

# STC8A8K64D4 系列单片机 技术参考手册

STC MCU

# 目录

<b>1</b>	<b>概述</b> .....	<b>1</b>
<b>2</b>	<b>特性、价格及管脚</b> .....	<b>2</b>
2.1	STC8A8K64D4-LQFP64/48/44, PDIP40 系列 .....	2
2.1.1	特性及价格 (有 16 位硬件乘除法器 MDU16, 准 16 位单片机) .....	2
2.1.2	管脚图, 最小系统 .....	5
2.1.3	管脚说明 .....	9
<b>3</b>	<b>功能脚切换</b> .....	<b>16</b>
3.1	功能脚切换相关寄存器 .....	16
3.1.1	总线速度控制寄存器 (BUS_SPEED) .....	17
3.1.2	外设端口切换控制寄存器 1 (P_SW1), 串口 1、CCP、SPI 切换 .....	17
3.1.3	外设端口切换控制寄存器 2 (P_SW2), 串口 2/3/4、I2C、比较器输出切换 .....	17
3.1.4	时钟选择寄存器 (MCLKOCR) .....	18
3.1.5	增强型 PWM 控制寄存器 (PWMnCR) .....	18
3.1.6	LCM 接口配置寄存器 (LCMIFCFG) .....	21
3.2	范例程序 .....	22
3.2.1	串口 1 切换 .....	22
3.2.2	串口 2 切换 .....	23
3.2.3	串口 3 切换 .....	25
3.2.4	串口 4 切换 .....	26
3.2.5	SPI 切换 .....	28
3.2.6	PWM 切换 .....	29
3.2.7	PCA/CCP/PWM 切换 .....	32
3.2.8	I2C 切换 .....	34
3.2.9	比较器输出切换 .....	35
3.2.10	主时钟输出切换 .....	37
<b>4</b>	<b>封装尺寸图</b> .....	<b>40</b>
4.1	LQFP44 封装尺寸图 (12mm*12mm) .....	40
4.2	LQFP48 封装尺寸图 (9mm*9mm) .....	41
4.3	QFN48 封装尺寸图 (6mm*6mm) .....	42
4.4	LQFP64 封装尺寸图 (12mm*12mm) .....	43
4.5	QFN64 封装尺寸图 (8mm*8mm) .....	44
4.6	STC8A8K64D4 系列单片机命名规则 .....	45
<b>5</b>	<b>ISP 下载及典型应用线路图</b> .....	<b>46</b>
5.1	STC8A8K64D4 系列 ISP 下载应用线路图 .....	46
5.1.1	使用 STC-USB Link1D 工具下载, 支持在线和脱机下载 .....	46
5.1.2	软件模拟 USB 直接 ISP 下载, 建议尝试, 不支持仿真 (5V 系统) .....	48
5.1.3	软件模拟 USB 直接 ISP 下载, 建议尝试, 不支持仿真 (3.3V 系统) .....	50
5.1.4	使用 RS-232 转换器下载, 也可支持仿真 (使用高精度 ADC) .....	52
5.1.5	使用 RS-232 转换器下载, 也可支持仿真 (使用一般精度 ADC) .....	53
5.1.6	使用 PL2303-GL 下载, 也可支持仿真 .....	54

5.1.7	使用通用 USB 转串口工具下载, 支持 ISP 在线下载, 也可支持仿真 .....	55
5.1.8	使用 U8-Mini 工具下载, 支持 ISP 在线和脱机下载, 也可支持仿真 .....	56
5.1.9	使用 U8W 工具下载, 支持 ISP 在线和脱机下载, 也可支持仿真 .....	57
5.1.10	单片机电源控制参考电路 .....	59
5.2	STC-ISP 下载软件高级应用 .....	60
5.2.1	发布项目程序 .....	60
5.2.2	程序加密后传输 (防烧录时串口分析出程序) .....	64
5.2.3	发布项目程序+程序加密后传输结合使用 .....	68
5.2.4	用户自定义下载 (实现不停电下载) .....	69
5.3	驱动程序数字签名的相关说明 .....	73
5.3.1	关于驱动程序强制数字签名 .....	73
5.3.2	Windows 10 关闭驱动程序强制数字签名步骤 .....	74
<b>6</b>	<b>时钟、复位、省电模式与系统电源管理 .....</b>	<b>79</b>
6.1	系统时钟控制 .....	79
6.1.1	系统时钟选择寄存器 (CLKSEL) .....	79
6.1.2	时钟分频寄存器 (CLKDIV) .....	80
6.1.3	内部高速高精度 IRC 控制寄存器 (HIRCCR) .....	80
6.1.4	外部振荡器控制寄存器 (XOSCCR) .....	80
6.1.5	内部 32KHz 低速 IRC 控制寄存器 (IRC32KCR) .....	81
6.1.6	主时钟输出控制寄存器 (MCLKOCR) .....	81
6.2	STC8A8K64D4 系列内部 IRC 频率调整 .....	82
6.2.1	IRC 频段选择寄存器 (IRCBAND) .....	82
6.2.2	内部 IRC 频率调整寄存器 (IRTRIM) .....	82
6.2.3	内部 IRC 频率微调寄存器 (LIRTRIM) .....	83
6.2.4	时钟分频寄存器 (CLKDIV) .....	83
6.2.5	分频出 3MHz 用户工作频率, 并用户动态改变频率追频示例 .....	84
6.3	系统复位 .....	87
6.3.1	看门狗复位 (WDT_CONTR) .....	88
6.3.2	软件复位 (IAP_CONTR) .....	90
6.3.3	低压复位 (RSTCFG) .....	91
6.3.4	低电平上电复位参考电路 (一般不需要) .....	92
6.3.5	低电平按键手动复位参考电路 .....	92
6.3.6	传统 8051 高电平上电复位参考电路 .....	93
6.4	外部晶振及外部时钟电路 .....	94
6.4.1	外部晶振输入电路 .....	94
6.4.2	外部时钟输入电路 (P1.6 不可用作普通 I/O) .....	94
6.5	时钟停振/省电模式与系统电源管理 .....	94
6.5.1	电源控制寄存器 (PCON) .....	94
6.6	掉电唤醒定时器 .....	96
6.6.1	掉电唤醒定时器计数寄存器 (WKTCL, WKTCH) .....	96
6.7	范例程序 .....	97
6.7.1	选择系统时钟源 .....	97
6.7.2	主时钟分频输出 .....	99
6.7.3	看门狗定时器应用 .....	101

6.7.4	软复位实现自定义下载 .....	103
6.7.5	低压检测 .....	105
6.7.6	省电模式 .....	107
6.7.7	使用 INT0/INT1/INT2/INT3/INT4 管脚中断唤醒省电模式 .....	109
6.7.8	使用 T0/T1/T2/T3/T4 管脚中断唤醒 MCU 省电模式 .....	112
6.7.9	使用 RxD/RxD2/RxD3/RxD4 管脚中断唤醒 MCU 省电模式 .....	117
6.7.10	使用 I2C 的 SDA 脚唤醒 MCU 省电模式 .....	120
6.7.11	使用掉电唤醒定时器唤醒省电模式 .....	123
6.7.12	LVD 中断唤醒省电模式, 建议配合使用掉电唤醒定时器 .....	125
6.7.13	比较器中断唤醒省电模式, 建议配合使用掉电唤醒定时器 .....	127
6.7.14	使用 LVD 功能检测工作电压 (电池电压) .....	130
<b>7</b>	<b>存储器 .....</b>	<b>135</b>
7.1	程序存储器 .....	135
7.2	数据存储器 .....	136
7.2.1	内部 RAM .....	136
7.2.2	程序状态寄存器 (PSW) .....	136
7.2.3	内部扩展 RAM, XRAM, XDATA .....	137
7.2.4	辅助寄存器 (AUXR) .....	138
7.2.5	外部扩展 RAM, XRAM, XDATA .....	139
7.2.6	总线速度控制寄存器 (BUS_SPEED) .....	139
7.2.7	8051 中可位寻址的数据存储器 .....	141
7.3	存储器中的特殊参数, 在 ISP 下载时可烧录进程序 FLASH .....	143
7.4	只读特殊功能寄存器中存储的唯一 ID 号和重要参数 (CHIPID) .....	145
7.4.1	CHIP 之全球唯一 ID 号解读 .....	146
7.4.2	CHIP 之内部参考信号源解读 .....	146
7.4.3	CHIP 之内部 32K 的 IRC 振荡频率解读 .....	147
7.4.4	CHIP 之高精度 IRC 参数解读 .....	148
7.4.5	CHIP 之测试时间参数解读 .....	149
7.4.6	CHIP 之芯片封装形式编号解读 .....	149
7.5	范例程序 .....	150
7.5.1	读取内部 1.19V 参考信号源值 (从 CHIPID 中读取) .....	150
7.5.2	读取内部 1.19V 参考信号源值 (从 Flash 程序存储器 (ROM) 中读取) .....	153
7.5.3	读取内部 1.19V 参考信号源值 (从 RAM 中读取) .....	156
7.5.4	读取全球唯一 ID 号 (从 CHIPID 中读取) .....	159
7.5.5	读取全球唯一 ID 号 (从 Flash 程序存储器 (ROM) 中读取) .....	163
7.5.6	读取全球唯一 ID 号 (从 RAM 中读取) .....	166
7.5.7	读取 32K 掉电唤醒定时器的频率 (从 CHIPID 中读取) .....	169
7.5.8	读取 32K 掉电唤醒定时器的频率 (从 Flash 程序存储器 (ROM) 中读取) .....	173
7.5.9	读取 32K 掉电唤醒定时器的频率 (从 RAM 中读取) .....	176
7.5.10	用户自定义内部 IRC 频率 (从 CHIPID 中读取) .....	179
7.5.11	用户自定义内部 IRC 频率 (从 Flash 程序存储器 (ROM) 中读取) .....	186
7.5.12	用户自定义内部 IRC 频率 (从 RAM 中读取) .....	191
<b>8</b>	<b>特殊功能寄存器 .....</b>	<b>194</b>
8.1	STC8A8K64D4-64Pin/48Pin 系列 .....	194

8.2	特殊功能寄存器列表 .....	196
<b>9</b>	<b>I/O 口 .....</b>	<b>206</b>
9.1	I/O 口相关寄存器 .....	206
9.1.1	端口数据寄存器 (Px) .....	207
9.1.2	端口模式配置寄存器 (PxM0, PxM1) .....	208
9.1.3	端口上拉电阻控制寄存器 (PxPU) .....	208
9.1.4	端口施密特触发控制寄存器 (PxNCS) .....	209
9.1.5	端口电平转换速度控制寄存器 (PxSR) .....	209
9.1.6	端口驱动电流控制寄存器 (PxDR) .....	209
9.1.7	端口数字信号输入使能控制寄存器 (PxIE) .....	210
9.2	配置 I/O 口 .....	211
9.3	I/O 的结构图 .....	213
9.3.1	准双向口 (弱上拉) .....	213
9.3.2	推挽输出 .....	213
9.3.3	高阻输入 .....	214
9.3.4	开漏输出 .....	214
9.3.5	新增 4.1K 上拉电阻 .....	215
9.3.6	如何设置 I/O 口对外输出速度 .....	215
9.3.7	如何设置 I/O 口电流驱动能力 .....	216
9.3.8	如何降低 I/O 口对外辐射 .....	216
9.4	范例程序 .....	217
9.4.1	端口模式设置 .....	217
9.4.2	双向口读写操作 .....	218
9.5	一种典型三极管控制电路 .....	221
9.6	典型发光二极管控制电路 .....	221
9.7	混合电压供电系统 3V/5V 器件 I/O 口互连 .....	222
9.8	如何让 I/O 口上电复位时为低电平 .....	223
9.9	利用 74HC595 驱动 8 个数码管 (串行扩展, 3 根线) 的线路图 .....	224
9.10	I/O 口直接驱动 LED 数码管应用线路图 .....	225
9.11	用 STC 系列 MCU 的 I/O 口直接驱动段码 LCD .....	226
<b>10</b>	<b>指令系统 .....</b>	<b>245</b>
<b>11</b>	<b>中断系统 .....</b>	<b>249</b>
11.1	STC8A8K64D4 系列中断源 .....	249
11.2	STC8A8K64D4 中断结构图 .....	252
11.3	STC8A8K64D4 系列中断列表 .....	253
11.4	中断相关寄存器 .....	256
11.4.1	中断使能寄存器 (中断允许位) .....	258
11.4.2	中断请求寄存器 (中断标志位) .....	263
11.4.3	中断优先级寄存器 .....	267
11.5	范例程序 .....	273
11.5.1	INT0 中断 (上升沿和下降沿), 可同时支持上升沿和下降沿 .....	273
11.5.2	INT0 中断 (下降沿) .....	275
11.5.3	INT1 中断 (上升沿和下降沿), 可同时支持上升沿和下降沿 .....	276
11.5.4	INT1 中断 (下降沿) .....	278

11.5.5	INT2 中断 (下降沿), 只支持下降沿中断.....	280
11.5.6	INT3 中断 (下降沿), 只支持下降沿中断.....	282
11.5.7	INT4 中断 (下降沿), 只支持下降沿中断.....	284
11.5.8	定时器 0 中断.....	286
11.5.9	定时器 1 中断.....	287
11.5.10	定时器 2 中断.....	289
11.5.11	定时器 3 中断.....	291
11.5.12	定时器 4 中断.....	294
11.5.13	UART1 中断.....	296
11.5.14	UART2 中断.....	298
11.5.15	UART3 中断.....	301
11.5.16	UART4 中断.....	303
11.5.17	LVD 中断.....	306
11.5.18	SPI 中断.....	308
11.5.19	比较器中断.....	310
11.5.20	I2C 中断.....	312
<b>12</b>	<b>普通 I/O 口均可中断, 不是传统外部中断.....</b>	<b>316</b>
12.1	I/O 口中断相关寄存器.....	316
12.1.1	端口中断使能寄存器 (PxINTE).....	317
12.1.2	端口中断标志寄存器 (PxINTF).....	317
12.1.3	端口中断模式配置寄存器 (PxIM0, PxIM1).....	318
12.1.4	端口中断优先级控制寄存器 (PINIPL, PINIPH).....	318
12.1.5	端口中断掉电唤醒使能寄存器 (PxWKUE).....	318
12.2	范例程序.....	320
12.2.1	P0 口下降沿中断.....	320
12.2.2	P1 口上升沿中断.....	324
12.2.3	P2 口低电平中断.....	327
12.2.4	P3 口高电平中断.....	331
<b>13</b>	<b>定时器/计数器.....</b>	<b>336</b>
13.1	定时器的相关寄存器.....	336
13.2	定时器 0/1.....	338
13.2.1	定时器 0/1 控制寄存器 (TCON).....	338
13.2.2	定时器 0/1 模式寄存器 (TMOD).....	338
13.2.3	定时器 0 模式 0 (16 位自动重载模式).....	339
13.2.4	定时器 0 模式 1 (16 位不可重载模式).....	340
13.2.5	定时器 0 模式 2 (8 位自动重载模式).....	341
13.2.6	定时器 0 模式 3 (不可屏蔽中断 16 位自动重载, 实时操作系统节拍器).....	341
13.2.7	定时器 1 模式 0 (16 位自动重载模式).....	342
13.2.8	定时器 1 模式 1 (16 位不可重载模式).....	343
13.2.9	定时器 1 模式 2 (8 位自动重载模式).....	344
13.2.10	定时器 0 计数寄存器 (TL0, TH0).....	344
13.2.11	定时器 1 计数寄存器 (TL1, TH1).....	344
13.2.12	辅助寄存器 1 (AUXR).....	344
13.2.13	中断与时钟输出控制寄存器 (INTCLKO).....	345

13.2.14	定时器 0 定时计算公式	345
13.2.15	定时器 1 定时计算公式	346
13.3	定时器 2 (24 位定时器, 8 位预分频+16 位定时)	347
13.3.1	辅助寄存器 1 (AUXR)	347
13.3.2	中断与时钟输出控制寄存器 (INTCLKO)	347
13.3.3	定时器 2 计数寄存器 (T2L, T2H)	347
13.3.4	定时器 2 的 8 位预分频寄存器 (TM2PS)	347
13.3.5	定时器 2 工作模式	348
13.3.6	定时器 2 计算公式	348
13.4	定时器 3/4 (24 位定时器, 8 位预分频+16 位定时)	349
13.4.1	定时器 4/3 控制寄存器 (T4T3M)	349
13.4.2	定时器 3 计数寄存器 (T3L, T3H)	349
13.4.3	定时器 4 计数寄存器 (T4L, T4H)	349
13.4.4	定时器 3 的 8 位预分频寄存器 (TM3PS)	350
13.4.5	定时器 4 的 8 位预分频寄存器 (TM4PS)	350
13.4.6	定时器 3 工作模式	350
13.4.7	定时器 4 工作模式	352
13.4.8	定时器 3 计算公式	352
13.4.9	定时器 4 计算公式	353
13.5	范例程序	354
13.5.1	定时器 0 (模式 0—16 位自动重载), 用作定时	354
13.5.2	定时器 0 (模式 1—16 位不自动重载), 用作定时	355
13.5.3	定时器 0 (模式 2—8 位自动重载), 用作定时	357
13.5.4	定时器 0 (模式 3—16 位自动重载不可屏蔽中断), 用作定时	359
13.5.5	定时器 0 (外部计数—扩展 T0 为外部下降沿中断)	361
13.5.6	定时器 0 (测量脉宽—INT0 高电平宽度)	363
13.5.7	定时器 0 (模式 0), 时钟分频输出	365
13.5.8	定时器 1 (模式 0—16 位自动重载), 用作定时	367
13.5.9	定时器 1 (模式 1—16 位不自动重载), 用作定时	369
13.5.10	定时器 1 (模式 2—8 位自动重载), 用作定时	371
13.5.11	定时器 1 (外部计数—扩展 T1 为外部下降沿中断)	372
13.5.12	定时器 1 (测量脉宽—INT1 高电平宽度)	374
13.5.13	定时器 1 (模式 0), 时钟分频输出	376
13.5.14	定时器 1 (模式 0) 做串口 1 波特率发生器	378
13.5.15	定时器 1 (模式 2) 做串口 1 波特率发生器	382
13.5.16	定时器 2 (16 位自动重载), 用作定时	386
13.5.17	定时器 2 (外部计数—扩展 T2 为外部下降沿中断)	388
13.5.18	定时器 2, 时钟分频输出	390
13.5.19	定时器 2 做串口 1 波特率发生器	392
13.5.20	定时器 2 做串口 2 波特率发生器	396
13.5.21	定时器 2 做串口 3 波特率发生器	400
13.5.22	定时器 2 做串口 4 波特率发生器	404
13.5.23	定时器 3 (16 位自动重载), 用作定时	408
13.5.24	定时器 3 (外部计数—扩展 T3 为外部下降沿中断)	410

13.5.25	定时器 3, 时钟分频输出 .....	412
13.5.26	定时器 3 做串口 3 波特率发生器 .....	414
13.5.27	定时器 4 (16 位自动重载), 用作定时 .....	418
13.5.28	定时器 4 (外部计数—扩展 T4 为外部下降沿中断) .....	421
13.5.29	定时器 4, 时钟分频输出 .....	423
13.5.30	定时器 4 做串口 4 波特率发生器 .....	425
<b>14</b>	<b>串口通信 .....</b>	<b>430</b>
14.1	串口功能脚切换 .....	430
14.2	串口相关寄存器 .....	430
14.3	串口 1 .....	432
14.3.1	串口 1 控制寄存器 (SCON) .....	432
14.3.2	串口 1 数据寄存器 (SBUF) .....	432
14.3.3	电源管理寄存器 (PCON) .....	433
14.3.4	辅助寄存器 1 (AUXR) .....	433
14.3.5	串口 1 模式 0, 模式 0 波特率计算公式 .....	433
14.3.6	串口 1 模式 1, 模式 1 波特率计算公式 .....	434
14.3.7	串口 1 模式 2, 模式 2 波特率计算公式 .....	437
14.3.8	串口 1 模式 3, 模式 3 波特率计算公式 .....	437
14.3.9	自动地址识别 .....	438
14.3.10	串口 1 从机地址控制寄存器 (SADDR, SADEN) .....	438
14.4	串口 2 .....	440
14.4.1	串口 2 控制寄存器 (S2CON) .....	440
14.4.2	串口 2 数据寄存器 (S2BUF) .....	440
14.4.3	串口 2 模式 0, 模式 0 波特率计算公式 .....	440
14.4.4	串口 2 模式 1, 模式 1 波特率计算公式 .....	441
14.5	串口 3 .....	443
14.5.1	串口 3 控制寄存器 (S3CON) .....	443
14.5.2	串口 3 数据寄存器 (S3BUF) .....	443
14.5.3	串口 3 模式 0, 模式 0 波特率计算公式 .....	443
14.5.4	串口 3 模式 1, 模式 1 波特率计算公式 .....	444
14.6	串口 4 .....	446
14.6.1	串口 4 控制寄存器 (S4CON) .....	446
14.6.2	串口 4 数据寄存器 (S4BUF) .....	446
14.6.3	串口 4 模式 0, 模式 0 波特率计算公式 .....	446
14.6.4	串口 4 模式 1, 模式 1 波特率计算公式 .....	447
14.7	串口注意事项 .....	449
14.8	范例程序 .....	450
14.8.1	串口 1 使用定时器 2 做波特率发生器 .....	450
14.8.2	串口 1 使用定时器 1 (模式 0) 做波特率发生器 .....	453
14.8.3	串口 1 使用定时器 1 (模式 2) 做波特率发生器 .....	457
14.8.4	串口 2 使用定时器 2 做波特率发生器 .....	461
14.8.5	串口 3 使用定时器 2 做波特率发生器 .....	465
14.8.6	串口 3 使用定时器 3 做波特率发生器 .....	469
14.8.7	串口 4 使用定时器 2 做波特率发生器 .....	473



14.8.8	串口 4 使用定时器 4 做波特率发生器 .....	477
14.8.9	串口多机通讯 .....	482
14.8.10	串口转 LIN 总线 .....	483
<b>15</b>	<b>比较器, 掉电检测, 内部 1.19V 参考信号源 .....</b>	<b>492</b>
15.1	比较器输出功能脚切换 .....	492
15.2	比较器内部结构图 .....	492
15.3	比较器相关的寄存器 .....	493
15.3.1	比较器控制寄存器 1 (CMPCR1) .....	493
15.3.2	比较器控制寄存器 2 (CMPCR2) .....	494
15.3.3	比较器扩展配置寄存器 (CMPEXCFG) .....	494
15.4	范例程序 .....	496
15.4.1	比较器的使用 (中断方式) .....	496
15.4.2	比较器的使用 (查询方式) .....	499
15.4.3	比较器作外部掉电检测 (掉电过程中应及时保存用户数据到 EEPROM 中) .....	502
15.4.4	比较器检测工作电压 (电池电压) .....	503
<b>16</b>	<b>IAP/EEPROM/DATA-FLASH .....</b>	<b>508</b>
16.1	EEPROM 操作时间 .....	508
16.2	EEPROM 相关的寄存器 .....	508
16.2.1	EEPROM 数据寄存器 (IAP_DATA) .....	508
16.2.2	EEPROM 地址寄存器 (IAP_ADDR) .....	509
16.2.3	EEPROM 命令寄存器 (IAP_CMD) .....	509
16.2.4	EEPROM 触发寄存器 (IAP_TRIG) .....	509
16.2.5	EEPROM 控制寄存器 (IAP_CONTR) .....	509
16.2.6	EEPROM 等待时间控制寄存器 (IAP_TPS) .....	510
16.3	EEPROM 大小及地址 .....	511
16.4	范例程序 .....	513
16.4.1	EEPROM 基本操作 .....	513
16.4.2	使用 MOVC 读取 EEPROM .....	516
16.4.3	使用串口送出 EEPROM 数据 .....	520
16.4.4	串口 1 读写 EEPROM-带 MOVC 读 .....	524
16.4.5	口令擦除写入-多扇区备份-串口 1 操作 .....	531
<b>17</b>	<b>ADC 模数转换, 内部 1.19V 参考信号源 .....</b>	<b>540</b>
17.1	ADC 相关的寄存器 .....	540
17.1.1	ADC 控制寄存器 (ADC_CONTR), PWM 触发 ADC 控制 .....	540
17.1.2	ADC 配置寄存器 (ADCCFG) .....	541
17.1.3	ADC 转换结果寄存器 (ADC_RES, ADC_RESL) .....	542
17.1.4	ADC 时序控制寄存器 (ADCTIM) .....	543
17.1.5	ADC 扩展配置寄存器 (ADCEXCFG) .....	544
17.2	ADC 相关计算公式 .....	545
17.2.1	ADC 速度计算公式 .....	545
17.2.2	ADC 转换结果计算公式 .....	545
17.2.3	反推 ADC 输入电压计算公式 .....	545
17.2.4	反推工作电压计算公式 .....	546
17.3	12 位 ADC 静态特性 .....	546

17.4	ADC 应用参考线路图.....	547
17.4.1	高精度 ADC 应用.....	547
17.4.2	ADC 一般应用 (对 ADC 精度要求不高的应用) .....	548
17.5	范例程序.....	549
17.5.1	ADC 基本操作 (查询方式) .....	549
17.5.2	ADC 基本操作 (中断方式) .....	551
17.5.3	格式化 ADC 转换结果.....	553
17.5.4	ADC 自动转换多次取平均值.....	556
17.5.5	利用 ADC 第 15 通道测量外部电压或电池电压.....	558
17.5.6	ADC 做电容感应触摸按键.....	562
17.5.7	ADC 作按键扫描应用线路图.....	574
17.5.8	检测负电压参考线路图 .....	576
17.5.9	常用加法电路在 ADC 中的应用.....	577
<b>18</b>	<b>PCA/CCP/PWM 应用 .....</b>	<b>578</b>
18.1	PCA 功能脚切换 .....	578
18.2	PCA 相关的寄存器 .....	578
18.2.1	PCA 控制寄存器 (CCON) .....	579
18.2.2	PCA 模式寄存器 (CMOD) .....	579
18.2.3	PCA 计数器寄存器 (CL, CH) .....	580
18.2.4	PCA 模块模式控制寄存器 (CCAPMn) .....	580
18.2.5	PCA 模块模式捕获值/比较值寄存器 (CCAPnL, CCAPnH) .....	580
18.2.6	PCA 模块 PWM 模式控制寄存器 (PCA_PWMn) .....	581
18.3	PCA 工作模式 .....	582
18.3.1	捕获模式.....	582
18.3.2	软件定时器模式.....	582
18.3.3	高速脉冲输出模式.....	583
18.3.4	PWM 脉宽调制模式及频率计算公式.....	583
18.4	利用 CCP/PCA/PWM 模块实现 8~16 位 DAC 的参考线路图.....	587
18.5	范例程序.....	588
18.5.1	PCA 输出 PWM (6/7/8/10 位) .....	588
18.5.2	PCA 捕获测量脉冲宽度 .....	591
18.5.3	PCA 实现 16 位软件定时 .....	595
18.5.4	PCA 输出高速脉冲 .....	598
18.5.5	PCA 扩展外部中断 .....	601
<b>19</b>	<b>精度可达 15 位的增强型 PWM.....</b>	<b>604</b>
19.1	增强型 PWM 输出功能脚切换.....	604
19.2	PWM 相关的寄存器 .....	606
19.2.1	增强型 PWM 全局配置寄存器 (PWMSET) .....	607
19.2.2	增强型 PWM 配置寄存器 (PWMCFG) .....	607
19.2.3	PWM 中断标志寄存器 (PWMIF) .....	608
19.2.4	PWM 异常检测控制寄存器 (PWMnFDCR) .....	608
19.2.5	PWM 计数器寄存器 (PWMCH, PWMCL) .....	609
19.2.6	PWM 时钟选择寄存器 (PWMCKS), 输出频率计算公式 .....	609
19.2.7	PWM 触发 ADC 计数器寄存器 (PWMTADC) .....	610

19.2.8	PWM 电平输出设置计数值寄存器 (PWMnT1, PWMnT2) .....	610
19.2.9	PWM 通道控制寄存器 (PWMnCR) .....	611
19.2.10	PWM 通道电平保持控制寄存器 (PWMnHLD) .....	612
19.3	范例程序 .....	613
19.3.1	输出任意周期和任意占空比的波形 .....	613
19.3.2	两路 PWM 实现互补对称带死区控制的波形 .....	615
19.3.3	PWM 实现渐变灯 (呼吸灯) .....	619
19.3.4	使用 PWM 触发 ADC 转换 .....	623
<b>20</b>	<b>同步串行外设接口 SPI.....</b>	<b>628</b>
20.1	SPI 功能脚切换 .....	628
20.2	SPI 相关的寄存器 .....	628
20.2.1	SPI 状态寄存器 (SPSTAT) .....	628
20.2.2	SPI 控制寄存器 (SPCTL), SPI 速度控制 .....	628
20.2.3	SPI 数据寄存器 (SPDAT) .....	629
20.3	SPI 通信方式 .....	630
20.3.1	单主单从 .....	630
20.3.2	互为主从 .....	630
20.3.3	单主多从 .....	631
20.4	配置 SPI .....	632
20.5	数据模式 .....	634
20.6	范例程序 .....	635
20.6.1	SPI 单主单从系统主机程序 (中断方式) .....	635
20.6.2	SPI 单主单从系统从机程序 (中断方式) .....	637
20.6.3	SPI 单主单从系统主机程序 (查询方式) .....	639
20.6.4	SPI 单主单从系统从机程序 (查询方式) .....	641
20.6.5	SPI 互为主从系统程序 (中断方式) .....	644
20.6.6	SPI 互为主从系统程序 (查询方式) .....	646
<b>21</b>	<b>I<sup>2</sup>C 总线.....</b>	<b>650</b>
21.1	I <sup>2</sup> C 功能脚切换 .....	650
21.2	I <sup>2</sup> C 相关的寄存器 .....	650
21.3	I <sup>2</sup> C 主机模式 .....	651
21.3.1	I <sup>2</sup> C 配置寄存器 (I2CCFG), 总线速度控制 .....	651
21.3.2	I <sup>2</sup> C 主机控制寄存器 (I2CMSCR) .....	652
21.3.3	I <sup>2</sup> C 主机辅助控制寄存器 (I2CMSAUX) .....	653
21.3.4	I <sup>2</sup> C 主机状态寄存器 (I2CMSST) .....	653
21.4	I <sup>2</sup> C 从机模式 .....	655
21.4.1	I <sup>2</sup> C 从机控制寄存器 (I2CSLCR) .....	655
21.4.2	I <sup>2</sup> C 从机状态寄存器 (I2CSLST) .....	655
21.4.3	I <sup>2</sup> C 从机地址寄存器 (I2CSLADR) .....	657
21.4.4	I <sup>2</sup> C 数据寄存器 (I2CTXD, I2CRXD) .....	658
21.5	范例程序 .....	659
21.5.1	I <sup>2</sup> C 主机模式访问 AT24C256 (中断方式) .....	659
21.5.2	I <sup>2</sup> C 主机模式访问 AT24C256 (查询方式) .....	665
21.5.3	I <sup>2</sup> C 主机模式访问 PCF8563 .....	671

21.5.4	I <sup>2</sup> C 从机模式 (中断方式)	676
21.5.5	I <sup>2</sup> C 从机模式 (查询方式)	681
21.5.6	测试 I <sup>2</sup> C 从机模式代码的主机代码	685
<b>22</b>	<b>LCM 接口</b>	<b>692</b>
22.1	LCM 接口功能脚切换	692
22.2	LCM 相关的寄存器	692
22.2.1	LCM 接口配置寄存器 (LCMIFCFG)	693
22.2.2	LCM 接口配置寄存器 2 (LCMIFCFG2)	693
22.2.3	LCM 接口控制寄存器 (LCMIFCR)	694
22.2.4	LCM 接口状态寄存器 (LCMIFSTA)	694
22.2.5	LCM 接口数据寄存器 (LCMIFDATL, LCMIFDATH)	694
22.3	LCD 接口时序图	695
22.3.1	I8080 模式	695
22.3.2	M6800 模式	696
<b>23</b>	<b>DMA</b>	<b>697</b>
23.1	DMA 相关的寄存器	697
23.2	存储器与存储器之间的数据读写 (M2M_DMA)	699
23.2.1	M2M_DMA 配置寄存器 (DMA_M2M_CFG)	699
23.2.2	M2M_DMA 控制寄存器 (DMA_M2M_CR)	700
23.2.3	M2M_DMA 状态寄存器 (DMA_M2M_STA)	700
23.2.4	M2M_DMA 传输总字节寄存器 (DMA_M2M_AMT)	700
23.2.5	M2M_DMA 传输完成字节寄存器 (DMA_M2M_DONE)	701
23.2.6	M2M_DMA 发送地址寄存器 (DMA_M2M_TXAx)	701
23.2.7	M2M_DMA 接收地址寄存器 (DMA_M2M_RXAx)	701
23.3	ADC 数据自动存储 (ADC_DMA)	702
23.3.1	ADC_DMA 配置寄存器 (DMA_ADC_CFG)	702
23.3.2	ADC_DMA 控制寄存器 (DMA_ADC_CR)	702
23.3.3	ADC_DMA 状态寄存器 (DMA_ADC_STA)	702
23.3.4	ADC_DMA 接收地址寄存器 (DMA_ADC_RXAx)	702
23.3.5	ADC_DMA 配置寄存器 2 (DMA_ADC_CFG2)	703
23.3.6	ADC_DMA 通道使能寄存器 (DMA_ADC_CHSWx)	703
23.3.7	ADC_DMA 的数据存储格式	704
23.4	SPI 与存储器之间的数据交换 (SPI_DMA)	706
23.4.1	SPI_DMA 配置寄存器 (DMA_SPI_CFG)	706
23.4.2	SPI_DMA 控制寄存器 (DMA_SPI_CR)	706
23.4.3	SPI_DMA 状态寄存器 (DMA_SPI_STA)	707
23.4.4	SPI_DMA 传输总字节寄存器 (DMA_SPI_AMT)	707
23.4.5	SPI_DMA 传输完成字节寄存器 (DMA_SPI_DONE)	707
23.4.6	SPI_DMA 发送地址寄存器 (DMA_SPI_TXAx)	707
23.4.7	SPI_DMA 接收地址寄存器 (DMA_SPI_RXAx)	707
23.4.8	SPI_DMA 配置寄存 2 器 (DMA_SPI_CFG2)	708
23.5	串口 1 与存储器之间的数据交换 (UR1T_DMA, UR1R_DMA)	709
23.5.1	UR1T_DMA 配置寄存器 (DMA_UR1T_CFG)	709
23.5.2	UR1T_DMA 控制寄存器 (DMA_UR1T_CR)	709

23.5.3	UR1T_DMA 状态寄存器 (DMA_UR1T_STA) .....	709
23.5.4	UR1T_DMA 传输总字节寄存器 (DMA_UR1T_AMT) .....	710
23.5.5	UR1T_DMA 传输完成字节寄存器 (DMA_UR1T_DONE) .....	710
23.5.6	UR1T_DMA 发送地址寄存器 (DMA_UR1T_TXAx) .....	710
23.5.7	UR1R_DMA 配置寄存器 (DMA_UR1R_CFG) .....	710
23.5.8	UR1R_DMA 控制寄存器 (DMA_UR1R_CR) .....	710
23.5.9	UR1R_DMA 状态寄存器 (DMA_UR1R_STA) .....	711
23.5.10	UR1R_DMA 传输总字节寄存器 (DMA_UR1R_AMT) .....	711
23.5.11	UR1R_DMA 传输完成字节寄存器 (DMA_UR1R_DONE) .....	711
23.5.12	UR1R_DMA 接收地址寄存器 (DMA_UR1T_RXAx) .....	711
23.6	串口 2 与存储器之间的数据交换 (UR2T_DMA, UR2R_DMA) .....	712
23.6.1	UR2T_DMA 配置寄存器 (DMA_UR2T_CFG) .....	712
23.6.2	UR2T_DMA 控制寄存器 (DMA_UR2T_CR) .....	712
23.6.3	UR2T_DMA 状态寄存器 (DMA_UR2T_STA) .....	712
23.6.4	UR2T_DMA 传输总字节寄存器 (DMA_UR2T_AMT) .....	713
23.6.5	UR2T_DMA 传输完成字节寄存器 (DMA_UR2T_DONE) .....	713
23.6.6	UR2T_DMA 发送地址寄存器 (DMA_UR2T_TXAx) .....	713
23.6.7	UR2R_DMA 配置寄存器 (DMA_UR2R_CFG) .....	713
23.6.8	UR2R_DMA 控制寄存器 (DMA_UR2R_CR) .....	713
23.6.9	UR2R_DMA 状态寄存器 (DMA_UR2R_STA) .....	714
23.6.10	UR2R_DMA 传输总字节寄存器 (DMA_UR2R_AMT) .....	714
23.6.11	UR2R_DMA 传输完成字节寄存器 (DMA_UR2R_DONE) .....	714
23.6.12	UR2R_DMA 接收地址寄存器 (DMA_UR2T_RXAx) .....	714
23.7	串口 3 与存储器之间的数据交换 (UR3T_DMA, UR3R_DMA) .....	715
23.7.1	UR3T_DMA 配置寄存器 (DMA_UR3T_CFG) .....	715
23.7.2	UR3T_DMA 控制寄存器 (DMA_UR3T_CR) .....	715
23.7.3	UR3T_DMA 状态寄存器 (DMA_UR3T_STA) .....	715
23.7.4	UR3T_DMA 传输总字节寄存器 (DMA_UR3T_AMT) .....	716
23.7.5	UR3T_DMA 传输完成字节寄存器 (DMA_UR3T_DONE) .....	716
23.7.6	UR3T_DMA 发送地址寄存器 (DMA_UR3T_TXAx) .....	716
23.7.7	UR3R_DMA 配置寄存器 (DMA_UR3R_CFG) .....	716
23.7.8	UR3R_DMA 控制寄存器 (DMA_UR3R_CR) .....	716
23.7.9	UR3R_DMA 状态寄存器 (DMA_UR3R_STA) .....	717
23.7.10	UR3R_DMA 传输总字节寄存器 (DMA_UR3R_AMT) .....	717
23.7.11	UR3R_DMA 传输完成字节寄存器 (DMA_UR3R_DONE) .....	717
23.7.12	UR3R_DMA 接收地址寄存器 (DMA_UR3T_RXAx) .....	717
23.8	串口 4 与存储器之间的数据交换 (UR4T_DMA, UR4R_DMA) .....	718
23.8.1	UR4T_DMA 配置寄存器 (DMA_UR4T_CFG) .....	718
23.8.2	UR4T_DMA 控制寄存器 (DMA_UR4T_CR) .....	718
23.8.3	UR4T_DMA 状态寄存器 (DMA_UR4T_STA) .....	718
23.8.4	UR4T_DMA 传输总字节寄存器 (DMA_UR4T_AMT) .....	719
23.8.5	UR4T_DMA 传输完成字节寄存器 (DMA_UR4T_DONE) .....	719
23.8.6	UR4T_DMA 发送地址寄存器 (DMA_UR4T_TXAx) .....	719
23.8.7	UR4R_DMA 配置寄存器 (DMA_UR4R_CFG) .....	719

23.8.8	UR4R_DMA 控制寄存器 (DMA_UR4R_CR) .....	719
23.8.9	UR4R_DMA 状态寄存器 (DMA_UR4R_STA) .....	720
23.8.10	UR4R_DMA 传输总字节寄存器 (DMA_UR4R_AMT) .....	720
23.8.11	UR4R_DMA 传输完成字节寄存器 (DMA_UR4R_DONE) .....	720
23.8.12	UR4R_DMA 接收地址寄存器 (DMA_UR4T_RXAx) .....	720
23.9	LCM 与存储器之间的数据读写 (LCM_DMA) .....	721
23.9.1	LCM_DMA 配置寄存器 (DMA_LCM_CFG) .....	721
23.9.2	LCM_DMA 控制寄存器 (DMA_LCM_CR) .....	721
23.9.3	LCM_DMA 状态寄存器 (DMA_LCM_STA) .....	722
23.9.4	LCM_DMA 传输总字节寄存器 (DMA_LCM_AMT) .....	722
23.9.5	LCM_DMA 传输完成字节寄存器 (DMA_LCM_DONE) .....	722
23.9.6	LCM_DMA 发送地址寄存器 (DMA_LCM_TXAx) .....	722
23.9.7	LCM_DMA 接收地址寄存器 (DMA_LCM_RXAx) .....	722
23.10	范例程序 .....	723
23.10.1	串口 1 中断模式与电脑收发测试 - DMA 接收超时中断 .....	723
23.10.2	串口 1 中断模式与电脑收发测试 - DMA 数据校验 .....	728
<b>24</b>	<b>增强型双数据指针 .....</b>	<b>735</b>
24.1	相关的特殊功能寄存器 .....	735
24.1.1	第 1 组 16 位数据指针寄存器 (DPTR0) .....	735
24.1.2	第 2 组 16 位数据指针寄存器 (DPTR1) .....	735
24.1.3	数据指针控制寄存器 (DPS) .....	735
24.1.4	数据指针控制寄存器 (TA) .....	736
24.2	范例程序 .....	738
24.2.1	示例代码 1 .....	738
24.2.2	示例代码 2 .....	739
<b>25</b>	<b>MDU16 硬件 16 位乘除法器 .....</b>	<b>741</b>
25.1	相关的特殊功能寄存器 .....	741
25.1.1	操作数 1 数据寄存器 (MD0~MD3) .....	741
25.1.2	操作数 2 数据寄存器 (MD4~MD5) .....	741
25.1.3	MDU 模式控制寄存器 (ARCON), 运算所需时钟数 .....	742
25.1.4	MDU 操作控制寄存器 (OPCON) .....	742
25.2	关于 MDU16 的网友应用杂谈 (提供思路, 仅供参考) .....	744
25.3	范例程序 .....	746
<b>附录 A</b>	<b>编译器 (汇编器) / 仿真器 / 头文件使用指南 .....</b>	<b>748</b>
<b>附录 B</b>	<b>STC-ISP 下载软件高级应用 .....</b>	<b>757</b>
B.1	发布项目程序 .....	757
B.2	程序加密后传输 (防烧录时串口分析出程序) .....	761
B.3	发布项目程序+程序加密后传输结合使用 .....	765
B.4	用户自定义下载 (实现不停电下载) .....	766
<b>附录 C</b>	<b>如何测试 I/O 口 .....</b>	<b>770</b>
<b>附录 D</b>	<b>如何让传统的 8051 单片机学习板可仿真 .....</b>	<b>771</b>
<b>附录 E</b>	<b>STC-USB 驱动程序安装说明 .....</b>	<b>773</b>
<b>附录 F</b>	<b>USB 下载步骤演示 .....</b>	<b>836</b>
<b>附录 G</b>	<b>RS485 自动控制或 I/O 口控制线路图 .....</b>	<b>840</b>

