STC micro™
宏 晶 科 技

STC8G 系列单片机
技术参考手册

资料更新日期：**2019/10/22**

目录

1 概述 1
2 特性及价格 2
2.1 STC8G1K08 系列特性及价格 2
3 管脚及说明 4
3.1 管脚图 4
3.1.1 STC8G1K08 系列管脚图 4
3.2 管脚说明 6
3.2.1 STC8G1K08 系列管脚说明 6
3.3 功能脚切换 9
3.3.1 功能脚切换相关寄存器 9
3.4 范例程序 10
3.4.1 串口 1 切换 10
3.4.2 串口 2 切换 11
3.4.3 SPI切换 12
3.4.4 PCA/CCP/PWM切换 13
3.4.5 I2C切换 15
3.4.6 比较器输出切换 16
3.4.7 主时钟输出切换 17
4 封装尺寸图 19
4.1 TSSOP20 封装尺寸图 19
4.2 QFN20 封装尺寸图（3mm*3mm） 20
4.3 SOP16 封装尺寸图 21
4.4 SOP8 封装尺寸图 22
4.5 STC8 系列单片机命名规则 23
5 ISP下载及典型应用线路图 24
5.1 STC8G系列ISP下载应用线路图 24
5.1.1 使用RS-232 转换器下载 24
5.1.2 使用PL2303-SA下载 25
5.1.3 使用PL2303-GL下载 26
5.1.4 使用U8-Mini工具下载 27
5.1.5 使用U8W工具下载 28
5.1.6 USB直接ISP下载 29
6 时钟、复位与电源管理 30
6.1 系统时钟控制 30
6.2 STC8G系列内部IRC频率调整 33
6.3 系统复位 35
6.4 系统电源管理 37
6.5 范例程序 38
6.5.1 选择系统时钟源 38
6.5.2 主时钟分频输出 40

6.5.3 看门狗定时器应用 41
6.5.4 软复位实现自定义下载 42
6.5.5 低压检测 44
6.5.6 省电模式 45
6.5.7 使用INT0/INT1/INT2/INT3/INT4 中断唤醒MCU 47
6.5.8 使用T0/T1/T2/T3/T4 中断唤醒MCU 50
6.5.9 使用RxD/RxD2 中断唤醒MCU 54
6.5.10 使用LVD中断唤醒MCU 56
6.5.11 使用CCP0/CCP1/CCP2 中断唤醒MCU 58
6.5.12 CMP中断唤醒MCU 60
6.5.13 使用LVD功能检测工作电压（电池电压） 62
7 存储器 67
7.1 程序存储器 67
7.2 数据存储器 68
7.2.1 内部RAM 68
7.2.2 内部扩展RAM 69
7.3 存储器中的特殊参数 70
7.3.1 读取Bandgap电压值 (从ROM中读取) 71
7.3.2 读取Bandgap电压值 (从RAM中读取) 74
7.3.3 读取全球唯一ID号 (从ROM中读取) 76
7.3.4 读取全球唯一ID号 (从RAM中读取) 79
7.3.5 读取 32K掉电唤醒定时器的频率 (从ROM中读取) 82
7.3.6 读取 32K掉电唤醒定时器的频率 (从RAM中读取) 84
7.3.7 用户自定义内部IRC频率 (从ROM中读取) 87
7.3.8 用户自定义内部IRC频率 (从RAM中读取) 89
8 特殊功能寄存器 91
8.1 STC8G1K08 系列 91
8.2 特殊功能寄存器列表 92
9 I/O口 95
9.1 I/O口相关寄存器 95
9.2 配置I/O口 98
9.3 I/O的结构图 99
9.3.1 准双向口（弱上拉） 99
9.3.2 推挽输出 99
9.3.3 高阻输入 99
9.3.4 开漏输出 100
9.4 范例程序 101
9.4.1 端口模式设置 101
9.4.2 双向口读写操作 102
9.4.3 用STC系列MCU的I/O口直接驱动段码LCD 103
10 指令系统 123
11 中断系统 127
11.1 STC8G系列中断源 127
11.2 STC8 中断结构图 129

11.3 STC8 系列中断列表 130
11.4 中断相关寄存器 132
11.4.1 中断使能寄存器（中断允许位） 132
11.4.2 中断请求寄存器（中断标志位） 135
11.4.3 中断优先级寄存器 136
11.5 范例程序 139
11.5.1 INT0 中断（上升沿和下降沿） 139
11.5.2 INT0 中断（下降沿） 140
11.5.3 INT1 中断（上升沿和下降沿） 141
11.5.4 INT1 中断（下降沿） 143
11.5.5 INT2 中断（下降沿） 144
11.5.6 INT3 中断（下降沿） 146
11.5.7 INT4 中断（下降沿） 147
11.5.8 定时器 0 中断 148
11.5.9 定时器 1 中断 150
11.5.10 定时器 2 中断 151
11.5.11 UART1 中断 153
11.5.12 UART2 中断 155
11.5.13 ADC中断 157
11.5.14 LVD中断 159
11.5.15 PCA中断 160
11.5.16 SPI中断 163
11.5.17 CMP中断 164
11.5.18 I2C中断 166
12 定时器/计数器 169
12.1 定时器的相关寄存器 169
12.2 定时器 0/1 170
12.3 定时器 2 173
12.4 掉电唤醒定时器 174
12.5 范例程序 175
12.5.1 定时器 0（模式 0－16 位自动重载） 175
12.5.2 定时器 0（模式 1－16 位不自动重载） 176
12.5.3 定时器 0（模式 2－8 位自动重载） 178
12.5.4 定时器 0（模式 3－16 位自动重载不可屏蔽中断） 179
12.5.5 定时器 0（外部计数－扩展T0 为外部下降沿中断） 180
12.5.6 定时器 0（测量脉宽－INT0 高电平宽度） 182
12.5.7 定时器 0（时钟分频输出） 183
12.5.8 定时器 1（模式 0－16 位自动重载） 185
12.5.9 定时器 1（模式 1－16 位不自动重载） 186
12.5.10 定时器 1（模式 2－8 位自动重载） 187
12.5.11 定时器 1（外部计数－扩展T1 为外部下降沿中断） 189
12.5.12 定时器 1（测量脉宽－INT1 高电平宽度） 190
12.5.13 定时器 1（时钟分频输出） 192
12.5.14 定时器 1（模式 0）做串口 1 波特率发生器 193

12.5.15 定时器 1（模式 2）做串口 1 波特率发生器 197
12.5.16 定时器 2（16 位自动重载） 200
12.5.17 定时器 2（外部计数－扩展T2 为外部下降沿中断） 202
12.5.18 定时器 2（时钟分频输出） 203
12.5.19 定时器 2 做串口 1 波特率发生器 205
12.5.20 定时器 2 做串口 2 波特率发生器 208
13 串口通信 213
13.1 串口相关寄存器 213
13.2 串口 1 214
13.2.1 串口 1 模式 0 215
13.2.2 串口 1 模式 1 216
13.2.3 串口 1 模式 2 219
13.2.4 串口 1 模式 3 219
13.2.5 自动地址识别 220
13.3 串口 2 222
13.3.1 串口 2 模式 0 222
13.3.2 串口 2 模式 1 223
13.4 串口注意事项 225
13.5 范例程序 226
13.5.1 串口 1 使用定时器 2 做波特率发生器 226
13.5.2 串口 1 使用定时器 1（模式 0）做波特率发生器 229
13.5.3 串口 1 使用定时器 1（模式 2）做波特率发生器 232
13.5.4 串口 2 使用定时器 2 做波特率发生器 236
14 比较器，掉电检测，内部固定比较电压 240
14.1 比较器内部结构图 240
14.2 比较器相关的寄存器 241
14.3 范例程序 243
14.3.1 比较器的使用（中断方式） 243
14.3.2 比较器的使用（查询方式） 245
14.3.3 比较器作外部掉电检测 247
14.3.4 比较器检测工作电压（电池电压） 249
15 IAP/EEPROM 253
15.1 EEPROM相关的寄存器 253
15.2 EEPROM大小及地址 255
15.3 范例程序 257
15.3.1 EEPROM基本操作 257
15.3.2 使用MOVC读取EEPROM 260
15.3.3 使用串口送出EEPROM数据 263
16 ADC模数转换 268
16.1 ADC相关的寄存器 268
16.2 范例程序 271
16.2.1 ADC基本操作（查询方式） 271
16.2.2 ADC基本操作（中断方式） 272
16.2.3 格式化ADC转换结果 274

16.2.4 利用ADC第 16 通道测量外部电压或电池电压 276
17 PCA/CCP/PWM应用 279
17.1 PCA相关的寄存器 279
17.2 PCA工作模式 282
17.2.1 捕获模式 282
17.2.2 软件定时器模式 282
17.2.3 高速脉冲输出模式 283
17.2.4 PWM脉宽调制模式 283
17.3 范例程序 287
17.3.1 PCA输出PWM（6/7/8/10 位） 287
17.3.2 PCA捕获测量脉冲宽度 289
17.3.3 PCA实现 16 位软件定时 292
17.3.4 PCA输出高速脉冲 295
17.3.5 PCA扩展外部中断 298
18 同步串行外设接口SPI 301
18.1 SPI相关的寄存器 301
18.2 SPI通信方式 303
18.2.1 单主单从 303
18.2.2 互为主从 303
18.2.3 单主多从 304
18.3 配置SPI 305
18.4 数据模式 307
18.5 范例程序 308
18.5.1 SPI单主单从系统主机程序（中断方式） 308
18.5.2 SPI单主单从系统从机程序（中断方式） 310
18.5.3 SPI单主单从系统主机程序（查询方式） 311
18.5.4 SPI单主单从系统从机程序（查询方式） 313
18.5.5 SPI互为主从系统程序（中断方式） 315
18.5.6 SPI互为主从系统程序（查询方式） 317
19 I^2C总线 320
19.1 I^2C相关的寄存器 320
19.2 I^2C主机模式 321
19.3 I^2C从机模式 324
19.4 范例程序 327
19.4.1 I^2C主机模式访问AT24C256（中断方式） 327
19.4.2 I^2C主机模式访问AT24C256（查询方式） 332
19.4.3 I^2C主机模式访问PCF8563 338
19.4.4 I^2C从机模式（中断方式） 343
19.4.5 I^2C从机模式（查询方式） 347
19.4.6 测试I^2C从机模式代码的主机代码 351
20 增强型双数据指针 357
20.1 范例程序 359
20.1.1 示例代码 1 359
20.1.2 示例代码 2 359

附录A 无 361
附录B 无 366
附录C 使用第三方MCU对STC8G系列单片机进行ISP下载范例程序 429
附录D 电气特性 437
附录E 应用注意事项 439
附录F STC8G系列头文件 440
附录G 更新记录 444

1 概述

STC8G 系列单片机是不需要外部晶振和外部复位的单片机，是以超强抗干扰/超低价/高速/低功耗为目标的 8051 单片机，在相同的工作频率下，STC8G 系列单片机比传统的 8051 约快 12 倍（速度快 11.2~13.2 倍），依次按顺序执行完全部的 111 条指令，STC8G 系列单片机仅需 147 个时钟，而传统 8051 则需要 1944 个时钟。STC8G 系列单片机是 STC 生产的单时钟/机器周期(1T)的单片机，是宽电压/高速/高可靠/低功耗/强抗静电/较强抗干扰的新一代 8051 单片机，超级加密。指令代码完全兼容传统 8051。

MCU 内部集成高精度 R/C 时钟(±0.3%，常温下+25℃)，-1.35%～+1.3%温飘(-40℃~+85℃)，-0.76%～+0.98%温飘(-20℃~+65℃)。ISP 编程时 4MHz~35MHz 宽范围可设置（注意：温度范围为-40℃~+85℃时，最高频率须控制在 35MHz 以下），可彻底省掉外部昂贵的晶振和外部复位电路(内部已集成高可靠复位电路，ISP 编程时 4 级复位门槛电压可选)。

MCU 内部有 3 个可选时钟源：内部高精度 IRC 时钟（ISP 下载时可进行调节）、内部 32KHz 的低速 IRC、外部 4M~33M 晶振或外部时钟信号。用户代码中可自由选择时钟源，时钟源选定后可再经过 8-bit 的分频器分频后再将时钟信号提供给 CPU 和各个外设（如定时器、串口、SPI 等）。

MCU 提供两种低功耗模式：IDLE 模式和 STOP 模式。IDLE 模式下，MCU 停止给 CPU 提供时钟，CPU 无时钟，CPU 停止执行指令，但所有的外设仍处于工作状态，此时功耗约为 1.0mA（6MHz 工作频率）。STOP 模式即为主时钟停振模式，即传统的掉电模式/停电模式/停机模式，此时 CPU 和全部外设都停止工作，功耗可降低到 0.6uA@VCC=5.0V，0.4uA@VCC=3.3V。

MCU 提供了丰富的数字外设（2 个串口、3 个定时器、3 组 PCA 以及 I^2C、SPI）接口与模拟外设（超高速 ADC、比较器），可满足广大用户的设计需求。

STC8G 系列单片机内部集成了增强型的双数据指针。通过程序控制, 可实现数据指针自动递增或递减功能以及两组数据指针的自动切换功能。

产品线	UART	定时器	ADC	增强型 PWM	高级 PWM	RTC	PCA	比较器	SPI	I2C	备注
STC8G1K08 系列	●	●	●				●	●	●	●	

2 特性及价格

2.1 STC8G1K08 系列特性及价格

➢ 选型价格（不需要外部晶振、不需要外部复位，10 位 ADC，15 通道）

单片机型号	工作电压（V）	Flash 程序存储器 10 万次 字节	内部 DATA RAM	大容量扩展 SRAM 字节	强大的双 DPTR 可增可减	EEPROM 10 万次 字节	I/O 口最多数量	串口并可掉电唤醒	SPI	I²C	定时器/计数器（T0-T2 外部管脚也可掉电唤醒）	16 位高级 PWM 定时器 互补对称死区	15 位增强型 PWM（带死区控制）	PCA/CCP/PWM（可当外部中断并可掉电唤醒）	掉电唤醒专用定时器	15 路高速 ADC(8 路 PWM 可当 8 路 D/A 使用)	比较器（可当 1 路 A/D，可作外部掉电检测）	内部低压检测中断并可掉电唤醒	看门狗 复位定时器	内部高可靠复位（可选复位门槛电压）	内部高精准时钟（24MHz 可调）	可对外输出时钟及复位	程序加密后传输（防拦截）	可设置下次更新程序需口令	支持 RS485 下载	支持 USB 直接下载	本身就可在线仿真	封装				2019 年新品供货信息
																												TSSOP20	QFN20 (3mm*3mm)	SOP16	SOP8	
STC8G1K08	1.9-5.5	8K	256B	1K	2	4K	18	2	有	有	3	-	-	3	有	10 位	有	有	有	4 级	有	是	有	是	是	是	是	¥1.15	¥1.20	暂未生产		已供样
STC8G1K12	1.9-5.5	12K	256B	1K	2	IAP	18	2	有	有	3	-	-	3	有	10 位	有	有	有	4 级	有	是	有	是	是	是	是	¥1.30	¥1.35			

➢ 内核

- ✓ 超高速 8051 内核（1T），比传统 8051 约快 12 倍以上
- ✓ 指令代码完全兼容传统 8051
- ✓ 16 个中断源，4 级中断优先级
- ✓ 支持在线仿真

➢ 工作电压

- ✓ 1.9V～5.5V
- ✓ 内建 LDO

➢ 工作温度

- ✓ -40℃～85℃

➢ Flash 存储器

- ✓ 最大 12K 字节 FLASH 空间，用于存储用户代码
- ✓ 支持用户配置 EEPROM 大小，512 字节单页擦除，擦写次数可达 10 万次以上
- ✓ 支持在系统编程方式（ISP）更新用户应用程序，无需专用编程器
- ✓ 支持单芯片仿真，无需专用仿真器，理论断点个数无限制

➢ SRAM

- ✓ 128 字节内部直接访问 RAM（DATA）
- ✓ 128 字节内部间接访问 RAM（IDATA）
- ✓ 1024 字节内部扩展 RAM（内部 XDATA）

➢ 时钟控制

- ✓ 内部高精度 IRC（ISP 编程时可进行上下调整）
  - ⌖ 误差±0.3%（常温下 25℃）
  - ⌖ -1.35%～+1.30%温漂（全温度范围，-40℃～85℃）
  - ⌖ -0.76%～+0.98%温漂（温度范围，-20℃～65℃）

- ✓ 内部 32KHz 低速 IRC（误差较大）
- ✓ 外部晶振（4MHz～33MHz）和外部时钟

  用户可自由选择上面的 3 种时钟源

➢ **复位**

- ✓ 硬件复位
  - ⌖ 上电复位
  - ⌖ 复位脚复位，出厂时 P5.4 默认为 I/O 口，ISP 下载时可将 P5.4 管脚设置为复位脚（**注意：当设置 P5.4 管脚为复位脚时，复位电平为低电平**）
  - ⌖ 看门狗溢出复位
  - ⌖ 低压检测复位，提供 4 级低压检测电压：2.0V（实测为 1.90V～2.04V）、2.4V（实测为 2.30V～2.50V）、2.7V（实测为 2.61V～2.82V）、3.0V（实测为 2.90V～3.13V）。

    每级低压检测电压都是由一个上限电压和一个下限电压组成的电压范围，当工作电压从 5V/3.3V 向下掉到低压检测的下限门槛电压时，低压检测生效；当电压从 0V 上升到低压检测的上限门槛电压时，低压检测生效。
- ✓ 软件复位
  - ⌖ 软件方式写复位触发寄存器

➢ **中断**

- ✓ 提供 16 个中断源：INT0、INT1、INT2、INT3、INT4、定时器 0、定时器 1、定时器 2、串口 1、串口 2、ADC 模数转换、LVD 低压检测、SPI、I^2C、比较器、PCA/CCP/PWM
- ✓ 提供 4 级中断优先级

➢ **数字外设**

- ✓ 3 个 16 位定时器：定时器 0、定时器 1、定时器 2，其中定时器 0 的模式 3 具有 NMI（不可屏蔽中断）功能，定时器 0 和定时器 1 的模式 0 为 16 位自动重载模式
- ✓ 2 个高速串口：串口 1、串口 2，波特率时钟源最快可为 FOSC/4
- ✓ 3 组 16 位 PCA 模块：CCP0、CCP1、CCP2，可用于捕获、高速脉冲输出，及 6/7/8/10 位的 PWM 输出
- ✓ SPI：支持主机模式和从机模式以及主机/从机自动切换
- ✓ I^2C：支持主机模式和从机模式

➢ **模拟外设**

- ✓ 超高速 ADC，支持 **10 位精度** 15 通道（通道 0～通道 14）的模数转换
- ✓ ADC 的通道 15 用于测试内部参考电压（芯片在出厂时，内部参考电压调整为 **1.19V**，误差±1%）
- ✓ 比较器，一组比较器附近

➢ **GPIO**

- ✓ 最多可达 18 个 GPIO：P1.0~P1.7、P3.0~P3.7、P5.4~P5.5
- ✓ 所有的 GPIO 均支持如下 4 种模式：准双向口模式、强推挽输出模式、开漏输出模式、高阻输入模式
- ✓ **除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口时必须先设置 I/O 口模式**
- ✓ **另外每个 I/O 均可独立使能内部 4K 上拉电阻**

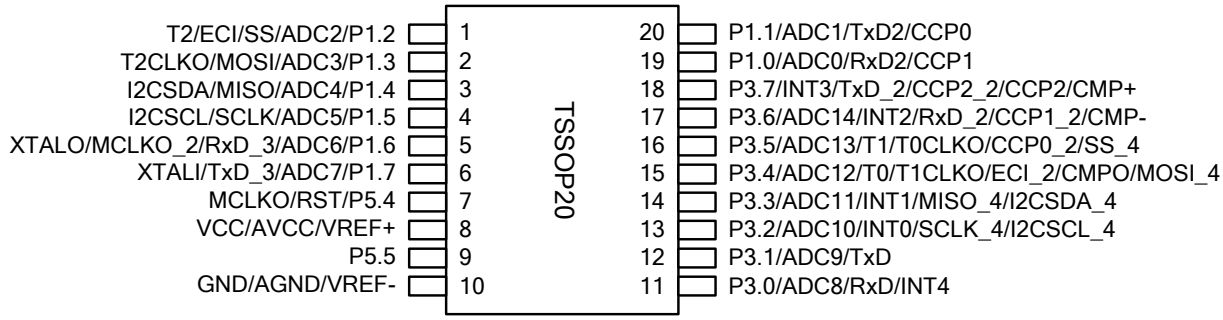
➢ **封装**

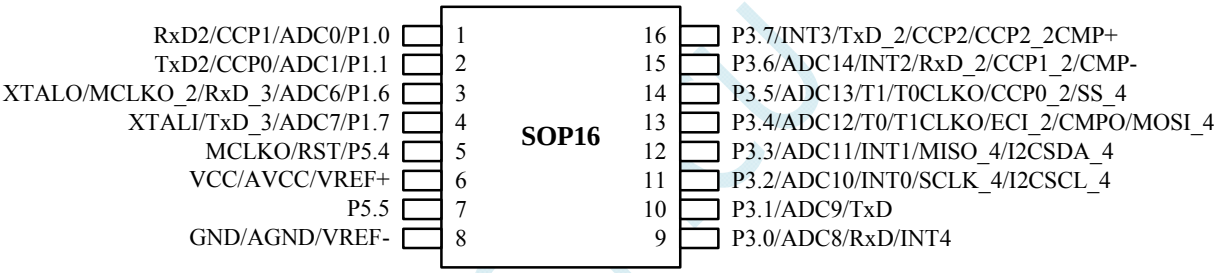
- ✓ TSOP20、QFN20（3mm*3mm）、SOP16（暂未生产）、SOP8（暂未生产）

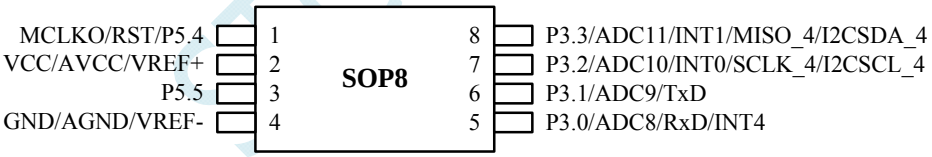
3 管脚及说明

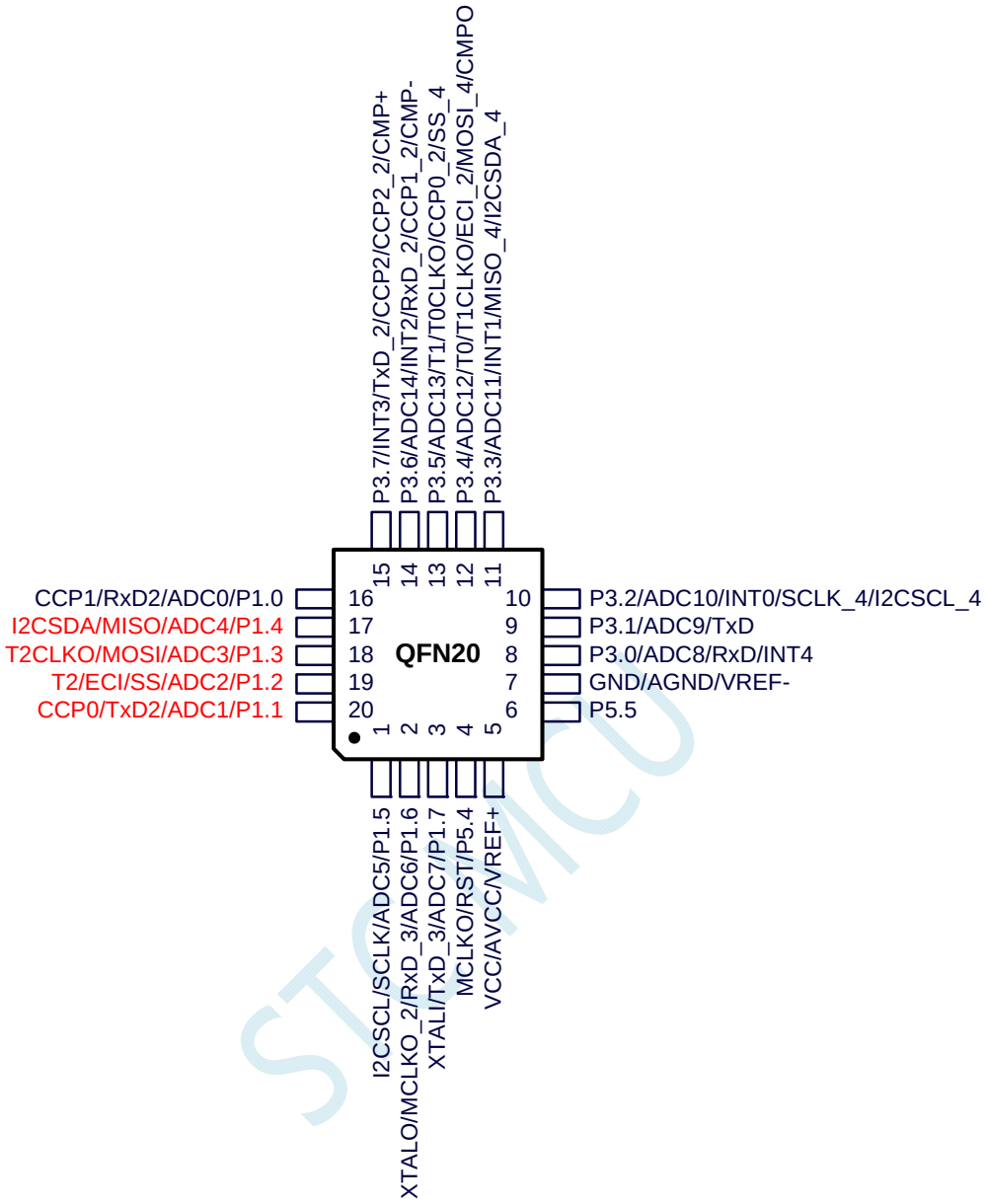
3.1 管脚图

3.1.1 STC8G1K08 系列管脚图









注意：
1、除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口时必须先设置 I/O 口模式
2、所有的 I/O 口均可以设置为准双向口模式、强推挽输出模式、开漏输出模式或者高阻输入模式，另外每个 I/O 均可独立使能内部 4K 上拉电阻
3、当使能 P5.4 口为复位脚时，复位电平为低电平
4、对于 STC8G1K08 系列 B 版芯片，P5.4 作 I/O 口使用时，电流不要超过 50mA，也不要有强的冲击
5、STC8G1K08 系列 B 版芯片所支持的 USB 下载为 I/O 口软件模拟的 USB，由于受制成、温度等多方面因素的影响，会导致有一定比例的芯片无法进行 USB 下载，经实际测试，无法 USB 下载的比例可能在 5%～8%

3.2 管脚说明

3.2.1 STC8G1K08 系列管脚说明

编号				名称	类型	说明
TSSOP20	QFN20	SOP16	SOP8			
1	19			P1.2	I/O	标准 I/O 口
				ADC2	I	ADC 模拟输入通道 2
				SS	I/O	SPI 从机选择
				T2	I	定时器 2 外部时钟输入
				ECI	I	PCA 的外部脉冲输入
2	18			P1.3	I/O	标准 I/O 口
				ADC3	I	ADC 模拟输入通道 3
				T2CLKO	O	定时器 2 时钟分频输出
				MOSI	I/O	SPI 主机输出从机输入
3	17			P1.4	I/O	标准 I/O 口
				ADC4	I	ADC 模拟输入通道 4
				MISO	I/O	SPI 主机输入从机输出
				SDA	I/O	I2C 接口的数据线
4	1			P1.5	I/O	标准 I/O 口
				ADC5	I	ADC 模拟输入通道 5
				SCLK	I/O	SPI 的时钟脚
				SCL	I/O	I2C 的时钟线
5	2	3		P1.6	I/O	标准 I/O 口
				ADC6	I	ADC 模拟输入通道 6
				RxD_3	I	串口 1 的接收脚
				MCLKO_2	O	主时钟分频输出
				XTALO	O	外部晶振脚
6	3	4		P1.7	I/O	标准 I/O 口
				ADC7	I	ADC 模拟输入通道 7
				TxD_3	O	串口 1 的发送脚
				XTALI	I	外部晶振脚
7	4	5	1	P5.4	I/O	标准 I/O 口
				RST	I	复位引脚
				MCLKO	O	主时钟分频输出
				SDA_2	I/O	I2C 的数据线
8	5	6	2	VCC	VCC	电源脚
				AVCC	VCC	ADC 电源
				VREF+	I	ADC 的参考电压脚
9	6	7	3	P5.5	I/O	标准 I/O 口
				SCL_2	I/O	I2C 的时钟线
10	7	8	4	GND	GND	地线
				AGND	GND	ADC 地线
				VREF-	I	ADC 的参考电压地线

编号				名称	类型	说明
TSSOP20	QFN20	SOP16	SOP8			
11	8	9	5	P3.0	I/O	标准 I/O 口
				RxD	I	串口 1 的接收脚
				ADC8	I	ADC 模拟输入通道 8
				INT4	I	外部中断 4
12	9	10	6	P3.1	I/O	标准 I/O 口
				TxD	O	串口 1 的发送脚
				ADC9	I	ADC 模拟输入通道 9
13	10	11	7	P3.2	I/O	标准 I/O 口
				INT0	I	外部中断 0
				ADC10	I	ADC 模拟输入通道 10
				SCLK_4	I/O	SPI 的时钟脚
14	11	12	8	P3.3	I/O	标准 I/O 口
				INT1	I	外部中断 1
				ADC11	I	ADC 模拟输入通道 11
				MISO_4	I/O	SPI 主机输入从机输出
15	12	13		P3.4	I/O	标准 I/O 口
				T0	I	定时器 0 外部时钟输入
				T1CLKO	O	定时器 1 时钟分频输出
				ADC12	I	ADC 模拟输入通道 12
				ECI_2	I	PCA 的外部脉冲输入
				CMPO	O	比较器输出
				MOSI_4	I/O	SPI 主机输出从机输入
16	13	14		P3.5	I/O	标准 I/O 口
				T1	I	定时器 1 外部时钟输入
				T0CLKO	O	定时器 0 时钟分频输出
				ADC13	I	ADC 模拟输入通道 13
				CCP0_2	I/O	PCA 的捕获输入和脉冲输出
				SS_4	I	SPI 的从机选择脚（主机为输出）
17	14	15		P3.6	I/O	标准 I/O 口
				INT2	I	外部中断 2
				RxD_2	I	串口 1 的接收脚
				ADC14	I	ADC 模拟输入通道 14
				CCP1_2	I/O	PCA 的捕获输入和脉冲输出
				CMP-	I	比较器负极输入

编号				名称	类型	说明
TSSOP20	QFN20	SOP16	SOP8			
18	15	16		P3.7	I/O	标准 I/O 口
				INT3	I	外部中断 3
				TxD_2	O	串口 1 的发送脚
				CCP2	I/O	PCA 的捕获输入和脉冲输出
				CCP2_2	I/O	PCA 的捕获输入和脉冲输出
				CMP+	I	比较器正极输入
19	16	1		P1.0	I/O	标准 I/O 口
				RxD2	O	串口 2 的接收脚
				ADC0	I	ADC 模拟输入通道 0
				CCP1	I/O	PCA 的捕获输入和脉冲输出
20	20	2		P1.1	I/O	标准 I/O 口
				TxD2	O	串口 2 的发送脚
				ADC1	I	ADC 模拟输入通道 1
				CCP0	I/O	PCA 的捕获输入和脉冲输出

3.3 功能脚切换

STC8G 系列单片机的特殊外设串口 1、串口 2、SPI、PCA、I²C 以及总线控制脚可以在多个 I/O 直接进行切换，以实现一个外设当作多个设备进行分时复用。

3.3.1 功能脚切换相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P_SW1	外设端口切换寄存器 1	A2H	S1_S[1:0]		CCP_S[1:0]		SPI_S[1:0]		0	-	nn00,000x
P_SW2	外设端口切换寄存器 2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	-	-	S2_S	0x00,0xx0

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
MCLKOCR	主时钟输出控制寄存器	FE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000

外设端口切换控制寄存器 1

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

S1_S[1:0]：串口 1 功能脚选择位

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

CCP_S[1:0]：PCA 功能脚选择位

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

SPI_S[1:0]：SPI 功能脚选择位

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

外设端口切换控制寄存器 2

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

I2C_S[1:0]：I²C 功能脚选择位

I2C_S[1:0]	SCL	SDA
00	P1.5	P1.4
01	P5.5	P5.4

CMPO_S：比较器输出脚选择位

CMPO_S	CMPO

0	P3.4
1	P4.1

S2_S：串口 2 功能脚选择位

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

时钟选择寄存器

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

MCLKO_S：主时钟输出脚选择位

MCLKO_S	MCLKO
0	P5.4
1	P1.6

3.4 范例程序

3.4.1 串口 1 切换

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW1       DATA    0A2H

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

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

            SJMP    $
```

```
        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"

sfr     P_SW1   =   0xa2;

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

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

    while (1);
}
```

3.4.2 串口 2 切换

汇编代码

```
;测试工作频率为11.0592MHz

P_SW2       DATA    0BAH

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
```

```
        MOV     P5M0, #00H
        MOV     P5M1, #00H

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

        SJMP    $

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"

sfr     P_SW2   =   0xba;

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

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

    while (1);
}
```

3.4.3 SPI切换

汇编代码

```
;测试工作频率为11.0592MHz

P_SW1   DATA    0A2H

P1M1    DATA    091H
P1M0    DATA    092H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
```

```
MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

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

        SJMP    $

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"

sfr     P_SW1   =   0xa2;

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

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

    while (1);
}
```

3.4.4 PCA/CCP/PWM切换

汇编代码

```
;测试工作频率为11.0592MHz

P_SW1       DATA    0A2H

P1M1        DATA    091H
```

```
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

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

            SJMP    $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"

sfr     P_SW1   =   0xa2;

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

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

    while (1);
}
```

3.4.5 I2C切换

汇编代码

```
;测试工作频率为11.0592MHz

P_SW2       DATA    0BAH

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P_SW2,#00H          ;SCL/P1.5, SDA/P1.4
;           MOV     P_SW2,#10H          ;SCL_2/P5.5, SDA_2/P5.4

            SJMP    $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"

sfr     P_SW2   =   0xba;

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
        P1M0 = 0x00;
        P1M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;
```

```
        P_SW2 = 0x00;                   //SCL/P1.5, SDA/P1.4
//      P_SW2 = 0x10;                   //SCL_2/P5.5, SDA_2/P5.4

        while (1);
}
```

3.4.6 比较器输出切换

汇编代码

```
;测试工作频率为11.0592MHz

P_SW2       DATA    0BAH

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

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

            SJMP    $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"

sfr     P_SW2   =   0xba;

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

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

```
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

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

    while (1);
}
```

3.4.7 主时钟输出切换

汇编代码

```
;测试工作频率为11.0592MHz

P_SW2       DATA    0BAH

CLKOCR      EQU     0FE05H

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P_SW2,#80H
            MOV     A,#04H          ;IRC24M/4 output via MCLKO/P5.4
;           MOV     A,#84H          ;IRC24M/4 output via MCLKO_2/P1.6
            MOV     DPTR,#CLKOCR
            MOVX    @DPTR,A
            MOV     P_SW2,#00H

            SJMP    $

            END
```

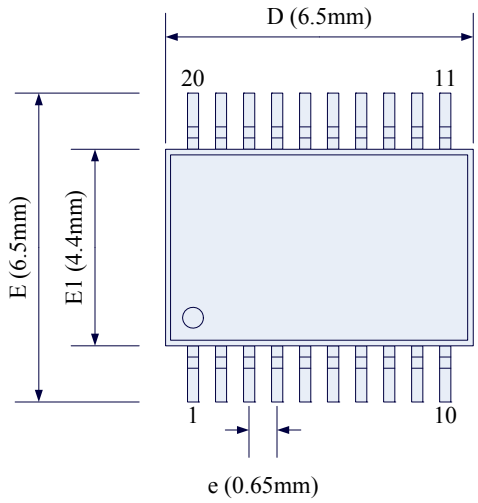
C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
```

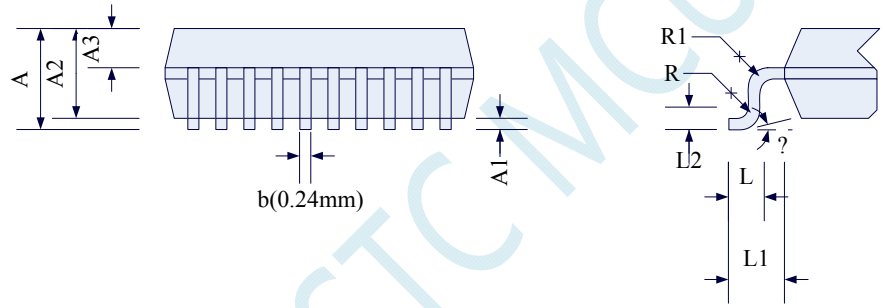
```
#define   CLKOCR      (*(unsigned char volatile xdata *)0xfe00)

sfr       P_SW2       =   0xba;

sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;
    CLKOCR = 0x04;                  //IRC24M/4 output via MCLKO/P5.4
//  CLKOCR = 0x84;                  //IRC24M/4 output via MCLKO_2/P1.6
    P_SW2 = 0x00;

    while (1);
}
```

4 封装尺寸图

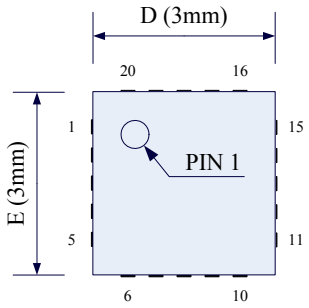
4.1 TSSOP20 封装尺寸图

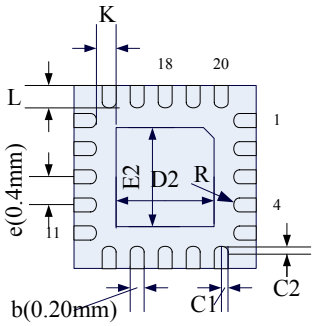


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.20
A1	0.05	-	0.15
A2	0.90	1.00	1.05
A3	0.34	0.44	0.54
b	0.20	0.24	0.28
D	6.40	6.50	6.60
E	6.20	6.50	6.60
E1	4.30	4.40	4.50
e	0.65BSC		
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.09	-	-
R2	0.09	-	-

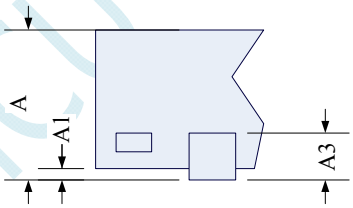


4.2 QFN20 封装尺寸图（3mm*3mm）

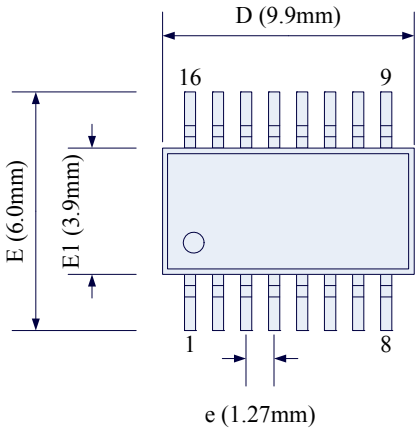




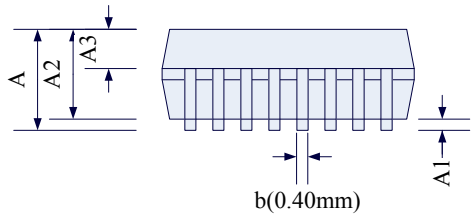
一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0	0.02	0.05
A2	0.50	0.55	0.60
A3	-	0.20REF	-
b	0.15	0.20	0.25
D	2.90	3.00	3.10
E	2.90	3.00	3.10
D2	1.40	1.50	1.60
E2	1.40	1.50	1.60
e	0.30	0.40	0.50
L	0.35	0.40	0.45
K	0.35REF		
R	0.085	-	-
C1	-	0.07	-
C2	-	0.07	-

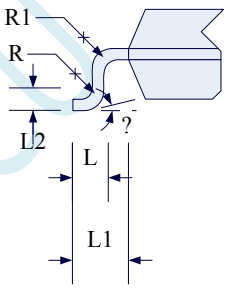


4.3 SOP16 封装尺寸图

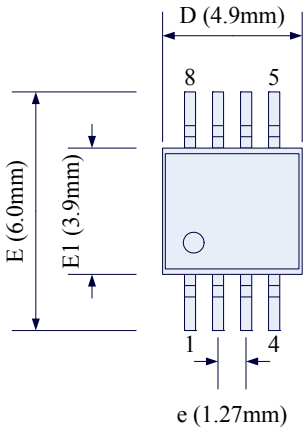


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	1.35	1.60	1.75
A1	0.10	0.15	0.25
A2	1.25	1.45	1.65
A3	0.55	0.65	0.75
b	0.35	0.40	0.45
D	9.80	9.90	10.00
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e		1.27BSC	
L	0.45	0.60	0.80
L1		1.04REF	
L2		0.25BSC	
R1	0.07	-	-
R2	0.07	-	-

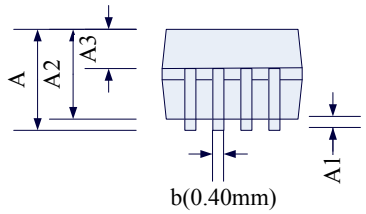


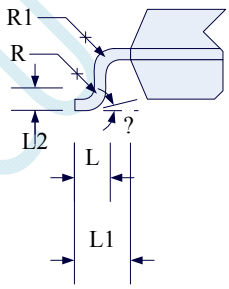


4.4 SOP8 封装尺寸图

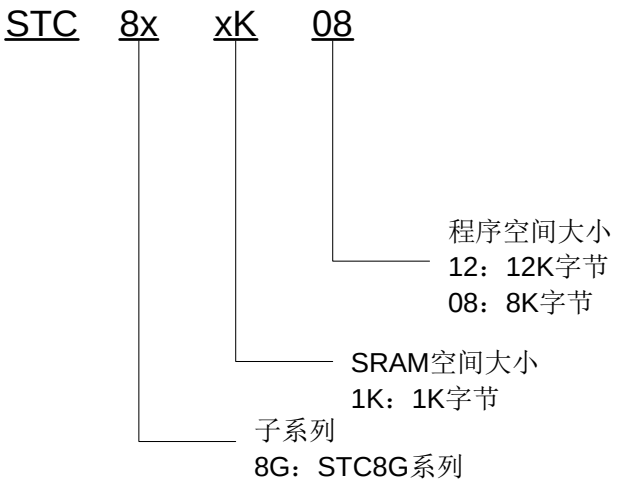


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	1.35	1.60	1.75
A1	0.10	0.15	0.25
A2	1.25	1.45	1.65
A3	0.55	0.65	0.75
b	0.35	0.40	0.45
D	4.80	4.90	5.00
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e		1.27BSC	
L	0.45	0.60	0.80
L1		1.04REF	
L2		0.25BSC	
R1	0.07	-	-
R2	0.07	-	-





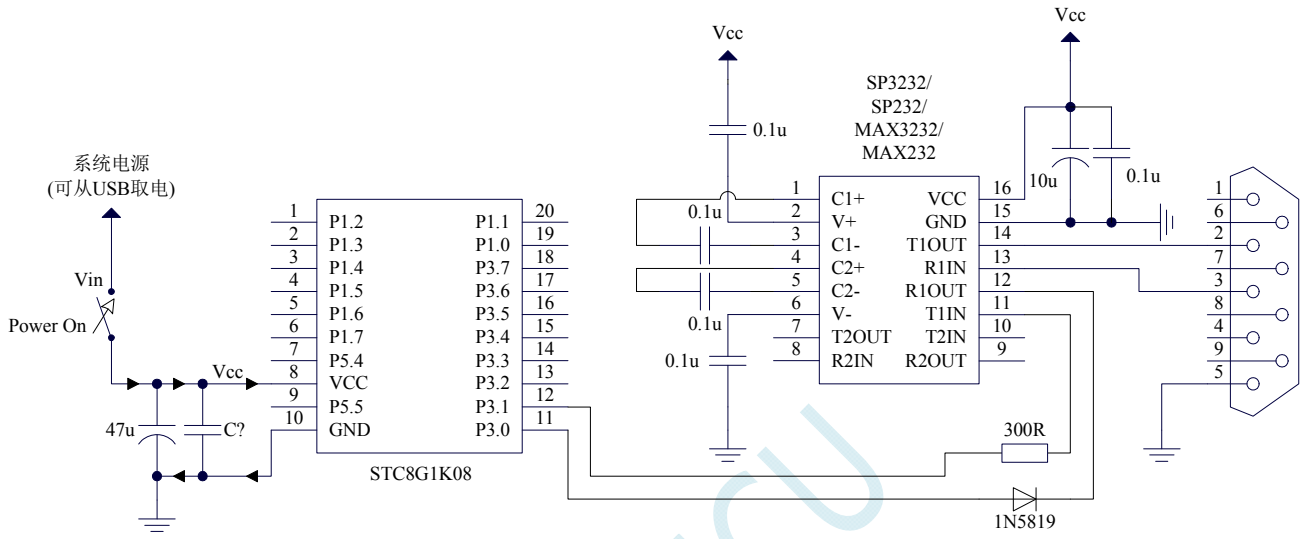
4.5 STC8 系列单片机命名规则



5 ISP下载及典型应用线路图

5.1 STC8G系列ISP下载应用线路图

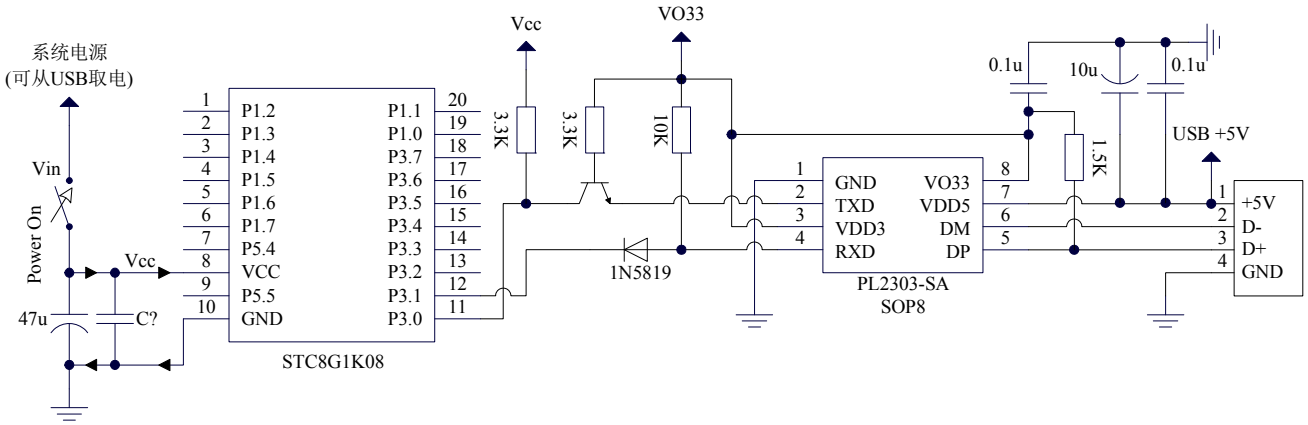
5.1.1 使用RS-232 转换器下载



47u钽电容（封装3528）参考价<RMB ￥0.16
22u独石电容（封装0603）参考价<RMB ￥0.038
10u独石电容（封装0603）参考价<RMB ￥0.028
0.1u独石电容（封装0603）参考价<RMB ￥0.005

	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

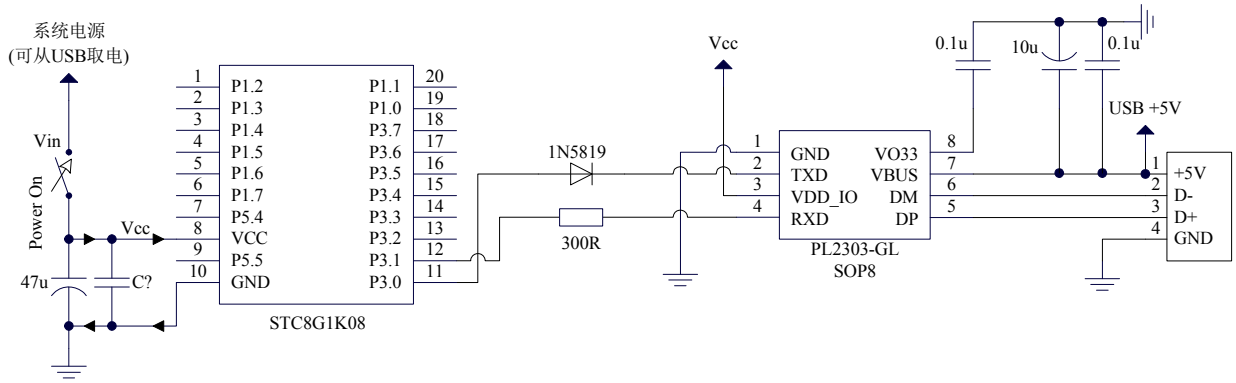
5.1.2 使用PL2303-SA下载



	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

47u钽电容（封装3528）参考价<RMB ￥0.16
22u独石电容（封装0603）参考价<RMB ￥0.038
10u独石电容（封装0603）参考价<RMB ￥0.028
0.1u独石电容（封装0603）参考价<RMB ￥0.005

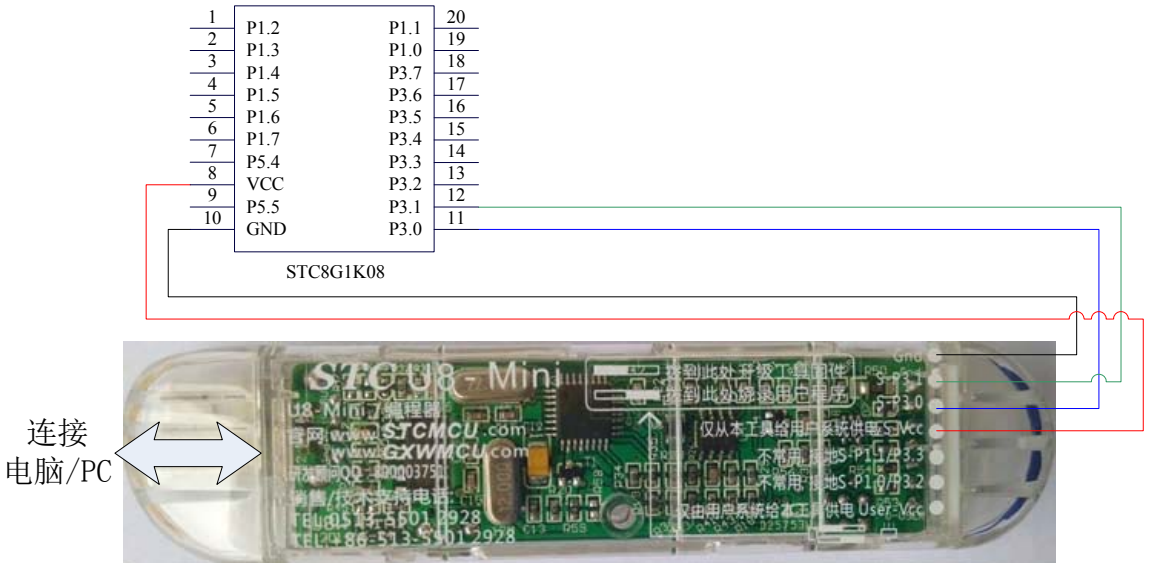
5.1.3 使用PL2303-GL下载



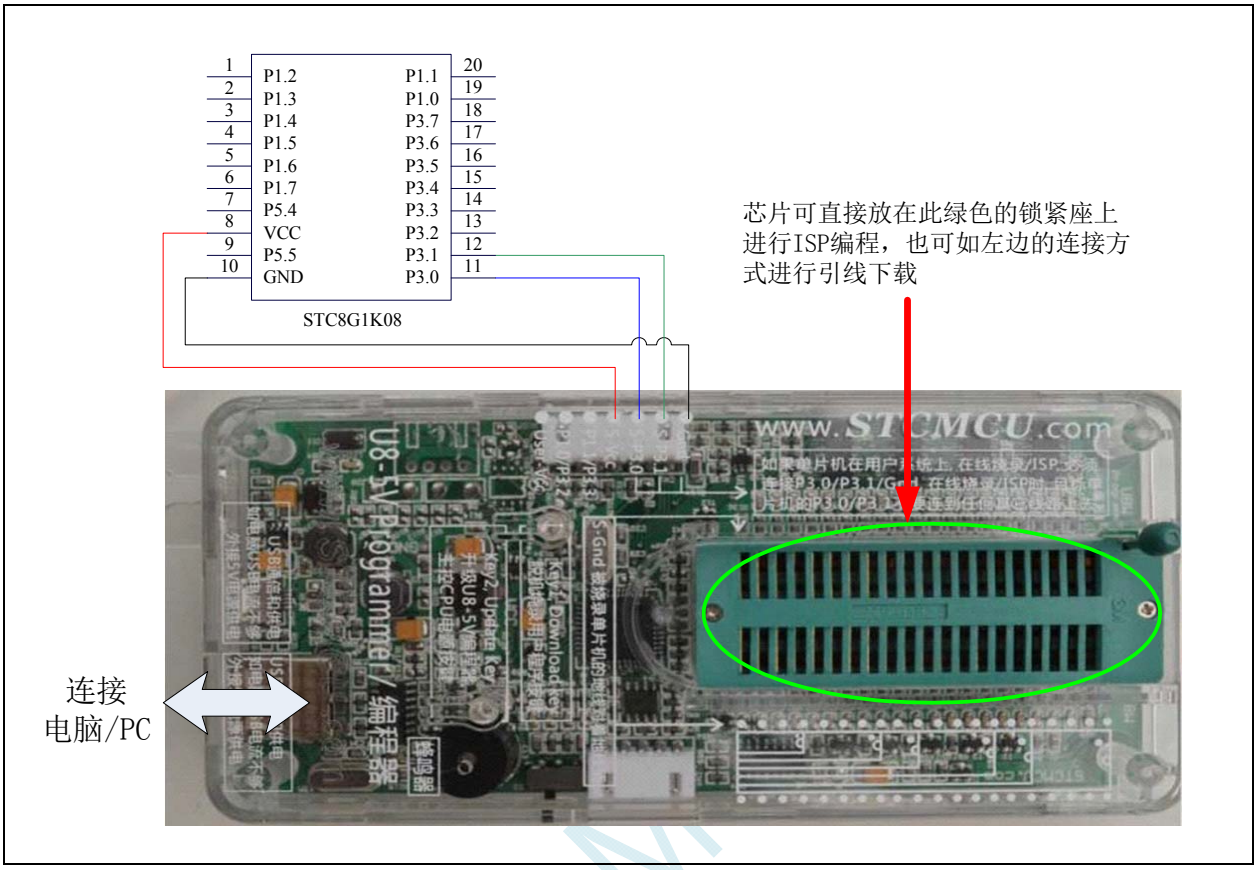
	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

47u钽电容（封装3528）参考价<RMB ￥0.16
22u独石电容（封装0603）参考价<RMB ￥0.038
10u独石电容（封装0603）参考价<RMB ￥0.028
0.1u独石电容（封装0603）参考价<RMB ￥0.005

5.1.4 使用U8-Mini工具下载

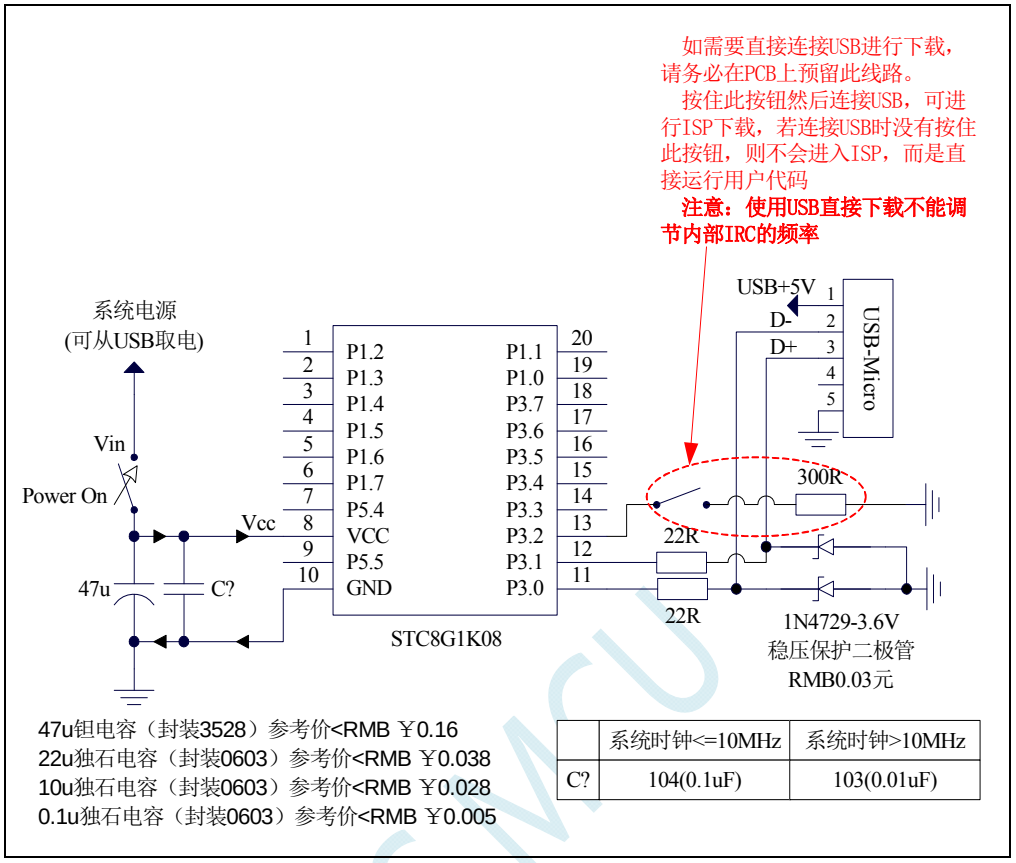


5.1.5 使用U8W工具下载



5.1.6 USB直接ISP下载

注：使用 USB 下载时需要将 P3.2 接 GND 才可进行正常下载



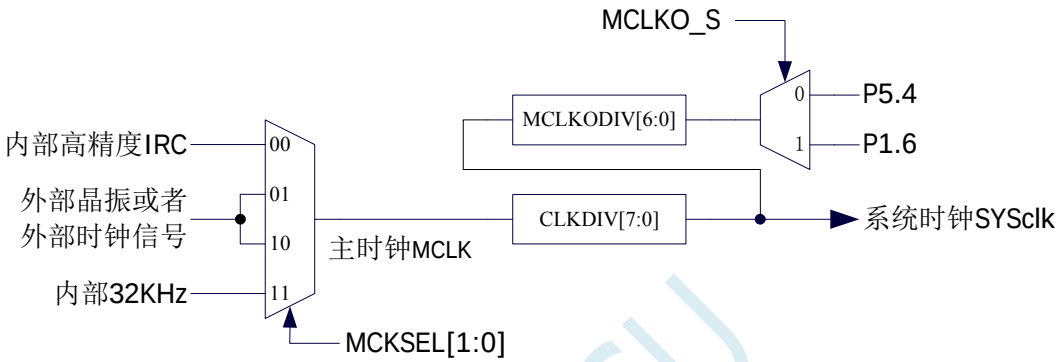
	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

6 时钟、复位与电源管理

6.1 系统时钟控制

系统时钟控制器为单片机的 CPU 和所有外设系统提供时钟源，系统时钟有 3 个时钟源可供选择：内部高精度 IRC、内部 32KHz 的 IRC（误差较大）和外部晶振。用户可通过程序分别使能和关闭各个时钟源，以及内部提供时钟分频以达到降低功耗的目的。

单片机进入掉电模式后，时钟控制器将会关闭所有的时钟源



系统时钟结构图

相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CKSEL	时钟选择寄存器	FE00H	-						MCKSEL[1:0]		xxxx,xx00
CLKDIV	时钟分频寄存器	FE01H									0000,0100
IRC24MCR	内部振荡器控制寄存器	FE02H	ENIRC24M	-	-	-	-	-	-	IRC24MST	1xxx,xxx0
XOSCCR	外部晶振控制寄存器	FE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST	00xx,xxx0
IRC32KCR	内部 32K 振荡器控制寄存器	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0
MCLKOCR	主时钟输出控制寄存器	FE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000

CKSEL（系统时钟选择寄存器）

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

MCKSEL[1:0]：主时钟源选择

MCKSEL[1:0]	主时钟源
00	内部高精度 IRC
01	外部晶体振荡器或外部输入时钟信号
10	
11	内部 32KHz 低速 IRC

CLKDIV（时钟分频寄存器）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0

CLKDIV	FE01H	

CLKDIV：主时钟分频系数。系统时钟 SYSCLK 是对主时钟 MCLK 进行分频后的时钟信号。

CLKDIV	系统时钟频率
0	MCLK/1
1	MCLK/1
2	MCLK/2
3	MCLK/3
…	…
x	MCLK/x
…	…
255	MCLK/255

IRC24MCR（内部高精度 IRC 控制寄存器）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRC24MCR	FE02H	ENIRC24M	-	-	-	-	-	-	IRC24MST

ENIRC24M：内部高精度 IRC 使能位

0：关闭内部高精度 IRC

1：使能内部高精度 IRC

IRC24MST：内部高精度 IRC 频率稳定标志位。（只读位）

当内部的 IRC 从停振状态开始使能后，必须经过一段时间，振荡器的频率才会稳定，当振荡器频率稳定后，时钟控制器会自动将 IRC24MST 标志位置 1。所以当用户程序需要将时钟切换到使用内部 IRC 时，首先必须设置 ENIRC24M=1 使能振荡器，然后一直查询振荡器稳定标志位 IRC24MST，直到标志位变为 1 时，才可进行时钟源切换。

XOSCCR（外部振荡器控制寄存器）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
XOSCCR	FE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST

ENXOSC：外部晶体振荡器使能位

0：关闭外部晶体振荡器

1：使能外部晶体振荡器

XITYPE：外部时钟源类型

0：外部时钟源是外部时钟信号（或有源晶振）。信号源只需连接单片机的 XTALI（P1.7）

1：外部时钟源是晶体振荡器。信号源连接单片机的 XTALI（P1.7）和 XTALO（P1.6）

XOSCST：外部晶体振荡器频率稳定标志位。（只读位）

当外部晶体振荡器从停振状态开始使能后，必须经过一段时间，振荡器的频率才会稳定，当振荡器频率稳定后，时钟控制器会自动将 XOSCST 标志位置 1。所以当用户程序需要将时钟切换到使用外部晶体振荡器时，首先必须设置 ENXOSC=1 使能振荡器，然后一直查询振荡器稳定标志位 XOSCST，直到标志位变为 1 时，才可进行时钟源切换。

IRC32KCR（内部 32KHz 低速 IRC 控制寄存器）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRC32KCR	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST

ENIRC32K：内部 32K 低速 IRC 使能位

0：关闭内部 32K 低速 IRC

1：使能内部 32K 低速 IRC

IRC32KST：内部 32K 低速 IRC 频率稳定标志位。（只读位）

当内部 32K 低速 IRC 从停振状态开始使能后，必须经过一段时间，振荡器的频率才会稳定，当振荡器频率稳定后，时钟控制器会自动将 IRC32KST 标志位置 1。所以当用户程序需要将时钟切换到使用内部 32K 低速 IRC 时，首先必须设置 ENIRC32K=1 使能振荡器，然后一直查询振荡器稳定标志位 IRC32KST，直到标志位变为 1 时，才可进行时钟源切换。

MCLKOCR（主时钟输出控制寄存器）

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

MCLKODIV[6:0]：主时钟输出分频系数

（**注意：主时钟分频输出的时钟源是经过 CLKDIV 分频后的系统时钟**）

MCLKODIV[6:0]	系统时钟分频输出频率
0000000	不输出时钟
0000001	SYSClk/1
0000010	SYSClk /2
0000011	SYSClk /3
...	...
1111110	SYSClk /126
1111111	SYSClk /127

MCLKO_S：系统时钟输出管脚选择

0：系统时钟分频输出到 P5.4 口

1：系统时钟分频输出到 P1.6 口

6.2 STC8G系列内部IRC频率调整

STC8G 系列单片机内部均集成有一颗高精度内部 IRC 振荡器。在用户使用 ISP 下载软件进行下载时，ISP 下载软件会根据用户所选择/设置的频率自动进行调整，一般频率值可调整到±0.3%以下，调整后的频率在全温度范围内（－40℃～85℃）的温漂可达-1.35%～1.30%。

STC8G 系列内部 IRC 有两个频段，频段的中心频率分别为 20MHz 和 33MHz，20M 频段的调节范围约为 14.7MHz～26MHz，33M 频段的调节范围约为 24.5MHz～42.2MHz（注意：不同的芯片以及不同的生成批次可能会有约 5%左右的制造误差）。**经实际测试，部分芯片的最高工作频率只能为 39MHz，所以为了安全起见，建议用户在 ISP 下载时设置 IRC 频率不要高于 35MHz**。

注意：对于一般用户，内部 IRC 频率的调整可以不用关心，因为频率调整工作在进行 ISP 下载时已经自动完成了。所以若用户不需要自行调整频率，那么下面相关的 4 个寄存器也不能随意修改，否则可能会导致工作频率变化。

内部 IRC 频率调整主要使用下面的 4 个寄存器进行调整

相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
IRCBAND	IRC 频段选择	9DH	-	-	-	-	-	-	-	SEL	0000,00nn
LIRTRIM	IRC 频率微调寄存器	9EH	-	-	-	-	-	-	LIRTRIM[1:0]		0000,00nn
IRTRIM	IRC 频率调整寄存器	9FH	IRTRIM[7:0]								nnnn,nnnn
CLKDIV	时钟分频寄存器	FE01H	CLKDIV[7:0]								0000,0100

IRC 频段选择寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRCBAND	9DH	-	-	-	-	-	-	-	SEL

SEL：频段选择

0：选择 20MHz 频段

1：选择 33MHz 频段

内部 IRC 频率调整寄存器

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

IRTRIM[7:0]：内部高精度 IRC 频率调整寄存器

IRTRIM 可对 IRC 频率进行 256 个等级的调整，每个等级所调整的频率值在整体上呈线性分布，局部会有波动。宏观上，每一级所调整的频率约为 0.24%，即 IRTRIM 为（n+1）时的频率比 IRTRIM 为（n）时的频率约快 0.24%。但由于 IRC 频率调整并非每一级都是 0.24%（每一级所调整频率的最大值约为 0.55%，最小值约为 0.02%，整体平均值约为 0.24%），所以会造成局部波动。

内部 IRC 频率微调寄存器

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

LIRTRIM[1:0]：内部高精度 IRC 频率微调寄存器

LIRTRIM 可对 IRC 频率进行 3 个等级的调整，3 个等级所调整的频率范围如下表所示：

LIRTRIM[1:0]	调整的频率范围
00	不微调
01	调整约 0.10%
10	调整约 0.04%
11	调整约 0.10%

CLKDIV（时钟分频寄存器）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CLKDIV	FE01H								

CLKDIV：主时钟分频系数。系统时钟 SYSCLK 是对主时钟 MCLK 进行分频后的时钟信号。

CLKDIV	系统时钟频率
0	MCLK/1
1	MCLK/1
2	MCLK/2
3	MCLK/3
…	…
x	MCLK/x
…	…
255	MCLK/255

STC8G 系列内部的两个频段的可调范围分别为 14.7MHz～26MHz 和 24.5MHz～42.2MHz。虽然 33MHz 频段的上限可调到 40MHz 以上，但芯片内部的程序存储器无法运行到 40MHz 以上的速度，所以用户在 ISP 下载时设置内部 IRC 频率不能高于 40MHz，一般建议用户设置为 35MHz 以下。若用户需要较低的工作频率时，可使用 CLKDIV 寄存器对调节后的频率进行分频，例如用户需要 11.0592MHz 的频率，使用内部 IRC 直接调整是无法得到这个频率的，但可将内部 IRC 调整到 22.1184MHz，在使用 CLKDIV 进行 2 分频即可得到 11.0592MHz。

6.3 系统复位

STC8G 系列单片机的复位分为硬件复位和软件复位两种。

硬件复位时，所有的寄存器的值会复位到初始值，系统会重新读取所有的硬件选项。同时根据硬件选项所设置的上电等待时间进行上电等待。硬件复位主要包括：

- 上电复位
- 低压复位
- 复位脚复位（低电平复位）
- 看门狗复位

软件复位时，除与时钟相关的寄存器保持不变外，其余的所有寄存器的值会复位到初始值，软件复位不会重新读取所有的硬件选项。软件复位主要包括：

- 写 IAP_CONTR 的 SWRST 所触发的复位

相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
WDT_CONTR	看门狗控制寄存器	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]			0x00,0000
IAP_CONTR	IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-				0000,xxxx
RSTCFG	复位配置寄存器	FFH	-	ENLVR	-	P54RST	-	-	LVDS[1:0]		0000,0000

WDT_CONTR（看门狗控制寄存器）

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

WDT_FLAG：看门狗溢出标志

看门狗发生溢出时，硬件自动将此位置 1，需要软件清零。

EN_WDT：看门狗使能位

0：对单片机无影响

1：启动看门狗定时器

CLR_WDT：看门狗定时器清零

0：对单片机无影响

1：清零看门狗定时器，硬件自动将此位复位

IDL_WDT：IDLE 模式时的看门狗控制位

0：IDLE 模式时看门狗停止计数

1：IDLE 模式时看门狗继续计数

WDT_PS[2:0]：看门狗定时器时钟分频系数

WDT_PS[2:0]	分频系数	12M 主频时的溢出时间	20M 主频时的溢出时间
000	2	≈ 65.5 毫秒	≈ 39.3 毫秒
001	4	≈ 131 毫秒	≈ 78.6 毫秒
010	8	≈ 262 毫秒	≈ 157 毫秒
011	16	≈ 524 毫秒	≈ 315 毫秒
100	32	≈ 1.05 秒	≈ 629 毫秒
101	64	≈ 2.10 秒	≈ 1.26 秒
110	128	≈ 4.20 秒	≈ 2.52 秒
111	256	≈ 8.39 秒	≈ 5.03 秒

看门狗溢出时间计算公式如下：

$$看门狗溢出时间 = \frac{12 \times 32768 \times 2^{(WDT\_PS+1)}}{SYSclk}$$

IAP_CONTR（IAP 控制寄存器）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-			

SWBS：软件复位启动选择

0：软件复位后从用户程序区开始执行代码。用户数据区的数据保持不变。

1：软件复位后从系统 ISP 区开始执行代码。用户数据区的数据会被初始化。

SWRST：软件复位触发位

0：对单片机无影响

1：触发软件复位

RSTCFG（复位配置寄存器）

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

ENLVR：低压复位控制位

0：禁止低压复位。当系统检测到低压事件时，会产生低压中断

1：使能低压复位。当系统检测到低压事件时，自动复位

P54RST：RST 管脚功能选择

0：RST 管脚用作普通 I/O 口（P5.4）

1：RST 管脚用作复位脚（低电平复位）

LVDS[1:0]：低压检测门槛电压设置

LVDS[1:0]	低压检测门槛电压
00	2.0V
01	2.4V
10	2.7V
11	3.0V

6.4 系统电源管理

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000

PCON（电源控制寄存器）

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

LVDF：低压检测标志位。当系统检测到低压事件时，硬件自动将此位置 1，并向 CPU 提出中断请求。此位需要用户软件清零。

POF：上电标志位。当硬件自动将此位置 1。

PD：掉电模式控制位

0：无影响

1：单片机进入掉电模式，CPU 以及全部外设均停止工作。唤醒后硬件自动清零。

IDL：IDLE（空闲）模式控制位

0：无影响

1：单片机进入 IDLE 模式，只有 CPU 停止工作，其他外设依然在运行。唤醒后硬件自动清零

6.5 范例程序

6.5.1 选择系统时钟源

汇编代码

