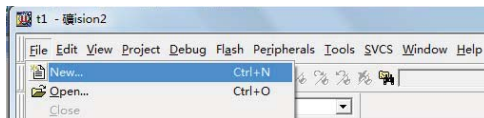
(4) After save the new project, the dialog "Select Device for Target" will be popup, shown below. users can select MCU type in "Data base" listing. **STC MCU** choose **Intel 80/87C58**.



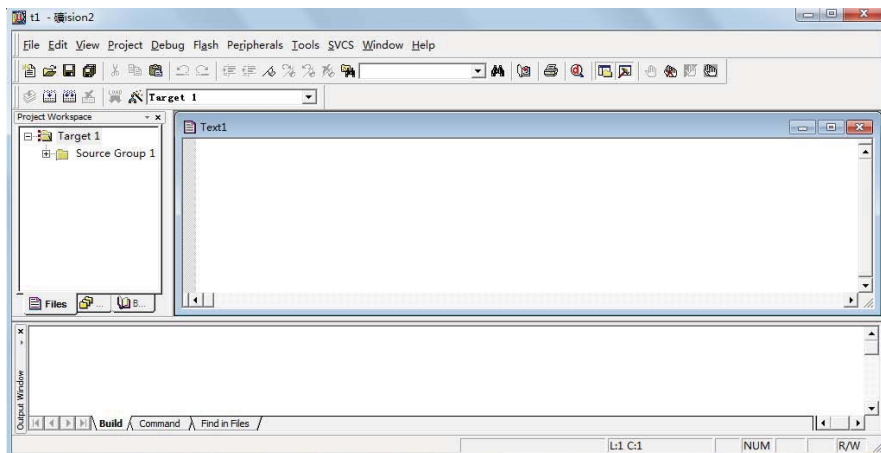
- (5) After select MCU type, Keil uVision 2 will ask whether copy standard 8051 startup code (STARTUP.51) to project folder and add file to project or not. In general conditions, click No .



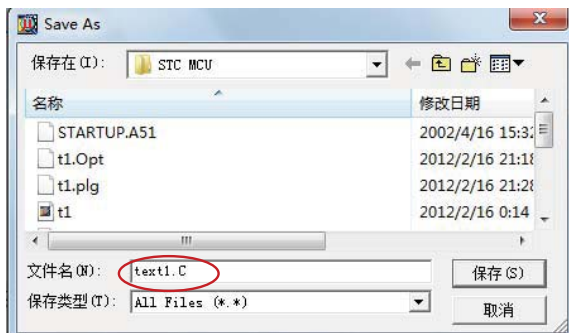
- (6) Start to write a program after finish creating project, click New option in File menu. See the following figure



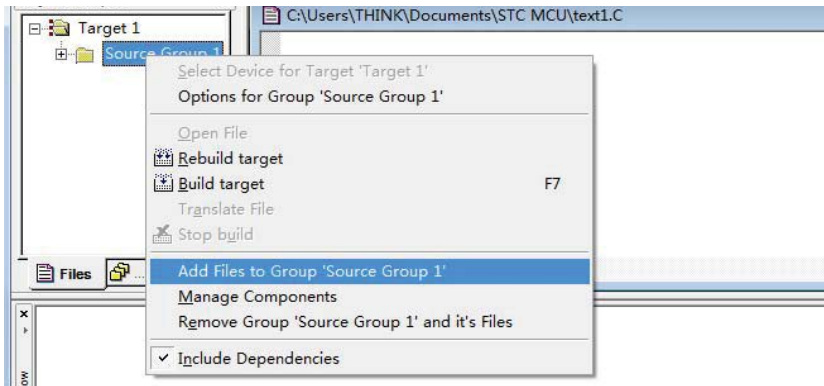
The interface after creating a new file is shown below.



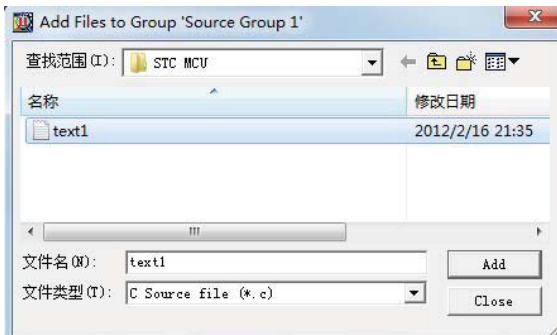
- click Save as option in File menu could save the new file. When saving, the file extension also need to key in. The extension name for C program file is .C and for assembly file is .ASM (case insensitive).



- (7) Add application program to project: click the "+" in front of "Target 1", and then Right-click "Source Group 1", popup menu as follows.

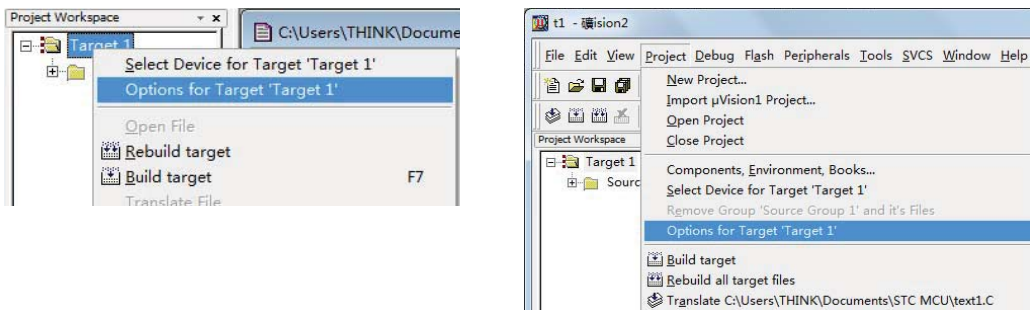


Click "Add File to Group 'Source Group 1'", then poppu the following dialog.



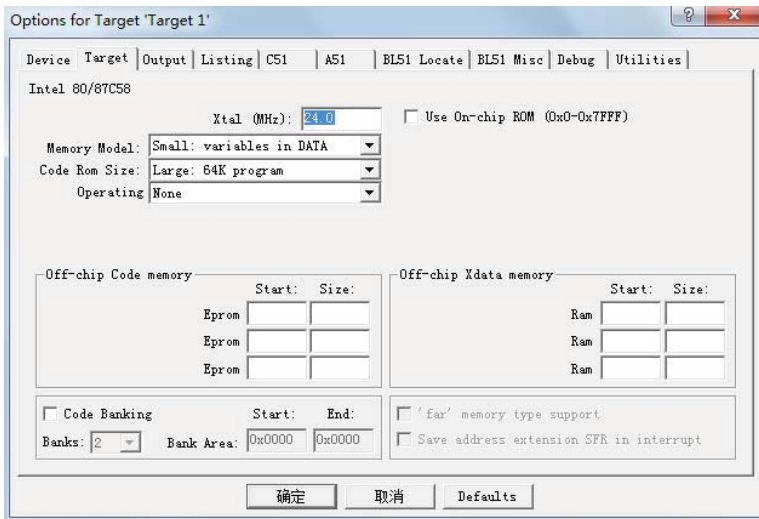
Choose file "text1.c" (example), click the "Add" to finish adding application program to project.

- (8) Environment Settings: Right-click "Target 1" and choose " Options for Target 'Target1' " in pull-down menu or Project→ Options for Target 'Target1', "Options for Target 'Target1' " dialog will be popup



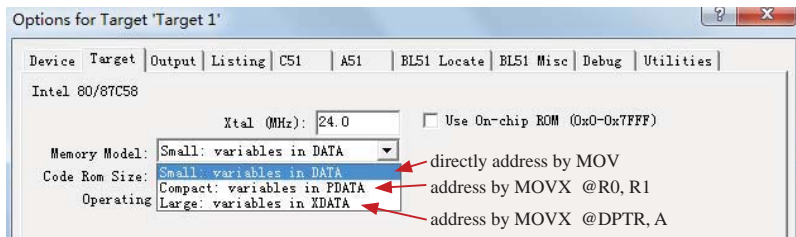


## "Options for Target 'Target1' " dialog



The dialog of "Options for Target 'Target1' " have several options, such as "Device" selection, "Target" attribute, "Output" attribute, "C51" compiler attribute, "A51" compiler attribute, "BL51" linker attribute, "Debug" attribute and so on. These options except the following ones generally do not be set by users.

### Data memory model setting



Start and End address of program area is defaulted from 0x0000 to 0xFFFF, shown as following figure. The default start and end address is correct.



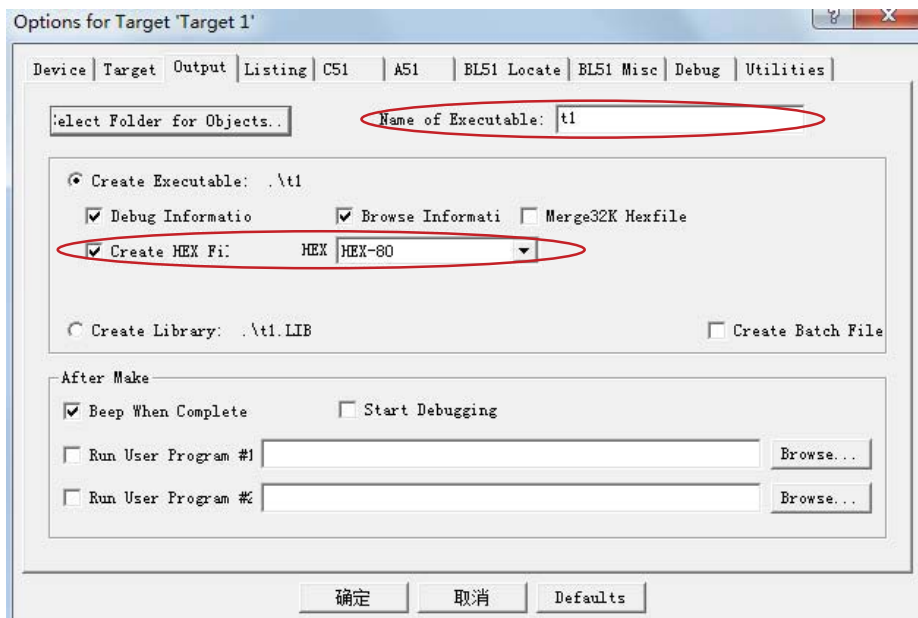
However, the following start and end address is illegal, it must be corrected.



The steps of correcting the start and end address is shown below: check “Code Banking” firstly and then revise the start and end address in “Bank Area”. Last, remove the tick of “Code Banking”



Automatically create HEX file when compiling and linking. Click "Output" and choose "Create HEX File" with a tick, see the following figure.



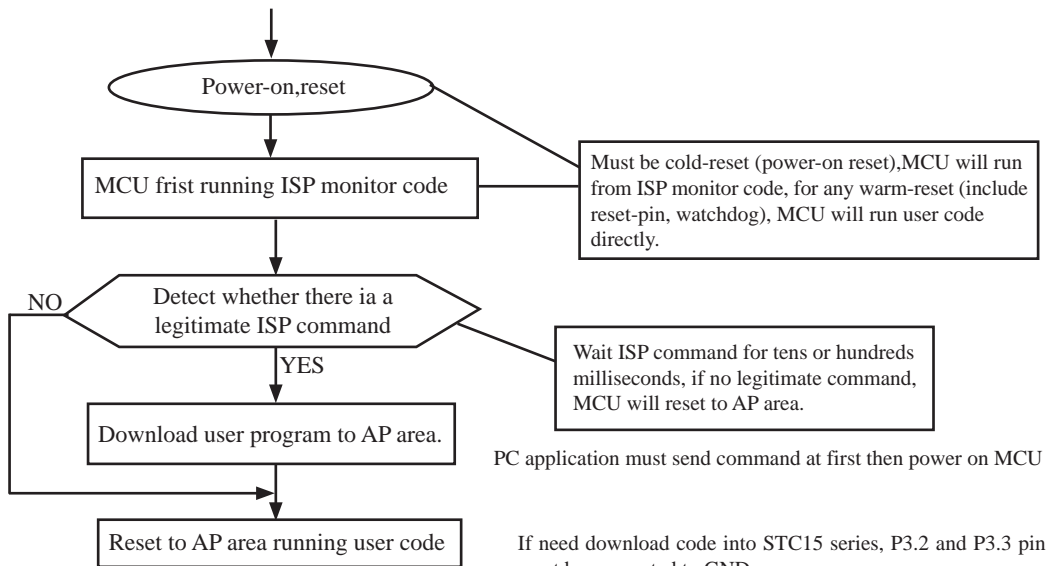
## 16.2 ISP Programmer / Burner

STC has Special STC-ISP cheap programming tools

STC-ISP programming tools are classified as follow

	STC12/11/10/89/90 series 40-pin MCU special ISP Programming tool
	STC12/11/10/89/90 series 32-pin MCU special ISP Programming tool
STC12/11/10/89/90 series	STC12/11/10/89/90 series 28-pin MCU special ISP Programming tool
ISP programming tools	STC12/11/10/89/90 series 20-pin MCU special ISP Programming tool
STC-ISP Programming Tools	STC12/11/10/89/90 series 18-pin MCU special ISP Programming tool
	STC12/11/10/89/90 series 16-pin MCU special ISP Programming tool
	STC15 series ISP programming tool

### 16.2.1 In-System-Programming (ISP) principle

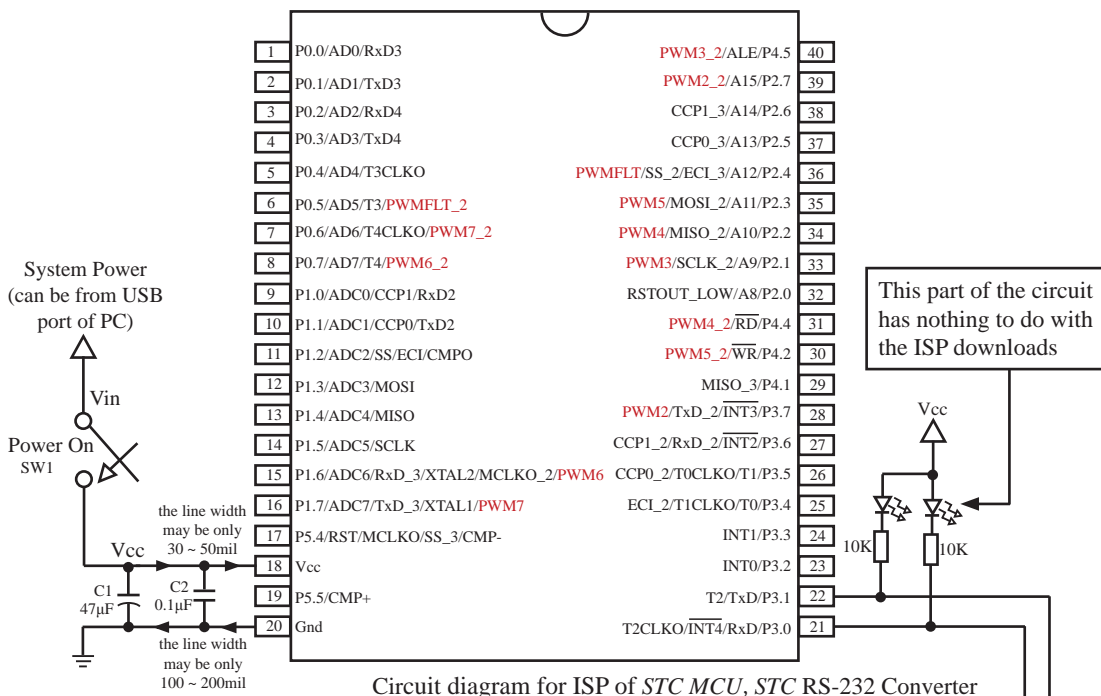


If need download code into STC15 series, P3.2 and P3.3 pin must be connected to GND

If you chose the "Next program code, P3.2/P3.3 need=0/0" option, then the next time you need to re-download the program, first of all must be connected P3.2 and P3.3 to GND

## 16.2.2 Application Circuit Diagram for ISP of STC15W4K32S4 series

### 16.2.2.1 Application Circuit Diagram for ISP using RS-232 Converter



Note P0 ports can be multiplexed as Address/ Data bus not as A/D Converter. 8 channels of A/D Converter are on P1.

Consequently P0.x/ADx means that P0.x can be used as Address/Data bus, while P1.x/ADCx means P1.x can be used as A/D conversion channel in the pin map.

Please power on the target MCU after press down the button "Download/Program" on STC-ISP.exe when burning code to MCU.

Internal highly reliable Reset, so external reset circuit can be completely removed.

P5.4/RST/MCLKO pin factory defaults to the I/O port, which can be set as RST reset pin(active high) through the STC-ISP programmer.

Internal high-precise R/C clock(  $\pm 3\%$  ),  $\pm 1\%$  temperature drift (  $-40 \sim +85$  ) while  $\pm 0.6\%$  in normal temperature (  $-20 \sim +65$  ), so external expensive crystal can be completely removed.

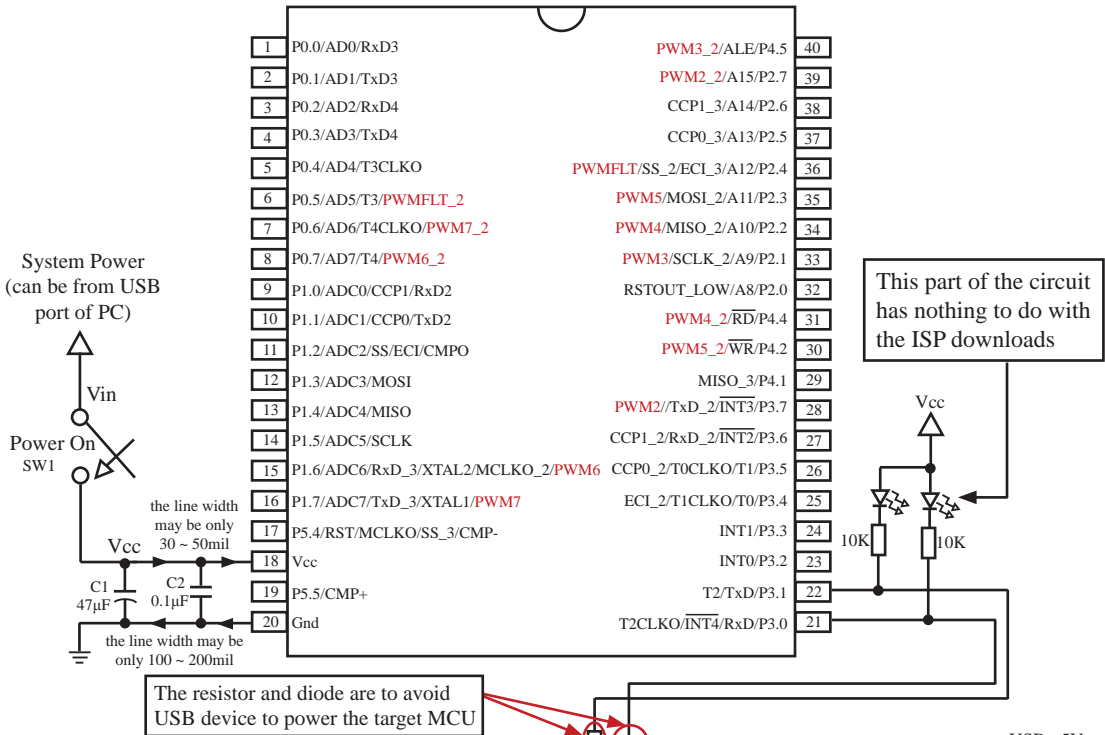
Recommend to add decoupling capacitor C1(47µF) and C2(0.1µF) between Vcc and Gnd that can remove power noise and improve the anti-interference ability.

---

Users in their target system, such as the P3.0/P3.1 through the RS-232 level shifter connected to the computer after the conversion of ordinary RS-232 serial port to connect the system programming / upgrading client software. If the user panel recommended no RS-232 level converter, should lead to a socket, with Gnd/P3.1/P3.0/Vcc four signal lines, so that the user system can be programmed directly. Of course, if the six signal lines can lead to Gnd/P3.1/P3.0/Vcc/P1.1/P3.2 as well, because you can download the program by P1.0/P3.3 ISP ban. If you can Gnd/P3.1/P3.0/Vcc/P1.1/P1.0/Reset seven signal lines leads to better, so you can easily use "offline download board (no computer)" .

ISP programming on the Theory and Application Guide to see "STC15W4K32S4 Series MCU Development / Programming Tools Help"section. In addition, we have standardized programming download tool, the user can then program into the goal in the above systems, you can borrow on top of it RS-232 level shifter connected to the computer to download the program used to do. Programming a chip roughly be a few seconds, faster than the ordinary universal programmer much faster, there is no need to buy expensive third-party programmer. PC STC-ISP software downloaded from the website

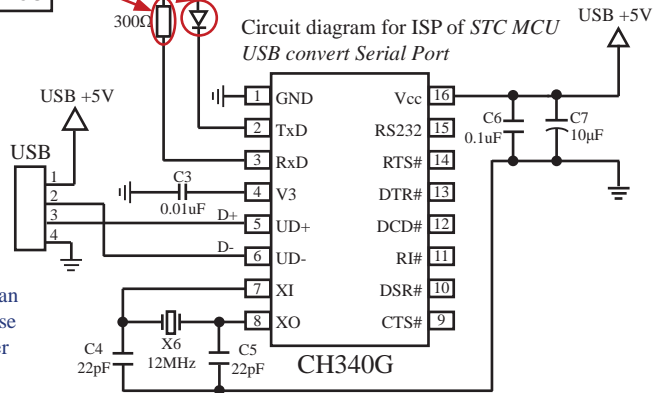
## 16.2.2.2 Application Circuit Diagram for ISP using USB to convert Serial Port



Note P0 ports can be multiplexed as Address/Data bus not as A/D Converter. 8 channels of A/D Converter are on P1.

Consequently P0.x/ADx means that P0.x can be used as Address/Data bus, while P1.x/ADCx means P1.x can be used as A/D conversion channel in the pin map.

Recommend to choose CH340G ( Its pins are not compatible with CH341's, but whose price less than RMB 1.1 yuan is more cheap), also you can choose PL2303(its price is less than RMB 1.0 yuan), refer to [www.wch.cn](http://www.wch.cn) for more detail.



Internal highly reliable Reset, so external reset circuit can be completely removed.

P5.4/RST/MCLKO pin factory defaults to the I/O port, which can be set as RST reset pin(active high) through the STC-ISP programmer.

Internal high-precise R/C clock ( ±3% ), ±1% temperature drift ( -40 ~ +85 ) while ±0.6% in normal temperature ( -20 ~ +65 ), so external expensive crystal can be completely removed.