<b>附录 H</b>	<b>STC 工具使用说明书</b> .....	<b>841</b>
H.1	概述.....	841
H.2	系统可编程 (ISP) 流程说明.....	841
H.3	USB 型联机/脱机下载工具 U8W/U8W-Mini.....	842
H.3.1	安装 U8W/U8W-Mini 驱动程序.....	844
H.3.2	U8W 的功能介绍.....	847
H.3.3	U8W 的在线联机下载使用说明.....	848
H.3.4	U8W 的脱机下载使用说明.....	851
H.3.5	U8W-Mini 的功能介绍.....	859
H.3.6	U8W-Mini 的在线联机下载使用说明.....	860
H.3.7	U8W-Mini 的脱机下载使用说明.....	861
H.3.8	制作/更新 U8W/U8W-Mini.....	867
H.3.9	U8W/U8W-Mini 设置直通模式 (可用于仿真).....	869
H.3.10	U8W/U8W-Mini 的参考电路.....	869
H.4	STC 通用 USB 转串口工具.....	871
H.4.1	STC 通用 USB 转串口工具外观图.....	871
H.4.2	STC 通用 USB 转串口工具布局图.....	872
H.4.3	STC 通用 USB 转串口工具驱动安装.....	873
H.4.4	使用 STC 通用 USB 转串口工具下载程序到 MCU.....	874
H.4.5	使用 STC 通用 USB 转串口工具仿真用户代码.....	876
H.5	应用线路图.....	883
H.5.1	U8W 工具应用参考线路图.....	883
H.5.2	STC 通用 USB 转串口工具应用参考线路图.....	883
<b>附录 I</b>	<b>STC 仿真使用说明书</b> .....	<b>885</b>
I.1	概述.....	885
I.2	安装 Keil 软件.....	886
I.3	安装仿真驱动.....	887
I.4	串口直接仿真.....	890
I.4.1	制作串口仿真芯片.....	890
I.4.2	在 Keil 软件中进行串口仿真设置.....	893
I.4.3	在 Keil 软件中使用串口进行仿真.....	895
I.5	USB 直接仿真 (目前只有 STC8H8K64U-B 版本芯片支持).....	897
I.5.1	制作 USB 仿真芯片.....	897
I.5.2	在 Keil 软件中进行 USB 仿真设置.....	901
I.5.3	在 Keil 软件中使用 USB 进行仿真.....	903
<b>附录 J</b>	<b>U8W 下载工具中 RS485 部分线路图</b> .....	<b>905</b>
<b>附录 K</b>	<b>运行用户程序时收到用户命令后自动启动 ISP 下载(不停电)</b> .....	<b>906</b>
<b>附录 L</b>	<b>使用 STC 的 IAP 系列单片机开发自己的 ISP 程序</b> .....	<b>908</b>
<b>附录 M</b>	<b>用户程序复位到系统区进行 ISP 下载的方法 (不停电)</b> .....	<b>920</b>
<b>附录 N</b>	<b>使用第三方 MCU 对 STC8A8K64D4 系列单片机进行 ISP 下载范例程序</b> .....	<b>926</b>
<b>附录 O</b>	<b>使用第三方应用程序调用 STC 发布项目程序对单片机进行 ISP 下载</b> .....	<b>934</b>
<b>附录 P</b>	<b>在 Keil 中建立多文件项目的方法</b> .....	<b>938</b>
<b>附录 Q</b>	<b>关于中断号大于 31 在 Keil 中编译出错的处理</b> .....	<b>942</b>
<b>附录 R</b>	<b>电气特性</b> .....	<b>952</b>

R.1	绝对最大额定值 .....	952
R.2	直流特性 (3.3V) .....	953
R.3	直流特性 (5.0V) .....	955
R.4	内部 IRC 温漂特性 (参考温度 25℃) .....	956
R.5	低压复位门槛电压 (测试温度 25℃) .....	956
附录 S	应用注意事项.....	<b>957</b>
S.1	STC8A8K64D4-64Pin/48Pin 系列.....	957
附录 T	触摸按键的 PCB 设计指导.....	<b>958</b>
附录 U	QFN/DFN 封装元器件焊接方法.....	<b>960</b>
附录 V	STC8A8K64D4 系列单片机取代 STC8A8K64S4A12 系列的注意事项.....	<b>963</b>
附录 W	更新记录.....	<b>965</b>

STC MCU



# 1 概述

STC8A8K64D4 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，STC8A8K64D4 系列单片机比传统的 8051 约快 12 倍（速度快 11.2~13.2 倍），依次按顺序执行全部的 111 条指令，STC8A8K64D4 系列单片机仅需 147 个时钟，而传统 8051 则需要 1944 个时钟。STC8A8K64D4 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( $\pm 0.3\%$ ，常温下 $+25^{\circ}\text{C}$ )， $-1.38\% \sim +1.42\%$  温飘( $-40^{\circ}\text{C} \sim +85^{\circ}\text{C}$ )， $-0.88\% \sim +1.05\%$  温飘( $-20^{\circ}\text{C} \sim +65^{\circ}\text{C}$ )。ISP 编程时 4MHz~45MHz 宽范围可设置（注意：温度范围为 $-40^{\circ}\text{C} \sim +85^{\circ}\text{C}$ 时，最高频率须控制在 45MHz 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  $0.6\mu\text{A}@V_{\text{CC}}=5.0\text{V}$ ， $0.4\mu\text{A}@V_{\text{CC}}=3.3\text{V}$ 。

掉电模式可以使用 INT0(P3.2)、INT1(P3.3)、INT2(P3.6)、INT3(P3.7)、INT4(P3.0)、T0(P3.4)、T1(P3.5)、T2(P1.2)、T3(P0.4)、T4(P0.6)、RXD2(P1.0/P4.0)、RXD3(P0.0/P5.0)、RXD4(P0.2/P5.0)、CCP0(P1.7/P2.3/P7.0/P3.3)、CCP1(P1.6/P2.4/P7.1/P3.2)、CCP2(P1.5/P2.5/P7.2/P3.1)、CCP3(P1.4/P2.6/P7.3/P3.0)、I2C\_SDA(P1.4/P2.4/P3.3)、SPI\_SS(P1.2/P2.2/P3.5)以及所有端口的 I/O 中断、比较器中断、低压检测中断、掉电唤醒定时器唤醒。

MCU 提供了丰富的数字外设（串口、定时器、PCA、增强型 PWM 以及 I<sup>2</sup>C、SPI）接口与模拟外设（速度高达 800K 即每秒 80 万次采样的 12 位\*15 路超高速 ADC、比较器），可满足广大用户的设计需求。

STC8A8K64D4 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	I/O	UART	定时器	ADC	增强型 PWM	PCA	CMP	SPI	I2C	MDU16	I/O 中断	LCM	DMA
STC8A8K64D4 系列-64Pin/48Pin	59	4	5	15 <sub>ch</sub> *12 <sub>b</sub>	●	●	●	●	●	●	●	●	●

## 2 特性、价格及管脚

### 2.1 STC8A8K64D4-LQFP64/48/44, PDIP40 系列

#### 2.1.1 特性及价格(有 16 位硬件乘法器 MDU16, 准 16 位单片机)

##### 选型价格(不需要外部晶振、不需要外部复位)

单片机型号	工作电压(V)	Flash 程序存储器 10 万次 字节	xdata 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	MDU16 硬件 16 位乘法器	DMA 控制 ADC、串口、SPI、LCM 等	8080/6800 LCM 彩屏接口驱动(8 位和 16 位)	I <sup>2</sup> C	SPI	所有的 I/O 口均支持中断并可掉电唤醒	定时器/计数器(T0-T1)外部管脚也可掉电唤醒	15 位增强型 PWM 满足舞台灯光要求	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(可选复位门檻电压)	内部高精度时钟(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	部分封装及含税价格				2021 年新品供货信息	
																														LQFP64	LQFP48	LQFP44	PDIP40		
STC8A8K16D4	1.9-5.5	16K	8K	2	48K	59	4	有	有	有	有	有	有	5	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	是	√	√	√	√	现货
STC8A8K32D4	1.9-5.5	32K	8K	2	32K	59	4	有	有	有	有	有	有	5	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	是	√	√	√	√	
STC8A8K48D4	1.9-5.5	48K	8K	2	16K	59	4	有	有	有	有	有	有	5	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	是	√	√	√	√	
STC8A8K60D4	1.9-5.5	60K	8K	2	4K	59	4	有	有	有	有	有	有	5	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	是	√	√	√	√	
STC8A8K64D4	1.9-5.5	64K	8K	2	IAP	59	4	有	有	有	有	有	有	5	8	4	有	12 位	有	有	有	4 级	有	是	有	是	是	是	是	是	√	√	√	√	

##### 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
- ✓ 43 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°C~85°C (超温度范围应用请参考电器特性章节说明)

##### Flash 存储器

- ✓ 最大 64K 字节 FLASH 程序存储器 (ROM), 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

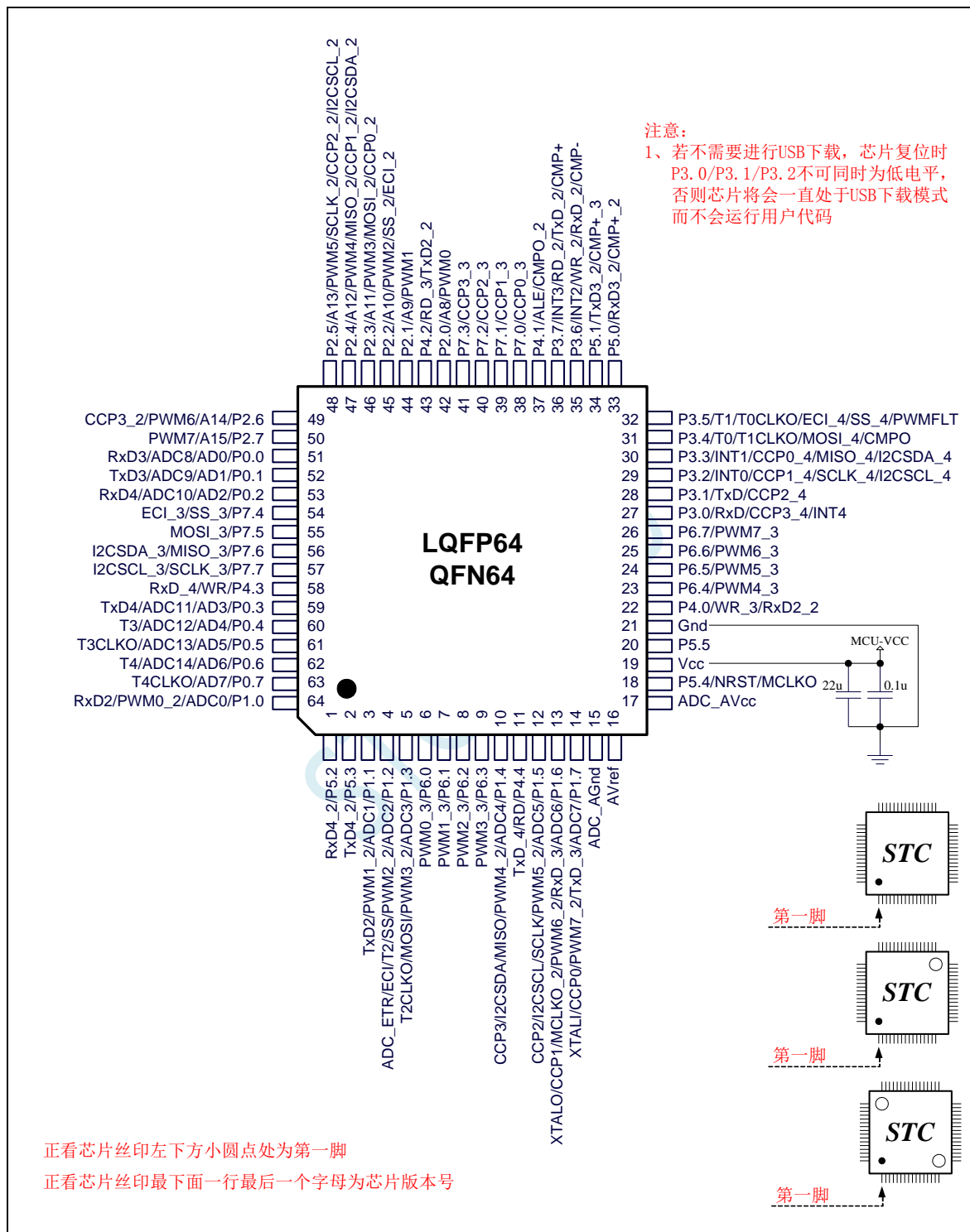
- ✓ 128 字节内部直接访问 RAM (DATA)
- ✓ 128 字节内部间接访问 RAM (IDATA)

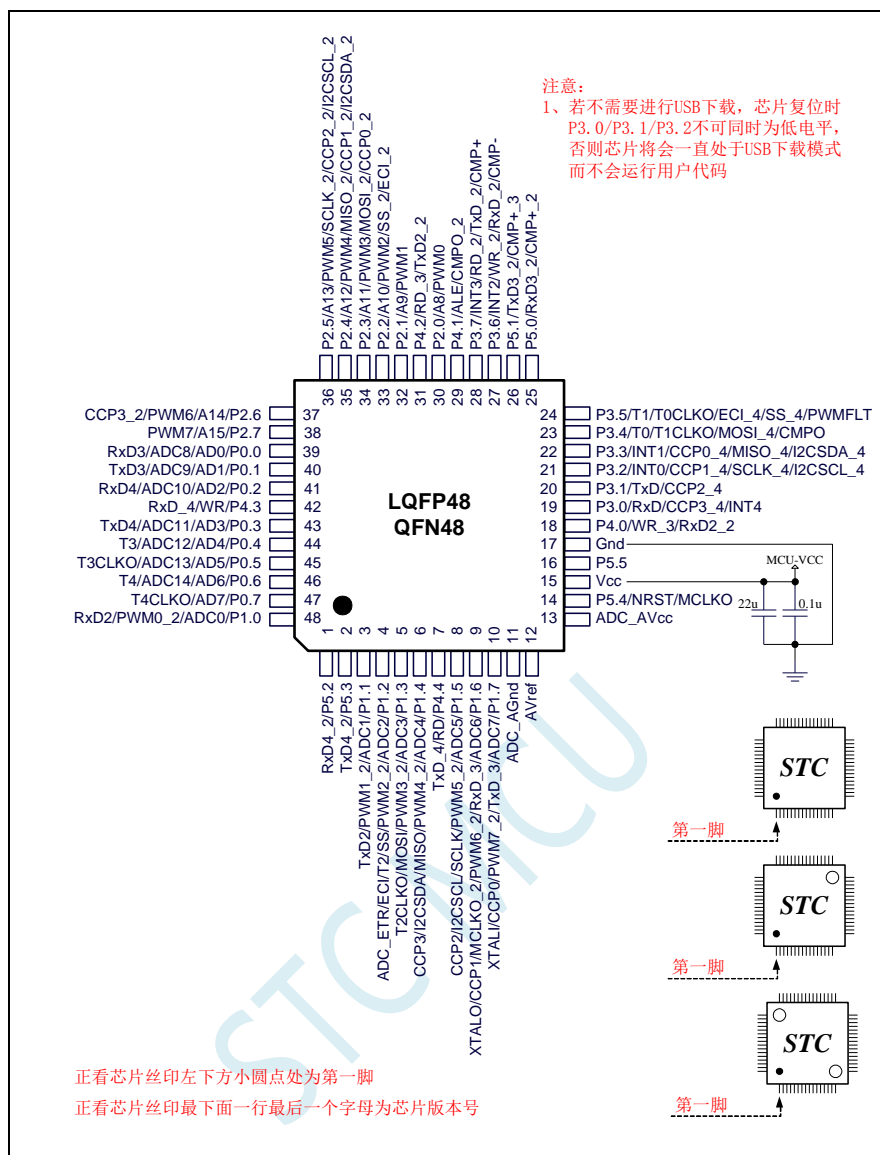
- ✓ 8192 字节内部扩展 RAM (内部 XDATA)
- 时钟控制
  - ✓ 内部高精度 IRC (4MHz~45MHz, ISP 编程时可进行上下调整, 还可以用户软件分频到较低的频率工作, 如 100KHz)
    - ⊕ 误差±0.3% (常温下 25℃)
    - ⊕ -1.38%~+1.42%温漂 (全温度范围, -40℃~85℃)
    - ⊕ -0.88%~+1.05%温漂 (温度范围, -20℃~65℃)
  - ✓ 内部 32KHz 低速 IRC (误差较大)
  - ✓ 外部晶振 (4MHz~45MHz) 和外部时钟
- 复位
  - ✓ 硬件复位
    - ⊕ 上电复位, 实测电压值为 1.69V~1.82V。 (**在芯片未使能低压复位功能时有效**)  
上电复位电压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 向下掉到上电复位的下限门槛电压时, 芯片处于复位状态; 当电压从 0V 上升到上电复位的上限门槛电压时, 芯片解除复位状态。
    - ⊕ 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 43 个中断源: INT0 (支持上升沿和下降沿中断)、INT1 (支持上升沿和下降沿中断)、INT2 (只支持下降沿中断)、INT3 (只支持下降沿中断)、INT4 (只支持下降沿中断)、定时器 0、定时器 1、定时器 2、定时器 3、定时器 4、串口 1、串口 2、串口 3、串口 4、ADC 模数转换、LVD 低压检测、SPI、I2C、比较器、PCA/CCP/PWM、增强型 PWM、增强型 PWM 异常检测、所有的 I/O 中断 (8 组)、LCD 驱动中断、串口 1 的 DMA 接收和发送中断、串口 2 的 DMA 接收和发送中断、串口 3 的 DMA 接收和发送中断、串口 4 的 DMA 接收和发送中断、SPI 的 DMA 中断、ADC 的 DMA 中断、LCD 驱动的 DMA 中断以及存储器到存储器的 DMA 中断。
  - ✓ 提供 4 级中断优先级
  - ✓ 时钟停振模式下可以唤醒的中断: INT0(P3.2)、INT1(P3.3)、INT2(P3.6)、INT3(P3.7)、INT4(P3.0)、T0(P3.4)、T1(P3.5)、T2(P1.2)、T3(P0.4)、T4(P0.6)、RXD(P3.0/P3.6/P1.6/P4.3)、RXD2(P1.0/P4.0)、RXD3(P0.0/P5.0)、RXD4(P0.2/P5.0)、CCP0(P1.7/P2.3/P7.0/P3.3)、CCP1(P1.6/P2.4/P7.1/P3.2)、CCP2(P1.5/P2.5/P7.2/P3.1)、CCP3(P1.4/P2.6/P7.3/P3.0)、I2C\_SDA(P1.4/P2.4/P3.3)、SPL\_SS(P1.2/P2.2/P3.5)以及所有端口的 I/O 中断、比较器中断、低压检测中断、掉电唤醒定时器唤醒
- 数字外设
  - ✓ 5 个 16 位定时器: 定时器 0、定时器 1、定时器 2、定时器 3、定时器 4, 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
  - ✓ 4 个高速串口: 串口 1、串口 2、串口 3、串口 4, 波特率时钟源最快可为 FOSC/4
  - ✓ 4 组 16 位 PCA 模块: CCP0、CCP1、CCP2、CCP3, 可用于捕获、高速脉冲输出, 及 6/7/8/10 位的 PWM

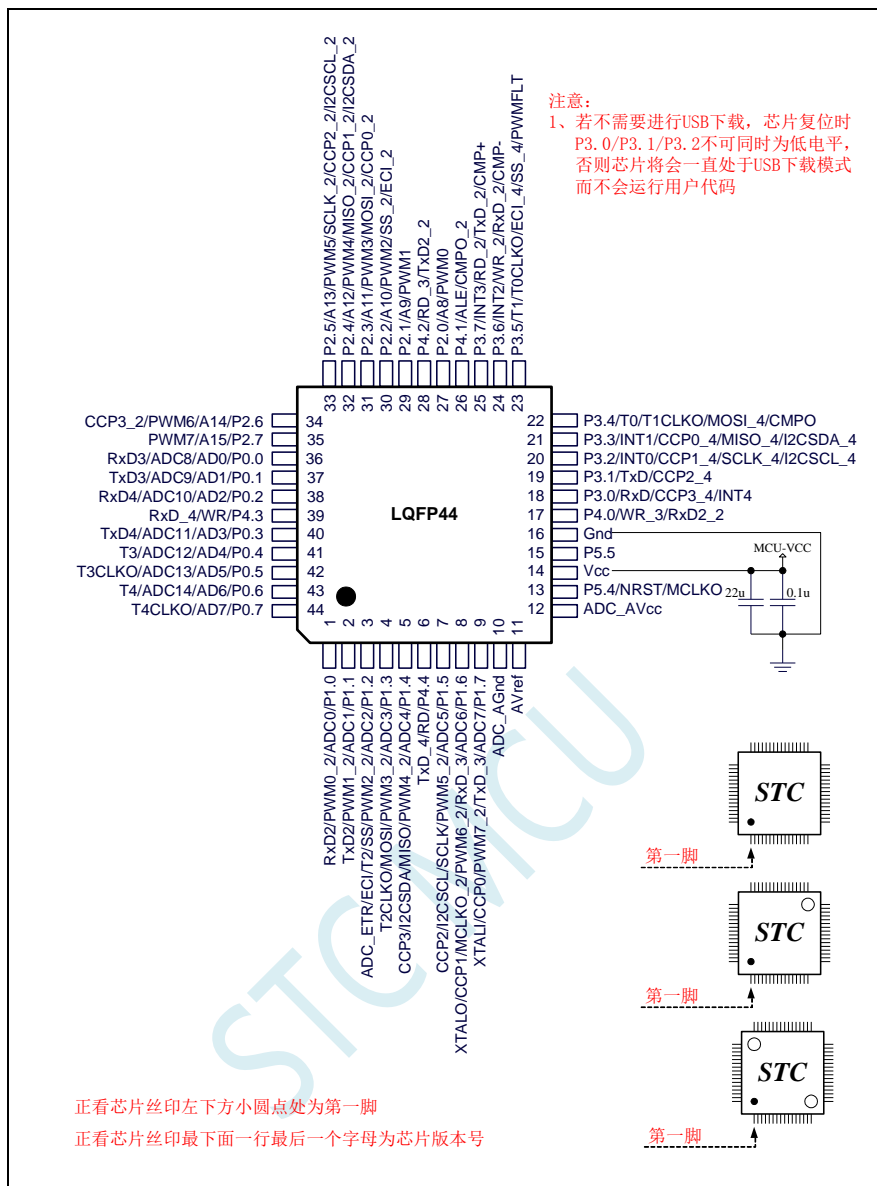
## 输出

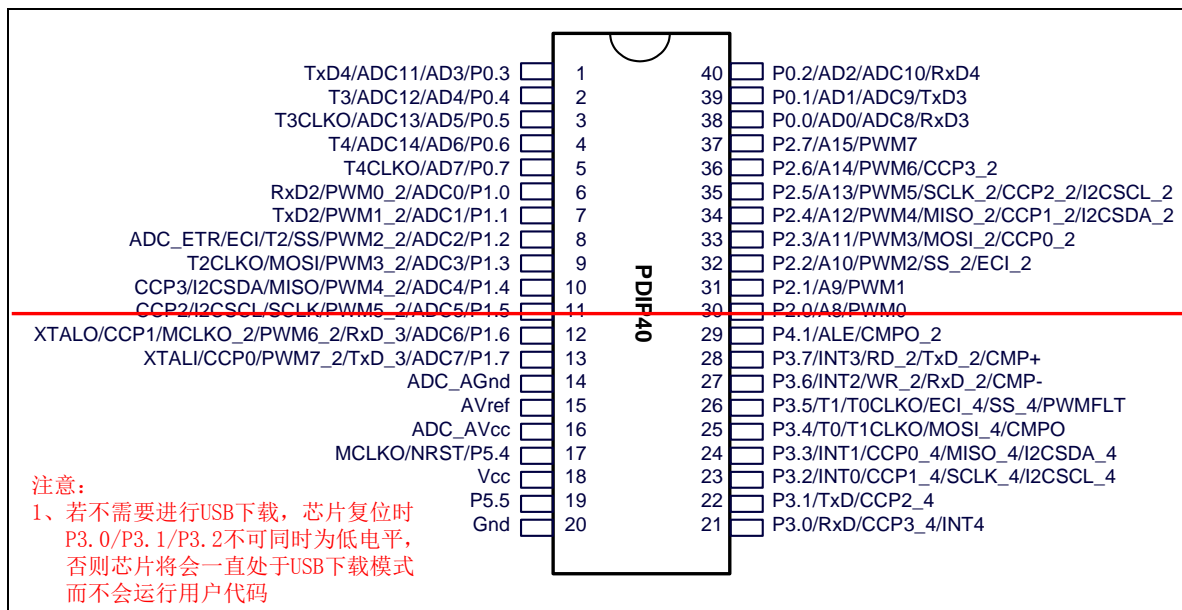
- ✓ 8 组 15 位增强型 PWM, 可实现带死区的控制信号, 并支持外部异常检测功能, 另外还有 4 组传统的 PCA/CCP/PWM 可作 PWM
  - ✓ SPI: 支持主机模式和从机模式以及主机/从机自动切换
  - ✓ I<sup>2</sup>C: 支持主机模式和从机模式
  - ✓ **MDU16:** 硬件 16 位乘法器 (支持 32 位除以 16 位、16 位除以 16 位、16 位乘 16 位、数据移位以及数据规格化等运算)
  - ✓ **I/O 口中断:** 所有的 I/O 均支持中断, 每组 I/O 中断有独立的中断入口地址, 所有的 I/O 中断可支持 4 种中断模式: 高电平中断、低电平中断、上升沿中断、下降沿中断  
(本系列的 I/O 口中断可以进行掉电唤醒, 且有 4 级中断优先级)
  - ✓ **LCD 驱动模块:** 支持 8080 和 6800 两种接口以及 8 位和 16 位数据宽度
  - ✓ **DMA:** 支持 SPI 移位接收数据到存储器、SPI 移位发送存储器的数据、串口 1/2/3/4 接收数据到的存储器、串口 1/2/3/4 发送存储器的数据、ADC 自动采样数据到存储器 (同时计算平均值)、LCD 驱动发送存储器的数据、以及存储器到存储器的数据复制
  - ✓ **硬件数字 ID:** 支持 32 字节
- **模拟外设**
- ✓ 超高速 ADC, 支持 **12 位高精度** 15 通道 (通道 0~通道 14) 的模数转换, **速度最快能达到 800K (每秒进行 80 万次 ADC 转换)**
  - ✓ ADC 的通道 15 用于测试内部 1.19V 参考信号源 (芯片在出厂时, 内部参考信号源已调整为 1.19V)
  - ✓ 比较器, 一组比较器 (比较器的正端可选择 CMP+ 端口、CMP+\_2、CMP+\_3 和所有的 ADC 输入端口, 比较器的负端可选择 CMP- 端口和内部 1.19V 的参考源, 所以比较器可当作多路比较器进行分时复用)
  - ✓ DAC: 8 组增强型 PWM 定时器可当 8 路 DAC 使用、4 路 PCA 可当 4 路 DAC 使用
- **GPIO**
- ✓ 最多可达 59 个 GPIO: P0.0~P0.7、P1.0~P1.7、P2.0~P2.7、P3.0~P3.7、P4.0~P4.4、P5.0~P5.5、P6.0~P6.7、P7.0~P7.7
  - ✓ 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**
- ✓ LQFP64、LQFP48、LQFP44

## 2.1.2 管脚图, 最小系统











## 2.1.3 管脚说明

编号				名称	类型	说明
LQFP64	LQFP48	LQFP44	PDIP40			
1	1			P5.2	I/O	标准 IO 口
				RxD4_2	I	串口 4 的接收脚
2	2			P5.3	I/O	标准 IO 口
				TxD4_2	O	串口 4 的发送脚
3	3	2	7	P1.1	I/O	标准 IO 口
				ADC1	I	ADC 模拟输入通道 1
				PWM1_2	O	增强 PWM 通道 1 输出脚
				TxD2	O	串口 2 的发送脚
4	4	3	8	P1.2	I/O	标准 IO 口
				ADC2	I	ADC 模拟输入通道 2
				PWM2_2	O	增强 PWM 通道 2 输出脚
				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
				ADC_ETR	I	ADC 外部触发输入
5	5	4	9	P1.3	I/O	标准 IO 口
				ADC3	I	ADC 模拟输入通道 3
				PWM3_2	O	增强 PWM 通道 3 输出脚
				MOSI	I/O	SPI 主机输出从机输入
				T2CLKO	O	定时器 2 时钟分频输出
6				P6.0	I/O	标准 IO 口
				PWM0_3	O	增强 PWM 通道 0 输出脚
7				P6.1	I/O	标准 IO 口
				PWM1_3	O	增强 PWM 通道 1 输出脚
8				P6.2	I/O	标准 IO 口
				PWM2_3	O	增强 PWM 通道 2 输出脚
9				P6.3	I/O	标准 IO 口
				PWM3_3	O	增强 PWM 通道 3 输出脚

编号				名称	类型	说明
LQFP64	LQFP48	LQFP44	PDIP40			
10	6	5	10	P1.4	I/O	标准 IO 口
				ADC4	I	ADC 模拟输入通道 4
				PWM4_2	O	增强 PWM 通道 4 输出脚
				MISO	I/O	SPI 主机输入从机输出
				SDA	I/O	I2C 接口的数据线
				CCP3	I/O	PCA 的捕获输入和脉冲输出
11	7	6		P4.4	I/O	标准 IO 口
				RD	O	外部总线的读信号线
				TxD_4	O	串口 1 的发送脚
12	8	7	11	P1.5	I/O	标准 IO 口
				ADC5	I	ADC 模拟输入通道 5
				PWM5_2	O	增强 PWM 通道 5 输出脚
				SCLK	I/O	SPI 的时钟脚
				SCL	I/O	I2C 的时钟线
				CCP2	I/O	PCA 的捕获输入和脉冲输出
13	9	8	12	P1.6	I/O	标准 IO 口
				ADC6	I	ADC 模拟输入通道 6
				RxD_3	I	串口 1 的接收脚
				PWM6_2	O	增强 PWM 通道 6 输出脚
				MCLKO_2	O	主时钟分频输出
				CCP1	I/O	PCA 的捕获输入和脉冲输出
				XTALO	O	外部晶振的输出脚
14	10	9	13	P1.7	I/O	标准 IO 口
				ADC7	I	ADC 模拟输入通道 7
				TxD_3	O	串口 1 的发送脚
				PWM7_2	O	增强 PWM 通道 7 输出脚
				CCP0	I/O	PCA 的捕获输入和脉冲输出
				XTALI	I	外部晶振/外部时钟的输入脚
15	11	10	14	ADC_AGnd	GND	ADC 地线
16	12	11	15	AVref	I	ADC 的参考电压脚
17	13	12	16	ADC_AVcc	VCC	ADC 电源脚
18	14	13	17	P5.4	I/O	标准 IO 口
				NRST	I	复位引脚 (低电平复位)
				MCLKO	O	主时钟分频输出
19	15	14	18	Vcc	VCC	电源脚
20	16	15	19	P5.5	I/O	标准 IO 口
21	17	16	20	Gnd	GND	地线

编号				名称	类型	说明
LQFP64	LQFP48	LQFP44	PDIP40			
22	18	17		P4.0	I/O	标准 IO 口
				WR_3	O	外部总线的写信号线
				RxD2_2	I	串口 2 的接收脚
23				P6.4	I/O	标准 IO 口
				PWM4_3	O	增强 PWM 通道 4 输出脚
24				P6.5	I/O	标准 IO 口
				PWM5_3	O	增强 PWM 通道 5 输出脚
25				P6.6	I/O	标准 IO 口
				PWM6_3	O	增强 PWM 通道 6 输出脚
26				P6.7	I/O	标准 IO 口
				PWM7_3	O	增强 PWM 通道 7 输出脚
27	19	18	21	P3.0	I/O	标准 IO 口
				RxD	I	串口 1 的接收脚
				CCP3_4	I/O	PCA 的捕获输入和脉冲输出
				INT4	I	外部中断 4
28	20	19	22	P3.1	I/O	标准 IO 口
				TxD	O	串口 1 的发送脚
				CCP2_4	I/O	PCA 的捕获输入和脉冲输出
29	21	20	23	P3.2	I/O	标准 IO 口
				INT0	I	外部中断 0
				CCP1_4	I/O	PCA 的捕获输入和脉冲输出
				SCLK_4	I/O	SPI 的时钟脚
				SCL_4	I/O	I2C 的时钟线
30	22	21	24	P3.3	I/O	标准 IO 口
				INT1	I	外部中断 1
				CCP0_4	I/O	PCA 的捕获输入和脉冲输出
				MISO_4	I/O	SPI 主机输入从机输出
				SDA_4	I/O	I2C 接口的数据线
31	23	22	25	P3.4	I/O	标准 IO 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
				MOSI_4	I/O	SPI 主机输出从机输入
				CMPO	O	比较器输出

编号				名称	类型	说明
LQFP64	LQFP48	LQFP44	PDIP40			
32	24	23	26	P3.5	I/O	标准 IO 口
				T1	I	定时器 1 外部时钟输入
				TOCLKO	O	定时器 0 时钟分频输出
				ECL_4	I	PCA 的外部脉冲输入
				SS_4	I	SPI 的从机选择脚 (主机为输出)
				PWMFLT	I	增强 PWM 的外部异常检测脚
33	25			P5.0	I/O	标准 IO 口
				RxD3_2	I	串口 3 的接收脚
				CMP+_2	I	比较器正极输入
34	26			P5.1	I/O	标准 IO 口
				TxD3_2	O	串口 3 的发送脚
				CMP+_3	I	比较器正极输入
35	27	24	27	P3.6	I/O	标准 IO 口
				INT2	I	外部中断 2
				WR_2	O	外部总线的写信号线
				RxD_2	I	串口 1 的接收脚
				CMP-	I	比较器负极输入
36	28	25	28	P3.7	I/O	标准 IO 口
				INT3	I	外部中断 3
				RD_2	O	外部总线的读信号线
				TxD_2	O	串口 1 的发送脚
				CMP+	I	比较器正极输入
37	29	26	29	P4.1	I/O	标准 IO 口
				ALE	O	地址锁存信号
				CMPO_2	O	比较器输出
38				P7.0	I/O	标准 IO 口
				CCP0_3	I/O	PCA 的捕获输入和脉冲输出
39				P7.1	I/O	标准 IO 口
				CCP1_3	I/O	PCA 的捕获输入和脉冲输出
40				P7.2	I/O	标准 IO 口
				CCP2_3	I/O	PCA 的捕获输入和脉冲输出
41				P7.3	I/O	标准 IO 口
				CCP3_3	I/O	PCA 的捕获输入和脉冲输出
42	30	27	30	P2.0	I/O	标准 IO 口
				A8	I	地址总线
				PWM0	O	增强 PWM 通道 0 输出脚
43	31	28		P4.2	I/O	标准 IO 口
				RD_3	O	外部总线的读信号线
				TxD2_2	O	串口 2 的发送脚

编号				名称	类型	说明
LQFP64	LQFP48	LQFP44	PDIP40			
44	32	29	31	P2.1	I/O	标准 IO 口
				A9	I	地址总线
				PWM1	O	增强 PWM 通道 1 输出脚
45	33	30	32	P2.2	I/O	标准 IO 口
				A10	I	地址总线
				PWM2	O	增强 PWM 通道 2 输出脚
				SS_2	I	SPI 的从机选择脚 (主机为输出)
46	34	31	33	ECL_2	I	PCA 的外部脉冲输入
				P2.3	I/O	标准 IO 口
				A11	I	地址总线
				PWM3	O	增强 PWM 通道 3 输出脚
47	35	32	34	MOSI_2	I/O	SPI 主机输出从机输入
				CCP0_2	I/O	PCA 的捕获输入和脉冲输出
				P2.4	I/O	标准 IO 口
				A12	I	地址总线
				PWM4	O	增强 PWM 通道 4 输出脚
				MISO_2	I/O	SPI 主机输入从机输出
48	36	33	35	SDA_2	I/O	I2C 接口的数据线
				CCP1_2	I/O	PCA 的捕获输入和脉冲输出
				P2.5	I/O	标准 IO 口
				A13	I	地址总线
				PWM5	O	增强 PWM 通道 5 输出脚
				SCLK_2	I/O	SPI 的时钟脚
49	37	34	36	SCL_2	I/O	I2C 的时钟线
				CCP2_2	I/O	PCA 的捕获输入和脉冲输出
				P2.6	I/O	标准 IO 口
				A14	I	地址总线
50	38	35	37	PWM6	O	增强 PWM 通道 6 输出脚
				CCP3_2	I/O	PCA 的捕获输入和脉冲输出
				P2.7	I/O	标准 IO 口
51	39	36	38	A15	I	地址总线
				PWM7	O	增强 PWM 通道 7 输出脚
				P0.0	I/O	标准 IO 口
				AD0	I	地址总线
				ADC8	I	ADC 模拟输入通道 8
				RxD3	I	串口 3 的接收脚

编号				名称	类型	说明
LQFP64	LQFP48	LQFP44	PDIP40			
52	40	37	39	P0.1	I/O	标准 IO 口
				AD1	I	地址总线
				ADC9	I	ADC 模拟输入通道 9
				TxD3	O	串口 3 的发送脚
53	41	38	40	P0.2	I/O	标准 IO 口
				AD2	I	地址总线
				ADC10	I	ADC 模拟输入通道 10
				RxD4	I	串口 4 的接收脚
54				P7.4	I/O	标准 IO 口
				SS_3	I	SPI 的从机选择脚 (主机为输出)
				ECL_3	I	PCA 的外部脉冲输入
55				P7.5	I/O	标准 IO 口
				MOSI_3	I/O	SPI 主机输出从机输入
56				P7.6	I/O	标准 IO 口
				MISO_3	I/O	SPI 主机输入从机输出
				SDA_3	I/O	I2C 接口的数据线
57				P7.7	I/O	标准 IO 口
				SCLK_3	I/O	SPI 的时钟脚
				SCL_3	I/O	I2C 的时钟线
58	42	39		P4.3	I/O	标准 IO 口
				WR	O	外部总线的写信号线
				RxD_4	I	串口 1 的接收脚
59	43	40	1	P0.3	I/O	标准 IO 口
				AD3	I	地址总线
				ADC11	I	ADC 模拟输入通道 11
				TxD4	O	串口 4 的发送脚
60	44	41	2	P0.4	I/O	标准 IO 口
				AD4	I	地址总线
				ADC12	I	ADC 模拟输入通道 12
				T3	I	定时器 3 外部时钟输入

编号				名称	类型	说明
LQFP64	LQFP48	LQFP44	PDIP40			
61	45	42	3	P0.5	I/O	标准 IO 口
				AD5	I	地址总线
				ADC13	I	ADC 模拟输入通道 13
				T3CLKO	O	定时器 3 时钟分频输出
62	46	43	4	P0.6	I/O	标准 IO 口
				AD6	I	地址总线
				ADC14	I	ADC 模拟输入通道 14
				T4	I	定时器 4 外部时钟输入
63	47	44	5	P0.7	I/O	标准 IO 口
				AD7	I	地址总线
				T4CLKO	O	定时器 4 时钟分频输出
64	48	1	6	P1.0	I/O	标准 IO 口
				ADC0	I	ADC 模拟输入通道 0
				PWM0_2	O	增强 PWM 通道 0 输出脚
				RxD2	I	串口 2 的接收脚

## 3 功能脚切换

STC8A8K64D4 系列单片机的特殊外设串口、SPI、PCA、I<sup>2</sup>C 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。

### 3.1 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		
BUS_SPEED	总线速度控制寄存器	A1H	RW_S[1:0]							SPEED[1:0]		00xx,xx00
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	S4_S	S3_S	S2_S	0x00,0000	

符号	描述	地址	位地址与符号								复位值	
			B7	B6	B5	B4	B3	B2	B1	B0		
PWM0CR	PWM0 控制寄存器	FF14H	ENC0O	C0INI	-	C0_S[1:0]		EC0I	EC0T2SI	EC0T1SI	00x0,0000	
PWM1CR	PWM1 控制寄存器	FF1CH	ENC1O	C1INI	-	C1_S[1:0]		EC1I	EC1T2SI	EC1T1SI	00x0,0000	
PWM2CR	PWM2 控制寄存器	FF24H	ENC2O	C2INI	-	C2_S[1:0]		EC2I	EC2T2SI	EC2T1SI	00x0,0000	
PWM3CR	PWM3 控制寄存器	FF2CH	ENC3O	C3INI	-	C3_S[1:0]		EC3I	EC3T2SI	EC3T1SI	00x0,0000	
PWM4CR	PWM4 控制寄存器	FF34H	ENC4O	C4INI	-	C4_S[1:0]		EC4I	EC4T2SI	EC4T1SI	00x0,0000	
PWM5CR	PWM5 控制寄存器	FF3CH	ENC5O	C5INI	-	C5_S[1:0]		EC5I	EC5T2SI	EC5T1SI	00x0,0000	
PWM6CR	PWM6 控制寄存器	FF44H	ENC6O	C6INI	-	C6_S[1:0]		EC6I	EC6T2SI	EC6T1SI	00x0,0000	
PWM7CR	PWM7 控制寄存器	FF4CH	ENC7O	C7INI	-	C7_S[1:0]		EC7I	EC7T2SI	EC7T1SI	00x0,0000	
MCLKOCR	主时钟输出控制寄存器	FE05H	MCLKO_S	MCLKODIV[6:0]								0000,0000
LCMIFCFG	LCM 接口配置寄存器	FE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFPS[1:0]		D16_D8	M68_80	0x00,0000	
LCMIFCFG2	LCM 接口配置寄存器 2	FE51H		LCMIFEXPS[1:0]		SETUPT[2:0]			HOLDT[1:0]		x000,0000	



### 3.1.1 总线速度控制寄存器 (BUS\_SPEED)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
BUS_SPEED	A1H	RW_S[1:0]						SPEED[1:0]	

RW\_S[1:0]: 外部总线 RD/WR 控制线选择位

RW_S[1:0]	RD	WR
00	P4.4	P4.3
01	P3.7	P3.6
10	P4.2	P4.0
11	保留	

### 3.1.2 外设端口切换控制寄存器 1 (P\_SW1), 串口 1、CCP、SPI 切换

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
11	P4.3	P4.4

CCP\_S[1:0]: PCA 功能脚选择位

CCP_S[1:0]	ECI	CCP0	CCP1	CCP2	CCP3
00	P1.2	P1.7	P1.6	P1.5	P1.4
01	P2.2	P2.3	P2.4	P2.5	P2.6
10	P7.4	P7.0	P7.1	P7.2	P7.3
11	P3.5	P3.3	P3.2	P3.1	P3.0

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	P2.2	P2.3	P2.4	P2.5
10	P7.4	P7.5	P7.6	P7.7
11	P3.5	P3.4	P3.3	P3.2

### 3.1.3 外设端口切换控制寄存器 2 (P\_SW2), 串口 2/3/4、I2C、比较器输出切换

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

EAXFR: 扩展 RAM 区特殊功能寄存器 (XFR) 访问控制寄存器

0: 禁止访问 XFR

1: 使能访问 XFR。

**当需要访问 XFR 时, 必须先将 EAXFR 置 1, 才能对 XFR 进行正常的读写**

I2C\_S[1:0]: I<sup>2</sup>C 功能脚选择位

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

CMPO\_S: 比较器输出脚选择位

CMPO_S	CMPO
0	P3.4
1	P4.1

S4\_S: 串口 4 功能脚选择位

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

S3\_S: 串口 3 功能脚选择位

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

S2\_S: 串口 2 功能脚选择位

S2_S	RxD2	TxD2
0	P1.0	P1.1
1	P4.0	P4.2

### 3.1.4 时钟选择寄存器 (MCLKOCR)

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

### 3.1.5 增强型 PWM 控制寄存器 (PWMnCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0CR	FF14H	ENC0O	C0INI	-	C0_S[1:0]	EC0I	EC0T2SI	EC0T1SI	
PWM1CR	FF1CH	ENC1O	C1INI	-	C1_S[1:0]	EC1I	EC1T2SI	EC1T1SI	
PWM2CR	FF24H	ENC2O	C2INI	-	C2_S[1:0]	EC2I	EC2T2SI	EC2T1SI	
PWM3CR	FF2CH	ENC3O	C3INI	-	C3_S[1:0]	EC3I	EC3T2SI	EC3T1SI	
PWM4CR	FF34H	ENC4O	C4INI	-	C4_S[1:0]	EC4I	EC4T2SI	EC4T1SI	

PWM5CR	FF3CH	ENC5O	C5INI	-	C5_S[1:0]	EC5I	EC5T2SI	EC5T1SI
PWM6CR	FF44H	ENC6O	C6INI	-	C6_S[1:0]	EC6I	EC6T2SI	EC6T1SI
PWM7CR	FF4CH	ENC7O	C7INI	-	C7_S[1:0]	EC7I	EC7T2SI	EC7T1SI