```
;测试工作频率为11.0592MHz

P_SW2       DATA    0BAH

CKSEL       EQU     0FE00H
CLKDIV      EQU     0FE01H
IRC24MCR    EQU     0FE02H
XOSCCR      EQU     0FE03H
IRC32KCR    EQU     0FE04H

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P_SW2,#80H
            MOV     A,#00H          ;选择内部IRC ( 默认 )
            MOV     DPTR,#CKSEL
            MOVX    @DPTR,A
            MOV     P_SW2,#00H

;           MOV     P_SW2,#80H
;           MOV     A,#0C0H         ;启动外部晶振
;           MOV     DPTR,#XOSCCR
;           MOVX    @DPTR,A
;           MOVX    A,@DPTR
;           JNB     ACC.0,$-1       ;等待时钟稳定
;           CLR     A               ;时钟不分频
;           MOV     DPTR,#CLKDIV
            MOVX    @DPTR,A
;           MOV     A,#01H          ;选择外部晶振
;           MOV     DPTR,#CKSEL
;           MOVX    @DPTR,A
;           MOV     P_SW2,#00H

;           MOV     P_SW2,#80H
;           MOV     A,#80H          ;启动内部32K IRC
;           MOV     DPTR,#IRC32KCR
```

```
;               MOVX    @DPTR,A
;               MOVX    A,@DPTR
;               JNB     ACC.0,$-1               ;等待时钟稳定
;               CLR     A                       ;时钟不分频
;               MOV     DPTR,#CLKDIV
;               MOVX    @DPTR,A
;               MOV     A,#03H                  ;选择内部32K
;               MOV     DPTR,#CKSEL
;               MOVX    @DPTR,A
;               MOV     P_SW2,#00H

                JMP     $

                END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

#define     CKSEL       (*(unsigned char volatile xdata *)0xfe00)
#define     CLKDIV      (*(unsigned char volatile xdata *)0xfe01)
#define     IRC24MCR    (*(unsigned char volatile xdata *)0xfe02)
#define     XOSCCR      (*(unsigned char volatile xdata *)0xfe03)
#define     IRC32KCR    (*(unsigned char volatile xdata *)0xfe04)

sfr         P_SW2       =   0xba;

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;
    CKSEL = 0x00;                               //选择内部IRC ( 默认 )
    P_SW2 = 0x00;

/*
    P_SW2 = 0x80;
    XOSCCR = 0xc0;                              //启动外部晶振
    while (!(XOSCCR & 1));                      //等待时钟稳定
    CLKDIV = 0x00;                              //时钟不分频
    CKSEL = 0x01;                               //选择外部晶振
    P_SW2 = 0x00;
*/
```

```
/*
    P_SW2 = 0x80;
    IRC32KCR = 0x80;                    //启动内部32K IRC
    while (!(IRC32KCR & 1));            //等待时钟稳定
    CLKDIV = 0x00;                      //时钟不分频
    CKSEL = 0x03;                       //选择内部32K
    P_SW2 = 0x00;
*/
    while (1);
}
```

6.5.2 主时钟分频输出

汇编代码

```
;测试工作频率为11.0592MHz

P_SW2       DATA    0BAH

MCLKOCR     EQU     0FE05H

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P_SW2,#80H
;           MOV     A,#01H          ;主时钟输出到P5.4 口
;           MOV     A,#02H          ;主时钟2 分频输出到P5.4 口
            MOV     A,#04H          ;主时钟4 分频输出到P5.4 口
;           MOV     A,#84H          ;主时钟4 分频输出到P1.6 口
            MOV     DPTR,#MCLKOCR
            MOVX    @DPTR,A
            MOV     P_SW2,#00H

            JMP     $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz
```

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

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

sfr        P_SW2      =    0xba;

sfr        P1M1       =    0x91;
sfr        P1M0       =    0x92;
sfr        P3M1       =    0xb1;
sfr        P3M0       =    0xb2;
sfr        P5M1       =    0xc9;
sfr        P5M0       =    0xca;

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;
//  MCLKOCR = 0x01;                 //主时钟输出到 P5.4 口
//  MCLKOCR = 0x02;                 //主时钟 2 分频输出到 P5.4 口
    MCLKOCR = 0x04;                 //主时钟 4 分频输出到 P5.4 口
//  MCLKOCR = 0x84;                 //主时钟 4 分频输出到 P1.6 口
    P_SW2 = 0x00;

    while (1);
}
```

6.5.3 看门狗定时器应用

汇编代码

```
;测试工作频率为 11.0592MHz

WDT_CONTR  DATA    0C1H

P1M1       DATA    091H
P1M0       DATA    092H
P3M1       DATA    0B1H
P3M0       DATA    0B2H
P5M1       DATA    0C9H
P5M0       DATA    0CAH

           ORG     0000H
           LJMP    MAIN

           ORG     0100H
MAIN:
           MOV     SP, #5FH
           MOV     P1M0, #00H
           MOV     P1M1, #00H
           MOV     P3M0, #00H
           MOV     P3M1, #00H
           MOV     P5M0, #00H
```

```
            MOV         P5M1, #00H

;           MOV         WDT_CONTR,#23H      ;使能看门狗,溢出时间约为0.5s
            MOV         WDT_CONTR,#24H      ;使能看门狗,溢出时间约为1s
;           MOV         WDT_CONTR,#27H      ;使能看门狗,溢出时间约为8s
            CLR         P3.2                ;测试端口
LOOP:
;           ORL         WDT_CONTR,#10H      ;清看门狗,否则系统复位
            JMP         LOOP

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

//  WDT_CONTR = 0x23;               //使能看门狗,溢出时间约为0.5s
    WDT_CONTR = 0x24;               //使能看门狗,溢出时间约为1s
//  WDT_CONTR = 0x27;               //使能看门狗,溢出时间约为8s
    P32 = 0;                        //测试端口

    while (1)
    {
//      WDT_CONTR |= 0x10;          //清看门狗,否则系统复位
    }
}
```

6.5.4 软复位实现自定义下载

汇编代码

```
;测试工作频率为11.0592MHz

IAP_CONTR   DATA        0C7H

P1M1        DATA        091H
```

```
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            SETB    P3.2
            SETB    P3.3
LOOP:
            JB      P3.2,LOOP
            JB      P3.3,LOOP
            MOV     IAP_CONTR,#60H      ;检查到P3.2 和P3.3 同时为0 时复位到ISP
            JMP     $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P32 = 1;                            //测试端口
    P33 = 1;                            //测试端口
```

```
    while (1)
    {
        if (!P32 && !P33)
        {
            IAP_CONTR |= 0x60;                  //检查到P3.2 和P3.3 同时为0 时复位到ISP
        }
    }
}
```

6.5.5 低压检测

汇编代码

```
;测试工作频率为11.0592MHz

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

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

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0033H
            LJMP    LVDISR

            ORG     0100H
LVDISR:
            ANL     PCON,#NOT LVDF      ;清中断标志
            CPL     P3.2                ;测试端口
            RETI
MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            ANL     PCON,#NOT LVDF      ;上电后需要先清LVDF 标志
;           MOV     RSTCFG,#ENLVR | LVD3V0  ;使能3.0V 时低压复位,不产生LVD 中断
            MOV     RSTCFG,#LVD3V0      ;使能3.0V 时低压中断
            SETB    ELVD                ;使能LVD 中断
            SETB    EA
            JMP     $
```

```
        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

void Lvd_Isr() interrupt 6
{
    PCON &= ~LVDF;                  //清中断标志
    P32 = ~P32;                     //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    PCON &= ~LVDF;                  //测试端口
//  RSTCFG = ENLVR | LVD3V0;        //使能 3.0V 时低压复位,不产生 LVD 中断
    RSTCFG = LVD3V0;                //使能 3.0V 时低压中断
    ELVD = 1;                       //使能 LVD 中断
    EA = 1;

    while (1);
}
```

6.5.6 省电模式

汇编代码

```
;测试工作频率为11.0592MHz

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

```
P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0003H
            LJMP    INT0ISR

            ORG     0100H
INT0ISR:
            CPL     P3.4                ;测试端口
            RETI
MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            SETB    EX0                 ;使能 INT0 中断,用于唤醒 MCU
            SETB    EA
            NOP
            NOP
;           MOV     PCON,#IDL           ;MCU 进入 IDLE 模式
            MOV     PCON,#PD            ;MCU 进入掉电模式
            NOP
            NOP
            CLR     P3.5                ;测试端口
            JMP     $

            END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

sfr       P1M1      =     0x91;
sfr       P1M0      =     0x92;
sfr       P3M1      =     0xb1;
sfr       P3M0      =     0xb2;
sfr       P5M1      =     0xc9;
sfr       P5M0      =     0xca;

void INT0_Isr() interrupt 0
```

```
{
    P34 = ~P34;                                 //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    EX0 = 1;                                    //使能 INT0 中断,用于唤醒 MCU
    EA = 1;
    _nop_();
    _nop_();
    PCON = IDL;                                 //MCU 进入 IDLE 模式
//  PCON = PD;                                  //MCU 进入掉电模式
    _nop_();
    _nop_();
    P35 = 0;

    while (1);
}
```

6.5.7 使用INT0/INT1/INT2/INT3/INT4 中断唤醒MCU

汇编代码

```
;测试工作频率为 11.0592MHz

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

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

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

            ORG     0100H
```

```
INT0ISR:
        CPL     P1.0            ;测试端口
        RETI
INT1ISR:
        CPL     P1.0            ;测试端口
        RETI
INT2ISR:
        CPL     P1.0            ;测试端口
        RETI
INT3ISR:
        CPL     P1.0            ;测试端口
        RETI
INT4ISR:
        CPL     P1.0            ;测试端口
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        CLR     IT0             ;使能 INT0 上升沿和下降沿中断
;       SETB    IT0             ;使能 INT0 下降沿中断
        SETB    EX0             ;使能 INT0 中断

        CLR     IT1             ;使能 INT1 上升沿和下降沿中断
;       SETB    IT1             ;使能 INT1 下降沿中断
        SETB    EX1             ;使能 INT1 中断

        MOV     INTCLKO,#EX2    ;使能 INT2 下降沿中断
        ORL     INTCLKO,#EX3    ;使能 INT3 下降沿中断
        ORL     INTCLKO,#EX4    ;使能 INT4 下降沿中断

        SETB    EA

        MOV     PCON,#02H       ;MCU 进入掉电模式
        NOP                     ;掉电唤醒后立即进入中断服务程序
        NOP
LOOP:
        CPL     P1.1
        JMP     LOOP

        END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void INT0_Isr() interrupt 0
{
    P10 = !P10;                                 //测试端口
}

void INT1_Isr() interrupt 2
{
    P10 = !P10;                                 //测试端口
}

void INT2_Isr() interrupt 10
{
    P10 = !P10;                                 //测试端口
}

void INT3_Isr() interrupt 11
{
    P10 = !P10;                                 //测试端口
}

void INT4_Isr() interrupt 16
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IT0 = 0;                                    //使能 INT0 上升沿和下降沿中断
//  IT0 = 1;                                    //使能 INT0 下降沿中断
    EX0 = 1;                                    //使能 INT0 中断

    IT1 = 0;                                    //使能 INT1 上升沿和下降沿中断
//  IT1 = 1;                                    //使能 INT1 下降沿中断
    EX1 = 1;                                    //使能 INT1 中断

    INTCLKO = EX2;                              //使能 INT2 下降沿中断
    INTCLKO |= EX3;                             //使能 INT3 下降沿中断
    INTCLKO |= EX4;                             //使能 INT4 下降沿中断

    EA = 1;
```

```
    PCON = 0x02;                    //MCU 进入掉电模式
    _nop_();                        //掉电唤醒后立即进入中断服务程序
    _nop_();

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

6.5.8 使用T0/T1/T2/T3/T4 中断唤醒MCU

汇编代码

```
;测试工作频率为11.0592MHz

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

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

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

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

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

            ORG     0100H
TM0ISR:
            CPL     P1.0            ;测试端口
            RETI
```

```
TM1ISR:
        CPL     P1.0                ;测试端口
        RETI
TM2ISR:
        CPL     P1.0                ;测试端口
        ANL     AUXINTIF,#NOT T2IF  ;清中断标志
        RETI
TM3ISR:
        CPL     P1.0                ;测试端口
        ANL     AUXINTIF,#NOT T3IF  ;清中断标志
        RETI
TM4ISR:
        CPL     P1.0                ;测试端口
        ANL     AUXINTIF,#NOT T4IF  ;清中断标志
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     TMOD,#00H
        MOV     TL0,#66H            ;65536-11.0592M/12/1000
        MOV     TH0,#0FCH
        SETB    TR0                 ;启动定时器
        SETB    ET0                 ;使能定时器中断

        MOV     TL1,#66H            ;65536-11.0592M/12/1000
        MOV     TH1,#0FCH
        SETB    TR1                 ;启动定时器
        SETB    ET1                 ;使能定时器中断

        MOV     T2L,#66H            ;65536-11.0592M/12/1000
        MOV     T2H,#0FCH
        MOV     AUXR,#10H           ;启动定时器
        MOV     IE2,#ET2            ;使能定时器中断

        MOV     T3L,#66H            ;65536-11.0592M/12/1000
        MOV     T3H,#0FCH
        MOV     T4T3M,#08H          ;启动定时器
        ORL     IE2,#ET3            ;使能定时器中断

        MOV     T4L,#66H            ;65536-11.0592M/12/1000
        MOV     T4H,#0FCH
        ORL     T4T3M,#80H          ;启动定时器
        ORL     IE2,#ET4            ;使能定时器中断

        SETB    EA

        MOV     PCON,#02H           ;MCU 进入掉电模式
        NOP                         ;掉电唤醒后不会立即进入中断服务程序,
                                    ;而是等到定时器溢出后才会进入中断服务程序
        NOP
LOOP:
        CPL     P1.1
```

```
        JMP     LOOP

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void TM0_Isr() interrupt 1
{
    P10 = !P10;                                 //测试端口
}

void TM1_Isr() interrupt 3
{
    P10 = !P10;                                 //测试端口
}

void TM2_Isr() interrupt 12
{
    P10 = !P10;                                 //测试端口
    AUXINTIF &= ~T2IF;                          //清中断标志
}

void TM3_Isr() interrupt 19
{
    P10 = !P10;                                 //测试端口
    AUXINTIF &= ~T3IF;                          //清中断标志
}
```

```
void TM4_Isr() interrupt 20
{
    P10 = !P10;                                 //测试端口
    AUXINTIF &= ~T4IF;                          //清中断标志
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;
    TL0 = 0x66;                                 //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                    //启动定时器
    ET0 = 1;                                    //使能定时器中断

    TL1 = 0x66;                                 //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                                    //启动定时器
    ET1 = 1;                                    //使能定时器中断

    T2L = 0x66;                                 //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;                                //启动定时器
    IE2 = ET2;                                  //使能定时器中断

    T3L = 0x66;                                 //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x08;                               //启动定时器
    IE2 |= ET3;                                 //使能定时器中断

    T4L = 0x66;                                 //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M |= 0x80;                              //启动定时器
    IE2 |= ET4;                                 //使能定时器中断

    EA = 1;

    PCON = 0x02;                                //MCU 进入掉电模式
    _nop_();                                    //掉电唤醒后不会立即进入中断服务程序,
                                                //而是等到定时器溢出后才会进入中断服务程序
    _nop_();

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

6.5.9 使用RxD/RxD2 中断唤醒MCU

汇编代码

```
;测试工作频率为11.0592MHz

IE2         DATA    0AFH
ES2         EQU     01H

P_SW1       DATA    0A2H
P_SW2       DATA    0BAH

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART1ISR
            ORG     0043H
            LJMP    UART2ISR

            ORG     0100H
UART1ISR:
            RETI
UART2ISR:
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P_SW1,#00H          ;RXD/P3.0 下降沿唤醒
;           MOV     P_SW1,#40H          ;RXD_2/P3.6 下降沿唤醒
;           MOV     P_SW1,#80H          ;RXD_3/P1.6 下降沿唤醒
;           MOV     P_SW1,#0C0H         ;RXD_4/P4.3 下降沿唤醒

            MOV     P_SW2,#00H          ;RXD2/P1.0 下降沿唤醒
;           MOV     P_SW2,#01H          ;RXD2_2/P4.0 下降沿唤醒

            SETB    ES                  ;使能串口中断
            MOV     IE2,#ES2            ;使能串口中断
            SETB    EA

            MOV     PCON,#02H           ;MCU 进入掉电模式
            NOP                         ;掉电唤醒后不会进入中断服务程序,
            NOP
LOOP:
            CPL     P1.1
            JMP     LOOP
```

```
END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr         IE2         =   0xaf;
#define     ES2             0x01

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

sbit        P11         =   P1^1;

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

void UART1_Isr() interrupt 4
{
}

void UART2_Isr() interrupt 8
{
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW1 = 0x00;                   //RXD/P3.0 下降沿唤醒
//  P_SW1 = 0x40;                   //RXD_2/P3.6 下降沿唤醒
//  P_SW1 = 0x80;                   //RXD_3/P1.6 下降沿唤醒
//  P_SW1 = 0xc0;                   //RXD_4/P4.3 下降沿唤醒

    P_SW2 = 0x00;                   //RXD2/P1.0 下降沿唤醒
//  P_SW2 = 0x01;                   //RXD2_2/P4.0 下降沿唤醒

    ES = 1;                         //使能串口中断
    IE2 = ES2;                      //使能串口中断
    EA = 1;

    PCON = 0x02;                    //MCU 进入掉电模式
    _nop_();                        //掉电唤醒后不会进入中断服务程序,
    _nop_();

    while (1)
```

```
    {
        P11 = ~P11;
    }
}
```

6.5.10 使用LVD中断唤醒MCU

汇编代码

```
;测试工作频率为11.0592MHz

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

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

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0033H
            LJMP    LVDISR

            ORG     0100H
LVDISR:
            ANL     PCON,#NOT LVDF  ;清中断标志
            CPL     P1.0            ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            ANL     PCON,#NOT LVDF  ;上电需要清中断标志
            MOV     RSTCFG,# LVD3V0 ;设置 LVD 电压为3.0V
            SETB    ELVD            ;使能 LVD 中断
            SETB    EA

            MOV     PCON,#02H       ;MCU 进入掉电模式
            NOP                     ;掉电唤醒后立即进入中断服务程序
            NOP
LOOP:
            CPL     P1.1
            JMP     LOOP
```

END

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

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

sfr     P1M1      =   0x91;
sfr     P1M0      =   0x92;
sfr     P3M1      =   0xb1;
sfr     P3M0      =   0xb2;
sfr     P5M1      =   0xc9;
sfr     P5M0      =   0xca;

void LVD_Isr() interrupt 6
{
    PCON &= ~LVDF;                        //清中断标志
    P10 = !P10;                           //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    PCON &= ~LVDF;                        //上电需要清中断标志
    RSTCFG = LVD3V0;                      //设置LVD 电压为3.0V
    ELVD = 1;                             //使能LVD 中断
    EA = 1;

    PCON = 0x02;                          //MCU 进入掉电模式
    _nop_();                              //掉电唤醒后立即进入中断服务程序
    _nop_();

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

6.5.11 使用CCP0/CCP1/CCP2 中断唤醒MCU

汇编代码

```
;测试工作频率为11.0592MHz

CCON    DATA    0D8H
CF      BIT     CCON.7
CR      BIT     CCON.6
CCF2    BIT     CCON.2
CCF1    BIT     CCON.1
CCF0    BIT     CCON.0
CMOD    DATA    0D9H
CL      DATA    0E9H
CH      DATA    0F9H
CCAPM0  DATA    0DAH
CCAP0L  DATA    0EAH
CCAP0H  DATA    0FAH
PCA_PWM0 DATA   0F2H
CCAPM1  DATA    0DBH
CCAP1L  DATA    0EBH
CCAP1H  DATA    0FBH
PCA_PWM1 DATA   0F3H
CCAPM2  DATA    0DCH
CCAP2L  DATA    0ECH
CCAP2H  DATA    0FCH
PCA_PWM2 DATA   0F4H

P_SW1   DATA    0A2H

P1M1    DATA    091H
P1M0    DATA    092H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
        ORG     003BH
        LJMP    PCAISR

        ORG     0100H
PCAISR:
        ANL     CCON,#NOT 8FH       ;清中断标志
        CPL     P1.0                ;测试端口
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     CCON,#00H
        MOV     CMOD,#08H           ;PCA 时钟为系统时钟
```

```
        MOV     CCAPM0,#31H     ;使能 CCP0 口边沿唤醒功能
        MOV     CCAPM1,#31H     ;使能 CCP1 口边沿唤醒功能
        MOV     CCAPM2,#31H     ;使能 CCP2 口边沿唤醒功能
        SETB    CR              ;启动 PCA 计时器
        SETB    EA

        MOV     PCON,#02H       ;MCU 进入掉电模式
        NOP                     ;掉电唤醒后立即进入中断服务程序
        NOP

LOOP:
        CPL     P1.1
        JMP     LOOP

        END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

sfr     P_SW1   =   0xa2;

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void PCA_Isr() interrupt 7
{
```

```
        CCON &= ~0x8f;                          //清中断标志
        P10 = !P10;                             //测试端口
}

void main()
{
        P1M0 = 0x00;
        P1M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        CCON = 0x00;
        CMOD = 0x08;                            //PCA 时钟为系统时钟
        CCAPM0 = 0x31;                          //使能 CCP0 口边沿唤醒功能
        CCAPM1 = 0x31;                          //使能 CCP1 口边沿唤醒功能
        CCAPM2 = 0x31;                          //使能 CCP2 口边沿唤醒功能

        CR = 1;                                 //启动 PCA 计时器
        EA = 1;

        PCON = 0x02;                            //MCU 进入掉电模式
        _nop_();                                //掉电唤醒后立即进入中断服务程序
        _nop_();

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

6.5.12 CMP中断唤醒MCU

汇编代码

```
;测试工作频率为 11.0592MHz

CMPCR1      DATA        0E6H
CMPCR2      DATA        0E7H

P1M1        DATA        091H
P1M0        DATA        092H
P3M1        DATA        0B1H
P3M0        DATA        0B2H
P5M1        DATA        0C9H
P5M0        DATA        0CAH

            ORG         0000H
            LJMP        MAIN
            ORG         00ABH
            LJMP        CMPISR

            ORG         0100H
CMPISR:
            ANL         CMPCR1,#NOT 40H         ;清中断标志
            CPL         P1.0                    ;测试端口
```

```
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     CMPCR2,#00H
            MOV     CMPCR1,#80H         ;使能比较器模块
            ORL     CMPCR1,#30H         ;使能比较器边沿中断
            ANL     CMPCR1,#NOT 08H     ;P3.6 为 CMP+输入脚
            ORL     CMPCR1,#04H         ;P3.7 为 CMP-输入脚
            ORL     CMPCR1,#02H         ;使能比较器输出
            SETB    EA

            MOV     PCON,#02H           ;MCU 进入掉电模式
            NOP                         ;掉电唤醒后立即进入中断服务程序
            NOP

LOOP:
            CPL     P1.1
            JMP     LOOP

            END
```

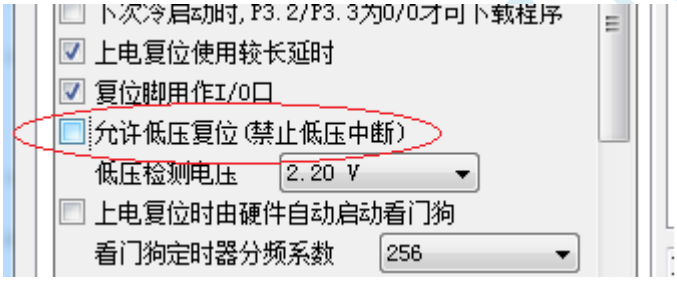
C 语言代码

```
//测试工作频率为11.0592MHz

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

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

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void CMP_Isr() interrupt 21
{
    CMPCR1 &= ~0x40;                    //清中断标志
    P10 = !P10;                         //测试端口
}

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

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

    CMPCR2 = 0x00;
    CMPCR1 = 0x80;                  //使能比较器模块
    CMPCR1 |= 0x30;                 //使能比较器边沿中断
    CMPCR1 &= ~0x08;                //P3.6 为CMP+输入脚
    CMPCR1 |= 0x04;                 //P3.7 为CMP-输入脚
    CMPCR1 |= 0x02;                 //使能比较器输出
    EA = 1;

    PCON = 0x02;                    //MCU 进入掉电模式
    _nop_();                        //掉电唤醒后立即进入中断服务程序
    _nop_();

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

6.5.13 使用LVD功能检测工作电压（电池电压）

若需要使用 LVD 功能检测电池电压，则在 ISP 下载时需要将低压复位功能去掉，如下图 “允许低压复位（禁止低压中断）”的硬件选项的勾选项需要去掉



汇编代码

```
;测试工作频率为11.0592MHz

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

LVDF        EQU     20H             ;PCON.5

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            JMP     MAIN
```

```
        ORG     0100H
MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

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

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

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

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

        MOV     RSTCFG,#LVD2V2
        CALL    DELAY
        ANL     PCON,#NOT LVDF
        CALL    DELAY
        MOV     A,PCON
        ANL     A,#LVDF
        JZ      SKIP
```

```
                MOV         A,B
                CLR         C
                RRC         A
                MOV         B,A

SKIP:
                MOV         A,B
                CPL         A
                MOV         P2,A                ;P2.3~P2.0 显示电池电量
                JMP         LOOP

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

                END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

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

#define     LVDF            0x20                //PCON.5

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

void delay()
{
    int i;

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

```
}

void main()
{
    unsigned char power;

    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    PCON &= ~LVDF;
    RSTCFG = LVD3V0;

    while (1)
    {
        power = 0x0f;

        RSTCFG = LVD3V0;
        delay();
        PCON &= ~LVDF;
        delay();
        if (PCON & LVDF)
        {
            power >>= 1;
            RSTCFG = LVD2V7;
            delay();
            PCON &= ~LVDF;
            delay();
            if (PCON & LVDF)
            {
                power >>= 1;
                RSTCFG = LVD2V4;
                delay();
                PCON &= ~LVDF;
                delay();
                if (PCON & LVDF)
                {
                    power >>= 1;
                    RSTCFG = LVD2V2;
                    delay();
                    PCON &= ~LVDF;
                    delay();
                    if (PCON & LVDF)
                    {
                        power >>= 1;
                    }
                }
            }
        }
        RSTCFG = LVD3V0;
        P2 = ~power;     //P2.3~P2.0 显示电池电量
    }
}
```

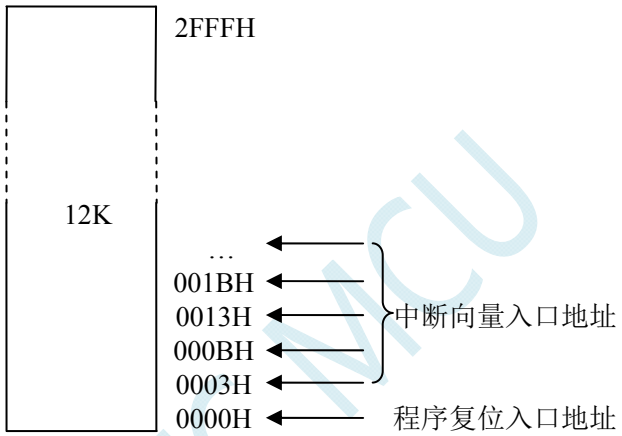
7 存储器

STC8G 系列单片机的程序存储器和数据存储器是各自独立编址的。由于没有提供访问外部程序存储器的总线，所有单片机的所有程序存储器都是片上 Flash 存储器，不能访问外部程序存储器。

STC8G 系列单片机内部集成了大容量的数据存储器，STC8G1K08 系列单片机内部有 1024+256 字节的数据存储器。STC8G 系列单片机内部的数据存储器在物理和逻辑上都分为两个地址空间:内部 RAM(256 字节)和内部扩展 RAM。其中内部 RAM 的高 128 字节的数据存储器与特殊功能寄存器(SFRs)地址重叠，实际使用时通过不同的寻址方式加以区分。

7.1 程序存储器

程序存储器用于存放用户程序、数据以及表格等信息。STC8G 系列单片内部集成了 12K 字节的 Flash 程序存储器。



单片机复位后，程序计数器(PC)的内容为 0000H，从 0000H 单元开始执行程序。另外中断服务程序的入口地址(又称中断向量)也位于程序存储器单元。在程序存储器中，每个中断都有一个固定的入口地址，当中断发生并得到响应后，单片机就会自动跳转到相应的中断入口地址去执行程序。外部中断 0（INT0）的中断服务程序的入口地址是 0003H，定时器/计数器 0（TIMER0）中断服务程序的入口地址是 000BH，外部中断 1（INT1)的中断服务程序的入口地址是 0013H,定时器/计数器 1(TIMER1)的中断服务程序的入口地址是 001BH 等。更多的中断服务程序的入口地址(中断向量)请参考中断介绍章节。

由于相邻中断入口地址的间隔区间仅仅有 8 个字节，一般情况下无法保存完整的中断服务程序，因此在中断响应的地址区域存放一条无条件转移指令，指向真正存放中断服务程序的空间去执行。

STC8G 系列单片机中都包含有 Flash 数据存储器（EEPROM）。以字节为单位进行读/写数据，以 512 字节为页单位进行擦除，可在线反复编程擦写 10 万次以上，提高了使用的灵活性和方便性。

7.2 数据存储器

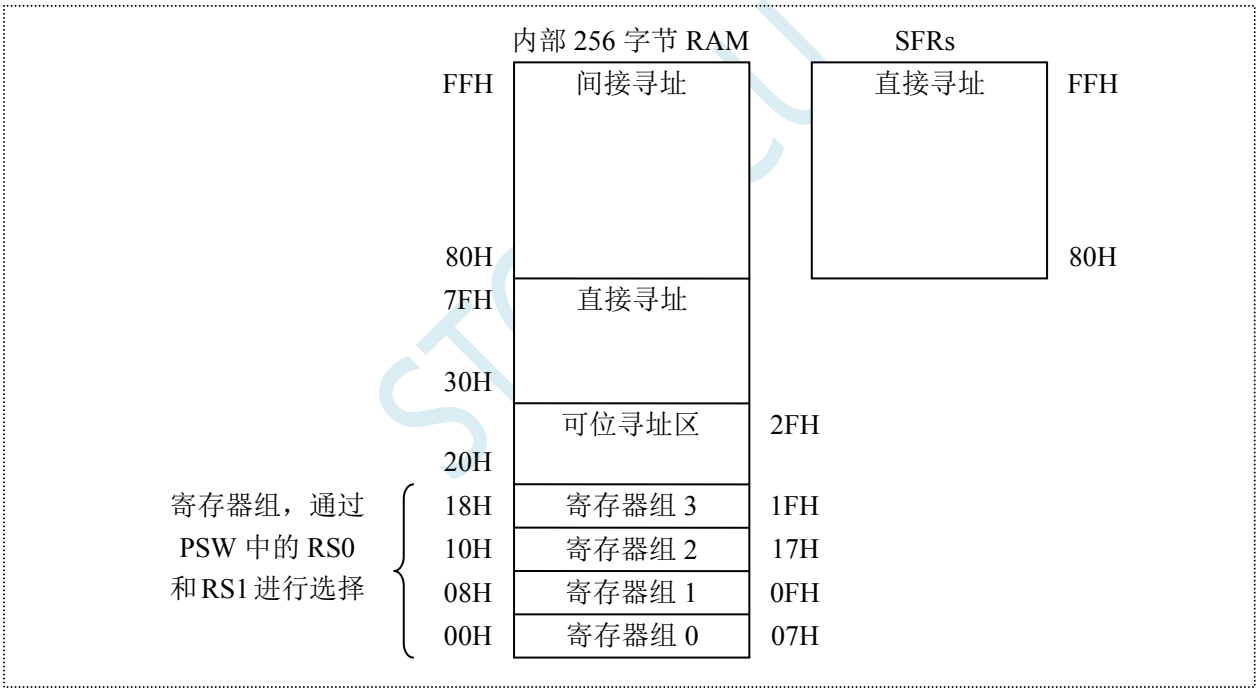
STC8G 系列单片机内部集成的 RAM 可用于存放程序执行的中间结果和过程数据。

单片机系列	内部直接访问 RAM（DATA）	内部直接访问 RAM（IDATA）	内部扩展 RAM（XDATA）
STC8G1K08 系列	128 字节	128 字节	1024 字节

7.2.1 内部RAM

内部 RAM 共 256 字节，可分为 2 个部分：低 128 字节 RAM 和高 128 字节 RAM。低 128 字节的数据存储器与传统 8051 兼容，既可直接寻址也可间接寻址。高 128 字节 RAM（在 8052 中扩展了高 128 字节 RAM）与特殊功能寄存器区共用相同的逻辑地址，都使用 80H~FFH，但在物理上是分别独立的，使用时通过不同的寻址方式加以区分。高 128 字节 RAM 只能间接寻址，特殊功能寄存器区只可直接寻址。

内部 RAM 的结构如下图所示：



低 128 字节 RAM 也称通用 RAM 区。通用 RAM 区又可分为工作寄存器组区，可位寻址区，用户 RAM 区和堆栈区。工作寄存器组区地址从 00H~1FH 共 32 字节单元，分为 4 组，每一组称为一个寄存器组，每组包含 8 个 8 位的工作寄存器，编号均为 R0 ~ R7，但属于不同的物理空间。通过使用工作寄存器组，可以提高运算速度。R0~R7 是常用的寄存器，提供 4 组是因为 1 组往往不够用。程序状态字 PSW 寄存器中的 RS1 和 RS0 组合决定当前使用的工作寄存器组，见下面 PSW 寄存器的介绍。

PSW（程序状态寄存器）

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

RS1，RS0：工作寄存器选择位

RS1	RS0	工作寄存器组（R0~R7）

0	0	第 0 组（00H～07H）
0	1	第 1 组（08H～0FH）
1	0	第 2 组（10H～17H）
1	1	第 3 组（18H～1FH）

可位寻址区的地址从 20H ~ 2FH 共 16 个字节单元。20H~2FH 单元既可像普通 RAM 单元一样按字节存取，也可以对单元中的任何一位单独存取，共 128 位，所对应的逻辑位地址范围是 00H~7FH。位地址范围是 00H~7FH，内部 RAM 低 128 字节的地址也是 00H~7FH，从外表看，二者地址是一样的，实际上二者具有本质的区别；位地址指向的是一个位，而字节地址指向的是一个字节单元，在程序中使用不同的指令区分。

内部 RAM 中的 30H~FFH 单元是用户 RAM 和堆栈区。一个 8 位的堆栈指针(SP)，用于指向堆栈区。单片机复位后，堆栈指针 SP 为 07H，指向了工作寄存器组 0 中的 R7，因此，用户初始化程序都应对 SP 设置初值，一般设置在 80H 以后的单元为宜。

堆栈指针是一个 8 位专用寄存器。它指示出堆栈顶部在内部 RAM 块中的位置。系统复位后，SP 初始化位 07H，使得堆栈事实上由 08H 单元开始，考虑 08H~1FH 单元分别属于工作寄存器组 1~3，若在程序设计中用到这些区，则最好把 SP 值改变为 80H 或更大的值为宜。STC8 系列单片机的堆栈是向上生长的，即将数据压入堆栈后，SP 内容增大。

7.2.2 内部扩展RAM

STC8G 系列单片机片内除了集成 256 字节的内部 RAM 外，还集成了内部的扩展 RAM。访问内部扩展 RAM 的方法和传统 8051 单片机访问外部扩展 RAM 的方法相同，但是不影响 P0 口(数据总线和高八位地址总线)、P2 口(低八位地址总线)、以及 RD、WR 和 ALE 等端口上的信号。

在汇编语言中，内部扩展 RAM 通过 MOVX 指令访问，

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

在 C 语言中，可使用 xdata/pdata 声明存储类型即可。如：

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

注：pdata 即为 xdata 的低 256 字节，在 C 语言中订阅变量为 pdata 类型后，编译器会自动将变量分配在 XDATA 的 0000H~00FFH 区域，并使用 MOVX @Ri,A 和 MOVX A@Ri 进行访问。

单片机内部扩展 RAM 是否可以访问，受辅助寄存器 AUXR 中的 EXTRAM 位控制。

AUXR（辅助寄存器）

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

EXTRAM：扩展 RAM 访问控制

0：访问内部扩展 RAM。

1：内部扩展 RAM 被禁用。

7.3 存储器中的特殊参数

STC8G 系列单片机内部的数据存储器和程序存储器中保存有与芯片相关的一些特殊参数，包括：全球唯一 ID 号、32K 掉电唤醒定时器的频率、内部 Bandgap 电压值以及 IRC 参数。

这些参数在程序存储器（ROM）中的存放地址分别如下：

参数名称	保存地址		参数说明
	STC8G1K08	STC8G1K12	
全球唯一 ID 号	1FF9H~1FFFH	2FF9H~2FFFH	7 字节
Bandgap 电压值	1FF7H~1FF8H	2FF7H~2FF8H	电压单位为毫伏
32K 掉电唤醒定时器的频率	1FF5H~1FF6H	2FF5H~2FF6H	单位 Hz
22.1184MHz 的 IRC 参数	1FF4H	2FF4H	—
24MHz 的 IRC 参数	1FF3H	2FF3H	—

这些参数在数据存储器（RAM）中的存放地址分别如下：

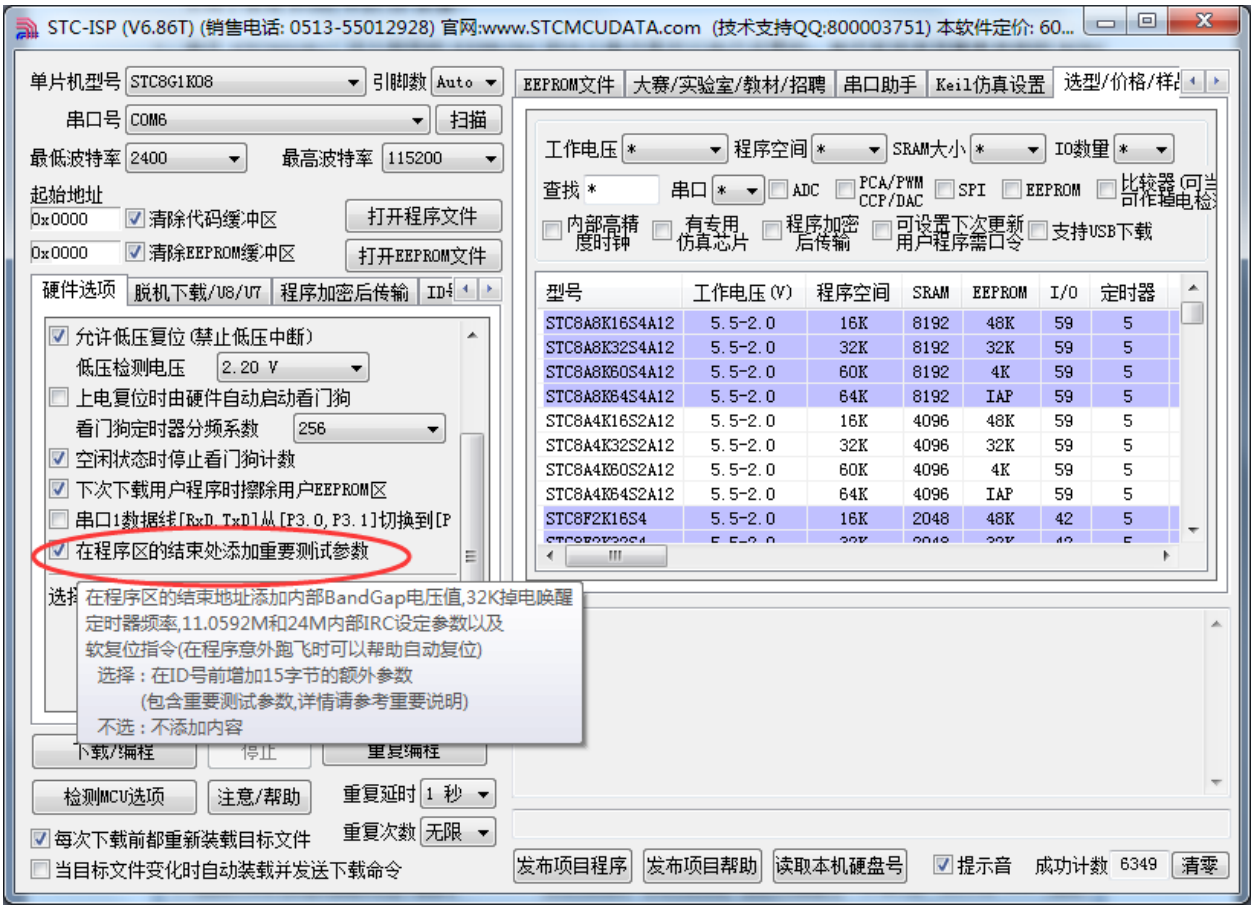
参数名称	保存地址	参数说明
Bandgap 电压值	idata：0EFH~0F0H	电压单位为毫伏，高字节在前
全球唯一 ID 号	idata：0F1H~0F7H	7 字节
32K 掉电唤醒定时器的频率	idata：0F8H~0F9H	单位 Hz，高字节在前
22.1184MHz 的 IRC 参数	idata：0FAH	—
24MHz 的 IRC 参数	idata：0FBH	—

特别说明

1、由于 RAM 中的参数可能被修改，所以一般不建议用户使用，特别是用户使用 ID 号进行加密时，强烈建议用于读取 ROM 中的 ID 数据。

2、由于 STC8G1K12 这个型号的 EEPROM 的大小用户是可以自己设置的，有可能将保存重要参数的 ROM 空间设置为 EEPROM 而人为的将重要参数擦除或修改，所以使用这个型号进行 ID 号进行加密时可能需要考虑这个问题。

3、默认情况下，程序存储器中只有全球唯一 ID 号的数据，而 Bandgap 电压值、32K 掉电唤醒定时器的频率以及 IRC 参数都是没有的，需要在 ISP 下载时选择如下图所示的选项才可用。



7.3.1 读取Bandgap电压值 (从ROM中读取)

汇编代码

```
;测试工作频率为11.0592MHz

AUXR    DATA    8EH
BGV     EQU     01FF7H          ;STC8G1K08
;BGV    EQU     02FF7H          ;STC8G1K12

BUSY    BIT     20H.0

P1M1    DATA    091H
P1M0    DATA    092H
```

```
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

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

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

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

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            LCALL   UART_INIT
            SETB    ES
            SETB    EA

            MOV     DPTR,#BGV
            CLR     A
            MOVC    A,@A+DPTR           ;读取 Bandgap 电压的高字节
            LCALL   UART_SEND
            MOV     A,#1
            MOVC    A,@A+DPTR           ;读取 Bandgap 电压的低字节
            LCALL   UART_SEND
```

```
LOOP:
        JMP     LOOP

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

sfr AUXR = 0x8e;

sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

bit busy;
int *BGV;

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

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

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

void main()
```

```
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    BGV = (int code *)0x1ff7;                    // STC8G1K08
//  BGV = (int code *)0x2ff7;                    // STC8G1K12
    UartInit();
    ES = 1;
    EA = 1;
    UartSend(*BGV >> 8);                         //读取 Bandgap 电压的高字节
    UartSend(*BGV);                              //读取 Bandgap 电压的低字节

    while (1);
}
```

7.3.2 读取Bandgap电压值 (从RAM中读取)

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA    8EH
BGV         DATA    0EFH

BUSY        BIT     20H.0

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

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

UART_INIT:
            MOV     SCON,#50H
            MOV     TMOD,#00H
```

```
        MOV     TL1,#0E8H               ;65536-11059200/115200/4=0FFE8H
        MOV     TH1,#0FFH
        SETB    TR1
        MOV     AUXR,#40H
        CLR     BUSY
        RET

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

MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        LCALL   UART_INIT
        SETB    ES
        SETB    EA

        MOV     R0,#BGV
        MOV     A,@R0                   ;读取 Bandgap 电压的高字节
        LCALL   UART_SEND
        INC     R0
        MOV     A,@R0                   ;读取 Bandgap 电压的低字节
        LCALL   UART_SEND

LOOP:
        JMP     LOOP

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

sfr     AUXR    =   0x8e;

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

bit     busy;
int     *BGV;
```

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

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

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

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    BGV = (int idata *)0xef;
    UartInit();
    ES = 1;
    EA = 1;
    UartSend(*BGV >> 8);                    //读取 Bandgap 电压的高字节
    UartSend(*BGV);                         //读取 Bandgap 电压的低字节

    while (1);
}
```


7.3.3 读取全球唯一ID号 (从ROM中读取)

汇编代码

```
;测试工作频率为11.0592MHz
```

```
AUXR        DATA    8EH
ID          EQU     01FF9H          ; STC8G1K08
;ID         EQU     02FF9H          ; STC8G1K12

BUSY        BIT     20H.0

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

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

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

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

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            LCALL   UART_INIT
            SETB    ES
            SETB    EA
```

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

LOOP:
                JMP         LOOP

                END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

sfr         AUXR        =   0x8e;

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

bit         busy;
char        *ID;

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

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

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

void main()
{
	P1M0 = 0x00;
	P1M1 = 0x00;
	P3M0 = 0x00;
	P3M1 = 0x00;
	P5M0 = 0x00;
	P5M1 = 0x00;

	char i;

	ID = (char code *)0x1ff9;				// STC8G1K08
//	ID = (char code *)0x2ff9;				// STC8G1K12
	UartInit();
	ES = 1;
	EA = 1;

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

	while (1);
}
```

7.3.4 读取全球唯一ID号 (从RAM中读取)

汇编代码

```
;测试工作频率为11.0592MHz

AUXR	DATA	8EH
ID	DATA	0F1H

BUSY	BIT	20H.0

P1M1	DATA	091H
P1M0	DATA	092H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P5M1	DATA	0C9H
P5M0	DATA	0CAH

	ORG	0000H
	LJMP	MAIN
	ORG	0023H
	LJMP	UART_ISR

	ORG	0100H
```

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

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

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

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            LCALL   UART_INIT
            SETB    ES
            SETB    EA

            MOV     R0,#ID
            MOV     R1,#7
NEXT:       MOV     A,@R0
            LCALL   UART_SEND
            INC     R0
            DJNZ    R1,NEXT

LOOP:
            JMP     LOOP

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

#define     FOSC            11059200UL
```

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

sfr       AUXR        =   0x8e;

sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      *ID;

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

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

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

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    char i;

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

```
    EA = 1;

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

    while (1);
}
```

7.3.5 读取 32K掉电唤醒定时器的频率 (从ROM中读取)

汇编代码

```
;测试工作频率为11.0592MHz

AUXR        DATA    8EH
F32K        EQU     01FF5H          ; STC8G1K08
;F32K       EQU     02FF5H          ; STC8G1K12

BUSY        BIT     20H.0

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

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

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

UART_SEND:
            JB      BUSY,$
            SETB    BUSY
```

```
                MOV         SBUF,A
                RET

MAIN:
                MOV         SP, #5FH
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                LCALL       UART_INIT
                SETB        ES
                SETB        EA

                MOV         DPTR,#F32K
                CLR         A
                MOVC        A,@A+DPTR           ;读取 32K 频率的高字节
                LCALL       UART_SEND
                INC         DPTR
                CLR         A
                MOVC        A,@A+DPTR           ;读取 32K 频率的低字节
                LCALL       UART_SEND

LOOP:
                JMP         LOOP

                END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

sfr         AUXR        =   0x8e;

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

bit         busy;
int         *F32K;

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

```
        if (RI)
        {
                RI = 0;
        }
}

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

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

void main()
{
        P1M0 = 0x00;
        P1M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        F32K = (int code *)0x1ff5;                      // STC8G1K08
//      F32K = (int code *)0x2ff5;                      // STC8G1K12
        UartInit();
        ES = 1;
        EA = 1;

        UartSend(*F32K >> 8);                           //读取 32K 频率的高字节
        UartSend(*F32K);                                //读取 32K 频率的低字节

        while (1);
}
```


7.3.6 读取 32K掉电唤醒定时器的频率 (从RAM中读取)

汇编代码

```
;测试工作频率为11.0592MHz

AUXR        DATA        8EH
F32K        DATA        0F8H

BUSY        BIT         20H.0

P1M1        DATA        091H
P1M0        DATA        092H
P3M1        DATA        0B1H
```

```
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

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

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

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

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            LCALL   UART_INIT
            SETB    ES
            SETB    EA

            MOV     R0,#F32K
            MOV     A,@R0               ;读取 32K 频率的高字节
            LCALL   UART_SEND
            INC     R0
            MOV     A,@R0               ;读取 32K 频率的低字节
            LCALL   UART_SEND

LOOP:
            JMP     LOOP
```

```
END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

sfr        AUXR        =    0x8e;

sfr        P1M1        =    0x91;
sfr        P1M0        =    0x92;
sfr        P3M1        =    0xb1;
sfr        P3M0        =    0xb2;
sfr        P5M1        =    0xc9;
sfr        P5M0        =    0xca;

bit        busy;
int        *F32K;

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

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

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

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

```
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    F32K = (int idata *)0xf8;
    UartInit();
    ES = 1;
    EA = 1;

    UartSend(*F32K >> 8);                    //读取 32K 频率的高字节
    UartSend(*F32K);                         //读取 32K 频率的低字节

    while (1);
}
```

7.3.7 用户自定义内部IRC频率 (从ROM中读取)

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH
CKSEL       EQU     0FE00H
CLKDIV      EQU     0FE01H

IRCCR       DATA    09FH

IRC22M      EQU     0FDF4H          ; STC8A8K64S4A10
IRC24M      EQU     0FDF3H
;IRC22M     EQU     0EFF4H          ; STC8A8K60S4A10
;IRC24M     EQU     0EFF3H
;IRC22M     EQU     07FF4H          ; STC8A8K32S4A10
;IRC24M     EQU     07FF3H
;IRC22M     EQU     03FF4H          ; STC8A8K16S4A10
;IRC24M     EQU     03FF3H

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

;           MOV     DPTR,#IRC22M        ;装载 22.1184MHz 的 IRC 参数
```

```
;               CLR         A
;               MOVC        A,@A+DPTR
;               MOV         IRCCR,A
                MOV         DPTR,#IRC24M            ;装载 24MHz 的 IRC 参数
                CLR         A
                MOVC        A,@A+DPTR
                MOV         IRCCR,A

                MOV         P_SW2,#80H
                MOV         A,#0                    ;主时钟不预分频
                MOV         DPTR,#CLKDIV
                MOVX        @DPTR,A
                MOV         A,#40H                  ;主时钟 4 分频输出到 P5.4 口
                MOV         DPTR,#CKSEL
                MOVX        @DPTR,A
                MOV         P_SW2,#00H

                JMP         $

                END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

sfr         P_SW2       =   0xba;
sfr         IRCCR       =   0x9f;

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

char        *IRC22M;
char        *IRC24M;

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IRC22M = (char code *)0xfdf4;               // STC8A8K64S4A10
    IRC24M = (char code *) 0xfdf3;
//  IRC22M = (char code *)0xeff4;               // STC8A8K60S4A10
//  IRC24M = (char code *) 0xeff3;
//  IRC22M = (char code *)0x7ff4;               // STC8A8K32S4A10
//  IRC24M = (char code *) 0x7ff3;
```

```
//      IRC22M = (char code *)0x3ff4;                   // STC8A8K16S4A10
//      IRC24M = (char code *) 0x3ff3;

//      IRCCR = *IRC22M;                                //装载 22.1184MHz 的 IRC 参数
        IRCCR = *IRC24M;                                //装载 24MHz 的 IRC 参数

        P_SW2 = 0x80;
        CLKDIV = 0;                                     //主时钟不预分频
        CKSEL = 0x40;                                   //主时钟 4 分频输出到 P5.4 口
        P_SW2 = 0x00;

        while (1);
}
```

7.3.8 用户自定义内部IRC频率 (从RAM中读取)

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH
CKSEL       EQU     0FE00H
CLKDIV      EQU     0FE01H

IRCCR       DATA    09FH

IRC22M      DATA    0FAH
IRC24M      DATA    0FBH

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

;           MOV     R0,#IRC22M                      ;装载 22.1184MHz 的 IRC 参数
;           MOV     IRCCR,@R0
            MOV     R0,#IRC24M                      ;装载 24MHz 的 IRC 参数
            MOV     IRCCR,@R0

            MOV     P_SW2,#80H
            MOV     A,#0                            ;主时钟不预分频
            MOV     DPTR,#CLKDIV
            MOVX    @DPTR,A
            MOV     A,#40H                          ;主时钟 4 分频输出到 P5.4 口
```

```
        MOV     DPTR,#CKSEL
        MOVX    @DPTR,A
        MOV     P_SW2,#00H

        JMP     $

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

sfr       P_SW2     =   0xba;
sfr       IRCCR     =   0x9f;

sfr       P1M1      =   0x91;
sfr       P1M0      =   0x92;
sfr       P3M1      =   0xb1;
sfr       P3M0      =   0xb2;
sfr       P5M1      =   0xc9;
sfr       P5M0      =   0xca;

char      *IRC22M;
char      *IRC24M;

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IRC22M = (char idata *)0xfa;
    IRC24M = (char idata *) 0xfb;
//  IRCCR = *IRC22M;                    //装载22.1184MHz的IRC参数
    IRCCR = *IRC24M;                    //装载24MHz的IRC参数

    P_SW2 = 0x80;
    CLKDIV = 0;                         //主时钟不预分频
    CKSEL = 0x40;                       //主时钟4分频输出到P5.4口
    P_SW2 = 0x00;

    while (1);
}
```

8 特殊功能寄存器

8.1 STC8G1K08 系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H		CH	CCAP0H	CCAP1H	CCAP2H			RSTCFG
F0H	B		PCA_PWM0	PCA_PWM1	PCA_PWM2	IAP_TPS		
E8H		CL	CCAP0L	CCAP1L	CCAP2L			AUXINTIF
E0H	ACC			DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2		ADCCFG	
D0H	PSW						T2H	T2L
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H		WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2		ADC_CONTR	ADC_RES	ADC_RESL	
B0H	P3	P3M1	P3M0			IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH			TA	IE2
A0H			P_SW1					
98H	SCON	SBUF	S2CON	S2BUF		IRCBAND	LIRTRIM	IRTRIM
90H	P1	P1M1	P1M0					
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H		SP	DPL	DPH				PCON

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FEA8H	ADCTIM							
FEA0H			TM2PS					
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE30H		P1IE		P3IE				
FE28H		P1DR		P3DR		P5DR		
FE20H		P1SR		P3SR		P5SR		
FE18H		P1NCS		P3NCS		P5NCS		
FE10H		P1PU		P3PU		P5PU		
FE00H	CKSEL	CLKDIV	IRC24MCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDB	

8.2 特殊功能寄存器列表

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
SP	堆栈指针	81H									0000,0111
DPL	数据指针（低字节）	82H									0000,0000
DPH	数据指针（高字节）	83H									0000,0000
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
TMOD	定时器模式寄存器	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000,0000
TL0	定时器 0 低 8 为寄存器	8AH									0000,0000
TL1	定时器 1 低 8 为寄存器	8BH									0000,0000
TH0	定时器 0 高 8 为寄存器	8CH									0000,0000
TH1	定时器 1 高 8 为寄存器	8DH									0000,0000
AUXR	辅助寄存器 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
INTCLKO	中断与时钟输出控制寄存器	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
P1	P1 端口	90H									1111,1111
P1M1	P1 口配置寄存器 1	91H									1111,1111
P1M0	P1 口配置寄存器 0	92H									0000,0000
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	串口 1 数据寄存器	99H									0000,0000
S2CON	串口 2 控制寄存器	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0x00,0000
S2BUF	串口 2 数据寄存器	9BH									0000,0000
IRCBAND	IRC 频段选择检测	9DH	-	-	-	-	-	-	-	SEL	xxxx,xxxn
LIRTRIM	IRC 频率微调寄存器	9EH	-	-	-	-	-	-	LIRTRIM[1:0]		xxxx,xxnn
IRTRIM	IRC 频率调整寄存器	9FH	IRTRIM[7:0]								nnnn,nnnn
P_SW1	外设端口切换寄存器 1	A2H	S1_S[1:0]		CCP_S[1:0]		SPI_S[1:0]		0	-	nn00,000x
IE	中断允许寄存器	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000,0000
SADDR	串口 1 从机地址寄存器	A9H									0000,0000
WKTCL	掉电唤醒定时器低字节	AAH									1111,1111
WKTCH	掉电唤醒定时器高字节	ABH	WKTEN								0111,1111
TA	DPTR 时序控制寄存器	AEH									0000,0000
IE2	中断允许寄存器 2	AFH	-	-	-	-	-	ET2	ESPI	ES2	x000,0000
P3	P3 端口	B0H									1111,1111
P3M1	P3 口配置寄存器 1	B1H									1111,1100
P3M0	P3 口配置寄存器 0	B2H									0000,0000
IP2	中断优先级控制寄存器 2	B5H	-	PI2C	PCMP	PX4	-	-	PSPI	PS2	x000,xx00
IP2H	高中断优先级控制寄存器 2	B6H	-	PI2CH	PCMPH	PX4H	-	-	PSPIH	PS2H	x000,xx00
IPH	高中断优先级控制寄存器	B7H	PPCAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	0000,0000
IP	中断优先级控制寄存器	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000,0000
SADEN	串口 1 从机地址屏蔽寄存器	B9H									0000,0000
P_SW2	外设端口切换寄存器 2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	-	-	S2_S	0x00,0xx0
ADC_CONTR	ADC 控制寄存器	BCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]				000x,0000

ADC_RES	ADC 转换结果高位寄存器	BDH									0000,0000
ADC_RESL	ADC 转换结果低位寄存器	BEH									0000,0000
WDT_CONTR	看门狗控制寄存器	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]			0xn0,nnnn
IAP_DATA	IAP 数据寄存器	C2H									1111,1111
IAP_ADDRH	IAP 高地址寄存器	C3H									0000,0000
IAP_ADDRL	IAP 低地址寄存器	C4H									0000,0000
IAP_CMD	IAP 命令寄存器	C5H	-	-	-	-	-	-	CMD[1:0]		xxxx,xx00
IAP_TRIG	IAP 触发寄存器	C6H									0000,0000
IAP_CONTR	IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-	0000,xxxx
P5	P5 端口	C8H	-	-			-	-	-	-	xx11,xxxx
P5M1	P5 口配置寄存器 1	C9H	-	-			-	-	-	-	xx11,xxxx
P5M0	P5 口配置寄存器 0	CAH	-	-			-	-	-	-	xx00,xxxx
SPSTAT	SPI 状态寄存器	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
SPCTL	SPI 控制寄存器	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]		0000,0100
SPDAT	SPI 数据寄存器	CFH									0000,0000
PSW	程序状态字寄存器	D0H	CY	AC	F0	RS1	RS0	OV	-	P	0000,00x0
T2H	定时器 2 高字节	D6H									0000,0000
T2L	定时器 2 低字节	D7H									0000,0000
CCON	PCA 控制寄存器	D8H	CF	CR	-	-	-	CCF2	CCF1	CCF0	00xx,x000
CMOD	PCA 模式寄存器	D9H	CIDL	-	-	-	CPS[2:0]			ECF	0xxx,0000
CCAPM0	PCA 模块 0 模式控制寄存器	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAPM1	PCA 模块 1 模式控制寄存器	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CCAPM2	PCA 模块 2 模式控制寄存器	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
ADCCFG	ADC 配置寄存器	DEH	-	-	RESFMT	-	SPEED[3:0]				xx0x,0000
ACC	累加器	E0H									0000,0000
DPS	DPTR 指针选择器	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL	0000,0xx0
DPL1	第二组数据指针（低字节）	E4H									0000,0000
DPH1	第二组数据指针（高字节）	E5H									0000,0000
CMPCR1	比较器控制寄存器 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
CMPCR2	比较器控制寄存器 2	E7H	INVCMPO	DISFLT	LCDTY[5:0]						0000,0000
CL	PCA 计数器低字节	E9H									0000,0000
CCAP0L	PCA 模块 0 低字节	EAH									0000,0000
CCAP1L	PCA 模块 1 低字节	EBH									0000,0000
CCAP2L	PCA 模块 2 低字节	ECH									0000,0000
AUXINTIF	扩展外部中断标志寄存器	EFH	-	INT4IF	INT3IF	INT2IF	-	-	-	T2IF	x000,xxx0
B	B 寄存器	F0H									0000,0000
PCA_PWM0	PCA0 的 PWM 模式寄存器	F2H	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L	0000,0000
PCA_PWM1	PCA1 的 PWM 模式寄存器	F3H	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L	0000,0000
PCA_PWM2	PCA2 的 PWM 模式寄存器	F4H	EBS2[1:0]		XCCAP2H[1:0]		XCCAP2L[1:0]		EPC2H	EPC2L	0000,0000
IAP_TPS	IAP 等待时间控制寄存器	F5H	-	-	IAPTPS[5:0]						xx00,0000
CH	PCA 计数器高字节	F9H									0000,0000
CCAP0H	PCA 模块 0 高字节	FAH									0000,0000
CCAP1H	PCA 模块 1 高字节	FBH									0000,0000
CCAP2H	PCA 模块 2 高字节	FCH									0000,0000

RSTCFG	复位配置寄存器	FFH	-	ENLVR	-	P54RST	-	-	LVDS[1:0]	xnxn,xxnn

下列特殊功能寄存器为扩展 SFR，逻辑地址位于 XDATA 区域，访问前需要将 P_SW2（BAH）寄存器的最高位（EAXFR）置 1，然后使用 MOVX A,@DPTR 和 MOVX @DPTR,A 指令进行访问

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CKSEL	时钟选择寄存器	FE00H	-	-	-	-	-	-	MCKSEL[1:0]		xxxx,xx00
CLKDIV	时钟分频寄存器	FE01H									0000,0100
IRC24MCR	内部 24M 振荡器控制寄存器	FE02H	ENIRC24M	-	-	-	-	-	-	IRC24MST	1xxx,xxx0
XOSCCR	外部晶振控制寄存器	FE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST	00xx,xxx0
IRC32KCR	内部 32K 振荡器控制寄存器	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0
MCLKOCR	主时钟输出控制寄存器	FE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000
IRCDB	内部高速振荡器去抖控制	FE06H	IRCDB_PAR[7:0]								1000,0000
P1PU	P1 口上拉电阻控制寄存器	FE11H									0000,0000
P3PU	P3 口上拉电阻控制寄存器	FE13H									0000,0000
P5PU	P5 口上拉电阻控制寄存器	FE15H	-	-			-	-	-	-	xx00,xxxx
P1NCS	P1 口施密特触发控制寄存器	FE19H									0000,0000
P3NCS	P3 口施密特触发控制寄存器	FE1BH									0000,0000
P5NCS	P5 口施密特触发控制寄存器	FE1DH	-	-			-	-	-	-	xx00,xxxx
P1SR	P1 口电平转换速率寄存器	FE21H									0000,0000
P3SR	P3 口电平转换速率寄存器	FE23H									0000,0000
P5SR	P5 口电平转换速率寄存器	FE25H	-	-			-	-	-	-	xx00,xxxx
P1DR	P1 口驱动电流控制寄存器	FE29H									1111,1111
P3DR	P3 口驱动电流控制寄存器	FE2BH									1111,1111
P5DR	P5 口驱动电流控制寄存器	FE2DH	-	-			-	-	-	-	xx11,xxxx
P1IE	P1 口输入使能控制寄存器	FE31H									1111,1111
P3IE	P3 口输入使能控制寄存器	FE33H									1111,1111
I2CCFG	I^2C 配置寄存器	FE80H	ENI2C	MSSL	MSSPEED[6:1]						0000,0000
I2CMSCR	I^2C 主机控制寄存器	FE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I^2C 主机状态寄存器	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I^2C 从机控制寄存器	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I^2C 从机状态寄存器	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I^2C 从机地址寄存器	FE85H	SLADR[6:0]							MA	0000,0000
I2CTXD	I^2C 数据发送寄存器	FE86H									0000,0000
I2CRXD	I^2C 数据接收寄存器	FE87H									0000,0000
I2CMSAUX	I^2C 主机辅助控制寄存器	FE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0
TM2PS	定时器 2 时钟预分频寄存器	FEA2H									0000,0000
ADCTIM	ADC 时序控制寄存器	FEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]					0010,1010

