MEGAWIN
MAKE YOU WIN

8051 内核-单片机

MA86E/L508
说明书

北京菱电科技有限公司
Tel:010-82674978

版本: **A1.3**

特征

- 1-T 80C51 中央处理器
- **MA86E/L508 8K** 的程序存储空间
  - ISP 程序空间可以选择 0.5KB/1KB/1.5KB……
  - 灵活 IAP 空间大小
  - 密码保护程序区访问
  - Flash 写/擦 次数：2000 次@ IAP 空间 0.5KB, 1000 次@ IAP 空间 1KB, …
  - Flash 数据保留时间： 100 年 25℃
  - MA86E/L508 出厂默认空间设置
    - AP 程序空间（6KB, 0000h~17FFh）
    - IAP 数据空间（1KB, 1800h~1BFFh）
    - ISP 引导码空间（1KB, 1C00h~1FFFh）（保留在线烧录用，用户程序无法更动）
- 片内 256 字节数据存储器
- 中断控制
  - 8 个中断源, 4 个优先级
  - 二个外部中断， nINT0 和 nINT1
  - 所有的外部中断支持高/低或上升/下降沿触发
- 两个 16-位 定时/计数, Timer 0 和 Timer 1.
  - T0CKO 在 P34 和 T1CKO 在 P35
  - T0/T1 可以选择 X12 模式
  - 支持 PWM 模式
  - T0 时钟源支持可编程预分频器
- 可编程的 16 位定时/计数阵列 (PCA)及 4 通道 **PWM**
  - 自动重载的 16 位基准时间
  - 2 个比较/捕获模块和 2 个 PWM 模块
  - 捕获模式
  - 16 位软件定时模式
  - 高速输出模式
  - 可变分辨率 PWM 模式, 最多到 8-位
- 增强型 UART (S0)
  - 帧误差侦测
  - 自动地址识别
  - 在模式 0 中可选时钟极性

- 模式 4 中支持 SPI 主控模式

- 16 GPIOs 具有键盘中断功能
- 双线接口的开始/结束侦测
- 8-位 ADC
  - 转换速度可编程，最快速到 200 ksps
  - 8 通道的单端输入
- 可程序控制的看门狗时间, 时钟源来自 ILRCO
  - 通过 CPU 或上电复位一次性使能
  - 看门狗溢出会中断或复位 CPU
  - 掉电（watch mode)模式支持看门狗功能
- 实时时钟（RTC）
  - 0.5 秒 ~ 64 秒 可程序控制的中断周期
  - 21-位长度系统时间
- 蜂鸣器功能
- 28 脚的封装片最多 26 个 GPIOs
  - P3 可以设置准双向口模式，推挽输出模式，漏极开路输出模式，仅输入模式
  - P1, P2, P4.0 和 P4.1 可以设置为推挽输出模式，漏极开路输出模式
  - P4.0, P4.1 和 P3.6 公用 XTAL2, XTAL1 和 RST
- 多种省电模式：掉电模式， 空闲模式，慢频模式，副频模式，RTC 模式，watch 模式和 monitor 模式
  - 所有的中断能唤醒空闲模式
  - **7** 种中断源能唤醒掉电模式
  - 慢频模式和副频模式支持低速 MCU 运转
  - RTC 模式在掉电模式支持 RTC 复位 CPU
  - Watch 模式掉电模式支持 WDT 复位 CPU
  - Monitor 模式在掉电模式下支持 BOD0 复位 CPU (仅仅 L-系列)
  - 在掉电模式下通过 IHRCO 快速唤醒 CPU (典型值 1uS)
- 低电压检测: VDD 4.0V E-系列 和 VDD 2.6V L-系列
  - 中断 CPU 或 复位 CPU
  - 在掉电模式下唤醒 CPU (仅仅 L-系列)
- 工作电压:
  - MA86E508: 4.2V~5.5V，要求 Flash 写操作（ISP/IAP/ICP）最小电压 4.5V
  - MA86L508: 2.4V~3.6V.，要求 Flash 写操作（ISP/IAP/ICP）最小电压 2.7V

- 工作频率范围：25MHz(最大)
  - MA86E508: 0 – 12MHz @ 4.2V – 5.5V 和 0 – 25MHz @ 4.5V – 5.5V
  - MA86L508: 0 – 12MHz @ 2.4V – 3.6V 和 0 – 25MHz @ 2.7V – 3.6V
- 时钟源种类
  - 内部震荡 24MHz/22.118MHz (IHRCO)：典型值，工厂校对到±1%
  - 外部晶振模式，支持 32.768KHz 震荡器
  - 内部低频低功耗 RC 震荡(ILRCO) 64KHz
  - 外部时钟输入(ECKI) **P4.0/XTAL2**
  - IHRCO 输出 **P4.0/XTAL2**
- 工作温度:
  - 工业级(-40℃ to +85℃)*
- 封装类型:
  - SOP28: MA86E/L508AS28
  - SOP20: MA86E/L508AS20
  - SOP16: MA86E/L508AS16

*:抽样检测

目 录

特征 3
目 录 7
1. 概述 11
2. 方框图 12
3. 特殊功能寄存器 13
3.1. SFR 地址表 13
3.2. SFR 位分配表 14
3.3. 辅助 SFR 地址表 (P 页) 17
3.4. 辅助 SFR 位分配表(P 页) 18
4. 引脚结构 19
4.1. 封装指南 19
4.2. 引脚定义 20
4.3. 交错功能转换 23
5. 8051 CPU 功能描述 25
5.1. CPU 寄存器 25
5.2. CPU 时序 27
5.3. CPU 寻址模式 28
6. 存储器组织 29
6.1. 片内程序存储器 29
6.2. 片内数据存储器 30
6.3. 关于 C51 编译器的声明标识符 32
7. 数据指针寄存器(DPTR) 33
8. 系统时钟 34
8.1. 时钟结构 35
8.2. 时钟寄存器 36
8.3. 系统时钟示例代码 39
9. 看门狗定时器(WDT) 45
9.1. WDT 结构 45
9.2. WDT 在掉电模式和空闲模式期间 45
9.3. WDT 寄存器 46
9.4. WDT 硬件选项 47
9.5. WDT 示例代码 49
10. 实时时钟(RTC)/系统时间 52
10.1. RTC 结构 52
10.2. RTC 寄存器 53
10.3. RTC 示例代码 55
11. 系统复位 58
11.1. 复位源 58
11.2. 上电复位 58
11.3. 外部复位 59
11.4. 软件复位 59

11.5. 掉电检测器（Brown-Out）复位 60
11.6. WDT 复位 61
11.7. 非法地址复位 61
11.8. 复位示例代码 62
12. 电源管理 63
12.1. 电源监控检测器 63
12.2. 省电模式 64
12.2.1. 低速模式 64
12.2.2. 副频模式 64
12.2.3. RTC 模式 64
12.2.4. Watch 模式 64
12.2.5. Monitor 模式 (仅 L-系列) 64
12.2.6. 空闲模式 64
12.2.7. 掉电模式 65
12.2.8. 中断唤醒掉电模式 66
12.2.9. 复位唤醒掉电模式 66
12.2.10. KBI 键盘唤醒掉电模式 66
12.2.11. 安全并且快速从 XTAL 唤醒模式 66
12.3. 电源控制寄存器 67
12.4. 电源控制示例代码 70
13. 输入输出口配置 78
13.1. IO 结构 78
13.1.1. 端口 3 准双向口结构 78
13.1.2. 端口 3 推挽输出结构 79
13.1.3. 端口 3 仅是输入（高阻抗输入）模式 80
13.1.4. 端口 3 开漏输出结构 80
13.1.5. 通用端口漏极开路输出结构 80
13.1.6. 通用端口推挽输出结构 81
13.1.7. 通用端口输入结构 82
13.2. 输入输出口寄存器 83
13.2.1. 端口 1 寄存器 83
13.2.2. 端口 2 寄存器 84
13.2.3. 端口 3 寄存器 85
13.2.4. 端口 4 寄存器 85
13.2.5. 上拉控制寄存器 86
13.3. GPIO 示例代码 88
14. 中断 89
14.1. 中断结构 89
14.2. 中断源 91
14.3. 中断使能 92
14.4. 中断优先级 93
14.5. 中断处理 93
14.6. TI 的特别中断向量 94
14.7. nINT0/nINT1 输入源选择 95
14.8. 中断寄存器 96
14.9. 中断示例代码 102
15. 定时器/计数器 104

15.1. 定时器 0 和定时器 1 104
15.1.1. 模式 0 结构 104
15.1.2. 模式 1 结构 106
15.1.3. 模式 2 结构 107
15.1.4. 模式 3 结构 108
15.1.5. 定时器 0/1 可编程时钟输出 109
15.1.6. 定时器 0/1 寄存器 111
15.1.7. 定时器 0/1 示例代码 114
16. 串行口(UART) 121
16.1. 串行口模式 0 122
16.2. 串行口模式 1 125
16.3. 串行口模式 2 和模式 3 126
16.4. 帧错误检测 126
16.5. 多处理器通讯 127
16.6. 自动地址识别 127
16.7. 波特率设置 130
16.7.1. 模式 0 的波特率 130
16.7.2. 模式 2 的波特率 130
16.7.3. 模式 1 和 3 的波特率 130
16.8. 串行口模式 4 (SPI 主机) 133
16.9. 串行口寄存器 136
16.10. 串行口示例代码 141
17. 可编程计数器阵列(PCA) 147
17.1. PCA 概述 147
17.2. PCA 定时器/计数器 148
17.3. 比较/捕获模块 151
17.4. PCA 工作模式 155
17.4.1. 捕获模式 155
17.4.2. 16 位软件定时器模式 156
17.4.3. 高速输出模式 157
17.4.4. PWM 模式 158
17.5. PCA 示例代码 161
18. 键盘中断(KBI) 164
18.1. 键盘中断结构 164
18.2. 键盘中断寄存器 165
18.3. 键盘中断示例代码 166
19. 串行接口侦测 168
19.1. 串行接口侦测结构 168
19.2. 串行接口侦测寄存器 169
19.3. SIDF 示例代码 170
20. 蜂鸣器 192
20.1. 蜂鸣器寄存器 192
20.2. 蜂鸣器示例代码 193
21. 8-位 ADC 194
21.1. ADC 结构 194
21.2. ADC 工作 194
21.2.1. ADC 输入通道 195

21.2.2. 启动一个转换 195
21.2.3. ADC 转换时间 195
21.2.4. I/O 引脚用于 ADC 功能 195
21.2.5. 空闲和掉电模式 196
21.3. ADC 寄存器 196
21.4. ADC 示例代码 198
22. ISP 和 IAP 201
22.1. MA86E/L508 Flash 存储空间配置 201
22.2. MA86E/L508 Flash 在 ISP/IAP 的访问 202
22.2.1. ISP/IAP Flash 编程模式 202
22.2.2. ISP/IAP Flash 读取模式 204
22.3. ISP 操作 206
22.3.1. 硬件访问 ISP 206
22.3.2. 软件访问 ISP 206
22.3.3. ISP 注意事项 207
22.4. 在应用可编程 (IAP) 208
22.4.1. IAP-存储空间边界/范围 208
22.4.2. IAP-存储空间更新数据 208
22.4.3. IAP 注意事项 209
22.5. ISP/IAP 寄存器 210
22.6. ISP/IAP 示例代码 213
23. P 页 SFR 访问 220
23.1. P 页示例代码 224
24. 辅助特殊功能寄存器 228
25. 硬件选项 232
26. 应用注意事项 234
26.1. 电源电路 234
26.2. 复位电路 234
26.3. XTAL 振荡电路 235
26.4. ICP 接口电路 236
27. 电气特性 237
27.1. 绝对最大额定值 237
27.2. DC 特性 238
27.3. 外部时钟特性 242
27.4. IHRCO 特性 243
27.5. ILRCO 特性 243
27.6. Flash 特性 245
27.7. 串口时序特性 245
27.8. ADC 特性 246
28. 指令集 248
29. 封装尺寸 252
29.1. SOP-28 252
29.2. SOP-20 253
29.3. SOP-16 254
30. 版本历史 255

1. 概述

MA86E/L508是基于80C51的高效1-T结构的单芯片微处理器， 每条指令需要1~6 时钟信号 (比标准的8051快6~7 倍), 与标准8051指令集兼容。因此在与标准8051有同样的处理能力的情况下，**MA86E/L508**只需要非常低的运行速度，同时由此能很大程度的减少耗电量。

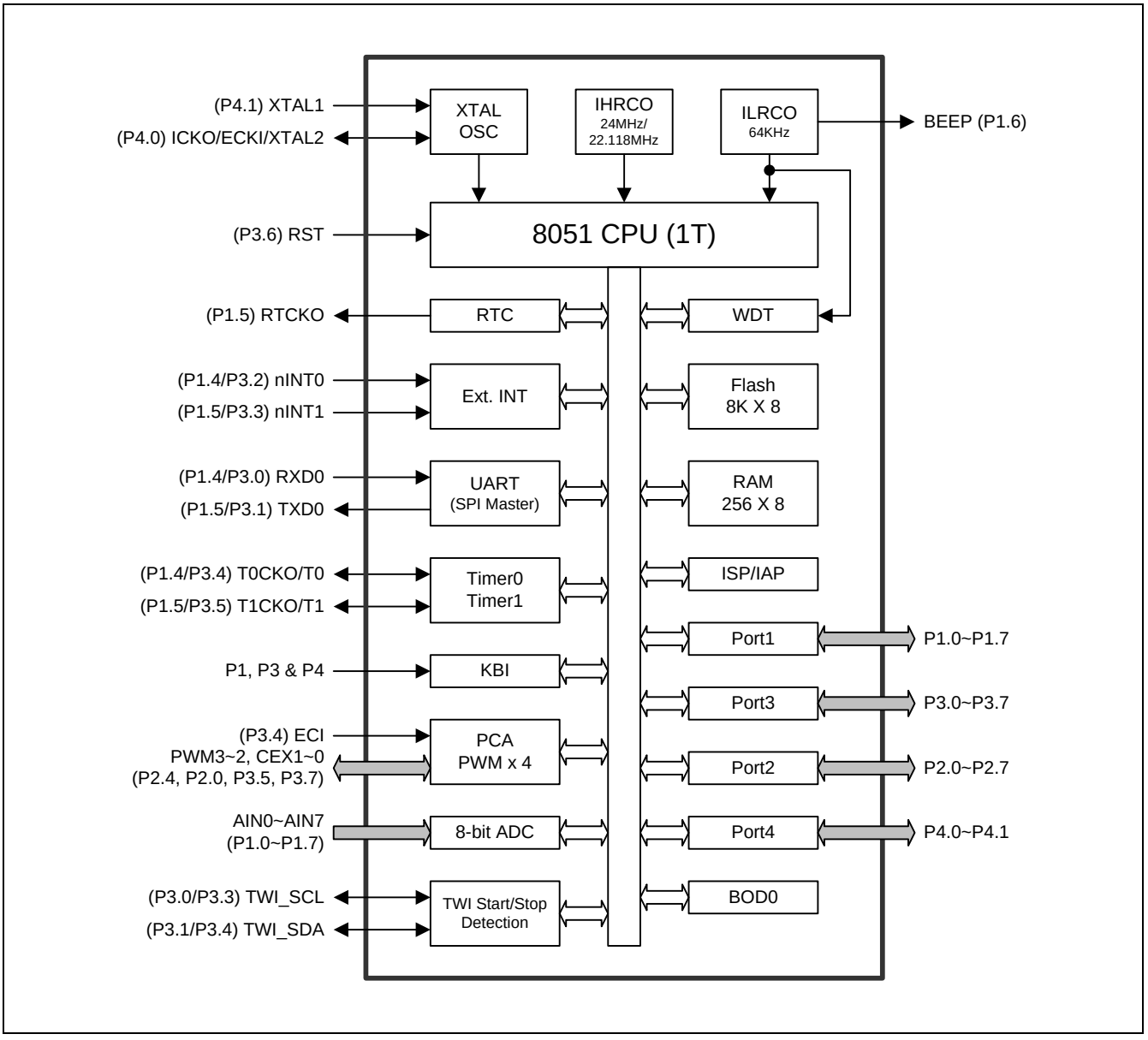
MA86E/L508有 8K字节的内置Flash存储器用于保存代码和数据。Flash存储器可以通过串行模式编程 (ICP, 在电路编程) 或者 ISP模式 (在系统编程)进行编程的能力。同时，也提供在应用编程(IAP)的能力。ISP和ICP让使用者无需从产品中取下微控制器就可以下载新的代码；IAP意味着应用程序正在运行时，微控制器能够在Flash中写入非易失数据。这些功能都由内建的电荷泵提供编程用的高压。

MA86E/L508除了80C52 MCU的标准功能（例如 256 字节的随机存储器，三个8位I/O口，二个外部中断，一个多源4级中断控制和二个16位定时/计数器)外， **MA86E/L508** 有二个额外的 I/O 口 (P4.0 和 P4.1), 键盘中断，8位ADC，一个4通道的PCA，看门狗定时器，实时时钟模式，一个低电压检测器，一个内部的晶振 (与 P4.0 和 P4.1 共用)， 一个高精度的内部振荡(IHRCO)，一个低速的内部 RC 振荡器 (ILRCO) 和一个多功能的增型的串口 (EUART) .

MA86E/L508 有多种工作模式可以减少耗电量： 空闲， 掉电模式，慢频模式， 副频模式， RTC模式，watch 模式和 monitor 模式。 在空闲模式下，CPU被冻结而外围模块和中断系统依然活动。在掉电模式下，随机存储器RAM和特殊功能寄存器SFR的值被保存，而其他所有功能被终止。最重要的是，在掉电模式下的微控制器可以被多种中断或复位唤醒。在慢频模式， 使用者可以通过8位的系统时钟分频器减慢系统速度以减少耗电量。选择副频模式系统时钟来自内部低速振荡器(ILRCO)CPU 用一个特别慢的速度在运行。 RTC 模式在所有的模式下支持实时时钟功能，watch 模式, 在掉电模式或空闲模式下保持WDT正常运行来唤醒CPU。Monitor 模式，在掉电模式检测电压，当电压特别低的时候会复位。

2. 方框图

图 2–1. 方框图


(P4.1) XTAL1
(P4.0) ICKO/ECKI/XTAL2
XTAL OSC
IHRCO 24MHz/ 22.118MHz
ILRCO 64KHz
BEEP (P1.6)
(P3.6) RST
8051 CPU (1T)
(P1.5) RTCKO
RTC
WDT
(P1.4/P3.2) nINT0
(P1.5/P3.3) nINT1
Ext. INT
Flash 8K X 8
(P1.4/P3.0) RXD0
(P1.5/P3.1) TXD0
UART (SPI Master)
RAM 256 X 8
(P1.4/P3.4) T0CKO/T0
(P1.5/P3.5) T1CKO/T1
Timer0 Timer1
ISP/IAP
Port1
P1.0~P1.7
P1, P3 & P4
KBI
Port3
P3.0~P3.7
(P3.4) ECI
PWM3~2, CEX1~0
(P2.4, P2.0, P3.5, P3.7)
PCA PWM x 4
Port2
P2.0~P2.7
AIN0~AIN7
(P1.0~P1.7)
8-bit ADC
Port4
P4.0~P4.1
(P3.0/P3.3) TWI_SCL
(P3.1/P3.4) TWI_SDA
TWI Start/Stop Detection
BOD0

3. 特殊功能寄存器

3.1. SFR 地址表

表 3–1. SFR 地址表

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8	--	CH	CCAP0H	CCAP1H	CCAP2H	CCAP3H	--	--
F0	B	PAOE	PCAPWM0	PCAPWM1	PCAPWM2	PCAPWM3	--	--
E8	P4	CL	CCAP0L	CCAP1L	CCAP2L	CCAP3L	--	--
E0	ACC	WDTCR	IFD	IFADRH	IFADRL	IFMT	SCMD	ISPCR
D8	CCON	CMOD	CCAPM0	CCAPM1	--	--	--	--
D0	PSW	--	--	--	--	--	P3KBIE	P1KBIE
C8	--	--	--	--	--	--	**CLRL**	**CHRL**
C0	--	--	--	--	ADCON0	--	ADCDH	CKCON0
B8	IP0L	SADEN	--	--	--	--	**RTCCR**	--
B0	P3	P3M0	P3M1	P4M0	PUCON0	--	**RTCTM**	IP0H
A8	IE	SADDR	--	--		EIE1	EIP1L	EIP1H
A0	P2	AUXR0	AUXR1	AUXR2	--	--	--	--
98	SCON	SBUF	--	--	--	--	--	--
90	P1	P1M0	P1AIO	--	--	P2M0	**BOREV**	PCON1
88	TCON	TMOD	TL0	TL1	TH0	TH1	SFIE	--
80	--	SP	DPL	DPH	--	--	--	PCON0
	0/8	**1/9**	**2/A**	**3/B**	**4/C**	**5/D**	**6/E**	**7/F**

3.2. SFR 位分配表

表 3–2. 位分配表

符号	描述	地址	位地址及符号								复位值
			Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0	
SP	堆栈指针	81H									00000111B
DPL	数据指针低 8 位	82H									00000000B
DPH	数据指针高 8 位	83H									00000000B
PCON0	电源控制寄存器 0	87H	SMOD1	SMOD0	GF	POF	GF1	GF0	PD	IDL	00010000B
TCON	*定时器控制寄存器*	*88H*	*TF1*	*TR1*	*TF0*	*TR0*	*IE1*	*IT1*	*IE0*	*IT0*	*00000000B*
TMOD	定时器模式寄存器	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	00000000B
TL0	定时器 0 低 8 位	8AH									00000000B
TL1	定时器 1 低 8 位	8BH									00000000B
TH0	定时器 0 高 8 位	8CH									00000000B
TH1	定时器 1 高 8 位	8DH									00000000B
SFIE	系统标志中断使能	8EH	UTIE	**SDIFIE**	--	**RTCFIE**	KBIFIE	--	BOF0IE	WDTFIE	00x00x00B
P1	*端口 1*	*90H*	*P1.7*	*P1.6*	*P1.5*	*P1.4*	*P1.3*	*P1.2*	*P1.1*	*P1.0*	*11111111B*
P1M0	P1 模式寄存器 0	91H	P1M0.7	P1M0.6	P1M0.5	P1M0.4	P1M0.3	P1M0.2	P1M0.1	P1M0.0	00000000B
P1AIO	P1 仅模拟输入寄存器	92H	P17AIO	P16AIO	P15AIO	P14AIO	P13AIO	P12AIO	P11AIO	P10AIO	00000000B
P2M0	P2 模式寄存器 0	95H	P2M0.7	P2M0.6	P2M0.5	P2M0.4	P2M0.3	P2M0.2	P2M0.1	P2M0.0	00000000B
BOREV	位序相反寄存器	96H	BOREV.7	BOREV.6	BOREV.5	BOREV.4	BOREV.3	BOREV.2	BOREV.1	BOREV.0	00000000B
PCON1	电源控制寄存器 1	97H	SWRF	EXRF	--	**RTCF**	KBIF	--	BOF0	WDTF	00x00x00B
SCON	串行口控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	Ti	RI	00000000B
SBUF	串行口缓冲寄存器	99H									xxxxxxxxB
P2	*端口 2*	*A0H*	*P2.7*	*P2.6*	*P2.5*	*P2.4*	*P2.3*	*P2.2*	*P2.1*	*P2.0*	*11111111B*
AUXR0	辅助寄存器 0	A1H	P40OC1	P40OC0	P40FD	**T0XL**	P1FS1	P1FS0	INT1H	INT0H	00000000B
AUXR1	辅助寄存器 1	A2H	P3TWI	P4S0MI	P2PCA	**XTOR**	**STAF**	**STOF**	**BPOC1**	**BPOC0**	00000000B
AUXR2	辅助寄存器 2	A3H	--	BTI	**URM0X3**	**SM3**	T1X12	T0X12	T1CKOE	T0CKOE	00000000B
IE	中断使能	A8H	EA	GF4	--	ES	ET1	EX1	ET0	EX0	00000000B
SADDR	从机地址	A9H									00000000B
EIE1	扩展中断使能 1	ADH	--	--	--	--	ESF	EPCA	EADC	--	xxxx000xB
EIP1L	扩展中断优先级 1 低	AEH	--	--	--	--	PSFL	PPCAL	PADCL	--	xxxx000xB
EIP1H	扩展中断优先级 1 高	AFH	--	--	--	--	PSFH	PPCAH	PADCH	--	xxxx000xB
P3	*端口 3*	*B0H*	*P3.7*	*P3.6*	*P3.5*	*P3.4*	*P3.3*	*P3.2*	*P3.1*	*P3.0*	*11111111B*
P3M0	P3 模式寄存器 0	B1H	P3M0.7	P3M0.6	P3M0.5	P3M0.4	P3M0.3	P3M0.2	P3M0.1	P3M0.0	00000000B
P3M1	P3 模式寄存器 1	B2H	P3M1.7	P3M1.6	P3M1.5	P3M1.4	P3M1.3	P3M1.2	P3M1.1	P3M1.0	00000000B
P4M0	P4 模式寄存器 0	B3H	--	--	--	--	--	--	P4M0.1	P4M0.0	xxxxxx00B

PUCON0	上拉控制寄存器 0	B4H	--	PU40	PU21	PU20	PU11	PU10	--	--	00000000B
RTCTM	RTC 定时器寄存器	B6H	RTCCS1	RTCCS0	RTCCT5	RTCCT4	RTCCT3	RTCCT2	RTCCT1	RTCCT0	01111111B
IP0H	中断优先级 0 高	B7H	--	--	--	PSH	PT1H	PX1H	PT0H	PX0H	00000000B
IP0L	中断优先级 0 低	*B8H*	--	--	--	*PSL*	*PT1L*	*PX1L*	*PT0L*	*PX0L*	*00000000B*
SADEN	从机地址屏蔽	B9H									00000000B
RTCCR	RTC 控制寄存器	BEH	RTCE	RTCOE	RTCRL5	RTCRL4	RTCRL3	RTCRL2	RTCRL1	RTCRL0	0x111111B
ADCON0	ADC 控制寄存器 0	C4H	ADCEN	SPEED1	SPEED0	ADCI	ADCS	CHS2	CHS1	CHS0	00000000B
ADCDH	ADC 数据高	C6H	ADCV.7	ADCV.6	ADCV.5	ADCV.4	ADCV.3	ADCV.2	ADCV.1	ADCV.0	00000000B
CKCON0	时钟控制寄存器 0	C7H	AFS	--	--	--	--	SCKS2	SCKS1	SCKS0	0xxxx001B
CLRL	PCA 基准定时器重载寄存器低	CEH	CLRL.7	CLRL.6	CLRL.5	CLRL.4	CLRL.3	CLRL.2	CLRL.1	CLRL.0	00000000B
CHRL	PCA 基准定时器重载寄存器高	CFH	CHRL.7	CHRL.6	CHRL.5	CHRL.4	CHRL.3	CHRL.2	CHRL.1	CHRL.0	00000000B
PSW	*程序状态字*	*D0H*	*CY*	*AC*	*F0*	*RS1*	*RS0*	*OV*	*F1*	*P*	*00000000B*
P3KBIE	P3 KBI 使能	D6H	P37KBIE	P36KBIE	P35KBIE	P34KBIE	**P41KBIE**	**P40KBIE**	P31KBIE	P30KBIE	00000000B
P1KBIE	P1 KBI 使能	D7H	P17KBIE	P16KBIE	P15KBIE	P14KBIE	P13KBIE	P12KBIE	P11KBIE	P10KBIE	00000000B
CCON	*PCA 控制寄存器*	*D8H*	*CF*	*CR*	--	--	--	--	*CCF1*	*CCF0*	*00xxx000B*
CMOD	PCA 模式寄存器	D9H	CIDL	--	--	--	**CPS2**	CPS1	CPS0	ECF	**0xxx0000B**
CCAPM0	PCA 模块 0 模式	DAH	--	ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0	x0000000B
CCAPM1	PCA 模块 1 模式	DBH	--	ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1	x0000000B
ACC	*累加器*	*E0H*	*ACC.7*	*ACC.6*	*ACC.5*	*ACC.4*	*ACC.3*	*ACC.2*	*ACC.1*	*ACC.0*	*00000000B*
WDTCR	看门狗控制寄存器	E1H	WREN	NSW	ENW	CLW	WIDL	PS2	PS1	PS0	00000000B xxx0xxxxB
IFD	ISP Flash 数据	E2H									11111111B
IFADRH	ISP Flash 地址高 8 位	E3H									00000000B
IFADRL	ISP Flash 地址低 8 位	E4H									00000000B
IFMT	ISP 模式表	E5H	--	--	--	--	--	MS2	MS1	MS0	xxxx0000B
SCMD	ISP 系列命令	E6H									xxxxxxxxB
ISPCR	ISP 控制寄存器	E7H	ISPEN	BS	SRST	CFAIL	--	--	--	--	0000xxxxB
P4	*端口 4*	*E8H*	--	--	--	--	--	--	*P4.1*	*P4.0*	*xxxxxx11B*
CL	PCA 基准定时器低	E9H	CL.7	CL.6	CL.5	CL.4	CL.3	CL.2	CL.1	CL.0	00000000B
CCAP0L	PCA 模块 0 捕获低 8 位	EAH									00000000B
CCAP1L	PCA 模块 1 捕获低 8 位	EBH									00000000B
CCAP2L	PCA 模块 2 捕获低 8 位	ECH									00000000B

CCAP3L	PCA 模块 3 捕获低 8 位	EDH									00000000B
B	*B 寄存器*	*F0H*	*F7H*	*F6H*	*F5H*	*F4H*	*F3H*	*F2H*	*F1H*	*F0H*	*00000000B*
PAOE		F1H									00011001B
PCAPWM0	PCA PWM0 模式	F2H	--	--	--	--	--	P0INV	EPC0H	EPC0L	xxxxx000B
PCAPWM1	PCA PWM1 模式	F3H	--	--	--	--	--	P1INV	EPC1H	EPC1L	xxxxx000B
PCAPWM2	PCA PWM2 模式	F4H	PWM2	--	--	--	--	P2INV	EPC2H	EPC2L	0xxxx000B
PCAPWM3	PCA PWM3 模式	F5H	PWM3	--	--	--	--	P3INV	EPC3H	EPC3L	0xxxx000B
CH	PCA 基准定时器高	F9H	CH.7	CH.6	CH.5	CH.4	CH.3	CH.2	CH.1	CH.0	00000000B
CCAP0H	PCA 模块 0 捕获高 8 位	FAH									00000000B
CCAP1H	PCA 模块 1 捕获高 8 位	FBH									00000000B
CCAP2H	PCA 模块 2 捕获高 8 位	FCH									00000000B
CCAP3H	PCA 模块 3 捕获高 8 位	FDH									00000000B

3.3. 辅助 SFR 地址表 (P 页)

MA86E/L508 特殊功能寄存器（SFR）有一个辅助索引 P 页，它写的方法跟标准的 8051 特殊功能寄存器的不一样。象访问 ISP/IAP 一样通过设置 IFMT 和 SCMD 来访问这个辅助的特殊功能寄存器。P 页有 256 字节空间有用到的为 5 个物理字节地址和 5 个逻辑字节地址。5 个物理字节地址包括 IAPLB, CKCON2, PCON2, SPCON0 和 DCON0. 5 个逻辑字节地址包括 PCON0, PCON1, CKCON0 , RTCCR 和 WDTCR. 访问这 5 个逻辑地址会得到相同的特殊功能（SFR）值. 更多细节请参考 “23 P 页 SFR 访问”章节。

表 3–3. 辅助 SFR 地址表 (P 页)

		0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8		--	--	--	--	--	--	--	--
F0		--	--	--	--	--	--	--	--
E8		--	--	--	--	--	--	--	--
E0		--	**WDTCR**	--	--	--	--	--	--
D8		--	--	--	--	--	--	--	--
D0		--	--	--	--	--	--	--	--
C8		--	--	--	--	--	--	--	--
C0		--	--	--	--	--	--	--	**CKCON0**
B8		--	--	--	--	--	--	**RTCCR**	--
B0		--	--	--	--	--	--	--	--
A8		--	--	--	--	--	--	--	--
A0		--	--	--	--	--	--	--	--
98		--	--	--	--	--	--	--	--
90		--	--	--	--	--	--	--	**PCON1**
88		--	--	--	--	--	--	--	--
80		--	--	--	--	--	--	--	**PCON0**
78		--	--	--	--	--	--	--	--
70		--	--	--	--	--	--	--	--
68		--	--	--	--	--	--	--	--
60		--	--	--	--	--	--	--	--
58		--	--	--	--	--	--	--	--
50		--	--	--	--	--	--	--	--
48		**SPCON0**	--	--	--	**DCON0**	--	--	--
40		**CKCON2**	--	--	--	**PCON2**	--	--	--
38		--	--	--	--	--	--	--	--
30		--	--	--	--	--	--	--	--
28		--	--	--	--	--	--	--	--
20		--	--	--	--	--	--	--	--
18		--	--	--	--	--	--	--	--
10		--	--	--	--	--	--	--	--
08		--	--	--	--	--	--	--	--
00		--	--	--	**IAPLB**	--	--	--	--
		0/8	**1/9**	**2/A**	**3/B**	**4/C**	**5/D**	**6/E**	**7/F**

3.4. 辅助 SFR 位分配表(P 页)

表 3–4. 辅助 SFR 位分配表(P 页)

符号	描述	地址	位地址及符号								复位值
			Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0	
物理字节											
IAPLB	IAP 低边界	03H	IAPLB6	IAPLB5	IAPLB4	IAPLB3	IAPLB2	IAPLB1	IAPLB0	--	00011000B
CKCON2	时钟控制 2	40H	XTGS1	XTGS0	XTALE	IHRCOE	0	0	OSCS1	OSCS0	11010000B
PCON2	电源控制 2	44H	0	AWBOD0	0	0	0	0	BO0RE	**1**	00000001B
SPCON0	SFR 页控制 0	48H	RTCCTL	0	0	WRCTL	0	CKCTL0	PWCTL1	PWCTL0	00000000B
DCON0	器件控制 0	4CH	HSE	IAPO	0	0	0	0	RSTIO	0	10000010B
逻辑字节											
PCON0	电源控制 0	87H	SMOD1	SMOD0	GF	POF	GF1	GF0	PD	IDL	00010000B
PCON1	电源控制 1	97H	SWRF	EXRF	--	**RTCF**	KBIF	--	BOF0	WDTF	00x00x00B
RTCCR	RTC 控制寄存器	BEH	RTCE	RTCOE	RTCRL5	RTCRL4	RTCRL3	RTCRL2	RTCRL1	RTCRL0	0x111111B
CKCON0	时钟控制 0	C7H	AFS	--	--	--	--	SCKS2	SCKS1	SCKS0	0xxxx001B
WDTCR	看门狗控制寄存器	E1H	WREN	NSW	ENW	CLW	WIDL	PS2	PS1	PS0	00000000B xxx0xxxxB

写入 P 页 SFR 例程：

```
IFADRH = 0x00;
ISPCR = ISPEN;                  //使能 IAP/ISP 功能
IFMT = MS2;                     // P 页写，IFMT =0x04
IFADRL = SPCON0;                //相对应的 P 页 SFR 设置 P 页 SFR 地址
IFD |= CKCTL0;                  // 设置 CKCTL0
SCMD = 0x46;                    //
SCMD = 0xB9;                    //
IFMT = Flash_Standby;           // IAP/ISP 备用模式，IFMT =0x00
ISPCR &= ~ISPEN;
```

4. 引脚结构

4.1. 封装指南

图 4–1. MA86E/L508AS28 顶视图

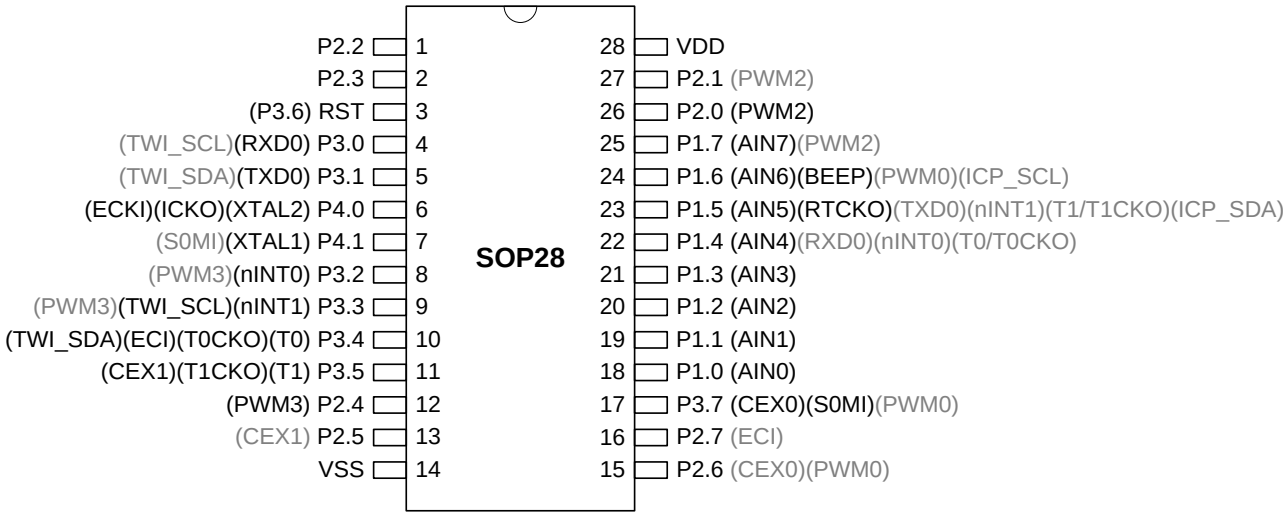


图 4–2. MA86E/L508AS20 顶视图

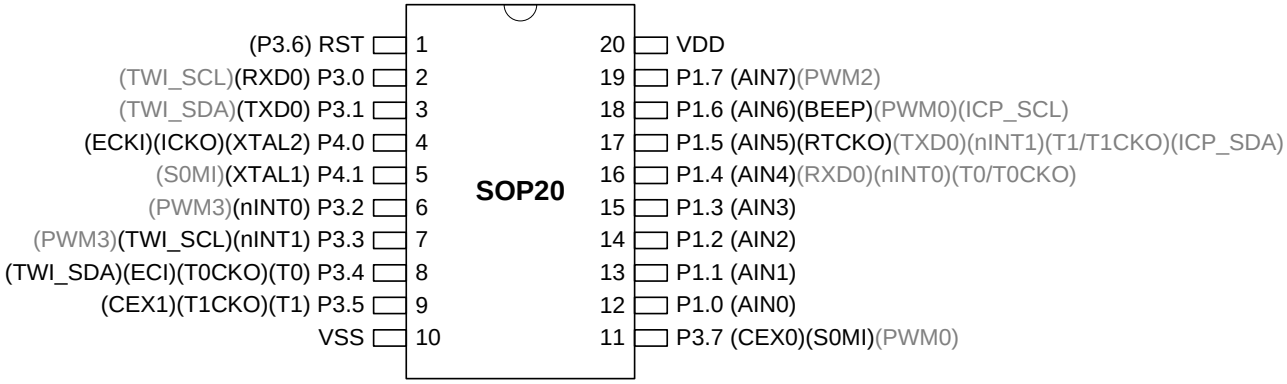
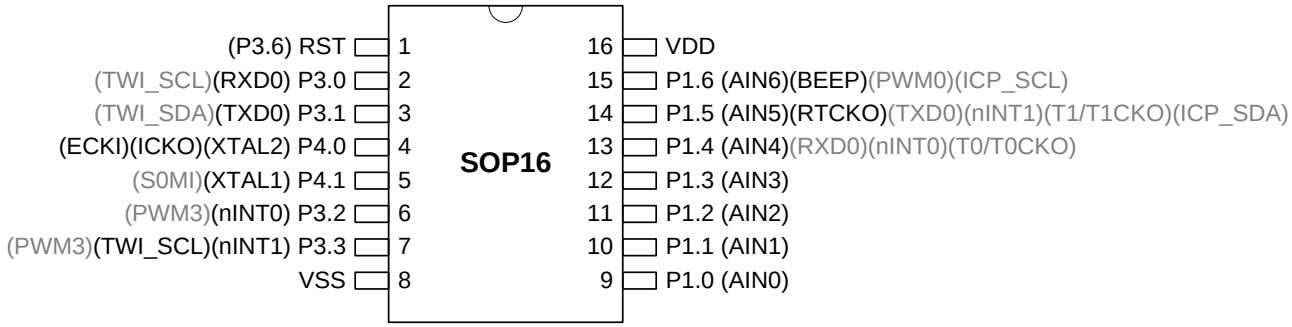


图 4–3. MA86E/L508AS16 顶视图



4.2. 引脚定义

表 4–1. 引脚定义

助记符	引脚号			I/O 类型	描述
	28-Pin SOP	20-Pin SOP	16-Pin SOP		
P1.0 (AIN0)	18	12	9	I/O	* 端口 P1.0 * AIN0: ADC 通道 0 模拟输入
P1.1 (AIN1)	19	13	10	I/O	*端口 P1.1. * AIN1: ADC 通道 1 模拟输入
P1.2 (AIN2)	20	14	11	I/O	*端口 P1.2. * AIN2: ADC 通道 2 模拟输入
P1.3 (AIN3)	21	15	12	I/O	*端口 P1.3. * AIN3: ADC 通道 3 模拟输入
P1.4 (AIN4) (RXD0) (nINT0) (T0/T0CKO)	22	16	13	I/O	*端口 P1.4. * AIN4: ADC 通道 4 模拟输入 * RXD0: UART0 RXD0 的交错功能 * nINT0: nINT0 输入的交错功能 * T0/T0CKO: T0 输入和 T0CKO 输出的交错功能
P1.5 (AIN5) (RTCKO) (TXD3) (nINT1) (T1/T1CKO) **(ICP_SDA)**	23	17	14	I/O	*端口 P1.5. * AIN5: ADC 通道 5 模拟输入 * **RTCKO**: RTC 时钟输入 * TXD3: UART0 TXD0 的交错功能 * nINT1: nINT1 输入的交错功能 * T1/T1CKO: T1 输入和 T1CKO 输出的交错功能 * ICP_SDA: ICP 接口的串行数据
P1.6 (AIN6) (BEEP) (PWM0) **(ICP_SCL)**	24	18	15	I/O	*端口 P1.6. * AIN6: ADC 通道 6 模拟输入 * **BEEP**: 蜂鸣器输出 * PWM0: CEX0 PWM 输出的第二输出 * ICP_SCL: ICP 接口的串行时钟
P1.7 (AIN7) (PWM2)	25	19	--	I/O	*端口 P1.7. * AIN7: ADC 通道 7 模拟输入 * PWM2: PWM2 输出的第二输出
P2.0 (PWM2)	26	--		I/O	*端口 P2.0. * PWM2: PCA 模块 2 的 PWM2 输出.
P2.1 (PWM2)	27	--		I/O	*端口 P2.1. * PWM2: PWM2 输出的第三输出

P2.2	1	--		I/O	*端口 P2.2.
P2.3	2	--		I/O	*端口 P2.3.
P2.4 (PWM3)	12	--		I/O	*端口 P2.4. * PWM3: PCA 模块 3 的 PWM 输出.
P2.5 (CEX1)	13	--		I/O	*端口 P2.5. * CEX1: PCA 模块 1 的 CEX1 I/O 的交错功能
P2.6 (CEX0) (PWM0)	15	--		I/O	*端口 P2.6. * CEX0: PCA 模块 0 的 CEX0 I/O 的交错功能 * PWM0: CEX0 PWM 输出的第二输出
P2.7 (ECI)	16	--		I/O	*端口 P2.7. * ECI: PCA 的 ECI 输入的交错功能
P3.0 (RXD0) (TWI_SCL)	4	2	2	I/O	*端口 P3.0. * RXD0: UART0 串行输入口 * TWI_SCL: TWI SCL 输入的交错功能
P3.1 (TXD0) (TWI_SDA)	5	3	3	I/O	*端口 P3.1. * TXD0: UART0 串行输出口 * TWI_SDA: TWI SDA 输入的交错功能
P3.2 (nINT0) (PWM3)	8	6	6	I/O	*端口 P3.2. * nINT0: 外部中断 0 输入 * PWM3: PWM3 输出的第二输出
P3.3 (nINT1) (TWI_SCL) (PWM3)	9	7	7	I/O	*端口 P3.3. * nINT1: 外部中断 1 输入 * TWI_SCL: TWI 启动/停止侦测的 SCL 输入 * PWM3: PWM3 输出的第三输出
P3.4 (T0) (T0CKO) (ECI) (TWI_SDA)	10	8	--	I/O	*端口 P3.4. * T0: 定时器/计数器 0 外部输入 * T0CKO: 定时器 0 的可编程时钟输出 * ECI: PCA 外部时钟输入 * TWI_SDA: TWI 启动/停止侦测的 SDA 输入
P3.5 (T1) (T1CKO) (CEX1)	11	9	--	I/O	*端口 P3.5. * T1: 定时器/计数器 1 外部输入 * T1CKO: 定时器 1 的可编程时钟输出 * CEX1: PCA 模块 1 的外部 I/O.
P3.7 (CEX0) (S0MI)	17	11	--	I/O	*端口 P3.7 * CEX0: PCA 模块 0 的外部 I/O. * S0MI: 在 UART0 上的 SPI 主机输入

(PWM0)					* PWM0: CEX0 PWM 输出的第二输出
P4.0 (XTAL2) (ECKI) (ICKO)	6	4	4	I/O O I O	*端口 P4.0. * XTAL2: 片上晶振振荡电路的输出 * ECKI: 外部时钟输入模式，这是时钟输入引脚 * ICKO: IHRCO 时钟输出
P4.1 (XTAL1) (S0MI)	7	5	5	I/O I I/O	*端口 P4.1. * XTAL1: 片上晶振振荡电路的输入 * S0MI: S0MI 输入的交错功能
RST (P3.6)	3	1	1	I I/O	* RST: 外部复位输入，高电平有效 *端口 P3.6. RST 的交错功能作为 P3.6
VDD	28	20	16	P	电源输入。E 型是 5V， L 型是 3.3V
VSS	14	10	8	G	地, 0V

4.3. 交错功能转换

许多 I/O 口，除了有普通的 I/O 功能外，还有内部周边设备的其它功能，如作为 UART, Timer 0, Timer1, nINT0, nINT1, S0MI 和 TWSI 的侦测功能, 端口 1, 端口 2 和端口 3 的交错功能为默认状态. 但是, 使用者可以设置 AUXR1 的位 P3TWI, P4S0MI 和 P2PCA 来选择其对应的功能， 设置寄存器 AUXR0 的位 P1FS1~0 切换到其它功能。这是特别有用的软件设计 。

AUXR0: 辅助寄存器 0

SFR 页 = 普通

SFR 地址 = 0xA1 复位值 = 0000-0000

7	6	5	4	3	2	1	0
P40OC1	P40OC0	P40FD	T0XL	P1FS1	P1FS0	INT1H	INT0H
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~6: P4.0 功能配置控制位 1 和 0. 这两位只有当 IHRCO 作为系统时钟输入时才有实际作用. 在晶振模式, XTAL2 和 XTAL1 互为交错功能 P4.0 和 P4.1. 在外部时钟输入模式, P4.0 专用于时钟输入. 在 IHRCO 工作期间, P4.0 可提供以下选择 GPIO 或 时钟源产生器. 当 P40OC[1:0] 索引到非 P4.0 GPIO 功能, P4.0 将驱动 IHRCO 输出为其它设备提供时钟源。

P40OC[1:0]	P4.0 功能	I/O 模式
00	P4.0	By P4M0.0
01	IHRCO	By P4M0.0
10	IHRCO/2	By P4M0.0
11	IHRCO/4	By P4M0.0

当 P4.0 作为时钟输出功能时, 相对应的设置 P4M0.0 为 “1” 它能选择 P4.0 作为推挽输出模式

Bit 3~2: P1.4 和 P1.5 交错功能选择.

P1FS[1:0]	P1.4	P1.5
00	P1.4	P1.5
01	RXD0 的输入	TXD0 的输出
10	nINT0 的输入	nINT1 的输入
11	T0 的输入或 T0CKO 的输出	T1 的输入或 T1CKO 的输出

AUXR1: 辅助寄存器 1

SFR 页　　=普通

SFR 地址　　= 0xA2　　复位值= 0000-0000

7	6	5	4	3	2	1	0
P3TWI	P4S0MI	P2PCA	XTOR	STAF	STOF	BPOC1	BPOC1
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: P3TWI, TWI 接口在 P3.0~P3.1.

0: 禁止 TWI 功能转移到 P30 & P31.

1: 设置 TWI 功能在 P3 口依照以下定义.

'TWI_SCL' 功能从 P3.3 转移到 P3.0.

'TWI_SDA' 功能从 P3.4 转移到 P3.1.

Bit 6: P4S0MI, S0MI 串口 SPI 主控模式输入脚 P4.1 控制位.

0: 禁止 S0MI 功能转移到 P4.

1: 设置 S0MI 功能在 P4.1 依照以下设置.

'S0MI' 在 P3.7 功能转移到 P4.1.

Bit 5: P2PCA, 所有的 PCA 信号 Port 2 控制位.

0: 禁止 PCA 功能转移到 P2.

1: 设置 PCA 在 Port 2 依照以下定义.

'ECI' 功能从 P3.4 转移到 P2.7.

'CEX0' 功能从 P3.7 转移到 P2.6.

'CEX1' 功能从 P3.5 转移到 P2.5.

'CEX2' 功能从 P2.0 保持在 P2.0.

'CEX3' 功能从 P2.4 保持在 P2.4.

5. 8051 CPU 功能描述

5.1. CPU 寄存器

PSW: **程序状态字**

SFR 页　　= 普通

SFR 地址　　= 0xD0　　复位值 = 0000-0000

7	6	5	4	3	2	1	0
CY	AC	F0	RS1	RS0	OV	F1	P
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

CY: 进位标志

AC: 辅助进位标志

F0: 用户可设定的标志位 0

RS1: 寄存器组选择位 1

RS0: 寄存器组选择位 0

OV: 溢出标志

F1: 用户可设定的标志位 1

P: 奇偶标志

程序状态字 (PSW) 包含反映 CPU 当前状态的几个状态位。PSW 属于特殊功能寄存器 SFR 区，包含进位标志，辅助进位标志 (应用于 BCD 操作)， 两个寄存器组选择位，溢出标志， 奇偶标志和两个用户可设定的标志位。

进位标志，不仅有算术运算的进位功能，也充当许多布尔运算的“累加器”。

RS0 和 RS1 被用来选择 4 组中的任意一组寄存器组，详见内部 RAM 章节 6.2。

奇偶位反映 1S 内累加器数字和的状况， 1S 内累加器中数字和是奇数则 P=1 否则 P=0

SP: **堆栈**

SFR 页　　=普通

SFR 地址　　= 0x81　　复位值= 0000-0111

7	6	5	4	3	2	1	0
SP.7	SP.6	SP.5	SP.4	SP.3	SP.2	SP.1	SP.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

堆栈指针保持栈顶位置，每执行一个 PUSH 指令，SP 会自动增加，SP 缺损值为 0X07H。

***DPL:* 数据指针低字节**

SFR 页 =普通

SFR 地址 = 0x82 复位值= 0000-0000

7	6	5	4	3	2	1	0
DPL.7	DPL.6	DPL.6	DPL.4	DPL.3	DPL.2	DPL.1	DPL.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

DPL 是 DPTR 的低字节，DPTR 用来间接访问 XRAM 和程序空间。

***DPH:* 数据指针高字节**

SFR 页 =普通

SFR 地址 = 0x83 复位值= 0000-0000

7	6	5	4	3	2	1	0
DPH.7	DPH.6	DPH.5	DPH.4	DPH.3	DPH.2	DPH.1	DPH.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

DPH 是 16 位 DPTR 的高字节，DPTR 用来间接访问 XRAM 和程序空间。

ACC: 累加器

SFR 页 =普通

SFR 地址 = 0xE0 复位值= 0000-0000

7	6	5	4	3	2	1	0
ACC.7	ACC.6	ACC.5	ACC.4	ACC.3	ACC.2	ACC.1	ACC.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

算术运算的累加器。

B: B 寄存器

SFR 页 =普通

SFR 地址 = 0xF0 复位值= 0000-0000

7	6	5	4	3	2	1	0
B.7	B.6	B.5	B.4	B.3	B.2	B.1	B.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

另一个算术运算的累加器。

5.2. CPU 时序

MA86E/L508是基于80C51的高效1-T结构的单芯片微处理器，与8051指令集兼容，每条指令需要1~6个时钟信号(比标准8051快6~7倍)。使用流线型结构同标准的8051结构比较大大增加了指令完成的速度 ，指令的时序也和标准的8051不同。

多数 8051 执行指令，一个区别是建立在机器周期和时钟周期之间，机器周期来自 2 到 12 个时钟周期长度。然而，1-T 结构的 80C51 执行指令是基于单独的时钟周期时序。所有指令时序被指定在时钟周期期间。关于 1T-80C51 指令更详细的说明，请参考“28 章指令集”，这里有每一条指令的助记符、字节数、时钟周期数。

5.3. CPU 寻址模式

直接寻址*(DIR)*

直接寻址时操作数用指令中一个 8 位地址的区域表示，只有内部数据存储器和特殊功能寄存器可以直接寻址。

间接寻址*(IND)*

间接寻址时指令用一个包含操作数地址的寄存器表示，内部和外部存储器均可间接寻址。

8 位地址的地址寄存器可以是选中区的 R0 或 R1 或堆栈指针，16 位地址的地址寄存器只能是 16 位的“数据指针”寄存器，DPTR。

寄存器操作（寻址）*(REG)*

包含从 R0 到 R7 的寄存器区可以被某些指令存取，这些指令的操作码中用 3 位寄存器说明。存取寄存器的指令有更高的代码效率，因为这种模式减少了一个地址字节。当指令被执行时，其中被选取的区一个 8 位寄存器被存取。执行时，用 PSW 寄存器中两位区选择位来选择四分之一区。

特殊寄存器寻址（寄存器间接寻址）

一些指令具有一个特定的寄存器，例如，一些指令常用于累加器，或数据指针等等，所以没有需要指向它的地址字节。操作码本身就行了。有关累加器的指令 A 就是累加器的特殊操作码。

立即寻址*(IMM)*

常量的数值可以在程序存储器中跟随操作码。

索引寻址

索引寻址只能访问程序存储器，且只读。这种寻址模式用查表法读取程序存储器。一个16位基址寄存器（数据指针DPTR或程序计数器PC）指向表的基地址，累加器提供偏移量。程序存储器中表项目地址由基地址加上累加器数据后形成。另一种索引寻址方式是利用“case jump”指令。跳转指令中的目标地址是基地址加上累加器数据后的值。

6. 存储器组织

像所有的 80C51 一样，**MA86E/L508** 的程序存储器和数据存储器的地址空间是分开的，这样 8 位微处理器可以通过一个 8 位的地址快速而有效的访问数据存储器。

程序存储器(ROM)只能读取，不能写入。最大可以达到 8K 字节。在 **MA86E/L508** 中，所有的程序存储器都是片上 Flash 存储器。因为没有设计外部程序使能 (/EA)和编程使能 (/PSEN) 信号，所以不允许外接程序存储器。

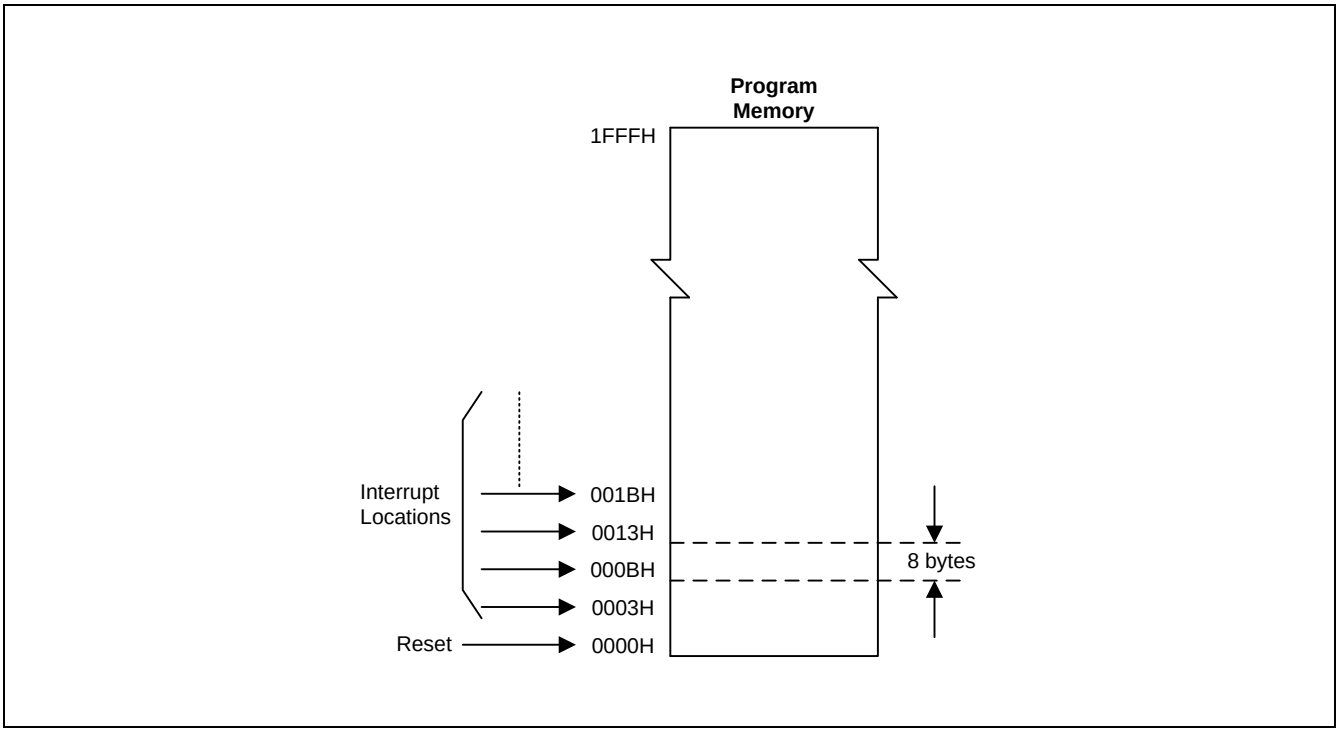
数据存储器使用与程序存储器不同的地址空间。**MA86E/L508** 只有 256 字节的内部没有任何外部的数据存储器。

6.1. 片内程序存储器

程序存储器用来保存让 CPU 进行处理的程序代码，如图 6-1 所示。复位后，CPU 从地址为 0000H 的地方开始运行，用户应用代码的起始部分应该放在这里。为了响应中断，中断服务位置(被称为中断矢量)应该位于程序存储器。每个中断在程序存储器中有一个固定的起始地址，中断使 CPU 跳到这个地址运行中断服务程序。举例来说， 外部中断 0 被指定到地址 0003H，如果使用外部中断 0，那么它的中断服务程序一定是从 0003H 开始的。如果中断未被使用，那么这些地址就可以被一般的程序使用。

中断服务程序的起始地址之间有 8 字节的地址间隔：外部中断 0，*0003H*；定时器 0，*000BH*；外部中断 1，*0013H*；定时器 1，*001BH* 等等。如果中断服务程序足够短，它完全可以放在这 8 字节的空间中。如果其他的中断也被使用的话，较长的中断服务程序可以通过一条跳转指令越过后面的中断服务起始地址。

图 6–1. 程序存储器



6.2. 片内数据存储器

图 6-2 向**MA86E/L508**使用者展示了内部和外部数据存储器的空间划分。内部数据存储器被划分为三部分，通常被称为低 128 字节 RAM ，高 128 字节 RAM 和 128 字节 SFR 空间。内部数据存储器的地址线只有 8 位宽，因此地址空间只有 256 字节。SFR 空间的地址高于 7FH，用直接地址访问；而用间接访问的方法访问高 128 字节的 RAM。这样虽然 SFR 和高 128 字节 RAM 占用相同的地址空间（80H—FFH），但他们实际上是分开的。

如图 6-3 所示，低 128 字节 RAM 与所有 80C51 一样。最低的 32 字节被划分为 4 组每组 8 字节的寄存器组。指令中称这些寄存器为 R0 到 R7。程序状态字 (PSW) 中的两位用于选择哪组寄存器被使用。这使得程序空间能够被更有效的使用，因为对寄存器访问的指令比使用直接地址的指令短。接下来的 16 字节是可以位寻址的存储器空间。80C51 的指令集包含一个位操作指令集，这区域中的 128 位可以被这些指令直接使用。位地址从 00H 开始到 7FH 结束。

所有的低 128 字节 RAM 都可以用直接或间接地址访问，而高 128 字节 RAM 只能用间接地址访问。

图 6-4 给出了特殊功能寄存器 (SFR) 的概览。SFR包括端口寄存器，定时器和外围器件控制器，这些寄存器只能用直接地址访问。SFR 空间中有16个地址同时支持位寻址和直接寻址。可以位寻址的 SFR 的地址末位是 0H 或8H。

图 6–2. 数据存储器

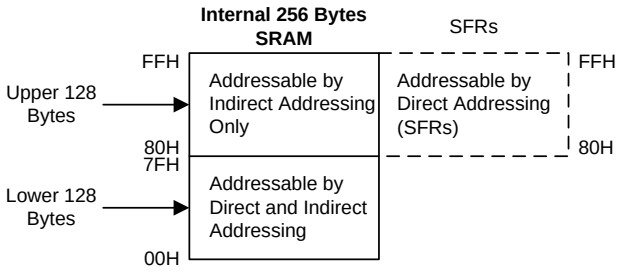


图 6–3. 内部 RAM 的低 128 字节

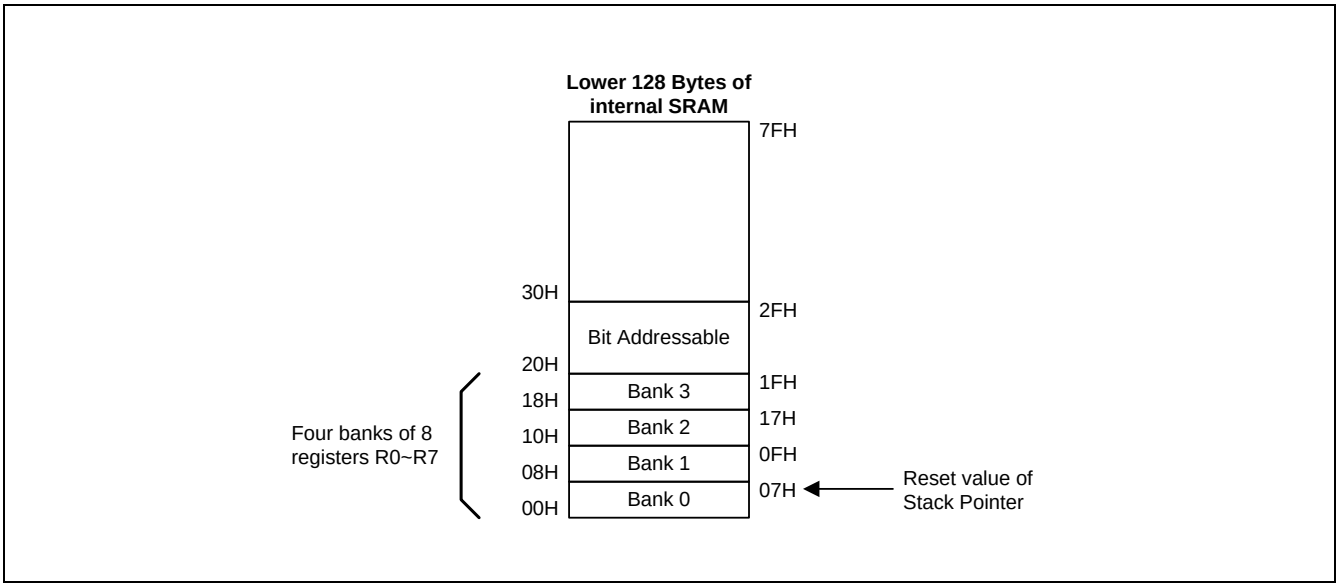
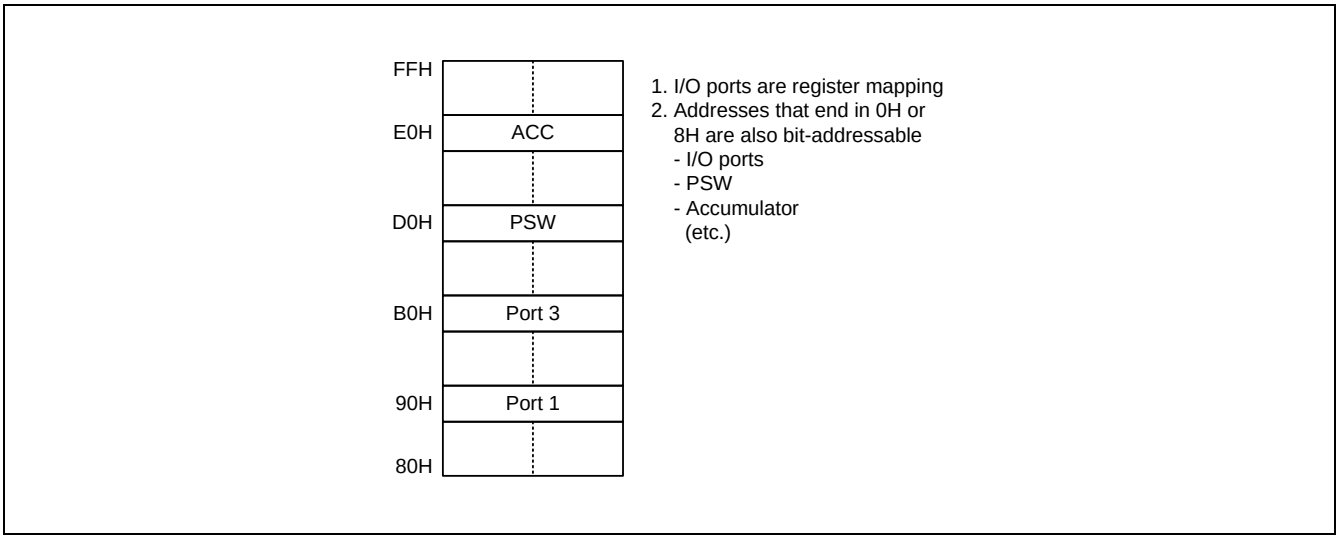


图 6–4. SFR 空间



6.3. 关于 C51 编译器的声明标识符

data

128 字节的内部数据存储空间 (00h~7Fh)。使用除 MOVX 和 MOVC 以外的指令，可以直接或间接的访问。全部或部分的堆栈可能保存在此区域中。

idata

间接数据。256 字节的内部数据存储空间 (00h~FFh) 使用除 MOVX 和 MOVC 以外的指令间接访问。全部或部分的堆栈可能保存在此区域中。此区域包括 data 区 和 data 区以上的 128 字节。

sfr

特殊功能寄存器。CPU 寄存器和外围部件控制/状态寄存器，只能通过直接地址访问。

xdata

没有外部数据或片上的扩展 RAM (XRAM)。

pdata

没有分页的外部数据(256 字节) 或片上的扩展 RAM（XRAM）。

code

8K 程序存储空间。通过“MOVC @A+DTPR”访问，作为程序部分被读取。

7. 数据指针寄存器(DPTR)

MA86E/L508 DPTR 只有一种设置。 **MA86E/L508** 不支持访问外部存储器和 MOVX 指令。

8. 系统时钟

系统时钟有 4 个时钟源：内部快频 RC 震荡器 (IHRCO)，外部晶振 ，内部慢频 RC 震荡器(ILRCO) 和外部频率输入。如图 9-1 所示 **MA86E/L508 系统时钟结构**。

MA86E/L508 默认值是 IHRCO **24.0MHz** MHz 2 分频作为系统时钟，并保留晶振脚 P4.0/P4.1 普通 I/O 口的特性。软件可以根据应用要求自由切换 4 种时钟的任意一种作为系统时钟， 但必须等时钟稳定后才能切换。如果软件选择外部时钟模式， 脚 P4.0 和 P4.1 分配给 XTAL2 和 XTAL1. 并且 P4.0/P4.1 普通 I/O 功能失效。 在外部时钟输入模式 (ECKI)， 时钟源来自 P4.0 ， P4.1 仍然是普通 I/O 口。

在设置 XTALE(CKCON2.4)使能外部晶体震荡，当外部晶体震荡稳定时 XTOR (AUXR1.4)被内部硬件置位便于软件切换外部震荡，XTOR 只能读，MCU 必须在切换晶体震荡为系统时钟之前检测此标志位。

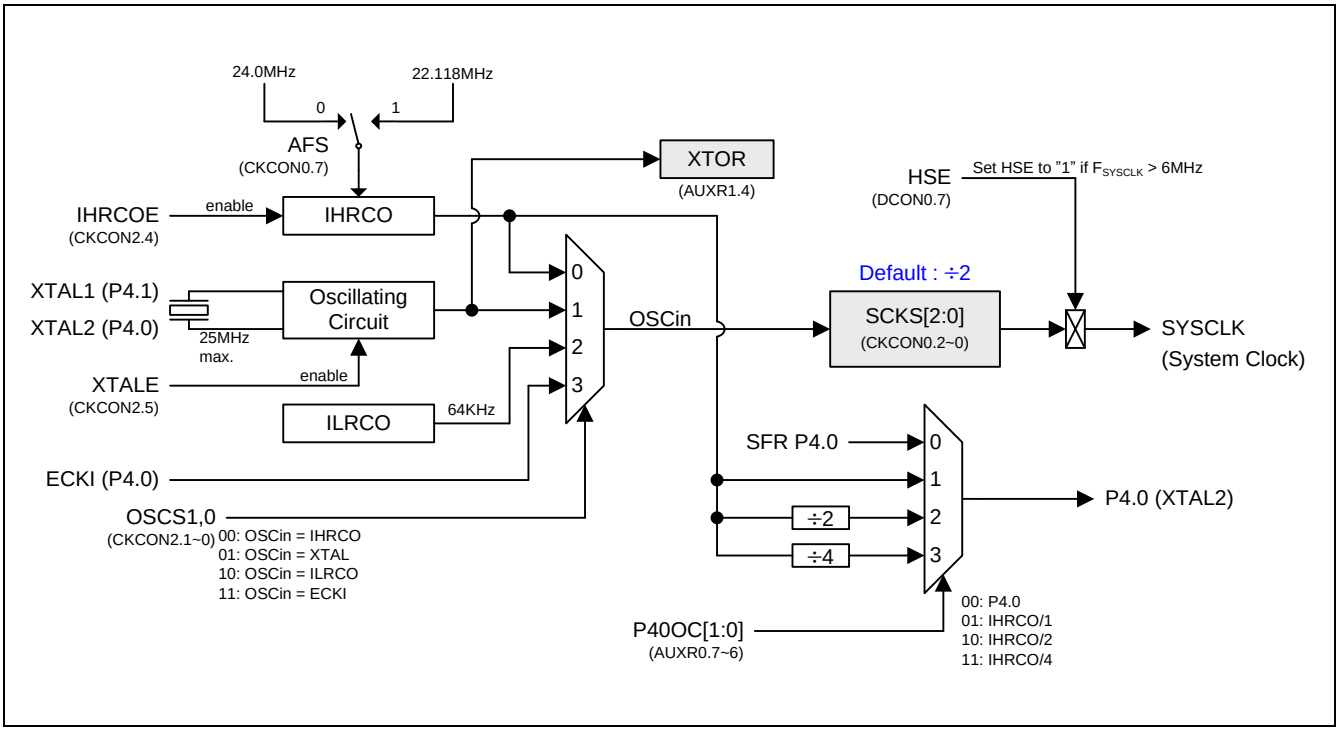
软件可以选择 IHRCO 两种频率，另一种是 **22.118MHz**。 通过软件设置位 AFS （CKCON0.7）来选择， 两种 IHRCO 频率 22.118MHz 和 24.0 MHz 都是高精度系统时钟源。IHRCO 详细的描述请参考“27.4 节 IHRCO ”。在 IHRCO O 模式， P4.0 可以作为全时钟（*OSCin*）输出或 2 分频时钟（*OSCin/2*）输出或 4 分频时钟（*OSCin/4*）输出给其他系统时钟源应用。

时钟分配器分配 4 种时钟源的一种为系统时钟 *SYSCLK*，如下图所示：图 9–1。用户能通过设置 SCKS2~SCKS0 位 (CKCON0 寄存器) 来获得理性的时钟。**MA86E/L508** 在上电复位后系统时钟的默认是 2 分频。

8.1. 时钟结构

图 8-1 显示了 **MA86E/L508** 的时钟系统. 系统时钟可以于外部晶振或内部振荡作为来源。

图 8–1. 系统时钟



8.2. 时钟寄存器

CKCON0:时钟控制寄存器 0

SFR 页 = 普通及 P 页

SFR 地址 = 0xC7 复位值 = 0xxx-x001

7	6	5	4	3	2	1	0
AFS	0	0	0	0	SCKS2	SCKS1	SCKS0
R/W	**W**	**W**	**W**	**W**	**R/W**	**R/W**	**R/W**

Bit 7: AFS, 频率选择

0: 选择 IHRCO 输出 **24.0MHz**。

1: 选择 IHRCO 输出 **22.118MHz**。

Bit 6~3: 保留位。 写 CKCON0 时，这 4 个位必须写“0” 。

Bit 2~0: SCKS2 ~ SCKS0, 系统时钟分频器选择位， SCKS[2:0] 缺损值“001” 选择的系统时钟是 OSCin/2.