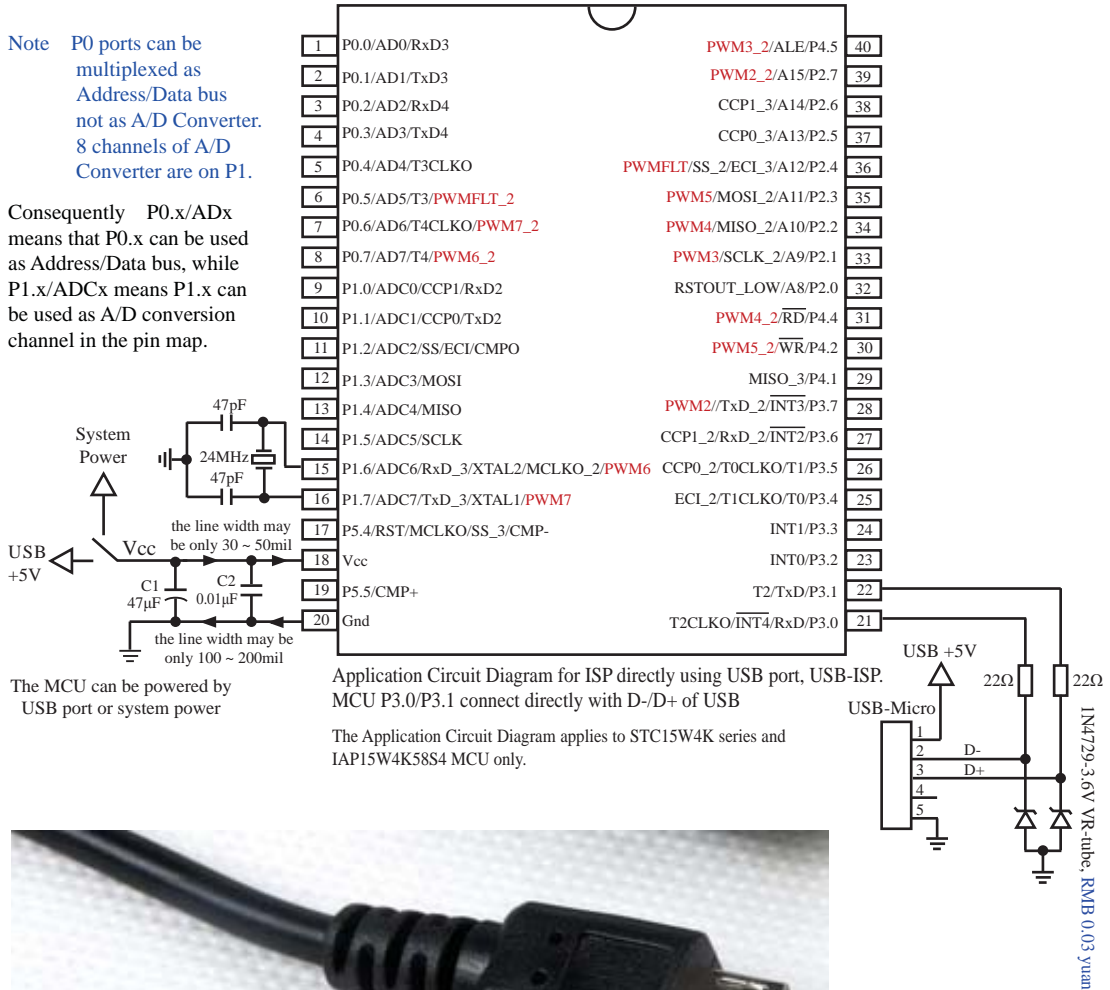
Recommend to add decoupling capacitor C1(47μF) and C2(0.1μF) between V<sub>CC</sub> and Gnd that can remove power noise and improve the anti-interference ability.

### 16.2.2.3 Application Circuit Diagram for ISP directly using USB port

—P3.0/P3.1 of STC15W4K series and IAP15W4K58S4 connect directly with D-/D+ of USB

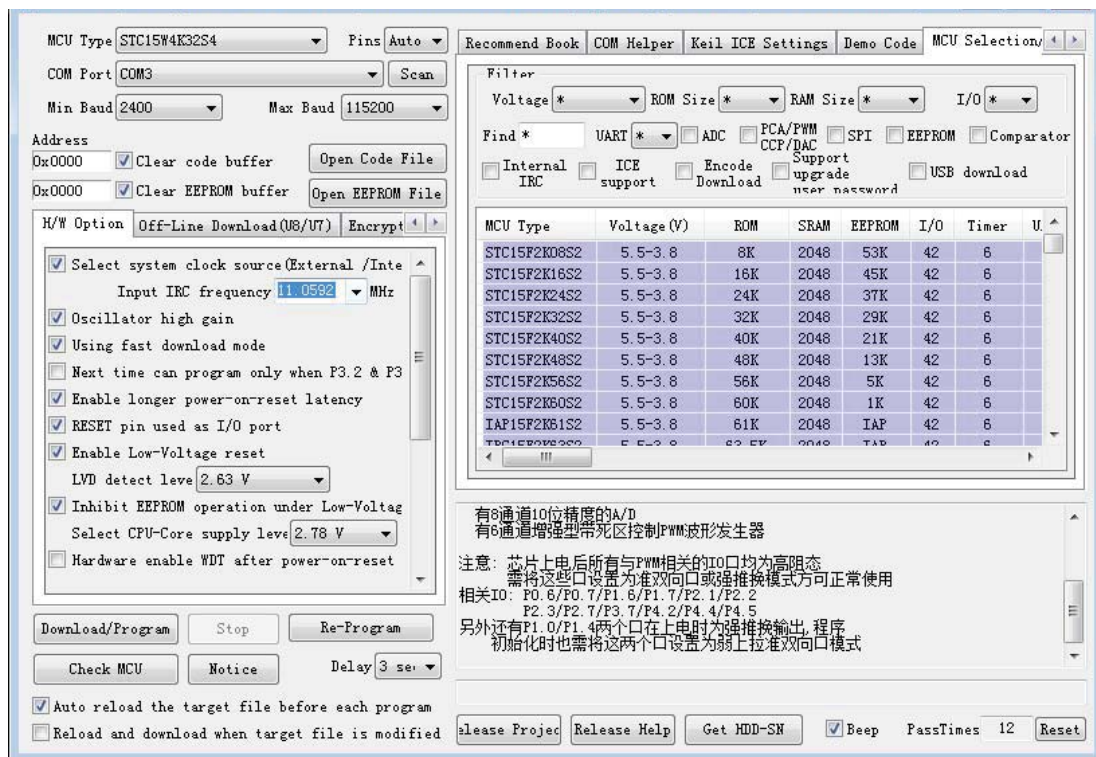
Note P0 ports can be multiplexed as Address/Data bus not as A/D Converter. 8 channels of A/D Converter are on P1.

Consequently P0.x/ADx means that P0.x can be used as Address/Data bus, while P1.x/ADCx means P1.x can be used as A/D conversion channel in the pin map.



USB-Micro

## 16.2.3 PC Side Control Software Usage



The overview of STC15xx-isp-V6.82 interface is shown above. The STC-ISP software has added many new features, such as scanning COM port, Baudrate tool, Timer tool, Delay tool and so on.



The image shows a software interface for programming an MCU. The interface includes several sections with callout boxes providing explanations:

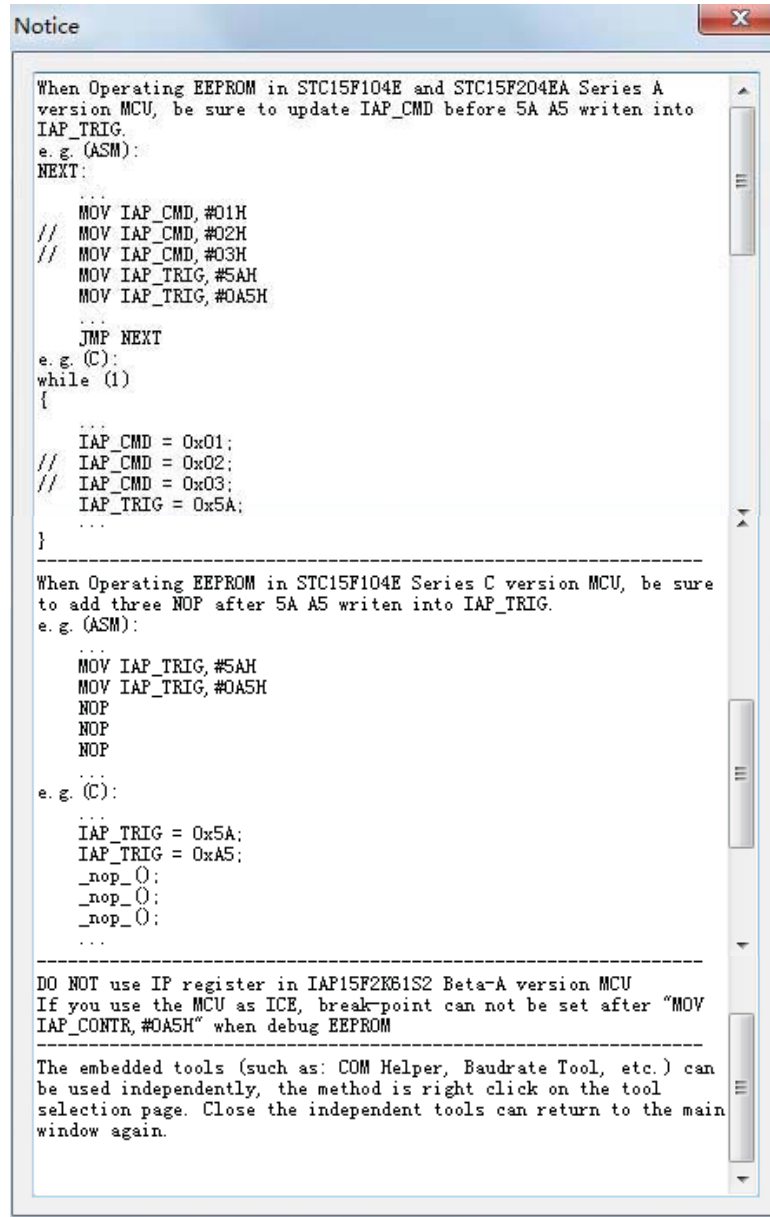
- MCU Type Selection:** A dropdown menu is set to "STC15W4K32S4".
- Scan the usable serial port in current system:** A "Scan" button is located next to the "COM Port" dropdown, which is set to "COM3".
- According to actual situation, the user selects the appropriate maximum baud rate:** The "Max Baud" dropdown is set to "115200".
- Open user program code file:** An "Open Code File" button is located next to the "Address" field.
- Whether to use the faster speed of the internal oscillator frequency for download:** A checkbox "Select system clock source (External / Internal)" is checked. Below it, "Input IRC frequency" is set to "11.0592 MHz".
- Enable Low-Voltage reset, controls reset or not while the Low-Voltage event:** A checkbox "Enable Low-Voltage reset" is checked. Below it, "LVD detect level" is set to "2.63 V".
- Low-Voltage Detector Parameter, it adjust the threshold voltage level of the built-in Low-Voltage detector:** A callout box explains that the LVD detect level should be adjusted based on the oscillator frequency (20MHz+). For a 3V chip, the threshold should be more than 2.5V. For a 5V chip, it should be more than 4.11V.
- All new settings are valid in the next power-on:** A callout box points to the "Re-Program" button.
- Press this button when mass production:** A callout box points to the "Re-Program" button.
- Enable the option in debugging stage:** A callout box points to the "Auto reload the target file before each program" checkbox.

Other visible settings include:

- "Clear code buffer" and "Clear EEPROM buffer" checkboxes are checked.
- "H/W Option" includes "Off-Line Download (U8/U7)" and "Encrypt".
- "Oscillator high gain" and "Using fast download mode" are checked.
- "Next time can program only when P3.2 & P3" is unchecked.
- "Enable longer power-on-reset latency" and "RESET pin used as I/O port" are checked.
- "Inhibit EEPROM operation under Low-Voltage" is checked, with "Select CPU-Core supply level" set to "2.78 V".
- "Hardware enable WDT after power-on-reset" is unchecked, with "Watch-Dog-Timer prescaler" set to "256".
- "WDT stop count while MCU in idle mode" is checked.
- "Erase all EEPROM data next time program c" is unchecked.
- "P2.0 power-on reset state" is unchecked.
- "Switch UART1 [Rx/D, Tx/D] from [P3.0, P3.1] to [P3.6, P3.7], P3.7 output P3.6's level" is unchecked.
- "P3.7 pin as PUSH-PULL mode" is unchecked.
- "Add a reset code in front of ID" is unchecked.

Buttons at the bottom include "Download/Program", "Stop", "Re-Program", "Check MCU", "Notice", and "Delay 3 se.". There are also checkboxes for "Auto reload the target file before each program" and "Reload and download when target file is modified".

Click "Notice / Help " in the interface will show the following dialog.



RS485 Control Interface is shown below

MCU Type  Pins

COM Port

Min Baud  Max Baud

Address  
0x0000  Clear code buffer   
0x0000  Clear EEPROM buffer

Self-Define Download **RS485 Control** Auto Inc

**RX Control Setting**  
RX Control PIN   
RX Control Level

**TX Control Setting**  
TX Control PIN   
TX Control Level

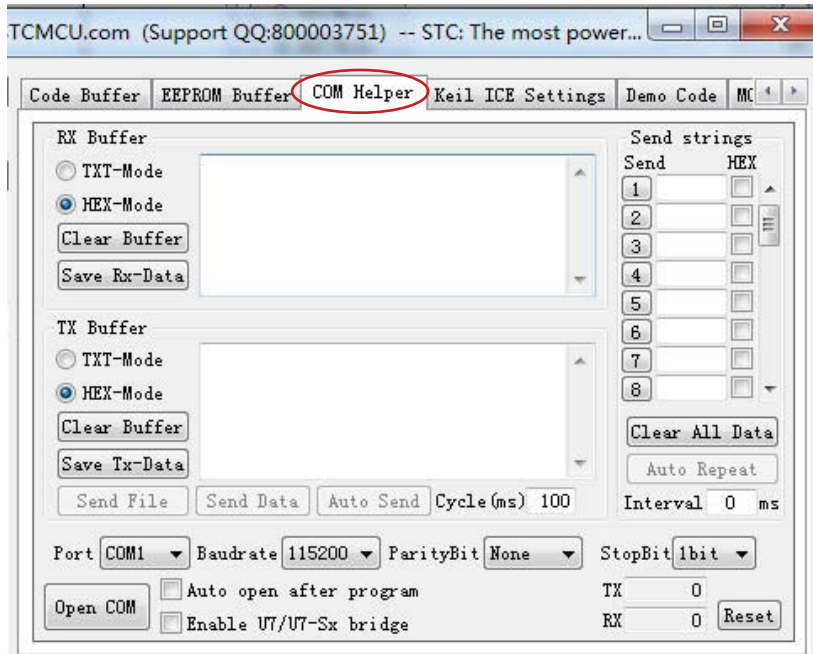
Enable RS485 control at next program  
 Program with RS485 at this time  
[RS485 using helper](#)

RS485 function only for the following serie   
STC15F2K60S2/STC15L2K60S2  
IAP15F2K61S2/IAP15L2K61S2  
STC15F104W/STC15L104W

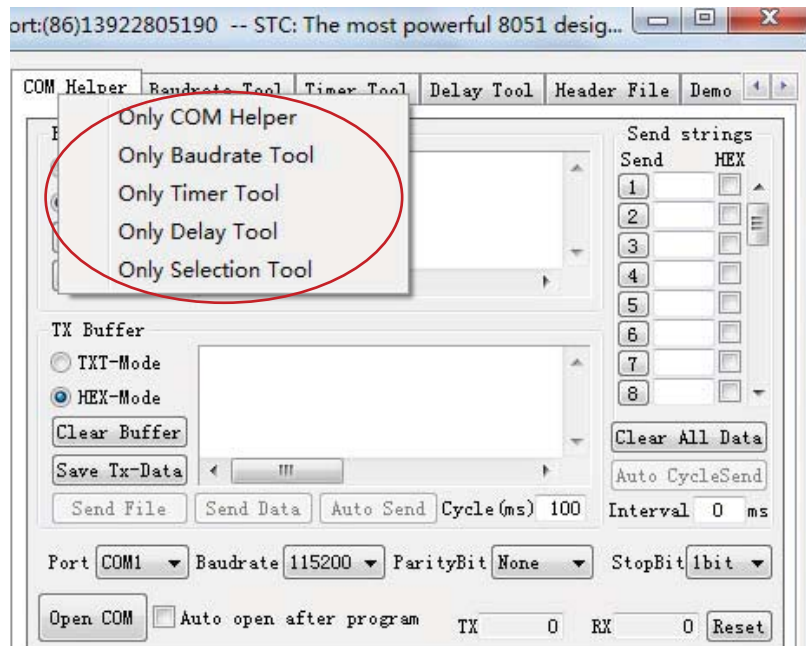
Delay

Auto reload the target file before each program  
 Reload and download when target file is modified

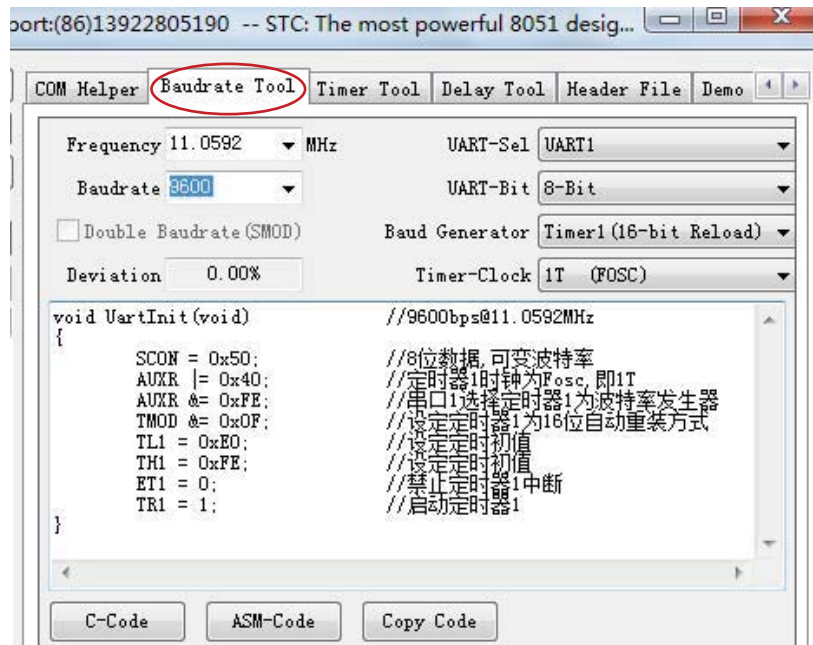
COM Helper Interface is shown below



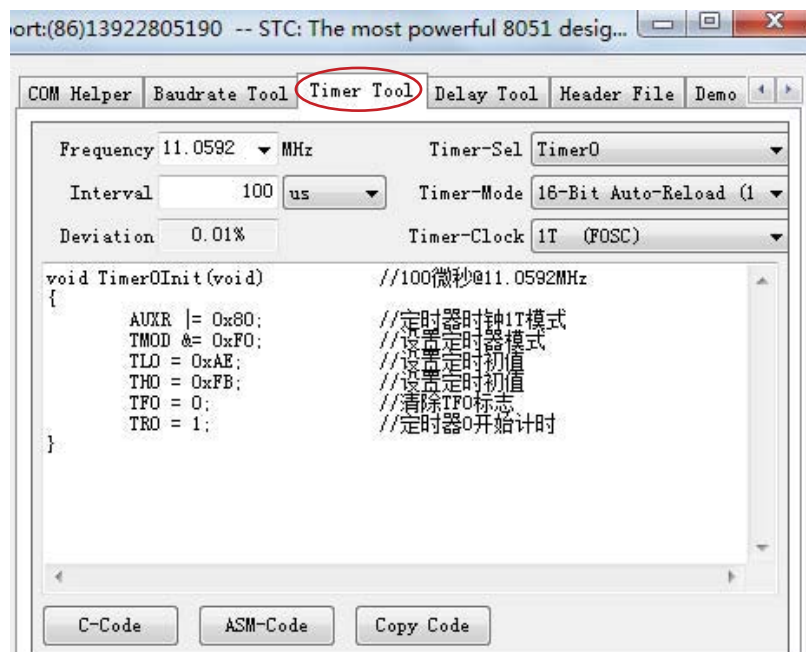
The embedded tools (such as: COM Helper, Baudrate Tool, etc.) can be used independently, the method is right click on the tool selection page. Close the independent tools can return to the main window again.



