MEGAWIN
MAKE YOU WIN

8051 内核-单片机

MA805-64/MA806-64
说明书

版本: **A1.01**

2015/09 version A1.01

功能

- 1-T 80C51 CPU
- MA805-64 有 64K 字节闪存（Flash ROM）
  - ISP 存储器空间 1.5KB
  - IAP 大小为 2.5KB（默认）。AP 代码没有用到的 AP 闪存区可以设置作 IAP 用途。
  - AP 及 IAP 存取区域

型号	AP 大小	IAP 大小
MA805-64	60KB（最大）	2.5KB（默认）

- 片上 256 字节随机存取储存器和 1024 字节片上扩展存储器(XRAM)
- 双数据指针
- 为适应慢速 SRAM 或外围设备而可改变时长的 MOVX 指令
- 三个 16 位定时/计数器：定时器 0、定时器 1 及定时器 2
  - 三个定时器输出口（T0CKO 对应 P34、T1CKO 对应 P35、T2CKO 对应 P10）
  - T0/T1/T2 时钟可以选择 X12 模式
- 有 6 个 PCA （可编程计数器阵列）
  - 捕获模式
  - 16 位软件定时器模式
  - 高速输出模式
  - PWM（脉宽调节器）模式
- 增强型 UART (S0)
  - 帧错误侦测
  - 自动地址识别
  - 速度改进机制(X2/X4 模式)
- 副 UART (S1)
  - 专用的波特率发生器
  - S1 与 S0 复用波特率发生器
- 中断控制器
  - 14 个中断源，4 个优先级中断能力
  - 4 个外部中断输入 nINT0、 nINT1、 nINT2 及 nINT3
  - nINT0/nINT1 触发类型：低电平或下降沿
  - nINT2/nINT3 触发类型：低电平、下降沿、高电平或上升沿
- 8 通道 10 位 ADC
  - 可编程最高转换速率达 200 ksps
  - 高达 8 路外部输入 (Single-ended)
- 主/从 SPI 串行接口
- 端口 2（P2）键盘中断
- 可编程看门狗定时器（WDT）
  - 通过软件或上电一次性使能
  - MCU 掉电模式下的 WDT 工作选项
- LQFP48 封装最大有 45 个普通 I/O 口（GPIO）

- P0, P1, P2, P3, P4, P5 能被配置为准双向口、上拉输出、集电极开漏输出以及高阻输入
- P6.0 和 P6.1 仅仅适合作准双向口模式及复用为 XTAL2 和 XTAL1

- 两种省电模式：空闲模式（idle）和掉电模式（power-down）
  - 空闲模式能被所有中断唤醒
  - 掉电模式能被四个外部中断和键盘中断唤醒
- 掉电检测器（Brown-Out Detector）：对于 MA805 系列是 VDD 低于 4.2V ，对于 MA806 是 VDD 低于 2.4V
  - 掉电检测能选择复位或产生触发 BOD 中断
- 工作电压
  - MA805-64：4.5V~5.5V.
  - MA806-64：2.4V~3.6V. Flash 相关操作不得低于 2.7V（ISP/IAP/ICP）
- 工作温度
  - 工业级（-40℃ 至 +85℃)*
- 工作频率：24MHz(最大)
  - 外部晶体振荡器模式
  - 内部有一个频漂为+/- 1%的高频率 RC 振荡器，内部振荡（出厂默认）频率为 22.1184MHz**
  - 内部高频率 RC 振荡输出脚为 XTAL2/P6.0.
  - 外部时钟输入脚为 XTAL2/P6.0
  - 内部低频率 RC 振荡器支持
- 封装类型
  - LQFP48：MA805-64AD
  - LQFP44：MA805-64AD44
  - PDIP40：MA805-64AE （快速开发验证功能用，不推荐批量生产）

*：样品测试结果。

**：室温内测试结果。

在室溫 @ 25℃下，频漂为+/- 1%,

在工作溫度@ -20 ~ 50℃範圍內，频漂为+/- 2%,

在工作溫度@ -40 ~ 85℃範圍內，频漂为+/- 4%,.

目录

1. 概述

MA805-64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1~7个时钟信号（比标准8051快6~7倍），与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，MA805-64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。

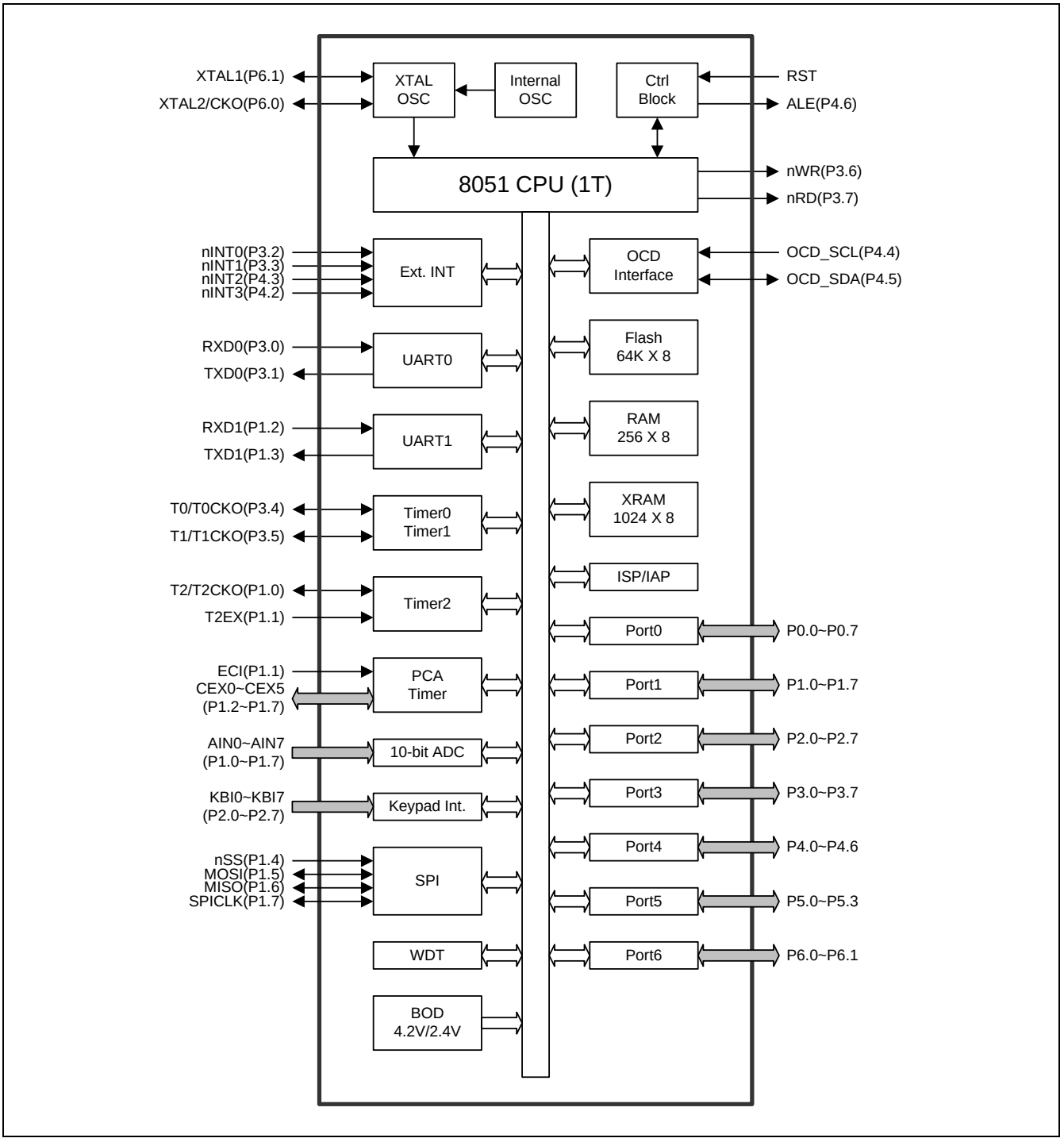
MA805-64拥有64K字节的内置Flash存储器用于保存代码和数据。Flash存储器可以通过并行模式编程，也拥有通过在系统编程(ISP)的能力。同时，也提供在应用编程(IAP)的能力。ISP让使用者无需从产品中取下微控制器就可以下载新的代码；IAP意味着应用程序正在运行时，微控制器能够在Flash中写入非易失数据。这些功能不需要外部提供高电压而仅仅依靠内建的电荷泵提供编程用的高压。

MA805-64除保留了标准80C52的所有功能(例如 256 字节的随机存储器、四个8位I/O口、两个外部中断、一个多源4级中断控制器、三个定时/计数器)外，还增加了许多其它功能，MA805-64有两个额外的I/O口(P4 P5)口、1024字节的片上扩展随机存储器 (XRAM)、两个能选择高/低电平触发的额外外部中断、10位的模/数转换器、一个六通道的PCA、SPI、副UART接口、辅助键盘中断、一个看门狗定时器、一个掉电检测器（Brown-out Detector）、一个片上晶体振荡器（复用P6.0和P6.1）、一个高精度的内部振荡器、一个用于多处理器进行通讯的增强型通用串行通讯口EUART，它有速度改进机制（速度能选择X2/X4模式）。

MA805-64有两种节能模式和8位的系统时钟分频器，以减少耗电量。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的内容被保存，而其它所有功能被终止。最重要的是，在掉电模式下的微控制器可以被外部中断唤醒。并且使用者可以通过8位的系统时钟分频器减慢系统速度以减少耗电量。

另外，MA805-64装备有笙泉特有的为在电路调试（ICE）设计的在片上调试（OCD）接口，在片上调试（OCD）提供片上无资源占用的在系统调试，对于ICE来说，一些功能是必要的而且已经提供了，例如复位、运行、停止、单步、运行到光标和断点。使用者使用ICE不需要准备任何的固件或者适配器接头。所有要做的事情就是准备一个四脚的连接座连接到OCD接口上。这个强有力的功能将使开发变得非常容易。

2. 方框图


XTAL1(P6.1)
XTAL2/CKO(P6.0)
XTAL OSC
Internal OSC
Ctrl Block
RST
ALE(P4.6)
8051 CPU (1T)
nWR(P3.6)
nRD(P3.7)
nINT0(P3.2)
nINT1(P3.3)
nINT2(P4.3)
nINT3(P4.2)
Ext. INT
OCD Interface
OCD_SCL(P4.4)
OCD_SDA(P4.5)
RXD0(P3.0)
TXD0(P3.1)
UART0
Flash 64K X 8
RXD1(P1.2)
TXD1(P1.3)
UART1
RAM 256 X 8
T0/T0CKO(P3.4)
T1/T1CKO(P3.5)
Timer0 Timer1
XRAM 1024 X 8
ISP/IAP
T2/T2CKO(P1.0)
T2EX(P1.1)
Timer2
Port0
P0.0~P0.7
ECI(P1.1)
CEX0~CEX5 (P1.2~P1.7)
PCA Timer
Port1
P1.0~P1.7
AIN0~AIN7 (P1.0~P1.7)
10-bit ADC
Port2
P2.0~P2.7
KBI0~KBI7 (P2.0~P2.7)
Keypad Int.
Port3
P3.0~P3.7
Port4
P4.0~P4.6
nSS(P1.4)
MOSI(P1.5)
MISO(P1.6)
SPICLK(P1.7)
SPI
Port5
P5.0~P5.3
WDT
Port6
P6.0~P6.1
BOD 4.2V/2.4V

3. 特殊功能寄存器（SFRs）

3.1. SFR 映射图

	页	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8	0 F	P5	CH	CCAP0H	CCAP1H	CCAP2H	CCAP3H	CCAP4H	CCAP5H
F0	0 F	B	--	PCAPWM0	PCAPWM1	PCAPWM2	PCAPWM3	PCAPWM4	PCAPWM5
E8	0 F	P4	CL	CCAP0L	CCAP1L	CCAP2L	CCAP3L	CCAP4L	CCAP5L
E0	0 F	ACC	WDTCR	IFD	IFADRH	IFADRL	IFMT	SCMD	ISPCR
D8	0 F	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2	CCAPM3	CCAPM4	CCAPM5
D0	0 F	PSW	--	--	--	--	KBPATN	KBCON	KBMASK
C8	0 F	T2CON P6	T2MOD	RCAP2L	RCAP2H	TL2	TH2	--	--
C0	0 F	XICON	--	--	--	--	ADCON	ADCV	PCON2
B8	0 F	IP0L	SADEN	--	--	--	--	ADCVL	--
B0	0 F	P3	P3M0	P3M1	P4M0	P4M1	P5M0	P5M1	IP0H
A8	0 F	IE	SADDR	--	--	SFRPI	EIE1	EIP1L	EIP1H
A0	0 F	P2	--	AUXR1	--	--	--	AUXR2	--
98	0 1	SCON0 SCON1	SBUF0 SBUF1	SCFG S1BRT	--	--	--	--	--
90	0 F	P1	P1M0	P1M1	P0M0	P0M1	P2M0	P2M1	PCON1
88	0 F	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR0	STRETCH
80	0 F	P0	SP	DPL	DPH	SPSTAT	SPCON	SPDAT	PCON0
		0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F

3.2. SFR 位分配

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	
P0	*Port 0*	*80H*	*P0.7*	*P0.6*	*P0.5*	*P0.4*	*P0.3*	*P0.2*	*P0.1*	*P0.0*	*11111111B*
SP	Stack Pointer	81H									00000111B
DPL	Data Pointer Low	82H									00000000B
DPH	Data Pointer High	83H									00000000B
SPSTAT	SPI Status Register	84H	SPIF	WCOL	--	--	--	--	--	--	00xxxxxxB
SPCON	SPI Control Register	85H	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	00000100B
SPDAT	SPI Data Register	86H									00000000B
PCON0	Power Control 0	87H	SMOD1	SMOD0	GF	POF	GF1	GF0	PD	IDL	00010000B
TCON	*Timer Control*	*88H*	*TF1*	*TR1*	*TF0*	*TR0*	*IE1*	*IT1*	*IE0*	*IT0*	*00000000B*
TMOD	Timer Mode	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	00000000B
TL0	Timer Low 0	8AH									00000000B
TL1	Timer Low 1	8BH									00000000B
TH0	Timer High 0	8CH									00000000B
TH1	Timer High 1	8DH									00000000B
AUXR0	Auxiliary Register 0	8EH	P60OC1	P60OC0	P60FD	P34FD	MOVXFD	ADRJ	EXTRAM	--	0000000xB
STRETCH	MOVX Timing Stretch	8FH	EMAI1	--	ALES1	ALES0	RWSH	RWS2	RWS1	RWS0	0x000000B
P1	*Port 1*	*90H*	*P1.7*	*P1.6*	*P1.5*	*P1.4*	*P1.3*	*P1.2*	*P1.1*	*P1.0*	*11111111B*
P1M0	P1 Mode Register 0	91H	P1M0.7	P1M0.6	P1M0.5	P1M0.4	P1M0.3	P1M0.2	P1M0.1	P1M0.0	00000000B
P1M1	P1 Mode Register 1	92H	P1M1.7	P1M1.6	P1M1.5	P1M1.4	P1M1.3	P1M1.2	P1M1.1	P1M1.0	00000000B
P0M0	P0 Mode Register 0	93H	P0M0.7	P0M0.6	P0M0.5	P0M0.4	P0M0.3	P0M0.2	P0M0.1	P0M0.0	00000000B
P0M1	P0 Mode Register 1	94H	P0M1.7	P0M1.6	P0M1.5	P0M1.4	P0M1.3	P0M1.2	P0M1.1	P0M1.0	00000000B
P2M0	P2 Mode Register 0	95H	P2M0.7	P2M0.6	P2M0.5	P2M0.4	P2M0.3	P2M0.2	P2M0.1	P2M0.0	00000000B
P2M1	P2 Mode Register 1	96H	P2M1.7	P2M1.6	P2M1.5	P2M1.4	P2M1.3	P2M1.2	P2M1.1	P2M1.0	00000000B
PCON1	Power Control 1	97H	SWRF	EXRF	BORF	IARF	--	--	--	BOF	0000xxx0B
SCON0	*Serial 0 Control*	*98H*	*SM00 /FE*	*SM10*	*SM20*	*REN0*	*TB80*	*RB80*	*TI0*	*RI0*	*00000000B*
SCON1	*Serial 1 Control*	*98H*	*SM01*	*SM11*	*SM21*	*REN1*	*TB81*	*RB81*	*TI1*	*RI1*	*00000000B*
SBUF0	Serial 0 Buffer	99H									xxxxxxxxB
SBUF1	Serial 1 Buffer	99H									xxxxxxxxB
SCFG		9AH	URTS	SMOD2	URM0X6	S1TR	S1MOD1	S1X12	--	--	000000xxB
S1BRT	S1 Baud-Rate Timer	9AH	S1BRT.7	S1BRT.6	S1BRT.5	S1BRT.4	S1BRT.3	S1BRT.2	S1BRT.1	S1BRT.0	00000000B
P2	*Port 2*	*A0H*	*P2.7*	*P2.6*	*P2.5*	*P2.4*	*P2.3*	*P2.2*	*P2.1*	*P2.0*	*11111111B*
AUXR1	Auxiliary Register 1	A2H	P4KBI	P4PCA	P5SPI	P4S1	--	--	--	DPS	0000xxx0B
AUXR2	Auxiliary Register 2	A6H	T0X12	T1X12	--	--	--	--	T1CKOE	T0CKOE	00xxxx00B
IE	Interrupt Enable	A8H	EA	--	ET2	ES0	ET1	EX1	ET0	EX0	0x000000B
SADDR	Slave Address	A9H									00000000B
SFRPI	SFR 页 Index	ACH	--	--	--	--	IDX3	IDX2	IDX1	IDX0	xxxx0000B
EIE1	Extended INT Enable 1	ADH	--	--	EKB	ES1	EBD	EPCA	EADC	ESPI	xx000000B
EIP1L	Ext. INT Priority 1 Low	AEH	--	--	PKBL	PS1L	PBDL	PPCAL	PADCL	PSPIL	xx000000B
EIP1H	Ext. INT Priority 1 High	AFH	--	--	PKBH	PS1H	PBDH	PPCAH	PADCH	PSPIH	xx000000B
P3	*Port 3*	*B0H*	*P3.7*	*P3.6*	*P3.5*	*P3.4*	*P3.3*	*P3.2*	*P3.1*	*P3.0*	*11111111B*
P3M0	P3 Mode Register 0	B1H	P3M0.7	P3M0.6	P3M0.5	P3M0.4	P3M0.3	P3M0.2	P3M0.1	P3M0.0	00000000B
P3M1	P3 Mode Register 1	B2H	P3M1.7	P3M1.6	P3M1.5	P3M1.4	P3M1.3	P3M1.2	P3M1.1	P3M1.0	00000000B
P4M0	P4 Mode Register 0	B3H	--	P4M0.6	P4M0.5	P4M0.4	P4M0.3	P4M0.2	P4M0.1	P4M0.0	x0000000B
P4M1	P4 Mode Register 1	B4H	--	P4M1.6	P4M1.5	P4M1.4	P4M1.3	P4M1.2	P4M1.1	P4M1.0	x0000000B
P5M0	P5 Mode Register 0	B5H	--	--	--	--	P5M0.3	P5M0.2	P5M0.1	P5M0.0	xxxx0000B
P5M1	P5 Mode Register 1	B6H	--	--	--	--	P5M1.3	P5M1.2	P5M1.1	P5M1.0	Xxxx0000B
IP0H	Interrupt Priority 0 High	B7H	PX3H	PX2H	PT2H	PSH	PT1H	PX1H	PT0H	PX0H	00000000B
IP0L	*Interrupt Priority Low*	*B8H*	***PX3L***	***PX2L***	*PT2L*	*PSL*	*PT1L*	*PX1L*	*PT0L*	*PX0L*	*00000000B*
SADEN	Slave Address Mask	B9H									00000000B
ADCVL	ADC result Low	BEH	--	--	--	--	--	--	ADCV.1	ADCV.0	xx001010B
XICON	External INT Control	C0H	**IT3H**	EX3	IE3	IT3	**IT2H**	EX2	IE2	IT2	00000000B
ADCON	ADC Control	C5H	ADCEN	SPEED1	SPEED0	ADCI	ADCS	CHS2	CHS1	CHS0	00000000B

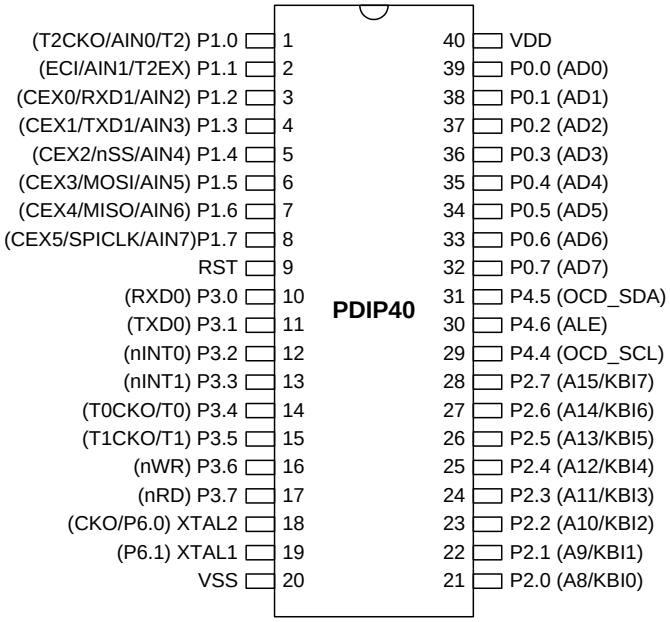
ADCV	ADC result	C6H	ADCV.9	ADCV.8	ADCV.7	ADCV.6	ADCV.5	ADCV.4	ADCV.3	ADCV.2	00000000B
PCON2	Clock Control 0	C7H	OSCDR	--	--	--	--	SCKS2	SCKS1	SCKS0	xxxxx000B
T2CON	*Timer 2 Control*	*C8H*	*TF2*	*EXF2*	*RCLK*	*TCLK*	*EXEN2*	*TR2*	*C/T2*	*CP/RL*	*00000000B*
P6	*Port 6*	*C8H*	*--*	*--*	*--*	*--*	*--*	*--*	*P6.1*	*P6.0*	*xxxxxx11B*
T2MOD	Timer2 mode	C9H	--	--	--	**T2X12**	--	--	T2OE	DCEN	xxx0xx00B
RCAP2L	Timer2 Capture Low	CAH									00000000B
RCAP2H	Timer2 Capture High	CBH									00000000B
TL2	Timer Low 2	CCH									00000000B
TH2	Timer High 2	CDH									00000000B
PSW	*Program Status Word*	*D0H*	*CY*	*AC*	*F0*	*RS1*	*RS0*	*OV*	*F1*	*P*	*00000000B*
KBPATN	Keypad Pattern	D5H									11111111B
KBCON	Keypad Control	D6H							PATNS	KBIF	xxxxxx00B
KBMASK	Keypad Int. Mask	D7H									00000000B
CCON	*PCA Control Reg.*	*D8H*	*CF*	*CR*	*CCF5*	*CCF4*	*CCF3*	*CCF2*	*CCF1*	*CCF0*	*00000000B*
CMOD	PCA Mode Reg.	D9H	CIDL	FEOV	--	--	--	CPS1	CPS0	ECF	00xxx000B
CCAPM0	PCA Module0 Mode	DAH	--	ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0	x0000000B
CCAPM1	PCA Module1 Mode	DBH	--	ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1	x0000000B
CCAPM2	PCA Module2 Mode	DCH	--	ECOM2	CAPP2	CAPN2	MAT2	TOG2	PWM2	ECCF2	x0000000B
CCAPM3	PCA Module3 Mode	DDH	--	ECOM3	CAPP3	CAPN3	MAT3	TOG3	PWM3	ECCF3	x0000000B
CCAPM4	PCA Module4 Mode	DEH	--	ECOM4	CAPP4	CAPN4	MAT4	TOG4	PWM4	ECCF4	x0000000B
CCAPM5	PCA Module5 Mode	DFH	--	ECOM5	CAPP5	CAPN5	MAT5	TOG5	PWM5	ECCF5	x0000000B
ACC	*Accumulator*	*E0H*	*ACC.7*	*ACC.6*	*ACC.5*	*ACC.4*	*ACC.3*	*ACC.2*	*ACC.1*	*ACC.0*	*00000000B*
WDTCR	Watch-dog-timer Control register	E1H	WRF	--	ENW	CLW	WIDL	PS2	PS1	PS0	0x000000B
IFD	ISP Flash data	E2H									11111111B
IFADRH	ISP Flash address High	E3H									00000000B
IFADRL	ISP Flash Address Low	E4H									00000000B
IFMT	ISP Mode Table	E5H	--	--	--	MS4	MS3	MS2	MS1	MS0	xxxx0000B
IAPLB	IAP Low Boundary	注 1	IAPLB6	IAPLB5	IAPLB4	IAPLB3	IAPLB2	IAPLB1	IAPLB0	--	IFR7
AUXRA	Auxiliary Register A	注 1	DBOD	BORE	OCDE	ILRCOE	XTALE	IHRCOE	OSCS1	OSCS0	00100100B
AUXRB	Auxiliary Register A	注 1	--	--	--	IAP0	LPM3	LPM2	--	LPM0	xxx000x0B
SCMD	ISP Serial Command	E6H									xxxxxxxxB
ISPCR	ISP Control Register	E7H	ISPEN	BS	SRST	CFAIL	--	PCKS2	PCKS1	PCKS0	0000x000B
P4	*Port 4*	*E8H*	*--*	*P4.6*	*P4.5*	*P4.4*	*P4.3*	*P4.2*	*P4.1*	*P4.0*	*x1111111B*
CL	PCA base timer Low	E9H									00000000B
CCAP0L	PCA module0 Capture Low	EAH									00000000B
CCAP1L	PCA module1 capture Low	EBH									00000000B
CCAP2L	PCA module2 capture Low	ECH									00000000B
CCAP3L	PCA module3 capture Low	EDH									00000000B
CCAP4L	PCA module4 capture Low	EEH									00000000B
CCAP5L	PCA module5 capture Low	EFH									00000000B
B	*B Register*	*F0H*	*F7H*	*F6H*	*F5H*	*F4H*	*F3H*	*F2H*	*F1H*	*F0H*	*00000000B*
PCAPWM0	PCA PWM0 Mode	F2H	**P0RS1**	**P0RS0**	**P0PS2**	**P0PS1**	**P0PS0**	**P0INV**	EPC0H	EPC0L	00000000B
PCAPWM1	PCA PWM1 Mode	F3H	**P1RS1**	**P1RS0**	**P1PS2**	**P1PS1**	**P1PS0**	**P1INV**	EPC1H	EPC1L	00000000B
PCAPWM2	PCA PWM2 Mode	F4H	**P2RS1**	**P2RS0**	**P2PS2**	**P2PS1**	**P2PS0**	**P2INV**	EPC2H	EPC2L	00000000B
PCAPWM3	PCA PWM3 Mode	F5H	**P3RS1**	**P3RS0**	**P3PS2**	**P3PS1**	**P3PS0**	**P3INV**	EPC3H	EPC3L	00000000B
PCAPWM4	PCA PWM4 Mode	F6H	**P4RS1**	**P4RS0**	**P4PS2**	**P4PS1**	**P4PS0**	**P4INV**	EPC4H	EPC4L	00000000B
PCAPWM5	PCA PWM5 Mode	F7H	**P5RS1**	**P5RS0**	**P5PS2**	**P5PS1**	**P5PS0**	**P5INV**	EPC5H	EPC5L	00000000B
P5	*Port 5*	*F8H*					*P5.3*	*P5.2*	*P5.1*	*P5.0*	*xxxx1111B*
CH	PCA base timer High	F9H									00000000B
CCAP0H	PCA Module0 capture High	FAH									00000000B
CCAP1H	PCA Module1 capture High	FBH									00000000B
CCAP2H	PCA Module2 capture High	FCH									00000000B

CCAP3H	PCA Module3 capture High	FDH									00000000B
CCAP4H	PCA Module4 capture High	FEH									00000000B
CCAP5H	PCA Module5 capture High	FFH									00000000B

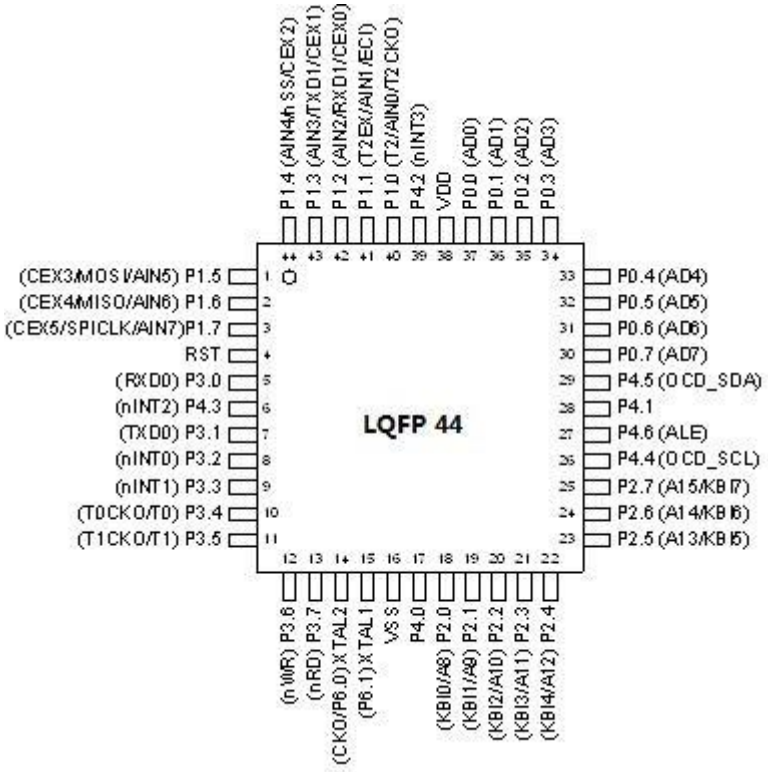
注 1: 此寄存器的地址由 IFMT 和 SCMD 决定。更多的信息请参考 IFMT 寄存器的描述。

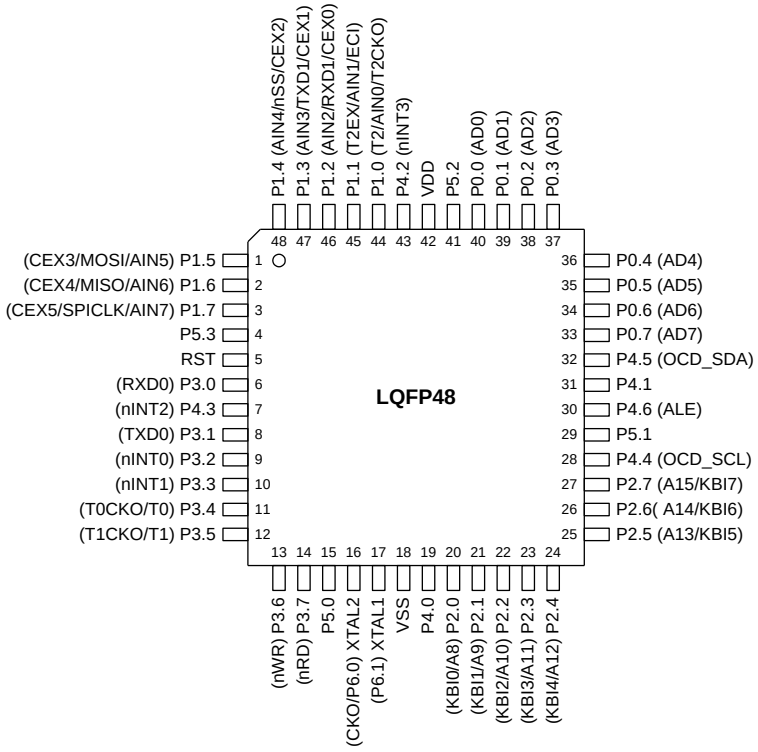
4. 引脚

4.1. 引脚结构



（PDIP-40 快速开发验证功能用，不推荐批量生产）




LQFP48
1 (CEX3/MOSI/AIN5) P1.5
2 (CEX4/MISO/AIN6) P1.6
3 (CEX5/SPICLK/AIN7) P1.7
4 P5.3
5 RST
6 (RXD0) P3.0
7 (nINT2) P4.3
8 (TXD0) P3.1
9 (nINT0) P3.2
10 (nINT1) P3.3
11 (T0CKO/T0) P3.4
12 (T1CKO/T1) P3.5
13 (nWR) P3.6
14 (nRD) P3.7
15 P5.0
16 (CKO/P6.0) XTAL2
17 (P6.1) XTAL1
18 VSS
19 P4.0
20 (KBI0/A8) P2.0
21 (KBI1/A9) P2.1
22 (KBI2/A10) P2.2
23 (KBI3/A11) P2.3
24 (KBI4/A12) P2.4
25 P2.5 (A13/KBI5)
26 P2.6(A14/KBI6)
27 P2.7 (A15/KBI7)
28 P4.4 (OCD_SCL)
29 P5.1
30 P4.6 (ALE)
31 P4.1
32 P4.5 (OCD_SDA)
33 P0.7 (AD7)
34 P0.6 (AD6)
35 P0.5 (AD5)
36 P0.4 (AD4)
37 P0.3 (AD3)
38 P0.2 (AD2)
39 P0.1 (AD1)
40 P0.0 (AD0)
41 P5.2
42 VDD
43 P4.2 (nINT3)
44 P1.0 (T2/AIN0/T2CKO)
45 P1.1 (T2EX/AIN1/ECI)
46 P1.2 (AIN2/RXD1/CEX0)
47 P1.3 (AIN3/TXD1/CEX1)
48 P1.4 (AIN4/nSS/CEX2)

4.2. 引脚定义

助记符	40-Pin PDIP *	44-Pin LQFP	48-Pin LQFP	I/O 类型	描 述
P0.0 (AD0)	39	37	40	I/O	* Port 0.0. * AD0: multiplexed A0/D0 during external data memory access.
P0.1 (AD1)	38	36	39	I/O	* Port 0.1. * AD1: multiplexed A1/D1 during external data memory access.
P0.2 (AD2)	37	35	38	I/O	* Port 0.2. * AD2: multiplexed A2/D2 during external data memory access.
P0.3 (AD3)	36	34	37	I/O	* Port 0.3. * AD3: multiplexed A3/D3 during external data memory access.
P0.4 (AD4)	35	33	36	I/O	* Port 0.4. * AD4: multiplexed A4/D4 during external data memory access.
P0.5 (AD5)	34	32	35	I/O	* Port 0.5. * AD5: multiplexed A5/D5 during external data memory access.
P0.6 (AD6)	33	31	34	I/O	* Port 0.6. * AD6: multiplexed A6/D6 during external data memory access.
P0.7 (AD7)	32	30	33	I/O	* Port 0.7. * AD7: multiplexed A7/D7 during external data memory access.
P1.0 (T2) (AIN0) (T2CKO)	1	40	44	I/O	* Port 1.0. * T2: Timer/Counter 2 external input. * AIN0: ADC channel-0 analog input. * T2CKO: programmable clock-out from Timer 2.
P1.1 (T2EX) (AIN1) (ECI)	2	41	45	I/O	* Port 1.1. * T2EX: Timer/Counter 2 Reload/Capture/Direction control. * AIN1: ADC channel-1 analog input. * ECI: PCA external clock input.
P1.2 (AIN2) (RXD1) (CEX0)	3	42	46	I/O	* Port 1.2. * AIN2: ADC channel-2 analog input. * RXD1: UART1 serial input port. * CEX0: PCA module-0 external I/O.
P1.3 (AIN3) (TXD1) (CEX1)	4	43	47	I/O	* Port 1.3. * AIN3: ADC channel-3 analog input. * TXD1: UART1 serial output port. * CEX1: PCA module-1 external I/O.
P1.4 (AIN4) (nSS) (CEX2)	5	44	48	I/O	* Port 1.4. * AIN4: ADC channel-4 analog input. * nSS: SPI Slave select. * CEX2: PCA module-2 external I/O.
P1.5 (AIN5) (MOSI) (CEX3)	6	1	1	I/O	* Port 1.5. * AIN5: ADC channel-5 analog input. * MOSI: SPI master out & slave in. * CEX3: PCA module-3 external I/O.
P1.6 (AIN6) (MISO) (CEX4)	7	2	2	I/O	* Port 1.6. * AIN6: ADC channel-6 analog input. * MISO: SPI master in & slave out. * CEX4: PCA module-4 external I/O.
P1.7	8	3	3	I/O	* Port 1.7.