SCKS[2:0]	系统时钟
0 0 0	OSCin/1
0 0 1	OSCin /2
0 1 0	OSCin /4
0 1 1	OSCin /8
1 0 0	OSCin /16
1 0 1	OSCin /32
1 1 0	OSCin /64
1 1 1	OSCin /128

CKCON2: 时钟控制寄存器 2

SFR 页 = **仅 P 页**

SFR 地址 = 0x40 复位值= 1101-xx00

7	6	5	4	3	2	1	0
XTGS1	XTGS0	XTALE	IHRCOE	**0**	**0**	OSCS1	OSCS0
R/W	**R/W**	**R/W**	**R/W**	**W**	**W**	**R/W**	**R/W**

Bit 7~6: XTGS1~XTGS0, OSC 驱动控制寄存器.

XTGS1, XTGS0	增益定义
0, 0	32.768K 的增益
1, 1	2MHz ~ 25MHz 的增益
其他	保留

Bit 5: XTALE, 外部晶振(XTAL) 使能。

0: 禁止 XTAL 震荡电路。 在这种情况下， XTAL2 XTAL1 表现为 Port 4.0 Port 4.1.

1: 使能 XTAL 震荡电路。如果软件设置这个位，软件必须判断 **XTOR** (AUXR1.4)此位，表明晶体震荡已经稳定，可以作为系统时钟选项。

Bit 4： IHRCOE，内部快频 RC 震荡使能位。默认侄是 MCU 时钟源

0： 禁止内部快频 RC 震荡电路。

1： 使能内部快频 RC 震荡电路。如果软件设置这个位，必须等待 **32 us** IHRCO 才能稳定输出。

Bit 3~2： 保留位。写 CKCON0 时，这 2 个位必须写“0”。

Bit 1~0：OSCS[1:0]，OSCin 时钟选择。OSCin 缺损选项是 IHRCO。

OSCS[1:0]	OSCin 时钟选择
0 0	IHRCO
0 1	XTAL
1 0	ILRCO
1 1	ECKI, (P4.0)作为 OSCin 的外部时钟输入。

AUXR0: 辅助寄存器 0

SFR 页 = 普通

SFR 地址 = 0xA1 复位值= 0000-0000

7	6	5	4	3	2	1	0
P40OC1	P40OC0	P40FD	T0XL	P1FS1	P1FS0	INT1H	INT0H
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~6: P40 输出配置控制位 1 和位 0，这两位仅仅当内部 RC 震荡（IHRCO ）被选择为系统时钟源时有效。这种情况， XTAL2 和 XTAL1 改变功能作 P4.0 和 P4.1。在外部时钟输入模式，P4.0 专用于时钟输入脚，在内部震荡模式下，P4.0 为普通 I/O 或时钟源发生器提供下列选项， 当 P40OC[1:0] 索引为非 P4.0 GPIO 功能时，P4.0 将驱动内部振荡器（IHRCO ） 输出为其它设备提供时钟源。

P40OC[1:0]	P4.0 function	I/O mode
00	**P4.0**	By P4M0.0
01	**IHRCO/1**	By P4M0.0
10	IHRCO/2	By P4M0.0
11	IHRCO/4	By P4M0.0

当 P4.0 作为时钟输出功能，设置 P4M0.0 为 “1” 选者 P4.0 为推挽输出

Bit 5: P40FD, P4.0 快速驱动标志。

0: P4.0 默认驱动输出

1: P4.0 快速驱动输出使能。若 P4.0 被配置为时钟输出， 当 P4.0 输出频率大于 12MHz （5V） 或者大于 6MHz （3V） 时使能此位。

AUXR1: 辅助寄存器 1

SFR 页 =普通

SFR 地址 = 0xA2 复位值= 0000-0000

7	6	5	4	3	2	1	0
P3TWI	P4S0MI	P2PCA	XTOR	STAF	STOF	BPOC1	BPOC0
R/W	**R/W**	**R/W**	**R**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 1: XTOR, 外部晶体震荡就绪标志位。只读.

0: 外部晶体震荡没有就绪.

1: 外部晶体震荡就绪.。当 XTALE 已经使能， XTOR 传达晶体震荡可以启动记数。.

DCON0: 设备控制 0

SFR 页 = 仅 P 页

SFR 地址 = 0x4C **复位值 = 10xx-xx1x**

7	6	5	4	3	2	1	0
HSE	IAPO	0	0	0	0	RSTIO	0
W	**W**	**W**	**W**	**W**	**W**	**W**	**W**

Bit 7: HSE, 高速工作使能

0: MCU 运行在低速模式降低了内部电路的速度从而减少了功耗。

1: 如果 F_{SYSCLK} > 6MHz 时，使能 MCU 全速运行。

8.3. 系统时钟示例代码

(1)规定功能: IHRCO 从 24MHz 更改到 22.118MHz

汇编语言代码范例:

```
ORL     CKCON0,#(AFS)                   ; 选择 IHRCO 输出 22.118MHz，AFS=0x80
```

C 语言代码范例:

```
CKCON0 |= AFS;                          //选择 IHRCO 输出 22.118MHz，AFS=0x80
```

(2). 规定功能: 系统时钟(SYSCLK)更改为OSCin/1 (默认为OSCin/2)

汇编语言代码范例:

```
ANL     CKCON0,#(AFS)                   ; 设置 SCKS[2:0] = 0 来选择系统时钟(SYSCLK)为 OSCin/1
```

C 语言代码范例:

```
CKCON0 &= ~(SCKS2 | SCKS1 | SCKS0);     //系统时钟(SYSCLK)为 OSCin/1
                                        // SCKS[2:0], 系统时钟(SYSCLK)分频
                                        //     0      | OSCin/1
                                        //     1      | OSCin/2
                                        //     2      | OSCin/4
                                        //     3      | OSCin/8
                                        //     4      | OSCin/16
                                        //     5      | OSCin/32
                                        //     6      | OSCin/64
                                        //     7      | OSCin/128
```

(3). 规定功能: 当 MCU 使用 IHRCO 或 ILRCO 作为时钟源时，选择外部晶振(XTAL)作为时钟源(OSCin) (默认为 IHRCO)

汇编语言代码范例:

```
    MOV     IFADRL,#(CKCON2)            ; 索引 P 页地址为 CKCON2
    CALL    _page_p_sfr_read            ; 读取 CKCON2 的数据

    ORL     IFD,#(XTGS1 | XTGS0 | XTALE)  ; 使能外部晶振(XTALE)并且设置为高增益 (对非 32768Hz 的应用)
                                        ; 对 32768Hz 的外部晶振(XTAL), 设置 XTGS1 = XTGS0 = 0
    CALL    _page_p_sfr_write           ; 写数据到 CKCON2,系统时钟(SYSCLK )必须小于 25MHz

check_XTOR:                             ; 检测外部晶振(XTAL)振荡准备好
    MOV     A,AUXR1
    JNB     ACC.4,check_XTOR            ; 等待 XTOR(AUXR1.4)为 1

    ANL     IFD,#~(OSCS1 | OSCS0)       ; OSCin 时钟源更改为外部晶振(XTAL)
    ORL     IFD,#(OSCS0)
    CALL    _page_p_sfr_write           ; 写数据到 CKCON2

    ANL     IFD,#~(IHRCOE)              ; 如果 MCU 从 IHRCO 更改之后禁止 IHRCO
    CALL    _page_p_sfr_write           ; 写数据到 CKCON2
```

C 语言代码范例:

```
    IFADRL = CKCON2;                    //索引 P 页地址为 CKCON2
    page_p_sfr_read();                  //读取 CKCON2 的数据

    IFD |= XTGS1 | XTGS0 | XTALE;       //使能外部晶振(XTALE)并且设置为高增益 (对非 32768Hz 的应用)
                                        //对 32768Hz 的外部晶振(XTAL), 设置 XTGS1 = XTGS0 = 0
    page_p_sfr_write ();                //写数据到 CKCON2,系统时钟(SYSCLK )必须小于 25MHz

    while(AUXR1 & XTOR == 0x00);        //检测外部晶振(XTAL)振荡准备好
                                        //等待 XTOR(AUXR1.4)为 1

    IFD &= ~(OSCS1 | OSCS0);            // OSCin 时钟源更改为外部晶振(XTAL)
    IFD |= OSCS0;
    page_p_sfr_write ();                //写数据到 CKCON2

    IFD &= ~IHRCOE;                     //如果 MCU 从 IHRCO 更改之后禁止 IHRCO
    page_p_sfr_write();                 //写数据到 CKCON2
```

(4). 规定功能: 当 MCU 使用 IHRCO, ECKI 或 XTAL 作为时钟源时，选择 ILRCO 作为时钟源(OSCin) (默认为 IHRCO)

汇编语言代码范例:

```
MOV     IFADRL,#(CKCON2)          ; 索引 P 页地址为 CKCON2
CALL    _page_p_sfr_read          ; 读取 CKCON2 的数据

ANL     IFD,#~(OSCS1 | OSCS0)     ; OSCin 时钟源更改为 ILRCO
ORL     IFD,#(OSCS1)
CALL    _page_p_sfr_write         ; 写数据到 CKCON2

ANL     IFD,#~(XTALE | IHRCOE)    ; 禁止 XTAL 和 IHRCO
CALL    _page_p_sfr_write         ; 写数据到 CKCON2

MOV     IFADRL,#(DCON0)           ; 索引 P 页地址为 DCON0
CALL    _page_p_sfr_read          ; 读取 DCON0 的数据

ANL     IFD,#~(HSE)               ; 当系统时钟(SYSCLK ≤ 6MHz)时为了省电禁止 HSE
CALL    _page_p_sfr_write         ; 写数据到 DCON0
```

C 语言代码范例:

```
IFADRL = CKCON2;                  //索引 P 页地址为 CKCON2
page_p_sfr_read();                //读取 CKCON2 的数据

IFD = ~(OSCS1 | OSCS0);           // OSCin 时钟源更改为 ILRCO
IFD |= OSCS1;
page_p_sfr_write();               //写数据到 CKCON2

IFD &= ~(XTALE | IHRCOE);         //禁止 XTAL 和 IHRCO
page_p_sfr_write();               //写数据到 CKCON2

IFADRL = DCON0;                   //索引 P 页地址为 DCON0
page_p_sfr_read();                //读取 DCON0 的数据

IFD &= ~HSE;                      //当系统时钟(SYSCLK ≤ 6MHz)时为了省电禁止 HSE
page_p_sfr_write();               //写数据到 DCON0
```

(5). 规定功能: 当 MCU 使用 IHRCO 或 ILRCO 作为时钟源时，选择 ECKI 作为时钟源(OSCin) (默认为 IHRCO)

汇编语言代码范例:

```
MOV     IFADRL,#(CKCON2)          ; 索引 P 页地址为 CKCON2
CALL    _page_p_sfr_read          ; 读取 CKCON2 的数据

ORL     IFD,#(OSCS1 | OSCS0)      ; OSCin 时钟源更改为 ECKI
CALL    _page_p_sfr_write         ; 写数据到 CKCON2,系统时钟(SYSCLK )必须小于 25MHz

ANL     IFD,#~(XTALE | IHRCOE)    ; 禁止 IHRCO 和 XTAL
CALL    _page_p_sfr_write         ; 写数据到 CKCON2
```

C 语言代码范例:

```
IFADRL = CKCON2;                  // 索引 P 页地址为 CKCON2
page_p_sfr_read();                //读取 CKCON2 的数据

IFD |= OSCS1 | OSCS0;             // OSCin 时钟源更改为 ECKI
page_p_sfr_write ();              //写数据到 CKCON2,系统时钟(SYSCLK )必须小于 25MHz

IFD &= ~(XTALE | IHRCOE);         //禁止 IHRCO 和 XTAL
page_p_sfr_write ();              //写数据到 CKCON2
```

(6). 规定功能: 当 MCU 使用 ILRCO, ECKI 或 XTAL 作为时钟源时，选择 IHRCO 作为时钟源(OSCin)

汇编语言代码范例:

```
MOV     IFADRL,#(CKCON2)          ; 索引 P 页地址为 CKCON2
CALL    _page_p_sfr_read          ; 读取 CKCON2 的数据

ORL     IFD,#(IHRCOE)             ; 使能 IHRCO
CALL    _page_p_sfr_write         ; 写数据到 CKCON2

Delay_32us

ANL     IFD,#~(OSCS1 | OSCS0)     ; OSCin 时钟源更改为 IHRCO
```

```
CALL    _page_p_sfr_write          ; 写数据到 CKCON2

ANL     IFD,#~(XTALE)              ; 禁止 XTAL
CALL    _page_p_sfr_write          ; 写数据到 CKCON2
```

C 语言代码范例:

```
IFADRL = CKCON2;                   //索引 P 页地址为 CKCON2
page_p_sfr_read();                 //读取 CKCON2 的数据

IFD |= IHRCOE;                     // 使能 IHRCO
page_p_sfr_write();                //写数据到 CKCON2

Delay 32us

IFD &= ~(OSCS1 | OSCS0);           // OSCin 时钟源更改为 IHRCO
page_p_sfr_write();                //写数据到 CKCON2

IFD &= ~ XTALE;                    // 禁止 XTAL
page_p_sfr_write();                //写数据到 CKCON2
```

(7). 规定功能: IHRCO 频率输出在 P4.0

汇编语言代码范例:

```
MOV     P4M0,#P4M00                ; 设置 P4.0 为推挽输出模式
ANL     AUXR0,#~(P40OC1|P40OC0)    ; P4.0 更改为通用输入输出口(GPIO)功能
ORL     AUXR0,#(P40OC0|P4FD)       ; P4.0 = IHRCO 频率 + 引脚快速驱动
                                   ; P40OC[1:0]  |  P4.0
                                   ; 00          |  GPIO
                                   ; 01          |  IHRCO/1
                                   ; 10          |  IHRCO/2
                                   ; 11          |  IHRCO/4
```

C 语言代码范例:

```
    P4M0 |= P4M00;                          //设置 P4.0 为推挽输出模式
    AUXR0 &= ~(P40OC0 | P40OC1);             // P4.0 更改为通用输入输出口(GPIO)功能
    AUXR0 |= (P40OC0 | P4FD);                // P4.0 输出 IHRCO/1
//    AUXR0 = P40OC1|P4FD;                   // P4.0 输出 IHROC/2
//    AUXR0 = P40OC1|P40OC0|P4FD;            // P4.0 输出 IHRCO/4
```

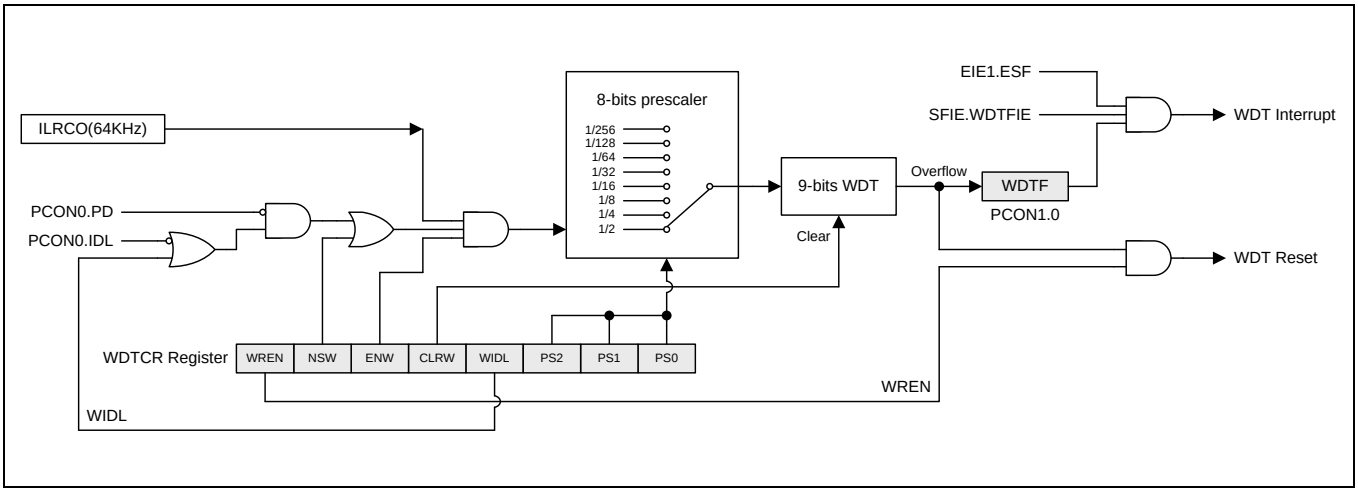
9. 看门狗定时器(WDT)

9.1. WDT 结构

看门狗定时器 (WDT) 用来使程序从跑飞或死机状态恢复的一个手段。WDT 由一个 9 位独立定时器、一个 8 分频器和一个控制寄存器(WDTCR)组成。 图 9-1 显示 **MA86E/L508** WDT 结构框图。

当 WDT 使能，时钟源来自 64KHz ILRCO。WDT 溢出会设置位 WDTF PCON1.0， 也能产生中断通过使能位 WDTFIE (SFIE.0) 和 ESF (EIE1.3). 溢出也能触发系统复位通过设置位 WREN (WDTCR.7) 。软件可以在溢出之前在 CLRW 位 (WDTCR.4)上写“1” 来清除它，可以阻止 WDT 溢出。
一旦 WDT 使能通过设置位 ENW， 将没有办法使之失效除非上电复位或在 page-p SFR 覆盖 ENW，能清除位 ENW。 WDTCR 会保持以前的值不会改变在硬件(RST-pin)复位、软件复位和 WDT 复位后 。
WREN, NSW 和 ENW 都是一次性使能生效，写“1”使能。在 Page-P 中写“0”到位 WDTCR.7~5 禁止 WREN, NSW 和 ENW 使用。祥见“9.3 WDT 寄存器” WDT 章节和“23 P 页 SFR 访问”P 页访问章节。

图 9–1. 看门狗定时器



9.2. WDT 在掉电模式和空闲模式期间

空闲模式， 位标志 WIDL (WDTCR.3) 决定 WDT 是否计数。 设置这个位能让 WDT 在空闲模式一直计数。如果硬件选项 WDTRCO 使能， WDT 会一直保持计数不管位 WIDL 设置情况 。

掉电模式， ILRCO 不会停如果 NSW (WDTCR.6) 使能。 这会让 WDT 保持计数即使掉电模式下(Watch 模式)。WDT 溢出后，软件能设置进入中断或复位 唤醒 CPU。

9.3. WDT 寄存器

WDTCR: 看门狗定时器控制寄存器

SFR 页 = 普通及 P 页

SFR 地址 = 0xE1 POR = 0000-0111 (xxx0_xxxx 硬件选项设定)

7	6	5	4	3	2	1	0
WREN	NSW	ENW	CLRW	WIDL	PS2	PS1	PS0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: WREN, WDT 复位使能标志， 初始值随硬件选项 WRENO。

0: WDT 溢出不产生复位 。 WDT 溢出标志 WDTF 可以供软件检测或触发中断。

1: WDT 溢出产生系统复位。 一旦 WREN 已经设置， 不能用软件在常规页中清除，**但在 P 页中，软件能修改其值 “0” 或 “1”。**

Bit 6: NSW. 不停止的 WDT 标志。 初始值随硬件选项 NSWDT.

0: WDT 停止计数 MCU 在掉电模式。

1: WDT 永远不会停止计数 MCU 在掉电模式(Watch 模式)或空闲模式。 一旦 NSW 已经设置， 不能用软件在常规页中清除， **但在 P 页中，软件能修改其值 “0” 或 “1”。**

Bit 5: ENW. 使能 WDT 标志。

0: 禁止 WDT 运行。

1: 使能 WDT，一旦 ENW 已经被设置，不能用软件在常规页中清除，**但在 P 页中，软件能修改其值 “0” 或 “1”。**

Bit 4: CLRW. WDT 计数器清零位.

0: 在此位写“0”WDT 没有任何操作 .

1: 在此位写 “1” 会清除 9 位 WDT 计数器到 000H， 注意此位不能写“0”清除，设置此位清除 WDT 重新计数。

Bit 3: WIDL. WDT 空闲模式控制。

0: WDT 停止计数 MCU 在空闲模式。

1: WDT 保持计数 MCU 在空闲模式。

Bit 2~0: PS2 ~ PS0，选择分频器输出作 WDT 基础时钟输入（分频系数设置）

PS[2:0]	预分频值	WDT 周期
0 0 0	2	15 ms
0 0 1	4	31 ms
0 1 0	8	62 ms
0 1 1	16	124 ms
1 0 0	32	248 ms
1 0 1	64	496 ms
1 1 0	128	992 ms
1 1 1	256	1.984 S

PCON1: 电源控制寄存器 1

SFR 页 =普通及 P 页

SFR 地址 = 0x97 POR = 00x0-0x00

7	6	5	4	3	2	1	0
SWRF	EXRF	--	RTCF	KBIF	--	BOF0	WDTF
R/W	**R/W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 0: WDTF, WDT 溢出标志。

0: 必须由软件写**“1”** 清除， 软件写“:0” 不操作。

1: 当 WDT 溢出时硬件置位此位，写 **“1”** 清除 WDTF。

9.4. WDT 硬件选项

除了由软件初始化外，WDTCR 寄存器还能在上电的时候由硬件选项 WRENO,NSWDT,HWENW,HWWIDL 和 HWPS[2:0]来自动初始化，这些选项通过通用编程器来编程，如下所述。

如果 HWENW 编程为“使能”，则硬件在上电时为 WDTCR 寄存器作如下的初始化工作：（1）ENWI 位置 1。（2）载入 WRENO 的值到 WREN 位。(3)载入 NSWDT 的值到 NSW 位。(4)载入 HWWIDL 的值到 WIDL 位。（5）载入 HWPS【2：0】的值到 PS【2：0】位。

如果 HWENW 和 WDSFWP 都被编程为“使能”，则硬件仍然会在上电时由 WDT 硬件选项初始化 WDTCR 寄存器的内容。之后，任何对 WDTCR 的位的写动作都会被忽略，除了写“1”到 WDTCR.4(CLRW)位来清 WDT 之外，即使通过对 P 页 SFR 的操作机制也不行。

WRENO:

☑:使能: 置位 WDTCR.WREN 以使能 WDTF 系统复位功能。

☐:禁止: 清除 WDTCR.WREN 以禁止 WDTF 系统复位功能。

NSWDT: 不停止 WDT

☑:使能：使能 WDT 在掉电模式也保持运行，设置位 WDTCR.NSW (watch 模式)。

☐:禁止: 禁止 WDT 在掉电模式下运行，清除位 WDTCR.NSW (禁止 Watch 模式)。

HWENW: 硬件加载 WDTCR 的“ENW”

☑:使能： 上电时自动硬件使能看门狗定时器，并且自动加载 WRENO, NSWDT, HWWIDL 和 HWPS2~0 的值到 WDTCR 中。

☐:上电时看门狗定时器（WDT）不自动使能。

HWWIDL, HWPS2, HWPS1, HWPS0:

当 HWENW 被使能，上电复位时，这四个保险丝位将被载入到特殊功能寄存器 WDTCR 中。

WDSFWP:

☑:使能. WDT 特殊功能寄存器 WDTCR 位 WREN, NSW, WIDL, PS2, PS1 和 PS0 软件写保护。

☐:禁止. WDT 特殊功能寄存器 WDTCR 位 WREN, NSW, WIDL, PS2, PS1 和 PS0 软件写保护。

9.5. WDT 示例代码

(1) 规定功能: 使能 WDT 并且选择 WDT 周期为 248 毫秒(ms)

汇编语言代码范例:

```
ANL     PCON1,#(WDTF)                   ; 清除 WDTF 标志(写“1”)
MOV     WDTCR,#(ENW | CLRW | PS2)       ; 使能 WDT 计数器并且设置 WDT 周期为 248 毫秒(ms)
```

C 语言代码范例:

```
PCON1 &= WDTF;                          //清除 WDTF 标志(写“1”)
WDTCR = (ENW | CLRW | PS2);             //使能 WDT 计数器并且设置 WDT 周期为 248 毫秒(ms)
                                        // PS[2:0]   | WDT  周期选择
                                        //     0     | 15ms
                                        //     1     | 31ms
                                        //     2     | 62ms
                                        //     3     | 124ms
                                        //     4     | 248ms
                                        //     5     | 496ms
                                        //     6     | 992ms
                                        //     7     | 1.984s
```

(2) 规定功能: 如何禁止 WDT

汇编语言代码范例:

```
MOV     IFD,WDTCR                       ; 读取 WDTCR 数据
ANL     IFD,#~(ENW)                     ; 清除 ENW 而禁止 WDT

MOV     IFADRL,#(WDTCR_P)               ; 索引 P 页地址为 WDTCR_P
CALL    _page_p_sfr_write               ; 写数据到 WDTCR
```

C 语言代码范例:

```
IFD = WDTCR;                            //读取 WDTCR 数据
IFD &= ~ENW;                            //清除 ENW 而禁止 WDT
```

```
IFADRL = WDTCR_P;                    //索引 P 页地址为 WDTCR_P
page_p_sfr_write();                  //写数据到 WDTCR
```

(3). 规定功能: 使能 WDT 复位功能并且选择 WDT 周期为 62 毫秒(ms)

汇编语言代码范例:

```
ANL     PCON1,#(WDTF)                ; 清除 WDTF 标志(写“1”)
MOV     WDTCR,#(WREN | CLRW | PS1)   ; 使能 WDT 复位功能并且设置 WDT 周期为 62 毫秒(ms)

ORL     WDTCR,#(ENW)                 ; 使能 WDT 计数器, WDT 运行
```

C 语言代码范例:

```
PCON1 &= WDTF;                       //清除 WDTF 标志(写“1”)
WDTCR = WREN | CLRW | PS1;           //使能 WDT 复位功能并且设置 WDT 周期为 62 毫秒(ms)

WDTCR |= ENW;                        //使能 WDT 计数器, WDT 运行
```

(4). 规定功能:使能 WDTCR 的写保护

汇编语言代码范例:

```
ANL     PCON1,#(WDTF)                ; 清除 WDTF 标志(写“1”)
MOV     WDTCR,#(ENW | CLRW | PS2)    ; 使能 WDT 计数器并且设置 WDT 周期为 248 毫秒(ms)

MOV     IFADRL,#(SPCON0)             ; 索引 P 页地址为 SPCON0
CALL    _page_p_sfr_read             ; 读取 SPCON0 数据

ORL     IFD,#(WRCTL)                 ; 使能 WDTCR 的写保护
CALL    _page_p_sfr_write            ; 写数据到 SPCON0

MOV     IFD,WDTCR                    ; 读取 WDTCR 数据
ORL     IFD,#(CLRW)                  ; 使能 CLRW

MOV     IFADRL,#(WDTCR_P)            ; 索引 P 页地址为 WDTCR_P
CALL    _page_p_sfr_write            ; 写数据到 WDTCR 而清零 WDT 计数器
```

C 语言代码范例:

```
PCON1 &= WDTF;                          //清除 WDTF 标志(写“1”)
WDTCR = ENW | CLRW | PS2;               //使能 WDT 计数器并且设置 WDT 周期为 248 毫秒(ms)

IFADRL = SPCON0;                        //索引 P 页地址为 SPCON0
page_p_sfr_read();                      //读取 SPCON0 数据

IFD |= WRCTL;                           //使能 WDTCR 的写保护
page_p_sfr_write();                     //  写数据到 SPCON0

IFD = WDTCR;                            //读取 WDTCR 数据
IFD |= CLRW;                            //  使能 CLRW

IFADRL = WDTCR_P;                       //索引 P 页地址为 WDTCR_P
page_p_sfr_write();                     //写数据到 WDTCR  而清零 WDT 计数器
```

10. 实时时钟(RTC)/系统时间

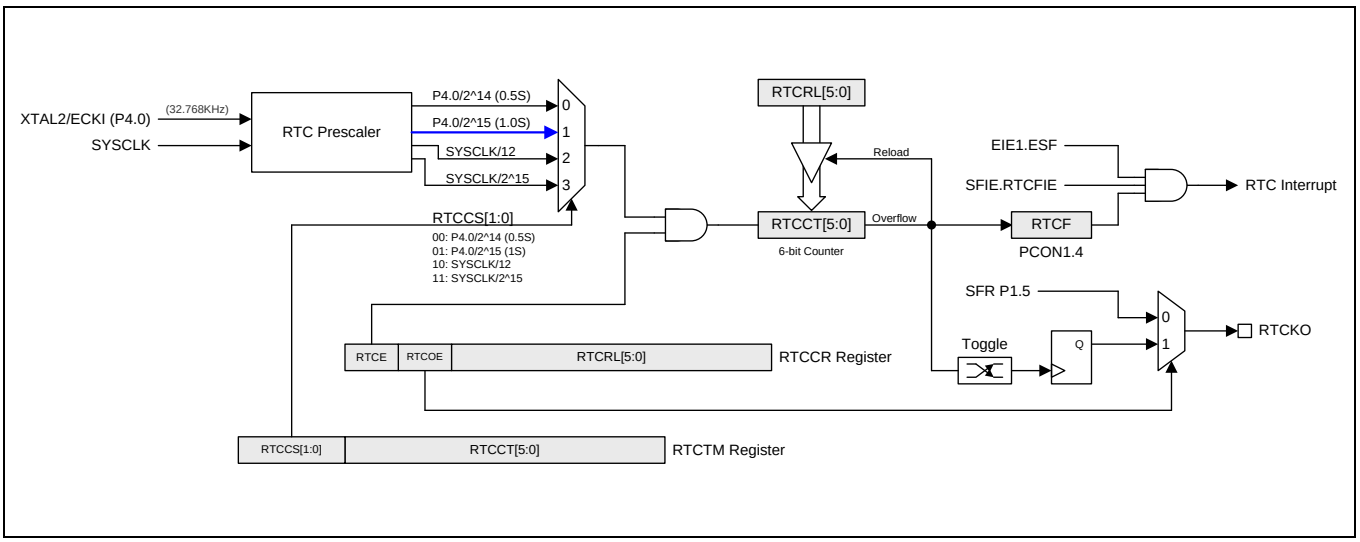
10.1. RTC 结构

MA86E/L508 有一个简单的实时时钟允许使用者不停的记一个准确的时间在其它设备在掉电模式下。实时时钟能用于唤醒或中断源。实时时钟是一个 21 位的计数器包含 14/15 位的一个预分频器和一个 6 位的重载计数器。当其溢出，这个计数器会被重新加载并且 RTCF 旗标被设置。预分频器的时钟源来自内部系统时钟(SYSCLK)或者 XTAL 震荡器，条件是 XTAL 震荡器不可以作为系统时钟。图 10–1 显示 **MA86E/L508** RTC 结构。

RTC 模组输入是 32.768KHz 震荡器可以程控提供时间段为 0.5S 到 64S。这个计数器也可以提供一个定时功能为 SYSCLK/12 或 SYSCLK/2^15 一个短的定时功能或一个长的系统定时功能。最大的系统溢出时间是 SYSCLK/2^21。

如果 XTAL 震荡器被用于系统时钟，P4.0 仍然作为 RTC 时钟输入源。只有上电、复位会重置 RTC 和它相应的特殊功能寄存器为默认值

图 10–1. 实时时钟计数器



10.2. RTC 寄存器

RTCCR: 实时时钟控制寄存器

SFR 页 = 普通及 P 页

SFR 地址 = 0xBE POR = 0011-1111

7	6	5	4	3	2	1	0
RTCE	RTCOE	RTCRL.5	RTCRL.4	RTCRL.3	RTCRL.2	RTCRL.1	RTCRL.0
R/W	**W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: RTCE, RTC 使能.

0: 停止 RTC 计数器, RTCCT.

1: 使能 RTC 计数器并且当 RTCCT 溢出时置位 RTCF，当 RTCE 被设置，CPU 不能访问 RTCTM，只有当 RTCE 被清除后才能访问。

Bit 6: RTCOE, RTC 输出使能。 RTCKO 输出频率是 (RTC 溢出率)/2.

0: 禁止 RTCKO 输出.

1: 使能 RTCKO 输出.

Bit 5~0: RTCRL[5:0], RTC 计数器重载值寄存器。 当寄存器被 CPU 访问，且 RTCCT 溢出时寄存器值会被重载到 RTCCT 。

RTCTM: 实时时钟定时器寄存器

SFR 页 = 普通

SFR 地址 = 0xB6 POR = 0111-1111

7	6	5	4	3	2	1	0
RTCCS.1	RTCCS.0	RTCCT.5	RTCCT.4	RTCCT.3	RTCCT.2	RTCCT.1	RTCCT.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~6: RTCCS.1~0, RTC 时钟选择. 缺损值是 “01”

RTCCS[1:0]	时钟源	RTC 中断周期	最小周期
0 0	P4.0/2^14	0.5S ~ 32S 当 P4.0 = 32768Hz	0.5S
0 1	P4.0/2^15	1S ~ 64S 当 P4.0 = 32768Hz	1S
1 0	SYSCLK/12	1us ~ 64us 当 SYSCLK = 12MHz	1us
1 1	SYSCLK/2^15	2.73ms ~ 174.72ms 当 SYSCLK = 12MHz	2.73ms

Bit 5~0: RTCCT[5:0], RTC 计数器寄存器。 通过选择不同的时钟源 RTCCS[1:0]来选择 RTC 功能或系统定时功能。 当计数器溢出，置位 RTCF 旗标并且 RTCFIE 使能会产生系统旗标中断。 最大的 RTC 溢出时间为 64 秒。

PCON1: 电源控制寄存器 1

SFR 页 = 普通及 P 页

SFR 地址 = 0x97 POR = 00x0-0x00

7	6	5	4	3	2	1	0
SWRF	EXRF	--	**RTCF**	KBIF	--	BOF0	WDTF
R/W	**R/W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 4: RTCF, RTC 溢出旗标

0: 这位必须通过软件写**“1”**清除，软件写“0”不操作。

1: 当 RTCCT 溢出此位仅仅被硬件置位，写**“1”**清除 RTCF。

SFIE: 系统旗标中断使能寄存器

SFR 页 = 普通

SFR 地址 = 0x8E POR = 00x0-0x00

7	6	5	4	3	2	1	0
UTIE	SDIFIE	--	**RTCFIE**	KBFIE	--	BOF0IE	WDTFIE
R/W	**R/W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 4: RTCFIE, 使能 Enable RTCF (PCON1.4) 中断。

0: 禁止 RTCF 中断。

1: 使能 RTCF 中断。 如果使能。 RTCF 能唤醒 CPU 在空闲模式或掉电模式

10.3. RTC 示例代码

(1). 规定功能: 使能外部振荡(XTAL) 32.768KHz 作为 RTC 的应用

汇编语言代码范例:

```
    MOV     IFADRL,#(CKCON2)          ; 索引 P 页地址为 CKCON2
    CALL    _page_p_sfr_read          ; 读取 CKCON2 数据

    ANL     IFD,#~(XTGS1 | XTGS0)     ; 设置对 32.768KHz 的外部振荡(XTAL)为低增益
    ORL     IFD,#(XTALE)              ; 使能外部振荡(XTAL) 振荡
    CALL    _page_p_sfr_write         ; 写数据到 CKCON2

check_XTOR_0:                         ; 检测外部振荡(XTAL) 振荡准备好
    MOV     A,AUXR1
    JNB     ACC.4,check_XTOR_0        ; 等待 XTOR(AUXR1.4) 为 1
```

C 语言代码范例:

```
    IFADRL = CKCON2;                  // 索引 P 页地址为 CKCON2
    page_p_sfr_read();                //读取 CKCON2 数据

    IFD &= ~( XTGS1 | XTGS0 );        //设置对 32.768KHz 的外部振荡(XTAL)为低增益
    IFD |= XTALE;                     //使能外部振荡(XTAL) 振荡
    page_p_sfr_write();               // 写数据到 CKCON2

    while( AUXR1&XTOR == 0x00 );      //检测外部振荡(XTAL) 振荡准备好
                                      //等待 XTOR(AUXR1.4) 为 1
```

(2) 规定功能: 使能 174.72 毫秒(ms)周期的系统定时器中断 (默认的系统时钟 SYSCLK = IHRCO/2 = 12MHz)

汇编语言代码范例:

```
    ORG     0003Bh
SystemFlag_ISR:
    ANL     PCON1,#(RTCF)             ; 清除 RTC 标志 (写“1”)
    RETI
```

```
main:
    ANL     PCON1,#(RTCF)                   ; 清除 RTC 标志(写“1”)

    MOV     RTCTM,#(RTCCS1 | RTCCS0)        ; 选择 SYSCLK/2^15 作为 RTC 计数器时钟源
                                            ; RTCCT[5:0] = 0  为 174.72 毫秒(ms)周期

    MOV     RTCCR,#(RTCE)                   ; 设置 RTC 重载计数, RTCRL[5:0] = 0 为 174.72 毫秒(ms)周期
                                            ; 使能 RTC 计数器

    ORL     SFIE,#(RTCFIE)                  ; 使能 RTC 中断
    ORL     EIE1,#(ESF)                     ; 使能系统标志中断
    SETB    EA                              ; 使能全局中断
```

C 语言代码范例:

```
void SystemFlag_ISR (void) interrupt 7
{
    PCON1 &= RTCF;                          //清除 RTC 标志  (写“1”)
}

viod main (void)
{
    PCON1 &= RTCF;                          //清除 RTC 标志  (写“1”)

    RTCTM = RTCCS1 | RTCCS0;                //选择 SYSCLK/2^15 作为 RTC 计数器时钟源
                                            // RTCRL[5:0] = 0 为 174.72 毫秒(ms)周期
    RTCCR = RTCE;                           //设置 RTC 重载计数, RTCRL[5:0] = 0 为 174.72 毫秒(ms)周期
                                            //使能 RTC 计数器

    SFIE |= RTCFIE;                         //使能 RTC 中断
    EIE1 |= ESF;                            //使能系统标志中断
    EA = 1;                                 //使能全局中断
}
```

(3). 规定功能: 使能 RTCKO 输出 SYSCLK/12/2

汇编语言代码范例:

```
ORL     P1M0,#20H                   ; 设置 RTCKO (P1.5)为推挽输出模式
MOV     RTCTM,#0BFH                 ; RTC 时钟选择 SYSCLK/12 并且设置 RTCCT[5:0] = 3Fh
MOV     RTCCR,#03FH                 ; 设置 RTCRL[5:0] = 3Fh
ORL     RTCCR,#(RTCE|RTCOE)         ; 使能 RTC 计数器并且 RTCKO 输出
```

C 语言代码范例:

```
P1M0 |= 0x20;                       //设置 RTCKO (P1.5)为推挽输出模式
RTCTM = 0xBF;                       // RTC 时钟选择 SYSCLK/12 并且设置 RTCCT[5:0] = 3Fh
RTCCR |= 0x3F;                      //  设置 RTCRL[5:0] = 3Fh
RTCCR |= (RTCE | RTCOE);            //使能 RTC 计数器并且 RTCKO 输出
```

11. 系统复位

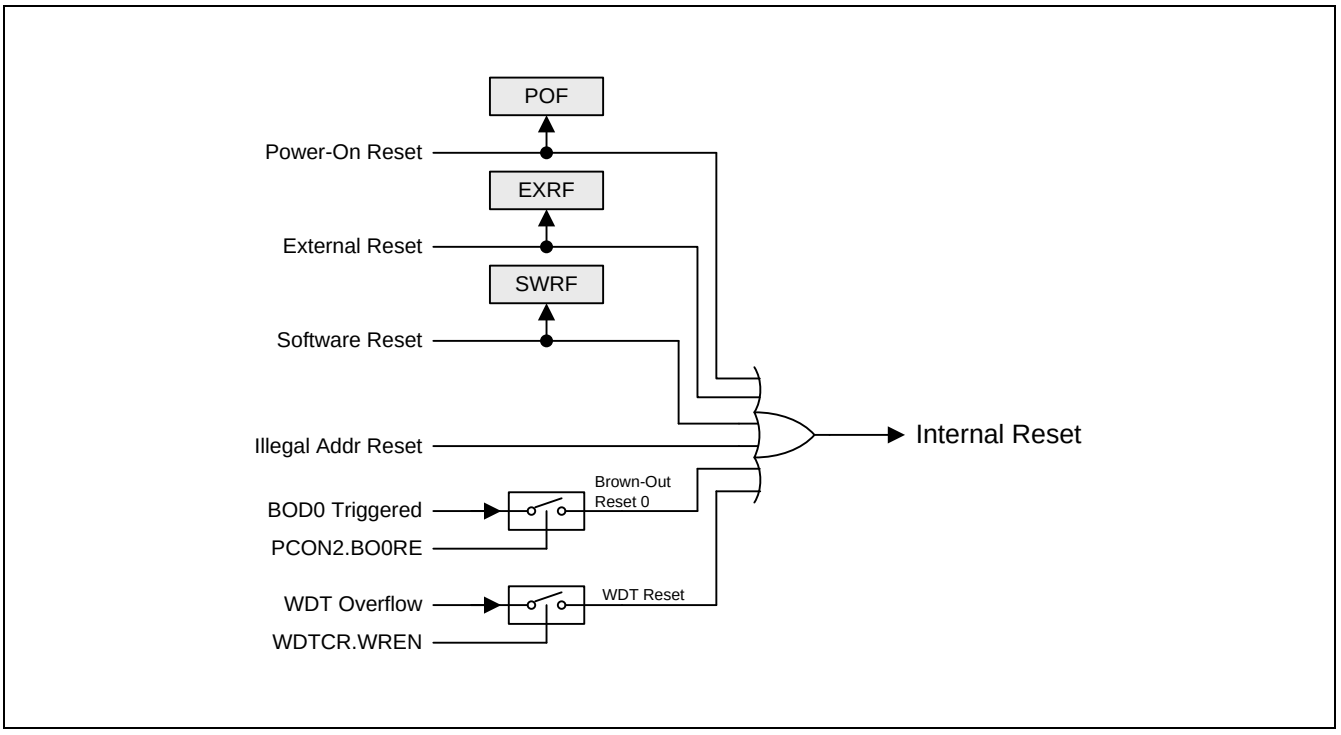
复位期间， 所有的 I/O 寄存器都设置为初始值，程序会根据 OR 设置选择从复位向量的 0000H 开始运行，或者根据 OR 设置从 ISP 地址开始运行。The **MA86E/L508** 有 6 种复位源 ： 上电复位， 外部复位， 软件复位， 非法地址复位， WDT 复位和 低电压复位。 如图所示：图 11-1 系统复位源（ **MA86E/L508**）。

下面的选项描述复位产生源及其相应的控制寄存器和指示标志。

11.1. 复位源

图 11–1 展示了 **MA86E/L508** 的复位系统，和所有复位源

图 11–1. 系统复位源



11.2. 上电复位

上电复位 (POR)用于在电源上电过程中产生一个复位信号。微控制器在 VDD 电压上升到 V_{POR} (POR 开始电压)电压之前将保持复位状态。 VDD 电压降到 V_{POR} 之下后微控制器将再次进入复位状态。 在一个电源周期中， 如果需要再产生一次上电复位 VDD 必须降到 V_{POR} 之下。

PCON0: 电源控制寄存器 0

SFR 页　　= 普通及 P 页

SFR 地址　　= 0x87　　POR = 0001-0000, 复位值 = 000X-0000

7	6	5	4	3	2	1	0
SMOD1	SMOD0	GF	**POF**	GF1	GF0	PD	IDL
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 4: POF. 上电旗标.

0: 这标志必须通过软件清零以便认出下一个复位类型。

1: 当VDD从0 伏上升到正常电压时硬件复位， POF 也能有软件置位。

上电标志 POF 在上电过程中由硬件置“1” 或当 VDD 电压降到 V_{POR} 电压之下时由硬件置“1”。它能通过软件来清除但不受任何热复位（譬如：外部 RST 引脚复位、掉电检测器 Brown-Out 复位、软件(ISPCR.5)复位和 WDT 复位）的影响。它帮助用户检测 CPU 是否从上电开始运行。注意：POF 必须由软件清除。

11.3. 外部复位

保持复位引脚 RST 至少 24 个振荡周期的高电平，将产生一个复位信号，为确保 MCU 正常工作，必须在 RET 引脚上连接可靠的硬件复位电路。

PCON1: 电源控制寄存器 1

SFR 页　　=普通及 P 页

SFR 地址　　= 0x97　　POR = 00x0-0x00

7	6	5	4	3	2	1	0
SWRF	**EXRF**	--	RTCF	KBIF	--	BOF0	WDTF
R/W	**R/W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 6: EXRF，外部复位标志。

0: 这位必须通过软件清零，写**“1”** 清零，写 “:0” 无效。

1: 若外部复位产生则被硬件置位，写**“1”** 清零 EXRF。

11.4. 软件复位

软件通过对 SWRST(ISPCR.5) 位写“1”触发一个系统热复位， 软件复位后，硬件置位 SWRF 标志(PCON1.7)。SWBS 标志决定 CPU 是从 ISP 还是 AP 区域开始运行程序。

ISPCR: ISP 控制寄存器

SFR 页 = 普通

SFR 地址 = 0xE5 复位值 = 0000-xxxx

7	6	5	4	3	2	1	0
ISPEN	**SWBS**	**SWRST**	CFAIL	-	--	--	--
R/W	**R/W**	**R/W**	**R/W**	**W**	**W**	**W**	**W**

Bit 6: SWBS, 软件引导控制

0: 用来选择软件复位后从 AP-空间。

1: 用来选择软件复位后从 ISP-空间。

Bit 5: SWRST, 软件复位触发控制

0: 无操作

1: 产生软件系统复位，它将被硬件自动清除。

PCON1: 电源控制寄存器 1

SFR 页 =普通及 P 页

SFR 地址 = 0x97 POR = 00x0-0x00

7	6	5	4	3	2	1	0
SWRF	EXRF	--	RTCF	KBIF	--	BOF0	WDTF
R/W	**R/W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 7: SWRF, 软件复位标志。

0: 这位必须通过软件清零 ，写“1” 清零，写 “0”无操作。

1: 软件复位产生时硬件置位此位，写“1” 清零 SWRF。

11.5. 掉电检测器（Brown-Out）复位

MA86E/L508 中，掉电检测器(BOD0)检测电源电压（VDD），掉电检测器(BOD0) 的检测固定点为 VDD=2.6V（L 系列）/VDD=4.2V（E 系列）， 如果 VDD 电压低于 BOD0 检测点， 则置位 BOF0 标志，如果 BO0RE (PCON2.1) 被使能， BOD0 事件将触发一个 CPU 复位并置位 BOF0 指示一个掉点检测器（BOD0） 复位发生。

PCON1: 电源控制寄存器 1

SFR 页 =普通及 P 页

SFR 地址 = 0x97 POR = 00x0-0x00

7	6	5	4	3	2	1	0
SWRF	EXRF	--	RTCF	KBIF	--	**BOF0**	WDTF
R/W	**R/W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 1: BOF0, BOF0 (复位) 旗标.

0: 这位必须通过软件清零 ，写“1” 清零，写 “0”无操作。

1: 当 VDD 电压碰到 BOD0 检测点时，硬件置位此位，写“1” 清零。如果 BO0RE (PCON2.1) 被使能，BOD0 事件将触发一个 CPU 复位并置位 BOF0 指示一个掉点检测器（BOD0）复位发生。

11.6. WDT 复位

当 WDT 使能开始计数， WDT 溢出时置位 WDTF 标志 。 如果 WREN (WDTCR.7) 使能， WDT 溢出将引起一个系统热复位， 软件可以读 WDTF 标志来确认 WDT 复位发生。

PCON1: 电源控制寄存器 1

SFR 页 =普通及 P 页

SFR 地址 = 0x97 POR = 00x0-0x00

7	6	5	4	3	2	1	0
SWRF	EXRF	--	RTCF	KBIF	--	BOF0	**WDTF**
R/W	**R/W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 0: WDTF, WDT 溢出/复位 标志。

0: 这位必须通过软件清零 ，写“1” 清零，写 “0”无操作。

1: 当 WDT 溢出产生时硬件置位此位，写“1” 清零。如果 位 WREN (WDTCR.7) 被设置， WDTF 标志指示一个 WDT 复位产生。

11.7. 非法地址复位

MA86E/L508, 中，如果程序运行到非法地址诸如超过 ROM 限制程序地址时触发一个 CPU 热复位并置位 IARF（PCON1.4）标志，以指示一个非法地址复位发生。

11.8. 复位示例代码

(1) 规定功能: 触发一个软件复位

汇编语言代码范例:

```
ORL     ISPCR,#SWRST                ; 触发一个软件复位
```

C 语言代码范例:

```
ISPCR |= SWRST;                     //触发一个软件复位
```

(2). 规定功能: 使能 BOD0 复位

汇编语言代码范例:

```
MOV     IFADRL,#PCON2               ; 索引 P 页地址为 PCON2
CALL    _page_p_sfr_read            ; 读取 PCON2 数据

ORL     IFD,#BO0RE                  ; 使能 BOD0 复位功能
CALL    _page_p_sfr_write           ; 写数据到 PCON2
```

C 语言代码范例:

```
IFADRL = PCON2;                     // 索引 P 页地址为 PCON2
page_p_sfr_read();                  // 读取 PCON2 数据

IFD |= BO0RE;                       //使能 BOD0 复位功能
page_p_sfr_write();                 // 写数据到 PCON2
```

12. 电源管理

MA86E/L508 支持一个电源监测模块（掉电侦察器(BOD0)模块），和 7 种电源节能模式：空闲模式（IDLE）、掉电模式（Power-Down）、慢频模式、 副频模式、RTC 模式、Watch 模式 、 Monitor 模式。

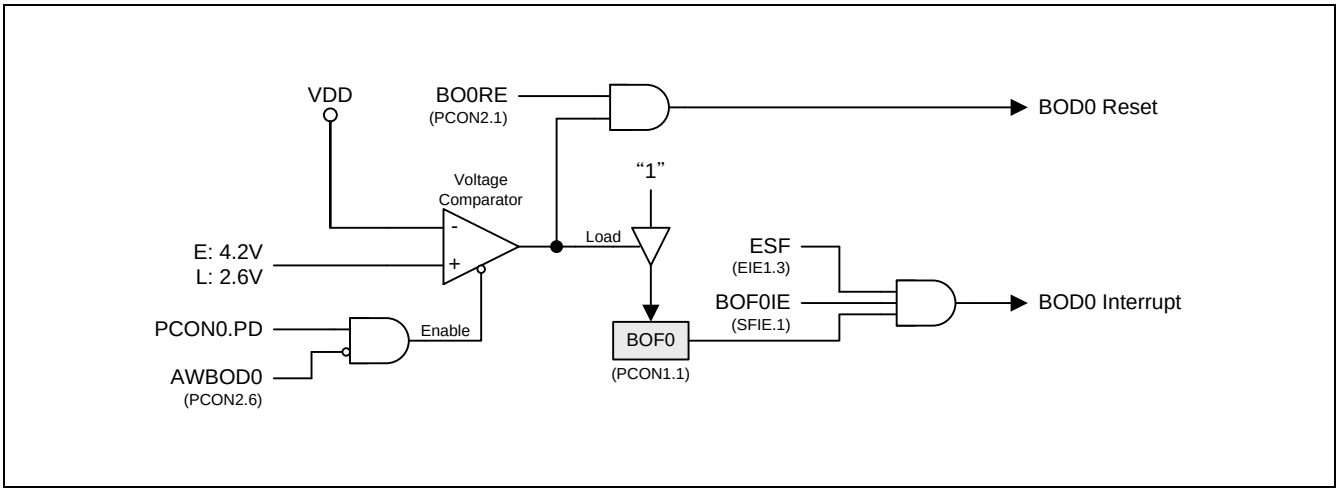
通过 BOF0 标志位 BOD0 报告电源状态 ， 软件可以通过这个状态产生中断或复位。6 种电源节能模式提供不同的节能应用， 通过对 CKCON0, CKCON2, RTCTR, PCON0, PCON1, PCON2 和 WDTCR 寄存器的访问来操作这些电源事件。

12.1. 电源监控检测器

MA86E/L508，有一个片上检测器 (BOD0)通过比较固定的触发电压来检测芯片电压， 图 12-1 是 BOD0 功能逻辑图， BOD0 检测固定触发电压为 VDD=4.2V （ **5V** 应用） 和 2.6V 在（**3.3V**） 应用。 当 VDD 降到触发电压以下时， BOF0 (PCON1.1)标志被置位， 如果 ESF (EIE1.3) 和 BOF0IE (SFIE.1) 被使能， 不管是普通模式或空闲模式都能产生一个中断请求以响应(BOD0)事件， 如果 AWBOD0 (PCON2.6)使能，这个中断也能唤醒掉电模式。

当 BO0RE (PCON2.1) 被使能， BOD0 事件产生一个系统复位并硬件置位 BOF0 指示一个 BOD0 复位事件已经产生。在普通模式和空闲模式下 BOD0 事件能重新启动 CPU，如果 AWBOD0 (PCON2.6)位被使能，也能重新启动掉电模式。

图 12–1. 电源监控检测器 0



12.2. 省电模式

12.2.1. 低速模式

程序设置位 SCKS2~SCKS0(CKCON0 寄存器, 参考系统时钟选项“8 系统时钟”)为非 0/0/0 值，可以减慢 MCU 的工作速度达到节能的目的， 使用者考量在特殊的程序段使用合适的慢速度，原则上不应该影响系统的其他功能。而且，应该在普通的程序段恢复到正常的速度。

12.2.2. 副频模式

设置 OSCS1~0 选择 OSCS1~0 作为系统时钟，MCU 的工作频率会慢下来， 64KHz ILRCO 系统频率使 MCU 工作在特别慢的速度和功耗下， 另外设置 SCKS2~SCKS0 位 (CKCON0 寄存器, 参考系统时钟选项“9 系统时钟”) 使用者可以使 MCU 的速度最低到 500HZ。

12.2.3. RTC 模式

MA86E/L508 有一个简单的 RTC 模块允许用户在设备部分掉电时继续运行准确的定时器。在 RTC 模式，RTC 模块作为一个“时钟”功能并且能在 RTC 溢出时唤醒芯片的掉电模式。详细描述请参考 10 章“实时时钟(RTC)/系统定时器”。

12.2.4. Watch 模式

如果看门狗被使能并且 NSW 位被设置 ， 看门狗在掉电模式保持运行，这个在 **MA86E/L508 应用**中叫 Watch 模式。 当 WDT 溢出， 软件选择中断或系统复位来唤醒 CPU 并硬件置位 WDTF 。 通过定义 WDT 预分频最大唤醒时间能到 2 秒， 详细描述请参考 9 章“看门狗定时器（WDT）”和 14 章“中断”。

12.2.5. Monitor 模式 (仅 L-系列)

如果 AWBOD1 (PCON3.3) 被设置， 即使在掉电模式下，掉电检测功能 BOD1 会有效，这就是 **MA86E/L508 应**用中的 Monitor 模式。 当 BOD1 触发到检测电压， 软件选择中断或系统复位来唤醒 CPU 并硬件置位 BOF1 。详细描述请参考 “电源监控检测器”和 14 章”中断” ， 这功能仅仅实用于 L-系列。

12.2.6. 空闲模式

可以通过软件的方式置 PCON.IDL 位，使设备进入空闲模式。 在空闲模式下，系统不会给 CPU 提供时钟 CPU 状态、RAM、SP、PC、PSW、ACC 被保护起来。I/O 端口也保持当前的逻辑状态。空闲模式保持外部设置当有中断来时能唤醒 CPU， 空闲模式下定时器 0、定时器 1、UART、RTC 、KBI、BOD0 仍然处于工作状态。在空闲模式下 PCA 和 WDT 唤醒 CPU 有条件制约。 任何使能的中断源或复位都能终止空闲模式， 一个中断会退出空闲模式， 并同时进入中断服务程序，只有在中断返回后才会开始执行进入空闲模式指令之后的程序。

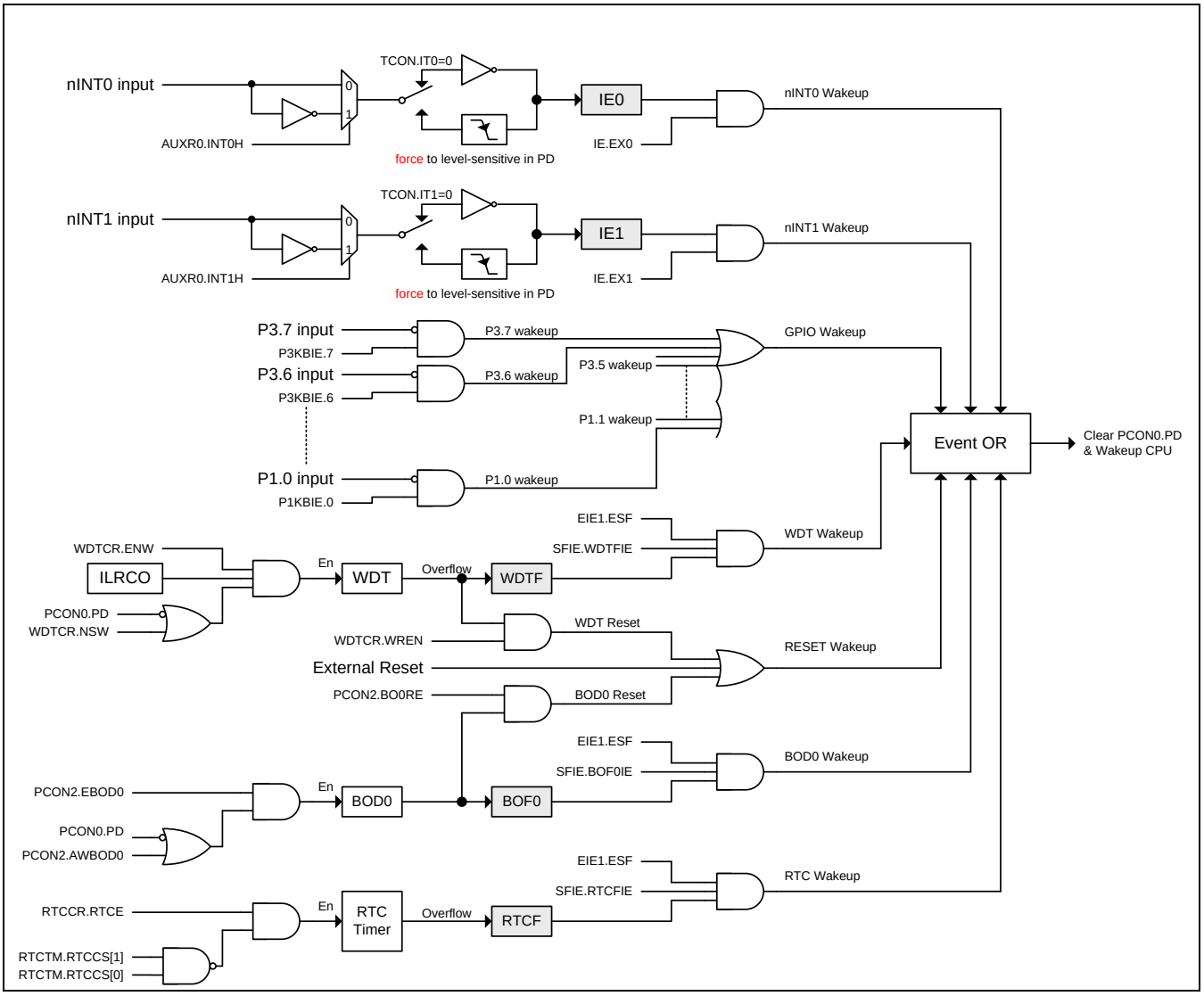
ADC 输入通道必须在 **P1AIO** 特殊功能寄存器中设置为“仅仅是模拟输入”当 MCU 在空闲模式或掉电模式

12.2.7. 掉电模式

可以通过软件的方法置位 PCON. PD 使设备进入掉电模式， 掉电模式下，震荡器停止震荡， Flash 存储器掉电以节约电能， 只有上电电路继续刷新电源，在减少 VDD 的时候 RAM 的内容仍然会被保持；但如果电源电压低于芯片工作电压，特殊功能寄存器 SFR 的内容就不一定能保持住。外部复位、上电复位、外部中断、使能的 KBI 使能的 BOD0 或 使能的没有停止的 WDT 能使系统退出掉电模式。

如果有下列情况发生，使用者至少要等 4 微秒后才能进入或再次进入掉电模式： 刚开始运行代码(任何形式的复位后面)，或者刚刚退出掉电模式。 为了在掉电模式达到最小功耗， 软件必须设置所有的 I/O 为悬浮状态， 包含封装中没有漏出来的 I/O。例如：**P2.7~P2.0 在 MA86E/L508AE20 (SOP20) 的封装中都没有秀出来， 软件必须设置 P2 (A0H) SFR 控制位为 “0” (输出低) 来避免脚位在掉电模式中处于悬浮状态。**
图 12–2 标示 **MA86E/L508** 在掉电模式中唤醒的进程。

图 12–2. 掉电模式唤醒结构



12.2.8. 中断唤醒掉电模式

两个外部中断都能终止掉电模式， 外部中断 nINT0 (P3.2), nINT1 (P3.3) 能退出掉电模式， 为了能唤醒掉电模式，中断 nINT0, nINT1 必须使能并且设置为电平触发操作，如果外部中断使能且设置是边沿触发（上升或下降），他们会被硬件强置为电平触发（低电平或高电平）。

一个中断终止掉电模式，唤醒时间取决内部定时。 当中断口产生下降沿时 ， 掉电模式被终止，震荡重新启动， 并且一个内部计数器开始计数， 在内部计数器没有计满之前内部时钟不允许被应用 CPU 也不能运行指令。 计数溢出后， 中断服务程序开始工作， 为了避免中断被重复触发， 中断服务程序在返回前应该被禁止，中断口低电平应保持足够长的时间以等待系统问题。

12.2.9. 复位唤醒掉电模式

如果 P3.6 设置为 RST 脚 ，RST 脚唤醒有点类似于中断， 复位脚有上升沿电平时系统退出掉电模式， 震荡重新启动， 且一个内部计数器开始计数， 在内部计数器没有计满之前内部时钟不允许被应用 CPU 也不能运行指令。复位脚必须保持长时间的高电平以保证系统完全复位，复位脚变低电平时开始执行程序。

值得注意的是当空闲模式被硬件复位唤醒时，前两个机器周期（内部复位没有取得控制权）程序正常从进入 IDLE 模式的后一条指令执行，这时内部硬件是禁止访问内部 RAM 的，但访问 I/O 端口没有被禁止，为了保证不可预料的写 I/O 口，在进入 IDLE 指令后不要放置写 I/O 口或外部存储器的指令（最好加两到三个 NOP 指令）。

12.2.10. KBI 键盘唤醒掉电模式

MA86E/L508 的键盘中断，P1.7 ~ P1.0, P3.7~P3.4, P41, P40, P31 和 P30 具有唤醒能力，可以通过 KBI 模块的控制寄存器 P1KBIE 、 P3KBIE ， P3.2/nINT0 、 P3.3/nINT1 进行使能。

通过使能 KBI 唤醒掉电模式有点类似中断唤醒， 当使能 KBI 的 IO 口有低电平时系统退出掉电模式， 震荡重新启动， 且一个内部计数器开始计数， 在内部计数器没有计满之前内部时钟不允许被应用 CPU 也不能运行指令。计数溢出后，CPU 响应 KBI 中断并入中断服务程序， 细节请参考 18 章“键盘中断(KBI)”。

12.2.11. 安全并且快速从 XTAL 唤醒模式

如果 MCU 运行在晶振模式并且需要进入掉电及唤醒过程，**MA86E/L508** 给系统提供一个“安全并且快速唤醒”的过程。当 MCU 在晶振模式下准备进入掉电模式时，软件使能 IHRCO 并且系统时钟（SYSCLK）切换到 IHRCO；然后 MCU 进入掉电模式并且等待唤醒事件。在唤醒触发之后，MCU 能立刻从 IHRCO 唤醒去处理系统。（一般唤醒时间大约是 **1us**，IHRCO 唤醒详情请参考 28.4“IHRCO 特性”）并且晶振振荡稳定后用软件通过 XTOR 切换。如果 XTOR 置位，系统时钟（SYSCLK）从 IHRCO 切换到晶振模式而得到精确的基准时钟工作。

12.3. 电源控制寄存器

PCON0: 电源控制寄存器 ***0***

SFR 页 = 普通及 P 页

SFR 地址 = 0x87 POR = 0001-0000, 复位值 = 000x-0000

7	6	5	4	3	2	1	0
SMOD1	SMOD0	--	POF	GF1	GF0	PD	IDL
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 4: POF0，上电标志 0

0: 这位必须由软件清零，写“1”清零。

1: 当上电复位产生时硬件置位此位。

Bit 1: PD，掉电控制位 。

0: 软件清零或任何一个退出掉电模式的事件发生时硬件清零。

1: 置位则激活掉电操作（即进入掉电模式）。

Bit 0: IDL，空闲模式控制位。

0: 软件清零或任何一个退出空闲模式的事件发生时硬件清零。

1: 置位则激活空闲操作（即进入空闲模式）。

PCON1: 电源控制寄存器 ***1***

SFR 页 =普通及 P 页

SFR 地址 = 0x97 POR = 00x0-0x00

7	6	5	4	3	2	1	0
SWRF	EXRF	--	RTCF	KBIF	--	BOF0	WDTF
R/W	**R/W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 7: SWRF，软件复位标志。

0: 这位必须由软件清零，写“1”清零。

1: 当软件复位产生时硬件置位此位。

Bit 6: EXRF，外部复位标志。

0: 这位必须由软件清零，写“1”清零。

1: 当外部复位产生时硬件置位此位。

Bit 5: 保留. 当写 PCON1 时，软件必须在这些位上写“0”

Bit 4: RTCF, RTC 溢出标志。

0: 这位必须由软件清零，写“1”清零，写“0”无操作 。

1:当 RTCCT 溢出时硬件置位此位 ， 写 **“1”** 清除 RTCF.