C0\_S[1:0]: 增强型 PWM 通道 0 输出脚选择位

C0_S[1:0]	PWM0
00	P2.0
01	P1.0
10	P6.0
11	保留

C1\_S[1:0]: 增强型 PWM 通道 1 输出脚选择位

C1_S[1:0]	PWM1
00	P2.1
01	P1.1
10	P6.1
11	保留

C2\_S[1:0]: 增强型 PWM 通道 2 输出脚选择位

C2_S[1:0]	PWM2
00	P2.2
01	P1.2
10	P6.2
11	保留

C3\_S[1:0]: 增强型 PWM 通道 3 输出脚选择位

C3_S[1:0]	PWM3
00	P2.3
01	P1.3
10	P6.3
11	保留

C4\_S[1:0]: 增强型 PWM 通道 4 输出脚选择位

C4_S[1:0]	PWM4
00	P2.4
01	P1.4
10	P6.4
11	保留

C5\_S[1:0]: 增强型 PWM 通道 5 输出脚选择位

C5_S[1:0]	PWM5
00	P2.5
01	P1.5
10	P6.5
11	保留

C6\_S[1:0]: 增强型 PWM 通道 6 输出脚选择位

C6_S[1:0]	PWM6
00	P2.6
01	P1.6
10	P6.6

11	保留
----	----

C7\_S[1:0]: 增强型 PWM 通道 7 输出脚选择位

C7_S[1:0]	PWM7
00	P2.7
01	P1.7
10	P6.7
11	保留

STC MCU

### 3.1.6 LCM 接口配置寄存器 (LCMIFCFG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCFG	FE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80
LCMIFCFG2	FE51H		LCMIFCPS[1:0]		SETUPT[2:0]			HOLDT[1:0]	

LCMIFCPS[1:0]: LCM 接口控制脚选择位

LCMIFCPS [1:0]	RS	I8080 的读信号 RD M6800 的使能信号 E	I8080 的写信号 WR M6800 的读写信号 RW
00	P4.1	P4.4	P4.3
01	P4.1	P3.7	P3.6
10	P4.1	P4.2	P4.0
11	P4.0	P3.7	P3.6

LCMIFDPS[1:0]: LCM 接口数据脚选择位

LCMIFDPS [1:0]	D16_D8	数据高字节 DB[15:8]	数据低字节 DB[7:0]
00	0	N/A	P2[7:0]
01	0	N/A	P6[7:0]
10	0	N/A	P2[7:0]
11	0	N/A	P6[7:0]
00	1	P2[7:0]	P0[7:0]
01	1	P6[7:0]	P2[7:0]
10	1	P2[7:0]	P7[7:0]
11	1	P6[7:0]	P7[7:0]

## 3.2 范例程序

### 3.2.1 串口 1 切换

#### C 语言代码

---



---

```
//测试工作频率为 11.0592MHz
```

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

```
sfr      P_SW1      = 0xa2;

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

```
void main()
```

```
{
```

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

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

```
    while (1);
```

```
}
```

---



---

#### 汇编代码

```
;测试工作频率为 11.0592MHz
```

```
P_SW1      DATA      0A2H

P0M1      DATA      093H
P0M0      DATA      094H
```

```

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

        ORG      0000H
        LJMP    MAIN

MAIN:    ORG      0100H

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

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

        SJMP   $

        END

```

### 3.2.2 串口 2 切换

#### C 语言代码

```
//测试工作频率为 11.0592MHz
```

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

```

sfr      P_SW2      = 0xba;

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

```

```

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

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

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

    while (1);
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

```

P_SW2    DATA    0BAH

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

                ORG    0000H
                LJMP   MAIN

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

```



```

MOV    P4M0, #00H
MOV    P4M1, #00H
MOV    P5M0, #00H
MOV    P5M1, #00H

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

SJMP   $

END

```

### 3.2.3 串口 3 切换

#### C 语言代码

```
//测试工作频率为 11.0592MHz
```

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

```

sfr    P_SW2      = 0xba;

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

```

```
void main()
```

```

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

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

    while (1);
}

```

#### 汇编代码

;测试工作频率为 11.0592MHz

```

P_SW2      DATA      0BAH

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

          ORG          0000H
          LJMP         MAIN

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

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

          SJMP        $

          END

```

### 3.2.4 串口 4 切换

#### C 语言代码

//测试工作频率为 11.0592MHz

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

```

sfr      P_SW2      = 0xba;

sfr      P0M1       = 0x93;
sfr      P0M0       = 0x94;
sfr      P1M1       = 0x91;
sfr      P1M0       = 0x92;

```

```

sfr    P2M1    =    0x95;
sfr    P2M0    =    0x96;
sfr    P3M1    =    0xb1;
sfr    P3M0    =    0xb2;
sfr    P4M1    =    0xb3;
sfr    P4M0    =    0xb4;
sfr    P5M1    =    0xc9;
sfr    P5M0    =    0xca;

```

```

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

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

    while (1);
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

```

P_SW2      DATA      0BAH

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

                ORG      0000H
                LJMP     MAIN

                ORG      0100H

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H

```

```

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

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

SJMP     $

END

```

### 3.2.5 SPI 切换

#### C 语言代码

//测试工作频率为 11.0592MHz

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

```

sfr      P_SW1      = 0xa2;

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

```

```
void main()
```

```
{
```

```

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

```

```

P_SW1 = 0x00;          //SS/P1.2, MOSI/P1.3, MISO/P1.4, SCLK/P1.5
// P_SW1 = 0x04;      //SS_2/P2.2, MOSI_2/P2.3, MISO_2/P2.4, SCLK_2/P2.5
// P_SW1 = 0x08;      //SS_3/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);
}
```

## 汇编代码

;测试工作频率为11.0592MHz

```
P_SW1      DATA      0A2H

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

ORG        0000H
LJMP      MAIN

MAIN:      ORG        0100H

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

MOV        P_SW1, #00H ;SS/P1.2, MOSI/P1.3, MISO/P1.4, SCLK/P1.5
;
MOV        P_SW1, #04H ;SS_2/P2.2, MOSI_2/P2.3, MISO_2/P2.4, SCLK_2/P2.5
;
MOV        P_SW1, #08H ;SS_3/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
```

## 3.2.6 PWM 切换

### 汇编代码

```
P_SW2      DATA      0BAH
PWM0CR     EQU        0FF14H
```

```

PWM1CR    EQU    0FF1CH
PWM2CR    EQU    0FF24H
PWM3CR    EQU    0FF2CH
PWM4CR    EQU    0FF34H
PWM5CR    EQU    0FF3CH
PWM6CR    EQU    0FF44H
PWM7CR    EQU    0FF4CH

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

                ORG    0000H
                LJMP   MAIN

                ORG    0100H
MAIN:
                MOV    SP, #3FH
                MOV    P0M0, #00H
                MOV    P0M1, #00H
                MOV    P1M0, #00H
                MOV    P1M1, #00H
                MOV    P2M0, #00H
                MOV    P2M1, #00H
                MOV    P3M0, #00H
                MOV    P3M1, #00H
                MOV    P4M0, #00H
                MOV    P4M1, #00H
                MOV    P5M0, #00H
                MOV    P5M1, #00H

                MOV    P_SW2, #80H
                MOV    A, #00H                ;PWM0/P2.0
                MOV    A, #08H                ;PWM0_2/P1.0
                MOV    A, #10H                ;PWM0_3/P6.0
                MOV    DPTR, #PWM0CR
                MOVX   @DPTR, A
                MOV    A, #00H                ;PWM1/P2.1
                MOV    A, #08H                ;PWM1_2/P1.1
                MOV    A, #10H                ;PWM1_3/P6.1
                MOV    DPTR, #PWM1CR
                MOVX   @DPTR, A
                MOV    A, #00H                ;PWM2/P2.2
                MOV    A, #08H                ;PWM2_2/P1.2
                MOV    A, #10H                ;PWM2_3/P6.2
                MOV    DPTR, #PWM2CR
                MOVX   @DPTR, A
                MOV    A, #00H                ;PWM3/P2.3
                MOV    A, #08H                ;PWM3_2/P1.3
                MOV    A, #10H                ;PWM3_3/P6.3

```

```

MOV DPTR,#PWM3CR
MOVX @DPTR,A
MOV A,#00H ;PWM4/P2.4
; MOV A,#08H ;PWM4_2/P1.4
; MOV A,#10H ;PWM4_3/P6.4
MOV DPTR,#PWM4CR
MOVX @DPTR,A
MOV A,#00H ;PWM5/P2.5
; MOV A,#08H ;PWM5_2/P1.5
; MOV A,#10H ;PWM5_3/P6.5
MOV DPTR,#PWM5CR
MOVX @DPTR,A
MOV A,#00H ;PWM6/P2.6
; MOV A,#08H ;PWM6_2/P1.6
; MOV A,#10H ;PWM6_3/P6.6
MOV DPTR,#PWM6CR
MOVX @DPTR,A
MOV A,#00H ;PWM7/P2.7
; MOV A,#08H ;PWM7_2/P1.7
; MOV A,#10H ;PWM7_3/P6.7
MOV DPTR,#PWM7CR
MOVX @DPTR,A
MOV P_SW2,#00H

SJMP $

END

```

## C 语言代码

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

```

#define PWM0CR (*(unsigned char volatile xdata *)0xff14)
#define PWM1CR (*(unsigned char volatile xdata *)0xff1c)
#define PWM2CR (*(unsigned char volatile xdata *)0xff24)
#define PWM3CR (*(unsigned char volatile xdata *)0xff2c)
#define PWM4CR (*(unsigned char volatile xdata *)0xff34)
#define PWM5CR (*(unsigned char volatile xdata *)0xff3c)
#define PWM6CR (*(unsigned char volatile xdata *)0xff44)
#define PWM7CR (*(unsigned char volatile xdata *)0xff4c)

```

```

sfr P_SW2 = 0xba;

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

```

```

void main()
{
    P0M0 = 0x00;
}

```

```

P0MI = 0x00;
P1M0 = 0x00;
P1MI = 0x00;
P2M0 = 0x00;
P2MI = 0x00;
P3M0 = 0x00;
P3MI = 0x00;
P4M0 = 0x00;
P4MI = 0x00;
P5M0 = 0x00;
P5MI = 0x00;

P_SW2 = 0x80;
PWM0CR = 0x00;           //PWM0/P2.0
// PWM0CR = 0x08;       //PWM0_2/P1.0
// PWM0CR = 0x10;       //PWM0_3/P6.0
PWM1CR = 0x00;           //PWM1/P2.1
// PWM1CR = 0x08;       //PWM1_2/P1.1
// PWM1CR = 0x10;       //PWM1_3/P6.1
PWM2CR = 0x00;           //PWM2/P2.2
// PWM2CR = 0x08;       //PWM2_2/P1.2
// PWM2CR = 0x10;       //PWM2_3/P6.2
PWM3CR = 0x00;           //PWM3/P2.3
// PWM3CR = 0x08;       //PWM3_2/P1.3
// PWM3CR = 0x10;       //PWM3_3/P6.3
PWM4CR = 0x00;           //PWM4/P2.4
// PWM4CR = 0x08;       //PWM4_2/P1.4
// PWM4CR = 0x10;       //PWM4_3/P6.4
PWM5CR = 0x00;           //PWM5/P2.5
// PWM5CR = 0x08;       //PWM5_2/P1.5
// PWM5CR = 0x10;       //PWM5_3/P6.5
PWM6CR = 0x00;           //PWM6/P2.6
// PWM6CR = 0x08;       //PWM6_2/P1.6
// PWM6CR = 0x10;       //PWM6_3/P6.6
PWM7CR = 0x00;           //PWM7/P2.7
// PWM7CR = 0x08;       //PWM7_2/P1.7
// PWM7CR = 0x10;       //PWM7_3/P6.7
P_SW2 = 0x00;

while (1);
}

```

### 3.2.7 PCA/CCP/PWM 切换

#### 汇编代码

<i>P_SW1</i>	<i>DATA</i>	<i>0A2H</i>
<i>P0MI</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1MI</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2MI</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3MI</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4MI</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5MI</i>	<i>DATA</i>	<i>0C9H</i>



```

P5M0    DATA    0CAH

          ORG     0000H
          LJMP   MAIN

          ORG     0100H
MAIN:
          MOV     SP, #3FH
          MOV     P0M0, #00H
          MOV     P0M1, #00H
          MOV     P1M0, #00H
          MOV     P1M1, #00H
          MOV     P2M0, #00H
          MOV     P2M1, #00H
          MOV     P3M0, #00H
          MOV     P3M1, #00H
          MOV     P4M0, #00H
          MOV     P4M1, #00H
          MOV     P5M0, #00H
          MOV     P5M1, #00H

          MOV     P_SW1, #00H           ;ECI/P1.2, CCP0/P1.7, CCP1/P1.6, CCP2/P1.5, CCP3/P1.4
;          MOV     P_SW1, #10H          ;ECI_2/P2.2, CCP0_2/P2.3, CCP1_2/P2.4, CCP2_2/P2.5, CCP3_2/P2.6
;          MOV     P_SW1, #20H          ;ECI_3/P7.4, CCP0_3/P7.0, CCP1_3/P7.1, CCP2_3/P7.2, CCP3_3/P7.3
;          MOV     P_SW1, #30H          ;ECI_4/P3.5, CCP0_4/P3.3, CCP1_4/P3.2, CCP2_4/P3.1, CCP3_4/P3.0

          SJMP   $

          END

```

## C 语言代码

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

```

sfr      P_SW1      = 0xa2;

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

```

```
void main()
```

```

{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
}

```

```

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

P_SW1 = 0x00;           //ECI/P1.2, CCP0/P1.7, CCP1/P1.6, CCP2/P1.5, CCP3/P1.4
// P_SW1 = 0x10;       //ECI_2/P2.2, CCP0_2/P2.3, CCP1_2/P2.4, CCP2_2/P2.5, CCP3_2/P2.6
// P_SW1 = 0x20;       //ECI_3/P7.4, CCP0_3/P7.0, CCP1_3/P7.1, CCP2_3/P7.2, CCP3_3/P7.3
// P_SW1 = 0x30;       //ECI_4/P3.5, CCP0_4/P3.3, CCP1_4/P3.2, CCP2_4/P3.1, CCP3_4/P3.0

while (1);
}

```

## 3.2.8 I2C 切换

### C 语言代码

//测试工作频率为 11.0592MHz

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

```

sfr      P_SW2      = 0xba;

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

```

```
void main()
{
```

```

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

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

```

```
while (1);
```

}

---

**汇编代码**


---

;测试工作频率为 11.0592MHz

```

P_SW2      DATA      0BAH

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

          ORG          0000H
          LJMP        MAIN

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

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

          SJMP        $

          END

```

---

### 3.2.9 比较器输出切换

---

**C 语言代码**


---

//测试工作频率为 11.0592MHz

#include "reg51.h"

```

sfr    P_SW2    = 0xba;

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

```

```

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

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

    while (1);
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

```

P_SW2      DATA      0BAH

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

            ORG        0000H
            LJMP      MAIN

            ORG        0100H

```

MAIN:

```

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

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

SJMP    $

END

```

### 3.2.10 主时钟输出切换

#### C 语言代码

//测试工作频率为 11.0592MHz;

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

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

```
sfr     P_SW2      = 0xba;
```

```
sfr     P0M1      = 0x93;
```

```
sfr     P0M0      = 0x94;
```

```
sfr     P1M1      = 0x91;
```

```
sfr     P1M0      = 0x92;
```

```
sfr     P2M1      = 0x95;
```

```
sfr     P2M0      = 0x96;
```

```
sfr     P3M1      = 0xb1;
```

```
sfr     P3M0      = 0xb2;
```

```
sfr     P4M1      = 0xb3;
```

```
sfr     P4M0      = 0xb4;
```

```
sfr     P5M1      = 0xc9;
```

```
sfr     P5M0      = 0xca;
```

```
void main()
```

```
{
```

```
    P0M0 = 0x00;
```

```
    P0M1 = 0x00;
```

```
    P1M0 = 0x00;
```

```
    P1M1 = 0x00;
```

```
    P2M0 = 0x00;
```

```
    P2M1 = 0x00;
```

```
    P3M0 = 0x00;
```

```
    P3M1 = 0x00;
```

```

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

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

while (1);
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