(AIN7) (SPICLK) (CEX5)					* AIN7: ADC channel-7 analog input. * SPICLK: SPI clock, output for master and input for slave. * CEX5: PCA module-5 external I/O.
P2.0 (A8) (KBI0)	21	18	20	I/O	* Port 2.0. * A8: A8 output during external data memory access. * KBI0: keypad input 0.
P2.1 (A9) (KBI1)	22	19	21	I/O	* Port 2.1. * A9: A9 output during external data memory access. * KBI1: keypad input 1.
P2.2 (A10) (KBI2)	23	20	22	I/O	* Port 2.2. * A10: A10 output during external data memory access. * KBI2: keypad input 2.
P2.3 (A11) (KBI3)	24	21	23	I/O	* Port 2.3. * A11: A11 output during external data memory access. * KBI3: keypad input 3.
P2.4 (A12) (KBI4)	25	22	24	I/O	* Port 2.4. * A12: A12 output during external data memory access. * KBI4: keypad input 4.
P2.5 (A13) (KBI5)	26	23	25	I/O	* Port 2.5. * A13: A13 output during external data memory access. * KBI5: keypad input 5.
P2.6 (A14) (KBI6)	27	24	26	I/O	* Port 2.6. * A14: A14 output during external data memory access. * KBI6: keypad input 6.
P2.7 (A15) (KBI7)	28	25	27	I/O	* Port 2.7. * A15: A15 output during external data memory access. * KBI7: keypad input 7.
P3.0 (RXD0)	10	5	6	I/O	* Port 3.0. * RXD0: UART0 serial input port.
P3.1 (TXD0)	11	7	8	I/O	* Port 3.1. * TXD0: UART0 serial output port.
P3.2 (nINT0)	12	8	9	I/O	* Port 3.2. * nINT0: external interrupt 0 input.
P3.3 (nINT1)	13	9	10	I/O	* Port 3.3. * nINT1: external interrupt 1 input.
P3.4 (T0) (T0CKO)	14	10	11	I/O	* Port 3.4. * T0: Timer/Counter 0 external input. * T0CKO: programmable clock-out from Timer 0.
P3.5 (T1) (T1CKO)	15	11	12	I/O	* Port 3.5. * T1: Timer/Counter 1 external input. * T1CKO: programmable clock-out from Timer 1.
P3.6 (nWR)	16	12	13	I/O	* Port 3.6. * nWR: external data memory write strobe.
P3.7 (nRD)	17	13	14	I/O	* Port 3 bit-7. * nRD: external data memory read strobe.
P4.0	-	17	19	I/O	* Port 4.0.
P4.1	-	28	31	I/O	* Port 4.1.
P4.2 (nINT3)	-	39	43	I/O	* Port 4.2. * nINT3: external interrupt 3 input.
P4.3 (nINT2)	-	6	7	I/O	* Port 4.3. * nINT2: external interrupt 2 input.
P4.4 (OCD_SCL)	29	26	28	I/O	* Port 4.4. * OCD_SCL: OCD interface, serial clock.
P4.5	31	29	32	I/O	* Port 4.5. * OCD_SDA: OCD interface, serial data.
P4.6 (ALE)	30	27	30	I/O	* Port 4.6. * ALE: Address Latch Enable, output pulse for latching

					the low byte of the address during an access cycle to external data memory.
P5.0	-	-	15	I/O	* Port 5.0.
P5.1	-	-	29	I/O	* Port 5.1.
P5.2	-	-	41	I/O	* Port 5.2.
P5.3	-	-	4	I/O	* Port 5.3.
P6.0 (CKO) (ECKI) (XTAL2)	18	14	16	I/O O I O	* Port 6.0. It is only accessed in SFR 页 “F”. * CKO: Enable internal High frequency RC-Oscillator output. * ECKI: In external clock input mode, this is clock input pin. *XTAL2: Output of on-chip crystal oscillating circuit.
P6.1 (XTAL1)	19	15	17	I/O I	* Port 6.1. It is only accessed in SFR 页 “F”. *XTAL1: Input of on-chip crystal oscillating circuit.
RST	9	4	5	I	*RST: External RESET input, high active.
VDD	40	38	42	I	Power supply.
VSS	20	16	18	I	Ground, 0 V reference.

*: （PDIP-40 快速开发验证功能用，不推荐批量生产）

4.3. 引脚功能重映像

许多 I/O 引脚,除了正常的 I/O 功能之外,也有其它复用功能。默认情况下,P2 和 P1 被键盘中断、PCA、SPI 和 UART1 复用。但是，使用者可以通过设定 AUXR1 寄存器的 P4KB、P4PCA、P5SPI 和 P4S1 控制位使上面的那些功能映像到 P4 上。当所需要的引脚数多于 40 个的时候，此功能尤其有用。注意，任何时候这四个控制位只能有一个被置位。

AUXR1：辅助控制寄存器 1

SFR 页 = 全部

SFR 地址 = 0xA2 复位值 = 0000-XXX0

7	6	5	4	3	2	1	0
P4KBI	P4PCA	P5SPI	P4S1	--	--	--	DPS
R/W	R/W	R/W	R/W	R	R	R	R/W

Bit 7: P4KBI, KBI 功能映像到 P4/P5
0: 禁止 KBI 功能映像到 P4/P5。
1: 设置 KBI 功能映像到 P4/P5，作如下定义：
P2.0 上的‘KBI0’功能映像到 P4.0
P2.1 上的‘KBI1’功能映像到 P4.1
P2.2 上的‘KBI2’功能映像到 P4.2
P2.3 上的‘KBI3’功能映像到 P4.3
P2.4 上的‘KBI4’功能映像到 P5.1
P2.5 上的‘KBI5’功能映像到 P5.0
P2.6 上的‘KBI6’功能映像到 P5.2
P2.7 上的‘KBI7’功能映像到 P5.3

Bit 6: P4PCA, PCA 功能映像到 P4/P5
0: 禁止 PCAI 功能映像到 P4/P5。
1: 设置 PCA 功能映像到 P4/P5，作如下定义：
P1.1 上的‘ECI’功能映像到 P4.2
P1.2 上的‘CEX0’功能映像到 P4.0
P1.3 上的‘CEX1’功能映像到 P4.1
P1.4 上的‘CEX2’功能映像到 P5.0
P1.5 上的‘CEX3’功能映像到 P5.1
P1.6 上的‘CEX4’功能映像到 P5.2
P1.7 上的‘CEX5’功能映像到 P5.3

Bit 5: P5SPI, SPI 接口使能/禁止
0: 禁止 SPI 功能映像到 P5
1: 使能 SPI 功能映像到 P5，作如下定义：
P1.4 上的‘/SS’功能映像到 P5.0
P1.5 上的‘MOSI’功能映像到 P5.1
P1.6 上的‘MISO’功能映像到 P5.2
P1.7 上的‘SPICLK’功能映像到 P5.3

Bit 4: P4S1, 串口 1 (UART1) 功能映像到 P4.0/P4.1.
0: 禁止 UART1 功能映像到 P4。
1: 设置 UART1 功能映像到 P4，作如下定义：
P1.2 上的‘RXD1’功能映像到 P4.0
P1.3 上的‘TXD1’功能映像到 P4.1

5. 8051 CPU 功能描述

5.1. CPU 寄存器

PSW: 程序状态字

SFR 页 = 全部

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

当最后一个算数运算有进位（加）或借位（减）的时候，该位被置位。

其它的算术运算将它清除为逻辑 0。

AC: 辅助进位标志。(对于 BCD 运算)

当最后一个算数运算向高四位有进位（加）或借位（减）的时候，该位被置位。

其它的算术运算将它清除为逻辑 0。

F0: 标志 0

可位寻址，通常作为用户使用的软件控制标志位。

RS1: 寄存器组选择位 1

RS0: 寄存器组选择位 0

(RS1, RS0) 工作寄存器组和地址

(0, 0) Bank 0 (00H~07H)

(0, 1) Bank 1 (08H~0FH)

(1, 0) Bank 2 (10H~17H)

(1, 1) Bank 3 (18H~1FH)

OV: 溢出标志

这位在下列的环境之下被设定成 1：

- ADD， ADDC， SUBB 指令引起的数据的溢出；
- MUL 指令的结果引起的溢出 (结果超过 255)；
- DIV 指令除数为零；

ADD， ADDC， SUBB， MUL， DIV 指令的其它结果将该位清 0。

F1: 标志 1

可位寻址，通常作为用户使用的软件控制标志位。

P: 奇偶标志

每个指令周期由硬件置 1 或清 0，用来指示累加器中“1”为奇数个或偶数个。

(注意：PSW 寄存器可位寻址，所有的被释放的位能被软件设定或清除。)

SP: 堆栈指针

SFR 页 = 全部

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

DPL: 数据指针低

SFR 页 = 全部

SFR 地址 = 0x82 复位值 = 0000-0000

7	6	5	4	3	2	1	0
DPL[7]	DPL[6]	DPL[5]	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

DPH: 数据指针高

SFR 页 = 全部

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

B: B 寄存器

SFR 页 = 全部

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

5.2. CPU 时序

MA805-64是一个基于高性能1-T结构80C51 CPU的单芯片微型控制器，与8051兼容指令，每条指令执行需要1~7 个时钟周期(大约是标准8051芯片的6~7倍)。它使用管道结构，此结构的指令吞吐量超过标准8051结构。指令时序也不同于标准8051。

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“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 寻址方式

直接寻址(DIR)

在直接寻址中，操作数通过在指令中的一个 8 位地址被指定。仅内部 RAM 和 SFRs 能被直接寻址。

直接寻址

在指令中直接给出操作数地址的就属于直接寻址。此时，指令中的操作数部分就是操作数的地址。例如指令：

```
MOV  A,4FH              ;(A)←(4FH)
```

可用于直接寻址的空间是，内部数据 RAM 的低 128 字节及特殊功能寄存器 SFRs。

寄存器间接寻址

由指令中指出某一个寄存器的内容作为操作数的地址。内部 RAM 和外部 RAM 都能通过间接寻址方式进行访问。使用当前工作寄存器组中的 R0 或 R1 存放操作数单元的地址指针（8 位地址），在执行 PUSH（压栈）和 POP（出栈）指令时采用堆栈指针 SP 作寄存间接寻址。而如果地址是 16 位时就只能使用 DPTR 数据指针作间接寻址了。例如指令：

```
MOV  A,@R0              ;(A) ←((R0))
MOVX A,@R1              ;(A) ←((R1))
MOVX A,@DPTR            ;(A) ←((DPTR))
```

寄存器寻址(REG)

寄存器寻址就是以通过寄存器的内容作为操作数。在指令的助记符号中直接以寄存器的名字来表示操作数的地址。例如指令：

```
MOV  A,R0               ;(A) ←(R0)
ADD  A,R0               ;(A) ←(Acc)+(R0)
```

能用于这种寻址方式的寄存器还有 ACC、B、DPTR、AB(双字节)和 CY(位累加器)。

变址寻址

以某个寄存器的内容作为基本地址，然后在这个基本地址基础上加上地址偏移量才是真正的操作数地址。例如指令：

```
MOVC A,@A+DPTR          ;(A) ←((A)+(DPTR))
```

不论用 DPTR 或是 PC 作为基址指针，变址寻址方式都只适用于 8051 的程序存储器，通常用于读取数据表。

立即寻址

指令中地址码部分给出的就是操作数。即取出指令的同时立即得到了操作数。例如指令：

```
MOV  A,#4FH                 ;(A) ←6FH
```

相对寻址

相对寻址时，由程序计数器 PC 提供的基地址与指令中提供的偏移量 rel 相加，得到操作数的地址。这时指出的地址是操作数与现行指令的相对位置。例如指令：

```
SJMP rel                    ;PC←(PC)+2+rel
```

位寻址

操作数是二进制数的某一位，其位地址出现在指令中，例如指令：

```
SETB bit                    ;(bit) ←1
```

6. 存储器

像所有的 80C51 一样， MA805-64 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 64K 字节。在 MA805-64 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。

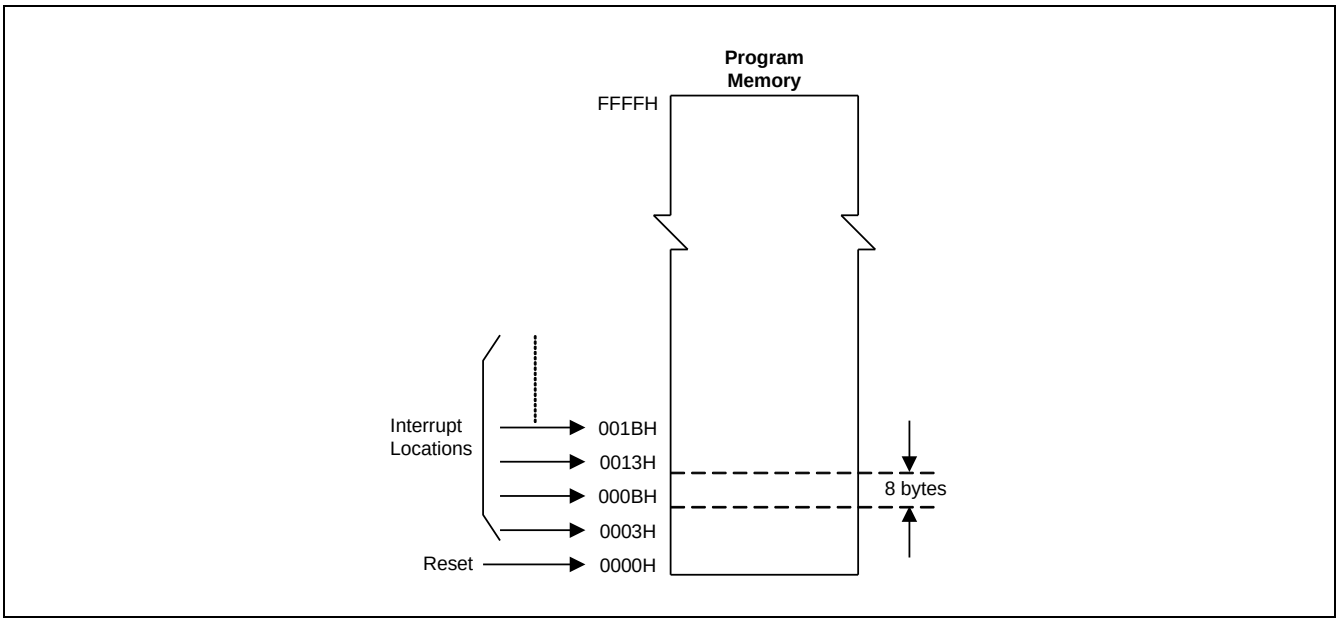
数据存储器使用与程序存储器不同的地址空间。MA805-64 有 256 字节的内部 RAM 和 1024 字节的片上扩展存储器(XRAM)。

6.1. 片上程序存储器

程序存储器用来保存让 CPU 进行处理的程序代码，如图 7-1 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。如果其它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 7-1 程序存储器



6.2. 片上数据存储器

图 7-2 向 MA805-64 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，但他们实际上是分开的。

如图 7-3 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 7-4 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 个地址同时支持位地址和字节地址。可以使用位地址寻址的 SFR 的地址末位是 0H 或 8H。

为了访问外部数据存储器，EXTRAM 位应该被设为 1。访问外部数据存储器可以使用一个 16 位地址 (使用‘MOVX @DPTR’)或一个 8 位地址 (使用 ‘MOVX @Ri’)，下面详细说明。

用8位地址访问

8 位地址通常使用 1 根或更多的 I/O 口标明 RAM 的页数。如果使用 8 位地址，在访问外部存储器的周期中，P2 寄存器保持 P2 引脚的状态，这将保证页的访问。 图 7-5 展示了一个 2K 字节外部数据存储器的硬件配置。P0 口作为地址和数据总线复用，而 P2 口的三根线用于标明 RAM 的页数。处理器产生/RD 和/WR (P3.7 和 P3.6 附加功能)信号控制存储器。当然也可以使用其它的 I/O 口而非 P2 口来标明 RAM 的页数。

用16位地址访问

16 位地址通常用于访问 64K 字节的外部数据存储器。图 7-6 展示了一个 64K 字节外部数据存储器的硬件配置。当使用 16 位地址的时候，除了 P0，nRD 和 nWR 的动作以外，地址的高字节通过 P2 口输出，并且在读或写周期中是被锁定的。

无论如何，地址的低字节和数据字节在 P0 口是时分复用的。ALE (地址锁存使能) 被用来使地址字节被外部锁存器锁存，地址字节在 ALE 负跳变时有效。在写周期中，数据在 nWR 有效之前在 P0 口出现，直到 nWR 无效的时候消失。在读周期中，数据在 nRD 信号无效之前被 P0 口接受。在任何外部存储器访问期间，CPU 向 P0 口锁存器(特殊功能寄存器)写 0FFH，以消除任何可能被锁存的数据。

访问片上扩展存储器 (XRAM)， EXTRAM 位应该被设为 0。图 7-2，这 1024 字节的 XRAM (0000H to 03FFH) 通过外部访问指令 MOVX 间接存取。 对 XRAM 的访问没有任何地址信号、地址锁存信号和读写控制信号的输出。这意味着 P0、 P2、 P4.6(ALE)、 P3.6 (nWR) 和 P3.7 (nRD) 在访问 XRAM 期间保持不变。

图 7-2 数据存储器

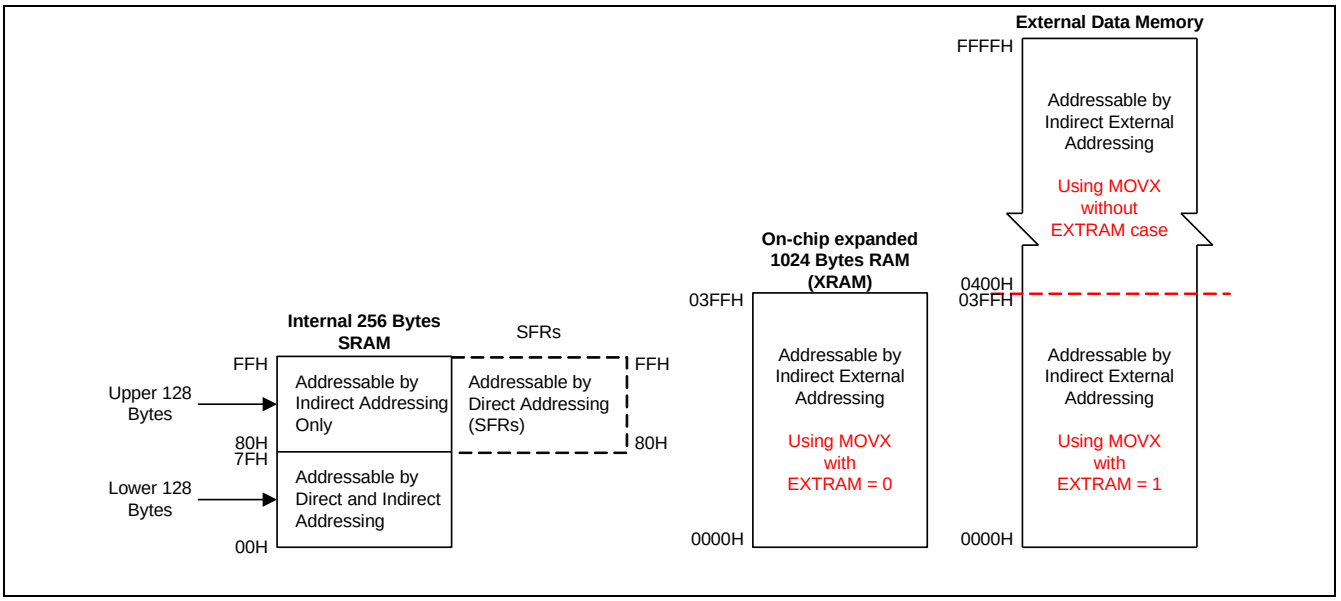


图 7-3 内部 RAM 的低 128 字节

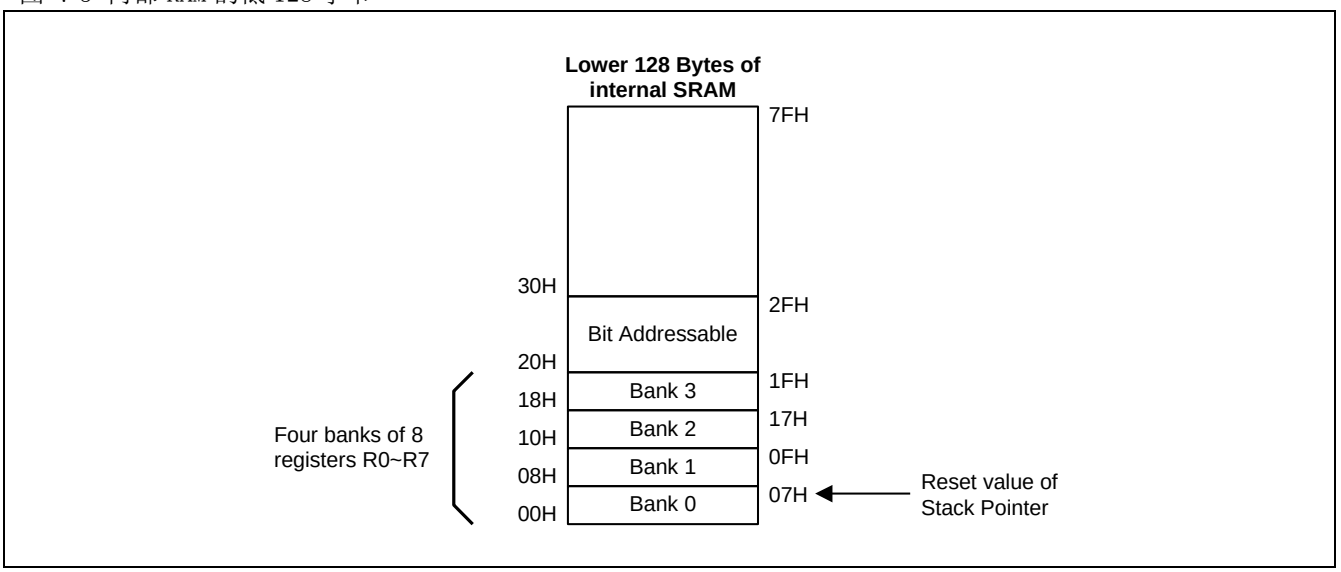


图 7-4 特殊功能寄存器空间

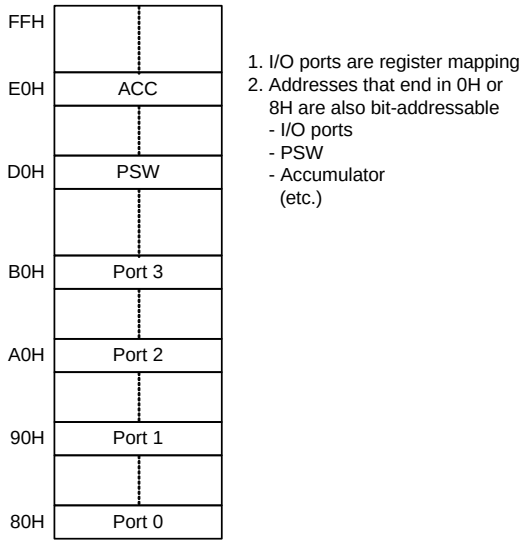
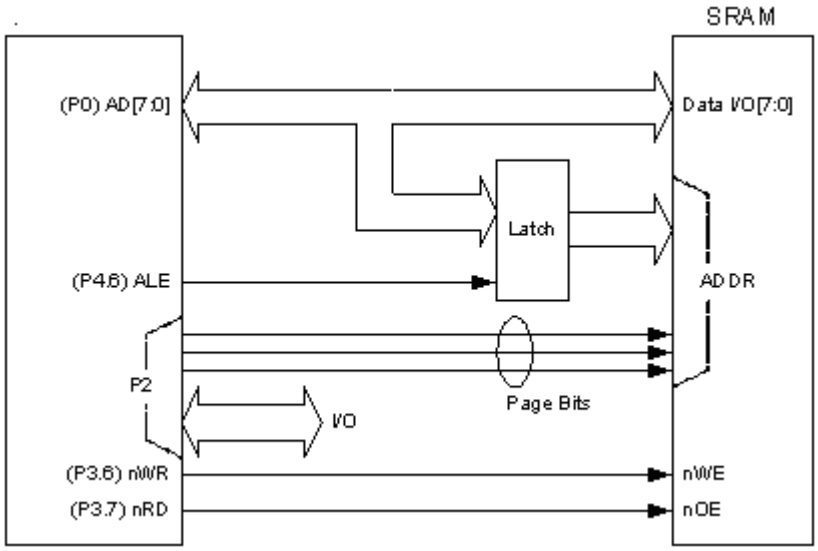


图 7-5 通过 8 位地址访问外部 RAM(使用‘MOVX @ Ri’ 和页位选择)



注：在这种情况下,P2 口的其它位（脚）可做一般 I/O 口使用。

图 7-6 通过 16 位地址访问外部 RAM(使用‘MOVX @ DPTR’)

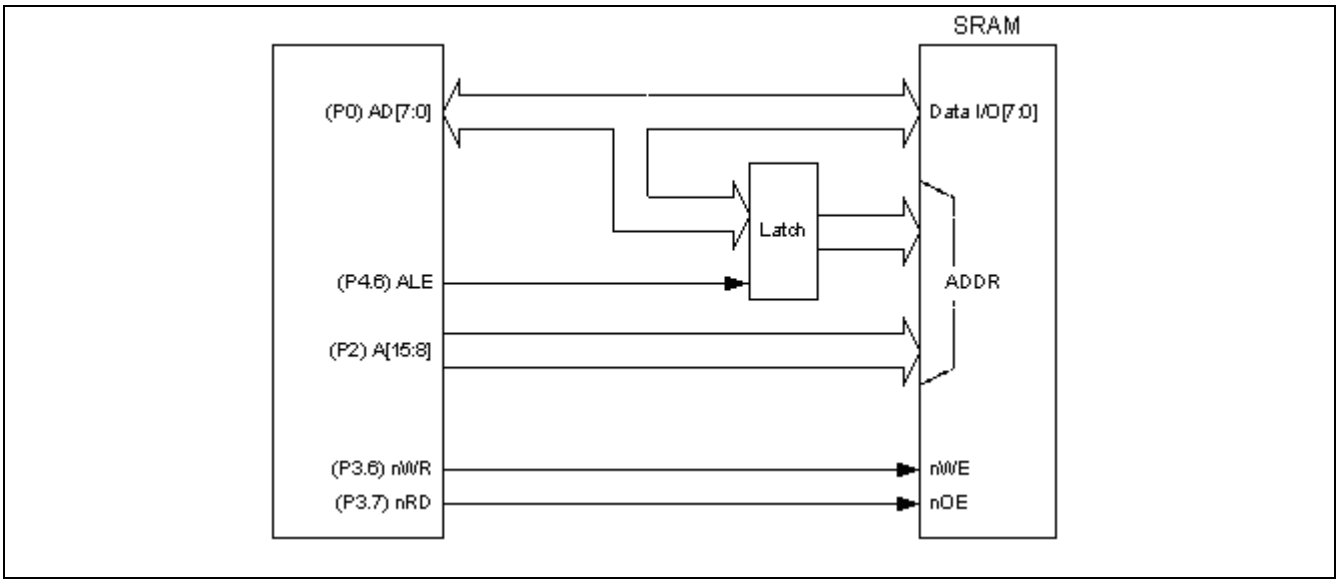
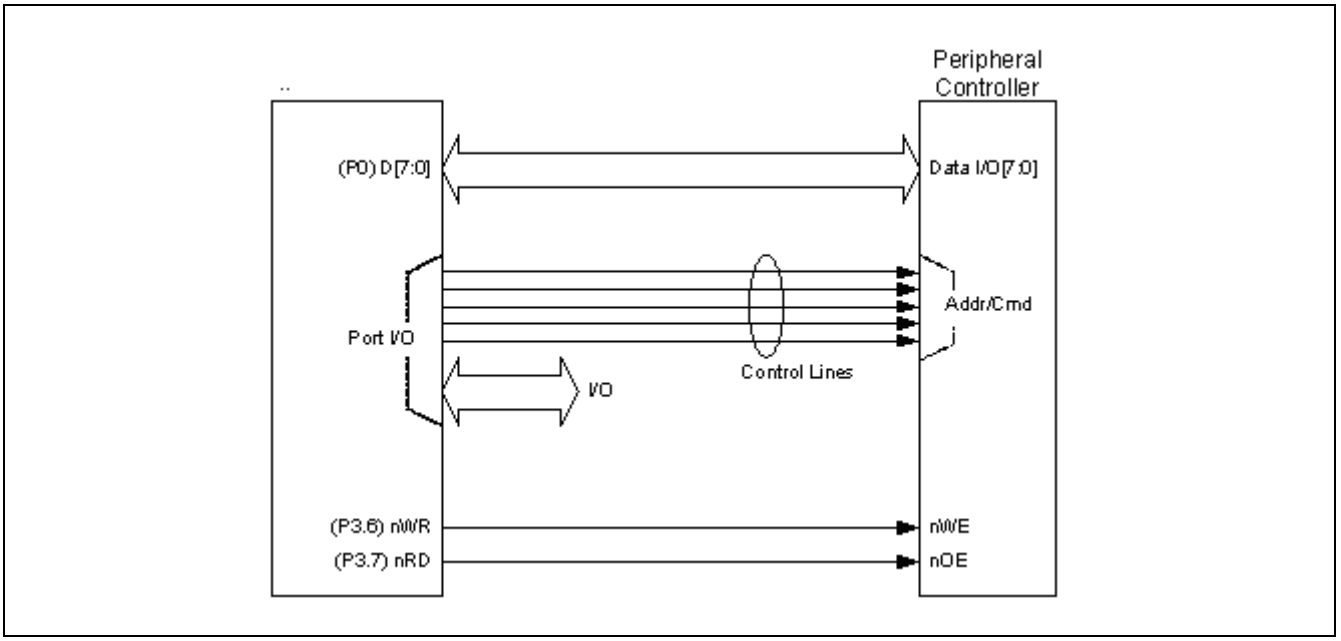


图 7-7 通过 I/O 口配置地址访问外部 RAM



注:这适合先进先出（FIFO）结构的访问，譬如像 NAND FLASH 类型的应用。

6.3. 片上扩展 RAM (XRAM)

访问片上扩展 RAM(XRAM)，参考图 7-2，EXTRAM 位应该被清零。这 1024 字节的 XRAM (地址从 0000H 到 03FFH) 被外部访问指令“MOVX @Ri”或“MOVX @DPTR” 间接访问。在 KEIL-C51 编译器中，使用“pdata”或“xdata”声明变量分配到 XRAM 中。编译后，被“pdata”或“xdata”声明过的变量将分别通过“MOVX @Ri”或“MOVX @DPTR”指令进行存取，这样 MA805-64 硬体才能正确访问 XRAM。

6.4. 外部数据存储器的存取

AUXR0: 辅助寄存器 0

SFR 页 = 全部

SFR 地址 = 0x8E 复位值 = 0000-0X0X

7	6	5	4	3	2	1	0
P60OC1	P60OC0	P60FD	P34FD	MOVXFD	ADRJ	EXTRAM	--
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R

Bit 3: MOVXFD, MOVX 输出信号快速驱动使能
0: MOVX 输出信号为默认速度
1: MOVX 输出信号设为快速驱动。如果有外部存储器，MOVX@DPTR 或 MOVX@Ri，MOVX 输出信号需要快速驱动以延长 ALE/RD/WR 脉冲频率超过 12MHz @5V 或 6MHz @3.3V。

Bit 1: EXTRAM, 外部数据 RAM 使能
0: 使能片上扩展数据存储器(XRAM 1024 字节)
1: 禁止片上扩展数据存储器

Stretch: MOVX 延长寄存器

SFR 页 = 全部

SFR 地址 = 0x8F 复位值 = 0X00-0000

7	6	5	4	3	2	1	0
EMAI1	--	ALES1	ALES0	RWSH	RWS2	RWS1	RWS0
R/W	R	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7: EMAI1, EMAI1 配置外部数据存储器访问接口模式如下：
0: 复用地址/数据
1: 无地址状态访问

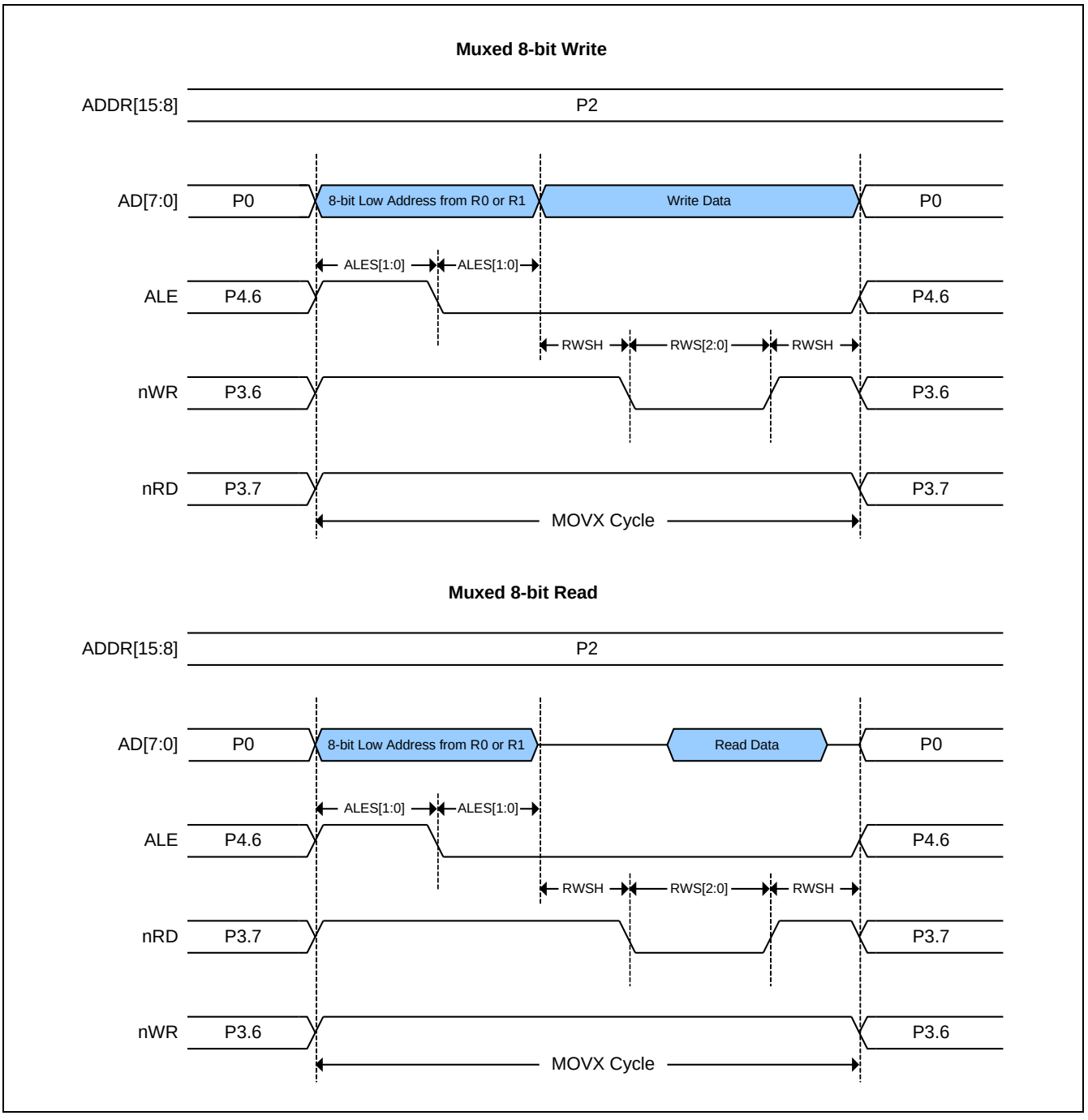
Bit 6: 保留。当对 STRETCH 进行写操作的时候此位必须写“0”

