

STC16F 系列单片机 技术参考手册

STC MCU

目录

1	概述.....	1
2	特性及价格.....	2
2.1	STC16F40K128 系列特性及价格.....	2
3	管脚及说明.....	5
3.1	管脚图.....	5
3.1.1	STC16F40K128 系列管脚图.....	5
3.2	管脚说明.....	7
3.2.1	STC16F40K128 系列管脚说明.....	7
3.3	功能脚切换.....	14
3.3.1	功能脚切换相关寄存器.....	14
3.3.2	外设端口切换控制寄存器 1 (P_SW1).....	14
3.3.3	外设端口切换控制寄存器 2 (P_SW2).....	15
3.3.4	时钟选择寄存器 (MCLKOCR).....	15
3.3.5	T3/T4 选择寄存器 (T3T4PIN).....	15
3.3.6	高级 PWM 选择寄存器 (PWMn_PS).....	16
3.3.7	高级 PWM 功能脚选择寄存器 (PWMx_ETRPS).....	17
3.4	范例程序.....	19
3.4.1	串口 1 切换.....	19
3.4.2	串口 2 切换.....	20
3.4.3	串口 3 切换.....	21
3.4.4	串口 4 切换.....	22
3.4.5	SPI 切换.....	23
3.4.6	I2C 切换.....	24
3.4.7	比较器输出切换.....	25
3.4.8	主时钟输出切换.....	26
4	封装尺寸图.....	28
4.1	LQFP64S 封装尺寸图 (12mm*12mm).....	28
4.2	LQFP48 封装尺寸图 (9mm*9mm).....	29
4.3	QFN64 封装尺寸图 (8mm*8mm) (暂无).....	30
4.4	STC16F 系列单片机命名规则.....	31
5	开发环境的建立与 ISP 下载.....	32
5.1	安装 Keil.....	32
5.2	添加型号和头文件到 Keil.....	35
5.3	使用 ISP 进行烧录.....	37
5.4	ISP 下载典型应用线路图.....	39
5.4.1	使用 PL2303-GL 下载.....	39
5.4.2	使用通用 USB 转串口工具下载.....	40
5.4.3	使用 U8-Mini 工具下载, 支持 ISP 在线和脱机下载.....	41
5.4.4	使用 U8W 工具下载, 支持 ISP 在线和脱机下载.....	42
5.4.5	U8W 直通模式, 可用于仿真、串口通信.....	43

5.4.6	硬件 USB 直接 ISP 下载, 仅支持 ISP 下载, 暂不支持仿真	44
6	时钟、复位与电源管理	46
6.1	系统时钟控制	46
6.2	相关寄存器	46
6.2.1	系统时钟选择寄存器 (CKSEL)	46
6.2.2	时钟分频寄存器 (CLKDIV)	47
6.2.3	内部高精度 IRC 控制寄存器 (IRC24MCR)	47
6.2.4	外部振荡器控制寄存器 (XOSCCR)	47
6.2.5	内部 32KHz 低速 IRC 控制寄存器 (IRC32KCR)	47
6.2.6	内部 PLL 控制寄存器 (PLLCR)	48
6.2.7	主时钟输出控制寄存器 (MCLKOCR)	48
6.3	STC16F 系列内部 IRC 频率调整	50
6.3.1	IRC 频段选择寄存器 (IRCBAND)	50
6.3.2	内部 IRC 频率调整寄存器 (IRTRIM)	50
6.3.3	内部 IRC 频率微调寄存器 (LIRTRIM)	51
6.3.4	时钟分频寄存器 (CLKDIV)	51
6.4	系统复位	52
6.4.1	看门狗控制寄存器 (WDT_CONTR)	52
6.4.2	IAP 控制寄存器 (IAP_CONTR)	53
6.4.3	复位配置寄存器 (RSTCFG)	53
6.4.4	低电平上电复位参考电路 (一般不需要)	54
6.4.5	低电平按键复位参考电路	54
6.4.6	传统 8051 高电平上电复位参考电路	55
6.5	外部晶振及外部时钟电路	56
6.5.1	外部晶振输入电路	56
6.5.2	外部时钟输入电路 (P1.6 不可用作普通 I/O)	56
6.6	系统电源管理	57
6.6.1	电源控制寄存器 (PCON)	57
6.7	掉电唤醒定时器	58
6.7.1	掉电唤醒定时器计数寄存器 (WKTCL, WKTCH)	58
6.8	范例程序	60
6.8.1	选择系统时钟源	60
6.8.2	主时钟分频输出	62
6.8.3	看门狗定时器应用	63
6.8.4	软复位实现自定义下载	64
6.8.5	低压检测	65
6.8.6	省电模式	67
6.8.7	使用 INT0/INT1/INT2/INT3/INT4 管脚中断唤醒省电模式	69
6.8.8	使用 T0/T1/T2/T3/T4 管脚中断唤醒省电模式	72
6.8.9	使用 RxD/RxD2/RxD3/RxD4 管脚中断唤醒省电模式	75
6.8.10	使用 I2C 的 SDA 脚唤醒 MCU 省电模式	78
6.8.11	使用掉电唤醒定时器唤醒省电模式	80
6.8.12	LVD 中断唤醒省电模式, 建议配合使用掉电唤醒定时器	81
6.8.13	比较器中断唤醒省电模式, 建议配合使用掉电唤醒定时器	83

6.8.14	使用 LVD 功能检测工作电压（电池电压）	85
7	存储器	90
7.1	程序存储器	91
7.2	数据存储器	92
7.2.1	内部 RAM	92
7.2.2	程序状态寄存器（PSW）	94
7.2.3	内部扩展 RAM	95
7.2.4	辅助寄存器（AUXR）	95
7.2.5	外部扩展 RAM, XRAM, XDATA	95
7.2.6	总线速度控制寄存器（BUS_SPEED）	95
8	特殊功能寄存器	97
8.1	STC16F40K128 系列	97
8.2	特殊功能寄存器列表	98
9	I/O 口	105
9.1	I/O 口相关寄存器	105
9.1.1	端口数据寄存器（Px）	107
9.1.2	端口模式配置寄存器（PxM0, PxM1）	107
9.1.3	端口上拉电阻控制寄存器（PxPU）	108
9.1.4	端口施密特触发控制寄存器（PxNCS）	108
9.1.5	端口电平转换速度控制寄存器（PxSR）	108
9.1.6	端口驱动电流控制寄存器（PxDR）	109
9.1.7	端口数字信号输入使能控制寄存器（PxIE）	109
9.2	配置 I/O 口	110
9.3	I/O 的结构图	111
9.3.1	准双向口（弱上拉）	111
9.3.2	推挽输出	111
9.3.3	高阻输入	111
9.3.4	开漏输出	112
9.3.5	新增 4.1K 上拉电阻	113
9.3.6	如何设置 I/O 口对外输出速度	113
9.3.7	如何设置 I/O 口电流驱动能力	114
9.3.8	如何降低 I/O 口对外辐射	114
9.4	范例程序	115
9.4.1	端口模式设置	115
9.4.2	双向口读写操作	116
9.5	一种典型三极管控制电路	118
9.6	典型发光二极管控制电路	118
9.7	混合电压供电系统 3V/5V 器件 I/O 口互连	119
9.8	如何让 I/O 口上电复位时为低电平	120
9.9	利用 74HC595 驱动 8 个数码管(串行扩展,3 根线)的线路图	121
10	中断系统	122
10.1	STC16F 系列中断源	122
10.2	STC16F 中断结构图	124
10.3	STC16F 系列中断列表	125

10.4	中断相关寄存器	127
10.4.1	中断使能寄存器（中断允许位）	127
10.4.2	中断请求寄存器（中断标志位）	131
10.4.3	中断优先级寄存器	133
10.5	范例程序	136
10.5.1	INT0 中断（上升沿和下降沿），可同时支持上升沿和下降沿	136
10.5.2	INT0 中断（下降沿）	137
10.5.3	INT1 中断（上升沿和下降沿），可同时支持上升沿和下降沿	139
10.5.4	INT1 中断（下降沿）	140
10.5.5	INT2 中断（下降沿），只支持下降沿中断	141
10.5.6	INT3 中断（下降沿），只支持下降沿中断	143
10.5.7	INT4 中断（下降沿），只支持下降沿中断	144
10.5.8	定时器 0 中断	146
10.5.9	定时器 1 中断	147
10.5.10	定时器 2 中断	148
10.5.11	定时器 3 中断	150
10.5.12	定时器 4 中断	151
10.5.13	UART1 中断	153
10.5.14	UART2 中断	155
10.5.15	UART3 中断	156
10.5.16	UART4 中断	158
10.5.17	ADC 中断	160
10.5.18	LVD 中断	162
10.5.19	比较器中断	163
10.5.20	SPI 中断	165
10.5.21	I2C 中断	166
11	定时器/计数器	169
11.1	定时器的相关寄存器	169
11.2	定时器 0/1	171
11.2.1	定时器 0/1 控制寄存器（TCON）	171
11.2.2	定时器 0/1 模式寄存器（TMOD）	171
11.2.3	定时器 0 模式 0（16 位自动重载模式）	172
11.2.4	定时器 0 模式 1（16 位不可重载模式）	173
11.2.5	定时器 0 模式 2（8 位自动重载模式）	174
11.2.6	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）	174
11.2.7	定时器 1 模式 0（16 位自动重载模式）	175
11.2.8	定时器 1 模式 1（16 位不可重载模式）	176
11.2.9	定时器 1 模式 2（8 位自动重载模式）	176
11.2.10	定时器 0 计数寄存器（TL0，TH0）	177
11.2.11	定时器 1 计数寄存器（TL1，TH1）	177
11.2.12	辅助寄存器 1（AUXR）	177
11.2.13	中断与时钟输出控制寄存器（INTCLKO）	178
11.2.14	定时器 0 计算公式	179
11.2.15	定时器 1 计算公式	179

11.3	定时器 2.....	180
11.3.1	辅助寄存器 1 (AUXR)	180
11.3.2	中断与时钟输出控制寄存器 (INTCLKO)	180
11.3.3	定时器 2 计数寄存器 (T2L, T2H)	180
11.3.4	定时器 2 工作模式	181
11.3.5	定时器 2 计算公式	181
11.4	定时器 3/4.....	182
11.4.1	定时器 3/4 功能脚切换	182
11.4.2	定时器 4/3 控制寄存器 (T4T3M)	182
11.4.3	定时器 3 计数寄存器 (T3L, T3H)	182
11.4.4	定时器 4 计数寄存器 (T4L, T4H)	183
11.4.5	定时器 3 工作模式	183
11.4.6	定时器 4 工作模式	184
11.4.7	定时器 3 计算公式	184
11.4.8	定时器 4 计算公式	184
11.5	范例程序	185
11.5.1	定时器 0 (模式 0—16 位自动重载), 用作定时	185
11.5.2	定时器 0 (模式 1—16 位不自动重载), 用作定时	186
11.5.3	定时器 0 (模式 2—8 位自动重载), 用作定时	188
11.5.4	定时器 0 (模式 3—16 位自动重载不可屏蔽中断), 用作定时	189
11.5.5	定时器 0 (外部计数—扩展 T0 为外部下降沿中断)	190
11.5.6	定时器 0 (测量脉宽—INT0 高电平宽度)	192
11.5.7	定时器 0 (模式 0), 时钟分频输出	193
11.5.8	定时器 1 (模式 0—16 位自动重载), 用作定时	195
11.5.9	定时器 1 (模式 1—16 位不自动重载), 用作定时	196
11.5.10	定时器 1 (模式 2—8 位自动重载), 用作定时	197
11.5.11	定时器 1 (外部计数—扩展 T1 为外部下降沿中断)	199
11.5.12	定时器 1 (测量脉宽—INT1 高电平宽度)	200
11.5.13	定时器 1 (模式 0), 时钟分频输出	202
11.5.14	定时器 1 (模式 0) 做串口 1 波特率发生器	203
11.5.15	定时器 1 (模式 2) 做串口 1 波特率发生器	206
11.5.16	定时器 2 (16 位自动重载), 用作定时	210
11.5.17	定时器 2 (外部计数—扩展 T2 为外部下降沿中断)	211
11.5.18	定时器 2, 时钟分频输出	213
11.5.19	定时器 2 做串口 1 波特率发生器	214
11.5.20	定时器 2 做串口 2 波特率发生器	217
11.5.21	定时器 2 做串口 3 波特率发生器	221
11.5.22	定时器 2 做串口 4 波特率发生器	224
11.5.23	定时器 3 (16 位自动重载), 用作定时	227
11.5.24	定时器 3 (外部计数—扩展 T3 为外部下降沿中断)	229
11.5.25	定时器 3, 时钟分频输出	230
11.5.26	定时器 3 做串口 3 波特率发生器	232
11.5.27	定时器 4 (16 位自动重载), 用作定时	235
11.5.28	定时器 4 (外部计数—扩展 T4 为外部下降沿中断)	236

11.5.29	定时器 4, 时钟分频输出	238
11.5.30	定时器 4 做串口 4 波特率发生器	239
12	串口通信	243
12.1	串口功能脚切换	243
12.2	串口相关寄存器	243
12.3	串口 1	245
12.3.1	串口 1 控制寄存器 (SCON)	245
12.3.2	串口 1 数据寄存器 (SBUF)	245
12.3.3	电源管理寄存器 (PCON)	246
12.3.4	辅助寄存器 1 (AUXR)	246
12.3.5	串口 1 模式 0, 模式 0 波特率计算公式	246
12.3.6	串口 1 模式 1, 模式 1 波特率计算公式	247
12.3.7	串口 1 模式 2, 模式 2 波特率计算公式	250
12.3.8	串口 1 模式 3, 模式 3 波特率计算公式	250
12.3.9	自动地址识别	251
12.3.10	串口 1 从机地址控制寄存器 (SADDR, SADEN)	251
12.4	串口 2	253
12.4.1	串口 2 控制寄存器 (S2CON)	253
12.4.2	串口 2 数据寄存器 (S2BUF)	253
12.4.3	串口 2 模式 0, 模式 0 波特率计算公式	253
12.4.4	串口 2 模式 1, 模式 1 波特率计算公式	254
12.5	串口 3	256
12.5.1	串口 3 控制寄存器 (S3CON)	256
12.5.2	串口 3 数据寄存器 (S3BUF)	256
12.5.3	串口 3 模式 0, 模式 0 波特率计算公式	256
12.5.4	串口 3 模式 1, 模式 1 波特率计算公式	257
12.6	串口 4	259
12.6.1	串口 4 控制寄存器 (S4CON)	259
12.6.2	串口 4 数据寄存器 (S4BUF)	259
12.6.3	串口 4 模式 0, 模式 0 波特率计算公式	259
12.6.4	串口 4 模式 1, 模式 1 波特率计算公式	260
12.7	串口注意事项	262
12.8	范例程序	263
12.8.1	串口 1 使用定时器 2 做波特率发生器	263
12.8.2	串口 1 使用定时器 1 (模式 0) 做波特率发生器	266
12.8.3	串口 1 使用定时器 1 (模式 2) 做波特率发生器	269
12.8.4	串口 2 使用定时器 2 做波特率发生器	273
12.8.5	串口 3 使用定时器 2 做波特率发生器	276
12.8.6	串口 3 使用定时器 3 做波特率发生器	279
12.8.7	串口 4 使用定时器 2 做波特率发生器	283
12.8.8	串口 4 使用定时器 4 做波特率发生器	286
13	比较器, 掉电检测, 内部固定比较电压	290
13.1	比较器内部结构图	290
13.2	比较器功能脚切换	290

13.3	比较器相关的寄存器	291
13.3.1	比较器控制寄存器 1 (CMPCR1)	291
13.3.2	比较器控制寄存器 2 (CMPCR2)	291
13.4	范例程序	293
13.4.1	比较器的使用 (中断方式)	293
13.4.2	比较器的使用 (查询方式)	295
13.4.3	比较器的多路复用应用 (比较器+ADC 输入通道)	297
13.4.4	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)	298
13.4.5	比较器检测工作电压 (电池电压)	299
14	IAP/EEPROM, 暂不支持	303
14.1	EEPROM 相关的寄存器	303
14.1.1	EEPROM 数据寄存器 (IAP_DATA)	303
14.1.2	EEPROM 地址寄存器 (IAP_ADDR)	303
14.1.3	EEPROM 命令寄存器 (IAP_CMD)	304
14.1.4	EEPROM 触发寄存器 (IAP_TRIG)	304
14.1.5	EEPROM 控制寄存器 (IAP_CONTR)	304
14.1.6	EEPROM 擦除等待时间控制寄存器 (IAP_TPS)	304
15	ADC 模数转换	306
15.1	ADC 相关的寄存器	306
15.1.1	ADC 控制寄存器 (ADC_CONTR)	306
15.1.2	ADC 配置寄存器 (ADCCFG)	307
15.1.3	ADC 转换结果寄存器 (ADC_RES, ADC_RES1)	308
15.1.4	ADC 时序控制寄存器 (ADCTIM)	308
15.2	ADC 静态特性	310
15.3	ADC 相关计算公式	310
15.3.1	ADC 速度计算公式	310
15.3.2	ADC 转换结果计算公式	310
15.3.3	反推 ADC 输入电压计算公式	310
15.3.4	反推工作电压计算公式	311
15.4	ADC 应用参考线路图	312
15.4.1	一般精度 ADC 参考线路图	312
15.4.2	高精度 ADC 参考线路图	312
15.5	范例程序	313
15.5.1	ADC 基本操作 (查询方式)	313
15.5.2	ADC 基本操作 (中断方式)	314
15.5.3	格式化 ADC 转换结果	316
15.5.4	ADC 作按键扫描应用线路图	318
15.5.5	检测负电压参考线路图	319
15.5.6	常用加法电路在 ADC 中的应用	320
16	同步串行外设接口 SPI	321
16.1	SPI 功能脚切换	321
16.2	SPI 相关的寄存器	321
16.2.1	SPI 状态寄存器 (SPSTAT)	321
16.2.2	SPI 控制寄存器 (SPCTL), SPI 速度控制	321

16.2.3	SPI 数据寄存器 (SPDAT)	322
16.3	SPI 通信方式	323
16.3.1	单主单从	323
16.3.2	互为主从	323
16.3.3	单主多从	324
16.4	配置 SPI	325
16.5	数据模式	327
16.6	范例程序	328
16.6.1	SPI 单主单从系统主机程序 (中断方式)	328
16.6.2	SPI 单主单从系统从机程序 (中断方式)	330
16.6.3	SPI 单主单从系统主机程序 (查询方式)	331
16.6.4	SPI 单主单从系统从机程序 (查询方式)	333
16.6.5	SPI 互为主从系统程序 (中断方式)	334
16.6.6	SPI 互为主从系统程序 (查询方式)	337
17	I²C 总线	340
17.1	I ² C 功能脚切换	340
17.2	I ² C 相关的寄存器	340
17.3	I ² C 主机模式	341
17.3.1	I2C 配置寄存器 (I2CCFG), 总线速度控制	341
17.3.2	I2C 主机控制寄存器 (I2CMSCR)	342
17.3.3	I2C 主机辅助控制寄存器 (I2CMSAUX)	343
17.3.4	I2C 主机状态寄存器 (I2CMSST)	343
17.4	I ² C 从机模式	345
17.4.1	I2C 从机控制寄存器 (I2CSLCR)	345
17.4.2	I2C 从机状态寄存器 (I2CSLST)	345
17.4.3	I2C 从机地址寄存器 (I2CSLADR)	346
17.4.4	I2C 数据寄存器 (I2CTXD, I2CRXD)	347
17.5	范例程序	348
17.5.1	I ² C 主机模式访问 AT24C256 (中断方式)	348
17.5.2	I ² C 主机模式访问 AT24C256 (查询方式)	353
17.5.3	I ² C 主机模式访问 PCF8563	358
17.5.4	I ² C 从机模式 (中断方式)	363
17.5.5	I ² C 从机模式 (查询方式)	368
17.5.6	测试 I ² C 从机模式代码的主机代码	371
18	高级 PWM	377
18.1	简介	380
18.2	主要特性	380
18.3	时基单元	381
18.3.1	读写 16 位计数器	382
18.3.2	16 位 PWMA_ARR 寄存器的写操作	382
18.3.3	预分频器	382
18.3.4	向上计数模式	382
18.3.5	向下计数模式	384
18.3.6	中间对齐模式 (向上/向下计数)	386

18.3.7	重复计数器	388
18.4	时钟/触发控制器	389
18.4.1	预分频时钟 (CK_PSC)	389
18.4.2	内部时钟源 (fMASTER)	390
18.4.3	外部时钟源模式 1	390
18.4.4	外部时钟源模式 2	391
18.4.5	触发同步	392
18.4.6	与 PWMB 同步	395
18.5	捕获/比较通道	398
18.5.1	16 位 PWMA_CCRi 寄存器的写流程	400
18.5.2	输入模块	400
18.5.3	输入捕获模式	400
18.5.4	输出模块	403
18.5.5	强制输出模式	404
18.5.6	输出比较模式	404
18.5.7	PWM 模式	405
18.5.8	使用刹车功能 (PWMFLT)	411
18.5.9	在外部事件发生时清除 OCiREF 信号	413
18.5.10	编码器接口模式	414
18.6	中断	416
18.7	PWMA/PWMB 寄存器描述	417
18.7.1	功能脚切换 (PWMx_PS)	417
18.7.2	高级 PWM 功能脚选择寄存器 (PWMx_ETRPS)	418
18.7.3	输出使能寄存器 (PWMx_ENO)	418
18.7.4	输出附加使能寄存器 (PWMx_IOAUX)	419
18.7.5	控制寄存器 1 (PWMx_CR1)	420
18.7.6	控制寄存器 2 (PWMx_CR2), 及实时触发 ADC	421
18.7.7	从模式控制寄存器 (PWMx_SMCR)	423
18.7.8	外部触发寄存器 (PWMx_ETR)	425
18.7.9	中断使能寄存器 (PWMx_IER)	425
18.7.10	状态寄存器 1 (PWMx_SR1)	426
18.7.11	状态寄存器 2 (PWMx_SR2)	427
18.7.12	事件产生寄存器 (PWMx_EGR)	427
18.7.13	捕获/比较模式寄存器 1 (PWMx_CCMR1)	428
18.7.14	捕获/比较模式寄存器 2 (PWMx_CCMR2)	431
18.7.15	捕获/比较模式寄存器 3 (PWMx_CCMR3)	432
18.7.16	捕获/比较模式寄存器 4 (PWMx_CCMR4)	433
18.7.17	捕获/比较使能寄存器 1 (PWMx_CCER1)	434
18.7.18	捕获/比较使能寄存器 2 (PWMx_CCER2)	435
18.7.19	计数器高 8 位 (PWMx_CNTRH)	436
18.7.20	计数器低 8 位 (PWMx_CNTRL)	436
18.7.21	预分频器高 8 位 (PWMx_PSCRH), 输出频率计算公式	436
18.7.22	预分频器低 8 位 (PWMx_PSCRL)	436
18.7.23	自动重装载寄存器高 8 位 (PWMx_ARRH)	437

18.7.24	自动重装载寄存器低 8 位 (PWMx_ARRL)	437
18.7.25	重复计数器寄存器 (PWMx_RCR)	437
18.7.26	捕获/比较寄存器 1/5 高 8 位 (PWMx_CCR1H)	437
18.7.27	捕获/比较寄存器 1/5 低 8 位 (PWMx_CCR1L)	438
18.7.28	捕获/比较寄存器 2/6 高 8 位 (PWMx_CCR2H)	438
18.7.29	捕获/比较寄存器 2/6 低 8 位 (PWMx_CCR2L)	438
18.7.30	捕获/比较寄存器 3/7 高 8 位 (PWMx_CCR3H)	438
18.7.31	捕获/比较寄存器 3/7 低 8 位 (PWMx_CCR3L)	438
18.7.32	捕获/比较寄存器 4/8 高 8 位 (PWMx_CCR4H)	438
18.7.33	捕获/比较寄存器 4/8 低 8 位 (PWMx_CCR4L)	439
18.7.34	刹车寄存器 (PWMx_BKR)	439
18.7.35	死区寄存器 (PWMx_DTR)	440
18.7.36	输出空闲状态寄存器 (PWMx_OISR)	440
18.8	范例程序	442
18.8.1	六步 PWM 驱动无刷直流马达(带 HALL)	442
18.8.2	BLDC 无刷直流电机驱动(无 HALL)	451
18.8.3	正交编码器模式	458
18.8.4	单脉冲模式 (触发控制脉冲输出)	459
18.8.5	门控模式 (输入电平使能计数器)	460
18.8.6	外部时钟模式	462
18.8.7	输入捕获模式测量脉冲周期 (捕获上升沿到上升沿或者下降沿到下降沿)	463
18.8.8	输入捕获模式测量脉冲高电平宽度 (捕获上升沿到下降沿)	464
18.8.9	输入捕获模式测量脉冲低电平宽度 (捕获下降沿到上升沿)	465
18.8.10	输入捕获模式同时测量脉冲周期和占空比	466
18.8.11	带死区控制的 PWM 互补输出	467
18.8.12	PWM 端口做外部中断 (下降沿中断或者上升沿中断)	468
18.8.13	输出任意周期和任意占空比的波形	468
18.8.14	使用 PWM 的 CEN 启动 PWMA 定时器, 实时触发 ADC	469
18.8.15	利用 PWM 实现 16 位 DAC 的参考线路图	470
18.8.16	利用 PWM 实现互补 SPWM	471
19	USB 通用串行总线	475
19.1	USB 相关的寄存器	475
19.1.1	USB 控制寄存器 (USBCON)	475
19.1.2	USB 时钟控制寄存器 (USBCLK)	476
19.1.3	USB 间址地址寄存器 (USBADDR)	477
19.1.4	USB 间址数据寄存器 (USBDATA)	477
19.2	USB 控制器寄存器 (SIE)	477
19.2.1	USB 功能地址寄存器 (FADDR)	478
19.2.2	USB 电源控制寄存器 (POWER)	478
19.2.3	USB 端点 IN 中断标志位 (INTRIN1)	479
19.2.4	USB 端点 OUT 中断标志位 (INTROUT1)	479
19.2.5	USB 电源中断标志 (INTRUSB)	480
19.2.6	USB 端点 IN 中断允许寄存器 (INTRIN1E)	480
19.2.7	USB 端点 OUT 中断允许寄存器 (INTROUT1E)	481

19.2.8	USB 电源中断允许寄存器 (INTRUSB)	481
19.2.9	USB 数据帧号寄存器 (FRAME _n)	481
19.2.10	USB 端点索引寄存器 (INDEX)	481
19.2.11	IN 端点的最大数据包大小 (INMAXP)	482
19.2.12	USB 端点 0 控制状态寄存器 (CSR0)	482
19.2.13	IN 端点控制状态寄存器 1 (INCSR1)	483
19.2.14	IN 端点控制状态寄存器 2 (INCSR2)	483
19.2.15	OUT 端点的最大数据包大小 (OUTMAXP)	484
19.2.16	OUT 端点控制状态寄存器 1 (OUTCSR1)	484
19.2.17	OUT 端点控制状态寄存器 2 (OUTCSR2)	485
19.2.18	USB 端点 0 的 OUT 长度 (COUNT0)	485
19.2.19	USB 端点的 OUT 长度 (OUTCOUNT _n)	485
19.2.20	USB 端点的 FIFO 数据访问寄存器 (FIFO _n)	485
19.2.21	USB 跟踪控制寄存器 (UTRKCTL)	486
19.2.22	USB 跟踪状态寄存器 (UTRKSTS)	486
19.3	范例程序	487
19.3.1	HID 人机接口设备范例	487
20	CAN 总线	498
20.1	CAN 功能脚切换	498
20.2	CAN 相关的寄存器	498
20.2.1	辅助寄存器 2 (AUXR2)	498
20.2.2	CAN 总线中断控制寄存器 (CANICR)	498
20.2.3	CAN 总线地址寄存器 (CANAR)	499
20.2.4	CAN 总线数据寄存器 (CANDR)	499
20.3	CAN 内部功能寄存器	499
20.3.1	CAN 模式寄存器 (MR)	500
20.3.2	CAN 命令寄存器 (CMR)	500
20.3.3	CAN 状态寄存器 (SR)	500
20.3.4	CAN 中断/应答寄存器 (ISR/IACK)	501
20.3.5	CAN 中断寄存器 (IMR)	502
20.3.6	CAN 数据帧接收计数器 (RMC)	502
20.3.7	CAN 总线时钟寄存器 0 (BTR0)	502
20.3.8	CAN 总线时钟寄存器 1 (BTR1)	503
20.3.9	CAN 总线数据帧发送缓存 (TXBUF _n)	503
20.3.10	CAN 总线数据帧接收缓存 (RXBUF _n)	503
20.3.11	CAN 总线验收代码寄存器 (ACR _n)	503
20.3.12	CAN 总线验收屏蔽寄存器 (AMR _n)	504
20.3.13	CAN 总线错误信息寄存器 (ECC)	506
20.3.14	CAN 总线接收错误计数器 (RXERR)	506
20.3.15	CAN 总线发送错误计数器 (TXERR)	506
20.3.16	CAN 总线仲裁丢失寄存器 (ALC)	507
20.4	范例程序	508
20.4.1	CAN 总线帧格式	508
20.4.2	CAN 总线标准帧收发范例	509

20.4.3	CAN 总线扩展帧收发范例.....	512
21	LIN 总线.....	516
21.1	LIN 功能脚切换	516
21.2	LIN 相关的寄存器	516
21.2.1	辅助寄存器 2 (AUXR2)	516
21.2.2	LIN 总线中断控制寄存器 (LINICR)	516
21.2.3	LIN 总线地址寄存器 (LINAR)	517
21.2.4	LIN 总线数据寄存器 (LINDR)	517
21.3	LIN 内部功能寄存器	517
21.3.1	LIN 数据寄存器 (LBUF)	517
21.3.2	LIN 数据地址寄存器 (LSEL)	518
21.3.3	LIN 帧 ID 寄存器 (LID)	518
21.3.4	LIN 错误寄存器 (LER)	518
21.3.5	LIN 中断使能寄存器 (LIE)	518
21.3.6	LIN 状态寄存器 (只读寄存器) (LSR)	518
21.3.7	LIN 控制寄存器 (只写寄存器) (LCR)	519
21.3.8	LIN 波特率寄存器 (DLL/DLH)	519
21.3.9	LIN 帧头延时计数寄存器 (HDRL/HDRH)	519
21.3.10	LIN 帧头延时分频寄存器 (HDP)	519
21.4	范例程序.....	521
21.4.1	LIN 总线主机收发范例	521
21.4.2	LIN 总线从机收发范例	525
21.4.3	使用串口模拟 LIN 总线范例	529
21.4.4	LIN 总线时序介绍	535
22	32 位硬件乘除单元, MDU32	538
22.1	相关的特殊功能寄存器	538
22.2	指令执行时间.....	538
22.3	MDU32-算术运算.....	538
22.3.1	32 位乘法.....	538
22.3.2	32 位无符号除法.....	539
22.3.3	32 位有符号除法.....	539
22.4	范例程序.....	540
23	单精度浮点运算器, FPMU	543
23.1	FPMU – FLOATING POINT MATH UNIT SUMMARY	543
23.2	Execution time table.....	543
23.3	FPMU – BASIC ARITHMETIC OPERATIONS.....	544
23.3.1	32-BIT REAL DATA ADDITION.....	544
23.3.2	32-BIT REAL DATA SUBTRACTION.....	544
23.3.3	32-BIT REAL DATA MULTIPLICATION	545
23.3.4	32-BIT REAL DATA DIVISION	545
23.3.5	32-BIT REAL DATA SQUARE ROOT	545
23.3.6	32-BIT REAL DATA COMPARISON	545
23.3.7	32-BIT REAL DATA EXAMINE INPUT DATA.....	546
23.4	FPMU TRIGONOMETRIC OPERATIONS.....	546

23.4.1	32-BIT REAL DATA SINE	547
23.4.2	32-BIT REAL DATA COSINE	547
23.4.3	32-BIT REAL DATA TANGENT	547
23.4.4	32-BIT REAL DATA ARCTANGENT	547
23.5	FPMU DATA CONVERSION OPERATIONS	548
23.5.1	CONVERSION 32-BIT REAL DATA TO 8-BIT INTEGER	548
23.5.2	CONVERSION 32-BIT REAL DATA TO 16-BIT INTEGER	548
23.5.3	CONVERSION 32-BIT REAL DATA TO 32-BIT INTEGER	548
23.5.4	CONVERSION 8-BIT INTEGER TO 32-BIT REAL DATA	549
23.5.5	CONVERSION 16-BIT INTEGER TO 32-BIT REAL DATA	549
23.5.6	CONVERSION 32-BIT INTEGER TO 32-BIT REAL DATA	549
23.6	FPMU COPROCESSOR CONTROL OPERATIONS.....	549
23.6.1	INITIALIZE COPROCESSOR.....	550
23.6.2	CLEAR EXCEPTIONS	550
23.6.3	READ STATUS REGISTER.....	550
23.6.4	WRITE STATUS REGISTER	550
23.6.5	READ CONTROL REGISTER	550
23.6.6	WRITE CONTROL REGISTER.....	550
23.7	范例程序	551
附录 A	指令集.....	554
A.1	INSTRUCTIONS SET BRIEF	554
A.1.1	BINARY MODE AND SOURCE MODE.....	554
A.1.2	INSTRUCTION SET NOTES	554
A.1.3	INSTRUCTION SET BRIEF – FUNCTIONAL ORDER.....	555
A.1.4	INSTRUCTION SET BRIEF – HEXADECIMAL ORDER.....	562
A.2	INSTRUCTIONS SET DETAILS	570
附录 B	电气特性.....	654
附录 C	注意事项.....	656
附录 D	STC16 使用 Keil 开发注意事项	657
附录 E	STC16 头文件	659
附录 F	更新记录.....	679

1 概述

STC16F 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 16 位 8051 单片机，在相同的工作频率下，STC16F 系列单片机比传统的 8051 约快 70 倍。

STC16F 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 16 位 8051 单片机，超级加密。

MCU 内部集成高精度 R/C 时钟(±0.3%，常温下+25℃)，-1.38%~+1.42%温飘(-40℃~+85℃)，-0.88%~+1.05%温飘(-20℃~+65℃)。ISP 编程时 4MHz~35MHz 宽范围可设置（注意：温度范围为-40℃~+85℃时，最高频率须控制在 35MHz 以下），可彻底省掉外部昂贵的晶振和外部复位电路(内部已集成高可靠复位电路，ISP 编程时 4 级复位门槛电压可选)。

MCU 内部有 4 个可选时钟源：内部 20M~40MHz 高精度 IRC 时钟(可适当调高或调低)、内部 32KHz 的低速 IRC、外部 4M~33M 晶振或外部时钟信号以及内部 PLL 输出时钟。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 1.3mA（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 0.1uA 以下。

MCU 提供了丰富的数字外设（4 个串口、5 个定时器、2 组针对三相电机控制能够输出互补/对称/带死区控制信号的 16 位高级 PWM 定时器以及 I²C、SPI、USB、CAN、LIN）接口与模拟外设（超高速 12 位 ADC、比较器），可满足广大用户的设计需求。

STC16F 系列单片机有 268 条强大的指令，包含 32 位加减法指令。数学系处理单元扩充了 32 位硬件乘除单元 MDU32（包含 32 位除以 32 位和 32 位乘以 32 位），使 STC16F 计算能力达到了 32 位单片机的水平。部分单片机还扩充了单精度浮点运算器，比没有单精度浮点运算器的通用 32 位 MCU 还要强。

STC16F 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	高级 PWM	比较器	SPI	I2C	USB	CAN	LIN	FPMU	MDU32
STC16F40K128 系列	60	4	5	15 _{CH} *12 _B	●	●	●	●	●	●	●	●	●

2 特性及价格

2.1 STC16F40K128 系列特性及价格

➤ 选型价格（不需要外部晶振、不需要外部复位，12 位 ADC，15 通道）

单片机型号	工作电压 (V)	Flash 程序存储器 10 万次 字节	edata 内部扩展 DATA RAM 可做堆栈或变量 字节	xdata 内部大容量扩展 SRAM 可做变量 字节	强大的双 DPTR 可增可减	EEPROM 10 万次 字节 (暂不支持)	I/O 口最多数量	串口并可掉电唤醒	全速 USB	CAN 总线	LIN 总线	SPI	I ² C	MDU32 硬件 32 位乘除法器	FPU 单精度浮点运算器	定时器/计数器 (10-14 外部管脚也可掉电唤醒)	16 位高级 PWM 定时器 互补对称死区控制	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	内部高精度时钟 (36MHz 以下可调)	内部高可靠复位 (可选复位门电压)	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	支持硬件 USB 直接下载和硬件 USB 仿真	本身就可在线仿真 (后续支持)	封装			2020 年新品供货信息
STC16F40K128	1.9-5.5V	120K	8K	32K	2	IAP	60	4	有	有	有	有	有	有	有	5	2	有	12 位	有	有	有	4 级	有	是	有	是	是	是	-	有	-	有	送样

➤ 内核

- ✓ 超高速 16 位 8051 内核 (1T)，比传统 8051 约快 70 倍以上
- ✓ 24 个中断源，4 级中断优先级
- ~~✗ 支持在线仿真 (后续支持)~~

➤ 工作电压

- ✓ 1.9V~5.5V
- ✓ 内建 LDO

➤ 工作温度

- ✓ -40℃~85℃

➤ Flash 存储器

- ✓ 最大 120K 字节 FLASH 程序存储器 (ROM)，用于存储用户代码，前 60K (FE0000H-FEEFFFH)，后 60K (FF0000H-FFFFFH)
- ~~✗ 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 (暂不支持)~~
- ✓ 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 (后续支持)~~

➤ SRAM

- ✓ 8K 字节内部 SRAM (EDATA)
- ✓ 32K 字节内部扩展 RAM (内部 XDATA)
- ✓ xdata 使用注意：

定义变量时可将单字节变量定义在 xdata 里面，多字节 (2 字节、4 字节) 变量需要定义在 edata 里面。
后续版本会取消这种使用限制。

➤ 时钟控制

- ✓ 内部高精度 IRC (ISP 编程时可进行上下调整)

- ✦ 误差±0.3% (常温下 25℃)
- ✦ -1.35%~+1.30%温漂 (全温度范围, -40℃~85℃)
- ✦ -0.76%~+0.98%温漂 (温度范围, -20℃~65℃)

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

用户可自由选择上面的 4 种时钟源

➤ 复位

- ✓ 硬件复位

- ✦ 上电复位, 复位电压值为 1.7V~1.9V。 (在芯片未使能低压复位功能时有效)
- ✦ 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、2.4V、2.7V、3.0V。

- ✓ 软件复位

- ✦ 软件方式写复位触发寄存器

➤ 中断

- ✓ 提供 24 个中断源: INT0、INT1、INT2、INT3、INT4、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、PWMA、PWMB、USB、CAN、LIN
- ✓ 提供 4 级中断优先级

➤ 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
- ✓ 2 组高级 PWM, 可实现 8 通道 (4 组互补对称) 带死区的控制的 PWM, 并支持外部异常检测功能
- ✓ SPI: 支持主机模式和从机模式以及主机/从机自动切换
- ✓ I²C: 支持主机模式和从机模式
- ✓ ICE: 硬件支持仿真
- ✓ USB: USB2.0/USB1.1 兼容全速 USB, 6 个双向端点, 支持 4 种端点传输模式 (控制传输、中断传输、批量传输和同步传输), 每个端点拥有 64 字节的缓冲区
- ✓ CAN: 一个 CAN 2.0 控制单元
- ✓ LIN: 一个 LIN 1.3、2.1 控制单元
- ✓ MDU32: 硬件 32 位乘法器 (包含 32 位除以 32 位、32 位乘以 32 位)
- ✓ FPMU: 单精度浮点运算器

➤ 模拟外设

- ✓ ADC: 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, ADC 的通道 15 用于测试内部参考电压 (芯片在出厂时, 内部参考电压调整为 1.19V, 误差±1%)
- ✓ 比较器: 一组比较器

➤ GPIO

- ✓ 最多可达 60 个 GPIO: P0.0~P0.7、P1.0~P1.7 (无 P1.2)、P2.0~P2.7、P3.0~P3.7、P4.0~P4.7、P5.0~P5.4、P6.0~P6.7、P7.0~P7.7
- ✓ 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 IO 口上电后的状态均为高阻输入状态, 用户在使用 IO 口时必须先设置 IO 口模式
- ✓ 另外每个 I/O 均可独立使能内部 4K 上拉电阻

➤ 封装

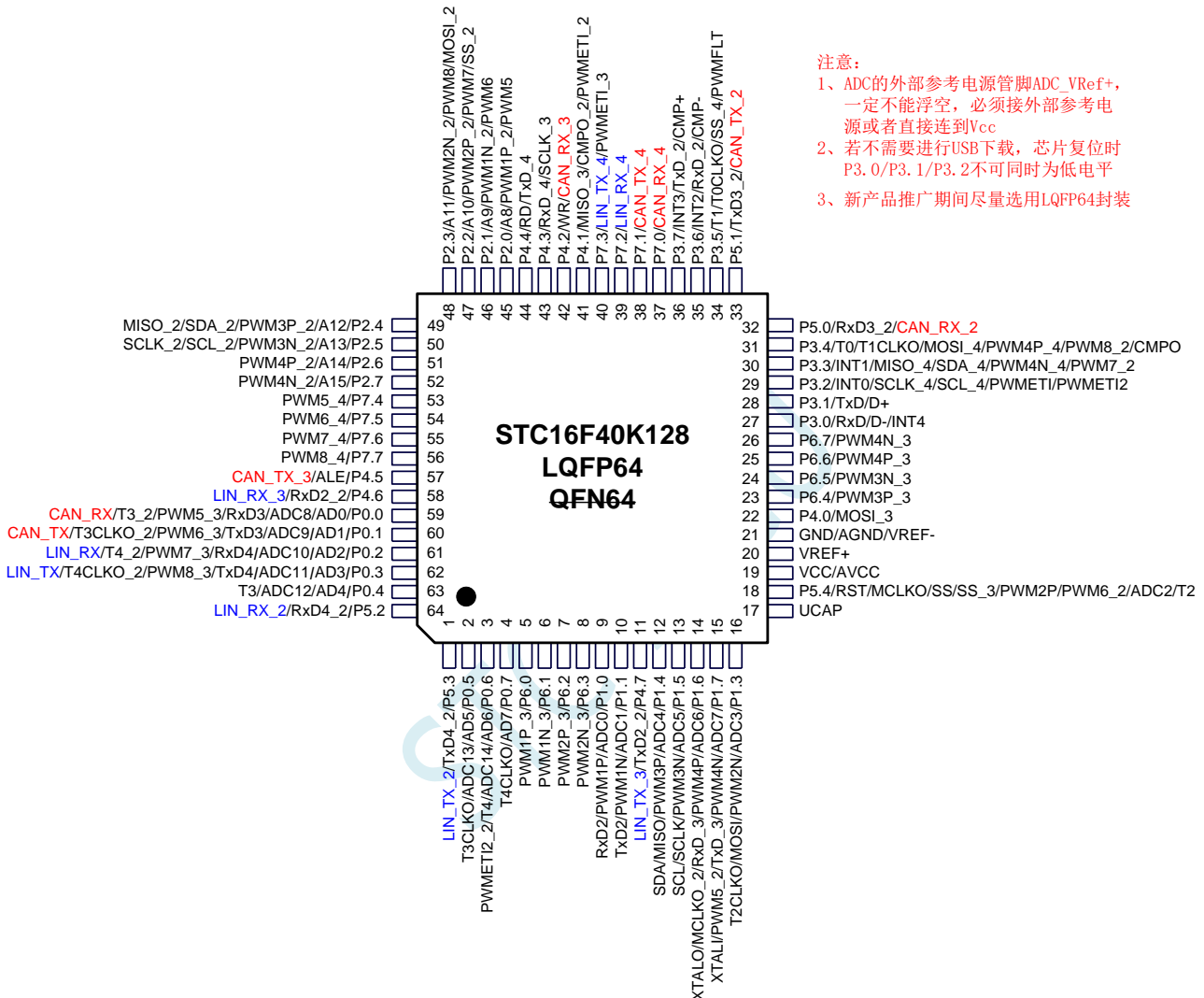
- ✓ LQFP64、QFN64、LQFP48

STC MCU

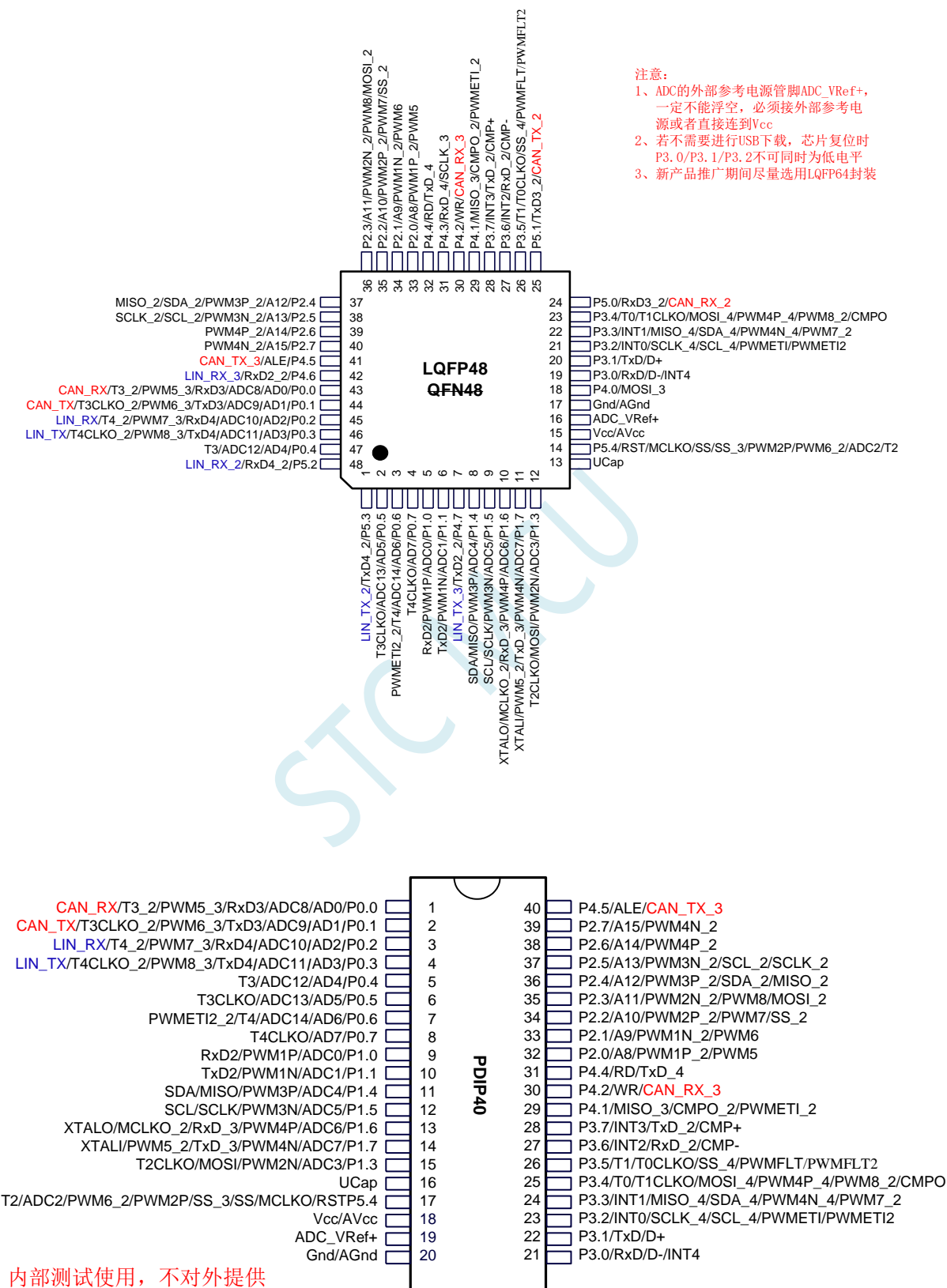
3 管脚及说明

3.1 管脚图

3.1.1 STC16F40K128 系列管脚图



- 注意:**
- 1、除 P3.0 和 P3.1 外, 其余所有 I/O 口上电后的状态均为高阻输入状态, 用户在使用 I/O 口时必须先设置 I/O 口模式
 - 2、所有的 I/O 口均可以设置为准双向口模式、强推挽输出模式、开漏输出模式或者高阻输入模式, 另外每个 I/O 均可独立使能内部 4K 上拉电阻
 - 3、当使能 P5.4 口为复位脚时, 复位电平为低电平



3.2 管脚说明

3.2.1 STC16F40K128 系列管脚说明

编号		名称	类型	说明
LQFP64	LQFP48			
1	1	P5.3	I/O	标准 IO 口
		TxD4_2	O	串口 4 的发送脚
		LIN_TX_2	O	LIN 总线发送脚
2	2	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
3	3	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
		PWMFLT2_2	I	增强 PWM 的外部异常检测脚
4	4	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
5		P6.0	I/O	标准 IO 口
		PWM1P_3	I/O	PWM1 的捕获输入和脉冲输出正极
6		P6.1	I/O	标准 IO 口
		PWM1N_3	I/O	PWM1 的捕获输入和脉冲输出负极
7		P6.2	I/O	标准 IO 口
		PWM2P_3	I/O	PWM2 的捕获输入和脉冲输出正极
8		P6.3	I/O	标准 IO 口
		PWM2N_3	I/O	PWM2 的捕获输入和脉冲输出负极
9	5	P1.0	I/O	标准 IO 口
		ADC0	I	ADC 模拟输入通道 0
		PWM1P	I/O	PWM1 的捕获输入和脉冲输出正极
		RxD2	I	串口 2 的接收脚
10	6	P1.1	I/O	标准 IO 口
		ADC1	I	ADC 模拟输入通道 1
		PWM1N	I/O	PWM1 的捕获输入和脉冲输出负极
		TxD2	I	串口 2 的发送脚

编号		名称	类型	说明
LQFP64	LQFP48			
11	7	P4.7	I/O	标准 IO 口
		TxD2_2	I	串口 2 的发送脚
		LIN_TX_3	O	LIN 总线发送脚
12	8	P1.4	I/O	标准 IO 口
		ADC4	I	ADC 模拟输入通道 4
		PWM3P	I/O	PWM3 的捕获输入和脉冲输出正极
		MISO	I/O	SPI 主机输入从机输出
		SDA	I/O	I2C 接口的数据线
13	9	P1.5	I/O	标准 IO 口
		ADC5	I	ADC 模拟输入通道 5
		PWM3N	I/O	PWM3 的捕获输入和脉冲输出负极
		SCLK	I/O	SPI 的时钟脚
		SCL	I/O	I2C 的时钟线
14	10	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
		PWM4P	I/O	PWM4 的捕获输入和脉冲输出正极
		MCLKO_2	O	主时钟分频输出
		XTALO	O	外部晶振的输出脚
15	11	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
		PWM4N	I/O	PWM4 的捕获输入和脉冲输出负极
		PWM5_2	I/O	PWM5 的捕获输入和脉冲输出
		XTALI	I	外部晶振/外部时钟的输入脚
16	12	P1.3	I/O	标准 IO 口
		ADC3	I	ADC 模拟输入通道 3
		MOSI	I/O	SPI 主机输出从机输入
		PWM2N	I/O	PWM2 的捕获输入和脉冲输出负极
		T2CLKO	O	定时器 2 时钟分频输出
17	13	UCAP	I	USB 内核电源稳压脚
18	14	P5.4	I/O	标准 IO 口
		RST	I	复位引脚
		MCLKO	O	主时钟分频输出
		SS_3	I	SPI 的从机选择脚（主机为输出）
		SS	I	SPI 的从机选择脚（主机为输出）
		PWM2P	I/O	PWM2 的捕获输入和脉冲输出正极
		PWM6_2	I/O	PWM6 的捕获输入和脉冲输出
		T2	I	定时器 2 外部时钟输入
		ADC2	I	ADC 模拟输入通道 2

编号		名称	类型	说明
LQFP64	LQFP48			
19	15	Vcc	VCC	电源脚
		AVcc	VCC	ADC 电源脚
20	16	Vref+	I	ADC 的参考电压脚
21	17	Gnd	GND	地线
		Agnd	GND	ADC 地线
		Vref-	I	ADC 的参考电压地线
22	18	P4.0	I/O	标准 IO 口
		MOSI_3	I/O	SPI 主机输出从机输入
23		P6.4	I/O	标准 IO 口
		PWM3P_3	I/O	PWM3 的捕获输入和脉冲输出正极
24		P6.5	I/O	标准 IO 口
		PWM3N_3	I/O	PWM3 的捕获输入和脉冲输出负极
25		P6.6	I/O	标准 IO 口
		PWM4P_3	I/O	PWM4 的捕获输入和脉冲输出正极
26		P6.7	I/O	标准 IO 口
		PWM4N_3	I/O	PWM4 的捕获输入和脉冲输出负极
27	19	P3.0	I/O	标准 IO 口
		D-	I/O	USB 数据口
		RxD	I	串口 1 的接收脚
		INT4	I	外部中断 4
28	20	P3.1	I/O	标准 IO 口
		D+	I/O	USB 数据口
		TxD	O	串口 1 的发送脚
29	21	P3.2	I/O	标准 IO 口
		INT0	I	外部中断 0
		SCLK_4	I/O	SPI 的时钟脚
		SCL_4	I/O	I2C 的时钟线
		PWMETI	I	PWM 外部触发输入脚
		PWMETI2	I	PWM 外部触发输入脚 2

编号		名称	类型	说明
LQFP64	LQFP48			
30	22	P3.3	I/O	标准 IO 口
		INT1	I	外部中断 1
		MISO_4	I/O	SPI 主机输入从机输出
		SDA_4	I/O	I2C 接口的数据线
		PWM4N_4	I/O	PWM4 的捕获输入和脉冲输出负极
		PWM7_2	I/O	PWM7 的捕获输入和脉冲输出
31	23	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
		PWM4P_4	I/O	PWM4 的捕获输入和脉冲输出正极
		PWM8_2	I/O	PWM8 的捕获输入和脉冲输出
		CMPO	O	比较器输出
32	24	P5.0	I/O	标准 IO 口
		RxD3_2	I	串口 3 的接收脚
		CAN_RX_2	I	CAN 总线接收脚
33	25	P5.1	I/O	标准 IO 口
		TxD3_2	O	串口 3 的发送脚
		CAN_TX_2	O	CAN 总线发送脚
34	26	P3.5	I/O	标准 IO 口
		T1	I	定时器 1 外部时钟输入
		T0CLKO	O	定时器 0 时钟分频输出
		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
35	27	P3.6	I/O	标准 IO 口
		INT2	I	外部中断 2
		RxD_2	I	串口 1 的接收脚
		CMP-	I	比较器负极输入
36	28	P3.7	I/O	标准 IO 口
		INT3	I	外部中断 3
		TxD_2	O	串口 1 的发送脚
		CMP+	I	比较器正极输入
37		P7.0	I/O	标准 IO 口
		CAN_RX_4	I	CAN 总线接收脚
38		P7.1	I/O	标准 IO 口
		CAN_TX_4	O	CAN 总线发送脚
39		P7.2	I/O	标准 IO 口
		LIN_RX_4	I	LIN 总线接收脚

编号		名称	类型	说明
LQFP64	LQFP48			
40		P7.3	I/O	标准 IO 口
		LIN_TX_4	O	LIN 总线发送脚
		PWMETI_3	I	PWM 外部触发输入脚
41	29	P4.1	I/O	标准 IO 口
		MISO_3	I/O	SPI 主机输入从机输出
		CMPO_2	O	比较器输出
		PWMETI_3	I	PWM 外部触发输入脚
42	30	P4.2	I/O	标准 IO 口
		WR	O	外部总线的写信号线
		CAN_RX_3	I	CAN 总线接收脚
43	31	P4.3	I/O	标准 IO 口
		RxD_4	I	串口 1 的接收脚
		SCLK_3	I/O	SPI 的时钟脚
44	32	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
45	33	P2.0	I/O	标准 IO 口
		A8	I	地址总线
		PWM1P_2	I/O	PWM1 的捕获输入和脉冲输出正极
		PWM5	I/O	PWM5 的捕获输入和脉冲输出
46	34	P2.1	I/O	标准 IO 口
		A9	I	地址总线
		PWM1N_2	I/O	PWM1 的捕获输入和脉冲输出负极
		PWM6	I/O	PWM6 的捕获输入和脉冲输出
47	35	P2.2	I/O	标准 IO 口
		A10	I	地址总线
		SS_2	I	SPI 的从机选择脚（主机为输出）
		PWM2P_2	I/O	PWM2 的捕获输入和脉冲输出正极
		PWM7	I/O	PWM7 的捕获输入和脉冲输出
48	36	P2.3	I/O	标准 IO 口
		A11	I	地址总线
		MOSI_2	I/O	SPI 主机输出从机输入
		PWM2N_2	I/O	PWM2 的捕获输入和脉冲输出负极
		PWM8	I/O	PWM8 的捕获输入和脉冲输出
49	37	P2.4	I/O	标准 IO 口
		A12	I	地址总线
		MISO_2	I/O	SPI 主机输入从机输出
		SDA_2	I/O	I2C 接口的数据线
		PWM3P_2	I/O	PWM3 的捕获输入和脉冲输出正极

编号		名称	类型	说明
LQFP64	LQFP48			
50	38	P2.5	I/O	标准 IO 口
		A13	I	地址总线
		SCLK_2	I/O	SPI 的时钟脚
		SCL_2	I/O	I2C 的时钟线
		PWM3N_2	I/O	PWM3 的捕获输入和脉冲输出负极
51	39	P2.6	I/O	标准 IO 口
		A14	I	地址总线
		PWM4P_2	I/O	PWM4 的捕获输入和脉冲输出正极
52	40	P2.7	I/O	标准 IO 口
		A15	I	地址总线
		PWM4N_2	I/O	PWM4 的捕获输入和脉冲输出负极
53		P7.4	I/O	标准 IO 口
		PWM5_4	I/O	PWM5 的捕获输入和脉冲输出
54		P7.5	I/O	标准 IO 口
		PWM6_4	I/O	PWM6 的捕获输入和脉冲输出
55		P7.6	I/O	标准 IO 口
		PWM7_4	I/O	PWM7 的捕获输入和脉冲输出
56		P7.7	I/O	标准 IO 口
		PWM8_4	I/O	PWM8 的捕获输入和脉冲输出
57	41	P4.5	I/O	标准 IO 口
		ALE	O	地址锁存信号
		CAN_TX_3	O	CAN 总线发送脚
58	42	P4.6	I/O	标准 IO 口
		RxD2_2	I	串口 2 的接收脚
		LIN_RX_3	I	LIN 总线接收脚
59	43	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
		PWM5_3	I/O	PWM5 的捕获输入和脉冲输出
		CAN_RX	I	CAN 总线接收脚

编号		名称	类型	说明
LQFP64	LQFP48			
60	44	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
		PWM6_3	I/O	PWM6 的捕获输入和脉冲输出
		CAN_TX	O	CAN 总线发送脚
61	45	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
		PWM7_3	I/O	PWM7 的捕获输入和脉冲输出
		LIN_RX	I	LIN 总线接收脚
62	46	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
		PWM8_3	I/O	PWM8 的捕获输入和脉冲输出
		LIN_TX	O	LIN 总线发送脚
63	47	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
64	48	P5.2	I/O	标准 IO 口
		RxD4_2	I	串口 4 的接收脚
		LIN_RX_2	I	LIN 总线接收脚

3.3 功能脚切换

STC16F 系列单片机的特殊外设串口、SPI、PWM、I²C、CAN、LIN 以及总线控制脚可以在多个 I/O 直接进行切换，以实现一个外设当作多个设备进行分时复用。

3.3.1 功能脚切换相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P_SW1	外设端口切换寄存器 1	9AH	S1_S[1:0]		CAN_S[1:0]		SPI_S[1:0]		LIN_S[1:0]		nn00,0000
P_SW2	外设端口切换寄存器 2	9BH	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	
MCLKOCR	主时钟输出控制寄存器	7EFE07H	MCLKO_S	MCLKODIV[6:0]							0000,0000
PWMA_ETRPS	PWMA 的 ERT 切换寄存器	7EFEB0H							BRKAPS	ETRAPS[1:0]	xxx,x000
PWMB_ETRPS	PWMB 的 ERT 切换寄存器	7EFEB4H							BRKBPS	ETRBPS[1:0]	xxx,x000
PWMA_PS	PWMA 切换寄存器	7EFEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]		0000,0000
PWMB_PS	PWMB 切换寄存器	7EFEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]		0000,0000
T3T4PIN	T3/T4 选择寄存器	7EFEACH	-	-	-	-	-	-	-	T3T4SEL	xxxx,xxx0

3.3.2 外设端口切换控制寄存器 1 (P_SW1)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	9AH	S1_S[1:0]		CAN_S[1:0]		SPI_S[1:0]		LIN_S[1:0]	

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

CAN_S[1:0]: CAN 功能脚选择位

CAN_S[1:0]	CAN_RX	CAN_TX
00	P0.0	P0.1
01	P5.0	P5.1
10	P4.2	P4.5
11	P7.0	P7.1

LIN_S[1:0]: LIN 功能脚选择位

LIN_S[1:0]	LIN_RX	LIN_TX
00	P0.2	P0.3
01	P5.2	P5.3
10	P4.6	P4.7
11	P7.2	P7.3

SPI_S[1:0]: SPI 功能脚选择位

SPI_S[1:0]	SS	MOSI	MISO	SCLK
------------	----	------	------	------

00	P5.4	P1.3	P1.4	P1.5
01	P2.2	P2.3	P2.4	P2.5
10	P5.4	P4.0	P4.1	P4.3
11	P3.5	P3.4	P3.3	P3.2

3.3.3 外设端口切换控制寄存器 2 (P_SW2)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW2	9BH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S

I2C_S[1:0]: I²C 功能脚选择位

I2C_S[1:0]	SCL	SDA
00	P1.5	P1.4
01	P2.5	P2.4
10	-	-
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	P4.6	P4.7

3.3.4 时钟选择寄存器 (MCLKOCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
MCLKOCR	7EFE07H	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

3.3.5 T3/T4 选择寄存器 (T3T4PIN)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T3T4PIN	7EFEACH	-	-	-	-	-	-	-	T3T4SEL

T3T4SEL: T3/T3CLKO/T4/T4CLKO 脚选择位

T3T4SEL	T3	T3CLKO	T4	T4CLKO
---------	----	--------	----	--------

0	P0.4	P0.5	P0.6	P0.7
1	P0.0	P0.1	P0.2	P0.3

3.3.6 高级 PWM 选择寄存器 (PWMn_PS)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PS	7EFEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]	
PWMB_PS	7EFEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]	

C1PS[1:0]: 高级 PWM 通道 1 输出脚选择位

C1PS[1:0]	PWM1P	PWM1N
00	P1.0	P1.1
01	P2.0	P2.1
10	P6.0	P6.1
11	-	-

C2PS[1:0]: 高级 PWM 通道 2 输出脚选择位

C2PS[1:0]	PWM2P	PWM2N
00	P5.4	P1.3
01	P2.2	P2.3
10	P6.2	P6.3
11	-	-

C3PS[1:0]: 高级 PWM 通道 3 输出脚选择位

C3PS[1:0]	PWM3P	PWM3N
00	P1.4	P1.5
01	P2.4	P2.5
10	P6.4	P6.5
11	-	-

C4PS[1:0]: 高级 PWM 通道 4 输出脚选择位

C4PS[1:0]	PWM4P	PWM4N
00	P1.6	P1.7
01	P2.6	P2.7
10	P6.6	P6.7
11	P3.4	P3.3

C5PS[1:0]: 高级 PWM 通道 5 输出脚选择位

C5PS[1:0]	PWM5
00	P2.0
01	P1.7
10	P0.0
11	P7.4

C6PS[1:0]: 高级 PWM 通道 6 输出脚选择位

C6PS[1:0]	PWM6
00	P2.1
01	P5.4
10	P0.1
11	P7.5

C7PS[1:0]: 高级 PWM 通道 7 输出脚选择位

C7PS[1:0]	PWM7
00	P2.2
01	P3.3
10	P0.2
11	P7.6

C8PS[1:0]: 高级 PWM 通道 8 输出脚选择位

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

3.3.7 高级 PWM 功能脚选择寄存器 (PWM_x_ETRPS)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ETRPS	7EFEB0H						BRKAPS	ETRAPS[1:0]	
PWMB_ETRPS	7EFEB4H						BRKBPS	ETRBPS[1:0]	

ETRAPS[1:0]: 高级 PWMA 的外部触发脚 ERI 选择位

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

ETRBPS[1:0]: 高级 PWMB 的外部触发脚 ERIB 选择位

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

BRKAPS: 高级 PWMA 的刹车脚 PWMFLT 选择位

BRKAPS	PWMFLT
0	P3.5
1	比较器的输出

BRKBPS: 高级 PWMB 的刹车脚 PWMFLT2 选择位

BRKBPS	PWMFLT2
0	P3.5
1	比较器的输出

STC MCU

3.4 范例程序

3.4.1 串口 1 切换

C 语言代码

//测试工作频率为 11.0592MHz

#include "stc16.h"

//头文件见附录

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

\$include (STC16.INC)

;头文件见附录

ORG 0000H

LJMP MAIN

ORG 0100H

MAIN:

MOV SP, #5FH

MOV P0M0, #00H

MOV P0M1, #00H

MOV P1M0, #00H

MOV P1M1, #00H

MOV P2M0, #00H

MOV P2M1, #00H

MOV P3M0, #00H

MOV P3M1, #00H

MOV P4M0, #00H

MOV P4M1, #00H

MOV P5M0, #00H

MOV P5M1, #00H

```

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

SJMP     $

END

```

3.4.2 串口 2 切换

C 语言代码

//测试工作频率为 11.0592MHz

```

#include "stc16.h"          //头文件见附录

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

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

    while (1);
}

```

汇编代码

;测试工作频率为 11.0592MHz

```

$include (STC16.INC)      ;头文件见附录

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
    
```

3.4.3 串口 3 切换

C 语言代码

```

//测试工作频率为 11.0592MHz

#include "stc16.h"      //头文件见附录

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

$include (STC16.INC)      ;头文件见附录

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

```

3.4.4 串口 4 切换

C 语言代码

//测试工作频率为 11.0592MHz

```

#include "stc16.h"      //头文件见附录

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

```

$include (STC16.INC)      ;头文件见附录

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
```

3.4.5 SPI 切换

C 语言代码

//测试工作频率为 11.0592MHz

```
#include "stc16.h"           //头文件见附录

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

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

    while (1);
}
```

汇编代码

;测试工作频率为 11.0592MHz

```
$include (STC16.INC)         ;头文件见附录

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/P5.4, MOSI_3/P4.0, MISO_3/P4.1, SCLK_3/P4.3
;        MOV      P_SW1, #0CH         ;SS_4/P3.5, MOSI_4/P3.4, MISO_4/P3.3, SCLK_4/P3.2

        SJMP     $

END

```

3.4.6 I2C 切换

C 语言代码

//测试工作频率为11.0592MHz

#include "stc16.h"

//头文件见附录

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
```

```
$include (STC16.INC)           ;头文件见附录
```

```
                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_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
```

3.4.7 比较器输出切换

C 语言代码

```
//测试工作频率为 11.0592MHz
```

```
#include "stc16.h"           //头文件见附录
```

```
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

\$include (STC16.INC) ;头文件见附录

```
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
```

3.4.8 主时钟输出切换

C 语言代码

//测试工作频率为 11.0592MHz

#include "stc16.h" ;头文件见附录

```
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; //IRC/4 output via MCLKO/P5.4
// MCLKOCR = 0x84; //IRC/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

\$include (STC16.INC)

;头文件见附录

```
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

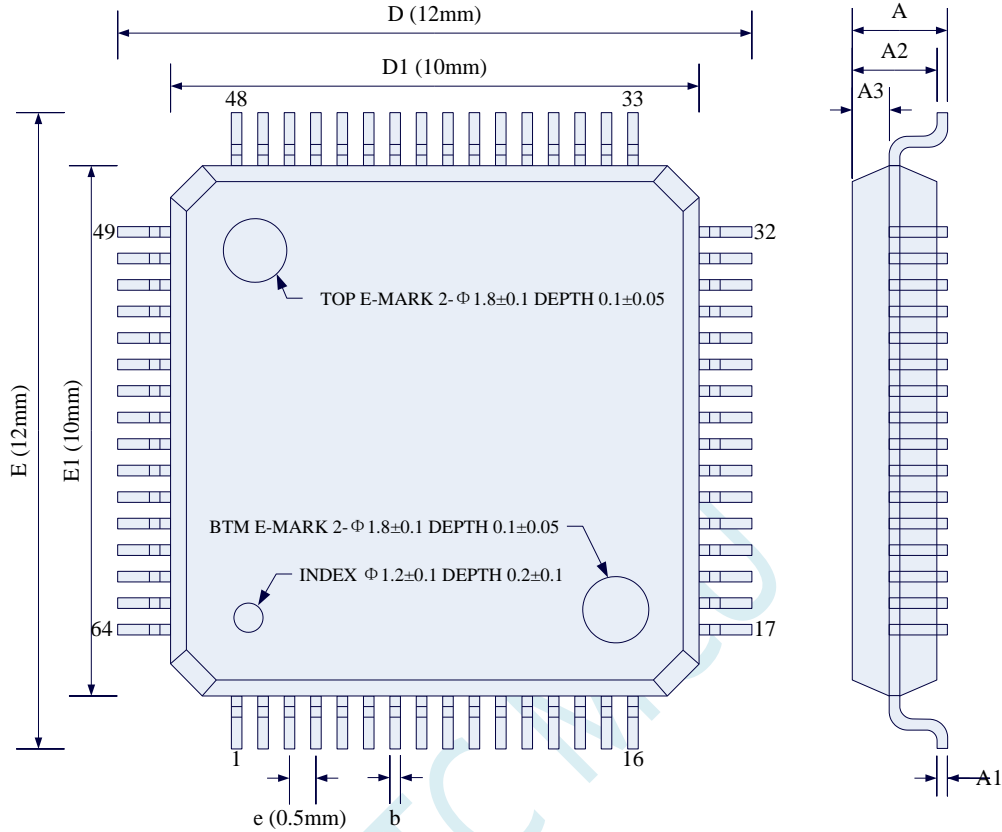
MOV P_SW2, #80H
MOV A, #04H ;IRC/4 output via MCLKO/P5.4
; MOV A, #84H ;IRC/4 output via MCLKO_2/P1.6
MOV WR6, #WORD0 MCLKOCR
MOV WR4, #WORD2 MCLKOCR
MOV @DR4, R11
MOV P_SW2, #00H

SJMP $

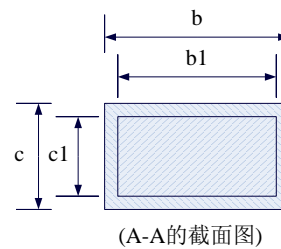
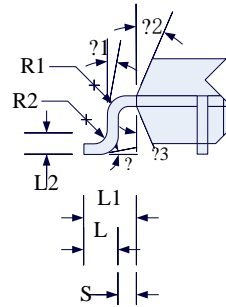
END
```

4 封装尺寸图

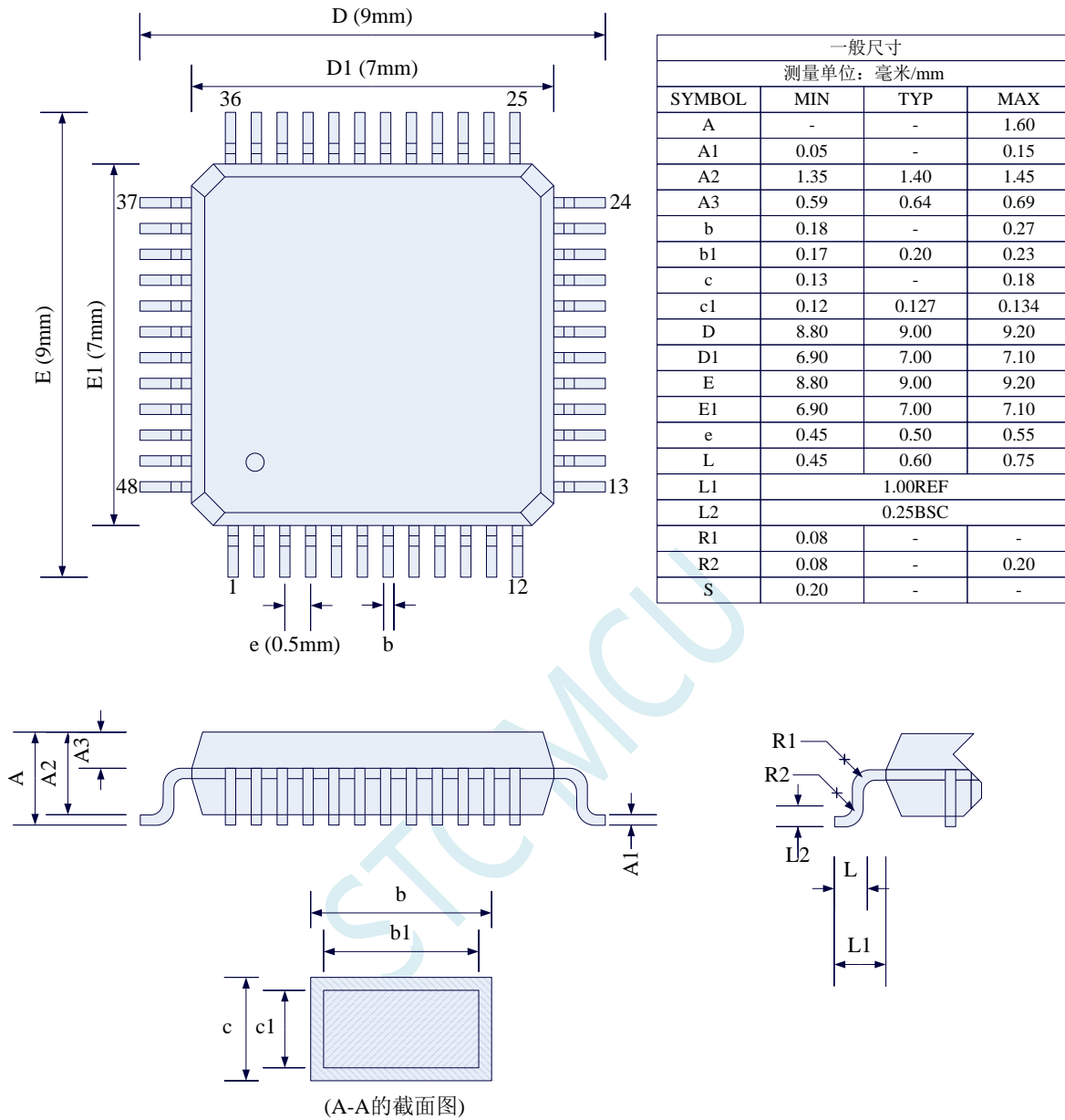
4.1 LQFP64S 封装尺寸图 (12mm*12mm)



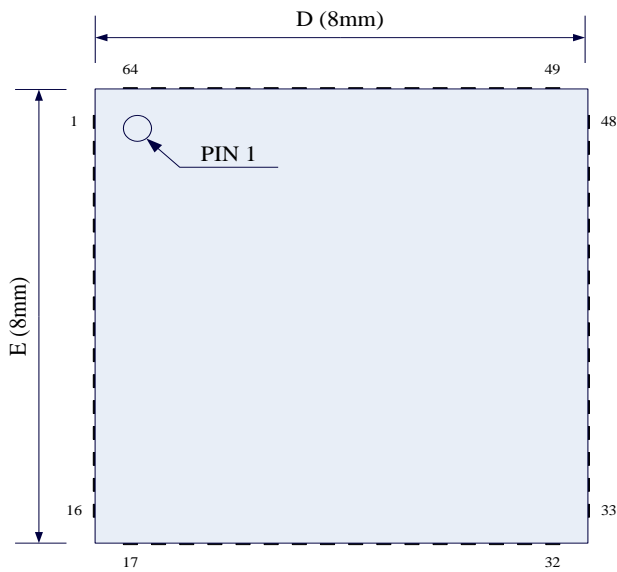
一般尺寸			
测量单位: 毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.18	-	0.27
b1	0.17	0.20	0.23
c	0.13	-	0.18
c1	0.12	0.127	0.134
D	11.80	12.00	12.20
D1	9.90	10.00	10.10
E	11.80	12.00	12.20
E1	9.90	10.00	10.10
e	0.50BSC		
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-
?	0°	3.5°	7°
?1	0°	-	-
?2	11°	12°	13°
?3	11°	12°	13°



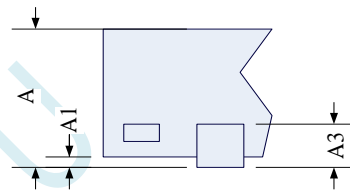
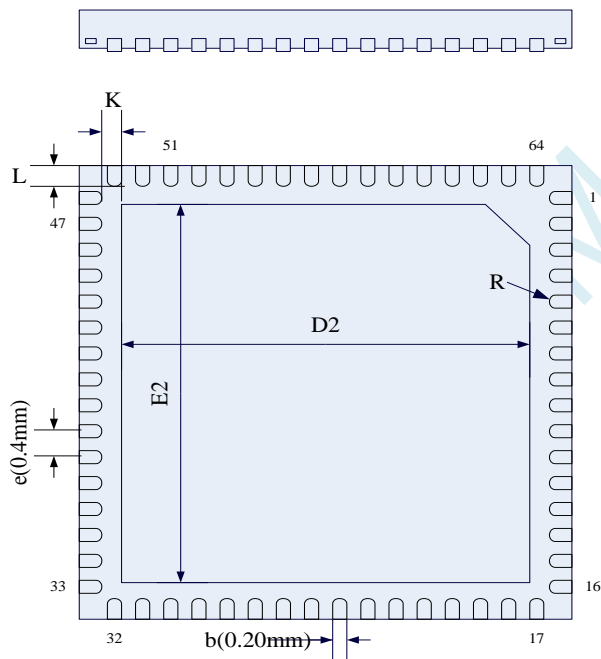
4.2 LQFP48 封装尺寸图 (9mm*9mm)



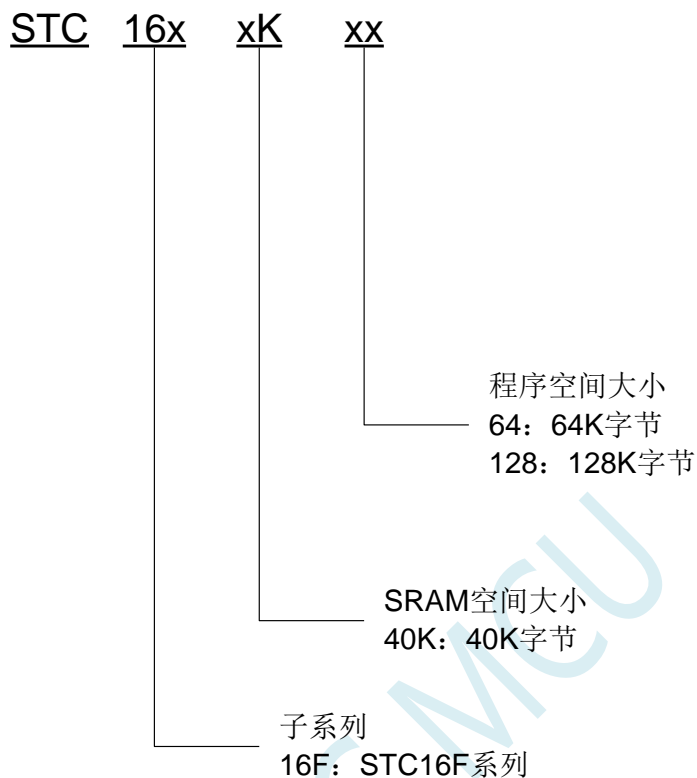
4.3 QFN64 封装尺寸图 (8mm*8mm) (暂无)



一般尺寸			
测量单位: 毫米/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0.00	0.02	0.05
A2	0.50	0.55	0.60
A3	0.20REF		
b	0.15	0.20	0.25
D	7.90	8.00	8.10
E	7.90	8.00	8.10
D2	5.90	6.00	6.10
E2	5.90	6.00	6.10
e	0.30	0.40	0.50
L	0.30	0.40	0.50
K	0.40		
R	0.09	-	-



4.4 STC16F 系列单片机命名规则



5 开发环境的建立与 ISP 下载

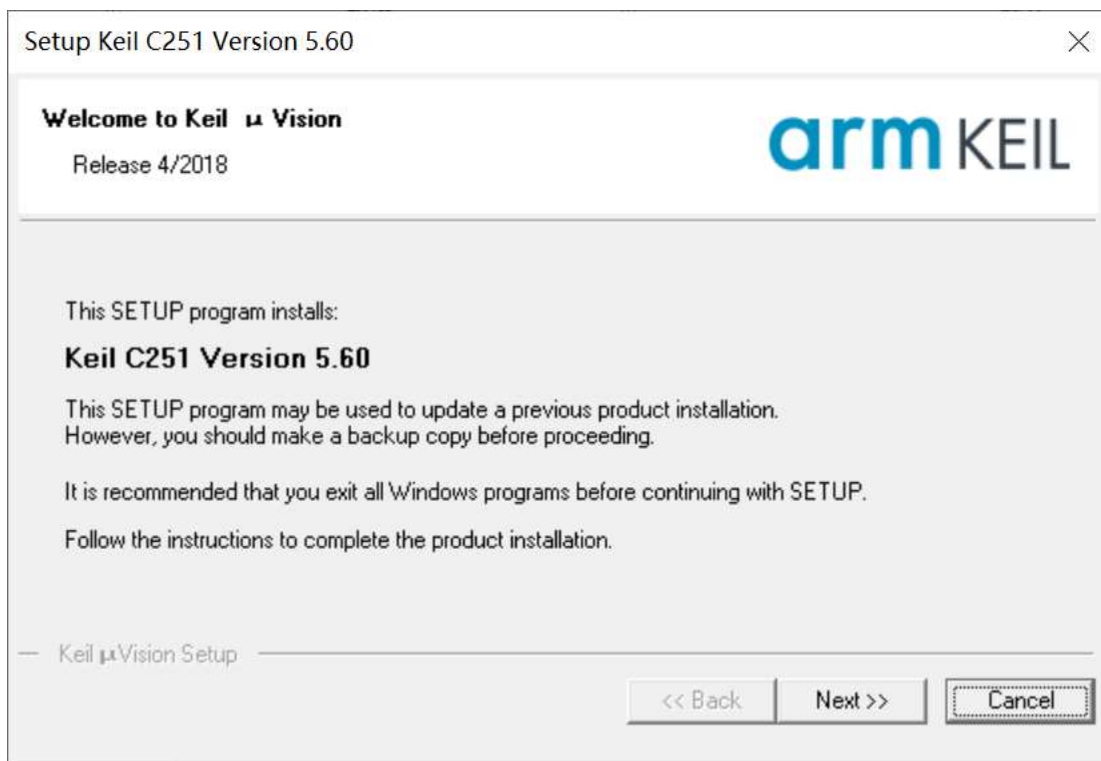
5.1 安装 Keil

首先登录 Keil 官网，下载最新版的 C251 安装包，下载链接如下：

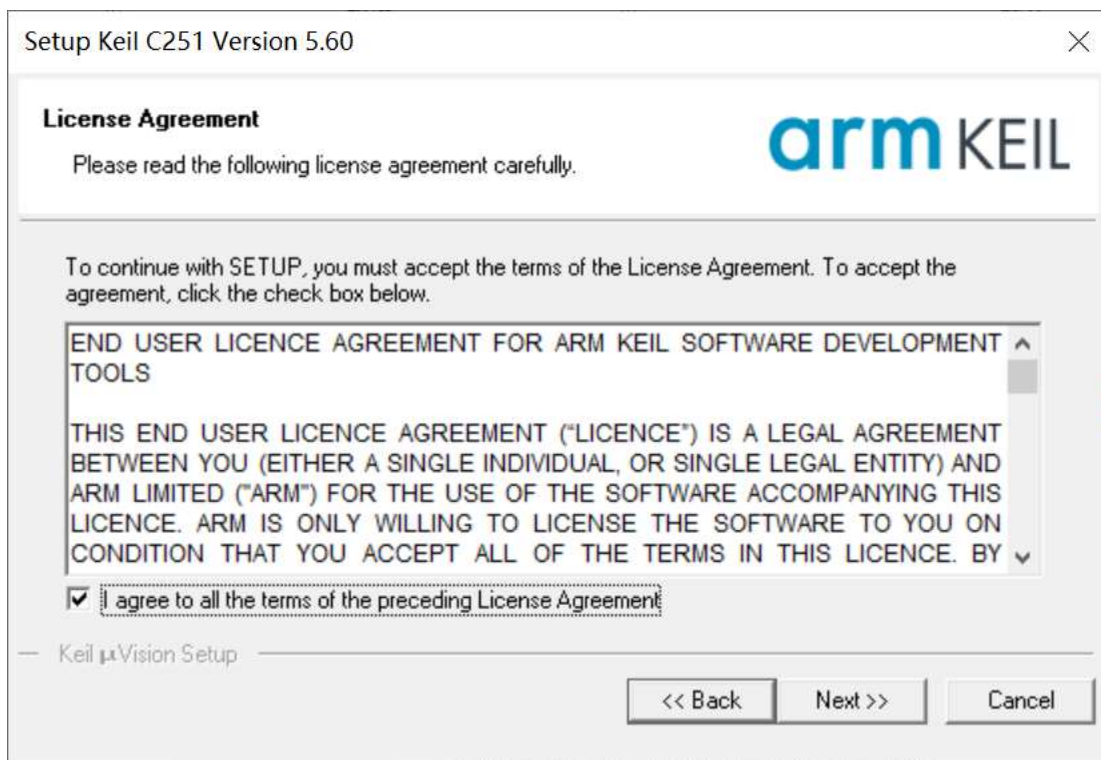
[Keil Product Downloads](#)

信息随便填写，点确定后进入下载页面进行下载。

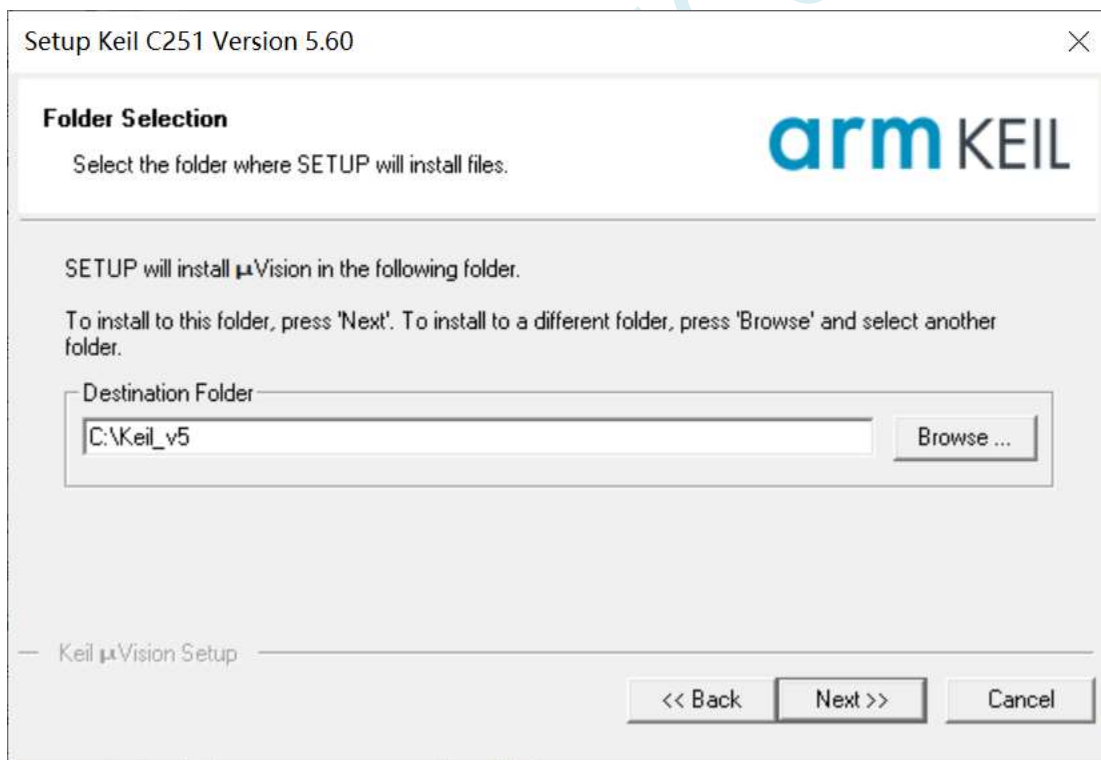
双击下载的安装包开始安装，点击“Next”：



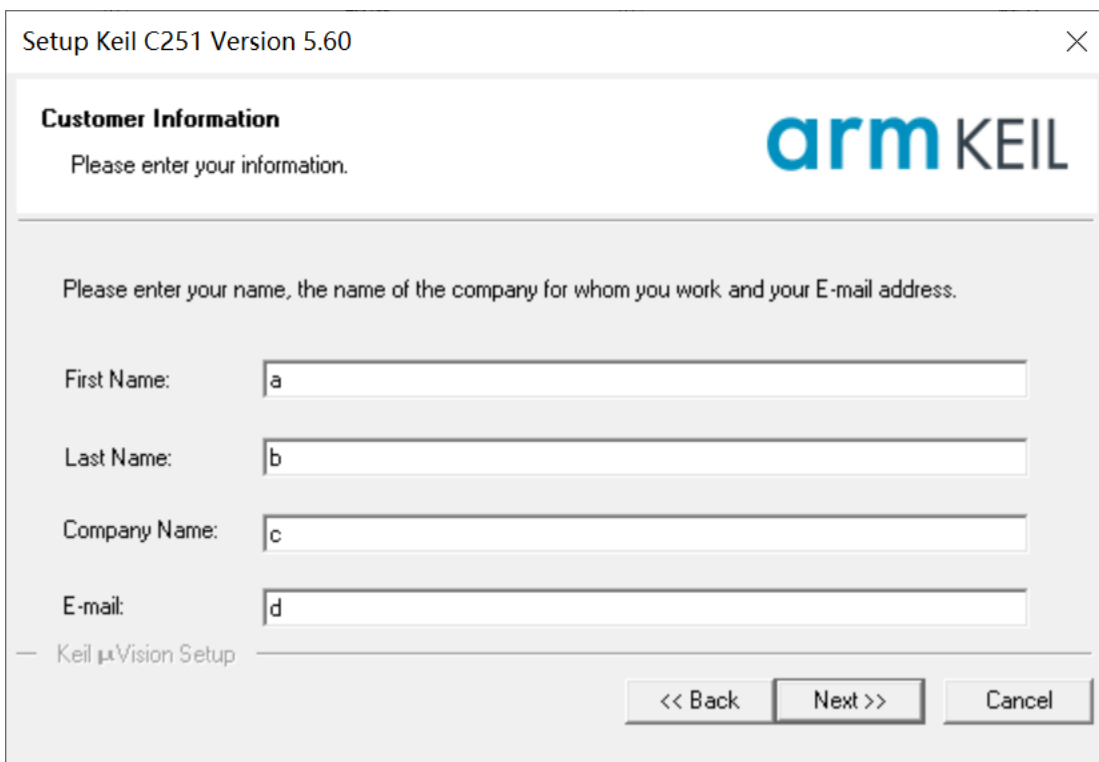
勾选“I agree to all the terms of the preceding License Agreement”，然后点击“Next”：



选择安装目录，然后点击“Next”：



填写个人信息，然后点击“Next”：



Setup Keil C251 Version 5.60

Customer Information

Please enter your information.

Please enter your name, the name of the company for whom you work and your E-mail address.

First Name:

Last Name:

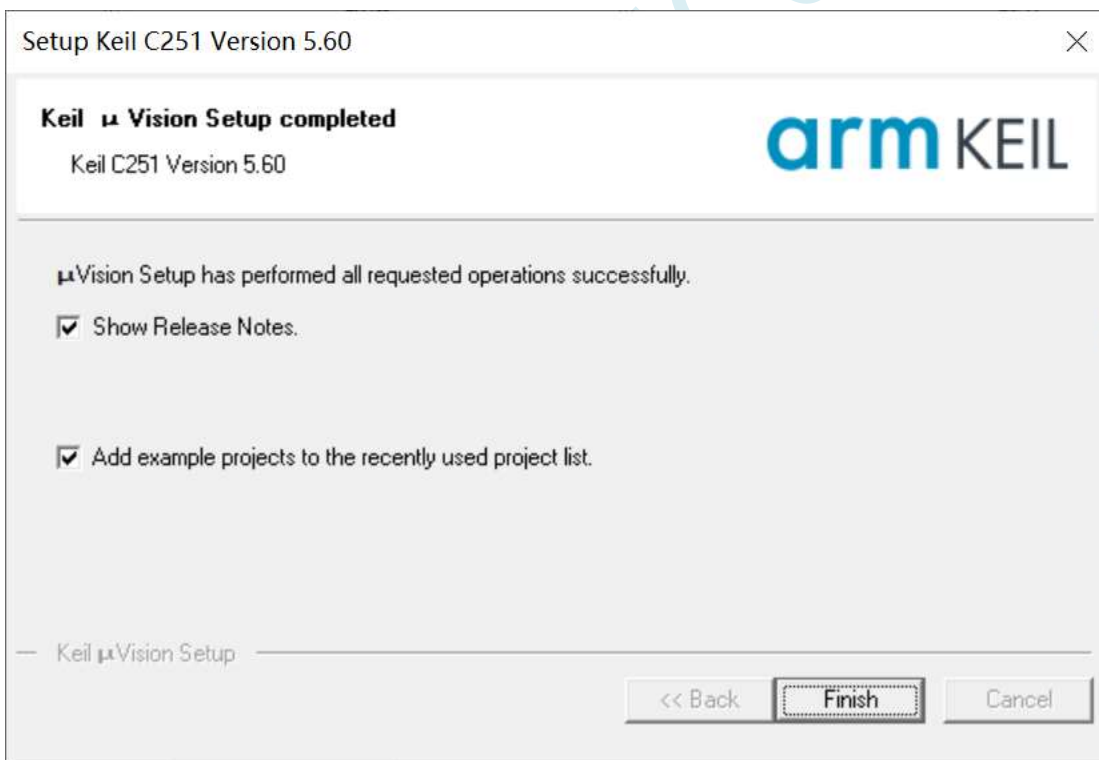
Company Name:

E-mail:

— Keil μ Vision Setup —

<< Back Next >> Cancel

安装完成，点击“Finish”结束。



Setup Keil C251 Version 5.60

Keil μ Vision Setup completed

Keil C251 Version 5.60

μ Vision Setup has performed all requested operations successfully.

☒ Show Release Notes.

☒ Add example projects to the recently used project list.

— Keil μ Vision Setup —

<< Back Finish Cancel

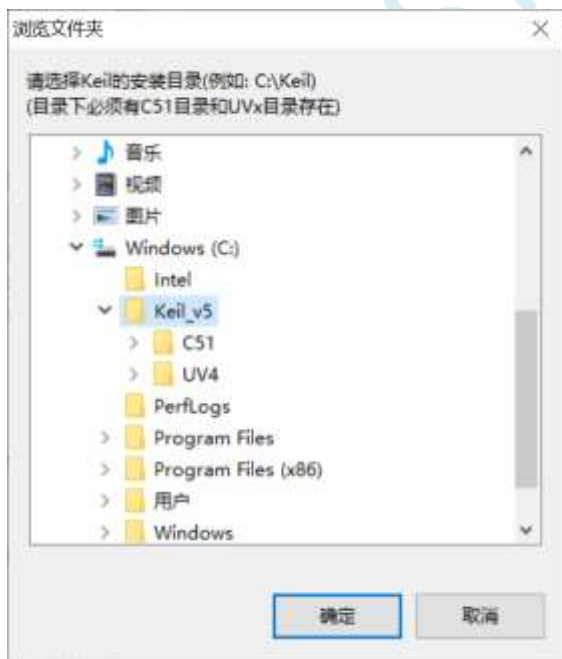
5.2 添加型号和头文件到 Keil

使用 Keil 之前需要先安装 STC 的仿真驱动。STC 的仿真驱动的安装步骤如下:

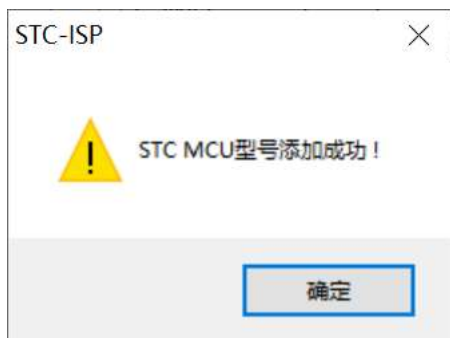
首先开 STC 的 ISP 下载软件, 然后在软件右边功能区的“Keil 仿真设置”页面中点击“添加型号和头文件到 Keil 中”按钮:



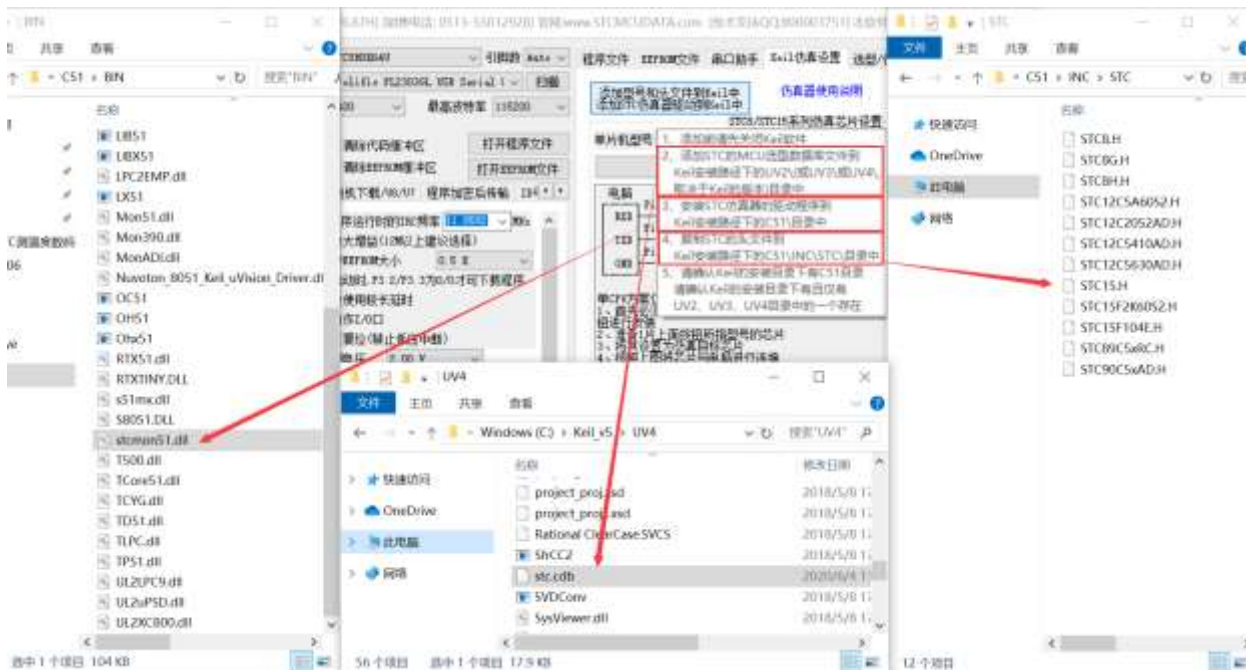
按下后会出现如下画面:



将目录定位到 Keil 软件的安装目录, 然后确定。安装成功后会弹出如下的提示框:



在 Keil 的相关目录中可以看到如下的文件，即表示驱动正确安装了



5.3 使用 ISP 进行烧录

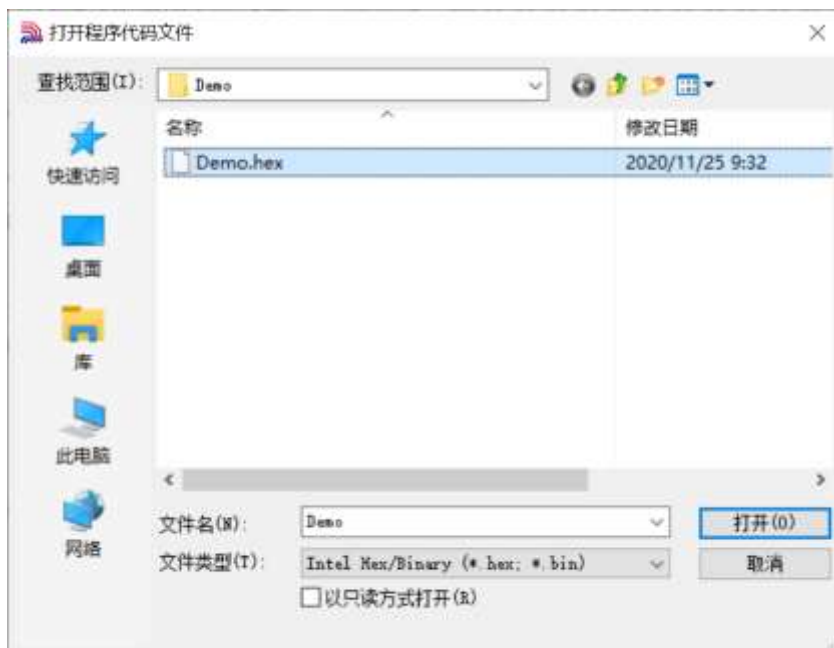
首先使用串口工具将待烧录芯片与电脑正确连接，然后打开 STC 的 ISP 下载软件（例如：“STC-ISP (Ver6.87Q)” 及其以后的版本）



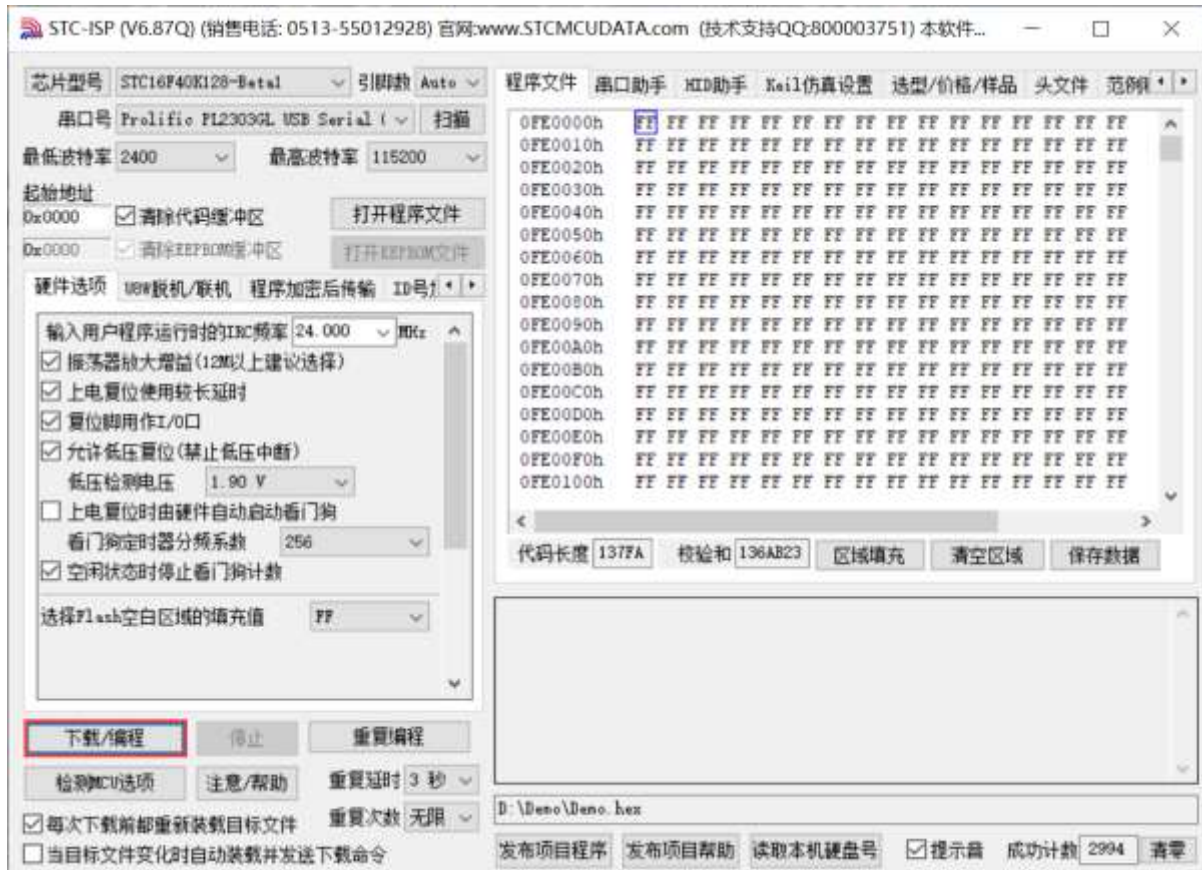
在上面的界面中，下面几点需要注意：

- 1、选择待烧录的单片机型号，如：“STC16F40K128-Beta1”
- 2、串口必须选择串口工具所对应的串口号。

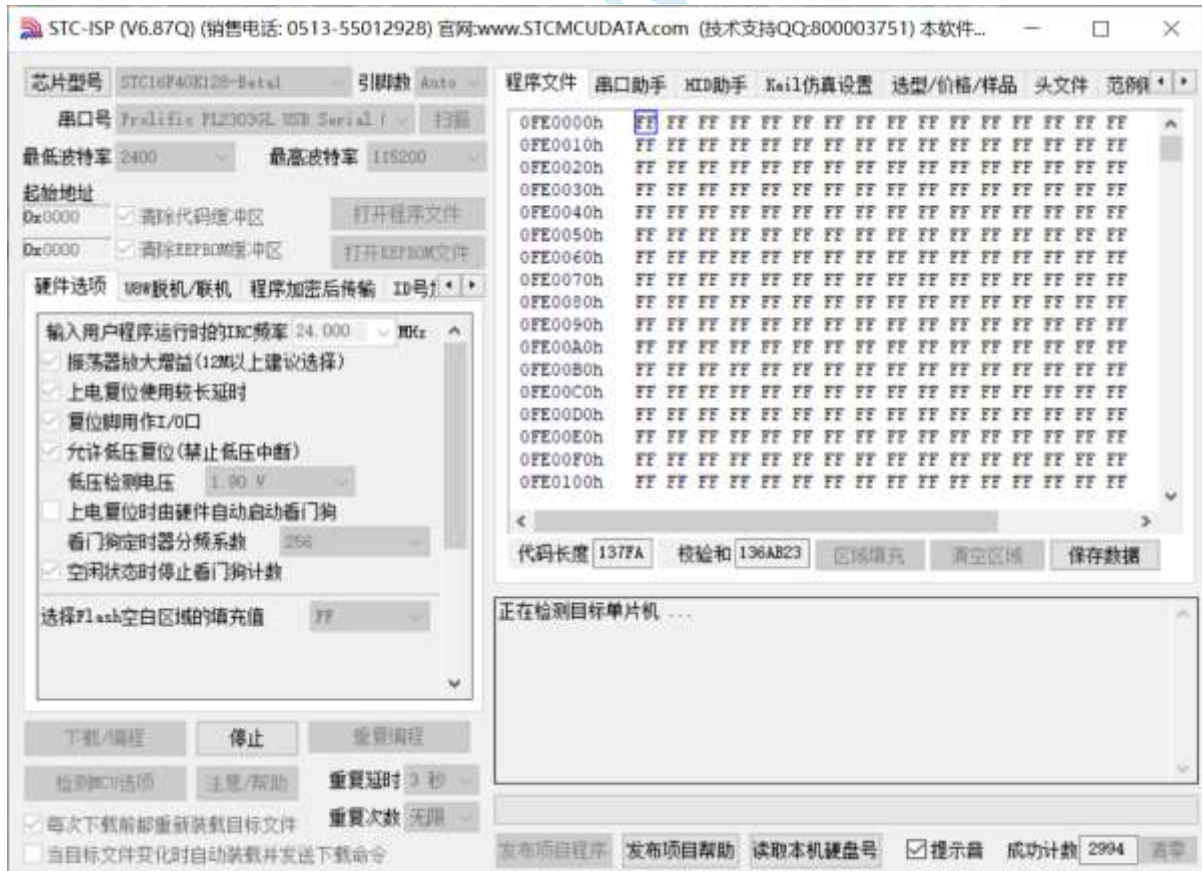
点击界面中的“打开程序文件”按钮，在出现的打开程序代码文件的对话框中选择需要下载的文件：



文件正确打开后，设置对应的硬件选项，然后点击界面中的“下载/编程”按钮开始下载流程：



此时 ISP 软件开始试图与单片机进行握手，检测目标单片机，如下图：

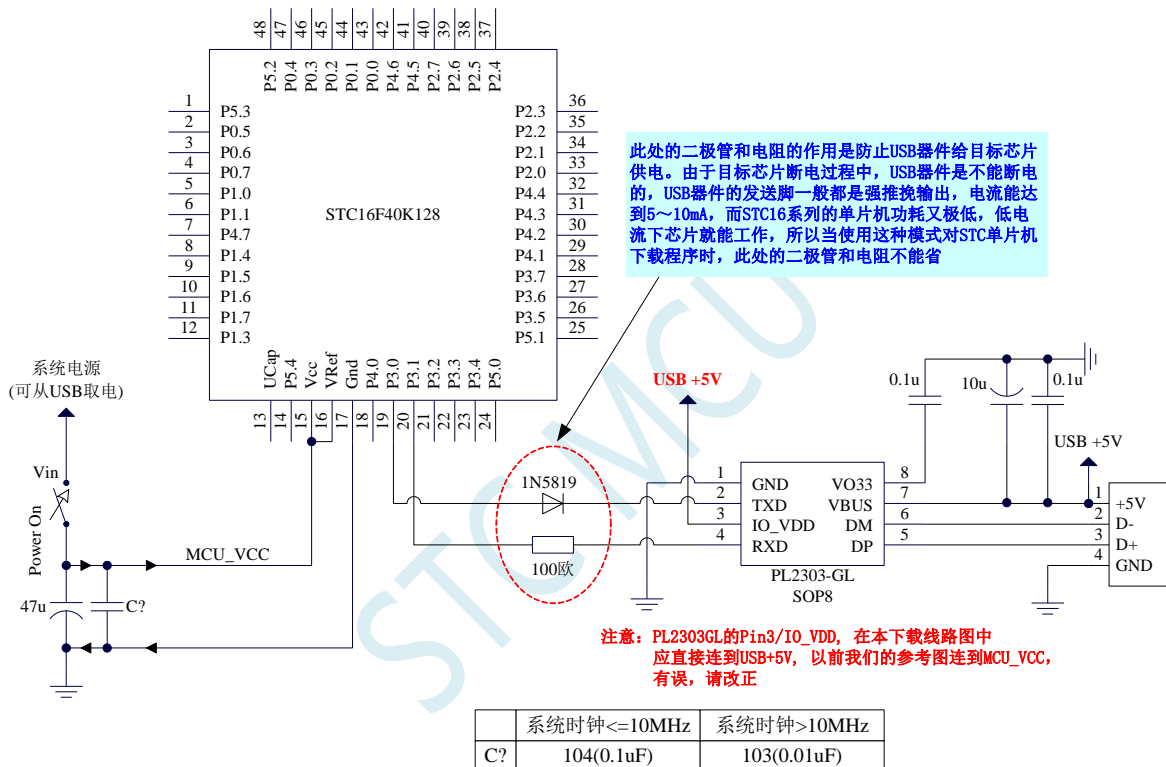


接下来需要给单片机进行上电，上电复位过程中检测到握手信号就进入下载流程开始下载。若下载成功，会出现操作成功的提示画面。



5.4 ISP 下载典型应用线路图

5.4.1 使用 PL2303-GL 下载



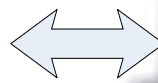
ISP 下载步骤:

- 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）
- 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

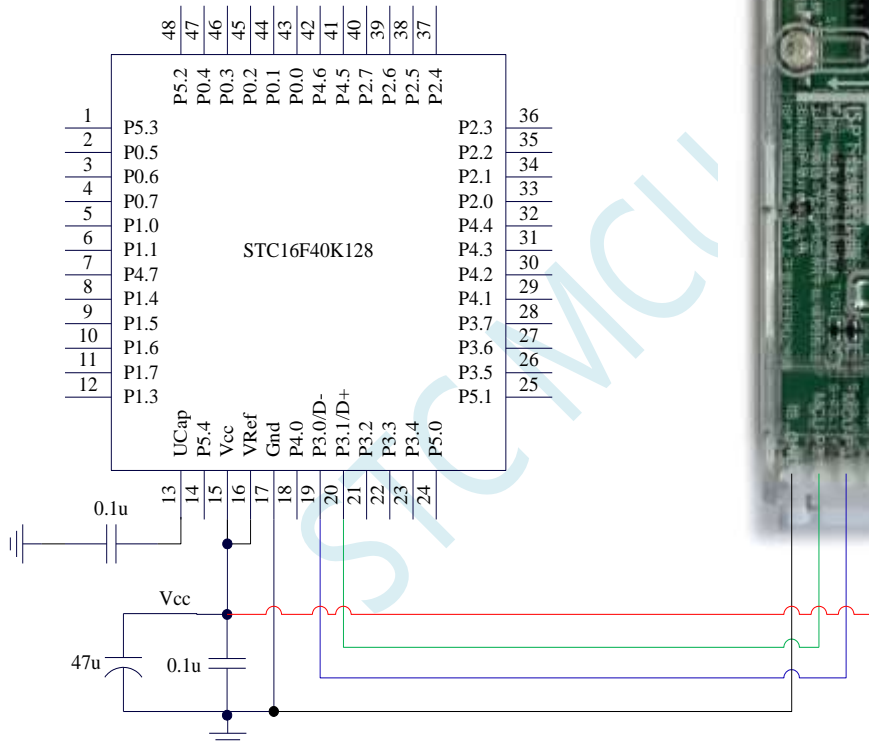
注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

5.4.2 使用通用 USB 转串口工具下载

连接
电脑/PC



通用USB转串口工具
(人民币30元)



ISP 下载步骤:

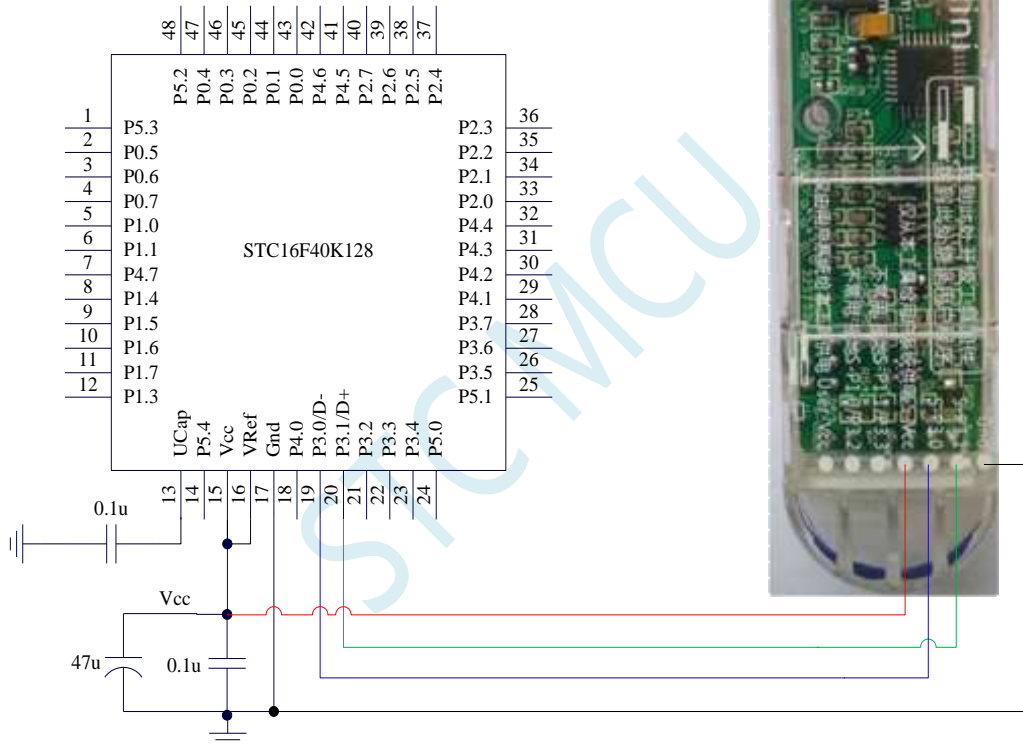
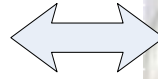
- 1、按照如图所示的连接方式将通用 USB 转串口工具和目标芯片连接
- 2、按下电源按钮，确定目标芯片处于**停电状态**（上电指示灯为灭的状态）。**注意：工具第一次上电时是不对外供电的，因此若是第一次上电使用此工具，可跳过此步。**
- 3、点击 STC-ISP 下载软件中的“下载/编程”按钮
- 4、再次按下电源按钮，给目标芯片上电（上电指示灯为亮的状态）
- 5、开始 ISP 下载

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。

5.4.3 使用 U8-Mini 工具下载，支持 ISP 在线和脱机下载

STC USB型 脱机/联机烧录工具
U8W Mini (人民币50元)

连接
电脑/PC



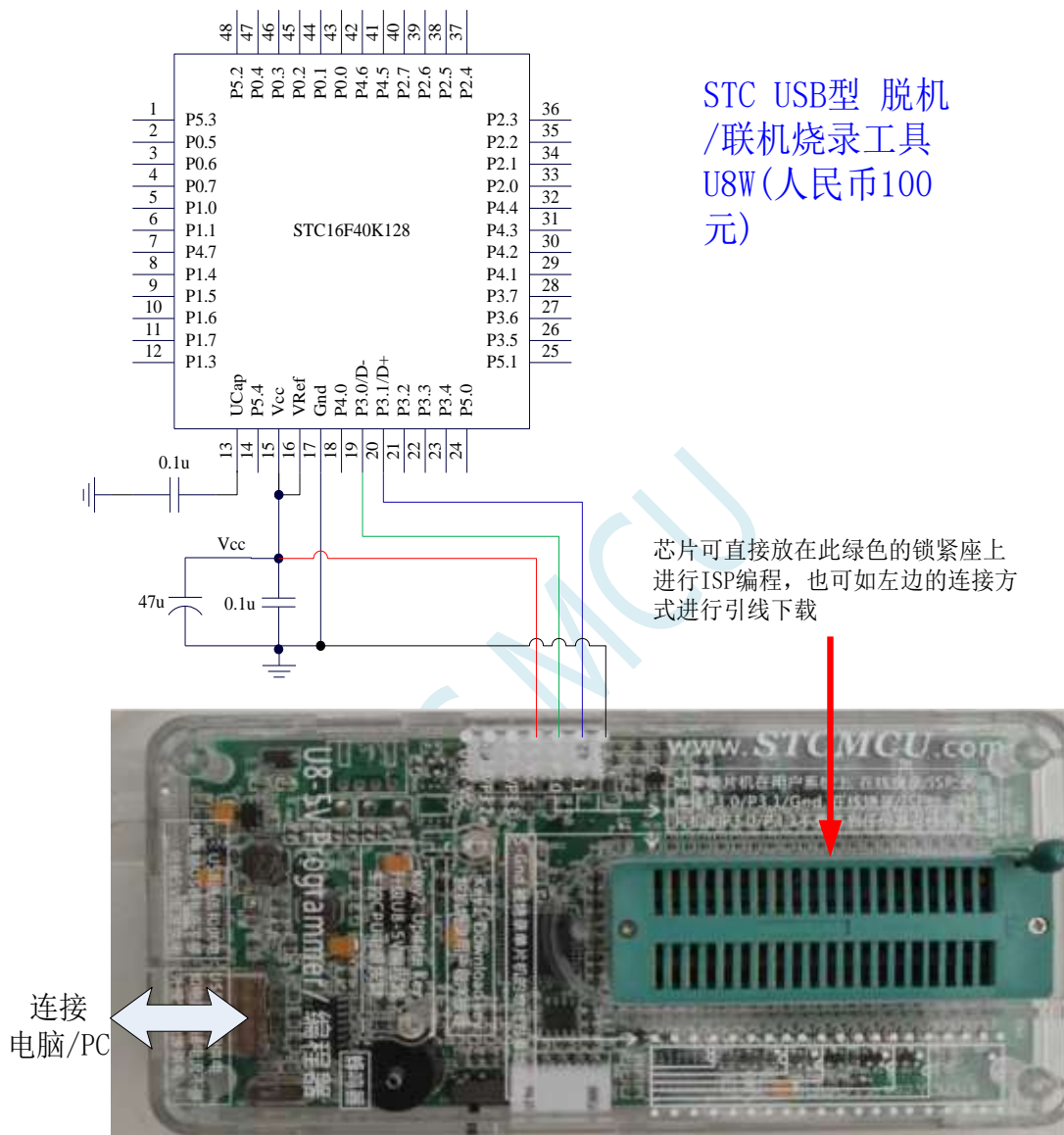
ISP 下载步骤:

- 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。

5.4.4 使用 U8W 工具下载，支持 ISP 在线和脱机下载



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

开始 ISP 下载

5.4.5 U8W 直通模式，可用于仿真、串口通信

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：

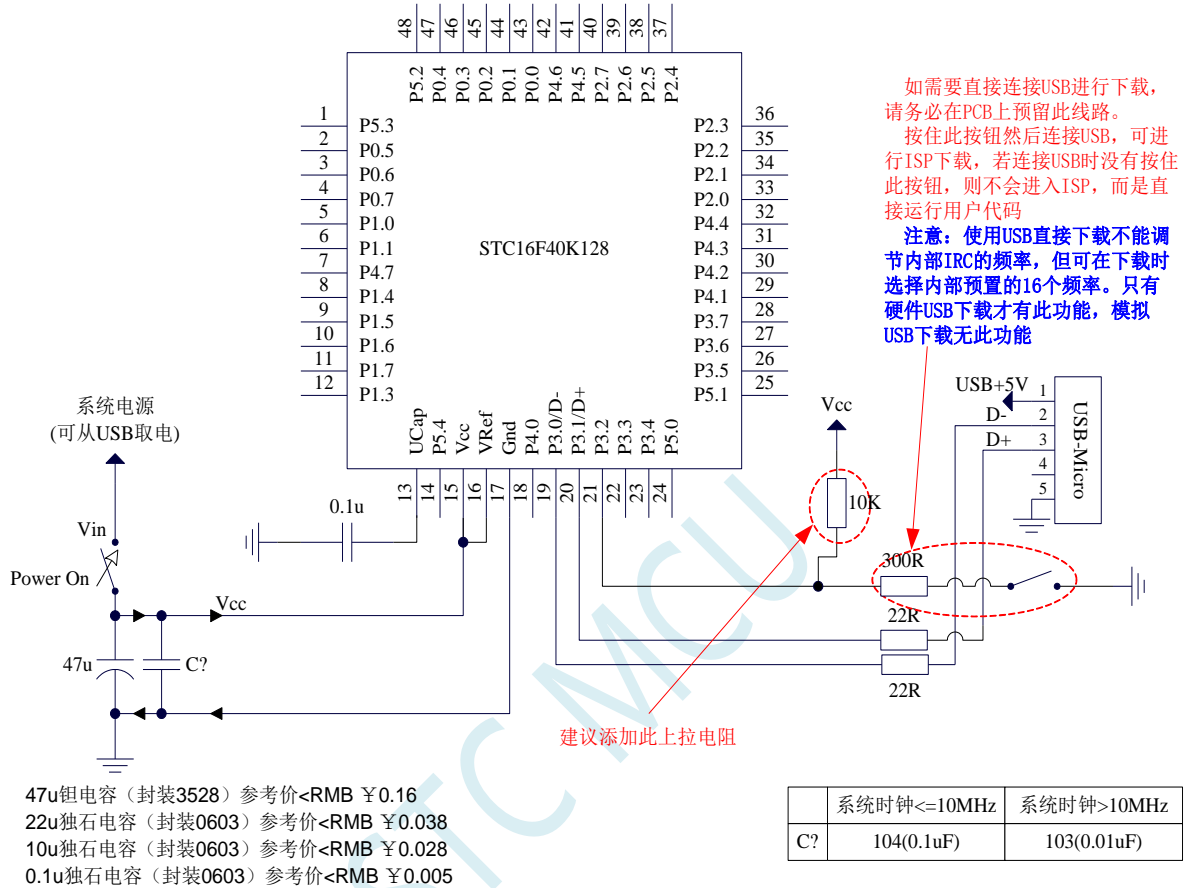
- 1、首先 U8W/U8W-Mini 固件必须升级到 v1.37 及以上版本
- 2、U8W/U8W-Mini 上电后为正常下载模式，此时按住工具上的 Key1（下载）按键不要松开，再按一下 Key2（电源）按键，然后放开 Key2（电源）按键，再松开 Key1（下载）按键，U8W/U8W-Mini 会进入 USB 转串口直通模式。（按下 Key1 → 按下 Key2 → 松开 Key2 → 松开 Key1）
- 3、进入直通模式的 U8W/U8W-Mini 工具只是简单的 USB 转串口不具备脱机下载功能，若需要恢复 U8W/U8W-Mini 的原有功能，只需要再次单独按一下 Key2（电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

STC MCU

5.4.6 硬件 USB 直接 ISP 下载，仅支持 ISP 下载，暂不支持仿真

注 1: 使用 USB 下载时需要将 P3.2 接 Gnd 才可进行正常下载

注 2: 若不需要进行 USB 下载，芯片复位时 P3.0/P3.1/P3.2 不可同时为低电平，否则芯片将会一直处于 USB 下载模式而不会运行用户代码

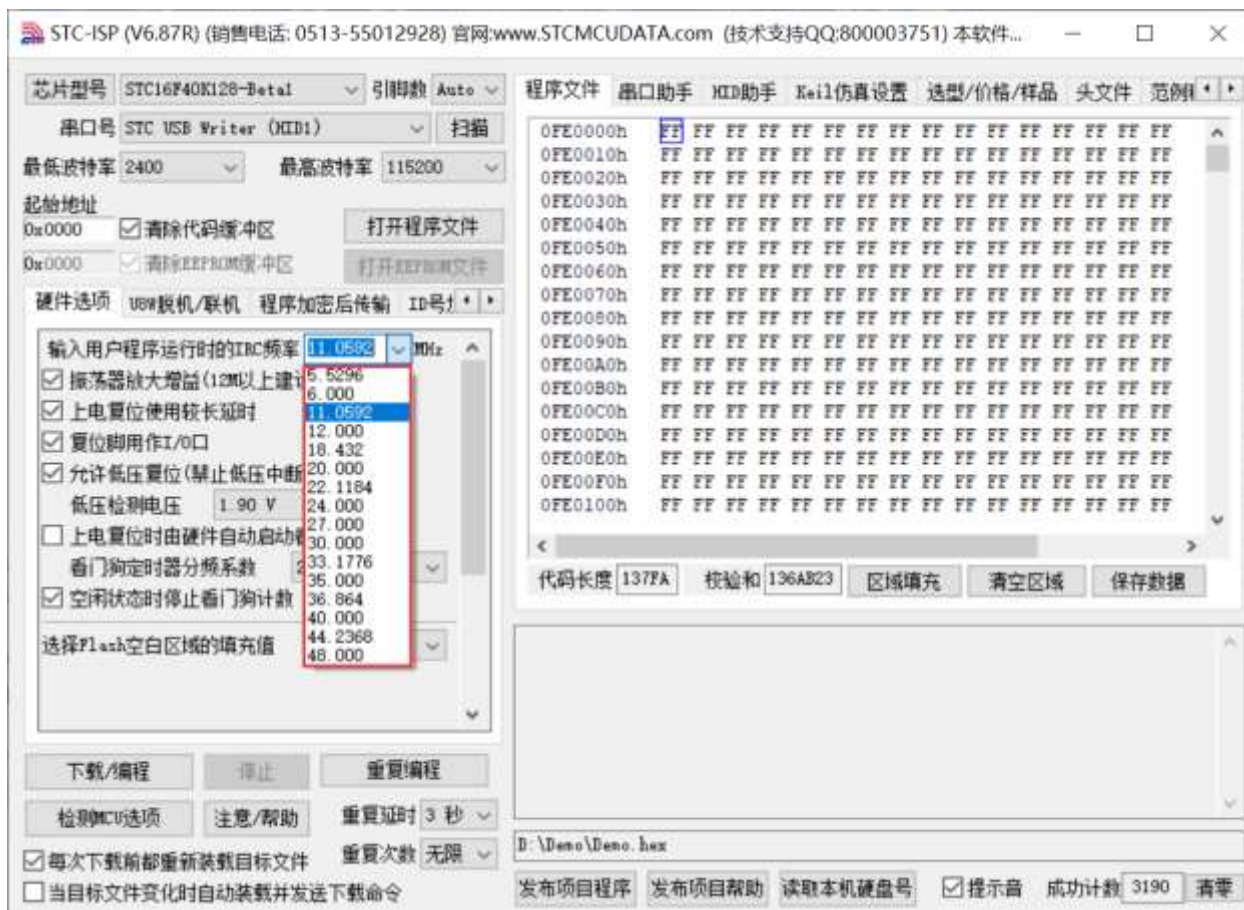


ISP 下载步骤:

- 1、将目标芯片停电
- 2、P3.0/P3.1 按照如图所示的连接方式与 USB 端口连接好
- 3、将 P3.2 与 GND 短接
- 4、给目标芯片上电，并等待 STC-ISP 下载软件中自动识别出“STC USB Writer (HID1)”
- 5、点击下载软件中的“下载/编程”按钮（注意：与串口下载的操作顺序不同）
- 6、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。

当用户使用硬件 USB 对 STC16F40K128 系列进行 ISP 下载时不能调节内部 IRC 的频率，但用户可用选择内部预置的 16 个频率（分别是 5.5296M、6M、11.0592M、12M、18.432M、20M、22.1184M、24M、27M、30M、33.1776M、35M、36.864M、40M、44.2368M 和 48M）。下载时用户只能从频率下拉列表中进行选择其中之一，而不能手动输入其他频率。（使用串口下载则可用输入 4M~48M 之间的任意频率）。详情见下页的图示：

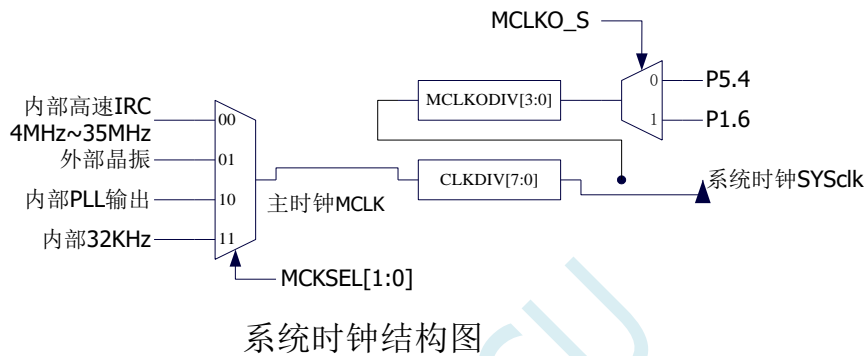


6 时钟、复位与电源管理

6.1 系统时钟控制

系统时钟控制器为单片机的 CPU 和所有外设系统提供时钟源，系统时钟有 4 个时钟源可供选择：内部高精度 IRC、内部 32KHz 的 IRC（误差较大）、外部晶振、内部 PLL 输出时钟。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



6.2 相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CKSEL	时钟选择寄存器	7EFE00H	-	-	-	-	-	-	MCKSEL[1:0]		xxxx,xx00
CLKDIV	时钟分频寄存器	7EFE01H									0000,0100
IRC24MCR	内部 24M 振荡器控制寄存器	7EFE02H	ENIRC24M	-	-	-	-	-	-	IRC24MST	1xxx,xxx0
XOSCCR	外部晶振控制寄存器	7EFE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST	00xx,xxx0
IRC32KCR	内部 32K 振荡器控制寄存器	7EFE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0
PLLCR	内部 PLL 控制寄存器	7EFE05H	ENPLL	PLLSEL[1:0]		PLLINCTRL[1:0]		PLLTOMSEL[1:0]		PLLST	0000,0000
USCON1	USB 内核电压控制寄存器 1	7EFE06H	ENLDO33	TRIMPLL[1:0]		TRIMLSO33[3:0]				-	0000,0000
MCLKOCR	主时钟输出控制寄存器	7EFE07H	MCLKO_S	MCLKODIV[6:0]							0000,0000

6.2.1 系统时钟选择寄存器（CKSEL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CKSEL	7EFE00H	-						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	外部晶振
10	内部 PLL 时钟输出
11	内部 32KHz 低速 IRC

6.2.2 时钟分频寄存器 (CLKDIV)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CLKDIV	7EFE01H								

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

6.2.3 内部高精度 IRC 控制寄存器 (IRC24MCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRC24MCR	7EFE02H	ENIRC24M	-	-	-	-	-	-	IRC24MST

ENIRC24M: 内部高精度 IRC 使能位

- 0: 关闭内部高精度 IRC
- 1: 使能内部高精度 IRC

IRC24MST: 内部高精度 IRC 频率稳定标志位。(只读位)

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

6.2.4 外部振荡器控制寄存器 (XOSCCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
XOSCCR	7EFE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST

ENXOSC: 外部晶体振荡器使能位

- 0: 关闭外部晶体振荡器
- 1: 使能外部晶体振荡器

XITYPE: 外部时钟源类型

- 0: 外部时钟源是外部时钟信号 (或有源晶振)。信号源只需连接单片机的 XTALI (P1.7)
- 1: 外部时钟源是晶体振荡器。信号源连接单片机的 XTALI (P1.7) 和 XTALO (P1.6)

XOSCST: 外部晶体振荡器频率稳定标志位。(只读位)

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

6.2.5 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	7EFE04H	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。

6.2.6 内部 PLL 控制寄存器 (PLLCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PLLCR	7EFE05H	ENPLL	PLLSSEL[1:0]		PLLINCTRL[1:0]		PLLTOMSEL[1:0]		PLLST

ENPLL: 内部 PLL 使能位

0: 关闭内部 PLL

1: 使能内部 PLL

PLLSSEL[1:0]: PLL 时钟源选择

PLLSSEL[1:0]	PLL 时钟源
00	内部高精度 IRC
01	外部晶振
10	内部 32KHz 低速 IRC
11	P1.7 口的时钟输入

PLLINCTRL[1:0]: PLL 时钟源分频控制 (产生 6MHz 提供给 USBPLL)

PLLINCTRL [1:0]	分频控制
00	不分频
01	2 分频
10	4 分频
11	内部高精度 IRC2 分频

PLLTOMSEL[1:0]: PLL 输出时钟到主时钟 MCLK 控制

PLLTOMSEL [1:0]	MCLK 时钟
00	PLL 分频后的 6MHz
01	PLL 输出的 24MHz
10	PLL 输出的 32MHz
11	PLL 输出的 48MHz

PLLST: PLL 输出时钟稳定标志位。(只读位)

0: 内部 PLL 输出时钟未稳定

1: 内部 PLL 输出时钟已稳定

6.2.7 主时钟输出控制寄存器 (MCLKOCR)

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

MCLKODIV[6:0]: 主时钟输出分频系数

(注意: 主时钟分频输出的时钟源是经过 CLKDIV 分频后的系统时钟)

MCLKODIV[6:0]	系统时钟分频输出频率
0000000	不输出时钟
0000001	SYSClk/1

0000010	SYSClk /2
0000011	SYSClk /3
...	...
1111110	SYSClk /126
1111111	SYSClk /127

MCLKO_S: 系统时钟输出管脚选择

0: 系统时钟分频输出到 P5.4 口

1: 系统时钟分频输出到 P1.6 口

6.3 STC16F 系列内部 IRC 频率调整

STC16F 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%。

STC16F 系列内部 IRC 有两个频段，频段的中心频率分别为 20MHz 和 35MHz，20M 频段的调节范围约为 15.5MHz~27MHz，35M 频段的调节范围约为 27.5MHz~47MHz（注意：不同的芯片以及不同的生成批次可能会有约 5%左右的制造误差）。**经实际测试，部分芯片的最高工作频率只能为 39.5MHz，所以为了安全起见，建议用户在 ISP 下载时设置 IRC 频率不要高于 35MHz。**

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

内部 IRC 频率调整主要使用下面的 4 个寄存器进行调整

相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
IRCBAND	IRC 频段选择	A9H	-	-	-	-	-	-	-	SEL	0000,00nn
LIRTRIM	IRC 频率微调寄存器	9EH	-	-	-	-	-	-	LIRTRIM[1:0]		0000,00nn
IRTRIM	IRC 频率调整寄存器	9FH	IRTRIM[7:0]								nnnn,nnnn
CLKDIV	时钟分频寄存器	7EFE01H	CLKDIV[7:0]								0000,0100

6.3.1 IRC 频段选择寄存器（IRCBAND）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRCBAND	A9H	-	-	-	-	-	-	-	SEL

SEL：频段选择

0：选择 20MHz 频段

1：选择 35MHz 频段

6.3.2 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。

6.3.3 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	-	-	-	-	-	-	IRTRIM[1:0]	

LIRTRIM[1:0]: 内部高精度 IRC 频率微调寄存器

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

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

6.3.4 时钟分频寄存器（CLKDIV）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CLKDIV	7EFE01H								

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

STC16F 系列内部的两个频段的可调范围分别为 15.5MHz~27MHz 和 27.5MHz~47MHz, 所以两个频段直接存在调节盲区(如 27MHz~27.5MHz 之间的频率)。虽然 35MHz 频段的上限可调到 40MHz 以上, 但芯片内部的程序存储器无法运行到 40MHz 以上的速度, 所以用户在 ISP 下载时设置内部 IRC 频率不能高于 40MHz, 一般建议用户设置为 35MHz 以下。若用户需要较低的工作频率时, 可使用 CLKDIV 寄存器对调节后的频率进行分频, 例如用户需要 11.0592MHz 的频率, 使用内部 IRC 直接调整是无法得到这个频率的, 但可将内部 IRC 调整到 22.1184MHz, 在使用 CLKDIV 进行 2 分频即可得到 11.0592MHz。

6.4 系统复位

STC16F 系列单片机的复位分为硬件复位和软件复位两种。

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

- 上电复位
- 低压复位
- 复位脚复位（低电平复位）
- 看门狗复位

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

- 写 IAP_CONTR 的 SWRST 所触发的复位

相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
WDT_CONTR	看门狗控制寄存器	FDH	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	-	IAP_WT[2:0]			0000,x000
RSTCFG	复位配置寄存器	FFH	-	ENLVR	-	P54RST	-	-	LVDS[1:0]		0000,0000

6.4.1 看门狗控制寄存器（WDT_CONTR）

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

WDT_FLAG：看门狗溢出标志

看门狗发生溢出时，硬件自动将此位置 1，需要软件清零。

EN_WDT：看门狗使能位

- 0：对单片机无影响
- 1：启动看门狗定时器

CLR_WDT：看门狗定时器清零

- 0：对单片机无影响
- 1：清零看门狗定时器，硬件自动将此位复位

IDL_WDT：IDLE 模式时的看门狗控制位

- 0：IDLE 模式时看门狗停止计数
- 1：IDLE 模式时看门狗继续计数

WDT_PS[2:0]：看门狗定时器时钟分频系数

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

看门狗溢出时间计算公式如下:

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

6.4.2 IAP 控制寄存器 (IAP_CONTR)

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

6.4.3 复位配置寄存器 (RSTCFG)

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

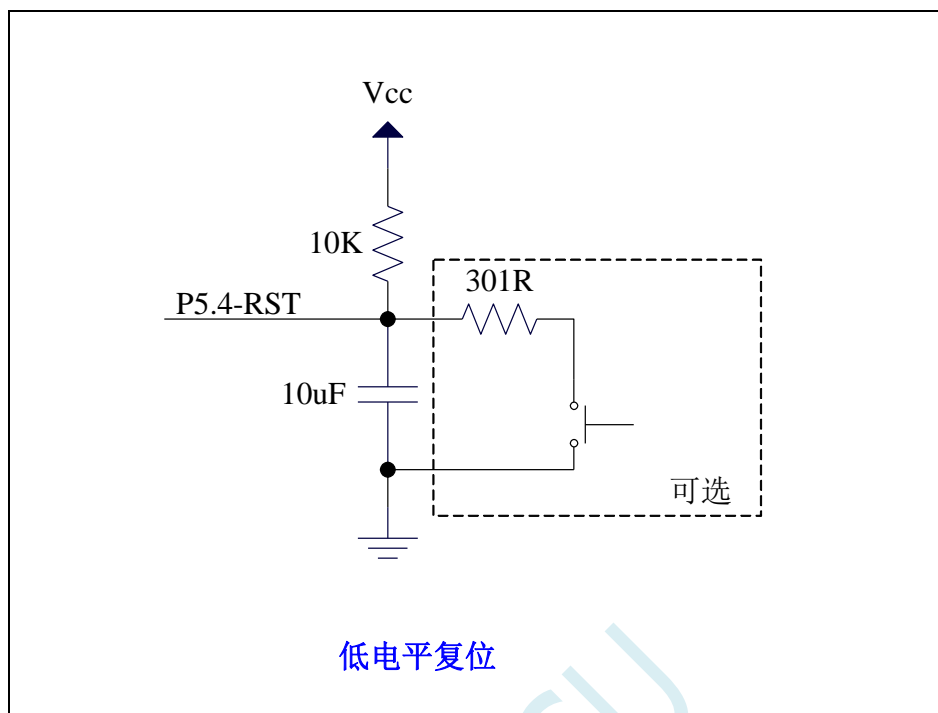
0: RST 管脚用作普通 I/O 口 (P54)

1: RST 管脚用作复位脚 (低电平复位)

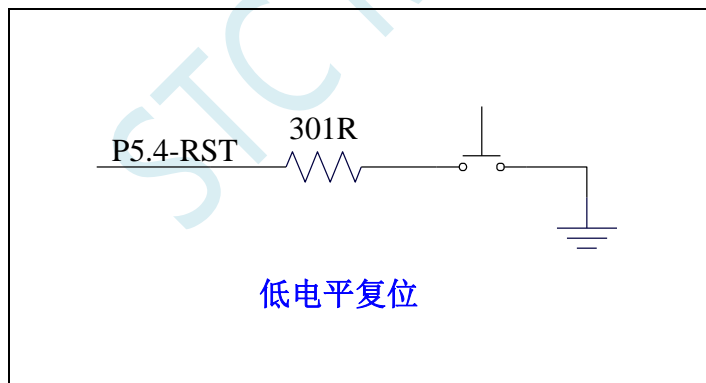
LVDS[1:0]: 低压检测门槛电压设置

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

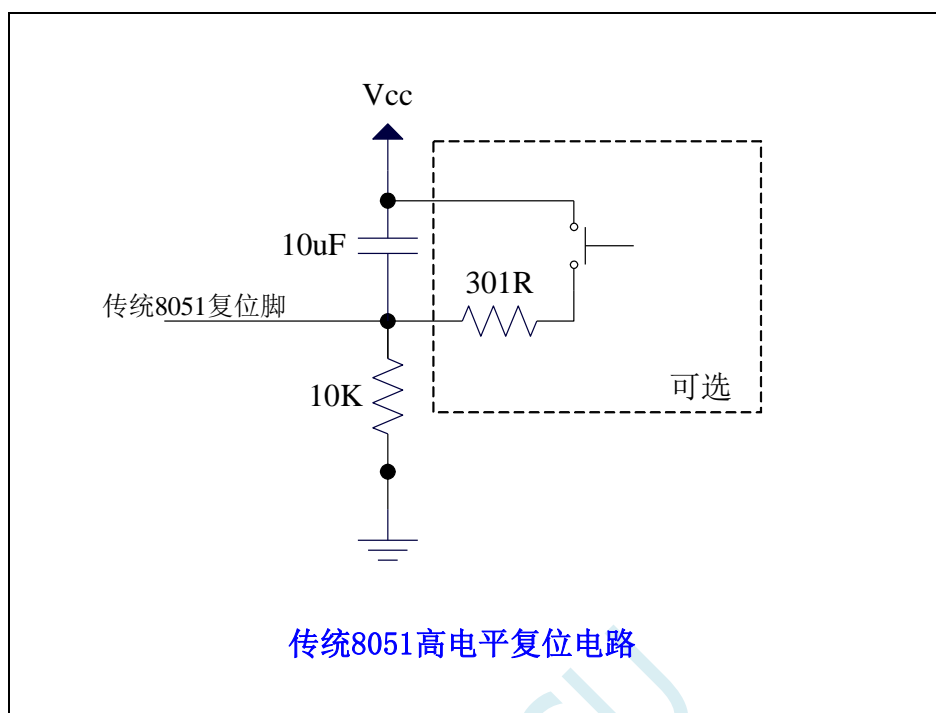
6.4.4 低电平上电复位参考电路（一般不需要）



6.4.5 低电平按键复位参考电路



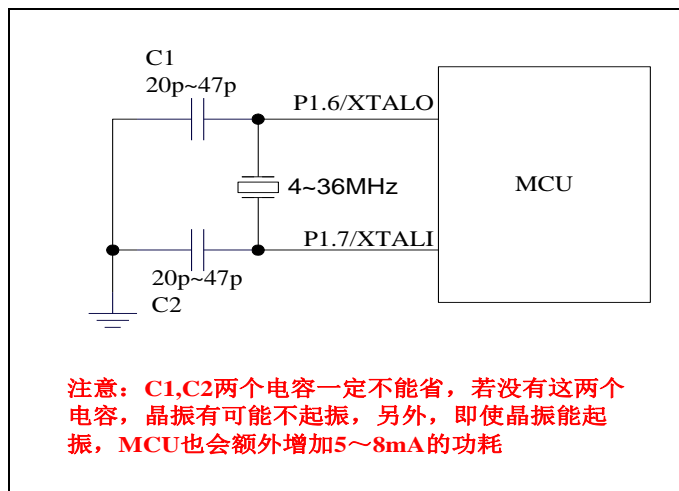
6.4.6 传统 8051 高电平上电复位参考电路



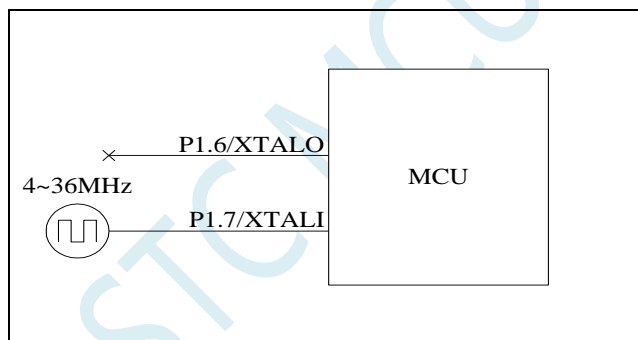
上图为传统 8051 的高电平复位电路，STC16 的复位为低电平复位，与传统复位电路不同

6.5 外部晶振及外部时钟电路

6.5.1 外部晶振输入电路



6.5.2 外部时钟输入电路（P1.6 不可用作普通 I/O）



6.6 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

6.6.1 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: 上电标志位。当硬件自动将此位置 1。

PD: 掉电模式控制位

0: 无影响

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

（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）。

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(P5.4)、T3(P0.4)、T4(P0.6)、RXD(P3.0/P3.6/P1.6/P4.3)、RXD2(P1.0/P4.6)、RXD3(P0.0/P5.0)、RXD4(P0.2/P5.2)、I2C_SDA(P1.4/P2.4/P3.3)以及比较器中断、低压检测中断、掉电唤醒定时器唤醒。

6.7 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。

6.7.1 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 STC16F 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（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，其中掉电唤醒定时器计数一次的时间就是由该时钟决定的。内部掉电唤醒定时器的时钟频率约为 32KHz，当然误差较大。用户可以通过读 RAM 区 F8H 和 F9H 的内容（F8H 存放频率的高字节，F9H 存放低字节）来获取内部掉电唤醒专用定时器出厂时所记录的时钟频率。

掉电唤醒专用定时器计数时间的计算公式如下所示：（F_{wt}为我们从 RAM 区 F8H 和 F9H 获取到的内部掉电唤醒专用定时器的时钟频率）

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

假设 F_{wt}=32KHz，则有：

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

STC MCU

6.8 范例程序

6.8.1 选择系统时钟源

C 语言代码

//测试工作频率为11.0592MHz

#include "stc16.h"

//头文件见附录

void main()

{

 P0M0 = 0x00;

 P0M1 = 0x00;

 P1M0 = 0x00;

 P1M1 = 0x00;

 P2M0 = 0x00;

 P2M1 = 0x00;

 P3M0 = 0x00;

 P3M1 = 0x00;

 P4M0 = 0x00;

 P4M1 = 0x00;

 P5M0 = 0x00;

 P5M1 = 0x00;

 P_SW2 = 0x80;

 CKSEL = 0x00;

//选择内部IRC (默认)

 P_SW2 = 0x00;

/*

 P_SW2 = 0x80;

 XOSCCR = 0xc0;

 while (!(XOSCCR & 1));

 CLKDIV = 0x00;

 CKSEL = 0x01;

 P_SW2 = 0x00;

//启动外部晶振

//等待时钟稳定

//时钟不分频

//选择外部晶振

*/

/*

 P_SW2 = 0x80;

 IRC32KCR = 0x80;

 while (!(IRC32KCR & 1));

 CLKDIV = 0x00;

 CKSEL = 0x03;

 P_SW2 = 0x00;

//启动内部32K IRC

//等待时钟稳定

//时钟不分频

//选择内部32K

*/

 while (1);

}

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC)

;头文件见附录

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, #00H                                ;选择内部IRC (默认)
MOV          WR6, #WORD0 CKSEL
MOV          WR4, #WORD2 CKSEL
MOV          @DR4, R11
MOV          P_SW2, #00H

; MOV          P_SW2, #80H
; MOV          A, #0C0H                                ;启动外部晶振
; MOV          WR6, #WORD0 XOSCCR
; MOV          WR4, #WORD2 XOSCCR
; MOV          @DR4, R11
; MOV          R11, @DR4
; JNB          ACC.0, $-1                            ;等待时钟稳定
; CLR          A                                    ;时钟不分频
; MOV          WR6, #WORD0 CLKDIV
; MOV          WR4, #WORD2 CLKDIV
; MOV          @DR4, R11
; MOV          A, #01H                                ;选择外部晶振
; MOV          WR6, #WORD0 CKSEL
; MOV          WR4, #WORD2 CKSEL
; MOV          @DR4, R11
; MOV          P_SW2, #00H

; MOV          P_SW2, #80H
; MOV          A, #80H                                ;启动内部32K IRC
; MOV          WR6, #WORD0 IRC32KCR
; MOV          WR4, #WORD2 IRC32KCR
; MOV          @DR4, R11
; MOV          R11, @DR4
; JNB          ACC.0, $-1                            ;等待时钟稳定
; CLR          A                                    ;时钟不分频
; MOV          WR6, #WORD0 CLKDIV
; MOV          WR4, #WORD2 CLKDIV
; MOV          @DR4, R11
; MOV          A, #03H                                ;选择内部32K
; MOV          WR6, #WORD0 CKSEL
; MOV          WR4, #WORD2 CKSEL
; MOV          @DR4, R11
; MOV          P_SW2, #00H

JMP          $

```

END

6.8.2 主时钟分频输出

C 语言代码

//测试工作频率为 11.0592MHz

#include "stc16.h"

//头文件见附录

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

\$include (STC16.INC)

;头文件见附录

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      WR6, #WORD0 MCLKOCR
MOV      WR4, #WORD2 MCLKOCR
MOV      @DR4, R11
MOV      P_SW2, #00H

JMP      $

END

```

6.8.3 看门狗定时器应用

C 语言代码

//测试工作频率为 11.0592MHz

#include "stc16.h"

//头文件见附录

void main()

{

P0M0 = 0x00;

P0M1 = 0x00;

P1M0 = 0x00;

P1M1 = 0x00;

P2M0 = 0x00;

P2M1 = 0x00;

P3M0 = 0x00;

P3M1 = 0x00;

P4M0 = 0x00;

P4M1 = 0x00;

P5M0 = 0x00;

P5M1 = 0x00;

// WDT_CONTR = 0x23;

WDT_CONTR = 0x24;

// WDT_CONTR = 0x27;

P32 = 0;

//使能看门狗,溢出时间约为 0.5s

//使能看门狗,溢出时间约为 1s

//使能看门狗,溢出时间约为 8s

//测试端口

while (1)

{

// WDT_CONTR = 0x33;

WDT_CONTR = 0x34;

// WDT_CONTR = 0x37;

//清看门狗,否则系统复位

//清看门狗,否则系统复位

//清看门狗,否则系统复位

Display();

Scankey();

MotorDriver();

//显示模块

//按键扫描模块

//电机驱动模块

}

}

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC)

;头文件见附录

```
                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
```

6.8.4 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 "stc16.h"

//头文件见附录

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; //检查到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

\$include (STC16F.INC)

;头文件见附录

```
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
```

6.8.5 低压检测

C 语言代码

//测试工作频率为11.0592MHz

#include "stc16.h" //头文件见附录

```
#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
```

```
#define LVDF       0x20      //PCON.5
```

```
void Lvd_Isr() interrupt 6
{
    PCON &= ~LVDF;          //清中断标志
    P32 = ~P32;             //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;          //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

\$include (STC16.INC) ;头文件见附录

```
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
```

```
LVDF       EQU      20H      ;PCON.5
```

```
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

```

6.8.6 省电模式

C 语言代码

//测试工作频率为11.0592MHz

```

#include "stc16.h"          //头文件见附录
#include "intrins.h"

```

```

#define IDL      0x01        //PCON.0
#define PD       0x02        //PCON.1

```

```

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

\$include (STC16F.INC)

;头文件见附录

```

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

        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
;        MOV        PCON,#IDL        ;MCU 进入 IDLE 模式
        MOV        PCON,#PD        ;MCU 进入掉电模式
        NOP
        NOP
        NOP
        CLR        P3.5        ;测试端口
        JMP        $
END

```

6.8.7 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 "stc16.h" //头文件见附录

#include "intrins.h"

#define EX2 0x10

#define EX3 0x20

#define EX4 0x40

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)
{
    P1I = ~P1I;
}

```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16F.INC)

;头文件见附录

```

EX2      EQU      10H
EX3      EQU      20H
EX4      EQU      40H

ORG      0000H
LJMP     MAIN

ORG      0003H
LJMP     INT0ISR

ORG      0013H
LJMP     INT1ISR

ORG      0053H
LJMP     INT2ISR

ORG      005BH
LJMP     INT3ISR

ORG      0083H
LJMP     INT4ISR

```



```

INT0ISR:      ORG          0100H

                CPL          P1.0          ;测试端口
                RETI

INT1ISR:      CPL          P1.0          ;测试端口
                RETI

INT2ISR:      CPL          P1.0          ;测试端口
                RETI

INT3ISR:      CPL          P1.0          ;测试端口
                RETI

INT4ISR:      CPL          P1.0          ;测试端口
                RETI

MAIN:

                MOV          SP, #5FH
                MOV          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          INTCLK0, #EX2 ;使能 INT2 下降沿中断
                ORL          INTCLK0, #EX3 ;使能 INT3 下降沿中断
                ORL          INTCLK0, #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

```

6.8.8 使用 T0/T1/T2/T3/T4 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 "stc16.h" //头文件见附录

#include "intrins.h"

#define ET2 0x04

#define ET3 0x20

#define ET4 0x40

#define T2IF 0x01

#define T3IF 0x02

#define T4IF 0x04

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

 TMOD = 0x00;

```
TL0 = 0x66; //65536-11.0592M/12/1000
TH0 = 0xfc;
TR0 = 1; //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)
{
    P1I = ~P1I;
}
}
```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC)

;头文件见附录

```
ET2 EQU 04H
ET3 EQU 20H
ET4 EQU 40H
```

```
T2IF EQU 01H
T3IF EQU 02H
T4IF EQU 04H
```

```
ORG 0000H
LJMP MAIN
ORG 000BH
LJMP TM0ISR
ORG 001BH
```

```

    LJMP    TM1ISR
    ORG     0063H
    LJMP    TM2ISR
    ORG     009BH
    LJMP    TM3ISR
    ORG     00A3H
    LJMP    TM4ISR