```

P_SW2      DATA      0BAH

MCLKOCR    EQU        0FE05H

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

                ORG        0000H
                LJMP       MAIN

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

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

```

*SJMP*      \$

*END*

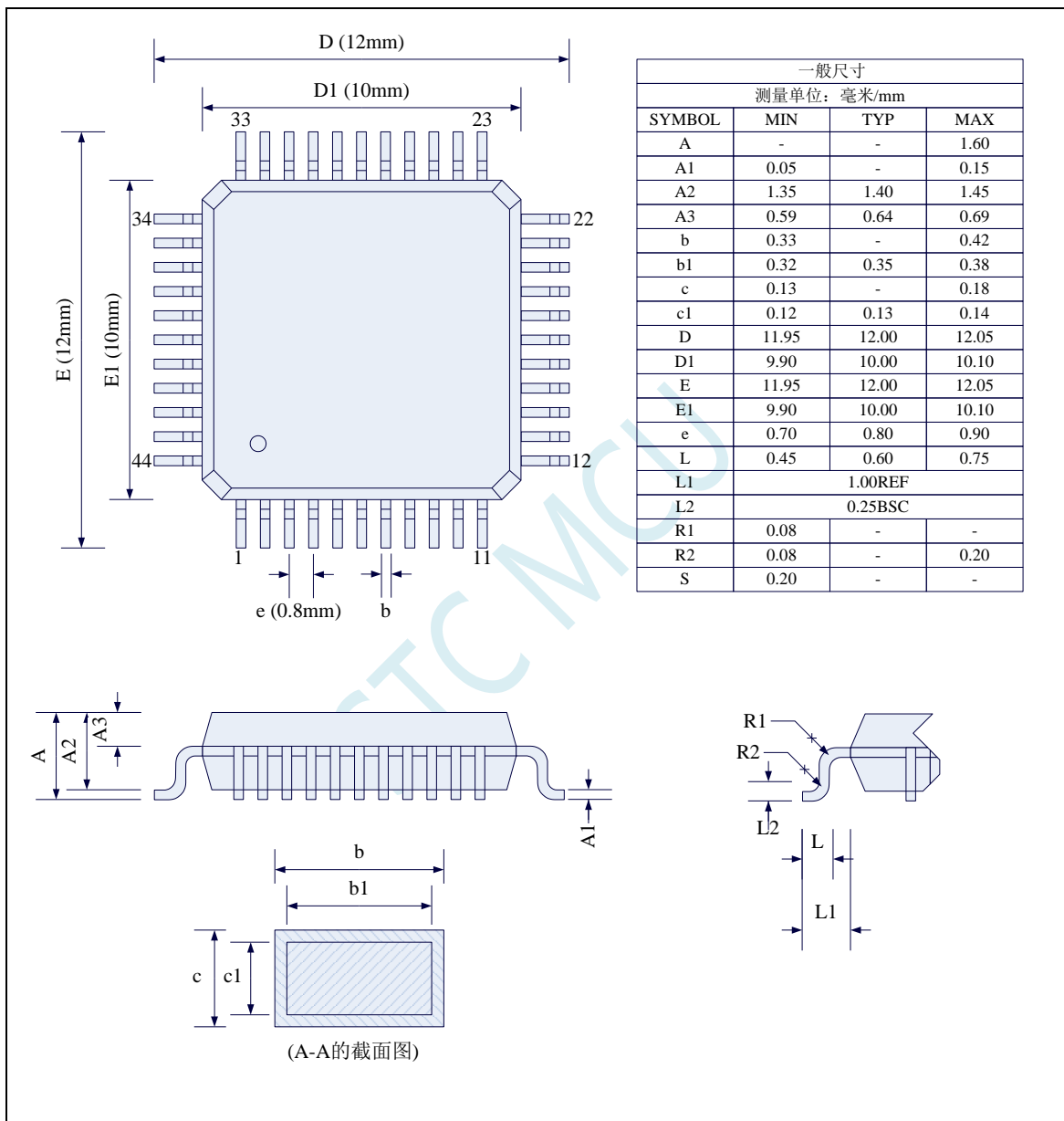
---

---

STC MCU

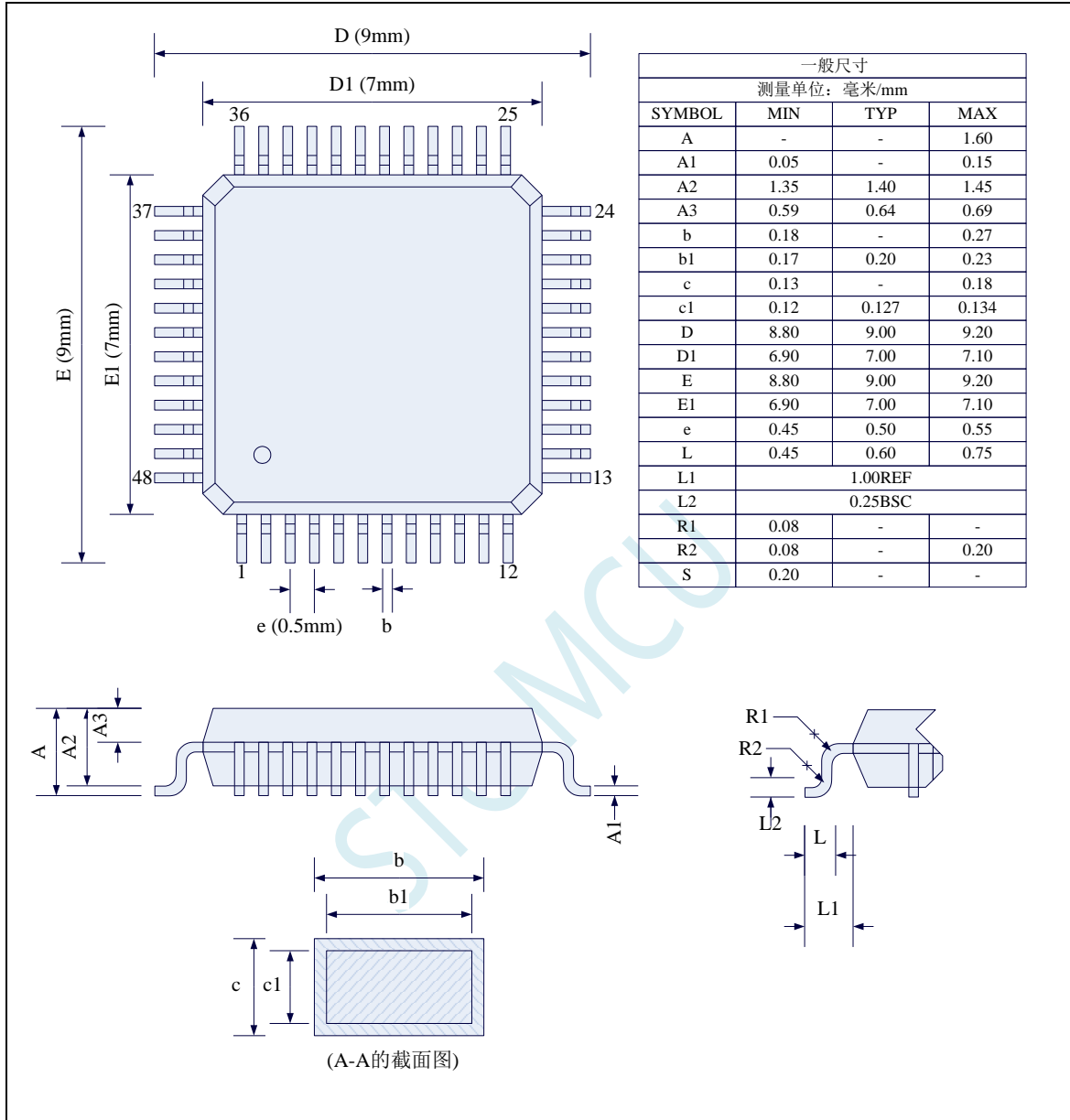
## 4 封装尺寸图

### 4.1 LQFP44 封装尺寸图 (12mm\*12mm)

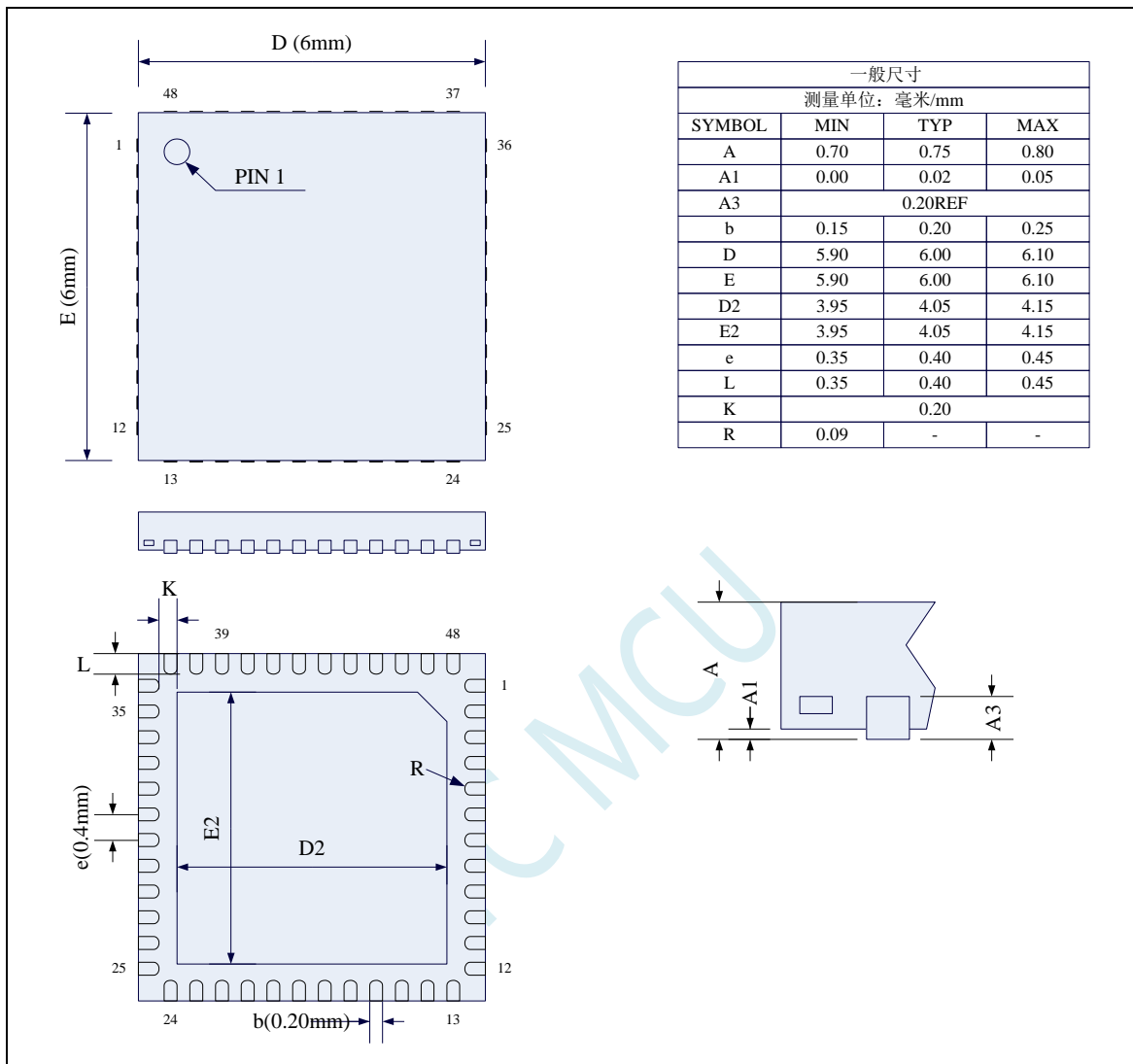




## 4.2 LQFP48 封装尺寸图 (9mm\*9mm)

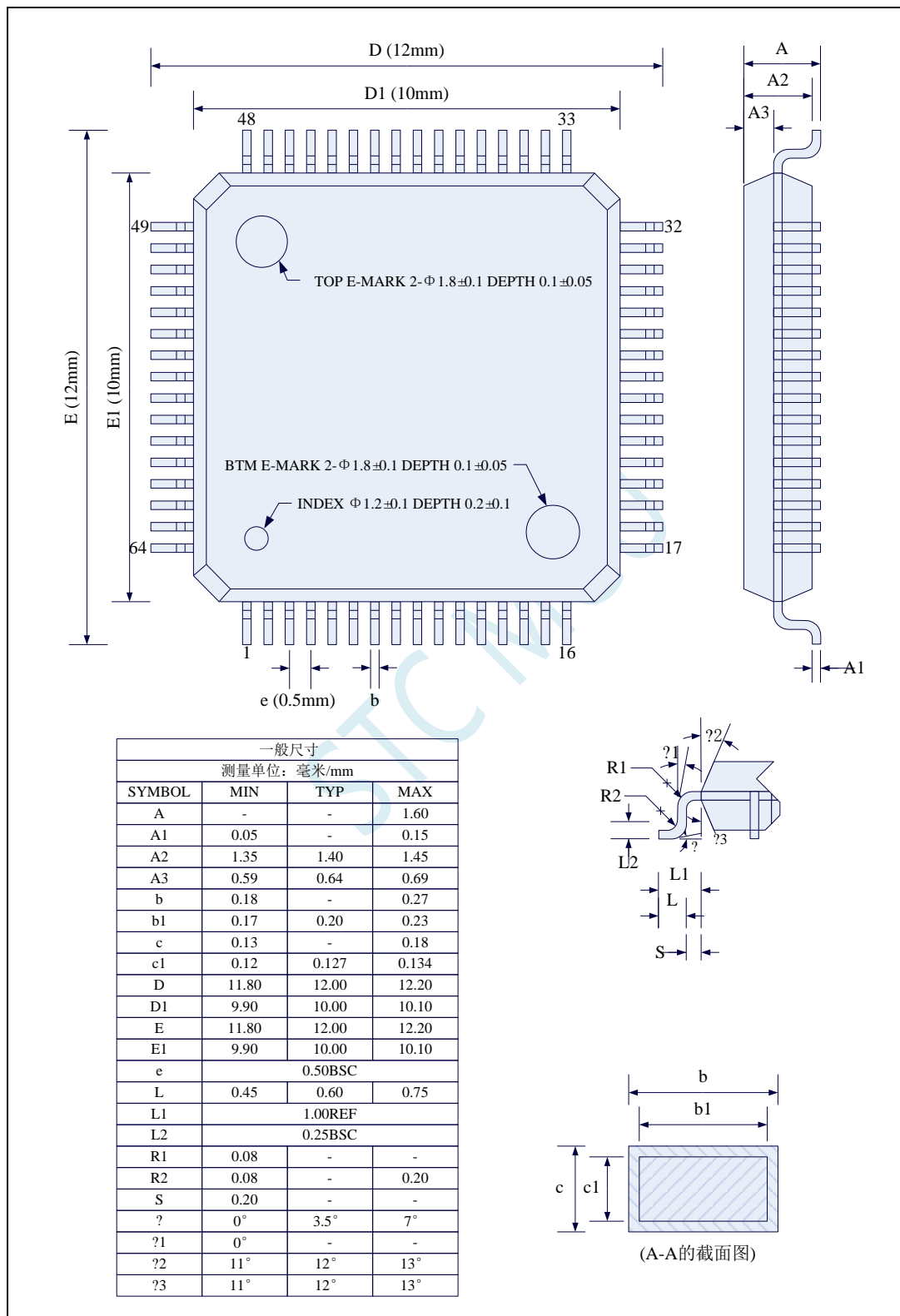


### 4.3 QFN48 封装尺寸图 (6mm\*6mm)

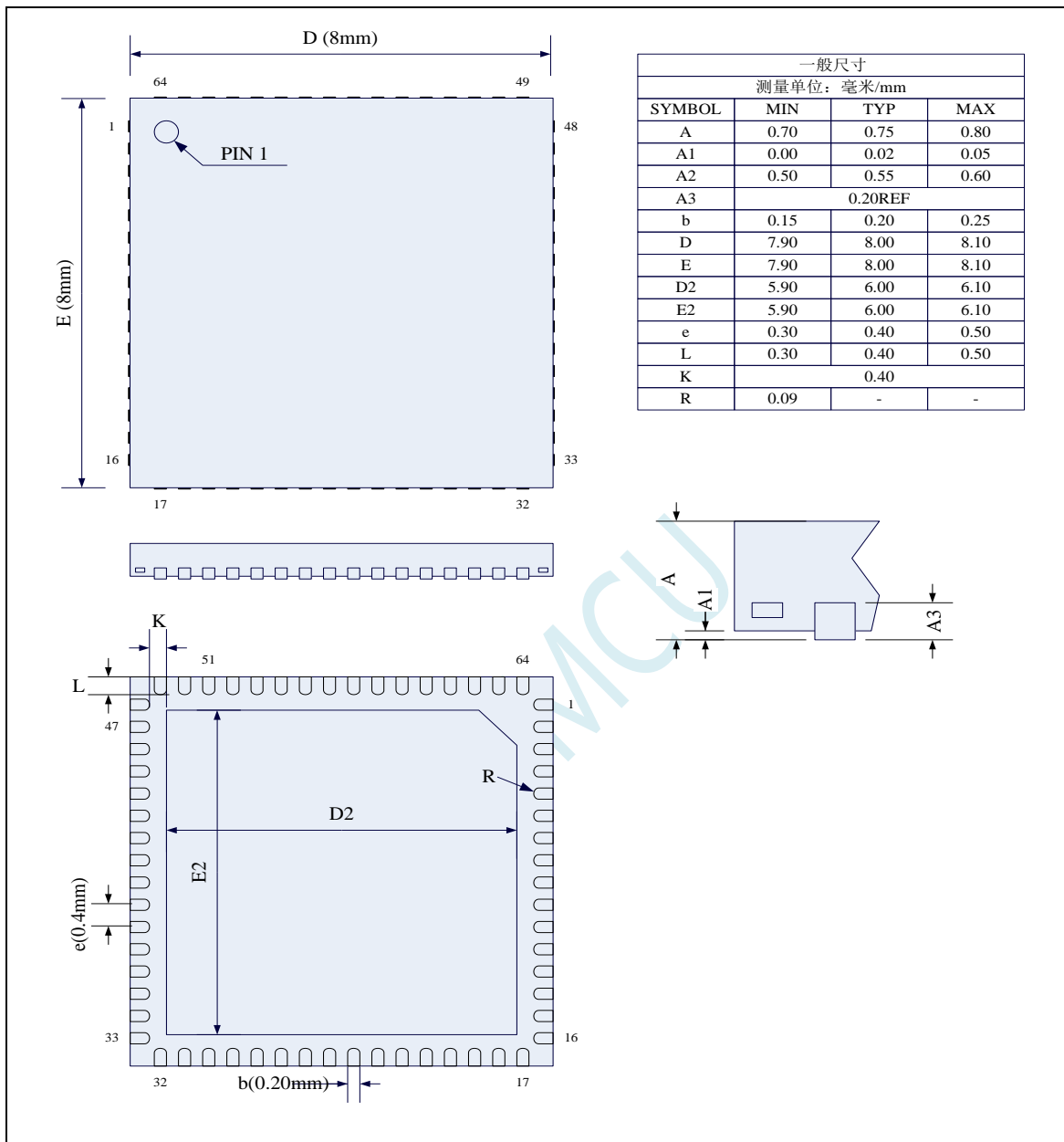


STC 现有 QFN48 封装芯片的背面金属片 (衬底), 在芯片内部并未接地, 在用户的 PCB 板上可以接地, 也可以不接地, 不会对芯片性能造成影响

### 4.4 LQFP64 封装尺寸图 (12mm\*12mm)

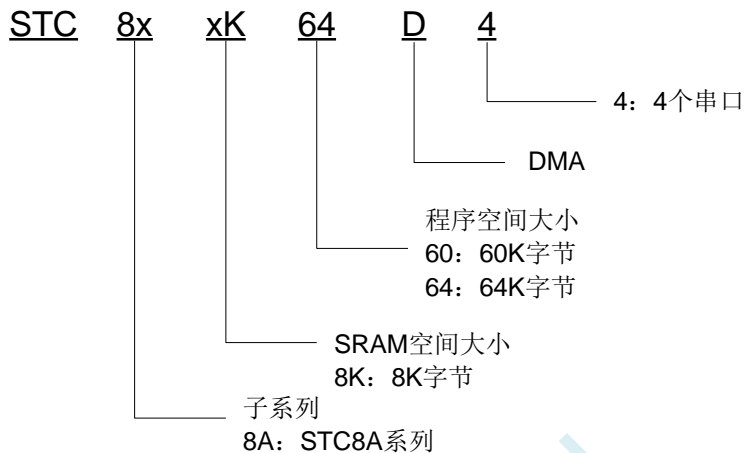


### 4.5 QFN64 封装尺寸图 (8mm\*8mm)



STC 现有 QFN64 封装芯片的背面金属片 (衬底), 在芯片内部并未接地, 在用户的 PCB 板上可以接地, 也可以不接地, 不会对芯片性能造成影响

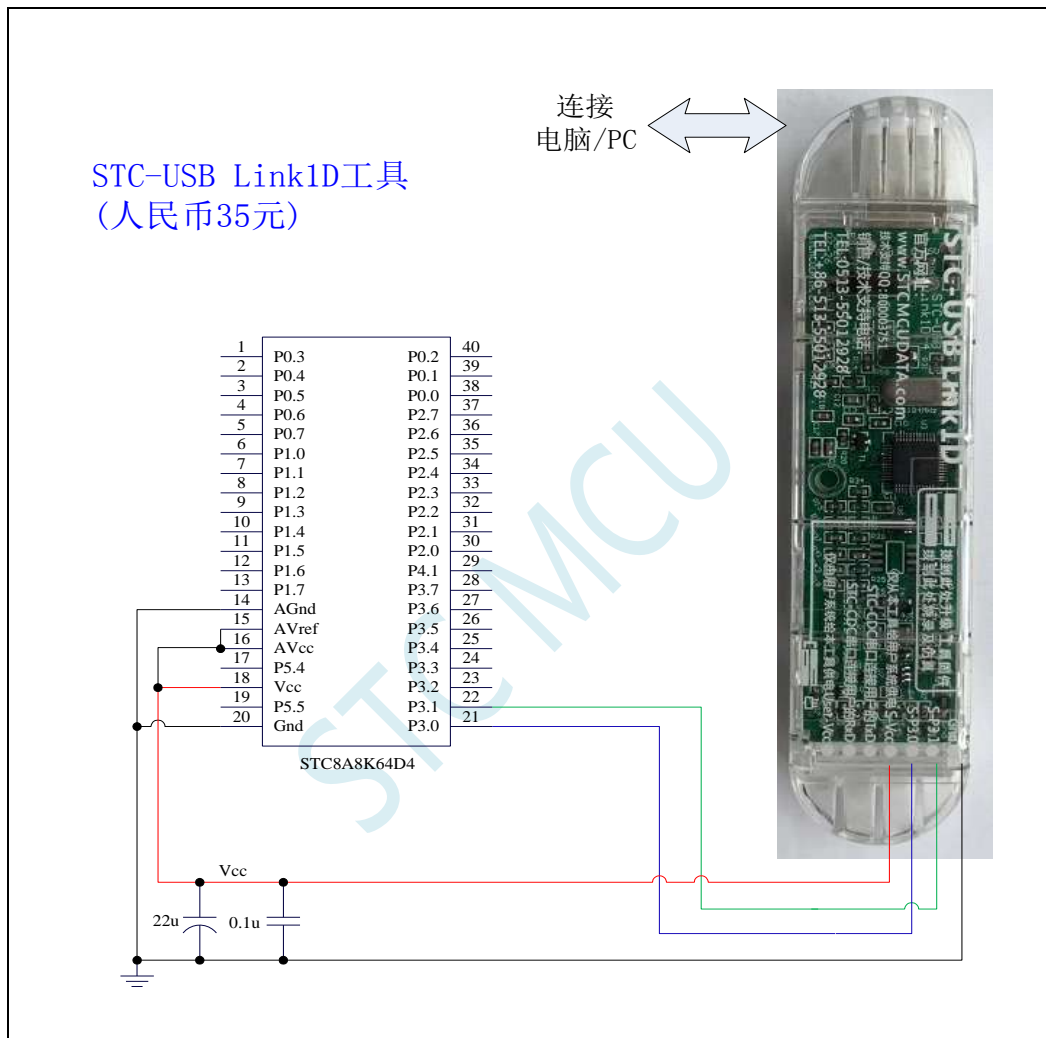
## 4.6 STC8A8K64D4 系列单片机命名规则



## 5 ISP 下载及典型应用线路图

### 5.1 STC8A8K64D4 系列 ISP 下载应用线路图

#### 5.1.1 使用 STC-USB Link1D 工具下载，支持在线和脱机下载

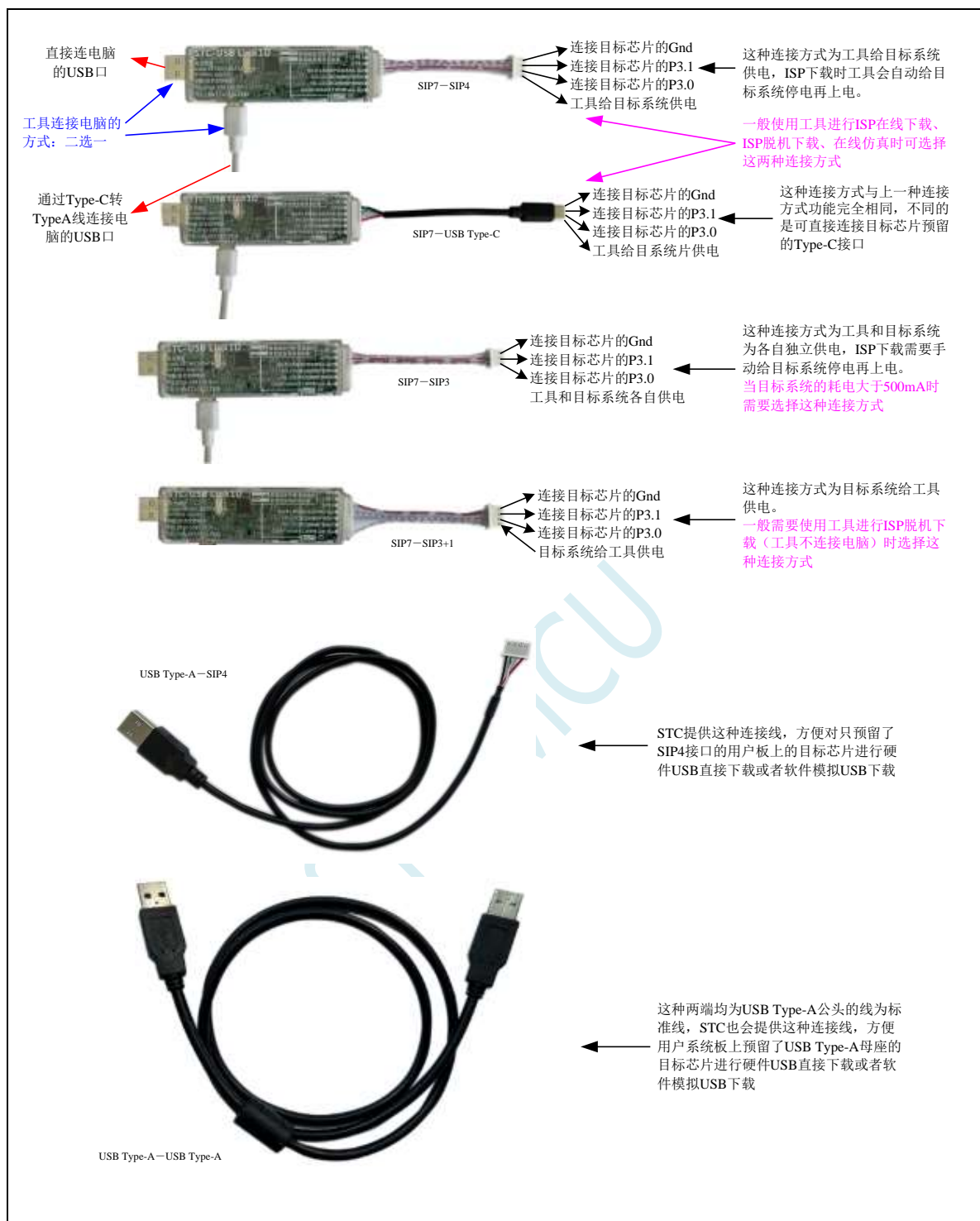


#### ISP 下载步骤:

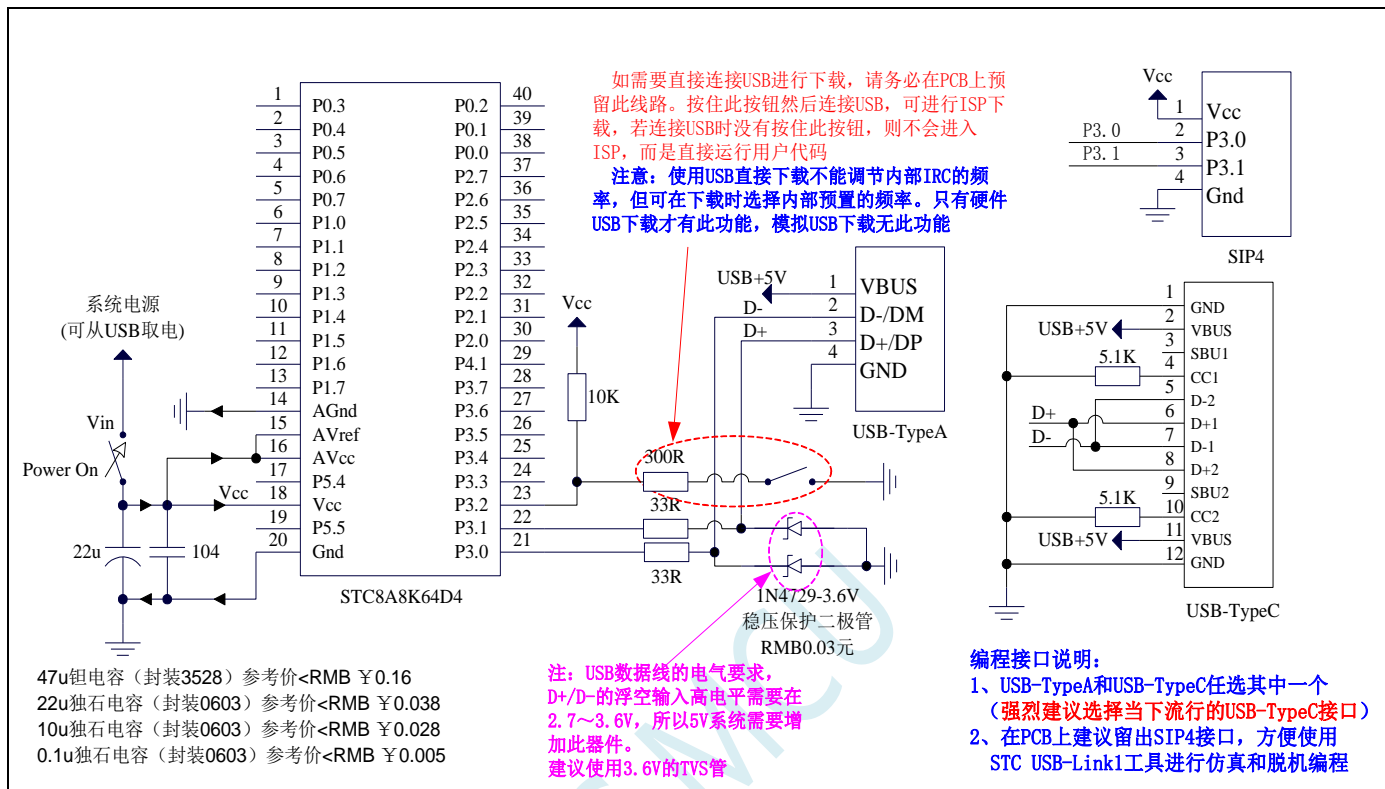
- 1、按照如图所示的连接方式将 STC-USB Link1D 和目标芯片连接
- 2、点击 STC-ISP 下载软件中的“下载/编程”按钮
- 3、开始 ISP 下载

注意：若是使用 STC-USB Link1D 给目标系统供电，目标系统的总电流不能大于 200mA，否则会导致下载失败。

注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

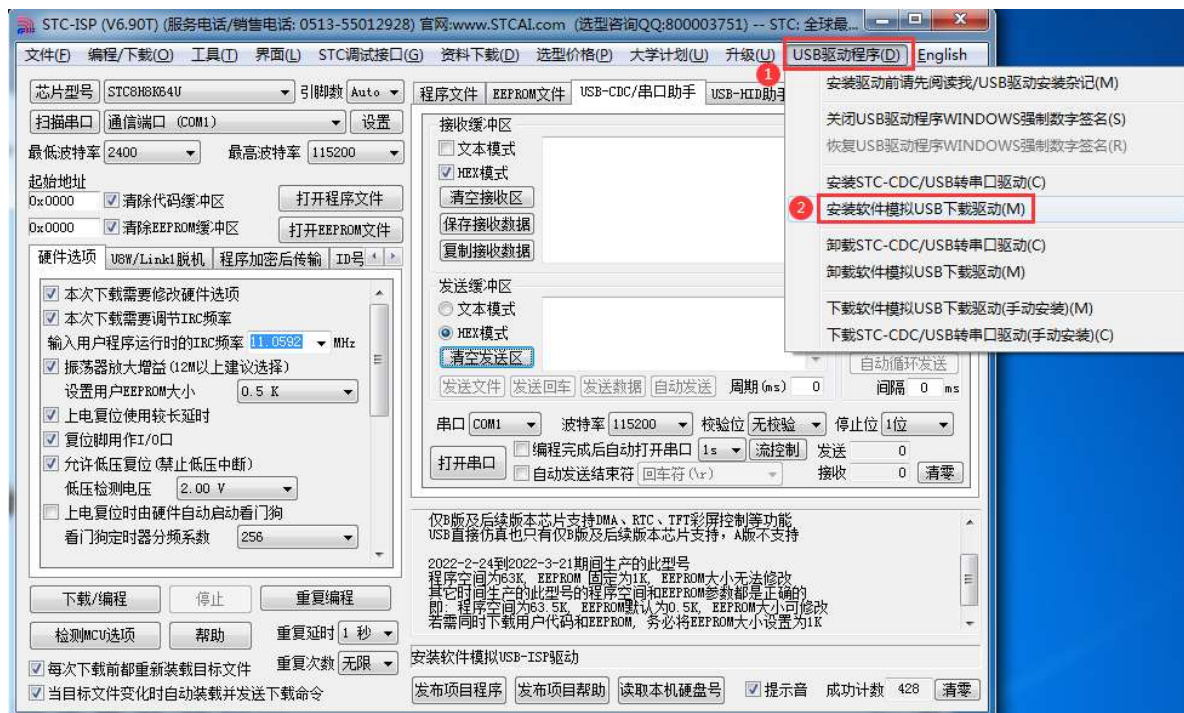


### 5.1.2 软件模拟 USB 直接 ISP 下载，建议尝试，不支持仿真（5V 系统）



现在 STC 的不带硬件 USB 的 STC8G/STC8H 的 MCU，基本都支持用软件模拟 USB 下载用户程序，因为走的是 USB-SCAN 通信协议，不管任何版本操作系统，都要安装驱动。在 STC-ISP 下载软件如下图所示的地方安装驱动。

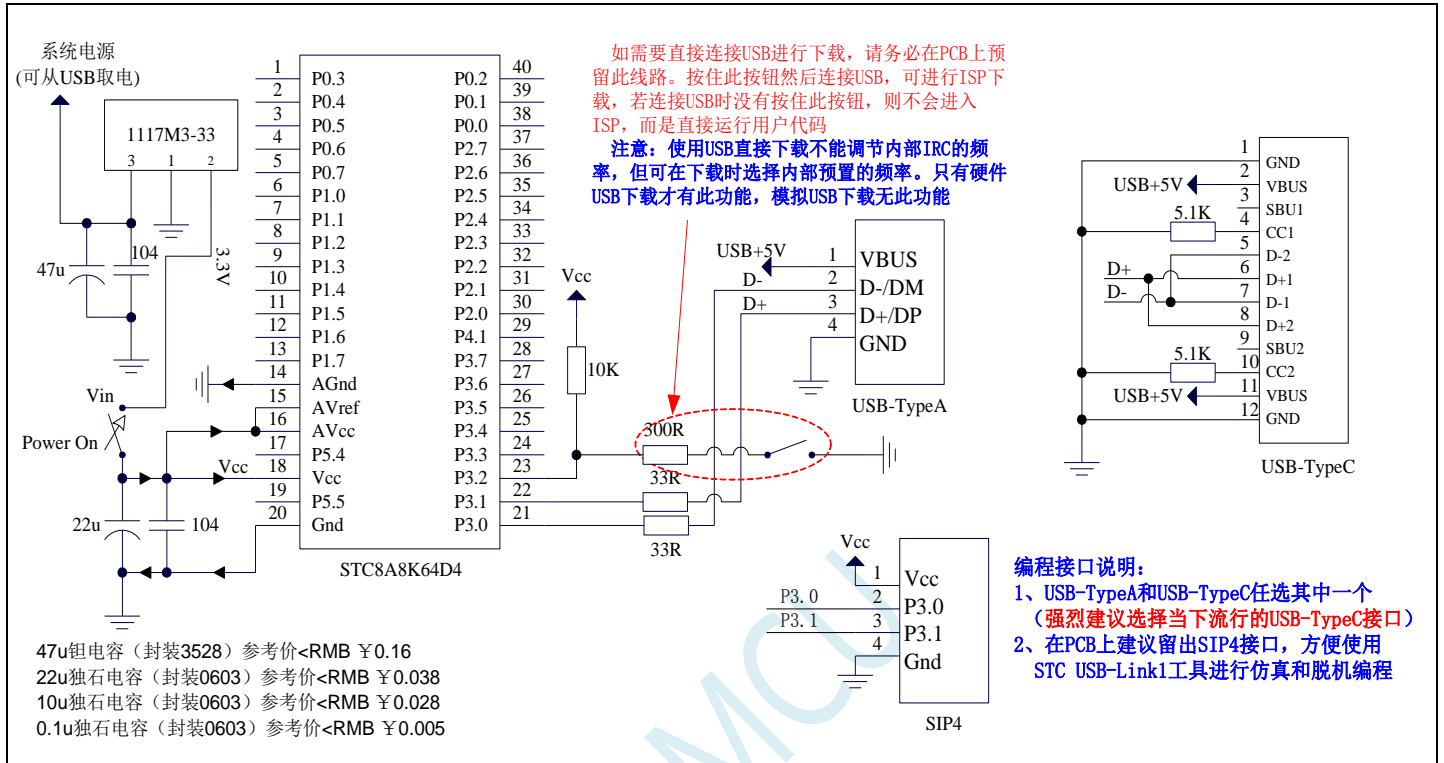




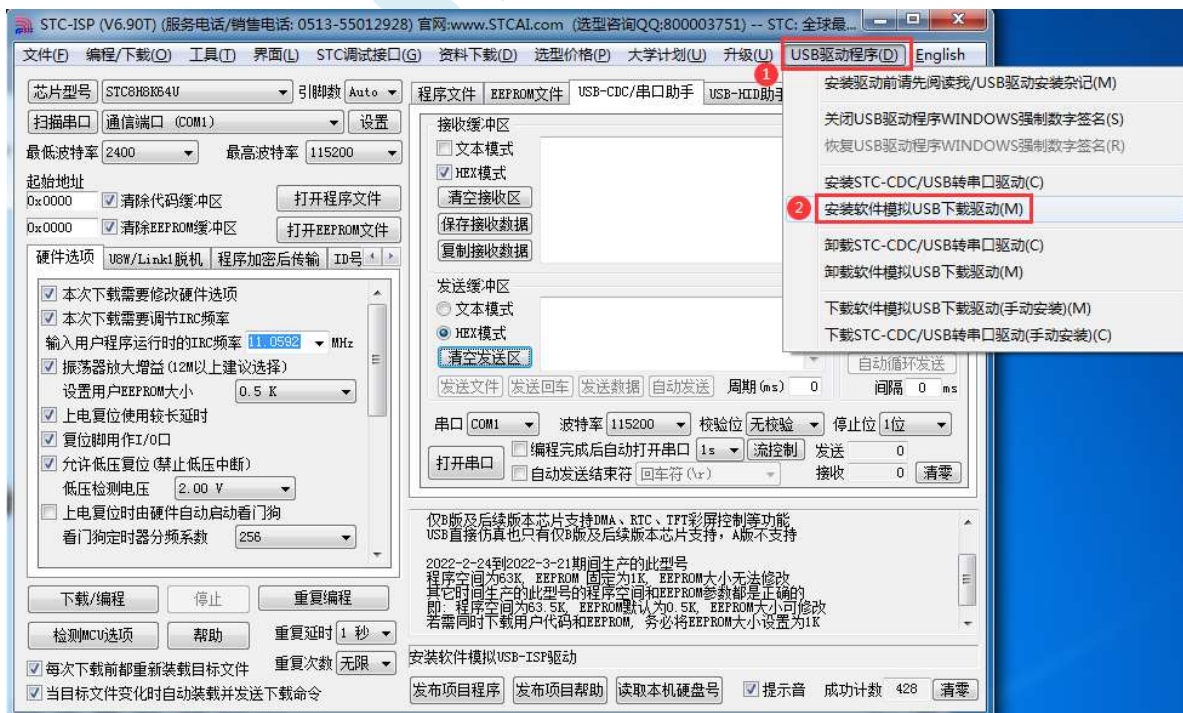
#### ISP 下载步骤:

- 1、D-/P3.0, D+/P3.1 与 PC-USB 端口连接好
- 2、将 P3.2 与 GND 短接, 实验箱子上的 P3.2/INT0 按键按下
- 3、给目标芯片重新上电。若目标芯片已经停电, 直接上电即可; 若目标芯片处于通电状态, 则需给目标芯片停电再上电(冷启动)。等待 STC-ISP 下载软件中自动识别出“STC USB Writer (HID1)”识别出来后, 就与 P3.2 状态无关了(此时就不需要一直按着 P3.2 口了, 一直按着把手累坏了不要紧, 将按键按坏就麻烦了)。
- 4、点击下载软件中的“下载/编程”按钮(注意: USB 下载与串口下载的操作顺序不同, 千万千万不要先点击下载按钮, 一定到等到电脑端识别出“STC USB Writer (HID1)”设备后, 才能点击下载按钮开始下载)

### 5.1.3 软件模拟 USB 直接 ISP 下载，建议尝试，不支持仿真（3.3V 系统）



现在 STC 的不带硬件 USB 的 STC8G/STC8H 的 MCU，基本都支持用软件模拟 USB 下载用户程序，因为走的是 USB-SCAN 通信协议，不管任何版本操作系统，都要安装驱动。在 STC-ISP 下载软件如下图所示的地方安装驱动。

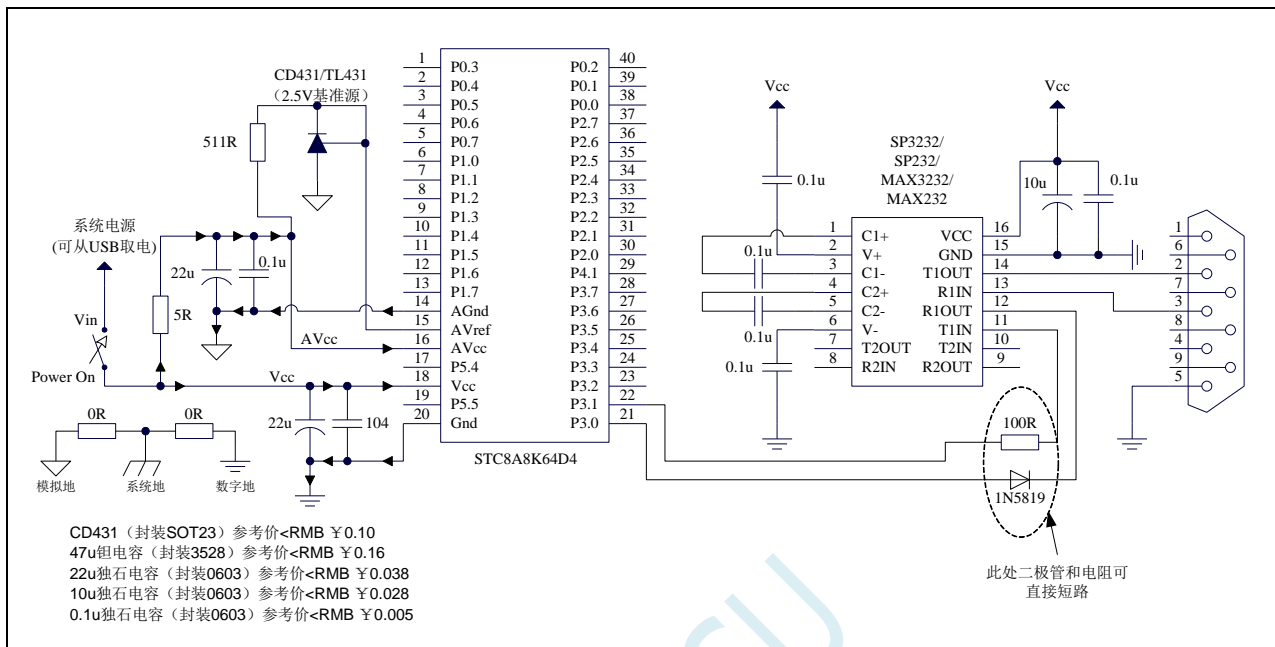


**ISP 下载步骤:**

- 1、D-/P3.0, D+/P3.1 与 PC-USB 端口连接好
- 2、将 P3.2 与 GND 短接, 实验箱板子上的 P3.2/INT0 按键按下
- 3、给目标芯片重新上电。若目标芯片已经停电, 直接上电即可; 若目标芯片处于通电状态, 则需给目标芯片停电再上电(冷启动)。等待 STC-ISP 下载软件中自动识别出“STC USB Writer (HID1)”识别出来后, 就与 P3.2 状态无关了(此时就不需要一直按着 P3.2 口了, 一直按着把手累坏了不要紧, 将按键按坏就麻烦了)。
- 4、点击下载软件中的“下载/编程”按钮(注意: USB 下载与串口下载的操作顺序不同, 千万千万不要先点击下载按钮, 一定到等到电脑端识别出“STC USB Writer (HID1)”设备后, 才能点击下载按钮开始下载)

STC MCU

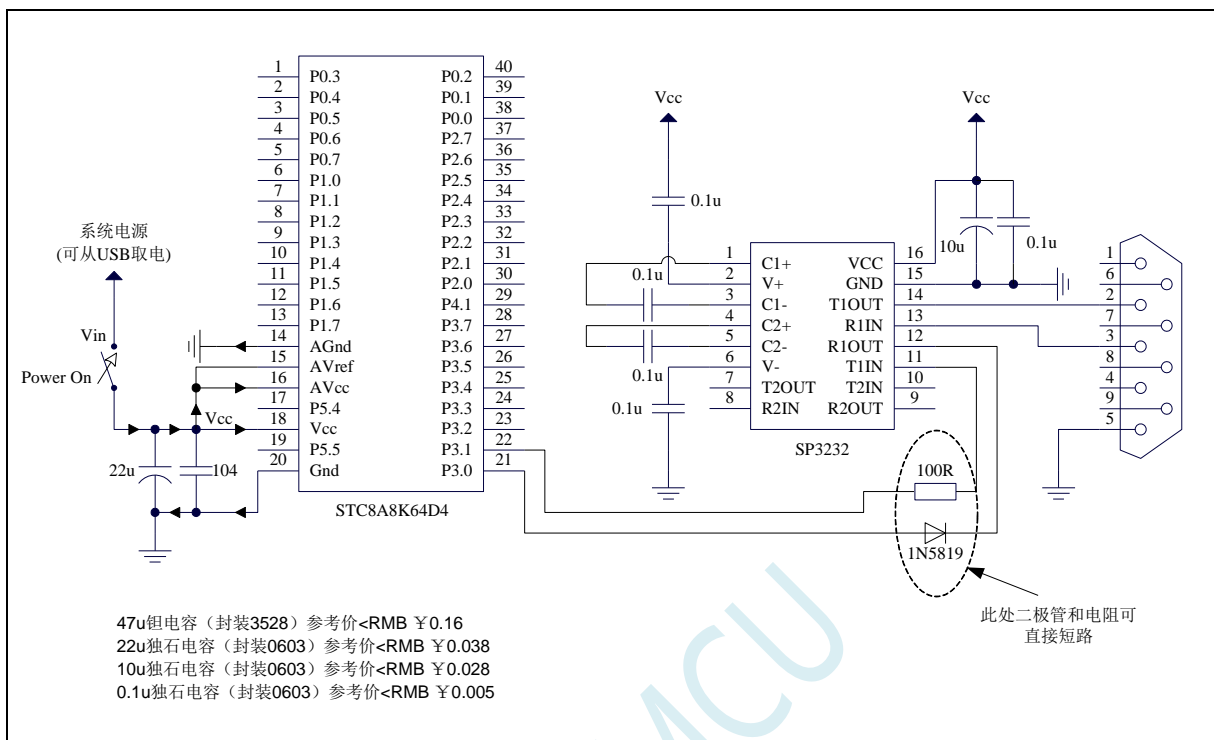
### 5.1.4 使用 RS-232 转换器下载, 也可支持仿真 (使用高精度 ADC)



#### ISP 下载步骤:

- 1、给目标芯片停电
- 2、点击 STC-ISP 下载软件中的“下载/编程”按钮
- 3、给目标芯片上电
- 4、开始 ISP 下载

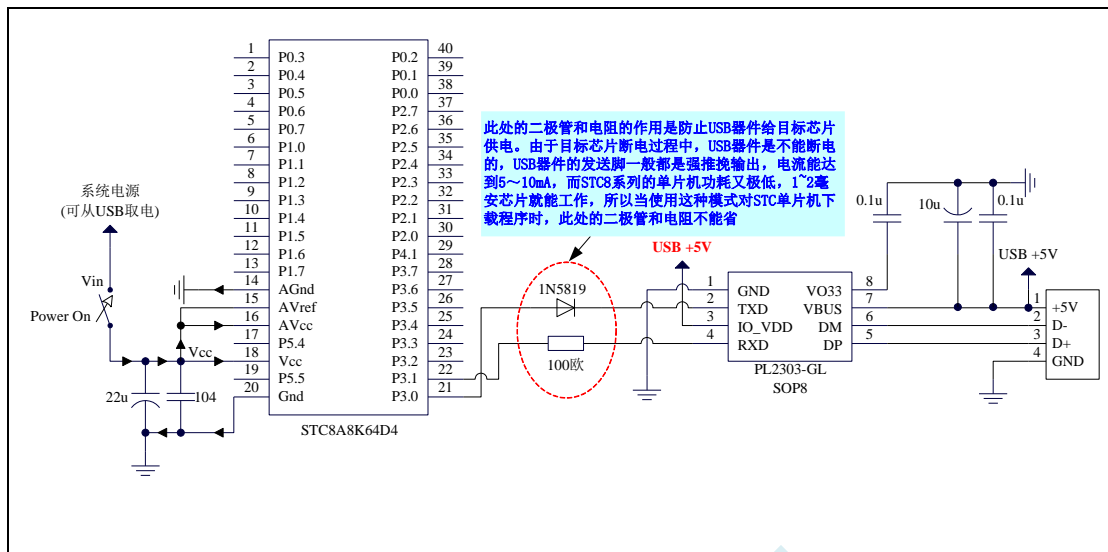
### 5.1.5 使用 RS-232 转换器下载，也可支持仿真（使用一般精度 ADC）



#### ISP 下载步骤:

- 1、给目标芯片停电
- 2、点击 STC-ISP 下载软件中的“下载/编程”按钮
- 3、给目标芯片上电
- 4、开始 ISP 下载

## 5.1.6 使用 PL2303-GL 下载，也可支持仿真

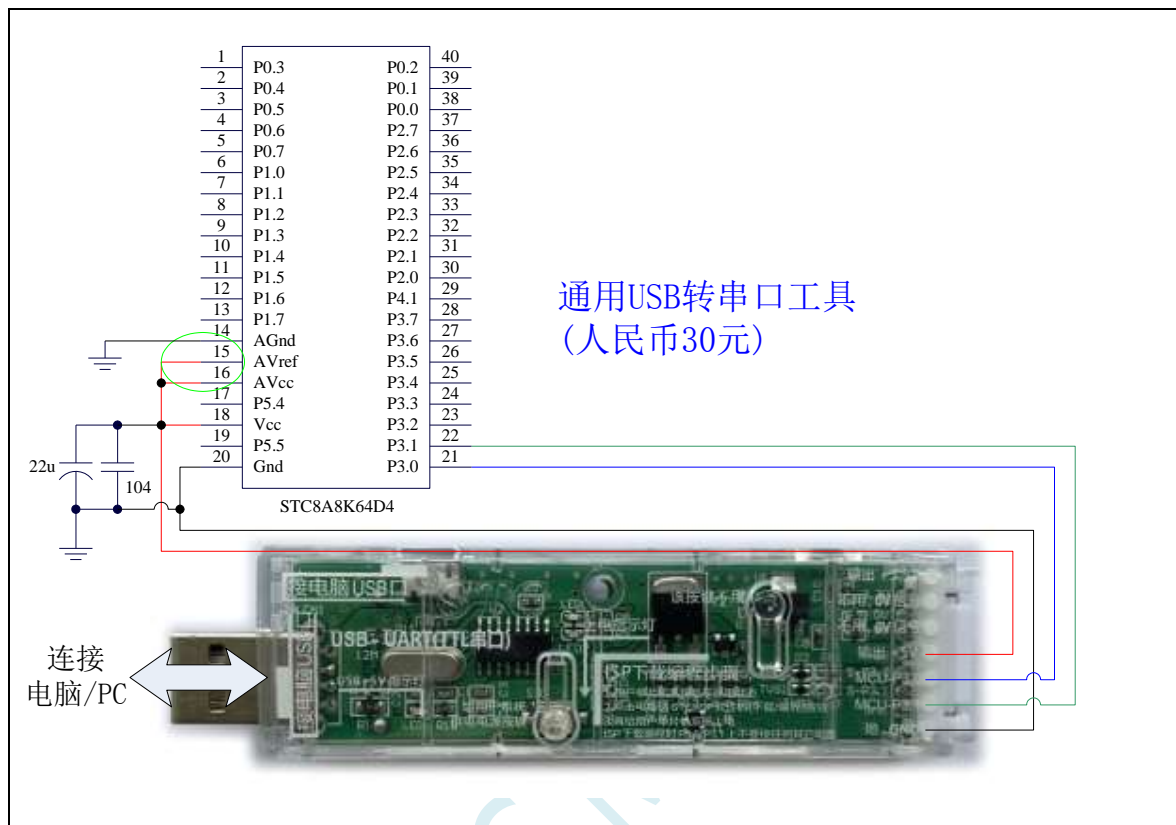


### ISP 下载步骤:

- 1、给目标芯片停电，注意不能给 USB 转串口芯片停电（如：CH340、PL2303-GL 等）  
注意：PL2303-SA 的部分波特率误差非常大，建议使用 PL2303-GL
- 2、由于 USB 转串口芯片的发送脚一般都是强推挽输出，必须在目标芯片的 P3.0 口和 USB 转串口芯片的发送脚之间串接一个二极管，否则目标芯片无法完全断电，达不到给目标芯片停电的目标。
- 3、点击 STC-ISP 下载软件中的“下载/编程”按钮
- 4、给目标芯片上电
- 5、开始 ISP 下载

**注意：**目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

### 5.1.7 使用通用 USB 转串口工具下载，支持 ISP 在线下载，也可支持仿真

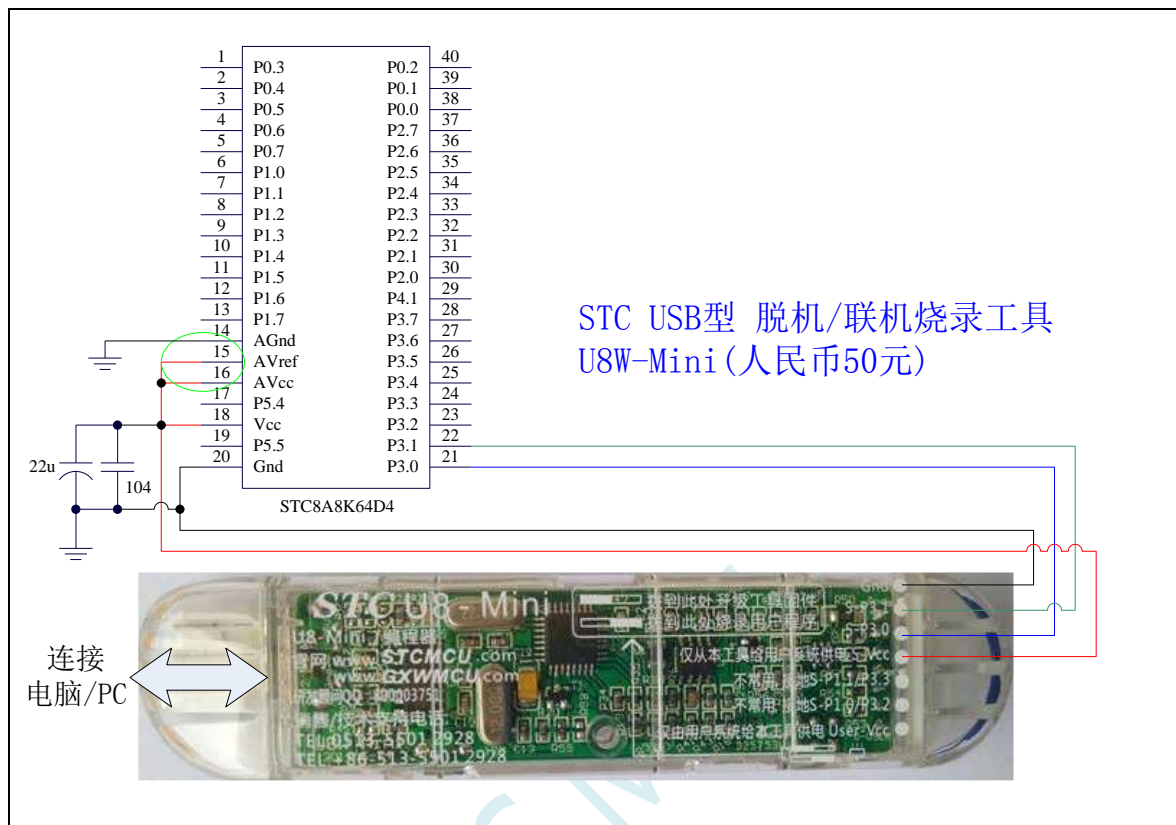


#### ISP 下载步骤:

- 1、按照如图所示的连接方式将通用 USB 转串口工具和目标芯片连接
- 2、按下电源按钮，确定目标芯片处于**停电状态**（上电指示灯为灭的状态）。**注意：工具第一次上电时是不对外供电的，因此若是第一次上电使用此工具，可跳过此步。**
- 3、点击 STC-ISP 下载软件中的“下载/编程”按钮
- 4、再次按下电源按钮，给目标芯片上电（上电指示灯为亮的状态）
- 5、开始 ISP 下载

**注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。**

## 5.1.8 使用 U8-Mini 工具下载，支持 ISP 在线和脱机下载，也可支持仿真



### ISP 下载步骤:

- 1、按照如图所示的连接方式将 U8-Mini 和目标芯片连接
- 2、点击 STC-ISP 下载软件中的“下载/编程”按钮
- 3、开始 ISP 下载

注意：若是使用 U8-Mini 给目标系统供电，目标系统的总电流不能大于 200mA，否则会导致下载失败。

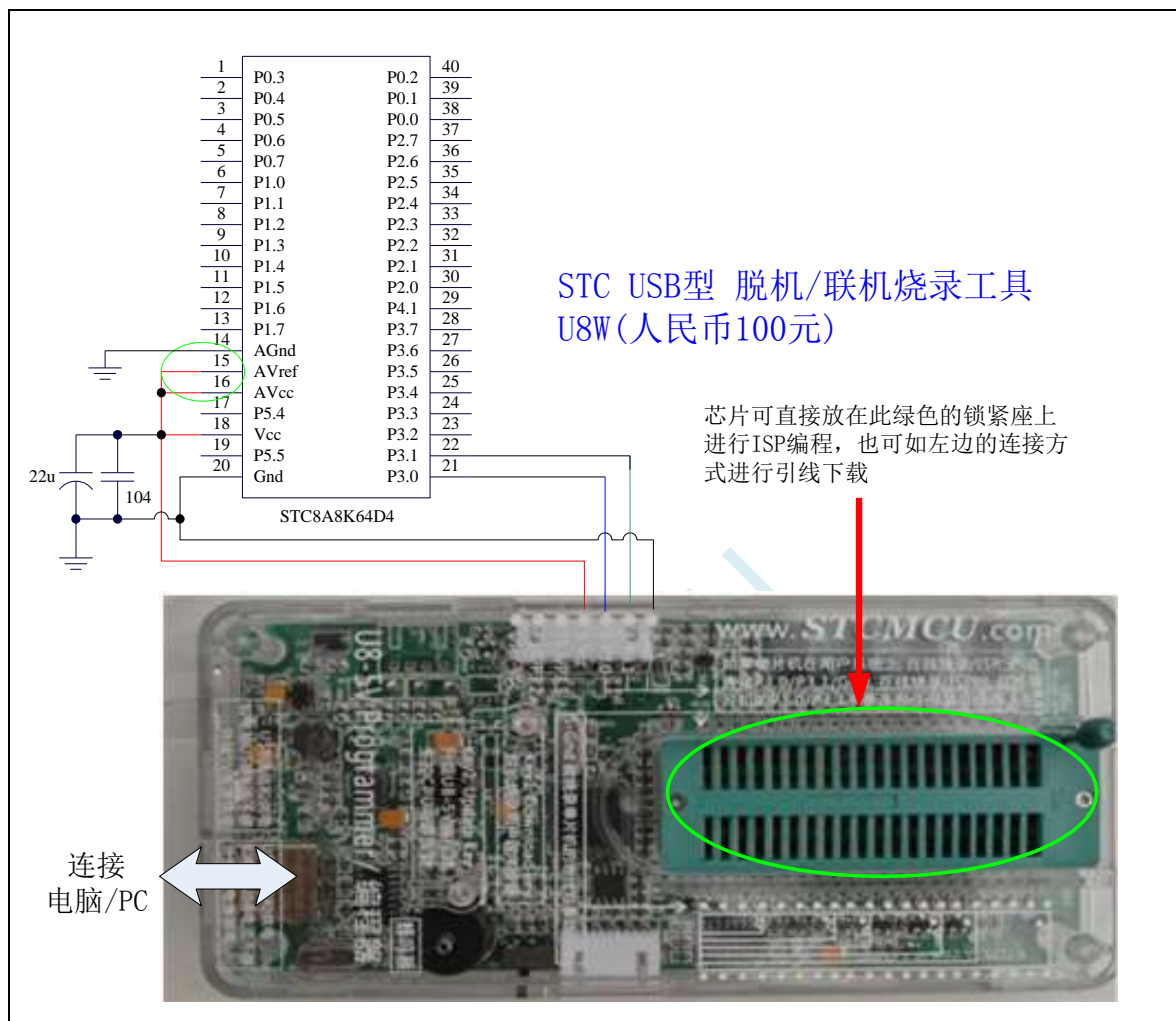
**注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。**

若要使用 U8-Mini 进行仿真，首先必须将 U8-Mini 设置为直通模式。U8W/U8W-Mini 实现 USB 转串口直通模式的方法如下：

- 1、首先 U8W/U8W-Mini 固件必须升级到 v1.37 及以上版本
- 2、U8W/U8W-Mini 上电后为正常下载模式，此时按住工具上的 Key1（下载）按键不要松开，再按一下 Key2（电源）按键，然后松开 Key2（电源）按键，再松开 Key1（下载）按键，U8W/U8W-Mini 会进入 USB 转串口直通模式。（按下 Key1 → 按下 Key2 → 松开 Key2 → 松开 Key1）
- 3、进入直通模式的 U8W/U8W-Mini 工具只是简单的 USB 转串口不具备脱机下载功能，若需要恢复 U8W/U8W-Mini 的原有功能，只需要再次单独按一下 Key2（电源）按键 即可



### 5.1.9 使用 U8W 工具下载，支持 ISP 在线和脱机下载，也可支持仿真



#### ISP 下载步骤（连线方式）：

- 1、按照如图所示的连接方式将 U8W 和目标芯片连接
- 2、点击 STC-ISP 下载软件中的“下载/编程”按钮
- 3、开始 ISP 下载

注意：若是使用 U8W 给目标系统供电，目标系统的总电流不能大于 200mA，否则会导致下载失败。

#### ISP 下载步骤（在板方式）：

- 1、将芯片按照 1 脚靠近锁紧扳手、管脚向下靠齐的方向放置好目标芯片
- 2、点击 STC-ISP 下载软件中的“下载/编程”按钮
- 3、开始 ISP 下载

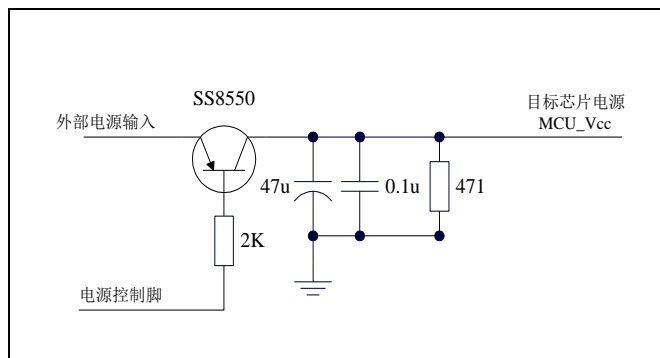
**注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。**

若要使用 U8W 进行仿真，首先必须将 U8W 设置为直通模式。U8W/U8W-Mini 实现 USB 转串口直通模式的方法如下：

- 1、首先 U8W/U8W-Mini 固件必须升级到 v1.37 及以上版本
- 2、U8W/U8W-Mini 上电后为正常下载模式，此时按住工具上的 Key1（下载）按键不要松开，再按一下 Key2（电源）按键，然后放开 Key2（电源）按键 后，再松开 Key1（下载）按键，U8W/U8W-Mini 会进入 USB 转串口直通模式。（按下 Key1 → 按下 Key2 → 松开 Key2 → 松开 Key1）
- 3、进入直通模式的 U8W/U8W-Mini 工具只是简单的 USB 转串口不具备脱机下载功能，若需要恢复 U8W/U8W-Mini 的原有功能，只需要再次单独按一下 Key2（电源）按键 即可

STC MCU

### 5.1.10 单机电源控制参考电路



## 5.2 STC-ISP 下载软件高级应用

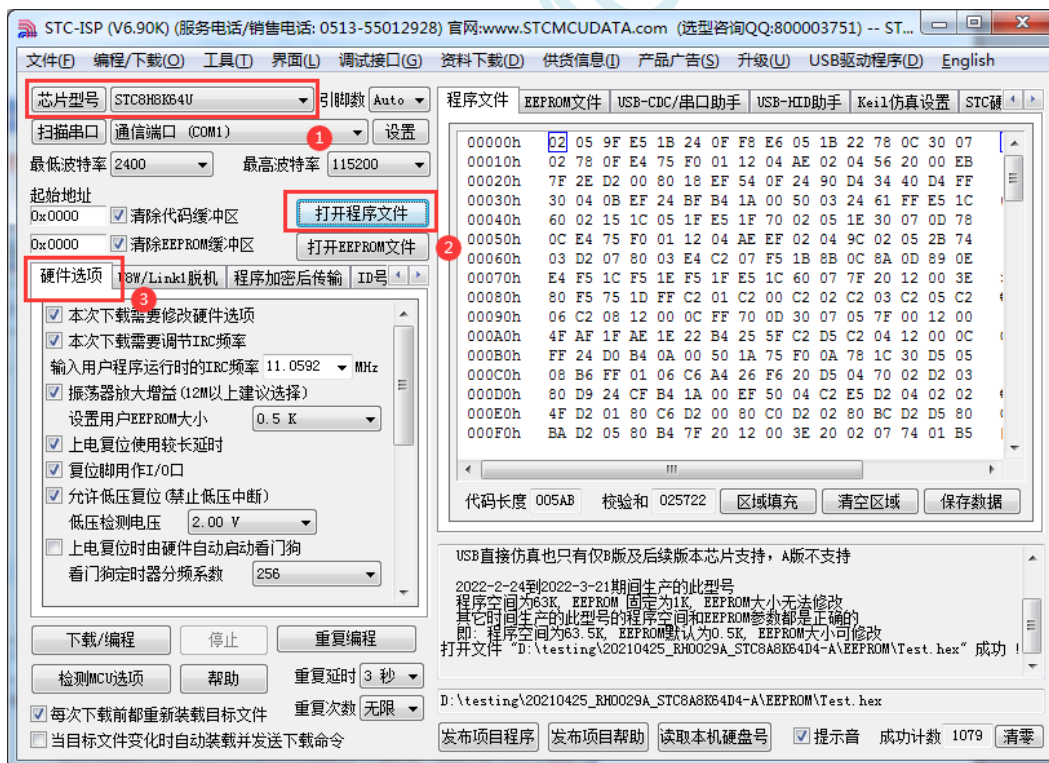
### 5.2.1 发布项目程序

发布项目程序功能主要是将用户的程序代码与相关的选项设置打包成为一个可以直接对目标芯片进行下载编程的超级简单的用户自己界面的可执行文件。

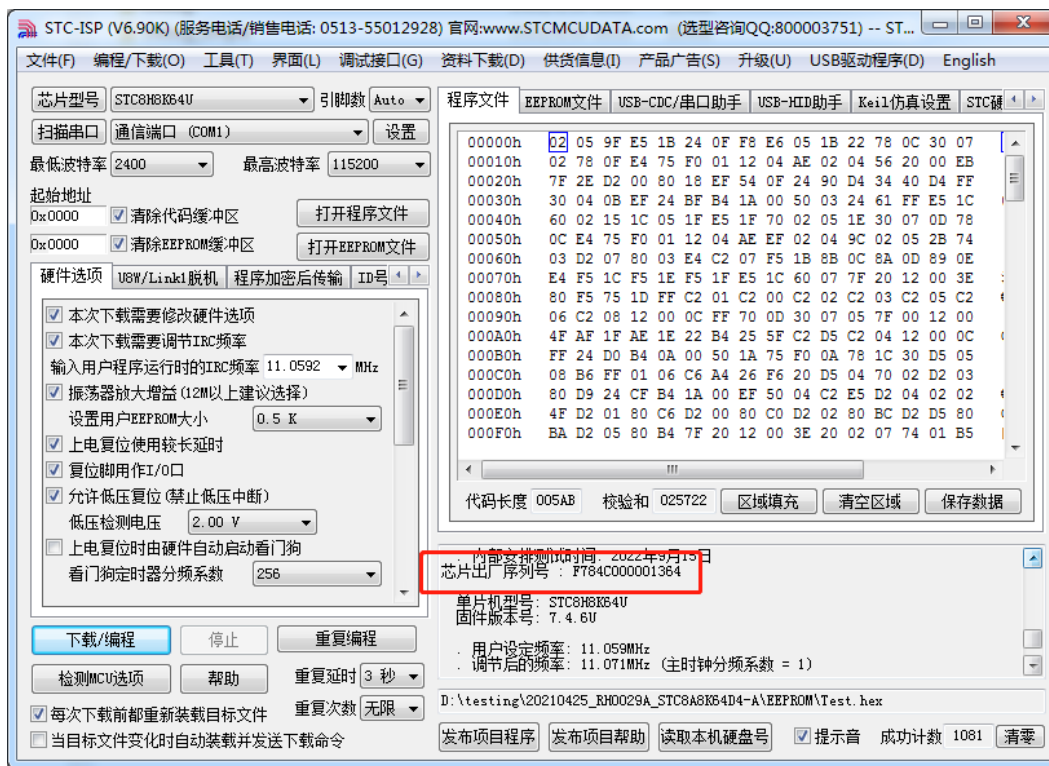
关于界面, 用户可以自己进行定制(用户可以自行修改发布项目程序的标题、按钮名称以及帮助信息), 同时用户还可以指定目标电脑的硬盘号和目标芯片的 ID 号, 指定目标电脑的硬盘号后, 便可以控制发布应用程序只能在指定的电脑上运行(防止烧录人员将程序轻易从电脑盗走, 如通过网络发走, 如通过 U 盘烤走, 防不胜防, 当然盗走你的电脑那就没办法那, 所以 STC 的脱机下载工具比电脑烧录安全, 能限制可烧录芯片数量, 让前台文员小姐烧, 让老板娘烧都可以), 拷贝到其它电脑, 应用程序不能运行。同样的, 当指定了目标芯片的 ID 号后, 那么用户代码只能下载到具有相应 ID 号的目标芯片中(对于一台设备要卖几千万的产品特别有用---坦克, 可以发给客户自己升级, 不需冒着生命危险跑到战火纷飞的伊拉克升级软件啦), 对于 ID 号不一致的其它芯片, 不能进行下载编程。

发布项目程序详细的操作步骤如下:

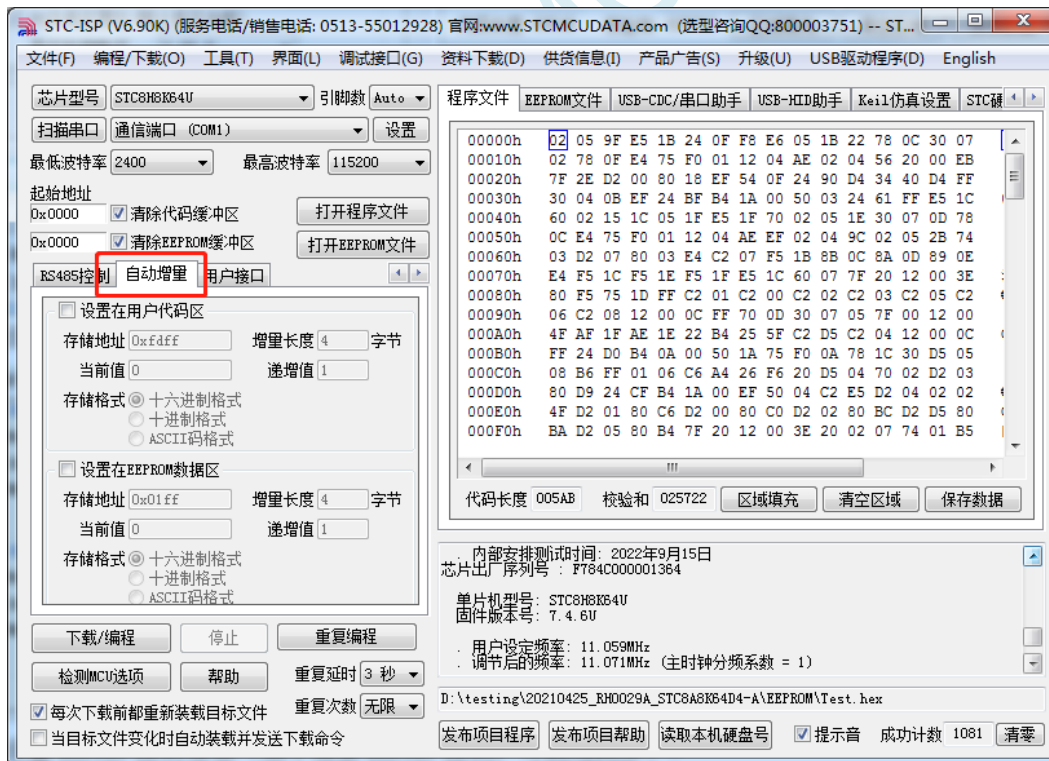
- 1、首先选择目标芯片的型号
- 2、打开程序代码文件
- 3、设置好相应的硬件选项



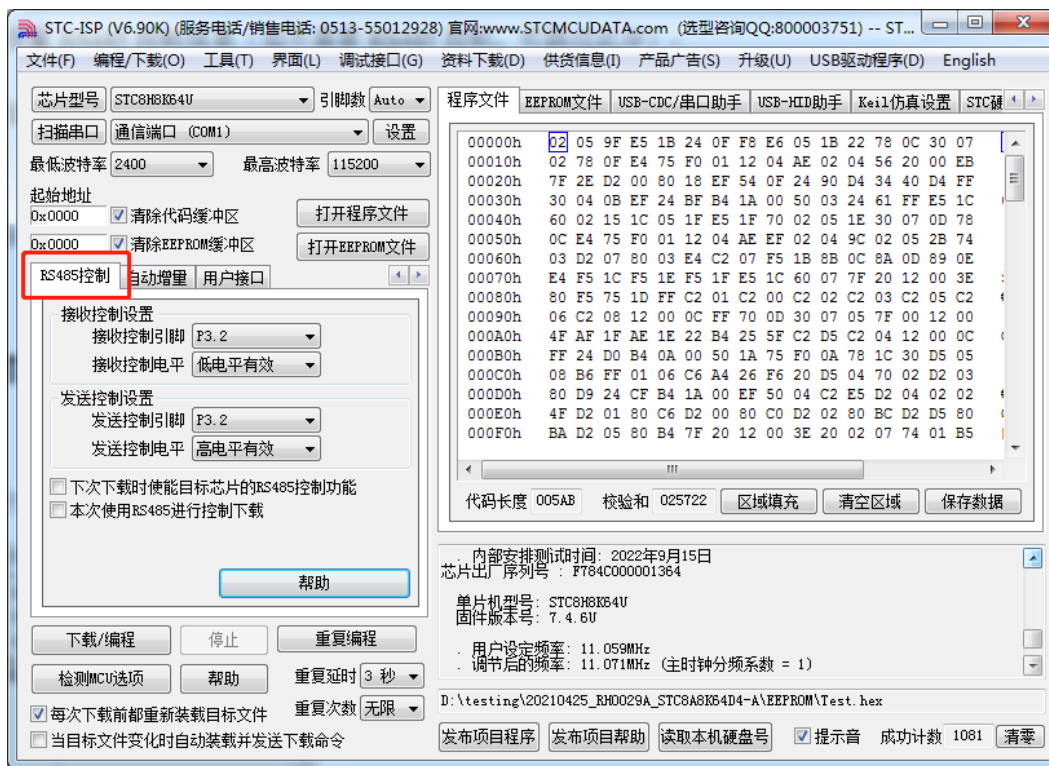
- 4、试烧一下芯片, 并记下目标芯片的 ID 号, 如下图所示, 该芯片的 ID 号即为“F784C00001364” (如不需要对目标芯片的 ID 号进行校验, 可跳过此步)



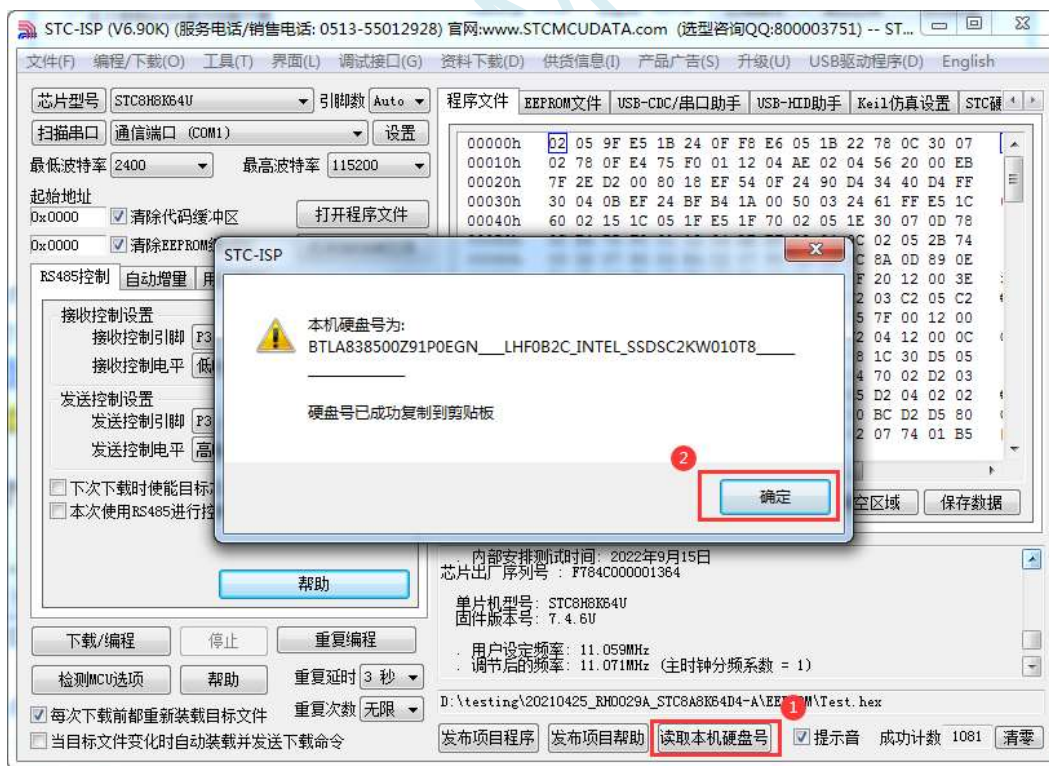
5、设置自动增量（如不需要自动增量，可跳过此步）



6、设置 RS485 控制信息（如不需要 RS485 控制，可跳过此步）



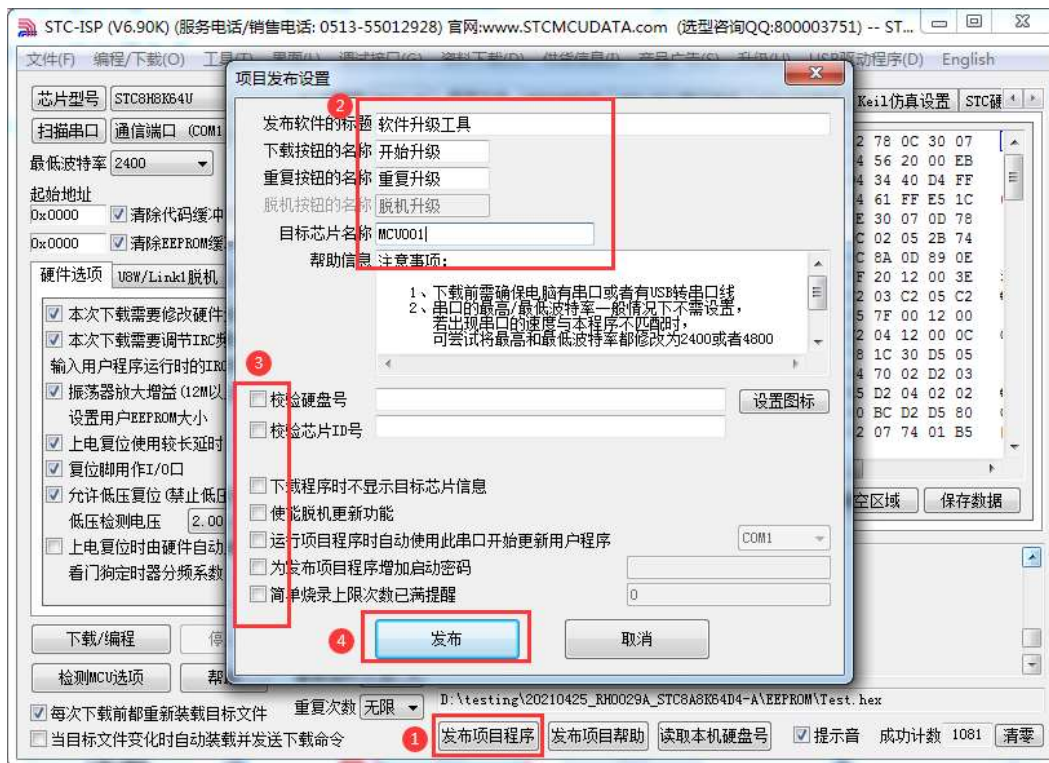
- 7、点击界面上的“读取本机硬盘号”按钮，并记下目标电脑的硬盘号（如不需要对目标电脑的硬盘号进行校验，可跳过此步）



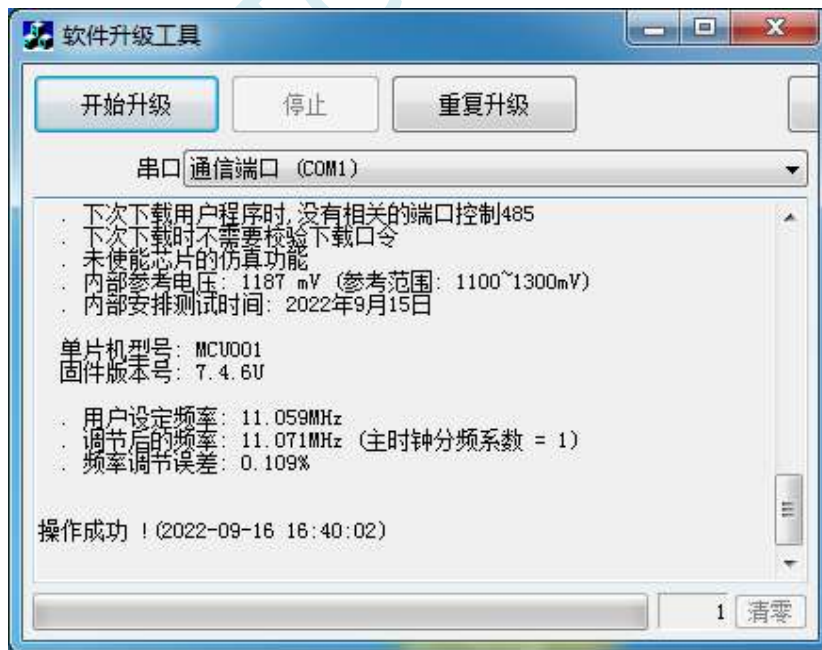
- 8、点击“发布项目程序”按钮，进入发布应用程序的设置界面。
- 9、根据各自的需要，修改发布软件的标题、下载按钮的名称、重复下载按钮的名称、自动增量的名称以及帮助信息
- 10、若需要校验目标电脑的硬盘号,则需要勾选上“校验硬盘号”，并在后面的文本框内输入前面所记

下的目标电脑的硬盘号

- 11、若需要校验目标芯片的 ID 号，则需要勾选上“校验芯片 ID 号”，并在后面的文本框内输入前面所记下的目标芯片的 ID 号



- 12、最后点击发布按钮，将项目发布程序保存，即可得到相应的可执行文件。发布的项目程序打界面如下图



## 5.2.2 程序加密后传输（防烧录时串口分析出程序）

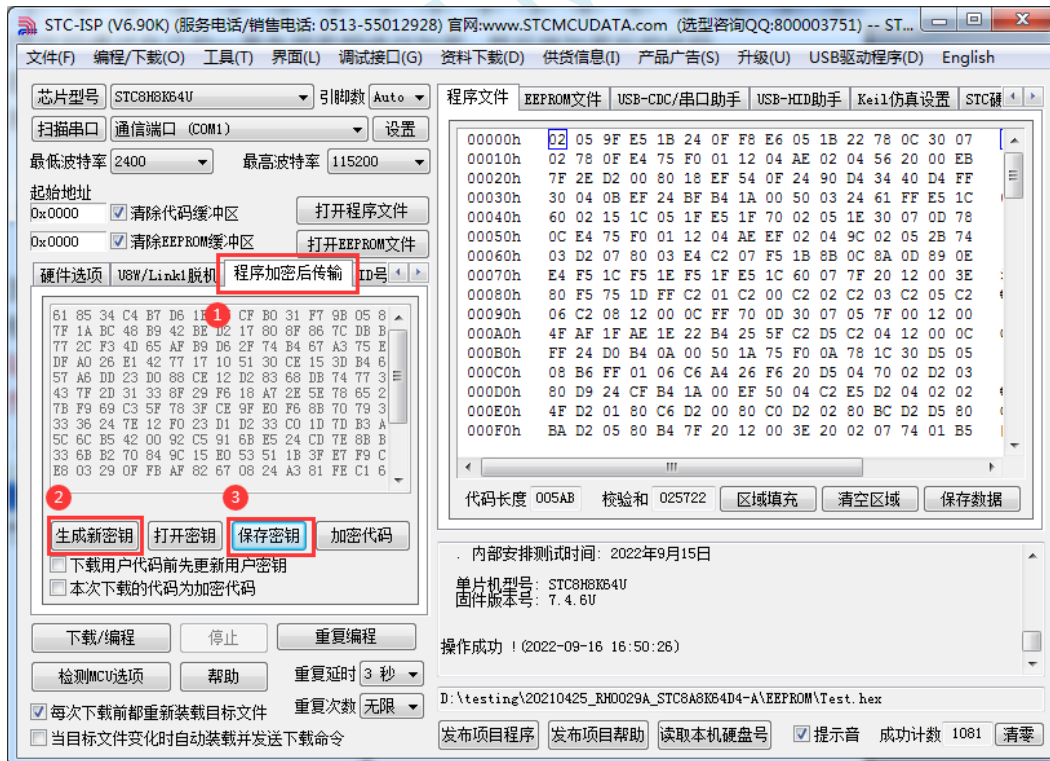
目前，所有的普通串口下载烧录编程都是采用**明码通信**的（电脑和目标芯片通信时，或脱机下载板和目标芯片通信时），问题：如果烧录人员通过分析下载烧录编程时串口通信的数据，高手是可以在烧录时在串口上引 2 根线出来，通过分析串口通信的数据分析出实际的用户程序代码的。当然用 **STC 的脱机下载板烧程序总比用电脑烧程序强**（防止烧录人员将程序轻易从电脑盗走，如通过网络发走，如通过 U 盘拷走，防不胜防，当然盗走你的电脑那就没办法那，所以 **STC 的脱机下载工具比电脑烧录安全，让前台文员小姐烧，让老板娘烧都可以**）。即使是 **STC 全球首创的脱机下载工具**，对于要防止天才的不法分子在脱机下载工具烧录的过程中通过分析串口通信的数据分析出实际的用户程序代码，也是没有办法达到要求的，这就需要用到最新的 **STC 单片机所提供的程序加密后传输功能**。

程序加密后传输下载是用户先将程序代码通过自己的一套专用密钥进行加密，然后将加密后的代码再通过串口下载，此时下载传输的是加密文件，通过串口分析出来的是加密后的乱码，如不通过派人潜入你公司盗窃你电脑里面的加密密钥，就无任何价值，便可起到防止在烧录程序时被烧录人员通过监测串口分析出代码的目的。

程序加密后传输功能的使用需要如下的几个步骤：

### 1、生成并保存新的密钥

如下图，进入到“程序加密后传输”页面，点击“生成新密钥”按钮，即可在缓冲区显示新生成的 256 字节的密钥。然后点击“保存密钥”按钮，即可将生成的新密钥保存为以“.K”为扩展名的的密钥文件（**注意：这个密钥文件一定要保存好，以后发布的代码文件都需要使用这个密钥加密，而且这个密钥的生成是非重复的，即任何时候都不可能生成两个完全相同的密钥，所以一旦密钥文件丢失将无法重新获得**）。例如我们将密钥保存为“New.k”。



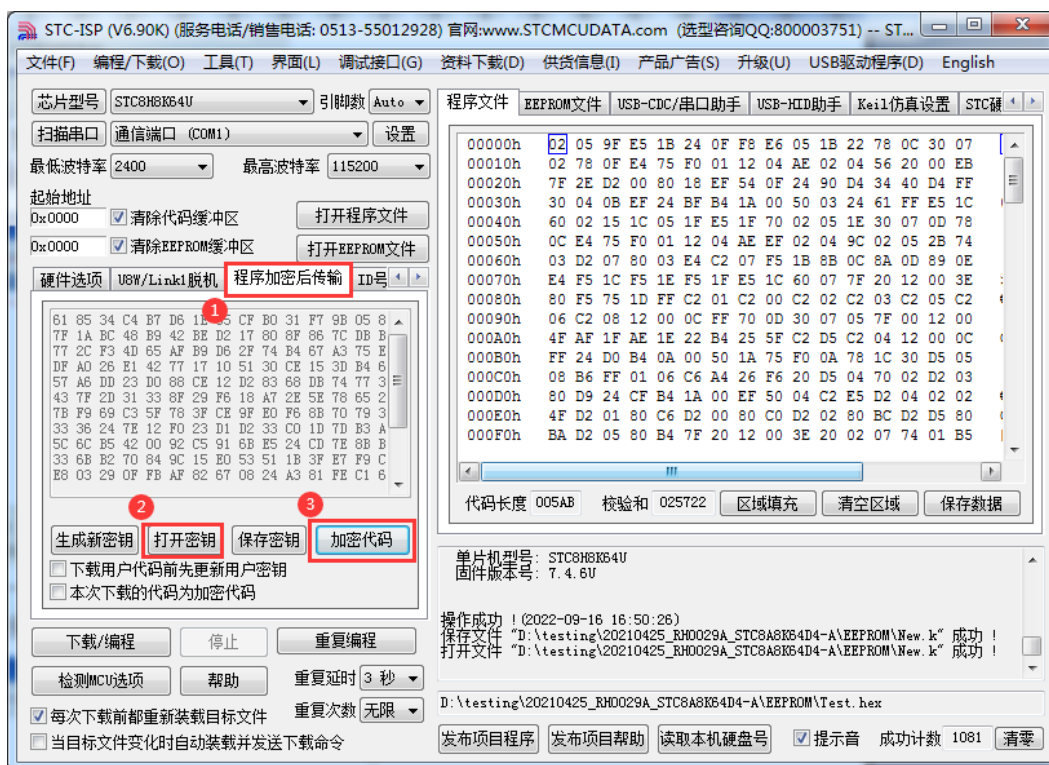
### 2、对代码文件加密

加密文件前，需要先打开我们自己的密钥。若缓冲区中存放的已经是我们的密钥，则不要再打开。

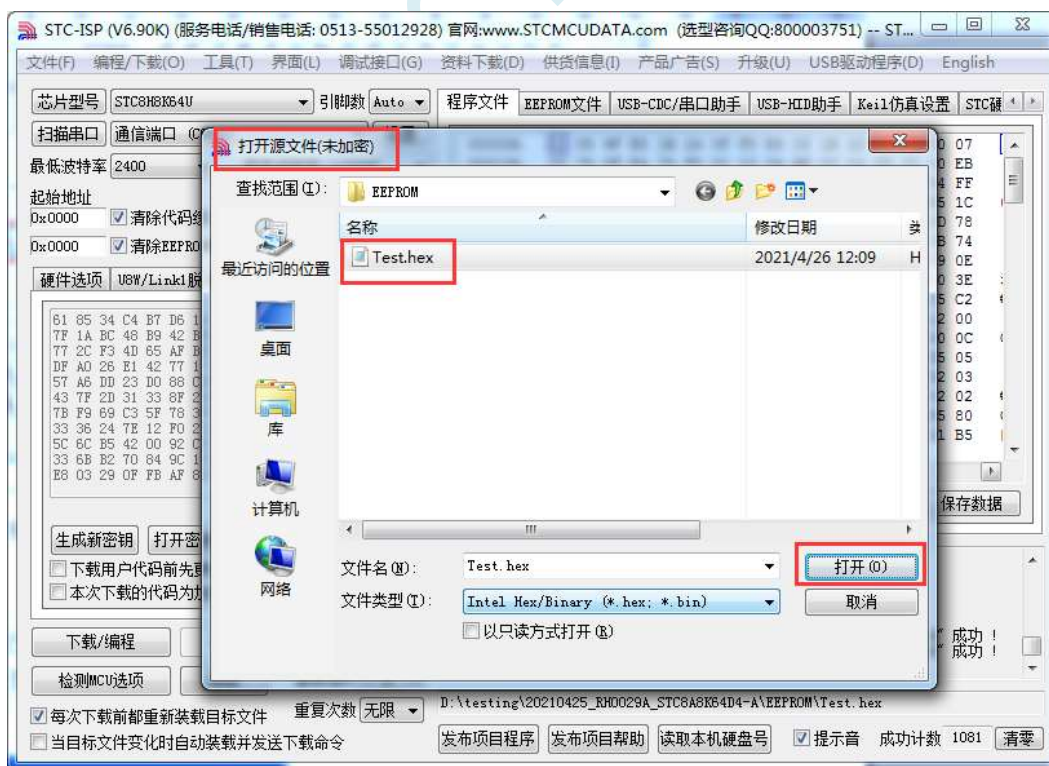
如下图，在“程序加密后传输”页面中点击“打开密钥”按钮，打开我们之前保存的密钥文件，

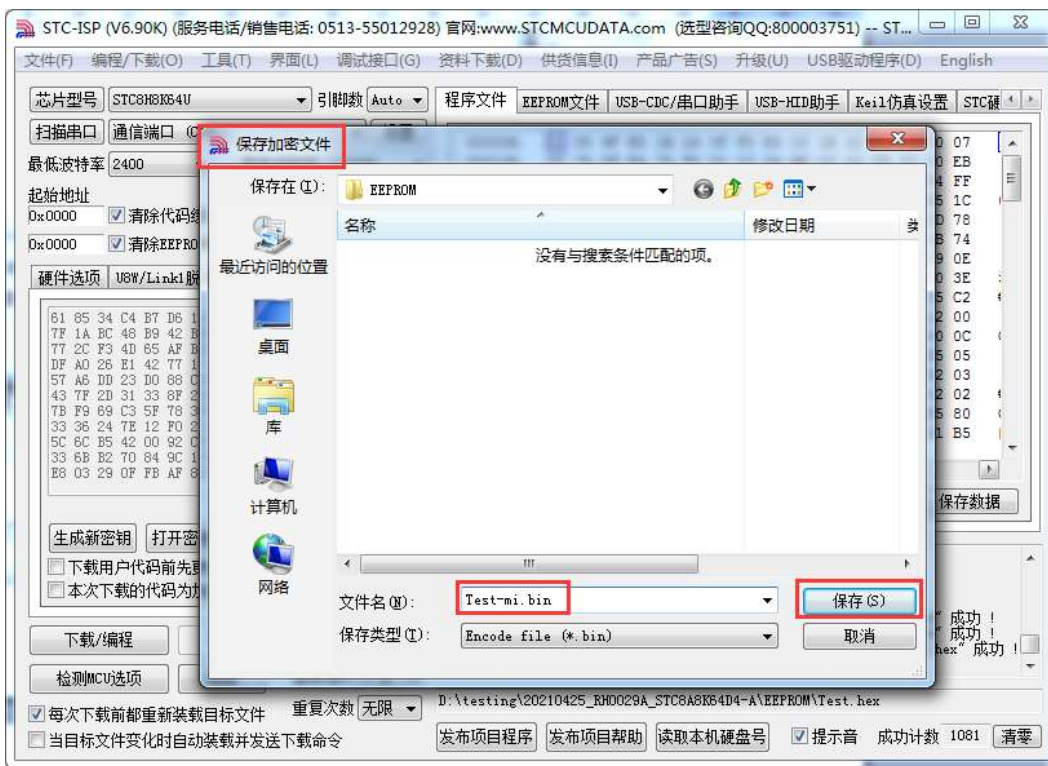


例如“New.k”，然后返回到“程序加密后传输”页面中点击“加密代码”按钮，如下图所示，首先会弹出“打开源文件（未加密）”的对话框，此时选择的是原始的未加密的代码文件



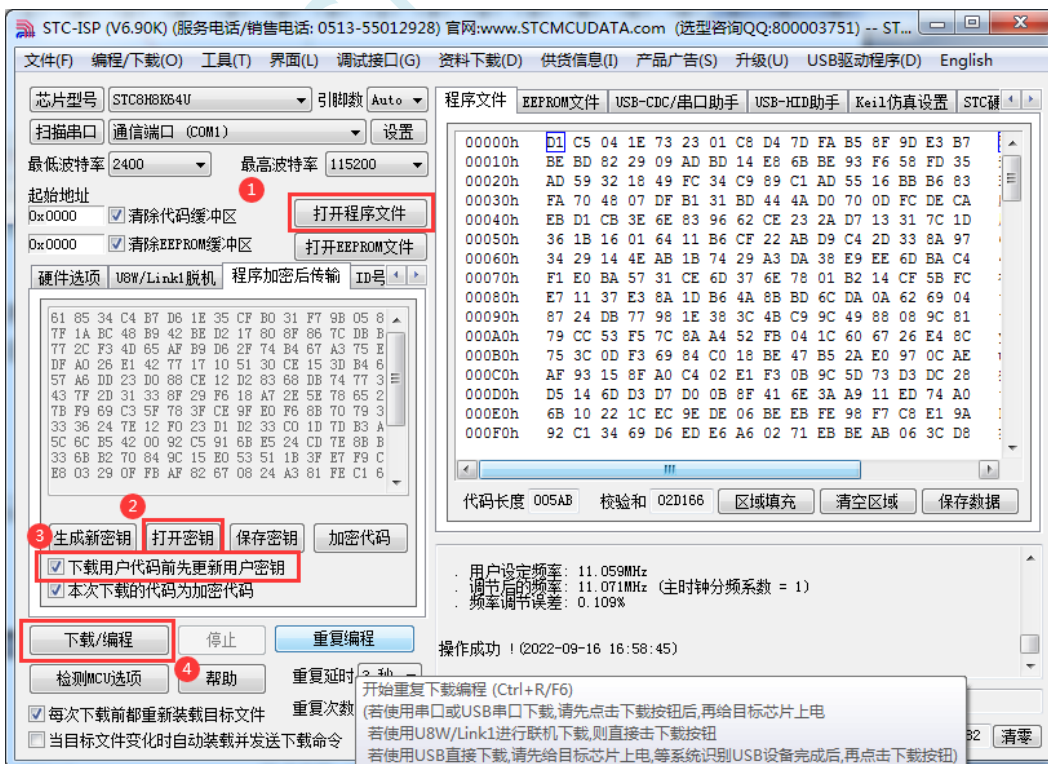
点击打开按钮后，马上会有会弹出一个类似的对话框，但此时是对加密后的文件进行保存的对话框。如下图所示，点击保存按钮即可保存加密后的文件。





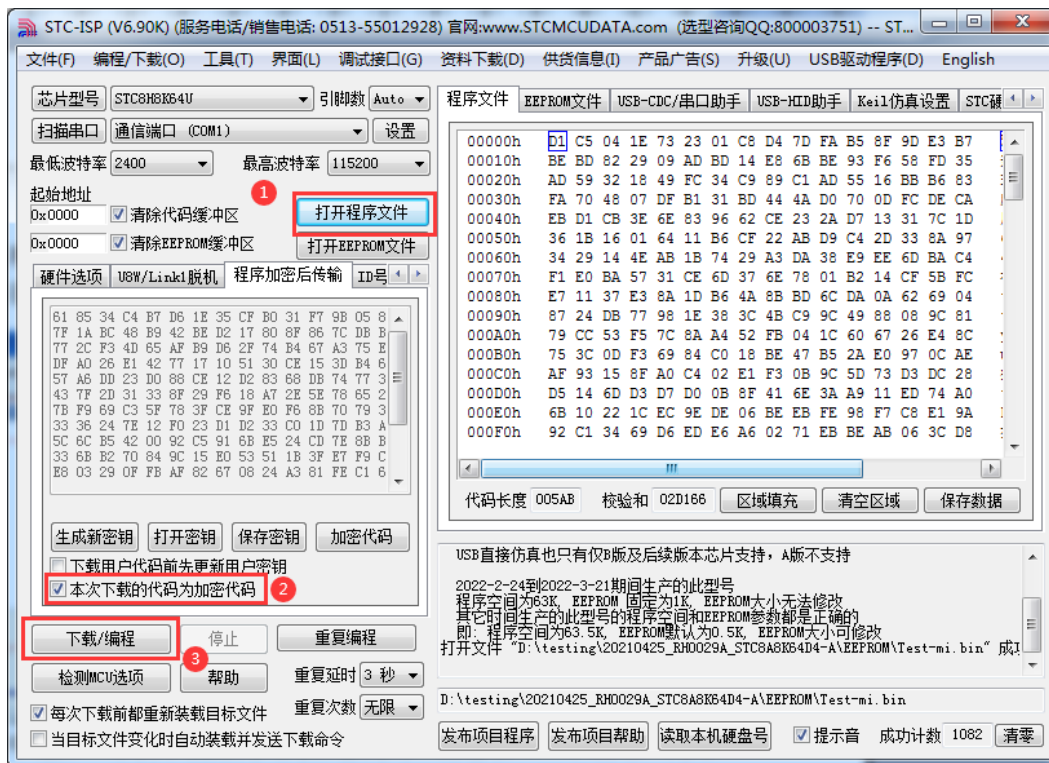
3、将用户密钥更新到目标芯片中

更新密钥前，需要先打开我们自己的密钥。若缓冲区中存放的已经是我们的密钥，则不要再打开。如下图，在“自定义加密下载”页面中点击“打开密钥”按钮，打开我们之前保存的密钥文件，例如“New.k”。密钥打开后，如下图所示，勾选上“下载用户代码前先更新用户密钥”选项和“本次下载的代码为加密代码”的选项，然后打开我们之前加密过后的文件，打开后点击界面左下角的“下载/编程”按钮，按正常方式对目标芯片下载完成即可更新用户密钥。



#### 4、加密更新用户代码

密钥更新成功后, 目标芯片便具有接收加密代码并还原的功能。此时若需要再次升级/更新代码, 则只需要参考第二步的方法, 将目标代码进行加密, 然后如下图



对于一片新的 STC 单片机, 可将步骤 3 和步骤 4 合并完成, 即将密钥更新到目标单片机的同时也可将加密后的代码一并下载到单片机中, 若已经执行过步骤 3 (即已经将密钥更新到目标芯片中了), 则后续的代码更新就只需要按照步骤 4, 只需要在“程序加密后传输”页面中选择“本次下载的代码为加密代码”的选项 (“下载用户代码前先更新用户密钥”选项不需要选了), 然后打开我们之前加过加密后的文件, 打开后点击界面左下角的“下载/编程”按钮, 按正常方式对目标芯片下载即可完成用用户自己专用的加密文件更新用户代码的目的 (防止在烧录程序时被烧录人员通过监测串口分析出代码的目的)。

### 5.2.3 发布项目程序+程序加密后传输结合使用

发布项目程序与程序加密后传输两项新的特殊功能可以结合在一起使用。首先程序加密后传输可以确保用户代码在烧录编程时串口通信传输过程当中的保密性，而发布项目程序可实现让最终使用者远程升级功能（方案公司的人员不需要亲自到场）。所以两项功能结合起来使用，非常适用于方案公司/生产商在软件需要更新时，让最终使用者自己对终端产品进行软件更新的目的，又确保现场烧录人员无法通过串口分析出有用程序，强烈建议方案公司使用。

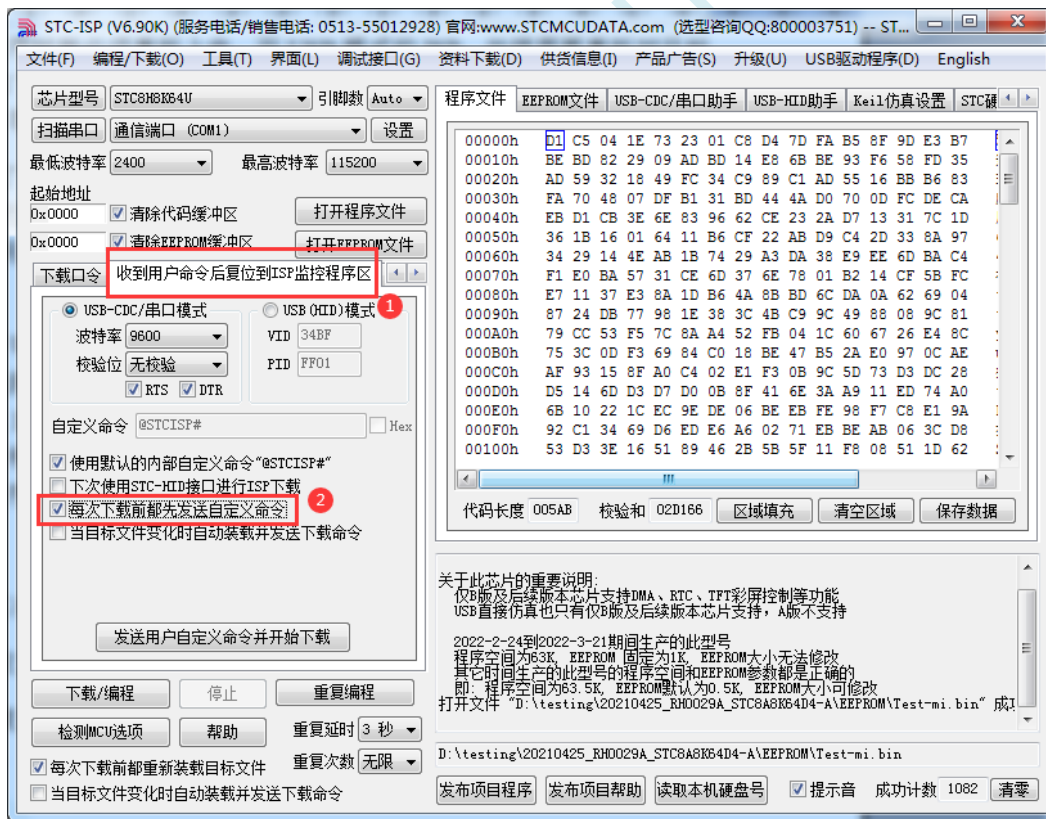
STC MCU

## 5.2.4 用户自定义下载（实现不停电下载）

将用户的目标程序下载到STC单片机是通过执行单片机内部的ISP系统代码和上位机进行串口或者USB通讯来实现的。但STC单片机内部的ISP系统代码只有在每次重新停电再上电时才会被执行，这就要求用户每次需要对目标单片机更新程序时就必须重新上电，而USB模式的ISP，处理需要重新对目标芯片上电外，还需要在上电时将P3.2口下拉到GND。对于处于开发阶段的项目，需要频繁的修改代码、更新代码，每次下载都需要重新上电会导致操作非常麻烦。

STC单片机在硬件设计时，增加了一个软复位寄存器（IAP\_CONTR），让用户可以通过设置此寄存器来决定CPU复位后重新执行用户代码还是复位到ISP区执行ISP系统代码。当向IAP\_CONTR寄存器写入0x20时，CPU复位后重新执行用户代码；当向IAP\_CONTR寄存器写入0x60时，CPU复位后复位到ISP区执行ISP系统代码。

要实现不停电进行ISP下载，用户可以在程序中设计一段代码，例如检测一个特殊的按键、或者监控串口等待一个特殊的串口命令，当检测到满足下载条件时，就通过软件触发软复位寄存器复位到ISP区执行ISP系统代码，从而实现不停电ISP下载。当触发条件是外部按键时，则在用户代码中实时监控按键状态即可。若要实现STC-ISP软件和用户触发软复位完全同步，则需要使用STC-ISP软件中所提供的“收到用户命令后复位到ISP监控程序区”这个功能。



实现不停电 ISP 下载的步骤如下:

1、编写用户代码，并在用户代码中添加串口命令监控程序

(参考代码如下，测试单片机型号为 STC8H8K64U)