Bit 5~4: ALES[1:0], EMAI ALE 脉宽选择位。仅当 EMAI 为复用模式时有效
00: ALE 高和 ALE 低脉冲宽度为 1 个 SYSCLK 周期
01: ALE 高和 ALE 低脉冲宽度为 2 个 SYSCLK 周期
10: ALE 高和 ALE 低脉冲宽度为 3 个 SYSCLK 周期
11: ALE 高和 ALE 低脉冲宽度为 4 个 SYSCLK 周期

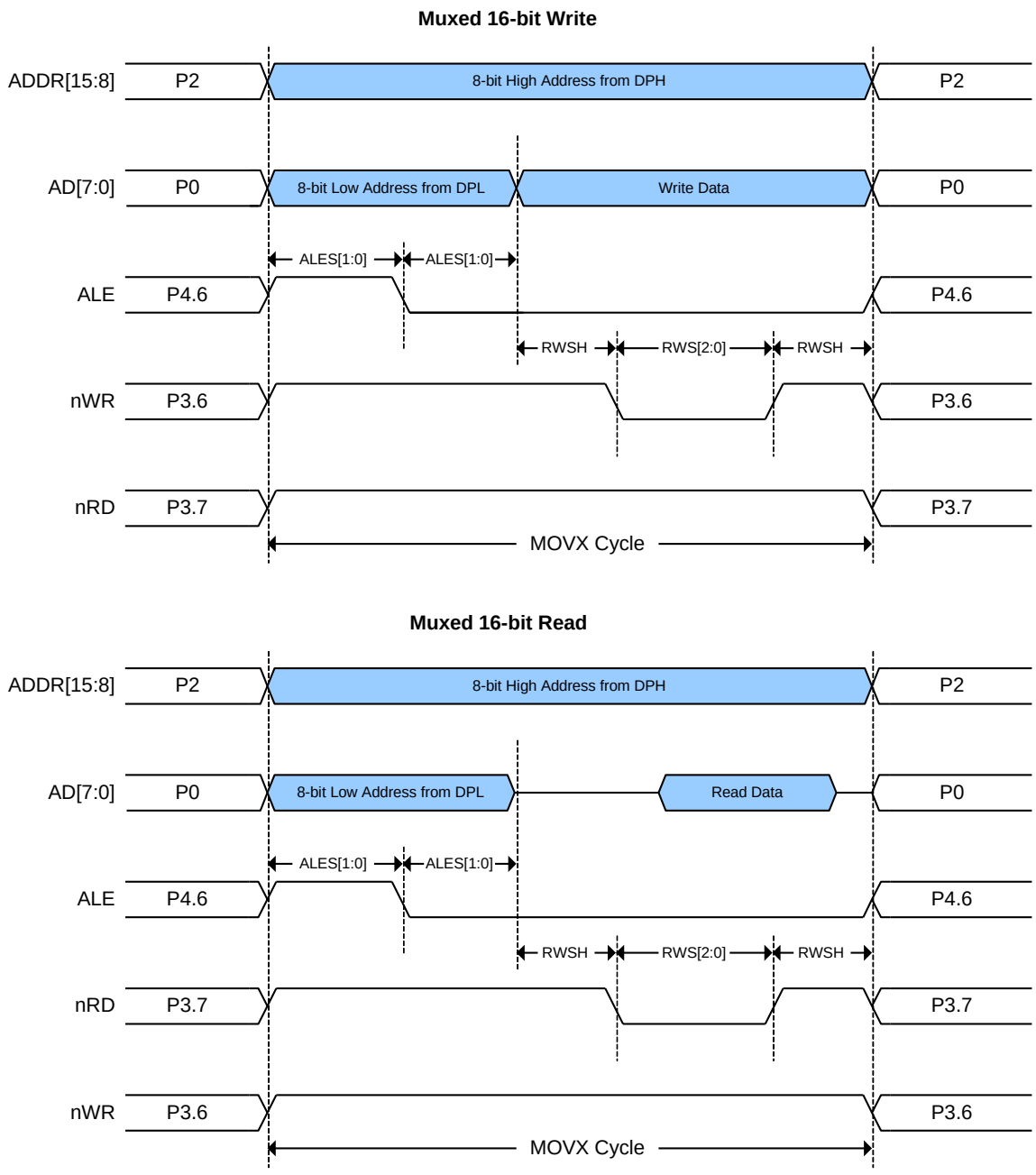
Bit 3: RWSH, EMAI 读/写脉冲设置/保持时间控制
0: /RD 和 /WR 命令设置/保持时间为 1 个 SYSCLK 周期
1: /RD 和 /WR 命令设置/保持时间为 2 个 SYSCLK 周期

Bit 2~0: RWS[2:0], EMAI 读/写命令脉冲宽度设置位
000: /RD 和 /WR 脉冲宽度为 1 个 SYSCLK 周期
001: /RD 和 /WR 脉冲宽度为 2 个 SYSCLK 周期
010: /RD 和 /WR 脉冲宽度为 3 个 SYSCLK 周期
011: /RD 和 /WR 脉冲宽度为 4 个 SYSCLK 周期
100: /RD 和 /WR 脉冲宽度为 5 个 SYSCLK 周期
101: /RD 和 /WR 脉冲宽度为 6 个 SYSCLK 周期
110: /RD 和 /WR 脉冲宽度为 7 个 SYSCLK 周期
111: /RD 和 /WR 脉冲宽度为 8 个 SYSCLK 周期

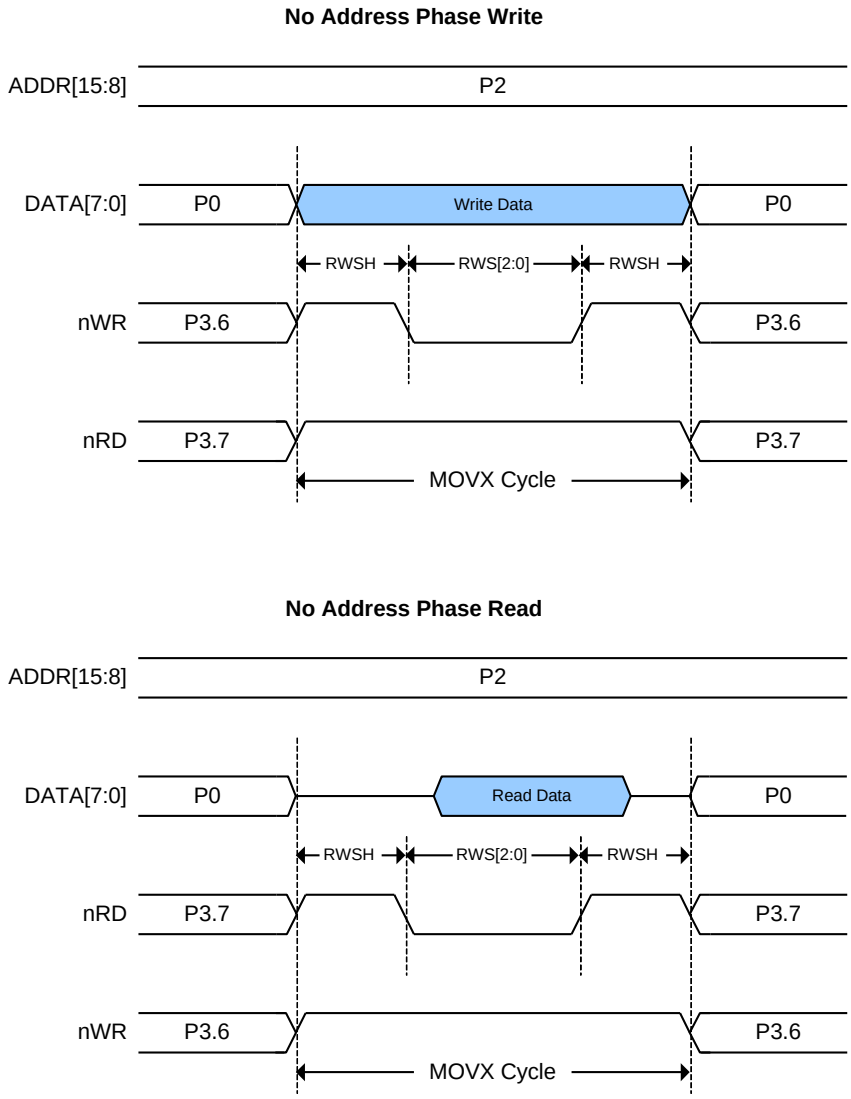
6.4.1.8 位 MOVX 复用模式



6.4.2. 16 位 MOVX 复用模式



6.4.3. MOVX 无地址状态模式



6.5. 关于 C51 编译器的声明标识符

C51 编译器的声明识别符与 MA805-64 存储空间的对应关系如下：

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
外部数据或片上的扩展 RAM (XRAM)。通过“MOVX @DPTR”指令访问标准 80C51 的 64K 存储空间。MA805-64 有 1024 字节的片上 xdata 存储空间。

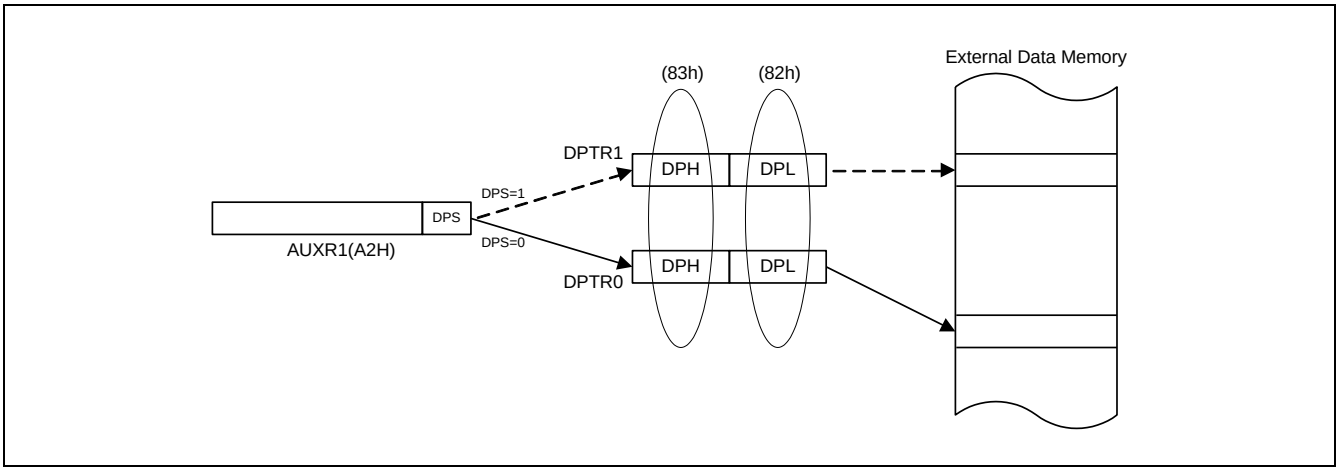
pdata
分页的外部数据(256 字节) 或片上的扩展 RAM（XRAM）：重叠的 256 字节的存储器地址通过“MOVX @Ri”指令访问。MA805-64 有 256 字节片上 pdata 存储器它与片上 xdata 存储器共享。

code
60K 程序存储空间。通过“MOVC @A+DTPR”访问，作为程序部分被读取。MA805-64 有 60K 字节的片上 code 存储器。

7. 双数据指针寄存器 (DPTR)

如图 8-1 所示的双 DPTR 结构是能让芯片指定外部数据存储器的定位地址的一种方法。有两个 16 位 DPTR 寄存器，和一个称作为 DPS(AUXR1.0)的控制位，允许在程序代码和外部存储器之间的切换。

图 8-1 双 DPTR



DPTR 指令

使用 DPS 位的六条指令参考 DPTR 的当前选择，如下：

```
INC     DPTR            ; 数据指针加 1
MOV     DPTR,#data16    ; DPTR 加载 16 位常量
MOVC    A，@A+DPTR      ; 将代码字节移动到 ACC
MOVX    A，@DPTR        ; 移动外部 RAM(16 位地址)到 ACC
MOVX    @DPTR，A        ; 移动 ACC 到外部 RAM(16 位地址)
JMP     @A+DPTR         ; 直接跳转到 DPTR
```

备注：在存取内部以及外部数据区切换，或是数据区超过内嵌的 1KB(0~0x3FF)数据区间切换，地址给定后，需要多加 NOP 确保资料读取无误。

AUXR1：辅助控制寄存 1

SFR 页 = 全部

SFR 地址 = 0xA2 复位值 = 0000-XXX0

7	6	5	4	3	2	1	0
P4KBI	P4PCA	P5SPI	P4S1	--	--	--	DPS
R/W	R/W	R/W	R/W	R	R	R	R/W

Bit 0: DPTR 选择位，用来在 DPTR0 和 DPTR1 之间切换

0: 选择 DPTR0.

1: 选择 DPTR1.

DPS	被选择的 DPTR
0	DPTR0
1	DPTR1

8. I/O 口

MA805-64 有下列 I/O 端口：P0.0~P0.7、P1.0~P1.7、P2.0~P2.7、P3.0~P3.7、P4.0~P4.6、P5.0~P5.3。 ALE 引脚

与 P4.6 有互换功能。如果选择内部振荡器做系统时钟输入则 XTAL2 和 XTAL1 被配置为 P6.0 和 P6.1。准确的可用 I/O 引脚数量由封装类型决定。见表 9-1。

表 9-1 可用 I/O 引脚数量

封装类型	I/O 口引脚	I/O 口引脚数量
40-pin PDIP*	P0.0~P0.7, P1.0~P1.7, P2.0~P2.7, P3.0~P3.7, P4.4, P4.5, P4.6, XTAL2(P6.0), XTAL1(P6.1)	35 或 37 (使用内部振荡器)
44-pin PQFP	P0.0~P0.7, P1.0~P1.7, P2.0~P2.7, P3.0~P3.7, P4.0~P4.6, XTAL2(P6.0), XTAL1(P6.1)	39 或 41 (使用内部振荡器)
48-pin LQFP	P0.0~P0.7, P1.0~P1.7, P2.0~P2.7, P3.0~P3.7, P4.0~P4.6, P5.0~P5.3, XTAL2(P6.0), XTAL1(P6.1)	43 或 45 (使用内部振荡器)

*: PDIP-40 快速开发验证功能用，不推荐批量生产

8.1. I/O 口结构

除 P6.0 和 P6.1 外，所有端口可通过软件个别的、独立的配置为四种之中的一种类型。这四种类型有：准双向口(标准 8051 的 I/O 端口)、上拉输出、集电极开漏输出和输入(高阻抗输入)。每个端口有两个模式寄存器来选择各端口引脚的输出类型。 P6.0 和 P6.1 只能为准双向口

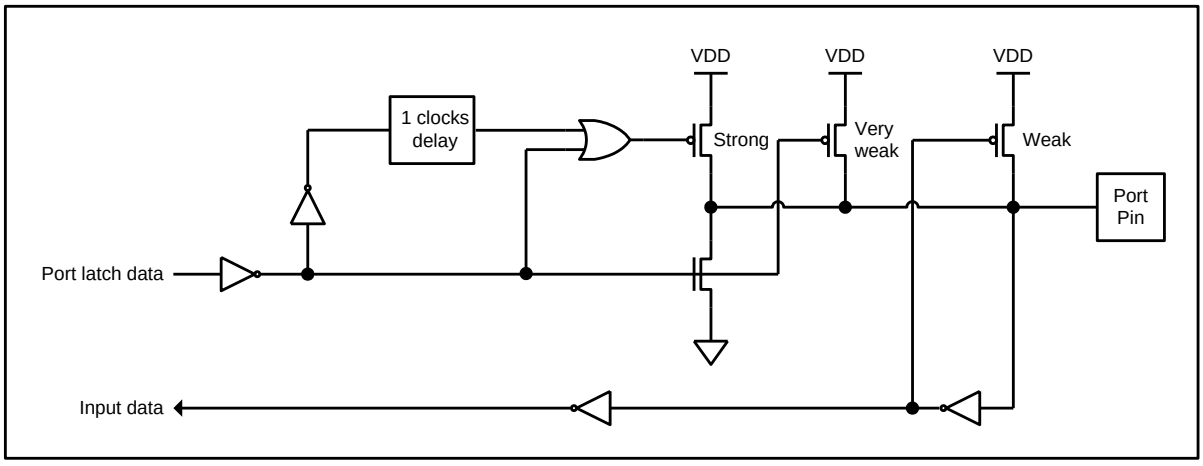
下面描述这四种类型的 I/O 模式的配置。

8.1.1. 准双向口

端口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。

准双向端口配置如图 9-1 所示。

图 9-1 准双向 I/O

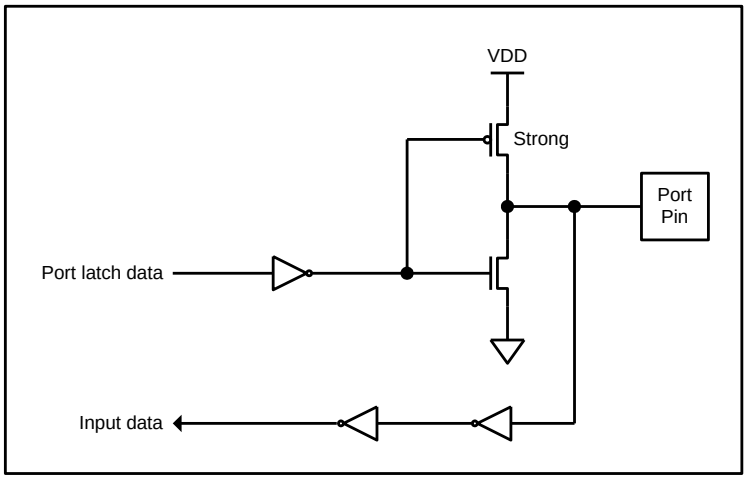


8.1.2. 推挽输出

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。

准双向端口配置如图 9-2 所示。

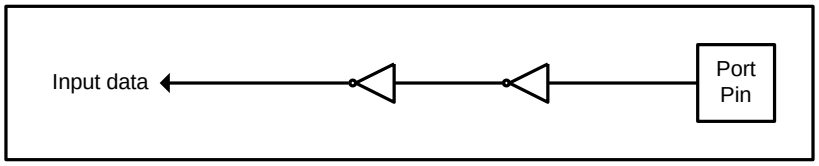
图 9-2 推挽输出



8.1.3. 仅输入模式（高阻抗输入）

仅输入配置在引脚上没有任何上拉电阻，如下图 9-3 所示。

图 9-3 仅输入

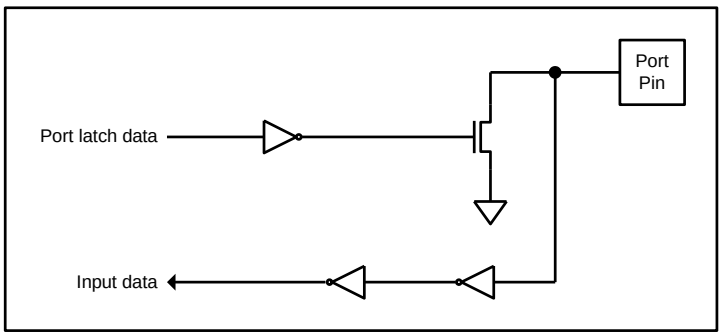


8.1.4. 开漏输出

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。

开漏输出端口配置如图 9-4 所示。

图 9-4 开漏输出



8.2. I/O 口寄存器

除 P6 外，MA805-64 的所有端口可通过软件个别的、独立的配置为四种之中的一种类型，基于位位基础，如表 9-2 所示。每个端口有两个模式寄存器来选择各端口引脚的输出类型。

表 9-2 端口配置设定

PxM0.y	PxM1.y	端口模式
0	0	准双向端口
0	1	推挽输出
1	0	仅输入(高阻抗输入)
1	1	集电极开漏输出

这里 x=0~4(端口号)，y=0~7(端口引脚号)。寄存器 PxM0 和 PxM1 列表如下。

8.2.1. 端口 0 寄存器

P0: 端口 0 数据寄存器

SFR 页 = 全部

SFR 地址 = 0x80 复位值 = 1111-1111

7	6	5	4	3	2	1	0
P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7~0: P0.7~P0.0 通过软件置位/清零

P0M0: 端口 0 模式寄存器 0

SFR 地址 = 0x93

SFR 页 = 全部 复位值 = 0000-0000

7	6	5	4	3	2	1	0
P0M0.7	P0M0.6	P0M0.5	P0M0.4	P0M0.3	P0M0.2	P0M0.1	P0M0.0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

P0M1: 端口 0 模式寄存器 1

SFR 页 = 全部

SFR 地址 = 0x94 复位值 = 0000-0000

7	6	5	4	3	2	1	0
P0M1.7	P0M1.6	P0M1.5	P0M1.4	P0M1.3	P0M1.2	P0M1.1	P0M1.0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

8.2.2. 端口 1 寄存器

P1: 端口 1 数据寄存器

SFR 页 = 全部

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 = 全部

SFR 地址 = 0x91 上电+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

P1M1: 端口 1 模式寄存器 1

SFR 页 = 全部

SFR 地址 = 0x92 上电+复位值 = 0000-0000

7	6	5	4	3	2	1	0
P1M1.7	P1M1.6	P1M1.5	P1M1.4	P1M1.3	P1M1.2	P1M1.1	P1M1.0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

8.2.3. 端口 2 寄存器

P2: 端口 2 数据寄存器

SFR 页 = 全部

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 = 全部

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

P2M1: 端口 2 模式寄存器 1

SFR 页 = 全部

SFR 地址 = 0x96 复位值 = 0000-0000

7	6	5	4	3	2	1	0
P2M1.7	P2M1.6	P2M1.5	P2M1.4	P2M1.3	P2M1.2	P2M1.1	P2M1.0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

8.2.4. 端口 3 寄存器

P3: 端口 3 数据寄存器

SFR 页 = 全部

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 通过软件置位/清零

P3M0: 端口 3 模式寄存器 0

SFR 地址 = 0xB1

SFR 页 = 全部 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 = 全部

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

8.2.5. 端口 4 寄存器

P4: 端口 4 数据寄存器

SFR 页 = 全部

SFR 地址 = 0xE8 复位值 = x111-1111

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

Bit 6~0: P4.6~P4.0 通过软件置位/清零

P4.6 复用作 ALE 引脚。当处理器执行片外存储器访问 MOVX 时，在 MOVX 周期中这引脚作 ALE 功能。

P4M0: 端口 4 模式寄存器 0

SFR 页 = 全部

SFR 地址 = 0xB3 复位值 = x000-0000

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

P4M1: 端口 4 模式寄存器 1

SFR 页 = 全部

SFR 地址 = 0xB4 复位值 = x000-0000

7	6	5	4	3	2	1	0
--	P4M1.6	P4M1.5	P4M1.4	P4M1.3	P4M1.2	P4M1.1	P4M1.0
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W

8.2.6. 端口 5 寄存器

P5: 端口 5 数据寄存器

SFR 页 = 全部

SFR 地址 = 0xF8 复位值 = xxxx-1111

7	6	5	4	3	2	1	0
--	--	--	--	P5.3	P5.2	P5.1	P5.0
R	R	R	R	R/W	R/W	R/W	R/W

Bit 3~0: P5.3~P5.0 通过软件置位/清零

P5M0: 端口 5 模式寄存器 0

SFR 页 = 全部

SFR 地址 = 0xB5 复位值 = xxxx-0000

7	6	5	4	3	2	1	0
--	--	--	--	P5M0.3	P5M0.2	P5M0.1	P5M0.0
R	R	R	R	R/W	R/W	R/W	R/W

P5M1: 端口 5 模式寄存器 1

SFR 页 = 全部

SFR 地址 = 0xB6 复位值 = xxxx-0000

7	6	5	4	3	2	1	0
--	--	--	--	P5M1.3	P5M1.2	P5M1.1	P5M1.0
R	R	R	R	R/W	R/W	R/W	R/W

8.2.7. 端口 6 寄存器

P6: 端口 6 数据寄存器

SFR 页 = **F** only

SFR 地址 = **0xC8** 复位值 = xxxx-xx11

7	6	5	4	3	2	1	0
--	--	--	--	--	--	P6.1	P6.0
R	**R**	**R**	**R**	**R**	**R**	**R/W**	**R/W**

Bit 7~2: 保留

Bit 1~0: P6.1~P6.0 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 P6.1 和 P6.0，它们仅仅支持准双向口模式。

8.3. GPIO 示例代码

(1). 功能需求: 设置P1.0 为仅输入模式

汇编语言代码范例:

```
P1Mn0           EQU             01h

    ORL     P1M0, #P1Mn0
    ANL     P1M1, #(0FFh + P1Mn0)          ; 配置 P1.0 为仅输入模式
    SETB    P1.0                           ; 设置 P1.0 数据为“1”而使能输入模式
```

C 语言代码范例:

```
#define     P1Mn0           0x01

    P1M0 |= P1Mn0;
    P1M1 &= ~P1Mn0;                        //配置 P1.0 为仅输入模式
    P10 = 1;                               //设置 P1.0 数据为“1”而使能输入模式
```

(2). 功能需求: 设置 P1.0 为推挽输出

汇编语言代码范例:

```
P1Mn0           EQU             01h

    ANL     P1M0, #(0FFh - P1Mn0)
    ORL     P1M1, #P1Mn0                   ; /配置 P1.0 为推挽输出
    SETB    P1.0
```

C 语言代码范例:

```
#define     P1Mn0           0x01

    P1M0 &= ~P1Mn0;
    P1M1 |= P1Mn0;                         //配置 P1.0 为推挽输出
    P10 = 1;                               // P10 输出高电平
```

(3). 功能需求: 设置 P1.0 为漏极开路输出模式

汇编语言代码范例:

```
P1Mn0           EQU             01h

    ORL     P1M0, #P1Mn0
    ORL     P1M1, #P1Mn0                ; 配置 P1.0 为漏极开路输出
    SETB    P1.0                        ; 设置 P1.0 数据为“1”而使能漏极开路输出模式
```

C 语言代码范例:

```
#define     P1Mn0           0x01

    P1M0 |= P1Mn0;
    P1M1 |= P1Mn0;                      //配置 P1.0 为漏极开路输出
    P10 = 1;                            //设置 P1.0 数据为“1”而使能漏极开路输出模式
```

9. 中断

MA805-64 有四级中断优先级的 14 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、EIP1H 及 XICON。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。

9.1. 中断结构

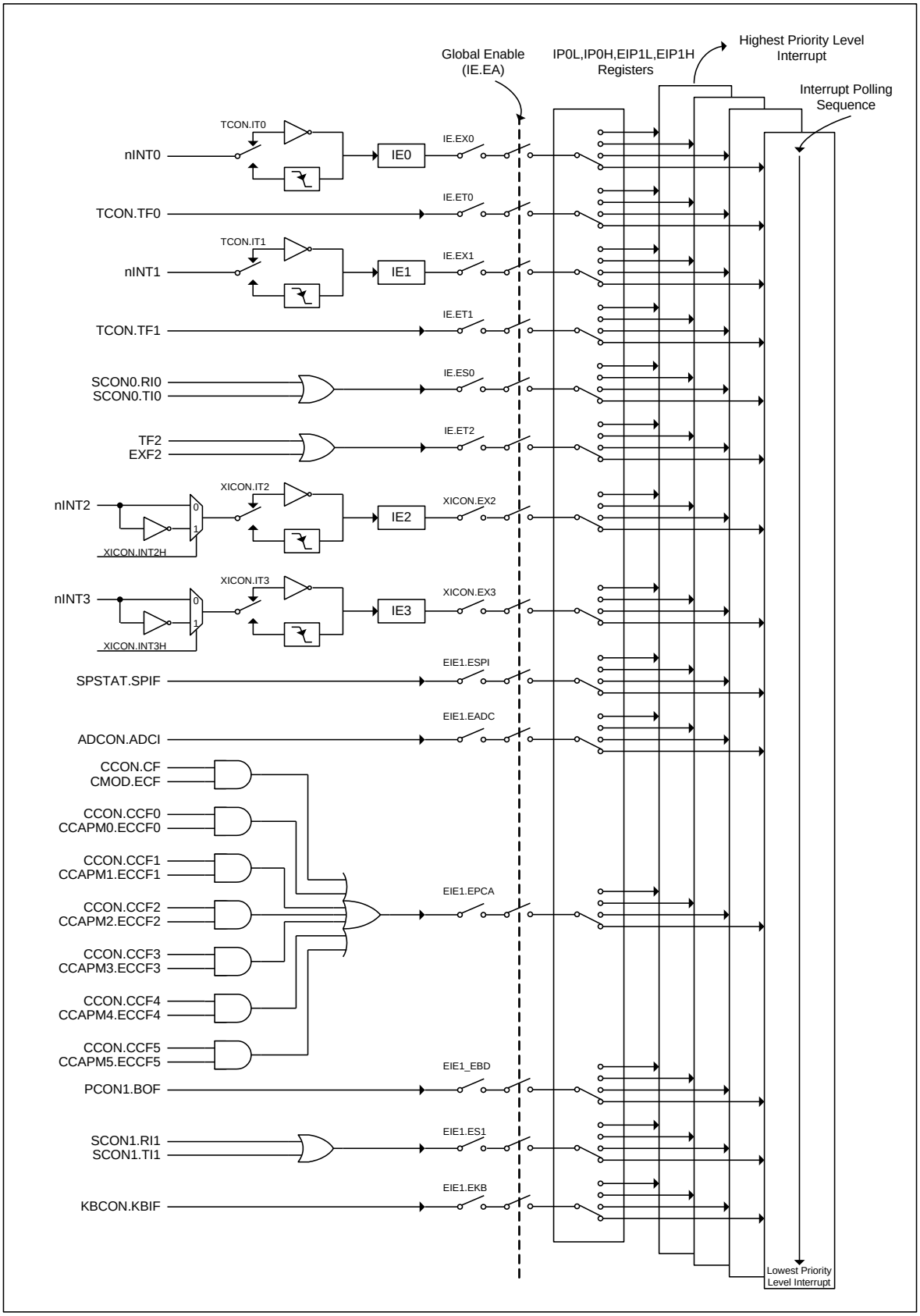
表 10-1 列出了所有的中断源。使能位被允许，中断请求时硬件会产生一个中断请求标志，当然，总中断使能位 EA（IE 寄存器)必须使能。中断请求位能由软件置 1 或清零，这和硬件置 1 或清零结果相同。同理，中断可以由软件产生或取消，中断优先级位决定每个中断产生的优先级，多个中断同时产生时依照中断优先级顺序处理。中断向量地址表示中断服务程序的入口地址。

图 10-1 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 10-1. 中断源

序号	中断名称	使能位	中断请求位	优先级位	优先级	中断向量地址	C51 向量
#1	外部中断 0（nINT0）	EX0	IE0	PX0H, PX0L	(高优先)	0003H	0
#2	定时器 0	ET0	TF0	PT0H, PT0L	…	000Bh	1
#3	外部中断 1（nINT1）	EX1	IE1	PX1H, PX1L	…	0013H	2
#4	定时器 1	ET1	TF1	PT1H, PT1L	…	001BH	3
#5	串口 0	ES0	RI0, TI0	PS0H, PS0L	…	0023H	4
#6	定时器 2	ET2	TF2, EXF2	PT2H, PT2L	…	002Bh	5
#7	外部中断 2（nINT2）	EX2	IE2	PX2H, PX2L	…	0033H	6
#8	外部中断 3（nINT3）	EX3	IE3	PX3H, PX3L	…	003BH	7
#9	SPI	ESPI	SPIF	PSPIH, PSPIL	…	0043H	8
#10	ADC	EADC	ADCI	PADCH, PADCL	…	004Bh	9
#11	PCA	EPCA	CF, CCFn (n=0~5)	PPCAH, PPCAL	…	0053H	10
#12	掉电检测器（Brownout Detection）	EBD	BOF	PBDH, PBDL	…	005BH	11
#13	串口 1	ES1	RI1, TI1	PS1H, PS1L	…	0063H	12
#14	键盘中断	EKB	KBIF	PKBH, PKBL	(低优先)	006BH	13

图 10-1 中断系统


Global Enable
(IE.EA)
IP0L,IP0H,EIP1L,EIP1H
Registers
Highest Priority Level
Interrupt
Interrupt Polling
Sequence
TCON.IT0
nINT0
IE0
IE.EX0
IE.ET0
TCON.TF0
TCON.IT1
nINT1
IE1
IE.EX1
IE.ET1
TCON.TF1
SCON0.RI0
SCON0.TI0
IE.ES0
TF2
EXF2
IE.ET2
XICON.IT2
nINT2
XICON.INT2H
IE2
XICON.EX2
XICON.IT3
nINT3
XICON.INT3H
IE3
XICON.EX3
SPSTAT.SPIF
EIE1.ESPI
ADCON.ADCI
EIE1.EADC
CCON.CF
CMOD.ECF
CCON.CCF0
CCAPM0.ECCF0
CCON.CCF1
CCAPM1.ECCF1
CCON.CCF2
CCAPM2.ECCF2
CCON.CCF3
CCAPM3.ECCF3
CCON.CCF4
CCAPM4.ECCF4
CCON.CCF5
CCAPM5.ECCF5
EIE1.EPCA
PCON1.BOF
EIE1_EBD
SCON1.RI1
SCON1.TI1
EIE1.ES1
KBCON.KBIF
EIE1.EKB
Lowest Priority
Level Interrupt

9.2. 中断寄存器

IE: 中断使能寄存器

SFR 页 = 全部

SFR 地址 = 0xA8 复位值 = 0X00-0000

7	6	5	4	3	2	1	0
EA	--	ET2	ES0	ET1	EX1	ET0	EX0
R/W	R	R/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: 保留。当对 IE 进行写的时候必须对此位写“0”

Bit 5: ET2，定时器 2 中断使能
0: 禁止定时器 2 中断
1: 使能定时器 2 中断

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

XICON: 外部中断控制寄存器

SFR 页 = 全部

SFR 地址 = 0xC0 复位值 = 0000-0000

7	6	5	4	3	2	1	0
INT3H	EX3	IE3	IT3	INT2H	EX2	IE2	IT2
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7: INT3H, nINT3 高电平/上升沿触发使能
0: P4.2 上的低电平或下降沿触发 nINT3
1: P4.2 上的高电平或上升沿触发 nINT3