    ORG     0100H
TM0ISR:
    CPL     P1.0           ;测试端口
    RETI
TM1ISR:
    CPL     P1.0           ;测试端口
    RETI
TM2ISR:
    CPL     P1.0           ;测试端口
    RETI
TM3ISR:
    CPL     P1.0           ;测试端口
    RETI
TM4ISR:
    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     TMOD, #00H
    MOV     TL0, #66H      ;65536-11.0592M/12/1000
    MOV     TH0, #0FCH
    SETB    TR0           ;启动定时器
    SETB    ET0           ;使能定时器中断

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

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

    MOV     T3L, #66H      ;65536-11.0592M/12/1000
    MOV     T3H, #0FCH
    MOV     T4T3M, #08H    ;启动定时器

```

```

        ORL        IE2,#ET3                ;使能定时器中断

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

        SETB       EA

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

        NOP
        NOP
        NOP
LOOP:
        CPL        P1.1
        JMP        LOOP

        END

```

6.8.9 使用 RxD/RxD2/RxD3/RxD4 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 "stc16.h" //头文件见附录

#include "intrins.h"

#define ES2 0x01

#define ES3 0x08

#define ES4 0x10

```

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 = 1; //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 = 1;

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

while (1)
{
    P1I = ~P1I;
}
}

```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16F.INC)

;头文件见附录

```

ES2      EQU      01H
ES3      EQU      08H
ES4      EQU      10H

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

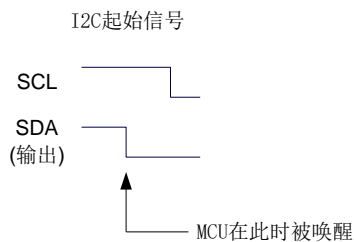
    MOV          PCON, #02H        ;MCU 进入掉电模式
    NOP
    NOP
    NOP
    NOP

LOOP:
    CPL          P1.1
    JMP          LOOP

    END

```

6.8.10 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 "stc16.h"

//头文件见附录

#include "intrins.h"

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;
//    P_SW2 = 0x10;
//    P_SW2 = 0x30;
    P_SW2 /= 0x80;
    I2CCFG = 0x80;
    I2CSLCR = 0x40;
    EA = 1;
```

//SDA/P1.4 下降沿唤醒

//SDA_2/P2.4 下降沿唤醒

//SDA_4/P3.3 下降沿唤醒

//使能 I2C 模块的从机模式

//使能起始信号中断

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

//MCU 进入掉电模式

//掉电唤醒后不会进入中断服务程序

```
    while (1)
    {
        P1I = ~P1I;
    }
}
```


}

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC)

;头文件见附录

ORG 0000H

LJMP MAIN

ORG 00C3H

LJMP I2CISR

I2CISR: ORG 0100H

PUSH ACC

PUSH DPH

PUSH DPL

ORL PSW2,#80H

MOV WR6,#WORD0 I2CSLST

MOV WR4,#WORD2 I2CSLST

MOV R11,@DR4

ANL A,#NOT 40H

MOV @DR4,R11

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 A,#80H

MOV WR6,#WORD0 I2CCFG

MOV WR4,#WORD2 I2CCFG

MOV @DR4,R11

;使能 I2C 模块的从机模式

MOV A,#40H

MOV WR6,#WORD0 I2CSLCR

MOV WR4,#WORD2 I2CSLCR

MOV @DR4,R11

;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

```

6.8.11 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 "stc16.h"           //头文件见附录
#include "intrins.h"

```

```
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;
```

//MCU 进入掉电模式

```
        _nop_();
```

```
        _nop_();
```

```
        _nop_();
```

```
        _nop_();
```

```
        P11 = ~P11;
```

```
    }
```

```
}
```

汇编代码

;测试工作频率为11.0592MHz

```

$include (STC16.INC)           ;头文件见附录

```

```
    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      ; 设定掉电唤醒时钟约为 1 秒钟
        MOV      WKTCH, #87H

LOOP:

        NOP
        NOP
        MOV      PCON, #02H      ; MCU 进入掉电模式
        NOP
        NOP
        NOP
        CPL      P1.1
        JMP      LOOP

        END

```

6.8.12 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）。

C 语言代码

```
//测试工作频率为11.0592MHz
```

```
#include "stc16.h"      //头文件见附录
#include "intrins.h"
```

```
#define ENLVR      0x40      //RSTCFG.6
#define LVD2V0     0x00      //LVD@2.0V
#define LVD2V4     0x01      //LVD@2.4V
#define LVD2V7     0x02      //LVD@2.7V
```

```

#define    LVD3V0          0x03          //LVD@3.0V

#define    LVDF            0x20          //PCON.5

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

\$include (STC16.INC)

;头文件见附录

```

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

LVDF        EQU      20H      ;PCON.5

            ORG      0000H
            LJMP     MAIN
            ORG      0033H

```

```

        LJMP      LVDISR

LVDISR:
        ORG      0100H

        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                      ;掉电唤醒后立即进入中断服务程序
        NOP
        NOP
        NOP

LOOP:
        CPL      P1.1
        JMP      LOOP

        END

```

6.8.13 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）。

C 语言代码

```
//测试工作频率为11.0592MHz
```

```
#include "stc16.h"          //头文件见附录
#include "intrins.h"
```

```

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;           //P3.7 为 CMP+ 输入脚
    CMPCR1 &= ~0x04;           //内部参考电压为 CMP- 输入脚
    // CMPCR1 /= 0x04;          //P3.6 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

\$include (STC16F.INC)

;头文件见附录

```

                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     ;P3.7 为 CMP+ 输入脚
ANL      CMPCR1, #NOT 04H     ;内部参考电压为 CMP- 输入脚
; ORL      CMPCR1, #04H       ;P3.6 为 CMP- 输入脚
ORL      CMPCR1, #02H         ;使能比较器输出
SETB     EA

MOV     PCON, #02H           ;MCU 进入掉电模式
NOP
NOP
NOP
NOP

```

LOOP:

```

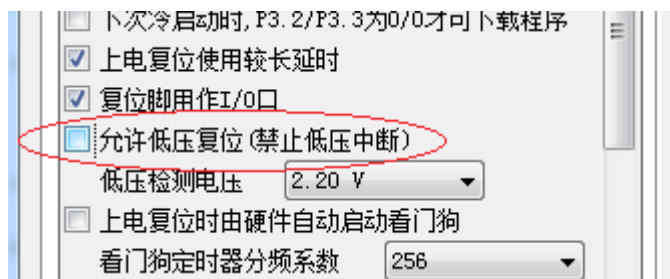
CPL     P1.1
JMP     LOOP

```

END

6.8.14 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



C 语言代码

```
//测试工作频率为11.0592MHz
```

```
#include "stc16.h"
#include "intrins.h"
```

```
//头文件见附录
```

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

#define LVD2V0        0x00          //LVD@2.0V
#define LVD2V4        0x01          //LVD@2.4V
#define LVD2V7        0x02          //LVD@2.7V
#define LVD3V0        0x03          //LVD@3.0V

#define LVDF          0x20          //PCON.5

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 = LVD2V0;
            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

\$include (STC16F.INC)

;头文件见附录

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
------	-----	-----	---------

```

    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,#LVD2V0
        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
        JMP        LOOP

```

;P2.3~P2.0 显示电池电量

DELAY:

MOV R0,#100

NEXT:

NOP

NOP

NOP

NOP

DJNZ R0,NEXT

RET

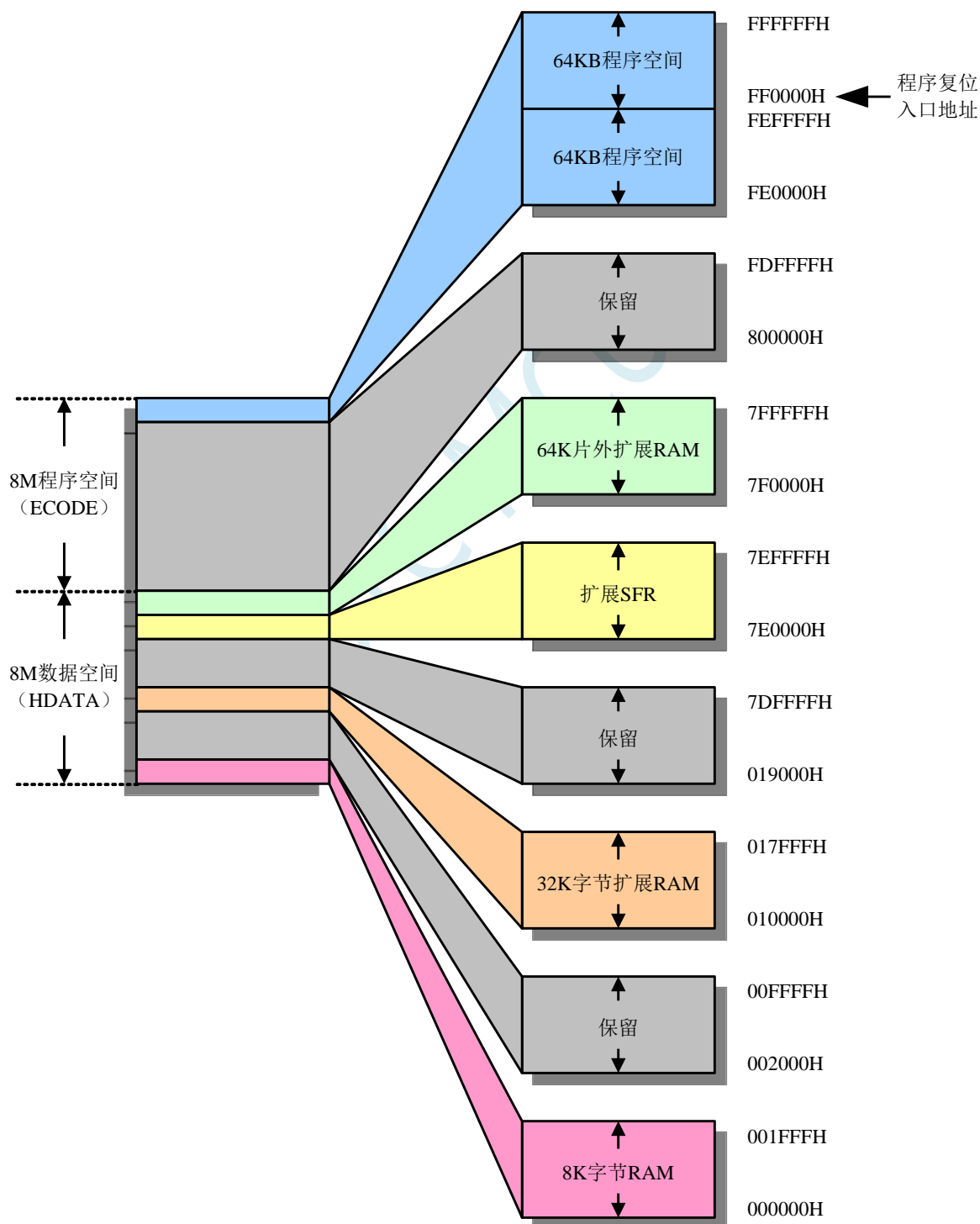
END

STC MCU

7 存储器

STC16F 系列单片机的程序存储器和数据存储器是统一编址的。STC16F 系列单片机提供 24 位寻址空间，最多能够访问 16M 的存储器。由于没有提供访问外部程序存储器的总线，所有单片机的所有程序存储器都是片上 Flash 存储器，不能访问外部程序存储器。

STC16F 系列单片机内部集成了大容量的数据存储器。STC16F 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 和内部扩展 RAM。

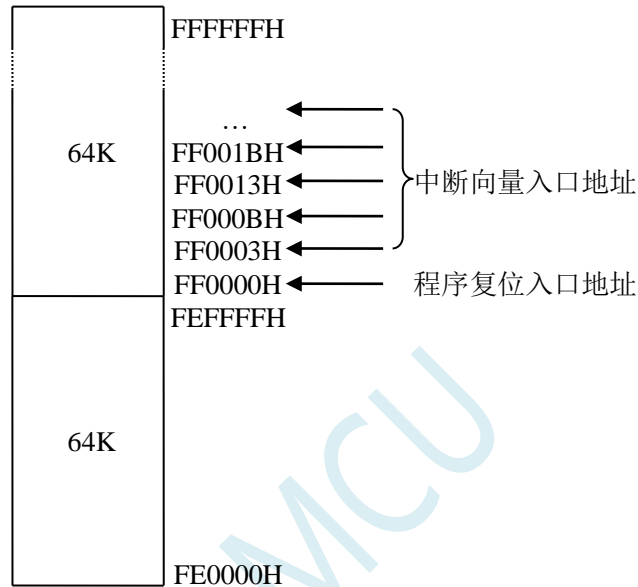


7.1 程序存储器

程序存储器用于存放用户程序、数据以及表格等信息。

STC16F40K128 系列单片内部集成了 128K 字节的 Flash 程序存储器 (ROM)。地址范围位于 16M 寻址空间中的 FE0000H~FFFFFFH。

第一次送样芯片可使用的程序存储器空间只有 120K，前 60K (FE0000H~FEEFFFH)，后 60K (FF0000H~FFFFFFH)。



单片机复位后，程序计数器(PC)的内容为 FF0000H，从 FF0000H 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 FF0003H，定时器/计数器 0 (TIMER0) 中断服务程序的入口地址是 FF000BH，外部中断 1 (INT1) 的中断服务程序的入口地址是 FF0013H，定时器/计数器 1 (TIMER1) 的中断服务程序的入口地址是 FF001BH 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。

~~STC16F 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。(暂不支持)~~

7.2 数据存储

STC16F 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。

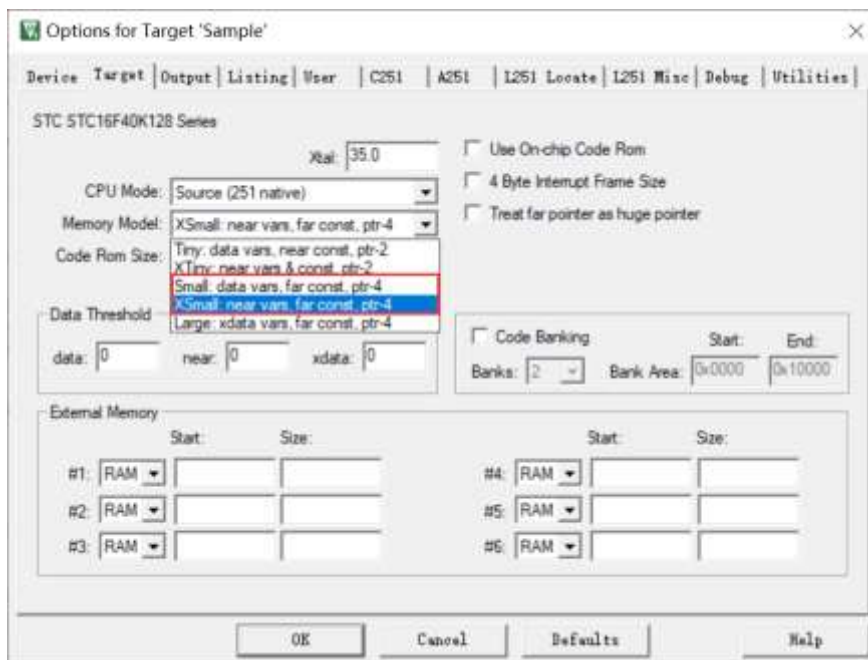
单片机系列	内部 RAM (EDATA)	内部扩展 RAM (XDATA)
STC16F40K128 系列	8K 字节	32K 字节

EDATA: 单时钟周期存取, 访问速度快, 可做堆栈使用, 成本高;

XDATA: 存取需要 3 个时钟周期, 访问速度慢, 寻址空间可达 8M, 成本低。

Keil 选项 Memory Model 设置:

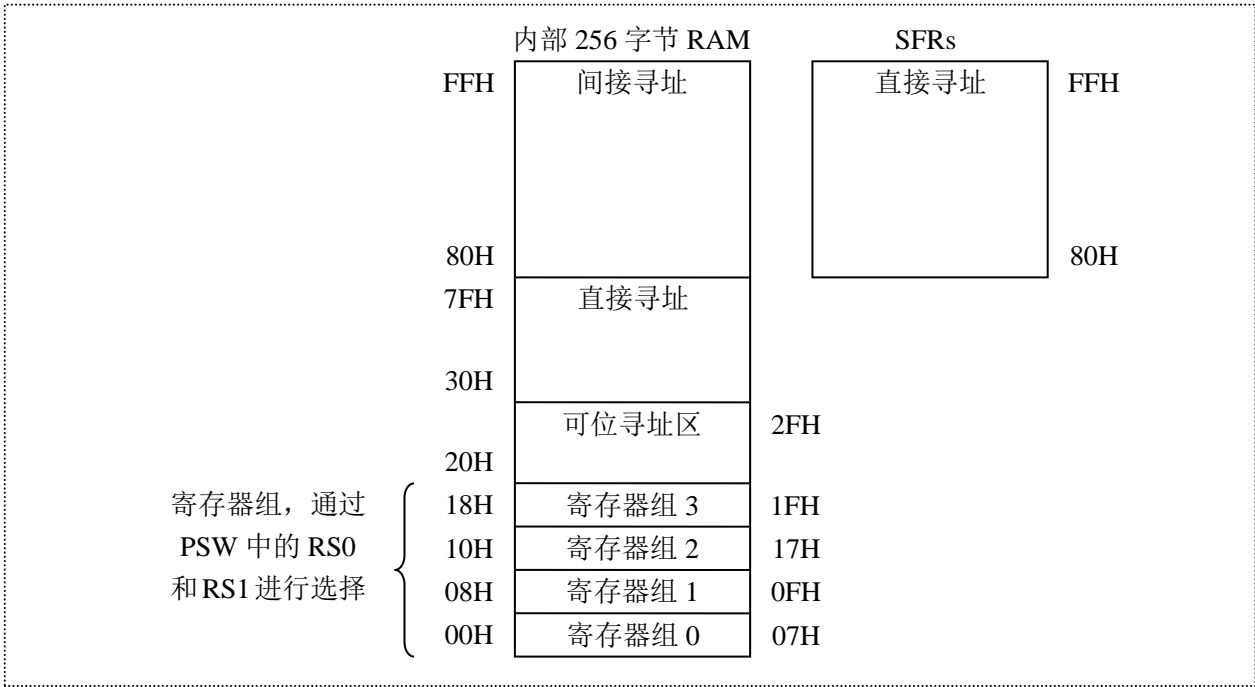
推荐设置为“Small”或者“XSmall”模式, 这两种模式默认将变量定义在内部 RAM(edata), 单时钟周期存取, 访问速度快; 不推荐使用“Large”模式, 该模式默认将变量定义在内部扩展 RAM(xdata)里面, 存取需要 3 个时钟周期, 访问速度慢:



7.2.1 内部 RAM

内部 RAM 共 8K 字节, 8K 字节低端的 256 字节与 8051 的 256 字节 DATA 完全兼容, 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。

低端 256 字节 RAM 的结构如下图所示:



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	RS1	RS0	OV	-	P

RS1, RS0: 工作寄存器选择位

RS1	RS0	工作寄存器组 (R0~R7)
0	0	第 0 组 (00H~07H)
0	1	第 1 组 (08H~0FH)
1	0	第 2 组 (10H~17H)
1	1	第 3 组 (18H~1FH)

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

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

堆栈指针是一个 8 位专用寄存器。它指示出堆栈顶部在内部 RAM 块中的位置。系统复位后，SP 初始化为 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。STC16F 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，且堆栈可提高到 32K。

7.2.3 内部扩展 RAM

STC16F 系列单片机片内除了集成 32K 字节的扩展 RAM。访问内部扩展 RAM 的方法和传统 8051 单片机访问外部扩展 RAM 的方法相同，但是不影响 P0 口(数据总线和高八位地址总线)、P2 口(低八位地址总线)、以及 RD、WR 和 ALE 等端口上的信号。

在汇编语言中，内部扩展 RAM 通过 MOVX 指令访问，

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

在 C 语言中，可使用 xdata/pdata 声明存储类型即可。如：

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

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

使用 MOVX A,@Ri 和 MOVX @Ri,A 时，XRAM[23:16]地址由 MXAX 设置，XRAM[15:8]由 P2 口设置，XRAM[7:0]由 Ri 设置，由于 STC16 的 XRAM 地址范围为 0x10000~0x17FFF，所以 STC16 使用 pdata 定义变量时，需要设置 MXAX = 0x01（上电默认值），P2 = 0x00~0x7F。

xdata 使用注意：

定义变量时可将单字节变量定义在 xdata 里面，多字节（2 字节、4 字节）变量需要定义在 edata 里面。后续版本会取消这种使用限制。

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	93H	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

EXTRAM：扩展 RAM 访问控制

0：访问内部扩展 RAM。

1：内部扩展 RAM 被禁用。

7.2.5 外部扩展 RAM，XRAM，XDATA

STC16 系列单片机具有扩展 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。STC16 系列单片机控制外部 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	BFH	RW_S[1:0]					SPEED[2:0]		

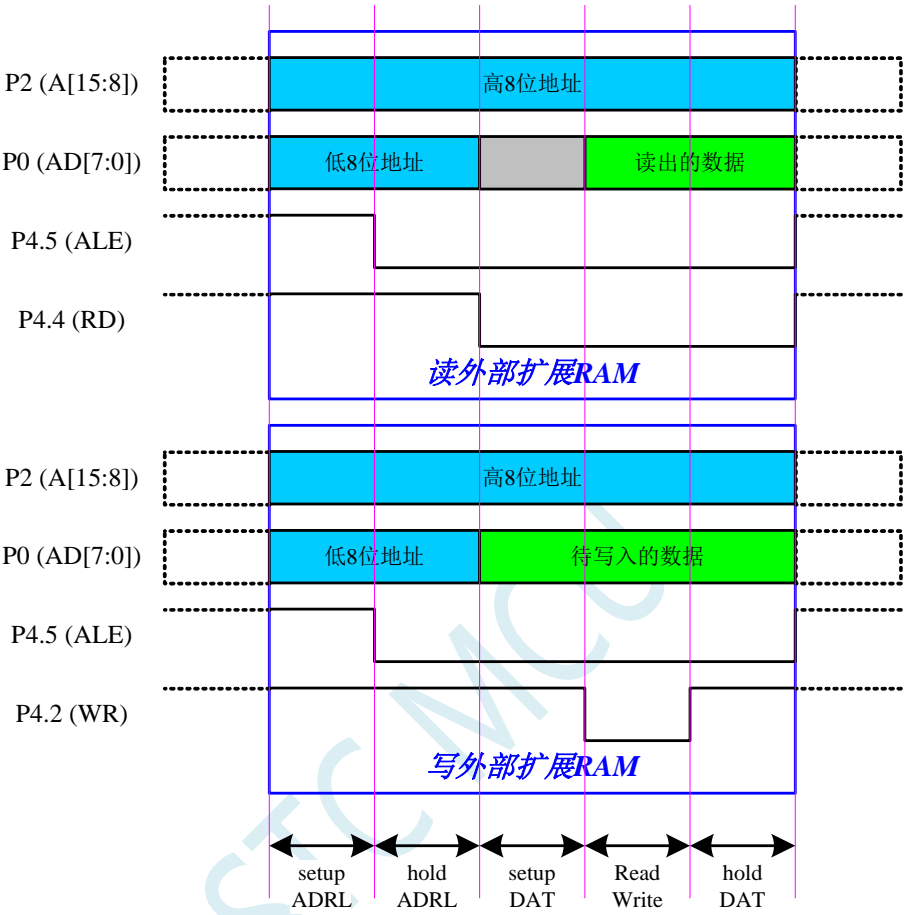
RW_S[1:0]：RD/WR 控制线选择位

00：P4.4 为 RD，P4.2 为 WR

x1: 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）

读写外部扩展 RAM 时序如下图所示:



8 特殊功能寄存器

8.1 STC16F40K128 系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H	P7	P3M1	P3M0	P4M1	P4M0	WDTCR	IP2	RSTCFG
F0H	B	P0M1	P0M0	P1M1	P1M0	P2M1	P2M0	IAPTPS
E8H	P6		MXAX	TA	P5M1	P5M0	P6M1	P6M0
E0H	ACC	SADDR	SADEN	S2CON	S2BUF	S3CON	S3BUF	IE2
D8H			AUXINTIF	T4T3M	ADC_CONTR	ADC_RES	ADC_RESL	ADCCFG
D0H	PSW	PSW1	P7M1	P7M0				
C8H	P5	T4H	T4L	T3H	T3L	T2H	T2L	
C0H	P4		IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SPSTAT	SPCTL	SPDAT	S4CON	S4BUF	SPH	
B0H	P3	LINICR	LINAR	LINDR	CMPCR1	CMPCR2	IP2H	IPH
A8H	IE	IRCBAND	WKTCL	WKTCH				
A0H	P2	CANICR	CANAR	CANDR				
98H	SCON	SBUF	P_SW1	P_SW2			LIRTRIM	IRTRIM
90H	P1	USBCON	WTST	AUXR	AUXR2	INTCLKO	USBADR	USBDAT
88H	TCON	TMOD	TL0	TL1	TH0	TH1	CKCON	DMAIR
80H	P0	SP	DPL	DPH	DPXL	USBCLK	DPS	PCON

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
7EFCF0H	MD3	MD2	MD1	MD0	MD5	MD4	ARCON	OPCON
7EFEF8H	PWMB_CCR2L	PWMB_CCR3H	PWMB_CCR3L	PWMB_CCR4H	PWMB_CCR4L	PWMB_BKR	PWMB_DTR	PWMB_OISR
7EFEF0H	PWMB_PSCRH	PWMB_PSCRL	PWMB_ARRH	PWMB_ARRL	PWMB_RCR	PWMB_CCR1H	PWMB_CCR1L	PWMB_CCR2H
7EFE8H	PWMB_CCMR1	PWMB_CCMR2	PWMB_CCMR3	PWMB_CCMR4	PWMB_CCER1	PWMB_CCER2	PWMB_CNTRH	PWMB_CNTRL
7EFE0H	PWMB_CR1	PWMB_CR2	PWMB_SMC	PWMB_ETR	PWMB_IER	PWMB_SR1	PWMB_SR2	PWMB_EGR
7EFED8H	PWMA_CCR2L	PWMA_CCR3H	PWMA_CCR3L	PWMA_CCR4H	PWMA_CCR4L	PWMA_BKR	PWMA_DTR	PWMA_OISR
7EFED0H	PWMA_PSCRH	PWMA_PSCRL	PWMA_ARRH	PWMA_ARRL	PWMA_RCR	PWMA_CCR1H	PWMA_CCR1L	PWMA_CCR2H
7EFEC8H	PWMA_CCMR1	PWMA_CCMR2	PWMA_CCMR3	PWMA_CCMR4	PWMA_CCER1	PWMA_CCER2	PWMA_CNTRH	PWMA_CNTRL
7EFEC0H	PWMA_CR1	PWMA_CR2	PWMA_SMC	PWMA_ETR	PWMA_IER	PWMA_SR1	PWMA_SR2	PWMA_EGR
7EFEB0H	PWMA_ETRPS	PWMA_ENO	PWMA_PS	PWMA_IOAUX	PWMB_ETRPS	PWMB_ENO	PWMB_PS	PWMB_IOAUX
7EFA8H				ADCTIM	T3T4PIN			
7EFA0H			TM2PS	TM3PS	TM4PS			
7EFE88H	I2CMSAUX							
7EFE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
7EFE30H	P0IE	P1IE						
7EFE28H	P0DR	P1DR	P2DR	P3DR	P4DR	P5DR	P6DR	P7DR
7EFE20H	P0SR	P1SR	P2SR	P3SR	P4SR	P5SR	P6SR	P7SR
7EFE18H	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS	P6NCS	P7NCS
7EFE10H	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU	P6PU	P7PU
7EFE08H	IRC48MCR	IRC48ATRIM	ITR48BTRIM	IRCDB				
7EFE00H	CKSEL	CLKDIV	IRC24MCR	XOCCR	IRC32KCR	PLLCR	USBCON1	MCLKOCR

8.2 特殊功能寄存器列表

符号	描述	地址	位地址与符号								复位值		
			B7	B6	B5	B4	B3	B2	B1	B0			
P0	P0 端口	80H									1111,1111		
SP	堆栈指针	81H									0000,0111		
DPL	数据指针（低字节）	82H									0000,0000		
DPH	数据指针（高字节）	83H									0000,0000		
DPXL	数据指针（最高字节）	84H									0000,0000		
USBCLK	USB 时钟控制寄存器	85H	ENCKM	PCKI[1:0]		CRE	TST_USB	TST_PHY	PHYTST[1:0]		0010,0000		
DPS	DPTR 指针选择器	86H	ID1	ID0	TSL	AU1	AU0	-	-	SEL	0000,0xx0		
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000		
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000		
TMOD	定时器模式寄存器	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000,0000		
TL0	定时器 0 低 8 位寄存器	8AH									0000,0000		
TL1	定时器 1 低 8 位寄存器	8BH									0000,0000		
TH0	定时器 0 高 8 位寄存器	8CH									0000,0000		
TH1	定时器 1 高 8 位寄存器	8DH									0000,0000		
CKCON	XRAM 控制寄存器	8EH	-	-	-	T1M	T0M	CKCON[2:0]			0000,0000		
DMAIR	FMU DMA 指令寄存器	8FH									0000,0000		
P1	P1 端口	90H									1111,1111		
USBCON	USB 控制寄存器	91H	ENUSB	USBRST	PS2M	PUEN	PDEN	DFREC	DP	DM	0000,0000		
WTST	程序读取控制寄存器	92H									0000,0111		
AUXR	辅助寄存器 1	93H	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001		
AUXR2	辅助寄存器 2	94H	-	P40ADR	ADIBF	TXLNRX	-	-	CANEN	LINEN	0000,0000		
INTCLKO	中断与时钟输出控制寄存器	95H	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000		
USBADR	USB 地址寄存器	96H	BUSY	AUTORD	UADR[5:0]							0000,0000	
USBDAT	USB 数据寄存器	97H											0000,0000
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000		
SBUF	串口 1 数据寄存器	99H											0000,0000
P_SW1	外设端口切换寄存器 1	9AH	S1_S[1:0]		CAN_S[1:0]		SPI_S[1:0]		LIN_S[1:0]		nn00,000x		
P_SW2	外设端口切换寄存器 2	9BH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S	0x00,0000		
LIRTRIM	IRC 频率微调寄存器	9EH	-	-	-	-	-	-	LIRTRIM[1:0]		0000,00nn		
IRTRIM	IRC 频率调整寄存器	9FH	IRTRIM[7:0]								nnnn,nnnn		
P2	P2 端口	A0H											1111,1111
CANICR	CANBUS 中断控制寄存器	A1H	-	-	-	-	PCANH	CANIF	ECAN	PCANL	0000,0000		
CANAR	CANBUS 地址寄存器	A2H											0000,0000
CANDR	CANBUS 数据寄存器	A3H											0000,0000
IE	中断允许寄存器	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000,0000		
IRCBAND	IRC 频段选择检测	A9H	-	-	-	-	-	-	-	SEL	xxxx,xxxn		
WKTCL	掉电唤醒定时器低字节	AAH											1111,1111
WKTCH	掉电唤醒定时器高字节	ABH	WKTEN										0111,1111
P3	P3 端口	B0H											1111,1111

LINICR	LINBUS 中断控制寄存器	B1H					PLINH	LINIF	ELIN	PLINL	0000,0000
LINAR	LINBUS 地址寄存器	B2H									0000,0000
LINDR	LINBUS 数据寄存器	B3H									0000,0000
CMPCR1	比较器控制寄存器 1	B4H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
CMPCR2	比较器控制寄存器 2	B5H	INVCMP0	DISFLT	LCDTY[5:0]						0000,0000
IP2H	高中断优先级控制寄存器 2	B6H		PI2CH	PCMPH	PX4H	PPWMB	PUSBH	PSPIH	PS2H	0000,0000
IPH	高中断优先级控制寄存器	B7H	PPWMAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	x000,0000
IP	中断优先级控制寄存器	B8H	PPWMA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	x000,0000
SPSTAT	SPI 状态寄存器	B9H	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
SPCTL	SPI 控制寄存器	BAH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]		0000,0100
SPDAT	SPI 数据寄存器	BBH									0000,0000
S4CON	串口 4 控制寄存器	BCH	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
S4BUF	串口 4 数据寄存器	BDH									0000,0000
SPH	堆栈指针高字节	BEH									0000,0000
BUS_SPEED	总线速度控制寄存器	BFH	RW_S[1:0]		-	-	-	SPEED[2:0]			00xx,x000
P4	P4 端口	C0H									1111,1111
IAP_DATA	IAP 数据寄存器	C2H									1111,1111
IAP_ADDRH	IAP 高地址寄存器	C3H									0000,0000
IAP_ADDRL	IAP 低地址寄存器	C4H									0000,0000
IAP_CMD	IAP 命令寄存器	C5H	-	-	-	-	-	-	CMD[1:0]		xxxx,xx00
IAP_TRIG	IAP 触发寄存器	C6H									0000,0000
IAP_CONTR	IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-	0000,xxxx
P5	P5 端口	C8H	-	-	-						xxx1,1111
T4H	定时器 4 高字节	C9H									0000,0000
T4L	定时器 4 低字节	CAH									0000,0000
T3H	定时器 3 高字节	CBH									0000,0000
T3L	定时器 3 低字节	CCH									0000,0000
T2H	定时器 2 高字节	CDH									0000,0000
T2L	定时器 2 低字节	CEH									0000,0000
PSW	程序状态字寄存器	D0H	CY	AC	F0	RS1	RS0	OV	-	P	0000,00x0
PSW1	程序状态字 1 寄存器	D1H	CY	AC	N	RS1	RS0	OV	Z		0000,000x
P7M1	P7 口配置寄存器 1	D2H									1111,1111
P7M0	P7 口配置寄存器 0	D3H									0000,0000
AUXINTIF	扩展外部中断标志寄存器	DAH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF	x000,x000
T4T3M	定时器 4/3 控制寄存器	DBH	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000,0000
ADC_CONTR	ADC 控制寄存器	DCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]				000x,0000
ADC_RES	ADC 转换结果高位寄存器	DDH									0000,0000
ADC_RES1	ADC 转换结果低位寄存器	DEH									0000,0000
ADCCFG	ADC 配置寄存器	DFH	-	-	RESFMT	-	SPEED[3:0]				xx0x,0000
ACC	累加器	E0H									0000,0000
SADDR	串口 1 从机地址寄存器	E1H									0000,0000
SADEN	串口 1 从机地址屏蔽寄存器	E2H									0000,0000
S2CON	串口 2 控制寄存器	E3H	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
S2BUF	串口 2 数据寄存器	E4H									0000,0000

S3CON	串口 3 控制寄存器	E5H	S3SM0	S3ST4	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S3BUF	串口 3 数据寄存器	E6H									0000,0000
IE2	中断允许寄存器 2	E7H	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000,0000
P6	P6 端口	E8H									1111,1111
MXAX	MOVX 扩展地址寄存器	EAH									0000,0001
TA	DPTR 时序控制寄存器	EBH									0000,0000
P5M1	P5 口配置寄存器 1	ECH	-	-	-						xxx1,1111
P5M0	P5 口配置寄存器 0	EDH	-	-	-						xxx0,0000
P6M1	P6 口配置寄存器 1	EEH									1111,1111
P6M0	P6 口配置寄存器 0	EFH									0000,0000
B	B 寄存器	F0H									0000,0000
P0M1	P0 口配置寄存器 1	F1H									0000,0000
P0M0	P0 口配置寄存器 0	F2H									0000,0000
P1M1	P1 口配置寄存器 1	F3H									0000,0000
P1M0	P1 口配置寄存器 0	F4H									0000,0000
P2M1	P2 口配置寄存器 1	F5H									0000,0000
P2M0	P2 口配置寄存器 0	F6H									0000,0000
IAP_TPS	IAP 等待时间控制寄存器	F7H	-	-	IAPTPS[5:0]					xx00,0000	
P7	P7 端口	F8H									1111,1111
P3M1	P3 口配置寄存器 1	F9H									n000,0000
P3M0	P3 口配置寄存器 0	FAH									n000,0000
P4M1	P4 口配置寄存器 1	FBH									0000,0000
P4M0	P4 口配置寄存器 0	FCH									0000,0000
WDT_CONTR	看门狗控制寄存器	FDH	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]			0x00,0000
IP2	中断优先级控制寄存器 2	FEH	-	PI2C	PCMP	PX4	PPWMB	PPWMA	PSPI	PS2	0000,0000
RSTCFG	复位配置寄存器	FFH	-	ENLVR	-	P54RST	-	-	LVDS[1:0]		0000,0000

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

MOV A,#00H

MOV WR6,#WORD0 CKSEL ; CKSEL 可换为需要访问的寄存器

MOV WR4,#WORD2 CKSEL

MOV @DR4,R11

和

MOV WR6,#WORD0 CKSEL ; CKSEL 可换为需要访问的寄存器

MOV WR4,#WORD2 CKSEL

MOV R11,@DR4

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CKSEL	时钟选择寄存器	7EFE00H	-	-	-	-	-	-	MCKSEL[1:0]		xxxx,xx00

CLKDIV	时钟分频寄存器	7EFE01H									0000,0100
IRC24MCR	内部 24M 振荡器控制寄存器	7EFE02H	ENIRC24M	-	-	-	-	-	-	IRC24MST	1xxx,xxx0
XOSCCR	外部晶振控制寄存器	7EFE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST	00xx,xxx0
IRC32KCR	内部 32K 振荡器控制寄存器	7EFE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0
PLLCR	PLL 控制寄存器	7EFE05H	ENPLL	PLLSSEL[1:0]		PLLINCTL[1:0]		PLLTOMSEL[1:0]		PLLST	0000,0000
USBCON1	USB 控制寄存器 1	7EFE06H	ENLDO33	TRIMPLL[1:0]		TRIMLDO33[3:0]				-	0000,0000
MCLKOCR	主时钟输出控制寄存器	7EFE07H	MCKLO_S	MCLKODIV[6:0]							0000,0000
IRC48MCR	内部 48M 振荡器控制寄存器	7EFE08H	ENIRC48M	-	-	-	-	-	-	IRC48MST	1xxx,xxx0
IRC48ATRIM	48M 振荡器 TRIM 寄存器 A	7EFE09H				ATRIM[4:0]					0000,0000
IRC48BTRIM	48M 振荡器 TRIM 寄存器 B	7EFE0AH									0000,0000
IRCDDB	内部高速振荡器去抖控制	7EFE0BH									1000,0000
P0PU	P0 口上拉电阻控制寄存器	7EFE10H									0000,0000
P1PU	P1 口上拉电阻控制寄存器	7EFE11H									0000,0000
P2PU	P2 口上拉电阻控制寄存器	7EFE12H									0000,0000
P3PU	P3 口上拉电阻控制寄存器	7EFE13H									0000,0000
P4PU	P4 口上拉电阻控制寄存器	7EFE14H									0000,0000
P5PU	P5 口上拉电阻控制寄存器	7EFE15H	-	-	-						xxx0,0000
P6PU	P6 口上拉电阻控制寄存器	7EFE16H									0000,0000
P7PU	P7 口上拉电阻控制寄存器	7EFE17H									0000,0000
P0NCS	P0 口施密特触发控制寄存器	7EFE18H									0000,0000
P1NCS	P1 口施密特触发控制寄存器	7EFE19H									0000,0000
P2NCS	P2 口施密特触发控制寄存器	7EFE1AH									0000,0000
P3NCS	P3 口施密特触发控制寄存器	7EFE1BH									0000,0000
P4NCS	P4 口施密特触发控制寄存器	7EFE1CH									0000,0000
P5NCS	P5 口施密特触发控制寄存器	7EFE1DH	-	-	-						xxx0,0000
P6NCS	P6 口施密特触发控制寄存器	7EFE1EH									0000,0000
P7NCS	P7 口施密特触发控制寄存器	7EFE1FH									0000,0000
P0SR	P0 口电平转换速率寄存器	7EFE20H									1111,1111
P1SR	P1 口电平转换速率寄存器	7EFE21H									1111,1111
P2SR	P2 口电平转换速率寄存器	7EFE22H									1111,1111
P3SR	P3 口电平转换速率寄存器	7EFE23H									1111,1111
P4SR	P4 口电平转换速率寄存器	7EFE24H									1111,1111
P5SR	P5 口电平转换速率寄存器	7EFE25H	-	-	-						xxx1,1111
P6SR	P6 口电平转换速率寄存器	7EFE26H									1111,1111
P7SR	P7 口电平转换速率寄存器	7EFE27H									1111,1111
P0DR	P0 口驱动电流控制寄存器	7EFE28H									1111,1111
P1DR	P1 口驱动电流控制寄存器	7EFE29H									1111,1111
P2DR	P2 口驱动电流控制寄存器	7EFE2AH									1111,1111
P3DR	P3 口驱动电流控制寄存器	7EFE2BH									1111,1111
P4DR	P4 口驱动电流控制寄存器	7EFE2CH									1111,1111
P5DR	P5 口驱动电流控制寄存器	7EFE2DH	-	-	-						xxx1,1111
P6DR	P6 口驱动电流控制寄存器	7EFE2EH									1111,1111
P7DR	P7 口驱动电流控制寄存器	7EFE2FH									1111,1111
P0IE	P0 口输入使能控制寄存器	7EFE30H									1111,1111

P1IE	P1 口输入使能控制寄存器	7EFE31H									1111,1111
P3IE	P3 口输入使能控制寄存器	7EFE33H									1111,1111
I2CCFG	I ² C 配置寄存器	7EFE80H	ENI2C	MSSL	MSSPEED[6:1]					0000,0000	
I2CMSCR	I ² C 主机控制寄存器	7EFE81H	EMSI	-	-	-	MSCMD[3:0]			0xxx,0000	
I2CMSST	I ² C 主机状态寄存器	7EFE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I ² C 从机控制寄存器	7EFE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I ² C 从机状态寄存器	7EFE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I ² C 从机地址寄存器	7EFE85H	SLADR[6:0]							MA	0000,0000
I2CTXD	I ² C 数据发送寄存器	7EFE86H									0000,0000
I2CRXD	I ² C 数据接收寄存器	7EFE87H									0000,0000
I2CMSAUX	I ² C 主机辅助控制寄存器	7EFE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0
TM2PS	定时器 2 时钟预分频寄存器	7EFEA2H									0000,0000
TM3PS	定时器 3 时钟预分频寄存器	7EFEA3H									0000,0000
TM4PS	定时器 4 时钟预分频寄存器	7EFEA4H									0000,0000
ADCTIM	ADC 时序控制寄存器	7EFEABH	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]					0010,1010
T3T4PIN	T3/T4 选择寄存器	7EFEACH	-	-	-	-	-	-	-	T3T4SEL	xxxx,xxx0
PWMA_ETRPS	PWMA 的 ETR 选择寄存器	7EFEB0H						BRKAPS	ETRAPPS[1:0]		xxxx,x000
PWMA_ENO	PWMA 输出使能控制	7EFEB1H	ENO4N	ENO4P	ENO3N	ENO3P	ENO2N	ENO2P	ENO1N	ENO1P	0000,0000
PWMA_PS	PWMA 输出脚选择寄存器	7EFEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]		0000,0000
PWMA_IOAUX	PWMA 辅助寄存器	7EFEB3H	AUX4N	AUX4P	AUX3N	AUX3P	AUX2N	AUX2P	AUX1N	AUX1P	0000,0000
PWMB_ETRPS	PWMB 的 ETR 选择寄存器	7EFEB4H						BRKBPS	ETRBPS[1:0]		xxxx,x000
PWMB_ENO	PWMB 输出使能控制	7EFEB5H	-	ENO8P	-	ENO7P	-	ENO6P	-	ENO5P	x0x0,x0x0
PWMB_PS	PWMB 输出脚选择寄存器	7EFEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]		0000,0000
PWMB_IOAUX	PWMB 辅助寄存器	7EFEB7H	-	AUX8P	-	AUX7P	-	AUX6P	-	AUX5P	x0x0,x0x0
PWMA_CR1	PWMA 控制寄存器 1	7EFEC0H	ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN	0000,0000
PWMA_CR2	PWMA 控制寄存器 2	7EFEC1H	-	MMS[2:0]			-	COMS	-	CCPC	x000,x0x0
PWMA_SMC	PWMA 从模式控制寄存器	7EFEC2H	MSM	TS[2:0]			-	SMS[2:0]			0000,x000
PWMA_ETR	PWMA 外部触发寄存器	7EFEC3H	ETP	ECE	ETPS[1:0]		ETF[3:0]				0000,0000
PWMA_IER	PWMA 中断使能寄存器	7EFEC4H	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE	0000,0000
PWMA_SR1	PWMA 状态寄存器 1	7EFEC5H	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF	0000,0000
PWMA_SR2	PWMA 状态寄存器 2	7EFEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-	xxx0,000x
PWMA_EGR	PWMA 事件发生寄存器	7EFEC7H	BG	TG	COMG	CC4G	CC3G	CC2G	CC1G	UG	0000,0000
PWMA_CCMR1	PWMA 捕获模式寄存器 1	7EFEC8H	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]		0000,0000
	PWMA 比较模式寄存器 1		IC1F[3:0]			IC1PSC[1:0]		CC1S[1:0]		0000,0000	
PWMA_CCMR2	PWMA 捕获模式寄存器 2	7EFEC9H	OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		0000,0000
	PWMA 比较模式寄存器 2		IC2F[3:0]			IC2PSC[1:0]		CC2S[1:0]		0000,0000	
PWMA_CCMR3	PWMA 捕获模式寄存器 3	7EFECAH	OC3CE	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]		0000,0000
	PWMA 比较模式寄存器 3		IC3F[3:0]			IC3PSC[1:0]		CC3S[1:0]		0000,0000	
PWMA_CCMR4	PWMA 捕获模式寄存器 4	7EFECBH	OC4CE	OC4M[2:0]			OC4PE	OC4FE	CC4S[1:0]		0000,0000
	PWMA 比较模式寄存器 4		IC4F[3:0]			IC4PSC[1:0]		CC4S[1:0]		0000,0000	
PWMA_CCER1	PWMA 捕获比较使能寄存器 1	7EFECCH	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E	0000,0000
PWMA_CCER2	PWMA 捕获比较使能寄存器 2	7EFECDH	CC4NP	CC4NE	CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E	0000,0000

PWMA_CNTRH	PWMA 计数器高字节	7EFECEH	CNT[15:8]								0000,0000
PWMA_CNTRL	PWMA 计数器低字节	7EFECFH	CNT[7:0]								0000,0000
PWMA_PSCRH	PWMA 预分频高字节	7EFED0H	PSC[15:8]								0000,0000
PWMA_PSCRL	PWMA 预分频低字节	7EFED1H	PSC[7:0]								0000,0000
PWMA_ARRH	PWMA 自动重装寄存器高字节	7EFED2H	ARR[15:8]								0000,0000
PWMA_ARRL	PWMA 自动重装寄存器低字节	7EFED3H	ARR[7:0]								0000,0000
PWMA_RCR	PWMA 重复计数器寄存器	7EFED4H	REP[7:0]								0000,0000
PWMA_CCR1H	PWMA 比较捕获寄存器 1 高位	7EFED5H	CCR1[15:8]								0000,0000
PWMA_CCR1L	PWMA 比较捕获寄存器 1 低位	7EFED6H	CCR1[7:0]								0000,0000
PWMA_CCR2H	PWMA 比较捕获寄存器 2 高位	7EFED7H	CCR2[15:8]								0000,0000
PWMA_CCR2L	PWMA 比较捕获寄存器 2 低位	7EFED8H	CCR2[7:0]								0000,0000
PWMA_CCR3H	PWMA 比较捕获寄存器 3 高位	7EFED9H	CCR3[15:8]								0000,0000
PWMA_CCR3L	PWMA 比较捕获寄存器 3 低位	7EFEDA	CCR3[7:0]								0000,0000
PWMA_CCR4H	PWMA 比较捕获寄存器 4 高位	7EFEDBH	CCR4[15:8]								0000,0000
PWMA_CCR4L	PWMA 比较捕获寄存器 4 低位	7EFEDCH	CCR4[7:0]								0000,0000
PWMA_BKR	PWMA 刹车寄存器	7EFEDDH	MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[1:0]		0000,000x
PWMA_DTR	PWMA 死区控制寄存器	7EFEDEH	DTG[7:0]								0000,0000
PWMA_OISR	PWMA 输出空闲状态寄存器	7EFEDFH	OIS4N	OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1	0000,0000
PWMB_CR1	PWMB 控制寄存器 1	7EFEE0H	ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN	0000,0000
PWMB_CR2	PWMB 控制寄存器 2	7EFEE1H	-	MMS[2:0]			-	COMS	-	CCPC	x000,x0x0
PWMB_SMC	PWMB 从模式控制寄存器	7EFEE2H	MSM	TS[2:0]			-	SMS[2:0]			0000,x000
PWMB_ETR	PWMB 外部触发寄存器	7EFEE3H	ETP	ECE	ETPS[1:0]		ETF[3:0]				0000,0000
PWMB_IER	PWMB 中断使能寄存器	7EFEE4H	BIE	TIE	COMIE	CC8IE	CC7IE	CC6IE	CC5IE	UIE	0000,0000
PWMB_SR1	PWMB 状态寄存器 1	7EFEE5H	BIF	TIF	COMIF	CC8IF	CC7IF	CC6IF	CC5IF	UIF	0000,0000
PWMB_SR2	PWMB 状态寄存器 2	7EFEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-	xxx0,000x
PWMB_EGR	PWMB 事件发生寄存器	7EFEE7H	BG	TG	COMG	CC8G	CC7G	CC6G	CC5G	UG	0000,0000
PWMB_CCMR1	PWMB 捕获模式寄存器 1	7EFEE8H	OC5CE	OC5M[2:0]			OC5PE	OC5FE	CC5S[1:0]		0000,0000
	PWMB 比较模式寄存器 1		IC5F[3:0]			IC5PSC[1:0]		CC5S[1:0]		0000,0000	
PWMB_CCMR2	PWMB 捕获模式寄存器 2	7EFEE9H	OC6CE	OC6M[2:0]			OC6PE	OC6FE	CC6S[1:0]		0000,0000
	PWMB 比较模式寄存器 2		IC6F[3:0]			IC6PSC[1:0]		CC6S[1:0]		0000,0000	
PWMB_CCMR3	PWMB 捕获模式寄存器 3	7EFEEAH	OC7CE	OC7M[2:0]			OC7PE	OC7FE	CC7S[1:0]		0000,0000
	PWMB 比较模式寄存器 3		IC7F[3:0]			IC7PSC[1:0]		CC7S[1:0]		0000,0000	
PWMB_CCMR4	PWMB 捕获模式寄存器 4	7EFEEBH	OC8CE	OC8M[2:0]			OC8PE	OC8FE	CC8S[1:0]		0000,0000
	PWMB 比较模式寄存器 4		IC8F[3:0]			IC8PSC[1:0]		CC8S[1:0]		0000,0000	
PWMB_CCER1	PWMB 捕获比较使能寄存器 1	7EFEECH	-	-	CC6P	CC6E	-	-	CC5P	CC5E	xx00,xx00
PWMB_CCER2	PWMB 捕获比较使能寄存器 2	7EFEEFH	-	-	CC8P	CC8E	-	-	CC7P	CC7E	xx00,xx00
PWMB_CNTRH	PWMB 计数器高字节	7EFEEEH	CNT[15:8]								0000,0000
PWMB_CNTRL	PWMB 计数器低字节	7EFEEFH	CNT[7:0]								0000,0000
PWMB_PSCRH	PWMB 预分频高字节	7EFEF0H	PSC[15:8]								0000,0000
PWMB_PSCRL	PWMB 预分频低字节	7EFEF1H	PSC[7:0]								0000,0000
PWMB_ARRH	PWMB 自动重装寄存器高字节	7EFEF2H	ARR[15:8]								0000,0000
PWMB_ARRL	PWMB 自动重装寄存器低字节	7EFEF3H	ARR[7:0]								0000,0000

PWMB_RCR	PWMB 重复计数器寄存器	7EFEF4H	REP[7:0]								0000,0000
PWMB_CCR1H	PWMB 比较捕获寄存器 1 高位	7EFEF5H	CCR1[15:8]								0000,0000
PWMB_CCR1L	PWMB 比较捕获寄存器 1 低位	7EFEF6H	CCR1[7:0]								0000,0000
PWMB_CCR2H	PWMB 比较捕获寄存器 2 高位	7EFEF7H	CCR2[15:8]								0000,0000
PWMB_CCR2L	PWMB 比较捕获寄存器 2 低位	7EFEF8H	CCR2[7:0]								0000,0000
PWMB_CCR3H	PWMB 比较捕获寄存器 3 高位	7EFEF9H	CCR3[15:8]								0000,0000
PWMB_CCR3L	PWMB 比较捕获寄存器 3 低位	7EFEFAH	CCR3[7:0]								0000,0000
PWMB_CCR4H	PWMB 比较捕获寄存器 4 高位	7EFEFBH	CCR4[15:8]								0000,0000
PWMB_CCR4L	PWMB 比较捕获寄存器 4 低位	7EFEFCH	CCR4[7:0]								0000,0000
PWMB_BKR	PWMB 刹车寄存器	7EFEFDH	MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[1:0]	-	0000,000x
PWMB_DTR	PWMB 死区控制寄存器	7EFEFEH	DTG[7:0]								0000,0000
PWMB_OISR	PWMB 输出空闲状态寄存器	7EFEFFH	-	OIS8	-	OIS7	-	OIS6	-	OIS5	x0x0,x0x0
MD3	MDU 数据寄存器	7EFCF0H	MD3[7:0]								0000,0000
MD2	MDU 数据寄存器	7EFCF1H	MD2[7:0]								0000,0000
MD1	MDU 数据寄存器	7EFCF2H	MD1[7:0]								0000,0000
MD0	MDU 数据寄存器	7EFCF3H	MD0[7:0]								0000,0000
MD5	MDU 数据寄存器	7EFCF4H	MD5[7:0]								0000,0000
MD4	MDU 数据寄存器	7EFCF5H	MD4[7:0]								0000,0000
ARCON	MDU 模式控制寄存器	7EFCF6H	MODE[2:0]				SC[4:0]				0000,0000
OPCON	MDU 操作控制寄存器	7EFCF7H	-	MDOV	-	-	-	-	RST	ENOP	0000,0000

9 I/O 口

STC16 系列单片机所有的 I/O 口均有 4 种工作模式：准双向口/弱上拉（标准 8051 输出口模式）、推挽输出/强上拉、高阻输入（电流既不能流入也不能流出）、开漏输出。可使用软件对 I/O 口的工作模式进行容易配置。

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

9.1 I/O 口相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P0	P0 端口	80H	P07	P06	P05	P04	P03	P02	P01	P00	1111,1111
P1	P1 端口	90H	P17	P16	P15	P14	P13	P12	P11	P10	1111,1111
P2	P2 端口	A0H	P27	P26	P25	P24	P23	P22	P21	P20	1111,1111
P3	P3 端口	B0H	P37	P36	P35	P34	P33	P32	P31	P30	1111,1111
P4	P4 端口	C0H	P47	P46	P45	P44	P43	P42	P41	P40	1111,1111
P5	P5 端口	C8H	-	-	-	P54	P53	P52	P51	P50	xxx1,1111
P6	P6 端口	E8H	P67	P66	P65	P64	P63	P62	P61	P60	1111,1111
P7	P7 端口	F8H	P77	P76	P75	P74	P73	P72	P71	P70	1111,1111
P0M0	P0 口配置寄存器 0	F2H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1	0000,0000
P0M1	P0 口配置寄存器 1	F1H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0	1111,1111
P1M0	P1 口配置寄存器 0	F4H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1	0000,0000
P1M1	P1 口配置寄存器 1	F3H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0	1111,1111
P2M0	P2 口配置寄存器 0	F6H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1	0000,0000
P2M1	P2 口配置寄存器 1	F5H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0	1111,1111
P3M0	P3 口配置寄存器 0	FAH	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1	0000,0000
P3M1	P3 口配置寄存器 1	F9H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0	1111,1100
P4M0	P4 口配置寄存器 0	FCH	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1	0000,0000
P4M1	P4 口配置寄存器 1	FBH	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0	1111,1111
P5M0	P5 口配置寄存器 0	EDH	-	-	-	P54M1	P53M1	P52M1	P51M1	P50M1	xxx0,0000
P5M1	P5 口配置寄存器 1	ECH	-	-	-	P54M0	P53M0	P52M0	P51M0	P50M0	xxx1,1111
P6M0	P6 口配置寄存器 0	EFH	P67M1	P66M1	P65M1	P64M1	P63M1	P62M1	P61M1	P60M1	0000,0000
P6M1	P6 口配置寄存器 1	EEH	P67M0	P66M0	P65M0	P64M0	P63M0	P62M0	P61M0	P60M0	1111,1111
P7M0	P7 口配置寄存器 0	D3H	P77M1	P76M1	P75M1	P74M1	P73M1	P72M1	P71M1	P70M1	0000,0000
P7M1	P7 口配置寄存器 1	D2H	P77M0	P76M0	P75M0	P74M0	P73M0	P72M0	P71M0	P70M0	1111,1111

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P0PU	P0 口上拉电阻控制寄存器	7EFE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU	0000,0000
P1PU	P1 口上拉电阻控制寄存器	7EFE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU	0000,0000
P2PU	P2 口上拉电阻控制寄存器	7EFE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU	0000,0000
P3PU	P3 口上拉电阻控制寄存器	7EFE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU	0000,0000
P4PU	P4 口上拉电阻控制寄存器	7EFE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU	0000,0000

P5PU	P5 口上拉电阻控制寄存器	7EFE15H	-	-	-	P54PU	P53PU	P52PU	P51PU	P50PU	xxx0,0000
P6PU	P6 口上拉电阻控制寄存器	7EFE16H	P67PU	P66PU	P65PU	P64PU	P63PU	P62PU	P61PU	P60PU	0000,0000
P7PU	P7 口上拉电阻控制寄存器	7EFE17H	P77PU	P76PU	P75PU	P74PU	P73PU	P72PU	P71PU	P70PU	0000,0000
P0NCS	P0 口施密特触发控制寄存器	7EFE18H	P07NCS	P06NCS	P05NCS	P04NCS	P03NCS	P02NCS	P01NCS	P00NCS	0000,0000
P1NCS	P1 口施密特触发控制寄存器	7EFE19H	P17NCS	P16NCS	P15NCS	P14NCS	P13NCS	P12NCS	P11NCS	P10NCS	0000,0000
P2NCS	P2 口施密特触发控制寄存器	7EFE1AH	P27NCS	P26NCS	P25NCS	P24NCS	P23NCS	P22NCS	P21NCS	P20NCS	0000,0000
P3NCS	P3 口施密特触发控制寄存器	7EFE1BH	P37NCS	P36NCS	P35NCS	P34NCS	P33NCS	P32NCS	P31NCS	P30NCS	0000,0000
P4NCS	P4 口施密特触发控制寄存器	7EFE1CH	P47NCS	P46NCS	P45NCS	P44NCS	P43NCS	P42NCS	P41NCS	P40NCS	0000,0000
P5NCS	P5 口施密特触发控制寄存器	7EFE1DH	-	-	-	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS	xxx0,0000
P6NCS	P6 口施密特触发控制寄存器	7EFE1EH	P67NCS	P66NCS	P65NCS	P64NCS	P63NCS	P62NCS	P61NCS	P60NCS	0000,0000
P7NCS	P7 口施密特触发控制寄存器	7EFE1FH	P77NCS	P76NCS	P75NCS	P74NCS	P73NCS	P72NCS	P71NCS	P70NCS	0000,0000
P0SR	P0 口电平转换速率寄存器	7EFE20H	P07SR	P06SR	P05SR	P04SR	P03SR	P02SR	P01SR	P00SR	1111,1111
P1SR	P1 口电平转换速率寄存器	7EFE21H	P17SR	P16SR	P15SR	P14SR	P13SR	P12SR	P11SR	P10SR	1111,1111
P2SR	P2 口电平转换速率寄存器	7EFE22H	P27SR	P26SR	P25SR	P24SR	P23SR	P22SR	P21SR	P20SR	1111,1111
P3SR	P3 口电平转换速率寄存器	7EFE23H	P37SR	P36SR	P35SR	P34SR	P33SR	P32SR	P31SR	P30SR	1111,1111
P4SR	P4 口电平转换速率寄存器	7EFE24H	P47SR	P46SR	P45SR	P44SR	P43SR	P42SR	P41SR	P40SR	1111,1111
P5SR	P5 口电平转换速率寄存器	7EFE25H	-	-	-	P54SR	P53SR	P52SR	P51SR	P50SR	xxx1,1111
P6SR	P6 口电平转换速率寄存器	7EFE26H	P67SR	P66SR	P65SR	P64SR	P63SR	P62SR	P61SR	P60SR	1111,1111
P7SR	P7 口电平转换速率寄存器	7EFE27H	P77SR	P76SR	P75SR	P74SR	P73SR	P72SR	P71SR	P70SR	1111,1111
P0DR	P0 口驱动电流控制寄存器	7EFE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR	1111,1111
P1DR	P1 口驱动电流控制寄存器	7EFE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR	1111,1111
P2DR	P2 口驱动电流控制寄存器	7EFE2AH	P27DR	P26DR	P25DR	P24DR	P23DR	P22DR	P21DR	P20DR	1111,1111
P3DR	P3 口驱动电流控制寄存器	7EFE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR	1111,1111
P4DR	P4 口驱动电流控制寄存器	7EFE2CH	P47DR	P46DR	P45DR	P44DR	P43DR	P42DR	P41DR	P40DR	1111,1111
P5DR	P5 口驱动电流控制寄存器	7EFE2DH	-	-	-	P54DR	P53DR	P52DR	P51DR	P50DR	xxx1,1111
P6DR	P6 口驱动电流控制寄存器	7EFE2EH	P67DR	P66DR	P65DR	P64DR	P63DR	P62DR	P61DR	P60DR	1111,1111
P7DR	P7 口驱动电流控制寄存器	7EFE2FH	P77DR	P76DR	P75DR	P74DR	P73DR	P72DR	P71DR	P70DR	1111,1111
P0IE	P0 口输入使能控制寄存器	7EFE30H	-	P06IE	P05IE	P04IE	P03IE	P02IE	P11IE	P00IE	x111,1111
P1IE	P1 口输入使能控制寄存器	7EFE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE	1111,1111
P3IE	P3 口输入使能控制寄存器	7EFE33H	P37IE	P36IE	P35IE	P34IE	P33IE	P32IE	P31IE	P30IE	1111,1111

9.1.1 端口数据寄存器 (Px)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0	80H	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
P1	90H	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
P2	A0H	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
P3	B0H	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
P4	C0H	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
P5	C8H	-	-	-	P5.4	P5.3	P5.2	P5.1	P5.0
P6	E8H	P6.7	P6.6	P6.5	P6.4	P6.3	P6.2	P6.1	P6.0
P7	F8H	P7.7	P7.6	P7.5	P7.4	P7.3	P7.2	P7.1	P7.0

读写端口状态

写 0: 输出低电平到端口缓冲区

写 1: 输出高电平到端口缓冲区

读: 直接读端口管脚上的电平

9.1.2 端口模式配置寄存器 (PxM0, PxM1)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0M0	F2H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0
P0M1	F1H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1
P1M0	F4H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0
P1M1	F3H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1
P2M0	F6H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0
P2M1	F5H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1
P3M0	FAH	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0
P3M1	F9H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1
P4M0	FCH	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0
P4M1	FBH	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1
P5M0	EDH	-	-	-	P54M0	P53M0	P52M0	P51M0	P50M0
P5M1	ECH	-	-	-	P54M1	P53M1	P52M1	P51M1	P50M1
P6M0	EFH	P67M0	P66M0	P65M0	P64M0	P63M0	P62M0	P61M0	P60M0
P6M1	EEH	P67M1	P66M1	P65M1	P64M1	P63M1	P62M1	P61M1	P60M1
P7M0	D3H	P77M0	P76M0	P75M0	P74M0	P73M0	P72M0	P71M0	P70M0
P7M1	D2H	P77M1	P76M1	P75M1	P74M1	P73M1	P72M1	P71M1	P70M1

配置端口的模式

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

9.1.3 端口上拉电阻控制寄存器 (PxPU)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0PU	7EFE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU
P1PU	7EFE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU
P2PU	7EFE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU
P3PU	7EFE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU
P4PU	7EFE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU
P5PU	7EFE15H	-	-	-	P54PU	P53PU	P52PU	P51PU	P50PU
P6PU	7EFE16H	P67PU	P66PU	P65PU	P64PU	P63PU	P62PU	P61PU	P60PU
P7PU	7EFE17H	P77PU	P76PU	P75PU	P74PU	P73PU	P72PU	P71PU	P70PU

端口内部4.1K上拉电阻控制位（注：P3.0和P3.1口上的上拉电阻可能会略小一些）

- 0: 禁止端口内部的 4.1K 上拉电阻
1: 使能端口内部的 4.1K 上拉电阻

9.1.4 端口施密特触发控制寄存器 (PxNCS)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0NCS	7EFE18H	P07NCS	P06NCS	P05NCS	P04NCS	P03NCS	P02NCS	P01NCS	P00NCS
P1NCS	7EFE19H	P17NCS	P16NCS	P15NCS	P14NCS	P13NCS	P12NCS	P11NCS	P10NCS
P2NCS	7EFE1AH	P27NCS	P26NCS	P25NCS	P24NCS	P23NCS	P22NCS	P21NCS	P20NCS
P3NCS	7EFE1BH	P37NCS	P36NCS	P35NCS	P34NCS	P33NCS	P32NCS	P31NCS	P30NCS
P4NCS	7EFE1CH	P47NCS	P46NCS	P45NCS	P44NCS	P43NCS	P42NCS	P41NCS	P40NCS
P5NCS	7EFE1DH	-	-	-	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS
P6NCS	7EFE1EH	P67NCS	P66NCS	P65NCS	P64NCS	P63NCS	P62NCS	P61NCS	P60NCS
P7NCS	7EFE1FH	P77NCS	P76NCS	P75NCS	P74NCS	P73NCS	P72NCS	P71NCS	P70NCS

端口施密特触发控制位

- 0: **使能**端口的施密特触发功能。（上电复位后默认使能施密特触发）
1: **禁止**端口的施密特触发功能。

9.1.5 端口电平转换速度控制寄存器 (PxSR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0SR	7EFE20H	P07SR	P06SR	P05SR	P04SR	P03SR	P02SR	P01SR	P00SR
P1SR	7EFE21H	P17SR	P16SR	P15SR	P14SR	P13SR	P12SR	P11SR	P10SR
P2SR	7EFE22H	P27SR	P26SR	P25SR	P24SR	P23SR	P22SR	P21SR	P20SR
P3SR	7EFE23H	P37SR	P36SR	P35SR	P34SR	P33SR	P32SR	P31SR	P30SR
P4SR	7EFE24H	P47SR	P46SR	P45SR	P44SR	P43SR	P42SR	P41SR	P40SR
P5SR	7EFE25H	-	-	-	P54SR	P53SR	P52SR	P51SR	P50SR
P6SR	7EFE26H	P57SR	P66SR	P65SR	P64SR	P63SR	P62SR	P61SR	P60SR
P7SR	7EFE27H	P77SR	P76SR	P75SR	P74SR	P73SR	P72SR	P71SR	P70SR

控制端口电平转换的速度

- 0: 电平转换速度快，相应的上下冲会比较大
1: 电平转换速度慢，相应的上下冲比较小

9.1.6 端口驱动电流控制寄存器 (PxDR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0DR	7EFE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR
P1DR	7EFE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR
P2DR	7EFE2AH	P27DR				P26DR			
P3DR	7EFE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR
P4DR	7EFE2CH	P47DR	P46DR	P45DR	P44DR	P43DR	P42DR	P41DR	P40DR
P5DR	7EFE2DH	-	-	-	P54DR	P53DR	P52DR	P51DR	P50DR
P6DR	7EFE2EH	P67DR	P66DR	P65DR	P64DR	P63DR	P62DR	P61DR	P60DR
P7DR	7EFE2FH	P77DR	P76DR	P75DR	P74DR	P73DR	P72DR	P71DR	P70DR

控制端口的驱动能力

- 0: 一般驱动能力
- 1: 增强驱动能力

9.1.7 端口数字信号输入使能控制寄存器 (PxIE)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0IE	7EFE30H	-	P06IE	P05IE	P04IE	P03IE	P02IE	P11IE	P00IE
P1IE	7EFE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE
P3IE	7EFE33H	P37IE	P36IE	P35IE	P34IE	P33IE	P32IE	P31IE	P30IE

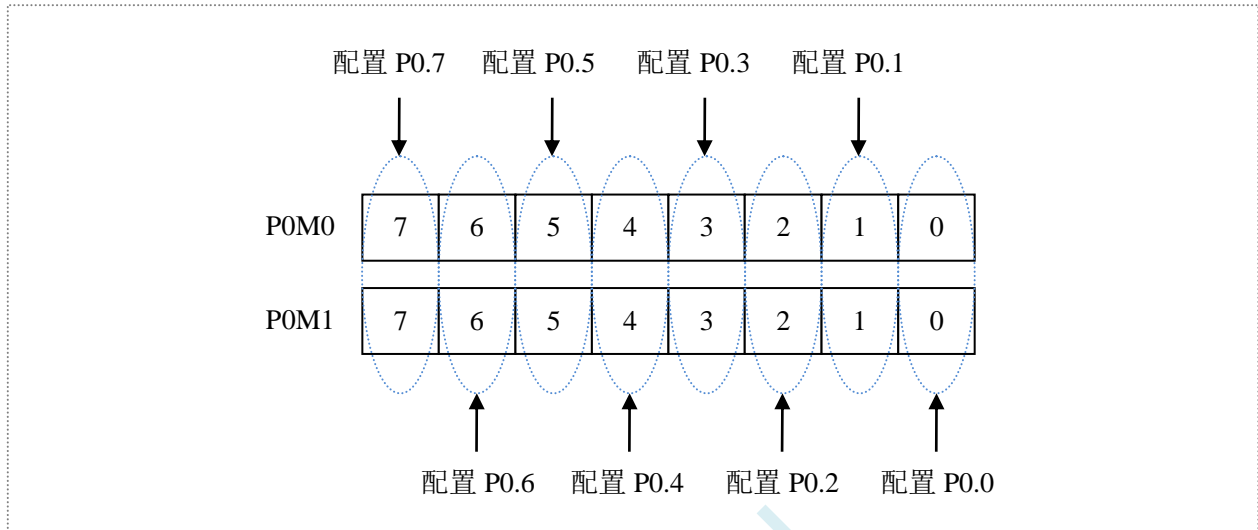
数字信号输入使能控制

- 0: 禁止数字信号输入。若 I/O 被当作比较器输入口、ADC 输入口或者触摸按键输入口等模拟口时，进入时钟停振模式前，必须设置为 0，否则会有额外的耗电。
- 1: 使能数字信号输入。若 I/O 被当作数字口时，必须设置为 1，否 MCU 无法读取外部端口的电平。

9.2 配置 I/O 口

每个 I/O 的配置都需要使用两个寄存器进行设置。

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



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

PnM0 与 PnM1 的组合方式如下表所示

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

注：n = 0, 1, 2, 3, 4, 5, 6, 7

注意：

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

9.3 I/O 的结构图

9.3.1 准双向口（弱上拉）

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

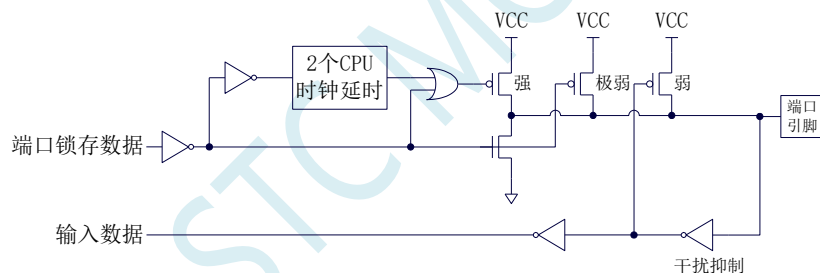
在 3 个上拉晶体管中，有 1 个上拉晶体管称为“弱上拉”，当端口寄存器为 1 且引脚本身为 1 时打开。此上拉提供基本驱动电流使准双向口输出为 1。如果一个引脚输出为 1 而由外部装置下拉到低时，弱上拉关闭而“极弱上拉”维持开状态，为了把这个引脚强拉为低，外部装置必须有足够的灌电流能力使引脚上的电压降到门槛电压以下。对于 5V 单片机，“弱上拉”晶体管的电流约 250uA；对于 3.3V 单片机，“弱上拉”晶体管的电流约 150uA。

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

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

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

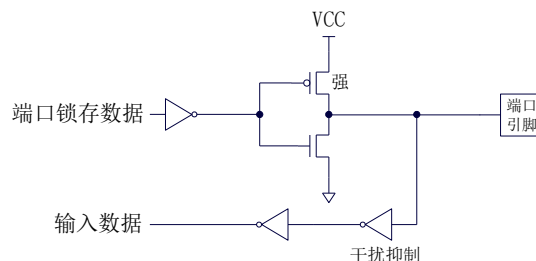
准双向口（弱上拉）输出如下图所示：



9.3.2 推挽输出

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

强推挽引脚配置如下图所示：

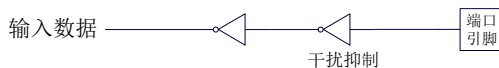


9.3.3 高阻输入

电流既不能流入也不能流出

输入口带有一个施密特触发输入以及一个干扰抑制电路

高阻输入引脚配置如下图所示:



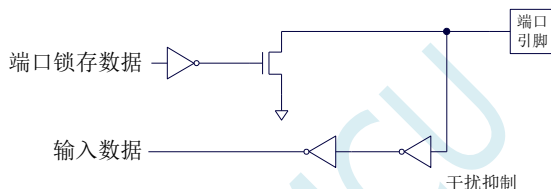
9.3.4 开漏输出

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

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

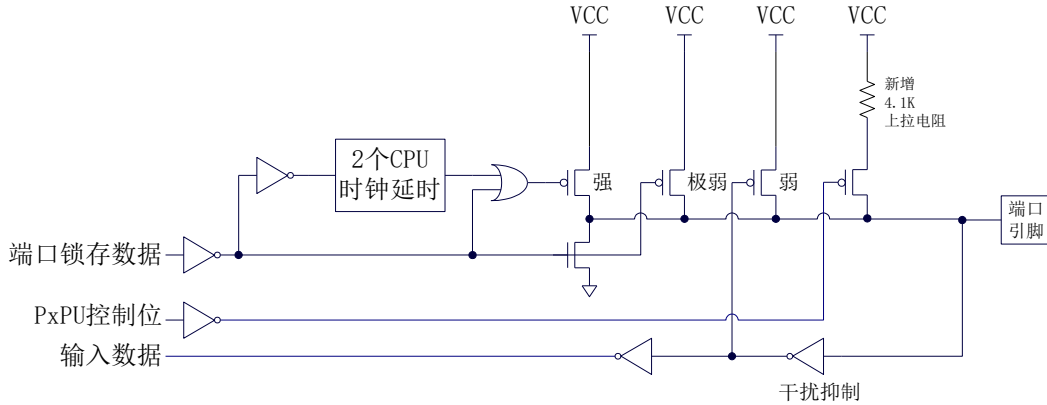
开漏端口带有一个施密特触发输入以及一个干扰抑制电路。

输出端口配置如下图所示:



9.3.5 新增 4.1K 上拉电阻

STC16 系列所有的 I/O 口内部均可使能一个大约 4.1K 的上拉电阻（由于制造误差，上拉电阻的范围可能为 3K~5K）



端口上拉电阻控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0PU	7EFE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU
P1PU	7EFE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU
P2PU	7EFE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU
P3PU	7EFE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU
P4PU	7EFE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU
P5PU	7EFE15H	-	-	-	P54PU	P53PU	P52PU	P51PU	P50PU
P6PU	7EFE16H	P67PU	P66PU	P65PU	P64PU	P63PU	P62PU	P61PU	P60PU
P7PU	7EFE17H	P77PU	P76PU	P75PU	P74PU	P73PU	P72PU	P71PU	P70PU

端口内部4.1K上拉电阻控制位（注：P3.0和P3.1口上的上拉电阻可能会略小一些）

- 0：禁止端口内部的 4.1K 上拉电阻
- 1：使能端口内部的 4.1K 上拉电阻

9.3.6 如何设置 I/O 口对外输出速度

当用户需要 I/O 口对外输出较快的频率时，可通过加大 I/O 口驱动电流以及增加 I/O 口电平转换速度以达到提高 I/O 口对外输出速度

设置 PxSR 寄存器，可用于控制 I/O 口电平转换速度，设置为 0 时相应的 I/O 口为快速翻转，设置为 1 时为慢速翻转。

设置 PxDR 寄存器，可用于控制 I/O 口驱动电流大小，设置为 1 时 I/O 输出为一般驱动电流，设置为 0 时为强驱动电流

9.3.7 如何设置 I/O 口电流驱动能力

若需要改变 I/O 口的电流驱动能力，可通过设置 PxDR 寄存器来实现

设置 PxDR 寄存器，可用于控制 I/O 口驱动电流大小，设置为 1 时 I/O 输出为一般驱动电流，设置为 0 时为强驱动电流

9.3.8 如何降低 I/O 口对外辐射

由于设置 PxSR 寄存器，可用于控制 I/O 口电平转换速度，设置 PxDR 寄存器，可用于控制 I/O 口驱动电流大小

当需要降低 I/O 口对外的辐射时，需要将 PxSR 寄存器设置为 1 以降低 I/O 口电平转换速度，同时需要将 PxDR 寄存器设为 1 以降低 I/O 驱动电流，最终达到降低 I/O 口对外辐射

STC MCU

9.4 范例程序

9.4.1 端口模式设置

C 语言代码

//测试工作频率为11.0592MHz

#include "stc16.h" //头文件见附录
#include "intrins.h"

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

    while (1);
}
```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC) ;头文件见附录

```
ORG 0000H
LJMP MAIN

ORG 0100H
MAIN:
MOV SP, #5FH

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

JMP $

END
```

9.4.2 双向口读写操作

C 语言代码

//测试工作频率为11.0592MHz

#include "stc16.h" //头文件见附录

#include "intrins.h"

void main()

{

P0M0 = 0x00;

P0M1 = 0x00;

P1M0 = 0x00;

P1M1 = 0x00;

P2M0 = 0x00;

P2M1 = 0x00;

P3M0 = 0x00;

P3M1 = 0x00;

P4M0 = 0x00;

P4M1 = 0x00;

P5M0 = 0x00;

P5M1 = 0x00;

P0M0 = 0x00;

//设置 P0.0~P0.7 为双向口模式

P0M1 = 0x00;

P00 = 1;

//P0.0 口输出高电平

P00 = 0;

//P0.0 口输出低电平

P00 = 1;

//读取端口前先使能内部弱上拉电阻

nop();

//等待两个时钟

nop();

//

CY = P00;

//读取端口状态

while (1);

}

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC)

;头文件见附录

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

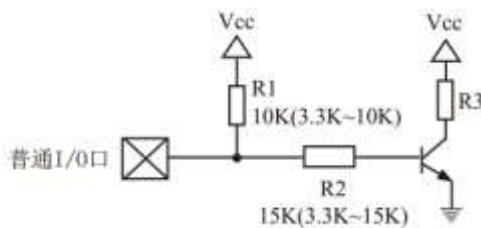
MOV P2M1, #00H

MOV P3M0, #00H

MOV P3M1, #00H

<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>MOV</i>	<i>P0M0, #00H</i>	;设置 P0.0~P0.7 为双向口模式
<i>MOV</i>	<i>P0M1, #00H</i>	
<i>SETB</i>	<i>P0.0</i>	;P0.0 口输出高电平
<i>CLR</i>	<i>P0.0</i>	;P0.0 口输出低电平
<i>SETB</i>	<i>P0.0</i>	;读取端口前先使能内部弱上拉电阻
<i>NOP</i>		;等待两个时钟
<i>NOP</i>		
<i>MOV</i>	<i>C, P0.0</i>	;读取端口状态
<i>JMP</i>	<i>\$</i>	
<i>END</i>		

9.5 一种典型三极管控制电路



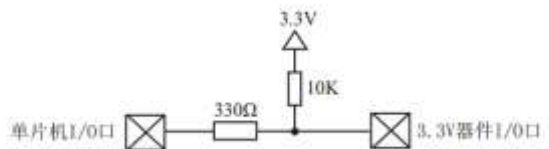
如果上拉控制, 建议加上拉电阻 $R1(3.3K \sim 10K)$, 如果不加上拉电阻 $R1(3.3K \sim 10K)$, 建议 $R2$ 的值在 $15K$ 以上, 或用强推挽输出。

9.6 典型发光二极管控制电路



9.7 混合电压供电系统 3V/5V 器件 I/O 口互连

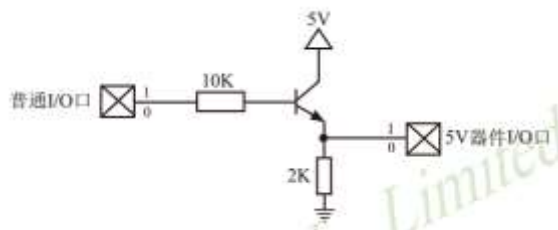
STC 系列宽电压单片机工作在 5V 时, 如需要直接连接 3.3V 器件时, 为防止 3.3V 器件承受不了 5V, 可将相应的单片机 I/O 口先串一个 330Ω 的限流电阻到 3.3V 器件 I/O 口, 程序初始化时将单片机的 I/O 口设置成开漏配置, 断开内部上拉电阻, 相应的 3.3V 器件 I/O 口外部加 10K 上拉电阻到 3.3V 器件的 Vcc, 这样高电平是 3.3V, 低电平是 0V, 输入输出一切正常。



STC 宽电压单片机工作在 3V 时, 如需要直接连接 5V 器件时, 为防止 3V 单片机承受不了 5V, 如果相应的 I/O 口是输入, 可在该 I/O 口上串接一个隔离二极管, 隔离高压部分。外部信号电压高于单片机工作电压时截止, I/O 口因内部上拉到高电平, 所以读 I/O 口状态是高电平; 外部信号电压为低时导通, I/O 口被钳位在 0.7V, 小于 0.8V 时单片机读 I/O 口状态是低电平。



STC 宽电压单片机工作在 3V 时, 如需要直接连接 5V 器件时, 为防止 3V 单片机承受不了 5V, 如果相应的 I/O 口是输出, 可用一个 NPN 三极管隔离, 电路如下:



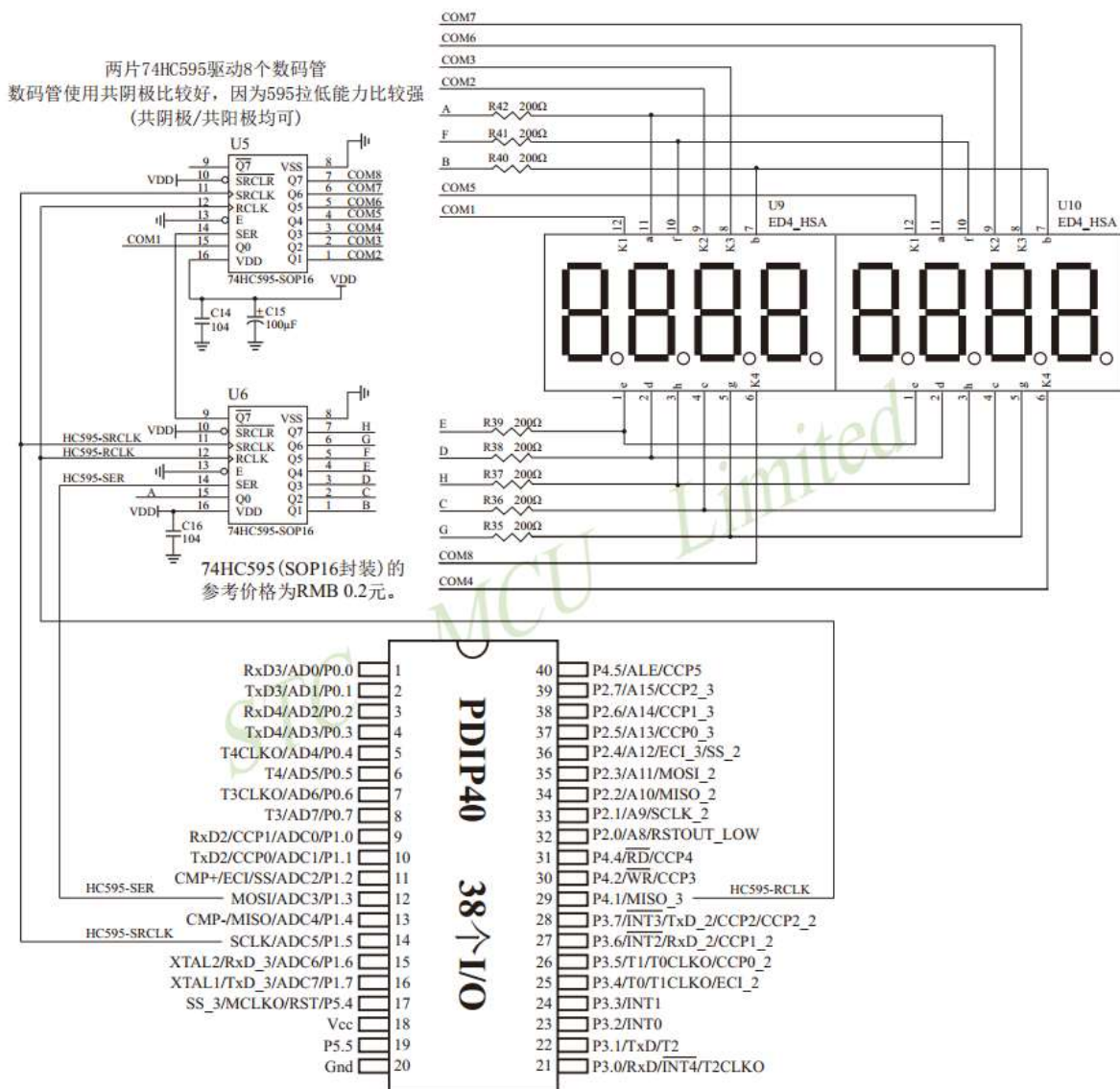
9.8 如何让 I/O 口上电复位时为低电平

普通 8051 单片机上电复位时普通 I/O 口为弱上拉(准双向口)高电平输出, 而很多实际应用要求上电时某些 I/O 口为低电平输出, 否则所控制的系统(如马达)就会误动作, 现 STC 的单片机由于既有弱上拉输出又有强推挽输出, 就可以很轻松的解决此问题。

现可在 STC 的单片机 I/O 口上加一个下拉电阻(10K 左右), 这样上电复位时, 除了下载口 P3.0 和 P3.1 为弱上拉(准双向口)外, 其他 I/O 口均为高阻输入模式, 而外部有下拉电阻, 所以该 I/O 口上电复位时外部为低电平。如果要将此 I/O 口驱动为高电平, 可将此 I/O 口设置为强推挽输出, 而强推挽输出时, I/O 口驱动电流可达 20mA, 故肯定可以将该口驱动为高电平输出。



9.9 利用 74HC595 驱动 8 个数码管(串行扩展,3 根线)的线路图



10 中断系统

中断系统是为使 CPU 具有对外界紧急事件的实时处理能力而设置的。

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

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

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

10.1 STC16F 系列中断源

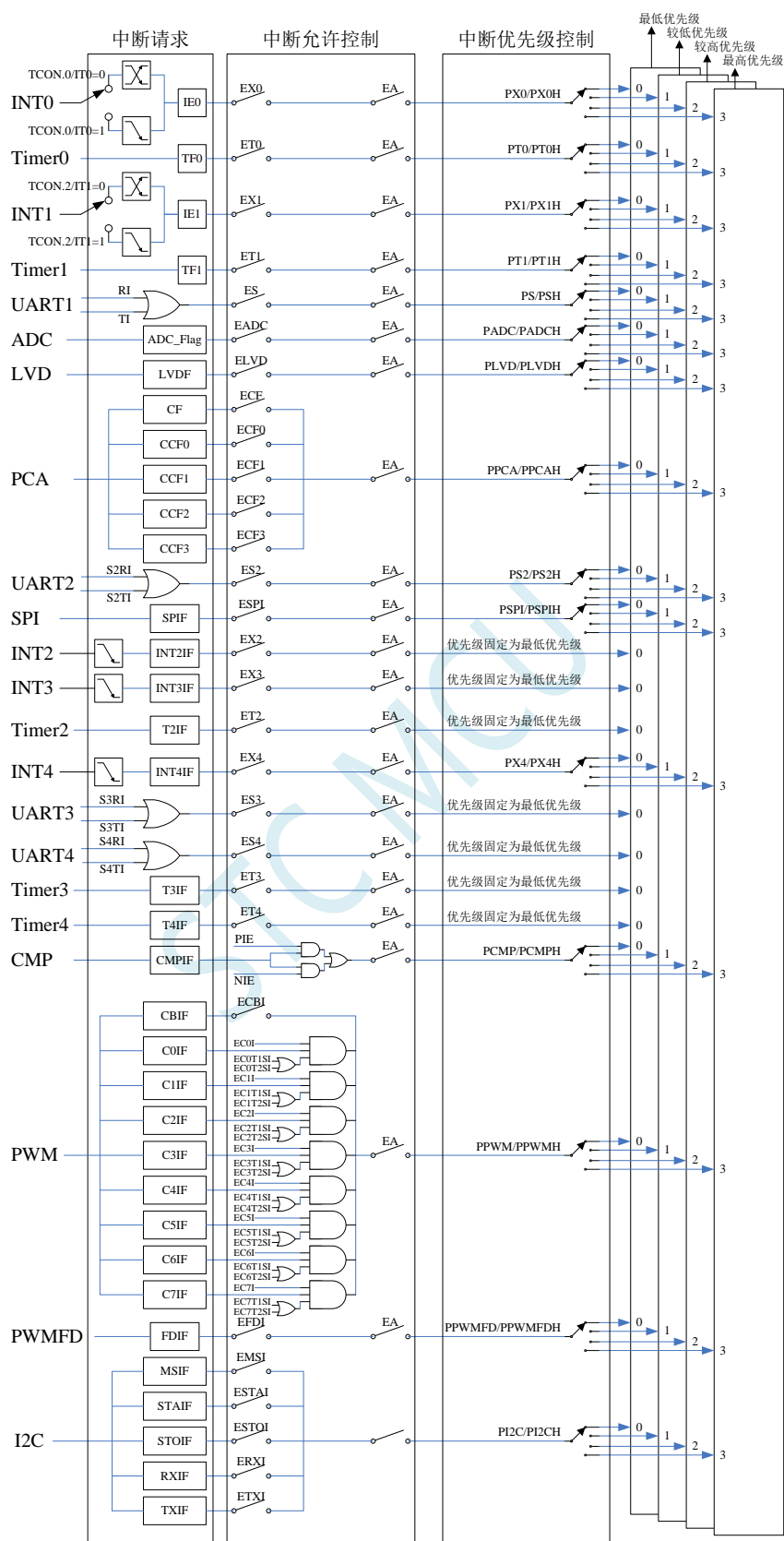
下表中√表示对应的系列有相应的中断源

中断源	STC16F40K128 系列
外部中断 0 中断（INT0）	√
定时器 0 中断（Timer0）	√
外部中断 1 中断（INT1）	√
定时器 1 中断（Timer1）	√
串口 1 中断（UART1）	√
模数转换中断（ADC）	√
低压检测中断（LVD）	√
串口 2 中断（UART2）	√
串行外设接口中断（SPI）	√
外部中断 2 中断（INT2）	√
外部中断 3 中断（INT3）	√
定时器 2 中断（Timer2）	√
外部中断 4 中断（INT4）	√
串口 3 中断（UART3）	√
串口 4 中断（UART4）	√
定时器 3 中断（Timer3）	√

定时器 4 中断 (Timer4)	√
比较器中断 (CMP)	√
I2C 总线中断	√
PWMA	√
PWMB	√
USB 中断	√
CAN 中断	√
LIN 中断	√

STC MCU

10.2 STC16F 中断结构图



10.3 STC16F 系列中断列表

中断源	中断向量	次序	优先级设置	优先级	中断请求位	中断允许位
INT0	FF0003H	0	PX0PX0H	0/1/2/3	IE0	EX0
Timer0	FF000BH	1	PT0,PT0H	0/1/2/3	TF0	ET0
INT1	FF0013H	2	PX1,PX1H	0/1/2/3	IE1	EX1
Timer1	FF001BH	3	PT1,PT1H	0/1/2/3	TF1	ET1
UART1	FF0023H	4	PS,PSH	0/1/2/3	RI TI	ES
ADC	FF002BH	5	PADC,PADCH	0/1/2/3	ADC_FLAG	EADC
LVD	FF0033H	6	PLVD,PLVDH	0/1/2/3	LVDF	ELVD
PWMA	FF003BH	7	PPWMA,PPWMAH	0/1/2/3	PWMA_SR	PWMA_IER
UART2	FF0043H	8	PS2,PS2H	0/1/2/3	S2RI S2TI	ES2
SPI	FF004BH	9	PSPI,PSPIH	0/1/2/3	SPIF	ESPI
INT2	FF0053H	10		0	INT2IF	EX2
INT3	FF005BH	11		0	INT3IF	EX3
Timer2	FF0063H	12		0	T2IF	ET2
INT4	FF0083H	16	PX4,PX4H	0/1/2/3	INT4IF	EX4
UART3	FF008BH	17	PS3,PS3H	0/1/2/3	S3RI S3TI	ES3
UART4	FF0093H	18	PS4,PS4H	0/1/2/3	S4RI S4TI	ES4
Timer3	FF009BH	19		0	T3IF	ET3
Timer4	FF00A3H	20		0	T4IF	ET4
CMP	FF00ABH	21	PCMP,PCMPH	0/1/2/3	CMPIF	PIE NIE
USB	FF00B3H	22	PUSB,PUSBH	0/1/2/3	USB Events	EUSB
PWMB	FF00BBH	23	PPWMB,PPWMBH	0/1/2/3	PWMB_SR	PWMB_IER
I2C	FF00C3H	24	PI2C,PI2CH	0/1/2/3	MSIF	EMSI
					STAIF	ESTAI
					RXIF	ERXI
					TXIF	ETXI
					STOIF	ESTOI
CANBUS	FF00E3H	28	PCANL,PCANH	0/1/2/3	ALI	ALIM
					EWI	EWIM
					EPI	EPIM
					RI	RIM
					TI	TIM
					BEI	BEIM
					DOI	DOIM
LINBUS	FF00EBH	29	PLINL,PLINH	0/1/2/3	ABORT	ABORTE

					ERR	ERRE
					RDY	RDYE
					LID	LIDE

在 C 语言中声明中断服务程序

```

void  INT0_Routine(void)    interrupt  0;
void  TM0_Routine(void)    interrupt  1;
void  INT1_Routine(void)    interrupt  2;
void  TM1_Routine(void)    interrupt  3;
void  UART1_Routine(void)   interrupt  4;
void  ADC_Routine(void)     interrupt  5;
void  LVD_Routine(void)     interrupt  6;
void  PWMA_Routine(void)    interrupt  7;
void  UART2_Routine(void)   interrupt  8;
void  SPI_Routine(void)     interrupt  9;
void  INT2_Routine(void)    interrupt 10;
void  INT3_Routine(void)    interrupt 11;
void  TM2_Routine(void)     interrupt 12;
void  INT4_Routine(void)    interrupt 16;
void  UART3_Routine(void)   interrupt 17;
void  UART4_Routine(void)   interrupt 18;
void  TM3_Routine(void)     interrupt 19;
void  TM4_Routine(void)     interrupt 20;
void  CMP_Routine(void)     interrupt 21;
void  USB_Routine(void)     interrupt 22;
void  PWMB_Routine(void)    interrupt 23;
void  I2C_Routine(void)     interrupt 24;
void  CAN_Routine(void)     interrupt 28;
void  LIN_Routine(void)     interrupt 29;

```


10.4 中断相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
IE	中断允许寄存器	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000,0000
IE2	中断允许寄存器 2	E7H	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000,0000
INTCLKO	中断与时钟输出控制寄存器	95H	-	EX4	EX3	EX2	-	T2CKO	T1CKO	T0CKO	x000,x000
IP	中断优先级控制寄存器	B8H	PPWMA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	x000,0000
IPH	高中断优先级控制寄存器	B7H	PPWMAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	x000,0000
IP2	中断优先级控制寄存器 2	FEH	-	PI2C	PCMP	PX4	PPWMB	PUSB	PSPI	PS2	0000,0000
IP2H	高中断优先级控制寄存器 2	B6H	-	PI2CH	PCMPH	PX4H	PPWMBH	PUSBH	PSPIH	PS2H	0000,0000
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
AUXINTIF	扩展外部中断标志寄存器	DAH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF	x000,x000
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
S2CON	串口 2 控制寄存器	E3H	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
S3CON	串口 3 控制寄存器	E5H	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S4CON	串口 4 控制寄存器	BCH	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
CANICR	CANBUS 中断控制寄存器	A1H	-	-	-	-	PCANH	CANIF	ECAN	PCANL	0000,0000
LINICR	LINBUS 中断控制寄存器	B1H	-	-	-	-	PLINH	LINIF	ELIN	PLINL	0000,0000
CMPCR1	比较器控制寄存器 1	B4H	CMPEM	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
SPSTAT	SPI 状态寄存器	B9H	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
ADC_CONTR	ADC 控制寄存器	DCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]				000x,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
I2CMSCR	I ² C 主机控制寄存器	7EFE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I ² C 主机状态寄存器	7EFE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I ² C 从机控制寄存器	7EFE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I ² C 从机状态寄存器	7EFE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
PWMA_IER	PWMA 中断使能寄存器	7EFEC4H	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE	0000,0000
PWMA_SR1	PWMA 状态寄存器 1	7EFEC5H	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF	0000,0000
PWMA_SR2	PWMA 状态寄存器 2	7EFEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-	xxx0,000x
PWMB_IER	PWMB 中断使能寄存器	7EFEE4H	BIE	TIE	COMIE	CC8IE	CC7IE	CC6IE	CC5IE	UIE	0000,0000
PWMB_SR1	PWMB 状态寄存器 1	7EFEE5H	BIF	TIF	COMIF	CC8IF	CC7IF	CC6IF	CC5IF	UIF	0000,0000
PWMB_SR2	PWMB 状态寄存器 2	7EFEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-	xxx0,000x

10.4.1 中断使能寄存器（中断允许位）

IE（中断使能寄存器）

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

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

各中断源自己的中断允许控制位控制。

0: CPU 屏蔽所有的中断申请

1: CPU 开放中断

ELVD: 低压检测中断允许位。

0: 禁止低压检测中断

1: 允许低压检测中断

EADC: A/D 转换中断允许位。

0: 禁止 A/D 转换中断

1: 允许 A/D 转换中断

ES: 串行口 1 中断允许位。

0: 禁止串行口 1 中断

1: 允许串行口 1 中断

ET1: 定时/计数器 T1 的溢出中断允许位。

0: 禁止 T1 中断

1: 允许 T1 中断

EX1: 外部中断 1 中断允许位。

0: 禁止 INT1 中断

1: 允许 INT1 中断

ET0: 定时/计数器 T0 的溢出中断允许位。

0: 禁止 T0 中断

1: 允许 T0 中断

EX0: 外部中断 0 中断允许位。

0: 禁止 INT0 中断

1: 允许 INT0 中断

IE2 (中断使能寄存器 2)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IE2	E7H	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

ET4: 定时/计数器 T4 的溢出中断允许位。

0: 禁止 T4 中断

1: 允许 T4 中断

ET3: 定时/计数器 T3 的溢出中断允许位。

0: 禁止 T3 中断

1: 允许 T3 中断

ES4: 串行口 4 中断允许位。

0: 禁止串行口 4 中断

1: 允许串行口 4 中断

ES3: 串行口 3 中断允许位。

0: 禁止串行口 3 中断

1: 允许串行口 3 中断

ET2: 定时/计数器 T2 的溢出中断允许位。

0: 禁止 T2 中断

1: 允许 T2 中断

ESPI: SPI 中断允许位。

0: 禁止 SPI 中断

1: 允许 SPI 中断

ES2: 串行口 2 中断允许位。

0: 禁止串行口 2 中断

1: 允许串行口 2 中断

INTCLKO (外部中断与时钟输出控制寄存器)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTCLKO	95H	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

EX4: 外部中断 4 中断允许位。

0: 禁止 INT4 中断

1: 允许 INT4 中断

EX3: 外部中断 3 中断允许位。

0: 禁止 INT3 中断

1: 允许 INT3 中断

EX2: 外部中断 2 中断允许位。

0: 禁止 INT2 中断

1: 允许 INT2 中断

CMPCR1 (比较器控制寄存器 1)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	B4H	COMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES

PIE: 比较器上升沿中断允许位。

0: 禁止比较器上升沿中断

1: 允许比较器上升沿中断

NIE: 比较器下降沿中断允许位。

0: 禁止比较器下降沿中断

1: 允许比较器下降沿中断

I2C 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	7EFE81H	EMSI	-	-	-	-	MSCMD[2:0]		
I2CSLCR	7EFE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST

EMSI: I²C 主机模式中断允许位。

0: 禁止 I²C 主机模式中断

1: 允许 I²C 主机模式中断

ESTAI: I²C 从机接收 START 事件中断允许位。

0: 禁止 I²C 从机接收 START 事件中断

1: 允许 I²C 从机接收 START 事件中断

ERXI: I²C 从机接收数据完成事件中断允许位。

0: 禁止 I²C 从机接收数据完成事件中断

1: 允许 I²C 从机接收数据完成事件中断

ETXI: I²C 从机发送数据完成事件中断允许位。

0: 禁止 I²C 从机发送数据完成事件中断

1: 允许 I²C 从机发送数据完成事件中断

ESTOI: I²C从机接收STOP事件中断允许位。

0: 禁止 I²C 从机接收 STOP 事件中断

1: 允许 I²C 从机接收 STOP 事件中断

PWMA 中断使能寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_IER	7EFEC4H	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE

BIE: PWMA刹车中断允许位。

0: 禁止 PWMA 刹车中断

1: 允许 PWMA 刹车中断

TIE: PWMA触发中断允许位。

0: 禁止 PWMA 触发中断

1: 允许 PWMA 触发中断

COMIE: PWMA比较中断允许位。

0: 禁止 PWMA 比较中断

1: 允许 PWMA 比较中断

CC4IE: PWMA捕获比较通道4中断允许位。

0: 禁止 PWMA 捕获比较通道 4 中断

1: 允许 PWMA 捕获比较通道 4 中断

CC3IE: PWMA捕获比较通道3中断允许位。

0: 禁止 PWMA 捕获比较通道 3 中断

1: 允许 PWMA 捕获比较通道 3 中断

CC2IE: PWMA捕获比较通道2中断允许位。

0: 禁止 PWMA 捕获比较通道 2 中断

1: 允许 PWMA 捕获比较通道 2 中断

CC1IE: PWMA捕获比较通道1中断允许位。

0: 禁止 PWMA 捕获比较通道 1 中断

1: 允许 PWMA 捕获比较通道 1 中断

UIE: PWMA更新中断允许位。

0: 禁止 PWMA 更新中断

1: 允许 PWMA 更新中断

PWMB 中断使能寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMB_IER	7EFEE4H	BIE	TIE	COMIE	CC8IE	CC7IE	CC6IE	CC5IE	UIE

BIE: PWMB刹车中断允许位。

0: 禁止 PWMB 刹车中断

1: 允许 PWMB 刹车中断

TIE: PWMB触发中断允许位。

0: 禁止 PWMB 触发中断

1: 允许 PWMB 触发中断

COMIE: PWMB比较中断允许位。

0: 禁止 PWMB 比较中断

1: 允许 PWMB 比较中断

CC8IE: PWMB捕获比较通道8中断允许位。

0: 禁止 PWMB 捕获比较通道 8 中断

1: 允许 PWMB 捕获比较通道 8 中断

CC7IE: PWMB捕获比较通道7中断允许位。

0: 禁止 PWMB 捕获比较通道 7 中断

1: 允许 PWMB 捕获比较通道 7 中断

CC6IE: PWMB捕获比较通道6中断允许位。

0: 禁止 PWMB 捕获比较通道 6 中断

1: 允许 PWMB 捕获比较通道 6 中断

CC5IE: PWMB捕获比较通道5中断允许位。

0: 禁止 PWMB 捕获比较通道 5 中断

1: 允许 PWMB 捕获比较通道 5 中断

UIE: PWMB更新中断允许位。

0: 禁止 PWMB 更新中断

1: 允许 PWMB 更新中断

10.4.2 中断请求寄存器（中断标志位）

定时器控制寄存器

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

TF1: 定时器1溢出中断标志。中断服务程序中，硬件自动清零。

TF0: 定时器0溢出中断标志。中断服务程序中，硬件自动清零。

IE1: 外部中断1中断请求标志。中断服务程序中，硬件自动清零。

IE0: 外部中断0中断请求标志。中断服务程序中，硬件自动清零。

中断标志辅助寄存器

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

INT4IF: 外部中断4中断请求标志。中断服务程序中，硬件自动清零。

INT3IF: 外部中断3中断请求标志。中断服务程序中，硬件自动清零。

INT2IF: 外部中断2中断请求标志。中断服务程序中，硬件自动清零。

T4IF: 定时器4溢出中断标志。中断服务程序中，硬件自动清零。

T3IF: 定时器3溢出中断标志。中断服务程序中，硬件自动清零。

T2IF: 定时器2溢出中断标志。中断服务程序中，硬件自动清零。

串口控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI
S2CON	E3H	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI
S3CON	E5H	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI
S4CON	BCH	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

TI: 串口1发送完成中断请求标志。需要软件清零。

RI: 串口1接收完成中断请求标志。需要软件清零。

S2TI: 串口2发送完成中断请求标志。需要软件清零。

S2RI: 串口2接收完成中断请求标志。需要软件清零。

S3TI: 串口3发送完成中断请求标志。需要软件清零。

S3RI: 串口3接收完成中断请求标志。需要软件清零。

S4TI: 串口4发送完成中断请求标志。需要软件清零。

S4RI: 串口4接收完成中断请求标志。需要软件清零。

电源管理寄存器

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

LVDF: 低压检测中断请求标志。需要软件清零。

ADC 控制寄存器

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

ADC_FLAG: ADC转换完成中断请求标志。需要软件清零。

SPI 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	B9H	SPIF	WCOL	-	-	-	-	-	-

SPIF: SPI数据传输完成中断请求标志。需要软件清零。

比较器控制寄存器 1

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	B4H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES

CMPIF: 比较器中断请求标志。需要软件清零。

I2C 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSST	7EFE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO
I2CSLST	7EFE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO

MSIF: I²C主机模式中断请求标志。需要软件清零。

ESTAI: I²C从机接收START事件中断请求标志。需要软件清零。

ERXI: I²C从机接收数据完成事件中断请求标志。需要软件清零。

ETXI: I²C从机发送数据完成事件中断请求标志。需要软件清零。

ESTOI: I²C从机接收STOP事件中中断请求标志。需要软件清零。

PWMA 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SR1	7EFEC5H	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF
PWMA_SR2	7EFEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-

BIF: PWMA刹车中断请求标志。需要软件清零。

TIF: PWMA触发中断请求标志。需要软件清零。

COMIF: PWMA比较中断请求标志。需要软件清零。

CC4IF: PWMA通道4发生捕获比较中断请求标志。需要软件清零。

CC3IF: PWMA通道3发生捕获比较中断请求标志。需要软件清零。

CC2IF: PWMA通道2发生捕获比较中断请求标志。需要软件清零。

CC1IF: PWMA通道1发生捕获比较中断请求标志。需要软件清零。

TIF: PWMA更新中断请求标志。需要软件清零。

CC4OF: PWMA通道4发生重复捕获中断请求标志。需要软件清零。

CC3OF: PWMA通道3发生重复捕获中断请求标志。需要软件清零。

CC2OF: PWMA通道2发生重复捕获中断请求标志。需要软件清零。

CC1OF: PWMA通道1发生重复捕获中断请求标志。需要软件清零。

PWMB 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMB_SR1	7EFEE5H	BIF	TIF	COMIF	CC8IF	CC7IF	CC6IF	CC5IF	UIF
PWMB_SR2	7EFEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-

BIF: PWMB刹车中断请求标志。需要软件清零。

TIF: PWMB触发中断请求标志。需要软件清零。

COMIF: PWMB比较中断请求标志。需要软件清零。

CC8IF: PWMB通道8发生捕获比较中断请求标志。需要软件清零。

CC7IF: PWMB通道7发生捕获比较中断请求标志。需要软件清零。

CC6IF: PWMB通道6发生捕获比较中断请求标志。需要软件清零。

CC5IF: PWMB通道5发生捕获比较中断请求标志。需要软件清零。

TIF: PWMB更新中断请求标志。需要软件清零。

CC8OF: PWMB通道8发生重复捕获中断请求标志。需要软件清零。

CC7OF: PWMB通道7发生重复捕获中断请求标志。需要软件清零。

CC6OF: PWMB通道6发生重复捕获中断请求标志。需要软件清零。

CC5OF: PWMB通道5发生重复捕获中断请求标志。需要软件清零。

10.4.3 中断优先级寄存器

中断优先级控制寄存器

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

PX0H,PX0: 外部中断0中断优先级控制位

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

PT0H,PT0: 定时器0中断优先级控制位

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

PX1H,PX1: 外部中断1中断优先级控制位

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

PT1H,PT1: 定时器1中断优先级控制位

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

PSH,PS: 串口1中断优先级控制位

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

PADCH,PADC: ADC中断优先级控制位

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

PLVDH,PLVD: 低压检测中断优先级控制位

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

PS2H,PS2: 串口2中断优先级控制位

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

11: 串口 2 中断优先级为 3 级 (最高级)

PS3H,PS3: 串口3中断优先级控制位

00: 串口 3 中断优先级为 0 级 (最低级)

01: 串口 3 中断优先级为 1 级 (较低级)

10: 串口 3 中断优先级为 2 级 (较高级)

11: 串口 3 中断优先级为 3 级 (最高级)

PS4H,PS4: 串口4中断优先级控制位

00: 串口 4 中断优先级为 0 级 (最低级)

01: 串口 4 中断优先级为 1 级 (较低级)

10: 串口 4 中断优先级为 2 级 (较高级)

11: 串口 4 中断优先级为 3 级 (最高级)

PSPIH,PSPI: SPI中断优先级控制位

00: SPI 中断优先级为 0 级 (最低级)

01: SPI 中断优先级为 1 级 (较低级)

10: SPI 中断优先级为 2 级 (较高级)

11: SPI 中断优先级为 3 级 (最高级)

PPWMAH,PPWMA: 高级PWMA中断优先级控制位

00: 高级 PWMA 中断优先级为 0 级 (最低级)

01: 高级 PWMA 中断优先级为 1 级 (较低级)

10: 高级 PWMA 中断优先级为 2 级 (较高级)

11: 高级 PWMA 中断优先级为 3 级 (最高级)

PPWMBH,PPWMB: 高级PWMB中断优先级控制位

00: 高级 PWMB 中断优先级为 0 级 (最低级)

01: 高级 PWMB 中断优先级为 1 级 (较低级)

10: 高级 PWMB 中断优先级为 2 级 (较高级)

11: 高级 PWMB 中断优先级为 3 级 (最高级)

PX4H,PX4: 外部中断4中断优先级控制位

00: INT4 中断优先级为 0 级 (最低级)

01: INT4 中断优先级为 1 级 (较低级)

10: INT4 中断优先级为 2 级 (较高级)

11: INT4 中断优先级为 3 级 (最高级)

PCMPH,PCMP: 比较器中断优先级控制位

00: CMP 中断优先级为 0 级 (最低级)

01: CMP 中断优先级为 1 级 (较低级)

10: CMP 中断优先级为 2 级 (较高级)

11: CMP 中断优先级为 3 级 (最高级)

PI2CH,PI2C: I2C中断优先级控制位

00: I2C 中断优先级为 0 级 (最低级)

01: I2C 中断优先级为 1 级 (较低级)

10: I2C 中断优先级为 2 级 (较高级)

11: I2C 中断优先级为 3 级 (最高级)

PUSBH,PUSB: USB中断优先级控制位

00: USB 中断优先级为 0 级 (最低级)

01: USB 中断优先级为 1 级 (较低级)

10: USB 中断优先级为 2 级 (较高级)

11: USB 中断优先级为 3 级 (最高级)

10.5 范例程序

10.5.1 INT0 中断（上升沿和下降沿），可同时支持上升沿和下降沿

C 语言代码

//测试工作频率为 11.0592MHz

#include "stc16.h"

//头文件见附录

#include "intrins.h"

void INT0_Isr() interrupt 0

{

if (INT0)

//判断上升沿和下降沿

{

P10 = !P10;

//测试端口

}

else

{

P11 = !P11;

//测试端口

}

}

void main()

{

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

EX0 = 1;

//使能 INT0 中断

EA = 1;

while (1);

}

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC)

;头文件见附录

```
ORG 0000H
LJMP MAIN
ORG 0003H
LJMP INT0ISR

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

RISING:
CPL P1.1 ;测试端口
RETI

MAIN:
MOV SP, #5FH
MOV 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 EX0 ;使能INT0 中断
SETB EA
JMP $

END
```

10.5.2 INT0 中断（下降沿）

C 语言代码

//测试工作频率为11.0592MHz

#include "stc16.h" //头文件见附录

#include "intrins.h"

void INT0_Isr() interrupt 0

```
{
    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 = 1;
EX0 = 1;
EA = 1;
```

```
//使能INT0 下降沿中断
//使能INT0 中断
```

```
while (1);
```

```
}
```

汇编代码

```
;测试工作频率为11.0592MHz
```

```
$include (STC16F.INC)
```

```
;头文件见附录
```

```
ORG 0000H
LJMP MAIN
ORG 0003H
LJMP INT0ISR
```

```
INT0ISR: ORG 0100H
```

```
CPL P1.0
RETI
```

```
;测试端口
```

```
MAIN:
```

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

```
SETB IT0
SETB EX0
SETB EA
JMP $
```

```
;使能INT0 下降沿中断
;使能INT0 中断
```

```
END
```

10.5.3 INT1 中断（上升沿和下降沿），可同时支持上升沿和下降沿

C 语言代码

//测试工作频率为11.0592MHz

#include "stc16.h"
#include "intrins.h"

//头文件见附录

void INT1_Isr() interrupt 2

```
{  
    if (INT1)                                //判断上升沿和下降沿  
    {  
        P10 = !P10;                          //测试端口  
    }  
    else  
    {  
        P11 = !P11;                          //测试端口  
    }  
}
```

void main()

```
{  
    P0M0 = 0x00;  
    P0M1 = 0x00;  
    P1M0 = 0x00;  
    P1M1 = 0x00;  
    P2M0 = 0x00;  
    P2M1 = 0x00;  
    P3M0 = 0x00;  
    P3M1 = 0x00;  
    P4M0 = 0x00;  
    P4M1 = 0x00;  
    P5M0 = 0x00;  
    P5M1 = 0x00;  
  
    IT1 = 0;                                //使能INT1 上升沿和下降沿中断  
    EX1 = 1;                                //使能INT1 中断  
    EA = 1;  
  
    while (1);  
}
```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC)

;头文件见附录

```
ORG    0000H  
LJMP   MAIN  
ORG    0013H  
LJMP   INT1ISR  
  
ORG    0100H
```

INT1ISR:

JB **INT1,RISING** ;判断上升沿和下降沿
CPL **P1.0** ;测试端口
RETI

RISING:

CPL **P1.1** ;测试端口
RETI

MAIN:

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

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

END

10.5.4 INT1 中断（下降沿）

C 语言代码

//测试工作频率为11.0592MHz

#include "stc16.h" //头文件见附录
#include "intrins.h"

void INT1_Isr() interrupt 2

{
 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;

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

while (1);
}
```

汇编代码

;测试工作频率为11.0592MHz

```
$include (STC16.INC)           ;头文件见附录

ORG      0000H
LJMP     MAIN
ORG      0013H
LJMP     INT1ISR

ORG      0100H
INT1ISR:
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

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

END
```

10.5.5 INT2 中断（下降沿），只支持下降沿中断

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"           //头文件见附录
#include "intrins.h"

#define EX2                   0x10
```

```
#define EX3 0x20
#define EX4 0x40

void INT2_Isr() interrupt 10
{
    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;

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

    while (1);
}
```

汇编代码

;测试工作频率为11.0592MHz

```
$include (STC16.INC) ;头文件见附录

EX2 EQU 10H
EX3 EQU 20H
EX4 EQU 40H

ORG 0000H
LJMP MAIN
ORG 0053H
LJMP INT2ISR

INT2ISR:
ORG 0100H
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    INTCLKO, #EX2    ;使能INT2 中断
SETB   EA
JMP    $

END
```

10.5.6 INT3 中断（下降沿），只支持下降沿中断

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"    //头文件见附录
#include "intrins.h"

#define EX2            0x10
#define EX3            0x20
#define EX4            0x40

void INT3_Isr() interrupt 11
{
    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;

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

    while (1);
}
```

汇编代码

;测试工作频率为11.0592MHz

```
$include (STC16.INC)    ;头文件见附录

EX2            EQU            10H
```

```
EX3      EQU      20H
EX4      EQU      40H

                ORG      0000H
                LJMP     MAIN
                ORG      005BH
                LJMP     INT3ISR

INT3ISR:      ORG      0100H

                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      INTCLK0, #EX3 ;使能INT3 中断
                SETB     EA
                JMP      $

                END
```

10.5.7 INT4 中断（下降沿），只支持下降沿中断

C 语言代码

```
//测试工作频率为11.0592MHz

#include "stc16.h"          //头文件见附录
#include "intrins.h"

#define EX2      0x10
#define EX3      0x20
#define EX4      0x40

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;
```

```
INTCLKO = EX4;
EA = 1;
```

```
//使能INT4 中断
```

```
while (1);
```

```
}
```

汇编代码

```
;测试工作频率为11.0592MHz
```

```
$include (STC16.INC)
```

```
;头文件见附录
```

```
EX2      EQU      10H
EX3      EQU      20H
EX4      EQU      40H
```

```
ORG      0000H
LJMP     MAIN
ORG      0083H
LJMP     INT4ISR
```

```
INT4ISR:  ORG      0100H
```

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

```
END
```

10.5.8 定时器 0 中断

C 语言代码

//测试工作频率为11.0592MHz

#include "stc16.h" //头文件见附录

#include "intrins.h"

void TM0_Isr() interrupt 1

{

 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;

 TMOD = 0x00;

 TL0 = 0x66;

 //65536-11.0592M/12/1000

 TH0 = 0xfc;

 TR0 = 1;

 //启动定时器

 ET0 = 1;

 //使能定时器中断

 EA = 1;

 while (1);

}

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC)

;头文件见附录

ORG 0000H

LJMP MAIN

ORG 000BH

LJMP TM0ISR

ORG 0100H

TM0ISR:

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

JMP      $

END

```

10.5.9 定时器 1 中断

C 语言代码

//测试工作频率为 11.0592MHz

```

#include "stc16.h"           //头文件见附录
#include "intrins.h"

void TM1_Isr() interrupt 3
{
    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;

    TMOD = 0x00;
    TL1 = 0x66;              //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                 //启动定时器
    ET1 = 1;                 //使能定时器中断
}

```

```
EA = 1;

while (1);
}
```

汇编代码

;测试工作频率为11.0592MHz

```
$include (STC16.INC) ;头文件见附录

ORG 0000H
LJMP MAIN
ORG 001BH
LJMP TMIISR

TMIISR:
ORG 0100H
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 TMOD, #00H
MOV TL1, #66H ;65536-11.0592M/12/1000
MOV TH1, #0FCH
SETB TR1 ;启动定时器
SETB ET1 ;使能定时器中断
SETB EA

JMP $

END
```

10.5.10 定时器 2 中断

C 语言代码

```
//测试工作频率为11.0592MHz

#include "stc16.h" //头文件见附录
#include "intrins.h"

#define ET2 0x04
```

```
#define T2IF 0x01

void TM2_Isr() interrupt 12
{
    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;

    T2L = 0x66; //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10; //启动定时器
    IE2 = ET2; //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC) ;头文件见附录

```
ET2 EQU 04H
T2IF EQU 01H

ORG 0000H
LJMP MAIN
ORG 0063H
LJMP TM2ISR

ORG 0100H
TM2ISR:
    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    T2L, #66H           ;65536-11.0592M/12/1000
MOV    T2H, #0FCH
MOV    AUXR, #10H         ;启动定时器
MOV    IE2, #ET2          ;使能定时器中断
SETB   EA

JMP    $

END
```

10.5.11 定时器 3 中断

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"          //头文件见附录
#include "intrins.h"
```

```
#define ET3          0x20
#define T3IF         0x02
```

```
void TM3_Isr() interrupt 19
{
    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;

    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
    EA = 1;

    while (1);
}
```


汇编代码

;测试工作频率为 11.0592MHz

```
$include (STC16.INC)           ;头文件见附录

ET3      EQU      20H
T3IF     EQU      02H

        ORG      0000H
        LJMP     MAIN
        ORG      009BH
        LJMP     TM3ISR

TM3ISR:   ORG      0100H

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

        JMP      $

END
```

10.5.12 定时器 4 中断

C 语言代码

//测试工作频率为 11.0592MHz

```
#include "stc16.h"           //头文件见附录
#include "intrins.h"

#define ET3      0x20
#define ET4      0x40
#define T3IF     0x02
#define T4IF     0x04
```

```

void TM4_Isr() interrupt 20
{
    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;

    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;

    while (1);
}

```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC) ;头文件见附录

```

ET3      EQU      20H
ET4      EQU      40H
T3IF     EQU      02H
T4IF     EQU      04H

```

```

ORG      0000H
LJMP     MAIN
ORG      00A3H
LJMP     TM4ISR

```

```

ORG      0100H
TM4ISR:

```

```

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

JMP      $

END

```

10.5.13 UART1 中断

C 语言代码

//测试工作频率为11.0592MHz

```

#include "stc16.h"
#include "intrins.h"

```

//头文件见附录

```
void UART1_Isr() interrupt 4
```

```

{
    if (TI)
    {
        TI = 0;           //清中断标志
        P10 = !P10;       //测试端口
    }
    if (RI)
    {
        RI = 0;           //清中断标志
        P11 = !P11;       //测试端口
    }
}

```

```
void main()
```

```

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

    SCON = 0x50;
    T2L = 0xe8;           //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
}

```

```
AUXR = 0x15;           //启动定时器
ES = 1;                 //使能串口中断
EA = 1;
SBUF = 0x5a;           //发送测试数据

while (1);
}
```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC)

;头文件见附录

```
ORG 0000H
LJMP MAIN
ORG 0023H
LJMP UART1ISR

ORG 0100H
UART1ISR:
JNB TI,CHECKRI
CLR TI           ;清中断标志
CPL P1.0        ;测试端口

CHECKRI:
JNB RI,ISREXIT
CLR RI          ;清中断标志
CPL P1.1        ;测试端口

ISREXIT:
RETI

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

MOV SCON,#50H
MOV T2L,#0E8H   ;65536-11059200/115200/4=0FFE8H
MOV T2H,#0FFH
MOV AUXR,#15H   ;启动定时器
SETB ES         ;使能串口中断
SETB EA
MOV SBUF,#5AH   ;发送测试数据

JMP $

END
```

10.5.14 UART2 中断

C 语言代码

//测试工作频率为11.0592MHz

#include "stc16.h" //头文件见附录

#include "intrins.h"

#define ES2 0x01

void UART2_Isr() interrupt 8

```
{
    if (S2CON & 0x02)
    {
        S2CON &= ~0x02; //清中断标志
        P12 = !P12; //测试端口
    }
    if (S2CON & 0x01)
    {
        S2CON &= ~0x01; //清中断标志
        P13 = !P13; //测试端口
    }
}
```

void main()

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

    S2CON = 0x10;
    T2L = 0xe8; //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x14; //启动定时器
    IE2 = ES2; //使能串口中断
    EA = 1;
    S2BUF = 0x5a; //发送测试数据

    while (1);
}
```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC) ;头文件见附录

ES2 EQU 01H

```

        ORG      0000H
        LJMP     MAIN
        ORG      0043H
        LJMP     UART2ISR

UART2ISR:
        ORG      0100H
        PUSH     ACC
        PUSH     PSW
        MOV       A,S2CON
        JNB       ACC.1,CHECKRI
        ANL       S2CON,#NOT 02H      ;清中断标志
        CPL       P1.2                ;测试端口
CHECKRI:
        MOV       A,S2CON
        JNB       ACC.0,ISREXIT
        ANL       S2CON,#NOT 01H      ;清中断标志
        CPL       P1.3                ;测试端口
ISREXIT:
        POP       PSW
        POP       ACC
        RETI

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

        MOV       S2CON,#10H
        MOV       T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
        MOV       T2H,#0FFH
        MOV       AUXR,#14H           ;启动定时器
        MOV       IE2,#ES2           ;使能串口中断
        SETB      EA
        MOV       S2BUF,#5AH         ;发送测试数据

        JMP       $

END

```

10.5.15 UART3 中断

C 语言代码

//测试工作频率为 11.0592MHz

#include "stc16.h"

//头文件见附录

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

```
#define ES3 0x08
```

```
void UART3_Isr() interrupt 17
```

```
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;           //清中断标志
        P12 = !P12;               //测试端口
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;           //清中断标志
        P13 = !P13;               //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;

    S3CON = 0x10;
    T2L = 0xe8;                //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x14;               //启动定时器
    IE2 = ES3;                 //使能串口中断
    EA = 1;
    S3BUF = 0x5a;              //发送测试数据

    while (1);
}
```

汇编代码

;测试工作频率为 11.0592MHz

```
$include (STC16.INC) ;头文件见附录
```

```
ES3 EQU 08H

ORG 0000H
LJMP MAIN
ORG 008BH
LJMP UART3ISR

ORG 0100H
UART3ISR:
```

```

        PUSH    ACC
        PUSH    PSW
        MOV     A,S3CON
        JNB     ACC.1,CHECKRI
        ANL     S3CON,#NOT 02H      ;清中断标志
        CPL     P1.2                ;测试端口
CHECKRI:
        MOV     A,S3CON
        JNB     ACC.0,ISREXIT
        ANL     S3CON,#NOT 01H      ;清中断标志
        CPL     P1.3                ;测试端口
ISREXIT:
        POP     PSW
        POP     ACC
        RETI

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

        MOV     S3CON,#10H
        MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
        MOV     T2H,#0FFH
        MOV     AUXR,#14H           ;启动定时器
        MOV     IE2,#ES3           ;使能串口中断
        SETB    EA
        MOV     S3BUF,#5AH         ;发送测试数据

        JMP     $

END

```

10.5.16 UART4 中断

C 语言代码

//测试工作频率为 11.0592MHz

```
#include "stc16.h"      //头文件见附录
#include "intrins.h"
```

```
#define ES4             0x10
```

```
void UART4_Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
```



```

        S4CON &= ~0x02;           //清中断标志
        P12 = !P12;               //测试端口
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;           //清中断标志
        P13 = !P13;               //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;

    S4CON = 0x10;
    T2L = 0xe8;                   //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x14;                 //启动定时器
    IE2 = ES4;                   //使能串口中断
    EA = 1;
    S4BUF = 0x5a;                //发送测试数据

    while (1);
}

```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC)

;头文件见附录

```

ES4      EQU      10H

          ORG      0000H
          LJMP     MAIN
          ORG      0093H
          LJMP     UART4ISR

          ORG      0100H
UART4ISR:
          PUSH     ACC
          PUSH     PSW
          MOV      A,S4CON
          JNB      ACC.1,CHECKRI
          ANL      S4CON,#NOT 02H      ;清中断标志
          CPL      P1.2                ;测试端口
CHECKRI:
          MOV      A,S4CON

```

```
ISREXIT:
    JNB     ACC.0,ISREXIT
    ANL     S4CON,#NOT 01H      ;清中断标志
    CPL     P1.3                ;测试端口

    POP     PSW
    POP     ACC
    RETI

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

    MOV     S4CON,#10H
    MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
    MOV     T2H,#0FFH
    MOV     AUXR,#14H          ;启动定时器
    MOV     IE2,#ES4           ;使能串口中断
    SETB    EA
    MOV     S4BUF,#5AH         ;发送测试数据

    JMP     $

END
```

10.5.17 ADC 中断

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"          //头文件见附录
#include "intrins.h"
```

```
void ADC_Isr() interrupt 5
{
    ADC_CONTR &= ~0x20;      //清中断标志
    P0 = ADC_RES;            //测试端口
    P2 = ADC_RES;            //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;

ADCCFG = 0x00;
ADC_CONTR = 0xc0;           //使能并启动ADC 模块
EADC = 1;                   //使能ADC 中断
EA = 1;

while (1);
```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC)

;头文件见附录

```
ORG      0000H
LJMP     MAIN
ORG      002BH
LJMP     ADCISR

ADCISR:  ORG      0100H
        ANL      ADC_CONTR,#NOT 20H    ;清中断标志
        MOV      P0,ADC_RES            ;测试端口
        MOV      P2,ADC_RES            ;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      ADCCFG,#00H
        MOV      ADC_CONTR,#0C0H      ;使能并启动ADC 模块
        SETB     EADC                  ;使能ADC 中断
        SETB     EA

        JMP      $

END
```

10.5.18 LVD 中断

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"           //头文件见附录
#include "intrins.h"

#define ENLVR                0x40           //RSTCFG.6
#define LVD2V2               0x00           //LVD@2.2V
#define LVD2V4               0x01           //LVD@2.4V
#define LVD2V7               0x02           //LVD@2.7V
#define LVD3V0               0x03           //LVD@3.0V

#define LVDF                  0x20           //PCON.5

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;

    while (1);
}
```

汇编代码

;测试工作频率为11.0592MHz

```
$include (STC16.INC)        ;头文件见附录

ENLVR    EQU    40H         ;RSTCFG.6
LVD2V2    EQU    00H         ;LVD@2.2V
LVD2V4    EQU    01H         ;LVD@2.4V
LVD2V7    EQU    02H         ;LVD@2.7V
LVD3V0    EQU    03H         ;LVD@3.0V
```

```
LVDF      EQU      20H      ;PCON.5

      ORG      0000H
      LJMP     MAIN
      ORG      0033H
      LJMP     LVDISR

LVDISR:    ORG      0100H

      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
      JMP      $

      END
```

10.5.19 比较器中断

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"      //头文件见附录
#include "intrins.h"
```

```
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; // P3.7 为 CMP+ 输入脚
CMPCR1 &= ~0x04; // 内部参考电压为 CMP- 输入脚
// CMPCR1 |= 0x04; // P3.6 为 CMP- 输入脚
CMPCR1 |= 0x02; // 使能比较器输出
EA = 1;

while (1);
}

```

汇编代码

; 测试工作频率为 11.0592MHz

\$include (STC16F.INC)

; 头文件见附录

```

ORG 0000H
LJMP MAIN
ORG 00ABH
LJMP CMPIR

CMPIR:
ORG 0100H
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 ; P3.7 为 CMP+ 输入脚
ANL CMPCR1, #NOT 04H ; 内部参考电压为 CMP- 输入脚
; ORL CMPCR1, #04H ; P3.6 为 CMP- 输入脚

```

ORL

SETB

JMP

END

CMPCRI,#02H

EA

\$

;使能比较器输出

10.5.20 SPI 中断

C 语言代码

//测试工作频率为11.0592MHz

#include "stc16.h" //头文件见附录

#include "intrins.h"

#define ESPI 0x02

void SPI_Isr() interrupt 9

{

SPSTAT = 0xc0; //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;

SPCTL = 0x50; //使能SPI 主机模式

SPSTAT = 0xc0; //清中断标志

IE2 = ESPI; //使能SPI 中断

EA = 1;

SPDAT = 0x5a; //发送测试数据

while (1);

}

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC) ;头文件见附录

ESPI EQU 02H

ORG 0000H

```
        LJMP      MAIN
        ORG       004BH
        LJMP      SPIISR

        ORG       0100H
SPIISR:  MOV       SPSTAT,#0C0H      ;清中断标志
        CPL       P1.0             ;测试端口
        RETI

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

        MOV       SPCTL,#50H      ;使能 SPI 主机模式
        MOV       SPSTAT,#0C0H   ;清中断标志
        MOV       IE2,#ESPI      ;使能 SPI 中断
        SETB      EA
        MOV       SPDAT,#5AH     ;发送测试数据

        JMP       $

        END
```

10.5.21 I2C 中断

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"          //头文件见附录
#include "intrins.h"
```

```
void I2C_Isr() interrupt 24
```

```
{
    char bak;
    bak = P_SW2;
    P_SW2 /= 0x80;
    if (I2CMSST & 0x40)
    {
        I2CMSST &= ~0x40;      //清中断标志
        P10 = !P10;           //测试端口
    }
    P_SW2 = bak;
}
```



```

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

    P_SW2 = 0x80;
    I2CCFG = 0xc0;           //使能 I2C 主机模式
    I2CMSCR = 0x80;         //使能 I2C 中断;
    P_SW2 = 0x00;
    EA = 1;

    P_SW2 = 0x80;
    I2CMSCR = 0x81;         //发送起始命令
    P_SW2 = 0x00;

    while (1);
}

```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC)

;头文件见附录

```

                ORG      0000H
                LJMP     MAIN
                ORG      00C3H
                LJMP     I2CISR

I2CISR:         ORG      0100H
                PUSH     ACC
                PUSH     DPL
                PUSH     DPH
                PUSH     P_SW2
                MOV       P_SW2,#80H
                MOV       WR6,#WORD0 I2CMSST
                MOV       WR4,#WORD2 I2CMSST
                MOV       R11,@DR4
                ANL       A,#NOT 40H           ;清中断标志
                MOV       @DR4,R11
                CPL       P1.0                ;测试端口
                POP       P_SW2
                POP       DPH
                POP       DPL
                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, #80H
MOV      A, #0C0H           ;使能 I2C 主机模式
MOV      WR6, #WORD0 I2CCFG
MOV      WR4, #WORD2 I2CCFG
MOV      @DR4, R11
MOV      A, #80H           ;使能 I2C 中断
MOV      WR6, #WORD0 I2CMSCR
MOV      WR4, #WORD2 I2CMSCR
MOV      @DR4, R11
MOV      P_SW2, #00H
SETB     EA

MOV      P_SW2, #80H
MOV      A, #081H          ;发送起始命令
MOV      WR6, #WORD0 I2CMSCR
MOV      WR4, #WORD2 I2CMSCR
MOV      @DR4, R11
MOV      P_SW2, #00H

JMP      $

END
```

11 定时器/计数器

STC16F 系列单片机内部设置了 5 个 16 位定时器/计数器。5 个 16 位定时器 T0、T1、T2、T3 和 T4 都具有计数方式和定时方式两种工作方式。对定时器/计数器 T0 和 T1，用它们在特殊功能寄存器 TMOD 中相对应的控制位 C/T 来选择 T0 或 T1 为定时器还是计数器。对定时器/计数器 T2，用特殊功能寄存器 AUXR 中的控制位 T2_C/T 来选择 T2 为定时器还是计数器。对定时器/计数器 T3，用特殊功能寄存器 T4T3M 中的控制位 T3_C/T 来选择 T3 为定时器还是计数器。对定时器/计数器 T4，用特殊功能寄存器 T4T3M 中的控制位 T4_C/T 来选择 T4 为定时器还是计数器。定时器/计数器的核心部件是一个加法计数器，其本质是对脉冲进行计数。只是计数脉冲来源不同：如果计数脉冲来自系统时钟，则为定时方式，此时定时器/计数器每 12 个时钟或者每 1 个时钟得到一个计数脉冲，计数值加 1；如果计数脉冲来自单片机外部引脚，则为计数方式，每来一个脉冲加 1。

当定时器/计数器 T0、T1 及 T2 工作在定时模式时，特殊功能寄存器 AUXR 中的 T0x12、T1x12 和 T2x12 分别决定是系统时钟/12 还是系统时钟/1（不分频）后让 T0、T1 和 T2 进行计数。当定时器/计数器 T3 和 T4 工作在定时模式时，特殊功能寄存器 T4T3M 中的 T3x12 和 T4x12 分别决定是系统时钟/12 还是系统时钟/1（不分频）后让 T3 和 T4 进行计数。当定时器/计数器工作在计数模式时，对外部脉冲计数不分频。

定时器/计数器 0 有 4 种工作模式：模式 0（16 位自动重装载模式），模式 1（16 位不可重装载模式），模式 2（8 位自动重装模式），模式 3（不可屏蔽中断的 16 位自动重装载模式）。定时器/计数器 1 除模式 3 外，其他工作模式与定时器/计数器 0 相同。T1 在模式 3 时无效，停止计数。定时器 T2 的工作模式固定为 16 位自动重装载模式。T2 可以当定时器使用，也可以当串口的波特率发生器和可编程时钟输出。定时器 3、定时器 4 与定时器 T2 一样，它们的工作模式固定为 16 位自动重装载模式。T3/T4 可以当定时器使用，也可以当串口的波特率发生器和可编程时钟输出。

11.1 定时器的相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
TMOD	定时器模式寄存器	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000,0000
TL0	定时器 0 低 8 位寄存器	8AH									0000,0000
TL1	定时器 1 低 8 位寄存器	8BH									0000,0000
TH0	定时器 0 高 8 位寄存器	8CH									0000,0000
TH1	定时器 1 高 8 位寄存器	8DH									0000,0000
AUXR	辅助寄存器 1	93H	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
INTCLKO	中断与时钟输出控制寄存器	95H	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
WKTCL	掉电唤醒定时器低字节	AAH									1111,1111
WKTCH	掉电唤醒定时器高字节	ABH	WKTEN								0111,1111
T4T3M	定时器 4/3 控制寄存器	DBH	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000,0000
T4H	定时器 4 高字节	C9H									0000,0000
T4L	定时器 4 低字节	CAH									0000,0000
T3H	定时器 3 高字节	CBH									0000,0000
T3L	定时器 3 低字节	CCH									0000,0000
T2H	定时器 2 高字节	CDH									0000,0000
T2L	定时器 2 低字节	CEH									0000,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
TM2PS	定时器 2 时钟预分频寄存器	7EFEA2H									0000,0000
TM3PS	定时器 3 时钟预分频寄存器	7EFEA3H									0000,0000
TM4PS	定时器 4 时钟预分频寄存器	7EFEA4H									0000,0000

11.2 定时器 0/1

11.2.1 定时器 0/1 控制寄存器 (TCON)

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

TF1: T1溢出中断标志。T1被允许计数以后,从初值开始加1计数。当产生溢出时由硬件将TF1位置“1”,并向CPU请求中断,一直保持到CPU响应中断时,才由硬件清“0”(也可由查询软件清“0”)。

TR1: 定时器T1的运行控制位。该位由软件置位和清零。当GATE (TMOD.7) =0, TR1=1时就允许T1开始计数, TR1=0时禁止T1计数。当GATE (TMOD.7) =1, TR1=1且INT1输入高电平时,才允许T1计数。

TF0: T0溢出中断标志。T0被允许计数以后,从初值开始加1计数,当产生溢出时,由硬件置“1”TF0,向CPU请求中断,一直保持CPU响应该中断时,才由硬件清0(也可由查询软件清0)。

TR0: 定时器T0的运行控制位。该位由软件置位和清零。当GATE (TMOD.3) =0, TR0=1时就允许T0开始计数, TR0=0时禁止T0计数。当GATE (TMOD.3) =1, TR0=1且INT0输入高电平时,才允许T0计数, TR0=0时禁止T0计数。

IE1: 外部中断1请求源 (INT1/P3.3) 标志。IE1=1, 外部中断向CPU请求中断,当CPU响应该中断时由硬件清“0”IE1。

IT1: 外部中断源1触发控制位。IT1=0, 上升沿或下降沿均可触发外部中断1。IT1=1, 外部中断1程控为下降沿触发方式。

IE0: 外部中断0请求源 (INT0/P3.2) 标志。IE0=1外部中断0向CPU请求中断,当CPU响应外部中断时,由硬件清“0”IE0(边沿触发方式)。

IT0: 外部中断源0触发控制位。IT0=0, 上升沿或下降沿均可触发外部中断0。IT0=1, 外部中断0程控为下降沿触发方式。

11.2.2 定时器 0/1 模式寄存器 (TMOD)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TMOD	89H	T1_GATE	T1_C/T	T1_M1	T1_M0	T0_GATE	T0_C/T	T0_M1	T0_M0

T1_GATE: 控制定时器1, 置1时只有在INT1脚为高及TR1控制位置1时才可打开定时器/计数器1。

T0_GATE: 控制定时器0, 置1时只有在INT0脚为高及TR0控制位置1时才可打开定时器/计数器0。

T1_C/T: 控制定时器1用作定时器或计数器, 清0则用作定时器(对内部系统时钟进行计数), 置1用作计数器(对引脚T1/P3.5外部脉冲进行计数)。

T0_C/T: 控制定时器0用作定时器或计数器, 清0则用作定时器(对内部系统时钟进行计数), 置1用作计数器(对引脚T0/P3.4外部脉冲进行计数)。

T1_M1/T1_M0: 定时器定时器/计数器1模式选择

T1_M1	T1_M0	定时器/计数器1工作模式
0	0	16位自动重载模式 当[TH1,TL1]中的16位计数值溢出时,系统会自动将内部16位重载寄存器中的重载值装入[TH1,TL1]中。

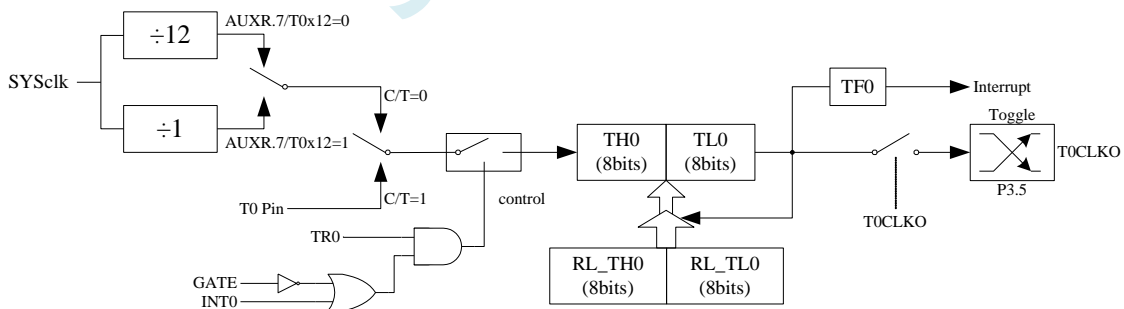
0	1	16位不自动重载模式 当[TH1,TL1]中的16位计数值溢出时，定时器1将从0开始计数
1	0	8位自动重载模式 当TL1中的8位计数值溢出时，系统会自动将TH1中的重载值装入TL1中。
1	1	T1停止工作

T0_M1/T0_M0: 定时器/计数器0模式选择

T0_M1	T0_M0	定时器/计数器0工作模式
0	0	16位自动重载模式 当[TH0,TL0]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[TH0,TL0]中。
0	1	16位不自动重载模式 当[TH0,TL0]中的16位计数值溢出时，定时器0将从0开始计数
1	0	8位自动重载模式 当TL0中的8位计数值溢出时，系统会自动将TH0中的重载值装入TL0中。
1	1	不可屏蔽中断的16位自动重载模式 与模式0相同，不可屏蔽中断，中断优先级最高，高于其他所有中断的优先级，并且不可关闭，可用作操作系统的系统节拍定时器，或者系统监控定时器。

11.2.3 定时器 0 模式 0（16 位自动重载模式）

此模式下定时器/计数器 0 作为可自动重载的 16 位计数器，如下图所示：



定时器/计数器 0 的模式 0：16 位自动重载模式

当 GATE=0 (TMOD.3) 时，如 TR0=1，则定时器计数。GATE=1 时，允许由外部输入 INT0 控制定时器 0，这样可实现脉宽测量。TR0 为 TCON 寄存器内的控制位，TCON 寄存器各位的具体功能描述见上节 TCON 寄存器的介绍。

当 C/T=0 时，多路开关连接到系统时钟的分频输出，T0 对内部系统时钟计数，T0 工作在定时方式。当 C/T=1 时，多路开关连接到外部脉冲输入 P3.4/T0，即 T0 工作在计数方式。

STC 单片机的定时器 0 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T0 的速率由特殊功能寄存器 AUXR 中的 T0x12 决定，如果 T0x12=0，T0 则工作在 12T 模式；如果 T0x12=1，T0 则工作在 1T 模式

定时器 0 有两个隐藏的寄存器 RL_TH0 和 RL_TL0。RL_TH0 与 TH0 共有同一个地址，RL_TL0 与 TL0 共有同一个地址。当 TR0=0 即定时器/计数器 0 被禁止工作时，对 TL0 写入的内容会同时写入 RL_TL0，对 TH0 写入的内容也会同时写入 RL_TH0。当 TR0=1 即定时器/计数器 0 被允许工作时，对 TL0 写入内容，实际上不是写入当前寄存器 TL0 中，而是写入隐藏的寄存器 RL_TL0 中，对 TH0 写入内容，实际上也不是写入当前寄存器 TH0 中，而是写入隐藏的寄存器 RL_TH0，这样可以巧妙地实现 16 位重装定时器。当读 TH0 和 TL0 的内容时，所读的内容就是 TH0 和 TL0 的内容，而不是 RL_TH0 和 RL_TL0 的内容。

当定时器 0 工作在模式 0（TMOD[1:0]/[M1,M0]=00B）时，[TH0,TL0]的溢出不仅置位 TF0，而且会自动将[RL_TH0,RL_TL0]的内容重新装入[TH0,TL0]。

当 T0CLKO/INT_CLKO.0=1 时，P3.5/T1 管脚配置为定时器 0 的时钟输出 T0CLKO。输出时钟频率为 $T0 \text{ 溢出率}/2$ 。

如果 C/T=0，定时器/计数器 T0 对内部系统时钟计数，则：

T0 工作在 1T 模式（AUXR.7/T0x12=1）时的输出时钟频率 = $(SYSclk)/(65536-[RL_TH0, RL_TL0])/2$

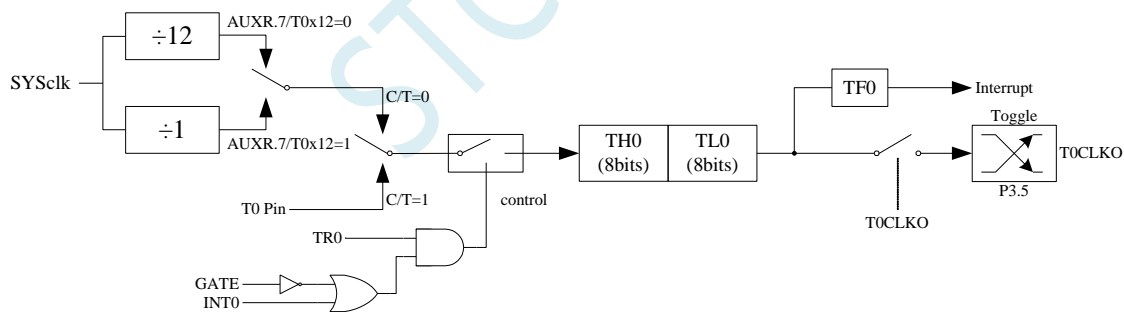
T0 工作在 12T 模式（AUXR.7/T0x12=0）时的输出时钟频率 = $(SYSclk)/12/(65536-[RL_TH0, RL_TL0])/2$

如果 C/T=1，定时器/计数器 T0 是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = $(T0_Pin_CLK) / (65536-[RL_TH0, RL_TL0])/2$

11.2.4 定时器 0 模式 1（16 位不可重载模式）

此模式下定时器/计数器 0 工作在 16 位不可重载模式，如下图所示：



定时器/计数器 0 的模式 1：16 位不可重载模式

此模式下，定时器/计数器 0 配置为 16 位不可重载模式，由 TL0 的 8 位和 TH0 的 8 位所构成。TL0 的 8 位溢出向 TH0 进位，TH0 计数溢出置位 TCON 中的溢出标志位 TF0。

当 GATE=0(TM0D.3)时，如 TR0=1，则定时器计数。GATE=1 时，允许由外部输入 INT0 控制定时器 0，这样可实现脉宽测量。TR0 为 TCON 寄存器内的控制位，TCON 寄存器各位的具体功能描述见上节 TCON 寄存器的介绍。

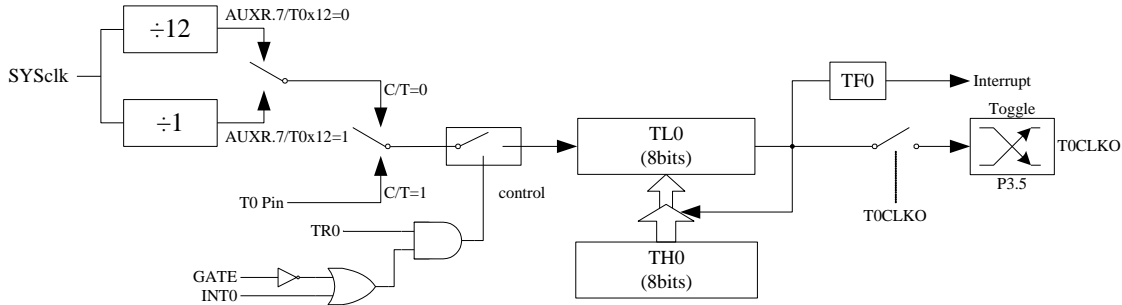
当 C/T=0 时，多路开关连接到系统时钟的分频输出，T0 对内部系统时钟计数，T0 工作在定时方式。当 C/T=1 时，多路开关连接到外部脉冲输入 P3.4/T0，即 T0 工作在计数方式。

STC 单片机的定时器 0 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；

另外一种 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T0 的速率由特殊功能寄存器 AUXR 中的 T0x12 决定，如果 T0x12=0，T0 则工作在 12T 模式；如果 T0x12=1，T0 则工作在 1T 模式。

11.2.5 定时器 0 模式 2（8 位自动重载模式）

此模式下定时器/计数器 0 作为可自动重载的 8 位计数器，如下图所示：



定时器/计数器 0 的模式 2：8 位自动重载模式

TL0 的溢出不仅置位 TF0，而且将 TH0 的内容重新装入 TL0，TH0 内容由软件预置，重装时 TH0 内容不变。

当 T0CLKO/INT_CLKO.0=1 时，P3.5/T1 管脚配置为定时器 0 的时钟输出 T0CLKO。输出时钟频率为 $T0 \text{ 溢出率}/2$ 。

如果 C/T=0，定时器/计数器 T0 对内部系统时钟计数，则：

T0 工作在 1T 模式（AUXR.7/T0x12=1）时的输出时钟频率 = $(SYSclk)/(256-TH0)/2$

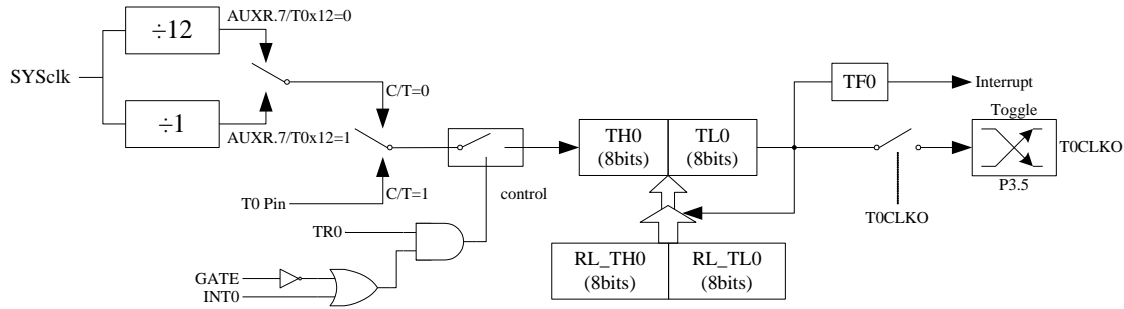
T0 工作在 12T 模式（AUXR.7/T0x12=0）时的输出时钟频率 = $(SYSclk)/12/(256-TH0)/2$

如果 C/T=1，定时器/计数器 T0 是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = $(T0_Pin_CLK) / (256-TH0)/2$

11.2.6 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）

对定时器/计数器 0，其工作模式模式 3 与工作模式 0 是一样的（下图定时器模式 3 的原理图，与工作模式 0 是一样的）。唯一不同的是：当定时器/计数器 0 工作在模式 3 时，只需允许 ET0/IE.1(定时器/计数器 0 中断允许位)，不需要允许 EA/IE.7(总中断使能位)就能打开定时器/计数器 0 的中断，此模式下的定时器/计数器 0 中断与总中断使能位 EA 无关，一旦工作在模式 3 下的定时器/计数器 0 中断被打开(ET0=1)，那么该中断是不可屏蔽的，该中断的优先级是最高的，即该中断不能被任何中断所打断，而且该中断打开后既不受 EA/IE.7 控制也不再受 ET0 控制，当 EA=0 或 ET0=0 时都不能屏蔽此中断。故将此模式称为不可屏蔽中断的 16 位自动重载模式。

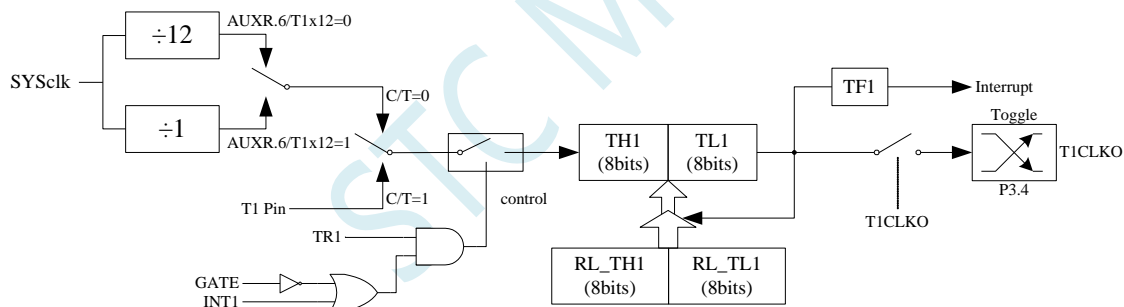


定时器/计数器 0 的模式 3：不可屏蔽中断的 16 位自动重载模式

注意：当定时器/计数器 0 工作在模式 3(不可屏蔽中断的 16 位自动重载模式)时，不需要允许 EA/IE.7(总中断使能位)，只需允许 ET0/IE.1(定时器/计数器 0 中断允许位)就能打开定时器/计数器 0 的中断，此模式下的定时器/计数器 0 中断与总中断使能位 EA 无关。一旦此模式下的定时器/计数器 0 中断被打开后，该定时器/计数器 0 中断优先级就是最高的，它不能被其它任何中断所打断(不管是比定时器/计数器 0 中断优先级低的中断还是比其优先级高的中断，都不能打断此时的定时器/计数器 0 中断)，而且该中断打开后既不受 EA/IE.7 控制也不再受 ET0 控制了，清零 EA 或 ET0 都不能关闭此中断。

11.2.7 定时器 1 模式 0（16 位自动重载模式）

此模式下定时器/计数器 1 作为可自动重载的 16 位计数器，如下图所示：



定时器/计数器 1 的模式 0：16 位自动重载模式

当 GATE=0 (TMOD.7) 时，如 TR1=1，则定时器计数。GATE=1 时，允许由外部输入 INT1 控制定时器 1，这样可实现脉宽测量。TR1 为 TCON 寄存器内的控制位，TCON 寄存器各位的具体功能描述见上节 TCON 寄存器的介绍。

当 C/T=0 时，多路开关连接到系统时钟的分频输出，T1 对内部系统时钟计数，T1 工作在定时方式。当 C/T=1 时，多路开关连接到外部脉冲输入 P3.5/T1，即 T1 工作在计数方式。

STC 单片机的定时器 1 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T1 的速率由特殊功能寄存器 AUXR 中的 T1x12 决定，如果 T1x12=0，T1 则工作在 12T 模式；如果 T1x12=1，T1 则工作在 1T 模式。

定时器 1 有两个隐藏的寄存器 RL_TH1 和 RL_TL1。RL_TH1 与 TH1 共有同一个地址，RL_TL1 与 TL1 共有同一个地址。当 TR1=0 即定时器/计数器 1 被禁止工作时，对 TL1 写入的内容会同时写入 RL_TL1，对 TH1 写入的内容也会同时写入 RL_TH1。当 TR1=1 即定时器/计数器 1 被允许工作时，对 TL1 写入内容，实