Bit 3: KBIF 键盘中断标志。

0: 这位必须由软件清零，写“1”清零。

1: 当键盘中断复位产生时硬件置位此位。

Bit 2:保留当写 PCON1 寄存器时此位必须填“0”。

Bit 1: BOF0, Brown-Out 侦察标志 0.

0: 这位必须由软件清零，写“1”清零。

1: 当电源监控复位产生时硬件置位此位(E: 4.2V, L: 2.4V)。

Bit 0: WDTF, WDT 溢出标志

0: 这位必须由软件清零，写“1”清零

1:当 WDT 溢出产生时硬件置位此位

PCON2: 电源控制寄存器 ***2***

SFR 页 = **仅 P 页**

SFR 地址 = 0x44 POR = x0xx-xx01

7	6	5	4	3	2	1	0
--	**AWBOD0**	--	--	--	--	BO0RE	**1**
W	**R/W**	**W**	**W**	**W**	**W**	**R/W**	**R/W**

Bit 7: 保留,写 PCON2 寄存器时此位必须填“0”。

Bit 6: AWBOD0，在掉电模式下（PD）使能电源监控模式（BOD0）控制位。

0: 电源监控模式（BOD0）在掉电模式下失效。

1: 电源监控模式（BOD0）在掉电模式下有效。

Bit 5~2: 保留,写 PCON2 寄存器时此几位必须填“0”。

Bit 1: BO0RE, BOD0 复位使能标志，初始为 OR1.BO0REO 取反值。

0: 当 BOF0 已经设置，禁止电源监控（BOD0）系统复位。

1: 当 BOF0 已经设置，使能电源监控（BOD0）系统复位（VDD 触到 4.2V(E) 或 2.4V(L)）。

Bit 0: 保留给测试用，写 PCON2 寄存器时此位必须填“1”。

P1KBIE: 端口 1 KBI 使能控制寄存器

SFR 页 = 普通

SFR 地址 = 0xD7 复位值 = 0000-0000

7	6	5	4	3	2	1	0
P17KBIE	P16KBIE	P15KBIE	P14KBIE	P13KBIE	P12KBIE	P11KBIE	P10KBIE
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~0: 每个 P1 引脚键盘输入功能使能控制位。

0: 相对应的引脚键盘输入功能失效

1: 相对应的引脚键盘输入功能使能，低电压触发。

P3KBIE: 端口 3 KBI 使能控制寄存器

SFR 页 =普通

SFR 地址 = 0xD6 复位值= 0000-0000

7	6	5	4	3	2	1	0
P37KBIE	P36KBIE	P35KBIE	P34KBIE	**P41KBIE**	**P40KBIE**	P31KBIE	P30KBIE
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~0: P3.7 ~ P3.4, P4.1, P4.0, P3.1 和 P3.0 引脚键盘输入功能使能控制位。

0: 相对应的引脚键盘输入功能禁止

1: 相对应的引脚键盘输入功能使能，低电压触发。

12.4. 电源控制示例代码

(1) 规定功能: 选择系统时钟分频为OSCin/128 的低速模式 (默认为OSCin/2)

汇编语言代码范例:

```
ORL     CKCON0,#(SCKS0 | SCKS1 | SCKS2)    ; 选择系统时钟分频为 OSCin/128

MOV     IFADRL,#DCON0                      ; 索引 P 页地址为 DCON0
CALL    _page_p_sfr_read                   ; 读取 DCON0 数据

ANL     IFD,#~(HSE)                        ; 当系统时钟 SYSCLK ≤ 6MHz 为了省电禁止 HSE
CALL    _page_p_sfr_write                  ; 写数据到 DCON0
```

C 语言代码范例:

```
CKCON0 |= (SCKS2 | SCKS1 | SCKS0);      //选择系统时钟分频为 OSCin/128.

IFADRL = DCON0;                         // 索引 P 页地址为 DCON0
page_p_sfr_read();                      // 读取 DCON0 数据.

IFD &= ~HSE;                            //当系统时钟 SYSCLK ≤ 6MHz 为了省电禁止 HSE
page_p_sfr_write();                     // 写数据到 DCON0
```

(2) 规定功能: 选择系统时钟分频为OSCin/2 的副频模式 (默认为OSCin/2=64KHz/2=32KHz)

汇编语言代码范例:

```
MOV     IFADRL,#CKCON2                  ; 索引 P 页地址为 CKCON2
CALL    _page_p_sfr_read                ; 读取 CKCON2 数据

ANL     IFD,#~(OSCS1|OSCS0)             ; OSCin 时钟源更改为 ILRCO
ORL     IFD,#OSCS1
CALL    _page_p_sfr_write               ; 写数据到 CKCON2

ANL     IFD,#~(IHRCOE|XTALE)            ; 禁止 IHRCO 和 XTAL
CALL    _page_p_sfr_write               ; 写数据到 CKCON2
```

```
MOV     IFADRL,#DCON0)          ; 索引 P 页地址为 DCON0
CALL    _page_p_sfr_read        ; 读取 DCON0 数据

ANL     IFD,#~(HSE)             ; 当系统时钟 SYSCLK ≤ 6MHz 为了省电禁止 HSE
CALL    _page_p_sfr_write       ; 写数据到 DCON0

MOV     A,CKCON0                ; 选择系统时钟为 OSCin/2
ANL     A,#~(SCKS2|SCKS1|SCKS0)
ORL     A,#SCKS0
MOV     CKCON0,A
```

C 语言代码范例:

```
IFADRL = CKCON2;                // 索引 P 页地址为 CKCON2
page_p_sfr_read();              // 读取 CKCON2 数据

IFD &= ~(OSCS1 | OSCS0);        // OSCin 时钟源更改为 ILRCO
IFD |= OSCS1;
page_p_sfr_write();             // 写数据到 CKCON2

IFD = IFD & ~(IHRCOE|XTALE);    //禁止 IHRCO 和 XTAL
page_p_sfr_write();             // 写数据到 CKCON2

IFADRL = DCON0;                 // 索引 P 页地址为 DCON0
page_p_sfr_read();              // 读取 DCON0 数据

IFD = IFD & ~(HSE);             //当系统时钟 SYSCLK ≤ 6MHz 为了省电禁止 HSE
page_p_sfr_write();             // 写数据到 DCON0

ACC = CKCON0;                   // 选择系统时钟为 OSCin/2
ACC &= ~(SCKS2 | SCKS1 | SCKS0);
ACC |= SCKS0;
CKCON0 = ACC;
```

(3). 规定功能: 现在 MCU 运行在 32.768KHz 外部振荡(XTAL)模式

汇编语言代码范例:

```
    MOV     IFADRL,#CKCON2              ; 索引 P 页地址为 CKCON2
    CALL    _page_p_sfr_read            ; 读取 CKCON2 数据

    ANL     IFD,#~(XTGS1|XTGS0)         ; 对 32.768KHz 外部振荡(XTAL)设置为低增益
    ORL     IFD,#(XTALE)                ; 使能外部振荡(XTAL)振荡
    CALL    _page_p_sfr_write           ; 写数据到 CKCON2

check_XTOR_0:                           ; 检测外部振荡(XTAL)振荡准备好
    MOV     A,AUXR1
    JNB     ACC.4,check_XTOR_0          ; 等待 XTOR(AUXR1.4)为 1

    ANL     IFD,#~(OSCS1|OSCS0)         ; OSCin 时钟源更改为外部振荡(XTAL)32.768KHz
    ORL     IFD,#OSCS0
    CALL    _page_p_sfr_write           ; 写数据到 CKCON2

    ANL     IFD,#~(IHRCOE)              ; 如果 MCU 从 IHRCO 切换过来则禁止 IHRCO
    CALL    _page_p_sfr_write           ; 写数据到 CKCON2

    MOV     IFADRL,#DCON0               ; 索引 P 页地址为 DCON0
    CALL    _page_p_sfr_read            ; 读取 DCON0 数据

    ANL     IFD,#~(HSE)                 ; 当系统时钟 SYSCLK ≤ 6MHz 为了省电禁止 HSE
    CALL    _page_p_sfr_write           ; 写数据到 DCON0

    ANL     CKCON0,#~(SCKS2|SCKS1|SCKS0) ; 系统时钟 SYSCLK = OSCin/1 = 32.768KHz
```

C 语言代码范例:

```
    IFADRL = CKCON2;                    // 索引 P 页地址为 CKCON2
    page_p_sfr_read();                  // 读取 CKCON2 数据

    IFD &= ~( XTGS1 | XTGS0 );          //对 32.768KHz 外部振荡(XTAL)设置为低增益
    IFD |= XTALE;                       //使能外部振荡(XTAL)振荡
    page_p_sfr_write();                 // 写数据到 CKCON2
```

```
while( AUXR1&XTOR == 0x00 );              //检测外部振荡(XTAL)振荡准备好
                                          //等待 XTOR(AUXR1.4)为 1

IFD &= ~(OSCS1 | OSCS0);                  // OSCin 时钟源更改为外部振荡(XTAL)32.768KHz
IFD |= OSCS0;
page_p_sfr_write ();                      // 写数据到 CKCON2

IFD &= ~IHRCOE;                           // 如果 MCU 从 IHRCO 切换过来则禁止 IHRCO
page_p_sfr_write();                       // 写数据到 CKCON2.

IFADRL = DCON0;                           // 索引 P 页地址为 DCON0
page_p_sfr_read();                        // 读取 DCON0 数据.

IFD &= ~HSE;                              //当系统时钟 SYSCLK ≤ 6MHz 为了省电禁止 HSE
page_p_sfr_write();                       // 写数据到 DCON0

CKCON0 &= ~(SCKS2 | SCKS1 | SCKS0);   // 系统时钟 SYSCLK = OSCin/1 = 32.768KHz
```

(4). 规定功能: 使能 2 秒(s)周期的 Watch 模式

汇编语言代码范例:

```
    ORG       0003Bh
SystemFlag_ISR:
    ANL       PCON1,#(WDTF)                 ; 清除 WDT 标志(写“1”)
    RETI

main:
    ANL       PCON1,#WDTF                   ; 清除 WDT 标志 (写“1”)
    ORL       WDTCR,#(NSW|ENW|PS2|PS1|PS0)
                                            ;使能 WDT 和 NSW (对 watch 模式)
                                            ;设置 PS[2:0] = 7 来选择 WDT 周期为 1.984 秒(s)

    ORL       SFIE,#WDTFIE                  ; 使能 WDT 中断
    ORL       EIE1,#ESF                     ; 使能系统标志中断
    SETB      EA                            ; 使能全局中断
```

```
    ORL     PCON0,#PD                   ; 设置 MCU 为掉电模式

;   MCU 等待唤醒
```

C 语言代码范例:

```
void SystemFlag_ISR (void) interrupt 7
{
    PCON1 &= WDTF;                      //清除 WDT 标志(写“1”)
}

viod main (void)
{
    PCON1 &= WDTF;                      //清除 WDT 标志 (写“1”)
    WDTCR |= (NSW | ENW | PS2 | PS1 | PS0);   //使能 WDT 和 NSW (对 watch 模式)
                                        //设置 PS[2:0] = 7 来选择 WDT 周期为 1.984 秒(s)

    SFIE |= WDTFIE;                     //使能 WDT 中断
    EIE1 |= ESF;                        //使能系统标志中断
    EA = 1;                             //使能全局中断

    PCON0 |= PD;                        //设置 MCU 为掉电模式

//  MCU 等待唤醒
}
```

(5). 规定功能: Monitor 模式 (仅 L-系列)

汇编语言代码范例:

```
    ORG     0003Bh
SystemFlag_ISR:
    ANL     PCON1,#(BOF0)               ; 清除 BOD0 标志(写“1”)
    RETI

main:
    MOV     IFADRL,#PCON2               ; 索引 P 页地址为 PCON2
    CALL    _page_p_sfr_read            ; 读取 PCON2 数据
```

```
    ORL     IFD,#AWBOD0                 ; 在掉电模式使能 BOD0 工作
    CALL    _page_p_sfr_write           ; 写数据到 PCON2

    ORL     SFIE,#BOF0IE                ; 使能 BOF0 中断
    ORL     EIE1,#ESF                   ; 使能系统标志中断
    SETB    EA                          ; 使能全局中断

    ORL     PCON0,#PD                   ; 设置 MCU 为掉电模式

;   MCU 等待唤醒
```

C 语言代码范例:

```
void SystemFlag_ISR() interrupt 7
{
    PCON1 &= BOF0;                      // 清除 BOD0 标志(写“1”)
}

void main()
{
    IFADRL = PCON2;                     // 索引 P 页地址为 PCON2
    page_p_sfr_read();                  // 读取 PCON2 数据

    IFD |= AWBOD0;                      //在掉电模式使能 BOD0 工作
    page_p_sfr_write();                 // 写数据到 PCON2


    SFIE |= BOF0IE;                     //使能 BOF0 中断
    EIE1 |= ESF;                        //使能系统标志中断
    EA = 1;                             //使能全局中断

    PCON0 |= PD;                        //设置 MCU 为掉电模式

//  MCU 等待唤醒
}
```

(6). 规定功能:外部晶振 (XTAL)在掉电模式下安全并且快速唤醒

汇编语言代码范例:

```
    MOV     IFADRL,#CKCON2          ; 索引 P 页地址为 CKCON2
    CALL    _page_p_sfr_read        ; 读取 CKCON2 数据

    ORL     IFD,#IHRCOE             ; 使能 IHRCO
    CALL    _page_p_sfr_write       ; 写数据到 CKCON2

    Delay_32us

    ANL     IFD,#~(OSCS1|OSCS0)     ; OSCin 时钟源更改到 IHRCO
    CALL    _page_p_sfr_write       ; 写数据到 CKCON2

    ORL     PCON0,#PD               ; 设置 MCU 为掉电

;   MCU 等待唤醒

check_XTOR:                         ; 检测外部振荡(XTAL)振荡准备好
    MOV     A,AUXR1
    JNB     ACC.4,check_XTOR        ; 等待 XTOR(AUXR1.4)为 1

    ANL     IFD,#~(OSCS1 | OSCS0)   ; OSCin 时钟源更改为外部振荡(XTAL)
    ORL     IFD,#(OSCS0)
    CALL    _page_p_sfr_write       ; 写数据到 CKCON2

    ANL     IFD,#~(IHRCOE)          ; 如果 MCU 从 IHRCO 切换过来则禁止 IHRCO
    CALL    _page_p_sfr_write       ; 写数据到 CKCON2
```

C 语言代码范例:

```
    IFADRL = CKCON2;                // 索引 P 页地址为 CKCON2
    page_p_sfr_read();              // 读取 CKCON2 数据

    IFD |= IHRCOE;                  // 使能 IHRCO
    page_p_sfr_write();             // 写数据到 CKCON2
```

```
    Delay_32us();

    IFD &= ~(OSCS1 | OSCS0);                // OSCin 时钟源更改到 IHRCO
    page_p_sfr_write();                     //  写数据到 CKCON2

    PCON0 |= PD;                            //设置 MCU 为掉电

//  MCU 等待唤醒

    while(AUXR1 & XTOR == 0x00);            //检测外部振荡(XTAL)振荡准备好
                                            //等待 XTOR(AUXR1.4)为 1

    IFD &= ~(OSCS1 | OSCS0);                // OSCin 时钟源更改为外部振荡(XTAL)
    IFD |= OSCS0;
    page_p_sfr_write ();                    //  写数据到 CKCON2

    IFD &= ~IHRCOE;                         //如果 MCU 从 IHRCO 切换过来则禁止 IHRCO
    page_p_sfr_write();                     //  写数据到 CKCON2.
```

13. 输入输出口配置

MA86E/L508 有下列 I/O 端口：P1.0~P1.7, P2.0~P2.7, P3.0~P3.7 及 P4.0~P4.1。RST 引脚与 P3.6 有互换功能。如果选择外部振荡器做系统时钟输入则 P4.0 和 P4.1 被配置为 XTAL2 和 XTAL1。准确的可用 I/O 引脚数量由封装类型决定。见表 13-1。

表 13–1. 可用引脚数量

封装类型	I/O 引脚	引脚数量
28-pin SOP	P1.0~P1.7, P2.0~P2.7, P3.0~P3.5, P3.7, P3.6(RST), P4.0 (ECKI/XTAL2), P4.1 (XTAL1)	26 or 25 (RST selected) or 24 (RST & ECKI selected) or 23 (RST & XTAL selected)
20-pin SOP	P1.0~P1.7, P3.0~P3.5, P3.7, P3.6(RST), P4.0 (ECKI/XTAL2), P4.1 (XTAL1)	18 or 17 (RST selected) or 16 (RST & ECKI selected) or 15 (RST & XTAL selected)
16-pin SOP	P1.0~P1.7, P3.0~P3.3, P3.6(RST), P4.0 (ECKI/XTAL2), P4.1 (XTAL1)	14 or 13 (RST selected) or 12 (RST & ECKI selected) or 11 (RST & XTAL selected)

13.1. IO 结构

MA86E/L508 输入输出口分成两个配置类型。第一类仅仅是端口 3 有四种模式，这四种模式有：准双向口(标准 8051 的 I/O 端口)、推挽输出、漏极开路输出和输入(高阻抗输入)。

其它口属于第二类，这些口有两种模式分别是推挽输出和上拉电阻的漏极开路输出。默认设置是开漏输出高，意味着是高阻状态的输入模式。

下面描述这四种类型的 I/O 模式的配置。

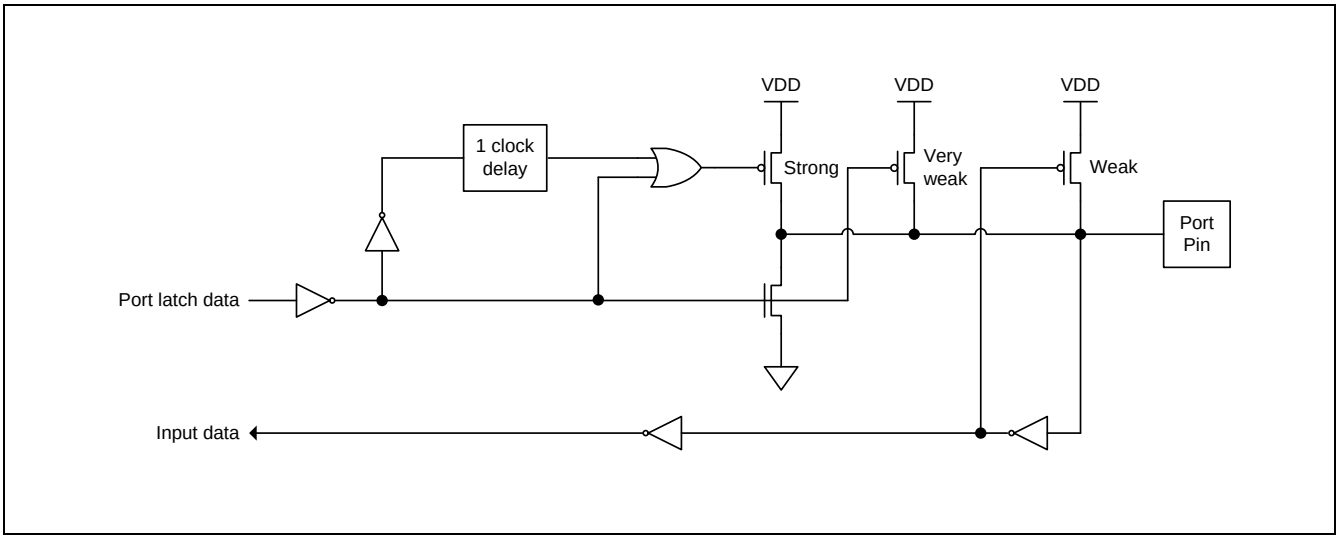
13.1.1. 端口 3 准双向口结构

端口 3 引脚工作在准双向模式时与标准 8051 端口引脚类似。一个准双向端口用作输入和输出时不需要对端口重新配置。这是因为端口输出逻辑高时，弱上拉，允许外部器件拉低引脚。当输出低时，强的驱动能力可吸收大电流。在准双向输出时有三个上拉晶体管用于不同的目的。

其中的一种上拉，称为微上拉，只要端口寄存器的引脚包含逻辑 1 则打开。如果引脚悬空，则这种非常弱上拉提供一个非常小的电流将引脚拉高。第二种上拉称为“弱上拉”，端口寄存器的引脚包含逻辑 1 时且引脚自身也在逻辑电平时打开。这种上拉对准双向引脚提供主要的电流源输出为 1。如果引脚被外部器件拉低，这个弱上拉关闭，只剩一个微上拉。为了在这种条件下将引脚拉低，外部器件不得不吸收超过弱上拉功率的电流，且拉低引脚在输入的极限电压之下。第三种上拉称为“强”上拉。这种上拉用于加速准双向端口的上升沿跳变，当端口寄存器发生从逻辑 0 到逻辑 1 跳变时，强上拉打开一个 CPU 时钟，快速将端口引脚拉高。

准双向端口 3 配置如图 13-1 所示。

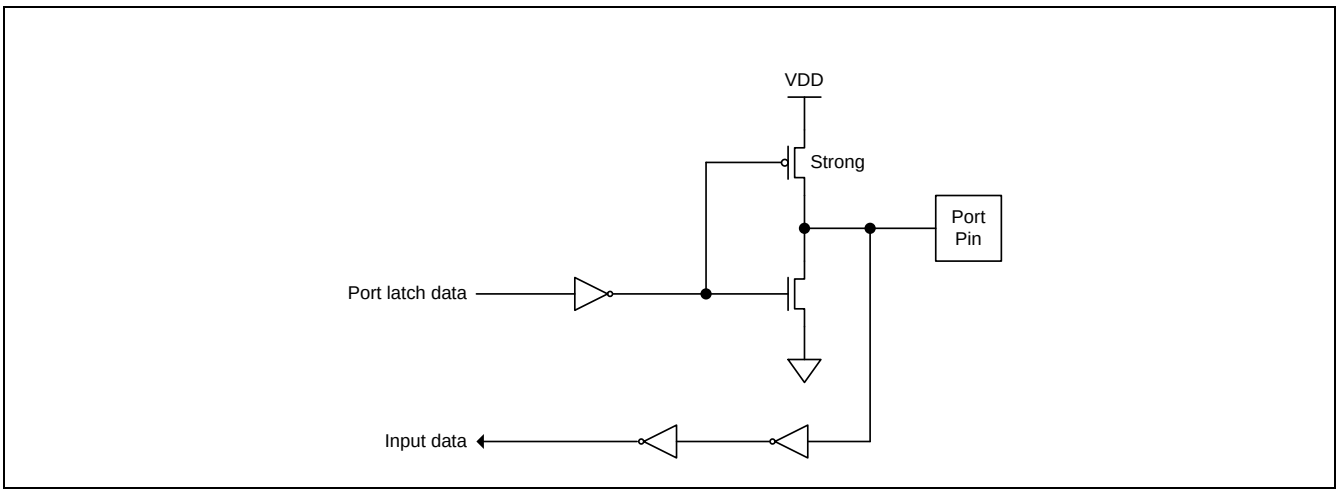
图 13–1. 端口 3 准双向口



13.1.2. 端口 3 推挽输出结构

端口 3 推挽输出配置与开漏输出、准双向输出模式有着相同的下拉结构，但是当端口寄存器包含逻辑 1 时提供一个连续的强上拉。当一个端口输出需要更大的电流时可配置为推挽输出模式。另外，在这种配置下端口的输入路径与准双向模式相同。 端口 3 推挽输出配置见 13-2。

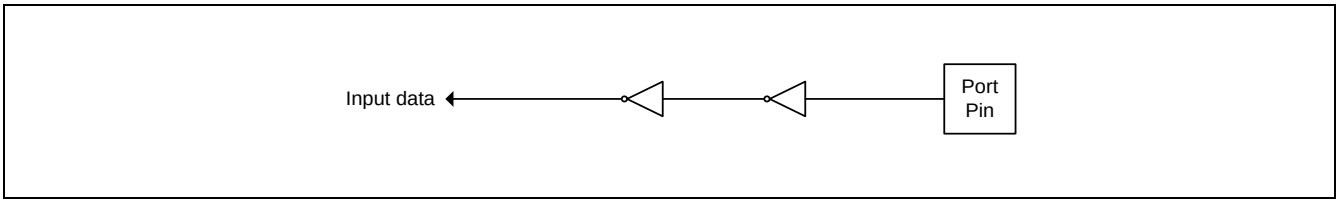
图 13–2. P 端口 3 推挽输出



13.1.3. 端口 3 仅是输入（高阻抗输入）模式

仅输入配置在引脚上没有任何上拉电阻，如下图 13-3 所示。

图 13–3. 端口 3 仅输入模式

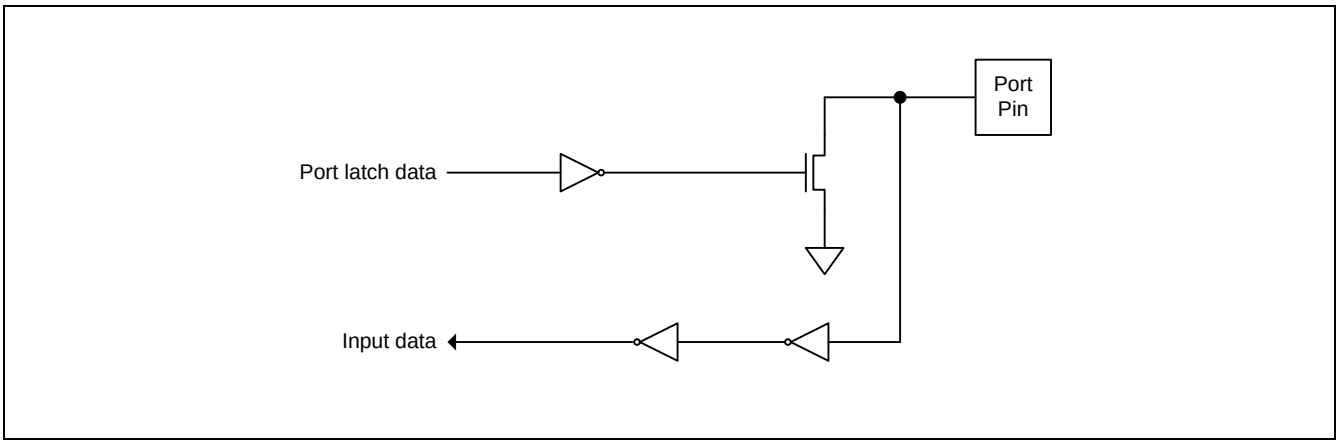


13.1.4. 端口 3 开漏输出结构

端口 3 配置为开漏输出时，当端口寄存器包含逻辑 0 时，关闭所有上拉，只有端口引脚的下拉晶体管。在应用中使用这个配置，端口引脚必须有外部上拉，典型的是将电阻接到 VDD。这个模式的下拉和准双向端口的模式相同。另外，在这种配置下端口的输入路径与准双向模式相同。

开漏输出端口 3 配置如图 13-4 所示。

图 13–4. 端口 3 开漏输出

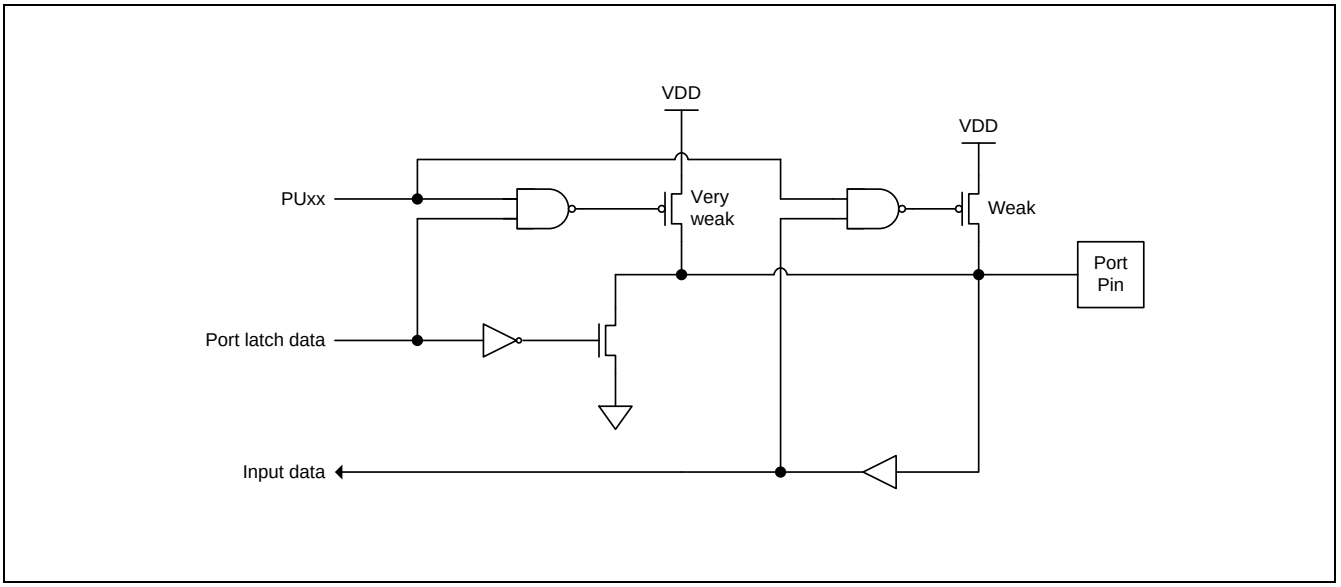


13.1.5. 通用端口漏极开路输出结构

当端口数据寄存器写”0”时，通用端口的漏极开路输出仅是驱动端口的下拉晶体管。在应用中使用这个配置，端口引脚可以选择外部上拉或 PUCON0 置位使能片内的内部上拉。

通用端口漏极开路结构如图 13-5 所示。

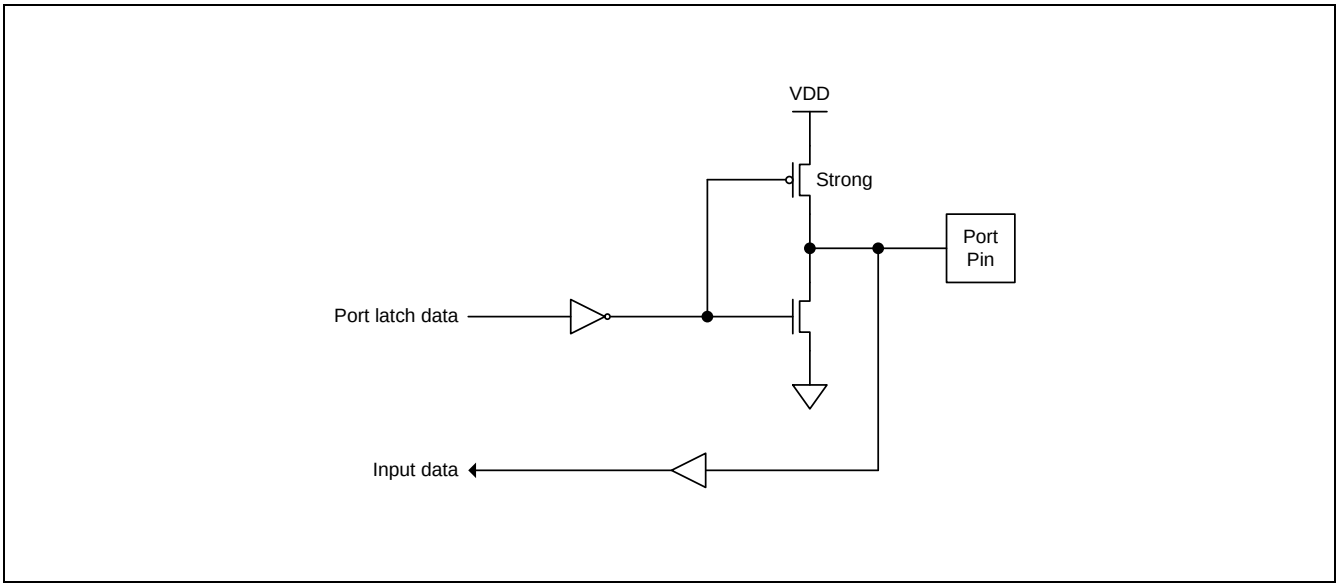
图 13–5. 通用端口漏极开路输出



13.1.6. 通用端口推挽输出结构

通用端口推挽输出结构与漏极开路输出有相同的下拉结构，但是端口数据寄存器写”1”提供一个强上拉输出。通用端口推挽输出模式应用在需要强的源电流输出。另外，此结构下的端口输入与漏极开路输出模式一样。
通用端口推挽输出结构如图 13-6 所示。

图 13–6. 通用端口推挽输出



13.1.7. 通用端口输入结构

端口引脚在设置为漏极开路模式并且端口数据寄存器写“1”的情况下可以作为输入口。例如，P1M0.7=0 及 P1.7 写入”1”，这样 P1.7 就定义为输入口。

13.2. 输入输出口寄存器

MA86E/L508 所有的端口都是独立的配置通过软件选择其工作模式， 唯有端口 3 可配置为四种之中的一种工作模式，如表 13-2 所示。每个 3 端口都有两个模式寄存器来选择各端口引脚的输出类型。

表 13–2. 端口 3 配置设定

P3M0.y	P3M1.y	端口模式
0	0	准双向
0	1	推挽输出
1	0	输入口 (高阻抗输入)
1	1	漏极开路输出

这里 y=0~7(端口引脚号)。寄存器 P3M0 和 P3M1 列举了每个引脚的描述。

其它的通用口引脚有两种模式见表 14-3，一个模式寄存器位选择每个引脚的输出类型。

表 13–3. 端口配置设定

PxM0.y	端口模式
0	漏极开路输出
1	推挽输出

这里 x=1, 2, 4 (端口)，y=0~7(端口引脚号)。寄存器 P1M0 和 P4M0 列举了每个引脚的描述。

13.2.1. 端口 1 寄存器

P1: 端口 1 寄存器

SFR 页 = 普通

SFR 地址 = 0x90 复位值 = 1111-1111

7	6	5	4	3	2	1	0
P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~0: P1.7~P1.0 通过软件置位/清零 。

P1M0: 端口 1 模式寄存器 0

SFR 页 =普通

SFR 地址 = 0x91 复位值= 0000-0000

7	6	5	4	3	2	1	0
P1M0.7	P1M0.6	P1M0.5	P1M0.4	P1M0.3	P1M0.2	P1M0.1	P1M0.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

0: 端口定义为漏极开路输出。

1: 端口定义为推挽输出。

P1AIO: 端口 1 仅仅模拟输入

SFR 页 =普通

SFR 地址 = 0x92 复位值= 0000-0000

7	6	5	4	3	2	1	0
P17AIO	**P16AIO**	**P15AIO**	**P14AIO**	**P13AIO**	**P12AIO**	**P11AIO**	**P10AIO**
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

0: 端口有数字和模拟输入的能力。

1: 端口仅仅只有模拟输入的功能给 ADC 输入应用。当此位被设置相应的端口 PIN 会永远读到 “0”。

13.2.2. 端口 2 寄存器

P2: 端口 2 寄存器

SFR 页 =普通

SFR 地址 = 0xA0 复位值= 1111-1111

7	6	5	4	3	2	1	0
P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~0: P2.7~P2.0 通过软件置位/清零。

P2M0: 端口 2 模式寄存器 0

SFR 页 =普通

SFR 地址 = 0x95 复位值= 0000-0000

7	6	5	4	3	2	1	0
P2M0.7	P2M0.6	P2M0.5	P2M0.4	P2M0.3	P2M0.2	P2M0.1	P2M0.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

0: 端口定义为漏极开路输出。

1: 端口定义为推挽输出。

13.2.3. 端口 3 寄存器

P3: 端口 3 寄存器

SFR 页 =普通

SFR 地址 = 0xB0 复位值= 1111-1111

7	6	5	4	3	2	1	0
P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~0: P3.7~P3.0 能被 CPU 置位/清零 。

P3M0: 端口 3 模式寄存器 0

SFR 页 =普通

SFR 地址 = 0xB1 复位值= 0000-0000

7	6	5	4	3	2	1	0
P3M0.7	P3M0.6	P3M0.5	P3M0.4	P3M0.3	P3M0.2	P3M0.1	P3M0.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

P3M1: 端口 3 模式寄存器 1

SFR 页 =普通

SFR 地址 = 0xB2 复位值= 0000-0000

7	6	5	4	3	2	1	0
P3M1.7	P3M1.6	P3M1.5	P3M1.4	P3M1.3	P3M1.2	P3M1.1	P3M1.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

13.2.4. 端口 4 寄存器

P4: 端口 4 寄存器

SFR 页 =普通

SFR 地址 = 0xE8 复位值= xxxx-xx11

7	6	5	4	3	2	1	0
--	--	--	--	--	--	P4.1	P4.0
W	**W**	**W**	**W**	**W**	**W**	**R/W**	**R/W**

Bit 7~2: 保留.

Bit 1~0: P4.1~P4.0 通过软件置位/清零。当内部振荡器被使能做系统时钟时这两个 I/O 被激活，这样 XTAL1 和 XTAL2 成为 P4.1 和 P4.0。

P4M0: 端口 4 模式寄存器 0

SFR 页 =普通

SFR 地址 = 0xB3 复位值= xxxx-xx00

7	6	5	4	3	2	1	0
--	--	--	--	--	--	P4M0.1	P4M0.0
W	**W**	**W**	**W**	**W**	**W**	**R/W**	**R/W**

0: 端口定义为漏极开路输出。

1: 端口定义为推挽输出。

13.2.5. 上拉控制寄存器

PUCON0: 端口上拉控制寄存器 0

SFR 页 =普通

SFR 地址 = 0xB4 复位值= x000-00xx

7	6	5	4	3	2	1	0
--	PU40	P2PU1	P2PU0	PU11	PU10	--	--
W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**W**	**W**

Bit 7: 保留位。当 PUCON0 改写时，此位必须软件写”0”。

Bit 6: 端口 4 低四位上拉使能控制

0: 在漏极开路输出模式禁止 P4.0 和 P4.1 上拉。

1: 在漏极开路输出模式使能 P4.0 和 P4.1 上拉。

如果在 P4 推挽模式设置此位，在相应的推挽使能脚设置此位无效，因为 GPIO 设计，没有相关的控制逻辑。

Bit 5: 端口 2 高四位上拉使能控制

0: 在漏极开路输出模式禁止 P2.7 ~ P2.4 上拉。

1: 在漏极开路输出模式使能 P2.7 ~ P2.4 上拉。

Bit 4: 端口 2 低四位上拉使能控制

0: 在漏极开路输出模式禁止 P2.3 ~ P2.0 上拉。

1: 在漏极开路输出模式使能 P2.3 ~ P2.0 上拉。

Bit 3: 端口 1 高四位上拉使能控制

0: 在漏极开路输出模式禁止 P1.7~P1.4 上拉。

1: 在漏极开路输出模式使能 P1.7~P1.4 上拉。

如果在 P1 推挽模式设置此位，在相应的推挽使能脚设置此位无效，因为 GPIO 设计，没有相关的控制逻辑。

Bit 2: 端口 1 低四位上拉使能控制

0: 在漏极开路输出模式禁止 P1.3~P1.0 上拉。

1: 在漏极开路输出模式使能 P1.3~P1.0 上拉。

如果在 P1 推挽模式设置此位，在相应的推挽使能脚设置此位无效，因为 GPIO 设计，没有相关的控制逻辑。

Bit 1~0: 保留位。当 PUCON0 改写时，这两位必须软件写”0”。

13.3. GPIO 示例代码

(1). 规定功能: 设置 P1.0 为片内上拉电阻使能的输入模式

汇编语言代码范例:

```
ANL     P1M0,#~P1M00            ; 配置 P1.0 为漏极开路模式
SETB    P10                     ; 设置 P1.0 数据为“1”而使能输入模式
ORL     PUCON0,#PU10            ; 使能 P1.3~P1.0 片内上拉电阻
```

C 语言代码范例:

```
P1M0 &= P1M00;                  //配置 P1.0 为漏极开路模式
P10 = 1;                        //设置 P1.0 数据为“1”而使能输入模式
PUCON0 |= PU10;                 //使能 P1.3~P1.0 片内上拉电阻
```

(2). 规定功能: 选择 RST 引脚为 P3.6

汇编语言代码范例:

```
MOV     IFADRL,#DCON0           ; 索引 P 页地址为 DCON0
CALL    _page_p_sfr_read        ; 读取 DCON0 数据

ANL     IFD,#~(RSTIO)           ; 选择 I/O 功能为 P36
CALL    _page_p_sfr_write       ; 写数据到 DCON0
```

C 语言代码范例:

```
IFADRL = DCON0;                 // 索引 P 页地址为 DCON0
page_p_sfr_read();              // 读取 DCON0 数据.

IFD &= ~RSTIO;                  //选择 I/O 功能为 P36
page_p_sfr_write();             // 写数据到 DCON0
```

14. 中断

MA86E/L508 有四级中断优先级的 8 个中断源。与这四级中断有关联的特殊功能寄存器有 IE、IP0L、IP0H、EIE1、EIP1L 及 EIP1H。IP0H (中断优先级 0 高位)和 EIP1H (外部中断优先级 1 高位) 寄存器设置四级中断优先级。四级中断优先级结构为中断源的应用提供了很大的便利。

14.1. 中断结构

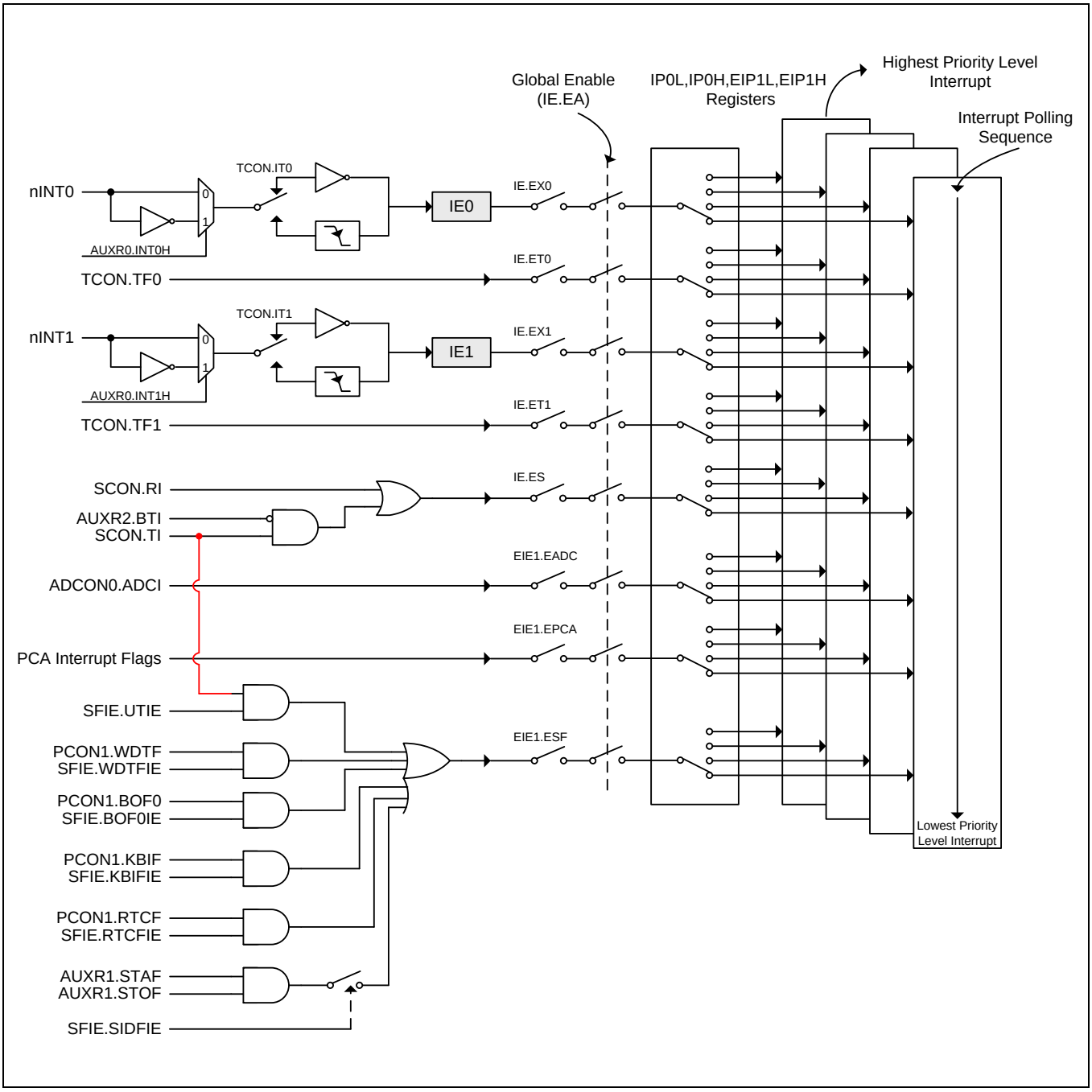
表 14-1列出了所有的中断源。使能位被允许，中断请求时硬件会产生一个中断请求标志，当然，总中断使能位EA (IE 寄存器)必须使能。中断请求位能由软件置1或清零，这和硬件置1或清零结果相同。同理，中断可以由软件产生或取消，中断优先级位决定每个中断产生的优先级，多个中断同时产生时依照中断优先级顺序处理。中断向量地址表示中断服务程序的入口地址。

图 14-1 展示了整个中断系统。每一个中断将在下面部分做简单的描述。

表 14–1. 中断源

序号	中断源	使能位	请求位	优先级位	优先级顺序	中断向量地址
#1	外部中断 0, nINT0	EX0	IE0	[PX0H, PX0L]	(Highest)	0003H
#2	定时器 0	ET0	TF0	[PT0H, PT0L]	…	000Bh
#3	外部中断 1, nINT1	EX1	IE1	[PX1H, PX1L]	…	0013H
#4	定时器 1	ET1	TF1	[PT1H, PT1L]	…	001BH
#5	串行口	ES	RI, TI	[PSH, PSL]	…	0023H
#6	ADC	EADC	ADCI	[PADCH, PADCL]	…	002BH
#7	PCA	EPCA	CF, CCFn (n=0~1)	[PPCAH, PPCAL]	…	0033H
#8	系统标志	ESF	BOF0,WDTF KBIF,RTCF STAF,STOF (TI)	[PSFH, PSFL]	(Lowest)	003BH

图 14–1. 中断系统


Global Enable
(IE.EA)
IP0L,IP0H,EIP1L,EIP1H
Registers
Highest Priority Level
Interrupt
Interrupt Polling
Sequence
nINT0
TCON.IT0
IE0
IE.EX0
AUXR0.INT0H
TCON.TF0
IE.ET0
nINT1
TCON.IT1
IE1
IE.EX1
AUXR0.INT1H
TCON.TF1
IE.ET1
SCON.RI
AUXR2.BTI
SCON.TI
IE.ES
ADCON0.ADCI
EIE1.EADC
PCA Interrupt Flags
EIE1.EPCA
SFIE.UTIE
PCON1.WDTF
SFIE.WDTFIE
EIE1.ESF
PCON1.BOF0
SFIE.BOF0IE
PCON1.KBIF
SFIE.KBIFIE
PCON1.RTCF
SFIE.RTCFIE
AUXR1.STAF
AUXR1.STOF
SFIE.SIDFIE
Lowest Priority
Level Interrupt

14.2. 中断源

表 14–2. 中断标志

序号	中断源名称	中断请求位	位位置
#1	外部中断 0, nINT0	IE0	TCON.1
#2	定时器 0	TF0	TCON.5
#3	外部中断, nINT1	IE1	TCON.3
#4	定时器 1	TF1	TCON.7
#5	串行口 0	RI0 TI0	SCON0.0 SCON0.1
#6	ADC	ADCI	ADCON0.4
#7	PCA	CF, CCFn (n=0~1)	CCON.7 CCON.1~0
#8	系统标志	WDTF BOF0 KBIF RTCF STAF STOF (TI)	PCON1.0 PCON1.1 PCON1.3 PCON1.4 AUXR1.3 AUXR1.2 SCON.1

外部中断 nINT0 和 nINT1 分别通过 TCON 的 IT0 和 IT1 可以设置成电平触发或边沿触发。实际产生的中断标志位是 TCON 的 IE0 和 IE1。产生外部中断时，如果是边沿触发，进入中断服务程序后由硬件清除中断标志位，如果中断是电平触发，由外部请求源而不是由片内硬件控制请求标志。

定时 0 和定时器 1 中断由 TF0 和 TF1 产生，在大多数情况下，由各自的定时器/计数器翻转而置位。当产生定时器中断时，进入中断服务程序后由片内硬件清除标志位。

串口中断由 RI 和 TI 的逻辑产生。进入中断服务程序后，这些标志均不会被硬件清除。实际上，中断服务程序通常需要确定是由 R I 还是 TI 产生的中断，然后由软件清除中断标志。

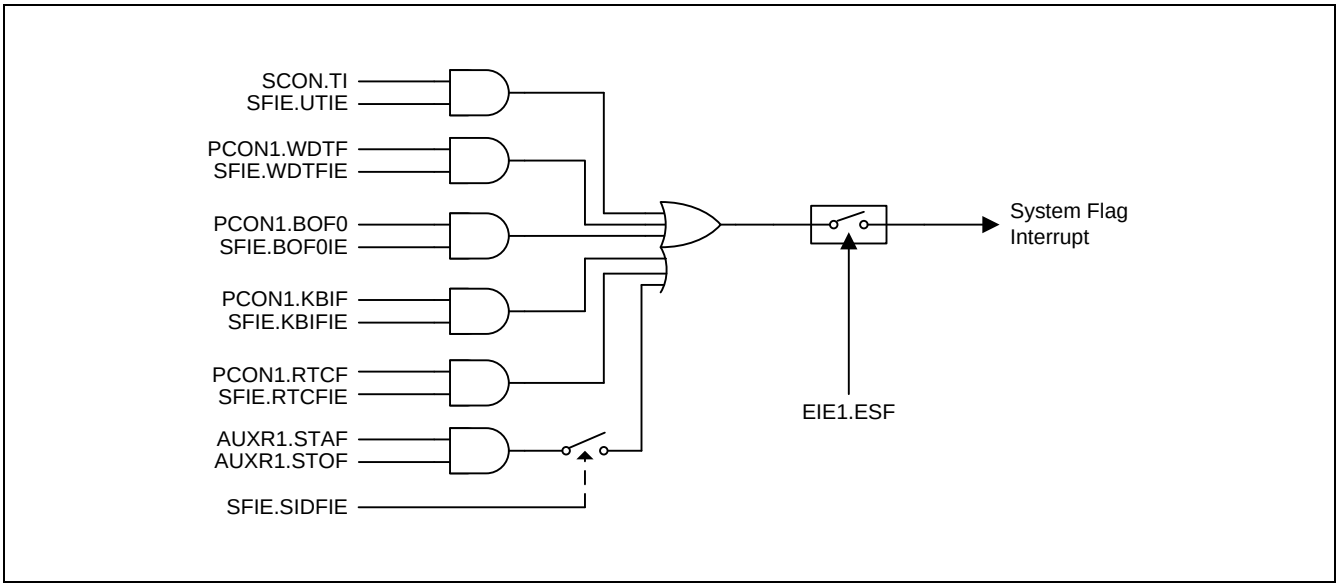
ADC 中断由 ADCON0 中 的 ADCI 产生。进入中断服务程序后，这些标志均不会被硬件清除。

PCA 中断由 CCON 中的 CF, CCF1 和 CCF0 产生，进入中断服务程序后，这些标志均不会被硬件清除。中断服务程序应该去查看这些标志响应相应的请求服务，然后由软件清除中断标志。

系统标志中断由 STAF, STOF, RTCF, **KBIF, BOF0** 和 **WDTF** 产生。STAF 和 STOF 由串口侦测置位并且保存在 AUXR1。其它标志位则保存在 PCON1。 RTCF 由 RTC 计数器溢出置位。 KBIF 由 KBI 事件置位。BOF0 在检测

到低电压时置位。WDTF 看门狗溢出置位。串行口 TI 标志可通过 UTIE 置位来选择到与系统标志中断一样的中断向量。这些标志位进入中断服务程序后不会由片内硬件清除标志位。

图 14–2. 系统标志中断配置



所有产生中断的位可以由软件置位或清零，和硬件置位或清零产生同样的结果。换句话说，中断可以在软件里产生，或者挂起的中断可以在软件里取消。

14.3. 中断使能

表 14–3. 中断使能控制

序号	中断源名称	使能位	位位置
#1	外部中断 0, nINT0	EX0	IE.0
#2	定时器 0	ET0	IE.1
#3	外部中断, nINT1	EX1	IE.2
#4	定时器 1	ET1	IE.3
#5	串行口 0	ES0	IE.4
#6	ADC	EADC	EIE1.1
#7	PCA	EPCA	EIE1.2
#8	系统标志	ESF & (UTIE, SDIFIE, RTCFIE, KBIFIE, BOF0IE, WDTFIE)	EIE1.3 & SFIE.7,6,4,3,1,0

MA86E/L508 有 8 个可用的中断源。通过设置寄存器 IE 和 EIE1 能对每个中断进行使能和禁止操作。IE 也包括了一个全局使能位，EA，清它可以立刻禁止所有中断。如果 EA 置“1”， 中断单独的被相应的使能位使能或禁止。EA 清“0”，所有中断被禁止。

14.4. 中断优先级

中断服务的优先级组合和 80C51 是一样的，除了有 4 个优先级比 80C51 多两个外。优先级设置位(见表 14-1) 决定每个中断的优先级别。IP0L, IP0H, EIP1L 和 EIP1H 组合成 4 级优先级中断。表 14–4 列出了设置位的值和每个组合对应的优先级

表 14–4. 中断优先级

{IPH.x , IPL.x}	优先级
11	1 (highest)
10	2
01	3
00	4

每一个中断源都有两个相对应的位表示它的优先级。一个在 IPnH 寄存器 ，另一个在 IPnL 寄存器。高优先级中断不会被低优先级中断打断。如果两个如果同时收到两个不同优先级的中断请求，高优先级的请求将被处理。如果同时收到同样优先级的请求，内部轮询顺序将决定哪个请求被处理。表 14-2 列出了同优先级的内部轮询顺序和中断向量地址。

14.5. 中断处理

每个系统时钟周期都会采样中断标志位。在下一个系统时钟，这采样值被轮询。如果在第一个周期其中一个标志被置位，那么第二个周期（轮询周期）就会发现它，并且中断系统会产生一个硬件 LCALL 调用相应的的中断服务程序，只要没有被下列的条件给阻止：

阻止条件:

- 一个同等或高优先级的中断正在处理。
- 当前周期(轮询周期)不是正在执行的指令的最后一个周期。
- 正在执行指令 RETI 或正在写和中断相关的寄存器(IE, IP0L, IPH, EIE1, EIP1L 及 EIP1H 寄存器)。

上述三种情况将锁定 LCALL 指令使之暂时不能访问中断服务程序；第二种情况在引导任何中断服务程序之前必须执行完；第三种情况保证 RETI 的执行或写中断相关的寄存器（如 IE 或 IP 等）能顺利完成，在中断进入引导程序之前至少运行一个以上的指令周期。

14.6. TI 的特别中断向量

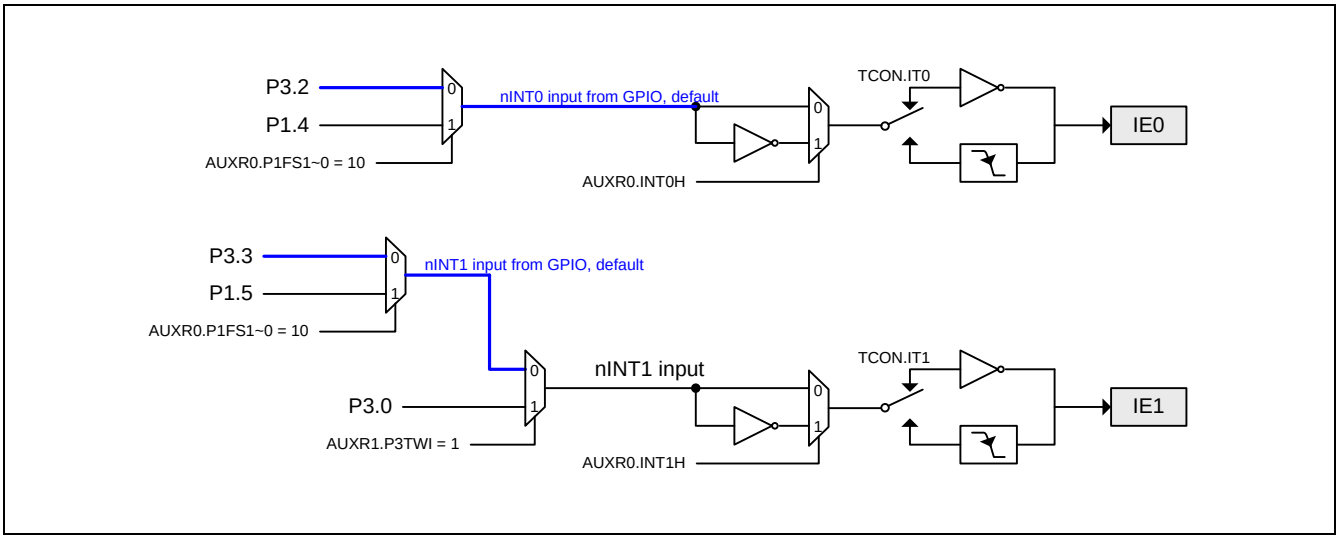
TI 标志位的串行口中断可以通过 BTI (AUXR2.6)来屏蔽。如果 BTI 置位，则 TI 置位也不会产生串行口中断，这样串行口中断仅仅是 RI 标志位中断。

如果 UTIE (SFIE.7)置位，TI 标志位会产生系统标志中断。在此模式下，TI 中断与 KBIF, BOF0 和 WDTF 一起共用系统标志中断向量。

14.7. nINT0/nINT1 输入源选择

MA86E/L508 提供灵活的 nINT0 和 nINT1 源选择，和在芯片上的串行接口共享端口输入。这使得在掉电模式下支持额外的用外部通信设备远程唤醒的功能。nINT0/nINT1 输入可以被选择为捕捉端口变化的接口引脚，并且设置它们作为一个中断输入事件去唤醒 MCU。用 INT0H(AUXR0.0)和 INT1H(AUXR0.1)可以配置端口改变监测低电平/下将沿或高电平/上升沿。 在掉电模式下，外部中断的配置不管是下降沿还是上升沿，都会强迫为电平触发。

图 14–3. nINT0/nINT1 端口引脚选择配置



14.8. 中断寄存器

TCON: 定时器/计数器控制寄存器

SFR 页 = 普通

SFR 地址 = 0x88 复位值 = 0000-0000

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 3: IE1, 外部中断 1 请求标志。

0: 如果是边沿触发的中断则在进入中断向量后硬件清零。

1: 外部中断 1 由边沿或电平触发（由 IT1 设置）硬件置标志。

Bit 2: IT1: 外部中断 1 类型控制位

0: 软件选择低电平触发外部中断 0。如果 INT1H (AUXR0.1)置位，则 nINT1 高电平触发外部中断 0。

1: 软件选择下降沿触发外部中断 0。如果 INT1H (AUXR0.1)置位，则 nINT1 上升沿触发外部中断 0。

Bit 1: IE0, 外部中断 0 请求标志。

0: 如果是边沿触发的中断则在进入中断向量后硬件清零。

1: 外部中断 0 由边沿或电平触发（由 IT0 设置）硬件置标志。

Bit 0: IT0: 外部中断 0 类型控制位

0: 软件选择低电平触发外部中断 0。如果 INT0H (AUXR0.0)置位，则 nINT0 高电平触发外部中断 0。

1: 软件选择下降沿触发外部中断 0。如果 INT0H (AUXR0.0)置位，则 nINT0 上升沿触发外部中断 0。

IE: 中断使能寄存器

SFR 页 =普通

SFR 地址 = 0xA8 复位值= 0xx0-0000

7	6	5	4	3	2	1	0
EA	--	--	ES	ET1	EX1	ET0	EX0
R/W	**W**	**W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: EA, 总中断使能位

0: 全局禁止所有中断。

1: 全局使能所有中断

Bit 6- Bit 5 : 保留。当写 IE 时，软件必须在这些位上写”0”

Bit 4: ES, 串口 0 中断使能

0: 禁止串口 0 中断

1: 使能串口 0 中断

Bit 3: ET1, 定时器 1 中断使能

0: 禁止定时器 1 中断

1: 使能定时器 1 中断

Bit 2: EX1, 外部中断 1 中断使能

0: 禁止外部中断 1

1: 使能外部中断 1

Bit 1: ET0, 定时器 0 中断使能

0: 禁止定时器 0 中断

1: 使能定时器 0 中断

Bit 0: EX0, 外部中断 0 中断使能

0: 禁止外部中断 0

1: 使能外部中断 0

EIE1: 扩展中断使能 1 寄存器

SFR 页 =普通

SFR 地址 = 0xAD 复位值= xxxx-0xxx

7	6	5	4	3	2	1	0
--	--	--	--	ESF	EPCA	EADC	--
W	**W**	**W**	**W**	**R/W**	**R/W**	**R/W**	**W**

Bit 7~4: 保留。当写 EIE1 时，软件必须在这些位上写”0”。

Bit 3: ESF,系统标志中断使能

0: 当 PCON1 的 KBIF,BOF0,WDTF 或 SCON 的 TI 置位时禁止中断

1: 当 PCON1 的 KBIF,BOF0,WDTF 或 SCON 的 TI 置位并且在 SFIE 里相对应的系统中断标志被使能时使能中断

Bit 2: EPCA, PCA 中断使能

0: 禁止 PCA 中断

1: 使能 PCA 中断

Bit 1: EADC, ADC 中断使能

0: 当 ADC 模块的 ADCON0.ADCI 置位时禁止中断

1: 当 ADC 模块的 ADCON0.ADCI 置位时使能中断

Bit 0: 保留。当写 EIE1 时，软件必须在这位上写”0”。

SFIE: 系统标志中断使能寄存器

SFR 页 =普通

SFR 地址 = 0x8E POR = 00x0-0x00

7	6	5	4	3	2	1	0
UTIE	SDIFIE	--	RTCFIE	KBIFIE	--	BOF0IE	WDTFIE
R/W	**W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 7: UART TI 使能在系统标志中断

0: 禁止系统标志中断向量共享给 TI

1: 设置 TI 标志将与系统标志中断共享中断向量

Bit 6: SDIFIE, 串行接口监测标志中断使能

0: 禁止 SDIF(STAF 或 STOF) 中断

1: 使能 SDIF(STAF 或 STOF) 中断

Bit 5: 保留。当写 SFIE 时，软件必须在这位上写”0”。

Bit 4: RTCFIE, RTCF (PCON1.4) 中断使能

0: 禁止 RTCF 中断

1: 使能 RTCF 中断

Bit 3: KBIFIE, KBIF (PCON1.3)中断使能

0: 清零禁止 KBIF 中断。

1: 置位使能 KBIF 中断。

Bit 2: 保留。当写 SFIE 时，软件必须在这位上写”0”。

Bit 1: BOF0IE, BOF0 (PCON1.1) 中断使能

0: 清零禁止 BOF0 中断。

1: 置位使能 BOF0 中断。

Bit 0: WDTFIE, WDTF (PCON1.0)中断使能

0: 禁止 WDTF 中断

1: 使能 WDTF 中断

IP0L: 中断优先级 0 低位寄存器

SFR 页 =普通

SFR 地址 = 0xB8 复位值= xxx0-0000

7	6	5	4	3	2	1	0
--	--	--	PSL	PT1L	PX1L	PT0L	PX0L
W	**W**	**W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~5: 保留。当写 IP0L 时，软件必须在这些位上写”0”。

Bit 4: PSL, 串口中断优先级低位寄存器

Bit 3: PT1L, Timer 1 中断优先级低位寄存器

Bit 2: PX1L, 外部中断 1 优先级低位寄存器

Bit 1: PT0L, Timer 0 中断优先级低位寄存器

Bit 0: PX0L, 外部中断 0 优先级低位寄存器

IP0H: 中断优先级 0 高位寄存器

SFR 页 =普通

SFR 地址 = 0xB7 复位值= xxx0-0000

7	6	5	4	3	2	1	0
--	--	--	PSH	PT1H	PX1H	PT0H	PX0H
W	**W**	**W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~5: 保留。当写 IP0H 时，软件必须在这些位上写”0”。

Bit 4: PSH, 串口中断优先级高位寄存器

Bit 3: PT1H, Timer 1 中断优先级高位寄存器

Bit 2: PX1H, 外部中断 1 优先级高位寄存器

Bit 1: PT0H, Timer 0 中断优先级高位寄存器

Bit 0: PX0H, 外部中断 0 优先级高位寄存器

EIP1L: 扩展中断优先级 1 低寄存器

SFR 页 =普通

SFR 地址 = 0xAE 复位值= xxxx-000x

7	6	5	4	3	2	1	0
--	--	--	--	PSFL	PPCAL	PADCL	--
W	**W**	**W**	**W**	**R/W**	**R/W**	**R/W**	**W**

Bit 7~4: 保留。当写 EIP1L 时，软件必须在这些位上写”0”。

Bit 3: PSFL, 系统标志中断优先级低位寄存器

Bit 2: PPCAL, PCA 中断优先级低位寄存器

Bit 1: PADCL, ADC 中断优先级低位寄存器

Bit 0: 保留。当写 EIP1L 时，软件必须在这位上写”0”。

EIP1H: 扩展中断优先级 1 高寄存器

SFR 页 =普通

SFR 地址 = 0xAF 复位值= xxxx-000x

7	6	5	4	3	2	1	0
--	--	--	--	PSFH	PPCAH	PADCH	--
W	**W**	**W**	**W**	**R/W**	**W**	**W**	**W**

Bit 7~4: 保留。当写 EIP1H 时，软件必须在这些位上写”0”。

Bit 3: PSFH, 系统标志中断优先级高位寄存器

Bit 2: PPCAH, PCA 中断优先级高位寄存器

Bit 1: PADCH, ADC 中断优先级高位寄存器

Bit 0: 保留。当写 EIP1H 时，软件必须在这些位上写”0”。

AUXR0: 辅助寄存器 0

SFR 页 =普通

SFR 地址 = 0xA1 复位值= 0000-0000

7	6	5	4	3	2	1	0
P40OC1	P40OC0	P40FD	T0XL	P1FS1	P1FS0	**INT1H**	**INT0H**
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 1: INT1H, INT1 高电平/上升沿触发使能