Bit 6: EX3, 外部中断 3 使能

0: 禁止外部中断 3
1: 使能外部中断 3

Bit 5: IE3, 外部中断 3 请求标志
0: 当中断被响应后此位被硬件清零。也可以通过软件清零。
1: 当外部中断被侦察到后此位被硬件置位。也可以通过软件置位。

Bit 4: IT3, 外部中断 3 类型控制位
0: 软件选择低电平触发。如果 INT3H 为 1，则为高电平触发
1: 软件选择下降沿触发。如果 INT3H 为 1，则为上升沿触发

Bit 3: INT2H, nINT2 高电平/上升沿触发使能
0: P4.3 上的低电平或下降沿触发 nINT2
1: P4.3 上的高电平或上升沿触发 nINT2

Bit 2: EX2, 外部中断 2 使能
0: 禁止外部中断 2
1: 使能外部中断 2

Bit 1: IE2, 外部中断 2 请求标志
0: 当中断被响应后此位被硬件清零。也可以通过软件清零。
1: 当外部中断被侦察到后此位被硬件置位。也可以通过软件置位。

Bit 0: IT2, 外部中断 2 类型控制位
0: 软件选择低电平触发。如果 INT2H 为 1，则为高电平触发
1: 软件选择下降沿触发。如果 INT3H 为 1，则为上升沿触发

EIE1：扩展中断使能寄存器

SFR 页　　= 全部
SFR 地址　　= 0xAD　　复位值 = XX00-0000

7	6	5	4	3	2	1	0
--	--	EKBI	ES1	EBD	EPCA	EADC	ESPI
R	R	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7~6: 保留。当对 IEI1 进行写的时候必须对这些位写“0”

Bit 5: EKBI, 键盘中断使能
0: 当键盘控制模块里的 KBCON.KBIF 为 1 时禁止中断
1: 当键盘控制模块里的 KBCON.KBIF 为 1 时使能中断

Bit 4: ES1,串口 1(UART1)中断使能
0: 禁止串口 1 中断
1: 使能串口 1 中断

Bit 3: EBD,（Brown-out）掉电检测中断使能
0: 当掉电检测模块的 PCON1.BOF 为 1 时禁止中断
1: 当掉电检测模块的 PCON1.BOF 为 1 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: EACI, ADC 中断使能

0: 当模数转换器（ADC）模块的 ADCON.ADCI 为 1 时禁止中断
1: 当模数转换器（ADC）模块的 ADCON.ADCI 为 1 时使能中断

Bit 0: ESPI, SPI 中断使能
0: 当 SPI 模块的 SPSTAT.SPIF 为 1 时禁止中断
1: 当 SPI 模块的 SPSTAT.SPIF 为 1 时使能中断

IP0L：中断优先级 0 寄存器低

SFR 页　　= 全部
SFR 地址　　= 0xB8　　复位值 = 0000-0000

7	6	5	4	3	2	1	0
PX3L	**PX2L**	PT2L	PSL	PT1L	PX1L	PT0L	PX0L
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: PX3L, 外部中断 3 优先级低位
Bit 6: PX2L, 外部中断 2 优先级低位
Bit 5: PT2L, 定时器 2 中断优先级低位
Bit 4: PSL, 串口中断优先级低位
Bit 3: PT1L, 定时器 1 中断优先级低位
Bit 2: PX1L, 外部中断 1 优先级低位
Bit 1: PT0L, 定时器 0 中断优先级低位
Bit 0: PX0L, 外部中断 0 优先级低位

IP0H：中断优先级 0 寄存器高

SFR 页　　= 全部
SFR 地址　　= 0xB7　　复位值 = 0000-0000

7	6	5	4	3	2	1	0
PX3H	PX2H	PT2H	PSH	PT1H	PX1H	PT0H	PX0H
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: PX3H, 外部中断 3 优先级高位
Bit 6: PX2H, 外部中断 2 优先级高位
Bit 5: PT2H, 定时器 2 中断优先级高位
Bit 4: PSH, 串口中断优先级高位
Bit 3: PT1H, 定时器 1 中断优先级高位
Bit 2: PX1H, 外部中断 1 优先级高位
Bit 1: PT0H, 定时器 0 中断优先级高位
Bit 0: PX0H, 外部中断 0 优先级高位

EIP1L：扩展中断优先级 1 寄存器低

SFR 页　　= 全部
SFR 地址　　= 0xAE　　复位值 = XX00-0000

7	6	5	4	3	2	1	0
--	--	PKBL	PS1L	PBDL	PPCAL	PADCL	PSPIL
R	**R**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~6: 保留。当对 EIP1L 进行写的时候必须对这些位写“0”
Bit 5: PKBL, 键盘中断优先级低位
Bit 4: PS1L, UART1 中断优先级低位
Bit 3: PBDL,掉电检测中断优先级低位
Bit 2: PPCAL, PCA 中断优先级低位

Bit 1: PADCL, ADC 中断优先级低位
Bit 0: PSPIL, SPI 中断优先级低位

EIP1H: 扩展中断优先级 1 寄存器高

SFR 页 = 全部
SFR 地址 = 0xAF 复位值 XX00-0000

7	6	5	4	3	2	1	0
--	--	PKBH	PS1H	PBDH	PPCAH	PADCH	PSPIH
R	R	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7~6: 保留。当对 EIP1H 进行写的时候必须对这些位写“0”
Bit 5: PKBH, 键盘中断优先级高位
Bit 4: PS1H, UART1 中断优先级高位
Bit 3: PBDH, 掉电检测器中断优先级高位
Bit 2: PPCAH, PCA 中断优先级高位
Bit 1: PADCH, ADC 中断优先级高位
Bit 0: PSPIH, SPI 中断优先级高位

IP0L, IP0H, EIP1L 和 EIP1H 组合成四级优先级中断如下表。

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

MA805-64 有 14 个可用的中断源，通过设置寄存器 IE、EIE1 和 XICON 能对每个中断进行使能和禁止操作。需注意 IE 中有个总中断允许位 EA，EA 置‘1’则使能全局中断（使能和禁止由各自单独的设置位决定）。EA 清‘0’，所有中断被禁止。

每一个中断有两个相应位表现它的优先级。一个是 IPxH 寄存器，另外一个是 IPxL 寄存器。如果两个优先级的中断同时产生，高优先级的中断总是优先处理。低优先级的中断在处理过程中可以被高优先级的中断打断，等高优先级的中断处理完后低优先级的中断才能从被打断处继续执行。同等优先级的中断同时产生，执行顺序由中断向量决定顺序执行，中断向量越小的优先级越高（见下表）。

中断源	中断向量地址	C51 中断向量	中断优先级
外部中断 0（nINT0）	0003H	0	1 (优先级高)
定时器 0	000BH	1	2
外部中断 1（nINT1）	0013H	2	3
定时器 1	001BH	3	4
串口 0	0023H	4	5
定时器 2	002BH	5	6
外部中断 2（nINT2）	0033H	6	7
外部中断 3（nINT3）	003BH	7	8
SPI	0043H	8	9
ADC	004BH	9	10
PCA	0053H	10	11
掉电检测器（Brownout Detection）	005BH	11	12
串口 1	0063H	12	13
键盘中断	006BH	13	14（优先级低）

外部中断 nINT0、nINT1、nINT2 和 nINT3 分别通过 TCON 的 IT0、IT1、XICON 的 IT2、IT3 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，IE1,XICON 的 IE2 和 IE3。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(分别由各自的定时/计数寄存器控制，定时器 0 工作在模式 3 时除外)产生。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 0 中断由 RI 和 TI 的逻辑或产生。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。

定时器 2 中断由 TF2 和 EXF2 的逻辑或产生。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 TF2 还是 EXF2 产生的中断，然后由软件清除中断标志。

SPI 串口中断

ADC 由 ADCON.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 寄存器中的 CF, CCF5, CCF4, CCF3, CCF2, CCF1 及 CCF0 的逻辑或产生。进入中断服务程序后，这些标志均不会被硬件清除。中断服务程序通常需要查询这些位以确定中断源，然后由软件清除中断标志。

BOD 中断由 PCON1 寄存器的 BOD 位产生，当掉电检测器(Brownout-Detector)侦查到低电压事件后硬件置位此位。进入中断服务程序后，这标志不会被硬件清除。

串口 1 中断由 RI1 和 TI1 的逻辑或产生。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 RI1 还是 TI1 产生的中断，然后由软件清除中断标志。

键盘中断由 KBCON.KBIF 产生。进入中断服务程序后，这标志均不会被硬件清除。

所有这些产生中断的位都可通过软件置位或清零,与通过硬件置位或清零的效果相同。简而言之，中断可由软件产生，推迟或取消。

硬件如何响应中断

每个机器周期都会采样中断标志位。如果没有下列阻止条件，前一个指令周期中产生中断标志位置位，接下来的指令周期中将以 LCALL 调用中断服务程序，下列几种情况将 LCALL 指令锁定：

阻止条件：

1. 另一个高优先级的中断正在处理;
2. 当前机器周期不是正在执行的指令的最后一个机器周期;
3. 正在执行指令 RETI 或正在写和中断相关的寄存器(IE, IP0L, IPH, XICON, 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。

每个机器周期硬件都会重复查询中断请求标志，查询到的值都是前一个系统时钟的值。值得注意的是如果一个被触发的中断标志由于上面的阻止条件发生了而没得到响应，并且此标志没有持续到阻止条件消失，这样的中断请求将得不到响应。也就是说，中断处理本身不能锁存中断，譬如外部电平中断若在电平出现时被屏蔽，而在中断识别之前电平消失，它被完全忽略。也就是每个查询都是新状态。

9.3. 中断示例代码

(1). 功能需求: 在掉电模式下设置 INT0 高电平唤醒 MCU

汇编语言代码范例:

```
PX0             EQU         01h
PX0H                        EQU         01h
PD              EQU         02h

    ORG     0000h
    JMP     main

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

main:

    SETB    P3.2                        ;

    ORL     IP,#PX0                     ; 选择 INT0 中断优先级
    ORL     IPH,#PX0H                   ;

    JNB     P3.2, $                     ; 确认 P3.2 输入高

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

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

    JMP     $
```

C 语言代码范例:

```
#define     PX0             0x01
#define     PX0H                        0x01
#define     PD              0x02
```

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

void main(void)
{
    P32 = 1;

    IP |= PX0;                                  //选择 INT0 中断优先级
    IPH |= PX0H;

    while(!P32);                                //确认 P3.2 输入高

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

    PCON |= PD;                                 //设置 MCU 进入掉电模式

    while(1);
```

10. 定时器/计数器

MA805-64 有三个 16 位的定时器/计数器：定时器 0、定时器 1 和定时器 2。每一个包含两个 8 位寄存器 THx 和 TLx(这里，x=0、1 或 2)。它们可配置为定时器或事件计数器。

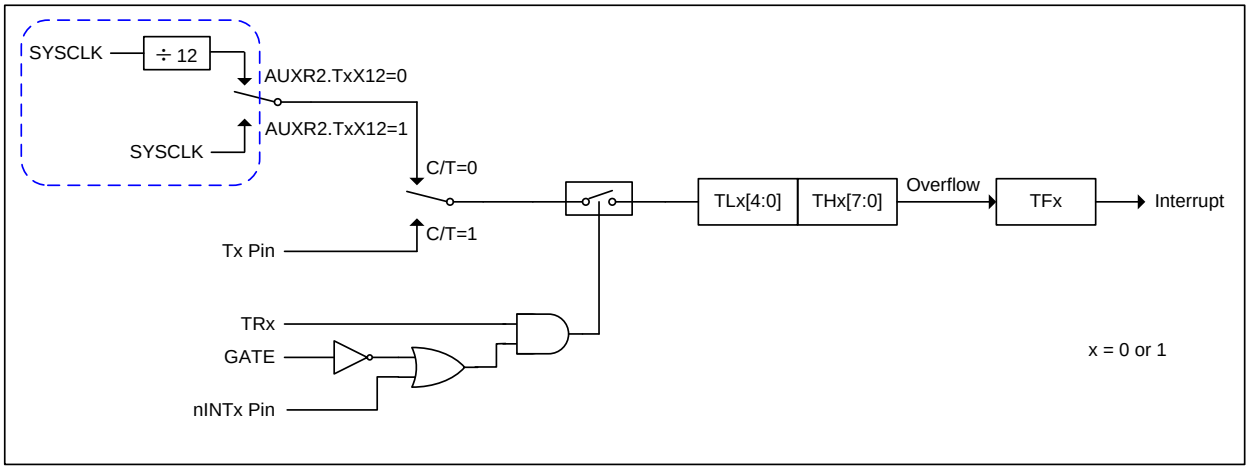
定时器功能，TLx 寄存器每 12 个系统时钟周期（标准 C51 的机器周期）或每 1 个系统时钟周期（是标准 C51 的 12 倍）加 1，通过软件设置 AUXR2.T0X12、 AUXR2.T1X12 和 T2MOD.T2X12 位来选择 。每 12 个系统时钟周期加一，计数速率是 1/12 的晶振频率。

计数器功能，根据对应的外部输入引脚的下降沿 T0、T1 或 T2 寄存器加 1。在这功能中，每个定时器时钟周期对外部输入信号(T0、T1 和 T2 引脚)进行采样，当采样信号出现一个高电平接着一个低电平，计数加 1。当检测到跳变时新计数值出现在寄存器中。

10.1. 定时器 0 和定时器 1

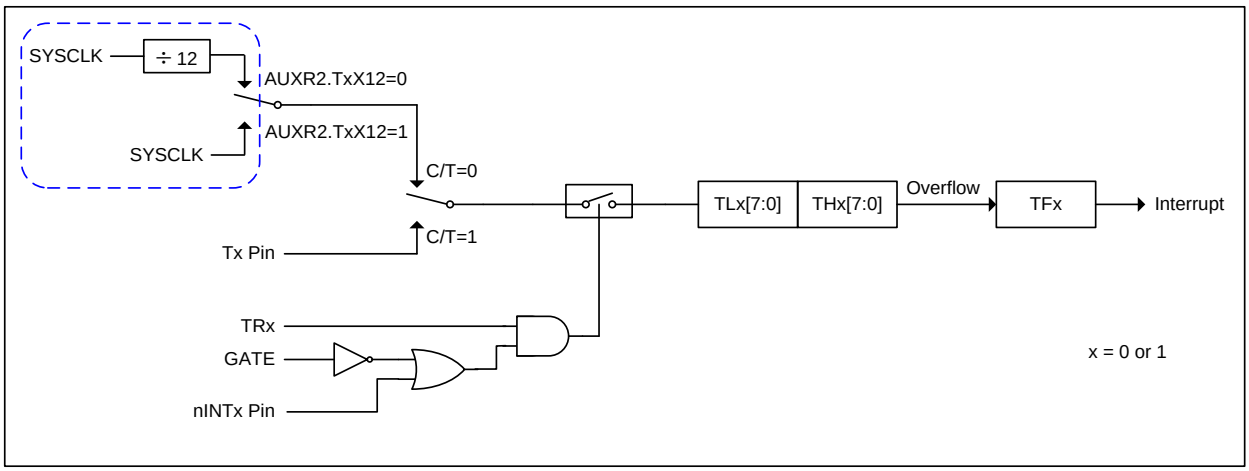
10.1.1. 模式 0

在这个模式，定时器寄存器配置为一个 13 位寄存器。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。
13 位寄存器包含 THx 的所有 8 位和 TLx 的低 5 位。TLx 的高 3 位是不确定的可以忽略。置位运行标志(TRx)不会清除寄存器。意思是说用户在开始计数前应对 THx 和 TLx 进行初始化。



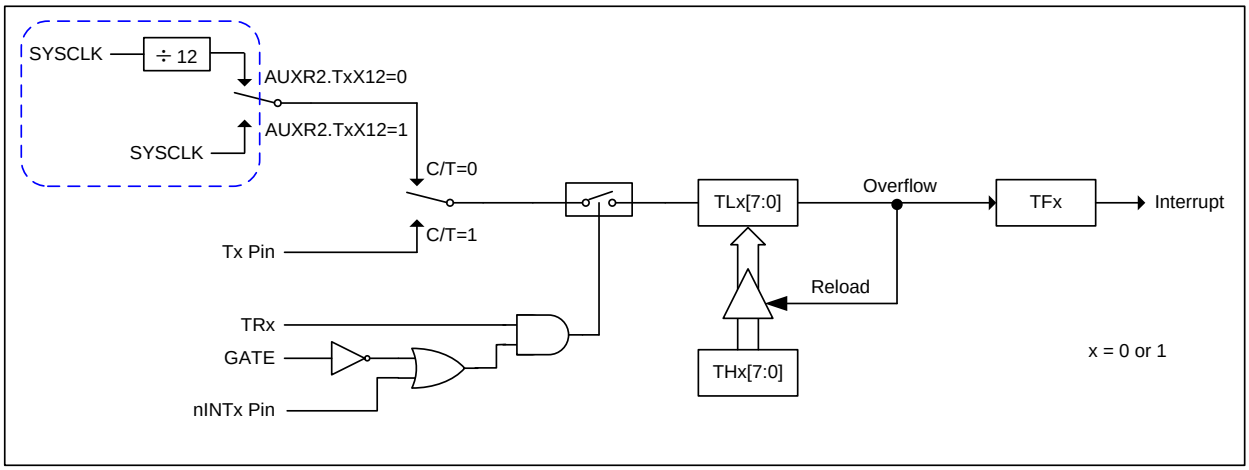
10.1.2. 模式 1

除了定时器的寄存器使用全部 16 位外，模式 1 和模式 0 是相同的。在这个模式，THx 和 TLx 串联，没有预分频。



10.1.3. 模式 2

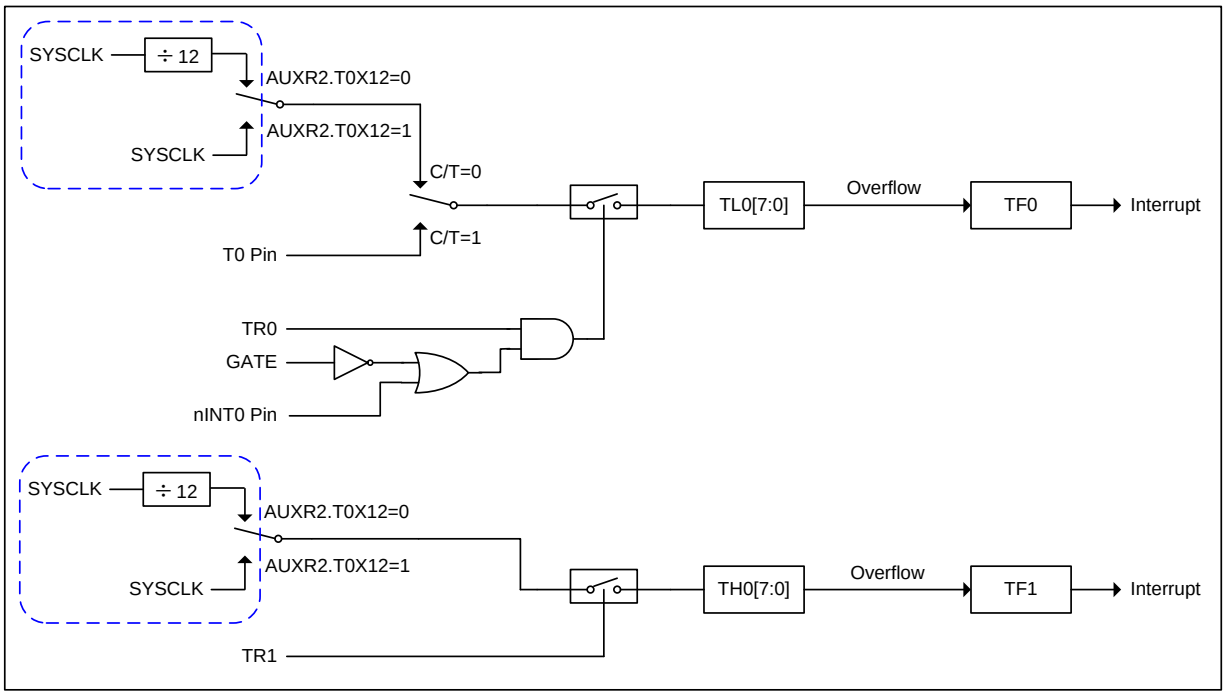
模式 2 配置定时器寄存器为一个自动加载的 8 位计数器(TLx)。TLx 溢出不仅置位 TFx，而且也将 THx 的内容加载到 TLx，THx 内容由软件预置，加载不会改变 THx 的值。定时器 0 和定时器 1 的模式 2 运作时相同的。



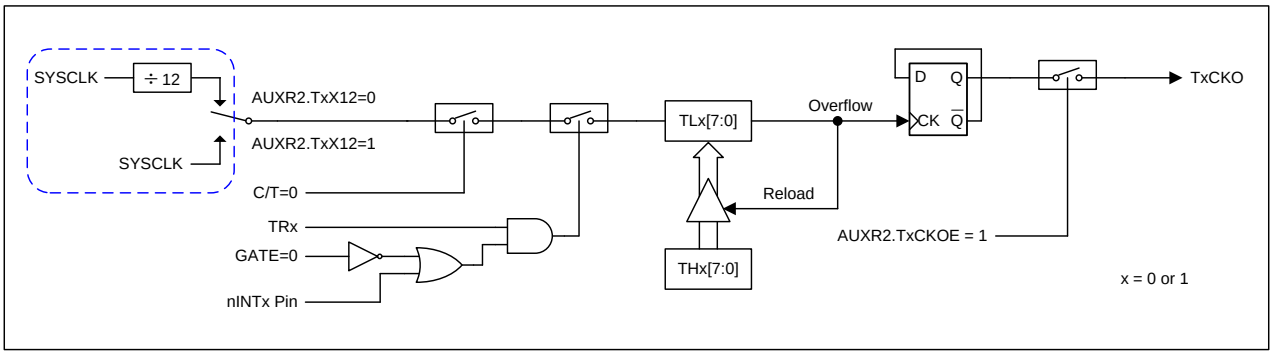
10.1.4. 模式 3

定时器 1 在模式 3 保持计数值。效果和设置 TR1=0 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)且接替定时器 1 来使用 TR1 和 TF1，TH0 控制定时器 1 中断。



10.1.5. 定时器时钟输出



$$\text{T0/T1 Clock-out Frequency} = \frac{\text{SYSCLK Frequency}}{n \times (256 - \text{THx})}$$

; n=24, if TxX12=0
; n=2, if TxX12=1
; x = 0 or 1 & C/T = 0

10.1.6. 定时器 0/1 寄存器

TMOD: 定时器/计数器模式控制寄存器

SFR 页 =全部

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
←---------- 定时器 1 ----------→				←---------- 定时器 0 ----------→			

Bit 7/3: Gate, 门控位

0: 禁止门控位。当门控制位清零时，只要 TR0 或 TR1 置 1 则定时器 0 或 1 使能。

1: 使能门控位。当门控位置位时，只有在 nINT0 或 nINT1 引脚是高电平且 TR0 或 TR1 控制位置位时定时器/计数器 0 或 1 使能。

Bit 6/2: C/T, 定时器或计数器功能选择位。

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 T0 或 T1 引脚输入)。

Bit 5~4/1~0: 工作模式选择

M1	M0	工作模式
0	0	13 位定时器/计数器
0	1	16 位定时器/计数器。THx 与 TLx 串联，没有分频器
1	0	8 位自动重载定时器/计数器。THx 保持一个值，并在每次溢出时加载到 TLx
1	1	(定时器 0)TL0 是一个 8 位定时器/计数器并通过标准定时器 0 的控制位控制。TH0 仅仅是一个 8 位定时器通过定时器 1 的控制位控制
1	1	(定时器 1)定时器/计数器停止

TCON: 定时器/计数器控制寄存器

SFR 页 =全部

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: 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: 软件置位开启定时器/计数器 0。

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 1。
1: 软件选择下降沿触发外部中断 1。

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。
1: 软件选择下降沿触发外部中断 0。

TL0: 定时器 0 低

SFR 页 =全部
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]	TL0[2]	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

SFR 页 =全部
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

SFR 页 =全部
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 1 高

SFR 页 =全部
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 =全部
SFR 地址 = 0xA6 复位值 = 00XX-XX00

7	6	5	4	3	2	1	0

T0X12	T1X12	--	--	--	--	T1CKOE	T0CKOE
R/W	R/W	R	R	R	R	R/W	R/W

Bit 7: T0X12，当 C/T=0 时，定时器 0 的时钟源选择。
0：清零选择 SYSCLK/12 作为时钟源。
1：置位选择 SYSCLK 作为时钟源。

Bit 6: T1X12，当 C/T=0 时，定时器 1 的时钟源选择。
0：清零选择 SYSCLK/12 作为时钟源。
1：置位选择 SYSCLK 作为时钟源。

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 时钟从 P3.5 输出

Bit 0: T0CKOE，定时器 0 时钟输出使能
0：禁止定时器 1 时钟输出
1：使能定时器 1 时钟从 P3.4 输出

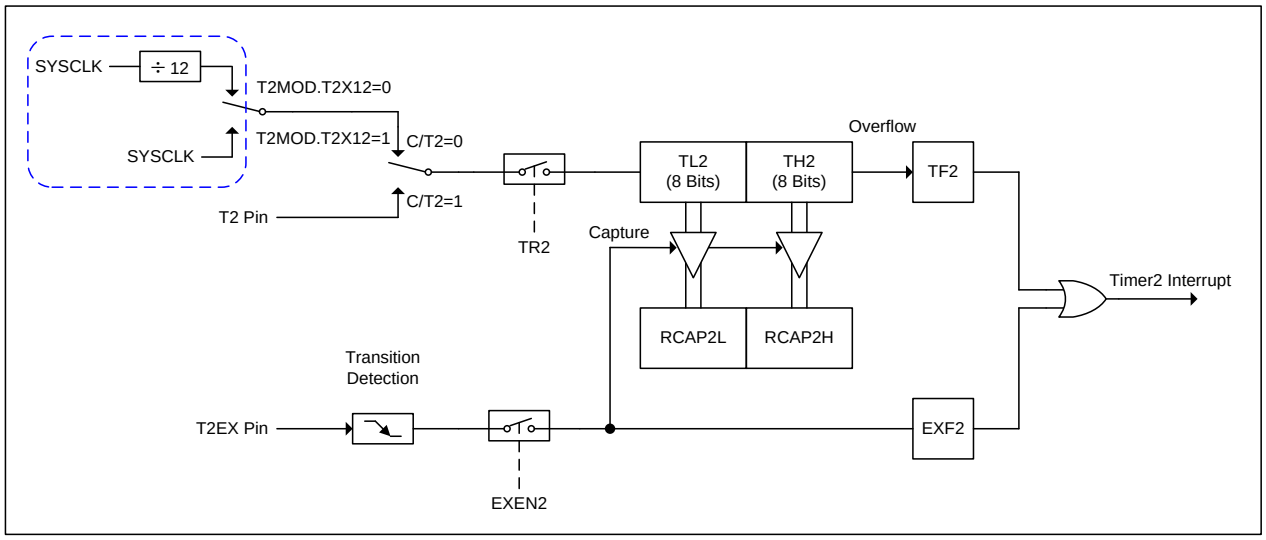
10.2. 定时器 2

定时器 2 是一个 16 位定时器/计数器，既可作为一个定时器也可以作为一个事件计数器，通过专用寄存器 T2CON 的 C/T2 位来选择。定时器 2 有四种工作模式：捕获、自动加载(向上或向下计数)、波特率发生器和可编程时钟输出，通过专用寄存器 T2CON 和 T2MOD 来选择。

10.2.1. 捕获模式(CP)

在捕获模式，有两个选项通过 T2CON 中的 EXEN2 位来选择。如果 EXEN2=0，定时器 2 做为一个 16 位的定时器或计数器，向上溢出，定时器 2 溢出时 TF2 置位。这位可以用来产生中断(通过使能 IE 寄存器中的定时器 2 中断位)。如果 EXEN2=1，定时器 2 仍然向上， 当外部输入信号 T2EX 由下降沿跳变时引起定时器 2 的寄存器 TH2 和 TL2 分别对应的捕获到 RCAP2H 和 RCAP2L。另外，T2EX 的跳变引起 T2CON 的 EXF2 置位，且 EXF2 位(象 TF2)将产生一个中断(中断向量的位置和定时器 2 溢出中断位置相同)。捕获模式图解如下图。(在这个模式 TL2 和 TH2 没有加载值。直到从 T2EX 捕获事件发生，在 T2EX 引脚跳变或 Fosc/12 的脉冲产生时计数器仍然保持计数)。

图 11-5 定时器 2 捕获模式



10.2.2. 自动加载模式（AR）

在 16 位自动加载模式，定时器既可配置成定时器也可以配置成计数器(C/T2 在 T2CON 寄存器)，接着编程向上或向下计数。计数方向由 T2MOD 寄存器的 DCEN 位来决定(向下计数使能)。在复位之后，DCEN=0 意思是默认为定时器 2 向上计数。如果 DCEN 置位，定时器 2 向上或向下计数由 T2EX 引脚的值来决定。

图 11-6 示 DCEN=0，自动使能定时器 2 向上计数。这个模式有两个选项可以通过 T2CON 寄存器的 EXEN2 位来选择。如果 EXEN2=0，定时器向上计数 0XFFFF 接着计数将置位 TF2(溢出标志位)。这将引起定时器 2 的寄存器将 RCAP2L 和 RCAP2H 的值加载。RCAP2L 和 RCAP2H 的值由软件预置。如果 EXEN2=1，一个溢出或在输入 T2EX 的一个负跳变将触发加载 16 位值。跳变将置位 EXF2 位。当 TF2 或 EXF2 置 1 时，如果定时器 2 中断使能，将产生中断。

图 11-6 定时器 2 自动加载模式 (DCEN=0)

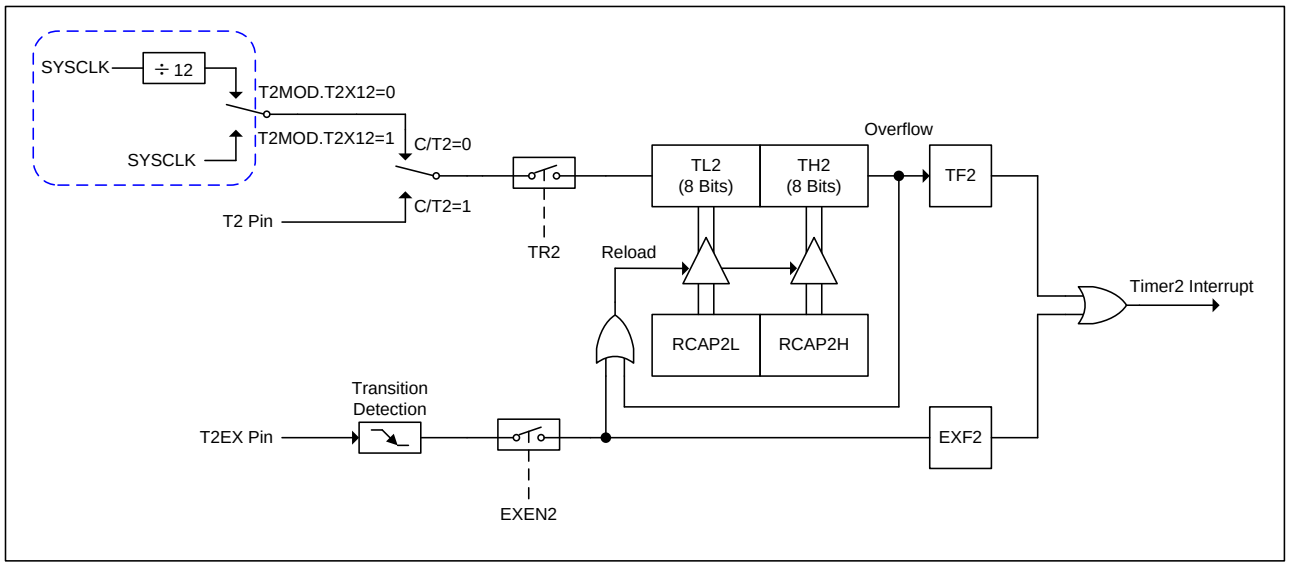
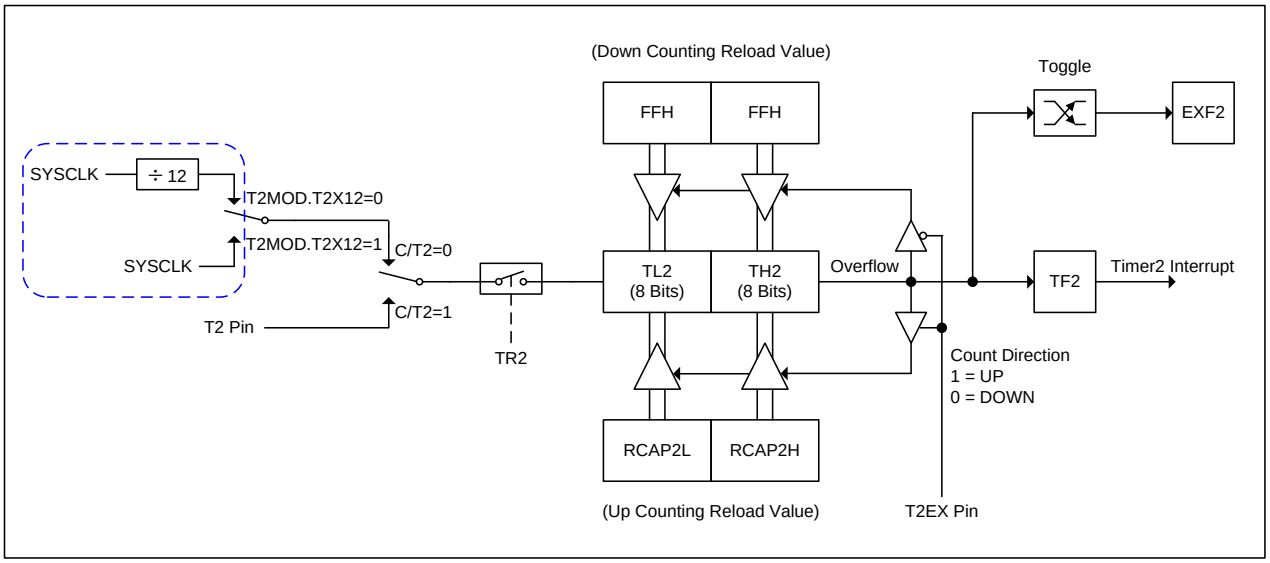


图 12-7 示 DCEN=1，使能定时器 2 向上或向下计数。这种模式下允许 T2EX 引脚控制计数方向。当 T2EX 的引脚为逻辑 1 时定时器 2 向上计数。定时器 2 在 0FFFFH 时溢出并置位 TF2 标志位，如果中断使能将产生中断。溢出也将引起 RCAP2L 和 RCAP2H 的 16 位值加载到定时器的寄存器 TL2 和 TH2。当 T2EX 的引脚为逻辑 0 时定时器 2 向下计数。当 TL2 和 TH2 和存储在 RCAP2L 和 RCAP2H 的值相等时将产生下溢。下溢将置位 TF2 标志位并将 0FFFFH 加载到定时器的寄存器 TL2 和 TH2。

当定时器 2 下溢或上溢时外部标志位 EXF2 将被触发。如果需要 EXF2 可作为 17 位分辨率。EXF2 标志位在这个模式下不会产生中断。

图 11-7 定时器 2 自动加载模式(DCEN=1)



10.2.3. 波特率发生器模式（BRG）

T2CON 寄存器的 RCLK 和/或 RCLK 位允许串行口发送和接收波特率既可源自定时器 1 或定时器 2。当 TCLK=0 时，定时器 1 作为串行口传送波特率发生器。当 TCLK=1，定时器 2 作为串行口传送波特率发生器。RCLK 对串行口接收波特率有相同的功能。有了这两位，串行口可能有不同的接收和发送波特率，一个通过定时器 1 来产生，另一个通过定时器 2 来产生。

图 11-8 所示定时器 2 在波特率发生器模式为 UART 引擎产生 RX 和 TX 时钟（见图 12 6 ）。波特率发生器模式像自动加载模式，翻转时将把寄存器 RCAP2H 和 RCAP2L 的值加载到定时器 2 的寄存器，RCAP2H 和 RCAP2L 的值由软件预置。

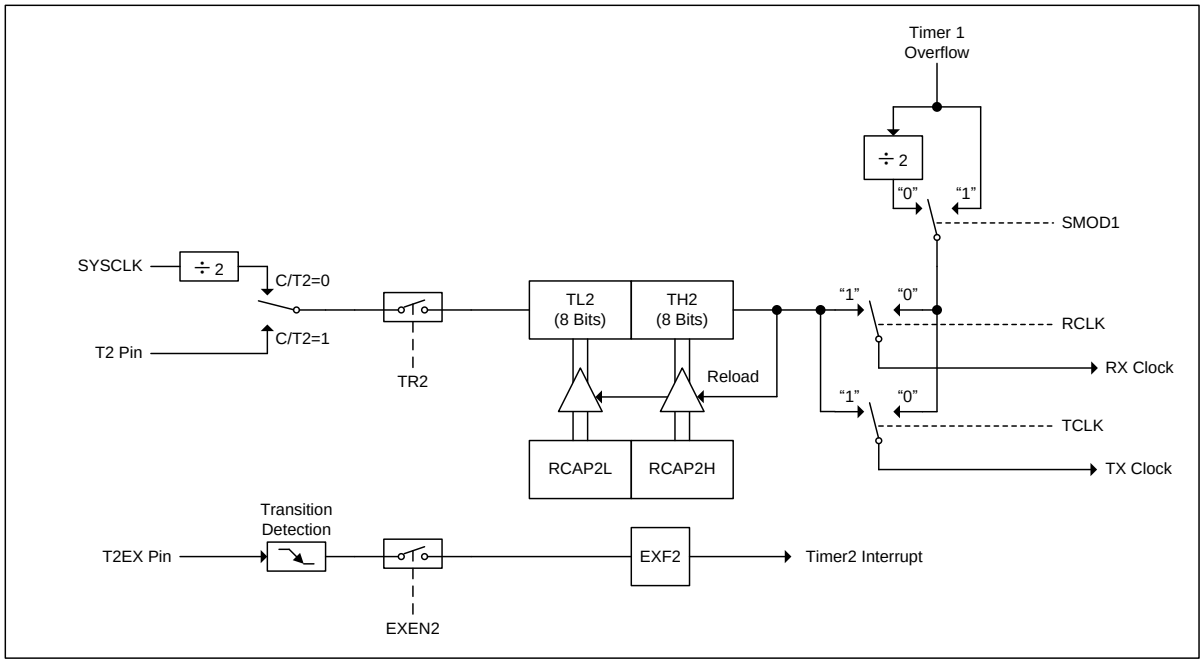
定时器 2 作为一个波特率发生器模式，只有在 T2CON 寄存器的位 RCLK 和/或 TCLK=1 为 1 时有效。注意 TH2 翻转不会置位 TF2，也不会产生中断。因而，当定进器 2 在波特率发生器模式时定时器中断不需要禁止。如果 EXEN2(T2 外部中断使能位)置位，T2EX(定时器/计数器 2 触发输入)的负跳变将置位 EXF2(T2 外部标志位)，但是不会引起从(RCAP2H，RCAP2L)到(TH2，TL2 的)重载。因此，当定时器 2 作为波特率发生器时，如果需要的话，T2EX 也可以作为传统的外部中断。

当定时器 2 在波特率发生器模式时，不能试着去读 TH2 和 TL2。作为一个波特率发生器，定时器 2 在 1/2 的系统时钟频率或从 T2 引脚的异步时增 1；在这些条件下，读写操作将会不正确。寄存器 RCAP2 可以读，但是不可以写，因为写和重载重叠并引起写和/或加载错误。在进入定时器 2 或 RCAP2 寄存器时定时器不可以关闭(清零 TR2)。

注：

当用定时器 2 作为波特率发生器参考 12.7.3 模式 1 和 3 的波特率速率设置。

图 11-8 定时器 2 波特率发生器模式



10.2.4. 定时器 2 的可编程时钟输出模式

定时器 2 有时钟输出模式(当 CP/RL2=0 且 T2OE=1)。此模式，定时器 2 作为一个输出占空比为 50%的可编程时钟发生器，所产生的时钟从 P1.0 引脚输出来。输出频率由系统时钟频率(SYSCLK)和在 RCAP2H、RCAP2L 寄存器的加载值来决定，如下公式：

$$\text{T2 Clock-out Frequency} = \frac{\text{SYSCLK Frequency}}{4 \times (65536 - (\text{RCAP2H, RCAP2L}))}$$

这里[RCAP2H,RCAP2L]=RCAP2H、RCAP2L 内容产生的一个 16 位无符号数。

定时器 2 的可编程时钟输出模式编程步骤如下：

1. 置位 T2MOD 寄存器的 T2OE 位。
2. 清除 T2CON 寄存器的 C/T2 位。
3. 从公式计算出 16 位加载值并输入到 RCAP2H 和 RCAP2L 寄存器。
4. 在 TH2 和 TL2 输入一个 16 位初始值。可以和重载值相等。
5. 设置 T2CON 的 TR2 控制位开启动定时器。

在时钟输出模式，定时器 2 翻转不会产生中断，这和用作波特率发生器时相似。可同时使用定时器 2 作为一个波特率发生器和时钟发生器。注意，波特率和时钟输出都由定时器 2 的溢出速率来决定。

注：
(1) 此模式定时器 2 溢出标志 TF2 从来不会被置 1。
(2) SYSCLK=12MHz 时，定时器 2 可编程输出频率范围为 45.7Hz 到 3MHz。

10.2.5. 定时器 2 寄存器

T2MOD：定时器/计数器 2 模式控制寄存器

SFR 页 =全部

SFR 地址 = 0xC9 复位= XXX0-XX00

7	6	5	4	3	2	1	0
--	--	--	**T2X12**	--	--	T2OE	DCEN
R	**R**	**R**	**R/W**	**R**	**R**	**R/W**	**R/W**

Bit 7~5: 保留。当对 T2MOD 进行写操作的时候这些位必须写“0”

Bit 4: T2X12, 定时器 2 时钟源选择
0: 选择 SYSCLK/12 作为时钟源（当捕获模式和自动载入模式 T2CON.C/T2 = 0 时）
1: 选择 SYSCLK 作为时钟源（当捕获模式和自动载入模式 T2CON.C/T2 = 0 时）

Bit 3~2: 保留。当对 T2MOD 进行写操作的时候这些位必须写“0”

Bit 1: T2OE, 定时器 2 时钟输出使能位
0: 禁止定时器 2 时钟输出
1: 使能定时器 2 时钟输出

Bit 0: DCEN, 定时器 2 向下计数使能位
0: 定时器 2 向上计数
1: 定时器 2 向下计数

T2CON：定时器/计数器 2 模式控制寄存器

SFR 页 = 仅 **0 页**

SFR 地址 = 0xC8 复位值 = 0000-0000

7	6	5	4	3	2	1	0
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7: TF2, 定时器2溢出标志
0: TF2 必须由软件清零
1: 当定时器2产生溢出时TF2被置位。当RCLK=1或TCLK=1时，TF2不会被置位。

Bit 6: EXF2, 定时器2外部标志位
0: EXF2必须由软件清零
1: 在EXEN2=1时，且在T2EX上有负跳变时加载或捕获将引起置位。当定时器2中断使能时，EXF2=1时将引起CPU进入定时器2中断向量程序。EXF2必须通过软件清零。EXF2在向上/向下计数器模式不会产生中断。

Bit 5: RCLK, 接收时钟控制位
0: 使用定时器1溢出脉冲来产生接收时钟。
1: 在模式1和模式3时，串行口使用定时器2溢出脉冲来产生接收时钟

Bit 4: TCLK, 传送时钟控制位
0: 使用定时器1溢出脉冲来产生发送时钟
1: 在模式1和模式3时，串行口使用定时器2溢出脉冲来产生发送时钟

Bit 3: EXEN2, 定时器2外部使能位
0: 定时器2忽略T2EX引脚上的事件
1: 如果定时器2没有用作串行口时钟，在T2EX的负跳变时捕获或加载并作为结果。

Bit 2: TR2, 定时器2运行控制位
0: 停止定时器2
1: 启动定时器2

Bit 1: C/T2, 定时器或计数器选择
0: 选择定时器2作为内部定时器功能
1: 选择定时器2作为外部事件计数器(下降沿触发)。

Bit 0: CP/-RL2, 捕获/加载控制位
0: 如果EXEN2=1，定时器2溢出或T2EX上有负跳变时将产生自动加载。
1: 如果EXEN2=1，在T2EX的负跳变时将产生捕获。
当 RCLK=1 或 TCLK=1 时，这一位被忽略并在定时器 2 溢出时强制加载。

当DCEN=0时，定时器2能用作标准8052(总是向上)一样的功能。当DCEN=1时，定时器2能根据T2EX引脚（P1.0）上的逻辑电平进行向上计数或向下计数。定时器2的工作模式如下表所示：

RCLK + TCLK	CP/-RL2	TR2	DCEN	T2OE	模式
x	x	0	x	0	定时器关闭
1	x	1	0	0	波特率发生器
0	1	1	0	0	16位捕获
0	0	1	0	0	16位自动加载（仅向上计数）
0	0	1	1	0	16位自动加载(向上计数或向下计数)
0	0	1	0	1	可编程时钟输出

TL2：定时器 2 低

SFR 页 =全部

SFR 地址 = 0xCC 复位值 = 0000-0000

7	6	5	4	3	2	1	0
TL2.7	TL2.6	TL2.5	TL2.4	TL2.3	TL2.2	TL2.1	TL2.0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

TH2：定时器 2 高

SFR 页 =全部

SFR 地址 = 0xCD 复位值 = 0000-0000

7	6	5	4	3	2	1	0
TH2[7]	TH2[6]	TH2[5]	TH2[4]	TH2[3]	TH2[2]	TH2[1]	TH2[0]
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

RCAP2L：定时器 2 捕获寄存器低

SFR 页 =全部

SFR 地址 = 0xCA 复位值 = 0000-0000

7	6	5	4	3	2	1	0
RCAP2L[7]	RCAP2L[6]	RCAP2L[5]	RCAP2L[4]	RCAP2L[3]	RCAP2L[2]	RCAP2L[1]	RCAP2L[1]
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

RCAP2H：定时器 2 捕获寄存器高

SFR 页 =全部

SFR 地址 = 0xCB 复位值 = 0000-0000

7	6	5	4	3	2	1	0
RCAP2H[7]	RCAP2H[6]	RCAP2H[5]	RCAP2H[4]	RCAP2H[3]	RCAP2H[2]	RCAP2H[1]	RCAP2H[0]
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

10.2.6. 定时器 0/1 示例代码

(1). 功能需求: 定时器 T0 以 10KHz 的频率唤醒空闲模式，SYSCLK = 12MHz 晶振

汇编语言代码范例:

```
T0M0            EQU             01h
T0M1            EQU             02h
PT0         EQU             02h
PT0H            EQU             02h
IDL         EQU             01h

    ORG     0000h
    JMP     main

    ORG     0000Bh
time0_isr:
    to do…
    RETI

main:                                                   ;
    MOV     TH0,#(256-100)              ; 设置定时器 0 溢出率为 = SYSCLK x 100
    MOV     TL0,#(256-100)              ;
    ANL     TMOD,#0F0h                  ; 设置定时器为模式 2
    ORL     TMOD,#T0M1                  ;
    CLR     TF0                                 ; 清定时器 0 标志位

    ORL     IP,#PT0                             ; 选择定时器 0 中断优先级
    ORL     IPH,#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     PCON,#IDL                   ; 设置 MCU 进入空闲模式
    JMP     $
```

C 语言代码范例:

```
#define     T0M0            0x01
#define     T0M1            0x02
```

```
#define     PT0             0x02
#define     PT0H                0x02
#define     IDL             0x01

void time0_isr(void) interrupt 1
{
    To do…
}
void main(void)
{
    TH0 = TL0 = (256-100);                  //设置定时器 0 溢出率为  = SYSCLK x 100
    TMOD &= 0xF0;                           // S 设置定时器为模式 2
    TMOD |= T0M1;
    TF0 = 0;                                //清定时器 0 标志位

    IP |= PT0;                              //选择定时器 0 中断优先级
    IPH |= 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
    PCON =IDL;                              //设置 MCU 进入空闲模式
    while(1);
}
```

(2). 功能需求: 选择定时器 0 时钟源为 SYSCLK (使能 T0X12)

汇编语言代码范例:

```
T0M0            EQU             01h
T0M1            EQU             02h
PT0         EQU             02h
PT0H            EQU             02h
T0X12           EQU             80h

    ORG     0000h
    JMP     main

    ORG     0000Bh
time0_isr:
    to do…
    RETI

main:
    ORL     AUXR, #T0X12                ; 选择定时器 0 时钟源为 SYSCLK
    CLR     TF0                         ; 清定时器 0 标志位

    ORL     IP,#PT0                     ; 选择定时器 0 中断优先级
    ORL     IPH,#PT0H                   ;

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

    MOV     TH0, #(256 - 240)           ;中断间隔 20us
    MOV     TL0, #(256 - 240)           ;

    ANL     TMOD,#0F0h                  ; 设置定时器 0 为模式 2
    ORL     TMOD,#T0M1                  ;


    SETB    TR0                         ; 启动定时器 0
    JMP     $
```

C 语言代码范例:

```
#define     T0M0                0x01
#define     T0M1                0x02
#define     PT0             0x02
#define     PT0H                0x02
#define     T0X12               0x80

    AUXR |= T0X12                       //选择定时器 0 时钟源为 SYSCLK
    TF0 = 0;

    IP |= PT0;                          //选择定时器 0 中断优先级
    IPH |= PT0H;

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

    TH0 = TL0 = (256 - 240);

    TMOD &= 0xF0;                       //设置定时器 0 为模式 2
    TMOD |= T0M1;

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

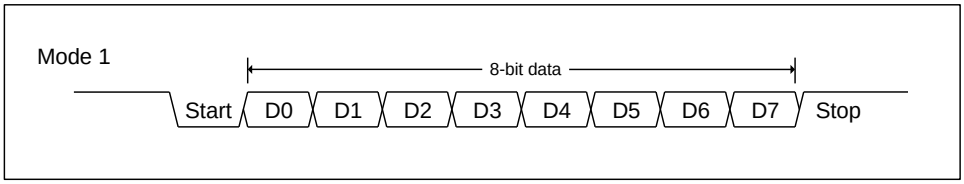
11. 串行口 0 (UART0)

MA805-64 支持全双工的串行口，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 SBUF0 来访问。写到 SBUF0 加载到传送寄存器，当从 SBUF0 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 RXD0(P3.0)传送和接收。TXD0(P3.1)总是作为输出移位时钟。波特率可通过 SCFG 寄存器的 URM0X6 位选择为系统时钟频率的 1/12 或 1/2。

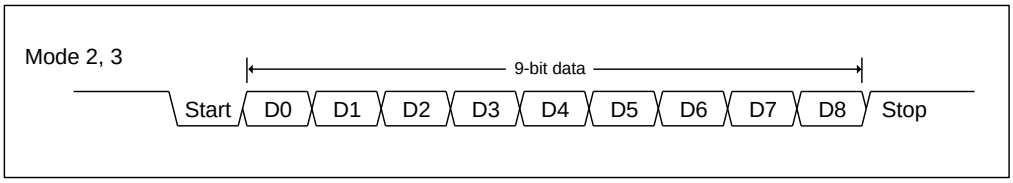
模式 1：10 位通过 TXD0 传送或通过 RXD0 接收，数据帧包括一个起始位(0)，8 个数据位(低位优先)，和一个停止位(1)。在接收时，停止位进入到专用寄存器(SCON0)的 RB80。波特率是可变的。

图 112-1 模式 1 数据帧



模式 2：11 位通过 TXD0 传送或通过 RXD0 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)。在传送时，第 9 个数据位(TB80 在 SCON0 寄存器)可以分配为 0 或者 1。例如，奇偶检验位(P，在 PSW 寄存器)可以移到 TB80 中。在接收时，第九个数据位到 SCON0 寄存器中的 RB80，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。也就是 Fosc/64 或 Fosc/32。

图 11-2 模式 2、3 数据帧



模式 3：11 位通过 TXD0 传送或通过 RXD0 接收，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)。实际上，模式 3 和模式 2 除了波特率不相同之外其它的都相同。模式 3 的波特率是可变的。

在四种模式中，使用 SBUF0 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 RI0=0 且 REN0=1 时启动接收。在其它模式，在 REN0=1 时，收到起始位时启动接收。

除了标准操作外，UART0 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。

11.1. UART0 模式 0 详述

串行数据通过 RXD0 读入和输出，TXD0 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 SCFG 寄存器中的 URM0X6 选择为系统时钟的 1/12 或 1/2。图 12-3 显示了串口模式 0 的简化功能框图。
使用 SBUF0 作为一个目的寄存器可通过任何指令来启动传输。“写到 SBUF0”信号触发 UART0 引擎开始发送。SBUF0 里面的数据在 TXD0（P3.1）脚的每一个上升沿移出到 RXD0（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。见图 12-4。

当 REN0=1 和 RI0=0 时接收启动。在下一个指令周期，RX0 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 RXD0（P3.0）脚并移到寄存中。八个下降沿移位时钟过后，硬件置 RI0 为 1 标志接收完成。见图 12-5。

图 11-3 串行口 0 模式 0

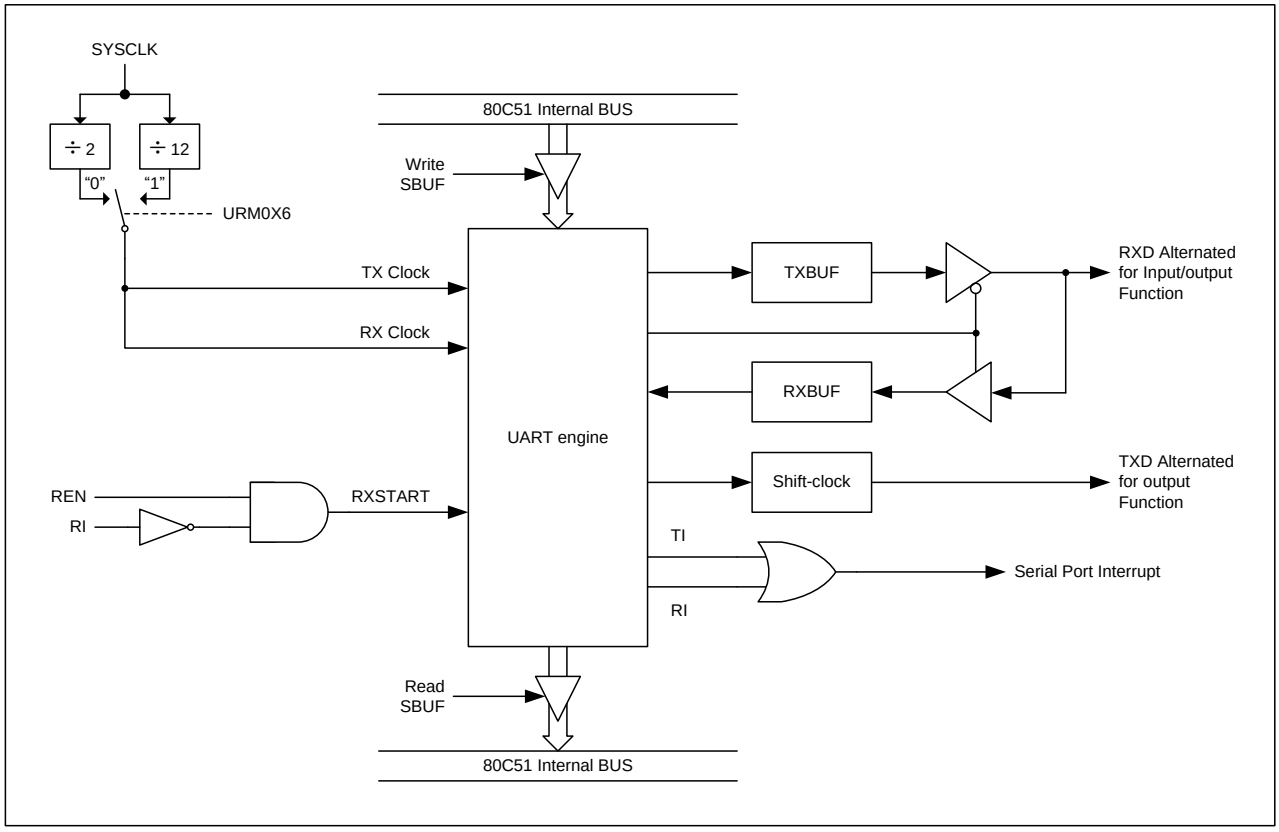


图 11-4 模式 0 发送波形

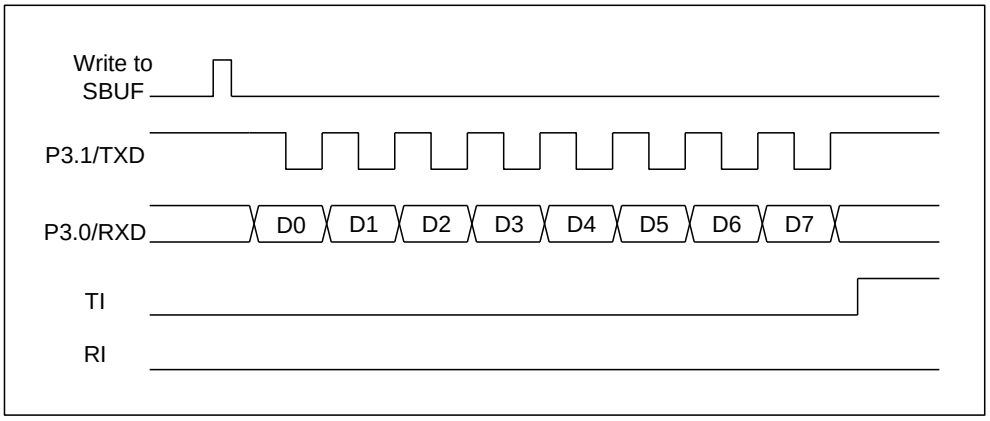
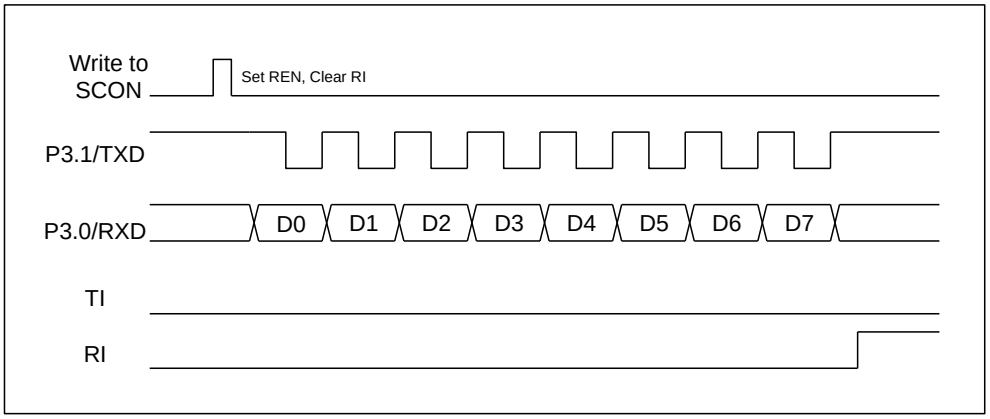


图 11-5 模式 0 接收波形



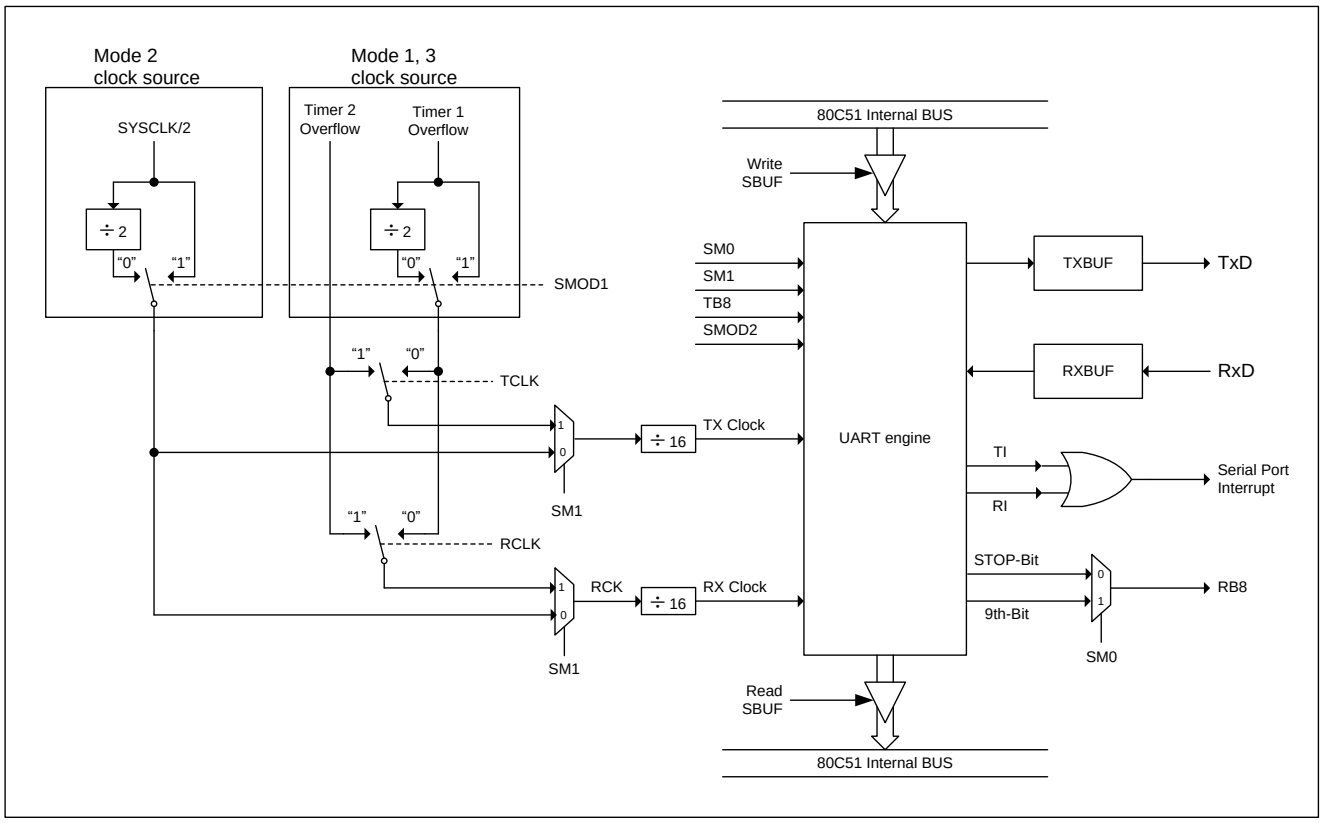
11.2. UART0 模式 1 详述

通过 TXD0 发送 10 位数据或通过 RXD0 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 SCON0 的 RB80，波特率由定时器 1 或定时器 2 的溢出速率来决定。见图 12-1 和图 12-6。

使用 SBUF0 作为目的寄存器的任何指令来启动传输。写到 SBUF0 的信号请求 UART0 引擎开始发送，当收到一个发送请求后，UART0 将在 TX 时钟的上升沿开始发送。SBUF0 中的数据从 TXD0 引脚串行输出，数据帧如图 12-1 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 TI0 表示发送结束。

当串行口 0 控制器在 RCK 采样时钟下检测到在 RXD0 有 1 到 0 跳变的起始位时接收开始。在 RXD0 引脚上的数据将被串行口 0 的位侦查器采样。当收到停止位后，硬件置位 RI0 表示接收结束并把停止位加载到 SCON0 寄存器的 RB80。

图 11-6 串行 0 模式 1、2、3



11.3. UART0 模式 2、3 详述

通过 TXD0 传送 11 位或通过 RXD0 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(TB80)可分配为 0 或 1。在接收时，数据的第 9 位将进入到 SCON0 的 RB80。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。

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。见图 12-2 和图 12-6。

写到 SBUF0 的信号请求 UART0 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，UART0 将在 TX 时钟的上升沿开始发送。SBUF0 中的数据从 TXD0 引脚串行输出，数据帧如图 12-2 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 TI0 表示发送结束。

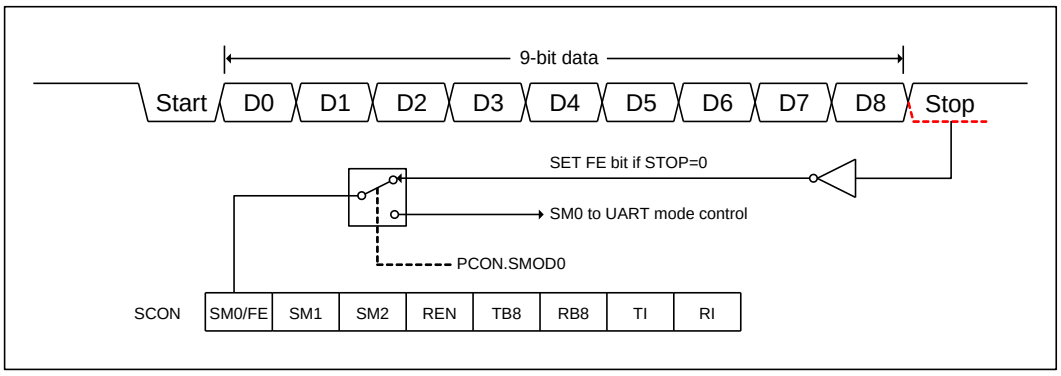
当串行口 0 控制器在 RCK 采样时钟下检测到在 RXD0 有 1 到 0 跳变的起始位时接收开始。在 RXD0 引脚上的数据将被串行口 0 的位侦查器采样。当收数据接收完后，硬件置位 RI0 表示接收结束并把第 9 位加载到 SCON0 寄存器的 RB80。

在四种模式中，使用 SBUF0 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 RI0=0 且 REN0=1 时启动接收。在其它模式，在 REN0=1 时，收到有 1 到 0 跳变的起始位时启动接收。

11.4. 帧错误检测

开启帧错误检测功能后，UART0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SCON0寄存器的FE标志位。FE标志位和SM00标志位共享SCON0.7，SMOD0标志位(PCON.6)决定SCON0.7究竟代表哪个标志，如果SMOD0位（PCON0.6）置位则SCON0.7就是FE标志，SMOD0位清零则SCON0.7就是SM00标志。当SCON0.7代表FE时，只能软件清零。参考图12-7。

图 11-7 UART0 帧错误检测



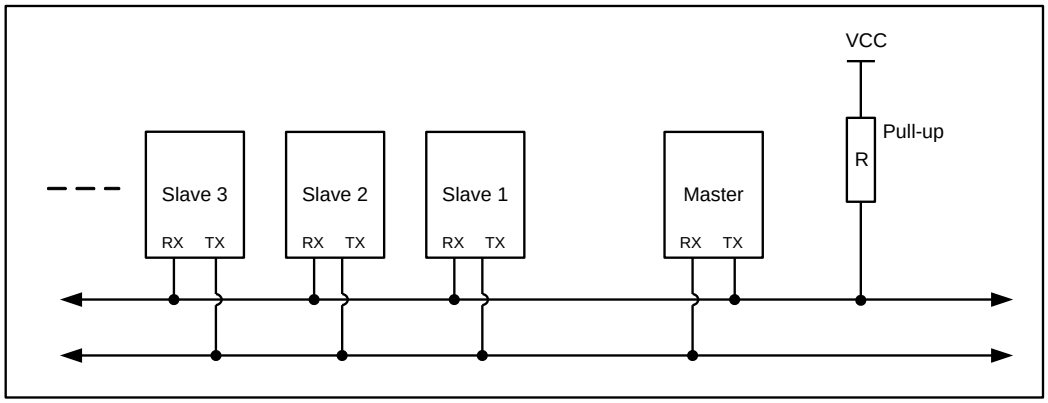
11.5. 多处理器通讯

模式 2 和 3 在用作多处理器通讯时有特殊的规定。在这两种模式，接收 9 个数据位。第 9 个数据位存入 RB80，接着进来一个停止位。端口可以编程为：在 RB80=1 时，当收到停止位后，串口中断将激活。这种特征通过设置 SM20 位(在 SCON0 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 SM20=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 SM20 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 SM20 置位，并继续他们的工作，忽略进来的数据字节。

SM20 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 SM20=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 11-8 UART0 多处理器通讯



11.6. 自动地址识别

自动地址识别通过硬件比较可以让 UART0 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。该功能通过设定 SCON0 的 SM20 位来开启。

在 9 位数据 UART0 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(RI0)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 12-9。在 8 位模式，即模式 1 下，如果 SM20 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 RI0 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
Given = 1100 00X0	Given = 1100 000X

上面的例子中 SADDR 是相同的值，而使用 SADEN 数据来区分两个从机。

从机 0 要求第 0 位必须为 0，并忽略第 1 位的值；从机 1 要求第 1 位必须为 0，并忽略第 0 位的值。从机 0 的唯一地址是 1100 0010，而从机 1 的唯一地址是 1100 0001，地址 1100 0000 是可以同时寻找到从机 0 和从机 1 的。

下面一个更为复杂的系统可以寻址到从机 1 和从机 2，而不会寻址到从机 0：

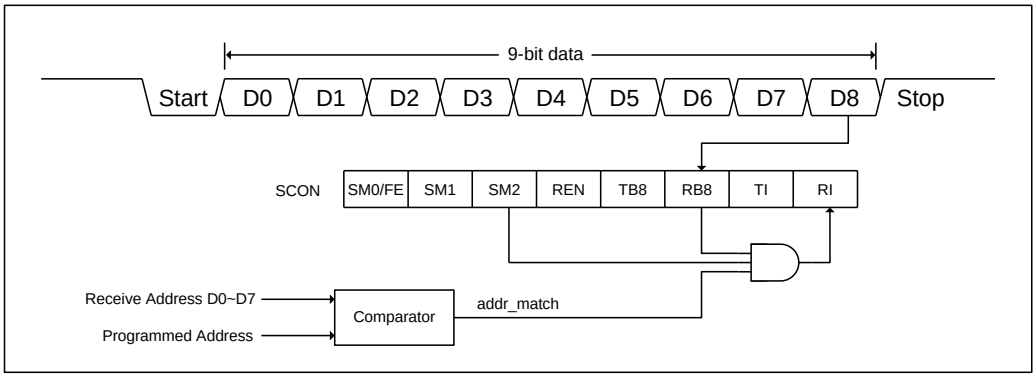
从机 0	从机 1	从机 2
SADDR = 1100 0000	SADDR = 1110 0000	SADDR = 1110 0000
SADEN = 1111 1001	SADEN = 1111 1010	SADEN = 1111 1100
Given = 1100 0XX0	Given = 1110 0X0X	Given = 1110 00XX

上面的例子中，3 个从机的低 3 位地址不一样，从机 0 要求第 0 位必须为 0，1110 0110 可以唯一寻址从机 0；从机 1 要求第 1 位必须为 0,1110 0101 可以唯一寻址从机 1;从机 2 要求第 2 位必须为 0,它的唯一地址是 1110 0011。为了寻址到从机 0 和从机 1 而不会寻址到从机 2，可以使用地址 1110 0100，因为这个地址第 2 位是 1。