实际上不是写入当前寄存器 TH1 中, 而是写入隐藏的寄存器 RL_TH1 中, 对 TH1 写入内容, 实际上也不是写入当前寄存器 TH1 中, 而是写入隐藏的寄存器 RL_TH1, 这样可以巧妙地实现 16 位重载定时器。当读 TH1 和 TL1 的内容时, 所读的内容就是 TH1 和 TL1 的内容, 而不是 RL_TH1 和 RL_TL1 的内容。

当定时器 1 工作在模式 1 (TMOD[5:4]/[M1,M0]=00B) 时, [TH1,TL1]的溢出不仅置位 TF1, 而且会自动将[RL_TH1,RL_TL1]的内容重新装入[TH1,TL1]。

当 T1CLKO/INT_CLKO.1=1 时, P3.4/T0 管脚配置为定时器 1 的时钟输出 T1CLKO。输出时钟频率为 **T1 溢出率/2**。

如果 C/T=0, 定时器/计数器 T1 对内部系统时钟计数, 则:

T1 工作在 1T 模式 (AUXR.6/T1x12=1) 时的输出时钟频率 = (SYSclk)/(65536-[RL_TH1, RL_TL1])/2

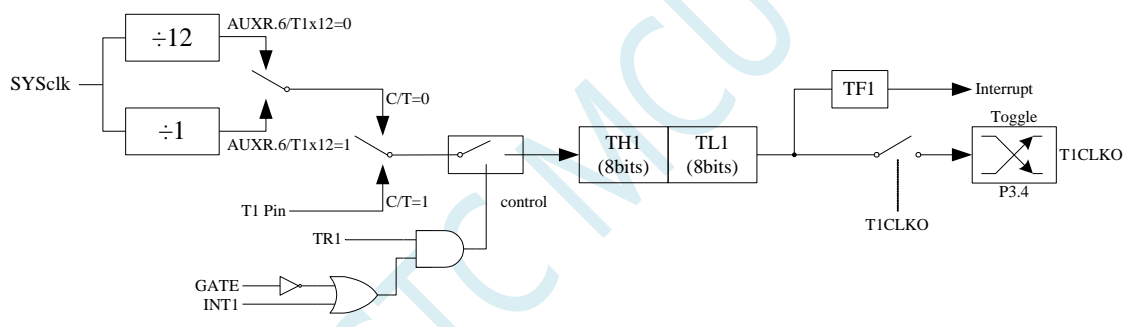
T1 工作在 12T 模式 (AUXR.6/T1x12=0) 时的输出时钟频率 = (SYSclk)/12/(65536-[RL_TH1, RL_TL1])/2

如果 C/T=1, 定时器/计数器 T1 是对外部脉冲输入(P3.5/T1)计数, 则:

输出时钟频率 = (T1_Pin_CLK) / (65536-[RL_TH1, RL_TL1])/2

11.2.8 定时器 1 模式 1 (16 位不可重载模式)

此模式下定时器/计数器 1 工作在 16 位不可重载模式, 如下图所示:



定时器/计数器 1 的模式 1: 16 位不可重载模式

此模式下, 定时器/计数器 1 配置为 16 位不可重载模式, 由 TL1 的 8 位和 TH1 的 8 位所构成。TL1 的 8 位溢出向 TH1 进位, TH1 计数溢出置位 TCON 中的溢出标志位 TF1。

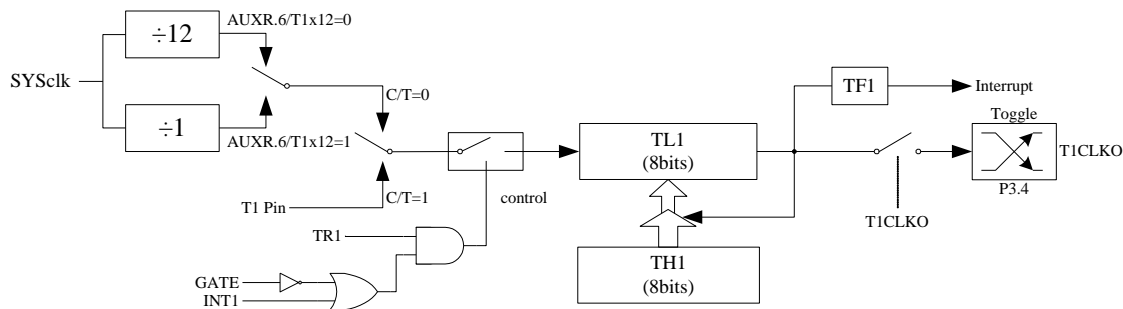
当 GATE=0(TM0D.7)时, 如 TR1=1, 则定时器计数。GATE=1 时, 允许由外部输入 INT1 控制定时器 1, 这样可实现脉宽测量。TR1 为 TCON 寄存器内的控制位, TCON 寄存器各位的具体功能描述见上节 TCON 寄存器的介绍。

当 C/T=0 时, 多路开关连接到系统时钟的分频输出, T1 对内部系统时钟计数, T1 工作在定时方式。当 C/T=1 时, 多路开关连接到外部脉冲输入 P3.5/T1, 即 T1 工作在计数方式。

STC 单片机的定时器 1 有两种计数速率: 一种是 12T 模式, 每 12 个时钟加 1, 与传统 8051 单片机相同; 另外一种 1T 模式, 每个时钟加 1, 速度是传统 8051 单片机的 12 倍。T1 的速率由特殊功能寄存器 AUXR 中的 T1x12 决定, 如果 T1x12=0, T1 则工作在 12T 模式; 如果 T1x12=1, T1 则工作在 1T 模式。

11.2.9 定时器 1 模式 2 (8 位自动重载模式)

此模式下定时器/计数器 1 作为可自动重载的 8 位计数器, 如下图所示:



定时器/计数器 1 的模式 2：8 位自动重载模式

TL1 的溢出不仅置位 TF1，而且将 TH1 的内容重新装入 TL1，TH1 内容由软件预置，重装时 TH1 内容不变。

当 T1CLKO/INT_CLKO.1=1 时，P3.4/T0 管脚配置为定时器 1 的时钟输出 T1CLKO。输出时钟频率为 **T1 溢出率/2**。

如果 C/T=0，定时器/计数器 T1 对内部系统时钟计数，则：

T1 工作在 1T 模式（AUXR.6/T1x12=1）时的输出时钟频率 = $(\text{SYSclk}) / (256 - \text{TH1}) / 2$

T1 工作在 12T 模式（AUXR.6/T1x12=0）时的输出时钟频率 = $(\text{SYSclk}) / 12 / (256 - \text{TH1}) / 2$

如果 C/T=1，定时器/计数器 T1 是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = $(\text{T1_Pin_CLK}) / (256 - \text{TH1}) / 2$

11.2.10 定时器 0 计数寄存器（TL0，TH0）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TL0	8AH								
TH0	8CH								

当定时器/计数器0工作在16位模式（模式0、模式1、模式3）时，TL0和TH0组合成为一个16位寄存器，

TL0为低字节，TH0为高字节。若为8位模式（模式2）时，TL0和TH0为两个独立的8位寄存器。

11.2.11 定时器 1 计数寄存器（TL1，TH1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TL1	8BH								
TH1	8DH								

当定时器/计数器1工作在16位模式（模式0、模式1）时，TL1和TH1组合成为一个16位寄存器，TL1为低

字节，TH1为高字节。若为8位模式（模式2）时，TL1和TH1为两个独立的8位寄存器。

11.2.12 辅助寄存器 1（AUXR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	93H	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

T0x12：定时器0速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1: 1T 模式, 即 CPU 时钟不分频分频 (FOSC/1)

T1x12: 定时器1速度控制位

0: 12T 模式, 即 CPU 时钟 12 分频 (FOSC/12)

1: 1T 模式, 即 CPU 时钟不分频分频 (FOSC/1)

11.2.13 中断与时钟输出控制寄存器 (INTCLKO)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTCLKO	95H	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

T0CLKO: 定时器0时钟输出控制

0: 关闭时钟输出

1: 使能 P3.5 口的是定时器 0 时钟输出功能

当定时器 0 计数发生溢出时, P3.5 口的电平自动发生翻转。

T1CLKO: 定时器1时钟输出控制

0: 关闭时钟输出

1: 使能 P3.4 口的是定时器 1 时钟输出功能

当定时器 1 计数发生溢出时, P3.4 口的电平自动发生翻转。

11.2.14 定时器 0 计算公式

定时器模式	定时器速度	周期计算公式
模式0/3 (16位自动重载)	1T	定时器周期 = $\frac{65536 - [TH0, TL0]}{SYSclk}$ (自动重载)
	12T	定时器周期 = $\frac{65536 - [TH0, TL0]}{SYSclk} \times 12$ (自动重载)
模式1 (16位不自动重载)	1T	定时器周期 = $\frac{65536 - [TH0, TL0]}{SYSclk}$ (需软件装载)
	12T	定时器周期 = $\frac{65536 - [TH0, TL0]}{SYSclk} \times 12$ (需软件装载)
模式2 (8位自动重载)	1T	定时器周期 = $\frac{256 - TH0}{SYSclk}$ (自动重载)
	12T	定时器周期 = $\frac{256 - TH0}{SYSclk} \times 12$ (自动重载)

11.2.15 定时器 1 计算公式

定时器模式	定时器速度	周期计算公式
模式0 (16位自动重载)	1T	定时器周期 = $\frac{65536 - [TH1, TL1]}{SYSclk}$ (自动重载)
	12T	定时器周期 = $\frac{65536 - [TH1, TL1]}{SYSclk} \times 12$ (自动重载)
模式1 (16位不自动重载)	1T	定时器周期 = $\frac{65536 - [TH1, TL1]}{SYSclk}$ (需软件装载)
	12T	定时器周期 = $\frac{65536 - [TH1, TL1]}{SYSclk} \times 12$ (需软件装载)
模式2 (8位自动重载)	1T	定时器周期 = $\frac{256 - TH1}{SYSclk}$ (自动重载)
	12T	定时器周期 = $\frac{256 - TH1}{SYSclk} \times 12$ (自动重载)

11.3 定时器 2

11.3.1 辅助寄存器 1（AUXR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	93H	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

TR2: 定时器2的运行控制位

- 0: 定时器 2 停止计数
- 1: 定时器 2 开始计数

T2_C/T: 控制定时器0用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T2/P1.2外部脉冲进行计数）。

T2x12: 定时器2速度控制位

- 0: 12T 模式，即 CPU 时钟 12 分频（FOSC/12）
- 1: 1T 模式，即 CPU 时钟不分频分频（FOSC/1）

11.3.2 中断与时钟输出控制寄存器（INTCLKO）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTCLKO	95H	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

T2CLKO: 定时器2时钟输出控制

- 0: 关闭时钟输出
- 1: 使能 P1.3 口的是定时器 2 时钟输出功能
当定时器 2 计数发生溢出时，P1.3 口的电平自动发生翻转。

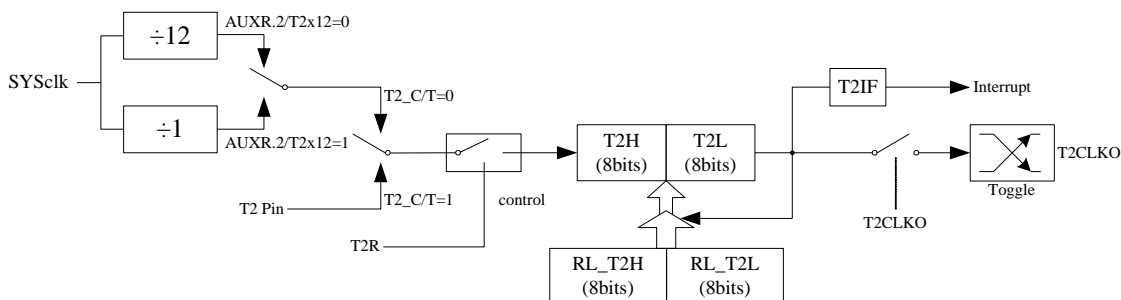
11.3.3 定时器 2 计数寄存器（T2L，T2H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T2L	CEH								
T2H	CDH								

定时器/计数器2的工作模式固定为16位重载模式，T2L和T2H组合成为一个16位寄存器，T2L为低字节，T2H为高字节。当[T2H,T2L]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[T2H,T2L]中。

11.3.4 定时器 2 工作模式

定时器/计数器2的原理框图如下:



定时器/计数器 2 的工作模式：16 位自动重载模式

T2R/AUXR.4 为 AUXR 寄存器内的控制位，AUXR 寄存器各位的具体功能描述见上节 AUXR 寄存器的介绍。

当 T2_C/T=0 时，多路开关连接到系统时钟输出，T2 对内部系统时钟计数，T2 工作在定时方式。当 T2_C/T=1 时，多路开关连接到外部脉冲输入 T2，即 T2 工作在计数方式。

STC 单片机的定时器 2 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T2 的速率由特殊功能寄存器 AUXR 中的 T2x12 决定，如果 T2x12=0，T2 则工作在 12T 模式；如果 T2x12=1，T2 则工作在 1T 模式。

定时器 2 有两个隐藏的寄存器 RL_T2H 和 RL_T2L。RL_T2H 与 T2H 共有同一个地址，RL_T2L 与 T2L 共有同一个地址。当 T2R=0 即定时器/计数器 2 被禁止工作时，对 T2L 写入的内容会同时写入 RL_T2L，对 T2H 写入的内容也会同时写入 RL_T2H。当 T2R=1 即定时器/计数器 2 被允许工作时，对 T2L 写入内容，实际上不是写入当前寄存器 T2L 中，而是写入隐藏的寄存器 RL_T2L 中，对 T2H 写入内容，实际上也不是写入当前寄存器 T2H 中，而是写入隐藏的寄存器 RL_T2H，这样可以巧妙地实现 16 位重载定时器。当读 T2H 和 T2L 的内容时，所读的内容就是 T2H 和 T2L 的内容，而不是 RL_T2H 和 RL_T2L 的内容。

[T2H,T2L]的溢出不仅置位中断请求标志位（T2IF），使 CPU 转去执行定时器 2 的中断程序，而且会自动将[RL_T2H,RL_T2L]的内容重新装入[T2H,T2L]。

11.3.5 定时器 2 计算公式

定时器速度	周期计算公式
1T	定时器周期 = $\frac{65536 - [T2H, T2L]}{SYSclk}$ (自动重载)
12T	定时器周期 = $\frac{65536 - [T2H, T2L]}{SYSclk} \times 12$ (自动重载)

11.4 定时器 3/4

11.4.1 定时器 3/4 功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T3T4PIN	7EFEACH	-	-	-	-	-	-	-	T3T4SEL

T3T4SEL: T3/T3CLKO/T4/T4CLKO 脚选择位

T3T4SEL	T3	T3CLKO	T4	T4CLKO
0	P0.4	P0.5	P0.6	P0.7
1	P0.0	P0.1	P0.2	P0.3

11.4.2 定时器 4/3 控制寄存器 (T4T3M)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	DBH	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

TR4: 定时器4的运行控制位

0: 定时器 4 停止计数

1: 定时器 4 开始计数

T4_C/T: 控制定时器4用作定时器或计数器, 清0则用作定时器 (对内部系统时钟进行计数), 置1用作计数器 (对引脚T4/P0.6外部脉冲进行计数)。

T4x12: 定时器4速度控制位

0: 12T 模式, 即 CPU 时钟 12 分频 (FOSC/12)

1: 1T 模式, 即 CPU 时钟不分频 (FOSC/1)

T4CLKO: 定时器4时钟输出控制

0: 关闭时钟输出

1: 使能 P0.7 口的是定时器 4 时钟输出功能

当定时器 4 计数发生溢出时, P0.7 口的电平自动发生翻转。

TR3: 定时器3的运行控制位

0: 定时器 3 停止计数

1: 定时器 3 开始计数

T3_C/T: 控制定时器3用作定时器或计数器, 清0则用作定时器 (对内部系统时钟进行计数), 置1用作计数器 (对引脚T3/P0.4外部脉冲进行计数)。

T3x12: 定时器3速度控制位

0: 12T 模式, 即 CPU 时钟 12 分频 (FOSC/12)

1: 1T 模式, 即 CPU 时钟不分频 (FOSC/1)

T3CLKO: 定时器3时钟输出控制

0: 关闭时钟输出

1: 使能 P0.5 口的是定时器 3 时钟输出功能

当定时器 3 计数发生溢出时, P0.5 口的电平自动发生翻转。

11.4.3 定时器 3 计数寄存器 (T3L, T3H)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T3L	CCH								

T3H	CBH	
-----	-----	--

定时器/计数器3的工作模式固定为16位重载模式，T3L和T3H组合成为一个16位寄存器，T3L为低字节，T3H为高字节。当[T3H,T3L]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[T3H,T3L]中。

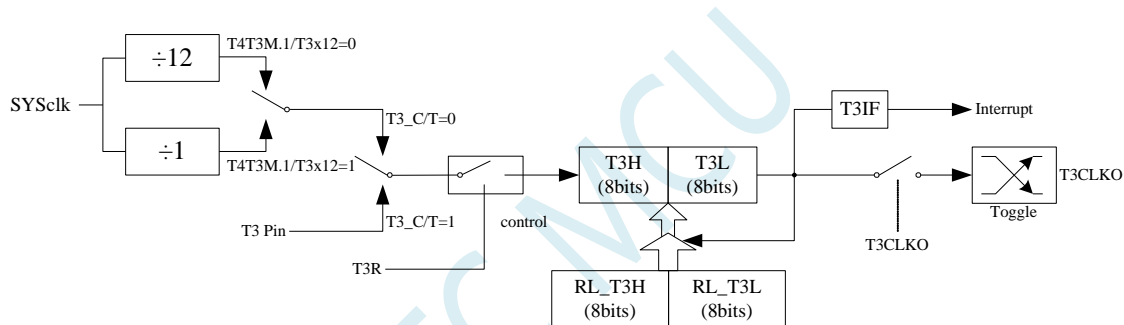
11.4.4 定时器 4 计数寄存器（T4L，T4H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T4L	CAH								
T4H	C9H								

定时器/计数器4的工作模式固定为16位重载模式，T4L和T4H组合成为一个16位寄存器，T4L为低字节，T4H为高字节。当[T4H,T4L]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[T4H,T4L]中。

11.4.5 定时器 3 工作模式

定时器/计数器3的原理框图如下：



定时器/计数器 3 的工作模式：16 位自动重载模式

T3R/T4T3M.3 为 T4T3M 寄存器内的控制位，T4T3M 寄存器各位的具体功能描述见上节 T4T3M 寄存器的介绍。

当 T3_C/T=0 时，多路开关连接到系统时钟输出，T3 对内部系统时钟计数，T3 工作在定时方式。当 T3_C/T=1 时，多路开关连接到外部脉冲输入 T3，即 T3 工作在计数方式。

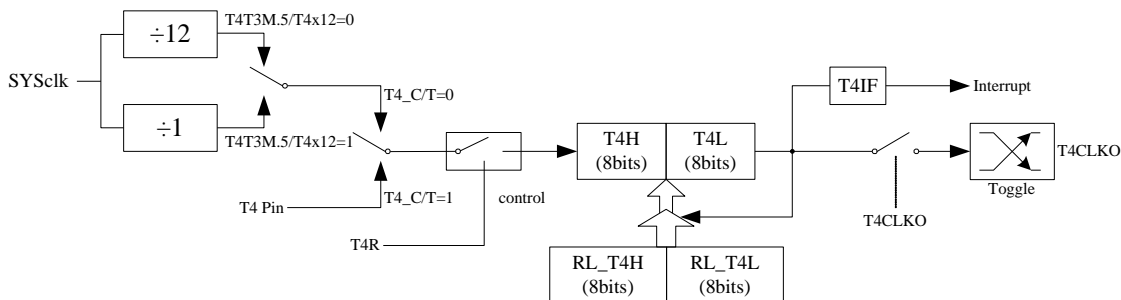
STC 单片机的定时器 3 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T3 的速率由特殊功能寄存器 T4T3M 中的 T3x12 决定，如果 T3x12=0，T3 则工作在 12T 模式；如果 T3x12=1，T3 则工作在 1T 模式。

定时器 3 有两个隐藏的寄存器 RL_T3H 和 RL_T3L。RL_T3H 与 T3H 共有同一个地址，RL_T3L 与 T3L 共有同一个地址。当 T3R=0 即定时器/计数器 3 被禁止工作时，对 T3L 写入的内容会同时写入 RL_T3L，对 T3H 写入的内容也会同时写入 RL_T3H。当 T3R=1 即定时器/计数器 3 被允许工作时，对 T3L 写入内容，实际上不是写入当前寄存器 T3L 中，而是写入隐藏的寄存器 RL_T3L 中，对 T3H 写入内容，实际上也不是写入当前寄存器 T3H 中，而是写入隐藏的寄存器 RL_T3H，这样可以巧妙地实现 16 位重载定时器。当读 T3H 和 T3L 的内容时，所读的内容就是 T3H 和 T3L 的内容，而不是 RL_T3H 和 RL_T3L 的内容。

[T3H,T3L]的溢出不仅置位中断请求标志位（T3IF），使 CPU 转去执行定时器 3 的中断程序，而且会自动将[RL_T3H,RL_T3L]的内容重新装入[T3H,T3L]。

11.4.6 定时器 4 工作模式

定时器/计数器 4 的原理框图如下:



定时器/计数器 4 的工作模式：16 位自动重载模式

T4R/T4T3M.7 为 T4T3M 寄存器内的控制位，T4T3M 寄存器各位的具体功能描述见上节 T4T3M 寄存器的介绍。

当 T4_C/T=0 时，多路开关连接到系统时钟输出，T4 对内部系统时钟计数，T4 工作在定时方式。当 T4_C/T=1 时，多路开关连接到外部脉冲输入 T4，即 T4 工作在计数方式。

STC 单片机的定时器 4 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T4 的速率由特殊功能寄存器 T4T3M 中的 T4x12 决定，如果 T4x12=0，T4 则工作在 12T 模式；如果 T4x12=1，T4 则工作在 1T 模式。

定时器 4 有两个隐藏的寄存器 RL_T4H 和 RL_T4L。RL_T4H 与 T4H 共有同一个地址，RL_T4L 与 T4L 共有同一个地址。当 T4R=0 即定时器/计数器 4 被禁止工作时，对 T4L 写入的内容会同时写入 RL_T4L，对 T4H 写入的内容也会同时写入 RL_T4H。当 T4R=1 即定时器/计数器 4 被允许工作时，对 T4L 写入内容，实际上不是写入当前寄存器 T4L 中，而是写入隐藏的寄存器 RL_T4L 中，对 T4H 写入内容，实际上也不是写入当前寄存器 T4H 中，而是写入隐藏的寄存器 RL_T4H，这样可以巧妙地实现 16 位重载定时器。当读 T4H 和 T4L 的内容时，所读的内容就是 T4H 和 T4L 的内容，而不是 RL_T4H 和 RL_T4L 的内容。

[T4H,T4L]的溢出不仅置位中断请求标志位 (T4IF)，使 CPU 转去执行定时器 4 的中断程序，而且会自动将[RL_T4H,RL_T4L]的内容重新装入[T4H,T4L]。

11.4.7 定时器 3 计算公式

定时器速度	周期计算公式
1T	定时器周期 = $\frac{65536 - [T3H, T3L]}{SYSclk}$ (自动重载)
12T	定时器周期 = $\frac{65536 - [T3H, T3L]}{SYSclk} \times 12$ (自动重载)

11.4.8 定时器 4 计算公式

定时器速度	周期计算公式
-------	--------

1T	定时器周期 = $\frac{65536 - [T4H, T4L]}{SYSclk}$ (自动重载)
12T	定时器周期 = $\frac{65536 - [T4H, T4L]}{SYSclk} \times 12$ (自动重载)

11.5 范例程序

11.5.1 定时器 0（模式 0—16 位自动重载），用作定时

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "stc16.h"           //头文件见附录
#include "intrins.h"

void TM0_Isr() interrupt 1
{
    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;

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

$include (STC16.INC)        ;头文件见附录

ORG 0000H
```

```

        LJMP      MAIN
        ORG       000BH
        LJMP      TM0ISR

TM0ISR:
        ORG       0100H
        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       TMOD, #00H      ;模式0
        MOV       TL0, #66H       ;65536-11.0592M/12/1000
        MOV       TH0, #0FCH
        SETB      TR0             ;启动定时器
        SETB      ET0             ;使能定时器中断
        SETB      EA

        JMP       $

END

```

11.5.2 定时器 0（模式 1—16 位不自动重载），用作定时

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"           //头文件见附录
#include "intrins.h"
```

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

```
{
    TL0 = 0x66;               //重设定时参数
    TH0 = 0xfc;
    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;

TMOD = 0x01;           //模式1
TL0 = 0x66;           //65536-11.0592M/12/1000
TH0 = 0xfc;
TR0 = 1;              //启动定时器
ET0 = 1;              //使能定时器中断
EA = 1;

while (1);
}

```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16F.INC)

;头文件见附录

```

ORG 0000H
LJMP MAIN
ORG 000BH
LJMP TM0ISR

TM0ISR:
ORG 0100H
MOV TL0,#66H           ;重设定参数
MOV TH0,#0FCH
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 TMOD,#01H          ;模式1
MOV TL0,#66H           ;65536-11.0592M/12/1000
MOV TH0,#0FCH
SETB TR0               ;启动定时器
SETB ET0               ;使能定时器中断
SETB EA

```

JMP *\$*

END

11.5.3 定时器 0（模式 2—8 位自动重载），用作定时

C 语言代码

//测试工作频率为 11.0592MHz

#include "stc16.h" //头文件见附录
#include "intrins.h"

void TM0_Isr() interrupt 1

{
 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;

 TMOD = 0x02; //模式 2
 TL0 = 0xf4; //256-11.0592M/12/76K
 TH0 = 0xf4;
 TR0 = 1; //启动定时器
 ET0 = 1; //使能定时器中断
 EA = 1;

 while (1);
}

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC) ;头文件见附录

 ORG 0000H
 LJMP MAIN
 ORG 000BH
 LJMP TM0ISR

 ORG 0100H
TM0ISR:
 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    TMOD, #02H           ;模式2
MOV    TL0, #0F4H           ;256-11.0592M/12/76K
MOV    TH0, #0F4H
SETB   TR0                  ;启动定时器
SETB   ET0                  ;使能定时器中断
SETB   EA

JMP     $

END
```

11.5.4 定时器 0（模式 3—16 位自动重载不可屏蔽中断），用作定时

C 语言代码

//测试工作频率为 11.0592MHz

```
#include "stc16.h"           //头文件见附录
#include "intrins.h"
```

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

```
{
    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;

TMOD = 0x03;           //模式3
TL0 = 0x66;           //65536-11.0592M/12/1000
TH0 = 0xfc;
TR0 = 1;               //启动定时器
ET0 = 1;               //使能定时器中断
// EA = 1;             //不受EA 控制

while (1);
}

```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16F.INC) ;头文件见附录

```

ORG 0000H
LJMP MAIN
ORG 000BH
LJMP TM0ISR

ORG 0100H
TM0ISR:
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 TMOD, #03H ;模式3
MOV TL0, #66H ;65536-11.0592M/12/1000
MOV TH0, #0FCH
SETB TR0 ;启动定时器
SETB ET0 ;使能定时器中断
; SETB EA ;不受EA 控制

JMP $

END

```

11.5.5 定时器 0（外部计数—扩展 T0 为外部下降沿中断）

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"           //头文件见附录
#include "intrins.h"

void TM0_Isr() interrupt 1
{
    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;

    TMOD = 0x04;              //外部计数模式
    TL0 = 0xff;
    TH0 = 0xff;
    TR0 = 1;                  //启动定时器
    ET0 = 1;                  //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

;测试工作频率为11.0592MHz

```
$include (STC16.INC)         ;头文件见附录

    ORG    0000H
    LJMP   MAIN
    ORG    000BH
    LJMP   TM0ISR

    ORG    0100H
TM0ISR:
    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      TMOD, #04H           ;外部计数模式
MOV      TL0, #0FFH
MOV      TH0, #0FFH
SETB     TR0                 ;启动定时器
SETB     ET0                 ;使能定时器中断
SETB     EA

JMP      $

END

```

11.5.6 定时器 0（测量脉宽—INT0 高电平宽度）

C 语言代码

//测试工作频率为11.0592MHz

```

#include "stc16.h"
#include "intrins.h"

```

//头文件见附录

```

void INT0_Isr() interrupt 0
{
    P0 = TL0;
    P1 = TH0;
}

```

//TL0 为测量值低字节
//TH0 为测量值高字节

```

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

```

```

    AUXR = 0x80;
    TMOD = 0x08;
    TL0 = 0x00;
    TH0 = 0x00;
    while (INT0);
    TR0 = 1;
    IT0 = 1;
    EX0 = 1;

```

//IT 模式
//使能 GATE, INT0 为 1 时使能计时

//等待 INT0 为低
//启动定时器
//使能 INT0 下降沿中断

```
EA = 1;

while (1);

}
```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC)

;头文件见附录

```
ORG 0000H
LJMP MAIN
ORG 0003H
LJMP INT0ISR

ORG 0100H
INT0ISR:
MOV P0,TL0 ;TL0 为测量值低字节
MOV P1,TH0 ;TH0 为测量值高字节
RETI

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

MOV AUXR,#80H ;IT 模式
MOV TMOD,#08H ;使能 GATE,INT0 为1 时使能计时
MOV TL0,#00H
MOV TH0,#00H
JB INT0,$ ;等待 INT0 为低
SETB TR0 ;启动定时器
SETB IT0 ;使能 INT0 下降沿中断
SETB EX0
SETB EA

JMP $

END
```

11.5.7 定时器 0（模式 0），时钟分频输出

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"           //头文件见附录
#include "intrins.h"

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

    TMOD = 0x00;              //模式0
    TL0 = 0x66;               //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                  //启动定时器
    INTCLKO = 0x01;           //使能时钟输出

    while (1);
}
```

汇编代码

;测试工作频率为 11.0592MHz

```
$include (STC16.INC)         ;头文件见附录

    ORG    0000H
    LJMP   MAIN

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

    MOV    TMOD, #00H        ;模式0
    MOV    TL0, #66H         ;65536-11.0592M/12/1000
    MOV    TH0, #0FCH
    SETB   TR0               ;启动定时器
    MOV    INTCLKO, #01H     ;使能时钟输出

    JMP    $
```

END

11.5.8 定时器 1（模式 0—16 位自动重载），用作定时

C 语言代码

//测试工作频率为 11.0592MHz

#include "stc16.h"

//头文件见附录

#include "intrins.h"

void TM1_Isr() interrupt 3

{

 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;

 TMOD = 0x00;

//模式 0

 TL1 = 0x66;

//65536-11.0592M/12/1000

 TH1 = 0xfc;

 TR1 = 1;

//启动定时器

 ET1 = 1;

//使能定时器中断

 EA = 1;

 while (1);

}

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC)

;头文件见附录

ORG 0000H

LJMP MAIN

ORG 001BH

LJMP TMIISR

ORG 0100H

TMIISR:

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

JMP    $

END
```

11.5.9 定时器 1（模式 1—16 位不自动重载），用作定时

C 语言代码

// 测试工作频率为 11.0592MHz

#include "stc16.h"

// 头文件见附录

#include "intrins.h"

void TM1_Isr() interrupt 3

```
{
    TL1 = 0x66;           // 重设定定时参数
    TH1 = 0xfc;
    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;
}
```

```
    TMOD = 0x10;                // 模式 1
    TL1 = 0x66;                 // 65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                   // 启动定时器
    ET1 = 1;                   // 使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

; 测试工作频率为 11.0592MHz

\$include (STC16.INC)

; 头文件见附录

```
    ORG      0000H
    LJMP     MAIN
    ORG      001BH
    LJMP     TMIISR

TMIISR:    ORG      0100H
    MOV      TL1, #66H          ; 重设定定时参数
    MOV      TH1, #0FCH
    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      TMOD, #10H        ; 模式 1
    MOV      TL1, #66H        ; 65536-11.0592M/12/1000
    MOV      TH1, #0FCH
    SETB     TR1              ; 启动定时器
    SETB     ET1              ; 使能定时器中断
    SETB     EA

    JMP      $

END
```

11.5.10 定时器 1（模式 2—8 位自动重载），用作定时

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"           //头文件见附录
#include "intrins.h"

void TM1_Isr() interrupt 3
{
    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;

    TMOD = 0x20;              //模式2
    TL1 = 0xf4;               //256-11.0592M/12/76K
    TH1 = 0xf4;
    TR1 = 1;                  //启动定时器
    ET1 = 1;                  //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

;测试工作频率为11.0592MHz

```
$include (STC16.INC)         ;头文件见附录

        ORG        0000H
        LJMP       MAIN
        ORG        001BH
        LJMP       TMIISR

        ORG        0100H
TMIISR:
        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      TMOD, #20H      ;模式2
MOV      TL1, #0F4H      ;256-11.0592M/12/76K
MOV      TH1, #0F4H
SETB     TR1              ;启动定时器
SETB     ET1              ;使能定时器中断
SETB     EA

JMP      $

END

```

11.5.11 定时器 1（外部计数—扩展 T1 为外部下降沿中断）

C 语言代码

//测试工作频率为 11.0592MHz

#include "stc16.h"

//头文件见附录

#include "intrins.h"

void TM1_Isr() interrupt 3

{

 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;

 TMOD = 0x40;

//外部计数模式

 TL1 = 0xff;

 TH1 = 0xff;

 TR1 = 1;

//启动定时器

 ET1 = 1;

//使能定时器中断

 EA = 1;

 while (1);

}

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC) ;头文件见附录

```
ORG 0000H
LJMP MAIN
ORG 001BH
LJMP TM1ISR

ORG 0100H
TM1ISR:
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 TMOD, #40H ;外部计数模式
MOV TL1, #0FFH
MOV TH1, #0FFH
SETB TR1 ;启动定时器
SETB ET1 ;使能定时器中断
SETB EA

JMP $

END
```

11.5.12 定时器 1（测量脉宽—INT1 高电平宽度）

C 语言代码

//测试工作频率为 11.0592MHz

#include "stc16.h" //头文件见附录

#include "intrins.h"

void INT1_Isr() interrupt 2

```
{
    P0 = TL1; ;TL1 为测量值低字节
    P1 = TH1; ;TH1 为测量值高字节
}
```

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

    AUXR = 0x40;                //IT 模式
    TMOD = 0x80;                //使能 GATE,INT1 为 1 时使能计时
    TL1 = 0x00;
    TH1 = 0x00;
    while (INT1);               //等待 INT1 为低
    TR1 = 1;                    //启动定时器
    IT1 = 1;                    //使能 INT1 下降沿中断
    EX1 = 1;
    EA = 1;

    while (1);
}
```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC)

;头文件见附录

```
                ORG      0000H
                LJMP     MAIN
                ORG      0013H
                LJMP     INT1ISR

                ORG      0100H
INT1ISR:
                MOV      P0,TL1      ;TL1 为测量值低字节
                MOV      P1,TH1      ;TH1 为测量值高字节
                RETI

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

```
MOV    P5M1, #00H

MOV    AUXR, #40H           ;IT 模式
MOV    TMOD, #80H          ;使能 GATE, INT1 为 1 时使能计时
MOV    TL1, #00H
MOV    TH1, #00H
JB     INT1, $              ;等待 INT1 为低
SETB   TR1                 ;启动定时器
SETB   IT1                 ;使能 INT1 下降沿中断
SETB   EX1
SETB   EA

JMP     $

END
```

11.5.13 定时器 1（模式 0），时钟分频输出

C 语言代码

//测试工作频率为 11.0592MHz

```
#include "stc16.h"
#include "intrins.h"
```

//头文件见附录

```
void main()
```

```
{
```

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

```
    TMOD = 0x00;
    TL1 = 0x66;
    TH1 = 0xfc;
    TR1 = 1;
    INTCLKO = 0x02;
```

//模式 0

//65536-11.0592M/12/1000

//启动定时器

//使能时钟输出

```
    while (1);
```

```
}
```

汇编代码

;测试工作频率为 11.0592MHz

```
$include (STC16.INC)
```

;头文件见附录

```
ORG    0000H
LJMP   MAIN
```

```

ORG          0100H

MAIN:

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

MOV          TMOD, #00H          ; 模式 0
MOV          TL1, #66H          ; 65536-11.0592M/12/1000
MOV          TH1, #0FCH
SETB         TR1              ; 启动定时器
MOV          INTCLKO, #02H      ; 使能时钟输出

JMP          $

END

```

11.5.14 定时器 1（模式 0）做串口 1 波特率发生器

C 语言代码

// 测试工作频率为 11.0592MHz

#include "stc16.h" // 头文件见附录

#include "intrins.h"

```

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

```

```

bit      busy;
char     wptr;
char     rptr;
char     buffer[16];

```

```

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

```

```
}

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

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

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}

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

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC)

;头文件见附录

BUSY BIT 20H.0
WPTR DATA 21H
RPTR DATA 22H
BUFFER DATA 23H ;16 bytes

ORG 0000H
LJMP MAIN
ORG 0023H
LJMP UART_ISR

ORG 0100H

UART_ISR:

PUSH ACC
PUSH PSW
MOV PSW,#08H

JNB TI,CHKRI
CLR TI
CLR BUSY

CHKRI:

JNB RI,UARTISR_EXIT
CLR RI
MOV A,WPTR
ANL A,#0FH
ADD A,#BUFFER
MOV R0,A
MOV @R0,SBUF
INC WPTR

UARTISR_EXIT:

POP PSW
POP ACC
RETI

UART_INIT:

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

UART_SEND:

JB BUSY,\$
SETB BUSY
MOV SBUF,A
RET

UART_SENDSTR:

```

CLR      A
MOVC     A,@A+DPTR
JZ       SENDEND
LCALL    UART_SEND
INC      DPTR
JMP      UART_SENDSTR

```

SENDEND:

```
RET
```

MAIN:

```

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

LCALL    UART_INIT
SETB     ES
SETB     EA

MOV      DPTR,#STRING
LCALL    UART_SENDSTR

```

LOOP:

```

MOV      A,RPTR
XRL      A,WPTR
ANL      A,#0FH
JZ       LOOP
MOV      A,RPTR
ANL      A,#0FH
ADD      A,#BUFFER
MOV      R0,A
MOV      A,@R0
LCALL    UART_SEND
INC      RPTR
JMP      LOOP

```

```
STRING:  DB      'Uart Test !',0DH,0AH,00H
```

```
END
```

11.5.15 定时器 1（模式 2）做串口 1 波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz
```

```
#include "stc16.h"
```

```
//头文件见附录
```



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

```
#define FOSC 11059200UL  
#define BRT (256 - FOSC / 115200 / 32)
```

```
bit busy;  
char wptr;  
char rptr;  
char buffer[16];
```

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

```
{  
    if (TI)  
    {  
        TI = 0;  
        busy = 0;  
    }  
    if (RI)  
    {  
        RI = 0;  
        buffer[wptr++] = SBUF;  
        wptr &= 0x0f;  
    }  
}
```

```
void UartInit()
```

```
{  
    SCON = 0x50;  
    TMOD = 0x20;  
    TL1 = BRT;  
    TH1 = BRT;  
    TRI = 1;  
    AUXR = 0x40;  
    wptr = 0x00;  
    rptr = 0x00;  
    busy = 0;  
}
```

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

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

```
void UartSendStr(char *p)
```

```
{  
    while (*p)  
    {  
        UartSend(*p++);  
    }  
}
```

```
void main()
```

```
{  
    P0M0 = 0x00;  
    P0M1 = 0x00;  
    P1M0 = 0x00;  
    P1M1 = 0x00;
```

```

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

UartInit();
ES = 1;
EA = 1;
UartSendStr("Uart Test !\r\n");

while (1)
{
    if (rptr != wptr)
    {
        UartSend(buffer[rptr++]);
        rptr &= 0x0f;
    }
}

```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC)

;头文件见附录

BUSY	BIT	20H.0	
WPTR	DATA	21H	
RPTR	DATA	22H	
BUFFER	DATA	23H	;16 bytes

```

ORG    0000H
LJMP   MAIN
ORG    0023H
LJMP   UART_ISR

ORG    0100H

```

UART_ISR:

```

PUSH    ACC
PUSH    PSW
MOV     PSW,#08H

JNB     TI,CHKRI
CLR     TI
CLR     BUSY

```

CHKRI:

```

JNB     RI,UARTISR_EXIT
CLR     RI
MOV     A,WPTR
ANL     A,#0FH
ADD     A,#BUFFER
MOV     R0,A
MOV     @R0,SBUF
INC     WPTR

```

UARTISR_EXIT:

POP PSW
POP ACC
RETI

UART_INIT:

MOV SCON,#50H
MOV TMOD,#20H
MOV TL1,#0FDH ;256-11059200/115200/32=0FDH
MOV TH1,#0FDH
SETB TR1
MOV AUXR,#40H
CLR BUSY
MOV WPTR,#00H
MOV RPTR,#00H
RET

UART_SEND:

JB BUSY,\$
SETB BUSY
MOV SBUF,A
RET

UART_SENDSTR:

CLR A
MOVC A,@A+DPTR
JZ SENDEND
LCALL UART_SEND
INC DPTR
JMP UART_SENDSTR

SENDEND:

RET

MAIN:

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

LCALL UART_INIT
SETB ES
SETB EA

MOV DPTR,#STRING
LCALL UART_SENDSTR

LOOP:

MOV A,RPTR
XRL A,WPTR
ANL A,#0FH

```
JZ      LOOP
MOV     A,RPTR
ANL     A,#0FH
ADD     A,#BUFFER
MOV     R0,A
MOV     A,@R0
LCALL   UART_SEND
INC     RPTR
JMP     LOOP

STRING:  DB      'Uart Test !',0DH,0AH,00H

END
```

11.5.16 定时器 2（16 位自动重载），用作定时

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"           //头文件见附录
#include "intrins.h"
```

```
#define ET2      0x04
#define T2IF     0x01
```

```
void TM2_Isr() interrupt 12
```

```
{
    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;

    T2L = 0x66;                //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;               //启动定时器
    IE2 = ET2;                 //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC)

;头文件见附录

ET2 EQU 04H
T2IF EQU 01H

ORG 0000H
LJMP MAIN
ORG 0063H
LJMP TM2ISR

TM2ISR: ORG 0100H

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

JMP \$

END

11.5.17 定时器 2（外部计数—扩展 T2 为外部下降沿中断）

C 语言代码

//测试工作频率为 11.0592MHz

#include "stc16.h" //头文件见附录
#include "intrins.h"

#define ET2 0x04
#define T2IF 0x01

void TM2_Isr() interrupt 12

{
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;

    T2L = 0xff;
    T2H = 0xff;
    AUXR = 0x18;           // 设置外部计数模式并启动定时器
    IE2 = ET2;             // 使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

#include (STC16.INC)      ;头文件见附录

ET2      EQU      04H
T2IF     EQU      01H

        ORG      0000H
        LJMP     MAIN
        ORG      0063H
        LJMP     TM2ISR

        ORG      0100H
TM2ISR:  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    T2L,#0FFH
MOV    T2H,#0FFH
MOV    AUXR,#18H      ;设置外部计数模式并启动定时器
MOV    IE2,#ET2       ;使能定时器中断
SETB   EA

JMP    $

END
```

11.5.18 定时器 2，时钟分频输出

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"      //头文件见附录
#include "intrins.h"
```

```
void main()
```

```
{
```

```
    P0M0 = 0x00;
```

```
    P0M1 = 0x00;
```

```
    P1M0 = 0x00;
```

```
    P1M1 = 0x00;
```

```
    P2M0 = 0x00;
```

```
    P2M1 = 0x00;
```

```
    P3M0 = 0x00;
```

```
    P3M1 = 0x00;
```

```
    P4M0 = 0x00;
```

```
    P4M1 = 0x00;
```

```
    P5M0 = 0x00;
```

```
    P5M1 = 0x00;
```

```
    T2L = 0x66;
```

```
//65536-11.0592M/12/1000
```

```
    T2H = 0xfc;
```

```
    AUXR = 0x10;
```

```
//启动定时器
```

```
    INTCLKO = 0x04;
```

```
//使能时钟输出
```

```
    while (1);
```

```
}
```

汇编代码

;测试工作频率为11.0592MHz

```
$include (STC16.INC)      ;头文件见附录
```

```
ORG    0000H
```

```
LJMP   MAIN
```

```
ORG    0100H
```

```
MAIN:
```

```
MOV    SP, #5FH
```

```
MOV    P0M0, #00H
```

```
MOV    P0M1, #00H
```

```

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

MOV      T2L, #66H           ;65536-11.0592M/12/1000
MOV      T2H, #0FCH
MOV      AUXR, #10H          ;启动定时器
MOV      INTCLKO, #04H       ;使能时钟输出

JMP      $

END

```

11.5.19 定时器 2 做串口 1 波特率发生器

C 语言代码

//测试工作频率为11.0592MHz

```

#include "stc16.h"           //头文件见附录
#include "intrins.h"

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

bit      busy;
char     wptr;
char     rptr;
char     buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    T2L = BRT;
    T2H = BRT >> 8;
}

```



```
AUXR = 0x15;
wptr = 0x00;
rptr = 0x00;
busy = 0;
}

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

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}

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

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16F.INC)

;头文件见附录

BUSY BIT 20H.0
WPTR DATA 21H

```

RPTR      DATA      22H
BUFFER    DATA      23H                                ;16 bytes

      ORG          0000H
      LJMP         MAIN
      ORG          0023H
      LJMP         UART_ISR

      ORG          0100H

UART_ISR:
      PUSH         ACC
      PUSH         PSW
      MOV          PSW,#08H

      JNB          TI,CHKRI
      CLR          TI
      CLR          BUSY

CHKRI:
      JNB          RI,UARTISR_EXIT
      CLR          RI
      MOV          A,WPTR
      ANL          A,#0FH
      ADD          A,#BUFFER
      MOV          R0,A
      MOV          @R0,SBUF
      INC          WPTR

UARTISR_EXIT:
      POP          PSW
      POP          ACC
      RETI

UART_INIT:
      MOV          SCON,#50H
      MOV          T2L,#0E8H                                ;65536-11059200/115200/4=0FFE8H
      MOV          T2H,#0FFH
      MOV          AUXR,#15H
      CLR          BUSY
      MOV          WPTR,#00H
      MOV          RPTR,#00H
      RET

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

UART_SENDSTR:
      CLR          A
      MOVC         A,@A+DPTR
      JZ           SENDEND
      LCALL        UART_SEND
      INC          DPTR
      JMP          UART_SENDSTR

SENDEND:
      RET

MAIN:

```

```

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

LCALL    UART_INIT
SETB     ES
SETB     EA

MOV      DPTR, #STRING
LCALL    UART_SENDSTR

```

LOOP:

```

MOV      A, RPTR
XRL      A, WPTR
ANL      A, #0FH
JZ       LOOP
MOV      A, RPTR
ANL      A, #0FH
ADD      A, #BUFFER
MOV      R0, A
MOV      A, @R0
LCALL    UART_SEND
INC      RPTR
JMP      LOOP

```

STRING: **DB** 'Uart Test !', 0DH, 0AH, 00H

END

11.5.20 定时器 2 做串口 2 波特率发生器

C 语言代码

//测试工作频率为 11.0592MHz

#include "stc16.h" //头文件见附录

#include "intrins.h"

```

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

```

```

bit      busy;
char     wptr;
char     rptr;
char     buffer[16];

```

void Uart2Isr() interrupt 8

```
{
    if (S2CON & 0x02)
    {
        S2CON &= ~0x02;
        busy = 0;
    }
    if (S2CON & 0x01)
    {
        S2CON &= ~0x01;
        buffer[wptr++] = S2BUF;
        wptr &= 0x0f;
    }
}

void Uart2Init()
{
    S2CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart2Send(char dat)
{
    while (busy);
    busy = 1;
    S2BUF = dat;
}

void Uart2SendStr(char *p)
{
    while (*p)
    {
        Uart2Send(*p++);
    }
}

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

    Uart2Init();
    IE2 = 0x01;
    EA = 1;
    Uart2SendStr("Uart Test !\r\n");
}
```

```

while (1)
{
    if (rptr != wptr)
    {
        Uart2Send(buffer[rptr++]);
        rptr &= 0x0f;
    }
}
}

```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC) ;头文件见附录

```

BUSY      BIT        20H.0
WPTR      DATA      21H
RPTR      DATA      22H
BUFFER    DATA      23H                ;16 bytes

```

```

ORG       0000H
LJMP      MAIN
ORG       0043H
LJMP      UART2_ISR

```

```
ORG       0100H
```

UART2_ISR:

```

PUSH      ACC
PUSH      PSW
MOV       PSW,#08H

MOV       A,S2CON
JNB       ACC.1,CHKRI
ANL       S2CON,#NOT 02H
CLR       BUSY

```

CHKRI:

```

JNB       ACC.0,UART2ISR_EXIT
ANL       S2CON,#NOT 01H
MOV       A,WPTR
ANL       A,#0FH
ADD       A,#BUFFER
MOV       R0,A
MOV       @R0,S2BUF
INC       WPTR

```

UART2ISR_EXIT:

```

POP       PSW
POP       ACC
RETI

```

UART2_INIT:

```

MOV       S2CON,#10H
MOV       T2L,#0E8H                ;65536-11059200/115200/4=0FFE8H
MOV       T2H,#0FFH
MOV       AUXR,#14H
CLR       BUSY
MOV       WPTR,#00H

```

```
        MOV        RPTR,#00H
        RET

UART2_SEND:
        JB         BUSY,$
        SETB       BUSY
        MOV        S2BUF,A
        RET

UART2_SENDSTR:
        CLR        A
        MOVC       A,@A+DPTR
        JZ         SEND2END
        LCALL      UART2_SEND
        INC        DPTR
        JMP        UART2_SENDSTR

SEND2END:
        RET

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

        LCALL      UART2_INIT
        MOV        IE2,#01H
        SETB       EA

        MOV        DPTR,#STRING
        LCALL      UART2_SENDSTR

LOOP:
        MOV        A,RPTR
        XRL        A,WPTR
        ANL        A,#0FH
        JZ         LOOP
        MOV        A,RPTR
        ANL        A,#0FH
        ADD        A,#BUFFER
        MOV        R0,A
        MOV        A,@R0
        LCALL      UART2_SEND
        INC        RPTR
        JMP        LOOP

STRING:  DB         'Uart Test !',0DH,0AH,00H

        END
```

11.5.21 定时器 2 做串口 3 波特率发生器

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"           //头文件见附录
#include "intrins.h"

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

```
bit      busy;
char     wptr;
char     rptr;
char     buffer[16];
```

```
void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}
```

```
void Uart3Init()
{
    S3CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}
```

```
void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}
```

```
void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3Send(*p++);
    }
}
```

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

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC) ;头文件见附录

BUSY	BIT	20H.0	
WPTR	DATA	21H	
RPTR	DATA	22H	
BUFFER	DATA	23H	;16 bytes

```
ORG 0000H
LJMP MAIN
ORG 008BH
LJMP UART3_ISR
```

```
ORG 0100H
```

UART3_ISR:

```
PUSH ACC
PUSH PSW
MOV PSW,#08H

MOV A,S3CON
JNB ACC.1,CHKRI
ANL S3CON,#NOT 02H
CLR BUSY
```

CHKRI:

```
JNB ACC.0,UART3ISR_EXIT
```



```

        ANL        S3CON,#NOT 01H
        MOV        A,WPTR
        ANL        A,#0FH
        ADD        A,#BUFFER
        MOV        R0,A
        MOV        @R0,S3BUF
        INC        WPTR
UART3ISR_EXIT:
        POP        PSW
        POP        ACC
        RETI

UART3_INIT:
        MOV        S3CON,#10H
        MOV        T2L,#0E8H                ;65536-11059200/115200/4=0FFE8H
        MOV        T2H,#0FFH
        MOV        AUXR,#14H
        CLR        BUSY
        MOV        WPTR,#00H
        MOV        RPTR,#00H
        RET

UART3_SEND:
        JB         BUSY,$
        SETB       BUSY
        MOV        S3BUF,A
        RET

UART3_SENDSTR:
        CLR        A
        MOVC       A,@A+DPTR
        JZ         SEND3END
        LCALL      UART3_SEND
        INC        DPTR
        JMP        UART3_SENDSTR
SEND3END:
        RET

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

        LCALL      UART3_INIT
        MOV        IE2,#08H
        SETB       EA

        MOV        DPTR,#STRING
        LCALL      UART3_SENDSTR

```

LOOP:

```
MOV    A,RPTR
XRL    A,WPTR
ANL    A,#0FH
JZ     LOOP
MOV    A,RPTR
ANL    A,#0FH
ADD    A,#BUFFER
MOV    R0,A
MOV    A,@R0
LCALL  UART3_SEND
INC    RPTR
JMP    LOOP
```

STRING: DB 'Uart Test !',0DH,0AH,00H

END

11.5.22 定时器 2 做串口 4 波特率发生器

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"
#include "intrins.h"
```

//头文件见附录

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

```
bit    busy;
char    wptr;
char    rptr;
char    buffer[16];
```

void Uart4Isr() interrupt 18

```
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}
```

void Uart4Init()

```
{
    S4CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
```

```
wptr = 0x00;
rptr = 0x00;
busy = 0;
}

void Uart4Send(char dat)
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4Send(*p++);
    }
}

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

    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC)

;头文件见附录

BUSY	BIT	20H.0
WPTR	DATA	21H
RPTR	DATA	22H

```

BUFFER      DATA      23H                      ;16 bytes

                ORG      0000H
                LJMP     MAIN
                ORG      0093H
                LJMP     UART4_ISR

                ORG      0100H

UART4_ISR:
                PUSH     ACC
                PUSH     PSW
                MOV       PSW,#08H

                MOV       A,S4CON
                JNB       ACC.1,CHKRI
                ANL        S4CON,#NOT 02H
                CLR        BUSY

CHKRI:
                JNB       ACC.0,UART4ISR_EXIT
                ANL        S4CON,#NOT 01H
                MOV       A,WPTR
                ANL        A,#0FH
                ADD        A,#BUFFER
                MOV       R0,A
                MOV       @R0,S4BUF
                INC        WPTR

UART4ISR_EXIT:
                POP       PSW
                POP       ACC
                RETI

UART4_INIT:
                MOV       S4CON,#10H
                MOV       T2L,#0E8H                ;65536-11059200/115200/4=0FFE8H
                MOV       T2H,#0FFH
                MOV       AUXR,#14H
                CLR        BUSY
                MOV       WPTR,#00H
                MOV       RPTR,#00H
                RET

UART4_SEND:
                JB         BUSY,$
                SETB       BUSY
                MOV        S4BUF,A
                RET

UART4_SENDSTR:
                CLR        A
                MOVC       A,@A+DPTR
                JZ          SEND4END
                LCALL       UART4_SEND
                INC         DPTR
                JMP         UART4_SENDSTR

SEND4END:
                RET

MAIN:

```

```

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

LCALL    UART4_INIT
MOV      IE2, #10H
SETB     EA

MOV      DPTR, #STRING
LCALL    UART4_SENDSTR

```

LOOP:

```

MOV      A, RPTR
XRL      A, WPTR
ANL      A, #0FH
JZ       LOOP
MOV      A, RPTR
ANL      A, #0FH
ADD      A, #BUFFER
MOV      R0, A
MOV      A, @R0
LCALL    UART4_SEND
INC      RPTR
JMP      LOOP

```

STRING: DB 'Uart Test !', 0DH, 0AH, 00H

END

11.5.23 定时器 3（16 位自动重载），用作定时

C 语言代码

//测试工作频率为11.0592MHz

```

#include "stc16.h"          //头文件见附录
#include "intrins.h"

```

```

#define ET2      0x04
#define ET3      0x20
#define ET4      0x40
#define T2IF     0x01
#define T3IF     0x02
#define T4IF     0x04

```

```

void TM3_Isr() interrupt 19
{

```

```
    P10 = !P10;                //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;

    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
    EA = 1;

    while (1);
}
```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC)

;头文件见附录

```
ET2      EQU      04H
ET3      EQU      20H
ET4      EQU      40H
T2IF     EQU      01H
T3IF     EQU      02H
T4IF     EQU      04H
```

```
ORG      0000H
LJMP     MAIN
ORG      009BH
LJMP     TM3ISR
```

TM3ISR: ORG 0100H

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

JMP      $

END

```

11.5.24 定时器 3（外部计数—扩展 T3 为外部下降沿中断）

C 语言代码

//测试工作频率为11.0592MHz

```

#include "stc16.h"           //头文件见附录
#include "intrins.h"

```

```

#define ET2      0x04
#define ET3      0x20
#define ET4      0x40
#define T2IF     0x01
#define T3IF     0x02
#define T4IF     0x04

```

```

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

    T3L = 0x66;                //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x0c;              //设置外部计数模式并启动定时器
    IE2 = ET3;                 //使能定时器中断
    EA = 1;
}

```

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

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC) ;头文件见附录

```
ET2      EQU      04H  
ET3      EQU      20H  
ET4      EQU      40H  
T2IF     EQU      01H  
T3IF     EQU      02H  
T4IF     EQU      04H
```

```
ORG      0000H  
LJMP     MAIN  
ORG      009BH  
LJMP     TM3ISR
```

```
TM3ISR:  ORG      0100H  
          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      T3L, #66H ;65536-11.0592M/12/1000  
          MOV      T3H, #0FCH  
          MOV      T4T3M, #0CH ;设置外部计数模式并启动定时器  
          MOV      IE2, #ET3 ;使能定时器中断  
          SETB     EA  
  
          JMP      $
```

END

11.5.25 定时器 3，时钟分频输出

C 语言代码

//测试工作频率为11.0592MHz


```
#include "stc16.h"           //头文件见附录
#include "intrins.h"

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

    T3L = 0x66;               //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x09;            //使能时钟输出并启动定时器

    while (1);
}
```

汇编代码

;测试工作频率为 11.0592MHz

```
$include (STC16.INC)         ;头文件见附录

    ORG    0000H
    LJMP   MAIN

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

    MOV    T3L, #66H          ;65536-11.0592M/12/1000
    MOV    T3H, #0FCH
    MOV    T4T3M, #09H        ;使能时钟输出并启动定时器

    JMP     $

    END
```

11.5.26 定时器 3 做串口 3 波特率发生器

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"           //头文件见附录
#include "intrins.h"

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

```
bit      busy;
char     wptr;
char     rptr;
char     buffer[16];
```

```
void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}
```

```
void Uart3Init()
{
    S3CON = 0x50;
    T3L = BRT;
    T3H = BRT >> 8;
    T4T3M = 0x0a;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}
```

```
void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}
```

```
void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3Send(*p++);
    }
}
```

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

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC) ;头文件见附录

BUSY	BIT	20H.0	
WPTR	DATA	21H	
RPTR	DATA	22H	
BUFFER	DATA	23H	;16 bytes

```
ORG 0000H
LJMP MAIN
ORG 008BH
LJMP UART3_ISR
```

```
ORG 0100H
```

UART3_ISR:

```
PUSH ACC
PUSH PSW
MOV PSW,#08H

MOV A,S3CON
JNB ACC.1,CHKRI
ANL S3CON,#NOT 02H
CLR BUSY
```

CHKRI:

```
JNB ACC.0,UART3ISR_EXIT
```

```

        ANL        S3CON,#NOT 01H
        MOV        A,WPTR
        ANL        A,#0FH
        ADD        A,#BUFFER
        MOV        R0,A
        MOV        @R0,S3BUF
        INC        WPTR
UART3ISR_EXIT:
        POP        PSW
        POP        ACC
        RETI

UART3_INIT:
        MOV        S3CON,#50H
        MOV        T3L,#0E8H                ;65536-11059200/115200/4=0FFE8H
        MOV        T3H,#0FFH
        MOV        T4T3M,#0AH
        CLR        BUSY
        MOV        WPTR,#00H
        MOV        RPTR,#00H
        RET

UART3_SEND:
        JB         BUSY,$
        SETB       BUSY
        MOV        S3BUF,A
        RET

UART3_SENDSTR:
        CLR        A
        MOVC       A,@A+DPTR
        JZ         SEND3END
        LCALL      UART3_SEND
        INC        DPTR
        JMP        UART3_SENDSTR
SEND3END:
        RET

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

        LCALL      UART3_INIT
        MOV        IE2,#08H
        SETB       EA

        MOV        DPTR,#STRING
        LCALL      UART3_SENDSTR

```

LOOP:

```
MOV    A,RPTR
XRL    A,WPTR
ANL    A,#0FH
JZ     LOOP
MOV    A,RPTR
ANL    A,#0FH
ADD    A,#BUFFER
MOV    R0,A
MOV    A,@R0
LCALL  UART3_SEND
INC    RPTR
JMP    LOOP
```

STRING: DB 'Uart Test !',0DH,0AH,00H

END

11.5.27 定时器 4（16 位自动重载），用作定时

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"           //头文件见附录
#include "intrins.h"
```

```
#define ET2          0x04
#define ET3          0x20
#define ET4          0x40
#define T2IF         0x01
#define T3IF         0x02
#define T4IF         0x04
```

```
void TM4_Isr() interrupt 20
{
    P10 = !P10;           //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;

    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;

while (1);

}
```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC)

;头文件见附录

```
ET2      EQU      04H
ET3      EQU      20H
ET4      EQU      40H
T2IF     EQU      01H
T3IF     EQU      02H
T4IF     EQU      04H
```

```
ORG      0000H
LJMP     MAIN
ORG      00A3H
LJMP     TM4ISR
```

TM4ISR: ORG 0100H

```
CPL      P1.0
RETI
```

;测试端口

MAIN:

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

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

JMP \$

END

11.5.28 定时器 4（外部计数—扩展 T4 为外部下降沿中断）

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"           //头文件见附录
#include "intrins.h"

#define ET2      0x04
#define ET3      0x20
#define ET4      0x40
#define T2IF     0x01
#define T3IF     0x02
#define T4IF     0x04

void TM4_Isr() interrupt 20
{
    P10 = !P10;               //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;

    T4L = 0x66;               //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0xc0;             //设置外部计数模式并启动定时器
    IE2 = ET4;                //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

;测试工作频率为11.0592MHz

```
$include (STC16.INC)         ;头文件见附录

ET2      EQU      04H
ET3      EQU      20H
ET4      EQU      40H
T2IF     EQU      01H
T3IF     EQU      02H
T4IF     EQU      04H

        ORG      0000H
        LJMP     MAIN
        ORG      00A3H
        LJMP     TM4ISR
```

```
ORG      0100H

TM4ISR:  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      T4L, #66H          ;65536-11.0592M/12/1000
          MOV      T4H, #0FCH
          MOV      T4T3M, #0C0H      ;设置外部计数模式并启动定时器
          MOV      IE2, #ET4         ;使能定时器中断
          SETB     EA

          JMP      $

          END
```

11.5.29 定时器 4，时钟分频输出

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"          //头文件见附录
#include "intrins.h"
```

```
void main()
```

```
{
```

```
    P0M0 = 0x00;
```

```
    P0M1 = 0x00;
```

```
    P1M0 = 0x00;
```

```
    P1M1 = 0x00;
```

```
    P2M0 = 0x00;
```

```
    P2M1 = 0x00;
```

```
    P3M0 = 0x00;
```

```
    P3M1 = 0x00;
```

```
    P4M0 = 0x00;
```

```
    P4M1 = 0x00;
```

```
    P5M0 = 0x00;
```

```
    P3M1 = 0x00;
```

```
    T4L = 0x66;
```

```
    //65536-11.0592M/12/1000
```

```
    T4H = 0xfc;
```



```
T4T3M = 0x90; //使能时钟输出并启动定时器

while (1);
}
```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC) ;头文件见附录

```
ORG 0000H
LJMP MAIN

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

MOV T4L, #66H ;65536-11.0592M/12/1000
MOV T4H, #0FCH
MOV T4T3M, #90H ;使能时钟输出并启动定时器

JMP $

END
```

11.5.30 定时器 4 做串口 4 波特率发生器

C 语言代码

//测试工作频率为 11.0592MHz

```
#include "stc16.h" //头文件见附录
#include "intrins.h"
```

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

```
bit busy;
char wptr;
char rptr;
char buffer[16];
```

```
void Uart4Isr() interrupt 18
{
```

```
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}

void Uart4Init()
{
    S4CON = 0x50;
    T4L = BRT;
    T4H = BRT >> 8;
    T4T3M = 0xa0;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart4Send(char dat)
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4Send(*p++);
    }
}

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

    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SendStr("Uart Test !\r\n");
}
```

```

while (1)
{
    if (rptr != wptr)
    {
        Uart4Send(buffer[rptr++]);
        rptr &= 0x0f;
    }
}
}

```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC)

;头文件见附录

BUSY	BIT	20H.0	
WPTR	DATA	21H	
RPTR	DATA	22H	
BUFFER	DATA	23H	;16 bytes

```

ORG    0000H
LJMP   MAIN
ORG    0093H
LJMP   UART4_ISR

```

```
ORG    0100H
```

UART4_ISR:

```

PUSH   ACC
PUSH   PSW
MOV     PSW,#08H

MOV     A,S4CON
JNB     ACC.1,CHKRI
ANL     S4CON,#NOT 02H
CLR     BUSY

```

CHKRI:

```

JNB     ACC.0,UART4ISR_EXIT
ANL     S4CON,#NOT 01H
MOV     A,WPTR
ANL     A,#0FH
ADD     A,#BUFFER
MOV     R0,A
MOV     @R0,S4BUF
INC     WPTR

```

UART4ISR_EXIT:

```

POP     PSW
POP     ACC
RETI

```

UART4_INIT:

```

MOV     S4CON,#50H
MOV     T4L,#0E8H           ;65536-11059200/115200/4=0FFE8H
MOV     T4H,#0FFH
MOV     T4T3M,#0A0H
CLR     BUSY
MOV     WPTR,#00H
MOV     RPTR,#00H

```

RET

UART4_SEND:

JB BUSY,\$
SETB BUSY
MOV S4BUF,A
RET

UART4_SENDSTR:

CLR A
MOVC A,@A+DPTR
JZ SEND4END
LCALL UART4_SEND
INC DPTR
JMP UART4_SENDSTR

SEND4END:

RET

MAIN:

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

LCALL UART4_INIT
MOV IE2,#10H
SETB EA

MOV DPTR,#STRING
LCALL UART4_SENDSTR

LOOP:

MOV A,RPTR
XRL A,WPTR
ANL A,#0FH
JZ LOOP
MOV A,RPTR
ANL A,#0FH
ADD A,#BUFFER
MOV R0,A
MOV A,@R0
LCALL UART4_SEND
INC RPTR
JMP LOOP

STRING: DB 'Uart Test !',0DH,0AH,00H

END

12 串口通信

STC16F 系列单片机具有 4 个全双工异步串行通信接口。每个串行口由 2 个数据缓冲器、一个移位寄存器、一个串行控制寄存器和一个波特率发生器等组成。每个串行口的数据缓冲器由 2 个互相独立的接收、发送缓冲器构成，可以同时发送和接收数据。

STC16F 系列单片机的串口 1 有 4 种工作方式，其中两种方式的波特率是可变的，另两种是固定的，以供不同应用场合选用。串口 2/串口 3/串口 4 都只有两种工作方式，这两种方式的波特率都是可变的。用户可用软件设置不同的波特率和选择不同的工作方式。主机可通过查询或中断方式对接收/发送进行程序处理，使用十分灵活。

串口 1、串口 2、串口 3、串口 4 的通讯口均可以通过功能管脚的切换功能切换到多组端口，从而可以将一个通讯口分时复用为多个通讯口。

12.1 串口功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	9AH	S1_S[1:0]		CAN_S[1:0]		SPI_S[1:0]		LIN_S[1:0]	

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

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW2	9BH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S

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	P4.6	P4.7

12.2 串口相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	串口 1 数据寄存器	99H									0000,0000

S2CON	串口 2 控制寄存器	E3H	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
S2BUF	串口 2 数据寄存器	E4H									0000,0000
S3CON	串口 3 控制寄存器	E5H	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S3BUF	串口 3 数据寄存器	E6H									0000,0000
S4CON	串口 4 控制寄存器	BCH	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
S4BUF	串口 4 数据寄存器	BDH									0000,0000
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
AUXR	辅助寄存器 1	93H	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
SADDR	串口 1 从机地址寄存器	E1H									0000,0000
SADEN	串口 1 从机地址屏蔽寄存器	E2H									0000,0000

12.3 串口 1

12.3.1 串口 1 控制寄存器 (SCON)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

SM0/FE: 当PCON寄存器中的SMOD0位为1时, 该位为帧错误检测标志位。当UART在接收过程中检测到一个无效停止位时, 通过UART接收器将该位置1, 必须由软件清零。当PCON寄存器中的SMOD0位为0时, 该位和SM1一起指定串口1的通信工作模式, 如下表所示:

SM0	SM1	串口1工作模式	功能说明
0	0	模式0	同步移位串行方式
0	1	模式1	可变波特率8位数据方式
1	0	模式2	固定波特率9位数据方式
1	1	模式3	可变波特率9位数据方式

SM2: 允许模式 2 或模式 3 多机通信控制位。当串口 1 使用模式 2 或模式 3 时, 如果 SM2 位为 1 且 REN 位为 1, 则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位 (即 RB8) 来筛选地址帧, 若 RB8=1, 说明该帧是地址帧, 地址信息可以进入 SBUF, 并使 RI 为 1, 进而在中断服务程序中再进行地址号比较; 若 RB8=0, 说明该帧不是地址帧, 应丢掉且保持 RI=0。在模式 2 或模式 3 中, 如果 SM2 位为 0 且 REN 位为 1, 接收机处于地址帧筛选被禁止状态, 不论收到的 RB8 为 0 或 1, 均可使接收到的信息进入 SBUF, 并使 RI=1, 此时 RB8 通常为校验位。模式 1 和模式 0 为非多机通信方式, 在这两种方式时, SM2 应设置为 0。

REN: 允许/禁止串口接收控制位

0: 禁止串口接收数据

1: 允许串口接收数据

TB8: 当串口 1 使用模式 2 或模式 3 时, TB8 为要发送的第 9 位数据, 按需要由软件置位或清 0。在模式 0 和模式 1 中, 该位不用。

RB8: 当串口 1 使用模式 2 或模式 3 时, RB8 为接收到的第 9 位数据, 一般用作校验位或者地址帧/数据帧标志位。在模式 0 和模式 1 中, 该位不用。

TI: 串口 1 发送中断请求标志位。在模式 0 中, 当串口发送数据第 8 位结束时, 由硬件自动将 TI 置 1, 向主机请求中断, 响应中断后 TI 必须用软件清零。在其他模式中, 则在停止位开始发送时由硬件自动将 TI 置 1, 向 CPU 发请求中断, 响应中断后 TI 必须用软件清零。

RI: 串口 1 接收中断请求标志位。在模式 0 中, 当串口接收第 8 位数据结束时, 由硬件自动将 RI 置 1, 向主机请求中断, 响应中断后 RI 必须用软件清零。在其他模式中, 串行接收到停止位的中间时刻由硬件自动将 RI 置 1, 向 CPU 发中断申请, 响应中断后 RI 必须由软件清零。

12.3.2 串口 1 数据寄存器 (SBUF)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SBUF	99H								

SBUF: 串口 1 数据接收/发送缓冲区。SBUF 实际是 2 个缓冲器, 读缓冲器和写缓冲器, 两个操作分别对应两个不同的寄存器, 1 个是只写寄存器 (写缓冲器), 1 个是只读寄存器 (读缓冲器)。对 SBUF 进行读操作, 实际是读取串口接收缓冲区, 对 SBUF 进行写操作则是触发串口开始发送数据。

12.3.3 电源管理寄存器 (PCON)

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

SMOD: 串口 1 波特率控制位

0: 串口 1 的各个模式的波特率都不加倍

1: 串口 1 模式 1、模式 2、模式 3 的波特率加倍

SMOD0: 帧错误检测控制位

0: 无帧错检测功能

1: 使能帧错误检测功能。此时 SCON 的 SM0/FE 为 FE 功能, 即为帧错误检测标志位。

12.3.4 辅助寄存器 1 (AUXR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	93H	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

UART_M0x6: 串口 1 模式 0 的通讯速度控制

0: 串口 1 模式 0 的波特率不加倍, 固定为 $F_{osc}/12$

1: 串口 1 模式 0 的波特率 6 倍速, 即固定为 $F_{osc}/12 \times 6 = F_{osc}/2$

S1ST2: 串口 1 波特率发射器选择位

0: 选择定时器 1 作为波特率发射器

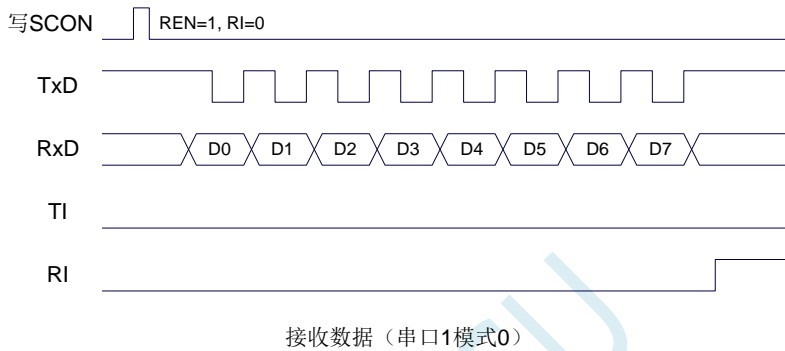
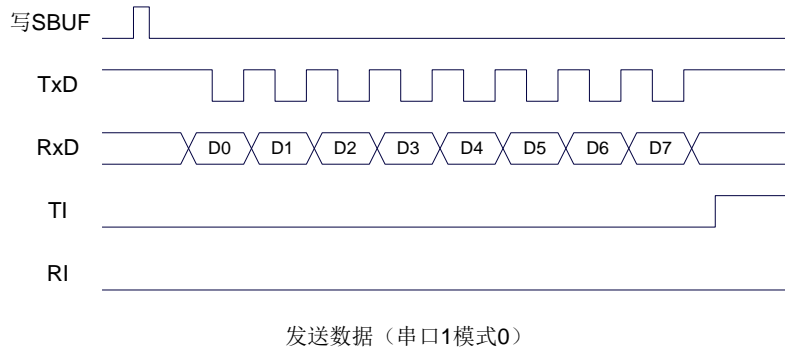
1: 选择定时器 2 作为波特率发射器

12.3.5 串口 1 模式 0, 模式 0 波特率计算公式

当串口 1 选择工作模式为模式 0 时, 串行通信接口工作在同步移位寄存器模式, 当串行口模式 0 的通信速度设置位 UART_M0x6 为 0 时, 其波特率固定为系统时钟频率的 12 分频 ($SYSClk/12$); 当设置 UART_M0x6 为 1 时, 其波特率固定为系统时钟频率的 2 分频 ($SYSClk/2$)。RxD 为串行通讯的数据口, TxD 为同步移位脉冲输出脚, 发送、接收的是 8 位数据, 低位在先。

模式 0 的发送过程: 当主机执行将数据写入发送缓冲器 SBUF 指令时启动发送, 串行口即将 8 位数据以 $SYSClk/12$ 或 $SYSClk/2$ (由 UART_M0x6 确定是 12 分频还是 2 分频) 的波特率从 RxD 管脚输出(从低位到高位), 发送完中断标志 TI 置 1, TxD 管脚输出同步移位脉冲信号。当写信号有效后, 相隔一个时钟, 发送控制端 SEND 有效(高电平), 允许 RxD 发送数据, 同时允许 TxD 输出同步移位脉冲。一帧 (8 位) 数据发送完毕时, 各控制端均恢复原状态, 只有 TI 保持高电平, 呈中断申请状态。在再次发送数据前, 必须用软件将 TI 清 0。

模式 0 的接收过程: 首先将接收中断请求标志 RI 清零并置位允许接收控制位 REN 时启动模式 0 接收过程。启动接收过程后, RxD 为串行数据输入端, TxD 为同步脉冲输出端。串行接收的波特率为 $SYSClk/12$ 或 $SYSClk/2$ (由 UART_M0x6 确定是 12 分频还是 2 分频)。当接收完成一帧数据 (8 位) 后, 控制信号复位, 中断标志 RI 被置 1, 呈中断申请状态。当再次接收时, 必须通过软件将 RI 清 0



工作于模式 0 时，必须清 0 多机通信控制位 SM2，使之不影响 TB8 位和 RB8 位。由于波特率固定为 SYSclk/12 或 SYSclk/2，无需定时器提供，直接由单片机的时钟作为同步移位脉冲。

串口 1 模式 0 的波特率计算公式如下表所示（SYSclk 为系统工作频率）：

UART_M0x6	波特率计算公式
0	$\text{波特率} = \frac{\text{SYSclk}}{12}$
1	$\text{波特率} = \frac{\text{SYSclk}}{2}$

12.3.6 串口 1 模式 1，模式 1 波特率计算公式

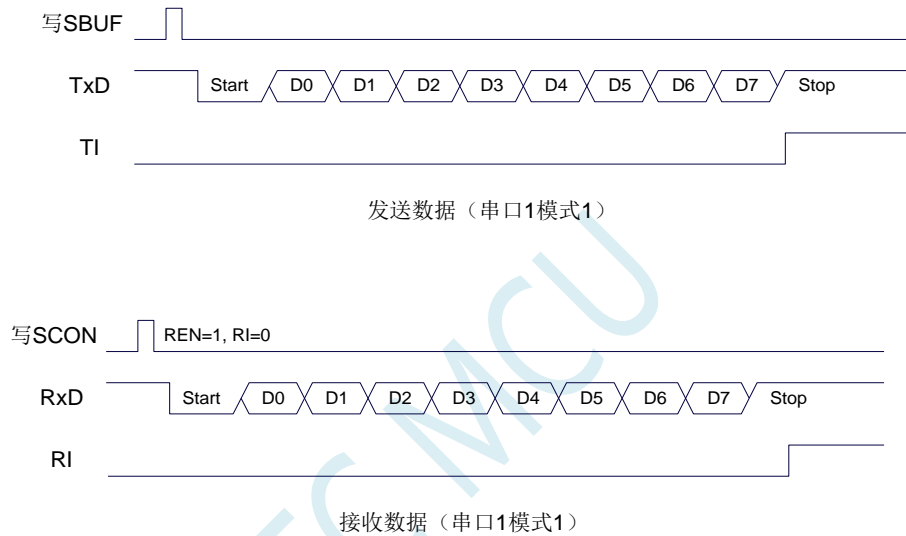
当软件设置 SCON 的 SM0、SM1 为“01”时，串行口 1 则以模式 1 进行工作。此模式为 8 位 UART 格式，一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，即可根据需要进行设置波特率。TxD 为数据发送口，RxD 为数据接收口，串行口全双工接受/发送。

模式 1 的发送过程：串行通信模式发送时，数据由串行发送端 TxD 输出。当主机执行一条写 SBUF 的指令就启动串行通信的发送，写“SBUF”信号还把“1”装入发送移位寄存器的第 9 位，并通知 TX 控制单元开始发送。移位寄存器将数据不断右移送 TxD 端口发送，在数据的左边不断移入“0”作补充。当数据的最高位移到移位寄存器的输出位置，紧跟其后的是第 9 位“1”，在它的左边各位全为“0”，这个状态条件，使 TX 控制单元作最后一次移位输出，然后使允许发送信号“SEND”失效，完成一帧信息的发送，并置位中断请求位 TI，即 TI=1，向主机请求中断处理。

模式 1 的接收过程: 当软件置位接收允许标志位 REN, 即 REN=1 时, 接收器便对 RxD 端口的信号进行检测, 当检测到 RxD 端口发送从“1”→“0”的下降沿跳变时就启动接收器准备接收数据, 并立即复位波特率发生器的接收计数器, 将 1FFH 装入移位寄存器。接收的数据从接收移位寄存器的右边移入, 已装入的 1FFH 向左边移出, 当起始位“0”移到移位寄存器的最左边时, 使 RX 控制器作最后一次移位, 完成一帧的接收。若同时满足以下两个条件:

- RI=0;
- SM2=0 或接收到的停止位为 1。

则接收到的数据有效, 实现装载入 SBUF, 停止位进入 RB8, RI 标志位被置 1, 向主机请求中断, 若上述两条件不能同时满足, 则接收到的数据作废并丢失, 无论条件满足与否, 接收器重又检测 RxD 端口上的“1”→“0”的跳变, 继续下一帧的接收。接收有效, 在响应中断后, RI 标志位必须由软件清 0。通常情况下, 串行通信工作于模式 1 时, SM2 设置为“0”。



串口 1 的波特率是可变的, 其波特率可由定时器 1 或者定时器 2 产生。当定时器采用 1T 模式时 (12 倍速), 相应的波特率的速度也会相应提高 12 倍。

串口 1 模式 1 的波特率计算公式如下表所示: (SYSclk 为系统工作频率)

选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536 - \frac{\text{SYSclk}}{4 \times \text{波特率}}$
	12T	定时器2重载值 = $65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{波特率}}$
定时器1模式0	1T	定时器1重载值 = $65536 - \frac{\text{SYSclk}}{4 \times \text{波特率}}$
	12T	定时器1重载值 = $65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{波特率}}$
定时器1模式2	1T	定时器1重载值 = $256 - \frac{2^{\text{SMOD}} \times \text{SYSclk}}{32 \times \text{波特率}}$
	12T	定时器1重载值 = $256 - \frac{2^{\text{SMOD}} \times \text{SYSclk}}{12 \times 32 \times \text{波特率}}$

下面为常用频率与常用波特率所对应定时器的重载值

频率 (MHz)	波特率	定时器 2		定时器 1 模式 0		定时器 1 模式 2			
		1T 模式	12T 模式	1T 模式	12T 模式	SMOD=1		SMOD=0	
						1T 模式	12T 模式	1T 模式	12T 模式
11.0592	115200	FFE8H	FFFEH	FFE8H	FFFEH	FAH	-	FDH	-
	57600	FFD0H	FFFCH	FFD0H	FFFCH	F4H	FFH	FAH	-
	38400	FFB8H	FFFAH	FFB8H	FFFAH	EEH	-	F7H	-
	19200	FF70H	FFF4H	FF70H	FFF4H	DCH	FDH	EEH	-
	9600	FEE0H	FFE8H	FEE0H	FFE8H	B8H	FAH	DCH	FDH
18.432	115200	FFD8H	-	FFD8H	-	F6H	-	FBH	-
	57600	FFB0H	-	FFB0H	-	ECH	-	F6H	-
	38400	FF88H	FFF6H	FF88H	FFF6H	E2H	-	F1H	-
	19200	FF10H	FFECH	FF10H	FFECH	C4H	FBH	E2H	-
	9600	FE20H	FFD8H	FE20H	FFD8H	88H	F6H	C4H	FBH
22.1184	115200	FFD0H	FFFCH	FFD0H	FFFCH	F4H	FFH	FAH	-
	57600	FFA0H	FFF8H	FFA0H	FFF8H	E8H	FEH	F4H	FFH
	38400	FF70H	FFF4H	FF70H	FFF4H	DCH	FDH	EEH	-
	19200	FEE0H	FFE8H	FEE0H	FFE8H	B8H	FAH	DCH	FDH
	9600	FDC0H	FFD0H	FDC0H	FFD0H	70H	F4H	B8H	FAH

12.3.7 串口 1 模式 2，模式 2 波特率计算公式

当 SM0、SM1 两位为 10 时，串行口 1 工作在模式 2。串行口 1 工作模式 2 为 9 位数据异步通信 UART 模式，其一帧的信息由 11 位组成：1 位起始位，8 位数据位（低位在先），1 位可编程位（第 9 位数据）和 1 位停止位。发送时可编程位（第 9 位数据）由 SCON 中的 TB8 提供，可软件设置为 1 或 0，或者可将 PSW 中的奇/偶校验位 P 值装入 TB8（TB8 既可作为多机通信中的地址数据标志位，又可作为数据的奇偶校验位）。接收时第 9 位数据装入 SCON 的 RB8。TxD 为发送端口，RxD 为接收端口，以全双工模式进行接收/发送。

模式 2 的波特率固定为系统时钟的 64 分频或 32 分频（取决于 PCON 中 SMOD 的值）

串口 1 模式 2 的波特率计算公式如下表所示（SYSclk 为系统工作频率）：

SMOD	波特率计算公式
0	$\text{波特率} = \frac{\text{SYSclk}}{64}$
1	$\text{波特率} = \frac{\text{SYSclk}}{32}$

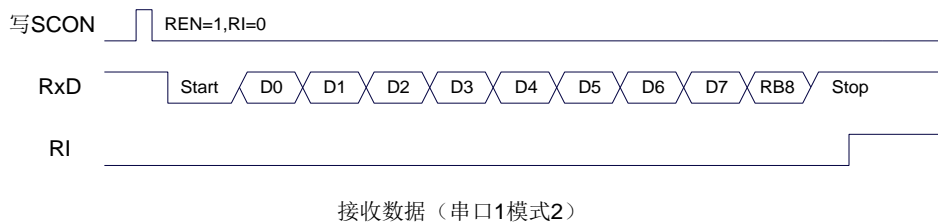
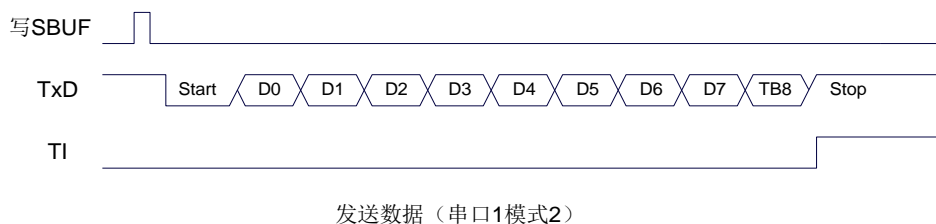
模式 2 和模式 1 相比，除波特率发生源略有不同，发送时由 TB8 提供给移位寄存器第 9 数据位不同外，其余功能结构均基本相同，其接收/发送操作过程及时序也基本相同。

当接收器接收完一帧信息后必须同时满足下列条件：

- RI=0
- SM2=0 或者 SM2=1 且接收到的第 9 数据位 RB8=1。

当上述两条件同时满足时，才将接收到的移位寄存器的数据装入 SBUF 和 RB8 中，RI 标志位被置 1，并向主机请求中断处理。如果上述条件有一个不满足，则刚接收到移位寄存器中的数据无效而丢失，也不置位 RI。无论上述条件满足与否，接收器又重新开始检测 RxD 输入端口的跳变信息，接收下一帧的输入信息。在模式 2 中，接收到的停止位与 SBUF、RB8 和 RI 无关。

通过软件对 SCON 中的 SM2、TB8 的设置以及通信协议的约定，为多机通信提供了方便。



12.3.8 串口 1 模式 3，模式 3 波特率计算公式

当 SM0、SM1 两位为 11 时，串行口 1 工作在模式 3。串行通信模式 3 为 9 位数据异步通信 UART

模式，其一帧的信息由 11 位组成：1 位起始位，8 位数据位（低位在先），1 位可编程位（第 9 位数据）和 1 位停止位。发送时可编程位（第 9 位数据）由 SCON 中的 TB8 提供，可软件设置为 1 或 0，或者可将 PSW 中的奇/偶校验位 P 值装入 TB8（TB8 既可作为多机通信中的地址数据标志位，又可作为数据的奇偶校验位）。接收时第 9 位数据装入 SCON 的 RB8。TxD 为发送端口，RxD 为接收端口，以全双工模式进行接收/发送。

模式 3 和模式 1 相比，除发送时由 TB8 提供给移位寄存器第 9 数据位不同外，其余功能结构均基本相同，其接收‘发送操作过程及时序也基本相同。

当接收器接收完一帧信息后必须同时满足下列条件：

- RI=0
- SM2=0 或者 SM2=1 且接收到的第 9 数据位 RB8=1。

当上述两条件同时满足时，才将接收到的移位寄存器的数据装入 SBUF 和 RB8 中，RI 标志位被置 1，并向主机请求中断处理。如果上述条件有一个不满足，则刚接收到移位寄存器中的数据无效而丢失，也不置位 RI。无论上述条件满足与否，接收器又重新开始检测 RxD 输入端口的跳变信息，接收下一帧的输入信息。在模式 3 中，接收到的停止位与 SBUF、RB8 和 RI 无关。

通过软件对 SCON 中的 SM2、TB8 的设置以及通信协议的约定，为多机通信提供了方便。



串口 1 模式 3 的波特率计算公式与模式 1 是完全相同的。请参考模式 1 的波特率计算公式。

12.3.9 自动地址识别

12.3.10 串口 1 从机地址控制寄存器（SADDR, SADEN）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SADDR	E1H								
SADEN	E2H								

SADDR：从机地址寄存器

SADEN：从机地址屏蔽位寄存器

自动地址识别功能典型应用在多机通讯领域，其主要原理是从机系统通过硬件比较功能来识别来自于主机串口数据流中的地址信息，通过寄存器 SADDR 和 SADEN 设置的本机的从机地址，硬件自动对从机地址进行过滤，当来自于主机的从机地址信息与本机所设置的从机地址相匹配时，硬件产生串口

断；否则硬件自动丢弃串口数据，而不产生中断。当众多处于空闲模式的从机链接在一起时，只有从机地址相匹配的从机才会从空闲模式唤醒，从而可以大大降低从机 MCU 的功耗，即使从机处于正常工作状态也可避免不停地进入串口中断而降低系统执行效率。

要使用串口的自动地址识别功能，首先需要将参与通讯的 MCU 的串口通讯模式设置为模式 2 或者模式 3（通常都选择波特率可变的模式 3，因为模式 2 的波特率是固定的，不便于调节），并开启从机的 SCON 的 SM2 位。对于串口模式 2 或者模式 3 的 9 位数据位中，第 9 位数据（存放在 RB8 中）为地址/数据的标志位，当第 9 位数据为 1 时，表示前面的 8 位数据（存放在 SBUF 中）为地址信息。当 SM2 被设置为 1 时，从机 MCU 会自动过滤掉非地址数据（第 9 位为 0 的数据），而对 SBUF 中的地址数据（第 9 位为 1 的数据）自动与 SADDR 和 SADEN 所设置的本机地址进行比较，若地址相匹配，则会将 RI 置“1”，并产生中断，否则不予处理本次接收的串口数据。

从机地址的设置是通过 SADDR 和 SADEN 两个寄存器进行设置的。SADDR 为从机地址寄存器，里面存放本机的从机地址。SADEN 为从机地址屏蔽位寄存器，用于设置地址信息中的忽略位，设置方法如下：

例如

SADDR = 11001010

SADEN = 10000001

则匹配地址为 1xxxxxx0

即，只要主机送出的地址数据中的 bit0 为 0 且 bit7 为 1 就可以和本机地址相匹配

再例如

SADDR = 11001010

SADEN = 00001111

则匹配地址为 xxxx1010

即，只要主机送出的地址数据中的低 4 位为 1010 就可以和本机地址相匹配，而高 4 为被忽略，可以为任意值。

主机可以使用广播地址（FFH）同时选中所有的从机来进行通讯。

12.4 串口 2

12.4.1 串口 2 控制寄存器 (S2CON)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S2CON	E3H	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI

S2SM0: 指定串口2的通信工作模式, 如下表所示:

S2SM0	串口2工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S2SM2: 允许串口 2 在模式 1 时允许多机通信控制位。在模式 1 时, 如果 S2SM2 位为 1 且 S2REN 位为 1, 则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位 (即 S2RB8) 来筛选地址帧: 若 S2RB8=1, 说明该帧是地址帧, 地址信息可以进入 S2BUF, 并使 S2RI 为 1, 进而在中断服务程序中再进行地址号比较; 若 S2RB8=0, 说明该帧不是地址帧, 应丢掉且保持 S2RI=0。在模式 1 中, 如果 S2SM2 位为 0 且 S2REN 位为 1, 接收机处于地址帧筛选被禁止状态。不论收到的 S2RB8 为 0 或 1, 均可使接收到的信息进入 S2BUF, 并使 S2RI=1, 此时 S2RB8 通常为校验位。模式 0 为非多机通信方式, 在这种方式时, 要设置 S2SM2 应为 0。

S2REN: 允许/禁止串口接收控制位

0: 禁止串口接收数据

1: 允许串口接收数据

S2TB8: 当串口 2 使用模式 1 时, S2TB8 为要发送的第 9 位数据, 一般用作校验位或者地址帧/数据帧标志位, 按需要由软件置位或清 0。在模式 0 中, 该位不用。

S2RB8: 当串口 2 使用模式 1 时, S2RB8 为接收到的第 9 位数据, 一般用作校验位或者地址帧/数据帧标志位。在模式 0 中, 该位不用。

S2TI: 串口 2 发送中断请求标志位。在停止位开始发送时由硬件自动将 S2TI 置 1, 向 CPU 发请求中断, 响应中断后 S2TI 必须用软件清零。

S2RI: 串口 2 接收中断请求标志位。串行接收到停止位的中间时刻由硬件自动将 S2RI 置 1, 向 CPU 发中断申请, 响应中断后 S2RI 必须由软件清零。

12.4.2 串口 2 数据寄存器 (S2BUF)

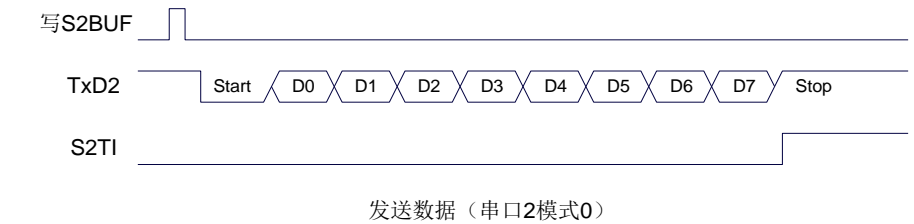
符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S2BUF	E4H								

S2BUF: 串口 1 数据接收/发送缓冲区。S2BUF 实际是 2 个缓冲器, 读缓冲器和写缓冲器, 两个操作分别对应两个不同的寄存器, 1 个是只写寄存器 (写缓冲器), 1 个是只读寄存器 (读缓冲器)。

对 S2BUF 进行读操作, 实际是读取串口接收缓冲区, 对 S2BUF 进行写操作则是触发串口开始发送数据。

12.4.3 串口 2 模式 0, 模式 0 波特率计算公式

串行口 2 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位: 1 位起始位, 8 位数据位 (低位在先) 和 1 位停止位。波特率可变, 可根据需要进行设置波特率。Tx D2 为数据发送口, Rx D2 为数据接收口, 串行口全双工接受/发送。



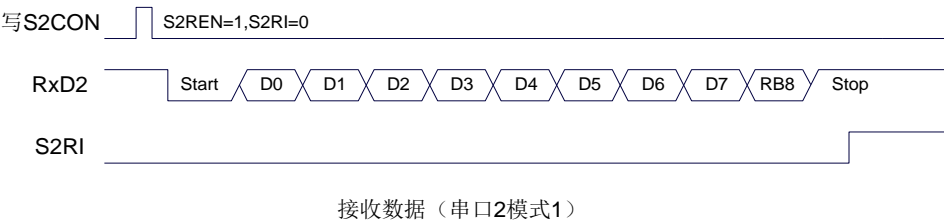
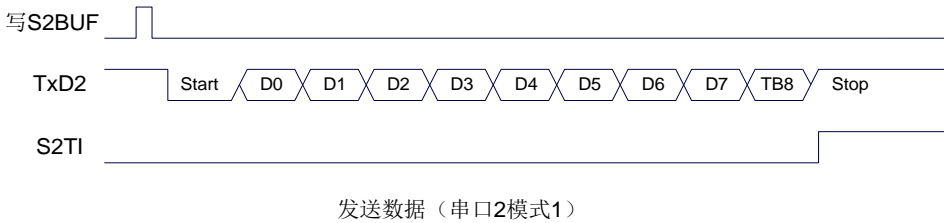
串口 2 的波特率是可变的，其波特率由定时器 2 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

串口 2 模式 0 的波特率计算公式如下表所示：（SYSclk 为系统工作频率）

选择定时器	定时器速度	波特率计算公式
定时器2	1T	$\text{定时器2重载值} = 65536 - \frac{\text{SYSclk}}{4 \times \text{波特率}}$
	12T	$\text{定时器2重载值} = 65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{波特率}}$

12.4.4 串口 2 模式 1，模式 1 波特率计算公式

串行口 2 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位：1 位起始位，9 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD2 为数据发送口，RxD2 为数据接收口，串行口全双工接受/发送。



串口 2 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

STC MCU

12.5 串口 3

12.5.1 串口 3 控制寄存器 (S3CON)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S3CON	E5H	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI

S3SM0: 指定串口3的通信工作模式, 如下表所示:

S3SM0	串口3工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S3ST3: 选择串口 3 的波特率发生器

0: 选择定时器 2 为串口 3 的波特率发生器

1: 选择定时器 3 为串口 3 的波特率发生器

S3SM2: 允许串口 3 在模式 1 时允许多机通信控制位。在模式 1 时, 如果 S3SM2 位为 1 且 S3REN 位为 1, 则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位 (即 S3RB8) 来筛选地址帧: 若 S3RB8=1, 说明该帧是地址帧, 地址信息可以进入 S3BUF, 并使 S3RI 为 1, 进而在中断服务程序中再进行地址号比较; 若 S3RB8=0, 说明该帧不是地址帧, 应丢掉且保持 S3RI=0。在模式 1 中, 如果 S3SM2 位为 0 且 S3REN 位为 1, 接收机处于地址帧筛选被禁止状态。不论收到的 S3RB8 为 0 或 1, 均可使接收到的信息进入 S3BUF, 并使 S3RI=1, 此时 S3RB8 通常为校验位。模式 0 为非多机通信方式, 在这种方式时, 要设置 S3SM2 应为 0。

S3REN: 允许/禁止串口接收控制位

0: 禁止串口接收数据

1: 允许串口接收数据

S3TB8: 当串口 3 使用模式 1 时, S3TB8 为要发送的第 9 位数据, 一般用作校验位或者地址帧/数据帧标志位, 按需要由软件置位或清 0。在模式 0 中, 该位不用。

S3RB8: 当串口 3 使用模式 1 时, S3RB8 为接收到的第 9 位数据, 一般用作校验位或者地址帧/数据帧标志位。在模式 0 中, 该位不用。

S3TI: 串口 3 发送中断请求标志位。在停止位开始发送时由硬件自动将 S3TI 置 1, 向 CPU 发请求中断, 响应中断后 S3TI 必须用软件清零。

S3RI: 串口 3 接收中断请求标志位。串行接收到停止位的中间时刻由硬件自动将 S3RI 置 1, 向 CPU 发中断申请, 响应中断后 S3RI 必须由软件清零。

12.5.2 串口 3 数据寄存器 (S3BUF)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S3BUF	E6H								

S3BUF: 串口 1 数据接收/发送缓冲区。S3BUF 实际是 2 个缓冲器, 读缓冲器和写缓冲器, 两个操作分别对应两个不同的寄存器, 1 个是只写寄存器 (写缓冲器), 1 个是只读寄存器 (读缓冲器)。对 S3BUF 进行读操作, 实际是读取串口接收缓冲区, 对 S3BUF 进行写操作则是触发串口开始发送数据。

12.5.3 串口 3 模式 0, 模式 0 波特率计算公式

串行口 3 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位: 1 位起始位, 8 位数据位 (低位在先) 和 1 位停止位。波特率可变, 可根据需要进行设置波特率。TxD3 为数据

发送口，RxD3 为数据接收口，串行口全双工接受/发送。



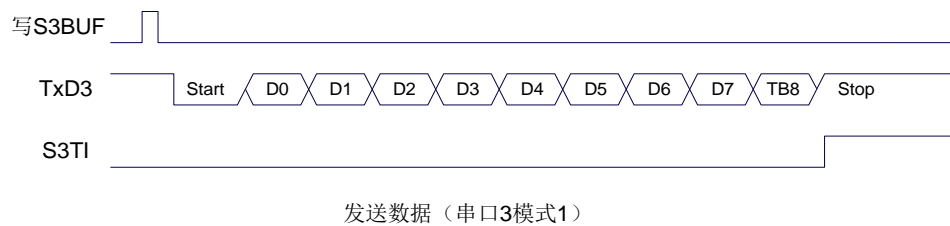
串口 3 的波特率是可变的，其波特率可由定时器 2 或定时器 3 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

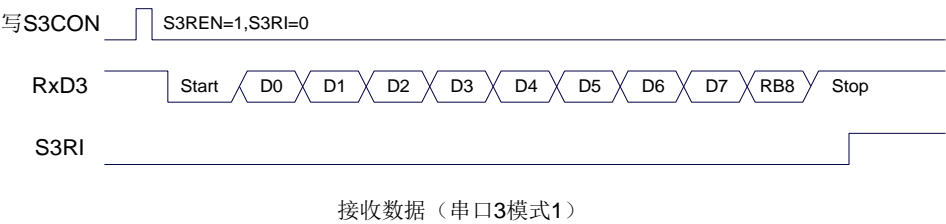
串口 3 模式 0 的波特率计算公式如下表所示：（SYSclk 为系统工作频率）

选择定时器	定时器速度	波特率计算公式
定时器2	1T	$\text{定时器2重载值} = 65536 - \frac{\text{SYSclk}}{4 \times \text{波特率}}$
	12T	$\text{定时器2重载值} = 65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{波特率}}$
定时器3	1T	$\text{定时器3重载值} = 65536 - \frac{\text{SYSclk}}{4 \times \text{波特率}}$
	12T	$\text{定时器3重载值} = 65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{波特率}}$

12.5.4 串口 3 模式 1，模式 1 波特率计算公式

串行口 3 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位：1 位起始位，9 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD3 为数据发送口，RxD3 为数据接收口，串行口全双工接受/发送。





串口 3 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

12.6 串口 4

12.6.1 串口 4 控制寄存器 (S4CON)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S4CON	BCH	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

S4SM0: 指定串口4的通信工作模式, 如下表所示:

S4SM0	串口4工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S4ST4: 选择串口 4 的波特率发生器

0: 选择定时器 2 为串口 4 的波特率发生器

1: 选择定时器 4 为串口 4 的波特率发生器

S4SM2: 允许串口 4 在模式 1 时允许多机通信控制位。在模式 1 时, 如果 S4SM2 位为 1 且 S4REN 位为 1, 则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位 (即 S4RB8) 来筛选地址帧: 若 S4RB8=1, 说明该帧是地址帧, 地址信息可以进入 S4BUF, 并使 S4RI 为 1, 进而在中断服务程序中再进行地址号比较; 若 S4RB8=0, 说明该帧不是地址帧, 应丢掉且保持 S4RI=0。在模式 1 中, 如果 S4SM2 位为 0 且 S4REN 位为 1, 接收机处于地址帧筛选被禁止状态。不论收到的 S4RB8 为 0 或 1, 均可使接收到的信息进入 S4BUF, 并使 S4RI=1, 此时 S4RB8 通常为校验位。模式 0 为非多机通信方式, 在这种方式时, 要设置 S4SM2 应为 0。

S4REN: 允许/禁止串口接收控制位

0: 禁止串口接收数据

1: 允许串口接收数据

S4TB8: 当串口 4 使用模式 1 时, S4TB8 为要发送的第 9 位数据, 一般用作校验位或者地址帧/数据帧标志位, 按需要由软件置位或清 0。在模式 0 中, 该位不用。

S4RB8: 当串口 4 使用模式 1 时, S4RB8 为接收到的第 9 位数据, 一般用作校验位或者地址帧/数据帧标志位。在模式 0 中, 该位不用。

S4TI: 串口 4 发送中断请求标志位。在停止位开始发送时由硬件自动将 S4TI 置 1, 向 CPU 发请求中断, 响应中断后 S4TI 必须用软件清零。

S4RI: 串口 4 接收中断请求标志位。串行接收到停止位的中间时刻由硬件自动将 S4RI 置 1, 向 CPU 发中断申请, 响应中断后 S4RI 必须由软件清零。

12.6.2 串口 4 数据寄存器 (S4BUF)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S4BUF	BDH								

S4BUF: 串口 1 数据接收/发送缓冲区。S4BUF 实际是 2 个缓冲器, 读缓冲器和写缓冲器, 两个操作分别对应两个不同的寄存器, 1 个是只写寄存器 (写缓冲器), 1 个是只读寄存器 (读缓冲器)。对 S4BUF 进行读操作, 实际是读取串口接收缓冲区, 对 S4BUF 进行写操作则是触发串口开始发送数据。

12.6.3 串口 4 模式 0, 模式 0 波特率计算公式

串行口 4 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位: 1 位起始位, 8 位数据位 (低位在先) 和 1 位停止位。波特率可变, 可根据需要进行设置波特率。TxD4 为数据发送口, RxD4 为数据接收口, 串行口全双工接受/发送。



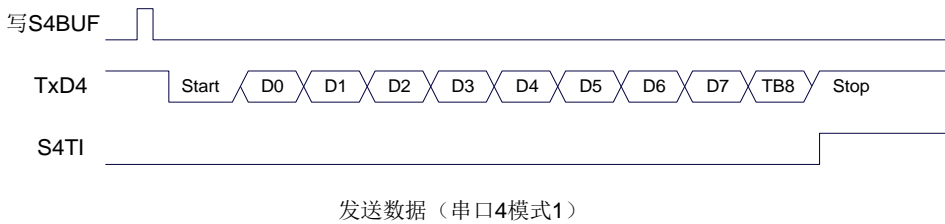
串口 4 的波特率是可变的, 其波特率可由定时器 2 或定时器 4 产生。当定时器采用 1T 模式时 (12 倍速), 相应的波特率的速度也会相应提高 12 倍。

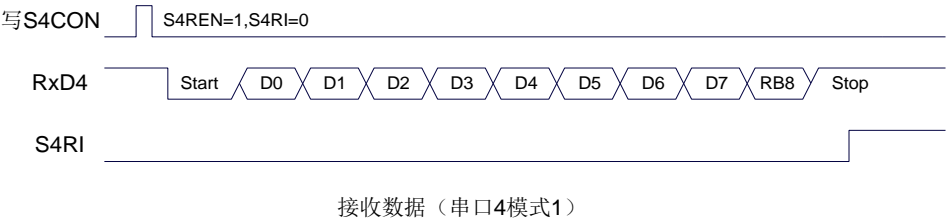
串口 4 模式 0 的波特率计算公式如下表所示: (SYSclk 为系统工作频率)

选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536 - \frac{\text{SYSclk}}{4 \times \text{波特率}}$
	12T	定时器2重载值 = $65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{波特率}}$
定时器4	1T	定时器4重载值 = $65536 - \frac{\text{SYSclk}}{4 \times \text{波特率}}$
	12T	定时器4重载值 = $65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{波特率}}$

12.6.4 串口 4 模式 1, 模式 1 波特率计算公式

串行口 4 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位: 1 位起始位, 9 位数据位 (低位在先) 和 1 位停止位。波特率可变, 可根据需要进行设置波特率。TxD4 为数据发送口, RxD4 为数据接收口, 串行口全双工接受/发送。



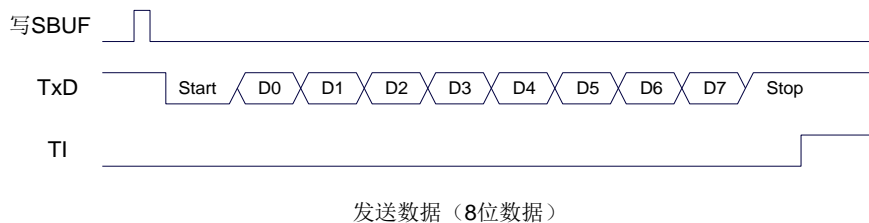


串口 4 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

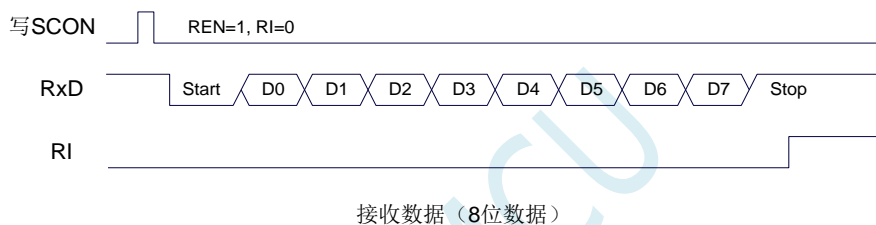
12.7 串口注意事项

关于串口中断请求有如下问题需要注意: (串口 1、串口 2、串口 3、串口 4 均类似, 下面以串口 1 为例进行说明)

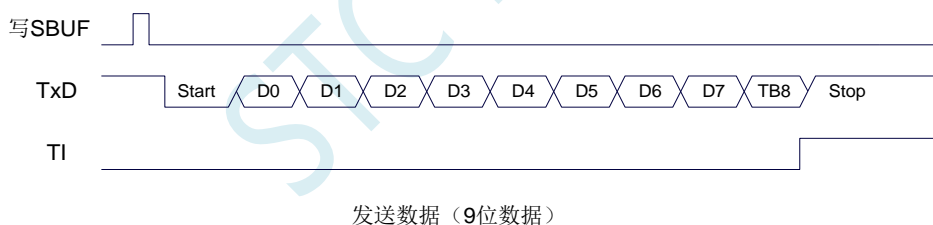
8 位数据模式时, 发送完成整个停止位后产生 **TI** 中断请求, 如下图所示:



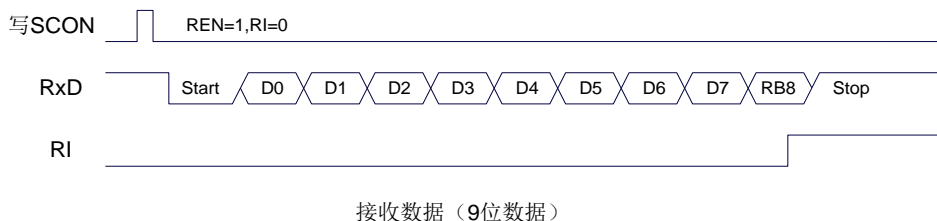
8 位数据模式时, 接收完成一半个停止位后产生 **RI** 中断请求, 如下图所示:



9 位数据模式时, 发送完成整个第 9 位数据位后产生 **TI** 中断请求, 如下图所示:



9 位数据模式时, 接收完成一半个第 9 位数据位后产生 **RI** 中断请求, 如下图所示:



12.8 范例程序

12.8.1 串口 1 使用定时器 2 做波特率发生器

C 语言代码

//测试工作频率为 11.0592MHz

#include "stc16.h" //头文件见附录
#include "intrins.h"

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

bit busy;
char wptr;
char rptr;
char buffer[16];

void UartIsr() interrupt 4

```
{  
    if (TI)  
    {  
        TI = 0;  
        busy = 0;  
    }  
    if (RI)  
    {  
        RI = 0;  
        buffer[wptr++] = SBUF;  
        wptr &= 0x0f;  
    }  
}
```

void UartInit()

```
{  
    SCON = 0x50;  
    T2L = BRT;  
    T2H = BRT >> 8;  
    AUXR = 0x15;  
    wptr = 0x00;  
    rptr = 0x00;  
    busy = 0;  
}
```

void UartSend(char dat)

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

void UartSendStr(char *p)

```
{  
    while (*p)  
    {  
        UartSend(*p++);  
    }
```

```
    }
}

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

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC) ;头文件见附录

BUSY	BIT	20H.0	
WPTR	DATA	21H	
RPTR	DATA	22H	
BUFFER	DATA	23H	;16 bytes

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

```
ORG 0100H
```

UART_ISR:

```
PUSH ACC
PUSH PSW
MOV PSW,#08H

JNB TI,CHKRI
CLR TI
CLR BUSY
```

CHKRI:

```

JNB      RI,UARTISR_EXIT
CLR      RI
MOV      A,WPTR
ANL      A,#0FH
ADD      A,#BUFFER
MOV      R0,A
MOV      @R0,SBUF
INC      WPTR

```

UARTISR_EXIT:

```

POP      PSW
POP      ACC
RETI

```

UART_INIT:

```

MOV      SCON,#50H
MOV      T2L,#0E8H                ;65536-11059200/115200/4=0FFE8H
MOV      T2H,#0FFH
MOV      AUXR,#15H
CLR      BUSY
MOV      WPTR,#00H
MOV      RPTR,#00H
RET

```

UART_SEND:

```

JB      BUSY,$
SETB    BUSY
MOV      SBUF,A
RET

```

UART_SENDSTR:

```

CLR      A
MOVC     A,@A+DPTR
JZ      SENDEND
LCALL    UART_SEND
INC      DPTR
JMP      UART_SENDSTR

```

SENDEND:

```

RET

```

MAIN:

```

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

LCALL    UART_INIT
SETB     ES
SETB     EA

```

```

MOV    DPTR,#STRING
LCALL  UART_SENDSTR

```

LOOP:

```

MOV    A,RPTR
XRL    A,WPTR
ANL    A,#0FH
JZ     LOOP
MOV    A,RPTR
ANL    A,#0FH
ADD    A,#BUFFER
MOV    R0,A
MOV    A,@R0
LCALL  UART_SEND
INC    RPTR
JMP    LOOP

```

```

STRING: DB    'Uart Test !',0DH,0AH,00H

```

```

END

```

12.8.2 串口 1 使用定时器 1（模式 0）做波特率发生器

C 语言代码

//测试工作频率为 11.0592MHz

```

#include "stc16.h"           //头文件见附录
#include "intrins.h"

```

```

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

```

```

bit    busy;
char    wptr;
char    rptr;
char    buffer[16];

```

```

void UartIsr() interrupt 4

```

```

{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

```

```

void UartInit()

```

```

{
    SCON = 0x50;
    TMOD = 0x00;
}

```

```
    TL1 = BRT;
    TH1 = BRT >> 8;
    TRI = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

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

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}

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

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC)

;头文件见附录

```

BUSY      BIT      20H.0
WPTR     DATA     21H
RPTR     DATA     22H
BUFFER   DATA     23H                      ;16 bytes

      ORG      0000H
      LJMP     MAIN
      ORG      0023H
      LJMP     UART_ISR

      ORG      0100H

UART_ISR:
      PUSH     ACC
      PUSH     PSW
      MOV      PSW,#08H

      JNB      TI,CHKRI
      CLR      TI
      CLR      BUSY

CHKRI:
      JNB      RI,UARTISR_EXIT
      CLR      RI
      MOV      A,WPTR
      ANL      A,#0FH
      ADD      A,#BUFFER
      MOV      R0,A
      MOV      @R0,SBUF
      INC      WPTR

UARTISR_EXIT:
      POP      PSW
      POP      ACC
      RETI

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

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

UART_SENDSTR:
      CLR      A
      MOVC     A,@A+DPTR
      JZ       SENDEND
      LCALL    UART_SEND
      INC      DPTR

```

```

        JMP          UART_SENDSTR
SENDEND:
        RET

MAIN:

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

        LCALL       UART_INIT
        SETB        ES
        SETB        EA

        MOV         DPTR, #STRING
        LCALL       UART_SENDSTR

LOOP:

        MOV         A, RPTR
        XRL         A, WPTR
        ANL         A, #0FH
        JZ          LOOP
        MOV         A, RPTR
        ANL         A, #0FH
        ADD         A, #BUFFER
        MOV         R0, A
        MOV         A, @R0
        LCALL       UART_SEND
        INC         RPTR
        JMP         LOOP

STRING:  DB          'Uart Test !', 0DH, 0AH, 00H

        END

```

12.8.3 串口 1 使用定时器 1（模式 2）做波特率发生器

C 语言代码

//测试工作频率为11.0592MHz

```

#include "stc16.h"          //头文件见附录
#include "intrins.h"

#define  FOSC              11059200UL
#define  BRT               (256 - FOSC / 115200 / 32)

bit      busy;

```

```
char    wptr;
char    rptr;
char    buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x20;
    TL1 = BRT;
    TH1 = BRT;
    TRI = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

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

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}

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



```

P5M0 = 0x00;
P5M1 = 0x00;

UartInit();
ES = 1;
EA = 1;
UartSendStr("Uart Test !\r\n");

while (1)
{
    if (rptr != wptr)
    {
        UartSend(buffer[rptr++]);
        rptr &= 0x0f;
    }
}

```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC)

;头文件见附录

BUSY	BIT	20H.0	
WPTR	DATA	21H	
RPTR	DATA	22H	
BUFFER	DATA	23H	;16 bytes

```

ORG    0000H
LJMP   MAIN
ORG    0023H
LJMP   UART_ISR

```

```
ORG    0100H
```

UART_ISR:

```

PUSH   ACC
PUSH   PSW
MOV     PSW,#08H

```

```

JNB     TI,CHKRI
CLR     TI
CLR     BUSY

```

CHKRI:

```

JNB     RI,UARTISR_EXIT
CLR     RI
MOV     A,WPTR
ANL     A,#0FH
ADD     A,#BUFFER
MOV     R0,A
MOV     @R0,SBUF
INC     WPTR

```

UARTISR_EXIT:

```

POP     PSW
POP     ACC
RETI

```

UART_INIT:

```

MOV     SCON,#50H
MOV     TMOD,#20H
MOV     TL1,#0FDH           ;256-11059200/115200/32=0FDH
MOV     TH1,#0FDH
SETB    TR1
MOV     AUXR,#40H
CLR     BUSY
MOV     WPTR,#00H
MOV     RPTR,#00H
RET

```

UART_SEND:

```

JB      BUSY,$
SETB    BUSY
MOV     SBUF,A
RET

```

UART_SENDSTR:

```

CLR     A
MOVC    A,@A+DPTR
JZ      SENDEND
LCALL   UART_SEND
INC     DPTR
JMP     UART_SENDSTR

```

SENDEND:

```
RET
```

MAIN:

```

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

LCALL   UART_INIT
SETB    ES
SETB    EA

MOV     DPTR,#STRING
LCALL   UART_SENDSTR

```

LOOP:

```

MOV     A,RPTR
XRL     A,WPTR
ANL     A,#0FH
JZ      LOOP
MOV     A,RPTR
ANL     A,#0FH
ADD     A,#BUFFER
MOV     R0,A
MOV     A,@R0

```

```

        LCALL      UART_SEND
        INC       RPTR
        JMP       LOOP

STRING:    DB      'Uart Test !,0DH,0AH,00H'

        END

```

12.8.4 串口 2 使用定时器 2 做波特率发生器

C 语言代码

//测试工作频率为11.0592MHz

#include "stc16.h" //头文件见附录

#include "intrins.h"

#define FOSC 11059200UL

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

bit busy;

char wptr;

char rptr;

char buffer[16];

void Uart2Isr() interrupt 8

```

{
    if (S2CON & 0x02)
    {
        S2CON &= ~0x02;
        busy = 0;
    }
    if (S2CON & 0x01)
    {
        S2CON &= ~0x01;
        buffer[wptr++] = S2BUF;
        wptr &= 0x0f;
    }
}

```

void Uart2Init()

```

{
    S2CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

```

void Uart2Send(char dat)

```

{
    while (busy);
    busy = 1;
    S2BUF = dat;
}

```

```
void Uart2SendStr(char *p)
{
    while (*p)
    {
        Uart2Send(*p++);
    }
}

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

    Uart2Init();
    IE2 = 0x01;
    EA = 1;
    Uart2SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart2Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC) ;头文件见附录

<i>BUSY</i>	<i>BIT</i>	<i>20H.0</i>	
<i>WPTR</i>	<i>DATA</i>	<i>21H</i>	
<i>RPTR</i>	<i>DATA</i>	<i>22H</i>	
<i>BUFFER</i>	<i>DATA</i>	<i>23H</i>	<i>;16 bytes</i>
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>0043H</i>	
	<i>LJMP</i>	<i>UART2_ISR</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>UART2_ISR:</i>	<i>PUSH</i>	<i>ACC</i>	

```

        PUSH        PSW
        MOV         PSW,#08H

        MOV         A,S2CON
        JNB         ACC.1,CHKRI
        ANL         S2CON,#NOT 02H
        CLR         BUSY

CHKRI:
        JNB         ACC.0,UART2ISR_EXIT
        ANL         S2CON,#NOT 01H
        MOV         A,WPTR
        ANL         A,#0FH
        ADD         A,#BUFFER
        MOV         R0,A
        MOV         @R0,S2BUF
        INC         WPTR

UART2ISR_EXIT:
        POP         PSW
        POP         ACC
        RETI

UART2_INIT:
        MOV         S2CON,#10H
        MOV         T2L,#0E8H                ;65536-11059200/115200/4=0FFE8H
        MOV         T2H,#0FFH
        MOV         AUXR,#14H
        CLR         BUSY
        MOV         WPTR,#00H
        MOV         RPTR,#00H
        RET

UART2_SEND:
        JB          BUSY,$
        SETB        BUSY
        MOV         S2BUF,A
        RET

UART2_SENDSTR:
        CLR         A
        MOVC        A,@A+DPTR
        JZ          SEND2END
        LCALL       UART2_SEND
        INC         DPTR
        JMP         UART2_SENDSTR

SEND2END:
        RET

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

```

```

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

LCALL    UART2_INIT
MOV      IE2, #01H
SETB     EA

MOV      DPTR, #STRING
LCALL    UART2_SENDSTR

```

LOOP:

```

MOV      A, RPTR
XRL      A, WPTR
ANL      A, #0FH
JZ       LOOP
MOV      A, RPTR
ANL      A, #0FH
ADD      A, #BUFFER
MOV      R0, A
MOV      A, @R0
LCALL    UART2_SEND
INC      RPTR
JMP      LOOP

```

```

STRING:  DB      'Uart Test !', 0DH, 0AH, 00H

END

```

12.8.5 串口 3 使用定时器 2 做波特率发生器

C 语言代码

//测试工作频率为 11.0592MHz

```

#include "stc16.h"          //头文件见附录
#include "intrins.h"

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

bit      busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}

```

```
}

void Uart3Init()
{
    S3CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3Send(*p++);
    }
}

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

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC)

;头文件见附录

```
BUSY      BIT      20H.0
WPTR      DATA    21H
RPTR      DATA    22H
BUFFER    DATA    23H                ;16 bytes
```

```
ORG      0000H
LJMP     MAIN
ORG      008BH
LJMP     UART3_ISR
```

```
ORG      0100H
```

UART3_ISR:

```
PUSH     ACC
PUSH     PSW
MOV       PSW,#08H
```

```
MOV       A,S3CON
JNB       ACC.1,CHKRI
ANL       S3CON,#NOT 02H
CLR       BUSY
```

CHKRI:

```
JNB       ACC.0,UART3ISR_EXIT
ANL       S3CON,#NOT 01H
MOV       A,WPTR
ANL       A,#0FH
ADD       A,#BUFFER
MOV       R0,A
MOV       @R0,S3BUF
INC       WPTR
```

UART3ISR_EXIT:

```
POP       PSW
POP       ACC
RETI
```

UART3_INIT:

```
MOV       S3CON,#10H
MOV       T2L,#0E8H                ;65536-11059200/115200/4=0FFE8H
MOV       T2H,#0FFH
MOV       AUXR,#14H
CLR       BUSY
MOV       WPTR,#00H
MOV       RPTR,#00H
RET
```

UART3_SEND:

```
JB        BUSY,$
SETB     BUSY
MOV       S3BUF,A
RET
```

UART3_SENDSTR:

```
CLR       A
MOVC      A,@A+DPTR
```



```

        JZ          SEND3END
        LCALL       UART3_SEND
        INC         DPTR
        JMP         UART3_SENDSTR
SEND3END:
        RET

MAIN:

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

        LCALL       UART3_INIT
        MOV        IE2, #08H
        SETB       EA

        MOV        DPTR, #STRING
        LCALL       UART3_SENDSTR

LOOP:

        MOV        A, RPTR
        XRL        A, WPTR
        ANL        A, #0FH
        JZ         LOOP
        MOV        A, RPTR
        ANL        A, #0FH
        ADD        A, #BUFFER
        MOV        R0, A
        MOV        A, @R0
        LCALL       UART3_SEND
        INC        RPTR
        JMP        LOOP

STRING:  DB         'Uart Test !', 0DH, 0AH, 00H

        END

```

12.8.6 串口 3 使用定时器 3 做波特率发生器

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"          //头文件见附录
#include "intrins.h"
```

```
#define  FOSC      11059200UL
```

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

```
bit busy;  
char wptr;  
char rptr;  
char buffer[16];
```

```
void Uart3Isr() interrupt 17
```

```
{  
    if (S3CON & 0x02)  
    {  
        S3CON &= ~0x02;  
        busy = 0;  
    }  
    if (S3CON & 0x01)  
    {  
        S3CON &= ~0x01;  
        buffer[wptr++] = S3BUF;  
        wptr &= 0x0f;  
    }  
}
```

```
void Uart3Init()
```

```
{  
    S3CON = 0x50;  
    T3L = BRT;  
    T3H = BRT >> 8;  
    T4T3M = 0x0a;  
    wptr = 0x00;  
    rptr = 0x00;  
    busy = 0;  
}
```

```
void Uart3Send(char dat)
```

```
{  
    while (busy);  
    busy = 1;  
    S3BUF = dat;  
}
```

```
void Uart3SendStr(char *p)
```

```
{  
    while (*p)  
    {  
        Uart3Send(*p++);  
    }  
}
```

```
void main()
```

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

```

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

Uart3Init();
IE2 = 0x08;
EA = 1;
Uart3SendStr("Uart Test !\r\n");

while (1)
{
    if (rptr != wptr)
    {
        Uart3Send(buffer[rptr++]);
        rptr &= 0x0f;
    }
}

```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC)

;头文件见附录

BUSY	BIT	20H.0	
WPTR	DATA	21H	
RPTR	DATA	22H	
BUFFER	DATA	23H	;16 bytes

```

ORG 0000H
LJMP MAIN
ORG 008BH
LJMP UART3_ISR

```

```
ORG 0100H
```

UART3_ISR:

```

PUSH ACC
PUSH PSW
MOV PSW,#08H

```

```

MOV A,S3CON
JNB ACC.1,CHKRI
ANL S3CON,#NOT 02H
CLR BUSY

```

CHKRI:

```

JNB ACC.0,UART3ISR_EXIT
ANL S3CON,#NOT 01H
MOV A,WPTR
ANL A,#0FH
ADD A,#BUFFER
MOV R0,A
MOV @R0,S3BUF
INC WPTR

```

UART3ISR_EXIT:

```

POP PSW
POP ACC
RETI

```

UART3_INIT:

```

MOV      S3CON,#50H
MOV      T3L,#0E8H           ;65536-11059200/115200/4=0FFE8H
MOV      T3H,#0FFH
MOV      T4T3M,#0AH
CLR      BUSY
MOV      WPTR,#00H
MOV      RPTR,#00H
RET

```

UART3_SEND:

```

JB        BUSY,$
SETB     BUSY
MOV      S3BUF,A
RET

```

UART3_SENDSTR:

```

CLR      A
MOVC     A,@A+DPTR
JZ       SEND3END
LCALL    UART3_SEND
INC      DPTR
JMP      UART3_SENDSTR

```

SEND3END:

```
RET
```

MAIN:

```

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

LCALL    UART3_INIT
MOV      IE2,#08H
SETB     EA

MOV      DPTR,#STRING
LCALL    UART3_SENDSTR

```

LOOP:

```

MOV      A,RPTR
XRL      A,WPTR
ANL      A,#0FH
JZ       LOOP
MOV      A,RPTR
ANL      A,#0FH
ADD      A,#BUFFER
MOV      R0,A
MOV      A,@R0

```

```
LCALL    UART3_SEND
INC      RPTR
JMP      LOOP
```

```
STRING:  DB      'Uart Test !',0DH,0AH,00H
```

```
END
```

12.8.7 串口 4 使用定时器 2 做波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz
```

```
#include "stc16.h"           //头文件见附录
```

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

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

```
bit      busy;
char      wptr;
char      rptr;
char      buffer[16];
```

```
void Uart4Isr() interrupt 18
```

```
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}
```

```
void Uart4Init()
```

```
{
    S4CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}
```

```
void Uart4Send(char dat)
```

```
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}
```

```
void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4Send(*p++);
    }
}

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

    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptra != wptra)
        {
            Uart4Send(buffer[rptra++]);
            rptra &= 0x0f;
        }
    }
}
```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC) ;头文件见附录

<i>BUSY</i>	<i>BIT</i>	<i>20H.0</i>	
<i>WPTR</i>	<i>DATA</i>	<i>21H</i>	
<i>RPTR</i>	<i>DATA</i>	<i>22H</i>	
<i>BUFFER</i>	<i>DATA</i>	<i>23H</i>	<i>;16 bytes</i>
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>0093H</i>	
	<i>LJMP</i>	<i>UART4_ISR</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>UART4_ISR:</i>	<i>PUSH</i>	<i>ACC</i>	

```

        PUSH        PSW
        MOV         PSW,#08H

        MOV         A,S4CON
        JNB         ACC.1,CHKRI
        ANL         S4CON,#NOT 02H
        CLR         BUSY

CHKRI:
        JNB         ACC.0,UART4ISR_EXIT
        ANL         S4CON,#NOT 01H
        MOV         A,WPTR
        ANL         A,#0FH
        ADD         A,#BUFFER
        MOV         R0,A
        MOV         @R0,S4BUF
        INC         WPTR

UART4ISR_EXIT:
        POP         PSW
        POP         ACC
        RETI

UART4_INIT:
        MOV         S4CON,#10H
        MOV         T2L,#0E8H
        MOV         T2H,#0FFH
        MOV         AUXR,#14H
        CLR         BUSY
        MOV         WPTR,#00H
        MOV         RPTR,#00H
        RET

UART4_SEND:
        JB          BUSY,$
        SETB        BUSY
        MOV         S4BUF,A
        RET

UART4_SENDSTR:
        CLR         A
        MOV         A,@A+DPTR
        JZ          SEND4END
        LCALL       UART4_SEND
        INC         DPTR
        JMP         UART4_SENDSTR

SEND4END:
        RET

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

```

```

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

LCALL    UART4_INIT
MOV      IE2, #10H
SETB     EA

MOV      DPTR, #STRING
LCALL    UART4_SENDSTR

```

LOOP:

```

MOV      A, RPTR
XRL      A, WPTR
ANL      A, #0FH
JZ       LOOP
MOV      A, RPTR
ANL      A, #0FH
ADD      A, #BUFFER
MOV      R0, A
MOV      A, @R0
LCALL    UART4_SEND
INC      RPTR
JMP      LOOP

```

```

STRING:  DB      'Uart Test !', 0DH, 0AH, 00H

END

```

12.8.8 串口 4 使用定时器 4 做波特率发生器

C 语言代码

//测试工作频率为11.0592MHz

```

#include "stc16.h"          //头文件见附录
#include "intrins.h"

```

```

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

```

```

bit      busy;
char     wptr;
char     rptr;
char     buffer[16];

```

```

void Uart4Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}

```



```
    }  
}  
  
void Uart4Init()  
{  
    S4CON = 0x50;  
    T4L = BRT;  
    T4H = BRT >> 8;  
    T4T3M = 0xa0;  
    wptr = 0x00;  
    rptr = 0x00;  
    busy = 0;  
}  
  
void Uart4Send(char dat)  
{  
    while (busy);  
    busy = 1;  
    S4BUF = dat;  
}  
  
void Uart4SendStr(char *p)  
{  
    while (*p)  
    {  
        Uart4Send(*p++);  
    }  
}  
  
void main()  
{  
    P0M0 = 0x00;  
    P0M1 = 0x00;  
    P1M0 = 0x00;  
    P1M1 = 0x00;  
    P2M0 = 0x00;  
    P2M1 = 0x00;  
    P3M0 = 0x00;  
    P3M1 = 0x00;  
    P4M0 = 0x00;  
    P4M1 = 0x00;  
    P5M0 = 0x00;  
    P5M1 = 0x00;  
  
    Uart4Init();  
    IE2 = 0x10;  
    EA = 1;  
    Uart4SendStr("Uart Test !\r\n");  
  
    while (1)  
    {  
        if (rptr != wptr)  
        {  
            Uart4Send(buffer[rptr++]);  
            rptr &= 0x0f;  
        }  
    }  
}
```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC)

;头文件见附录

```
BUSY      BIT      20H.0
WPTR      DATA    21H
RPTR      DATA    22H
BUFFER    DATA    23H                ;16 bytes
```

```
ORG      0000H
LJMP     MAIN
ORG      0093H
LJMP     UART4_ISR
```

```
ORG      0100H
```

UART4_ISR:

```
PUSH     ACC
PUSH     PSW
MOV       PSW,#08H
```

```
MOV       A,S4CON
JNB       ACC.1,CHKRI
ANL       S4CON,#NOT 02H
CLR       BUSY
```

CHKRI:

```
JNB       ACC.0,UART4ISR_EXIT
ANL       S4CON,#NOT 01H
MOV       A,WPTR
ANL       A,#0FH
ADD       A,#BUFFER
MOV       R0,A
MOV       @R0,S4BUF
INC       WPTR
```

UART4ISR_EXIT:

```
POP       PSW
POP       ACC
RETI
```

UART4_INIT:

```
MOV       S4CON,#50H
MOV       T4L,#0E8H                ;65536-11059200/115200/4=0FFE8H
MOV       T4H,#0FFH
MOV       T4T3M,#0A0H
CLR       BUSY
MOV       WPTR,#00H
MOV       RPTR,#00H
RET
```

UART4_SEND:

```
JB        BUSY,$
SETB     BUSY
MOV       S4BUF,A
RET
```

UART4_SENDSTR:

```
CLR       A
MOVC     A,@A+DPTR
```

```
JZ          SEND4END
LCALL       UART4_SEND
INC         DPTR
JMP         UART4_SENDSTR

SEND4END:
RET

MAIN:

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

LCALL       UART4_INIT
MOV         IE2, #10H
SETB       EA

MOV         DPTR, #STRING
LCALL       UART4_SENDSTR

LOOP:

MOV         A, RPTR
XRL        A, WPTR
ANL        A, #0FH
JZ         LOOP
MOV         A, RPTR
ANL        A, #0FH
ADD        A, #BUFFER
MOV        R0, A
MOV        A, @R0
LCALL       UART4_SEND
INC        RPTR
JMP        LOOP

STRING:    DB          'Uart Test !', 0DH, 0AH, 00H

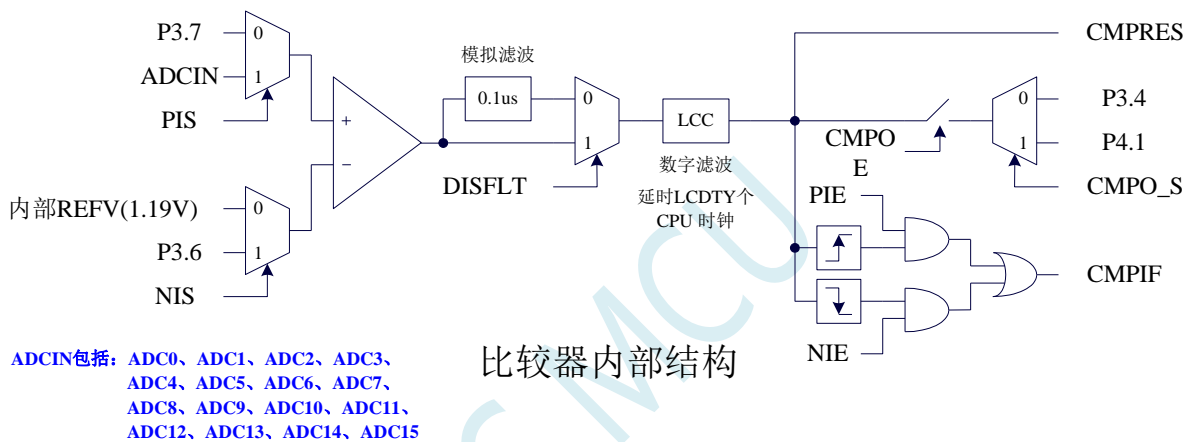
END
```

13 比较器，掉电检测，内部固定比较电压

STC16F 系列单片机内部集成了一个比较器。比较器的正极可以是 P3.7 端口或者 ADC 的模拟输入通道，而负极可以 P3.6 端口或者是内部 BandGap 经过 OP 后的 REFV 电压（内部固定比较电压）。

比较器内部有可程序控制的两级滤波：模拟滤波和数字滤波。模拟滤波可以过滤掉比较输入信号中的毛刺信号，数字滤波可以等待输入信号更加稳定后再进行比较。比较结果可直接通过读取内部寄存器位获得，也可将比较器结果正向或反向输出到外部端口。将比较结果输出到外部端口可用作外部事件的触发信号和反馈信号，可扩大比较的应用范围。

13.1 比较器内部结构图



比较器内部结构

13.2 比较器功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW2	9BH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S

CMPO_S: 比较器输出脚选择位

CMPO_S	CMPO
0	P3.4
1	P4.1

13.3 比较器相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CMPCR1	比较器控制寄存器 1	B4H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
CMPCR2	比较器控制寄存器 2	B5H	INVCMP0	DISFLT	LCDTY[5:0]						0000,0000

13.3.1 比较器控制寄存器 1 (CMPCR1)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	B4H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES

CMPEN: 比较器模块使能位

- 0: 关闭比较功能
- 1: 使能比较功能

CMPIF: 比较器中断标志位。当 PIE 或 NIE 被使能后, 若产生相应的中断信号, 硬件自动将 CMPIF 置 1, 并向 CPU 提出中断请求。此标志位必须用户软件清零。

(注意: 没有使能比较器中断时, 硬件不会设置此中断标志, 即使用查询方式访问比较器时, 不能查询此中断标志)

PIE: 比较器上升沿中断使能位。

- 0: 禁止比较器上升沿中断。
- 1: 使能比较器上升沿中断。使能比较器的比较结果由 0 变成 1 时产生中断请求。

NIE: 比较器下降沿中断使能位。

- 0: 禁止比较器下降沿中断。
- 1: 使能比较器下降沿中断。使能比较器的比较结果由 1 变成 0 时产生中断请求。

PIS: 比较器的正极选择位

- 0: 选择外部端口 P3.7 为比较器正极输入源。
- 1: 通过 ADC_CONTR 中的 ADC_CHS 位选择 ADC 的模拟输入端作为比较器正极输入源。

(注意 1: 当比较器正极选择 ADC 输入通道时, 请务必打开 ADC_CONTR 寄存器中的 ADC 电源控制位 **ADC_POWER** 和 ADC 通道选择位 **ADC_CHS**)

(注意 2: 当需要使用比较器中断唤醒掉电模式/时钟停振模式时, 比较器正极必须选择 P3.7, 不能使用 ADC 输入通道)

NIS: 比较器的负极选择位

- 0: 选择内部 BandGap 经过 OP 后的电压 REFV 作为比较器负极输入源。(芯片在出厂时, 内部参考信号源调整为 1.19V)
- 1: 选择外部端口 P3.6 为比较器负极输入源。

CMPOE: 比较器结果输出控制位

- 0: 禁止比较器结果输出
- 1: 使能比较器结果输出。比较器结果输出到 P3.4 或者 P4.1 (由 P_SW2 中的 CMPO_S 进行设定)

CMPRES: 比较器的比较结果。此位为只读。

- 0: 表示 CMP+的电平低于 CMP-的电平
- 1: 表示 CMP+的电平高于 CMP-的电平

CMPRES 是经过数字滤波后的输出信号, 而不是比较器的直接输出结果。

13.3.2 比较器控制寄存器 2 (CMPCR2)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR2	B5H	INVCMP0	DISFLT	LCDTY[5:0]					

INVCMP0: 比较器结果输出控制

0: 比较器结果正向输出。若 CMPRES 为 0, 则 P3.4/P4.1 输出低电平, 反之输出高电平。

1: 比较器结果反向输出。若 CMPRES 为 0, 则 P3.4/P4.1 输出高电平, 反之输出低电平。

DISFLT: 模拟滤波功能控制

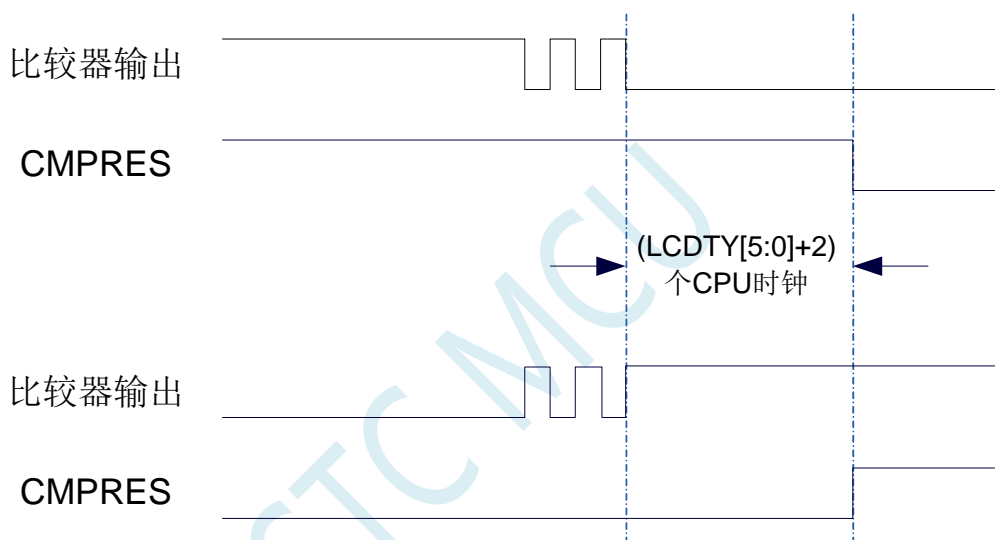
0: 使能 0.1us 模拟滤波功能

1: 关闭 0.1us 模拟滤波功能, 可略微提高比较器的比较速度。

LCDTY[5:0]: 数字滤波功能控制

数字滤波功能即为数字信号去抖动功能。当比较结果发生上升沿或者下降沿变化时, 比较器检测变化后的信号必须维持 LCDTY 所设置的 CPU 时钟数不发生变化, 才认为数据变化是有效的; 否则将视同信号无变化。

注意: 当使能数字滤波功能后, 芯片内部实际的等待时钟需额外增加两个状态机切换时间, 即若 LCDTY 设置为 0 时, 为关闭数字滤波功能; 若 LCDTY 设置为非 0 值 n ($n=1\sim63$) 时, 则实际的数字滤波时间为 $(n+2)$ 个系统时钟



13.4 范例程序

13.4.1 比较器的使用（中断方式）

C 语言代码

//测试工作频率为11.0592MHz

#include "stc16.h"

//头文件见附录

#include "intrins.h"

void CMP_Isr() interrupt 21

```
{
    CMPCR1 &= ~0x40;                //清中断标志
    if (CMPCR1 & 0x01)
    {
        P10 = !P10;                //下降沿中断测试端口
    }
    else
    {
        P11 = !P11;                //上升沿中断测试端口
    }
}
```

void main()

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

    CMPCR2 = 0x00;
    CMPCR2 &= ~0x80;                //比较器正向输出
    // CMPCR2 |= 0x80;                //比较器反向输出
    CMPCR2 &= ~0x40;                //使能0.1us 滤波
    // CMPCR2 |= 0x40;                //禁止0.1us 滤波
    // CMPCR2 &= ~0x3f;                //比较器结果直接输出
    CMPCR2 |= 0x10;                //比较器结果经过16个去抖时钟后输出
    CMPCR1 = 0x00;
    CMPCR1 |= 0x30;                //使能比较器边沿中断
    // CMPCR1 &= ~0x20;                //禁止比较器上升沿中断
    // CMPCR1 |= 0x20;                //使能比较器上升沿中断
    // CMPCR1 &= ~0x10;                //禁止比较器下降沿中断
    // CMPCR1 |= 0x10;                //使能比较器下降沿中断
    CMPCR1 &= ~0x08;                //P3.7 为CMP+输入脚
    // CMPCR1 |= 0x08;                //ADC 输入脚为CMP+输入脚
    // CMPCR1 &= ~0x04;                //内部1.19V 参考信号源为CMP-输入脚
    CMPCR1 |= 0x04;                //P3.6 为CMP-输入脚
    // CMPCR1 &= ~0x02;                //禁止比较器输出
}
```

CMPCR1 /= 0x02;

//使能比较器输出

CMPCR1 /= 0x80;

//使能比较器模块

EA = 1;

while (1);

}

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC)

;头文件见附录

ORG 0000H

LJMP MAIN

ORG 00ABH

LJMP CMPISR

CMPISR: ORG 0100H

PUSH ACC

ANL CMPCR1,#NOT 40H

;清中断标志

MOV A,CMPCR1

JB ACC.0,RSING

FALLING:

CPL P1.0

;下降沿中断测试端口

POP ACC

RETI

RSING:

CPL P1.1

;上升沿中断测试端口

POP ACC

RETI

MAIN:

MOV SP,#5FH

MOV P0M0,#00H

MOV P0M1,#00H

MOV P1M0,#00H

MOV P1M1,#00H

MOV P2M0,#00H

MOV P2M1,#00H

MOV P3M0,#00H

MOV P3M1,#00H

MOV P4M0,#00H

MOV P4M1,#00H

MOV P5M0,#00H

MOV P5M1,#00H

MOV CMPCR2,#00H

ANL CMPCR2,#NOT 80H

;比较器正向输出

; ORL CMPCR2,#80H

;比较器反向输出

ANL CMPCR2,#NOT 40H

;使能 0.1us 滤波

; ORL CMPCR2,#40H

;禁止 0.1us 滤波

; ANL CMPCR2,#NOT 3FH

;比较器结果直接输出

ORL CMPCR2,#10H

;比较器结果经过 16 个去抖时钟后输出

MOV CMPCR1,#00H

ORL CMPCR1,#30H

;使能比较器边沿中断

; ANL CMPCR1,#NOT 20H

;禁止比较器上升沿中断


```

;      ORL      CMPCR1,#20H      ;使能比较器上升沿中断
;      ANL      CMPCR1,#NOT 10H  ;禁止比较器下降沿中断
;      ORL      CMPCR1,#10H      ;使能比较器下降沿中断
;      ANL      CMPCR1,#NOT 08H  ;P3.7 为 CMP+ 输入脚
;      ORL      CMPCR1,#08H      ;ADC 输入脚为 CMP+ 输入脚
;      ANL      CMPCR1,#NOT 04H  ;内部 1.19V 参考信号源为 CMP- 输入脚
;      ORL      CMPCR1,#04H      ;P3.6 为 CMP- 输入脚
;      ANL      CMPCR1,#NOT 02H  ;禁止比较器输出
;      ORL      CMPCR1,#02H      ;使能比较器输出
;      ORL      CMPCR1,#80H      ;使能比较器模块
;      SETB     EA

JMP     $

END

```

13.4.2 比较器的使用（查询方式）

C 语言代码

//测试工作频率为 11.0592MHz

```

#include "stc16.h"
#include "intrins.h"

```

//头文件见附录

```
void main()
```

```
{
```

```
    P0M0 = 0x00;
```

```
    P0M1 = 0x00;
```

```
    P1M0 = 0x00;
```

```
    P1M1 = 0x00;
```

```
    P2M0 = 0x00;
```

```
    P2M1 = 0x00;
```

```
    P3M0 = 0x00;
```

```
    P3M1 = 0x00;
```

```
    P4M0 = 0x00;
```

```
    P4M1 = 0x00;
```

```
    P5M0 = 0x00;
```

```
    P5M1 = 0x00;
```

```
    CMPCR2 = 0x00;
```

```
    CMPCR2 &= ~0x80;
```

```
//    CMPCR2 /= 0x80;
```

```
    CMPCR2 &= ~0x40;
```

```
//    CMPCR2 /= 0x40;
```

```
//    CMPCR2 &= ~0x3f;
```

```
    CMPCR2 /= 0x10;
```

```
    CMPCR1 = 0x00;
```

```
    CMPCR1 /= 0x30;
```

```
//    CMPCR1 &= ~0x20;
```

```
//    CMPCR1 /= 0x20;
```

```
//    CMPCR1 &= ~0x10;
```

```
//    CMPCR1 /= 0x10;
```

```
    CMPCR1 &= ~0x08;
```

```
//    CMPCR1 /= 0x08;
```

```
//    CMPCR1 &= ~0x04;
```

```
    CMPCR1 /= 0x04;
```

//比较器正向输出

//比较器反向输出

//使能 0.1us 滤波

//禁止 0.1us 滤波

//比较器结果直接输出

//比较器结果经过 16 个去抖时钟后输出

//使能比较器边沿中断

//禁止比较器上升沿中断

//使能比较器上升沿中断

//禁止比较器下降沿中断

//使能比较器下降沿中断

//P3.7 为 CMP+ 输入脚

//ADC 输入脚为 CMP+ 输入脚

//内部 1.19V 参考信号源为 CMP- 输入脚

//P3.6 为 CMP- 输入脚

```

//  CMPCR1 &= ~0x02;           //禁止比较器输出
CMPCR1 /= 0x02;               //使能比较器输出
CMPCR1 /= 0x80;               //使能比较器模块

while (1)
{
    P10 = CMPCR1 & 0x01;       //读取比较器比较结果
}

```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC)

;头文件见附录

```

ORG 0000H
LJMP MAIN

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

MOV CMPCR2, #00H
ANL CMPCR2, #NOT 80H           ;比较器正向输出
; ORL CMPCR2, #80H           ;比较器反向输出
ANL CMPCR2, #NOT 40H          ;使能 0.1us 滤波
; ORL CMPCR2, #40H          ;禁止 0.1us 滤波
; ANL CMPCR2, #NOT 3FH      ;比较器结果直接输出
ORL CMPCR2, #10H              ;比较器结果经过 16 个去抖时钟后输出
MOV CMPCR1, #00H
ORL CMPCR1, #30H              ;使能比较器边沿中断
; ANL CMPCR1, #NOT 20H      ;禁止比较器上升沿中断
; ORL CMPCR1, #20H          ;使能比较器上升沿中断
; ANL CMPCR1, #NOT 10H      ;禁止比较器下降沿中断
; ORL CMPCR1, #10H          ;使能比较器下降沿中断
ANL CMPCR1, #NOT 08H          ;P3.7 为 CMP+ 输入脚
; ORL CMPCR1, #08H          ;ADC 输入脚为 CMP+ 输入脚
; ANL CMPCR1, #NOT 04H      ;内部 1.19V 参考信号源为 CMP- 输入脚
ORL CMPCR1, #04H              ;P3.6 为 CMP- 输入脚
; ANL CMPCR1, #NOT 02H      ;禁止比较器输出
ORL CMPCR1, #02H              ;使能比较器输出
ORL CMPCR1, #80H              ;使能比较器模块

LOOP:
MOV A, CMPCR1
MOV C, ACC.0

```

MOV **P1.0,C**
JMP **LOOP**

;读取比较器比较结果

END

13.4.3 比较器的多路复用应用（比较器+ADC 输入通道）

由于比较器的正极可以选择 ADC 的模拟输入通道，因此可以通过多路选择器和分时复用可实现多个比较器的应用。

注意：当比较器正极选择 ADC 输入通道时，请务必打开 ADC_CONTR 寄存器中的 ADC 电源控制位 **ADC_POWER** 和 ADC 通道选择位 **ADC_CHS**

C 语言代码

//测试工作频率为11.0592MHz

#include "stc16.h" //头文件见附录
#include "intrins.h"

void main()

{

P0M0 = 0x00;

P0M1 = 0x00;

P1M0 = 0x00;

P1M1 = 0x00;

P2M0 = 0x00;

P2M1 = 0x00;

P3M0 = 0x00;

P3M1 = 0x00;

P4M0 = 0x00;

P4M1 = 0x00;

P5M0 = 0x00;

P5M1 = 0x00;

P1M0 &= 0xfe;

//设置P1.0 为输入口

P1M1 |= 0x01;

ADC_CONTR = 0x80;

//使能ADC 模块并选择P1.0 为ADC 输入脚

CMPCR2 = 0x00;

CMPCR1 = 0x00;

CMPCR1 |= 0x08;

//ADC 输入脚为CMP+输入脚

CMPCR1 |= 0x04;

//P3.6 为CMP-输入脚

CMPCR1 |= 0x02;

//使能比较器输出

CMPCR1 |= 0x80;

//使能比较器模块

while (1);

}

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC) ;头文件见附录

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

ANL        P1M0, #0FEH      ;设置 P1.0 为输入口
ORL        P1M1, #01H
MOV        ADC_CONTR, #80H  ;使能 ADC 模块并选择 P1.0 为 ADC 输入脚

MOV        CMPCR2, #00H
MOV        CMPCR1, #00H

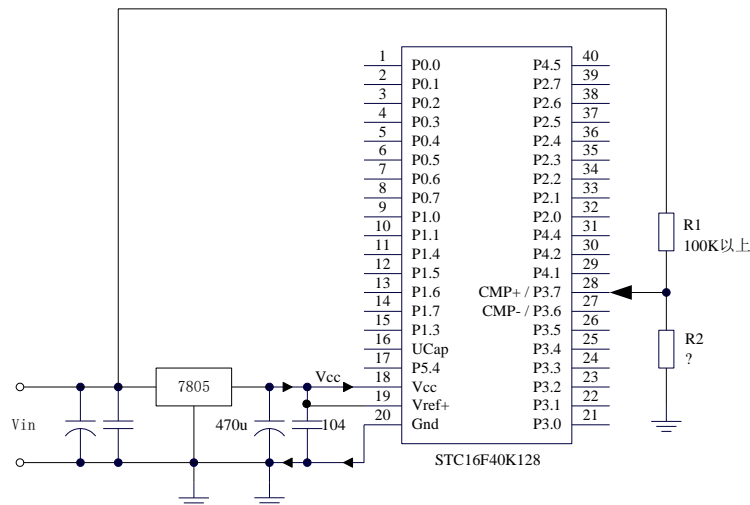
ORL        CMPCR1, #08H      ;ADC 输入脚为 CMP+ 输入脚
ORL        CMPCR1, #04H      ;P3.6 为 CMP- 输入脚
ORL        CMPCR1, #02H      ;使能比较器输出
ORL        CMPCR1, #80H      ;使能比较器模块