```
#include "stc8h.h"

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

char code *STCISPCMD = "@STCISP#"; //自定义下载命令
char index;

void uart_isr() interrupt 4
{
    char dat;

    if (TI)
    {
        TI = 0;
    }

    if (RI)
    {
        RI = 0;
        dat = SBUF; //接收串口数据

        if (dat == STCISPCMD[index]) //判断接收的数据和当前的命令字符是否匹配
        {
            index++; //若匹配则索引+1
            if (STCISPCMD[index] == '\0') //判断命令是否配完成
                IAP_CONTR = 0x60; //若匹配完成则软复位到 ISP
        }
        else
        {
            index = 0; //若不匹配,则需要从头开始
            if (dat == STCISPCMD[index])
                index++;
        }
    }
}

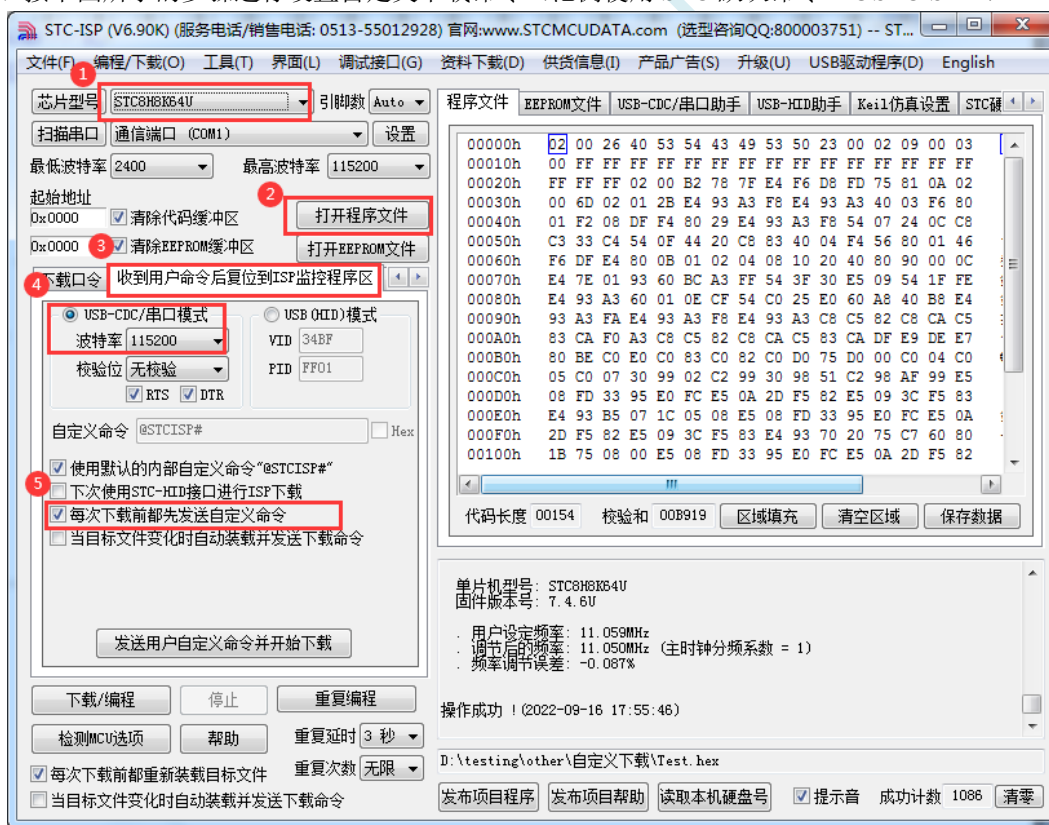
void main()
{
    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
```