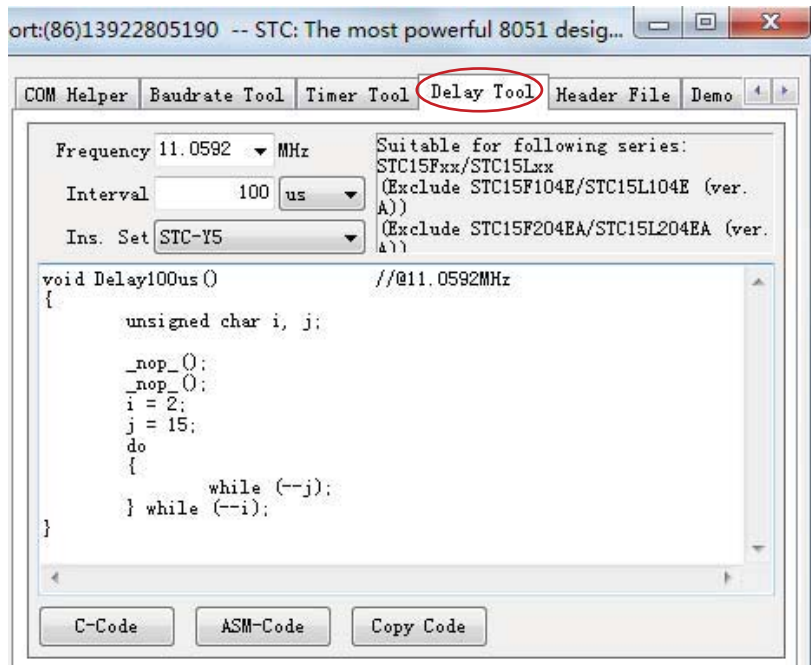
Baudrate tool Interface is shown below



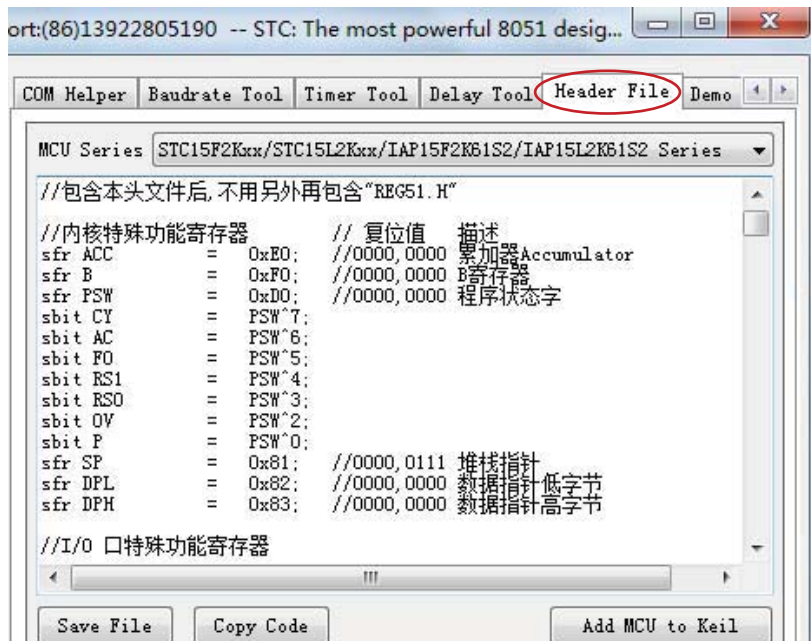
Timer tool Interface is shown below



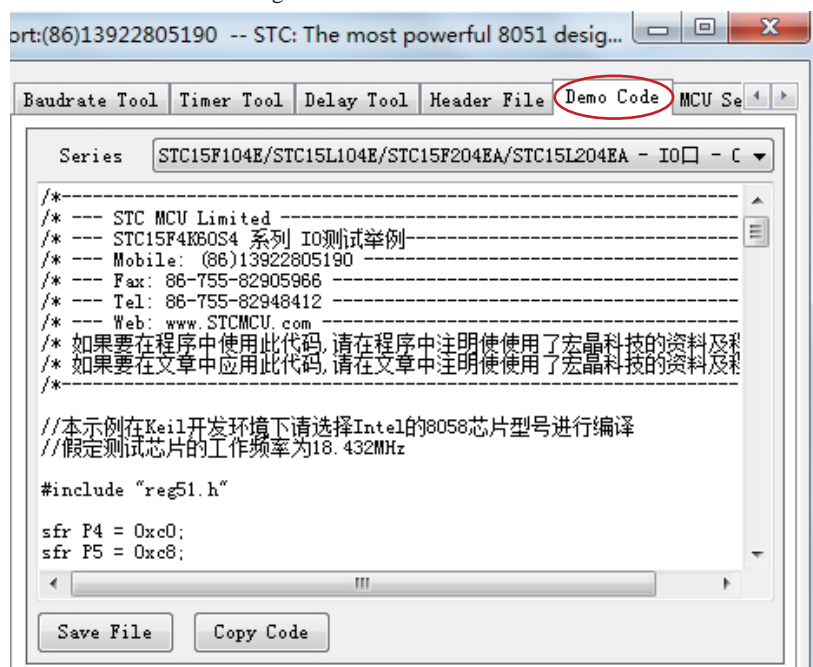
Delay tool Interface is shown below



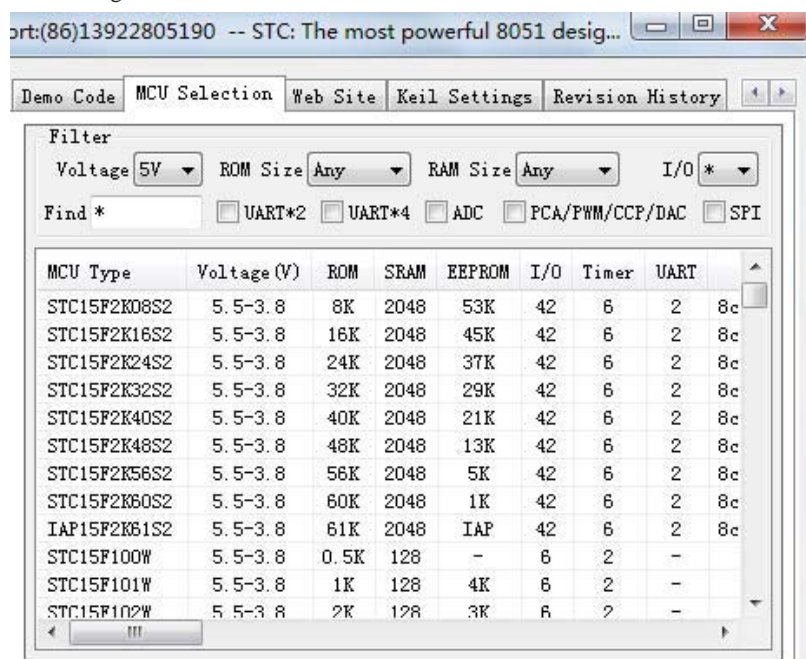
Dialog of Header file is shown below



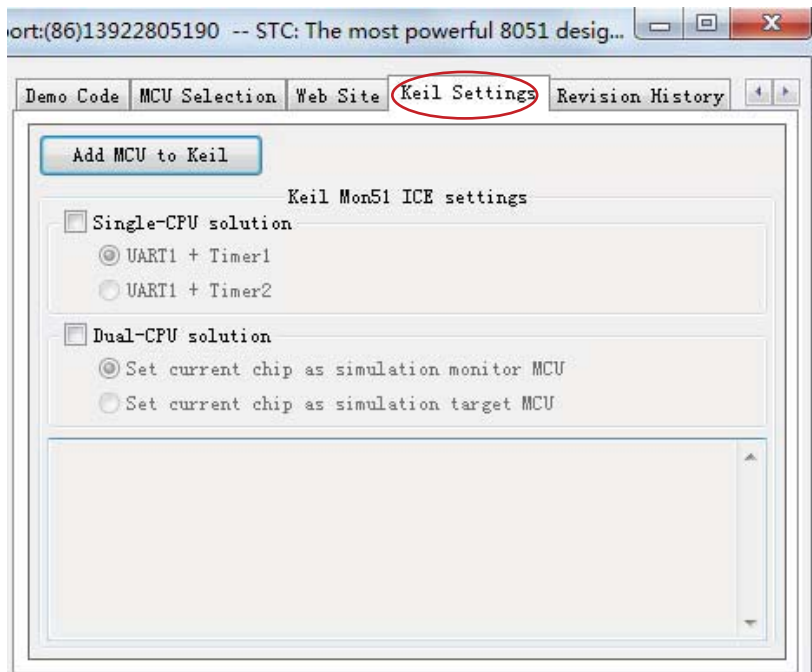
There is an demo code designed into STC-ISP software.



Next dialog shows the STC MCU Selection



The keil settings interface is shown below



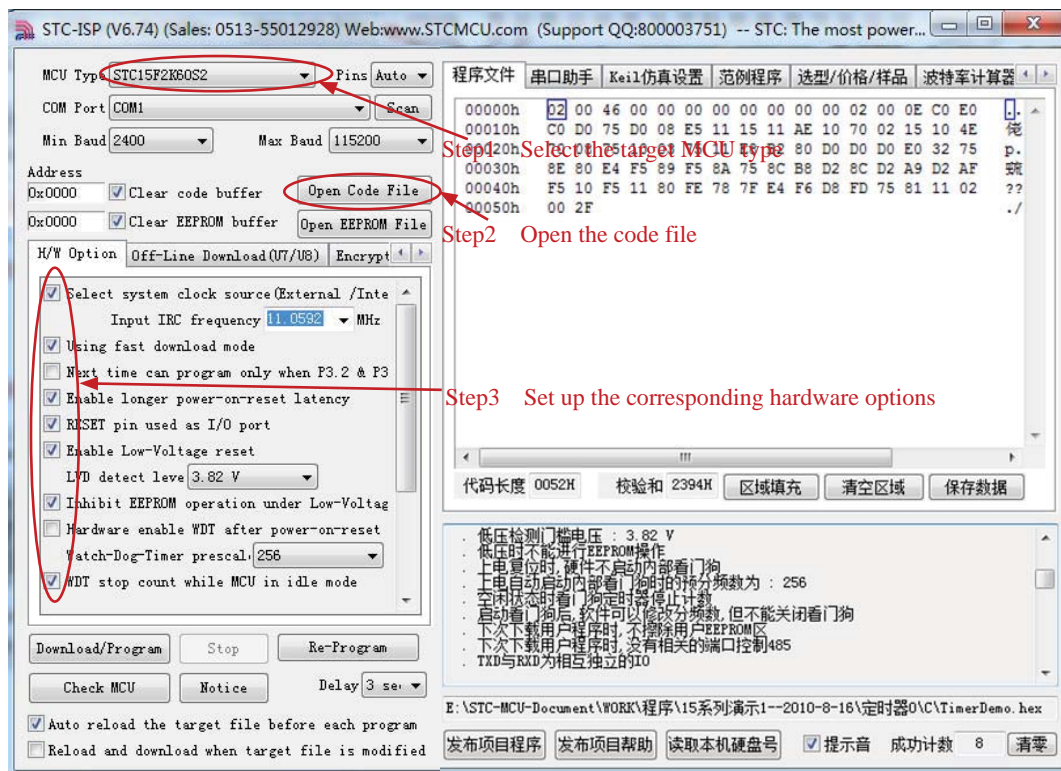


## 16.2.4 How to Release Project

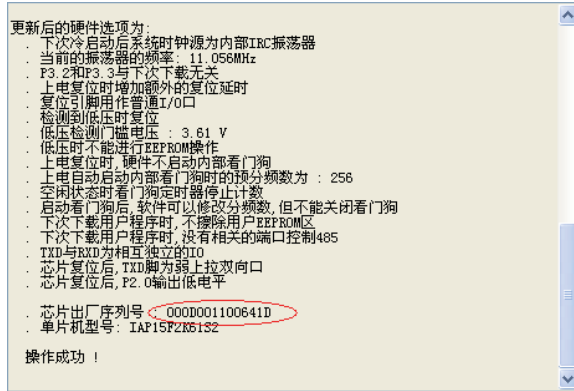
The release project is a function to bound the user code and related options into a direct download program executable file. On the interface, users can customize (users can modify and publish the application title, the buttons name and help information), at the same time, the user can also specify the target computer's hard disk ID number and the target chip ID. It can control the release application program can only run in the specified computer but cannot run on the other computers. Similarly, when the target chip ID is specified, then the user code can only be downloaded to the target chip has a corresponding ID number, for the other mcu whil do not programming.

The detailed steps are as follows:

1. At first, to select the target MCU type
2. Open the code file
3. Set up the corresponding hardware options



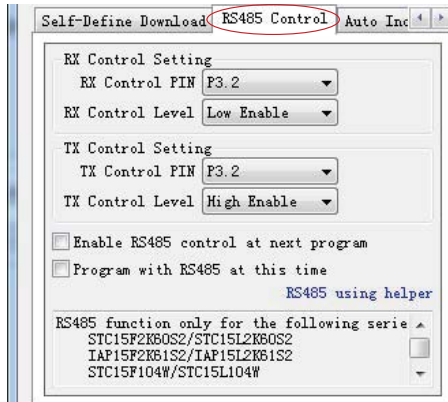
4. Test downloading a chip, and remember the target chip ID  
(without check the target chip ID can skip this step)



5. Set automatically increment  
(neednot automatically increment, can skip this step)



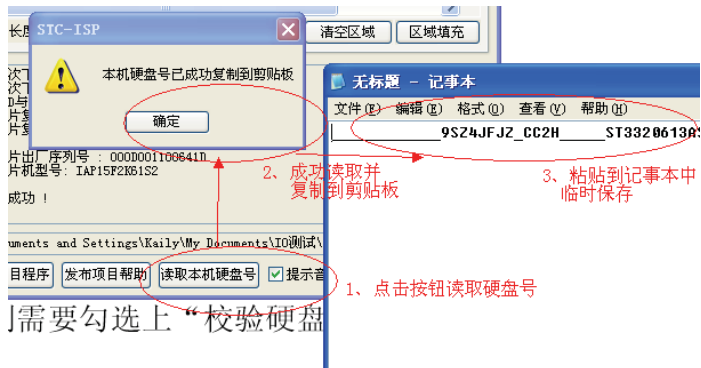
6. Set 485 control options (no using 485 Control, can skip this step)



7. Set custom download command  
(no using this function, can skip this step)



8. Click "read the hard disk" button, and remember the target computer's hard disk number  
(neednot to check the target hard disk, can skip this step)

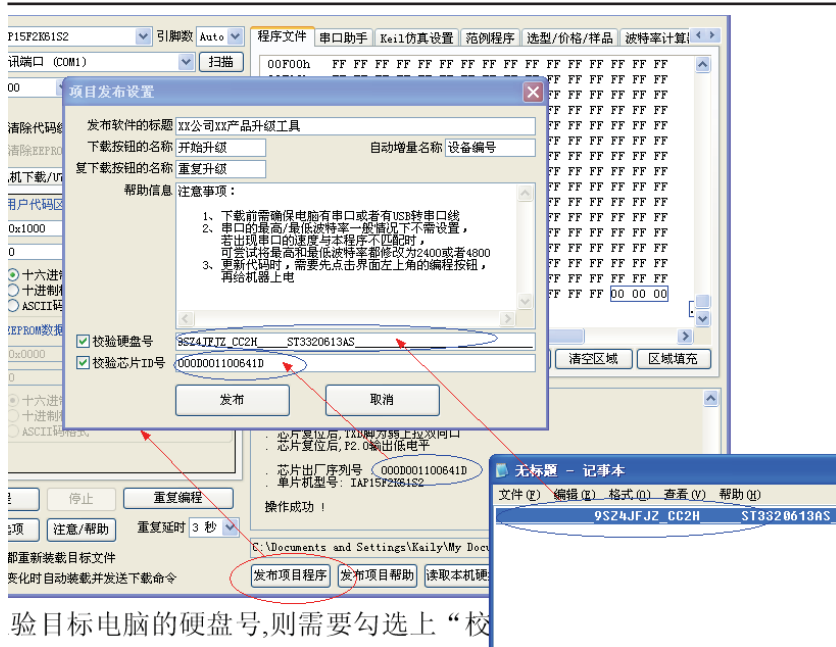


9. Click "Release project" button, enter the release application program settings interface

10. According to your needs, modified release software titles, the buttons name, repeat the download button name, automatic increment name and help informations

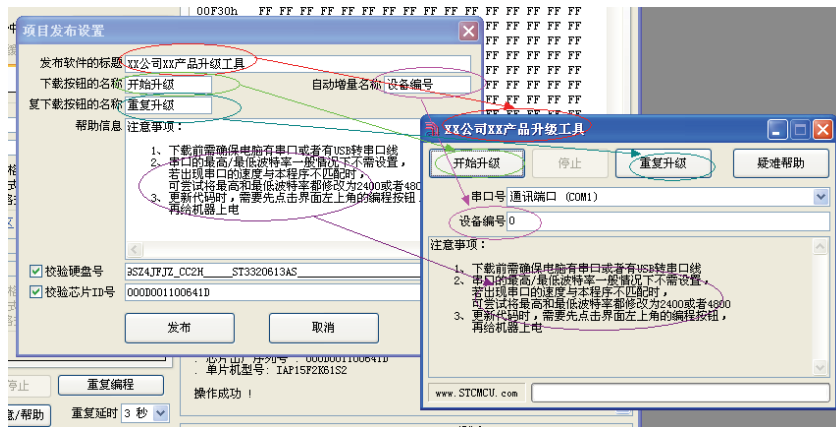
11. If you need to check the target computer's hard disk number, to check the "Check HDD-SN", and fill the target computer hard disk number to the following edit box.

12. If you need to check the target chip ID, to check the "check MCU ID", and fill the target MCU ID to the following edit box.



验目标电脑的硬盘号,则需要勾选上“校

13. Click the Publish button, then you can obtain the corresponding executable file



Notice:

The new function which checking HDD-SN and checking MCU ID is only for the following series:

- STC15F2K60S2/STC15L2K60S2
- IAP15F2K61S2/IAP15L2K61S2
- STC15F104W/STC15L104W
- IAP15F105W/STC15L105W
- STC15W104SW/IAP15W105W
- STC15W201S/IAP15W205S
- STC15F408AD/STC15L408AD
- IAP15F413AD/IAP15L413AD

## 16.2.5 How to Encrypt User Code by Software STC15-ISP-Ver6.82

STC-ISP (V6.82H) (Sales: 0513-55012928) Web:www.S

MCU Type: STC15W4K32S4 Pins: Auto

COM Port: COM3 Scan

Min Baud: 2400 Max Baud: 115200

Address  
0x0000  Clear code buffer Open Code File  
0x0000  Clear EEPROM buffer Open EEPROM File

Encryption Download User Password Self-Defi

How to download encryption code  
1. Press "Open Code File" to open a encryption  
2. Select "The code ready to download is encr  
3. Press "Download/Program" button

How to encrypt code  
1. Press "Create KEY" or "Load KEY"  
2. Press "Encode File"  
3. Choose unencrypted file in open dialog bo  
4. enter the encrypted file name in the save

How to update IC internal key  
1. Press "Create KEY" or "Load KEY"

Encryption download helper  
Create KEY Load KEY Save KEY Encode File

Update the user KEY before download user cod  
 The code ready to download is encrypted code

Download/Program Stop Re-Program  
Check MCU Notice Delay 3 se

Auto reload the target file before each program  
 Reload and download when target file is modified

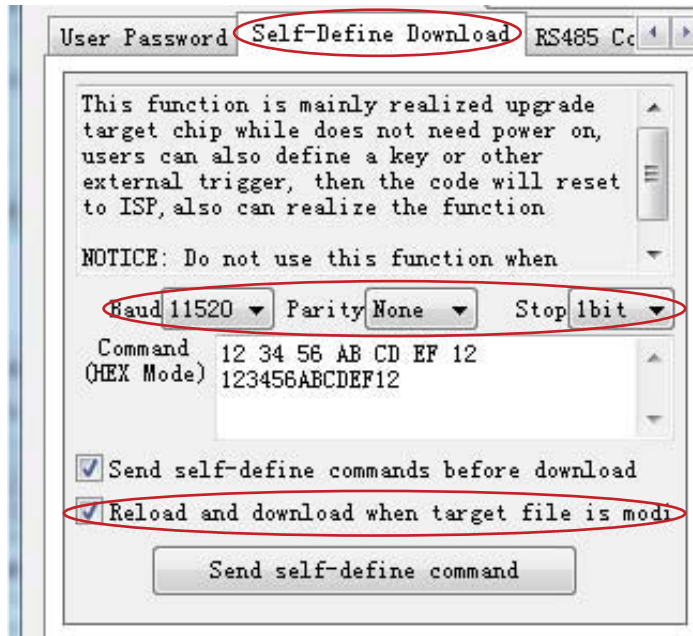
Create a new random key

The code ready to download is encrypted code  
Choice: Yes, it's encrypted, and MCU will use the built-in user key to decrypt the code automatically  
No-Choice: No, it's not encrypted, and MCU will program the code directly.

## 16.2.6 Self-Defined Download and Demo Program

This function is mainly realized upgrade target chip while does not need power on, users can also define a key or other external trigger, then the code will reset to ISP, also can realize the function.

If using the function, the PC-side application also need to make the following settings



NOTICE : Do not use this function when using the U8 programmer

```

/*-----*/
/* --- Exam Program using software to realize self-defined download -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/

```

```

#include <reg51.h>
#include <instrins.h>

```

```

sfr IAP_CONTR = 0xc7;
sbit MCU_Start_Led = P1^7;

```

---

```

#define Self_Define_ISP_Download_Command      0x22
#define RELOAD_COUNT          0xfb           //18.432MHz,12T,SMOD=0,9600bps
//#define RELOAD_COUNT        0xf6           //18.432MHz,12T,SMOD=0,4800bps
//#define RELOAD_COUNT        0xec           //18.432MHz,12T,SMOD=0,2400bps
//#define RELOAD_COUNT        0xd8           //18.432MHz,12T,SMOD=0,1200bps

```

```

void serial_port_initial(void);
void send_UART(unsigned char);
void UART_Interrupt_Receive(void);
void soft_reset_to_ISP_Monitor(void);
void delay(void);
void display_MCU_Start_Led(void);

```

```

void main(void)
{
    unsigned char i = 0;

    serial_port_initial();           //Initial UART
    display_MCU_Start_Led();         //Turn on the work LED
    send_UART(0x34);                 //Send UART test data
    send_UART(0xa7);                 // Send UART test data
    while (1);
}

```

```

void send_UART(unsigned char i)
{
    ES = 0;                          //Disable serial interrupt
    TI = 0;                          //Clear TI flag
    SBUF = i;                         //send this data
    while (!TI);                      //wait for the data is sent
    TI = 0;                          //clear TI flag
    ES = 1;                          //enable serial interrupt
}

```

```

void UART_Interrupt)Receive(void) interrupt 4 using 1
{
    unsigned char k = 0;
    if (RI)
    {
        RI = 0;
        k = SBUF;

        if (k == Self_Define_ISP_Command)           //check the serial data
        {
            delay();                               //delay 1s
            delay();                               //delay 1s
            soft_reset_to_ISP_Monitor();
        }
    }
}

```

---

---

```
        if (TI)
        {
            TI = 0;
        }
    }

void soft_reset_to_ISP_Monitor(void)
{
    IAP_CONTR = 0x60;           //0110,0000 soft reset system to run ISP monitor
}

void delay(void)
{
    unsigned int j = 0;
    unsigned int g = 0;
    for (j=0; j<5; j++)
    {
        for (g=0; g<60000; g++)
        {
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
        }
    }
}

void display_MCU_Start_Led(void)
{
    unsigned char i = 0;
    for (i=0; i<3; i++)
    {
        MCU_Start_Led = 0;      //Turn on work LED
        dejay();
        MCU_Start_Led = 1;      //Turn off work LED
        dejay();
        MCU_Start_Led = 0;      //Turn on work LED
    }
}
```



# 16.3 Emulator of STC15 series MCU

We provide specific emulator of STC15 series now. But for STC old MCU (such as STC12/11/10 series, STC89/90 series, STC15F204EA and STC15F104E series), we do not provide specific emulator, if you have a traditional 8051 emulator, you can use it to simulate STC old MCU's some 8052 basic functions.

## 1. Hardware Environment:

The present simulation is double CPU simulation: monitoring CPU and target CPU.

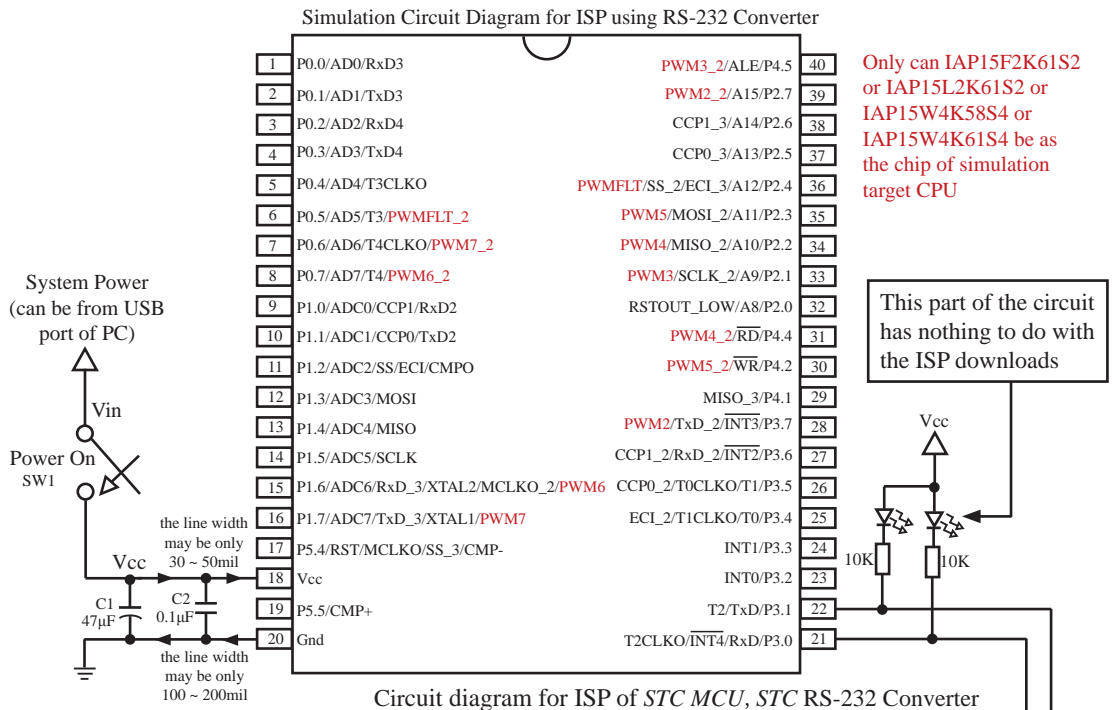
Monitoring CPU is in charge of communicating with Keil-C51 development environment and controlling the target CPU.

The chip of simulation target CPU must be IAP15F2K61S2 or IAP15L2K61S2 or IAP15W4K58S4 or IAP15W4K61S4.

Simulation target CPU can be directly welded on the user's system.