LOOP:      JMP          LOOP

END

```

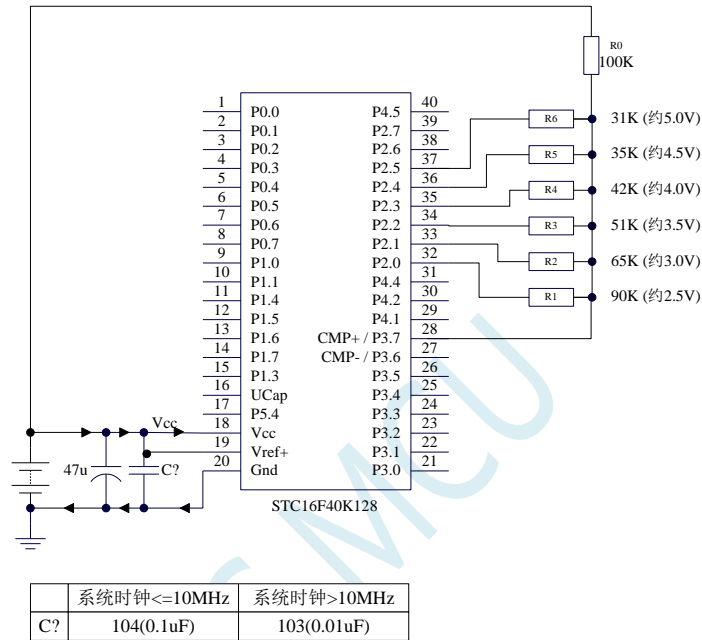
13.4.4 比较器作外部掉电检测（掉电过程中应及时保存用户数据到 EEPROM 中）



上图中电阻 R1 和 R2 对稳压块 7805 的前端电压进行分压，分压后的电压作为比较器 CMP+ 的外部输入与内部 1.19V 参考信号源进行比较。

一般当交流电在 220V 时, 稳压块 7805 前端的直流电压为 11V, 但当交流电压降到 160V 时, 稳压块 7805 前端的直留电压为 8.5V。当稳压块 7805 前端的直留电压低于或等于 8.5V 时, 该前端输入的直留电压被电阻 R1 和 R2 分压到比较器正极输入端 CMP+, CMP+端输入电压低于内部 1.19V 参考信号源, 此时可产生比较器中断, 这样在掉电检测时就有充足的时间将数据保存到 EEPROM 中。当稳压块 7805 前端的直留电压高于 8.5V 时, 该前端输入的直流电压被电阻 R1 和 R2 分压到比较器正极输入端 CMP+, CMP+端输入电压高于内部 1.19V 参考信号源, 此时 CPU 可继续正常工作。

13.4.5 比较器检测工作电压（电池电压）



上图中, 利用电阻分压的原理可以近似的测量出 MCU 的工作电压（选通的通道, MCU 的 IO 口输出低电平, 端口电压值接近 Gnd, 未选通的通道, MCU 的 IO 口输出开漏模式的高, 不影响其他通道）。

比较器的负端选择内部 1.19V 参考信号源, 正端选择通过电阻分压后输入到 CMP+管脚的电压值。

初始化时 P2.5~P2.0 口均设置为开漏模式, 并输出高。首先 P2.0 口输出低电平, 此时若 Vcc 电压低于 2.5V 则比较器的比较值为 0, 反之若 Vcc 电压高于 2.5V 则比较器的比较值为 1;

若确定 Vcc 高于 2.5V, 则将 P2.0 口输出高, P2.1 口输出低电平, 此时若 Vcc 电压低于 3.0V 则比较器的比较值为 0, 反之若 Vcc 电压高于 3.0V 则比较器的比较值为 1;

若确定 Vcc 高于 3.0V, 则将 P2.1 口输出高, P2.2 口输出低电平, 此时若 Vcc 电压低于 3.5V 则比较器的比较值为 0, 反之若 Vcc 电压高于 3.5V 则比较器的比较值为 1;

若确定 Vcc 高于 3.5V, 则将 P2.2 口输出高, P2.3 口输出低电平, 此时若 Vcc 电压低于 4.0V 则比较器的比较值为 0, 反之若 Vcc 电压高于 4.0V 则比较器的比较值为 1;

若确定 Vcc 高于 4.0V, 则将 P2.3 口输出高, P2.4 口输出低电平, 此时若 Vcc 电压低于 4.5V 则比较器的比较值为 0, 反之若 Vcc 电压高于 4.5V 则比较器的比较值为 1;

若确定 Vcc 高于 4.5V, 则将 P2.4 口输出高, P2.5 口输出低电平, 此时若 Vcc 电压低于 5.0V 则比较器的比较值为 0, 反之若 Vcc 电压高于 5.0V 则比较器的比较值为 1。

C 语言代码

//测试工作频率为11.0592MHz

#include "stc16.h" //头文件见附录

#include "intrins.h"

void delay ()

{

 char i;

 for (i=0; i<20; i++);

}

void main()

{

 unsigned char v;

 P0M0 = 0x00;

 P0M1 = 0x00;

 P1M0 = 0x00;

 P1M1 = 0x00;

 P2M0 = 0x00;

 P2M1 = 0x00;

 P3M0 = 0x00;

 P3M1 = 0x00;

 P4M0 = 0x00;

 P4M1 = 0x00;

 P5M0 = 0x00;

 P5M1 = 0x00;

 P2M0 = 0x3f;

//P2.5~P2.0 初始化为开漏模式

 P2M1 = 0x3f;

 P2 = 0xff;

 CMPCR2 = 0x10;

//比较器结果经过16个去抖时钟后输出

 CMPCR1 = 0x00;

 CMPCR1 &= ~0x08;

//P3.7 为CMP+输入脚

 CMPCR1 &= ~0x04;

//内部1.19V 参考信号源为CMP-输入脚

 CMPCR1 &= ~0x02;

//禁止比较器输出

 CMPCR1 |= 0x80;

//使能比较器模块

 while (1)

 {

 v = 0x00;

//电压<2.5V

 P2 = 0xfe;

//P2.0 输出0

 delay();

 if (!(CMPCR1 & 0x01)) goto ShowVol;

 v = 0x01;

//电压>2.5V

 P2 = 0xfd;

//P2.1 输出0

 delay();

 if (!(CMPCR1 & 0x01)) goto ShowVol;

 v = 0x03;

//电压>3.0V

 P2 = 0xfb;

//P2.2 输出0

 delay();

 if (!(CMPCR1 & 0x01)) goto ShowVol;

 v = 0x07;

//电压>3.5V

 P2 = 0xf7;

//P2.3 输出0

 delay();

 if (!(CMPCR1 & 0x01)) goto ShowVol;

```

        v = 0x0f;                                // 电压>4.0V
        P2 = 0xef;                                // P2.4 输出 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x1f;                                // 电压>4.5V
        P2 = 0xdf;                                // P2.5 输出 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x3f;                                // 电压>5.0V
ShowVol:
        P2 = 0xff;
        P0 = ~v;
    }
}

```

汇编代码

; 测试工作频率为 11.0592MHz

\$include (STC16F.INC)

; 头文件见附录

```

        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        P2M0, #00111111B           ; P2.5~P2.0 初始化为开漏模式
        MOV        P2M1, #00111111B
        MOV        P2, #0FFH
        MOV        CMPCR2, #10H               ; 比较器结果经过 16 个去抖时钟后输出
        MOV        CMPCR1, #00H
        ANL        CMPCR1, #NOT 08H           ; P3.7 为 CMP+ 输入脚
        ANL        CMPCR1, #NOT 04H           ; 内部 1.19V 参考信号源为 CMP- 输入脚
        ANL        CMPCR1, #NOT 02H           ; 禁止比较器输出
        ORL        CMPCR1, #80H               ; 使能比较器模块

LOOP:    MOV        R0, #00000000B             ; 电压<2.5V
        MOV        P2, #11111110B             ; P2.0 输出 0
        CALL       DELAY
        MOV        A, CMPCR1
        JNB        ACC.0, SKIP
        MOV        R0, #00000001B             ; 电压>2.5V
        MOV        P2, #11111110B             ; P2.1 输出 0
        CALL       DELAY
        MOV        A, CMPCR1

```

```
JNB      ACC.0,SKIP
MOV      R0,#00000011B      ;电压>3.0V
MOV      P2,#11111011B      ;P2.2 输出 0
CALL     DELAY
MOV      A,CMPCR1
JNB      ACC.0,SKIP
MOV      R0,#00000111B      ;电压>3.5V
MOV      P2,#11110111B      ;P2.3 输出 0
CALL     DELAY
MOV      A,CMPCR1
JNB      ACC.0,SKIP
MOV      R0,#00001111B      ;电压>4.0V
MOV      P2,#11101111B      ;P2.4 输出 0
CALL     DELAY
MOV      A,CMPCR1
JNB      ACC.0,SKIP
MOV      R0,#00011111B      ;电压>4.5V
MOV      P2,#11011111B      ;P2.5 输出 0
CALL     DELAY
MOV      A,CMPCR1
JNB      ACC.0,SKIP
MOV      R0,#00111111B      ;电压>5.0V
SKIP:    MOV      P2,#11111111B
MOV      A,R0
CPL      A
MOV      P0,A      ;P0.5~P0.0 口显示电压
JMP      LOOP

DELAY:   MOV      R0,#20
DJNZ     R0,$
RET

END
```


14 IAP/EEPROM，暂不支持

STC16F 系列单片机内部集成了大容量的 EEPROM。利用 ISP/IAP 技术可将内部 Data Flash 当 EEPROM，擦写次数在 10 万次以上。EEPROM 可分为若干个扇区，每个扇区包含 512 字节。

注意：EEPROM 的写操作只能将字节中的 1 写为 0，当需要将字节中的 0 写为 1，则必须执行扇区擦除操作。EEPROM 的读/写操作是以 1 字节为单位进行，而 EEPROM 擦除操作是以 1 扇区（512 字节）为单位进行，在执行擦除操作时，如果目标扇区中有需要保留的数据，则必须预先将这些数据读取到 RAM 中暂存，待擦除完成后再将保存的数据和需要更新的数据一起再写回 EEPROM/DATA-FLASH。

所以在使用 EEPROM 时，建议同一次修改的数据放在同一个扇区，不是同一次修改的数据放在不同的扇区，不一定要用满。数据存储器的擦除操作是按扇区进行的（每扇区 512 字节）。

EEPROM 可用于保存一些需要在应用过程中修改并且掉电不丢失的参数数据。在用户程序中，可以对 EEPROM 进行字节读/字节编程/扇区擦除操作。在工作电压偏低时，建议不要进行 EEPROM 操作，以免发送数据丢失的情况。

14.1 EEPROM 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
IAP_DATA	IAP 数据寄存器	C2H									1111,1111
IAP_ADDRH	IAP 高地址寄存器	C3H									0000,0000
IAP_ADDRL	IAP 低地址寄存器	C4H									0000,0000
IAP_CMD	IAP 命令寄存器	C5H	-	-	-	-	-	-	CMD[1:0]		xxxx,xx00
IAP_TRIG	IAP 触发寄存器	C6H									0000,0000
IAP_CONTR	IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-	0000,xxxx
IAP_TPS	IAP 等待时间控制寄存器	F7H	-	-	IAPTPS[5:0]						xx00,0000

14.1.1 EEPROM 数据寄存器（IAP_DATA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_DATA	C2H								

在进行 EEPROM 的读操作时，命令执行完成后读出的 EEPROM 数据保存在 IAP_DATA 寄存器中。在进行 EEPROM 的写操作时，在执行写命令前，必须将待写入的数据存放在 IAP_DATA 寄存器中，再发送写命令。擦除 EEPROM 命令与 IAP_DATA 寄存器无关。

14.1.2 EEPROM 地址寄存器（IAP_ADDR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_ADDRH	C3H								
IAP_ADDRL	C4H								

EEPROM 进行读、写、擦除操作的目标地址寄存器。IAP_ADDRH 保存地址的高字节，IAP_ADDRL 保存地址的低字节

14.1.3 EEPROM 命令寄存器 (IAP_CMD)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CMD	C5H	-	-	-	-	-	-	CMD[1:0]	

CMD[1:0]: 发送EEPROM操作命令

00: 空操作

01: 读 EEPROM 命令。读取目标地址所在的 1 字节。

10: 写 EEPROM 命令。写目标地址所在的 1 字节。

11: 擦除 EEPROM。擦除目标地址所在的 1 页（1 扇区/512 字节）。

14.1.4 EEPROM 触发寄存器 (IAP_TRIG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TRIG	C6H								

设置完成 EEPROM 读、写、擦除的命令寄存器、地址寄存器、数据寄存器以及控制寄存器后，需向触发寄存器 IAP_TRIG 依次写入 5AH、A5H（顺序不能交换）两个触发命令来触发相应的读、写、擦除操作。操作完成后，EEPROM 地址寄存器 IAP_ADDRH、IAP_ADDRL 和 EEPROM 命令寄存器 IAP_CMD 的内容不变。如果接下来要对下一个地址的数据进行操作，需手动更新地址寄存器 IAP_ADDRH 和寄存器 IAP_ADDRL 的值。

注意：每次 EEPROM 操作时，都要对 IAP_TRIG 先写入 5AH，再写入 A5H，相应的命令才会生效。写完触发命令后，CPU 会处于 IDLE 等待状态，直到相应的 IAP 操作执行完成后 CPU 才会从 IDLE 状态返回正常状态继续执行 CPU 指令。

14.1.5 EEPROM 控制寄存器 (IAP_CONTR)

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

IAPEN: EEPROM操作使能控制位

0: 禁止 EEPROM 操作

1: 使能 EEPROM 操作

SWBS: 软件复位选择控制位，（需要与SWRST配合使用）

0: 软件复位后从用户代码开始执行程序

1: 软件复位后从系统 ISP 监控代码区开始执行程序

SWRST: 软件复位控制位

0: 无动作

1: 产生软件复位

CMD_FAIL: EEPROM操作失败状态位，需要软件清零

0: EEPROM 操作正确

1: EEPROM 操作失败

14.1.6 EEPROM 擦除等待时间控制寄存器 (IAP_TPS)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TPS	F7H	-	-	IAPTPS[5:0]					

需要根据工作频率进行设置

若工作频率为12MHz,则需要将IAP_TPS设置为12;若工作频率为24MHz,则需要将IAP_TPS设置为24,其他频率以此类推。

STC MCU

15 ADC 模数转换

STC16F 系列单片机内部集成了一个 12 位高速 A/D 转换器（注：第 16 通道只能用于检测内部参考电压，参考电压值出厂时校准为 1.19V，由于制造误差，实际电压值可能在 1.178V~1.202V 之间）。ADC 的时钟频率为系统频率 2 分频再经过用户设置的分频系数进行再次分频（ADC 的时钟频率范围为 SYSclk/2/1~SYSclk/2/16）。

ADC 转换结果的数据格式有两种：左对齐和右对齐。可方便用户程序进行读取和引用。

注意：ADC 的第 15 通道只能用于检测内部参考信号源，参考信号源值出厂时校准为 1.19V，由于制造误差以及测量误差，导致实际的内部参考信号源相比 1.19V，大约有±1%的误差。如果用户需要知道每一颗芯片的准确内部参考信号源值，可外接精准参考信号源，然后利用 ADC 的第 15 通道进行测量标定。

如果芯片有 ADC 的外部参考电源管脚 ADC_VRef+，则一定不能浮空，必须接外部参考电源或者直接连接到 VCC。

15.1 ADC 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
ADC_CONTR	ADC 控制寄存器	DCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]			000x,0000	
ADC_RES	ADC 转换结果高位寄存器	DDH									0000,0000
ADC_RESL	ADC 转换结果低位寄存器	DEH									0000,0000
ADCCFG	ADC 配置寄存器	DFH	-	-	RESFMT	-	SPEED[3:0]			xx0x,0000	

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
ADCTIM	ADC 时序控制寄存器	7EFEABH	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]					0010,1010

15.1.1 ADC 控制寄存器（ADC_CONTR）

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

ADC_POWER: ADC 电源控制位

0: 关闭 ADC 电源

1: 打开 ADC 电源。

建议进入空闲模式和掉电模式前将 ADC 电源关闭，以降低功耗

特别注意：

- 1、给 MCU 的内部 ADC 模块电源打开后，需等待约 1ms，等 MCU 内部的 ADC 电源稳定后再让 ADC 工作；
- 2、适当加长对外部信号的采样时间，就是对 ADC 内部采样保持电容的充电或放电时间，时间够，内部才能和外部电势相等。

ADC_START: ADC 转换启动控制位。写入 1 后开始 ADC 转换，转换完成后硬件自动将此位清零。

0: 无影响。即使 ADC 已经开始转换工作，写 0 也不会停止 A/D 转换。

1: 开始 ADC 转换，转换完成后硬件自动将此位清零。

ADC_FLAG: ADC 转换结束标志位。当 ADC 完成一次转换后，硬件会自动将此位置 1，并向 CPU 提

出中断请求。此标志位必须软件清零。

ADC_CHS[3:0]: ADC 模拟通道选择位

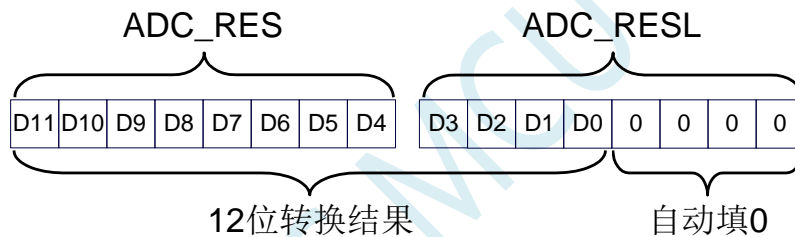
ADC_CHS[3:0]	ADC 通道	ADC_CHS[3:0]	ADC 通道
0000	P1.0	1000	P0.0
0001	P1.1	1001	P0.1
0010	P5.4	1010	P0.2
0011	P1.3	1011	P0.3
0100	P1.4	1100	P0.4
0101	P1.5	1101	P0.5
0110	P1.6	1110	P0.6
0111	P1.7	1111	测试内部 1.19V

15.1.2 ADC 配置寄存器 (ADCCFG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADCCFG	DFH	-	-	RESFMT	-	SPEED[3:0]			

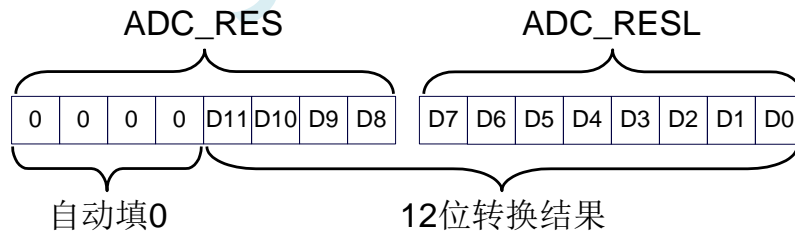
RESFMT: ADC 转换结果格式控制位

0: 转换结果左对齐。ADC_RES 保存结果的高 8 位, ADC_RESL 保存结果的低 4 位。格式如下:



RESFMT=0

1: 转换结果右对齐。ADC_RES 保存结果的高 4 位, ADC_RESL 保存结果的低 8 位。格式如下:



RESFMT=1

SPEED[3:0]: 设置 ADC 时钟 {FADC=SYSclk/2/(SPEED+1)}

SPEED[3:0]	ADC 时钟频率
0000	SYSclk/2/1
0001	SYSclk/2/2
0010	SYSclk/2/3
...	...
1101	SYSclk/2/14
1110	SYSclk/2/15

1111	SYSclk/2/16
------	-------------

15.1.3 ADC 转换结果寄存器 (ADC_RES, ADC_RESL)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	DDH								
ADC_RESL	DEH								

当 A/D 转换完成后, 转换结果会自动保存到 ADC_RES 和 ADC_RESL 中。保存结果的数据格式请参考 ADC_CFG 寄存器中的 RESFMT 设置。

15.1.4 ADC 时序控制寄存器 (ADCTIM)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADCTIM	7EFEABH	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]				

CSSETUP: ADC 通道选择时间控制 Tsetup

CSSETUP	ADC 时钟数
0	1 (默认值)
1	2

CSHOLD[1:0]: ADC 通道选择保持时间控制 Thold

CSHOLD[1:0]	ADC 时钟数
00	1
01	2 (默认值)
10	3
11	4

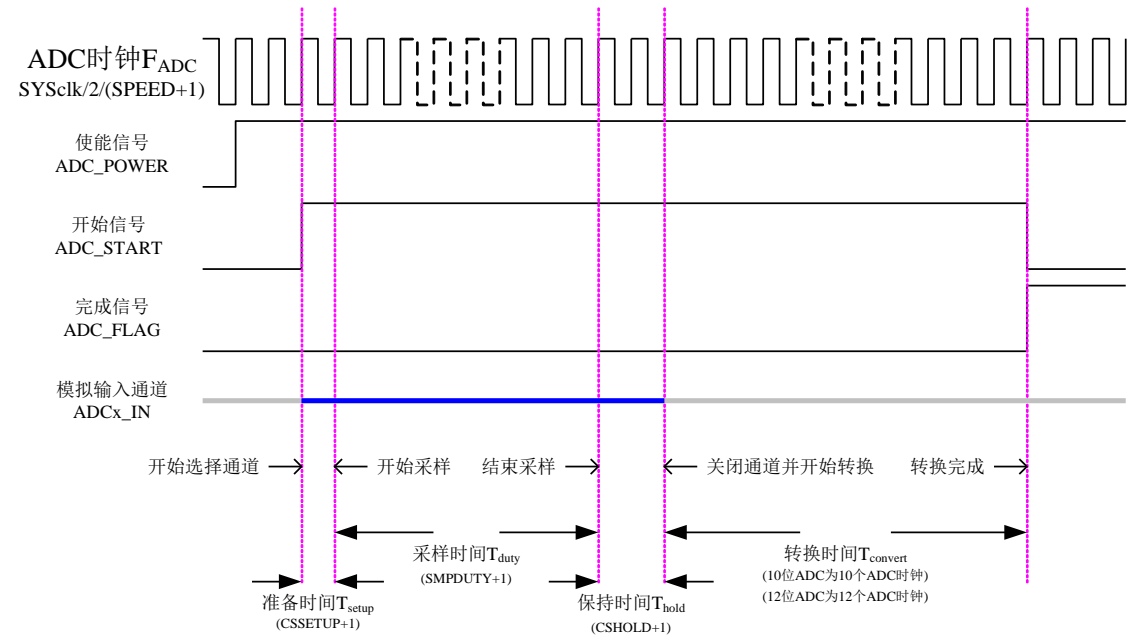
SMPDUTY[4:0]: ADC 模拟信号采样时间控制 Tduty (注意: SMPDUTY 一定不能设置小于 01010B)

SMPDUTY[4:0]	ADC 时钟数
00000	1
00001	2
...	...
01010	11 (默认值)
...	...
11110	31
11111	32

ADC 数模转换时间: T_{convert}

12 位 ADC 的转换时间固定为 12 个 ADC 工作时钟

一个完整的 ADC 转换时间为: $T_{\text{setup}} + T_{\text{duty}} + T_{\text{hold}} + T_{\text{convert}}$, 如下图所示



ADC整体转换时序图

15.2 ADC 静态特性

符号	描述	最小值	典型值	最大值	单位
RES	分辨率	-	12	-	Bits
E _T	整体误差	-	0.5	1	LSB
E _O	偏移误差	-	-0.1	1	LSB
E _G	增益误差	-	0	1	LSB
E _D	微分非线性误差	-	0.7	1.5	LSB
E _I	积分非线性误差	-	1	2	LSB
R _{AIN}	通道等效电阻	-	∞	-	ohm
R _{ESD}	采样保持电容前串接的抗静电电阻	-	700	-	ohm
C _{ADC}	内部采样保持电容	-	16.5	-	pF

15.3 ADC 相关计算公式

15.3.1 ADC 速度计算公式

ADC 的转换速度由 ADCCFG 寄存器中的 SPEED 和 ADCTIM 寄存器共同控制。转换速度的计算公式如下所示：

$$12\text{位ADC转换速度} = \frac{\text{MCU工作频率SYSclk}}{2 \times (\text{SPEED}[3:0] + 1) \times [(\text{CSSETUP} + 1) + (\text{CSHOLD} + 1) + (\text{SMPDUTY} + 1) + 12]}$$

注意：

- 12 位 ADC 的速度不能高于 800KHz
- SMPDUTY 的值不能小于 10，建议设置为 15
- CSSETUP 可使用上电默认值 0
- CHOLD 可使用上电默认值 1（ADCTIM 建议设置为 3FH）

15.3.2 ADC 转换结果计算公式

$$12\text{位ADC转换结果} = 4096 \times \frac{\text{ADC被转换通道的输入电压Vin}}{\text{ADC外部参考源的电压}} \quad (\text{有独立ADC_Vref+管脚})$$

15.3.3 反推 ADC 输入电压计算公式

$$\text{ADC被转换通道的输入电压Vin} = \text{ADC外部参考源的电压} \times \frac{12\text{位ADC转换结果}}{4096} \quad (\text{有独立ADC_Vref+管脚})$$

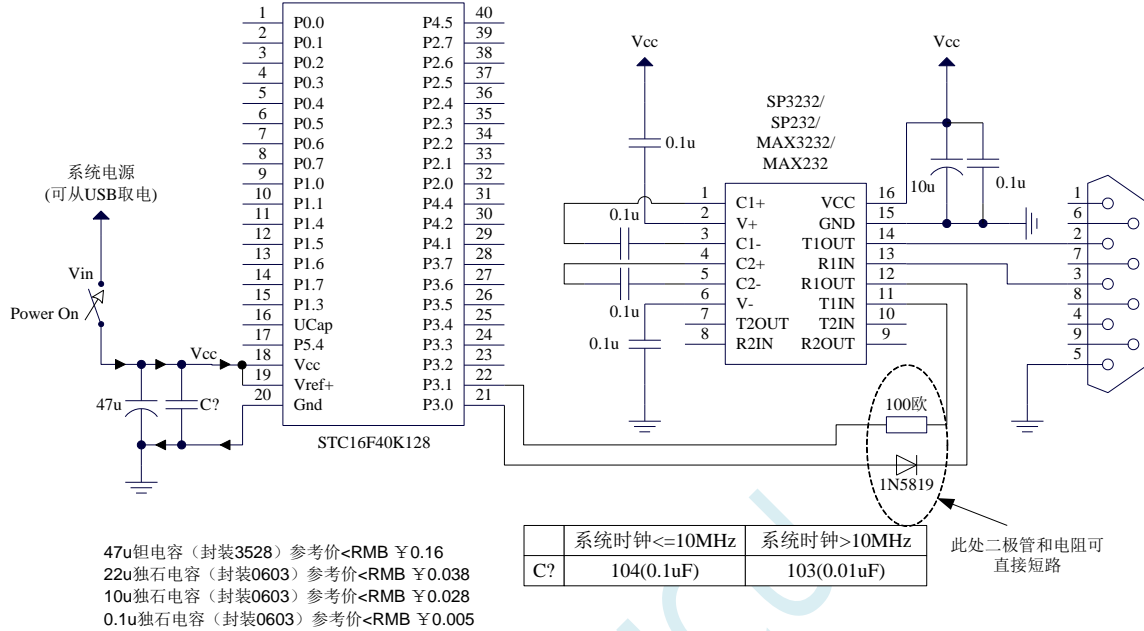
15.3.4 反推工作电压计算公式

当需要使用 ADC 输入电压和 ADC 转换结果反推工作电压时，若目标芯片无独立的 ADC_Vref+管脚，则可直接测量并使用下面公式，若目标芯片有独立 ADC_Vref+管脚时，则必须将 ADC_Vref+管脚连接到 Vcc 管脚。

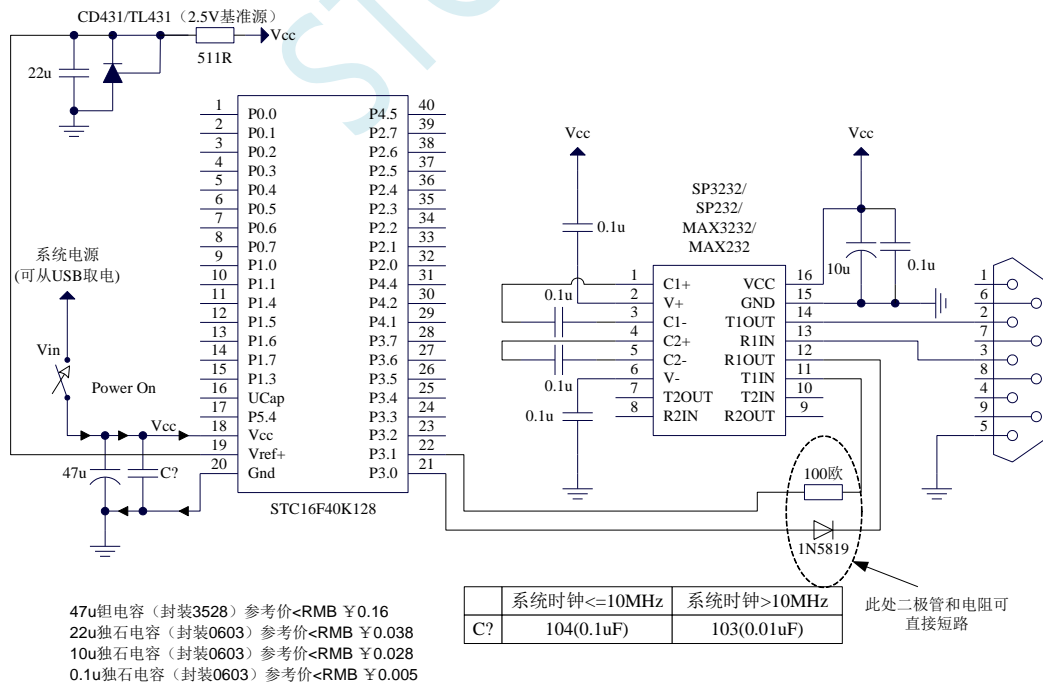
$$\text{MCU工作电压Vcc} = 4096 \times \frac{\text{ADC被转换通道的输入电压Vin}}{\text{12位ADC转换结果}}$$

15.4 ADC 应用参考线路图

15.4.1 一般精度 ADC 参考线路图



15.4.2 高精度 ADC 参考线路图



15.5 范例程序

15.5.1 ADC 基本操作（查询方式）

C 语言代码

//测试工作频率为11.0592MHz

#include "stc16.h" //头文件见附录
#include "intrins.h"

void main()

```
{  
    P0M0 = 0x00;  
    P0M1 = 0x00;  
    P1M0 = 0x00;  
    P1M1 = 0x00;  
    P2M0 = 0x00;  
    P2M1 = 0x00;  
    P3M0 = 0x00;  
    P3M1 = 0x00;  
    P4M0 = 0x00;  
    P4M1 = 0x00;  
    P5M0 = 0x00;  
    P5M1 = 0x00;  
  
    P1M0 = 0x00; //设置P1.0 为ADC 口  
    P1M1 = 0x01;  
    P_SW2 /= 0x80;  
    ADCTIM = 0x3f; //设置ADC 内部时序  
    P_SW2 &= 0x7f;  
    ADCCFG = 0x0f; //设置ADC 时钟为系统时钟/2/16/16  
    ADC_CONTR = 0x80; //使能ADC 模块  
  
    while (1)  
    {  
        ADC_CONTR /= 0x40; //启动AD 转换  
        _nop_();  
        _nop_();  
        while (!(ADC_CONTR & 0x20)); //查询ADC 完成标志  
        ADC_CONTR &= ~0x20; //清完成标志  
        P2 = ADC_RES; //读取ADC 结果  
    }  
}
```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC) ;头文件见附录

```
ORG 0000H  
LJMP MAIN  
  
ORG 0100H  
MAIN:  
MOV SP, #5FH
```

```

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

MOV      P1M0, #00H          ;设置 P1.0 为 ADC 口
MOV      P1M1, #01H
MOV      P_SW2, #80H
MOV      A, #3FH
MOV      WR6, #WORD0 ADCTIM  ;设置 ADC 内部时序
MOV      WR4, #WORD2 ADCTIM
MOV      @DR4, R11
MOV      P_SW2, #00H
MOV      ADCCFG, #0FH        ;设置 ADC 时钟为系统时钟/2/16/16
MOV      ADC_CONTR, #80H     ;使能 ADC 模块

LOOP:
    ORL      ADC_CONTR, #40H  ;启动 AD 转换
    NOP
    NOP
    MOV      A, ADC_CONTR     ;查询 ADC 完成标志
    JNB      ACC.5, $-2
    ANL      ADC_CONTR, #NOT 20H ;清完成标志
    MOV      P2, ADC_RES      ;读取 ADC 结果

    SJMP     LOOP

END

```

15.5.2 ADC 基本操作（中断方式）

C 语言代码

//测试工作频率为 11.0592MHz

```

#include "stc16.h"          //头文件见附录
#include "intrins.h"

```

```

void ADC_Isr() interrupt 5
{
    ADC_CONTR &= ~0x20;      //清中断标志
    P2 = ADC_RES;            //读取 ADC 结果
    ADC_CONTR |= 0x40;       //继续 AD 转换
}

```

```

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

```

```

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

P1M0 = 0x00;                //设置 P1.0 为 ADC 口
P1M1 = 0x01;
P_SW2 /= 0x80;
ADCTIM = 0x3f;              //设置 ADC 内部时序
P_SW2 &= 0x7f;
ADCCFG = 0x0f;              //设置 ADC 时钟为系统时钟/2/16/16
ADC_CONTR = 0x80;           //使能 ADC 模块
EADC = 1;                   //使能 ADC 中断
EA = 1;
ADC_CONTR /= 0x40;          //启动 AD 转换

while (1);
}

```

汇编代码

;测试工作频率为 11.0592MHz;

\$include (STC16F.INC)

;头文件见附录

```

ORG      0000H
LJMP     MAIN
ORG      002BH
LJMP     ADCISR

ORG      0100H
ADCISR:
ANL      ADC_CONTR,#NOT 20H    ;清完成标志
MOV      P2,ADC_RES           ;读取 ADC 结果
ORL      ADC_CONTR,#40H       ;继续 AD 转换
RETI

```

MAIN:

```

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

MOV      P1M0,#00H            ;设置 P1.0 为 ADC 口
MOV      P1M1,#01H
MOV      P_SW2,#80H

```

```

MOV      A,#3FH
MOV      WR6,#WORD0 ADCTIM      ;设置ADC 内部时序
MOV      WR4,#WORD2 ADCTIM
MOV      @DR4,R11
MOV      P_SW2,#00H
MOV      ADCCFG,#0FH            ;设置ADC 时钟为系统时钟/2/16/16
MOV      ADC_CONTR,#80H        ;使能ADC 模块
SETB     EADC                   ;使能ADC 中断
SETB     EA
ORL       ADC_CONTR,#40H        ;启动AD 转换

SJMP     $

END

```

15.5.3 格式化 ADC 转换结果

C 语言代码

//测试工作频率为11.0592MHz

```

#include "stc16.h"           //头文件见附录
#include "intrins.h"

```

```

void main()
{

```

```

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

```

```

    P1M0 = 0x00;                //设置P1.0 为ADC 口

```

```

    P1M1 = 0x01;

```

```

    P_SW2 /= 0x80;

```

```

    ADCTIM = 0x3f;              //设置ADC 内部时序

```

```

    P_SW2 &= 0x7f;

```

```

    ADCCFG = 0x0f;              //设置ADC 时钟为系统时钟/2/16/16

```

```

    ADC_CONTR = 0x80;           //使能ADC 模块

```

```

    ADC_CONTR /= 0x40;          //启动AD 转换

```

```

    _nop_();

```

```

    _nop_();

```

```

    while (!(ADC_CONTR & 0x20)); //查询ADC 完成标志

```

```

    ADC_CONTR &= ~0x20;         //清完成标志

```

```

    ADCCFG = 0x00;              //设置结果左对齐

```

```

    ACC = ADC_RES;              //A 存储ADC 的12 位结果的高8 位

```

```

    B = ADC_RES;                //B[7:4]存储ADC 的12 位结果的低4 位,B[3:0]为0

```

```

//  ADCCFG = 0x20;

```

```

//设置结果右对齐

```

```

//  ACC = ADC_RES;

```

```

//A[3:0]存储ADC 的12 位结果的高4 位,A[7:4]为0

```

```
// B = ADC_RES;
```

```
// B 存储 ADC 的 12 位结果的低 8 位
```

```
while (1);
```

```
}
```

汇编代码

```
;测试工作频率为 11.0592MHz
```

```
$include (STC16F.INC)
```

```
;头文件见附录
```

```
ORG 0000H
LJMP MAIN
```

```
ORG 0100H
```

```
MAIN:
```

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

```
MOV P1M0, #00H ;设置 P1.0 为 ADC 口
```

```
MOV P1M1, #01H
```

```
MOV P_SW2, #80H
```

```
MOV A, #3FH
```

```
MOV WR6, #WORD0 ADCTIM ;设置 ADC 内部时序
```

```
MOV WR4, #WORD2 ADCTIM
```

```
MOV @DR4, R11
```

```
MOV P_SW2, #00H
```

```
MOV ADCCFG, #0FH ;设置 ADC 时钟为系统时钟/2/16/16
```

```
MOV ADC_CONTR, #80H ;使能 ADC 模块
```

```
ORL ADC_CONTR, #40H ;启动 AD 转换
```

```
NOP
```

```
NOP
```

```
MOV A, ADC_CONTR ;查询 ADC 完成标志
```

```
JNB ACC.5, $-2
```

```
ANL ADC_CONTR, #NOT 20H ;清完成标志
```

```
MOV ADCCFG, #00H ;设置结果左对齐
```

```
MOV A, ADC_RES ;A 存储 ADC 的 12 位结果的高 8 位
```

```
MOV B, ADC_RES ;B[7:4] 存储 ADC 的 12 位结果的低 4 位, B[3:0] 为 0
```

```
; MOV ADCCFG, #20H ;设置结果右对齐
```

```
; MOV A, ADC_RES ;A[3:0] 存储 ADC 的 12 位结果的高 4 位, A[7:4] 为 0
```

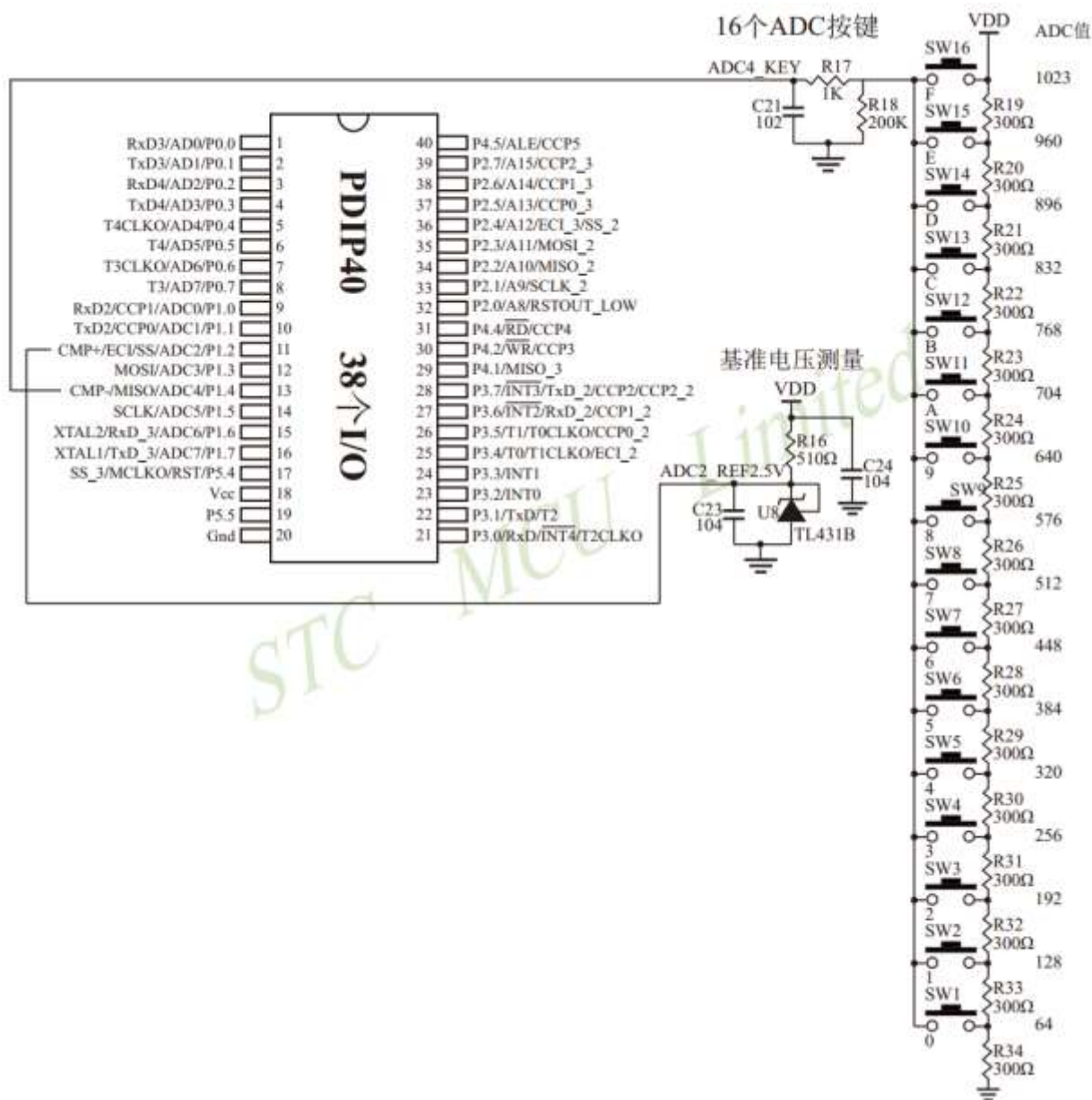
```
; MOV B, ADC_RES ;B 存储 ADC 的 12 位结果的低 8 位
```

```
SJMP $
```

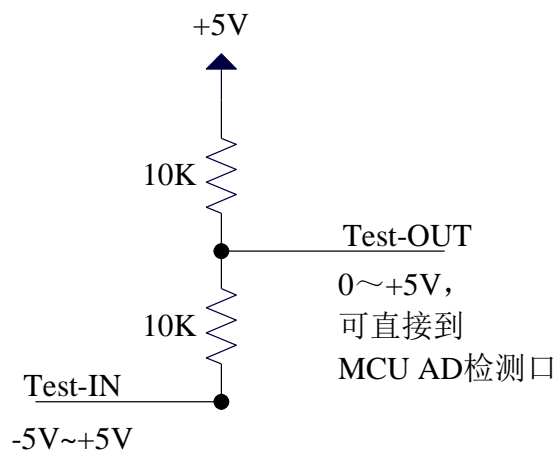
```
END
```

15.5.4 ADC 作按键扫描应用线路图

读 ADC 键的方法：每隔 10ms 左右读一次 ADC 值，并且保存最后 3 次的读数，其变化比较小时再判断键。判断键有效时，允许一定的偏差，比如 ± 16 个字的偏差。

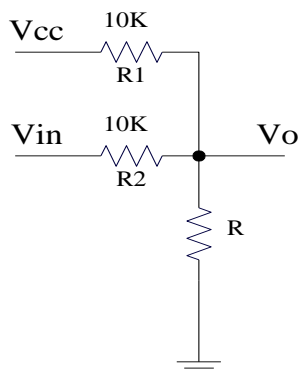


15.5.5 检测负电压参考线路图

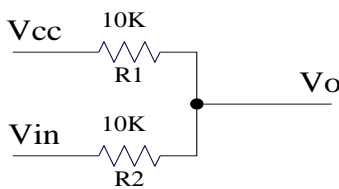


负压转换电路

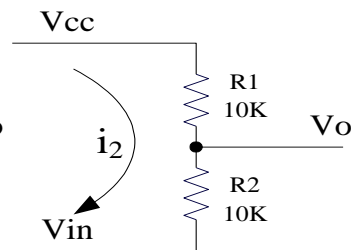
15.5.6 常用加法电路在 ADC 中的应用



常用加法电路



简化加法电路



变形成为分压电路形式

参照分压电路得到公式 1

公式 1: $V_o = V_{in} + i_2 \cdot R_2$

公式 2: $i_2 = (V_{cc} - V_{in}) / (R_1 + R_2)$ {条件: 流向 V_o 的电流 ≈ 0 }

将 $R_1=R_2$ 代入公式 2 得公式 3

公式 3: $i_2 = (V_{cc} - V_{in}) / 2R_2$

将公式 3 代入公式 1 得公式 4

公式 4: $V_o = (V_{cc} + V_{in}) / 2$

根据公式 4, 可以将以上电路看成加法电路。

在单片机的模数转换测量中, 要求被测电压大于 0 并且小于 V_{cc} 。如果被测电压小于 0V, 可以利用加法电路将被测电压提升到 0V 以上。此时对被测电压的变化范围有一定的要求:

把上述条件代入公式 4 可得到下面 2 式

$(V_{cc} + V_{in}) / 2 > 0$ 即 $V_{in} > -V_{cc}$

$(V_{cc} + V_{in}) / 2 < V_{cc}$ 即 $V_{in} < V_{cc}$

上面 2 式可以合起来: **$-V_{cc} < V_{in} < V_{cc}$**

16 同步串行外设接口 SPI

STC16F 系列单片机内部集成了一种高速串行通信接口——SPI 接口。SPI 是一种全双工的高速同步通信总线。STC16F 系列集成的 SPI 接口提供了两种操作模式：主模式和从模式。

16.1 SPI 功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	9AH	S1_S[1:0]		CAN_S[1:0]		SPI_S[1:0]		LIN_S[1:0]	

SPI_S[1:0]: SPI 功能脚选择位

SPI_S[1:0]	SS	MOSI	MISO	SCLK
00	P5.4	P1.3	P1.4	P1.5
01	P2.2	P2.3	P2.4	P2.5
10	P5.4	P4.0	P4.1	P4.3
11	P3.5	P3.4	P3.3	P3.2

16.2 SPI 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
SPSTAT	SPI 状态寄存器	B9H	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
SPCTL	SPI 控制寄存器	BAH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]		0000,0100
SPDAT	SPI 数据寄存器	BBH									0000,0000

16.2.1 SPI 状态寄存器（SPSTAT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	B9H	SPIF	WCOL	-	-	-	-	-	-

SPIF: SPI 中断标志位。

当发送/接收完成 1 字节的数据后，硬件自动将此位置 1，并向 CPU 提出中断请求。当 SSIG 位被设置为 0 时，由于 SS 管脚电平的变化而使得设备的主/从模式发生改变时，此标志位也会被硬件自动置 1，以标志设备模式发生变化。

注意：此标志位必须用户通过软件方式向此位写 1 进行清零。

WCOL: SPI 写冲突标志位。

当 SPI 在进行数据传输的过程中写 SPDAT 寄存器时，硬件将此位置 1。

注意：此标志位必须用户通过软件方式向此位写 1 进行清零。

16.2.2 SPI 控制寄存器（SPCTL），SPI 速度控制

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPCTL	BAH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]	

SSIG: SS 引脚功能控制位

0: SS 引脚确定器件是主机还是从机

1: 忽略 SS 引脚功能，使用 MSTR 确定器件是主机还是从机

SPEN: SPI 使能控制位

0: 关闭 SPI 功能

- 1: 使能 SPI 功能
- DORD: SPI 数据位发送/接收的顺序
- 0: 先发送/接收数据的高位 (MSB)
- 1: 先发送/接收数据的低位 (LSB)
- MSTR: 器件主/从模式选择位
- 设置主机模式:
- 若 SSIG=0, 则 SS 管脚必须为高电平且设置 MSTR 为 1
- 若 SSIG=1, 则只需要设置 MSTR 为 1 (忽略 SS 管脚的电平)
- 设置从机模式:
- 若 SSIG=0, 则 SS 管脚必须为低电平 (与 MSTR 位无关)
- 若 SSIG=1, 则只需要设置 MSTR 为 0 (忽略 SS 管脚的电平)
- CPOL: SPI 时钟极性控制
- 0: SCLK 空闲时为低电平, SCLK 的前时钟沿为上升沿, 后时钟沿为下降沿
- 1: SCLK 空闲时为高电平, SCLK 的前时钟沿为下降沿, 后时钟沿为上升沿
- CPHA: SPI 时钟相位控制
- 0: 数据 SS 管脚为低电平驱动第一位数据并在 SCLK 的后时钟沿改变数据, 前时钟沿采样数据 (必须 SSIG=0)
- 1: 数据在 SCLK 的前时钟沿驱动, 后时钟沿采样
- SPR[1:0]: SPI 时钟频率选择

SPR[1:0]	SCLK 频率
00	SYSclk/4
01	SYSclk/8
10	SYSclk/16
11	SYSclk/32

16.2.3 SPI 数据寄存器 (SPDAT)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPDAT	BBH								

SPI 发送/接收数据缓冲器。

16.3 SPI 通信方式

SPI 的通信方式通常有 3 种：单主单从（一个主机设备连接一个从机设备）、互为主从（两个设备连接，设备和互为主机和从机）、单主多从（一个主机设备连接多个从机设备）

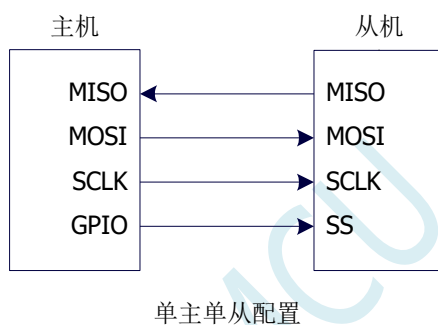
16.3.1 单主单从

两个设备相连，其中一个设备固定作为主机，另外一个固定作为从机。

主机设置：SSIG 设置为 1，MSTR 设置为 1，固定为主机模式。主机可以使用任意端口连接从机的 SS 管脚，拉低从机的 SS 脚即使能从机

从机设置：SSIG 设置为 0，SS 管脚作为从机的片选信号。

单主单从连接配置图如下所示：



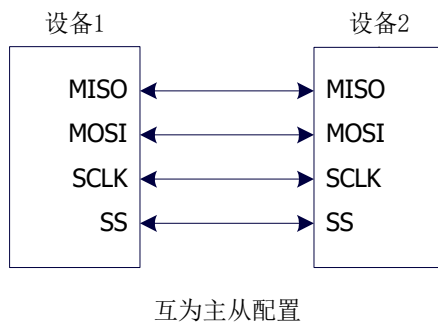
16.3.2 互为主从

两个设备相连，主机和从机不固定。

设置方法 1：两个设备初始化时都设置为 SSIG 设置为 0，MSTR 设置为 1，且将 SS 脚设置为双向口模式输出高电平。此时两个设备都是不忽略 SS 的主机模式。当其中一个设备需要启动传输时，可将自己的 SS 脚设置为输出模式并输出低电平，拉低对方的 SS 脚，这样另一个设备就被强行设置为从机模式了。

设置方法 2：两个设备初始化时都将自己设置成忽略 SS 的从机模式，即将 SSIG 设置为 1，MSTR 设置为 0。当其中一个设备需要启动传输时，先检测 SS 管脚的电平，如果时候高电平，就将自己设置成忽略 SS 的主模式，即可进行数据传输了。

互为主从连接配置图如下所示：



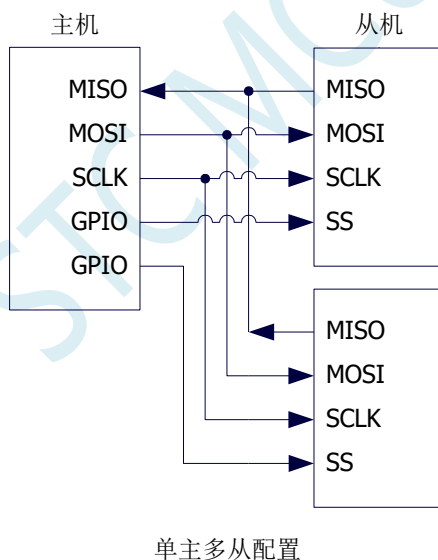
16.3.3 单主多从

多个设备相连，其中一个设备固定作为主机，其他设备固定作为从机。

主机设置：SSIG 设置为 1，MSTR 设置为 1，固定为主机模式。主机可以使用任意端口分别连接各个从机的 SS 管脚，拉低其中一个从机的 SS 脚即可使能相应的从机设备

从机设置：SSIG 设置为 0，SS 管脚作为从机的片选信号。

单主多从连接配置图如下所示：



16.4 配置 SPI

控制位			通信端口				说明
SPEN	SSIG	MSTR	SS	MISO	MOSI	SCLK	
0	x	x	x	输入	输入	输入	关闭 SPI 功能, SS/MOSI/MISO/SCLK 均为普通 IO
1	0	0	0	输出	输入	输入	从机模式, 且被选中
1	0	0	1	高阻	输入	输入	从机模式, 但未被选中
1	0	1→0	0	输出	输入	输入	从机模式, 不忽略 SS 且 MSTR 为 1 的主机模式, 当 SS 管脚被拉低时, MSTR 将被硬件自动清零, 工作模式将被被动设置为从机模式
1	0	1	1	输入	高阻	高阻	主机模式, 空闲状态
					输出	输出	主机模式, 激活状态
1	1	0	x	输出	输入	输入	从机模式
1	1	1	x	输入	输出	输出	主机模式

从机模式的注意事项:

当 CPHA=0 时, SSIG 必须为 0 (即不能忽略 SS 脚)。在每次串行字节开始还发送前 SS 脚必须拉低, 并且在串行字节发送完后须重新设置为高电平。SS 管脚为低电平时不能对 SPDAT 寄存器执行写操作, 否则将导致一个写冲突错误。CPHA=0 且 SSIG=1 时的操作未定义。

当 CPHA=1 时, SSIG 可以置 1 (即可以忽略脚)。如果 SSIG=0, SS 脚可在连续传输之间保持低有效 (即一直固定为低电平)。这种方式适用于固定单主单从的系统。

主机模式的注意事项:

在 SPI 中, 传输总是由主机启动的。如果 SPI 使能 (SPEN=1) 并选择作为主机时, 主机对 SPI 数据寄存器 SPDAT 的写操作将启动 SPI 时钟发生器和数据的传输。在数据写入 SPDAT 之后的半个到一个 SPI 位时间后, 数据将出现在 MOSI 脚。写入主机 SPDAT 寄存器的数据从 MOSI 脚移出发送到从机的 MOSI 脚。同时从机 SPDAT 寄存器的数据从 MISO 脚移出发送到主机的 MISO 脚。

传输完一个字节后, SPI 时钟发生器停止, 传输完成标志 (SPIF) 置位, 如果 SPI 中断使能则会产生一个 SPI 中断。主机和从机 CPU 的两个移位寄存器可以看作是一个 16 位循环移位寄存器。当数据从主机移位传送到从机的同时, 数据也以相反的方向移入。这意味着在一个移位周期中, 主机和从机的数据相互交换。

通过 SS 改变模式

如果 SPEN=1, SSIG=0 且 MSTR=1, SPI 使能为主机模式, 并将 SS 脚可配置为输入模式化或准双向口模式。这种情况下, 另外一个主机可将该脚驱动为低电平, 从而将该器件选择为 SPI 从机并向其发送数据。为了避免争夺总线, SPI 系统将该从机的 MSTR 清零, MOSI 和 SCLK 强制变为输入模式, 而 MISO 则变为输出模式, 同时 SPSTAT 的 SPIF 标志位置 1。

用户软件必须一直对 MSTR 位进行检测, 如果该位被一个从机选择动作而被动清零, 而用户想继续将 SPI 作为主机, 则必须重新设置 MSTR 位, 否则将一直处于从机模式。

写冲突

SPI 在发送时为单缓冲, 在接收时为双缓冲。这样在前一次发送尚未完成之前, 不能将新的数据写

入移位寄存器。当发送过程中对数据寄存器 SPDAT 进行写操作时，WCOL 位将被置 1 以指示发生数据写冲突错误。在这种情况下，当前发送的数据继续发送，而新写入的数据将丢失。

当对主机或从机进行写冲突检测时，主机发生写冲突的情况是很罕见的，因为主机拥有数据传输的完全控制权。但从机有可能发生写冲突，因为当主机启动传输时，从机无法进行控制。

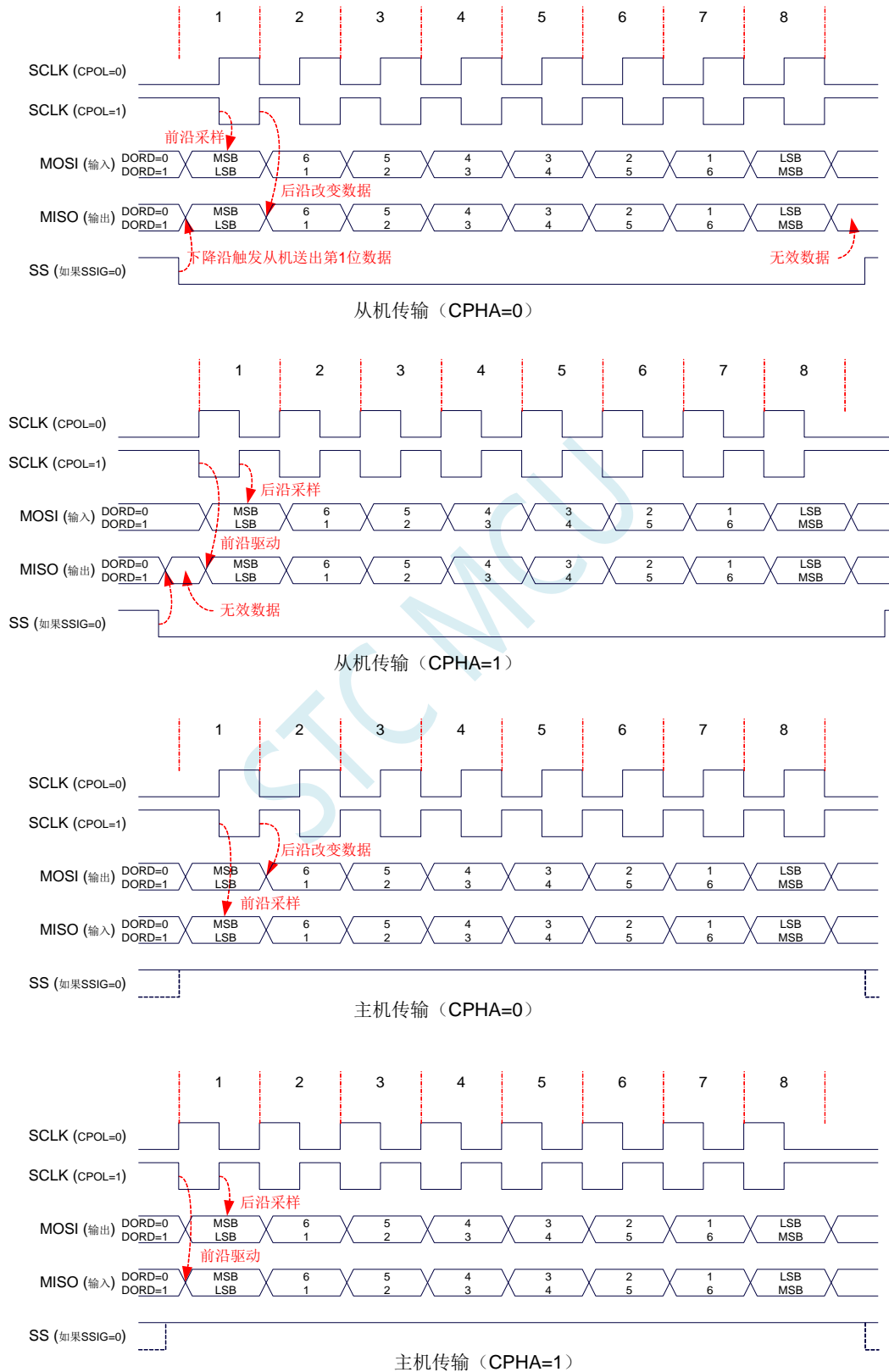
接收数据时，接收到的数据传送到一个并行读数据缓冲区，这样将释放移位寄存器以进行下一个数据的接收。但必须在下个字符完全移入之前从数据寄存器中读出接收到的数据，否则，前一个接收数据将丢失。

WCOL 可通过软件向其写入“1”清零。

STC MCU

16.5 数据模式

SPI 的时钟相位控制位 CPHA 可以让用户设定数据采样和改变时的时钟沿。时钟极性位 CPOL 可以让用户设定时钟极性。下面图例显示了不同时钟相位、极性设置下 SPI 通讯时序。



16.6 范例程序

16.6.1 SPI 单主单从系统主机程序（中断方式）

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"           //头文件见附录
#include "intrins.h"

#define   ESPI                0x02

sbit     SS                   =   P1^0;
sbit     LED                  =   P1^1;

bit      busy;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;           //清中断标志
    SS = 1;                  //拉高从机的SS 管脚
    busy = 0;
    LED = !LED;              //测试端口
}

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

    LED = 1;
    SS = 1;
    busy = 0;

    SPCTL = 0x50;            //使能 SPI 主机模式
    SPSTAT = 0xc0;          //清中断标志
    IE2 = ESPI;              //使能 SPI 中断
    EA = 1;

    while (1)
    {
        while (busy);
        busy = 1;
        SS = 0;              //拉低从机 SS 管脚
        SPDAT = 0x5a;        //发送测试数据
    }
}
```

}

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC)

;头文件见附录

ESPI EQU 02H

BUSY BIT 20H.0

SS BIT P1.0

LED BIT P1.1

ORG 0000H

LJMP MAIN

ORG 004BH

LJMP SPIISR

ORG 0100H

SPIISR:

MOV SPSTAT,#0C0H

;清中断标志

SETB SS

;拉高从机的 SS 管脚

CLR BUSY

CPL LED

RETI

MAIN:

MOV SP, #5FH

MOV P0M0, #00H

MOV P0M1, #00H

MOV P1M0, #00H

MOV P1M1, #00H

MOV P2M0, #00H

MOV P2M1, #00H

MOV P3M0, #00H

MOV P3M1, #00H

MOV P4M0, #00H

MOV P4M1, #00H

MOV P5M0, #00H

MOV P5M1, #00H

SETB LED

SETB SS

CLR BUSY

MOV SPCTL,#50H

;使能 SPI 主机模式

MOV SPSTAT,#0C0H

;清中断标志

MOV IE2,#ESPI

;使能 SPI 中断

SETB EA

LOOP:

JB BUSY,\$

SETB BUSY

CLR SS

;拉低从机 SS 管脚

MOV SPDAT,#5AH

;发送测试数据

JMP LOOP

END

16.6.2 SPI 单主单从系统从机程序（中断方式）

C 语言代码

//测试工作频率为11.0592MHz

#include "stc16.h" //头文件见附录
#include "intrins.h"

#define ESPI 0x02
sbit LED = P1^1;

```
void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0; //清中断标志
    SPDAT = SPDAT; //将接收到的数据回传给主机
    LED = !LED; //测试端口
}
```

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

    SPCTL = 0x40; //使能SPI 从机模式
    SPSTAT = 0xc0; //清中断标志
    IE2 = ESPI; //使能SPI 中断
    EA = 1;

    while (1);
}
```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC) ;头文件见附录

```
ESPI EQU 02H
LED BIT P1.1

ORG 0000H
LJMP MAIN
ORG 004BH
LJMP SPIISR
```

```

        ORG            0100H

SPIISR:
        MOV            SPSTAT,#0C0H        ;清中断标志
        MOV            SPDAT,SPDAT        ;将接收到的数据回传给主机
        CPL            LED
        RETI

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

        MOV            SPCTL,#40H        ;使能 SPI 从机模式
        MOV            SPSTAT,#0C0H        ;清中断标志
        MOV            IE2,#ESPI        ;使能 SPI 中断
        SETB            EA

        JMP            $

END

```

16.6.3 SPI 单主单从系统主机程序（查询方式）

C 语言代码

//测试工作频率为 11.0592MHz

```

#include "stc16.h"        //头文件见附录
#include "intrins.h"

```

```

#define    ESPI            0x02

```

```

sbit      SS              =    P1^0;
sbit      LED             =    P1^1;

```

```

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

```

```

P5M0 = 0x00;
P5M1 = 0x00;

LED = 1;
SS = 1;

SPCTL = 0x50;           //使能 SPI 主机模式
SPSTAT = 0xc0;          //清中断标志

while (1)
{
    SS = 0;              //拉低从机 SS 管脚
    SPDAT = 0x5a;        //发送测试数据
    while (!(SPSTAT & 0x80)); //查询完成标志
    SPSTAT = 0xc0;        //清中断标志
    SS = 1;              //拉高从机的 SS 管脚
    LED = !LED;          //测试端口
}
}

```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16F.INC)

;头文件见附录

ESPI EQU 02H

SS BIT P1.0

LED BIT P1.1

ORG 0000H

LJMP MAIN

ORG 0100H

MAIN:

MOV SP, #5FH

MOV P0M0, #00H

MOV P0M1, #00H

MOV P1M0, #00H

MOV P1M1, #00H

MOV P2M0, #00H

MOV P2M1, #00H

MOV P3M0, #00H

MOV P3M1, #00H

MOV P4M0, #00H

MOV P4M1, #00H

MOV P5M0, #00H

MOV P5M1, #00H

SETB LED

SETB SS

MOV SPCTL, #50H

;使能 SPI 主机模式

MOV SPSTAT, #0C0H

;清中断标志

LOOP:

CLR SS

;拉低从机 SS 管脚

MOV SPDAT, #5AH

;发送测试数据

```
MOV      A,SPSTAT      ;查询完成标志
JNB      ACC.7,$-2
MOV      SPSTAT,#0C0H  ;清中断标志
SETB     SS
CPL      LED
JMP      LOOP

END
```

16.6.4 SPI 单主单从系统从机程序（查询方式）

C 语言代码

//测试工作频率为 11.0592MHz

```
#include "stc16.h"      //头文件见附录
#include "intrins.h"

#define  ESPI            0x02
sbit    LED              =  P1^1;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;      //清中断标志
}

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

    SPCTL = 0x40;        //使能 SPI 从机模式
    SPSTAT = 0xc0;      //清中断标志

    while (1)
    {
        while (!(SPSTAT & 0x80));    //查询完成标志
        SPSTAT = 0xc0;              //清中断标志
        SPDAT = SPDAT;              //将接收到的数据回传给主机
        LED = !LED;                 //测试端口
    }
}
```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC) ;头文件见附录

ESPI EQU 02H
LED BIT P1.1

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 SPCTL, #40H ;使能 SPI 从机模式
MOV SPSTAT, #0C0H ;清中断标志

LOOP:

MOV A, SPSTAT ;查询完成标志
JNB ACC.7, \$-2
MOV SPSTAT, #0C0H ;清中断标志
MOV SPDAT, SPDAT ;将接收到的数据回传给主机
CPL LED
JMP LOOP

END

16.6.5 SPI 互为主从系统程序（中断方式）

C 语言代码

//测试工作频率为 11.0592MHz

#include "stc16.h" ;头文件见附录
#include "intrins.h"

#define ESPI 0x02

sbit SS = P1^0;
sbit LED = P1^1;
sbit KEY = P0^0;

void SPI_Isr() interrupt 9
{

SPSTAT = 0xc0; //清中断标志
if (SPCTL & 0x10)


```

{
    SS = 1;
    SPCTL = 0x40;
}
else
{
    SPDAT = SPDAT;
}
LED = !LED;
}

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

    LED = 1;
    KEY = 1;
    SS = 1;

    SPCTL = 0x40;
    SPSTAT = 0xc0;
    IE2 = ESPI;
    EA = 1;

    while (1)
    {
        if (!KEY)
        {
            SPCTL = 0x50;
            SS = 0;
            SPDAT = 0x5a;
            while (!KEY);
        }
    }
}

```

//主机模式
//拉高从机的SS 管脚
//重新设置为从机待机

//从机模式
//将接收到的数据回传给主机

//测试端口

//使能 SPI 从机模式进行待机
//清中断标志
//使能 SPI 中断

//等待按键触发

//使能 SPI 主机模式
//拉低从机 SS 管脚
//发送测试数据
//等待按键释放

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16F.INC)

;头文件见附录

ESPI EQU 02H

SS BIT P1.0

LED BIT P1.1

KEY BIT P0.0

```

        ORG      0000H
        LJMP     MAIN
        ORG      004BH
        LJMP     SPIISR

SPIISR:   ORG      0100H

        PUSH     ACC
        MOV      SPSTAT,#0C0H      ;清中断标志
        MOV      A,SPCTL
        JB       ACC.4,MASTER

SLAVE:    MOV      SPDAT,SPDAT      ;将接收到的数据回传给主机
        JMP      ISREXIT

MASTER:   SETB    SS                ;拉高从机的SS 管脚
        MOV      SPCTL,#40H        ;重新设置为从机待机

ISREXIT:  CPL      LED
        POP      ACC
        RETI

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

        SETB     SS
        SETB     LED
        SETB     KEY

        MOV      SPCTL,#40H        ;使能 SPI 从机模式进行待机
        MOV      SPSTAT,#0C0H      ;清中断标志
        MOV      IE2,#ESPI        ;使能 SPI 中断
        SETB     EA

LOOP:     JB      KEY,LOOP          ;等待按键触发
        MOV      SPCTL,#50H        ;使能 SPI 主机模式
        CLR      SS                ;拉低从机 SS 管脚
        MOV      SPDAT,#5AH        ;发送测试数据
        JNB      KEY,$             ;等待按键释放
        JMP      LOOP

        END

```

16.6.6 SPI 互为主从系统程序（查询方式）

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"           //头文件见附录
#include "intrins.h"
```

```
#define  ESPI                0x02
```

```
sbit    SS                  =  P1^0;
```

```
sbit    LED                 =  P1^1;
```

```
sbit    KEY                 =  P0^0;
```

```
void main()
```

```
{
```

```
    P0M0 = 0x00;
```

```
    P0M1 = 0x00;
```

```
    P1M0 = 0x00;
```

```
    P1M1 = 0x00;
```

```
    P2M0 = 0x00;
```

```
    P2M1 = 0x00;
```

```
    P3M0 = 0x00;
```

```
    P3M1 = 0x00;
```

```
    P4M0 = 0x00;
```

```
    P4M1 = 0x00;
```

```
    P5M0 = 0x00;
```

```
    P5M1 = 0x00;
```

```
    LED = 1;
```

```
    KEY = 1;
```

```
    SS = 1;
```

```
    SPCTL = 0x40;
```

```
    SPSTAT = 0xc0;
```

//使能 SPI 从机模式进行待机

//清中断标志

```
    while (1)
```

```
    {
```

```
        if (!KEY)
```

//等待按键触发

```
        {
```

```
            SPCTL = 0x50;
```

//使能 SPI 主机模式

```
            SS = 0;
```

//拉低从机 SS 管脚

```
            SPDAT = 0x5a;
```

//发送测试数据

```
            while (!KEY);
```

//等待按键释放

```
        }
```

```
        if (SPSTAT & 0x80)
```

```
        {
```

```
            SPSTAT = 0xc0;
```

//清中断标志

```
            if (SPCTL & 0x10)
```

```
            {
```

//主机模式

```
                SS = 1;
```

//拉高从机的 SS 管脚

```
                SPCTL = 0x40;
```

//重新设置为从机待机

```
            }
```

```
        } else
```

```
        {
```

//从机模式

```
            SPDAT = SPDAT;
```

//将接收到的数据回传给主机

```

    }
    LED = !LED;           //测试端口
  }
}
}

```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC)

;头文件见附录

ESPI EQU 02H

SS BIT P1.0

LED BIT P1.1

KEY BIT P0.0

ORG 0000H

LJMP MAIN

ORG 0100H

MAIN:

MOV SP, #5FH

MOV P0M0, #00H

MOV P0M1, #00H

MOV P1M0, #00H

MOV P1M1, #00H

MOV P2M0, #00H

MOV P2M1, #00H

MOV P3M0, #00H

MOV P3M1, #00H

MOV P4M0, #00H

MOV P4M1, #00H

MOV P5M0, #00H

MOV P5M1, #00H

SETB SS

SETB LED

SETB KEY

MOV SPCTL, #40H

;使能 SPI 从机模式进行待机

MOV SPSTAT, #0C0H

;清中断标志

LOOP:

JB KEY, SKIP

;等待按键触发

MOV SPCTL, #50H

;使能 SPI 主机模式

CLR SS

;拉低从机 SS 管脚

MOV SPDAT, #5AH

;发送测试数据

JNB KEY, \$

;等待按键释放

SKIP:

MOV A, SPSTAT

JNB ACC.7, LOOP

MOV SPSTAT, #0C0H

;清中断标志

MOV A, SPCTL

JB ACC.4, MASTER

SLAVE:

MOV SPDAT, SPDAT

;将接收到的数据回传给主机

CPL LED

```
MASTER:      JMP      LOOP
              SETB    SS          ;拉高从机的SS 管脚
              MOV     SPCTL,#40H  ;重新设置为从机待机
              CPL      LED
              JMP      LOOP

              END
```



17 I²C 总线

STC16F 系列的单片机内部集成了一个 I²C 串行总线控制器。I²C 是一种高速同步通讯总线，通讯使用 SCL（时钟线）和 SDA（数据线）两线进行同步通讯。对于 SCL 和 SDA 的端口分配，STC16F 系列的单片机提供了切换模式，可将 SCL 和 SDA 切换到不同的 I/O 口上，以方便用户将一组 I²C 总线当作多组进行分时复用。

与标准 I²C 协议相比较，忽略了如下两种机制：

- 发送起始信号（START）后不进行仲裁
- 时钟信号（SCL）停留在低电平时不进行超时检测

STC16F 系列的 I²C 总线提供了两种操作模式：主机模式（SCL 为输出口，发送同步时钟信号）和从机模式（SCL 为输入口，接收同步时钟信号）

17.1 I²C 功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW2	9BH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S

I2C_S[1:0]: I²C 功能脚选择位

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

17.2 I²C 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
I2CCFG	I ² C 配置寄存器	7EFE80H	ENI2C	MSSL	MSSPEED[5:0]						0000,0000
I2CMSCR	I ² C 主机控制寄存器	7EFE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I ² C 主机状态寄存器	7EFE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I ² C 从机控制寄存器	7EFE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I ² C 从机状态寄存器	7EFE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I ² C 从机地址寄存器	7EFE85H	SLADR[6:0]							MA	0000,0000
I2CTXD	I ² C 数据发送寄存器	7EFE86H									0000,0000
I2CRXD	I ² C 数据接收寄存器	7EFE87H									0000,0000
I2CMSAUX	I ² C 主机辅助控制寄存器	7EFE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0

17.3 I²C 主机模式

17.3.1 I2C 配置寄存器 (I2CCFG)，总线速度控制

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CCFG	7EFE80H	ENI2C	MSSL	MSSPEED[5:0]					

ENI2C: I²C 功能使能控制位

0: 禁止 I²C 功能

1: 允许 I²C 功能

MSSL: I²C 工作模式选择位

0: 从机模式

1: 主机模式

MSSPEED[5:0]: I²C 总线速度（等待时钟数）控制，**I2C 总线速度 = $F_{osc} / 2 / (MSSPEED * 2 + 4)$**

MSSPEED[5:0]	对应的时钟数
0	4
1	6
2	8
...	...
x	$2x+4$
...	...
62	128
63	130

只有当 I²C 模块工作在主机模式时，MSSPEED 参数设置的等待参数才有效。此等待参数主要用于主机模式的以下几个信号：

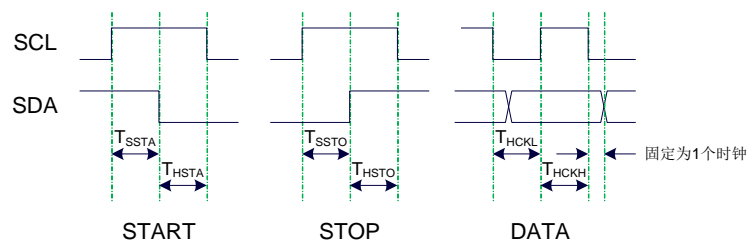
T_{SSTA}: 起始信号的建立时间（Setup Time of START）

T_{HSTA}: 起始信号的保持时间（Hold Time of START）

T_{SSTO}: 停止信号的建立时间（Setup Time of STOP）

T_{HSTO}: 停止信号的保持时间（Hold Time of STOP）

T_{HCKL}: 时钟信号的低电平保持时间（Hold Time of SCL Low）



例 1: 当 MSSPEED=10 时， $T_{SSTA}=T_{HSTA}=T_{SSTO}=T_{HSTO}=T_{HCKL}=24/F_{OSC}$

例 2: 当 24MHz 的工作频率下需要 400K 的 I2C 总线速度时，

$$MSSPEED = (24M / 400K / 2 - 4) / 2 = 13$$

17.3.2 I2C 主机控制寄存器 (I2CMSCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	7EFE81H	EMSI	-	-	-	-	MSCMD[2:0]		

EMSI: 主机模式中中断使能控制位

0: 关闭主机模式的中断

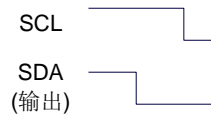
1: 允许主机模式的中断

MSCMD[3:0]: 主机命令

0000: 待机, 无动作。

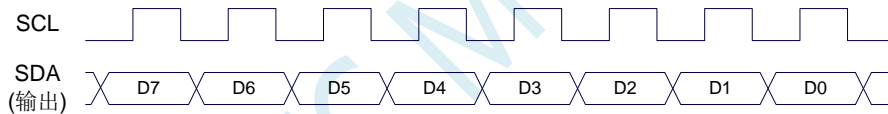
0001: 起始命令。

发送 START 信号。如果当前 I²C 控制器处于空闲状态, 即 MSBUSY (I2CMSST.7) 为 0 时, 写此命令会使控制器进入忙状态, 硬件自动将 MSBUSY 状态位置 1, 并开始发送 START 信号; 若当前 I²C 控制器处于忙状态, 写此命令可触发发送 START 信号。发送 START 信号的波形如下图所示:



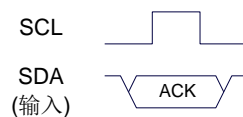
0010: 发送数据命令。

写此命令后, I²C 总线控制器会在 SCL 管脚上产生 8 个时钟, 并将 I2CTXD 寄存器里面数据按位送到 SDA 管脚上 (先发送高位数据)。发送数据的波形如下图所示:



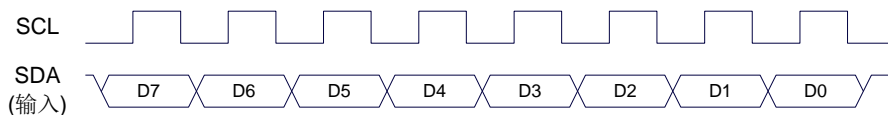
0011: 接收 ACK 命令。

写此命令后, I²C 总线控制器会在 SCL 管脚上产生 1 个时钟, 并将从 SDA 端口上读取的数据保存到 MSACKI (I2CMSST.1)。接收 ACK 的波形如下图所示:



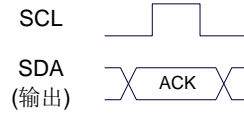
0100: 接收数据命令。

写此命令后, I²C 总线控制器会在 SCL 管脚上产生 8 个时钟, 并将从 SDA 端口上读取的数据依次左移到 I2CRXD 寄存器 (先接收高位数据)。接收数据的波形如下图所示:



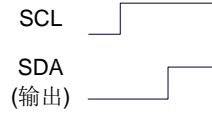
0101: 发送 ACK 命令。

写此命令后, I²C 总线控制器会在 SCL 管脚上产生 1 个时钟, 并将 MSACKO (I2CMSST.0) 中的数据发送到 SDA 端口。发送 ACK 的波形如下图所示:



0110: 停止命令。

发送 STOP 信号。写此命令后, I²C 总线控制器开始发送 STOP 信号。信号发送完成后, 硬件自动将 MSBUSY 状态位清零。STOP 信号的波形如下图所示:



0111: 保留。

1000: 保留。

1001: 起始命令+发送数据命令+接收 ACK 命令。

此命令为命令 0001、命令 0010、命令 0011 三个命令的组合, 下此命令后控制器会依次执行这三个命令。

1010: 发送数据命令+接收 ACK 命令。

此命令为命令 0010、命令 0011 两个命令的组合, 下此命令后控制器会依次执行这两个命令。

1011: 接收数据命令+发送 ACK(0)命令。

此命令为命令 0100、命令 0101 两个命令的组合, 下此命令后控制器会依次执行这两个命令。

注意: 此命令所返回的应答信号固定为 ACK (0), 不受 MSACKO 位的影响。

1100: 接收数据命令+发送 NAK(1)命令。

此命令为命令 0100、命令 0101 两个命令的组合, 下此命令后控制器会依次执行这两个命令。

注意: 此命令所返回的应答信号固定为 NAK (1), 不受 MSACKO 位的影响。

17.3.3 I²C 主机辅助控制寄存器 (I2CMSAUX)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSAUX	7EFE88H	-	-	-	-	-	-	-	WDTA

WDTA: 主机模式时 I²C 数据自动发送允许位

0: 禁止自动发送

1: 使能自动发送

若自动发送功能被使能, 当 MCU 执行完成对 I2CTXD 数据寄存器的写操作后, I²C 控制器会自动触发“1010”命令, 即自动发送数据并接收 ACK 信号。

17.3.4 I²C 主机状态寄存器 (I2CMSST)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSST	7EFE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO

MSBUSY: 主机模式时 I²C 控制器状态位 (只读位)

0: 控制器处于空闲状态

1: 控制器处于忙碌状态

当 I²C 控制器处于主机模式时, 在空闲状态下, 发送完成 START 信号后, 控制器便进入到忙碌状态, 忙碌状态会一直维持到成功发送完成 STOP 信号, 之后状态会再次恢复到空闲状态。

MSIF: 主机模式的中断请求位 (中断标志位)。当处于主机模式的 I²C 控制器执行完成寄存器 I2CMSCR

中 MSCMD 命令后产生中断信号，硬件自动将此位 1，向 CPU 发请求中断，响应中断后 MSIF 位必须用软件清零。

MSACKI: 主机模式时，发送“0011”命令到 I2CMSCR 的 MSCMD 位后所接收到的 ACK 数据。

MSACKO: 主机模式时，准备将要发送出去的 ACK 信号。当发送“0101”命令到 I2CMSCR 的 MSCMD 位后，控制器会自动读取此位的数据当作 ACK 发送到 SDA。

STC MCU

17.4 I²C 从机模式

17.4.1 I2C 从机控制寄存器 (I2CSLCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLCR	7EFE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST

ESTAI: 从机模式时接收到 START 信号中断允许位

0: 禁止从机模式时接收到 START 信号时发生中断

1: 使能从机模式时接收到 START 信号时发生中断

ERXI: 从机模式时接收到 1 字节数据后中断允许位

0: 禁止从机模式时接收到数据后发生中断

1: 使能从机模式时接收到 1 字节数据后发生中断

ETXI: 从机模式时发送完成 1 字节数据后中断允许位

0: 禁止从机模式时发送完成数据后发生中断

1: 使能从机模式时发送完成 1 字节数据后发生中断

ESTOI: 从机模式时接收到 STOP 信号中断允许位

0: 禁止从机模式时接收到 STOP 信号时发生中断

1: 使能从机模式时接收到 STOP 信号时发生中断

SLRST: 复位从机模式

17.4.2 I2C 从机状态寄存器 (I2CSLST)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLST	7EFE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	-	SLACKI	SLACKO

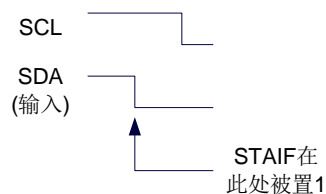
SLBUSY: 从机模式时 I²C 控制器状态位 (只读位)

0: 控制器处于空闲状态

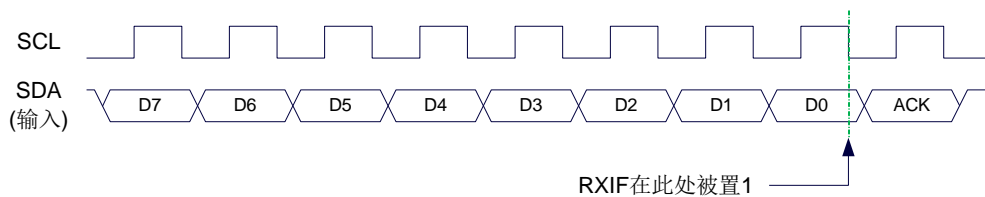
1: 控制器处于忙碌状态

当 I²C 控制器处于从机模式时, 在空闲状态下, 接收到主机发送 START 信号后, 控制器会继续检测之后的设备地址数据, 若设备地址与当前 I2CSLADR 寄存器中所设置的从机地址像匹配时, 控制器便进入到忙碌状态, 忙碌状态会一直维持到成功接收到主机发送 STOP 信号, 之后状态会再次恢复到空闲状态。

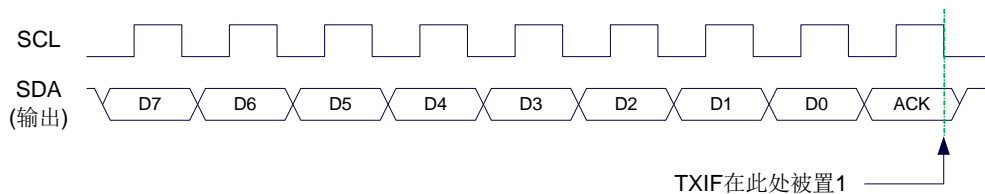
STAIF: 从机模式时接收到 START 信号后的中断请求位。从机模式的 I²C 控制器接收到 START 信号后, 硬件会自动将此位置 1, 并向 CPU 发请求中断, 响应中断后 STAIF 位必须用软件清零。STAIF 被置 1 的时间点如下图所示:



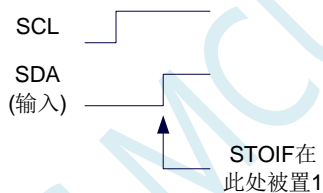
RXIF: 从机模式时接收到 1 字节的数据后的中断请求位。从机模式的 I²C 控制器接收到 1 字节的数据后, 在第 8 个时钟的下降沿时硬件会自动将此位置 1, 并向 CPU 发请求中断, 响应中断后 RXIF 位必须用软件清零。RXIF 被置 1 的时间点如下图所示:



TXIF: 从机模式时发送完成 1 字节的数据后的中断请求位。从机模式的 I²C 控制器发送完成 1 字节的数据并成功接收到 1 位 ACK 信号后，在第 9 个时钟的下降沿时硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 TXIF 位必须用软件清零。TXIF 被置 1 的时间点如下图所示：

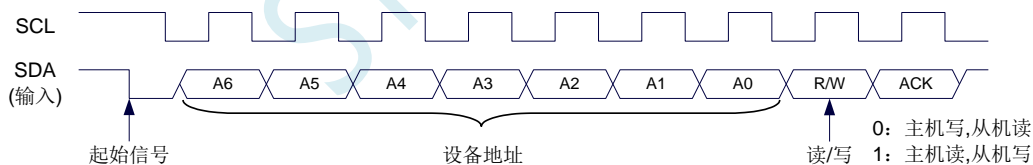


STOIF: 从机模式时接收到 STOP 信号后的中断请求位。从机模式的 I²C 控制器接收到 STOP 信号后，硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 STOIF 位必须用软件清零。STOIF 被置 1 的时间点如下图所示：



SLACKI: 从机模式时，接收到的 ACK 数据。

SLACKO: 从机模式时，准备将要发送出去的 ACK 信号。



17.4.3 I2C 从机地址寄存器 (I2CSLADR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLADR	7EFE85H	SLADR[6:0]							MA

SLADR[6:0]: 从机设备地址

当 I²C 控制器处于从机模式时，控制器在接收到 START 信号后，会继续检测接下来主机发送出的设备地址数据以及读/写信号。当主机发送出的设备地址与 SLADR[6:0]中所设置的从机设备地址相匹配时，控制器才会向 CPU 发出中断求，请求 CPU 处理 I²C 事件；否则若设备地址不匹配，I²C 控制器继续继续监控，等待下一个起始信号，对下一个设备地址继续匹配。

MA: 从机设备地址匹配控制

0: 设备地址必须与 SLADR[6:0]继续匹配

1: 忽略 SLADR 中的设置，匹配所有的设备地址

17.4.4 I2C 数据寄存器（I2CTXD, I2CRXD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CTXD	7EFE86H								
I2CRXD	7EFE87H								

I2CTXD 是 I²C 发送数据寄存器，存放将要发送的 I²C 数据

I2CRXD 是 I²C 接收数据寄存器，存放接收完成的 I²C 数据

STC MCU

17.5 范例程序

17.5.1 I²C 主机模式访问 AT24C256（中断方式）

C 语言代码

//测试工作频率为11.0592MHz

#include "stc16.h" //头文件见附录
#include "intrins.h"

sbit SDA = P1^4;
sbit SCL = P1^5;

bit busy;

void I2C_Isr() interrupt 24

```
{  
    char bak;  
    bak = P_SW2;  
    P_SW2 |= 0x80;  
    if (I2CMSST & 0x40)  
    {  
        I2CMSST &= ~0x40; //清中断标志  
        busy = 0;  
    }  
    P_SW2 = bak;  
}
```

void Start()

```
{  
    busy = 1;  
    I2CMSCR = 0x81; //发送 START 命令  
    while (busy);  
}
```

void SendData(char dat)

```
{  
    I2CTXD = dat; //写数据到数据缓冲区  
    busy = 1;  
    I2CMSCR = 0x82; //发送 SEND 命令  
    while (busy);  
}
```

void RecvACK()

```
{  
    busy = 1;  
    I2CMSCR = 0x83; //发送读 ACK 命令  
    while (busy);  
}
```

char RecvData()

```
{  
    busy = 1;  
    I2CMSCR = 0x84; //发送 RECV 命令  
    while (busy);  
    return I2CRXD;  
}
```

```
}

void SendACK()
{
    I2CMSST = 0x00;           //设置ACK 信号
    busy = 1;
    I2CMSCR = 0x85;           //发送ACK 命令
    while (busy);
}

void SendNAK()
{
    I2CMSST = 0x01;           //设置NAK 信号
    busy = 1;
    I2CMSCR = 0x85;           //发送ACK 命令
    while (busy);
}

void Stop()
{
    busy = 1;
    I2CMSCR = 0x86;           //发送STOP 命令
    while (busy);
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

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

    P_SW2 = 0x80;

    I2CCFG = 0xe0;           //使能 I2C 主机模式
    I2CMSST = 0x00;
    EA = 1;
}
```

```
Start(); //发送起始命令
SendData(0xa0); //发送设备地址+写命令
RecvACK();
SendData(0x00); //发送存储地址高字节
RecvACK();
SendData(0x00); //发送存储地址低字节
RecvACK();
SendData(0x12); //写测试数据1
RecvACK();
SendData(0x78); //写测试数据2
RecvACK();
Stop(); //发送停止命令

Delay(); //等待设备写数据

Start(); //发送起始命令
SendData(0xa0); //发送设备地址+写命令
RecvACK();
SendData(0x00); //发送存储地址高字节
RecvACK();
SendData(0x00); //发送存储地址低字节
RecvACK();
Start(); //发送起始命令
SendData(0xa1); //发送设备地址+读命令
RecvACK();
P0 = RecvData(); //读取数据1
SendACK();
P2 = RecvData(); //读取数据2
SendNAK();
Stop(); //发送停止命令

P_SW2 = 0x00;

while (1);
}
```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC)

;头文件见附录

SDA	BIT	P1.4
SCL	BIT	P1.5
BUSY	BIT	20H.0
	ORG	0000H
	LJMP	MAIN
	ORG	00C3H
	LJMP	I2CISR
	ORG	0100H
I2CISR:		
	PUSH	ACC
	PUSH	DPL
	PUSH	DPH
	MOV	WR6,#WORD0 I2CMSST


```

MOV      WR4,#WORD2 I2CMSST
MOV      R11,@DR4
ANL      A,#NOT 40H          ;清中断标志
MOV      @DR4,R11
CLR      BUSY                ;复位忙标志

POP      DPH
POP      DPL
POP      ACC
RETI

START:
SETB     BUSY
MOV      A,#10000001B        ;发送 START 命令
MOV      WR6,#WORD0 I2CMSCR
MOV      WR4,#WORD2 I2CMSCR
MOV      @DR4,R11
JMP      WAIT

SENDDATA:
MOV      WR6,#WORD0 I2CTXD    ;写数据到数据缓冲区
MOV      WR4,#WORD2 I2CTXD
MOV      @DR4,R11
SETB     BUSY
MOV      A,#10000010B        ;发送 SEND 命令
MOV      WR6,#WORD0 I2CMSCR
MOV      WR4,#WORD2 I2CMSCR
MOV      @DR4,R11
JMP      WAIT

RECVACK:
SETB     BUSY
MOV      A,#10000011B        ;发送读 ACK 命令
MOV      WR6,#WORD0 I2CMSCR
MOV      WR4,#WORD2 I2CMSCR
MOV      @DR4,R11
JMP      WAIT

RECVDATA:
SETB     BUSY
MOV      A,#10000100B        ;发送 RECV 命令
MOV      WR6,#WORD0 I2CMSCR
MOV      WR4,#WORD2 I2CMSCR
MOV      @DR4,R11
CALL     WAIT
MOV      WR6,#WORD0 I2CRXD
MOV      WR4,#WORD2 I2CRXD
MOV      R11,@DR4            ;从数据缓冲区读取数据
RET

SENDACK:
MOV      A,#00000000B        ;设置 ACK 信号
MOV      WR6,#WORD0 I2CMSST
MOV      WR4,#WORD2 I2CMSST
MOV      @DR4,R11
SETB     BUSY
MOV      A,#10000101B        ;发送 ACK 命令
MOV      WR6,#WORD0 I2CMSCR
MOV      WR4,#WORD2 I2CMSCR
MOV      @DR4,R11
JMP      WAIT

SENDNAK:
MOV      A,#00000001B        ;设置 NAK 信号

```

```

MOV        WR6,#WORD0 I2CMSST
MOV        WR4,#WORD2 I2CMSST
MOV        @DR4,R11
SETB       BUSY
MOV        A,#10000101B           ;发送ACK 命令
MOV        WR6,#WORD0 I2CMSCR
MOV        WR4,#WORD2 I2CMSCR
MOV        @DR4,R11
JMP        WAIT

STOP:
SETB       BUSY
MOV        A,#10000110B           ;发送STOP 命令
MOV        WR6,#WORD0 I2CMSCR
MOV        WR4,#WORD2 I2CMSCR
MOV        @DR4,R11
JMP        WAIT

WAIT:
JB         BUSY,$                 ;等待命令发送完成
RET

DELAY:
MOV        R0,#0
MOV        R1,#0

DELAY1:
NOP
NOP
NOP
NOP
DJNZ       R1,DELAY1
DJNZ       R0,DELAY1
RET

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

MOV        P_SW2,#80H

MOV        A,#11100000B           ;设置I2C 模块为主机模式
MOV        WR6,#WORD0 I2CCFG
MOV        WR4,#WORD2 I2CCFG
MOV        @DR4,R11
MOV        A,#00000000B
MOV        WR6,#WORD0 I2CMSST
MOV        WR4,#WORD2 I2CMSST
MOV        @DR4,R11
SETB       EA

```

```

CALL    START                ;发送起始命令
MOV     A,#0A0H
CALL    SENDDATA             ;发送设备地址+写命令
CALL    RECVACK
MOV     A,#000H               ;发送存储地址高字节
CALL    SENDDATA
CALL    RECVACK
MOV     A,#000H               ;发送存储地址低字节
CALL    SENDDATA
CALL    RECVACK
MOV     A,#12H                ;写测试数据1
CALL    SENDDATA
CALL    RECVACK
MOV     A,#78H                ;写测试数据2
CALL    SENDDATA
CALL    RECVACK
CALL    STOP                  ;发送停止命令

CALL    DELAY                 ;等待设备写数据

CALL    START                ;发送起始命令
MOV     A,#0A0H               ;发送设备地址+写命令
CALL    SENDDATA
CALL    RECVACK
MOV     A,#000H               ;发送存储地址高字节
CALL    SENDDATA
CALL    RECVACK
MOV     A,#000H               ;发送存储地址低字节
CALL    SENDDATA
CALL    RECVACK
CALL    START                ;发送起始命令
MOV     A,#0A1H               ;发送设备地址+读命令
CALL    SENDDATA
CALL    RECVACK
CALL    RECVDATA              ;读取数据1
MOV     P0,A
CALL    SENDACK
CALL    RECVDATA              ;读取数据2
MOV     P2,A
CALL    SENDNAK
CALL    STOP                  ;发送停止命令

JMP     $

END

```

17.5.2 I²C 主机模式访问 AT24C256（查询方式）

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"           //头文件见附录
#include "intrins.h"
```

```
sbit    SDA    =    P1^4;
sbit    SCL    =    P1^5;
```

```
void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;           //发送 START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;             //写数据到数据缓冲区
    I2CMSCR = 0x02;           //发送 SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;           //发送读 ACK 命令
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;           //发送 RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;           //设置 ACK 信号
    I2CMSCR = 0x05;           //发送 ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;           //设置 NAK 信号
    I2CMSCR = 0x05;           //发送 ACK 命令
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;           //发送 STOP 命令
    Wait();
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
```

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

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

    P_SW2 = 0x80;

    I2CCFG = 0xe0; //使能 I2C 主机模式
    I2CMSST = 0x00;

    Start(); //发送起始命令
    SendData(0xa0); //发送设备地址+写命令
    RecvACK();
    SendData(0x00); //发送存储地址高字节
    RecvACK();
    SendData(0x00); //发送存储地址低字节
    RecvACK();
    SendData(0x12); //写测试数据 1
    RecvACK();
    SendData(0x78); //写测试数据 2
    RecvACK();
    Stop(); //发送停止命令

    Delay(); //等待设备写数据

    Start(); //发送起始命令
    SendData(0xa0); //发送设备地址+写命令
    RecvACK();
    SendData(0x00); //发送存储地址高字节
    RecvACK();
    SendData(0x00); //发送存储地址低字节
    RecvACK();
    Start(); //发送起始命令
    SendData(0xa1); //发送设备地址+读命令
    RecvACK();
    P0 = RecvData(); //读取数据 1
    SendACK();
    P2 = RecvData(); //读取数据 2
    SendNAK();
    Stop(); //发送停止命令

    P_SW2 = 0x00;
```

```

    while (1);
}

```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC)

;头文件见附录

```

SDA      BIT      P1.4
SCL      BIT      P1.5

```

```

ORG      0000H
LJMP     MAIN

```

```

START:   ORG      0100H

```

```

MOV      A,#00000001B      ;发送START 命令
MOV      WR6,#WORD0 I2CMSCR
MOV      WR4,#WORD2 I2CMSCR
MOV      @DR4,R11
JMP      WAIT

```

SENDDATA:

```

MOV      WR6,#WORD0 I2CTXD      ;写数据到数据缓冲区
MOV      WR4,#WORD2 I2CTXD
MOV      @DR4,R11
MOV      A,#00000010B      ;发送SEND 命令
MOV      WR6,#WORD0 I2CMSCR
MOV      WR4,#WORD2 I2CMSCR
MOV      @DR4,R11
JMP      WAIT

```

RECVACK:

```

MOV      A,#00000011B      ;发送读ACK 命令
MOV      WR6,#WORD0 I2CMSCR
MOV      WR4,#WORD2 I2CMSCR
MOV      @DR4,R11
JMP      WAIT

```

RCVDATA:

```

MOV      A,#00000100B      ;发送RECV 命令
MOV      WR6,#WORD0 I2CMSCR
MOV      WR4,#WORD2 I2CMSCR
MOV      @DR4,R11
CALL     WAIT
MOV      WR6,#WORD0 I2CRXD
MOV      WR4,#WORD2 I2CRXD
MOV      R11,@DR4      ;从数据缓冲区读取数据
RET

```

SENDACK:

```

MOV      A,#00000000B      ;设置ACK 信号
MOV      WR6,#WORD0 I2CMSST
MOV      WR4,#WORD2 I2CMSST
MOV      @DR4,R11
MOV      A,#00000101B      ;发送ACK 命令
MOV      WR6,#WORD0 I2CMSCR
MOV      WR4,#WORD2 I2CMSCR
MOV      @DR4,R11
JMP      WAIT

```

SENDNAK:

```

MOV      A,#00000001B      ;设置 NAK 信号
MOV      WR6,#WORD0 I2CMSST
MOV      WR4,#WORD2 I2CMSST
MOV      @DR4,R11
MOV      A,#00000101B      ;发送 ACK 命令
MOV      WR6,#WORD0 I2CMSCR
MOV      WR4,#WORD2 I2CMSCR
MOV      @DR4,R11
JMP      WAIT

STOP:
MOV      A,#00000110B      ;发送 STOP 命令
MOV      WR6,#WORD0 I2CMSCR
MOV      WR4,#WORD2 I2CMSCR
MOV      @DR4,R11
JMP      WAIT

WAIT:
MOV      WR6,#WORD0 I2CMSST
MOV      WR4,#WORD2 I2CMSST
MOV      R11,@DR4
JNB      ACC.6,WAIT
ANL      A,#NOT 40H        ;清中断标志
MOV      @DR4,R11
RET

DELAY:
MOV      R0,#0
MOV      R1,#0

DELAY1:
NOP
NOP
NOP
NOP
DJNZ     R1,DELAY1
DJNZ     R0,DELAY1
RET

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

MOV      P_SW2,#80H

MOV      A,#11100000B      ;设置 I2C 模块为主机模式
MOV      WR6,#WORD0 I2CCFG
MOV      WR4,#WORD2 I2CCFG
MOV      @DR4,R11
MOV      A,#00000000B
MOV      WR6,#WORD0 I2CMSST

```

```

MOV      WR4,#WORD2 I2CMSST
MOV      @DR4,R11

CALL     START          ;发送起始命令
MOV      A,#0A0H
CALL     SENDDATA        ;发送设备地址+写命令
CALL     RECVACK
MOV      A,#000H          ;发送存储地址高字节
CALL     SENDDATA
CALL     RECVACK
MOV      A,#000H          ;发送存储地址低字节
CALL     SENDDATA
CALL     RECVACK
MOV      A,#12H          ;写测试数据1
CALL     SENDDATA
CALL     RECVACK
MOV      A,#78H          ;写测试数据2
CALL     SENDDATA
CALL     RECVACK
CALL     STOP            ;发送停止命令

CALL     DELAY           ;等待设备写数据

CALL     START          ;发送起始命令
MOV      A,#0A0H          ;发送设备地址+写命令
CALL     SENDDATA
CALL     RECVACK
MOV      A,#000H          ;发送存储地址高字节
CALL     SENDDATA
CALL     RECVACK
MOV      A,#000H          ;发送存储地址低字节
CALL     SENDDATA
CALL     RECVACK
CALL     START          ;发送起始命令
MOV      A,#0A1H          ;发送设备地址+读命令
CALL     SENDDATA
CALL     RECVACK
CALL     RECVDATA        ;读取数据1
MOV      P0,A
CALL     SENDACK
CALL     RECVDATA        ;读取数据2
MOV      P2,A
CALL     SENDNAK
CALL     STOP            ;发送停止命令

JMP      $

END

```

17.5.3 I²C 主机模式访问 PCF8563

C 语言代码

//测试工作频率为11.0592MHz

```

#include "stc16.h"          //头文件见附录
#include "intrins.h"

```



```
sbit    SDA        =    P1^4;
sbit    SCL        =    P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                //发送 START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                  //写数据到数据缓冲区
    I2CMSCR = 0x02;                //发送 SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                //发送读 ACK 命令
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                //发送 RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                //设置 ACK 信号
    I2CMSCR = 0x05;                //发送 ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                //设置 NAK 信号
    I2CMSCR = 0x05;                //发送 ACK 命令
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                //发送 STOP 命令
    Wait();
}

void Delay()
{
    int i;
```

```

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

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

    P_SW2 = 0x80;

    I2CCFG = 0xe0;           //使能 I2C 主机模式
    I2CMSST = 0x00;

    Start();                 //发送起始命令
    SendData(0xa2);          //发送设备地址+ 写命令
    RecvACK();
    SendData(0x02);          //发送存储地址
    RecvACK();
    SendData(0x00);          //设置秒值
    RecvACK();
    SendData(0x00);          //设置分钟值
    RecvACK();
    SendData(0x12);          //设置小时值
    RecvACK();
    Stop();                  //发送停止命令

    while (1)
    {
        Start();             //发送起始命令
        SendData(0xa2);       //发送设备地址+ 写命令
        RecvACK();
        SendData(0x02);       //发送存储地址
        RecvACK();
        Start();             //发送起始命令
        SendData(0xa3);       //发送设备地址+ 读命令
        RecvACK();
        P0 = RecvData();       //读取秒值
        SendACK();
        P2 = RecvData();       //读取分钟值
        SendACK();
        P3 = RecvData();       //读取小时值
        SendNAK();
    }
}

```

```

        Stop();                                     //发送停止命令

        Delay();
    }
}

```

汇编代码

;测试工作频率为11.0592MHz

\$include (STC16.INC)

;头文件见附录

```

SDA      BIT      P1.4
SCL      BIT      P1.5

```

```

        ORG      0000H
        LJMP     MAIN

```

```

START:   ORG      0100H

```

```

        MOV      A,#00000001B           ;发送START 命令
        MOV      WR6,#WORD0 I2CMSCR
        MOV      WR4,#WORD2 I2CMSCR
        MOV      @DR4,R11
        JMP      WAIT

```

SENDDATA:

```

        MOV      WR6,#WORD0 I2CTXD      ;写数据到数据缓冲区
        MOV      WR4,#WORD2 I2CTXD
        MOV      @DR4,R11
        MOV      A,#00000010B           ;发送SEND 命令
        MOV      WR6,#WORD0 I2CMSCR
        MOV      WR4,#WORD2 I2CMSCR
        MOV      @DR4,R11
        JMP      WAIT

```

RECVACK:

```

        MOV      A,#00000011B           ;发送读ACK 命令
        MOV      WR6,#WORD0 I2CMSCR
        MOV      WR4,#WORD2 I2CMSCR
        MOV      @DR4,R11
        JMP      WAIT

```

RECVDATA:

```

        MOV      A,#00000100B           ;发送RECV 命令
        MOV      WR6,#WORD0 I2CMSCR
        MOV      WR4,#WORD2 I2CMSCR
        MOV      @DR4,R11
        CALL     WAIT
        MOV      WR6,#WORD0 I2CRXD
        MOV      WR4,#WORD2 I2CRXD
        MOV      R11,@DR4               ;从数据缓冲区读取数据
        RET

```

SENDACK:

```

        MOV      A,#00000000B           ;设置ACK 信号
        MOV      WR6,#WORD0 I2CMSST
        MOV      WR4,#WORD2 I2CMSST
        MOV      @DR4,R11
        MOV      A,#00000101B           ;发送ACK 命令
        MOV      WR6,#WORD0 I2CMSCR
        MOV      WR4,#WORD2 I2CMSCR
        MOV      @DR4,R11

```

```

JMP      WAIT

SENDNAK:
MOV      A,#00000001B          ;设置 NAK 信号
MOV      WR6,#WORD0 I2CMSST
MOV      WR4,#WORD2 I2CMSST
MOV      @DR4,R11
MOV      A,#00000101B          ;发送 ACK 命令
MOV      WR6,#WORD0 I2CMSCR
MOV      WR4,#WORD2 I2CMSCR
MOV      @DR4,R11
JMP      WAIT

STOP:
MOV      A,#00000110B          ;发送 STOP 命令
MOV      WR6,#WORD0 I2CMSCR
MOV      WR4,#WORD2 I2CMSCR
MOV      @DR4,R11
JMP      WAIT

WAIT:
MOV      WR6,#WORD0 I2CMSST
MOV      WR4,#WORD2 I2CMSST
MOV      R11,@DR4
JNB      ACC.6,WAIT
ANL      A,#NOT 40H          ;清中断标志
MOV      @DR4,R11
RET

DELAY:
MOV      R0,#0
MOV      R1,#0

DELAY1:
NOP
NOP
NOP
NOP
DJNZ     R1,DELAY1
DJNZ     R0,DELAY1
RET

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

MOV      P_SW2,#80H

MOV      A,#11100000B          ;设置 I2C 模块为主机模式
MOV      WR6,#WORD0 I2CCFG
MOV      WR4,#WORD2 I2CCFG
MOV      @DR4,R11

```

```

MOV      A,#00000000B
MOV      WR6,#WORD0 I2CMSST
MOV      WR4,#WORD2 I2CMSST
MOV      @DR4,R11

CALL     START           ;发送起始命令
MOV      A,#0A2H
CALL     SENDDATA        ;发送设备地址+写命令
CALL     RECVACK
MOV      A,#002H         ;发送存储地址
CALL     SENDDATA
CALL     RECVACK
MOV      A,#00H          ;设置秒值
CALL     SENDDATA
CALL     RECVACK
MOV      A,#00H          ;设置分钟值
CALL     SENDDATA
CALL     RECVACK
MOV      A,#12H          ;设置小时值
CALL     SENDDATA
CALL     RECVACK
CALL     STOP            ;发送停止命令

LOOP:    CALL     START           ;发送起始命令
MOV      A,#0A2H         ;发送设备地址+写命令
CALL     SENDDATA
CALL     RECVACK
MOV      A,#002H         ;发送存储地址
CALL     SENDDATA
CALL     RECVACK
CALL     START           ;发送起始命令
MOV      A,#0A3H         ;发送设备地址+读命令
CALL     SENDDATA
CALL     RECVACK
CALL     RECVDATA        ;读取秒值
MOV      P0,A
CALL     SENDACK
CALL     RECVDATA        ;读取分钟值
MOV      P2,A
CALL     SENDACK
CALL     RECVDATA        ;读取小时值
MOV      P3,A
CALL     SENDNAK
CALL     STOP            ;发送停止命令

CALL     DELAY

JMP      LOOP

END

```

17.5.4 I²C 从机模式（中断方式）

C 语言代码

//测试工作频率为11.0592MHz

```

#include "stc16.h"           //头文件见附录
#include "intrins.h"

sbit    SDA      =  P1^4;
sbit    SCL      =  P1^5;

bit      isda;              //设备地址标志
bit      isma;              //存储地址标志
unsigned char    addr;
unsigned char edata    buffer[32];

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;

    if (I2CSLST & 0x40)
    {
        I2CSLST &= ~0x40;                //处理 START 事件
    }
    else if (I2CSLST & 0x20)
    {
        I2CSLST &= ~0x20;                //处理 RECV 事件
        if (isda)
        {
            isda = 0;                    //处理 RECV 事件 (RECV DEVICE ADDR)
        }
        else if (isma)
        {
            isma = 0;                    //处理 RECV 事件 (RECV MEMORY ADDR)
            addr = I2CRXD;
            I2CTXD = buffer[addr];
        }
        else
        {
            buffer[addr++] = I2CRXD;      //处理 RECV 事件 (RECV DATA)
        }
    }
    else if (I2CSLST & 0x10)
    {
        I2CSLST &= ~0x10;                //处理 SEND 事件
        if (I2CSLST & 0x02)
        {
            I2CTXD = 0xff;                //接收到 NAK 则停止读取数据
        }
        else
        {
            I2CTXD = buffer[++addr];      //接收到 ACK 则继续读取数据
        }
    }
    else if (I2CSLST & 0x08)
    {
        I2CSLST &= ~0x08;                //处理 STOP 事件
        isda = 1;
        isma = 1;
    }

    _pop_(P_SW2);
}

```

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

    P_SW2 = 0x80;

    I2CCFG = 0x81;
    I2CSLADR = 0x5a;

    I2CSLST = 0x00;
    I2CSLCR = 0x78;
    EA = 1;

    isda = 1;
    isma = 1;
    addr = 0;
    I2CTXD = buffer[addr];

    while (1);
}
```

//使能 I2C 从机模式

//设置从机设备地址寄存器 I2CSLADR=0101_1010B

//即 I2CSLADR[7:1]=010_1101B,MA=0B。

//由于 MA 为 0,主机发送的设备地址必须与

//I2CSLADR[7:1]相同才能访问此 I2C 从机设备。

//主机若需要写数据则要发送 5AH(0101_1010B)

//主机若需要读数据则要发送 5BH(0101_1011B)

//使能从机模式中断

//用户变量初始化

汇编代码

```
;测试工作频率为 11.0592MHz

#include (STC16.INC)           ;头文件见附录

SDA      BIT      P1.4
SCL      BIT      P1.5
ISDA     BIT      20H.0      ;设备地址标志
ISMA     BIT      20H.1      ;存储地址标志

ADDR     DATA     21H

                ORG      0000H
                LJMP     MAIN
                ORG      00C3H
                LJMP     I2CISR

                ORG      0100H
I2CISR:
                PUSH     ACC
                PUSH     PSW
```

```

        PUSH        DPL
        PUSH        DPH
        MOV         WR6,#WORD0 I2CSLST
        MOV         WR4,#WORD2 I2CSLST
        MOV         R11,@DR4                ;检测从机状态
        JB          ACC.6,STARTIF
        JB          ACC.5,RXIF
        JB          ACC.4,TXIF
        JB          ACC.3,STOPIF
ISREXIT:
        POP         DPH
        POP         DPL
        POP         PSW
        POP         ACC
        RETI
STARTIF:
        ANL         A,#NOT 40H                ;处理 START 事件
        MOV         @DR4,R11
        JMP         ISREXIT
RXIF:
        ANL         A,#NOT 20H                ;处理 RECV 事件
        MOV         @DR4,R11
        MOV         WR6,#WORD0 I2CRXD
        MOV         WR4,#WORD2 I2CRXD
        MOV         R11,@DR4
        JBC         ISDA,RXDA
        JBC         ISMA,RXMA
        MOV         R0,ADDR                ;处理 RECV 事件 (RECV DATA)
        MOVX        @R0,A
        INC         ADDR
        JMP         ISREXIT
RXDA:
        JMP         ISREXIT                ;处理 RECV 事件 (RECV DEVICE ADDR)
RXMA:
        MOV         ADDR,A                ;处理 RECV 事件 (RECV MEMORY ADDR)
        MOV         R0,A
        MOVX        A,@R0
        MOV         WR6,#WORD0 I2CTXD
        MOV         WR4,#WORD2 I2CTXD
        MOV         @DR4,R11
        JMP         ISREXIT
TXIF:
        ANL         A,#NOT 10H                ;处理 SEND 事件
        MOV         @DR4,R11
        JB          ACC.1,RXNAK
        INC         ADDR
        MOV         R0,ADDR
        MOVX        A,@R0
        MOV         WR6,#WORD0 I2CTXD
        MOV         WR4,#WORD2 I2CTXD
        MOV         @DR4,R11
        JMP         ISREXIT
RXNAK:
        MOV         A,#0FFH
        MOV         WR6,#WORD0 I2CTXD
        MOV         WR4,#WORD2 I2CTXD
        MOV         @DR4,R11
        JMP         ISREXIT
STOPIF:

```



```

ANL      A,#NOT 08H      ;处理 STOP 事件
MOV      @DR4,R11
SETB     ISDA
SETB     ISMA
JMP      ISREXIT

```

MAIN:

```

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

MOV      P_SW2,#80H

MOV      A,#10000001B      ;使能 I2C 从机模式
MOV      WR6,#WORD0 I2CCFG
MOV      WR4,#WORD2 I2CCFG
MOV      @DR4,R11
MOV      A,#01011010B      ;设置从机设备地址寄存器 I2CSLADR=0101_1010B
                                ;即 I2CSLADR[7:1]=010_1101B,MA=0B。
                                ;由于 MA 为 0,主机发送的设备地址必须与
                                ;I2CSLADR[7:1]相同才能访问此 I2C 从机设备。
                                ;主机若需要写数据则要发送 5AH(0101_1010B)
                                ;主机若需要读数据则要发送 5BH(0101_1011B)

MOV      WR6,#WORD0 I2CSLADR
MOV      WR4,#WORD2 I2CSLADR
MOV      @DR4,R11
MOV      A,#00000000B
MOV      WR6,#WORD0 I2CSLST
MOV      WR4,#WORD2 I2CSLST
MOV      @DR4,R11
MOV      A,#01111000B      ;使能从机模式中断
MOV      WR6,#WORD0 I2CSLCR
MOV      WR4,#WORD2 I2CSLCR
MOV      @DR4,R11

SETB     ISDA      ;用户变量初始化
SETB     ISMA
CLR      A
MOV      ADDR,A
MOV      R0,A
MOVX     A,@R0
MOV      WR6,#WORD0 I2CTXD
MOV      WR4,#WORD2 I2CTXD
MOV      @DR4,R11

SETB     EA

SJMP     $

```

END

17.5.5 I²C 从机模式（查询方式）

C 语言代码

//测试工作频率为11.0592MHz

#include "stc16.h" //头文件见附录
#include "intrins.h"

sbit SDA = P1^4;
sbit SCL = P1^5;

bit isda; //设备地址标志
bit isma; //存储地址标志
unsigned char addr;
unsigned char edata buffer[32];

void main()

{

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

P_SW2 = 0x80;

I2CCFG = 0x81;
I2CSLADR = 0x5a;

I2CSLST = 0x00;
I2CSLCR = 0x00;

isda = 1;
isma = 1;
addr = 0;
I2CTXD = buffer[addr];

while (1)

{

if (I2CSLST & 0x40)

{

I2CSLST &= ~0x40;

}

//使能I2C 从机模式

//设置从机设备地址寄存器 I2CSLADR=0101_1010B

//即 I2CSLADR[7:1]=010_1101B,MA=0B。

//由于MA 为0,主机发送的设备地址必须与

//I2CSLADR[7:1]相同才能访问此 I2C 从机设备。

//主机若需要写数据则要发送 5AH(0101_1010B)

//主机若需要读数据则要发送 5BH(0101_1011B)

//禁止从机模式中断

//用户变量初始化

//处理 START 事件

```
else if (I2CSLST & 0x20)
{
    I2CSLST &= ~0x20;                //处理 RECV 事件
    if (isda)
    {
        isda = 0;                    //处理 RECV 事件 (RECV DEVICE ADDR)
    }
    else if (isma)
    {
        isma = 0;                    //处理 RECV 事件 (RECV MEMORY ADDR)
        addr = I2CRXD;
        I2CTXD = buffer[addr];
    }
    else
    {
        buffer[addr++] = I2CRXD;     //处理 RECV 事件 (RECV DATA)
    }
}
else if (I2CSLST & 0x10)
{
    I2CSLST &= ~0x10;                //处理 SEND 事件
    if (I2CSLST & 0x02)
    {
        I2CTXD = 0xff;               //接收到 NAK 则停止读取数据
    }
    else
    {
        I2CTXD = buffer[++addr];     //接收到 ACK 则继续读取数据
    }
}
else if (I2CSLST & 0x08)
{
    I2CSLST &= ~0x08;                //处理 STOP 事件
    isda = 1;
    isma = 1;
}
}
```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC) ;头文件见附录

SDA	BIT	P1.4	
SCL	BIT	P1.5	
ISDA	BIT	20H.0	;设备地址标志
ISMA	BIT	20H.1	;存储地址标志
ADDR	DATA	21H	
	ORG	0000H	
	LJMP	MAIN	
	ORG	0100H	
MAIN:	MOV	SP, #5FH	
	MOV	P0M0, #00H	

```

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

MOV      P_SW2, #80H

MOV      A, #10000001B          ; 使能 I2C 从机模式
MOV      WR6, #WORD0 I2CCFG
MOV      WR4, #WORD2 I2CCFG
MOV      @DR4, R11
MOV      A, #01011010B          ; 设置从机设备地址寄存器 I2CSLADR=0101_1010B
                                   ; 即 I2CSLADR[7:1]=010_1101B, MA=0B。
                                   ; 由于 MA 为 0, 主机发送的设备地址必须与
                                   ; I2CSLADR[7:1] 相同才能访问此 I2C 从机设备。
                                   ; 主机若需要写数据则要发送 5AH(0101_1010B)
                                   ; 主机若需要读数据则要发送 5BH(0101_1011B)

MOV      WR6, #WORD0 I2CSLADR
MOV      WR4, #WORD2 I2CSLADR
MOV      @DR4, R11
MOV      A, #00000000B
MOV      WR6, #WORD0 I2CSLST
MOV      WR4, #WORD2 I2CSLST
MOV      @DR4, R11
MOV      A, #00000000B          ; 禁止从机模式中断
MOV      WR6, #WORD0 I2CSLCR
MOV      WR4, #WORD2 I2CSLCR
MOV      @DR4, R11

SETB     ISDA                    ; 用户变量初始化
SETB     ISMA
CLR      A
MOV      ADDR, A
MOV      R0, A
MOVX     A, @R0
MOV      WR6, #WORD0 I2CTXD
MOV      WR4, #WORD2 I2CTXD
MOV      @DR4, R11

LOOP:
MOV      WR6, #WORD0 I2CSLST
MOV      WR4, #WORD2 I2CSLST
MOV      R11, @DR4              ; 检测从机状态
JB       ACC.6, STARTIF
JB       ACC.5, RXIF
JB       ACC.4, TXIF
JB       ACC.3, STOPIF
JMP      LOOP

STARTIF:
ANL      A, #NOT 40H            ; 处理 START 事件
MOV      @DR4, R11
JMP      LOOP

```

RXIF:

```

ANL      A,#NOT 20H           ;处理 RECV 事件
MOV      @DR4,R11
MOV      WR6,#WORD0 I2CRXD
MOV      WR4,#WORD2 I2CRXD
MOV      R11,@DR4
JBC      ISDA,RXDA
JBC      ISMA,RXMA
MOV      R0,ADDR             ;处理 RECV 事件 (RECV DATA)
MOVB     @R0,A
INC      ADDR
JMP      LOOP

```

RXDA:

```

JMP      LOOP                ;处理 RECV 事件 (RECV DEVICE ADDR)

```

RXMA:

```

MOV      ADDR,A              ;处理 RECV 事件 (RECV MEMORY ADDR)
MOV      R0,A
MOVB     A,@R0
MOV      WR6,#WORD0 I2CTXD
MOV      WR4,#WORD2 I2CTXD
MOV      @DR4,R11
JMP      LOOP

```

TXIF:

```

ANL      A,#NOT 10H          ;处理 SEND 事件
MOV      @DR4,R11
JB       ACC.1,RXNAK
INC      ADDR
MOV      R0,ADDR
MOVB     A,@R0
MOV      WR6,#WORD0 I2CTXD
MOV      WR4,#WORD2 I2CTXD
MOV      @DR4,R11
JMP      LOOP

```

RXNAK:

```

MOV      A,#0FFH
MOV      WR6,#WORD0 I2CTXD
MOV      WR4,#WORD2 I2CTXD
MOV      @DR4,R11
JMP      LOOP

```

STOPIF:

```

ANL      A,#NOT 08H          ;处理 STOP 事件
MOV      @DR4,R11
SETB     ISDA
SETB     ISMA
JMP      LOOP

```

```

END

```

17.5.6 测试 I²C 从机模式代码的主机代码

C 语言代码

```

//测试工作频率为11.0592MHz

```

```

#include "stc16.h"           //头文件见附录
#include "intrins.h"

```

```
sbit      SDA      =  P1^4;
sbit      SCL      =  P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;           //发送 START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;             //写数据到数据缓冲区
    I2CMSCR = 0x02;           //发送 SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;           //发送读 ACK 命令
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;           //发送 RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;           //设置 ACK 信号
    I2CMSCR = 0x05;           //发送 ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;           //设置 NAK 信号
    I2CMSCR = 0x05;           //发送 ACK 命令
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;           //发送 STOP 命令
    Wait();
}

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

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

P_SW2 = 0x80;

I2CCFG = 0xe0; //使能 I2C 主机模式
I2CMSST = 0x00;

Start(); //发送起始命令
SendData(0x5a); //发送设备地址(010_1101B)+写命令(0B)
RecvACK();
SendData(0x00); //发送存储地址
RecvACK();
SendData(0x12); //写测试数据 1
RecvACK();
SendData(0x78); //写测试数据 2
RecvACK();
Stop(); //发送停止命令

Start(); //发送起始命令
SendData(0x5a); //发送设备地址(010_1101B)+写命令(0B)
RecvACK();
SendData(0x00); //发送存储地址高字节
RecvACK();
Start(); //发送起始命令
SendData(0x5b); //发送设备地址(010_1101B)+读命令(1B)
RecvACK();
P0 = RecvData(); //读取数据 1
SendACK();
P2 = RecvData(); //读取数据 2
SendNAK();
Stop(); //发送停止命令

P_SW2 = 0x00;

while (1);
}
```

汇编代码

;测试工作频率为 11.0592MHz

\$include (STC16.INC) ;头文件见附录

SDA	BIT	P1.4
SCL	BIT	P1.5
	ORG	0000H
	LJMP	MAIN
	ORG	0100H

START:

```

MOV    A,#00000001B           ;发送 START 命令
MOV    WR6,#WORD0 I2CMSCR
MOV    WR4,#WORD2 I2CMSCR
MOV    @DR4,R11
JMP    WAIT

```

SENDDATA:

```

MOV    WR6,#WORD0 I2CTXD       ;写数据到数据缓冲区
MOV    WR4,#WORD2 I2CTXD
MOV    @DR4,R11
MOV    A,#00000010B           ;发送 SEND 命令
MOV    WR6,#WORD0 I2CMSCR
MOV    WR4,#WORD2 I2CMSCR
MOV    @DR4,R11
JMP    WAIT

```

RECVACK:

```

MOV    A,#00000011B           ;发送读 ACK 命令
MOV    WR6,#WORD0 I2CMSCR
MOV    WR4,#WORD2 I2CMSCR
MOV    @DR4,R11
JMP    WAIT

```

RECVDATA:

```

MOV    A,#00000100B           ;发送 RECV 命令
MOV    WR6,#WORD0 I2CMSCR
MOV    WR4,#WORD2 I2CMSCR
MOV    @DR4,R11
CALL   WAIT
MOV    WR6,#WORD0 I2CRXD
MOV    WR4,#WORD2 I2CRXD
MOV    R11,@DR4               ;从数据缓冲区读取数据
RET

```

SENDACK:

```

MOV    A,#00000000B           ;设置 ACK 信号
MOV    WR6,#WORD0 I2CMSST
MOV    WR4,#WORD2 I2CMSST
MOV    @DR4,R11
MOV    A,#00000101B           ;发送 ACK 命令
MOV    WR6,#WORD0 I2CMSCR
MOV    WR4,#WORD2 I2CMSCR
MOV    @DR4,R11
JMP    WAIT

```

SENDNAK:

```

MOV    A,#00000001B           ;设置 NAK 信号
MOV    WR6,#WORD0 I2CMSST
MOV    WR4,#WORD2 I2CMSST
MOV    @DR4,R11
MOV    A,#00000101B           ;发送 ACK 命令
MOV    WR6,#WORD0 I2CMSCR
MOV    WR4,#WORD2 I2CMSCR
MOV    @DR4,R11
JMP    WAIT

```

STOP:

```

MOV    A,#00000110B           ;发送 STOP 命令
MOV    WR6,#WORD0 I2CMSCR
MOV    WR4,#WORD2 I2CMSCR
MOV    @DR4,R11
JMP    WAIT

```

WAIT:

```

MOV    WR6,#WORD0 I2CMSST

```



```

MOV      WR4,#WORD2 I2CMSST
MOV      R11,@DR4
JNB      ACC.6,WAIT
ANL      A,#NOT 40H           ;清中断标志
MOV      @DR4,R11
RET

DELAY:
MOV      R0,#0
MOV      R1,#0

DELAY1:
NOP
NOP
NOP
NOP
DJNZ     R1,DELAY1
DJNZ     R0,DELAY1
RET

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

MOV      P_SW2,#80H

MOV      A,#11100000B           ;设置 I2C 模块为主机模式
MOV      WR6,#WORD0 I2CCFG
MOV      WR4,#WORD2 I2CCFG
MOV      @DR4,R11
MOV      A,#00000000B
MOV      WR6,#WORD0 I2CMSST
MOV      WR4,#WORD2 I2CMSST
MOV      @DR4,R11