0: 保留 nINT1 引脚在低电平或下降沿触发 INT1。

1: 使能 nINT1 引脚在高电平或上升沿触发 INT1。

Bit 0: INT0H, INT0 高电平/上升沿触发使能

0: 保留 nINT0 引脚在低电平或下降沿触发 INT0。

1: 使能 nINT0 引脚在高电平或上升沿触发 INT0。

14.9. 中断示例代码

(1). 规定功能: 在掉电模式下设置 INT0 高电平唤醒 MCU

汇编语言代码范例:

```
    ORG     00003h
ext_int0_isr:
    to do.....
    RETI

main:

    SETB    P32                         ;

    ORL     IP0L,#PX0L                  ; 选择 INT0 中断优先级
    ORL     IP0H,#PX0H                  ;

    ORL     AUXR0,#INT0H                ; 设置 INT0 高电平激活

    JB      P32,$                       ; 确认 P3.2 输入低

    SETB    EX0                         ; 使能 INT0 中断
    CLR     IE0                         ; 清除 INT0 标志
    SETB    EA                          ; 使能全局中断

    ORL     PCON0,#PD                   ; 设置 MCU 进入掉电模式
```

C 语言代码范例:

```
void ext_int0_isr(void) interrupt 0
{
    To do……
}

void main(void)
{
    P32 = 1;
```

```
    IP0L |= PX0L;                           //选择 INT0 中断优先级
    IP0H |= PX0H;

    AUXR0 |= INT0H;                         //设置 INT0 高电平激活

    while(P32);                             //确认 P3.2 输入低

    EX0 = 1;                                //使能 INT0 中断
    IE0 = 0;                                //清除 INT0 标志
    EA = 1;                                 //使能全局中断

    PCON0 |= PD;                            //设置 MCU 进入掉电模式
}
```

15. 定时器/计数器

MA86E/L508 有两个定时器/计数器：定时器 0 和定时器 1。定时器 0/1 可以被配置成定时器或事件计数器。

在“定时器”功能中，定时器率是被预分频为每 12 个时钟周期寄存器值加 1。换句话说，它计数标准 C51 的机器周期。AUXR2.T0X12 和 AUXR2.T1X12 用来设置定时器 0/1 的定时器率为 1 个时钟周期。AUXR0.T0XL 与 AUXR2.T0X12 结合选择定时器 0 的预分频（SYSCLK/48 and SYSCLK/192）。

在“计数器”功能中，寄存器响应相对应的输入脚 T0 或 T1 从“1”到“0”的变化进行递增。在这个功能中，每个定时器率周期，外部输入脚被采样。当采样出现在这一周期是一高电平，在下一周期是低电平时，计数加 1。在检测到变化之后的这个周期结束后，新的计数值出现在寄存器上。

15.1. 定时器 0 和定时器 1

15.1.1. 模式 0 结构

定时器寄存器被配置成一个 PWM 发生器。计数器所有位从全 1 翻转到全 0，置位定时器中断标志位 TFx。当 TRx=1 且 GATE=0 或 INTx=1，定时器使能输入计数。定时器 0 和定时器 1 的模式 0 运作时相同的。定时器 0/1 的 PWM 功能见图 15-1 和图 15-2。

图 15–1. 定时器 0 模式 0 结构

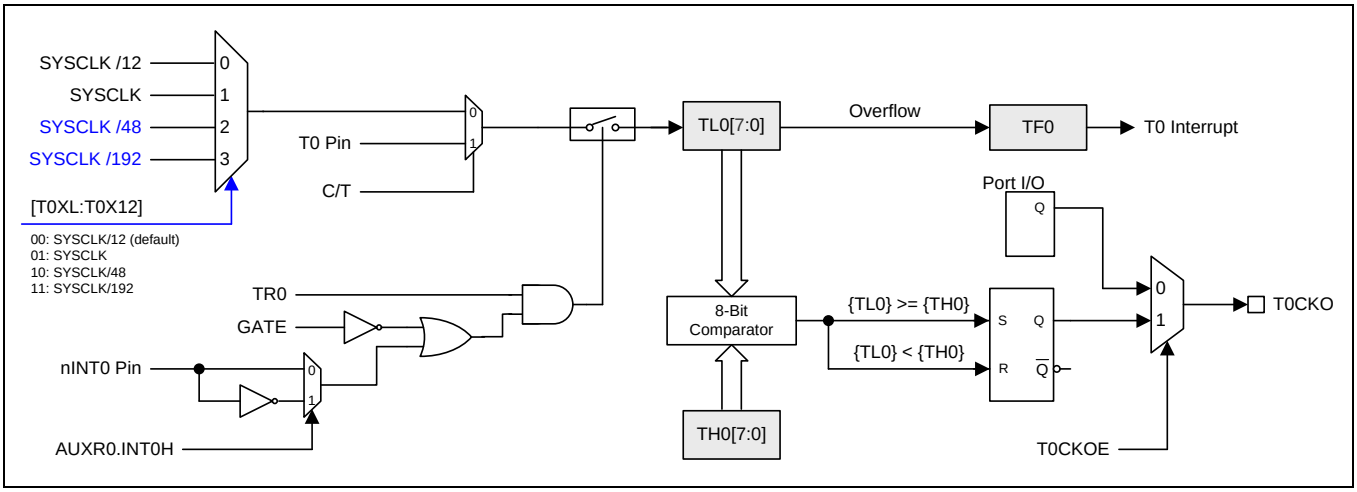
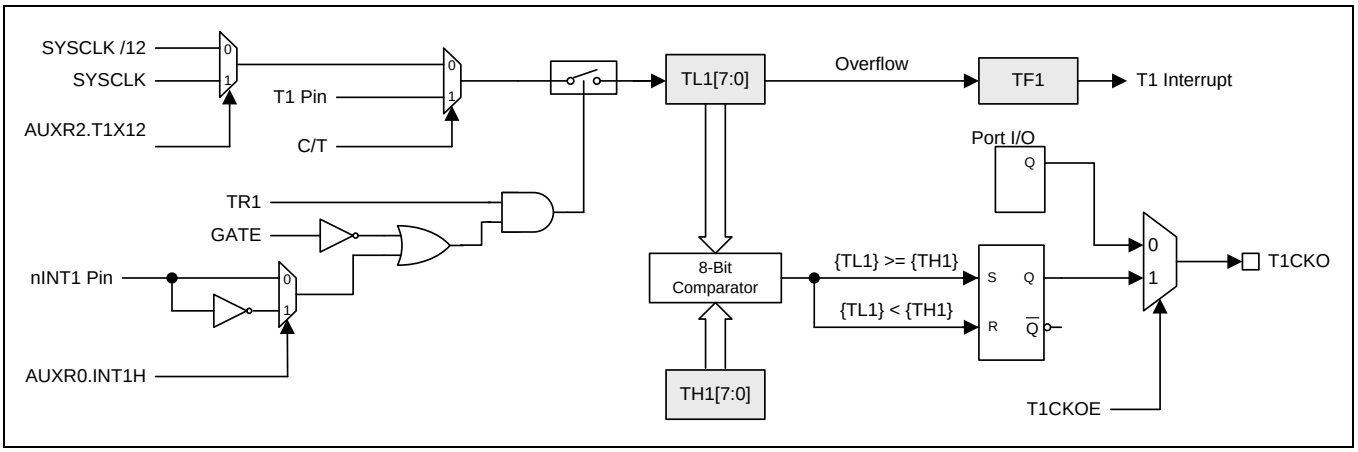


图 15–2. 定时器 1 模式 0 结构



15.1.2. 模式 1 结构

定时器 0/1 在模式 1 配置为 16 位的定时器或计数器。GATE, INTx 和 TRx 的功能与模式 0 一样。图 15-3 和图 15-4 是定时器 0 和定时器 1 的结构图。

图 15–3. 定时器 0 模式 1 结构

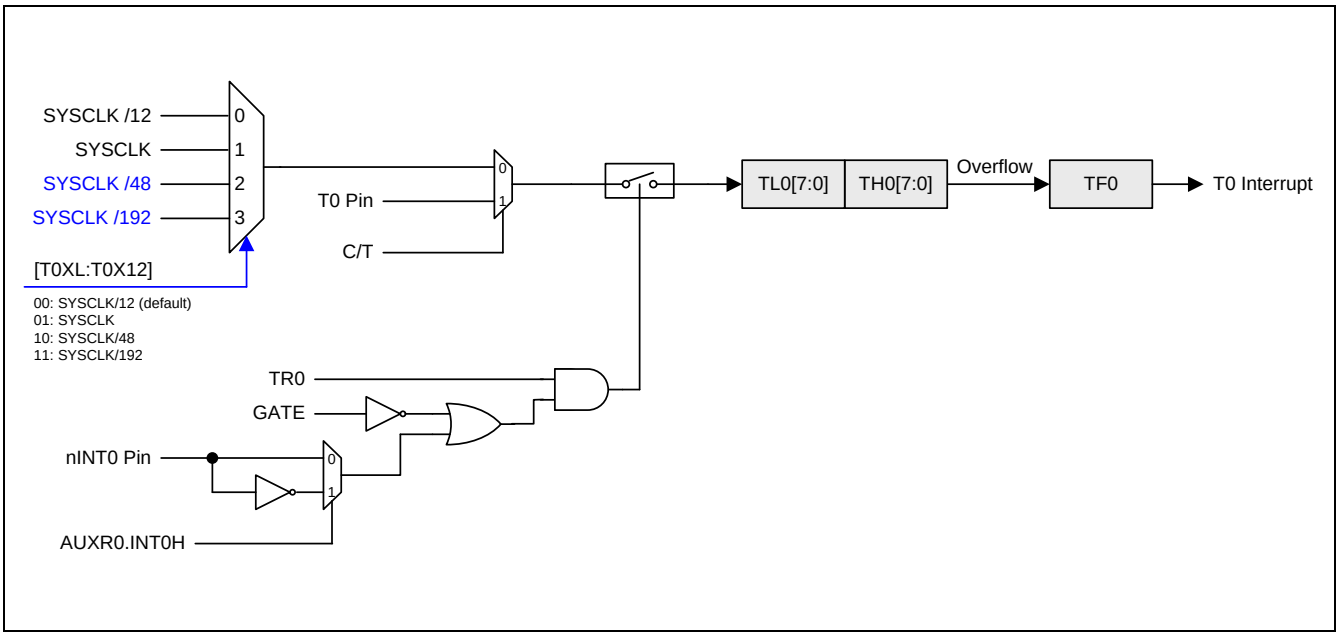
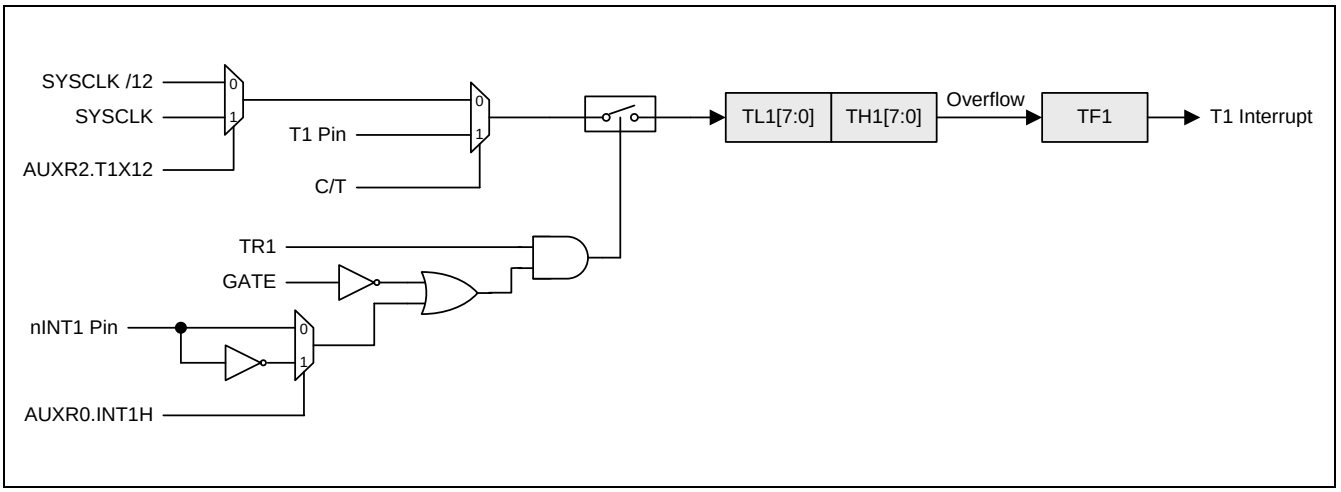


图 15–4. 定时器 1 模式 1 结构



15.1.3. 模式 2 结构

模式 2 配置定时器寄存器为一个自动加载的 8 位计数器(TLx)。TLx 溢出不仅置位 TFx，而且也将 THx 的内容加载到 TLx，THx 内容由软件预置，加载不会改变 THx 的值。定时器 0 和定时器 1 的模式 2 运作时相同的。

图 15–5. 定时器 0 模式 2 结构

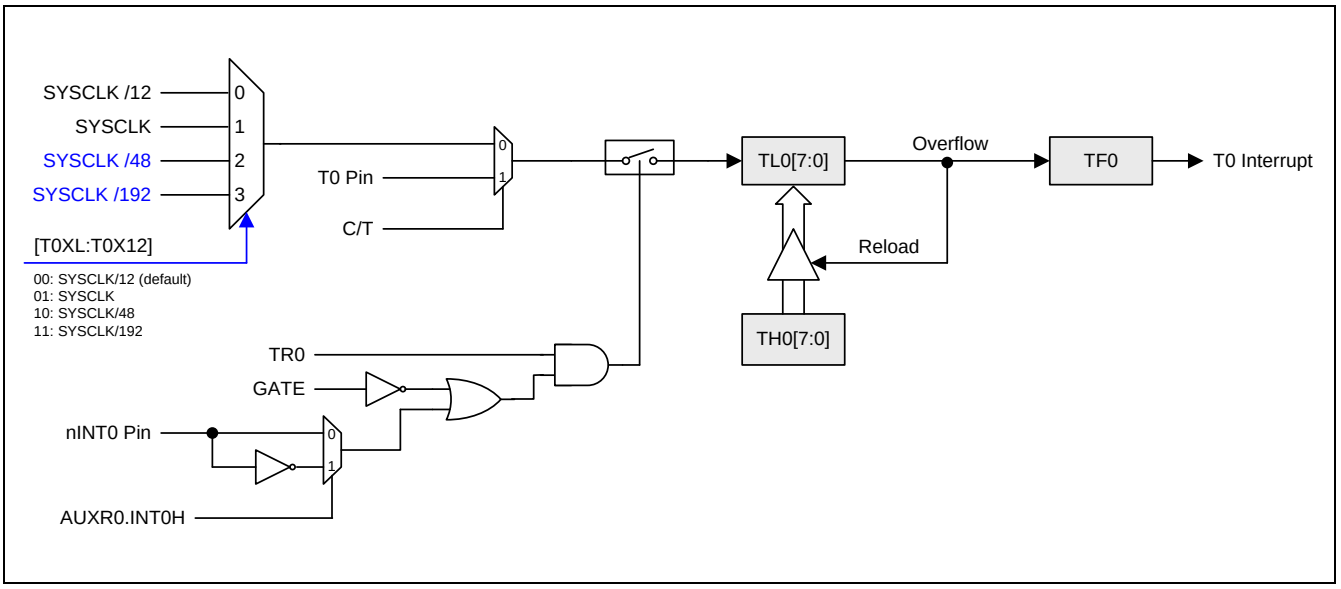
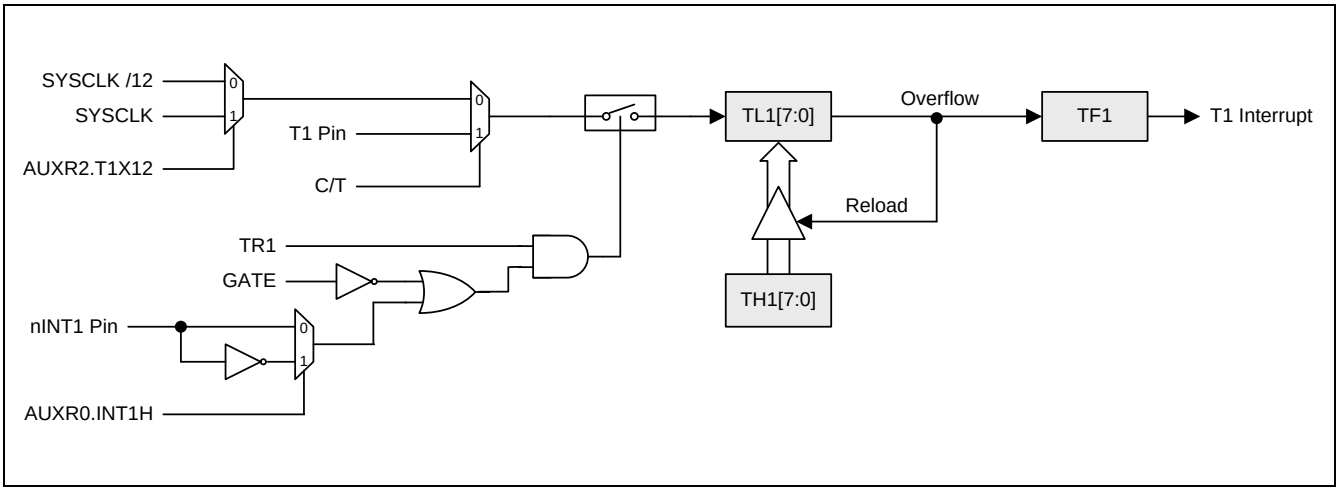


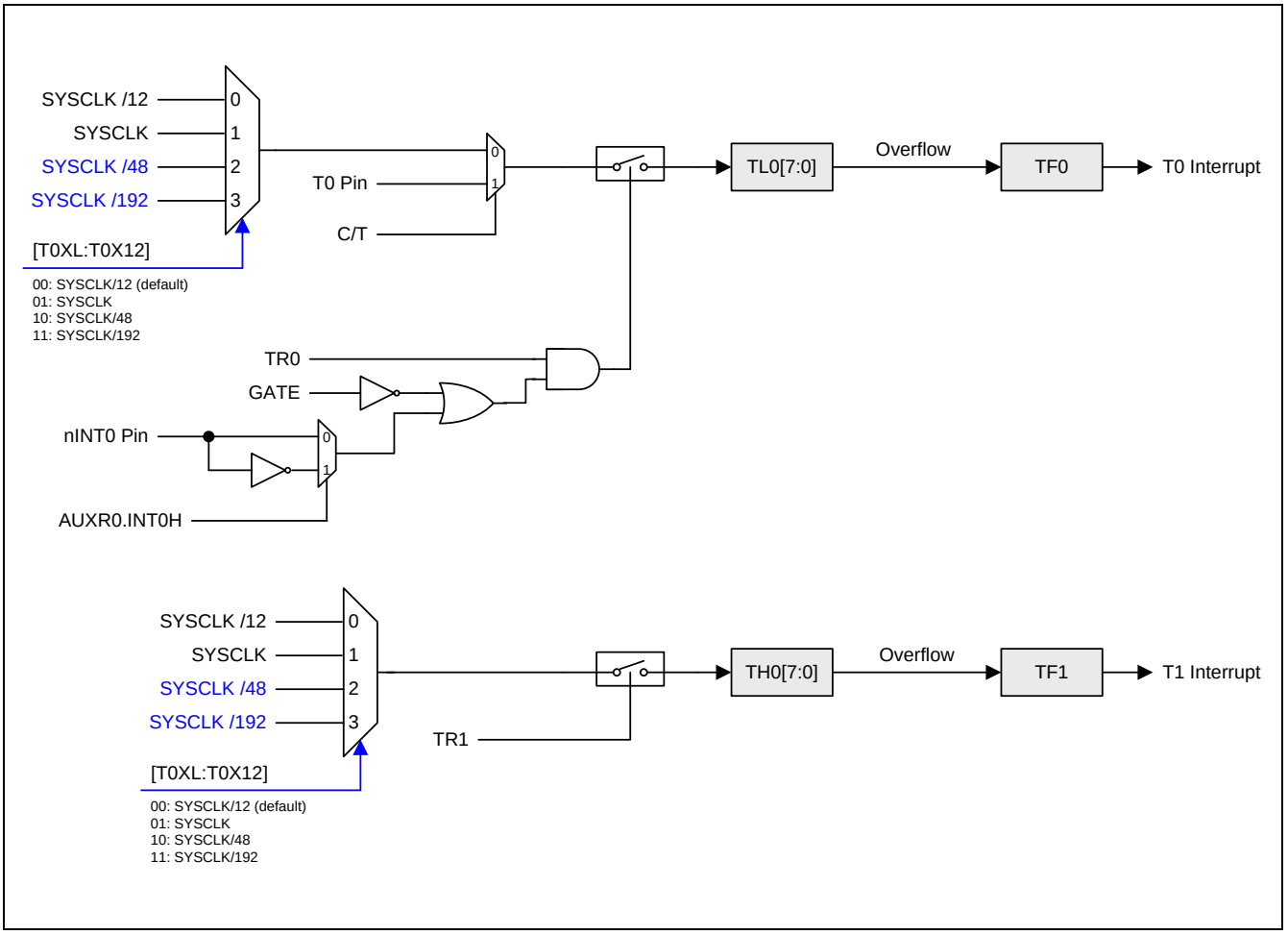
图 15–6. 定时器 1 模式 2 结构



15.1.4. 模式 3 结构

定时器1在模式3保持计数值。效果和设置TR1=1一样。定时器0在模式3建立TL0和TH0两个独立的计数器。TL0使用定时器0控制位：C/T、GATE、TR0、INT0和TF0。TH0锁定为定时器功能(不能成为外部事件计数器)且接替Timer1来使用TR1和TF1，TH0控制Timer 1中断。

图 15–7. 定时器 0 模式 3 结构



15.1.5. 定时器 0/1 可编程时钟输出

定时器 0 和定时器 1 有一个时钟输出模式(C/Tx=0&TxCKOE=1)。 在此模式下，定时器 0 或定时器 1 工作在 8 位自动重载定时器模式，用作占空比为 50%的可编程时钟发生器。产生的时钟分别在 P3.4(T0CKO)和 P3.5(T1CKO)输出。定时器 0 八位定时器(TL0)由输入时钟 (SYSCLK，SYSCLK/12， SYSCLK/48 或 SYSCLK/192) 加 1，定时器 1 八位定时器(TL1)由输入时钟 (SYSCLK/12 或 SYSCLK) 加 1。定时器从一个装载的初值一直计数到溢出，一旦溢出(TH0,TH1) 的值就会载入到(TL0, TL1)继续计数。下面是输出频率的计算公式：

图 15–8. 定时器 0 时钟输出公式

$$\text{T0 Clock-out Frequency} = \frac{\text{SYSCLK Frequency}}{n \times (256 - \text{THx})}$$

; n=24, if {T0XL,T0X12}=00
; n=2, if {T0XL,T0X12}=01
; n=96, if {T0XL,T0X12}=10
; n=384, if {T0XL,T0X12}=11
; C/T = 0

图 15–9. 定时器 1 时钟输出公式

$$\text{T1 Clock-out Frequency} = \frac{\text{SYSCLK Frequency}}{n \times (256 - \text{TH1})}$$

; n=24, if T1X12=0
; n=2, if T1X12=1
; C/T = 0

注意:

(1) Timer 0/1 溢出标志，TF0/1,在定时器 0/1 溢出时被置位，但不产生中断。

(2) 对于 SYSCLK=12MHz & TxX12=0, 定时器 0/1 可编程输出频率范围是 1.95KHz 到 500KHz.

(3) 对于 SYSCLK=12MHz & TxX12=1, 定时器 0/1 可编程输出频率范围是 23.43KHz 到 6MHz.

(2) 对于 SYSCLK=12MHz，TxX12=0 并且 T0XL=1, 定时器 0 可编程输出频率范围是 488Hz 到 125KHz.

(3) 对于 SYSCLK=12MHz，TxX12=1 并且 T0XL=1, 定时器 0 可编程输出频率范围是 122KHz 到 31.25KHz.

图 15–10. 定时器 0 在时钟输出模式

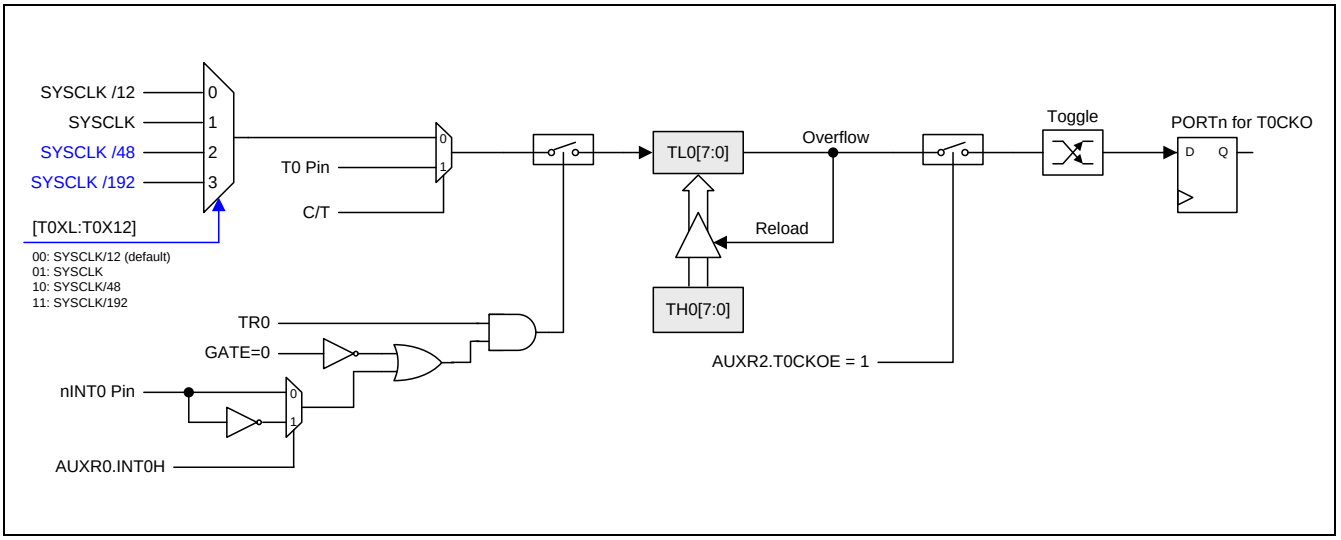
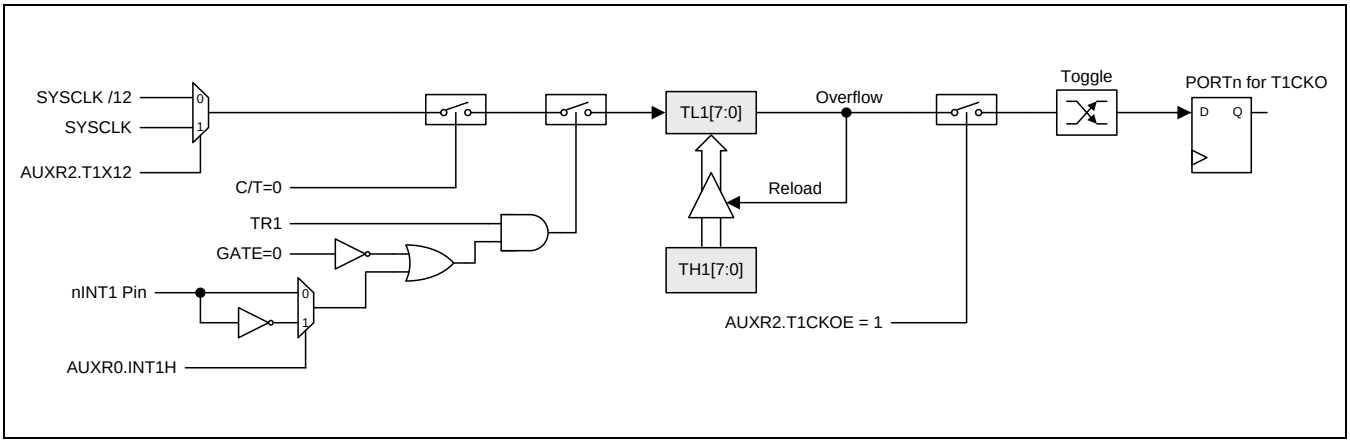


图 15–11. 定时器 1 在时钟输出模式



定时器 0/1 在时钟输出模式如何编程

- 选择 AUXR2. T0X12 和 AUXR0.T0XL 确定定时器 0 的时钟源，或 AUXR2. T1X12 确定定时器 1 的时钟来源
- 置位 AUXR2 寄存器的 T0CKOE/T1CKOE 位
- 清零 TMOD 寄存器的 C/T 位
- 按公式确定 8 位重载值，并将其写入 TH0/TH1 寄存器
- 同样的值作为初始值写入 TL0/TL1 寄存器
- 置位 TCON 寄存器的 TR0/TR1 位启动 Timer 0/1.

在时钟输出模式定时器 0/1 不会产生中断，这个跟定时器 1 被用作波特率发生器一样。这样定时器 1 就可以同时作为时钟输出和波特率发生器。注意：时钟输出频率和波特率都是跟定时器 1 的溢出率一样。

15.1.6. 定时器 0/1 寄存器

TCON: 定时器/计数器控制寄存器

SFR 页 = 普通

SFR 地址 = 0x88 复位值 = 0000-0000

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: TF1, 定时器 1 溢出标志位。

0: 处理器进入中断向量程序由硬件清零，或由软件清零。

1: 定时器/计数器溢出时由硬件置位，或由软件置位。

Bit 6: TR1, 定时器 1 运行控制位.

0: 软件清零关闭定时器/计数器 1。

1: 软件置位开启定时器/计数器 1。

Bit 5: TF0, 定时器 0 溢出标志位.

0: 处理器进入中断向量程序由硬件清零，或由软件清零。

1: 定时器/计数器溢出时由硬件置位，或由软件置位。

Bit 4: TR0, 定时器 0 运行控制位.

0: 软件清零关闭定时器/计数器 0

1: 软件置位打开定时器/计数器 0.

TMOD: 定时器/计数器模式控制寄存器

SFR 页 =普通

SFR 地址 = 0x89 复位值= 0000-0000

7	6	5	4	3	2	1	0
GATE	C/T	M1	M0	GATE	C/T	M1	M0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

|←---------- Timer1 ----------→|←----------Timer0 ----------→|

Bit 7/3: Gate, 定时器 1/0 门控制位

0: 禁止定时器 1/0 门控位

1: 使能定时器 1/0 门控位。当它置 1 时，只有在/INT1 或/INT0 引脚为高电平且 TR1 或 TR0 控制位置位时，定时器 1/0 才会使能。

Bit 6/2: C/T, 定时器或计数器功能选择

0: 清零为定时器功能，从内部系统时钟输入

1: 置位为计数器功能，从 T1/T0 引脚输入

Bit 5~4/1~0: 工作模式选择

M1	M0	工作模式
0	0	定时器/计数器 0/1 作为 8 位 PWM 发生器
0	1	16 位定时器/计数器
1	0	8 位自动重载定时器/计数器
1	1 (定时器 0)	TL0 是 8 位定时器/计数器，TH0 仅是个 8 位定时器
1	1 (定时器 1)	定时器/计数器 1 停止

TL0: 定时器 0 低 8 位寄存器

SFR 页 =普通

SFR 地址 = 0x8A 复位值= 0000-0000

7	6	5	4	3	2	1	0
TL0.7	TL0.6	TL0.5	TL0.4	TL0.3	TL02	TL0.1	TL0.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

TH0: 定时器 0 高 8 位寄存器

SFR 页 =普通

SFR 地址 = 0x8C 复位值= 0000-0000

7	6	5	4	3	2	1	0
TH0.7	TH0.6	TH0.5	TH0.4	TH0.3	TH0.2	TH0.1	TH0.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

TL1: 定时器 1 低 8 位寄存器

SFR 页 =普通

SFR 地址 = 0x8B 复位值= 0000-0000

7	6	5	4	3	2	1	0
TL1.7	TL1.6	TL1.5	TL1.4	TL1.3	TL1.2	TL1.1	TL1.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

TH1: 定时器 0 高 8 位寄存器

SFR 页 =普通

SFR 地址 = 0x8D 复位值= 0000-0000

7	6	5	4	3	2	1	0
TH1.7	TH1.6	TH1.5	TH1.4	TH1.3	TH1.2	TH1.1	TH1.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

AUXR2: 辅助寄存器 2

SFR 页 =普通

SFR 地址 = 0xA3 复位值= 0000-0000

7	6	5	4	3	2	1	0
--	BTI	URM0X3	SM3	T1X12	T0X12	T1CKOE	T0CKOE
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 3: T1X12, 当 C/T=0，定时器 1 时钟源选项

0: 清零选择 SYSCLK/12.

1: 置位选择 SYSCLK 作为时钟源

Bit 2: T0X12, 当 C/T=0，定时器 0 时钟源选项

0:清零选择 SYSCLK/12.

1: 置位选择 SYSCLK 作为时钟源

T0XL, T0X12	定时器 0 时钟原选择
0 0	SYSCLK/12
0 1	SYSCLK
1 0	SYSCLK/48
1 1	SYSCLK/192

Bit 1: T1CKOE, 定时器 1 时钟输出使能

0: 禁止定时器 1 时钟输出

1: 使能定时器 1 在 P3.5.上输出

Bit 0: T0CKOE, 定时器 0 时钟输出使能

0: 禁止定时器 0 时钟输出

1: 使能定时器 0 在 P3.4.上输出

15.1.7. 定时器 0/1 示例代码

(1). 规定功能:系统时钟 SYSCLK = ILRCO 时定时器 T0 以 640Hz 的频率唤醒空闲模式

汇编语言代码范例:

```
    ORG     0000Bh
time0_isr:
    to do…
    RETI

main:                                   ;
;   //选择系统时钟 Sysclk 为 ILRCO
    MOV     IFADRL,#(CKCON2)            ; 索引 P 页地址为 CKCON2
    CALL    _page_p_sfr_read            ; 读取 CKCON2 数据

    ANL     IFD,#~(OSCS1 | OSCS0)       ; OSCin 时钟源更改为 ILRCO
    ORL     IFD,#(OSCS1)
    CALL    _page_p_sfr_write           ; 写数据到 CKCON2

    ANL     IFD,#~(XTALE | IHRCOE)      ; 禁止 XTAL 和 IHRCO
    CALL    _page_p_sfr_write           ; 写数据到 CKCON2

    MOV     IFADRL,#(DCON0)             ; 索引 P 页地址为 DCON0
    CALL    _page_p_sfr_read            ; 读取 DCON0 数据

    ANL     IFD,#~(HSE)                 ; 当系统时钟 SYSCLK ≤ 6MHz 为了省电禁止 HSE
    CALL    _page_p_sfr_write           ; 写数据到 DCON0

    ANL     CKCON0,#(AFS)               ; 选择 SCKS[2:0] = 0 = OSCin/1

    ORL     AUXR2,#T0X12                ; 选择 SYSCLK/1 作为定时器 0 时钟输入
    ANL     AUXR0,#~T0XL                ;

    MOV     TH0,#(256-100)              ; 设置定时器 0 溢出率= SYSCLK x 100
    MOV     TL0,#(256-100)              ;
    ANL     TMOD,#(0F0h|T0M1)           ; 设置定时器 0 工作在模式 2
    ORL     TMOD,#T0M1                  ;
    CLR     TF0                         ; 清除定时器 0 标志
```

```
    ORL     IP0L,#PT0L                  ; 选择定时器 0 中断优先级
    ORL     IP0H,#PT0H                  ;

    SETB    ET0                         ; 使能定时器 0 中断
    SETB    EA                          ; 使能全局中断

    SETB    TR0                         ; 启动定时器 0 运行

    ORL     PCON0,#IDL                  ; 设置 MCU 进入空闲模式
```

C 语言代码范例:

```
void time0_isr(void) interrupt 1
{
    To do…
}

void main(void)
{
    IFADRL = CKCON2;                    // 索引 P 页地址为 CKCON2
    page_p_sfr_read();                  // 读取 CKCON2 数据.

    IFD = ~(OSCS1 | OSCS0);             // OSCin 时钟源更改为 ILRCO
    IFD |= OSCS1;
    page_p_sfr_write();                 // 写数据到 CKCON2

    IFD &= ~(XTALE | IHRCOE);           //禁止 XTAL 和 IHRCO
    page_p_sfr_write();                 // 写数据到 CKCON2

    IFADRL = DCON0;                     // 索引 P 页地址为 DCON0
    page_p_sfr_read();                  // 读取 DCON0 数据

    IFD &= ~HSE;                        //当系统时钟 SYSCLK ≤ 6MHz 为了省电禁止 HSE
    page_p_sfr_write();                 // 写数据到 DCON0

    CKCON0 &= AFS;                      // 选择 SCKS[2:0] = 0 = OSCin/1
```

```
    AUXR2 |= T0X12;                     //选择 SYSCLK/1 作为定时器 0 时钟输入
    AUXR0 &= ~T0XL;

    TH0 = TL0 = (256-100);              //设置定时器 0 溢出率= SYSCLK x 100
    TMOD &= 0xF0;                       //设置定时器 0 工作在模式 2
    TMOD |= T0M1;
    TF0 = 0;                            //清除定时器 0 标志

    IP0L |= PT0L;                       //选择定时器 0 中断优先级
    IP0H |= PT0H;

    ET0 = 1;                            //使能定时器 0 中断
    EA = 1;                             //使能全局中断

    TR0 = 1;                            //启动定时器 0 运行

    PCON0=IDL;                          //设置 MCU 进入空闲模式
}
```

(2). 规定功能: SYSCLK/48 时钟源设置定时器 0 的时钟输出

汇编语言代码范例:

```
    CLR     TR0                         ;

    ANL     P3M0,#0EFh                  ; 设置 P3.4(T0CKO)为推挽输出
    ORL     P3M1,#010h                  ;
    ORL     AUXR2,#T0CKOE               ; 使能 T0CKO

    ANL     AUXR2,#~T0X12               ; 选择 SYSCLK/48 作为定时器 0 时钟输入
    ORL     AUXR0,#T0XL                 ;

    MOV     TH0,#0FFh                   ;
    MOV     TL0,#0FFh                   ;

    ANL     TMOD,#0F0h                  ; 设置定时器 0 工作在模式 2
    ORL     TMOD,#T0M1                  ;
```

```
SETB    TR0                         ; 启动定时器 0 运行
```

C 语言代码范例:

```
TR0 = 0;

P3M0 &= 0xEF;                       //设置 P3.4(T0CKO)为推挽输出
P3M1 |= 0x10;
AUXR2 |= T0CKOE;                    //  使能 T0CKO

AUXR2 &= ~T0X12;                    //选择 SYSCLK/48 作为定时器 0 时钟输入
AUXR0 |= T0XL;

TH0 = TL0 = 0xFF;

TMOD &= 0xF0;                       //设置定时器 0 工作在模式 2
TMOD |= T0M1;

TR0 = 1;                            //启动定时器 0 运行
```

(3). 规定功能: SYSCLK 时钟源设置定时器 1 的时钟输出

汇编语言代码范例:

```
ORL     P3M1,#020h                  ; 设置 P3.5(T1CKO)为推挽输出
ANL     P3M0,#0DFh                  ;

ORL     AUXR2,#(T1X12|T1CKOE)       ; 选择 SYSCLK 作为定时器 1 时钟输入
                                    ; 使能 T1CKO

MOV     TH1,#0FFh                   ;
MOV     TL1,#0FFh                   ;

ANL     TMOD,#00Fh                  ; 设置定时器 1 工作在模式 2
ORL     TMOD,#T1M1                  ;

SETB    TR1                         ; 启动定时器 1 运行
```

C 语言代码范例:

```
P3M1 |= 0x20;                          //设置 P3.5(T1CKO)为推挽输出
P3M0 &= 0xDF;

AUXR2 |= (T1X12|T1CKOE);               //选择 SYSCLK 作为定时器 1 时钟输入
                                       // 使能 T1CKO

TH1 = TL1 = 0xFF;

TMOD &= 0x0F;                          //设置定时器 1 工作在模式 2
TMOD |= T1M1;

TR1 = 1;                               //启动定时器 1 运行
```

(4). 规定功能: 设置定时器 0 工作在模式 0 输出 25% PWM, PWM 频率= 46.875K, SYSCLK = 12MHz

汇编语言代码范例:

```
CLR     TR0                            ; 关闭定时器 0 运行

ORL     P3M1,#010h                     ; 设置 P3.4(T0CKO)为推挽输出
ANL     P3M0,#0EFh                     ;

ANL     AUXR0,#~T0XL                   ;
ORL     AUXR2,#(T0X12|T0CKOE);         ; 选择 SYSCLK 作为定时器 0 时钟输入
                                       ; 使能 T0CKO

MOV     TH0,#0C0h                      ; 设置 PWM 占空比= 25%
MOV     TL0,#000h                      ;

ANL     TMOD,#0F0h                     ; 设置定时器 0 为 PWM 功能工作在模式 0

SETB    TR0                            ; 启动定时器 0 运行
```

C 语言代码范例:

```
TR0 = 0;                                //关闭定时器 0 运行

P3M0 &= 0xEF;                           //设置 P3.4(T0CKO)为推挽输出
P3M1 |= 0x10;

AUXR0 &= ~T0XL;
AUXR2 |= (T0X12|T0CKOE);                //选择 SYSCLK 作为定时器 0 时钟输入
                                        //  使能 T0CKO

TH0 = 0xC0;                             //设置 PWM 占空比= 25%
TL0 = 0x00;

TMOD &= 0xF0;                           //设置定时器 0 为 PWM 功能工作在模式 0

TR0 = 1;                                //启动定时器 0 运行
```

(5). 规定功能: 设置定时器 1 工作在模式 0 输出 75% PWM, PWM 频率= 46.875K, SYSCLK = 12MHz

汇编语言代码范例:

```
CLR     TR1                             ; 关闭定时器 1 运行

ORL     P3M1,#020h                      ; 设置 P3.5(T1CKO)为推挽输出
ANL     P3M0,#0DFh                      ;

ORL     AUXR2,#(T1X12|T1CKOE);          ; 选择 SYSCLK 作为定时器 1 时钟输入
                                        ; 使能 T1CKO

MOV     TH1,#040h                       ; 设置 PWM 占空比= 75%
MOV     TL1,#000h                       ;

ANL     TMOD,#00Fh                      ; 设置定时器 1 为 PWM 功能工作在模式 0

SETB    TR1                             ; 启动定时器 1 运行
```

C 语言代码范例:

```
```

```
TR1 = 0;                                //关闭定时器 1 运行

P3M0 &= 0xDF;                           //设置 P3.5(T1CKO)为推挽输出
P3M1 |= 0x20;

AUXR2 |= (T1X12|T1CKOE);                //选择 SYSCLK 作为定时器 1 时钟输入
                                        //  使能 T1CKO

TH1 = 0x40;                             //设置 PWM 占空比= 75%
TL1 = 0x00;

TMOD &= 0x0F;                           //设置定时器 1 为 PWM 功能工作在模式 0

TR1 = 1;                                //启动定时器 1 运行
```

16. 串行口(UART)

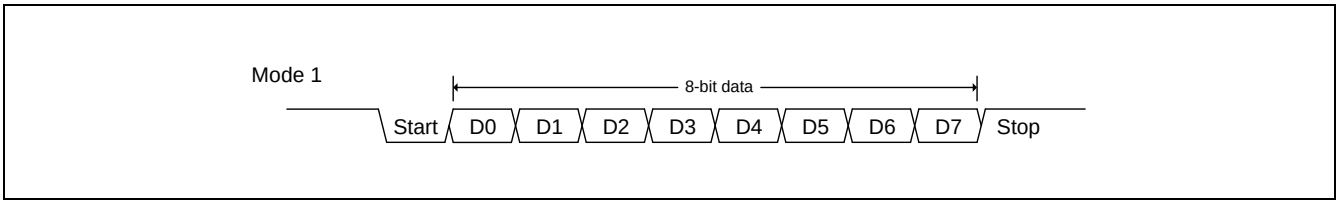
MA86E/L508 支持一个全双工的串行口，意思是可以同时发送和接收数据。它有一个接收缓冲，意味着在前一个接收到的字节没有从寄存器读出前，就可以开始接收第二个字节。但是，如果第一个字节在第二个字节接收完成前仍然没有被读出，则其中的一个字节将会丢失。串行口的接收和发送寄存器都通过特殊寄存器 SBUF 来访问。写到 SBUF0 加载到传送寄存器，当从 SBUF 读时是一个物理上独立分离的接收寄存器。

串行口可以工作在四种模式：模式 0 提供同步通讯同时模式 1、2 和模式 3 提供异步通讯。异步通讯作为一个全双工的通用异步收发器(UART)，可以同时发送和接收并使用不同的波特率。

模式 0： 8 位数据(低位先出)通过 RXD(P3.0)传送和接收。TXD(P3.1)总是作为输出移位时钟。波特率可通过 AUXR2 寄存器的 URM0X6 位选择为系统时钟频率的 1/12 或 1/4。**MA86E/L508** 串行口模式 0 的时钟极性可以软件选择，在数据的移入或移出之前由 P3.1 的状态决定。图 16-4 和图 16-5 显示了模式 0 的时钟极性波形。

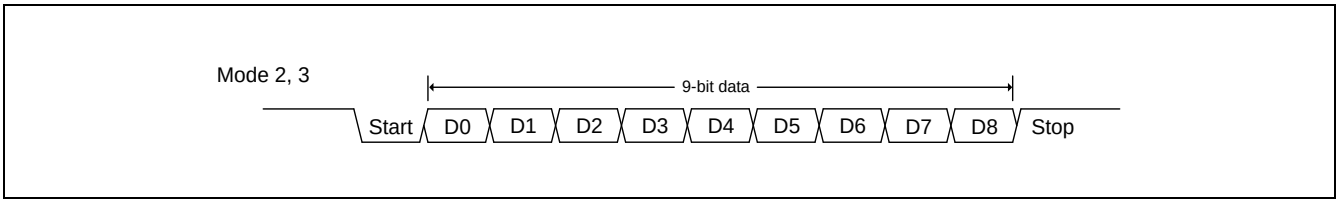
模式 1：10 位通过 TXD 传送或通过 RXD 接收，数据帧包括一个起始位(0)，8 个数据位(低位优先)，和一个停止位(1)见图 16–1.。在接收时，停止位进入到专用寄存器(SCON)的 RB8。波特率是可变的。

图 16–1. 模式 1 数据帧



模式 2：11 位通过 TXD 传送或通过 RXD 接收，数据帧包括一个起始位(0)，8 个数据位(低位优先)，一个可编程的第九个数据位和一个停止位(1) 见图 16–2。在传送时，第 9 个数据位(TB8 在 SCON 寄存器)可以分配为 0 或者 1。在接收时，第九个数据位到 SCON 寄存器中的 RB8，同时忽略停止位。波特率可以配置为 1/32 或 1/64 的系统时钟频率。

图 16–2. 模式 2, 3 数据帧



模式 3：模式 3 除了波特率可变之外其它的都和模式 2 相同。

在四种模式中，使用 SBUF 作为一个目的寄存器，可以通过任何指令发起传输。在模式 0，当 RI=0 且 REN=1 时启动接收。在其它模式，在 REN=1 时，收到起始位时启动接收。

除了标准操作外，UART 还能具有侦察丢失停止位的帧错误和自动地址识别的功能。

16.1. 串行口模式 0

串行数据通过RXD读入和输出，TXD输出移位时钟。接收和发送 8 位数据：8 个数据位(低位优先)。波特率可通过AUXR2寄存器中的URM0X6选择为系统时钟的 1/12或1/4。图 16–3 显示了串口模式 0 的简化功能框图。

使用 SBUF 作为一个目的寄存器可通过任何指令来启动传输。“写到 SBUF”信号触发 UART 引擎开始发送。SBUF 里面的数据在 TXD（P3.1）脚的每一个上升沿移出到 RXD（P3.0）脚。八个上升沿移位时钟过后，硬件置 TI 为 1 标志发送完成。图 16–4 显示了模式 0 的传送时序图。

当 REN=1 和 RI=0 时接收启动。在下一个指令周期，RX 控制单元写 11111110 到接收移位寄存器，且在下一个时钟阶段激活接收。

接收使能移位时钟选择输出功能 P3.1 引脚。当接收激活时，在移位时钟的下降沿采样 RXD（P3.0）脚并移到寄存中。八个下降沿移位时钟过后，硬件置 RI 为 1 标志接收完成。图 16–5 显示了模式 0 的接收时序图。串行口模式 0 的时钟极性可以软件选择设定，在串行数据传送的移位之前由 P3.1 的状态决定。如果 P3.1 设置为逻辑高，则时钟极性与标准 8051 一样。如果 P3.1 设置为逻辑高，则时钟极性与标准 8051 相反。

图 16–3. 串行口模式 0

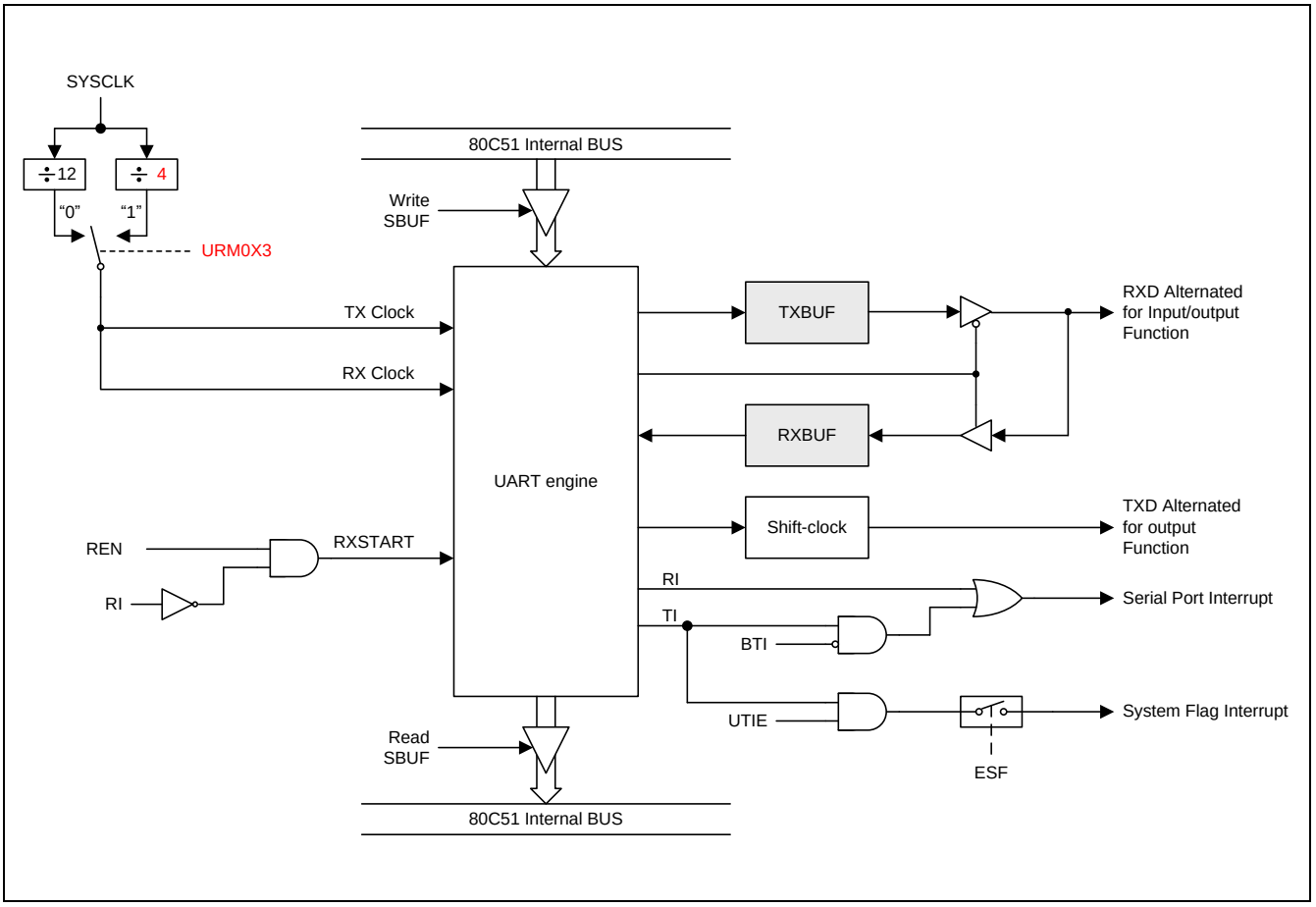

SYSCLK
÷12
÷ 4
“0”
“1”
URM0X3
80C51 Internal BUS
Write
SBUF
TX Clock
RX Clock
TXBUF
RXBUF
UART engine
Shift-clock
RXD Alternated
for Input/output
Function
TXD Alternated
for output
Function
REN
RXSTART
RI
RI
TI
BTI
Serial Port Interrupt
UTIE
System Flag Interrupt
ESF
Read
SBUF
80C51 Internal BUS

图 16–4. 模式 0 传送波形

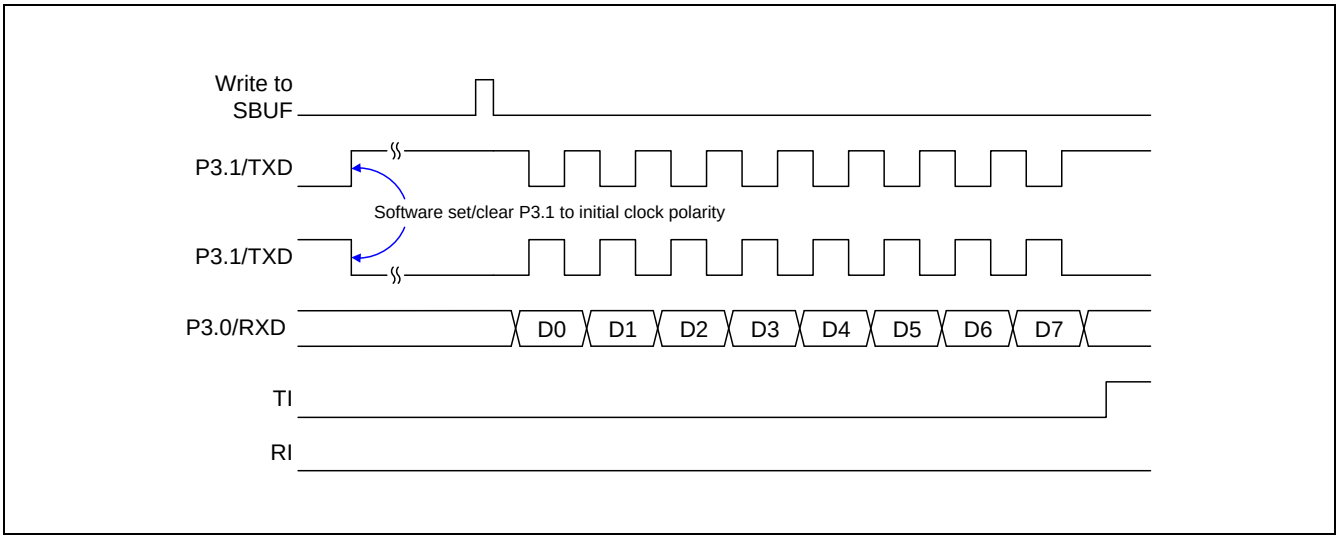
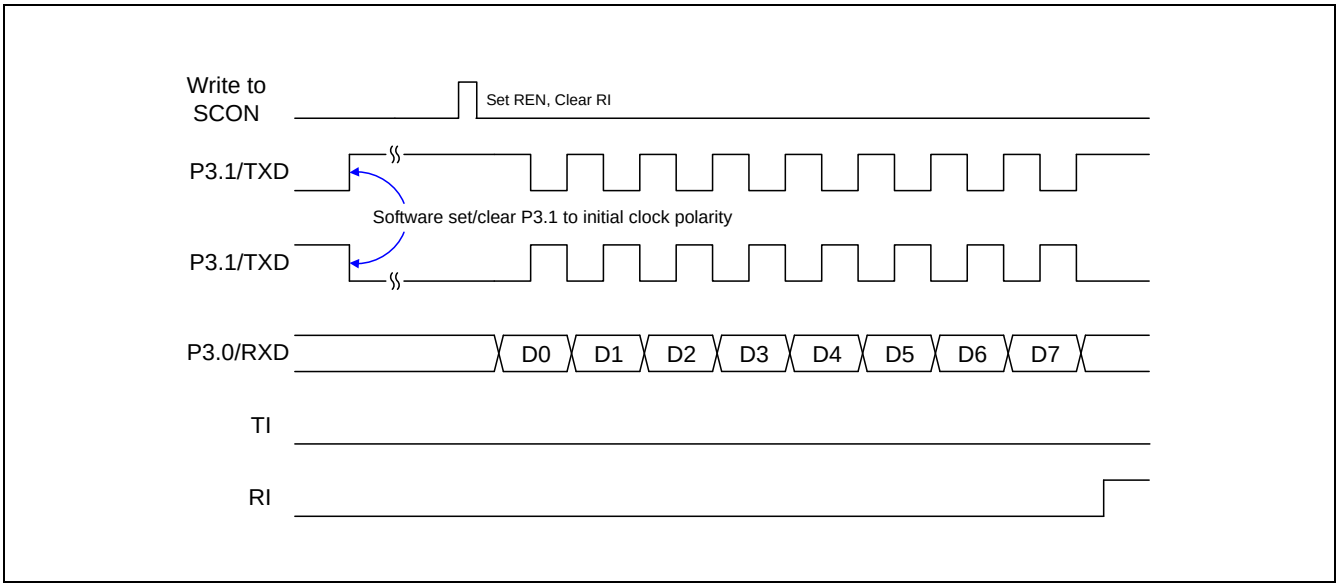


图 16–5. 模式 0 接收波形



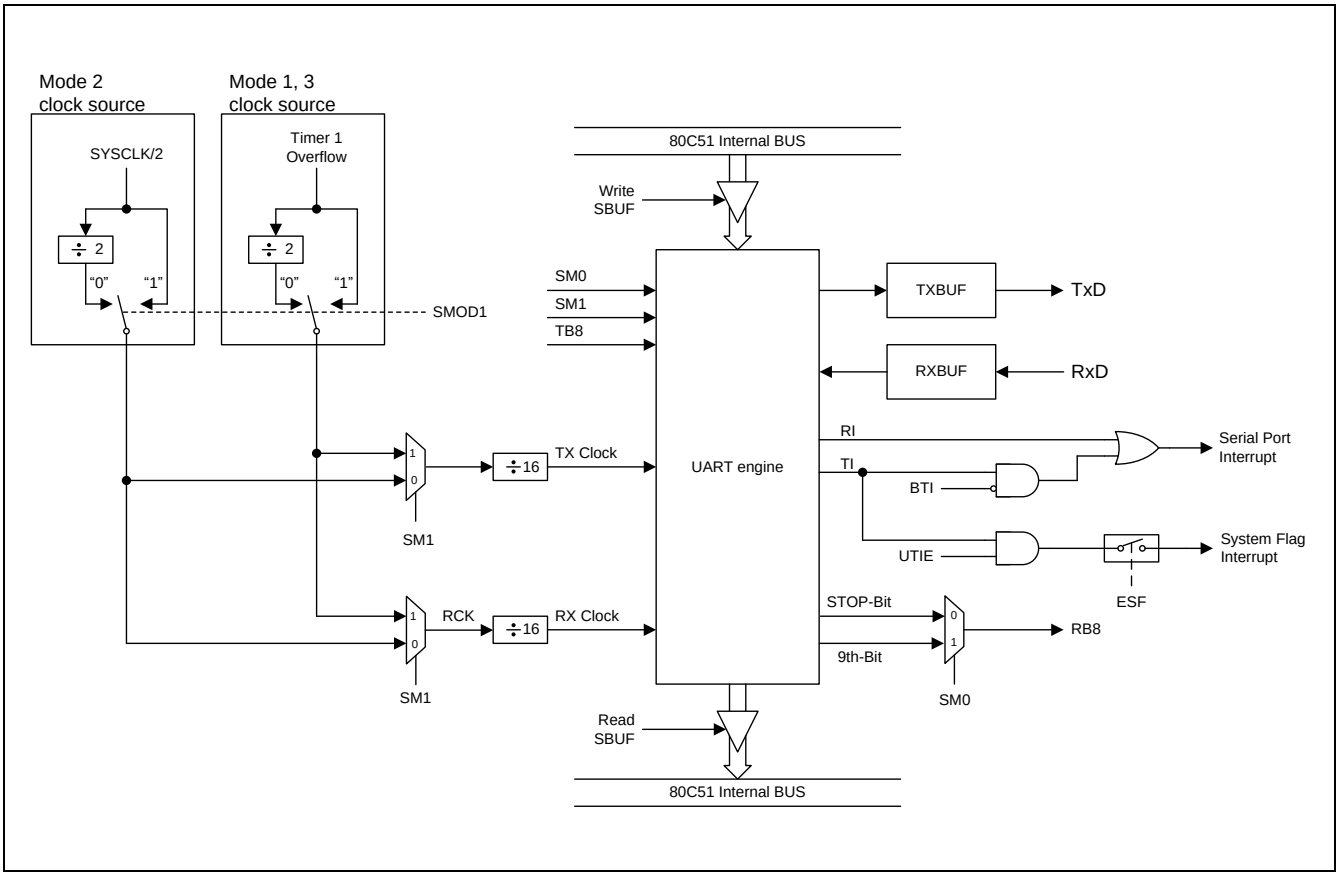
16.2. 串行口模式 1

通过 TXD 发送 10 位数据或通过 RXD 接收 10 位数据：一个起始位(0)，8 个数据位(低位先出)，和一个停止位(1)。在接收时，停止位进入 SCON 的 RB8，波特率由定时器 1 的溢出速率来决定。图 16–1 显示了模式 1 的数据帧和图 16–6 显示了模式 1 的串行口简化功能图。

使用 SBUF 作为目的寄存器的任何指令来启动传输。写到 SBUF 的信号请求 UART 引擎开始发送，当收到一个发送请求后，UART 将在 TX 时钟的上升沿开始发送。SBUF 中的数据从 TXD 引脚串行输出，数据帧如图 16–1 所示及数据宽度根据 TX 时钟不同而不同。当 8 位数据发送完后，硬件将置位 TI 表示发送结束。

当串行口控制器在 RCK 采样时钟下检测到在 RXD 有 1 到 0 跳变的起始位时接收开始。在 RXD 引脚上的数据将被串行口的位侦查器采样。当收到停止位后，硬件置位 RI 表示接收结束并把停止位加载到 SCON 寄存器的 RB8。

图 16–6. 串行口模式 1, 2, 3



16.3. 串行口模式 2 和模式 3

通过 TXD 传送 11 位或通过 RXD 接收 11 位：一个起始位(0)，8 个数据位(低位在先)，一个可编程的第 9 个数据位和一个停止位(1)。在传送时，数据的第 9 位(TB8)可分配为 0 或 1。在接收时，数据的第 9 位将进入到 SCON 的 RB8。在模式 2 波特率可编程为 1/16，1/32 或 1/64 的系统时钟频率。模式 3 可以产生可以从定时器 1 或定时器 2 产生可变的波特率。

图 16–2 列出了模式 2 和模式 3 的数据帧。图 16–6 列出了模式 2 和模式 3 的串行口功能图。接收部分和模式 1 相同。与模式 1 传送部分不同的仅仅是传送移位寄存器的第 9 位。

写到 SBUF 的信号请求 UART 引擎加载 TB8 到发送移位寄存器的第 9 位并开始发送，当收到一个发送请求后，UART 将在 TX 时钟的上升沿开始发送。SBUF 中的数据从 TXD 引脚串行输出，数据帧如图 16–2 所示及数据宽度根据 TX 时钟不同而不同。当 9 位数据发送完后，硬件将置位 TI 表示发送结束。

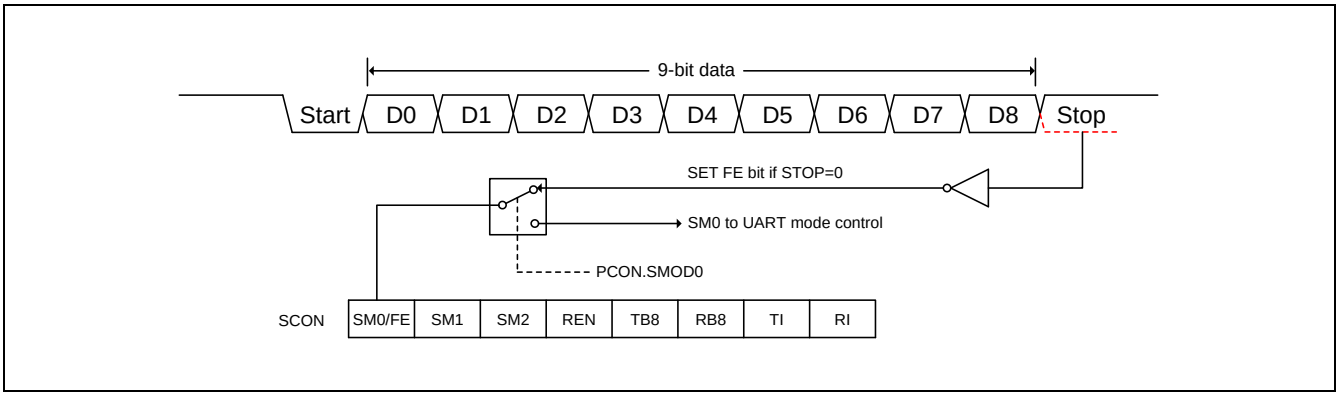
当串行口控制器在 RCK 采样时钟下检测到在 RXD 有 1 到 0 跳变的起始位时接收开始。在 RXD 引脚上的数据将被串行口的位侦查器采样。当收数据接收完后，硬件置位 RI 表示接收结束并把第 9 位加载到 SCON 寄存器的 RB8。

在四种模式中，使用 SBUF 作为一个目的寄存器，可以通过任何指令发起传输。在模式 0，当 RI=0 且 REN=1 时启动接收。在其它模式，在 REN=1 时，收到有 1 到 0 跳变的起始位时启动接收。

16.4. 帧错误检测

开启帧错误检测功能后，UART会在通讯中检测是否丢失停止位，如果丢失一个停止位，就设置SCON寄存器的FE标志位。FE标志位和SM0标志位共享SCON.7，SMOD0标志位(PCON.6)决定SCON.7究竟代表哪个标志，如果SMOD0位（PCON.6）置位则SCON.7就是FE标志，SMOD0位清零则SCON.7就是SM0标志。当SCON.7代表FE时，只能软件清零。参考图 16–7.。

图 16–7. UART 帧错误检测



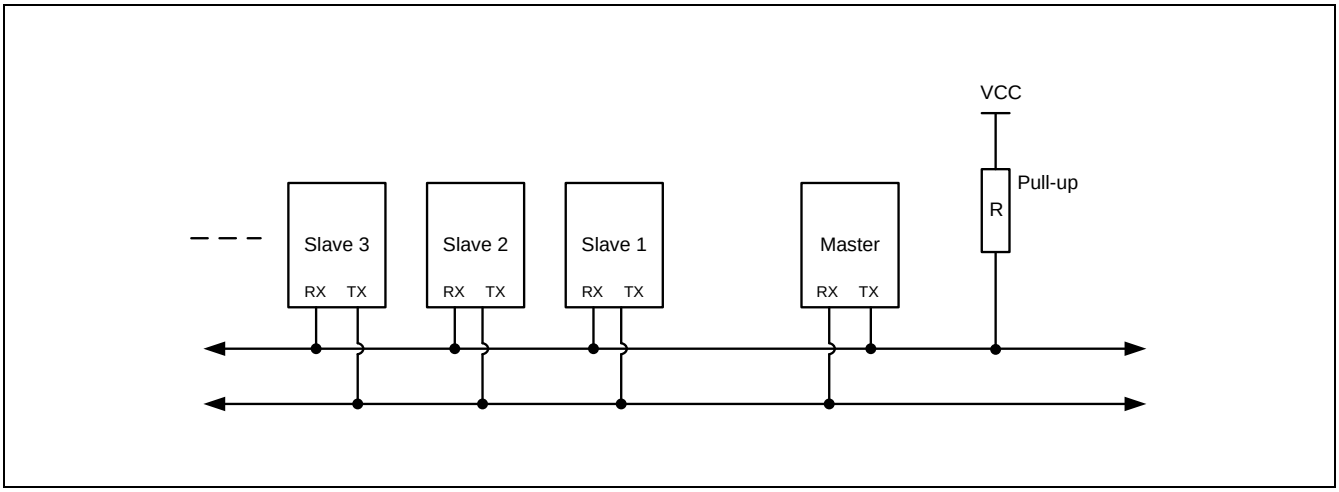
16.5. 多处理器通讯

模式 2 和 3 在用作多处理器通讯时有特殊的规定见图 16–8.。在这两种模式，接收 9 个数据位。第 9 个数据位存入 RB8，接着进来一个停止位。端口可以编程为：在 RB8=1 时，当收到停止位后，串口中断将激活。这种特征通过设置 SM2 位(在 SCON 寄存器中)来使能。这种方式用于多处理器系统如下：

当主处理器想传送一个数据块到多个从机中的某一个时，首先传送想要传送的目标地址标识符的地址。地址字节与数据字节的区别在于，在地址字节中第 9 位为 1，数据字节中为 0。当 SM2=1 时，收到一个数据字节将不会产生中断。然而一个地址字节将引发所有从机中断。因而所有的从机可以检测收到的字节是否是自己的地址。从机地址将清除 SM2 位并准备好接收即将进来的所有数据。从机地址不匹配的将保持 SM2 置位，并继续他们的工作，忽略进来的数据字节。

SM2 在模式 0 和模式 1 没有影响，但是可以用来检测停止位的有效性。在接收模式 1 中，如果 SM2=1，除非收到一个有效的停止位否则接收中断不会被激活

图 16–8. UART 多处理器通讯



16.6. 自动地址识别

自动地址识别通过硬件比较可以让 UART 识别串行码流中的地址部分，该功能免去了使用软件识别时需要大量代码的麻烦。该功能通过置位 SCON 的 SM2 位来开启。

在 9 位数据 UART 模式下，即模式 2 和模式 3，收到特定地址或广播地址时自动置位接收中断(RI)标志，9 位模式的第 9 位信息为 1 表明接收的是一个地址而不是数据。自动地址识别功能请参考图 16–9。在 8 位模式，即模式 1 下，如果 SM2 置位并且在 8 位地址与给定地址或广播地址核对一致后收到有效停止位则 RI 置位。 模式 0 是移位寄存器模式，SM20 被忽略。

使用自动地址识别功能可以让一个主机选择性的同一个或多个从机进行通讯，所有从机可以使用广播地址接收信息。增加了 SADDR 从机地址寄存器和 SADEN 地址掩码寄存器。

SADEN 用来定义 SADDR 中的那些位是“无关紧要”的，SADEN 掩码和 SADDR 寄存器进行逻辑与来定义供主机寻址从机的“给定” 地址，该地址让多个从机进行排他性的识别。

下面的实例帮助理解这个方案的通用性:

从机 0	从机 1
SADDR = 1100 0000	SADDR = 1100 0000
SADEN = 1111 1101	SADEN = 1111 1110
特定地址= 1100 00X0	特定地址= 1100 000X

上例中 SADDR 相同，而 SADEN 不同以区分两个分机。从机 0 要求位 0 等于 0，并忽略位 1。 从机 1 要求位 1 等于 0，并忽略位 0。从机 0 的唯一地址是 1100 0010，因为从机 1 要求位 1 等于 0。从机 1 的唯一地址是 1100 0001，因为位 0 等于 1 可以屏蔽从机 0。一个地址的位 0=0 并且位 1=0，可以同时选择两个从机。因此，地址 1100 0000 可以同时寻址从机 0 和从机 1。

下面是一个更为复杂的系统，可选择从机 0 和从机 1，而屏蔽从机 2：

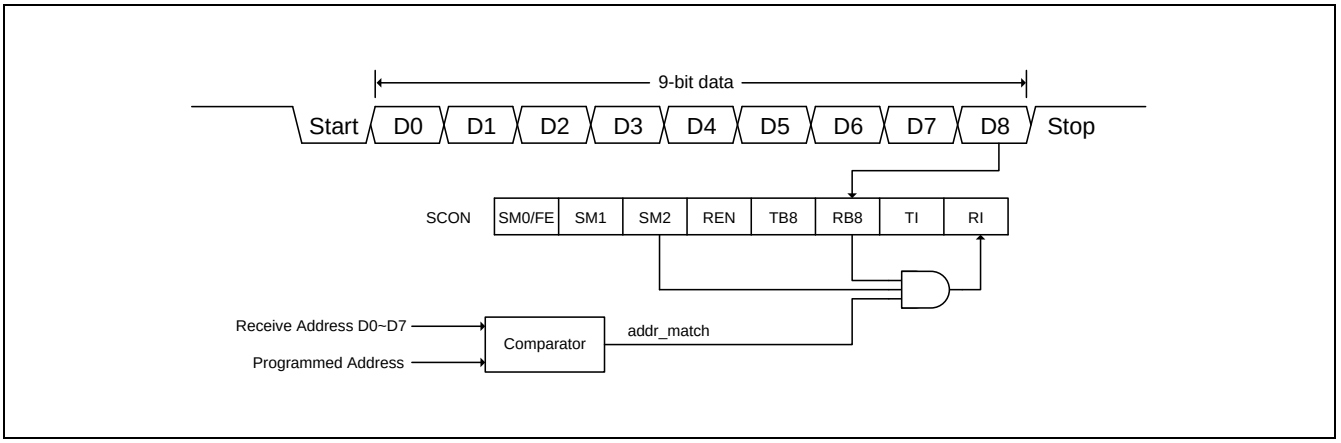
从机 0	从机 1	从机 2
SADDR = 1100 0000	SADDR = 1110 0000	SADDR = 1110 0000
SADEN = 1111 1001	SADEN = 1111 1010	SADEN = 1111 1100
特定地址= 1100 0XX0	特定地址= 1110 0X0X	特定地址= 1110 00XX

上述例子中，三个从机地址只有低 3 位不同。从机 0 要求位 0 等于 0，可通过 1110 0110 单独寻址。从机 1 要求位 1 等于 0，可通过 1110 0101 单独寻址。从机 2 要求位 2 等于 0，可通过 1110 0011 单独寻址。为了选择从机 0 和从机 1，同时屏蔽从机 2，可以使用地址 1110 0100，因为要屏蔽从机 2，这个地址的位 2 必须为 1。

每个从机的广播地址是由 SADDR 和 SADEN 逻辑或所得，结果为 0 的位视为无关位。大多数情况下，无关位被认为是 1，这样广播地址就是 0xFF。.