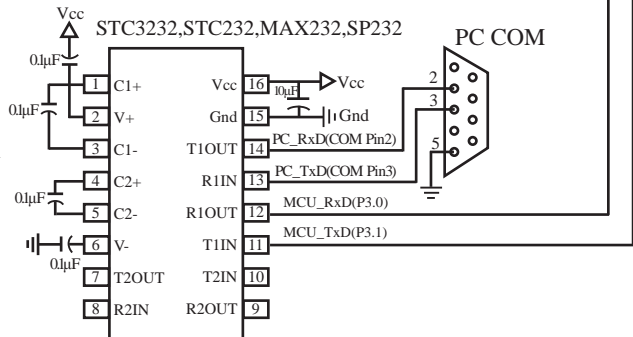
Monitoring CPU is designed in the monitoring CPU board sold by STC.

Recommend that the power of target CPU and user's system should be supplied by monitoring CPU.

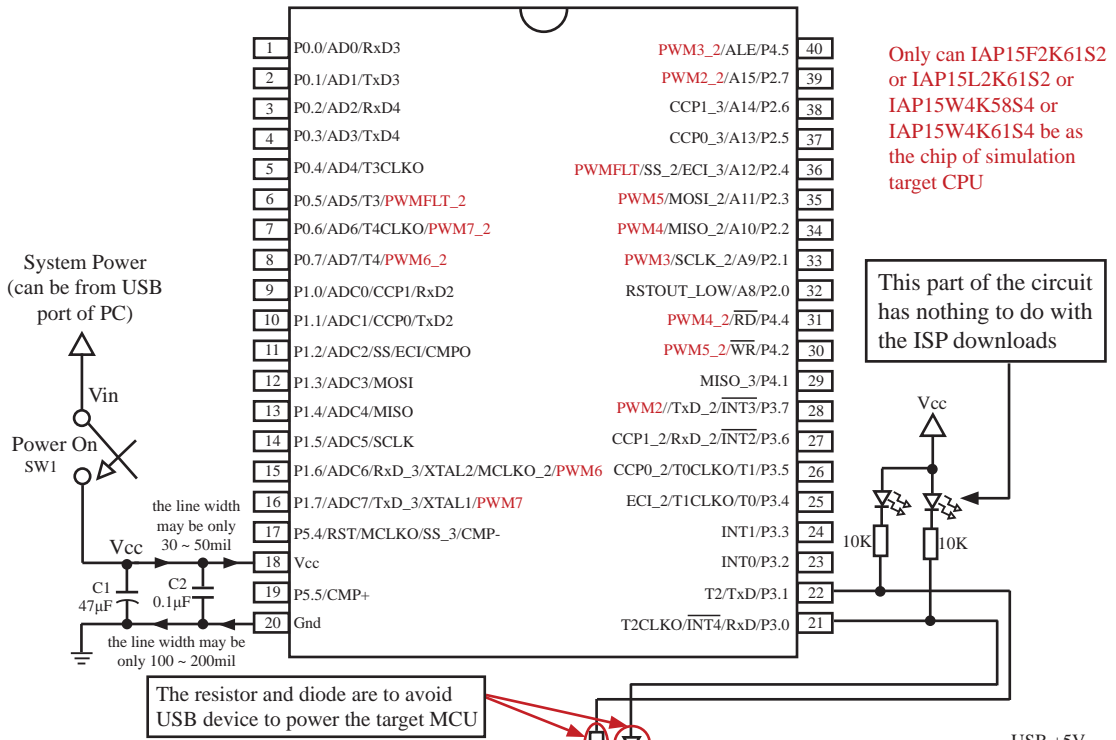


Note P0 ports can be multiplexed as Address/ Data bus not as A/D Converter. 8 channels of A/D Converter are on P1.

Consequently P0.x/ADx means that P0.x can be used as Address/Data bus, while P1.x/ADCx means P1.x can be used as A/D conversion channel in the pin map.



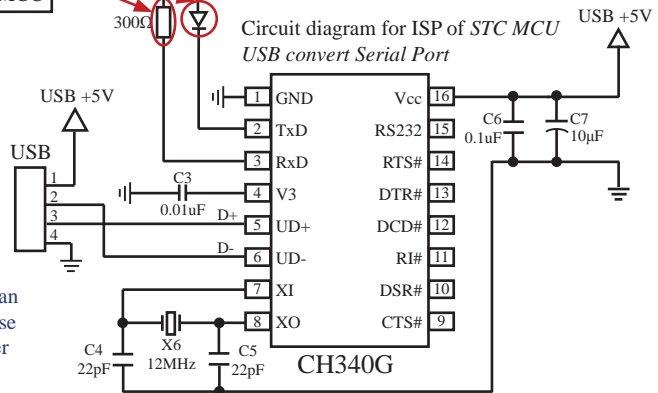
## Simulation Circuit Diagram for ISP using USB to convert Serial Port



Note P0 ports can be multiplexed as Address/ Data bus not as A/D Converter. 8 channels of A/D Converter are on P1.

Consequently P0.x/ADx means that P0.x can be used as Address/Data bus, while P1.x/ADCx means P1.x can be used as A/D conversion channel in the pin map.

Recommend to choose CH340G ( Its pins are not compatible with CH341's, but whose price less than RMB 1.1 yuan is more cheap), also you can choose PL2303(its price is less than RMB 1.0 yuan), refer to [www.wch.cn](http://www.wch.cn) for more detail.



Internal highly reliable Reset, so external reset circuit can be completely removed.

P5.4/RST/MCLKO pin factory defaults to the I/O port, which can be set as RST reset pin(active high) through the STC-ISP programmer.

Internal high-precise R/C clock ( $\pm 3\%$ ),  $\pm 1\%$  temperature drift ( $-40 \sim +85$ ) while  $\pm 0.6\%$  in normal temperature ( $-20 \sim +65$ ), so external expensive crystal can be completely removed.

Recommend to add decoupling capacitor C1(47 $\mu$ F) and C2(0.1 $\mu$ F) between Vcc and Gnd that can remove power noise and improve the anti-interference ability.

## 2. Software Environment:

The code of reset entrance can be written as follows by assembly.

```

ORG    0000H                ;entrance address of reset
LJMP   RESET                ;make use of LJMP instruction
...
ORG    100H                 ;other interrupt vectors
RESET:                ;address of user's code
...
                                ;reset entrance
                                ;user's code

```

## 3. Resources occupied by simulation code

Space of Program memory: The last 6K bytes of program memory is occupied by simulation code.

If utilizing IAP15F2K61S2/IAP15L2K61S2/IAP15W4K61S4 MCU to simulate, user program only can make use of 55K bytes (0x0000~0xDBFF) of program memory and not occupy the last 6K bytes (from 0xDC00 to 0xF3FF)

Common RAM(data,idata): 0 byte

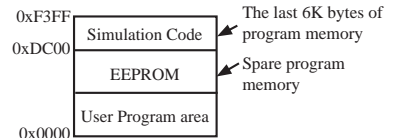
XRAM(xdata) : 768 bytes(0x0400 – 0x06FF, don't be occupied by user program)

I/O: P3.0 / P3.1

P3.0/INT4/T2CLKO and P3.1/T2 can not be used in user program.

For IAP series MCU, the EEPROM operation is achieved by using the spare program memory.

The EEPROM of IAP15F2K61S2 MCU is shown below.



## 4. STC-ISP Operating

## 5. Keil Setting

The image shows two overlapping dialog boxes in the Keil IDE. The background dialog is 'Options for Target 'Target 1'', and the foreground dialog is 'Target Setup'. Red circles and arrows highlight specific settings, with numbered steps 1 through 7 explaining them.

- Step 1:** Select the Debug option (in the 'Options for Target' dialog).
- Step 2:** Use hardware to simulate (selecting 'Use: Keil Monitor-51 Driver' in the 'Options for Target' dialog).
- Step 3:** Select Monitor-51 driver (selecting 'Settings' in the 'Options for Target' dialog).
- Step 4:** Select "Run To Main" option (selecting 'Run to main()' in the 'Options for Target' dialog).
- Step 5:** Set the "Monitor-51" (selecting 'COM1' and '9600' in the 'Target Setup' dialog).
- Step 6:** Set COM number and Baud-rate (selecting 'COM1' and '9600' in the 'Target Setup' dialog).
- Step 7:** Set the cache options (checking 'Cache DATA (SFR)', 'Cache XDATA', 'Cache IDATA', and 'Cache CODE' in the 'Target Setup' dialog).

## 6. Notice

- If a "Halt" order is carried out during running application program, the following dialog would be popup. Please click "Continue" button to return running application

The dialog box displays the following text:

Failed to stop application!

**Continue** Return to running application.

**Stop Debugging** Stops debugging session

Notes:

- Here are some reasons why this message appears:
  - The application has disabled the serial or global
  - The serial interface on the target system is not an interrupt (e.g. when using an external UART).
  - The target system does not work because of a

```

65:          _nop_();
C:0x0113    00      NOP
66:      }
C:0x0114    E511    MOV     A,0x11
C:0x0116    1511    DEC     0x11
C:0x0118    7002    JNZ     C:011C
C:0x011A    1510    DEC     0x10
C:0x011C    80C3    SJMP   C:00E1
67:  }
C:0x011E    22      RET
9: void tm0() interrupt 1 using 1
10: {
11:     static unsigned char i=0;
C:0x011F    C0E0    PUSH   ACC(0xE0)
C:0x0121    C0D0    PUSH   PSW(0xD0)
C:0x0123    75D008  MOV    PSW(0xD0),#0x08
12:     TH0 = T256Hz >> 8;
C:0x0126    758CE8  MOV    TH0(0x8C),#0xE8
13:     TLO = T256Hz;
C:0x0129    758A90  MOV    TLO(0x8A),#P1(0x90)
14:     while (i-- == 0)
C:0x012C    AF12    MOV    R7,0x12
C:0x012E    1512    DEC    0x12

```

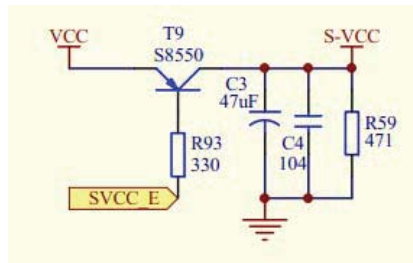
---

# Chapter 17 How to Program Slave Chip by Master Chip

## ——the Slave Chip is only for STC15 series MCU

When utilizing master chip (such as single chip, ARM, DSP and so on) to program the slave chip (STC15 series MCU) by tool STC-ISP Writer/Programmer, you must first stop the slave chip, and then, send download instruction to slave chip (namely STC15 series MCU) by master chip, lastly, give the slave chip for the power-on from master chip. Only by doing so can you utilize master chip (such as single chip, ARM, DSP and so on) to program the STC15 series MCU (Slave chip) by tool STC-ISP Writer/Programmer correctly.

Because master chip (such as single chip, ARM, DSP and so on) need to control the slave chip (STC15 series MCU) to power on during the process of utilizing master chip to program the slave chip by tool STC-ISP Writer/Programmer, That the power switch of the slave chip circuit can be controlled by any one of I/O ports of master chip. The circuit diagram of power supply for slave chip (STC15 series MCU) is shown below, for you reference.



To control the slave chip (STC15 series MCU) to power on by master chip (such as single chip, ARM, DSP and so on), you can connect the SVCC\_E in above figure to any one of I/O ports of master chip.

The demo code is shown below that utilizing master chip (such as single chip, ARM, DSP and so on) to program the slave chip (STC15 series MCU) by tool STC-ISP Writer/Programmer :

```
/*-----*/
/* --- Demo code of utilizing master chip to program the slave chip (STC15 series MCU) -----*/
/* If you want to use the program or the program referenced in the -----*/
/* article, please specify in which data and procedures from STC -----*/
/*---- In Keil C development environment, select the Intel 8052 to compiling -----*/
/*---- And only contain <reg51.h> as header file -----*/
/*-----*/
```

```
//suppose the frequency of test chip is 11.0592MHz
```

---

// Note : When utilizing master chip (such as single chip, ARM, DSP and so on) to program the slave chip (STC15 series MCU) by tool STC-ISP Writer/Programmer, you must first stop the slave chip, and then, send download instruction to slave chip (namely STC15 series MCU) by master chip, lastly, give the slave chip for the power on from master chip Download

#include "reg51.h"

typedef bit BOOL;  
typedef unsigned char BYTE;  
typedef unsigned short WORD;  
typedef unsigned long DWORD;

//Define macro and constant

#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 //Operating Frequency of master chip  
#define BR(n) (65536 - FOSC/4/(n)) //Baud generate of master chip UART  
#define T1MS (65536 - FOSC/1000) //Initial value of 1ms of master chip timer

##define FUSER 11059200L //Operating Frequency of STC15 series target chip  
##define FUSER 12000000L //Operating Frequency of STC15 series target chip  
##define FUSER 18432000L //Operating Frequency of STC15 series target chip  
##define FUSER 22118400L //Operating Frequency of STC15 series target chip  
#define FUSER 24000000L //Operating Frequency of STC15 series target chip  
#define RL(n) (65536 - FUSER/4/(n)) //Baud generate of STC15 series target chip UART

//Define SFR

sfr AUXR = 0x8e;

//Define variable

BOOL f1ms; //Flag bit of 1ms  
BOOL UartBusy; //Flag bit of UART busy  
BOOL UartReceived; //Flag bit of UART received  
BYTE UartRecvStep; //data controlling receiving by UART  
BYTE TimeOut; //Timeout counter of UART  
BYTE xdata TxBuffer[256]; //Data buffer to be sended by UART  
BYTE xdata RxBuffer[256]; //Data buffer to be received by UART  
char code DEMO[256]; //Demo code data

---

```
//Declare the function
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
void main(void)
{
    while (1)
    {
        Initial();

        if (Download(DEMO, 0x0100))
        {
            //Downlaod 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;
            DelayXms(500);
            P3 = 0x00;
            DelayXms(500);
            P3 = 0xff;
        }

        else
        {
            //Download unsuccessfully
            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;
    }
}

```

```

//Interrupt service routine of 1ms Timer
void tm0(void) interrupt 1 using 1
{
    static BYTE Counter100;

    flms = TRUE;
    if (Counter100-- == 0)
    {
        Counter100 = 100;
        if (TimeOut) TimeOut--;
    }
}

```

```

//interrupt service routine of UART
void uart(void) interrupt 4 using 1
{
    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;
}
}

```

---

```

}

//Initialize system
void Initial(void)
{
    UartBusy = FALSE;

    SCON = 0xd0;                //UART mode must be 8 bits data +1 bit parity-check
    AUXR = 0xc0;
    TMOD = 0x00;
    TH0 = HIBYTE(T1MS);
    TL0 = LOBYTE(T1MS);
    TR0 = 1;
    TH1 = HIBYTE(BR(MINBAUD));
    TL1 = LOBYTE(BR(MINBAUD));
    TR1 = 1;
    ET0 = 1;
    ES = 1;
    EA = 1;
}

//Xms Delay program
void DelayXms(WORD x)
{
    do
    {
        f1ms = FALSE;
        while (!f1ms);
    } while (x--);
}

//Send program of UART data
BYTE UartSend(BYTE dat)
{
    while (UartBusy);

    UartBusy = TRUE;
    ACC = dat;
    TB8 = P;
    SBUF = ACC;

    return dat;
}

```

---

---

```

//Initialize UART
void CommInit(void)
{
    UartRecvStep = 0;
    TimeOut = 20;
    UartReceived = FALSE;
}

//Send UART data
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();
}

//program the STC15 series MCU
BOOL Download(BYTE *pdat, long size)
{
    BYTE arg;
    BYTE cnt;
    WORD addr;

    //Handsake
    CommInit();
    while (1)
    {
        if (UartRecvStep == 0)

```

---

```

        {
            UartSend(0x7f);
            DelayXms(10);
        }
        if (UartReceived)
        {
            arg = RxBuffer[4];
            if (RxBuffer[0] == 0x50) break;
            return FALSE;
        }
    }

//Set parameter
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] = 0xc3;
CommSend(8);
while (1)
{
    if (TimeOut == 0) return FALSE;
    if (UartReceived)
    {
        if (RxBuffer[0] == 0x01) break;
        return FALSE;
    }
}

//make preparations
TH1 = HIBYTE(BR(MAXBAUD));
TL1 = LOBYTE(BR(MAXBAUD));
DelayXms(10);
TxBuffer[0] = 0x05;
CommSend(1);
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;
CommSend(2);
TimeOut = 100;
while (1)
{
    if (TimeOut == 0) return FALSE;
    if (UartReceived)
    {
        if (RxBuffer[0] == 0x03) break;
        return FALSE;
    }
}
//write the code
DelayXms(10);
addr = 0;
TxBuffer[0] = 0x22;
while (addr < size)
{
    TxBuffer[1] = HIBYTE(addr);
    TxBuffer[2] = LOBYTE(addr);
    cnt = 0;
    while (addr < size)
    {
        TxBuffer[cnt+3] = pdat[addr];
        addr++;
        cnt++;
        if (cnt >= 128) break;
    }
    CommSend(cnt + 3);
    while (1)
    {
        if (TimeOut == 0) return FALSE;
        if (UartReceived)
        {
            if ((RxBuffer[0] == 0x02) && (RxBuffer[1] == 'T')) break;
            return FALSE;
        }
    }
}

```

---

---

```

        TxBuffer[0] = 0x02;
    }

    DelayXms(10);
    for (cnt=0; cnt<128; cnt++)
    {
        TxBuffer[cnt] = 0xff;
    }
    TxBuffer[0] = 0x04;
    TxBuffer[1] = 0x00;
    TxBuffer[2] = 0x00;
    TxBuffer[34] = 0xfd;
    TxBuffer[62] = arg;
    TxBuffer[63] = 0x7f;
    TxBuffer[64] = 0xf7;
    TxBuffer[65] = 0x7b;
    TxBuffer[66] = 0x1f;
    CommSend(67);
    while (1)
    {
        if (TimeOut == 0) return FALSE;
        if (UartReceived)
        {
            if ((RxBuffer[0] == 0x04) && (RxBuffer[1] == 'T')) break;
            return FALSE;
        }
    }

    //download complete
    return TRUE;
}

char code DEMO[256] =
{
    0x02,  0x00,  0x5E,  0x12,  0x00,  0x4B,  0x75,  0xB0,
    0xEF,  0x12,  0x00,  0x2C,  0x75,  0xB0,  0xDF,  0x12,
    0x00,  0x2C,  0x75,  0xB0,  0xFE,  0x12,  0x00,  0x2C,
    0x75,  0xB0,  0xFD,  0x12,  0x00,  0x2C,  0x75,  0xB0,
    0xFB,  0x12,  0x00,  0x2C,  0x75,  0xB0,  0xF7,  0x12,
    0x00,  0x2C,  0x80,  0xDA,  0xE4,  0xFF,  0xFE,  0xE4,
    0xFD,  0xFC,  0x0D,  0xBD,  0x00,  0x01,  0x0C,  0xBC,
    0x01,  0xF8,  0xBD,  0xF4,  0xF5,  0x0F,  0xBF,  0x00,
    0x01,  0x0E,  0xBE,  0x03,  0xEA,  0xBF,  0xE8,  0xE7,
    0x02,  0x00,  0x4B,  0x75,  0x80,  0xFF,  0x75,  0x90,

```

---





---

# Appendix A: Assembly Language Programming

## INTRODUCTION

Assembly language is a computer language lying between the extremes of machine language and high-level language like Pascal or C use words and statements that are easily understood by humans, although still a long way from "natural" language. Machine language is the binary language of computers. A machine language program is a series of binary bytes representing instructions the computer can execute.

Assembly language replaces the binary codes of machine language with easy to remember "mnemonics" that facilitate programming. For example, an addition instruction in machine language might be represented by the code "10110011". It might be represented in assembly language by the mnemonic "ADD". Programming with mnemonics is obviously preferable to programming with binary codes.

Of course, this is not the whole story. Instructions operate on data, and the location of the data is specified by various "addressing modes" embedded in the binary code of the machine language instruction. So, there may be several variations of the ADD instruction, depending on what is added. The rules for specifying these variations are central to the theme of assembly language programming.

An assembly language program is not executable by a computer. Once written, the program must undergo translation to machine language. In the example above, the mnemonic "ADD" must be translated to the binary code "10110011". Depending on the complexity of the programming environment, this translation may involve one or more steps before an executable machine language program results. As a minimum, a program called an "assembler" is required to translate the instruction mnemonics to machine language binary codes. A further step may require a "linker" to combine portions of program from separate files and to set the address in memory at which the program may execute. We begin with a few definitions.

An assembly language program is a program written using labels, mnemonics, and so on, in which each statement corresponds to a machine instruction. Assembly language programs, often called source code or symbolic code, cannot be executed by a computer.

A machine language program is a program containing binary codes that represent instructions to a computer. Machine language programs, often called object code, are executable by a computer.

An assembler is a program that translates an assembly language program into a machine language program. The machine language program (object code) may be in "absolute" form or in "relocatable" form. In the latter case, "linking" is required to set the absolute address for execution.

A linker is a program that combines relocatable object programs (modules) and produces an absolute object program that is executable by a computer. A linker is sometimes called a "linker/locator" to reflect its separate functions of combining relocatable modules (linking) and setting the address for execution (locating).

A segment is a unit of code or data memory. A segment may be relocatable or absolute. A relocatable segment has a name, type, and other attributes that allow the linker to combine it with other partial segments, if required, and to correctly locate the segment. An absolute segment has no name and cannot be combined with other segments.

A module contains one or more segments or partial segments. A module has a name assigned by the user. The module definitions determine the scope of local symbols. An object file contains one or more modules. A module may be thought of as a "file" in many instances.

A program consists of a single absolute module, merging all absolute and relocatable segments from all input modules. A program contains only the binary codes for instructions (with address and data constants) that are understood by a computer.

---

## ASSEMBLER OPERATION

There are many assembler programs and other support programs available to facilitate the development of applications for the 8051 microcontroller. Intel's original MCS-51 family assembler, ASM51, is no longer available commercially. However, it set the standard to which the others are compared.

ASM51 is a powerful assembler with all the bells and whistles. It is available on Intel development systems and on the IBM PC family of microcomputers. Since these "host" computers contain a CPU chip other than the 8051, ASM51 is called a cross assembler. An 8051 source program may be written on the host computer (using any text editor) and may be assembled to an object file and listing file (using ASM51), but the program may not be executed. Since the host system's CPU chip is not an 8051, it does not understand the binary instruction in the object file. Execution on the host computer requires either hardware emulation or software simulation of the target CPU. A third possibility is to download the object program to an 8051-based target system for execution.

ASM51 is invoked from the system prompt by  
ASM51 source\_file [assembler\_controls]

The source file is assembled and any assembler controls specified take effect. The assembler receives a source file as input (e.g., PROGRAM.SRC) and generates an object file (PROGRAM.OBJ) and listing file (PROGRAM.LST) as output. This is illustrated in Figure 1.

Since most assemblers scan the source program twice in performing the translation to machine language, they are described as two-pass assemblers. The assembler uses a location counter as the address of instructions and the values for labels. The action of each pass is described below.

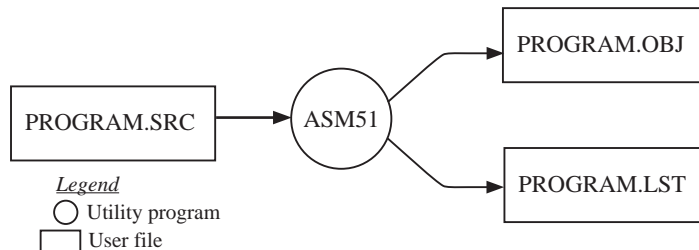


Figure 1 Assembling a source program

### Pass one

During the first pass, the source file is scanned line-by-line and a symbol table is built. The location counter defaults to 0 or is set by the ORG (set origin) directive. As the file is scanned, the location counter is incremented by the length of each instruction. Define data directives (DBs or DWs) increment the location counter by the number of bytes defined. Reserve memory directives (DSs) increment the location counter by the number of bytes reserved.