CALL     START                   ;发送起始命令
MOV      A,#5AH
;
CALL     SENDDATA                ;发送设备地址(010_1101B)+写命令(0B)
CALL     RECVACK
MOV      A,#000H                 ;发送存储地址
CALL     SENDDATA
CALL     RECVACK
MOV      A,#12H                  ;写测试数据 1
CALL     SENDDATA
CALL     RECVACK
MOV      A,#78H                  ;写测试数据 2
CALL     SENDDATA
CALL     RECVACK
CALL     STOP                    ;发送停止命令

```

<i>CALL</i>	<i>DELAY</i>	;等待设备写数据
<i>CALL</i>	<i>START</i>	;发送起始命令
<i>MOV</i>	<i>A,#5AH</i>	;发送设备地址(010_1101B)+写命令(0B)
<i>CALL</i>	<i>SENDDATA</i>	
<i>CALL</i>	<i>RECVACK</i>	
<i>MOV</i>	<i>A,#000H</i>	;发送存储地址
<i>CALL</i>	<i>SENDDATA</i>	
<i>CALL</i>	<i>RECVACK</i>	
<i>CALL</i>	<i>START</i>	;发送起始命令
<i>MOV</i>	<i>A,#5BH</i>	;发送设备地址(010_1101B)+读命令(1B)
<i>CALL</i>	<i>SENDDATA</i>	
<i>CALL</i>	<i>RECVACK</i>	
<i>CALL</i>	<i>RECVDATA</i>	;读取数据 1
<i>MOV</i>	<i>P0,A</i>	
<i>CALL</i>	<i>SENDACK</i>	
<i>CALL</i>	<i>RECVDATA</i>	;读取数据 2
<i>MOV</i>	<i>P2,A</i>	
<i>CALL</i>	<i>SENDNAK</i>	
<i>CALL</i>	<i>STOP</i>	;发送停止命令
<i>JMP</i>	<i>\$</i>	
<i>END</i>		

18 高级 PWM

STC16 系列的单片机内部集成了 8 通道 16 位高级 PWM 定时器，分成两组周期可不同的 PWM，分别命名为 PWMA 和 PWMB，可分别单独设置。第一组 PWM/PWMA 可配置成 4 组互补/对称/死区控制的 PWM 或捕捉外部信号，第二组 PWM/PWMB 可配置成 4 路 PWM 输出或捕捉外部信号。

第一组 PWM/PWMA 的时钟频率可以是系统时钟经过寄存器 [PWMA_PSCRH](#) 和 [PWMA_PSCRL](#) 进行分频后的时钟，分频值可以是 1~65535 之间的任意值。第二组 PWM/PWMB 的时钟频率可以是系统时钟经过寄存器 [PWMB_PSCRH](#) 和 [PWMB_PSCRL](#) 进行分频后的时钟，分频值可以是 1~65535 之间的任意值。两组 PWM 的时钟频率可分别独立设置。

第一组 PWM 定时器/PWMA 有 4 个通道（PWM1P/PWM1N、PWM2P/PWM2N、PWM3P/PWM3N、PWM4P/PWM4N），每个通道都可独立实现 PWM 输出（可设置带死区的互补对称 PWM 输出）、捕获和比较功能；第二组 PWM 定时器/PWMB 有 4 个通道（PWM5、PWM6、PWM7、PWM8），每个通道也可独立实现 PWM 输出、捕获和比较功能。两组 PWM 定时器唯一的区别是第一组可输出带死区的互补对称 PWM，而第二组只能输出单端的 PWM，其他功能完全相同。下面关于高级 PWM 定时器的介绍只以第一组为例进行说明。

当使用第一组 PWM 定时器输出 PWM 波形时，可单独使能 PWM1P/PWM2P/PWM3P/PWM4P 输出，也可单独使能 PWM1N/PWM2N/PWM3N/PWM4N 输出。例如：若单独使能了 PWM1P 输出，则 PWM1N 就不能再独立输出，除非 PWM1P 和 PWM1N 组成一组互补对称输出。PWMA 的 4 路输出是可分别独立设置的，例如：可单独使能 PWM1P 和 PWM2N 输出，也可单独使能 PWM2N 和 PWM3N 输出。若需要使用第一组 PWM 定时器进行捕获功能或者测量脉宽时，输入信号只能从每路的正端输入，即只有 PWM1P/PWM2P/PWM3P/PWM4P 才有捕获功能和测量脉宽功能。

两组高级 PWM 定时器对外部信号进行捕获时，可选择上升沿捕获或者下降沿捕获。如果需要同时捕获上升沿和下降沿，则可将输入信号同时接入到两路 PWM，使能其中一路捕获上升沿，另外一路捕获下降沿即可。**更强悍的是，将外部输入信号同时接入到两路 PWM 时，可同时捕获信号的周期值和占空比值。**

STC 三种硬件 PWM 比较：

兼容传统 8051 的 PCA/CCP/PWM：可输出 PWM 波形、捕获外部输入信号以及输出高速脉冲。可对外输出 6 位/7 位/8 位/10 位的 PWM 波形，6 位 PWM 波形的频率为 PCA 模块时钟源频率/64；7 位 PWM 波形的频率为 PCA 模块时钟源频率/128；8 位 PWM 波形的频率为 PCA 模块时钟源频率/256；10 位 PWM 波形的频率为 PCA 模块时钟源频率/1024。捕获外部输入信号，可捕获上升沿、下降沿或者同时捕获上升沿和下降沿。

STC8G 系列的 15 位增强型 PWM：只能对外输出 PWM 波形，无输入捕获功能。对外输出 PWM 的频率以及占空比均可任意设置。通过软件干预，可实现多路互补/对称/带死区的 PWM 波形。有外部异常检测功能以及实时触发 ADC 转换功能。

STC16/STC8H 系列的 16 位高级 PWM 定时器：是目前 STC 功能最强的 PWM，可对外输出任意频率以及任意占空比的 PWM 波形。无需软件干预即可输出互补/对称/带死区的 PWM 波形。能捕获外部输入信号，可捕获上升沿、下降沿或者同时捕获上升沿和下降沿，测量外部波形时，可同时测量波形的周期值和占空比值。有正交编码功能、外部异常检测功能以及实时触发 ADC 转换功能。

下面的说明中，PWMA 代表第一组 PWM 定时器，PWMB 代表第二组 PWM 定时器

第 1 组高级 PWM 定时器/PWMA 内部信号说明

TI1：外部时钟输入信号 1（PWM1P 管脚信号或者 PWM1P/PWM2P/PWM3P 相异或后的信号）

TI1F: 经过 IC1F 数字滤波后的 TI1 信号

TI1FP: 经过 CC1P/CC2P 边沿检测器后的 TI1F 信号

TI1F_ED: TI1F 的边沿信号

TI1FP1: 经过 CC1P 边沿检测器后的 TI1F 信号

TI1FP2: 经过 CC2P 边沿检测器后的 TI1F 信号

IC1: 通过 CC1S 选择的通道 1 的捕获输入信号

OC1REF: 输出通道 1 输出的参考波形 (中间波形)

OC1: 通道 1 的主输出信号 (经过 CC1P 极性处理后的 OC1REF 信号)

OC1N: 通道 1 的互补输出信号 (经过 CC1NP 极性处理后的 OC1REF 信号)

TI2: 外部时钟输入信号 2 (PWM2P 管脚信号)

TI2F: 经过 IC2F 数字滤波后的 TI2 信号

TI2F_ED: TI2F 的边沿信号

TI2FP: 经过 CC1P/CC2P 边沿检测器后的 TI2F 信号

TI2FP1: 经过 CC1P 边沿检测器后的 TI2F 信号

TI2FP2: 经过 CC2P 边沿检测器后的 TI2F 信号

IC2: 通过 CC2S 选择的通道 2 的捕获输入信号

OC2REF: 输出通道 2 输出的参考波形 (中间波形)

OC2: 通道 2 的主输出信号 (经过 CC2P 极性处理后的 OC2REF 信号)

OC2N: 通道 2 的互补输出信号 (经过 CC2NP 极性处理后的 OC2REF 信号)

TI3: 外部时钟输入信号 3 (PWM3P 管脚信号)

TI3F: 经过 IC3F 数字滤波后的 TI3 信号

TI3F_ED: TI3F 的边沿信号

TI3FP: 经过 CC3P/CC4P 边沿检测器后的 TI3F 信号

TI3FP3: 经过 CC3P 边沿检测器后的 TI3F 信号

TI3FP4: 经过 CC4P 边沿检测器后的 TI3F 信号

IC3: 通过 CC3S 选择的通道 3 的捕获输入信号

OC3REF: 输出通道 3 输出的参考波形 (中间波形)

OC3: 通道 3 的主输出信号 (经过 CC3P 极性处理后的 OC3REF 信号)

OC3N: 通道 3 的互补输出信号 (经过 CC3NP 极性处理后的 OC3REF 信号)

TI4: 外部时钟输入信号 4 (PWM4P 管脚信号)

TI4F: 经过 IC4F 数字滤波后的 TI4 信号

TI4F_ED: TI4F 的边沿信号

TI4FP: 经过 CC3P/CC4P 边沿检测器后的 TI4F 信号

TI4FP3: 经过 CC3P 边沿检测器后的 TI4F 信号

TI4FP4: 经过 CC4P 边沿检测器后的 TI4F 信号

IC4: 通过 CC4S 选择的通道 4 的捕获输入信号

OC4REF: 输出通道 4 输出的参考波形 (中间波形)

OC4: 通道 4 的主输出信号 (经过 CC4P 极性处理后的 OC4REF 信号)

OC4N: 通道 4 的互补输出信号 (经过 CC4NP 极性处理后的 OC4REF 信号)

ITR1: 内部触发输入信号 1

ITR2: 内部触发输入信号 2

TRC: 固定为 TI1_ED

TRGI: 经过 TS 多路选择器后的触发输入信号

TRGO: 经过 MMS 多路选择器后的触发输出信号

ETR: 外部触发输入信号 (PWMETI1 管脚信号)

ETRP: 经过 ETP 边沿检测器以及 ETPS 分频器后的 ETR 信号

ETRF: 经过 ETF 数字滤波后的 ETRP 信号

BRK: 刹车输入信号 (PWMFLT)

CK_PSC: 预分频时钟, PWMA_PSCR 预分频器的输入时钟

CK_CNT: PWMA_PSCR 预分频器的输出时钟, PWM 定时器的时钟

第 2 组高级 PWM 定时器/PWMB 内部信号说明

TI5: 外部时钟输入信号 5 (PWM5 管脚信号或者 PWM5/PWM6/PWM7 相异或后的信号)

TI5F: 经过 IC5F 数字滤波后的 TI5 信号

TI5FP: 经过 CC5P/CC6P 边沿检测器后的 TI5F 信号

TI5F_ED: TI5F 的边沿信号

TI5FP5: 经过 CC5P 边沿检测器后的 TI5F 信号

TI5FP6: 经过 CC6P 边沿检测器后的 TI5F 信号

IC5: 通过 CC5S 选择的通道 5 的捕获输入信号

OC5REF: 输出通道 5 输出的参考波形 (中间波形)

OC5: 通道 5 的主输出信号 (经过 CC5P 极性处理后的 OC5REF 信号)

TI6: 外部时钟输入信号 6 (PWM6 管脚信号)

TI6F: 经过 IC6F 数字滤波后的 TI6 信号

TI6F_ED: TI6F 的边沿信号

TI6FP: 经过 CC5P/CC6P 边沿检测器后的 TI6F 信号

TI6FP5: 经过 CC5P 边沿检测器后的 TI6F 信号

TI6FP6: 经过 CC6P 边沿检测器后的 TI6F 信号

IC6: 通过 CC6S 选择的通道 6 的捕获输入信号

OC6REF: 输出通道 6 输出的参考波形 (中间波形)

OC6: 通道 6 的主输出信号 (经过 CC6P 极性处理后的 OC6REF 信号)

TI7: 外部时钟输入信号 7 (PWM7 管脚信号)

TI7F: 经过 IC7F 数字滤波后的 TI7 信号

TI7F_ED: TI7F 的边沿信号

TI7FP: 经过 CC7P/CC8P 边沿检测器后的 TI7F 信号

TI7FP7: 经过 CC7P 边沿检测器后的 TI7F 信号

TI7FP8: 经过 CC8P 边沿检测器后的 TI7F 信号

IC7: 通过 CC7S 选择的通道 7 的捕获输入信号

OC7REF: 输出通道 7 输出的参考波形 (中间波形)

OC7: 通道 7 的主输出信号 (经过 CC7P 极性处理后的 OC7REF 信号)

TI8: 外部时钟输入信号 8 (PWM8 管脚信号)

TI8F: 经过 IC8F 数字滤波后的 TI8 信号

TI8F_ED: TI8F 的边沿信号

TI8FP: 经过 CC7P/CC8P 边沿检测器后的 TI8F 信号

TI8FP7: 经过 CC7P 边沿检测器后的 TI8F 信号

TI8FP8: 经过 CC8P 边沿检测器后的 TI8F 信号

IC8: 通过 CC8S 选择的通道 8 的捕获输入信号

OC8REF: 输出通道 8 输出的参考波形（中间波形）

OC8: 通道 8 的主输出信号（经过 CC8P 极性处理后的 OC8REF 信号）

18.1 简介

PWMB 与 PWMA 唯一的区别是 PWMA 可输出带死区的互补对称 PWM，而 PWMB 则只能输出单端的 PWM，其他功能完全相同。下面关于高级 PWM 的介绍只以 PWMA 为例进行说明。

PWMA 由一个 16 位的自动装载计数器组成，它由一个可编程的预分频器驱动。

PWMA 适用于许多不同的用途：

- 基本的定时
- 测量输入信号的脉冲宽度（输入捕获）
- 产生输出波形（输出比较，PWM 和单脉冲模式）
- 对应与不同事件（捕获，比较，溢出，刹车，触发）的中断
- 与 PWMB 或者外部信号（外部时钟，复位信号，触发和使能信号）同步

PWMA 广泛的适用于各种控制应用中，包括那些需要中间对齐模式 PWM 的应用，该模式支持互补输出和死区时间控制。

PWMA 的时钟源可以是内部时钟，也可以是外部的信号，可以通过配置寄存器来进行选择。

18.2 主要特性

PWMA 的特性包括：

- 16 位向上、向下、向上/下自动装载计数器
- 允许在指定数目的计数器周期之后更新定时器寄存器的重复计数器
- 16 位可编程（可以实时修改）预分频器，计数器时钟频率的分频系数为 1~65535 之间的任意数值
- 同步电路，用于使用外部信号控制定时器以及定时器互联
- 多达 4 个独立通道可以配置成：
 - 输入捕获
 - 输出比较
 - PWM 输出（边缘或中间对齐模式）
 - 六步 PWM 输出
 - 单脉冲模式输出
 - 支持 4 个死区时间可编程的通道上互补输出
- 刹车输入信号（PWMFLT）可以将定时器输出信号置于复位状态或者一个确定状态
- 外部触发输入引脚（PWMETI）
- 产生中断的事件包括：

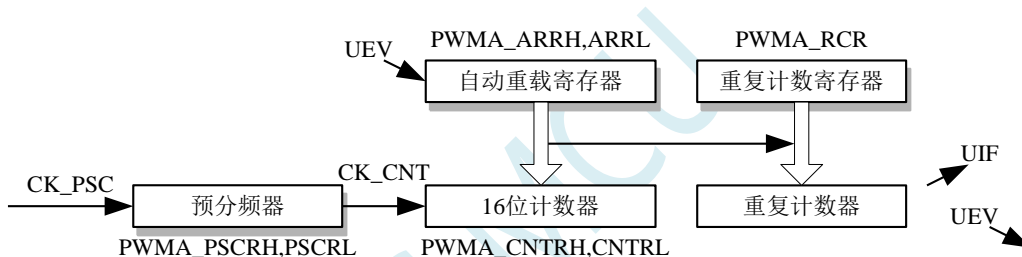
- 更新: 计数器向上溢出/向下溢出, 计数器初始化 (通过软件或者内部/外部触发)
- 触发事件 (计数器启动、停止、初始化或者由内部/外部触发计数)
- 输入捕获
- 输出比较
- 刹车信号输入

18.3 时基单元

PWMA 的时基单元包含:

- 16 位向上/向下计数器
- 16 位自动重载寄存器
- 重复计数器
- 预分频器

PWMA 时基单元



16 位计数器、预分频器、自动重载寄存器和重复计数器寄存器都可以通过软件进行读写操作。

自动重载寄存器由预装载寄存器和影子寄存器组成。

可在在两种模式下写自动重载寄存器:

- 自动预装载已使能 (PWMA_CR1 寄存器的 ARPE 位为 1)。
在此模式下, 写入自动重载寄存器的数据将被保存在预装载寄存器中, 并在下一个更新事件 (UEV) 时传送到影子寄存器。
 - 自动预装载已禁止 (PWMA_CR1 寄存器的 ARPE 位为 0)。
在此模式下, 写入自动重载寄存器的数据将立即写入影子寄存器。
- 更新事件的产生条件:
- 计数器向上或向下溢出。
 - 软件置位了 PWMA_EGR 寄存器的 UG 位。
 - 时钟/触发控制器产生了触发事件。

在预装载使能时 (ARPE=1), 如果发生了更新事件, 预装载寄存器中的数值 (PWMA_ARR) 将写入影子寄存器中, 并且 PWMA_PSCR 寄存器中的值将写入预分频器中。

置位 PWMA_CR1 寄存器的 UDIS 位将禁止更新事件 (UEV)。

预分频器的输出 CK_CNT 驱动计数器, 而 CK_CNT 仅在 IM1_CR1 寄存器的计数器使能位 (CEN)

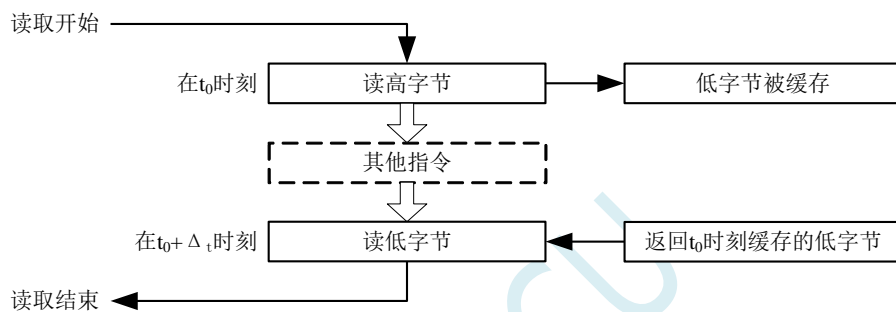
被置位时才有效。

注意：实际的计数器在 CEN 位使能的一个时钟周期后才开始计数。

18.3.1 读写 16 位计数器

写计数器的操作没有缓存，在任何时候都可以写 PWMA_CNTRH 和 PWMA_CNTRL 寄存器，因此为避免写入了错误的数值，一般建议不要在计数器运行时写入新的数值。

读计数器的操作带有 8 位的缓存。用户必须先读定时器的高字节，在用户读了高字节后，低字节将被自动缓存，缓存的数据将会一直保持直到 16 位数据的读操作完成。



18.3.2 16 位 PWMA_ARR 寄存器的写操作

预装载寄存器中的值将写入 16 位的 PWMA_ARR 寄存器中，此操作由两条指令完成，每条指令写入 1 个字节。必须先写高字节，后写低字节。

影子寄存器在写入高字节时被锁定，并保持到低字节写完。

18.3.3 预分频器

预分频器的实现：

PWMA 的预分频器基于一个由 16 位寄存器（PWMA_PSCR）控制的 16 位计数器。由于这个控制寄存器带有缓冲器，因此它能够在运行时被改变。预分频器可以将计数器的时钟频率按 1 到 65536 之间的任意值分频。预分频器的值由预装载寄存器写入，保存了当前使用值的影子寄存器在低字节写入时被载入。由于需两次单独的写操作来写 16 位寄存器，因此必须保证高字节先写入。新的预分频器的值在下次更新事件到来时被采用。对 PWMA_PSCR 寄存器的读操作通过预装载寄存器完成。

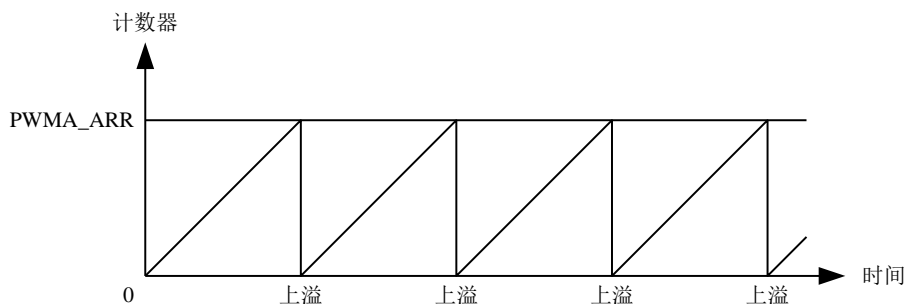
计数器的频率计算公式： $f_{CK_CNT} = f_{CK_PSC} / (PSCR[15:0] + 1)$

18.3.4 向上计数模式

在向上计数模式中，计数器从 0 计数到用户定义的比较值（PWMA_ARR 寄存器的值），然后重新

从 0 开始计数并产生一个计数器溢出事件, 此时如果 PWMA_CR1 寄存器的 UDIS 位是 0, 将会产生一个更新事件 (UEV)。

向上计数模式的计数器



通过软件方式或者通过使用触发控制器置位 PWMA_EGR 寄存器的 UG 位同样也可以产生一个更新事件。

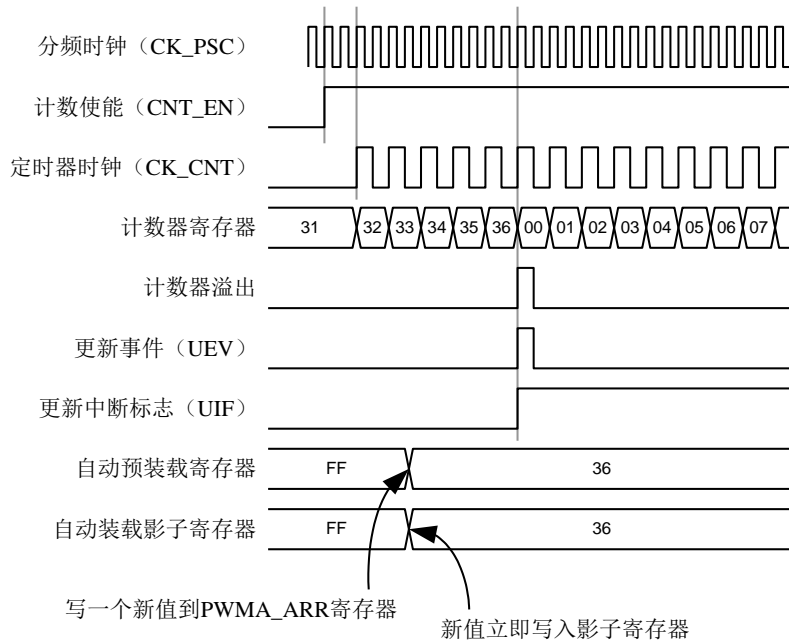
使用软件置位 PWMA_CR1 寄存器的 UDIS 位, 可以禁止更新事件, 这样可以避免在更新预装载寄存器时更新影子寄存器。在 UDIS 位被清除之前, 将不产生更新事件。但是在应该产生更新事件时, 计数器仍会被清 0, 同时预分频器的计数也被清 0 (但预分频器的数值不变)。此外, 如果设置了 PWMA_CR1 寄存器中的 URS 位 (选择更新请求), 设置 UG 位将产生一个更新事件 UEV, 但硬件不设置 UIF 标志 (即不产生中断请求)。这是为了避免在捕获模式下清除计数器时, 同时产生更新和捕获中断。

当发生一个更新事件时, 所有的寄存器都被更新, 硬件依据 URS 位同时设置更新标志位 (PWMA_SR 寄存器的 UIF 位) :

- 自动装载影子寄存器被重新置入预装载寄存器的值 (PWMA_ARR)。
- 预分频器的缓存器被置入预装载寄存器的值 (PWMA_PSC 寄存器的内容)。

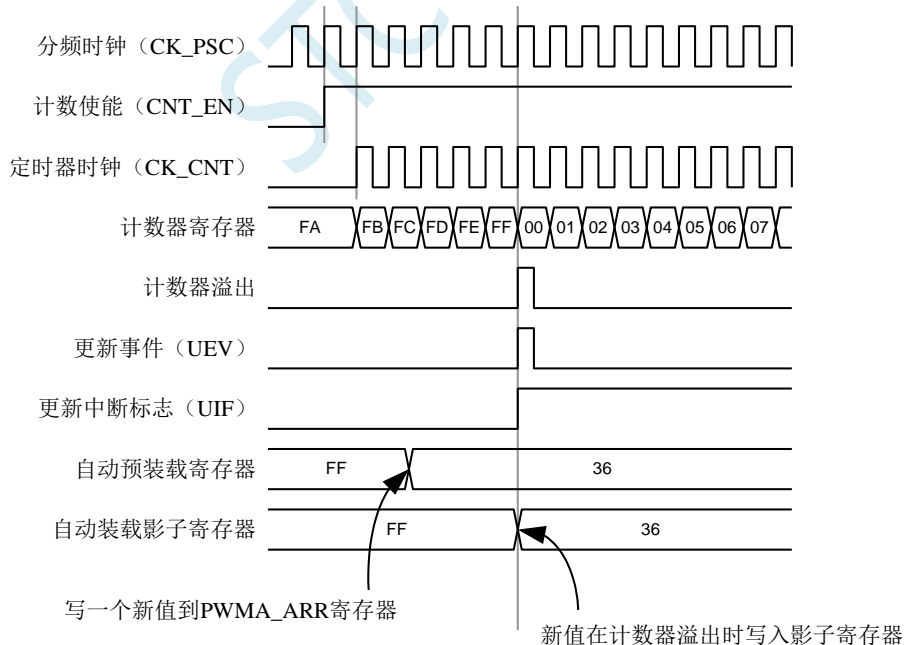
下图给出一些例子, 说明当 PWMA_ARR=0x36 时, 计数器在不同时钟频率下的动作。图中预分频为 2, 因此计数器的时钟 (CK_CNT) 频率是预分频时钟 (CK_PSC) 频率的一半。图中禁止了自动装载功能 (ARPE=0), 所以在计数器达到 0x36 时, 计数器溢出, 影子寄存器立刻被更新, 同时产生一个更新事件。

当 ARPE=0 (ARR 不预装载), 预分频为 2 时的计数器更新:



下图的预分频为 1，因此 CK_CNT 的频率与 CK_PSC 一致。图中使能了自动重载 (ARPE=1)，所以在计数器达到 0xFF 产生溢出。0x36 将在溢出时被写入，同时产生一个更新事件。

ARPE=1(PWMA_ARR 预装载) 预分频为 1 时的计数器更新：

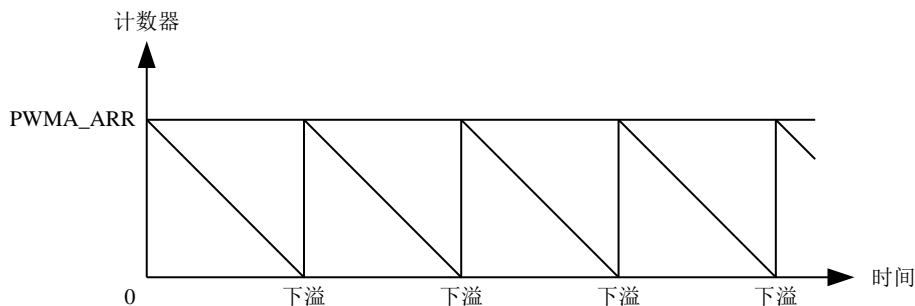


18.3.5 向下计数模式

在向下模式中，计数器从自动装载的值 (PWMA_ARR 寄存器的值) 开始向下计数到 0，然后再从

自动装载的值重新开始计数，并产生一个计数器向下溢出事件。如果 PWMA_CR1 寄存器的 UDIS 位被清除，还会产生一个更新事件（UEV）。

向下计数模式的计数器



通过软件方式或者通过使用触发控制器置位 PWMA_EGR 寄存器的 UG 位同样也可以产生一个更新事件。

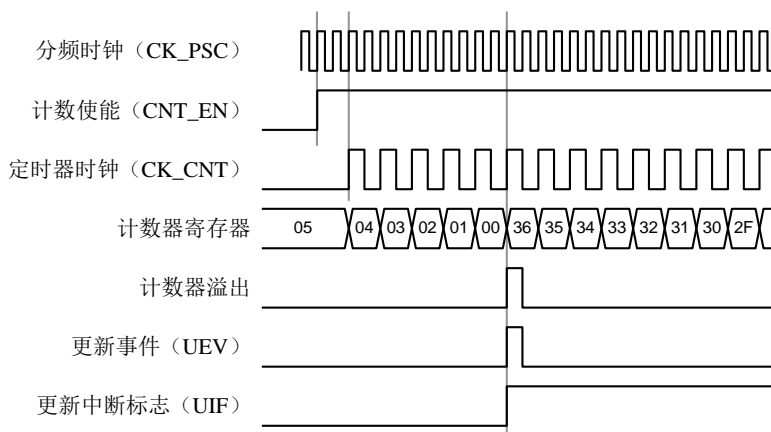
置位 PWMA_CR1 寄存器的 UDIS 位可以禁止 UEV 事件。这样可以避免在更新预装载寄存器时更新影子寄存器。因此 UDIS 位清除之前不会产生更新事件。然而，计数器仍会从当前自动加载值重新开始计数，并且预分频器的计数器重新从 0 开始（但预分频器不能被修改）。此外，如果设置了 PWMA_CR1 寄存器中的 URS 位（选择更新请求），设置 UG 位将产生一个更新事件 UEV 但不设置 UIF 标志（因此不产生中断），这是为了避免在发生捕获事件并清除计数器时，同时产生更新和捕获中断。

当发生更新事件时，所有的寄存器都被更新，硬件依据 URS 位同时设置更新标志位（PWMA_SR 寄存器的 UIF 位）：

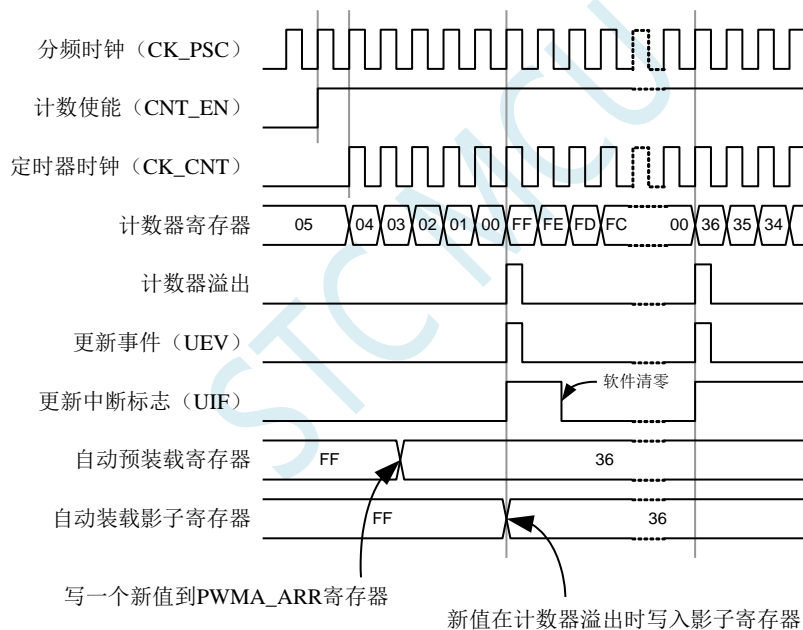
- 自动装载影子寄存器被重新置入预装载寄存器的值（PWMA_ARR）。
- 预分频器的缓存器被置入预装载寄存器的值（PWMA_PSC 寄存器的内容）。

以下是一些当 PWMA_ARR=0x36 时，计数器在不同时钟频率下的图表。下图描述了在向下计数模式下，预装载不使能时新的数值在下个周期时被写入。

ARPE=0（ARR 不预装载），预分频为 2 时的计数器更新：



ARPE=1 (ARR 预装载)，预分频为 1 时的计数器更新

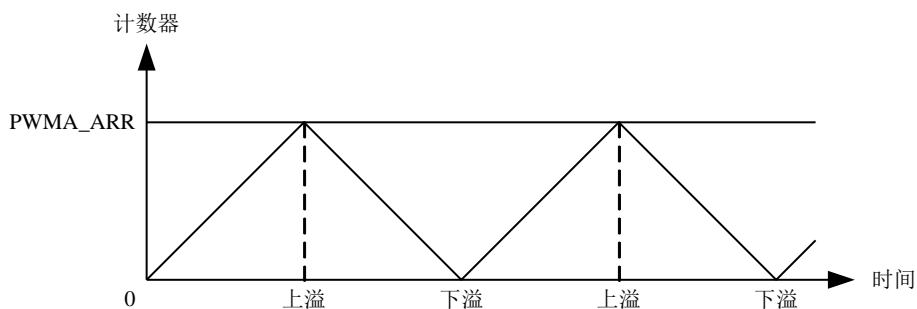


18.3.6 中间对齐模式（向上/向下计数）

在中央对齐模式，计数器从 0 开始计数到 PWMA_ARR 寄存器的值，产生一个计数器上溢事件，然后从 PWMA_ARR 寄存器的值向下计数到 0 并且产生一个计数器下溢事件；然后再从 0 开始重新计数。

在此模式下，不能写入 PWMA_CR1 中的 DIR 方向位。它由硬件更新并指示当前的计数方向。

中央对齐模式的计数器



如果定时器带有重复计数器, 在重复了指定次数 (PWMA_RCR 的值) 的向上和向下溢出之后会产生更新事件 (UEV)。否则每一次的向上向下溢出都会产生更新事件。

通过软件方式或者通过使用触发控制器置位 PWMA_EGR 寄存器的 UG 位同样也可以产生一个更新事件。此时, 计数器重新从 0 开始计数, 预分频器也重新从 0 开始计数。

设置 PWMA_CR1 寄存器中的 UDIS 位可以禁止 UEV 事件。这样可以避免在更新预装载寄存器时更新影子寄存器。因此 UDIS 位被清为 0 之前不会产生更新事件。然而, 计数器仍会根据当前自动重加载的值, 继续向上或向下计数。如果定时器带有重复计数器, 由于重复寄存器没有双重的缓冲, 新的重复数值将立刻生效, 因此在修改时需要小心。此外, 如果设置了 PWMA_CR1 寄存器中的 URS 位 (选择更新请求), 设置 UG 位将产生一个更新事件 UEV 但不设置 UIF 标志 (因此不产生中断), 这是为了避免在发生捕获事件并清除计数器时, 同时产生更新和捕获中断。

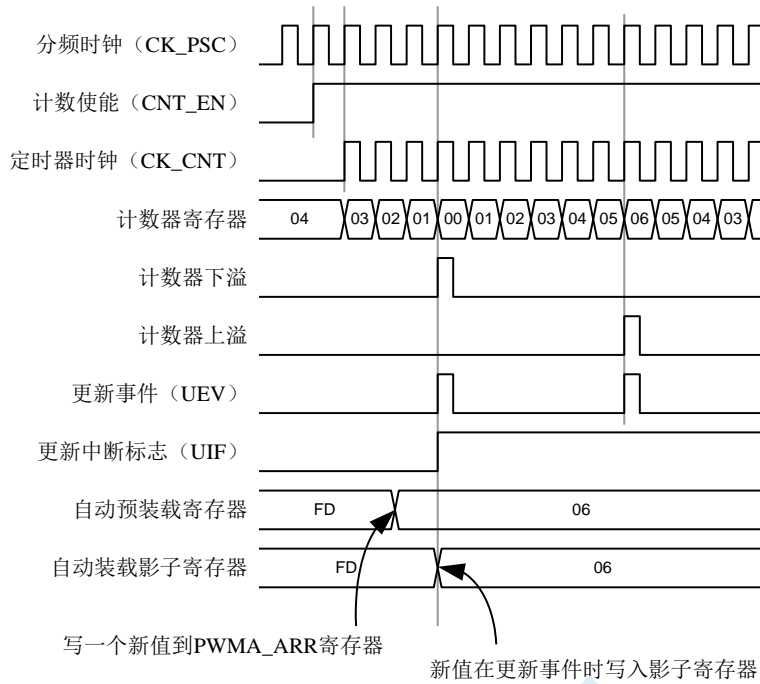
当发生更新事件时, 所有的寄存器都被更新, 硬件依据 URS 位更新标志位 (PWMA_SR 寄存器中的 UIF 位):

- 预分频器的缓存器被加载为预装载 (PWMA_PSC 寄存器) 的值。
- 当前的自动加载寄存器被更新为预装载值 (PWMA_ARR 寄存器中的内容)。

要注意到如果因为计数器溢出而产生更新, 自动重装载寄存器将在计数器重载入之前被更新, 因此下一个周期才是预期的值 (计数器被装载为新的值)。

以下是一些计数器在不同时钟频率下的操作的例子:

内部时钟分频因子为 1, PWMA_ARR=0x6, ARPE=1



使用中央对齐模式的提示:

- 启动中央对齐模式时, 计数器将按照原有的向上/向下的配置计数。也就是说 PWMA_CR1 寄存器中的 DIR 位将决定计数器是向上还是向下计数。此外, 软件不能同时修改 DIR 位和 CMS 位的值。
- 不推荐在中央对齐模式下, 计数器正在计数时写计数器的值, 这将导致不能预料的后果。具体的说:
 - 向计数器写入了比自动装载值更大的数值时 (PWMA_CNT > PWMA_ARR), 但计数器的计数方向不发生改变。例如计数器已经向上溢出, 但计数器仍然向上计数。
 - 向计数器写入了 0 或者 PWMA_ARR 的值, 但更新事件不发生。
- 安全使用中央对齐模式的计数器的方法是在启动计数器之前先用软件 (置位 PWMA_EGR 寄存器的 UG 位) 产生一个更新事件, 并且不在计数器计数时修改计数器的值。

18.3.7 重复计数器

时基单元解释了计数器向上/向下溢出时更新事件 (UEV) 是如何产生的, 然而事实上它只能在重复计数器的值达到 0 的时候产生。这个特性对产生 PWM 信号非常有用。

这意味着在每 N 次计数上溢或下溢时, 数据从预装载寄存器传输到影子寄存器 (PWMA_ARR 自动重载入寄存器, PWMA_PSC 预装载寄存器, 还有在比较模式下的捕获/比较寄存器 PWMA_CCRx), N 是 PWMA_RCR 重复计数寄存器中的值。

重复计数器在下述任一条件成立时递减:

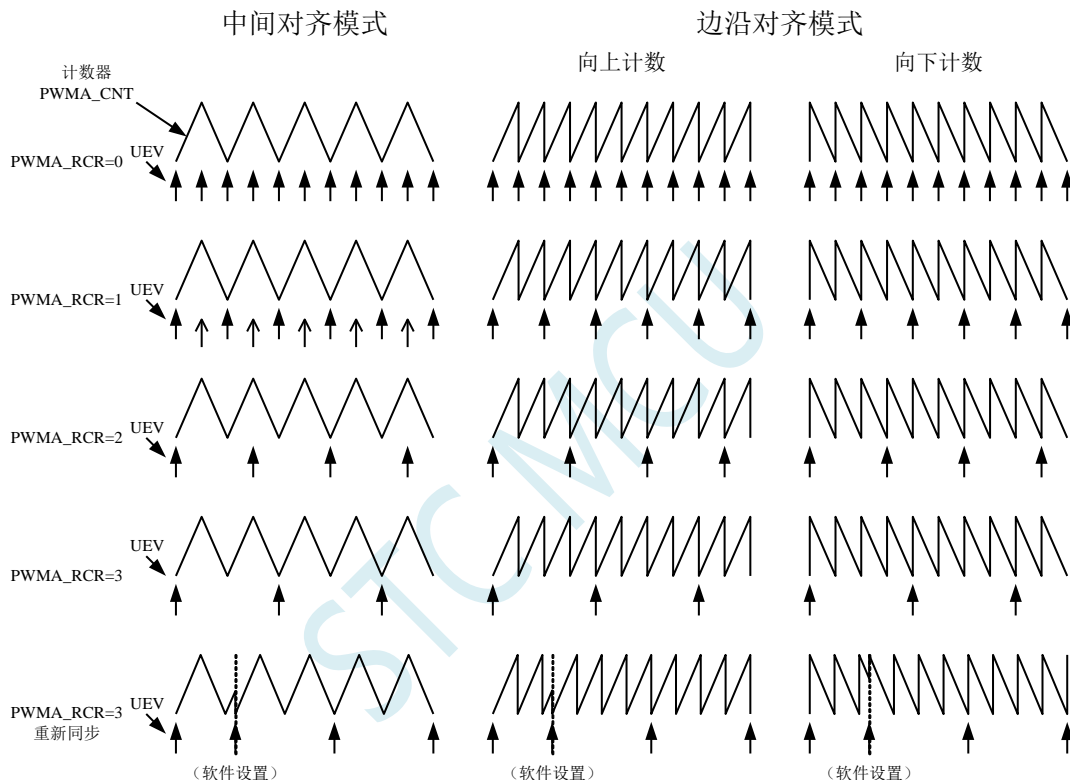
- 向上计数模式下每次计数器向上溢出时
- 向下计数模式下每次计数器向下溢出时

- 中央对齐模式下每次上溢和每次下溢时。

虽然这样限制了 PWM 的最大循环周期为 128，但它能够在每个 PWM 周期 2 次更新占空比。在中央对齐模式下，因为波形是对称的，如果每个 PWM 周期中仅刷新一次比较寄存器，则最大的分辨率为 $2 \cdot t_{CK_PSC}$ 。

重复计数器是自动加载的，重复速率由 PWMA_RCR 寄存器的值定义。当更新事件由软件产生（通过设置 PWMA_EGR 中的 UG 位）或者通过硬件的时钟/触发控制器产生，则无论重复计数器的值是多少，立即发生更新事件，并且 PWMA_RCR 寄存器中的内容被重载入到重复计数器。

不同模式下更新速率的例子，及 PWMA_RCR 的寄存器设置



18.4 时钟/触发控制器

时钟/触发控制器允许用户选择计数器的时钟源，输入触发信号和输出信号，

18.4.1 预分频时钟 (CK_PSC)

时基单元的预分频时钟 (CK_PSC) 可以由以下源提供：

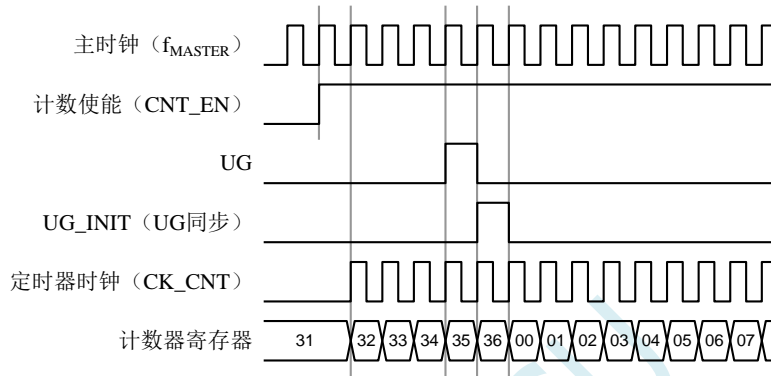
- 内部时钟 (fMASTER)
- 外部时钟模式 1：外部时钟输入 (TIX)
- 外部时钟模式 2：外部触发输入 ETR
- 内部触发输入 (ITRx)：使用一个定时器做为另一个定时器的预分频时钟。

18.4.2 内部时钟源 (fMASTER)

如果同时禁止了时钟/触发模式控制器和外部触发输入 (PWMA_SMCR 寄存器的 SMS=000, PWMA_ETR 寄存器的 ECE=0), 则 CEN、DIR 和 UG 位是实际上的控制位, 并且只能被软件修改 (UG 位仍被自动清除)。一旦 CEN 位被写成 1, 预分频器的时钟就由内部时钟提供。

下图描述了控制电路和向上计数器在普通模式下, 不带预分频器时的操作。

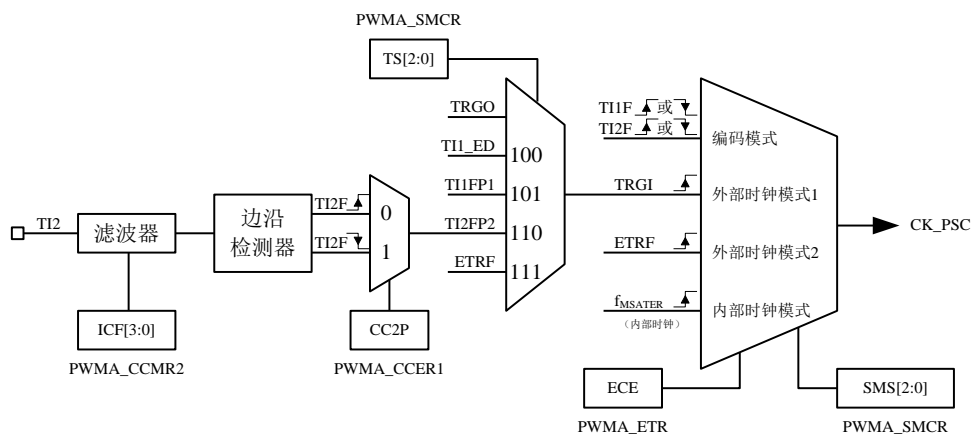
普通模式下的控制电路, f_{MASTER} 分频因子为 1



18.4.3 外部时钟源模式 1

当 PWMA_SMCR 寄存器的 SMS=111 时, 此模式被选中。计数器可以在选定输入端的每个上升沿或下降沿计数。

TI2 外部时钟连接例子



例如, 要配置向上计数器在 TI2 输入端的上升沿计数, 使用下列步骤:

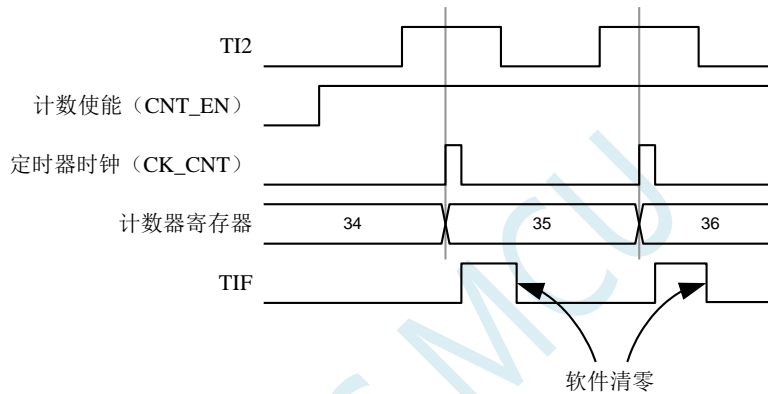
1. 配置 PWMA_CCMR2 寄存器的 CC2S=01, 使用通道 2 检测 TI2 输入的上升沿

2. 配置 PWMA_CCMR2 寄存器的 IC2F[3:0]位, 选择输入滤波器带宽 (如果不需要滤波器, 保持 IC2F=0000) 注: 捕获预分频器不用作触发, 所以不需要对它进行配置, 同样也不需要配置 TI2S 位, 他们仅用来选择输入捕获源。
3. 配置 PWMA_CCER1 寄存器的 CC2P=0, 选定上升沿极性
4. 配置 PWMA_SMCR 寄存器的 SMS=111, 配置计数器使用外部时钟模式 1
5. 配置 PWMA_SMCR 寄存器的 TS=110, 选定 TI2 作为输入源
6. 设置 PWMA_CR1 寄存器的 CEN=1, 启动计数器

当上升沿出现在 TI2, 计数器计数一次, 且触发标识位 (PWMA_SR1 寄存器的 TIF 位) 被置 1, 如果使能了中断 (在 PWMA_IER 寄存器中配置) 则会产生中断请求。

在 TI2 的上升沿和计数器实际时钟之间的延时取决于在 TI2 输入端的重新同步电路。

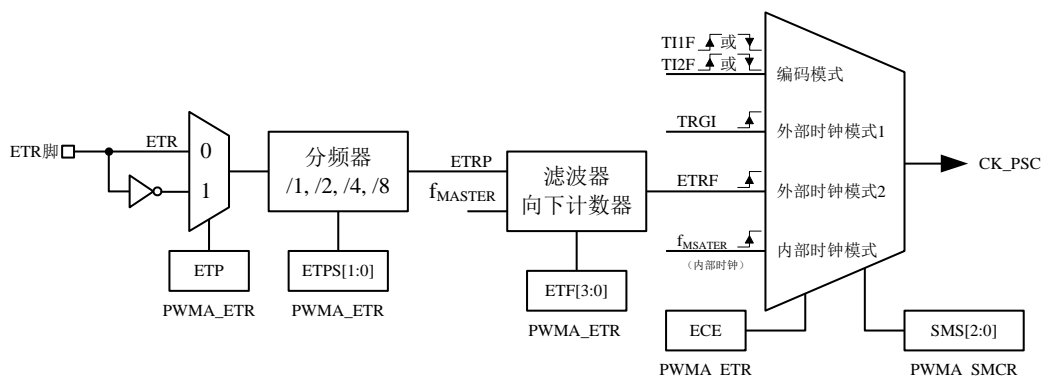
外部时钟模式 1 下的控制电路



18.4.4 外部时钟源模式 2

计数器能够在外部触发输入 ETR 信号的每一个上升沿或下降沿计数。将 PWMA_ETR 寄存器的 ECE 位写 1, 即可选定此模式。

外部触发输入的总体框图:

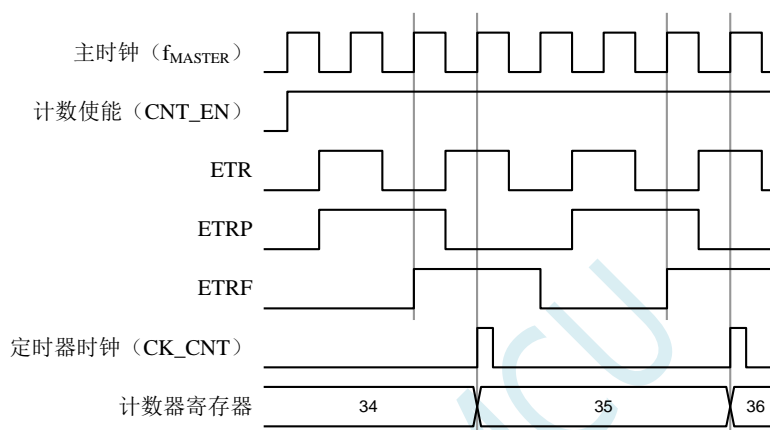


例如，要配置计数器在 ETR 信号的每 2 个上升沿时向上计数一次，需使用下列步骤：

1. 本例中不需要滤波器，配置 PWMA_ETR 寄存器的 ETF[3:0]=0000
2. 设置预分频器，配置 PWMA_ETR 寄存器的 ETPS[1:0]=01
3. 选择 ETR 的上升沿检测，配置 PWMA_ETR 寄存器的 ETP=0
4. 开启外部时钟模式 2，配置 PWMA_ETR 寄存器中的 ECE=1
5. 启动计数器，写 PWMA_CR1 寄存器的 CEN=1

计数器在每 2 个 ETR 上升沿计数一次。

外部时钟模式 2 下的控制电路



18.4.5 触发同步

PWMA 的计数器使用三种模式与外部的触发信号同步：

- 标准触发模式
- 复位触发模式
- 门控触发模式

标准触发模式

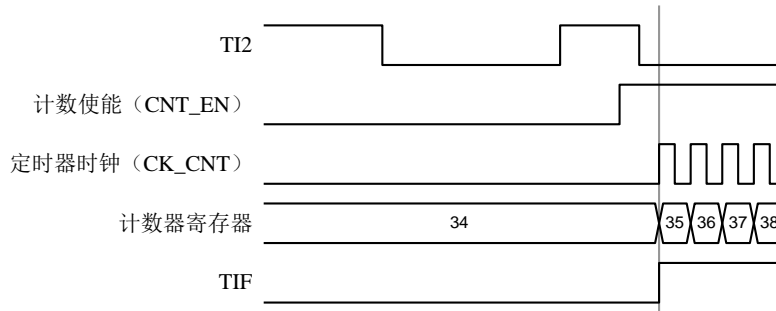
计数器的使能 (CEN) 依赖于选中的输入端上的事件。

在下面的例子中，计数器在 TI2 输入的上升沿开始向上计数：

1. 配置 PWMA_CCER1 寄存器的 CC2P=0，选择 TI2 的上升沿做为触发条件。
2. 配置 PWMA_SMCR 寄存器的 SMS=110，选择计数器为触发模式。配置 PWMA_SMCR 寄存器的 TS=110，选择 TI2 作为输入源。

当 TI2 出现一个上升沿时，计数器开始在内部时钟驱动下计数，同时置位 TIF 标志。TI2 上升沿和计数器启动计数之间的延时取决于 TI2 输入端的重同步电路。

标准触发模式的控制电路



复位触发模式

在发生一个触发输入事件时，计数器和它的预分频器能够重新被初始化。同时，如果 PWMA_CR1 寄存器的 URS 位为低，还产生一个更新事件 UEV，然后所有的预装载寄存器（PWMA_ARR，PWMA_CCRx）都会被更新。

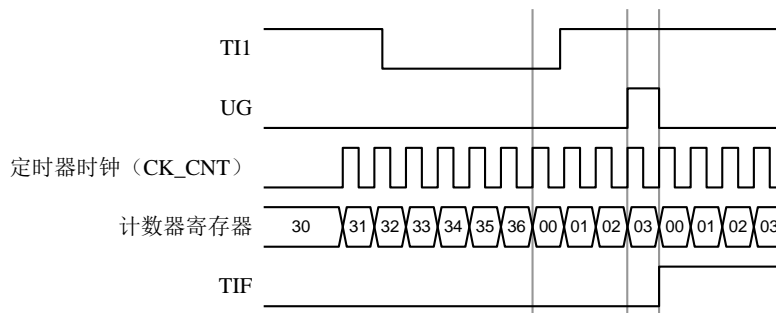
在以下的例子中，TI1 输入端的上升沿导致向上计数器被清零：

1. 配置 PWMA_CCER1 寄存器的 CC1P=0 来选择 TI1 的极性（只检测 TI1 的上升沿）。
2. 配置 PWMA_SMCR 寄存器的 SMS=100，选择定时器为复位触发模式。配置 PWMA_SMCR 寄存器的 TS=101，选择 TI1 作为输入源。
3. 配置 PWMA_CR1 寄存器的 CEN=1，启动计数器。

计数器开始依据内部时钟计数，然后正常计数直到 TI1 出现一个上升沿。此时，计数器被清零然后从 0 重新开始计数。同时，触发标志(PWMA_SR1 寄存器的 TIF 位)被置位，如果使能了中断(PWMA_IER 寄存器的 TIE 位)，则产生一个中断请求。

下图显示当自动重装载寄存器 PWMA_ARR=0x36 时的动作。在 TI1 上升沿和计数器的实际复位之间的延时取决于 TI1 输入端的重同步电路。

复位触发模式下的控制电路



门控触发模式

计数器由选中的输入端信号的电平使能。

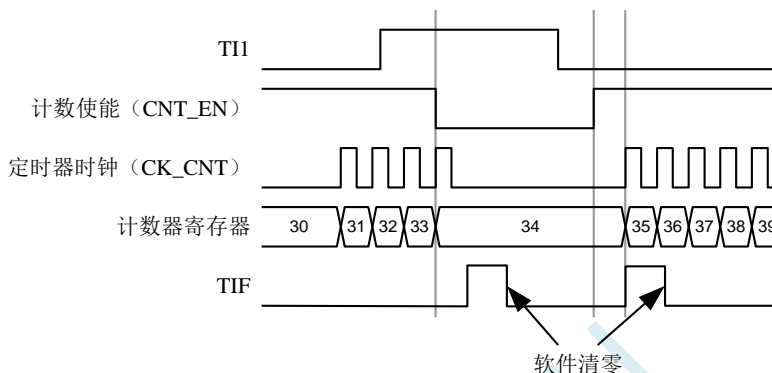
在如下的例子中，计数器只在 TI1 为低时向上计数：

1. 配置 PWMA_CCER1 寄存器的 CC1P=1 来确定 TI1 的极性（只检测 TI1 上的低电平）。

2. 配置 PWMA_SMCR 寄存器的 SMS=101, 选择定时器为门控触发模式, 配置 PWMA_SMCR 寄存器中 TS=101, 选择 TI1 作为输入源。
3. 配置 PWMA_CR1 寄存器的 CEN=1, 启动计数器 (在门控模式下, 如果 CEN=0, 则计数器不能启动, 不论触发输入电平如何)。

只要 TI1 为低, 计数器开始依据内部时钟计数, 一旦 TI1 变高则停止计数。当计数器开始或停止时 TIF 标志位都会被置位。TI1 上升沿和计数器实际停止之间的延时取决于 TI1 输入端的重同步电路。

门控触发模式下的控制电路



外部时钟模式 2 联合触发模式

外部时钟模式 2 可以与另一个输入信号的触发模式一起使用。例如, ETR 信号被用作外部时钟的输入, 另一个输入信号可用作触发输入 (支持标准触发模式, 复位触发模式和门控触发模式)。注意不能通过 PWMA_SMCR 寄存器的 TS 位把 ETR 配置成 TRGI。

在下面的例子中, 一旦在 TI1 上出现一个上升沿, 计数器即在 ETR 的每一个上升沿向上计数一次:

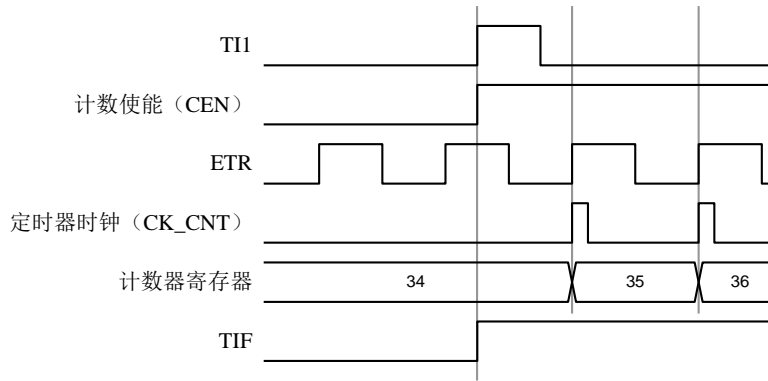
1. 通过 PWMA_ETR 寄存器配置外部触发输入电路。配置 ETPS=00 禁止预分频, 配置 ETP=0 监测 ETR 信号的上升沿, 配置 ECE=1 使能外部时钟模式 2。
2. 配置 PWMA_CCER1 寄存器的 CC1P=0 来选择 TI1 的上升沿触发。
3. 配置 PWMA_SMCR 寄存器的 SMS=110 来选择定时器为触发模式。配置 PWMA_SMCR 寄存器的 TS=101 来选择 TI1 作为输入源。

当 TI1 上出现一个上升沿时, TIF 标志被设置, 计数器开始在 ETR 的上升沿计数。

TI1 信号的上升沿和计数器实际时钟之间的延时取决于 TI1 输入端的重同步电路。

ETR 信号的上升沿和计数器实际时钟之间的延时取决于 ETRP 输入端的重同步电路。

外部时钟模式 2+触发模式下的控制电路



18.4.6 与 PWMB 同步

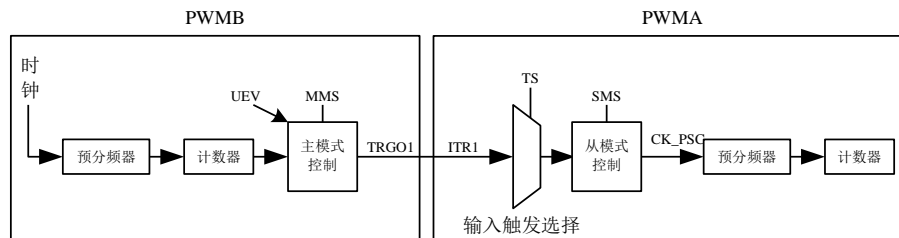
在芯片中，定时器在内部互相联结，用于定时器的同步或链接。当某个定时器配置成主模式时，可以输出触发信号（TRGO）到那些配置为从模式的定时器来完成复位操作、启动操作、停止操作或者作为那些定时器的驱动时钟。

使用 PWMB 的 TRGO 作为 PWMA 的预分频时钟

例如，用户可以配置 PWMB 作为 PWMA 的预分频时钟，需进行如下配置：

1. 配置 PWMB 为主模式，使得在每个更新事件（UEV）时输出周期性的触发信号。配置 PWMB_CR2 寄存器的 MMS=010，使每个更新事件时 TRGO 能输出一个上升沿。
2. PWMB 输出的 TRGO 信号链接到 PWMA。PWMA 需要配置成触发从模式，使用 ITR2 作为输入触发信号。以上操作可以通过配置 PWMA_SMCR 寄存器的 TS=010 实现。
3. 配置 PWMA_SMCR 寄存器的 SMS=111 将时钟/触发控制器设置为外部时钟模式 1。此操作将使 PWMB 输出的周期性触发信号 TRGO 的上升沿驱动 PWMA 的时钟。
4. 最后，置位 PWMB 的 CEN 位（PWMB_CR1 寄存器中），使能两个 PWM。

主/触发从模式的定时器例子



使用 PWMB 使能 PWMA

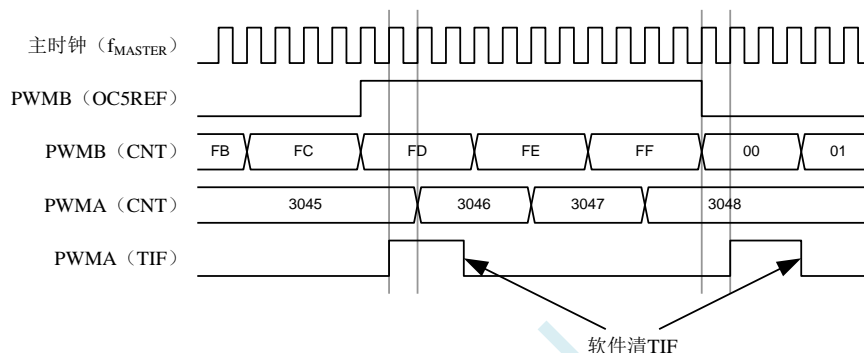
在本例中，我们用 PWMB 的比较输出使能 PWMA。PWMA 仅在 PWMB 的 OC1REF 信号为高时按照自己的驱动时钟计数。两个 PWM 都使用 4 分频的 f_{MASTER} 为时钟（ $f_{CK_CNT} = f_{MASTER}/4$ ）。

1. 配置 PWMB 为主模式，将比较输出信号（OC5REF）作为触发信号输出。（配置 PWMB_CR2 寄存器的 MMS=100）。

2. 配置 PWMB 的 OC5REF 信号的波形 (PWMB_CCMR1 寄存器)。
3. 配置 PWMA 把 PWMB 的输出作为自己的触发输入信号 (配置 PWMA_SMCR 寄存器的 TS=010)。
4. 配置 PWMA 为门控触发模式 (配置 PWMA_SMCR 寄存器的 SMS=101)。
5. 置位 CEN 位 (PWMA_CR1 寄存器)，使能 PWMA。
6. 置位 CEN 位 (PWMB_CR1 寄存器)，使能 PWMB。

注意：两个 PWM 的时钟并不同步，但仅影响 PWMA 的使能信号。

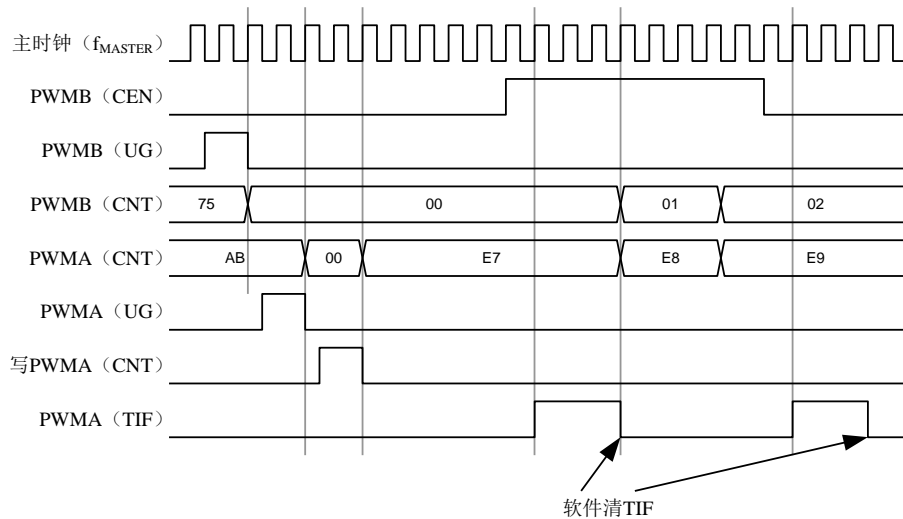
PWMB 的输出门控触发 PWMA



上图中，PWMA 的计数器和预分频器都没有在启动前初始化，所以都是从现有值开始计数的。如果在启动 PWMB 之前复位两个定时器，用户就可以写入期望的数值到 PWMA 的计数器，使之从指定值开始计数。对 PWMA 的复位操作可以通过软件写 PWMA_EGR 寄存器的 UG 位实现。

在下面这个例子中，我们使 PWMB 和 PWMA 同步。PWMB 为主模式并从 0 启动计数。PWMA 为触发从模式，并从 0xE7 启动计数。两个 PWM 采用相同的分频系数。当清除 PWMB_CR1 寄存器的 CEN 位时，PWMB 被禁止，同时 PWMA 停止计数。

1. 配置 PWMB 为主模式，将比较输出信号 (OC5REF) 作为触发信号输出。(配置 PWMB_CR2 寄存器的 MMS=100)。
2. 配置 PWMB 的 OC5REF 信号的波形 (PWMB_CCMR1 寄存器)。
3. 配置 PWMA 把 PWMB 的输出作为自己的触发输入信号 (配置 PWMA_SMCR 寄存器的 TS=010)。
4. 配置 PWMA 为门控触发模式 (配置 PWMA_SMCR 寄存器的 SMS=101)。
5. 通过对 UG 位 (PWMB_EGR 寄存器) 写 1，复位 PWMB。
6. 通过对 UG 位 (PWMA_EGR 寄存器) 写 1，复位 PWMA。
7. 将 0xE7 写入 PWMA 的计数器中 (PWMA_CNTRL)，初始化 PWMA。
8. 通过对 CEN 位 (PWMA_CR1 寄存器) 写 1，使能 PWMA。
9. 通过对 CEN 位 (PWMB_CR1 寄存器) 写 1，启动 PWMB。
10. 通过对 CEN 位 (PWMB_CR1 寄存器) 写 0，停止 PWMB。



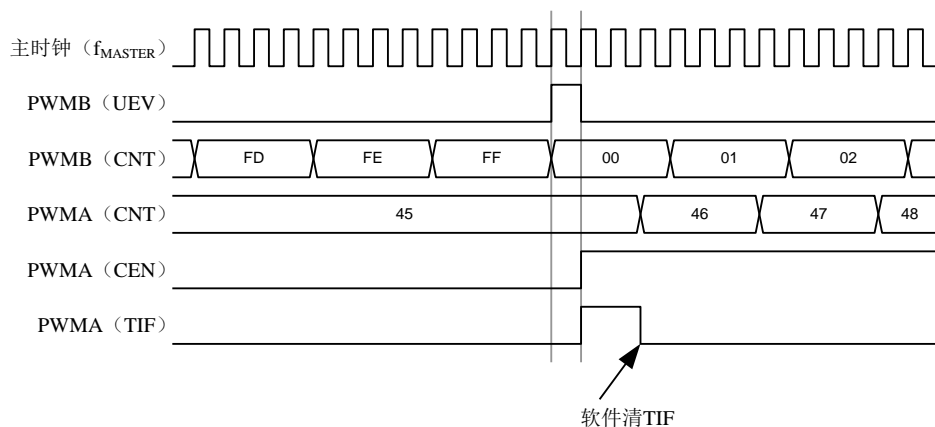
使用 PWMB 启动 PWMA

在本例中，我们用 PWMB 的更新事件来启动 PWMA。

PWMA 在 PWMB 发生更新事件时按照 PWMA 自己的驱动时钟从它的现有值开始计数(可以是非 0 值)。PWMA 在收到触发信号后自动使能 CEN 位，并开始计数，一直持续到用户向 PWMA_CR1 寄存器的 CEN 位写 0。两个 PWM 都使用 4 分频的 f_{MASTER} 作为驱动时钟 ($f_{CK_CNT} = f_{MASTER}/4$)。

1. 配置 PWMB 为主模式，输出更新信号 (UEV)。(配置 PWMB_CR2 寄存器的 MMS=010)。
2. 配置 PWMB 的周期 (PWMB_ARR 寄存器)。
3. 配置 PWMA 用 PWMB 的输出作为输入的触发信号 (配置 PWMA_SMCR 寄存器的 TS=010)。
4. 配置 PWMA 为触发模式 (配置 PWMA_SMCR 寄存器的 SMS=110)。
5. 置位 CEN 位 (PWMB_CR1 寄存器) 启动 PWMB。

PWMB 的更新事件 (PWMB-UEV) 触发 PWMA



如同前面的例子，用户也可以在启动计数器前对它们初始化。

用外部信号同步的触发两个 PWM

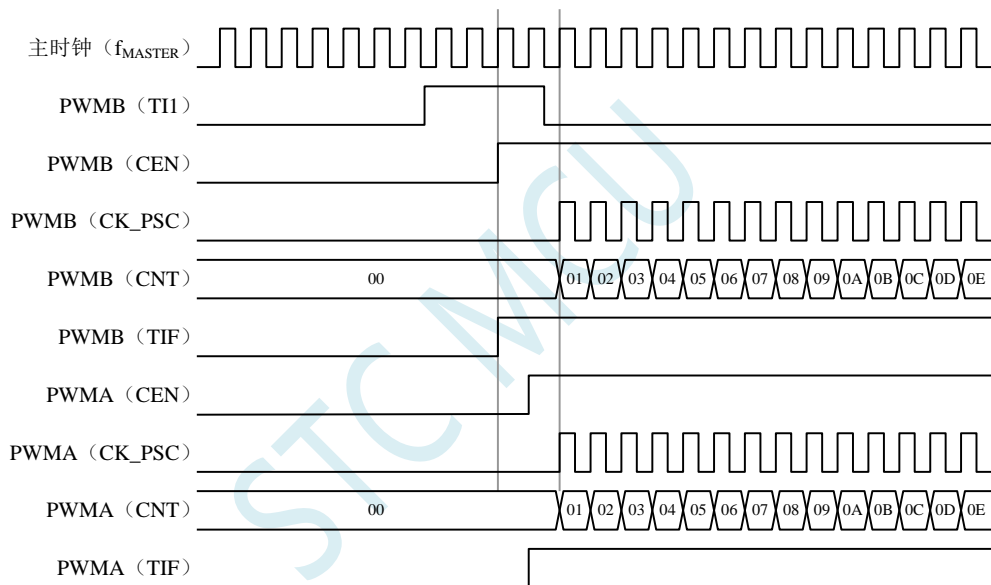
在本例中, 使用 TI1 的上升沿使能 PWMB, 并同时使能 PWMA。为了保持定时器的对齐, PWMB 需要配置成主/从模式 (对于 TI1 信号为从模式, 对于 PWMA 为主模式)。

1. 配置 PWMB 为主模式, 以输出使能信号作为 PWMA 的触发 (配置 PWMB_CR2 寄存器的 MMS=001)。
2. 配置 PWMB 为从模式, 把 TI1 信号作为输入的触发信号 (配置 PWMB_SMCR 寄存器的 TS=100)。
3. 配置 PWMB 的触发模式 (配置 PWMB_SMCR 寄存器的 SMS=110)。
4. 配置 PWMB 为主/从模式 (配置 PWMB_SMCR 寄存器的 MSM=1)。
5. 配置 PWMA 以 PWMB 的输出为输入触发信号 (配置 PWMA_SMCR 寄存器的 TS=010)。
6. 配置 PWMA 的触发模式 (配置 PWMA_SMCR 寄存器的 SMS=110)。

当 TI1 上出现上升沿时, 两个定时器同步的开始计数, 并且 TIF 位都被置起。

注意: 在本例中, 两个定时器在启动前都进行了初始化 (设置 UG 位), 所以它们都从 0 开始计数, 但是用户也可以通过修改计数器寄存器 (PWMA_CNT) 来插入一个偏移量, 这样的话, 在 PWMB 的 CK_PSC 信号和 CNT_EN 信号间会插入延时。

PWMB 的 TI1 信号触发 PWMB 和 PWMA

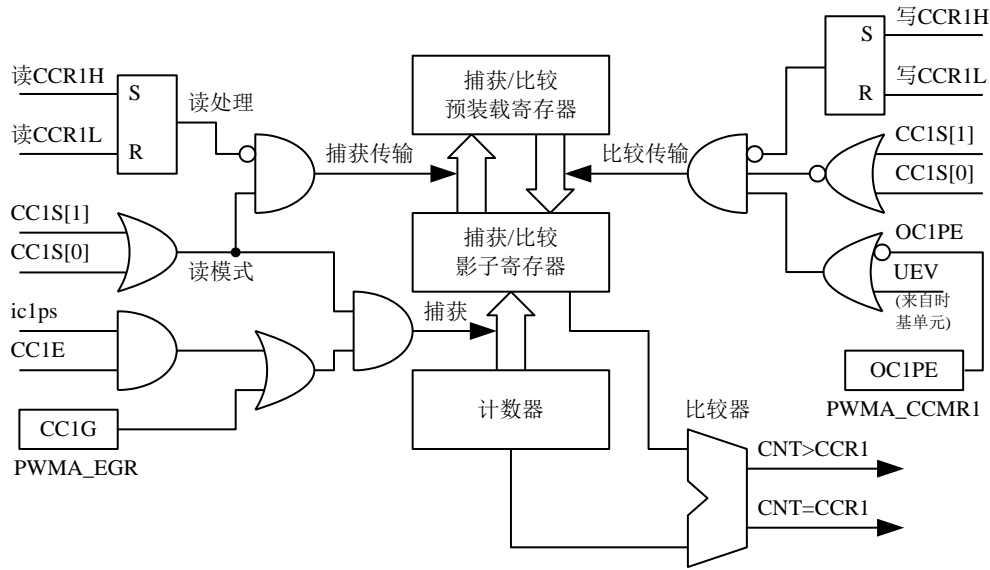


18.5 捕获/比较通道

PWM1P、PWM2P、PWM3P、PWM4P 可以用作输入捕获, PWM1P/PWM1N、PWM2P/PWM2N、PWM3P/PWM3N、PWM4P/PWM4N 可以输出比较, 这个功能可以通过配置捕获/比较通道模式寄存器 (PWMA_CCMRi) 的 CCiS 通道选择位来实现, 此处的 i 代表 1~4 的通道数。

每一个捕获/比较通道都是围绕着一个捕获/比较寄存器 (包含影子寄存器) 来构建的, 包括捕获的输入部分 (数字滤波、多路复用和预分频器) 和输出部分 (比较器和输出控制)。

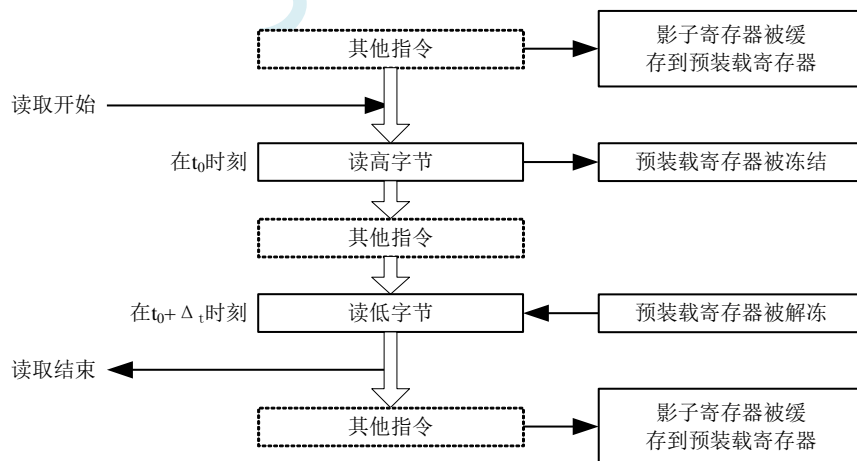
捕获/比较通道 1 的主要电路 (其他通道与此类似)



捕获/比较模块由一个预装载寄存器和一个影子寄存器组成。读写过程仅操作预装载寄存器。在捕获模式下，捕获发生在影子寄存器上，然后再复制到预装载寄存器中。在比较模式下，预装载寄存器的内容被复制到影子寄存器中，然后影子寄存器的内容和计数器进行比较。

当通道被配置成输出模式时（PWMA_CCMRi 寄存器的 CCiS=0），可以随时访问 PWMA_CCRi 寄存器。

当通道被配置成输入模式时，对 PWMA_CCRi 寄存器的读操作类似于计数器的读操作。当捕获发生时，计数器的内容被捕获到 PWMA_CCRi 影子寄存器，随后再复制到预装载寄存器中。在读操作进行中，预装载寄存器是被冻结的。



上图描述了 16 位的 CCRi 寄存器的读操作流程，被缓存的数据将保持不变直到读流程结束。

在整个读流程结束后，如果仅仅读了 PWMA_CCRiL 寄存器，返回计数器数值的低位（LS）。

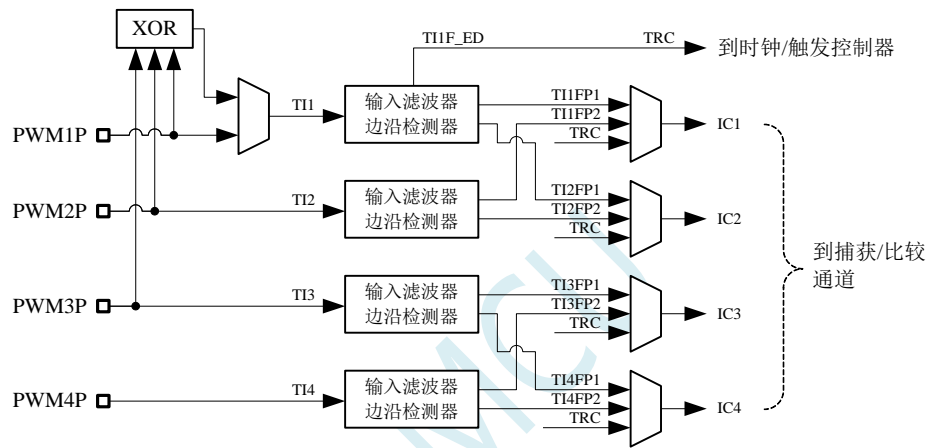
如果在读了低位（LS）数据以后再读高位（MS）数据，将不再返回同样的低位数据。

18.5.1 16 位 PWMA_CCRi 寄存器的写流程

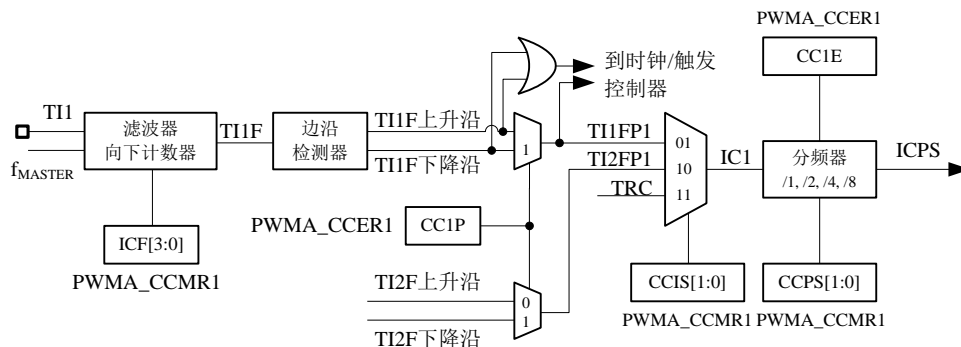
16 位 PWMA_CCRi 寄存器的写操作通过预装载寄存器完成。必需使用两条指令来完成整个流程，一条指令对应一个字节。必需先写高位字节（MS）。在写高位字节（MS）时，影子寄存器的更新被禁止直到低位字节（LS）的写操作完成。

18.5.2 输入模块

输入模块的框图



如图，输入部分对相应的 TI_x 输入信号采样，并产生一个滤波后的信号 TI_xF。然后，一个带极性选择的边缘监测器产生一个信号（TI_xFP_x），它可以作为触发模式控制器的输入触发或者作为捕获控制。该信号通过预分频后进入捕获寄存器（IC_xPS）。



18.5.3 输入捕获模式

在输入捕获模式下，当检测到 IC_i 信号上相应的边沿后，计数器的当前值被锁存到捕获/比较寄存器

(PWMA_CCRx) 中。当发生捕获事件时, 相应的 CCiIF 标志 (PWMA_SR 寄存器) 被置 1。

如果 PWMA_IER 寄存器的 CCiIE 位被置位, 也就是使能了中断, 则将产生中断请求。如果发生捕获事件时 CCiIF 标志已经为高, 那么重复捕获标志 CCiOF (PWMA_SR2 寄存器) 被置 1。写 CCiIF=0 或读取存储在 PWMA_CCRiL 寄存器中的捕获数据都可清除 CCiIF。写 CCiOF=0 可清除 CCiOF。

PWM 输入信号上升沿时捕获

以下例子说明如何在 TI1 输入的上升沿时捕获计数器的值到 PWMA_CCR1 寄存器中, 步骤如下:

1. 选择有效输入端: 例如 PWMA_CCR1 连接到 TI1 输入, 所以写入 PWMA_CCR1 寄存器中的 CC1S=01, 此时通道被配置为输入, 并且 PWMA_CCR1 寄存器变为只读。
2. 根据输入信号 TIi 的特点, 可通过配置 PWMA_CCMRi 寄存器中的 ICiF 位来设置相应的输入滤波器的滤波时间。假设输入信号在最多 5 个时钟周期的时间内抖动, 我们须配置滤波器的带宽长于 5 个时钟周期; 因此我们可以连续采样 8 次, 以确认在 TI1 上一次真实的边沿变换, 即在 TIMi_CCMR1 寄存器中写入 IC1F=0011, 此时, 只有连续采样到 8 个相同的 TI1 信号, 信号才为有效 (采样频率为 f_{MASTER})。
3. 选择 TI1 通道的有效转换边沿, 在 PWMA_CCER1 寄存器中写入 CC1P=0 (上升沿)。
4. 配置输入预分频器。在本例中, 我们希望捕获发生在每一个有效的电平转换时刻, 因此预分频器被禁止 (写 PWMA_CCMR1 寄存器的 IC1PS=00)。
5. 设置 PWMA_CCER1 寄存器的 CC1E=1, 允许捕获计数器的值到捕获寄存器中。
6. 如果需要, 通过设置 PWMA_IER 寄存器中的 CC1IE 位允许相关中断请求。

当发生一个输入捕获时:

- 当产生有效的电平转换时, 计数器的值被传送到 PWMA_CCR1 寄存器。
- CC1IF 标志被设置。当发生至少 2 个连续的捕获时, 而 CC1IF 未曾被清除时, CC1OF 也被置 1。
- 如设置了 CC1IE 位, 则会产生一个中断。

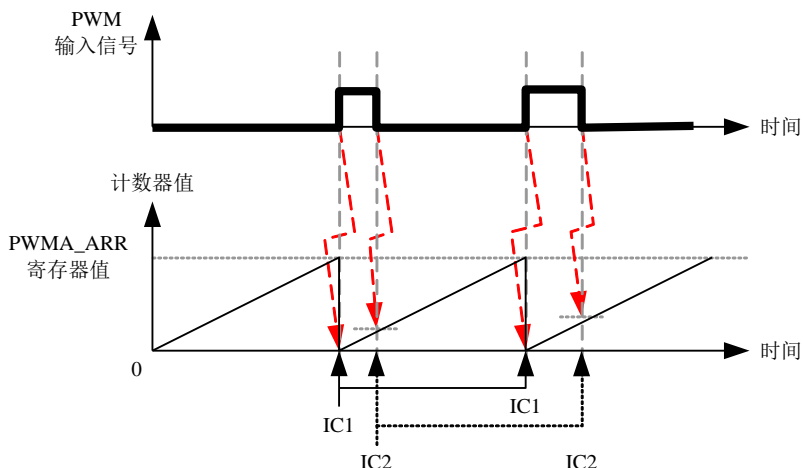
为了处理捕获溢出事件 (CC1OF 位), 建议在读出重复捕获标志之前读取数据, 这是为了避免丢失在读出捕获溢出标志之后和读取数据之前可能产生的重复捕获信息。

注意: 设置 PWMA_EGR 寄存器中相应的 CCiG 位, 可以通过软件产生输入捕获中断。

PWM 输入信号测量

该模式是输入捕获模式的一个特例, 除下列区别外, 操作与输入捕获模式相同:

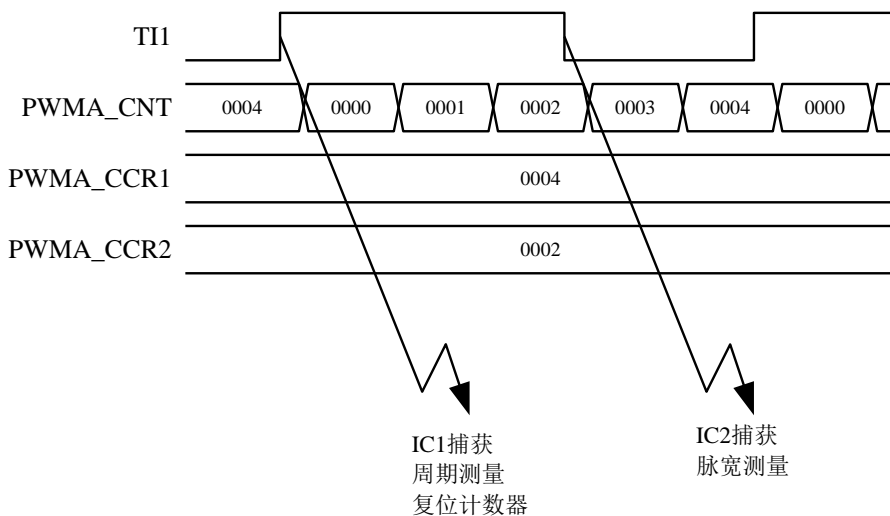
- 两个 ICi 信号被映射至同一个 TIi 输入。
- 这两个 ICi 信号的有效边沿的极性相反。
- 其中一个 TIiFP 信号被作为触发输入信号, 而触发模式控制器被配置成复位触发模式。



例如，你可以用以下方式测量 TI1 上输入的 PWM 信号的周期（PWMA_CCR1 寄存器）和占空比（PWMA_CCR2 寄存器）。（具体取决于 f_{MASTER} 的频率和预分频器的值）

1. 选择 PWMA_CCR1 的有效输入：置 PWMA_CCMR1 寄存器的 CC1S=01（选中 TI1）。
2. 选择 TI1FP1 的有效极性（用来捕获数据到 PWMA_CCR1 中和清除计数器）：置 CC1P=0（上升沿有效）。
3. 选择 PWMA_CCR2 的有效输入：置 PWMA_CCMR2 寄存器的 CC2S=10（选中 TI1FP2）。
4. 选择 TI1FP2 的有效极性（捕获数据到 PWMA_CCR2）：置 CC2P=1（下降沿有效）。
5. 选择有效的触发输入信号：置 PWMA_SMCR 寄存器中的 TS=101（选择 TI1FP1）。
6. 配置触发模式控制器为复位触发模式：置 PWMA_SMCR 中的 SMS=100。
7. 使能捕获：置 PWMA_CCER1 寄存器中 CC1E=1，CC2E=1。

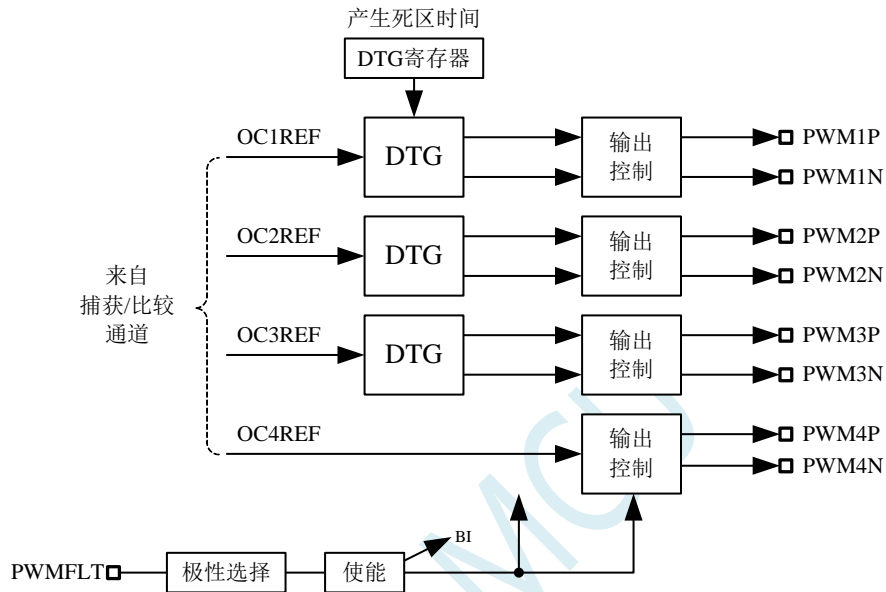
PWM 输入信号测量实例



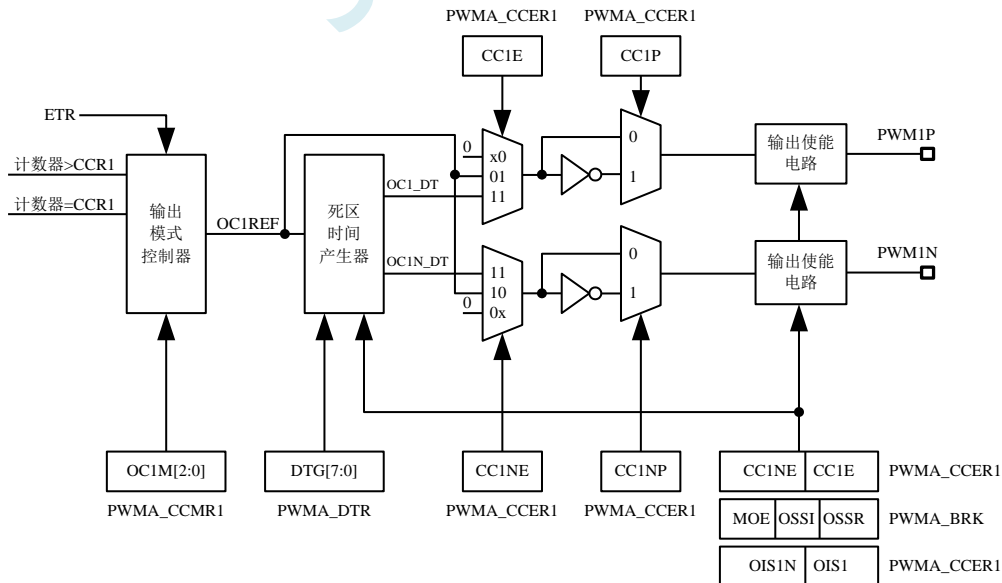
18.5.4 输出模块

输出模块会产生一个用来做参考的中间波形，称为 OCiREF（高有效）。刹车功能和极性的处理都在模块的最后处理。

输出模块框图



通道 1 详细的带互补输出的输出模块框图（其他通道类似）



18.5.5 强制输出模式

在输出模式下，输出比较信号能够直接由软件强制为高或低状态，而不依赖于输出比较寄存器和计数器间的比较结果。

置 PWMA_CCMRi 寄存器的 OCiM=101，可强制 OCiREF 信号为低。

置 PWMA_CCMRi 寄存器的 OCiM=100，可强制 OCiREF 信号为低。

OCi/OCiN 的输出是高还是低则取决于 CCIp/CCiNP 极性标志位。

该模式下，在 PWMA_CCRi 影子寄存器和计数器之间的比较仍然在进行，相应的标志也会被修改，也仍然会产生相应的中断。

18.5.6 输出比较模式

此模式用来控制一个输出波形或者指示一段给定的时间已经达到。

当计数器与捕获/比较寄存器的内容相匹配时，有如下操作：

- 根据不同的输出比较模式，相应的 OCi 输出信号：
 - 保持不变（OCiM=000）
 - 设置为有效电平（OCiM=001）
 - 设置为无效电平（OCiM=010）
 - 翻转（OCiM=011）
- 设置中断状态寄存器中的标志位（PWMA_SR1 寄存器中的 CCIIF 位）。
- 若设置了相应的中断使能位（PWMA_IER 寄存器中的 CCiIE 位），则产生一个中断。

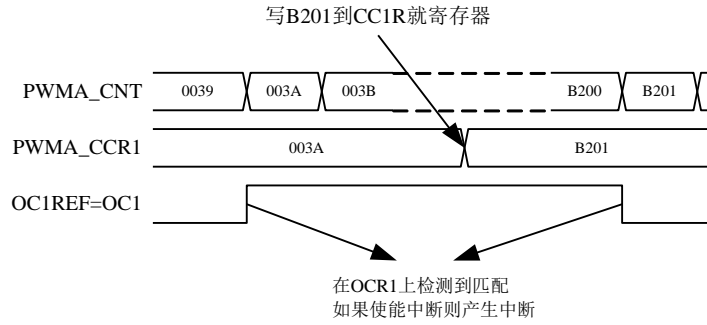
PWMA_CCMRi 寄存器的 OCiM 位用于选择输出比较模式，而 PWMA_CCMRi 寄存器的 CCIp 位用于选择有效和无效的电平极性。PWMA_CCMRi 寄存器的 OCiPE 位用于选择 PWMA_CCRi 寄存器是否需要使用预装载寄存器。在输出比较模式下，更新事件 UEV 对 OCiREF 和 OCi 输出没有影响。时间精度为计数器的一个计数周期。输出比较模式也能用来输出一个单脉冲。

输出比较模式的配置步骤：

1. 选择计数器时钟（内部、外部或者预分频器）。
2. 将相应的数据写入 PWMA_ARR 和 PWMA_CCRi 寄存器中。
3. 如果要产生一个中断请求，设置 CCiIE 位。
4. 选择输出模式步骤：
 1. 设置 OCiM=011，在计数器与 CCRi 匹配时翻转 OCiM 管脚的输出
 2. 设置 OCiPE = 0，禁用预装载寄存器
 3. 设置 CCIp = 0，选择高电平为有效电平
 4. 设置 CCiE = 1，使能输出
 5. 设置 PWMA_CR1 寄存器的 CEN 位来启动计数器

PWMA_CCRi 寄存器能够在任何时候通过软件进行更新以控制输出波形，条件是未使用预装载寄存器（OCiPE=0），否则 PWMA_CCRi 的影子寄存器只能在发生下一次更新事件时被更新。

输出比较模式，翻转 OC1



18.5.7 PWM 模式

脉冲宽度调制 (PWM) 模式可以产生一个由 PWMA_ARR 寄存器确定频率, 由 PWMA_CCRi 寄存器确定占空比的信号。

在 PWMA_CCMRi 寄存器中的 OCiM 位写入 110 (PWM 模式 1) 或 111 (PWM 模式 2), 能够独立地设置每个 OCi 输出通道产生一路 PWM。必须设置 PWMA_CCMRi 寄存器的 OCiPE 位使能相应的预装载寄存器, 也可以设置 PWMA_CR1 寄存器的 ARPE 位使能自动重载的预装载寄存器 (在向上计数模式或中央对称模式中)。

由于仅当发生一个更新事件的时候, 预装载寄存器才能被传送到影子寄存器, 因此在计数器开始计数之前, 必须通过设置 PWMA_EGR 寄存器的 UG 位来初始化所有的寄存器。

OCi 的极性可以通过软件在 PWMA_CCERi 寄存器中的 CCIp 位设置, 它可以设置为高电平有效或低电平有效。OCi 的输出使能通过 PWMA_CCERi 和 PWMA_BKR 寄存器中的 CCIe、MOE、OISi、OSSR 和 OSSi 位的组合来控制。

在 PWM 模式 (模式 1 或模式 2) 下, PWMA_CNT 和 PWMA_CCRi 始终在进行比较, (依据计数器的计数方向) 以确定是否符合 $PWMA_CCRi \leq PWMA_CNT$ 或者 $PWMA_CNT \leq PWMA_CCRi$ 。

根据 PWMA_CR1 寄存器中 CMS 位域的状态, 定时器能够产生边沿对齐的 PWM 信号或中央对齐的 PWM 信号。

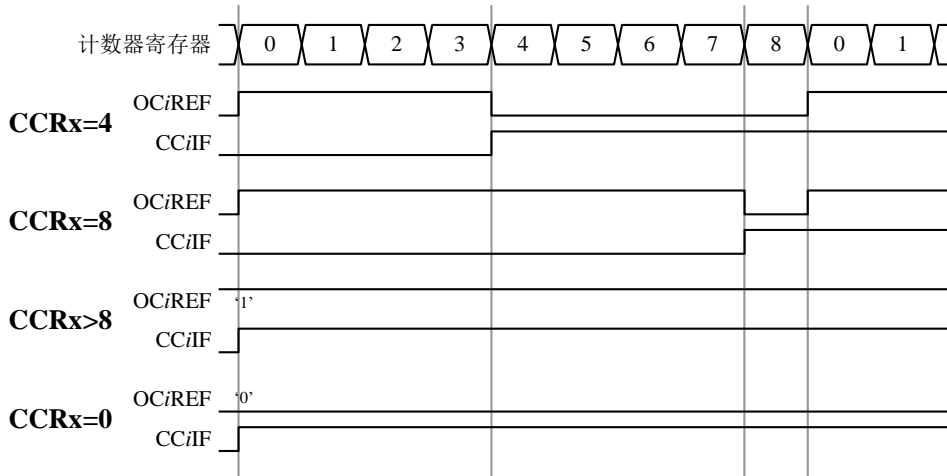
PWM 边沿对齐模式

向上计数配置

当 PWMA_CR1 寄存器中的 DIR 位为 0 时, 执行向上计数。

下面是一个 PWM 模式 1 的例子。当 $PWMA_CNT < PWMA_CCRi$ 时, PWM 参考信号 OCiREF 为高, 否则为低。如果 PWMA_CCRi 中的比较值大于自动重载值 (PWMA_ARR), 则 OCiREF 保持为 '1'。如果比较值为 0, 则 OCiREF 保持为 '0'。

边沿对齐, PWM 模式 1 的波形 (ARR=8)



向下计数的配置

当 PWMA_CR1 寄存器的 DIR 位为 1 时，执行向下计数。

在 PWM 模式 1 时，当 PWMA_CNT>PWMA_CCRi 时参考信号 OCiREF 为低，否则为高。如果 PWMA_CCRi 中的比较值大于 PWMA_ARR 中的自动重装载值，则 OCiREF 保持为‘1’。该模式下不能产生 0% 的 PWM 波形。

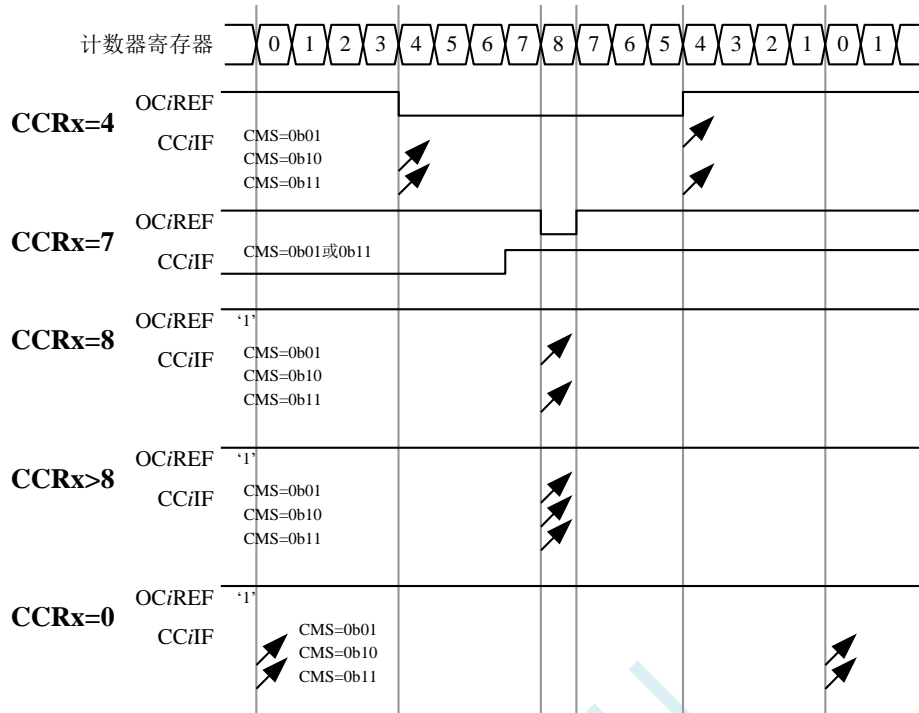
PWM 中央对齐模式

当 PWMA_CR1 寄存器中的 CMS 位不为‘00’时为中央对齐模式（所有其他的配置对 OCiREF/OCi 信号都有相同的作用）。

根据不同的 CMS 位的设置，比较标志可以在计数器向上计数，向下计数，或向上和向下计数时被置 1。PWMA_CR1 寄存器中的计数方向位（DIR）由硬件更新，不要用软件修改它。

下面给出了一些中央对齐的 PWM 波形的例子：

- PWMA_ARR=8
- PWM 模式 1
- 标志位在以下三种情况下被置位：
 - 只有在计数器向下计数时（CMS=01）
 - 只有在计数器向上计数时（CMS=10）
 - 在计数器向上和向下计数时（CMS=11）
- 中央对齐的 PWM 波形（ARR=8）



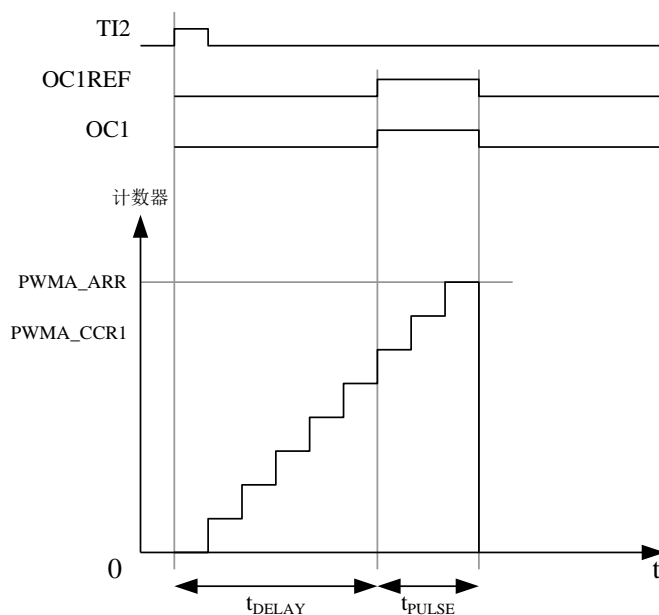
单脉冲模式

单脉冲模式（OPM）是前述众多模式的一个特例。这种模式允许计数器响应一个激励，并在一个程序可控的延时之后产生一个脉宽可控的脉冲。

可以通过时钟/触发控制器启动计数器，在输出比较模式或者 PWM 模式下产生波形。设置 PWMA_CR1 寄存器的 OPM 位将选择单脉冲模式，此时计数器自动地在下一个更新事件 UEV 时停止。仅当比较值与计数器的初始值不同时，才能产生一个脉冲。启动之前（当定时器正在等待触发），必须如下配置：

- 向上计数方式：计数器 $CNT < CCRi \leq ARR$ ，
- 向下计数方式：计数器 $CNT > CCRi$ 。

单脉冲模式图例



例如，在从 TI2 输入脚上检测到一个上升沿之后延迟 t_{DELAY} ，在 OC1 上产生一个 t_{PULSE} 宽度的正脉冲：（假定 IC2 作为触发 1 通道的触发源）

- 置 PWMA_CCMR2 寄存器的 CC2S=01，把 IC2 映射到 TI2。
- 置 PWMA_CCER1 寄存器的 CC2P=0，使 IC2 能够检测上升沿。
- 置 PWMA_SMCR 寄存器的 TS=110，使 IC2 作为时钟/触发控制器的触发源（TRGI）。
- 置 PWMA_SMCR 寄存器的 SMS=110（触发模式），IC2 被用来启动计数器。OPM 的波形由写入比较寄存器的数值决定（要考虑时钟频率和计数器预分频器）。
- t_{DELAY} 由 PWMA_CCR1 寄存器中的值定义。
- t_{PULSE} 由自动装载值和比较值之间的差值定义（PWMA_ARR – PWMA_CCR1）。
- 假定当发生比较匹配时要产生从 0 到 1 的波形，当计数器达到预装载值时要产生一个从 1 到 0 的波形，首先要置 PWMA_CCMR1 寄存器的 OCiM=111，进入 PWM 模式 2，根据需要有选择的设置 PWMA_CCMR1 寄存器的 OCiPE=1，置位 PWMA_CR1 寄存器中的 ARPE，使能预装载寄存器，然后在 PWMA_CCR1 寄存器中填写比较值，在 PWMA_ARR 寄存器中填写自动装载值，设置 UG 位来产生一个更新事件，然后等待在 TI2 上的一个外部触发事件。

在这个例子中，PWMA_CR1 寄存器中的 DIR 和 CMS 位应该置低。

因为只需要一个脉冲，所以设置 PWMA_CR1 寄存器中的 OPM=1，在下一个更新事件（当计数器从自动装载值翻转到 0）时停止计数。

OCx 快速使能（特殊情况）

在单脉冲模式下，对 Ti_i 输入脚的边沿检测会设置 CEN 位以启动计数器，然后计数器和比较值间的比较操作产生了单脉冲的输出。但是这些操作需要一定的时钟周期，因此它限制了可得到的最小延时 t_{DELAY} 。

如果要以最小延时输出波形，可以设置 PWMA_CCMRi 寄存器中的 OCiFE 位，此时强制 OCiREF（和 OCx）直接响应激励而不再依赖比较的结果，输出的波形与比较匹配时的波形一样。OCiFE 只在

通道配置为 PWMA 和 PWMB 模式时起作用。

互补输出和死区插入

PWMA 能够输出两路互补信号，并且能够管理输出的瞬时关断和接通，这段时间通常被称为死区，用户应该根据连接的输出器件和它们的特性（电平转换的延时、电源开关的延时等）来调整死区时间。

配置 PWMA_CCERi 寄存器中的 CCIp 和 CCIpN 位，可以为每一个输出独立地选择极性（主输出 OCi 或互补输出 OCiN）。

互补信号 OCi 和 OCiN 通过下列控制位的组合进行控制：PWMA_CCERi 寄存器的 CCIe 和 CCIeN 位，PWMA_BKR 寄存器中的 MOE、OISi、OISiN、OSSI 和 OSSR 位。特别的是，在转换到 IDLE 状态时（MOE 下降到 0）死区控制被激活。

同时设置 CCIe 和 CCIeN 位将插入死区，如果存在刹车电路，则还要设置 MOE 位。每一个通道都有一个 8 位的死区发生器。参考信号 OCiREF 可以产生 2 路输出 OCi 和 OCiN。

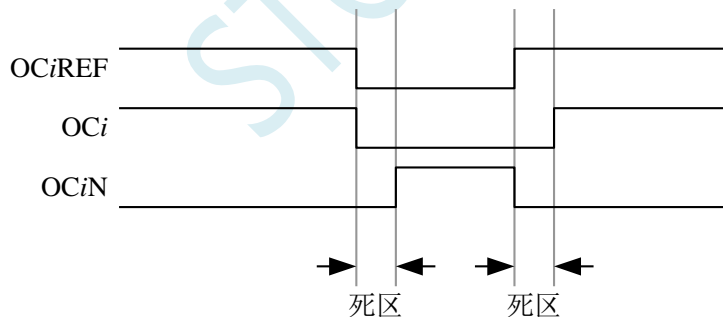
如果 OCi 和 OCiN 为高有效：

- OCi 输出信号与参考信号相同，只是它的上升沿相对于参考信号的上升沿有一个延迟。
- OCiN 输出信号与参考信号相反，只是它的上升沿相对于参考信号的下降沿有一个延迟。

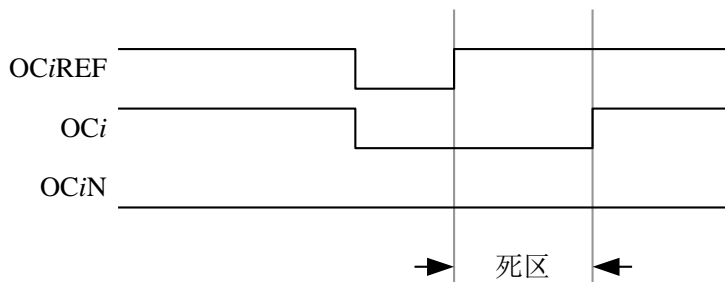
如果延迟大于当前有效的输出宽度（OCi 或者 OCiN），则不会产生相应的脉冲。

下列几张图显示了死区发生器的输出信号和当前参考信号 OCiREF 之间的关系。（假设 CCIp=0、CCIpN=0、MOE=1、CCIe=1 并且 CCIeN=1）

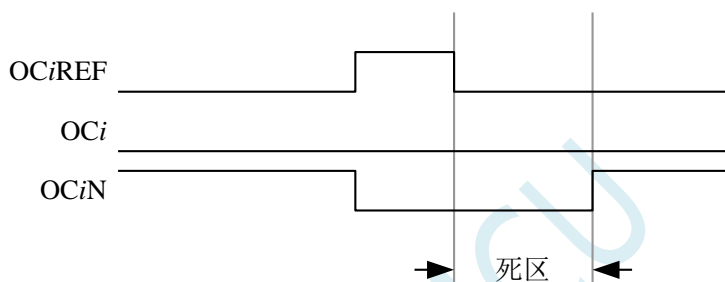
带死区插入的互补输出



死区波形延迟大于负脉冲



死区波形延迟大于正脉冲



每一个通道的死区延时都是相同的，是由 PWMA_DTR 寄存器中的 DTG 位编程配置。

重定向 OCiREF 到 OCi 或 OCiN

在输出模式下（强制输出、输出比较或 PWM 输出），通过配置 PWMA_CCERi 寄存器的 CCiE 和 CCiNE 位，OCiREF 可以被重定向到 OCi 或者 OCiN 的输出。

这个功能可以在互补输出处于无效电平时，在某个输出上送出一个特殊的波形（例如 PWM 或者静态有效电平）。另一个作用是，让两个输出同时处于无效电平，或同时处于有效电平（此时仍然是带死区的互补输出）。

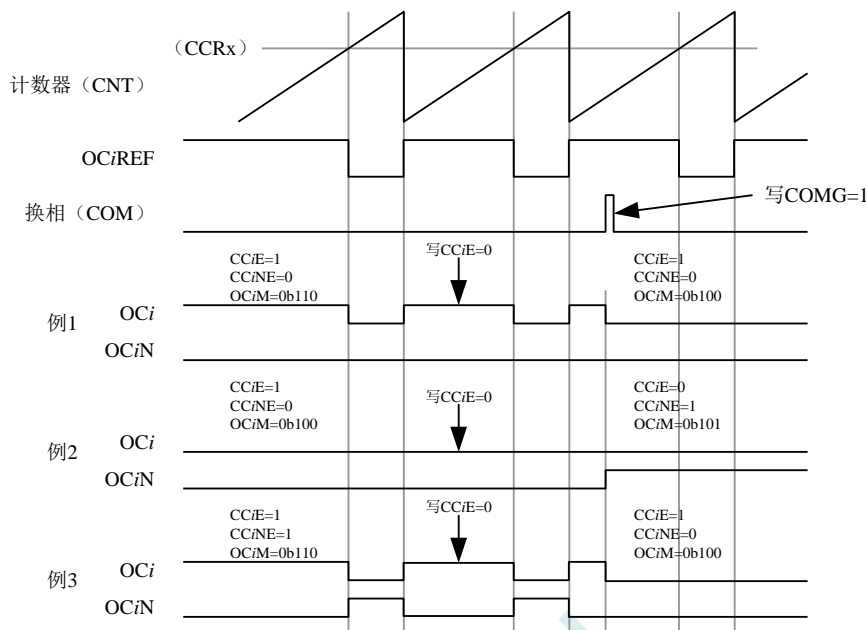
注：当只使能 OCiN（CCiE=0, CCiNE=1）时，它不会反相，而当 OCiREF 变高时立即有效。例如，如果 CCiNP=0，则 OCiN=OCiREF。另一方面，当 OCi 和 OCiN 都被使能时（CCiE=CCiNE=1），当 OCiREF 为高时 OCi 有效；而 OCiN 相反，当 OCiREF 低时 OCiN 变为有效。

针对马达控制的六步 PWM 输出

当在一个通道上需要互补输出时，预装载位有 OCiM、CCiE 和 CCiNE。在发生 COM 换相事件时，这些预装载位被传送到影子寄存器位。这样你就可以预先设置好下一步骤配置，并在同一个时刻同时修改所有通道的配置。COM 可以通过设置 PWMA_EGR 寄存器的 COMG 位由软件产生，或在 TRGI 上升沿由硬件产生。

下图显示当发生 COM 事件时，三种不同配置下 OCx 和 OCxN 输出。

产生六步 PWM，使用 COM 的例子（OSSR=1）



18.5.8 使用刹车功能（PWMFLT）

刹车功能常用于马达控制中。当使用刹车功能时，依据相应的控制位（PWMA_BKR 寄存器中的 MOE、OSSI 和 OSSR 位），输出使能信号和无效电平都会被修改。

系统复位后，刹车电路被禁止，MOE 位为低。设置 PWMA_BKR 寄存器中的 BKE 位可以使能刹车功能。刹车输入信号的极性可以通过配置同一个寄存器中的 BKP 位选择。BKE 和 BKP 可以被同时修改。

MOE 下降沿相对于时钟模块可以是异步的，因此在实际信号（作用在输出端）和同步控制位（在 PWMA_BKR 寄存器中）之间设置了一个再同步电路。这个再同步电路会在异步信号和同步信号之间产生延迟。特别的，如果当它为低时写 MOE=1，则读出它之前必须先插入一个延时（空指令）才能读到正确的值。这是因为写入的是异步信号而读的是同步信号。

当发生刹车时（在刹车输入端出现选定的电平），有下述动作：

- MOE 位被异步地清除，将输出置于无效状态、空闲状态或者复位状态（由 OSSI 位选择）。这个特性在 MCU 的振荡器关闭时依然有效。
- 一旦 MOE=0，每一个输出通道输出由 PWMA_OISR 寄存器的 OISi 位设定的电平。如果 OSSI=0，则定时器不再控制输出使能信号，否则输出使能信号始终为高。
- 当使用互补输出时：
 - 输出首先被置于复位状态即无效的状态（取决于极性）。这是异步操作，即使定时器没有时钟时，此功能也有效。
 - 如果定时器的时钟依然存在，死区生成器将会重新生效，在死区之后根据 OISi 和 OISiN 位指示的电平驱动输出端口。即使在这种情况下，OCi 和 OCiN 也不能被同时驱动到有效的电平。注：因为重新同步 MOE，死区时间比通常情况下长一些（大约 2 个时钟周期）。
- 如果设置了 PWMA_IER 寄存器的 BIE 位，当刹车状态标志（PWMA_SR1 寄存器中的 BIF 位）

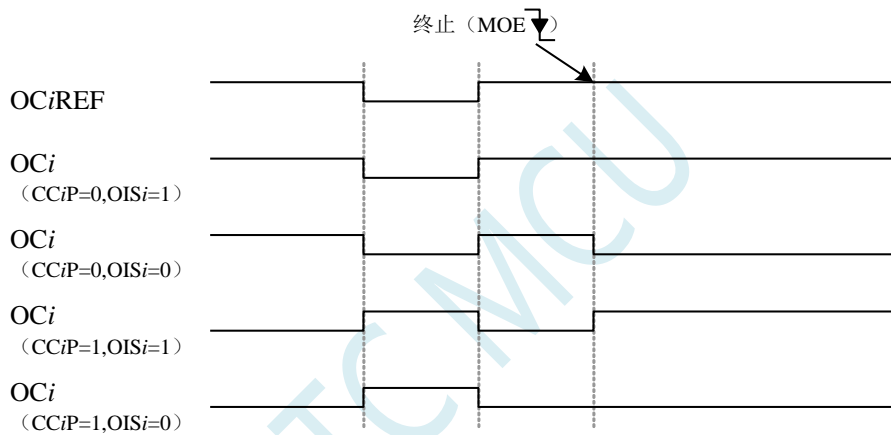
为'1'时, 则产生一个中断。

- 如果设置了 PWMA_BKR 寄存器中的 AOE 位, 在下一个更新事件 UEV 时 MOE 位被自动置位。例如这可以用来进行波形控制, 否则, MOE 始终保持低直到被再次置'1'。这个特性可以被用在安全方面, 你可以把刹车输入连到电源驱动的报警输出、热敏传感器或者其他安全器件上。

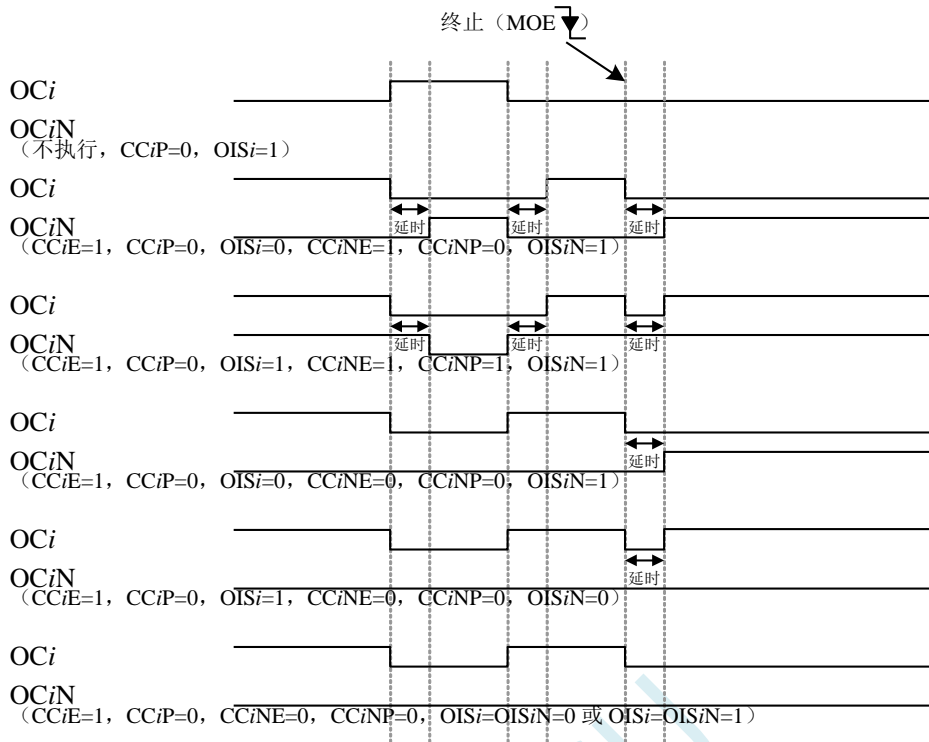
注: 刹车输入为电平有效。所以, 当刹车输入有效时, 不能同时 (自动地或者通过软件) 设置 MOE。同时, 状态标志 BIF 不能被清除。

刹车由 BRK 输入 (BKIN) 产生, 它的有效极性是可编程的, 且由 PWMA_BKR 寄存器的 BKE 位开启或禁止。除了刹车输入和输出管理, 刹车电路中还实现了写保护以保证应用程序的安全。它允许用户冻结几个配置参数 (OCi 极性和被禁止时的状态, OCiM 配置, 刹车使能和极性)。用户可以通过 PWMA_BKR 寄存器的 LOCK 位, 从三种级别的保护中选择一种。在 MCU 复位后 LOCK 位域只能被修改一次。

刹车响应的输出 (不带互补输出的通道)



带互补输出的刹车响应的输出 (PWMA 互补输出)



18.5.9 在外部事件发生时清除 OCiREF 信号

对于一个给定的通道, 在 ETRF 输入端 (设置 PWMA_CCMRi 寄存器中对应的 OCiCE 位为'1') 的高电平能够把 OCiREF 信号拉低, OCiREF 信号将保持为低直到发生下一次的更新事件 UEV。该功能只能用于输出比较模式和 PWM 模式, 而不能用于强制模式。

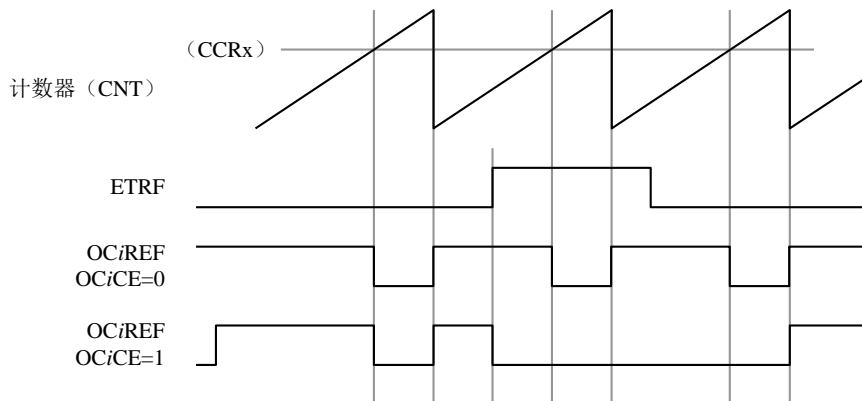
例如, OCiREF 信号可以联到一个比较器的输出, 用于控制电流。这时, ETR 必须配置如下:

1. 外部触发预分频器必须处于关闭: PWMA_ETR 寄存器中的 ETPS[1:0]=00。
2. 必须禁止外部时钟模式 2: PWMA_ETR 寄存器中的 ECE=0。
3. 外部触发极性 (ETP) 和外部触发滤波器 (ETF) 可以根据需要配置。

下图显示了当 ETRF 输入变为高时, 对应不同 OCiCE 的值, OCiREF 信号的动作。

在这个例子中, 定时器 PWMA 被置于 PWM 模式。

ETR 清除 PWMA 的 OCiREF



18.5.10 编码器接口模式

编码器接口模式一般用于马达控制。

选择编码器接口模式的方法是：

- 如果计数器只在 TI2 的边沿计数，则置 PWMA_SMCR 寄存器中的 SMS=001；
- 如果只在 TI1 边沿计数，则置 SMS=010；
- 如果计数器同时在 TI1 和 TI2 边沿计数，则置 SMS=011。

通过设置 PWMA_CCER1 寄存器中的 CC1P 和 CC2P 位，可以选择 TI1 和 TI2 极性；如果需要，还可以对输入滤波器编程。

两个输入 TI1 和 TI2 被用来作为增量编码器的接口。假定计数器已经启动（PWMA_CR1 寄存器中的 CEN=1），则计数器在每次 TI1FP1 或 TI2FP2 上产生有效跳变时计数。TI1FP1 和 TI2FP2 是 TI1 和 TI2 在通过输入滤波器和极性控制后的信号。如果没有滤波和极性变换，则 TI1FP1=TI1，TI2FP2=TI2。根据两个输入信号的跳变顺序，产生了计数脉冲和方向信号。依据两个输入信号的跳变顺序，计数器向上或向下计数，同时硬件对 PWMA_CR1 寄存器的 DIR 位进行相应的设置。不管计数器是依靠 TI1 计数、依靠 TI2 计数或者同时依靠 TI1 和 TI2 计数，在任一输入端（TI1 或者 TI2）的跳变都会重新计算 DIR 位。

编码器接口模式基本上相当于使用了一个带有方向选择的外部时钟。这意味着计数器只在 0 到 PWMA_ARR 寄存器的自动装载值之间连续计数（根据方向，或是 0 到 ARR 计数，或是 ARR 到 0 计数）。所以在开始计数之前必须配置 PWMA_ARR。在这种模式下捕获器、比较器、预分频器、重复计数器、触发输出特性等仍工作如常。编码器模式和外部时钟模式 2 不兼容，因此不能同时操作。

编码器接口模式下，计数器依照增量编码器的速度和方向被自动的修改，因此计数器的内容始终指示着编码器的位置，计数方向与相连的传感器旋转的方向对应。

下表列出了所有可能的组合（假设 TI1 和 TI2 不同时变换）。

计数方向与编码器信号的关系

有效边沿	相对信号的电平 (TI1FP1 对应 TI2, TI2FP2 对应 TI1)	TI1FP1 信号		TI2FP2 信号	
		上升	下降	上升	下降
仅在 TI1 计数	高	向下计数	向上计数	不计数	不计数

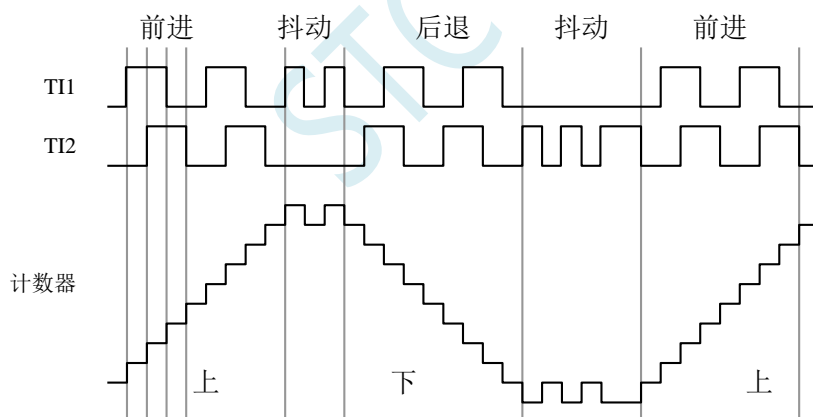
	低	向上计数	向下计数	不计数	不计数
仅在 TI2 计数	高	不计数	不计数	向上计数	向下计数
	低	不计数	不计数	向下计数	向上计数
在 TI1 和 TI2 上计数	高	向下计数	向上计数	向上计数	向下计数
	低	向上计数	向下计数	向下计数	向上计数

一个外部的增量编码器可以直接与 MCU 连接而不需要外部接口逻辑。但是，一般使用比较器将编码器的差分输出转换成数字信号，这大大增加了抗噪声干扰能力。编码器输出的第三个信号表示机械零点，可以把它连接到一个外部中断输入并触发一个计数器复位。

下面是一个计数器操作的实例，显示了计数信号的产生和方向控制。它还显示了当选择了双边沿时，输入抖动是如何被抑制的；抖动可能会在传感器的位置靠近一个转换点时产生。在这个例子中，我们假定配置如下：

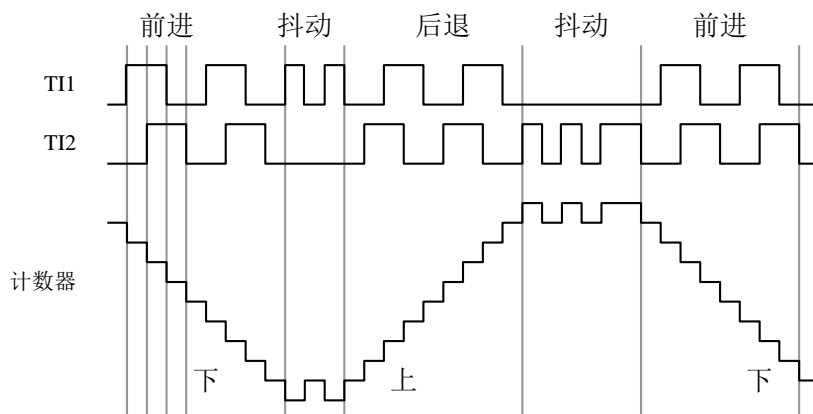
- CC1S=01 (PWMA_CCMR1 寄存器, IC1FP1 映射到 TI1)
- CC2S=01 (PWMA_CCMR2 寄存器, IC2FP2 映射到 TI2)
- CC1P=0 (PWMA_CCER1 寄存器, IC1 不反相, IC1=TI1)
- CC2P=0 (PWMA_CCER1 寄存器, IC2 不反相, IC2=TI2)
- SMS=011 (PWMA_SMCR 寄存器, 所有的输入均在上升沿和下降沿有效)。
- CEN=1 (PWMA_CR1 寄存器, 计数器使能)

编码器模式下的计数器操作实例



下图为当 IC1 极性反相时计数器的操作实例 (CC1P=1, 其他配置与上例相同)

IC1 反相的编码器接口模式实例



当定时器配置成编码器接口模式时，提供传感器当前位置的信息。使用另外一个配置在捕获模式下的定时器测量两个编码器事件的间隔，可以获得动态的信息（速度、加速度、减速度）。指示机械零点的编码器输出可被用做此目的。根据两个事件间的间隔，可以按照一定的时间间隔读出计数器。如果可能的话，你可以把计数器的值锁存到第三个输入捕获寄存器（捕获信号必须是周期的并且可以由另一个定时器产生）。

18.6 中断

PWMA/PWMB 各有 8 个中断请求源：

- 刹车中断
- 触发中断
- COM 事件中断
- 输入捕捉/输出比较 4 中断
- 输入捕捉/输出比较 3 中断
- 输入捕捉/输出比较 2 中断
- 输入捕捉/输出比较 1 中断
- 更新事件中断（如：计数器上溢，下溢及初始化）

为了使用中断特性，对每个被使用的中断通道，设置 PWMA_IER/PWMB_IER 寄存器中相应的中断使能位：即 BIE，TIE，COMIE，CCiIE，UIE 位。通过设置 PWMA_EGR/PWMB_EGR 寄存器中的相应位，也可以用软件产生上述各个中断源。

18.7 PWMA/PWMB 寄存器描述

18.7.1 功能脚切换 (PWMx_PS)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PS	7EFEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]	
PWMB_PS	7EFEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]	

C1PS[1:0]: 高级 PWM 通道 1 输出脚选择位

C1PS[1:0]	PWM1P	PWM1N
00	P1.0	P1.1
01	P2.0	P2.1
10	P6.0	P6.1
11	-	-

C2PS[1:0]: 高级 PWM 通道 2 输出脚选择位

C2PS[1:0]	PWM2P	PWM2N
00	P5.4	P1.3
01	P2.2	P2.3
10	P6.2	P6.3
11	-	-

C3PS[1:0]: 高级 PWM 通道 3 输出脚选择位

C3PS[1:0]	PWM3P	PWM3N
00	P1.4	P1.5
01	P2.4	P2.5
10	P6.4	P6.5
11	-	-

C4PS[1:0]: 高级 PWM 通道 4 输出脚选择位

C4PS[1:0]	PWM4P	PWM4N
00	P1.6	P1.7
01	P2.6	P2.7
10	P6.6	P6.7
11	P3.4	P3.3

C5PS[1:0]: 高级 PWM 通道 5 输出脚选择位

C5PS[1:0]	PWM5
00	P2.0
01	P1.7
10	P0.0
11	P7.4

C6PS[1:0]: 高级 PWM 通道 6 输出脚选择位

C6PS[1:0]	PWM6
00	P2.1
01	P5.4
10	P0.1
11	P7.5

C7PS[1:0]: 高级 PWM 通道 7 输出脚选择位

C7PS[1:0]	PWM7
00	P2.2
01	P3.3
10	P0.2
11	P7.6

C8PS[1:0]: 高级 PWM 通道 8 输出脚选择位

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

18.7.2 高级 PWM 功能脚选择寄存器 (PWM_x_ETRPS)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ETRPS	7EFEB0H						BRKAPS	ETRAPS[1:0]	
PWMB_ETRPS	7EFEB4H						BRKBPS	ETRBPS[1:0]	

ETRAPS[1:0]: 高级 PWMA 的外部触发脚 ERI 选择位

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

ETRBPS[1:0]: 高级 PWMB 的外部触发脚 ERIB 选择位

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

BRKAPS: 高级 PWMA 的刹车脚 PWMFLT 选择位

BRKAPS	PWMFLT
0	P3.5
1	比较器的输出

BRKBPS: 高级 PWMB 的刹车脚 PWMFLT2 选择位

BRKBPS	PWMFLT2
0	P3.5
1	比较器的输出

18.7.3 输出使能寄存器 (PWM_x_ENO)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ENO	7EFEB1H	ENO4N	ENO4P	ENO3N	ENO3P	ENO2N	ENO2P	ENO1N	ENO1P
PWMB_ENO	7EFEB5H	-	ENO8P	-	ENO7P	-	ENO6P	-	ENO5P

ENO8P: PWM8 输出控制位

0: 禁止 PWM8 输出

1: 使能 PWM8 输出

ENO7P: PWM7 输出控制位

0: 禁止 PWM7 输出

1: 使能 PWM7 输出

ENO6P: PWM6 输出控制位

0: 禁止 PWM6 输出

1: 使能 PWM6 输出

ENO5P: PWM5 输出控制位

0: 禁止 PWM5 输出

1: 使能 PWM5 输出

ENO4N: PWM4N 输出控制位

0: 禁止 PWM4N 输出

1: 使能 PWM4N 输出

ENO4P: PWM4P 输出控制位

0: 禁止 PWM4P 输出

1: 使能 PWM4P 输出

ENO3N: PWM3N 输出控制位

0: 禁止 PWM3N 输出

1: 使能 PWM3N 输出

ENO3P: PWM3P 输出控制位

0: 禁止 PWM3P 输出

1: 使能 PWM3P 输出

ENO2N: PWM2N 输出控制位

0: 禁止 PWM2N 输出

1: 使能 PWM2N 输出

ENO2P: PWM2P 输出控制位

0: 禁止 PWM2P 输出

1: 使能 PWM2P 输出

ENO1N: PWM1N 输出控制位

0: 禁止 PWM1N 输出

1: 使能 PWM1N 输出

ENO1P: PWM1P 输出控制位

0: 禁止 PWM1P 输出

1: 使能 PWM1P 输出

18.7.4 输出附加使能寄存器 (PWMx_IOAUX)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_IOAUX	7EFEB3H	AUX4N	AUX4P	AUX3N	AUX3P	AUX2N	AUX2P	AUX1N	AUX1P
PWMB_IOAUX	7EFEB7H	-	AUX8P	-	AUX7P	-	AUX6P	-	AUX5P

AUX8P: PWM8 输出附加控制位

0: PWM8 的输出直接由 ENO8P 控制

1: PWM8 的输出由 ENO8P 和 PWMB_BKR 共同控制

AUX7P: PWM7 输出附加控制位

- 0: PWM7 的输出直接由 ENO7P 控制
- 1: PWM7 的输出由 ENO7P 和 PWMB_BKR 共同控制
- AUX6P: PWM6 输出附加控制位
 - 0: PWM6 的输出直接由 ENO6P 控制
 - 1: PWM6 的输出由 ENO6P 和 PWMB_BKR 共同控制
- AUX5P: PWM5 输出附加控制位
 - 0: PWM5 的输出直接由 ENO5P 控制
 - 1: PWM5 的输出由 ENO5P 和 PWMB_BKR 共同控制
- AUX4N: PWM4N 输出附加控制位
 - 0: PWM4N 的输出直接由 ENO4N 控制
 - 1: PWM4N 的输出由 ENO4N 和 PWMA_BKR 共同控制
- AUX4P: PWM4P 输出附加控制位
 - 0: PWM4P 的输出直接由 ENO4P 控制
 - 1: PWM4P 的输出由 ENO4P 和 PWMA_BKR 共同控制
- AUX3N: PWM3N 输出附加控制位
 - 0: PWM3N 的输出直接由 ENO3N 控制
 - 1: PWM3N 的输出由 ENO3N 和 PWMA_BKR 共同控制
- AUX3P: PWM3P 输出附加控制位
 - 0: PWM3P 的输出直接由 ENO3P 控制
 - 1: PWM3P 的输出由 ENO3P 和 PWMA_BKR 共同控制
- AUX2N: PWM2N 输出附加控制位
 - 0: PWM2N 的输出直接由 ENO2N 控制
 - 1: PWM2N 的输出由 ENO2N 和 PWMA_BKR 共同控制
- AUX2P: PWM2P 输出附加控制位
 - 0: PWM2P 的输出直接由 ENO2P 控制
 - 1: PWM2P 的输出由 ENO2P 和 PWMA_BKR 共同控制
- AUX1N: PWM1N 输出附加控制位
 - 0: PWM1N 的输出直接由 ENO1N 控制
 - 1: PWM1N 的输出由 ENO1N 和 PWMA_BKR 共同控制
- AUX1P: PWM1P 输出附加控制位
 - 0: PWM1P 的输出直接由 ENO1P 控制
 - 1: PWM1P 的输出由 ENO1P 和 PWMA_BKR 共同控制

18.7.5 控制寄存器 1 (PWMx_CR1)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CR1	7EFEC0H	ARPEA	CMSA[1:0]		DIRA	OPMA	URSA	UDISA	CENA
PWMB_CR1	7EFEE0H	ARPEB	CMSB[1:0]		DIRB	OPMB	URSB	UDISB	CENB

- ARPE_n: 自动预装载允许位 (n=A,B)
- 0: PWM_n_ARR 寄存器没有缓冲, 它可以被直接写入
 - 1: PWM_n_ARR 寄存器由预装载缓冲器缓冲

CMS_n[1:0]: 选择对齐模式 (n= A,B)

CMS _n [1:0]	对齐模式	说明
------------------------	------	----

00	边沿对齐模式	计数器依据方向位（DIR）向上或向下计数
01	中央对齐模式1	计数器交替地向上和向下计数。 配置为输出的通道的输出比较中断标志位， 只在计数器向下计数时被置1。
10	中央对齐模式2	计数器交替地向上和向下计数。 配置为输出的通道的输出比较中断标志位， 只在计数器向上计数时被置1。
11	中央对齐模式3	计数器交替地向上和向下计数。 配置为输出的通道的输出比较中断标志位， 在计数器向上和向下计数时均被置1。

注 1：在计数器开启时（CEN=1），不允许从边沿对齐模式转换到中央对齐模式。

注 2：在中央对齐模式下，编码器模式（SMS=001, 010, 011）必须被禁止。

DIRn: 计数器的计数方向（n= A,B）

0: 计数器向上计数；

1: 计数器向下计数。

注：当计数器配置为中央对齐模式或编码器模式时，该位为只读。

OPMn: 单脉冲模式（n= A,B）

0: 在发生更新事件时，计数器不停止；

1: 在发生下一次更新事件时，清除 CEN 位，计数器停止。

URSn: 更新请求源（n= A,B）

0: 如果 UDIS 允许产生更新事件，则下述任一事件产生一个更新中断：

- 寄存器被更新（计数器上溢/下溢）
- 软件设置 UG 位
- 时钟/触发控制器产生的更新

1: 如果 UDIS 允许产生更新事件，则只有当下列事件发生时才产生更新中断，并 UIF 置 1：

- 寄存器被更新（计数器上溢/下溢）

UDISn: 禁止更新（n= A,B）

0: 一旦下列事件发生，产生更新（UEV）事件：

- 计数器溢出/下溢
- 产生软件更新事件
- 时钟/触发模式控制器产生的硬件复位 被缓存的寄存器被装入它们的预装载值。

1: 不产生更新事件，影子寄存器（ARR、PSC、CCR_x）保持它们的值。如果设置了 UG 位或时钟/触发控制器发出了一个硬件复位，则计数器和预分频器被重新初始化。

CENn: 允许计数器（n= A,B）

0: 禁止计数器；

1: 使能计数器。

注：在软件设置了 CEN 位后，外部时钟、门控模式和编码器模式才能工作。然而触发模式可以自动地通过硬件设置 CEN 位。

18.7.6 控制寄存器 2（PWM_x_CR2），及实时触发 ADC

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CR2	7EFEC1H	TI1S	MMSA[2:0]			-	COMSA	-	CCPCA

PWMB_CR2	7EFEE1H	TI5S	MMSB[2:0]	-	COMSB	-	CCPCB
----------	---------	------	-----------	---	-------	---	-------

TI1S: 第一组 PWM/PWMA 的 TI1 选择

0: PWM1P 输入管脚连到 TI1 (数字滤波器的输入);

1: PWM1P、PWM2P 和 PWM3P 管脚经异或后连到第一组 PWM 的 TI1。

TI5S: 第二组 PWM/PWMB 的 TI5 选择

0: PWM5 输入管脚连到 TI5 (数字滤波器的输入);

1: PWM5、PWM6 和 PWM7 管脚经异或后连到第二组 PWM 的 TI5。

MMSA[2:0]: 主模式选择

MMSA[2:0]	主模式	说明
000	复位	PWMA_EGR寄存器的UG位被用于作为触发输出 (TRGO)。如果触发输入 (时钟/触发控制器配置为复位模式) 产生复位, 则TRGO上的信号相对实际的复位会有一个延迟
001	使能	计数器使能信号被用于作为触发输出 (TRGO)。其用于启动ADC, 以便控制在一段时间内使能ADC。计数器使能信号是通过CEN控制位和门控模式下的触发输入信号的逻辑或产生。除非选择了主/从模式, 当计数器使能信号受控于触发输入时, TRGO上会有一个延迟。 注: 当需要使用PWM触发ADC转换时, 需要先设置ADC_CONTR寄存器中的ADC_POWER、ADC_CHS以及ADC_EPWMT, 当PWM产生TRGO内部信号时, 系统会自动设置ADC_START来启动AD转换。详细使用请参考范例程序“使用PWM的CEN启动PWMA定时器, 实时触发ADC”
010	更新	更新事件被选为触发输出 (TRGO)
011	比较脉冲	一旦发生一次捕获或一次比较成功, 当CC1IF标志被置1时, 触发输出送出一个正脉冲 (TRGO)
100	比较	OC1REF信号被用于作为触发输出 (TRGO)
101	比较	OC2REF信号被用于作为触发输出 (TRGO)
110	比较	OC3REF信号被用于作为触发输出 (TRGO)
111	比较	OC4REF信号被用于作为触发输出 (TRGO)

MMSB[2:0]: 主模式选择

MMSB[2:0]	主模式	说明
000	复位	PWMB_EGR寄存器的UG位被用于作为触发输出 (TRGO)。如果触发输入 (时钟/触发控制器配置为复位模式) 产生复位, 则TRGO上的信号相对实际的复位会有一个延迟
001	使能	计数器使能信号被用于作为触发输出 (TRGO)。其用于启动多个PWM, 以便控制在一段时间内使能从PWM。计数器使能信号是通过CEN控制位和门控模式下的触发输入信号的逻辑或产生。除非选择了主/从模式, 当计数器使能信号受控于触发输入时, TRGO上会有一个延迟。

010	更新	更新事件被选为触发输出 (TRGO)
011	比较脉冲	一旦发生一次捕获或一次比较成功, 当CC5IF标志被置1时, 触发输出送出一个正脉冲 (TRGO)
100	比较	OC5REF信号被用于作为触发输出 (TRGO)
101	比较	OC6REF信号被用于作为触发输出 (TRGO)
110	比较	OC7REF信号被用于作为触发输出 (TRGO)
111	比较	OC8REF信号被用于作为触发输出 (TRGO)

注: 只有第一组 PWM 的 TRGO 可用于触发启动 ADC

注: 只有第二组 PWM 的 TRGO 可用于第一组 PWM 的 ITR2

COMSn: 捕获/比较控制位的更新控制选择 (n=A,B)

0: 当 CCPCn=1 时, 只有在 COMG 位置 1 的时候这些控制位才被更新

1: 当 CCPCn=1 时, 只有在 COMG 位置 1 或 TRGI 发生上升沿的时候这些控制位才被更新

CCPCn: 捕获/比较预装载控制位 (n= A,B)

0: CCIE, CCINE, CCiP, CCiNP 和 OCIM 位不是预装载的

1: CCIE, CCINE, CCiP, CCiNP 和 OCIM 位是预装载的; 设置该位后, 它们只在设置了 COMG 位后被更新。

注: 该位只对具有互补输出的通道起作用。

18.7.7 从模式控制寄存器(PWMx_SMCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SMCR	7EFEC2H	MSMA		TSA[2:0]		-		SMSA[2:0]	
PWMB_SMCR	7EFEE2H	MSMB		TSB[2:0]		-		SMSB[2:0]	

MSMn: 主/从模式 (n= A,B)

0: 无作用

1: 触发输入 (TRGI) 上的事件被延迟了, 以允许 PWMn 与它的从 PWM 间的完美同步 (通过 TRGO)

TSA[2:0]: 触发源选择

TSA[2:0]	触发源
000	-
001	-
010	内部触发 ITR2
011	-
100	TI1的边沿检测器 (TI1F_ED)
101	滤波后的定时器输入1 (TI1FP1)
110	滤波后的定时器输入2 (TI2FP2)
111	外部触发输入 (ETRF)

TSB[2:0]: 触发源选择

TSB[2:0]	触发源
000	-
001	-
010	-

011	-
100	TI5的边沿检测器 (TI5F_ED)
101	滤波后的定时器输入1 (TI5FP5)
110	滤波后的定时器输入2 (TI5FP6)
111	外部触发输入 (ETRF)

注：这些位只能在 SMS=000 时被改变，以避免在改变时产生错误的边沿检测。

SMSA[2:0]：时钟/触发/从模式选择

SMSA[2:0]	功能	说明
000	内部时钟模式	如果CEN=1，则预分频器直接由内部时钟驱动
001	编码器模式1	根据TI1FP1的电平，计数器在TI2FP2的边沿向上/下计数
010	编码器模式2	根据TI2FP2的电平，计数器在TI1FP1的边沿向上/下计数
011	编码器模式3	根据另一个输入的电平，计数器在TI1FP1和TI2FP2的边沿向上/下计数
100	复位模式	在选中的触发输入 (TRGI) 的上升沿时重新初始化计数器，并且产生一个更新 寄存器的信号
101	门控模式	当触发输入 (TRGI) 为高时，计数器的时钟开启。一旦触发输入变为低，则计数器停止（但不复位）。计数器的启动和停止都是受控的
110	触发模式	计数器在触发输入TRGI的上升沿启动（但不复位），只有计数器的启动是受控的
111	外部时钟模式1	选中的触发输入 (TRGI) 的上升沿驱动计数器。 注：如果TI1F_ED被选为触发输入 (TS=100) 时，不要使用门控模式。这是因为TI1F_ED在每次TI1F变化时只是输出一个脉冲，然而门控模式是要检查触发输入的电平

SMSB[2:0]：时钟/触发/从模式选择

SMSB[2:0]	功能	说明
000	内部时钟模式	如果CEN=1，则预分频器直接由内部时钟驱动
001	编码器模式1	根据TI5FP5的电平，计数器在TI6FP6的边沿向上/下计数
010	编码器模式2	根据TI6FP6的电平，计数器在TI5FP5的边沿向上/下计数
011	编码器模式3	根据另一个输入的电平，计数器在TI5FP5和TI6FP6的边沿向上/下计数
100	复位模式	在选中的触发输入 (TRGI) 的上升沿时重新初始化计数器，并且产生一个更新 寄存器的信号
101	门控模式	当触发输入 (TRGI) 为高时，计数器的时钟开启。一旦触发输入变为低，则计数器停止（但不复位）。计数器的启动和停止都是受控的
110	触发模式	计数器在触发输入TRGI的上升沿启动（但不复位），只有计数器的启动是受控的

111	外部时钟模式1	选中的触发输入 (TRGI) 的上升沿驱动计数器。 注: 如果TI5F_ED被选为触发输入 (TS=100) 时, 不要使用门控模式。这是因为TI5F_ED在每次TI5F变化时只是输出一个脉冲, 然而门控模式是要检查触发输入的电平
-----	---------	--

18.7.8 外部触发寄存器(PWMx_ETR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ETR	7EFEC3H	ETP1	ECEA	ETPSA[1:0]		ETFA[3:0]			
PWMB_ETR	7EFEE3H	ETP2	ECEB	ETPSB[1:0]		ETFB[3:0]			

ETPn: 外部触发 ETR 的极性 (n= A,B)

0: 高电平或上升沿有效

1: 低电平或下降沿有效

ECEn: 外部时钟使能 (n= A,B)

0: 禁止外部时钟模式 2

1: 使能外部时钟模式 2, 计数器的时钟为 ETRF 的有效沿。

注 1: ECE 置 1 的效果与选择把 TRGI 连接到 ETRF 的外部时钟模式 1 相同 (PWMn_SMCR 寄存器中, SMS=111, TS=111)。

注 2: 外部时钟模式 2 可与下列模式同时使用: 触发标准模式; 触发复位模式; 触发门控模式。但是, 此时 TRGI 决不能与 ETRF 相连 (PWMn_SMCR 寄存器中, TS 不能为 111)。

注 3: 外部时钟模式 1 与外部时钟模式 2 同时使能, 外部时钟输入为 ETRF。

ETPSn: 外部触发预分频器外部触发信号 EPRP 的频率最大不能超过 fMASTER/4。可用预分频器来降低 ETRP 的频率, 当 EPRP 的频率很高时, 它非常有用: (n= A,B)

00: 预分频器关闭

01: EPRP 的频率/2

02: EPRP 的频率/4

03: EPRP 的频率/8

ETFn[3:0]: 外部触发滤波器选择, 该位域定义了 ETRP 的采样频率及数字滤波器长度。 (n= A,B)

ETFn[3:0]	时钟数	ETF[3:0]	时钟数
0000	1	1000	48
0001	2	1001	64
0010	4	1010	80
0011	8	1011	96
0100	12	1100	128
0101	16	1101	160
0110	24	1110	192
0111	32	1111	256

18.7.9 中断使能寄存器(PWMx_IER)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_IER	7EFEC4H	BIEA	TIEA	COMIEA	CC4IE	CC3IE	CC2IE	CC1IE	UIEA

PWMB_IER	7EFEE4H	BIEB	TIEB	COMIEB	CC8IE	CC7IE	CC6IE	CC5IE	UIEB
----------	---------	------	------	--------	-------	-------	-------	-------	------

BIEn: 允许刹车中断 (n= A,B)

0: 禁止刹车中断;

1: 允许刹车中断。

TIE: 触发中断使能 (n= A,B)

0: 禁止触发中断;

1: 使能触发中断。

COMIE: 允许 COM 中断 (n= A,B)

0: 禁止 COM 中断;

1: 允许 COM 中断。

CCnIE: 允许捕获/比较 n 中断 (n=1,2,3,4,5,6,7,8)

0: 禁止捕获/比较 n 中断;

1: 允许捕获/比较 n 中断。

UIEn: 允许更新中断 (n= A,B)

0: 禁止更新中断;

1: 允许更新中断。

18.7.10 状态寄存器 1(PWMx_SR1)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SR1	7EFEC5H	BIFA	TIFA	COMIFA	CC4IF	CC3IF	CC2IF	CC1IF	UIFA
PWMB_SR1	7EFEE5H	BIFB	TIFB	COMIFB	CC8IF	CC7IF	CC6IF	CC5IF	UIFB

BIFn: 刹车中断标记。一旦刹车输入有效, 由硬件对该位置 1。如果刹车输入无效, 则该位可由软件清

0。(n= A,B)

0: 无刹车事件产生

1: 刹车输入上检测到有效电平

TIFn: 触发器中断标记。当发生触发事件时由硬件对该位置 1。由软件清 0。(n= A,B)

0: 无触发器事件产生

1: 触发中断等待响应

COMIFn: COM 中断标记。一旦产生 COM 事件该位由硬件置 1。由软件清 0。(n= A,B)

0: 无 COM 事件产生

1: COM 中断等待响应

CC8IF: 捕获/比较8中断标记, 参考CC1IF描述

CC7IF: 捕获/比较7中断标记, 参考CC1IF描述

CC6IF: 捕获/比较6中断标记, 参考CC1IF描述

CC5IF: 捕获/比较5中断标记, 参考CC1IF描述

CC4IF: 捕获/比较4中断标记, 参考CC1IF描述

CC3IF: 捕获/比较3中断标记, 参考CC1IF描述

CC2IF: 捕获/比较2中断标记, 参考CC1IF描述

CC1IF: 捕获/比较1中断标记。

如果通道CC1配置为输出模式:

当计数器值与比较值匹配时该位由硬件置1, 但在中心对称模式下除外。它由软件清0。

0: 无匹配发生;

1: PWMA_CNT 的值与 PWMA_CCR1 的值匹配。

注: 在中心对称模式下, 当计数器值为 0 时, 向上计数, 当计数器值为 ARR 时, 向下计数 (它从

0 向上计数到 ARR-1，再由 ARR 向下计数到 1）。因此，对所有的 SMS 位值，这两个值都不置标记。但是，如果 CCR1>ARR，则当 CNT 达到 ARR 值时，CC1IF 置 1。

如果通道CC1配置为输入模式:

当捕获事件发生时该位由硬件置1，它由软件清0或通过读PWMA_CCR1L清0。

0: 无输入捕获产生

1: 计数器值已被捕获至 PWMA_CCR1

UIFn: 更新中断标记 当产生更新事件时该位由硬件置 1。它由软件清 0。（n= A,B）

0: 无更新事件产生

1: 更新事件等待响应。当寄存器被更新时该位由硬件置 1

– 若 PWMn_CR1 寄存器的 UDIS=0，当计数器上溢或下溢时

– 若 PWMn_CR1 寄存器的 UDIS=0、URS=0，当设置 PWMn_EGR 寄存器的 UG 位软件对计数器 CNT 重新初始化时

– 若 PWMn_CR1 寄存器的 UDIS=0、URS=0，当计数器 CNT 被触发事件重新初始化时

18.7.11 状态寄存器 2(PWMx_SR2)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SR2	7EFEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-
PWMB_SR2	7EFEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-

CC8OF: 捕获/比较8重复捕获标记。参见CC1OF描述。

CC7OF: 捕获/比较7重复捕获标记。参见CC1OF描述。

CC6OF: 捕获/比较6重复捕获标记。参见CC1OF描述。

CC5OF: 捕获/比较5重复捕获标记。参见CC1OF描述。

CC4OF: 捕获/比较4重复捕获标记。参见CC1OF描述。

CC3OF: 捕获/比较3重复捕获标记。参见CC1OF描述。

CC2OF: 捕获/比较2重复捕获标记。参见CC1OF描述。

CC1OF: 捕获/比较1重复捕获标记。仅当相应的通道被配置为输入捕获时，该标记可由硬件置1。写0可清除该位。

0: 无重复捕获产生;

1: 计数器的值被捕获到 PWMA_CCR1 寄存器时，CC1IF 的状态已经为 1。

18.7.12 事件产生寄存器 (PWMx_EGR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_EGR	7EFEC7H	BGA	TGA	COMGA	CC4G	CC3G	CC2G	CC1G	UGA
PWMB_EGR	7EFEE7H	BGB	TGB	COMGB	CC8G	CC7G	CC6G	CC5G	UGB

BGn: 产生刹车事件。该位由软件置 1，用于产生一个刹车事件，由硬件自动清 0（n= A,B）

0: 无动作

1: 产生一个刹车事件。此时 MOE=0、BIF=1，若开启对应的中断（BIE=1），则产生相应的中断

TGn: 产生触发事件。该位由软件置 1，用于产生一个触发事件，由硬件自动清 0（n= A,B）

0: 无动作

1: TIF=1，若开启对应的中断（TIE=1），则产生相应的中断

COMGn: 捕获/比较事件，产生控制更新。该位由软件置 1，由硬件自动清 0（n= A,B）

0: 无动作

1: CCPC=1，允许更新 CCIE、CCINE、CCiP、CCiNP、OCIM 位。

注：该位只对拥有互补输出的通道有效

CC8G：产生捕获/比较 8 事件。参考 CC1G 描述

CC7G：产生捕获/比较 7 事件。参考 CC1G 描述

CC6G：产生捕获/比较 6 事件。参考 CC1G 描述

CC5G：产生捕获/比较 5 事件。参考 CC1G 描述

CC4G：产生捕获/比较 4 事件。参考 CC1G 描述

CC3G：产生捕获/比较 3 事件。参考 CC1G 描述

CC2G：产生捕获/比较 2 事件。参考 CC1G 描述

CC1G：产生捕获/比较 1 事件。产生捕获/比较 1 事件。该位由软件置 1，用于产生一个捕获/比较事件，由硬件自动清 0。

0：无动作；

1：在通道 CC1 上产生一个捕获/比较事件。

若通道 CC1 配置为输出：设置 CC1IF=1，若开启对应的中断，则产生相应的中断。

若通道 CC1 配置为输入：当前的计数器值被捕获至 PWMA_CCR1 寄存器，设置 CC1IF=1，若开启对应的中断，则产生相应的中断。若 CC1IF 已经为 1，则设置 CC1OF=1。

UGn：产生更新事件 该位由软件置 1，由硬件自动清 0。（n=A,B）

0：无动作；

1：重新初始化计数器，并产生一个更新事件。

注意预分频器的计数器也被清 0（但是预分频系数不变）。若在中心对称模式下或 DIR=0（向上计数）则计数器被清 0；若 DIR=1（向下计数）则计数器取 PWMn_ARR 的值。

18.7.13 捕获/比较模式寄存器 1（PWMx_CCMR1）

通道可用于输入（捕获模式）或输出（比较模式），通道的方向由相应的 CCnS 位定义。该寄存器其它位的作用在输入和输出模式下不同。OCxx 描述了通道在输出模式下的功能，ICxx 描述了通道在输入模式下的功能。因此必须注意，同一个位在输出模式和输入模式下的功能是不同的。

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR1	7EFEC8H	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
PWMB_CCMR1	7EFEE8H	OC5CE	OC5M[2:0]			OC5PE	OC5FE	CC5S[1:0]	

OCnCE：输出比较 n 清零使能。该位用于使能使用 PWMETI 引脚上的外部事件来清通道 n 的输出信号（OCnREF）（n=1,5）

0：OCnREF 不受 ETRF 输入的影响；

1：一旦检测到 ETRF 输入高电平，OCnREF=0。

OCnM[2:0]：输出比较 n 模式。该 3 位定义了输出参考信号 OCnREF 的动作，而 OCnREF 决定了 OCn 的值。OCnREF 是高电平有效，而 OCn 的有效电平取决于 CCnP 位。（n=1,5）

OCnM[2:0]	模式	说明
000	冻结	PWMn_CCR1 与 PWMn_CNT 间的比较对 OCnREF 不起作用
001	匹配时设置通道 n 的输出为有效电平	当 PWMn_CCR1 = PWMn_CNT 时，OCnREF 输出高
010	匹配时设置通道 n 的输出为无效电平	当 PWMn_CCR1 = PWMn_CNT 时，OCnREF 输出低
011	翻转	当 PWMn_CCR1 = PWMn_CNT 时，翻转 OCnREF

100	强制为无效电平	强制OCnREF为低
101	强制为有效电平	强制OCnREF为高
110	PWM模式1	在向上计数时, 当PWMn_CNT<PWMn_CCR1时 OCnREF输出高, 否则OCnREF输出低 在向下计数时, 当PWMn_CNT>PWMn_CCR1时 OCnREF输出低, 否则OCnREF输出高
111	PWM模式2	在向上计数时, 当PWMn_CNT<PWMn_CCR1时 OCnREF输出低, 否则OCnREF输出高 在向下计数时, 当PWMn_CNT>PWMn_CCR1时 OCnREF输出高, 否则OCnREF输出低

注 1: 一旦 LOCK 级别设为 3 (PWMn_BKR 寄存器中的 LOCK 位) 并且 CCnS=00 (该通道配置成输出) 则该位不能被修改。

注 2: 在 PWM 模式 1 或 PWM 模式 2 中, 只有当比较结果改变了或在输出比较模式中从冻结模式切换到 PWM 模式时, OCnREF 电平才改变。

注 3: 在有互补输出的通道上, 这些位是预装载的。如果 PWMn_CR2 寄存器的 CCPC=1, OCM 位只有在 COM 事件发生时, 才从预装载位取新值。

OCnPE: 输出比较 n 预装载使能 (n=1,5)

0: 禁止 PWMn_CCR1 寄存器的预装载功能, 可随时写入 PWMn_CCR1 寄存器, 并且新写入的数值立即起作用。

1: 开启 PWMn_CCR1 寄存器的预装载功能, 读写操作仅对预装载寄存器操作, PWMn_CCR1 的预装载值在更新事件到来时被加载至当前寄存器中。

注 1: 一旦 LOCK 级别设为 3 (PWMn_BKR 寄存器中的 LOCK 位) 并且 CCnS=00 (该通道配置成输出) 则该位不能被修改。

注 2: 为了操作正确, 在 PWM 模式下必须使能预装载功能。但在单脉冲模式下 (PWMn_CR1 寄存器的 OPM=1), 它不是必须的。

OCnFE: 输出比较 n 快速使能。该位用于加快 CC 输出对触发输入事件的响应。(n=1,5)

0: 根据计数器与 CCRn 的值, CCn 正常操作, 即使触发器是打开的。当触发器的输入有一个有效沿时, 激活 CCn 输出的最小延时为 5 个时钟周期。

1: 输入到触发器的有效沿的作用就象发生了一次比较匹配。因此, OC 被设置为比较电平而与比较结果无关。采样触发器的有效沿和 CC1 输出间的延时被缩短为 3 个时钟周期。OCFE 只在通道被配置成 PWMA 或 PWMB 模式时起作用。

CC1S[1:0]: 捕获/比较 1 选择。这两位定义通道的方向 (输入/输出), 及输入脚的选择

CC1S[1:0]	方向	输入脚
00	输出	
01	输入	IC1映射在TI1FP1上
10	输入	IC1映射在TI2FP1上
11	输入	IC1映射在TRC上。此模式仅工作在内部触发器输入被选中时 (由PWMA_SMCR寄存器的TS位选择)

CC5S[1:0]: 捕获/比较 5 选择。这两位定义通道的方向 (输入/输出), 及输入脚的选择

CC5S[1:0]	方向	输入脚
00	输出	

01	输入	IC5映射在TI5FP5上
10	输入	IC5映射在TI6FP5上
11	输入	IC5映射在TRC上。此模式仅工作在内部触发器输入被选中时（由PWM5_SMCR寄存器的TS位选择）

注：CC1S 仅在通道关闭时（PWMA_CCER1 寄存器的 CC1E=0）才是可写的。

注：CC5S 仅在通道关闭时（PWM5_CCER1 寄存器的 CC5E=0）才是可写的。

通道配置为捕获输入模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR1	7EFEC8H	IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]	
PWMB_CCMR1	7EFEE8H	IC5F[3:0]				IC5PSC[1:0]		CC5S[1:0]	

ICnF[3:0]：输入捕获 n 滤波器选择，该位域定义了 TIn 的采样频率及数字滤波器长度。（n=1,5）

ICnF[3:0]	时钟数	ICnF[3:0]	时钟数
0000	1	1000	48
0001	2	1001	64
0010	4	1010	80
0011	8	1011	96
0100	12	1100	128
0101	16	1101	160
0110	24	1110	192
0111	32	1111	256

注：即使对于带互补输出的通道，该位域也是非预装载的，并且不会考虑 CCPC（PWMn_CR2 寄存器）的值

ICnPSC[1:0]：输入/捕获 n 预分频器。这两位定义了 CCn 输入（IC1）的预分频系数。（n=1,5）

00：无预分频器，捕获输入口上检测到的每一个边沿都触发一次捕获

01：每 2 个事件触发一次捕获

10：每 4 个事件触发一次捕获

11：每 8 个事件触发一次捕获

CC1S[1:0]：捕获/比较 1 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC1S[1:0]	方向	输入脚
00	输出	
01	输入	IC1映射在TI1FP1上
10	输入	IC1映射在TI2FP1上
11	输入	IC1映射在TRC上。此模式仅工作在内部触发器输入被选中时（由PWMA_SMCR寄存器的TS位选择）

CC5S[1:0]：捕获/比较 5 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC5S[1:0]	方向	输入脚
00	输出	
01	输入	IC5映射在TI5FP5上
10	输入	IC5映射在TI6FP5上

11	输入	IC5映射在TRC上。此模式仅工作在内部触发器输入被选中时（由PWM5_SMCN寄存器的TS位选择）
----	----	--

注：CC1S 仅在通道关闭时（PWMA_CCER1 寄存器的 CC1E=0）才是可写的。

注：CC5S 仅在通道关闭时（PWM5_CCER1 寄存器的 CC5E=0）才是可写的。

18.7.14 捕获/比较模式寄存器 2（PWMx_CCMR2）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR2	7EFEC9H	OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]	
PWMB_CCMR2	7EFEE9H	OC6CE	OC6M[2:0]			OC6PE	OC6FE	CC6S[1:0]	

OCnCE：输出比较 n 清零使能。该位用于使能使用 PWMETI 引脚上的外部事件来清通道 n 的输出信号（OCnREF）（n=2,6）

0：OCnREF 不受 ETRF 输入的影响；

1：一旦检测到 ETRF 输入高电平，OCnREF=0。

OCnM[2:0]：输出比较 2 模式，参考 OC1M。（n=2,6）

OCnPE：输出比较 2 预装载使能，参考 OP1PE。（n=2,6）

CC2S[1:0]：捕获/比较 2 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC2S[1:0]	方向	输入脚
00	输出	
01	输入	IC2映射在TI2FP2上
10	输入	IC2映射在TI1FP2上
11	输入	IC2映射在TRC上。

CC6S[1:0]：捕获/比较 6 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC6S[1:0]	方向	输入脚
00	输出	
01	输入	IC6映射在TI6FP6上
10	输入	IC6映射在TI5FP6上
11	输入	IC6映射在TRC上。

通道配置为捕获输入模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR2	7EFEC9H	IC2F[3:0]				IC2PSC[1:0]		CC2S[1:0]	
PWMB_CCMR2	7EFEE9H	IC6F[3:0]				IC6PSC[1:0]		CC6S[1:0]	

ICnF[3:0]：输入捕获 n 滤波器选择，参考 IC1F。（n=2,6）

ICnPSC[1:0]：输入/捕获 n 预分频器，参考 IC1PSC。（n=2,6）

CC2S[1:0]：捕获/比较 2 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC2S[1:0]	方向	输入脚
00	输出	
01	输入	IC2映射在TI2FP2上
10	输入	IC2映射在TI1FP2上

11	输入	IC2映射在TRC上。
----	----	-------------

CC6S[1:0]: 捕获/比较 6 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC6S[1:0]	方向	输入脚
00	输出	
01	输入	IC6映射在TI6FP6上
10	输入	IC6映射在TI5FP6上
11	输入	IC6映射在TRC上。

18.7.15 捕获/比较模式寄存器 3 (PWM_x_CCMR3)

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR3	7EFECAH	OC3CE	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]	
PWMB_CCMR3	7EFEEAH	OC7CE	OC7M[2:0]			OC7PE	OC7FE	CC7S[1:0]	

OCnCE: 输出比较 n 清零使能。该位用于使能使用 PWMETI 引脚上的外部事件来清通道 n 的输出信号 (OCnREF) (n=3,7)

0: OCnREF 不受 ETRF 输入的影响;

1: 一旦检测到 ETRF 输入高电平, OCnREF=0。

OCnM[2:0]: 输出比较 3 模式, 参考 OC1M。 (n=3,7)

OCnPE: 输出比较 3 预装载使能, 参考 OP1PE。 (n=3,7)

CC3S[1:0]: 捕获/比较 3 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC3S[1:0]	方向	输入脚
00	输出	
01	输入	IC3映射在TI3FP3上
10	输入	IC3映射在TI4FP3上
11	输入	IC3映射在TRC上。

CC7S[1:0]: 捕获/比较 7 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC7S[1:0]	方向	输入脚
00	输出	
01	输入	IC7映射在TI7FP7上
10	输入	IC7映射在TI8FP7上
11	输入	IC7映射在TRC上。

通道配置为捕获输入模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR3	7EFECAH	IC3F[3:0]				IC3PSC[1:0]		CC3S[1:0]	
PWMB_CCMR3	7EFEEAH	IC7F[3:0]				IC7PSC[1:0]		CC7S[1:0]	

ICnF[3:0]: 输入捕获 n 滤波器选择, 参考 IC1F。 (n=3,7)

ICnPSC[1:0]: 输入/捕获 n 预分频器, 参考 IC1PSC。 (n=3,7)

CC3S[1:0]: 捕获/比较 3 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC3S[1:0]	方向	输入脚
-----------	----	-----

00	输出	
01	输入	IC3映射在TI3FP3上
10	输入	IC3映射在TI4FP3上
11	输入	IC3映射在TRC上。

CC7S[1:0]: 捕获/比较 7 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC7S[1:0]	方向	输入脚
00	输出	
01	输入	IC7映射在TI7FP7上
10	输入	IC7映射在TI8FP7上
11	输入	IC7映射在TRC上。

18.7.16 捕获/比较模式寄存器 4 (PWMx_CCMR4)

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR4	7EFECBH	OC4CE	OC4M[2:0]			OC4PE	OC4FE	CC4S[1:0]	
PWMB_CCMR4	7EFEEBH	OC8CE	OC8M[2:0]			OC8PE	OC8FE	CC8S[1:0]	

OCnCE: 输出比较 n 清零使能。该位用于使能使用 PWMETI 引脚上的外部事件来清通道 n 的输出信号 (OCnREF) (n=4,8)

0: OCnREF 不受 ETRF 输入的影响;

1: 一旦检测到 ETRF 输入高电平, OCnREF=0。

OCnM[2:0]: 输出比较 n 模式, 参考 OC1M。 (n=4,8)

OCnPE: 输出比较 n 预装载使能, 参考 OP1PE。 (n=4,8)

CC4S[1:0]: 捕获/比较 4 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC4S[1:0]	方向	输入脚
00	输出	
01	输入	IC4映射在TI4FP4上
10	输入	IC4映射在TI3FP4上
11	输入	IC4映射在TRC上。

CC8S[1:0]: 捕获/比较 8 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC8S[1:0]	方向	输入脚
00	输出	
01	输入	IC8映射在TI8FP8上
10	输入	IC8映射在TI7FP8上
11	输入	IC8映射在TRC上。

通道配置为捕获输入模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR4	7EFECBH	IC4F[3:0]				IC4PSC[1:0]		CC4S[1:0]	
PWMB_CCMR4	7EFEEBH	IC8F[3:0]				IC8PSC[1:0]		CC8S[1:0]	

ICnF[3:0]: 输入捕获 n 滤波器选择, 参考 IC1F。 (n=4,8)

ICnPSC[1:0]: 输入/捕获 n 预分频器, 参考 IC1PSC。(n=4,8)

CC4S[1:0]: 捕获/比较 4 选择。这两位定义通道的方向(输入/输出), 及输入脚的选择

CC4S[1:0]	方向	输入脚
00	输出	
01	输入	IC4映射在TI4FP4上
10	输入	IC4映射在TI3FP4上
11	输入	IC4映射在TRC上。

CC8S[1:0]: 捕获/比较 8 选择。这两位定义通道的方向(输入/输出), 及输入脚的选择

CC8S[1:0]	方向	输入脚
00	输出	
01	输入	IC8映射在TI8FP8上
10	输入	IC8映射在TI7FP8上
11	输入	IC8映射在TRC上。

18.7.17 捕获/比较使能寄存器 1 (PWMx_CCER1)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCER1	7EFECCH	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E
PWMB_CCER1	7EFECH	-	-	CC6P	CC6E	-	-	CC5P	CC5E

CC6P: OC6 输入捕获/比较输出极性。参考 CC1P

CC6E: OC6 输入捕获/比较输出使能。参考 CC1E

CC5P: OC5 输入捕获/比较输出极性。参考 CC1P

CC5E: OC5 输入捕获/比较输出使能。参考 CC1E

CC2NP: OC2N 比较输出极性。参考 CC1NP

CC2NE: OC2N 比较输出使能。参考 CC1NE

CC2P: OC2 输入捕获/比较输出极性。参考 CC1P

CC2E: OC2 输入捕获/比较输出使能。参考 CC1E

CC1NP: OC1N 比较输出极性

0: 高电平有效;

1: 低电平有效。

注 1: 一旦 LOCK 级别 (PWMA_BKR 寄存器中的 LOCK 位) 设为 3 或 2 且 CC1S=00 (通道配置为输出), 则该位不能被修改。

注 2: 对于有互补输出的通道, 该位是预装载的。如果 CCPC=1 (PWMA_CR2 寄存器), 只有在 COM 事件发生时, CC1NP 位才从预装载位中取新值。

CC1NE: OC1N 比较输出使能

0: 关闭比较输出。

1: 开启比较输出, 其输出电平依赖于 MOE、OSSI、OSSR、OIS1、OIS1N 和 CC1E 位的值。

注: 对于有互补输出的通道, 该位是预装载的。如果 CCPC=1 (PWMA_CR2 寄存器), 只有在 COM 事件发生时, CC1NE 位才从预装载位中取新值。

CC1P: OC1 输入捕获/比较输出极性

CC1 通道配置为输出:

0: 高电平有效

1: 低电平有效

CC1 通道配置为输入或者捕获:

0: 捕获发生在 TI1F 或 TI2F 的上升沿;

1: 捕获发生在 TI1F 或 TI2F 的下降沿。

CC1E: OC1 输入捕获/比较输出使能

0: 关闭输入捕获/比较输出;

1: 开启输入捕获/比较输出。

注 1: 一旦 LOCK 级别 (PWMA_BKR 寄存器中的 LOCK 位) 设为 3 或 2, 则该位不能被修改。

注 2: 对于有互补输出的通道, 该位是预装载的。如果 CCPC=1 (PWMA_CR2 寄存器), 只有在 COM 事件发生时, CC1P 位才从预装载位中取新值。

带刹车功能的互补输出通道 OC_i 和 OC_{iN} 的控制位

控制位					输出状态	
MOE	OSSI	OSSR	CC _i E	CC _i NE	OC _i 输出状态	OC _{iN} 输出状态
1	X	0	0	0	输出禁止	输出禁止
		0	0	1	输出禁止	带极性的 OC _i REF
		0	1	0	带极性的 OC _i REF	输出禁止
		0	1	1	带极性和死区的 OC _i REF	带极性和死区的反向 OC _i REF
		1	0	0	输出禁止	输出禁止
		1	0	1	关闭状态 (输出使能且为无效电平) OC _i =CC _i P	带极性的 OC _i REF
		1	1	0	带极性的 OC _i REF	关闭状态 (输出使能且为无效电平) OC _{iN} =CC _{iN} P
		1	1	1	带极性和死区的 OC _i REF	带极性和死区的反向 OC _i REF
0	0	X	X	X	输出禁止	
	1				关闭状态 (输出使能且为无效电平) 异步地: OC _i =CC _i P, OC _{iN} =CC _{iN} P; 然后, 若时钟存在: 经过一个死区时间后 OC _i =OIS _i , OC _{iN} =OIS _{iN} , 假设 OIS _i 与 OIS _{iN} 并不都对应 OC _i 和 OC _{iN} 的有效电平。	

注: 管脚连接到互补的 OC_i 和 OC_{iN} 通道的外部 I/O 管脚的状态, 取决于 OC_i 和 OC_{iN} 通道状态和 GPIO 寄存器。

18.7.18 捕获/比较使能寄存器 2 (PWM_x_CCER2)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCER2	7EFECDH	CC4NP	CC4NE	CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E
PWMB_CCER2	7EFEEDH	-	-	CC8P	CC8E	-	-	CC7P	CC7E

CC8P: OC8 输入捕获/比较输出极性。参考 CC1P

CC8E: OC8 输入捕获/比较输出使能。参考 CC1E

CC7P: OC7 输入捕获/比较输出极性。参考 CC1P

CC7E: OC7 输入捕获/比较输出使能。参考 CC1E

CC4NP: OC4N 比较输出极性。参考 CC1NP

CC4NE: OC4N 比较输出使能。参考 CC1NE

CC4P: OC4 输入捕获/比较输出极性。参考 CC1P

CC4E: OC4 输入捕获/比较输出使能。参考 CC1E

CC3NP: OC3N 比较输出极性。参考 CC1NP

CC3NE: OC3N 比较输出使能。参考 CC1NE

CC3P: OC3 输入捕获/比较输出极性。参考 CC1P

CC3E: OC3 输入捕获/比较输出使能。参考 CC1E

18.7.19 计数器高 8 位 (PWM_x_CNTRH)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CNTRH	7EFECEH	CNT1[15:8]							
PWMB_CNTRH	7EFEFEH	CNT2[15:8]							

CNTn[15:8]: 计数器的高 8 位值 (n= A,B)

18.7.20 计数器低 8 位 (PWM_x_CNTRL)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CNTRL	7EFECFH	CNT1[7:0]							
PWMB_CNTRL	7EFEFFH	CNT2[7:0]							

CNTn[7:0]: 计数器的低 8 位值 (n= A,B)

18.7.21 预分频器高 8 位 (PWM_x_PSCRH), 输出频率计算公式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PSCRH	7EFED0H	PSC1[15:8]							
PWMB_PSCRH	7EFEF0H	PSC2[15:8]							

PSCn[15:8]: 预分频器的高 8 位值。(n= A,B)

预分频器用于对 CK_PSC 进行分频。计数器的时钟频率(fCK_CNT)等于 fCK_PSC/(PSCR[15:0]+1)。PSCR 包含了当更新事件产生时装入当前预分频器寄存器的值(更新事件包括计数器被 TIM_EGR 的 UG 位清 0 或被工作在复位模式的从控制器清 0)。这意味着为了使新的值起作用, 必须产生一个更新事件。

PWM 输出频率计算公式

PWMA 和 PWMB 两组 PWM 的输出频率计算公式相同, 且每组可设置不同的频率。

对齐模式	PWM输出频率计算公式
边沿对齐	$\text{PWM输出频率} = \frac{\text{系统工作频率SYSclk}}{(\text{PWMx_PSCR} + 1) \times (\text{PWMx_AAR} + 1)}$
中间对齐	$\text{PWM输出频率} = \frac{\text{系统工作频率SYSclk}}{(\text{PWMx_PSCR} + 1) \times \text{PWMx_AAR} \times 2}$

18.7.22 预分频器低 8 位 (PWM_x_PSCRL)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
----	----	----	----	----	----	----	----	----	----

PWMA_PSCRL	7EFED1H	PSC1[7:0]
PWMB_PSCRL	7EFEF1H	PSC2[7:0]

PSCn[7:0]: 预分频器的低 8 位值。(n= A,B)

18.7.23 自动重装载寄存器高 8 位 (PWMx_ARRH)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ARRH	7EFED2H	ARR1[15:8]							
PWMB_ARRH	7EFEF2H	ARR2[15:8]							

ARRn[15:8]: 自动重装载高 8 位值 (n= A,B)

ARR 包含了将要装载入实际的自动重装载寄存器的值。当自动重装载的值为 0 时, 计数器不工作。

18.7.24 自动重装载寄存器低 8 位 (PWMx_ARRL)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ARRL	7EFED3H	ARR1[7:0]							
PWMB_ARRL	7EFEF3H	ARR2[7:0]							

ARRn[7:0]: 自动重装载低 8 位值 (n= A,B)

18.7.25 重复计数器寄存器 (PWMx_RCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_RCR	7EFED4H	REP1[7:0]							
PWMB_RCR	7EFEF4H	REP2[7:0]							

REPn[7:0]: 重复计数器值 (n= A,B)

开启了预装载功能后, 这些位允许用户设置比较寄存器的更新速率(即周期性地从预装载寄存器 传输到当前寄存器); 如果允许产生更新中断, 则会同时影响产生更新中断的速率。每次向下计数器 REP_CNT 达到 0, 会产生一个更新事件并且计数器 REP_CNT 重新从 REP 值开始计数。由于 REP_CNT 只有在周期更新事件 U_RC 发生时才重载 REP 值, 因此对 PWMn_RCR 寄存器写入的新值只在下次周期更新事件发生时才起作用。这意味着在 PWM 模式中, (REP+1) 对应着:

- 在边沿对齐模式下, PWM 周期的数目;
- 在中心对称模式下, PWM 半周期的数目。

18.7.26 捕获/比较寄存器 1/5 高 8 位 (PWMx_CCR1H)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR1H	7EFED5H	CCR1[15:8]							
PWMB_CCR5H	7EFEF5H	CCR5[15:8]							

CCRn[15:8]: 捕获/比较 n 的高 8 位值 (n=1,5)

若 CCn 通道配置为输出: CCRn 包含了装入当前比较值(预装载值)。如果在 PWMn_CCMR1 寄存器(OCnPE 位)中未选择预装载功能, 写入的数值会立即传输至当前寄存器中。否则只有当更新事件发生时, 此预装载值才传输至当前捕获/比较 n 寄存器中。当前比较值同计数器 PWMn_CNT 的值相比较, 并在 OCn 端口上产生输出信号。

若 CCn 通道配置为输入: CCRn 包含了上一次输入捕获事件发生时的计数器值(此时该寄存器为

只读)。

18.7.27 捕获/比较寄存器 1/5 低 8 位 (PWM_x_CCR1L)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR1L	7EFED6H	CCR1[7:0]							
PWMB_CCR5L	7EFEF6H	CCR5[7:0]							

CCRn[7:0]: 捕获/比较 n 的低 8 位值 (n=1,5)

18.7.28 捕获/比较寄存器 2/6 高 8 位 (PWM_x_CCR2H)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR2H	7EFED7H	CCR2[15:8]							
PWMB_CCR6H	7EFEF7H	CCR6[15:8]							

CCRn[15:8]: 捕获/比较 n 的高 8 位值 (n=2,6)

18.7.29 捕获/比较寄存器 2/6 低 8 位 (PWM_x_CCR2L)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR2L	7EFED8H	CCR2[7:0]							
PWMB_CCR6L	7EFEF8H	CCR6[7:0]							

CCRn[7:0]: 捕获/比较 n 的低 8 位值 (n=2,6)

18.7.30 捕获/比较寄存器 3/7 高 8 位 (PWM_x_CCR3H)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR3H	7EFED9H	CCR3[15:8]							
PWMB_CCR7H	7EFEF9H	CCR7[15:8]							

CCRn[15:8]: 捕获/比较 n 的高 8 位值 (n=3,7)

18.7.31 捕获/比较寄存器 3/7 低 8 位 (PWM_x_CCR3L)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR3L	7EFEDA H	CCR3[7:0]							
PWMB_CCR7L	7EFEFA H	CCR7[7:0]							

CCRn[7:0]: 捕获/比较 n 的低 8 位值 (n=3,7)

18.7.32 捕获/比较寄存器 4/8 高 8 位 (PWM_x_CCR4H)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR4H	7EFEDBH	CCR4[15:8]							
PWMB_CCR8H	7EFEFBH	CCR8[15:8]							

CCRn[15:8]: 捕获/比较 n 的高 8 位值 (n=4,8)

18.7.33 捕获/比较寄存器 4/8 低 8 位 (PWM_x_CCR4L)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR4L	7EFEDCH	CCR4[7:0]							
PWMB_CCR8L	7EFEFCH	CCR8[7:0]							

CCRN[7:0]: 捕获/比较 n 的低 8 位值 (n=4,8)

18.7.34 刹车寄存器 (PWM_x_BKR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_BRK	7EFEDDH	MOEA	AOEA	BKPA	BKEA	OSSRA	OSSIA	LOCKA[1:0]	
PWMB_BRK	7EFEFDH	MOEB	AOEB	BKPB	BKEB	OSSRB	OSSIB	LOCKB[1:0]	

MOEn: 主输出使能。一旦刹车输入有效, 该位被硬件异步清 0。根据 AOE 位的设置值, 该位可以由软件置 1 或被自动置 1。它仅对配置为输出的通道有效。(n=A,B)

0: 禁止 OC 和 OCN 输出或强制为空闲状态

1: 如果设置了相应的使能位 (PWM_n_CCERX 寄存器的 CCIE 位), 则使能 OC 和 OCN 输出。

AOEn: 自动输出使能 (n=A,B)

0: MOE 只能被软件置 1;

1: MOE 能被软件置 1 或在下一个更新事件被自动置 1 (如果刹车输入无效)。

注: 一旦 LOCK 级别 (PWM_n_BKR 寄存器中的 LOCK 位) 设为 1, 则该位不能被修改

BKPn: 刹车输入极性 (n=A,B)

0: 刹车输入低电平有效

1: 刹车输入高电平有效

注: 一旦 LOCK 级别 (PWM_n_BKR 寄存器中的 LOCK 位) 设为 1, 则该位不能被修改

BKEn: 刹车功能使能 (n=A,B)

0: 禁止刹车输入 (BRK)

1: 开启刹车输入 (BRK)

注: 一旦 LOCK 级别 (PWM_n_BKR 寄存器中的 LOCK 位) 设为 1, 则该位不能被修改。

OSSRn: 运行模式下“关闭状态”选择。该位在 MOE=1 且通道设为输出时有效 (n=A,B)

0: 当 PWM 不工作时, 禁止 OC/OCN 输出 (OC/OCN 使能输出信号=0);

1: 当 PWM 不工作时, 一旦 CCiE=1 或 CCiNE=1, 首先开启 OC/OCN 并输出无效电平, 然后置 OC/OCN 使能输出信号=1。

注: 一旦 LOCK 级别 (PWM_n_BKR 寄存器中的 LOCK 位) 设为 2, 则该位不能被修改。

OSSIn: 空闲模式下“关闭状态”选择。该位在 MOE=0 且通道设为输出时有效。(n=A,B)

0: 当 PWM 不工作时, 禁止 OC/OCN 输出 (OC/OCN 使能输出信号=0);

1: 当 PWM 不工作时, 一旦 CCiE=1 或 CCiNE=1, OC/OCN 首先输出其空闲电平, 然后 OC/OCN 使能输出信号=1。

注: 一旦 LOCK 级别 (PWM_n_BKR 寄存器中的 LOCK 位) 设为 2, 则该位不能被修改。

LOCKn[1:0]: 锁定设置。该位为防止软件错误而提供的写保护措施 (n=A,B)

LOCKn[1:0]	保护级别	保护内容
00	无保护	寄存器无写保护
01	锁定级别1	不能写入PWM _n _BKR寄存器的BKE、BKP、AOE位和 PWM _n _OISR寄存器的OISI位
10	锁定级别2	不能写入锁定级别1中的各位,

		也不能写入CC极性位以及OSSR/OSSI位
11	锁定级别3	不能写入锁定级别2中的各位， 也不能写入CC控制位

注：由于 BKE、BKP、AOE、OSSR、OSSI 位可被锁定（依赖于 LOCK 位），因此在第一次写 PWMn_BKR 寄存器时必须对它们进行设置。

18.7.35 死区寄存器（PWMx_DTR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_DTR	7EFEDEH	DTGA[7:0]							
PWMB_DTR	7EFEFEH	DTGB[7:0]							

DTGn[7:0]：死区发生器设置。（n= A,B）

这些位定义了插入互补输出之间的死区持续时间。（ t_{CK_PSC} 为 PWMn 的时钟脉冲）

DTGn[7:5]	死区时间
000	$DTGn[7:0] * t_{CK_PSC}$
001	
010	
011	
100	
101	$(64 + DTGn[6:0]) * 2 * t_{CK_PSC}$
110	$(32 + DTGn[5:0]) * 8 * t_{CK_PSC}$
111	$(32 + DTGn[4:0]) * 16 * t_{CK_PSC}$

18.7.36 输出空闲状态寄存器（PWMx_OISR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_OISR	7EFEDFH	OIS4N	OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1
PWMB_OISR	7EFEFFH	-	OIS8	-	OIS7	-	OIS6	-	OIS5

OIS8：空闲状态时 OC8 输出电平

OIS7：空闲状态时 OC7 输出电平

OIS6：空闲状态时 OC6 输出电平

OIS5：空闲状态时 OC5 输出电平

OIS4N：空闲状态时 OC4N 输出电平

OIS4：空闲状态时 OC4 输出电平

OIS3N：空闲状态时 OC3N 输出电平

OIS3：空闲状态时 OC3 输出电平

OIS2N：空闲状态时 OC2N 输出电平

OIS2：空闲状态时 OC2 输出电平

OIS1N：空闲状态时 OC1N 输出电平

0：当 MOE=0 时，则在一个死区时间后，OC1N=0；

1：当 MOE=0 时，则在一个死区时间后，OC1N=1。

OIS1：空闲状态时 OC1 输出电平

- 0: 当 MOE=0 时, 如果 OC1N 使能, 则在一个死区后, OC1=0;
- 1: 当 MOE=0 时, 如果 OC1N 使能, 则在一个死区后, OC1=1。

STC MCU

18.8 范例程序

18.8.1 六步 PWM 驱动无刷直流马达(带 HALL)

C 语言代码