每个从机的广播地址 SADDR 和 SADEN 的逻辑或，0 按不需关心处理。大部分情况下，使用 FF 作为广播地址。

复位后，SADDR（SFR 地址 0xA9）和 SADEN（SFR 地址 0xB9）值均为 0，这样可以接收所有地址的信息，也就有效的禁用了自动地址识别模式，从而使该处理器运行于标准 80C51 的 UART 下。

图 11-9 自动地址识别



注： (1)收到匹配地址后(addr_match=1),清 SM20 以接收数据字节
(2) 收完全部数据字节后,置 SM20 为 1 以等待下一个地址

11.7. 波特率设置

位AUXR2.T1X12、SCFG.URM0X6 和SCFG .SMOD2 提供一个新的波特率选项设置，如下所列：

11.7.1. 模式 0 波特率

$$\text{Mode 0 Baud Rate} = \frac{F_{SYSCLK}}{n}$$; n=12, if URM0X6=0 ; n=2, if URM0X6=1

注:
如果 URM0X6=0，波特率公式跟标准 8051 一样。

11.7.2. 模式 2 波特率

$$\text{Mode 2 Baud Rate} = \frac{2^{SMOD1} \times 2^{(SMOD2 \times 2)}}{64} \times F_{SYSCLK}$$

注:
如果 SMOD2=0，波特率公式跟标准 8051 一样。如果 SMOD2=1，波特率设置有增强功能。下表定义了模式 2 波特率发生器由 SMOD2 因数决定的波特率设置。

SMOD2	SMOD1	Baud Rate	Note

0	0	Default Baud Rate	Standard function
0	1	Double Baud Rate	Standard function
1	0	Double Baud Rate X2	Enhanced function
1	1	Double Baud Rate X4	Enhanced function

11.7.3. 模式 1 和 3 波特率

使用定时器 1 作波特率发生器

$$\text{Mode 1, 3 Baud Rate} = \frac{2^{SMOD1} \times 2^{(SMOD2 \times 2)}}{32} \times \frac{F_{SYSCLK}}{12 \times (256 - TH1)} \quad ; T1X12=0$$

$$\text{or} = \frac{2^{SMOD1} \times 2^{(SMOD2 \times 2)}}{32} \times \frac{F_{SYSCLK}}{1 \times (256 - TH1)} \quad ; T1X12=1$$

注:

如果 SMOD2=0，T1X12=0，波特率公式跟标准 8051 一样。如果 SMOD2=1，波特率设置有增强功能。下表定义了定时器 1 波特率发生器由 SMOD2 因数决定的波特率设置。

SMOD2	SMOD1	Baud Rate	Note
0	0	Default Baud Rate	Standard function
0	1	Double Baud Rate	Standard function
1	0	Double Baud Rate X2	Enhanced function
1	1	Double Baud Rate X4	Enhanced function

使用定时器 2 作波特率发生器

当定时器 2 作波特率发生器时（T2CON 寄存器中的 TCLK 或 RCLK 任一位为‘1’），波特率如下：

$$\text{Mode 1, 3 Baud Rate} = \frac{2^{(SMOD2 + 1) \times SMOD1} \times F_{SYSCLK}}{32 \times (65536 - (RCAP2H, RCAP2L))}$$

注:

如果 SMOD2=0，波特率公式跟标准 8051 一样。如果 SMOD2=1，波特率设置有增强功能。下表定义了定时器 2 波特率发生器由 SMOD2 因数决定的波特率设置。

SMOD2	SMOD1	Baud Rate	Note
0	X	Default Baud Rate	Standard function
1	0	Double Baud Rate	Enhanced function
1	1	Double Baud Rate X2	Enhanced function

11.8. 串行口 0 寄存器

四个工作模式除了波特率设置不同外其它都与标准 8051 相同。 PCON, AUXR2 和 SCFG 三个寄存器与波特率设置有关。

SCON0：串行口 0 控制寄存器

SFR 页 = 仅 **0 页**

SFR 地址 = 0x98 复位值 = 0000-0000

7	6	5	4	3	2	1	0
SM00/FE	SM10	SM20	REN0	TB80	RB80	TI0	RI0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7: FE, 帧错误位。 SMOD0 位必须置 1 来使能访问 FE 位。
0: 当收到有效的帧时 FE 不会自动清除，但是可以用软件清除。
1: 当接收器检测到一个无效的停止位时这位置 1。

Bit 7: SM00,串行口 0 模式位 0(SMOD0 必须为 0 来访问 SM00 位)
Bit 6: SM10,串行口 0 模式位 1

SM00	SM10	模式	描述	波特率
0	0	0	移位寄存器	SYSCLK/12 或 /2
0	1	1	8-bit UART	可变的
1	0	2	9-bit UART	SYSCLK/64, /32, /16 或 /8
1	1	3	9-bit UART	可变的

Bit 5: 串行口 0 模式位 2
0: 禁止 SM20 功能
1: 在模式 2 和 3 时使能地址自动识别，如果 SM20=1 那么 RI0 将不能设置，除非接收到的第 9 位数据(RB80)为 1，指示是一个地址，并且接收到的字节是本机地址或者是一个广播地址；在模式 1，如果 SM20=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本机地址或者是一个广播地址；在模式 0，SM20 可以为 0。

Bit 4: REN0, 使能串行接收
0: 软件清零将禁止接收。
1: 通过软件置 1 接收使能。

Bit 3: TB80, 在模式 2 和 3 时第 9 位数据被传送，根据需要通过软件置位或清零。

Bit 2: RB80, 在模式 2 和 3 时收到的第 9 位数据。在模式 1，如果 SM20=0，RB80 是收到数据的停止位。在模式 0，RB80 没有使用。

Bit 1: TI0. 发送中断标志
0: 必须由软件清零。
1: 在模式 0 时，在第 8 位个数据位时序后由硬件置位。其它模式中，在发送停止位之初由硬件置位。

Bit 0: RI0. 接收中断标志
0: 必须由软件清零。
1: 在模式 0 时，在第 8 位个数据位时序后由硬件置位。其它模式中(除留意 SM20 外)，在接收停止位的中间时刻由硬件置位

SBUF0: 串行口 0 缓冲寄存器

SFR 页 =仅 **0 页**
SFR 地址 = 0x99 复位值 = XXXX-XXXX

7	6	5	4	3	2	1	0
SBUF0[7]	SBUF0[6]	SBUF0[5]	SBUF0[4]	SBUF0[3]	SBUF0[2]	SBUF0[1]	SBUF0[0]
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7~0: 在发送和接收时作缓冲寄存器

SADDR：从机地址寄存器

SFR 页 =全部

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：从机地址屏蔽寄存器

SFR 页 =全部

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

当地址自动识别功能启用后，可用 SADDR 和 SADEN 组合来预置地址，事实上，SADEN 是 SADDR 的“屏蔽”寄存器，如下图所示

SADDR = 1100 0000

SADEN = 1111 1101

Given = 1100 00x0 ⟶ 这“Given”从机地址将被选中

Bit1 作“不关心”处理

每个从对象的广播地址为 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，从而忽略“Given”地址的全部地址位和“广播”地址的全部地址位而导致自动地址识别功能无效。

PCON0：电源控制寄存器 0

SFR 页 =全部

SFR 地址 = 0x87 复位值 = 00X1-0000

7	6	5	4	3	2	1	0
SMOD1	SMOD0	--	POF	GF1	GF0	PD	IDL
R/W	R/W	R	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 双倍波特率

1: 使能 UART 双倍波特率(模式 1, 2,或 3.)

Bit 6: SMOD0, 帧错误选则

0: SCON.7 作 SM0 功能

1: SCON.7 作 FE 功能。注：当帧错误后不管 SMOD0 什么状态 FE 都将置 1。

SCFG：串行口配置寄存器

SFR 页 =仅 **0 页**

SFR 地址 = 0x9A 复位值 = 0000-00XX

7	6	5	4	3	2	1	0
URTS	SMOD2	URM0X6	S1TR	S1MOD	S1TX12	--	--
R/W	R/W	R/W	R/W	R/W	R/W	R	R

Bit 7: URTS, UART0 定时器选择

0: 模式 1 和模式 3 时选择定时器 1 或定时器 2 作波特率发生器

1: UART0 的模式 1 或模式 3 中当定时器 1 被选择作波特率发生器时，定时器 1 溢出信号被 UART1 波特率定时器溢出信号取代。（参考图 13-1）

Bit 6: SMOD2, 额外双倍波特率选择
0: 禁止 UART0 额外双倍波特率
1: 使能 UART0 额外双倍波特率

Bit 5: URM0X6, 串行口模式 0 波特率选择
0: 选择 SYSCLK/12 作 UART 模式 0 波特率
1: 选择 SYSCLK/2 作 UART 模式 0 波特率

Bit 1~0: 保留。当对 SCFG 进行写操作的时候这些位必须写“0”

AUXR2: 辅助寄存器 2

SFR 页 =全部
SFR 地址 = 0x87 上电+复位值 = 00XX-XX00

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

Bit 6: T1X12, 当 C/T=0 时，定时器 1 时钟源选择
0: 选择 SYSCLK/12。
1: 选择 SYSCLK 作时钟源。若在模式 1 和模式 3 中 UART0 选择定时器 1 作波特率发生器则速率是标准 8051 的 12 倍。

12. 串行口 1 (UART1)

MA805-64装备有第2个UART（以后就称作UART1），和第1个UART一样，也有4种运行模式，两个UART的区别如下：

(1) UART1没有增强功能：帧错误检测和自动地址识别。

(2) UART1使用特定的波特率定时器作为其波特率发生器。

(3) UART1使用端口P1.3 (TXD1) 和 P1.2 (RXD1) 分别作为接收和发送端口。

两个 UART 可以不同或相同模式、不同或相同通讯速率同时工作。

12.1. 串行口 1 波特率

12.1.1. 模式 0 波特率

$$\text{S1 Mode 0 Baud Rate} = \frac{F_{SYSCLK}}{12}$$

注:

如果 URM0X6=0，波特率公式跟标准 8051 一样。.

12.1.2. 模式 2 波特率

$$\text{S1 Mode 2 Baud Rate} = \frac{2^{S1MOD1}}{64} \times F_{SYSCLK}$$

12.1.3. 模式 1、3 波特率

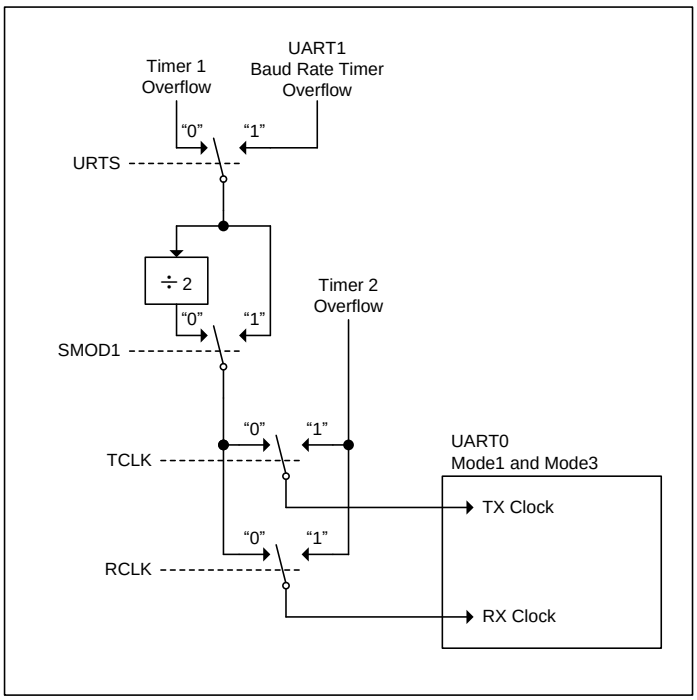
$$\text{S1 Mode 1, 3 Baud Rate} = \frac{2^{S1MOD1}}{32} \times \frac{F_{SYSCLK}}{12 \times (256 - S1BRT)} \quad ; S1X12=0$$

$$\text{or} = \frac{2^{S1MOD1}}{32} \times \frac{F_{SYSCLK}}{1 \times (256 - S1BRT)} \quad ; S1X12=1$$

12.2. UART0 使用 UART1 波特率发生器

UART0 的模式 1 和模式 3，用户能通过清零 T2CON 寄存器中的 TCLK 和 RCLK 选择定时器 1 作波特率发生器，这时若 SCFG 寄存器中的 URTS 位为 1 则定时器 1 溢出信号被 UART1 波特率定时器溢出信号取代。也就是说，用户通过设置 RCLK=0, TCLK=0 及 URTS=1 即可采用 UART1 波特率定时器作为 UART0 在模式 1 或模式 3 时的波特率发生器。这样定时器 1 可以作其它应用。当然，如果 UART1（模式 1 或模式 3）在此时同时运行，则这两个 UART 将具有相同的波特率。

图 123-1. UART0 增加的波特率源



12.3. 串行口 1 寄存器

下面的特殊功能寄存器与 UART1 有关：

SCON1：串行口 1 控制寄存器

SFR 页 = **仅 1 页**

SFR 地址 = 0x98 上电+复位值 = 0000-0000

7	6	5	4	3	2	1	0
SM01	SM11	SM21	REN1	TB81	RB81	TI1	RI1
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7: SM01,串行口 1 模式位 0

Bit 6: SM11,串行口 1 模式位 1

SM01	SM11	Mode	Description	Baud Rate
0	0	0	shift register	SYSCLK/12
0	1	1	8-bit UART	variable
1	0	2	9-bit UART	SYSCLK/64, /32
1	1	3	9-bit UART	variable

Bit 5: 串行口 1 模式位 2

0: 禁止 SM21 功能

1: 在模式 2 和 3 时使能地址自动识别，如果 SM21=1 那么 RI1 将不能设置，除非接收到的第 9 位数据(RB81)为 1，指示是一个地址，并且接收到的字节是本机地址或者是一个广播地址；在模式 1，如果 SM21=1 那么 RI1 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本机地址或者是一个广播地址；在模式 0，SM21 可以为 0。

Bit 4: REN1, 使能串行接收

0: 软件清零将禁止接收。

1: 通过软件置 1 接收使能。

Bit 3: TB81, 在模式 2 和 3 时第 9 位数据被传送，根据需要通过软件置位或清零。

Bit 2: RB81, 在模式 2 和 3 时收到的第 9 位数据。在模式 1，如果 SM21=0，RB80 是收到数据的停止位。在模式 0，RB81 没有使用。

Bit 1: TI1. 发送中断标志
0: 必须由软件清零。
1: 在模式 0 时，在第 8 位个数据位时序后由硬件置位。其它模式中，在发送停止位之初由硬件置位。

Bit 0: RI1. 接收中断标志
0: 必须由软件清零。
1: 在模式 0 时，在第 8 位个数据位时序后由硬件置位。其它模式中(除留意 SM21 外)，在接收停止位的中间时刻由硬件置位

SBUF1: 串行口 1 缓冲寄存器

SFR 页 =仅 **1 页**
SFR 地址 = 0x99 上电+复位值 = XXXX-XXXX

7	6	5	4	3	2	1	0
SBUF1[7]	SBUF1[6]	SBUF1[5]	SBUF1[4]	SBUF1[3]	SBUF1[2]	SBUF1[1]	SBUF1[0]
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~0: 在发送和接收时作缓冲寄存器

S1BRT: 串行口 1 波特率定时器重载寄存器

SFR 页 =仅 **1 页**
SFR 地址 = 0x9A 上电+复位值 = 0000-0000

7	6	5	4	3	2	1	0
S1BRT[7]	S1BRT[6]	S1BRT[5]	S1BRT[4]	S1BRT[3]	S1BRT[2]	S1BRT[1]	S1BRT[0]
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~0: 它用于波特率定时器发生器重载变量，工作类似于定时器 1。

SCFG: 串行口配置寄存器

SFR 页 =仅 **0 页**
SFR 地址 = 0x9A 上电+复位值 = 0000-00XX

7	6	5	4	3	2	1	0
URTS	SMOD2	URM0X6	S1TR	S1MOD	S1TX12	--	--
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R**	**R**

Bit 7: URTS, UART0 定时器选择
0: 模式 1 和模式 3 时选择定时器 1 或定时器 2 作波特率发生器
1: UART0 的模式 1 或模式 3 中当定时器 1 被选择作波特率发生器时，定时器 1 溢出信号被 UART1 波特率定时器溢出信号取代。（参考图 13-1）

Bit 4: S1TR, UART1 波特率定时器控制为
0: 关闭 S1BRT.
1: 开启 S1BRT.

Bit 3: S1MOD, UART1 双倍波特率选择
0: 禁止 UART1 双倍波特率功能
1: 使能 UART1 双倍波特率功能

Bit 2: S1TX12, UART1 波特率定时器时钟源选择
0: 选择 SYSCLK/12 作 S1BRT 的时钟源

1：选择 SYSCLK 作 S1BRT 的时钟源

Bit 1~0：保留。当对 SCFG 进行写操作的时候这些位必须写“0”

12.4. 串行口示例代码

(1). 功能需求: 串行口输入 RI 唤醒空闲模式

汇编语言代码范例:

```
PS              EQU         10h
PSH             EQU         10h


    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:
;   Process
    CLR     RI                          ; 清除 RI 标志
    RETI                                ; 中断返回

TI_ISR:
;   Process
    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

    CALL    UART_Baud_Rate_Setting      ;参考 "錯誤! 找不到參照來源。 ~ 錯誤! 找不到參照來源。" 获得更多信息

    MOV     IP,#PSL                     ; 选择串行口中断优先级
    MOV     IPH,#PSH                    ;

    SETB    ES                          ; 使能串行口中断
```

```
    SETB    EA                          ；使能全局中断

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

C 语言代码范例:

```
#define     PS              0x10
#define     PSH             0x10

void uart_ri_idle_isr(void) interrupt 4
{   if(RI)
    {
        RI=0;
        // to do ...
    }

    if(TI)
    {
        TI=0;
        // to do ...
    }
}

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

    UART_Baud_Rate_Setting()                //参考 “11.7 波特率设置 “ 获得更多信息

    IP = PSL;                               //选择串行口中断优先级
    IPH = PSH;                              //

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

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

13. 可编程计数器阵列（PCA）

MA805-64 带有一个可编程计数器阵列(PCA)，该功能与标准定时/计数器相比以更少的 CPU 占用提供了更多的定时能力。它的优点包括减少了软件复杂度并提高了精度。

13.1. PCA 概述

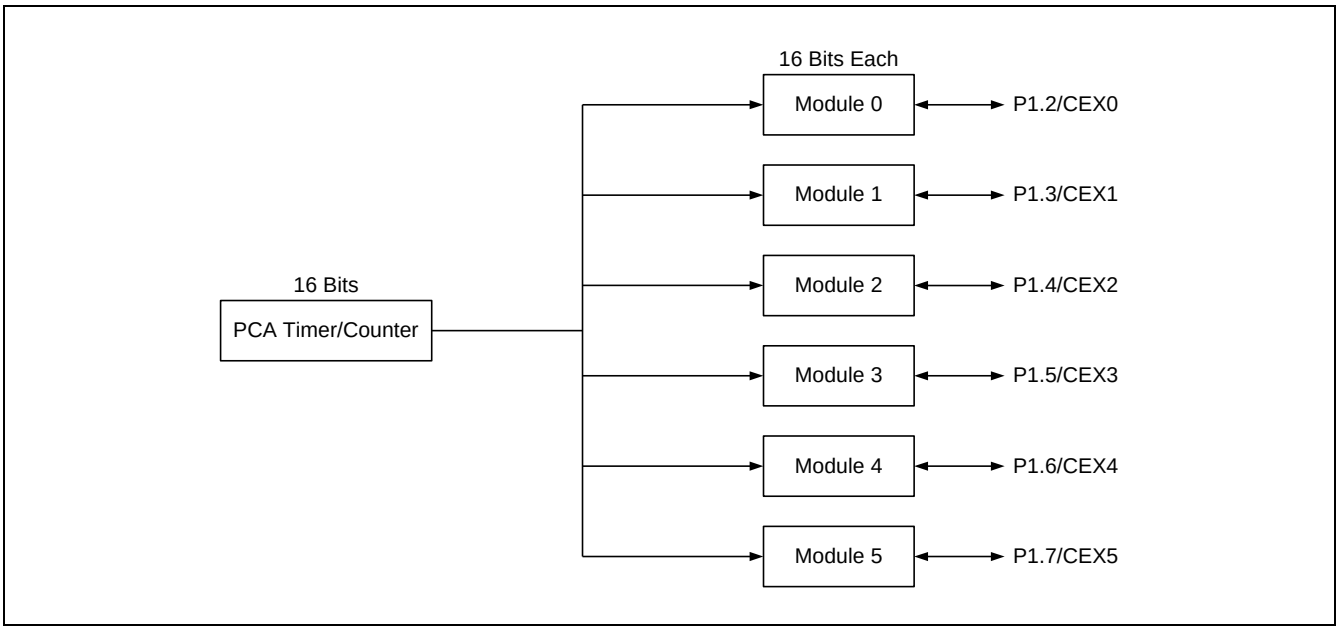
PCA 由一个专用定时/计数器作为一个 6 组比较/捕获模块的时间基础，图 14-1 显示了 PCA 的功能方框图。需要注意的是 PCA 定时器和模块都是 16 位的。如果一个外部事件同一个模块关联，那末该功能就和相应的端口 1 引脚共享。若某模块没有使用端口引脚，这个引脚还可以用作标准 I/O。

6 组比较/捕获模块中的每一组都可以编程为如下任意模式：

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

所有这些模式将在后面的章节进行详细讨论。这里，让我们先看看如何设置 PCA 定时器和模块。

图 14-1. PCA 方框图



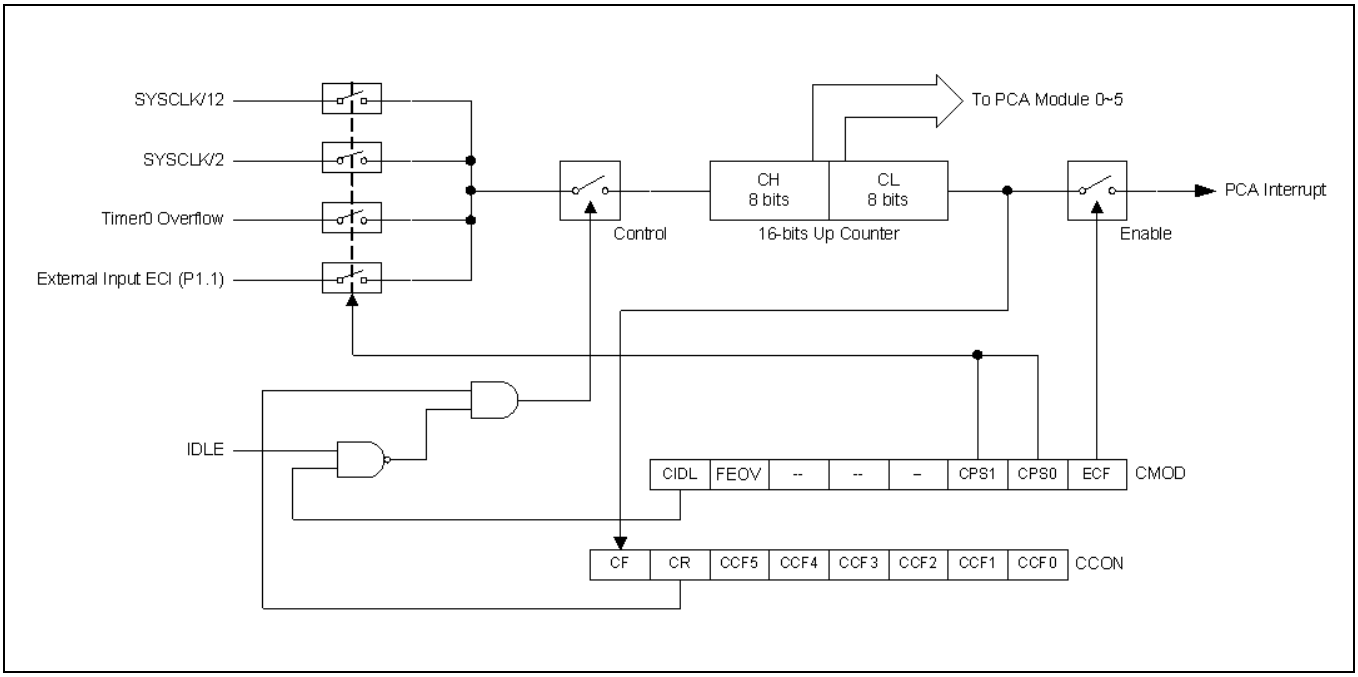
13.2. PCA 定时器/计数器

PCA 的定时器/计数器由一个可以自由运行的 16 位定时器组成，如图 14-2 所示分为 CH 和 CL（计数变量的高低字节）高低两部分，它是所有模块的共有时间基础，它的时钟输入可以从以下来源选择：

- 1/12 系统时钟频率
- 1/2 系统时钟频率
- 定时器 0 溢出，可以让低频时钟源输入到 PCA 定时器
- 外部时钟输入，ECI（P1.1）引脚的 1-0 反转

特殊功能寄存器 CMOD 包含了计数脉冲选择位（CPS1 和 CPS0）来指定 PCA 定时器时钟源。这个寄存器也包括了 ECF 位来使能计数器溢出中断。此外，用户可以在待机模式下设置计数器待机位（CIDL），来关闭 PCA 定时器，这样可以进一步降低待机模式下的功耗。

图 14-2. PCA 定时器/计数器



CMOD: PCA 计数器模式寄存器

SFR 页 =全部

SFR 地址 = 0xD9 复位值 = 0xxx-x000

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

Bit 7: CIDL, PCA 计数器空闲模式控制
0: 让 PCA 计数器在空闲模式下继续运行
1: 空闲模式下关闭 PCA 计数器
Bit 6:FEOV, 最大的计数器{CL} 值 FE
FEOV=0 最大的 CL 计数器值为 FF
FEOV=1 最大的 CL 计数器值为 FE

Bit 5~3: 保留。当对 CMOD 进行写操作的时候这些位必须写“0”

Bit 2~1: CPS1-CPS0, PCA 计数器时钟源选择位

CPS1	CPS0	PCA 时钟源
0	0	内部时钟，(system clock)/12
0	1	内部时钟， (system clock)/2
1	0	定时器 0 溢出
1	1	从 ECItic 引脚输入的外部时钟

Bit 0: ECF, 使能 PCA 计数器溢出中断
0: ECF=1 当 CF 位（CCON 寄存器中）置位时禁止中断
1: ECF=1 当 CF 位（CCON 寄存器中）置位时使能中断

如下所示的 CCON 寄存器包含 PCA 运行控制位和 PCA 定时器与每个模块的标志。要运行 PCA，CR 位（CCON.6）必须软件置 1，要关闭 PCA，可以清零该位。PCA 计数器溢出时，CF (CCON.7)置位，并且若 CMOD 寄存器的 ECF 为置位，还会产生一个中断，CF 位只能软件清零。CCF0 到 CCF5 是模块 0 到模块 5 的相应中断标志位，当发生一个匹配或捕获事件时，硬件置位，这些位也必须软件清零。PCA 中断系统如图 14-3 所示。

CCON: PCA Counter Control Register
SFR 页 =全部
SFR 地址 = 0xD8 复位值 = 0000-0000

7	6	5	4	3	2	1	0
CF	CR	CCF5	CCF4	CCF3	CCF2	CCF1	CCF0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7: CF, PCA 计数器溢出标志
0: 只能软件清零
1: 溢出时硬件置位，CF 标志在 CMOD 寄存器的 ECF 位置位时会产生一个中断，CF 可以硬件或软件置位

Bit 6: CR, PCA 计数器运行控制
0: 停止 PCA 计数器
1: 启动 PCA 计数器

Bit 5: CCF5, PCA 模块 5 中断标志
0: 必须软件清零
1: 发生一个匹配或捕获时硬件置位

Bit 4: CCF4, PCA 模块 4 中断标志
0: 必须软件清零
1: 发生一个匹配或捕获时硬件置位

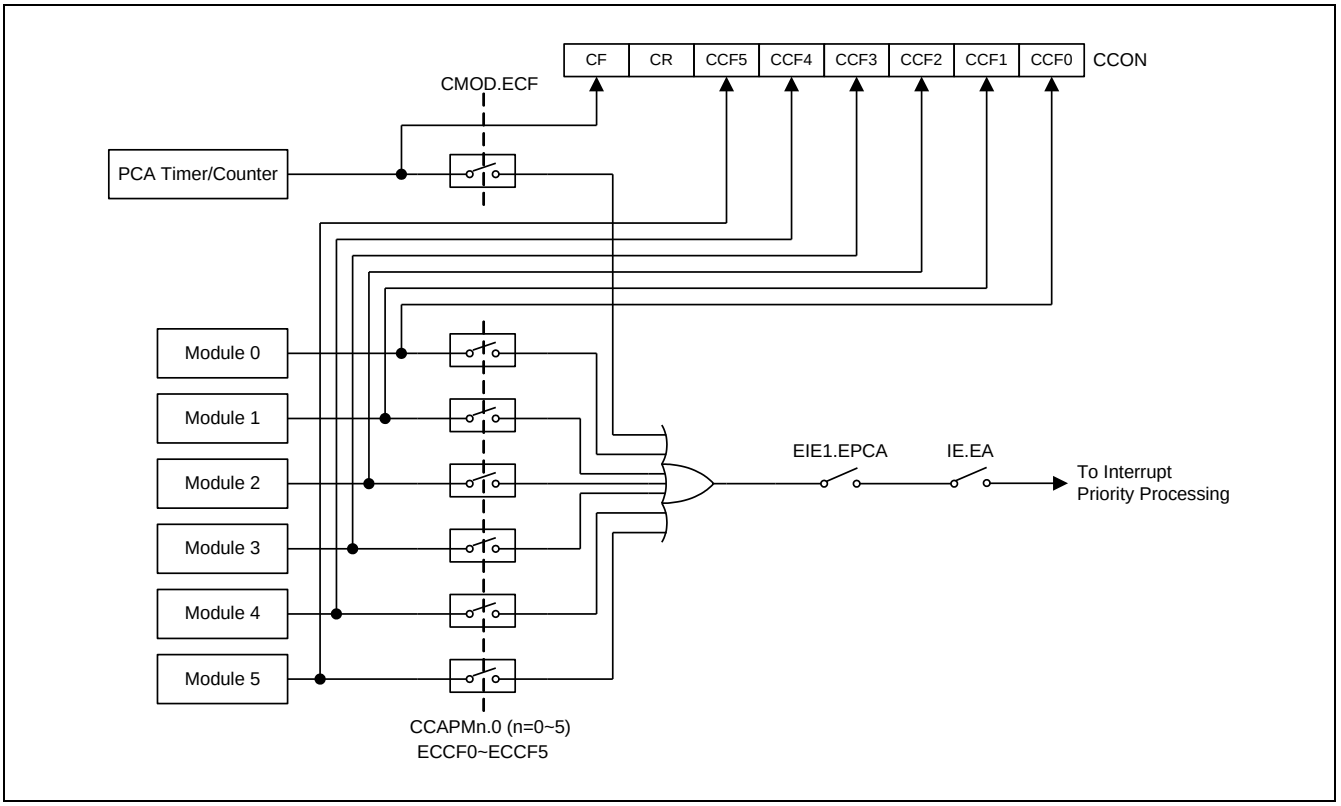
Bit 3: CCF3, PCA 模块 3 中断标志
0: 必须软件清零
1: 发生一个匹配或捕获时硬件置位

Bit 2: CCF2, PCA 模块 2 中断标志
0: 必须软件清零
1: 发生一个匹配或捕获时硬件置位

Bit 1: CCF1, PCA 模块 1 中断标志
0: 必须软件清零
1: 发生一个匹配或捕获时硬件置位

Bit 0: CCF0, PCA 模块 0 中断标志
0: 必须软件清零
1: 发生一个匹配或捕获时硬件置位

图 14-3. PCA 中断系统



13.3. 比较/捕获模块

6 组比较/捕获模块中的每一组都有一个模式寄存器，叫做 CCAPMn（n 代表 0，1，2，3，4，5），来选择其工作模式。ECCFn 位控制当中断标志置位时每个模块的中断开启/关闭。

CCAPMn：PCA 模块比较/捕获寄存器，n=0~5

SFR 页 =全部

SFR 地址 = 0xDA~0xDF 复位值 = x000-0000

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

Bit 7: 保留。当对 CCAPMn 进行写操作的时候这位必须写“0”

Bit 6: ECOMn, 比较器使能
0: 禁止数字比较器功能
1: 使能数字比较器功能

Bit 5: CAPPn, 上升沿捕获使能
0: 禁止在 CEXn 引脚侦察到上升沿时的 PCA 捕获功能
1: 使能在 CEXn 引脚侦察到上升沿时的 PCA 捕获功能

Bit 4: CAPNn, 下降沿捕获使能
0: 禁止在 CEXn 引脚侦察到下降沿时的 PCA 捕获功能
1: 使能在 CEXn 引脚侦察到下降沿时的 PCA 捕获功能

Bit 3: MATn, 匹配控制
0: 禁止数字比较器匹配事件去置位 CCFn
1: PCA 计数器同相应模块的比较/捕获寄存器匹配时设置 CCON 寄存器的 CCFn 位

Bit 2: TOGn, 翻转控制
0: 禁止数字比较器匹配事件去设置 CEXn
1: PCA 计数器同相应模块的比较/捕获寄存器匹配时设置 CEXn 引脚翻转

Bit 1: PWMn, PWM 控制
0: 禁止 PCA 模块中的 PWM 模块
1: 使能 PWM 功能,并设置 CEXn 引脚用作脉宽调制输出引脚

Bit 0: ECCFn, 使能 CCFn 中断
0: 禁止 CCON 寄存器中的比较/捕获标志位 CCFn 产生中断
1: 使能 CCON 寄存器中的比较/捕获标志位 CCFn 产生中断

注：CAPNn (CCAPMn.4)位和 CAPPn (CCAPMn.5)位决定了捕获发生时信号脉冲沿，若这两位同时设置，则上下降沿都会发生捕获。

每一个模块都有一对 8 位比较/捕获寄存器（CCAPnH, CCAPnL）与其相关联。这些寄存器用来保存捕获事件发生时或比较事件发生时的值。当某个模块用作 PWM 模式时，除了上面两个寄存器外，还有一个扩展的寄存器 PCAPWMn 被用来扩展输出占空比的范围，扩展的范围从 0%到 100%，步距是 1/256。

PCAPWMn：PWM 模式辅助寄存器，n=0~5

SFR 页 =全部

SFR 地址 = 0xF2~0xF7 复位值 = 0000-0000

7	6	5	4	3	2	1	0
PnRS1	PnRS0	PnPS2	PnPS1	PnPS0	PnINV	ECAPnH	ECAPnL
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ECAPnH: 扩展的第 9 位（最高位），用在 PWM 模式下，与 CCAPnH 联用并成为其第 9 位
ECAPnL: 扩展的第 9 位（最高位），用在 PWM 模式下，与 CCAPnL 联用并成为其第 9 位