Each time a label is found at the beginning of a line, it is placed in the symbol table along with the current value of the location counter. Symbols that are defined using equate directives (EQUs) are placed in the symbol table along with the "equated" value. The symbol table is saved and then used during pass two.

### Pass two

During pass two, the object and listing files are created. Mnemonics are converted to opcodes and placed in the output files. Operands are evaluated and placed after the instruction opcodes. Where symbols appear in the operand field, their values are retrieved from the symbol table (created during pass one) and used in calculating the correct data or addresses for the instructions.

Since two passes are performed, the source program may use "forward references", that is, use a symbol before it is defined. This would occur, for example, in branching ahead in a program.

---

The object file, if it is absolute, contains only the binary bytes (00H-0FH) of the machine language program. A relocatable object file will also contain a symbol table and other information required for linking and locating. The listing file contains ASCII text codes (02H-7EH) for both the source program and the hexadecimal bytes in the machine language program.

A good demonstration of the distinction between an object file and a listing file is to display each on the host computer's CRT display (using, for example, the TYPE command on MS-DOS systems). The listing file clearly displays, with each line of output containing an address, opcode, and perhaps data, followed by the program statement from the source file. The listing file displays properly because it contains only ASCII text codes. Displaying the object file is a problem, however. The output will appear as "garbage", since the object file contains binary codes of an 8051 machine language program, rather than ASCII text codes.

## ASSEMBLY LANGUAGE PROGRAM FORMAT

Assembly language programs contain the following:

- Machine instructions
- Assembler directives
- Assembler controls
- Comments

Machine instructions are the familiar mnemonics of executable instructions (e.g., ANL). Assembler directives are instructions to the assembler program that define program structure, symbols, data, constants, and so on (e.g., ORG). Assembler controls set assembler modes and direct assembly flow (e.g., \$TITLE). Comments enhance the readability of programs by explaining the purpose and operation of instruction sequences.

Those lines containing machine instructions or assembler directives must be written following specific rules understood by the assembler. Each line is divided into "fields" separated by space or tab characters. The general format for each line is as follows:

```
[label:]  mnemonic  [operand]  [, operand]  [...]  [:comment]
```

Only the mnemonic field is mandatory. Many assemblers require the label field, if present, to begin on the left in column 1, and subsequent fields to be separated by space or tab characters. With ASM51, the label field needn't begin in column 1 and the mnemonic field needn't be on the same line as the label field. The operand field must, however, begin on the same line as the mnemonic field. The fields are described below.

### Label Field

A label represents the address of the instruction (or data) that follows. When branching to this instruction, this label is used in the operand field of the branch or jump instruction (e.g., SJMP SKIP).

Whereas the term "label" always represents an address, the term "symbol" is more general. Labels are one type of symbol and are identified by the requirement that they must terminate with a colon(:). Symbols are assigned values or attributes, using directives such as EQU, SEGMENT, BIT, DATA, etc. Symbols may be addresses, data constants, names of segments, or other constructs conceived by the programmer. Symbols do not terminate with a colon. In the example below, PAR is a symbol and START is a label (which is a type of symbol).

```
PAR      EQU      500          ;"PAR" IS A SYMBOL WHICH
                          ;REPRESENTS THE VALUE 500
START:   MOV      A,#0FFH     ;"START" IS A LABEL WHICH
                          ;REPRESENTS THE ADDRESS OF
                          ;THE MOV INSTRUCTION
```

A symbol (or label) must begin with a letter, question mark, or underscore (\_); must be followed by letters, digit, "?", or "\_"; and can contain up to 31 characters. Symbols may use upper- or lowercase characters, but they are treated the same. Reserved words (mnemonics, operators, predefined symbols, and directives) may not be used.

---

## Mnemonic Field

Instruction mnemonics or assembler directives go into mnemonic field, which follows the label field. Examples of instruction mnemonics are ADD, MOV, DIV, or INC. Examples of assembler directives are ORG, EQU, or DB.

## Operand Field

The operand field follows the mnemonic field. This field contains the address or data used by the instruction. A label may be used to represent the address of the data, or a symbol may be used to represent a data constant. The possibilities for the operand field are largely dependent on the operation. Some operations have no operand (e.g., the RET instruction), while others allow for multiple operands separated by commas. Indeed, the possibilities for the operand field are numerous, and we shall elaborate on these at length. But first, the comment field.

## Comment Field

Remarks to clarify the program go into comment field at the end of each line. Comments must begin with a semicolon (;). Each line may be comment lines by beginning them with a semicolon. Subroutines and large sections of a program generally begin with a comment block—several lines of comments that explain the general properties of the section of software that follows.

## Special Assembler Symbols

Special assembler symbols are used for the register-specific addressing modes. These include A, R0 through R7, DPTR, PC, C and AB. In addition, a dollar sign (\$) can be used to refer to the current value of the location counter. Some examples follow.

```
SETB  C
INC   DPTR
JNB   TI, $
```

The last instruction above makes effective use of ASM51's location counter to avoid using a label. It could also be written as

```
HERE: JNB   TI, HERE
```

## Indirect Address

For certain instructions, the operand field may specify a register that contains the address of the data. The commercial "at" sign (@) indicates address indirection and may only be used with R0, R1, the DPTR, or the PC, depending on the instruction. For example,

```
ADD   A, @R0
MOVC  A, @A+PC
```

The first instruction above retrieves a byte of data from internal RAM at the address specified in R0. The second instruction retrieves a byte of data from external code memory at the address formed by adding the contents of the accumulator to the program counter. Note that the value of the program counter, when the add takes place, is the address of the instruction following MOVC. For both instruction above, the value retrieved is placed into the accumulator.

## Immediate Data

Instructions using immediate addressing provide data in the operand field that become part of the instruction. Immediate data are preceded with a pound sign (#). For example,

---

```
CONSTANT    EQU    100
            MOV    A , #0FEH
            ORL    40H , #CONSTANT
```

All immediate data operations (except MOV DPTR,#data) require eight bits of data. The immediate data are evaluated as a 16-bit constant, and then the low-byte is used. All bits in the high-byte must be the same (00H or FFH) or the error message "value will not fit in a byte" is generated. For example, the following instructions are syntactically correct:

```
MOV    A , #0FF00H
MOV    A , #00FFH
```

But the following two instructions generate error messages:

```
MOV    A , #0FE00H
MOV    A , #01FFH
```

If signed decimal notation is used, constants from -256 to +255 may also be used. For example, the following two instructions are equivalent (and syntactically correct):

```
MOV    A , #-256
MOV    A , #0FF00H
```

Both instructions above put 00H into accumulator A.

## Data Address

Many instructions access memory locations using direct addressing and require an on-chip data memory address (00H to 7FH) or an SFR address (80H to 0FFH) in the operand field. Predefined symbols may be used for the SFR addresses. For example,

```
MOV    A , 45H
MOV    A , SBUF           ;SAME AS MOV A, 99H
```

## Bit Address

One of the most powerful features of the 8051 is the ability to access individual bits without the need for masking operations on bytes. Instructions accessing bit-addressable locations must provide a bit address in internal data memory (00h to 7FH) or a bit address in the SFRs (80H to 0FFH).

There are three ways to specify a bit address in an instruction: (a) explicitly by giving the address, (b) using the dot operator between the byte address and the bit position, and (c) using a predefined assembler symbol. Some examples follow.

```
SETB   0E7H           ;EXPLICIT BIT ADDRESS
SETB   ACC.7         ;DOT OPERATOR (SAME AS ABOVE)
JNB    TI , $        ;"TI" IS A PRE-DEFINED SYMBOL
JNB    99H , $       ;(SAME AS ABOVE)
```

## Code Address

A code address is used in the operand field for jump instructions, including relative jumps (SJMP and conditional jumps), absolute jumps and calls (ACALL, AJMP), and long jumps and calls (LJMP, LCALL).

The code address is usually given in the form of a label.

ASM51 will determine the correct code address and insert into the instruction the correct 8-bit signed offset, 11-bit page address, or 16-bit long address, as appropriate.

---

## Generic Jumps and Calls

ASM51 allows programmers to use a generic JMP or CALL mnemonic. "JMP" can be used instead of SJMP, AJMP or LJMP; and "CALL" can be used instead of ACALL or LCALL. The assembler converts the generic mnemonic to a "real" instruction following a few simple rules. The generic mnemonic converts to the short form (for JMP only) if no forward references are used and the jump destination is within -128 locations, or to the absolute form if no forward references are used and the instruction following the JMP or CALL instruction is in the same 2K block as the destination instruction. If short or absolute forms cannot be used, the conversion is to the long form.

The conversion is not necessarily the best programming choice. For example, if branching ahead a few instructions, the generic JMP will always convert to LJMP even though an SJMP is probably better. Consider the following assembled instructions sequence using three generic jumps.

LOC	OBJ	LINE	SOURCE		
1234		1		ORG	1234H
1234	04	2	START:	INC	A
1235	80FD	3		JMP	START ;ASSEMBLES AS SJMP
12FC		4		ORG	START + 200
12FC	4134	5		JMP	START ;ASSEMBLES AS AJMP
12FE	021301	6		JMP	FINISH ;ASSEMBLES AS LJMP
1301	04	7	FINISH:	INC	A
		8		END	

The first jump (line 3) assembles as SJMP because the destination is before the jump ( i.e., no forward reference) and the offset is less than -128. The ORG directive in line 4 creates a gap of 200 locations between the label START and the second jump, so the conversion on line 5 is to AJMP because the offset is too great for SJMP. Note also that the address following the second jump (12FEH) and the address of START (1234H) are within the same 2K page, which, for this instruction sequence, is bounded by 1000H and 17FFH. This criterion must be met for absolute addressing. The third jump assembles as LJMP because the destination (FINISH) is not yet defined when the jump is assembled (i.e., a forward reference is used). The reader can verify that the conversion is as stated by examining the object field for each jump instruction.

## ASSEMBLE-TIME EXPRESSION EVALUATION

Values and constants in the operand field may be expressed three ways: (a) explicitly (e.g.,0EFH), (b) with a predefined symbol (e.g., ACC), or (c) with an expression (e.g.,2 + 3). The use of expressions provides a powerful technique for making assembly language programs more readable and more flexible. When an expression is used, the assembler calculates a value and inserts it into the instruction.

All expression calculations are performed using 16-bit arithmetic; however, either 8 or 16 bits are inserted into the instruction as needed. For example, the following two instructions are the same:

```
MOV  DPTR,  #04FFH + 3
MOV  DPTR,  #0502H           ;ENTIRE 16-BIT RESULT USED
```

If the same expression is used in a "MOV A,#data" instruction, however, the error message "value will not fit in a byte" is generated by ASM51. An overview of the rules for evaluating expressions follows.

---

## Number Bases

The base for numeric constants is indicated in the usual way for Intel microprocessors. Constants must be followed with "B" for binary, "O" or "Q" for octal, "D" or nothing for decimal, or "H" for hexadecimal. For example, the following instructions are the same:

```
MOV  A, #15H
MOV  A, #1111B
MOV  A, #0FH
MOV  A, #17Q
MOV  A, #15D
```

Note that a digit must be the first character for hexadecimal constants in order to differentiate them from labels (i.e., "0A5H" not "A5H").

## Character Strings

Strings using one or two characters may be used as operands in expressions. The ASCII codes are converted to the binary equivalent by the assembler. Character constants are enclosed in single quotes (''). Some examples follow.

```
CJNE  A, #'Q', AGAIN
SUBB  A, #'0'           ;CONVERT ASCII DIGIT TO BINARY DIGIT
MOV   DPTR, #'AB'
MOV   DPTR, #4142H     ;SAME AS ABOVE
```

## Arithmetic Operators

The arithmetic operators are

```
+      addition
-      subtraction
*      multiplication
/      division
MOD    modulo (remainder after division)
```

For example, the following two instructions are same:

```
MOV  A, 10 +10H
MOV  A, #1AH
```

The following two instructions are also the same:

```
MOV  A, #25 MOD 7
MOV  A, #4
```

Since the MOD operator could be confused with a symbol, it must be separated from its operands by at least one space or tab character, or the operands must be enclosed in parentheses. The same applies for the other operators composed of letters.

## Logical Operators

The logical operators are

```
OR     logical OR
AND    logical AND
XOR    logical Exclusive OR
NOT    logical NOT (complement)
```

---

The operation is applied on the corresponding bits in each operand. The operator must be separated from the operands by space or tab characters. For example, the following two instructions are the same:

```
MOV  A, # '9' AND 0FH
MOV  A, #9
```

The NOT operator only takes one operand. The following three MOV instructions are the same:

```
THREE      EQU    3
MINUS_THREE EQU    -3
MOV        A, # (NOT THREE) + 1
MOV        A, #MINUS_THREE
MOV        A, #11111101B
```

## Special Operators

The special operators are

```
SHR  shift right
SHL  shift left
HIGH high-byte
LOW  low-byte
()   evaluate first
```

For example, the following two instructions are the same:

```
MOV  A, #8 SHL 1
MOV  A, #10H
```

The following two instructions are also the same:

```
MOV  A, #HIGH 1234H
MOV  A, #12H
```

## Relational Operators

When a relational operator is used between two operands, the result is always false (0000H) or true (FFFFH).

The operators are

```
EQ  =      equals
NE  <>     not equals
LT  <      less than
LE  <=     less than or equal to
GT  >      greater than
GE  >=     greater than or equal to
```

Note that for each operator, two forms are acceptable (e.g., "EQ" or "="). In the following examples, all relational tests are "true":

```
MOV  A, #5 = 5
MOV  A, #5 NE 4
MOV  A, # 'X' LT 'Z'
MOV  A, # 'X' >= 'X'
MOV  A, # $ > 0
MOV  A, #100 GE 50
```



---

So, the assembled instructions are equal to

```
MOV    A, #0FFH
```

Even though expressions evaluate to 16-bit results (i.e., 0FFFFH), in the examples above only the low-order eight bits are used, since the instruction is a move byte operation. The result is not considered too big in this case, because as signed numbers the 16-bit value FFFFH and the 8-bit value FFH are the same (-1).

## Expression Examples

The following are examples of expressions and the values that result:

<b>Expression</b>	<b>Result</b>
'B' - 'A'	0001H
8/3	0002H
155 MOD 2	0001H
4 * 4	0010H
8 AND 7	0000H
NOT 1	FFFEH
'A' SHL 8	4100H
LOW 65535	00FFH
(8 + 1) * 2	0012H
5 EQ 4	0000H
'A' LT 'B'	FFFFH
3 <= 3	FFFFHss

A practical example that illustrates a common operation for timer initialization follows: Put -500 into Timer 1 registers TH1 and TL1. In using the HIGH and LOW operators, a good approach is

```
VALUE    EQU    -500
          MOV    TH1, #HIGH VALUE
          MOV    TL1, #LOW VALUE
```

The assembler converts -500 to the corresponding 16-bit value (FE0CH); then the HIGH and LOW operators extract the high (FEH) and low (0CH) bytes, as appropriate for each MOV instruction.

## Operator Precedence

The precedence of expression operators from highest to lowest is

```
( )
HIGH LOW
* / MOD SHL SHR
+ -
EQ NE LT LE GT GE = <> < <= > >=
NOT
AND
OR XOR
```

When operators of the same precedence are used, they are evaluated left to right.

Examples:

<b>Expression</b>	<b>Value</b>
HIGH ('A' SHL 8)	0041H
HIGH 'A' SHL 8	0000H
NOT 'A' - 1	FFBFH
'A' OR 'A' SHL 8	4141H

---

## ASSEMBLER DIRECTIVES

Assembler directives are instructions to the assembler program. They are not assembly language instructions executable by the target microprocessor. However, they are placed in the mnemonic field of the program. With the exception of DB and DW, they have no direct effect on the contents of memory.

ASM51 provides several categories of directives:

- Assembler state control (ORG, END, USING)
- Symbol definition (SEGMENT, EQU, SET, DATA, IDATA, XDATA, BIT, CODE)
- Storage initialization/reservation (DS, DBIT, DB, DW)
- Program linkage (PUBLIC, EXTRN, NAME)
- Segment selection (RSEG, CSEG, DSEG, ISEG, ESEG, XSEG)

Each assembler directive is presented below, ordered by category.

### Assembler State Control

**ORG (Set Origin)**      The format for the ORG (set origin) directive is

```
ORG     expression
```

The ORG directive alters the location counter to set a new program origin for statements that follow. A label is not permitted. Two examples follow.

```
ORG     100H                                 ;SET LOCATION COUNTER TO 100H
ORG     ($ + 1000H) AND 0F00H               ;SET TO NEXT 4K BOUNDARY
```

The ORG directive can be used in any segment type. If the current segment is absolute, the value will be an absolute address in the current segment. If a relocatable segment is active, the value of the ORG expression is treated as an offset from the base address of the current instance of the segment.

**End**                    The format of the END directive is

```
END
```

END should be the last statement in the source file. No label is permitted and nothing beyond the END statement is processed by the assembler.

**Using**                 The format of the USING directive is

```
USING   expression
```

This directive informs ASM51 of the currently active register bank. Subsequent uses of the predefined symbolic register addresses AR0 to AR7 will convert to the appropriate direct address for the active register bank. Consider the following sequence:

```
USING   3
PUSH    AR7
USING   1
PUSH    AR7
```

The first push above assembles to PUSH 1FH (R7 in bank 3), whereas the second push assembles to PUSH 0FH (R7 in bank 1).

Note that USING does not actually switch register banks; it only informs ASM51 of the active bank. Executing 8051 instructions is the only way to switch register banks. This is illustrated by modifying the example above as follows:

---

```

MOV   PSW, #00011000B      ;SELECT REGISTER BANK 3
USING 3
PUSH  AR7                  ;ASSEMBLE TO PUSH 1FH
MOV   PSW, #00001000B      ;SELECT REGISTER BANK 1
USING 1
PUSH  AR7                  ;ASSEMBLE TO PUSH 0FH

```

## Symbol Definition

The symbol definition directives create symbols that represent segment, registers, numbers, and addresses. None of these directives may be preceded by a label. Symbols defined by these directives may not have been previously defined and may not be redefined by any means. The SET directive is the only exception. Symbol definition directives are described below.

**Segment** The format for the SEGMENT directive is shown below.

```

symbol          SEGMENT      segment_type

```

The symbol is the name of a relocatable segment. In the use of segments, ASM51 is more complex than conventional assemblers, which generally support only "code" and "data" segment types. However, ASM51 defines additional segment types to accommodate the diverse memory spaces in the 8051. The following are the defined 8051 segment types (memory spaces):

- CODE (the code segment)
- XDATA (the external data space)
- DATA (the internal data space accessible by direct addressing, 00H–07H)
- IDATA (the entire internal data space accessible by indirect addressing, 00H–07H)
- BIT (the bit space; overlapping byte locations 20H–2FH of the internal data space)

For example, the statement

```

EPROM          SEGMENT      CODE

```

declares the symbol EPROM to be a SEGMENT of type CODE. Note that this statement simply declares what EPROM is. To actually begin using this segment, the RSEG directive is used (see below).

**EQU (Equate)** The format for the EQU directive is

```

Symbol          EQU          expression

```

The EQU directive assigns a numeric value to a specified symbol name. The symbol must be a valid symbol name, and the expression must conform to the rules described earlier.

The following are examples of the EQU directive:

```

N27            EQU    27          ;SET N27 TO THE VALUE 27
HERE           EQU    $          ;SET "HERE" TO THE VALUE OF
                                ;THE LOCATION COUNTER
CR             EQU    0DH        ;SET CR (CARRIAGE RETURN) TO 0DH
MESSAGE:      DB    'This is a message'
LENGTH        EQU    $ - MESSAGE ;"LENGTH" EQUALS LENGTH OF "MESSAGE"

```

**Other Symbol Definition Directives** The SET directive is similar to the EQU directive except the symbol may be redefined later, using another SET directive.

---

The DATA, IDATA, XDATA, BIT, and CODE directives assign addresses of the corresponding segment type to a symbol. These directives are not essential. A similar effect can be achieved using the EQU directive; if used, however, they evoke powerful type-checking by ASM51. Consider the following two directives and four instructions:

```
FLAG1      EQU    05H
FLAG2      BIT    05H
           SETB   FLAG1
           SETB   FLAG2
           MOV    FLAG1, #0
           MOV    FLAG2, #0
```

The use of FLAG2 in the last instruction in this sequence will generate a "data segment address expected" error message from ASM51. Since FLAG2 is defined as a bit address (using the BIT directive), it can be used in a set bit instruction, but it cannot be used in a move byte instruction. Hence, the error. Even though FLAG1 represents the same value (05H), it was defined using EQU and does not have an associated address space. This is not an advantage of EQU, but rather, a disadvantage. By properly defining address symbols for use in a specific memory space (using the directives BIT, DATA, XDATA, ect.), the programmer takes advantage of ASM51's powerful type-checking and avoids bugs from the misuse of symbols.

### Storage Initialization/Reservation

The storage initialization and reservation directives initialize and reserve space in either word, byte, or bit units. The space reserved starts at the location indicated by the current value of the location counter in the currently active segment. These directives may be preceded by a label. The storage initialization/reservation directives are described below.

**DS (Define Storage)** The format for the DS (define storage) directive is

```
[label:] DS expression
```

The DS directive reserves space in byte units. It can be used in any segment type except BIT. The expression must be a valid assemble-time expression with no forward references and no relocatable or external references. When a DS statement is encountered in a program, the location counter of the current segment is incremented by the value of the expression. The sum of the location counter and the specified expression should not exceed the limitations of the current address space.

The following statement create a 40-byte buffer in the internal data segment:

```
           DSEG   AT    30H      ;PUT IN DATA SEGMENT (ABSOLUTE, INTERNAL)
LENGTH    EQU    40
BUFFER:   DS     LENGRH      ;40 BYTES RESERVED
```

The label BUFFER represents the address of the first location of reserved memory. For this example, the buffer begins at address 30H because "AT 30H" is specified with DSEG. The buffer could be cleared using the following instruction sequence:

```
           MOV    R7, #LENGTH
           MOV    R0, #BUFFER
LOOP:     MOV    @R0, #0
           DJNZ   R7, LOOP
           (continue)
```

---

To create a 1000-byte buffer in external RAM starting at 4000H, the following directives could be used:

```
XSTART      EQU    4000H
XLENGTH     EQU    1000
            XSEG    AT    XSTART
XBUFFER:    DS    XLENGTH
```

This buffer could be cleared with the following instruction sequence:

```
        MOV    DPTR, #XBUFFER
LOOP:   CLR    A
        MOVX   @DPTR, A
        INC    DPTR
        MOV    A, DPL
        CJNE  A, #LOW (XBUFFER + XLENGTH + 1), LOOP
        MOV    A, DPH
        CJNE  A, #HIGH (XBUFFER + XLENGTH + 1), LOOP
        (continue)
```

This is an excellent example of a powerful use of ASM51's operators and assemble-time expressions. Since an instruction does not exist to compare the data pointer with an immediate value, the operation must be fabricated from available instructions. Two compares are required, one each for the high- and low-bytes of the DPTR. Furthermore, the compare-and-jump-if-not-equal instruction works only with the accumulator or a register, so the data pointer bytes must be moved into the accumulator before the CJNE instruction. The loop terminates only when the data pointer has reached XBUFFER + LENGTH + 1. (The "+1" is needed because the data pointer is incremented after the last MOVX instruction.)