```
//测试工作频率为11.0592MHz
```

```
#include "stc16.h"          //头文件见附录
```

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

```
typedef unsigned char u8;
```

```
typedef unsigned int u16;
```

```
#define TRUE 1
```

```
#define FALSE 0
```

```
#define RV09_CH 6
```

```
#define PWMA_Period ((u16)0x0180)
```

```
#define PWMA_STPulse ((u16)342)
```

```
#define START 0x1A
```

```
#define RUN 0x1B
```

```
#define STOP 0x1C
```

```
#define IDLE 0x1D
```

```
#define PWMA_OCMODE_MASK ((u8)0x70)
```

```
#define PWMA_OCCE_ENABLE ((u8)0x80)
```

```
#define PWMA_OCCE_DISABLE ((u8)0x00)
```

```
#define PWMA_OCMODE_TIMING ((u8)0x00)
```

```
#define PWMA_OCMODE_ACTIVE ((u8)0x10)
```

```
#define PWMA_OCMODE_INACTIVE ((u8)0x20)
```

```
#define PWMA_OCMODE_TOGGLE ((u8)0x30)
```

```
#define PWMA_FORCE_INACTIVE ((u8)0x40)
```

```
#define PWMA_FORCE_ACTIVE ((u8)0x50)
```

```
#define PWMA_OCMODE_PWM1 ((u8)0x60)
```

```
#define PWMA_OCMODE_PWM2 ((u8)0x70)
```

```
#define CC1_POLARITY_HIGH ((u8)0x02)
```

```
#define CC1N_POLARITY_HIGH ((u8)0x08)
```

```
#define CC2_POLARITY_HIGH ((u8)0x20)
```

```
#define CC2N_POLARITY_HIGH ((u8)0x80)
```

```
#define CC1_POLARITY_LOW ((u8)~0x02)
```

```
#define CC1N_POLARITY_LOW ((u8)~0x08)
```

```
#define CC2_POLARITY_LOW ((u8)~0x20)
```

```
#define CC2N_POLARITY_LOW ((u8)~0x80)
```

```
#define CC1_OCENABLE ((u8)0x01)
```

```
#define CC1N_OCENABLE ((u8)0x04)
```

```
#define CC2_OCENABLE ((u8)0x10)
```

```
#define CC2N_OCENABLE ((u8)0x40)
```

```
#define CC1_OCDISABLE ((u8)~0x01)
```

```
#define CC1N_OCDISABLE ((u8)~0x04)
```

```
#define CC2_OCDISABLE ((u8)~0x10)
```

```
#define CC2N_OCDISABLE ((u8)~0x40)
```

```
#define CC3_POLARITY_HIGH ((u8)0x02)
```

```
#define CC3N_POLARITY_HIGH ((u8)0x08)
```

```
#define CC4_POLARITY_HIGH ((u8)0x20)
```

```

#define CC4N_POLARITY_HIGH      ((u8)0x80)
#define CC3_POLARITY_LOW        ((u8)~0x02)
#define CC3N_POLARITY_LOW       ((u8)~0x08)
#define CC4_POLARITY_LOW        ((u8)~0x20)
#define CC4N_POLARITY_LOW       ((u8)~0x80)
#define CC3_OCENABLE            ((u8)0x01)
#define CC3N_OCENABLE           ((u8)0x04)
#define CC4_OCENABLE            ((u8)0x10)
#define CC4N_OCENABLE           ((u8)0x40)
#define CC3_OCDISABLE           ((u8)~0x01)
#define CC3N_OCDISABLE          ((u8)~0x04)
#define CC4_OCDISABLE           ((u8)~0x10)
#define CC4N_OCDISABLE          ((u8)~0x40)

```

```
void LED_OUT(u8 X);
```

```
//LED 单字节串行移位函数
```

```

unsigned char code LED_0F[] =
{
    0xC0,0xF9,0xA4,0xB0,
    0x99,0x92,0x82,0xF8,
    0x80,0x90,0x8C,0xBF,
    0xC6,0xA1,0x86,0xFF,
    0xBF
};

```

```

#define DIO          P23
#define RCLK         P24
#define SCLK         P25

```

```
//串行数据输入
```

```
//时钟脉冲信号——上升沿有效
```

```
//打入信号——上升沿有效
```

```

void DelayXus(unsigned char delayTime);
void DelayXms(unsigned char delayTime);
unsigned int ADC_Convert(u8 ch);
void PWM_Init(void);
void SPEED_ADJ();
unsigned char RD_HALL();
void MOTOR_START();
void MOTOR_STOP();
unsigned char KEY_detect();
void LED4_Display(unsigned int dat,unsigned char num);

```

```

unsigned char Display_num=1;
unsigned int Display_dat=0;
unsigned int Motor_speed;
unsigned char Motor_sta = IDLE;
unsigned char BRK_occur=0;
unsigned int PWMB_CAPI_v=0;
unsigned int CAPI_avg=0;
unsigned char CAPI_cnt=0;
unsigned long CAPI_sum=0;

```

```

void main(void)
{

```

```
    P_SW2 = 0x80;
```

```
    PI = 0x00;
```

```
    P0M1 = 0x0C;
```

```
    P0M0 = 0x01;
```

```
    P1M1 = 0xc0;
```

```
    P1M0 = 0x3F;
```

```

P2M1 = 0x00;
P2M0 = 0x38;
P3M1 = 0x28;
P3M0 = 0x00;

ET0=1;
TR0=1;

ADCCFG = 0x0f;
ADC_CONTR = 0x80;

PWMA_ENO = 0x3F;           //PWMA 输出使能
PWMB_ENO = 0x00;           //PWMB 输出使能
PWMA_PS = 0x00;            //PWMA pin 选择
PWMB_PS = 0xd5;            //PWMB pin 选择

/*****
输出比较模式 PWMx_duty = [CCRx/(ARR + 1)]*100
*****/
/*****PWMB 接 hall 传感器*****/
//////////时基单元//////////
PWMB_PSCRL = 15;
PWMB_ARRH = 0xff;          //自动重载寄存器, 计数器 overflow 点
PWMB_ARRL = 0xff;
PWMB_CCR8H = 0x00;
PWMB_CCR8L = 0x05;

//////////通道配置//////////
PWMB_CCMR1 = 0x43;         //通道模式配置
PWMB_CCMR2 = 0x41;
PWMB_CCMR3 = 0x41;
PWMB_CCMR4 = 0x70;
PWMB_CCER1 = 0x11;
PWMB_CCER2 = 0x11;

//////////模式配置//////////
PWMB_CR2 = 0xf0;
PWMB_CR1 = 0x81;
PWMB_SMCR = 0x44;

//////////使能& 中断配置//////////
PWMB_BKR = 0x80;          //主输出使能
PWMB_IER = 0x02;          //使能中断

/*****PWMA 控制马达换相*****/
//////////时基单元//////////
PWMA_PSCRH = 0x00;        //预分频寄存器
PWMA_PSCRL = 0x00;
PWMA_ARRH = (u8)(PWMA_Period >> 8);
PWMA_ARRL = (u8)(PWMA_Period);

//////////通道配置//////////
PWMA_CCMR1 = 0x70;        //通道模式配置
PWMA_CCMR2 = 0x70;
PWMA_CCMR3 = 0x70;
PWMA_CCER1 = 0x11;        //配置通道输出使能和极性
PWMA_CCER2 = 0x01;        //配置通道输出使能和极性
PWMA_OISR = 0xAA;        //配置 MOE=0 时各通道输出电平

```

//////////模式配置//////////

```
PWMA_CR1 = 0xA0;
PWMA_CR2 = 0x24;
PWMA_SMCR = 0x20;
```

//////////使能& 中断配置//////////

```
PWMA_BKR = 0x1c;
PWMA_CR1 |= 0x01; //使能计数器
```

```
EA = 1;
while (1)
```

```
{
    P22=~P22;
    Display_dat = Motor_speed; //Motor_speed
```

```
switch(Motor_sta)
```

```
{
```

```
case START:
```

```
    MOTOR_START();
    Motor_sta = RUN;
    break;
```

```
case RUN:
```

```
    SPEED_ADJ();
    if((KEY_detect() == 2)||(BRK_occur == TRUE))
        Motor_sta = STOP;
```

```
    break;
```

```
case STOP:
```

```
    MOTOR_STOP();
    Motor_sta = IDLE;
    break;
```

```
case IDLE:
```

```
    if(KEY_detect()==1)
        Motor_sta = START;
    BRK_occur = FALSE;
    Motor_speed = 0;
    CAPI_avg = 0;
    CAPI_cnt = 0;
    CAPI_sum = 0;
    break;
```

```
}
```

```
}
```

```
}
```

```
void TIM0_ISR() interrupt 1
```

```
{
```

```
    TH0=0xf0;
    if(Display_num>8)
        Display_num=1;
    LED4_Display(Display_dat,Display_num);
    Display_num=(Display_num<<1);
```

```
}
```

```
void PWMA_ISR() interrupt 7
```

```
{
```

```
    if((PWMA_SR1 & 0x20))
    {
        switch(RD_HALL())
        {
```

```

    case 3:
        PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR3 /= PWMA_FORCE_INACTIVE;
        PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR1 /= PWMA_OCMODE_PWM2;
        break;
    case 2:
        PWMA_CCER1 &= CC2N_POLARITY_LOW;
        PWMA_CCER2 /= CC3N_POLARITY_HIGH;
        break;
    case 6:
        PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR1 /= PWMA_FORCE_INACTIVE;
        PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR2 /= PWMA_OCMODE_PWM2;
        break;
    case 4:
        PWMA_CCER1 /= CC1N_POLARITY_HIGH;
        PWMA_CCER2 &= CC3N_POLARITY_LOW;
        break;
    case 5:
        PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR2 /= PWMA_FORCE_INACTIVE;
        PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR3 /= PWMA_OCMODE_PWM2;
        break;
    case 1:
        PWMA_CCER1 &= CC1N_POLARITY_LOW;
        PWMA_CCER1 /= CC2N_POLARITY_HIGH;
        break;
}

CAP1_sum += PWMB_CAP1_v;
CAP1_cnt++;
if(CAP1_cnt==128)
{
    CAP1_cnt=0;
    CAP1_avg = (CAP1_sum>>7);
    CAP1_sum = 0;
    Motor_speed = 5000000/CAP1_avg;
}

PWMA_SRI &= ~0x20; //清零
}
if((PWMA_SRI & 0x80)) //BRK
{
    BRK_occur = TRUE;
    PWMA_SRI &= ~0x80; //清零
}
}

void PWMB_ISR() interrupt 23
{
    if((PWMB_SRI & 0x02))
    {
        PWMB_CAP1_v = PWMB_CCR5H;
        PWMB_CAP1_v = (PWMB_CAP1_v<<8) + PWMB_CCR5L;
        PWMB_SRI &= ~0x02;
    }
}

```



```
}

void DelayXus(unsigned char delayTime)
{
    int i = 0;
    while( delayTime--)
    {
        for( i = 0 ; i < 1 ; i++);
    }
}

void DelayXms( unsigned char delayTime )
{
    int i = 0;
    while( delayTime--)
    {
        for( i = 0 ; i < 2 ; i++)
        {
            DelayXus(100);
        }
    }
}

unsigned int ADC_Convert(u8 ch)
{
    u16 res=0;

    ADC_CONTR &= ~0x0f;
    ADC_CONTR /= ch;
    ADC_CONTR /= 0x40;
    DelayXus(1);
    while (!(ADC_CONTR & 0x20));
    ADC_CONTR &= ~0x20;

    res = ADC_RES;
    res = (res<<2)+(ADC_RESL>>6);
    return res;
}

void SPEED_ADJ()
{
    u16 ADC_result;

    ADC_result = (ADC_Convert(RV09_CH)/3);
    PWMA_CCR1H = (u8)(ADC_result >> 8);           //计数器比较值
    PWMA_CCR1L = (u8)(ADC_result);
    PWMA_CCR2H = (u8)(ADC_result >> 8);
    PWMA_CCR2L = (u8)(ADC_result);
    PWMA_CCR3H = (u8)(ADC_result >> 8);
    PWMA_CCR3L = (u8)(ADC_result);
}

unsigned char RD_HALL()
{
    unsigned char Hall_sta = 0;

    (P17)? (Hall_sta|=0x01) : (Hall_sta&=~0x01);
    (P54)? (Hall_sta|=0x02) : (Hall_sta&=~0x02);
    (P33)? (Hall_sta|=0x04) : (Hall_sta&=~0x04);
}
```

```

    return Hall_sta;
}

void MOTOR_START()
{
    u16 temp;
    u16 ADC_result;

    PWMA_CCR1H = (u8)(PWMA_STPulse >> 8);           //计数器比较值
    PWMA_CCR1L = (u8)(PWMA_STPulse);
    PWMA_CCR2H = (u8)(PWMA_STPulse >> 8);
    PWMA_CCR2L = (u8)(PWMA_STPulse);
    PWMA_CCR3H = (u8)(PWMA_STPulse >> 8);
    PWMA_CCR3L = (u8)(PWMA_STPulse);
    PWMA_BKR /= 0x80;                                //主输出使能相当于总开关
    PWMA_IER /= 0xA0;                                //使能中断

    switch(RD_HALL())
    {
    case 1:
        PWMA_CCER1 &= CC1N_POLARITY_LOW;
        PWMA_CCER1 /= CC2N_POLARITY_HIGH;
        PWMA_CCER2 &= CC3N_POLARITY_LOW;
        PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR3 /= PWMA_FORCE_INACTIVE;
        PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR2 /= PWMA_FORCE_INACTIVE;
        PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR1 /= PWMA_OCMODE_PWM2;
        break;
    case 3:
        PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR3 /= PWMA_FORCE_INACTIVE;
        PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR2 /= PWMA_FORCE_INACTIVE;
        PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR1 /= PWMA_OCMODE_PWM2;
        PWMA_CCER1 &= CC1N_POLARITY_LOW;
        PWMA_CCER1 &= CC2N_POLARITY_LOW;
        PWMA_CCER2 /= CC3N_POLARITY_HIGH;
        break;
    case 2:
        PWMA_CCER1 &= CC1N_POLARITY_LOW;
        PWMA_CCER1 &= CC2N_POLARITY_LOW;
        PWMA_CCER2 /= CC3N_POLARITY_HIGH;
        PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR1 /= PWMA_FORCE_INACTIVE;
        PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR2 /= PWMA_OCMODE_PWM2;
        PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR3 /= PWMA_FORCE_INACTIVE;
        break;
    case 6:
        PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR1 /= PWMA_FORCE_INACTIVE;
        PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR2 /= PWMA_OCMODE_PWM2;
        PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
    }
}

```

```

    PWMA_CCMR3 /= PWMA_FORCE_INACTIVE;
    PWMA_CCER1 /= CC1N_POLARITY_HIGH;
    PWMA_CCER1 &= CC2N_POLARITY_LOW;
    PWMA_CCER2 &= CC3N_POLARITY_LOW;
    break;
case 4:
    PWMA_CCER1 /= CC1N_POLARITY_HIGH;
    PWMA_CCER1 &= CC2N_POLARITY_LOW;
    PWMA_CCER2 &= CC3N_POLARITY_LOW;
    PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
    PWMA_CCMR1 /= PWMA_FORCE_INACTIVE;
    PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
    PWMA_CCMR2 /= PWMA_FORCE_INACTIVE;
    PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
    PWMA_CCMR3 /= PWMA_OCMODE_PWM2;
    break;
case 5:
    PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
    PWMA_CCMR1 /= PWMA_FORCE_INACTIVE;
    PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
    PWMA_CCMR2 /= PWMA_FORCE_INACTIVE;
    PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
    PWMA_CCMR3 /= PWMA_OCMODE_PWM2;
    PWMA_CCER1 &= CC1N_POLARITY_LOW;
    PWMA_CCER1 /= CC2N_POLARITY_HIGH;
    PWMA_CCER2 &= CC3N_POLARITY_LOW;
    break;
}
ADC_result = (ADC_Convert(RV09_CH)/3);

for(temp = PWMA_STPulse; temp > ADC_result; temp--)
{
    PWMA_CCR1H = (u8)(temp >> 8);           // 计数器比较值
    PWMA_CCR1L = (u8)(temp);
    PWMA_CCR2H = (u8)(temp >> 8);
    PWMA_CCR2L = (u8)(temp);
    PWMA_CCR3H = (u8)(temp >> 8);
    PWMA_CCR3L = (u8)(temp);
    DelayXms(10);
}
}

void MOTOR_STOP()
{
    PWMA_BKR &= ~0x80;
    PWMA_IER &= ~0xA0;
}

void LED4_Display (u16 dat,u8 num)
{
    switch(num)
    {
        case 0x01:
            LED_OUT(LED_0F[(dat/1)%10]);
            LED_OUT(0x01);
            RCLK = 0;
            RCLK = 1;
            break;
        case 0x02:

```

```
    LED_OUT(LED_0F[(dat/10)%10]);
    LED_OUT(0x02);
    RCLK = 0;
    RCLK = 1;
    break;
case 0x04:
    LED_OUT(LED_0F[(dat/100)%10]);
    LED_OUT(0x04);
    RCLK = 0;
    RCLK = 1;
    break;
case 0x08:
    LED_OUT(LED_0F[(dat/1000)%10]);
    LED_OUT(0x08);
    RCLK = 0;
    RCLK = 1;
    break;
}
}
```

```
void LED_OUT(u8 X)
{
    u8 i;

    for(i=8;i>=1;i--)
    {
        if (X&0x80) DIO=1;
        else DIO=0;
        X<<=1;
        SCLK = 0;
        SCLK = 1;
    }
}
```

```
unsigned char KEY_detect()
{
    if(!P02)
    {
        DelayXms(10);
        if(!P02)
        {
            return 1;
        }
        else return 0;
    }
    else if(!P03)
    {
        DelayXms(10);
        if(!P03)
        {
            return 2;
        }
        else return 0;
    }
    else return 0;
}
```

18.8.2 BLDC 无刷直流电机驱动(无 HALL)

C 语言代码

//测试工作频率为 11.0592MHz
 //本例程实现如下功能: 通过 3 组 PWM 通道控制无霍尔马达运转
 //本例程仅适用 57BL02 马达在 24V 无负载条件下演示

#include "stc16.h" //头文件见附录
 #include "intrins.h"

typedef unsigned char u8;
 typedef unsigned int u16;

#define TRUE 1
 #define FALSE 0

#define RV09_CH 6

#define PWMA_Period ((u16)280)
 #define PWMA_STPulse ((u16)245)

#define START 0x1A
 #define RUN 0x1B
 #define STOP 0x1C
 #define IDLE 0x1D

#define PWMA_OCMODE_MASK ((u8)0x70)
 #define PWMA_OCCE_ENABLE ((u8)0x80)
 #define PWMA_OCCE_DISABLE ((u8)0x00)
 #define PWMA_OCMODE_TIMING ((u8)0x00)
 #define PWMA_OCMODE_ACTIVE ((u8)0x10)
 #define PWMA_OCMODE_INACTIVE ((u8)0x20)
 #define PWMA_OCMODE_TOGGLE ((u8)0x30)
 #define PWMA_FORCE_INACTIVE ((u8)0x40)
 #define PWMA_FORCE_ACTIVE ((u8)0x50)
 #define PWMA_OCMODE_PWM1 ((u8)0x60)
 #define PWMA_OCMODE_PWM2 ((u8)0x70)
 #define CC1_POLARITY_HIGH ((u8)0x02)
 #define CC1N_POLARITY_HIGH ((u8)0x08)
 #define CC2_POLARITY_HIGH ((u8)0x20)
 #define CC2N_POLARITY_HIGH ((u8)0x80)
 #define CC1_POLARITY_LOW ((u8)~0x02)
 #define CC1N_POLARITY_LOW ((u8)~0x08)
 #define CC2_POLARITY_LOW ((u8)~0x20)
 #define CC2N_POLARITY_LOW ((u8)~0x80)
 #define CC1_OCENABLE ((u8)0x01)
 #define CC1N_OCENABLE ((u8)0x04)
 #define CC2_OCENABLE ((u8)0x10)
 #define CC2N_OCENABLE ((u8)0x40)
 #define CC1_OCDISABLE ((u8)~0x01)
 #define CC1N_OCDISABLE ((u8)~0x04)
 #define CC2_OCDISABLE ((u8)~0x10)
 #define CC2N_OCDISABLE ((u8)~0x40)
 #define CC3_POLARITY_HIGH ((u8)0x02)
 #define CC3N_POLARITY_HIGH ((u8)0x08)
 #define CC4_POLARITY_HIGH ((u8)0x20)
 #define CC4N_POLARITY_HIGH ((u8)0x80)

```

#define CC3_POLARITY_LOW      ((u8)~0x02)
#define CC3N_POLARITY_LOW     ((u8)~0x08)
#define CC4_POLARITY_LOW      ((u8)~0x20)
#define CC4N_POLARITY_LOW     ((u8)~0x80)
#define CC3_OCENABLE          ((u8)0x01)
#define CC3N_OCENABLE         ((u8)0x04)
#define CC4_OCENABLE          ((u8)0x10)
#define CC4N_OCENABLE         ((u8)0x40)
#define CC3_OCDISABLE         ((u8)~0x01)
#define CC3N_OCDISABLE        ((u8)~0x04)
#define CC4_OCDISABLE         ((u8)~0x10)
#define CC4N_OCDISABLE        ((u8)~0x40)

```

```

void UART_INIT();
void DelayXus(unsigned char delayTime);
void DelayXms( unsigned char delayTime);
unsigned int ADC_Convert(u8 ch);
void PWM_Init(void);
void SPEED_ADJ();
unsigned char RD_HALL();
void MOTOR_START();
void MOTOR_STOP();
unsigned char KEY_detect();

```

```

unsigned char Timer0_cnt=0xb0;
unsigned int HA=0;
unsigned int Motor_speed;
unsigned char Motor_sta = IDLE;
unsigned char BRK_occur=0;
unsigned int PWMB_CAP1_v=0;
unsigned int CAP1_avg=0;
unsigned char CAP1_cnt=0;
unsigned long CAP1_sum=0;

```

```

void main(void)
{
    unsigned int temp=0;
    unsigned int ADC_result=0;

```

```

    P_SW2= 0x80;
    P1 = 0x00;
    P0M1 = 0x0C;
    P0M0 = 0x01;
    P1M1 = 0xc0;
    P1M0 = 0x3F;
    P2M1 = 0x00;
    P2M0 = 0x38;
    P3M1 = 0x88;
    P3M0 = 0x02;

```

```

    ET0=1;
    TR0=0;
    ADCCFG = 0x0f;
    ADC_CONTR = 0x80;

```

```

    PWMA_ENO = 0x3F;           //PWMA 输出使能
    PWMB_ENO = 0x00;           //PWMB 输出使能
    PWMA_PS = 0x00;            //PWMA pin 选择
    PWMB_PS = 0xD5;            //PWMB pin 选择

```

```

/*****

```

```

输出比较模式 PWMx_duty = [CCRx/(ARR + 1)]*100

```

```

*****/

```

```

/*****PWMB BMF 输入 *****/

```

```

////////// 时基单元 //////////

```

```

PWMB_PSCRL = 15;

```

```

PWMB_ARRH = 0xff;

```

```

//自动重载寄存器, 计数器 overflow 点

```

```

PWMB_ARRL = 0xff;

```

```

PWMB_CCR8H = 0x00;

```

```

PWMB_CCR8L = 0x05;

```

```

////////// 通道配置 //////////

```

```

PWMB_CCMR1 = 0xf3;

```

```

//通道模式配置

```

```

PWMB_CCMR2 = 0xf1;

```

```

PWMB_CCMR3 = 0xf1;

```

```

PWMB_CCMR4 = 0x70;

```

```

PWMB_CCER1 = 0x11;

```

```

PWMB_CCER2 = 0x11;

```

```

////////// 模式配置 //////////

```

```

PWMB_CR2 = 0xf0;

```

```

PWMB_CR1 = 0x81;

```

```

PWMB_SMCR = 0x44;

```

```

////////// 使能 & 中断配置 //////////

```

```

PWMB_BKR = 0x80;

```

```

//主输出使能

```

```

PWMB_IER = 0x02;

```

```

//使能中断

```

```

/*****PWMA 控制马达换相 *****/

```

```

////////// 时基单元 //////////

```

```

PWMA_PSCRH = 0x00;

```

```

//预分频寄存器

```

```

PWMA_PSCRL = 0x00;

```

```

PWMA_ARRH = (u8)(PWMA_Period >> 8);

```

```

PWMA_ARRL = (u8)(PWMA_Period);

```

```

////////// 通道配置 //////////

```

```

PWMA_CCMR1 = 0x70;

```

```

//通道模式配置

```

```

PWMA_CCMR2 = 0x70;

```

```

PWMA_CCMR3 = 0x70;

```

```

PWMA_CCER1 = 0x11;

```

```

//配置通道输出使能和极性

```

```

PWMA_CCER2 = 0x01;

```

```

//配置通道输出使能和极性

```

```

PWMA_OISR = 0xAA;

```

```

//配置 MOE=0 时各通道输出电平

```

```

////////// 模式配置 //////////

```

```

PWMA_CR1 = 0xA0;

```

```

PWMA_CR2 = 0x24;

```

```

PWMA_SMCR = 0x20;

```

```

PWMA_BKR = 0x0c;

```

```

////////// 使能 & 中断配置 //////////

```

```

PWMA_CR1 |= 0x01;

```

```

//使能计数器

```

```

EA = 1;

```

```

UART_INIT();

```

```

while (1)

```

```

{

```

```

    switch(Motor_sta)

```

```

    {

```

```

        case START:

```

```

            MOTOR_START();

```

```

            Motor_sta = RUN;

```

```

            for(temp = PWMA_STPulse; temp > ADC_result; temp--)

```

```

            //开环启动

```

```

            {

```

```

                ADC_result = (ADC_Convert(RV09_CH)/4);

```

```

        PWMA_CCR1H = (u8)(temp >> 8);
        PWMA_CCR1L = (u8)(temp);
        PWMA_CCR2H = (u8)(temp >> 8);
        PWMA_CCR2L = (u8)(temp);
        PWMA_CCR3H = (u8)(temp >> 8);
        PWMA_CCR3L = (u8)(temp);
        DelayXms(10);
    }
    break;
case RUN:
    SPEED_ADJ(); // 马达调速
    if((BRK_occur == TRUE))
        Motor_sta = STOP;
    break;
case STOP:
    MOTOR_STOP();
    Motor_sta = IDLE;
    break;
case IDLE:
    if(KEY_detect()==1)
        Motor_sta = START; // 启动马达
        BRK_occur = FALSE;
        Motor_speed = 0;
        CAP1_avg = 0;
        CAP1_cnt = 0;
        CAP1_sum = 0;
        break;
    }
}

void TIM0_ISR() interrupt 1
{
    if(Motor_sta == START)
    {
        if(Timer0_cnt<0xe0) Timer0_cnt++;
        TH0=Timer0_cnt;

        switch(HA%6)
        {
        case 0:
            PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
            PWMA_CCMR3 /= PWMA_FORCE_INACTIVE;
            PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
            PWMA_CCMR1 /= PWMA_OCMODE_PWM2;
            break;
        case 1:
            PWMA_CCER1 &= CC2N_POLARITY_LOW;
            PWMA_CCER2 /= CC3N_POLARITY_HIGH;
            break;
        case 2:
            PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
            PWMA_CCMR1 /= PWMA_FORCE_INACTIVE;
            PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
            PWMA_CCMR2 /= PWMA_OCMODE_PWM2;
            break;
        case 3:
            PWMA_CCER1 /= CC1N_POLARITY_HIGH;
            PWMA_CCER2 &= CC3N_POLARITY_LOW;

```



```

        break;
    case 4:
        PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR2 /= PWMA_FORCE_INACTIVE;
        PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR3 /= PWMA_OCMODE_PWM2;
        break;
    case 5:
        PWMA_CCER1 &= CC1N_POLARITY_LOW;
        PWMA_CCER1 /= CC2N_POLARITY_HIGH;
        break;
    }
    HA++;
}

if(Motor_sta == RUN)
{
    TR0=0;
    switch(RD_HALL())
    {
    case 3:
        PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR3 /= PWMA_FORCE_INACTIVE;
        PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR1 /= PWMA_OCMODE_PWM2;
        break;
    case 1:
        PWMA_CCER1 &= CC2N_POLARITY_LOW;
        PWMA_CCER2 /= CC3N_POLARITY_HIGH;
        break;
    case 5:
        PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR1 /= PWMA_FORCE_INACTIVE;
        PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR2 /= PWMA_OCMODE_PWM2;
        break;
    case 4:
        PWMA_CCER1 /= CC1N_POLARITY_HIGH;
        PWMA_CCER2 &= CC3N_POLARITY_LOW;
        break;
    case 6:
        PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR2 /= PWMA_FORCE_INACTIVE;
        PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR3 /= PWMA_OCMODE_PWM2;
        break;
    case 2:
        PWMA_CCER1 &= CC1N_POLARITY_LOW;
        PWMA_CCER1 /= CC2N_POLARITY_HIGH;
        break;
    }
    }
}

void PWMA_ISR() interrupt 7
{
    if((PWMA_SRI & 0x20))
    {
        P00=0;
    }
}

```

```

    CAPI_sum += PWMB_CAPI_v;
    CAPI_cnt++;
    if(CAPI_cnt==128)
    {
        CAPI_cnt=0;
        CAPI_avg = (CAPI_sum>>7);
        CAPI_sum = 0;
        Motor_speed = 5000000/CAPI_avg;
    }
    PWMA_SRI &=~0x20; //清零
}
if((PWMA_SRI & 0x80)) //BRK
{
    BRK_occur = TRUE;
    PWMA_SRI &=~0x80; //清零
}
}

void PWMB_ISR() interrupt 23
{
    unsigned char ccr_tmp=0;

    if((PWMB_SRI & 0X02))
    {
        ccr_tmp = PWMB_CCR5H;
        if(ccr_tmp>1) //软件滤波
        {
            PWMB_CAPI_v = ccr_tmp;
            PWMB_CAPI_v = (PWMB_CAPI_v<<8) + PWMB_CCR5L;
            if(Motor_sta == RUN) //换向delay 计时
            {
                TR0=1;
                TH0 = 256-(PWMB_CAPI_v>>9);
            }
        }
        PWMB_SRI &=~0X02;
    }
}

void UART_INIT()
{
    SCON = 0x50; //8 位可变波特率
    AUXR = 0x40; //定时器1 为1T 模式
    TMOD = 0x20; //定时器1 为模式0(16 位自动重载)

    TL1 = 254;
    TH1 = 254;
    // ETI = 0;
    TRI = 1;
}

void DelayXus(unsigned char delayTime)
{
    int i = 0;
    while( delayTime--)
    {
        for( i = 0 ; i < 1 ; i++);
    }
}

```

```
void DelayXms( unsigned char delayTime )
```

```
{
    int i = 0;
    while( delayTime-- )
    {
        for( i = 0 ; i < 2 ; i++ )
        {
            DelayXus(100);
        }
    }
}
```

```
unsigned int ADC_Convert(u8 ch)
```

```
{
    u16 res=0;

    ADC_CONTR &= ~0x0f;
    ADC_CONTR /= ch;
    ADC_CONTR /= 0x40;
    DelayXus(1);
    while (!(ADC_CONTR & 0x20));
    ADC_CONTR &= ~0x20;

    res = ADC_RES;
    res = (res<<2)+(ADC_RESL>>6);

    if (res < 360) res=360;
    if (res > 900) res=900;

    return res;
}
```

```
void SPEED_ADJ()
```

```
{
    u16 ADC_result;

    ADC_result = (ADC_Convert(RV09_CH)/4);           //调速旋钮ADC 采样
    PWMA_CCR1H = (u8)(ADC_result >> 8);             //计数器比较值
    PWMA_CCR1L = (u8)(ADC_result);
    PWMA_CCR2H = (u8)(ADC_result >> 8);
    PWMA_CCR2L = (u8)(ADC_result);
    PWMA_CCR3H = (u8)(ADC_result >> 8);
    PWMA_CCR3L = (u8)(ADC_result);
}
```

```
unsigned char RD_HALL()
```

```
//读霍尔传感器
```

```
{
    unsigned char Hall_sta = 0;

    DelayXus(40);
    (P17)? (Hall_sta/=0x01) : (Hall_sta&=~0x01);
    (P54)? (Hall_sta/=0x02) : (Hall_sta&=~0x02);
    (P33)? (Hall_sta/=0x04) : (Hall_sta&=~0x04);

    return Hall_sta;
}
```

```
void MOTOR_START()
```

```

{
    PWMA_CCR1H = (u8)(PWMA_STPulse >> 8);           // 计数器比较值
    PWMA_CCR1L = (u8)(PWMA_STPulse);
    PWMA_CCR2H = (u8)(PWMA_STPulse >> 8);
    PWMA_CCR2L = (u8)(PWMA_STPulse);
    PWMA_CCR3H = (u8)(PWMA_STPulse >> 8);
    PWMA_CCR3L = (u8)(PWMA_STPulse);
    PWMA_BKR /= 0x80;                                // 主输出使能相当于总开关
    PWMA_IER = 0x00;                                // 使能中断
    TR0 = 1;

    while (HA < 6*20);

    PWMA_IER = 0xa0;                                // 使能中断
}

void MOTOR_STOP()
{
    PWMA_BKR &= ~0x80;
    PWMA_IER &= ~0x20;
}

unsigned char KEY_detect()
{
    if(!P37)
    {
        DelayXms(10);
        if(!P37)
        {
            return 1;
        }
        else return 0;
    }
    else if(!P03)
    {
        DelayXms(10);
        if(!P03)
        {
            return 2;
        }
        else return 0;
    }
    else return 0;
}

```

18.8.3 正交编码器模式

C 语言代码

//测试工作频率为 11.0592MHz

```

#include "stc16.h"           // 头文件见附录
#include "intrins.h"

```

```

unsigned char cnt_H, cnt_L;

```

```

void main(void)
{

```

```

P_SW2 = 0x80;

PIMI = 0x0f;
PIM0 = 0x00;

PWMA_ENO = 0x00;           //配置成TRGI 的pin 需关掉ENO 对应bit 并配成input
PWMA_PS = 0x00;           //00:PWM at P1

PWMA_PSCRH = 0x00;         //预分频寄存器
PWMA_PSCRL = 0x00;

PWMA_CCMR1 = 0x21;         //通道模式配置为输入, 接编码器,滤波器4 时钟
PWMA_CCMR2 = 0x21;         //通道模式配置为输入, 接编码器,滤波器4 时钟

PWMA_SMCR = 0x03;         //编码器模式3

PWMA_CCER1 = 0x55;         //配置通道使能和极性
PWMA_CCER2 = 0x55;         //配置通道使能和极性

PWMA_IER = 0x02;          //使能中断

PWMA_CR1 |= 0x01;          //使能计数器

EA = 1;

while (1);
}

/***** PWM 中断读编码器计数值 *****/
void PWMA_ISR() interrupt 7
{
    if (PWMA_SRI & 0X02)
    {
        P03 = ~P03;
        cnt_H = PWMA_CCR1H;
        cnt_L = PWMA_CCR1L;
        PWMA_SRI &= ~0X02;
    }
}

```

18.8.4 单脉冲模式（触发控制脉冲输出）

C 语言代码

//测试工作频率为11.0592MHz

```

#include "stc16.h"           //头文件见附录
#include "intrins.h"

void main(void)
{
    P_SW2 = 0x80;

    P0M1 = 0x00;
    P0M0 = 0xFF;
    PIMI = 0x0c;
    PIM0 = 0xF3;

```

```

PWMA_ENO = 0xF3;                                     //IO 输出 PWM
PWMA_PS = 0x00;                                       //00:PWM at P1

/*****
PWMx_duty = [CCRx/(ARR + 1)]*100
*****/
//配置成 TRGI 的 pin 需关掉 ENO 对应 bit 并配成 input
PWMA_PSCRH = 0x00;                                   //预分频寄存器
PWMA_PSCRL = 0x00;
PWMA_DTR = 0x00;                                     //死区时间配置

PWMA_CCMR1 = 0x68;                                   //通道模式配置
PWMA_CCMR2 = 0x01;                                   //配置成输入通道
PWMA_CCMR3 = 0x68;
PWMA_CCMR4 = 0x68;

PWMA_SMCR = 0x66;

PWMA_ARRH = 0x08;                                    //自动重载寄存器, 计数器 overflow 点
PWMA_ARRL = 0x00;

PWMA_CCR1H = 0x04;                                   //计数器比较值
PWMA_CCR1L = 0x00;
PWMA_CCR2H = 0x02;
PWMA_CCR2L = 0x00;
PWMA_CCR3H = 0x01;
PWMA_CCR3L = 0x00;
PWMA_CCR4H = 0x01;
PWMA_CCR4L = 0x00;

PWMA_CCER1 = 0x55;                                   //配置通道输出使能和极性
PWMA_CCER2 = 0x55;                                   //配置通道输出使能和极性

PWMA_BKR = 0x80;                                     //主输出使能 相当于总开关
PWMA_IER = 0x02;                                     //使能中断
PWMA_CR1 = 0x08;                                     //单脉冲模式
PWMA_CR1 |= 0x01;                                    //使能计数器

EA = 1;
while (1);
}

void PWMA_ISR() interrupt 7
{
    if (PWMA_SRI & 0X02)
    {
        P03 = ~P03;
        PWMA_SRI &= ~0X02;
    }
}

```

18.8.5 门控模式（输入电平使能计数器）

C 语言代码

//测试工作频率为11.0592MHz

```

#include "stc16.h"           //头文件见附录
#include "intrins.h"

void main(void)
{
    P_SW2 = 0x80;

    P0M1 = 0x00;
    P0M0 = 0xFF;
    P1M1 = 0x00;
    P1M0 = 0xFF;
    P3M1 = 0x04;
    P3M0 = 0x00;

    PWMA_ENO = 0xFF;          //IO 输出 PWM
    PWMA_PS = 0x00;          //00:PWM at P1

    /**
    PWMx_duty = [CCRx/(ARR + 1)]*100
    */
    //配置成TRGI 的 pin 需关掉ENO 对应 bit 并配成 input
    PWMA_PSCRH = 0x00;        //预分频寄存器
    PWMA_PSCRL = 0x00;
    PWMA_DTR = 0x00;          //死区时间配置

    PWMA_CCMR1 = 0x68;        //通道模式配置
    PWMA_CCMR2 = 0x68;        //配置成输入通道
    PWMA_CCMR3 = 0x68;
    PWMA_CCMR4 = 0x68;

    PWMA_SMCR = 0x75;         //门控触发模式 ETRF 输入

    PWMA_ARRH = 0x08;         //自动重装载寄存器, 计数器 overflow 点
    PWMA_ARRL = 0x00;

    PWMA_CCR1H = 0x04;        //计数器比较值
    PWMA_CCR1L = 0x00;        //
    PWMA_CCR2H = 0x02;        //
    PWMA_CCR2L = 0x00;        //
    PWMA_CCR3H = 0x01;        //
    PWMA_CCR3L = 0x00;        //
    PWMA_CCR4H = 0x01;        //
    PWMA_CCR4L = 0x00;        //

    PWMA_CCER1 = 0x55;        //配置通道输出使能和极性
    PWMA_CCER2 = 0x55;        //配置通道输出使能和极性

    PWMA_BKR = 0x80;          //主输出使能 相当于总开关
    PWMA_IER = 0x02;          //使能中断

    PWMA_CRI /= 0x01;         //使能计数器

    EA = 1;
    while (1);
}

void PWMA_ISR() interrupt 7
{

```

```

    if(PWMA_SR1 & 0X02)
    {
        P03 = ~P03;
        PWMA_SR1 &= ~0X02;
    }
}

```

18.8.6 外部时钟模式

C 语言代码

//测试工作频率为11.0592MHz

```

#include "stc16.h"           //头文件见附录
#include "intrins.h"

```

```

void main(void)
{

```

```

    P_SW2 = 0x80;

```

```

    P0M1 = 0x00;
    P0M0 = 0xFF;
    P1M1 = 0x00;
    P1M0 = 0xFF;
    P3M1 = 0x04;
    P3M0 = 0x00;

```

```

    PWMA_ENO = 0xFF;           //IO 输出 PWM
    PWMA_PS = 0x00;           //00:PWM at P1

```

/******

PWMx_duty = [CCRx/(ARR + 1)]*100

*****/

//配置成TRGI 的pin 需关掉ENO 对应bit 并配成input

```

    PWMA_PSCRH = 0x00;           //预分频寄存器
    PWMA_PSCRL = 0x00;
    PWMA_DTR = 0x00;           //死区时间配置

```

```

    PWMA_CCMR1 = 0x68;           //通道模式配置
    PWMA_CCMR2 = 0x68;           //配置成输入通道
    PWMA_CCMR3 = 0x68;
    PWMA_CCMR4 = 0x68;

```

```

    PWMA_SMCR = 0x77;           //ETRF 输入

```

```

    PWMA_ARRH = 0x08;           //自动重装载寄存器, 计数器 overflow 点
    PWMA_ARRL = 0x00;

```

```

    PWMA_CCR1H = 0x04;           //计数器比较值
    PWMA_CCR1L = 0x00;
    PWMA_CCR2H = 0x02;
    PWMA_CCR2L = 0x00;
    PWMA_CCR3H = 0x01;
    PWMA_CCR3L = 0x00;
    PWMA_CCR4H = 0x01;
    PWMA_CCR4L = 0x00;

```

```

    PWMA_CCER1 = 0x55;           //配置通道输出使能和极性

```



```

    PWMA_CCER2 = 0x55; //配置通道输出使能和极性

    PWMA_BKR = 0x80; //主输出使能 相当于总开关
    PWMA_IER = 0x02; //使能中断
    PWMA_CRI /= 0x01; //使能计数器

    EA = 1;
    while (1);
}

void PWMA_ISR() interrupt 7
{
    if(PWMA_SR1 & 0X02)
    {
        P03 = ~P03;
        PWMA_SR1 &= ~0X02;
    }
}

```

18.8.7 输入捕获模式测量脉冲周期（捕获上升沿到上升沿或者下降沿到下降沿）

C 语言代码

//测试工作频率为 11.0592MHz

```

#include "stc16.h" //头文件见附录
#include "intrins.h"

int cap;
int cap_new;
int cap_old;

void main(void)
{
    P_SW2 = 0x80;

    P0M1 = 0x00;
    P0M0 = 0x00;
    P1M1 = 0x00;
    P1M0 = 0x00;

    /*配置成 TRGI 的 pin 需关掉 ENO 对应 bit 并配成 input*/
    PWMA_ENO = 0x00; //IO 输出 PWM
    PWMA_PS = 0x00; //00: PWM at P1

    PWMA_CCMR1 = 0x01; //配置成输入通道
    PWMA_SMCR = 0x56;
    PWMA_CCER1 = 0x01; //配置通道输出使能和极性

    PWMA_IER = 0x02; //使能中断
    PWMA_CRI /= 0x01; //使能计数器

    EA = 1;
    while (1);
}

```

```

/* 通道1 输入, 捕获数据通过PWMA_CCR1H / PWMA_CCR1L 读取 */
void PWMA_ISR() interrupt 7
{
    if(PWMA_SR1 & 0x02)
    {
        cap_old = cap_new;
        cap_new = PWMA_CCR1H;           // 读取 CCR1H
        cap_new = (cap_new << 8) + PWMA_CCR1L; // 读取 CCR1L
        cap = cap_new - cap_old;
        PWMA_SR1 &= ~0x02;
    }
}

```

18.8.8 输入捕获模式测量脉冲高电平宽度（捕获上升沿到下降沿）

C 语言代码

//测试工作频率为11.0592MHz

```

#include "stc16.h"           //头文件见附录
#include "intrins.h"

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

    P_SW2 = 0x80;

    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;
    PWMA_CCMR2 = 0x02;
    PWMA_CCER1 = 0x11;
    PWMA_CCER1 |= 0x00;
    PWMA_CCER1 |= 0x20;
    PWMA_CR1 = 0x01;

    PWMA_IER = 0x04;
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 7
{
    unsigned int cnt;
    unsigned int cnt1;
    unsigned int cnt2;

    if (PWMA_SR1 & 0x04)
    {

```

// (CC1 捕获 T1I 上升沿, CC2 捕获 T1I 下降沿)

// CC1 为输入模式, 且映射到 TI1FP1 上

// CC2 为输入模式, 且映射到 TI1FP2 上

// 使能 CC1/CC2 上的捕获功能

// 设置捕获极性为 CC1 的上升沿

// 设置捕获极性为 CC2 的下降沿

// 使能 CC2 捕获中断

```

    PWMA_SR1 &= ~0x04;

    cnt1 = (PWMA_CCR1H << 8) + PWMA_CCR1L;
    cnt2 = (PWMA_CCR2H << 8) + PWMA_CCR2L;
    cnt = cnt2 - cnt1;           //差值即为高电平宽度
}
}

```

18.8.9 输入捕获模式测量脉冲低电平宽度（捕获下降沿到上升沿）

C 语言代码

//测试工作频率为 11.0592MHz

```

#include "stc16.h"           //头文件见附录
#include "intrins.h"

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

    P_SW2 = 0x80;

    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;
    PWMA_CCMR2 = 0x02;
    PWMA_CCER1 = 0x11;
    PWMA_CCER1 |= 0x00;
    PWMA_CCER1 |= 0x20;
    PWMA_CCR1 = 0x01;

    PWMA_IER = 0x02;
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 7
{
    unsigned int cnt;
    unsigned int cnt1;
    unsigned int cnt2;

    if (PWMA_SR1 & 0x02)
    {
        PWMA_SR1 &= ~0x02;

        cnt1 = (PWMA_CCR1H << 8) + PWMA_CCR1L;
        cnt2 = (PWMA_CCR2H << 8) + PWMA_CCR2L;
        cnt = cnt1 - cnt2;           //差值即为低电平宽度
    }
}

```

}

18.8.10 输入捕获模式同时测量脉冲周期和占空比

注意：只有 PWM1P、PWM2P、PWM5、PWM6 这些端口上才可同时测量周期和占空比

C 语言代码

//测试工作频率为 11.0592MHz

#include "stc16.h"

//头文件见附录

#include "intrins.h"

void main()

{

P1M0 = 0x00;

P1M1 = 0x00;

P3M0 = 0x00;

P3M1 = 0x00;

P5M0 = 0x00;

P5M1 = 0x00;

P_SW2 = 0x80;

PWMA_CCER1 = 0x00;

PWMA_CCMR1 = 0x01;

PWMA_CCMR2 = 0x02;

PWMA_CCER1 = 0x11;

PWMA_CCER1 |= 0x00;

PWMA_CCER1 |= 0x20;

PWMA_SMCR = 0x54;

PWMA_CR1 = 0x01;

PWMA_IER = 0x06;

EA = 1;

while (1);

}

void PWMA_ISR() interrupt 7

{

unsigned int cnt;

if (PWMA_SRI & 0x02)

{

PWMA_SRI &= ~0x02;

cnt = (PWMA_CCR1H << 8) + PWMA_CCR1L; //CC1 捕获周期宽度

}

if (PWMA_SRI & 0x04)

{

PWMA_SRI &= ~0x04;

cnt = (PWMA_CCR2H << 8) + PWMA_CCR2L; //CC2 捕获占空比 (高电平宽度)

}

}

18.8.11带死区控制的 PWM 互补输出

C 语言代码

//测试工作频率为11.0592MHz

```
#include "stc16.h"
#include "intrins.h"
```

//头文件见附录

```
void main(void)
{
```

```
    P_SW2 = 0x80;
```

```
    P0M1 = 0x00;
    P0M0 = 0xFF;
    P1M1 = 0x00;
    P1M0 = 0xFF;
```

```
    PWMA_ENO = 0xFF;
    PWMA_PS = 0x00;
```

//IO 输出 PWM
//00:PWM at P1

/******

```
PWMx_duty = [CCRx/(ARR + 1)]*100
```

*****/

```
    PWMA_PSCRH = 0x00;
    PWMA_PSCRL = 0x00;
    PWMA_DTR = 0x10;
```

//预分频寄存器
//死区时间配置

```
    PWMA_CCMR1 = 0x68;
    PWMA_CCMR2 = 0x68;
    PWMA_CCMR3 = 0x68;
    PWMA_CCMR4 = 0x68;
```

//通道模式配置

```
    PWMA_ARRH = 0x08;
    PWMA_ARRL = 0x00;
```

//自动重装载寄存器, 计数器 overflow 点

```
    PWMA_CCR1H = 0x04;
    PWMA_CCR1L = 0x00;
    PWMA_CCR2H = 0x02;
    PWMA_CCR2L = 0x00;
    PWMA_CCR3H = 0x01;
    PWMA_CCR3L = 0x00;
    PWMA_CCR4H = 0x01;
    PWMA_CCR4L = 0x00;
```

//计数器比较值

```
    PWMA_CCER1 = 0x55;
    PWMA_CCER2 = 0x55;
```

//配置通道输出使能和极性
//配置通道输出使能和极性

```
    PWMA_BKR = 0x80;
    PWMA_IER = 0x02;
    PWMA_CR1 = 0x01;
```

//主输出使能 相当于总开关
//使能中断
//使能计数器

```
    EA = 1;
    while (1);
```

```
}
```

```
void PWMA_ISR() interrupt 7
```

```
{  
    if(PWMA_SR1 & 0X02)  
    {  
        P03 = ~P03;  
        PWMA_SR1 &=~0X02;  
    }  
}
```

18.8.12 PWM 端口做外部中断（下降沿中断或者上升沿中断）

C 语言代码

//测试工作频率为 11.0592MHz

```
#include "stc16.h"           //头文件见附录  
#include "intrins.h"
```

```
void main(void)
```

```
{  
    P_SW2 = 0x80;  
  
    P1M1 = 0x00;  
    P1M0 = 0x00;  
    P3M1 = 0x00;  
    P3M0 = 0x00;  
  
    P_SW2 = 0x80;           //(捕获 PWM1P 上升沿/下降沿)  
  
    PWMA_CCER1 = 0x00;  
    PWMA_CCMR1 = 0x01;     //CC1 为输入模式,且映射到 TIIFP1 上  
    PWMA_CCER1 = 0x01;     //使能 CC1 上的捕获功能  
    PWMA_CCER1 |= 0x00;    //设置捕获极性为 CC1 的上升沿  
    // PWMA_CCER1 |= 0x02; //设置捕获极性为 CC1 的下降沿  
    PWMA_CR1 = 0x01;  
    PWMA_IER = 0x02;  
    EA = 1;  
  
    while (1);  
}
```

```
void PWMA_ISR() interrupt 7
```

```
{  
    if(PWMA_SR1 & 0X02)  
    {  
        P37 = ~P37;  
        PWMA_SR1 &=~0X02;  
    }  
}
```

18.8.13 输出任意周期和任意占空比的波形

C 语言代码

//测试工作频率为 11.0592MHz

```
#include "stc16.h"           //头文件见附录
```

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

```
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;
    PWMA_CCER1 = 0x00;           //写 CCMRx 前必须先清零 CCERx 关闭通道
    PWMA_CCMR1 = 0x60;           //设置 CC1 为PWMA 输出模式
    PWMA_CCER1 = 0x01;           //使能 CC1 通道
    PWMA_CCR1H = 0x01;           //设置占空比时间
    PWMA_CCR1L = 0x00;
    PWMA_ARRH = 0x02;           //设置周期时间
    PWMA_ARRL = 0x00;
    PWMA_ENO = 0x01;           //使能 PWM1P 端口输出
    PWMA_BKR = 0x80;           //使能主输出
    PWMA_CR1 = 0x01;           //开始计时

    while (1);
}
```

18.8.14使用 PWM 的 CEN 启动 PWMA 定时器，实时触发 ADC

C 语言代码

```
//测试工作频率为11.0592MHz
```

```
#include "stc16.h"           //头文件见附录
#include "intrins.h"
```

```
#define  ADC_POWER           0x80
#define  ADC_START           0x40
#define  ADC_FLAG            0x20
#define  ADC_EPWMT           0x10
```

```
void delay()
```

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

```
void main()
```

```
{
    P1M0 = 0x00;
    P1M1 = 0x01;
    P3M0 = 0x00;
```

```
P3M1 = 0x00;

P_SW2 /= 0x80;

ADC_CONTR = ADC_POWER / ADC_EPWMT / 0;           //选择 P1.0 为 ADC 输入通道
delay();                                           //等待 ADC 电源稳定
EADC = 1;

PWMA_CR2 = 0x10;                                   //CEN 信号为 TRGO, 可用于触发 ADC
PWMA_ARRH = 0x13;
PWMA_ARRL = 0x38;
PWMA_IER = 0x01;
PWMA_CR1 = 0x01;                                   //设置 CEN 启动 PWMA 定时器, 实时触发 ADC
EA = 1;

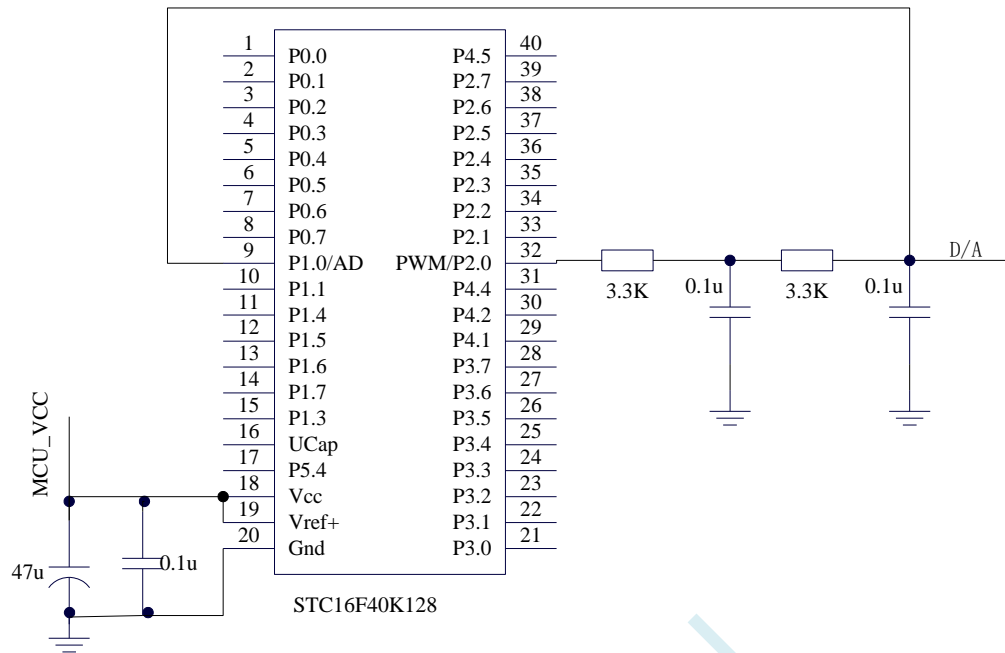
while (1);
}

void ADC_ISR() interrupt 5
{
    ADC_CONTR &= ~ADC_FLAG;
}

void PWMA_ISR() interrupt 7
{
    if(PWMA_SR1 & 0x01)
    {
        PWMA_SR1 &= ~0x01;
    }
}
```

18.8.15 利用 PWM 实现 16 位 DAC 的参考线路图

STC16 系列单片机的高级 PWM 定时器可输出 16 位的 PWM 波形, 再经过两级低通滤波即可产生 16 位的 DAC 信号, 通过调节 PWM 波形的高电平占空比即可实现 DAC 信号的改变。应用线路图如下图所示, 输出的 DAC 信号可输入到 MCU 的 ADC 进行反馈测量。



18.8.16 利用 PWM 实现互补 SPWM

高级 PWM 定时器 PWM1P/PWM1N, PWM2P/PWM2N, PWM3P/PWM3N, PWM4P/PWM4N 每个通道都可独立实现 PWM 输出, 或者两两互补对称输出。演示使用 PWM1P, PWM1N 产生互补的 SPWM。主时钟选择 24MHZ, PWM 时钟选择 1T, PWM 周期 2400, 死区 12 个时钟(0.5us), 正弦波表用 200 点, 输出正弦波频率 = $24000000 / 2400 / 200 = 50 \text{ Hz}$ 。

本程序仅仅是一个 SPWM 的演示程序, 用户可以通过上面的计算方法修改 PWM 周期和正弦波的点数和幅度。本程序输出频率固定, 如果需要变频, 请用户自己设计变频方案。

C 语言代码

//测试工作频率为24MHz

```
#include "stc16.h"           //头文件见附录
#include "intrins.h"
```

```
#define MAIN_Fosc 24000000L //定义主时钟
```

```
typedef unsigned char u8;
typedef unsigned int u16;
typedef unsigned long u32;
```

/****** 用户定义宏 *****/

```
#define PWMA_1 0x00 //P:P1.0 N:P1.1
#define PWMA_2 0x01 //P:P2.0 N:P2.1
#define PWMA_3 0x02 //P:P6.0 N:P6.1
```

```

#define PWMB_1      0x00      //P:P1.2/P5.4 N:P1.3
#define PWMB_2      0x04      //P:P2.2 N:P2.3
#define PWMB_3      0x08      //P:P6.2 N:P6.3

#define PWM3_1      0x00      //P:P1.4 N:P1.5
#define PWM3_2      0x10      //P:P2.4 N:P2.5
#define PWM3_3      0x20      //P:P6.4 N:P6.5

#define PWM4_1      0x00      //P:P1.6 N:P1.7
#define PWM4_2      0x40      //P:P2.6 N:P2.7
#define PWM4_3      0x80      //P:P6.6 N:P6.7
#define PWM4_4      0xC0      //P:P3.4 N:P3.3

#define ENO1P       0x01
#define ENO1N       0x02
#define ENO2P       0x04
#define ENO2N       0x08
#define ENO3P       0x10
#define ENO3N       0x20
#define ENO4P       0x40
#define ENO4N       0x80

```

/****** 本地变量声明 ******/

```

unsigned int code T_SinTable[]=
{
    1220, 1256, 1292, 1328, 1364, 1400, 1435, 1471,
    1506, 1541, 1575, 1610, 1643, 1677, 1710, 1742,
    1774, 1805, 1836, 1866, 1896, 1925, 1953, 1981,
    2007, 2033, 2058, 2083, 2106, 2129, 2150, 2171,
    2191, 2210, 2228, 2245, 2261, 2275, 2289, 2302,
    2314, 2324, 2334, 2342, 2350, 2356, 2361, 2365,
    2368, 2369, 2370, 2369, 2368, 2365, 2361, 2356,
    2350, 2342, 2334, 2324, 2314, 2302, 2289, 2275,
    2261, 2245, 2228, 2210, 2191, 2171, 2150, 2129,
    2106, 2083, 2058, 2033, 2007, 1981, 1953, 1925,
    1896, 1866, 1836, 1805, 1774, 1742, 1710, 1677,
    1643, 1610, 1575, 1541, 1506, 1471, 1435, 1400,
    1364, 1328, 1292, 1256, 1220, 1184, 1148, 1112,
    1076, 1040, 1005, 969, 934, 899, 865, 830,
    797, 763, 730, 698, 666, 635, 604, 574,
    544, 515, 487, 459, 433, 407, 382, 357,
    334, 311, 290, 269, 249, 230, 212, 195,
    179, 165, 151, 138, 126, 116, 106, 98,
    90, 84, 79, 75, 72, 71, 70, 71,
    72, 75, 79, 84, 90, 98, 106, 116,
    126, 138, 151, 165, 179, 195, 212, 230,
    249, 269, 290, 311, 334, 357, 382, 407,
    433, 459, 487, 515, 544, 574, 604, 635,
    666, 698, 730, 763, 797, 830, 865, 899,
    934, 969, 1005, 1040, 1076, 1112, 1148, 1184,
};

```

u16 PWMA_Duty;

u8 PWM_Index;

//SPWM 查表索引

/****** 主函数 ******/

```

void main(void)
{

```

```

P0M1 = 0;   P0M0 = 0;           // 设置为准双向口
P1M1 = 0;   P1M0 = 0;           // 设置为准双向口
P2M1 = 0;   P2M0 = 0;           // 设置为准双向口
P3M1 = 0;   P3M0 = 0;           // 设置为准双向口
P4M1 = 0;   P4M0 = 0;           // 设置为准双向口
P5M1 = 0;   P5M0 = 0;           // 设置为准双向口
P6M1 = 0;   P6M0 = 0;           // 设置为准双向口
P7M1 = 0;   P7M0 = 0;           // 设置为准双向口

PWMA_Duty = 1220;

P_SW2 /= 0x80;

PWMA_CCER1 = 0x00;              // 写 CCMRx 前必须先清零 CCxE 关闭通道
PWMA_CCER2 = 0x00;
PWMA_CCMR1 = 0x60;              // 通道模式配置
// PWMA_CCMR2 = 0x60;
// PWMA_CCMR3 = 0x60;
// PWMA_CCMR4 = 0x60;
PWMA_CCER1 = 0x05;              // 配置通道输出使能和极性
// PWMA_CCER2 = 0x55;

PWMA_ARRH = 0x09;               // 设置周期时间
PWMA_ARRL = 0x60;

PWMA_CCR1H = (u8)(PWMA_Duty >> 8); // 设置占空比时间
PWMA_CCR1L = (u8)(PWMA_Duty);

PWMA_DTR = 0x0C;               // 设置死区时间

PWMA_ENO = 0x00;
PWMA_ENO /= ENO1P;              // 使能输出
PWMA_ENO /= ENO1N;              // 使能输出
// PWMA_ENO /= ENO2P;           // 使能输出
// PWMA_ENO /= ENO2N;           // 使能输出
// PWMA_ENO /= ENO3P;           // 使能输出
// PWMA_ENO /= ENO3N;           // 使能输出
// PWMA_ENO /= ENO4P;           // 使能输出
// PWMA_ENO /= ENO4N;           // 使能输出

PWMA_PS = 0x00;                 // 高级 PWM 通道输出脚选择位
PWMA_PS /= PWMA_3;              // 选择 PWMA_3 通道
// PWMA_PS /= PWMB_3;           // 选择 PWMB_3 通道
// PWMA_PS /= PWM3_3;           // 选择 PWM3_3 通道
// PWMA_PS /= PWM4_3;           // 选择 PWM4_3 通道

PWMA_BKR = 0x80;                // 使能主输出
PWMA_IER = 0x01;                // 使能中断
PWMA_CRI /= 0x01;               // 开始计时

P_SW2 &= 0x7f;

EA = 1;                          // 打开总中断

while (1)
{
}
}

```

/****** 中断函数 ******/

void PWMA_ISR() interrupt 7

{

P_SW2 /= 0x80;

if (PWMA_SR1 & 0x01)

{

PWMA_SR1 &= ~0x01;

PWMA_Duty = T_SinTable[PWM_Index];

if (++PWM_Index >= 200)

PWM_Index = 0;

PWMA_CCR1H = (u8)(PWMA_Duty >> 8); // 设置占空比时间

PWMA_CCR1L = (u8)(PWMA_Duty);

}

PWMA_SR1 = 0;

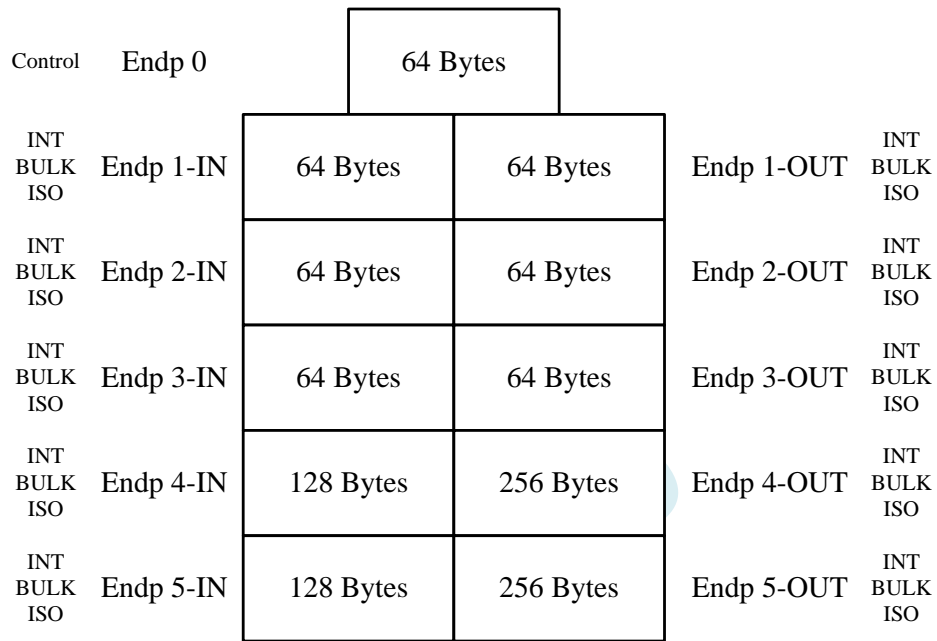
P_SW2 &= 0x7f;

}

19 USB 通用串行总线

STC16F 系列单片机内部集成 USB2.0/USB1.1 兼容全速 USB，6 个双向端点，支持 4 种端点传输模式（控制传输、中断传输、批量传输和同步传输），每个端点拥有 64 字节的缓冲区。

USB 模块共有 1280 字节的 FIFO，结构如下：



19.1 USB 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
USBCLK	USB 时钟控制寄存器	85H	ENCKM	PCKI[1:0]		CRE	TST_USB	TST_PHY	PHYTST[1:0]		0010,0000
USBCON	USB 控制寄存器	91H	ENUSB	USBRST	PS2M	PUEN	PDEN	DFREC	DP	DM	0000,0000
USBADR	USB 地址寄存器	96H	BUSY	AUTORD	UADR[5:0]						0000,0000
USBDAT	USB 数据寄存器	97H									0000,0000

19.1.1 USB 控制寄存器（USBCON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USBCON	91H	ENUSB	USBRST	PS2M	PUEN	PDEN	DFREC	DP	DM

ENUSB：USB 功能与 USB 时钟控制位

- 0：关闭 USB 功能与 USB 时钟
- 1：使能 USB 功能与 USB 时钟

ENRST：USB 复位设置控制位

- 0：关闭 USB 复位设置
- 1：使能 USB 复位

PS2M: PS2 mode 功能控制位

- 0: 关闭 PS2 mode 功能
- 1: 使能 PS2 mode 功能

PUEN: DP/DM 端口上 1.5K 上拉电阻控制位

- 0: 禁止上拉电阻
- 1: 使能上拉电阻

PDEN: DP/DM 端口上 500K 下拉电阻控制位

- 0: 禁止下拉电阻
- 1: 使能下拉电阻

DFREC: 差分接收状态位 (只读)

- 0: 当前 DP/DM 的差分状态为 “0”
- 1: 当前 DP/DM 的差分状态为 “1”

DP: D+端口状态 (PS2 为 0 时只读, PS2 为 1 时可读写)

- 0: 当前 D+为逻辑 0 电平
- 1: 当前 D+为逻辑 1 电平

DM: D-端口状态 (PS2 为 0 时只读, PS2 为 1 时可读写)

- 0: 当前 D-为逻辑 0 电平
- 1: 当前 D-为逻辑 1 电平

19.1.2 USB 时钟控制寄存器 (USBCLK)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USBCLK	85H	ENCKM	PCKI[1:0]		CRE	TST_USB	TST_PHY	PHYTST[1:0]	

ENCKM: PLL 倍频控制

- 0: 禁止 PLL 倍频
- 1: 使能 PLL 倍频

PCKI[1:0]: PLL 时钟选择

PCKI[1:0]	PLL 时钟源
00	6M
01	12M(default)
10	24M
11	IRC/2

CRE: 时钟追频控制位

- 0: 禁止时钟追频
- 1: 使能时钟追频

TST_USB: USB 测试模式

- 0: 禁止 USB 测试模式
- 1: 使能 USB 测试模式

TST_PHY: PHY 测试模式

- 0: 禁止 PHY 测试模式
- 1: 使能 PHY 测试模式

PHYTST[1:0]: USB PHY 测试

PHYTST[1:0]	方式	DP	DM
00	方式 0: 正常	x	x
01	方式 1: 强制 “1”	1	0
10	方式 2: 强制 “0”	0	1

11	方式 3: 强制单端 “0”	0	0
----	----------------	---	---

19.1.3 USB 间址地址寄存器 (USBADDR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USBADDR	96H	BUSY	AUTORD	UADR[5:0]					

BUSY: USB 寄存器读忙标志位

写 0: 无意义

写 1: 启动 USB 间接寄存器的读操作, 地址由 USBADDR 设定

读 0: USBDATA 寄存器中的数据有效

读 1: USBDATA 寄存器中的数据无效, USB 正在读取间接寄存器

AUTORD: USB 寄存器自动读标志, 用于 USB 的 FIFO 的块读取

写 0: 每次读取间接 USB 寄存器都必须先写 BUSY 标志位

写 1: 当软件读取 USBDATA 时, 下一个 USB 间接寄存器的读取将自动启动 (USBADDR 不变)

UADR[5:0]: USB 间接寄存器的地址

19.1.4 USB 间址数据寄存器 (USBDATA)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
USBDATA	97H	UDAT[7:0]							

UDAT[7:0]: 用于间接读写 USB 寄存器

19.2 USB 控制器寄存器 (SIE)

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
30H	UTRKCTL	UTRKSTS						
28H								
20H	FIFO0	FIFO1	FIFO2	FIFO3	FIFO4	FIFO5		
18H								
10H	INMAXP	CSR0 INCSR1	INCSR2	OUTMAXP	OUTCSR1	OUTCSR2	COUNT0 OUTCOUNT1	OUTCOUNT2
08H		INTROUT1E		INTRUSBE	FRAME1	FRAME2	INDEX	
00H	FADDR	POWER	INTRIN1	-	INTROUT1	-	INTRUSB	INTRIN1E

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
FADDR	USB 功能地址寄存器	00H	UPDATE	UADDR[6:0]							0000,0000
POWER	USB 电源管理寄存器	01H	ISOUD	-	-	-	USBRST	USBRSU	USBSUS	ENSUS	0xxx,0000
INTRIN1	USB 端点 IN 中断标志位	02H	-	-	EP5INIF	EP4INIF	EP3INIF	EP2INIF	EP1INIF	EP0IF	xx00,0000
INTROUT1	USB 端点 OUT 中断标志位	04H	-	-	EP5OUTIF	EP4OUTIF	EP3OUTIF	EP2OUTIF	EP1OUTIF	-	xx00,000x
INTRUSB	USB 电源中断标志位	06H	-	-	-	-	SOFIF	RSTIF	RSUIF	SUSIF	xxxx,0000
INTRIN1E	USB 端点 IN 中断允许位	07H	-	-	EP5INIE	EP4INIE	EP3INIE	EP2INIE	EP1INIE	EP0IE	xx11,1111
INTROUT1E	USB 端点 OUT 中断允许位	09H	-	-	EP5OUTIE	EP4OUTIE	EP3OUTIE	EP2OUTIE	EP1OUTIE	-	xx11,111x
INTRUSBE	USB 电源中断允许位	0BH	-	-	-	-	SOFIE	RSTIE	RSUIE	SUSIE	xxxx,0110
FRAME1	USB 数据帧号低字节	0CH	FRAME[7:0]								0000,0000

FRAME2	USB 数据帧号高字节	0DH	-	-	-	-	-	FRAME[10:8]			xxxx,x000
INDEX	USB 端点索引寄存器	0EH	-	-	-	-	-	INDEX[2:0]			xxxx,x000
INMAXP	IN 端点的最大数据包大小	10H	INMAXP[7:0]								0000,0000
CSR0	端点 0 控制状态寄存器	11H	SSUEND	SOPRDY	SDSTL	SUEND	DATEND	STSTL	IPRDY	OPRDY	0000,0000
INCSR1	IN 端点控制状态寄存器 1	11H	CLRDT	STSTL	SDSTL	FLUSH	-	UNDRUN	FIFONE	IPRDY	0000,x000
INCSR2	IN 端点控制状态寄存器 2	12H	AUTOSET	ISO	MODE	ENDMA	FCDT	-	-	-	0010,0xxx
OUTMAXP	OUT 端点的最大数据包大小	13H	OUTMAXP[7:0]								0000,0000
OUTCSR1	OUT 端点控制状态寄存器 1	14H	CLRDT	STSTL	SDSTL	FLUSH	DATERR	OVRRUN	FIFOFUL	OPRDY	0000,0000
OUTCSR2	OUT 端点控制状态寄存器 2	15H	AUTOCLR	ISO	ENDMA	DMAMD	-	-	-	-	0000,xxxx
COUNT0	端点 0 的 OUT 长度	16H	-	OUTCNT0[6:0]							x000,0000
OUTCOUNT1	USB 端点 OUT 长度低字节	16H	OUTCNT[7:0]								0000,0000
OUTCOUNT2	USB 端点 OUT 长度高字节	17H	-	-	-	-	-	OUTCNT[10:8]			xxxx,x000
FIFO0	端点 0 的 FIFO 访问寄存器	20H	FIFO0[7:0]								0000,0000
FIFO1	端点 1 的 FIFO 访问寄存器	21H	FIFO1[7:0]								0000,0000
FIFO2	端点 2 的 FIFO 访问寄存器	22H	FIFO2[7:0]								0000,0000
FIFO3	端点 3 的 FIFO 访问寄存器	23H	FIFO3[7:0]								0000,0000
FIFO4	端点 4 的 FIFO 访问寄存器	24H	FIFO4[7:0]								0000,0000
FIFO5	端点 5 的 FIFO 访问寄存器	25H	FIFO5[7:0]								0000,0000
UTRKCTL	USB 跟踪控制寄存器	30H	FTM1	FTM0	INTV[1:0]		ENST5	RES[2:0]			1011,1011
UTRKSTS	USB 跟踪状态寄存器	31H	INTVCNT[3:0]				STS[1:0]		TST_UTRK	UTRK_RDY	1111,00x0

19.2.1 USB 功能地址寄存器 (FADDR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
FADDR	00H	UPDATE	UADDR[6:0]						

UPDATE: 更新 USB 功能地址

0: 最后的 UADDR 地址已生效

1: 最后的 UADDR 地址还未生效

UADDR[6:0]: 保存 USB 的 7 位功能地址

19.2.2 USB 电源控制寄存器 (POWER)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
POWER	01H	ISOUD	-	-	-	USBRST	USBRSU	USBSUS	ENSUS

ISOUD (ISO Update): ISO 更新

0: 当软件向 IPRDY 写“1”时, USB 在收到下一个 IN 令牌后发送数据包

1: 当软件向 IPRDY 写“1”时, USB 在收到 SOF 令牌后发送数据包, 如果 SOF 令牌之前收到 IN 令牌, 则 USB 发送长度为 0 的数据包

USBRST (USB Reset): USB 复位控制位

向此位写“1”, 可强制产生异步 USB 复位。读取此位可以获得当前总线上的复位状态信息

0: 总线上没有检测到复位信号

1: 总线上检测到了复位信号

USBRSU (USB Resume): USB 恢复控制位

以软件方式在总线上强制产生恢复信号, 以便将 USB 设备从挂起方式进行远程唤醒。当 USB 处于挂起模式 (USBSUS=1) 时, 向此位写“1”, 将强制在 USB 总线上产生恢复信号, 软件应在 10-15ms 后向此位写“0”, 以结束恢复信号。软件向 USBRSU 写入“0”后将产生 USB 恢复中断, 此时硬件会自动将 USBSUS 清“0”

USBSUS（USB Suspend）：USB 挂起控制位

当 USB 进入挂起方式时，此位被硬件置“1”。当以软件方式在总线上强制产生恢复信号后或者在总线上检测到恢复信号时且在读取了 INTRUSB 寄存器后，硬件自动将此位清“0”。

ENSUS（Enable Suspend Detection）：使能 USB 挂起方式检测

0：禁止挂起检测，USB 将忽略总线上的挂起信号

1：使能挂起检测，当检测到总线上的挂起信号，USB 将进入挂起方式

19.2.3 USB 端点 IN 中断标志位（INTRIN1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTRIN1	02H	-	-	EP5INIF	EP4INIF	EP3INIF	EP2INIF	EP1INIF	EP0IF

EP5INIF：端点 5 的 IN 中断标志位

0：端点 5 的 IN 中断无效

1：端点 5 的 IN 中断有效

EP4INIF：端点 4 的 IN 中断标志位

0：端点 4 的 IN 中断无效

1：端点 4 的 IN 中断有效

EP3INIF：端点 3 的 IN 中断标志位

0：端点 3 的 IN 中断无效

1：端点 3 的 IN 中断有效

EP2INIF：端点 2 的 IN 中断标志位

0：端点 2 的 IN 中断无效

1：端点 2 的 IN 中断有效

EP1INIF：端点 1 的 IN 中断标志位

0：端点 1 的 IN 中断无效

1：端点 1 的 IN 中断有效

EP0IF：端点 0 的 IN/OUT 中断标志位

0：端点 0 的 IN/OUT 中断无效

1：端点 0 的 IN/OUT 中断有效

在软件读取 INTRIN1 寄存器后，硬件将自动清除 INTRIN1 中的所有的中断标志

19.2.4 USB 端点 OUT 中断标志位（INTROUT1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTROUT1	04H	-	-	EP5OUTIF	EP4OUTIF	EP3OUTIF	EP2OUTIF	EP1OUTIF	-

EP5OUTIF：端点 5 的 OUT 中断标志位

0：端点 5 的 OUT 中断无效

1：端点 5 的 OUT 中断有效

EP4OUTIF：端点 4 的 OUT 中断标志位

0：端点 4 的 OUT 中断无效

1：端点 4 的 OUT 中断有效

EP3OUTIF：端点 3 的 OUT 中断标志位

0：端点 3 的 OUT 中断无效

1：端点 3 的 OUT 中断有效

EP2OUTIF：端点 2 的 OUT 中断标志位

0：端点 2 的 OUT 中断无效

1：端点 2 的 OUT 中断有效

EP1OUTIF: 端点 1 的 OUT 中断标志位

0: 端点 1 的 OUT 中断无效

1: 端点 1 的 OUT 中断有效

在软件读取 INTROUT1 寄存器后, 硬件将自动清除 INTROUT1 中的所有的中断标志

19.2.5 USB 电源中断标志 (INTRUSB)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTRUSB	06H	-	-	-	-	SOFIF	RSTIF	RSUIF	SUSIF

SOFIF: USB 帧起始信号中断标志

0: USB 帧起始信号中断无效

1: USB 帧起始信号中断有效

RSTIF: USB 复位信号中断标志

0: USB 复位信号中断无效

1: USB 复位信号中断有效

RSUIF: USB 恢复信号中断标志

0: USB 恢复信号中断无效

1: USB 恢复信号中断有效

SUSIF: USB 挂起信号中断标志

0: USB 挂起信号中断无效

1: USB 挂起信号中断有效

在软件读取 INTRUSB 寄存器后, 硬件将自动清除 INTRUSB 中的所有的中断标志

19.2.6 USB 端点 IN 中断允许寄存器 (INTRIN1E)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTRIN1E	07H	-	-	EP5INIE	EP4INIE	EP3INIE	EP2INIE	EP1INIE	EP0IE

EP5INIE: 端点 5 的 IN 中断控制位

0: 禁止端点 5 的 IN 中断

1: 允许端点 5 的 IN 中断

EP4INIE: 端点 4 的 IN 中断控制位

0: 禁止端点 4 的 IN 中断

1: 允许端点 4 的 IN 中断

EP3INIE: 端点 3 的 IN 中断控制位

0: 禁止端点 3 的 IN 中断

1: 允许端点 3 的 IN 中断

EP2INIE: 端点 2 的 IN 中断控制位

0: 禁止端点 2 的 IN 中断

1: 允许端点 2 的 IN 中断

EP1INIE: 端点 1 的 IN 中断控制位

0: 禁止端点 1 的 IN 中断

1: 允许端点 1 的 IN 中断

EP0IE: 端点 0 的 IN/OUT 中断控制位

0: 禁止端点 0 的 IN/OUT 中断

1: 允许端点 0 的 IN/OUT 中断

19.2.7 USB 端点 OUT 中断允许寄存器 (INTROUT1E)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTROUT1E	09H	-	-	EP5OUTIE	EP4OUTIE	EP3OUTIE	EP2OUTIE	EP1OUTIE	-

EP5OUTIE: 端点 5 的 OUT 中断控制位

0: 禁止端点 5 的 OUT 中断

1: 允许端点 5 的 OUT 中断

EP4OUTIE: 端点 4 的 OUT 中断控制位

0: 禁止端点 4 的 OUT 中断

1: 允许端点 4 的 OUT 中断

EP3OUTIE: 端点 3 的 OUT 中断控制位

0: 禁止端点 3 的 OUT 中断

1: 允许端点 3 的 OUT 中断

EP2OUTIE: 端点 2 的 OUT 中断控制位

0: 禁止端点 2 的 OUT 中断

1: 允许端点 2 的 OUT 中断

EP1OUTIE: 端点 1 的 OUT 中断控制位

0: 禁止端点 1 的 OUT 中断

1: 允许端点 1 的 OUT 中断

19.2.8 USB 电源中断允许寄存器 (INTRUSB)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTRUSB	0BH	-	-	-	-	SOFIE	RSTIE	RSUIE	SUSIE

SOFIE: USB 帧起始信号中断控制位

0: 禁止 USB 帧起始信号中断

1: 允许 USB 帧起始信号中断

RSTIE: USB 复位信号中断控制位

0: 禁止 USB 复位信号中断

1: 允许 USB 复位信号中断

RSUIE: USB 恢复信号中断控制位

0: 禁止 USB 恢复信号中断

1: 允许 USB 恢复信号中断

SUSIE: USB 挂起信号中断控制位

0: 禁止 USB 挂起信号中断

1: 允许 USB 挂起信号中断

19.2.9 USB 数据帧号寄存器 (FRAME_n)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
FRAME1	0CH	FRAME[7:0]							
FRAME2	0DH	-	-	-	-	-	FRAME[10:8]		

FRAME[10:0]: 用于保存最后接收到的数据帧的 11 位帧号

19.2.10 USB 端点索引寄存器 (INDEX)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INDEX	0EH	-	-	-	-	-	INDEX[2:0]		

INDEX[2:0]: 选择 USB 端点

INDEX[2:0]	目标端点
000	端点 0
001	端点 1
010	端点 2
011	端点 3
100	端点 4
101	端点 5

19.2.11 IN 端点的最大数据包大小 (INMAXP)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INMAXP	10H	INMAXP[7:0]							

INMAXP[7:0]: 设置 USB 的 IN 端点最大数据包的大小

当需要获取/设置此信息时, 首先必须使用 INDEX 来选择目标端点 0~5

19.2.12 USB 端点 0 控制状态寄存器 (CSR0)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CSR0	11H	SSUEND	SOPRDY	SDSTL	SUEND	DATEND	STSTL	IPRDY	OPRDY

SSUEND (Serviced Setup End)

SETUP 结束事件处理完成标志。当处理完成 SETUP 结束事件(SUEND)后, 软件需要设置 SSUEND 标志位, 硬件检测到 SSUEND 被写入“1”时自动将 SUEND 位清“0”。

SOPRDY (Serviced OPRDY)

OPRDY 事件处理完成标志。当处理完成从端点 0 接收到的数据包后, 软件需要设置 SOPRDY 标志位, 硬件检测到 SOPRDY 被写入“1”时自动将 OPRDY 位清“0”。

SDSTL (Send Stall)

当接收到错误的条件或者不支持的请求时, 可以向此位写“1”来结束当前的数据传输。当 STALL 信号被发送后, 硬件自动将此位清“0”。

SUEND (Setup End)

SETUP 安装包结束标志。当一次控制传输在软件向 DATAEND 位写“1”之前结束时, 硬件将此只读位置“1”。当软件向 SSUEND 写“1”后, 硬件将该位清“0”。

DATEND (Data End)

数据结束。软件应在下列情况下将此位写“1”:

- 1、当发送最后一个数据包后, 固件向 IPRDY 写“1”时;
 - 2、当发送一个零长度数据包后, 固件向 IPRDY 写“1”时;
 - 3、当接收完最后一个数据包后, 固件向 SOPRDY 写“1”时;
- 该位将被硬件自动清“0”

STSTL (Sent Stall)

STALL 信号发送完成标志。发送完成 STALL 信号后, 硬件将该位置“1”。该位必须用软件清“0”。

IPRDY (IN Packet Ready)

IN 数据包准备完成标志。软件应在将一个要发送的数据包装入到端点 0 的 FIFO 后, 将该位置“1”。在发生下列条件之一时, 硬件将该位清“0”:

- 1、数据包已发送时;
- 2、数据包被一个 SETUP 包覆盖时;
- 3、数据包被一个 OUT 包覆盖时;

OPRDY (OUT Packet Ready)

OUT 数据包准备完成标志。当收到一个 OUT 数据包时，硬件将该只读位置“1”，并产生中断。

该位只在软件向 SOPRDY 位写“1”时才被清“0”。

当需要获取/设置此信息时，首先必须使用 INDEX 来选择目标端点 0

19.2.13 IN 端点控制状态寄存器 1 (INCSR1)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UBCSR1	11H	CLRDT	STSTL	SDSTL	FLUSH	-	UNDRUN	FIFONE	IPRDY

CLRDT (Clear Data Toggle)：复位 IN 数据切换位。

当 IN 端点由于被重新配置或者被 STALL 而需要将数据切换位复位到“0”时，软件需要向此数据位写“1”。

STSTL (Sent Stall)：STALL 信号发送完成标志。

当 STALL 信号被发送完成后，硬件会将该位置“1”（此时 FIFO 被清空，IPRDY 位被清“0”）。该标志必须用软件清“0”。

SDSTL (Send Stall)：STALL 信号发送请求位。

软件应向该位写“1”以产生 STALL 信号作为对一个 IN 令牌的应答。软件应向该位写“0”以结束 STALL 信号。该位对 ISO 方式没有影响。

FLUSH (FIFO Flush)：清除 IN 端点的 FIFO 的下一个数据包。

向该位写“1”将从 IN 端点 FIFO 中清除待发送的下一个数据包。FIFO 指针被复位，IPRDY 位被清“0”。如果 FIFO 中包含多个数据包，软件必须对每个数据包向 FLUSH 写“1”。当 FIFO 清空完成后，硬件将 FLUSH 位清“0”。

UNDRUN (Data Underrun)：数据不足。

该位的功能取决于 IN 端点的方式：

ISO 方式：在 IPRDY 为“0”且收到一个 IN 令牌后发送了一个零长度数据包时，该位被置“1”。

中断/批量方式：当使用 NAK 作为对一个 IN 令牌的应答时，该位被置“1”。

该位必须用软件清“0”。

FIFONE (FIFO Not Empty)：IN 端点的 FIFO 非空标志

0: IN 端点的 FIFO 为空

1: IN 端点的 FIFO 包含有一个或者多个数据包

IPRDY (IN Packet Ready)：IN 数据包准备完成标志。

软件应在将一个要发送的数据包装入到端点的 FIFO 后，将该位置“1”。在发生下列条件之一时，硬件将该位清“0”：

- 1、数据包已发送时；
- 2、自动设置被使能 (AUTOSET = ‘1’) 且端点 IN 的 FIFO 数据包达到 INMAXP 所设置的值；
- 3、如果端点处于同步方式且 ISOUD 为“1”，在收到下一个 SOF 之前 IPRDY 的读出值总是为 0。

需要获取/设置此信息时，首先必须使用 INDEX 来选择目标端点 1~5

19.2.14 IN 端点控制状态寄存器 2 (INCSR2)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INCSR2	12H	AUTOSET	ISO	MODE	ENDMA	FCDT	-	-	-

AUTOSET：自动设置 IPRDY 标志控制位。

0: 禁止自动设置 IPRDY 标志

1: 使能自动设置 IPRDY（必须向 IN FIFO 中装载的数据达到 INMAXP 所设置的值，否则 IPRDY 标志必须手动设置）

ISO：同步传输使能。

0: 端点被配置为批量/中断传输方式

1: 端点被配置为同步传输方式

MODE: 端点方向选择位。

0: 选择端点方向为 OUT

1: 选择端点方向为 IN

ENDMA: IN 端点的 DMA 控制

0: 禁止 IN 端点的 DMA 请求

1: 使能 IN 端点的 DMA 请求

FCDT: 强制 DATA0/DATA1 数据切换设置。

0: 端点数据只在发送完一个数据包后且收到 ACK 时切换。

1: 端点数据在每发送完一个数据包后被强制切换, 不管是否收到 ACK。

需要获取/设置此信息时, 首先必须使用 INDEX 来选择目标端点 1~5

19.2.15 OUT 端点的最大数据包大小 (OUTMAXP)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OUTMAXP	13H	OUTMAXP[7:0]							

OUTMAXP[7:0]: 设置 USB 的 OUT 端点最大数据包的大小

需要获取/设置此信息时, 首先必须使用 INDEX 来选择目标端点 1~5

19.2.16 OUT 端点控制状态寄存器 1 (OUTCSR1)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OUTCSR1	14H	CLRDT	STSTL	SDSTL	FLUSH	DATERR	OVRUN	FIFOFUL	OPRDY

CLRDT (Clear Data Toggle): 复位 OUT 数据切换位。

当 OUT 端点由于被重新配置或者被 STALL 而需要将数据切换位复位到“0”时, 软件需要向此数据位写“1”。

STSTL (Sent Stall): STALL 信号发送完成标志。

当 STALL 信号被发送完成后, 硬件会将该位置“1”。该标志必须用软件清“0”。

SDSTL (Send Stall): STALL 信号发送请求位。

软件应向该位写“1”以产生 STALL 信号作为对一个 OUT 令牌的应答。软件应向该位写“0”以结束 STALL 信号。该位对 ISO 方式没有影响。

FLUSH (FIFO Flush): 清除 OUT 端点的 FIFO 的下一个数据包。

向该位写“1”将从 OUT 端点 FIFO 中清除下一个数据包。FIFO 指针被复位, OPRDY 位被清“0”。

如果 FIFO 中包含多个数据包, 软件必须对每个数据包向 FLUSH 写“1”。当 FIFO 清空完成后, 硬件将 FLUSH 位清“0”。

OVRUN (Data Overrun): 数据溢出。

当一个输入数据包不能被装入到 OUT 端点 FIFO 时, 该位被硬件置“1”。该位只在 ISO 方式有效。该位必须用软件清“0”。

0: 无数据溢出

1: 自该标志最后一次被清除以来, 因 FIFO 已满导致数据包丢失

FIFOFUL (FIFO Full): OUT 端点的 FIFO 数据满标志。

0: OUT 端点的 FIFO 未滿

1: OUT 端点的 FIFO 已滿

OPRDY (OUT Packet Ready): OUT 数据包接收完成标志。

当有数据包可用时硬件将该位置“1”。软件应在将每个数据包从 OUT 端点 FIFO 卸载后将该位清“0”。

需要获取/设置此信息时, 首先必须使用 INDEX 来选择目标端点 1~5

19.2.17 OUT 端点控制状态寄存器 2 (OUTCSR2)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OUTCSR2	15H	AUTOCLR	ISO	ENDMA	DMAMD	-	-	-	-

AUTOCLR: 自动清除 OPRDY 标志控制位。

0: 禁止自动清除 OPRDY 标志

1: 使能自动清除 OPRDY (必须从 OUT FIFO 中下载的数据达到 OUTMAXP 所设置的值, 否则 OPRDY 标志必须手动清除)

ISO: 同步传输使能。

0: 端点被配置为批量/中断传输方式

1: 端点被配置为同步传输方式

ENDMA: OUT 端点的 DMA 控制

0: 禁止 OUT 端点的 DMA 请求

1: 使能 OUT 端点的 DMA 请求

DMAMD: 设置 OUT 端点的 DMA 模式

需要获取/设置此信息时, 首先必须使用 INDEX 来选择目标端点 1~5

19.2.18 USB 端点 0 的 OUT 长度 (COUNT0)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
COUNT0	16H	-	OUTCNT0[6:0]						

OUTCNT0[6:0]: 端点 0 的 OUT 字节长度

COUNT0 专用于保存端点 0 最后接收到的 OUT 数据包的数据长度 (由于端点 0 数据包最长只能为 64 字节, 所以只需要 7 位)。此长度值只在端点 0 的 OPRDY 位为“1”时才有效。

需要获取此长度信息时, 首先必须使用 INDEX 来选择目标端点 0

19.2.19 USB 端点的 OUT 长度 (OUTCOUNTn)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OUTCOUNT1	16H	OUTCNT[7:0]							
OUTCOUNT2	17H	-	-	-	-	-	OUTCNT[10:8]		

OUTCNT[10:0]: 端点的 OUT 字节长度

OUTCOUNT1 和 OUTCOUNT2 联合组成一个 11 位的数, 保存最后 OUT 数据包的数据长度, 适用于端点 1~5。此长度值只在端点 1~5 的 OPRDY 位为“1”时才有效。

需要获取此长度信息时, 首先必须使用 INDEX 来选择目标端点 1~5

19.2.20 USB 端点的 FIFO 数据访问寄存器 (FIFO_n)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
FIFO0	20H	FIFO0[7:0]							
FIFO1	21H	FIFO1[7:0]							
FIFO2	22H	FIFO2[7:0]							
FIFO3	23H	FIFO3[7:0]							
FIFO4	24H	FIFO4[7:0]							
FIFO5	25H	FIFO5[7:0]							

FIFO_n[7:0]: USB 各个端点的 IN/OUT 数据间接访问寄存器

19.2.21 USB 跟踪控制寄存器 (UTRKCTL)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UTRKCTL	30H	FTM1	FTM0	INTV[1:0]		ENST5	RES[2:0]		

FTM1: HFOSC 微调控制位

0: 硬件使用粗调和微调为调整频率

1: 硬件仅使用微调位调整 HFOSC

FTM0: FTM1 为 0 时有效

0: UTRK 使用全部 128 级微调

1: UTRK 禁止使用最大及最小 12 级之微调

INTV[1:0]: 选择 UTRK 更新周期

INTV[1:0]	周期
00	2ms
01	4ms
10	8ms
11	16ms

ENST5: 使能加 5 阶和减 5 阶

0: 校准上下限为 10%

1: 校准上下限为 20%

RES[2:0]: UTRK 自动调节分辨率设置

RES[2:0]	分辨率	调节值
000	8	0.067%
001	12	0.100%
010	16	0.133%
011	24	0.200%
100	28	0.233%
101	32	0.267%
110	48	0.4%
111	64	0.5%

19.2.22 USB 跟踪状态寄存器 (UTRKSTS)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
UTRKSTS	31H	INTVCNT[3:0]				STS[1:0]		TST_UTRK	UTRK_RDY

INTVCNT[3:0]: 内部计数状态

STS[1:0]: UTRK 状态

STS[1:0]	状态
00	INC 5
01	DEC 5
10	INC 1
11	DEC 1

TST_UTRK: UTRK 测试模式控制

0: 禁止 UTRK 测试模式

1: 使能 UTRK 测试模式

UTRK_RDY: UTRK 校准完成状态位

19.3 范例程序

19.3.1 HID 人机接口设备范例

C 语言代码

//测试工作频率为11.0592MHz

#include "stc16.h" //头文件见附录
#include "intrins.h"

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

#define FADDR 0
#define POWER 1
#define INTRINI 2
#define EP5INIF 0x20
#define EP4INIF 0x10
#define EP3INIF 0x08
#define EP2INIF 0x04
#define EP1INIF 0x02
#define EP0IF 0x01
#define INTROUTI 4
#define EP5OUTIF 0x20
#define EP4OUTIF 0x10
#define EP3OUTIF 0x08
#define EP2OUTIF 0x04
#define EP1OUTIF 0x02
#define INTRUSB 6
#define SOFIF 0x08
#define RSTIF 0x04
#define RSUIF 0x02
#define SUSIF 0x01
#define INTRINIE 7
#define EP5INIE 0x20
#define EP4INIE 0x10
#define EP3INIE 0x08
#define EP2INIE 0x04
#define EP1INIE 0x02
#define EP0IE 0x01
#define INTROUTIE 9
#define EP5OUTIE 0x20
#define EP4OUTIE 0x10
#define EP3OUTIE 0x08
#define EP2OUTIE 0x04
#define EP1OUTIE 0x02
#define INTRUSBE 11
#define SOFIE 0x08
#define RSTIE 0x04
#define RSUIE 0x02
#define SUSIE 0x01
#define FRAME1 12
#define FRAME2 13
#define INDEX 14
#define INMAXP 16

```

#define CSR0          17
#define SSUEND        0x80
#define SOPRDY        0x40
#define SDSTL         0x20
#define SUEND         0x10
#define DATEND        0x08
#define STSTL         0x04
#define IPRDY         0x02
#define OPRDY         0x01
#define INCSR1        17
#define INCLRDT        0x40
#define INSTSTL       0x20
#define INSDSTL       0x10
#define INFLUSH       0x08
#define INUNDRUN      0x04
#define INFIFONE      0x02
#define INIPRDY       0x01
#define INCSR2        18
#define INAUTOSET     0x80
#define INISO         0x40
#define INMODEIN      0x20
#define INMODEOUT     0x00
#define INENDMA       0x10
#define INFCDT        0x08
#define OUTMAXP       19
#define OUTCSR1       20
#define OUTCLRDT      0x80
#define OUTSTSTL      0x40
#define OUTSDSTL      0x20
#define OUTFLUSH      0x10
#define OUTDATERR     0x08
#define OUTOVERRUN    0x04
#define UTFIFOFUL     0x02
#define OUTOPRDY      0x01
#define OUTCSR2       21
#define OUTAUTOCLR    0x80
#define OUTISO        0x40
#define OUTENDMA      0x20
#define OUTDMAMD      0x10
#define COUNT0        22
#define OUTCOUNT1    22
#define OUTCOUNT2    23
#define FIFO0         32
#define FIFO1         33
#define FIFO2         34
#define FIFO3         35
#define FIFO4         36
#define FIFO5         37
#define UTRKCTL       48
#define UTRKSTS       49

#define EPIDLE        0
#define EPSTATUS      1
#define EPDATAIN      2
#define EPDATAOUT     3
#define EPSTALL       -1

#define GET_STATUS    0x00
#define CLEAR_FEATURE 0x01

```

```

#define SET_FEATURE           0x03
#define SET_ADDRESS          0x05
#define GET_DESCRIPTOR        0x06
#define SET_DESCRIPTOR        0x07
#define GET_CONFIG            0x08
#define SET_CONFIG            0x09
#define GET_INTERFACE         0x0A
#define SET_INTERFACE         0x0B
#define SYNCH_FRAME           0x0C

#define GET_REPORT            0x01
#define GET_IDLE              0x02
#define GET_PROTOCOL          0x03
#define SET_REPORT            0x09
#define SET_IDLE              0x0A
#define SET_PROTOCOL          0x0B

#define DESC_DEVICE           0x01
#define DESC_CONFIG           0x02
#define DESC_STRING           0x03
#define DESC_HIDREPORT        0x22

#define STANDARD_REQUEST      0x00
#define CLASS_REQUEST         0x20
#define VENDOR_REQUEST        0x40
#define REQUEST_MASK          0x60

```

typedef struct

```

{
    BYTE    bmRequestType;
    BYTE    bRequest;
    BYTE    wValueL;
    BYTE    wValueH;
    BYTE    wIndexL;
    BYTE    wIndexH;
    BYTE    wLengthL;
    BYTE    wLengthH;
}SETUP;

```

typedef struct

```

{
    BYTE    bStage;
    WORD    wResidue;
    BYTE    *pData;
}EP0STAGE;

```

```

void UsbInit();
BYTE ReadReg(BYTE addr);
void WriteReg(BYTE addr, BYTE dat);
BYTE ReadFifo(BYTE fifo, BYTE *pdat);
void WriteFifo(BYTE fifo, BYTE *pdat, BYTE cnt);

```

```

char code DEVIDESC[18];
char code CONFIGDESC[41];
char code HIDREPORTDESC[27];
char code LANGIDDESC[4];
char code MANUFACTDESC[8];
char code PRODUCTDESC[30];

```

```
SETUP Setup;
EP0STAGE Ep0Stage;
BYTE xdata HidFreature[64];
BYTE xdata HidInput[64];
BYTE xdata HidOutput[64];

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;

    UsbInit();

    EA = 1;

    while (1);
}

BYTE ReadReg(BYTE addr)
{
    BYTE dat;

    while (USBADR & 0x80);
    USBADR = addr / 0x80;
    while (USBADR & 0x80);
    dat = USBDAT;

    return dat;
}

void WriteReg(BYTE addr, BYTE dat)
{
    while (USBADR & 0x80);
    USBADR = addr & 0x7f;
    USBDAT = dat;
}

BYTE ReadFifo(BYTE fifo, BYTE *pdat)
{
    BYTE cnt;
    BYTE ret;

    ret = cnt = ReadReg(COUNT0);
    while (cnt--)
    {
        *pdat++ = ReadReg(fifo);
    }

    return ret;
}
```

```

}

void WriteFifo(BYTE fifo, BYTE *pdat, BYTE cnt)
{
    while (cnt--)
    {
        WriteReg(fifo, *pdat++);
    }
}

void DelayXns(WORD delayTime)
{
    while( delayTime-- );
}

void UsbInit()
{
    P3M0 = 0x00;
    P3M1 = 0x03;

    P_SW2 /= 0x80;
    PLLCR = (1<<7)/(0<<5)/(1<<3)/(1<<1);           // enable PLL
    DelayXns(100);
    CKSEL = 0x02;                                     // 选择系统时钟源为内部 pll 输出
    CLKDIV = 0;
    P_SW2 &= ~0x80;
    USBCLK = 0x90;
    USBCON = 0x90;

    WriteReg(FADDR, 0x00);
    WriteReg(POWER, 0x08);
    WriteReg(INTRINIE, 0x3f);
    WriteReg(INTROUTIE, 0x3f);
    WriteReg(INTRUSBE, 0x00);
    WriteReg(POWER, 0x01);

    Ep0Stage.bStage = EPIDLE;
}

void usb_isr() interrupt 22
{
    BYTE intrusb;
    BYTE intrin;
    BYTE introut;
    BYTE csr;
    BYTE cnt;
    WORD len;

    intrusb = ReadReg(INTRUSB);
    intrin = ReadReg(INTRIN1);
    introut = ReadReg(INTROUT1);

    if (intrusb & RSTIF)
    {
        WriteReg(INDEX, 1);
        WriteReg(INCSRI, INCLRDT);
        WriteReg(INDEX, 1);
        WriteReg(OUTCSRI, OUTCLRDT);
        Ep0Stage.bStage = EPIDLE;
    }
}

```

```

}

if (intrin & EP0IF)
{
    WriteReg(INDEX, 0);
    csr = ReadReg(CSR0);
    if (csr & STSTL)
    {
        WriteReg(CSR0, csr & ~STSTL);
        Ep0Stage.bStage = EPIDLE;
    }
    if (csr & SUEND)
    {
        WriteReg(CSR0, csr / SSUEND);
    }

    switch (Ep0Stage.bStage)
    {
    case EPIDLE:
        if (csr & OPRDY)
        {
            Ep0Stage.bStage = EPSTATUS;
            ReadFifo(FIFO0, (BYTE *)&Setup);
            ((BYTE *)&Ep0Stage.wResidue)[0] = Setup.wLengthH;
            ((BYTE *)&Ep0Stage.wResidue)[1] = Setup.wLengthL;
            switch (Setup.bmRequestType & REQUEST_MASK)
            {
            case STANDARD_REQUEST:
                switch (Setup.bRequest)
                {
                case SET_ADDRESS:
                    WriteReg(FADDR, Setup.wValueL);
                    break;
                case SET_CONFIG:
                    WriteReg(INDEX, 1);
                    WriteReg(INCSR2, INMODEIN);
                    WriteReg(INMAXP, 8);
                    WriteReg(INDEX, 1);
                    WriteReg(INCSR2, INMODEOUT);
                    WriteReg(OUTMAXP, 8);
                    WriteReg(INDEX, 0);
                    break;
                case GET_DESCRIPTOR:
                    Ep0Stage.bStage = EPDATAIN;
                    switch (Setup.wValueH)
                    {
                    case DESC_DEVICE:
                        Ep0Stage.pData = DEVIDDESC;
                        len = sizeof(DEVIDDESC);
                        break;
                    case DESC_CONFIG:
                        Ep0Stage.pData = CONFIGDESC;
                        len = sizeof(CONFIGDESC);
                        break;
                    case DESC_STRING:
                        switch (Setup.wValueL)
                        {
                        case 0:
                            Ep0Stage.pData = LANGIDDESC;

```

```

        len = sizeof(LANGIDDESC);
        break;
    case 1:
        Ep0Stage.pData = MANUFACTDESC;
        len = sizeof(MANUFACTDESC);
        break;
    case 2:
        Ep0Stage.pData = PRODUCTDESC;
        len = sizeof(PRODUCTDESC);
        break;
    default:
        Ep0Stage.bStage = EPSTALL;
        break;
    }
    break;
case DESC_HIDREPORT:
    Ep0Stage.pData = HIDREPORTDESC;
    len = sizeof(HIDREPORTDESC);
    break;
default:
    Ep0Stage.bStage = EPSTALL;
    break;
}
if (len < Ep0Stage.wResidue)
{
    Ep0Stage.wResidue = len;
}
break;
default:
    Ep0Stage.bStage = EPSTALL;
    break;
}
break;
case CLASS_REQUEST:
    switch (Setup.bRequest)
    {
    case GET_REPORT:
        Ep0Stage.pData = HidFreature;
        Ep0Stage.bStage = EPDATAIN;
        break;
    case SET_REPORT:
        Ep0Stage.pData = HidFreature;
        Ep0Stage.bStage = EPDATAOUT;
        break;
    case SET_IDLE:
        break;
    case GET_IDLE:
    case GET_PROTOCOL:
    case SET_PROTOCOL:
    default:
        Ep0Stage.bStage = EPSTALL;
        break;
    }
    break;
default:
    Ep0Stage.bStage = EPSTALL;
    break;
}

```

```

switch (Ep0Stage.bStage)
{
case EPDATAIN:
    WriteReg(CSR0, SOPRDY);
    goto L_Ep0SendData;
    break;
case EPDATAOUT:
    WriteReg(CSR0, SOPRDY);
    break;
case EPSTATUS:
    WriteReg(CSR0, SOPRDY / DATEND);
    Ep0Stage.bStage= EPIDLE;
    break;
case EPSTALL:
    WriteReg(CSR0, SOPRDY / SDSTL);
    Ep0Stage.bStage= EPIDLE;
    break;
}
}
break;
case EPDATAIN:
    if (!(csr & IPRDY))
    {
L_Ep0SendData:
        cnt = Ep0Stage.wResidue > 64 ? 64 : Ep0Stage.wResidue;
        WriteFifo(FIFO0, Ep0Stage.pData, cnt);
        Ep0Stage.wResidue -= cnt;
        Ep0Stage.pData += cnt;
        if (Ep0Stage.wResidue == 0)
        {
            WriteReg(CSR0, IPRDY / DATEND);
            Ep0Stage.bStage= EPIDLE;
        }
        else
        {
            WriteReg(CSR0, IPRDY);
        }
    }
    break;
case EPDATAOUT:
    if (csr & OPRDY)
    {
        cnt = ReadFifo(FIFO0, Ep0Stage.pData);
        Ep0Stage.wResidue -= cnt;
        Ep0Stage.pData += cnt;
        if (Ep0Stage.wResidue == 0)
        {
            WriteReg(CSR0, SOPRDY / DATEND);
            Ep0Stage.bStage= EPIDLE;
        }
        else
        {
            WriteReg(CSR0, SOPRDY);
        }
    }
    break;
}
}

```



```

if (intrin & EPIINIF)
{
    WriteReg(INDEX, 1);
    csr = ReadReg(INCSR1);
    if (csr & INSTSTL)
    {
        WriteReg(INCSR1, INCLRDT);
    }
    if (csr & INUNDRUN)
    {
        WriteReg(INCSR1, 0);
    }
}

if (introut & EPIOUTIF)
{
    WriteReg(INDEX, 1);
    csr = ReadReg(OUTCSR1);
    if (csr & OUTSTSTL)
    {
        WriteReg(OUTCSR1, OUTCLRDT);
    }
    if (csr & OUTOPRDY)
    {
        ReadFifo(FIFO1, HidOutput);
        WriteReg(OUTCSR1, 0);

        WriteReg(INDEX, 1);
        WriteFifo(FIFO1, HidOutput, 64);
        WriteReg(INCSR1, INIPRDY);
    }
}
}

char code DEVICEDESC[18] =
{
    0x12, //bLength(18);
    0x01, //bDescriptorType(Device);
    0x00,0x02, //bcdUSB(2.00);
    0x00, //bDeviceClass(0);
    0x00, //bDeviceSubClass(0);
    0x00, //bDeviceProtocol(0);
    0x40, //bMaxPacketSize0(64);
    0x54,0x53, //idVendor(5354);
    0x80,0x43, //idProduct(4380);
    0x00,0x01, //bcdDevice(1.00);
    0x01, //iManufacturer(1);
    0x02, //iProduct(2);
    0x00, //iSerialNumber(0);
    0x01, //bNumConfigurations(1);
};

char code CONFIGDESC[41] =
{
    0x09, //bLength(9);
    0x02, //bDescriptorType(Configuration);
    0x29,0x00, //wTotalLength(41);
    0x01, //bNumInterfaces(1);
    0x01, //bConfigurationValue(1);

```

```

0x00, //iConfiguration(0);
0x80, //bmAttributes(BUSPower);
0x32, //MaxPower(100mA);

0x09, //bLength(9);
0x04, //bDescriptorType(Interface);
0x00, //bInterfaceNumber(0);
0x00, //bAlternateSetting(0);
0x02, //bNumEndpoints(2);
0x03, //bInterfaceClass(HID);
0x00, //bInterfaceSubClass(0);
0x00, //bInterfaceProtocol(0);
0x00, //iInterface(0);

0x09, //bLength(9);
0x21, //bDescriptorType(HID);
0x01,0x01, //bcdHID(1.01);
0x00, //bCountryCode(0);
0x01, //bNumDescriptors(1);
0x22, //bDescriptorType(HID Report);
0x1b,0x00, //wDescriptorLength(27);

0x07, //bLength(7);
0x05, //bDescriptorType(Endpoint);
0x81, //bEndpointAddress(EndPoint1 as IN);
0x03, //bmAttributes(Interrupt);
0x40,0x00, //wMaxPacketSize(64);
0x01, //bInterval(10ms);

0x07, //bLength(7);
0x05, //bDescriptorType(Endpoint);
0x01, //bEndpointAddress(EndPoint1 as OUT);
0x03, //bmAttributes(Interrupt);
0x40,0x00, //wMaxPacketSize(64);
0x01, //bInterval(10ms);
};

char code HIDREPORTDESC[27] =
{
    0x05,0x0c, //USAGE_PAGE(Consumer);
    0x09,0x01, //USAGE(Consumer Control);
    0xa1,0x01, //COLLECTION(Application);
    0x15,0x00, // LOGICAL_MINIMUM(0);
    0x25,0xff, // LOGICAL_MAXIMUM(255);
    0x75,0x08, // REPORT_SIZE(8);
    0x95,0x40, // REPORT_COUNT(64);
    0x09,0x01, // USAGE(Consumer Control);
    0xb1,0x02, // FEATURE(Data,Variable);
    0x09,0x01, // USAGE(Consumer Control);
    0x81,0x02, // INPUT(Data,Variable);
    0x09,0x01, // USAGE(Consumer Control);
    0x91,0x02, // OUTPUT(Data,Variable);
    0xc0, //END_COLLECTION;
};

char code LANGIDDESC[4] =
{
    0x04,0x03,
    0x09,0x04,

```

```
};
```

```
char code MANUFACTDESC[8] =
```

```
{
```

```
    0x08,0x03,
```

```
    'S',0,
```

```
    'T',0,
```

```
    'C',0,
```

```
};
```

```
char code PRODUCTDESC[30] =
```

```
{
```

```
    0x1e,0x03,
```

```
    'S',0,
```

```
    'T',0,
```

```
    'C',0,
```

```
    ' ',0,
```

```
    'U',0,
```

```
    'S',0,
```

```
    'B',0,
```

```
    ' ',0,
```

```
    'D',0,
```

```
    'e',0,
```

```
    'v',0,
```

```
    'i',0,
```

```
    'c',0,
```

```
    'e',0,
```

```
};
```

20 CAN 总线

STC16F 系列单片机内部集成 CAN 总线功能单元，支持 CAN 2.0 协议。

主要功能如下：

- 标准帧和扩展帧信息的接收和传送
- 64 字节的接收 FIFO
- 在标准和扩展格式中都有单/双验收滤波器
- 发送、接收的错误计数器
- 总线错误分析

20.1 CAN 功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	9AH	S1_S[1:0]		CAN_S[1:0]		SPI_S[1:0]		LIN_S[1:0]	

CAN_S[1:0]: CAN 功能脚选择位

CAN_S[1:0]	CAN_RX	CAN_TX
00	P0.0	P0.1
01	P5.0	P5.1
10	P4.2	P4.5
11	P7.0	P7.1

20.2 CAN 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CANICR	CANBUS 中断控制寄存器	A1H	-	-	-	-	PCANH	CANIF	ECAN	PCANL	0000,0000
CANAR	CANBUS 地址寄存器	A2H									0000,0000
CANDR	CANBUS 数据寄存器	A3H									0000,0000
AUXR2	辅助寄存器 2	94H	-	P40ADR	ADIBF	TXLNRX	-	-	CANEN	LINEN	0000,0000

20.2.1 辅助寄存器 2 (AUXR2)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR2	94H	-	P40ADR	ADIBF	TXLNRX	-	-	CANEN	LINEN

CANEN: CAN 总线使能控制位

- 0: 关闭 CAN 功能
- 1: 使能 CAN 功能

20.2.2 CAN 总线中断控制寄存器 (CANICR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CANICR	A1H	-	-	-	-	PCANH	CANIF	ECAN	PCANL

ECAN: CAN 总线中断使能控制位

0: 关闭 CAN 中断

1: 使能 CAN 中断

CANIF: CAN 总线中断请求标志位, 需软件清零。

PCANH, PCANL: CAN 中断优先级控制位

PCANH	PCANL	优先级
0	0	0 (最低)
0	1	1
1	0	2
1	1	3 (最高)

20.2.3 CAN 总线地址寄存器 (CANAR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CANAR	A2H								

20.2.4 CAN 总线数据寄存器 (CANDR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CANDR	A3H								

20.3 CAN 内部功能寄存器

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
18H	ECC	RXERR	TXERR	ALC				
10H	ACR0	ACR1	ACR2	ACR3	AMR0	AMR1	AMR2	AMR3
08H	TXBUF0	TXBUF1	TXBUF2	TXBUF	RXBUF0	RXBUF1	RXBUF2	RXBUF3
00H	MR	CMR	SR	ISR	IMR	RMC	BTR0	BTR1

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
MR	0x00						RM	LOM	AFM
CMR	0x01						TR	AT	
SR	0x02	RBS	DSO	TBS		RS	TS	ES	BS
ISR/IACK	0x03		ALI	EWI	EPI	RI	TI	BEI	DOI
IMR	0x04		ALIM	EWIM	EPIM	RIM	TIM	BEIM	DOIM
RMC	0x05				RMC4	RMC3	RMC2	RMC1	RMC0
BTR0	0x06	SJW.1	SJW.0	BRP.5	BRP.4	BRP.3	BRP.2	BRP.1	BRP.0
BTR1	0x07	SAM	TSG2.2	TSG2.1	TSG2.0	TSG1.3	TSG1.2	TSG1.1	TSG1.0
TXBUF0	0x08	frame byte n							
TXBUF1	0x09	frame byte n+1							
TXBUF2	0x0A	frame byte n+2							
TXBUF3	0x0B	frame byte n+3							
RXBUF0	0x0C	frame byte n							

RXBUF1	0x0D	frame byte n+1							
RXBUF2	0x0E	frame byte n+2							
RXBUF3	0x0F	frame byte n+3							
ACR0	0x10	ACR0							
ACR1	0x11	ACR1							
ACR2	0x12	ACR2							
ACR3	0x13	ACR3							
AMR0	0x14	AMR0							
AMR1	0x15	AMR1							
AMR2	0x16	AMR2							
AMR3	0x17	AMR3							
ECC	0x18	RXWRN	TXWRN	EDIR	ACKER	FRMER	CRCER	STFER	BER
RXERR	0x19	RXERR							
TXERR	0x1A	TXERR							
ALC	0x1B				ALC.4	ALC.3	ALC.2	ALC.1	ALC.0

20.3.1 CAN 模式寄存器 (MR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
MR	0x00	X	X	X	X	X	RM	LOM	AFM

RM:CAN 模块的 RESET MODE

0: 关闭 RESET MODE

1: 使能 RESET MODE

LOM:CAN 模块的 LISTEN ONLY MODE

0: 关闭 LISTEN ONLY MODE

1: 使能 LISTEN ONLY MODE

AFM:CAN 模块的接收滤波选择 (参见 ACR 寄存器描述)

0: 接受滤波器采用双滤波设置

1: 接受滤波器采用单滤波设置

20.3.2 CAN 命令寄存器 (CMR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMR	0x01	X	X	X	X	X	TR	AT	X

TR:CAN 模块发送请求

0:

1: 发起一次帧传输

AT:CAN 模块传输终止

0:

1: 终止当前的帧传输

20.3.3 CAN 状态寄存器 (SR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SR	0x02	RBS	DSO	TBS	X	RS	TS	ES	BS

RBS:接收 BUFFER 状态

0: 接收 BUFFER 无数据帧

1: 接收 BUFFER 有数据帧

DSO:接收 FIFO 溢出循环标志

0: 接收 FIFO 没有溢出循环产生

1: 接收 FIFO 有溢出循环产生

TBS:CAN 模块发送 BUFFER 状态

0: 发送 BUFFER 禁止, CPU 不可对 BUFFER 进行写操作

1: 发送 BUFFER 空闲, CPU 可对 BUFFER 进行写操作

RS:CAN 模块接收状态

0: CAN 模块接收空闲

1: CAN 模块正在接收数据帧

TS:CAN 模块发送状态

0: CAN 模块发送空闲

1: CAN 模块正在发送数据帧

ES:CAN 模块错误状态

0: CAN 模块错误寄存器值未达到 96

1: CAN 模块至少有一个错误寄存器的值达到或超过了 96

BS:CAN 模块 BUS-OFF 状态

0: CAN 模块不在 BUS-OFF 状态

1: CAN 模块在 BUS-OFF 状态当 CAN 控制器发生错误的次数超过 255 次, 就会触发 BUS-OFF 错误。一般发生 BUS-OFF 的条件是 CAN 总线受周围环境干扰, 导致 CAN 发送端发送到总线的数据被 BUS 总线判断为异常, 但异常的次数超过 255 次, BUS 总线自动设置为 BUS-OFF 状态, 此时总线处于忙的状态, 数据无法发送, 也无法接收。

20.3.4 CAN 中断/应答寄存器 (ISR/IACK)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ISR/IACK	0x03	X	ALI	EWI	EPI	RI	TI	BEI	DOI

ALI:仲裁丢失中断

0:

1: 仲裁丢失, 写 1 清零

EWI:错误警告中断

0:

1: 当 SR 寄存器中的 ES 或者 BS 值为 1 时, 该位置位。写 1 清零。

EPI:CAN 模块被动错误中断

0:

1: 当 CAN 错误寄存器操作被动错误计数值时, 该位置位。写 1 清零。

RI:CAN 模块接收中断

0:

1: CAN 模块接收 BUFFER 中存在数据帧, 用户需要对 RI 写 1, 以减少接收信息计数器 (RMC) 值。

TI:CAN 模块发送中断

0:

1: CAN 模块数据帧发送完成。用户需要对 TI 写 1, 复位发送 BUFFER 的写指针。

BEI:CAN 模块总线错误中断

0:

1: CAN 模块在接收或者发送过程中产生了总线错误

DOI:CAN 模块接收溢出中断

0:

1: CAN 模块接收 FIFO 溢出

20.3.5 CAN 中断寄存器 (IMR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IMR	0x04	X	ALIM	EWIM	EPIM	RIM	TIM	BEIM	DOIM

ALIM:仲裁丢失中断

0: 仲裁丢失中断屏蔽

1: 仲裁丢失中断开启

EWIM:错误警告中断

0: 错误警告中断屏蔽

1: 错误警告中断开启

EPIM:CAN 模块被动错误中断

0: CAN 模块被动错误中断屏蔽

1: CAN 模块被动错误中断开启

RI:CAN 模块接收中断

0: CAN 模块接收中断屏蔽

1: CAN 模块接收中断开启

TI:CAN 模块发送中断

0: CAN 模块发送中断屏蔽

1: CAN 模块发送中断开启

BEI:CAN 模块总线错误中断

0: CAN 模块总线错误中断屏蔽

1: CAN 模块总线错误中断开启

DOI:CAN 模块接收溢出中断

0: CAN 模块接收溢出中断屏蔽

1: CAN 模块接收溢出中断开启

20.3.6 CAN 数据帧接收计数器 (RMC)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
RMC	0x05	X	X	X	RMC4	RMC3	RMC2	RMC1	RMC0

RMC:数据帧接收计数器

20.3.7 CAN 总线时钟寄存器 0 (BTR0)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
BTR0	0x06	SJW.1	SJW.0	BRP.5	BRP.4	BRP.3	BRP.2	BRP.1	BRP.0

BRP:CAN 波特率分频系数

BRP= BTR0[5:0]+1; CAN 模块内部时钟 $t_q = t_{CLK} * BRP$; $t_{CLK} = 1 / f_{XTAL}$ (主频 2 分频)

SJW: :重新同步跳跃宽度

$SJW = SJW.1 * 2 + SJW.0$

20.3.8 CAN 总线时钟寄存器 1 (BTR1)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
BTR1	0x07	SAM	TSG2.2	TSG2.1	TSG2.0	TSG1.3	TSG1.2	TSG1.1	TSG1.0

TSG1:同步采样段 1

TSG2:同步采样段 2

SAM:总线电平采样次数

0: 总线电平采样 1 次

1: 总线电平采样 3 次

CAN 波特率=1/正常时间位

正常时间位= (1+ (TSG1+1) + (TSG2+1)) * tq

20.3.9 CAN 总线数据帧发送缓存 (TXBUF_n)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TXBUF0	0x08	Frame byte n							
TXBUF1	0x09	Frame byte n+1							
TXBUF2	0x0A	Frame byte n+2							
TXBUF3	0x0B	Frame byte n+3							

发送 BUFFER 包含 4 个寄存器: TXBUF0, TXBUF1, TXBUF2, TXBUF3。

每当 TXBUF3 寄存器被写的时候, BUFFER 指针自动加 1, TXBUF0, TXBUF1, TXBUF2, TXBUF3, 被写入 BUFFER。

20.3.10 CAN 总线数据帧接收缓存 (RXBUF_n)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
RXBUF0	0x0C	Frame byte n							
RXBUF1	0x0D	Frame byte n+1							
RXBUF2	0x0E	Frame byte n+2							
RXBUF3	0x0F	Frame byte n+3							

接收 BUFFER 包含 4 个寄存器: RXBUF0, RXBUF1, RXBUF2, RXBUF3。

每当 RXBUF3 寄存器被写的时候, BUFFER 指针自动加 1, RXBUF0, RXBUF1, RXBUF2, RXBUF3, 被写入 BUFFER。一帧 CAN 的数据最长是 16 个 BYTE, 所以接收一帧数据需要循环读取 RXBUF 四次。CAN 模块的 RXFIFO 是一个 64BYTE 的 FIFO, 在数据量为 1 个 BYTE 的时候, 最多能存储 21 帧数据。数据 8 个 BYTE 的时候, 最多能存储 5 帧数据。接收帧的数量可以读取 RMC(RECEIVE MESSAGE COUNTER) 寄存器获得。

CAN 总线验收滤波器

在验收滤波器的帮助下 CAN 控制器能够允许 RXFIFO 只接收同识别码和验收滤波器中预设值相一致的信息验收滤波器通过验收代码寄存器 ACR 和验收屏蔽寄存器 AMR 来定义。

20.3.11 CAN 总线验收代码寄存器 (ACR_n)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ACR0	0x10	ACR0							
ACR1	0x11	ACR1							
ACR2	0x12	ACR2							
ACR3	0x13	ACR3							

20.3.12 CAN 总线验收屏蔽寄存器 (AMRn)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AMR0	0x14	AMR0							
AMR1	0x15	AMR1							
AMR2	0x16	AMR2							
AMR3	0x17	AMR3							

滤波的方式有两种，由模式寄存器中的 AFM (MR.0) 位选择：单滤波器模式 (AFM 位是 1)、双滤波器模式 (AFM 位是 0)。

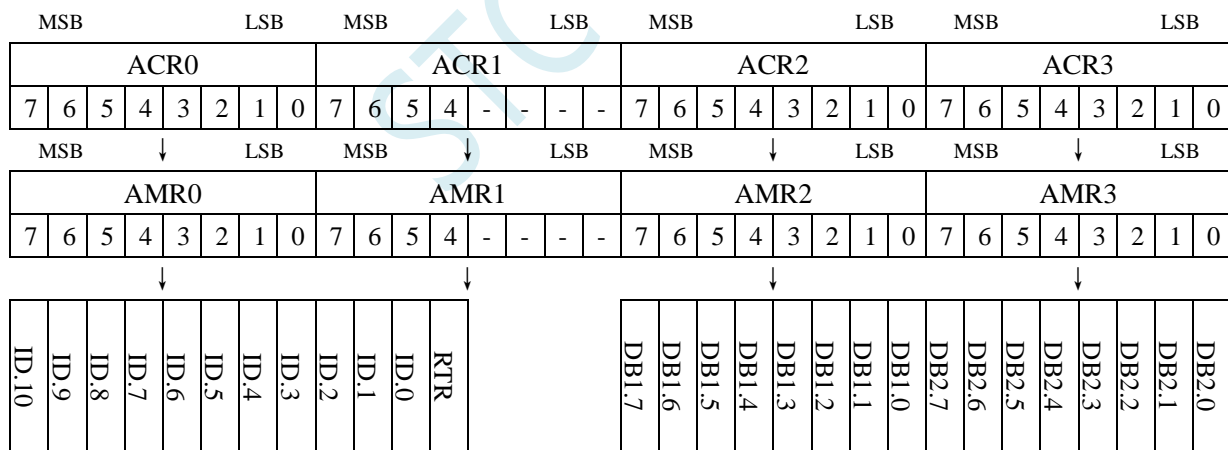
滤波的规则是：每一位验收屏蔽分别对应每一位验收代码，当该位验收屏蔽位为“1”的时候（即设为无关），接收的相应帧 ID 位无论是否和相应的验收代码位相同均会表示为接收；当验收屏蔽位为“0”的时候（即设为相关），只有相应的帧 ID 位和相应的验收代码位值相同的情况才会表示为接收。

只有在所有的位都表示为接收的时候，CAN 控制器才会接收该报文。

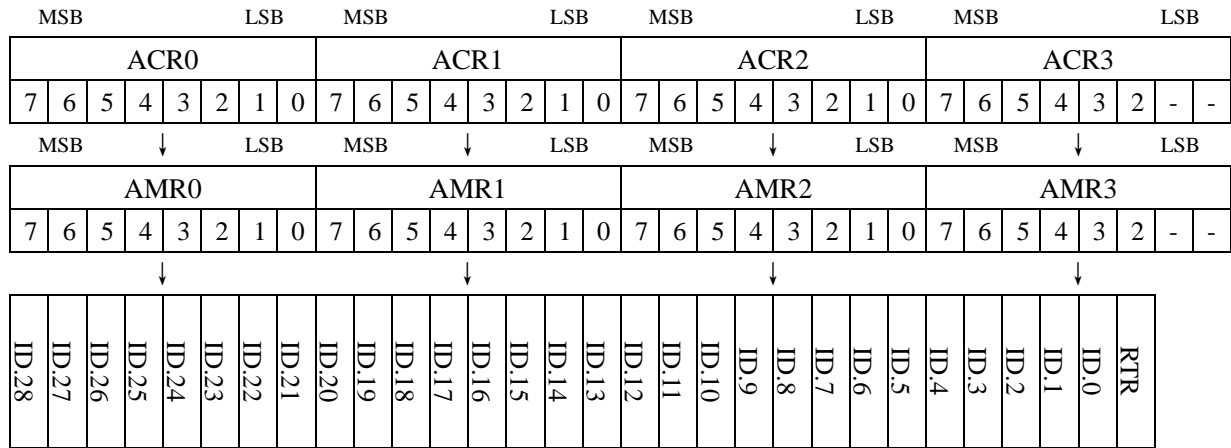
(1) 单滤波器的配置

这种滤波器配置定义了一个长滤波器（4 字节、32 位），由 4 个验收码寄存器和 4 个验收屏蔽寄存器组成的验收滤波器，滤波器字节和信息字节之间位的对应关系取决于当前接收帧格式。接收 CAN 标准帧单滤波器配置：对于标准帧，11 位标识符、RTR 位、数据场前两个字节参与滤波；对于参与滤波的数据，所有 AMR 为 0 的位所对应的 ACR 位和参与滤波数据的对应位必须相同才算验收通过；如果由于置位 RTR=1 位而没有数据字节，或因为设置相应的数据长度代码而没有或只有一个数据字节信息，报文也会被接收。对于一个成功接收的报文，所有单个位在滤波器中的比较结果都必须为“接受”；注意 AMR1 和 ACR1 的低四位是不用的，为了和将来的产品兼容，这些位可通过设置 AMR1.3、AMR1.2、AMR1.1 和 AMR1.0 为 1 而定为“不影响”。

接收 CAN 标准帧时单滤波器配置：



接收 CAN 扩展帧时单滤波器配置:

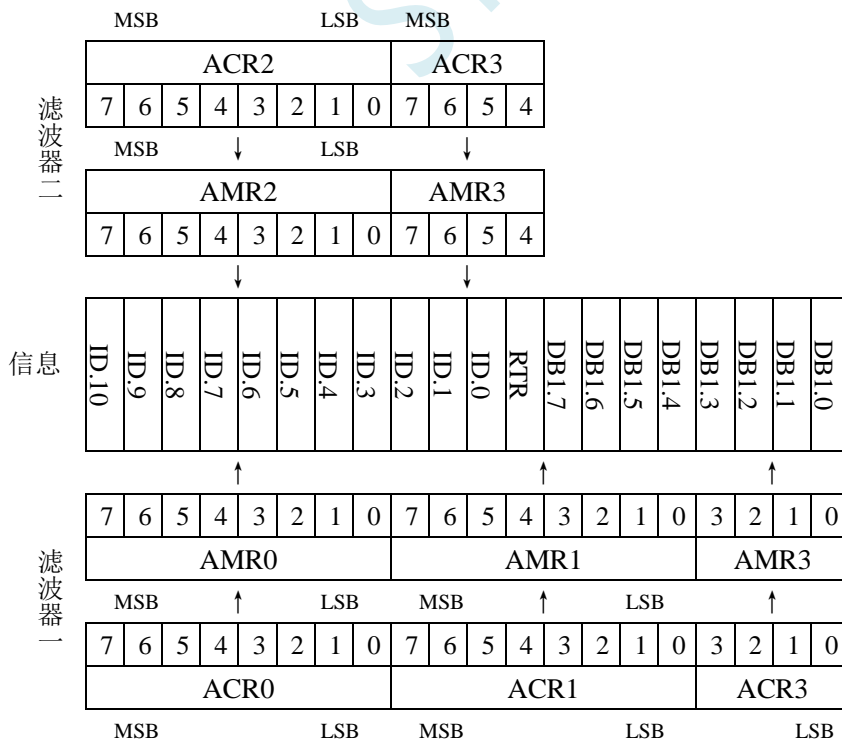


(2) 双滤波器的配置

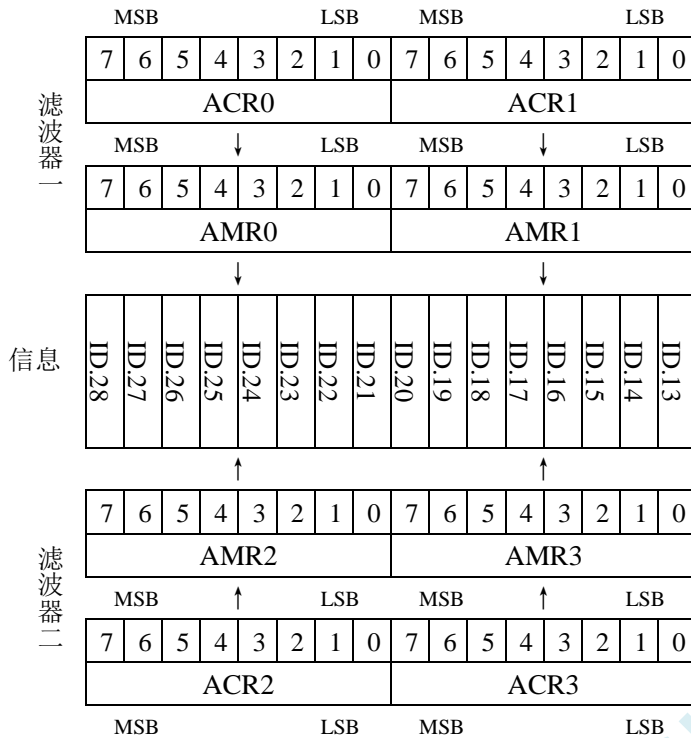
这种配置可以定义两个短滤波器，由 4 个 ACR 和 4 个 AMR 构成两个短滤波器。总线上的信息只要通过任意一个滤波器就被接收。滤波器字节和信息字节之间位的对应关系取决于当前接收的帧格式。

接收 CAN 标准帧双滤波器的配置：如果接收的是标准帧信息，被定义的两个滤波器是不一样的。第一个滤波器由 ACR0、ACR1、AMR0、AMR1 以及 ACR3、AMR3 低 4 位组成，11 位标识符、RTR 位和数据场第 1 字节参与滤波；第二个滤波器由 ACR2、AMR2 以及 ACR3、AMR3 高 4 位组成，11 位标识符和 RTR 位参与滤波。为了成功接收信息，在所有单个位的比较时，应至少有一个滤波器表示接受。RTR 位置为“1”或数据长度代码是“0”，表示没有数据字节存在；只要从开始到 RTR 位的部分都被表示接收，信息就可以通过滤波器 1。如果没有数据字节向滤波器请求过滤，AMR1 和 AMR3 的低 4 位必须被置为“1”，即“不影响”。此时，两个滤波器的识别工作都是验证包括 RTR 位在外的整个标准识别码。

接收 CAN 标准帧时双滤波器配置:



接收 CAN 扩展帧时双滤波器配置:



20.3.13 CAN 总线错误信息寄存器 (ECC)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ECC	0x18	RXWRN	TXWRN	EDIR	ACKER	FRMER	CRCER	STFER	BER

RXWRN: 当 RXERR 大于等于 96 时该位置位。

TXWRN: 当 TXERR 大于等于 96 时该位置位。

EDIR: 传输错误方向。0: 发送时发生错误。1: 接收时发生错误。

ACKER: ACK 错误。

FRMER: 帧格式错误。

CRCER: CRC 错误。

STFER: 位填充错误。

BER: 位错误。

20.3.14 CAN 总线接收错误计数器 (RXERR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
RXERR	0x19	RXERR							

RXERR: 计数器值代表当前接收错误计数值。该寄存器只读。当 BUS-OFF 事件发生后, 该计数器值由硬件清零。

20.3.15 CAN 总线发送错误计数器 (TXERR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TXERR	0x1A	TXERR							

TXERR: 发送错误计数寄存器反映了发送错误计数器的当前值, 工作模式中, 该寄存器只读, 硬件复位后寄存器被初始化为 0。当 BUS-OFF 时间发生后错误计数器被初始化为 127 来计算总线定义的最小时间 (128 个总线空闲信号)。这段时间里读发送错误计数器将反映出总线关闭恢复的状态。

信息。

20.3.16 CAN 总线仲裁丢失寄存器 (ALC)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ALC	0x1B	X	X	X	ALC.4	ALC.3	ALC.2	ALC.1	ALC.0

ALC: 这个寄存器包括了仲裁丢失的位置的信息。当前仲裁丢失中断被处理（应答）之后，该值会在下一次仲裁丢失时更新。具体数值对应下表：

Bits					数值	描述
ALC4	ALC3	ALC2	ALC1	ALC0		
0	0	0	0	0	0	Arbit lost in ID28/11
0	0	0	0	1	1	Arbit lost in ID27/10
0	0	0	1	0	2	Arbit lost in ID26/9
0	0	0	1	1	3	Arbit lost in ID25/8
0	0	1	0	0	4	Arbit lost in ID24/7
0	0	1	0	1	5	Arbit lost in ID23/6
0	0	1	1	0	6	Arbit lost in ID22/5
0	0	1	1	1	7	Arbit lost in ID21/4
0	1	0	0	0	8	Arbit lost in ID20/3
0	1	0	0	1	9	Arbit lost in ID19/2
0	1	0	1	0	10	Arbit lost in ID18/1
0	1	0	1	1	11	Arbit lost in ID17/0
0	1	1	0	0	12	Arbit lost in SRTR/RTR
0	1	1	0	1	13	Arbit lost in IDE bit
0	1	1	1	0	14	Arbit lost in ID16*
0	1	1	1	1	15	Arbit lost in ID15*
1	0	0	0	0	16	Arbit lost in ID14*
1	0	0	0	1	17	Arbit lost in ID13*
1	0	0	1	0	18	Arbit lost in ID12*
1	0	0	1	1	19	Arbit lost in ID11*
1	0	1	0	0	20	Arbit lost in ID10*
1	0	1	0	1	21	Arbit lost in ID9*
1	0	1	1	0	22	Arbit lost in ID8*
1	0	1	1	1	23	Arbit lost in ID7*
1	1	0	0	0	24	Arbit lost in ID6*
1	1	0	0	1	25	Arbit lost in ID5*
1	1	0	1	0	26	Arbit lost in ID4*
1	1	0	1	1	27	Arbit lost in ID3*
1	1	1	0	0	28	Arbit lost in ID2*
1	1	1	0	1	29	Arbit lost in ID1*
1	1	1	1	0	30	Arbit lost in ID0*
1	1	1	1	1	31	Arbit lost in RTR

20.4 范例程序

20.4.1 CAN 总线帧格式

CAN2.0B 标准帧

CAN 标准帧信息为 11 个字节，包括两部分：信息和数据部分。前 3 个字节为信息部分。

位置	B7	B6	B5	B4	B3	B2	B1	B0
字节 01	FF	RTR	-	-	DLC（数据长度）			
字节 02	CAN ID[10:3]							
字节 03	CAN ID[2:0]			-	-	-	-	-
字节 04	数据 1							
字节 05	数据 2							
字节 06	数据 3							
字节 07	数据 4							
字节 08	数据 5							
字节 09	数据 6							
字节 10	数据 7							
字节 11	数据 8							

字节 1 为帧信息。第 7 位（FF）表示帧格式，在标准帧中，FF=0；第 6 位（RTR）表示帧的类型，RTR=0 表示为数据帧，RTR=1 表示为远程帧；DLC 表示在数据帧时实际的数据长度。

字节 2、3 为报文识别码，11 位有效。

字节 4~11 为数据帧的实际数据，远程帧时无效。

CAN2.0B 扩展帧

CAN 扩展帧信息为 13 个字节，包括两部分，信息和数据部分。前 5 个字节为信息部分。

位置	B7	B6	B5	B4	B3	B2	B1	B0
字节 01	FF	RTR	-	-	DLC（数据长度）			
字节 02	CAN ID[28:21]							
字节 03	CAN ID[20:13]							
字节 04	CAN ID[12:5]							
字节 05	CAN ID[4:0]					-	-	-
字节 06	数据 1							
字节 07	数据 2							
字节 08	数据 3							
字节 09	数据 4							
字节 10	数据 5							
字节 11	数据 6							
字节 12	数据 7							
字节 13	数据 8							

字节 1 为帧信息。第 7 位（FF）表示帧格式，在扩展帧中，FF=1；第 6 位（RTR）表示帧的类型，RTR=0 表示为数据帧，RTR=1 表示为远程帧；DLC 表示在数据帧时实际的数据长度。

字节 2~5 为报文识别码，其高 29 位有效。

字节 6~13 数据帧的实际数据，远程帧时无效。

20.4.2 CAN 总线标准帧收发范例

C 语言代码

//测试工作频率为24MHz

#include "stc16.h" //头文件见附录

#include "intrins.h"

typedef unsigned char u8;

typedef unsigned int u16;

typedef unsigned long u32;

#define MAIN_Fosc 24000000UL

/****** 用户定义宏 ******/

//CAN 总线波特率=Fcclk/((1+(TSG1+1)+(TSG2+1))*(BRP+1)*2)

#define TSG1 2 //0~15

#define TSG2 1 //0~7

#define BRP 3 //0~63

//24000000/((1+3+2)*4*2)=500KHz

#define SJW 1 //重新同步跳跃宽度

/*******/

u16 CAN_ID;

u8 RX_BUF[8];

u8 TX_BUF[8];

void CANInit();

void CanSendMsg(u16 canid, u8 *pdat);

void main(void)

{

WTST = 0;

P0M1 = 0; P0M0 = 0;

//设置为准双向口

P1M1 = 0; P1M0 = 0;

//设置为准双向口

P2M1 = 0; P2M0 = 0;

//设置为准双向口

P3M1 = 0; P3M0 = 0;

//设置为准双向口

P4M1 = 0; P4M0 = 0;

//设置为准双向口

P5M1 = 0; P5M0 = 0;

//设置为准双向口

P6M1 = 0; P6M0 = 0;

//设置为准双向口

P7M1 = 0; P7M0 = 0;

//设置为准双向口

CANInit();

EA = 1;

//打开总中断

CAN_ID = 0x0345;

TX_BUF[0] = 0x11;

TX_BUF[1] = 0x22;

TX_BUF[2] = 0x33;

TX_BUF[3] = 0x44;

TX_BUF[4] = 0x55;

TX_BUF[5] = 0x66;

TX_BUF[6] = 0x77;

TX_BUF[7] = 0x88;

```
while(1)
{
    if(!P32)                                     //按一下P32 口按键 发送一帧固定数据
    {
        CanSendMsg(CAN_ID,TX_BUF);
        while(!P32);                             //防止重发
    }
}

u8 CanReadReg(u8 addr)
{
    u8 dat;
    CANAR = addr;
    dat = CANDR;
    return dat;
}

void CanWriteReg(u8 addr, u8 dat)
{
    CANAR = addr;
    CANDR = dat;
}

void CanReadFifo(u8 *pdat)
{
    pdat[0] = CanReadReg(RX_BUF0);
    pdat[1] = CanReadReg(RX_BUF1);
    pdat[2] = CanReadReg(RX_BUF2);
    pdat[3] = CanReadReg(RX_BUF3);

    pdat[4] = CanReadReg(RX_BUF0);
    pdat[5] = CanReadReg(RX_BUF1);
    pdat[6] = CanReadReg(RX_BUF2);
    pdat[7] = CanReadReg(RX_BUF3);

    pdat[8] = CanReadReg(RX_BUF0);
    pdat[9] = CanReadReg(RX_BUF1);
    pdat[10] = CanReadReg(RX_BUF2);
    pdat[11] = CanReadReg(RX_BUF3);

    pdat[12] = CanReadReg(RX_BUF0);
    pdat[13] = CanReadReg(RX_BUF1);
    pdat[14] = CanReadReg(RX_BUF2);
    pdat[15] = CanReadReg(RX_BUF3);
}

u16 CanReadMsg(u8 *pdat)
{
    u8 i;
    u16 CanID;
    u8 buffer[16];

    CanReadFifo(buffer);
    CanID = ((buffer[1] << 8) + buffer[2]) >> 5;
    for(i=0;i<8;i++)
    {
        pdat[i] = buffer[i+3];
    }
}
```



```

    }
    return CanID;
}

void CanSendMsg(u16 canid, u8 *pdat)
{
    u16 CanID;

    CanID = canid << 5;
    CanWriteReg(TX_BUF0, 0x08); //标准帧, 数据帧, bit3~bit0: 数据长度(DLC)
    CanWriteReg(TX_BUF1, (u8)(CanID >> 8));
    CanWriteReg(TX_BUF2, (u8)CanID);
    CanWriteReg(TX_BUF3, pdat[0]);

    CanWriteReg(TX_BUF0, pdat[1]);
    CanWriteReg(TX_BUF1, pdat[2]);
    CanWriteReg(TX_BUF2, pdat[3]);
    CanWriteReg(TX_BUF3, pdat[4]);

    CanWriteReg(TX_BUF0, pdat[5]);
    CanWriteReg(TX_BUF1, pdat[6]);
    CanWriteReg(TX_BUF2, pdat[7]);

    CanWriteReg(TX_BUF3, 0x00);
    CanWriteReg(CMR, 0x04); //发起一次帧传输
}

void CANSetBaudrate()
{
    CanWriteReg(BTR0, (SJW << 6) + BRP);
    CanWriteReg(BTR1, (TSG2 << 4) + TSG1);
}

void CANInit()
{
    CANSetBaudrate(); //设置波特率

    CanWriteReg(ACR0, 0x00); //总线验收代码寄存器
    CanWriteReg(ACR1, 0x00);
    CanWriteReg(ACR2, 0x00);
    CanWriteReg(ACR3, 0x00);
    CanWriteReg(AMR0, 0xFF); //总线验收屏蔽寄存器
    CanWriteReg(AMR1, 0xFF);
    CanWriteReg(AMR2, 0xFF);
    CanWriteReg(AMR3, 0xFF);

    CanWriteReg(IMR, 0xFF); //中断寄存器
    CanWriteReg(MR, 0x00);

    P_SW1 = 0;
    CANICR = 0x02; //CAN 中断使能
    AUXR2 /= 0x02; //CAN 模块被使能
}

void CANBUS_Interrupt(void) interrupt 28
{
    u8 isr;

    isr = CanReadReg(ISR);
}

```

```

    if((isr & 0x04) == 0x04)
    {
        CANAR = 0x03;
        CANDR = 0x04;                                //CLR FLAG
    }
    if((isr & 0x08) == 0x08)
    {
        CANAR = 0x03;
        CANDR = 0x08;                                //CLR FLAG

        CAN_ID = CanReadMsg(RX_BUF);                  //接收 CAN 总线数据

        CanSendMsg(CAN_ID+1,RX_BUF);                  //CANID 加 1, 原样发送 CAN 总线数据
    }
}

```

20.4.3 CAN 总线扩展帧收发范例

C 语言代码

//测试工作频率为 24MHz

```

#include "stc16.h"          //头文件见附录
#include "intrins.h"

```

```

typedef unsigned char  u8;
typedef unsigned int   u16;
typedef unsigned long   u32;

```

```

#define MAIN_Fosc        24000000UL

```

/****** 用户定义宏 *****/

//CAN 总线波特率=Fclk/((1+(TSG1+1)+(TSG2+1))*(BRP+1)*2)

```

#define TSG1    2          //0~15

```

```

#define TSG2    1          //0~7

```

```

#define BRP     3          //0~63

```

//24000000/((1+3+2)*4*2)=500KHz

```

#define SJW     1          //重新同步跳跃宽度

```

/****** 用户定义宏 *****/

```

u32 CAN_ID;

```

```

u8 RX_BUF[8];

```

```

u8 TX_BUF[8];

```

```

void CANInit();

```

```

void CanSendMsg(u32 canid, u8 *pdat);

```

```

void main(void)

```

```

{

```

```

    WTST = 0;

```

```

    P0M1 = 0;    P0M0 = 0;

```

//设置为准双向口

```

    P1M1 = 0;    P1M0 = 0;

```

//设置为准双向口

```

    P2M1 = 0;    P2M0 = 0;

```

//设置为准双向口

```

    P3M1 = 0;    P3M0 = 0;

```

//设置为准双向口

```

P4M1 = 0;   P4M0 = 0;           // 设置为准双向口
P5M1 = 0;   P5M0 = 0;           // 设置为准双向口
P6M1 = 0;   P6M0 = 0;           // 设置为准双向口
P7M1 = 0;   P7M0 = 0;           // 设置为准双向口

CANInit();

EA = 1;           // 打开总中断

CAN_ID = 0x01234567;
TX_BUF[0] = 0x11;
TX_BUF[1] = 0x22;
TX_BUF[2] = 0x33;
TX_BUF[3] = 0x44;
TX_BUF[4] = 0x55;
TX_BUF[5] = 0x66;
TX_BUF[6] = 0x77;
TX_BUF[7] = 0x88;

while(1)
{
    if(!P32)           // 按一下 P32 口按键 发送一帧固定数据
    {
        CanSendMsg(CAN_ID, TX_BUF);
        while(!P32);   // 防止重发
    }
}

u8 CanReadReg(u8 addr)
{
    u8 dat;
    CANAR = addr;
    dat = CANDR;
    return dat;
}

void CanWriteReg(u8 addr, u8 dat)
{
    CANAR = addr;
    CANDR = dat;
}

void CanReadFifo(u8 *pdat)
{
    pdat[0] = CanReadReg(RX_BUF0);
    pdat[1] = CanReadReg(RX_BUF1);
    pdat[2] = CanReadReg(RX_BUF2);
    pdat[3] = CanReadReg(RX_BUF3);

    pdat[4] = CanReadReg(RX_BUF0);
    pdat[5] = CanReadReg(RX_BUF1);
    pdat[6] = CanReadReg(RX_BUF2);
    pdat[7] = CanReadReg(RX_BUF3);

    pdat[8] = CanReadReg(RX_BUF0);
    pdat[9] = CanReadReg(RX_BUF1);
    pdat[10] = CanReadReg(RX_BUF2);
    pdat[11] = CanReadReg(RX_BUF3);
}

```

```

    pdat[12] = CanReadReg(RX_BUF0);
    pdat[13] = CanReadReg(RX_BUF1);
    pdat[14] = CanReadReg(RX_BUF2);
    pdat[15] = CanReadReg(RX_BUF3);
}

u32 CanReadMsg(u8 *pdat)
{
    u8 i;
    u32 CanID;
    u8 buffer[16];

    CanReadFifo(buffer);
    CanID = (((u32)buffer[1] << 24) + ((u32)buffer[2] << 16) + ((u32)buffer[3] << 8) + buffer[4]) >> 3;
    for(i=0;i<8;i++)
    {
        pdat[i] = buffer[i+5];
    }
    return CanID;
}

void CanSendMsg(u32 canid, u8 *pdat)
{
    u32 CanID;

    CanID = canid << 3;
    CanWriteReg(TX_BUF0, 0x88); //扩展帧, 数据帧, bit3~bit0: 数据长度(DLC)
    CanWriteReg(TX_BUF1, (u8)(CanID>>24));
    CanWriteReg(TX_BUF2, (u8)(CanID>>16));
    CanWriteReg(TX_BUF3, (u8)(CanID>>8));

    CanWriteReg(TX_BUF0, (u8)CanID);
    CanWriteReg(TX_BUF1, pdat[0]);
    CanWriteReg(TX_BUF2, pdat[1]);
    CanWriteReg(TX_BUF3, pdat[2]);

    CanWriteReg(TX_BUF0, pdat[3]);
    CanWriteReg(TX_BUF1, pdat[4]);
    CanWriteReg(TX_BUF2, pdat[5]);
    CanWriteReg(TX_BUF3, pdat[6]);

    CanWriteReg(TX_BUF0, pdat[7]);
    CanWriteReg(TX_BUF1, 0x00);
    CanWriteReg(TX_BUF2, 0x00);
    CanWriteReg(TX_BUF3, 0x00);

    CanWriteReg(CMR, 0x04); //发起一次帧传输
}

void CANSetBaudrate()
{
    CanWriteReg(BTR0, (SJW << 6) + BRP);
    CanWriteReg(BTR1, (TSG2 << 4) + TSG1);
}

void CANInit()
{
    CANSetBaudrate(); //设置波特率
}

```

```

    CanWriteReg(ACR0,0x00);           //总线验收代码寄存器
    CanWriteReg(ACR1,0x00);
    CanWriteReg(ACR2,0x00);
    CanWriteReg(ACR3,0x00);
    CanWriteReg(AMR0,0xFF);           //总线验收屏蔽寄存器
    CanWriteReg(AMR1,0xFF);
    CanWriteReg(AMR2,0xFF);
    CanWriteReg(AMR3,0xFF);

    CanWriteReg(IMR,0xFF);           //中断寄存器
    CanWriteReg(MR,0x00);

    P_SW1 = 0;
    CANICR = 0x02;                   //CAN 中断使能
    AUXR2 /= 0x02;                   //CAN 模块被使能
}

void CANBUS_Interrupt(void) interrupt 28
{
    u8 isr;

    isr = CanReadReg(ISR);
    if((isr & 0x04) == 0x04)
    {
        CANAR = 0x03;
        CANDR = 0x04;               //CLR FLAG
    }

    if((isr & 0x08) == 0x08)
    {
        CANAR = 0x03;
        CANDR = 0x08;               //CLR FLAG