复位后，SADDR(SFR 地址 0xA9)和 SADEN(SFR 地址 0xB9)的值均为 0，这样就产生了一个所有位都是无关位的特定地址，和所有位都是无关位的广播地址一样。这实际上禁止了自动地址识别模式，从而可让微处理器使用没有这个功能的标准 80C51 的 UART。

图 16–9. 自动地址识别



注意:

(1) 地址匹配后(addr_match=1), 清除 SM2 以接收数据字节

(2) 收完全部数据字节后,置 SM2 为 1 以等待下一个地址

16.7. 波特率设置

AUXR2寄存器的位T1X12和URM0X3为波特率设置提供了一个新的选项。如下所示：

16.7.1. 模式 0 的波特率

图 16–10. 模式 0 波特率计算公式

$$\text{Mode 0 Baud Rate} = \frac{F_{SYSCLK}}{n} \quad \begin{array}{l} ; \text{n=12, if URM0X3=0} \\ ; \text{n=4, if URM0X3=1} \end{array}$$

注意: 如果 URM0X3=0,波特率公式和标准 8051 一样。

表 16–1. 串行口模式 0 波特率范例

系统时钟	URM0X3	模式 0 波特率
12MHz	0	1M bps
12MHz	1	3M bps
24MHz	0	2M bps
24MHz	1	6M bps

16.7.2. 模式 2 的波特率

图 16–11. 模式 2 波特率计算公式

$$\text{Mode 2 Baud Rate} = \frac{2^{SMOD1}}{64} \times (F_{SYSCLK})$$

表 16–2. 串行口模式 2 波特率范例

系统时钟	SMOD1	模式 2 波特率
22.1184MHz	0	345.6K bps
22.1184MHz	1	172.8K bps
24MHz	0	750K bps
24MHz	1	375K bps

16.7.3. 模式 1 和 3 的波特率

使用定时器 1 作为波特率发生器, 图 16–12. 模式 1/3 波特率计算公式

$$\text{Mode 1, 3 Baud Rate} = \frac{2^{SMOD1}}{32} \times \frac{F_{SYSCLK}}{n \times (256 - TH1)} \quad \begin{array}{l} ; \text{n=12, if T1X12=0} \\ ; \text{n=1, if T1X12=1} \end{array}$$

表 16-1 到表 16-6 例举了各种常用的波特率及它们如何从定时器 1 的 8 位自动装载模式获取得到。

表 16–3. 定时器 1 在 F_{SYSCLK}=11.0592MHz 时产生常用波特率

波特率	**TH1**, 自动装载值					
	T1X12=0			**T1X12=1**		
	SMOD1=0	SMOD1=1	误差	SMOD1=0	SMOD1=1	误差
1200	232	208	0.0%	--	--	--
2400	244	232	0.0%	112	--	0.0%
4800	250	244	0.0%	184	112	0.0%
9600	253	250	0.0%	220	184	0.0%
14400	254	252	0.0%	232	208	0.0%
19200	--	253	0.0%	238	220	0.0%
28800	255	254	0.0%	244	232	0.0%
38400	--	--	--	247	238	0.0%
57600	--	255	0.0%	250	244	0.0%
115200	--	--	--	253	250	0.0%
230400	--	--	--	--	253	0.0%

表 16–4. 定时器 1 在 F_{SYSCLK}=22.1184MHz 时产生常用波特率

波特率	**TH1**, 自动装载值					
	T1X12=0			**T1X12=1**		
	SMOD1=0	SMOD1=1	误差	SMOD1=0	SMOD1=1	误差
1200	208	160	0.0%	--	--	--
2400	232	208	0.0%	--	--	0.0%
4800	244	232	0.0%	112	--	0.0%
9600	250	244	0.0%	184	112	0.0%
14400	252	248	0.0%	208	160	0.0%
19200	253	250	0.0%	220	184	0.0%
28800	254	252	0.0%	232	208	0.0%
38400	--	253	0.0%	238	220	0.0%
57600	255	254	0.0%	244	232	0.0%
115200	--	255	0.0%	250	244	0.0%
230400	--	--	--	253	250	0.0%
460800	--	--	--	--	253	0.0%

表 16–5. 定时器 1 在 F_{SYSCLK}=12.0MHz 时产生常用波特率

波特率	**TH1**, 自动装载值					
	T1X12=0			**T1X12=1**		
	SMOD=0	SMOD=1	误差	SMOD=0	SMOD=1	误差
1200	230	204	0.16%	--	--	--
2400	243	230	0.16%	100	--	0.16%
4800	--	243	0.16%	178	100	0.16%
9600	--	--	--	217	178	0.16%
14400	--	--	--	230	204	0.16%
19200	--	--	--	--	217	0.16%
28800	--	--	--	243	230	0.16%
38400	--	--	--	246	236	2.34%
57600	--	--	--	--	243	0.16%
115200	--	--	--	--	--	--

表 16–6. 定时器 1 在 F_{SYSCLK}=24.0MHz 时产生常用波特率

波特率	**TH1**, 自动装载值					
	T1X12=0			**T1X12=1**		
	SMOD=0	SMOD=1	误差	SMOD=0	SMOD=1	误差
1200	204	152	0.16%	--	--	--
2400	230	204	0.16%	--	--	--
4800	243	230	0.16%	100	--	0.16%
9600	--	243	0.16%	178	100	0.16%
14400	--	--	--	204	152	0.16%
19200	--	--	--	217	178	0.16%
28800	--	--	--	230	204	0.16%
38400	--	--	--	--	217	0.16%
57600	--	--	--	243	230	0.16%
115200	--	--	--	--	243	0.16%

16.8. 串行口模式 4 (SPI 主机)

MA86E/L508 串行口嵌入了一个额外的模式 4 支持 SPI 主机引擎。模式 4 由 SM3, SM0 和 SM1 选择。表 16–7 展示了 **MA86E/L508** 的串行口模式定义。

表 16–7. 串行口模式选择

SM31	SM01	SM11	模式	描述	波特率
0	0	0	0	移位寄存器	SYSCLK/12 or SYSCLK/4
0	0	1	1	8-bit UART	可变
0	1	0	2	9-bit UART	SYSCLK/64, /32
0	1	1	3	9-bit UART	可变
1	0	0	4	**SPI 主机**	SYSCLK/12 or SYSCLK/4
1	0	1	5	保留	保留
1	1	0	6	保留	保留
1	1	1	7	保留	保留

URM0X3 也是控制 SPI 的传输速度。如果 URM0X3 = 0,则 SPI 的时钟频率是 SYSCLK/12。如果 URM0X3 = 1, 则 SPI 的时钟频率是 SYSCLK/4。

MA86E/L508 的 SPI 主机使用 TXD 作为 SPICLK, RXD 作为 MOSI,以及 S0MI 作为 MISO。而 nSS 由 MCU 软件选择在其它端口引脚。 图 16–13 展示了 SPI 连接。他支持多从机通讯架构见

图 16–13. 串行口模式 4,单主机和单从机架构

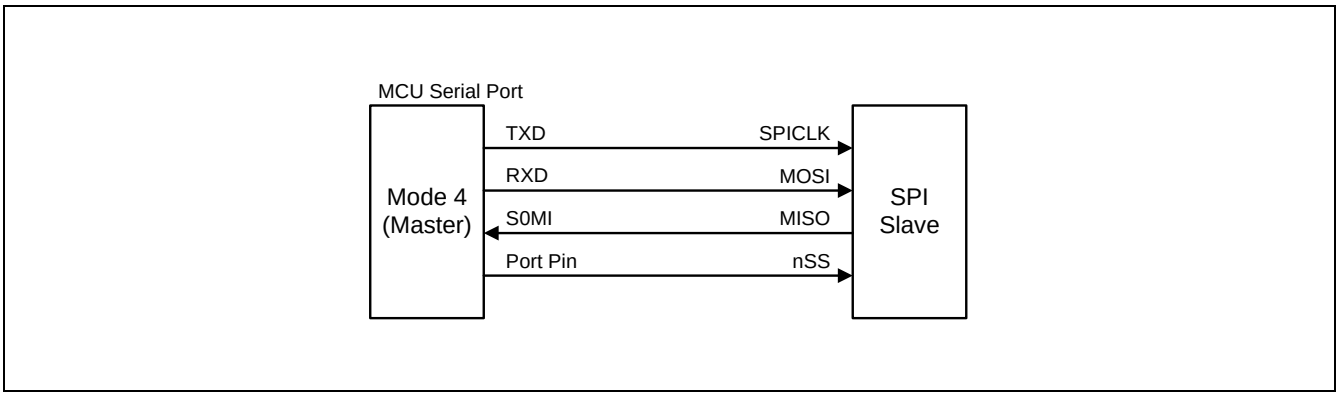
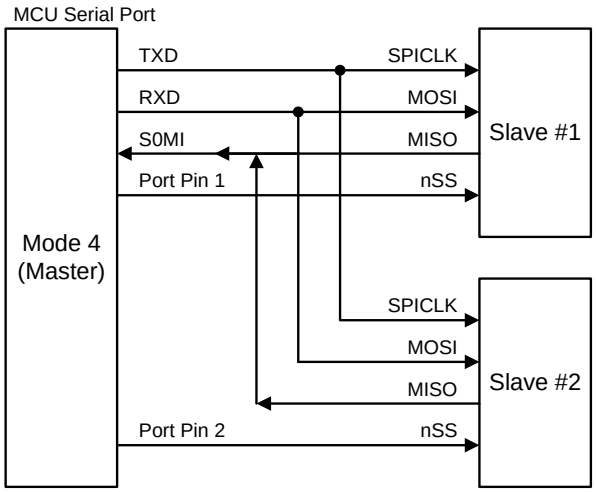


图 16–14. 串行口模式 4, 单主机和多从机架构



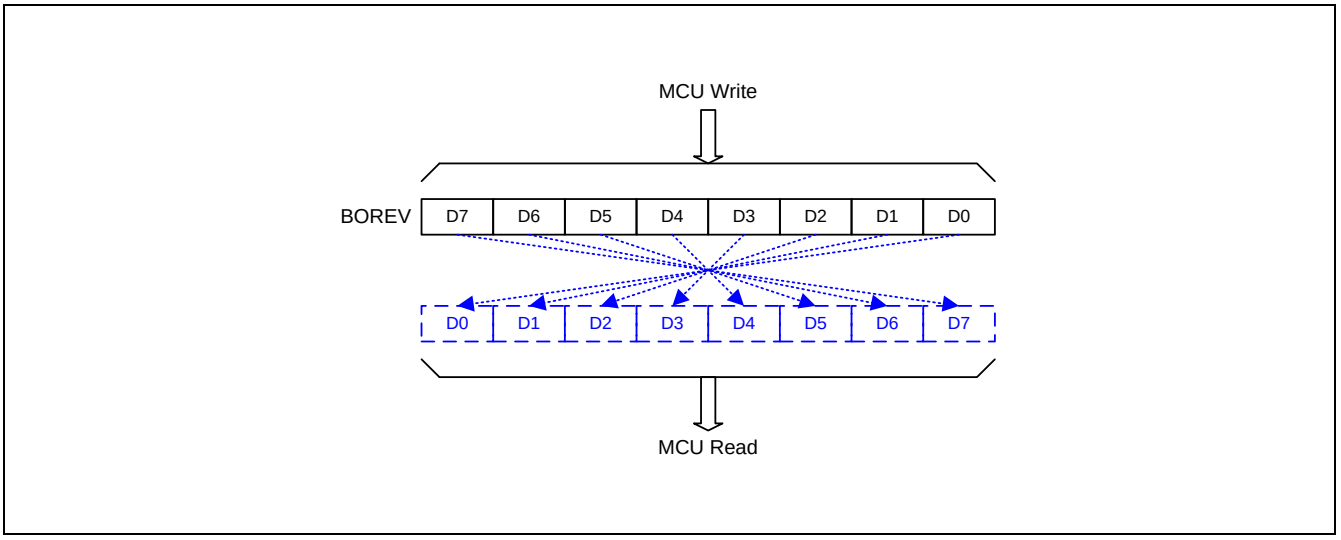
SPI 主机能满足笙泉 MA82/84 系列 MCU（由 CPOL, CPHA 和 DORD 选择）的全功能 SPI 模块的传输。在 CPOL 和 CPHA 条件下, **MA86E/L508** 很容易初始化 SPI 的时钟(TXD/P3.1)极性去适合他们使用 。 表 16–8 展示了串行口模式 4 的 4 个 SPI 工作模式。

表 16–8. 串行口模式 4 的 SPI 模式结构

SPI 模式	CPOL	CPHA	**MA86E/L508** 结构
0	0	0	清除 P3.1 为“0”
1	0	1	清除 P3.1 为“0”
2	1	0	设置 P3.1 为“1”
3	1	1	设置 P3.1 为“1”

SPI 系列传输的位序控制(DORD)，**MA86E/L508** 提供了一个软件编程可以相反的位序控制（SFR, BOREV）。 在 MCU 写一个数据格式（MSB）到 BOREV ， MCU 通过读 BOREV 得到一个数据格式（LSB）回来。SPI 主机引擎在串行口模式 4 与串行口模式 0 一样都是(LSB)传输。为了支持 SPI(MSB)移位传输, MCU 必须使用(BOREV)写/读取操作来翻转 SPI 输入/输出传送的数据位序。图 16–15 展示了 BOREV 结构。

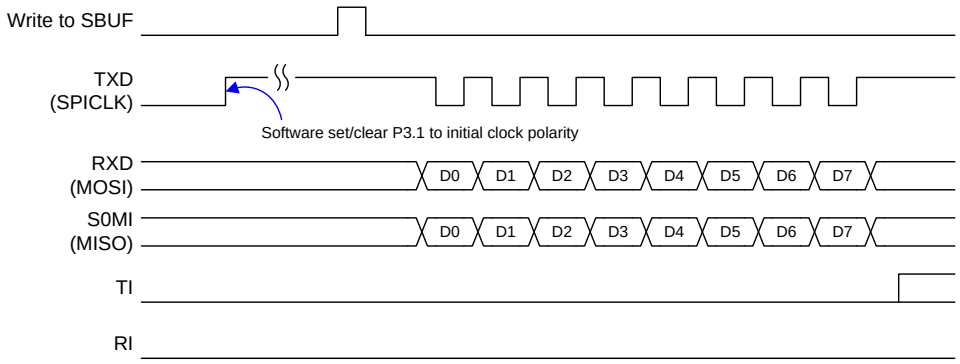
图 16–15. SFR BOREV 读取/写结构



通过指令使用 SBUF 作为目标寄存器初始化传送。“写数据到 SBUF”触发 UART 引擎开始传送。SBUF 的数据被移位到作为 MOSI 串口数据的 RXD 引脚。SPI 移位时钟由作为 SPICLK 输出的 TXD 引脚输出。在 8 个移位时钟的上升沿之后，TI 被硬件声明传送结束。同时 S0MI 引脚的内容被采样并且移位到移位寄存器。然后“读取 SBUF”能获取 SPI 的移入数据。

图 16–16 展示了模式 4 传送波形。RI 在模式 4 不被声明。

图 16–16. 串行口模式 4 传送波形



16.9. 串行口寄存器

串行口的四种操作模式除波特率的设定之外都与标准的 8051 相同。此三个寄存器 PCON, AUXR 和 AUXR2 是与波特率的设定有关

SCON: 串行口控制寄存器

SFR 页 = 普通

SFR 地址 = 0x98 复位值 = 0000-0000

7	6	5	4	3	2	1	0
SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: FE, 帧错误位。SMOD0 必须置位才能访问 FE 位。

0: FE 位不会被有效的帧清零，它应当被软件清零。

1: 当检测到一个无效的停止位时，该位被接收器置位。

Bit 7: 串行口模式位 0， (SMOD0 必须 = 0 才能访问位 SM0)

Bit 6: 串行口模式位 1

SM31	SM01	SM11	模式	描述	波特率
0	0	0	0	移位寄存器	SYSCLK/12 or SYSCLK/4
0	0	1	1	8-bit UART	可变
0	1	0	2	9-bit UART	SYSCLK/64, /32
0	1	1	3	9-bit UART	可变
1	0	0	4	SPI 主机	SYSCLK/12 or SYSCLK/4
1	0	1	5	保留	保留
1	1	0	6	保留	保留
1	1	1	7	保留	保留

Bit 5: 串行口模式位 2

0: 禁止 SM2 功能

1: 在模式 2 和 3 时使能地址自动识别，如果 SM2=1 那么 RI 将不被置位，除非接收到的第 9 位数据(RB8)为 1，表示是一个地址，并且接收到的字节是特定地址或者是一个广播地址；在模式 1，如果 SM2=1 那么 RI0 将不能被激活除非收到一个有效的停止位，并且接收到的字节是特定地址或者是一个广播地址；在模式 0，SM2 应为 0。

Bit 4: REN, 使能串行接收

0: 软件清零是禁止串行接收。

1: 软件置位是使能串行接收

Bit 3: TB8, 在模式 2 和 3 下此位是被传送数据的第 9 位。由软件清零或置位。

Bit 2: RB8, 在模式 2 和 3 下此位是接收到数据的第 9 位。在模式 1 下，如果 SM2 = 0, RB8 则是接收到的停止位。在模式 0，RB8 没有用到。

Bit 1: TI. 传送中断标志位

0: 必须由软件清零。

1: 在模式 0 下第 8 位传送时间结束硬件设置，在其它模式下传送停止位开始硬件设置。

Bit 0: RI. 接收中断标志位

0: 必须由软件清零。

1: 在模式 0 下第 8 位接收时间结束硬件置位，在其它模式下接收停止位一半时开始硬件置位(除了关注 SM2 之外)。

SBUF: 串口缓冲寄存器

SFR 页 =普通

SFR 地址 = 0x99 复位值= XXXX-XXXX

7	6	5	4	3	2	1	0
SBUF.7	SBUF.6	SBUF.5	SBUF.4	SBUF.3	SBUF.2	SBUF.1	SBUF.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~0: 作为传送和接收的缓冲寄存器。.

SADDR: 从地址寄存器

SFR 页 =普通

SFR 地址 = 0xA9 复位值= 0000-0000

7	6	5	4	3	2	1	0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

SADEN: 从地址屏蔽寄存器

SFR 页 =普通

SFR 地址 = 0xB9 复位值= 0000-0000

7	6	5	4	3	2	1	0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

SADDR 寄存器 和 SADEN 寄存器结合形成用于自动地址识别的特定/广播地址。实际上，SADEN 作为用于 SADDR 寄存器的“屏蔽”寄存器功能。如下所示：

SADDR = 1100 0000

SADEN = 1111 1101

特定地址 = 1100 00x0 → 这特定的从机地址将被选中，除了位 1 作不关心处理外

每个从设备的广播地址是 SADDR 和 SADEN 进行逻辑“或”的结果，结果中为“0”的位将被忽略。在系统复位后，SADDR 和 SADEN 都被初始化为 0，从而“不在乎”所有的特定地址和“不在乎”所有的广播地址，而导致自动地址识别功能无效。

PCON0: 电源控制寄存器 0

SFR 页 =普通及 P 页

SFR 地址 = 0x87 POR = 00X1-0000, 复位值= 00X0-0000

7	6	5	4	3	2	1	0
SMOD1	SMOD0	GF	POF	GF1	GF0	PD	IDL
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: SMOD1, 双倍波特率控制位

0:禁止 UART 的双倍波特率

1: 使能 UART 在模式 1，2，3 的双倍波特率

Bit 6: SMOD0, 帧错误选择.

0: SCON.7 是 SM0 功能

1: SCON.7 是 FE 功能。注意，在一个帧错误后，FE 将被置位，不管 SMOD0 的状态。

AUXR2: 辅助寄存器 2

SFR 页 =普通

SFR 地址 = 0xA3 复位值= 0000-0000

7	6	5	4	3	2	1	0
--	BTI	URM0X3	SM3	T1X12	T0X12	T1CKOE	T0CKOE
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 6: BTI, 阻止 TI 在串行口中断

0: 保留 TI 作为串行口中断源。

1: 阻止 TI 作为串行口中断源。

Bit 5: URM0X3, 串行口模式 0 和模式 4 波特率选择

0: 清零选择 SYSCLK/12 作为 UART 模式 0 和模式 4 的波特率。

1: 置位选择 SYSCLK/4 作为 UART 模式 0 和模式 4 的波特率。

Bit 4: SM3, 串行口模式控制位 3

0: 禁止串行口模式 4

1:使能 SM3 去控制串行口模式 4，SPI 主

Bit 3: T1X12, 当 C/T=0 时定时器 1 时钟源选择

0: 清零选择 SYSCLK/12.

1: 置位选择 SYSCLK 作为时钟源

SFIE: 系统标志中断使能寄存器

SFR 页 =普通

SFR 地址 = 0x8E POR = 00x0-0x00

7	6	5	4	3	2	1	0
UTIE	SDIFIE	--	RTCFIE	KBIFIE	--	BOF0IE	WDTFIE
R/W	**R/W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 7: UTIE, 在系统标志中断里使能 UART TI

0: 禁止在系统标志中断里中断向量共享给 TI

1: 设置 TI 标志将与系统标志中断共享中断向量

BOREV: 位序相反寄存器

SFR 页 =普通

SFR 地址 = 0x96 复位值= 0000-0000

7	6	5	4	3	2	1	0
BOREV.7	BOREV.6	BOREV.5	BOREV.4	BOREV.3	BOREV.2	BOREV.1	BOREV.0
W	**W**	**W**	**W**	**W**	**W**	**W**	**W**
BOREV.0	BOREV.1	BOREV.2	BOREV.3	BOREV.4	BOREV.5	BOREV.6	BOREV.7
R	**R**	**R**	**R**	**R**	**R**	**R**	**R**

这个寄存器作用于读取数据跟写入数据位序相反的功能。

16.10. 串行口示例代码

(1). 规定功能:串行口输入 RI 唤醒空闲模式

汇编语言代码范例:

```
    ORG     00023h
uart_ri_idle_isr:
    JB      RI,RI_ISR                   ;判断是否串行输入中断
    JB      TI,TI_ISR                   ;判断是否串行发送中断
    RETI                                ;中断返回

RI_ISR:
;   中断处理
    CLR     RI                          ;清除 RI 标志
    RETI                                ;中断返回

TI_ISR:
;   中断处理
    CLR     TI                          ; 清除 TI 标志
    RETI                                ; 中断返回

main:
    CLR     TI                          ; 清除 TI 标志
    CLR     RI                          ; 清除 RI 标志
    SETB    SM1                         ;
    SETB    REN                         ; 8 位的模式 2，接收使能

    MOV     IP0L,#PSL                   ; 选择串行口 S0 中断优先级
    MOV     IP0H,#PSH                   ;

    SETB    ES                          ; 使能串行口 S0 中断
    SETB    EA                          ; 使能全局中断

    ORL     PCON0,#IDL;                 ; 设置 MCU 进入空闲模式
```

C 语言代码范例:

```
void uart_ri_idle_isr(void) interrupt 4
{    if(RI)                                  //判断是否串行输入中断
     {
          RI=0;                              //清除 RI 标志
          // to do ...
     }

     if(TI)                                  //判断是否串行发送中断
     {
          TI=0;                              //清除 TI 标志
          // to do ...
     }
}

void main(void)
{
     TI = RI = 0;                            //清除 TI 和 RI 标志
     SM1 = REN = 1;                          // 8 位的模式 2，接收使能

     IP0L = PSL;                             //选择串行口 S0 中断优先级
     IP0H = PSH;                             //

     ES = 1;                                 //使能串行口 S0 中断
     EA = 1;                                 //  使能全局中断

     PCON |= IDL;                            //设置 MCU 进入空闲模式
}
```

(2). 规定功能: 串行口 S0 模式 4 工作在 SPI 主机模式 0/1, SYSCLK = 24MHz

汇编语言代码范例:

```
;    SETB     nSS                           ; 用户定义通用输入输出口(GPIO)
     CLR      P31                           ; 对 SPI CPOL=0, SPICLK 初始状态是逻辑低
                                            ; CPHA = 0 或 1

; 初始化
     MOV      P3M1,#0x03                    ; 设置 P3.1 (SPICLK)为推挽输出模式
```

```
    MOV     P3M0,#0x00              ; 设置 P3.0 (MOSI) 为推挽输出模式
                                    ; P37 (MISO) 默认为准双向模式

    ANL     AUXR2,#~URM0X3          ; SPICLK = SYSCLK/12 = 24MHz/12 = 2MHz
;   ORL     AUXR2,#URM0X3           ; 或 SPICLK = SYSCLK/4 = 24MHz/4 = 6MHz

    ORL     AUXR2,#SM3              ; 设置串行口 S0 模式 4 为 SPI 主机工作
    MOV     SCON,#0x00

;   传送数据
;   CLR     nSS                     ; 用户定义通用输入输出口(GPIO)
    MOV     SBUF,#0x55              ; 写 1st SPI 数据
    JNB     TI,$                    ; 等待 SPI 主机发送结束
    MOV     data_reg,SBUF           ; 读取 1st SPI 输入数据
    CLR     TI

    MOV     SBUF,#0xAA;             ; 写 2nd SPI 数据
    JNB     TI,$                    ; 等待 SPI 主机发送结束
    MOV     data_reg,SBUF           ; 读取 2nd SPI 输入数据
    CLR     TI
;   SETB    nSS                     ; 用户定义通用输入输出口(GPIO)
```

C 语言代码范例:

```
//  nSS=High;                       //用户定义通用输入输出口(GPIO)
    P31 = 0;                        //对 SPI CPOL=0, SPICLK 初始状态是逻辑低
                                    // CPHA = 0 或 1

    P3M1 = 0x03;                    //设置 P3.1 (SPICLK)为推挽输出模式
    P3M0 = 0x00;                    //设置 P3.0 (MOSI) 为推挽输出模式
                                    // P37 (MISO) 默认为准双向模式

    AUXR2 &= ~URM0X3;               // SPICLK = SYSCLK/12 = 24MHz/12 = 2MHz
//  AUXR2 |= URM0X3;                //或 SPICLK = SYSCLK/4 = 24MHz/4 = 6MHz

    AUXR2 |= SM3;                   //设置串行口 S0 模式 4 为 SPI 主机工作
    SCON = 0x00;
```

```
//      Transtor Data
//      nSS = Low;                          //用户定义通用输入输出口(GPIO)
        SBUF = 0x55;                        // 写 1st SPI 数据
        while(!TI);                         //等待 SPI 主机发送结束
        reg = SBUF;                         // 读取 1st SPI 输入数据
        TI = 0;

        SBUF = 0xAA;                        // 写 2nd SPI 数据
        while(!TI);                         //等待 SPI 主机发送结束
        reg = SBUF;                         // 读取 2nd SPI 输入数据
        TI = 0;

        //nSS = High;                       //用户定义通用输入输出口(GPIO)
```

(3). 规定功能: 串行口 S0 模式 4 工作在 SPI 主机模式 2/3, SYSCLK = 24MHz, ***MSB 高位在先***

汇编语言代码范例:

```
;    SETB     nSS                          ; 用户定义通用输入输出口(GPIO)
     SETB     P31                          ; 对 SPI CPOL=1, SPICLK 初始状态是逻辑高
                                           ; CPHA = 0 或 1

; initial
     MOV      P3M1,#0x03                   ; 设置 P3.1 (SPICLK)为推挽输出模式
     MOV      P3M0,#0x00                   ; 设置 P3.0 (MOSI) 为推挽输出模式
                                           ; P37 (MISO) 默认为准双向模式

     ANL      AUXR2,#~URM0X3               ; SPICLK = SYSCLK/12 = 24MHz/12 = 2MHz
;    ORL      AUXR2,#URM0X3                ; 或 SPICLK = SYSCLK/4 = 24MHz/4 = 6MHz

     ORL      AUXR2,#SM3                   ; 设置串行口 S0 模式 4 为 SPI 主机工作
     MOV      SCON,#0x00

;  Transtor Data
;    CLR      nSS                          ; 用户定义通用输入输出口(GPIO)
     MOV      BOREV,#0x5A                  ; 相反的数据输出位序为 MSB 高位在先
     MOV      SBUF,BOREV                   ; 写 1st SPI 数据
```

```
    JNB     TI,$                        ; 等待 SPI 主机发送结束
    MOV     BOREV,SBUF                  ; 读取 1st SPI 输入数据
    MOV     data_reg,BOREV              ; 相反的数据输入位序为 LSB 低位在先
    CLR     TI

    MOV     BOREV,#0xA5;                ; 相反的数据输出位序为 MSB 高位在先
    MOV     SBUF,BOREV;                 ; 写 2nd SPI 数据

    JNB     TI,$                        ; 等待 SPI 主机发送结束
    MOV     BOREV,SBUF                  ; 读取 2nd SPI 输入数据
    MOV     data_reg,BOREV              ; 相反的数据输入位序为 LSB 低位在先
    CLR     TI
;   SETB    nSS                         ; 用户定义通用输入输出口(GPIO)
```

C 语言代码范例:

```
    unsigned char reg;
//  nSS=High;                           //用户定义通用输入输出口(GPIO)
    P31 = 1;                            //对 SPI CPOL=1, SPICLK 初始状态是逻辑高
                                        // CPHA = 0 或 1

    P3M1 = 0x03;                        //设置 P3.1 (SPICLK)为推挽输出模式
    P3M0 = 0x00;                        //设置 P3.0 (MOSI) 为推挽输出模式
                                        // P37 (MISO) 默认为准双向模式

    AUXR2 &= ~URM0X3;                   // SPICLK = SYSCLK/12 = 24MHz/12 = 2MHz
//  AUXR2 |= URM0X3;                    // 或 SPICLK = SYSCLK/4 = 24MHz/4 = 6MHz

    AUXR2 |= SM3;                       //设置串行口 S0 模式 4 为 SPI 主机工作
    SCON = 0x00;

//    Transtor Data
//    nSS = Low;                        //用户定义通用输入输出口(GPIO)
    BOREV = 0x5A;                       //相反的数据输出位序为 MSB 高位在先
    _nop_ ();                           // 防止 C 编译器优化
    SBUF = BOREV;                       // 写 1st SPI 数据
```

```
while(!TI);                          //等待 SPI 主机发送结束
BOREV = SBUF;                        // 读取 1st SPI 输入数据
_nop_ ();                            //防止 C 编译器优化
reg = BOREV;                         //相反的数据输入位序为 LSB 低位在先
TI = 0;

BOREV = 0xA5;                        //相反的数据输出位序为 MSB 高位在先
_nop_ ();                            //防止 C 编译器优化
SBUF = BOREV;                        // 写 2nd SPI 数据
while(!TI);                          //等待 SPI 主机发送结束
BOREV = SBUF;                        // 读取 2nd SPI 输入数据
_nop_ ();                            //防止 C 编译器优化
reg = BOREV;                         //相反的数据输入位序为 LSB 低位在先
TI = 0;
//nSS = High;                        //用户定义通用输入输出口(GPIO)
```

17. 可编程计数器阵列(PCA)

MA86E/L508 预备了一个可编程计数器阵列 (PCA), 它与标准的定时器/计数器相比用较少的 CPU 干预提供了更多的时间功能。它的优点包括降低软件开销和提供精确度。

17.1. PCA 概述

PCA 包含一个专用的定时器/计数器，它给一个系列的 4 个比较/捕获模块提供时基服务。图 17-1 列出了 PCA 的方框图。注意 PCA 定时器和模块都是 16 位的。如果一个外部事件被关联到一个模块，那它功能与相对应的引脚共享。如果模块没有用到引脚，则这引脚仍用于标准 I/O 口。

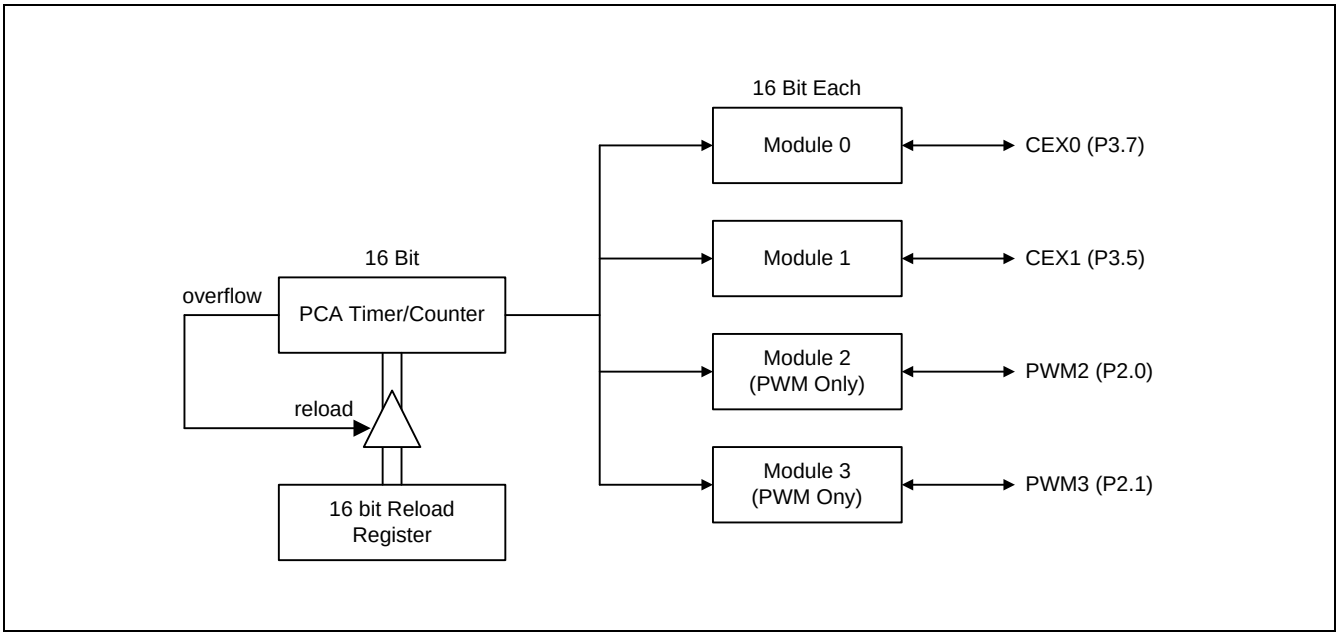
模块 0 和模块 1 可以被编程为下列模式的任一种：

- 上升和/或下将沿捕获
- 软件定时器
- 高速输出
- 脉宽调制 (PWM) 输出

模块 2 和模块 3 仅支持 PWM 输出模式。

所有模式将在之后详细说明。然而，让我们首先看看怎样设置 PCA 定时器和模块。

图 17–1. PCA 方框图



17.2. PCA 定时器/计数器

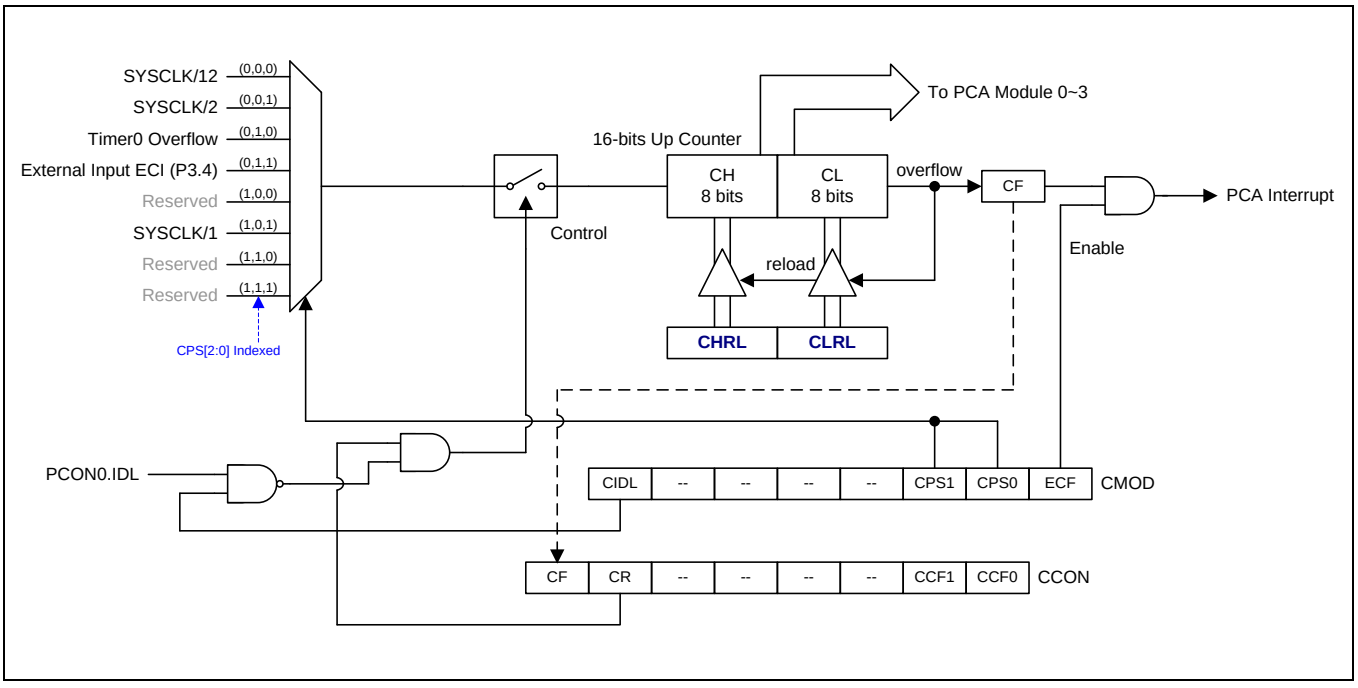
PCA 定时器/计数器是一个自动装载的 16 位定时器包含 CH 和 CL 寄存器(计数值的高和低字节)，CHRL, CLRL（重装载值的高和低字节）如图 17–2 所示。CHRL, CLRL 在 CH+CL 计数器每一次溢出时会自动装载到 CH，CL；这样能改变 PCA 的周期给不同的 PWM 分辨率，比如 7 位的 PWM。

它是所有模块的通用时基，它时钟输入可以由下列源选择：

- 1/12 系统时钟频率
- 1/2 系统时钟频率
- 定时器 0 溢出,它允许一个慢些时钟输入范围给定时器
- 外部时钟输入, 1 到 0 变化，在 ECI 引脚 (P3.4).
- 直接来自系统时钟频率

特殊功能寄存器 CMOD 包含用于 PCA 定时器输入的计数脉冲选择位(**CPS2,** CPS1 和 CPS0) 。这个寄存器也包括 ECF 位，当计数器溢出，它就使能一个中断。另外，用户有在 Idle 模式是否运行 PCA 计数器的选项，通过设置计数器 Idle 位(CIDL)来选择。这能进一步的在 Idle 模式下降低功耗。

图 17–2. PCA 定时器/计数器



CMOD: PCA 计数器模式寄存器

SFR 页 =普通

SFR 地址 = 0xD9 复位值= 0xxx-0000

7	6	5	4	3	2	1	0
CIDL	--	--	--	**CPS2**	CPS1	CPS0	ECF
R/W	**W**	**W**	**W**	**W**	**R/W**	**R/W**	**R/W**

Bit 7: CIDL, PCA 计数器 Idle 控制

0:在 Idle 模式下 PCA 计数器继续运行

1: 在 Idle 模式下关闭 PCA 计数器

Bit 6~4: 保留。当写 CMOD 时，软件必须在这些位上写”0”。

Bit 3~1: CPS1-CPS0, PCA 计数器时钟源选择位，**CPS2** 只能写

CPS2	CPS1	CPS0	PCA 时钟源
0	0	0	内部时钟, SYSCLK/12
0	0	1	内部时钟, SYSCLK/2
0	1	0	定时器 0 溢出
0	1	1	ECI 引脚外部时钟
1	0	0	保留
1	0	1	内部时钟, SYSCLK/1
1	1	0	保留
1	1	1	保留

Bit 0: ECF, 使能 PCA 计数器溢出中断

0: 当 CF 位（在 CCON 寄存器里）置 1 时禁止中断

1: 当 CF 位（在 CCON 寄存器里）置 1 时使能中断

下面列出的 CCON 寄存器包含了 PCA 的控制位和 PCA 定时器及每个模块的标志位。要运行 PCA，必须软件置位 CR(CCON.6)，清零关闭 PCA。 PCA 计数器溢出，置位 CF(CCON.7)，如果 CMOD 寄存器的 ECF 位为 1，则会产生一个中断。CF 位仅由软件清零。 CCF0 和 CCF1 是模块 0 和模块 1 的中断标志位，各自的，当一个匹配或者捕获发生时，它们由硬件置位。这些标志位也仅能由软件清零。 PCA 中断系统如图 17–3 所示。

CCON: PCA 计数器控制寄存器

SFR 页 =普通

SFR 地址 = 0xD8 复位值= 00xx-xx00

7	6	5	4	3	2	1	0
CF	CR	--	--	--	--	CCF1	CCF0
R/W	**R/W**	**W**	**W**	**W**	**W**	**R/W**	**R/W**

Bit 7: CF, PCA 计数器溢出标志

0: 仅由软件清零.

1: 计数器溢出时由硬件置位。如果 CMOD 里的 ECF 为 1，则 CF 标志位能产生一个中断。CF 可以由硬件或软件置位。

Bit 6: CR, PCA 计数器运行控制位

0: 必须软件清零关闭 PCA 计数器。

1: 软件置位打开 PCA 计数器。.

Bit 5~2: 保留。当写 CCON 时，软件必须在这些位上写”0”。

Bit 1: CCF1, PCA 模块 1 中断标志

0: 必须由软件清零

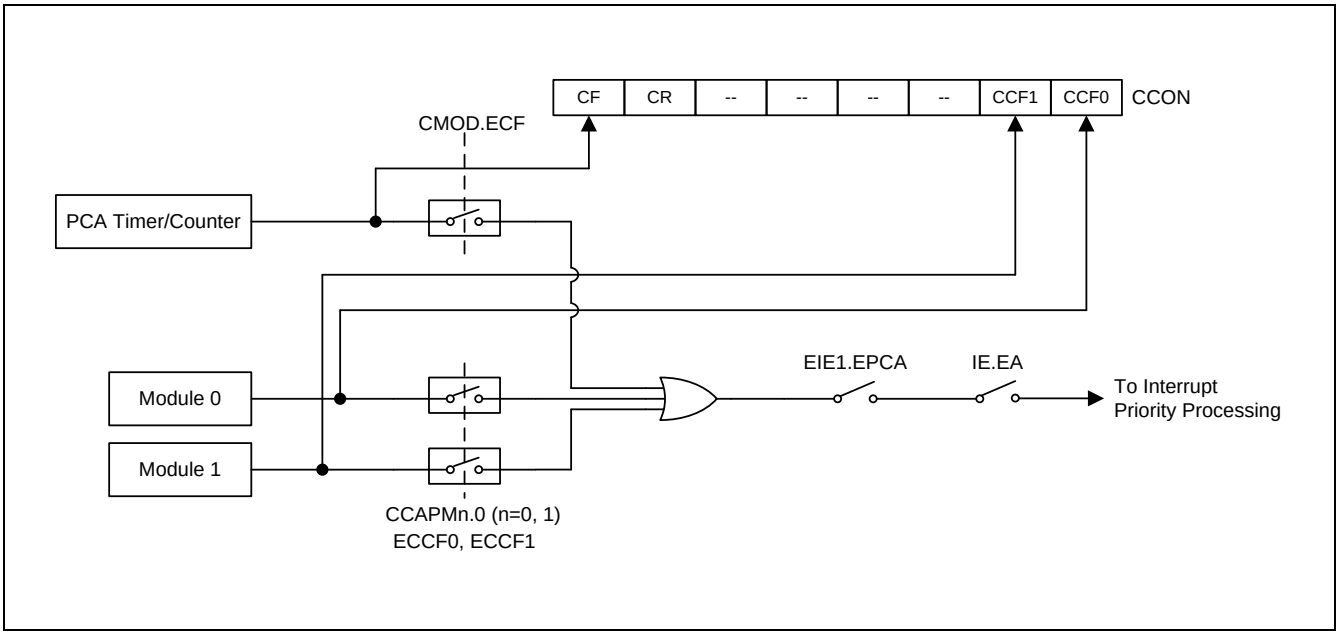
1: 当一个匹配或捕获发生时由硬件置位。

Bit 0: CCF0, PCA 模块 0 中断标志

0: 必须由软件清零

1: 当一个匹配或捕获发生时由硬件置位。

图 17–3. PCA 中断系统



17.3. 比较/捕获模块

每个比较/捕获模块 0 和 1 都有一个叫做 CCAPMn(n=0 或 1)的模式寄存器，用来选择要执行的哪个功能。注意位 ECCFn 用来在模块的中断标志位置 1 时使能产生一个中断。

CCAPMn: PCA 模块比较/铺货寄存器, n=0 or 1

SFR 页　　=普通

SFR 地址　　= 0xDA~0xDB　　复位值= x000-0000

7	6	5	4	3	2	1	0
--	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn
W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: 保留。当写 CCAPMn 时，软件必须在这个位上写”0”。

Bit 6: ECOMn, 使能比较器

0: 禁止数字比较器功能。

1: 使能数字比较器功能。

Bit 5: CAPPn, 捕获上升沿使能

0: 禁止 PCA 捕获功能在 CEXn 上升沿上检测。

1: 使能 PCA 捕获功能在 CEXn 上升沿上检测。

Bit 4: CAPNn, 捕获下降沿使能

0: 禁止 PCA 捕获功能在 CEXn 下降沿上检测。

1: 使能 PCA 捕获功能在 CEXn 下降沿上检测。

Bit 3: MATn, 匹配控制

0: 禁止数字比较器匹配事件置位 CCFn。

1: PCA 计数器和模块的比较/捕获寄存器匹配导致 CCON 上的 CCFn 位置 1。

Bit 2: TOGn, 反转控制

0: 禁止数字比较器匹配事件反转 CEXn。

1: PCA 计数器和模块的比较/捕获寄存器匹配导致 CEXn 引脚的反转。

Bit 1: PWMn, PWM 控制.

0: 禁止在 PCM 模块的 PWM 模式。

1: 使能 PWM 功能，CEXn 引脚被用来作为脉宽调制输出。

Bit 0: ECCFn, 使能 CCFn 中断。

0: 禁止 CCON 寄存器里的比较/捕获标志 CCFn 产生中断。

1: 使能 CCON 寄存器里的比较/捕获标志 CCFn 产生中断。

注意：位 CAPNn(CCAPMn.4)和 CAPPn(CCAPMn.5)决定捕获输入脚触发的边缘。如果两个位都置 1，则两个边沿都使能，不管什么变化都会捕获。

每个模块都有一对 8 位比较/捕获寄存器（CCAPnH,CCAPnL）与之对应。这些寄存器用来存储捕获事件或比较事件发生时的时间。当一个模块用于 PWM 模式时，在上述两个寄存器之外，一个额外的寄存器 PCAPWMn 用来提升输出占空比的范围。提升后的占空比的范围是从 0%到 100%，步长是 1/256。

CCAPnH: PCA 模块 n 捕获高寄存器, n=0~3

SFR 页 =普通

SFR 地址 = 0xFA~0xFD 复位值= 0000-0000

7	6	5	4	3	2	1	0
CCAPnH.7	CCAPnH.6	CCAPnH.5	CCAPnH.4	CCAPnH.3	CCAPnH.2	CCAPnH.1	CCAPnH.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

CCAPnL: PCA 模块 n 捕获低寄存器, n=0~3

SFR 页 =普通

SFR 地址 = 0xEA~0xED 复位值= 0000-0000

7	6	5	4	3	2	1	0
CCAPnL.7	CCAPnL.6	CCAPnL.5	CCAPnL.4	CCAPnL.3	CCAPnL.2	CCAPnL.1	CCAPnL.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

PCAPWMn: PWM 模式辅助寄存器，n=0, 1

SFR 页 =普通

SFR 地址 = 0xF2~0xF3 复位值= xxxx-x000

7	6	5	4	3	2	1	0
--	--	--	--	--	PnINV	ECAPnH	ECAPnL
W	**W**	**W**	**W**	**W**	**R/W**	**R/W**	**R/W**

Bit 7~3: 保留。当写 PCAPWMn 时，软件必须在这些位上写”0”。

Bit 2:反转 PWMn 上的 PWM 输出。.

0: 不反转 PWM 输出

1: 反转 PWM 输出

Bit 1: ECAPnH, 扩展第 9 位（最高位），与 CCAPnH 关联组成一个用于 PWM 模式的 9 位寄存器。

Bit 0: ECAPnL, 扩展第 9 位 (最高位) ，与 CCAPnL 关联组成一个用于 PWM 模式的 9 位寄存器。

PCAPWMn: PWM 模式辅助寄存器， n=2, 3

SFR 页 = 普通

SFR 地址 = 0xF4~0xF5 复位值 = 0xxx-x000

7	6	5	4	3	2	1	0
PWMn	--	--	--	--	PnINV	ECAPnH	ECAPnL
R/W	**W**	**W**	**W**	**W**	**R/W**	**R/W**	**R/W**

Bit 7: PWMn, PWM 控制

0: 禁止在 PCA 模块 2 和模块 3 上的 PWM 模式

1: 使能 PWM 功能，PWM2 或 PWM3 引脚用来作为一个脉宽调制输出。

Bit 6~3: 保留。当写 PCAPWMn 时，软件必须在这些位上写”0”。

Bit 2:反转 PWMn 上的 PWM 输出。.

0: 不反转 PWM 输出

1: 反转 PWM 输出

Bit 1: ECAPnH, 扩展第 9 位（最高位），与 CCAPnH 关联组成一个用于 PWM 模式的 9 位寄存器。

Bit 0: ECAPnL, 扩展第 9 位 (最高位) ，与 CCAPnL 关联组成一个用于 PWM 模式的 9 位寄存器。

17.4. PCA 工作模式

表 17–1 列出了各种 PCA 功能的 CCAPMn 寄存器的设置。

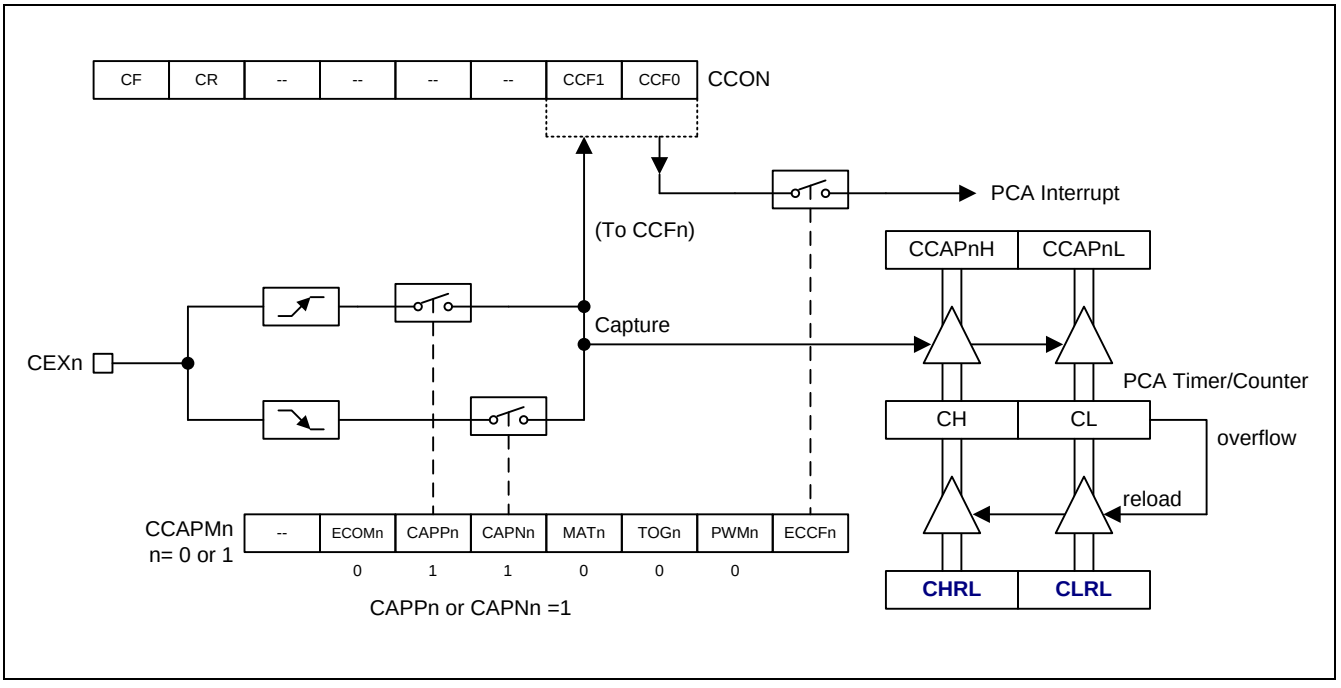
表 17–1. PCA 模块模式

ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	模块功能
0	0	0	0	0	0	0	不工作
X	1	0	0	0	0	X	16 位捕获 CEXn 上的上升沿
X	0	1	0	0	0	X	16 位捕获 CEXn 上的下降沿
X	1	1	0	0	0	X	16 位捕获 CEXn 上的变化
1	0	0	1	0	0	X	16 位软件定时器
1	0	0	1	1	0	X	16 位高速输出
1	0	0	0	0	1	0	8 位脉宽调制器 (PWM)

17.4.1. 捕获模式

为了将某一PCA模块用作捕获模式，那个模块的位CAPN和CAPP，任何一个或两个必须置1。模块的外部CEX输入脚变化时采样。当一个有效的变化发生时，PCA硬件将把PCA计数器寄存器（CH和CL）的值载入到模块的捕获寄存器（CCAPnL和CCAPnH）。如果模块对应的CCFn和ECCFn位都置1，将会产生一个中断。仅模块0和模块1支持捕获模式。

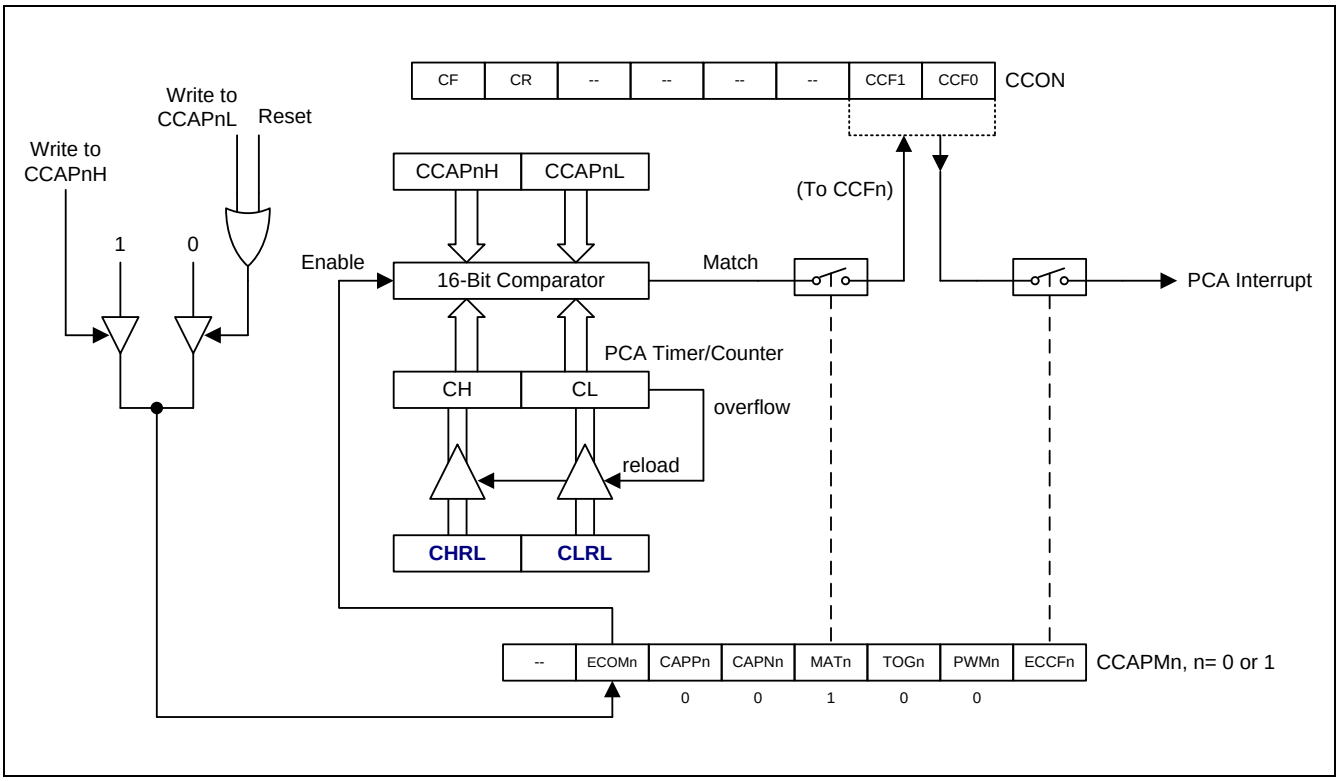
图 17–4. PCA 捕获模式



17.4.2. 16 位软件定时器模式

模块的CCAPMn寄存器的ECOM和MAT位同时置1，可以将PCA模块用作软件定时器。 PCA定时器将与模块捕获寄存器作比较，当相匹配时，如果模块的CCFn和ECCFn位都置1，则会产生一个中断。仅模块0和模块1支持软件定时器模式。

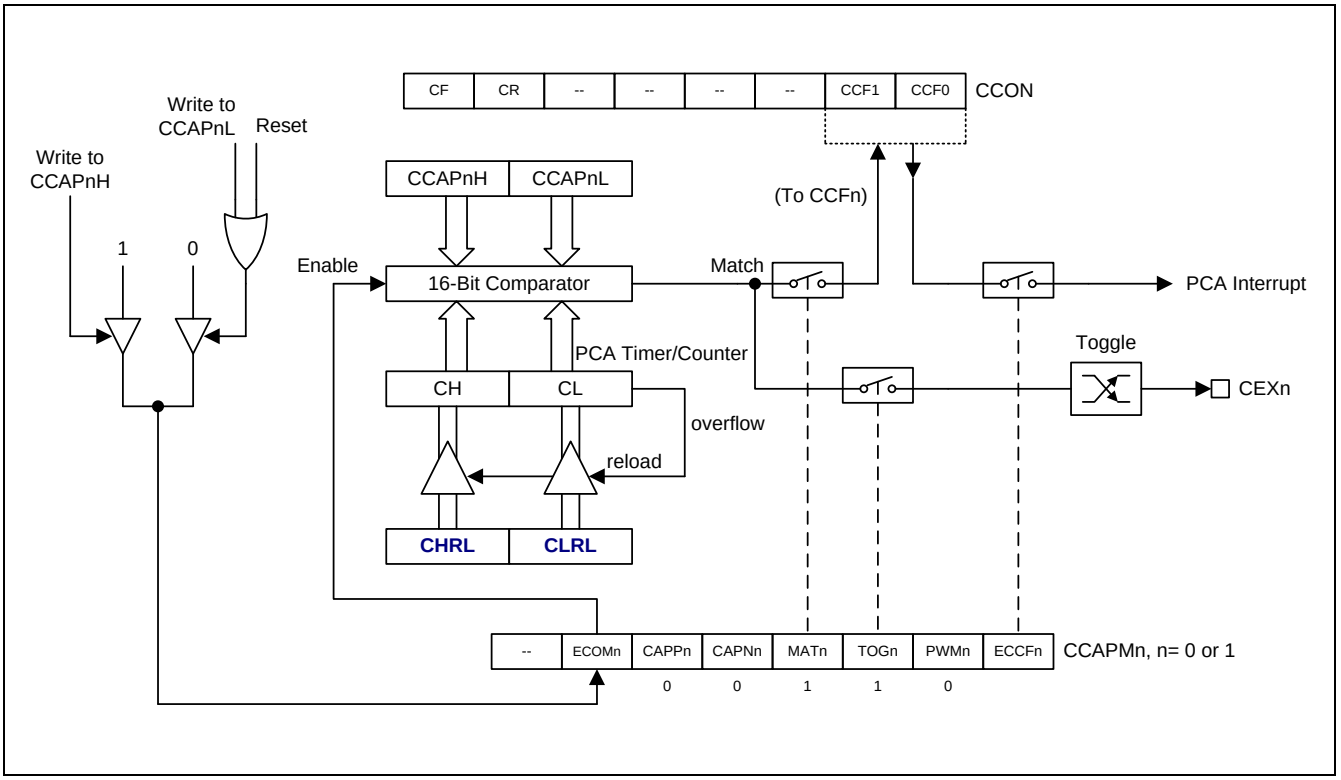
图 17–5. PCA 软件定时器模式



17.4.3. 高速输出模式

在这个模式，每次PCA计数器和模块捕获寄存器相匹配时都会触发PCA模块相对应的CEX输出。为了激活这个模块，模块的CCAPMn寄存器的TOG, MAT 和 ECOM 位必须置1.