**DBIT** The format for the DBIT (define bit) directive is,

```
[label:]    DBIT    expression
```

The DBIT directive reserves space in bit units. It can be used only in a BIT segment. The expression must be a valid assemble-time expression with no forward references. When the DBIT statement is encountered in a program, the location counter of the current (BIT) segment is incremented by the value of the expression. Note that in a BIT segment, the basic unit of the location counter is bits rather than bytes. The following directives create three flags in an absolute bit segment:

```
        BSEG           ;BIT SEGMENT (ABSOLUTE)
KEFLAG:  DBIT    1    ;KEYBOARD STATUS
PRFLAG:  DBIT    1    ;PRINTER STATUS
DKFLAG:  DBIT    1    ;DISK STATUS
```

Since an address is not specified with BSEG in the example above, the address of the flags defined by DBIT could be determined (if one wishes to do so) by examining the symbol table in the .LST or .M51 files. If the definitions above were the first use of BSEG, then KBFLAG would be at bit address 00H (bit 0 of byte address 20H). If other bits were defined previously using BSEG, then the definitions above would follow the last bit defined.

**DB (Define Byte)** The format for the DB (define byte) directive is,

```
[label:]    DB    expression [, expression] [...]
```

The DB directive initializes code memory with byte values. Since it is used to actually place data constants in code memory, a CODE segment must be active. The expression list is a series of one or more byte values (each of which may be an expression) separated by commas.

---

The DB directive permits character strings (enclosed in single quotes) longer than two characters as long as they are not part of an expression. Each character in the string is converted to the corresponding ASCII code. If a label is used, it is assigned the address of the first byte. For example, the following statements

```

                CSEG AT      0100H
SQUARES:      DB    0, 1, 4, 9, 16, 25           ;SQUARES OF NUMBERS 0-5
MESSAGE:      DB    'Login:', 0                 ;NULL-TERMINATED CHARACTER STRING

```

When assembled, result in the following hexadecimal memory assignments for external code memory:

Address	Contents
0100	00
0101	01
0102	04
0103	09
0104	10
0105	19
0106	4C
0107	6F
0108	67
0109	69
010A	6E
010B	3A
010C	00

**DW (Define Word)** The format for the DW (define word) directive is  
 [label:] DW expression [, expression] [...]

The DW directive is the same as the DB directive except two memory locations (16 bits) are assigned for each data item. For example, the statements

```

CSEG AT      200H
DW    $, 'A', 1234H, 2, 'BC'

```

result in the following hexadecimal memory assignments:

Address	Contents
0200	02
0201	00
0202	00
0203	41
0204	12
0205	34
0206	00
0207	02
0208	42
0209	43

## Program Linkage

Program linkage directives allow the separately assembled modules (files) to communicate by permitting intermodule references and the naming of modules. In the following discussion, a "module" can be considered a "file." (In fact, a module may encompass more than one file.)

---

**Public**        The format for the PUBLIC (public symbol) directive is

PUBLIC            symbol    [, symbol]   [...]

The PUBLIC directive allows the list of specified symbols to be known and used outside the currently assembled module. A symbol declared PUBLIC must be defined in the current module. Declaring it PUBLIC allows it to be referenced in another module. For example,

PUBLIC    INCHAR, OUTCHR, INLINE, OUTSTR

**Extrn**        The format for the EXTRN (external symbol) directive is

EXTRN            segment\_type (symbol [, symbol] [...], ...)

The EXTRN directive lists symbols to be referenced in the current module that are defined in other modules. The list of external symbols must have a segment type associated with each symbol in the list. (The segment types are CODE, XDATA, DATA, IDATA, BIT, and NUMBER. NUMBER is a type-less symbol defined by EQU.) The segment type indicates the way a symbol may be used. The information is important at link-time to ensure symbols are used properly in different modules.

The PUBLIC and EXTRN directives work together. Consider the two files, MAIN.SRC and MESSAGES.SRC. The subroutines HELLO and GOOD\_BYE are defined in the module MESSAGES but are made available to other modules using the PUBLIC directive. The subroutines are called in the module MAIN even though they are not defined there. The EXTRN directive declares that these symbols are defined in another module.

MAIN.SRC:

```
EXTRN            CODE (HELLO, GOOD_BYE)
...
CALL            HELLO
...
CALL            GOOD_BYE
...
END
```

MESSAGES.SRC:

```
PUBLIC            HELLO, GOOD_BYE
...
HELLO:           (begin subroutine)
...
RET
GOOD_BYE:        (begin subroutine)
...
RET
...
END
```

Neither MAIN.SRC nor MESSAGES.SRC is a complete program; they must be assembled separately and linked together to form an executable program. During linking, the external references are resolved with correct addresses inserted as the destination for the CALL instructions.

**Name**        The format for the NAME directive is

NAME    module\_name

---

All the usual rules for symbol names apply to module names. If a name is not provided, the module takes on the file name (without a drive or subdirectory specifier and without an extension). In the absence of any use of the NAME directive, a program will contain one module for each file. The concept of "modules," therefore, is somewhat cumbersome, at least for relatively small programming problems. Even programs of moderate size (encompassing, for example, several files complete with relocatable segments) needn't use the NAME directive and needn't pay any special attention to the concept of "modules." For this reason, it was mentioned in the definition that a module may be considered a "file," to simplify learning ASM51. However, for very large programs (several thousand lines of code, or more), it makes sense to partition the problem into modules, where, for example, each module may encompass several files containing routines having a common purpose.

## Segment Selection Directives

When the assembler encounters a segment selection directive, it diverts the following code or data into the selected segment until another segment is selected by a segment selection directive. The directive may select a previously defined relocatable segment or optionally create and select absolute segments.

**RSEG (Relocatable Segment)** The format for the RSEG (relocatable segment) directive is

RSEG                    segment\_name

Where "segment\_name" is the name of a relocatable segment previously defined with the SEGMENT directive. RSEG is a "segment selection" directive that diverts subsequent code or data into the named segment until another segment selection directive is encountered.

**Selecting Absolute Segments** RSEG selects a relocatable segment. An "absolute" segment, on the other hand, is selected using one of the directives:

CSEG    (AT address)  
DSEG    (AT address)  
ISEG    (AT address)  
BSEG    (AT address)  
XSEG    (AT address)

These directives select an absolute segment within the code, internal data, indirect internal data, bit, or external data address spaces, respectively. If an absolute address is provided (by indicating "AT address"), the assembler terminates the last absolute address segment, if any, of the specified segment type and creates a new absolute segment starting at that address. If an absolute address is not specified, the last absolute segment of the specified type is continued. If no absolute segment of this type was previously selected and the absolute address is omitted, a new segment is created starting at location 0. Forward references are not allowed and start addresses must be absolute.

Each segment has its own location counter, which is always set to 0 initially. The default segment is an absolute code segment; therefore, the initial state of the assembler is location 0000H in the absolute code segment. When another segment is chosen for the first time, the location counter of the former segment retains the last active value. When that former segment is reselected, the location counter picks up at the last active value. The ORG directive may be used to change the location counter within the currently selected segment.

## ASSEMBLER CONTROLS

Assembler controls establish the format of the listing and object files by regulating the actions of ASM51. For the most part, assembler controls affect the look of the listing file, without having any effect on the program itself. They can be entered on the invocation line when a program is assembled, or they can be placed in the source file. Assembler controls appearing in the source file must be preceded with a dollar sign and must begin in column 1.



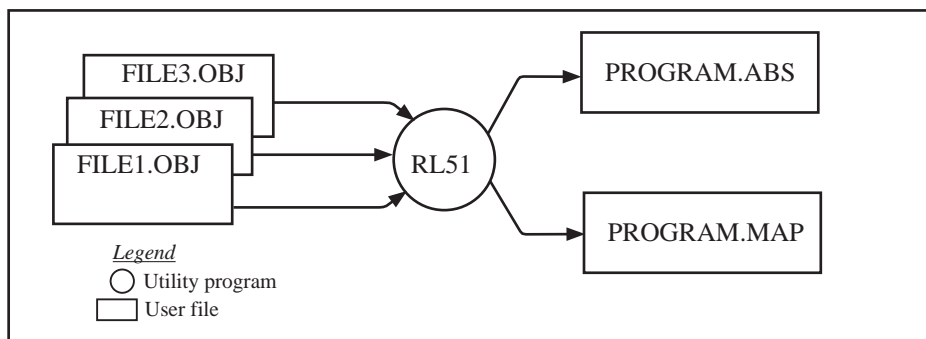
---

There are two categories of assembler controls: primary and general. Primary controls can be placed in the invocation line or at the beginning of the source program. Only other primary controls may precede a primary control. General controls may be placed anywhere in the source program.

## LINKER OPERATION

In developing large application programs, it is common to divide tasks into subprograms or modules containing sections of code (usually subroutines) that can be written separately from the overall program. The term "modular programming" refers to this programming strategy. Generally, modules are relocatable, meaning they are not intended for a specific address in the code or data space. A linking and locating program is needed to combine the modules into one absolute object module that can be executed.

Intel's RL51 is a typical linker/locator. It processes a series of relocatable object modules as input and creates an executable machine language program (PROGRAM, perhaps) and a listing file containing a memory map and symbol table (PROGRAM.M51). This is illustrated in following figure.



Linker operation

As relocatable modules are combined, all values for external symbols are resolved with values inserted into the output file. The linker is invoked from the system prompt by

```
RL51 input_list [T0 output_file] [location_controls]
```

The *input\_list* is a list of relocatable object modules (files) separated by commas. The *output\_list* is the name of the output absolute object module. If none is supplied, it defaults to the name of the first input file without any suffix. The *location\_controls* set start addresses for the named segments.

For example, suppose three modules or files (MAIN.OBJ, MESSAGES.OBJ, and SUBROUTINES.OBJ) are to be combined into an executable program (EXAMPLE), and that these modules each contain two relocatable segments, one called EPROM of type CODE, and the other called ONCHIP of type DATA. Suppose further that the code segment is to be executable at address 4000H and the data segment is to reside starting at address 30H (in internal RAM). The following linker invocation could be used:

```
RS51 MAIN.OBJ, MESSAGES.OBJ, SUBROUTINES.OBJ TO EXAMPLE & CODE  
(EPROM (4000H) DATA (ONCHIP (30H))
```

Note that the ampersand character "&" is used as the line continuation character.

If the program begins at the label START, and this is the first instruction in the MAIN module, then execution begins at address 4000H. If the MAIN module was not linked first, or if the label START is not at the beginning of MAIN, then the program's entry point can be determined by examining the symbol table in the listing file EXAMPLE.M51 created by RL51. By default, EXAMPLE.M51 will contain only the link map. If a symbol table is desired, then each source program must have used the SDEBUG control. The following table shows the assembler controls supported by ASM51.

Assembler controls supported by ASM51				
NAME	PRIMARY/ GENERAL	DEFAULT	ABBREV.	MEANING
DATE (date)	P	DATE( )	DA	Place string in header (9 char. max.)
DEBUG	P	NODEBUG	DB	Outputs debug symbol information to object file
EJECT	G	not applicable	EJ	Continue listing on next page
ERRORPRINT (file)	P	NOERRORPRINT	EP	Designates a file to receive error messages in addition to the listing file (defaults to console)
NOERRORPRINT	P	NOERRORPRINT	NOEP	Designates that error messages will be printed in listing file only
GEN	G	GENONLY	GO	List only the fully expanded source as if all lines generated by a macro call were already in the source file
GENONLY	G	GENONLY	NOGE	List only the original source text in the listing file
INCLUDED(file)	G	not applicable	IC	Designates a file to be included as part of the program
LIST	G	LIST	LI	Print subsequent lines of source code in listing file
NOLIST	G	LIST	NOLI	Do not print subsequent lines of source code in listing file
MACRO (men_precent)	P	MACRO(50)	MR	Evaluate and expand all macro calls. Allocate percentage of free memory for macro processing
NOMACRO	P	MACRO(50)	NOMR	Do not evaluate macro calls
MOD51	P	MOD51	MO	Recognize the 8051-specific predefined special function registers
NOMOD51	P	MOD51	NOMO	Do not recognize the 8051-specific predefined special function registers
OBJECT(file)	P	OBJECT(source.OBJ)	OJ	Designates file to receive object code
NOOBJECT	P	OBJECT(source.OBJ)	NOOJ	Designates that no object file will be created
PAGING	P	PAGING	PI	Designates that listing file be broken into pages and each will have a header
NOPAGING	P	PAGING	NOPI	Designates that listing file will contain no page breaks
PAGELNGTH (N)	P	PAGELNGT(60)	PL	Sets maximum number of lines in each page of listing file (range=10 to 65536)
PAGE WIDTH (N)	P	PAGEWIDTH(120)	PW	Set maximum number of characters in each line of listing file (range = 72 to 132)
PRINT(file)	P	PRINT(source.LST)	PR	Designates file to receive source listing
NOPRINT	P	PRINT(source.LST)	NOPR	Designates that no listing file will be created
SAVE	G	not applicable	SA	Stores current control settings from SAVE stack
RESTORE	G	not applicable	RS	Restores control settings from SAVE stack
REGISTERBANK (rb,...)	P	REGISTERBANK(0)	RB	Indicates one or more banks used in program module
NOREGISTER- BANK	P	REGISTERBANK(0)	NORB	Indicates that no register banks are used
SYMBOLS	P	SYMBOLS	SB	Creates a formatted table of all symbols used in program
NOSYMBOLS	P	SYMBOLS	NOSB	Designates that no symbol table is created
TITLE(string)	G	TITLE( )	TT	Places a string in all subsequent page headers (max.60 characters)
WORKFILES (path)	P	same as source	WF	Designates alternate path for temporary workfiles
XREF	P	NOXREF	XR	Creates a cross reference listing of all symbols used in program
NOXREF	P	NOXREF	NOXR	Designates that no cross reference list is created

---

## MACROS

The macro processing facility (MPL) of ASM51 is a "string replacement" facility. Macros allow frequently used sections of code be defined once using a simple mnemonic and used anywhere in the program by inserting the mnemonic. Programming using macros is a powerful extension of the techniques described thus far. Macros can be defined anywhere in a source program and subsequently used like any other instruction. The syntax for macro definition is

```
    %*DEFINE      (call_pattern)      (macro_body)
```

Once defined, the call pattern is like a mnemonic; it may be used like any assembly language instruction by placing it in the mnemonic field of a program. Macros are made distinct from "real" instructions by preceding them with a percent sign, "%". When the source program is assembled, everything within the macro-body, on a character-by-character basis, is substituted for the call-pattern. The mystique of macros is largely unfounded. They provide a simple means for replacing cumbersome instruction patterns with primitive, easy-to-remember mnemonics. The substitution, we reiterate, is on a character-by-character basis—nothing more, nothing less.

For example, if the following macro definition appears at the beginning of a source file,

```
    %*DEFINE      (PUSH_DPTR)
                  (PUSH  DPH
                   PUSH  DPL
                   )
```

then the statement

```
    %PUSH_DPTR
```

will appear in the .LST file as

```
    PUSH  DPH
    PUSH  DPL
```

The example above is a typical macro. Since the 8051 stack instructions operate only on direct addresses, pushing the data pointer requires two PUSH instructions. A similar macro can be created to POP the data pointer.

There are several distinct advantages in using macros:

- A source program using macros is more readable, since the macro mnemonic is generally more indicative of the intended operation than the equivalent assembler instructions.
- The source program is shorter and requires less typing.
- Using macros reduces bugs
- Using macros frees the programmer from dealing with low-level details.

The last two points above are related. Once a macro is written and debugged, it is used freely without the worry of bugs. In the PUSH\_DPTR example above, if PUSH and POP instructions are used rather than push and pop macros, the programmer may inadvertently reverse the order of the pushes or pops. (Was it the high-byte or low-byte that was pushed first?) This would create a bug. Using macros, however, the details are worked out once—when the macro is written—and the macro is used freely thereafter, without the worry of bugs.

Since the replacement is on a character-by-character basis, the macro definition should be carefully constructed with carriage returns, tabs, ect., to ensure proper alignment of the macro statements with the rest of the assembly language program. Some trial and error is required.

There are advanced features of ASM51's macro-processing facility that allow for parameter passing, local labels, repeat operations, assembly flow control, and so on. These are discussed below.

---

## Parameter Passing

A macro with parameters passed from the main program has the following modified format:

```
%*DEFINE      (macro_name (parameter_list)) (macro_body)
```

For example, if the following macro is defined,

```
%*DEFINE      (CMPA# (VALUE))
              (CJNE  A, #%VALUE, $ + 3
               )
```

then the macro call

```
%CMPA# (20H)
```

will expand to the following instruction in the .LST file:

```
CJNE  A, #20H, $ + 3
```

Although the 8051 does not have a "compare accumulator" instruction, one is easily created using the CJNE instruction with "\$+3" (the next instruction) as the destination for the conditional jump. The CMPA# mnemonic may be easier to remember for many programmers. Besides, use of the macro unburdens the programmer from remembering notational details, such as "\$+3."

Let's develop another example. It would be nice if the 8051 had instructions such as

```
JUMP  IF ACCUMULATOR GREATER THAN X
JUMP  IF ACCUMULATOR GREATER THAN OR EQUAL TO X
JUMP  IF ACCUMULATOR LESS THAN X
JUMP  IF ACCUMULATOR LESS THAN OR EQUAL TO X
```

but it does not. These operations can be created using CJNE followed by JC or JNC, but the details are tricky. Suppose, for example, it is desired to jump to the label GREATER\_THAN if the accumulator contains an ASCII code greater than "Z" (5AH). The following instruction sequence would work:

```
CJNE  A, #5BH, $÷3
JNC   GREATER_THAN
```

The CJNE instruction subtracts 5BH (i.e., "Z" + 1) from the content of A and sets or clears the carry flag accordingly. CJNE leaves C=1 for accumulator values 00H up to and including 5AH. (Note: 5AH-5BH<0, therefore C=1; but 5BH-5BH=0, therefore C=0.) Jumping to GREATER\_THAN on the condition "not carry" correctly jumps for accumulator values 5BH, 5CH, 5DH, and so on, up to FFH. Once details such as these are worked out, they can be simplified by inventing an appropriate mnemonic, defining a macro, and using the macro instead of the corresponding instruction sequence. Here's the definition for a "jump if greater than" macro:

```
%*DEFINE      (JGT (VALUE, LABEL))
              (CJNE  A, #%VALUE+1, $+3   ;JGT
               JNC   %LABEL
               )
```

To test if the accumulator contains an ASCII code greater than "Z," as just discussed, the macro would be called as

```
%JGT  ('Z', GREATER_THAN)
```

ASM51 would expand this into

```
CJNE  A, #5BH, $+3   ;JGT
JNC   GREATER_THAN
```

The JGT macro is an excellent example of a relevant and powerful use of macros. By using macros, the programmer benefits by using a meaningful mnemonic and avoiding messy and potentially bug-ridden details.

---

## Local Labels

Local labels may be used within a macro using the following format:

```
%*DEFINE      (macro_name [(parameter_list)])
                [LOCAL list_of_local_labels] (macro_body)
```

For example, the following macro definition

```
%*DEFINE      (DEC_DPTR) LOCAL SKIP
                (DEC  DPL                ;DECREMENT DATA POINTER
                 MOV  A, DPL
                 CJNE A, #0FFH, %SKIP
                 DEC  DPL
%SKIP:        )
```

would be called as

```
%DEC_DPTR
```

and would be expanded by ASM51 into

```
DEC  DPL                ;DECREMENT DATA POINTER
MOV  A, DPL
CJNE A, #0FFH, SKIP00
DEC  DPH
SKIP00:
```

Note that a local label generally will not conflict with the same label used elsewhere in the source program, since ASM51 appends a numeric code to the local label when the macro is expanded. Furthermore, the next use of the same local label receives the next numeric code, and so on.

The macro above has a potential "side effect." The accumulator is used as a temporary holding place for DPL. If the macro is used within a section of code that uses A for another purpose, the value in A would be lost. This side effect probably represents a bug in the program. The macro definition could guard against this by saving A on the stack. Here's an alternate definition for the DEC\_DPTR macro:

```
%*DEFINE      (DEC_DPTR) LOCAL SKIP
                (PUSHACC
                 DEC  DPL                ;DECREMENT DATA POINTER
                 MOV  A, DPL
                 CJNE A, #0FFH, %SKIP
                 DEC  DPH
%SKIP:        POP  ACC
                )
```

## Repeat Operations

This is one of several built-in (predefined) macros. The format is

```
%REPEAT      (expression)      (text)
```

For example, to fill a block of memory with 100 NOP instructions,

```
%REPEAT      (100)
(NOP
)
```

---

## Control Flow Operations

The conditional assembly of section of code is provided by ASM51's control flow macro definition. The format is

```
%IF (expression) THEN (balanced_text)
[ELSE (balanced_text)] FI
```

For example,

```
INTRENAL      EQU      1      ;1 = 8051 SERIAL I/O DRIVERS
                                   ;0 = 8251 SERIAL I/O DRIVERS
                                   .
                                   .
                                   %IF (INTERNAL) THEN
(INCHAR:      .                ;8051 DRIVERS
                                   .
OUTCHR:       .
                                   .
                                   ) ELSE
(INCHAR:      .                ;8251 DRIVERS
                                   .
OUTCHR:       .
                                   .
                                   )
```

In this example, the symbol INTERNAL is given the value 1 to select I/O subroutines for the 8051's serial port, or the value 0 to select I/O subroutines for an external UART, in this case the 8251. The IF macro causes ASM51 to assemble one set of drivers and skip over the other. Elsewhere in the program, the INCHAR and OUTCHR subroutines are used without consideration for the particular hardware configuration. As long as the program is assembled with the correct value for INTERNAL, the correct subroutine is executed.

---

# Appendix B: 8051 C Programming

## ADVANTAGES AND DISADVANTAGES OF 8051 C

The advantages of programming the 8051 in C as compared to assembly are:

- Offers all the benefits of high-level, structured programming languages such as C, including the ease of writing subroutines
- Often relieves the programmer of the hardware details that the compiler handles on behalf of the programmer
- Easier to write, especially for large and complex programs
- Produces more readable program source codes

Nevertheless, 8051 C, being very similar to the conventional C language, also suffers from the following disadvantages:

- Processes the disadvantages of high-level, structured programming languages.
- Generally generates larger machine codes
- Programmer has less control and less ability to directly interact with hardware

To compare between 8051 C and assembly language, consider the solutions to the Example—Write a program using Timer 0 to create a 1KHz square wave on P1.0.