        CAN_ID = CanReadMsg(RX_BUF); //接收 CAN 总线数据

        CanSendMsg(CAN_ID+1,RX_BUF); //CANID 加 1, 原样发送 CAN 总线数据
    }
}

```

21 LIN 总线

STC16F 系列单片机内部集成 LIN 总线功能单元，支持 LIN2.1 和 LIN1.3 协议协议。

主要功能如下：

- 帧头自动处理
- 可以在主、从两种模式之间切换
- 超时检测
- 错误分析

21.1 LIN 功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	9AH	S1_S[1:0]		CAN_S[1:0]		SPI_S[1:0]		LIN_S[1:0]	

LIN_S[1:0]: LIN 功能脚选择位

LIN_S[1:0]	LIN_RX	LIN_TX
00	P0.2	P0.3
01	P5.2	P5.3
10	P4.6	P4.7
11	P7.2	P7.3

21.2 LIN 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
LINICR	LINBUS 中断控制寄存器	B1H	-	-	-	-	PLINH	LINIF	ELIN	PLINL	0000,0000
LINAR	LINBUS 地址寄存器	B2H									0000,0000
LINDR	LINBUS 数据寄存器	B3H									0000,0000
AUXR2	辅助寄存器 2	94H	-	P40ADR	ADIBF	TXLNRX	-	-	CANEN	LINEN	0000,0000

21.2.1 辅助寄存器 2（AUXR2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR2	94H	-	P40ADR	ADIBF	TXLNRX	-	-	CANEN	LINEN

LINEN: LIN 总线使能控制位

- 0: 关闭 LIN 功能
- 1: 使能 LIN 功能

21.2.2 LIN 总线中断控制寄存器（LINICR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LINICR	B1H	-	-	-	-	PLINH	LINIF	ELIN	PLINL

ELIN: LIN 总线中断使能控制位

- 0: 关闭 LIN 中断

1: 使能 LIN 中断

LINIF: LIN 总线中断请求标志位, 硬件清零 (读取 LSR 清零)。

PLINH, PLINL: LIN 中断优先级控制位

PLINH	PLINL	优先级
0	0	0 (最低)
0	1	1
1	0	2
1	1	3 (最高)

21.2.3 LIN 总线地址寄存器 (LINAR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LINAR	B2H								

21.2.4 LIN 总线数据寄存器 (LINDR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LINDR	B3H								

21.3 LIN 内部功能寄存器

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
08H	HDRL	HDRH	HDP					
00H	LBUF	LSEL	LID	LER	LIE	LSR	DLL	DLH

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0		
	LBUF	0x00	LBUF								
	LSEL	0x01	ANIC				INDEX				
	LID	0x02			ID						
	LER	0x03		OVER	SYNCR	TOVER	CHKER	PER	BITER	FER	
	LIE	0x04					ABORTE	ERRE	RDYE	LIDE	
	LSR	0x05	LOG SIZE				ABORT	ERR	RDY	LID	
	LCR	0x05	MODE	LIN13	SIZE					CMD	
	DLL	0x06	DLL								
	DLH	0x07	SYNC	DLH							
	HDRL	0x08	HDRL								
	HDRH	0x09	HDRH								
	HDP	0x0A			HDP						

21.3.1 LIN 数据寄存器 (LBUF)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
----	----	----	----	----	----	----	----	----	----

LBUF	0x00	LBUF
------	------	------

LBUF 是 LIN 模块内部数据 FIFO 的数据接口。

21.3.2 LIN 数据地址寄存器 (LSEL)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LSEL	0x01	AINC				INDEX			

AINC 地址自增

0: 写入 LIN 内部 FIFO 的数据地址由 INDEX 决定。

1: 数据地址自增, 每操作一次 LBUF, LIN 内部 FIFO 的数据地址自动加一。

21.3.3 LIN 帧 ID 寄存器 (LID)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LID	0x02			ID					

LID 帧 ID, 如果 LCR[5:2]=4'b1111, 那么 LID[5:4]的组合代表数据帧中数据的数量:

00: 2 字节

01: 2 字节

10: 4 字节

11: 8 字节

21.3.4 LIN 错误寄存器 (LER)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LER	0x03		OV	SYN	TOV	CHK	PER	BIT	FER

OV: over run error。当 LIN 正在运行命令 (RDY 为低) 时, 用户发送了新的命令到 LIN。

SYN: slave 模式下才有, LIN 接收帧同步时产生错误。

TOV: timeout 错误。在最大允许时间内, LIN 没有收到完整的数据帧。

CHK: checksum 错误。

PER: Parity error, 没有正确接收到受保护的 ID 段。

BIT: 位错误, 标志 LIN 发送的数据位与总线监测到的状态不一致。

FER: 帧错误, 接收到的数据没有包含有效的停止位。

读取清除错误寄存器。

21.3.5 LIN 中断使能寄存器 (LIE)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LIE	0x04					ABORTE	ERRE	RDYE	LIDE

ABORTE: 使能终止中断。

ERRE: 使能错误中断。

RDYE: 使能 Ready 中断。

LIDE: 使能 head 中断。

21.3.6 LIN 状态寄存器 (只读寄存器) (LSR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LSR	0x05	LOG SIZE				ABORT	ERR	RDY	LID

LOG SIZE: LOG 模式时, 检测到的数据长度。

ABORT: 终止命令正在执行中。

ERR: 错误状态。

RDY: Ready 状态, 可以执行新的命令。

LID: 接收到了正确的 header。

Bit0~Bit3 读取后清零。

21.3.7 LIN 控制寄存器 (只写寄存器) (LCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LCR	0x05	MODE	LIN13	SIZE				CMD	

MODE: LIN 模式选择

0: 从模式

1: 主模式。

LIN13: 校验和选择。

0: 增强校验和, LIN2.1 协议。

1: 普通校验和, LIN1.3 协议。

SIZE: 数据长度

设置	长度	设置	长度
0000	0 字节	1000	8 字节
0001	1 字节	1001	保留
0010	2 字节	1010	保留
0011	3 字节	1011	保留
0100	4 字节	1100	保留
0101	5 字节	1101	保留
0110	6 字节	1110	LOG MODE
0111	7 字节	1111	数据长度由 LID[5:4]决定

CMD: LIN 命令。

00: Abort 命令。

01: 主模式 Send header 命令。

10: TX response。

11: RX response。

21.3.8 LIN 波特率寄存器 (DLL/DLH)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DLL	0x06	DLL							
DLH	0x07	SYNC	DLH						

DLL 和 DLH 决定了 LIN 模块的波特率。波特率 = SYSclk/16/DL。

SYNC: 同步模式, 只有在从模式下, 才有用。

21.3.9 LIN 帧头延时计数寄存器 (HDRL/HDRH)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
HDRL	0x08	HDRL							
HDRH	0x09	HDRH							

延时(ms) = (HDR*1000)/ SYSclk。

21.3.10 LIN 帧头延时分频寄存器 (HDP)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
----	----	----	----	----	----	----	----	----	----

HDP	0x0A			HDP
-----	------	--	--	-----

HDR 和 HDP 共同定义了一个帧头的延时时间,这个时间用于定义软件配置完发送帧头的命令与 LIN 模块实际发送 head 帧头的时间延时。

STC MCU

21.4 范例程序

21.4.1 LIN 总线主机收发范例

C 语言代码

//测试工作频率为24MHz

#include "stc16.h"

//头文件见附录

#include "intrins.h"

typedef unsigned char u8;

typedef unsigned int u16;

typedef unsigned long u32;

#define MAIN_Fosc 24000000UL

sbit SLP_N = P5^3;

//LIN 收发器控制脚0: 休眠; 1: 工作

#define LIN_MODE 1

//0: LIN2.1; 1: LIN1.3

#define FRAME_LEN 8

//数据长度: 8 字节

u8 Lin_ID;

u8 TX_BUF[8];

u8 Key1_cnt;

u8 Key2_cnt;

bit Key1_Flag;

bit Key2_Flag;

void LinInit();

void LinSendMsg(u8 lid, u8 *pdat);

void LinSendHeader(u8 lid);

void delay_ms(u8 ms);

void main(void)

{

WTST = 0;

P0M1 = 0; P0M0 = 0;

//设置为准双向口

P1M1 = 0; P1M0 = 0;

//设置为准双向口

P2M1 = 0; P2M0 = 0;

//设置为准双向口

P3M1 = 0; P3M0 = 0;

//设置为准双向口

P4M1 = 0; P4M0 = 0;

//设置为准双向口

P5M1 = 0; P5M0 = 0;

//设置为准双向口

P6M1 = 0; P6M0 = 0;

//设置为准双向口

P7M1 = 0; P7M0 = 0;

//设置为准双向口

P_SW2 /= 0x80;

P0PU = 0x0c;

//LIN_TX, LIN_RX 脚开后内部上拉

P_SW2 &= ~0x80;

Lin_ID = 0x32;

LinInit();

EA = 1;

//打开总中断

SLP_N = 1;

TX_BUF[0] = 0x11;

```

TX_BUF[1] = 0x22;
TX_BUF[2] = 0x33;
TX_BUF[3] = 0x44;
TX_BUF[4] = 0x55;
TX_BUF[5] = 0x66;
TX_BUF[6] = 0x77;
TX_BUF[7] = 0x88;

while(1)
{
    delay_ms(1);
    if(!P32)
    {
        if(!Key1_Flag)
        {
            Key1_cnt++;
            if(Key1_cnt > 50) //50ms 防抖
            {
                P46 = ~P46;
                Key1_Flag = 1;
                LinSendMsg(Lin_ID, TX_BUF); //主机发送完整帧
            }
        }
    }
    else
    {
        Key1_cnt = 0;
        Key1_Flag = 0;
    }

    if(!P33)
    {
        if(!Key2_Flag)
        {
            Key2_cnt++;
            if(Key2_cnt > 50) //50ms 防抖
            {
                P47 = ~P47;
                Key2_Flag = 1;
                LinSendHeader(0x13); //主机发送 Header，由特定标识符从机发送应答数据，拼成一个完整的帧
            }
        }
    }
    else
    {
        Key2_cnt = 0;
        Key2_Flag = 0;
    }
}

}

void delay_ms(u8 ms)
{
    u16 i;
    do{
        i = MAIN_Fosc / 6000;
        while(--i);
    }while(--ms);
}

```

```
u8 LinReadReg(u8 addr)
{
    u8 dat;
    LINAR = addr;
    dat = LINDR;
    return dat;
}

void LinWriteReg(u8 addr, u8 dat)
{
    LINAR = addr;
    LINDR = dat;
}

void LinReadMsg(u8 *pdat)
{
    u8 i;

    LinWriteReg(LSEL, 0x80); //地址自增, 从 0 开始
    for(i=0; i<FRAME_LEN; i++)
    {
        pdat[i] = LinReadReg(LBUF);
    }
}

void LinSetMsg(u8 *pdat)
{
    u8 i;

    LinWriteReg(LSEL, 0x80); //地址自增, 从 0 开始
    for(i=0; i<FRAME_LEN; i++)
    {
        LinWriteReg(LBUF, pdat[i]);
    }
}

void LinSetID(u8 lid)
{
    LinWriteReg(LID, lid); //设置总线ID
}

u8 GetLinError(void)
{
    u8 sta;
    sta = LinReadReg(LER); //读取清除错误寄存器
    return sta;
}

u8 WaitLinReady(void)
{
    u8 lsr;
    do{
        lsr = LinReadReg(LSR);
    }while(!(lsr & 0x02)); //判断 ready 状态
    return lsr;
}

void SendAbortCmd(void)
```

```

{
    LinWriteReg(LCR,0x80);                //主模式 Send Abort
}

void SendHeadCmd(void)
{
    LinWriteReg(LCR,0x81);                //主模式 Send Header
}

void SendDatCmd(void)
{
    u8 lcr_val;
    lcr_val = 0x82+(LIN_MODE<<6)+(FRAME_LEN<<2);
    LinWriteReg(LCR,lcr_val);
}

void ResponseTxCmd(void)
{
    u8 lcr_val;
    lcr_val = 0x02+(LIN_MODE<<6)+(FRAME_LEN<<2);
    LinWriteReg(LCR,lcr_val);
}

void ResponseRxCmd(void)
{
    u8 lcr_val;
    lcr_val = 0x03+(LIN_MODE<<6)+(FRAME_LEN<<2);
    LinWriteReg(LCR,lcr_val);
}

void LinSendMsg(u8 lid, u8 *pdat)
{
    LinSetID(lid);                        //设置总线ID
    LinSetMsg(pdat);

    SendHeadCmd();                        //主模式 Send Seader
    WaitLinReady();                       //等待ready 状态
    GetLinError();                        //读取清除错误寄存器

    SendDatCmd();                         //Send Data
    WaitLinReady();                       //等待ready 状态
    GetLinError();                        //读取清除错误寄存器
}

void LinSendHeader(u8 lid)
{
    LinSetID(lid);                        //设置发送Response 的从机总线ID

    SendHeadCmd();                        //主模式 send header
    WaitLinReady();                       //等待ready 状态
    GetLinError();                        //读取清除错误寄存器

    ResponseRxCmd();                      //RX response
    WaitLinReady();                       //等待ready 状态
    GetLinError();                        //读取清除错误寄存器

    LinReadMsg(TX_BUF);                   //接收Lin 总线从机发送的应答数据
}

```

```

void LinSetBaudrate(u16 brt)
{
    u16 tmp;
    tmp = (MAIN_Fosc >> 4) / brt;
    LinWriteReg(DLH,(u8)(tmp>>8));           // 设置波特率
    LinWriteReg(DLL,(u8)tmp);
}

void LinSetHeadDelay(u8 base_ms, u8 prescaler)
{
    u16 tmp;
    tmp = (MAIN_Fosc * base_ms) / 1000;      // 注意base_ms 取值范围，计算结果不要超过 16 位上限
    LinWriteReg(HDRH,(u8)(tmp>>8));
    LinWriteReg(HDRL,(u8)tmp);               // 设置帧头延时计数
    LinWriteReg(HDP,prescaler);              // 设置帧头延时分频
}

void LinInit()
{
    P_SW1 = 0x00;                            // 选择 P0.2,P0.3
    LINICR = 0x02;                            // LIN 模块中断使能
    AUXR2 /= 0x01;                            // LIN 模块被使能

    GetLinError();                            // 读取清除错误寄存器
    LinWriteReg(LIE,0x00);                    // LIE 中断使能寄存器
    LinSetBaudrate(9600);                     // 设置波特率
    LinSetHeadDelay(0x01,0x00);               // 设置帧头延时
}

```

21.4.2 LIN 总线从机收发范例

C 语言代码

//测试工作频率为 24MHz

```

#include "stc16.h"           //头文件见附录
#include "intrins.h"

typedef unsigned char  u8;
typedef unsigned int   u16;
typedef unsigned long  u32;

#define MAIN_Fosc      24000000UL

sbit SLP_N  = P5^3;          //LIN 收发器控制脚 0: 休眠; 1: 工作

#define LIN_MODE      1      //0: LIN2.1; 1: LIN1.3
#define FRAME_LEN     8      //数据长度: 8 字节

u8 Lin_ID;
u8 TX_BUF[8];
bit RxFlag;

void LinInit();
void LinReadMsg(u8 *pdat);
void ResponseRxCmd(void);

```

```
u8 WaitLinReady(void);
```

```
u8 GetLinError(void);
```

```
void main(void)
```

```
{
    WTST = 0;
    P0M1 = 0;   P0M0 = 0;           // 设置为准双向口
    P1M1 = 0;   P1M0 = 0;           // 设置为准双向口
    P2M1 = 0;   P2M0 = 0;           // 设置为准双向口
    P3M1 = 0;   P3M0 = 0;           // 设置为准双向口
    P4M1 = 0;   P4M0 = 0;           // 设置为准双向口
    P5M1 = 0;   P5M0 = 0;           // 设置为准双向口
    P6M1 = 0;   P6M0 = 0;           // 设置为准双向口
    P7M1 = 0;   P7M0 = 0;           // 设置为准双向口

    P_SW2 /= 0x80;
    P0PU = 0x0c;                     // LIN_TX, LIN_RX 脚开后内部上拉
    P_SW2 &= ~0x80;

    Lin_ID = 0x32;
    LinInit();
    EA = 1;                          // 打开总中断

    SLP_N = 1;
    TX_BUF[0] = 0x11;
    TX_BUF[1] = 0x22;
    TX_BUF[2] = 0x33;
    TX_BUF[3] = 0x44;
    TX_BUF[4] = 0x55;
    TX_BUF[5] = 0x66;
    TX_BUF[6] = 0x77;
    TX_BUF[7] = 0x88;

    while(1)
    {
        if(RxFlag == 1)
        {
            ResponseRxCmd();          // RX response
            WaitLinReady();           // 等待 ready 状态
            GetLinError();            // 读取清除错误寄存器

            LinReadMsg(TX_BUF);       // 接收 Lin 总线数据
            RxFlag = 0;
        }
    }
}

u8 LinReadReg(u8 addr)
{
    u8 dat;
    LINAR = addr;
    dat = LINDR;
    return dat;
}

void LinWriteReg(u8 addr, u8 dat)
{
    LINAR = addr;
    LINDR = dat;
}
```



```
}

void LinReadMsg(u8 *pdat)
{
    u8 i;

    LinWriteReg(LSEL,0x80); //地址自增,从0开始

    for(i=0;i<FRAME_LEN;i++)
    {
        pdat[i] = LinReadReg(LBUF);
    }
}

void LinSetMsg(u8 *pdat)
{
    u8 i;

    LinWriteReg(LSEL,0x80); //地址自增,从0开始
    for(i=0;i<FRAME_LEN;i++)
    {
        LinWriteReg(LBUF,pdat[i]);
    }
}

void LinSetID(u8 lid)
{
    LinWriteReg(LID,lid); //设置总线ID
}

u8 GetLinError(void)
{
    u8 sta;
    sta = LinReadReg(LER); //读取清除错误寄存器
    return sta;
}

u8 WaitLinReady(void)
{
    u8 lsr;
    do{
        lsr = LinReadReg(LSR);
    }while(!(lsr & 0x02)); //判断ready 状态
    return lsr;
}

void SendAbortCmd(void)
{
    LinWriteReg(LCR,0x80); //主模式 Send Abort
}

void SendHeadCmd(void)
{
    LinWriteReg(LCR,0x81); //主模式 Send Header
}

void SendDatCmd(void)
{
    u8 lcr_val;
```

```

    lcr_val = 0x82+(LIN_MODE<<6)+(FRAME_LEN<<2);
    LinWriteReg(LCR,lcr_val);
}

void ResponseTxCmd(void)
{
    u8 lcr_val;
    lcr_val = 0x02+(LIN_MODE<<6)+(FRAME_LEN<<2);
    LinWriteReg(LCR,lcr_val);
}

void ResponseRxCmd(void)
{
    u8 lcr_val;
    lcr_val = 0x03+(LIN_MODE<<6)+(FRAME_LEN<<2);
    LinWriteReg(LCR,lcr_val);
}

void LinTxResponse(u8 *pdat)
{
    LinSetMsg(pdat);
    ResponseTxCmd();
    WaitLinReady();
    GetLinError();
}

void LinSetBaudrate(u16 brt)
{
    u16 tmp;
    tmp = (MAIN_Fosc >> 4) / brt;
    LinWriteReg(DLH,(u8)(tmp>>8));
    LinWriteReg(DLL,(u8)tmp);
}

void LinSetHeadDelay(u8 base_ms, u8 prescaler)
{
    u16 tmp;
    tmp = (MAIN_Fosc * base_ms) / 1000;
    LinWriteReg(HDRH,(u8)(tmp>>8));
    LinWriteReg(HDRL,(u8)tmp);
    LinWriteReg(HDP,prescaler);
}

void LinInit()
{
    P_SW1 = 0x00;
    LINICR = 0x02;
    AUXR2 /= 0x01;
    GetLinError();
    LinWriteReg(LIE,0x0F);
    LinSetBaudrate(9600);
    LinSetHeadDelay(0x01,0x00);
}

void LinBUS_Interrupt(void) interrupt LIN_VECTOR
{
    u8 isr;

```

//TX response
//等待ready 状态
//读取清除错误寄存器

//注意base_ms 取值范围，计算结果不要超过16 位上限
//设置帧头延时计数
//设置帧头延时分频

//选择P0.2,P0.3
//LIN 模块中断使能
//LIN 模块被使能
//读取清除错误寄存器
//LIR 中断使能寄存器
//设置波特率
//设置帧头延时

```

    isr = LinReadReg(LSR);
    if((isr & 0x03) == 0x03)                                     //接收到Header, 处于Ready 状态
    {
        isr = LinReadReg(LER);
        if(isr == 0x00)                                         //没有产生错误
        {
            P46 = ~P46;

            isr = LinReadReg(LID);
            if(isr == 0x12)                                     //判断是否从机响应ID
            {
                LinTxResponse(TX_BUF);                         //返回响应数据
            }
            else
            {
                RxFlag = 1;
            }
        }
    }
    else
    {
        isr = LinReadReg(LER);                                 //读取清除错误寄存器
    }
}

```

21.4.3 使用串口模拟 LIN 总线范例

LIN (Local Interconnect Network) 总线是基于 UART/SCI (通用异步收发器/串行接口) 的低成本串行通讯协议。LIN 的字节场 和 UART 一样都是 8N1 的格式, 但是 LIN 的帧间隔场是连续 13 个显性电平, 与串口的 8 位不符, 这里通过切换波特率的方式输出 13 个显性电平宽度的信号作为间隔场。

以 9600 波特率为例:

13 个显性信号时间 = $13/9600 = 1354\mu s$

转换成发送一个字节 0x00, 1 个起始位 + 8 个数据位 + 1 个停止位, 其中包含 9 个显性电平。

转换波特率 = $9/1354\mu s = 6647$ 波特率

本例程测试方法: UART1 通过串口工具连接电脑; UART2 外接 LIN 收发器, 连接 LIN 总线。将电脑串口发送的数据转发到 LIN 总线; 从 LIN 总线接收到的数据转发到电脑串口。

C 语言代码

//测试工作频率为24MHz

```

#include "stc16.h"      //头文件见附录
#include "intrins.h"

```

```

typedef unsigned char u8;
typedef unsigned int u16;
typedef unsigned long u32;

```

```

#define MAIN_Fosc      24000000UL

```

```

#define Baudrate1      (65536UL - (MAIN_Fosc / 4) / 9600UL)
#define Baudrate2      (65536UL - (MAIN_Fosc / 4) / 9600UL) //发送数据传输波特率

#define Baudrate_Break (65536UL - (MAIN_Fosc / 4) / 6647UL) //发送显性间隔信号波特率

#define UART1_BUF_LENGTH 32
#define UART2_BUF_LENGTH 32

#define LIN_ID          0x31

sbit SLP_N = P2^4; //LIN 收发器控制脚0: 休眠; 1: 工作

u8 TX1_Cnt; //发送计数
u8 RX1_Cnt; //接收计数
u8 TX2_Cnt; //发送计数
u8 RX2_Cnt; //接收计数
bit B_TX1_Busy; //发送忙标志
bit B_TX2_Busy; //发送忙标志
u8 RX1_TimeOut;
u8 RX2_TimeOut;

u8 xdata RX1_Buffer[UART1_BUF_LENGTH]; //接收缓冲
u8 xdata RX2_Buffer[UART2_BUF_LENGTH]; //接收缓冲

void UART1_config(void);
void UART2_config(void);
void delay_ms(u8 ms);
void UART1_TxByte(u8 dat);
void UART2_TxByte(u8 dat);
void Lin_Send(u8 *puts);
void SetTimer2Baudrate(u16 dat);

void main(void)
{
    u8 i;

    WTST = 0;
    P0M1 = 0; P0M0 = 0; //设置为准双向口
    P1M1 = 0; P1M0 = 0; //设置为准双向口
    P2M1 = 0; P2M0 = 0; //设置为准双向口
    P3M1 = 0; P3M0 = 0; //设置为准双向口
    P4M1 = 0; P4M0 = 0; //设置为准双向口
    P5M1 = 0; P5M0 = 0; //设置为准双向口
    P6M1 = 0; P6M0 = 0; //设置为准双向口
    P7M1 = 0; P7M0 = 0; //设置为准双向口

    UART1_config();
    UART2_config();
    EA = 1; //允许全局中断
    SLP_N = 1;

    while(1)
    {
        delay_ms(1);
        if(RX1_TimeOut > 0)
        {
            if(--RX1_TimeOut == 0) //超时,则串口接收结束
            {
                if(RX1_Cnt > 0)
            }
        }
    }
}

```

```

        {
            Lin_Send(RX1_Buffer);           //将UART1 收到的数据发送到LIN 总线上
        }
        RX1_Cnt = 0;
    }
}

if(RX2_TimeOut > 0)
{
    if(--RX2_TimeOut == 0)                //超时,则串口接收结束
    {
        if(RX2_Cnt > 0)
        {
            for (i=0; i < RX2_Cnt; i++)    //遇到停止符0 结束
            {
                UART1_TxByte(RX2_Buffer[i]); //从LIN 总线收到的数据发送到UART1
            }
        }
        RX2_Cnt = 0;
    }
}
}

void delay_ms(u8 ms)
{
    u16 i;
    do{
        i = MAIN_Fosc / 6000;
        while(--i);
    }while(--ms);
}

u8 Lin_CheckPID(u8 id)                   //ID 转PID
{
    u8 pid;
    u8 P0 ;
    u8 P1 ;

    P0 = (((id)^(id>>1)^(id>>2)^(id>>4))&0x01)<<6 ;
    P1 = ((~((id>>1)^(id>>3)^(id>>4)^(id>>5)))&0x01)<<7 ;

    pid = id/P0/P1 ;

    return pid;
}

static u8 LINCalcChecksum(u8 *dat)       //计算LIN1.3 经典校验码
{
    u16 sum = 0;
    u8 i;

    for(i = 0; i < 8; i++)
    {
        sum += dat[i];
        if(sum & 0xFF00)
        {
            sum = (sum & 0x00FF) + 1;
        }
    }
}

```

```

    }
    sum ^= 0x00FF;
    return (u8)sum;
}

void Lin_SendBreak(void)
{
    SetTimer2Baudrate((u16)Baudrate_Break);
    UART2_TxByte(0);
    SetTimer2Baudrate((u16)Baudrate2);
}

void Lin_Send(u8 *puts)
{
    u8 i;

    Lin_SendBreak();                //发送帧间隔场
    UART2_TxByte(0x55);            //发送同步场
    UART2_TxByte(Lin_CheckPID(LIN_ID)); //发送标识符
    for(i=0;i<8;i++)
    {
        UART2_TxByte(puts[i]);
    }
    UART2_TxByte(LINCalcChecksum(puts));
}

void UART1_TxByte(u8 dat)
{
    SBUF = dat;
    B_TX1_Busy = 1;
    while(B_TX1_Busy);
}

void UART2_TxByte(u8 dat)
{
    S2BUF = dat;
    B_TX2_Busy = 1;
    while(B_TX2_Busy);
}

void SetTimer2Baudrate(u16 dat)
{
    AUXR &= ~(1<<4);                //Timer stop
    AUXR &= ~(1<<3);                //Timer2 set As Timer
    AUXR /= (1<<2);                 //Timer2 set as 1T mode
    T2H = dat / 256;
    T2L = dat % 256;
    IE2 &= ~(1<<2);                //禁止中断
    AUXR /= (1<<4);                 //Timer run enable
}

void UART1_config(void)
{
    TRI = 0;
    AUXR &= ~0x01;                //SI BRT Use Timer1;
    AUXR /= (1<<6);                //Timer1 set as 1T mode
    TMOD &= ~(1<<6);                //Timer1 set As Timer
    TMOD &= ~0x30;                 //Timer1_16bitAutoReload;
    TH1 = (u8)(Baudrate1 / 256);
}

```

```

    TL1 = (u8)(Baudrate1 % 256);
    ET1 = 0;                                     //禁止中断
    INTCLKO &= ~0x02;                             //不输出时钟
    TRI = 1;

    SCON = (SCON & 0x3f) | 0x40;                 //UART1 模式8 位数据 可变波特率
    ES = 1;                                       //允许中断
    REN = 1;                                     //允许接收
    P_SW1 &= 0x3f;                               //UART1 切换至 P3.0 P3.1

    B_TX1_Busy = 0;
    TX1_Cnt = 0;
    RX1_Cnt = 0;
}

void UART2_config(void)
{
    SetTimer2Baudrate((u16)Baudrate2);
    S2CON &= ~(1<<7);                             // 8 位数据 1 位起始位 1 位停止位 无校验
    IE2 |= 1;                                     //允许中断
    S2CON |= (1<<4);                             //允许接收
    P_SW2 &= ~0x01;                               //UART2 switch to: 0: P1.0 P1.1, 1: P4.6 P4.7

    B_TX2_Busy = 0;
    TX2_Cnt = 0;
    RX2_Cnt = 0;
}

void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        if(RX1_Cnt >= UART1_BUF_LENGTH) RX1_Cnt = 0;
        RX1_Buffer[RX1_Cnt] = SBUF;
        RX1_Cnt++;
        RX1_TimeOut = 5;
    }

    if(TI)
    {
        TI = 0;
        B_TX1_Busy = 0;
    }
}

void UART2_int (void) interrupt 8
{
    if((S2CON & 1) != 0)
    {
        S2CON &= ~1;                             //清除标志位
        if(RX2_Cnt >= UART2_BUF_LENGTH) RX2_Cnt = 0;
        RX2_Buffer[RX2_Cnt] = S2BUF;
        RX2_Cnt++;
        RX2_TimeOut = 5;
    }

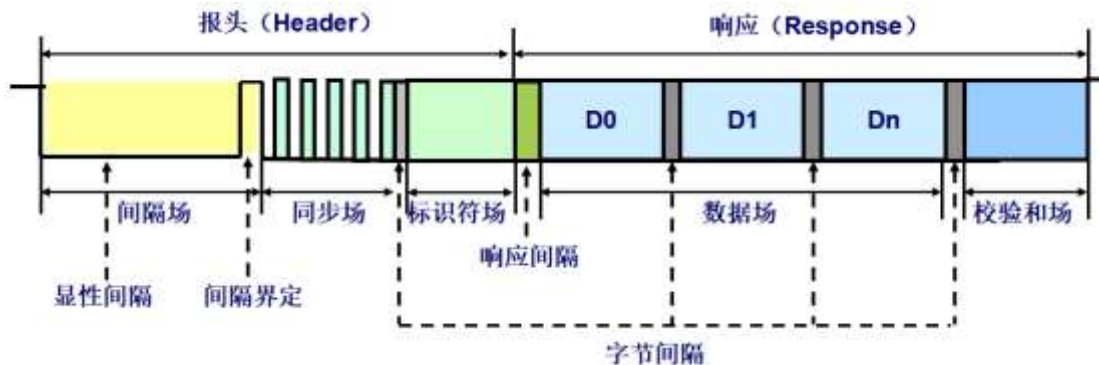
    if((S2CON & 2) != 0)
    {

```

```
        S2CON &= ~2;
        B_TX2_Busy = 0;
    }
}
```

STC MCU

21.4.4 LIN 总线时序介绍



1、字节场

- 1) 基于 UART/SCI 的通信格式;
- 2) 发送一个字节需要 10 个位时间 (TBIT)

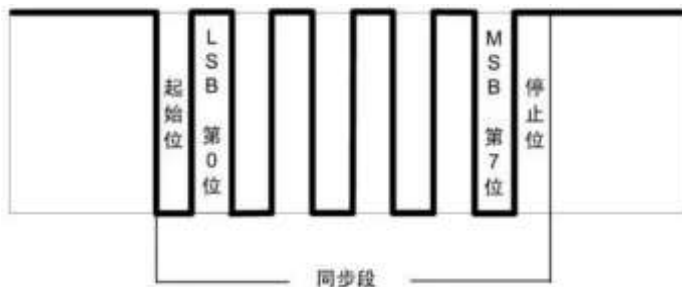
2、间隔场

- 1) 表示一帧报文的起始，由主节点发出;
- 2) 间隔信号至少由 13 个显性位组成;
- 3) 间隔界定符至少由 1 个隐形位组成;
- 4) 间隔场是唯一一个不符合字节场格式的场;
- 5) 从节点需要检测到至少连续 11 个显性位才认为是间隔信号;



3、同步场

- 1) 确保所有从节点使用与节点相同的波特率发送和接收数据;
- 2) 一个字节，结构固定：0x55;



4、标识符场

- 1) ID 的范围从 0 到 63 (0x3f);
- 2) 奇偶校验符 (Parity) P0, P1

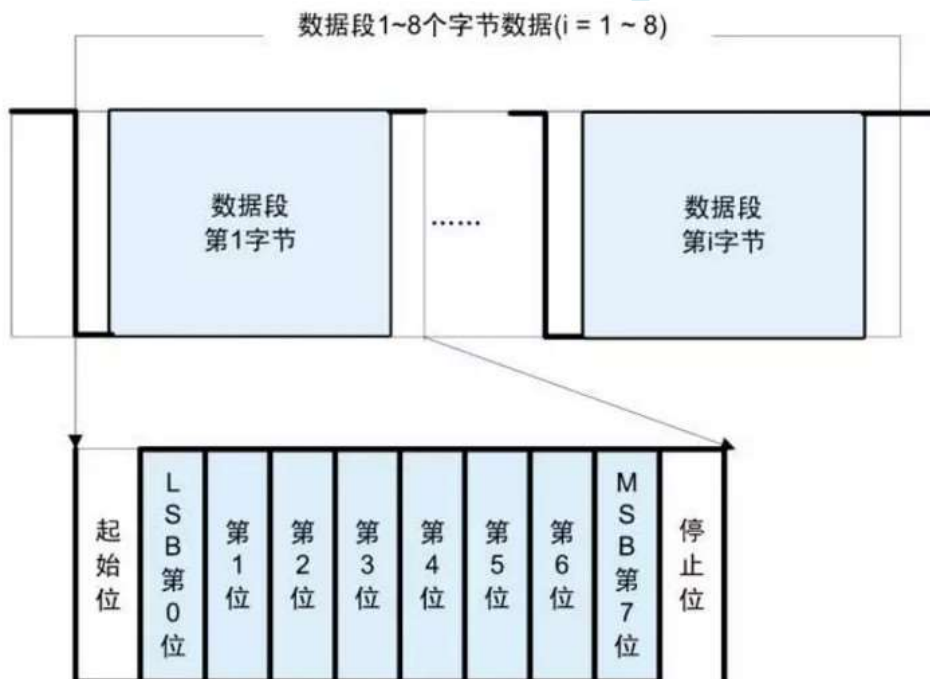
$$P0 = ID0 \oplus ID1 \oplus ID2 \oplus ID4$$

$$P1 = \neg (ID1 \oplus ID3 \oplus ID4 \oplus ID5)$$



5、数据场

- 1) 数据场长度 1 到 8 个字节;
- 2) 低字节先发, 低位先发;
- 3) 如果某信号长度超过 1 个字节采用低位在前的方式发送 (小端);



6、校验和场

用于校验接收的数据是否正确

- 1) 经典校验 (Classic Checksum) 仅校验数据场 (LIN1.3)
 - 2) 增强校验 (Enhance Checksum) 校验标识符场与数据场内容 (LIN2.0、LIN2.1)
- 标识符为 0x3C 和 0x3D 的帧只能使用经典校验
- 计算方法: 反转 8 位求和



22 32 位硬件乘除单元，MDU32

这个乘法和除法单元（称为 MDU32）提供快速的 32 位算术运算。MDU32 支持无符号和补码有符号整数操作数。MDU32 由专用的直接内存访问控制模块（称为 DMA）。所有 MDU32 算术操作都是通过向 DMA 控件写入 DMA 指令来启动的寄存器 DMAIR。MDU32 模块执行的所有算术运算的操作数和结果位于寄存器 R0-R7。

注意：

1. DMA 模块执行算术运算所需的执行时间，包括：
 - ◆ 操作数从 DR0-DR4 寄存器加载到 MDU32 模块
 - ◆ MDU32 算术运算
 - ◆ 从 MDU32 模块到 R0-R7 寄存器的结果存储
2. 处理器执行 C 编译算术函数所需的执行时间，包括：
 - ◆ DMA 指令写入 DMAIR 寄存器
 - ◆ 操作数从 DR0-DR4 寄存器加载到 MDU32 模块
 - ◆ MDU32 算术运算
 - ◆ 从 MDU32 模块到 R0-R7 寄存器的结果存储
 - ◆ 从函数返回（RET 指令）

22.1 相关的特殊功能寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DMAIR	FMU DMA 指令寄存器	8FH	DMAIR. 7	DMAIR. 6	DMAIR. 5	DMAIR. 4	DMAIR. 3	DMAIR. 2	DMAIR. 1	DMAIR. 0	0000,0000

22.2 指令执行时间

操作	DMA 指令码	机器周期
32 位乘法	0x02	3 clk
32 位无符号除法	0x04	19 clk
32 位有符号除法	0x06	21 clk

22.3 MDU32—算术运算

下面介绍 MDU32 模块可以执行的所有算术运算。

22.3.1 32 位乘法

32 位乘法运算是对两个无符号或有符号的补码整数参数执行的。第一个参数位于 R4-R7 寄存器中，第二个参数位于 R0-R3 寄存器中。运算结果存储到 R4-R7 寄存器。

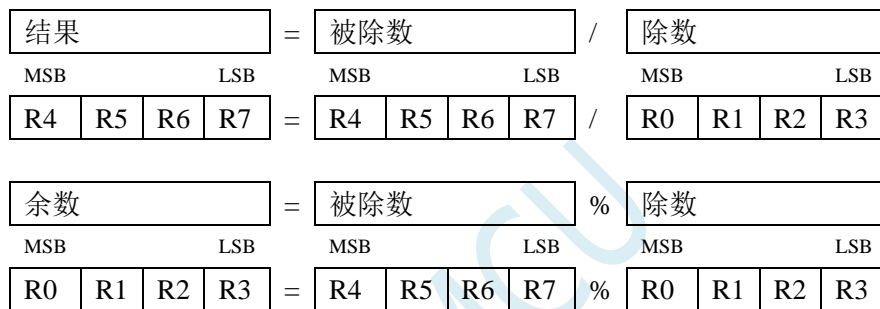


DMA 指令码: 0x02

执行时间: 3clk

22.3.2 32 位无符号除法

对两个无符号整数参数执行 32 位无符号除法运算。第一个参数“被除数”是位于 R4-R7 寄存器中，第二个参数“除数”位于 R0-R3 寄存器中。结果存储到 R4-R7 寄存器。余数在 R0-R3 中返回。除以零返回 0xFFFFFFFF。

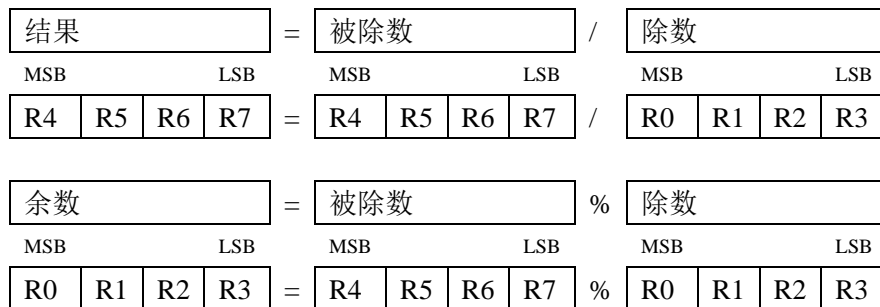


DMA 指令码: 0x04

执行时间: 19clk

22.3.3 32 位有符号除法

对两个有符号的补码参数执行 32 位有符号除法运算。第一个参数“被除数”位于 R4-R7 寄存器中，第二个参数“除数”位于 R0-R3 寄存器中。结果存储到 R4-R7 寄存器。余数在 R0-R3 中返回。除以零返回 0xFFFFFFFF。

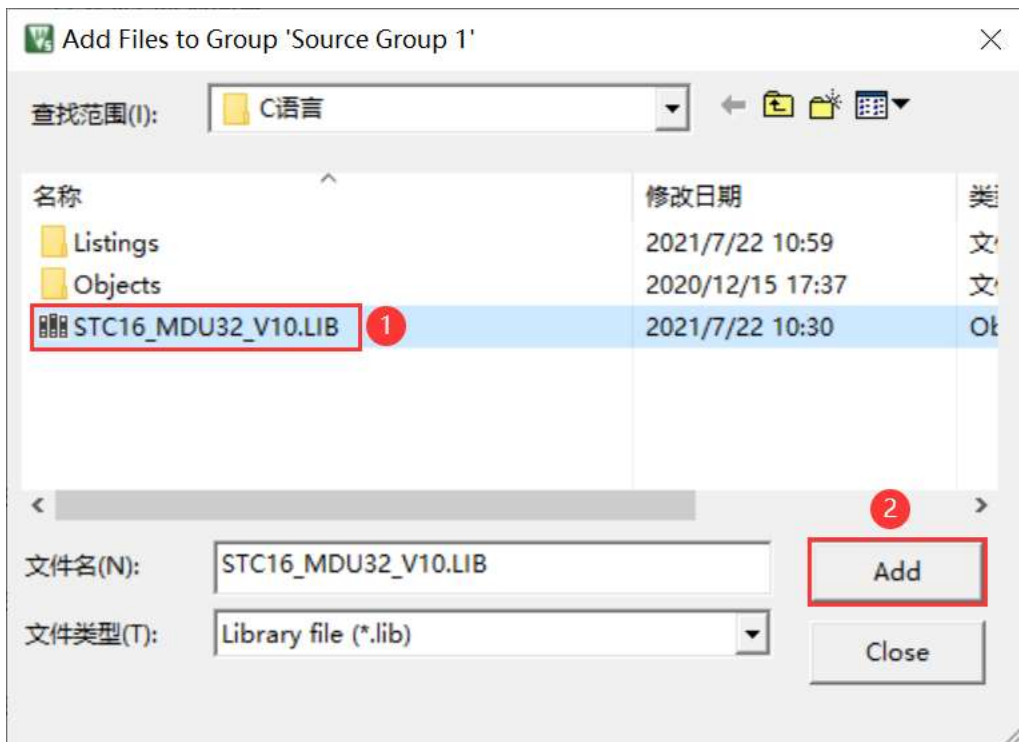


DMA 指令码: 0x06

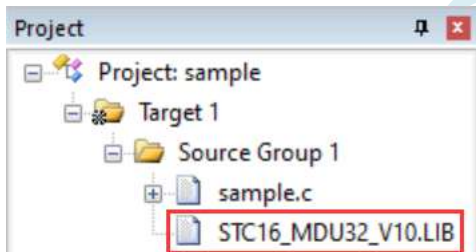
执行时间: 21clk

22.4 范例程序

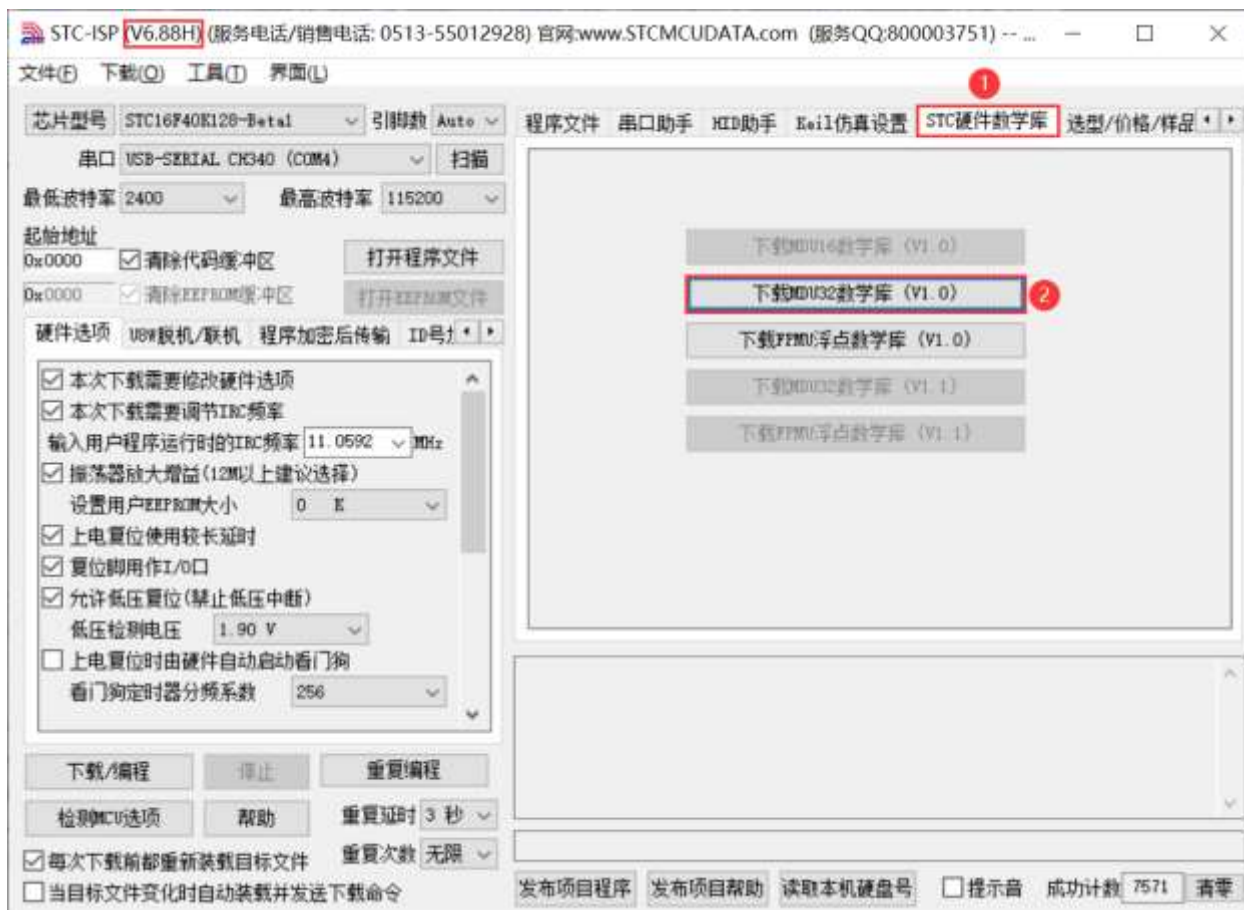
当要使用 32 位硬件乘除单元时,只要在 keil 项目中加入库文件“STC16_MDU32_Vxx.LIB”即可:



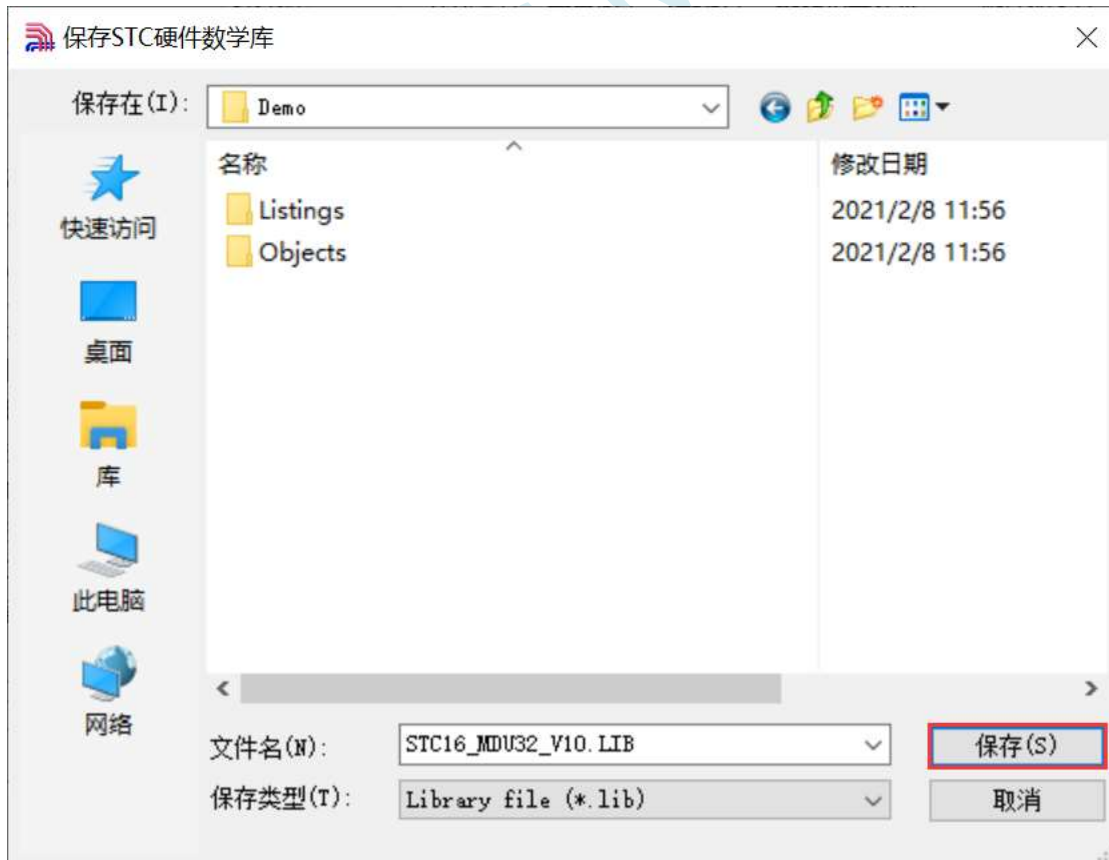
添加库文件到项目:



库文件获取方法: 通过 STC-ISP 软件 V6.88H 及其以后版本, 点击“STC 硬件数学库”标签获取。



点击需要下载的 MDU32 数学库按钮，选择保存的位置，点击“保存”按钮即可：



C 语言代码

//测试工作频率为24MHz

```
#include "stc16.h"           //头文件见附录
#include "intrins.h"
```

```
volatile unsigned long int near uint1, uint2;
volatile unsigned long int near xuint;
```

```
volatile long int sint1, sint2;
volatile long int xsint;
```

```
void main(void)
```

```
{
    WTST = 0;
    P0M1 = 0;   P0M0 = 0;           //设置为准双向口
    P1M1 = 0;   P1M0 = 0;           //设置为准双向口
    P2M1 = 0;   P2M0 = 0;           //设置为准双向口
    P3M1 = 0;   P3M0 = 0;           //设置为准双向口
    P4M1 = 0;   P4M0 = 0;           //设置为准双向口
    P5M1 = 0;   P5M0 = 0;           //设置为准双向口
    P6M1 = 0;   P6M0 = 0;           //设置为准双向口
    P7M1 = 0;   P7M0 = 0;           //设置为准双向口

    P10 = 0;
    sint1 = 0x31030F05;
    sint2 = 0x00401350;
    xsint = sint1 * sint2;

    uint1 = 5;
    uint2 = 50;
    xuint = uint1 * uint2;

    uint1 = 528745;
    uint2 = 654689;
    xuint = uint1 / uint2;

    sint1 = 2000000000;
    sint2 = 2134135177;
    xsint = sint1 / sint2;

    sint1 = -2000000000;
    sint2 = -2134135177;
    xsint = sint1 / sint2;

    sint1 = -2000000000;
    sint2 = 2134135177;
    xsint = sint1 / sint2;
    P10 = 1;

    while(1);
}
```

23 单精度浮点运算器，FPMU

23.1 FPMU – FLOATING POINT MATH UNIT SUMMARY

This floating point math unit (called FPMU) provides fast 32-bit real arithmetic operations. FPMU supports addition, subtraction, multiplication, division, square root, comparison and trigonometric functions: sine, cosine, tangent and arctangent. It has built-in conversion instructions from integer type to floating point type and vice versa. The input numbers format is according to IEEE-754 standard. The FPMU is controlled by dedicated direct memory access module (called DMA). All FPMU arithmetic operations are initiated by writing a DMA instruction to the DMA control register called DMAIR (0x8F). Operands (or their pointers) and result (or its pointer) of all arithmetic operations performed by FPMU module are located in registers R0-R7 of current bank.

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DMAIR	FPMU DMA 指令寄存器	8FH	DMAIR. 7	DMAIR. 6	DMAIR. 5	DMAIR. 4	DMAIR. 3	DMAIR. 2	DMAIR. 1	DMAIR. 0	0000,0000

All supported functions are listed below.

- ◆ FADD, FSUB – addition, subtraction
- ◆ FMUL, FDIV – multiplication, division
- ◆ FSQRT – square root
- ◆ FXAM – examine input data
- ◆ FUCOM – comparison
- ◆ FSIN, FCOS – sine, cosine
- ◆ FTAN – tangent
- ◆ FATAN – arctangent
- ◆ FCLD, FILD, FLLD – 8-/16-/32-bit integer to float
- ◆ FCST,FIST,FLSD – float to 8-/16-/32-bit integer

23.2 Execution time table

Operation	DMA instruction code	Execution time
32-BIT REAL DATA ADDITION	0x1C (28)	31– 40 clk
32-BIT REAL DATA SUBTRACTION	0x1D (29)	31 – 40 clk
32-BIT REAL DATA MULTIPLICATION	0x1E (30)	26 - 34 clk
32-BIT REAL DATA DIVISION	0x1F (31)	58 - 67 clk
32-BIT REAL DATA SQUARE ROOT	0x20 (32)	50 - 54 clk
32-BIT REAL DATA COMPARISON	0x21 (33)	18 clk
32-BIT REAL DATA EXAMINE INPUT DATA	0x22 (34)	15 clk
32-BIT REAL DATA SINE	0x2D (45)	32 - 270 clk
32-BIT REAL DATA COSINE	0x2E (46)	32 - 270 clk
32-BIT REAL DATA TANGENT	0x2F (47)	58 - 258 clk

32-BIT REAL DATA ARCTANGENT	0x30 (48)	62 - 175 clk
CONVERSION 32-BIT REAL DATA TO 8-BIT INTEGER	0x23 (35)	19 - 30 clk
CONVERSION 32-BIT REAL DATA TO 16-BIT INTEGER	0x24 (36)	19 - 30 clk
CONVERSION 32-BIT REAL DATA TO 32-BIT INTEGER	0x25 (37)	23 - 39 clk
CONVERSION 8-BIT INTEGER TO 32-BIT REAL DATA	0x27 (39)	23- 33 clk
CONVERSION 16-BIT INTEGER TO 32-BIT REAL DATA	0x28 (40)	23- 33 clk
CONVERSION 32-BIT INTEGER TO 32-BIT REAL DATA	0x29 (41)	24- 33 clk
INITIALIZE COPROCESSOR	0x31 (49)	2 clk
CLEAR EXCEPTIONS	0x32 (50)	4 clk
READ STATUS REGISTER	0x33 (51)	4 clk
WRITE STATUS REGISTER	0x34 (52)	4 clk
READ CONTROL REGISTER	0x35 (53)	4 clk
WRITE CONTROL REGISTER	0x36 (54)	4 clk

23.3 FPMU – BASIC ARITHMETIC OPERATIONS

This chapter describes all the arithmetic operations which can be executed by FPMU module using DMA controller. All operands must be located inside DATA memory. The result of operation is also stored in DATA memory space in current bank of R0-R7 as selected by PSW (0xD0) bits.

23.3.1 32-BIT REAL DATA ADDITION

The 32-bit real addition operation takes two real arguments. First argument is located in R0:R1:R2:R3 registers (Sign:Exponent:Mantissa) and the second argument is located in R4:R5:R6:R7 registers (Sign:Exponent:Mantissa). Result of the operation is loaded to R4:R5:R6:R7 register.

$$\begin{array}{|c|c|c|c|} \hline \text{result} & & & \\ \hline \text{MSB} & & \text{LSB} & \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline \text{AR} & & & \\ \hline \text{MSB} & & \text{LSB} & \\ \hline \end{array} + \begin{array}{|c|c|c|c|} \hline \text{BR} & & & \\ \hline \text{MSB} & & \text{LSB} & \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|c|} \hline \text{R4} & \text{R5} & \text{R6} & \text{R7} \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline \text{R4} & \text{R5} & \text{R6} & \text{R7} \\ \hline \end{array} + \begin{array}{|c|c|c|c|} \hline \text{R0} & \text{R1} & \text{R2} & \text{R3} \\ \hline \end{array}$$

DMA instruction code: 0x1C(28)

Execution time: 31 - 40 clk

23.3.2 32-BIT REAL DATA SUBTRACTION

The 32-bit real subtraction operation takes two real arguments. First argument is located in R0:R1:R2:R3 registers (Sign:Exponent:Mantissa) and the second argument is located in R4:R5:R6:R7 registers (Sign:Exponent:Mantissa). Result of the operation is loaded to R4:R5:R6:R7 register.

$$\begin{array}{|c|c|c|c|} \hline \text{result} & & & \\ \hline \text{MSB} & & \text{LSB} & \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline \text{AR} & & & \\ \hline \text{MSB} & & \text{LSB} & \\ \hline \end{array} - \begin{array}{|c|c|c|c|} \hline \text{BR} & & & \\ \hline \text{MSB} & & \text{LSB} & \\ \hline \end{array}$$

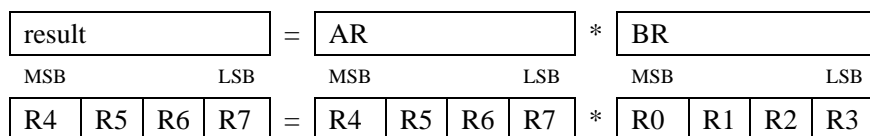
$$\begin{array}{|c|c|c|c|} \hline \text{R4} & \text{R5} & \text{R6} & \text{R7} \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline \text{R4} & \text{R5} & \text{R6} & \text{R7} \\ \hline \end{array} - \begin{array}{|c|c|c|c|} \hline \text{R0} & \text{R1} & \text{R2} & \text{R3} \\ \hline \end{array}$$

DMA instruction code: 0x1D(29)

Execution time: 31 - 40 clk

23.3.3 32-BIT REAL DATA MULTIPLICATION

The 32-bit real multiplication operation takes two double real arguments. First argument is located in R0:R1:R2:R3 registers (Sign:Exponent:Mantissa) and the second argument is located in R4:R5:R6:R7 registers (Sign:Exponent:Mantissa). Result of the operation is loaded to R4:R5:R6:R7 register.

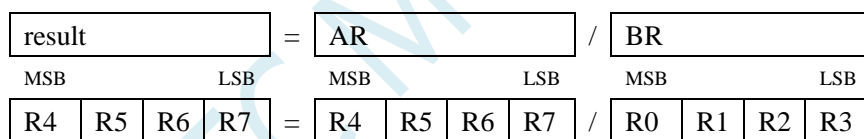


DMA instruction code: 0x1E(30)

Execution time: 26 - 34 clk

23.3.4 32-BIT REAL DATA DIVISION

The 32-bit real division operation takes two double real arguments. First argument is located in R0:R1:R2:R3 registers (Sign:Exponent:Mantissa) and the second argument is located in R4:R5:R6:R7 registers (Sign:Exponent:Mantissa). Result of the operation is loaded to R4:R5:R6:R7 register.

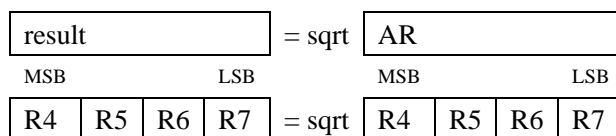


DMA instruction code: 0x1F(31)

Execution time: 58 - 67 clk

23.3.5 32-BIT REAL DATA SQUARE ROOT

The 32-bit real square root operation takes one argument which address is located in R7 register. Result of the operation is loaded to R4:R5:R6:R7 register.



DMA instruction code: 0x20(32)

Execution time: 50 - 54 clk

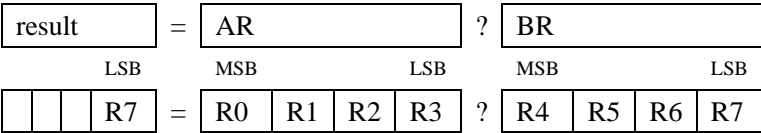
23.3.6 32-BIT REAL DATA COMPARISON

The 32-bit real comparison operation takes two double precision arguments. First argument is located in R0:R1:R2:R3 registers (Sign:Exponent:Mantissa) and the second argument is located in R4:R5:R6:R7 registers (Sign:Exponent:Mantissa).

(Sign:Exponent:Mantissa). Result of the operation is stored to R7 register.

R7.3	R7.2	R7.1	R7.0	Result	Description
0	0	0	0	0x0001	A>B
1	0	0	0	0x0000	A=B
0	0	0	1	0xFFFF	A<B
1	1	0	1	unchanged	Unordered

Table 1. Compare code description



DMA instruction code: 0x21(33)

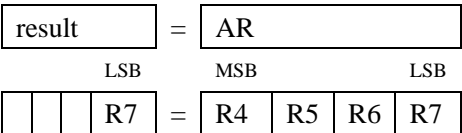
Execution time: 18 clk

23.3.7 32-BIT REAL DATA EXAMINE INPUT DATA

The 32-bit real examine operation takes one argument which address is located in R4:R5:R6:R7 registers. Result of the operation is stored to R7 register.

R7[7:4]	R7.3	R7.2	R7.1	R7.0	Description
0000	0	0	0	0	Plus NAN
0000	0	0	1	1	Minus NAN
0000	0	1	0	0	Plus Normal
0000	0	1	1	0	Minus Normal
0000	0	1	0	1	Plus Infinity
0000	0	1	1	1	Minus Infinity
0000	1	0	0	0	Plus Zero
0000	1	0	1	0	Minus Zero
0000	1	1	0	0	Plus Denormal
0000	1	1	1	0	Minus Denormal

Table 2. Examine condition code description



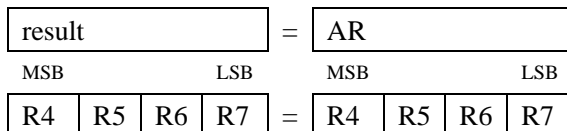
DMA instruction code: 0x22(34)

Execution time: 15 clk

23.4 FPMU TRIGONOMETRIC OPERATIONS

23.4.1 32-BIT REAL DATA SINE

The 32-bit real sine takes one argument which address is located in R0:R1:R2:R3 registers. Result of the operation is loaded to R4:R5:R6:R7 register.

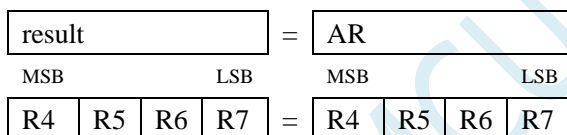


DMA instruction code: 0x2D(45)

Execution time: 32 - 270 clk

23.4.2 32-BIT REAL DATA COSINE

The 32-bit real cosine takes one argument which address is located in R7 register. Result of the operation is loaded to R4:R5:R6:R7 register.

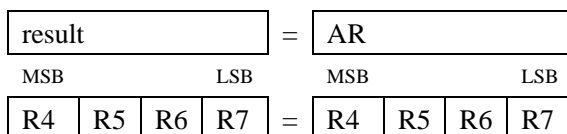


DMA instruction code: 0x2E(46)

Execution time: 32 - 270 clk

23.4.3 32-BIT REAL DATA TANGENT

The 32-bit real tangent takes one argument which address is located in R7 register. Result of the operation is loaded to R4:R5:R6:R7 register.

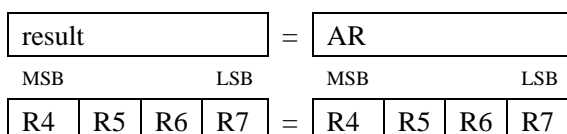


DMA instruction code: 0x2F(47)

Execution time: 58 - 258 clk

23.4.4 32-BIT REAL DATA ARCTANGENT

The 32-bit real arctangent takes one argument which address is located in R7 register. Result of the operation is loaded to R4:R5:R6:R7 register.



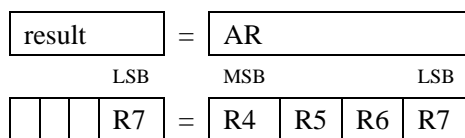
DMA instruction code: 0x30(48)

Execution time: 62 - 175 clk

23.5 FPMU DATA CONVERSION OPERATIONS

23.5.1 CONVERSION 32-BIT REAL DATA TO 8-BIT INTEGER

The conversion float to 8-bit integer takes one argument which address is located in R0:R1:R2:R3 registers. Result of the operation is stored to R7 register.

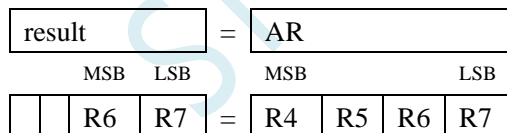


DMA instruction code: 0x23(35)

Execution time: 19 - 30 clk

23.5.2 CONVERSION 32-BIT REAL DATA TO 16-BIT INTEGER

The conversion float to 16-bit integer takes one argument which address is located in R0:R1:R2:R3 registers. Result of the operation is stored to R6 - R7 registers.

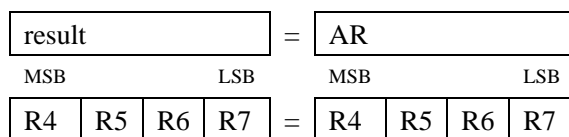


DMA instruction code: 0x24(36)

Execution time: 19 - 30 clk

23.5.3 CONVERSION 32-BIT REAL DATA TO 32-BIT INTEGER

The conversion float to 32-bit integer takes one argument which address is located in R0:R1:R2:R3 registers. Result of the operation is stored to R4 - R7 registers.

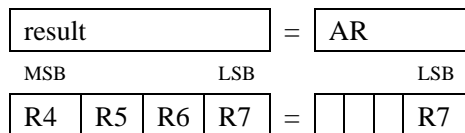


DMA instruction code: 0x25(37)

Execution time: 23 - 39 clk

23.5.4 CONVERSION 8-BIT INTEGER TO 32-BIT REAL DATA

The conversion 8-bit integer data to float takes one 2's complement integer argument which is located in R7 register. Result of the operation is loaded to R4:R5:R6:R7 registers.



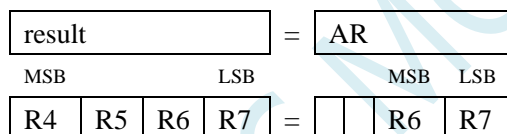
DMA instruction code: 0x27(39)

Execution time: 23 - 33 clk

23.5.5 CONVERSION 16-BIT INTEGER TO 32-BIT REAL

DATA

The conversion 16-bit integer data to float takes one 2's complement integer argument which is located in R6:R7 registers. Result of the operation is loaded to R4:R5:R6:R7 register.



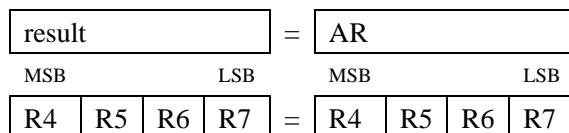
DMA instruction code: 0x28(40)

Execution time: 23 - 33 clk

23.5.6 CONVERSION 32-BIT INTEGER TO 32-BIT REAL

DATA

The conversion 32-bit integer data to float takes one 2's complement integer argument which is located in R4:R5:R6:R7 registers. Result of the operation is loaded to R4:R5:R6:R7 register.



DMA instruction code: 0x29(41)

Execution time: 24 - 33 clk

23.6 FPMU COPROCESSOR CONTROL OPERATIONS

23.6.1 INITIALIZE COPROCESSOR

This instruction behaves as hardware reset with one exception. The data registers AR, BR remain unchanged.

DMA instruction code: 0x31(49)

Execution time: 2 clk

23.6.2 CLEAR EXCEPTIONS

All status register bits are cleared.

DMA instruction code: 0x32(50)

Execution time: 4 clk

23.6.3 READ STATUS REGISTER

This instruction reads status register of coprocessor and stores to the R7 CPU register.

DMA instruction code: 0x33(51)

Execution time: 4 clk

23.6.4 WRITE STATUS REGISTER

This instruction writes operation argument located in R0 register to status register of coprocessor. The new state of coprocessor is stored back to the R7 register.

DMA instruction code: 0x34(52)

Execution time: 4 clk

23.6.5 READ CONTROL REGISTER

This instruction reads control register of coprocessor and stores it to the R0 CPU register.

DMA instruction code: 0x35(53)

Execution time: 4 clk

23.6.6 WRITE CONTROL REGISTER

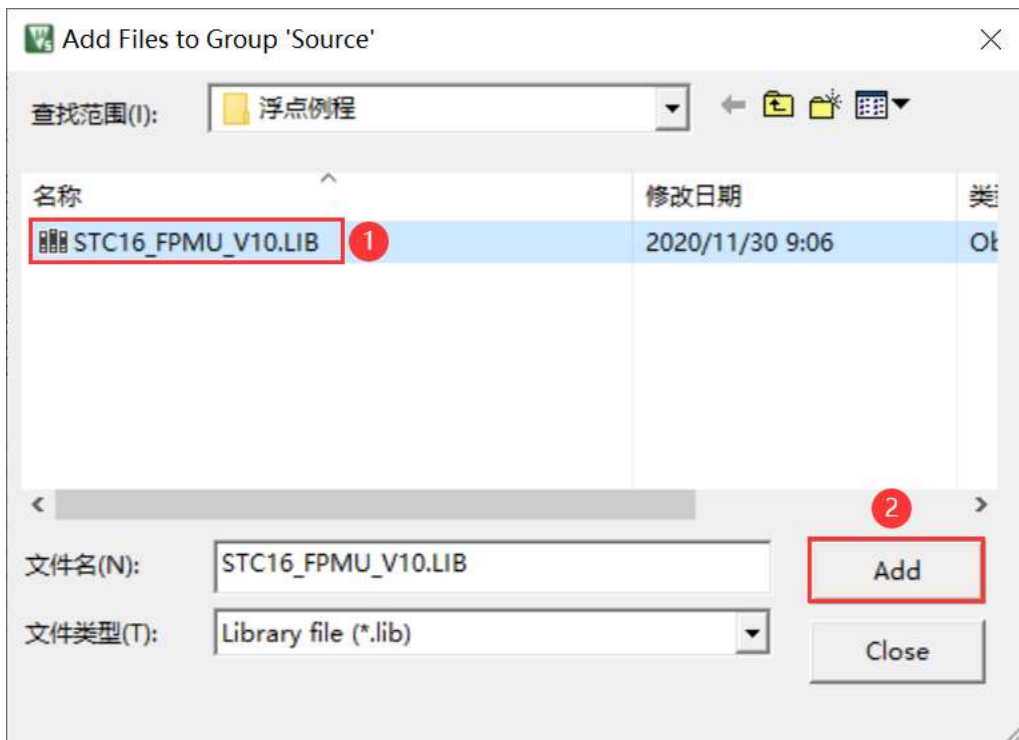
This instruction writes operation argument located in R0 to control register of coprocessor. The new state of coprocessor is stored back to the R7 register.

DMA instruction code: 0x36(54)

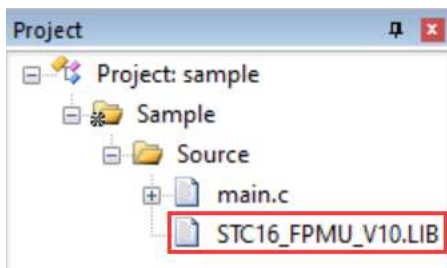
Execution time: 4 clk

23.7 范例程序

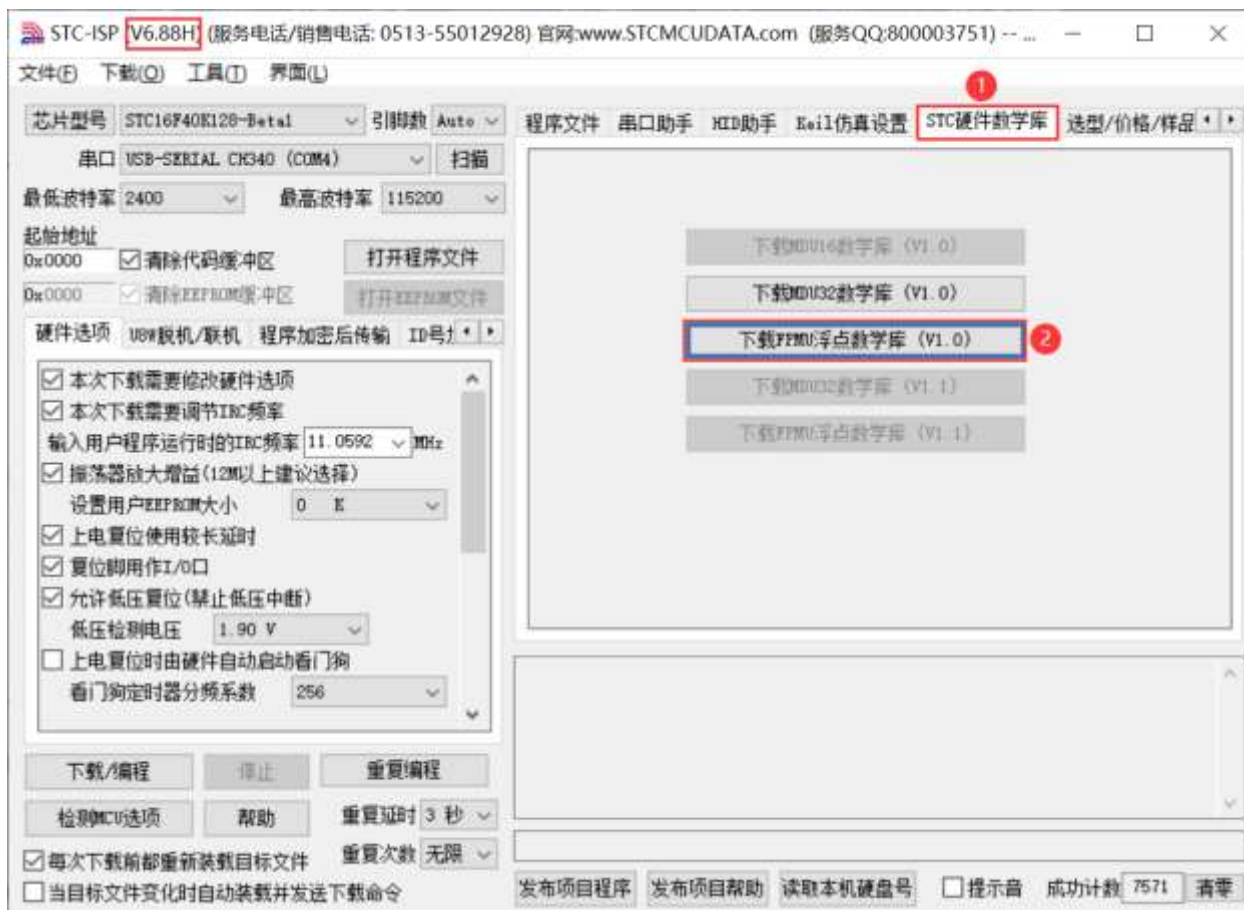
当要使用硬件浮点单元时，只要在 keil 项目中加入库文件“STC16_FPMU_Vxx.LIB”即可：



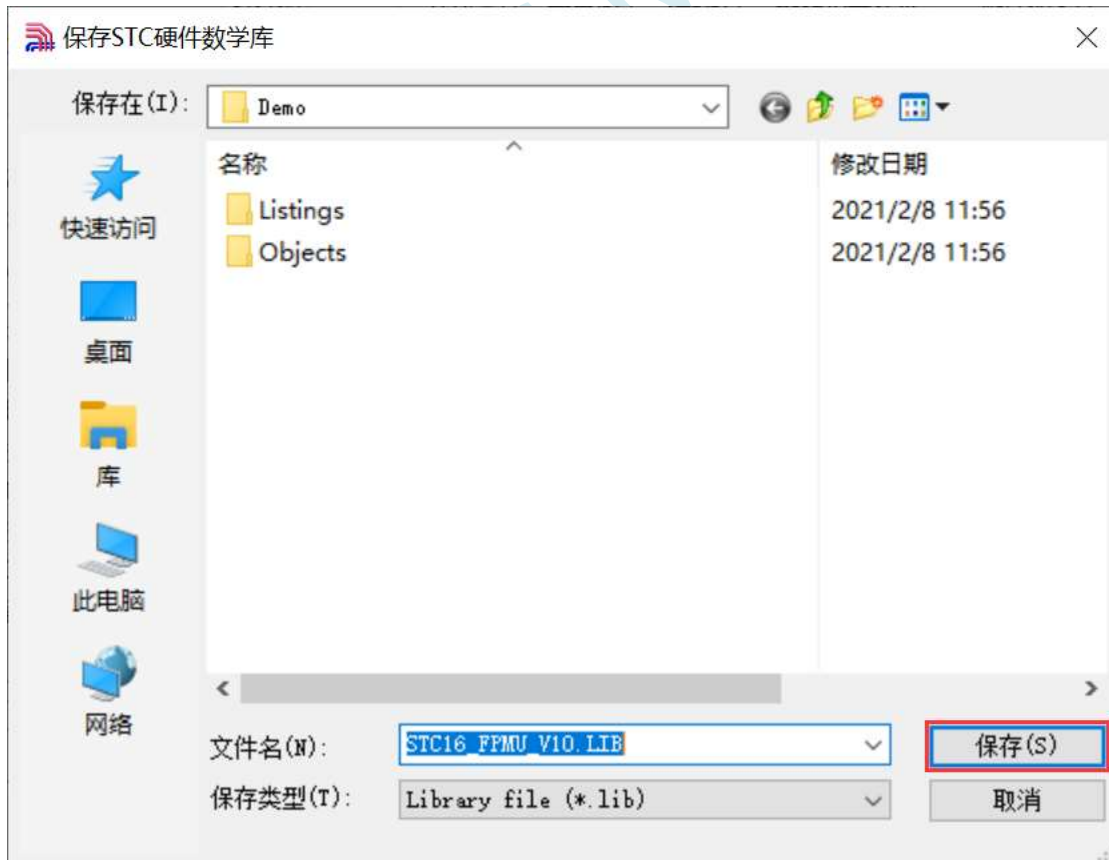
添加库文件到项目：



库文件获取方法：通过 STC-ISP 软件 V6.88H 及其以后版本，点击“STC 硬件数学库”标签获取。



点击需要下载的 FPMU 数学库按钮，选择保存的位置，点击“保存”按钮即可：



C 语言代码

//测试工作频率为24MHz

```
#include "stc16.h"           //头文件见附录
#include "intrins.h"
#include <math.h>

float data cfl1=3.9;
float data cfl2=5.1;
float data cfl3;

void main(void)
{
    WTST = 0;
    P0M1 = 0;   P0M0 = 0;           //设置为准双向口
    P1M1 = 0;   P1M0 = 0;           //设置为准双向口
    P2M1 = 0;   P2M0 = 0;           //设置为准双向口
    P3M1 = 0;   P3M0 = 0;           //设置为准双向口
    P4M1 = 0;   P4M0 = 0;           //设置为准双向口
    P5M1 = 0;   P5M0 = 0;           //设置为准双向口
    P6M1 = 0;   P6M0 = 0;           //设置为准双向口
    P7M1 = 0;   P7M0 = 0;           //设置为准双向口

    P10 = 0;
    cfl3 = cfl1*cfl2;
    cfl3 = cfl1/cfl2-cfl3;
    cfl3 = cfl1*cfl2+cfl3;
    cfl3 = cfl1/cfl2*sin(cfl3);
    cfl3 = cfl1/cfl2*cos(cfl3);
    cfl3 = cfl1/cfl2*tan(cfl3);
    cfl3 = cfl1/cfl2*sqrt(cfl3);
    cfl3 = cfl1/cfl2*atan(cfl3);
    P10 = 1;