```
P2M0 = 0x00; P2M1 = 0x00;
P3M0 = 0x00; P3M1 = 0x00;
```

```
SCON = 0x50; //串口初始化
AUXR = 0x40;
TMOD = 0x00;
TH1 = BAUD >> 8;
TL1 = BAUD;
TR1 = 1;
ES = 1;
EA = 1;
```

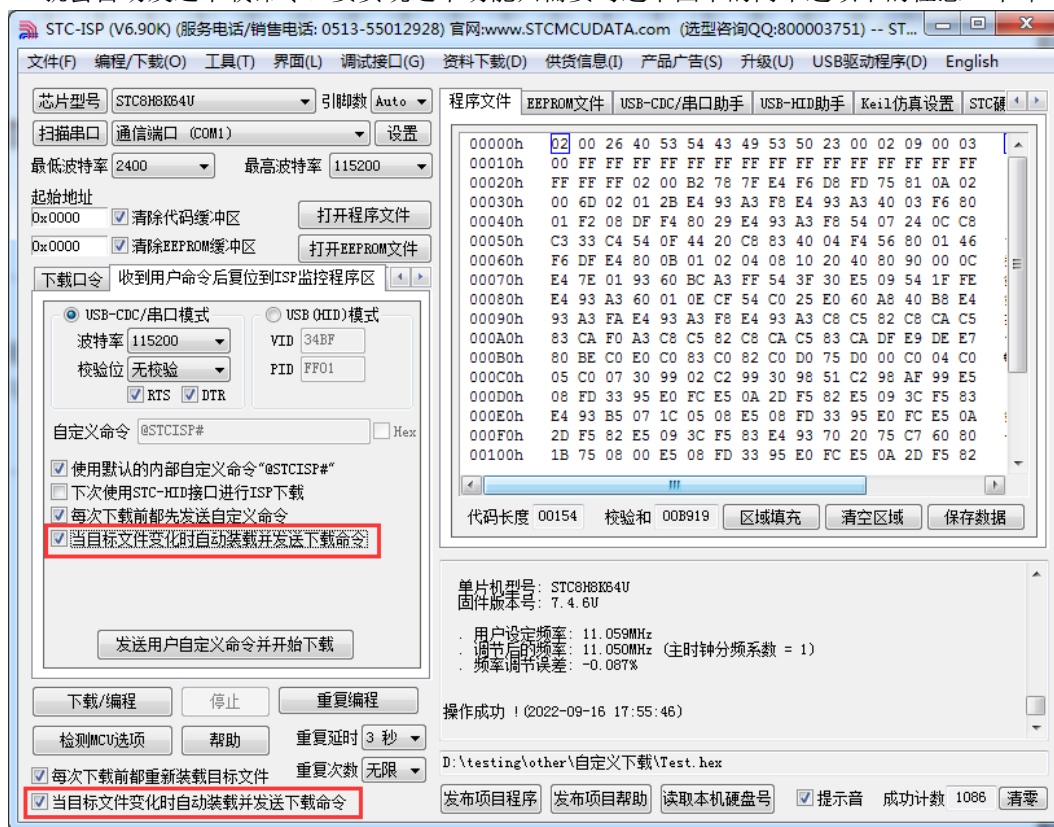
```
index = 0; //初始化命令
while (1);
}
```

## 2、按下图所示的步骤进行设置自定义下载命令（范例使用 STC 默认命令 “@STCISP#”）



## 3、第一次下载时需要为目标单片机重新上电，之后的每次更新只需要点击下载软件中的“下载/编程”按钮，下载软件自动将下载命令发送给目标单片机，目标单片机接收到命令后自动复位到系统 ISP 区，即可实现不停电更新用户代码。

4、STC-ISP 还可实现项目开发阶段，完全自动下载功能，即当下载软件侦测到目标代码被更新了，就会自动发送下载命令。要实现这个功能只需要勾选下图中的两个选项中的任意一个即可



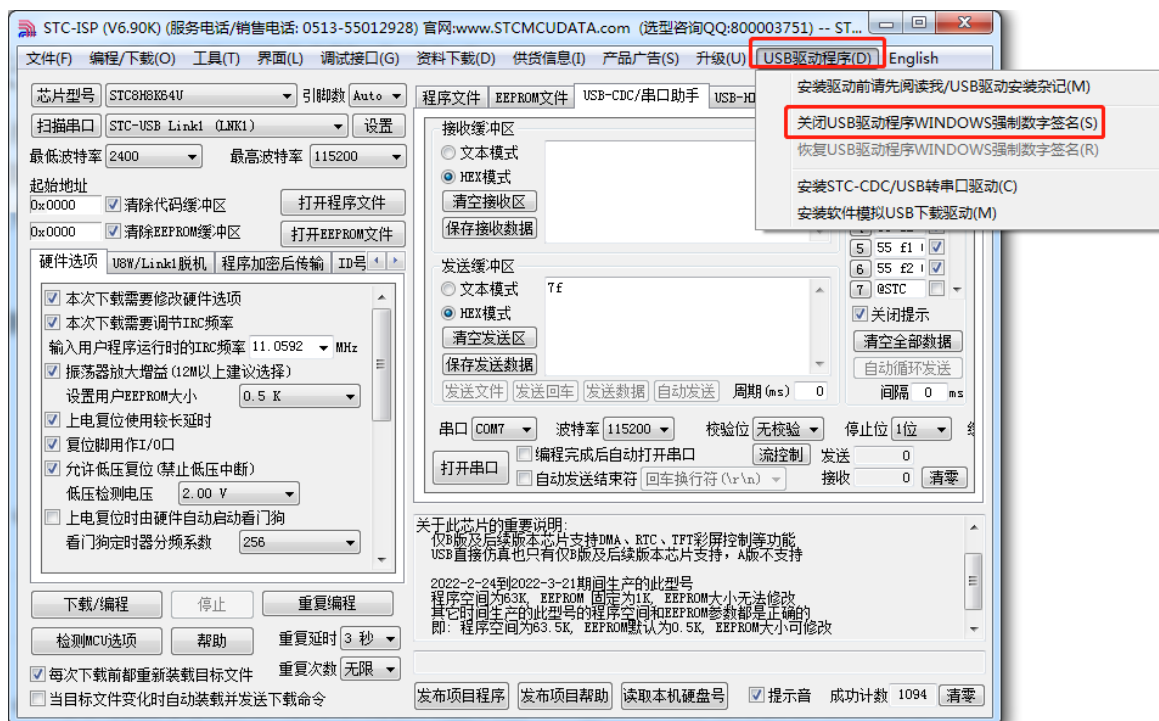


## 5.3 驱动程序数字签名的相关说明

### 5.3.1 关于驱动程序强制数字签名

Windows 系统从 Vista 版本开始的 64 位 Windows 系统就对驱动程序强制需要数字签名才能安装, 否则驱动无法安装, USB 设备也无法正常使用。目前 STC 提供的驱动程序暂时没有做 WHQL 认证(微软徽标认证), 在没有关闭驱动程序强制数字签名前, 在从 Vista 系统开始的 64 位 Windows 系统中可能无法安装成功。

STC-ISP 最新的下载软件提供了一个简单关闭驱动程序强制数字签名的功能, 菜单目录如下图所示:

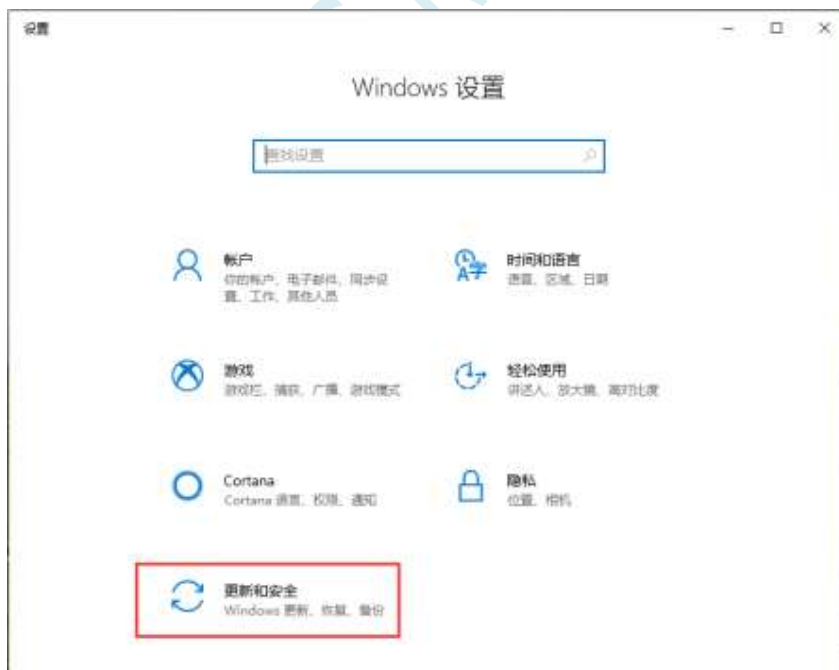
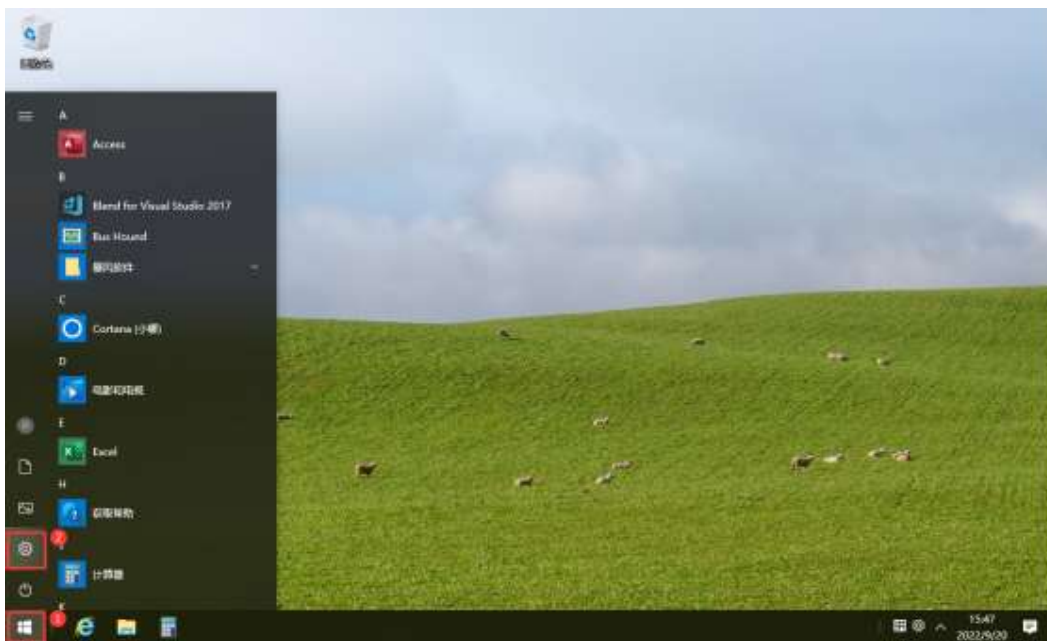


本菜单项是使用系统的 BCDEDIT 工具自动设置的 TestSigning 和 NoIntegrityChecks 两项参数以达到临时关闭驱动程序强制数字签名的目的。对于使能了安全启动的系统会无法使用 BCDEDIT 工具, 需要在 BIOS 中将安全启动关闭 (将 Secure Boot 设置为 Disable)

目前测试 WinXP/Win7-32/Win7-64 系统均可正常关闭数字签名, 但 Win10 及后续系统无效。要在 Win10 及后续系统中关闭数字签名仍然需要从系统启动菜单中选择禁止强制数字签名来实现。详细步骤请参考下一小节

### 5.3.2 Windows 10 关闭驱动程序强制数字签名步骤

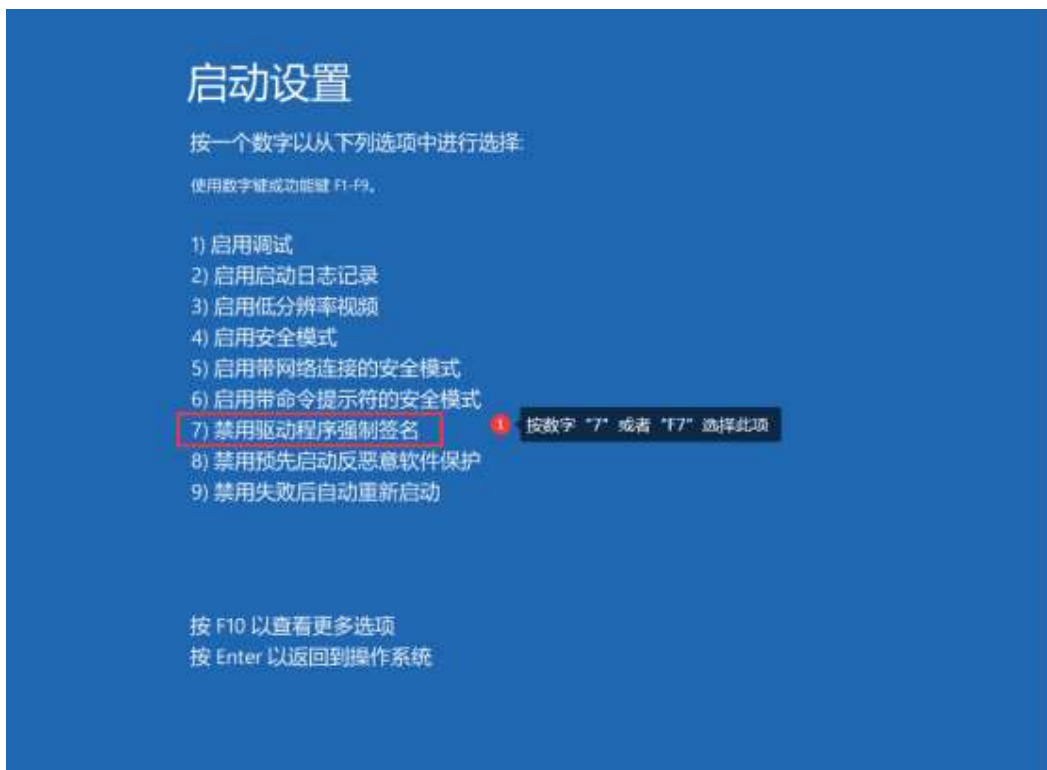
在“开始”菜单中选择“设置”功能











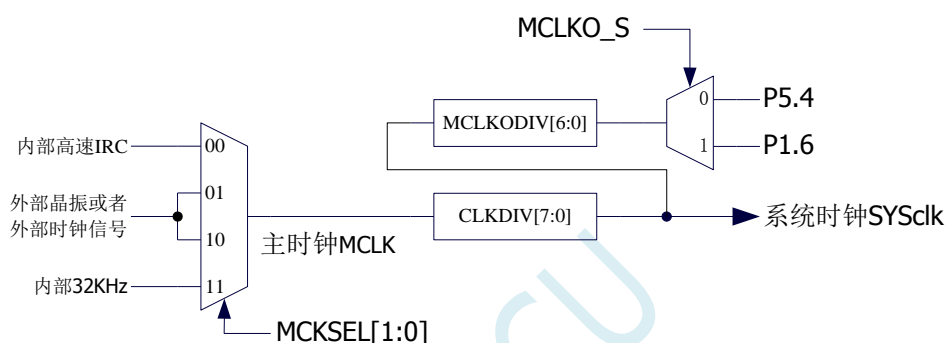
系统重新启动后，驱动程序的强制数字签名即可临时被关闭

## 6 时钟、复位、省电模式与系统电源管理

### 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		
CLKSEL	时钟选择寄存器	FE00H	-								MCKSEL[1:0]	xxxx,xx00
CLKDIV	时钟分频寄存器	FE01H	-									nnnn,nnnn
HIRCCR	内部高速振荡器控制寄存器	FE02H	ENHIRC	-	-	-	-	-	-	-	HIRCST	1xxx,xxx0
XOSCCR	外部晶振控制寄存器	FE03H	ENXOSC	XITYPE	XCFILTER[1:0]	NMXCG	-	-	-	XOSCST	0000,0xx0	
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	
IRCDDB	内部 IRC 起振去抖控制	FE06H	IRCDDB[7:0]									1000,0000

#### 6.1.1 系统时钟选择寄存器 (CLKSEL)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0	
CLKSEL	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

## 6.1.2 时钟分频寄存器 (CLKDIV)

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

注意: 用户程序复位后, 系统会自动根据上次 ISP 下载时所设定工作频率所需的分频系数来设置此寄存器的初始值

## 6.1.3 内部高速高精度 IRC 控制寄存器 (HIRCCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
HIRCCR	FE02H	ENHIRC	-	-	-	-	-	-	HIRCST

ENHIRC: 内部高速高精度 IRC 使能位

- 0: 关闭内部高精度 IRC
- 1: 使能内部高精度 IRC

HIRCST: 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 HIRCST 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 ENHIRC=1 使能振荡器, 然后一直查询振荡器稳定标志位 HIRCST, 直到标志位变为 1 时, 才可进行时钟源切换。

## 6.1.4 外部振荡器控制寄存器 (XOSCCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
XOSCCR	FE03H	ENXOSC	XITYPE	XCFILTER[1:0]	NMXCG	-	-	-	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) (此时 P1.6 口固定为高阻输入模式, 可用于读取外部数字信号或当作 ADC 输入, 但一般不建议使用, 因为旁边的 P1.7 口有高频振荡信号会对 P1.6 的信号有影响)
- 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。

XCFILTER[1:0]: 外部晶体振荡器抗干扰控制寄存器

00: 外部晶体振荡器频率在 48M 及以下时可选择此项

01: 外部晶体振荡器频率在 24M 及以下时可选择此项

1x: 外部晶体振荡器频率在 12M 及以下时可选择此项

NMXCG: 外部晶体振荡器增益控制

0: 低增益

1: 高增益

## 6.1.5 内部 32KHz 低速 IRC 控制寄存器 (IRC32KCR)

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。

## 6.1.6 主时钟输出控制寄存器 (MCLKOCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
MCLKOCR	FE05H	MCKO_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

MCKO\_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 STC8A8K64D4 系列内部 IRC 频率调整

STC8A8K64D4 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%。

STC8A8K64D4 系列内部 IRC 有 4 个频段, 频段的中心频率分别为 6MHz、10MHz、27MHz 和 44MHz, 6M 频段的调节范围约为 4.5MHz~8MHz, 10M 频段的调节范围约为 7.5MHz~13.5MHz, 27M 频段的调节范围约为 20.5MHz~36MHz, 44M 频段的调节范围约为 29MHz~55MHz (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)。建议用户在 ISP 下载时设置 IRC 频率不要高于 45MHz。

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 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。

若用户需要在自己的代码中动态选择芯片预置的频率, 请参考预置频率列表以及“用户自定义内部 IRC 频率”的范例程序

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[1:0]	0000,00nn
LIRTRIM	IRC 频率微调寄存器	9EH	-	-	-	-	-	-	-	LIRTRIM	0000,000n
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]							nnnn,nnnn	

### 6.2.1 IRC 频段选择寄存器 (IRCBAND)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRCBAND	9DH	-	-	-	-	-	-	-	SEL[1:0]

SEL[1:0]: 频段选择

00: 选择 6MHz 频段

01: 选择 10MHz 频段

10: 选择 27MHz 频段

11: 选择 44MHz 频段

### 6.2.2 内部 IRC 频率调整寄存器 (IRTRIM)

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。

### 6.2.3 内部 IRC 频率微调寄存器 (LIRTRIM)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LIRTRIM	9EH	-	-	-	-	-	-	-	LIRTRIM

LIRTRIM: 内部高精度 IRC 频率微调寄存器

### 6.2.4 时钟分频寄存器 (CLKDIV)

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

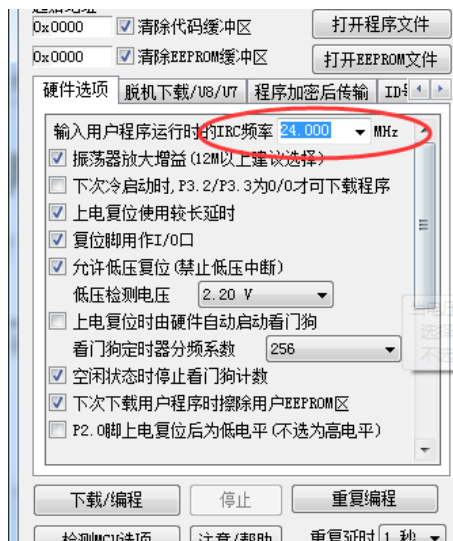
CLKDIV: 主时钟分频系数。系统时钟 SYSCLK 是对主时钟 MCLK 进行分频后的时钟信号。

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

## 6.2.5 分频出 3MHz 用户工作频率，并用用户动态改变频率追频示例

为得到 3MHz 的频率，可使用  $24\text{MHz} \div 8$  的方法。

首先在进行 ISP 下载时选择内部 IRC 工作频率为 24MHz，如下图所示，



然后在代码中选择时钟源为内部 IRC，并使用 CLKDIV 寄存器进行 8 分频。

### C 语言代码

```
//测试工作频率为24MHz
```

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

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

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

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

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

```

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

P_SW2 = 0x80;
CLKSEL = 0x00;           //选择内部IRC (默认)
CLKDIV = 0x08;          //时钟8分频
P_SW2 = 0x00;

IRTRIM++;               //IRC 频率向上3%进行微调 (注意判断边界)
// IRTRIM--;           //IRC 频率向下3%进行微调 (注意判断边界)

while (1);
}

```

## 汇编代码

;测试工作频率为24MHz

<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>
<i>IRTRIM</i>	<i>DATA</i>	<i>09FH</i>
<i>CLKSEL</i>	<i>EQU</i>	<i>0FE00H</i>
<i>CLKDIV</i>	<i>EQU</i>	<i>0FE01H</i>
<i>HIRCCR</i>	<i>EQU</i>	<i>0FE02H</i>
<i>XOSCCR</i>	<i>EQU</i>	<i>0FE03H</i>
<i>IRC32KCR</i>	<i>EQU</i>	<i>0FE04H</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>0100H</i>
<i>MAIN:</i>	<i>MOV</i>	<i>SP, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>
	<i>MOV</i>	<i>P1M1, #00H</i>

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

MOV    P_SW2, #80H
MOV    A, #00H                ;选择内部IRC
MOV    DPTR, #CLKSEL
MOVX   @DPTR, A
MOV    A, #08H                ;时钟8分频
MOV    DPTR, #CLKDIV
MOVX   @DPTR, A
MOV    P_SW2, #00H

INC    IRTRIM                ;IRC 频率向上3%进行微调 (注意判断边界)
DEC    IRTRIM                ;IRC 频率向下3%进行微调 (注意判断边界)

JMP    $

END
```

## 6.3 系统复位

STC8A8K64D4 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，POR，1.7V 附近
- 低压复位，LVD-RESET（2.0V，2.4V，2.7V，3.0V 附近）
- 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	
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

### 6.3.1 看门狗复位 (WDT\_CONTR)

在工业控制/汽车电子/航空航天等需要高可靠性的系统中, 为了防止“系统在异常情况下, 受到干扰, MCU/CPU 程序跑飞, 导致系统长时间异常工作”, 通常是引进看门狗, 如果 MCU/CPU 不在规定的时间内按要求访问看门狗, 就认为 MCU/CPU 处于异常状态, 看门狗就会强制 MCU/CPU 复位, 使系统重新从头开始执行用户程序。

STC8 系列的看门狗复位是热启动复位中的硬件复位之一。STC8 系列单片机引进此功能, 使单片机系统可靠性设计变得更加方便、简洁。STC8 系列看门狗复位状态结束后, 系统固定从 ISP 监控程序区启动, 与看门狗复位前 IAP\_CONTR 寄存器的 SWBS 无关 (**注意: 此处与 STC15 系列 MCU 不同**)

#### 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

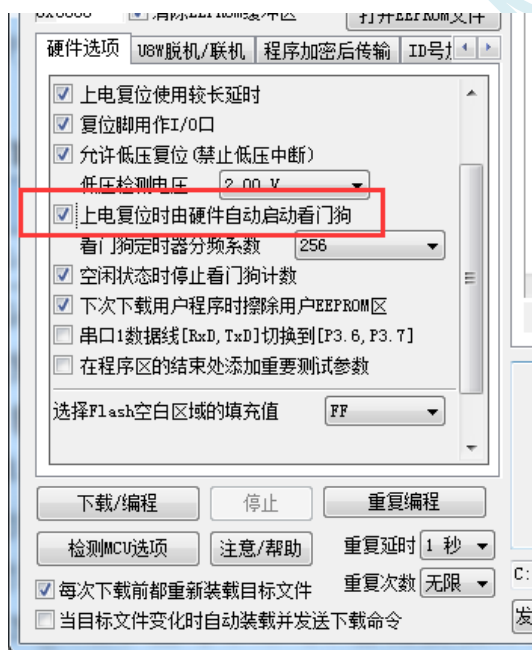
看门狗发生溢出时, 硬件自动将此位置 1, 需要软件清零。

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。

**注意:** 看门狗定时器可使用软件方式启动, 也可硬件自动启动, 一旦看门狗定时器启动后, 软件将无法关闭, 必须对单片机进行重新上电才可关闭。软件启动看门狗只需要对 EN\_WDT 位写 1 即可。若需要硬件启动看门狗, 则需要在 ISP 下载时进行如下图所示的设置:



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:

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

## 6.3.2 软件复位 (IAP\_CONTR)

**IAP\_CONTR (IAP 控制寄存器)** 对 IAP 控制寄存器写 60H, 可达到对单片机冷启动的效果

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

### 6.3.3 低压复位 (RSTCFG)

#### 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

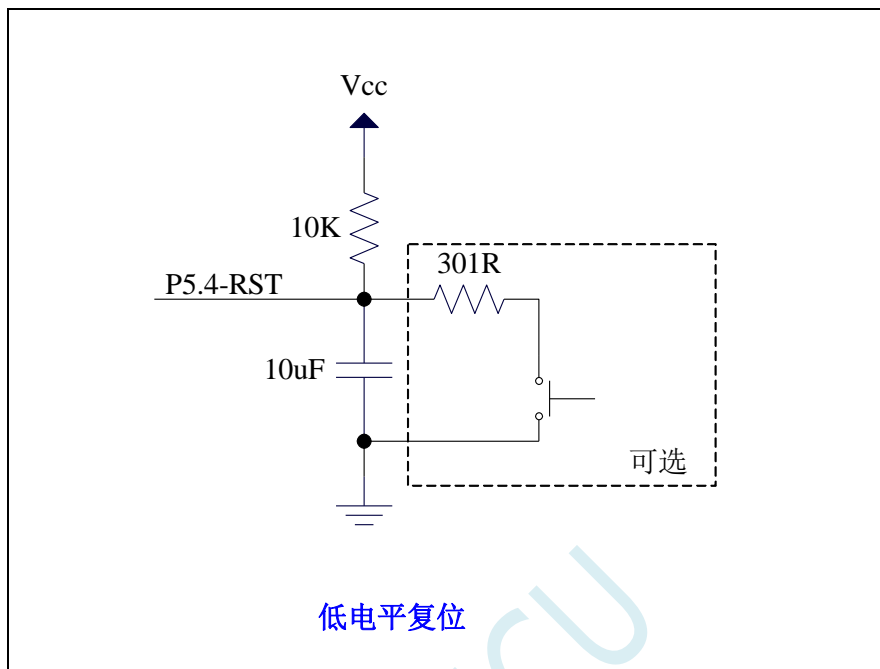
0: RST 管脚用作普通 I/O 口 (P5.4)

1: RST 管脚用作复位脚 (**低电平复位**)

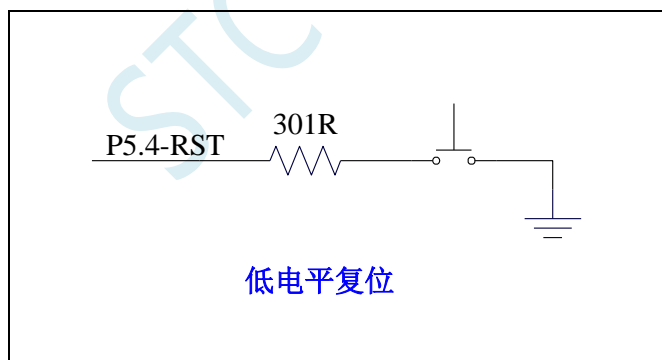
LVDS[1:0]: 低压检测阈值电压设置

LVDS[1:0]	低压检测阈值电压
00	2.0V
01	2.4V
10	2.7V
11	3.0V

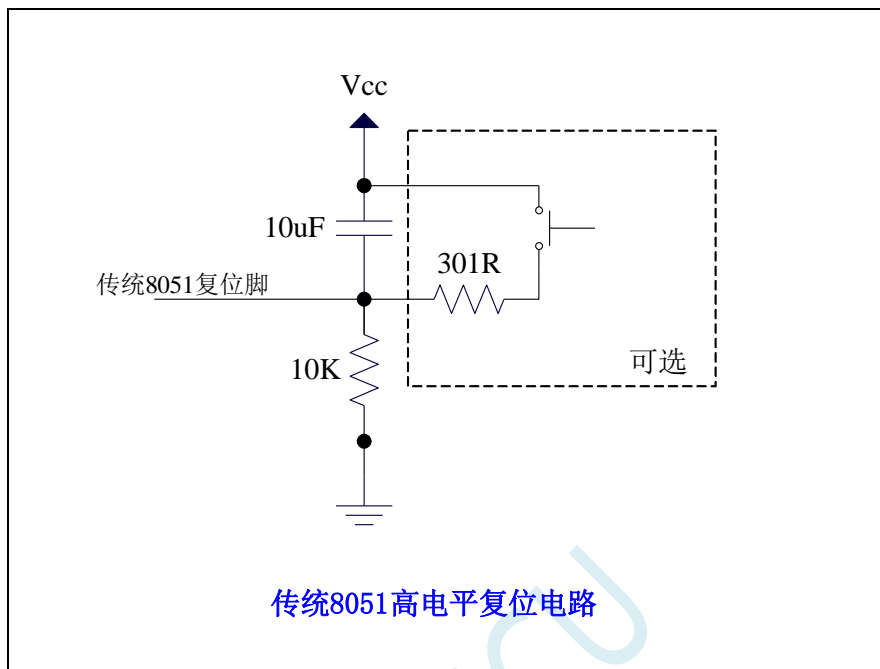
### 6.3.4 低电平上电复位参考电路（一般不需要）



### 6.3.5 低电平按键手动复位参考电路



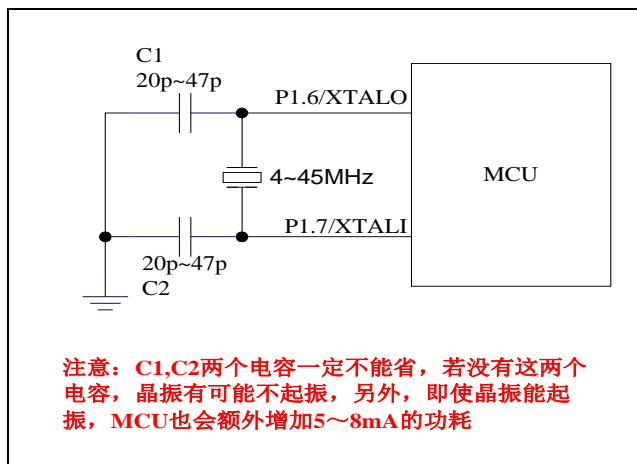
### 6.3.6 传统 8051 高电平上电复位参考电路



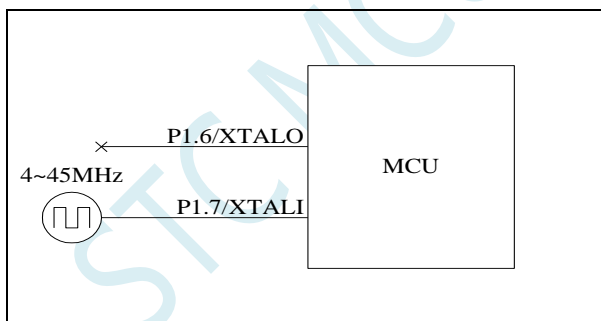
上图为传统 8051 的高电平复位电路，STC8A8K64D4 的复位为低电平复位，与传统复位电路不同

## 6.4 外部晶振及外部时钟电路

### 6.4.1 外部晶振输入电路



### 6.4.2 外部时钟输入电路 (P1.6 不可用作普通 I/O)



## 6.5 时钟停振/省电模式与系统电源管理

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

### 6.5.1 电源控制寄存器 (PCON)

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: 上电复位标志位。MCU 每次重新上电后, 硬件自动将此位置 1, 可软件将此位清零。

PD: 时钟停振模式/掉电模式/停电模式控制位

0: 无影响

1: 单片机进入时钟停振模式/掉电模式/停电模式, CPU 以及全部外设均停止工作。唤醒后硬件自动清零。(注: 时钟停振模式下, CPU 和全部的外设均停止工作, 但 SRAM 和 XRAM 中的数据是一直维持不变的)

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

注: 虽然 LVD 和比较器均可唤醒时钟停振模式化, 但时钟停振省电模式下, 不建议启动 LVD 和比较器, 否则硬件系统还会自动启动内部 1.19V 的高精准参考源, 这个高精准参考源有相应的抗温漂和调校线路, 大约会额外增加 300uA 的耗电, 而 MCU 进入时钟停振模式后, 3.3V 工作电压时只耗约 0.4uA 的电流, 所以进入时钟停振模式时不建议开 LVD 和比较器。如果确实需要用, 建议开启掉电唤醒定时器, 掉电唤醒定时器只会增加约 1.4uA 的耗电, 这个耗电一般系统是可以接受的。让掉电唤醒定时器每 5 秒唤醒一次 MCU, 唤醒后可用 LVD、比较器、ADC 检测外部电池电压, 检测工作约耗时 1mS 后再进入时钟停振/省电模式, 这样增加的平均电流小于 1uA, 则整体功耗大约为 2.8uA (0.4uA + 1.4uA + 1uA)。

STC MCU

## 6.6 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。

### 6.6.1 掉电唤醒定时器计数寄存器（WKTCL, WKTCH）

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 32767 次，则应对{WKTCH[6:0],WKTCL[7:0]}写入 7FFE<sub>H</sub>（即 32766）。（**计数值 0 和计数值 32767 为内部保留值，用户不能使用**）。内部掉电唤醒定时器有自己的内部时钟，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，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）

$$\text{掉电唤醒定时器定时时间} = \frac{10^6 \times 16 \times \text{计数次数}}{F_{wt}} \quad (\text{微秒})$$

假设  $F_{wt}=32\text{KHz}$ ，则有：

{WKTCH[6:0],WKTCL[7:0]}	掉电唤醒专用定时器计数时间
<del>0 (内部保留)</del>	
1	$10^6 \div 32\text{K} \times 16 \times (1+1) \approx 1 \text{ 毫秒}$
9	$10^6 \div 32\text{K} \times 16 \times (1+9) \approx 5 \text{ 毫秒}$
99	$10^6 \div 32\text{K} \times 16 \times (1+99) \approx 50 \text{ 毫秒}$
999	$10^6 \div 32\text{K} \times 16 \times (1+999) \approx 0.5 \text{ 秒}$
4095	$10^6 \div 32\text{K} \times 16 \times (1+4095) \approx 2 \text{ 秒}$
32766	$10^6 \div 32\text{K} \times 16 \times (1+32766) \approx 16 \text{ 秒}$
<del>32767 (内部保留)</del>	



## 6.7 范例程序

### 6.7.1 选择系统时钟源

#### C 语言代码

---



---

```
//测试工作频率为 11.0592MHz
```

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

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

```
sfr P_SW2 = 0xba;

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

```
void main()
{
```

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

```
    P_SW2 = 0x80;
    CLKSEL = 0x00;
    P_SW2 = 0x00;
```

```
//选择内部IRC(默认)
```

```
/*
```

```
    P_SW2 = 0x80;
    XOSCCR = 0xc0;
    while (!(XOSCCR & 1));
    CLKDIV = 0x00;
```

```
//启动外部晶振
//等待时钟稳定
//时钟不分频
```

```

    CLKSEL = 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
    CLKSEL = 0x03;       //选择内部32K
    P_SW2 = 0x00;
*/
    while (1);
}

```

## 汇编代码

;测试工作频率为11.0592MHz

<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>
<i>CLKSEL</i>	<i>EQU</i>	<i>0FE00H</i>
<i>CLKDIV</i>	<i>EQU</i>	<i>0FE01H</i>
<i>HIRCCR</i>	<i>EQU</i>	<i>0FE02H</i>
<i>XOSCCR</i>	<i>EQU</i>	<i>0FE03H</i>
<i>IRC32KCR</i>	<i>EQU</i>	<i>0FE04H</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>0100H</i>
<i>MAIN:</i>	<i>MOV</i>	<i>SP, #5FH</i>
	<i>MOV</i>	<i>P0M0, #00H</i>
	<i>MOV</i>	<i>P0M1, #00H</i>
	<i>MOV</i>	<i>P1M0, #00H</i>
	<i>MOV</i>	<i>P1M1, #00H</i>
	<i>MOV</i>	<i>P2M0, #00H</i>
	<i>MOV</i>	<i>P2M1, #00H</i>
	<i>MOV</i>	<i>P3M0, #00H</i>
	<i>MOV</i>	<i>P3M1, #00H</i>
	<i>MOV</i>	<i>P4M0, #00H</i>
	<i>MOV</i>	<i>P4M1, #00H</i>
	<i>MOV</i>	<i>P5M0, #00H</i>
	<i>MOV</i>	<i>P5M1, #00H</i>

```

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

JMP      $

END

```

## 6.7.2 主时钟分频输出

### 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      P0MI      = 0x93;
sfr      P0M0      = 0x94;
sfr      P1MI      = 0x91;
sfr      P1M0      = 0x92;
sfr      P2MI      = 0x95;

```

```

sfr    P2M0    = 0x96;
sfr    P3M1    = 0xb1;
sfr    P3M0    = 0xb2;
sfr    P4M1    = 0xb3;
sfr    P4M0    = 0xb4;
sfr    P5M1    = 0xc9;
sfr    P5M0    = 0xca;

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

    P_SW2 = 0x80;
//    MCLKOCR = 0x01;           //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);
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

```

P_SW2      DATA      0BAH

MCLKOCR    EQU        0FE05H

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

                ORG      0000H
                LJMP    MAIN

                ORG      0100H

MAIN:

```

```

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

MOV      P_SW2, #80H
; MOV      A, #01H           ;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

```

## 6.7.3 看门狗定时器应用

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

```

```

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

```

```

PIM0 = 0x00;
PIM1 = 0x00;
P2M0 = 0x00;
P2M1 = 0x00;
P3M0 = 0x00;
P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 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 = 0x33; //清看门狗,否则系统复位
// WDT_CONTR = 0x34; //清看门狗,否则系统复位
// WDT_CONTR = 0x37; //清看门狗,否则系统复位

Display(); //显示模块
Scankey(); //按键扫描模块
MotorDriver(); //电机驱动模块
}
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

```

WDT_CONTR DATA 0C1H

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

ORG 0000H
LJMP MAIN

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

```

```

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

;        MOV      WDT_CONTR,#23H      ;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:
;        MOV      WDT_CONTR,#33H      ;清看门狗,否则系统复位
;        MOV      WDT_CONTR,#34H      ;清看门狗,否则系统复位
;        MOV      WDT_CONTR,#37H      ;清看门狗,否则系统复位

LCALL    DISPLAY      ;显示模块
LCALL    SCANKEY      ;按键扫描模块
LCALL    MOTORDRIVER  ;电机驱动模块
JMP     LOOP

END

```

## 6.7.4 软复位实现自定义下载

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

```

```

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

```

```

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

P32 = 1; //测试端口
P33 = 1; //测试端口

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

```

## 汇编代码

;测试工作频率为 11.0592MHz

```

IAP_CONTR    DATA    0C7H

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

                ORG     0000H
                LJMP   MAIN

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

                SETB   P3.2

```



```

        SETB      P3.3
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

```

## 6.7.5 低压检测

### 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      P0MI        = 0x93;
sfr      P0M0        = 0x94;
sfr      P1MI        = 0x91;
sfr      P1M0        = 0x92;
sfr      P2MI        = 0x95;
sfr      P2M0        = 0x96;
sfr      P3MI        = 0xb1;
sfr      P3M0        = 0xb2;
sfr      P4MI        = 0xb3;
sfr      P4M0        = 0xb4;
sfr      P5MI        = 0xc9;
sfr      P5M0        = 0xca;

```

```

void Lvd_Isr() interrupt 6

```

```

{
    PCON &= ~LVDF;      //清中断标志
    P32 = ~P32;        //测试端口
}

```

```

void main()

```

```

{
    P0M0 = 0x00;
    P0MI = 0x00;
    P1M0 = 0x00;
    P1MI = 0x00;
    P2M0 = 0x00;
    P2MI = 0x00;
    P3M0 = 0x00;
}

```

```

P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 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);
}

```

## 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

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

ORG       0000H
LJMP     MAIN
ORG       0033H
LJMP     LVDISR

LVDISR:  ORG       0100H

ANL      PCON,#NOT LVDF ;清中断标志
CPL      P3.2           ;测试端口
RETI

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

```

```

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

ANL      PCON, #NOT LVDF      ;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

```

## 6.7.6 省电模式

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

void INT0_Isr() interrupt 0
{
    P34 = ~P34;                //测试端口
}

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

```

```

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

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

while (1);
}

```

## 汇编代码

;测试工作频率为11.0592MHz

```

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

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

        ORG      0000H
        LJMP     MAIN

        ORG      0003H
        LJMP     INT0ISR

INT0ISR: ORG      0100H

        CPL      P3.4      ;测试端口
        RETI

MAIN:   MOV      SP, #5FH
        MOV      P0M0, #00H
        MOV      P0M1, #00H
        MOV      P1M0, #00H

```

```

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

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

END

```

## 6.7.7 使用 INT0/INT1/INT2/INT3/INT4 管脚中断唤醒省电模式

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

```

```
void INT0_Isr() interrupt 0
{
    P10 = !P10;           //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()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IT0 = 0;              //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

    INTCLK0 = EX2;        //使能INT2 下降沿中断
    INTCLK0 /= EX3;       //使能INT3 下降沿中断
    INTCLK0 /= EX4;       //使能INT4 下降沿中断

    EA = 1;

    PCON = 0x02;          //MCU 进入掉电模式
    _nop_();              //掉电模式被唤醒后,MCU 首先会执行此语句
                           //然后再进入中断服务程序

    _nop_();
}
```

```

    _nop_();
    _nop_();

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

```

## 汇编代码

;测试工作频率为 11.0592MHz

<i>INTCLKO</i>	<i>DATA</i>	<i>8FH</i>	
<i>EX2</i>	<i>EQU</i>	<i>10H</i>	
<i>EX3</i>	<i>EQU</i>	<i>20H</i>	
<i>EX4</i>	<i>EQU</i>	<i>40H</i>	
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>	
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>	
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>	
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>	
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>	
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>	
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>	
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>	
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>	
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>	
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>	
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>	
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>0003H</i>	
	<i>LJMP</i>	<i>INT0ISR</i>	
	<i>ORG</i>	<i>0013H</i>	
	<i>LJMP</i>	<i>INT1ISR</i>	
	<i>ORG</i>	<i>0053H</i>	
	<i>LJMP</i>	<i>INT2ISR</i>	
	<i>ORG</i>	<i>005BH</i>	
	<i>LJMP</i>	<i>INT3ISR</i>	
	<i>ORG</i>	<i>0083H</i>	
	<i>LJMP</i>	<i>INT4ISR</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>INT0ISR:</i>	<i>CPL</i>	<i>P1.0</i>	;测试端口
	<i>RETI</i>		
<i>INT1ISR:</i>	<i>CPL</i>	<i>P1.0</i>	;测试端口
	<i>RETI</i>		
<i>INT2ISR:</i>	<i>CPL</i>	<i>P1.0</i>	;测试端口
	<i>RETI</i>		
<i>INT3ISR:</i>	<i>CPL</i>	<i>P1.0</i>	;测试端口
	<i>RETI</i>		
<i>INT4ISR:</i>			

```

        CPL            P1.0                ;测试端口
        RETI

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

        CLR          IT0                  ;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                               ;掉电模式被唤醒后,MCU 首先会执行此语句
                                           ;然后再进入中断服务程序

        NOP
        NOP
        NOP
LOOP:
        CPL          P1.1
        JMP          LOOP

        END

```

## 6.7.8 使用 T0/T1/T2/T3/T4 管脚中断唤醒 MCU 省电模式

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

```

```
void TM0_Isr() interrupt 1
```

```
{
    P10 = !P10;           //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
}
```

```
void TM3_Isr() interrupt 19
```

```
{
    P10 = !P10;           //测试端口
}
```

```
void TM4_Isr() interrupt 20
```

```
{
    P10 = !P10;           //测试端口
}
```

```
void main()
```

```
{
    P0M0 = 0x00;
    P0M1 = 0x00;
}
```

```

PIM0 = 0x00;
PIM1 = 0x00;
P2M0 = 0x00;
P2M1 = 0x00;
P3M0 = 0x00;
P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