A solution written below in 8051 C language:

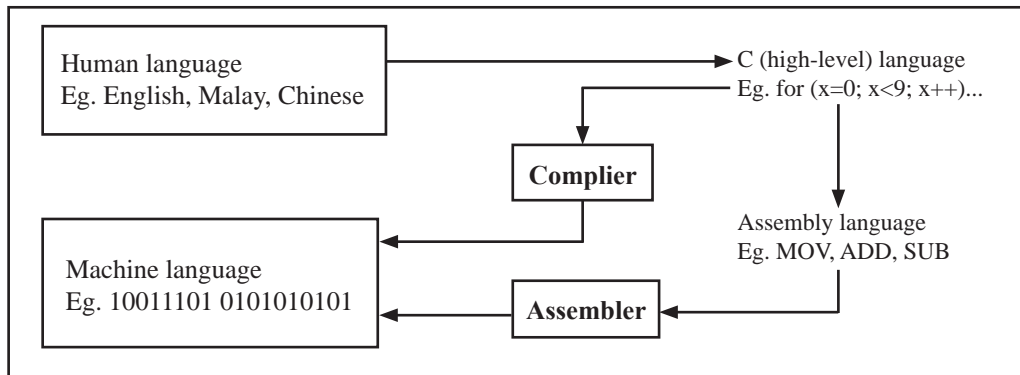
```
sbit portbit = P1^0;          /*Use variable portbit to refer to P1.0*/
main ()
{
    TMOD = 1;
    while (1)
    {
        TH0 = 0xFE;
        TL0 = 0xC;
        TR0 = 1;
        while (TF0 !=1);
        TR0 = 0;
        TF0 = 0;
        portbit = !(P1.^0);
    }
}
```

A solution written below in assembly language:

```
                ORG    8100H
                MOV    TMOD, #01H    ;16-bit timer mode
LOOP:          MOV    TH0, #0FEH    ;-500 (high byte)
                MOV    TL0, #0CH    ;-500 (low byte)
                SETB   TR0          ;start timer
WAIT:         JNB    TF0, WAIT      ;wait for overflow
                CLR    TR0          ;stop timer
                CLR    TF0          ;clear timer overflow flag
                CPL    P1.0         ;toggle port bit
                SJMP   LOOP         ;repeat
                END
```

Notice that both the assembly and C language solutions for the above example require almost the same number of lines. However, the difference lies in the readability of these programs. The C version seems more human than assembly, and is hence more readable. This often helps facilitate the human programmer's efforts to write even very complex programs. The assembly language version is more closely related to the machine code, and though less readable, often results in more compact machine code. As with this example, the resultant machine code from the assembly version takes 83 bytes while that of the C version requires 149 bytes, an increase of 79.5%!

The human programmer's choice of either high-level C language or assembly language for talking to the 8051, whose language is machine language, presents an interesting picture, as shown in following figure.



Conversion between human, high-level, assembly, and machine language

## 8051 C COMPILERS

We saw in the above figure that a compiler is needed to convert programs written in 8051 C language into machine language, just as an assembler is needed in the case of programs written in assembly language. A compiler basically acts just like an assembler, except that it is more complex since the difference between C and machine language is far greater than that between assembly and machine language. Hence the compiler faces a greater task to bridge that difference.

Currently, there exist various 8051 C compiler, which offer almost similar functions. All our examples and programs have been compiled and tested with Keil's  $\mu$  Vision 2 IDE by Keil Software, an integrated 8051 program development environment that includes its C51 cross compiler for C. A cross compiler is a compiler that normally runs on a platform such as IBM compatible PCs but is meant to compile programs into codes to be run on other platforms such as the 8051.

## DATA TYPES

8051 C is very much like the conventional C language, except that several extensions and adaptations have been made to make it suitable for the 8051 programming environment. The first concern for the 8051 C programmer is the data types. Recall that a data type is something we use to store data. Readers will be familiar with the basic C data types such as int, char, and float, which are used to create variables to store integers, characters, or floating-points. In 8051 C, all the basic C data types are supported, plus a few additional data types meant to be used specifically with the 8051.

The following table gives a list of the common data types used in 8051 C. The ones in bold are the specific 8051 extensions. The data type **bit** can be used to declare variables that reside in the 8051's bit-addressable locations (namely byte locations 20H to 2FH or bit locations 00H to 7FH). Obviously, these bit variables can only store bit values of either 0 or 1. As an example, the following C statement:

```
bit flag = 0;
```

declares a bit variable called flag and initializes it to 0.



---

### Data types used in 8051 C language

Data Type	Bits	Bytes	Value Range
<b>bit</b>	1		0 to 1
signed char	8	1	-128 to +127
unsigned char	8	1	0 to 255
enum	16	2	-32768 to +32767
signed short	16	2	-32768 to +32767
unsigned short	16	2	0 to 65535
signed int	16	2	-32768 to +32767
unsigned int	16	2	0 to 65535
signed long	32	4	-2,147,483,648 to +2,147,483,647
unsigned long	32	4	0 to 4,294,967,295
float	32	4	$\pm 1.175494E-38$ to $\pm 3.402823E+38$
<b>sbit</b>	1		0 to 1
<b>sfr</b>	8	1	0 to 255
<b>sfr16</b>	16	2	0 to 65535

The data type **sbit** is somewhat similar to the bit data type, except that it is normally used to declare 1-bit variables that reside in special function registers (SFRs). For example:

```
sbit    P = 0xD0;
```

declares the **sbit** variable P and specifies that it refers to bit address D0H, which is really the LSB of the PSW SFR. Notice the difference here in the usage of the assignment ("=") operator. In the context of **sbit** declarations, it indicates what address the **sbit** variable resides in, while in **bit** declarations, it is used to specify the initial value of the **bit** variable.

Besides directly assigning a bit address to an **sbit** variable, we could also use a previously defined **sfr** variable as the base address and assign our **sbit** variable to refer to a certain bit within that **sfr**. For example:

```
sfr     PSW = 0xD0;  
sbit    P = PSW^0;
```

This declares an **sfr** variable called PSW that refers to the byte address D0H and then uses it as the base address to refer to its LSB (bit 0). This is then assigned to an **sbit** variable, P. For this purpose, the caret symbol (^) is used to specify bit position 0 of the PSW.

A third alternative uses a constant byte address as the base address within which a certain bit is referred. As an illustration, the previous two statements can be replaced with the following:

```
sbit    P = 0xD0 ^ 0;
```

Meanwhile, the **sfr** data type is used to declare byte (8-bit) variables that are associated with SFRs. The statement:

```
sfr     IE = 0xA8;
```

declares an **sfr** variable IE that resides at byte address A8H. Recall that this address is where the Interrupt Enable (IE) SFR is located; therefore, the **sfr** data type is just a means to enable us to assign names for SFRs so that it is easier to remember.

The **sfr16** data type is very similar to **sfr** but, while the **sfr** data type is used for 8-bit SFRs, **sfr16** is used for 16-bit SFRs. For example, the following statement:

```
sfr16   DPTR = 0x82;
```

declares a 16-bit variable DPTR whose lower-byte address is at 82H. Checking through the 8051 architecture, we find that this is the address of the DPL SFR, so again, the **sfr16** data type makes it easier for us to refer to the SFRs by name rather than address. There's just one thing left to mention. When declaring **sbit**, **sfr**, or **sfr16** variables, remember to do so outside main, otherwise you will get an error.

In actual fact though, all the SFRs in the 8051, including the individual flag, status, and control bits in the bit-addressable SFRs have already been declared in an include file, called `reg51.h`, which comes packaged with most 8051 C compilers. By using `reg51.h`, we can refer for instance to the interrupt enable register as simply `IE` rather than having to specify the address `A8H`, and to the data pointer as `DPTR` rather than `82H`. All this makes 8051 C programs more human-readable and manageable. The contents of `reg51.h` are listed below.

```

/*-----
REG51.H
Header file for generic 8051 microcontroller.
-----*/

/* BYTE Register */
sfr  P0    = 0x80;
sfr  P1    = 0x90;
sfr  P2    = 0xA0;
sfr  P3    = 0xB0;
sfr  PSW   = 0xD0;
sfr  ACC   = 0xE0;
sfr  B     = 0xF0;
sfr  SP    = 0x81;
sfr  DPL   = 0x82;
sfr  DPH   = 0x83;
sfr  PCON  = 0x87;
sfr  TCON  = 0x88;
sfr  TMOD  = 0x89;
sfr  TL0   = 0x8A;
sfr  TL1   = 0x8B;
sfr  TH0   = 0x8C;
sfr  TH1   = 0x8D;
sfr  IE    = 0xA8;
sfr  IP    = 0xB8;
sfr  SCON  = 0x98;
sfr  SBUF  = 0x99;
/* BIT Register */
/* PSW */
sbit  CY    = 0xD7;
sbit  AC    = 0xD6;
sbit  F0    = 0xD5;
sbit  RS1   = 0xD4;
sbit  RS0   = 0xD3;
sbit  OV    = 0xD2;
sbit  P     = 0xD0;
/* TCON */
sbit  TF1   = 0x8F;
sbit  TR1   = 0x8E;
sbit  TF0   = 0x8D;
sbit  TR0   = 0x8C;

sbit  IE1   = 0x8B;
sbit  IT1   = 0x8A;
sbit  IE0   = 0x89;
sbit  IT0   = 0x88;
/* IE */
sbit  EA    = 0xAF;
sbit  ES    = 0xAC;
sbit  ET1   = 0xAB;
sbit  EX1   = 0xAA;
sbit  ET0   = 0xA9;
sbit  EX0   = 0xA8;
/* IP */
sbit  PS    = 0xBC;
sbit  PT1   = 0xBB;
sbit  PX1   = 0xBA;
sbit  PT0   = 0xB9;
sbit  PX0   = 0xB8;
/* P3 */
sbit  RD    = 0xB7;
sbit  WR    = 0xB6;
sbit  T1    = 0xB5;
sbit  T0    = 0xB4;
sbit  INT1  = 0xB3;
sbit  INT0  = 0xB2;
sbit  TXD   = 0xB1;
sbit  RXD   = 0xB0;
/* SCON */
sbit  SM0   = 0x9F;
sbit  SM1   = 0x9E;
sbit  SM2   = 0x9D;
sbit  REN   = 0x9C;
sbit  TB8   = 0x9B;
sbit  RB8   = 0x9A;
sbit  TI    = 0x99;
sbit  RI    = 0x98;

```

---

## MEMORY TYPES AND MODELS

The 8051 has various types of memory space, including internal and external code and data memory. When declaring variables, it is hence reasonable to wonder in which type of memory those variables would reside. For this purpose, several memory type specifiers are available for use, as shown in following table.

Memory types used in 8051 C language	
Memory Type	Description (Size)
code	Code memory (64 Kbytes)
data	Directly addressable internal data memory (128 bytes)
idata	Indirectly addressable internal data memory (256 bytes)
bdata	Bit-addressable internal data memory (16 bytes)
xdata	External data memory (64 Kbytes)
pdata	Paged external data memory (256 bytes)

The first memory type specifier given in above table is **code**. This is used to specify that a variable is to reside in code memory, which has a range of up to 64 Kbytes. For example:

```
char    code    errmsg[ ] = "An error occurred" ;
```

declares a char array called errmsg that resides in code memory.

If you want to put a variable into data memory, then use either of the remaining five data memory specifiers in above table. Though the choice rests on you, bear in mind that each type of data memory affect the speed of access and the size of available data memory. For instance, consider the following declarations:

```
signed int  data  num1;  
bit  bdata  numbit;  
unsigned int  xdata  num2;
```

The first statement creates a signed int variable num1 that resides in internal **data** memory (00H to 7FH). The next line declares a bit variable numbit that is to reside in the bit-addressable memory locations (byte addresses 20H to 2FH), also known as **bdata**. Finally, the last line declares an unsigned int variable called num2 that resides in external data memory, **xdata**. Having a variable located in the directly addressable internal data memory speeds up access considerably; hence, for programs that are time-critical, the variables should be of type **data**. For other variants such as 8052 with internal data memory up to 256 bytes, the **idata** specifier may be used. Note however that this is slower than data since it must use indirect addressing. Meanwhile, if you would rather have your variables reside in external memory, you have the choice of declaring them as **pdata** or **xdata**. A variable declared to be in **pdata** resides in the first 256 bytes (a page) of external memory, while if more storage is required, **xdata** should be used, which allows for accessing up to 64 Kbytes of external data memory.

What if when declaring a variable you forget to explicitly specify what type of memory it should reside in, or you wish that all variables are assigned a default memory type without having to specify them one by one? In this case, we make use of **memory models**. The following table lists the various memory models that you can use.

Memory models used in 8051 C language	
Memory Model	Description
Small	Variables default to the internal data memory (data)
Compact	Variables default to the first 256 bytes of external data memory (pdata)
Large	Variables default to external data memory (xdata)

---

A program is explicitly selected to be in a certain memory model by using the C directive, #pragma. Otherwise, the default memory model is **small**. It is recommended that programs use the small memory model as it allows for the fastest possible access by defaulting all variables to reside in internal data memory.

The **compact** memory model causes all variables to default to the first page of external data memory while the **large** memory model causes all variables to default to the full external data memory range of up to 64 Kbytes.

## ARRAYS

Often, a group of variables used to store data of the same type need to be grouped together for better readability. For example, the ASCII table for decimal digits would be as shown below.

ASCII table for decimal digits	
Decimal Digit	ASCII Code In Hex
0	30H
1	31H
2	32H
3	33H
4	34H
5	35H
6	36H
7	37H
8	38H
9	39H

To store such a table in an 8051 C program, an array could be used. An array is a group of variables of the same data type, all of which could be accessed by using the name of the array along with an appropriate index.

The array to store the decimal ASCII table is:

```
int    table [10] =
    {0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39};
```

Notice that all the elements of an array are separated by commas. To access an individual element, an index starting from 0 is used. For instance, table[0] refers to the first element while table[9] refers to the last element in this ASCII table.

## STRUCTURES

Sometime it is also desired that variables of different data types but which are related to each other in some way be grouped together. For example, the name, age, and date of birth of a person would be stored in different types of variables, but all refer to the person's personal details. In such a case, a structure can be declared. A structure is a group of related variables that could be of different data types. Such a structure is declared by:

```
struct    person {
            char name;
            int age;
            long DOB;
        };
```

Once such a structure has been declared, it can be used like a data type specifier to create structure variables that have the member's name, age, and DOB. For example:

```
struct    person    grace = {"Grace", 22, 01311980};
```

---

would create a structure variable `grace` to store the name, age, and data of birth of a person called Grace. Then in order to access the specific members within the person structure variable, use the variable name followed by the dot operator (`.`) and the member name. Therefore, `grace.name`, `grace.age`, `grace.DOB` would refer to Grace's name, age, and data of birth, respectively.

## POINTERS

When programming the 8051 in assembly, sometimes register such as R0, R1, and DPTR are used to store the addresses of some data in a certain memory location. When data is accessed via these registers, indirect addressing is used. In this case, we say that R0, R1, or DPTR are used to point to the data, so they are essentially pointers.

Correspondingly in C, indirect access of data can be done through specially defined pointer variables. Pointers are simply just special types of variables, but whereas normal variables are used to directly store data, pointer variables are used to store the addresses of the data. Just bear in mind that whether you use normal variables or pointer variables, you still get to access the data in the end. It is just whether you go directly to where it is stored and get the data, as in the case of normal variables, or first consult a directory to check the location of that data before going there to get it, as in the case of pointer variables.

Declaring a pointer follows the format:

```
data_type *pointer_name;
```

where

<code>data_type</code>	refers to which type of data that the pointer is pointing to
<code>*</code>	denotes that this is a pointer variable
<code>pointer_name</code>	is the name of the pointer

As an example, the following declarations:

```
int * numPtr
int num;
numPtr = &num;
```

first declares a pointer variable called `numPtr` that will be used to point to data of type `int`. The second declaration declares a normal variable and is put there for comparison. The third line assigns the address of the `num` variable to the `numPtr` pointer. The address of any variable can be obtained by using the address operator, `&`, as is used in this example. Bear in mind that once assigned, the `numPtr` pointer contains the address of the `num` variable, not the value of its data.

The above example could also be rewritten such that the pointer is straightaway initialized with an address when it is first declared:

```
int num;
int * numPtr = &num;
```

In order to further illustrate the difference between normal variables and pointer variables, consider the following, which is not a full C program but simply a fragment to illustrate our point:

```
int num = 7;
int * numPtr = &num;
printf ("%d\n", num);
printf ("%d\n", numPtr);
printf ("%d\n", &num);
printf ("%d\n", *numPtr);
```

---

The first line declares a normal variable, `num`, which is initialized to contain the data 7. Next, a pointer variable, `numPtr`, is declared, which is initialized to point to the address of `num`. The next four lines use the `printf()` function, which causes some data to be printed to some display terminal connected to the serial port. The first such line displays the contents of the `num` variable, which is in this case the value 7. The next displays the contents of the `numPtr` pointer, which is really some weird-looking number that is the address of the `num` variable. The third such line also displays the address of the `num` variable because the address operator is used to obtain `num`'s address. The last line displays the actual data to which the `numPtr` pointer is pointing, which is 7. The `*` symbol is called the indirection operator, and when used with a pointer, indirectly obtains the data whose address is pointed to by the pointer. Therefore, the output display on the terminal would show:

```
7
13452 (or some other weird-looking number)
13452 (or some other weird-looking number)
7
```

## A Pointer's Memory Type

Recall that pointers are also variables, so the question arises where they should be stored. When declaring pointers, we can specify different types of memory areas that these pointers should be in, for example:

```
int *xdata numPtr = &num;
```

This is the same as our previous pointer examples. We declare a pointer `numPtr`, which points to data of type `int` stored in the `num` variable. The difference here is the use of the memory type specifier `xdata` after the `*`. This specifies that pointer `numPtr` should reside in external data memory (**xdata**), and we say that the pointer's memory type is **xdata**.

## Typed Pointers

We can go even further when declaring pointers. Consider the example:

```
int data *xdata numPtr = &num;
```

The above statement declares the same pointer `numPtr` to reside in external data memory (**xdata**), and this pointer points to data of type `int` that is itself stored in the variable `num` in internal data memory (**data**). The memory type specifier, **data**, before the `*` specifies the *data memory type* while the memory type specifier, **xdata**, after the `*` specifies the pointer memory type.

Pointer declarations where the data memory types are explicitly specified are called typed pointers. Typed pointers have the property that you specify in your code where the data pointed to by pointers should reside. The size of typed pointers depends on the data memory type and could be one or two bytes.

## Untyped Pointers

When we do not explicitly state the data memory type when declaring pointers, we get untyped pointers, which are generic pointers that can point to data residing in any type of memory. Untyped pointers have the advantage that they can be used to point to any data independent of the type of memory in which the data is stored. All untyped pointers consist of 3 bytes, and are hence larger than typed pointers. Untyped pointers are also generally slower because the data memory type is not determined or known until the compiled program is run at runtime. The first byte of untyped pointers refers to the data memory type, which is simply a number according to the following table. The second and third bytes are, respectively, the higher-order and lower-order bytes of the address being pointed to.

An untyped pointer is declared just like normal C, where:

```
int *xdata numPtr = &num;
```

does not explicitly specify the memory type of the data pointed to by the pointer. In this case, we are using untyped pointers.

---

Data memory type values stored in first byte of untyped pointers	
Value	Data Memory Type
1	idata
2	xdata
3	pdata
4	data/bdata
5	code

## FUNCTIONS

In programming the 8051 in assembly, we learnt the advantages of using subroutines to group together common and frequently used instructions. The same concept appears in 8051 C, but instead of calling them subroutines, we call them **functions**. As in conventional C, a function must be declared and defined. A function definition includes a list of the number and types of inputs, and the type of the output (return type), plus a description of the internal contents, or what is to be done within that function.

The format of a typical function definition is as follows:

```
return_type  function_name (arguments)  [memory] [reentrant] [interrupt] [using]
{
    ...
}
```

where

return_type	refers to the data type of the return (output) value
function_name	is any name that you wish to call the function as
arguments	is the list of the type and number of input (argument) values
memory	refers to an explicit memory model (small, compact or large)
reentrant	refers to whether the function is reentrant (recursive)
interrupt	indicates that the function is actually an ISR
using	explicitly specifies which register bank to use

Consider a typical example, a function to calculate the sum of two numbers:

```
int sum (int a, int b)
{
    return a + b;
}
```

This function is called `sum` and takes in two arguments, both of type `int`. The return type is also `int`, meaning that the output (return value) would be an `int`. Within the body of the function, delimited by braces, we see that the return value is basically the sum of the two arguments. In our example above, we omitted explicitly specifying the options: `memory`, `reentrant`, `interrupt`, and `using`. This means that the arguments passed to the function would be using the default small memory model, meaning that they would be stored in internal data memory. This function is also by default non-recursive and a normal function, not an ISR. Meanwhile, the default register bank is bank 0.

## Parameter Passing

In 8051 C, parameters are passed to and from functions and used as function arguments (inputs). Nevertheless, the technical details of where and how these parameters are stored are transparent to the programmer, who does not need to worry about these technicalities. In 8051 C, parameters are passed through the register or through memory. Passing parameters through registers is faster and is the default way in which things are done. The registers used and their purpose are described in more detail below.

---

Registers used in parameter passing				
Number of Argument	Char / 1-Byte Pointer	INT / 2-Byte Pointer	Long/Float	Generic Pointer
1	R7	R6 & R7	R4–R7	R1–R3
2	R5	R4 & R5	R4–R7	
3	R3	R2 & R3		

Since there are only eight registers in the 8051, there may be situations where we do not have enough registers for parameter passing. When this happens, the remaining parameters can be passed through fixed memory locations. To specify that all parameters will be passed via memory, the NOREGPARMs control directive is used. To specify the reverse, use the REGPARMs control directive.

### Return Values

Unlike parameters, which can be passed by using either registers or memory locations, output values must be returned from functions via registers. The following table shows the registers used in returning different types of values from functions.

Registers used in returning values from functions		
Return Type	Register	Description
bit	Carry Flag (C)	
char/unsigned char/1-byte pointer	R7	
int/unsigned int/2-byte pointer	R6 & R7	MSB in R6, LSB in R7
long/unsigned long	R4–R7	MSB in R4, LSB in R7
float	R4–R7	32-bit IEEE format
generic pointer	R1–R3	Memory type in R3, MSB in R2, LSB in R1



---

## Appendix C: Indirect addressing inner 256B RAM

```
;/*-----*/
;/* --- STC 1T Series MCU the inner 256B normal RAM (indirect addressing) Demo -----*/
;/* If you want to use the program or the program referenced in the -----*/
;/* article, please specify in which data and procedures from STC -----*/
;/*-----*/
```

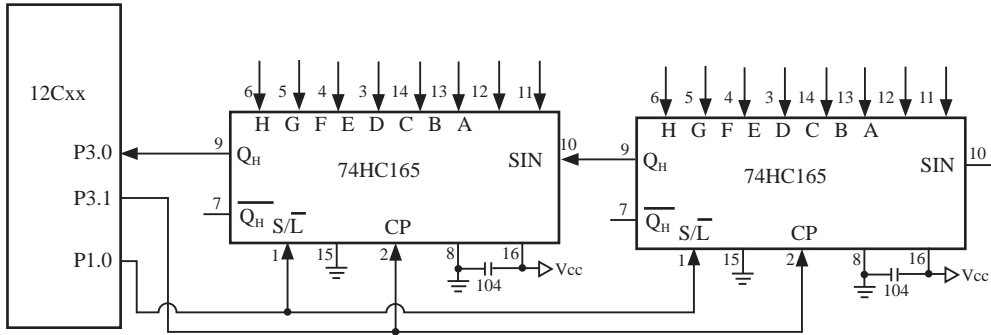
```
TEST_CONST EQU 5AH
;TEST_RAM EQU 03H
    ORG 0000H
    LJMP INITIAL
    ORG 0050H
INITIAL:
    MOV R0, #253
    MOV R1, #3H
TEST_ALL_RAM:
    MOV R2, #0FFH
TEST_ONE_RAM:
    MOV A, R2
    MOV @R1, A
    CLR A
    MOV A, @R1
    CJNE A, 2H, ERROR_DISPLAY
    DJNZ R2, TEST_ONE_RAM
    INC R1
    DJNZ R0, TEST_ALL_RAM
OK_DISPLAY:
    MOV P1, #11111110B
Wait1:
    SJMP Wait1
ERROR_DISPLAY:
    MOV A, R1
    MOV P1, A
Wait2:
    SJMP Wait2
    END
```

## Appendix D: Using Serial port to Expand I/O Ports

STC15 series MCU serial port mode0 can be used for expand IO if UART is free in your application. UART Mode0 is a synchronous shift register, the baudrate is fixed at  $f_{osc}/12$ , RXD pin (P3.0) is the data I/O port, and TXD pin (P3.1) is clock output port, data width is 8 bits, always sent / received the lowest bit at first.