    while(1);
}
```

附录A 指令集

A.1 INSTRUCTIONS SET BRIEF

A.1.1 BINARY MODE AND SOURCE MODE

Binary mode and source mode refer to two ways of assigning op-codes to the instruction set for the STC16 architecture. Depending on the application, binary mode or source mode may produce more efficient code. The binary mode refers to MCU51's standard op-codes. The source mode refers to MCU251 specific opcode set which expands instructions set by additional operations and addressing modes. The special mnemonic 0xA5 is used to make distinction between specific instruction in each mode. All unused op-codes are correctly decoded and executed as a NOP.

A.1.2 INSTRUCTION SET NOTES

The STC16 has five different addressing modes: immediate, direct, register, indirect and relative. In the immediate addressing mode the data is contained in the opcode. By direct addressing an eight bit address is a part of the opcode, by register addressing, a register is selected in the opcode for the operation. In the indirect addressing mode, a register is selected in the opcode to point to the address used by the operation. The relative addressing mode is used for jump instructions. The following tables give a survey about the instruction set cycles of the STC16 micro-controller core. One cycle is equal to one clock period. Table 1 and Table 2 contain notes for mnemonics used in Instruction set tables. Tables 3-7 show instruction hexadecimal codes, number of bytes and machine cycles that each instruction takes to execute.

Rn	Byte register R0-R7 of the currently selected bank
N	Byte register index 0-7
rrr	Binary representation of n
Rm	Byte register R0-R15 from register file
Rmd	Destination register
Rms	Source register
m,md,ms	Byte register index: m, md, ms = 0-15
ssss	Binary representation of m or md
SSSS	Binary representation of ms
WR	Word registers WR0,WR2,...,WR30
WRjd	Destination register
WRjs	Source register
@WRj	Indirect memory location (0x0000-0xFFFF) addressed by word registers
@WRj+dis	Indirect memory location (0x0000-0xFFFF) addressed by word registers + displacement value from 0 to 64kB Word
j, jd, js	register index: j, jd, js =0-30
tttt	Binary representation on j or jd
TTTT	Binary representation of js
DRk	Double word register DR0,DR4,...,DR28,DR56,DR60
DRkd	Destination register

DRks	Source register
@DRk	Indirect memory location (0x000000-0xFFFFF) addressed by double word registers
@DRk+dis k,	Indirect memory location (0x000000-0xFFFFF) addressed by double word registers + displacement value from 0 to 64kB Dword register index: k, kd, ks = 0, 4, 8, ..., 28, 56, 60
kd, ks uuuu	Binary representation of k or kd
UUUU	Binary representation of ks
dir8	128 internal RAM locations, any Special Function Registers
dir16	A 16-bit memory address (0x000000 – 0x00FFFF)
@Ri	Indirect memory location (0x00-0xFF) addressed by register R0 or R1
#data	8-bit constant included in instruction
#data16	16-bit constant included as bytes 2 and 3 of instruction
#0data16	32-bit constant with upper word filled with zeros; the lower word is included as bytes 2 and 3 of an instruction
#1data16	32-bit constant with upper word filled with ones; the lower word is included as bytes 2 and 3 of instruction
#short v v	A constant equal to 1,2, or 4 included in instruction Binary representation of #short
bit	A directly addressed bit in memory locations (0x20-0x7F) or in any defined SFR
bit51	A directly addressed bit (bit number = 0x00-0xFF) in memory or an SFR. Bits 0x00-0x7F are the 128 bits in byte locations 0x20-0x2F in the internal RAM. Bits 0x80-0xFF are the 128 bits in the 16 SFR's with addresses that end in 0 or 8: 0x80, 0x88, 0x90,..., 0xF0, 0xF8
A	Accumulator

Table 1. Notes on data addressing modes

addr24	A 24-bit destination address anywhere within 16MB address space. It is used for ECALL and EJMP instructions
addr16	Destination address for LCALL and LJMP may be anywhere within the 64-Kbyte of program memory address space.
addr11	Destination address for ACALL and AJMP will be within the same 2-Kbyte page of program memory as the first byte of the following instruction.
rel	SJMP and all conditional jumps include an 8-bit offset byte. Range is +127/-128 bytes relative to the first byte of the following instruction

Table 2. Notes on program addressing modes

-	The instruction does not modify the flag.
√	The instruction sets or clears the flag, as appropriate.
1	The instruction sets the flag.
0	The instruction clears the flag.

Table 3. Notes on flag description

A.1.3 INSTRUCTION SET BRIEF – FUNCTIONAL ORDER

Total instruction execution time depends on WTST (0x92) value. Each table below shows Cycles value in case when WTST=0. General formula to count total number of Cycles for each instruction is:

$$\text{instruction.cycles} = \text{Cycles} + \text{nrPRGACS} * \text{WTST}$$

where $\text{nrPRGACS} (0 \text{ or } 1)$

If instruction accessing XDM memory then CKCON[2:0] value multilied by nrXDMACS (1 or 2) should be added to count correct number of cycles.

$$\text{instruction.cycles} = \text{Cycles} + \text{nrPRGACS} * \text{WTST} + \text{nrXDMACS} * \text{CKCON}$$

The one Cycle is equal to one CLK period. The instructions specific for binary mode are marked as **red**. These instructions require 0xA5 prefix (ESC) before OPCODE, when are executed in source mode. The instructions specific for source mode are marked as **blue**. These instructions require 0xA5 prefix (ESC) before OPCODE, when are executed in binary mode. All other instructions are always available regardless of CPU mode.

A.1.3.1 ARITHMETIC OPERATIONS

Mnemonic	Description	Code	Bytes	Cycles
ADD A,Rn	Add register to accumulator	0x28-0x2F	1	1
ADD A,dir8	Add direct byte to accumulator	0x25	2	1
ADD A,@Ri	Add indirect RAM to accumulator	0x26-0x27	1	1
ADD A,#data	Add immediate data to accumulator	0x24	2	1
ADD Rm,Rm	Add byte register to byte register	0x2C	2	1
ADD WRj,WRj	Add word register to word register	0x2D	2	1
ADD DRk,DRk	Add dword register to dword register	0x2E	2	1
ADD reg,op2⁽³⁾	Add operand to Rm, DWj or DRk	0x2F	Note 1	Note 2
ADDC A,Rn	Add register to accumulator with carry flag	0x38-0x3F	1	1
ADDC A,dir8	Add direct byte to A with carry flag	0x35	2	1
ADDC A,@Ri	Add indirect RAM to A with carry flag	0x36-0x37	1	1
ADDC A,#data	Add immediate data to A with carry flag	0x34	2	1
SUBB A,Rn	Subtract register from A with borrow	0x98-0x9F	1	1
SUBB A,dir8	Subtract direct byte from A with borrow	0x95	2	1
SUBB A,@Ri	Subtract indirect RAM from A with borrow	0x96-0x97	1	1
SUBB A,#data	Subtract immediate data from A with borrow	0x94	2	1
SUB Rm,Rm	Subtract byte register from byte register	0x9C	2	1
SUB WRj,WRj	Subtract word register from word register	0x9D	2	1
SUB DRk,DRk	Subtract dword register from dword register	0x9E	2	1
SUB reg,op2⁽³⁾	Subtract operand from Rm, DWj or DRk	0x9F	Note 1	Note 2
CMP Rm,Rm	Compare two byte registers	0xBC	2	1
CMP WRj,WRj	Compare two word registers	0xBD	2	1
CMP DRk,DRk	Compare two dword registers	0xBE	2	1
CMP reg,op2⁽³⁾	Compare Rm, DWj or DRk with operand	0xBF	Note 1	Note 2
INC A	Increment accumulator	0x04	1	1
INC Rn	Increment register	0x08-0x0F	1	1
INC dir8	Increment direct byte	0x05	2	1
INC @Ri	Increment indirect RAM	0x06-0x07	1	1
INC reg,#short⁽³⁾	Increment Rm, DWj or DRk	0x0B	2	Note 2
DEC A	Decrement accumulator	0x14	1	1

DEC Rn	Decrement register	0x18-0x1F	1	1
DEC dir8	Decrement direct byte	0x15	1	1
DEC @Ri	Decrement indirect RAM	0x16-0x17	2	1
DEC reg,#short ⁽³⁾	Decrement Rm, DWj or DRk	0x0B	2	Note 2
INC DPTR	Increment data pointer	0xA3	1	1
MUL A,B	Multiply A and B	0xA4	1	1
MUL Rm,Rm	Multiply byte registers	0xAC	2	1
MUL WRj,WRj	Multiply word registers	0xAD	2	1
DIV A,B	Divide A by B	0x84	1	6
DIV Rm,Rm	Divide byte registers	0x8C	1	6
DIV WRj,WRj	Divide word registers	0x8D	1	10
DA A	Decimal adjust accumulator	0xD4	1	3

Table 4. Arithmetic operations

Note1: Bytes required for instruction depends on addressing mode determined by the following byte(s). Please refer to the instructions set details.

Note2: Cycles required for instruction depends on addressing mode determined by the following byte(s). Please refer to the instructions set details.

Note3: The operands and addressing modes depend on the following byte(s). All options are described in the instructions set details.

A.1.3.2 LOGIC OPERATIONS

Mnemonic	Description	Code	Bytes	Cycles
ANL A,Rn	AND register to accumulator	0x58-0x5F	1	1
ANL A,dir8	AND direct byte to accumulator	0x55	2	1
ANL A,@Ri	AND indirect RAM to accumulator	0x56-0x57	1	1
ANL A,#data	AND immediate data to accumulator	0x54	2	1
ANL dir8,A	AND accumulator to direct byte	0x52	2	1
ANL dir8,#data	AND immediate data to direct byte	0x53	3	1
ANL Rm,Rm	AND two byte registers	0x5C	2	1
ANL WRj,WRj	AND two word registers	0x5D	2	1
ANL reg,op2 ⁽³⁾	AND operand to Rm, DWj or DRk	0x5E	Note 1	Note 2
ORL A,Rn	OR register to accumulator	0x48-0x4F	1	1
ORL A,dir8	OR direct byte to accumulator	0x45	2	1
ORL A,@Ri	OR indirect RAM to accumulator	0x46-0x47	1	1
ORL A,#data	OR immediate data to accumulator	0x44	2	1
ORL dir8,A	OR accumulator to direct byte	0x42	2	1
ORL dir8,#data	OR immediate data to direct byte	0x43	3	1
ORL Rm,Rm	OR two byte registers	0x4C	2	1
ORL WRj,WRj	OR two word registers	0x4D	2	1
ORL reg,op2 ⁽³⁾	OR operand to Rm, DWj or DRk	0x4E	Note 1	Note 2

XRL A,Rn	Exclusive OR register to accumulator	0x68-0x6F	1	1
XRL A,dir8	Exclusive OR direct byte to accumulator	0x65	2	1
XRL A,@Ri	Exclusive OR indirect RAM to accumulator	0x66-0x67	1	1
XRL A,#data	Exclusive OR immediate data to accumulator	0x64	2	1
XRL dir8,A	Exclusive OR accumulator to direct byte	0x62	2	1
XRL dir8,#data	Exclusive OR immediate data to direct byte	0x63	3	1
XRL Rm,Rm	Exclusive OR two byte registers	0x6C	2	1
XRL WRj,WRj	Exclusive OR two word registers	0x6D	2	1
XRL reg,op2 ⁽³⁾	Exclusive OR operand to Rm, DWj or DRk	0x6E	Note 1	Note 2
CLR A	Clear accumulator	0xE4	1	1
CPL A	Complement accumulator	0xF4	1	1
RL A	Rotate accumulator left	0x23	1	1
RLC A	Rotate accumulator left through carry	0x33	1	1
RR A	Rotate accumulator right	0x03	1	1
RRC A	Rotate accumulator right through carry	0x13	1	1
SRA reg ⁽³⁾	Shift Rm or DWj right through the MSB	0x0E	2	1
SRL reg ⁽³⁾	Shift Rm or DWj right	0x1E	2	1
SLL reg ⁽³⁾	Shift Rm or DWj left	0x3E	2	1
SWAP A	Swap nibbles within the accumulator	0xC4	1	1

Table 5. Logic operations

Note1: Bytes required for instruction depends on addressing mode determined by the following byte(s). Please refer to the instructions set details.

Note2: Cycles required for instruction depends on addressing mode determined by the following byte(s). Please refer to the instructions set details.

Note3: The operands and addressing modes depend on the following byte(s). All options are described in the instructions set details.

A.1.3.3 BOOLEAN MANIPULATION

Mnemonic	Description	Code	Bytes	Cycles
CLR C	Clear carry flag	0xC3	1	1
CLR bit	Clear direct bit	0xC2	2	1
SETB C	Set carry flag	0xD3	1	1
SETB bit	Set direct bit	0xD2	2	1
CPL C	Complement carry flag	0xB3	1	1
CPL bit	Complement direct bit	0xB2	2	1
ANL C,bit	AND direct bit to carry flag	0x82	2	1
ANL C,/bit	AND complement of direct bit to carry	0xB0	2	1
ORL C,bit	OR direct bit to carry flag	0x72	2	1
ORL C,/bit	OR complement of direct bit to carry	0xA0	2	1
MOV C,bit	Move direct bit to carry flag	0xA2	2	1

MOV bit,C	Move carry flag to direct bit	0x92	2	1
Bit instr ⁽¹⁾	Set of bit instructions (MCU251 specific)	0xA9	3	1

Table 6. Boolean manipulation

A.1.3.4 DATA TRANSFERS

Mnemonic	Description	Code	Bytes	Cycles
MOV A,Rn	Move register to accumulator	0xE8-0xEF	1	1
MOV A,dir8	Move direct byte to accumulator	0xE5	2	1
MOV A,@Ri	Move indirect RAM to accumulator	0xE6-0xE7	1	1
MOV A,#data	Move immediate data to accumulator	0x74	2	1
MOV Rn,A	Move accumulator to register	0xF8-0xFF	1	1
MOV Rn,dir8	Move direct byte to register	0xA8-0xAF	2	1
MOV Rn,#data	Move immediate data to register	0x78-0x7F	2	1
MOV dir8,A	Move accumulator to direct byte	0xF5	2	1
MOV dir8,Rn	Move register to direct byte	0x88-0x8F	2	1
MOV dir8,dir8	Move direct byte to direct byte	0x85	3	1
MOV dir8,@Ri	Move indirect RAM to direct byte	0x86-0x87	2	1
MOV dir8,#data	Move immediate data to direct byte	0x75	3	1
MOV @Ri,A	Move accumulator to indirect RAM	0xF6-0xF7	1	1
MOV @Ri,dir8	Move direct byte to indirect RAM	0xA6-0xA7	2	1
MOV @Ri,#data	Move immediate data to indirect RAM	0x76-0x77	2	1
MOV Rm,Rm	Move byte register to byte register	0x7C	2	1
MOV WRj,WRj	Move word register to word register	0x7D	2	1
MOV DRk,DRk	Move dword register to dword register	0x7E	2	1
MOV reg,op2 ⁽³⁾	Move operand to Rm, DWj or DRk	0x5F	Note 1	Note 2
MOV WRj,@DRk	Move indirect (24-bit) RAM to WRj	0x0B	3	4
MOV @DRk,WRj	Move WRj to indirect (24-bit) RAM	0x1B	3	6
MOV Rm,@WRj+dis	Move indirect (16-bit) RAM with 16-bit displacement to Rm	0x09	4	1
MOV @WRj+dis,Rm	Move Rm to indirect (16-bit) RAM 16-bit displacement	0x19	4	1
MOV Rm,@DRk+dis	Move indirect (24-bit) RAM with 16-bit displacement to Rm	0x29	4	3
MOV @DRk+dis,Rm	Move Rm to indirect (24-bit) RAM with 16-bit displacement	0x39	4	4
MOV WRj,@WRj+dis	Move indirect (16-bit) RAM with 16-bit displacement to WRj	0x49	4	1
MOV @WRj+dis,WRj	Move WRj to indirect (16-bit) RAM with 16-bit displacement	0x59	4	3
MOV WRj,@DRk+dis	Move indirect (24-bit) RAM with 16-bit displacement to WRj	0x69	4	4
MOV @DRk+dis,WRj	Move WRj to indirect (24-bit) RAM with 16-bit displacement	0x79	4	6
MOV op1,reg ⁽³⁾	Move Rm, DWj or DRk to operand	0x7A	Note 1	Note 2
MOV DRk,#data16 ⁽⁴⁾	Move immediate 16-bit data to high word of dword register	0x7A	4	1
MOVZ WRj,Rm	Move byte register to word register with zero extension	0x0A	2	1

MOVS WRj,Rm	Move byte register to word register with sign extension	0x1A	2	1
MOV DPTR,#data16	Load 16-bit constant into active DPTR	0x90	3	1
MOVC A,@A+DPTR	Move code byte relative to DPTR to accumulator	0x93	1	4
MOVC A,@A+PC	Move code byte relative to PC to accumulator	0x83	1	3
MOVX A,@Ri	Move external RAM (8-bit address) to A	0xE2-0xE3	1	2*
MOVX A,@DPTR	Move external RAM (16-bit address) to A	0xE0	1	2*
MOVX @Ri,A	Move A to external memory (8-bit address)	0xF2-0xF3	1	2*
MOVX @DPTR,A	Move A to external memory (16-bit address)	0xF0	1	2*
PUSH dir8	Push direct byte onto IDM stack	0xC0	2	1
POP dir8	Pop direct byte from IDM stack	0xD0	2	1
PUSH op1 ⁽³⁾	Push operand onto IDM stack	0xCA	Note 1	Note 2
POP op1 ⁽³⁾	Pop operand from IDM stack	0xDA	Note 1	Note 2
XCH A,Rn	Exchange register with accumulator	0xC8-0xCF	1	1
XCH A,dir8	Exchange direct byte with accumulator	0xC5	2	1
XCH A,@Ri	Exchange indirect RAM with accumulator	0xC6-0xC7	1	1
XCHD A,@Ri	Exchange low-order nibble indirect RAM with A	0xD6-0xD7	1	3

Table 7. Data transfer

* Cycles depend on STRETCH register. Table shows values with STRETCH=0 (CKCON (0x8E) bits [2:0]).

Note1: Bytes required for instruction depends on addressing mode determined by the following byte(s). Please refer to the instructions set details.

Note2: Cycles required for instruction depends on addressing mode determined by the following byte(s). Please refer to the instructions set details.

Note3: The operands and addressing modes depend on the following byte(s). All options are described in the instructions set details.

Note4: The MOVH instruction has the same first byte of opcode as MOV op1,reg group. The instructions are distinguished by value on the second byte.

A.1.3.5 PROGRAM BRANCHES

Mnemonic	Description	Code	Bytes	Cycles
ACALL addr11	Absolute subroutine call	0x11-0xF1	2	3
LCALL addr16	Long direct subroutine call	0x12	3	3
ECALL addr24	Extended direct subroutine call	0x9A	4	3
ECALL @DRk	Extended indirect subroutine call	0x99	2	3
LCALL @WRj	Long indirect subroutine call	0x99	2	3
RET	Return from subroutine	0x22	1	3
ERET	Extended return from subroutine	0xAA	1	3
RETI	Return from interrupt	0x32	1	3
AJMP addr11	Absolute jump	0x01-0xE1	2	3
LJMP addr16	Long direct jump	0x02	3	3
EJMP addr24	Extended direct jump	0x8A	4	3

LJMP @WRj	Long indirect jump	0x89	2	3
EJMP @DRk	Extended indirect jump	0x89	2	3
SJMP rel	Short jump (relative address)	0x80	2	3
JMP @A+DPTR	Jump indirect relative to the DPTR	0x73	1	3
JZ rel	Jump if accumulator is zero	0x60	2	1/3
JNZ rel	Jump if accumulator is not zero	0x70	2	1/3
JC rel	Jump if carry flag is set	0x40	2	1/3
JNC	Jump if carry flag is not set	0x50	2	1/3
JB bit,rel	Jump if direct bit is set	0x20	3	1/3
JNB bit,rel	Jump if direct bit is not set	0x30	3	1/3
JBC bit,dir8 rel	Jump if direct bit is set and clear bit	0x10	3	1/3
JSLE rel	Jump if less than or equal (signed)	0x08	2	1/3
JSG rel	Jump if greater than (signed)	0x18	2	1/3
JLE rel	Jump if less than or equal	0x28	2	1/3
JG rel	Jump if greater than	0x38	2	1/3
JSL rel	Jump if less than (signed)	0x48	2	1/3
JSGE rel	Jump if greater than or equal (signed)	0x58	2	1/3
JE rel	Jump if equal	0x68	2	1/3
JNE rel	Jump if not equal	0x78	2	1/3
CJNE A,dir8 rel	Compare direct byte to A and jump if not equal	0xB5	3	1/3
CJNE A,#data rel	Compare immediate to A and jump if not equal	0xB4	3	1/3
CJNE Rn,#data rel	Compare immediate to reg. and jump if not equal	0xB8-0xBF	3	1/3
CJNE @Ri,#data rel	Compare immediate to ind. and jump if not equal	0xB6-0xB7	3	1/3
DJNZ Rn,rel	Decrement register and jump if not zero	0xD8-0xDF	2	1/3
DJNZ dir8,rel	Decrement direct byte and jump if not zero	0xD5	3	1/3
NOP	No operation	0x00	1	1

Table 8. Program branches

Note: Not taken conditional jump executes within 1 CLK period.

A.1.3.6 SPECIAL INSTRUCTIONS

Mnemonic	Description	Code	Bytes	Cycles
TRAP	Trap interrupt – implement as NOP	0xB9	1	1
ESC	Escape	0xA5	1	1

Table 9. Special instructions

A.1.4 INSTRUCTION SET BRIEF – HEXADECIMAL ORDER

注: STC16 采用的是 Source mode

A.1.4.1 BINARY MODE

Opcode	Mnemonic	Opcode	Mnemonic
00 H	NOP	30 H	JNB bit,rel
01 H	AJMP addr11	31 H	ACALL addr11
02 H	LJMP addr16	32 H	RETI
03 H	RR A	33 H	RLC A
04 H	INC A	34 H	ADDC A,#data
05 H	INC direct	35 H	ADDC A,direct
06 H	INC @R0	36 H	ADDC A,@R0
07 H	INC @R1	37 H	ADDC A,@R1
08 H	INC R0	38 H	ADDC A,R0
09 H	INC R1	39 H	ADDC A,R1
0A H	INC R2	3A H	ADDC A,R2
0B H	INC R3	3B H	ADDC A,R3
0C H	INC R4	3C H	ADDC A,R4
0D H	INC R5	3D H	ADDC A,R5
0E H	INC R6	3E H	ADDC A,R6
0F H	INC R7	3F H	ADDC A,R7
10 H	JBC bit,rel	40 H	JC rel
11 H	ACALL addr11	41 H	AJMP addr11
12 H	LCALL addr16	42 H	ORL direct,A
13 H	RRC A	43 H	ORL direct,#data
14 H	DEC A	44 H	ORL A,#data
15 H	DEC direct	45 H	ORL A,direct
16 H	DEC @R0	46 H	ORL A,@R0
17 H	DEC @R1	47 H	ORL A,@R1
18 H	DEC R0	48 H	ORL A,R0
19 H	DEC R1	49 H	ORL A,R1
1A H	DEC R2	4A H	ORL A,R2
1B H	DEC R3	4B H	ORL A,R3
1C H	DEC R4	4C H	ORL A,R4
1D H	DEC R5	4D H	ORL A,R5
1E H	DEC R6	4E H	ORL A,R6
1F H	DEC R7	4F H	ORL A,R7

20 H	JB bit,rel	50 H	JNC rel
21 H	AJMP addr11	51 H	ACALL addr11
22 H	RET	52 H	ANL direct,A
23 H	RL A	53 H	ANL direct,#data
24 H	ADD A,#data	54 H	ANL A,#data
25 H	ADD A,direct	55 H	ANL A,direct
26 H	ADD A,@R0	56 H	ANL A,@R0
27 H	ADD A,@R1	57 H	ANL A,@R1
28 H	ADD A,R0	58 H	ANL A,R0
29 H	ADD A,R1	59 H	ANL A,R1
2A H	ADD A,R2	5A H	ANL A,R2
2B H	ADD A,R3	5B H	ANL A,R3
2C H	ADD A,R4	5C H	ANL A,R4
2D H	ADD A,R5	5D H	ANL A,R5
2E H	ADD A,R6	5E H	ANL A,R6
2F H	ADD A,R7	5F H	ANL A,R7

Opcode	Mnemonic	Opcode	Mnemonic
60 H	JZ rel	90 H	MOV DPTR,#data16
61 H	AJMP addr11	91 H	ACALL addr11
62 H	XRL direct,A	92 H	MOV bit,C
63 H	XRL direct,#data	93 H	MOVC A,@A+DPTR
64 H	XRL A,#data	94 H	SUBB A,#data
65 H	XRL A,direct	95 H	SUBB A,direct
66 H	XRL A,@R0	96 H	SUBB A,@R0
67 H	XRL A,@R1	97 H	SUBB A,@R1
68 H	XRL A,R0	98 H	SUBB A,R0
69 H	XRL A,R1	99 H	SUBB A,R1
6A H	XRL A,R2	9A H	SUBB A,R2
6B H	XRL A,R3	9B H	SUBB A,R3
6C H	XRL A,R4	9C H	SUBB A,R4
6D H	XRL A,R5	9D H	SUBB A,R5
6E H	XRL A,R6	9E H	SUBB A,R6
6F H	XRL A,R7	9F H	SUBB A,R7
70 H	JNZ rel	A0 H	ORL C,bit
71 H	ACALL addr11	A1 H	AJMP addr11
72 H	ORL C,direct	A2 H	MOV C,bit
73 H	JMP @A+DPTR	A3 H	INC DPTR
74 H	MOV A,#data	A4 H	MUL AB
75 H	MOV direct,#data	A5 H	ESC

76 H	MOV @R0,#data	A6 H	MOV @R0,direct
77 H	MOV @R1,#data	A7 H	MOV @R1,direct
78 H	MOV R0.#data	A8 H	MOV R0,direct
79 H	MOV R1.#data	A9 H	MOV R1,direct
7A H	MOV R2.#data	AA H	MOV R2,direct
7B H	MOV R3.#data	AB H	MOV R3,direct
7C H	MOV R4.#data	AC H	MOV R4,direct
7D H	MOV R5.#data	AD H	MOV R5,direct
7E H	MOV R6.#data	AE H	MOV R6,direct
7F H	MOV R7.#data	AF H	MOV R7,direct
80 H	SJMP rel	B0 H	ANL C,bit
81 H	AJMP addr11	B1 H	ACALL addr11
82 H	ANL C,bit	B2 H	CPL bit
83 H	MOVC A,@A+PC	B3 H	CPL C
84 H	DIV AB	B4 H	CJNE A,#data,rel
85 H	MOV direct,direct	B5 H	CJNE A,direct,rel
86 H	MOV direct,@R0	B6 H	CJNE @R0,#data,rel
87 H	MOV direct,@R1	B7 H	CJNE @R1,#data,rel
88 H	MOV direct,R0	B8 H	CJNE R0,#data,rel
89 H	MOV direct,R1	B9 H	CJNE R1,#data,rel
8A H	MOV direct,R2	BA H	CJNE R2,#data,rel
8B H	MOV direct,R3	BB H	CJNE R3,#data,rel
8C H	MOV direct,R4	BC H	CJNE R4,#data,rel
8D H	MOV direct,R5	BD H	CJNE R5,#data,rel
8E H	MOV direct,R6	BE H	CJNE R6,#data,rel
8F H	MOV direct,R7	BF H	CJNE R7,#data,rel

Opcode	Mnemonic	Opcode	Mnemonic
C0 H	PUSH direct	E0 H	MOVX A,@DPTR
C1 H	AJMP addr11	E1 H	AJMP addr11
C2 H	CLR bit	E2 H	MOVX A,@R0
C3 H	CLR C	E3 H	MOVX A,@R1
C4 H	SWAP A	E4 H	CLR A
C5 H	XCH A, direct	E5 H	MOV A, direct
C6 H	XCH A,@R0	E6 H	MOV A,@R0
C7 H	XCH A,@R1	E7 H	MOV A,@R1
C8 H	XCH A,R0	E8 H	MOV A,R0
C9 H	XCH A,R1	E9 H	MOV A,R1
CA H	XCH A,R2	EA H	MOV A,R2
CB H	XCH A,R3	EB H	MOV A,R3

CC H	XCH A,R4	EC H	MOV A,R4
CD H	XCH A,R5	ED H	MOV A,R5
CE H	XCH A,R6	EE H	MOV A,R6
CF H	XCH A,R7	EF H	MOV A,R7
D0 H	POP direct	F0 H	MOVX @DPTR,A
D1 H	ACALL addr11	F1 H	ACALL addr11
D2 H	SETB bit	F2 H	MOVX @R0,A
D3 H	SETB C	F3 H	MOVX @R1,A
D4 H	DA A	F4 H	CPL A
D5 H	DJNZ direct, rel	F5 H	MOV direct, A
D6 H	XCHD A,@R0	F6 H	MOV @R0,A
D7 H	XCHD A,@R1	F7 H	MOV @R1,A
D8 H	DJNZ R0,rel	F8 H	MOV R0,A
D9 H	DJNZ R1,rel	F9 H	MOV R1,A
DA H	DJNZ R2,rel	FA H	MOV R2,A
DB H	DJNZ R3,rel	FB H	MOV R3,A
DC H	DJNZ R4,rel	FC H	MOV R4,A
DD H	DJNZ R5,rel	FD H	MOV R5,A
DE H	DJNZ R6,rel	FE H	MOV R6,A
DF H	DJNZ R7,rel	FF H	MOV R7,A

Table 10. Instruction set brief for binary mode in hexadecimal order

A.1.4.2 SOURCE MODE

Opcode	Mnemonic	Opcode	Mnemonic
00 H	NOP	30 H	JNB bit,rel
01 H	AJMP addr11	31 H	ACALL addr11
02 H	LJMP addr16	32 H	RETI
03 H	RR A	33 H	RLC A
04 H	INC A	34 H	ADDC A,#data
05 H	INC direct	35 H	ADDC A,direct
06 H	-	36 H	-
07 H	-	37 H	-
08 H	JSLE rel	38 H	JG rel
09 H	MOV Rm,@WRj+dis	39 H	MOV @DRk+dis,Rm
0A H	MOVZ WRj,Rm	3A H	-
0B H	INC R,#short MOV WRj,@DRk	3B H	-
0C H	-	3C H	-

0D H	-	3D H	-
0E H	SRA reg	3E H	SLL reg
0F H	-	3F H	-
10 H	JBC bit,rel	40 H	JC rel
11 H	ACALL addr11	41 H	AJMP addr11
12 H	LCALL addr16	42 H	ORL direct,A
13 H	RRC A	43 H	ORL direct,#data
14 H	DEC A	44 H	ORL A,#data
15 H	DEC direct	45 H	ORL A,direct
16 H	-	46 H	-
17 H	-	47 H	-
18 H	JSG rel	48 H	JSL rel
19 H	MOV @WRj+dis,Rm	49 H	MOV WRj,@WRj+dis
1A H	MOVS WRj,Rm	4A H	-
1B H	DEC R,#short MOV @DRk, WRj	4B H	-
1C H	-	4C H	ORL Rm,Rm
1D H	-	4D H	ORL WRj,WRj
1E H	SRL reg	4E H	ORL reg,op2
1F H	-	4F H	-
20 H	JB bit.rel	50 H	JNC rel
21 H	AJMP addr11	51 H	ACALL addr11
22 H	RET	52 H	ANL direct,A
23 H	RL A	53 H	ANL direct,#data
24 H	ADD A,#data	54 H	ANL A,#data
25 H	ADD A,direct	55 H	ANL A,direct
26 H	-	56 H	-
27 H	-	57 H	-
28 H	JLE rel	58 H	JSGE rel
29 H	MOV Rm,@DRk+dis	59 H	MOV @WRj+dis,WRj
2A H	-	5A H	-
2B H	-	5B H	-
2C H	ADD Rm,Rm	5C H	ANL Rm,Rm
2D H	ADD WRj,WRj	5D H	ANL WRj,WRj
2E H	ADD reg,op2	5E H	ANL reg,op2
2F H	ADD DRk,DRk	5F H	-

Opcode	Mnemonic	Opcode	Mnemonic
60 H	JZ rel	90 H	MOV DPTR,#data16
61 H	AJMP addr11	91 H	ACALL addr11
62 H	XRL direct,A	92 H	MOV bit,C

63 H	XRL direct,#data	93 H	MOVC A,@A+DPTR
64 H	XRL A,#data	94 H	SUBB A,#data
65 H	XRL A,direct	95 H	SUBB A,direct
66 H	-	96 H	-
67 H	-	97 H	-
68 H	JE rel	98 H	-
69 H	MOV WRj,@DRk+dis	99 H	LCALL @WRj ECALL @DRk
6A H	-	9A H	-
6B H	-	9B H	-
6C H	XRL Rm,Rm	9C H	SUB Rm,Rm
6D H	XRL WRj,WRj	9D H	SUB WRj,WRj
6E H	XRL reg,op2	9E H	SUB reg,op2
6F H	-	9F H	SUB DRk,DRk
70 H	JNZ rel	A0 H	ORL C,bit
71 H	ACALL addr11	A1 H	AJMP addr11
72 H	ORL C,direct	A2 H	MOV C,bit
73 H	JMP @A+DPTR	A3 H	INC DPTR
74 H	MOV A,#data	A4 H	MUL AB
75 H	MOV direct,#data	A5 H	ESC
76 H	-	A6 H	-
77 H	-	A7 H	-
78 H	JNE	A8 H	-
79 H	MOV @DRk+dis,WRj	A9 H	Bit instructions
7A H	MOVH DRk,#data16 MOV op1,reg	AA H	ERET
7B H	-	AB H	-
7C H	MOV Rm,Rm	AC H	MUL Rm,Rm
7D H	MOV WRj,WRj	AD H	MUL WRj,WRj
7E H	MOV reg,op2	AE H	-
7F H	MOV DRk,DRk	AF H	-
80 H	SJMP rel	B0 H	ANL C,bit
81 H	AJMP addr11	B1 H	ACALL addr11
82 H	ANL C,bit	B2 H	CPL bit
83 H	MOVC A,@A+PC	B3 H	CPL C
84 H	DIV AB	B4 H	CJNE A,#data,rel
85 H	MOV direct,direct	B5 H	CJNE A,direct,rel
86 H	-	B6 H	-
87 H	-	B7 H	-
88 H	-	B8 H	TRAP

89 H	LJMP @WRj EJMP @DRk	B9 H	-
8A H	EJMP addr24	BA H	-
8B H	-	BB H	-

8C H	DIV Rm,Rm	BC H	CMP Rm,Rm
8D H	DIV WRj,WRj	BD H	CMP WRj,WRj
8E H	-	BE H	CMP reg,op2
8F H	-	BF H	CMP DRk,DRk

Opcode	Mnemonic	Opcode	Mnemonic
C0 H	PUSH direct	E0 H	MOVX A,@DPTR
C1 H	AJMP addr11	E1 H	AJMP addr11
C2 H	CLR bit	E2 H	MOVX A,@R0
C3 H	CLR C	E3 H	MOVX A,@R1
C4 H	SWAP A	E4 H	CLR A
C5 H	XCH A, direct	E5 H	MOV A, direct
C6 H	-	E6 H	-
C7 H	-	E7 H	-
C8 H	-	E8 H	-
C9 H	-	E9 H	-
CA H	PUSH op1	EA H	-
CB H	-	EB H	-
CC H	-	EC H	-
CD H	-	ED H	-
CE H	-	EE H	-
CF H	-	EF H	-
D0 H	POP direct	F0 H	MOVX @DPTR,A
D1 H	ACALL addr11	F1 H	ACALL addr11
D2 H	SETB bit	F2 H	MOVX @R0,A
D3 H	SETB C	F3 H	MOVX @R1,A
D4 H	DA A	F4 H	CPL A
D5 H	DJNZ direct, rel	F5 H	MOV direct, A
D6 H	-	F6 H	-
D7 H	-	F7 H	-
D8 H	-	F8 H	-
D9 H	-	F9 H	-
DA H	POP op1	FA H	-
DB H	-	FB H	-
DC H	-	FC H	-

DD H	-	FD H	-
DE H	-	FE H	-
DF H	-	FF H	-

Table 11. Instruction set brief for source mode in hexadecimal order

STC MCU

A.2 INSTRUCTIONS SET DETAILS

ACALL <addr11>

Function: Absolute call

Description: ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the stack pointer twice. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC, opcode bits 7-5, and the second byte of the instruction. The subroutine called must therefore start within the same 2K block of program memory as the first byte of the instruction following ACALL. No flags are affected.

Flags:	CY	AC	OV	N	Z
	—	—	—	—	—

Binary Mode

Source Mode

Bytes:

2

2

Cycles:

3

3

Hex:

Encoding

Encoding

[Encoding] 11H, 31H, 51H, 71H, 91H, B1H, D1H, F1H

A10	A9	A8	1	0	0	0	1	A7	A6	A5	A4	A3	A2	A1	A0
-----	----	----	---	---	---	---	---	----	----	----	----	----	----	----	----

Operation:

ACALL

$(PC) \leftarrow (PC) + 2$

$(SP) \leftarrow (SP) + 1$

$((SP)) \leftarrow (PC.7:0)$

$(SP) \leftarrow (SP) + 1$

$((SP)) \leftarrow (PC.15:8)$

$(PC.10:0) \leftarrow \text{page address}$

ADD <dest>,<src>

Function: Adds source to the destination operand.

Description: Adds the source operand to the destination operand, which can be a register or the accumulator, leaving the result in the register or accumulator. If there is a carryout of bit 7 (CY), the CY flag is set. If byte variables are added, and if there is a carry out of bit 3 (AC), the AC flag is set. For addition of unsigned integers, the CY flag indicates that an overflow occurred. If there is a carry out of bit 6 but not out of bit 7, or a carry out of bit 7 but not bit 6, the OV flag is set. When adding signed integers, the OV flag indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands. Bit 6 and bit 7 in this description refer to the most significant byte of the operand (8, 16, or 32bit). The result affects N and Z flags. The source operand allowed addressing modes are register, direct, register-indirect, and immediate.

ADD A,Rn

Operation: (PC) $\leftarrow (PC) + 1$

(A) $\leftarrow (A) + (Rn)$

Flags:	CY	AC	OV	N	Z
	✓	✓	✓	✓	✓

[Encoding] 28H – 2FH

0	0	1	0	1	r	r	r
---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes:

1

2

Cycles:

1

1

Hex:

Encoding

[A5]Encoding

ADD A,Direct

Operation:

(PC)

 $\leftarrow (PC) + 2$

(A)

 $\leftarrow (A) + (\text{direct})$

Flags:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Encoding] 25H

0	0	1	0	0	1	0	1	direct address
---	---	---	---	---	---	---	---	----------------

Binary Mode

Source Mode

Bytes:

2

2

Cycles:

1

1

Hex:

Encoding

Encoding

ADD A,@Ri

Operation:

(PC)

 $\leftarrow (PC) + 1$

(A)

 $\leftarrow (A) + ((Ri))$

Flags:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Encoding] 26H, 27H

0	0	1	0	0	1	1	i
---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes:

1

2

Cycles:

1

1

Hex:

Encoding

[A5]Encoding

ADD A,#DATA

Operation:

(PC)

 $\leftarrow (PC) + 2$

(A)

 $\leftarrow (A) + \#data$

Flags:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Encoding] 24H

0	0	1	0	0	1	0	0	immediate data
---	---	---	---	---	---	---	---	----------------

Binary Mode

Source Mode

Bytes:

2

2

Cycles:

1

1

Hex:

Encoding

Encoding

ADD Rmd,Rms

Operation:

(PC)

 $\leftarrow (PC) + 2$

(Rmd)

 $\leftarrow (Rms) + (Rmd)$

Flags:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Encoding] 2CH

0	0	1	0	1	1	0	0	s	s	s	s	S	S	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode**Source Mode****Bytes:**

3

2

Cycles:

1

1

Hex:

[A5]Encoding

Encoding

ADD WRjd,WRjs**Operation:**

(PC)

 $\leftarrow (PC) + 2$

(WRjd)

 $\leftarrow (WRjs) + (WRjd)$ **Flags:**

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[Encoding] 2DH

0	0	1	0	1	1	0	1	t	t	t	t	T	T	T	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode**Source Mode****Bytes:**

3

2

Cycles:

1

1

Hex:

[A5]Encoding

Encoding

ADD DRkd,DRks**Operation:**

(PC)

 $\leftarrow (PC) + 2$

(DRkd)

 $\leftarrow (DRks) + (DRkd)$ **Flags:**

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[Encoding] 2FH

0	0	1	0	1	1	1	1	u	u	u	u	U	U	U	U
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode**Source Mode****Bytes:**

3

2

Cycles:

1

1

Hex:

[A5]Encoding

Encoding

ADD Rm,#DATA**Operation:**

(PC)

 $\leftarrow (PC) + 3$

(Rm)

 $\leftarrow (Rm) + \#data$ **Flags:**

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Encoding] 2E(s)0H

0	0	1	0	1	1	1	0	s	s	s	s	0	0	0	0	#data
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-------

Binary Mode**Source Mode****Bytes:**

4

3

Cycles:

1

1

Hex:

[A5]Encoding

Encoding

ADD WRj,#DATA16

Operation: (PC) $\leftarrow (PC) + 4$

(WRj) $\leftarrow (WRj) + \#data16$

Flags:	CY	AC	OV	N	Z
	✓	-	✓	✓	✓

[Encoding] 2E(t)4H

0	0	1	0	1	1	1	0	t	t	t	t	0	1	0	0	#data hi	#data lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode

Source Mode

Bytes: 5 4

Cycles: 1 1

Hex: [A5]Encoding Encoding

ADD DRk,#0DATA16

Operation: (PC) $\leftarrow (PC) + 4$

(DRk) $\leftarrow (DRk) + \#data16$

Flags:	CY	AC	OV	N	Z
	✓	-	✓	✓	✓

[Encoding] 2E(u)8H

0	0	1	0	1	1	1	0	u	u	u	u	1	0	0	0	#data hi	#data lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode

Source Mode

Bytes: 5 4

Cycles: 1 1

Hex: [A5]Encoding Encoding

ADD Rm,DIR8

Operation: (PC) $\leftarrow (PC) + 3$

(Rm) $\leftarrow (Rm) + (dir8)$

Flags:	CY	AC	OV	N	Z
	✓	✓	✓	✓	✓

[Encoding] 2E(s)1H

0	0	1	0	1	1	1	0	s	s	s	s	0	0	0	1	dir8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

Binary Mode

Source Mode

Bytes: 4 3

Cycles: 1 1

Hex: [A5]Encoding Encoding

ADD WRj,DIR8

Operation: (PC) $\leftarrow (PC) + 3$

(WRj) $\leftarrow (WRj) + (dir8)$

Flags:	CY	AC	OV	N	Z
	✓	-	✓	✓	✓

[Encoding] 2E(t)5H

0	0	1	0	1	1	1	0	t	t	t	t	0	1	0	1	dir8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

Binary Mode

Source Mode

Bytes: 4 3

Cycles: 1 (2 for SFR) 1 (2 for SFR)

Hex: [A5]Encoding Encoding

ADD Rm,DIR16**Operation:** (PC) $\leftarrow (PC) + 4$ (Rm) $\leftarrow (Rm) + (dir16)$

Flags:	CY	AC	OV	N	Z
	✓	✓	✓	✓	✓

[Encoding] 2E(s)3H

0	0	1	0	1	1	1	0	s	s	s	s	0	0	1	1	dir16 hi	dir16 lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode**Source Mode****Bytes:** 5

4

Cycles: 1

1

Hex: [A5]Encoding

Encoding

ADD WRj,DIR16**Operation:** (PC) $\leftarrow (PC) + 4$ (WRj) $\leftarrow (WRj) + (dir16)$

Flags:	CY	AC	OV	N	Z
	✓	-	✓	✓	✓

[Encoding] 2E(t)7H

0	0	1	0	1	1	1	0	t	t	t	t	0	1	1	1	dir16 hi	dir16 lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode**Source Mode****Bytes:** 5

4

Cycles: 2

2

Hex: [A5]Encoding

Encoding

ADD Rm,@DRk**Operation:** (PC) $\leftarrow (PC) + 3$ (Rm) $\leftarrow (Rm) + ((DRk))$

Flags:	CY	AC	OV	N	Z
	✓	✓	✓	✓	✓

[Encoding] 2E(u)B(s)0H

0	0	1	0	1	1	1	0	u	u	u	u	1	0	1	1	s	s	s	s	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode**Source Mode****Bytes:** 4

3

Cycles: 3

3

Hex: [A5]Encoding

Encoding

ADDC A,<src-byte>**Function:** Adds A and the source operand, then adds one (1) if CY is set, and puts the result in A.

Description: ADDC simultaneously adds the byte variable indicated, the carry flag and the accumulator contents, leaving the result in the accumulator. The carry and auxiliary carry flags are set, respectively, if there is a carry out of bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred. OV is set if there is a carry out of bit 6 but not out of bit 7, or a carry out of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands. The N and Z flags also are affected according to the result. Four source

operand-addressing modes are allowed: register= direct, register- indirect, or immediate.

Flags:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

ADDC A,Rn

Operation: (PC) $\leftarrow (PC) + 1$
(A) $\leftarrow (A) + (C) + (Rn)$

[Encoding] 38H – 3FH

0	0	1	1	1	r	r	r
---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 1 2

Cycles: 1 1

Hex: Encoding [A5]Encoding

ADDC A,DIRECT

Operation: (PC) $\leftarrow (PC) + 2$
(A) $\leftarrow (A) + (C) + (\text{direct})$

[Encoding] 35H

0	0	1	1	0	1	0	1	direct address
---	---	---	---	---	---	---	---	----------------

Binary Mode **Source Mode**

Bytes: 2 2

Cycles: 1 1

Hex: Encoding Encoding

ADDC A,@Ri

Operation: (PC) $\leftarrow (PC) + 1$
(A) $\leftarrow (A) + (C) + ((Ri))$

[Encoding] 36H, 37H

0	0	1	1	0	1	1	i
---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 1 2

Cycles: 1 1

Hex: Encoding [A5]Encoding

ADDC A,#DATA

Operation: (PC) $\leftarrow (PC) + 2$
(A) $\leftarrow (A) + (C) + \#data$

[Encoding] 34H

0	0	1	1	0	1	0	0	immediate data
---	---	---	---	---	---	---	---	----------------

Binary Mode **Source Mode**

Bytes: 2 2

Cycles: 1 1

Hex: Encoding Encoding

AJMP addr11

Function: Absolute jump

Description: AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (after incrementing the PC twice), op code bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2K block of program memory as the first byte of the instruction following AJMP.

Flags:	CY	AC	OV	N	Z
	—	—	—	—	—

Operation: (PC) \leftarrow (PC) + 2
(PC10-0) \leftarrow page address

[Encoding] 01H, 21H, 41H, 61H, 81H, A1H, C1H, E1H

a10	a9	a8	0	0	0	0	1	a7	a6	a5	a4	a3	a2	a1	a0
-----	----	----	---	---	---	---	---	----	----	----	----	----	----	----	----

Binary Mode **Source Mode**

Bytes: 2 2

Cycles: 3 3

Hex: Encoding Encoding

ANL <dest>,<src>

Function: Logical AND for byte operands

Description: Performs the bitwise logical-AND operation between the specified variables and stores the results in the destination variable. The two operands allow 10 addressing mode combinations. When the destination is the register or accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data. The N and Z flags are affected according to the result.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

ANL A,Rn

Operation: (PC) \leftarrow (PC) + 1
(A) \leftarrow (A) and (Rn)

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 58H – 5FH

0	1	0	1	1	r	r	r
---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 1 2

Cycles: 1 1

Hex: Encoding [A5]Encoding

ANL A,Direct

Operation: (PC) \leftarrow (PC) + 2
(A) \leftarrow (A) and (direct)

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 55H

0	1	0	1	0	1	0	1	direct address
---	---	---	---	---	---	---	---	----------------

	Binary Mode	Source Mode
Bytes:	2	2
Cycles:	1	1
Hex:	Encoding	Encoding

ANL A,@Ri

Operation: (PC) $\leftarrow (PC) + 1$
 (A) $\leftarrow (A) \text{ and } ((Ri))$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 56H, 57H

0	1	0	1	0	1	1	i
---	---	---	---	---	---	---	---

	Binary Mode	Source Mode
Bytes:	1	2
Cycles:	1	1
Hex:	Encoding	[A5]Encoding

ANL A,#DATA

Operation: (PC) $\leftarrow (PC) + 2$
 (A) $\leftarrow (A) \text{ and } \#data$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 54H

0	1	0	1	0	1	0	0	immediate data
---	---	---	---	---	---	---	---	----------------

	Binary Mode	Source Mode
Bytes:	2	2
Cycles:	1	1
Hex:	Encoding	Encoding

ANL Direct,A

Operation: (PC) $\leftarrow (PC) + 2$
 (direct) $\leftarrow (\text{direct}) \text{ and } (A)$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 52H

0	1	0	1	0	0	1	0	direct address
---	---	---	---	---	---	---	---	----------------

	Binary Mode	Source Mode
Bytes:	2	2
Cycles:	1	1
Hex:	Encoding	Encoding

ANL Direct,#DATA

Operation: (PC) $\leftarrow (PC) + 3$
 (direct) $\leftarrow (\text{direct}) \text{ and } \#data$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 53H

0	1	0	1	0	0	1	1	direct address	immediate data
---	---	---	---	---	---	---	---	----------------	----------------

Binary Mode

Source Mode

Bytes: 3 3

Cycles: 1 1

Hex: Encoding Encoding

ANL Rmd,Rms

Operation: (PC) $\leftarrow (PC) + 2$
 (Rmd) $\leftarrow (Rms)$ and (Rmd)

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 5CH

0	1	0	1	1	1	0	0	s	s	s	s	S	S	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 3 2

Cycles: 1 1

Hex: [A5]Encoding Encoding

ANL WRjd,WRjs

Operation: (PC) $\leftarrow (PC) + 2$
 (WRjd) $\leftarrow (WRjs)$ and (WRjd)

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 2DH

0	0	1	0	1	1	1	0	1	t	t	t	t	T	T	T	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 3 2

Cycles: 1 1

Hex: [A5]Encoding Encoding

ANL Rm,#DATA

Operation: (PC) $\leftarrow (PC) + 3$
 (Rm) $\leftarrow (Rm)$ and #data

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 5E(s)0H

0	1	0	1	1	1	1	0	s	s	s	s	0	0	0	0	#data
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-------

Binary Mode

Source Mode

Bytes: 4 3

Cycles: 1 1

Hex: [A5]Encoding Encoding

ANL WRj,#DATA16

Operation: (PC) $\leftarrow (PC) + 4$
 (WRj) $\leftarrow (WRj)$ and #data16

Flags:	CY	AC	OV	N	Z
--------	----	----	----	---	---

	-	-	-	✓	✓
[Encoding]	5E(t)4H				
	0	1	0	1	1 1 1 0 t t t t 0 1 0 0 #data hi #data lo
	Binary Mode		Source Mode		
Bytes:	5		4		
Cycles:	1		1		
Hex:	[A5]Encoding		Encoding		

ANL Rm,DIR8

Operation:	(PC)	$\leftarrow (PC) + 3$			
	(Rm)	$\leftarrow (Rm)$ and (dir8)			
Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓
[Encoding]	5E(s)1H				
	0	1	0	1	1 1 1 0 s s s s 0 0 0 1 dir8
	Binary Mode		Source Mode		
Bytes:	4		3		
Cycles:	1		1		
Hex:	[A5]Encoding		Encoding		

ANL WRj,DIR8

Operation:	(PC)	$\leftarrow (PC) + 3$			
	(WRj)	$\leftarrow (WRj)$ and (dir8)			
Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓
[Encoding]	5E(t)5H				
	0	1	0	1	1 1 1 0 t t t t 0 1 0 1 dir8
	Binary Mode		Source Mode		
Bytes:	4		3		
Cycles:	1 (2 for SFR)		1 (2 for SFR)		
Hex:	[A5]Encoding		Encoding		

ANL Rm,DIR16

Operation:	(PC)	$\leftarrow (PC) + 4$			
	(Rm)	$\leftarrow (Rm)$ and (dir16)			
Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓
[Encoding]	5E(s)3H				
	0	1	0	1	1 1 1 0 s s s s 0 0 1 1 dir16 hi dir16 lo
	Binary Mode		Source Mode		
Bytes:	5		4		
Cycles:	1		1		
Hex:	[A5]Encoding		Encoding		

ANL WRj,DIR16

Operation:	(PC)	$\leftarrow (PC) + 4$
------------	------	-----------------------

	(WRj)	$\leftarrow (\text{WRj})$ and (dir16)			
Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓
[Encoding]	5E(t)7H				
	0	1	0	1	1 1 1 0 t t t t 0 1 1 1 dir16 hi dir16 lo
	Binary Mode		Source Mode		
Bytes:	5		4		
Cycles:	2		2		
Hex:	[A5]Encoding		Encoding		

ANL Rm, @WRj

Operation:	(PC)	$\leftarrow (\text{PC}) + 3$			
	(Rm)	$\leftarrow (\text{Rm})$ and ((WRj))			
Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓
[Encoding]	5E(t)9(s)0H				
	0	1	0	1	1 1 1 0 t t t t 1 0 0 1 s s s s 0 0 0 0
	Binary Mode		Source Mode		
Bytes:	4		3		
Cycles:	1		1		
Hex:	[A5]Encoding		Encoding		

ANL Rm, @DRk

Operation:	(PC)	$\leftarrow (\text{PC}) + 3$			
	(Rm)	$\leftarrow (\text{Rm})$ and ((DRk))			
Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓
[Encoding]	5E(u)B(s)0H				
	0	1	0	1	1 1 1 0 u u u u 1 0 1 1 s s s s 0 0 0 0
	Binary Mode		Source Mode		
Bytes:	4		3		
Cycles:	3		3		
Hex:	[A5]Encoding		Encoding		

ANL C, <src-bit>

Function:	Logical AND for bit operands
Description:	If the Boolean value of the source bit is logic 0 then clear the carry flag; otherwise leave the carry flag in its current state. A slash ("/" preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected. Only direct bit addressing is allowed for the source operand.

ANL C, bit51

Operation:	(PC)	$\leftarrow (\text{PC}) + 2$			
	(A)	$\leftarrow (\text{C})$ and (bit51)			
Flags:	CY	AC	OV	N	Z
	✓	-	-	-	-

[Encoding] 82H

1	0	0	0	0	0	1	0	bit address
---	---	---	---	---	---	---	---	-------------

Binary Mode

Source Mode

Bytes: 2

2

Cycles: 1

1

Hex: Encoding

Encoding

ANL C,/bit51

Operation: (PC) $\leftarrow (PC) + 2$ (B) $\leftarrow (C) \text{ and } /(bit51)$

Flags:

CY	AC	OV	N	Z
✓	-	-	-	-

[Encoding] 80H

1	0	1	1	0	0	0	0	bit address
---	---	---	---	---	---	---	---	-------------

Binary Mode

Source Mode

Bytes: 2

2

Cycles: 1

1

Hex: Encoding

Encoding

ANL C,bit

Operation: (PC) $\leftarrow (PC) + 3$ (A) $\leftarrow (C) \text{ and } (bit)$

Flags:

CY	AC	OV	N	Z
✓	-	-	-	-

[Encoding] A98(y)H

1	0	1	0	1	0	0	1	1	0	0	0	0	y	y	y	dir addr
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------

Binary Mode

Source Mode

Bytes: 4

3

Cycles: 1

1

Hex: [A5]Encoding

Encoding

ANL C,/bit

Operation: (PC) $\leftarrow (PC) + 3$ (A) $\leftarrow (C) \text{ and } /(bit)$

Flags:

CY	AC	OV	N	Z
✓	-	-	-	-

[Encoding] A9F(y)H

1	0	1	0	1	0	0	1	1	1	1	1	0	y	y	y	dir addr
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------

Binary Mode

Source Mode

Bytes: 4

3

Cycles: 1

1

Hex: [A5]Encoding

Encoding

CJNE <dest-byte>,<src-byte>,rel

Function: Compare and jump if not equal.

Description: CJNE compares the magnitudes of the first two operands, and branches if their values are not equal.

The branch destination is computed by adding the signed relative displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected. The first two operands allow four addressing mode combinations: the accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant. The C, N and Z flags are affected.

CJNE A,Direct,rel

Operation:

	(PC)	$\leftarrow (PC) + 3$
if	(A)	\neq (direct)
then	(PC)	$\leftarrow (PC) + \text{relative offset}$
if	(A)	$<$ (direct)
then	(C)	$\leftarrow 1$
else	(A)	$\leftarrow 0$

Flags:	CY	AC	OV	N	Z
	✓	-	-	✓	✓

[Encoding] B5H

1	0	1	1	0	1	0	1	direct address	relative address
---	---	---	---	---	---	---	---	----------------	------------------

Binary Mode**Source Mode****Bytes:** 3 3**Cycles:** 1/3 1/3**Hex:** Encoding Encoding**CJNE A,#DATA,rel**

Operation:

	(PC)	$\leftarrow (PC) + 3$
if	(A)	\neq data
then	(PC)	$\leftarrow (PC) + \text{relative offset}$
if	(A)	$<$ data
then	(C)	$\leftarrow 1$
else	(C)	$\leftarrow 0$

Flags:	CY	AC	OV	N	Z
	✓	-	-	✓	✓

[Encoding] B4H

1	0	1	1	0	1	0	0	immediate data	relative address
---	---	---	---	---	---	---	---	----------------	------------------

Binary Mode**Source Mode****Bytes:** 3 3**Cycles:** 1/3 1/3**Hex:** Encoding Encoding**CJNE Rn,#DATA,rel**

Operation:

	(PC)	$\leftarrow (PC) + 3$
if	(Rn)	\neq data
then	(PC)	$\leftarrow (PC) + \text{relative offset}$
if	(Rn)	$<$ data
then	(C)	$\leftarrow 1$
else	(C)	$\leftarrow 0$

Flags:	CY	AC	OV	N	Z
	✓	-	-	✓	✓

[Encoding] B8H - BFH

1	0	1	1	1	r	r	r	immediate data	relative address
---	---	---	---	---	---	---	---	----------------	------------------

Binary Mode**Source Mode****Bytes:**

3

4

Cycles:

1/3

1/3

Hex:

Encoding

[A5]Encoding

CJNE @Ri,#DATA,rel**Operation:**

if (PC) $\leftarrow (PC) + 3$
 then ((Ri)) \neq data
 if (PC) $\leftarrow (PC) + \text{relative offset}$
 then ((Ri)) $<$ data
 then (C) $\leftarrow 1$
 else (C) $\leftarrow 0$

Flags:

CY	AC	OV	N	Z
✓	-	-	✓	✓

[Encoding]

B6H, B7H

1	0	1	1	0	1	1	i	immediate data	relative address
---	---	---	---	---	---	---	---	----------------	------------------

Binary Mode**Source Mode****Bytes:**

3

4

Cycles:

1/3

1/3

Hex:

Encoding

[A5]Encoding

CLR A**Function:** Clear accumulator**Description:** The accumulator is cleared (all bits set to zero). The N and Z flags are affected.**Operation:**

(PC) $\leftarrow (PC) + 1$
 (A) $\leftarrow 0$

Flags:

CY	AC	OV	N	Z
-	-	-	✓	✓

[Encoding]

E4H

1	1	1	0	0	1	0	0
---	---	---	---	---	---	---	---

Binary Mode**Source Mode****Bytes:**

1

1

Cycles:

1

1

Hex:

Encoding

Encoding

CLR bit51**Function:** Clear bit**Description:** The indicated bit is cleared (reset to zero). No other flags are affected.**Operation:**

(PC) $\leftarrow (PC) + 2$
 bit $\leftarrow 0$

Flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Encoding]

C2H

1	1	0	0	0	0	1	0	bit address
---	---	---	---	---	---	---	---	-------------

Binary Mode**Source Mode****Bytes:**

2

2

Cycles: 1 1
Hex: Encoding Encoding

CLR C

Function: Clear carry

Description: The carry flag is cleared (reset to zero). No other flags are affected.

Operation: (PC) $\leftarrow (PC) + 1$
 (C) $\leftarrow 0$

Flags:	CY	AC	OV	N	Z
	✓	-	-	-	-

[Encoding] C3H

1	1	0	0	0	0	1	1
---	---	---	---	---	---	---	---

Binary Mode Source Mode

Bytes: 1 1

Cycles: 1 1

Hex: Encoding Encoding

CLR bit

Function: Clear bit

Description: The indicated bit is cleared (reset to zero). No other flags are affected.

Operation: (PC) $\leftarrow (PC) + 3$
 (bit) $\leftarrow 0$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] A9C(y)H

1	1	0	0	1	0	0	1	1	1	0	0	0	y	y	y	dir addr
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------

Binary Mode Source Mode

Bytes: 4 3

Cycles: 1 1

Hex: [A5]Encoding Encoding

CMP <dest>,<src>

Function: Compare

Description: Subtracts the source operand from the destination operand. The result is not stored in the destination operand. If borrow is needed for bit 7, the CY (borrow) flag is set; otherwise, it is clear. When subtracting signed integers, the OV flag indicates a negative result when a negative value is subtracted from a positive value or a positive result when a positive value is subtracted from a negative value. Bit 7 in this description refers to the most significant byte of the operand (8, 16, or 32 bit). The AC is affected only for byte operand. The N and Z flags are affected according to the result. The source operand allows four addressing modes: register, direct, immediate and indirect.

CMP Rmd,Rms

Operation: (PC) $\leftarrow (PC) + 2$
 (Rmd) $\leftarrow -(Rms)$

Flags:	CY	AC	OV	N	Z
	✓	✓	✓	✓	✓

[Encoding] BCH

1	0	1	1	1	1	0	0	s	s	s	s	S	S	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 3

2

Cycles: 1

1

Hex: [A5]Encoding

Encoding

CMP WRjd,WRjs

Operation: (PC) $\leftarrow (PC) + 2$ (WRjd) $-(WRjs)$

Flags:

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[Encoding] BDH

1	0	1	1	1	1	0	1	t	t	t	t	T	T	T	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 3

2

Cycles: 1

1

Hex: [A5]Encoding

Encoding

CMP DRkd,DRks

Operation: (PC) $\leftarrow (PC) + 2$ (DRkd) $-(DRks)$

Flags:

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[Encoding] BFH

1	0	1	1	1	1	1	1	u	u	u	u	U	U	U	U
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 3

2

Cycles: 1

1

Hex: [A5]Encoding

Encoding

CMP Rm,#DATA

Operation: (PC) $\leftarrow (PC) + 3$ (Rm) $\#data$

Flags:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Encoding] BE(s)0H

1	0	1	1	1	1	1	0	s	s	s	s	0	0	0	0	#data
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-------

Binary Mode

Source Mode

Bytes: 4

3

Cycles: 1

1

Hex: [A5]Encoding

Encoding

CMP WRj,#DATA16

Operation: (PC) $\leftarrow (PC) + 4$ (WRj) $\#data16$

Flags:

CY	AC	OV	N	Z
✓	–	✓	✓	✓

[Encoding] BE(t)4H

1	0	1	1	1	1	1	0	t	t	t	t	0	1	0	0	#data hi	#data lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode**Source Mode****Bytes:**

5

4

Cycles:

1

1

Hex:

[A5]Encoding

Encoding

CMP DRk,#0DATA16**Operation:** (PC) $\leftarrow (PC) + 4$

(DRk) -#0data16

Flags:

CY	AC	OV	N	Z
✓	–	✓	✓	✓

[Encoding] BE(u)8H

1	0	1	1	1	1	1	0	u	u	u	u	1	0	0	0	#data hi	#data lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode**Source Mode****Bytes:**

5

4

Cycles:

1

1

Hex:

[A5]Encoding

Encoding

CMP DRk,#1DATA16**Operation:** (PC) $\leftarrow (PC) + 4$

(DRk) -#1data16

Flags:

CY	AC	OV	N	Z
✓	–	✓	✓	✓

[Encoding] BE(u)CH

1	0	1	1	1	1	1	0	u	u	u	u	1	1	0	0	#data hi	#data lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode**Source Mode****Bytes:**

5

4

Cycles:

1

1

Hex:

[A5]Encoding

Encoding

CMP Rm,Dir8**Operation:** (PC) $\leftarrow (PC) + 3$

(Rm) -(dir8)

Flags:

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Encoding] BE(s)1H

1	0	1	1	1	1	1	0	s	s	s	s	0	0	0	1	dir8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

Binary Mode**Source Mode****Bytes:**

4

3

Cycles:

1

1

Hex:

[A5]Encoding

Encoding

CMP WRj,Dir8

Operation: (PC) $\leftarrow (PC) + 3$
(WRj) $\leftarrow (\text{dir8})$

Flags:	CY	AC	OV	N	Z
	✓	-	✓	✓	✓

[Encoding] BE(t)5H

1	0	1	1	1	1	0	t	t	t	t	0	1	0	1	dir8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

Binary Mode

Source Mode

Bytes: 4

3

Cycles: 1 (2 for SFR) 1 (2 for SFR)

Hex: [A5]Encoding Encoding

CMP Rm,Dir16

Operation: (PC) $\leftarrow (PC) + 4$
(Rm) $\leftarrow (\text{dir16})$

Flags:	CY	AC	OV	N	Z
	✓	✓	✓	✓	✓

[Encoding] BE(s)3H

1	0	1	1	1	1	0	s	s	s	s	0	0	1	1	dir16 hi	dir16 lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode

Source Mode

Bytes: 5

4

Cycles: 1

1

Hex: [A5]Encoding Encoding

CMP WRj,Dir16

Operation: (PC) $\leftarrow (PC) + 4$
(WRj) $\leftarrow (\text{dir16})$

Flags:	CY	AC	OV	N	Z
	✓	-	✓	✓	✓

[Encoding] BE(t)7H

1	0	1	1	1	1	0	t	t	t	t	0	1	1	1	dir16 hi	dir16 lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode

Source Mode

Bytes: 4

3

Cycles: 1 (2 for SFR) 1 (2 for SFR)

Hex: [A5]Encoding Encoding

CMP Rm,@WRj

Operation: (PC) $\leftarrow (PC) + 3$
(Rm) $\leftarrow ((\text{WRj}))$

Flags:	CY	AC	OV	N	Z
	✓	✓	✓	✓	✓

[Encoding] BE(t)9(s)0H

1	0	1	1	1	1	0	t	t	t	t	1	0	0	1	s	s	s	s	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

BE (WRj) 9 (Rm) 0

Binary Mode

Source Mode

Bytes: 4

3

Cycles: 1

1

Hex: [A5]Encoding Encoding

CMP Rm,@DRk

Operation: (PC) \leftarrow (PC) + 3
 (Rm) \leftarrow ((DRk))

Flags:	CY	AC	OV	N	Z
	✓	✓	✓	✓	✓

[Encoding] BE(u)B(s)0H

1	0	1	1	1	1	1	0	u	u	u	u	1	0	1	1	s	s	s	s	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

BE (DRk) B (Rm) 0

Binary Mode	Source Mode
-------------	-------------

Bytes: 4 3

Cycles: 3 3

Hex: [A5]Encoding Encoding

CPL A**Function:** Complement accumulator

Description: Each bit of the accumulator is logically complemented (one's complement). Bits that previously contained a one are changed to zero and vice versa. Only the N and Z flags are affected.

Operation: (PC) \leftarrow (PC) + 1
 (A) \leftarrow /(A)

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] F4H

1	1	1	1	0	1	0	0
---	---	---	---	---	---	---	---

Binary Mode	Source Mode
-------------	-------------

Bytes: 1 1

Cycles: 1 1

Hex: Encoding Encoding

CPL Bit51**Function:** Complement bit

Description: The bit variable specified is complemented. A bit that had been a one is changed to zero and vice versa. No other flags are affected.

Note: When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, not the input pin.

Operation: (PC) \leftarrow (PC) + 2
 (bit51) \leftarrow /(bit51)

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] B2H

1	0	1	1	0	0	1	0	bit address
---	---	---	---	---	---	---	---	-------------

Binary Mode	Source Mode
-------------	-------------

Bytes: 2 2

Cycles: 1 1

Hex: Encoding Encoding

CPL C**Function:** Complement carry**Description:** The carry flag is complemented. A bit, which had been a one, is changed to zero and vice versa.

Operation: (PC) $\leftarrow (\text{PC}) + 1$
 (C) $\leftarrow \neg(\text{C})$

Flags:	CY	AC	OV	N	Z
	✓	-	-	-	-

[Encoding] B3H

1	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 1 1**Cycles:** 1 1**Hex:** Encoding Encoding**CPL Bit****Function:** Complement bit**Description:** The indicated bit is complemented.

Operation: (PC) $\leftarrow (\text{PC}) + 3$
 (bit) $\leftarrow \neg(\text{bit})$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] A9B(y)H

1	0	1	0	1	0	0	1	1	0	1	1	0	y	y	y	dir	addr
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----	------

Binary Mode **Source Mode**

Bytes: 4 3**Cycles:** 1 1**Hex:** [A5]Encoding Encoding**DA A****Function:** Decimal adjust accumulator for addition

Description: DA A adjusts the eight-bit value in the accumulator resulting from the earlier addition of two variables (each in packed BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition. If accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the accumulator producing the proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise. If the carry flag is now set, or if the four high-order bits now exceed nine (1010xxxx-1111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there were a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal additions. OV is not affected. All of this occurs during the one instruction cycle. Essentially; this instruction performs the decimal conversion by adding 00 H , 06 H , 60 H , or 66 H to the accumulator, depending on initial accumulator and PSW conditions. The C, N and Z flags are affected.

Note: DA A cannot simply convert a hexadecimal number in the accumulator to BCD notation, nor

does DA A apply to decimal subtraction.

Operation: $(PC) \leftarrow (PC) + 3$
 if $[[(A3-0) > 9] \wedge [(AC) = 1]]$
 then $(A3-0) \leftarrow (A3-0) + 6$
 next if $[[(A7-4) > 9] \wedge [(C) = 1]]$
 then $(A7-4) \leftarrow (A7-4) + 6$

Flags:	CY	AC	OV	N	Z
	✓	-	-	✓	✓

[Encoding] B4H

1	0	1	1	0	1	0	0
---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 1 1

Cycles: 3 3

Hex: Encoding Encoding

DEC byte

Function: Decrement

Description: The variable indicated is decrement by 1. Original values of 00H will underflow to 0FFH. Only the N and Z flags are affected. The following addressing modes are allowed: accumulator, register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

DEC A

Operation: $(PC) \leftarrow (PC) + 1$
 $(A) \leftarrow (A) - 1$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 14H

0	0	0	1	0	1	0	0
---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 1 1

Cycles: 1 1

Hex: Encoding Encoding

DEC Rn

Operation: $(PC) \leftarrow (PC) + 1$
 $(Rn) \leftarrow (Rn) - 1$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 18H - 1FH

0	0	0	1	1	r	r	r
---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 1 2

Cycles: 1 1

Hex: Encoding [A5]Encoding

DEC Direct**Operation:** (PC) $\leftarrow (PC) + 2$ (direct) $\leftarrow (\text{direct}) - 1$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 15H

0	0	0	1	0	1	0	1	direct address
---	---	---	---	---	---	---	---	----------------

Binary Mode**Source Mode****Bytes:** 2

2

Cycles: 1

1

Hex: Encoding

Encoding

DEC @Ri**Operation:** (PC) $\leftarrow (PC) + 1$ ((Ri)) $\leftarrow ((Ri)) - 1$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 16H, 17H

0	0	0	1	0	1	1	i
---	---	---	---	---	---	---	---

Binary Mode**Source Mode****Bytes:** 1

2

Cycles: 1

1

Hex: Encoding

[A5]Encoding

DEC <dest>,<src>**Function:** Decrement destination value**Description:** Decrements the specified variable at the destination operand by 1, 2, or 4. An original value of 00H underflows to 0FFH. The N and Z flags are affected. The value of decrement is coded as follows:

vv	value
00	1
01	2
10	3

DEC Rm,#short**Operation:** (PC) $\leftarrow (PC) + 2$ (Rm) $\leftarrow (Rm) - \text{\#short}$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 1B(s)(00v)H

0	0	0	1	1	0	1	1	s	s	s	s	0	0	v	v
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode**Source Mode****Bytes:** 3

2

Cycles: 1

1

Hex: [A5]Encoding

Encoding

DEC WRj,#short

Operation: (PC) $\leftarrow (PC) + 2$
 (WRj) $\leftarrow (WRj) - \#short$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 1B(t)(01v)H

0	0	0	1	1	0	1	1	t	t	t	t	0	1	v	v
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 3 2

Cycles: 1 1

Hex: [A5]Encoding Encoding

DEC DRk,#short

Operation: (PC) $\leftarrow (PC) + 2$
 (DRk) $\leftarrow (DRk) - \#short$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 1B(u)(11v)H

0	0	0	1	1	0	1	1	u	u	u	u	1	1	v	v
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 3 2

Cycles: 1 1

Hex: [A5]Encoding Encoding

DIV AB

Description: DIV AB divides the unsigned eight-bit integer in the accumulator by the unsigned eight-bit integer in register B. The accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared. The N and Z flags also are affected according to the result.

Exception: If B had originally contained 00 H, the values returned in the accumulator and B register will be undefined. Additionally, the overflow flag will be set. The carry flag is cleared in any case.

Other flags are undefined.

Operation: (PC) $\leftarrow (PC) + 1$
 (A15-8) $\leftarrow (A) / (B) - \text{result's bits } 15..8$
 (A7-0) $\leftarrow (A) / (B) - \text{result's bits } 7..0$

Flags:	CY	AC	OV	N	Z
	0	-	✓*	✓*	✓*

*) – if B originally contained 00H, then OV=1, N and Z are undefined

[Encoding] 84H

1	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 1 1

Cycles: 6 6

Hex: Encoding Encoding

DIV <dest>,<src>

Description: Divides the unsigned integer in the register by the unsigned integer operand in register addressing mode and clears the CY and OV flags. For byte operands (<dest>, <src> = Rmd,Rms) the result is 16 bits. The 8-bit quotient is stored in the higher byte of the word where Rmd resides; the 8-bit remainder is stored in the lower byte of the word where Rmd resides. For example: Register 1 contains 251 (0FBH) and register 5 contains 18 (12H). After executing the instruction DIV R1, R5 register 1 contains 13 (0DH or 00001101B); register 0 contains 17 (11H or 00010001B), since $251 = (13 \times 18) + 17$; and the CY and OV bits are clear. The CY flag is cleared. The N flag is set if the MSB of the quotient is set. The Z flag is set if the quotient is zero.

Exception: If <src> contains 00H, the values returned in both operands are undefined; the CY flag is cleared, OV flag is set, and the rest of the flags are undefined.

DIV Rmd,Rms

Operation: (PC) $\leftarrow (PC) + 2$
 (Rmd) remainder (Rmd) / (Rms) if <dest> md = 0,2,4,...,14
 (Rmd+1) quotient (Rmd) / (Rms)
 (Rmd-1) remainder (Rmd) / (Rms) if <dest> md = 1,3,5,...,15
 (Rmd) quotient (Rmd) / (Rms)

Flags:	CY	AC	OV	N	Z
	0	-	√*	√*	√*

*) – if source originally contained 0, then OV=1, N and Z are undefined

[Encoding] 8CH

1	0	0	0	1	1	0	0	s	s	s	s	S	S	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 3

2

Cycles: 6

6

Hex: [A5]Encoding

Encoding

DIV WRjd,WRjs

Operation: (PC) $\leftarrow (PC) + 2$
 (WRjd) remainder (WRjd) / (WRjs) if <dest> jd = 0,4,8,...,28
 (WRjd+1) quotient (WRjd) / (WRjs)
 (WRjd-1) remainder (WRjd) / (WRjs) if <dest> jd = 2,6,10,...,30
 (WRjd) quotient (WRjd) / (WRjs)

For word operands (<dest>,<src> = WRjd,WRjs) the 16-bit quotient is in WR(jd+2), and the 16-bit remainder is in WRjd. For example, for a destination register WR4, assume the quotient is 1122H and the remainder is 3344H. Then, the results are stored in these register file locations: (4)->0x33, (5)->0x44, (6)->0x11, (7)->0x22.

Flags:	CY	AC	OV	N	Z
	0	-	√*	√*	√*

*) – if source originally contained 0, then OV=1, N and Z are undefined

[Encoding] 8DH

1	0	0	0	1	1	0	1	t	t	t	t	T	T	T	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 3

2

Cycles: 10

10

Hex: [A5]Encoding

Encoding

DJNZ <byte>, <rel-addr>

Function: Decrement and jump if not zero

Description: DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. Original values of 00H will underflow to 0FFH. Only the N and Z flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction. The location decremented may be a register or directly addressed byte.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

DJNZ Rn, rel

Operation:

$$(PC) \leftarrow (PC) + 2$$

$$(Rn) \leftarrow (Rn) - 1$$

if $(Rn) \neq 0$

then $(PC) \leftarrow (PC) + rel$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] D8H - DFH

1	1	0	1	1	r	r	r	relative address
---	---	---	---	---	---	---	---	------------------

Binary Mode **Source Mode**

Bytes: 2 3

Cycles: 1/3 1/3

Hex: Encoding [A5]Encoding

DJNZ Direct, rel

Operation:

$$(PC) \leftarrow (PC) + 3$$

$$(direct) \leftarrow (direct) - 1$$

if $(direct) \neq 0$

then $(PC) \leftarrow (PC) + rel$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] D5H

1	1	0	1	0	1	0	1	direct address	relative address
---	---	---	---	---	---	---	---	----------------	------------------

Binary Mode **Source Mode**

Bytes: 3 4

Cycles: 3 3

Hex: Encoding [A5]Encoding

ECALL <dest>

Function: Extended call

Description: Calls a subroutine located at the specified address. The instruction adds four to the program counter to generate the address of the next instruction and then pushes the 24-bit result onto the stack (high byte first), incrementing the stack pointer by three. The 8 bits of the high word and the 16 bits of the low word of the PC are then loaded, respectively, with the second, third and fourth bytes of the ECALL

instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 16-Mbyte memory space. No flags are affected.

ECALL addr24

Operation: (PC) \leftarrow (PC) + 4
 (SP) \leftarrow (SP) + 1
 ((SP)) \leftarrow (PC.23:16)
 (SP) \leftarrow (SP) + 1
 ((SP)) \leftarrow (PC.15:8)
 (SP) \leftarrow (SP) + 1
 ((SP)) \leftarrow (PC.7:0)
 (PC) \leftarrow (addr.23:0)

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 9AH

1	0	0	1	1	0	1	0	addr 23-16	addr 15-8	addr 7-0
---	---	---	---	---	---	---	---	------------	-----------	----------

Binary Mode

Source Mode

Bytes: 5 4

Cycles: 3 3

Hex: [A5]Encoding Encoding

ECALL @DRk

Operation: (PC) \leftarrow (PC) + 4
 (SP) \leftarrow (SP) + 1
 ((SP)) \leftarrow (PC.23:16)
 (SP) \leftarrow (SP) + 1
 ((SP)) \leftarrow (PC.15:8)
 (SP) \leftarrow (SP) + 1
 ((SP)) \leftarrow (PC.7:0)
 (PC) \leftarrow ((DRk))

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 99(u)8H

1	0	0	1	1	0	0	1	u	u	u	u	1	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 3 2

Cycles: 3 3

Hex: [A5]Encoding Encoding

EJMP <dest>

Function: Extended jump

Description: Causes an unconditional branch to the specified address by loading the 8 bits of the high order and 16 bits of the low order words of the PC with the second, third, and fourth instruction bytes. The destination may be therefore be anywhere in the full 16-Mbyte memory space. No flags are affected.

EJMP addr24

Operation: (PC) $\leftarrow (\text{addr.23:0})$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 8AH

1	0	0	0	1	0	1	0	addr 23-16	addr 15-8	addr 7-0
---	---	---	---	---	---	---	---	------------	-----------	----------

Binary Mode **Source Mode**

Bytes: 5 4

Cycles: 3 3

Hex: [A5]Encoding Encoding

EJMP @DRk

Operation: (PC) $\leftarrow ((\text{DRk}))$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 89(u)8H

1	0	0	0	1	0	0	1	u	u	u	u	1	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 3 2

Cycles: 3 3

Hex: [A5]Encoding Encoding

ESC

Function: Switch to opposite mode (BINARY->SOURCE or SOURCE->BINARY)

Description: Execution continues at the following instruction in opposite mode. Other than the PC, no registers or flags are affected.

Operation: (PC) $\leftarrow (\text{PC}) + 1$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] A5H

1	0	1	0	0	1	0	1
---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 1 1

Cycles: 1 1

Hex: Encoding Encoding

ERET

Function: Extended return

Description: Pops byte 2, byte 1, and byte 0 of the 3-byte PC successively from the stack and decrements the stack pointer by 3. Program execution continues at the resulting address, which normally is the instruction immediately following ECALL. No flags are affected.

Operation: (PC.7:0) $\leftarrow ((\text{SP}))$
 (SP) $\leftarrow (\text{SP}) - 1$
 (PC.15:8) $\leftarrow ((\text{SP}))$
 (SP) $\leftarrow (\text{SP}) - 1$
 (PC.23:16) $\leftarrow ((\text{SP}))$
 (SP) $\leftarrow (\text{SP}) - 1$

Flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Encoding] AAH

1	0	1	0	1	0	1	0
---	---	---	---	---	---	---	---

Binary Mode**Source Mode****Bytes:**

2

1

Cycles:

3

3

Hex:

[A5]Encoding

Encoding

INC byte**Function:** Increment

Description: INC increments the indicated variable by 1. An original value of 0FFh will overflow to 00h. Only the N and Z flags are affected. Three addressing modes are allowed: register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

INC A**Operation:** (PC) $\leftarrow (PC) + 1$ (A) $\leftarrow (A) + 1$ **Flags:**

CY	AC	OV	N	Z
-	-	-	-	-

[Encoding] 04H

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

Binary Mode**Source Mode****Bytes:**

1

1

Cycles:

1

1

Hex:

Encoding

Encoding

INC Rn**Operation:** (PC) $\leftarrow (PC) + 1$ (Rn) $\leftarrow (Rn) + 1$ **Flags:**

CY	AC	OV	N	Z
-	-	-	-	-

[Encoding] 08H - 0FH

0	0	0	0	1	r	r	r
---	---	---	---	---	---	---	---

Binary Mode**Source Mode****Bytes:**

1

2

Cycles:

1

1

Hex:

Encoding

[A5]Encoding

INC Direct**Operation:** (PC) $\leftarrow (PC) + 1$ (direct) $\leftarrow (direct) + 1$ **Flags:**

CY	AC	OV	N	Z
-	-	-	-	-

[Encoding] 05H

0	0	0	0	0	1	0	1	direct address
---	---	---	---	---	---	---	---	----------------

Binary Mode**Source Mode****Bytes:** 2

2

Cycles: 1

1

Hex:

Encoding

Encoding

INC @Ri**Operation:** (PC) $\leftarrow (PC) + 1$ (Ri) $\leftarrow (Ri) + 1$ **Flags:**

CY	AC	OV	N	Z
-	-	-	-	-

[Encoding] 06H, 07H

0	0	0	0	0	0	1	1	i
---	---	---	---	---	---	---	---	---

Binary Mode**Source Mode****Bytes:** 1

2

Cycles: 1

1

Hex:

Encoding

[A5]Encoding

INC <dest>,<src>**Function:** Increment**Description:** Increments the specified variable by 1, 2, or 4. An original value of 0FFH overflows to 00H. Only the N and Z flags are affected. The value of increment is coded as follows:

vv	value
00	1
01	2
10	3

INC Rm,#short**Operation:** (PC) $\leftarrow (PC) + 2$ (Rm) $\leftarrow (Rm) + \#short$ **Flags:**

CY	AC	OV	N	Z
-	-	-	✓	✓

[Encoding] 0B(s)(00v)H

0	0	0	0	0	1	0	1	1	s	s	s	s	0	0	v	v
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode**Source Mode****Bytes:** 3

2

Cycles: 1

1

Hex:

[A5]Encoding

Encoding

INC WRj,#short**Operation:** (PC) $\leftarrow (PC) + 2$ (WRj) $\leftarrow (WRj) + \#short$ **Flags:**

CY	AC	OV	N	Z
-	-	-	✓	✓

[Encoding] 0B(t)(01v)H

0	0	0	0	0	1	0	1	1	t	t	t	t	0	1	v	v
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

	Binary Mode	Source Mode
Bytes:	3	2
Cycles:	1	1
Hex:	[A5]Encoding	Encoding

INC DRk,#short

Operation: (PC) \leftarrow (PC) + 2
(DRk) \leftarrow (DRk) + #short

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 0B(u)(11v)H

0	0	0	0	1	0	1	1	u	u	u	u	1	1	v	v
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

	Binary Mode	Source Mode
Bytes:	3	2
Cycles:	1	1
Hex:	[A5]Encoding	Encoding

INC DPTR

Function: Increment data pointer

Description: Increment the 16-bit data pointer by one. A 16-bit increment (modulo 2^{16}) is performed; an overflow of the low-order byte of the data pointer (DPL) from 0FFH to 00H will increment the high-order byte (DPH). Only the N and Z flags are affected. This is the only 16-bit register that can be incremented.

Operation: (PC) \leftarrow (PC) + 1
(DPTR) \leftarrow (DPTR) + 1

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] A3H

1	0	1	0	0	0	1	1
---	---	---	---	---	---	---	---

	Binary Mode	Source Mode
Bytes:	1	1
Cycles:	1	1
Hex:	Encoding	Encoding

JB

Function: Jump if bit is set

Description: If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. The bit tested is not modified. No flags are affected.

JB Bit51,rel

Operation: (PC) \leftarrow (PC) + 3
if (bit51) = 1
then (PC) \leftarrow (PC) + rel

Flags:	CY	AC	OV	N	Z
---------------	----	----	----	---	---

	-	-	-	-	-
[Encoding] 20H					
	0	0	1	0	0 0 0 0 bit address relative address
	Binary Mode		Source Mode		
Bytes:	3		3		
Cycles:	1/3		1/3		
Hex:	Encoding		Encoding		

JB Bit,rel

Operation:		$(PC) \leftarrow (PC) + 3$			
		if $(bit) = 1$			
		then $(PC) \leftarrow (PC) + rel$			
Flags:		CY	AC	OV	N Z
		-	-	-	-
[Encoding] A92(y)H					
	1	0	1	0	1 0 0 1 0 0 1 0 0 y y y direct addr rel.addr
		Binary Mode		Source Mode	
Bytes:	5	4			
Cycles:	1/3	1/3			
Hex:	[A5]Encoding	Encoding			

JBC

Function:	Jump if bit is set and clear bit
Description:	If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. In either case, clear the designated bit. The branch destination is computed by adding the signed relative displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.
	Note: When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, not the input pin.

JBC Bit51,rel

Operation:		$(PC) \leftarrow (PC) + 3$			
		if $(bit51) = 1$			
		then $(PC) \leftarrow (PC) + rel$			
Flags:		CY	AC	OV	N Z
		-	-	-	-
[Encoding] 10H					
	0	0	0	1	0 0 0 0 bit address relative address
		Binary Mode		Source Mode	
Bytes:	3	3			
Cycles:	1/3	1/3			
Hex:	Encoding	Encoding			

JBC Bit,rel

Operation:		$(PC) \leftarrow (PC) + 3$			
		if $(bit) = 1$			

then $(PC) \leftarrow (PC) + rel$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] A91(y)H

1	0	1	0	1	0	0	1	0	0	0	1	0	y	y	y	direct addr	rel.addr
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-------------	----------

Binary Mode

Source Mode

Bytes: 5

4

Cycles: 1/3

1/3

Hex: [A5]Encoding

Encoding

JC

Function: Jump if carry is set

Description: If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative- displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.

Operation: $(PC) \leftarrow (PC) + 2$

if $(C) = 1$

then $(PC) \leftarrow (PC) + rel$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 40H

0	1	0	0	0	0	0	0	relative address
---	---	---	---	---	---	---	---	------------------

Binary Mode

Source Mode

Bytes: 2

2

Cycles: 1/3

1/3

Hex: Encoding

Encoding

JE

Function: Jump if equal

Description: If the Z flag is set, branch to the address specified; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative displacement in the second instruction byte to the PC, after incrementing the PC twice.

Operation: $(PC) \leftarrow (PC) + 2$

if $(Z) = 1$

then $(PC) \leftarrow (PC) + rel$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 68H

0	1	1	0	1	0	0	0	relative address
---	---	---	---	---	---	---	---	------------------

Binary Mode

Source Mode

Bytes: 3

2

Cycles: 1/3

1/3

Hex: [A5]Encoding

Encoding

JG

Function: Jump if greater than

Description: If the Z flag and the CY flag are both clear, branch to the address specified; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative displacement in the second instruction byte to the PC, after incrementing the PC twice.

Operation: $(PC) \leftarrow (PC) + 2$
 if $(Z=0 \text{ and } C=0) = 1$
 then $(PC) \leftarrow (PC) + \text{rel}$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 38H

0	0	1	1	1	0	0	0	relative address
---	---	---	---	---	---	---	---	------------------

Binary Mode **Source Mode**

Bytes: 3 2

Cycles: 1/3 1/3

Hex: [A5]Encoding Encoding

JLE

Function: Jump if less than or equal

Description: If the Z flag or the CY flag is set, branch to the address specified; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative displacement in the second instruction byte to the PC, after incrementing the PC twice.

Operation: $(PC) \leftarrow (PC) + 2$
 if $(Z=1 \text{ and } C=1) = 1$
 then $(PC) \leftarrow (PC) + \text{rel}$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 28H

0	0	1	0	1	0	0	0	relative address
---	---	---	---	---	---	---	---	------------------

Binary Mode **Source Mode**

Bytes: 3 2

Cycles: 1/3 1/3

Hex: [A5]Encoding Encoding

JMP

Function: Jump indirect

Description: Add the eight-bit unsigned contents of the accumulator with the sixteen-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 2¹⁶): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the accumulator nor the data pointer is altered. No flags are affected.

Operation: $(PC) \leftarrow (A) + (DPTR)$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 73H

0	1	1	1	0	0	1	1
---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 1 1

Cycles: 1/3 1/3
Hex: Encoding Encoding

JNB

Function: Jump if bit is not set

Description: If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. The bit tested is not modified. No flags are affected.

JNB Bit51,rel

Operation: $(PC) \leftarrow (PC) + 3$
 if $(bit51) = 0$
 then $(PC) \leftarrow (PC) + rel$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 30H

0	0	1	1	0	0	0	0	bit address	relative address
---	---	---	---	---	---	---	---	-------------	------------------

Binary Mode

Source Mode

Bytes: 3 3
Cycles: 1/3 1/3
Hex: Encoding Encoding

JNB Bit,rel

Operation: $(PC) \leftarrow (PC) + 3$
 if $(bit) = 0$
 then $(PC) \leftarrow (PC) + rel$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] A93(y)H

1	0	1	0	1	0	0	1	0	0	1	1	0	y	y	y	direct addr	rel.addr
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-------------	----------

Binary Mode

Source Mode

Bytes: 5 4
Cycles: 1/3 1/3
Hex: [A5]Encoding Encoding

JNC

Function: Jump if carry is not set

Description: If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.

Operation: $(PC) \leftarrow (PC) + 2$
 if $(C) = 0$
 then $(PC) \leftarrow (PC) + rel$

Flags:	CY	AC	OV	N	Z
--------	----	----	----	---	---

	-	-	-	-	-
[Encoding]	50H				
	0	1	0	1	0 0 0 0 relative address
	Binary Mode		Source Mode		
Bytes:	2		2		
Cycles:	1/3		1/3		
Hex:	Encoding		Encoding		

JNE

Function: Jump if not equal

Description: If the Z flag is clear, branch to the address specified; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative displacement in the second instruction byte to the PC, after incrementing the PC twice.

Operation: $(PC) \leftarrow (PC) + 2$
 if $(Z) = 0$
 then $(PC) \leftarrow (PC) + rel$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding]	78H				
	0	1	1	1	1 0 0 0 relative address
	Binary Mode		Source Mode		
Bytes:	3		2		
Cycles:	1/3		1/3		
Hex:	[A5]Encoding		Encoding		

JNZ

Function: Jump if accumulator is not zero

Description: If any bit of the accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected.

Operation: $(PC) \leftarrow (PC) + 2$
 if $(A) \neq 0$
 then $(PC) \leftarrow (PC) + rel$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding]	70H				
	0	1	1	1	0 0 0 0 relative address
	Binary Mode		Source Mode		
Bytes:	2		2		
Cycles:	1/3		1/3		
Hex:	Encoding		Encoding		

JSG

Function: Jump if greater than(signed)

Description: If the Z flag is clear AND the N flag and the OV flag have the same value, branch to the address

specified; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative displacement in the second instruction byte to the PC, after incrementing the PC twice.

Operation: $(PC) \leftarrow (PC) + 2$
 if $(Z=0 \text{ and } N=OV)$
 then $(PC) \leftarrow (PC) + \text{rel}$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 18H

0	0	0	1	1	0	0	0	relative address
---	---	---	---	---	---	---	---	------------------

Binary Mode **Source Mode**

Bytes: 3 2

Cycles: 1/3 1/3

Hex: [A5]Encoding Encoding

JSGE

Function: Jump if greater than or equal(signed)

Description: If the N flag and the OV flag have the same value, branch to the address specified; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative displacement in the second instruction byte to the PC, after incrementing the PC twice.

Operation: $(PC) \leftarrow (PC) + 2$
 if $(N=OV)$
 then $(PC) \leftarrow (PC) + \text{rel}$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 58H

0	1	0	1	1	0	0	0	relative address
---	---	---	---	---	---	---	---	------------------

Binary Mode **Source Mode**

Bytes: 3 2

Cycles: 1/3 1/3

Hex: [A5]Encoding Encoding

JSL

Function: Jump if less than(signed)

Description: If the N flag and the OV flag have the same value, branch to the address specified; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative displacement in the second instruction byte to the PC, after incrementing the PC twice.

Operation: $(PC) \leftarrow (PC) + 2$
 if $(N \neq OV)$
 then $(PC) \leftarrow (PC) + \text{rel}$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 48H

0	1	0	0	1	0	0	0	relative address
---	---	---	---	---	---	---	---	------------------

Binary Mode **Source Mode**

Bytes: 3 2

Cycles: 1/3 1/3
Hex: [A5]Encoding Encoding

JSLE

Function: Jump if less than or equal(signed)

Description: If the Z flag is set OR if the the N flag and the OV flag have different values, branch to the address specified; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative displacement in the second instruction byte to the PC, after incrementing the PC twice.

Operation: $(PC) \leftarrow (PC) + 2$
 if $(Z=1 \text{ or } (N \neq OV))$
 then $(PC) \leftarrow (PC) + \text{rel}$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 08H

0	0	0	0	1	0	0	0	relative address
---	---	---	---	---	---	---	---	------------------

Binary Mode Source Mode

Bytes: 3 2
Cycles: 1/3 1/3
Hex: [A5]Encoding Encoding

JZ

Function: Jump if accumulator is zero

Description: If all bits of the accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected.

Operation: $(PC) \leftarrow (PC) + 2$
 if $(A) = 0$
 then $(PC) \leftarrow (PC) + \text{rel}$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 60H

0	1	1	0	0	0	0	0	relative address
---	---	---	---	---	---	---	---	------------------

Binary Mode Source Mode

Bytes: 2 2
Cycles: 1/3 1/3
Hex: Encoding Encoding

LCALL

Function: Long call

Description: Calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the stack pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin

anywhere in the full 64 Kbyte program memory address space. No flags are affected.

LCALL addr16

Operation: (PC) \leftarrow (PC) + 3
 (SP) \leftarrow (SP) + 1
 ((SP)) \leftarrow (PC.7:0)
 (SP) \leftarrow (SP) + 1
 ((SP)) \leftarrow (PC.15:8)
 (PC) \leftarrow (addr.15:0)

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 12H

0	0	0	1	0	0	1	0	addr 15-8	addr 7-0
---	---	---	---	---	---	---	---	-----------	----------

Binary Mode

Source Mode

Bytes: 3

3

Cycles: 3

3

Hex: Encoding

Encoding

LCALL @WRj

Operation: (PC) \leftarrow (PC) + 3
 (SP) \leftarrow (SP) + 1
 ((SP)) \leftarrow (PC.7:0)
 (SP) \leftarrow (SP) + 1
 ((SP)) \leftarrow (PC.15:8)
 (PC) \leftarrow ((WRj))

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 99(t)4H

1	0	0	1	1	0	0	1	t	t	t	t	0	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 3

2

Cycles: 3

3

Hex: [A5]Encoding

Encoding

LJMP

Function: Long jump

Description: LJMP causes an unconditional branch to the indicated address, by loading the high- order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.

LJMP addr16

Operation: (PC) \leftarrow (addr.15:0)

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 02H

0	0	0	0	0	0	1	0	addr 15-8	addr 7-0
---	---	---	---	---	---	---	---	-----------	----------

	Binary Mode	Source Mode
Bytes:	3	3
Cycles:	3	3
Hex:	Encoding	Encoding

LJMP @WRj

Operation: (PC) $\leftarrow ((WRj))$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 89(t)4H

1	0	0	0	1	0	0	1	t	t	t	t	0	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

	Binary Mode	Source Mode
Bytes:	3	2
Cycles:	3	3
Hex:	[A5]Encoding	Encoding

MOV

Function: Move variable

Description: The variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected. This is by far the most flexible operation. Twenty-four combinations of source and destination addressing modes are allowed.

MOV A,Rn

Operation: (PC) $\leftarrow (PC) + 1$
(A) $\leftarrow (Rn)$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] E8H - EFH

1	1	1	0	1	r	r	r
---	---	---	---	---	---	---	---

	Binary Mode	Source Mode
Bytes:	1	2
Cycles:	1	1
Hex:	Encoding	[A5]Encoding

MOV A,Direct

Operation: (PC) $\leftarrow (PC) + 2$
(A) $\leftarrow (\text{direct})$

Note: MOV A, ACC is a valid instruction.

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] E5H

1	1	1	0	0	1	0	1	direct address
---	---	---	---	---	---	---	---	----------------

	Binary Mode	Source Mode
Bytes:	2	2
Cycles:	1	1

Hex: Encoding Encoding

MOV A,@Ri**Operation:** (PC) $\leftarrow (PC) + 1$ (A) $\leftarrow ((Ri))$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] E6H, E7H

1	1	1	0	0	1	1	i
---	---	---	---	---	---	---	---

Binary Mode **Source Mode****Bytes:** 1 2**Cycles:** 1 1**Hex:** Encoding [A5]Encoding**MOV A,#DATA****Operation:** (PC) $\leftarrow (PC) + 2$ (A) $\leftarrow \#data$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 74H

0	1	1	1	0	1	0	0	immediate data
---	---	---	---	---	---	---	---	----------------

Binary Mode **Source Mode****Bytes:** 2 2**Cycles:** 1 1**Hex:** Encoding Encoding**MOV Rn,A****Operation:** (PC) $\leftarrow (PC) + 1$ (Rn) $\leftarrow (A)$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] F8H - FFH

1	1	1	1	1	r	r	r
---	---	---	---	---	---	---	---

Binary Mode **Source Mode****Bytes:** 1 2**Cycles:** 1 1**Hex:** Encoding [A5]Encoding**MOV Rn,Direct****Operation:** (PC) $\leftarrow (PC) + 2$ (Rn) $\leftarrow (direct)$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] A8H - AFH

1	0	1	0	1	r	r	r	direct address
---	---	---	---	---	---	---	---	----------------

Binary Mode **Source Mode**

Bytes: 2 3
Cycles: 1 1
Hex: Encoding [A5]Encoding

MOV Rn,#DATA

Operation: (PC) $\leftarrow (PC) + 2$

(Rn) $\leftarrow \#data$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 78H - 7FH

0	1	1	1	1	r	r	r	direct address
---	---	---	---	---	---	---	---	----------------

Binary Mode **Source Mode**

Bytes: 2 3

Cycles: 1 1

Hex: Encoding [A5]Encoding

MOV Direct,A

Operation: (PC) $\leftarrow (PC) + 2$

(direct) $\leftarrow (A)$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] F5H

1	1	1	1	0	1	0	1	direct address
---	---	---	---	---	---	---	---	----------------

Binary Mode **Source Mode**

Bytes: 2 2

Cycles: 1 1

Hex: Encoding [A5]Encoding

MOV Direct,Rn

Operation: (PC) $\leftarrow (PC) + 2$

(direct) $\leftarrow (Rn)$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 88H - 8FH

1	0	0	0	1	r	r	r	direct address
---	---	---	---	---	---	---	---	----------------

Binary Mode **Source Mode**

Bytes: 2 3

Cycles: 1 1

Hex: Encoding [A5]Encoding

MOV Direct,Direct

Operation: (PC) $\leftarrow (PC) + 3$

(direct) $\leftarrow (direct)$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 85H

1	0	0	0	0	1	0	1	direct address(source)	direct address(destination)
---	---	---	---	---	---	---	---	------------------------	-----------------------------

Binary Mode **Source Mode**

Bytes: 3 3

Cycles: 1 1

Hex: Encoding Encoding

MOV Direct, @Ri

Operation: (PC) $\leftarrow (PC) + 2$
 (direct) $\leftarrow ((Ri))$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 86H, 87H

1	0	0	0	0	1	1	i	direct address
---	---	---	---	---	---	---	---	----------------

Binary Mode **Source Mode**

Bytes: 2 3

Cycles: 1 1

Hex: Encoding [A5]Encoding

MOV Direct, #DATA

Operation: (PC) $\leftarrow (PC) + 2$
 (direct) $\leftarrow \#data$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 75H

0	1	1	1	0	1	0	1	direct address(source)	immediate data
---	---	---	---	---	---	---	---	------------------------	----------------

Binary Mode **Source Mode**

Bytes: 3 3

Cycles: 1 1

Hex: Encoding Encoding

MOV @Ri, A

Operation: (PC) $\leftarrow (PC) + 1$
 ((Ri)) $\leftarrow (A)$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] F6H, F7H

1	1	1	1	0	1	1	i
---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 1 2

Cycles: 1 1

Hex: Encoding [A5]Encoding

MOV @Ri, Direct

Operation: (PC) $\leftarrow (PC) + 2$
 ((Ri)) $\leftarrow (direct)$

Flags:	CY	AC	OV	N	Z
---------------	----	----	----	---	---

	-	-	-	-	-
[Encoding]	A6H, A7H				
	1	0	1	0	0 1 1 i direct address
	Binary Mode		Source Mode		
Bytes:	2		3		
Cycles:	1		1		
Hex:	Encoding		[A5]Encoding		

MOV @Ri,#DATA

Operation:	(PC)	$\leftarrow (PC) + 2$			
	((Ri))	$\leftarrow \#data$			
Flags:	CY	AC	OV	N	Z
	-	-	-	-	-
[Encoding]	76H, 77H				
	0	1	1	1	0 1 1 i immediate data
	Binary Mode		Source Mode		
Bytes:	2		3		
Cycles:	1		1		
Hex:	Encoding		[A5]Encoding		

MOV Rmd,Rms

Operation:	(PC)	$\leftarrow (PC) + 2$			
	(Rmd)	$\leftarrow (Rms)$			
Flags:	CY	AC	OV	N	Z
	-	-	-	-	-
[Encoding]	7CH				
	0	1	1	1	1 1 0 0 s s s s S S S S
	Binary Mode		Source Mode		
Bytes:	3		2		
Cycles:	1		1		
Hex:	[A5]Encoding		Encoding		

MOV WRjd,WRjs

Operation:	(PC)	$\leftarrow (PC) + 2$			
	(WRjd)	$\leftarrow (WRjs)$			
Flags:	CY	AC	OV	N	Z
	-	-	-	-	-
[Encoding]	7DH				
	0	1	1	1	1 1 0 1 t t t t T T T T
	Binary Mode		Source Mode		
Bytes:	3		2		
Cycles:	1		1		
Hex:	[A5]Encoding		Encoding		

MOV DRkd,DRks

Operation:	(PC)	$\leftarrow (PC) + 2$
------------	------	-----------------------

	(DRkd)	←(DRks)			
Flags:	CY	AC	OV	N	Z
	-	-	-	-	-
[Encoding]	7FH				
	0	1	1	1	1
	1	1	1	1	1
	u	u	u	u	u
	U	U	U	U	U
	Binary Mode		Source Mode		
Bytes:	3		2		
Cycles:	1		1		
Hex:	[A5]Encoding		Encoding		

MOV Rm,#DATA

Operation:	(PC)	←(PC) + 3			
	(Rm)	←#data			
Flags:	CY	AC	OV	N	Z
	-	-	-	-	-
[Encoding]	7E(s)0H				
	0	1	1	1	1
	1	1	1	0	0
	s	s	s	s	s
	0	0	0	0	0
	Binary Mode		Source Mode		
Bytes:	4		3		
Cycles:	1		1		
Hex:	[A5]Encoding		Encoding		

MOV WRj,#DATA16

Operation:	(PC)	←(PC) + 4			
	(WRj)	←#data16			
Flags:	CY	AC	OV	N	Z
	-	-	-	-	-
[Encoding]	7E(t)4H				
	0	1	1	1	1
	1	1	1	0	0
	t	t	t	t	t
	0	1	0	0	0
	Binary Mode		Source Mode		
Bytes:	5		4		
Cycles:	1		1		
Hex:	[A5]Encoding		Encoding		

MOV DRk,#0DATA16

Operation:	(PC)	←(PC) + 4			
	(DRk)	←#0data16			
Flags:	CY	AC	OV	N	Z
	-	-	-	-	-
[Encoding]	7E(u)8H				
	0	1	1	1	1
	1	1	1	0	0
	u	u	u	u	u
	1	0	0	0	0
	Binary Mode		Source Mode		
Bytes:	5		4		
Cycles:	1		1		
Hex:	[A5]Encoding		Encoding		

MOV DRk,#1DATA16

Operation: (PC) \leftarrow (PC) + 4
(DRk) \leftarrow #1data16

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 7E(u)CH

0	1	1	1	1	1	1	0	u	u	u	u	1	1	0	0	#data hi	#data lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode **Source Mode**

Bytes: 5 4

Cycles: 1 1

Hex: [A5]Encoding Encoding

MOV Rm,Dir8

Operation: (PC) \leftarrow (PC) + 3
(Rm) \leftarrow (dir8)

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 7E(s)1H

0	1	1	1	1	1	1	0	s	s	s	s	0	0	0	1	dir8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

Binary Mode **Source Mode**

Bytes: 4 3

Cycles: 1 1

Hex: [A5]Encoding Encoding

MOV WRj,Dir8

Operation: (PC) \leftarrow (PC) + 3
(WRj) \leftarrow (dir8)

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 7E(t)5H

0	1	1	1	1	1	0	t	t	t	t	0	1	0	1	dir8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

Binary Mode **Source Mode**

Bytes: 4 3

Cycles: 1 (2 for SFR) 1 (2 for SFR)

Hex: [A5]Encoding Encoding

MOV DRk,Dir8

Operation: (PC) \leftarrow (PC) + 3
(DRk) \leftarrow (dir8)

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 7E(u)DH

0	1	1	1	1	1	1	0	u	u	u	u	1	1	0	1	dir8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

Binary Mode **Source Mode**

Bytes: 4 3

Cycles: 2 (4 for SFR) 2 (4 for SFR)

Hex: [A5]Encoding Encoding

MOV Rm,Dir16

Operation: (PC) \leftarrow (PC) + 4
(Rm) \leftarrow (dir16)

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 7E(s)3H

0	1	1	1	1	1	0	s	s	s	s	0	0	1	1	dir16 hi	dir16 lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode **Source Mode**

Bytes: 5 4

Cycles: 1 1

Hex: [A5]Encoding Encoding

MOV WRj,Dir16

Operation: (PC) \leftarrow (PC) + 4
(WRj) \leftarrow (dir16)

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 7E(t)7H

0	1	1	1	1	1	0	t	t	t	t	0	1	1	1	dir16 hi	dir16 lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode **Source Mode**

Bytes: 5 4

Cycles: 2 2

Hex: [A5]Encoding Encoding

MOV DRk,Dir16

Operation: (PC) \leftarrow (PC) + 4
(DRk) \leftarrow (dir16)

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 7E(u)FH

0	1	1	1	1	1	0	u	u	u	u	1	1	1	1	dir16 hi	dir16 lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode **Source Mode**

Bytes: 5 4

Cycles: 2 2

Hex: [A5]Encoding Encoding

MOV Rm,@WRj

Operation: (PC) \leftarrow (PC) + 3
(Rm) \leftarrow ((WRj))

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 7E(t)9(s)0H

0	1	1	1	1	1	0	t	t	t	t	1	0	0	1	s	s	s	s	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 4 3
 Cycles: 2 2
 Hex: [A5]Encoding Encoding

MOV Rm,@DRk

Operation: (PC) $\leftarrow (PC) + 3$

(Rm) $\leftarrow ((DRk))$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 7E(u)B(s)0H

0	1	1	1	1	1	1	0	u	u	u	u	1	0	1	1	s	s	s	s	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 4 3
 Cycles: 3 3
 Hex: [A5]Encoding Encoding

MOV WRjd,@WRjs

Operation: (PC) $\leftarrow (PC) + 3$

(WRjd) $\leftarrow ((WRjs))$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 0B(u)8(s)0H

0	0	0	0	1	0	1	1	u	u	u	u	1	0	0	0	s	s	s	s	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 4 3
 Cycles: 1 1
 Hex: [A5]Encoding Encoding

MOV WRj,@DRk

Operation: (PC) $\leftarrow (PC) + 3$

(WRj) $\leftarrow ((DRk))$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 0B(u)A(t)0H

0	0	0	0	1	0	1	1	u	u	u	u	1	0	1	0	t	t	t	t	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 4 3
 Cycles: 4 4
 Hex: [A5]Encoding Encoding

MOV Dir8,Rm

Operation: (PC) $\leftarrow (PC) + 3$

(dir8) $\leftarrow (Rm)$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 7A(s)1H

0	1	1	1	1	0	1	0	s	s	s	s	0	0	0	1	dir8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

Binary Mode**Source Mode****Bytes:**

4

3

Cycles:

1

1

Hex:

[A5]Encoding

Encoding

MOV Dir8,WRj

Operation: (PC) \leftarrow (PC) + 3
(dir8) \leftarrow (WRj)

Flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Encoding]

7A(t)5H

0	1	1	1	1	0	1	0	t	t	t	t	0	1	0	1	dir8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

Binary Mode**Source Mode****Bytes:**

4

3

Cycles:

2

2

Hex:

[A5]Encoding

Encoding

MOV Dir8,DRk

Operation: (PC) \leftarrow (PC) + 3
(dir8) \leftarrow (DRk)

Flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Encoding]

7A(u)DH

0	1	1	1	1	0	1	0	u	u	u	u	1	1	0	1	dir8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

Binary Mode**Source Mode****Bytes:**

4

3

Cycles:

2

2

Hex:

[A5]Encoding

Encoding

MOV Dir16,Rm

Operation: (PC) \leftarrow (PC) + 4
(dir16) \leftarrow (Rm)

Flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Encoding]

7A(s)3H

0	1	1	1	1	0	1	0	s	s	s	s	0	0	1	1	dir16 hi	dir16 lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode**Source Mode****Bytes:**

5

4

Cycles:

1

1

Hex:

[A5]Encoding

Encoding

MOV Dir16,WRj

Operation: (PC) \leftarrow (PC) + 4
(dir16) \leftarrow (WRj)

Flags:

CY	AC	OV	N	Z
----	----	----	---	---

	-	-	-	-	-
[Encoding]	7A(t)7H				
	0	1	1	1	1 0 1 0 t t t t 0 1 1 1 dir16 hi dir16 lo
	Binary Mode		Source Mode		
Bytes:	5		4		
Cycles:	2		2		
Hex:	[A5]Encoding		Encoding		

MOV Dir16,DRk

Operation:	(PC)	$\leftarrow (PC) + 4$			
	(dir16)	$\leftarrow (DRk)$			
Flags:	CY	AC	OV	N	Z
	-	-	-	-	-
[Encoding]	7A(u)FH				
	0	1	1	1	1 0 1 0 u u u u 1 1 1 1 dir16 hi dir16 lo
	Binary Mode		Source Mode		
Bytes:	5		4		
Cycles:	2		2		
Hex:	[A5]Encoding		Encoding		

MOV @WRj,Rm

Operation:	(PC)	$\leftarrow (PC) + 3$			
	((WRj))	$\leftarrow (Rm)$			
Flags:	CY	AC	OV	N	Z
	-	-	-	-	-
[Encoding]	7A(t)9(s)0H				
	0	1	1	1	1 0 1 0 t t t t 1 0 0 1 s s s s 0 0 0 0
	Binary Mode		Source Mode		
Bytes:	4		3		
Cycles:	1		1		
Hex:	[A5]Encoding		Encoding		

MOV @DRk,Rm

Operation:	(PC)	$\leftarrow (PC) + 3$			
	((DRk))	$\leftarrow (Rm)$			
Flags:	CY	AC	OV	N	Z
	-	-	-	-	-
[Encoding]	7A(u)B(s)0H				
	0	1	1	1	1 0 1 0 u u u u 1 0 1 1 s s s s 0 0 0 0
	Binary Mode		Source Mode		
Bytes:	4		3		
Cycles:	4		4		
Hex:	[A5]Encoding		Encoding		

MOV @WRjd,WRjs

Operation:	(PC)	$\leftarrow (PC) + 3$
------------	------	-----------------------

	((WRjd))	←((WRjs))			
Flags:	CY	AC	OV	N	Z
	-	-	-	-	-
[Encoding]	1B(t)8(T)0H				
	0	0	0	1	1 0 1 1 t t t t 1 0 0 0 T T T T 0 0 0 0
	Binary Mode		Source Mode		
Bytes:	4		3		
Cycles:	3		3		
Hex:	[A5]Encoding		Encoding		

MOV @DRk,WRj

Operation:	(PC)	←(PC) + 3			
	((DRk))	←(WRj)			
Flags:	CY	AC	OV	N	Z
	-	-	-	-	-
[Encoding]	1B(u)A(t)0H				
	0	0	0	1	1 0 1 1 u u u u 1 0 1 0 t t t t 0 0 0 0
	Binary Mode		Source Mode		
Bytes:	4		3		
Cycles:	6		6		
Hex:	[A5]Encoding		Encoding		

MOV Rm,@WRj+dis

Operation:	(PC)	←(PC) + 4			
	(Rm)	←((WRj)+dis)			
Flags:	CY	AC	OV	N	Z
	-	-	-	-	-
[Encoding]	09H				
	0	0	0	0	1 0 0 1 s s s s t t t t dis hi dis lo
	Binary Mode		Source Mode		
Bytes:	5		4		
Cycles:	1		1		
Hex:	[A5]Encoding		Encoding		

MOV WRjd,@WRjs+dis

Operation:	(PC)	←(PC) + 4			
	(WRjd)	←((WRjs)+dis)			
Flags:	CY	AC	OV	N	Z
	-	-	-	-	-
[Encoding]	49H				
	0	1	0	0	1 0 0 1 t t t t T T T T dis hi dis lo
	Binary Mode		Source Mode		
Bytes:	5		4		
Cycles:	1		1		
Hex:	[A5]Encoding		Encoding		

MOV Rm,@DRk+dis

Operation: (PC) $\leftarrow (PC) + 4$
 (Rm) $\leftarrow ((DRk)+dis)$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 29H

0	0	1	0	1	0	0	1	s	s	s	s	u	u	u	u	dis hi	dis lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--------	--------

Binary Mode **Source Mode**

Bytes: 5 4

Cycles: 3 3

Hex: [A5]Encoding Encoding

MOV WRj,@DRk+dis

Operation: (PC) $\leftarrow (PC) + 4$
 (WRj) $\leftarrow ((DRk)+dis)$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 69H

0	1	1	0	1	0	0	1	t	t	t	t	u	u	u	u	dis hi	dis lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--------	--------

Binary Mode **Source Mode**

Bytes: 5 4

Cycles: 4 4

Hex: [A5]Encoding Encoding

MOV @WRj+dis,Rm

Operation: (PC) $\leftarrow (PC) + 4$
 ((WRj)+dis) $\leftarrow (Rm)$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 19H

0	0	0	1	1	0	0	1	s	s	s	s	t	t	t	t	dis hi	dis lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--------	--------

Binary Mode **Source Mode**

Bytes: 5 4

Cycles: 1 1

Hex: [A5]Encoding Encoding

MOV @WRjd+dis,WRjs

Operation: (PC) $\leftarrow (PC) + 4$
 ((WRjd)+dis) $\leftarrow (WRjs)$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 59H

0	1	0	1	1	0	0	1	T	T	T	T	t	t	t	t	dis hi	dis lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--------	--------

Binary Mode **Source Mode**

Bytes: 5 4

Cycles: 3 3

Hex: [A5]Encoding Encoding

MOV @DRk+dis,Rm

Operation: (PC) \leftarrow (PC) + 4
 ((DRk)+dis) \leftarrow (Rm)

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 39H

0	0	1	1	1	0	0	1	s	s	s	s	u	u	u	u	dis hi	dis lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--------	--------

Binary Mode **Source Mode**

Bytes: 5 4

Cycles: 4 4

Hex: [A5]Encoding Encoding

MOV @DRk+dis,WRj

Operation: (PC) \leftarrow (PC) + 4
 ((DRk)+dis) \leftarrow (WRj)

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 79H

0	1	1	1	1	0	0	1	t	t	t	t	u	u	u	u	dis hi	dis lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--------	--------

Binary Mode **Source Mode**

Bytes: 5 4

Cycles: 6 6

Hex: [A5]Encoding Encoding

MOV <dest-bit>,<src-bit>**Function:** Move bit data

Description: The Boolean variable indicated by the second operand (directly addressable bit) is copied into carry flag. No other register or flag is affected.

MOV C,Bit51

Operation: (PC) \leftarrow (PC) + 2
 (C) \leftarrow (bit51)

Flags:	CY	AC	OV	N	Z
	✓	-	-	-	-

[Encoding] A2H

1	0	1	0	0	0	1	0	bit address
---	---	---	---	---	---	---	---	-------------

Binary Mode **Source Mode**

Bytes: 2 2

Cycles: 1 1

Hex: Encoding Encoding

MOV C,Bit

Operation: (PC) \leftarrow (PC) + 3
 (C) \leftarrow (bit)

Flags:

CY	AC	OV	N	Z
✓	-	-	-	-

[Encoding] A9A(y)H

1	0	1	0	1	0	0	1	1	0	1	0	y	y	y	y	dir	addr
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----	------

Binary Mode**Source Mode****Bytes:**

4

3

Cycles:

1

1

Hex:

[A5]Encoding

Encoding

MOV Bit51,C**Operation:** (PC) \leftarrow (PC) + 2(bit51) \leftarrow (C)**Flags:**

CY	AC	OV	N	Z
-	-	-	-	-

[Encoding] 92H

1	0	0	1	0	0	1	0	bit	address
---	---	---	---	---	---	---	---	-----	---------

Binary Mode**Source Mode****Bytes:**

2

2

Cycles:

1

1

Hex:

Encoding

Encoding

MOV Bit,C**Operation:** (PC) \leftarrow (PC) + 3(bit) \leftarrow (C)**Flags:**

CY	AC	OV	N	Z
✓	-	-	-	-

[Encoding] A99(y)H

1	0	1	0	1	0	0	1	1	0	0	1	y	y	y	y	dir	addr
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----	------

Binary Mode**Source Mode****Bytes:**

4

3

Cycles:

1

1

Hex:

[A5]Encoding

Encoding

MOV DPTR,#DATA16**Function:** Load data pointer with a 16-bit constant

Description: The data pointer is loaded with the 16-bit constant indicated. The 16 bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected. This is the only instruction which moves 16 bits of data at once.

Operation: (PC) \leftarrow (PC) + 3DPH \leftarrow immediate data15..8DPL \leftarrow immediate data7..0**Flags:**

CY	AC	OV	N	Z
-	-	-	-	-

[Encoding] 90H

1	0	0	1	0	0	0	0	immediate data15..8	immediate data7..0
---	---	---	---	---	---	---	---	---------------------	--------------------

	Binary Mode	Source Mode
Bytes:	3	3
Cycles:	1	1
Hex:	Encoding	Encoding

MOVC

Function: Move code byte

Description: The MOVC instructions load the accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit accumulator contents and the contents of a sixteen-bit base register, which may be either the data pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added to the accumulator; otherwise, the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

MOVC A,@A+DPTR

Operation: (PC) $\leftarrow (PC) + 1$
(A) $\leftarrow ((A) + (DPTR))$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 93H

1	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---

Binary Mode	Source Mode
-------------	-------------

Bytes:	1	1
Cycles:	4	4
Hex:	Encoding	Encoding

MOVC A,@A+PC

Operation: (PC) $\leftarrow (PC) + 1$
(A) $\leftarrow ((A) + (PC))$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 83H

1	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

Binary Mode	Source Mode
-------------	-------------

Bytes:	1	1
Cycles:	3	3
Hex:	Encoding	Encoding

MOVH DRk,#DATA16

Function: Move immediate 16-bit data to the high word of a dword (double-word) register.

Description: Moves 16-bit immediate data to the high word of a dword (32-bit) register. The low word of the dword register is unchanged.

Operation: (PC) $\leftarrow (PC) + 4$
(DRk).31-16 $\leftarrow \#data16$

Flags:	CY	AC	OV	N	Z
---------------	----	----	----	---	---

	-	-	-	-	-
[Encoding]	7A(u)CH				
	0	1	1	1	1 0 1 0 u u u u 1 1 0 0 data hi data lo
	Binary Mode		Source Mode		
Bytes:	5		4		
Cycles:	1		1		
Hex:	[A5]Encoding		Encoding		

MOVS WRj,Rm

Function: Move 8-bit register to 16-bit register with sign extension

Description: Moves the contents of an 8-bit register to the low byte of a 16-bit register. The high byte of the 16-bit register is filled with the sign extension, which is obtained from the most significant bit of the 8-bit source register.

Operation: (PC) \leftarrow (PC) + 2
 (WRj).7-0 \leftarrow (Rm)
 (WRj).15-8 \leftarrow sign extension

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding]	1AH				
	0	0	0	1	1 0 1 0 t t t t s s s s
	Binary Mode		Source Mode		
Bytes:	3		2		
Cycles:	1		1		
Hex:	[A5]Encoding		Encoding		

MOVX

Function: Move external

Description: The MOVX instructions transfer data between the accumulator and a byte of external data memory, hence, the X appended to MOV. There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM. In the first type, the contents of R0 or R1 in the current register bank provides an eight-bit address, in the second type of MOVX instructions, the data pointer generates a sixteen-bit address.

MOVX A,@Ri

Operation: (PC) \leftarrow (PC) + 1
 (A) \leftarrow ((Ri))

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding]	E2H, E3H				
	1	1	1	0	0 0 1 i
	Binary Mode		Source Mode		
Bytes:	1		1		
Cycles:	3*		3*		
Hex:	Encoding		Encoding		

* MOVX cycles depend on STRETCH register. Shown values with STRETCH=0.

MOVX A,@DPTR

Operation: (PC) $\leftarrow (PC) + 1$
 (A) $\leftarrow ((DPTR))$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] E0H

1	1	1	0	0	0	0	0
---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 1 1

Cycles: 2* 2*

Hex: Encoding Encoding

* MOVX cycles depend on STRETCH register. Shown values with STRETCH=0.

MOVX @Ri,A

Operation: (PC) $\leftarrow (PC) + 1$
 ((Ri)) $\leftarrow (A)$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] F2H, F3H

1	1	1	1	0	0	1	i
---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 1 1

Cycles: 3* 3*

Hex: Encoding Encoding

* MOVX cycles depend on STRETCH register. Shown values with STRETCH=0.

MOVX @DPTR,A

Operation: (PC) $\leftarrow (PC) + 1$
 ((DPTR)) $\leftarrow (A)$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] F0H

1	1	1	1	0	0	0	0
---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 1 1

Cycles: 3* 3*

Hex: Encoding Encoding

* MOVX cycles depend on STRETCH register. Shown values with STRETCH=0.

MOVZ WRj,Rm

Function: Move 8-bit register to 16-bit register with zero extension

Description: Moves the contents of an 8-bit register to the low byte of a 16-bit register. The high byte of the 16-bit register is filled with the zeros.

Operation: (PC) $\leftarrow (PC) + 2$
 (WRj).7-0 $\leftarrow (Rm)$
 (WRj).15-8 $\leftarrow 0$

Flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Encoding] 0AH

0	0	0	0	1	0	1	0	t	t	t	t	s	s	s	s
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode**Source Mode****Bytes:**

3

2

Cycles:

1

1

Hex:

[A5]Encoding

Encoding

MUL**Function:** Multiply

Description: Multiplies the unsigned integer in the source register with the unsigned integer in the destination register. Only register addressing is allowed. For 8-bit operands, the result is 16 bits. The most significant byte of the result is stored in the low byte of the word where the destination register resides. The least significant byte is stored in the following byte register. The OV flag is set if the product is greater than 255 (0FFH); otherwise it is cleared. For 16-bit operands, the result is 32 bits. The most significant word is stored in the low word of the dword where the destination register resides. The least significant word is stored in the following word register. In this operation, the OV flag is set if the product is greater than 0FFFFH, otherwise it is cleared. The CY flag is always cleared. The N flag is set when the MSB of the result is set. The Z flag is set when the result is zero.

MUL AB

Operation: (PC) \leftarrow (PC) + 1
 (A) \leftarrow (A) \times (B) -result's bits 7..0
 (B) \leftarrow (A) \times (B) -result's bits 15..8

Flags:

CY	AC	OV	N	Z
0	-	✓	✓	✓

[Encoding] A4H

1	0	1	0	0	1	0	0
---	---	---	---	---	---	---	---

Binary Mode**Source Mode****Bytes:**

1

1

Cycles:

1

1

Hex:

Encoding

Encoding

MUL Rmd,Rms

Operation: (PC) \leftarrow (PC) + 2
 if <dest>md = 0,2,4..,14
 Rmd \leftarrow high byte of the Rmd \times Rms
 Rmd+1 \leftarrow low byte of the Rmd \times Rms
 if <dest>md = 1,3,5..,15
 Rmd-1 \leftarrow high byte of the Rmd \times Rms
 Rmd \leftarrow low byte of the Rmd \times Rms

Flags:

CY	AC	OV	N	Z
0	-	✓	✓	✓

[Encoding] ACH

1	0	1	0	1	1	0	0	s	s	s	s	S	S	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

	Binary Mode	Source Mode
Bytes:	3	2
Cycles:	1	1
Hex:	[A5]Encoding	Encoding

MUL WRjd,WRjs

Operation: $(PC) \leftarrow (PC) + 2$
 if $\langle \text{dest} \rangle \text{jd} = 0, 4, 8, \dots, 28$
 $\text{WRjd} \leftarrow \text{high byte of the } \text{WRjd} \times \text{WRjs}$
 $\text{WRjd}+2 \leftarrow \text{low byte of the } \text{WRjd} \times \text{WRjs}$
 if $\langle \text{dest} \rangle \text{jd} = 2, 6, 10, \dots, 30$
 $\text{WRjd}-2 \leftarrow \text{high byte of the } \text{WRjd} \times \text{WRjs}$
 $\text{WRjd} \leftarrow \text{low byte of the } \text{WRjd} \times \text{WRjs}$

Flags:	CY	AC	OV	N	Z
	0	-	✓	✓	✓

[Encoding] ADH

1	0	1	0	1	1	0	1	t	t	t	T	T	T	T	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

	Binary Mode	Source Mode
Bytes:	3	2
Cycles:	1	1
Hex:	[A5]Encoding	Encoding

NOP**Function:** No operation**Description:** Execution continues at the following instruction. Other than the PC, no registers or flags are affected.**Operation:** $(PC) \leftarrow (PC) + 1$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 00H

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

	Binary Mode	Source Mode
Bytes:	1	1
Cycles:	1	1
Hex:	Encoding	Encoding

ORL**Function:** Logical OR for variables

Description: Performs the bitwise logical-OR operation between the specified variables, storing the results in the destination operand. The destination operand can be a register, an accumulator or direct address. The two operands allow twelve addressing mode combinations. When the destination is the accumulator, the source can be register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data. When the destination is register the source can be register, immediate, direct and indirect addressing. Only the N and Z flags are affected.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

ORL A,Rn**Operation:** (PC) $\leftarrow (PC) + 1$ (A) $\leftarrow (A) \text{ or } (Rn)$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 48H - 4FH

0	1	0	0	1	r	r	r
---	---	---	---	---	---	---	---

Binary Mode**Source Mode****Bytes:** 1 2**Cycles:** 1 1**Hex:** Encoding [A5]Encoding**ORL A,Dir****Operation:** (PC) $\leftarrow (PC) + 2$ (A) $\leftarrow (A) \text{ or } (\text{direct})$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 45H

0	1	0	0	0	1	0	1	direct address
---	---	---	---	---	---	---	---	----------------

Binary Mode**Source Mode****Bytes:** 2 2**Cycles:** 1 1**Hex:** Encoding Encoding**ORL A,@Ri****Operation:** (PC) $\leftarrow (PC) + 1$ (A) $\leftarrow (A) \text{ or } ((Ri))$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 46H, 47H

0	1	0	0	0	1	1	i
---	---	---	---	---	---	---	---

Binary Mode**Source Mode****Bytes:** 1 2**Cycles:** 1 1**Hex:** Encoding [A5]Encoding**ORL A,#DATA****Operation:** (PC) $\leftarrow (PC) + 1$ (A) $\leftarrow (A) \text{ or } \#data$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 44H

0	1	0	0	0	1	0	0	immediate data
---	---	---	---	---	---	---	---	----------------

Binary Mode**Source Mode****Bytes:** 2 2

Cycles: 1 1
Hex: Encoding Encoding

ORL Dir,A

Operation: (PC) $\leftarrow (PC) + 1$
 (direct) $\leftarrow (\text{direct}) \text{ or } (A)$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 42H

0	1	0	0	0	0	1	0	direct address
---	---	---	---	---	---	---	---	----------------

Binary Mode **Source Mode**

Bytes: 2 2

Cycles: 1 1

Hex: Encoding Encoding

ORL Dir,#DATA

Operation: (PC) $\leftarrow (PC) + 1$
 (direct) $\leftarrow (\text{direct}) \text{ or } \#data$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 43H

0	1	0	0	0	0	1	1	direct address	immediate data
---	---	---	---	---	---	---	---	----------------	----------------

Binary Mode **Source Mode**

Bytes: 3 3

Cycles: 1 1

Hex: Encoding Encoding

ORL Rmd,Rms

Operation: (PC) $\leftarrow (PC) + 2$
 (Rmd) $\leftarrow (Rms) \text{ or } (Rmd)$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 4CH

0	1	0	0	1	1	0	0	s	s	s	s	S	S	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 3 2

Cycles: 1 1

Hex: [A5]Encoding Encoding

ORL WRjd,WRjs

Operation: (PC) $\leftarrow (PC) + 2$
 (WRjd) $\leftarrow (WRjs) \text{ or } (WRjd)$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 4DH

0	1	0	0	1	1	0	1	t	t	t	t	S	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

	Binary Mode	Source Mode
Bytes:	3	2
Cycles:	1	1
Hex:	[A5]Encoding	Encoding

ORL Rm,#DATA

Operation: (PC) $\leftarrow (PC) + 3$
 (Rm) $\leftarrow (Rm)$ or #data

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 4E(s)0H

0	1	0	0	1	1	1	0	s	s	s	s	0	0	0	0	#data
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-------

	Binary Mode	Source Mode
Bytes:	4	3
Cycles:	1	1
Hex:	[A5]Encoding	Encoding

ORL WRj,#DATA16

Operation: (PC) $\leftarrow (PC) + 4$
 (WRj) $\leftarrow (WRj)$ or #data16

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 4E(t)4H

0	1	0	0	1	1	1	0	t	t	t	t	0	1	0	0	#data hi	#data lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

	Binary Mode	Source Mode
Bytes:	5	4
Cycles:	1	1
Hex:	[A5]Encoding	Encoding

ORL Rm,Dir8

Operation: (PC) $\leftarrow (PC) + 3$
 (Rm) $\leftarrow (Rm)$ or (dir8)

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 4E(s)1H

0	1	0	0	1	1	1	0	s	s	s	s	0	0	0	1	dir8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

	Binary Mode	Source Mode
Bytes:	4	3
Cycles:	1	1
Hex:	[A5]Encoding	Encoding

ORL WRj,Dir8

Operation: (PC) $\leftarrow (PC) + 3$
 (WRj) $\leftarrow (WRj)$ or (dir8)

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 4E(t)5H

0	1	0	0	1	1	1	0	t	t	t	t	0	1	0	1	dir8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

Binary Mode

Source Mode

Bytes: 4

3

Cycles: 1 (2 for SFR)

1 (2 for SFR)

Hex: [A5]Encoding

Encoding

ORL Rm,Dir16

Operation: (PC) \leftarrow (PC) + 4
 (Rm) \leftarrow (Rm) or (dir16)

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 4E(s)3H

0	1	0	0	1	1	1	0	s	s	s	s	0	0	1	1	dir16 hi	dir16 lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode

Source Mode

Bytes: 5

4

Cycles: 1

1

Hex: [A5]Encoding

Encoding

ORL WRj,Dir16

Operation: (PC) \leftarrow (PC) + 4
 (WRj) \leftarrow (WRj) or (dir16)

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 4E(t)7H

0	1	0	0	1	1	1	0	t	t	t	t	0	1	1	1	dir16 hi	dir16 lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode

Source Mode

Bytes: 5

4

Cycles: 2

2

Hex: [A5]Encoding

Encoding

ORL Rm,@WRj

Operation: (PC) \leftarrow (PC) + 3
 (Rm) \leftarrow (Rm) or ((WRj))

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 4E(t)9(s)0H

0	1	0	0	1	1	1	0	t	t	t	t	1	0	0	1	s	s	s	s	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 4

3

Cycles: 1

1

Hex: [A5]Encoding

Encoding

ORL Rm,@DRk

Operation: (PC) \leftarrow (PC) + 3
 (Rm) \leftarrow (Rm) or ((DRk))

Flags:

CY	AC	OV	N	Z
-	-	-	✓	✓

[Encoding] 4E(u)B(s)0H

0	1	0	0	1	1	1	0	u	u	u	U	1	0	1	1	s	s	s	s	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes:

4

3

Cycles:

3

3

Hex:

[A5]Encoding

Encoding

ORL CY,<src-bit>

Function: Logical OR for bit variables

Description: Set the carry flag if the Boolean value is logic 1; leave the carry in its current state otherwise. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

ORL C,Bit51

Operation: (PC) \leftarrow (PC) + 2
(C) \leftarrow (C) or (bit51)

Flags:

CY	AC	OV	N	Z
✓	-	-	-	-

[Encoding] 72H

0	1	1	1	0	0	1	0	bit address
---	---	---	---	---	---	---	---	-------------

Binary Mode

Source Mode

Bytes:

2

2

Cycles:

1

1

Hex:

Encoding

Encoding

ORL C,/Bit51

Operation: (PC) \leftarrow (PC) + 2
(C) \leftarrow (C) or /(bit51)

Flags:

CY	AC	OV	N	Z
✓	-	-	-	-

[Encoding] A0H

1	0	1	0	0	0	0	0	bit address
---	---	---	---	---	---	---	---	-------------

Binary Mode

Source Mode

Bytes:

2

2

Cycles:

1

1

Hex:

Encoding

Encoding

ORL C,Bit

Operation: (PC) \leftarrow (PC) + 2
(C) \leftarrow (C) or (bit)

Flags:

CY	AC	OV	N	Z
✓	-	-	-	-

[Encoding] A97(y)H

1	0	1	0	1	0	0	1	0	1	1	1	0	y	y	y	dir address
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-------------

Binary Mode**Source Mode****Bytes:**

4

3

Cycles:

1

1

Hex:

[A5]Encoding

Encoding

ORL C/Bit**Operation:**

(PC)

 $\leftarrow (PC) + 2$

(C)

 $\leftarrow (C) \text{ or } /(bit)$ **Flags:**

CY

AC

OV

N

Z

✓

-

-

-

-

[Encoding]

A9E(y)H

1 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | y | y | y | dir address |**Binary Mode****Source Mode****Bytes:**

4

3

Cycles:

1

1

Hex:

[A5]Encoding

Encoding

POP**Function:**

POP from stack

Description:

Reads the contents of the on-chip RAM location addressed by the stack pointer, then decrements the stack pointer by one. The value read at the original RAM location is transferred to the newly addressed location, which can be 8-bit or 16-bit. No flags are affected.

POP Dir8**Operation:**

(PC)

 $\leftarrow (PC) + 2$

(dir8)

 $\leftarrow ((SP))$

(SP)

 $\leftarrow (SP) - 1$ **Flags:**

CY

AC

OV

N

Z

-

-

-

-

-

[Encoding]

DOH

1 1 | 0 | 1 | 0 | 0 | 0 | 0 | direct address |**Binary Mode****Source Mode****Bytes:**

2

2

Cycles:

1

1

Hex:

Encoding

Encoding

POP Rm**Operation:**

(PC)

 $\leftarrow (PC) + 2$

(Rm)

 $\leftarrow ((SP))$

(SP)

 $\leftarrow (SP) - 1$ **Flags:**

CY

AC

OV

N

Z

-

-

-

-

-

[Encoding]

DA(s)8H

1 1 | 0 | 1 | 1 | 0 | 1 | 0 | s | s | s | s | 1 | 0 | 0 | 0 |**Binary Mode****Source Mode**

Bytes: 3 2
Cycles: 1 1
Hex: [A5]Encoding Encoding

POP WRj

Operation: (PC) $\leftarrow (PC) + 2$
 (WRj) $\leftarrow ((SP))$
 (SP) $\leftarrow (SP) - 2$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] DA(t)9H

1	1	0	1	1	0	1	0	t	t	t	t	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode Source Mode

Bytes: 3 2
Cycles: 1 1
Hex: [A5]Encoding Encoding

POP DRk

Operation: (PC) $\leftarrow (PC) + 2$
 (DRk) $\leftarrow ((SP))$
 (SP) $\leftarrow (SP) - 3$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] DA(u)BH

1	1	0	1	1	0	1	0	u	u	u	u	1	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode Source Mode

Bytes: 3 2
Cycles: 1 1
Hex: [A5]Encoding Encoding

PUSH

Function: PUSH onto stack

Description: Increments the stack pointer by one. The contents of the specified variable are then copied into the on-chip RAM location addressed by the stack pointer. No flags are affected.

PUSH Dir8

Operation: (PC) $\leftarrow (PC) + 2$
 (SP) $\leftarrow (SP) + 1$
 ((SP)) $\leftarrow (\text{dir8})$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] COH

1	1	0	0	0	0	0	0	direct address
---	---	---	---	---	---	---	---	----------------

Binary Mode Source Mode

Bytes: 2 2
Cycles: 1 1

Hex: Encoding Encoding

PUSH #DATA

Operation: (PC) \leftarrow (PC) + 2
 (SP) \leftarrow (SP) + 1
 ((SP)) \leftarrow #data

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] CA02H

1	1	0	0	1	0	1	0	0	0	0	0	0	1	0	data
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

Binary Mode **Source Mode****Bytes:** 4 3**Cycles:** 1 1**Hex:** [A5]Encoding Encoding**PUSH #DATA16**

Operation: (PC) \leftarrow (PC) + 2
 (SP) \leftarrow (SP) + 1
 ((SP)) \leftarrow MSB #data
 (SP) \leftarrow (SP) + 1
 ((SP)) \leftarrow LSB #data

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] CA06H

1	1	0	0	1	0	1	0	0	0	0	0	0	1	1	0	data16 hi	data16 lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----------	-----------

Binary Mode **Source Mode****Bytes:** 5 4**Cycles:** 1 1**Hex:** [A5]Encoding Encoding**PUSH Rm**

Operation: (PC) \leftarrow (PC) + 2
 (SP) \leftarrow (SP) + 1
 ((SP)) \leftarrow (Rm)

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] CA(s)8H

1	1	0	0	1	0	1	0	s	s	s	s	1	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode **Source Mode****Bytes:** 3 2**Cycles:** 1 1**Hex:** [A5]Encoding Encoding**PUSH WRj**

Operation: (PC) \leftarrow (PC) + 2
 (SP) \leftarrow (SP) + 1

	((SP))	$\leftarrow(\text{WRj})$			
	(SP)	$\leftarrow(\text{SP}) + 1$			
Flags:	CY	AC	OV	N	Z
	-	-	-	-	-
[Encoding]	CA(t)9H				
	1	1	0	0	1
	Binary Mode		Source Mode		
Bytes:	3		2		
Cycles:	1		1		
Hex:	[A5]Encoding		Encoding		

PUSH DRk

Operation:	(PC)	$\leftarrow(\text{PC}) + 2$														
	(SP)	$\leftarrow(\text{SP}) + 1$														
	((SP))	$\leftarrow(\text{DRk})$														
	(SP)	$\leftarrow(\text{SP}) + 3$														
Flags:	CY	AC	OV	N	Z											
	-	-	-	-	-											
[Encoding]	CA(u)BH															
	1	1	0	0	1	0	1	0	u	u	u	u	1	0	1	1
	Binary Mode				Source Mode											
Bytes:	3				2											
Cycles:	1				1											
Hex:	[A5]Encoding				Encoding											

RET

Function:	Return from subroutine				
Description:	RET pops the high and low-order bytes of the PC successively from the stack, decrementing the stack pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.				
Operation:	(PC15-8)	$\leftarrow ((\text{SP}))$			
	(SP)	$\leftarrow (\text{SP}) - 1$			
	(PC7-0)	$\leftarrow ((\text{SP}))$			
	(SP)	$\leftarrow (\text{SP}) - 1$			
Flags:	CY	AC	OV	N	Z
	-	-	-	-	-
[Encoding]	22H				
	0	0	1	0	0
	Binary Mode		Source Mode		
Bytes:	1		1		
Cycles:	3		3		
Hex:	Encoding		Encoding		

RETI

Function:	Return from interrupt				
Description:	This instruction pops two or four bytes from the stack, depending on the INTR bit in the CONFIG1				

register. If INTR = 0, RETI pops the high and low bytes of the PC successively from the stack and uses them as the 16-bit return address in region FF:. The stack pointer is decremented by two. No other registers are affected, and neither PSW nor PSW1 is automatically restored to its pre-interrupt status. If INTR = 1, RETI pops four bytes from the stack: PSW1 and the three bytes of the PC. The three bytes of the PC are the return address, which can be anywhere in the 16-Mbyte memory space. The stack pointer is decremented by four. PSW1 is restored to its pre-interrupt status, but PSW is not restored to its pre-interrupt status. No other registers are affected. For either value of INTR, hardware restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. Program execution continues at the return address, which normally is the instruction immediately after the point at which the interrupt request was detected. If an interrupt of the same or lower-priority is pending when the RETI instruction is executed, that one instruction is executed before the pending interrupt is processed.

Operation:	INTR=1:		INTR=0:	
	(PC15-8)	$\leftarrow((SP))$	(PC15-8)	$\leftarrow((SP))$
	(SP)	$\leftarrow(SP) - 1$	(SP)	$\leftarrow(SP) - 1$
	(PC7-0)	$\leftarrow((SP))$	(PC7-0)	$\leftarrow((SP))$
	(SP)	$\leftarrow(SP) - 1$	(SP)	$\leftarrow(SP) - 1$
	(PC23-16)	$\leftarrow((SP))$		
	(SP)	$\leftarrow(SP) - 1$		
	PSW1	$\leftarrow((SP))$		
	(SP)	$\leftarrow(SP) - 1$		
Flags:	CY	AC	OV	N
	-	-	-	-
[Encoding]	32H			
	0 0 1 1		0 0 1 0	
	Binary Mode		Source Mode	
Bytes:	1		1	
Cycles:	3		3	
Hex:	Encoding		Encoding	

RL

Function: Rotate accumulator left

Description: The eight bits in the accumulator are rotated one bit to the left. The bit 7 is rotated into the bit 0 position. Only the N and Z flags are affected.

Operation: (PC) ←(PC) + 1

(An + 1) ←(An) n = 0-6

(A0) ←(A7)

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding]	23H					
	0 0 1 0			0 0 1 1		
	Binary Mode			Source Mode		
Bytes:	1			1		
Cycles:	1			1		
Hex:	Encoding			Encoding		

RLC

Function: Rotate accumulator left through carry flag

Description: The eight bits in the accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. The N and Z flags are also affected.

Operation: (PC) \leftarrow (PC) + 1
 (An + 1) \leftarrow (An) n = 0-6
 (A0) \leftarrow (C)
 (C) \leftarrow (A7)

Flags:	CY	AC	OV	N	Z
	✓	-	-	✓	✓

[Encoding] 33H

0	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 1 1

Cycles: 1 1

Hex: Encoding Encoding

RR

Function: Rotate accumulator right

Description: The eight bits in the accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. Only the N and Z flags are affected.

Operation: (PC) \leftarrow (PC) + 1
 (An) \leftarrow (An + 1) n = 0-6
 (A7) \leftarrow (A0)

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 03H

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 1 1

Cycles: 1 1

Hex: Encoding Encoding

RLC

Function: Rotate accumulator right through carry flag

Description: The eight bits in the accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position. The N and Z flags are also affected.

Operation: (PC) \leftarrow (PC) + 1
 (An) \leftarrow (An + 1) n = 0-6
 (A7) \leftarrow (C)
 (C) \leftarrow (A0)

Flags:	CY	AC	OV	N	Z
	✓	-	-	✓	✓

[Encoding] 13H

0	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---

Binary Mode**Source Mode****Bytes:**

1

1

Cycles:

1

1

Hex:

Encoding

Encoding

SETB**Function:** Set bit

Description: SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit.
No other flags are affected.

SETB C**Operation:** (PC) \leftarrow (PC) + 1(C) \leftarrow 1**Flags:**

CY	AC	OV	N	Z
✓	-	-	-	-

[Encoding] D3H

1	1	0	1	0	0	1	1
---	---	---	---	---	---	---	---

Binary Mode**Source Mode****Bytes:**

1

1

Cycles:

1

1

Hex:

Encoding

Encoding

SETB Bit51**Operation:** (PC) \leftarrow (PC) + 2(bit51) \leftarrow 1**Flags:**

CY	AC	OV	N	Z
-	-	-	-	-

[Encoding] D2H

1	1	0	1	0	0	1	0	bit address
---	---	---	---	---	---	---	---	-------------

Binary Mode**Source Mode****Bytes:**

2

2

Cycles:

3

3

Hex:

Encoding

Encoding

SETB Bit**Operation:** (PC) \leftarrow (PC) + 3(bit) \leftarrow 1**Flags:**

CY	AC	OV	N	Z
-	-	-	-	-

[Encoding] A9D(y)H

1	0	1	0	1	0	0	1	1	1	0	1	0	y	y	y	dir addr
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------

Binary Mode**Source Mode****Bytes:**

4

3

Cycles:

1

1

Hex:

[A5]Encoding

Encoding

SJMP**Function:** Short jump

Description: Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128 bytes preceding this instruction to 127 bytes following it. No flags are affected.

Note: Under the above conditions, the instruction following SJMP will be at 102H. Therefore, the displacement byte of the instruction will be the relative offset (0123H - 0102H)= 21H . In other words, an SJMP with a displacement of 0FEH would be a one-instruction infinite loop.

Operation: (PC) \leftarrow (PC) + 2
(PC) \leftarrow (PC) + rel

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] 80H

1	0	0	0	0	0	0	0	relative address
---	---	---	---	---	---	---	---	------------------

Binary Mode **Source Mode**

Bytes: 2 2
Cycles: 3 3
Hex: Encoding Encoding

SLL**Function:** Shift logical left by 1 bit

Description: Shifts the specified variable to the left by 1 bit, replacing the LSB with zero. The bit shifted out (MSB) is stored in the CY bit. The N and Z flag are also affected.

SLL Rm

Operation: (PC) \leftarrow (PC) + 2
(Rm).a + 1 \leftarrow (Rm).a
(Rm).0 \leftarrow 0
CY \leftarrow (Rm).7

Flags:	CY	AC	OV	N	Z
	✓	-	-	✓	✓

[Encoding] 3E(s)0H

0	0	1	1	1	1	0	s	s	s	s	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 3 2
Cycles: 1 1
Hex: [A5]Encoding Encoding

SLL WRj

Operation: (PC) \leftarrow (PC) + 2
(WRj).b + 1 \leftarrow (WRj).b
(WRj).0 \leftarrow 0
CY \leftarrow (WRj).15

Flags:	CY	AC	OV	N	Z
--------	----	----	----	---	---

	✓	-	-	✓	✓
[Encoding]	3E(t)4H				
	0	0	1	1	1 1 1 0
					t t t t
					0 1 0 0
	Binary Mode		Source Mode		
Bytes:	3		2		
Cycles:	1		1		
Hex:	[A5]Encoding		Encoding		

SRA

Function: Shift arithmetic right by 1 bit

Description: Shifts the specified variable to the arithmetic right by 1 bit. The MSB is unchanged. The bit shifted out (LSB) is stored in the CY bit. The N and Z flag are also affected.

SRA Rm

Operation: (PC) $\leftarrow (PC) + 2$
 (Rm).7 $\leftarrow (Rm).7$
 (Rm).a $\leftarrow (Rm).a + 1$
 CY $\leftarrow (Rm).0$

Flags:	CY	AC	OV	N	Z
	✓	-	-	✓	✓

[Encoding]	0E(s)0H				
	0	0	0	0	1 1 1 0
					s s s s
					0 0 0 0
	Binary Mode		Source Mode		
Bytes:	3		2		
Cycles:	1		1		
Hex:	[A5]Encoding		Encoding		

SRA WRj

Operation: (PC) $\leftarrow (PC) + 2$
 (WRj).15 $\leftarrow (WRj).15$
 (WRj).b $\leftarrow (WRj).b + 1$
 CY $\leftarrow (WRj).0$

Flags:	CY	AC	OV	N	Z
	✓	-	-	✓	✓

[Encoding]	0E(t)4H				
	0	0	0	0	1 1 1 0
					t t t t
					0 1 0 0
	Binary Mode		Source Mode		
Bytes:	3		2		
Cycles:	1		1		
Hex:	[A5]Encoding		Encoding		

SRL

Function: Shift logical right by 1 bit

Description: SRL shifts the specified variable to the right by 1 bit, replacing the MSB with a zero. The bit shifted out (LSB) is stored in the CY bit. The N and Z flag are also affected.

SRL Rm

Operation: (PC) $\leftarrow (PC) + 2$
 (Rm).7 $\leftarrow (Rm).0$
 (Rm).a $\leftarrow (Rm).a + 1$
 CY $\leftarrow (Rm).0$

Flags:	CY	AC	OV	N	Z
	✓	-	-	✓	✓

[Encoding] 1E(s)0H

0	0	0	1	1	1	1	0	s	s	s	s	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 3 2

Cycles: 1 1

Hex: [A5]Encoding Encoding

SRL WRj

Operation: (PC) $\leftarrow (PC) + 2$
 (WRj).15 $\leftarrow 0$
 (WRj).b $\leftarrow (WRj).b + 1$
 CY $\leftarrow (WRj).0$

Flags:	CY	AC	OV	N	Z
	✓	-	-	✓	✓

[Encoding] 1E(t)4H

0	0	0	1	1	1	1	0	t	t	t	t	0	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 3 2

Cycles: 1 1

Hex: [A5]Encoding Encoding

SUB

Function: Subtract

Description: Subtracts the specified variable from the destination operand, leaving the result in the destination operand. SUB sets the CY (borrow) flag if borrow is needed for bit 7. Otherwise, CY is clear. When subtracting signed integers, the OV flag indicates a negative number produced when a negative value is subtracted from a positive value or a positive result when a positive number is subtracted from a negative number. Bit 7 in this description refers to the most significant byte of the operand (8, 16, or 32 bit). The source operand allows four addressing modes: immediate, indirect, register and direct. All flags are affected, excepting AC, which is not affected for word and dword subtractions.

SUB Rmd,Rms

Operation: (PC) $\leftarrow (PC) + 2$
 (Rmd) $\leftarrow (Rms) - (Rmd)$

Flags:	CY	AC	OV	N	Z
	✓	✓	✓	✓	✓

[Encoding] 9CH

1	0	0	1	1	1	0	0	s	s	s	s	S	S	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 3 2
Cycles: 1 1
Hex: [A5]Encoding Encoding

SUB WRjd,WRjs

Operation: (PC) $\leftarrow (PC) + 2$
 (WRjd) $\leftarrow (WRjs) - (WRjd)$

Flags:	CY	AC	OV	N	Z
	✓	-	✓	✓	✓

[Encoding] 9DH

1	0	0	1	1	1	0	1	t	t	t	t	T	T	T	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode Source Mode

Bytes: 3 2
Cycles: 1 1
Hex: [A5]Encoding Encoding

SUB DRkd,DRks

Operation: (PC) $\leftarrow (PC) + 2$
 (DRkd) $\leftarrow (DRks) - (DRkd)$

Flags:	CY	AC	OV	N	Z
	✓	-	✓	✓	✓

[Encoding] 9FH

1	0	0	1	1	1	1	1	u	u	u	u	U	U	U	U
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode Source Mode

Bytes: 3 2
Cycles: 4 4
Hex: [A5]Encoding Encoding

SUB Rm,#DATA

Operation: (PC) $\leftarrow (PC) + 3$
 (Rm) $\leftarrow (Rm) - \#data$

Flags:	CY	AC	OV	N	Z
	✓	✓	✓	✓	✓

[Encoding] 9E(s)0H

1	0	0	1	1	1	1	0	s	s	s	s	0	0	0	0	#data
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-------

Binary Mode Source Mode

Bytes: 4 3
Cycles: 1 1
Hex: [A5]Encoding Encoding

SUB WRj,#DATA16

Operation: (PC) $\leftarrow (PC) + 4$
 (WRj) $\leftarrow (WRj) - \#data16$

Flags:	CY	AC	OV	N	Z
	✓	-	✓	✓	✓

[Encoding] 9E(t)4H

1	0	0	1	1	1	1	0	t	t	t	t	0	1	0	0	#data hi	#data lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode **Source Mode**

Bytes: 5 4

Cycles: 1 1

Hex: [A5]Encoding Encoding

SUB DRk,#0DATA16

Operation: (PC) $\leftarrow (PC) + 4$
 (DRk) $\leftarrow (DRk) - \#data16$

Flags:	CY	AC	OV	N	Z
	✓	-	✓	✓	✓

[Encoding] 9E(u)8H

1	0	0	1	1	1	1	0	u	u	u	u	1	0	0	0	#data hi	#data lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode **Source Mode**

Bytes: 5 4

Cycles: 1 1

Hex: [A5]Encoding Encoding

SUB Rm,Dir8

Operation: (PC) $\leftarrow (PC) + 3$
 (Rm) $\leftarrow (Rm) - (dir8)$

Flags:	CY	AC	OV	N	Z
	✓	✓	✓	✓	✓

[Encoding] 9E(s)1H

1	0	0	1	1	1	1	0	s	s	s	s	0	0	0	1	dir8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

Binary Mode **Source Mode**

Bytes: 4 3

Cycles: 1 1

Hex: [A5]Encoding Encoding

SUB WRj,Dir8

Operation: (PC) $\leftarrow (PC) + 3$
 (WRj) $\leftarrow (WRj) - (dir8)$

Flags:	CY	AC	OV	N	Z
	✓	-	✓	✓	✓

[Encoding] 9E(t)5H

1	0	0	1	1	1	1	0	t	t	t	t	0	1	0	1	dir8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

Binary Mode **Source Mode**

Bytes: 4 3

Cycles: 1 (2 for SFR) 1 (2 for SFR)

Hex: [A5]Encoding Encoding

SUB Rm,Dir16

Operation: (PC) $\leftarrow (PC) + 4$
 (Rm) $\leftarrow (Rm) - (dir16)$

Flags:	CY	AC	OV	N	Z
---------------	----	----	----	---	---

	✓	✓	✓	✓	✓	
[Encoding]	9E(s)3H					
	1	0	0	1	1	1
	1	1	1	0	s	s
	Binary Mode			Source Mode		
Bytes:	5			4		
Cycles:	1			1		
Hex:	[A5]Encoding			Encoding		

SUB WRj,Dir16

Operation:	(PC)	$\leftarrow (PC) + 4$				
	(WRj)	$\leftarrow (WRj) - (dir16)$				
Flags:	CY	AC	OV	N	Z	
	✓	-	✓	✓	✓	
[Encoding]	9E(t)7H					
	1	0	0	1	1	1
	1	1	1	0	t	t
	Binary Mode			Source Mode		
Bytes:	5			4		
Cycles:	2			2		
Hex:	[A5]Encoding			Encoding		

SUB Rm,@WRj

Operation:	(PC)	$\leftarrow (PC) + 3$				
	(Rm)	$\leftarrow (Rm) - ((WRj))$				
Flags:	CY	AC	OV	N	Z	
	✓	✓	✓	✓	✓	
[Encoding]	9E(t)9(s)0H					
	1	0	0	1	1	1
	1	1	1	0	t	t
	Binary Mode			Source Mode		
Bytes:	4			3		
Cycles:	1			1		
Hex:	[A5]Encoding			Encoding		

SUB Rm,@DRk

Operation:	(PC)	$\leftarrow (PC) + 3$				
	(Rm)	$\leftarrow (Rm) - ((DRk))$				
Flags:	CY	AC	OV	N	Z	
	✓	✓	✓	✓	✓	
[Encoding]	9E(u)B(s)0H					
	1	0	0	1	1	1
	1	1	1	0	u	u
	Binary Mode			Source Mode		
Bytes:	4			3		
Cycles:	3			3		
Hex:	[A5]Encoding			Encoding		

SUBB A,<src-byte>

Function: Subtract with borrow

Description: SUBB subtracts the indicated variable and the carry flag together from the accumulator, leaving the result in the accumulator. SUBB sets the carry (borrow) flag if borrow is needed for bit 7, and clears C otherwise. (If C was set before executing a SUBB instruction, this indicates that borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the accumulator along with the source operand). AC is set if borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6 but not into bit 7, or into bit 7 but not bit 6. When subtracting signed integers OV indicates, a negative number produced when a negative value is subtracted from a positive value or a positive result when a positive number is subtracted from a negative number. The source operand allows four addressing modes: register, direct, register-indirect, or immediate. All flags are affected.

SUBB A,Rn

Operation:	(PC)	$\leftarrow (PC) + 1$											
	(A)	$\leftarrow (A) - (C) - (Rn)$											
Flags:	CY	AC	OV	N	Z								
	✓	✓	✓	✓	✓								
[Encoding]	98H - 9FH												
	<table><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>r</td><td>r</td><td>r</td></tr></table>					1	0	0	1	1	r	r	r
1	0	0	1	1	r	r	r						
	Binary Mode		Source Mode										
Bytes:	1		2										
Cycles:	1		1										
Hex:	Encoding		[A5]Encoding										

SUBB A,Direct

Operation:	(PC)	$\leftarrow (PC) + 2$			
	(A)	$\leftarrow (A) - (C) - (\text{direct})$			
Flags:	CY	AC	OV	N	Z
	✓	✓	✓	✓	✓
[Encoding]	95H				
	1	0	0	1	0 1 0 1 direct address
	Binary Mode		Source Mode		
Bytes:	2		2		
Cycles:	1		1		
Hex:	Encoding		Encoding		

SUBB A,@Ri

Operation:	(PC)	$\leftarrow (PC) + 1$										
	(A)	$\leftarrow (A) - (C) - ((Ri))$										
Flags:	CY	AC	OV	N	Z							
	✓	✓	✓	✓	✓							
[Encoding]	96H, 97H											
	<table><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>i</td></tr></table>					1	0	0	1	0	1	1
1	0	0	1	0	1	1	i					
	Binary Mode		Source Mode									
Bytes:	1		2									
Cycles:	1		1									
Hex:	Encoding		[A5]Encoding									

SUBB A,#DATA

Operation: (PC) $\leftarrow (PC) + 2$
 (A) $\leftarrow (A) - (C) - \#data$

Flags:	CY	AC	OV	N	Z
	✓	✓	✓	✓	✓

[Encoding] 94H

1	0	0	1	0	1	0	0	immediate data
---	---	---	---	---	---	---	---	----------------

Binary Mode **Source Mode**

Bytes: 2 2

Cycles: 1 1

Hex: Encoding Encoding

SWAP A

Function: Swap nibbles within the accumulator

Description: SWAP A interchanges the low and high-order nibbles (four-bit fields) of the accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction. Only N and Z flags are affected.

Operation: (PC) $\leftarrow (PC) + 1$
 (A3-0) $\leftarrow (A7-4)$
 (A7-4) $\leftarrow (A3-0)$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] C4H

1	1	0	0	0	1	0	0
---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 1 1

Cycles: 1 1

Hex: Encoding Encoding

TRAP

Function: Executes as NOP

Operation: (PC) $\leftarrow (PC) + 1$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] B9H

1	0	1	1	1	0	0	1
---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 2 1

Cycles: 1 1

Hex: [A5]Encoding Encoding

XCH A,<byte>

Function: Exchange accumulator with byte variable

Description: XCH loads the accumulator with the contents of the indicated variable, at the same time writing the original accumulator contents to the indicated variable. The source/destination operand can use

register, direct, or register-indirect addressing. No flags are affected.

XCH A,Rn

Operation: (PC) $\leftarrow (PC) + 1$

(A) $\leftrightarrow (Rn)$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] C8H - CFH

1	1	0	0	1	r	r	r
---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 1 2

Cycles: 1 1

Hex: Encoding [A5]Encoding

XCH A,Direct

Operation: (PC) $\leftarrow (PC) + 2$

(A) $\leftrightarrow (\text{direct})$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] C5H

1	1	0	0	0	1	0	1	direct address
---	---	---	---	---	---	---	---	----------------

Binary Mode **Source Mode**

Bytes: 2 2

Cycles: 1 1

Hex: Encoding Encoding

XCH A,@Ri

Operation: (PC) $\leftarrow (PC) + 1$

(A) $\leftrightarrow ((Ri))$

Flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Encoding] C6H, C7H

1	1	0	0	0	1	1	i
---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 1 2

Cycles: 1 1

Hex: Encoding [A5]Encoding

XCHD A,@Ri

Function: Exchange digit

Description: XCHD exchanges the low-order nibble of the accumulator (bits 3-0, generally representing a hexadecimal or BCD digit), with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.

Operation: (PC) $\leftarrow (PC) + 1$

(A3-0) $\leftrightarrow ((Ri)3-0)$

Flags:	CY	AC	OV	N	Z
---------------	----	----	----	---	---

	-	-	-	-	-
[Encoding]	D6H, D7H				
	1	1	0	1	0 1 1 i
	Binary Mode		Source Mode		
Bytes:	1		2		
Cycles:	3		3		
Hex:	Encoding		[A5]Encoding		

XRL

Function: Logical Exclusive OR for variables

Description: Performs the bitwise logical Exclusive-OR operation between the specified variables, storing the results in the destination. The destination operand can be the accumulator, a register, or a direct address. The two operands allow 12 addressing mode combinations. When the destination is the accumulator or a register, the source addressing can be register, direct, register-indirect, or immediate; when the destination is a direct address, the source can be the accumulator or immediate data. Only the N and Z flags are affected.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

XRL A,Rn

Operation:	(PC)	$\leftarrow (PC) + 1$			
	(A)	$\leftarrow (A) \text{ xor } (Rn)$			
Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓
[Encoding]	68H - 6FH				
	0	1	1	0	1 r r r
	Binary Mode		Source Mode		
Bytes:	1		2		
Cycles:	1		1		
Hex:	Encoding		[A5]Encoding		

XRL A,Direct

Operation:	(PC)	$\leftarrow (PC) + 2$			
	(A)	$\leftarrow (A) \text{ xor } (\text{direct})$			
Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓
[Encoding]	65H				
	0	1	1	0	0 1 0 1 direct address
	Binary Mode		Source Mode		
Bytes:	2		3		
Cycles:	1		1		
Hex:	Encoding		[A5]Encoding		

XRL A,@Ri

Operation:	(PC)	$\leftarrow (PC) + 1$
	(A)	$\leftarrow (A) \text{ xor } ((Ri))$

Flags:

CY	AC	OV	N	Z
-	-	-	✓	✓

[Encoding] 66H, 67H

0	1	1	0	0	1	1	i
---	---	---	---	---	---	---	---

Binary Mode**Source Mode****Bytes:**

1

2

Cycles:

1

1

Hex:

Encoding

[A5]Encoding

XRL A,#DATA**Operation:**

(PC)

 $\leftarrow (PC) + 2$

(A)

 $\leftarrow (A) \text{ xor } \#data$ **Flags:**

CY	AC	OV	N	Z
-	-	-	✓	✓

[Encoding] 64H

0	1	1	0	0	1	0	1	immediate data
---	---	---	---	---	---	---	---	----------------

Binary Mode**Source Mode****Bytes:**

2

2

Cycles:

1

1

Hex:

Encoding

Encoding

XRL Direct,A**Operation:**

(PC)

 $\leftarrow (PC) + 2$

(direct)

 $\leftarrow (\text{direct}) \text{ xor } (A)$ **Flags:**

CY	AC	OV	N	Z
-	-	-	✓	✓

[Encoding] 62H

0	1	1	0	0	0	1	0	direct address
---	---	---	---	---	---	---	---	----------------

Binary Mode**Source Mode****Bytes:**

2

2

Cycles:

1

1

Hex:

Encoding

Encoding

XRL Direct,#DATA**Operation:**

(PC)

 $\leftarrow (PC) + 3$

(direct)

 $\leftarrow (\text{direct}) \text{ xor } \#data$ **Flags:**

CY	AC	OV	N	Z
-	-	-	✓	✓

[Encoding] 63H

0	1	1	0	0	0	1	1	direct address	immediate data
---	---	---	---	---	---	---	---	----------------	----------------

Binary Mode**Source Mode****Bytes:**

3

3

Cycles:

1

1

Hex:

Encoding

Encoding

XRL Rmd,Rms

Operation: (PC) $\leftarrow (PC) + 2$
 (Rmd) $\leftarrow (Rms) \text{ xor } (Rmd)$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 6CH

0	1	1	0	1	1	0	0	s	s	s	s	S	S	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 3

2

Cycles: 1

1

Hex: [A5]Encoding

Encoding

XRL WRjd,WRjs

Operation: (PC) $\leftarrow (PC) + 2$
 (WRjd) $\leftarrow (WRjs) \text{ xor } (WRjd)$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 6DH

0	1	1	0	1	1	0	1	t	t	t	t	T	T	T	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode

Source Mode

Bytes: 3

2

Cycles: 1

1

Hex: [A5]Encoding

Encoding

XRL Rm,#DATA

Operation: (PC) $\leftarrow (PC) + 3$
 (Rm) $\leftarrow (Rm) \text{ xor } \#data$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 6E(s)0H

0	1	1	0	1	1	1	0	s	s	s	s	0	0	0	0	#data
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-------

Binary Mode

Source Mode

Bytes: 4

3

Cycles: 1

1

Hex: [A5]Encoding

Encoding

XRL WRj,#DATA16

Operation: (PC) $\leftarrow (PC) + 4$
 (WRj) $\leftarrow (WRj) \text{ xor } \#data16$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 6E(t)4H

0	1	1	0	1	1	0	1	t	t	t	t	0	1	0	0	#data hi	#data lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode

Source Mode

Bytes: 5

4

Cycles: 1

1

Hex: [A5]Encoding

Encoding

XRL Rm,Dir8**Operation:** (PC) $\leftarrow (PC) + 3$ (Rm) $\leftarrow (Rm) \text{ xor } (dir8)$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 6E(s)1H

0	1	1	0	1	1	1	0	s	s	s	s	0	0	0	1	dir8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

Binary Mode**Source Mode****Bytes:** 4 3**Cycles:** 1 1**Hex:** [A5]Encoding Encoding**XRL WRj,Dir8****Operation:** (PC) $\leftarrow (PC) + 3$ (WRj) $\leftarrow (WRj) \text{ xor } (dir8)$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 6E(t)5H

0	1	1	0	1	1	0	1	t	t	t	t	0	1	0	1	dir8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

Binary Mode**Source Mode****Bytes:** 4 3**Cycles:** 1 (2 for SFR) 1 (2 for SFR)**Hex:** [A5]Encoding Encoding**XRL Rm,Dir16****Operation:** (PC) $\leftarrow (PC) + 4$ (Rm) $\leftarrow (Rm) \text{ xor } (dir16)$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 6E(s)3H

0	1	1	0	1	1	1	0	s	s	s	s	0	0	1	1	dir16 hi	dir16 lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode**Source Mode****Bytes:** 5 4**Cycles:** 2 2**Hex:** [A5]Encoding Encoding**XRL WRj,Dir16****Operation:** (PC) $\leftarrow (PC) + 4$ (WRj) $\leftarrow (WRj) \text{ xor } (dir16)$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 6E(t)7H

0	1	1	0	1	1	0	1	t	t	t	t	0	1	1	1	dir16 hi	dir16 lo
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------	----------

Binary Mode**Source Mode****Bytes:** 5 4

Cycles: 2 2
Hex: [A5]Encoding Encoding

XRL Rm, @WRj

Operation: (PC) $\leftarrow (PC) + 3$
(Rm) $\leftarrow (Rm) \text{ xor } ((WRj))$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 6E(t)9(s)0H

0	1	1	0	1	1	1	0	t	t	t	t	1	0	0	1	s	s	s	s	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 4 3
Cycles: 1 1
Hex: [A5]Encoding Encoding

XRL Rm, @DRk

Operation: (PC) $\leftarrow (PC) + 3$
(Rm) $\leftarrow (Rm) \text{ xor } ((DRk))$

Flags:	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Encoding] 6E(u)B(s)0H

0	1	1	0	1	1	1	0	u	u	u	u	1	0	1	1	s	s	s	s	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Binary Mode **Source Mode**

Bytes: 4 3
Cycles: 3 3
Hex: [A5]Encoding Encoding

附录B 电气特性

绝对最大额定值

参数	最小值	最大值	单位
存储温度	-55	+125	℃
工作温度	-40	+85	℃
工作电压	1.9	5.5	V
VDD 对地电压	-0.3	+5.5	V
I/O 口对地电压	-0.3	VDD+0.3	V

直流特性 (VSS=0V, VDD=5.0V, 测试温度=25℃)

标号	参数	范围				测试环境
		最小值	典型值	最大值	单位	
I _{PD}	掉电模式电流	-	0.6	-	uA	
I _{WKT}	掉电唤醒定时器	-	4.4	-	uA	
I _{LVD}	低压检测模块	-	430	-	uA	
I _{IDL}	空闲模式电流 (6MHz)	-	0.99	-	mA	
	空闲模式电流 (11.0592MHz)	-	1.14	-	mA	
	空闲模式电流 (24MHz)	-	1.37	-	mA	
	空闲模式电流 (内部 32KHz)	-	0.57	-	mA	
I _{NOR}	正常模式电流 (6MHz)	-	1.65	-	mA	
	正常模式电流 (11.0592MHz)	-	2.29	-	mA	
	正常模式电流 (24MHz)	-	3.49	-	mA	
	正常模式电流 (内部 32KHz)	-	0.57	-	mA	
I _{CC}	普通工作模式电流	-	4	20	mA	
V _{IL1}	输入低电平	-	-	1.32	V	打开施密特触发
		-	-	1.48	V	关闭施密特触发
V _{IH1}	输入高电平 (普通 I/O)	1.60	-	-	V	打开施密特触发
		1.54	-	-	V	关闭施密特触发
V _{IH2}	输入高电平 (复位脚)	1.60	-	1.32	V	
I _{OL1}	输出低电平的灌电流	-	20	-	mA	端口电压 0.45V
I _{OH1}	输出高电平电流 (双向模式)	200	270	-	uA	
I _{OH2}	输出高电平电流 (推挽模式)	-	20	-	mA	端口电压 2.4V
I _{IL}	逻辑 0 输入电流	-	-	50	uA	端口电压 0V
I _{TL}	逻辑 1 到 0 的转移电流	100	270	600	uA	端口电压 2.0V
R _{PU}	I/O 口上拉电阻	4.1	4.2	4.4	K Ω	

直流特性 (VSS=0V, VDD=3.3V, 测试温度=25℃)

标号	参数	范围				测试环境
		最小值	典型值	最大值	单位	
I _{PD}	掉电模式电流	-	0.4	-	uA	
I _{WKT}	掉电唤醒定时器	-	1.5	-	uA	
I _{LVD}	低压检测模块	-	364	-	uA	
I _{IDL}	空闲模式电流 (6MHz)	-	0.89	-	mA	
	空闲模式电流 (11.0592MHz)	-	1.05	-	mA	
	空闲模式电流 (24MHz)	-	1.28	-	mA	
	空闲模式电流 (内部 32KHz)	-	0.47	-	mA	
I _{NOR}	正常模式电流 (6MHz)	-	1.55	-	mA	
	正常模式电流 (11.0592MHz)	-	2.19	-	mA	
	正常模式电流 (24MHz)	-	3.38	-	mA	
	正常模式电流 (内部 32KHz)	-	0.47	-	mA	
I _{CC}	普通工作模式电流	-	4	20	mA	
V _{IL1}	输入低电平	-	-	0.99	V	打开施密特触发
		-	-	1.07	V	关闭施密特触发
V _{IH1}	输入高电平 (普通 I/O)	1.18	-	-	V	打开施密特触发
		1.09	-	-	V	关闭施密特触发
V _{IH2}	输入高电平 (复位脚)	1.18	-	0.99	V	
I _{OL1}	输出低电平的灌电流	-	20	-	mA	端口电压 0.45V
I _{OH1}	输出高电平电流 (双向模式)	200	270	-	uA	
I _{OH2}	输出高电平电流 (推挽模式)	-	20	-	mA	端口电压 2.4V
I _{IL}	逻辑 0 输入电流	-	-	50	uA	端口电压 0V
I _{TL}	逻辑 1 到 0 的转移电流	100	270	600	uA	端口电压 2.0V
R _{PU}	I/O 口上拉电阻	5.8	5.9	6.0	K Ω	

内部 IRC 温漂特性 (参考温度 25℃)

温度	最小值	范围 典型值	最大值
-40℃~85℃		-1.38%~+1.42%	
-20℃~65℃		-0.88%~+1.05%	

低压复位门槛电压 (测试温度 25℃)

级别	最小值	电压 典型值	最大值
POR		1.9V	
LVR0		2.0V	
LVR1		2.4V	
LVR2		2.7V	
LVR3		3.0V	

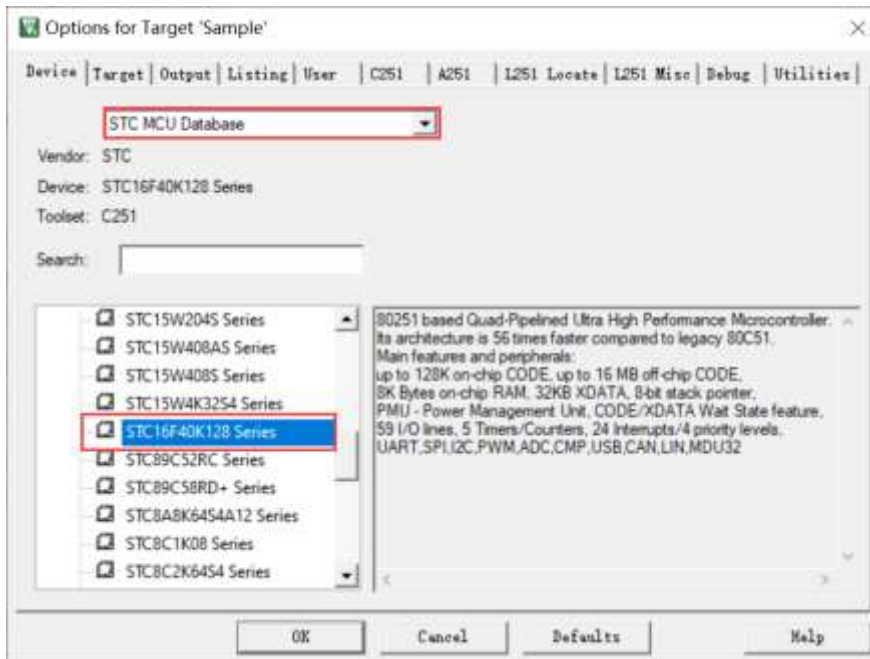
附录C 注意事项

1. 暂无 EEPROM/DATA-FLASH 功能，即相关的 IAP 寄存器不要使用；
2. P7.6/P7.7 暂无第 3 组 I2C 功能，有第 1 组、第 2 组、第 4 组 I2C 功能；
3. A 组 PWM2 暂时无法在 P5.4 口（PWM2P）进行捕获（可切换到其它口进行捕获），可对外输出；
4. A 组 PWM4 暂时无法在 P3.4(PWM4P_4)和 P6.6(PWM4P_3)口进行捕获（可切换到其它口进行捕获），可对外输出；
5. 暂无 POF 标志位；
6. C 语言注意事项：不要使用 sbit 关键字对传统 8 位 8051 以外的 16 位 8051 扩充地址空间进行位声明，后续版本会支持。
7. 第一次送样芯片可使用的程序存储器空间只有 120K，前 60K（FE0000H-FEEFFFH），后 60K（FF0000H- FFEFFFH），后 60K 的空间暂时还不能使用，后续改善。

附录D STC16 使用 Keil 开发注意事项

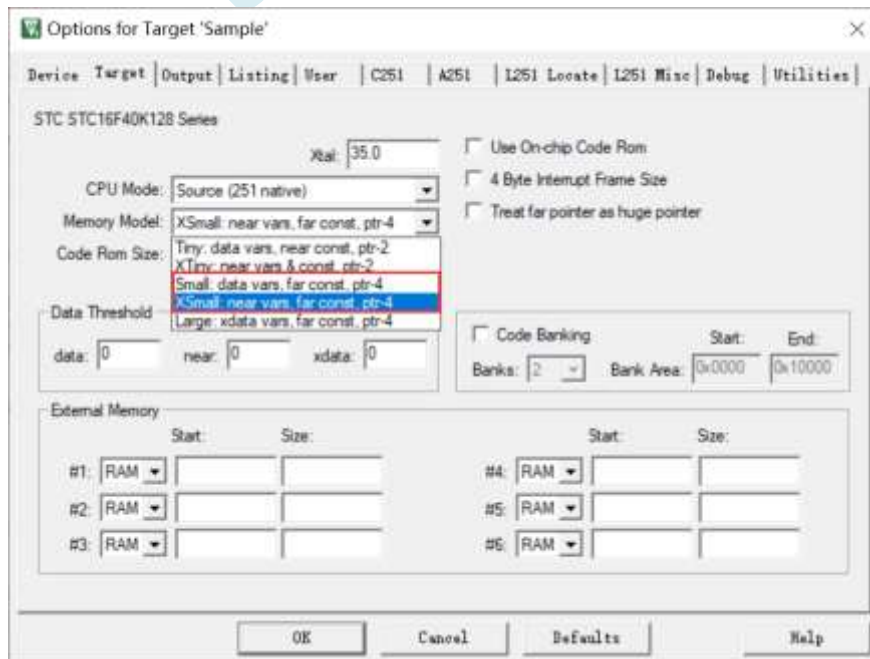
一、芯片选择

通过 STC-ISP 软件添加型号跟头文件到 Keil 中之后(详见 5.2 章节),在 Keil 的“Options for Target”→“Device”选择“STC MCU Database”栏目里 STC16 对应的型号:



二、Memory Model 设置

推荐设置为“Small”或者“XSmall”模式,这两种模式默认将变量定义在内部 RAM(edata),单时钟周期存取,访问速度快;不推荐使用“Large”模式,该模式默认将变量定义在内部扩展 RAM(xdata)里面,存取需要 3 个时钟周期,访问速度慢:



三、寄存器定义

STC 的 8 位芯片对于 16 位寄存器可以使用 int 方式进行定义跟使用, 例如:

```
#define PWMA_CCR1 (*(unsigned int volatile xdata *)0xfed5)
```

而 STC16 不能使用这种方式进行定义, 只能分开两个 8 位寄存器进行定义, 例如:

```
#define PWMA_CCR1H (*(unsigned char volatile far *)0x7fed5)
```

```
#define PWMA_CCR1L (*(unsigned char volatile far *)0x7fed6)
```

因为 8 位芯片里面这么定义, 最终访问是按字节访问。但是 STC16 如果这么定义, 会按 WORD 访问。SFR 和 XFR 都只能按字节访问。

四、变量定义

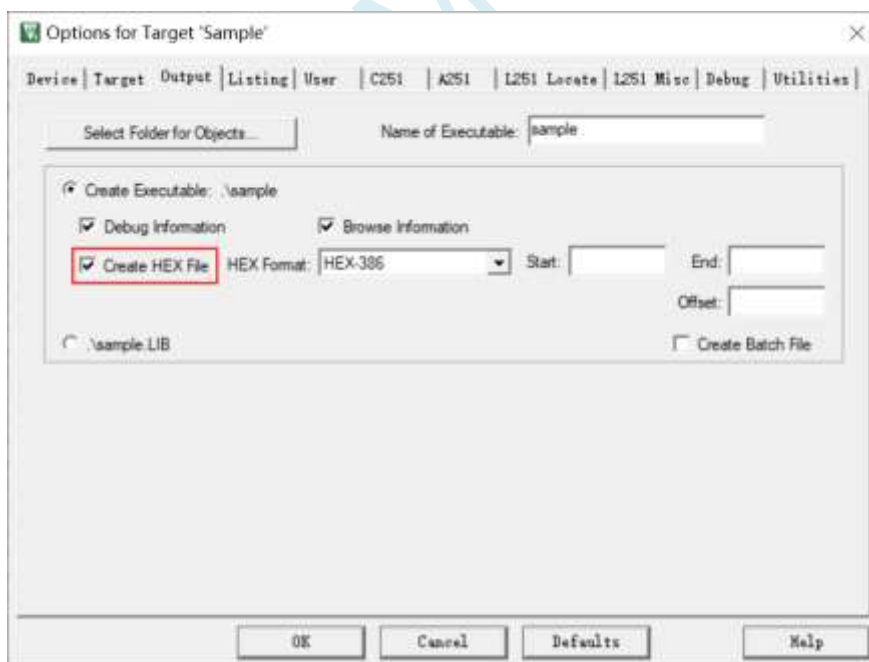
使用 MOVX A,@Ri 和 MOVX @Ri,A 时, XRAM[23:16]地址由 MXAX 设置, XRAM[15:8]由 P2 口设置, XRAM[7:0]由 Ri 设置, 由于 STC16 的 XRAM 地址范围为 0x10000~0x17FFF, 所以 STC16 使用 pdata 定义变量时, 需要设置 MXAX = 0x01 (上电默认值), P2 = 0x00~0x7F。建议使用 xdata, idata, edata, data 等其它方式进行定义。

五、xdata 使用注意

定义变量时可将单字节变量定义在 xdata 里面, 多字节 (2 字节、4 字节) 变量需要定义在 edata 里面。后续版本会取消这种使用限制。

六、生成 HEX 文件

在 Keil 的 “Options for Target” → “OutPut” 勾选 “Create HEX File” 选项, 编译后才会生成 HEX 文件:



附录E STC16 头文件

C 语言

```
#ifndef __STC16_H_
```

```
#define __STC16_H_
```

```
////////////////////////////////////
```

```
//SFR:80H-SFR:FFH
```

```
////////////////////////////////////
```

```
sfr P0 = 0x80;
```

```
sbit P00 = P0^0;
```

```
sbit P01 = P0^1;
```

```
sbit P02 = P0^2;
```

```
sbit P03 = P0^3;
```

```
sbit P04 = P0^4;
```

```
sbit P05 = P0^5;
```

```
sbit P06 = P0^6;
```

```
sbit P07 = P0^7;
```

```
sfr SP = 0x81;
```

```
sfr DPL = 0x82;
```

```
sfr DPH = 0x83;
```

```
sfr DPXL = 0x84;
```

```
sfr USBCLK = 0x85;
```

```
sfr DPS = 0x86;
```

```
sfr PCON = 0x87;
```

```
sfr TCON = 0x88;
```

```
sfr TMOD = 0x89;
```

```
sfr TL0 = 0x8a;
```

```
sfr TL1 = 0x8b;
```

```
sfr TH0 = 0x8c;
```

```
sfr TH1 = 0x8d;
```

```
sfr CKCON = 0x8e;
```

```
sfr DMAIR = 0x8f;
```

```
sfr P1 = 0x90;
```

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

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

```
sbit P12 = P1^2;
```

```
sbit P13 = P1^3;
```

```
sbit P14 = P1^4;
```

```
sbit P15 = P1^5;
```

```
sbit P16 = P1^6;
```

```
sbit P17 = P1^7;
```

sfr USBCON = 0x91;
sfr WTST = 0x92;
sfr AUXR = 0x93;
sfr AUXR2 = 0x94;
sfr INTCLKO = 0x95;
sfr USBADR = 0x96;
sfr USBDAT = 0x97;
sfr SCON = 0x98;
sfr SBUF = 0x99;
sfr P_SW1 = 0x9a;
sfr P_SW2 = 0x9b;
sfr BGTRIM = 0x9c;
sfr VRTRIM = 0x9d;
sfr LIRTRIM = 0x9e;
sfr IRTRIM = 0x9f;
sfr P2 = 0xa0;
sbit P20 = P2^0;
sbit P21 = P2^1;
sbit P22 = P2^2;
sbit P23 = P2^3;
sbit P24 = P2^4;
sbit P25 = P2^5;
sbit P26 = P2^6;
sbit P27 = P2^7;
sfr CANICR = 0xa1;
sfr CANAR = 0xa2;
sfr CANDR = 0xa3;
sfr MACDR0 = 0xa4;
sfr MACDR1 = 0xa5;
sfr MACDR2 = 0xa6;
sfr MACDR3 = 0xa7;
sfr IE = 0xa8;
sfr IRCBAND = 0xa9;
sfr WKTCL = 0xaa;
sfr WKTCH = 0xab;
sfr VOCTRL = 0xac;
sfr VOSEL = 0xad;
sfr MACADDR = 0xae;
sfr MACDATA = 0xaf;
sfr P3 = 0xb0;
sbit P30 = P3^0;
sbit P31 = P3^1;
sbit P32 = P3^2;
sbit P33 = P3^3;
sbit P34 = P3^4;
sbit P35 = P3^5;

sbit P36 = P3^6;
sbit P37 = P3^7;
sfr LINICR = 0xb1;
sfr LINAR = 0xb2;
sfr LINDR = 0xb3;
sfr CMPCR1 = 0xb4;
sfr CMPCR2 = 0xb5;
sfr IP2H = 0xb6;
sfr IPH = 0xb7;
sfr IP = 0xb8;
sfr SPSTAT = 0xb9;
sfr SPCTL = 0xba;
sfr SPDAT = 0xbb;
sfr S4CON = 0xbc;
sfr S4BUF = 0xbd;
sfr SPH = 0xbe;
sfr BUS_SPEED = 0xbf;
sfr P4 = 0xc0;
sbit P40 = P4^0;
sbit P41 = P4^1;
sbit P42 = P4^2;
sbit P43 = P4^3;
sbit P44 = P4^4;
sbit P45 = P4^5;
sbit P46 = P4^6;
sbit P47 = P4^7;
sfr DID = 0xc1;
sfr IAP_DATA = 0xc2;
sfr IAP_ADDRH = 0xc3;
sfr IAP_ADDRL = 0xc4;
sfr IAP_CMD = 0xc5;
sfr IAP_TRIG = 0xc6;
sfr IAP_CONTR = 0xc7;
sfr P5 = 0xc8;
sbit P50 = P5^0;
sbit P51 = P5^1;
sbit P52 = P5^2;
sbit P53 = P5^3;
sbit P54 = P5^4;
sbit P55 = P5^5;
sbit P56 = P5^6;
sbit P57 = P5^7;
sfr T4H = 0xc9;
sfr T4L = 0xca;
sfr T3H = 0xcb;
sfr T3L = 0xcc;

sfr T2H = 0xcd;
sfr T2L = 0xce;
sfr DEVICE = 0xcf;
sfr PSW = 0xd0;
sfr PSW1 = 0xd1;
sfr P7M1 = 0xd2;
sfr P7M0 = 0xd3;
sfr RTCCON = 0xd4;
sfr RTCSEC = 0xd5;
sfr RTCREG = 0xd6;
sfr RTCICR = 0xd7;
sfr RSTIF = 0xd8;
sfr ICECR = 0xd9;
sfr AUXINTIF = 0xda;
sfr T4T3M = 0xdb;
sfr ADC_CONTR = 0xdc;
sfr ADC_RES = 0xdd;
sfr ADC_RESL = 0xde;
sfr ADCCFG = 0xdf;
sfr ACC = 0xe0;
sfr SADDR = 0xe1;
sfr SADEN = 0xe2;
sfr S2CON = 0xe3;
sfr S2BUF = 0xe4;
sfr S3CON = 0xe5;
sfr S3BUF = 0xe6;
sfr IE2 = 0xe7;
sfr P6 = 0xe8;
sbit P60 = P6^0;
sbit P61 = P6^1;
sbit P62 = P6^2;
sbit P63 = P6^3;
sbit P64 = P6^4;
sbit P65 = P6^5;
sbit P66 = P6^6;
sbit P67 = P6^7;
sfr STATUS = 0xe9;
sfr MXAX = 0xea;
sfr TA = 0xeb;
sfr P5M1 = 0xec;
sfr P5M0 = 0xed;
sfr P6M1 = 0xee;
sfr P6M0 = 0xef;
sfr B = 0xf0;
sfr P0M1 = 0xf1;
sfr P0M0 = 0xf2;


```

sfr P1M1      = 0xf3;
sfr P1M0      = 0xf4;
sfr P2M1      = 0xf5;
sfr P2M0      = 0xf6;
sfr IAP_TPS    = 0xf7;
sfr P7        = 0xf8;
sbit P70      = P7^0;
sbit P71      = P7^1;
sbit P72      = P7^2;
sbit P73      = P7^3;
sbit P74      = P7^4;
sbit P75      = P7^5;
sbit P76      = P7^6;
sbit P77      = P7^7;
sfr P3M1      = 0xf9;
sfr P3M0      = 0xfa;
sfr P4M1      = 0xfb;
sfr P4M0      = 0xfc;
sfr WDT_CONTR = 0xfd;
sfr IP2        = 0xfe;
sfr RSTCFG    = 0xff;

```

```
/* BIT Register */
```

```
/* PSW */
```

```

sbit CY      = 0xD7;
sbit AC      = 0xD6;
sbit F0      = 0xD5;
sbit RS1     = 0xD4;
sbit RS0     = 0xD3;
sbit OV      = 0xD2;
sbit F1      = 0xD1;
sbit P       = 0xD0;

```

```
/* TCON */
```

```

sbit TF1     = 0x8F;
sbit TR1     = 0x8E;
sbit TF0     = 0x8D;
sbit TR0     = 0x8C;
sbit IE1     = 0x8B;
sbit IT1     = 0x8A;
sbit IE0     = 0x89;
sbit IT0     = 0x88;

```

```
/* IE */
```

```

sbit EA      = 0xAF;
sbit ELVD    = 0xAE;
sbit EADC    = 0xAD;

```

sbit ES = 0xAC;
sbit ETI = 0xAB;
sbit EXI = 0xAA;
sbit ET0 = 0xA9;
sbit EX0 = 0xA8;

/* IP */

sbit PS = 0xBC;
sbit PTI = 0xBB;
sbit PXI = 0xBA;
sbit PT0 = 0xB9;
sbit PX0 = 0xB8;

/* P3 */

sbit RD = 0xB7;
sbit WR = 0xB6;
sbit TI = 0xB5;
sbit T0 = 0xB4;
sbit INTI = 0xB3;
sbit INT0 = 0xB2;
sbit TXD = 0xB1;
sbit RXD = 0xB0;

/* SCON */

sbit SM0 = 0x9F;
sbit SM1 = 0x9E;
sbit SM2 = 0x9D;
sbit REN = 0x9C;
sbit TB8 = 0x9B;
sbit RB8 = 0x9A;
sbit TI = 0x99;
sbit RI = 0x98;

//////////////////////////////////////
 //7E:FE00H-7E:FEFFH
 //////////////////////////////////////

#define CKSEL (*(unsigned char volatile far *)0x7efe00)
#define CLKDIV (*(unsigned char volatile far *)0x7efe01)
#define IRC24MCR (*(unsigned char volatile far *)0x7efe02)
#define XOSCCR (*(unsigned char volatile far *)0x7efe03)
#define IRC32KCR (*(unsigned char volatile far *)0x7efe04)
#define PLLCR (*(unsigned char volatile far *)0x7efe05)
#define USBCON1 (*(unsigned char volatile far *)0x7efe06)
#define MCLKOCR (*(unsigned char volatile far *)0x7efe07)
#define IRC48MCR (*(unsigned char volatile far *)0x7efe08)
#define IRC48ATRIM (*(unsigned char volatile far *)0x7efe09)

```
#define IRC48BTRIM      (*(unsigned char volatile far *)0x7efe0a)
#define IRCDDB          (*(unsigned char volatile far *)0x7efe0b)

#define P0PU            (*(unsigned char volatile far *)0x7efe10)
#define P1PU            (*(unsigned char volatile far *)0x7efe11)
#define P2PU            (*(unsigned char volatile far *)0x7efe12)
#define P3PU            (*(unsigned char volatile far *)0x7efe13)
#define P4PU            (*(unsigned char volatile far *)0x7efe14)
#define P5PU            (*(unsigned char volatile far *)0x7efe15)
#define P6PU            (*(unsigned char volatile far *)0x7efe16)
#define P7PU            (*(unsigned char volatile far *)0x7efe17)
#define P0NCS           (*(unsigned char volatile far *)0x7efe18)
#define P1NCS           (*(unsigned char volatile far *)0x7efe19)
#define P2NCS           (*(unsigned char volatile far *)0x7efe1a)
#define P3NCS           (*(unsigned char volatile far *)0x7efe1b)
#define P4NCS           (*(unsigned char volatile far *)0x7efe1c)
#define P5NCS           (*(unsigned char volatile far *)0x7efe1d)
#define P6NCS           (*(unsigned char volatile far *)0x7efe1e)
#define P7NCS           (*(unsigned char volatile far *)0x7efe1f)
#define P0SR            (*(unsigned char volatile far *)0x7efe20)
#define P1SR            (*(unsigned char volatile far *)0x7efe21)
#define P2SR            (*(unsigned char volatile far *)0x7efe22)
#define P3SR            (*(unsigned char volatile far *)0x7efe23)
#define P4SR            (*(unsigned char volatile far *)0x7efe24)
#define P5SR            (*(unsigned char volatile far *)0x7efe25)
#define P6SR            (*(unsigned char volatile far *)0x7efe26)
#define P7SR            (*(unsigned char volatile far *)0x7efe27)
#define P0DR            (*(unsigned char volatile far *)0x7efe28)
#define P1DR            (*(unsigned char volatile far *)0x7efe29)
#define P2DR            (*(unsigned char volatile far *)0x7efe2a)
#define P3DR            (*(unsigned char volatile far *)0x7efe2b)
#define P4DR            (*(unsigned char volatile far *)0x7efe2c)
#define P5DR            (*(unsigned char volatile far *)0x7efe2d)
#define P6DR            (*(unsigned char volatile far *)0x7efe2e)
#define P7DR            (*(unsigned char volatile far *)0x7efe2f)
#define P0IE            (*(unsigned char volatile far *)0x7efe30)
#define P1IE            (*(unsigned char volatile far *)0x7efe31)

#define I2CCFG          (*(unsigned char volatile far *)0x7efe80)
#define I2CMSCR          (*(unsigned char volatile far *)0x7efe81)
#define I2CMSST          (*(unsigned char volatile far *)0x7efe82)
#define I2CSLCR          (*(unsigned char volatile far *)0x7efe83)
#define I2CSLST          (*(unsigned char volatile far *)0x7efe84)
#define I2CSLADR          (*(unsigned char volatile far *)0x7efe85)
#define I2CTXD           (*(unsigned char volatile far *)0x7efe86)
#define I2CRXD           (*(unsigned char volatile far *)0x7efe87)
```

```
#define I2CMSAUX          (*(unsigned char volatile far *)0x7efe88)

#define TIMER2PS          (*(unsigned char volatile far *)0x7efea2)
#define TIMER3PS          (*(unsigned char volatile far *)0x7efea3)
#define TIMER4PS          (*(unsigned char volatile far *)0x7efea4)

#define ADCTIM            (*(unsigned char volatile far *)0x7efeab)
#define T3T4PS            (*(unsigned char volatile far *)0x7efead)

#define PWM1_ETRPS        (*(unsigned char volatile far *)0x7efeb0)
#define PWM1_ENO          (*(unsigned char volatile far *)0x7efeb1)
#define PWM1_PS           (*(unsigned char volatile far *)0x7efeb2)
#define PWM1_IOAUX        (*(unsigned char volatile far *)0x7efeb3)
#define PWM2_ETRPS        (*(unsigned char volatile far *)0x7efeb4)
#define PWM2_ENO          (*(unsigned char volatile far *)0x7efeb5)
#define PWM2_PS           (*(unsigned char volatile far *)0x7efeb6)
#define PWM2_IOAUX        (*(unsigned char volatile far *)0x7efeb7)

#define PWMA_ETRPS        (*(unsigned char volatile far *)0x7efeb0)
#define PWMA_ENO          (*(unsigned char volatile far *)0x7efeb1)
#define PWMA_PS           (*(unsigned char volatile far *)0x7efeb2)
#define PWMA_IOAUX        (*(unsigned char volatile far *)0x7efeb3)
#define PWMB_ETRPS        (*(unsigned char volatile far *)0x7efeb4)
#define PWMB_ENO          (*(unsigned char volatile far *)0x7efeb5)
#define PWMB_PS           (*(unsigned char volatile far *)0x7efeb6)
#define PWMB_IOAUX        (*(unsigned char volatile far *)0x7efeb7)

#define PWM1_CRI          (*(unsigned char volatile far *)0x7efec0)
#define PWM1_CR2          (*(unsigned char volatile far *)0x7efec1)
#define PWM1_SMCR         (*(unsigned char volatile far *)0x7efec2)
#define PWM1_ETR          (*(unsigned char volatile far *)0x7efec3)
#define PWM1_IER          (*(unsigned char volatile far *)0x7efec4)
#define PWM1_SRI          (*(unsigned char volatile far *)0x7efec5)
#define PWM1_SR2          (*(unsigned char volatile far *)0x7efec6)
#define PWM1_EGR          (*(unsigned char volatile far *)0x7efec7)
#define PWM1_CCMR1        (*(unsigned char volatile far *)0x7efec8)
#define PWM1_CCMR2        (*(unsigned char volatile far *)0x7efec9)
#define PWM1_CCMR3        (*(unsigned char volatile far *)0x7efeca)
#define PWM1_CCMR4        (*(unsigned char volatile far *)0x7efecb)
#define PWM1_CCER1        (*(unsigned char volatile far *)0x7efecc)
#define PWM1_CCER2        (*(unsigned char volatile far *)0x7efecd)
#define PWM1_CNTRH        (*(unsigned char volatile far *)0x7efece)
#define PWM1_CNTRL        (*(unsigned char volatile far *)0x7efecf)
#define PWM1_PSCRH        (*(unsigned char volatile far *)0x7efed0)
#define PWM1_PSCRL        (*(unsigned char volatile far *)0x7efed1)
#define PWM1_ARRH         (*(unsigned char volatile far *)0x7efed2)
```

```
#define PWM1_ARRL      (*(unsigned char volatile far *)0x7efed3)
#define PWM1_RCR      (*(unsigned char volatile far *)0x7efed4)
#define PWM1_CCR1H     (*(unsigned char volatile far *)0x7efed5)
#define PWM1_CCR1L     (*(unsigned char volatile far *)0x7efed6)
#define PWM1_CCR2H     (*(unsigned char volatile far *)0x7efed7)
#define PWM1_CCR2L     (*(unsigned char volatile far *)0x7efed8)
#define PWM1_CCR3H     (*(unsigned char volatile far *)0x7efed9)
#define PWM1_CCR3L     (*(unsigned char volatile far *)0x7efeda)
#define PWM1_CCR4H     (*(unsigned char volatile far *)0x7efedb)
#define PWM1_CCR4L     (*(unsigned char volatile far *)0x7efedc)
#define PWM1_BKR      (*(unsigned char volatile far *)0x7efedd)
#define PWM1_DTR      (*(unsigned char volatile far *)0x7efede)
#define PWM1_OISR      (*(unsigned char volatile far *)0x7efedf)

#define PWM2_CR1       (*(unsigned char volatile far *)0x7efee0)
#define PWM2_CR2       (*(unsigned char volatile far *)0x7efee1)
#define PWM2_SMCR      (*(unsigned char volatile far *)0x7efee2)
#define PWM2_ETR       (*(unsigned char volatile far *)0x7efee3)
#define PWM2_IER       (*(unsigned char volatile far *)0x7efee4)
#define PWM2_SR1       (*(unsigned char volatile far *)0x7efee5)
#define PWM2_SR2       (*(unsigned char volatile far *)0x7efee6)
#define PWM2_EGR       (*(unsigned char volatile far *)0x7efee7)
#define PWM2_CCMR1     (*(unsigned char volatile far *)0x7efee8)
#define PWM2_CCMR2     (*(unsigned char volatile far *)0x7efee9)
#define PWM2_CCMR3     (*(unsigned char volatile far *)0x7efeea)
#define PWM2_CCMR4     (*(unsigned char volatile far *)0x7efeeb)
#define PWM2_CCER1     (*(unsigned char volatile far *)0x7efeec)
#define PWM2_CCER2     (*(unsigned char volatile far *)0x7efeed)
#define PWM2_CNTRH     (*(unsigned char volatile far *)0x7efeee)
#define PWM2_CNTRL     (*(unsigned char volatile far *)0x7efef)
#define PWM2_PSCRH     (*(unsigned char volatile far *)0x7efef0)
#define PWM2_PSCRL     (*(unsigned char volatile far *)0x7efef1)
#define PWM2_ARRH      (*(unsigned char volatile far *)0x7efef2)
#define PWM2_ARRL      (*(unsigned char volatile far *)0x7efef3)
#define PWM2_RCR       (*(unsigned char volatile far *)0x7efef4)
#define PWM2_CCR5H     (*(unsigned char volatile far *)0x7efef5)
#define PWM2_CCR5L     (*(unsigned char volatile far *)0x7efef6)
#define PWM2_CCR6H     (*(unsigned char volatile far *)0x7efef7)
#define PWM2_CCR6L     (*(unsigned char volatile far *)0x7efef8)
#define PWM2_CCR7H     (*(unsigned char volatile far *)0x7efef9)
#define PWM2_CCR7L     (*(unsigned char volatile far *)0x7efefa)
#define PWM2_CCR8H     (*(unsigned char volatile far *)0x7efefb)
#define PWM2_CCR8L     (*(unsigned char volatile far *)0x7efefc)
#define PWM2_BKR       (*(unsigned char volatile far *)0x7efefd)
#define PWM2_DTR       (*(unsigned char volatile far *)0x7efefe)
#define PWM2_OISR      (*(unsigned char volatile far *)0x7efeff)
```

```
#define PWMA_CRI      (*(unsigned char volatile far *)0x7efec0)
#define PWMA_CR2      (*(unsigned char volatile far *)0x7efec1)
#define PWMA_SMCR      (*(unsigned char volatile far *)0x7efec2)
#define PWMA_ETR      (*(unsigned char volatile far *)0x7efec3)
#define PWMA_IER      (*(unsigned char volatile far *)0x7efec4)
#define PWMA_SR1      (*(unsigned char volatile far *)0x7efec5)
#define PWMA_SR2      (*(unsigned char volatile far *)0x7efec6)
#define PWMA_EGR      (*(unsigned char volatile far *)0x7efec7)
#define PWMA_CCMR1     (*(unsigned char volatile far *)0x7efec8)
#define PWMA_CCMR2     (*(unsigned char volatile far *)0x7efec9)
#define PWMA_CCMR3     (*(unsigned char volatile far *)0x7efeca)
#define PWMA_CCMR4     (*(unsigned char volatile far *)0x7efecb)
#define PWMA_CCER1     (*(unsigned char volatile far *)0x7efecce)
#define PWMA_CCER2     (*(unsigned char volatile far *)0x7efecd)
#define PWMA_CNTRH     (*(unsigned char volatile far *)0x7efece)
#define PWMA_CNTRL     (*(unsigned char volatile far *)0x7efecf)
#define PWMA_PSCRH     (*(unsigned char volatile far *)0x7efed0)
#define PWMA_PSCRL     (*(unsigned char volatile far *)0x7efed1)
#define PWMA_ARRH      (*(unsigned char volatile far *)0x7efed2)
#define PWMA_ARRL      (*(unsigned char volatile far *)0x7efed3)
#define PWMA_RCR       (*(unsigned char volatile far *)0x7efed4)
#define PWMA_CCR1H     (*(unsigned char volatile far *)0x7efed5)
#define PWMA_CCR1L     (*(unsigned char volatile far *)0x7efed6)
#define PWMA_CCR2H     (*(unsigned char volatile far *)0x7efed7)
#define PWMA_CCR2L     (*(unsigned char volatile far *)0x7efed8)
#define PWMA_CCR3H     (*(unsigned char volatile far *)0x7efed9)
#define PWMA_CCR3L     (*(unsigned char volatile far *)0x7efeda)
#define PWMA_CCR4H     (*(unsigned char volatile far *)0x7efedb)
#define PWMA_CCR4L     (*(unsigned char volatile far *)0x7efedc)
#define PWMA_BKR       (*(unsigned char volatile far *)0x7efedd)
#define PWMA_DTR       (*(unsigned char volatile far *)0x7efede)
#define PWMA_OISR      (*(unsigned char volatile far *)0x7efedf)

#define PWMB_CRI      (*(unsigned char volatile far *)0x7efee0)
#define PWMB_CR2      (*(unsigned char volatile far *)0x7efee1)
#define PWMB_SMCR      (*(unsigned char volatile far *)0x7efee2)
#define PWMB_ETR      (*(unsigned char volatile far *)0x7efee3)
#define PWMB_IER      (*(unsigned char volatile far *)0x7efee4)
#define PWMB_SR1      (*(unsigned char volatile far *)0x7efee5)
#define PWMB_SR2      (*(unsigned char volatile far *)0x7efee6)
#define PWMB_EGR      (*(unsigned char volatile far *)0x7efee7)
#define PWMB_CCMR1     (*(unsigned char volatile far *)0x7efee8)
#define PWMB_CCMR2     (*(unsigned char volatile far *)0x7efee9)
#define PWMB_CCMR3     (*(unsigned char volatile far *)0x7efeea)
#define PWMB_CCMR4     (*(unsigned char volatile far *)0x7efeeb)
```

```

#define PWMB_CCER1      (*(unsigned char volatile far *)0x7efeec)
#define PWMB_CCER2      (*(unsigned char volatile far *)0x7efeed)
#define PWMB_CNTRH      (*(unsigned char volatile far *)0x7efeee)
#define PWMB_CNTRL      (*(unsigned char volatile far *)0x7efef)
#define PWMB_PSCRH      (*(unsigned char volatile far *)0x7efef0)
#define PWMB_PSCRL      (*(unsigned char volatile far *)0x7efef1)
#define PWMB_ARRH      (*(unsigned char volatile far *)0x7efef2)
#define PWMB_ARRL      (*(unsigned char volatile far *)0x7efef3)
#define PWMB_RCR      (*(unsigned char volatile far *)0x7efef4)
#define PWMB_CCR5H      (*(unsigned char volatile far *)0x7efef5)
#define PWMB_CCR5L      (*(unsigned char volatile far *)0x7efef6)
#define PWMB_CCR6H      (*(unsigned char volatile far *)0x7efef7)
#define PWMB_CCR6L      (*(unsigned char volatile far *)0x7efef8)
#define PWMB_CCR7H      (*(unsigned char volatile far *)0x7efef9)
#define PWMB_CCR7L      (*(unsigned char volatile far *)0x7efefa)
#define PWMB_CCR8H      (*(unsigned char volatile far *)0x7efefb)
#define PWMB_CCR8L      (*(unsigned char volatile far *)0x7efefc)
#define PWMB_BKR      (*(unsigned char volatile far *)0x7efefd)
#define PWMB_DTR      (*(unsigned char volatile far *)0x7efefe)
#define PWMB_OISR      (*(unsigned char volatile far *)0x7efeff)

```

```

////////////////////////////////////
//DCAN Control Register
////////////////////////////////////

```

```

#define MR      0x00
#define CMR     0x01
#define SR      0x02
#define ISR     0x03
#define IMR     0x04
#define RMC     0x05
#define BTR0    0x06
#define BTR1    0x07
#define TM0     0x06
#define TM1     0x07
#define TX_BUF0 0x08
#define TX_BUF1 0x09
#define TX_BUF2 0x0a
#define TX_BUF3 0x0b
#define RX_BUF0 0x0c
#define RX_BUF1 0x0d
#define RX_BUF2 0x0e
#define RX_BUF3 0x0f
#define ACR0    0x10
#define ACRI    0x11
#define ACR2    0x12

```

```

#define ACR3      0x13
#define AMR0      0x14
#define AMR1      0x15
#define AMR2      0x16
#define AMR3      0x17
#define ECC       0x18
#define TXERR     0x19
#define RXERR     0x1a
#define ALC       0x1b

```

```

////////////////////////////////////
//LIN Control Register
////////////////////////////////////

```

```

#define LBUF      0x00
#define LSEL      0x01
#define LID       0x02
#define LER       0x03
#define LIE       0x04
#define LSR       0x05
#define LCR       0x05
#define DLL       0x06
#define DLH       0x07
#define HDRL      0x08
#define HDRH      0x09
#define HDP       0x0A

```

```

////////////////////////////////////
//Interrupt Vector
////////////////////////////////////

```

```

#define INT0_VECTOR      0      //0003H
#define TMR0_VECTOR      1      //000BH
#define INT1_VECTOR      2      //0013H
#define TMR1_VECTOR      3      //001BH
#define UART1_VECTOR      4      //0023H
#define ADC_VECTOR       5      //002BH
#define LVD_VECTOR       6      //0033H
#define PWM1_VECTOR      7      //003BH
#define UART2_VECTOR      8      //0043H
#define SPI_VECTOR       9      //004BH
#define INT2_VECTOR     10      //0053H
#define INT3_VECTOR     11      //005BH
#define TMR2_VECTOR     12      //0063H
#define USER_VECTOR     13      //006BH
#define INT4_VECTOR     16      //0083H
#define UART3_VECTOR     17      //008BH
#define UART4_VECTOR     18      //0093H

```



```

#define TMR3_VECTOR      19      //009BH
#define TMR4_VECTOR      20      //00A3H
#define CMP_VECTOR       21      //00ABH
#define USB_VECTOR       22      //00B3H
#define PWM2_VECTOR      23      //00BBH
#define I2C_VECTOR       24      //00C3H
#define CAN_VECTOR       28      //00E3H
#define LIN_VECTOR       29      //00EBH

```

```

////////////////////////////////////

```

```

#endif

```

汇编

```

;STC16.INC

```

```

P0      DATA    080H
SP       DATA    081H
DPL      DATA    082H
DPH      DATA    083H
DPXL     DATA    084H
USBCLK   DATA    085H
DPS      DATA    086H
PCON     DATA    087H
TCON     DATA    088H
TMOD     DATA    089H
TL0      DATA    08AH
TL1      DATA    08BH
TH0      DATA    08CH
TH1      DATA    08DH
CKCON    DATA    08EH
DMAIR    DATA    08FH
P1       DATA    090H
USBCON   DATA    091H
WTST     DATA    092H
AUXR     DATA    093H
AUXR2    DATA    094H
INTCLKO  DATA    095H
USBADR   DATA    096H
USBDAT   DATA    097H
SCON     DATA    098H
SCON0    DATA    098H
SBUF     DATA    099H
SBUF0    DATA    099H
P_SW1    DATA    09AH
P_SW2    DATA    09BH
BGTRIM   DATA    09CH
VRTRIM   DATA    09DH
LITRIM   DATA    09EH
IRTRIM   DATA    09FH
P2       DATA    0A0H
CANICR   DATA    0A1H
CANAR    DATA    0A2H

```

CANDR	DATA	0A3H
MACDR0	DATA	0A4H
MACDR1	DATA	0A5H
MACDR2	DATA	0A6H
MACDR3	DATA	0A7H
IE	DATA	0A8H
IRCBAND	DATA	0A9H
WKTCL	DATA	0AAH
WKTCH	DATA	0ABH
VOCTRL	DATA	0ACH
VOSEL	DATA	0ADH
MACADDR	DATA	0AEH
MACDATA	DATA	0AFH
P3	DATA	0B0H
LINICR	DATA	0B1H
LINAR	DATA	0B2H
LINDR	DATA	0B3H
CMPCR1	DATA	0B4H
CMPCR2	DATA	0B5H
EIPH	DATA	0B6H
IPH	DATA	0B7H
IP	DATA	0B8H
SPSTAT	DATA	0B9H
SPCTL	DATA	0BAH
SPDAT	DATA	0BBH
S4CON	DATA	0BCH
S4BUF	DATA	0BDH
SPH	DATA	0BEH
BUSSPD	DATA	0BFH
P4	DATA	0C0H
DID	DATA	0C1H
IAPDAT	DATA	0C2H
IAPADRH	DATA	0C3H
IAPADRL	DATA	0C4H
IAPCMD	DATA	0C5H
IAPTRIG	DATA	0C6H
IAP_CONTR	DATA	0C7H
P5	DATA	0C8H
TH4	DATA	0C9H
TL4	DATA	0CAH
TH3	DATA	0CBH
TL3	DATA	0CCH
TH2	DATA	0CDH
TL2	DATA	0CEH
T4H	DATA	0C9H
T4L	DATA	0CAH
T3H	DATA	0CBH
T3L	DATA	0CCH
T2H	DATA	0CDH
T2L	DATA	0CEH
DEVICR	DATA	0CFH
PSW	DATA	0D0H
PSW1	DATA	0D1H
P7M1	DATA	0D2H
P7M0	DATA	0D3H
RTCCON	DATA	0D4H
RTCSEC	DATA	0D5H
RTCREG	DATA	0D6H
RTCICR	DATA	0D7H

<i>RSTIF</i>	<i>DATA</i>	<i>0D8H</i>
<i>ICECR</i>	<i>DATA</i>	<i>0D9H</i>
<i>AUXINTIF</i>	<i>DATA</i>	<i>0DAH</i>
<i>T4T3M</i>	<i>DATA</i>	<i>0DBH</i>
<i>ADC_CONTR</i>	<i>DATA</i>	<i>0DCH</i>
<i>ADC_RES</i>	<i>DATA</i>	<i>0DDH</i>
<i>ADC_RESL</i>	<i>DATA</i>	<i>0DEH</i>
<i>ADCCFG</i>	<i>DATA</i>	<i>0DFH</i>
<i>ACC</i>	<i>DATA</i>	<i>0E0H</i>
<i>SADDR</i>	<i>DATA</i>	<i>0E1H</i>
<i>SADEN</i>	<i>DATA</i>	<i>0E2H</i>
<i>S2CON</i>	<i>DATA</i>	<i>0E3H</i>
<i>S2BUF</i>	<i>DATA</i>	<i>0E4H</i>
<i>S3CON</i>	<i>DATA</i>	<i>0E5H</i>
<i>S3BUF</i>	<i>DATA</i>	<i>0E6H</i>
<i>EIE</i>	<i>DATA</i>	<i>0E7H</i>
<i>IE2</i>	<i>DATA</i>	<i>0E7H</i>
<i>P6</i>	<i>DATA</i>	<i>0E8H</i>
<i>STATUS</i>	<i>DATA</i>	<i>0E9H</i>
<i>MXAX</i>	<i>DATA</i>	<i>0EAH</i>
<i>TA</i>	<i>DATA</i>	<i>0EBH</i>
<i>P5M1</i>	<i>DATA</i>	<i>0ECH</i>
<i>P5M0</i>	<i>DATA</i>	<i>0EDH</i>
<i>P6M1</i>	<i>DATA</i>	<i>0EEH</i>
<i>P6M0</i>	<i>DATA</i>	<i>0EFH</i>
<i>B</i>	<i>DATA</i>	<i>0F0H</i>
<i>P0M1</i>	<i>DATA</i>	<i>0F1H</i>
<i>P0M0</i>	<i>DATA</i>	<i>0F2H</i>
<i>P1M1</i>	<i>DATA</i>	<i>0F3H</i>
<i>P1M0</i>	<i>DATA</i>	<i>0F4H</i>
<i>P2M1</i>	<i>DATA</i>	<i>0F5H</i>
<i>P2M0</i>	<i>DATA</i>	<i>0F6H</i>
<i>P7</i>	<i>DATA</i>	<i>0F8H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0F9H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0FAH</i>
<i>P4M1</i>	<i>DATA</i>	<i>0FBH</i>
<i>P4M0</i>	<i>DATA</i>	<i>0FCH</i>
<i>WDT_CONTR</i>	<i>DATA</i>	<i>0FDH</i>
<i>EIPL</i>	<i>DATA</i>	<i>0FEH</i>
<i>RSTCFG</i>	<i>DATA</i>	<i>0FFH</i>

<i>P07</i>	<i>BIT</i>	<i>P0.7</i>
<i>P06</i>	<i>BIT</i>	<i>P0.6</i>
<i>P05</i>	<i>BIT</i>	<i>P0.5</i>
<i>P04</i>	<i>BIT</i>	<i>P0.4</i>
<i>P03</i>	<i>BIT</i>	<i>P0.3</i>
<i>P02</i>	<i>BIT</i>	<i>P0.2</i>
<i>P01</i>	<i>BIT</i>	<i>P0.1</i>
<i>P00</i>	<i>BIT</i>	<i>P0.0</i>
<i>P17</i>	<i>BIT</i>	<i>P1.7</i>
<i>P16</i>	<i>BIT</i>	<i>P1.6</i>
<i>P15</i>	<i>BIT</i>	<i>P1.5</i>
<i>P14</i>	<i>BIT</i>	<i>P1.4</i>
<i>P13</i>	<i>BIT</i>	<i>P1.3</i>
<i>P12</i>	<i>BIT</i>	<i>P1.2</i>
<i>P11</i>	<i>BIT</i>	<i>P1.1</i>
<i>P10</i>	<i>BIT</i>	<i>P1.0</i>
<i>P27</i>	<i>BIT</i>	<i>P2.7</i>
<i>P26</i>	<i>BIT</i>	<i>P2.6</i>

<i>P25</i>	<i>BIT</i>	<i>P2.5</i>
<i>P24</i>	<i>BIT</i>	<i>P2.4</i>
<i>P23</i>	<i>BIT</i>	<i>P2.3</i>
<i>P22</i>	<i>BIT</i>	<i>P2.2</i>
<i>P21</i>	<i>BIT</i>	<i>P2.1</i>
<i>P20</i>	<i>BIT</i>	<i>P2.0</i>
<i>P37</i>	<i>BIT</i>	<i>P3.7</i>
<i>P36</i>	<i>BIT</i>	<i>P3.6</i>
<i>P35</i>	<i>BIT</i>	<i>P3.5</i>
<i>P34</i>	<i>BIT</i>	<i>P3.4</i>
<i>P33</i>	<i>BIT</i>	<i>P3.3</i>
<i>P32</i>	<i>BIT</i>	<i>P3.2</i>
<i>P31</i>	<i>BIT</i>	<i>P3.1</i>
<i>P30</i>	<i>BIT</i>	<i>P3.0</i>
<i>P47</i>	<i>BIT</i>	<i>P4.7</i>
<i>P46</i>	<i>BIT</i>	<i>P4.6</i>
<i>P45</i>	<i>BIT</i>	<i>P4.5</i>
<i>P44</i>	<i>BIT</i>	<i>P4.4</i>
<i>P43</i>	<i>BIT</i>	<i>P4.3</i>
<i>P42</i>	<i>BIT</i>	<i>P4.2</i>
<i>P41</i>	<i>BIT</i>	<i>P4.1</i>
<i>P40</i>	<i>BIT</i>	<i>P4.0</i>
<i>P57</i>	<i>BIT</i>	<i>P5.7</i>
<i>P56</i>	<i>BIT</i>	<i>P5.6</i>
<i>P55</i>	<i>BIT</i>	<i>P5.5</i>
<i>P54</i>	<i>BIT</i>	<i>P5.4</i>
<i>P53</i>	<i>BIT</i>	<i>P5.3</i>
<i>P52</i>	<i>BIT</i>	<i>P5.2</i>
<i>P51</i>	<i>BIT</i>	<i>P5.1</i>
<i>P50</i>	<i>BIT</i>	<i>P5.0</i>
<i>P67</i>	<i>BIT</i>	<i>P6.7</i>
<i>P66</i>	<i>BIT</i>	<i>P6.6</i>
<i>P65</i>	<i>BIT</i>	<i>P6.5</i>
<i>P64</i>	<i>BIT</i>	<i>P6.4</i>
<i>P63</i>	<i>BIT</i>	<i>P6.3</i>
<i>P62</i>	<i>BIT</i>	<i>P6.2</i>
<i>P61</i>	<i>BIT</i>	<i>P6.1</i>
<i>P60</i>	<i>BIT</i>	<i>P6.0</i>
<i>P77</i>	<i>BIT</i>	<i>P7.7</i>
<i>P76</i>	<i>BIT</i>	<i>P7.6</i>
<i>P75</i>	<i>BIT</i>	<i>P7.5</i>
<i>P74</i>	<i>BIT</i>	<i>P7.4</i>
<i>P73</i>	<i>BIT</i>	<i>P7.3</i>
<i>P72</i>	<i>BIT</i>	<i>P7.2</i>
<i>P71</i>	<i>BIT</i>	<i>P7.1</i>
<i>P70</i>	<i>BIT</i>	<i>P7.0</i>
<i>IT0</i>	<i>BIT</i>	<i>TCON.0</i>
<i>IE0</i>	<i>BIT</i>	<i>TCON.1</i>
<i>IT1</i>	<i>BIT</i>	<i>TCON.2</i>
<i>IE1</i>	<i>BIT</i>	<i>TCON.3</i>
<i>TR0</i>	<i>BIT</i>	<i>TCON.4</i>
<i>TF0</i>	<i>BIT</i>	<i>TCON.5</i>
<i>TR1</i>	<i>BIT</i>	<i>TCON.6</i>
<i>TF1</i>	<i>BIT</i>	<i>TCON.7</i>
<i>RI0</i>	<i>BIT</i>	<i>SCON0.0</i>
<i>TI0</i>	<i>BIT</i>	<i>SCON0.1</i>
<i>RB08</i>	<i>BIT</i>	<i>SCON0.2</i>

<i>TB08</i>	<i>BIT</i>	<i>SCON0.3</i>
<i>REN0</i>	<i>BIT</i>	<i>SCON0.4</i>
<i>SM02</i>	<i>BIT</i>	<i>SCON0.5</i>
<i>SM01</i>	<i>BIT</i>	<i>SCON0.6</i>
<i>SM00</i>	<i>BIT</i>	<i>SCON0.7</i>
<i>RI</i>	<i>BIT</i>	<i>SCON.0</i>
<i>TI</i>	<i>BIT</i>	<i>SCON.1</i>
<i>RB8</i>	<i>BIT</i>	<i>SCON.2</i>
<i>TB8</i>	<i>BIT</i>	<i>SCON.3</i>
<i>REN</i>	<i>BIT</i>	<i>SCON.4</i>
<i>SM2</i>	<i>BIT</i>	<i>SCON.5</i>
<i>SM1</i>	<i>BIT</i>	<i>SCON.6</i>
<i>SM0</i>	<i>BIT</i>	<i>SCON.7</i>

<i>EX0</i>	<i>BIT</i>	<i>IE.0</i>
<i>ET0</i>	<i>BIT</i>	<i>IE.1</i>
<i>EX1</i>	<i>BIT</i>	<i>IE.2</i>
<i>ET1</i>	<i>BIT</i>	<i>IE.3</i>
<i>ES0</i>	<i>BIT</i>	<i>IE.4</i>
<i>ES</i>	<i>BIT</i>	<i>IE.4</i>
<i>EADC</i>	<i>BIT</i>	<i>IE.5</i>
<i>ELVD</i>	<i>BIT</i>	<i>IE.6</i>
<i>EA</i>	<i>BIT</i>	<i>IE.7</i>

<i>PX0</i>	<i>BIT</i>	<i>IP.0</i>
<i>PT0</i>	<i>BIT</i>	<i>IP.1</i>
<i>PX1</i>	<i>BIT</i>	<i>IP.2</i>
<i>PT1</i>	<i>BIT</i>	<i>IP.3</i>
<i>PS0</i>	<i>BIT</i>	<i>IP.4</i>
<i>PS</i>	<i>BIT</i>	<i>IP.4</i>
<i>PADC</i>	<i>BIT</i>	<i>IP.5</i>
<i>PLVD</i>	<i>BIT</i>	<i>IP.6</i>
<i>PPWM</i>	<i>BIT</i>	<i>IP.6</i>

<i>P</i>	<i>BIT</i>	<i>PSW.0</i>
<i>F1</i>	<i>BIT</i>	<i>PSW.1</i>
<i>OV</i>	<i>BIT</i>	<i>PSW.2</i>
<i>RS0</i>	<i>BIT</i>	<i>PSW.3</i>
<i>RS1</i>	<i>BIT</i>	<i>PSW.4</i>
<i>F0</i>	<i>BIT</i>	<i>PSW.5</i>
<i>AC</i>	<i>BIT</i>	<i>PSW.6</i>
<i>CY</i>	<i>BIT</i>	<i>PSW.7</i>

<i>RXD</i>	<i>BIT</i>	<i>P3.0</i>
<i>TXD</i>	<i>BIT</i>	<i>P3.1</i>
<i>INT0</i>	<i>BIT</i>	<i>P3.2</i>
<i>INT1</i>	<i>BIT</i>	<i>P3.3</i>
<i>T0</i>	<i>BIT</i>	<i>P3.4</i>
<i>T1</i>	<i>BIT</i>	<i>P3.5</i>
<i>WR</i>	<i>BIT</i>	<i>P3.6</i>
<i>RD</i>	<i>BIT</i>	<i>P3.7</i>

<i>;ES2</i>	<i>BIT</i>	<i>EIE.0</i>
<i>;ESPI</i>	<i>BIT</i>	<i>EIE.1</i>
<i>;ET2</i>	<i>BIT</i>	<i>EIE.2</i>
<i>;ES3</i>	<i>BIT</i>	<i>EIE.3</i>
<i>;ES4</i>	<i>BIT</i>	<i>EIE.4</i>
<i>;ET3</i>	<i>BIT</i>	<i>EIE.5</i>
<i>;ET4</i>	<i>BIT</i>	<i>EIE.6</i>

CKSEL	EQU	007EFE00H
CLKDIV	EQU	007EFE01H
IRC24MCR	EQU	007EFE02H
XOSCCR	EQU	007EFE03H
IRC32KCR	EQU	007EFE04H
PLLSEL	EQU	007EFE05H
USBCON1	EQU	007EFE06H
MCLKOCR	EQU	007EFE07H
P0PU	EQU	007EFE10H
P1PU	EQU	007EFE11H
P2PU	EQU	007EFE12H
P3PU	EQU	007EFE13H
P4PU	EQU	007EFE14H
P5PU	EQU	007EFE15H
P6PU	EQU	007EFE16H
P7PU	EQU	007EFE17H
P0NCS	EQU	007EFE18H
P1NCS	EQU	007EFE19H
P2NCS	EQU	007EFE1AH
P3NCS	EQU	007EFE1BH
P4NCS	EQU	007EFE1CH
P5NCS	EQU	007EFE1DH
P6NCS	EQU	007EFE1EH
P7NCS	EQU	007EFE1FH
P0SR	EQU	007EFE20H
P1SR	EQU	007EFE21H
P2SR	EQU	007EFE22H
P3SR	EQU	007EFE23H
P4SR	EQU	007EFE24H
P5SR	EQU	007EFE25H
P6SR	EQU	007EFE26H
P7SR	EQU	007EFE27H
P0DR	EQU	007EFE28H
P1DR	EQU	007EFE29H
P2DR	EQU	007EFE2AH
P3DR	EQU	007EFE2BH
P4DR	EQU	007EFE2CH
P5DR	EQU	007EFE2DH
P6DR	EQU	007EFE2EH
P7DR	EQU	007EFE2FH
P0IE	EQU	007EFE30H
P1IE	EQU	007EFE31H
I2CCFG	EQU	007EFE80H
I2CMSCR	EQU	007EFE81H
I2CMSST	EQU	007EFE82H
I2CSLCR	EQU	007EFE83H
I2CSLST	EQU	007EFE84H
I2CSLADR	EQU	007EFE85H
I2CTXD	EQU	007EFE86H
I2CRXD	EQU	007EFE87H
I2CMSAUX	EQU	007EFE88H
TIMER2PS	EQU	007EFEA2H
TIMER3PS	EQU	007EFEA3H
TIMER4PS	EQU	007EFEA4H
ADCTIM	EQU	007EFEABH
TIMER34PIN	EQU	007EFEACH

PWMA_ETRPS	EQU	007EFEB0H
PWMA_ENO	EQU	007EFEB1H
PWMA_PS	EQU	007EFEB2H
PWMA_IOAUX	EQU	007EFEB3H
PWMB_ETRPS	EQU	007EFEB4H
PWMB_ENO	EQU	007EFEB5H
PWMB_PS	EQU	007EFEB6H
PWMB_IOAUX	EQU	007EFEB7H
PWMA_CRI	EQU	007EFEC0H
PWMA_CR2	EQU	007EFEC1H
PWMA_SMCR	EQU	007EFEC2H
PWMA_ETR	EQU	007EFEC3H
PWMA_IER	EQU	007EFEC4H
PWMA_SR1	EQU	007EFEC5H
PWMA_SR2	EQU	007EFEC6H
PWMA_EGR	EQU	007EFEC7H
PWMA_CCMR1	EQU	007EFEC8H
PWMA_CCMR2	EQU	007EFEC9H
PWMA_CCMR3	EQU	007EFECAH
PWMA_CCMR4	EQU	007EFECBH
PWMA_CCER1	EQU	007EFECCH
PWMA_CCER2	EQU	007EFECDH
PWMA_CNTRH	EQU	007EFECEH
PWMA_CNTRL	EQU	007EFECFH
PWMA_PSCRH	EQU	007EFED0H
PWMA_PSCRL	EQU	007EFED1H
PWMA_ARRH	EQU	007EFED2H
PWMA_ARRL	EQU	007EFED3H
PWMA_RCR	EQU	007EFED4H
PWMA_CCR1H	EQU	007EFED5H
PWMA_CCR1L	EQU	007EFED6H
PWMA_CCR2H	EQU	007EFED7H
PWMA_CCR2L	EQU	007EFED8H
PWMA_CCR3H	EQU	007EFED9H
PWMA_CCR3L	EQU	007EFEDAH
PWMA_CCR4H	EQU	007EFEDBH
PWMA_CCR4L	EQU	007EFEDCH
PWMA_BKR	EQU	007EFEDDH
PWMA_DTR	EQU	007EFEDFH
PWMA_OISR	EQU	007EFEDFH
PWMB_CRI	EQU	007EFEE0H
PWMB_CR2	EQU	007EFEE1H
PWMB_SMCR	EQU	007EFEE2H
PWMB_ETR	EQU	007EFEE3H
PWMB_IER	EQU	007EFEE4H
PWMB_SR1	EQU	007EFEE5H
PWMB_SR2	EQU	007EFEE6H
PWMB_EGR	EQU	007EFEE7H
PWMB_CCMR1	EQU	007EFEE8H
PWMB_CCMR2	EQU	007EFEE9H
PWMB_CCMR3	EQU	007EFEEAH
PWMB_CCMR4	EQU	007EFEEBH
PWMB_CCER1	EQU	007EFEECH
PWMB_CCER2	EQU	007EFEECH
PWMB_CNTRH	EQU	007EFEEEH
PWMB_CNTRL	EQU	007EFEEFH
PWMB_PSCRH	EQU	007EFEEFH
PWMB_PSCRL	EQU	007EFEEFH

<i>PWMB_ARRH</i>	<i>EQU</i>	<i>007EFEF2H</i>
<i>PWMB_ARRL</i>	<i>EQU</i>	<i>007EFEF3H</i>
<i>PWMB_RCR</i>	<i>EQU</i>	<i>007EFEF4H</i>
<i>PWMB_CCR5H</i>	<i>EQU</i>	<i>007EFEF5H</i>
<i>PWMB_CCR5L</i>	<i>EQU</i>	<i>007EFEF6H</i>
<i>PWMB_CCR6H</i>	<i>EQU</i>	<i>007EFEF7H</i>
<i>PWMB_CCR6L</i>	<i>EQU</i>	<i>007EFEF8H</i>
<i>PWMB_CCR7H</i>	<i>EQU</i>	<i>007EFEF9H</i>
<i>PWMB_CCR7L</i>	<i>EQU</i>	<i>007EFEFAH</i>
<i>PWMB_CCR8H</i>	<i>EQU</i>	<i>007EFEFBH</i>
<i>PWMB_CCR8L</i>	<i>EQU</i>	<i>007EFEFCH</i>
<i>PWMB_BKR</i>	<i>EQU</i>	<i>007EFEFDH</i>
<i>PWMB_DTR</i>	<i>EQU</i>	<i>007EFEFEH</i>
<i>PWMB_OISR</i>	<i>EQU</i>	<i>007EFEFFH</i>

STC MCU

附录F 更新记录

● 2021/09/07

1. 补充高级 PWM 功能脚选择寄存器 (PWMx_ETRPS) 介绍
2. 增加 U8W 直通模式介绍
3. 增加复位参考电路
4. 增加 6.5 章外部晶振及外部时钟电路
5. 修改部分寄存器地址

● 2021/07/22

1. 补充 STC16 数学处理单元介绍
2. 补充 MDU32 运算库的使用以及下载方法介绍
3. 补充 FPMU 运算库的使用以及下载方法介绍

● 2021/07/16

1. 添加 CAN 总线验收屏蔽寄存器的滤波方式对应图
2. 添加 32 位硬件乘除单元指令执行时间表
3. 修改单精度浮点运算器指令执行表位置

● 2021/02/24

1. 添加 xdata 使用注意事项

● 2021/02/05

1. 功能脚切换添加到每一个对应章节
2. 更新附录指令集内容

● 2021/01/14

1. 修改 PWM 相关的范例程序及头文件定义(XFR 地址不能用 int 进行定义)
2. 添加附录 D STC16 使用 Keil 开发注意事项

● 2021/01/06

1. 添加 21.3 LIN 总线章节范例程序

● 2020/12/30

1. 补充 11 定时器/计数器章节内容

2. 添加 13.3 比较器章节范例程序
3. 添加 15.4 ADC 章节范例程序
4. 添加 16.5 SPI 章节范例程序
5. 添加 17.4 I2C 章节范例程序
6. 补充 18 高级 PWM 章节内容及范例程序
7. 添加 19.3 USB 章节范例程序
8. 添加 20.3 CAN 总线章节范例程序
9. 添加 22.3 硬件乘除法章节范例程序
10. 添加 23.7 浮点数运算器章节范例程序

● 2020/12/17

1. 修改 XFR 寄存器地址错误
2. 补充 6.5 时钟、复位与电源管理章节范例程序
3. 补充 7 存储器章节内容
4. 补充 9 I/O 口章节内容及范例程序
5. 添加 10.5 中断系统章节范例程序
6. 添加 11.6 定时器/计数器章节范例程序
7. 添加 12.7 串口通信章节范例程序

● 2020/12/08

1. 添加 5 开发环境的建立与 ISP 下载
2. 添加 3.4 管脚切换范例程序
3. 添加 6.5 时钟、复位与电源管理章节范例程序
4. 修改浮点数运算章节加、减、乘、除指令寄存器顺序

● 2020/11/30

1. 完善指令集附录
2. 添加 LQFP48 封装图、DIP40 封装图

● 2020/11/26

1. 添加指令集附录
2. 添加 32 位硬件乘除单元, MDU32
3. 添加单精度浮点运算器, FPMU
4. 添加注意事项

● 2020/1/10

1. 完善文档

● 2020/1/9

1. 创建 STC16F 系列单片机技术参考手册文档