9 I/O口

STC8G 系列单片机最多有 18 个 I/O 口。所有的 I/O 口均有 4 种工作模式：准双向口/弱上拉（标准 8051 输出口模式）、推挽输出/强上拉、高阻输入（电流既不能流入也不能流出）、开漏输出。可使用软件对 I/O 口的工作模式进行容易配置。

注意：除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口时必须先设置 I/O 口模式

9.1 I/O口相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P1	P1 端口	90H									1111,1111
P3	P3 端口	B0H									1111,1111
P5	P5 端口	C8H	-	-			-				xx11,xxxx
P1M1	P1 口配置寄存器 1	91H									0000,0000
P1M0	P1 口配置寄存器 0	92H									0000,0000
P3M1	P3 口配置寄存器 1	B1H									n000,0000
P3M0	P3 口配置寄存器 0	B2H									n000,0000
P5M1	P5 口配置寄存器 1	C9H	-	-			-	-	-	-	xx11,xxxx
P5M0	P5 口配置寄存器 0	CAH	-	-			-	-	-	-	xx00,xxxx

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P1PU	P1 口上拉电阻控制寄存器	FE11H									0000,0000
P3PU	P3 口上拉电阻控制寄存器	FE13H									0000,0000
P5PU	P5 口上拉电阻控制寄存器	FE15H	-	-			-	-	-	-	xx00,xxxx
P1NCS	P1 口施密特触发控制寄存器	FE19H									0000,0000
P3NCS	P3 口施密特触发控制寄存器	FE1BH									0000,0000
P5NCS	P5 口施密特触发控制寄存器	FE1DH	-	-			-	-	-	-	xx00,xxxx
P1SR	P1 口电平转换速率寄存器	FE21H									0000,0000
P3SR	P3 口电平转换速率寄存器	FE23H									0000,0000
P5SR	P5 口电平转换速率寄存器	FE25H	-	-			-	-	-	-	xx00,xxxx
P1DR	P1 口驱动电流控制寄存器	FE29H									1111,1111
P3DR	P3 口驱动电流控制寄存器	FE2BH									1111,1111
P5DR	P5 口驱动电流控制寄存器	FE2DH	-	-			-	-	-	-	xx11,xxxx
P1IE	P1 口输入使能控制寄存器	FE31H									1111,1111
P3IE	P3 口输入使能控制寄存器	FE33H									1111,1111

端口数据寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P1	90H	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
P3	B0H	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0

P5	C8H	-	-	P5.5	P5.4	-	-	-	-

读写端口状态

写 0：输出低电平到端口缓冲区

写 1：输出高电平到端口缓冲区

读：直接读端口管脚上的电平

端口模式配置寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P1M0	92H								
P1M1	91H								
P3M0	B2H								
P3M1	B1H								
P5M0	CAH	-	-			-	-	-	-
P5M1	C9H	-	-			-	-	-	-

配置端口的模式

PnM1.x	PnM0.x	Pn.x 口工作模式
0	0	准双向口
0	1	推挽输出
1	0	高阻输入
1	1	开漏输出

端口上拉电阻控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P1PU	FE11H								
P3PU	FE13H								
P5PU	FE15H	-	-			-	-	-	-

端口内部3.7K上拉电阻控制位（注：P3.0和P3.1口上的上拉电阻可能会略小一些）

0：禁止端口内部的 3.7K 上拉电阻（实测为 4.2K 左右）

1：使能端口内部的 3.7K 上拉电阻（实测为 4.2K 左右）

端口施密特触发控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P1NCS	FE19H								
P3NCS	FE1BH								
P5NCS	FE1DH	-	-			-	-	-	-

端口施密特触发控制位

0：使能端口的施密特触发功能。（上电复位后默认使能施密特触发）

1：禁止端口的施密特触发功能。

端口电平转换速度控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P1SR	FE21H								
P3SR	FE23H								
P5SR	FE25H	-	-			-	-	-	-

控制端口电平转换的速度

0：电平转换速度快，相应的上下冲会比较大

1：电平转换速度慢，相应的上下冲比较小

端口驱动电流控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P1DR	FE29H								
P3DR	FE2BH								
P5DR	FE2DH	-	-			-	-	-	-

控制端口的驱动能力

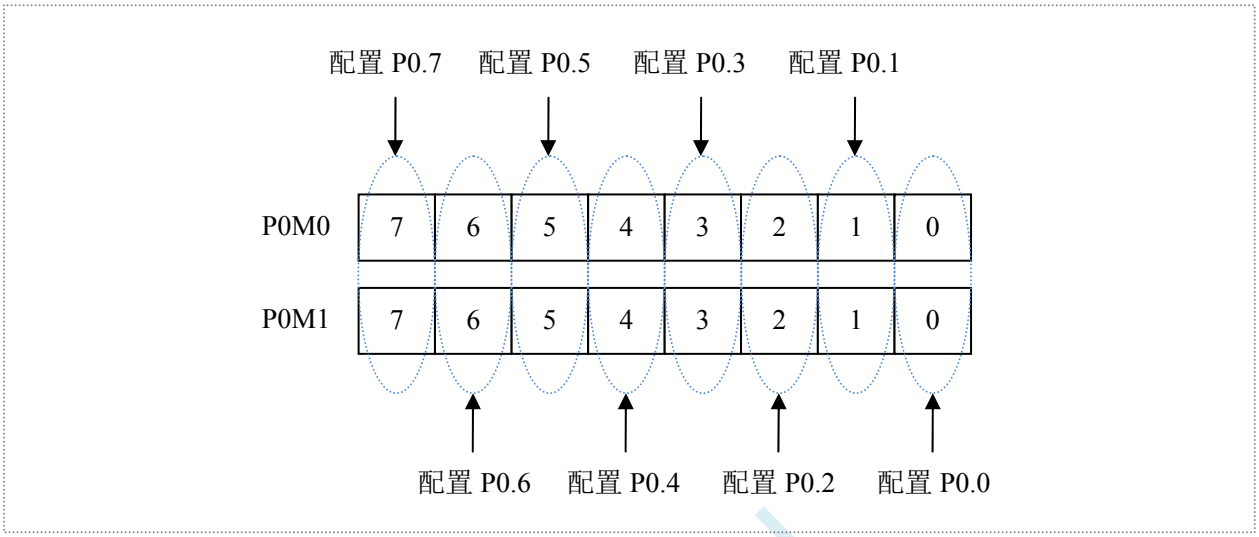
0：一般驱动能力

1：增强驱动能力

9.2 配置I/O口

每个 I/O 的配置都需要使用两个寄存器进行设置。

以 P0 口为例，配置 P0 口需要使用 P0M0 和 P0M1 两个寄存器进行配置，如下图所示：



即 P0M0 的第 0 位和 P0M1 的第 0 位组合起来配置 P0.0 口的模式
即 P0M0 的第 1 位和 P0M1 的第 1 位组合起来配置 P0.1 口的模式
其他所有 I/O 的配置都与此类似。

PnM0 与 PnM1 的组合方式如下表所示

PnM1	PnM0	I/O 口工作模式
0	0	准双向口（传统8051端口模式，弱上拉） 灌电流可达20mA，拉电流为270～150μA（存在制造误差）
0	1	推挽输出（强上拉输出，可达20mA，要加限流电阻）
1	0	高阻输入（电流既不能流入也不能流出）
1	1	开漏输出（Open-Drain），内部上拉电阻断开 开漏模式既可读外部状态也可对外输出（高电平或低电平）。如要正确读外部状态或需要对外输出高电平，需外加上拉电阻，否则读不到外部状态，也对外输不出高电平。

注：n = 0, 1, 2, 3, 4, 5, 6, 7

注意：

虽然每个 I/O 口在弱上拉（准双向口）/强推挽输出/开漏模式时都能承受 20mA 的灌电流（还是要加限流电阻，如 1K、560Ω、472Ω 等），在强推挽输出时能输出 20mA 的拉电流（也要加限流电阻），但整个芯片的工作电流推荐不要超过 90mA，即从 VCC 流入的电流建议不要超过 90mA，从 GND 流出电流建议不要超过 90mA，整体流入/流出电流建议都不要超过 90mA。

9.3 I/O的结构图

9.3.1 准双向口（弱上拉）

准双向口（弱上拉）输出类型可用作输出和输入功能而不需重新配置端口输出状态。这是因为当端口输出为 1 时驱动能力很弱，允许外部装置将其拉低。当引脚输出为低时，它的驱动能力很强，可吸收相当大的电流。准双向口有 3 个上拉晶体管适应不同的需要。

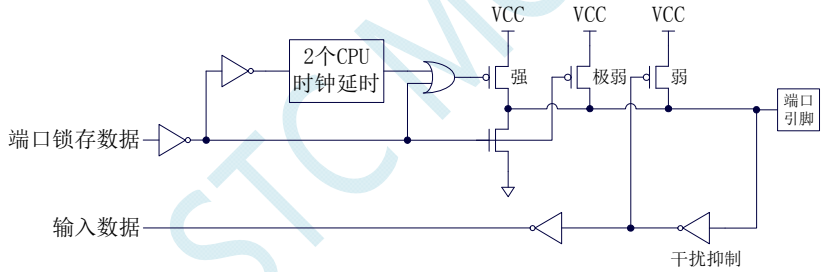
在 3 个上拉晶体管中，有 1 个上拉晶体管称为“弱上拉”，当端口寄存器为 1 且引脚本身也为 1 时打开。此上拉提供基本驱动电流使准双向口输出为 1。如果一个引脚输出为 1 而由外部装置下拉到低时，弱上拉关闭而“极弱上拉”维持开状态，为了把这个引脚强拉为低，外部装置必须有足够的灌电流能力使引脚上的电压降到门槛电压以下。对于 5V 单片机，“弱上拉”晶体管的电流约 250uA；对于 3.3V 单片机，“弱上拉”晶体管的电流约 150uA。

第 2 个上拉晶体管，称为“极弱上拉”，当端口锁存为 1 时打开。当引脚悬空时，这个极弱的上拉源产生很弱的上拉电流将引脚上拉为高电平。对于 5V 单片机，“极弱上拉”晶体管的电流约 18uA；对于 3.3V 单片机，“极弱上拉”晶体管的电流约 5uA。

第 3 个上拉晶体管称为“强上拉”。当端口锁存器由 0 到 1 跳变时，这个上拉用来加快准双向口由逻辑 0 到逻辑 1 转换。当发生这种情况时，强上拉打开约 2 个时钟以使引脚能够迅速地上拉到高电平。

准双向口（弱上拉）带有一个施密特触发输入以及一个干扰抑制电路。准双向口（弱上拉）读外部状态前,要先锁存为 ‘1’,才可读到外部正确的状态.

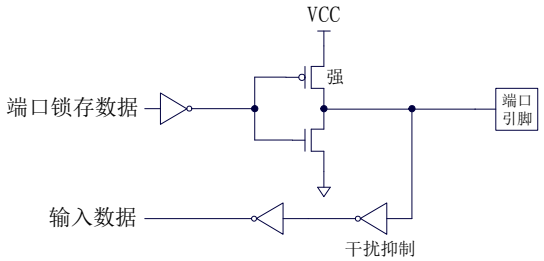
准双向口（弱上拉）输出如下图所示：



9.3.2 推挽输出

强推挽输出配置的下拉结构与开漏输出以及准双向口的下拉结构相同，但当锁存器为 1 时提供持续的强上拉。推挽模式一般用于需要更大驱动电流的情况。

强推挽引脚配置如下图所示：

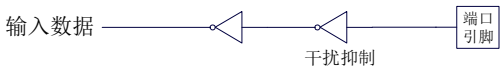


9.3.3 高阻输入

电流既不能流入也不能流出

输入口带有一个施密特触发输入以及一个干扰抑制电路

高阻输入引脚配置如下图所示：



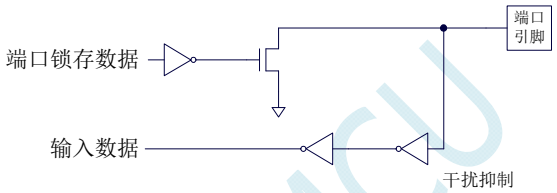
9.3.4 开漏输出

开漏模式既可读外部状态也可对外输出（高电平或低电平）。如要正确读外部状态或需要对外输出高电平，需外加上拉电阻。

当端口锁存器为 0 时，开漏输出关闭所有上拉晶体管。当作为一个逻辑输出高电平时，这种配置方式必须有外部上拉，一般通过电阻外接到 VCC。如果外部有上拉电阻，开漏的 I/O 口还可读外部状态，即此时被配置为开漏模式的 I/O 口还可作为输入 I/O 口。这种方式的下拉与准双向口相同。

开漏端口带有一个施密特触发输入以及一个干扰抑制电路。

输出端口配置如下图所示：



9.4 范例程序

9.4.1 端口模式设置

汇编代码

```
;测试工作频率为11.0592MHz

P0M0        DATA    094H
P0M1        DATA    093H
P1M0        DATA    092H
P1M1        DATA    091H
P2M0        DATA    096H
P2M1        DATA    095H
P3M0        DATA    0B2H
P3M1        DATA    0B1H
P4M0        DATA    0B4H
P4M1        DATA    0B3H
P5M0        DATA    0CAH
P5M1        DATA    0C9H
P6M0        DATA    0CCH
P6M1        DATA    0CBH
P7M0        DATA    0E2H
P7M1        DATA    0E1H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH

            MOV     P0M0,#00H           ;设置P0.0~P0.7为双向口模式
            MOV     P0M1,#00H
            MOV     P1M0,#0FFH          ;设置P1.0~P1.7为推挽输出模式
            MOV     P1M1,#00H
            MOV     P2M0,#00H           ;设置P2.0~P2.7为高阻输入模式
            MOV     P2M1,#0FFH
            MOV     P3M0,#0FFH          ;设置P3.0~P3.7为开漏模式
            MOV     P3M1,#0FFH

            JMP     $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

```
sfr     P3M0    =   0xb2;
sfr     P3M1    =   0xb1;
sfr     P4M0    =   0xb4;
sfr     P4M1    =   0xb3;
sfr     P5M0    =   0xca;
sfr     P5M1    =   0xc9;
sfr     P6M0    =   0xcc;
sfr     P6M1    =   0xcb;
sfr     P7M0    =   0xe2;
sfr     P7M1    =   0xe1;

void main()
{
    P0M0 = 0x00;                    //设置 P0.0~P0.7 为双向口模式
    P0M1 = 0x00;
    P1M0 = 0xff;                    //设置 P1.0~P1.7 为推挽输出模式
    P1M1 = 0x00;
    P2M0 = 0x00;                    //设置 P2.0~P2.7 为高阻输入模式
    P2M1 = 0xff;
    P3M0 = 0xff;                    //设置 P3.0~P3.7 为开漏模式
    P3M1 = 0xff;

    while (1);
}
```

9.4.2 双向口读写操作

汇编代码

```
;测试工作频率为 11.0592MHz

P0M0        DATA    094H
P0M1        DATA    093H
P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P0M0,#00H       ;设置 P0.0~P0.7 为双向口模式
            MOV     P0M1,#00H

            SETB    P0.0            ;P0.0 口输出高电平
            CLR     P0.0            ;P0.0 口输出低电平
```

```
        SETB    P0.0            ;读取端口前先使能内部弱上拉电阻
        NOP                     ;等待两个时钟
        NOP
        MOV     C,P0.0          ;读取端口状态

        JMP     $

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     P0M0    =   0x94;
sfr     P0M1    =   0x93;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;
sbit    P00     =   P0^0;

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P0M0 = 0x00;                //设置 P0.0~P0.7 为双向口模式
    P0M1 = 0x00;

    P00 = 1;                    //P0.0 口输出高电平
    P00 = 0;                    //P0.0 口输出低电平

    P00 = 1;                    //读取端口前先使能内部弱上拉电阻
    _nop_();                    //等待两个时钟
    _nop_();                    //
    CY = P00;                   //读取端口状态

    while (1);
}
```

9.4.3 用STC系列MCU的I/O口直接驱动段码LCD

当产品需要段码 LCD 显示时，如果使用不带 LCD 驱动器的 MCU，则需要外接 LCD 驱动 IC，这会增加成本和 PCB 面积。事实上，很多小项目，比如大量的小家电，需要显示的段码不多，常见的是 4 个 8 带小数点或时钟的冒号“：”，这样如果使用 I/O 口直接扫描显示，则会减小 PCB 面积，降低成本。

但是，本方案不合适驱动太多的段（ 占用 I/O 太多），也不合适非常低功耗的场合。

段码 LCD 驱动简单原理：如图 1 所示。

LCD 是一种特殊的液态晶体，在电场的作用下晶体的排列方向会发生扭转，因而改变其透光性，从而可以看到显示内容。LCD 有一个扭转阀值，当 LCD 两端电压高于此阀值时，显示内容，低于此阀值时，不显示。通常 LCD 有 3 个参数：工作电压、DUTY（对应 COM 数）和 BIAS（即偏压，对应阀值），比如 4.5V、1/4 DUTY、1/3 BIAS，表示 LCD 显示电压为 4.5V，4 个 COM，阀值大约是 1.5V，当加在某段 LCD 两端电压大于 1.5V 时（一般加 4.5V）显示，而加 1.5V 时不显示。但是 LCD 对于驱动电压的反应不是很明显的，比如加 2V 时，可能会微弱显示，这就是通常说的“鬼影”。所以要保证驱动显示时，要大于阀值电压比较多，而不显示时，要用比阀值小比较多的电压。

注意：LCD 的两端不能加直流电压，否则时间稍长就会损坏，所以要保证加在 LCD 两端的驱动电压的平均电压为 0。LCD 使用时分割扫描法，任何时候一个 COM 扫描有效，另外的 COM 处于无效状态。

驱动 1/4Duty 1/2BIAS 3V 的方案电路见图 1，LCD 扫描原理见图 3，MCU 为 3V 工作，用双向口做 COM，PUSH-PULL 或 STANDARD 输出口接 SEG，并且每个 COM 都接一个 47K 电阻到一个电容，RC 滤波后得到一个中点电压。在轮到某个 COM 扫描时，设置成 PUSH-PULL 输出，如果与本 COM 连接的 SEG 不显示，则 SEG 输出与 COM 同相，如果显示，则反相。扫描完后，这个 COM 的 I/O 就设置成高阻，这样这个 COM 就通过 47K 电阻连接到 1/2VDD 电压，而 SEG 继续输出方波，这样加在 LCD 上的电压，显示时是+-VDD，不显示时是+-1/2VDD，保证了 LCD 两端平均直流电压为 0。

驱动 1/4Duty 1/3BIAS 3V 的方案电路见图 4，LCD 扫描原理见图 5，，MCU 为 5V 工作，SEG 线通过电阻分压输出 1.5V、3.5V，COM 线通过电阻分压输出 0.5V、2.5V（高阻时）、4.5V。在轮到某个 COM 扫描时，设置成 PUSH-PULL 输出，如果与本 COM 连接的 SEG 不显示，则 SEG 输出与 COM 同相，如果显示，则反相。扫描完后，这个 COM 的 I/O 就设置成高阻，这样这个 COM 就通过 47K 电阻连接到 2.5V 电压，而 SEG 继续输出方波，这样加在 LCD 上的电压，显示时是+-3.0V，不显示时是+-1.0V，完全满足 LCD 的扫描要求。

当需要睡眠省电时，把所有 COM 和 SEG 驱动 I/O 全部输出低电平，LCD 驱动部分不会增加额外电流。

图 1：驱动 1/4Duty 1/2BIAS 3V LCD 的电路

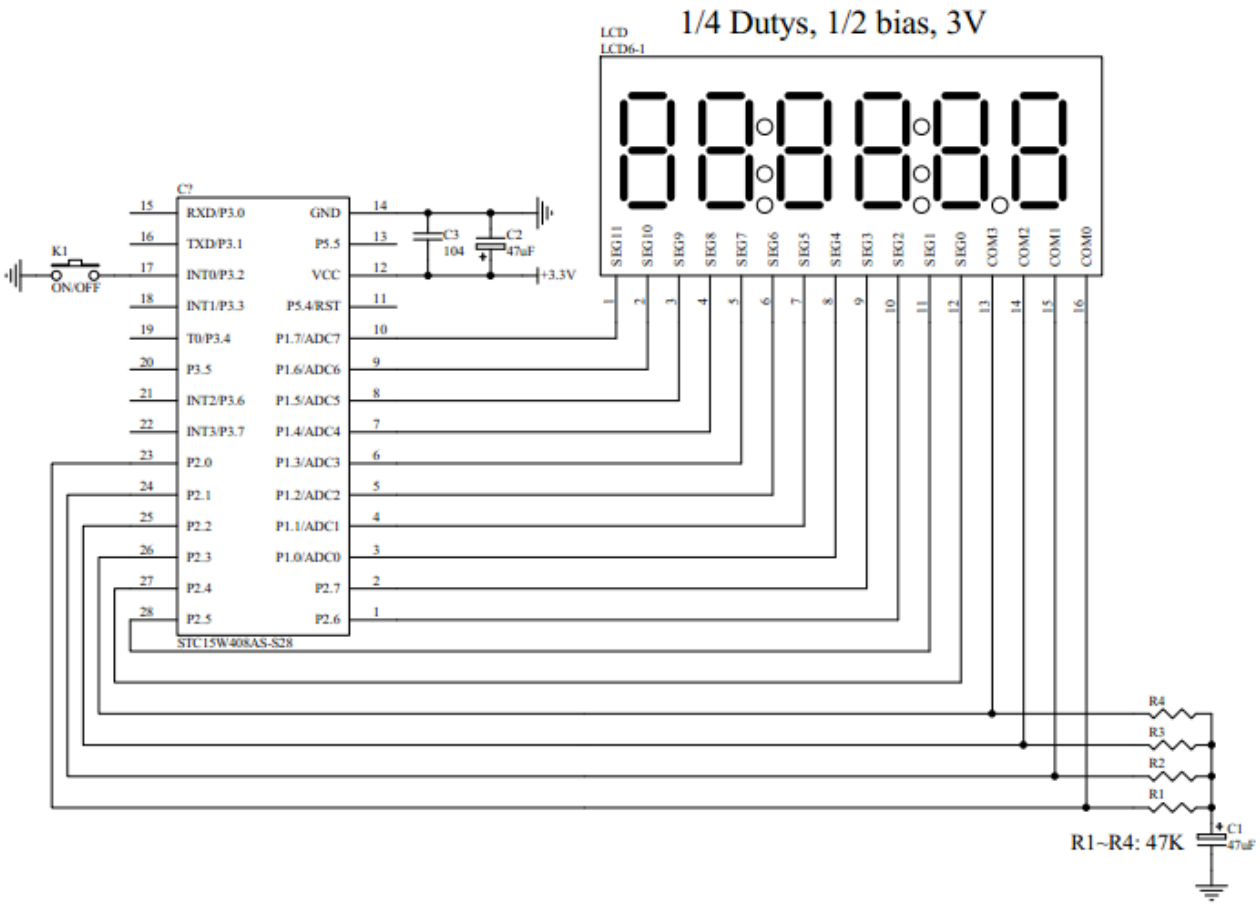

1/4 Dutys, 1/2 bias, 3V
LCD
LCD6-1
C?
STC15W408AS-S28
K1
ON/OFF
C3
104
C2
47uF
+3.3V
R4
R3
R2
R1
C1
47uF
R1~R4: 47K

图 2：段码名称图

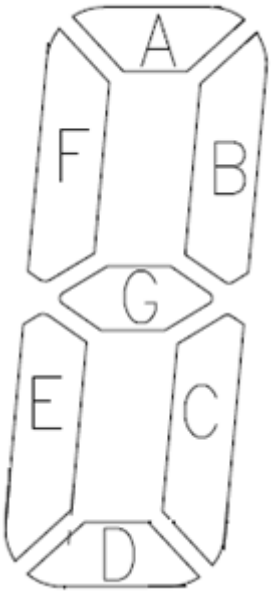

A
F
B
G
E
C
D

图 3：1/4Duty 1/2BIAS 扫描原理图

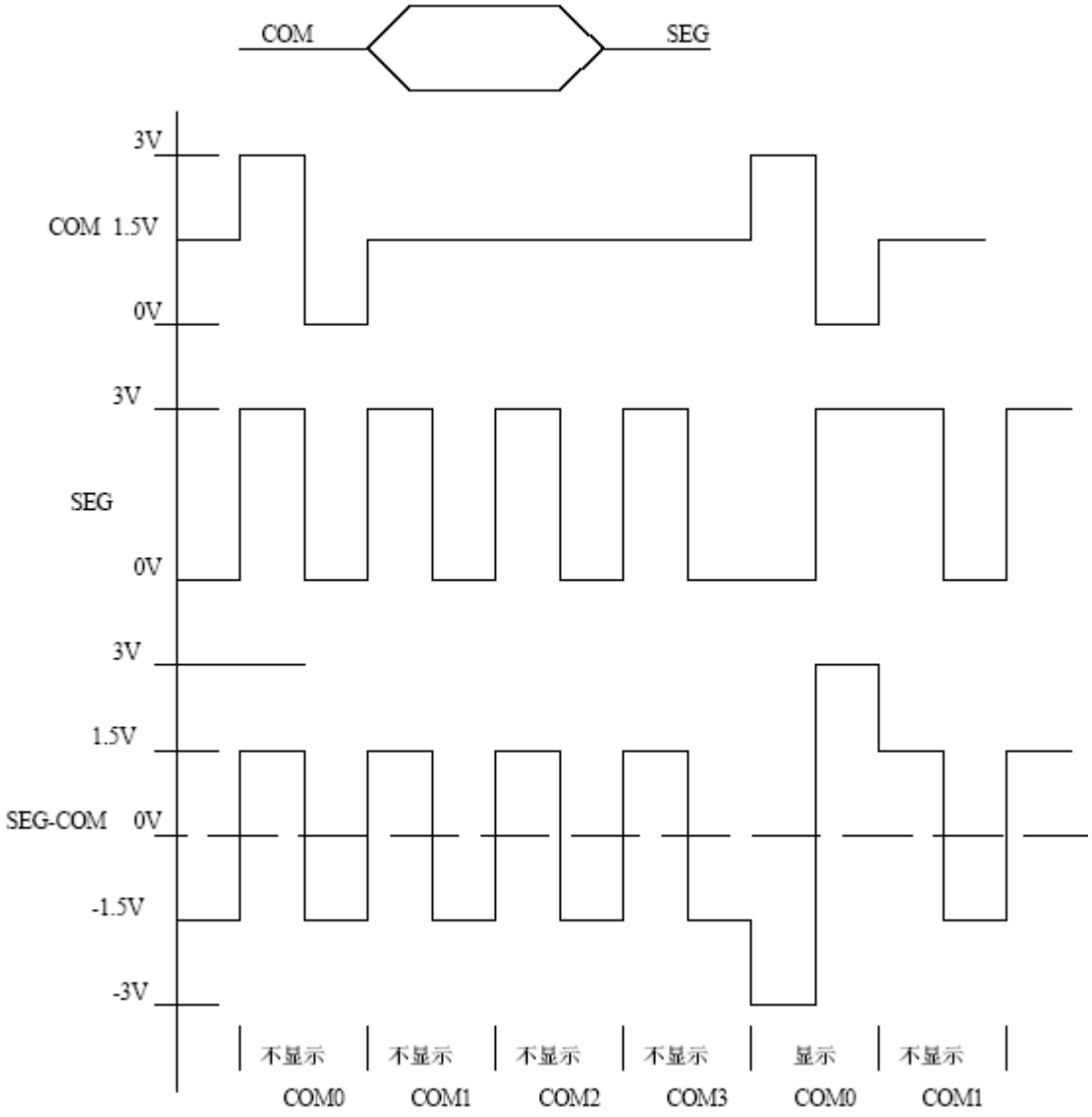


图 4：驱动 1/4Duty 1/3BIAS 3V LCD 的电路

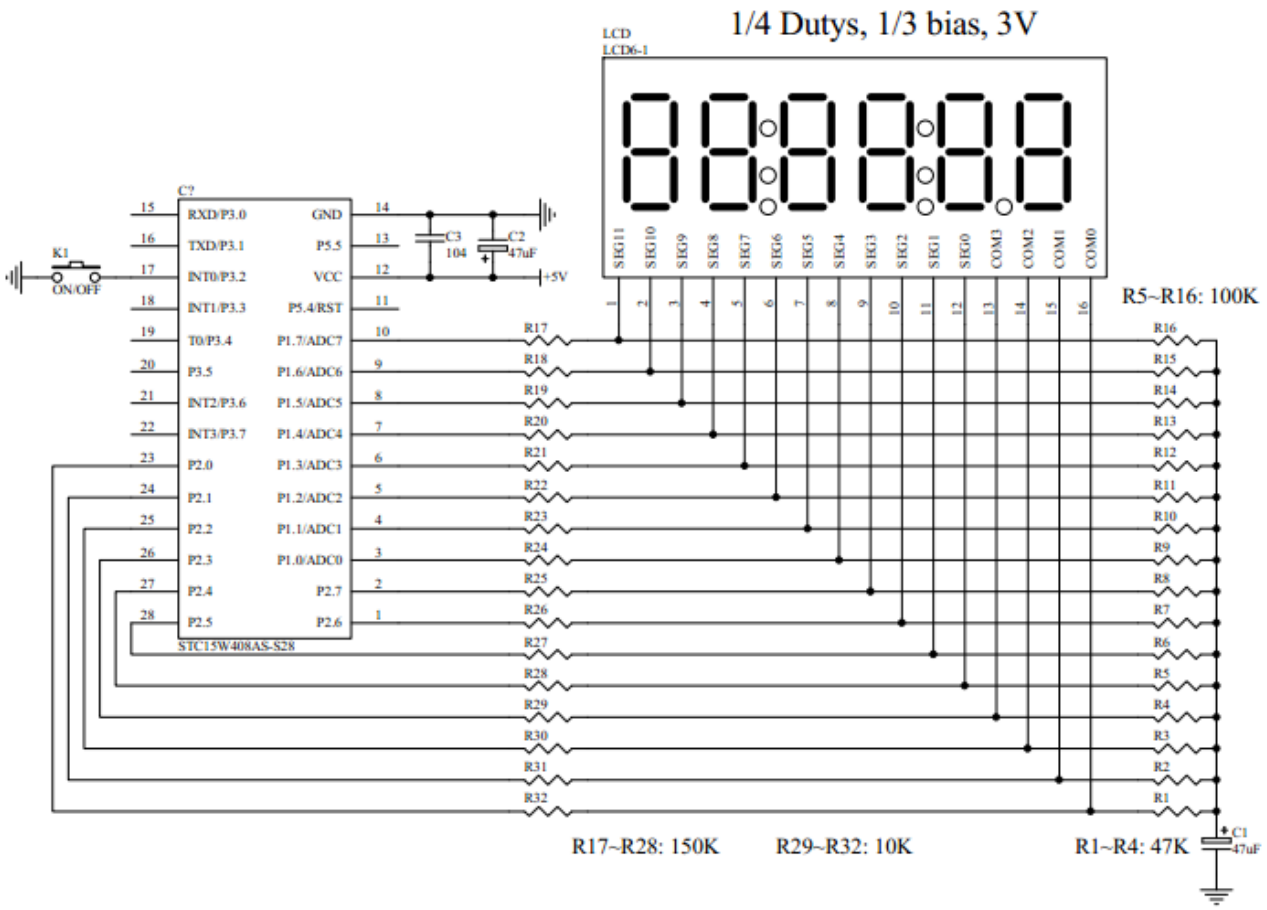
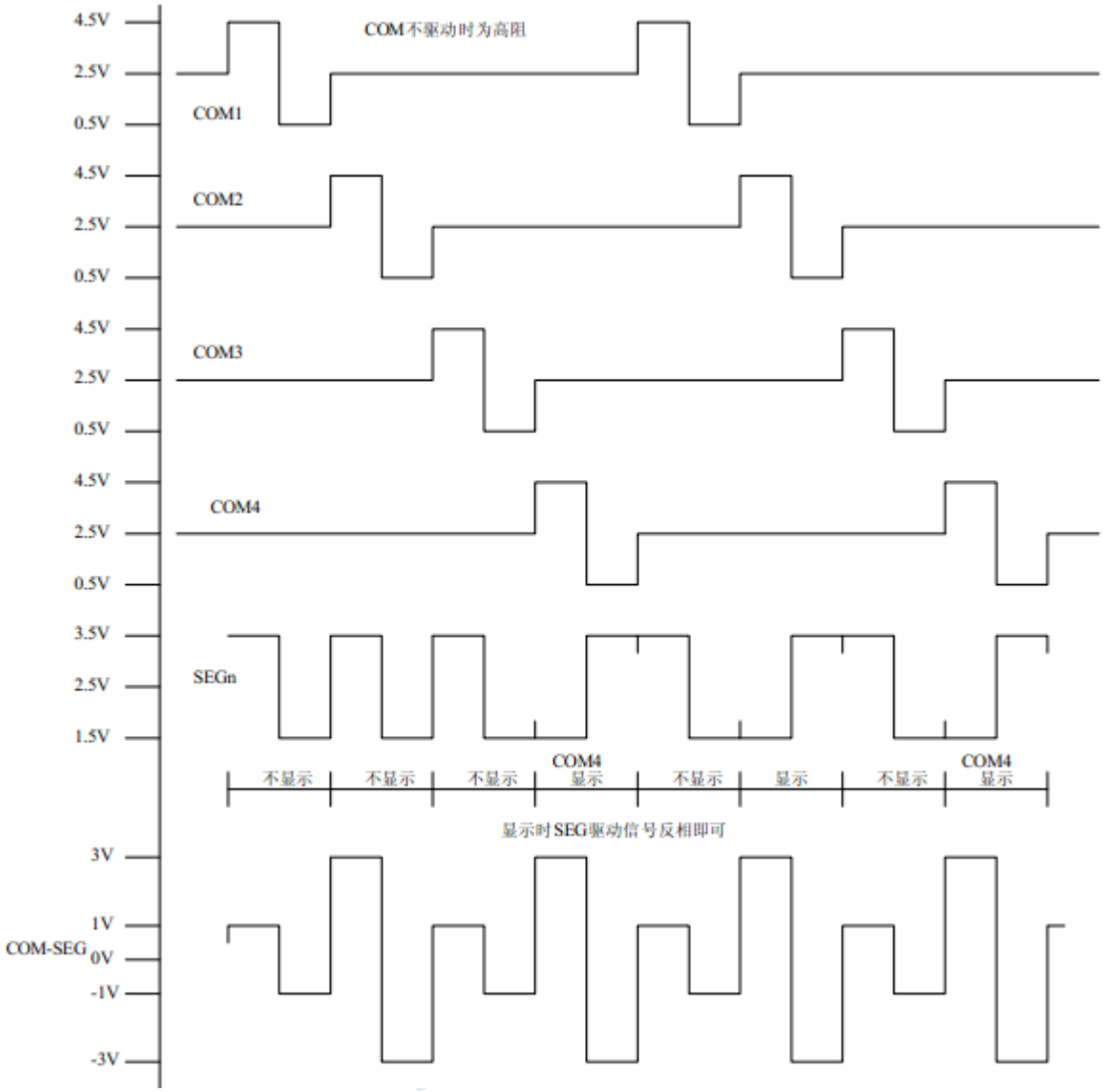


图 5：1/4Duty 1/3BIAS 扫描原理图



为了使用方便，显示内容放在一个显存中，其中的各个位与 LCD 的段一一对应，见图 6。

图 6：LCD 真值表和显存影射表

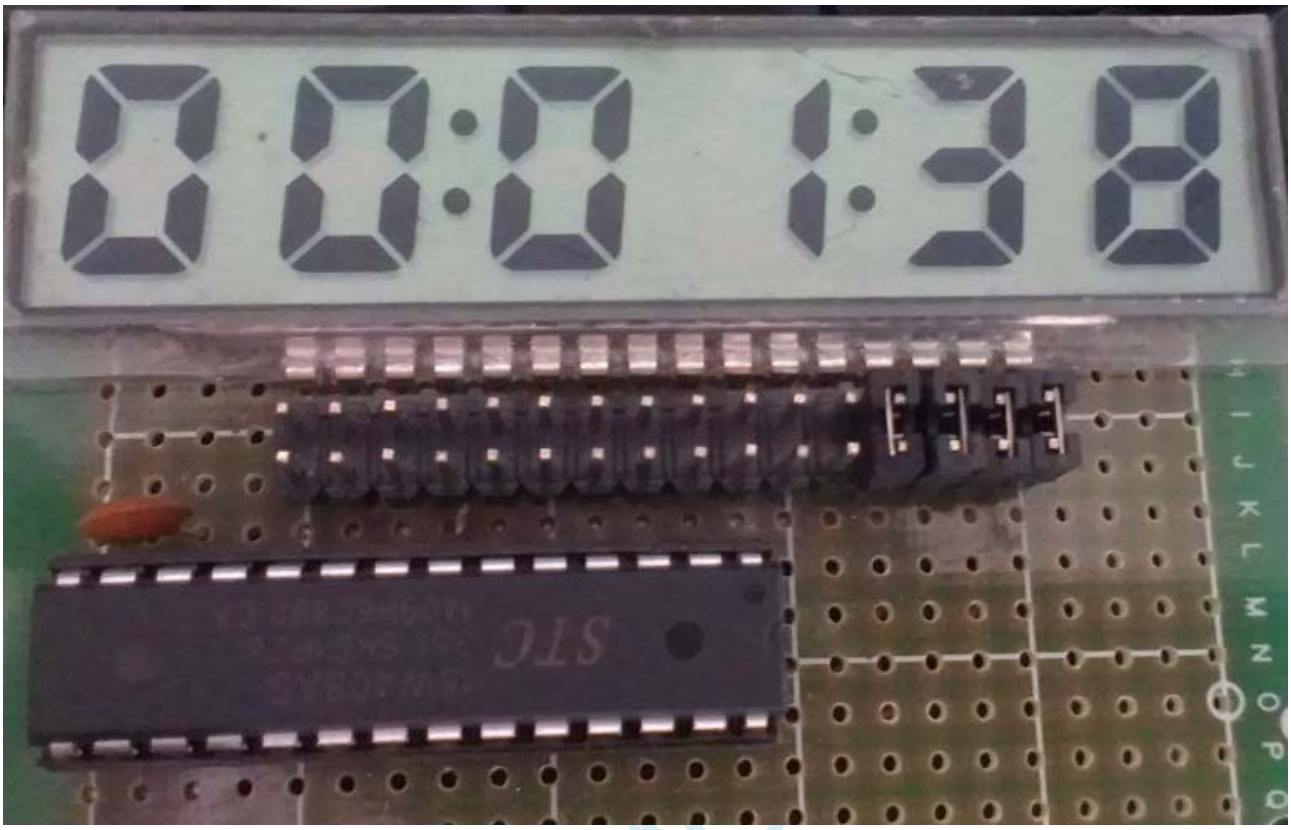
LCD真值表：

MCU PIN	P17	P16	P15	P14	P13	P12	P11	P10	P27	P26	P25	P24	P23	P22	P21	P20
LCD PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
LCD PIN name	SEG11	SEG10	SEG9	SEG8	SEG7	SEG6	SEG5	SEG4	SEG3	SEG2	SEG1	SEG0	COM3	COM2	COM1	COM0
	--	1D	2:	2D	2.	3D	4:	4D	4.	5D	5.	6D	COM3			
	1E	1C	2E	2C	3E	3C	4E	4C	5E	5C	6E	6C		COM2		
	1G	1B	2G	2B	3G	3B	4G	4B	5G	5B	6G	6B			COM1	
	1F	1A	2F	2A	3F	3A	4F	4A	5F	5A	6F	6A				COM0

显存影射表：

	B7	B6	B5	B4	B3	B2	B1	B0
buff[0]:	--	1D	2:	2D	2.	3D	4:	4D
buff[1]:	1E	1C	2E	2C	3E	3C	4E	4C
buff[2]:	1G	1B	2G	2B	3G	3B	4G	4B
buff[3]:	1F	1A	2F	2A	3F	3A	4F	4A
buff[4]:	4.	5D	5.	6D	--	--	--	--
buff[5]:	5E	5C	6E	6C	--	--	--	--
buff[6]:	5G	5B	6G	6B	--	--	--	--
buff[7]:	5F	5A	6F	6A	--	--	--	--

图 7：驱动效果照片



本 LCD 扫描程序仅需要两个函数：

1、LCD 段码扫描函数 void　LCD_scan(void)

程序隔一定的时间调用这个函数，就会将 LCD 显示缓冲的内容显示到 LCD 上，全部扫描一次需要 8 个调用周期，调用间隔一般是 1~2ms，假如使用 1ms，则扫描周期就是 8ms，刷新率就是 125HZ。

2、LCD 段码显示缓冲装载函数 void LCD_load(u8 n,u8 dat)

本函数用来将显示的数字或字符放在 LCD 显示缓冲中，比如 LCD_load(1,6)，就是要在第一个数字位置显示数字 6，支持显示 0~9，A~F，其它字符用户可以自己添加。

另外，用宏来显示、熄灭或闪烁冒号或小数点。

汇编代码

```
;用 STC8 系列测试 I/O 直接驱动段码 LCD(6 个 8 字 LCD, 1/4 Dutys, 1/3 bias)。
;上电后显示一个时间(时分秒).

;*************************************************************
P0M1        DATA        0x93
P0M0        DATA        0x94
P1M1        DATA        0x91
P1M0        DATA        0x92
P2M1        DATA        0x95
P2M0        DATA        0x96
P3M1        DATA        0xB1
P3M0        DATA        0xB2
P4M1        DATA        0xB3
P4M0        DATA        0xB4
P5M1        DATA        0xC9
P5M0        DATA        0xC
```

```
P6M1        DATA    0xCB
P6M0        DATA    0xCC
P7M1        DATA    0xE1
P7M0        DATA    0xE2
AUXR        DATA    0x8E
INT_CLKO    DATA    0x8F
IE2         DATA    0xAF
P4          DATA    0xC0
T2H         DATA    0xD6
T2L         DATA    0xD7

;*******************************************************************
DIS_BLACK   EQU     010H
DIS_        EQU     011H
DIS_A       EQU     00AH
DIS_B       EQU     00BH
DIS_C       EQU     00CH
DIS_D       EQU     00DH
DIS_E       EQU     00EH
DIS_F       EQU     00FH

B_2ms       BIT     20H.0           ;2ms 信号
B_Second    BIT     20H.1           ;秒信号
cnt_500ms   DATA    30H
second      DATA    31H
minute      DATA    32H
hour        DATA    33H
scan_index  DATA    34H

LCD_buff    DATA    40H             ;40H~47H

;*******************************************************************
            ORG     0000H
            LJMP    F_Main

            ORG     000BH
            LJMP    F_Timer0_Interrupt

;*******************************************************************
            ORG     0100H
F_Main:
            CLR     A
            MOV     P3M1, A         ;设置为准双向口
            MOV     P3M0, A
            MOV     P5M1, A         ;设置为准双向口
            MOV     P5M0, A

            MOV     P1M1, #0        ; segment 设置为推挽输出
            MOV     P1M0, #0ffh
            ANL     P2M1, #NOT 0f0h ; segment 设置为推挽输出
            ORL     P2M0, #0f0h
            ORL     P2M1, #00fH     ;全部COM 输出高阻, COM 为中点电压
            ANL     P2M0, #0f0H
            MOV     SP, #0D0H
            MOV     PSW, #0
            USING   0               ;选择第 0 组 R0~R7

;*******************************************************************
            MOV     R2, #8
```

```
                MOV         R0, #LCD_buff
L_ClearLcdRam:
                MOV         @R0, #0
                INC         R0
                DJNZ        R2, L_ClearLcdRam

                LCALL       F_Timer0_init
                SETB        EA

;               ORL         LCD_buff, #020H         ;显示时分间隔:
;               ORL         LCD_buff, #002H         ;显示分秒间隔:

                MOV         hour, #12
                MOV         minute, #00
                MOV         second, #00
                LCALL       F_LoadRTC               ;显示时间

;**********************************************************
L_Main_Loop:
                JNB         B_2ms, L_Main_Loop      ;2ms 节拍到
                CLR         B_2ms

                INC         cnt_500ms
                MOV         A, cnt_500ms
                CJNE        A, #250, L_Main_Loop
                                                    ;500ms 到
                MOV         cnt_500ms, #0;

                XRL         LCD_buff, #020H         ;闪烁时分间隔:
                XRL         LCD_buff, #002H         ;闪烁分秒间隔:

                CPL         B_Second
                JNB         B_Second, L_Main_Loop

                INC         second
                MOV         A, second
                CJNE        A, #60, L_Main_Load
                MOV         second, #0              ; 1 分钟到
                INC         minute
                MOV         A, minute
                CJNE        A, #60, L_Main_Load
                MOV         minute, #0;
                INC         hour
                MOV         A, hour
                CJNE        A, #24, L_Main_Load
                MOV         hour, #0                ;24 小时到
L_Main_Load:
                LCALL       F_LoadRTC               ;显示时间
                LJMP        L_Main_Loop

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

F_Timer0_init:
                CLR         TR0                     ; 停止计数
                ANL         TMOD, #0f0H
                SETB        ET0                     ; 允许中断
                ORL         TMOD, #0                ; 工作模式, 0: 16 位自动重装
                ANL         INT_CLKO, #NOT 0x01     ; 不输出时钟
                ORL         AUXR, #0x80             ; 1T mode
```

```
        MOV     TH0, #HIGH (-22118)     ; 2ms
        MOV     TL0, #LOW  (-22118)     ;
        SETB    TR0                     ; 开始运行
        RET

;*************************************************************
F_Timer0_Interrupt:     ;Timer0 1ms 中断函数
        PUSH    PSW                     ;PSW 入栈
        PUSH    ACC                     ;ACC 入栈
        PUSH    AR0
        PUSH    AR7
        PUSH    DPH
        PUSH    DPL

        LCALL   F_LCD_scan
        SETB    B_2ms

        POP     DPL
        POP     DPH
        POP     AR7
        POP     AR0
        POP     ACC                     ;ACC 出栈
        POP     PSW                     ;PSW 出栈
        RETI

;********************* 显示时间 **************************
F_LoadRTC:
        MOV     R6, #1                  ;LCD_load(1,hour/10);
        MOV     A, hour
        MOV     B, #10
        DIV     AB
        MOV     R7, A
        LCALL   F_LCD_load              ;R6 为第几个数字，为 1~6，R7 为要显示的数字

        MOV     R6, #2                  ;LCD_load(2,hour%10);
        MOV     A, hour
        MOV     B, #10
        DIV     AB
        MOV     R7, B
        LCALL   F_LCD_load              ;R6 为第几个数字，为 1~6，R7 为要显示的数字

        MOV     R6, #3                  ;LCD_load(3,minute/10);
        MOV     A, minute
        MOV     B, #10
        DIV     AB
        MOV     R7, A
        LCALL   F_LCD_load              ;R6 为第几个数字，为 1~6，R7 为要显示的数字

        MOV     R6, #4                  ;LCD_load(4,minute%10);
        MOV     A, minute
        MOV     B, #10
        DIV     AB
        MOV     R7, B
        LCALL   F_LCD_load              ;R6 为第几个数字，为 1~6，R7 为要显示的数字

        MOV     R6, #5                  ;LCD_load(5,second/10);
        MOV     A, second
        MOV     B, #10
        DIV     AB
```

```
                MOV         R7, A
                LCALL       F_LCD_load          ;R6 为第几个数字，为 1~6，R7 为要显示的数字

                MOV         R6, #6              ;LCD_load(6,second%10);
                MOV         A, second
                MOV         B, #10
                DIV         AB
                MOV         R7, B
                LCALL       F_LCD_load          ;R6 为第几个数字，为 1~6，R7 为要显示的数字

                RET

;*********************************************************
T_COM:
                DB          008H, 004H, 002H, 001H

F_LCD_scan:
                MOV         A, scan_index       ;j = scan_index >> 1;
                CLR         C
                RRC         A
                MOV         R7, A               ;R7 = j
                ADD         A, #LCD_buff
                MOV         R0, A               ;R0 = LCD_buff[j]
                ORL         P2M1, #00fH         ;全部 COM 输出高阻, COM 为中点电压
                ANL         P2M0, #0f0H

                MOV         A, scan_index
                JNB         ACC.0, L_LCD_Scan2  ;if(scan_index & 1) //反相扫描
                MOV         A, @R0              ;P1 = ~LCD_buff[j];
                CPL         A
                MOV         P1, A
                MOV         A, R0               ;P2 = ~(LCD_buff[j|4] & 0xf0);
                ADD         A, #4
                MOV         R0, A
                MOV         A, @R0
                ANL         A, #0f0H
                CPL         A
                MOV         P2, A
                SJMP        L_LCD_Scan3

L_LCD_Scan2:                                    ;正相扫描
                MOV         A, @R0              ;P1 = LCD_buff[j];
                MOV         P1, A
                MOV         A, R0               ;P2 = (LCD_buff[j|4] & 0xf0);
                ADD         A, #4
                MOV         R0, A
                MOV         A, @R0
                ANL         A, #0f0H
                MOV         P2, A

L_LCD_Scan3:
                MOV         DPTR, #T_COM        ;某个 COM 设置为推挽输出
                MOV         A, R7
                MOVC        A, @A+DPTR
                ORL         P2M0, A
                CPL         A
                ANL         P2M1, A

                INC         scan_index          ;if(++scan_index == 8)  scan_index = 0;
```

```
                MOV         A, scan_index
                CJNE        A, #8, L_QuitLcdScan
                MOV         scan_index, #0

L_QuitLcdScan:
                RET

;******************** 标准字库 **************************
T_Display:
;               0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
DB              03FH,006H,05BH,04FH,066H,06DH,07DH,007H,07FH,06FH,077H,07CH,039H,05EH,079H,071H
;               black  -
DB              000H,040H

;****************** 对第1~6 数字装载显示函数 算法简单 ****************************
F_LCD_load:                                         ;R6 为第几个数字，为1~6，R7 为要显示的数字
                MOV         DPTR, #T_Display        ;i = t_display[dat];
                MOV         A, R7
                MOVC        A, @A+DPTR
                MOV         B, A                    ;要显示的数字

                MOV         A, R6
                CJNE        A, #1, L_NotLoadChar1
                MOV         R0,                     #LCD_buff
                MOV         A, @R0
                MOV         C, B.3                  ;D
                MOV         ACC.6, C
                MOV         @R0, A

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

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

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

L_NotLoadChar1:
                CJNE        A, #2, L_NotLoadChar2
                MOV         R0,#LCD_buff
                MOV         A, @R0
                MOV         C, B.3                  ;D
```

```
        MOV     ACC.4, C
        MOV     @R0, A

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

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

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

L_NotLoadChar2:
        CJNE    A, #3, L_NotLoadChar3
        MOV     R0,#LCD_buff
        MOV     A, @R0
        MOV     C, B.3          ;D
        MOV     ACC.2, C
        MOV     @R0, A

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

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

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

```
        RET

L_NotLoadChar3:
        CJNE    A, #4, L_NotLoadChar4
        MOV     R0,#LCD_buff
        MOV     A, @R0
        MOV     C, B.3          ;D
        MOV     ACC.0, C
        MOV     @R0, A

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

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

        INC     R0
        MOV     A, @R0
        MOV     C, B.0          ;A
        MOV     ACC.0, C
        MOV     C, B.5          ;F
        MOV     ACC.1, C
        MOV     @R0, A
        RET

L_NotLoadChar4:
        CJNE    A, #5, L_NotLoadChar5
        MOV     R0,#LCD_buff+4
        MOV     A, @R0
        MOV     C, B.3          ;D
        MOV     ACC.6, C
        MOV     @R0, A

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

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

```
            INC     R0
            MOV     A, @R0
            MOV     C, B.0          ;A
            MOV     ACC.6, C
            MOV     C, B.5          ;F
            MOV     ACC.7, C
            MOV     @R0, A
            RET

L_NotLoadChar5:
            CJNE    A, #6, L_NotLoadChar6
            MOV     R0,#LCD_buff+4
            MOV     A, @R0
            MOV     C, B.3          ;D
            MOV     ACC.4, C
            MOV     @R0, A

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

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

            INC     R0
            MOV     A, @R0
            MOV     C, B.0          ;A
            MOV     ACC.4, C
            MOV     C, B.5          ;F
            MOV     ACC.5, C
            MOV     @R0, A
            RET
L_NotLoadChar6:
            RET

            END
```

C 语言代码

```
/*****************功能说明*****************
用STC15系列测试I/O直接驱动段码LCD(6个8字LCD, 1/4 Dutys, 1/3 bias)。
上电后显示一个时间(时分秒).
P3.2对地接一个开关,用来进入睡眠或唤醒.
*****************************************/

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

typedef     unsigned char       u8;
typedef     unsigned int        u16;
```

```
typedef     unsigned long       u32;

sfr AUXR = 0x8e;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;

/*********************本地常量声明**************************/
#define MAIN_Fosc               11059200L                   //定义主时钟

#define DIS_BLACK               0x10
#define DIS_                    0x11
#define DIS_A                   0x0A
#define DIS_B                   0x0B
#define DIS_C                   0x0C
#define DIS_D                   0x0D
#define DIS_E                   0x0E
#define DIS_F                   0x0F

#define LCD_SET_DP2             LCD_buff[0] |=  0x08
#define LCD_CLR_DP2             LCD_buff[0] &= ~0x08
#define LCD_FLASH_DP2           LCD_buff[0] ^=  0x08

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

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

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

#define LCD_SET_DP5             LCD_buff[4] |=  0x20
#define LCD_CLR_DP5             LCD_buff[4] &= ~0x20
#define LCD_FLASH_DP5           LCD_buff[4] ^=  0x20

#define P1n_standard(bitn)      P1M1 &= ~(bitn), P1M0 &= ~(bitn)
#define P1n_push_pull(bitn)     P1M1 &= ~(bitn), P1M0 |= (bitn)
#define P1n_pure_input(bitn)    P1M1 |= (bitn), P1M0 &= ~(bitn)
#define P1n_open_drain(bitn)    P1M1 |= (bitn), P1M0 |= (bitn)

#define P2n_standard(bitn)      P2M1 &= ~(bitn), P2M0 &= ~(bitn)
#define P2n_push_pull(bitn)     P2M1 &= ~(bitn), P2M0 |= (bitn)
#define P2n_pure_input(bitn)    P2M1 |= (bitn), P2M0 &= ~(bitn)
#define P2n_open_drain(bitn)    P2M1 |= (bitn), P2M0 |= (bitn)

/*********************本地变量声明**************************/
u8 cnt_500ms;
u8 second,minute,hour;
bit B_Second;
bit B_2ms;
u8 LCD_buff[8];
u8 scan_index;

/*********************本地函数声明**************************/
```

```
void LCD_load(u8 n,u8 dat);
void LCD_scan(void);
void LoadRTC(void);
void delay_ms(u8 ms);

/********************主函数*************************/
void main(void)
{
    u8    i;

    AUXR = 0x80;
    TMOD = 0x00;
    TL0 = (65536 - (MAIN_Fosc / 500));
    TH0 = (65536 - (MAIN_Fosc / 500)) >> 8;
    TR0 = 1;
    ET0 = 1;
    EA = 1;

                                                    //初始化 LCD 显存
    for(i=0; i<8; i++) LCD_buff[i] = 0;
    P2n_push_pull(0xf0);
    P1n_push_pull(0xff);                            //segment 设置为推挽输出

    LCD_SET_2M;                                     //显示时分间隔:
    LCD_SET_4M;                                     //显示分秒间隔:
    LoadRTC();                                      //显示时间

    while (1)
    {
        PCON |= 0x01;                               //进入空闲模式，由 Timer0 2ms 唤醒退出
        _nop_();
        _nop_();
        _nop_();

        if(B_2ms)                                   //2ms 节拍到
        {
            B_2ms = 0;

            if(++cnt_500ms >= 250)                  //500ms 到
            {
                cnt_500ms = 0;
            //  LCD_FLASH_2M;                       //闪烁时分间隔:
            //  LCD_FLASH_4M;                       //闪烁分秒间隔:

                B_Second = ~B_Second;
                if(B_Second)
                {
                    if(++second >= 60)              //1 分钟到
                    {
                        second = 0;
                        if(++minute >= 60)          //1 小时到
                        {
                            minute = 0;
                            if(++hour >= 24) hour = 0;  //24 小时到
                        }
                    }
                    LoadRTC();                      //显示时间
                }
            }
```

```
                if(!INT0)                                   //键按下，准备睡眠
                {
                    LCD_CLR_2M;                             //显示时分间隔:
                    LCD_CLR_4M;                             //显示分秒间隔:
                    LCD_load(1,DIS_BLACK);
                    LCD_load(2,DIS_BLACK);
                    LCD_load(3,0);
                    LCD_load(4,0x0F);
                    LCD_load(5,0x0F);
                    LCD_load(6,DIS_BLACK);

                    while(!INT0) delay_ms(10);              //等待释放按键
                    delay_ms(50);
                    while(!INT0) delay_ms(10);              //再次等待释放按键

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

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

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

                    LCD_SET_2M;                             //显示时分间隔:
                    LCD_SET_4M;                             //显示分秒间隔:
                    LoadRTC();                              //显示时间
                    TR0 = 1;                                //打开定时器
                    while(!INT0) delay_ms(10);              //等待释放按键
                    delay_ms(50);
                    while(!INT0) delay_ms(10);              //再次等待释放按键
                }
            }
        }
}

/********************延时函数*************************/
void delay_ms(u8 ms)
{
    unsigned int i;
    do{
        i = MAIN_Fosc / 13000;
        while(--i);                                         //14T per loop
    }while(--ms);
}

/********************* Timer0 中断函数***********************/
void timer0_int (void) interrupt 1
{
    LCD_scan();
    B_2ms = 1;
}
```

```
/********************* INT0 中断函数 *************************/
void INT0_int (void) interrupt 0
{
    EX0 = 0;
    IE0  = 0;
}

/******************** LCD 段码扫描函数 ***************************/
void  LCD_scan(void)                                    //5us @22.1184MHZ
{
    u8 code T_COM[4]={0x08,0x04,0x02,0x01};
    u8 j;

    j = scan_index >> 1;
    P2n_pure_input(0x0f);                               //全部 COM 输出高阻, COM 为中点电压
    if(scan_index & 1)                                  //反相扫描
    {
        P1 = ~LCD_buff[j];
        P2 = ~(LCD_buff[j|4] & 0xf0);
    }
    else                                                //正相扫描
    {
        P1 = LCD_buff[j];
        P2 = LCD_buff[j|4] & 0xf0;
    }
    P2n_push_pull(T_COM[j]);                            //某个 COM 设置为推挽输出
    if(++scan_index >= 8) scan_index = 0;
}

/******************* 对第 1~6 数字装载显示函数 ***************************/
void  LCD_load(u8 n, u8 dat)                            //n 为第几个数字，dat 为要显示的数字
{
    u8 code t_display[]={                               //标准字库
        //  0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
        0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71,
        //black  -
        0x00,0x40
    };
    u8 code T_LCD_mask[4]  = {~0xc0,~0x30,~0x0c,~0x03};
    u8 code T_LCD_mask4[4] = {~0x40,~0x10,~0x04,~0x01};
    u8 i,k;
    u8 *p;

    if((n == 0) || (n > 6)) return;
    i = t_display[dat];

    if(n <= 4)                                          //1~4
    {
        n--;
        p = LCD_buff;
    }
    else
    {
        n = n - 5;
        p = &LCD_buff[4];
    }

    k = 0;
```

```
    if(i & 0x08) k |= 0x40;                 //D
    *p = (*p & T_LCD_mask4[n]) | (k>>2*n);
    p++;

    k = 0;
    if(i & 0x04) k |= 0x40;                 //C
    if(i & 0x10) k |= 0x80;                 //E
    *p = (*p & T_LCD_mask[n]) | (k>>2*n);
    p++;

    k = 0;
    if(i & 0x02) k |= 0x40;                 //B
    if(i & 0x40) k |= 0x80;                 //G
    *p = (*p & T_LCD_mask[n]) | (k>>2*n);
    p++;

    k = 0;
    if(i & 0x01) k |= 0x40;                 //A
    if(i & 0x20) k |= 0x80;                 //F
    *p = (*p & T_LCD_mask[n]) | (k>>2*n);
}