图 17–6. PCA 高速输出模式



17.4.4. PWM 模式

所有的PCA模块都能用于PWM输出。输出频率取决于PCA时钟的时钟源。所有的模块有相同的输出频率，因为它们共用PCA时钟。

每个模块的占空比由模块的捕获寄存器CCAPnL和扩展的第9位ECAPnL所决定。当这9位｛0，[CL]｝的值小于{ECAPnl,[CCAPnL]}的9位的值时输出低电平，否则大于或等于则输出高电平。

当CL由0xFF溢出到0x00， { ECAPnL, [CCAPnL] } 将从{ ECAPnH, [CCAPnH] }的值重载。这允许无毛刺的更新PWM。模块的CCAPMn寄存器里的PWMn和ECOMn位必须置1来使能PWM模块。CLRL保存着CL重装载值在CL每一次溢出时能修改PWM的频率和分辨率。

使用9位比较，输出的占空比可以提升到实际上的0%到100%。占空比的计算公式是：

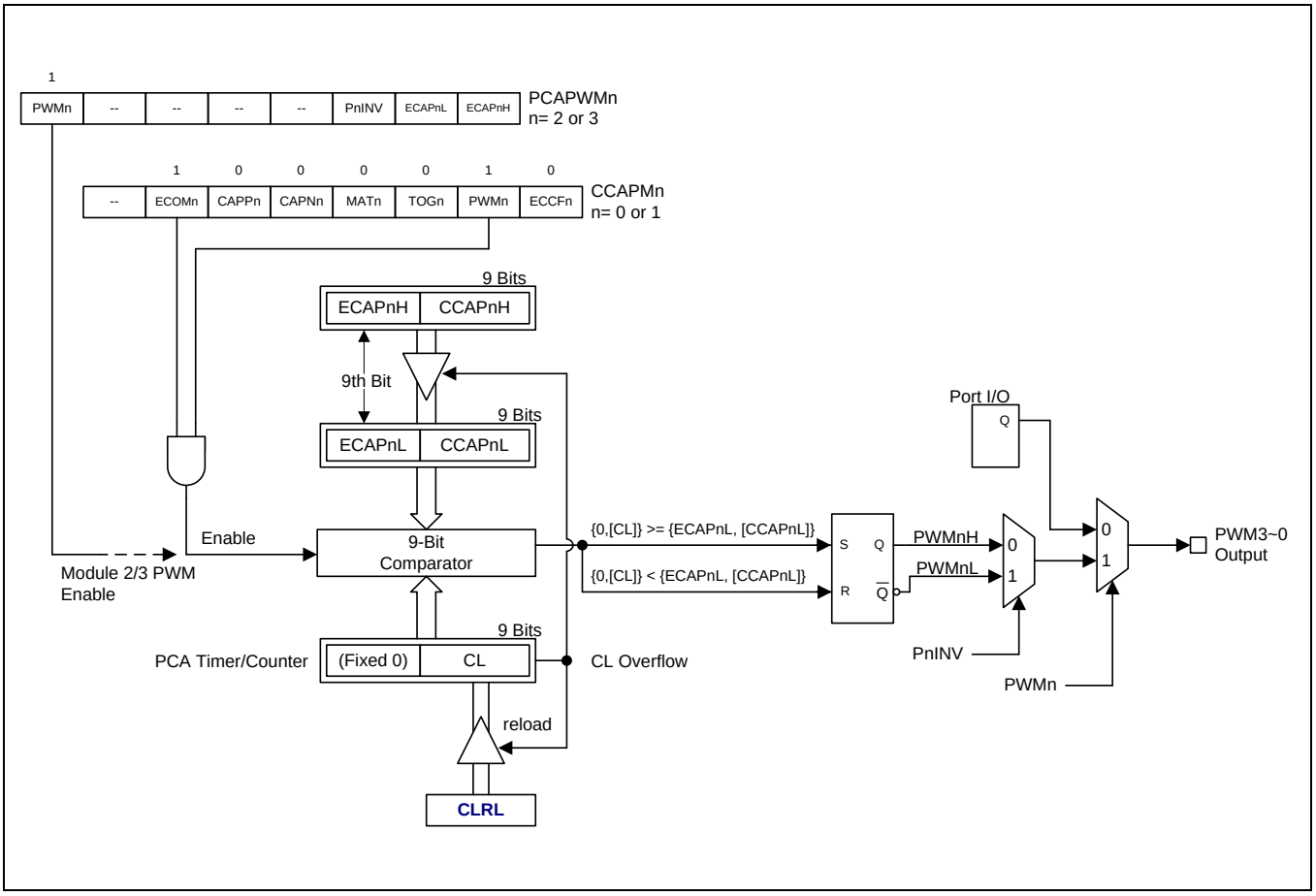
占空比= 1 – { ECAPnH, [CCAPnH] } / 256.

这里，*[CCAPnH]* 是CCAPnH寄存器的8位值，*ECAPnH* (PCAPWMn寄存器的位1)是一个位的值。因此*{ ECAPnH, [CCAPnH] }* 组成一个用于9位比较器的9位数值。

例如：

a. 如果 ECAPnH=0 & CCAPnH=0x00 (也就是0x000)，占空比是100%.
b. 如果ECAPnH=0 & CCAPnH=0x40 (也就是0x040) 占空比是75%.
c. 如果ECAPnH=0 & CCAPnH=0xC0 (也就是0x0C0)，占空比是25%.
d. 如果ECAPnH=1 & CCAPnH=0x00 (也就是0x100)，占空比是0%.

图 17–7. PCA PWM 模式



PAOE: PWM 额外输出使能寄存器

SFR 页 = 普通

SFR 地址 = 0xF1 复位值 = 0001-1001

7	6	5	4	3	2	1	0
P16OP0	**P33OP3**	**P32OP3**	**P24OP3**	**P37OP0**	**P21OP2**	**P17OP2**	**P20OP2**
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: P16OP0, P16 输出 PWM0

0: 禁止 P16 输出 PWM0。缺省值是禁止。

1: 使能 P16 输出 PWM0。

Bit 6: P33OP3, P33 输出 PWM3

0: 禁止 P33 输出 PWM3。 缺省值是禁止。

1: 使能 P33 输出 PWM3。

Bit 5: P32OP3, P32 输出 PWM3.

0: 禁止 P32 输出 PWM3。缺省值是禁止。

1: 使能 P32 输出 PWM3。

Bit 4: P24OP3, P24 输出 PWM3.

0: 禁止 P24 输出 PWM3。

1: 使能 P24 输出 PWM3。 缺省值是使能。

Bit 3: P37OP0, P37 或 P26 输出 PWM0。 CEX0 在 P37 或 P26，是由 P2PCA (AUXR1.5)可选的

0: 禁止 P37 或 P26 输出 PWM0。

1: 使能 P37 或 P26 输出 PWM0。 缺省值是使能。

Bit 2: P21OP2, P21 输出 PWM2.

0: 禁止 P21 输出 PWM2。 缺省值是禁止。

1: 使能 P21 输出 PWM2。

Bit 1: P17OP2, P17 输出 PWM2.

0: 禁止 P17 输出 PWM2。 缺省值是禁止。

1: 使能 P17 输出 PWM2。

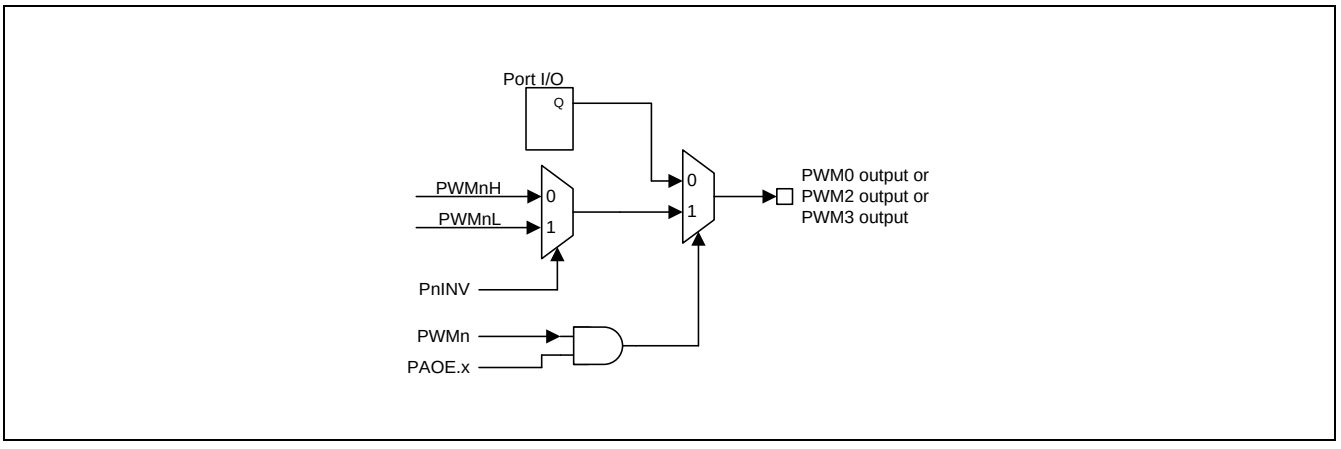
Bit 0: P20OP2, P20 输出 PWM2.

0: 禁止 P20 输出 PWM2。

1: 使能 P20 输出 PWM2。 缺省值是使能。

PAOE 提供一个 PWM 通道多个输出。PWM2 可以配置到 P2.0, P1.7 和 P2.1 三个端口同时输出。 PWM3 可以配置到 P2.4, P3.2 和 P3.3 三个端口同时输出。 PWM0 可以配置到 P3.7 和 P1.6 两个端口同时输出。**MA86E/L508** 的 PWM1 没有额外的输出通道。图 17–8 展示了 PWM 输出控制机构。图 17–8

图 17–8. PAOE 控制机构



17.5. PCA 示例代码

(1). 规定功能: 设置 PWM2/PWM3 输出占空比 25% 和 75%

汇编语言代码范例:

```
PWM2_PWM3:
        MOV     CCON,#00H                   ; 禁止 PCA 及清除 CCF0, CCF1, CF 标志
        MOV     CMOD,#02H                   ; PCA 时钟源=系统时钟 SYSCLK/ 2
;;;;;;;;; MOV   CMOD,#08H                   ; PCA 时钟源=系统时钟 SYSCLK / 1

        MOV     CH,#00H                     ; 状态初始化
        MOV     CL,#00H
        MOV     CHRL,#00H                   ; 重载初始化
        MOV     CLRL,#00H
        ;
        MOV     PCAPWM2,#PWM2               ; 使能 PCA 模块 2 (PWM 模式)
        MOV     CCAP2H,#0C0H                ; 25%
        MOV     CCAP2L,#0C0H

        MOV     PCAPWM3,#PWM3               ; 使能 PCA  模块 3 (PWM 模式)
        MOV     CCAP3H,#40H                 ; 75%
        MOV     CCAP3L,#40H
        ;
        MOV     P2M0,#00010001B             ; 使能 P2.0 和 P2.4 上拉
        SETB    CR                          ; 启动 PCA 的 PWM 输出
```

C 语言代码范例:

```
void main(void)
{
        // 设置 PCA
        CCON = 0x00;                        // 禁止 PCA 及清除 CCF0, CCF1, CF 标志
        CMOD = 0x02;                        // PCA 时钟源=系统时钟 SYSCLK / 2
//      CMOD = 0x08;                        // PCA 时钟源=系统时钟 SYSCLK / 1

        CL = 0x00; CH = 0x00;
        CHRL = 0x00; CLRL = 0x00;           // PCA 计数器范围
        //---------------------------------------------
```

```
    PCAPWM2 = PWM2;                         //使能 PCA 模块 2 (PWM 模式)
    CCAP2H = 0xC0; CCAP2L = 0xC0;           // 25%

    PCAPWM3 = PWM3;                         //使能 PCA  模块 3 (PWM 模式)
    CCAP3H = 0x40; CCAP3L = 0x40;           // 75 %
    //---------------------------------------------
    P2M0 = 0x11;
    CR = 1;                                 //  启动 PCA 的 PWM 输出

    while (1);
}
```

(2). 规定功能: 设置 PCA 为 6 位 PWM 输出 (然而 PWM 输出频率将是 4 倍的速度)

汇编语言代码范例:

```
PWM_6BIT:
    ; CEX0 = P3.7 (准双向)
    MOV     CCON,#00H                   ; 禁止 PCA 及清除 CCF0, CCF1, CF 标志
    MOV     CMOD,#02H                   ; PCA 时钟源=系统时钟 SYSCLK / 2

    MOV     CH,#0FFH                    ; 6 位模式  (65536-64)
    MOV     CL,#0C0H
    MOV     CHRL,#0FFH                  ; 6 位模式  (65536-64)
    MOV     CLRL,#0C0H
    ;
    MOV     CCAPM0,#ECOM0+PWM0          ; 模块 0 为 PWM 模式
    MOV     PCAPWM0,#00H
    MOV     CCAP0H,#0D0H                ; (占空比) 48/64 = 75 %
    MOV     CCAP0L,#0D0H                ; 256 - 48 = 204 (0D0H)
  ;

    SETB    CR                          ; 启动 CR
    RET
```

C 语言代码范例:

```
void main(void)
{
```

```
    // CEX0 = P3.7 (Quasi-bidirection)
    CCON = 0x00;                        //禁止 PCA 及清除 CCF0, CCF1, CF 标志
    CMOD = 0x02;                        // PCA 时钟源=系统时钟 SYSCLK / 2
    CCAPM0 = ECOM0+PWM0;                //  模块 0 为 PWM 模式
    PCAPWM0 = 0x00;
    //-----------------------------------------------
    CHRL = CH = (65536-64) >> 8;        // 6 位模式(65536-64)
    CLRL = CL = (65536-64);

    CCAP0L = 240;                       // (占空比) 16/64 = 25 %
    CCAP0H = 240;                       // 256 - 16 = 240
    //-----------------------------------------------
    CR    = 1;                          //启动 CR

    while (1);
}
```

18. 键盘中断(KBI)

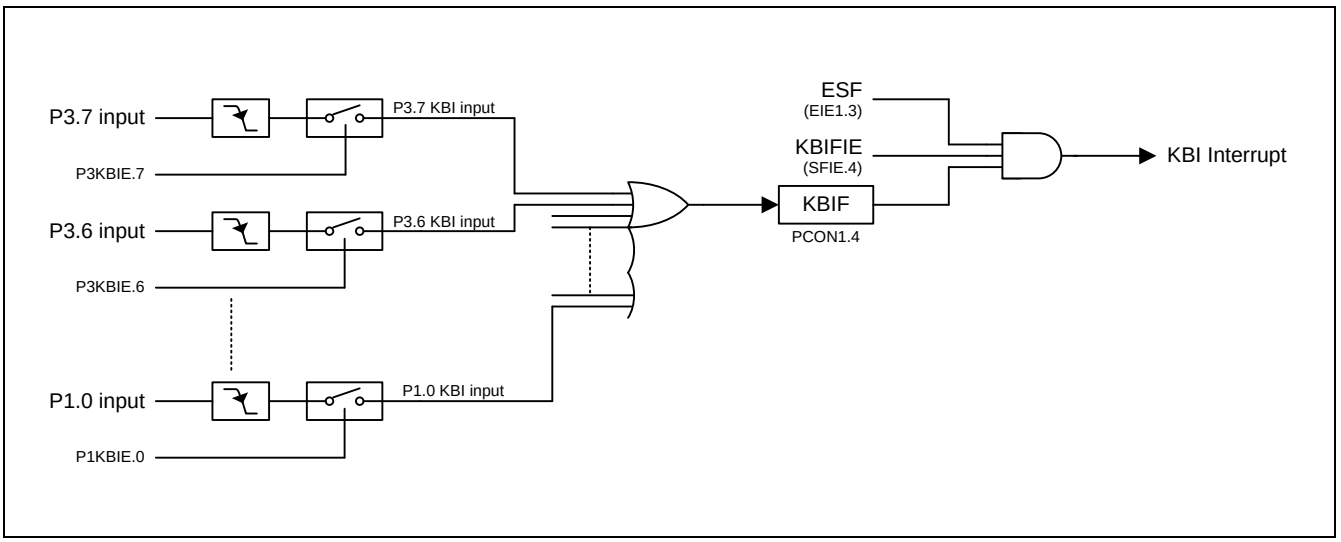
键盘中断功能是当键盘使能的脚出现下降沿时产生一个简单中断。此功能可用作键盘识别。 图 18–1 列出了键盘中断功能的结构。

此功能有两个特殊功能寄存器，P1KBIE 和 P3KBIE 的每一位控制相应的脚位在下降沿出现时使能或禁止键盘中断(KBI)。任意一个被识别的键盘中断（KBI）事件将会促使硬件设置中断标志（KBIF）如果相应的中断是使能的话则产生中断。不需要使能（KBIFIE）总中断，仅使能键盘中断（KBI）脚就可以在下降沿唤醒 CPU 的空闲模式或低电平唤醒 CPU 的掉电模式。

任一 KBI 使能端口引脚必须软件配置成输入模式。如端口 3 选择作为 KBI 功能的端口引脚,需要配置成仅输入模式，准双向模式输出高或开漏输出高。如端口 1 或端口 4 选择作为 KBI 功能的端口引脚,需要配置成开漏输出高。MCU 在 KBI 使能端口引脚上自身输出低也将触发 KBI 事件。软件设置端口 1 或端口 4 的片内上拉寄存器可提供一个弱输入高状态。端口 3 在准双向模式仅有片内上拉寄存器。

18.1. 键盘中断结构

图 18–1. 键盘中断结构



18.2. 键盘中断寄存器

P1KBIE: 端口 1 KBI 使能控制寄存器

SFR 页 = 普通

SFR 地址 = 0xD7 复位值 = 0000-0000

7	6	5	4	3	2	1	0
P17KBIE	P16KBIE	P15KBIE	P14KBIE	P13KBIE	P12KBIE	P11KBIE	P10KBIE
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~0: 端口 1 每个脚键盘中断使能位。

0: 禁止相应的脚产生键盘中断功能。

1: 使能相应的脚产生键盘中断功能

P3KBIE: 端口 3/4 KBI 使能控制寄存器

SFR 页 =普通

SFR 地址 = 0xD6 复位值= 0000-0000

7	6	5	4	3	2	1	0
P37KBIE	P36KBIE	P35KBIE	P34KBIE	**P41KBIE**	**P40KBIE**	P31KBIE	P30KBIE
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~0: 端口 P3.7 ~ P3.4, P4.1, P4.0, P3.1 和 P3.0 键盘中断使能位。

0: 禁止相应的脚产生键盘中断功能。

1: 使能相应的脚产生键盘中断功能。

PCON1: 电源控制寄存器 1

SFR 页 =普通及 P 页

SFR 地址 = 0x97 POR = 00x0-0X00

7	6	5	4	3	2	1	0
SWRF	EXRF	--	RTCF	**KBIF**	--	BOF0	WDTF
R/W	**R/W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 3: KBIF, 键盘中断标志

0: 这位必须由软件写“1”清零。软件写“0”无效。

1: 这位仅在 KBI 使能的引脚下将沿置位。 写“1”到这位清除 KBIF。掉电模式下，这位由已经使能 KBI 的引脚的低电平置位。

18.3. 键盘中断示例代码

(1). 规定功能: P1.3~P1.0 口执行键盘中断 KBI 功能

汇编语言代码范例:

```
    ORG     0003Bh
SystemFlag_ISR:                         ;
    PUSH    ACC                         ; ACC 压栈

    MOV     A,PCON1                     ;
    JNB     ACC.3,Not_KBIF_ISR          ;是否键盘中断 KBI
    To do……
    ANL     PCON1,#KBIF                 ; 清除键盘中断 KBI 标志(写 “1”)

Not_KBIF_ISR:
    POP     ACC                         ; ACC 退栈
    RETI                                ;中断返回

main:
    ORL     PUCON0,#PU10                ; 使能 P1.3~P1.0 片内上拉电阻
    ANL     P1M0,#0F0h                  ; 设置 P1.3~P1.0 为漏极开路输出
    ORL     P1,#00Fh                    ; 设置 P1.3~P1.0 为输入模式

    ORL     EIP1L,#PSFL                 ; 选择系统标志中断优先级
    ORL     EIP1H,#PSFH                 ;

    MOV     P1KBIE,#00Fh                ; 使能 P1.3~P1.0 键盘中断 KBI 功能
;   MOV     P3KBIE,#0xF3                ; 使能 P3 键盘中断 KBI 功能
;   MOV     P3KBIE,#0x0C                ; 使能 P4.1~0 键盘中断 KBI 功能

    ANL     PCON1,#KBIF                 ; 清除键盘中断 KBI 标志(写 “1”)

    ORL     SFIE,#KBIFIE                ; 使能 KBI 中断
    ORL     EIE1,#ESF                   ; 使能系统标志中断
    SETB    EA                          ; 使能全局中断

    ORL     PCON0,#PD                   ; 设置 MCU 进入掉电模式
```

C 语言代码范例:

```
void SystemFlag_ISR(void) interrupt 7
{    if(PCON1 & KBIF)
     {
          PCON1 &= KBIF;                          //清除键盘中断 KBI 标志(写 “1”)
     }
}

void main(void)
{
     PUCON0 |= PU10;                             //使能 P1.3~P1.0 片内上拉电阻
     P1M0 &= 0xF0;                               //设置 P1.3~P1.0 为漏极开路输出
     P1 |= 0x0F;                                 //设置 P1.3~P1.0 为输入模式

     EIP1L |= PSFL;                              //选择系统标志中断优先级
     EIP1H |= PSFH;

     P1KBIE = 0x0F;                              //使能 P1.3~P1.0 键盘中断 KBI 功能
//   P3KBIE = 0xF3;                              //使能 P3 键盘中断 KBI 功能
//   P3KBIE = 0x0C;                              //使能 P4.1~0 键盘中断 KBI 功能

     PCON1&= KBIF;                               //清除键盘中断 KBI 标志(写 “1”)

     SFIE |= KBIFIE;                             //使能 KBI 中断
     EIE1 |= ESF;                                //使能系统标志中断
     EA = 1;                                     //  使能全局中断

     PCON0 |= PD;                                //设置 MCU 进入掉电模式
}
```

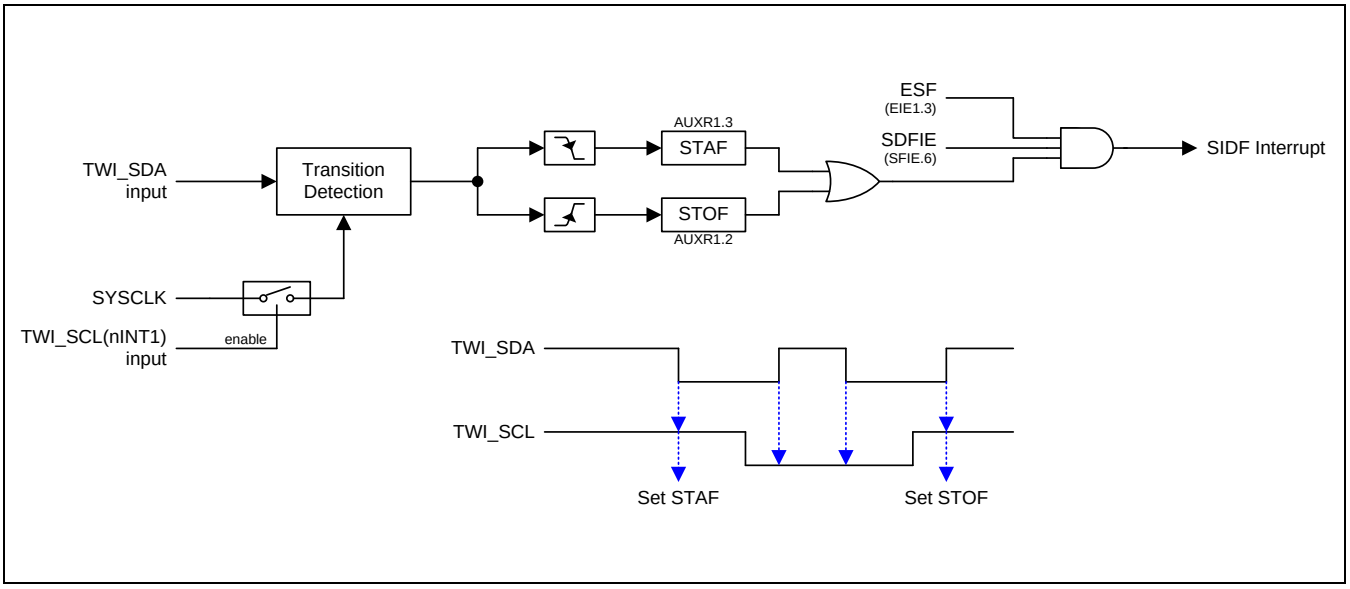
19. 串行接口侦测

串行接口侦测模块总是监控两线接口的“Start”和“Stop”状态。TW_SCL 是串行时钟信号和 TWI_SDA 是串行数据信号。如果任意一个相配的状态被侦测到，硬件置位 STAF 或 STOF 的标志。软件可以决定这两位标志或设置 SDFIE (SFIE.6)共享系统标志中断向量。并且 TWI_SCL 位于 nINT1 这样 MCU 可以通过 nINT1 中断来探测穿行数据。软件使用这些资源去完成可变的 TWI 从机设备。

19.1. 串行接口侦测结构

图 19–1 展示了 STAF 和 STOF 侦测结构，中断体系结构及事件侦测波形。

图 19–1. 串行接口侦测结构



19.2. 串行接口侦测寄存器

AUXR1: 辅助控制寄存器 1

SFR 页 = 普通

SFR 地址 = 0xA2 复位值 = 0000-0000

7	6	5	4	3	2	1	0
P3TWI	P4S0MI	P2PCA	XTOR	**STAF**	**STOF**	BPOC1	BPOC0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 3: STAF, TWI 的起始(Start)标志位检测

0: 软件写 0 清零。

1: 硬件置位表示 TWI 总线出现了起始位

Bit 2: STOF, TWI 的停止(Stop)标志侦测

0: 软件写 0 清零。

1: 硬件置位表示 TWI 总线出现了停止位

SFIE: 系统标志中断使能寄存器

SFR 页 =普通

SFR 地址 = 0x8E POR = 00x0-0x00

7	6	5	4	3	2	1	0
UTIE	SDIFIE	--	RTCFIE	KBIFIE	--	BOF0IE	WDTFIE
R/W	**W**	**W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**

Bit 6: SDIFIE, 串行接口检测标志中断使能

0: 禁止 SDIF(STAF or STOF) 中断

1: 使能 SDIF(STAF or STOF) 中断

19.3. SIDF 示例代码

在下面流程图中有两个示例代码完成 TWI 从机设备。第一个是完全中断模式，使用 STAF 和 STOF 中断来侦测 Start/Stop 事件并且使用 nINT 中断来探测串行数据输入。当 SYSCLK = 24MHz ，最大的 TWI 从机速度是 200K 位每秒（bps）。但是实际速度必须考虑系统应用中其它中断服务。

第二个示例代码是突发模式。软件仅使用 STAF 和 STOF 作为 TWI 事件侦测。然后软件决定 TWISCL 和 TWI_SDA 的端口状态。当 SYSCLK = 24MHz,在此模式通常速度是 200K 位每秒（bps）。

(1). 规定功能: TWI 从机当系统时钟 SYSCLK=24MHz 在全中断模式:

汇编语言代码范例:

```
$INCLUDE (REG_MA86E508.INC)

SLAVE_DEV_ADDR    EQU  20H              ;从机设备地址
DATA_LENGTH       EQU  32               ;缓冲区大小

;-------------------------------------------------------------------------
; TWSI 状态定义
;-------------------------------------------------------------------------
I2C_SlaveStandby  EQU  0x00
I2C_SLA_with_W    EQU  0x01
I2C_SLA_with_R    EQU  0x02
I2C_Disable       EQU  0x03
I2C_SL_W_ACK      EQU  0x04
I2C_SL_R_ACK      EQU  0x05
I2C_SL_R_NAK      EQU  0x06

;-------------------------------------------------------------------------
; TWSI 引脚定义
;-------------------------------------------------------------------------
SDA               EQU  P3.4
SCL               EQU  P3.3

;-------------------------------------------------------------------------
; 数据存储区
;-------------------------------------------------------------------------
CONTROLDATA SEGMENT DATA
        RSEG CONTROLDATA
ReceiveString:  DS      DATA_LENGTH     ; 数据缓冲区
STACK:          DS      40              ; 堆栈区大小
position:       DS      1
tempByte:       DS      1

ADDR:           DS      1
IICByte:        DS      1
Stage:          DS      1

BITDATA SEGMENT BIT
        RSEG BITDATA
firstByte:      DBIT    1               ; 接收 SLA+R/W 的标志位
completeAByte:  DBIT    1               ; 当发送/接收一个字节时设置完成标志位
Slave_RW:       DBIT    1               ; 清除 Slave_RW 设置传送

;-------------------------------------------------------------------------
; 程序代码区
;-------------------------------------------------------------------------
      CSEG    AT   0000H                ;复位起始地址= 0x0000
```

```
        JMP         ASSEMBLY_MAIN

        CSEG        AT   0013H                  ; EX0 中断服务(ISR)地址
        JMP         SCL_DETECT_ISR

        CSEG        AT   003BH                  ; 检测 STAF 或 STOF 中断服务(ISR)地址
        JMP         SystemFlag_ISR

TWSI_CS SEGMENT CODE
        RSEG        TWSI_CS
        USING       0

ASSEMBLY_MAIN:
        MOV         SP,#STACK                   ; 初始化堆栈指针(SP)
        ANL         CKCON0,#11111000B           ; 系统时钟 SYSCLK / 1
        CALL        INITIAL_TWSI                ; 初始化 TWSI

MAIN_LOOP:
        ; to do ...

        MOV         ACC,Stage
        XRL         A,#I2C_Disable
        JZ          MAIN_LOOP

        JNB         completeAByte,MAIN_LOOP     ; 等待接收/发送的事件

        ; ---------------------------------------------
        MOV         ACC,Stage
        CJNE        A,#I2C_SLA_with_W,SUBROUTINE_I2C_SLA_with_R

SUBROUTINE_I2C_SLA_with_W:
        MOV         R1,#ReceiveString           ; 接收初始化
        CLR         completeAByte               ; 清除事件标志位
        JMP         MAIN_LOOP

SUBROUTINE_I2C_SLA_with_R:
        CJNE        A,#I2C_SLA_with_R,SUBROUTINE_I2C_SL_W_ACK

        MOV         R1,#ReceiveString           ; 发送初始化
        MOV         A,@R1
        MOV         IICByte,A
        RLC         A                           ; 必须发送 MSB  高位在先到 SDA
        MOV         SDA,C
        CLR         completeAByte               ; 清除事件标志位
        JMP         MAIN_LOOP

SUBROUTINE_I2C_SL_W_ACK:
        CJNE        A,#I2C_SL_W_ACK,SUBROUTINE_I2C_SL_R_ACK
        MOV         @R1,IICByte                 ; 保存数据到接收字符串"ReceiveString"

        INC         R1                          ; 限制缓冲区索引
        CJNE        R1,#ReceiveString+DATA_LENGTH,$+3+2
        MOV         R1,#ReceiveString

        CLR         completeAByte               ; 清除事件标志位
        JMP         MAIN_LOOP

SUBROUTINE_I2C_SL_R_ACK:
        CJNE        A,#I2C_SL_R_ACK,SUBROUTINE_I2C_SL_R_NAK

        INC         R1                          ; 限制缓冲区索引
        CJNE        R1,#ReceiveString+DATA_LENGTH,$+3+2
        MOV         R1,#ReceiveString

        MOV         IICByte,@R1                 ; 从数据缓冲区准备数据
```

```
        MOV     ACC,@R1
        RLC     A
        MOV     SDA,C                   ; SDA = MSB 高位在先

        CLR     completeAByte           ; 清除事件标志位
        JMP     MAIN_LOOP

SUBROUTINE_I2C_SL_R_NAK:
        CJNE    A,#I2C_SL_R_NAK,MAIN_LOOP

        SETB    SDA                     ; 无响应事件
        CLR     completeAByte
        JMP     MAIN_LOOP


;------------------------------------------------------------------------
; 初始化 TWSI 中断(优先级)及触发模式
;------------------------------------------------------------------------
INITIAL_TWSI:
        ; 系统标志中断是最高优先级
        ORL     EIP1H,#08H
        ORL     EIP1L,#08H

        ; EX1 外部中断优先级
        ORL     IP0H,#00000100B
        ANL     IP0L,#11111011B

        ; 使能 ETWSI
        ORL     EIE1,#ESF
        ORL     SFIE,#SDIFIE
        SETB    EA

        ; P33 和 P34 是漏极开路模式在 TWSI
        MOV     P3M0,#18H
        MOV     P3M1,#18H

        ; 边缘侦测
        SETB    IT1
        ORL     AUXR0,#INT1H

        ; 声明从机设备地址
        MOV     ADDR,#SLAVE_DEV_ADDR
        MOV     Stage,#I2C_Disable
        CLR     completeAByte

        RET

;------------------------------------------------------------------------
; 初始化 TWSI 的 SDA (STAF 和 STOF)  边缘侦测
;------------------------------------------------------------------------
SystemFlag_ISR:
        PUSH    ACC
        PUSH    PSW

        MOV     ACC, AUXR1              ; 检测 STAF 或 STOF
        JB      ACC.3,STAF_ROUTINE
        JB      ACC.2,STOF_ROUTINE

EXIT_FLAG_ISR:
        POP     PSW
        POP     ACC
        RETI

STAF_ROUTINE:                           ; TWSI 启动
```

```
; 初始化 EX0 为上升沿检测和使能 EX0 中断
        ORL     AUXR0, #INT1H
        NOP
        CLR     IE1
        SETB    SDA
        SETB    EX1

        MOV     position,#0FFH              ;初始化 TWSI 的位置
        ANL     AUXR1,#~STAF                ; 清除 STAF 标志

        CLR     Slave_RW                    ; 清除接收一个字节或地址标志
        MOV     Stage,#I2C_SlaveStandby
        SETB    firstByte                   ; 地址字节标志
        JMP     EXIT_FLAG_ISR

STOF_ROUTINE:                               ; TSWI 的停止(stop)
        CLR     EX1                         ; 禁止 EX0 中断服务
        ANL     AUXR1,#~STOF                ; 清除 STOF 标志
        MOV     Stage,#I2C_Disable
        JMP     EXIT_FLAG_ISR

;--------------------------------------------------------------------------
;  通过 EX1 中断访问 SDA
;--------------------------------------------------------------------------
SCL_DETECT_ISR:
        PUSH    ACC
        PUSH    PSW

        INC     position
        JB      Slave_RW,SLAVE_READ
    ;--------------------------------------------------------------------------
SLAVE_WRITE:
        MOV     A,position
        CLR     C
        SUBB    A,#8
        JNC     SLAVE_WRITE_WAIT_FOR_ACK  ;是否应答(ACK)信号?

SLAVE_WRITE_8BITS:                          ; MSB~LSB (8 位)
        MOV     ACC,tempByte                ; 1. 反转 tempByte
        MOV     C,SDA
        RLC     A                           ; 2. 反转 SDA 到 tempByte.0
        MOV     tempByte,ACC
    ;
        MOV     A,position
        CJNE    A,#7,EXIT_SCL_DETECT_ISR
    ;
        ANL     AUXR0,#~INT1H               ; 应答(ACK)信号的下降沿
        NOP
        CLR     IE1
        JMP     EXIT_SCL_DETECT_ISR

SLAVE_WRITE_WAIT_FOR_ACK:                   ; 对 9 位 (应答(ACK)/非应答(NAK))
        JNZ     COMPLETE_WRITE_ONE_BYTE
        JNB     firstByte,SLAVE_WRITE_RESPONSE_ACK

        MOV     ACC,tempByte
        CLR     C
        RRC     A
        CJNE    A,ADDR,NOT_SLAVE_ADDR
SLAVE_WRITE_RESPONSE_ACK:
        CLR     SDA
        JMP     EXIT_SCL_DETECT_ISR
NOT_SLAVE_ADDR:
        CLR     EX1
        MOV     Stage,#I2C_Disable
```

```
        JMP     EXIT_SCL_DETECT_ISR

COMPLETE_WRITE_ONE_BYTE:                ; 第 9 位下降沿
        ORL     AUXR0,#INT1H            ; SCL 信号下降沿
        NOP
        CLR     IE1
        SETB    SDA
        SETB    completeAByte           ; 当接收一个字节时置'1'
        MOV     position,#0FFH          ; 复位位置

        JNB     firstByte,REPEAT_RECEIVE_MODE
        CLR     firstByte

        ; SLA+W 或 SLA+R ?
        CLR     Slave_RW
        MOV     ACC,tempByte
        JNB     ACC.0,SET_IN_SLAW_MODE
        SETB    Slave_RW

        ANL     AUXR0,#~INT1H           ;应答(ACK)信号下降沿
        NOP
        CLR     IE1

        MOV     Stage,#I2C_SLA_with_R
        JMP     EXIT_SCL_DETECT_ISR

SET_IN_SLAW_MODE:
        MOV     Stage,#I2C_SLA_with_W
        JMP     EXIT_SCL_DETECT_ISR

REPEAT_RECEIVE_MODE:
        MOV     IICByte,tempByte
        MOV     Stage,#I2C_SL_W_ACK
      ;-------------------------------------------------------------------------
EXIT_SCL_DETECT_ISR:
        POP     PSW
        POP     ACC
        RETI

      ;-------------------------------------------------------------------------
SLAVE_READ:
        MOV     A,position
        CLR     C
        SUBB    A,#7
        JNC     SLAVE_READ_WAIT_FOR_ACK ;是否应答(ACK)信号?
SLAVE_READ_8BITS:
        MOV     A,IICByte               ; 反转 tempByte.7 到 SDA
        RL      A
        MOV     IICByte,A
        RLC     A
        MOV     SDA,C
        JMP     EXIT_SCL_DETECT_ISR

SLAVE_READ_WAIT_FOR_ACK:
        SETB    SDA
        JNZ     COMPLETE_READ_ONE_BYTE
        JMP     EXIT_SCL_DETECT_ISR

COMPLETE_READ_ONE_BYTE:
        SETB    completeAByte           ; 当接收一个字节时置'1'
        MOV     position,#0FFH          ; 复位位置
        JNB     SDA,SET_I2C_SLAVE_READ_ACK
        MOV     Stage,#I2C_SL_R_NAK
        JMP     EXIT_SCL_DETECT_ISR

SET_I2C_SLAVE_READ_ACK:
```

```
        MOV         Stage,#I2C_SL_R_ACK
        JMP         EXIT_SCL_DETECT_ISR

;--------------------------------------------------------------------------------
        END
```

C 语言代码范例:

```
#include <REG_MA86E508.H>
#include <intrins.h>

#define SLAVE_DEV_ADDR  0x20                    // 声明从机设备地址
#define DATA_LENGTH     32                      // 缓冲区大小

//--------------------------------------------------------------------------------
// I2C 定义
//--------------------------------------------------------------------------------
#define I2C_SlaveStandby    0x00
#define I2C_SLA_with_W      0x01
#define I2C_SLA_with_R      0x02
#define I2C_Disable         0x03
#define I2C_SL_W_ACK        0x04                // SLA_W 的数据应答(ACK)
#define I2C_SL_R_ACK        0x05                // SLA_R 的数据应答(ACK)
#define I2C_SL_R_NAK        0x06                // SLA_R 的数据无应答(NAK)

//--------------------------------------------------------------------------------
// 全局变量定义
//--------------------------------------------------------------------------------
typedef     struct {
        unsigned char ADDR;
        unsigned char IICByte;
        unsigned char Stage:8;
        unsigned char completeAByte:1;
        unsigned char Slave_RW:1;
} _TWSI;

_TWSI twsi;
unsigned char tempByte;
unsigned char position;
bit firstByte;

//--------------------------------------------------------------------------------
// TWSI 引脚定义
//--------------------------------------------------------------------------------
sbit SDA = P3^4;
sbit SCL = P3^3;

//--------------------------------------------------------------------------------
// 初始化 TWSI 中断(优先级) 和触发模式
//--------------------------------------------------------------------------------
void INITIAL_TWSI ()
{
        // 系统标志是最高优先级
        EIP1H |= 0x08;
        EIP1L |= 0x08;

        // 外部中断 EX1 是一般优先级
        IP0H |= 0x08;
        IP0L &= ~0x08;

        EIE1 |= ESF;                            // 使能 ETWSI
        SFIE |= SDIFIE;
        EA = 1;
```

```
    // 在 TWSI 下 P33 和 P34 是漏极开路模式
    P3M0 = 0x18;
    P3M1 = 0x18;

    IT1 = 1;
    AUXR0 |= INT1H;

    //声明从机设备地址
    twsi.ADDR = SLAVE_DEV_ADDR;
    twsi.completeAByte = 0;
    twsi.Stage = I2C_Disable;
}

//-----------------------------------------------------------------------
// main()
//-----------------------------------------------------------------------
void main(void)
{
    unsigned char BufferIndex;
    unsigned char ReceiveString  [DATA_LENGTH];

    CKCON0 &= ~0x07;                        // 系统时钟 SYSCLK/ 1
    INITIAL_TWSI ();                        // 初始化中断和优先级

    while (1) {
        if (twsi.Stage != I2C_Disable) {
            if (twsi.completeAByte == 1) {
                switch (twsi.Stage)     {
                case I2C_SLA_with_W:
                    BufferIndex = 0;        //初始化缓冲区索引
                    twsi.completeAByte = 0;
                    twsi.Stage = I2C_SlaveStandby;
                    break;

                case I2C_SLA_with_R:
                    // prepare MSB on SDA pin
                    twsi.IICByte = ReceiveString [0];
                    SDA = twsi.IICByte & 0x80;

                    BufferIndex = 0;        //初始化缓冲区索引
                    twsi.completeAByte = 0;
                    twsi.Stage = I2C_SlaveStandby;
                    break;

                case I2C_SL_W_ACK:
                    ReceiveString [BufferIndex] = twsi.IICByte;
                    twsi.completeAByte = 0;

                    BufferIndex ++;         //限定缓冲区索引 0~31
                    BufferIndex &= 0x1F;
                    twsi.Stage = I2C_SlaveStandby;

                    break;

                case I2C_SL_R_ACK:
                    BufferIndex ++;         //限定缓冲区索引 0~31
                    BufferIndex &= 0x1F;

                    twsi.IICByte = ReceiveString [BufferIndex];
                    SDA = twsi.IICByte & 0x80;
                    twsi.completeAByte = 0;
                    twsi.Stage = I2C_SlaveStandby;
                    break;

                case I2C_SL_R_NAK:
```

```
                                SDA = 1;
                                twsi.completeAByte = 0;
                                twsi.Stage = I2C_SlaveStandby;
                                break;

                        }
                    }

                }

                // to do ...
        }

}

//---------------------------------------------------------------------------
// 初始化 TWSI 的 SDA (STAF 和 STOF)边缘侦测
//---------------------------------------------------------------------------
void SystemFlag_ISR    (void) interrupt 7
{
        unsigned char tempReg;

        tempReg = AUXR1;
        AUXR1 &= ~(STAF+STOF);                  // 清除 STAF 和 STOF 标志位

        if (tempReg & STOF)     {
                EX1 = 0;
                twsi.Stage = I2C_Disable;

        } else if (tempReg & STAF){
                AUXR0 |= INT1H;                 // SCL 上升缘侦测
                _nop_ ();
                IE1 = 0;
                SDA = 1;

                EX1 = 1;                        // 使能外部中断 EX1
                position = 0xFF;
                twsi.Slave_RW = 0;              // 清除接收一个自己或地址

                twsi.Stage = I2C_SlaveStandby;
                firstByte = 1;
        }
}

//---------------------------------------------------------------------------
// access SDA by EX1 interrupt

//---------------------------------------------------------------------------
void TWSI_EX1_ISR (void) interrupt 2
{
        position ++;

        if ((twsi.Slave_RW) == 0) {
                if (position < 8) {             // 0~7th 位
                        tempByte = tempByte    << 1;        // 6th
                        tempByte |= SDA;

                        if (position == 7) {            // 侦测下降沿
                                AUXR0 &= ~INT1H;
                                _nop_ ();
                                IE1 = 0;
                                return;
                        } else {
                                IE1 = 0;
                                return;
                        }
```

```
        } else if (position == 8){                  //  第 9 位(ACK)

            if (firstByte) {
                if ((tempByte >> 1) == twsi.ADDR) {
                    SDA = 0;
                } else {
                    EX1 = 0;
                    twsi.Stage = I2C_Disable;
                }

            } else {
                SDA = 0;
            }

        } else {
            position = 0xFF;                        //复位位置
            AUXR0 |= INT1H;                         //  下降沿侦测复位 SCL 中断
            _nop_ ();
            IE1 = 0;
            SDA = 1;

            if (firstByte) {
                firstByte = 0;

                if ((tempByte &     0x01) == 0x01) {
                    twsi.Slave_RW = 1;
                    twsi.Stage = I2C_SLA_with_R;

                    // for SCL falling edge    detection
                    AUXR0 &= ~INT1H;
                    _nop_ ();
                    IE1 = 0;

                } else {
                    twsi.Slave_RW = 0;
                    twsi.Stage = I2C_SLA_with_W;
                }
            } else {
                twsi.Stage = I2C_SL_W_ACK;
            }

            twsi.IICByte = tempByte;
            twsi.completeAByte = 1;         //  当发送一个字节时置'1'

        }
    } else {
        if (position < 7) {
            twsi.IICByte = twsi.IICByte << 1;       //  发送 6~0th 位到 SDA
            SDA = twsi.IICByte & 0x80;
        } else if (position == 8) {
            twsi.completeAByte = 1;         //当发送一个字节时置'1'
            position = 0xFF;                        //  复位位置

            if (SDA) {
                twsi.Stage = I2C_SL_R_NAK;
            } else {
                twsi.Stage = I2C_SL_R_ACK;
            }
            return;
        } else {                                    //  应答(ACK)/无应答(NAK)位
            SDA = 1;
        }

    }
}
```

(2). 规定功能: TWI 从机在系统时钟 SYSCLK=24MHz 时的突发模式:

汇编语言代码范例:

```
$INCLUDE (REG_MA86E508.INC)

SLAVE_DEV_ADDR    EQU       20H         ; 声明从机设备地址
DATA_LENGTH       EQU       32          ; 缓冲区大小

;-------------------------------------------------------------------------
; TWSI 状态定义
;-------------------------------------------------------------------------
I2C_SlaveStandby  EQU       0x00
I2C_SLA_with_W    EQU       0x01
I2C_SLA_with_R    EQU       0x02
I2C_Disable       EQU       0x03
I2C_SL_W_ACK      EQU       0x04
I2C_SL_R_ACK      EQU       0x05
I2C_SL_R_NAK      EQU       0x06

;-------------------------------------------------------------------------
; TWSI 引脚定义
;-------------------------------------------------------------------------
SDA               EQU       P3.4
SCL               EQU       P3.3

;-------------------------------------------------------------------------
; 数据区
;-------------------------------------------------------------------------
CONTROLDATA  SEGMENT DATA
             RSEG CONTROLDATA
ReceiveString:    DS        DATA_LENGTH  ; 数据缓冲区
STACK:            DS        40           ;堆栈区大小
position:         DS        1
tempByte:         DS        1

ADDR:             DS        1
IICByte:          DS        1
Stage:            DS        1

BITDATA SEGMENT BIT
             RSEG BITDATA
firstByte:        DBIT      1            ; 接收 SLA+R/W 标志位
completeAByte:    DBIT      1            ; 当发送/接收一个字节设置完成标志
Slave_RW:         DBIT      1            ; 清除 Slave_RW 设置传送
DisableTWSI:      DBIT      1            ; 清除禁止 TWSI 而激活 TWSI 发送接收
StartTWSI:        DBIT      1


;-------------------------------------------------------------------------
; 程序代码区
;-------------------------------------------------------------------------
        CSEG      AT 0000H                ;复位地址= 0x0000
        JMP       ASSEMBLY_MAIN

        CSEG      AT 0013H                ; 外部中断 EX0 中断服务(ISR)地址
        JMP       SCL_DETECT_ISR

        CSEG      AT 003BH                ; 侦测 STAF 或 STOF 中断服务(ISR)地址
        JMP       SystemFlag_ISR

TWSI_CS SEGMENT CODE
        RSEG      TWSI_CS
        USING     0

ASSEMBLY_MAIN:
```

```
        MOV     SP,#STACK                       ; 初始化堆栈指针(SP)
        ANL     CKCON0,#11111000B               ; 系统时钟 SYSCLK/ 1
        CALL    INITIAL_TWSI                    ; 初始化 TWSI

MAIN_LOOP:
        ; to do    ...

        MOV     ACC,Stage
        XRL     A,#I2C_Disable
        JZ      MAIN_LOOP

        JNB     completeAByte,MAIN_LOOP         ; 是否有事件?

        ; -------------------------------------------
        MOV     ACC,Stage
        CJNE    A,#I2C_SLA_with_W,SUBROUTINE_I2C_SLA_with_R

SUBROUTINE_I2C_SLA_with_W:
        MOV     R1,#ReceiveString               ; 初始化接收
        CLR     completeAByte                   ; 清除事件标志位
        JMP     MAIN_LOOP

SUBROUTINE_I2C_SLA_with_R:
        CJNE    A,#I2C_SLA_with_R,SUBROUTINE_I2C_SL_W_ACK

        MOV     R1,#ReceiveString               ; 初始化发送
        MOV     A,@R1
        MOV     IICByte,A
        RLC     A                               ;必须传送 MSB 高位在先到 SDA
        MOV     SDA,C
        CLR     completeAByte                   ; 清除事件标志位
        JMP     MAIN_LOOP

SUBROUTINE_I2C_SL_W_ACK:
        CJNE    A,#I2C_SL_W_ACK,SUBROUTINE_I2C_SL_R_ACK
        MOV     @R1,IICByte                     ; 保存数据到接收字符串"ReceiveString"

        INC     R1                              ; 限定缓冲区索引
        CJNE    R1,#ReceiveString+DATA_LENGTH,$+3+2
        MOV     R1,#ReceiveString

        CLR     completeAByte                   ; 清除事件标志位
        JMP     MAIN_LOOP

SUBROUTINE_I2C_SL_R_ACK:
        CJNE    A,#I2C_SL_R_ACK,SUBROUTINE_I2C_SL_R_NAK

        INC     R1                              ; 限定缓冲区索引
        CJNE    R1,#ReceiveString+DATA_LENGTH,$+3+2
        MOV     R1,#ReceiveString

        MOV     IICByte,@R1                     ; 从数据缓冲区准备数据
        MOV     ACC,@R1
        RLC     A
        MOV     SDA,C                           ; SDA = MSB 高位在先

        CLR     completeAByte                   ; 清除事件标志位
        JMP     MAIN_LOOP

SUBROUTINE_I2C_SL_R_NAK:
        CJNE    A,#I2C_SL_R_NAK,MAIN_LOOP

        SETB    SDA                             ; 无应答(NAK)事件
        CLR     completeAByte
        JMP     MAIN_LOOP
```

```
;-------------------------------------------------------------------
; 初始化 TWSI 中断 (优先级)和触发模式
;-------------------------------------------------------------------
INITIAL_TWSI:
    ; 系统标志是最高优先级
    ORL     EIP1H,#08H
    ORL     EIP1L,#08H

    ; 外部中断 EX1 一般优先级
    ORL     IP0H,#00000100B
    ANL     IP0L,#11111011B

    ; 使能 ETWSI
    ORL     EIE1,#ESF
    ORL     SFIE,#SDIFIE
    SETB EA

    ; 在 TWSI 中 P33 和 P34 是漏极开路模式
    MOV     P3M0,#18H
    MOV     P3M1,#18H

    ; 边缘侦测
    SETB    IT1
    ORL     AUXR0,#INT1H

    ; 声明从机设备地址
    MOV     ADDR,#SLAVE_DEV_ADDR
    MOV     Stage,#I2C_Disable
    CLR     completeAByte

    RET

;-------------------------------------------------------------------
; 初始化 TWSI 的 SDA(STAF 和 STOF)边缘侦测
;-------------------------------------------------------------------
SystemFlag_ISR:
    PUSH    ACC
    PUSH    PSW

    MOV     ACC, AUXR1                  ; 检测 STAF 或 STOF ?
    JB      ACC.3, STAF_ROUTINE
    JB      ACC.2, STOF_ROUTINE

EXIT_FLAG_ISR:
    POP     PSW
    POP     ACC
    RETI

STAF_ROUTINE:                           ; 启动 TWSI
    ORL     AUXR0, #INT1H  ; 初始化外部中断 EX0 为上升沿侦测和使能 EX0 中断
    NOP
    CLR     IE1
    SETB    SDA
    SETB    EX1

    CLR     DisableTWSI                 ; 清除禁止 TWSI 标志为 0 (=激活 TWSI)
    ANL     AUXR1, #~STAF               ; 清除 STAF 标志

    CLR     Slave_RW                    ; 清除接收一个字节或地址
    MOV     Stage,#I2C_SlaveStandby
    SETB    firstByte                   ; 地址字节标志
    SETB    StartTWSI
    JMP     EXIT_FLAG_ISR
```

```
STOF_ROUTINE:                           ; 停止 TSWI
        CLR     EX1                     ; 禁止外部中断 EX0 中断服务程序
        ANL     AUXR1,#~STOF            ; 清除 STOF 标志
        SETB    DisableTWSI             ; 禁止 TWSI (=TWSI 无效)
        MOV     Stage,#I2C_Disable
        JMP     EXIT_FLAG_ISR

;-------------------------------------------------------------------------
;   通过外部中断 EX1 访问 SDA
;-------------------------------------------------------------------------
SCL_DETECT_ISR:
        PUSH    ACC
        PUSH    PSW

        JNB     Slave_RW, SLAVE_WRITE
        JMP     SLAVE_READ
        ;-----------------------------------------------------------------------
SLAVE_WRITE:
        JNB     StartTWSI,$+5
        CLR     StartTWSI
        MOV     C, SDA                  ; MSB 高位在先–位 7
        MOV     A, tempByte             ; 左移 SDA 到 tempByte.0
        RLC     A
        MOV     tempByte, A

        CLR     IE1                     ; 等待 IE1
        JB      IE1, $+6
        JNB     DisableTWSI, $-3        ; 避免 STOF 事件

        JNB     StartTWSI,$+6
        LJMP    EXIT_SCL_DETECT_ISR
        MOV     C, SDA                  ; 位 6
        MOV     A, tempByte
        RLC     A
        MOV     tempByte, A
        CLR     IE1
        JB      IE1, $+6
        JNB     DisableTWSI, $-3

        JNB     StartTWSI,$+6
        LJMP    EXIT_SCL_DETECT_ISR
        MOV     C, SDA                  ; 位 5
        MOV     A, tempByte
        RLC     A
        MOV     tempByte, A
        CLR     IE1
        JB      IE1, $+6
        JNB     DisableTWSI, $-3

        JNB     StartTWSI,$+6
        LJMP    EXIT_SCL_DETECT_ISR
        MOV     C, SDA                  ; 位 4
        MOV     A, tempByte
        RLC     A
        MOV     tempByte, A
        CLR     IE1
        JB      IE1, $+6
        JNB     DisableTWSI, $-3

        JNB     StartTWSI,$+6
        LJMP    EXIT_SCL_DETECT_ISR
        MOV     C, SDA                  ; 位 3
        MOV     A, tempByte
        RLC     A
```

```
        MOV     tempByte, A
        CLR     IE1
        JB      IE1, $+6
        JNB     DisableTWSI, $-3

        JNB     StartTWSI,$+6
        LJMP    EXIT_SCL_DETECT_ISR
        MOV     C, SDA                          ; 位 2
        MOV     A, tempByte
        RLC     A
        MOV     tempByte, A
        CLR     IE1
        JB      IE1, $+6
        JNB     DisableTWSI, $-3

        JNB     StartTWSI,$+6
        LJMP    EXIT_SCL_DETECT_ISR
        MOV     C, SDA                          ; 位 1
        MOV     A, tempByte
        RLC     A
        MOV     tempByte, A
        CLR     IE1
        JB      IE1, $+6
        JNB     DisableTWSI, $-3

        JNB     StartTWSI,$+6
        LJMP    EXIT_SCL_DETECT_ISR
        MOV     C, SDA                          ; 位 0
        MOV     A, tempByte
        RLC     A
        MOV     tempByte, A
        CLR     IE1

        JB      DisableTWSI, EXIT_WITHOUT_COMPLETE_FLAG

        JNB     StartTWSI,$+6
        LJMP    EXIT_SCL_DETECT_ISR

        ANL     AUXR0, #~INT1H                  ; 设置外部中断 EX1 为下降沿侦测
        NOP
        CLR     IE1
        JB      IE1, $+6
        JNB     DisableTWSI, $-3

        JNB     firstByte,SLAVE_WRITE_RESPONSE_ACK

        MOV     ACC,tempByte
        CLR     C
        RRC     A
        CJNE    A,ADDR,NOT_SLAVE_ADDR
SLAVE_WRITE_RESPONSE_ACK:
        CLR     SDA
        JMP     COMPLETE_WRITE_ONE_BYTE

NOT_SLAVE_ADDR:
        CLR     EX1
        MOV     Stage,#I2C_Disable
        JMP     EXIT_SCL_DETECT_ISR

COMPLETE_WRITE_ONE_BYTE:                        ; 第 9 位下降沿
        CLR     IE1
        JB      IE1, $+6                        ; 等待第 9 位
        JNB     DisableTWSI, $-3

        SETB     SDA                            ; 设置 SDA 为输入
```

```
        ORL     AUXR0, #INT1H                   ; 设置外部中断 EX1 为边缘侦测
        NOP
        CLR     IE1

        SETB    completeAByte                   ; 当接收一个字节时置'1'
        JNB     firstByte,REPEAT_RECEIVE_MODE
        CLR     firstByte

        ; SLA+W   or SLA+R ?
        CLR     Slave_RW
        MOV     ACC,tempByte
        JNB     ACC.0,SET_IN_SLAW_MODE
        SETB    Slave_RW

        ANL     AUXR0,#~INT1H                   ; 应答信号(ACK)的下降沿
        NOP
        CLR     IE1

        MOV     Stage,#I2C_SLA_with_R
        JMP     EXIT_SCL_DETECT_ISR

SET_IN_SLAW_MODE:
        MOV     Stage,#I2C_SLA_with_W
        JMP     EXIT_SCL_DETECT_ISR

REPEAT_RECEIVE_MODE:
        MOV     IICByte,tempByte
        MOV     Stage,#I2C_SL_W_ACK
        ;-------------------------------------------------------------------------
EXIT_SCL_DETECT_ISR:
        POP     PSW
        POP     ACC
        RETI

EXIT_WITHOUT_COMPLETE_FLAG:
        CLR     completeAByte
        JMP     EXIT_SCL_DETECT_ISR
        ;-------------------------------------------------------------------------
SLAVE_READ:
        ; 必须发送(tempByte.7)在主程序中
        ; 并且设置外部中断 EX1(SCL)  为下降沿侦测
        JNB     StartTWSI,$+5
        CLR     StartTWSI

        MOV     A,IICByte                       ; 发送 IICByte.6
        RL      A
        MOV     IICByte,A
        RLC     A
        MOV     SDA,C

        JNB     StartTWSI,$+6
        LJMP    EXIT_SCL_DETECT_ISR
        CLR     IE1                             ; 等待 IE1
        JB      IE1, $+6
        JNB     DisableTWSI, $-3                ; 避免 STOF 事件

        JNB     StartTWSI,$+6
        LJMP    EXIT_SCL_DETECT_ISR
        MOV     A,IICByte                       ; 发送 IICByte.5
        RL      A
        MOV     IICByte,A
        RLC     A
        MOV     SDA,C
        CLR     IE1
        JB      IE1, $+6
        JNB     DisableTWSI, $-3
```

```
JNB     StartTWSI,$+6
LJMP    EXIT_SCL_DETECT_ISR
MOV     A,IICByte                       ; 发送 IICByte.4
RL      A
MOV     IICByte,A
RLC     A
MOV     SDA,C
CLR     IE1
JB      IE1, $+6
JNB     DisableTWSI, $-3

JNB     StartTWSI,$+6
LJMP    EXIT_SCL_DETECT_ISR
MOV     A,IICByte                       ; 发送 IICByte.3
RL      A
MOV     IICByte,A
RLC     A
MOV     SDA,C
CLR     IE1
JB      IE1, $+6
JNB     DisableTWSI, $-3

JNB     StartTWSI,$+6
LJMP    EXIT_SCL_DETECT_ISR
MOV     A,IICByte                       ; 发送 IICByte.2
RL      A
MOV     IICByte,A
RLC     A
MOV     SDA,C
CLR     IE1
JB      IE1, $+6
JNB     DisableTWSI, $-3

JNB     StartTWSI,$+6
LJMP    EXIT_SCL_DETECT_ISR
MOV     A,IICByte                       ; 发送 IICByte.1
RL      A
MOV     IICByte,A
RLC     A
MOV     SDA,C
CLR     IE1
JB      IE1, $+6
JNB     DisableTWSI, $-3

JNB     StartTWSI,$+6
LJMP    EXIT_SCL_DETECT_ISR
MOV     A,IICByte                       ; 发送 IICByte.0
RL      A
MOV     IICByte,A
RLC     A
MOV     SDA,C
CLR     IE1
JB      IE1, $+6
JNB     DisableTWSI, $-3

JNB     StartTWSI,$+6
LJMP EXIT_SCL_DETECT_ISR

SETB SDA                                ; 第 9 位-应答(ACK)/无应答(NAK)
CLR     IE1
JB      IE1, $+6
JNB     DisableTWSI, $-3

JNB     SDA,SET_I2C_SLAVE_READ_ACK
MOV     Stage,#I2C_SL_R_NAK
```

```
        JMP         COMPLETE_READ_ONE_BYTE

SET_I2C_SLAVE_READ_ACK:
        MOV         Stage,#I2C_SL_R_ACK

COMPLETE_READ_ONE_BYTE:
        CLR         IE1
        SETB        completeAByte                   ; 当发送一个字节只'1'

        JMP         EXIT_SCL_DETECT_ISR

;-------------------------------------------------------------------------------
            END
```

C 语言代码范例:

```
#include <REG_MA86E508.H>
#include <intrins.h>

#define SLAVE_DEV_ADDR  0x20                    // 声明从机设备地址
#define DATA_LENGTH     32                      // 缓冲区大小定义

//-------------------------------------------------------------------------------
// I2C 定义
//-------------------------------------------------------------------------------
#define I2C_SlaveStandby    0x00
#define I2C_SLA_with_W      0x01
#define I2C_SLA_with_R      0x02
#define I2C_Disable         0x03
#define I2C_SL_W_ACK        0x04                // SLA_W 为数据应答(ACK)
#define I2C_SL_R_ACK        0x05                // SLA_R 为数据应答(ACK)
#define I2C_SL_R_NAK        0x06                // SLA_R 为数据无应答(NAK)

//-------------------------------------------------------------------------------
// 全局变量定义
//-------------------------------------------------------------------------------
typedef     struct {
        unsigned char ADDR;
        unsigned char IICByte;
        unsigned char Stage:8;
        unsigned char completeAByte:1;
        unsigned char Slave_RW:1;
} _TWSI;

_TWSI twsi;
unsigned char tempByte;
bit firstByte,DisableTWSI,StartTWSI;

//-------------------------------------------------------------------------------
// TWSI 引脚定义
//-------------------------------------------------------------------------------
sbit SDA = P3^4;
sbit SCL = P3^3;

//-------------------------------------------------------------------------------
// 初始化 TWSI 中断 (优先级)和触发模式
//-------------------------------------------------------------------------------
void INITIAL_TWSI ()
{
        // 系统标志是最高优先级
        EIP1H |= 0x08;
        EIP1L |= 0x08;

        // 外部中断 EX1 是一般优先级
        IP0H |= 0x08;
        IP0L &= ~0x08;
```

```
	EIE1 |= ESF;                                    // 使能 ETWSI
	SFIE |= SDIFIE;
	EA = 1;

	//在 TWSI 中 P33 和 P34 是漏极开路模式
	P3M0 = 0x18;
	P3M1 = 0x18;

	IT1 = 1;
	AUXR0 |= INT1H;

	// 声明从机设备地址
	twsi.ADDR = SLAVE_DEV_ADDR;
	twsi.completeAByte = 0;
	twsi.Stage = I2C_Disable;
}

//-----------------------------------------------------------------------------
// main()
//-----------------------------------------------------------------------------
void main(void)
{
	unsigned char BufferIndex;
	unsigned char ReceiveString  [DATA_LENGTH];

	CKCON0 &= ~0x07;                                // 系统时钟 SYSCLK / 1
	INITIAL_TWSI ();                                // 初始化中断和优先级

	while (1) {
		if (twsi.Stage != I2C_Disable) {
			if (twsi.completeAByte == 1) {
				switch (twsi.Stage)	{
				case I2C_SLA_with_W:
					BufferIndex = 0;		// 初始化缓冲区索引
					twsi.completeAByte = 0;
					twsi.Stage = I2C_SlaveStandby;
					break;

				case I2C_SLA_with_R:
					// prepare MSB on SDA pin
					twsi.IICByte = ReceiveString [0];
					SDA = twsi.IICByte & 0x80;

					BufferIndex = 0;		//初始化缓冲区索引
					twsi.completeAByte = 0;
					twsi.Stage = I2C_SlaveStandby;
					break;

				case I2C_SL_W_ACK:
					ReceiveString [BufferIndex] = twsi.IICByte;
					twsi.completeAByte = 0;

					BufferIndex ++;		//限定缓冲区索引 0~31
					BufferIndex &= 0x1F;
					twsi.Stage = I2C_SlaveStandby;

					break;

				case I2C_SL_R_ACK:
					BufferIndex ++;		//限定缓冲区索引 0~31
					BufferIndex &= 0x1F;

					twsi.IICByte = ReceiveString [BufferIndex];
					SDA = twsi.IICByte & 0x80;
					twsi.completeAByte = 0;
					twsi.Stage = I2C_SlaveStandby;
```

```
                                    break;

                                case I2C_SL_R_NAK:
                                    SDA = 1;
                                    twsi.completeAByte = 0;
                                    twsi.Stage = I2C_SlaveStandby;
                                    break;

                                }
                            }

                        }

                        // to do ...
                    }

}

//--------------------------------------------------------------------------
//  初始化 TWSI 的 SDA (STAF 和 STOF)边缘侦测
//--------------------------------------------------------------------------
void SystemFlag_ISR (void) interrupt 7
{
    unsigned char tempReg;

    tempReg = AUXR1;
    AUXR1 &= ~(STAF+STOF);                  //  清除 STAF 和 STOF 标志
    if (tempReg & STOF) {
        EX1 = 0;
        DisableTWSI = 1;
        twsi.Stage = I2C_Disable;
    } else if (tempReg & STAF){
        AUXR0 |= INT1H;
        _nop_ ();
        IE1 = 0;
        SDA = 1;
        EX1 = 1;
        DisableTWSI = 0;                    //  避免误解

        twsi.Slave_RW = 0;                  //  接收一个字节或地址而清除

        twsi.Stage = I2C_SlaveStandby;
        firstByte = 1;
        StartTWSI = 1;
    }
}

// ---------------------------------------------------------------------------
//    通过外部中断 EX1  访问 SDA
// ---------------------------------------------------------------------------
void TWSI_EX1_ISR(void) interrupt 2
{
    if (twsi.Slave_RW == 0) {
        if (StartTWSI) {
            StartTWSI = 0;
        }
        tempByte = tempByte << 1;           //  位 7
        tempByte |= SDA;
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);
                                            //位 6
        if (StartTWSI) return;
        tempByte = tempByte << 1;
        tempByte |= SDA;
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);
```

```
if (StartTWSI) return;
tempByte = tempByte << 1;                    //位 5
tempByte |= SDA;
IE1 = 0;
while ((IE1 | DisableTWSI) == 0);

if (StartTWSI) return;
tempByte = tempByte << 1;                    //位 4
tempByte |= SDA;
IE1 = 0;
while ((IE1 | DisableTWSI) == 0);
if (StartTWSI) return;
tempByte = tempByte << 1;                    //位 3
tempByte |= SDA;
IE1 = 0;
while ((IE1 | DisableTWSI) == 0);

if (StartTWSI) return;
tempByte = tempByte << 1;                    //位 2
tempByte |= SDA;
IE1 = 0;
while ((IE1 | DisableTWSI) == 0);

if (StartTWSI) return;
tempByte = tempByte << 1;                    //位 1
tempByte |= SDA;
IE1 = 0;
while ((IE1 | DisableTWSI) == 0);

if (StartTWSI) return;
tempByte = tempByte << 1;                    //位 0
tempByte |= SDA;

AUXR0 &= ~INT1H;                             // SCL 的边缘侦测
_nop_ ();
IE1 = 0;
while ((IE1 | DisableTWSI) == 0);            //  第 0 个下降沿

if (StartTWSI) return;
if (DisableTWSI) return;

if (firstByte) {
    if ((tempByte >> 1) == SLAVE_DEV_ADDR) {
        SDA = 0;
    } else {
        EX1 = 0;
        twsi.Stage = I2C_Disable;
    }
} else {
    SDA = 0;
}

IE1 = 0;
while ((IE1 | DisableTWSI) == 0);
SDA = 1;

AUXR0 |= INT1H;                              // SCL 的上升缘侦测
_nop_ ();
IE1 = 0;

if (firstByte) {
    firstByte = 0;

    twsi.Slave_RW = (tempByte & 0x01);
    if (tempByte & 0x01) {
```