TMOD = 0x00;
TL0 = 0x66; //65536-11.0592M/12/1000
TH0 = 0xfc;
TR0 = 1; //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_();
_nop_();
_nop_();

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

```

## 汇编代码

;测试工作频率为11.0592MHz

```

T2L      DATA      0D7H
T2H      DATA      0D6H
T3L      DATA      0D5H

```

<i>T3H</i>	<i>DATA</i>	<i>0D4H</i>	
<i>T4L</i>	<i>DATA</i>	<i>0D3H</i>	
<i>T4H</i>	<i>DATA</i>	<i>0D2H</i>	
<i>T4T3M</i>	<i>DATA</i>	<i>0D1H</i>	
<i>AUXR</i>	<i>DATA</i>	<i>8EH</i>	
<i>IE2</i>	<i>DATA</i>	<i>0AFH</i>	
<i>ET2</i>	<i>EQU</i>	<i>04H</i>	
<i>ET3</i>	<i>EQU</i>	<i>20H</i>	
<i>ET4</i>	<i>EQU</i>	<i>40H</i>	
<i>AUXINTIF</i>	<i>DATA</i>	<i>0EFH</i>	
<i>T2IF</i>	<i>EQU</i>	<i>01H</i>	
<i>T3IF</i>	<i>EQU</i>	<i>02H</i>	
<i>T4IF</i>	<i>EQU</i>	<i>04H</i>	
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>	
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>	
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>	
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>	
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>	
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>	
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>	
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>	
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>	
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>	
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>	
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>	
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>000BH</i>	
	<i>LJMP</i>	<i>TM0ISR</i>	
	<i>ORG</i>	<i>001BH</i>	
	<i>LJMP</i>	<i>TM1ISR</i>	
	<i>ORG</i>	<i>0063H</i>	
	<i>LJMP</i>	<i>TM2ISR</i>	
	<i>ORG</i>	<i>009BH</i>	
	<i>LJMP</i>	<i>TM3ISR</i>	
	<i>ORG</i>	<i>00A3H</i>	
	<i>LJMP</i>	<i>TM4ISR</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>TM0ISR:</i>	<i>CPL</i>	<i>P1.0</i>	<i>;测试端口</i>
	<i>RETI</i>		
<i>TM1ISR:</i>	<i>CPL</i>	<i>P1.0</i>	<i>;测试端口</i>
	<i>RETI</i>		
<i>TM2ISR:</i>	<i>CPL</i>	<i>P1.0</i>	<i>;测试端口</i>
	<i>RETI</i>		
<i>TM3ISR:</i>	<i>CPL</i>	<i>P1.0</i>	<i>;测试端口</i>
	<i>RETI</i>		
<i>TM4ISR:</i>	<i>CPL</i>	<i>P1.0</i>	<i>;测试端口</i>
	<i>RETI</i>		

**MAIN:**

```

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

MOV     TMOD, #00H
MOV     TL0, #66H                ;65536-11.0592M/12/1000
MOV     TH0, #0FCH
SETB    TR0                    ;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
;T0/T1/T2/T3/T4 的外部管脚唤醒后,
;不进入中断服务程序, 只是继续执行程序,
;与INT0/INT1/INT2/INT3/INT4 的掉电唤醒不一样
;建议多加几个NOP 指令, 如3 个以上

NOP
NOP
NOP

LOOP:   CPL     P1.1
        JMP     LOOP

END

```

## 6.7.9 使用 RxD/RxD2/RxD3/RxD4 管脚中断唤醒 MCU 省电模式

### 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  
#define  ES3      0x08  
#define  ES4      0x10
```

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

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

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

```
void UART1_Isr() interrupt 4  
{  
}
```

```
void UART2_Isr() interrupt 8  
{  
}
```

```
void UART3_Isr() interrupt 17  
{  
}
```

```
void UART4_Isr() interrupt 18  
{  
}
```

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

```

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

P_SW1 = 0x00; //RXD/P3.0 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/P4.6 下降沿唤醒

P_SW2 = 0x00; //RXD3/P0.0 下降沿唤醒
// P_SW2 = 0x02; //RXD3_2/P5.0 下降沿唤醒

P_SW2 = 0x00; //RXD4/P0.2 下降沿唤醒
// P_SW2 = 0x04; //RXD4_2/P5.2 下降沿唤醒

ES = I; //使能串口中断
IE2 = ES2; //使能串口中断
IE2 /= ES3; //使能串口中断
IE2| = ES4; //使能串口中断
EA = I;

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();
_nop();
_nop();

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

```

## 汇编代码

;测试工作频率为 11.0592MHz

IE2	DATA	0AFH
ES2	EQU	01H
ES3	EQU	08H
ES4	EQU	10H
P_SW1	DATA	0A2H
P_SW2	DATA	0BAH
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H

```

P5M1      DATA      0C9H
P5M0      DATA      0CAH

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

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

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

          MOV         P_SW1, #00H      ;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/P4.6 下降沿唤醒

          MOV         P_SW2, #00H      ;RXD3/P0.0 下降沿唤醒
;          MOV         P_SW2, #02H     ;RXD3_2/P5.0 下降沿唤醒

          MOV         P_SW2, #00H      ;RXD4/P0.2 下降沿唤醒
;          MOV         P_SW2, #04H     ;RXD4_2/P5.2 下降沿唤醒

          SETB        ES                ; 使能串口中断
          MOV         IE2, #ES2        ; 使能串口中断
          ORL         IE2, #ES3        ; 使能串口中断
          ORL         IE2, #ES4        ; 使能串口中断
          SETB        EA

```

```

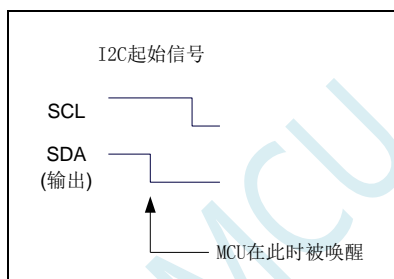
NOP
NOP
MOV          PCON,#02H
NOP
NOP
NOP
NOP
LOOP:
CPL          P1.1
JMP          LOOP

END

```

;RxD/RxD2/RxD3/RxD4 及相应的可切换的几组管脚下降  
;沿唤醒后, 不进入中断服务程序, 只是继续执行程序,  
;与INT0/INT1/INT2/INT3/INT4 的掉电唤醒不一样  
;建议多加几个NOP 指令, 如3 个以上  
;MCU 进入掉电模式  
;掉电唤醒后不会进入中断服务程序,

## 6.7.10 使用 I2C 的 SDA 脚唤醒 MCU 省电模式



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

sbit     P11        = P1^1;

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

```

```
void i2c_isr() interrupt 24
```



```

{
    P_SW2 /= 0x80;
    I2CSLST &= ~0x40;
}

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

    P_SW2 = 0x00; //SDA/P1.4 下降沿唤醒
    // P_SW2 = 0x10; //SDA_2/P2.4 下降沿唤醒
    // P_SW2 = 0x30; //SDA_4/P3.3 下降沿唤醒
    P_SW2 /= 0x80;
    I2CCFG = 0x80; //使能 I2C 模块的从机模式
    I2CSLCR = 0x40; //使能起始信号中断
    EA = 1;

    PCON = 0x02; //MCU 进入掉电模式
    _nop_();
    _nop_();
    _nop_();
    _nop_();

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

```

## 汇编代码

;测试工作频率为 11.0592MHz

P_SW2	DATA	0BAH
I2CCFG	XDATA	0FE80H
I2CSLCR	XDATA	0FE83H
I2CSLST	XDATA	0FE84H
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H

```

P4M0      DATA      0B4H
P5M1      DATA      0C9H
P5M0      DATA      0CAH

          ORG         0000H
          LJMP        MAIN
          ORG         00C3H
          LJMP        I2CISR

I2CISR:   ORG         0100H

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

MAIN:

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

          MOV         P_SW2,#00H           ;SDA/P1.4 下降沿唤醒
//          MOV         P_SW2,#10H        ;SDA_2/P2.4 下降沿唤醒
//          MOV         P_SW2,#30H        ;SDA_4/P3.3 下降沿唤醒
          ORL         P_SW2,#80H
          MOV         DPTR,#I2CCFG
          MOV         A,#80H
          MOVX        @DPTR,A             ;使能 I2C 模块的从机模式
          MOV         DPTR,# I2CSLCR
          MOV         A,#40H             ;使能起始信号中断
          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
          NOP
          NOP

LOOP:     CPL         P1.1
          JMP         LOOP

```

---



---

END

---



---

## 6.7.11 使用掉电唤醒定时器唤醒省电模式

### C 语言代码

---



---

```
//测试工作频率为 11.0592MHz
```

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

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

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

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

```
void main()
{
```

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

```
    WKTCL = 0xff;
    WKTCH = 0x87;
```

```
//设定掉电唤醒时钟约为 1 秒钟
```

```
    while (1)
    {
```

```
        _nop_();
        _nop_();
        PCON = 0x02;
        _nop_();
        _nop_();
        _nop_();
        _nop_();
```

```
//MCU 进入掉电模式
```

```

    P11 = ~P11;
  }
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

```

WKTCL    DATA    0AAH
WKTCH    DATA    0ABH

```

```

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

```

```

ORG      0000H
LJMP     MAIN

```

```

ORG      0100H

```

MAIN:

```

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

```

```

MOV      WKTCL, #0FFH
MOV      WKTCH, #87H

```

;设定掉电唤醒时钟约为 1 秒钟

LOOP:

```

NOP
NOP
MOV      PCON, #02H
NOP
NOP
NOP
NOP
CPL     P1.1
JMP     LOOP

```

;MCU 进入掉电模式

```

END

```

## 6.7.12 LVD 中断唤醒省电模式，建议配合使用掉电唤醒定时器

时钟停振省电模式下，不建议启动 LVD 和比较器，否则硬件系统还会自动启动内部 1.19V 的高精度参考源，这个高精度参考源有相应的抗温漂和调校线路，大约会额外增加 300uA 的耗电，而 MCU 进入时钟停振模式后，3.3V 工作电压时只耗约 0.4uA 的电流，所以进入时钟停振模式时不建议开 LVD 和比较器。如果确实需要用，建议开启掉电唤醒定时器，掉电唤醒定时器只会增加约 1.4uA 的耗电，这个耗电一般系统是可以接受的。让掉电唤醒定时器每 5 秒唤醒一次 MCU，唤醒后可用 LVD、比较器、ADC 检测外部电池电压，检测工作约耗时 1mS 后再进入时钟停振/省电模式，这样增加的平均电流小于 1uA，则整体功耗大约为 2.8uA (0.4uA + 1.4uA + 1uA)。

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

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

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

```

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

PCON &= ~LVDF; //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_();
_nop_();
_nop_();

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

```

## 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
P0M1	DATA	093H	
P0M0	DATA	094H	
P1M1	DATA	091H	
P1M0	DATA	092H	
P2M1	DATA	095H	
P2M0	DATA	096H	
P3M1	DATA	0B1H	
P3M0	DATA	0B2H	
P4M1	DATA	0B3H	
P4M0	DATA	0B4H	
P5M1	DATA	0C9H	
P5M0	DATA	0CAH	
	ORG	0000H	
	LJMP	MAIN	
	ORG	0033H	
	LJMP	LVDISR	
	ORG	0100H	

**LVDISR:**

```

ANL      PCON,#NOT LVDF      ;清中断标志
CPL      P1.0                ;测试端口
RETI

```

**MAIN:**

```

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

```

```

ANL      PCON,#NOT LVDF      ;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
NOP
NOP
NOP

```

**LOOP:**

```

CPL      P1.1
JMP      LOOP

```

```

END

```

## 6.7.13 比较器中断唤醒省电模式，建议配合使用掉电唤醒定时器

时钟停振省电模式下，不建议启动 LVD 和比较器，否则硬件系统还会自动启动内部 1.19V 的高精度参考源，这个高精度参考源有相应的抗温漂和调校线路，大约会额外增加 300uA 的耗电，而 MCU 进入时钟停振模式后，3.3V 工作电压时只耗约 0.4uA 的电流，所以进入时钟停振模式时不建议开 LVD 和比较器。如果确实需要用，建议开启掉电唤醒定时器，掉电唤醒定时器只会增加约 1.4uA 的耗电，这个耗电一般系统是可以接受的。让掉电唤醒定时器每 5 秒唤醒一次 MCU，唤醒后可用 LVD、比较器、ADC 检测外部电池电压，检测工作约耗时 1mS 后再进入时钟停振/省电模式，这样增加的平均电流小于 1uA，则整体功耗大约为 2.8uA (0.4uA + 1.4uA + 1uA)。

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

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

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

    CMPCR2 = 0x00;
    CMPCR1 = 0x80;           //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_();
    _nop_();
    _nop_();

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

```



## 汇编代码

;测试工作频率为11.0592MHz

```

CMPCR1    DATA    0E6H
CMPCR2    DATA    0E7H

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

                ORG    0000H
                LJMP   MAIN
                ORG    00ABH
                LJMP   CMPISR

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

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

                MOV    CMPCR2,#00H
                MOV    CMPCR1,#80H        ;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
                NOP
                NOP

```

LOOP:

```

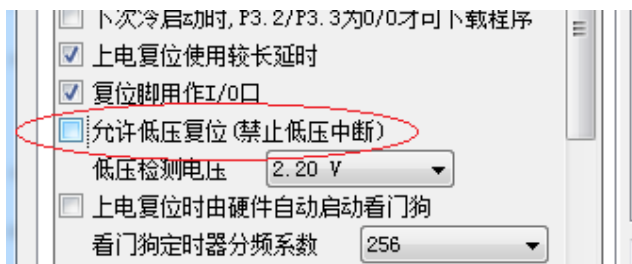
        CPL        PI.1
        JMP        LOOP
    END

```

## 6.7.14 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

（建议使用 ADC 的第 15 通道检测电池电压，见 ADC 章节）



### C 语言代码

//测试工作频率为 11.0592MHz

```

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

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

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

#define LVDF        0x20           //PCON.5

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

void delay()
{
    int i;

```

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

void main()
{
    unsigned char power;

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

    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
}
}

```

## 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

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

        ORG      0000H
        JMP      MAIN

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

        ANL      PCON, #NOT LVDF
        MOV      RSTCFG, #LVD3V0

```

**LOOP:**

```

MOV      B,#0FH

MOV      RSTCFG,#LVD3V0
CALL     DELAY
ANL      PCON,#NOT LVDF
CALL     DELAY
MOV      A,PCON
ANL      A,#LVDF
JZ       SKIP

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

MOV      RSTCFG,#LVD2V7
CALL     DELAY
ANL      PCON,#NOT LVDF
CALL     DELAY
MOV      A,PCON
ANL      A,#LVDF
JZ       SKIP

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

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*

---

---

STC MCU

## 7 存储器

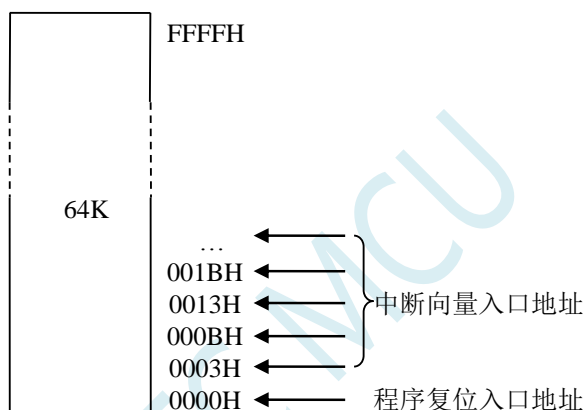
STC8A8K64D4 系列单片机的程序存储器和数据存储器是各自独立编址的。由于没有提供访问外部程序存储器的总线，单片机的所有程序存储器都是片上 Flash 存储器，不能访问外部程序存储器。

STC8A8K64D4 系列单片机内部集成了大容量的数据存储器。STC8A8K64D4 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。

STC8A8K64D4-64Pin/48Pin 系列单片内部集成了 64K 字节的 Flash 程序存储器 (ROM)。



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。

STC8A8K64D4 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

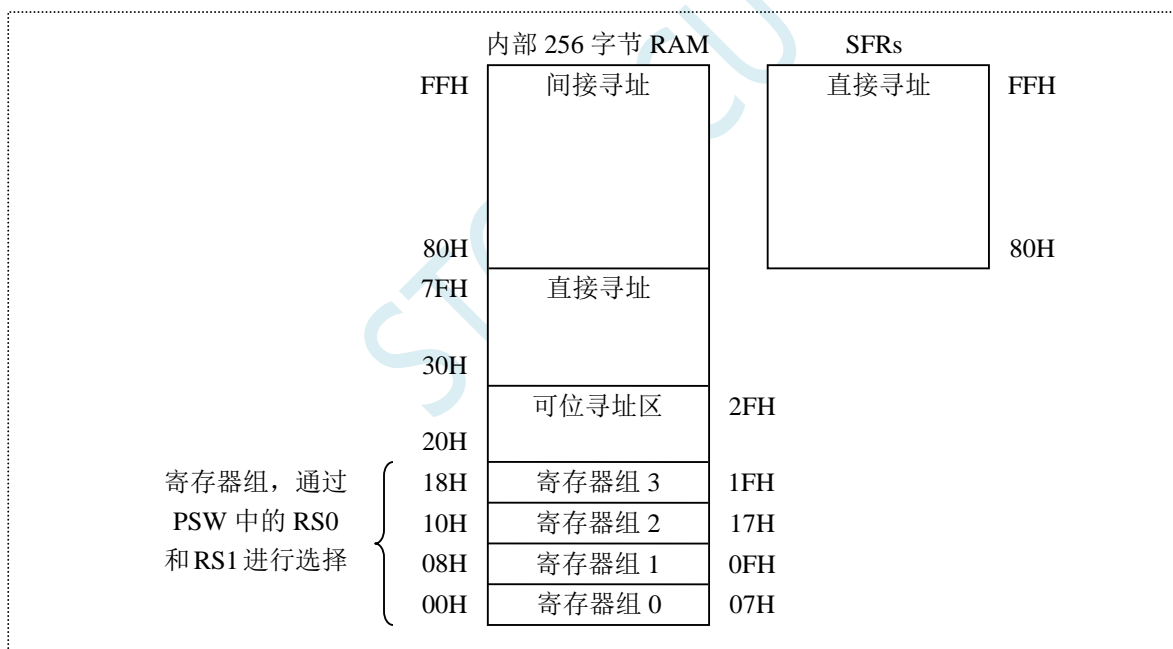
STC8A8K64D4 系列单片机内部集成的 RAM 可用于存放程序执行的中间结果和过程数据。

单片机系列	内部直接访问 RAM (DATA)	内部间接访问 RAM (IDATA)	内部扩展 RAM (XDATA)
STC8A8K64D4-64Pin/48Pin 系列	128 字节	128 字节	8192 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。

### 7.2.2 程序状态寄存器 (PSW)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PSW	D0H	CY	AC	F0	<b>RS1</b>	<b>RS0</b>	OV	F1	P



CY: 进/借位标志位。

AC: 辅组进/借位标志位。

F0: 用户标志位 0。

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)

OV: 溢出标志位。

F1: 用户标志位 1。

P: 奇偶校验标志位。

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 初始化为 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。

### 7.2.3 内部扩展 RAM, XRAM, XDATA

STC8A8K64D4 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 声明存储类型即可。如：

```
unsigned char xdata i;
```

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。

## 7.2.4 辅助寄存器 (AUXR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	<b>EXTRAM</b>	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。

STC MCU

## 7.2.5 外部扩展 RAM, XRAM, XDATA

STC8A8K64D4 系列封装管脚数为 40 及其以上的单片机具有扩展 64KB 外部数据存储器的能力。访问外部数据存储器期间, WR/RD/ALE 信号要有效。STC8A8K64D4 系列单片机新增了一个控制外部 64K 字节数据总线速度的特殊功能寄存器 BUS\_SPEED, 说明如下:

## 7.2.6 总线速度控制寄存器 (BUS\_SPEED)

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

RW\_S[1:0]: RD/WR 控制线选择位

00: P4.4 为 RD, P4.3 为 WR

01: P3.7 为 RD, P3.6 为 WR

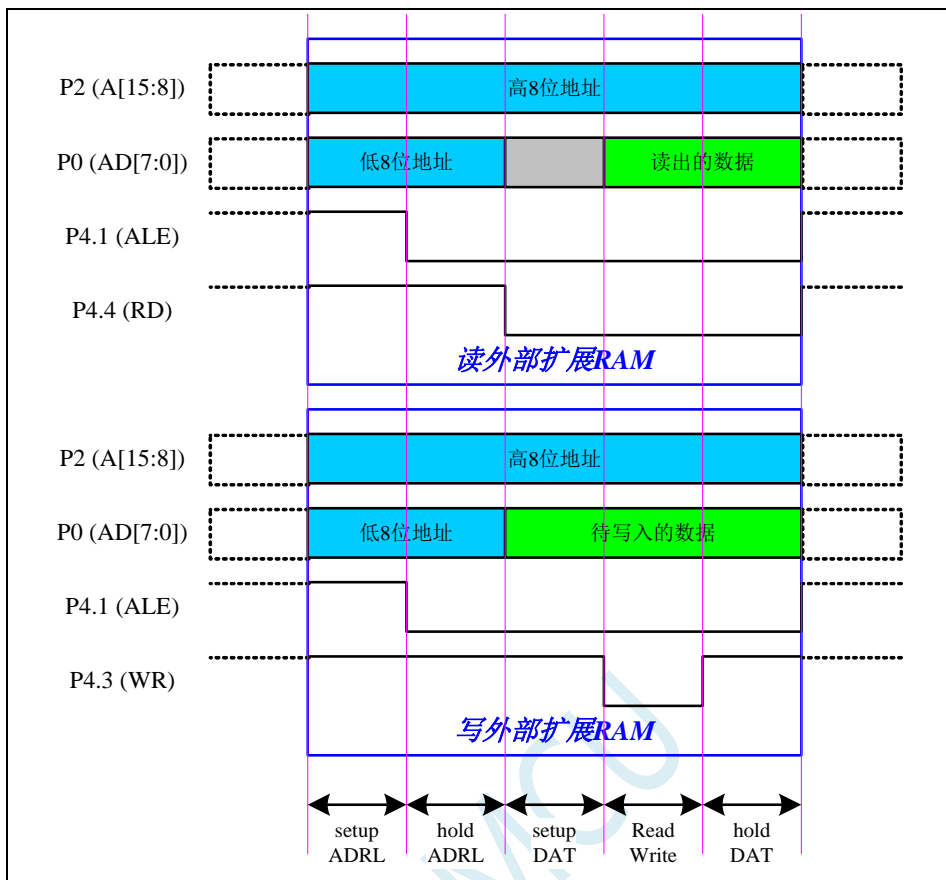
10: P4.2 为 RD, P4.0 为 WR

11: 保留

SPEED[2:0]: 总线读写速度控制 (读写数据时控制信号和数据信号的准备时间和保持时间)

指令	时钟数	
	访问内部扩展 RAM	访问外部扩展 RAM
MOVX A,@Ri	3	3+5*(SPEED+1)
MOVX @Ri,A	3	3+5*(SPEED+1)
MOVX A,@DPTR	2	2+5*(SPEED+1)
MOVX @DPTR,A	2	2+5*(SPEED+1)

读写外部扩展 RAM 时序如下图所示:



## 7.2.7 8051 中可位寻址的数据存储器

8051 单片机内部可位寻址的数据存储器包括两部分: 第一部分的地址范围为 00H~7FH, 第二部分的地址范围是 80H~FFH。00H~7FH 的位寻址区域是数据区 20H~2FH 这 16 个字节的映射; 而 80H~FFH 的位寻址区域则是所有的特殊功能寄存器中地址能被 8 整除的 16 个特殊功能寄存器 (包括 80H、88H、90H、98H、A0H、A8H、B0H、B8H、C0H、C8H、D0H、D8H、E0H、E8H、F0H、F8H) 的映射。

数据存储器地址	位寻址地址							
	B7	B6	B5	B4	B3	B2	B1	B0
F8H (P7)	FFH F8H.7	FEH F8H.6	FDH F8H.5	FCH F8H.4	FBH F8H.3	FAH F8H.2	F9H F8H.1	F8H F8H.0
F0H (B)	F7H F0H.7	F6H F0H.6	F5H F0H.5	F4H F0H.4	F3H F0H.3	F2H F0H.2	F1H F0H.1	F0H F0H.0
E8H (P6)	EFH E8H.7	EEH E8H.6	EDH E8H.5	ECH E8H.4	EBH E8H.3	EAH E8H.2	E9H E8H.1	E8H E8H.0
E0H (ACC)	E7H E0H.7	E6H E0H.6	E5H E0H.5	E4H E0H.4	E3H E0H.3	E2H E0H.2	E1H E0H.1	E0H E0H.0
D8H (CCON)	DFH D8H.7	DEH D8H.6	DDH D8H.5	DCH D8H.4	DBH D8H.3	DAH D8H.2	D9H D8H.1	D8H D8H.0
D0H (PSW)	D7H D0H.7	D6H D0H.6	D5H D0H.5	D4H D0H.4	D3H D0H.3	D2H D0H.2	D1H D0H.1	D0H D0H.0
C8H (P5)	CFH C8H.7	CEH C8H.6	CDH C8H.5	CCH C8H.4	CBH C8H.3	CAH C8H.2	C9H C8H.1	C8H C8H.0
C0H (P4)	C7H C0H.7	C6H C0H.6	C5H C0H.5	C4H C0H.4	C3H C0H.3	C2H C0H.2	C1H C0H.1	C0H C0H.0
B8H (IP)	BFH B8H.7	BEH B8H.6	BDH B8H.5	BCH B8H.4	BBH B8H.3	BAH B8H.2	B9H B8H.1	B8H B8H.0
B0H (P3)	B7H B0H.7	B6H B0H.6	B5H B0H.5	B4H B0H.4	B3H B0H.3	B2H B0H.2	B1H B0H.1	B0H B0H.0
A8H (IE)	AFH A8H.7	AEH A8H.6	ADH A8H.5	ACH A8H.4	ABH A8H.3	AAH A8H.2	A9H A8H.1	A8H A8H.0
A0H (P2)	A7H A0H.7	A6H A0H.6	A5H A0H.5	A4H A0H.4	A3H A0H.3	A2H A0H.2	A1H A0H.1	A0H A0H.0
98H (SCON)	9FH 98H.7	9EH 98H.6	9DH 98H.5	9CH 98H.4	9BH 98H.3	9AH 98H.2	99H 98H.1	98H 98H.0
90H (P1)	97H 90H.7	96H 90H.6	95H 90H.5	94H 90H.4	93H 90H.3	92H 90H.2	91H 90H.1	90H 90H.0
88H (TCON)	8FH 88H.7	8EH 88H.6	8DH 88H.5	8CH 88H.4	8BH 88H.3	8AH 88H.2	89H 88H.1	88H 88H.0
80H (P0)	87H 80H.7	86H 80H.6	85H 80H.5	84H 80H.4	83H 80H.3	82H 80H.2	81H 80H.1	80H 80H.0
2FH	7FH 2FH.7	7EH 2FH.6	7DH 2FH.5	7CH 2FH.4	7BH 2FH.3	7AH 2FH.2	79H 2FH.1	78H 2FH.0
2EH	77H 2EH.7	76H 2EH.6	75H 2EH.5	74H 2EH.4	73H 2EH.3	72H 2EH.2	71H 2EH.1	70H 2EH.0
2DH	6FH 2DH.7	6EH 2DH.6	6DH 2DH.5	6CH 2DH.4	6BH 2DH.3	6AH 2DH.2	69H 2DH.1	68H 2DH.0
2CH	67H 2CH.7	66H 2CH.6	65H 2CH.5	64H 2CH.4	63H 2CH.3	62H 2CH.2	61H 2CH.1	60H 2CH.0
2BH	5FH 2BH.7	5EH 2BH.6	5DH 2BH.5	5CH 2BH.4	5BH 2BH.3	5AH 2BH.2	59H 2BH.1	58H 2BH.0
2AH	57H 2AH.7	56H 2AH.6	55H 2AH.5	54H 2AH.4	53H 2AH.3	52H 2AH.2	51H 2AH.1	50H 2AH.0

<b>29H</b>	4FH 29H.7	4EH 29H.6	4DH 29H.5	4CH 29H.4	4BH 29H.3	4AH 29H.2	49H 29H.1	48H 29H.0
<b>28H</b>	47H 28H.7	46H 28H.6	45H 28H.5	44H 28H.4	43H 28H.3	42H 28H.2	41H 28H.1	40H 28H.0
<b>27H</b>	3FH 27H.7	3EH 27H.6	3DH 27H.5	3CH 27H.4	3BH 27H.3	3AH 27H.2	39H 27H.1	38H 27H.0
<b>26H</b>	37H 26H.7	36H 26H.6	35H 26H.5	34H 26H.4	33H 26H.3	32H 26H.2	31H 26H.1	30H 26H.0
<b>25H</b>	2FH 25H.7	2EH 25H.6	2DH 25H.5	2CH 25H.4	2BH 25H.3	2AH 25H.2	29H 25H.1	28H 25H.0
<b>24H</b>	27H 24H.7	26H 24H.6	25H 24H.5	24H 24H.4	23H 24H.3	22H 24H.2	21H 24H.1	20H 24H.0
<b>23H</b>	1FH 23H.7	1EH 23H.6	1DH 23H.5	1CH 23H.4	1BH 23H.3	1AH 23H.2	19H 23H.1	18H 23H.0
<b>22H</b>	17H 22H.7	16H 22H.6	15H 22H.5	14H 22H.4	13H 22H.3	12H 22H.2	11H 22H.1	10H 22H.0
<b>21H</b>	0FH 21H.7	0EH 21H.6	0DH 21H.5	0CH 21H.4	0BH 21H.3	0AH 21H.2	09H 21H.1	08H 21H.0
<b>20H</b>	07H 20H.7	06H 20H.6	05H 20H.5	04H 20H.4	03H 20H.3	02H 20H.2	01H 20H.1	00H 20H.0

## 7.3 存储器中的特殊参数，在 ISP 下载时可烧录进程序 FLASH

STC8A8K64D4 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 1.19V 参考信号源值以及 IRC 参数。

这些参数在 Flash 程序存储器（ROM）中的存放地址分别如下：

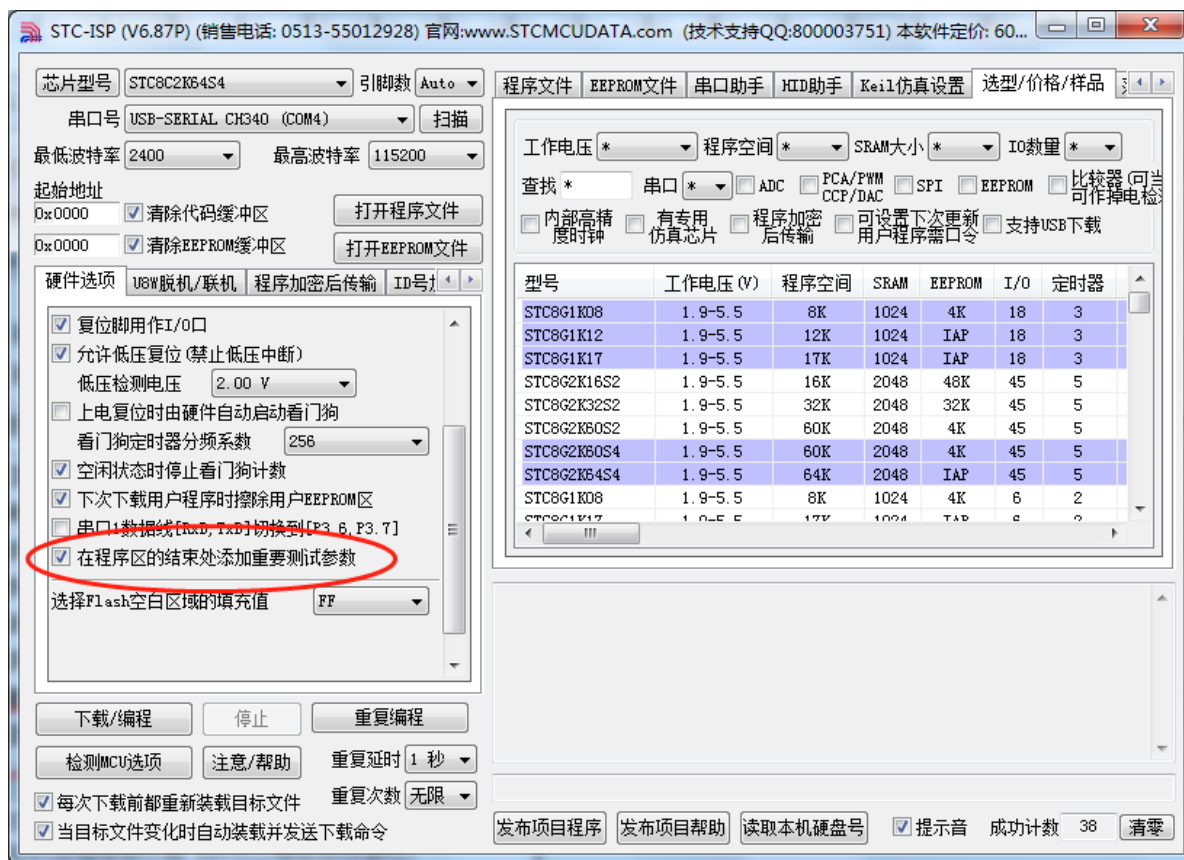
参数名称	保存地址				参数说明
	STC8A8K16D4	STC8A8K32D4	STC8A8K60D4	STC8A8K64D4	
全球唯一 ID 号	3FF9H~3FFFH	7FF9H~7FFFH	0EFF9H~0EFFFH	0FDF9H~0FDFFH	7 字节
内部 1.19V 参考信号源	3FF7H~3FF8H	7FF7H~7FF8H	0EFF7H~0EFF8H	0FDF7H~0FDF8H	毫伏（高字节在前）
32K 掉电唤醒定时器的频率	3FF5H~3FF6H	7FF5H~7FF6H	0EFF5H~0EFF6H	0FDF5H~0FDF6H	Hz（高字节在前）
22.1184MHz 的 IRC 参数	3FF4H	7FF4H	0EFF4H	0FDF4H	—
24MHz 的 IRC 参数	3FF3H	7FF3H	0EFF3H	0FDF3H	—
20MHz 的 IRC 参数	3FF2H	7FF2H	EFF2H	FDF2H	
27MHz 的 IRC 参数	3FF1H	7FF1H	EFF1H	FDF1H	
30MHz 的 IRC 参数	3FF0H	7FF0H	EFF0H	FDF0H	
33.1776MHz 的 IRC 参数	3FEFH	7FEFH	EFEFH	FDEFH	
35MHz 的 IRC 参数	3FEEH	7FEEH	EFEEH	FDEEH	
36.864MHz 的 IRC 参数	3FEDH	7FEDH	EFEDH	FDEDH	
40MHz 的 IRC 参数	3FECH	7FECH	EFECH	FDECH	
45MHz 的 IRC 参数	3FEBH	7FEBH	EFEBH	FDEBH	
6M 频段的 VRTRIM 参数	3FEAH	7FEAH	EFEAH	FDEAH	
10M 频段的 VRTRIM 参数	3FE9H	7FE9H	EFE9H	FDE9H	
27M 频段的 VRTRIM 参数	3FE8H	7FE8H	EFE8H	FDE8H	
44M 频段的 VRTRIM 参数	3FE7H	7FE7H	EFE7H	FDE7H	

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
内部 1.19V 参考信号源	idata: 0EFH~0F0H	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	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 FLASH 程序存储器（ROM）中的 ID 数据。
- 2、由于 STC8A8K64S4、STC8A8K64S2 这几个型号的 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 FLASH 程序存储器（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，Flash 程序存储器（ROM）中只有全球唯一 ID 号的数据，而内部 1.19V 参考信号源值、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。





## 7.4 只读特殊功能寄存器中存储的唯一 ID 号和重要参数 (CHIPID)

STC8A8K64D4 系列部分单片机内置有 32 字节的只读特殊功能寄存器 CHIPID。CHIPID 中内容, 用户程序只能读取, 不可修改。使用 CHIPID 中的数据对用户程序进行加密是 STC 官方推荐的最优方案。

### 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	
CHIPID00	硬件数字 ID00	FDE0H	全球唯一 ID 号 (第 0 字节)								nnnn,nnnn
CHIPID01	硬件数字 ID01	FDE1H	全球唯一 ID 号 (第 1 字节)								nnnn,nnnn
CHIPID02	硬件数字 ID02	FDE2H	全球唯一 ID 号 (第 2 字节)								nnnn,nnnn
CHIPID03	硬件数字 ID03	FDE3H	全球唯一 ID 号 (第 3 字节)								nnnn,nnnn
CHIPID04	硬件数字 ID04	FDE4H	全球唯一 ID 号 (第 4 字节)								nnnn,nnnn
CHIPID05	硬件数字 ID05	FDE5H	全球唯一 ID 号 (第 5 字节)								nnnn,nnnn
CHIPID06	硬件数字 ID06	FDE6H	全球唯一 ID 号 (第 6 字节)								nnnn,nnnn
CHIPID07	硬件数字 ID07	FDE7H	内部 1.19V 参考信号源 (高字节)								nnnn,nnnn
CHIPID08	硬件数字 ID08	FDE8H	内部 1.19V 参考信号源 (低字节)								nnnn,nnnn
CHIPID09	硬件数字 ID09	FDE9H	32K 掉电唤醒定时器的频率 (高字节)								nnnn,nnnn
CHIPID10	硬件数字 ID10	FDEAH	32K 掉电唤醒定时器的频率 (低字节)								nnnn,nnnn
CHIPID11	硬件数字 ID11	FDEBH	22.1184MHz 的 IRC 参数 (27M 频段)								nnnn,nnnn
CHIPID12	硬件数字 ID12	FDECH	24MHz 的 IRC 参数 (27M 频段)								nnnn,nnnn
CHIPID13	硬件数字 ID13	FDEDH	20MHz 的 IRC 参数 (27M 频段)								nnnn,nnnn
CHIPID14	硬件数字 ID14	FDEEH	27MHz 的 IRC 参数 (27M 频段)								nnnn,nnnn
CHIPID15	硬件数字 ID15	FDEFH	30MHz 的 IRC 参数 (27M 频段)								nnnn,nnnn
CHIPID16	硬件数字 ID16	FDF0H	33.1776MHz 的 IRC 参数 (27M 频段)								nnnn,nnnn
CHIPID17	硬件数字 ID17	FDF1H	35MHz 的 IRC 参数 (44M 频段)								nnnn,nnnn
CHIPID18	硬件数字 ID18	FDF2H	36.864MHz 的 IRC 参数 (44M 频段)								nnnn,nnnn
CHIPID19	硬件数字 ID19	FDF3H	40MHz 的 IRC 参数 (44M 频段)								nnnn,nnnn
CHIPID20	硬件数字 ID20	FDF4H	45MHz 的 IRC 参数 (44M 频段)								nnnn,nnnn
CHIPID21	硬件数字 ID21	FDF5H	6M 频段的 VRTRIM 参数								nnnn,nnnn
CHIPID22	硬件数字 ID22	FDF6H	10M 频段的 VRTRIM 参数								nnnn,nnnn
CHIPID23	硬件数字 ID23	FDF7H	27M 频段的 VRTRIM 参数								nnnn,nnnn
CHIPID24	硬件数字 ID24	FDF8H	44M 频段的 VRTRIM 参数								nnnn,nnnn
CHIPID25	硬件数字 ID25	FDF9H	00H								nnnn,nnnn
CHIPID26	硬件数字 ID26	FDFAH	用户程序空间结束地址 (高字节)								nnnn,nnnn
CHIPID27	硬件数字 ID27	FDFBH	芯片测试时间 (年)								nnnn,nnnn
CHIPID28	硬件数字 ID28	FDFCH	芯片测试时间 (月)								nnnn,nnnn
CHIPID29	硬件数字 ID29	FDFDH	芯片测试时间 (日)								nnnn,nnnn
CHIPID30	硬件数字 ID30	FDFEH	芯片封装形式编号								nnnn,nnnn
CHIPID31	硬件数字 ID31	FDFFH	5AH								nnnn,nnnn

## 7.4.1 CHIP 之全球唯一 ID 号解读

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID00	硬件数字 ID00	FDE0H	全球唯一 ID 号 (第 0 字节)								nnnn,nnnn
CHIPID01	硬件数字 ID01	FDE1H	全球唯一 ID 号 (第 1 字节)								nnnn,nnnn
CHIPID02	硬件数字 ID02	FDE2H	全球唯一 ID 号 (第 2 字节)								nnnn,nnnn
CHIPID03	硬件数字 ID03	FDE3H	全球唯一 ID 号 (第 3 字节)								nnnn,nnnn
CHIPID04	硬件数字 ID04	FDE4H	全球唯一 ID 号 (第 4 字节)								nnnn,nnnn
CHIPID05	硬件数字 ID05	FDE5H	全球唯一 ID 号 (第 5 字节)								nnnn,nnnn
CHIPID06	硬件数字 ID06	FDE6H	全球唯一 ID 号 (第 6 字节)								nnnn,nnnn

[CHIPID0, CHIPID1]: 16 位 MCU ID, 用于区别不同的单片机型号 (高位在前)。

STC8A8K64D4 系列常用的 MCU ID 如下表所示:

STC8A8K16D4 (F7F1)
STC8A8K32D4 (F7F2)
STC8A8K48D4 (F7F5)
STC8A8K60D4 (F7F3)
STC8A8K64D4 (F7F4)

[CHIPID2, CHIPID3]: 16 位测试机台编号 (高位在前)。

[CHIPID4, CHIPID5, CHIPID6]: 24 位测试流水编号 (高位在前)。

## 7.4.2 CHIP 之内部参考信号源解读

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID07	硬件数字 ID07	FDE7H	内部 1.19V 参考信号源 (高字节)								nnnn,nnnn
CHIPID08	硬件数字 ID08	FDE8H	内部 1.19V 参考信号源 (低字节)								nnnn,nnnn

[CHIPID7, CHIPID8]: 16 位内部参考信号源电压值 (高位在前)。

标准值为 1190 (04A6H), 单位为 mV, 即 1.19V。但实际的芯片由于存在制造误差。内部参考信号源的电压值并不会受工作电压 VCC 的影响, 所以内部参考信号源可以和 ADC 结合用于校准 ADC, 也可和比较器结合用于侦测工作电压。

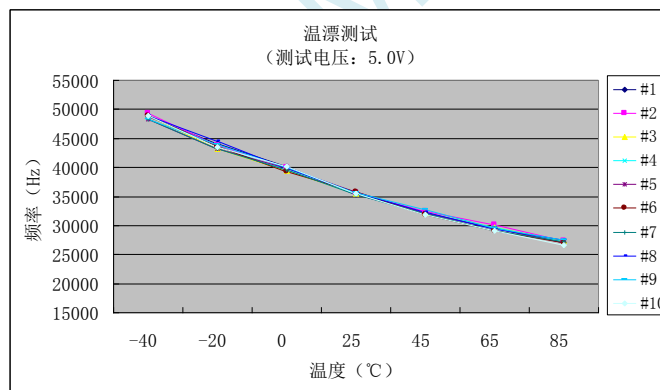
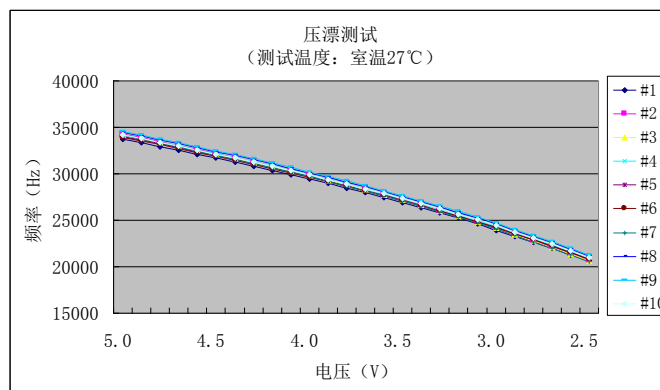
### 7.4.3 CHIP 之内部 32K 的 IRC 振荡频率解读