/********************显示时间 *************************/
void LoadRTC(void)
{
    LCD_load(1,hour/10);
    LCD_load(2,hour%10);
    LCD_load(3,minute/10);
    LCD_load(4,minute%10);
    LCD_load(5,second/10);
    LCD_load(6,second%10);
}
```

10 指令系统

助记符		指令说明	字节	时钟
ADD	A,Rn	寄存器内容加到累加器	1	1
ADD	A,direct	直接地址单元的数据加到累加器	2	1
ADD	A,@Ri	间接地址单元的数据加到累加器	1	1
ADD	A,#data	立即数加到累加器	2	1
ADDC	A,Rn	寄存器带进位加到累加器	1	1
ADDC	A,direct	直接地址单元的数据带进位加到累加器	2	1
ADDC	A,@Ri	间接地址单元的数据带进位加到累加器	1	1
ADDC	A,#data	立即数带进位加到累加器	2	1
SUBB	A,Rn	累加器带借位减寄存器内容	1	1
SUBB	A,direct	累加器带借位减直接地址单元的内容	2	1
SUBB	A,@Ri	累加器带借位减间接地址单元的内容	1	1
SUBB	A,#data	累加器带借位减立即数	2	1
INC	A	累加器加1	1	1
INC	Rn	寄存器加1	1	1
INC	direct	直接地址单元加1	2	1
INC	@Ri	间接地址单元加1	1	1
DEC	A	累加器减1	1	1
DEC	Rn	寄存器减1	1	1
DEC	direct	直接地址单元减1	2	1
DEC	@Ri	间接地址单元减1	1	1
INC	DPTR	地址寄存器DPTR加1	1	1
MUL	AB	A乘以B,B存放高字节,A存放低字节	1	2
DIV	AB	A除以B,B存放余数,A存放商	1	6
DA	A	累加器十进制调整	1	3
ANL	A,Rn	累加器与寄存器相与	1	1
ANL	A,direct	累加器与直接地址单元相与	2	1
ANL	A,@Ri	累加器与间接地址单元相与	1	1
ANL	A,#data	累加器与立即数相与	2	1
ANL	direct,A	直接地址单元与累加器相与	2	1
ANL	direct,#data	直接地址单元与立即数相与	3	1
ORL	A,Rn	累加器与寄存器相或	1	1
ORL	A,direct	累加器与直接地址单元相或	2	1
ORL	A,@Ri	累加器与间接地址单元相或	1	1
ORL	A,#data	累加器与立即数相或	2	1

ORL	direct,A	直接地址单元与累加器相或	2	1
ORL	direct,#data	直接地址单元与立即数相或	3	1
XRL	A,Rn	累加器与寄存器相异或	1	1
XRL	A,direct	累加器与直接地址单元相异或	2	1
XRL	A,@Ri	累加器与间接地址单元相异或	1	1
XRL	A,#data	累加器与立即数相异或	2	1
XRL	direct,A	直接地址单元与累加器相异或	2	1
XRL	direct,#data	直接地址单元与立即数相异或	3	1
CLR	A	累加器清0	1	1
CPL	A	累加器取反	1	1
RL	A	累加器循环左移	1	1
RLC	A	累加器带进位循环左移	1	1
RR	A	累加器循环右移	1	1
RRC	A	累加器带进位循环右移	1	1
SWAP	A	累加器高低半字节交换	1	1
CLR	C	清零进位位	1	1
CLR	bit	清0直接地址位	2	1
SETB	C	置1进位位	1	1
SETB	bit	置1直接地址位	2	1
CPL	C	进位位求反	1	1
CPL	bit	直接地址位求反	2	1
ANL	C,bit	进位位和直接地址位相与	2	1
ANL	C,/bit	进位位和直接地址位的反码相与	2	1
ORL	C,bit	进位位和直接地址位相或	2	1
ORL	C,/bit	进位位和直接地址位的反码相或	2	1
MOV	C,bit	直接地址位送入进位位	2	1
MOV	bit,C	进位位送入直接地址位	2	1
MOV	A,Rn	寄存器内容送入累加器	1	1
MOV	A,direct	直接地址单元中的数据送入累加器	2	1
MOV	A,@Ri	间接地址中的数据送入累加器	1	1
MOV	A,#data	立即数送入累加器	2	1
MOV	Rn,A	累加器内容送入寄存器	1	1
MOV	Rn,direct	直接地址单元中的数据送入寄存器	2	1
MOV	Rn,#data	立即数送入寄存器	2	1
MOV	direct,A	累加器内容送入直接地址单元	2	1
MOV	direct,Rn	寄存器内容送入直接地址单元	2	1
MOV	direct,direct	直接地址单元中的数据送入另一个直接地址单元	3	1

MOV	direct,@Ri	间接地址中的数据送入直接地址单元	2	1
MOV	direct,#data	立即数送入直接地址单元	3	1
MOV	@Ri,A	累加器内容送间接地址单元	1	1
MOV	@Ri,direct	直接地址单元数据送入间接地址单元	2	1
MOV	@Ri,#data	立即数送入间接地址单元	2	1
MOV	DPTR,#data16	16位立即数送入数据指针	3	1
MOVC	A,@A+DPTR	以DPTR为基地址变址寻址单元中的数据送入累加器	1	4
MOVC	A,@A+PC	以PC为基地址变址寻址单元中的数据送入累加器	1	3
MOVX	A,@Ri	扩展地址(8位地址)的内容送入累加器A中	1	3[1]
MOVX	A,@DPTR	扩展RAM(16位地址)的内容送入累加器A中	1	2[1]
MOVX	@Ri,A	将累加器A的内容送入扩展RAM(8位地址)中	1	3[1]
MOVX	@DPTR,A	将累加器A的内容送入扩展RAM(16位地址)中	1	2[1]
PUSH	direct	直接地址单元中的数据压入堆栈	2	1
POP	direct	栈底数据弹出送入直接地址单元	2	1
XCH	A,Rn	寄存器与累加器交换	1	1
XCH	A,direct	直接地址单元与累加器交换	2	1
XCH	A,@Ri	间接地址与累加器交换	1	1
XCHD	A,@Ri	间接地址的低半字节与累加器交换	1	1
ACALL	addr11	短调用子程序	2	3
LCALL	addr16	长调用子程序	3	3
RET		子程序返回	1	3
RETI		中断返回	1	3
AJMP	addr11	短跳转	2	3
LJMP	addr16	长跳转	3	3
SJMP	rel	相对跳转	2	3
JMP	@A+DPTR	相对于DPTR的间接跳转	1	4
JZ	rel	累加器为零跳转	2	1/3[2]
JNZ	rel	累加器非零跳转	2	1/3[2]
JC	rel	进位位为1跳转	2	1/3[2]
JNC	rel	进位位为0跳转	2	1/3[2]
JB	bit,rel	直接地址位为1则跳转	3	1/3[2]
JNB	bit,rel	直接地址位为0则跳转	3	1/3[2]
JBC	bit,rel	直接地址位为1则跳转,该位清0	3	1/3[2]
CJNE	A,direct,rel	累加器与直接地址单元不相等跳转	3	2/3[3]
CJNE	A,#data,rel	累加器与立即数不相等跳转	3	1/3[2]
CJNE	Rn,#data,rel	寄存器与立即数不相等跳转	3	2/3[3]
CJNE	@Ri,#data,rel	间接地址单元与立即数不相等跳转	3	2/3[3]

DJNZ Rn,rel	寄存器减1后非零跳转	2	2/3[3]
DJNZ direct,rel	直接地址单元减1后非零跳转	3	2/3[3]
NOP	空操作	1	1

[1]:访问外部扩展 RAM 时，指令的执行周期与寄存器 BUS_SPEED 中的 SPEED[1:0]位有关

[2]:对于条件跳转语句的执行时间会依据条件是否满足而不同。当条件不满足时，不会发生跳转而继续执行下一条指令，此时条件跳转语句的执行时间为 1 个时钟；当条件满足时，则会发生跳转，此时条件跳转语句的执行时间为 3 个时钟。

[3]:对于条件跳转语句的执行时间会依据条件是否满足而不同。当条件不满足时，不会发生跳转而继续执行下一条指令，此时条件跳转语句的执行时间为 2 个时钟；当条件满足时，则会发生跳转，此时条件跳转语句的执行时间为 3 个时钟。

11 中断系统

中断系统是为使 CPU 具有对外界紧急事件的实时处理能力而设置的。

当中央处理机 CPU 正在处理某件事的时候外界发生了紧急事件请求，要求 CPU 暂停当前的工作，转而去处理这个紧急事件，处理完以后，再回到原来被中断的地方，继续原来的工作，这样的过程称为中断。实现这种功能的部件称为中断系统，请示 CPU 中断的请求源称为中断源。微型机的中断系统一般允许多个中断源，当几个中断源同时向 CPU 请求中断，要求为它服务的时候，这就存在 CPU 优先响应哪一个中断源请求的问题。通常根据中断源的轻重缓急排队，优先处理最紧急事件的中断请求源，即规定每一个中断源有一个优先级别。CPU 总是先响应优先级别最高的中断请求。

当 CPU 正在处理一个中断源请求的时候（执行相应的中断服务程序），发生了另外一个优先级比它还高的中断源请求。如果 CPU 能够暂停对原来中断源的服务程序，转而去处理优先级更高的中断请求源，处理完以后，再回到原低级中断服务程序，这样的过程称为中断嵌套。这样的中断系统称为多级中断系统，没有中断嵌套功能的中断系统称为单级中断系统。

用户可以用关总中断允许位（EA/IE.7）或相应中断的允许位屏蔽相应的中断请求，也可以用打开相应的中断允许位来使 CPU 响应相应的中断申请，每一个中断源可以用软件独立地控制为开中断或关中断状态，部分中断的优先级别均可用软件设置。高优先级的中断请求可以打断低优先级的中断，反之，低优先级的中断请求不可以打断高优先级的中断。当两个相同优先级的中断同时产生时，将由查询次序来决定系统先响应哪个中断。

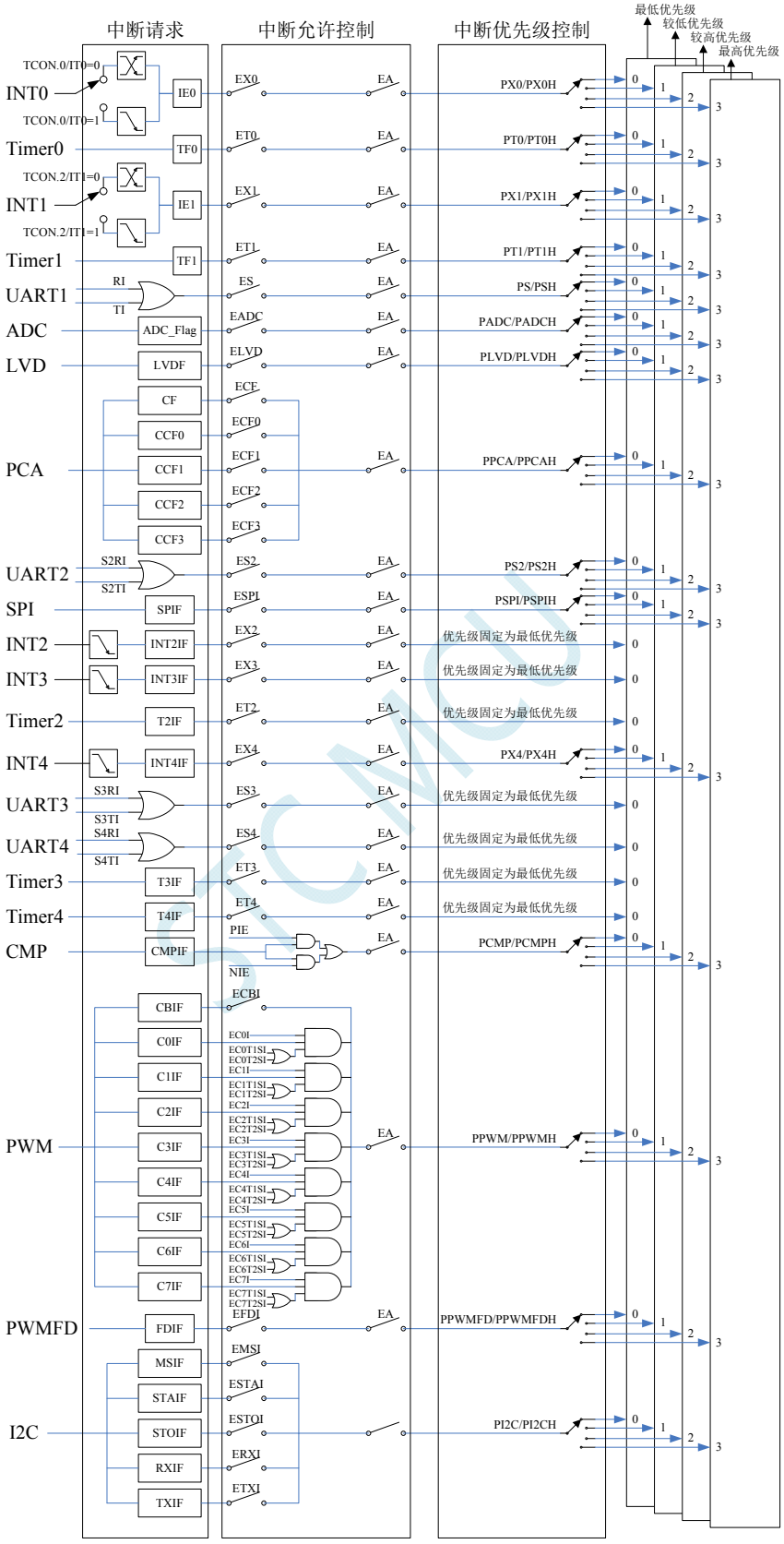
11.1 STC8G系列中断源

下表中 √ 表示对应的系列有相应的中断源

中断源	STC8G1K08系列
外部中断 0 中断（INT0）	√
定时器 0 中断（Timer0）	√
外部中断 1 中断（INT1）	√
定时器 1 中断（Timer1）	√
串口 1 中断（UART1）	√
模数转换中断（ADC）	√
低压检测中断（LVD）	√
捕获中断（CCP/PCA/PWM）	√
串口 2 中断（UART2）	√
串行外设接口中断（SPI）	√
外部中断 2 中断（INT2）	√
外部中断 3 中断（INT3）	√
定时器 2 中断（Timer2）	√
外部中断 4 中断（INT4）	√
串口 3 中断（UART3）	
串口 4 中断（UART4）	

定时器 3 中断（Timer3）	
定时器 4 中断（Timer4）	
比较器中断（CMP）	√
增强型 PWM 中断	
PWM 异常检测中断（PWMFD）	
I2C 总线中断	√

11.2 STC8 中断结构图



11.3 STC8 系列中断列表

中断源	中断向量	次序	优先级设置	优先级	中断请求位	中断允许位
INT0	0003H	0	PX0PX0H	0/1/2/3	IE0	EX0
Timer0	000BH	1	PT0,PT0H	0/1/2/3	TF0	ET0
INT1	0013H	2	PX1,PX1H	0/1/2/3	IE1	EX1
Timer1	001BH	3	PT1,PT1H	0/1/2/3	TF1	ET1
UART1	0023H	4	PS,PSH	0/1/2/3	RI || TI	ES
ADC	002BH	5	PADC,PADCH	0/1/2/3	ADC_FLAG	EADC
LVD	0033H	6	PLVD,PLVDH	0/1/2/3	LVDF	ELVD
PCA	003BH	7	PPCA,PPCAH	0/1/2/3	CF	ECF
					CCF0	ECCF0
					CCF1	ECCF1
					CCF2	ECCF2
					CCF3	ECCF3
UART2	0043H	8	PS2,PS2H	0/1/2/3	S2RI || S2TI	ES2
SPI	004BH	9	PSPI,PSPIH	0/1/2/3	SPIF	ESPI
INT2	0053H	10		0	INT2IF	EX2
INT3	005BH	11		0	INT3IF	EX3
Timer2	0063H	12		0	T2IF	ET2
INT4	0083H	16	PX4,PX4H	0/1/2/3	INT4IF	EX4
CMP	00ABH	21	PCMP,PCMPH	0/1/2/3	CMPIF	PIE|NIE
I2C	00C3H	24	PI2C,PI2CH	0/1/2/3	MSIF	EMSI
					STAIF	ESTAI
					RXIF	ERXI
					TXIF	ETXI
					STOIF	ESTOI

在 C 语言中声明中断服务程序

```
void    INT0_Routine(void)      interrupt  0;
void    TM0_Rountine(void)      interrupt  1;
void    INT1_Routine(void)      interrupt  2;
void    TM1_Rountine(void)      interrupt  3;
void    UART1_Routine(void)     interrupt  4;
void    ADC_Routine(void)       interrupt  5;
void    LVD_Routine(void)       interrupt  6;
void    PCA_Routine(void)       interrupt  7;
void    UART2_Routine(void)     interrupt  8;
void    SPI_Routine(void)       interrupt  9;
void    INT2_Routine(void)      interrupt  10;
void    INT3_Routine(void)      interrupt  11;
void    TM2_Routine(void)       interrupt  12;
void    INT4_Routine(void)      interrupt  16;
void    CMP_Routine(void)       interrupt  21;
void    I2C_Routine(void)       interrupt  24;
```

11.4 中断相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
IE	中断允许寄存器	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000,0000
IE2	中断允许寄存器 2	AFH	-	-	-	-	-	ET2	ESPI	ES2	xxxx,x000
INTCLKO	中断与时钟输出控制寄存器	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
IP	中断优先级控制寄存器	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000,0000
IPH	高中断优先级控制寄存器	B7H	PPCAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	0000,0000
IP2	中断优先级控制寄存器 2	B5H	-	PI2C	PCMP	PX4	-	-	PSPI	PS2	x000,xx00
IP2H	高中断优先级控制寄存器 2	B6H	-	PI2CH	PCMPH	PX4H	-	-	PSPIH	PS2H	x000,xx00
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
AUXINTIF	扩展外部中断标志寄存器	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF	x000,x000
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
S2CON	串口 2 控制寄存器	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
ADC_CONTR	ADC 控制寄存器	BCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]				000x,0000
SPSTAT	SPI 状态寄存器	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
CCON	PCA 控制寄存器	D8H	CF	CR	-	-	-	CCF2	CCF1	CCF0	00xx,x000
CMOD	PCA 模式寄存器	D9H	CIDL	-	-	-	CPS[2:0]			ECF	0xxx,0000
CCAPM0	PCA 模块 0 模式控制寄存器	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAPM1	PCA 模块 1 模式控制寄存器	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CCAPM2	PCA 模块 2 模式控制寄存器	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
CMPCR1	比较器控制寄存器 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
I2CMSCR	I^2C 主机控制寄存器	FE81H	EMSI	-	-	-	-	MSCMD[2:0]			0xxx,x000
I2CMSST	I^2C 主机状态寄存器	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I^2C 从机控制寄存器	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I^2C 从机状态寄存器	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000

11.4.1 中断使能寄存器（中断允许位）

IE（中断使能寄存器）

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

EA：总中断允许控制位。EA 的作用是使中断允许形成多级控制。即各中断源首先受 EA 控制;其次还受各中断源自己的中断允许控制位控制。

0：CPU 屏蔽所有的中断申请

1：CPU 开放中断

ELVD：低压检测中断允许位。

0：禁止低压检测中断

1：允许低压检测中断

EADC：A/D 转换中断允许位。

0：禁止 A/D 转换中断

1：允许 A/D 转换中断

ES：串行口 1 中断允许位。

0：禁止串行口 1 中断

1：允许串行口 1 中断

ET1：定时/计数器 T1 的溢出中断允许位。

0：禁止 T1 中断

1：允许 T1 中断

EX1：外部中断 1 中断允许位。

0：禁止 INT1 中断

1：允许 INT1 中断

ET0：定时/计数器 T0 的溢出中断允许位。

0：禁止 T0 中断

1：允许 T0 中断

EX0：外部中断 0 中断允许位。

0：禁止 INT0 中断

1：允许 INT0 中断

IE2（中断使能寄存器 2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	-	-	-	-	-	ET2	ESPI	ES2

ET2：定时/计数器 T2 的溢出中断允许位。

0：禁止 T2 中断

1：允许 T3 中断

ESPI：SPI 中断允许位。

0：禁止 SPI 中断

1：允许 SPI 中断

ES2：串行口 2 中断允许位。

0：禁止串行口 2 中断

1：允许串行口 2 中断

INTCLKO（外部中断与时钟输出控制寄存器）

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

EX4：外部中断 4 中断允许位。

0：禁止 INT4 中断

1：允许 INT4 中断

EX3：外部中断 3 中断允许位。

0：禁止 INT3 中断

1：允许 INT3 中断

EX2：外部中断 2 中断允许位。

0：禁止 INT2 中断

1：允许 INT2 中断

PCA/CCP/PWM 中断控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	D9H	CIDL	-	-	-	CPS[2:0]			ECF
CCAPM0	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0
CCAPM1	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1
CCAPM2	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2

ECF：PCA 计数器中断允许位。

0：禁止 PCA 计数器中断

1：允许 PCA 计数器中断

ECCF0：PCA 模块 0 中断允许位。

0：禁止 PCA 模块 0 中断

1：允许 PCA 模块 0 中断

ECCF1：PCA 模块 1 中断允许位。

0：禁止 PCA 模块 1 中断

1：允许 PCA 模块 1 中断

ECCF2：PCA 模块 2 中断允许位。

0：禁止 PCA 模块 2 中断

1：允许 PCA 模块 2 中断

CMPCR1（比较器控制寄存器 1）

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

PIE：比较器上升沿中断允许位。

0：禁止比较器上升沿中断

1：允许比较器上升沿中断

NIE：比较器下降沿中断允许位。

0：禁止比较器下降沿中断

1：允许比较器下降沿中断

I2C 控制寄存器

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

EMSI：I^2C主机模式中断允许位。

0：禁止 I^2C 主机模式中断

1：允许 I^2C 主机模式中断

ESTAI：I^2C从机接收START事件中断允许位。

0：禁止 I^2C 从机接收 START 事件中断

1：允许 I^2C 从机接收 START 事件中断

ERXI：I^2C从机接收数据完成事件中断允许位。

0：禁止 I^2C 从机接收数据完成事件中断

1：允许 I²C 从机接收数据完成事件中断

ETXI：I²C从机发送数据完成事件中断允许位。

0：禁止 I²C 从机发送数据完成事件中断

1：允许 I²C 从机发送数据完成事件中断

ESTOI：I²C从机接收STOP事件中断允许位。

0：禁止 I²C 从机接收 STOP 事件中断

1：允许 I²C 从机接收 STOP 事件中断

11.4.2 中断请求寄存器（中断标志位）

定时器控制寄存器

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

TF1：定时器1溢出中断标志。中断服务程序中，硬件自动清零。

TF0：定时器0溢出中断标志。中断服务程序中，硬件自动清零。

IE1：外部中断1中断请求标志。中断服务程序中，硬件自动清零。

IE0：外部中断0中断请求标志。中断服务程序中，硬件自动清零。

中断标志辅助寄存器

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

INT4IF：外部中断4中断请求标志。需要软件清零。

INT3IF：外部中断3中断请求标志。需要软件清零。

INT2IF：外部中断2中断请求标志。需要软件清零。

T4IF：定时器4溢出中断标志。需要软件清零。

T3IF：定时器3溢出中断标志。需要软件清零。

T2IF：定时器2溢出中断标志。需要软件清零。

串口控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI
S2CON	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI

TI：串口1发送完成中断请求标志。需要软件清零。

RI：串口1接收完成中断请求标志。需要软件清零。

S2TI：串口2发送完成中断请求标志。需要软件清零。

S2RI：串口2接收完成中断请求标志。需要软件清零。

电源管理寄存器

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

LVDF：低压检测中断请求标志。需要软件清零。

ADC 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADC_CONTR	BCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]			

ADC_FLAG：ADC转换完成中断请求标志。需要软件清零。

SPI 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	CDH	SPIF	WCOL	-	-	-	-	-	-

SPIF：SPI数据传输完成中断请求标志。需要软件清零。

PCA 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCON	D8H	CF	CR	-	-	-	CCF2	CCF1	CCF0

CF：PCA计数器中断请求标志。需要软件清零。

CCF2：PCA模块2中断请求标志。需要软件清零。

CCF1：PCA模块1中断请求标志。需要软件清零。

CCF0：PCA模块0中断请求标志。需要软件清零。

比较器控制寄存器 1

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

CMPIF：比较器中断请求标志。需要软件清零。

I2C 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSST	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO
I2CSLST	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO

MSIF：I^2C主机模式中断请求标志。需要软件清零。

ESTAI：I^2C从机接收START事件中断请求标志。需要软件清零。

ERXI：I^2C从机接收数据完成事件中断请求标志。需要软件清零。

ETXI：I^2C从机发送数据完成事件中断请求标志。需要软件清零。

ESTOI：I^2C从机接收STOP事件中断请求标志。需要软件清零。

11.4.3 中断优先级寄存器

中断优先级控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0
IPH	B7H	PPCAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H
IP2	B5H	-	PI2C	PCMP	PX4	-	-	PSPI	PS2
IP2H	B6H	-	PI2CH	PCMPH	PX4H	-	-	PSPIH	PS2H

PX0H,PX0：外部中断0中断优先级控制位

00：INT0 中断优先级为 0 级（最低级）
01：INT0 中断优先级为 1 级（较低级）
10：INT0 中断优先级为 2 级（较高级）
11：INT0 中断优先级为 3 级（最高级）

PT0H,PT0：定时器0中断优先级控制位

00：定时器 0 中断优先级为 0 级（最低级）
01：定时器 0 中断优先级为 1 级（较低级）
10：定时器 0 中断优先级为 2 级（较高级）
11：定时器 0 中断优先级为 3 级（最高级）

PX1H,PX1：外部中断1中断优先级控制位

00：INT1 中断优先级为 0 级（最低级）
01：INT1 中断优先级为 1 级（较低级）
10：INT1 中断优先级为 2 级（较高级）
11：INT1 中断优先级为 3 级（最高级）

PT1H,PT1：定时器1中断优先级控制位

00：定时器 1 中断优先级为 0 级（最低级）
01：定时器 1 中断优先级为 1 级（较低级）
10：定时器 1 中断优先级为 2 级（较高级）
11：定时器 1 中断优先级为 3 级（最高级）

PSH,PS：串口1中断优先级控制位

00：串口 1 中断优先级为 0 级（最低级）
01：串口 1 中断优先级为 1 级（较低级）
10：串口 1 中断优先级为 2 级（较高级）
11：串口 1 中断优先级为 3 级（最高级）

PADCH,PADC：ADC中断优先级控制位

00：ADC 中断优先级为 0 级（最低级）
01：ADC 中断优先级为 1 级（较低级）
10：ADC 中断优先级为 2 级（较高级）
11：ADC 中断优先级为 3 级（最高级）

PLVDH,PLVD：低压检测中断优先级控制位

00：LVD 中断优先级为 0 级（最低级）
01：LVD 中断优先级为 1 级（较低级）
10：LVD 中断优先级为 2 级（较高级）
11：LVD 中断优先级为 3 级（最高级）

PPCAH,PPCA：CCP/PCA/PWM中断优先级控制位

00：CCP/PCA/PWM 中断优先级为 0 级（最低级）
01：CCP/PCA/PWM 中断优先级为 1 级（较低级）
10：CCP/PCA/PWM 中断优先级为 2 级（较高级）
11：CCP/PCA/PWM 中断优先级为 3 级（最高级）

PS2H,PS2：串口2中断优先级控制位

00：串口 2 中断优先级为 0 级（最低级）
01：串口 2 中断优先级为 1 级（较低级）
10：串口 2 中断优先级为 2 级（较高级）

11：串口 2 中断优先级为 3 级（最高级）

PSPIH,PSPI：SPI中断优先级控制位

00：SPI 中断优先级为 0 级（最低级）
01：SPI 中断优先级为 1 级（较低级）
10：SPI 中断优先级为 2 级（较高级）
11：SPI 中断优先级为 3 级（最高级）

PX4H,PX4：外部中断4中断优先级控制位

00：INT4 中断优先级为 0 级（最低级）
01：INT4 中断优先级为 1 级（较低级）
10：INT4 中断优先级为 2 级（较高级）
11：INT4 中断优先级为 3 级（最高级）

PCMPH,PCMP：比较器中断优先级控制位

00：CMP 中断优先级为 0 级（最低级）
01：CMP 中断优先级为 1 级（较低级）
10：CMP 中断优先级为 2 级（较高级）
11：CMP 中断优先级为 3 级（最高级）

PI2CH,PI2C：I2C中断优先级控制位

00：I2C 中断优先级为 0 级（最低级）
01：I2C 中断优先级为 1 级（较低级）
10：I2C 中断优先级为 2 级（较高级）
11：I2C 中断优先级为 3 级（最高级）

11.5 范例程序

11.5.1 INT0 中断（上升沿和下降沿）

汇编代码

```
;测试工作频率为11.0592MHz

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0003H
            LJMP    INT0ISR

            ORG     0100H
INT0ISR:
            JB      INT0,RISING         ;判断上升沿和下降沿
            CPL     P1.0                ;测试端口
            RETI
RISING:
            CPL     P1.1                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            CLR     IT0                 ;使能 INT0 上升沿和下降沿中断
            SETB    EX0                 ;使能 INT0 中断
            SETB    EA
            JMP     $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;
```

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

void INT0_Isr() interrupt 0
{
    if (INT0)                                   //判断上升沿和下降沿
    {
        P10 = !P10;                             //测试端口
    }
    else
    {
        P11 = !P11;                             //测试端口
    }
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IT0 = 0;                                    //使能 INT0 上升沿和下降沿中断
    EX0 = 1;                                    //使能 INT0 中断
    EA = 1;

    while (1);
}
```

11.5.2 INT0 中断（下降沿）

汇编代码

```
;测试工作频率为 11.0592MHz

P1M1        DATA        091H
P1M0        DATA        092H
P3M1        DATA        0B1H
P3M0        DATA        0B2H
P5M1        DATA        0C9H
P5M0        DATA        0CAH

            ORG         0000H
            LJMP        MAIN
            ORG         0003H
            LJMP        INT0ISR

            ORG         0100H
INT0ISR:
            CPL         P1.0                    ;测试端口
            RETI

MAIN:
            MOV         SP, #5FH
            MOV         P1M0, #00H
            MOV         P1M1, #00H
```

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

        SETB    IT0                 ;使能 INT0 下降沿中断
        SETB    EX0                 ;使能 INT0 中断
        SETB    EA
        JMP     $

        END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void INT0_Isr() interrupt 0
{
    P10 = !P10;                     //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IT0 = 1;                        //使能 INT0 下降沿中断
    EX0 = 1;                        //使能 INT0 中断
    EA = 1;

    while (1);
}
```

11.5.3 INT1 中断（上升沿和下降沿）

汇编代码

```
;测试工作频率为 11.0592MHz

P1M1    DATA    091H
P1M0    DATA    092H
P3M1    DATA    0B1H
```

```
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0013H
            LJMP    INT1ISR

            ORG     0100H
INT1ISR:
            JB      INT1,RISING         ;判断上升沿和下降沿
            CPL     P1.0                ;测试端口
            RETI
RISING:
            CPL     P1.1                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            CLR     IT1                 ;使能 INT1 上升沿和下降沿中断
            SETB    EX1                 ;使能 INT1 中断
            SETB    EA
            JMP     $

            END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

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

void INT1_Isr() interrupt 2
{
    if (INT1)                           //判断上升沿和下降沿
    {
        P10 = !P10;                     //测试端口
    }
    else
    {
```

```
            P11 = !P11;                                 //测试端口
        }
}

void main()
{
        P1M0 = 0x00;
        P1M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        IT1 = 0;                                        //使能 INT1 上升沿和下降沿中断
        EX1 = 1;                                        //使能 INT1 中断
        EA = 1;

        while (1);
}
```

11.5.4 INT1 中断（下降沿）

汇编代码

```
;测试工作频率为 11.0592MHz

P1M1        DATA        091H
P1M0        DATA        092H
P3M1        DATA        0B1H
P3M0        DATA        0B2H
P5M1        DATA        0C9H
P5M0        DATA        0CAH

            ORG         0000H
            LJMP        MAIN
            ORG         0013H
            LJMP        INT1ISR

            ORG         0100H
INT1ISR:
            CPL         P1.0                            ;测试端口
            RETI

MAIN:
            MOV         SP, #5FH
            MOV         P1M0, #00H
            MOV         P1M1, #00H
            MOV         P3M0, #00H
            MOV         P3M1, #00H
            MOV         P5M0, #00H
            MOV         P5M1, #00H

            SETB        IT1                             ;使能 INT1 下降沿中断
            SETB        EX1                             ;使能 INT1 中断
            SETB        EA
            JMP         $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        P10             =   P1^0;

void INT1_Isr() interrupt 2
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IT1 = 1;                                    //使能 INT1 下降沿中断
    EX1 = 1;                                    //使能 INT1 中断
    EA = 1;

    while (1);
}
```

11.5.5 INT2 中断（下降沿）

汇编代码

```
;测试工作频率为11.0592MHz

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

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0053H
```

```
            LJMP        INT2ISR

            ORG         0100H
INT2ISR:
            CPL         P1.0                ;测试端口
            RETI

MAIN:
            MOV         SP, #5FH
            MOV         P1M0, #00H
            MOV         P1M1, #00H
            MOV         P3M0, #00H
            MOV         P3M1, #00H
            MOV         P5M0, #00H
            MOV         P5M1, #00H

            MOV         INTCLKO,#EX2        ;使能 INT2 中断
            SETB        EA
            JMP         $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sfr         INTCLKO     =   0x8f;
#define     EX2             0x10
#define     EX3             0x20
#define     EX4             0x40
sbit        P10         =   P1^0;

void INT2_Isr() interrupt 10
{
    P10 = !P10;                             //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    INTCLKO = EX2;                          //使能 INT2 中断
    EA = 1;
```

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

11.5.6 INT3 中断（下降沿）

汇编代码

```
;测试工作频率为11.0592MHz

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

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     005BH
            LJMP    INT3ISR

            ORG     0100H
INT3ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     INTCLKO,#EX3        ;使能 INT3 中断
            SETB    EA
            JMP     $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
```

```
sfr         P5M0        =   0xca;

sfr         INTCLKO     =   0x8f;
#define     EX2             0x10
#define     EX3             0x20
#define     EX4             0x40
sbit        P10         =   P1^0;

void INT3_Isr() interrupt 11
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    INTCLKO = EX3;                              //使能 INT3 中断
    EA = 1;

    while (1);
}
```

11.5.7 INT4 中断（下降沿）

汇编代码

```
;测试工作频率为 11.0592MHz

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

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0083H
            LJMP    INT4ISR

            ORG     0100H
INT4ISR:
            CPL     P1.0                    ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
```

```
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     INTCLKO,#EX4            ;使能 INT4 中断
        SETB    EA
        JMP     $

        END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sfr     INTCLKO =   0x8f;
#define EX2     0x10
#define EX3     0x20
#define EX4     0x40
sbit    P10     =   P1^0;

void INT4_Isr() interrupt 16
{
    P10 = !P10;                         //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    INTCLKO = EX4;                      //使能 INT4 中断
    EA = 1;

    while (1);
}
```

11.5.8 定时器 0 中断

汇编代码

```
;测试工作频率为 11.0592MHz
```

```
P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     000BH
            LJMP    TM0ISR

            ORG     0100H
TM0ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     TMOD,#00H
            MOV     TL0,#66H            ;65536-11.0592M/12/1000
            MOV     TH0,#0FCH
            SETB    TR0                 ;启动定时器
            SETB    ET0                 ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        P10         =   P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;                         //测试端口
}

void main()
```

```
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;
    TL0 = 0x66;                         //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                            //启动定时器
    ET0 = 1;                            //使能定时器中断
    EA = 1;

    while (1);
}
```

11.5.9 定时器 1 中断

汇编代码

```
;测试工作频率为11.0592MHz

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     001BH
            LJMP    TM1ISR

            ORG     0100H
TM1ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     TMOD,#00H
            MOV     TL1,#66H            ;65536-11.0592M/12/1000
            MOV     TH1,#0FCH
            SETB    TR1                 ;启动定时器
            SETB    ET1                 ;使能定时器中断
            SETB    EA

            JMP     $
```

```
        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM1_Isr() interrupt 3
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;
    TL1 = 0x66;                                 //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                                    //启动定时器
    ET1 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

11.5.10 定时器 2 中断

汇编代码

```
;测试工作频率为11.0592MHz

T2L     DATA    0D7H
T2H     DATA    0D6H
AUXR    DATA    8EH
IE2     DATA    0AFH
ET2     EQU     04H
AUXINTIF    DATA    0EFH
T2IF    EQU     01H

P1M1    DATA    091H
P1M0    DATA    092H
```

```
P3M1        DATA        0B1H
P3M0        DATA        0B2H
P5M1        DATA        0C9H
P5M0        DATA        0CAH

            ORG         0000H
            LJMP        MAIN
            ORG         0063H
            LJMP        TM2ISR

            ORG         0100H
TM2ISR:
            CPL         P1.0                ;测试端口
            ANL         AUXINTIF,#NOT T2IF  ;清中断标志
            RETI

MAIN:
            MOV         SP, #5FH
            MOV         P1M0, #00H
            MOV         P1M1, #00H
            MOV         P3M0, #00H
            MOV         P3M1, #00H
            MOV         P5M0, #00H
            MOV         P5M1, #00H

            MOV         T2L,#66H            ;65536-11.0592M/12/1000
            MOV         T2H,#0FCH
            MOV         AUXR,#10H           ;启动定时器
            MOV         IE2,#ET2            ;使能定时器中断
            SETB        EA

            JMP         $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr         T2L         =   0xd7;
sfr         T2H         =   0xd6;
sfr         AUXR        =   0x8e;
sfr         IE2         =   0xaf;
#define     ET2             0x04
sfr         AUXINTIF    =   0xef;
#define     T2IF            0x01

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        P10         =   P1^0;
```

```
void TM2_Isr() interrupt 12
{
    P10 = !P10;                                 //测试端口
    AUXINTIF &= ~T2IF;                          //清中断标志
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T2L = 0x66;                                 //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;                                //启动定时器
    IE2 = ET2;                                  //使能定时器中断
    EA = 1;

    while (1);
}
```

11.5.11 UART1 中断

汇编代码

```
;测试工作频率为11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART1ISR

            ORG     0100H
UART1ISR:
            JNB     TI,CHECKRI
            CLR     TI                          ;清中断标志
            CPL     P1.0                        ;测试端口
CHECKRI:
            JNB     RI,ISREXIT
            CLR     RI                          ;清中断标志
            CPL     P1.1                        ;测试端口
ISREXIT:
            RETI

MAIN:
```

```
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     SCON,#50H
        MOV     T2L,#0E8H       ;65536-11059200/115200/4=0FFE8H
        MOV     T2H,#0FFH
        MOV     AUXR,#15H       ;启动定时器
        SETB    ES              ;使能串口中断
        SETB    EA
        MOV     SBUF,#5AH       ;发送测试数据

        JMP     $

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

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

void UART1_Isr() interrupt 4
{
    if (TI)
    {
        TI = 0;                 //清中断标志
        P10 = !P10;             //测试端口
    }
    if (RI)
    {
        RI = 0;                 //清中断标志
        P11 = !P11;             //测试端口
    }
}

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

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

    SCON = 0x50;
    T2L = 0xe8;                         //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x15;                        //启动定时器
    ES = 1;                             //使能串口中断
    EA = 1;
    SBUF = 0x5a;                        //发送测试数据

    while (1);
}
```

11.5.12 UART2 中断

汇编代码

```
;测试工作频率为11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
S2CON       DATA    9AH
S2BUF       DATA    9BH
IE2         DATA    0AFH
ES2         EQU     01H

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0043H
            LJMP    UART2ISR

            ORG     0100H
UART2ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     A,S2CON
            JNB     ACC.1,CHECKRI
            ANL     S2CON,#NOT 02H      ;清中断标志
            CPL     P1.2                ;测试端口
CHECKRI:
            MOV     A,S2CON
            JNB     ACC.0,ISREXIT
            ANL     S2CON,#NOT 01H      ;清中断标志
            CPL     P1.3                ;测试端口
ISREXIT:
            POP     PSW
            POP     ACC
            RETI
```

```
MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     S2CON,#10H
        MOV     T2L,#0E8H       ;65536-11059200/115200/4=0FFE8H
        MOV     T2H,#0FFH
        MOV     AUXR,#14H       ;启动定时器
        MOV     IE2,#ES2        ;使能串口中断
        SETB    EA
        MOV     S2BUF,#5AH      ;发送测试数据

        JMP     $

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     S2CON   =   0x9a;
sfr     S2BUF   =   0x9b;
sfr     IE2     =   0xaf;
#define ES2         0x01

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P12     =   P1^2;
sbit    P13     =   P1^3;

void UART2_Isr() interrupt 8
{
    if (S2CON & 0x02)
    {
        S2CON &= ~0x02;                 //清中断标志
        P12 = !P12;                     //测试端口
    }
    if (S2CON & 0x01)
    {
        S2CON &= ~0x01;                 //清中断标志
        P13 = !P13;                     //测试端口
    }
```

```
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    S2CON = 0x10;
    T2L = 0xe8;                     //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x14;                    //启动定时器
    IE2 = ES2;                      //使能串口中断
    EA = 1;
    S2BUF = 0x5a;                   //发送测试数据

    while (1);
}
```

11.5.13 ADC中断

汇编代码

```
;测试工作频率为11.0592MHz

ADC_CONTR   DATA    0BCH
ADC_RES     DATA    0BDH
ADC_RESL    DATA    0BEH
ADCCFG      DATA    0DEH
EADC        BIT     IE.5

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     002BH
            LJMP    ADCISR

            ORG     0100H
ADCISR:
            ANL     ADC_CONTR,#NOT 20H  ;清中断标志
            MOV     P0,ADC_RES          ;测试端口
            MOV     P2,ADC_RESL         ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
```

```
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     ADCCFG,#00H
        MOV     ADC_CONTR,#0C0H     ;使能并启动 ADC 模块
        SETB    EADC                ;使能 ADC 中断
        SETB    EA

        JMP     $

        END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr     ADC_CONTR   =   0xbc;
sfr     ADC_RES     =   0xbd;
sfr     ADC_RESL    =   0xbe;
sfr     ADCCFG      =   0xde;
sbit    EADC        =   IE^5;

sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

void ADC_Isr() interrupt 5
{
    ADC_CONTR &= ~0x20;                 //清中断标志
    P0 = ADC_RES;                       //测试端口
    P2 = ADC_RESL;                      //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    ADCCFG = 0x00;
    ADC_CONTR = 0xc0;                   //使能并启动 ADC 模块
    EADC = 1;                           //使能 ADC 中断
    EA = 1;

    while (1);
}
```

11.5.14 LVD中断

汇编代码

```
;测试工作频率为11.0592MHz

RSTCFG      DATA    0FFH
ENLVR       EQU     40H             ;RSTCFG.6
LVD2V2      EQU     00H             ;LVD@2.2V
LVD2V4      EQU     01H             ;LVD@2.4V
LVD2V7      EQU     02H             ;LVD@2.7V
LVD3V0      EQU     03H             ;LVD@3.0V
ELVD        BIT     IE.6
LVDF        EQU     20H             ;PCON.5

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0033H
            LJMP    LVDISR

            ORG     0100H
LVDISR:
            ANL     PCON,#NOT LVDF  ;清中断标志
            CPL     P1.0            ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            ANL     PCON,#NOT LVDF  ;上电需要清中断标志
            MOV     RSTCFG,# LVD3V0 ;设置 LVD 电压为3.0V
            SETB    ELVD            ;使能 LVD 中断
            SETB    EA
            JMP     $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     RSTCFG      =   0xff;
#define ENLVR           0x40            //RSTCFG.6
```

```
#define   LVD2V2          0x00                    //LVD@2.2V
#define   LVD2V4          0x01                    //LVD@2.4V
#define   LVD2V7          0x02                    //LVD@2.7V
#define   LVD3V0          0x03                    //LVD@3.0V
sbit      ELVD      =     IE^6;
#define   LVDF            0x20                    //PCON.5

sfr       P1M1      =     0x91;
sfr       P1M0      =     0x92;
sfr       P3M1      =     0xb1;
sfr       P3M0      =     0xb2;
sfr       P5M1      =     0xc9;
sfr       P5M0      =     0xca;
sbit      P10       =     P1^0;

void LVD_Isr() interrupt 6
{
    PCON &= ~LVDF;                                //清中断标志
    P10 = !P10;                                   //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    PCON &= ~LVDF;                                //上电需要清中断标志
    RSTCFG = LVD3V0;                              //设置 LVD 电压为3.0V
    ELVD = 1;//使能 LVD 中断
    EA = 1;

    while (1);
}
```

11.5.15 PCA中断

汇编代码

```
;测试工作频率为11.0592MHz

CCON        DATA        0D8H
CF          BIT         CCON.7
CR          BIT         CCON.6
CCF3        BIT         CCON.3
CCF2        BIT         CCON.2
CCF1        BIT         CCON.1
CCF0        BIT         CCON.0
CMOD        DATA        0D9H
CL          DATA        0E9H
CH          DATA        0F9H
CCAPM0      DATA        0DAH
CCAP0L      DATA        0EAH
CCAP0H      DATA        0FAH
PCA_PWM0    DATA        0F2H
CCAPM1      DATA        0DBH
```

```
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H
CCAPM3      DATA    0DDH
CCAP3L      DATA    0EDH
CCAP3H      DATA    0FDH
PCA_PWM3    DATA    0F5H

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

            ORG     0100H
PCAISR:
            JNB     CF,ISREXIT
            CLR     CF              ;清中断标志
            CPL     P1.0            ;测试端口
ISREXIT:
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     CCON,#00H
            MOV     CMOD,#09H       ;PCA 时钟为系统时钟,使能 PCA 计时中断
            SETB    CR              ;启动 PCA 计时器
            SETB    EA

            JMP     $

            END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr     CCON    =   0xd8;
sbit    CF      =   CCON^7;
```

```
sbit    CR      =   CCON^6;
sbit    CCF3    =   CCON^3;
sbit    CCF2    =   CCON^2;
sbit    CCF1    =   CCON^1;
sbit    CCF0    =   CCON^0;
sfr     CMOD    =   0xd9;
sfr     CL      =   0xe9;
sfr     CH      =   0xf9;
sfr     CCAPM0  =   0xda;
sfr     CCAP0L  =   0xea;
sfr     CCAP0H  =   0xfa;
sfr     PCA_PWM0=   0xf2;
sfr     CCAPM1  =   0xdb;
sfr     CCAP1L  =   0xeb;
sfr     CCAP1H  =   0xfb;
sfr     PCA_PWM1=   0xf3;
sfr     CCAPM2  =   0xdc;
sfr     CCAP2L  =   0xec;
sfr     CCAP2H  =   0xfc;
sfr     PCA_PWM2=   0xf4;
sfr     CCAPM3  =   0xdd;
sfr     CCAP3L  =   0xed;
sfr     CCAP3H  =   0xfd;
sfr     PCA_PWM3=   0xf5;

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void PCA_Isr() interrupt 7
{
    if (CF)
    {
        CF = 0;                 //清中断标志
        P10 = !P10;             //测试端口
    }
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x09;                //PCA 时钟为系统时钟,使能 PCA 计时中断
    CR = 1;                     //启动 PCA 计时器
    EA = 1;

    while (1);
}
```

11.5.16 SPI中断

汇编代码

```
;测试工作频率为11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            MOV     SPSTAT,#0C0H        ;清中断标志
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     SPCTL,#50H          ;使能 SPI 主机模式
            MOV     SPSTAT,#0C0H        ;清中断标志
            MOV     IE2,#ESPI           ;使能 SPI 中断
            SETB    EA
            MOV     SPDAT,#5AH          ;发送测试数据

            JMP     $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     SPSTAT      =   0xcd;
sfr     SPCTL       =   0xce;
```

```
sfr     SPDAT   =   0xcf;
sfr     IE2     =   0xaf;
#define ESPI        0x02

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                  //清中断标志
    P10 = !P10;                     //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    SPCTL = 0x50;                   //使能 SPI 主机模式
    SPSTAT = 0xc0;                  //清中断标志
    IE2 = ESPI;                     //使能 SPI 中断
    EA = 1;
    SPDAT = 0x5a;                   //发送测试数据

    while (1);
}
```


11.5.17 CMP中断

汇编代码

```
;测试工作频率为 11.0592MHz

CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00ABH
            LJMP    CMPISR

            ORG     0100H
```

```
CMPISR:
        ANL     CMPCR1,#NOT 40H     ;清中断标志
        CPL     P1.0                ;测试端口
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     CMPCR2,#00H
        MOV     CMPCR1,#80H         ;使能比较器模块
        ORL     CMPCR1,#30H         ;使能比较器边沿中断
        ANL     CMPCR1,#NOT 08H     ;P3.6 为CMP+输入脚
        ORL     CMPCR1,#04H         ;P3.7 为CMP-输入脚
        ORL     CMPCR1,#02H         ;使能比较器输出
        SETB    EA

        JMP     $

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void CMP_Isr() interrupt 21
{
    CMPCR1 &= ~0x40;                //清中断标志
    P10 = !P10;                     //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;
```

```
    CMPCR2 = 0x00;
    CMPCR1 = 0x80;                  //使能比较器模块
    CMPCR1 |= 0x30;                 //使能比较器边沿中断
    CMPCR1 &= ~0x08;                //P3.6 为CMP+输入脚
    CMPCR1 |= 0x04;                 //P3.7 为CMP-输入脚
    CMPCR1 |= 0x02;                 //使能比较器输出
    EA = 1;

    while (1);
}
```

11.5.18 I2C中断

汇编代码

```
;测试工作频率为11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00C3H
            LJMP    I2CISR

            ORG     0100H
I2CISR:
            PUSH    ACC
            PUSH    DPL
            PUSH    DPH
            PUSH    P_SW2
            MOV     P_SW2,#80H
            MOV     DPTR,#I2CMSST
            MOVX    A,@DPTR
            ANL     A,#NOT 40H      ;清中断标志
            MOVX    @DPTR,A
            CPL     P1.0            ;测试端口
            POP     P_SW2
            POP     DPH
            POP     DPL
            POP     ACC
            RETI
```

```
MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#80H
        MOV     A,#0C0H             ;使能 I2C 主机模式
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#80H              ;使能 I2C 中断
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        MOV     P_SW2,#00H
        SETB    EA

        MOV     P_SW2,#80H
        MOV     A,#081H             ;发送起始命令
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        MOV     P_SW2,#00H

        JMP     $

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#d
efine   I2CRXD      (*(unsigned char volatile xdata *)0xfe87)

sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;
sbit    P10         =   P1^0;

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
```

```
    P_SW2 |= 0x80;
    if (I2CMSST & 0x40)
    {
        I2CMSST &= ~0x40;                   //清中断标志
        P10 = !P10;                         //测试端口
    }
    _pop_(P_SW2);
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;
    I2CCFG = 0xc0;                          //使能 I2C 主机模式
    I2CMSCR = 0x80;                         //使能 I2C 中断;
    P_SW2 = 0x00;
    EA = 1;

    P_SW2 = 0x80;
    I2CMSCR = 0x81;                         //发送起始命令
    P_SW2 = 0x00;

    while (1);
}
```

12 定时器/计数器

STC8G 系列单片机内部设置了 3 个 16 位定时器/计数器。3 个 16 位定时器 T0、T1 和 T2 都具有计数方式和定时方式两种工作方式。对定时器/计数器 T0 和 T1，用它们在特殊功能寄存器 TMOD 中相对应的控制位 C/T 来选择 T0 或 T1 为定时器还是计数器。对定时器/计数器 T2，用特殊功能寄存器 AUXR 中的控制位 T2_C/T 来选择 T2 为定时器还是计数器。定时器/计数器的核心部件是一个加法计数器，其本质是对脉冲进行计数。只是计数脉冲来源不同：如果计数脉冲来自系统时钟，则为定时方式，此时定时器/计数器每 12 个时钟或者每 1 个时钟得到一个计数脉冲，计数值加 1；如果计数脉冲来自单片机外部引脚（T0 为 P3.4，T1 为 P3.5，T2 为 P1.2），则为计数方式，每来一个脉冲加 1。

当定时器/计数器 T0、T1 及 T2 工作在定时模式时，特殊功能寄存器 AUXR 中的 T0x12、T1x12 和 T2x12 分别决定是系统时钟/12 还是系统时钟/1（不分频）后让 T0、T1 和 T2 进行计数。当定时器/计数器工作在计数模式时，对外部脉冲计数不分频。

定时器/计数器 0 有 4 种工作模式：模式 0（16 位自动重装载模式），模式 1（16 位不可重装载模式），模式 2（8 位自动重装模式），模式 3（不可屏蔽中断的 16 位自动重装载模式）。定时器/计数器 1 除模式 3 外，其他工作模式与定时器/计数器 0 相同。T1 在模式 3 时无效，停止计数。定时器 T2 的工作模式固定为 16 位自动重装载模式。T2 可以当定时器使用，也可以当串口的波特率发生器和可编程时钟输出。

12.1 定时器的相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
TMOD	定时器模式寄存器	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000,0000
TL0	定时器 0 低 8 为寄存器	8AH									0000,0000
TL1	定时器 1 低 8 为寄存器	8BH									0000,0000
TH0	定时器 0 高 8 为寄存器	8CH									0000,0000
TH1	定时器 1 高 8 为寄存器	8DH									0000,0000
AUXR	辅助寄存器 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
INTCLKO	中断与时钟输出控制寄存器	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
WKTCL	掉电唤醒定时器低字节	AAH									1111,1111
WKTCH	掉电唤醒定时器高字节	ABH	WKTEN								0111,1111
T2H	定时器 2 高字节	D6H									0000,0000
T2L	定时器 2 低字节	D7H									0000,0000

12.2 定时器 0/1

定时器 0/1 控制寄存器

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

TF1：T1溢出中断标志。T1被允许计数以后，从初值开始加1计数。当产生溢出时由硬件将TF1位置“1”，并向CPU请求中断，一直保持到CPU响应中断时，才由硬件清“0”（也可由查询软件清“0”）。

TR1：定时器T1的运行控制位。该位由软件置位和清零。当GATE（TMOD.7）=0，TR1=1时就允许T1开始计数，TR1=0时禁止T1计数。当GATE（TMOD.7）=1，TR1=1且INT1输入高电平时，才允许T1计数。

TF0：T0溢出中断标志。T0被允许计数以后，从初值开始加1计数，当产生溢出时，由硬件置“1”TF0，向CPU请求中断，一直保持CPU响应该中断时，才由硬件清0（也可由查询软件清0）。

TR0：定时器T0的运行控制位。该位由软件置位和清零。当GATE（TMOD.3）=0，TR0=1时就允许T0开始计数，TR0=0时禁止T0计数。当GATE（TMOD.3）=1，TR0=1且INT0输入高电平时，才允许T0计数，TR0=0时禁止T0计数。

IE1：外部中断1请求源（INT1/P3.3）标志。IE1=1，外部中断向CPU请求中断，当CPU响应该中断时由硬件清“0”IE1。

IT1：外部中断源1触发控制位。IT1=0，上升沿或下降沿均可触发外部中断1。IT1=1，外部中断1程控为下降沿触发方式。

IE0：外部中断0请求源（INT0/P3.2）标志。IE0=1外部中断0向CPU请求中断，当CPU响应外部中断时，由硬件清“0”IE0（边沿触发方式）。

IT0：外部中断源0触发控制位。IT0=0，上升沿或下降沿均可触发外部中断0。IT0=1，外部中断0程控为下降沿触发方式。

定时器 0/1 模式寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TMOD	89H	T1_GATE	T1_C/T	T1_M1	T1_M0	T0_GATE	T0_C/T	T0_M1	T0_M0

T1_GATE：控制定时器1，置1时只有在INT1脚为高及TR1控制位置1时才可打开定时器/计数器1。

T0_GATE：控制定时器0，置1时只有在INT0脚为高及TR0控制位置1时才可打开定时器/计数器0。

T1_C/T：控制定时器1用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T1/P3.5外部脉冲进行计数）。

T0_C/T：控制定时器0用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T0/P3.4外部脉冲进行计数）。

T1_M1/T1_M0：定时器定时器/计数器1模式选择

T1_M1	T1_M0	定时器/计数器1工作模式
0	0	16位自动重载模式 当[TH1,TL1]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[TH1,TL1]中。

0	1	16位不自动重载模式 当[TH1,TL1]中的16位计数值溢出时，定时器1将从0开始计数
1	0	8位自动重载模式 当TL1中的8位计数值溢出时，系统会自动将TH1中的重载值装入TL1中。
1	1	T1停止工作

T0_M1/T0_M0：定时器定时器/计数器0模式选择

T0_M1	T0_M0	定时器/计数器0工作模式
0	0	16位自动重载模式 当[TH0,TL0]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[TH0,TL0]中。
0	1	16位不自动重载模式 当[TH0,TL0]中的16位计数值溢出时，定时器0将从0开始计数
1	0	8位自动重载模式 当TL0中的8位计数值溢出时，系统会自动将TH0中的重载值装入TL0中。
1	1	16位自动重载模式 与模式0相同，产生不可屏蔽中断

定时器 0 计数寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TL0	8AH								
TH0	8CH								

当定时器/计数器0工作在16位模式（模式0、模式1、模式3）时，TL0和TH0组合成为一个16位寄存器，TL0为低字节，TH0为高字节。若为8位模式（模式2）时，TL0和TH0为两个独立的8位寄存器。

定时器 1 计数寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TL1	8BH								
TH1	8DH								

当定时器/计数器1工作在16位模式（模式0、模式1）时，TL1和TH1组合成为一个16位寄存器，TL1为低字节，TH1为高字节。若为8位模式（模式2）时，TL1和TH1为两个独立的8位寄存器。

辅助寄存器 1（AUXR）

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

T0x12：定时器0速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

T1x12：定时器1速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

中断与时钟输出控制寄存器（INTCLKO）

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

T0CLKO：定时器0时钟输出控制

0：关闭时钟输出

1：使能 P3.5 口的是定时器 0 时钟输出功能

当定时器 0 计数发生溢出时，P3.5 口的电平自动发生翻转。

T1CLKO：定时器1时钟输出控制

0：关闭时钟输出

1：使能 P3.4 口的是定时器 1 时钟输出功能

当定时器 1 计数发生溢出时，P3.4 口的电平自动发生翻转。

12.3 定时器 2

辅助寄存器 1（AUXR）

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

TR2：定时器2的运行控制位

0：定时器 2 停止计数

1：定时器 2 开始计数

T2_C/T：控制定时器0用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T2/P1.2外部脉冲进行计数）。

T2x12：定时器2速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

中断与时钟输出控制寄存器（INTCLKO）

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

T2CLKO：定时器2时钟输出控制

0：关闭时钟输出

1：使能 P1.3 口的是定时器 2 时钟输出功能

当定时器 2 计数发生溢出时，P1.3 口的电平自动发生翻转。

定时器 2 计数寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T2L	D7H								
T2H	D6H								

定时器/计数器2的工作模式固定为16位重载模式，T2L和T2H组合成为一个16位寄存器，T2L为低字节，T2H为高字节。当[T2H,T2L]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[T2H,T2L]中。

12.4 掉电唤醒定时器

内部掉电唤醒定时器是一个 15 位的计数器（由{WKTCH[6:0],WKTCL[7:0]}组成 15 位）。用于唤醒处于掉电模式的 MCU。

掉电唤醒定时器计数寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
WKTCL	AAH								
WKTCH	ABH	WKTEN							

WKTEN：掉电唤醒定时器的使能控制位

0：停用掉电唤醒定时器

1：启用掉电唤醒定时器

如果 STC8 系列单片机内置掉电唤醒专用定时器被允许（通过软件将 WKTCH 寄存器中的 WKTEN 位置 1），当 MCU 进入掉电模式/停机模式后，掉电唤醒专用定时器开始计数，当计数值与用户所设置的值相等时，掉电唤醒专用定时器将 MCU 唤醒。MCU 唤醒后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行。掉电唤醒之后，可以通过读 WKTCH 和 WKTCL 中的内容获取单片机在掉电模式中的睡眠时间。

这里请注意：用户在寄存器{WKTCH[6:0],WKTCL[7:0]}中写入的值必须比实际计数值少 1。如用户需计数 10 次，则将 9 写入寄存器{WKTCH[6:0],WKTCL[7:0]}中。同样，如果用户需计数 32768 次，则应对{WKTCH[6:0],WKTCL[7:0]}写入 7FFFH（即 32767）。

内部掉电唤醒定时器有自己的内部时钟，其中掉电唤醒定时器计数一次的时间就是由该时钟决定的。内部掉电唤醒定时器的时钟频率约为 32KHz，当然误差较大。用户可以通过读 RAM 区 F8H 和 F9H 的内容（F8H 存放频率的高字节，F9H 存放低字节）来获取内部掉电唤醒专用定时器出厂时所记录的时钟频率。

掉电唤醒专用定时器计数时间的计算公式如下所示：（F_{wt} 为我们从 RAM 区 F8H 和 F9H 获取到的内部掉电唤醒专用定时器的时钟频率）

$$\text{掉电唤醒定时器定时时间} = \frac{10^6 \times 16 \times \text{计数次数}}{F_{wt}} \text{(微秒)}$$

假设 F_{wt}=32KHz，则有：

{WKTCH[6:0],WKTCL[7:0]}	掉电唤醒专用定时器计数时间
0	$10^6 \div 32K \times 16 \times (1+0) \approx 0.5$ 毫秒
9	$10^6 \div 32K \times 16 \times (1+9) \approx 5$ 毫秒
99	$10^6 \div 32K \times 16 \times (1+99) \approx 50$ 毫秒
999	$10^6 \div 32K \times 16 \times (1+999) \approx 0.5$ 秒
4095	$10^6 \div 32K \times 16 \times (1+4095) \approx 2$ 秒
32767	$10^6 \div 32K \times 16 \times (1+32767) \approx 16$ 秒

12.5 范例程序

12.5.1 定时器 0（模式 0－16 位自动重载）

汇编代码

```
;测试工作频率为11.0592MHz

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     000BH
            LJMP    TM0ISR

            ORG     0100H
TM0ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     TMOD,#00H           ;模式0
            MOV     TL0,#66H            ;65536-11.0592M/12/1000
            MOV     TH0,#0FCH
            SETB    TR0                 ;启动定时器
            SETB    ET0                 ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;
```

```
sbit        P10             =   P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;                                //模式 0
    TL0 = 0x66;                                 //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                    //启动定时器
    ET0 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

12.5.2 定时器 0（模式 1－16 位不自动重载）

汇编代码

```
;测试工作频率为 11.0592MHz

P1M1        DATA        091H
P1M0        DATA        092H
P3M1        DATA        0B1H
P3M0        DATA        0B2H
P5M1        DATA        0C9H
P5M0        DATA        0CAH

            ORG         0000H
            LJMP        MAIN
            ORG         000BH
            LJMP        TM0ISR

            ORG         0100H
TM0ISR:
            MOV         TL0,#66H            ;重设定时参数
            MOV         TH0,#0FCH
            CPL         P1.0                ;测试端口
            RETI

MAIN:
            MOV         SP, #5FH
            MOV         P1M0, #00H
            MOV         P1M1, #00H
            MOV         P3M0, #00H
            MOV         P3M1, #00H
            MOV         P5M0, #00H
```

```
        MOV     P5M1, #00H

        MOV     TMOD,#01H       ;模式1
        MOV     TL0,#66H        ;65536-11.0592M/12/1000
        MOV     TH0,#0FCH
        SETB    TR0             ;启动定时器
        SETB    ET0             ;使能定时器中断
        SETB    EA

        JMP     $

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM0_Isr() interrupt 1
{
    TL0 = 0x66;                 //重设定时参数
    TH0 = 0xfc;
    P10 = !P10;                 //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x01;                //模式1
    TL0 = 0x66;                 //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                    //启动定时器
    ET0 = 1;                    //使能定时器中断
    EA = 1;

    while (1);
}
```

12.5.3 定时器 0（模式 2—8 位自动重载）

汇编代码

```
;测试工作频率为11.0592MHz

P1M1        DATA        091H
P1M0        DATA        092H
P3M1        DATA        0B1H
P3M0        DATA        0B2H
P5M1        DATA        0C9H
P5M0        DATA        0CAH

            ORG         0000H
            LJMP        MAIN
            ORG         000BH
            LJMP        TM0ISR

            ORG         0100H
TM0ISR:
            CPL         P1.0                ;测试端口
            RETI

MAIN:
            MOV         SP, #5FH
            MOV         P1M0, #00H
            MOV         P1M1, #00H
            MOV         P3M0, #00H
            MOV         P3M1, #00H
            MOV         P5M0, #00H
            MOV         P5M1, #00H

            MOV         TMOD,#02H           ;模式2
            MOV         TL0,#0F4H           ;256-11.0592M/12/76K
            MOV         TH0,#0F4H
            SETB        TR0                 ;启动定时器
            SETB        ET0                 ;使能定时器中断
            SETB        EA

            JMP         $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr         P1M1        =       0x91;
sfr         P1M0        =       0x92;
sfr         P3M1        =       0xb1;
sfr         P3M0        =       0xb2;
sfr         P5M1        =       0xc9;
sfr         P5M0        =       0xca;

sbit        P10         =       P1^0;
```

```
void TM0_Isr() interrupt 1
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x02;                                //模式 2
    TL0 = 0xf4;                                 //256-11.0592M/12/76K
    TH0 = 0xf4;
    TR0 = 1;                                    //启动定时器
    ET0 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

12.5.4 定时器 0（模式 3－16 位自动重载不可屏蔽中断）

汇编代码

```
;测试工作频率为 11.0592MHz

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     000BH
            LJMP    TM0ISR

            ORG     0100H
TM0ISR:
            CPL     P1.0                    ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     TMOD,#03H               ;模式 3
            MOV     TL0,#66H                ;65536-11.0592M/12/1000
            MOV     TH0,#0FCH
```

```
            SETB    TR0                 ;启动定时器
            SETB    ET0                 ;使能定时器中断
;           SETB    EA                  ;不受 EA 控制

            JMP     $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;                         //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x03;                        //模式 3
    TL0 = 0x66;                         //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                            //启动定时器
    ET0 = 1;                            //使能定时器中断
//  EA = 1;                             //不受 EA 控制

    while (1);
}
```

12.5.5 定时器 0（外部计数－扩展T0 为外部下降沿中断）

汇编代码

```
;测试工作频率为11.0592MHz

P1M1    DATA    091H
P1M0    DATA    092H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
```

```
P5M1        DATA        0C9H
P5M0        DATA        0CAH

            ORG         0000H
            LJMP        MAIN
            ORG         000BH
            LJMP        TM0ISR

            ORG         0100H
TM0ISR:
            CPL         P1.0                ;测试端口
            RETI

MAIN:
            MOV         SP, #5FH
            MOV         P1M0, #00H
            MOV         P1M1, #00H
            MOV         P3M0, #00H
            MOV         P3M1, #00H
            MOV         P5M0, #00H
            MOV         P5M1, #00H

            MOV         TMOD,#04H           ;外部计数模式
            MOV         TL0,#0FFH
            MOV         TH0,#0FFH
            SETB        TR0                 ;启动定时器
            SETB        ET0                 ;使能定时器中断
            SETB        EA

            JMP         $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        P10         =   P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;                             //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
```

```
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x04;                    //外部计数模式
    TL0 = 0xff;
    TH0 = 0xff;
    TR0 = 1;                        //启动定时器
    ET0 = 1;                        //使能定时器中断
    EA = 1;

    while (1);
}
```

12.5.6 定时器 0（测量脉宽－INT0 高电平宽度）

汇编代码

```
;测试工作频率为11.0592MHz

AUXR    DATA    8EH
P1M1    DATA    091H
P1M0    DATA    092H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
        ORG     0003H
        LJMP    INT0ISR

        ORG     0100H
INT0ISR:
        MOV     P0,TL0          ;TL0 为测量值低字节
        MOV     P1,TH0          ;TH0 为测量值低高字节
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     AUXR,#80H       ;1T 模式
        MOV     TMOD,#08H       ;使能 GATE,INT0 为 1 时使能计时
        MOV     TL0,#00H
        MOV     TH0,#00H
        JB      INT0,$          ;等待 INT0 为低
        SETB    TR0             ;启动定时器
        SETB    IT0             ;使能 INT0 下降沿中断
        SETB    EX0
        SETB    EA

        JMP     $
```

```
        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     AUXR    =   0x8e;

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void INT0_Isr() interrupt 0
{
    P0 = TL0; //TL0 为测量值低字节（会有约 10 个时钟的误差）
    P1 = TH0; //TH0 为测量值低高字节
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    AUXR = 0x80;                    //1T 模式
    TMOD = 0x08;                    //使能 GATE,INT0 为 1 时使能计时
    TL0 = 0x00;
    TH0 = 0x00;
    while (INT0);                   //等待 INT0 为低
    TR0 = 1;                        //启动定时器
    IT0 = 1;                        //使能 INT0 下降沿中断
    EX0 = 1;
    EA = 1;

    while (1);
}
```

12.5.7 定时器 0（时钟分频输出）

汇编代码

```
;测试工作频率为11.0592MHz

INTCLKO     DATA    8FH
P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
```

```
P5M1        DATA        0C9H
P5M0        DATA        0CAH

            ORG         0000H
            LJMP        MAIN

            ORG         0100H
MAIN:
            MOV         SP, #5FH
            MOV         P1M0, #00H
            MOV         P1M1, #00H
            MOV         P3M0, #00H
            MOV         P3M1, #00H
            MOV         P5M0, #00H
            MOV         P5M1, #00H

            MOV         TMOD,#00H           ;模式 0
            MOV         TL0,#66H            ;65536-11.0592M/12/1000
            MOV         TH0,#0FCH
            SETB        TR0                 ;启动定时器
            MOV         INTCLKO,#01H        ;使能时钟输出

            JMP         $

            END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr         INTCLKO     =   0x8f;

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;                            //模式 0
    TL0 = 0x66;                             //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                //启动定时器
    INTCLKO = 0x01;                         //使能时钟输出

    while (1);
}
```

12.5.8 定时器 1（模式 0－16 位自动重载）

汇编代码

```
;测试工作频率为11.0592MHz

P1M1        DATA        091H
P1M0        DATA        092H
P3M1        DATA        0B1H
P3M0        DATA        0B2H
P5M1        DATA        0C9H
P5M0        DATA        0CAH

            ORG         0000H
            LJMP        MAIN
            ORG         001BH
            LJMP        TM1ISR

            ORG         0100H
TM1ISR:
            CPL         P1.0                ;测试端口
            RETI

MAIN:
            MOV         SP, #5FH
            MOV         P1M0, #00H
            MOV         P1M1, #00H
            MOV         P3M0, #00H
            MOV         P3M1, #00H
            MOV         P5M0, #00H
            MOV         P5M1, #00H

            MOV         TMOD,#00H           ;模式0
            MOV         TL1,#66H            ;65536-11.0592M/12/1000
            MOV         TH1,#0FCH
            SETB        TR1                 ;启动定时器
            SETB        ET1                 ;使能定时器中断
            SETB        EA

            JMP         $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        P10         =   P1^0;
```

```
void TM1_Isr() interrupt 3
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;                                //模式 0
    TL1 = 0x66;                                 //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                                    //启动定时器
    ET1 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

12.5.9 定时器 1（模式 1—16 位不自动重载）

汇编代码

```
;测试工作频率为 11.0592MHz

P1M1    DATA    091H
P1M0    DATA    092H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
        ORG     001BH
        LJMP    TM1ISR

        ORG     0100H
TM1ISR:
        MOV     TL1,#66H                    ;重设定时参数
        MOV     TH1,#0FCH
        CPL     P1.0                        ;测试端口
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H
```

```
        MOV     TMOD,#10H           ;模式1
        MOV     TL1,#66H            ;65536-11.0592M/12/1000
        MOV     TH1,#0FCH
        SETB    TR1                 ;启动定时器
        SETB    ET1                 ;使能定时器中断
        SETB    EA

        JMP     $

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM1_Isr() interrupt 3
{
    TL1 = 0x66;                     //重设定时参数
    TH1 = 0xfc;
    P10 = !P10;                     //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x10;                    //模式1
    TL1 = 0x66;                     //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                        //启动定时器
    ET1 = 1;                        //使能定时器中断
    EA = 1;

    while (1);
}
```

12.5.10 定时器 1（模式 2—8 位自动重载）

汇编代码

```
;测试工作频率为11.0592MHz
```

```
P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     001BH
            LJMP    TM1ISR

            ORG     0100H
TM1ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     TMOD,#20H           ;模式2
            MOV     TL1,#0F4H           ;256-11.0592M/12/76K
            MOV     TH1,#0F4H
            SETB    TR1                 ;启动定时器
            SETB    ET1                 ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM1_Isr() interrupt 3
{
    P10 = !P10;                         //测试端口
}
```

```
void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x20;                    //模式 2
    TL1 = 0xf4;                     //256-11.0592M/12/76K
    TH1 = 0xf4;
    TR1 = 1;                        //启动定时器
    ET1 = 1;                        //使能定时器中断
    EA = 1;

    while (1);
}
```

12.5.11 定时器 1（外部计数—扩展T1 为外部下降沿中断）

汇编代码

```
;测试工作频率为 11.0592MHz

P1M1    DATA    091H
P1M0    DATA    092H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
        ORG     001BH
        LJMP    TM1ISR

        ORG     0100H
TM1ISR:
        CPL     P1.0            ;测试端口
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     TMOD,#40H       ;外部计数模式
        MOV     TL1,#0FFH
        MOV     TH1,#0FFH
        SETB    TR1             ;启动定时器
        SETB    ET1             ;使能定时器中断
        SETB    EA

        JMP     $
```

```
        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM1_Isr() interrupt 3
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x40;                                //外部计数模式
    TL1 = 0xff;
    TH1 = 0xff;
    TR1 = 1;                                    //启动定时器
    ET1 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