13.4. PCA 运行模式

表14-1显示了不同PCA功能对应的CCAPMn寄存器设置。

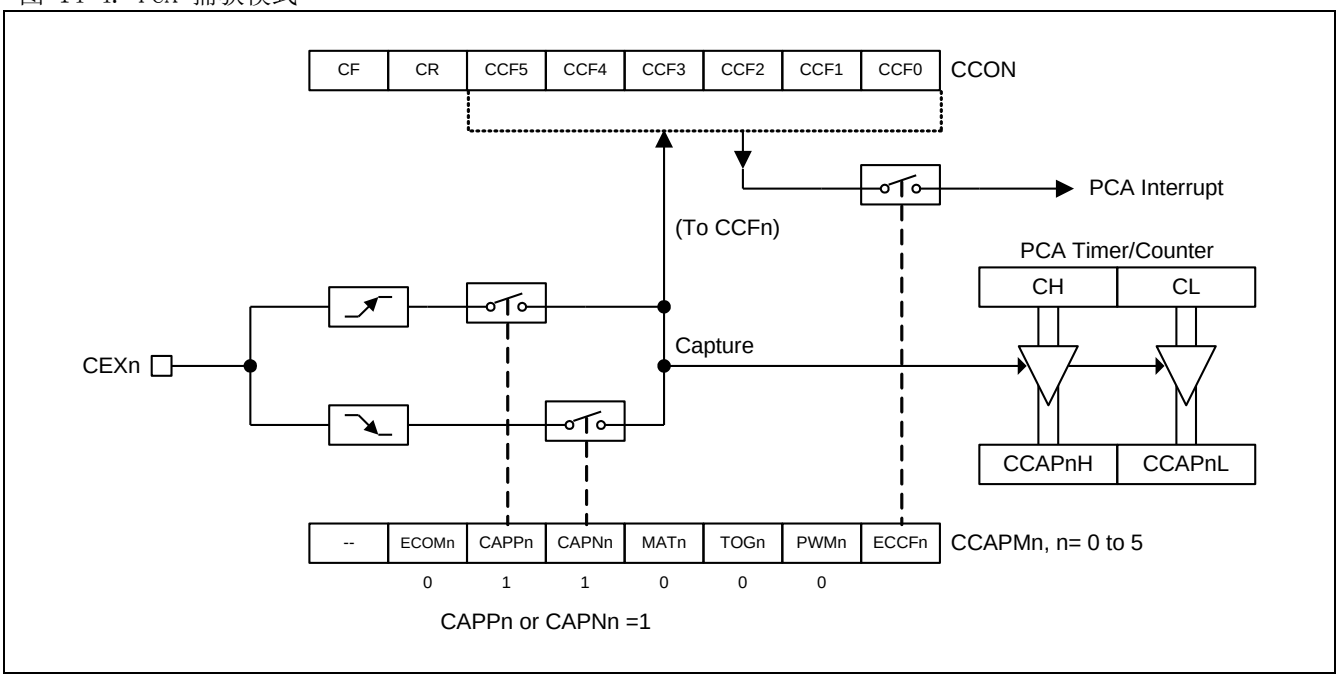
表 14-1. PCA 模块运行模式

ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	Module Function
0	0	0	0	0	0	0	无操作
X	1	0	0	0	0	X	16位CEXn引脚上升沿触发捕获模式
X	0	1	0	0	0	X	16位CEXn引脚下降沿触发捕获模式
X	1	1	0	0	0	X	16位CEXn引脚跳变触发捕获模式
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)

13.4.1. 捕获模式

要让某一 PCA 模块工作在捕获模式，模块的 CAPN、CAPP 任何一位或两位必须置位。外部 CEX 输入会在每次跳变时采样，当有效跳变发生时，PCA 硬件会将 PCA 计数器寄存器值装入模块的捕获寄存器（CH 放入 CCAPnH，CL 放入 CCAPnL）。若模块的 CCFn 和 ECCFn 标志置位，会产生一个中断。

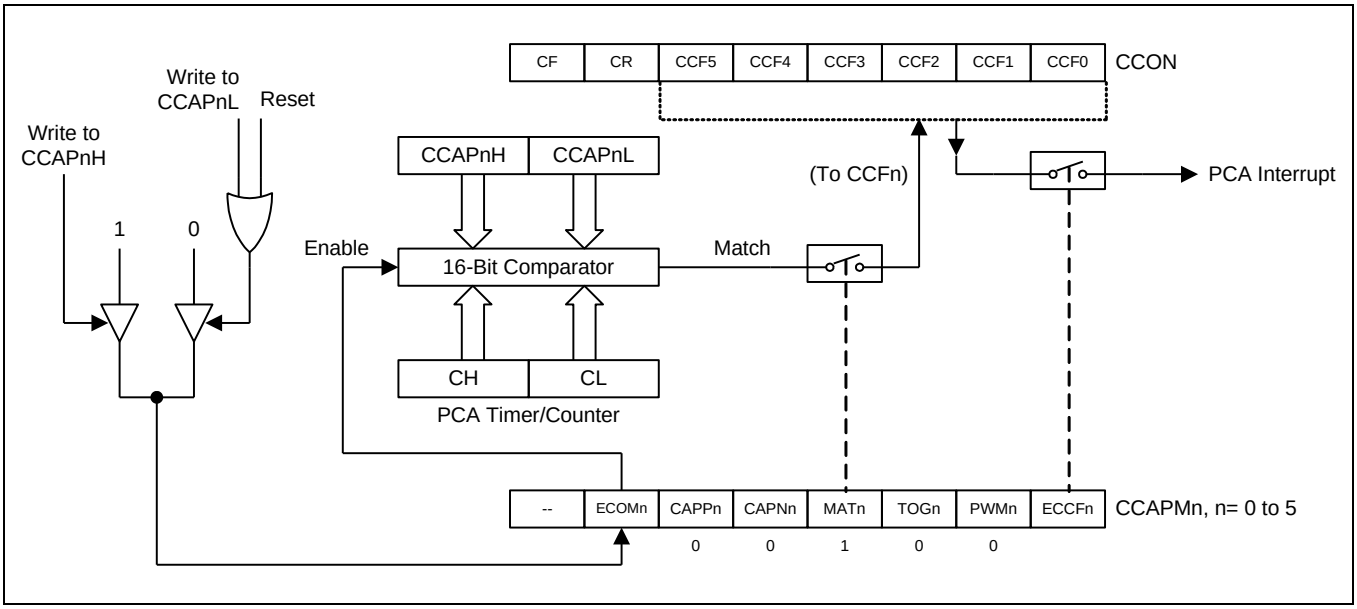
图 14-4. PCA 捕获模式



13.4.2. 16 位软件定时器模式

PCA 模块可以通过设置 CCAPMn 寄存器的 ECOM 位和 MAT 位来作为一个软件定时器使用，PCA 定时器与模块的捕获寄存器值进行比较，若相等且当 CCFn 和 ECCFn 位设置时会产生一个中断信号。

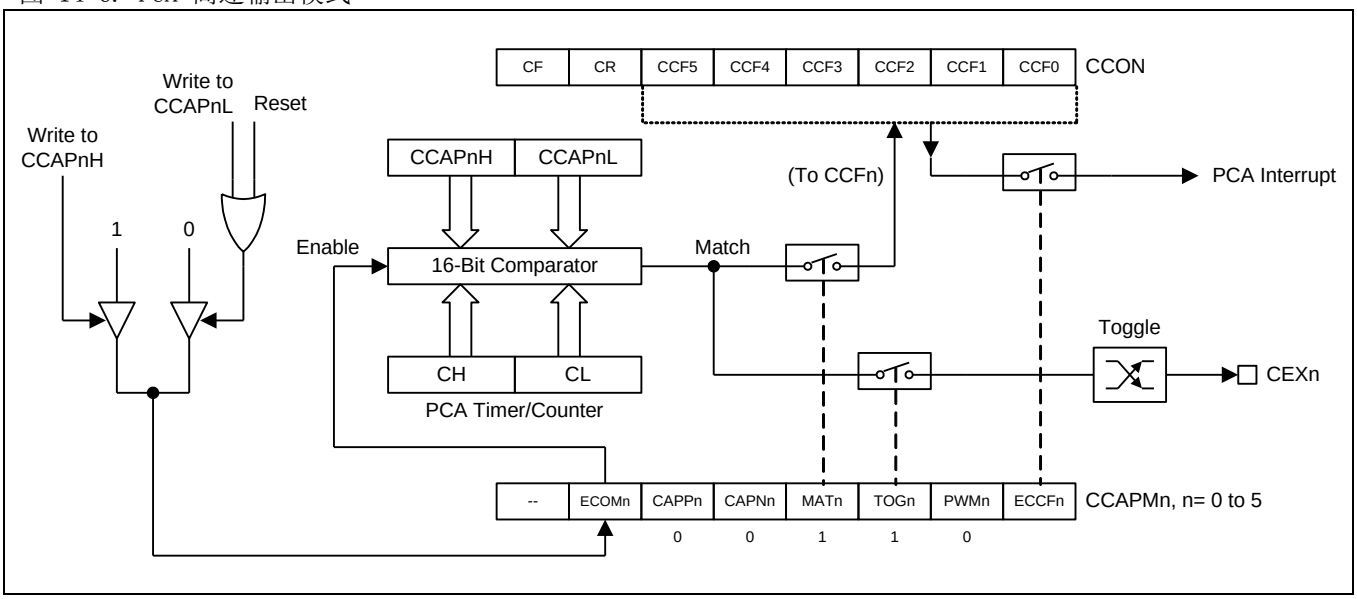
图 14-5. PCA 软件定时器模式



13.4.3. 高速输出模式

这种模式下，每当 PCA 计数器与模块捕获寄存器值相等时，CEX 的输出就翻转一次。为激活这种模式，CCAPMn 寄存器的 TOG、MAT 和 ECOM 位必须都置为 1。

图 14-6. PCA 高速输出模式



13.4.4. PWM 模式

所有PCA模块都可用作PWM输出，输出频率取决于PCA定时器的时钟源，所有的模块都有相同的输出频率，因为它们共享PCA定时器。

占空比取决于模块捕获寄存器CCAPnL 与扩展的第9位ECAPnL的值。当9位数据{0,[CL]}值小于{ ECAPnL, [CCAPnL] }组成的9位数据时，输出低电平，相等或大于时输出高电平。

当CL从0xFF到0x00溢出时， { ECAPnL, [CCAPnL] } 的值使用 { ECAPnH,[CCAPnH] }的值重载，这样可以允许无异常的情况下更新PWM。 模块的CCAPMn 寄存器PWMn 和 ECOMn 位必须置位以使能PWM模式。

使用9位比较，输出的占空比可以真正实现从0%到100%可调。占空比计算公式如下：

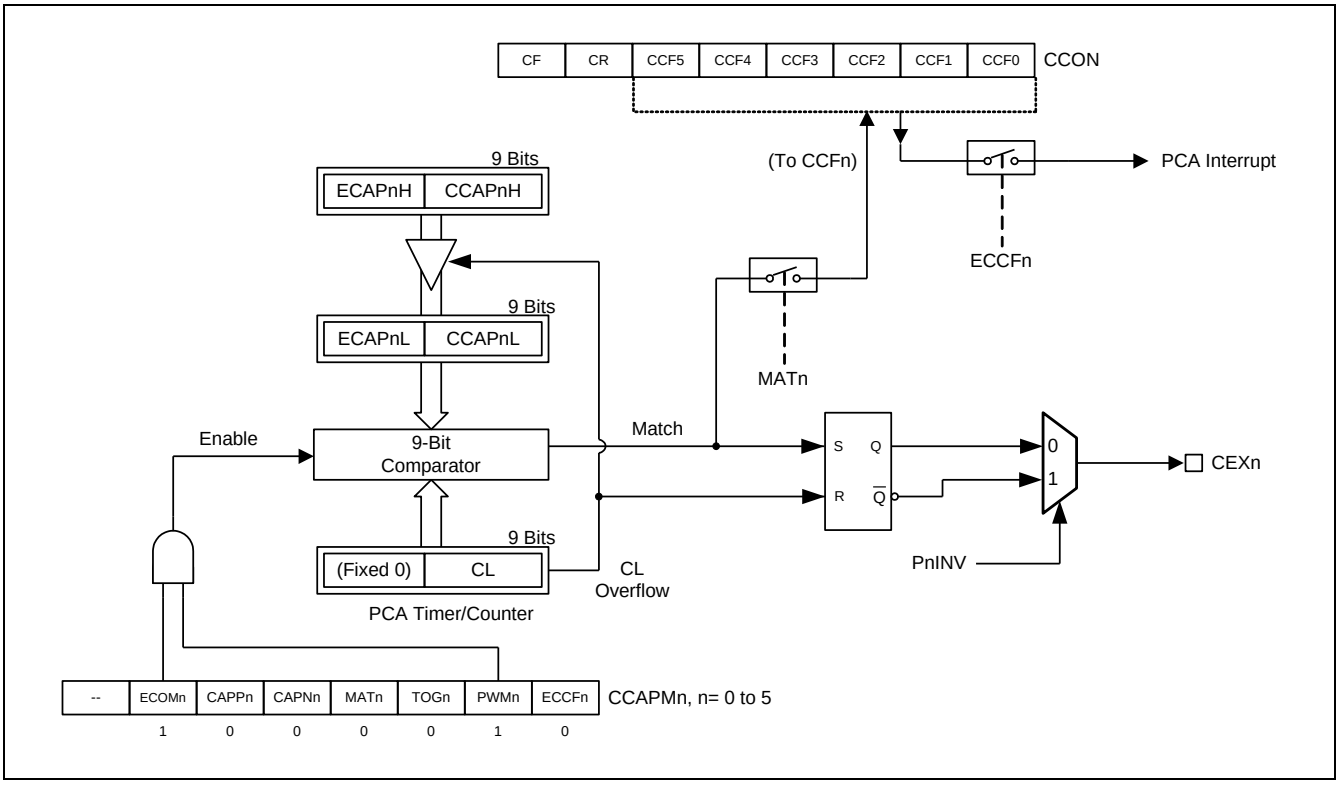
占空比 = 1 - { ECAPnH, [CCAPnH] } / 256。

这里， *[CCAPnH]* 是CCAPnH 寄存器的值， *ECAPnH*（ PCAPWMn 寄存器的第1位） 是1位值。所以， *{ ECAPnH, [CCAPnH] }* 组成了9位比较器用的9位值。

例如，

a. 若 ECAPnH=0 且 CCAPnH=0x00 (即9位值, 0x000), 占空比100%。
b. 若 ECAPnH=0 且 CCAPnH=0x40 (即9位值, 0x040), 占空比是75%。
c. 若 ECAPnH=0 且 CCAPnH=0xC0 (即9位值, 0x0C0), 占空比25%。
d. 若 ECAPnH=1 且 CCAPnH=0x00 (即9位值, 0x100), 占空比是0%。

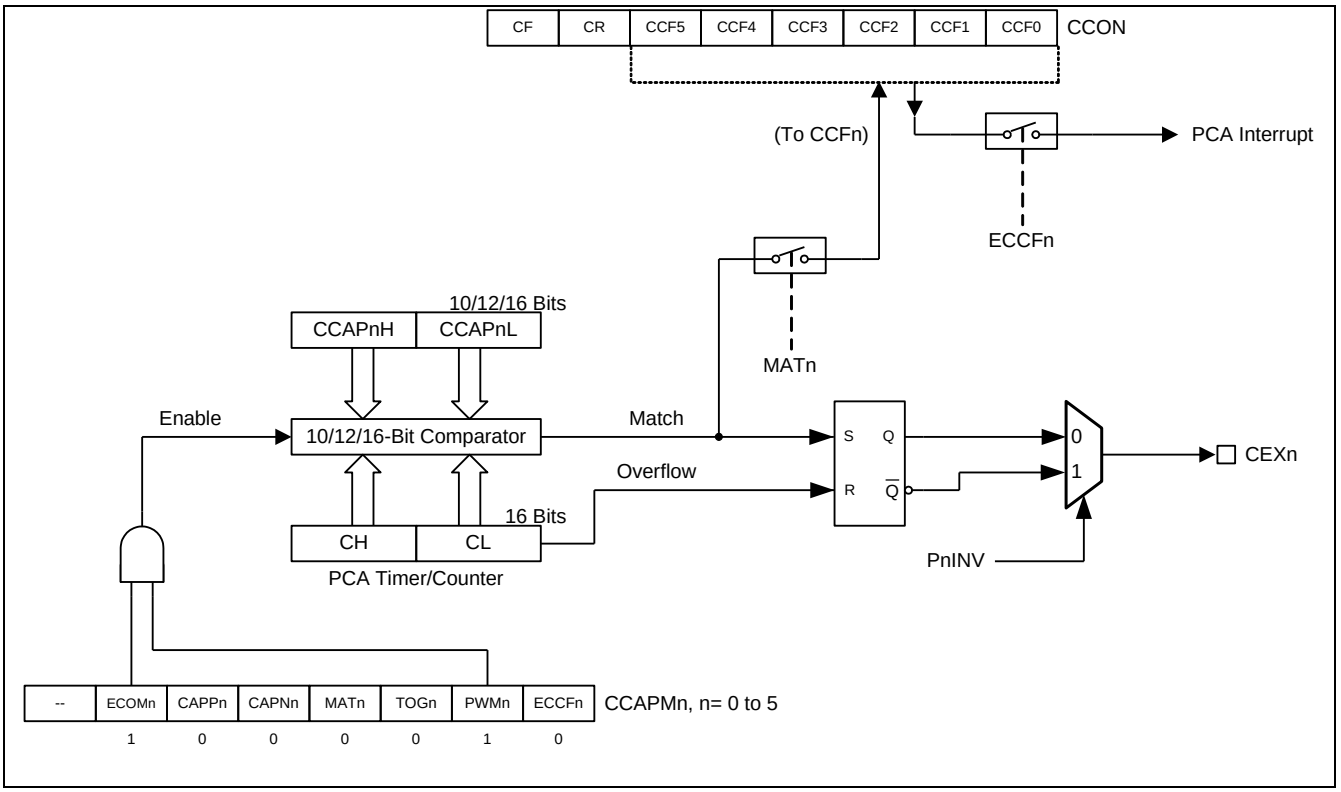
图 14-7. PCA PWM 模式



13.4.5. 增强 PWM 模式

MA805-64 提供可变的 PWM 模式以增强控制能力。有额外的 10/12/16 位 PWM 被分配给每一路及每一路 PWM 有不同的分辨率操作能力。

图 14-8. PCA 增强 PMW 模式



PCAPWMn: PWM 模式辅助寄存器, n=0~5

SFR 页 =全部

SFR 地址 = 0xF2~0xF7 上电+复位值 = 0000-0000

7	6	5	4	3	2	1	0
PnRS1	PnRS0	PnPS2	PnPS1	PnPS0	PnINV	ECAPnH	ECAPnL
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7~6: PnRS1~0, PWMn 分辨率设置 1~0.
00: 8 位 PWMn, 当{CH,CL} is 0xXXFF → 0xXX00 时溢出激活
01: 10 位 PWMn, 当{CH,CL} is 0xXx3FF → 0xXx[00]00 时溢出激活
10: 12 位 PWMn, 当{CH,CL} is 0xXxFFF → 0xXx000 时溢出激活
11: 16 位 PWMn, 当{CH,CL} is 0xFFFF → 0x0000 时溢出激活

Bit 5~3: PnPS2~0, PWMn 开始相位设置 2~0.
000: 使能的 PWM 通道在 0 度开始，并在数字比较器匹配时结束
001: 使能的 PWM 通道在 90 度开始，并在数字比较器匹配时结束
010: 使能的 PWM 通道在 180 度开始，并在数字比较器匹配时结束
011: 使能的 PWM 通道在 270 度开始，并在数字比较器匹配时结束
100: 使能的 PWM 通道在 120 度开始，并在数字比较器匹配时结束
101: 使能的 PWM 通道在 240 度开始，并在数字比较器匹配时结束
110: 使能的 PWM 通道在 60 度开始，并在数字比较器匹配时结束
111: 使能的 PWM 通道在 300 度开始，并在数字比较器匹配时结束

Bit 2: PnINV, 在 CEXn 上反向 PWM 输出
0: 不反向 PWM 输出
1: 反向 PWM 输出

Bit 1: ECAPnH: 扩展最高位, 联合 CCAPnH 形成 9 位寄存用于 8 位的 PWM 模式。 当然对于 10/12/16 位 PWM, 它将成为第 11/13/17 位寄存器

Bit 0: ECAPnL: 扩展最高位，联合 CCAPnL 形成 9 位寄存用于 8 位的 PWM 模式。 当然对于 10/12/16 位 PWM, 它将成为第 11/13/17 位寄存器

CMOD: PCA 计数器模式寄存器

SFR 页 =全部

SFR 地址 = 0xD9 上电+复位值 = 0xxx-x000

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

FEOV: 最大的计数器{CL} 值 FE

FEOV=0 最大的 CL 计数器值为 FF

FEOV=1 最大的 CL 计数器值为 FE

13.5. PCA 示例代码

(1). 功能需求: 设置 PWM2/PWM3 输出占空比 25% 和 75% 的波形

汇编语言代码范例:

```
PWM2        EQU     02h
ECOM2       EQU     40h
PWM3        EQU     02h
ECOM3       EQU     40h


PWM2_PWM3:
    MOV     CCON,#00H                   ; 停止 CR
    MOV     CMOD,#02H                       ; PCA 时钟源 = 系统时钟 / 2

    MOV     CCAPM2, #(ECOM2 + PWM2) ; 使能 PCA 模块 2 (PWM 模式)
    MOV     CCAP2H,#0C0H                    ; 占空比为 25%

    MOV     CCAPM3, #(ECOM3 + PWM3) ; 使能 PCA 模块 3 (PWM 模式)
    MOV     CCAP3H,#40H                     ; 占空比为 75%
    ;
    SETB    CR                              ; 启动 PCA
```

C 语言代码范例:

```
#define     PWM2            EQU     0x02
#define     ECOM2           EQU     0x40
#define     PWM3            EQU     0x02
#define     ECOM3           EQU     0x40

void main(void)
{
    // set PCA
    CCON = 0x00;                                        //禁止 PCA & 清 CCF0, CCF1, CCF2, CCF3,CF 标志
    CMOD = 0x02;                                        // PCA 时钟源 = 系统时钟 / 2

    CCAPM2 |= (ECOM2 | PWM2);                   //使能 PCA 模块 2 (PWM 模式)
    CCAP2H = 0xC0;                      //占空比为 25%

    CCAPM3 |= (ECOM3 | PWM3);                   //使能 PCA 模块 3 (PWM 模式)
    CCAP3H = 0x40;                          //占空比为 75 %
    //---------------------------------------------
    CR = 1;                                             //启动 PCA

    while (1);
}
```

(2). 功能需求: 设置模块 0 为 CEX0 上升沿捕捉模式，模块 1 为 PWM 输出占空比为 25%的波形

汇编语言代码范例:

```
PWM1        EQU         02h
CAPP0       EQU     20h
ECOM1       EQU         40h

PWM2_PWM3:
    MOV     CCON,#00H               ; 停止 CR
    MOV     CMOD,#02H                   ; PCA 时钟源 = 系统时钟 / 2

    ORL     CCAPM0, #CAPP0      ; 模块 0 为捕捉 CEX0 上升沿

    ORL     CCAPM1, #(ECOM1 + PWM1)     ; 模块 1 为 PWM 输出
    MOV     CCAP3H, #0C0h           ; 占空比 25 %
    SETB    CR                  ; 启动 PCA
```

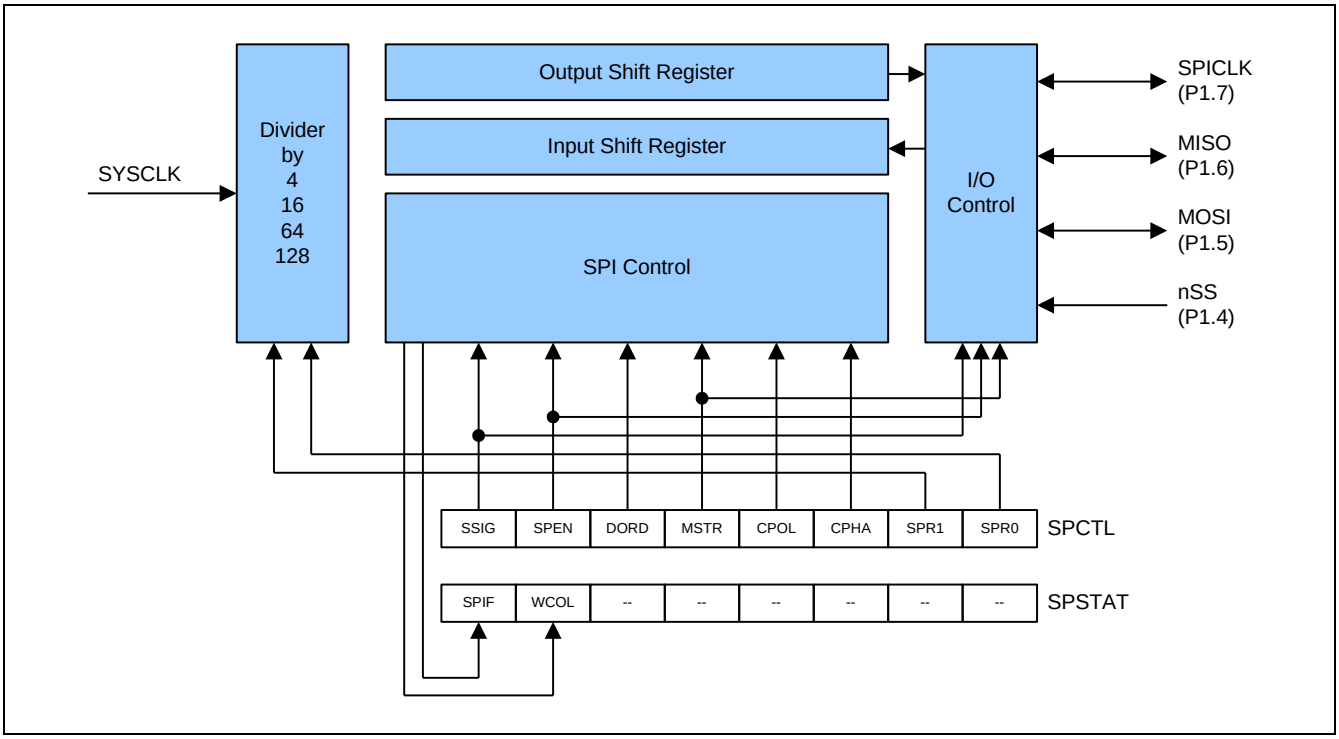
C 语言代码范例:

```
#define     PWM1            EQU     0x02
#define     CAPP0               EQU             0x20
#define     ECOM1           EQU     0x40


void main(void)
{
    // set PCA
    CCON = 0x00;                                    //禁止 PCA & 清 CCF0, CCF1, CCF2,CCF3,CF 标志
    CMOD = 0x02;                                    // PCA 时钟源 = 系统时钟 / 2

            CCAPM0 |= CAPP0;                    //模块 0 为捕捉 CEX0 上升沿

    CCAPM1 |= (ECOM1 | PWM1);               //模块 1 为 PWM 输出
    CCAP3H = 0xC0;                  //占空比 25 %

    CR = 1;                                         //启动 PCA

    while (1);
}
```

14. 串行外设接口（SPI）

MA805-64 提供了一个高速串行通讯接口（SPI）。SPI 接口是一种全双工、高速同步通讯总线，有两种操作模式：主机模式和从机模式。无论哪种模式，12MHz 系统时钟时支持高达 3Mbps 的通讯速度。MA805-64 的 SPI 状态寄存器（SPSTAT）有一个传送完成标志（SPIF）和写冲突标志（WCOL）。

图 15-1. SPI 方框图



SPI 接口有 4 个引脚：MISO (P1.6)， MOSI (P1.5)， SPICLK (P1.7) 和 nSS (P1.4)。

• SPICLK，MOSI 和 MISO 通常将两个或多个 SPI 设备连接在一起。数据从主机到从机使用 MOSI 引脚 (Master Out / Slave In 主出从入），从从机到主机使用 MISO 引脚(Master In / Slave Out 主入从出)。SPICLK 信号在主机模式时输出，从机模式时输入。若 SPI 接口禁用，即 SPEN (SPCTL.6) = 0，这些引脚可以作为普通 I/O 口使用。

• nSS 是从机选择端。典型配置中，SPI 主机可以使用其某个端口选择某一个 SPI 设备作为当前从机，一个 SPI 从机设备使用它的 nSS 引脚确定自己是否被选中。下面条件下 nSS 被忽略：

- 若 SPI 系统被禁用，即 SPEN (SPCTL.6) = 0 (复位值)；
- 若 SPI 作为主机运行，即 MSTR (SPCTL.4) = 1， 且 P1.4 (nSS) 被配置成输出；
- 若 nSS 被设置成忽略，即 SSIG (SPCTL.7) = 1， 这个端口作为普通 I/O 使用。

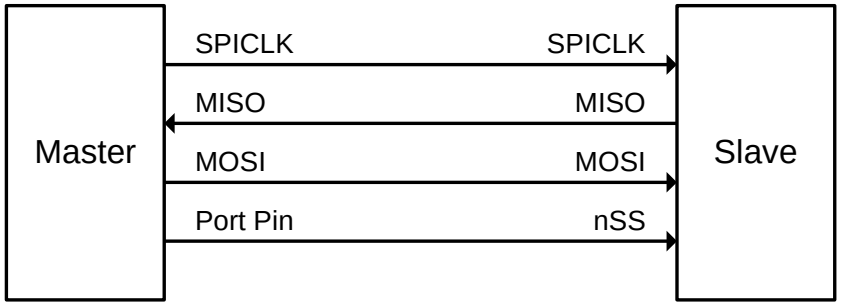
注意，即使 SPI 被配置成主机运行(MSTR=1)，它仍然可以被 nSS 引脚的低电平拉成从机(若 SSIG=0)，一旦发生这种情况，SPIF 位(SPSTAT.7)置位。(参见 15.2.3 节：nSS 引脚的模式改变)

14.1. 典型 SPI 配置

14.1.1. 单主机和单从机

对于主机：任何端口，包括 P1.4 (nSS)，都可以用来控制从机的 nSS 片选引脚。
对于从机：SSIG 为 ‘0’， nSS 引脚决定该设备是否被选中。

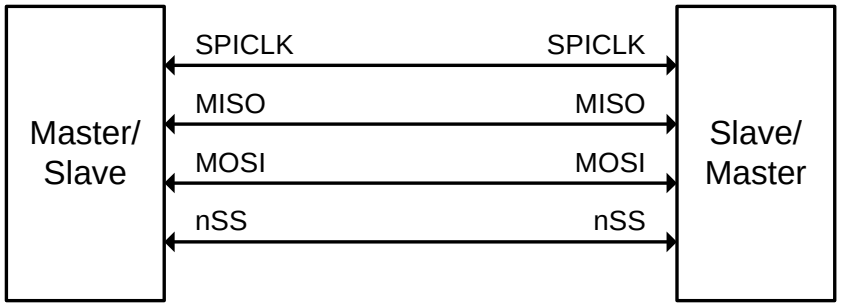
图 15-2. SPI 单主机和单从机配置



14.1.2. 双驱动器,可以是主机或从机

两个彼此连接的设备，均可成为主机或从机，没有 SPI 操作时，都可以被通过设置 MSTR=1， SSIG=0， P1.4 (nSS) 双向口配置成主机。任何一方要发起传输，它可以配置 P1.4 位输出并强行拉低，使另一个设备发生“被改成从机模式”事件。 (参见 15.2.3 节：nSS 引脚模式改变)

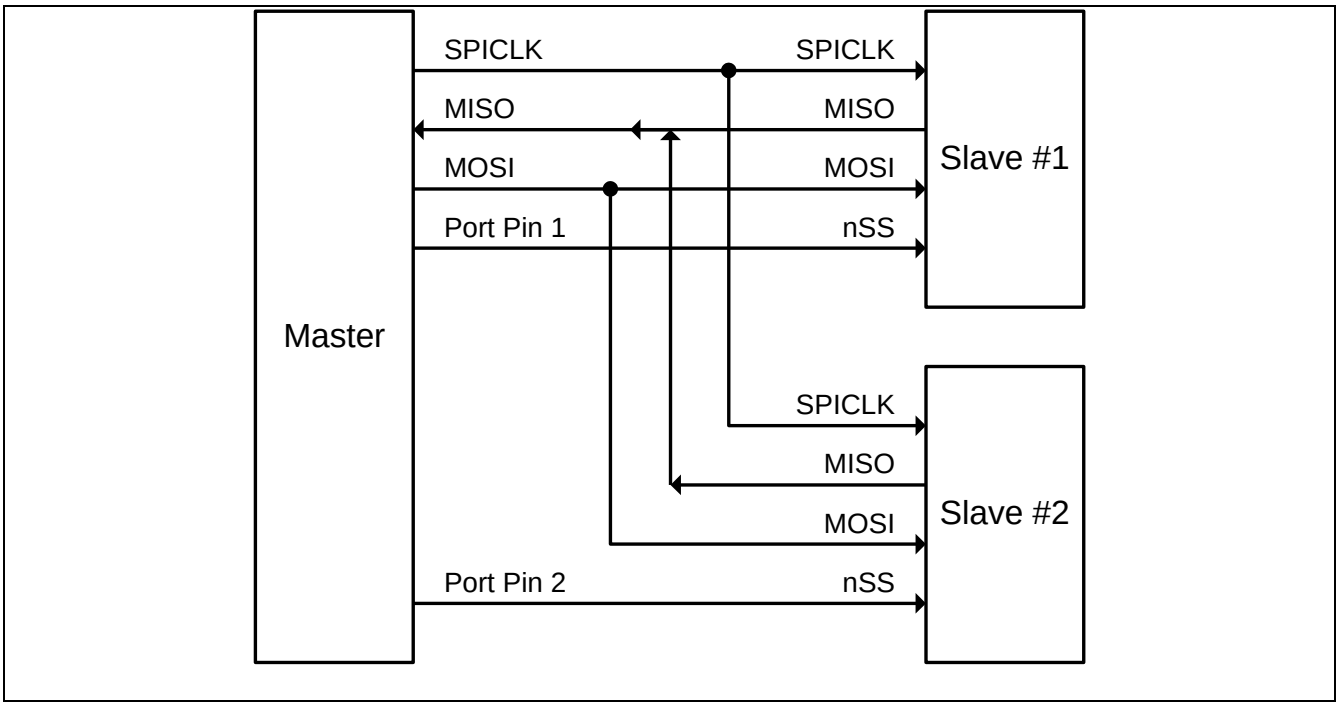
图 15-3. SPI 双驱动器，可以是主机或从机配置



14.1.3. 单主机和多从机

对于主机：任何端口，包括 P1.4 (nSS)，都可以用来控制从机的 nSS 片选引脚。
对于所有从机：SSIG 为 ‘0’，nSS 引脚决定该设备是否被选中。

图 15-4. SPI 单主机和多从机配置


Master
SPICLK
MISO
MOSI
Port Pin 1
Port Pin 2
SPICLK
MISO
MOSI
nSS
Slave #1
SPICLK
MISO
MOSI
nSS
Slave #2

14.2. 配置 SPI

表 15-1 显示了主/从机模式配置及使用方法和传输方向。

表 15-1. SPI 主从机选择

SPEN (SPCTL.6)	SSIG (SPCTL.7)	nSS -pin	MSTR (SPCTL.4)	模式	MISO 引脚	MOSI 引脚	SPICLK 引脚	备注
0	X	X	X	SPI 禁用	输入	输入	输入	P1.4~P1.7 用作通用 I/O
1	0	0	0	从机 (被选中)	输出	输入	输入	被选择为从机
1	0	1	0	从机 (未被选中)	高阻	输入	输入	未被选中
1	0	0	1 → 0	从机 (通过模式改变)	输出	输入	输入	若 nSS 被拉低,MSTR 被硬件自动清‘0’，模式被改为从机
1	0	1	1	主机 (待机)	输入	高阻	高阻	MOSI 和 SPICLK 在主机待机时被置为高阻，以防止总线冲突。
				主机 (活动)		输出	输出	MOSI 和 SPICLK 在主机活动时被上拉。
1	1	X	0	从机	输出	输入	输入	
1	1	X	1	主机	输入	输出	输出	