### (1) Using 74HC165 expand parallel input ports

Please refer to the following circuit which using 2 pcs 74HC165 to expand 16 input I/Os



74HC165 is a 8-bit parallel input shift register, when S/L (Shift/Load) pin is falling to low level, the parallel port data is read into internal register, and now, if S/L is raising to high and ClockDisable pin (15 pin) is low level, then clock signal from CP pin is enable. At this time register data will be output from the Dh pin (9 pin) with the clock input.

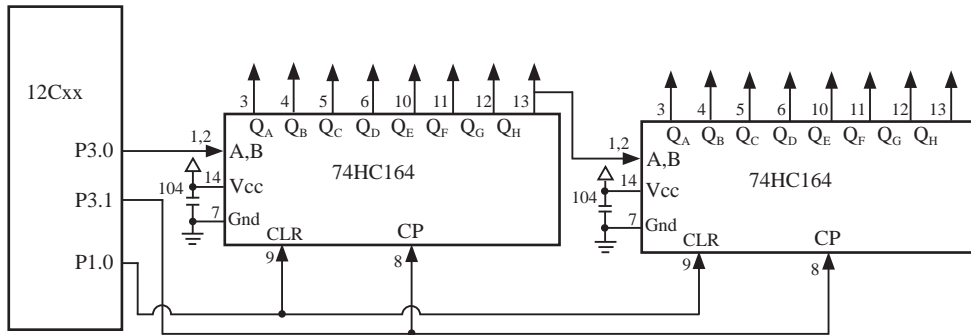
```

...
MOV R7,#05H           ;read 5 groups data
MOV R0,#20H          ;set buffer address
START: CLR P1.0       ;S/L = 0, load port data
      SETB P1.0       ;S/L = 1, lock data and enable clock
      MOV R1,#02H     ;2 bytes per group
RXDAT:MOV SCON,#00010000B ;set serial as mode 0 and enable receive data
WAIT:  JNB RI,WAIT    ;wait for receive complete
      CLR RI          ;clear receive complete flag
      MOV A,SBUF      ;read data to ACC
      MOV @R0,A       ;save data to buffer
      INC R0          ;modify buffer ptr
      DJNZ R1,RXDAT   ;read next byte
      DJNZ R7,START   ;read next group
...

```

(2) Using 74HC164 expand parallel output ports

Please refer to the following circuit which using 2 pcs 74HC164 to expand 16 output I/Os



When serial port is working in MODE0, the serial data is input/output from RXD(P3.0) pin and serial clock is output from TXD(P3.1). Serial data is always starting transmission from the lowest bit.

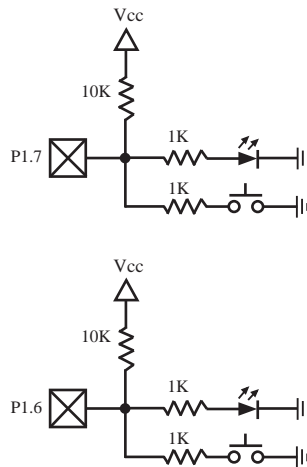
```

...
START: MOV R7,#02H           ;output 2 bytes data
        MOV R0,#30H         ;set buffer address
        MOV SCON,#00000000B ;set serial as mode 0
SEND:  MOV A,@R0            ;read data from buffer
        MOV SBUF,A          ;start send data
WAIT:  JNB TI,WAIT          ;wait for send complete
        CLR TI              ;clear send complete flag
        INC R0              ;modify buffer ptr
        DJNZ R7,SEND        ;send next data
...

```

---

## Appendix E: LED Driven by an I/O port and Key Scan



It can save a lot of I/O ports that STC15W4K32S4 MCU I/O ports can be used as the LED drivers and key detection concurrently because of their feature which they can be set to the weak pull, the strong pull (push-pull) output, only input (high impedance), open drain four modes.

When driving the LED, the I/O port should be set as strongly push-pull output, and the LED will be lighted when the output is high.

When testing the keys, the I/O port should be set as weak pull input, and then reading the status of external ports can test the keys.

---

## Appendix F: Notes of STC15 replacing Standard 8051

STC15 series MCU Timer0/Timer1/UART is fully compatible with the traditional 8051 MCU. After power on reset, the default input clock source is the divider 12 of system clock frequency, and UART baudrate generator defaults to Timer 1 and also can choose Timer 2 as its baud-rate generator. MCU instruction execution speed is faster than the traditional 8051 MCU 8 ~ 12 times in the same working environment, so software delay programs need to be adjusted.

### ALE

ALE pin is an disturbance source when traditional 8051's system clock frequency is too high. STC89xx series MCU add ALEOFF bit in AUXR register. While STC15 series MCU directly disable ALE pin dividing 6 the system clock output, and can remove ALE disturbance thoroughly. Please compare the following two registers.

AUXR register of STC89xx series

Mnemonic	Add	Name	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Reset Value
AUXR	8EH	Auxiliary register 0	-	-	-	-	-	-	EXTRAM	ALEOFF	xxxx,xx00

AUXR register of STC15 series

Mnemonic	Add	Name	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Reset Value
AUXR	8EH	Auxiliary register	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0000

### PSEN

Traditional 8051 execute external program through the PSEN signal. STC15 series is system MCU concept, integrated high-capacity internal program memory, do not need external program memory expansion generally, so have no PSEN signal.

### General Qusi-Bidirectional I/O

Traditional 8051 access I/O (signal transition or read status) timing is 12 clocks, STC15 series MCU is 4 clocks. When you need to read an external signal, if internal output a rising edge signal, for the traditional 8051, this process is 12 clocks, you can read at once, but for STC15W4K32S4 series MCU, this process is 4 clocks, when internal instructions is complete but external signal is not ready, so you must delay 1~2 nop operation.

### Port drive capability

STC15 series I/O port sink drive current is 20mA, has a strong drive capability, the port is not burn out when drive high current generally. STC89 series I/O port sink drive current is only 6mA, is not enough to drive high current. For the high current drive applications, it is strongly recommended to use STC15 series MCU.

### WatchDog

STC15 series MCU's watch dog timer control register (WDT\_CONTR) is location at C1H, add watch dog reset flag.

STC15W4K32S4 series WDT\_CONTR ( C1H )

Mnemonic	Add	Name	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Reset Value
WDT_CONTR	C1h	Wact-Dog-Timer Control register	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	PS2	PS1	PS0	xx00,0000

STC89 series WDT\_CONTR ( E1H )

Mnemonic	Add	Name	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Reset Value
WDT_CONTR	E1h	Wact-Dog-Timer Control register	-	-	EN_WDT	CLR_WDT	IDL_WDT	PS2	PS1	PS0	xx00,0000

STC15 series MCU auto enable watch dog timer after ISP upgrade, but not in STC89 series, so STC15 series's watch dog is more reliable.

## EEPROM

SFR associated with EEPROM

Mnemonic	STC12Cxx	STC89xx	Description
	Address		
IAP_DATA	C2H	E2H	ISP/IAP Flash data register
IAP_ADDRH	C3H	E3G	ISP/IAP Flash HIGH address register
IAP_ADDRL	C4H	E4H	ISP/IAP Flash LOW address register
IAP_CMD	C5H	E5H	ISP/IAP Flash command register
IAP_TRIG	C6H	E6H	ISP/IAP command trigger register
IAP_CONTR	C7H	E7H	ISP/IAP control register

STC15 series write 5AH and A5H sequential to trigger EEPROM flash command, and STC89 series write 46H and B9H sequential to trigger EEPROM flash command.

STC15 series EEPROM start address all location at 0000H, but STC89 series is not.

### Internal/external clock source

STC15 series MCU has a optional internal high reliable R/C oscilator, Generally, for 40/44 pin package MCU, set to use external crystal oscillator and for 20/18/16 pin package set to use internl RC oscilator in factory. When use ISP download program, user can arbitrarily choose internal RC oscilator or external crystal oscilator. STC89 series MCU can only choose external crystal oscillator.

### Power consumption

Power consumption consists of two parts: crystal oscillator amplifier circuits and digital circuits. For crystal oscillator amplifier circuits, STC15 series is lower then STC89 series. For digital circuits, the higher clock frequency, the greater the power consumption. STC15 series MCU instruction execution speed is faster than theSTC89 series MCU 3~24 times in the same working environment, so if you need to achieve the same efficiency, STC15 series required frequency is lower than STC89 series MCU.

### About reset circuit

For STC15 series MCU, if the system frequency is below 12MHz, the external reset circuit is not required. Reset pin can be connected to ground through the 1K resistor or can be connected directly to ground. The proposal to create PCB to retain RC reset circuit

### About Clock oscillator

For STC15 series MCU, if you need to use internal RC oscillator, XTAL1 pin and XTAL2 pin must be floating. If you use a external active crystal oscillator, clock signal input from XTAL1 pin and XTAL2 pin floating.

### About power

For STC15 series MCU, power at both ends need to add a 47uF electrolytic capacitor and a 0.1uF capacitor, to remove the coupling and filtering.

---

## Appendix G: Instruction Speed Boost Summary

The STC MCU instructions are fully compatible with the traditional 8051's, which are divided among five functional groups:

- Arithmetic
- Logical
- Data transfer
- Boolean variable
- Program branching

Instruction execution speed boost summary :

There are 111 instructions in MCU. For new STC15 series MCU

24 times faster execution speed than the traditional 8051	2
12 times faster execution speed than the traditional 8051	28
8 times faster execution speed than the traditional 8051	19
6 times faster execution speed than the traditional 8051	40
4.8 times faster execution speed than the traditional 8051	8
4 times faster execution speed than the traditional 8051	14

Based on the analysis of frequency of use order statistics, STC15 series MCU instruction execution speed is faster than the traditional 8051 MCU 8 ~ 12 times in the same working environment.

Instruction execution clock count (for new STC15 series)

1 clock instruction	22
2 clock instruction	37
3 clock instruction	31
4 clock instruction	12
5 clock instruction	8
6 clock instruction	1

It needs 283 clocks to finish executing at one time all 111 instructions for STC15 series, while it needs 1944 clocks for the traditional 8051 MCU. Obviously, the speed of executing instruction for STC15 series MCU has been greatly enhanced. The average speed of STC15 series is 8~12 times faster than traditional 8051 MCU

The following tables provide a quick reference chart showing all the 8051 and STC15 series MCU instructions. Once you are familiar with the instruction set, this chart should prove a handy and quick source of reference.

STC15 series MCU with super high-speed CPU core of STC-Y5 works 20% faster than STC early 1T series (such as STC12/STC11/STC10 series) at same clock frequency.

### ARITHMETIC OPERATIONS

Mnemonic	Description	Byte	Execution clocks of traditional 8051	Execution clocks of STC15 series (super high-speed 1T 8051 CPU core of STC-Y5)	Efficiency Improved
ADD A Rn	Add register to Accumulator	1	12	1	12x
ADD A direct	Add direct byte to Accumulator	2	12	2	6x
ADD A @Ri	Add indirect RAM to Accumulator	1	12	2	6x
ADD A #data	Add immediate data to Accumulator	2	12	2	6x
ADDC A Rn	Add register to Accumulator with Carry	1	12	1	12x
ADDC A direct	Add direct byte to Accumulator with Carry	2	12	2	6x
ADDC A @Ri	Add indirect RAM to Accumulator with Carry	1	12	2	6x
ADDC A #data	Add immediate data to Acc with Carry	2	12	2	6x
SUBB A Rn	Subtract Register from Acc with borrow	1	12	1	6x
SUBB A direct	Subtract direct byte from Acc with borrow	2	12	2	6x
SUBB A @Ri	Subtract indirect RAM from ACC with borrow	1	12	2	6x
SUBB A #data	Subtract immediate data from ACC with borrow	2	12	2	6x
INC A	Increment Accumulator	1	12	1	12x
INC Rn	Increment register	1	12	2	6x
INC direct	Increment direct byte	2	12	3	4x
INC @Ri	Increment direct RAM	1	12	3	4x
DEC A	Decrement Accumulator	1	12	1	12x
DEC Rn	Decrement Register	1	12	2	6x
DEC direct	Decrement direct byte	2	12	3	4x
DEC @Ri	Decrement indirect RAM	1	12	3	4x
INC DPTR	Increment Data Pointer	1	24	1	24x
MUL AB	Multiply A & B	1	48	2	24x
DIV AB	Divide A by B	1	48	6	8x
DA A	Decimal Adjust Accumulator	1	12	3	4x



## LOGICAL OPERATIONS

Mnemonic	Description	Byte	Execution clocks of traditional 8051	Execution clocks of STC15 series (super high-speed 1T 8051 CPU core of STC-Y5)	Efficiency Improved
ANL A Rn	AND Register to Accumulator	1	12	1	12x
ANL A direct	AND direct byte to Accumulator	2	12	2	6x
ANL A @Ri	AND indirect RAM to Accumulator	1	12	2	6x
ANL A #data	AND immediate data to Accumulator	2	12	2	6x
ANL direct A	AND Accumulator to direct byte	2	12	3	4x
ANL direct #data	AND immediate data to direct byte	3	24	3	8x
ORL A Rn	OR register to Accumulator	1	12	1	12x
ORL A direct	OR direct byte to Accumulator	2	12	2	6x
ORL A, @Ri	OR indirect RAM to Accumulator	1	12	2	6x
ORL A # data	OR immediate data to Accumulator	2	12	2	6x
ORL direct A	OR Accumulator to direct byte	2	12	3	4x
ORL direct #data	OR immediate data to direct byte	3	24	3	8x
XRL A Rn	Exclusive-OR register to Accumulator	1	12	1	12x
XRL A direct	Exclusive-OR direct byte to Accumulator	2	12	2	6x
XRL A @Ri	Exclusive-OR indirect RAM to Accumulator	1	12	2	6x
XRL A # data	Exclusive-OR immediate data to Accumulator	2	12	2	6x
XRL direct A	Exclusive-OR Accumulator to direct byte	2	12	3	4x
XRL direct #data	Exclusive-OR immediate data to direct byte	3	24	3	8x
CLR A	Clear Accumulator	1	12	1	12x
CPL A	Complement Accumulator	1	12	1	12x
RL A	Rotate Accumulator Left	1	12	1	12x
RLC A	Rotate Accumulator Left through the Carry	1	12	1	12x
RR A	Rotate Accumulator Right	1	12	1	12x
RRC A	Rotate Accumulator Right through the Carry	1	12	1	12x
SWAP A	Swap nibbles within the Accumulator	1	12	1	12x

## DATA TRANSFER

Mnemonic	Description	Byte	Execution clocks of traditional 8051	Execution clocks of STC15 series (super high-speed 1T 8051 CPU core of STC-Y5)	Efficiency Improved
MOV A, Rn	Move register to Accumulator	1	12	1	12x
MOV A, direct	Move direct byte to Accumulator	2	12	2	6x
MOV A, @Ri	Move indirect RAM to Accumulator	1	12	2	6x
MOV A, #data	Move immediate data to Accumulator	2	12	2	6x
MOV Rn, A	Move Accumulator to register	1	12	1	12x
MOV Rn, direct	Move direct byte to register	2	24	3	8x
MOV Rn, #data	Move immediate data to register	2	12	2	6x
MOV direct, A	Move Accumulator to direct byte	2	12	2	6x
MOV direct, Rn	Move register to direct byte	2	24	2	12x
MOV direct, direct	Move direct byte to direct	3	24	3	8x
MOV direct, @Ri	Move indirect RAM to direct byte	2	24	3	8x
MOV direct, #data	Move immediate data to direct byte	3	24	3	8x
MOV @Ri, A	Move Accumulator to indirect RAM	1	12	2	6x
MOV @Ri, direct	Move direct byte to indirect RAM	2	24	3	8x
MOV @Ri, #data	Move immediate data to indirect RAM	2	12	2	6x
MOV DPTR, #data16	Move immediate data to indirect RAM	3	24	3	8x
MOVC A, @A+DPTR	Move Code byte relative to DPTR to Acc	1	24	5	4.8x
MOVC A, @A+PC	Move Code byte relative to PC to Acc	1	24	4	6x
MOVX A, @Ri	Move on-chip expanded RAM(8-bit addr) to Acc. Read operation	1	24	3	8x
MOVX @Ri, A	Move Acc to on-chip expanded RAM(8-bit addr).Write operation.	1	24	4	8x
MOVX A, @DPTR	Move on-chip expanded RAM(16-bit addr) to Acc.Read operation.	1	24	2	12x
MOVX @DPTR, A	Move Acc to on-chip expanded RAM (16-bit addr). Write operation.	1	24	3	8x
MOVX A, @Ri	Move Acc to External RAM(8-bit addr). Read operation.	1	24	5xN+2 see the following illustration about the value of N	*Note1
MOVX @Ri, A	Move Acc to External RAM(8-bit addr). Write operation.	1	24	5xN+3	*Note1
MOVX A, @DPTR	Move External RAM(16-bit addr) to Acc. Read operation.	1	24	5xN+1	*Note1
MOVX @DPTR, A	Move Acc to External RAM (16-bit addr). Write operation.	1	24	5xN+2	*Note1
PUSH direct	Push direct byte onto stack	2	24	3	8x
POP direct	POP direct byte from stack	2	24	2	12x
XCH A, Rn	Exchange register with Accumulator	1	12	2	6x
XCH A, direct	Exchange direct byte with Accumulator	2	12	3	4x
XCH A, @Ri	Exchange indirect RAM with Accumulator	1	12	3	4x
XCHD A, @Ri	Exchange low-order Digit indirect RAM with Acc	1	12	3	4x

When EXRTS[1:0] = [0,0], N=1 in above formula;

When EXRTS[1:0] = [0,1], N=2 in above formula;

When EXRTS[1:0] = [1,0], N=4 in above formula;

When EXRTS[1:0] = [1,1], N=8 in above formula;

EXRTS[1 0] are the bit of B0 and B1 BUS\_SPEED

**BOOLEAN VARIABLE MANIPULATION**

Mnemonic		Description	Byte	Execution clocks of traditional 8051	Execution clocks of STC15 series (super high-speed 1T 8051 CPU core of STC-Y5)	Efficiency Improved
CLR	C	Clear Carry	1	12	1	12x
CLR	bit	Clear direct bit	2	12	3	4x
SETB	C	Set Carry	1	12	1	12x
SETB	bit	Set direct bit	2	12	3	4x
CPL	C	Complement Carry	1	12	1	12x
CPL	bit	Complement direct bit	2	12	3	4x
ANL	C, bit	AND direct bit to Carry	2	24	2	12x
ANL	C, /bit	AND complement of direct bit to Carry	2	24	2	12x
ORL	C, bit	OR direct bit to Carry	2	24	2	12x
ORL	C, /bit	OR complement of direct bit to Carry	2	24	2	12x
MOV	C, bit	Move direct bit to Carry	2	12	2	12x
MOV	bit, C	Move Carry to direct bit	2	24	3	8x
JC	rel	Jump if Carry is set	2	24	3	8x
JNC	rel	Jump if Carry not set	2	24	3	8x
JB	bit, rel	Jump if direct bit is set	3	24	5	4.8x
JNB	bit, rel	Jump if direct bit is not set	3	24	5	4.8x
JBC	bit, rel	Jump if direct bit is set & clear bit	3	24	5	4.8x

## PROGRAM BRANCHING

Mnemonic	Description	Byte	Execution clocks of traditional 8051	Execution clocks of STC15 series (super high-speed 1T 8051 CPU core of STC-Y5)	Efficiency Improved
ACALL   addr11	Absolute Subroutine Call	2	24	4	6x
LCALL   addr16	Long Subroutine Call	3	24	4	6x
RET	Return from Subroutine	1	24	4	6x
RETI	Return from interrupt	1	24	4	6x
AJMP    addr11	Absolute Jump	2	24	3	8x
LJMP    addr16	Long Jump	3	24	4	6x
SJMP    re1	Short Jump (relative addr)	2	24	3	8x
JMP     @A+DPTR	Jump indirect relative to the DPTR	1	24	5	4.8x
JZ     re1	Jump if Accumulator is Zero	2	24	4	6x
JNZ    re1	Jump if Accumulator is not Zero	2	24	4	6x
CJNE   A   direct   re1	Compare direct byte to Acc and jump if not equal	3	24	5	4.8x
CJNE   A   #data   re1	Compare immediate data to Acc and Jump if not equal	3	24	4	6x
CJNE   Rn   #data   re1	Compare immediate data to register and Jump if not equal	3	24	4	6x
CJNE   @Ri   #data   re1	Compare immediate data to indirect and jump if not equal	3	24	5	4.8x
DJNZ   Rn   re1	Decrement register and jump if not Zero	2	24	4	6x
DJNZ   direct   re1	Decrement direct byte and Jump if not Zero	3	24	5	4.8x
NOP	No Operation	1	12	1	12x

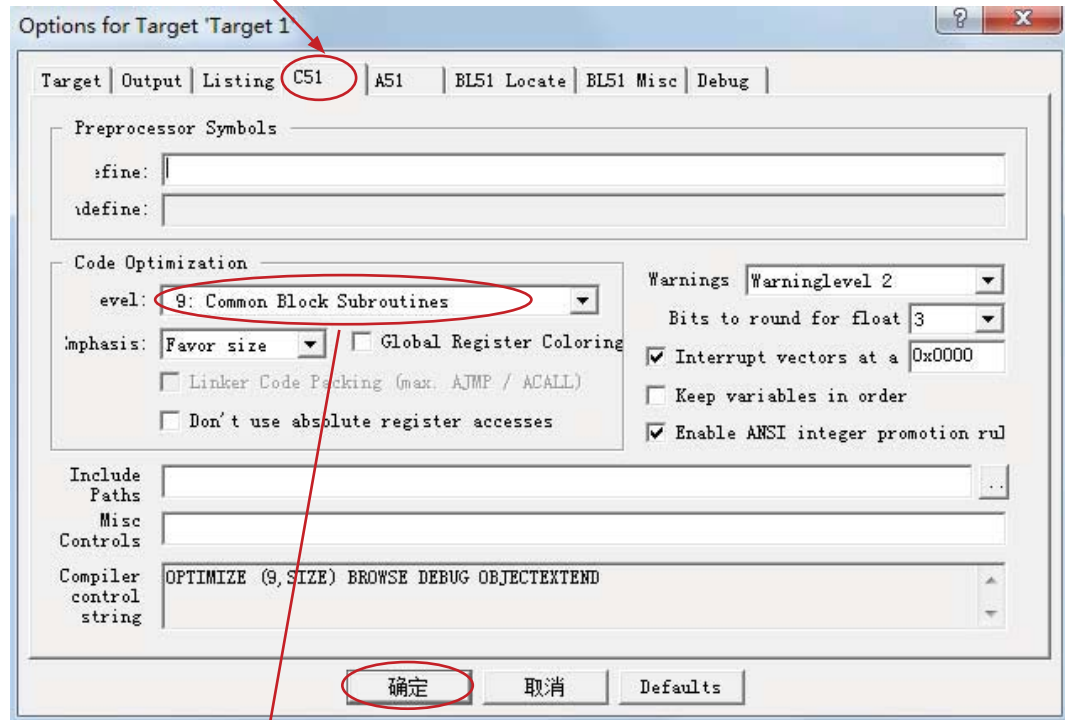
Update Date 2011-10-17

---

## Appendix H: How to reduce the Code Length by Keil C

Setting as shown below in Keil C can maximum reduce about 10K to the length of original code

1. Choose the "Options for Target" in "Project" menu
2. Choose the option "C51" in "Options for Target"



3. Code Optimization, 9 common block subroutines
4. Click "OK", compile the program once again.