12.5.12 定时器 1（测量脉宽－INT1 高电平宽度）

汇编代码

```
;测试工作频率为11.0592MHz

AUXR    DATA    8EH
P1M1    DATA    091H
P1M0    DATA    092H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
```

```
            LJMP    MAIN
            ORG     0013H
            LJMP    INT1ISR

            ORG     0100H
INT1ISR:
            MOV     P0,TL1              ;TL1 为测量值低字节
            MOV     P1,TH1              ;TH1 为测量值低高字节
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     AUXR,#40H           ;1T 模式
            MOV     TMOD,#80H           ;使能 GATE,INT1 为 1 时使能计时
            MOV     TL1,#00H
            MOV     TH1,#00H
            JB      INT1,$              ;等待 INT1 为低
            SETB    TR1                 ;启动定时器
            SETB    IT1                 ;使能 INT1 下降沿中断
            SETB    EX1
            SETB    EA

            JMP     $

            END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sfr         AUXR        =   0x8e;

void INT1_Isr() interrupt 2
{
    P0 = TL1; //TL1 为测量值低字节（会有约 11 个时钟的误差）
    P1 = TH1; //TH1 为测量值低高字节
}

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

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

    AUXR = 0x40;                    //1T 模式
    TMOD = 0x80;                    //使能 GATE,INT1 为1 时使能计时
    TL1 = 0x00;
    TH1 = 0x00;
    while (INT1);                   //等待 INT1 为低
    TR1 = 1;                        //启动定时器
    IT1 = 1;                        //使能 INT1 下降沿中断
    EX1 = 1;
    EA = 1;

    while (1);
}
```

12.5.13 定时器 1（时钟分频输出）

汇编代码

```
;测试工作频率为 11.0592MHz

INTCLKO     DATA    8FH
P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     TMOD,#00H       ;模式 0
            MOV     TL1,#66H        ;65536-11.0592M/12/1000
            MOV     TH1,#0FCH
            SETB    TR1             ;启动定时器
            MOV     INTCLKO,#02H    ;使能时钟输出

            JMP     $

            END
```

C 语言代码

```
//测试工作频率为 11.0592MHz
```

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

sfr         INTCLKO     =   0x8f;

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;                    //模式 0
    TL1 = 0x66;                     //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                        //启动定时器
    INTCLKO = 0x02;                 //使能时钟输出

    while (1);
}
```

12.5.14 定时器 1（模式 0）做串口 1 波特率发生器

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA        8EH

BUSY        BIT         20H.0
WPTR        DATA        21H
RPTR        DATA        22H
BUFFER      DATA        23H             ;16 bytes

P1M1        DATA        091H
P1M0        DATA        092H
P3M1        DATA        0B1H
P3M0        DATA        0B2H
P5M1        DATA        0C9H
P5M0        DATA        0CAH

            ORG         0000H
            LJMP        MAIN
            ORG         0023H
            LJMP        UART_ISR

            ORG         0100H

UART_ISR:
```

```
        PUSH    ACC
        PUSH    PSW
        MOV     PSW,#08H

        JNB     TI,CHKRI
        CLR     TI
        CLR     BUSY
CHKRI:
        JNB     RI,UARTISR_EXIT
        CLR     RI
        MOV     A,WPTR
        ANL     A,#0FH
        ADD     A,#BUFFER
        MOV     R0,A
        MOV     @R0,SBUF
        INC     WPTR
UARTISR_EXIT:
        POP     PSW
        POP     ACC
        RETI

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

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

UART_SENDSTR:
        CLR     A
        MOVC    A,@A+DPTR
        JZ      SENDEND
        LCALL   UART_SEND
        INC     DPTR
        JMP     UART_SENDSTR
SENDEND:
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        LCALL   UART_INIT
```

```
                SETB        ES
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

sfr         AUXR        =   0x8e;

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

bit         busy;
char        wptr;
char        rptr;
char        buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
```

```
            wptr &= 0x0f;
        }
}

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

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

void UartSendStr(char *p)
{
        while (*p)
        {
            UartSend(*p++);
        }
}

void main()
{
        P1M0 = 0x00;
        P1M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        UartInit();
        ES = 1;
        EA = 1;
        UartSendStr("Uart Test !\r\n");

        while (1)
        {
            if (rptr != wptr)
            {
                UartSend(buffer[rptr++]);
                rptr &= 0x0f;
            }
        }
}
```

12.5.15 定时器 1（模式 2）做串口 1 波特率发生器

汇编代码

```
;测试工作频率为11.0592MHz

AUXR        DATA    8EH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,SBUF
            INC     WPTR
UARTISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART_INIT:
            MOV     SCON,#50H
            MOV     TMOD,#20H
            MOV     TL1,#0FDH           ;256-11059200/115200/32=0FDH
            MOV     TH1,#0FDH
            SETB    TR1
            MOV     AUXR,#40H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
```

```
        RET

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

UART_SENDSTR:
        CLR     A
        MOVC    A,@A+DPTR
        JZ      SENDEND
        LCALL   UART_SEND
        INC     DPTR
        JMP     UART_SENDSTR
SENDEND:
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        LCALL   UART_INIT
        SETB    ES
        SETB    EA

        MOV     DPTR,#STRING
        LCALL   UART_SENDSTR

LOOP:
        MOV     A,RPTR
        XRL     A,WPTR
        ANL     A,#0FH
        JZ      LOOP
        MOV     A,RPTR
        ANL     A,#0FH
        ADD     A,#BUFFER
        MOV     R0,A
        MOV     A,@R0
        LCALL   UART_SEND
        INC     RPTR
        JMP     LOOP

STRING: DB      'Uart Test !',0DH,0AH,00H

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

```
#define   FOSC        11059200UL
#define   BRT         (256 - FOSC / 115200 / 32)

sfr       AUXR        =   0x8e;

sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x20;
    TL1 = BRT;
    TH1 = BRT;
    TR1 = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

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

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}
```

```
void main()
{
	P1M0 = 0x00;
	P1M1 = 0x00;
	P3M0 = 0x00;
	P3M1 = 0x00;
	P5M0 = 0x00;
	P5M1 = 0x00;

	UartInit();
	ES = 1;
	EA = 1;
	UartSendStr("Uart Test !\r\n");

	while (1)
	{
		if (rptr != wptr)
		{
			UartSend(buffer[rptr++]);
			rptr &= 0x0f;
		}
	}
}
```

12.5.16 定时器 2（16 位自动重载）

汇编代码

```
;测试工作频率为11.0592MHz

T2L		DATA	0D7H
T2H		DATA	0D6H
AUXR		DATA	8EH
IE2		DATA	0AFH
ET2		EQU	04H
AUXINTIF	DATA	0EFH
T2IF		EQU	01H

P1M1		DATA	091H
P1M0		DATA	092H
P3M1		DATA	0B1H
P3M0		DATA	0B2H
P5M1		DATA	0C9H
P5M0		DATA	0CAH

		ORG	0000H
		LJMP	MAIN
		ORG	0063H
		LJMP	TM2ISR

		ORG	0100H
TM2ISR:
		CPL	P1.0			;测试端口
		ANL	AUXINTIF,#NOT T2IF	;清中断标志
		RETI

MAIN:
		MOV	SP, #5FH
```

```
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     T2L,#66H                ;65536-11.0592M/12/1000
        MOV     T2H,#0FCH
        MOV     AUXR,#10H               ;启动定时器
        MOV     IE2,#ET2                ;使能定时器中断
        SETB    EA

        JMP     $

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2         0x04
sfr     AUXINTIF=   0xef;
#define T2IF        0x01

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM2_Isr() interrupt 12
{
    P10 = !P10;                         //测试端口
    AUXINTIF &= ~T2IF;                  //清中断标志
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T2L = 0x66;                         //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;                        //启动定时器
```

```
    IE2 = ET2;                                  //使能定时器中断
    EA = 1;

    while (1);
}
```

12.5.17 定时器 2（外部计数－扩展T2 为外部下降沿中断）

汇编代码

```
;测试工作频率为11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
IE2         DATA    0AFH
ET2         EQU     04H
AUXINTIF    DATA    0EFH
T2IF        EQU     01H

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0063H
            LJMP    TM2ISR

            ORG     0100H
TM2ISR:
            CPL     P1.0                ;测试端口
            ANL     AUXINTIF,#NOT T2IF  ;清中断标志
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     T2L,#0FFH
            MOV     T2H,#0FFH
            MOV     AUXR,#18H           ;设置外部计数模式并启动定时器
            MOV     IE2,#ET2            ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr         T2L             =   0xd7;
sfr         T2H             =   0xd6;
sfr         AUXR            =   0x8e;
sfr         IE2             =   0xaf;
#define     ET2                 0x04
sfr         AUXINTIF        =   0xef;
#define     T2IF                0x01

sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        P10             =   P1^0;

void TM2_Isr() interrupt 12
{
    P10 = !P10;                                 //测试端口
    AUXINTIF &= ~T2IF;                          //清中断标志
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T2L = 0xff;
    T2H = 0xff;
    AUXR = 0x18;                                //设置外部计数模式并启动定时器
    IE2 = ET2;                                  //使能定时器中断
    EA = 1;

    while (1);
}
```

12.5.18 定时器 2（时钟分频输出）

汇编代码

```
;测试工作频率为11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
INTCLKO     DATA    8FH
```

```
P1M1        DATA        091H
P1M0        DATA        092H
P3M1        DATA        0B1H
P3M0        DATA        0B2H
P5M1        DATA        0C9H
P5M0        DATA        0CAH

            ORG         0000H
            LJMP        MAIN

            ORG         0100H
MAIN:
            MOV         SP, #5FH
            MOV         P1M0, #00H
            MOV         P1M1, #00H
            MOV         P3M0, #00H
            MOV         P3M1, #00H
            MOV         P5M0, #00H
            MOV         P5M1, #00H

            MOV         T2L,#66H                ;65536-11.0592M/12/1000
            MOV         T2H,#0FCH
            MOV         AUXR,#10H               ;启动定时器
            MOV         INTCLKO,#04H            ;使能时钟输出

            JMP         $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr         T2L         =   0xd7;
sfr         T2H         =   0xd6;
sfr         AUXR        =   0x8e;
sfr         INTCLKO     =   0x8f;

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T2L = 0x66;                                 //65536-11.0592M/12/1000
    T2H = 0xfc;
```

```
    AUXR = 0x10;                                    //启动定时器
    INTCLKO = 0x04;                                 //使能时钟输出

    while (1);
}
```

12.5.19 定时器 2 做串口 1 波特率发生器

汇编代码

```
;测试工作频率为11.0592MHz

AUXR        DATA        8EH
T2H         DATA        0D6H
T2L         DATA        0D7H

BUSY        BIT         20H.0
WPTR        DATA        21H
RPTR        DATA        22H
BUFFER      DATA        23H                 ;16 bytes

P1M1        DATA        091H
P1M0        DATA        092H
P3M1        DATA        0B1H
P3M0        DATA        0B2H
P5M1        DATA        0C9H
P5M0        DATA        0CAH

            ORG         0000H
            LJMP        MAIN
            ORG         0023H
            LJMP        UART_ISR

            ORG         0100H

UART_ISR:
            PUSH        ACC
            PUSH        PSW
            MOV         PSW,#08H

            JNB         TI,CHKRI
            CLR         TI
            CLR         BUSY
CHKRI:
            JNB         RI,UARTISR_EXIT
            CLR         RI
            MOV         A,WPTR
            ANL         A,#0FH
            ADD         A,#BUFFER
            MOV         R0,A
            MOV         @R0,SBUF
            INC         WPTR
UARTISR_EXIT:
            POP         PSW
            POP         ACC
            RETI

UART_INIT:
            MOV         SCON,#50H
```

```
        MOV     T2L,#0E8H               ;65536-11059200/115200/4=0FFE8H
        MOV     T2H,#0FFH
        MOV     AUXR,#15H
        CLR     BUSY
        MOV     WPTR,#00H
        MOV     RPTR,#00H
        RET

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

UART_SENDSTR:
        CLR     A
        MOVC    A,@A+DPTR
        JZ      SENDEND
        LCALL   UART_SEND
        INC     DPTR
        JMP     UART_SENDSTR
SENDEND:
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        LCALL   UART_INIT
        SETB    ES
        SETB    EA

        MOV     DPTR,#STRING
        LCALL   UART_SENDSTR

LOOP:
        MOV     A,RPTR
        XRL     A,WPTR
        ANL     A,#0FH
        JZ      LOOP
        MOV     A,RPTR
        ANL     A,#0FH
        ADD     A,#BUFFER
        MOV     R0,A
        MOV     A,@R0
        LCALL   UART_SEND
        INC     RPTR
        JMP     LOOP

STRING: DB      'Uart Test !',0DH,0AH,00H

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

sfr        AUXR        =    0x8e;
sfr        T2H         =    0xd6;
sfr        T2L         =    0xd7;

sfr        P1M1        =    0x91;
sfr        P1M0        =    0x92;
sfr        P3M1        =    0xb1;
sfr        P3M0        =    0xb2;
sfr        P5M1        =    0xc9;
sfr        P5M0        =    0xca;

bit        busy;
char       wptr;
char       rptr;
char       buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x15;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

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

void UartSendStr(char *p)
```

```
{
    while (*p)
    {
        UartSend(*p++);
    }
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

12.5.20 定时器 2 做串口 2 波特率发生器

汇编代码

```
;测试工作频率为11.0592MHz

AUXR    DATA    8EH
T2H     DATA    0D6H
T2L     DATA    0D7H
S2CON   DATA    9AH
S2BUF   DATA    9BH
IE2     DATA    0AFH

BUSY    BIT     20H.0
WPTR    DATA    21H
RPTR    DATA    22H
BUFFER  DATA    23H                 ;16 bytes

P1M1    DATA    091H
P1M0    DATA    092H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
        ORG     0043H
```

```
            LJMP    UART2_ISR

            ORG     0100H

UART2_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S2CON
            JNB     ACC.1,CHKRI
            ANL     S2CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART2ISR_EXIT
            ANL     S2CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S2BUF
            INC     WPTR
UART2ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART2_INIT:
            MOV     S2CON,#10H
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART2_SEND:
            JB      BUSY,$
            SETB    BUSY
            MOV     S2BUF,A
            RET

UART2_SENDSTR:
            CLR     A
            MOVC    A,@A+DPTR
            JZ      SEND2END
            LCALL   UART2_SEND
            INC     DPTR
            JMP     UART2_SENDSTR
SEND2END:
            RET

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
```

```
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                LCALL       UART2_INIT
                MOV         IE2,#01H
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART2_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART2_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

sfr        AUXR         =    0x8e;
sfr        T2H          =    0xd6;
sfr        T2L          =    0xd7;
sfr        S2CON        =    0x9a;
sfr        S2BUF        =    0x9b;
sfr        IE2          =    0xaf;

sfr        P1M1         =    0x91;
sfr        P1M0         =    0x92;
sfr        P3M1         =    0xb1;
sfr        P3M0         =    0xb2;
sfr        P5M1         =    0xc9;
sfr        P5M0         =    0xca;

bit        busy;
char       wptr;
char       rptr;
char       buffer[16];

void Uart2Isr() interrupt 8
{
```

```
    if (S2CON & 0x02)
    {
        S2CON &= ~0x02;
        busy = 0;
    }
    if (S2CON & 0x01)
    {
        S2CON &= ~0x01;
        buffer[wptr++] = S2BUF;
        wptr &= 0x0f;
    }
}

void Uart2Init()
{
    S2CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart2Send(char dat)
{
    while (busy);
    busy = 1;
    S2BUF = dat;
}

void Uart2SendStr(char *p)
{
    while (*p)
    {
        Uart2Send(*p++);
    }
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart2Init();
    IE2 = 0x01;
    EA = 1;
    Uart2SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart2Send(buffer[rptr++]);
            rptr &= 0x0f;
```

```
            }
        }
}
```

13 串口通信

STC8G 系列单片机具有 2 个全双工异步串行通信接口(串口 1、串口 2)。每个串行口由 2 个数据缓冲器、一个移位寄存器、一个串行控制寄存器和一个波特率发生器等组成。每个串行口的数据缓冲器由 2 个互相独立的接收、发送缓冲器构成，可以同时发送和接收数据。

STC8G 系列单片机的串口 1 有 4 种工作方式，其中两种方式的波特率是可变的，另两种是固定的，以供不同应用场合选用。串口 2 只有两种工作方式，这两种方式的波特率都是可变的。用户可用软件设置不同的波特率和选择不同的工作方式。主机可通过查询或中断方式对接收/发送进行程序处理，使用十分灵活。

串口 1、串口 2 的通讯口均可以通过功能管脚的切换功能切换到多组端口，从而可以将一个通讯口分时复用为多个通讯口。

13.1 串口相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	串口 1 数据寄存器	99H									0000,0000
S2CON	串口 2 控制寄存器	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
S2BUF	串口 2 数据寄存器	9BH									0000,0000
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
AUXR	辅助寄存器 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
SADDR	串口 1 从机地址寄存器	A9H									0000,0000
SADEN	串口 1 从机地址屏蔽寄存器	B9H									0000,0000

13.2 串口 1

串口 1 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

SM0/FE：当PCON寄存器中的SMOD0位为1时，该位为帧错误检测标志位。当UART在接收过程中检测到一个无效停止位时，通过UART接收器将该位置1，必须由软件清零。当PCON寄存器中的SMOD0位为0时，该位和SM1一起指定串口1的通信工作模式，如下表所示：

SM0	SM1	串口1工作模式	功能说明
0	0	模式0	同步移位串行方式
0	1	模式1	可变波特率8位数据方式
1	0	模式2	固定波特率9位数据方式
1	1	模式3	可变波特率9位数据方式

SM2：允许模式 2 或模式 3 多机通信控制位。当串口 1 使用模式 2 或模式 3 时，如果 SM2 位为 1 且 REN 位为 1，则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位（即 RB8）来筛选地址帧，若 RB8=1，说明该帧是地址帧，地址信息可以进入 SBUF，并使 RI 为 1，进而在中断服务程序中再进行地址号比较；若 RB8=0，说明该帧不是地址帧，应丢掉且保持 RI=0。在模式 2 或模式 3 中，如果 SM2 位为 0 且 REN 位为 1，接收收机处于地址帧筛选被禁止状态，不论收到的 RB8 为 0 或 1，均可使接收到的信息进入 SBUF，并使 RI=1，此时 RB8 通常为校验位。模式 1 和模式 0 为非多机通信方式，在这两种方式时，SM2 应设置为 0。

REN：允许/禁止串口接收控制位

0：禁止串口接收数据

1：允许串口接收数据

TB8：当串口 1 使用模式 2 或模式 3 时，TB8 为要发送的第 9 位数据，按需要由软件置位或清 0。在模式 0 和模式 1 中，该位不用。

RB8：当串口 1 使用模式 2 或模式 3 时，RB8 为接收到的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位。在模式 0 和模式 1 中，该位不用。

TI：串口 1 发送中断请求标志位。在模式 0 中，当串口发送数据第 8 位结束时，由硬件自动将 TI 置 1，向主机请求中断，响应中断后 TI 必须用软件清零。在其他模式中，则在停止位开始发送时由硬件自动将 TI 置 1，向 CPU 发请求中断，响应中断后 TI 必须用软件清零。

RI：串口 1 接收中断请求标志位。在模式 0 中，当串口接收第 8 位数据结束时，由硬件自动将 RI 置 1，向主机请求中断，响应中断后 RI 必须用软件清零。在其他模式中，串行接收到停止位的中间时刻由硬件自动将 RI 置 1，向 CPU 发中断申请，响应中断后 RI 必须由软件清零。

串口 1 数据寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SBUF	99H								

SBUF：串口 1 数据接收/发送缓冲区。SBUF 实际是 2 个缓冲器，读缓冲器和写缓冲器，两个操作分别对应两个不同的寄存器，1 个是只写寄存器（写缓冲器），1 个是只读寄存器（读缓冲器）。对 SBUF 进行读操作，实际是读取串口接收缓冲区，对 SBUF 进行写操作则是触发串口开始发送数据。

电源管理寄存器

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

SMOD：串口 1 波特率控制位

0：串口 1 的各个模式的波特率都不加倍

1：串口 1 模式 1、模式 2、模式 3 的波特率加倍

SMOD0：帧错误检测控制位

0：无帧错检测功能

1：使能帧错误检测功能。此时 SCON 的 SM0/FE 为 FE 功能，即为帧错误检测标志位。

辅助寄存器 1

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

UART_M0x6：串口 1 模式 0 的通讯速度控制

0：串口 1 模式 0 的波特率不加倍，固定为 Fosc/12

1：串口 1 模式 0 的波特率 6 倍速，即固定为 Fosc/12*6 ＝ Fosc/2

S1ST2：串口 1 波特率发射器选择位

0：选择定时器 1 作为波特率发射器

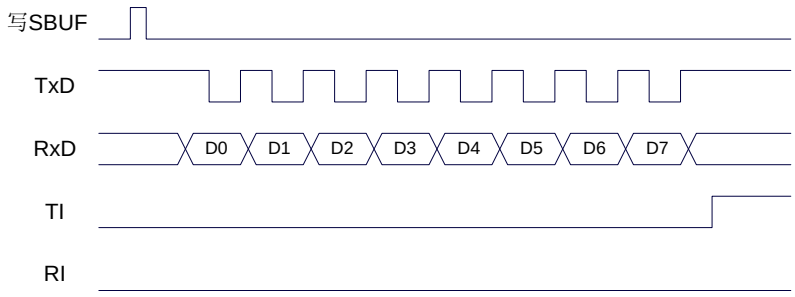
1：选择定时器 2 作为波特率发射器

13.2.1 串口 1 模式 0

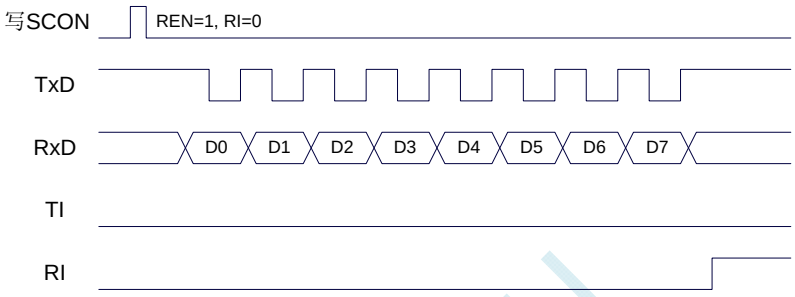
当串口 1 选择工作模式为模式 0 时，串行通信接口工作在同步移位寄存器模式，当串行口模式 0 的通信速度设置位 UART_M0x6 为 0 时，其波特率固定为系统时钟时钟的 12 分频（SYSclk/12）；当设置 UART_M0x6 为 1 时，其波特率固定为系统时钟频率的 2 分频（SYSclk/2）。RxD 为串行通讯的数据口，TxD 为同步移位脉冲输出脚，发送、接收的是 8 位数据，低位在先。

模式 0 的发送过程：当主机执行将数据写入发送缓冲器 SBUF 指令时启动发送，串行口即将 8 位数据以 SYSclk/12 或 SYSclk/2（由 UART_M0x6 确定是 12 分频还是 2 分频）的波特率从 RxD 管脚输出(从低位到高位)，发送完中断标志 TI 置 1，TxD 管脚输出同步移位脉冲信号。当写信号有效后，相隔一个时钟，发送控制端 SEND 有效(高电平)，允许 RxD 发送数据，同时允许 TxD 输出同步移位脉冲。一帧(8 位)数据发送完毕时，各控制端均恢复原状态，只有 TI 保持高电平，呈中断申请状态。在再次发送数据前，必须用软件将 TI 清 0。

模式 0 的接收过程：首先将接收中断请求标志 RI 清零并置位允许接收控制位 REN 时启动模式 0 接收过程。启动接收过程后，RxD 为串行数据输入端，TxD 为同步脉冲输出端。串行接收的波特率为 SYSclk/12 或 SYSclk/2（由 UART_M0x6 确定是 12 分频还是 2 分频）。当接收完成一帧数据（8 位）后，控制信号复位，中断标志 RI 被置 1，呈中断申请状态。当再次接收时，必须通过软件将 RI 清 0



发送数据（串口1模式0）



接收数据（串口1模式0）

工作于模式 0 时，必须清 0 多机通信控制位 SM2，使之不影响 TB8 位和 RB8 位。由于波特率固定为 SYSclk/12 或 SYSclk/2，无需定时器提供，直接由单片机的时钟作为同步移位脉冲。

串口 1 模式 0 的波特率计算公式如下表所示（SYSclk 为系统工作频率）：

UART_M0x6	波特率计算公式
0	波特率 $= \frac{\text{SYSclk}}{12}$
1	波特率 $= \frac{\text{SYSclk}}{2}$

13.2.2 串口 1 模式 1

当软件设置 SCON 的 SM0、SM1 为“01”时，串行口 1 则以模式 1 进行工作。此模式为 8 位 UART 格式，一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，即可根据需要进行设置波特率。TxD 为数据发送口，RxD 为数据接收口，串行口全双工接受/发送。

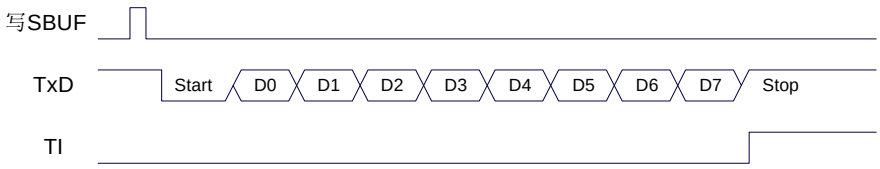
模式 1 的发送过程：串行通信模式发送时，数据由串行发送端 TxD 输出。当主机执行一条写 SBUF 的指令就启动串行通信的发送，写“SBUF”信号还把“1”装入发送移位寄存器的第 9 位，并通知 TX 控制单元开始发送。移位寄存器将数据不断右移送 TxD 端口发送，在数据的左边不断移入“0”作补充。当数据的最高位移到移位寄存器的输出位置，紧跟其后的是第 9 位“1”，在它的左边各位全为“0”，这个状态条件，使 TX 控制单元作最后一次移位输出，然后使允许发送信号“SEND”失效，完成一帧信息的发送，并置位中断请求位 TI，即 TI=1，向主机请求中断处理。

模式 1 的接收过程：当软件置位接收允许标志位 REN，即 REN=1 时，接收器便对 RxD 端口的信号进行检测，当检测到 RxD 端口发送从“1”→“0”的下降沿跳变时就启动接收器准备接收数据，并立即复位波特率发生器的接收计数器，将 1FFH 装入移位寄存器。接收的数据从接收移位寄存器的右边移入，已装入的 1FFH 向左边移出，当起始位"0"移到移位寄存器的最左边时，使 RX 控制器作最后一次移位，完成一帧的接收。若同时满足以下两个条件：

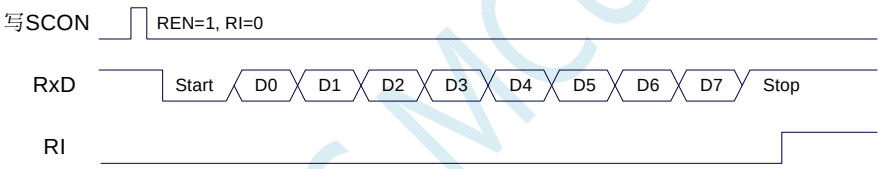
• RI=0；

• SM2=0 或接收到的停止位为 1。

则接收到的数据有效，实现装载入 SBUF，停止位进入 RB8，RI 标志位被置 1，向主机请求中断，若上述两条件不能同时满足，则接收到的数据作废并丢失，无论条件满足与否，接收器重又检测 RxD 端口上的"1"→"0"的跳变，继续下一帧的接收。接收有效，在响应中断后，RI 标志位必须由软件清 0。通常情况下，串行通信工作于模式 1 时，SM2 设置为"0"。



发送数据（串口1模式1）



接收数据（串口1模式1）

串口 1 的波特率是可变的，其波特率可由定时器 1 或者定时器 2 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

串口 1 模式 1 的波特率计算公式如下表所示：（SYSclk 为系统工作频率）

选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536 - \frac{SYSclk}{4\times 波特率}$
	12T	定时器2重载值 = $65536 - \frac{SYSclk}{12\times 4\times 波特率}$
定时器1模式0	1T	定时器1重载值 = $65536 - \frac{SYSclk}{4\times 波特率}$
	12T	定时器1重载值 = $65536 - \frac{SYSclk}{12\times 4\times 波特率}$
定时器1模式2	1T	定时器1重载值 = $256 - \frac{2^{SMOD}\times SYSclk}{32\times 波特率}$
	12T	定时器1重载值 = $256 - \frac{2^{SMOD}\times SYSclk}{12\times 32\times 波特率}$

下面为常用频率与常用波特率所对应定时器的重载值

频率 (MHz)	波特率	定时器 2		定时器 1 模式 0		定时器 1 模式 2			
						SMOD=1		SMOD=0	
		1T 模式	12T 模式	1T 模式	12T 模式	1T 模式	12T 模式	1T 模式	12T 模式
11.0592	115200	FFE8H	FFFEH	FFE8H	FFFEH	FAH	-	FDH	-
	57600	FFD0H	FFFCH	FFD0H	FFFCH	F4H	FFH	FAH	-
	38400	FFB8H	FFFAH	FFB8H	FFFAH	EEH	-	F7H	-
	19200	FF70H	FFF4H	FF70H	FFF4H	DCH	FDH	EEH	-
	9600	FEE0H	FFE8H	FEE0H	FFE8H	B8H	FAH	DCH	FDH
18.432	115200	FFD8H	-	FFD8H	-	F6H	-	FBH	-
	57600	FFB0H	-	FFB0H	-	ECH	-	F6H	-
	38400	FF88H	FFF6H	FF88H	FFF6H	E2H	-	F1H	-
	19200	FF10H	FFECH	FF10H	FFECH	C4H	FBH	E2H	-
	9600	FE20H	FFD8H	FE20H	FFD8H	88H	F6H	C4H	FBH
22.1184	115200	FFD0H	FFFCH	FFD0H	FFFCH	F4H	FFH	FAH	-
	57600	FFA0H	FFF8H	FFA0H	FFF8H	E8H	FEH	F4H	FFH
	38400	FF70H	FFF4H	FF70H	FFF4H	DCH	FDH	EEH	-
	19200	FEE0H	FFE8H	FEE0H	FFE8H	B8H	FAH	DCH	FDH
	9600	FDC0H	FFD0H	FDC0H	FFD0H	70H	F4H	B8H	FAH

13.2.3 串口 1 模式 2

当 SM0、SM1 两位为 10 时，串行口 1 工作在模式 2。串行口 1 工作模式 2 为 9 位数据异步通信 UART 模式，其一帧的信息由 11 位组成：1 位起始位，8 位数据位（低位在先），1 位可编程位（第 9 位数据）和 1 位停止位。发送时可编程位（第 9 位数据）由 SCON 中的 TB8 提供，可软件设置为 1 或 0，或者可将 PSW 中的奇/偶校验位 P 值装入 TB8（TB8 既可作为多机通信中的地址数据标志位，又可作为数据的奇偶校验位）。接收时第 9 位数据装入 SCON 的 RB8。TxD 为发送端口，RxD 为接收端口，以全双工模式进行接收/发送。

模式 2 的波特率固定为系统时钟的 64 分频或 32 分频（取决于 PCON 中 SMOD 的值）

串口 1 模式 2 的波特率计算公式如下表所示（SYSclk 为系统工作频率）：

SMOD	波特率计算公式
0	$\text{波特率} = \frac{\text{SYSclk}}{64}$
1	$\text{波特率} = \frac{\text{SYSclk}}{32}$

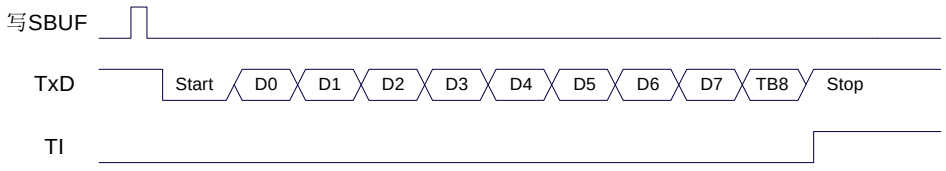
模式 2 和模式 1 相比，除波特率发生源略有不同，发送时由 TB8 提供给移位寄存器第 9 数据位不同外，其余功能结构均基本相同，其接收/发送操作过程及时序也基本相同。

当接收器接收完一帧信息后必须同时满足下列条件：

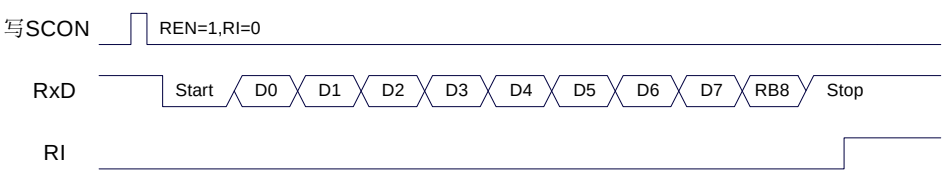
- RI=0
- SM2=0 或者 SM2=1 且接收到的第 9 数据位 RB8=1。

当上述两条件同时满足时，才将接收到的移位寄存器的数据装入 SBUF 和 RB8 中，RI 标志位被置 1，并向主机请求中断处理。如果上述条件有一个不满足，则刚接收到移位寄存器中的数据无效而丢失，也不置位 RI。无论上述条件满足与否，接收器又重新开始检测 RxD 输入端口的跳变信息，接收下一帧的输入信息。在模式 2 中，接收到的停止位与 SBUF、RB8 和 RI 无关。

通过软件对 SCON 中的 SM2、TB8 的设置以及通信 Ð 议的约定，为多机通信提供了方便。



发送数据（串口1模式2）



接收数据（串口1模式2）

13.2.4 串口 1 模式 3

当 SM0、SM1 两位为 11 时，串行口 1 工作在模式 3。串行通信模式 3 为 9 位数据异步通信 UART

模式，其一帧的信息由 11 位组成：1 位起始位，8 位数据位（低位在先），1 位可编程位（第 9 位数据）和 1 位停止位。发送时可编程位（第 9 位数据）由 SCON 中的 TB8 提供，可软件设置为 1 或 0，或者可将 PSW 中的奇/偶校验位 P 值装入 TB8（TB8 既可作为多机通信中的地址数据标志位，又可作为数据的奇偶校验位）。接收时第 9 位数据装入 SCON 的 RB8。TxD 为发送端口，RxD 为接收端口，以全双工模式进行接收/发送。

模式 3 和模式 1 相比，除发送时由 TB8 提供给移位寄存器第 9 数据位不同外，其余功能结构均基本相同，其接收‘发送操作过程及时序也基本相同。

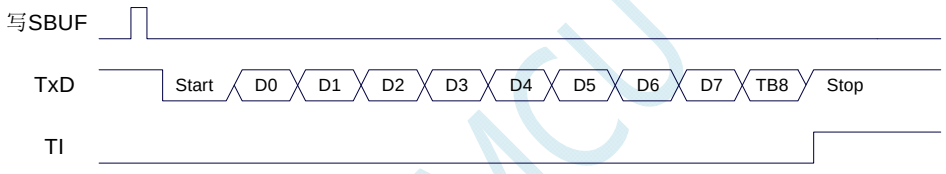
当接收器接收完一帧信息后必须同时满足下列条件：

• RI=0

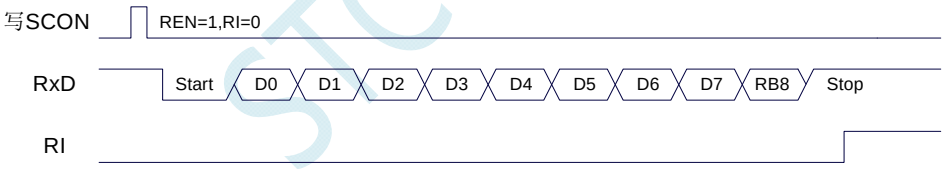
• SM2=0 或者 SM2=1 且接收到的第 9 数据位 RB8=1。

当上述两条件同时满足时，才将接收到的移位寄存器的数据装入 SBUF 和 RB8 中，RI 标志位被置 1，并向主机请求中断处理。如果上述条件有一个不满足，则刚接收到移位寄存器中的数据无效而丢失，也不置位 RI。无论上述条件满足与否，接收器又重新开始检测 RxD 输入端口的跳变信息，接收下一帧的输入信息。在模式 3 中，接收到的停止位与 SBUF、RB8 和 RI 无关。

通过软件对 SCON 中的 SM2、TB8 的设置以及通信协议的约定，为多机通信提供了方便。



发送数据（串口1模式3）



接收数据（串口1模式3）

串口 1 模式 3 的波特率计算公式与模式 1 是完全相同的。请参考模式 1 的波特率计算公式。

13.2.5 自动地址识别

串口 1 从机地址控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SADDR	A9H								
SADEN	B9H								

SADDR：从机地址寄存器

SADEN：从机地址屏蔽位寄存器

自动地址识别功能典型应用在多机通讯领域，其主要原理是从机系统通过硬件比较功能来识别来自于主机串口数据流中的地址信息，通过寄存器 SADDR 和 SADEN 设置的本机的从机地址，硬件自动对从机地址进行过滤，当来自于主机的从机地址信息与本机所设置的从机地址相匹配时，硬件产生串口中

断；否则硬件自动丢弃串口数据，而不产生中断。当众多处于空闲模式的从机链接在一起时，只有从机地址相匹配的从机才会从空闲模式唤醒，从而可以大大降低从机 MCU 的功耗，即使从机处于正常工作状态也可避免不停地进入串口中断而降低系统执行效率。

要使用串口的自动地址识别功能，首先需要将参与通讯的 MCU 的串口通讯模式设置为模式 2 或者模式 3（通常都选择波特率可变的模式 3，因为模式 2 的波特率是固定的，不便于调节），并开启从机的 SCON 的 SM2 位。对于串口模式 2 或者模式 3 的 9 位数据位中，第 9 位数据（存放在 RB8 中）为地址/数据的标志位，当第 9 位数据为 1 时，表示前面的 8 位数据（存放在 SBUF 中）为地址信息。当 SM2 被设置为 1 时，从机 MCU 会自动过滤掉非地址数据（第 9 位为 0 的数据），而对 SBUF 中的地址数据（第 9 位为 1 的数据）自动与 SADDR 和 SADEN 所设置的本机地址进行比较，若地址相匹配，则会将 RI 置“1”，并产生中断，否则不予处理本次接收的串口数据。

从机地址的设置是通过 SADDR 和 SADEN 两个寄存器进行设置的。SADDR 为从机地址寄存器，里面存放本机的从机地址。SADEN 为从机地址屏蔽位寄存器，用于设置地址信息中的忽略位，设置方法如下：

例如
SADDR ＝ 11001010
SADEN ＝ 10000001
则匹配地址为 1xxxxxx0
即，只要主机送出的地址数据中的 bit0 为 0 且 bit7 为 1 就可以和本机地址相匹配

再例如
SADDR ＝ 11001010
SADEN ＝ 00001111
则匹配地址为 xxxx1010
即，只要主机送出的地址数据中的低 4 位为 1010 就可以和本机地址相匹配，而高 4 为被忽略，可以为任意值。

主机可以使用广播地址（FFH）同时选中所有的从机来进行通讯。

13.3 串口 2

串口 2 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S2CON	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI

S2SM0：指定串口2的通信工作模式，如下表所示：

S2SM0	串口2工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S2SM2：允许串口 2 在模式 1 时允许多机通信控制位。在模式 1 时，如果 S2SM2 位为 1 且 S2REN 位为 1，则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位（即 S2RB8）来筛选地址帧：若 S2RB8=1，说明该帧是地址帧，地址信息可以进入 S2BUF，并使 S2RI 为 1，进而在中断服务程序中再进行地址号比较；若 S2RB8=0，说明该帧不是地址帧，应丢掉且保持 S2RI=0。在模式 1 中，如果 S2SM2 位为 0 且 S2REN 位为 1，接收收机处于地址帧筛选被禁止状态。不论收到的 S2RB8 为 0 或 1，均可使接收到的信息进入 S2BUF，并使 S2RI=1，此时 S2RB8 通常为校验位。模式 0 为非多机通信方式，在这种方式时，要设置 S2SM2 应为 0。

S2REN：允许/禁止串口接收控制位

0：禁止串口接收数据

1：允许串口接收数据

S2TB8：当串口 2 使用模式 1 时，S2TB8 为要发送的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位，按需要由软件置位或清 0。在模式 0 中，该位不用。

S2RB8：当串口 2 使用模式 1 时，S2RB8 为接收到的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位。在模式 0 中，该位不用。

S2TI：串口 2 发送中断请求标志位。在停止位开始发送时由硬件自动将 S2TI 置 1，向 CPU 发请求中断，响应中断后 S2TI 必须用软件清零。

S2RI：串口 2 接收中断请求标志位。串行接收到停止位的中间时刻由硬件自动将 S2RI 置 1，向 CPU 发中断申请，响应中断后 S2RI 必须由软件清零。

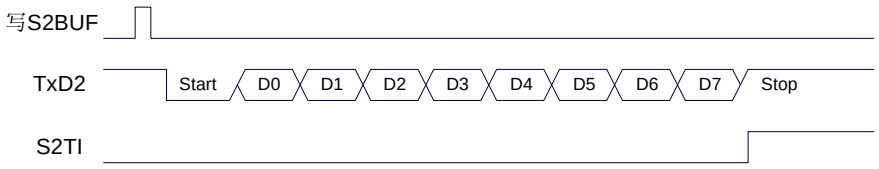
串口 2 数据寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S2BUF	9BH								

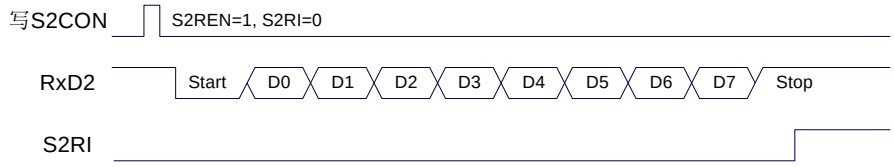
S2BUF：串口 1 数据接收/发送缓冲区。S2BUF 实际是 2 个缓冲器，读缓冲器和写缓冲器，两个操作分别对应两个不同的寄存器，1 个是只写寄存器（写缓冲器），1 个是只读寄存器（读缓冲器）。对 S2BUF 进行读操作，实际是读取串口接收缓冲区，对 S2BUF 进行写操作则是触发串口开始发送数据。

13.3.1 串口 2 模式 0

串行口 2 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD2 为数据发送口，RxD2 为数据接收口，串行口全双工接受/发送。



发送数据（串口2模式0）



接收数据（串口2模式0）

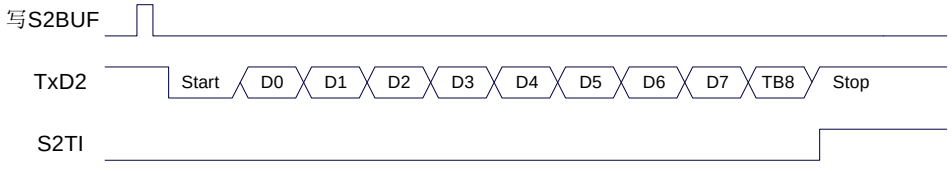
串口 2 的波特率是可变的，其波特率由定时器 2 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

串口 2 模式 0 的波特率计算公式如下表所示：（SYSclk 为系统工作频率）

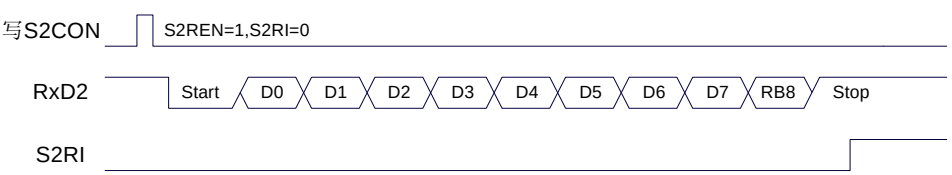
选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536-\frac{SYSclk}{4\times波特率}$
	12T	定时器2重载值 = $65536-\frac{SYSclk}{12\times4\times波特率}$

13.3.2 串口 2 模式 1

串行口 2 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位：1 位起始位，9 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD2 为数据发送口，RxD2 为数据接收口，串行口全双工接受/发送。



发送数据（串口2模式1）



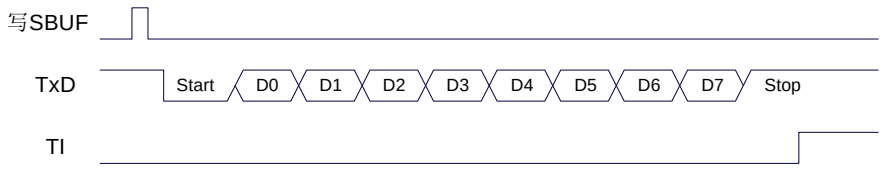
接收数据（串口2模式1）

串口 2 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

13.4 串口注意事项

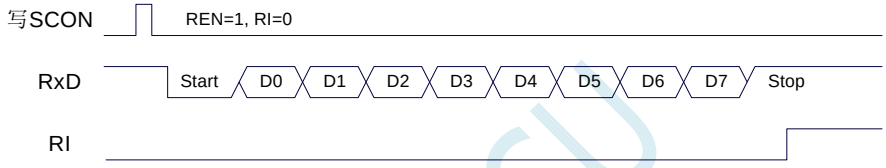
关于串口中断请求有如下问题需要注意：（串口 1、串口 2、串口 3、串口 4 均类似，下面以串口 1 为例进行说明）

8 位数据模式时，发送完成整个停止位后产生 TI 中断请求，如下图所示：



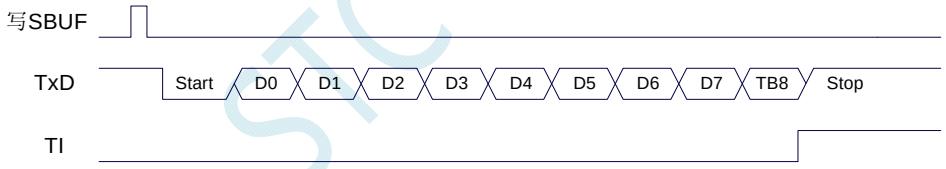
发送数据（8位数据）

8 位数据模式时，接收完成一半个停止位后产生 RI 中断请求，如下图所示：



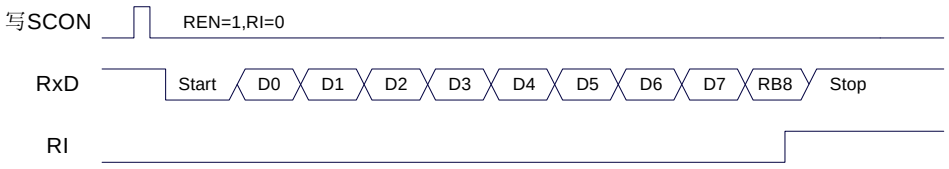
接收数据（8位数据）

9 位数据模式时，发送完成整个第 9 位数据位后产生 TI 中断请求，如下图所示：



发送数据（9位数据）

9 位数据模式时，接收完成一半个第 9 位数据位后产生 RI 中断请求，如下图所示：



接收数据（9位数据）

13.5 范例程序

13.5.1 串口 1 使用定时器 2 做波特率发生器

汇编代码

```
;测试工作频率为11.0592MHz

AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,SBUF
            INC     WPTR
UARTISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART_INIT:
            MOV     SCON,#50H
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#15H
            CLR     BUSY
```

```
                MOV         WPTR,#00H
                MOV         RPTR,#00H
                RET

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

UART_SENDSTR:
                CLR         A
                MOVC        A,@A+DPTR
                JZ          SENDEND
                LCALL       UART_SEND
                INC         DPTR
                JMP         UART_SENDSTR
SENDEND:
                RET

MAIN:
                MOV         SP, #5FH
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                LCALL       UART_INIT
                SETB        ES
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
```

```
#include "intrins.h"

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

sfr       AUXR        =   0x8e;
sfr       T2H         =   0xd6;
sfr       T2L         =   0xd7;

sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x15;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

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

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
```

```
    }
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

13.5.2 串口 1 使用定时器 1（模式 0）做波特率发生器

汇编代码

```
;测试工作频率为11.0592MHz

AUXR        DATA    8EH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H
```

```
            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,SBUF
            INC     WPTR
UARTISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

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

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

UART_SENDSTR:
            CLR     A
            MOVC    A,@A+DPTR
            JZ      SENDEND
            LCALL   UART_SEND
            INC     DPTR
            JMP     UART_SENDSTR
SENDEND:
            RET

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            LCALL   UART_INIT
            SETB    ES
            SETB    EA

            MOV     DPTR,#STRING
```

```
                LCALL       UART_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

sfr         AUXR        =   0x8e;

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

bit         busy;
char        wptr;
char        rptr;
char        buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}
```

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

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

void UartSendStr(char *p)
{
	while (*p)
	{
		UartSend(*p++);
	}
}

void main()
{
	P1M0 = 0x00;
	P1M1 = 0x00;
	P3M0 = 0x00;
	P3M1 = 0x00;
	P5M0 = 0x00;
	P5M1 = 0x00;

	UartInit();
	ES = 1;
	EA = 1;
	UartSendStr("Uart Test !\r\n");

	while (1)
	{
		if (rptr != wptr)
		{
			UartSend(buffer[rptr++]);
			rptr &= 0x0f;
		}
	}
}
```


13.5.3 串口 1 使用定时器 1（模式 2）做波特率发生器

汇编代码

```
;测试工作频率为11.0592MHz

AUXR		DATA		8EH
```

```
BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,SBUF
            INC     WPTR
UARTISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART_INIT:
            MOV     SCON,#50H
            MOV     TMOD,#20H
            MOV     TL1,#0FDH           ;256-11059200/115200/32=0FDH
            MOV     TH1,#0FDH
            SETB    TR1
            MOV     AUXR,#40H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

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

```
UART_SENDSTR:
        CLR     A
        MOVC    A,@A+DPTR
        JZ      SENDEND
        LCALL   UART_SEND
        INC     DPTR
        JMP     UART_SENDSTR
SENDEND:
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        LCALL   UART_INIT
        SETB    ES
        SETB    EA

        MOV     DPTR,#STRING
        LCALL   UART_SENDSTR

LOOP:
        MOV     A,RPTR
        XRL     A,WPTR
        ANL     A,#0FH
        JZ      LOOP
        MOV     A,RPTR
        ANL     A,#0FH
        ADD     A,#BUFFER
        MOV     R0,A
        MOV     A,@R0
        LCALL   UART_SEND
        INC     RPTR
        JMP     LOOP

STRING: DB      'Uart Test !',0DH,0AH,00H

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

#define   FOSC          11059200UL
#define   BRT           (256 - FOSC / 115200 / 32)

sfr       AUXR          =     0x8e;

sfr       P1M1          =     0x91;
sfr       P1M0          =     0x92;
```

```
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

bit         busy;
char        wptr;
char        rptr;
char        buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x20;
    TL1 = BRT;
    TH1 = BRT;
    TR1 = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

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

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
```

```
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

13.5.4 串口 2 使用定时器 2 做波特率发生器

汇编代码

```
;测试工作频率为11.0592MHz

AUXR    DATA    8EH
T2H     DATA    0D6H
T2L     DATA    0D7H
S2CON   DATA    9AH
S2BUF   DATA    9BH
IE2     DATA    0AFH

BUSY    BIT     20H.0
WPTR    DATA    21H
RPTR    DATA    22H
BUFFER  DATA    23H                 ;16 bytes

P1M1    DATA    091H
P1M0    DATA    092H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
        ORG     0043H
        LJMP    UART2_ISR

        ORG     0100H

UART2_ISR:
        PUSH    ACC
        PUSH    PSW
        MOV     PSW,#08H

        MOV     A,S2CON
        JNB     ACC.1,CHKRI
        ANL     S2CON,#NOT 02H
        CLR     BUSY
```

```
CHKRI:
        JNB     ACC.0,UART2ISR_EXIT
        ANL     S2CON,#NOT 01H
        MOV     A,WPTR
        ANL     A,#0FH
        ADD     A,#BUFFER
        MOV     R0,A
        MOV     @R0,S2BUF
        INC     WPTR
UART2ISR_EXIT:
        POP     PSW
        POP     ACC
        RETI

UART2_INIT:
        MOV     S2CON,#10H
        MOV     T2L,#0E8H               ;65536-11059200/115200/4=0FFE8H
        MOV     T2H,#0FFH
        MOV     AUXR,#14H
        CLR     BUSY
        MOV     WPTR,#00H
        MOV     RPTR,#00H
        RET

UART2_SEND:
        JB      BUSY,$
        SETB    BUSY
        MOV     S2BUF,A
        RET

UART2_SENDSTR:
        CLR     A
        MOVC    A,@A+DPTR
        JZ      SEND2END
        LCALL   UART2_SEND
        INC     DPTR
        JMP     UART2_SENDSTR
SEND2END:
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        LCALL   UART2_INIT
        MOV     IE2,#01H
        SETB    EA

        MOV     DPTR,#STRING
        LCALL   UART2_SENDSTR

LOOP:
        MOV     A,RPTR
        XRL     A,WPTR
```

```
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART2_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

sfr        AUXR         =    0x8e;
sfr        T2H          =    0xd6;
sfr        T2L          =    0xd7;
sfr        S2CON        =    0x9a;
sfr        S2BUF        =    0x9b;
sfr        IE2          =    0xaf;

sfr        P1M1         =    0x91;
sfr        P1M0         =    0x92;
sfr        P3M1         =    0xb1;
sfr        P3M0         =    0xb2;
sfr        P5M1         =    0xc9;
sfr        P5M0         =    0xca;

bit        busy;
char       wptr;
char       rptr;
char       buffer[16];

void Uart2Isr() interrupt 8
{
    if (S2CON & 0x02)
    {
        S2CON &= ~0x02;
        busy = 0;
    }
    if (S2CON & 0x01)
    {
        S2CON &= ~0x01;
        buffer[wptr++] = S2BUF;
        wptr &= 0x0f;
    }
}
```

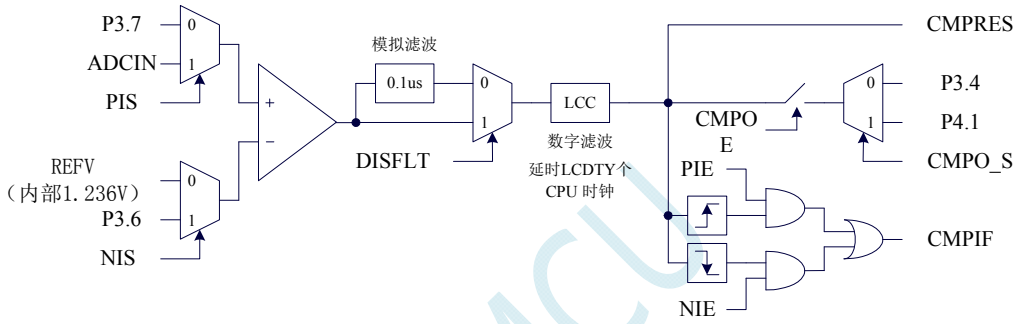
```
void Uart2Init()
{
    S2CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart2Send(char dat)
{
    while (busy);
    busy = 1;
    S2BUF = dat;
}

void Uart2SendStr(char *p)
{
    while (*p)
    {
        Uart2Send(*p++);
    }
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart2Init();
    IE2 = 0x01;
    EA = 1;
    Uart2SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart2Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

14 比较器，掉电检测，内部固定比较电压

STC8G 系列单片机内部集成了一个比较器。比较器的正极可以是 P3.7 端口或者 ADC 的模拟输入通道，而负极可以 P3.6 端口或者是内部 BandGap 经过 OP 后的 REFV 电压（内部固定比较电压）。

比较器内部有可程序控制的两级滤波：模拟滤波和数字滤波。模拟滤波可以过滤掉比较输入信号中的毛刺信号，数字滤波可以等待输入信号更加稳定后再进行比较。比较结果可直接通过读取内部寄存器位获得，也可将比较器结果正向或反向输出到外部端口。将比较结果输出到外部端口可用作外部事件的触发信号和反馈信号，可扩大比较的应用范围。

14.1 比较器内部结构图



比较器内部结构

14.2 比较器相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CMPCR1	比较器控制寄存器 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
CMPCR2	比较器控制寄存器 2	E7H	INVCMPO	DISFLT	LCDTY[5:0]						0000,0000

比较器控制寄存器 1

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

CMPEN：比较器模块使能位

0：关闭比较功能

1：使能比较功能

CMPIF：比较器中断标志位。当 PIE 或 NIE 被使能后，若产生相应的中断信号，硬件自动将 CMPIF 置 1，并向 CPU 提出中断请求。此标志位必须用户软件清零。

（注意：没有使能比较器中断时，硬件不会设置此中断标志，即使用查询方式访问比较器时，不能查询此中断标志）

PIE：比较器上升沿中断使能位。

0：禁止比较器上升沿中断。

1：使能比较器上升沿中断。使能比较器的比较结果由 0 变成 1 时产生中断请求。

NIE：比较器下降沿中断使能位。

0：禁止比较器下降沿中断。

1：使能比较器下降沿中断。使能比较器的比较结果由 1 变成 0 时产生中断请求。

PIS：比较器的正极选择位

0：选择外部端口 P3.7 为比较器正极输入源。

1：通过 ADC_CONTR 中的 ADC_CHS 位选择 ADC 的模拟输入端作为比较器正极输入源。

NIS：比较器的负极选择位

0：选择内部 BandGap 经过 OP 后的电压 REFV 作为比较器负极输入源（REFV 的电压值为 1.344V，由于制造误差，实际电压值可能在 1.34V～1.35V 之间）。

1：选择外部端口 P3.6 为比较器负极输入源。

CMPOE：比较器结果输出控制位

0：禁止比较器结果输出

1：使能比较器结果输出。比较器结果输出到 P3.4 或者 P4.1（由 P_SW2 中的 CMPO_S 进行设定）

CMPRES：比较器的比较结果。此位为只读。

0：表示 CMP+的电平低于 CMP-的电平

1：表示 CMP+的电平高于 CMP-的电平

CMPRES 是经过数字滤波后的输出信号，而不是比较器的直接输出结果。

比较器控制寄存器 2

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR2	E7H	INVCMPO	DISFLT	LCDTY[5:0]					

INVCMPO：比较器结果输出控制

0：比较器结果正向输出。若 CMPRES 为 0，则 P3.4/P4.1 输出低电平，反之输出高电平。

1：比较器结果反向输出。若 CMPRES 为 0，则 P3.4/P4.1 输出高电平，反之输出低电平。

DISFLT：模拟滤波功能控制

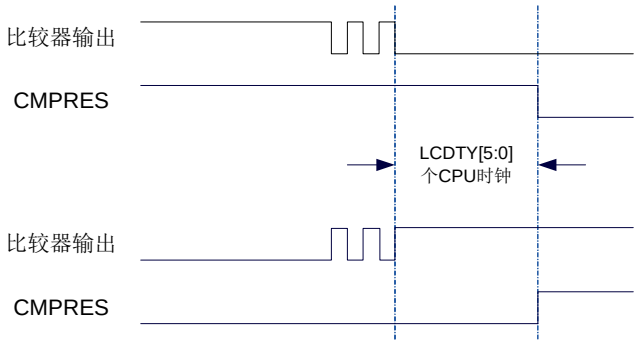
0：使能 0.1us 模拟滤波功能

1：关闭 0.1us 模拟滤波功能，可略微提高比较器的比较速度。

LCDTY[5:0]：数字滤波功能控制

数字滤波功能即为数字信号去抖动功能。当比较结果发生上升沿或者下降沿变化时，比较器侦测变化后的信号必须维持 LCDTY 所设置的 CPU 时钟数不发生变化，才认为数据变化是有效的；否则将视同信号无变化。

若 LCDTY 设置为 0 时表示关闭数字滤波功能。



14.3 范例程序

14.3.1 比较器的使用（中断方式）

汇编代码

```
;测试工作频率为11.0592MHz

CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00ABH
            LJMP    CMPISR

            ORG     0100H
CMPISR:
            PUSH    ACC
            ANL     CMPCR1,#NOT 40H     ;清中断标志
            MOV     A,CMPCR1
            JB      ACC.0,RSING
FALLING:
            CPL     P1.0                ;下降沿中断测试端口
            POP     ACC
            RETI
RSING:
            CPL     P1.1                ;上升沿中断测试端口
            POP     ACC
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     CMPCR2,#00H
            ANL     CMPCR2,#NOT 80H     ;比较器正向输出
;           ORL     CMPCR2,#80H         ;比较器反向输出
            ANL     CMPCR2,#NOT 40H     ;禁止 0.1us 滤波
;           ORL     CMPCR2,#40H         ;使能 0.1us 滤波
;           ANL     CMPCR2,#NOT 3FH     ;比较器结果直接输出
            ORL     CMPCR2,#10H         ;比较器结果经过 16 个去抖时钟后输出
            MOV     CMPCR1,#00H
            ORL     CMPCR1,#30H         ;使能比较器边沿中断
;           ANL     CMPCR1,#NOT 20H     ;禁止比较器上升沿中断
;           ORL     CMPCR1,#20H         ;使能比较器上升沿中断
;           ANL     CMPCR1,#NOT 10H     ;禁止比较器下降沿中断
```

```
;               ORL     CMPCR1,#10H             ;使能比较器下降沿中断
                ANL     CMPCR1,#NOT 08H         ;P3.7 为CMP+输入脚
;               ORL     CMPCR1,#08H             ;ADC 输入脚为CMP+输入教
;               ANL     CMPCR1,#NOT 04H         ;内部参考电压为CMP-输入脚
                ORL     CMPCR1,#04H             ;P3.6 为CMP-输入脚
;               ANL     CMPCR1,#NOT 02H         ;禁止比较器输出
                ORL     CMPCR1,#02H             ;使能比较器输出
                ORL     CMPCR1,#80H             ;使能比较器模块
                SETB    EA

                JMP     $

                END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

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

void CMP_Isr() interrupt 21
{
    CMPCR1 &= ~0x40;                        //清中断标志
    if (CMPCR1 & 0x01)
    {
        P10 = !P10;                         //下降沿中断测试端口
    }
    else
    {
        P11 = !P11;                         //上升沿中断测试端口
    }
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CMPCR2 = 0x00;
    CMPCR2 &= ~0x80;                        //比较器正向输出
//  CMPCR2 |= 0x80;                         //比较器反向输出
```

```
    CMPCR2 &= ~0x40;                    //禁止 0.1us 滤波
//  CMPCR2 |= 0x40;                     //使能 0.1us 滤波
//  CMPCR2 &= ~0x3f;                    //比较器结果直接输出
    CMPCR2 |= 0x10;                     //比较器结果经过 16 个去抖时钟后输出
    CMPCR1 = 0x00;
    CMPCR1 |= 0x30;                     //使能比较器边沿中断
//  CMPCR1 &= ~0x20;                    //禁止比较器上升沿中断
//  CMPCR1 |= 0x20;                     //使能比较器上升沿中断
//  CMPCR1 &= ~0x10;                    //禁止比较器下降沿中断
//  CMPCR1 |= 0x10;                     //使能比较器下降沿中断
    CMPCR1 &= ~0x08;                    //P3.7 为CMP+输入脚
//  CMPCR1 |= 0x08;                     //ADC 输入脚为CMP+输入教
//  CMPCR1 &= ~0x04;                    //内部参考电压为CMP-输入脚
    CMPCR1 |= 0x04;                     //P3.6 为CMP-输入脚
//  CMPCR1 &= ~0x02;                    //禁止比较器输出
    CMPCR1 |= 0x02;                     //使能比较器输出
    CMPCR1 |= 0x80;                     //使能比较器模块

    EA = 1;

    while (1);
}
```

14.3.2 比较器的使用（查询方式）

汇编代码

```
;测试工作频率为11.0592MHz

CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     CMPCR2,#00H
            ANL     CMPCR2,#NOT 80H     ;比较器正向输出
;           ORL     CMPCR2,#80H         ;比较器反向输出
            ANL     CMPCR2,#NOT 40H     ;禁止 0.1us 滤波
;           ORL     CMPCR2,#40H         ;使能 0.1us 滤波
;           ANL     CMPCR2,#NOT 3FH     ;比较器结果直接输出
            ORL     CMPCR2,#10H         ;比较器结果经过 16 个去抖时钟后输出
```

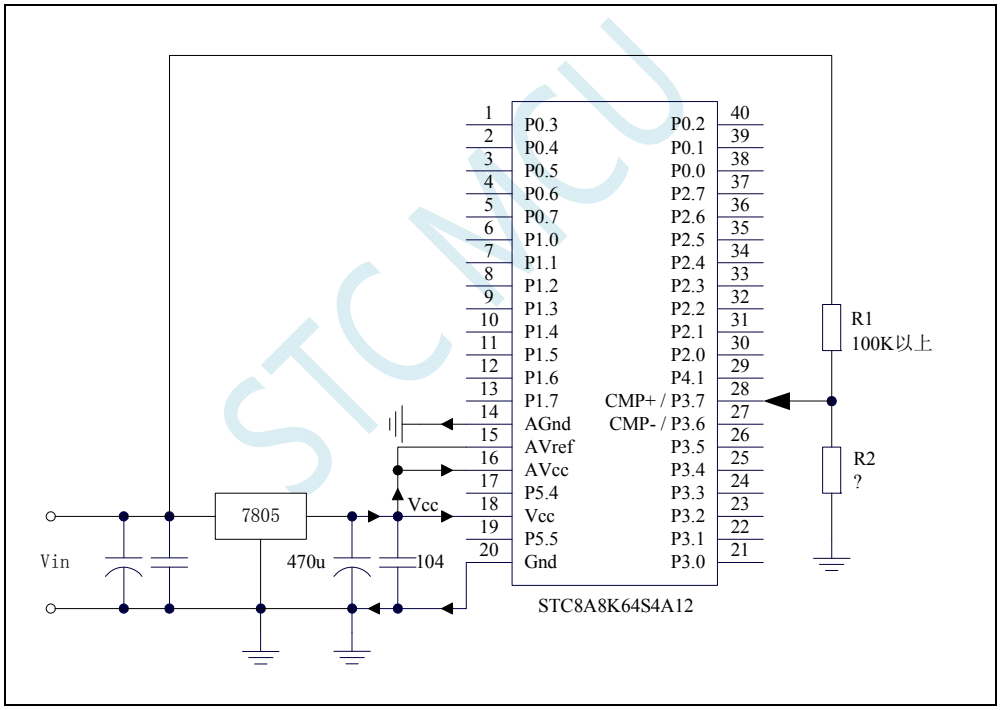
```
        MOV     CMPCR1,#00H
        ORL     CMPCR1,#30H         ;使能比较器边沿中断
;       ANL     CMPCR1,#NOT 20H     ;禁止比较器上升沿中断
;       ORL     CMPCR1,#20H         ;使能比较器上升沿中断
;       ANL     CMPCR1,#NOT 10H     ;禁止比较器下降沿中断
;       ORL     CMPCR1,#10H         ;使能比较器下降沿中断
        ANL     CMPCR1,#NOT 08H     ;P3.7 为CMP+输入脚
;       ORL     CMPCR1,#08H         ;ADC 输入脚为CMP+输入教
;       ANL     CMPCR1,#NOT 04H     ;内部参考电压为CMP-输入脚
        ORL     CMPCR1,#04H         ;P3.6 为CMP-输入脚
;       ANL     CMPCR1,#NOT 02H     ;禁止比较器输出
        ORL     CMPCR1,#02H         ;使能比较器输出
        ORL     CMPCR1,#80H         ;使能比较器模块

LOOP:
        MOV     A,CMPCR1
        MOV     C,ACC.0
        MOV     P1.0,C              ;读取比较器比较结果
        JMP     LOOP