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID09	硬件数字 ID09	FDE9H	32K 掉电唤醒定时器的频率 (高字节)								nnnn,nnnn
CHIPID10	硬件数字 ID10	FDEAH	32K 掉电唤醒定时器的频率 (低字节)								nnnn,nnnn

[CHIPID9, CHIPID10]: 16 位 32K IRC 振荡器频率值 (高位在前)。

标准值为 32768 (8000H), 单位为 Hz, 即 32.768KHz。但实际的芯片由于存在制造误差, 而且温漂和压漂均比较大。

内部 32K 振荡器的压漂测试线性图和温漂线性图如下:



## 7.4.4 CHIP 之高精度 IRC 参数解读

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID11	硬件数字 ID11	FDEBH	22.1184MHz 的 IRC 参数 (27M 频段)								nnnn,nnnn
CHIPID12	硬件数字 ID12	FDECH	24MHz 的 IRC 参数 (27M 频段)								nnnn,nnnn
CHIPID13	硬件数字 ID13	FDEDH	20MHz 的 IRC 参数 (27M 频段)								nnnn,nnnn
CHIPID14	硬件数字 ID14	FDEEH	27MHz 的 IRC 参数 (27M 频段)								nnnn,nnnn
CHIPID15	硬件数字 ID15	FDEFH	30MHz 的 IRC 参数 (27M 频段)								nnnn,nnnn
CHIPID16	硬件数字 ID16	FDF0H	33.1776MHz 的 IRC 参数 (27M 频段)								nnnn,nnnn
CHIPID17	硬件数字 ID17	FDF1H	35MHz 的 IRC 参数 (44M 频段)								nnnn,nnnn
CHIPID18	硬件数字 ID18	FDF2H	36.864MHz 的 IRC 参数 (44M 频段)								nnnn,nnnn
CHIPID19	硬件数字 ID19	FDF3H	40MHz 的 IRC 参数 (44M 频段)								nnnn,nnnn
CHIPID20	硬件数字 ID20	FDF4H	45MHz 的 IRC 参数 (44M 频段)								nnnn,nnnn
CHIPID21	硬件数字 ID21	FDF5H	6M 频段的 VRTRIM 参数								nnnn,nnnn
CHIPID22	硬件数字 ID22	FDF6H	10M 频段的 VRTRIM 参数								nnnn,nnnn
CHIPID23	硬件数字 ID23	FDF7H	27M 频段的 VRTRIM 参数								nnnn,nnnn
CHIPID24	硬件数字 ID24	FDF8H	44M 频段的 VRTRIM 参数								nnnn,nnnn

STC8A8K64D4 系列单片机，内部集成的高精度 IRC 分 4 个频段，每个频段对应的参考电压值在出厂时已进行了校准，当选择不同的频段时，只需要将相应频段的电压校准值填入 VRTRIM 寄存器即可。

4 个频段的中心频率分别为 6MHz、10MHz、27MHz 和 44MHz，由于制造误差，中心频率一般可能有 ±5% 的偏差，为了得到精确的用户频率，可使用 IRTRIM 对频率进行微调校准。使用 STC 官方提供的下载软件下载用户程序时，系统会根据用户所设定频率自动设置 VRTRIM 和 IRTRIM 寄存器。同时，在 CHIPID 也内部预置了 10 个常用频率的 IRTRIM 值以及 4 个频段的参考电压值校准值，让用户可以在程序运行过程中动态的修改工作频率。

[CHIPID11 : CHIPID20]: 10 个常用频率的 IRTRIM 值。括号里面的注解即为对应的频段

[CHIPID21 : CHIPID24]: 4 个频段的参考电压值校准值。

用户动态修改频率时，只需要将[CHIPID11 : CHIPID20]中的某个频率校准值读出并写入 IRTRIM 寄存器，同时根据该频率所对应的频段将[CHIPID21 : CHIPID24]中的某个电压校准值读出并写入 VRTRIM 寄存器即可。详细操作请参考后续章节的范例程序。

## 7.4.5 CHIP 之测试时间参数解读

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID27	硬件数字 ID27	FDFBH	芯片测试时间 (年)								nnnn,nnnn
CHIPID28	硬件数字 ID28	FDFCH	芯片测试时间 (月)								nnnn,nnnn
CHIPID29	硬件数字 ID29	FDFDH	芯片测试时间 (日)								nnnn,nnnn

测试时间的年、月、日参数均为 BCD 码。(例如: CHIPID27=0x21, CHIPID28=0x11, CHIPID29=0x18, 则目标芯片的生产测试日期为 2021 年 11 月 18 日)

## 7.4.6 CHIP 之芯片封装形式编号解读

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID30	硬件数字 ID30	FDFEH	芯片封装形式编号								nnnn,nnnn

封装编号	封装形式	封装编号	封装形式
0x00	DIP8	0x50	SOP32
0x01	SOP8	0x51	LQFP32
0x02	DFN8	0x52	QFN32
0x10	DIP16	0x53	PLCC32
0x11	SOP16	0x54	QFN32S
0x20	DIP18	0x60	PDIP40
0x21	SOP18	0x70	LQFP44
0x30	DIP20	0x71	PLCC44
0x31	SOP20	0x72	PQFP44
0x32	TSSOP20	0x80	LQFP48
0x33	LSSOP20	0x81	QFN48
0x34	QFN20	0x90	LQFP64
0x40	SKDIP28	0x91	LQFP64S
0x41	SOP28	0x92	LQFP64L
0x42	TSSOP28	0x93	LQFP64M
0x43	QFN28	0x94	QFN64

## 7.5 范例程序

### 7.5.1 读取内部 1.19V 参考信号源值 (从 CHIPID 中读取)

#### 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)
```

```
#define CPUIDBASE 0xfde0
```

```
#define ID_ADDR ((unsigned char volatile xdata*)(CPUIDBASE + 0x00))
```

```
#define VREF_ADDR (*(unsigned int volatile xdata*)(CPUIDBASE + 0x07))
```

```
#define F32K_ADDR (*(unsigned int volatile xdata*)(CPUIDBASE + 0x09))
```

```
#define T22M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x0b)) //22.1184MHz
```

```
#define T24M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x0c)) //24MHz
```

```
#define T20M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x0d)) //20MHz
```

```
#define T27M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x0e)) //27MHz
```

```
#define T30M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x0f)) //30MHz
```

```
#define T33M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x10)) //33.1776MHz
```

```
#define T35M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x11)) //35MHz
```

```
#define T36M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x12)) //36.864MHz
```

```
#define T40M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x13)) //40MHz
```

```
#define T45M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x14)) //45MHz
```

```
#define VRT6M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x15)) //VRTRIM_6M
```

```
#define VRT10M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x16)) //VRTRIM_10M
```

```
#define VRT27M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x17)) //VRTRIM_27M
```

```
#define VRT44M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x18)) //VRTRIM_44M
```

```
sfr AUXR = 0x8e;
```

```
sfr P_SW2 = 0xba;
```

```
sfr P0M1 = 0x93;
```

```
sfr P0M0 = 0x94;
```

```
sfr P1M1 = 0x91;
```

```
sfr P1M0 = 0x92;
```

```
sfr P2M1 = 0x95;
```

```
sfr P2M0 = 0x96;
```

```
sfr P3M1 = 0xb1;
```

```
sfr P3M0 = 0xb2;
```

```
sfr P4M1 = 0xb3;
```

```
sfr P4M0 = 0xb4;
```

```
sfr P5M1 = 0xc9;
```

```
sfr P5M0 = 0xca;
```

```
bit busy;
```

```
void UartIsr() interrupt 4
```

```
{
```

```
    if (TI)
```

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

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    T1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

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

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

    UartInit();
    ES = 1;
    EA = 1;
    P_SW2 = 0x80;
    UartSend(VREF_ADDR >> 8); //读取内部1.19V 参考信号源的高字节
    UartSend(VREF_ADDR);     //读取内部1.19V 参考信号源的低字节

    while (1);
}
```

---

## 汇编代码

---

;测试工作频率为11.0592MHz

**CPUIDBASE EQU 0FDE0H**

<i>ID_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 00H</i>	
<i>VREF_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 07H</i>	
<i>F32K_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 09H</i>	
<i>T22M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 0BH</i>	;22.1184MHz
<i>T24M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 0CH</i>	;24MHz
<i>T20M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 0DH</i>	;20MHz
<i>T27M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 0EH</i>	;27MHz
<i>T30M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 0FH</i>	;30MHz
<i>T33M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 10H</i>	;33.1776MHz
<i>T35M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 11H</i>	;35MHz
<i>T36M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 12H</i>	;36.864MHz
<i>T40M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 13H</i>	;40MHz
<i>T45M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 14H</i>	;45MHz
<i>VRT6M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 15H</i>	;VRTRIM_6M
<i>VRT10M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 16H</i>	;VRTRIM_10M
<i>VRT27M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 17H</i>	;VRTRIM_27M
<i>VRT44M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 18H</i>	;VRTRIM_44M
<i>AUXR</i>	<i>DATA</i>	<i>8EH</i>	
<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>	
<i>BUSY</i>	<i>BIT</i>	<i>20H.0</i>	
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>	
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>	
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>	
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>	
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>	
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>	
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>	
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>	
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>	
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>	
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>	
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>	
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>0023H</i>	
	<i>LJMP</i>	<i>UART_ISR</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>UART_ISR:</i>			
	<i>JNB</i>	<i>TI,CHKRI</i>	
	<i>CLR</i>	<i>TI</i>	
	<i>CLR</i>	<i>BUSY</i>	
<i>CHKRI:</i>			
	<i>JNB</i>	<i>RI,UARTISR_EXIT</i>	
	<i>CLR</i>	<i>RI</i>	
<i>UARTISR_EXIT:</i>			
	<i>RETI</i>		
<i>UART_INIT:</i>			
	<i>MOV</i>	<i>SCON,#50H</i>	
	<i>MOV</i>	<i>TMOD,#00H</i>	
	<i>MOV</i>	<i>TL1,#0E8H</i>	;65536-11059200/115200/4=0FFE8H
	<i>MOV</i>	<i>TH1,#0FFH</i>	
	<i>SETB</i>	<i>TRI</i>	



```

MOV     AUXR,#40H
CLR     BUSY
RET

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

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

LCALL  UART_INIT
SETB   ES
SETB   EA

MOV    P_SW2,#80H
MOV    DPTR,#VREF_ADDR
CLR    A
MOVX   A,@DPTR           ;读取内部 1.19V 参考信号源的高字节
LCALL  UART_SEND
INC    DPTR
MOVX   A,@DPTR           ;读取内部 1.19V 参考信号源的低字节
LCALL  UART_SEND

LOOP:
JMP    LOOP

END

```

## 7.5.2 读取内部 1.19V 参考信号源值 (从 Flash 程序存储器 (ROM) 中读取)

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

```
bit    busy;
int    *BGV;
```

```
void UartIsr() interrupt 4
```

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

```
void UartInit()
```

```
{
    SCON = 0x50;
    TMOD = 0x00;
    T1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}
```

```
void UartSend(char dat)
```

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

```
void main()
```

```
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
}
```

```

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

BGV = (int code *)0xeff7; // STC8A8K60S4
UartInit();
ES = 1;
EA = 1;
UartSend(*BGV >> 8); //读取内部1.19V 参考信号源的高字节
UartSend(*BGV); //读取内部1.19V 参考信号源的低字节

while (1);
}

```

## 汇编代码

;测试工作频率为11.0592MHz

```

AUXR      DATA      8EH
BGV       EQU        0EFF7H           ;STC8A8K60S4

BUSY      BIT        20H.0

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

ORG       0000H
LJMP     MAIN
ORG       0023H
LJMP     UART_ISR

ORG       0100H

UART_ISR:
JNB     TI,CHKRI
CLR     TI
CLR     BUSY

CHKRI:
JNB     RI,UARTISR_EXIT
CLR     RI

UARTISR_EXIT:
RETI

UART_INIT:
MOV     SCON,#50H
MOV     TMOD,#00H
MOV     TL1,#0E8H           ;65536-11059200/115200/4=0FFE8H
MOV     TH1,#0FFH

```

```

        SETB     TRI
        MOV      AUXR,#40H
        CLR      BUSY
        RET

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

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

        LCALL   UART_INIT
        SETB     ES
        SETB     EA

        MOV      DPTR,#BGV
        CLR      A
        MOVC    A,@A+DPTR      ;读取内部 1.19V 参考信号源的高字节
        LCALL   UART_SEND
        MOV      A,#1
        MOVC    A,@A+DPTR      ;读取内部 1.19V 参考信号源的低字节
        LCALL   UART_SEND

LOOP:
        JMP      LOOP

        END

```

### 7.5.3 读取内部 1.19V 参考信号源值 (从 RAM 中读取)

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

```
bit busy;
int *BGV;
```

```
void UartIsr() interrupt 4
```

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

```
void UartInit()
```

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

```
void UartSend(char dat)
```

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

```
void main()
```

```
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
}
```

```

P5M0 = 0x00;
P5M1 = 0x00;

BGV = (int idata *)0xef;
UartInit();
ES = 1;
EA = 1;
UartSend(*BGV >> 8); //读取内部 1.19V 参考信号源的高字节
UartSend(*BGV);      //读取内部 1.19V 参考信号源的低字节

while (1);
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

AUXR	DATA	8EH	
BGV	DATA	0EFH	
BUSY	BIT	20H.0	
P0M1	DATA	093H	
P0M0	DATA	094H	
P1M1	DATA	091H	
P1M0	DATA	092H	
P2M1	DATA	095H	
P2M0	DATA	096H	
P3M1	DATA	0B1H	
P3M0	DATA	0B2H	
P4M1	DATA	0B3H	
P4M0	DATA	0B4H	
P5M1	DATA	0C9H	
P5M0	DATA	0CAH	
	ORG	0000H	
	LJMP	MAIN	
	ORG	0023H	
	LJMP	UART_ISR	
	ORG	0100H	
UART_ISR:			
	JNB	TI,CHKRI	
	CLR	TI	
	CLR	BUSY	
CHKRI:			
	JNB	RI,UARTISR_EXIT	
	CLR	RI	
UARTISR_EXIT:			
	RETI		
UART_INIT:			
	MOV	SCON,#50H	
	MOV	TMOD,#00H	
	MOV	TL1,#0E8H	;65536-11059200/115200/4=0FFE8H
	MOV	TH1,#0FFH	
	SETB	TRI	
	MOV	AUXR,#40H	

```

        CLR        BUSY
        RET

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

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

        LCALL    UART_INIT
        SETB     ES
        SETB     EA

        MOV      R0,#BGV
        MOV      A,@R0          ;读取内部 1.19V 参考信号源的高字节
        LCALL    UART_SEND
        INC      R0
        MOV      A,@R0          ;读取内部 1.19V 参考信号源的低字节
        LCALL    UART_SEND

LOOP:
        JMP      LOOP

        END

```

## 7.5.4 读取全球唯一 ID 号 (从 CHIPID 中读取)

### 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)

#define CPUIDBASE 0xfde0

#define ID_ADDR ((unsigned char volatile xdata *) (CPUIDBASE + 0x00))
#define VREF_ADDR (*(unsigned int volatile xdata *) (CPUIDBASE + 0x07))

```

```

#define F32K_ADDR      (*(unsigned int volatile xdata*)(CPUIDBASE + 0x09))
#define T22M_ADDR      (*(unsigned char volatile xdata*)(CPUIDBASE + 0x0b)) //22.1184MHz
#define T24M_ADDR      (*(unsigned char volatile xdata*)(CPUIDBASE + 0x0c)) //24MHz
#define T20M_ADDR      (*(unsigned char volatile xdata*)(CPUIDBASE + 0x0d)) //20MHz
#define T27M_ADDR      (*(unsigned char volatile xdata*)(CPUIDBASE + 0x0e)) //27MHz
#define T30M_ADDR      (*(unsigned char volatile xdata*)(CPUIDBASE + 0x0f)) //30MHz
#define T33M_ADDR      (*(unsigned char volatile xdata*)(CPUIDBASE + 0x10)) //33.1776MHz
#define T35M_ADDR      (*(unsigned char volatile xdata*)(CPUIDBASE + 0x11)) //35MHz
#define T36M_ADDR      (*(unsigned char volatile xdata*)(CPUIDBASE + 0x12)) //36.864MHz
#define T40M_ADDR      (*(unsigned char volatile xdata*)(CPUIDBASE + 0x13)) //40MHz
#define T45M_ADDR      (*(unsigned char volatile xdata*)(CPUIDBASE + 0x14)) //45MHz
#define VRT6M_ADDR     (*(unsigned char volatile xdata*)(CPUIDBASE + 0x15)) //VRTRIM_6M
#define VRT10M_ADDR    (*(unsigned char volatile xdata*)(CPUIDBASE + 0x16)) //VRTRIM_10M
#define VRT27M_ADDR    (*(unsigned char volatile xdata*)(CPUIDBASE + 0x17)) //VRTRIM_27M
#define VRT44M_ADDR    (*(unsigned char volatile xdata*)(CPUIDBASE + 0x18)) //VRTRIM_44M

```

```

sfr AUXR              = 0x8e;
sfr P_SW2             = 0xba;

```

```

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

```

```

bit busy;

```

```

void UartIsr() interrupt 4

```

```

{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

```

```

void UartInit()

```

```

{
    SCON = 0x50;
    TMOD = 0x00;
    T1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

```

```

void UartSend(char dat)

```



```

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

void main()
{
    char i;

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

    UartInit();
    ES = 1;
    EA = 1;

    P_SW2 = 0x80;
    for (i=0; i<7; i++)
    {
        UartSend(ID_ADDR[i]);
    }

    while (1);
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

<b>CPUIDBASE</b>	<b>EQU</b>	<b>0FDE0H</b>	
<b>ID_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 00H</b>	
<b>VREF_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 07H</b>	
<b>F32K_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 09H</b>	
<b>T22M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 0BH</b>	;22.1184MHz
<b>T24M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 0CH</b>	;24MHz
<b>T20M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 0DH</b>	;20MHz
<b>T27M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 0EH</b>	;27MHz
<b>T30M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 0FH</b>	;30MHz
<b>T33M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 10H</b>	;33.1776MHz
<b>T35M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 11H</b>	;35MHz
<b>T36M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 12H</b>	;36.864MHz
<b>T40M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 13H</b>	;40MHz
<b>T45M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 14H</b>	;45MHz
<b>VRT6M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 15H</b>	;VRTRIM_6M
<b>VRT10M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 16H</b>	;VRTRIM_10M
<b>VRT27M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 17H</b>	;VRTRIM_27M
<b>VRT44M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 18H</b>	;VRTRIM_44M

```

AUXR      DATA      8EH
P_SW2     DATA      0BAH

BUSY      BIT        20H.0

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

          ORG        0000H
          LJMP       MAIN
          ORG        0023H
          LJMP       UART_ISR

          ORG        0100H

UART_ISR:
          JNB        TI,CHKRI
          CLR        TI
          CLR        BUSY

CHKRI:
          JNB        RI,UARTISR_EXIT
          CLR        RI

UARTISR_EXIT:
          RETI

UART_INIT:
          MOV        SCON,#50H
          MOV        TMOD,#00H
          MOV        T1L,#0E8H           ;65536-11059200/115200/4=0FFE8H
          MOV        TH1,#0FFH
          SETB       TRI
          MOV        AUXR,#40H
          CLR        BUSY
          RET

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

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

```

```

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

        LCALL   UART_INIT
        SETB   ES
        SETB   EA

        MOV     P_SW2, #80H
        MOV     DPTR, #ID_ADDR
        MOV     RI, #7
NEXT:    CLR     A
        MOVX   A, @DPTR
        LCALL   UART_SEND
        INC    DPTR
        DJNZ   RI, NEXT

LOOP:   JMP     LOOP

        END

```

## 7.5.5 读取全球唯一 ID 号 (从 Flash 程序存储器 (ROM) 中读取)

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

bit busy;
char *ID;

```

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

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

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

void main()
{
    char i;

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

    ID = (char code *)0xeff9; // STC8A8K60S4
    UartInit();
    ES = 1;
    EA = 1;

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

    while (1);
}
```

}

---

**汇编代码**


---

;测试工作频率为11.0592MHz

```

AUXR      DATA      8EH
ID         EQU        0EFF9H          ; STC8A8K60S4

BUSY      BIT        20H.0

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

          ORG        0000H
          LJMP       MAIN
          ORG        0023H
          LJMP       UART_ISR

          ORG        0100H

UART_ISR:
          JNB        TI,CHKRI
          CLR        TI
          CLR        BUSY

CHKRI:
          JNB        RI,UARTISR_EXIT
          CLR        RI

UARTISR_EXIT:
          RETI

UART_INIT:
          MOV        SCON,#50H
          MOV        TMOD,#00H
          MOV        T1L,#0E8H          ;65536-11059200/115200/4=0FFE8H
          MOV        TH1,#0FFH
          SETB       TRI
          MOV        AUXR,#40H
          CLR        BUSY
          RET

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

MAIN:
          MOV        SP, #5FH

```

```

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

LCALL   UART_INIT
SETB   ES
SETB   EA

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

LOOP:

        JMP   LOOP

END

```

## 7.5.6 读取全球唯一 ID 号 (从 RAM 中读取)

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

```

```
bit    busy;
char   *ID;

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

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

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

void main()
{
    char i;

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

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

    for (i=0; i<7; i++)
    {
```

```

    UartSend(ID[i]);
}

while (1);
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

```

AUXR      DATA      8EH
ID        DATA      0F1H

BUSY      BIT        20H.0

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

          ORG        0000H
          LJMP       MAIN
          ORG        0023H
          LJMP       UART_ISR

          ORG        0100H

UART_ISR:
          JNB        TI,CHKRI
          CLR        TI
          CLR        BUSY

CHKRI:
          JNB        RI,UARTISR_EXIT
          CLR        RI

UARTISR_EXIT:
          RETI

UART_INIT:
          MOV        SCON,#50H
          MOV        TMOD,#00H
          MOV        T1L,#0E8H           ;65536-11059200/115200/4=0FFE8H
          MOV        TH1,#0FFH
          SETB       TRI
          MOV        AUXR,#40H
          CLR        BUSY
          RET

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

```



```

        RET

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

        LCALL   UART_INIT
        SETB    ES
        SETB    EA

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

LOOP:   JMP     LOOP

        END

```

## 7.5.7 读取 32K 掉电唤醒定时器的频率 (从 CHIPID 中读取)

### 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)
```

```
#define CPUIDBASE 0xfde0
```

```
#define ID_ADDR      ((unsigned char volatile xdata *) (CPUIDBASE + 0x00))
#define VREF_ADDR    ((unsigned int volatile xdata *) (CPUIDBASE + 0x07))
#define F32K_ADDR    ((unsigned int volatile xdata *) (CPUIDBASE + 0x09))
#define T22M_ADDR    ((unsigned char volatile xdata *) (CPUIDBASE + 0x0b)) //22.1184MHz
#define T24M_ADDR    ((unsigned char volatile xdata *) (CPUIDBASE + 0x0c)) //24MHz
#define T20M_ADDR    ((unsigned char volatile xdata *) (CPUIDBASE + 0x0d)) //20MHz
#define T27M_ADDR    ((unsigned char volatile xdata *) (CPUIDBASE + 0x0e)) //27MHz
#define T30M_ADDR    ((unsigned char volatile xdata *) (CPUIDBASE + 0x0f)) //30MHz
#define T33M_ADDR    ((unsigned char volatile xdata *) (CPUIDBASE + 0x10)) //33.1776MHz
```

```

#define T35M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x11)) //35MHz
#define T36M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x12)) //36.864MHz
#define T40M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x13)) //40MHz
#define T45M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x14)) //45MHz
#define VRT6M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x15)) //VRTRIM_6M
#define VRT10M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x16)) //VRTRIM_10M
#define VRT27M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x17)) //VRTRIM_27M
#define VRT44M_ADDR (*(unsigned char volatile xdata*)(CPUIDBASE + 0x18)) //VRTRIM_44M

```

```

sfr AUXR = 0x8e;
sfr P_SW2 = 0xba;

```

```

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

```

```

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 main()

```

```

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

    F32K = (int code *)0xeff5; // STC8A8K60S4
    UartInit();
    ES = 1;
    EA = 1;

    P_SW2 = 0x80;
    UartSend(F32K_ADDR >> 8); // 读取 32K 频率的高字节
    UartSend(F32K_ADDR); // 读取 32K 频率的低字节

    while (1);
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

<b>CPUIDBASE</b>	<b>EQU</b>	<b>0FDE0H</b>	
<b>ID_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 00H</b>	
<b>VREF_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 07H</b>	
<b>F32K_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 09H</b>	
<b>T22M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 0BH</b>	;22.1184MHz
<b>T24M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 0CH</b>	;24MHz
<b>T20M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 0DH</b>	;20MHz
<b>T27M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 0EH</b>	;27MHz
<b>T30M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 0FH</b>	;30MHz
<b>T33M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 10H</b>	;33.1776MHz
<b>T35M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 11H</b>	;35MHz
<b>T36M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 12H</b>	;36.864MHz
<b>T40M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 13H</b>	;40MHz
<b>T45M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 14H</b>	;45MHz
<b>VRT6M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 15H</b>	;VRTRIM_6M
<b>VRT10M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 16H</b>	;VRTRIM_10M
<b>VRT27M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 17H</b>	;VRTRIM_27M
<b>VRT44M_ADDR</b>	<b>EQU</b>	<b>CPUIDBASE + 18H</b>	;VRTRIM_44M
<b>AUXR</b>	<b>DATA</b>	<b>8EH</b>	
<b>P_SW2</b>	<b>DATA</b>	<b>0BAH</b>	
<b>BUSY</b>	<b>BIT</b>	<b>20H.0</b>	
<b>P0M1</b>	<b>DATA</b>	<b>093H</b>	
<b>P0M0</b>	<b>DATA</b>	<b>094H</b>	

```

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

          ORG          0000H
          LJMP         MAIN
          ORG          0023H
          LJMP         UART_ISR

          ORG          0100H

UART_ISR:
          JNB          TI,CHKRI
          CLR          TI
          CLR          BUSY

CHKRI:
          JNB          RI,UARTISR_EXIT
          CLR          RI

UARTISR_EXIT:
          RETI

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

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

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

```

```

        LCALL    UART_INIT
        SETB    ES
        SETB    EA

        MOV     P_SW2,#80H
        MOV     DPTR,# F32K_ADDR
        CLR     A
        MOVX    A,@DPTR                ;读取 32K 频率的高字节
        LCALL    UART_SEND
        INC     DPTR
        CLR     A
        MOVX    A,@DPTR                ;读取 32K 频率的低字节
        LCALL    UART_SEND

LOOP:
        JMP     LOOP

        END

```

## 7.5.8 读取 32K 掉电唤醒定时器的频率 (从 Flash 程序存储器 (ROM) 中读取)

### 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    P0M1    = 0x93;
```

```
sfr    P0M0    = 0x94;
```

```
sfr    P1M1    = 0x91;
```

```
sfr    P1M0    = 0x92;
```

```
sfr    P2M1    = 0x95;
```

```
sfr    P2M0    = 0x96;
```

```
sfr    P3M1    = 0xb1;
```

```
sfr    P3M0    = 0xb2;
```

```
sfr    P4M1    = 0xb3;
```

```
sfr    P4M0    = 0xb4;
```

```
sfr    P5M1    = 0xc9;
```

```
sfr    P5M0    = 0xca;
```

```
bit    busy;
int    *F32K;
```

```
void UartIsr() interrupt 4
{
    if (TI)
    {

```

```

        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

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

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

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

    F32K = (int code *)0xeff5; // STC8A8K60S4
    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);
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

```

AUXR      DATA      8EH
F32K      EQU        0EFF5H ; STC8A8K60S4

```

```

BUSY      BIT      20H.0

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

      ORG      0000H
      LJMP     MAIN
      ORG      0023H
      LJMP     UART_ISR

      ORG      0100H

UART_ISR:
      JNB      TI,CHKRI
      CLR      TI
      CLR      BUSY

CHKRI:
      JNB      RI,UARTISR_EXIT
      CLR      RI

UARTISR_EXIT:
      RETI

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

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

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

```

```

MOV    P4M0, #00H
MOV    P4M1, #00H
MOV    P5M0, #00H
MOV    P5M1, #00H

LCALL  UART_INIT
SETB   ES
SETB   EA

MOV    DPTR, #F32K
CLR    A
MOVC   A, @A+DPTR           ; 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

```

## 7.5.9 读取 32K 掉电唤醒定时器的频率 (从 RAM 中读取)

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

bit busy;
int *F32K;

void UartIsr() interrupt 4
{
    if (TI)

```



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

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    T1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

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

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

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

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

    while (1);
}
```

---

## 汇编代码

---

;测试工作频率为11.0592MHz

AUXR

DATA

8EH

```

F32K      DATA      0F8H

BUSY      BIT        20H.0

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

          ORG        0000H
          LJMP       MAIN
          ORG        0023H
          LJMP       UART_ISR

          ORG        0100H

UART_ISR:
          JNB        TI,CHKRI
          CLR        TI
          CLR        BUSY

CHKRI:
          JNB        RI,UARTISR_EXIT
          CLR        RI

UARTISR_EXIT:
          RETI

UART_INIT:
          MOV        SCON,#50H
          MOV        TMOD,#00H
          MOV        T1L,#0E8H           ;65536-11059200/115200/4=0FFE8H
          MOV        TH1,#0FFH
          SETB       TRI
          MOV        AUXR,#40H
          CLR        BUSY
          RET

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

MAIN:
          MOV        SP, #5FH
          MOV        P0M0, #00H
          MOV        P0M1, #00H
          MOV        P1M0, #00H
          MOV        P1M1, #00H
          MOV        P2M0, #00H
          MOV        P2M1, #00H
          MOV        P3M0, #00H

```

```

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

LCALL   UART_INIT
SETB    ES
SETB    EA

MOV      R0, #F32K
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

```

## 7.5.10 用户自定义内部 IRC 频率 (从 CHIPID 中读取)

### C 语言代码

```
// 测试工作频率为 11.0592MHz
```

```

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

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

#define CPUIDBASE   0xfde0

#define ID_ADDR      ((unsigned char volatile xdata *)(CPUIDBASE + 0x00))
#define VREF_ADDR    (*(unsigned int volatile xdata *)(CPUIDBASE + 0x07))
#define F32K_ADDR    (*(unsigned int volatile xdata *)(CPUIDBASE + 0x09))
#define T22M_ADDR    (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0b)) //22.1184MHz
#define T24M_ADDR    (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0c)) //24MHz
#define T20M_ADDR    (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0d)) //20MHz
#define T27M_ADDR    (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0e)) //27MHz
#define T30M_ADDR    (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0f)) //30MHz
#define T33M_ADDR    (*(unsigned char volatile xdata *)(CPUIDBASE + 0x10)) //33.1776MHz
#define T35M_ADDR    (*(unsigned char volatile xdata *)(CPUIDBASE + 0x11)) //35MHz
#define T36M_ADDR    (*(unsigned char volatile xdata *)(CPUIDBASE + 0x12)) //36.864MHz
#define T40M_ADDR    (*(unsigned char volatile xdata *)(CPUIDBASE + 0x13)) //40MHz
#define T45M_ADDR    (*(unsigned char volatile xdata *)(CPUIDBASE + 0x14)) //45MHz
#define VRT6M_ADDR   (*(unsigned char volatile xdata *)(CPUIDBASE + 0x15)) //VRTRIM_6M
#define VRT10M_ADDR  (*(unsigned char volatile xdata *)(CPUIDBASE + 0x16)) //VRTRIM_10M
#define VRT27M_ADDR  (*(unsigned char volatile xdata *)(CPUIDBASE + 0x17)) //VRTRIM_27M
#define VRT44M_ADDR  (*(unsigned char volatile xdata *)(CPUIDBASE + 0x18)) //VRTRIM_44M

sfr      P_SW2      = 0xba;
sfr      IRCBAND    = 0x9d;

```

```
sfr    IRTRIM    = 0x9f;
sfr    VRTRIM    = 0xa6;

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

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

    // //选择20MHz
    // P_SW2 = 0x80;
    // CLKDIV = 0x04;
    // IRTRIM = T20M_ADDR;
    // VRTRIM = VRT27M_ADDR;
    // IRCBAND = 0x02;
    // CLKDIV = 0x00;

    // //选择22.1184MHz
    // P_SW2 = 0x80;
    // CLKDIV = 0x04;
    // IRTRIM = T22M_ADDR;
    // VRTRIM = VRT27M_ADDR;
    // IRCBAND = 0x02;
    // CLKDIV = 0x00;

    // //选择24MHz
    // P_SW2 = 0x80;
    // CLKDIV = 0x04;
    // IRTRIM = T24M_ADDR;
    // VRTRIM = VRT27M_ADDR;
    // IRCBAND = 0x02;
    // CLKDIV = 0x00;

    // //选择27MHz
    // P_SW2 = 0x80;
    // CLKDIV = 0x04;
    // IRTRIM = T27M_ADDR;
```

```

// VRTRIM = VRT27M_ADDR;
// IRCBAND = 0x02;
// CLKDIV = 0x00;

// //选择30MHz
// P_SW2 = 0x80;
// CLKDIV = 0x04;
// IRTRIM = T30M_ADDR;
// VRTRIM = VRT27M_ADDR;
// IRCBAND = 0x02;
// CLKDIV = 0x00;

// //选择33.1776MHz
// P_SW2 = 0x80;
// CLKDIV = 0x04;
// IRTRIM = T33M_ADDR;
// VRTRIM = VRT27M_ADDR;
// IRCBAND = 0x02;
// CLKDIV = 0x00;

// //选择35MHz
// P_SW2 = 0x80;
// CLKDIV = 0x04;
// IRTRIM = T35M_ADDR;
// VRTRIM = VRT44M_ADDR;
// IRCBAND = 0x03;
// CLKDIV = 0x00;

// //选择40MHz
// P_SW2 = 0x80;
// CLKDIV = 0x04;
// IRTRIM = T40M_ADDR;
// VRTRIM = VRT44M_ADDR;
// IRCBAND = 0x03;
// CLKDIV = 0x00;

// //选择45MHz
// P_SW2 = 0x80;
// CLKDIV = 0x04;
// IRTRIM = T45M_ADDR;
// VRTRIM = VRT44M_ADDR;
// IRCBAND = 0x03;
// CLKDIV = 0x00;

while (1);
}

```

## 汇编代码

;测试工作频率为11.0592MHz

**CPUIDBASE EQU 0FDE0H**

**ID\_ADDR EQU CPUIDBASE + 00H**

**VREF\_ADDR EQU CPUIDBASE + 07H**

**F32K\_ADDR EQU CPUIDBASE + 09H**

**T22M\_ADDR EQU CPUIDBASE + 0BH ;22.1184MHz**

**T24M\_ADDR EQU CPUIDBASE + 0CH ;24MHz**

**T20M\_ADDR EQU CPUIDBASE + 0DH ;20MHz**

<i>T27M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 0EH</i>	<i>;27MHz</i>
<i>T30M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 0FH</i>	<i>;30MHz</i>
<i>T33M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 10H</i>	<i>;33.1776MHz</i>
<i>T35M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 11H</i>	<i>;35MHz</i>
<i>T36M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 12H</i>	<i>;36.864MHz</i>
<i>T40M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 13H</i>	<i>;40MHz</i>
<i>T45M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 14H</i>	<i>;45MHz</i>
<i>VRT6M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 15H</i>	<i>;VRTRIM_6M</i>
<i>VRT10M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 16H</i>	<i>;VRTRIM_10M</i>
<i>VRT27M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 17H</i>	<i>;VRTRIM_27M</i>
<i>VRT44M_ADDR</i>	<i>EQU</i>	<i>CPUIDBASE + 18H</i>	<i>;VRTRIM_44M</i>

<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>
<i>CLKSEL</i>	<i>EQU</i>	<i>0FE00H</i>
<i>CLKDIV</i>	<i>EQU</i>	<i>0FE01H</i>

<i>IRCBAND</i>	<i>DATA</i>	<i>09DH</i>
<i>IRCTRIM</i>	<i>DATA</i>	<i>09FH</i>
<i>VRTRIM</i>	<i>DATA</i>	<i>0A6H</i>

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

<i>ORG</i>	<i>0000H</i>
<i>LJMP</i>	<i>MAIN</i>

<i>ORG</i>	<i>0100H</i>
------------	--------------

*MAIN:*

<i>MOV</i>	<i>SP, #5FH</i>
<i>MOV</i>	<i>P0M0, #00H</i>
<i>MOV</i>	<i>P0M1, #00H</i>
<i>MOV</i>	<i>P1M0, #00H</i>
<i>MOV</i>	<i>P1M1, #00H</i>
<i>MOV</i>	<i>P2M0, #00H</i>
<i>MOV</i>	<i>P2M1, #00H</i>
<i>MOV</i>	<i>P3M0, #00H</i>
<i>MOV</i>	<i>P3M1, #00H</i>
<i>MOV</i>	<i>P4M0, #00H</i>
<i>MOV</i>	<i>P4M1, #00H</i>
<i>MOV</i>	<i>P5M0, #00H</i>
<i>MOV</i>	<i>P5M1, #00H</i>

<i>;</i>	<i>;选择 20MHz</i>
<i>;</i>	<i>MOV P_SW2, #80H</i>
<i>;</i>	<i>MOV A, #4</i>
<i>;</i>	<i>MOV DPTR, #CLKDIV</i>
<i>;</i>	<i>MOVX @DPTR, A</i>
<i>;</i>	<i>MOV DPTR, #T20M_ADDR</i>
<i>;</i>	<i>CLR A</i>

```

;      MOVX      A,@DPTR
;      MOV       IRTRIM,A
;      MOV       DPTR,#VRT27M_ADDR
;      CLR       A
;      MOVX      A,@DPTR
;      MOV       VRTRIM,A
;      MOV       IRCBAND,#02H
;      MOV       A,#0
;      MOV       DPTR,#CLKDIV
;      MOVX      @DPTR,A
;      MOV       P_SW2,#00H

```

```

;      ;选择22.1184MHz
;      MOV       P_SW2,#80H
;      MOV       A,#4
;      MOV       DPTR,#CLKDIV
;      MOVX      @DPTR,A
;      MOV       DPTR,#T22M_ADDR
;      CLR       A
;      MOVX      A,@DPTR
;      MOV       IRTRIM,A
;      MOV       DPTR,#VRT27M_ADDR
;      CLR       A
;      MOVX      A,@DPTR
;      MOV       VRTRIM,A
;      MOV       IRCBAND,#02H
;      MOV       A,#0
;      MOV       DPTR,#CLKDIV
;      MOVX      @DPTR,A
;      MOV       P_SW2,#00H

```

```

;      ;选择24MHz
;      MOV       P_SW2,#80H
;      MOV       A,#4
;      MOV       DPTR,#CLKDIV
;      MOVX      @DPTR,A
;      MOV       DPTR,#T24M_ADDR
;      CLR       A
;      MOVX      A,@DPTR
;      MOV       IRTRIM,A
;      MOV       DPTR,#VRT27M_ADDR
;      CLR       A
;      MOVX      A,@DPTR
;      MOV       VRTRIM,A
;      MOV       IRCBAND,#02H
;      MOV       A,#0
;      MOV       DPTR,#CLKDIV
;      MOVX      @DPTR,A
;      MOV       P_SW2,#00H

```

```

;      ;选择27MHz
;      MOV       P_SW2,#80H
;      MOV       A,#4
;      MOV       DPTR,#CLKDIV
;      MOVX      @DPTR,A
;      MOV       DPTR,#T27M_ADDR
;      CLR       A
;      MOVX      A,@DPTR
;      MOV       IRTRIM,A

```

```

;      MOV      DPTR,#VRT27M_ADDR
;      CLR      A
;      MOVX     A,@DPTR
;      MOV      VRTRIM,A
;      MOV      IRCBAND,#02H
;      MOV      A,#0
;      MOV      DPTR,#CLKDIV
;      MOVX     @DPTR,A
;      MOV      P_SW2,#00H

;      ;选择30MHz
;      MOV      P_SW2,#80H
;      MOV      A,#4
;      MOV      DPTR,#CLKDIV
;      MOVX     @DPTR,A
;      MOV      DPTR,#T30M_ADDR
;      CLR      A
;      MOVX     A,@DPTR
;      MOV      IRTRIM,A
;      MOV      DPTR,#VRT27M_ADDR
;      CLR      A
;      MOVX     A,@DPTR
;      MOV      VRTRIM,A
;      MOV      IRCBAND,#02H
;      MOV      A,#0
;      MOV      DPTR,#CLKDIV
;      MOVX     @DPTR,A
;      MOV      P_SW2,#00H

;      ;选择33.1776MHz
;      MOV      P_SW2,#80H
;      MOV      A,#4
;      MOV      DPTR,#CLKDIV
;      MOVX     @DPTR,A
;      MOV      DPTR,#T33M_ADDR
;      CLR      A
;      MOVX     A,@DPTR
;      MOV      IRTRIM,A
;      MOV      DPTR,#VRT27M_ADDR
;      CLR      A
;      MOVX     A,@DPTR
;      MOV      VRTRIM,A
;      MOV      IRCBAND,#02H
;      MOV      A,#0
;      MOV      DPTR,#CLKDIV
;      MOVX     @DPTR,A
;      MOV      P_SW2,#00H

;      ;选择35MHz
;      MOV      P_SW2,#80H
;      MOV      A,#4
;      MOV      DPTR,#CLKDIV
;      MOVX     @DPTR,A
;      MOV      DPTR,#T35M_ADDR
;      CLR      A
;      MOVX     A,@DPTR
;      MOV      IRTRIM,A
;      MOV      DPTR,#VRT44M_ADDR
;      CLR      A

```



```

;      MOVX      A,@DPTR
;      MOV       VRTRIM,A
;      MOV       IRCBAND,#03H
;      MOV       A,#0
;      MOV       DPTR,#CLKDIV
;      MOVX     @DPTR,A
;      MOV       P_SW2,#00H

;
; ;选择 36.864MHz
;      MOV       P_SW2,#80H
;      MOV       A,#4
;      MOV       DPTR,#CLKDIV
;      MOVX     @DPTR,A
;      MOV       DPTR,#T36M_ADDR
;      CLR      A
;      MOVX     A,@DPTR
;      MOV       IRTRIM,A
;      MOV       DPTR,#VRT44M_ADDR
;      CLR      A
;      MOVX     A,@DPTR
;      MOV       VRTRIM,A
;      MOV       IRCBAND,#03H
;      MOV       A,#0
;      MOV       DPTR,#CLKDIV
;      MOVX     @DPTR,A
;      MOV       P_SW2,#00H

;
; ;选择 40MHz
;      MOV       P_SW2,#80H
;      MOV       A,#4
;      MOV       DPTR,#CLKDIV
;      MOVX     @DPTR,A
;      MOV       DPTR,#T40M_ADDR
;      CLR      A
;      MOVX     A,@DPTR
;      MOV       IRTRIM,A
;      MOV       DPTR,#VRT44M_ADDR
;      CLR      A
;      MOVX     A,@DPTR
;      MOV       VRTRIM,A
;      MOV       IRCBAND,#03H
;      MOV       A,#0
;      MOV       DPTR,#CLKDIV
;      MOVX     @DPTR,A
;      MOV       P_SW2,#00H

;
; ;选择 45MHz
;      MOV       P_SW2,#80H
;      MOV       A,#4
;      MOV       DPTR,#CLKDIV
;      MOVX     @DPTR,A
;      MOV       DPTR,#T36M_ADDR
;      CLR      A
;      MOVX     A,@DPTR
;      MOV       IRTRIM,A
;      MOV       DPTR,#VRT45M_ADDR
;      CLR      A
;      MOVX     A,@DPTR
;      MOV       VRTRIM,A

```