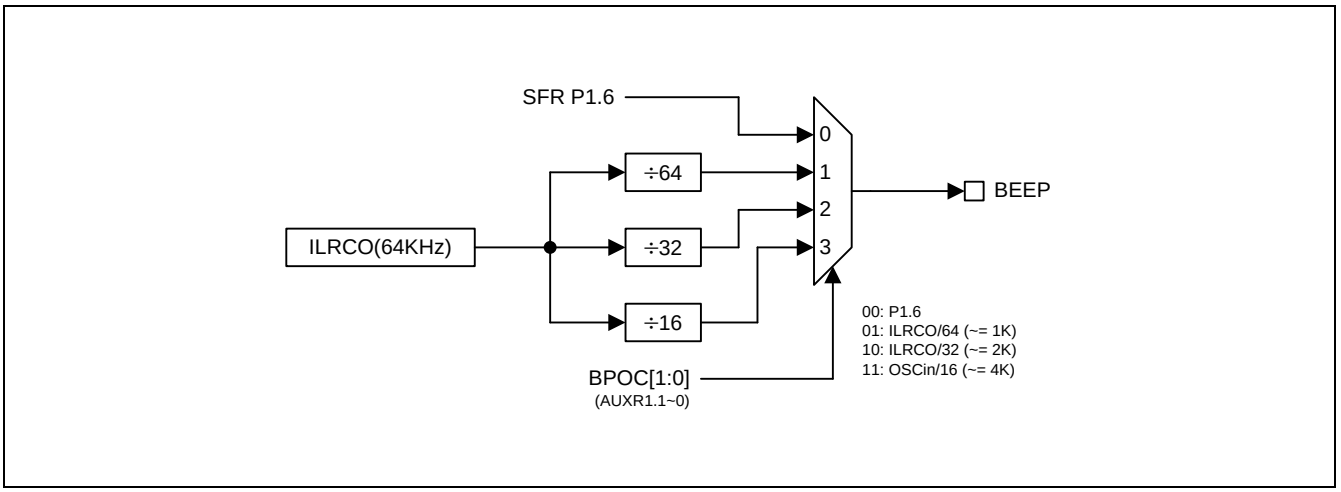
```
                twsi.Slave_RW = 1;
                twsi.Stage = I2C_SLA_with_R;
                // for SCL falling edge detection
                AUXR0 &= ~INT1H;
                _nop_ ();
                IE1 = 0;
            } else {
                twsi.Slave_RW = 0;
                twsi.Stage = I2C_SLA_with_W;
            }
        } else {
            twsi.Stage = I2C_SL_W_ACK;
        }
        twsi.IICByte = tempByte;
        if (DisableTWSI) return;
        twsi.completeAByte = 1;                 // 当发送一个字节置'1'
        P35 = 1;
    } else {
        if (StartTWSI) {
            StartTWSI = 0;
        }
        twsi.IICByte = twsi.IICByte << 1;
        SDA = twsi.IICByte & 0x80;              //位 6
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        if (StartTWSI) return;
        twsi.IICByte = twsi.IICByte << 1;
        SDA = twsi.IICByte & 0x80;              //位 5
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        if (StartTWSI) return;
        twsi.IICByte = twsi.IICByte << 1;
        SDA = twsi.IICByte & 0x80;              //位 4
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        if (StartTWSI) return;
        twsi.IICByte = twsi.IICByte << 1;
        SDA = twsi.IICByte & 0x80;              //位 3
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        if (StartTWSI) return;
        twsi.IICByte = twsi.IICByte << 1;
        SDA = twsi.IICByte & 0x80;              //位 2
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        if (StartTWSI) return;
        twsi.IICByte = twsi.IICByte << 1;
        SDA = twsi.IICByte & 0x80;              //位 1
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        if (StartTWSI) return;
        twsi.IICByte = twsi.IICByte << 1;
        SDA = twsi.IICByte & 0x80;              //位 0
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        SDA = 1;                                // 应答(ACK)
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);
        IE1 = 0;
```

```
            if (DisableTWSI) return;
            if (SDA) {
                twsi.Stage = I2C_SL_R_NAK;
            } else {
                twsi.Stage = I2C_SL_R_ACK;
            }
            twsi.completeAByte = 1;
        }
}
```

20. 蜂鸣器

蜂鸣器功能是在 BEEP 引脚上输出一个发声信号。信号范围大约 1, 2 或 4 kHz 由 ILRCO 分频产生。 图 20–1 展示了蜂鸣器信号发生器电路。 但是 ILRCO 不是很准确的时钟源。详细的 ILRCO 频率偏差范围请参考 “27.5 ILRCO”。

图 20–1. 蜂鸣器信号发生器



20.1. 蜂鸣器寄存器

AUXR1: 辅助控制寄存器 1

SFR 页 = 普通

SFR 地址 = 0xA2 复位值 = 0000-0000

7	6	5	4	3	2	1	0
P3TWI	P4S0MI	P2PCA	XTOR	STAF	STOF	**BPOC1**	**BPOC0**
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 1~0: BPOC1~0, 蜂鸣器输出控制位

BPOC[1:0]	P1.6 功能	I/O 模式
00	P1.6	P1M0.6 决定
01	ILRCO/64	P1M0.6 决定
10	ILRCO/32	P1M0.6 决定
11	ILRCO/16	P1M0.6 决定

P1.6 的蜂鸣器功能,推荐设置 P1M0.6 为“1”即选择 P1.6 作为推挽输出模式。

20.2. 蜂鸣器示例代码

(1). 规定功能: 设置蜂鸣器输出 1KHz

汇编语言代码范例:

```
ORL     P1M0,#40H                   ; 设置 P1.6 为推挽输出模式

ANL     AUXR1,#~(BPOC1|BPOC0)       ; 设置 P1.6 为通用输入输出(GPIO)功能
ORL     AUXR1,#BPOC0                ; BEEP = ILRCO/64 ~= 1KHz  蜂鸣器频率约为 1KHz
```

C 语言代码范例:

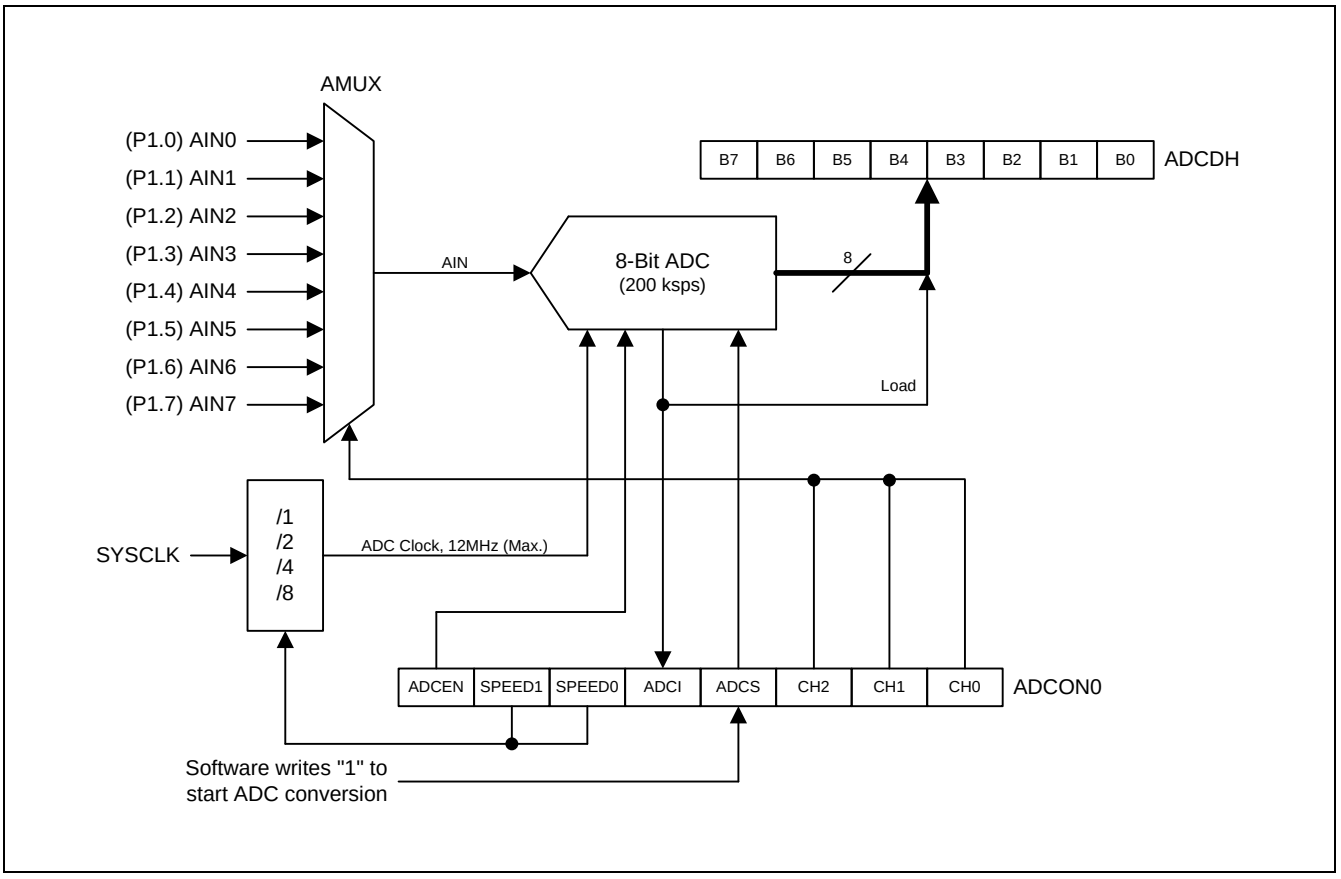
```
P1M0 = P1M0 | 0x40;                 //设置 P1.6 为推挽输出模式

AUXR1 &= ~(BPOC1 | BPOC0);          //设置 P1.6 为通用输入输出(GPIO)功能
AUXR1 |= BPOC0;                     // BEEP = ILRCO/64 ~= 1KHz   蜂鸣器频率约为 1KHz
```

21. 8-位 ADC

MA86E/L508 的 ADC 子系统是由一个模拟多路复用器（AMUX）和一个 200Ksps,8 位逐次逼近型 ADC 组成。如图 21–1 所示，AMUX 可以经由特殊功能寄存器去配置。ADC 操作在单端模式，可以配置成去测量 P1 口的任何引脚。ADC 子系统仅当 ADC 控制寄存器的 ADEN 位设置成逻辑 1 时被使能，当被设置成逻辑 0 时，ADC 子系统进入低功耗关闭状态。

21.1. ADC 结构

图 21–1. ADC 方框图



21.2. ADC 工作

ADC 的最大转换速度是 200 ksps. ADC 转换时钟由系统时钟分频而来，由寄存器 ADCON0 的 SPEED1~0 位决定。ADC 转换时钟不得超过 **12MHz**.

转换完成后(ADCI 是高),在 ADC 结果寄存器（ADCDH）里可以找到转换的结果。对于单端转换，结果是：

$$\text{ADC Result} = \frac{V_{IN} \times 256}{\text{VDD Voltage}}$$

21.2.1. ADC 输入通道

模拟多路复用器(AMUX)选择用于 ADC 的输入，允许 P1 口的任一引脚作为单端模式加以测量。如图 21–1 所述，通过寄存器 ADCON0 的 CHS2-0 位配置和选择 ADC 输入通道。 选择的脚相对于 GND 进行测量。

21.2.2. 启动一个转换

在使用 ADC 之前，用户应当:

1) 设置 ADCEN 这个位来打开 ADC 硬件。
2) 通过位 SPEED1 和 SPEED0 来配置 ADC 输入时钟。
3) 通过位 CHS2,CHS1 和 CHS0 来选择模拟输入通道。
4) 通过 P1,P1M0 和 P1AIO 配置已选择的输入脚（和 P1 共享）为仅输入模式。

现在，用户可以置 ADCS 位来开始 AD 转换。转换时间由 SPEED1 和 SPEED0 控制。一旦转换完成，硬件将自动清 ADCS 位，置位中断标志为 ADCI，同时将转换结果加载到 ADCH。

如上所述，中断标志位，当硬件置位后，表示了一个完整的转换。因此可以通过两个方式来检测转换是否完成:（1）总是用软件轮询中断标志位 ADCI。（2）通过置位 EADC(在 EIE1 寄存器了)和 EA(在 IE 寄存器)使能 ADC 中断，当转换完成后，CPU 会跳到中断服务程序。无论是(1)还是(2),ADCI 标志位都应当在下一个转换之前由软件清零。

21.2.3. ADC 转换时间

用户可以依照模拟输入信号的频率选择适当的转换速度。ADC 的最大输入时钟是 **12MHz,** 并且用 60 个 ADC 时钟作为一个固定的转换时间。用户可以配置 ADCON0 上的 SPEED1~0 来选择转换速率。例如，如果 Fosc=12MHz，并且选择了 ADCKS=SYSCLK/1，那么为了转换的准确度，模拟输入的频率应当不要超过 200KHz。 (转换率 = 12MHz/1/60 = 200KHz.)

21.2.4. I/O 引脚用于 ADC 功能

用于 A/D 转换的模拟输入引脚也有 I/O 口的数字输入和输出功能。为了得到更好的模拟性能，用于 ADC 的引脚应当禁止它的数字输出功能。它应当设置成仅输入模式，参考章节“14 输入输出口配置”。

当 ADCI7~0 作为一个模拟信号输入，并且这个脚不需要数字输入，软件可以在 P1AIO 里设置相对应的引脚作为仅模拟输入，来减少在数字输入缓存的功耗。

21.2.5. 空闲和掉电模式

在掉电模式，ADC 不能工作。如果 A/D 打开了，它将浪费一些功耗。因此，在进入空闲模式和掉电模式之前关掉 ADC 硬件（ADEN=0）将会节省一些功耗。

21.3. ADC 寄存器

ADCON0: ADC 控制寄存器 0

SFR 页 = 普通

SFR 地址 = 0xC4 复位值 = 0000-0000

7	6	5	4	3	2	1	0
ADCEN	SPEED1	SPEED0	ADCI	ADCS	CHS2	CHS1	CHS0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: ADCEN, ADC 使能

0: 清零关闭 ADC 模块

1: 置位打开 ADC 模块，置位 ADCS 之前必须至少有 5US 的 ADC 使能时间。

Bit 6~5: SPEED1 和 SPEED0, ADC 转换速度控制

SPEED[1:0]	ADC 时钟选择
0 0	SYSCLK/1
0 1	SYSCLK/2
1 0	SYSCLK/4
1 1	SYSCLK/8

推荐 ADC 时钟不要超过 12MHz。

Bit 4: ADCI, ADC 中断标志位.

0: 这位必须软件清零。

1: A/D 转换完成后置位这个位，如果 ADC 中断使能，则会进入 ADC 中断。

Bit 3: ADCS. ADC 转换启始

0: ADCS 不能被软件清零

1:软件置这个位来开始一个 A/D 转换。转换完成后，ADC 硬件将清除这个位，并且置位 ADCI，当 ADCS 或 ADCI 为高时，不能启动一个新的转换。

Bit 2~0: CHS2 ~ CHS1, ADC 模拟转换器输入通道选项。

单端模式：

CHS[2:0]	通道选择
0 0 0	AIN0 (P1.0)
0 0 1	AIN1 (P1.1)
0 1 0	AIN2 (P1.2)
0 1 1	AIN3 (P1.3)
1 0 0	AIN4 (P1.4)
1 0 1	AIN5 (P1.5)
1 1 0	AIN6 (P1.6)
1 1 1	AIN7 (P1.7)

ADCDH: ADC 数据寄存器

SFR 页 = 普通

SFR 地址 = 0xC6 复位值 = xxxx-xxxx

7	6	5	4	3	2	1	0
ADCDH.7	ADCDH.6	ADCDH.5	ADCDH.4	ADCDH.3	ADCDH.2	ADCDH.1	ADCDH.0
R	**R**	**R**	**R**	**R**	**R**	**R**	**R**

MA86E/L508 转换结果是一个 8 位无符号整型数据。输入测量值由'0'到 VDD x 255/256 计算，结果示例见下表：

输入电压	ADCDH
VDD x 255/256	0xFF
VDD x 128/256	0x80
VDD x 64/256	0x40
VDD x 32/256	0x20
0	0x00

P1AIO: 端口 1 仅模拟输入

SFR 页 = 普通

SFR 地址 = 0x92 复位值 = 0000-0000

7	6	5	4	3	2	1	0
P17AIO	P16AIO	P15AIO	P14AIO	P13AIO	P12AIO	P11AIO	P10AIO
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

0: Port 引脚有数字和模拟输入的功能

1: Port 引脚仅有模拟输了的功能， 当这个位置 1 后，读相对应的引脚寄存器位的值总是 0。

21.4. ADC 示例代码

(1). 规定功能: ADC 示例代码为系统时钟 SYSCLK=24MHz (200K), P1.0/P1.1/P1.2 是 P1AIO 设置的模拟输入:

汇编语言代码范例:

```
ADC_SAMPLE_P1x:
INITIAL_ADC_PIN:
        ORL     P1AIO,#00000111B            ; P1.0, P1.1, P1.2 =仅输入
        ANL     CKCON0,#11111000B           ; 系统时钟 SYSCLK/1

        MOV     ADCON0,#ADCEN               ; 使能 ADC 模块
        ; 延时 5 微妙(us)
        ; 调用....

Get_P10:
        MOV     ADCON0,#(ADCEN | SPEED1 | ADCS)
                                            ; 使能 ADC 模块和启动转换
                                            ; 转换速度为 200k 在系统时钟为 24MHz,选择 P1.0 为 ADC 输入引脚

        MOV     ACC,ADCON0                  ; 检测是否转换完成?
        JNB     ACC.4,$-3
        ANL     ADCON0,#~(ADCI | ADCS)      ; 清除 ADCI 和 ADCS
        MOV     AIN0_data,ADCDH             ; 保存 P1.0 的 ADC 数据
        ; to do ...

Get_P11:
        MOV     ADCON0,#(ADCEN | SPEED1 | ADCS | CHS0)  ; 选择 P1.1

        MOV     ACC,ADCON0                  ; 检测是否转换完成?
        JNB     ACC.4,$-3
        ANL     ADCON0,#~(ADCI | ADCS)      ; 清除 ADCI 和 ADCS
        MOV     AIN1_data,ADCDH             ; 保存 P1.1 的 ADC 数据
        ; to do ...

Get_P12:
        MOV     ADCON0,#(ADCEN | SPEED1 | ADCS | CHS1)  ; 选择 P1.2

        MOV     ACC,ADCON0                  ; 检测是否转换完成?
```

```
        JNB     ACC.4,$-3
        ANL     ADCON0,#~(ADCI | ADCS)          ; 清除 ADCI 和 ADCS
        MOV     AIN2_data,ADCDH                 ; 保存 P1.2 的 ADC 数据

        ; to do ...

        RET
```

C 语言代码范例:

```
void main(void)
{
    unsigned char AIN0_data, AIN1_data, AIN2_data;

    P1AIO = 0x07;                               // P1.0, P1.1, P1.2 =仅输入
    CKCON0 &= ~0x07;                            //系统时钟 SYSCLK /1

    ADCON0 = ADCEN;                             //使能 ADC 模块
    //延时 5 微妙(us)
    // ...


    //  选择 P1.0
    ADCON0 = ADCEN | SPEED1 | ADCS;
                                        //使能 ADC 模块和启动转换
                                        //转换速度为 200k 在系统时钟为 24MHz,选择 P1.0 为 ADC 输入引脚

    while ((ADCON0 & ADCI) == 0x00);            //等待完成
    ADCON0 &= ~(ADCI | ADCS);
    AIN0_data = ADCDH;
    // to do ...

    // select P1.1
    ADCON0 = ADCEN | SPEED1 | ADCS | CHS0;   //  选择 P1.1

    while ((ADCON0 & ADCI) == 0x00);            //等待完成
    ADCON0 &= ~(ADCI | ADCS);
    AIN1_data = ADCDH;
    // to do ...
```

```
    // select P1.2
    ADCON0 = ADCEN | SPEED1 | ADCS | CHS1;  //  选择 P1.2

    while ((ADCON0 & ADCI) == 0x00);                //等待完成
    ADCON0 &= ~(ADCI | ADCS);
    AIN2_data = ADCDH;

    // to do ...

    while (1);

}
```

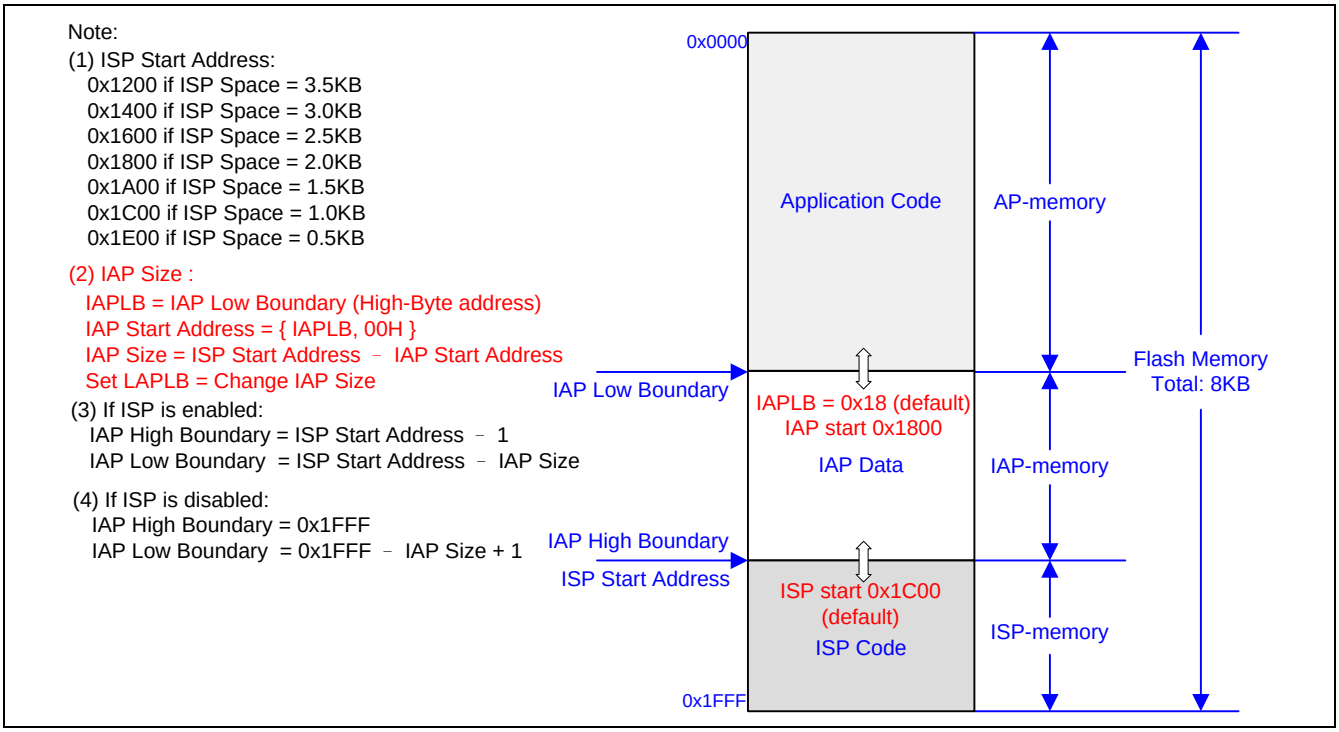
22. ISP 和 IAP

MA86E/L508 的 Flash 存储空间划分为 AP 存储空间，IAP 存储空间和 ISP 存储空间。AP 存储空间用来存储用户应用程序；IAP 存储空间用来存储非易失性数据；ISP 存储空间用来存储在线编程的引导码。系统在 ISP 空间运行时，MCU 可以修改 AP 和 IAP 来更新软件。如果 MCU 运行在 AP 空间，那么 MCU 就仅能修改 IAP 存储的数据。

22.1. MA86E/L508 Flash 存储空间配置

MA86E/L508 总共有 8K 字节的 Flash，图 22-1 显示了 Flash 的配置。ISP 存储空间可以被禁止或由硬件选项配置最大 3.5K 字节。IAP 存储空间大小由 IAP 低边界和高边界决定。IAP 低边界由 IAPLB 寄存器的值决定。IAP 高边界与 ISP 的起始地址相关，ISP 存储空间由硬件选项决定。IAPLB 寄存器值由硬件选项配置或 AP 软件编程设定。所有 AP, IAP 和 ISP 存储空间共享总 8K 字节的存储空间。

图 22–1. **MA86E/L508** Flash 存储空间配置



注意:

笙泉公司样品的默认设置是：1K ISP, 1K IAP 和加密。1K ISP 文件是有笙泉专利的通过一条线就能在线下载的 1-线 ISP 协议。1K IAP 大小可以通过应用程序软件来重新配置。

22.2. MA86E/L508 Flash 在 ISP/IAP 的访问

MA86E/L508 给 ISP 和 IAP 应用提供两种 flash 访问模式：编程模式及读取模式。MCU 软件使用这两种模式去更新 Flash 的数据和获取 Flash 的数据。本章展示了不同 Flash 模式的流程图和范例代码。

22.2.1. ISP/IAP Flash 编程模式

MA86E/L508 编程模式提供 Flash 存储空间的字节写操作来更新数据。IFADRH 和 IFADRL 指向 Flash 的物理字节地址。IFD 存储编程到 Flash 的内容。 图 22–2 展示了 ISP/IAP 操作的 Flash 字节编程流程。

图 22–2. ISP/IAP 字节编程流程

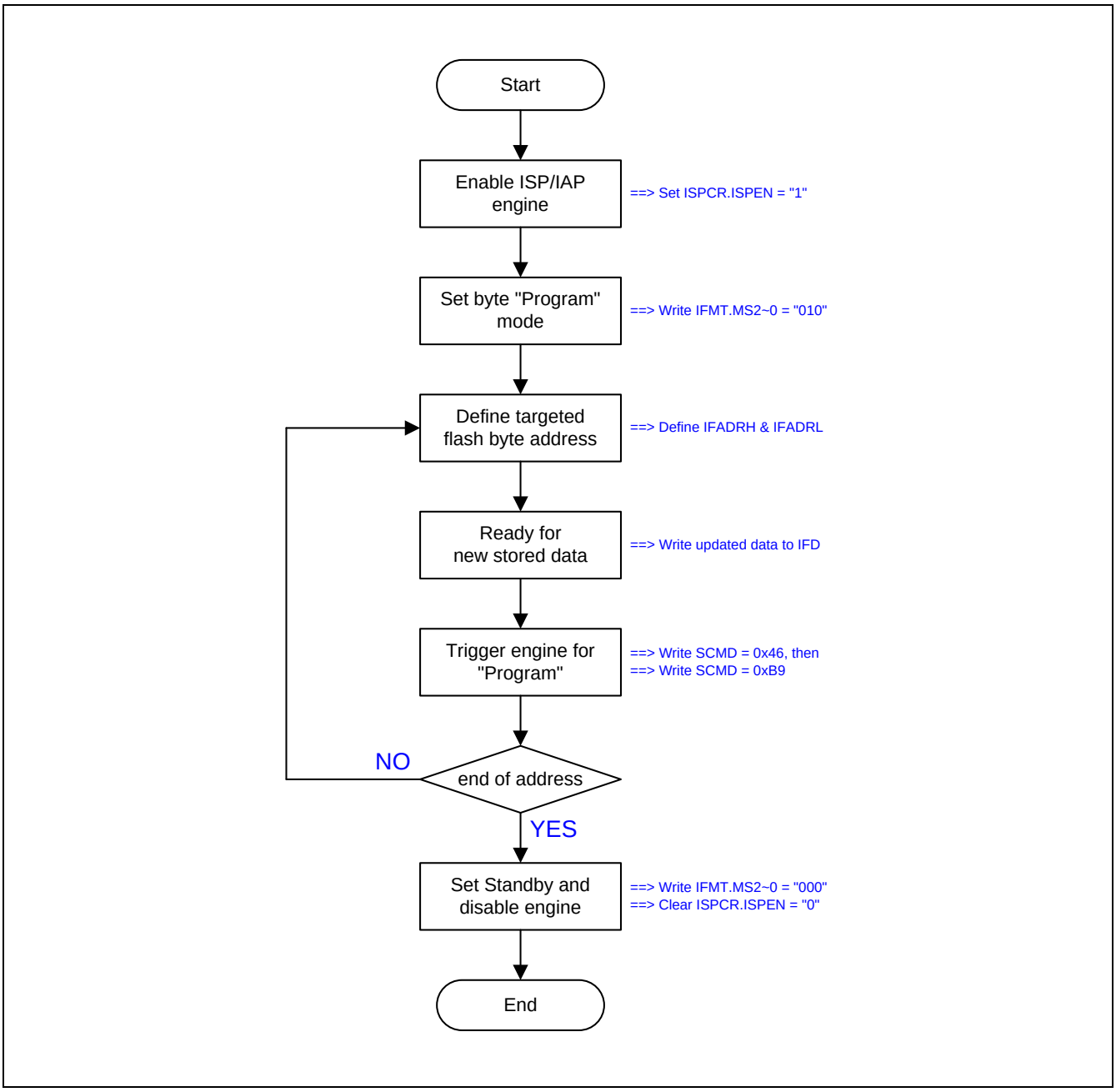


图 22–3 展示了 ISP/IAP 字节编程操作的示例代码

图 22–3. ISP/IAP 字节编程的示例代码

```
MOV     ISPCR,#10000011b ; ISPCR.7=1, 使能 ISP

MOV     IFMT,#02h          ; 选择 ISP/IAP 编程模式

MOV     IFADRH,??          ; 填写 [IFADRH,IFADRL]的字节地址
MOV     IFADRL,??          ;

MOV     IFD,??             ; 填写 IFD 的编程数据

MOV     SCMD,#46h          ;触发 ISP/IAP 处理
MOV     SCMD,#0B9h         ;

;此时, MCU 将暂停直到 ISP/IAP 处理的完成

MOV     IFMT,#00h          ; 选择 ISP/IAP 备用模式
MOV     ISPCR,#00000000b ; ISPCR.7 = 0, 禁止 ISP
```

22.2.2. ISP/IAP Flash 读取模式

MA86E/L508 读取模式提供从 Flash 存储空间获取已存储数据的字节读取操作。IFADRH 和 IFADRL 指向 Flash 的物理字节地址。IFD 存储从 Flash 读取到的内容。建议在数据编程或页擦除之后通过读取模式核对 Flash 数据。图 22–4 展示了 ISP/IAP 操作下的 Flash 字节读取流程。

图 22–4. ISP/IAP 字节读取流程

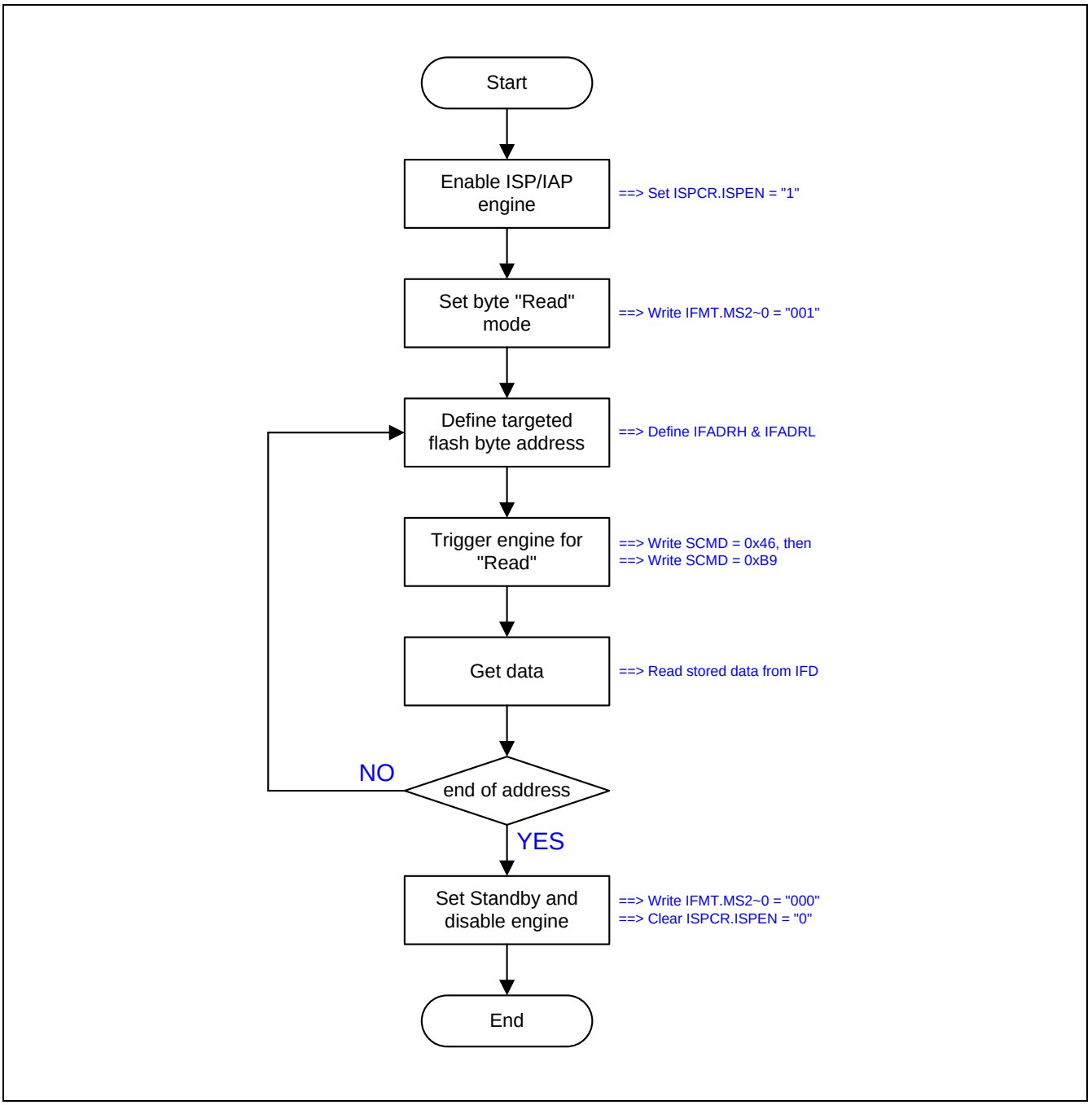


图 22–5 展示了 ISP/IAP 字节读取操作的范例代码

图 22–5. ISP/IAP 字节读取的范例代码

```
MOV     ISPCR,#10000011b ; ISPCR.7=1, 使能 ISP

MOV     IFMT,#01h        ; 选择读取模式

MOV     IFADRH,??        ; 填写 [IFADRH,IFADRL]的字节地址
MOV     IFADRL,??        ;

MOV     SCMD,#46h        ; 触发 ISP/IAP 处理
MOV     SCMD,#0B9h       ;

; 此时, MCU 将暂停直到 ISP/IAP 处理的完成

MOV     A,IFD            ; 此时,读取的数据存在 IFD

MOV     IFMT,#00h        ; 选择备用模式
MOV     ISPCR,#00000000b ; ISPCR.7 = 0, 禁止 ISP
```

22.3. ISP 操作

ISP 意指在系统可编程，不需要在实际的终端产品上移除 MCU 芯片就可以更新用户的应用程序(AP 存储空间)和非易失性应用数据(IAP 存储空间)。这个可使用性就有一个宽的现场应用范围。ISP 模式使用引导程序来编程 AP 存储空间和 IAP 存储空间。

注意:

(1) 在用 ISP 功能之前，使用者必须先配置 ISP-存储器空间并用通用烧写器或笙泉的烧写器插入 ISP 文件到 ISP-存储器中。

(2) ISP-存储器中的 ISP 文件 code 只能下载 AP-存储器和非易失的 IAP-存储器。

在 ISP 操作完成之后,软件写 “001”到 ISPCR.7 ~ ISPCR.5 这样会触发一个软件**复位**并且使 CPU 再启动到应用程序存储空间(AP)的 0x0000 地址。

如我们所知,ISP 代码的作用就是编程 AP 存储空间和 IAP 存储空间。因此，***MCU 为了执行 ISP 代码必须从 ISP 存储空间启动***。根据 MCU 如何从 ISP 存储空间启动，有两种方法执行在系统可编程。

22.3.1. 硬件访问 ISP

在上电复位时为了使 MCU 直接从 ISP 存储空间启动，MCU 的硬件选项 *HWBS* 和 *ISP 存储空间*必须使能。硬件选项的 ISP 进入方法叫做硬件访问。一旦 *HWBS* 和 *ISP 存储空间*使能,当上电复位时 MCU 总是从 ISP 存储空间启动去执行 ISP 代码(引导程序)。ISP 代码做的第一件事是核对是否有 ISP 请求。如果没有 ISP 请求,ISP 代码触发软件复位(设置 ISPCR.7~5 为“101”)使 MCU 在启动到 AP 存储空间去运行用户应用程序。

如果额外的硬件选项 HWBS2 与 HWBS 及 ISP 存储空间一起使能, MCU 在上电复位或**外部复位结束**之后总从 ISP 存储空间启动 。通过外部复位信号提供另外一个硬件访问进入 ISP 模式。第一上电复位之后，**MA86E/L508** 通过外部复位触发而执行 ISP 操作并且不用等待下一次的上电复位，这适合不断电系统去应用硬件访问 ISP 功能。

22.3.2. 软件访问 ISP

当 MCU 运行在 AP 存储空间时，软件访问 ISP 通过触发软件复位使 MCU 从 ISP 存储空间启动。这种情况，HWBS 或 HWBS2 不用使能。仅有的方法是当 MCU 运行在 AP 存储空间时同时设置 ISPCR.7~5 为“111”触发软件复位 MCU 从 ISP 存储空间启动。注意： ISP 存储空间必须通过硬件选项配置一个有效空间来保留 ISP 模式给软件访问 ISP 应用。

22.3.3. ISP 注意事项

开发 ISP 代码

尽管 ISP 存储空间的 ISP 代码是可编程的，ISP 存储空间在 MCU 的 Flash 中有一个 *ISP 起始地址*(见 图 22–1)，但是并不意味着你需要在你的源代码中加入这个偏移量（ *ISP 起始地址*)。代码偏移量硬件自动处理。用户只需像在 AP 存储空间开发应用程序一样开发。

ISP 期间中断

在触发 ISP/IAP flash 处理之后，内部 ISP 处理时 MCU 将停止一会儿直到处理完成。此时，如果中断已使能则中断事件将排队等待服务。一旦 ISP/IAP flash 处理完成，MCU 继续运行并且如果中断标志仍然有效则排队中的中断将立即服务。不过用户需要意识到下列事项：

(1) 当 MCU 停止在 ISP 处理时，中断不能实时服务。
(2) 低/高电平触发外部中断 nINTx,必须保持到 ISP 处理完成，否则将被忽略。

ISP 和空闲模式

MA86E/L508 不使用空闲模式执行 ISP 功能。反而 ISP/IAP 引擎操作 Flash 存储空间将冻结 CPU 的运行。一旦 ISP/IAP 运行结束，CPU 将继续并且推进紧跟着 ISP/AP 激活的指令。

ISP 的访问目标

如前所述, ISP 用来编程 AP 存储空间和 IAP 存储空间。一旦访问目标地址超出 IAP 存储空间的最后一个字节之外，硬件将自动忽略 ISP 处理的触发。这样 ISP 触发是无效的并且硬件不做任何事情。

ISP 的 Flash 持久期

内置 Flash 的持久期是 100 写周期，换句话说写周期不能超过 100 次。这样用户必须注意应用中需要频繁更新 AP 存储空间和 IAP 存储空间这一点。

22.4. 在应用可编程 (IAP)

MA86E/L508 内建一个在应用可编程(IAP)功能，当应用程序运行时在 Flash 存储空间里允许一些区域被应用成非易失性数据存储区。这个有用特点能使用在断电后还需要保存数据的应用中。这样不需要使用外部的串行 EEPROM (比如 93C46, 24C01, .., 等等)来保存非易失性的数据。

事实上，IAP的操作除了Flash存储空间被划分在不同的区域之外与ISP一样。ISP操作的可编程Flash范围在AP存储空间和IAP存储空间，而IAP操作的范围只在IAP存储空间。

注意:

*(1) **MA86E/L508**的IAP 特点，软件通过写SFR P页的IAPLB寄存器声明IAP 存储空间。IAP 存储空间也可以通过通用的烧入器/编程器或笙泉专利的烧入器/编程器来配置IAPLB 的初始值。*

(2) 执行IAP 的程序代码是在AP存储空间并且仅能编程IAP存储空间而不能编程ISP存储空间。

22.4.1. IAP-存储空间边界/范围

如果ISP 存储空间被声明，IAP存储空间范围由IAP和ISP起始地址决定如下列表：

IAP高边界= ISP起始地址 – 1.
IAP低边界= ISP起始地址 – IAP.

如果ISP 存储空间没有被声明，IAP存储空间范围由下列公式决定：

IAP高边界= 0x0FFF.
IAP低边界= 0x0FFF – IAP + 1.

例如，如果ISP 存储空间是1K 字节，这样ISP 的起始地址是0x0C00，并且IAP 存储空间是1K 字节，此时IAP 存储空间的范围就在0x0800 ~ 0x0BFF 。**MA86E/L508**的IAP 低边界由IAPLB 寄存器决定，IAPLB 寄存器可以在用户AP程序里用软件修改来调整IAP大小。

22.4.2. IAP-存储空间更新数据

ISP/IAP 相关的特殊功能寄存器见章节“22.5 ISP/IAP ”

在 AP 存储空间更新“一个字节”，用户能直接编程新数据到那个字节。在 AP 存储空间读取一个字节，用户可以使用 ***ISP/IAP Flash 读取模式*** 获取目标数据。

22.4.3. IAP 注意事项

IAP 期间中断

在触发 ISP/IAP flash 处理之后，内部 IAP 处理时 MCU 将停止一会儿直到处理完成。此时，如果中断已使能则中断事件将排队等待服务。一旦 ISP/IAP flash 处理完成，MCU 继续运行并且如果中断标志仍然有效则排队中的中断将立即服务。不过用户需要意识到下列事项：

(1) 当 MCU 停止在 IAP 处理时，中断不能实时服务。
(2) 低/高电平触发外部中断 nINTx,必须保持到 IAP 处理完成，否则将被忽略。

IAP 和空闲模式

MA86E/L508 不使用空闲模式执行 IAP 功能。反而 ISP/IAP 引擎操作 Flash 存储空间将冻结 CPU 的运行。一旦 ISP/IAP 运行结束，CPU 将继续并且推进紧跟着 ISP/AP 激活的指令。

IAP 的访问目标

如前所述, IAP 用来编程 IAP 存储空间。一旦访问目标地址不在 IAP 存储空间之内, 硬件将自动忽略 ISP 处理的触发。这样 IAP 触发是无效的并且硬件不做任何事情。

读取 IAP 数据的另一种方法

IAP 存储空间读取 Flash 数据，除了使用 Flash 的读取模式之外，另一个方法是使用“MOVC A,@A+DPTR”指令。这里，DPTR 和 ACC 各自填入想要的地址和偏移量。并且访问目标必须在 IAP 存储空间内，否则读取的数据将不确定。注意使用‘MOVC’指令比使用 Flash 的读取模式更快。

IAP 的 Flash 持久期

内置 Flash 的持久期是 100 写周期，换句话说写周期不能超过 100 次。这样用户必须注意应用中需要频繁更新 IAP 存储空间这一点。

22.5. ISP/IAP 寄存器

下面的特殊功能寄存器是与IAP/IAP和P页SFR的访问相关：

IFD: ISP/IAP Flash 数据寄存器

SFR 页 = 普通

SFR 地址 = 0xE2 复位值 = 1111-1111

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

IFD 是 ISP/IAP 操作的数据端口寄存器。在 ISP/IAP/P 页写操作时 IFD 的数据将被写入到期望的地址，在 ISP/IAP/P 页读操作时 IFD 的值是读到期望地址的数据。

IFADRH: ISP/IAP 高 8 位访问地址

SFR 页 =普通

SFR 地址 = 0xE3 复位值= 0000-0000

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

IFADRH 是所有 ISP/IAP 模式下的高 8 位地址。在 P 页模式下没有定义。

IFADRL: ISP/IAP 低 8 位访问地址

SFR 页 =普通

SFR 地址 = 0xE4 复位值= 0000-0000

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

IFADRL 是所有 ISP/IAP/P 页模式下的低 8 位地址。

IFMT: ISP/IAP Flash 模式表

SFR 页 =普通

SFR 地址 = 0xE5 复位值= xxxx-x000

7	6	5	4	3	2	1	0
--	--	--	--	--	MS.2	MS.1	MS.0
W	**W**	**W**	**W**	**W**	**R/W**	**R/W**	**R/W**

Bit 7~3: 保留。当写 IFMT 时，软件必须在这些位上写”0”。

Bit 2~0: ISP/IAP/ P 页操作模式选择

MS[2:0]	模式
0 0 0	备用
0 0 1	AP/IAP-存储器读
0 1 0	AP/IAP-存储器编程
0 1 1	保留
1 0 0	P 页 SFR 写
Others	保留

IFMT 是用来选择闪存是用执行众多的 ISP/IAP 功能还是选择 P 页寄存器的访问。

IAPLB: IAP 低边界

SFR 页 = **仅 P 页**

SFR 地址 = 0x03 复位值= 0001-1000

7	6	5	4	3	2	1	0
IAPLB							0
W	**W**	**W**	**W**	**W**	**W**	**W**	**W**

Bit 7~0: IAPLB 决定 IAP 存储区的最低边界。由于闪存有每页有 512 字节，所以 IAPLB 必须是偶数。

为了读 IAPLB，MCU 必须将 IMFT 设置成 IAPLB 读模式并且置位 ISPCR.ISPEN，然后顺序将 0x46 和 0xB9 写入 SCMD，最后 IAPLB 的值就会在 IFD 中。如果写 IAPLB，MCU 首先将 IAPLB 的设定值写入 IFD 中，其次选择 IMFT ，使能 ISPCR.ISPEN ； 然后顺序将 0x46 和 0xB9 写入 SCMD。这样 IAPLB 就会更新到最新的顺序。.

由 IAPLB 及 ISP 起始地址决定的 IAP 存储区见下列表。

IAP 低边界 = IAPLBx256, 及

IAP 高边界 = ISP 起始地址 – 1.

例如， IAPLB=0x12 及 ISP 起始地址是 0x1C00, 那么 IAP 存储区就是 0x1200 ~ 0x1BFF.

另外要注意一点，IAP 的低边界地址不能大于 ISP 的起始地址。

SCMD:系列命令数据寄存器

SFR 页　　=普通

SFR 地址　　= 0xE6　　复位值= xxxx-xxxx

7	6	5	4	3	2	1	0
SCMD							
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

SCMD 是激活 ISP/IAP/ P 页的命令端口。如果 SCMD 连续填入 0x46h, 0xB9h 并且 ISPCR.7 = 1，ISP/IAP/ P 页被激活

ISPCR: ISP 控制寄存器

SFR 页　　=普通

SFR 地址　　= 0xE7　　复位值= 0000-0xxx

7	6	5	4	3	2	1	0
ISPEN	SWBS	SWRST	CFAIL	--	--	--	--
R/W	**R/W**	**R/W**	**R/W**	**W**	**W**	**W**	**W**

Bit 7: ISPEN, ISP/IAP/P 页操作使能。

0: 所有的 ISP/IAP/ P 页编程/读都是被禁止的。

1: 使能 ISP/IAP/ P 页编程/读功能

Bit 6: SWBS, 软件执行起始选择控制。

0: 软件复位从主存储区开始执行。

1: 软件复位从 ISP 存储区开始执行。

Bit 5: SWRST, 软件复位触发控制。

0: 没有操作

1: 产生软件系统复位，硬件自动清零

Bit 4: CFAIL, ISP/IAP 操作命令失败指示。

0: 最后一次 ISP/IAP 命令成功。

1: 最后一次 ISP/IAP 命令失败。失败的原因是 flash 存储空间访问错误。

Bit 3~0: 保留。当写 ISPCR 时，软件必须在这个位上写”0”。

22.6. ISP/IAP 示例代码

(1). 规定功能: ISP/IAP flash 读取的通用功能子程序

汇编语言代码范例:

```
_ixp_read:
ixp_read:
    MOV     ISPCR,#ISPEN                ; 使能 ISP/IAP 功能
    MOV     IFMT,#MS0                   ; IFMT =0x01，ISP/IAP 为读取模式

    MOV     SCMD,#046h                  ;触发 ISP/IAP 处理
    MOV     SCMD,#0B9h                  ;

    MOV     IFMT,#000h                  ; IFMT =0x00，选择备用功能
    ANL     ISPCR,#~ISPEN               ; 禁止 ISP/IAP 功能

    RET
```

C 语言代码范例:

```
void ixp_read (void)
{
    ISPCR = ISPEN;                      //使能 ISP/IAP 功能
    IFMT = IxP_Flash_Read;              // IFMT =0x01，ISP/IAP 为读取模式

    SCMD = 0x46;                        //触发 ISP/IAP 处理
    SCMD = 0xB9;                        //

    IFMT = Flash_Standby;               //IFMT =0x00，选择备用功能
    ISPCR &= ~ISPEN;                    //禁止 ISP/IAP 功能
}
```

(2). 规定功能: ISP/IAP flash 编程的通用功能子程序

汇编语言代码范例:

```
_ixp_program:
```

```
ixp_program:
    PUSH    ACC

    PUSH    IFADRH                      ;
    PUSH    IFADRL                      ;
    PUSH    IFD                         ;

    MOV     IFADRL,#WDTCR               ;
    MOV     IFD,WDTCR                   ;
    ORL     IFD,#CLRW                   ;
    CALL    page_p_sfr_write            ;

    POP     IFD                         ;
    POP     IFADRL                      ;
    POP     IFADRH                      ;

    MOV     ISPCR,#ISPEN                ; 使能 ISP/IAP 功能
    MOV     IFMT,#MS1                   ; ISP/IAP 写模式，IFMT =0x02
    MOV     SCMD,#046h                  ;
    MOV     SCMD,#0B9h                  ;

    PUSH    IFD                         ;
    MOV     A,IFD                       ;
    CPL     A                           ;
    MOV     IFD,A                       ;

    MOV     IFMT,#MS0                   ; IFMT =0x01，ISP/IAP 为读取模式
    MOV     SCMD,#046h                  ;
    MOV     SCMD,#0B9h                  ;

;   if(reg[2] == IFD)
    POP     ACC                         ;
    CJNE    A,IFD,ixp_first_progrma_fail    ;
    JMP     ixp_progrma_Pass            ;

ixp_first_progrma_fail:
;   page_p_sfr_write (WDTCR_P,(WDTCR | CLRW)); //
    MOV     IFD,A                       ;
```

```
    PUSH    IFADRH                  ;
    PUSH    IFADRL                  ;
    PUSH    IFD                     ;

    MOV     IFADRL,#WDTCR           ;
    MOV     IFD,WDTCR               ;
    ORL     IFD,#CLRW               ;
    CALL    page_p_sfr_write        ;

    POP     IFD                     ;
    POP     IFADRL                  ;
    POP     IFADRH                  ;

    ANL     ISPCR,#~CFAIL           ;
    MOV     IFMT,#MS1               ; ISP/IAP 写模式，IFMT =0x02
    MOV     SCMD,#046h              ;
    MOV     SCMD,#0B9h              ;

    PUSH    IFD                     ;
    MOV     A,IFD                   ;
    CPL     A                       ;
    MOV     IFD,A                   ;

    MOV     IFMT,#MS0               ; IFMT =0x01，ISP/IAP 为读取模式
    MOV     SCMD,#046h              ;
    MOV     SCMD,#0B9h              ;

;   if(reg[2] == IFD)
    POP     ACC
    CJNE    A,IFD,ixp_second_progrma_Fail
ixp_progrma_Pass:
;   SETB    ixp_program_state       ; 成功为 1
    CLR     ixp_program_state       ; 成功为 0

end_ixp_program:
    MOV     IFMT,#000h              ;
    ANL     ISPCR,#~ISPEN           ;
```

```
    POP     ACC                         ;
    RET                                 ;

ixp_second_progrma_Fail:
;   CLR     ixp_program_state           ; 失败为 0
    SETB    ixp_program_state           ; 失败为 1
    JMP     end_ixp_program             ;
```

C 语言代码范例:

```
bit ixp_program(void)
{   unsigned char reg[3];

    reg[0] = IFADRH;                    //
    reg[1] = IFADRL;                    //
    reg[2] = IFD;

    IFADRL = WDTCR_P;                   //
    IFD = (WDTCR | CLRW);               //
    page_p_sfr_write ();                //

    IFADRH = reg[0];                    //
    IFADRL = reg[1];                    //
    IFD = reg[2];                       //

    ISPCR = ISPEN;                      //使能 ISP/IAP 功能
    IFMT = IxP_Flash_Program;           // ISP/IAP 写模式，IFMT =0x02

    SCMD = 0x46;
    SCMD = 0xB9;

    IFD = ~reg[2];                      //

    IFMT = IxP_Flash_Read;              // IFMT =0x01，ISP/IAP 为读取模式
    SCMD = 0x46;                        //
    SCMD = 0xB9;                        //

    if(reg[2] == IFD)                   //
```

```
    {
      IFMT = Flash_Standby;                  // IFMT =0x00，选择备用功能
      ISPCR &= ~ISPEN;                       //
      return(Pass);                          //
    }
  else                                       //
  {   IFADRL = WDTCR_P;                      //
      IFD = (WDTCR | CLRW);                  //
      page_p_sfr_write ();                   //

      IFADRH = reg[0];                       //
      IFADRL = reg[1];                       //
      IFD = reg[2];                          //

      ISPCR &= ~CFAIL;                       //
      IFMT = IxP_Flash_Program;              // ISP/IAP 写模式，IFMT =0x02

      SCMD = 0x46;                           //
      SCMD = 0xB9;                           //

      IFD = ~reg[2];                         //

      IFMT = IxP_Flash_Read;                 // IFMT =0x01，ISP/IAP 为读取模式
      SCMD = 0x46;                           //
      SCMD = 0xB9;                           //

      IFMT = Flash_Standby;                  // IFMT =0x00，选择备用功能
      ISPCR &= ~ISPEN;                       //

      if(reg[2] != IFD)                      //
      {
          return(Fail);                      //
      }
                else                         //
      {
          return(Pass);                      //
      }
  }
```

```
}
```

(3). 规定功能: 失败(CFAIL)检测的 ISP 示例代码,如果是失败(CFAIL)则写两次。

汇编语言代码范例:

```
isp_hw_approached:
    MOV     DPH,#High(00000h)           ;
    MOV     DPL,#LOW(00000h)            ;

isp_hw_write_loop:
    MOV     IFADRL,DPL                  ;
    MOV     IFADRH,DPH                  ;
    MOV     IFD,#055h                   ; ISP 编程数据
    CALL    ixp_program                 ;

;   JNB     ixp_program_state,isp_hw_write_fail
                                        ; 失败为 0

    JB      ixp_program_state,isp_hw_write_fail
                                        ; 失败为 1
isp_hw_write_next:
    INC     DPTR                        ;

    MOV     A,DPH                       ;
    CJNE    A,#01Ch,isp_hw_write_loop   ;

    ANL     ISPCR,#~SWBS                ;
    ORL     ISPCR,#SWRST                ;

isp_hw_write_fail:
;       to do ...
    JMP     isp_hw_write_next           ;
```

C 语言代码范例:

```
    unsigned short addr=0x0000;         //
    do{ IFADRH = (unsigned char)(addr >>8);
```

```
        IFADRL = (unsigned char)addr ;          //
        IFD = 0x55;                             //
        if(ixp_program() == Fail);              //
        {
//              to do ...
        }
    }while(++addr != 0x1C00);                   //
    ISPCR = SWRST;                              // 选择 AP 启动和软件复位
```

(4). 规定功能:软件启动 ISP 的示例代码

汇编语言代码范例:

```
    MOV     ISPCR,#(SWBS|SWRST)             ;
```

C 语言代码范例:

```
    ISPCR = (SWBS | SWRST) ;                //
```

(5). 规定功能: IAPLB 更改片内 flash 的 4K IAP 边界

汇编语言代码范例:

```
    MOV     IFADRL,#IAPLB                   ;
    MOV     IFD,#(HIGH(4096))&0xFE          ;
    CALL    page_p_sfr_write                ;
```

C 语言代码范例:

```
    IFADRL = IAPLB;                         //
    IFD = (((unsigned char)(4096 >> 8)) & 0xFE);   //
    page_p_sfr_write();                     //
```

23. P 页 SFR 访问

MA86E/L508 内建一个特别的 P 页寄存器 (P 页) 用来存储 MCU 操作的控制寄存器。这些特殊功能寄存器在不同 IFMT 下通过 ISP/IAP 操作来访问。在 P 页访问时，IFADRH 必须设置为”00”及 IFADRL 索引 P 页内特殊功能寄存器地址。如果 IFMT= 04H 则 P 页写操作，在 SCMD 激活之后 IFD 的数据会被载入到 IFADRL 索引的特殊功能寄存器。如果 IFMT= 05H 则 P 页读操作，在 SCMD 激活之后 IFD 的数据保存着 IFADRL 索引的特殊功能寄存器的值。

下面描述的是 P 页里的特殊功能寄存器：

IAPLB: IAP 低边界

SFR 页 = 仅 P 页

SFR 地址 = 0x03 复位值 = 0001-1000

7	6	5	4	3	2	1	0
IAPLB							**0**
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~0: IAPLB 决定 IAP 存储区的最低边界

读 IAPLB，MCU 必须先将 IAPLB 在 P 页的地址写如 IFADRL，IMFT 设置成 IAPLB 读模式，然后置位 ISPCR.ISPEN，然后顺序将 0x46 和 0xB9 写入 SCMD，最后 IAPLB 的值就会在 IFD 中。写 IAPLB，MCU 首先将 IAPLB 的新值写入 IFD 中，其次索引 IFADRL ，选择 IMFT ，使能 ISPCR.ISPEN ；然后设置 SCMD 。这样 IAPLB 就会更新到最新的顺序。.

由 IAPLB 及 ISP 起始地址决定的 IAP 存储区见下列表。

IAP 低边界 = IAPLBx256, 及

IAP 高边界 = ISP 起始地址 – 1.

例如， IAPLB=0x12 及 ISP 起始地址是 0x1C00, 那么 IAP 存储区就是 0x1200 ~ 0x1BFF.

另外要注意一点，IAP 的低边界地址不能大于 ISP 的起始地址。

CKCON2: 时钟控制寄存器 2

SFR 页　　=仅 P 页

SFR 地址 = 0x40　　复位值= 1101-xx00

7	6	5	4	3	2	1	0
XTGS1	XTGS0	XTALE	IHRCOE	**0**	**0**	OSCS1	OSCS0
R/W	**R/W**	**R/W**	**R/W**	**W**	**W**	**R/W**	**R/W**

Bit 7~6: XTGS1~XTGS0, 振荡驱动控制寄存器

XTGS1, XTGS0	增益定义
0, 0	32.768K 的增益
1, 1	2MHz ~ 25MHz 的增益
其他	保留

Bit 5: XTALE, 外部晶振(XTAL)使能

0: 禁止（XTAL）振荡电路。 此时 XTAL2 及 XTAL1 当做 P 4.0 及 P 4.1。

1: 使能（XTAL）振荡电路。如果此位是通过 CPU 软件来设置的话，软件必须检测 **XTOR** (AUXR1.4)是否为 1 来判断晶振是否准备好用于时钟选择。

Bit 4: IHRCOE, 内部高频 RC 振荡使能。默认值置 1 用于 MCU 时钟来源

0: 禁止内部高频 RC 振荡。

1: 使能内部高频 RC 振荡。如果此位是通过 CPU 软件来设置的话，则在 IHRCOE 使能之后需要 32 微秒才能稳定输出。

Bit 3~2: 保留。当写 CKCON2 时，软件必须在这个位上写”0”。

Bit 1~0: OSCS[1:0], 振荡输入来源选择. 振荡输入默认选择是 IHRCO。

OSCS[1:0]	振荡输入来源选择
0　0	IHRCO
0　1	XTAL
1　0	ILRCO
1　1	ECKI, 外部时钟输入 (P4.0)作为振荡输入

PCON2: 电源控制寄存器 2

SFR 页 = 仅 P 页

SFR 地址 = 0x44 复位值 = x0xx-xx01

7	6	5	4	3	2	1	0
--	AWBOD0	--	--	--	--	BO0RE	**1**
W	**R/W**	**W**	**W**	**W**	**W**	**R/W**	**W**

Bit 7: 保留。当写 PCON2 时，软件必须在这个位上写”0”。

Bit 2: AWBOD0, 在掉电模式下 BOD0 唤醒使能。仅 **MA86L508** 支持这个功能

0: 在掉电模式下禁止 BOD0 唤醒。

1: 在掉电模式下 BOD0 保持运行。

Bit 5~2: 保留。当写 PCON2 时，软件必须在这个位上写”0”。.

Bit 1: BO0RE, BOD0 复位使能. 它的初始值从 OR1.BO0REO 取反后加载

0: 禁止 BOF0 置位时 BOD0 触发一个系统复位。

1: 使能 BOF0 置位（4.2V(E) 或 2.4V{L}）时 BOD0 触发一个系统复位。

Bit 0: 保留。当写 PCON2 时，软件必须在这个位上写”1”。

SPCON0: SFR 页面控制 0

SFR 页 =仅 P 页

SFR 地址 = 0x48 复位值= xxx0-x000

7	6	5	4	3	2	1	0
RTCCTL	--	--	WRCTL	--	CKCTL0	PWCTL1	PWCTL0
W	**W**	**W**	**R/W**	**W**	**R/W**	**R/W**	**R/W**

Bit 7: RTCCTL. RTCCR SFR 访问控制

如果 RTCCTL 置位，将禁止 RTCCR SFR 在普通页做修改。RTCCR 在普通页仅保持 SFR 读取的功能。但是软件总是可以在 SFR 的 P 页修改 RTCCR。

Bit 6~5: 保留。当写 SPCON0 时，软件必须在这个位上写”0”。

Bit 4: WRCTL. WDTCR SFR 访问控制

如果 WRCTL 置位，它将禁止在通用页面上修改 WDTCR SFR，仅保持读这个特殊寄存器的功能。但在 SFR P 页上，软件总是有修改它的能力。

Bit 3: 保留。当写 SPCON0 时，软件必须在这个位上写”0”。

Bit 2: CKCTL0. CKCON0 SFR 访问控制

如果 CKCTL0 置位，它将禁止在通用页面上修改 CKCON0 SFR，仅保持读这个特殊寄存器的功能。但在 SFR P 页上，软件总是有修改它的能力。

Bit 1: PWCTL1. PCON1 SFR 访问控制

如果 PWCTL1 置位，它将禁止在通用页面上修改 PCON1 SFR，仅保持读这个特殊寄存器的功能。但在 SFR P 页上，软件总是有修改它的能力。

Bit 0: PWCTL0. PCON0 SFR 访问控制

如果 PWCTL0 置位，它将禁止在通用页面上修改 PCON0 SFR，仅保持读这个特殊寄存器的功能。但在 SFR P 页上，软件总是有修改它的能力。

DCON0: 设备控制 0

SFR 页 = 仅 P 页

SFR 地址 = 0x4C 复位值 = 10xx-xx1x

7	6	5	4	3	2	1	0
HSE	IAPO	--	--	--	--	RSTIO	--
R/W	**R/W**	**W**	**W**	**W**	**W**	**R/W**	**W**

Bit 7: HSE, 高速操作使能

0: 选择 MCU 运行在低速模式减缓内部电路的速度从而减少功耗。

1: 使能 MCU 高速运行(F_{SYSCLK} > 6MHz)。

Bit 6: IAPO, 仅用作为 IAP 功能用

0: 保持 IAP 区域可用于 IAP 功能和执行代码。

1: 禁止在 IAP 区域执行代码，该区域仅用于 IAP 功能。

Bit 5~2: 保留。当写 DCON0 时，软件必须在这个位上写”0”。

Bit 1: RSTIO, RST 引脚功能为复位脚

1: 选择 RST 引脚作为复位脚

0: 选择 RST 引脚作为 P3.6

Bit 0: 保留。当写 DCON0 时，软件必须在这个位上写”0”。

23.1. P 页示例代码

(1). 规定功能: P 页特殊功能寄存器(SFR)读取的通用功能子程序

汇编语言代码范例:

```
_page_p_sfr_read:
page_p_sfr_read:
    MOV     IFADRH,000h
    MOV     IFMT,#(MS2|MS0)             ; P 页读取，IFMT =0x05

    ANL     ISPCR,#CFAIL                ;
    ORL     ISPCR,#ISPEN                ; 使能 IAP/ISP 功能

    MOV     SCMD,#046h                  ;
    MOV     SCMD,#0B9h                  ;

    MOV     IFMT,#000h                  ; IAP/ISP 备用模式，IFMT =0x00
    ANL     ISPCR,#~ISPEN               ; 禁止 IAP/ISP 功能

    RET
```

C 语言代码范例:

```
void page_p_sfr_read (void)
{
    IFADRH = 0x00;                      //

    ISPCR = ISPEN;                      //使能 IAP/ISP 功能
    IFMT = (MS0 | MS2);                 // P 页读取，IFMT =0x05

    SCMD = 0x46;                        //
    SCMD = 0xB9;                        //

    IFMT = Flash_Standby;               // IAP/ISP 备用模式，IFMT =0x00
    ISPCR &= ~ISPEN;                    //禁止 IAP/ISP 功能
}
```

(2). 规定功能: P 页特殊功能寄存器(SFR)写的通用功能子程序

汇编语言代码范例:

```
_page_p_sfr_write:
page_p_sfr_write:
    MOV     IFADRH,000h                 ;

    MOV     ISPCR,#ISPEN                ; 使能 IAP/ISP 功能
    MOV     IFMT,#MS2                   ; P 页写，IFMT =0x04

    MOV     SCMD,#046h                  ;
    MOV     SCMD,#0B9h                  ;

    MOV     IFMT,#000h                  ; IAP/ISP 备用模式，IFMT =0x00
    ANL     ISPCR,#~ISPEN               ; 禁止 IAP/ISP 功能

    RET
```

C 语言代码范例:

```
void page_p_sfr_write (void)
{
    IFADRH = 0x00;

    ISPCR = ISPEN;                      //使能 IAP/ISP 功能
    IFMT = MS2;                         // P 页写，IFMT =0x04

    SCMD = 0x46;                        //
    SCMD = 0xB9;                        //

    IFMT = Flash_Standby;               // IAP/ISP 备用模式，IFMT =0x00
    ISPCR &= ~ISPEN;
}
```

(3). 规定功能: 使能 PWCTL0 在 P 页控制 PCON0.PD

汇编语言代码范例:

```
MOV     IFADRL,#SPCON0          ;
CALL    page_p_sfr_read         ;

ORL     IFD,#PWCTL0             ; 设置 PWCTL0
CALL    page_p_sfr_write        ;

MOV     IFD,PCON0               ; 设置 PCON0

ORL     IFD,#PD                 ; 写 PCON0 并且掉电
MOV     IFADRL,#PCON0_P         ;
CALL    page_p_sfr_write        ;
```

C 语言代码范例:

```
IFADRL = SPCON0;                //
page_p_sfr_read();              //

IFD |= PWCTL0;                  // 设置 PWCTL0
page_p_sfr_write();             //

IFD = PCON0;                    // 读取 PCON0

IFD |= PD;                      // 写 PCON0
IFADRL = PCON0_P;               //
page_p_sfr_write();             //
```

(4). 规定功能: 使能 CKCTL0 在 P 页更改系统时钟 SYSCLK 分频器 (CKCON0)

汇编语言代码范例:

```
MOV     IFADRL,#SPCON0          ;
CALL    page_p_sfr_read         ;

ORL     IFD,#CKCTL0             ; 设置 CKCTL0
```

```
CALL    page_p_sfr_write                ;

MOV     IFD,CKCON0                      ; 读取 CKCON0

ORL     IFD,#(AFS | SCKS0)              ; 写 CKCON0 及设置 AFS
MOV     IFADRL,#CKCON0_P                ; 系统时钟 SYSCLK / 2
CALL    page_p_sfr_write
```

C 语言代码范例:

```
IFADRL = SPCON0;                        //
page_p_sfr_read ();                     //

IFD |= CKCTL0;                          // 设置 CKCTL0
page_p_sfr_write();                     //

IFD = CKCON0;                           // 读取 CKCON0

IFD |= (AFS | SCKS0);                   //
IFADRL = CKCON0_P;                      //
page_p_sfr_write();                     // 写 CKCON0
```

24. 辅助特殊功能寄存器

AUXR0: 辅助寄存器 0

SFR 页 = 普通

SFR 地址 = 0xA1 复位值 = 0000-0000

7	6	5	4	3	2	1	0
P40OC1	P40OC0	P40FD	T0XL	P1FS1	P1FS0	INT1H	INT0H
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~6: P4.0 功能配置控制位 1 和 0。这两个仅当 IHRCO 被选择作为系统时钟时钟源时有效。在 Crystal 模式下，P4.0 和 P4.1 作为 XTAL2 和 XTAL1 功能。在外部时钟输入模式，P4.0 被定义为时钟输入引脚。在 IHROCO 模式下，P4.0 提供了下列选项用于 GPIO 或时钟源发生器。当 P40OC[1:0]索引到非 P4.0GPIO 功能时，P4.0 将驱动 IHRCO 输出，为其它器件提供时钟源。

P40OC[1:0]	P4.0 功能	I/O 模式
00	P4.0	By P4M0.0
01	IHRCO	By P4M0.0
10	IHRCO/2	By P4M0.0
11	IHRCO/4	By P4M0.0

对于 P4.0 时钟输出功能，建议置 P4M0.0 为 1，选择 P4.0 作为推挽输出模式。

Bit 5: P40FD, P4.0 快速驱动.