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

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

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CMPCR2 = 0x00;
    CMPCR2 &= ~0x80;                //比较器正向输出
//  CMPCR2 |= 0x80;                 //比较器反向输出
    CMPCR2 &= ~0x40;                //禁止 0.1us 滤波
//  CMPCR2 |= 0x40;                 //使能 0.1us 滤波
//  CMPCR2 &= ~0x3f;                //比较器结果直接输出
    CMPCR2 |= 0x10;                 //比较器结果经过 16 个去抖时钟后输出
    CMPCR1 = 0x00;
```

```
    CMPCR1 |= 0x30;                 //使能比较器边沿中断
//  CMPCR1 &= ~0x20;                //禁止比较器上升沿中断
//  CMPCR1 |= 0x20;                 //使能比较器上升沿中断
//  CMPCR1 &= ~0x10;                //禁止比较器下降沿中断
//  CMPCR1 |= 0x10;                 //使能比较器下降沿中断
    CMPCR1 &= ~0x08;                //P3.7 为CMP+输入脚
//  CMPCR1 |= 0x08;                 //ADC 输入脚为CMP+输入教
//  CMPCR1 &= ~0x04;                //内部参考电压为CMP-输入脚
    CMPCR1 |= 0x04;                 //P3.6 为CMP-输入脚
//  CMPCR1 &= ~0x02;                //禁止比较器输出
    CMPCR1 |= 0x02;                 //使能比较器输出
    CMPCR1 |= 0x80;                 //使能比较器模块

    while (1)
    {
        P10 = CMPCR1 & 0x01;        //读取比较器比较结果
    }
}
```

14.3.3 比较器作外部掉电检测



上图中电阻 R1 和 R2 对稳压块 7805 的前端电压进行分压，分压后的电压作为比较器 CMP+的外部输入与内部参考电压（约 1.344V 附近）进行比较。

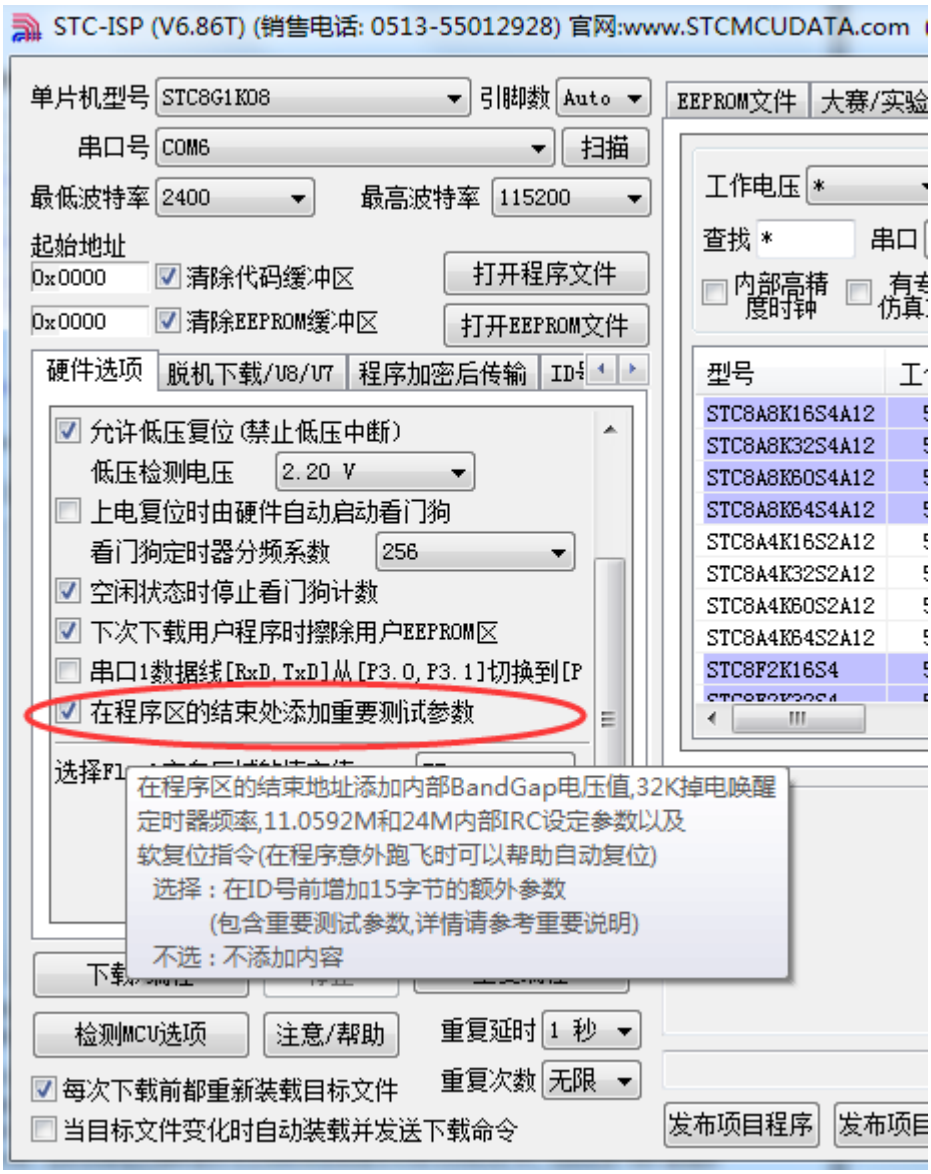
一般当交流电在 220V 时，稳压块 7805 前端的直流电压为 11V，但当交流电压降到 160V 时，稳压块 7805 前端的直留电压为 8.5V。当稳压块 7805 前端的直留电压低于或等于 8.5V 时，该前端输入的直留电压被电阻 R1 和 R2 分压到比较器正极输入端 CMP+，CMP+端输入电压低于内部参考电压，此时可产生比较器中断，这样在掉电检测时就有充足的时间将数据保存到 EEPROM 中。当稳压块 7805 前端的直留电压高高于 8.5V 时，该前端输入的直流电压被电阻 R1 和 R2 分压到比较器正极输入端 CMP+，CMP+端输入电压高于内部参考电压，此时 CPU 可继续正常工作。

内部参考电压即为内部 BandGap 经过 OP 后的电压 REFV，REFV 的电压值约在 1.344V 附近，由于制造误差，实际电压值可能在 1.34V～1.35V 之间。具体的数值要通过读取内部参考电压在内部 RAM

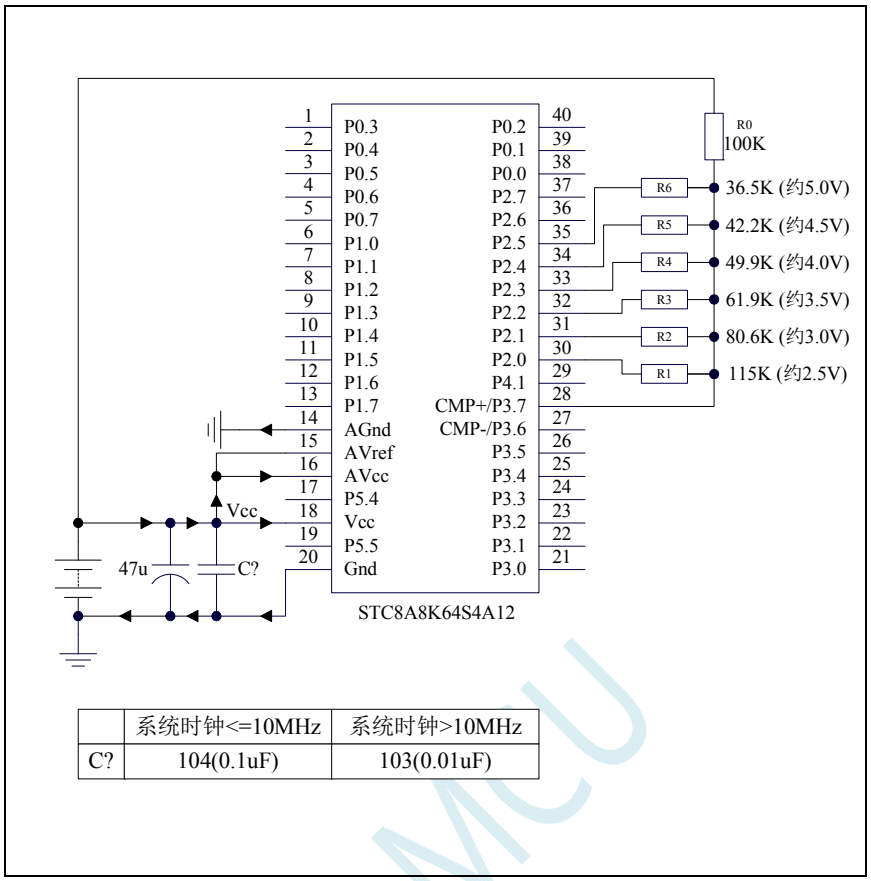
区或者 ROM 区所占用的地址的值获得。对于 STC8 系列，内部参考电压值在 RAM 和 ROM 中的存储地址如下表所示：

单片机型号	RAM 中存储的地址（高字节在前）	ROM 中存储的地址（高字节在前）
STC8G1K08	0EFH-0F0H	1FF7H-1FF8H
STC8G1K12	0EFH-0F0H	2FF7H-2FF8H

注：若需要从 ROM 中读取参考电压值，则需要在进行 ISP 下载时勾选下列选项



14.3.4 比较器检测工作电压（电池电压）



	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

上图中，利用电阻分压的原理可以近似的测量出 MCU 的工作电压（选通的通道，MCU 的 I/O 口输出低电平，端口电压值接近 GND，未选通的通道，MCU 的 I/O 口输出开漏模式的高，不影响其他通道）。

比较器的负端选择内部参考电压（约 1.344V），正端选择通过电阻分压后输入到 CMP+管脚的电压值。

初始化时 P2.5~P2.0 口均设置为开漏模式，并输出高。首先 P2.0 口输出低电平，此时若 VCC 电压低于 2.5V 则比较器的比较值为 0，反之若 VCC 电压高于 2.5V 则比较器的比较值为 1；

若确定 VCC 高于 2.5V，则将 P2.0 口输出高，P2.1 口输出低电平，此时若 VCC 电压低于 3.0V 则比较器的比较值为 0，反之若 VCC 电压高于 3.0V 则比较器的比较值为 1；

若确定 VCC 高于 3.0V，则将 P2.1 口输出高，P2.2 口输出低电平，此时若 VCC 电压低于 3.5V 则比较器的比较值为 0，反之若 VCC 电压高于 3.5V 则比较器的比较值为 1；

若确定 VCC 高于 3.5V，则将 P2.2 口输出高，P2.3 口输出低电平，此时若 VCC 电压低于 4.0V 则比较器的比较值为 0，反之若 VCC 电压高于 4.0V 则比较器的比较值为 1；

若确定 VCC 高于 4.0V，则将 P2.3 口输出高，P2.4 口输出低电平，此时若 VCC 电压低于 4.5V 则比较器的比较值为 0，反之若 VCC 电压高于 4.5V 则比较器的比较值为 1；

若确定 VCC 高于 4.5V，则将 P2.4 口输出高，P2.5 口输出低电平，此时若 VCC 电压低于 5.0V 则比较器的比较值为 0，反之若 VCC 电压高于 5.0V 则比较器的比较值为 1。

汇编代码

```
;测试工作频率为11.0592MHz
```

```
CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

P2M0        DATA    096H
P2M1        DATA    095H
P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P2M0,#00111111B     ;P2.5~P2.0 初始化为开漏模式
            MOV     P2M1,#00111111B
            MOV     P2,#0FFH
            MOV     CMPCR2,#10H         ;比较器结果经过 16 个去抖时钟后输出
            MOV     CMPCR1,#00H
            ANL     CMPCR1,#NOT 08H     ;P3.7 为 CMP+输入脚
            ANL     CMPCR1,#NOT 04H     ;内部参考电压为 CMP-输入脚
            ANL     CMPCR1,#NOT 02H     ;禁止比较器输出
            ORL     CMPCR1,#80H         ;使能比较器模块
LOOP:
            MOV     R0,#00000000B       ;电压<2.5V
            MOV     P2,#11111110B       ;P2.0 输出 0
            CALL    DELAY
            MOV     A,CMPCR1
            JNB     ACC.0,SKIP
            MOV     R0,#00000001B       ;电压>2.5V
            MOV     P2,#11111101B       ;P2.1 输出 0
            CALL    DELAY
            MOV     A,CMPCR1
            JNB     ACC.0,SKIP
            MOV     R0,#00000011B       ;电压>3.0V
            MOV     P2,#11111011B       P2.2 输出 0
            CALL    DELAY
            MOV     A,CMPCR1
            JNB     ACC.0,SKIP
            MOV     R0,#00000111B       ;电压>3.5V
            MOV     P2,#11110111B       ;P2.3 输出 0
            CALL    DELAY
            MOV     A,CMPCR1
            JNB     ACC.0,SKIP
            MOV     R0,#00001111B       ;电压>4.0V
            MOV     P2,#11101111B       ;P2.4 输出 0
            CALL    DELAY
            MOV     A,CMPCR1
```

```
                JNB         ACC.0,SKIP
                MOV         R0,#00011111B       ;电压>4.5V
                MOV         P2,#11011111B       ;P2.5 输出 0
                CALL        DELAY
                MOV         A,CMPCR1
                JNB         ACC.0,SKIP
                MOV         R0,#00111111B       ;电压>5.0V
SKIP:
                MOV         P2,#11111111B
                MOV         A,R0
                CPL         A
                MOV         P0,A                ;P0.5~P0.0 口显示电压
                JMP         LOOP

DELAY:
                MOV         R0,#20
                DJNZ        R0,$
                RET

                END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sfr         P2M0        =   0x96;
sfr         P2M1        =   0x95;

void delay ()
{
    char i;

    for (i=0; i<20; i++);
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    unsigned char v;
```

```
    P2M0 = 0x3f;                                    // P2.5~P2.0 初始化为开漏模式
    P2M1 = 0x3f;
    P2 = 0xff;

    CMPCR2 = 0x10;                                  //比较器结果经过 16 个去抖时钟后输出
    CMPCR1 = 0x00;
    CMPCR1 &= ~0x08;                                //P3.6 为CMP+输入脚
    CMPCR1 &= ~0x04;                                //内部参考电压为CMP-输入脚
    CMPCR1 &= ~0x02;                                //禁止比较器输出
    CMPCR1 |= 0x80;                                 //使能比较器模块

    while (1)
    {
        v = 0x00;                                   //电压<2.5V
        P2 = 0xfe;                                  //P2.0 输出 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x01;                                   //电压>2.5V
        P2 = 0xfd;                                  //P2.1 输出 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x03;                                   //电压>3.0V
        P2 = 0xfb;                                  //P2.2 输出 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x07;                                   //电压>3.5V
        P2 = 0xf7;                                  //P2.3 输出 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x0f;                                   //电压>4.0V
        P2 = 0xef;                                  //P2.4 输出 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x1f;                                   //电压>4.5V
        P2 = 0xdf;                                  //P2.5 输出 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x3f;                                   //电压>5.0V
ShowVol:
        P2 = 0xff;
        P0 = ~v;
    }
}
```

15 IAP/EEPROM

STC8G 系列单片机内部集成了大容量的 EEPROM。利用 ISP/IAP 技术可将内部 Data Flash 当 EEPROM，擦写次数在 10 万次以上。EEPROM 可分为若干个扇区，每个扇区包含 512 字节。使用时，建议同一次修改的数据放在同一个扇区，不是同一次修改的数据放在不同的扇区，不一定要用满。数据存储器的擦除操作是按扇区进行的。

EEPROM 可用于保存一些需要在应用过程中修改并且掉电不丢失的参数数据。在用户程序中，可以对 EEPROM 进行字节读/字节编程/扇区擦除操作。在工作电压偏低时，建议不要进行 EEPROM 操作，以免发送数据丢失的情况。

15.1 EEPROM相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
IAP_DATA	IAP 数据寄存器	C2H									1111,1111
IAP_ADDRH	IAP 高地址寄存器	C3H									0000,0000
IAP_ADDRL	IAP 低地址寄存器	C4H									0000,0000
IAP_CMD	IAP 命令寄存器	C5H	-	-	-	-	-	-	CMD[1:0]		xxxx,xx00
IAP_TRIG	IAP 触发寄存器	C6H									0000,0000
IAP_CONTR	IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-	0000,xxxx
IAP_TPS	IAP 等待时间控制寄存器	F5H	-	-	IAPTPS[5:0]						xx00,0000

EEPROM 数据寄存器（IAP_DATA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_DATA	C2H								

在进行 EEPROM 的读操作时，命令执行完成后读出的 EEPROM 数据保存在 IAP_DATA 寄存器中。在进行 EEPROM 的写操作时，在执行写命令前，必须将待写入的数据存放在 IAP_DATA 寄存器中，再发送写命令。擦除 EEPROM 命令与 IAP_DATA 寄存器无关。

EEPROM 地址寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_ADDRH	C3H								
IAP_ADDRL	C4H								

EEPROM 进行读、写、擦除操作的目标地址寄存器。IAP_ADDRH 保存地址的高字节，IAP_ADDRL 保存地址的低字节

EEPROM 命令寄存器（IAP_CMD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CMD	C5H	-	-	-	-	-	-	CMD[1:0]	

CMD[1:0]：发送EEPROM操作命令

00：空操作

01：读 EEPROM 命令。读取目标地址所在的 1 字节。

10：写 EEPROM 命令。写目标地址所在的 1 字节。

11：擦除 EEPROM。擦除目标地址所在的 1 页（1 扇区/512 字节）。

EEPROM 触发寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TRIG	C6H								

设置完成 EEPROM 读、写、擦除的命令寄存器、地址寄存器、数据寄存器以及控制寄存器后，需要向触发寄存器 IAP_TRIG 依次写入 5AH、A5H（顺序不能交换）两个触发命令来触发相应的读、写、擦除操作。操作完成后，EEPROM 地址寄存器 IAP_ADDRH、IAP_ADDRL 和 EEPROM 命令寄存器 IAP_CMD 的内容不变。如果接下来要对下一个地址的数据进行操作，需手动更新地址寄存器 IAP_ADDRH 和寄存器 IAP_ADDRL 的值。

注意：每次 EEPROM 操作时，都要对 IAP_TRIG 先写入 5AH，再写入 A5H，相应的命令才会生效。写完触发命令后，CPU 会处于 IDLE 等待状态，直到相应的 IAP 操作执行完成后 CPU 才会从 IDLE 状态返回正常状态继续执行 CPU 指令。

EEPROM 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-

IAPEN：EEPROM操作使能控制位

0：禁止 EEPROM 操作

1：使能 EEPROM 操作

SWBS：软件复位选择控制位，（需要与SWRST配合使用）

0：软件复位后从用户代码开始执行程序

1：软件复位后从系统 ISP 监控代码区开始执行程序

SWRST：软件复位控制位

0：无动作

1：产生软件复位

CMD_FAIL：EEPROM操作失败状态位，需要软件清零

0：EEPROM 操作正确

1：EEPROM 操作失败

EEPROM 擦除等待时间控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TPS	F5H	-	-	IAPTPS[5:0]					

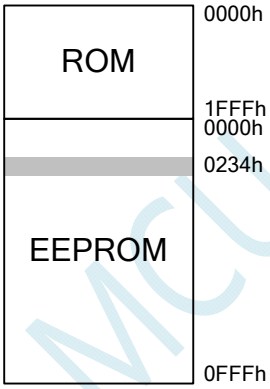
需要根据工作频率进行设置

若工作频率为12MHz，则需要将IAP_TPS设置为12；若工作频率为24MHz，则需要将IAP_TPS设置为24，其他频率以此类推。

15.2 EEPROM大小及地址

STC8G 系列单片机内部均有用于保存用户数据的 EEPROM。内部的 EEPROM 有 3 操作方式：读、写和擦除，其中擦除操作是以扇区为单位进行操作，每扇区为 512 字节，即每执行一次擦除命令就会擦除一个扇区，而读数据和写数据都是以字节为单位进行操作的，即每执行一次读或者写命令时只能读出或者写入一个字节。

STC8G 系列单片机内部的 EEPROM 的访问方式有两种：IAP 方式和 MOVC 方式。IAP 方式可对 EEPROM 执行读、写、擦除操作，但 MOVC 只能对 EEPROM 进行读操作，而不能进行写和擦除操作。无论是使用 IAP 方式还是使用 MOVC 方式访问 EEPROM，首先都需要设置正确的目标地址。IAP 方式时，目标地址与 EEPROM 实际的物理地址是一致的，均是从地址 0000H 开始访问，但若要使用 MOVC 指令进行读取 EEPROM 数据时，目标地址必须是在 EEPROM 实际的物理地址的基础上还有加上程序大小的偏移。下面以 STC8G1K08 这个型号为例，对目标地址进行详细说明：



STC8C1K08

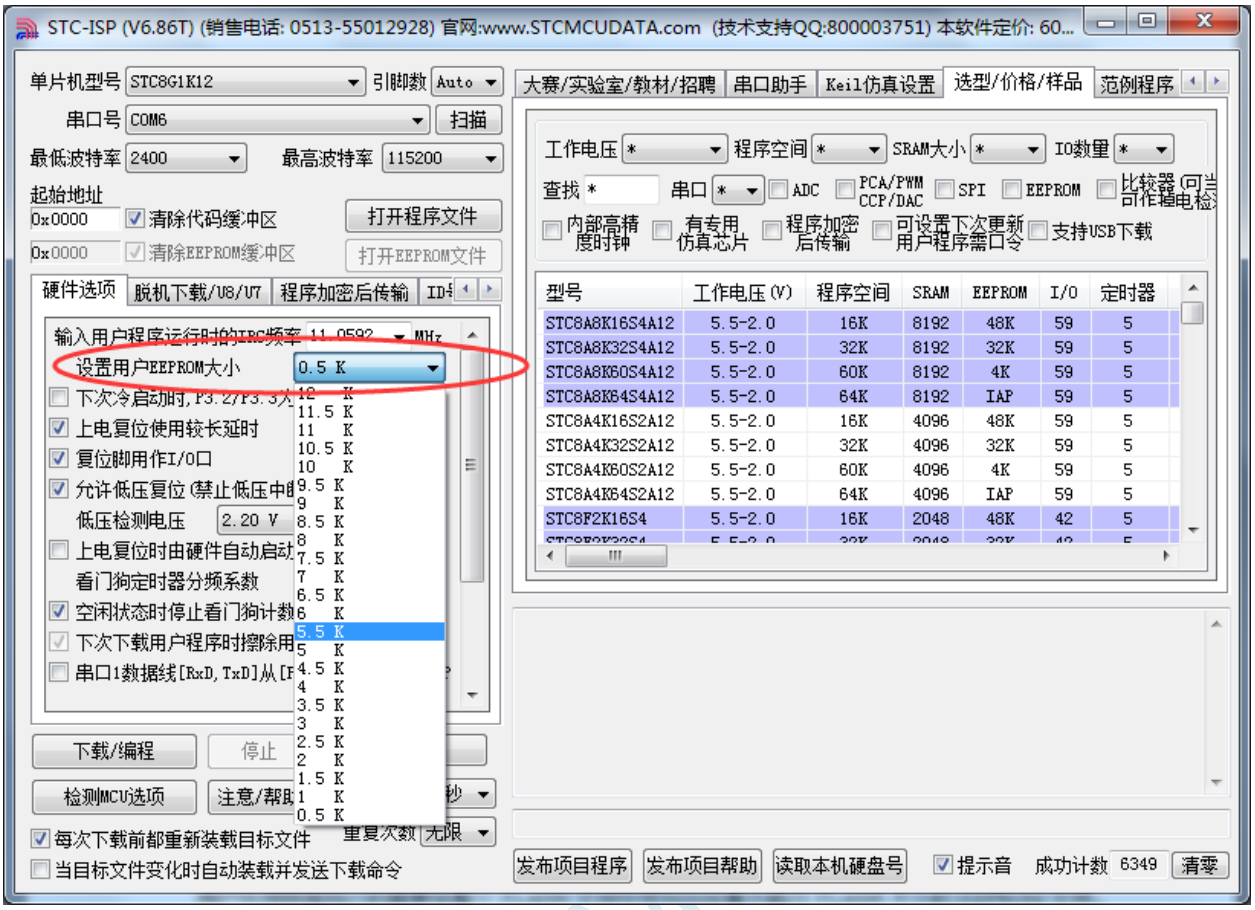
STC8G1K08 的程序空间为 8K 字节（0000h~1FFFh），EEPROM 空间为 4K（0000h~0FFFh）。当需要对 EEPROM 物理地址 0234h 的单元进行读、写、擦除时，若使用 IAP 方式进行访问时，设置的目标地址为 0234h，即 IAP_ADDRH 设置 02h，IAP_ADDRL 设置 34h，然后设置相应的触发命令即可对 0234h 单元进行正确操作了。但若是使用 MOVC 方式读取 EEPROM 的 0234h 单元，则必须在 0234h 的基础上还有加上 ROM 空间的大小 2000h，即必须将 DPTR 设置为 2234h，然后才能使用 MOVC 指令进行读取。

注意：由于擦除是以 512 字节为单位进行操作的，所以执行擦除操作时所设置的目标地址的低 9 位是无意义的。例如：执行擦除命令时，设置地址 0234H/0200H/0300H/03FFH，最终执行擦除的动作都是相同的，都是擦除 0200H~03FFH 这 512 字节。

不同型号内部 EEPROM 的大小及访问地址会存在差异，针对各个型号 EEPROM 的详细大小和地址请参考下表

型号	大小	扇区	IAP 方式读/写/擦除		MOVC 读取	
			起始地址	结束地址	起始地址	结束地址
STC8G1K08	4K	8	0000h	0FFFh	2000h	2FFFh
STC8G1K12	用户自定义[1]					

[1]：STC8G1K12 这个为特殊型号，这个型号的 EEPROM 大小是可用在 ISP 下载时用户自己设置的。如下图所示：



用户可用根据自己的需要在整个 FLASH 空间中规划出任意不超过 FLASH 大小的 EEPROM 空间，但需要注意：**EEPROM 总是从后向前进行规划的**。

例如：STC8G1K12 这个型号的 FLASH 为 12K，此时若用户想分出其中的 4K 作为 EEPROM 使用，则 EEPROM 的物理地址则为 12K 的最后 4K，物理地址为 2000h~2FFFh，当然，用户若使用 IAP 的方式进行访问，目标地址仍然从 0000h 开始，到 0FFFh 结束，当使用 MOVC 读取则需要从 2000h 开始，到 2FFFh 结束。

15.3 范例程序

15.3.1 EEPROM基本操作

汇编代码

```
;测试工作频率为11.0592MHz

IAP_DATA    DATA    0C2H
IAP_ADDRH   DATA    0C3H
IAP_ADDRL   DATA    0C4H
IAP_CMD     DATA    0C5H
IAP_TRIG    DATA    0C6H
IAP_CONTR   DATA    0C7H
IAP_TPS     DATA    0F5H

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H

IAP_IDLE:
            MOV     IAP_CONTR,#0        ;关闭IAP 功能
            MOV     IAP_CMD,#0          ;清除命令寄存器
            MOV     IAP_TRIG,#0         ;清除触发寄存器
            MOV     IAP_ADDRH,#80H      ;将地址设置到非IAP 区域
            MOV     IAP_ADDRL,#0
            RET

IAP_READ:
            MOV     IAP_CONTR,#80H      ;使能IAP
            MOV     IAP_TPS,#12         ;设置擦除等待参数12MHz
            MOV     IAP_CMD,#1          ;设置IAP 读命令
            MOV     IAP_ADDRL,DPL       ;设置IAP 低地址
            MOV     IAP_ADDRH,DPH       ;设置IAP 高地址
            MOV     IAP_TRIG,#5AH       ;写触发命令(0x5a)
            MOV     IAP_TRIG,#0A5H      ;写触发命令(0xa5)
            NOP
            MOV     A,IAP_DATA          ;读取IAP 数据
            LCALL   IAP_IDLE            ;关闭IAP 功能
            RET

IAP_PROGRAM:
            MOV     IAP_CONTR,#80H      ;使能IAP
            MOV     IAP_TPS,#12         ;设置擦除等待参数12MHz
            MOV     IAP_CMD,#2          ;设置IAP 写命令
            MOV     IAP_ADDRL,DPL       ;设置IAP 低地址
            MOV     IAP_ADDRH,DPH       ;设置IAP 高地址
            MOV     IAP_DATA,A          ;写IAP 数据
            MOV     IAP_TRIG,#5AH       ;写触发命令(0x5a)
```

```
            MOV     IAP_TRIG,#0A5H          ;写触发命令(0xa5)
            NOP
            LCALL   IAP_IDLE                ;关闭 IAP 功能
            RET

IAP_ERASE:
            MOV     IAP_CONTR,#80H          ;使能 IAP
            MOV     IAP_TPS,#12             ;设置擦除等待参数 12MHz
            MOV     IAP_CMD,#3              ;设置 IAP 擦除命令
            MOV     IAP_ADDRL,DPL           ;设置 IAP 低地址
            MOV     IAP_ADDRH,DPH           ;设置 IAP 高地址
            MOV     IAP_TRIG,#5AH           ;写触发命令(0x5a)
            MOV     IAP_TRIG,#0A5H          ;写触发命令(0xa5)
            NOP
            LCALL   IAP_IDLE                ;关闭 IAP 功能
            RET

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     DPTR,#0400H
            LCALL   IAP_ERASE
            MOV     DPTR,#0400H
            LCALL   IAP_READ
            MOV     P0,A                    ;P0=0FFH
            MOV     DPTR,#0400H
            MOV     A,#12H
            LCALL   IAP_PROGRAM
            MOV     DPTR,#0400H
            LCALL   IAP_READ
            MOV     P1,A                    ;P1=12H

            SJMP    $

            END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sfr     IAP_DATA    =   0xC2;
sfr     IAP_ADDRH   =   0xC3;
sfr     IAP_ADDRL   =   0xC4;
```

```
sfr         IAP_CMD     =   0xC5;
sfr         IAP_TRIG    =   0xC6;
sfr         IAP_CONTR   =   0xC7;
sfr         IAP_TPS     =   0xF5;

void IapIdle()
{
    IAP_CONTR = 0;                  //关闭 IAP 功能
    IAP_CMD = 0;                    //清除命令寄存器
    IAP_TRIG = 0;                   //清除触发寄存器
    IAP_ADDRH = 0x80;               //将地址设置到非 IAP 区域
    IAP_ADDRL = 0;
}

char IapRead(int addr)
{
    char dat;

    IAP_CONTR = 0x80;               //使能 IAP
    IAP_TPS = 12;                   //设置擦除等待参数 12MHz
    IAP_CMD = 1;                    //设置 IAP 读命令
    IAP_ADDRL = addr;               //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;          //设置 IAP 高地址
    IAP_TRIG = 0x5a;                //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                //写触发命令(0xa5)
    _nop_();
    dat = IAP_DATA;                 //读 IAP 数据
    IapIdle();                      //关闭 IAP 功能

    return dat;
}

void IapProgram(int addr, char dat)
{
    IAP_CONTR = 0x80;               //使能 IAP
    IAP_TPS = 12;                   //设置擦除等待参数 12MHz
    IAP_CMD = 2;                    //设置 IAP 写命令
    IAP_ADDRL = addr;               //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;          //设置 IAP 高地址
    IAP_DATA = dat;                 //写 IAP 数据
    IAP_TRIG = 0x5a;                //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                //写触发命令(0xa5)
    _nop_();
    IapIdle();                      //关闭 IAP 功能
}

void IapErase(int addr)
{
    IAP_CONTR = 0x80;               //使能 IAP
    IAP_TPS = 12;                   //设置擦除等待参数 12MHz
    IAP_CMD = 3;                    //设置 IAP 擦除命令
    IAP_ADDRL = addr;               //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;          //设置 IAP 高地址
    IAP_TRIG = 0x5a;                //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                //写触发命令(0xa5)
    _nop_();                        //
    IapIdle();                      //关闭 IAP 功能
}
```

```
void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IapErase(0x0400);
    P0 = IapRead(0x0400);                    //P0=0xff
    IapProgram(0x0400, 0x12);
    P1 = IapRead(0x0400);                    //P1=0x12

    while (1);
}
```

15.3.2 使用MOVC读取EEPROM

汇编代码

```
;测试工作频率为11.0592MHz

IAP_DATA    DATA    0C2H
IAP_ADDRH   DATA    0C3H
IAP_ADDRL   DATA    0C4H
IAP_CMD     DATA    0C5H
IAP_TRIG    DATA    0C6H
IAP_CONTR   DATA    0C7H
IAP_TPS     DATA    0F5H

IAP_OFFSET  EQU     2000H               ;STC8G1K08

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H

IAP_IDLE:
            MOV     IAP_CONTR,#0        ;关闭IAP 功能
            MOV     IAP_CMD,#0          ;清除命令寄存器
            MOV     IAP_TRIG,#0         ;清除触发寄存器
            MOV     IAP_ADDRH,#80H      ;将地址设置到非IAP 区域
            MOV     IAP_ADDRL,#0
            RET

IAP_READ:
            MOV     A,#LOW IAP_OFFSET   ;使用MOVC 读取EEPROM 需要加上相应的偏移
            ADD     A,DPL
            MOV     DPL,A
            MOV     A,@HIGH IAP_OFFSET
            ADDC    A,DPH
```

```
        MOV     DPH,A
        CLR     A
        MOVC    A,@A+DPTR           ;使用 MOVC 读取数据
        RET

IAP_PROGRAM:
        MOV     IAP_CONTR,#80H      ;使能 IAP
        MOV     IAP_TPS,#12         ;设置擦除等待参数 12MHz
        MOV     IAP_CMD,#2          ;设置 IAP 写命令
        MOV     IAP_ADDRL,DPL       ;设置 IAP 低地址
        MOV     IAP_ADDRH,DPH       ;设置 IAP 高地址
        MOV     IAP_DATA,A          ;写 IAP 数据
        MOV     IAP_TRIG,#5AH       ;写触发命令(0x5a)
        MOV     IAP_TRIG,#0A5H      ;写触发命令(0xa5)
        NOP
        LCALL   IAP_IDLE            ;关闭 IAP 功能
        RET

IAP_ERASE:
        MOV     IAP_CONTR,#80H      ;使能 IAP
        MOV     IAP_TPS,#12         ;设置擦除等待参数 12MHz
        MOV     IAP_CMD,#3          ;设置 IAP 擦除命令
        MOV     IAP_ADDRL,DPL       ;设置 IAP 低地址
        MOV     IAP_ADDRH,DPH       ;设置 IAP 高地址
        MOV     IAP_TRIG,#5AH       ;写触发命令(0x5a)
        MOV     IAP_TRIG,#0A5H      ;写触发命令(0xa5)
        NOP
        LCALL   IAP_IDLE            ;关闭 IAP 功能
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     DPTR,#0400H
        LCALL   IAP_ERASE
        MOV     DPTR,#0400H
        LCALL   IAP_READ
        MOV     P0,A                ;P0=0FFH
        MOV     DPTR,#0400H
        MOV     A,#12H
        LCALL   IAP_PROGRAM
        MOV     DPTR,#0400H
        LCALL   IAP_READ
        MOV     P1,A                ;P1=12H

        SJMP    $

        END
```

C 语言代码

```
//测试工作频率为 11.0592MHz
```

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

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sfr         IAP_DATA    =   0xC2;
sfr         IAP_ADDRH   =   0xC3;
sfr         IAP_ADDRL   =   0xC4;
sfr         IAP_CMD     =   0xC5;
sfr         IAP_TRIG    =   0xC6;
sfr         IAP_CONTR   =   0xC7;
sfr         IAP_TPS     =   0xF5;

#define     IAP_OFFSET  0x2000H                     //STC8G1K08

void IapIdle()
{
    IAP_CONTR = 0;                                  //关闭 IAP 功能
    IAP_CMD = 0;                                    //清除命令寄存器
    IAP_TRIG = 0;                                   //清除触发寄存器
    IAP_ADDRH = 0x80;                               //将地址设置到非 IAP 区域
    IAP_ADDRL = 0;
}

char IapRead(int addr)
{
    addr += IAP_OFFSET;                             //使用 MOVC 读取 EEPROM 需要加上相应的偏移
    return *(char code *)(addr);                    //使用 MOVC 读取数据
}

void IapProgram(int addr, char dat)
{
    IAP_CONTR = 0x80;                               //使能 IAP
    IAP_TPS = 12;                                   //设置擦除等待参数 12MHz
    IAP_CMD = 2;                                    //设置 IAP 写命令
    IAP_ADDRL = addr;                               //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;                          //设置 IAP 高地址
    IAP_DATA = dat;                                 //写 IAP 数据
    IAP_TRIG = 0x5a;                                //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                                //写触发命令(0xa5)
    _nop_();
    IapIdle();                                      //关闭 IAP 功能
}

void IapErase(int addr)
{
    IAP_CONTR = 0x80;                               //使能 IAP
    IAP_TPS = 12;  //设置擦除等待参数 12MHz
    IAP_CMD = 3;  //设置 IAP 擦除命令
    IAP_ADDRL = addr;                               //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;                          //设置 IAP 高地址
    IAP_TRIG = 0x5a;                                //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                                //写触发命令(0xa5)
    _nop_();                                        //
```

```
    IapIdle();                              //关闭 IAP 功能
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IapErase(0x0400);
    P0 = IapRead(0x0400);                   //P0=0xff
    IapProgram(0x0400, 0x12);
    P1 = IapRead(0x0400);                   //P1=0x12

    while (1);
}
```

15.3.3 使用串口送出EEPROM数据

汇编代码

```
;测试工作频率为11.0592MHz

AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H

IAP_DATA    DATA    0C2H
IAP_ADDRH   DATA    0C3H
IAP_ADDRL   DATA    0C4H
IAP_CMD     DATA    0C5H
IAP_TRIG    DATA    0C6H
IAP_CONTR   DATA    0C7H
IAP_TPS     DATA    0F5H

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H

UART_INIT:
            MOV     SCON,#5AH
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#15H
            RET

UART_SEND:
            JNB     TI,$
```

```
        CLR     TI
        MOV     SBUF,A
        RET

IAP_IDLE:
        MOV     IAP_CONTR,#0        ;关闭 IAP 功能
        MOV     IAP_CMD,#0          ;清除命令寄存器
        MOV     IAP_TRIG,#0         ;清除触发寄存器
        MOV     IAP_ADDRH,#80H      ;将地址设置到非 IAP 区域
        MOV     IAP_ADDRL,#0
        RET

IAP_READ:
        MOV     IAP_CONTR,#80H      ;使能 IAP
        MOV     IAP_TPS,#12         ;设置擦除等待参数 12MHz
        MOV     IAP_CMD,#1          ;设置 IAP 读命令
        MOV     IAP_ADDRL,DPL       ;设置 IAP 低地址
        MOV     IAP_ADDRH,DPH       ;设置 IAP 高地址
        MOV     IAP_TRIG,#5AH       ;写触发命令(0x5a)
        MOV     IAP_TRIG,#0A5H      ;写触发命令(0xa5)
        NOP
        MOV     A,IAP_DATA          ;读取 IAP 数据
        LCALL   IAP_IDLE            ;关闭 IAP 功能
        RET

IAP_PROGRAM:
        MOV     IAP_CONTR,#80H      ;使能 IAP
        MOV     IAP_TPS,#12         ;设置擦除等待参数 12MHz
        MOV     IAP_CMD,#2          ;设置 IAP 写命令
        MOV     IAP_ADDRL,DPL       ;设置 IAP 低地址
        MOV     IAP_ADDRH,DPH       ;设置 IAP 高地址
        MOV     IAP_DATA,A          ;写 IAP 数据
        MOV     IAP_TRIG,#5AH       ;写触发命令(0x5a)
        MOV     IAP_TRIG,#0A5H      ;写触发命令(0xa5)
        NOP
        LCALL   IAP_IDLE            ;关闭 IAP 功能
        RET

IAP_ERASE:
        MOV     IAP_CONTR,#80H      ;使能 IAP
        MOV     IAP_TPS,#12         ;设置擦除等待参数 12MHz
        MOV     IAP_CMD,#3          ;设置 IAP 擦除命令
        MOV     IAP_ADDRL,DPL       ;设置 IAP 低地址
        MOV     IAP_ADDRH,DPH       ;设置 IAP 高地址
        MOV     IAP_TRIG,#5AH       ;写触发命令(0x5a)
        MOV     IAP_TRIG,#0A5H      ;写触发命令(0xa5)
        NOP
        LCALL   IAP_IDLE            ;关闭 IAP 功能
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H
```

```
        LCALL   UART_INIT
        MOV     DPTR,#0400H
        LCALL   IAP_ERASE
        MOV     DPTR,#0400H
        LCALL   IAP_READ
        LCALL   UART_SEND
        MOV     DPTR,#0400H
        MOV     A,#12H
        LCALL   IAP_PROGRAM
        MOV     DPTR,#0400H
        LCALL   IAP_READ
        LCALL   UART_SEND

        SJMP    $

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sfr     AUXR    =   0x8e;
sfr     T2H     =   0xd6;
sfr     T2L     =   0xd7;

sfr     IAP_DATA    =   0xC2;
sfr     IAP_ADDRH   =   0xC3;
sfr     IAP_ADDRL   =   0xC4;
sfr     IAP_CMD     =   0xC5;
sfr     IAP_TRIG    =   0xC6;
sfr     IAP_CONTR   =   0xC7;
sfr     IAP_TPS     =   0xF5;

void UartInit()
{
    SCON = 0x5a;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x15;
}

void UartSend(char dat)
{
    while (!TI);
    TI = 0;
    SBUF = dat;
```

```
}

void IapIdle()
{
    IAP_CONTR = 0;                  //关闭 IAP 功能
    IAP_CMD = 0;                    //清除命令寄存器
    IAP_TRIG = 0;                   //清除触发寄存器
    IAP_ADDRH = 0x80;               //将地址设置到非 IAP 区域
    IAP_ADDRL = 0;
}

char IapRead(int addr)
{
    char dat;

    IAP_CONTR = 0x80;               //使能 IAP
    IAP_TPS = 12;                   //设置擦除等待参数 12MHz
    IAP_CMD = 1;                    //设置 IAP 读命令
    IAP_ADDRL = addr;               //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;          //设置 IAP 高地址
    IAP_TRIG = 0x5a;                //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                //写触发命令(0xa5)
    _nop_();
    dat = IAP_DATA;                 //读 IAP 数据
    IapIdle();                      //关闭 IAP 功能

    return dat;
}

void IapProgram(int addr, char dat)
{
    IAP_CONTR = 0x80;               //使能 IAP
    IAP_TPS = 12;                   //设置擦除等待参数 12MHz
    IAP_CMD = 2;                    //设置 IAP 写命令
    IAP_ADDRL = addr;               //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;          //设置 IAP 高地址
    IAP_DATA = dat;                 //写 IAP 数据
    IAP_TRIG = 0x5a;                //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                //写触发命令(0xa5)
    _nop_();
    IapIdle();                      //关闭 IAP 功能
}

void IapErase(int addr)
{
    IAP_CONTR = 0x80;               //使能 IAP
    IAP_TPS = 12;                   //设置擦除等待参数 12MHz
    IAP_CMD = 3;                    //设置 IAP 擦除命令
    IAP_ADDRL = addr;               //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;          //设置 IAP 高地址
    IAP_TRIG = 0x5a;                //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                //写触发命令(0xa5)
    _nop_();                        //
    IapIdle();                      //关闭 IAP 功能
}

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

```
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    IapErase(0x0400);
    UartSend(IapRead(0x0400));
    IapProgram(0x0400, 0x12);
    UartSend(IapRead(0x0400));

    while (1);
}
```

16 ADC模数转换

STC8G 系列单片机内部集成了一个 10 位 15 通道的高速 A/D 转换器（**注：第 16 通道只能用于检测内部参考电压，参考电压值出厂时校准为 1.19V，由于制造误差，实际电压值可能在 1.178V～1.202V 之间**）。ADC 的时钟频率为系统频率 2 分频再经过用户设置的分频系数进行再次分频（ADC 的时钟频率范围为 SYSclk/2/1～SYSclk/2/16）。

ADC 转换结果的数据格式有两种：左对齐和右对齐。可方便用户程序进行读取和引用。

16.1 ADC相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
ADC_CONTR	ADC 控制寄存器	BCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]				000x,0000
ADC_RES	ADC 转换结果高位寄存器	BDH									0000,0000
ADC_RESL	ADC 转换结果低位寄存器	BEH									0000,0000
ADCCFG	ADC 配置寄存器	DEH	-	-	RESFMT	-	SPEED[3:0]				xx0x,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
ADCTIM	ADC 时序控制寄存器	FEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]					0010,1010

ADC 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADC_CONTR	BCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]			

ADC_POWER：ADC 电源控制位

0：关闭 ADC 电源

1：打开 ADC 电源。

建议进入空闲模式和掉电模式前将 ADC 电源关闭，以降低功耗

ADC_START：ADC 转换启动控制位。写入 1 后开始 ADC 转换，转换完成后硬件自动将此位清零。

0：无影响。即使 ADC 已经开始转换工作，写 0 也不会停止 A/D 转换。

1：开始 ADC 转换，转换完成后硬件自动将此位清零。

ADC_FLAG：ADC 转换结束标志位。当 ADC 完成一次转换后，硬件会自动将此位置 1，并向 CPU 提出中断请求。此标志位必须软件清零。

ADC_CHS[3:0]：ADC 模拟通道选择位

ADC_CHS[3:0]	ADC 通道	ADC_CHS[3:0]	ADC 通道
0000	P1.0	1000	P3.0
0001	P1.1	1001	P3.1
0010	P1.2	1010	P3.2
0011	P1.3	1011	P3.3
0100	P1.4	1100	P3.4
0101	P1.5	1101	P3.5
0110	P1.6	1110	P3.6

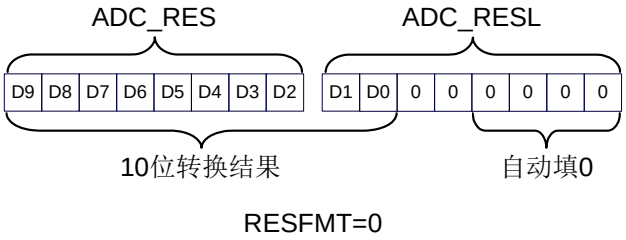
0111	P1.7	1111	测试内部 1.19V

ADC 配置寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADCCFG	DEH	-	-	RESFMT	-	SPEED[3:0]			

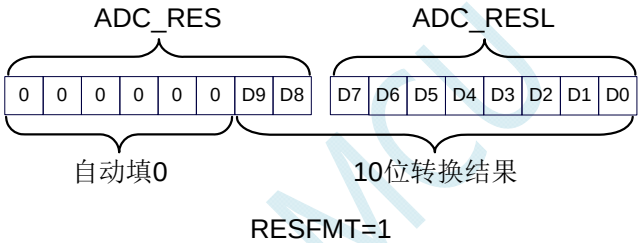
RESFMT：ADC 转换结果格式控制位

0：转换结果左对齐。ADC_RES 保存结果的高 8 位，ADC_RESL 保存结果的低 2 位。格式如下：



RESFMT=0

1：转换结果右对齐。ADC_RES 保存结果的高 2 位，ADC_RESL 保存结果的低 8 位。格式如下：



RESFMT=1

SPEED[3:0]：设置 ADC 时钟｛F_{ADC}＝SYSclk/2/(SPEED+1)｝

SPEED[3:0]	ADC 时钟频率
0000	SYSclk/2/1
0001	SYSclk/2/2
0010	SYSclk/2/3
...	. . .
1101	SYSclk/2/14
1110	SYSclk/2/15
1111	SYSclk/2/16

ADC 转换结果寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	BDH								
ADC_RESL	BEH								

当 A/D 转换完成后，10 为的转换结果会自动保存到 ADC_RES 和 ADC_RESL 中。保存结果的数据格式请参考 ADC_CFG 寄存器中的 RESFMT 设置。

ADC 时序控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADCTIM	FEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]				

CSSETUP：ADC 通道选择时间控制 T_{setup}

CSSETUP	ADC 时钟数
0	**1（默认值）**
1	2

CSHOLD[1:0]：ADC 通道选择保持时间控制 T_{hold}

CSHOLD[1:0]	ADC 时钟数
00	1
01	**2（默认值）**
10	3
11	4

SMPDUTY[4:0]：ADC 模拟信号采用时间控制 T_{duty}

SMPDUTY[4:0]	ADC 时钟数
00000	1
00001	2
...	...
01010	**11（默认值）**
...	...
11110	31
11111	32

16.2 范例程序

16.2.1 ADC基本操作（查询方式）

汇编代码

```
;测试工作频率为11.0592MHz

ADC_CONTR   DATA    0BCH
ADC_RES     DATA    0BDH
ADC_RESL    DATA    0BEH
ADCCFG      DATA    0DEH

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P1M0,#00H           ;设置P1.0为ADC口
            MOV     P1M1,#01H
            MOV     ADCCFG,#0FH         ;设置ADC时钟为系统时钟/2/16/16
            MOV     ADC_CONTR,#80H      ;使能ADC模块
LOOP:
            ORL     ADC_CONTR,#40H      ;启动AD转换
            NOP
            NOP
            MOV     A,ADC_CONTR         ;查询ADC完成标志
            JNB     ACC.5,$-2
            ANL     ADC_CONTR,#NOT 20H  ;清完成标志
            MOV     P2,ADC_RES          ;读取ADC结果

            SJMP    LOOP

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr         ADC_CONTR   =           0xbc;
```

```
sfr     ADC_RES     =     0xbd;
sfr     ADC_RESL    =     0xbe;
sfr     ADCCFG      =     0xde;

sfr     P1M1        =     0x91;
sfr     P1M0        =     0x92;
sfr     P3M1        =     0xb1;
sfr     P3M0        =     0xb2;
sfr     P5M1        =     0xc9;
sfr     P5M0        =     0xca;

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P1M0 = 0x00;                        //设置 P1.0 为 ADC 口
    P1M1 = 0x01;
    ADCCFG = 0x0f;                      //设置 ADC 时钟为系统时钟/2/16/16
    ADC_CONTR = 0x80;                   //使能 ADC 模块

    while (1)
    {
        ADC_CONTR |= 0x40;              //启动 AD 转换
        _nop_();
        _nop_();
        while (!(ADC_CONTR & 0x20));    //查询 ADC 完成标志
        ADC_CONTR &= ~0x20;             //清完成标志
        P2 = ADC_RES;                   //读取 ADC 结果
    }
}
```

16.2.2 ADC基本操作（中断方式）

汇编代码

```
;测试工作频率为 11.0592MHz

ADC_CONTR   DATA    0BCH
ADC_RES     DATA    0BDH
ADC_RESL    DATA    0BEH
ADCCFG      DATA    0DEH

EADC        BIT     IE.5

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     002BH
```

```
            LJMP        ADCISR

            ORG         0100H
ADCISR:
            ANL         ADC_CONTR,#NOT 20H      ;清完成标志
            MOV         P2,ADC_RES              ;读取 ADC 结果
            ORL         ADC_CONTR,#40H          ;继续 AD 转换
            RETI

MAIN:
            MOV         SP, #5FH
            MOV         P1M0, #00H
            MOV         P1M1, #00H
            MOV         P3M0, #00H
            MOV         P3M1, #00H
            MOV         P5M0, #00H
            MOV         P5M1, #00H

            MOV         P1M0,#00H               ;设置 P1.0 为 ADC 口
            MOV         P1M1,#01H
            MOV         ADCCFG,#0FH             ;设置 ADC 时钟为系统时钟/2/16/16
            MOV         ADC_CONTR,#80H          ;使能 ADC 模块
            SETB        EADC                    ;使能 ADC 中断
            SETB        EA
            ORL         ADC_CONTR,#40H          ;启动 AD 转换

            SJMP        $

            END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr         ADC_CONTR   =   0xbc;
sfr         ADC_RES     =   0xbd;
sfr         ADC_RESL    =   0xbe;
sfr         ADCCFG      =   0xde;

sbit        EADC        =   IE^5;

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

void ADC_Isr() interrupt 5
{
    ADC_CONTR &= ~0x20;                         //清中断标志
    P2 = ADC_RES;                               //读取 ADC 结果
    ADC_CONTR |= 0x40;                          //继续 AD 转换
}

void main()
```

```
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P1M0 = 0x00;                    //设置 P1.0 为 ADC 口
    P1M1 = 0x01;
    ADCCFG = 0x0f;                  //设置 ADC 时钟为系统时钟/2/16/16
    ADC_CONTR = 0x80;               //使能 ADC 模块
    EADC = 1;                       //使能 ADC 中断
    EA = 1;
    ADC_CONTR |= 0x40;              //启动 AD 转换

    while (1);
}
```

16.2.3 格式化ADC转换结果

汇编代码

```
;测试工作频率为 11.0592MHz

ADC_CONTR   DATA    0BCH
ADC_RES     DATA    0BDH
ADC_RESL    DATA    0BEH
ADCCFG      DATA    0DEH

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P1M0,#00H       ;设置 P1.0 为 ADC 口
            MOV     P1M1,#01H
            MOV     ADCCFG,#0FH     ;设置 ADC 时钟为系统时钟/2/16/16
            MOV     ADC_CONTR,#80H  ;使能 ADC 模块

            ORL     ADC_CONTR,#40H  ;启动 AD 转换
            NOP
            NOP
            MOV     A,ADC_CONTR     ;查询 ADC 完成标志
```

```
            JNB     ACC.5,$-2
            ANL     ADC_CONTR,#NOT 20H      ;清完成标志

            MOV     ADCCFG,#00H             ;设置结果左对齐
            MOV     A,ADC_RES               ;A 存储 ADC 的 10 位结果的高 8 位
            MOV     B,ADC_RESL              ;B[7:6]存储 ADC 的 10 位结果的低 2 位,B[5:0]为 0

;           MOV     ADCCFG,#20H             ;设置结果右对齐
;           MOV     A,ADC_RES               ;A[3:0]存储 ADC 的 10 位结果的高 2 位,A[7:2]为 0
;           MOV     B,ADC_RESL              ;B 存储 ADC 的 10 位结果的低 8 位

            SJMP    $

            END
```

C 语言代码

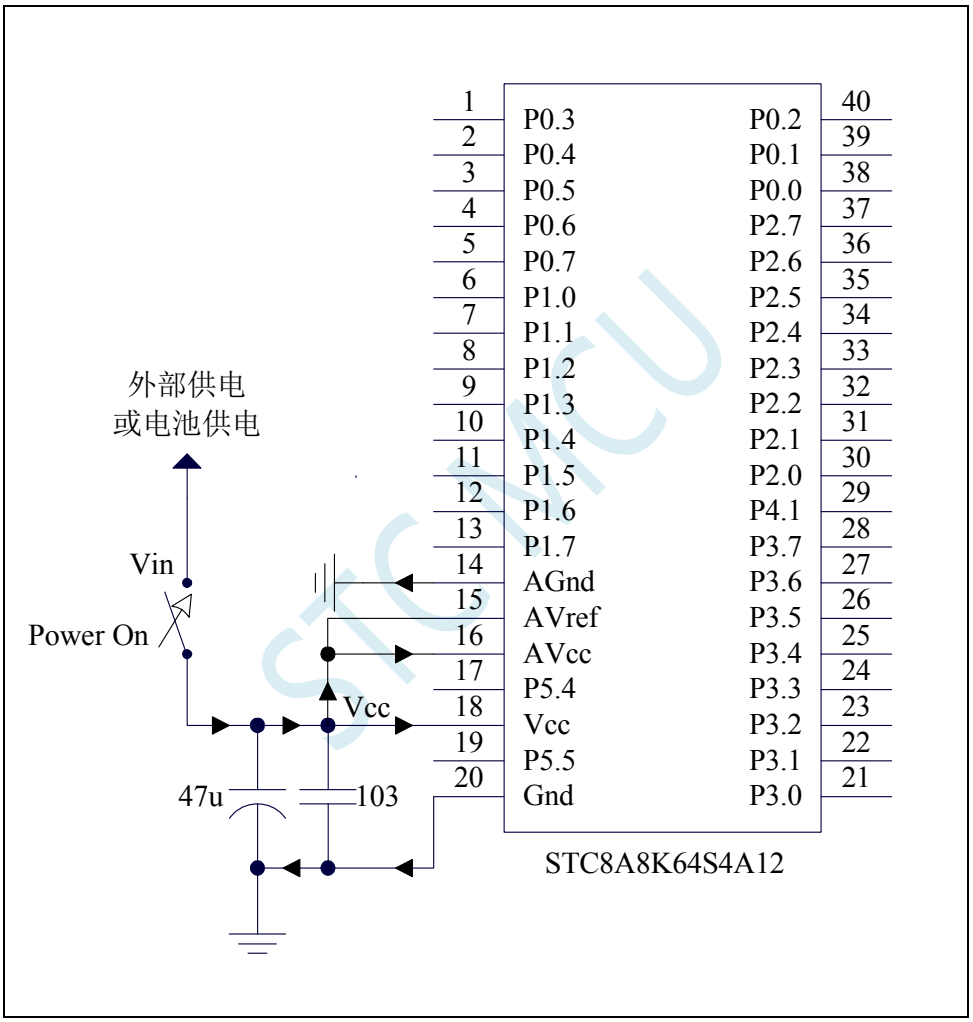
```
//测试工作频率为 11.0592MHz

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

sfr     ADC_CONTR   =   0xbc;
sfr     ADC_RES     =   0xbd;
sfr     ADC_RESL    =   0xbe;
sfr     ADCCFG      =   0xde;

sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P1M0 = 0x00;                        //设置 P1.0 为 ADC 口
    P1M1 = 0x01;
    ADCCFG = 0x0f;                      //设置 ADC 时钟为系统时钟/2/16/16
    ADC_CONTR = 0x80;                   //使能 ADC 模块
    ADC_CONTR |= 0x40;                  //启动 AD 转换
    _nop_();
    _nop_();
    while (!(ADC_CONTR & 0x20));        //查询 ADC 完成标志
    ADC_CONTR &= ~0x20;                 //清完成标志

    ADCCFG = 0x00;                      //设置结果左对齐
    ACC = ADC_RES;                      //A 存储 ADC 的 10 位结果的高 8 位
    B = ADC_RESL;                       //B[7:6]存储 ADC 的 10 位结果的低 2 位,B[5:0]为 0

//  ADCCFG = 0x20;                      //设置结果右对齐
//  ACC = ADC_RES;                      //A[1:0]存储 ADC 的 10 位结果的高 2 位,A[7:2]为 0
```

```
//      B = ADC_RESL;                                   //B 存储 ADC 的 10 位结果的低 8 位

        while (1);
}
```

16.2.4 利用ADC第 16 通道测量外部电压或电池电压

STC8 系列 ADC 的第 16 通道是用来测试内部 BandGap 参考电压的，由于内部 BandGap 参考电压很稳定，约为 1.19V，且不会随芯片的工作电压的改变而变化，所以可以通过测量内部 BandGap 参考电压，然后通过 ADC 的值便可反推出外部电压或外部电池电压。

下图为参考线路图:



C 语言代码

```
//测试工作频率为11.0592MHz

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

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

sfr        AUXR        =    0x8e;
```

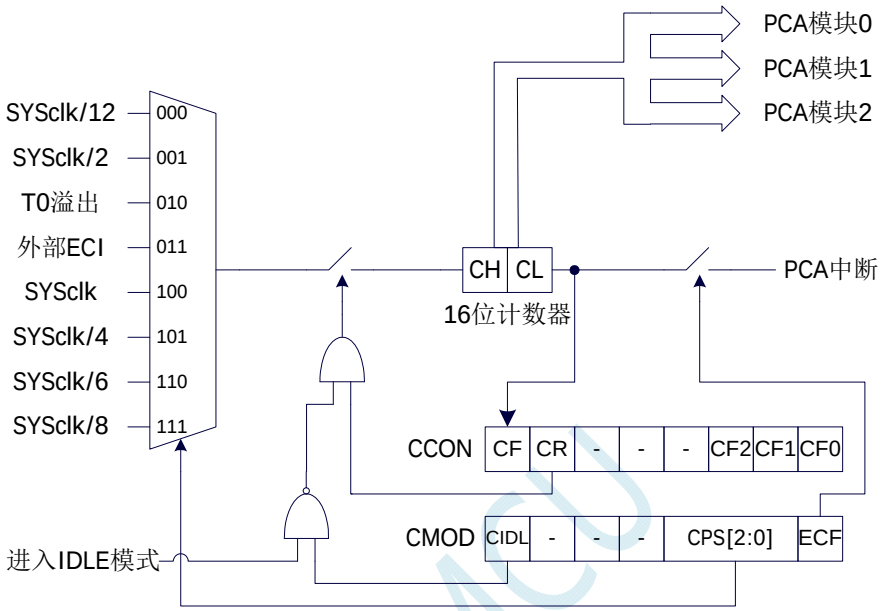
```
sfr         ADC_CONTR   =   0xbc;
sfr         ADC_RES     =   0xbd;
sfr         ADC_RESL    =   0xbe;
sfr         ADCCFG      =   0xde;

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

int         *BGV;                           //内部 Bandgap 电压值存放在 idata 中
                                            //idata 的 EFH 地址存放高字节
                                            //idata 的 F0H 地址存放低字节
                                            //电压单位为毫伏(mV)
bit         busy;

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

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

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

void ADCInit()
{
    ADCCFG = 0x2f;                          //设置 ADC 时钟为系统时钟/2/16/16
    ADC_CONTR = 0x8f;                       //使能 ADC 模块,并选择第 16 通道
}

int    ADCRead()
{
    int res;
```

```
    ADC_CONTR |= 0x40;                          //启动 AD 转换
    _nop_();
    _nop_();
    while (!(ADC_CONTR & 0x20));                //查询 ADC 完成标志
    ADC_CONTR &= ~0x20;                         //清完成标志
    res = (ADC_RES << 8) | ADC_RESL;            //读取 ADC 结果

    return res;
}

void main()
{
    int res;
    int vcc;
    int i;

    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    BGV = (int idata *)0xef;
    ADCInit();                                  //ADC 初始化
    UartInit();                                 //串口初始化

    ES = 1;
    EA = 1;

    ADCRead();
    ADCRead();                                  //前两个数据丢弃
    res = 0;
    for (i=0; i<8; i++)
    {
        res += ADCRead();                       //读取 8 次数据
    }
    res >>= 3;                                  //取平均值

    vcc = (int)(4095L * *BGV / res);            //计算 VREF 管脚电压,即电池电压
                                                //注意,此电压的单位为毫伏(mV)
    UartSend(vcc >> 8);                         //输出电压值到串口
    UartSend(vcc);

    while (1);
}
```

17 PCA/CCP/PWM应用

STC8G 系列单片机内部集成了 3 组可编程计数器阵列（PCA/CCP/PWM）模块，可用于软件定时器、外部脉冲捕获、高速脉冲输出和 PWM 脉宽调制输出。

PCA 内部含有一个特殊的 16 位计数器，3 组 PCA 模块均与之相连接。PCA 计数器的结构图如下：



PCA计数器结构图

17.1 PCA相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CCON	PCA 控制寄存器	D8H	CF	CR	-	-	-	CCF2	CCF1	CCF0	00xx,x000
CMOD	PCA 模式寄存器	D9H	CIDL	-	-	-	CPS[2:0]			ECF	0xxx,0000
CCAPM0	PCA 模块 0 模式控制寄存器	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAPM1	PCA 模块 1 模式控制寄存器	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CCAPM2	PCA 模块 2 模式控制寄存器	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
CL	PCA 计数器低字节	E9H									0000,0000
CCAP0L	PCA 模块 0 低字节	EAH									0000,0000
CCAP1L	PCA 模块 1 低字节	EBH									0000,0000
CCAP2L	PCA 模块 2 低字节	ECH									0000,0000
PCA_PWM0	PCA0 的 PWM 模式寄存器	F2H	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L	0000,0000
PCA_PWM1	PCA1 的 PWM 模式寄存器	F3H	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L	0000,0000
PCA_PWM2	PCA2 的 PWM 模式寄存器	F4H	EBS2[1:0]		XCCAP2H[1:0]		XCCAP2L[1:0]		EPC2H	EPC2L	0000,0000
CH	PCA 计数器高字节	F9H									0000,0000
CCAP0H	PCA 模块 0 高字节	FAH									0000,0000
CCAP1H	PCA 模块 1 高字节	FBH									0000,0000
CCAP2H	PCA 模块 2 高字节	FCH									0000,0000

PCA 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCON	D8H	CF	CR	-	-	-	CCF2	CCF1	CCF0

CF：PCA 计数器溢出中断标志。当 PCA 的 16 位计数器计数发生溢出时，硬件自动将此位置 1，并向 CPU 提出中断请求。此标志位需要软件清零。

CR：PCA 计数器允许控制位。

0：停止 PCA 计数

1：启动 PCA 计数

CCFn（n=0,1,2）：PCA 模块中断标志。当 PCA 模块发生匹配或者捕获时，硬件自动将此位置 1，并向 CPU 提出中断请求。此标志位需要软件清零。

PCA 模式寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	D9H	CIDL	-	-	-	CPS[2:0]			ECF

CIDL：空闲模式下是否停止 PCA 计数。

0：空闲模式下 PCA 继续计数

1：空闲模式下 PCA 停止计数

CPS[2:0]：PCA 计数脉冲源选择位

CPS[2:0]	PCA 的输入时钟源
000	系统时钟/12
001	系统时钟/2
010	定时器 0 的溢出脉冲
011	ECI 脚的外部输入时钟
100	系统时钟
101	系统时钟/4
110	系统时钟/6
111	系统时钟 8

ECF：PCA 计数器溢出中断允许位。

0：禁止 PCA 计数器溢出中断

1：使能 PCA 计数器溢出中断

PCA 计数器寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CL	E9H								
CH	F9H								

由 CL 和 CH 两个字节组合成一个 16 位计数器，CL 为低 8 位计数器，CH 为高 8 位计数器。每个 PCA 时钟 16 位计数器自动加 1。

PCA 模块模式控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCAPM0	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0
CCAPM1	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1
CCAPM2	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2

ECOMn：允许 PCA 模块 n 的比较功能
CCAPPn：允许 PCA 模块 n 进行上升沿捕获
CCAPNn：允许 PCA 模块 n 进行下降沿捕获
MATn：允许 PCA 模块 n 的匹配功能
TOGn：允许 PCA 模块 n 的高速脉冲输出功能
PWMn：允许 PCA 模块 n 的脉宽调制输出功能
ECCFn：允许 PCA 模块 n 的匹配/捕获中断

PCA 模块模式捕获值/比较值寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCAP0L	EAH								
CCAP1L	EBH								
CCAP2L	ECH								
CCAP0H	FAH								
CCAP1H	FBH								
CCAP2H	FCH								

当 PCA 模块捕获功能使能时，CCAPnL 和 CCAPnH 用于保存发生捕获时的 PCA 的计数值（CL 和 CH）；当 PCA 模块比较功能使能时，PCA 控制器会将当前 CL 和 CH 中的计数值与保存在 CCAPnL 和 CCAPnH 中的值进行比较，并给出比较结果；当 PCA 模块匹配功能使能时，PCA 控制器会将当前 CL 和 CH 中的计数值与保存在 CCAPnL 和 CCAPnH 中的值进行比较，看是否匹配（相等），并给出匹配结果。

PCA 模块 PWM 模式控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PCA_PWM0	F2H	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L
PCA_PWM1	F3H	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L
PCA_PWM2	F4H	EBS2[1:0]		XCCAP2H[1:0]		XCCAP2L[1:0]		EPC2H	EPC2L

EBSn[1:0]：PCA 模块 n 的 PWM 位数控制

EBSn[1:0]	PWM 位数	重载值	比较值
00	8 位 PWM	{EPCnH, CCAPnH[7:0]}	{EPCnL, CCAPnL[7:0]}
01	7 位 PWM	{EPCnH, CCAPnH[6:0]}	{EPCnL, CCAPnL[6:0]}
10	6 位 PWM	{EPCnH, CCAPnH[5:0]}	{EPCnL, CCAPnL[5:0]}
11	10 位 PWM	{EPCnH, XCCAPnH[1:0], CCAPnH[7:0]}	{EPCnL, XCCAPnL[1:0], CCAPnL[7:0]}

XCCAPnH[1:0]：10 位 PWM 的第 9 位和第 10 位的重载值
XCCAPnL[1:0]：10 位 PWM 的第 9 位和第 10 位的比较值
EPCnH：PWM 模式下，重载值的最高位（8 为 PWM 的第 9 位，7 位 PWM 的第 8 位，6 位 PWM 的第 7 位，10 位 PWM 的第 11 位）
EPCnL：PWM 模式下，比较值的最高位（8 为 PWM 的第 9 位，7 位 PWM 的第 8 位，6 位 PWM 的第 7 位，10 位 PWM 的第 11 位）
注意：在更新 10 位 PWM 的重载值时，必须先写高两位 XCCAPnH[1:0]，再写低 8 位 CCAPnH[7:0]。

17.2 PCA工作模式

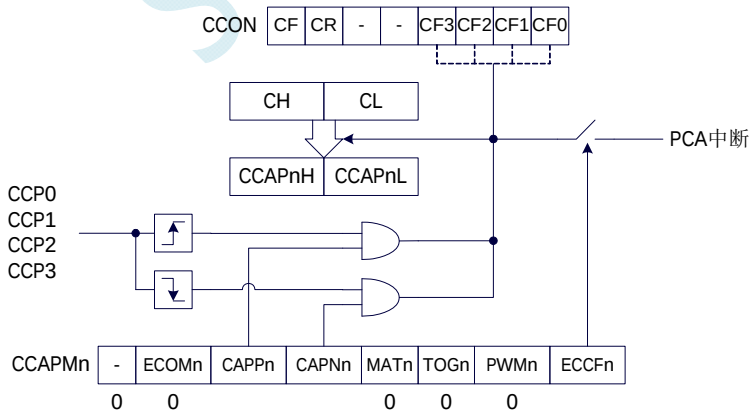
STC8 系列单片机共有 4 组 PCA 模块，每组模块都可独立设置工作模式。模式设置如下所示：

CCAPMn								模块功能
-	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	
-	0	0	0	0	0	0	0	无操作
-	1	0	0	0	0	1	0	6/7/8/10 位 PWM 模式，无中断
-	1	1	0	0	0	1	1	6/7/8/10 位 PWM 模式，产生上升沿中断
-	1	0	1	0	0	1	1	6/7/8/10 位 PWM 模式，产生下降沿中断
-	1	1	1	0	0	1	1	6/7/8/10 位 PWM 模式，产生边沿中断
-	0	1	0	0	0	0	x	16 位上升沿捕获
-	0	0	1	0	0	0	x	16 位下降沿捕获
-	0	1	1	0	0	0	x	16 位边沿捕获
-	1	0	0	1	0	0	x	16 位软件定时器
-	1	0	0	1	1	0	x	16 为高速脉冲输出

17.2.1 捕获模式

要使一个 PCA 模块工作在捕获模式，寄存器 CCAPMn 中的 CAPNn 和 CAPPn 至少有一位必须置 1（也可两位都置 1）。PCA 模块工作于捕获模式时，对模块的外部 CCP0/CCP1/CCP2 管脚的输入跳变进行采样。当采样到有效跳变时，PCA 控制器立即将 PCA 计数器 CH 和 CL 中的计数值装载到模块的捕获寄存器中 CCAPnL 和 CCAPnH，同时将 CCON 寄存器中相应的 CCFn 置 1。若 CCAPMn 中的 ECCFn 位被设置为 1，将产生中断。由于所有 PCA 模块的中断入口地址是共享的，所以在中断服务程序中需要判断是哪一个模块产生了中断，并注意中断标志位需要软件清零。

PCA 模块工作于捕获模式的结构图如下图所示：

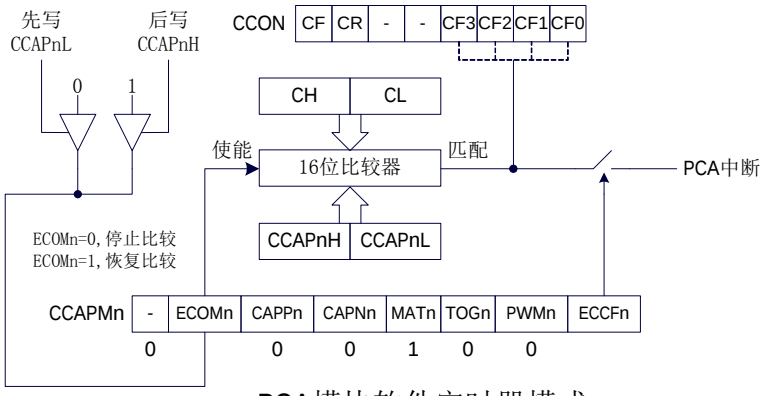


PCA模块捕获模式

17.2.2 软件定时器模式

通过置位 CCAPMn 寄存器的 ECOM 和 MAT 位，可使 PCA 模块用作软件定时器。PCA 计数器值 CL 和 CH 与模块捕获寄存器的值 CCAPnL 和 CCAPnH 相比较，当两者相等时，CCON 中的 CCFn 会被置 1，若 CCAPMn 中的 ECCFn 被设置为 1 时将产生中断。CCFn 标志位需要软件清零。

PCA 模块工作于软件定时器模式的结构图如下图所示：

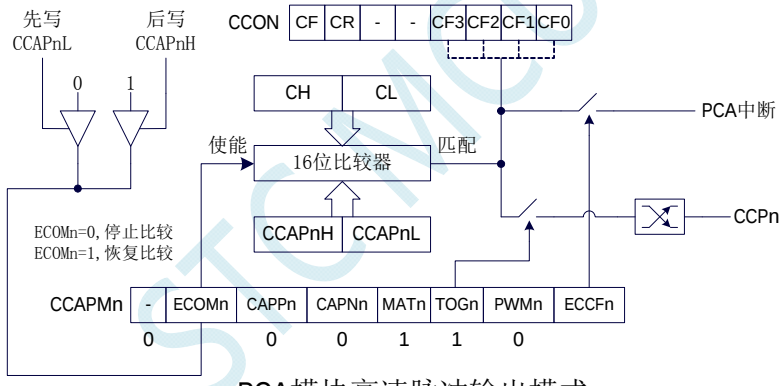


PCA模块软件定时器模式

17.2.3 高速脉冲输出模式

当 PCA 计数器的计数值与模块捕获寄存器的值相匹配时,PCA 模块的 CCPn 输出将发生翻转转。要激活高速脉冲输出模式，CCAPMn 寄存器的 TOGn、MATn 和 ECOMn 位必须都置 1。

PCA 模块工作于高速脉冲输出模式的结构图如下图所示：



PCA模块高速脉冲输出模式

17.2.4 PWM脉宽调制模式

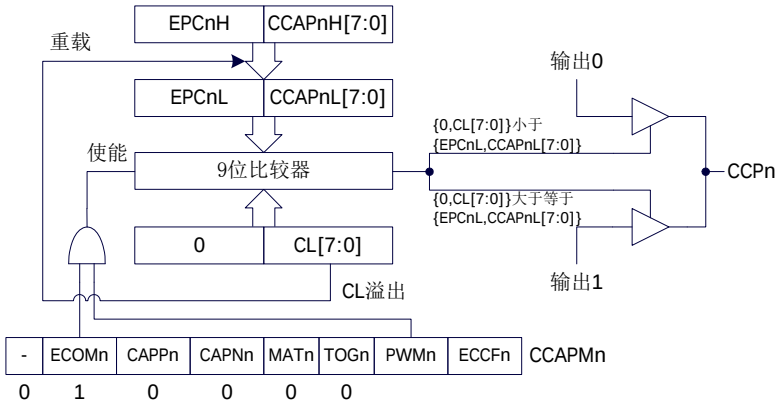
17.2.4.1 8 位PWM模式

脉宽调制是使用程序来控制波形的占空比、周期、相位波形的一种技术，在三相电机驱动、D/A 转换等场合有广泛的应用。STC8 系列单片机的 PCA 模块可以通过设定各自的 PCA_PWMn 寄存器使其工作于 8 位 PWM 或 7 位 PWM 或 6 位 PWM 或 10 位 PWM 模式。要使能 PCA 模块的 PWM 功能，模块寄存器 CCAPMn 的 PWMn 和 ECOMn 位必须置 1。

PCA_PWMn 寄存器中的 EBSn[1:0]设置为 00 时，PCA 模块 n 工作于 8 位 PWM 模式，此时将{0,CL[7:0]}与捕获寄存器{EPCnL,CCAPnL[7:0]}进行比较。当 PCA 模块工作于 8 位 PWM 模式时，由于所有模块共用一个 PCA 计数器，所有它们的输出频率相同。各个模块的输出占空比使用寄存器{EPCnL,CCAPnL[7:0]}进行设置。当{0,CL[7:0]}的值小于{EPCnL,CCAPnL[7:0]}时，输出为低电平；当

{0,CL[7:0]}的值等于或大于{EPCnL,CCAPnL[7:0]}时，输出为高电平。当 CL[7:0]的值由 FF 变为 00 溢出时，{EPCnH,CCAPnH[7:0]}的内容重新装载到{EPCnL,CCAPnL[7:0]}中。这样就可实现无干扰地更新 PWM。

PCA 模块工作于 8 位 PWM 模式的结构图如下图所示：

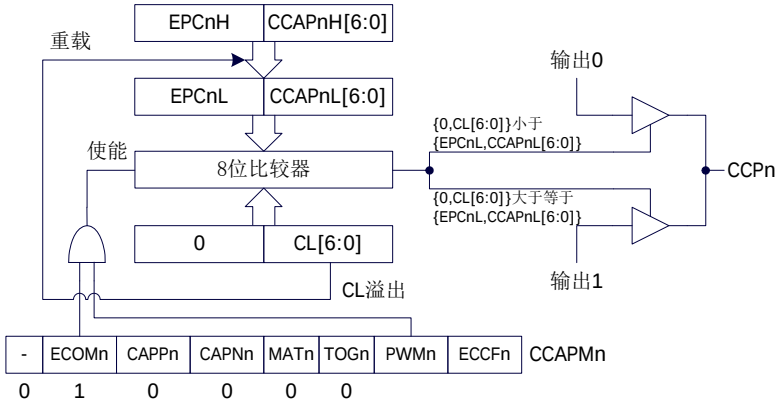


PCA模块8位PWM模式

17.2.4.2 7 位PWM模式

PCA_PWMn 寄存器中的 EBSn[1:0]设置为 01 时，PCA 模块 n 工作于 7 位 PWM 模式，此时将{0,CL[6:0]}与捕获寄存器{EPCnL,CCAPnL[6:0]}进行比较。当 PCA 模块工作于 7 位 PWM 模式时，由于所有模块共用一个 PCA 计数器，所有它们的输出频率相同。各个模块的输出占空比使用寄存器{EPCnL,CCAPnL[6:0]}进行设置。当{0,CL[6:0]}的值小于{EPCnL,CCAPnL[6:0]}时，输出为低电平；当{0,CL[6:0]}的值等于或大于{EPCnL,CCAPnL[6:0]}时，输出为高电平。当 CL[6:0]的值由 7F 变为 00 溢出时，{EPCnH,CCAPnH[6:0]}的内容重新装载到{EPCnL,CCAPnL[6:0]}中。这样就可实现无干扰地更新 PWM。

PCA 模块工作于 7 位 PWM 模式的结构图如下图所示：

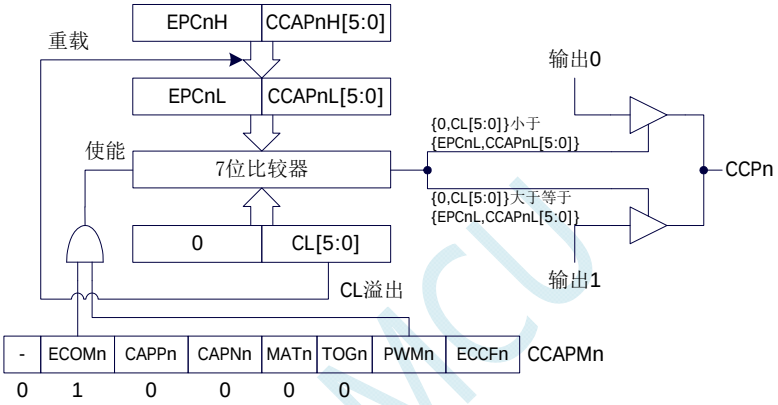


PCA模块7位PWM模式

17.2.4.3 6 位PWM模式

PCA_PWMn 寄存器中的 EBSn[1:0]设置为 10 时，PCA 模块 n 工作于 6 位 PWM 模式，此时将{0,CL[5:0]}与捕获寄存器{EPCnL,CCAPnL[5:0]}进行比较。当 PCA 模块工作于 6 位 PWM 模式时，由于所有模块共用一个 PCA 计数器，所有它们的输出频率相同。各个模块的输出占空比使用寄存器{EPCnL,CCAPnL[5:0]}进行设置。当{0,CL[5:0]}的值小于{EPCnL,CCAPnL[5:0]}时，输出为低电平；当{0,CL[5:0]}的值等于或大于{EPCnL,CCAPnL[5:0]}时，输出为高电平。当 CL[5:0]的值由 3F 变为 00 溢出时，{EPCnH,CCAPnH[5:0]}的内容重新装载到{EPCnL,CCAPnL[5:0]}中。这样就可实现无干扰地更新 PWM。

PCA 模块工作于 6 位 PWM 模式的结构图如下图所示：

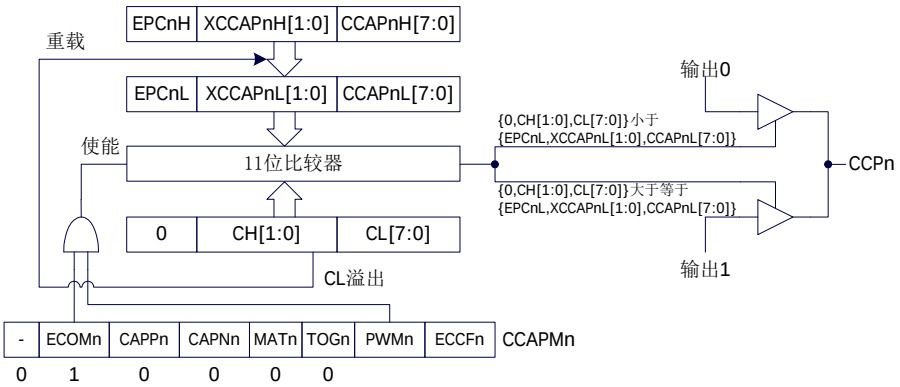


PCA模块6位PWM模式

17.2.4.4 10 位PWM模式

PCA_PWMn 寄存器中的 EBSn[1:0]设置为 11 时，PCA 模块 n 工作于 10 位 PWM 模式，此时将{CH[1:0],CL[7:0]}与捕获寄存器{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}进行比较。当 PCA 模块工作于 10 位 PWM 模式时，由于所有模块共用一个 PCA 计数器，所有它们的输出频率相同。各个模块的输出占空比使用寄存器{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}进行设置。当{CH[1:0],CL[7:0]}的值小于{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}时，输出为低电平；当{CH[1:0],CL[7:0]}的值等于或大于{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}时，输出为高电平。当{CH[1:0],CL[7:0]}的值由 3FF 变为 00 溢出时，{EPCnH,XCCAPnH[1:0],CCAPnH[7:0]}的内容重新装载到{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}中。这样就可实现无干扰地更新 PWM。

PCA 模块工作于 10 位 PWM 模式的结构图如下图所示：



PCA模块10位PWM模式

17.3 范例程序

17.3.1 PCA输出PWM（6/7/8/10 位）

汇编代码

```
;测试工作频率为11.0592MHz

CCON    DATA    0D8H
CF      BIT     CCON.7
CR      BIT     CCON.6
CCF2    BIT     CCON.2
CCF1    BIT     CCON.1
CCF0    BIT     CCON.0
CMOD    DATA    0D9H
CL      DATA    0E9H
CH      DATA    0F9H
CCAPM0  DATA    0DAH
CCAP0L  DATA    0EAH
CCAP0H  DATA    0FAH
PCA_PWM0 DATA   0F2H
CCAPM1  DATA    0DBH
CCAP1L  DATA    0EBH
CCAP1H  DATA    0FBH
PCA_PWM1 DATA   0F3H
CCAPM2  DATA    0DCH
CCAP2L  DATA    0ECH
CCAP2H  DATA    0FCH
PCA_PWM2 DATA   0F4H

P1M1    DATA    091H
P1M0    DATA    092H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     CCON,#00H
        MOV     CMOD,#08H       ;PCA 时钟为系统时钟
        MOV     CL,#00H
        MOV     CH,#0H
        MOV     CCAPM0,#42H     ;PCA 模块 0 为 PWM 工作模式
        MOV     PCA_PWM0,#80H   ;PCA 模块 0 输出 6 位 PWM
        MOV     CCAP0L,#20H     ;PWM 占空比为 50%[(40H-20H)/40H]
        MOV     CCAP0H,#20H
        MOV     CCAPM1,#42H     ;PCA 模块 1 为 PWM 工作模式
```

```
        MOV     PCA_PWM1,#40H       ;PCA 模块 1 输出 7 位 PWM
        MOV     CCAP1L,#20H         ;PWM 占空比为 75%[(80H-20H)/80H]
        MOV     CCAP1H,#20H
        MOV     CCAPM2,#42H         ;PCA 模块 2 为 PWM 工作模式
        MOV     PCA_PWM2,#00H       ;PCA 模块 2 输出 8 位 PWM
        MOV     CCAP2L,#20H         ;PWM 占空比为 87.5%[(100H-20H)/100H]
        MOV     CCAP2H,#20H
        SETB    CR                  ;启动 PCA 计时器

        JMP     $

        END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
```

```
    CMOD = 0x08;                        //PCA 时钟为系统时钟
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x42;                      //PCA 模块 0 为 PWM 工作模式
    PCA_PWM0 = 0x80;                    //PCA 模块 0 输出 6 位 PWM
    CCAP0L = 0x20;                      //PWM 占空比为 50%[(40H-20H)/40H]
    CCAP0H = 0x20;
    CCAPM1 = 0x42;                      //PCA 模块 1 为 PWM 工作模式
    PCA_PWM1 = 0x40;                    //PCA 模块 1 输出 7 位 PWM
    CCAP1L = 0x20;                      //PWM 占空比为 75%[(80H-20H)/80H]
    CCAP1H = 0x20;
    CCAPM2 = 0x42;                      //PCA 模块 2 为 PWM 工作模式
    PCA_PWM2 = 0x00;                    //PCA 模块 2 输出 8 位 PWM
    CCAP2L = 0x20;                      //PWM 占空比为 87.5%[(100H-20H)/100H]
    CCAP2H = 0x20;
    CR = 1;                             //启动 PCA 计时器

    while (1);
}
```

17.3.2 PCA捕获测量脉冲宽度

汇编代码

```
;测试工作频率为 11.0592MHz

CCON        DATA    0D8H
CF          BIT     CCON.7
CR          BIT     CCON.6
CCF2        BIT     CCON.2
CCF1        BIT     CCON.1
CCF0        BIT     CCON.0
CMOD        DATA    0D9H
CL          DATA    0E9H
CH          DATA    0F9H
CCAPM0      DATA    0DAH
CCAP0L      DATA    0EAH
CCAP0H      DATA    0FAH
PCA_PWM0    DATA    0F2H
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H

CNT         DATA    20H
COUNT0      DATA    21H             ;3 bytes
COUNT1      DATA    24H             ;3 bytes
LENGTH      DATA    27H             ;3 bytes, (COUNT1-COUNT0)

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH
```

```
        ORG     0000H
        LJMP    MAIN
        ORG     003BH
        LJMP    PCAISR

        ORG     0100H
PCAISR:
        PUSH    ACC
        PUSH    PSW
        JNB     CF,CHECKCCF0
        CLR     CF              ;清中断标志
        INC     CNT             ; PCA 计时溢出次数+1
CHECKCCF0:
        JNB     CCF0,ISREXIT
        CLR     CCF0
        MOV     COUNT0,COUNT1   ;备份上一次的捕获值
        MOV     COUNT0+1,COUNT1+1
        MOV     COUNT0+2,COUNT1+2
        MOV     COUNT1,CNT      ;保存本次的捕获值
        MOV     COUNT1+1,CCAP0H
        MOV     COUNT1+2,CCAP0L
        CLR     C               ;计算两次的捕获差值
        MOV     A,COUNT1+2
        SUBB    A,COUNT0+2
        MOV     LENGTH+2,A
        MOV     A,COUNT1+1
        SUBB    A,COUNT0+1
        MOV     LENGTH+1,A
        MOV     A,COUNT1
        SUBB    A,COUNT0
        MOV     LENGTH,A        ;LENGTH 保存的即为捕获的脉冲宽度
ISREXIT:
        POP     PSW
        POP     ACC
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        CLR     A
        MOV     CNT,A           ;用户变量初始化
        MOV     COUNT0,A
        MOV     COUNT0+1,A
        MOV     COUNT0+2,A
        MOV     COUNT1,A
        MOV     COUNT1+1,A
        MOV     COUNT1+2,A
        MOV     LENGTH,A
        MOV     LENGTH+1,A
        MOV     LENGTH+2,A

        MOV     CCON,#00H
```

```
        MOV     CMOD,#09H       ;PCA 时钟为系统时钟,使能 PCA 计时中断
        MOV     CL,#00H
        MOV     CH,#0H
        MOV     CCAPM0,#11H     ;PCA 模块 0 为 16 位捕获模式（下降沿捕获）
;       MOV     CCAPM0,#21H     ;PCA 模块 0 为 16 位捕获模式（上升沿捕获）
;       MOV     CCAPM0,#31H     ;PCA 模块 0 为 16 位捕获模式（边沿捕获）
        MOV     CCAP0L,#00H
        MOV     CCAP0H,#00H
        SETB    CR              ;启动 PCA 计时器
        SETB    EA

        JMP     $

        END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

unsigned char   cnt;            //存储 PCA 计时溢出次数
unsigned long   count0;         //记录上一次的捕获值
unsigned long   count1;         //记录本次的捕获值
unsigned long   length;         //存储信号的时间长度

void PCA_Isr() interrupt 7
{
    if (CF)
```

```
    {
        CF = 0;
        cnt++;                                  //PCA 计时溢出次数+1
    }
    if (CCF0)
    {
        CCF0 = 0;
        count0 = count1;                        //备份上一次的捕获值
        ((unsigned char *)&count1)[3] = CCAP0L;
        ((unsigned char *)&count1)[2] = CCAP0H;
        ((unsigned char *)&count1)[1] = cnt;
        ((unsigned char *)&count1)[0] = 0;
        length = count1 - count0;               //length 保存的即为捕获的脉冲宽度
    }
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    cnt = 0;                                    //用户变量初始化
    count0 = 0;
    count1 = 0;
    length = 0;
    CCON = 0x00;
    CMOD = 0x09;                                //PCA 时钟为系统时钟,使能 PCA 计时中断
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x11;                              //PCA 模块 0 为 16 位捕获模式（下降沿捕获）
    CCAPM0 = 0x21;                              //PCA 模块 0 为 16 位捕获模式（下降沿捕获）
    CCAPM0 = 0x31;                              //PCA 模块 0 为 16 位捕获模式（下降沿捕获）
    CCAP0L = 0x00;
    CCAP0H = 0x00;
    CR = 1;                                     //启动 PCA 计时器
    EA = 1;

    while (1);
}
```

17.3.3 PCA实现 16 位软件定时

汇编代码

```
;测试工作频率为 11.0592MHz

CCON    DATA    0D8H
CF      BIT     CCON.7
CR      BIT     CCON.6
CCF2    BIT     CCON.2
CCF1    BIT     CCON.1
CCF0    BIT     CCON.0
CMOD    DATA    0D9H
CL      DATA    0E9H
CH      DATA    0F9H
```

```
CCAPM0      DATA    0DAH
CCAP0L      DATA    0EAH
CCAP0H      DATA    0FAH
PCA_PWM0    DATA    0F2H
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H

T50HZ       EQU     2400H               ;11059200/12/2/50

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

            ORG     0100H
PCAISR:
            PUSH    ACC
            PUSH    PSW
            CLR     CCF0
            MOV     A,CCAP0L
            ADD     A,#LOW T50HZ
            MOV     CCAP0L,A
            MOV     A,CCAP0H
            ADDC    A,#HIGH T50HZ
            MOV     CCAP0H,A
            CPL     P1.0                ;测试端口,闪烁频率为50Hz
            POP     PSW
            POP     ACC
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     CCON,#00H
            MOV     CMOD,#00H           ;PCA 时钟为系统时钟/12
            MOV     CL,#00H
            MOV     CH,#0H
            MOV     CCAPM0,#49H         ;PCA 模块 0 为16 位定时器模式
            MOV     CCAP0L,#LOW T50HZ
            MOV     CCAP0H,#HIGH T50HZ
```

```
        SETB    CR                  ;启动 PCA 计时器
        SETB    EA

        JMP     $

        END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

#define     T50HZ           (11059200L / 12 / 2 / 50)

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

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        P10         =   P1^0;

unsigned int                value;

void PCA_Isr() interrupt 7
{
    CCF0 = 0;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T50HZ;

    P10 = !P10;                                 //测试端口
}
```

```
void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x00;                    //PCA 时钟为系统时钟/12
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x49;                  //PCA 模块 0 为 16 位定时器模式
    value = T50HZ;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T50HZ;
    CR = 1;                         //启动 PCA 计时器
    EA = 1;

    while (1);
}
```

17.3.4 PCA输出高速脉冲

汇编代码

```
;测试工作频率为 11.0592MHz

CCON        DATA    0D8H
CF          BIT     CCON.7
CR          BIT     CCON.6
CCF2        BIT     CCON.2
CCF1        BIT     CCON.1
CCF0        BIT     CCON.0
CMOD        DATA    0D9H
CL          DATA    0E9H
CH          DATA    0F9H
CCAPM0      DATA    0DAH
CCAP0L      DATA    0EAH
CCAP0H      DATA    0FAH
PCA_PWM0    DATA    0F2H
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H

T38K4HZ     EQU     90H             ;11059200/2/38400

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
```

```
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

            ORG     0100H
PCAISR:
            PUSH    ACC
            PUSH    PSW
            CLR     CCF0
            MOV     A,CCAP0L
            ADD     A,#LOW T38K4HZ
            MOV     CCAP0L,A
            MOV     A,CCAP0H
            ADDC    A,#HIGH T38K4HZ
            MOV     CCAP0H,A
            POP     PSW
            POP     ACC
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     CCON,#00H
            MOV     CMOD,#08H       ;PCA 时钟为系统时钟
            MOV     CL,#00H
            MOV     CH,#0H
            MOV     CCAPM0,#4DH     ;PCA 模块 0 为 16 位定时器模式并使能脉冲输出
            MOV     CCAP0L,#LOW T38K4HZ
            MOV     CCAP0H,#HIGH T38K4HZ
            SETB    CR              ;启动 PCA 计时器
            SETB    EA

            JMP     $

            END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

#define    T38K4HZ          (11059200L / 2 / 38400)

sfr        CCON        =    0xd8;
sbit       CF          =    CCON^7;
sbit       CR          =    CCON^6;
sbit       CCF2        =    CCON^2;
sbit       CCF1        =    CCON^1;
```

```
sbit    CCF0    =   CCON^0;
sfr     CMOD    =   0xd9;
sfr     CL      =   0xe9;
sfr     CH      =   0xf9;
sfr     CCAPM0  =   0xda;
sfr     CCAP0L  =   0xea;
sfr     CCAP0H  =   0xfa;
sfr     PCA_PWM0=   0xf2;
sfr     CCAPM1  =   0xdb;
sfr     CCAP1L  =   0xeb;
sfr     CCAP1H  =   0xfb;
sfr     PCA_PWM1=   0xf3;
sfr     CCAPM2  =   0xdc;
sfr     CCAP2L  =   0xec;
sfr     CCAP2H  =   0xfc;
sfr     PCA_PWM2=   0xf4;

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

unsigned int        value;

void PCA_Isr() interrupt 7
{
    CCF0 = 0;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T38K4HZ;
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x08;                    //PCA 时钟为系统时钟
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x4d;                  //PCA 模块 0 为 16 位定时器模式并使能脉冲输出
    value = T38K4HZ;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T38K4HZ;
    CR = 1;                         //启动 PCA 计时器
    EA = 1;

    while (1);
}
```

17.3.5 PCA扩展外部中断

汇编代码

```
;测试工作频率为11.0592MHz

CCON    DATA    0D8H
CF      BIT     CCON.7
CR      BIT     CCON.6
CCF2    BIT     CCON.2
CCF1    BIT     CCON.1
CCF0    BIT     CCON.0
CMOD    DATA    0D9H
CL      DATA    0E9H
CH      DATA    0F9H
CCAPM0  DATA    0DAH
CCAP0L  DATA    0EAH
CCAP0H  DATA    0FAH
PCA_PWM0 DATA   0F2H
CCAPM1  DATA    0DBH
CCAP1L  DATA    0EBH
CCAP1H  DATA    0FBH
PCA_PWM1 DATA   0F3H
CCAPM2  DATA    0DCH
CCAP2L  DATA    0ECH
CCAP2H  DATA    0FCH
PCA_PWM2 DATA   0F4H

P1M1    DATA    091H
P1M0    DATA    092H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
        ORG     003BH
        LJMP    PCAISR

        ORG     0100H
PCAISR:
        CLR     CCF0
        CPL     P1.0
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     CCON,#00H
        MOV     CMOD,#08H       ;PCA 时钟为系统时钟
        MOV     CL,#00H
        MOV     CH,#0H
```

```
        MOV     CCAPM0,#11H         ;扩展外部端口CCP0 为下降沿中断口
;       MOV     CCAPM0,#21H         ;扩展外部端口CCP0 为上升沿中断口
;       MOV     CCAPM0,#31H         ;扩展外部端口CCP0 为边沿中断口
        MOV     CCAP0L,#0
        MOV     CCAP0H,#0
        SETB    CR                  ;启动PCA 计时器
        SETB    EA

        JMP     $

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void PCA_Isr() interrupt 7
{
    CCF0 = 0;
    P10 = !P10;
}

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

```
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x08;                    //PCA 时钟为系统时钟
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x11;                  //扩展外部端口 CCP0 为下降沿中断口
//  CCAPM0 = 0x21;                  //扩展外部端口 CCP0 为上升沿中断口
//  CCAPM0 = 0x31;                  //扩展外部端口 CCP0 为边沿中断口
    CCAP0L = 0;
    CCAP0H = 0;
    CR = 1;                         //启动 PCA 计时器
    EA = 1;

    while (1);
}
```

18 同步串行外设接口SPI

STC8G 系列单片机内部集成了一种高速串行通信接口——SPI 接口。SPI 是一种全双工的高速同步通信总线。STC8G 系列集成的 SPI 接口提供了两种操作模式：主模式和从模式。

18.1 SPI相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
SPSTAT	SPI 状态寄存器	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
SPCTL	SPI 控制寄存器	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]		0000,0100
SPDAT	SPI 数据寄存器	CFH									0000,0000

SPI 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	CDH	SPIF	WCOL	-	-	-	-	-	-

SPIF：SPI 中断标志位。

当发送/接收完成 1 字节的数据后，硬件自动将此位置 1，并向 CPU 提出中断请求。当 SSIG 位被设置为 0 时，由于 SS 管脚电平的变化而使得设备的主/从模式发生改变时，此标志位也会被硬件自动置 1，以标志设备模式发生变化。

注意：此标志位必须用户通过软件方式向此位写 1 进行清零。

WCOL：SPI 写冲突标志位。

当 SPI 在进行数据传输的过程中写 SPDAT 寄存器时，硬件将此位置 1。

注意：此标志位必须用户通过软件方式向此位写 1 进行清零。

SPI 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPCTL	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]	

SSIG：SS 引脚功能控制位

0：SS 引脚确定器件是主机还是从机

1：忽略 SS 引脚功能，使用 MSTR 确定器件是主机还是从机

SPEN：SPI 使能控制位

0：关闭 SPI 功能

1：使能 SPI 功能

DORD：SPI 数据位发送/接收的顺序

0：先发送/接收数据的高位（MSB）

1：先发送/接收数据的低位（LSB）

MSTR：器件主/从模式选择位

设置主机模式：

若 SSIG＝0，则 SS 管脚必须为高电平且设置 MSTR 为 1

若 SSIG＝1，则只需要设置 MSTR 为 1（忽略 SS 管脚的电平）

设置从机模式：

若 SSIG＝0，则 SS 管脚必须为低电平（与 MSTR 位无关）

若 SSIG＝1，则只需要设置 MSTR 为 0（忽略 SS 管脚的电平）

CPOL：SPI 时钟极性控制

0：SCLK 空闲时为低电平，SCLK 的前时钟沿为上升沿，后时钟沿为下降沿

1：SCLK 空闲时为高电平，SCLK 的前时钟沿为下降沿，后时钟沿为上升沿

CPHA：SPI 时钟相位控制

0：数据 SS 管脚为低电平驱动第一位数据并在 SCLK 的后时钟沿改变数据，前时钟沿采样数据（必须 SSIG＝0）

1：数据在 SCLK 的前时钟沿驱动，后时钟沿采样

SPR[1:0]：SPI 时钟频率选择

SPR[1:0]	SCLK 频率
00	SYSclk/4
01	SYSclk/8
10	SYSclk/16
11	SYSclk/32

SPI 数据寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPDAT	CFH								

SPI 发送/接收数据缓冲器。

18.2 SPI通信方式

SPI 的通信方式通常有 3 种：单主单从（一个主机设备连接一个从机设备）、互为主从（两个设备连接，设备和互为主机和从机）、单主多从（一个主机设备连接多个从机设备）

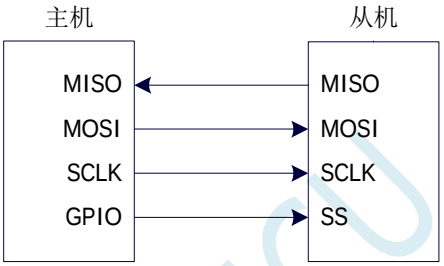
18.2.1 单主单从

两个设备相连，其中一个设备固定作为主机，另外一个固定作为从机。

主机设置：SSIG 设置为 1，MSTR 设置为 1，固定为主机模式。主机可以使用任意端口连接从机的 SS 管脚，拉低从机的 SS 脚即可使能从机

从机设置：SSIG 设置为 0，SS 管脚作为从机的片选信号。

单主单从连接配置图如下所示：



单主单从配置

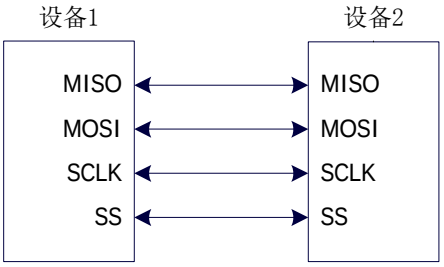
18.2.2 互为主从

两个设备相连，主机和从机不固定。

设置方法 1：两个设备初始化时都设置为 SSIG 设置为 0，MSTR 设置为 1，且将 SS 脚设置为双向口模式输出高电平。此时两个设备都是不忽略 SS 的主机模式。当其中一个设备需要启动传输时，可将自己的 SS 脚设置为输出模式并输出低电平，拉低对方的 SS 脚，这样另一个设备就被强行设置为从机模式了。

设置方法 2：两个设备初始化时都将自己设置成忽略 SS 的从机模式，即将 SSIG 设置为 1，MSTR 设置为 0。当其中一个设备需要启动传输时，先检测 SS 管脚的电平，如果时候高电平，就将自己设置成忽略 SS 的主模式，即可进行数据传输了。

互为主从连接配置图如下所示：



互为主从配置

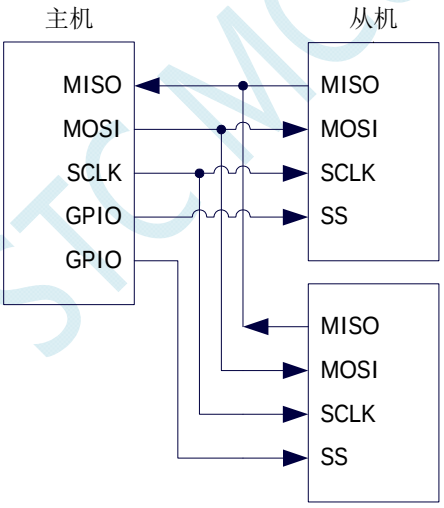
18.2.3 单主多从

多个设备相连，其中一个设备固定作为主机，其他设备固定作为从机。

主机设置：SSIG 设置为 1，MSTR 设置为 1，固定为主机模式。主机可以使用任意端口分别连接各个从机的 SS 管脚，拉低其中一个从机的 SS 脚即可使能相应的从机设备

从机设置：SSIG 设置为 0，SS 管脚作为从机的片选信号。

单主多从连接配置图如下所示：



单主多从配置

18.3 配置SPI

控制位			通信端口				说明
SPEN	SSIG	MSTR	SS	MISO	MOSI	SCLK	
0	x	x	x	输入	输入	输入	关闭SPI功能，SS/MOSI/MISO/SCLK均为普通I/O
1	0	0	0	输出	输入	输入	**从机模式**，且被选中
1	0	0	1	高阻	输入	输入	**从机模式**，但未被选中
1	0	1→0	0	输出	输入	输入	**从机模式**，不忽略 SS 且 MSTR 为 1 的主机模式，当 SS 管脚被拉低时，MSTR 将被硬件自动清零，工作模式将被被动设置为从机模式
1	0	1	1	输入	高阻	高阻	**主机模式**，空闲状态
					输出	输出	**主机模式**，激活状态
1	1	0	x	输出	输入	输入	**从机模式**
1	1	1	x	输入	输出	输出	**主机模式**

从机模式的注意事项：

当 CPHA＝0 时，SSIG 必须为 0（即不能忽略 SS 脚）。在每次串行字节开始还发送前 SS 脚必须拉低，并且在串行字节发送完后须重新设置为高电平。SS 管脚为低电平时不能对 SPDAT 寄存器执行写操作，否则将导致一个写冲突错误。CPHA=0 且 SSIG=1 时的操作未定义。

当 CPHA＝1 时，SSIG 可以置 1（即可以忽略脚）。如果 SSIG＝0，SS 脚可在连续传输之间保持低有效（即一直固定为低电平）。这种方式适用于固定单主单从的系统。

主机模式的注意事项：

在 SPI 中，传输总是由主机启动的。如果 SPI 使能（SPEN=1）并选择作为主机时，主机对 SPI 数据寄存器 SPDAT 的写操作将启动 SPI 时钟发生器和数据的传输。在数据写入 SPDAT 之后的半个到一个 SPI 位时间后，数据将出现在 MOSI 脚。写入主机 SPDAT 寄存器的数据从 MOSI 脚移出发送到从机的 MOSI 脚。同时从机 SPDAT 寄存器的数据从 MISO 脚移出发送到主机的 MISO 脚。

传输完一个字节后，SPI 时钟发生器停止，传输完成标志（SPIF）置位，如果 SPI 中断使能则会产生一个 SPI 中断。主机和从机 CPU 的两个移位寄存器可以看作是一个 16 位循环移位寄存器。当数据从主机移位传送到从机的同时，数据也以相反的方向移入。这意味着在一个移位周期中，主机和从机的数据相互交换。

通过 SS 改变模式

如果 SPEN=1，SSIG=0 且 MSTR=1，SPI 使能为主机模式，并将 SS 脚可配置为输入模式化或准双向口模式。这种情况下，另外一个主机可将该脚驱动为低电平，从而将该器件选择为 SPI 从机并向其发送数据。为了避免争夺总线，SPI 系统将该从机的 MSTR 清零，MOSI 和 SCLK 强制变为输入模式，而 MISO 则变为输出模式，同时 SPSTAT 的 SPIF 标志位置 1。

用户软件必须一直对 MSTR 位进行检测，如果该位被一个从机选择动作而被动清零，而用户想继续将 SPI 作为主机，则必须重新设置 MSTR 位，否则将一直处于从机模式。

写冲突

SPI 在发送时为单缓冲，在接收时为双缓冲。这样在前一次发送尚未完成之前，不能将新的数据写

入移位寄存器。当发送过程中对数据寄存器 SPDAT 进行写操作时，WCOL 位将被置 1 以指示发生数据写冲突错误。在这种情况下，当前发送的数据继续发送，而新写入的数据将丢失。

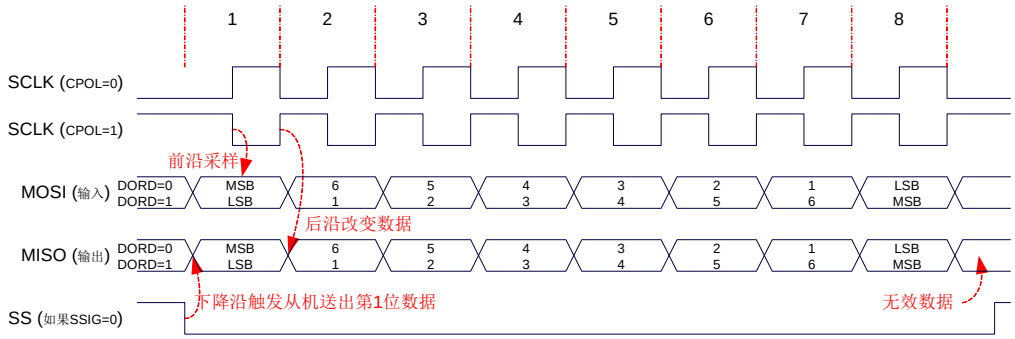
当对主机或从机进行写冲突检测时，主机发生写冲突的情况是很罕见的，因为主机拥有数据传输的完全控制权。但从机有可能发生写冲突，因为当主机启动传输时，从机无法进行控制。

接收数据时，接收到的数据传送到一个并行读数据缓冲区，这样将释放移位寄存器以进行下一个数据的接收。但必须在下个字符完全移入之前从数据寄存器中读出接收到的数据，否则，前一个接收数据将丢失。

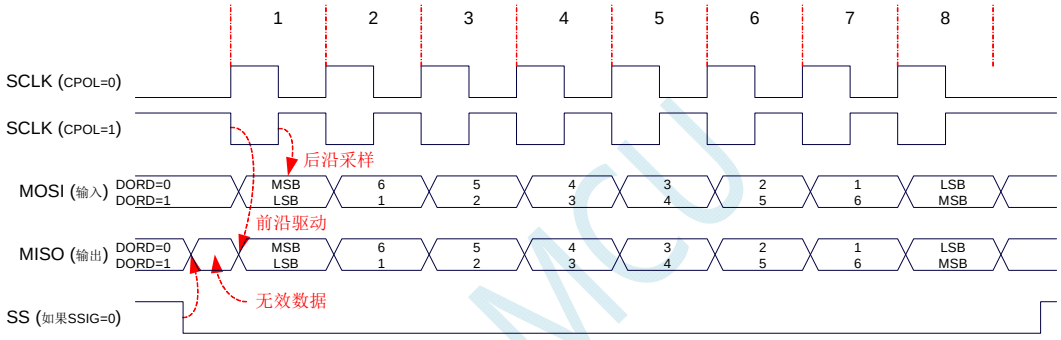
WCOL 可通过软件向其写入“1”清零。

18.4 数据模式

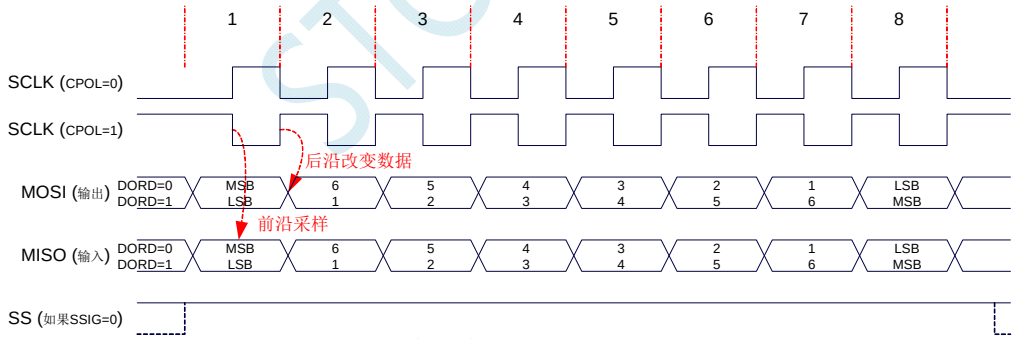
SPI 的时钟相位控制位 CPHA 可以让用户设定数据采样和改变时的时钟沿。时钟极性位 CPOL 可以让用户设定时钟极性。下面图例显示了不同时钟相位、极性设置下 SPI 通讯时序。



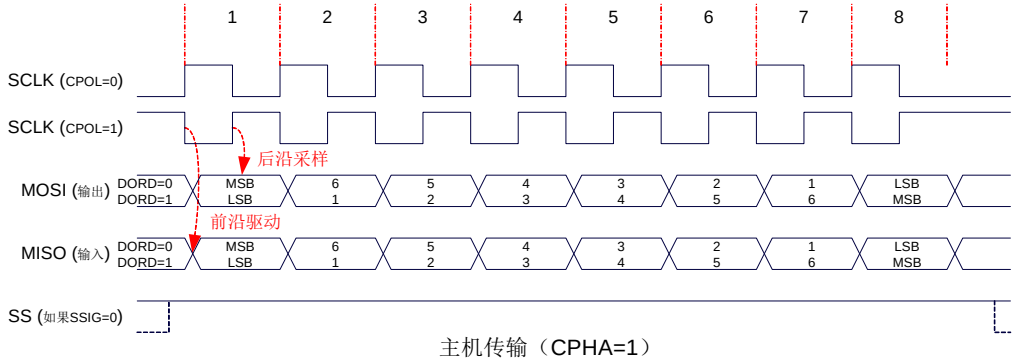
从机传输（CPHA=0）



从机传输（CPHA=1）



主机传输（CPHA=0）



主机传输（CPHA=1）

18.5 范例程序

18.5.1 SPI单主单从系统主机程序（中断方式）

汇编代码

```
;测试工作频率为11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

BUSY        BIT     20H.0
SS          BIT     P1.0
LED         BIT     P1.1

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            MOV     SPSTAT,#0C0H        ;清中断标志
            SETB    SS                  ;拉高从机的SS 管脚
            CLR     BUSY
            CPL     LED
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            SETB    LED
            SETB    SS
            CLR     BUSY

            MOV     SPCTL,#50H          ;使能SPI 主机模式
            MOV     SPSTAT,#0C0H        ;清中断标志
            MOV     IE2,#ESPI           ;使能SPI 中断
            SETB    EA

LOOP:
            JB      BUSY,$
            SETB    BUSY
```

```
        CLR     SS              ;拉低从机 SS 管脚
        MOV     SPDAT,#5AH      ;发送测试数据
        JMP     LOOP

        END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr     SPSTAT  =   0xcd;
sfr     SPCTL   =   0xce;
sfr     SPDAT   =   0xcf;
sfr     IE2     =   0xaf;
#define ESPI        0x02

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    SS      =   P1^0;
sbit    LED     =   P1^1;

bit     busy;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                  //清中断标志
    SS = 1;                         //拉高从机的 SS 管脚
    busy = 0;
    LED = !LED;                     //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    SS = 1;
    busy = 0;

    SPCTL = 0x50;                   //使能 SPI 主机模式
    SPSTAT = 0xc0;                  //清中断标志
    IE2 = ESPI;                     //使能 SPI 中断
    EA = 1;

    while (1)
    {
```

```
            while (busy);
            busy = 1;
            SS = 0;                                 //拉低从机 SS 管脚
            SPDAT = 0x5a;                           //发送测试数据
        }
}
```

18.5.2 SPI单主单从系统从机程序（中断方式）

汇编代码

```
;测试工作频率为11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

LED         BIT     P1.1

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            MOV     SPSTAT,#0C0H            ;清中断标志
            MOV     SPDAT,SPDAT             ;将接收到的数据回传给主机
            CPL     LED
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     SPCTL,#40H              ;使能 SPI 从机模式
            MOV     SPSTAT,#0C0H            ;清中断标志
            MOV     IE2,#ESPI               ;使能 SPI 中断
            SETB    EA

            JMP     $

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr         SPSTAT      =   0xcd;
sfr         SPCTL       =   0xce;
sfr         SPDAT       =   0xcf;
sfr         IE2         =   0xaf;
#define     ESPI            0x02

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        LED         =   P1^1;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                  //清中断标志
    SPDAT = SPDAT;                  //将接收到的数据回传给主机
    LED = !LED;                     //测试端口
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    SPCTL = 0x40;                   //使能 SPI 从机模式
    SPSTAT = 0xc0;                  //清中断标志
    IE2 = ESPI;                     //使能 SPI 中断
    EA = 1;

    while (1);
}
```

18.5.3 SPI单主单从系统主机程序（查询方式）

汇编代码

```
;测试工作频率为11.0592MHz

SPSTAT      DATA        0CDH
SPCTL       DATA        0CEH
SPDAT       DATA        0CFH
IE2         DATA        0AFH
ESPI        EQU         02H

SS          BIT         P1.0
```

```
LED         BIT         P1.1

P1M1        DATA        091H
P1M0        DATA        092H
P3M1        DATA        0B1H
P3M0        DATA        0B2H
P5M1        DATA        0C9H
P5M0        DATA        0CAH

            ORG         0000H
            LJMP        MAIN

            ORG         0100H
MAIN:
            MOV         SP, #5FH
            MOV         P1M0, #00H
            MOV         P1M1, #00H
            MOV         P3M0, #00H
            MOV         P3M1, #00H
            MOV         P5M0, #00H
            MOV         P5M1, #00H

            SETB        LED
            SETB        SS

            MOV         SPCTL,#50H          ;使能 SPI 主机模式
            MOV         SPSTAT,#0C0H        ;清中断标志

LOOP:
            CLR         SS                  ;拉低从机 SS 管脚
            MOV         SPDAT,#5AH          ;发送测试数据
            MOV         A,SPSTAT            ;查询完成标志
            JNB         ACC.7,$-2
            MOV         SPSTAT,#0C0H        ;清中断标志
            SETB        SS
            CPL         LED
            JMP         LOOP

            END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sfr     SPSTAT  =   0xcd;
sfr     SPCTL   =   0xce;
sfr     SPDAT   =   0xcf;
sfr     IE2     =   0xaf;
#define ESPI        0x02
```

```
sbit    SS      =   P1^0;
sbit    LED     =   P1^1;

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    SS = 1;

    SPCTL = 0x50;                   //使能 SPI 主机模式
    SPSTAT = 0xc0;                  //清中断标志

    while (1)
    {
        SS = 0;                     //拉低从机 SS 管脚
        SPDAT = 0x5a;               //发送测试数据
        while (!(SPSTAT & 0x80));   //查询完成标志
        SPSTAT = 0xc0;              //清中断标志
        SS = 1;                     //拉高从机的 SS 管脚
        LED = !LED;                 //测试端口
    }
}
```

18.5.4 SPI单主单从系统从机程序（查询方式）

汇编代码

```
;测试工作频率为 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

LED         BIT     P1.1

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

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

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

        MOV     SPCTL,#40H          ;使能 SPI 从机模式
        MOV     SPSTAT,#0C0H        ;清中断标志

LOOP:
        MOV     A,SPSTAT            ;查询完成标志
        JNB     ACC.7,$-2
        MOV     SPSTAT,#0C0H        ;清中断标志
        MOV     SPDAT,SPDAT         ;将接收到的数据回传给主机
        CPL     LED
        JMP     LOOP

        END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr     SPSTAT  =   0xcd;
sfr     SPCTL   =   0xce;
sfr     SPDAT   =   0xcf;
sfr     IE2     =   0xaf;
#define ESPI        0x02

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    LED     =   P1^1;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                  //清中断标志
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    SPCTL = 0x40;                   //使能 SPI 从机模式
    SPSTAT = 0xc0;                  //清中断标志

    while (1)
    {
```

```
            while (!(SPSTAT & 0x80));                   //查询完成标志
            SPSTAT = 0xc0;                              //清中断标志
            SPDAT = SPDAT;                              //将接收到的数据回传给主机
            LED = !LED;                                 //测试端口
        }
}
```

18.5.5 SPI互为主从系统程序（中断方式）

汇编代码

```
;测试工作频率为11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

SS          BIT     P1.0
LED         BIT     P1.1
KEY         BIT     P0.0

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            PUSH    ACC
            MOV     SPSTAT,#0C0H        ;清中断标志
            MOV     A,SPCTL
            JB      ACC.4,MASTER
SLAVE:
            MOV     SPDAT,SPDAT         ;将接收到的数据回传给主机
            JMP     ISREXIT
MASTER:
            SETB    SS                  ;拉高从机的SS 管脚
            MOV     SPCTL,#40H          ;重新设置为从机待机
ISREXIT:
            CPL     LED
            POP     ACC
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
```

```
                MOV         P5M1, #00H

                SETB        SS
                SETB        LED
                SETB        KEY

                MOV         SPCTL,#40H          ;使能 SPI 从机模式进行待机
                MOV         SPSTAT,#0C0H        ;清中断标志
                MOV         IE2,#ESPI           ;使能 SPI 中断
                SETB        EA

LOOP:
                JB          KEY,LOOP            ;等待按键触发
                MOV         SPCTL,#50H          ;使能 SPI 主机模式
                CLR         SS                  ;拉低从机 SS 管脚
                MOV         SPDAT,#5AH          ;发送测试数据
                JNB         KEY,$               ;等待按键释放
                JMP         LOOP

                END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr         SPSTAT      =   0xcd;
sfr         SPCTL       =   0xce;
sfr         SPDAT       =   0xcf;
sfr         IE2         =   0xaf;
#define     ESPI            0x02

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        SS          =   P1^0;
sbit        LED         =   P1^1;
sbit        KEY         =   P0^0;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                          //清中断标志
    if (SPCTL & 0x10)
    {                                       //主机模式
        SS = 1;                             //拉高从机的 SS 管脚
        SPCTL = 0x40;                       //重新设置为从机待机
    }
    else
    {                                       //从机模式
        SPDAT = SPDAT;                      //将接收到的数据回传给主机
    }
    LED = !LED;                             //测试端口
}
```

```
void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    KEY = 1;
    SS = 1;

    SPCTL = 0x40;                   //使能 SPI 从机模式进行待机
    SPSTAT = 0xc0;                  //清中断标志
    IE2 = ESPI;                     //使能 SPI 中断
    EA = 1;

    while (1)
    {
        if (!KEY)                   //等待按键触发
        {
            SPCTL = 0x50;           //使能 SPI 主机模式
            SS = 0;                 //拉低从机 SS 管脚
            SPDAT = 0x5a;           //发送测试数据
            while (!KEY);           //等待按键释放
        }
    }
}
```

18.5.6 SPI互为主从系统程序（查询方式）

汇编代码

```
;测试工作频率为11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

SS          BIT     P1.0
LED         BIT     P1.1
KEY         BIT     P0.0

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
```

```
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        SETB    SS
        SETB    LED
        SETB    KEY

        MOV     SPCTL,#40H          ;使能 SPI 从机模式进行待机
        MOV     SPSTAT,#0C0H        ;清中断标志

LOOP:
        JB      KEY,SKIP            ;等待按键触发
        MOV     SPCTL,#50H          ;使能 SPI 主机模式
        CLR     SS                  ;拉低从机 SS 管脚
        MOV     SPDAT,#5AH          ;发送测试数据
        JNB     KEY,$               ;等待按键释放
SKIP:
        MOV     A,SPSTAT
        JNB     ACC.7,LOOP
        MOV     SPSTAT,#0C0H        ;清中断标志
        MOV     A,SPCTL
        JB      ACC.4,MASTER
SLAVE:
        MOV     SPDAT,SPDAT         ;将接收到的数据回传给主机
        CPL     LED
        JMP     LOOP
MASTER:
        SETB    SS                  ;拉高从机的 SS 管脚
        MOV     SPCTL,#40H          ;重新设置为从机待机
        CPL     LED
        JMP     LOOP

        END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr     SPSTAT  =   0xcd;
sfr     SPCTL   =   0xce;
sfr     SPDAT   =   0xcf;
sfr     IE2     =   0xaf;
#define ESPI        0x02

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;
```

```
sbit    SS      =   P1^0;
sbit    LED     =   P1^1;
sbit    KEY     =   P0^0;

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    KEY = 1;
    SS = 1;

    SPCTL = 0x40;                       //使能 SPI 从机模式进行待机
    SPSTAT = 0xc0;                      //清中断标志

    while (1)
    {
        if (!KEY)                       //等待按键触发
        {
            SPCTL = 0x50;               //使能 SPI 主机模式
            SS = 0;                     //拉低从机 SS 管脚
            SPDAT = 0x5a;               //发送测试数据
            while (!KEY);               //等待按键释放
        }
        if (SPSTAT & 0x80)
        {
            SPSTAT = 0xc0;              //清中断标志
            if (SPCTL & 0x10)
            {                           //主机模式
                SS = 1;                 //拉高从机的 SS 管脚
                SPCTL = 0x40;           //重新设置为从机待机
            }
            else
            {                           //从机模式
                SPDAT = SPDAT;          //将接收到的数据回传给主机
            }
            LED = !LED;                 //测试端口
        }
    }
}
```

19 I²C总线

STC8G 系列的单片机内部集成了一个 I²C 串行总线控制器。I²C 是一种高速同步通讯总线，通讯使用 SCL（时钟线）和 SDA（数据线）两线进行同步通讯。对于 SCL 和 SDA 的端口分配，STC8G 系列的单片机提供了切换模式，可将 SCL 和 SDA 切换到不同的 I/O 口上，以方便用户将一组 I²C 总线当作多组进行分时复用。

与标准 I²C 协议相比较，忽略了如下两种机制：

- 发送起始信号（START）后不进行仲裁
- 时钟信号（SCL）停留在低电平时不进行超时检测

STC8G 系列的 I²C 总线提供了两种操作模式：主机模式（SCL 为输出口，发送同步时钟信号）和从机模式（SCL 为输入口，接收同步时钟信号）

19.1 I²C相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
I2CCFG	I²C 配置寄存器	FE80H	ENI2C	MSSL	MSSPEED[6:1]						0000,0000
I2CMSCR	I²C 主机控制寄存器	FE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I²C 主机状态寄存器	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I²C 从机控制寄存器	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I²C 从机状态寄存器	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I²C 从机地址寄存器	FE85H	SLADR[6:0]							MA	0000,0000
I2CTXD	I²C 数据发送寄存器	FE86H									0000,0000
I2CRXD	I²C 数据接收寄存器	FE87H									0000,0000
I2CMSAUX	I²C 主机辅助控制寄存器	FE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0

19.2 I²C主机模式

I²C 配置寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CCFG	FE80H	ENI2C	MSSL	MSSPEED[6:1]					

ENI2C：I²C 功能使能控制位

0：禁止 I²C 功能

1：允许 I²C 功能

MSSL：I²C 工作模式选择位

0：从机模式

1：主机模式

MSSPEED[6:1]：I²C 总线速度（等待时钟数）控制

MSSPEED[6:1]	对应的时钟数
0	1
1	3
2	5
…	…
x	2x+1
…	…
62	125
63	127

只有当 I²C 模块工作在主机模式时，MSSPEED 参数设置的等待参数才有效。此等待参数主要用于主机模式的以下几个信号：

T_{SSTA}：起始信号的建立时间（Setup Time of START）

T_{HSTA}：起始信号的保持时间（Hold Time of START）

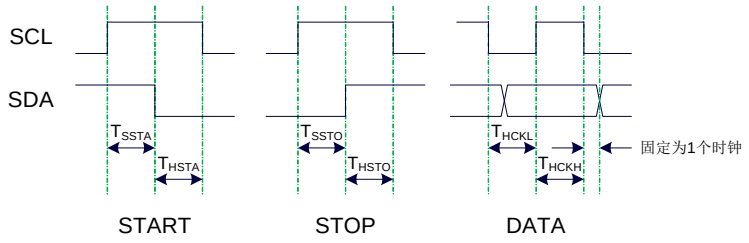
T_{SSTO}：停止信号的建立时间（Setup Time of STOP）

T_{HSTO}：停止信号的保持时间（Hold Time of STOP）

T_{HCKL}：时钟信号的低电平保持时间（Hold Time of SCL Low）

注意：

- 由于需要配合时钟同步机制，对于时钟信号的高电平保持时间（T_{HCKH}）至少为时钟信号的低电平保持时间（T_{HCKL}）的 1 倍长，而 T_{HCKH} 确切的长度取决于 SCL 端口的上拉速度。
- SDA 在 SCL 下降沿后的数据保持时间固定为 1 个时钟



I²C 主机控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	FE81H	EMSI	-	-	-	-	MSCMD[2:0]		

EMSI：主机模式中断使能控制位

0：关闭主机模式的中断

1：允许主机模式的中断

MSCMD[3:0]：主机命令

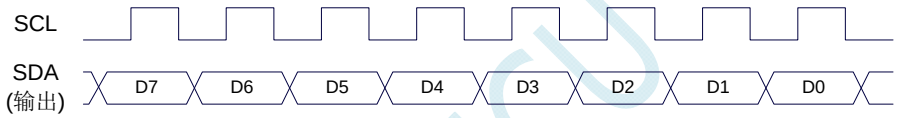
0000：待机，无动作。

0001：起始命令。

发送 START 信号。如果当前 I²C 控制器处于空闲状态，即 MSBUSY（I2CMSST.7）为 0 时，写此命令会使控制器进入忙状态，硬件自动将 MSBUSY 状态位置 1，并开始发送 START 信号；**若当前 I²C 控制器处于忙状态，写此命令可触发发送 START 信号**。发送 START 信号的波形如下图所示：

0010：发送数据命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 8 个时钟，并将 I2CTXD 寄存器里面数据按位送到 SDA 管脚上（先发送高位数据）。发送数据的波形如下图所示：

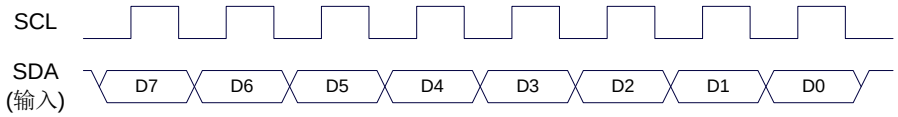


0011：接收 ACK 命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 1 个时钟，并将从 SDA 端口上读取的数据保存到 MSACKI（I2CMSST.1）。接收 ACK 的波形如下图所示：

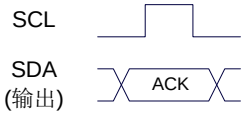
0100：接收数据命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 8 个时钟，并将从 SDA 端口上读取的数据依次左移到 I2CRXD 寄存器（先接收高位数据）。接收数据的波形如下图所示：



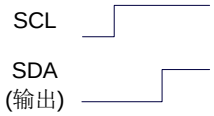
0101：发送 ACK 命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 1 个时钟，并将 MSACKO（I2CMSST.0）中的数据发送到 SDA 端口。发送 ACK 的波形如下图所示：



0110：停止命令。

发送 STOP 信号。写此命令后，I²C 总线控制器开始发送 STOP 信号。信号发送完成后，硬件自动将 MSBUSY 状态位清零。STOP 信号的波形如下图所示：



0111：保留。

1000：保留。

1001：起始命令+发送数据命令+接收 ACK 命令。

此命令为命令 0001、命令 0010、命令 0011 三个命令的组合，下此命令后控制器会依次执行这三个命令。

1010：发送数据命令+接收 ACK 命令。

此命令为命令 0010、命令 0011 两个命令的组合，下此命令后控制器会依次执行这两个命令。

1011：接收数据命令+发送 ACK(0)命令。

此命令为命令 0100、命令 0101 两个命令的组合，下此命令后控制器会依次执行这两个命令。

注意：此命令所返回的应答信号固定为 ACK（0），不受 MSACKO 位的影响。

1100：接收数据命令+发送 NAK(1)命令。

此命令为命令 0100、命令 0101 两个命令的组合，下此命令后控制器会依次执行这两个命令。

注意：此命令所返回的应答信号固定为 NAK（1），不受 MSACKO 位的影响。

I^2C 主机辅助控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSAUX	FE88H	-	-	-	-	-	-	-	WDTA

WDTA：主机模式时 I^2C 数据自动发送允许位

0：禁止自动发送

1：使能自动发送

若自动发送功能被使能，当 MCU 执行完成对 I2CTXD 数据寄存器的写操作后，I^2C 控制器会自动触发“1010”命令，即自动发送数据并接收 ACK 信号。

I^2C 主机状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSST	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO

MSBUSY：主机模式时 I^2C 控制器状态位（只读位）

0：控制器处于空闲状态

1：控制器处于忙碌状态

当 I^2C 控制器处于主机模式时，在空闲状态下，发送完成 START 信号后，控制器便进入到忙碌状态，忙碌状态会一直维持到成功发送完成 STOP 信号，之后状态会再次恢复到空闲状态。

MSIF：主机模式的中断请求位（中断标志位）。当处于主机模式的 I^2C 控制器执行完成寄存器 I2CMSCR 中 MSCMD 命令后产生中断信号，硬件自动将此位 1，向 CPU 发请求中断，响应中断后 MSIF 位必须用软件清零。

MSACKI：主机模式时，发送“0011”命令到 I2CMSCR 的 MSCMD 位后所接收到的 ACK 数据。

MSACKO：主机模式时，准备将要发送出去的 ACK 信号。当发送“0101”命令到 I2CMSCR 的 MSCMD 位后，控制器会自动读取此位的数据当作 ACK 发送到 SDA。

19.3 I²C从机模式

I²C 从机控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLCR	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST

ESTAI：从机模式时接收到 START 信号中断允许位

0：禁止从机模式时接收到 START 信号时发生中断

1：使能从机模式时接收到 START 信号时发生中断

ERXI：从机模式时接收到 1 字节数据后中断允许位

0：禁止从机模式时接收到数据后发生中断

1：使能从机模式时接收到 1 字节数据后发生中断

ETXI：从机模式时发送完成 1 字节数据后中断允许位

0：禁止从机模式时发送完成数据后发生中断

1：使能从机模式时发送完成 1 字节数据后发生中断

ESTOI：从机模式时接收到 STOP 信号中断允许位

0：禁止从机模式时接收到 STOP 信号时发生中断

1：使能从机模式时接收到 STOP 信号时发生中断

SLRST：复位从机模式

I²C 从机状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLST	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	-	SLACKI	SLACKO

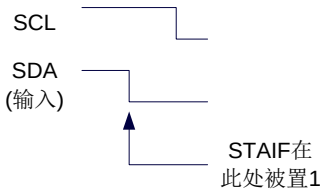
SLBUSY：从机模式时 I²C 控制器状态位（只读位）

0：控制器处于空闲状态

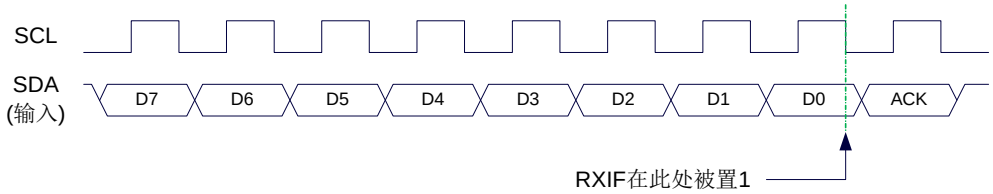
1：控制器处于忙碌状态

当 I²C 控制器处于从机模式时，在空闲状态下，接收到主机发送 START 信号后，控制器会继续检测之后的设备地址数据，若设备地址与当前 I2CSLADR 寄存器中所设置的从机地址像匹配时，控制器便进入到忙碌状态，忙碌状态会一直维持到成功接收到主机发送 STOP 信号，之后状态会再次恢复到空闲状态。