0: P4.0 输出默认驱动

1: P4.0 输出快速驱动使能。如果 P4.0 被配置为时钟输出，当 5V 应用 P4.0 输出频率超过 12MHz，或 3V 应用超过 6MHz 时，请使能这个位。

Bit 4: T0XL 是定时器 0 的预设值控制位。请参考 T0X12 (AUXR2.2)的 T0XL 功能定义。

Bit 3~2: P1.4 和 P1.5 交错功能选择

P1FS[1:0]	P1.4	P1.5
00	P1.4	P1.5
01	RXD0 的输入	TXD0 的输出
10	nINT0 的输入	nINT1 的输入
11	T0 的输入或 T0CKO 的输出	T1 的输入或 T1CKO 的输出

Bit 1: INT1H, INT1 高电平/上升沿触发使能

0: 保留 nINT1 口引脚上的低电平/下降沿触发 INT1

1: 设置 nINT1 口引脚上的高电平/上升沿触发 INT1

Bit 0: INT0H, INT0 高电平/上升沿触发使能

0: 保留 nINT0 口引脚上的低电平/下降沿触发 INT0

1: 设置 nINT0 口引脚上的高电平/上升沿触发 INT0

AUXR1: 辅助控制寄存器 1

SFR 页 =普通

SFR 地址 = 0xA2 复位值= 0000-0000

7	6	5	4	3	2	1	0
P3TWI	P4S0MI	P2PCA	XTOR	STAF	STOF	BPOC1	BPOC0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: P3TWI, TWI 接口在 P3.0~P3.1.

0: 禁止 TWI 功能移动到 P30 & P31.

1: 设置 TWI 功能在 P3.0&P3.1，定义如下:

P3.3 上的'TWI_SCL' 功能移动到 P3.0

P3.4 上的'TWI_SDA' 功能移动到 P3.1

Bit 6: P4S0MI, 串行口 SPI 主模式的 SOMI 输入在 P4.1.

0: 禁止 S0MI 功能移动到 P4.1.

1: 设置 S0MI 在 P4.1，定义如下:

P3.7 的'S0MI' 功能移动到 P4.1.

Bit 5: P2PCA, 所有 PCA 信号在 P2 口

0: 禁止 PCA 功能移动到 P2 口.

1: 设置 PCA 功能在 P2 口上，定义如下:

P3.4 上的'ECI' 功能移动到 P2.7.

P3.7 上的'CEX0' 功能移动到 P2.6.

P3.5 上的'CEX1' 功能移动到 P2.5.

P2.0 上的'CEX2' 功能被保留

P2.4 上的'CEX3' 功能被保留

Bit 4: XTOR, 晶振振荡准备好。仅读

0: 晶振振荡没准备好。

1: 晶振振荡准备好。当 XTALE 使能，XTOR 表示晶振振荡达到了启动计数。

Bit 3: STAF, TWI 起始位检测

0: 软件写”0”清除它

1: 硬件置位表示 TWI 总线出现了起始位

Bit 2: STOF, TWI 停止位检测.

0: 软件写”0”清除它

1: 硬件置位表示 TWI 总线出现了停止位

Bit 1~0: BPOC1~0,蜂鸣器输出控制位

BPOC[1:0]	P1.6 功能	I/O 模式
00	P1.6	By P1M0.6
01	ILRCO/64	By P1M0.6
10	ILRCO/32	By P1M0.6
11	ILRCO/16	By P1M0.6

P1.6 的蜂鸣器功能,推荐设置 P1M0.6 为“1”即选择 P1.6 作为推挽输出模式。

AUXR2: 辅助寄存器 2

SFR 页 =普通

SFR 地址 = 0xA3 复位值= 0000-0000

7	6	5	4	3	2	1	0
--	BTI	URM0X3	SM3	T1X12	T0X12	T1CKOE	T0CKOE
W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: 保留。当写 AUXR2 时，软件必须在这个位上写”0”。

Bit 6: BTI, 阻止 TI 在串行口中断。

0: 保留 TI 作为串行口中断源。

1: 阻止 TI 作为串行口中断源。

Bit 5: URM0X3, 串口模式 0 波特率选择

0: 选择 SYSCLK/12 作为 UART 模式 0 的波特率

1: 选择 SYSCLK/4 作为 UART 模式 0 的波特率

Bit 4: SM3, 串口模式控制位 3

0: 禁止串口模式 4

1: 使能 SM3 控制串口模式 4，SPI 主设备

Bit 3: T1X12, 当 C/T=0 时，定时器 1 时钟源选择。

0: 选择 SYSCLK/12 作为定时器 1 时钟源。

1: 选择 SYSCLK 作为定时器 1 时钟源。

Bit 2: T0X12, 当 C/T=0 时，定时器 0 时钟源选择。

T0XL, T0X12	定时器 0 时钟源选择
0 0	SYSCLK/12
0 1	SYSCLK
1 0	SYSCLK/48
1 1	SYSCLK/192

Bit 1: T1CKOE, 定时器 1 时钟输出使能

0: 禁止定时器 1 时钟输出。

1: 使能定时器 1 时钟在 P3.5 输出。

Bit 0: T0CKOE, 定时器 0 时钟输出使能。.

0: 禁止定时器 0 时钟输出

1: 使能定时器 0 时钟在 P3.4 输出。

BOREV: 位序相反寄存器

SFR 页 =普通

SFR 地址 = 0x96 复位值= 0000-0000

7	6	5	4	3	2	1	0
BOREV.7	BOREV.6	BOREV.5	BOREV.4	BOREV.3	BOREV.2	BOREV.1	BOREV.0
W	**W**	**W**	**W**	**W**	**W**	**W**	**W**
BOREV.0	BOREV.1	BOREV.2	BOREV.3	BOREV.4	BOREV.5	BOREV.6	BOREV.7
R	**R**	**R**	**R**	**R**	**R**	**R**	**R**

这个寄存器作用于读取数据跟写入数据位序相反的功能。

25. 硬件选项

MCU 的硬件选项定义了器件的性能，它不能由软件编程和控制。硬件选项仅能由通用编程器，“Megawin 8051 Writer U1” 或 “Megawin 8051 Writer U1 转接”(支持 ICP 编程功能，参见章节“26.4 编程功能”)来编程。整片擦除后，所有的硬件选项被设置成“禁止”状态，没有配置 ISP 空间和 IAP 空间 。 **MA86E/L508** 有下列的硬件选项

LOCK:

☑:使能. 加密上锁，使得用通用编程器读取代码锁定为 0xFF.

☐:禁止. 没有上锁

ISP 存储空间:

由其指定 ISP 空间的起始地址。它的高边界由 Flash 的结束地址限定，例如：0x0FFF。下表列举了 ISP 空间选项。默认设定， **MA86E/L508** ISP 空间被配置为 1K， 并嵌入了 Megawin 专用的 ISP 代码使用 Megawin 1-线 ISP 协议来执行在系统编程。

ISP-存储空间大小	ISP 起始地址
3.5K 字节	0x1200
3K 字节	0x1400
2.5K 字节	0x1600
2K 字节	0x1800
1.5K 字节	0x1A00
1K 字节	0x1C00
0.5K 字节	0x1E00
无 ISP 空间	--

HWBS:

☑:使能. 上电时，如果 ISP 空间有配置，则 MCU 从 ISP 空间启动

☐:禁止. MCU 总是从 AP 空间启动

HWBS2:

☑:使能. 如果 ISP 空间有配置，不仅上电，而且所有复位都是从 ISP 空间启动

☐:禁止. 由 HWBS 决定 MCU 从哪里启动

IAP 存储空间:

IAP 存储空间指定用户定义的 IAP 空间。IAP 存储空间可以由硬件选项或者 MCU 软件修改 IAPLB 来配置。默认，它被配置为 1KB.

BO0REO:

☑:使能. BOD0 将触发复位事件使得 CPU 从 AP 程序起始地址允许(2.2V)

☐:禁止. BOD0 不能触发 CPU 复位

WRENO:

☑:使能. 置位 WDTCR.WREN 使能 WDTF 产生一个系统复位

☐:禁止. 清零 WDTCR.WREN 禁止 WDTF 产生一个系统复位.

NSWDT: 不停止 WDT

☑:使能. 置位 WDTCR.NSW 在掉电模式下使能 WDT 运行 (watch 模式).

☐:禁止. 清零 WDTCR.NSW 在掉电模式下禁止 WDT 允许(禁止 Watch 模式)

HWENW: 硬件加载 WDTCR 的"ENW"

☑:使能. 上电后使能 WDT 并且加载 WRENO, NSWDT, HWWIDL 和 HWPS2~0 的内容到 WDTCR

☐:禁止. 上电后 WDT 不会自动使能

HWWIDL, HWPS2, HWPS1, HWPS0:

当 HWENW 使能，上电后这 4 个熔丝位的内容将被加载到 WDTCR

WDSFWP:

☑:使能. WDT 特殊寄存器，WDTCR 的 WREN, NSW, WIDL, PS2, PS1 和 PS0 位,将被写保护

☐:禁止. WDT 特殊寄存器，WDTCR 的 WREN, NSW, WIDL, PS2, PS1 和 PS0 位,由软件自由写

P36EN:

☑:使能. RSTIO (DCON0.1)将被清零，RST 引脚用作 P3.6 口

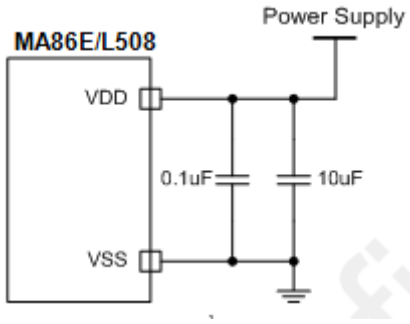
☐:禁止. RSTIO (DCON0.1)将被置位，保留 RST 引脚功能

26. 应用注意事项

26.1. 电源电路

MA86E/L508 的工作电源变化对于 E 类型可以从 4.2V 到 5.5V，对于 L 类型可以从 2.4V 到 3.6V，但是增加一些外部去耦和旁路电容是必须的，如图 26–1 所示。

图 26–1. 电源电路



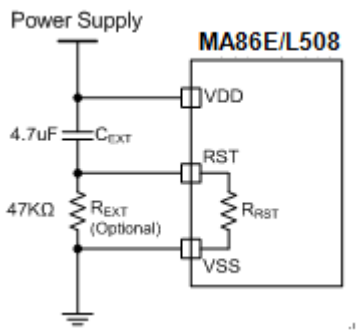
26.2. 复位电路

通常，上电可以成功产生上电复位，然而，为了上电时 MCU 产生一个可靠的复位，有必要加外部复位电路。外部复位电路如图 26–2 所示，它由一个连接到 VDD（电源）的电容 C_{EXT} 和一个连接到 VSS(地)的电阻组成。

一般的, R_{EXT} 是可选的，因为 RST 引脚有一个内部下拉电阻(R_{RST}). 这个对 VSS 的内部扩散电阻在仅使用一个外部对 VDD 的电容 C_{EXT} 时也可产生一个上电复位

R_{RST} 的值见章节“27.2 DC 特性特性”。

图 26–2. 复位电路



26.3. XTAL 振荡电路

为了能成功起振 (最大到 24MHz), 电容 C1 和 C2 是必须的，如图 26–3 所示。通常， C1 和 C2 使用相同的值。. 表 26–1 列举了 C1 & C2 在不同晶振下的值。

图 26–3. XTAL 振荡电路

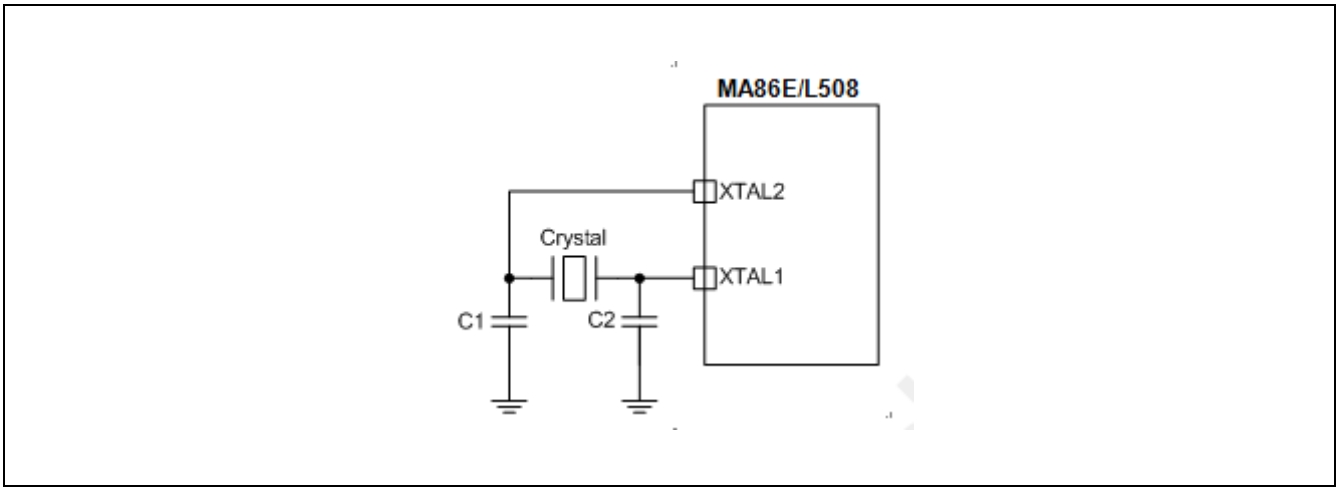


表 26–1. 振荡电路的电容 C1&C2 参照表

晶振	C1, C2 电容值
16MHz ~ 25MHz	10pF
6MHz ~ 16MHz	15pF
2MHz ~ 6MHz	33pF

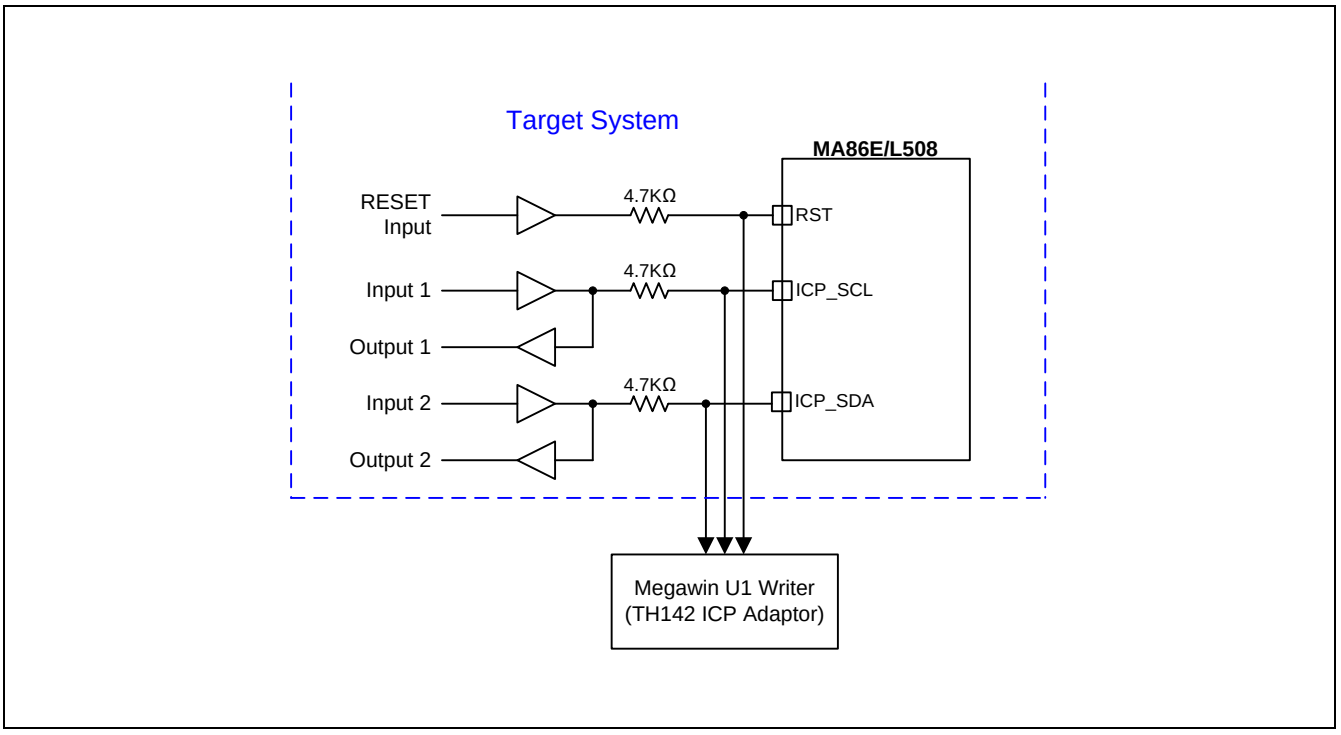
26.4. ICP 接口电路

MA86E/L508 包含一个笙泉专有的在芯片编程接口，它允许在元器件已经安装在产品上在芯片编程(ICP)。ICP 接口使用一个时钟线和个双向数据线完成主机向器件编程操作。

ICP 接口允许的 ICP_SCL/ICP_SDA 引脚与用户应用共享，使得可以实现在芯片 FLASH 编程。这是可行的，因为当芯片在 Halt 状态时执行 ICP 通信，此时芯片上的外围设备和用户软件都是失效的。在 halt 状态，ICP 接口能够安全的“借用”ICP_SCL (P1.6)和 ICP_SDA (P1.5) 引脚。 在大多应用中，必须用外部电阻来隔开 ICP 电路和用户应用电路。. 图 26–4.显示了一种典型的隔离方法。

强烈建议在目标系统建立 ICP 接口电路。它保留了整个软件编程和硬件选项配置的能力

图 26–4. ICP 接口电路



27. 电气特性

27.1. 绝对最大额定值

对 MA86E508:

参数	额定值	单位
工作温度	-40 ~ +85	°C
存储温度	-65 ~ + 150	°C
所有 I/O 口或复位脚对 VSS 电压	-0.5 ~ VDD + 0.5	V
VDD 对 VSS 电压	-0.5 ~ +6.0	V
通过 VDD 和 VSS 的最大电流	200	mA
任意端口最大输出灌电流	40	mA

*注: 器件超过"绝对最大额定值"列出的值，可能导致永久性损坏。上述值仅为运行条件极大值，我们不建议器件在该规范规定的范围外运行。器件长时间工作在最大值条件下，其可靠性会受到影响。

对 MA86L508:

参数	额定值	单位
工作温度	-40 ~ +85	°C
存储温度	-65 ~ + 150	°C
所有 I/O 口或复位脚对 VSS 电压	-0.3 ~ VDD + 0.3	V
VDD 对 VSS 电压	-0.3 ~ +4.2	V
通过 VDD 和 VSS 的最大电流	200	mA
任意端口最大输出灌电流	40	mA

*注: 器件超过"绝对最大额定值"列出的值，可能导致永久性损坏。上述值仅为运行条件极大值，我们不建议器件在该规范规定的范围外运行。器件长时间工作在最大值条件下，其可靠性会受到影响。

27.2. DC 特性

对 MA86E508: VDD = 5.0V±10%, VSS = 0V, T_A = 25 ℃并且每个机器周期执行 NOP 指令， 除非另有说明

标号	参数	测试环境	极限			单位
			最小	典型	最大	
输入/输出特性						
V_{IH1}	输入高电压(所有 I/O 端口)	除 RST, P4.0, P4.1 之外	2.0			V
V_{IH2}	输入高电压 (P4.0, P4.1)		3.5			V
V_{IH3}	输入高电压(RST/P3.6)		4.0			
V_{IL1}	输入低电压(所有 I/O 端口)	除 RST, P4.0, P4.1 之外			0.8	V
V_{IL2}	输入低电压(P4.0, P4.1)				1.0	V
V_{IL3}	输入低电压(RST/P3.6)				1.0	V
I_{IH}	输入高时漏电流(所有 I/O 端口)	V_{PIN} = VDD		0	10	uA
I_{IL1}	逻辑 0 输入电流 (P3 在准双向模式或输入口的片内上拉电阻)	V_{PIN} = 0.4V		28	60	uA
I_{IL2}	逻辑 0 输入电流 (所有仅输入或开漏口)	V_{PIN} = 0.4V		0	10	uA
I_{H2L}	逻辑 1 到 0 输入转换电流(P3 在准双向模式或输入口的片内上拉电阻)	V_{PIN} =1.8V		330	500	uA
I_{OH1}	输出高时电流(P3 在准双向模式)	V_{PIN} =2.4V	150	200		uA
I_{OH2}	输出高时电流(所有推挽输出口)	V_{PIN} =2.4V	12			mA
I_{OL1}	输出低时电流(所有 I/O 端口)	V_{PIN} =0.4V	12			mA
R_{RST}	内部复位下拉电阻			77		Kohm
电源功耗						
I_{OP1}	正常模式工作电流	SYSCLK = 24MHz @ IHRCO		**11**		mA
I_{OP2}		SYSCLK = 12MHz @ IHRCO		**6.8**		mA
I_{OP3}		SYSCLK = 6MHz @ IHRCO & HSE = 0		**4.8**		mA
I_{OP4}		SYSCLK = 3MHz @ IHRCO & HSE = 0		**2.9**		mA
I_{OP5}		SYSCLK = 24MHz @ XTAL		**11.5**		mA
I_{OP6}		SYSCLK = 12MHz @ XTAL		**7.2**		mA
I_{OP7}		SYSCLK = 6MHz @ XTAL & HSE = 0		**5**		mA
I_{OP8}		SYSCLK = 2MHz @ XTAL & HSE = 0		**2.6**		mA

I_{OPS1}	低速模式工作电流	SYSCLK = 24MHz/128 @ IHRCO **& HSE = 0**		**1.1**		mA
I_{IDLE1}	空闲模式工作电流	SYSCLK = 24MHz @ IHRCO		**4**		mA
I_{IDLE2}		SYSCLK = 24MHz @ XTAL		**4.9**		mA
I_{IDLE3}		SYSCLK = 24MHz/128 @ IHRCO		**1**		mA
I_{IDLE4}		SYSCLK = 24MHz/128 @ XTAL		**1.9**		mA
I_{IDLE5}		SYSCLK = 64KHz @ ILRCO		**22**		uA
I_{IDLE6}		**SYSCLK = 64KHz/128 @ ILRCO**		**13**		**uA**
I_{SUB1}	副频模式工作电流	SYSCLK = 64KHz @ ILRCO & HSE = 0		**60**		uA
I_{SUB2}		SYSCLK = 64KHz/128 @ ILRCO & HSE = 0		**13**		uA
I_{SUB2}		SYSCLK = 32768Hz @ XTAL & HSE = 0		**58**		uA
I_{WAT}	Watch 模式工作电流	在掉电模式下 WDT = 64KHz @ ILRCO		**2.5**		uA
I_{MON1}	Monitor 模式工作电流	在掉电模式下 BOD0 使能		**10**		uA
I_{RTC1}	**RTC** 模式工作电流	**RTC 工作在掉电模式**		**4.7**		**uA**
I_{PD1}	掉电模式工作电流			**0.1**	**5**	uA
BOD0 特性						
V_{BOD0}	BOD0 检测电压	T_A = -40℃ to +85℃	4.0(1)	4.2	4.4(1)	V
工作环境						
V_{PSR}	上电速度	T_A = -40℃ to +85℃	0.05			V/ms
V_{OP1}	工作速度在 0–25MHz	T_A = -40℃ to +85℃	4.5		5.5	V
V_{OP2}	工作速度在 0-12MHz	T_A = -40℃ to +85℃	4.2		5.5	V

(1) 数据基于特性结果，非产品测试所得。

对 **MA86L508:** VDD = 3.3V±10%, VSS = 0V, T_A = 25 ℃并且每个机器周期执行 NOP 指令，除非另有说明

标号	参数	测试环境	极限			单位
			最小	典型	最大	
输入/输出特性						
V_{IH1}	输入高电压(所有 I/O 端口)	除 RST, P4.0, P4.1 之外	2.0			V
V_{IH2}	输入高电压 (P4.0, P4.1)		2.4			V
V_{IH3}	输入高电压(RST/P3.6)		2.7			V
V_{IL1}	输入低电压(所有 I/O 端口)	除 RST, P4.0, P4.1 之外			0.8	V
V_{IL2}	输入低电压(P4.0, P4.1)				0.8	V
V_{IL3}	输入低电压(RST/P3.6)				0.8	V
I_{IH}	输入高时漏电流(所有 I/O 端口)	V_{PIN} = VDD		0	10	uA
I_{IL1}	逻辑 0 输入电流 (P3 在准双向模式或输入口的片内上拉电阻)	V_{PIN} = 0.4V		10	30	uA
I_{IL2}	逻辑 0 输入电流 (所有仅输入或开漏口)	V_{PIN} = 0.4V		0	10	uA
I_{H2L}	逻辑 1 到 0 输入转换电流(P3 在准双向模式或输入口的片内上拉电阻)	V_{PIN} =1.8V		120	250	uA
I_{OH1}	输出高时电流(P3 在准双向模式)	V_{PIN} =2.4V	40	80		uA
I_{OH2}	输出高时电流(所有推挽输出口)	V_{PIN} =2.4V	8			mA
I_{OL1}	输出低时电流(所有 I/O 端口)	V_{PIN} =0.4V	8			mA
R_{RST}	内部复位下拉电阻			93		Kohm
电源功耗						
I_{OP1}	正常模式工作电流	SYSCLK = 24MHz @ IHRCO		**11**		mA
I_{OP2}		SYSCLK = 12MHz @ IHRCO		**6.8**		mA
I_{OP3}		SYSCLK = 6MHz @ IHRCO & HSE = 0		**4.8**		mA
I_{OP4}		SYSCLK = 3MHz @ IHRCO & HSE = 0		**2.9**		mA
I_{OP5}		SYSCLK = 24MHz @ XTAL		**11**		mA
I_{OP6}		SYSCLK = 12MHz @ XTAL		**6.7**		mA
I_{OP7}		SYSCLK = 6MHz @ XTAL & HSE = 0		**4.3**		mA
I_{OP8}		SYSCLK = 2MHz @ XTAL & HSE = 0		**1.6**		mA
I_{OPS1}	低速模式工作电流	SYSCLK = 24MHz/128 @ IHRCO **& HSE = 0**		**1.1**		mA

I_{IDLE1}	空闲模式工作电流	SYSCLK = 24MHz @ IHRCO		**4**		mA
I_{IDLE2}		SYSCLK = 24MHz @ XTAL		**3.8**		mA
I_{IDLE3}		SYSCLK = 24MHz/128 @ IHRCO		**1**		mA
I_{IDLE4}		SYSCLK = 24MHz/128 @ XTAL		**0.75**		mA
I_{IDLE5}		SYSCLK = 64KHz @ ILRCO		**22**		uA
I_{IDLE6}		**SYSCLK = 64KHz/128 @ ILRCO**		**13**		**uA**
I_{SUB1}	副频模式工作电流	SYSCLK = 64KHz @ ILRCO& HSE = 0		**60**		uA
I_{SUB2}		SYSCLK = 64KHz/128 @ ILRCO & HSE = 0		**13**		uA
I_{SUB3}		SYSCLK = 32768Hz @ XTAL & HSE = 0		**60**		uA
I_{WAT}	Watch 模式工作电流	在掉电模式下 WDT = 64KHz @ ILRCO		**2.5**		uA
I_{MON1}	Monitor 模式工作电流	在掉电模式下 BOD0 使能		**10**		uA
I_{RTC1}	**RTC** 模式工作电流	**RTC 工作在掉电模式**		**1.8**		**uA**
I_{PD1}	掉电模式工作电流			**0.1**	**5**	uA
BOD0 特性						
V_{BOD0}	BOD0 检测电压	T_A = -40℃ to +85℃	2.45[(1)]	2.6	2.75[(1)]	V
工作环境						
V_{PSR}	上电速度	T_A = -40℃ to +85℃	0.05			V/ms
V_{OP1}	工作速度在 0–25MHz	T_A = -40℃ to +85℃	2.7		3.6	V
V_{OP2}	工作速度在 0-12MHz	T_A = -40℃ to +85℃	2.4		3.6	V

(1) 数据基于特性结果，非产品测试所得。

27.3. 外部时钟特性

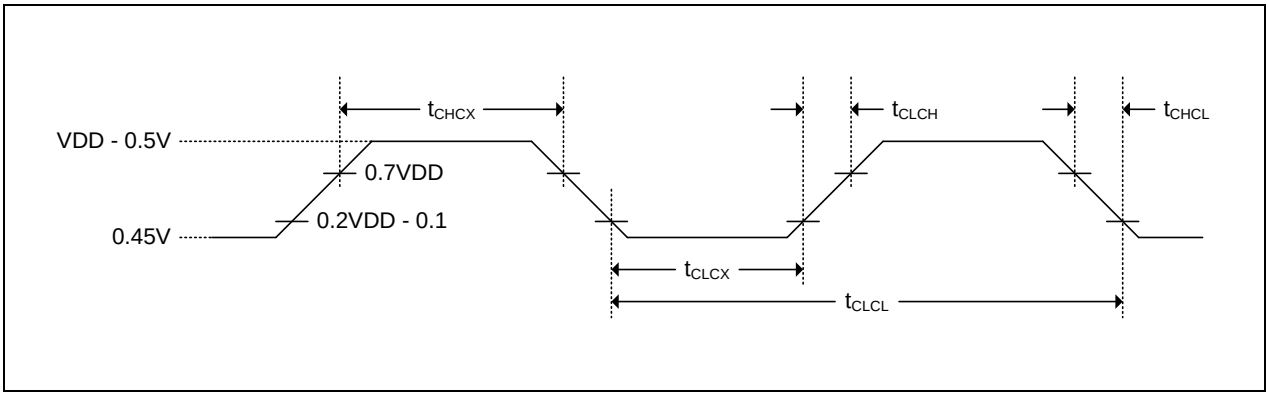
对 **MA86E508:** VDD = 4.5V ~ 5.5V, VSS = 0V, T_A = -40℃ to +85℃，除非另有说明

标号	参数	振荡				单位
		晶振模式		外部时钟输入模式		
		最小	最大	最小	最大	
$1/t_{CLCL}$	振荡频率	2	25	0	25	MHz
$1/t_{CLCL}$	振荡频率 (VDD = 4.2V ~ 5.5V)	2	12	0	12	MHz
t_{CLCL}	时钟周期	40		40		ns
t_{CHCX}	高时间	0.4T	0.6T	0.4T	0.6T	t_{CLCL}
t_{CLCX}	低时间	0.4T	0.6T	0.4T	0.6T	t_{CLCL}
t_{CLCH}	上升时间		5		5	ns
t_{CHCL}	下降时间		5		5	ns

对 **MA86L508:** VDD = 2.7V ~ 3.6V, VSS = 0V, T_A = -40℃ to +85℃，除非另有说明

标号	参数	振荡				单位
		晶振模式		外部时钟输入模式		
		最小	最大	最小	最大	
$1/t_{CLCL}$	振荡频率	2	25	0	25	MHz
$1/t_{CLCL}$	振荡频率 (VDD = 2.4V ~ 3.6V)	2	12	0	12	MHz
t_{CLCL}	时钟周期	40		40		ns
t_{CHCX}	高时间	0.4T	0.6T	0.4T	0.6T	t_{CLCL}
t_{CLCX}	低时间	0.4T	0.6T	0.4T	0.6T	t_{CLCL}
t_{CLCH}	上升时间		5		5	ns
t_{CHCL}	下降时间		5		5	ns

图 27–1. 外部时钟驱动波形



27.4. IHRCO 特性

对 MA86E508:

参数	测试环境	极限			单位
		最小	典型	最大	
电源电压		4.5		5.5	V
IHRCO 频率	TA = +25℃, AFS = 0		24		MHz
	TA = +25℃, AFS = 1		22.118		MHz
IHRCO 频率误差	TA = +25℃	-1.0		+1.0	%
(工厂校对后)	TA = -40℃ to +85℃	**-2.5**[(1)]		**+2.5**[(1)]	%
IHRCO 启动时间	TA = -40℃ to +85℃		**1**[(1)]	32[(1)]	us
IHRCO 功耗	TA = +25℃, VDD=5.0V		770		uA

[(1)] 数据基于特性结果，非产品测试所得。

对 MA86L508:

参数	测试环境	极限			单位
		最小	典型	最大	
电源电压		2.7		3.6	V
IHRCO 频率	TA = +25℃, AFS = 0		24		MHz
	TA = +25℃, AFS = 1		22.118		MHz
IHRCO 频率误差	TA = +25℃	-1.0		+1.0	%
(工厂校对后)	TA = -40℃ to +85℃	**-2.5**[(1)]		**+2.5**[(1)]	%
IHRCO 启动时间	TA = -40℃ to +85℃		**1**[(1)]	32[(1)]	us
IHRCO 功耗	TA = +25℃, VDD=**3.3V**		770		uA

[((1)] 数据基于特性结果，非产品测试所得。

27.5. ILRCO 特性

对 MA86E508:

参数	测试环境	极限			单位
		最小	典型	最大	
电源电压		4.2		5.5	V
ILRCO 频率	TA = +25℃		64		KHz
ILRCO 频率误差	TA = +25℃	-30[(1)]		+30[(1)]	%
	TA = -40℃ to +85℃	-50[(1)]		+50[(1)]	%
ILRCO 功耗	**TA = +25℃, VDD=5.0V**		**2**		**uA**

[(1)] 数据基于特性结果，非产品测试所得。

对 MA86L508:

参数	测试环境	极限			单位
		最小	典型	最大	
电源电压		2.4		3.6	V
ILRCO 频率	TA = +25℃		64		KHz
ILRCO 频率误差	TA = +25℃	-30(1)		+30(1)	%
	TA = -40℃ to +85℃	-50(1)		+50(1)	%
IHRCO 功耗	**TA = +25℃, VDD=3.3V**		**2**		**uA**

(1) 数据基于特性结果，非产品测试所得。

27.6. Flash 特性

对 MA86E508:

参数	测试环境	极限			单位
		最小	典型	最大	
电源电压	TA = -40℃ to +85℃	4.2		5.5	V
Flash 写 (擦除/编程)电压	TA = -40℃ to +85℃	4.5		5.5	V
Flash 擦除/编程次数	TA = -40℃ to +85℃	100			次
Flash 数据保存	TA = +25℃	100			年

对 MA86L508:

参数	测试环境	极限			单位
		最小	典型	最大	
电源电压	TA = -40℃ to +85℃	2.4		3.6	V
Flash 写 (擦除/编程)电压	TA = -40℃ to +85℃	2.7		3.6	V
Flash 擦除/编程次数	TA = -40℃ to +85℃	100			次
Flash 数据保存	TA = +25℃	100			年

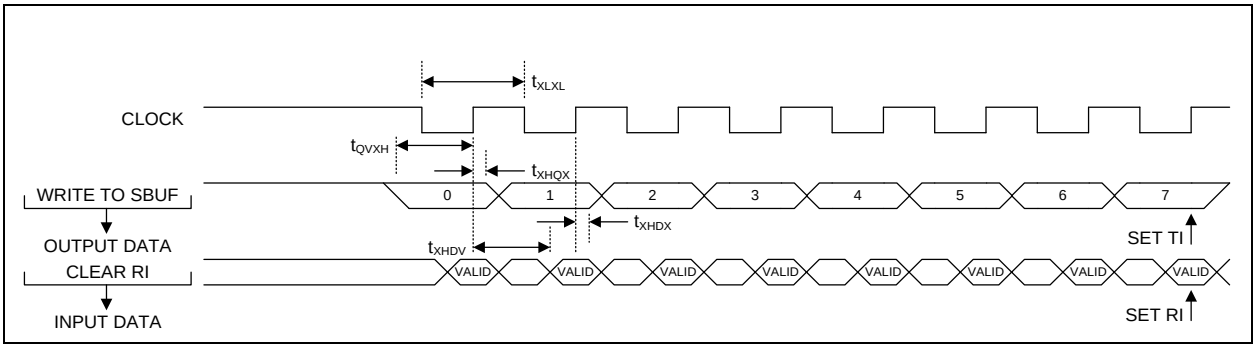
27.7. 串口时序特性

对 MA86E/L508 E-型: VDD = 5.0V±10%, VSS = 0V, T_A = -40℃ to +85℃, 除非另有说明

对 MA86E/L508 L-型: VDD = 3.3V±10%, VSS = 0V, T_A = -40℃ to +85℃, 除非另有说明

符号	参数	URM0X3 = 0		URM0X3 = 1		单位
		最小	最大	最小	最大	
t_{XLXL}	串行口时钟周期时间	12T		4T		T_{SYSCLK}
t_{QVXH}	设定输出数据到时钟上升沿	10T-20		T-20		ns
t_{XHQX}	时钟上升沿后输出数据保持时间	T-10		T-10		ns
t_{XHDX}	时钟上升沿后输入数据保持时间	0		0		ns
t_{XHDV}	时钟上升沿到输入数据有效		10T-20		4T-20	ns

图 27–2. 移位寄存器模式时序波形图



27.8. ADC 特性

对 **MA86E508:**VDD = 5.0V, VSS = 0V, T_A = -40℃ to +85℃，除非另有说明

参数	测试环境	极限			单位
		最小	典型	最大	
电源范围					
电源电压		4.2		5.5	V
DC 精确度					
分辨率			8		bits
整体非线性	VDD= 5.0V (200 ksps)		±0.5	±1	LSB
	VDD= 4.2V~5.5V (200 ksps)		±1	±1.5	LSB
	VDD= 4.2V~5.5V (25 ksps)		±1	±1.5	LSB
差动非线性	VDD= 4.2V ~ 5.5V		±0.5	±1	LSB
补偿误差	VDD= 4.2V ~ 5.5V			±1	LSB
转换率					
SAR 转换时钟				12	MHz
在 SAR 时钟的转换时间			60		clocks
吞吐率				200	ksps
模拟输入					
ADC 输入电压范围		0		VDD	V
输入电容			1		pF
功耗					
电源电流			1		mA

对 **MA86L508:** VDD = 3.3V, VSS = 0V, T_A = -40℃ to +85℃，除非另有说明

参数	测试环境	极限			单位
		最小	典型	最大	
电源范围					
电源电压		2.4		3.6	V
DC 精确度					
分辨率			8		bits
整体非线性	VDD= 3.0V~3.6V (200 ksps)		±1.5	±2	LSB
	VDD= 2.7V~3.6V (200 ksps)		±2.5	±3	LSB
	VDD= 2.4V~3.6V (200 ksps)		±8	±10	LSB
	VDD= 3.0V~3.6V (25 ksps)		±1	±1.5	LSB
	VDD= 2.7V~3.6V (25 ksps)		±1	±1.5	LSB
	VDD= 2.4V~3.6V (25 ksps)		±3	±4	LSB
差动非线性	VDD= 3.0V~3.6V (200 ksps)		±0.5	±1	LSB
	VDD= 2.7V~3.6V (200 ksps)		±1.5	±2	LSB
	VDD= 2.4V~3.6V (200 ksps)		±8	±10	LSB
	VDD= 3.0V~3.6V (25 ksps)		±0.5	±1	LSB
	VDD= 2.7V~3.6V (25 ksps)		±0.5	±1	LSB
	VDD= 2.4V~3.6V (25 ksps)		±3	±4	LSB
补偿误差	VDD= 2.4V ~ 3.6V			±1	LSB
转换率					
SAR 转换时钟				12	MHz
在 SAR 时钟的转换时间			60		clocks
吞吐率				200	ksps
模拟输入					
ADC 输入电压范围		0		VDD	V
输入电容			1		pF
功耗					
电源电流			0.8		mA

28. 指令集

表 28–1. 指令集

助记符	描述	字节	周期
数据传送			
MOV A,Rn	寄存器Rn中的内容送到累加器中	1	1
MOV A,direct	直接地址单元中的内容送到累加器中	2	2
MOV A,@Ri	工作寄存器Ri指向的地址单元中的内容送到累加器中	1	2
MOV A,#data	立即数送到累加器中	2	2
MOV Rn,A	累加器中内容送到寄存器Rn中	1	2
MOV Rn,direct	直接寻址单元中的内容送到寄存器Rn中	2	4
MOV Rn,#data	立即数直接送到寄存器Rn中	2	2
MOV direct,A	累加器送到直接地址单元	2	3
MOV direct,Rn	寄存器Rn中的内容送到直接地址单元	2	3
MOV direct,direct	直接地址单元中的内容送到另一个直接地址单元	3	4
MOV direct,@Ri	工作寄存器Ri指向的地址单元中的内容送到直接地址单元	2	4
MOV direct,#data	立即数送到直接地址单元	3	3
MOV @Ri,A	累加器送到以工作寄存器Ri指向的地址单元中	1	3
MOV @Ri,direct	直接地址单元中内容送到以工作寄存器Ri指向的地址单元中	2	3
MOV @Ri,#data	立即数送到以工作寄存器Ri指向的地址单元中	2	3
MOV DPTR,#data16	16位常数的高8位送到DPH，低8位送到DPL	3	3
MOVC A,@A+DPTR	以DPTR为基地址变址寻址单元中的内容送到累加器中	1	4
MOVC A,@A+PC	以PC为基地址变址寻址单元中的内容送到累加器中	1	4
MOVX A,@Ri	内置 XRM（8 位地址）的数据送入累加器中	1	不支持
MOVX A,@DPTR	内置 XRM（16 位地址）的数据送入累加器中	1	不支持
MOVX @Ri,A	累加器的数据送入内置 XRM（8 位地址）中	1	不支持
MOVX @DPTR,A	累加器的数据送入内置 XRM（16 位地址）中	1	不支持
MOVX A,@Ri	外部 XRM（8 位地址）的数据送入累加器中	1	不支持
MOVX A,@DPTR	外部 XRM（16 位地址）的数据送入累加器中	1	不支持
MOVX @Ri,A	累加器的数据送入外部 XRM（8 位地址）中	1	不支持
MOVX @DPTR,A	累加器的数据送入外部 XRM（16 位地址）中	1	不支持
PUSH direct	直接地址单元中的数据压入堆栈中	2	4
POP direct	出栈数据送到直接地址单元中	2	3

XCH A,Rn	累加器与寄存器Rn中的内容互换	1	3
XCH A,direct	累加器与直接地址单元中的内容互换	2	4
XCH A,@Ri	累加器与工作寄存器Ri指向的地址单元中内容互换	1	4
XCHD A,@Ri	累加器与工作寄存器Ri指向的地址单元中内容低半字节互换	1	4
算术运算 ***S***			
ADD A,Rn	Acc ← Acc+Rn	1	2
ADD A,direct	Acc ← Acc+direct	2	3
ADD A,@Ri	Acc ← Acc+Ri	1	3
ADD A,#data	Acc ← Acc+data	2	2
ADDC A,Rn	Acc ← Acc+Rn+C	1	2
ADDC A,direct	Acc ← Acc+direct+C	2	3
ADDC A,@Ri	Acc ← Acc+Ri+C	1	3
ADDC A,#data	Acc ← Acc+data+C	2	2
SUBB A,Rn	Acc ← Acc-Rn-C	1	2
SUBB A,direct	Acc ← Acc-direct-C	2	3
SUBB A,@Ri	Acc ← Acc-Ri-C	1	3
SUBB A,#data	Acc ← Acc-data-C	2	2
INC A	Acc ← Acc +1	1	2
INC Rn	Rn ← Rn +1	1	3
INC direct	direct ← direct +1	2	4
INC @Ri	Ri ← Ri +1	1	4
DEC A	DPTR ← DPTR +1	1	2
DEC Rn	Acc ← Acc - 1	1	3
DEC direct	Rn ← Rn -1	2	4
DEC @Ri	direct ← direct -1	1	4
INC DPTR	Ri ← Ri -1	1	1
MUL AB	两数相乘，结果高八位存入 B,低八位存入 A	1	4
DIV AB	Acc 除以 B，商存入 Acc,余数存入 B	1	5
DA A	Acc 作十进制调整	1	4
逻辑运算			
ANL A,Rn	累加器和寄存器Rn中的内容相“与”	1	2
ANL A,direct	累加器和直接地址单元中的内容相“与”	2	3
ANL A,@Ri	累加器和工作寄存器Ri指向的地址单元中的内容相“与”	1	3
ANL A,#data	累加器和立即数相“与”	2	2
ANL direct,A	直接地址单元中的内容和累加器相“与”	2	4
ANL direct,#data	直接地址单元中的内容和立即数相“与”	3	4

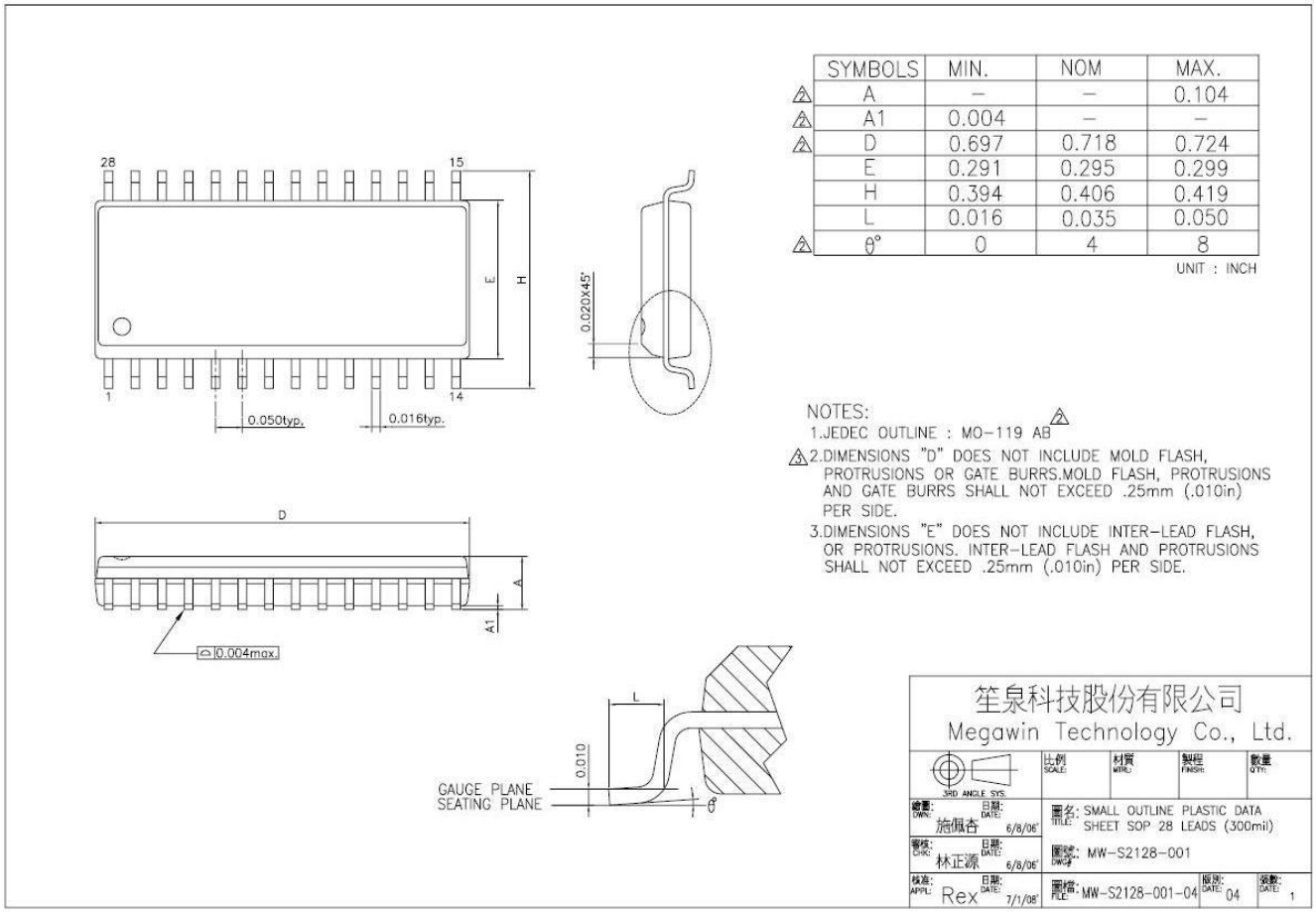
ORL A,Rn	累加器和寄存器Rn中的内容相“或”	1	2
ORL A,direct	累加器和直接地址单元中的内容相“或”	2	3
ORL A,@Ri	累加器和工作寄存器Ri指向的地址单元中的内容相“或”	1	3
ORL A,#data	累加器和立即数相“或”	2	2
ORL direct,A	直接地址单元中的内容和累加器相“或”	2	4
ORL direct,#data	直接地址单元中的内容和立即数相“或”	3	4
XRL A,Rn	累加器和寄存器Rn中的内容相“异或”	1	2
XRL A,direct	累加器和直接地址单元中的内容相“异或”	2	3
XRL A,@Ri	累加器和工作寄存器Ri指向的地址单元中的内容相“异或”	1	3
XRL A,#data	累加器和立即数相“异或”	2	2
XRL direct,A	直接地址单元中的内容和累加器相“异或”	2	4
XRL direct,#data	直接地址单元中的内容和立即数相“异或”	3	4
CLR A	累加器内容清“0”	1	1
CPL A	累加器按位取反	1	2
RL A	累加器循环左移一位	1	1
RLC A	累加器连同进位位CY循环左移一位	1	1
RR A	累加器循环右移一位	1	1
RRC A	累加器连同进位位CY循环右移一位	1	1
SWAP A	累加器高低半字节互换	1	1
位逻辑运算			
CLR C	清“0”进位位	1	1
CLR bit	清“0”直接地址位	2	4
SETB C	置“1”进位位	1	1
SETB bit	置“1”直接地址位	2	4
CPL C	进位位求反	1	1
CPL bit	直接地址位求反	2	4
ANL C,bit	进位位和直接地址位相“与”	2	3
ANL C,/bit	进位位和直接地址位的反码相“与”	2	3
ORL C,bit	进位位和直接地址位相“或”	2	3
ORL C,/bit	进位位和直接地址位的反码相“或”	2	3
MOV C,bit	直接地址位数据送入进位位	2	3
MOV bit,C	进位位数据送入直接地址位	2	4
位逻辑跳转			
JC rel	进位位为“1”则转移	2	3

JNC rel	进位位为“0”则转移	2	3
JB bit,rel	直接地址位为“1”则转移	3	4
JNB bit,rel	直接地址位为“0”则转移	3	4
JBC bit,rel	直接地址位为“1”则转移，且清“0”该位	3	5
程序跳转			
ACALL addr11	绝对短调用子程序，2K字节（页内）空间限制	2	6
LCALL addr16	绝对长调用子程序，64K字节空间限制	3	6
RET	子程序返回	1	4
RETI	中断子程序返回	1	4
AJMP addr11	绝对短转移，2K字节（页内）空间限制	2	3
LJMP addr16	绝对长转移，64K字节空间限制	3	4
SJMP rel	相对转移	2	3
JMP @A+DPTR	转移到DPTR加ACC所指间接地址	1	3
JZ rel	累加器为“0”则转移	2	3
JNZ rel	累加器不为“0”则转移	2	3
CJNE A,direct,rel	累加器中的内容不等于直接地址单元的内容，则转移到偏移量所指向的地址，否则程序往下执行	3	5
CJNE A,#data,rel	累加器中的内容不等于立即数，则转移到偏移量所指向的地址，否则程序往下执行	3	4
CJNE Rn,#data,rel	寄存器Rn中的内容不等于立即数，则转移到偏移量所指向的地址，否则程序往下执行	3	4
CJNE @Ri,#data,rel	工作寄存器Ri指向的地址单元中的内容不等于立即数，则转移到偏移量所指向的地址，否则程序往下执行	3	5
DJNZ Rn,rel	寄存器Rn中的内容减1，如不等于0，则转移到偏移量所指向的地址，否则程序往下执行	2	4
DJNZ direct,rel	直接地址单元中的内容减1，如不等于0，则转移到偏移量所指向的地址，否则程序往下执行	3	5
NOP	空操作指令	1	1

29. 封装尺寸

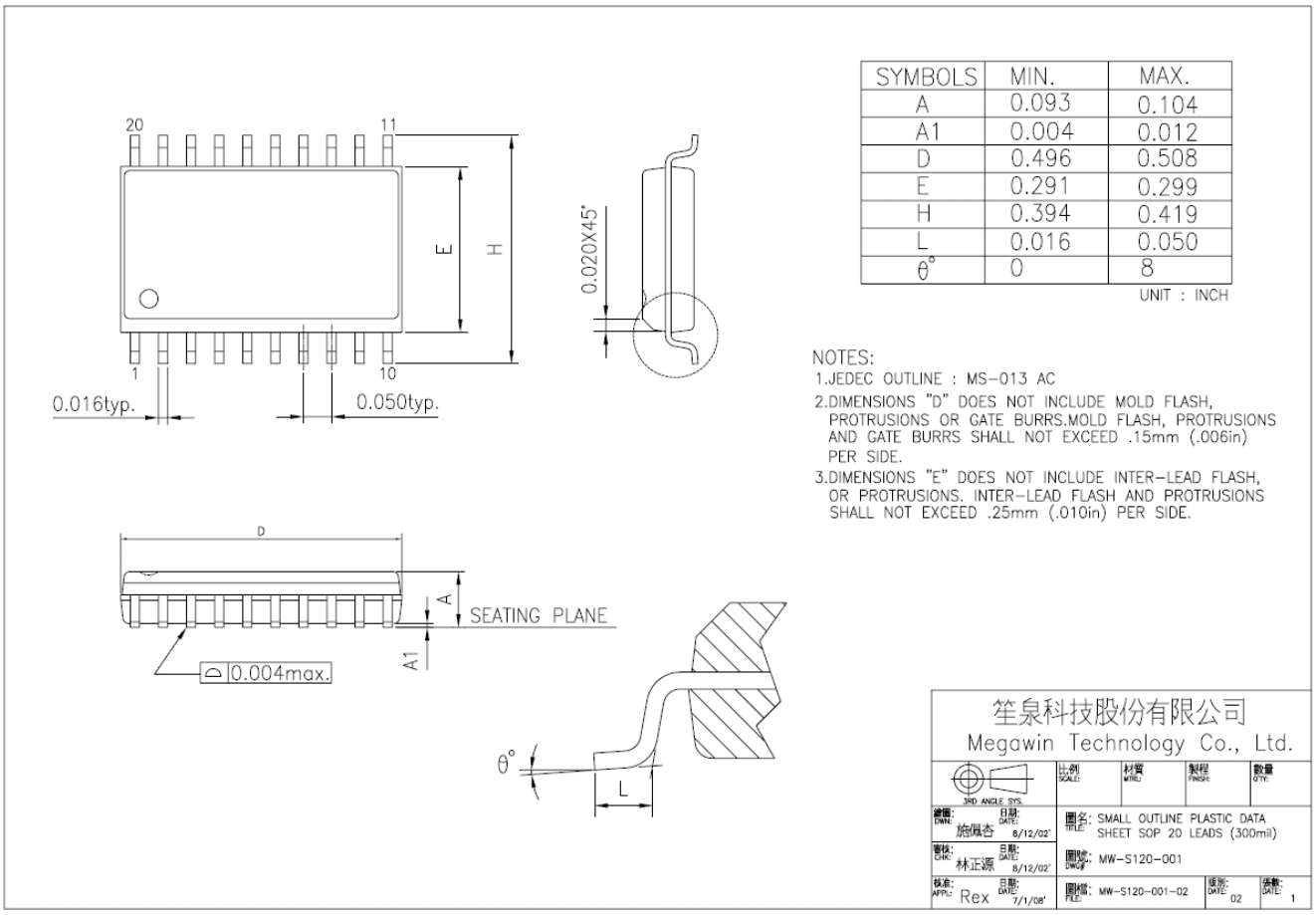
29.1. SOP-28

图 29–1. SOP-28



29.2. SOP-20

图 29–2. SOP-20



SYMBOLS	MIN.	MAX.
A	0.093	0.104
A1	0.004	0.012
D	0.496	0.508
E	0.291	0.299
H	0.394	0.419
L	0.016	0.050
θ°	0	8

UNIT : INCH

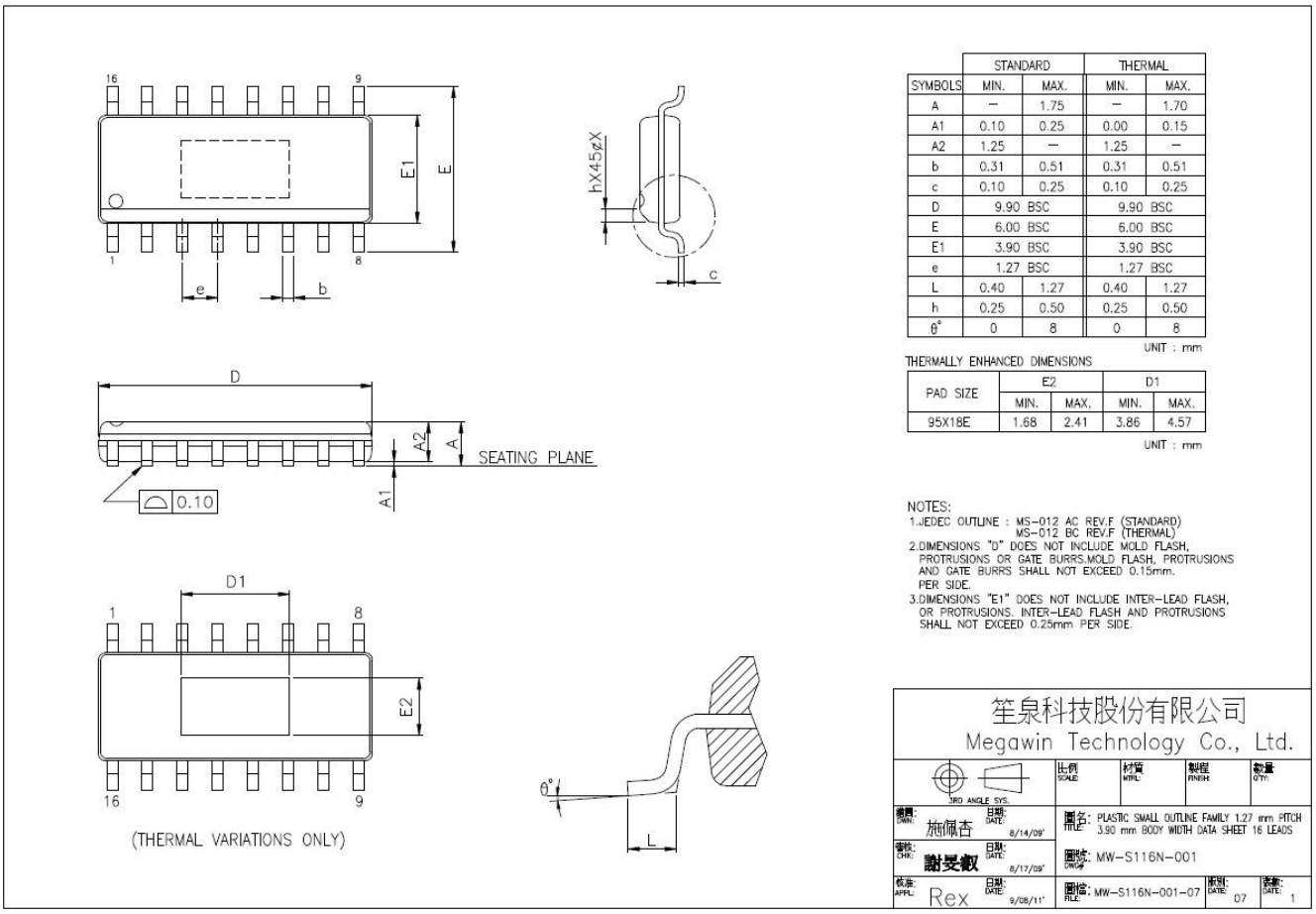
NOTES:
1.JEDEC OUTLINE : MS-013 AC
2.DIMENSIONS "D" DOES NOT INCLUDE MOLD FLASH, PROTRUSIONS OR GATE BURRS.MOLD FLASH, PROTRUSIONS AND GATE BURRS SHALL NOT EXCEED .15mm (.006in) PER SIDE.
3.DIMENSIONS "E" DOES NOT INCLUDE INTER-LEAD FLASH, OR PROTRUSIONS. INTER-LEAD FLASH AND PROTRUSIONS SHALL NOT EXCEED .25mm (.010in) PER SIDE.

笙泉科技股份有限公司
Megawin Technology Co., Ltd.

3RD ANGLE SYS.	比例 SCALE:	材質 MTRL:	製程 FINISH:	數量 QTY:
繪圖: DWN: 施佩杏 日期: DATE: 6/12/02'	圖名: TITLE: SMALL OUTLINE PLASTIC DATA SHEET SOP 20 LEADS (300mil)			
審核: CHK: 林正源 日期: DATE: 6/12/02'	圖號: DWG#: MW-S120-001			
核准: APPL: Rex 日期: DATE: 7/1/08'	圖檔: FILE: MW-S120-001-02	版別: DATE: 02	張數: DATE: 1	

29.3. SOP-16

图 29–3. SOP-16



SYMBOLS	STANDARD MIN.	STANDARD MAX.	THERMAL MIN.	THERMAL MAX.
A	—	1.75	—	1.70
A1	0.10	0.25	0.00	0.15
A2	1.25	—	1.25	—
b	0.31	0.51	0.31	0.51
c	0.10	0.25	0.10	0.25
D	9.90 BSC		9.90 BSC	
E	6.00 BSC		6.00 BSC	
E1	3.90 BSC		3.90 BSC	
e	1.27 BSC		1.27 BSC	
L	0.40	1.27	0.40	1.27
h	0.25	0.50	0.25	0.50
θ°	0	8	0	8

UNIT : mm

THERMALLY ENHANCED DIMENSIONS

PAD SIZE	E2 MIN.	E2 MAX.	D1 MIN.	D1 MAX.
95X18E	1.68	2.41	3.86	4.57

UNIT : mm

NOTES:
1.JEDEC OUTLINE : MS-012 AC REV.F (STANDARD)
MS-012 BC REV.F (THERMAL)
2.DIMENSIONS "D" DOES NOT INCLUDE MOLD FLASH, PROTRUSIONS OR GATE BURRS.MOLD FLASH, PROTRUSIONS AND GATE BURRS SHALL NOT EXCEED 0.15mm. PER SIDE.
3.DIMENSIONS "E1" DOES NOT INCLUDE INTER-LEAD FLASH, OR PROTRUSIONS. INTER-LEAD FLASH AND PROTRUSIONS SHALL NOT EXCEED 0.25mm PER SIDE.

笙泉科技股份有限公司
Megawin Technology Co., Ltd.

3RD ANGLE SYS.	比例 SCALE:	材質 MTRL:	製程 FINISH:	數量 QTY:
繪圖: 施佩杏	日期: 8/14/09'	圖名: PLASTIC SMALL OUTLINE FAMILY 1.27 mm PITCH 3.90 mm BODY WIDTH DATA SHEET 16 LEADS		
審核: 謝旻叡	日期: 8/17/09'	圖號: MW-S116N-001		
核准: Rex	日期: 9/08/11'	檔名: MW-S116N-001-07	版別: 07	表數: 1

30. 版本历史

表 30–1. 版本历史

版本	描述	日期
v0.70	1. 初版, v0.70.	2012/07/16
v0.80	1. 在特征章节，修改了 PCA, BOD 及时钟源描述。	2012/08/22
	2. 在特征章节，增加了 P2 在 GPIO 中的描述。	2012/08/22
	3. 在系统方框图，增加了 PCA 中的 PWM 通道数。	2012/08/22
	4. 在系统时钟章节，完成了 XTOR 功能及在 9.2 节增加了 DON0.HSE 描述，时钟寄存器。并且修改了 IHRCO 执行描述，请参考 IHRCO 特性。	2012/08/22
	5. 修改了图 9-1 中 P4.0 上的 IHRCO 输出。	2012/08/22
	6. 在 RTCTM 寄存器描述中 P6.0 修改为 P4.0.	2012/08/22
	7. 副频模式最低速度从 250Hz 修改为 500Hz。	2012/08/22
	8. 修改了 WDT 示意图。	2012/08/29
	9. 完成了定时器 0 时钟预设值的 T0XL 功能。	2012/08/29
	10. 修改了定时器 0/1 模式 0/1 的描述及 T0XL 的定时器 0 的时钟输出功能。	2012/08/29
	11. 在章节 16.1.6 中，修改了模式 0 的 TMOD 寄存器， PWM 产生器。	2012/08/29
	12.增加了 UART 模式 0 的时钟极性选择描述。	2012/08/29
	13. 修改了图 17-4 及 17-5 中 UART 模式 0 时钟极性选择。	2012/08/29
	14. 增加了 PCA 模块中 CPS2 来选择 SYSCLK/1。	2012/08/29
	15. 增加了 PAOE 描述并且修改了 ISP/IAP 描述。	2012/08/29
	16. 增加了在 KBI 中端口引脚输入模式定义描述。	2012/08/29
	17. 增加了串行接口侦测及蜂鸣器功能描述。	2012/08/29
	18. 更新了图 23-1	2012/08/29
	19. 修改了在 15.2 节中的系统标志中断描述。	2012/08/29
	20. 增加了每个功能的示例代码。	2012/08/29
A1	1. 增加 SOP-16 封装描述。	2013/01/09
A1.1	修改 Flash 写/擦次数说明	2013/03/27
A1.2	增加说明	2013/05/21
31.2	修正 IE 寄存器地址错误 E8h 为正确的 A8h	2013/10/09

免责声明

在此，笙泉（Megawin）代表“*Megawin Technology Co., Ltd.*”

生命支援

此产品并不是为医疗、救生或维持生命而设计的，并且当设备系统出现故障时，并不能合理地预示是否会对人身造成伤害。因此，当客户使用或出售用于上述应用的产品时，需要客户自己承担这样做的风险，笙泉公司并不会对不当地使用或出售我公司的产品而造成的任何损害进行赔偿。

更改权

笙泉保留产品的如下更改权，其中包括电路、标准单元、与/或软件 - 在此为提高设计的与/或性能的描述或内容。当产品在大批量生产时，有关变动将通过工程变更通知（ECN）进行通知。