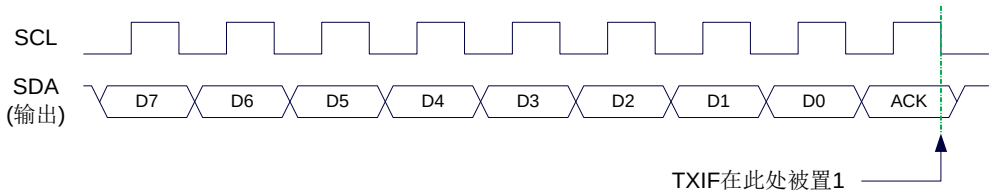
STAIF：从机模式时接收到 START 信号后的中断请求位。从机模式的 I²C 控制器接收到 START 信号后，硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 STAIF 位必须用软件清零。STAIF 被置 1 的时间点如下图所示：



RXIF：从机模式时接收到 1 字节的数据后的中断请求位。从机模式的 I²C 控制器接收到 1 字节的数据后，在第 8 个时钟的下降沿时硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 RXIF 位必须用软件清零。RXIF 被置 1 的时间点如下图所示：



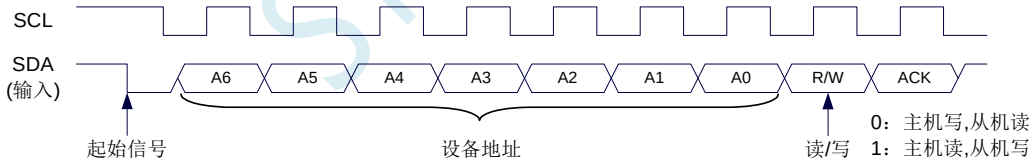
TXIF：从机模式时发送完成 1 字节的数据后的中断请求位。从机模式的 I²C 控制器发送完成 1 字节的数据并成功接收到 1 位 ACK 信号后，在第 9 个时钟的下降沿时硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 TXIF 位必须用软件清零。TXIF 被置 1 的时间点如下图所示：



STOIF：从机模式时接收到 STOP 信号后的中断请求位。从机模式的 I²C 控制器接收到 STOP 信号后，硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 STOIF 位必须用软件清零。STOIF 被置 1 的时间点如下图所示：

SLACKI：从机模式时，接收到的 ACK 数据。

SLACKO：从机模式时，准备将要发送出去的 ACK 信号。



I²C 从机地址寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLADR	FE85H	SLADR[6:0]							MA

SLADR[6:0]：从机设备地址

当 I²C 控制器处于从机模式时，控制器在接收到 START 信号后，会继续检测接下来主机发送出的设备地址数据以及读/写信号。当主机发送出的设备地址与 SLADR[6:0]中所设置的从机设备地址相匹配时，控制器才会向 CPU 发出中断求，请求 CPU 处理 I²C 事件；否则若设备地址不匹配，I²C 控制器继续继续监控，等待下一个起始信号，对下一个设备地址继续匹配。

MA：从机设备地址匹配控制

0：设备地址必须与 SLADR[6:0]继续匹配

1：忽略 SLADR 中的设置，匹配所有的设备地址

I²C 数据寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CTXD	FE86H								
I2CRXD	FE87H								

I2CTXD 是 I²C 发送数据寄存器，存放将要发送的 I²C 数据

I2CRXD 是 I²C 接收数据寄存器，存放接收完成的 I²C 数据

19.4 范例程序

19.4.1 I²C主机模式访问AT24C256（中断方式）

汇编代码

```
;测试工作频率为11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5

BUSY        BIT     20H.0

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00C3H
            LJMP    I2CISR

            ORG     0100H
I2CISR:
            PUSH    ACC
            PUSH    DPL
            PUSH    DPH

            MOV     DPTR,#I2CMSST       ;清中断标志
            MOVX    A,@DPTR
            ANL     A,#NOT 40H
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            CLR     BUSY                ;复位忙标志

            POP     DPH
            POP     DPL
            POP     ACC
            RETI

START:
            SETB    BUSY
            MOV     A,#10000001B        ;发送 START 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
```

```
            JMP     WAIT
SENDDATA:
            MOV     DPTR,#I2CTXD        ;写数据到数据缓冲区
            MOVX    @DPTR,A
            SETB    BUSY
            MOV     A,#10000010B        ;发送 SEND 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVACK:
            SETB    BUSY
            MOV     A,#10000011B        ;发送读 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVDATA:
            SETB    BUSY
            MOV     A,#10000100B        ;发送 RECV 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            CALL    WAIT
            MOV     DPTR,#I2CRXD        ;从数据缓冲区读取数据
            MOVX    A,@DPTR
            RET
SENDACK:
            MOV     A,#00000000B        ;设置 ACK 信号
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            SETB    BUSY
            MOV     A,#10000101B        ;发送 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDNAK:
            MOV     A,#00000001B        ;设置 NAK 信号
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            SETB    BUSY
            MOV     A,#10000101B        ;发送 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
STOP:
            SETB    BUSY
            MOV     A,#10000110B        ;发送 STOP 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
WAIT:
            JB      BUSY,$              ;等待命令发送完成
            RET

DELAY:
            MOV     R0,#0
            MOV     R1,#0
DELAY1:
            NOP
            NOP
            NOP
```

```
        NOP
        DJNZ    R1,DELAY1
        DJNZ    R0,DELAY1
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H
        MOV     P_SW2,#80H

        MOV     A,#11100000B        ;设置 I2C 模块为主机模式
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        SETB    EA

        CALL    START               ;发送起始命令
        MOV     A,#0A0H
        CALL    SENDDATA            ;发送设备地址+写命令
        CALL    RECVACK
        MOV     A,#000H             ;发送存储地址高字节
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H             ;发送存储地址低字节
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#12H              ;写测试数据 1
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#78H              ;写测试数据 2
        CALL    SENDDATA
        CALL    RECVACK
        CALL    STOP                ;发送停止命令

        CALL    DELAY               ;等待设备写数据

        CALL    START               ;发送起始命令
        MOV     A,#0A0H             ;发送设备地址+写命令
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H             ;发送存储地址高字节
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H             ;发送存储地址低字节
        CALL    SENDDATA
        CALL    RECVACK
        CALL    START               ;发送起始命令
        MOV     A,#0A1H             ;发送设备地址+读命令
        CALL    SENDDATA
        CALL    RECVACK
        CALL    RECVDATA            ;读取数据 1
        MOV     P0,A
```

```
        CALL    SENDACK
        CALL    RECVDATA            ;读取数据 2
        MOV     P2,A
        CALL    SENDNAK
        CALL    STOP                ;发送停止命令

        JMP     $

        END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD      (*(unsigned char volatile xdata *)0xfe87)

sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

sbit    SDA         =   P1^4;
sbit    SCL         =   P1^5;

bit     busy;

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;
    if (I2CMSST & 0x40)
    {
        I2CMSST &= ~0x40;                   //清中断标志
        busy = 0;
    }
    _pop_(P_SW2);
}

void Start()
{
    busy = 1;
    I2CMSCR = 0x81;                         //发送 START 命令
    while (busy);
}
```

```
void SendData(char dat)
{
    I2CTXD = dat;                       //写数据到数据缓冲区
    busy = 1;
    I2CMSCR = 0x82;                     //发送 SEND 命令
    while (busy);
}

void RecvACK()
{
    busy = 1;
    I2CMSCR = 0x83;                     //发送读 ACK 命令
    while (busy);
}

char RecvData()
{
    busy = 1;
    I2CMSCR = 0x84;                     //发送 RECV 命令
    while (busy);
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                     //设置 ACK 信号
    busy = 1;
    I2CMSCR = 0x85;                     //发送 ACK 命令
    while (busy);
}

void SendNAK()
{
    I2CMSST = 0x01;                     //设置 NAK 信号
    busy = 1;
    I2CMSCR = 0x85;                     //发送 ACK 命令
    while (busy);
}

void Stop()
{
    busy = 1;
    I2CMSCR = 0x86;                     //发送 STOP 命令
    while (busy);
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}
```

```
void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0xe0;                          //使能 I2C 主机模式
    I2CMSST = 0x00;
    EA = 1;

    Start();    //发送起始命令
    SendData(0xa0);                         //发送设备地址+写命令
    RecvACK();
    SendData(0x00);                         //发送存储地址高字节
    RecvACK();
    SendData(0x00);                         //发送存储地址低字节
    RecvACK();
    SendData(0x12);                         //写测试数据 1
    RecvACK();
    SendData(0x78);                         //写测试数据 2
    RecvACK();
    Stop();                                 //发送停止命令

    Delay();                                //等待设备写数据

    Start();                                //发送起始命令
    SendData(0xa0);                         //发送设备地址+写命令
    RecvACK();
    SendData(0x00);                         //发送存储地址高字节
    RecvACK();
    SendData(0x00);                         //发送存储地址低字节
    RecvACK();
    Start();                                //发送起始命令
    SendData(0xa1);                         //发送设备地址+读命令
    RecvACK();
    P0 = RecvData();                        //读取数据 1
    SendACK();
    P2 = RecvData();                        //读取数据 2
    SendNAK();
    Stop();                                 //发送停止命令

    P_SW2 = 0x00;

    while (1);
}
```

19.4.2 I²C主机模式访问AT24C256（查询方式）

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2           DATA            0BAH
```

```
I2CCFG      XDATA     0FE80H
I2CMSCR     XDATA     0FE81H
I2CMSST     XDATA     0FE82H
I2CSLCR     XDATA     0FE83H
I2CSLST     XDATA     0FE84H
I2CSLADR    XDATA     0FE85H
I2CTXD      XDATA     0FE86H
I2CRXD      XDATA     0FE87H

SDA         BIT       P1.4
SCL         BIT       P1.5

P1M1        DATA      091H
P1M0        DATA      092H
P3M1        DATA      0B1H
P3M0        DATA      0B2H
P5M1        DATA      0C9H
P5M0        DATA      0CAH

            ORG       0000H
            LJMP      MAIN

            ORG       0100H
START:
            MOV       A,#00000001B          ;发送 START 命令
            MOV       DPTR,#I2CMSCR
            MOVX      @DPTR,A
            JMP       WAIT
SENDDATA:
            MOV       DPTR,#I2CTXD          ;写数据到数据缓冲区
            MOVX      @DPTR,A
            MOV       A,#00000010B          ;发送 SEND 命令
            MOV       DPTR,#I2CMSCR
            MOVX      @DPTR,A
            JMP       WAIT
RECVACK:
            MOV       A,#00000011B          ;发送读 ACK 命令
            MOV       DPTR,#I2CMSCR
            MOVX      @DPTR,A
            JMP       WAIT
RECVDATA:
            MOV       A,#00000100B          ;发送 RECV 命令
            MOV       DPTR,#I2CMSCR
            MOVX      @DPTR,A
            CALL      WAIT
            MOV       DPTR,#I2CRXD          ;从数据缓冲区读取数据
            MOVX      A,@DPTR
            RET
SENDACK:
            MOV       A,#00000000B          ;设置 ACK 信号
            MOV       DPTR,#I2CMSST
            MOVX      @DPTR,A
            MOV       A,#00000101B          ;发送 ACK 命令
            MOV       DPTR,#I2CMSCR
            MOVX      @DPTR,A
            JMP       WAIT
SENDNAK:
            MOV       A,#00000001B          ;设置 NAK 信号
            MOV       DPTR,#I2CMSST
```

```
                MOVX    @DPTR,A
                MOV     A,#00000101B        ;发送 ACK 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
STOP:
                MOV     A,#00000110B        ;发送 STOP 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
WAIT:
                MOV     DPTR,#I2CMSST       ;清中断标志
                MOVX    A,@DPTR
                JNB     ACC.6,WAIT
                ANL     A,#NOT 40H
                MOVX    @DPTR,A
                RET

DELAY:
                MOV     R0,#0
                MOV     R1,#0
DELAY1:
                NOP
                NOP
                NOP
                NOP
                DJNZ    R1,DELAY1
                DJNZ    R0,DELAY1
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H
                MOV     P_SW2,#80H

                MOV     A,#11100000B        ;设置 I2C 模块为主机模式
                MOV     DPTR,#I2CCFG
                MOVX    @DPTR,A
                MOV     A,#00000000B
                MOV     DPTR,#I2CMSST
                MOVX    @DPTR,A

                CALL    START               ;发送起始命令
                MOV     A,#0A0H
                CALL    SENDDATA            ;发送设备地址+写命令
                CALL    RECVACK
                MOV     A,#000H             ;发送存储地址高字节
                CALL    SENDDATA
                CALL    RECVACK
                MOV     A,#000H             ;发送存储地址低字节
                CALL    SENDDATA
                CALL    RECVACK
                MOV     A,#12H              ;写测试数据 1
                CALL    SENDDATA
```

```
        CALL    RECVACK
        MOV     A,#78H          ;写测试数据 2
        CALL    SENDDATA
        CALL    RECVACK
        CALL    STOP            ;发送停止命令

        CALL    DELAY           ;等待设备写数据

        CALL    START           ;发送起始命令
        MOV     A,#0A0H         ;发送设备地址+写命令
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H         ;发送存储地址高字节
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H         ;发送存储地址低字节
        CALL    SENDDATA
        CALL    RECVACK
        CALL    START           ;发送起始命令
        MOV     A,#0A1H         ;发送设备地址+读命令
        CALL    SENDDATA
        CALL    RECVACK
        CALL    RECVDATA        ;读取数据 1
        MOV     P0,A
        CALL    SENDACK
        CALL    RECVDATA        ;读取数据 2
        MOV     P2,A
        CALL    SENDNAK
        CALL    STOP            ;发送停止命令

        JMP     $

        END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD      (*(unsigned char volatile xdata *)0xfe87)

sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;
```

```
sbit        SDA             =     P1^4;
sbit        SCL             =     P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                             //发送 START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                               //写数据到数据缓冲区
    I2CMSCR = 0x02;                             //发送 SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                             //发送读 ACK 命令
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                             //发送 RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                             //设置 ACK 信号
    I2CMSCR = 0x05;                             //发送 ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                             //设置 NAK 信号
    I2CMSCR = 0x05;                             //发送 ACK 命令
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                             //发送 STOP 命令
    Wait();
}

void Delay()
{
    int i;
```

```
    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0xe0;                          //使能 I2C 主机模式
    I2CMSST = 0x00;

    Start();                                //发送起始命令
    SendData(0xa0);                         //发送设备地址+写命令
    RecvACK();
    SendData(0x00);                         //发送存储地址高字节
    RecvACK();
    SendData(0x00);                         //发送存储地址低字节
    RecvACK();
    SendData(0x12);                         //写测试数据 1
    RecvACK();
    SendData(0x78);                         //写测试数据 2
    RecvACK();
    Stop();                                 //发送停止命令

    Delay();                                //等待设备写数据

    Start();                                //发送起始命令
    SendData(0xa0);                         //发送设备地址+写命令
    RecvACK();
    SendData(0x00);                         //发送存储地址高字节
    RecvACK();
    SendData(0x00);                         //发送存储地址低字节
    RecvACK();
    Start();                                //发送起始命令
    SendData(0xa1);                         //发送设备地址+读命令
    RecvACK();
    P0 = RecvData();                        //读取数据 1
    SendACK();
    P2 = RecvData();                        //读取数据 2
    SendNAK();
    Stop();                                 //发送停止命令

    P_SW2 = 0x00;

    while (1);
}
```

19.4.3 I²C主机模式访问PCF8563

汇编代码

```
;测试工作频率为11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
START:
            MOV     A,#00000001B        ;发送 START 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDDATA:
            MOV     DPTR,#I2CTXD        ;写数据到数据缓冲区
            MOVX    @DPTR,A
            MOV     A,#00000010B        ;发送 SEND 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVACK:
            MOV     A,#00000011B        ;发送读 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVDATA:
            MOV     A,#00000100B        ;发送 RECV 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            CALL    WAIT
            MOV     DPTR,#I2CRXD        ;从数据缓冲区读取数据
            MOVX    A,@DPTR
            RET
SENDACK:
            MOV     A,#00000000B        ;设置 ACK 信号
            MOV     DPTR,#I2CMSST
```

```
            MOVX    @DPTR,A
            MOV     A,#00000101B        ;发送 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDNAK:
            MOV     A,#00000001B        ;设置 NAK 信号
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            MOV     A,#00000101B        ;发送 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
STOP:
            MOV     A,#00000110B        ;发送 STOP 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
WAIT:
            MOV     DPTR,#I2CMSST       ;清中断标志
            MOVX    A,@DPTR
            JNB     ACC.6,WAIT
            ANL     A,#NOT 40H
            MOVX    @DPTR,A
            RET

DELAY:
            MOV     R0,#0
            MOV     R1,#0
DELAY1:
            NOP
            NOP
            NOP
            NOP
            DJNZ    R1,DELAY1
            DJNZ    R0,DELAY1
            RET

MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H
            MOV     P_SW2,#80H

            MOV     A,#11100000B        ;设置 I2C 模块为主机模式
            MOV     DPTR,#I2CCFG
            MOVX    @DPTR,A
            MOV     A,#00000000B
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A

            CALL    START               ;发送起始命令
            MOV     A,#0A2H
            CALL    SENDDATA            ;发送设备地址+写命令
            CALL    RECVACK
```

```
        MOV     A,#002H         ;发送存储地址
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#00H          ;设置秒值
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#00H          ;设置分钟值
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#12H          ;设置小时值
        CALL    SENDDATA
        CALL    RECVACK
        CALL    STOP            ;发送停止命令
LOOP:
        CALL    START           ;发送起始命令
        MOV     A,#0A2H         ;发送设备地址+写命令
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#002H         ;发送存储地址
        CALL    SENDDATA
        CALL    RECVACK
        CALL    START           ;发送起始命令
        MOV     A,#0A3H         ;发送设备地址+读命令
        CALL    SENDDATA
        CALL    RECVACK
        CALL    RECVDATA        ;读取秒值
        MOV     P0,A
        CALL    SENDACK
        CALL    RECVDATA        ;读取分钟值
        MOV     P2,A
        CALL    SENDACK
        CALL    RECVDATA        ;读取小时值
        MOV     P3,A
        CALL    SENDNAK
        CALL    STOP            ;发送停止命令

        CALL    DELAY

        JMP     LOOP

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD      (*(unsigned char volatile xdata *)0xfe87)
```

```
sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        SDA             =   P1^4;
sbit        SCL             =   P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                             //发送 START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                               //写数据到数据缓冲区
    I2CMSCR = 0x02;                             //发送 SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                             //发送读 ACK 命令
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                             //发送 RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                             //设置 ACK 信号
    I2CMSCR = 0x05;                             //发送 ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                             //设置 NAK 信号
    I2CMSCR = 0x05;                             //发送 ACK 命令
    Wait();
}

void Stop()
{
```

```
    I2CMSCR = 0x06;                         //发送 STOP 命令
    Wait();
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0xe0;                          //使能 I2C 主机模式
    I2CMSST = 0x00;

    Start();                                //发送起始命令
    SendData(0xa2);                         //发送设备地址+写命令
    RecvACK();
    SendData(0x02);                         //发送存储地址
    RecvACK();
    SendData(0x00);                         //设置秒值
    RecvACK();
    SendData(0x00);                         //设置分钟值
    RecvACK();
    SendData(0x12);                         //设置小时值
    RecvACK();
    Stop();                                 //发送停止命令

    while (1)
    {
        Start();                            //发送起始命令
        SendData(0xa2);                     //发送设备地址+写命令
        RecvACK();
        SendData(0x02);                     //发送存储地址
        RecvACK();
        Start();                            //发送起始命令
        SendData(0xa3);                     //发送设备地址+读命令
        RecvACK();
        P0 = RecvData();                    //读取秒值
        SendACK();
        P2 = RecvData();                    //读取分钟值
        SendACK();
        P3 = RecvData();                    //读取小时值
```

```
            SendNAK();
            Stop();                                 //发送停止命令

            Delay();
        }
}
```

19.4.4 I²C从机模式（中断方式）

汇编代码

```
;测试工作频率为11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5
ISDA        BIT     20H.0               ;设备地址标志
ISMA        BIT     20H.1               ;存储地址标志

ADDR        DATA    21H

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00C3H
            LJMP    I2CISR

            ORG     0100H
I2CISR:
            PUSH    ACC
            PUSH    PSW
            PUSH    DPL
            PUSH    DPH
            MOV     DPTR,#I2CSLST           ;检测从机状态
            MOVX    A,@DPTR
            JB      ACC.6,STARTIF
            JB      ACC.5,RXIF
            JB      ACC.4,TXIF
            JB      ACC.3,STOPIF
ISREXIT:
            POP     DPH
            POP     DPL
            POP     PSW
```

```
        POP     ACC
        RETI
STARTIF:
        ANL     A,#NOT 40H          ;处理 START 事件
        MOVX    @DPTR,A
        JMP     ISREXIT
RXIF:
        ANL     A,#NOT 20H          ;处理 RECV 事件
        MOVX    @DPTR,A
        MOV     DPTR,#I2CRXD
        MOVX    A,@DPTR
        JBC     ISDA,RXDA
        JBC     ISMA,RXMA
        MOV     R0,ADDR             ;处理 RECV 事件（RECV DATA）
        MOVX    @R0,A
        INC     ADDR
        JMP     ISREXIT
RXDA:
        JMP     ISREXIT             ;处理 RECV 事件（RECV DEVICE ADDR）
RXMA:
        MOV     ADDR,A              ;处理 RECV 事件（RECV MEMORY ADDR）
        MOV     R0,A
        MOVX    A,@R0
        MOV     DPTR,#I2CTXD
        MOVX    @DPTR,A
        JMP     ISREXIT
TXIF:
        ANL     A,#NOT 10H          ;处理 SEND 事件
        MOVX    @DPTR,A
        JB      ACC.1,RXNAK
        INC     ADDR
        MOV     R0,ADDR
        MOVX    A,@R0
        MOV     DPTR,#I2CTXD
        MOVX    @DPTR,A
        JMP     ISREXIT
RXNAK:
        MOVX    A,#0FFH
        MOV     DPTR,#I2CTXD
        MOVX    @DPTR,A
        JMP     ISREXIT
STOPIF:
        ANL     A,#NOT 08H          ;处理 STOP 事件
        MOVX    @DPTR,A
        SETB    ISDA
        SETB    ISMA
        JMP     ISREXIT

MAIN:
        MOV     SP, #5FH
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H
        MOV     P_SW2,#80H

        MOV     P_SW2,#80H
```

```
        MOV     A,#10000001B            ;使能 I2C 从机模式
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#01011010B            ;设置从机设备地址为 5A
        MOV     DPTR,#I2CSLADR
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CSLST
        MOVX    @DPTR,A
        MOV     A,#01111000B            ;使能从机模式中断
        MOV     DPTR,#I2CSLCR
        MOVX    @DPTR,A

        SETB    ISDA                    ;用户变量初始化
        SETB    ISMA
        CLR     A
        MOV     ADDR,A
        MOV     R0,A
        MOVX    A,@R0
        MOV     DPTR,#I2CTXD
        MOVX    @DPTR,A

        SETB    EA

        SJMP    $

        END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD      (*(unsigned char volatile xdata *)0xfe87)

sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

sbit    SDA         =   P1^4;
sbit    SCL         =   P1^5;

bit     isda;                           //设备地址标志
bit     isma;                           //存储地址标志
```

```
unsigned char           addr;
unsigned char pdata     buffer[256];

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;

    if (I2CSLST & 0x40)
    {
        I2CSLST &= ~0x40;                   //处理 START 事件
    }
    else if (I2CSLST & 0x20)
    {
        I2CSLST &= ~0x20;                   //处理 RECV 事件
        if (isda)
        {
            isda = 0;                       //处理 RECV 事件（RECV DEVICE ADDR）
        }
        else if (isma)
        {
            isma = 0;                       //处理 RECV 事件（RECV MEMORY ADDR）
            addr = I2CRXD;
            I2CTXD = buffer[addr];
        }
        else
        {
            buffer[addr++] = I2CRXD;        //处理 RECV 事件（RECV DATA）
        }
    }
    else if (I2CSLST & 0x10)
    {
        I2CSLST &= ~0x10;                   //处理 SEND 事件
        if (I2CSLST & 0x02)
        {
            I2CTXD = 0xff;                  //接收到 NAK 则停止读取数据
        }
        else
        {
            I2CTXD = buffer[++addr];        //接收到 ACK 则继续读取数据
        }
    }
    else if (I2CSLST & 0x08)
    {
        I2CSLST &= ~0x08;                   //处理 STOP 事件
        isda = 1;
        isma = 1;
    }

    _pop_(P_SW2);
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
```

```
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0x81;                          //使能 I2C 从机模式
    I2CSLADR = 0x5a;                        //设置从机设备地址为 5A
    I2CSLST = 0x00;
    I2CSLCR = 0x78;                         //使能从机模式中断
    EA = 1;

    isda = 1;                               //用户变量初始化
    isma = 1;
    addr = 0;
    I2CTXD = buffer[addr];

    while (1);
}
```

19.4.5 I²C从机模式（查询方式）

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5
ISDA        BIT     20H.0               ;设备地址标志
ISMA        BIT     20H.1               ;存储地址标志

ADDR        DATA    21H

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
```

```
        MOV     P5M0, #00H
        MOV     P5M1, #00H
        MOV     P_SW2,#80H

        MOV     P_SW2,#80H

        MOV     A,#10000001B        ;使能 I2C 从机模式
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#01011010B        ;设置从机设备地址为 5A
        MOV     DPTR,#I2CSLADR
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CSLST
        MOVX    @DPTR,A
        MOV     A,#00000000B        ;禁止从机模式中断
        MOV     DPTR,#I2CSLCR
        MOVX    @DPTR,A

        SETB    ISDA                ;用户变量初始化
        SETB    ISMA
        CLR     A
        MOV     ADDR,A
        MOV     R0,A
        MOVX    A,@R0
        MOV     DPTR,#I2CTXD
        MOVX    @DPTR,A

LOOP:
        MOV     DPTR,#I2CSLST       ;检测从机状态
        MOVX    A,@DPTR
        JB      ACC.6,STARTIF
        JB      ACC.5,RXIF
        JB      ACC.4,TXIF
        JB      ACC.3,STOPIF
        JMP     LOOP
STARTIF:
        ANL     A,#NOT 40H          ;处理 START 事件
        MOVX    @DPTR,A
        JMP     LOOP
RXIF:
        ANL     A,#NOT 20H          ;处理 RECV 事件
        MOVX    @DPTR,A
        MOV     DPTR,#I2CRXD
        MOVX    A,@DPTR
        JBC     ISDA,RXDA
        JBC     ISMA,RXMA
        MOV     R0,ADDR             ;处理 RECV 事件（RECV DATA）
        MOVX    @R0,A
        INC     ADDR
        JMP     LOOP
RXDA:
        JMP     LOOP                ;处理 RECV 事件（RECV DEVICE ADDR）
RXMA:
        MOV     ADDR,A              ;处理 RECV 事件（RECV MEMORY ADDR）
        MOV     R0,A
        MOVX    A,@R0
        MOV     DPTR,#I2CTXD
        MOVX    @DPTR,A
```

```
                JMP         LOOP
TXIF:
                ANL         A,#NOT 10H              ;处理 SEND 事件
                MOVX        @DPTR,A
                JB          ACC.1,RXNAK
                INC         ADDR
                MOV         R0,ADDR
                MOVX        A,@R0
                MOV         DPTR,#I2CTXD
                MOVX        @DPTR,A
                JMP         LOOP
RXNAK:
                MOVX        A,#0FFH
                MOV         DPTR,#I2CTXD
                MOVX        @DPTR,A
                JMP         LOOP
STOPIF:
                ANL         A,#NOT 08H              ;处理 STOP 事件
                MOVX        @DPTR,A
                SETB        ISDA
                SETB        ISMA
                JMP         LOOP

                END
```

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr         P_SW2       =   0xba;

#define     I2CCFG          (*(unsigned char volatile xdata *)0xfe80)
#define     I2CMSCR         (*(unsigned char volatile xdata *)0xfe81)
#define     I2CMSST         (*(unsigned char volatile xdata *)0xfe82)
#define     I2CSLCR         (*(unsigned char volatile xdata *)0xfe83)
#define     I2CSLST         (*(unsigned char volatile xdata *)0xfe84)
#define     I2CSLADR        (*(unsigned char volatile xdata *)0xfe85)
#define     I2CTXD          (*(unsigned char volatile xdata *)0xfe86)
#define     I2CRXD          (*(unsigned char volatile xdata *)0xfe87)

sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        SDA         =   P1^4;
sbit        SCL         =   P1^5;

bit         isda;                               //设备地址标志
bit         isma;                               //存储地址标志
unsigned char               addr;
unsigned char pdata         buffer[256];

void main()
```

```
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0x81;                          //使能 I2C 从机模式
    I2CSLADR = 0x5a;                        //设置从机设备地址为 5A
    I2CSLST = 0x00;
    I2CSLCR = 0x00;                         //禁止从机模式中断

    isda = 1;                               //用户变量初始化
    isma = 1;
    addr = 0;
    I2CTXD = buffer[addr];

    while (1)
    {
        if (I2CSLST & 0x40)
        {
            I2CSLST &= ~0x40;               //处理 START 事件
        }
        else if (I2CSLST & 0x20)
        {
            I2CSLST &= ~0x20;               //处理 RECV 事件
            if (isda)
            {
                isda = 0;                   //处理 RECV 事件（RECV DEVICE ADDR）
            }
            else if (isma)
            {
                isma = 0;                   //处理 RECV 事件（RECV MEMORY ADDR）
                addr = I2CRXD;
                I2CTXD = buffer[addr];
            }
            else
            {
                buffer[addr++] = I2CRXD;    //处理 RECV 事件（RECV DATA）
            }
        }
        else if (I2CSLST & 0x10)
        {
            I2CSLST &= ~0x10;               //处理 SEND 事件
            if (I2CSLST & 0x02)
            {
                I2CTXD = 0xff;              //接收到 NAK 则停止读取数据
            }
            else
            {
                I2CTXD = buffer[++addr];    //接收到 ACK 则继续读取数据
            }
        }
        else if (I2CSLST & 0x08)
        {
            I2CSLST &= ~0x08;               //处理 STOP 事件
```

```
                isda = 1;
                isma = 1;
            }
        }
}
```

19.4.6 测试I²C从机模式代码的主机代码

汇编代码

```
;测试工作频率为11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5

P1M1        DATA    091H
P1M0        DATA    092H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
START:
            MOV     A,#00000001B        ;发送START命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDDATA:
            MOV     DPTR,#I2CTXD        ;写数据到数据缓冲区
            MOVX    @DPTR,A
            MOV     A,#00000010B        ;发送SEND命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVACK:
            MOV     A,#00000011B        ;发送读ACK命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVDATA:
            MOV     A,#00000100B        ;发送RECV命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            CALL    WAIT
```

```
                MOV     DPTR,#I2CRXD            ;从数据缓冲区读取数据
                MOVX    A,@DPTR
                RET
SENDACK:
                MOV     A,#00000000B            ;设置 ACK 信号
                MOV     DPTR,#I2CMSST
                MOVX    @DPTR,A
                MOV     A,#00000101B            ;发送 ACK 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
SENDNAK:
                MOV     A,#00000001B            ;设置 NAK 信号
                MOV     DPTR,#I2CMSST
                MOVX    @DPTR,A
                MOV     A,#00000101B            ;发送 ACK 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
STOP:
                MOV     A,#00000110B            ;发送 STOP 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
WAIT:
                MOV     DPTR,#I2CMSST           ;清中断标志
                MOVX    A,@DPTR
                JNB     ACC.6,WAIT
                ANL     A,#NOT 40H
                MOVX    @DPTR,A
                RET

DELAY:
                MOV     R0,#0
                MOV     R1,#0
DELAY1:
                NOP
                NOP
                NOP
                NOP
                DJNZ    R1,DELAY1
                DJNZ    R0,DELAY1
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H
                MOV     P_SW2,#80H

                MOV     A,#11100000B            ;设置 I2C 模块为主机模式
                MOV     DPTR,#I2CCFG
                MOVX    @DPTR,A
                MOV     A,#00000000B
                MOV     DPTR,#I2CMSST
```

```
        MOVX    @DPTR,A

        CALL    START           ;发送起始命令
        MOV     A,#5AH          ;从机地址为5A
        CALL    SENDDATA        ;发送设备地址+写命令
        CALL    RECVACK
        MOV     A,#000H         ;发送存储地址
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#12H          ;写测试数据1
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#78H          ;写测试数据2
        CALL    SENDDATA
        CALL    RECVACK
        CALL    STOP            ;发送停止命令

        CALL    DELAY           ;等待设备写数据

        CALL    START           ;发送起始命令
        MOV     A,#5AH          ;发送设备地址+写命令
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H         ;发送存储地址
        CALL    SENDDATA
        CALL    RECVACK
        CALL    START           ;发送起始命令
        MOV     A,#5BH          ;发送设备地址+读命令
        CALL    SENDDATA
        CALL    RECVACK
        CALL    RECVDATA        ;读取数据1
        MOV     P0,A
        CALL    SENDACK
        CALL    RECVDATA        ;读取数据2
        MOV     P2,A
        CALL    SENDNAK
        CALL    STOP            ;发送停止命令

        JMP     $

        END
```

C 语言代码

```
//测试工作频率为11.0592MHz

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

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD      (*(unsigned char volatile xdata *)0xfe87)
```

```
sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        SDA             =   P1^4;
sbit        SCL             =   P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                             //发送 START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                               //写数据到数据缓冲区
    I2CMSCR = 0x02;                             //发送 SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                             //发送读 ACK 命令
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                             //发送 RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                             //设置 ACK 信号
    I2CMSCR = 0x05;                             //发送 ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                             //设置 NAK 信号
    I2CMSCR = 0x05;                             //发送 ACK 命令
    Wait();
}

void Stop()
{
```

```
    I2CMSCR = 0x06;                         //发送 STOP 命令
    Wait();
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0xe0;                          //使能 I2C 主机模式
    I2CMSST = 0x00;

    Start();                                //发送起始命令
    SendData(0x5a);                         //发送设备地址+写命令
    RecvACK();
    SendData(0x00);                         //发送存储地址
    RecvACK();
    SendData(0x12);                         //写测试数据 1
    RecvACK();
    SendData(0x78);                         //写测试数据 2
    RecvACK();
    Stop();                                 //发送停止命令

    Start();                                //发送起始命令
    SendData(0x5a);                         //发送设备地址+写命令
    RecvACK();
    SendData(0x00);                         //发送存储地址高字节
    RecvACK();
    Start();                                //发送起始命令
    SendData(0x5b);                         //发送设备地址+读命令
    RecvACK();
    P0 = RecvData();                        //读取数据 1
    SendACK();
    P2 = RecvData();                        //读取数据 2
    SendNAK();
    Stop();                                 //发送停止命令

    P_SW2 = 0x00;

    while (1);
```

```
}
```

20 增强型双数据指针

STC8G 系列的单片机内部集成了两组 16 位的数据指针。通过程序控制, 可实现数据指针自动递增或递减功能以及两组数据指针的自动切换功能

相关的特殊功能寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DPL	数据指针（低字节）	82H									0000,0000
DPH	数据指针（高字节）	83H									0000,0000
DPL1	第二组数据指针（低字节）	E4H									0000,0000
DPH1	第二组数据指针（高字节）	E5H									0000,0000
DPS	DPTR 指针选择器	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL	0000,0xx0
TA	DPTR 时序控制寄存器	AEH									0000,0000

第 1 组 16 位数据指针寄存器（DPTR0）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPL	82H								
DPH	83H								

DPL为低8位数据（低字节）

DPH为高8位数据（高字节）

DPL和DPH组合为第一组16位数据指针寄存器DPTR0

第 2 组 16 位数据指针寄存器（DPTR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPL1	E4H								
DPH1	E5H								

DPL1为低8位数据（低字节）

DPH1为高8位数据（高字节）

DPL1和DPH1组合为第二组16位数据指针寄存器DPTR1

数据指针控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPS	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL

ID1：控制DPTR1自动递增方式

0：DPTR1 自动递增

1：DPTR1 自动递减

ID0：控制DPTR0自动递增方式

0：DPTR0 自动递增

1：DPTR0 自动递减

TSL：DPTR0/DPTR1自动切换控制（自动对SEL进行取反）

0：关闭自动切换功能

1：使能自动切换功能

当 TSL 位被置 1 后，每当执行完成相关指令后，系统会自动将 SEL 位取反。

与 TSL 相关的指令包括如下指令：

```
MOV     DPTR,#data16
INC     DPTR
MOVC    A,@A+DPTR
MOVX    A,@DPTR
MOVX    @DPTR,A
```

AU1/AU0：使能DPTR1/DPTR0使用ID1/ID0控制位进行自动递增/递减控制

0：关闭自动递增/递减功能

1：使能自动递增/递减功能

注意：在写保护模式下，AU0 和 AU1 位无法直接单独使能，若单独使能 AU1 位，则 AU0 位也会被自动使能，若单独使能 AU0，没有效果。若需要单独使能 AU1 或者 AU0，则必须使用 TA 寄存器触发 DPS 的保护机制（参考 TA 寄存器的说明）。另外，只有执行下面的 3 条指令后才会对 DPTR0/DPTR1 进行自动递增/递减操作。3 条相关指令如下：

```
MOVC    A,@A+DPTR
MOVX    A,@DPTR
MOVX    @DPTR,A
```

SEL：选择DPTR0/DPTR1作为当前的目标DPTR

0：选择 DPTR0 作为目标 DPTR

1：选择 DPTR1 作为目标 DPTR

SEL 选择目标 DPTR 对下面指令有效：

```
MOV     DPTR,#data16
INC     DPTR
MOVC    A,@A+DPTR
MOVX    A,@DPTR
MOVX    @DPTR,A
JMP     @A+DPTR
```

数据指针控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TA	AEH								

TA寄存器是对DPS寄存器中的AU1和AU0进行写保护的。由于程序无法对DPS中的AU1和AU0进行单独的写入，所以当需要单独使能AU1或者AU0时，必须使用TA寄存器进行触发。TA寄存器是只写寄存器。当需要对AU1或者AU0进行单独使能时，必须按照如下的步骤进行操作：

```
CLR     EA          ;关闭中断（必需）
MOV     TA,#0AAH    ;写入触发命令序列 1
                    ;此处不能有其他任何指令
MOV     TA,#55H     ;写入触发命令序列 2
                    ;此处不能有其他任何指令
MOV     DPS,#xxH    ;写保护暂时关闭，可向 DPS 中写入任何值
                    ;DSP 再次进行写保护状态
SETB    EA          ;打开中断（如有必要）
```

20.1 范例程序

20.1.1 示例代码 1

将程序空间 1000H～1003H 的 4 个字节数据反向复制到扩展 RAM 的 0100H～0103H 中，即

C:1000H -> X:0103H
C:1001H -> X:0102H
C:1002H -> X:0101H
C:1003H -> X:0100H

汇编代码

```
;测试工作频率为 11.0592MHz

P1M1        DATA        091H
P1M0        DATA        092H
P3M1        DATA        0B1H
P3M0        DATA        0B2H
P5M1        DATA        0C9H
P5M0        DATA        0CAH

            ORG         0000H
            LJMP        MAIN

            ORG         0100H
MAIN:
            MOV         SP, #5FH
            MOV         P1M0, #00H
            MOV         P1M1, #00H
            MOV         P3M0, #00H
            MOV         P3M1, #00H
            MOV         P5M0, #00H
            MOV         P5M1, #00H

            MOV         DPS,#00100000B      ;使能 TSL,并选择 DPTR0
            MOV         DPTR,#1000H         ;将 1000H 写入 DPTR0 后选择 DPTR1 为 DPTR
            MOV         DPTR,#0103H         ;将 0103H 写入 DPTR1 中
            MOV         DPS,#10111000B      ;设置 DPTR1 为递减模式,DPTR0 为递加模式,使能 TSL
                                            ;AU0 和 AU1,并选择 DPTR0 为当前的 DPTR
            MOV         R7,#4               ;设置数据复制个数
COPY_NEXT:
            CLR         A                   ;
            MOVC        A,@A+DPTR           ;从 DPTR0 所指的程序空间读取数据,
                                            ;完成后 DPTR0 自动加 1 并将 DPTR1 设置为 DPTR
            MOVX        @DPTR,A             ;将 ACC 的数据写入到 DPTR1 所指的 XDATA 中,
                                            ;完成后 DPTR1 自动减 1 并将 DPTR0 设置为 DPTR
            DJNZ        R7,COPY_NEXT        ;

            SJMP        $

            END
```

20.1.2 示例代码 2

将扩展 RAM 的 0100H～0103H 中的数据依次发送到 P0 口

汇编代码

```
;测试工作频率为11.0592MHz

P1M1        DATA        091H
P1M0        DATA        092H
P3M1        DATA        0B1H
P3M0        DATA        0B2H
P5M1        DATA        0C9H
P5M0        DATA        0CAH

            ORG         0000H
            LJMP        MAIN

            ORG         0100H
MAIN:
            MOV         SP, #5FH
            MOV         P1M0, #00H
            MOV         P1M1, #00H
            MOV         P3M0, #00H
            MOV         P3M1, #00H
            MOV         P5M0, #00H
            MOV         P5M1, #00H

            CLR         EA                  ;关闭中断
            MOV         TA,#0AAH            ;写入 DPS 写保护触发命令 1
            MOV         TA,#55H             ;写入 DPS 写保护触发命令 2
            MOV         DPS,#00001000B      ;DPTR0 递增,单独使能 AU0,并选择 DPTR0
            SETB        EA                  ;打开中断
            MOV         DPTR,#0100H         ;将 0100H 写入 DPTR0 中
            MOVX        A,@DPTR             ;从 DPTR0 所指的 XRAM 读取数据后 DPTR0 自动加 1
            MOV         P0,A                ;数据输出到 P0 口
            MOVX        A,@DPTR             ;从 DPTR0 所指的 XRAM 读取数据后 DPTR0 自动加 1
            MOV         P0,A                ;数据输出到 P0 口
            MOVX        A,@DPTR             ;从 DPTR0 所指的 XRAM 读取数据后 DPTR0 自动加 1
            MOV         P0,A                ;数据输出到 P0 口
            MOVX        A,@DPTR             ;从 DPTR0 所指的 XRAM 读取数据后 DPTR0 自动加 1
            MOV         P0,A                ;数据输出到 P0 口

            SJMP        $

            END
```

附录C　使用第三方MCU对STC8G系列单片机进行ISP下载范例程序

C 语言代码

```
//注意:使用本代码对 STC8G 系列的单片机进行下载时,必须要执行了 Download 代码之后,
//才能给目标芯片上电,否则目标芯片将无法正确下载

#include "reg51.h"

typedef    bit              BOOL;
typedef    unsigned char    BYTE;
typedef    unsigned short   WORD;

//宏、常量定义
#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          //主控芯片工作频率
#define    BR(n)            (65536 - FOSC/4/(n))    //主控芯片串口波特率计算公式
#define    T1MS             (65536 - FOSC/1000)     //主控芯片 1ms 定时初值

#define    FUSER            24000000L          //STC8G 系列目标芯片工作频率
#define    RL(n)            (65536 - FUSER/4/(n))   //STC8G 系列目标芯片串口波特率计算公式

sfr        AUXR = 0x8e;
sfr        P3M1 = 0xB1;
sfr        P3M0 = 0xB2;

//变量定义
BOOL f1ms;                          //1ms 标志位
BOOL UartBusy;                      //串口发送忙标志位
BOOL UartReceived;                  //串口数据接收完成标志位
BYTE UartRecvStep;                  //串口数据接收控制
BYTE TimeOut;                       //串口通讯超时计数器
BYTE xdata TxBuffer[256];           //串口数据发送缓冲区
BYTE xdata RxBuffer[256];           //串口数据接收缓冲区
char code DEMO[256];                //演示代码数据

//函数声明
void Initial(void);
void DelayXms(WORD x);
BYTE UartSend(BYTE dat);
void CommInit(void);
void CommSend(BYTE size);
BOOL Download(BYTE *pdat, long size);
```

```
//主函数入口
void main(void)
{
	P3M0 = 0x00;
	P3M1 = 0x00;

	Initial();
	if (Download(DEMO, 256))
	{
		//下载成功
		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
	{
		//下载失败
		P3 = 0xff;
		DelayXms(500);
		P3 = 0xf3;
		DelayXms(500);
		P3 = 0xff;
		DelayXms(500);
		P3 = 0xf3;
		DelayXms(500);
		P3 = 0xff;
		DelayXms(500);
		P3 = 0xf3;
		DelayXms(500);
		P3 = 0xff;
	}

	while (1);
}

//1ms 定时器中断服务程序
void tm0(void) interrupt 1
{
	static BYTE Counter100;

	f1ms = TRUE;
	if (Counter100-- == 0)
	{
		Counter100 = 100;
		if (TimeOut) TimeOut--;
	}
}
```

```
//串口中断服务程序
void uart(void) interrupt 4
{
    static WORD RecvSum;
    static BYTE RecvIndex;
    static BYTE RecvCount;
    BYTE dat;

    if (TI)
    {
        TI = 0;
        UartBusy = FALSE;
    }

    if (RI)
    {
        RI = 0;
        dat  = SBUF;
        switch (UartRecvStep)
        {
        case 1:
            if (dat != 0xb9) goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 2:
            if (dat != 0x68) goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 3:
            if (dat != 0x00) goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 4:
            RecvSum = 0x68 + dat;
            RecvCount = dat - 6;
            RecvIndex = 0;
            UartRecvStep++;
            break;
        case 5:
            RecvSum += dat;
            RxBuffer[RecvIndex++] = dat;
            if (RecvIndex == RecvCount)  UartRecvStep++;
            break;
        case 6:
            if (dat != HIBYTE(RecvSum))goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 7:
            if (dat != LOBYTE(RecvSum))  goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 8:
            if (dat != 0x16) goto L_CheckFirst;
            UartReceived = TRUE;
            UartRecvStep++;
            break;
L_CheckFirst:
        case 0:
        default:
```

```
                CommInit();
                UartRecvStep = (dat == 0x46? 1 : 0);
                break;
            }
        }
    }
}

//系统初始化
void Initial(void)
{
    UartBusy = FALSE;

    SCON = 0xd0;                        //串口数据模式必须为 8 位数据+1 位偶检验
    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 延时程序
void DelayXms(WORD x)
{
    do
    {
        f1ms = FALSE;
        while (!f1ms);
    } while (x--);
}

//串口数据发送程序
BYTE UartSend(BYTE dat)
{
    while (UartBusy);

    UartBusy = TRUE;
    ACC = dat;
    TB8 = P;
    SBUF = ACC;

    return dat;
}

//串口通讯初始化
void CommInit(void)
{
    UartRecvStep = 0;
    TimeOut = 20;
    UartReceived = FALSE;
}

//发送串口通讯数据包
void CommSend(BYTE size)
```

```
{
	WORD sum;
	BYTE i;

	UartSend(0x46);
	UartSend(0xb9);
	UartSend(0x6a);
	UartSend(0x00);
	sum = size + 6 + 0x6a;
	UartSend(size + 6);
	for (i=0; i<size; i++)
	{
		sum  += UartSend(TxBuffer[i]);
	}
	UartSend(HIBYTE(sum));
	UartSend(LOBYTE(sum));
	UartSend(0x16);
	while (UartBusy);

	CommInit();
}

//对 STC15H 系列的芯片进行 ISP 下载程序
BOOL Download(BYTE *pdat, long size)
{
	BYTE arg;
	BYTE offset;
	BYTE cnt;
	WORD addr;

	//握手
	CommInit();
	while (1)
	{
		if (UartRecvStep == 0)
		{
			UartSend(0x7f);
			DelayXms(10);
		}
		if (UartReceived)
		{
			arg = RxBuffer[4];
			if (RxBuffer[0] == 0x50) break;
			return FALSE;
		}
	}

	//设置参数(设置从芯片使用最高的波特率以及擦除等待时间等参数)
	TxBuffer[0] = 0x01;
	TxBuffer[1] = arg;
	TxBuffer[2] = 0x40;
	TxBuffer[3] = HIBYTE(RL(MAXBAUD));
	TxBuffer[4] = LOBYTE(RL(MAXBAUD));
	TxBuffer[5] = 0x00;
	TxBuffer[6] = 0x00;
	TxBuffer[7] = 0x97;
	CommSend(8);
	while (1)
	{
```

```
        if (TimeOut == 0) return FALSE;
        if (UartReceived)
        {
            if (RxBuffer[0] == 0x01) break;
            return FALSE;
        }
    }

    //准备
    TH1 = HIBYTE(BR(MAXBAUD));
    TL1 = LOBYTE(BR(MAXBAUD));
    DelayXms(10);
    TxBuffer[0] = 0x05;
    TxBuffer[1] = 0x00;
    TxBuffer[2] = 0x00;
    TxBuffer[3] = 0x5a;
    TxBuffer[4] = 0xa5;
    CommSend(5);
    while (1)
    {
        if (TimeOut == 0) return FALSE;
        if (UartReceived)
        {
            if (RxBuffer[0] == 0x05) break;
            return FALSE;
        }
    }

    //擦除
    DelayXms(10);
    TxBuffer[0] = 0x03;
    TxBuffer[1] = 0x00;
    TxBuffer[2] = 0x00;
    TxBuffer[3] = 0x5a;
    TxBuffer[4] = 0xa5;
    CommSend(5);
    TimeOut = 100;
    while (1)
    {
        if (TimeOut == 0) return FALSE;
        if (UartReceived)
        {
            if (RxBuffer[0] == 0x03) break;
            return FALSE;
        }
    }

    //写用户代码
    DelayXms(10);
    addr = 0;
    TxBuffer[0] = 0x22;
    TxBuffer[3] = 0x5a;
    TxBuffer[4] = 0xa5;
    offset = 5;
    while (addr < size)
    {
        TxBuffer[1] = HIBYTE(addr);
        TxBuffer[2] = LOBYTE(addr);
        cnt    = 0;
```

```
        while (addr < size)
        {
            TxBuffer[cnt+offset] = pdat[addr];
            addr++;
            cnt++;
            if (cnt >= 128) break;
        }
        CommSend(cnt + offset);
        while (1)
        {
            if (TimeOut == 0) return FALSE;
            if (UartReceived)
            {
                if ((RxBuffer[0] == 0x02) && (RxBuffer[1] == 'T')) break;
                return FALSE;
            }
        }
        TxBuffer[0] = 0x02;
    }

    ////写硬件选项
    ////如果不需要修改硬件选项,此步骤可直接跳过,此时所有的硬件选项
    ////都维持不变,MCU 的频率为上一次所调节频率
    ////若写硬件选项,MCU 的内部 IRC 频率将被固定写为 24M, ,其他选项恢复为出厂设置
    ////建议:第一次使用 STC-ISP 下载软件将从芯片的硬件选项设置好
    ////以后再使用主芯片对从芯片下载程序时不写硬件选项
    //DelayXms(10);
    //for (cnt=0; cnt<128; cnt++)
    //{
    //    TxBuffer[cnt] = 0xff;
    //}
    //TxBuffer[0] = 0x04;
    //TxBuffer[1] = 0x00;
    //TxBuffer[2] = 0x00;
    //TxBuffer[3] = 0x5a;
    //TxBuffer[4] = 0xa5;
    //TxBuffer[33] = arg;
    //TxBuffer[34] = 0x00;
    //TxBuffer[35] = 0x01;
    //TxBuffer[41] = 0xbf;
    //TxBuffer[42] = 0xbd;                  //P5.4 为 I/O 口
    ////TxBuffer[42] = 0xad;                //P5.4 为复位脚
    //TxBuffer[43] = 0xf7;
    //TxBuffer[44] = 0xff;
    //CommSend(45);
    //while (1)
    //{
    //    if (TimeOut == 0) return FALSE;
    //    if (UartReceived)
    //    {
    //        if ((RxBuffer[0] == 0x04) && (RxBuffer[1] == 'T')) break;
    //        return FALSE;
    //    }
    //}

    // 下载完成
    return TRUE;
}
```

```
char code DEMO[256] =
{
    0x80,0x00,0x75,0xB2,0xFF,0x75,0xB1,0x00,0x05,0xB0,0x11,0x0E,0x80,0xFA,0xD8,0xFE,
    0xD9,0xFC,0x22,
};
```

备注：用户若需要设置不同的工作频率，可参考 7.3.7 和 7.3.8 章的范例代码

附录D　电气特性

绝对最大额定值

参数	最小值	最大值	单位
存储温度	-55	+125	℃
工作温度	-40	+85	℃
工作电压	1.9	5.5	V
VDD 对地电压	-0.3	+5.5	V
I/O 口对地电压	-0.3	VDD+0.3	V

直流特性（VSS=0V，VDD=5.0V，测试温度=25℃）（**STC8G1K08 系列**）

标号	参数	范围				测试环境
		最小值	典型值	最大值	单位	
I_{PD}	掉电模式电流	-	0.6	-	uA	
I_{WKT}	掉电唤醒定时器	-	3.6	-	uA	
I_{LVD}	低压检测模块	-	427	-	uA	
I_{IDL}	空闲模式电流（6MHz）	-	1.0	-	mA	
	空闲模式电流（11.0592MHz）	-	1.16	-	mA	
	空闲模式电流（24MHz）	-	1.38	-	mA	
	空闲模式电流（内部 32KHz）	-	0.56	-	mA	
I_{NOR}	正常模式电流（6MHz）	-	1.66	-	mA	
	正常模式电流（11.0592MHz）	-	2.33	-	mA	
	正常模式电流（24MHz）	-	3.54	-	mA	
	正常模式电流（内部 32KHz）	-	0.56	-	mA	
I_{CC}	普通工作模式电流	-	4	20	mA	
V_{IL1}	输入低电平	-	-	1.35	V	打开施密特触发
		-	-	1.48	V	关闭施密特触发
V_{IH1}	输入高电平（普通 I/O）	1.60	-	-	V	打开施密特触发
		1.50	-	-	V	关闭施密特触发
V_{IH2}	输入高电平（复位脚）	1.60	-	1.35	V	
I_{OL1}	输出低电平的灌电流	-	20	-	mA	端口电压 0.45V
I_{OH1}	输出高电平电流（双向模式）	200	270	-	uA	
I_{OH2}	输出高电平电流（推挽模式）	-	20	-	mA	端口电压 2.4V
I_{IL}	逻辑 0 输入电流	-	-	50	uA	端口电压 0V
I_{TL}	逻辑 1 到 0 的转移电流	100	270	600	uA	端口电压 2.0V
R_{PU}	I/O 口上拉电阻	4.1	4.2	4.4	KΩ	

直流特性（VSS=0V，VDD=3.3V，测试温度=25℃）（**STC8G1K08 系列**）

标号	参数	范围				测试环境
		最小值	典型值	最大值	单位	
I_{PD}	掉电模式电流	-	0.4	-	uA	
I_{WKT}	掉电唤醒定时器	-	1.3	-	uA	
I_{LVD}	低压检测模块	-	362	-	uA	
I_{IDL}	空闲模式电流（6MHz）	-	0.91	-	mA	
	空闲模式电流（11.0592MHz）	-	1.06	-	mA	
	空闲模式电流（24MHz）	-	1.30	-	mA	
	空闲模式电流（内部 32KHz）	-	0.47	-	mA	
I_{NOR}	正常模式电流（6MHz）	-	1.57	-	mA	
	正常模式电流（11.0592MHz）	-	2.23	-	mA	
	正常模式电流（24MHz）	-	3.43	-	mA	
	正常模式电流（内部 32KHz）	-	0.47	-	mA	
I_{CC}	普通工作模式电流	-	4	20	mA	
V_{IL1}	输入低电平	-	-	1.00	V	打开施密特触发
		-	-	1.06	V	关闭施密特触发
V_{IH1}	输入高电平（普通 I/O）	1.16	-	-	V	打开施密特触发
		1.07	-	-	V	关闭施密特触发
V_{IH2}	输入高电平（复位脚）	1.16	-	1.00	V	
I_{OL1}	输出低电平的灌电流	-	20	-	mA	端口电压 0.45V
I_{OH1}	输出高电平电流（双向模式）	200	270	-	uA	
I_{OH2}	输出高电平电流（推挽模式）	-	20	-	mA	端口电压 2.4V
I_{IL}	逻辑 0 输入电流	-	-	50	uA	端口电压 0V
I_{TL}	逻辑 1 到 0 的转移电流	100	270	600	uA	端口电压 2.0V
R_{PU}	I/O 口上拉电阻	5.8	5.9	6.0	KΩ	

内部 IRC 温漂特性（参考温度 25℃）

温度	范围
-40℃～85℃	−1.35%～+1.30%
-20℃～65℃	−0.76%～+0.98%

低压复位门槛电压（测试温度 25℃）

级别	电压
LVR0	2.0V
LVR1	2.4V
LVR2	2.7V
LVR3	3.0V

附录E　应用注意事项

关于 STC8G 系列 I/O 口的注意事项

1. **STC8G 系列芯片的 I/O 口，除了 ISP 下载口 P3.0 和 P3.1 外，其余的 I/O 口上电后的初始模式均为高阻输入模式，用户无法直接输出电平，所以用户在程序初始化的地方必须要使用 PxM0 和 PxM1 两个寄存器初始化相应的 I/O 模式，才能正常使用。**
2. **STC8G 系列芯片所有的 I/O 口均可以设置为准双向口模式、强推挽输出模式、开漏输出模式或者高阻输入模式，另外每个 I/O 均可独立使能内部 4K 上拉电阻**
3. **STC8G 系列芯片不会自动为特殊 I/O 设置 I/O 口模式，如 ADC 口、串口、I2C 口以及 SPI 口，必须用户自行将相应的口设置为合适的模式**
4. **若使能 P5.4 管脚为复位脚，则复位电平为低电平**
5. **定时器 0 的模式 3 在使能 NMI 中断时，必须保持 EA 为打开状态**
6. **对于 STC8G1K08 系列 B 版芯片，P5.4 作 I/O 口使用时，电流不要超过 50mA，也不要有强的冲击**
7. **STC8G1K08 系列 B 版芯片所支持的 USB 下载为 I/O 口软件模拟的 USB，由于受制成、温度等多方面因素的影响，会导致有一定比例的芯片无法进行 USB 下载，经实际测试，无法 USB 下载的比例可能在 5%～8%**

附录F　STC8G系列头文件

```
#ifndef    __STC8G_H_
#define    __STC8G_H_

////////////////////////////////////////////////

//包含本头文件后,不用另外再包含"REG51.H"

sfr     SP      =   0x81;
sfr     DPL     =   0x82;
sfr     DPH     =   0x83;
sfr     PCON    =   0x87;
sfr     TCON    =   0x88;
sbit    TF1     =   TCON^7;
sbit    TR1     =   TCON^6;
sbit    TF0     =   TCON^5;
sbit    TR0     =   TCON^4;
sbit    IE1     =   TCON^3;
sbit    IT1     =   TCON^2;
sbit    IE0     =   TCON^1;
sbit    IT0     =   TCON^0;
sfr     TMOD    =   0x89;
sfr     TL0     =   0x8a;
sfr     TL1     =   0x8b;
sfr     TH0     =   0x8c;
sfr     TH1     =   0x8d;
sfr     AUXR    =   0x8e;
sfr     INTCLKO =   0x8f;
sfr     P1      =   0x90;
sbit    P10     =   P1^0;
sbit    P11     =   P1^1;
sbit    P12     =   P1^2;
sbit    P13     =   P1^3;
sbit    P14     =   P1^4;
sbit    P15     =   P1^5;
sbit    P16     =   P1^6;
sbit    P17     =   P1^7;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     SCON    =   0x98;
sbit    SM0     =   SCON^7;
sbit    SM1     =   SCON^6;
sbit    SM2     =   SCON^5;
sbit    REN     =   SCON^4;
sbit    TB8     =   SCON^3;
sbit    RB8     =   SCON^2;
sbit    TI      =   SCON^1;
sbit    RI      =   SCON^0;
sfr     SBUF    =   0x99;
sfr     S2CON   =   0x9a;
sfr     S2BUF   =   0x9b;
sfr     IRCBAND =   0x9d;
sfr     LIRTRIM =   0x9e;
```

```
sfr     IRTRIM  =   0x9f;
sfr     P_SW1   =   0xa2;
sfr     IE      =   0xa8;
sbit    EA      =   IE^7;
sbit    ELVD    =   IE^6;
sbit    EADC    =   IE^5;
sbit    ES      =   IE^4;
sbit    ET1     =   IE^3;
sbit    EX1     =   IE^2;
sbit    ET0     =   IE^1;
sbit    EX0     =   IE^0;
sfr     SADDR   =   0xa9;
sfr     WKTCL   =   0xaa;
sfr     WKTCH   =   0xab;
sfr     TA      =   0xae;
sfr     IE2     =   0xaf;
sfr     P3      =   0xb0;
sbit    P30     =   P3^0;
sbit    P31     =   P3^1;
sbit    P32     =   P3^2;
sbit    P33     =   P3^3;
sbit    P34     =   P3^4;
sbit    P35     =   P3^5;
sbit    P36     =   P3^6;
sbit    P37     =   P3^7;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     IP2     =   0xb5;
sfr     IP2H    =   0xb6;
sfr     IPH     =   0xb7;
sfr     IP      =   0xb8;
sbit    PPCA    =   IP^7;
sbit    PLVD    =   IP^6;
sbit    PADC    =   IP^5;
sbit    PS      =   IP^4;
sbit    PT1     =   IP^3;
sbit    PX1     =   IP^2;
sbit    PT0     =   IP^1;
sbit    PX0     =   IP^0;
sfr     SADEN   =   0xb9;
sfr     P_SW2   =   0xba;
sfr     VOCTRL  =   0xbb;
sfr     ADC_CONTR   =   0xbc;
sfr     ADC_RES     =   0xbd;
sfr     ADC_RESL    =   0xbe;
sfr     WDT_CONTR   =   0xc1;
sfr     IAP_DATA    =   0xc2;
sfr     IAP_ADDRH   =   0xc3;
sfr     IAP_ADDRL   =   0xc4;
sfr     IAP_CMD     =   0xc5;
sfr     IAP_TRIG    =   0xc6;
sfr     IAP_CONTR   =   0xc7;
sfr     P5      =   0xc8;
sbit    P54     =   P5^4;
sbit    P55     =   P5^5;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;
sfr     SPSTAT  =   0xcd;
sfr     SPCTL   =   0xce;
```

```
sfr     SPDAT   =   0xcf;
sfr     PSW     =   0xd0;
sbit    CY      =   PSW^7;
sbit    AC      =   PSW^6;
sbit    F0      =   PSW^5;
sbit    RS1     =   PSW^4;
sbit    RS0     =   PSW^3;
sbit    OV      =   PSW^2;
sbit    P       =   PSW^0;
sfr     T2H     =   0xd6;
sfr     T2L     =   0xd7;
sfr     CCON    =   0xd8;
sbit    CF      =   CCON^7;
sbit    CR      =   CCON^6;
sbit    CCF2    =   CCON^2;
sbit    CCF1    =   CCON^1;
sbit    CCF0    =   CCON^0;
sfr     CMOD    =   0xd9;
sfr     CCAPM0  =   0xda;
sfr     CCAPM1  =   0xdb;
sfr     CCAPM2  =   0xdc;
sfr     ADCCFG  =   0xde;
sfr     ACC     =   0xe0;
sfr     DPS     =   0xe3;
sfr     DPL1    =   0xe4;
sfr     DPH1    =   0xe5;
sfr     CMPCR1  =   0xe6;
sfr     CMPCR2  =   0xe7;
sfr     CL      =   0xe9;
sfr     CCAP0L  =   0xea;
sfr     CCAP1L  =   0xeb;
sfr     CCAP2L  =   0xec;
sfr     AUXINTIF=   0xef;
sfr     B       =   0xf0;
sfr     PCA_PWM0=   0xf2;
sfr     PCA_PWM1=   0xf3;
sfr     PCA_PWM2=   0xf4;
sfr     IAP_TPS =   0xf5;
sfr     CH      =   0xf9;
sfr     CCAP0H  =   0xfa;
sfr     CCAP1H  =   0xfb;
sfr     CCAP2H  =   0xfc;
sfr     RSTCFG  =   0xff;

//如下特殊功能寄存器位于扩展 RAM 区域
//访问这些寄存器,需先将 P_SW2 的 BIT7 设置为 1,才可正常读写
#define CKSEL       (*(unsigned char volatile xdata *)0xfe00)
#define CLKDIV      (*(unsigned char volatile xdata *)0xfe01)
#define IRC24MCR    (*(unsigned char volatile xdata *)0xfe02)
#define XOSCCR      (*(unsigned char volatile xdata *)0xfe03)
#define IRC32KCR    (*(unsigned char volatile xdata *)0xfe04)
#define MCLKOCR     (*(unsigned char volatile xdata *)0xfe05)

#define P1PU        (*(unsigned char volatile xdata *)0xfe11)
#define P3PU        (*(unsigned char volatile xdata *)0xfe13)
#define P5PU        (*(unsigned char volatile xdata *)0xfe15)
#define P1NCS       (*(unsigned char volatile xdata *)0xfe19)
#define P3NCS       (*(unsigned char volatile xdata *)0xfe1b)
#define P5NCS       (*(unsigned char volatile xdata *)0xfe1d)
```

```
#define    P1SR          (*(unsigned char volatile xdata *)0xfe21)
#define    P3SR          (*(unsigned char volatile xdata *)0xfe23)
#define    P5SR          (*(unsigned char volatile xdata *)0xfe25)
#define    P1DR          (*(unsigned char volatile xdata *)0xfe29)
#define    P3DR          (*(unsigned char volatile xdata *)0xfe2b)
#define    P5DR          (*(unsigned char volatile xdata *)0xfe2d)
#define    P1IE          (*(unsigned char volatile xdata *)0xfe31)
#define    P3IE          (*(unsigned char volatile xdata *)0xfe33)
#define    I2CCFG        (*(unsigned char volatile xdata *)0xfe80)
#define    I2CMSCR       (*(unsigned char volatile xdata *)0xfe81)
#define    I2CMSST       (*(unsigned char volatile xdata *)0xfe82)
#define    I2CSLCR       (*(unsigned char volatile xdata *)0xfe83)
#define    I2CSLST       (*(unsigned char volatile xdata *)0xfe84)
#define    I2CSLADR      (*(unsigned char volatile xdata *)0xfe85)
#define    I2CTXD        (*(unsigned char volatile xdata *)0xfe86)
#define    I2CRXD        (*(unsigned char volatile xdata *)0xfe87)
#define    I2CMSAUX      (*(unsigned char volatile xdata *)0xfe88)
#define    TM2PS         (*(unsigned char volatile xdata *)0xfea2)
#define    ADCTIM        (*(unsigned char volatile xdata *)0xfea8)

//////////////////////////////////////////////////

#endif
```

附录G 更新记录

- **2019/10/22**

1. 增加 QFN20 管脚图
2. 增加 QFN20 封装尺寸图
3. 更新范例程序

- **2019/10/15**

1. 修正 LVR 的四级电压
2. 修正内部高精度 IRC 的温漂范围
3. 修正内部参考电压
4. 更新直流特性表格数据

- **2019/10/09**

1. 移除电源控制寄存器（VOCTRL）部分，STC8G 系列无此功能
2. 修改 LVR 的四级电压
3. 修正 IRC 的两个频率范围

- **2019/8/13**

1. 创建 STC8G 系列单片机技术参考手册文档