“X” 表示”无需关心”。

14.2.1. 从机注意事项

当 CPHA = 0 时，SSIG 必须为 0 且 nSS 引脚必须在每次串行字节传输前负跳变，传输结束恢复正常高电平。
注意 SPDAT 寄存器不能在 nSS 引脚低电平时写入；CPHA = 0，SSIG=1 的操作是未定义的。

当 CPHA =1 时，SSIG 可以为 0 或 1。若 SSIG=0，nSS 引脚可以在每次成功传输之间保持低电平（可以一直拉低），这种格式有时非常适合单固定主从机配置应用。

14.2.2. 主机注意事项

SPI 通讯中，传输总是由主机发起 。若 SPI 使能(SPEN=1)并作为主机运行，写入 SPI 数据寄存器(SPDAT) 数据即可启动 SPI 时钟生成器和数据传输器，大约半个到 1 个 SPI 位时间后写入 SPDAT 的数据开始出现在 MOSI 线上。

在开始传输之前，主机通过拉低相应 nSS 引脚选择一个从机作为当前从机。写入 SPDAT 寄存器德数据从主机 MOSI 引脚移出，同时从从机 MISO 移入主机 MISO 的数据也写入到主机的 SPDAT 寄存器中。

移出 1 字节后，SPI 时钟发生器停止，置传输完成标志 SPIF，若 SPI 中断使能则生成一个中断。主机 CPU 和从机 CPU 中的两个移位寄存器可以看成一个分开的 16 位环形移位寄存器，数据从主机移到从机同时数据也从从机移到主机。这意味着，在一次传输过程中，主从机数据进行了交换。

14.2.3. nSS 引脚的模式改变

若 SPEN=1、SSIG=0、MSTR=1 且 nSS 引脚=1，SPI 使能在主机模式，这种情况下，其它主机可以将 nSS 引脚拉低来选择该设备为从机并开始发送数据过来。为避免总线冲突，该 SPI 设备成为一个从机，MOSI 和 SPICLK 引脚被强制为输入端口，MISO 成为输出端口，SPSTAT 中 SPIF 标志置位，若此时 SPI 中断使能，则还会产生一个 SPI 中断。用户软件必须经常去检查 MSTR 位，若该位被从机选择清零而用户又想要继续保持该 SPI 主机模式，用户必须再次设置 MSTR 位，否则，将处于从机模式。

14.2.4. 数据冲突

SPI 在发送方向是单缓冲的，而在接收方向是双缓冲的。发送数据直到上一次数据发送完成后才能写入移位寄存器，数据发送过程中写入数据寄存器就会使 WCOL(SPSTAT.6) 置位来表明数据冲突。这种情况下，正在发送的数据继续发送，而刚写入数据寄存器造成冲突的数据就会丢失。

写冲突对于主从机都有可能发生，对于主机，这种现象并不多见，因为主机控制着数据的传送；然而对于从机，由于没有控制权，因此很可能会发生。

对于数据接收，接收的数据被传输到一个并行读数据缓冲器中，以便于移位寄存器再能接收新的字节。然而，接收的数据必须在下个字节完全移入前从数据寄存器 SPDAT 读出，否则前一个数据就会丢失。

WCOL 使用软件向其位写入’1’来清零。

14.2.5. SPI 时钟速率选择

SPI 时钟频率选择（主机模式）使用 SPCTL 寄存器的 SPR1 和 SPR0 位来设置，如表 15-2 所示

表 15-2. SPI 串行时钟速率

SPR1	SPR0	SPI 时钟速率 @ SYSCLK=12MHz	SYSCLK 分频
0	0	3 MHz	4
0	1	750 KHz	16
1	0	187.5 KHz	64
1	1	93.75 KHz	128

这里，SYSCLK 是系统时钟。

14.3. 数据模式

时钟相位位(CPHA) 位可以让用户设定数据采样和改变时的时钟沿。时钟极性位 CPOL 可以让用户设定时钟极性。下面图例显示了不同时钟相位、极性设置下 SPI 通讯时序。

图 15-5. SPI 从机传送格式,CPHA=0

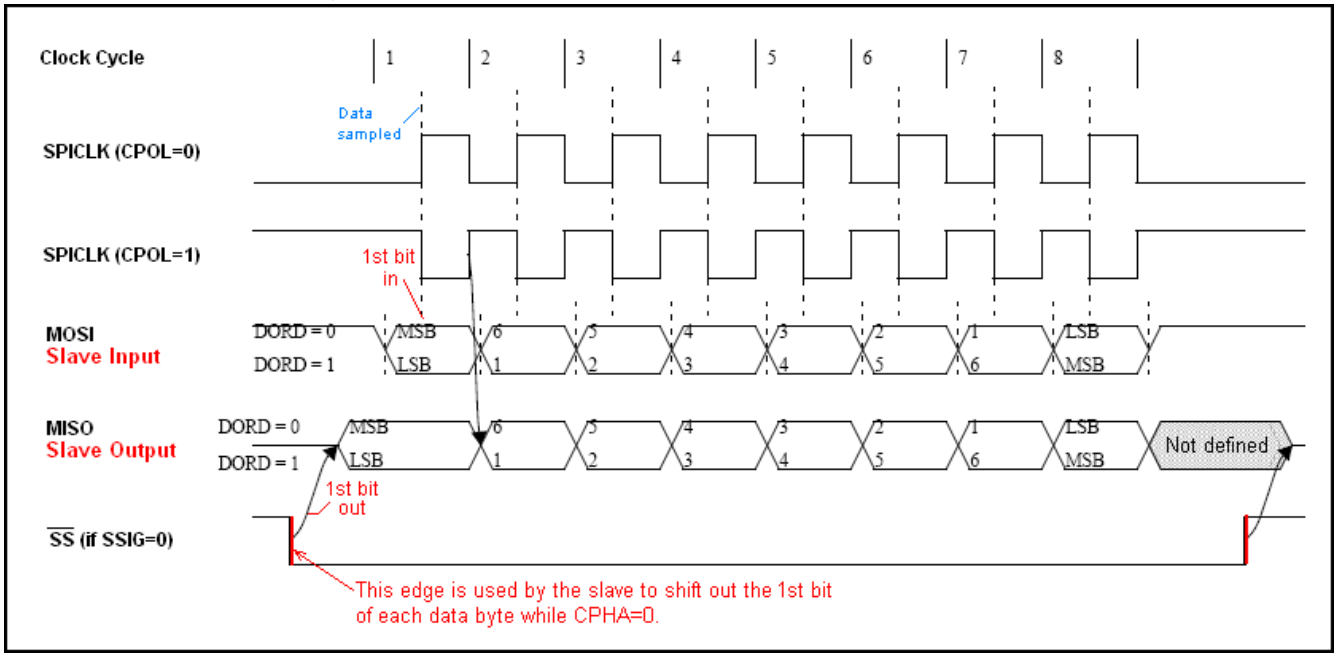


图 15-6. SPI 从机传送格式,CPHA=1

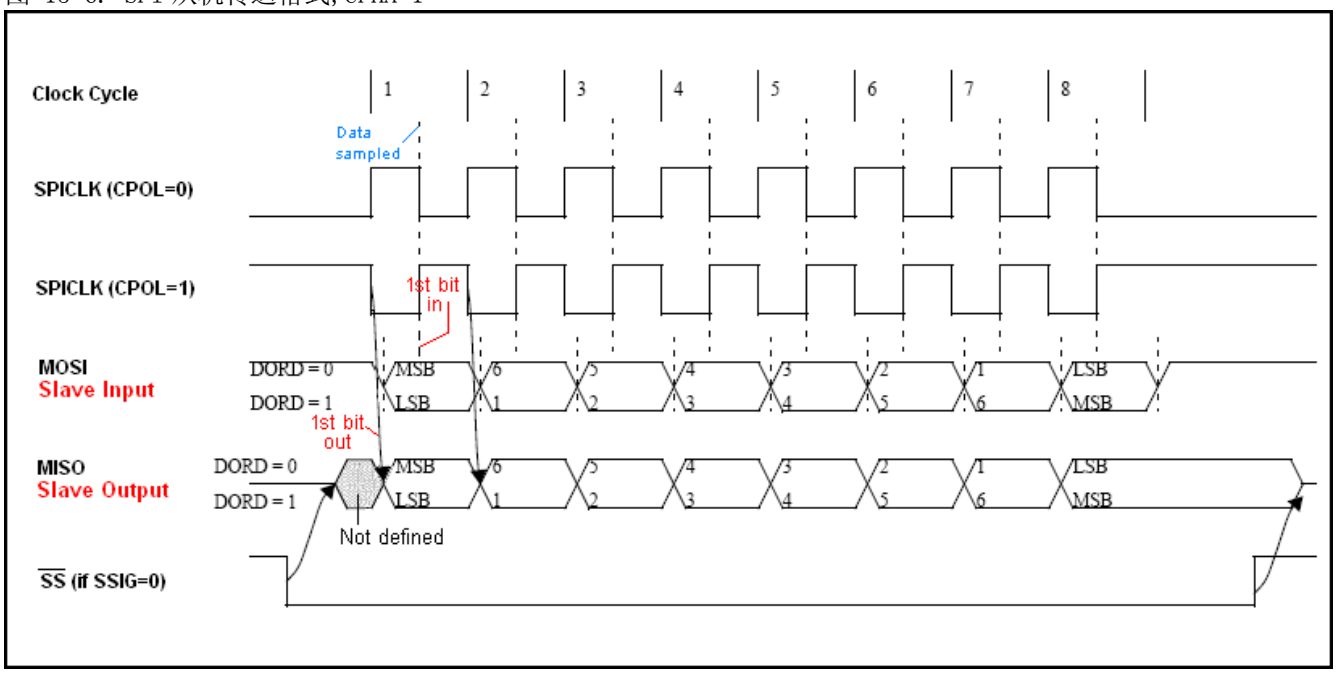


图 15-7. SPI 主机传送格式,CPHA=0

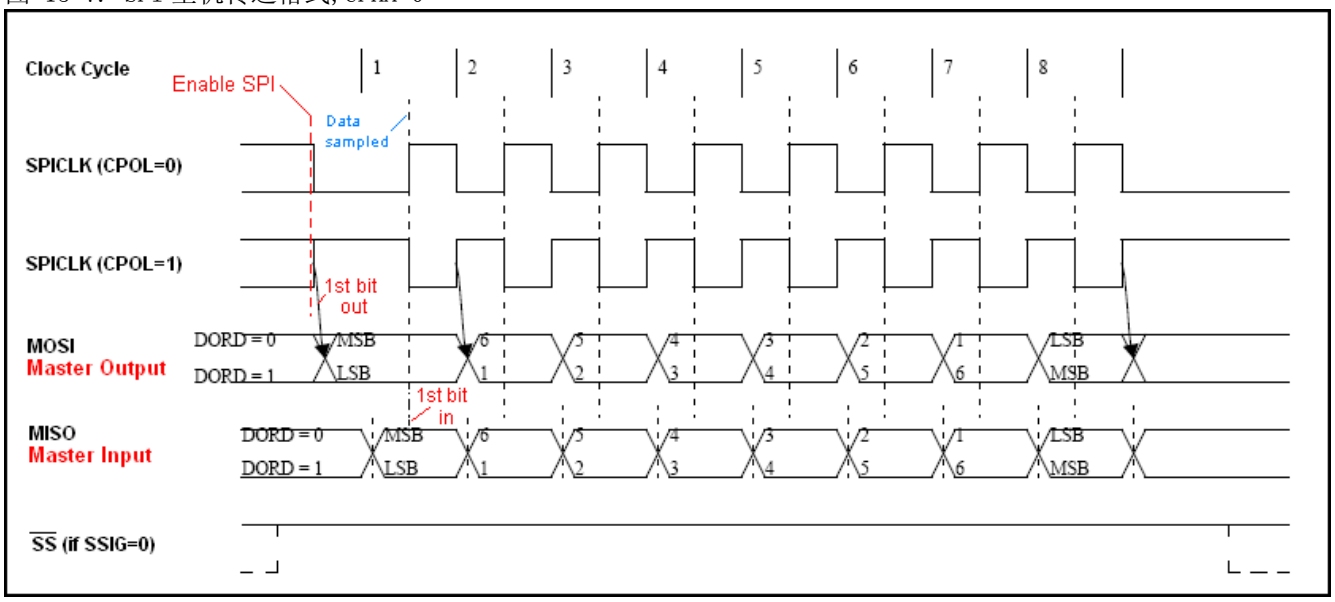
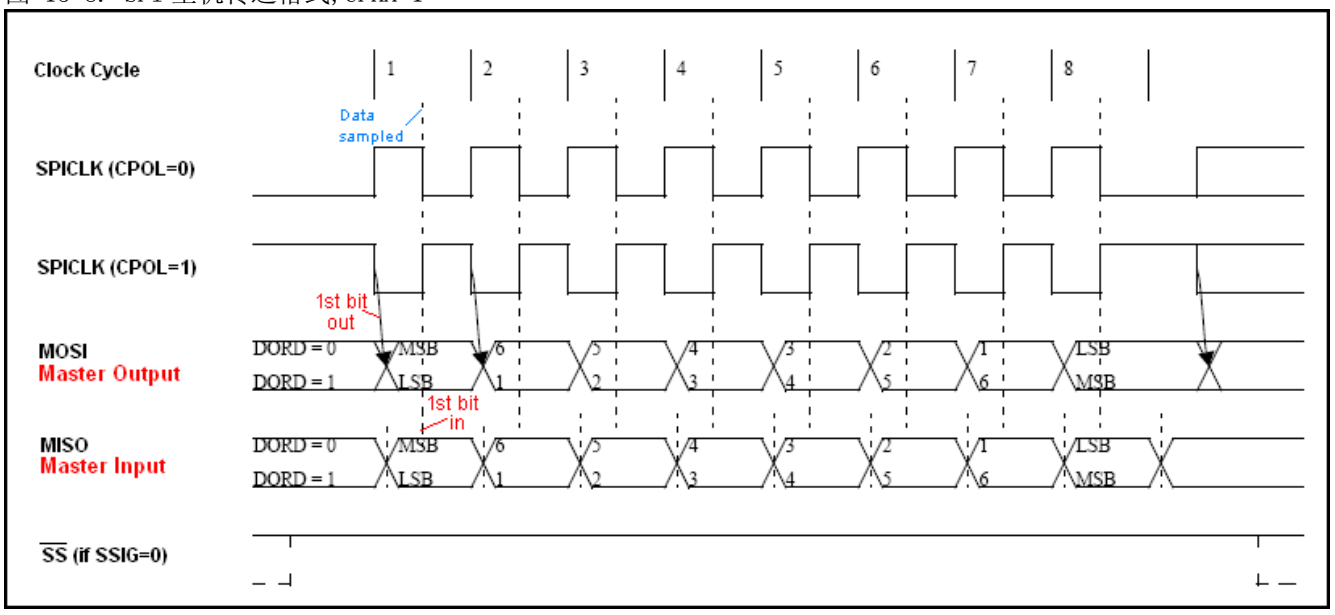


图 15-8. SPI 主机传送格式,CPHA=1



14.4. SPI 寄存器

下面是 SPI 操作相关的特殊功能寄存器

SPCON: SPI 控制寄存器

SFR 页 =全部

SFR 地址 = 0x85 复位= 0000-0100

7	6	5	4	3	2	1	0
SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7: SSIG, nSS 被忽略
0: nSS 引脚决定该设备是主机还是从机
1: MSTR 位决定该设备是主机还是从机

Bit 6: SPEN, SPI 使能
0: SPI 接口禁用，所有 SPI 引脚可作为通用 I/O 口使用
1: SPI 功能打开

Bit 5: DORD, SPI 数据顺序
0: 传送数据时先传数据字节最高位（MSB）
1: 传送数据时先传数据字节最低位（LSB）

Bit 4: MSTR, 主机/从机模式选择
0: 选择从 SPI 模式
1: 选择主 SPI 模式

Bit 3: CPOL, SPI I 时钟极性选择
0: SPICLK 待机是为低电平，SPICLK 时钟脉冲前沿是上升沿，而后沿是下降沿
1: SPICLK 待机是为高电平，SPICLK 时钟脉冲前沿是下降沿，而后沿是上升沿

Bit 2: CPHA, SPI 时钟相位选择
0: nSS 引脚低电平 (SSIG=0)开始放数据并在 SPICLK 后沿改变数据。 数据在 SPICLK 的前沿采样
1: SPICLK 脉冲前沿放数据，后沿采样
(*注 ： 若 SSIG=1, CPHA 必须不为 0，否则操作是未被定义的*)

Bit 1~0: SPR1-SPR0, SPI 时钟速率选择 (主机模式)
00: SYSCLK/4
01: SYSCLK/16
10: SYSCLK/64
11: SYSCLK/128 (这里的 SYSCLK 是系统时钟)

SPSTAT: SPI 状态寄存器

SFR 页 =全部

SFR 地址 = 0x84 复位= 00XX-XXXX

7	6	5	4	3	2	1	0
SPIF	WCOL	--	--	--	--	--	--
R/W	R/W	R	R	R	R	R	R

Bit 7: SPIF, SPI 传输完成标志
0: SPIF 标志通过软件在该位写入“1”来清零

1: 当一次串行传输完成时，SPIF 位置位，同时若 SPI 中断允许，会产生一个中断。若 nSS 引脚在主机模式下被拉低且 SSIG=0，SPIF 位也会置位以表明“模式改变”

Bit 6: WCOL, SPI 写冲突标志
0: WCOL 标志通过软件在该位写入“1”来清零
1: SPI 数据寄存器 SPDAT 在数据传输过程中被写入时，WCOL 置位(参见 15.2.4 节 :写 冲 突)

Bit 5~0: 保留

SPDAT: SPI 数据寄存器

SFR 页 =全部
SFR 地址 = 0x86 复位= 0000-0000

7	6	5	4	3	2	1	0
(MSB)							(LSB)
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

SPDAT 有两个物理缓冲器供传输过程中写入和读取，一个写入缓冲器，一个读取缓冲器

14.5. SPI 示例代码

(1). 功能需求: SPI 主机读和写，采样数据在上升沿并且时钟前沿是上升沿。

汇编语言范例:

```
CPHA    EQU     04h
CPOL    EQU     08h
MSTR    EQU     10h
SPEN    EQU     40h
SSIG    EQU     80h
SPIF    EQU     80h

Initial_SPI:                        ;初始化 SPI
    ORL     SPICTL, #(SSIG + SPEN + MSTR)       ;使能 SPI 主模式
    RET

SPI_Write:
    MOV     SPIDAT, R7              ;写自变量 R7
  wait_write:
    MOV     A, SPISTAT
    JNB     ACC.7, wait_write       ;等待传送完成
    ANL     SPISTAT, #(0FFh - SPIF) ;清除 SPI 中断标志
    RET

SPI_Read:
    MOV     SPIDAT, #0FFh           ;触发 SPI 读
  wait_read:
    MOV     A, SPISTAT
    JNB     ACC.7, wait_read        ;等待读完成
    ANL     SPISTAT, #(0FFh - SPIF)         ;清楚 SPI 中断标志
    MOV     A, SPIDAT               ;移动读的数据到 A
    RET
```

C 语言范例：

```
#define     CPHA        0x04
#define     CPOL        0x08
#define     MSTR        0x10
#define     SPEN        0x40
```

```
#define     SSIG            0x80
#define     SPIF            0x80

void Initial_SPI(void)
{
    SPICTL |= (SSIG | SPEN | MSTR);                 // 使能 SPI 主机模式
}

void SPI_Write(unsigned char arg)
{
    SPIDAT = arg;                                   //写自变量
    while(!(SPISTAT & SPIF));                       //等待传送完成
    SPISTAT &= ~SPIF;                               //清楚 SPI 中断标志
}

unsigned char SPI_Read(void)
{
    SPIDAT = 0xFF;                              //触发 SPI 读
    while(!SPISTAT & SPIF);                     //等待传送完成
    SPISTAT &= ~SPIF;                               //清除 SPI 中断标志
    return SPIDAT;
}
```

(2). 功能需求： SPI 主机读和写，采样数据在上升沿和时钟前沿为下降沿。

汇编语言范例:

```
CPHA    EQU     04h
CPOL    EQU     08h
MSTR        EQU         10h
SPEN        EQU             40h
SSIG        EQU             80h
SPIF    EQU     80h

Initial_SPI:                                    ;初始化 SPI
    ORL     SPICTL, #(SSIG + SPEN + MSTR + CPOL)                    ;使能 SPI 主机模式
    RET
```

```
SPI_Write:
    MOV     SPIDAT, R7                          ;写自变量 R7
  wait_write:
    MOV     A, SPISTAT
    JNB     ACC.7, wait_write                   ;等待传送完成
    ANL     SPISTAT, #(0FFh - SPIF)             ;清除 SPI 中断标志
    RET

SPI_Read:
    MOV     SPIDAT, #0FFh                       ;触发 SPI 读
  wait_read:
    MOV     A, SPISTAT
    JNB     ACC.7, wait_read                    ;等待读完成
    ANL     SPISTAT, #(0FFh - SPIF)                         ;清除 SPI 中断标志
    MOV     A, SPIDAT                           ;转移读的数据到 A
    RET
```

C 语言范例:

```
#define     CPHA        0x04
#define     CPOL        0x08
#define     MSTR            0x10
#define     SPEN        0x40
#define     SSIG        0x80
#define     SPIF        0x80

void Initial_SPI(void)
{
    SPICTL |= (SSIG | SPEN | MSTR | CPOL);      // 使能 SPI 主机模式
}

void SPI_Write(unsigned char arg)
{
    SPIDAT = arg;                               //写自变量
    while(!(SPISTAT & SPIF));                   //等待传送完成
    SPISTAT &= ~SPIF;                           //清除 SPI 中断标志
}
```

```
unsigned char SPI_Read(void)
{
    SPIDAT = 0xFF;                          //触发 SPI 读
    while(!SPISTAT & SPIF);                 //等待读完成
    SPISTAT &= ~SPIF;                           //清除 SPI 中断标志
    return SPIDAT;
}
```

(3). 功能需求: SPI 主机读和写,采样数据在下降沿并且时钟前言是上升沿。

汇编语言代码范例:

```
CPHA    EQU     04h
CPOL    EQU     08h
MSTR    EQU     10h
SPEN    EQU     40h
SSIG    EQU     80h
SPIF    EQU     80h

Initial_SPI:                                ; 初始化 SPI
    ORL     SPICTL, #(SSIG + SPEN + MSTR + CPHA)        ; 使能 SPI 主机模式
    RET

SPI_Write:
    MOV     SPIDAT, R7                      ; 写自变量 R7
  wait_write:
    MOV     A, SPISTAT
    JNB     ACC.7, wait_write               ; 等待传送完成
    ANL     SPISTAT, #(0FFh - SPIF)         ; 清除 SPI 中断标志
    RET

SPI_Read:
    MOV     SPIDAT, #0FFh                   ; 触发 SPI 读
  wait_read:
    MOV     A, SPISTAT
    JNB     ACC.7, wait_read                ; 等待读完成
    ANL     SPISTAT, #(0FFh - SPIF)         ; 清除 SPI 中断标志
    MOV     A, SPIDAT                       ;转移读的数据到 A
    RET
```

C 语言代码范例：

```
#define     CPHA        0x04
#define     CPOL        0x08
#define     MSTR        0x10
#define     SPEN        0x40
#define     SSIG        0x80
```

```
#define       SPIF                0x80

void Initial_SPI(void)
{
    SPICTL |= (SSIG | SPEN | MSTR | CPHA);              //使能 SPI 主机模式
}

void SPI_Write(unsigned char arg)
{
    SPIDAT = arg;                                       //写自变量
    while(!(SPISTAT & SPIF));                           //等待传送完成
    SPISTAT &= ~SPIF;                                   //清除 SPI 中断标志
}

unsigned char SPI_Read(void)
{
    SPIDAT = 0xFF;                                  //触发 SPI 读
    while(!SPISTAT & SPIF);                         //等待读完成
    SPISTAT &= ~SPIF;                                   //清除 SPI 中断标志
    return SPIDAT;
}
```

(4). 功能需求: SPI 主机读和写,采样数据在下降沿并且时钟前言是下降沿。

汇编语言代码范例:

```
CPHA    EQU     04h
CPOL    EQU     08h
MSTR    EQU     10h
SPEN    EQU     40h
SSIG    EQU     80h
SPIF    EQU     80h

Initial_SPI:                                    ;初始化 SPI
    ORL     SPICTL, #(SSIG + SPEN + MSTR + CPOL + CPHA)     ;使能 SPI 主机模式
    RET

SPI_Write:
    MOV     SPIDAT, R7                          ;写自变量 R7
  wait_write:
    MOV     A, SPISTAT
    JNB     ACC.7, wait_write                   ; 等待传送完成
    ANL     SPISTAT, #(0FFh - SPIF)             ; 清除 SPI 中断标志
    RET

SPI_Read:
    MOV     SPIDAT, #0FFh                       ;触发 SPI 读
  wait_read:
    MOV     A, SPISTAT
    JNB     ACC.7, wait_read                    ; 等待读完成
    ANL     SPISTAT, #(0FFh - SPIF)                     ; 清除 SPI 中断标志
    MOV     A, SPIDAT                           ; 转移读的数据到 A
    RET
```

C Code Example:

```
#define     CPHA        0x04
#define     CPOL        0x08
#define     MSTR        0x10
#define     SPEN        0x40
#define     SSIG        0x80
```

```
#define        SPIF                0x80

void Initial_SPI(void)
{
    SPICTL |= (SSIG | SPEN | MSTR | CPOL | CPHA);                    //使能  SPI  主机模式
}

void SPI_Write(unsigned char arg)
{
    SPIDAT = arg;                                                    //写自变量
    while(!(SPISTAT & SPIF));                                        //等待传送完成
    SPISTAT &= ~SPIF;                                                //清除  SPI  中断标志
}

unsigned char SPI_Read(void)
{
    SPIDAT = 0xFF;                                                   //触发 SPI  读
    while(!SPISTAT & SPIF);                                          //等待读完成
    SPISTAT &= ~SPIF;                                                //清除  SPI  中断标志
    return SPIDAT;
}
```

15. 键盘中断(KBI)

键盘中断功能主要用于当 P2 口等于或不等于某个值时产生一个中断，这个功能可以用作总线地址识别或键盘键码识别。

有 3 个特殊功能寄存器与此功能相关。键盘中断掩码寄存器(KBMASK) 用来定义 P2 口哪些引脚可以产生中断；键盘模式寄存器(KBPATN)用来定义与 P2 口值进行比较的值，比较匹配时硬件置键盘中断控制寄存器(KBCON)中的键盘中断标志(KBIF) ，若 EIE1 中的 EKBI 中断允许且 EA=1，则还会产生一个中断。键盘中断控制寄存器(KBCON)中的 PATN_SEL 位用来定义比较是“相等”还是“不等”匹配。

为了使用键盘中断作为“键盘”中断，用户需要设置 KBPATN=0xFF 和 PATN_SEL=0 (不相等)，然后将任意按键连接到 KBMASK 寄存器定义的相应 P2 口，按下时硬件就会置位中断标志 KBIF，并当中断使能时产生中断。这个中断可以将 CPU 从空闲模式或掉电模式下唤醒。这个功能在手持设备，电池供电系统等要求低功耗而且易用的设备上特别有用。

15.1. 键盘寄存器

下面是键盘中断操作相关的特殊功能寄存器：

KBPATN：键盘模式寄存器

SFR 页 =全部

SFR 地址 = 0xD5 复位= 1111-1111

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

Bit 7~0: KBPATN.7~0: 键盘模式

KBCON：键盘控制寄存器

SFR 页 =全部

SFR 地址 = 0xD6 复位= XXXX-XX00

7	6	5	4	3	2	1	0
--	--	--	--	--	--	PATN_SEL	KBIF
R	R	R	R	R	R	R/W	R/W

Bit 7~2: 保留.

Bit 1: PATN_SEL, 模式匹配极性选择
0: 键盘输入不等于 KBPATN 中用户定义的模式时产生中断 T
1: 键盘输入等于 KBPATN 中用户定义模式时产生中断

Bit 0: KBIF, 键盘中断标志
0: 必须由软件写入‘0’来清零
1: P2 端口值匹配 KBPATN、KBMASK、PATN_SEL 设置条件时置位

KBMASK：键盘中断掩码寄存器

SFR 页 =全部

SFR 地址 = 0xD7 复位= 0000-0000

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

KBMASK.7: 置位时，使能 P2.7 作为键盘中断源 (KBI7)
KBMASK.6: 置位时，使能 P2.6 作为键盘中断源 (KBI6)
KBMASK.5: 置位时，使能 P2.5 作为键盘中断源 (KBI5)
KBMASK.4: 置位时，使能 P2.4 作为键盘中断源 (KBI4)
KBMASK.3: 置位时，使能 P2.3 作为键盘中断源 (KBI3)
KBMASK.2: 置位时，使能 P2.2 作为键盘中断源 (KBI2)
KBMASK.1: 置位时，使能 P2.1 作为键盘中断源 (KBI1)
KBMASK.0: 置位时，使能 P2.0 作为键盘中断源 (KBI0)

15.2. 键盘中断示例代码

(1). 功能需求：设置 P2.3~P2.0 键盘中断源 *KBI*

汇编语言代码范例:

```
      Jmp       main;
   ORG       0006Bh
KBI_ISR:                                          ;
      PUSH      ACC                               ;

      ;To do    KBI key checking……

      ANL       KBCON,#(0FFh - KBIF)              ;清除 KBI 中断标志位 flag (写“0”)
      POP       ACC                               ;
      RETI                                        ;

main:
      ANL       P2M1,#0F0h                        ;设置 P2.3~P2.0 为输入口模式
      ORL       P2M0,#00Fh                        ;设置 P2.3~P2.0 为输入口模式

      MOV       KBMASK,#00Fh                      ;设置 P2.3~P2.0 的键盘中断 KBI 功能

      ANL       KBCON,# (0FFh - KBIF)             ;清除 KBI 中断标志位 flag (写“0”)

      ORL       AUXIE,#EKBI                       ;使能键盘中断
      SETB      EA                                ;使能全局中断

       Jmp      $;
```

C Code Example:

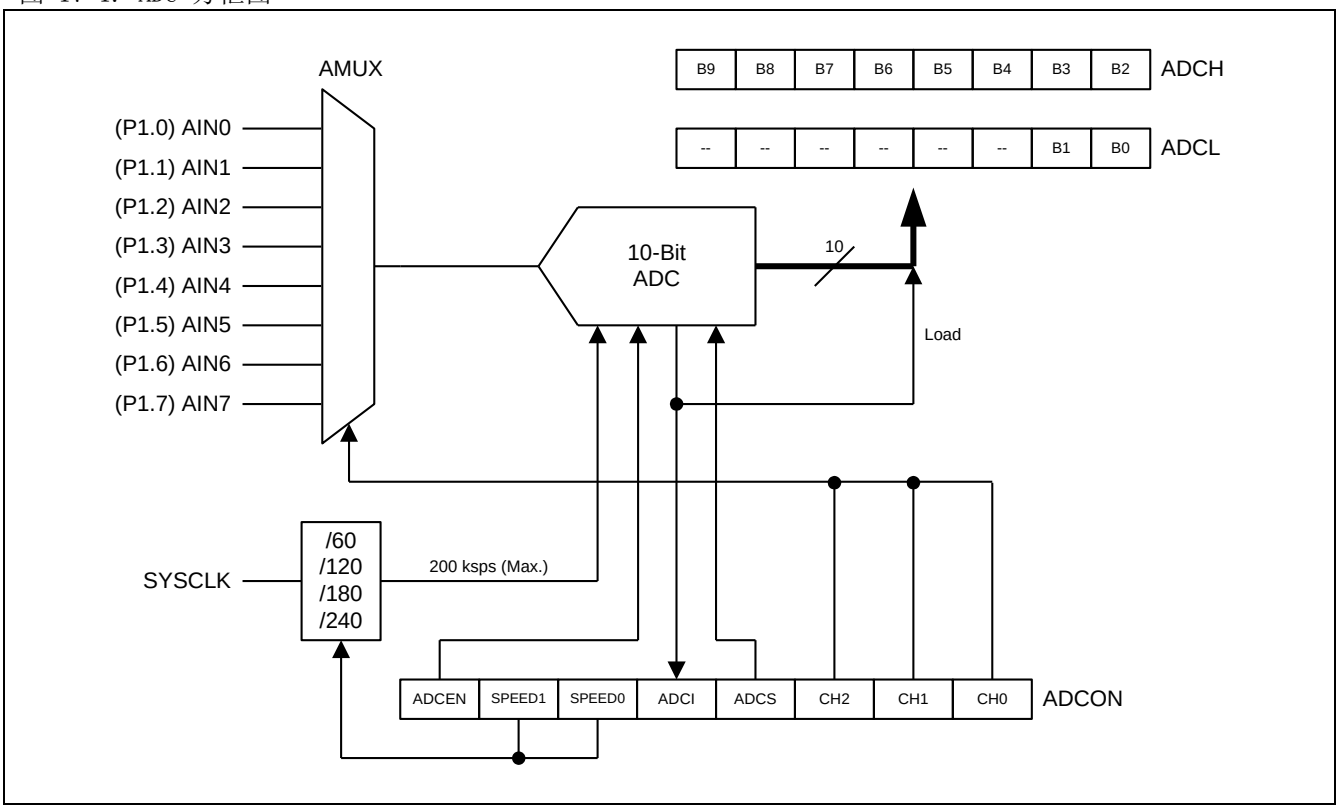
```
void KBI_ISR(void) interrupt 13
{
      // Do KBI key checking
      KBCON &= ~KBIF;                             //清除 KBI 中断标志位 flag (写“0”)
    }
}

void main(void)
{
      P2M1 &= 0xF0;                               //设置 P2.3~P2.0 为输入口模式
      P2M0 |= 0x0F;                               //设置 P2.3~P2.0 为输入口模式


      KBMASK = 0x0F;                              //设置 P2.3~P2.0 的键盘中断 KBI 功能

      KBCON &= ~KBIF;                             //清除 KBI 中断标志位 flag (写“0”)

      AUXIE |= EKBI;                              //使能键盘中断
      EA = 1;                                     //使能全局中断

   While(1);
}
```

16. 10 位模数转换器（ADC）

MA805-64 的 ADC 子系统由一个模拟多路器（AMUX）、一个 200ksps、10 逐次逼近型模数转换器组成。多路器可以通过特殊功能寄存器进行配置（如图 17-1）运行在单一节点模式，可以配置测量端口 1 的任何一个口。仅当 ADC 控制寄存器(ADCON) 的 ADEN 位被置逻辑 1 的时候 ADC 子系统被使能，ADEN 设置为逻辑 0 的话 ADC 子系统低电关闭。

16.1. ADC 结构

图 17-1. ADC 方框图



16.2. ADC 操作

ADC 最大转换速度可以达到 200 ksps，ADC 转换时钟由 ADCON 寄存器的 SPEED1、SPEED0 两位决定的系统时钟分频而来。**（备注：目前建议 ADC 操作时，系统时钟不要超过 20MHz）**

软件写“1” 到 ADCS 启动 ADC。转换完成后（ADCI 变为 1），转换结果从 ADC 结果寄存（ADCH，ADCL）中得到。作为单节点 ADC，转换结果是：

$$\text{ADC Result} = \frac{V_{IN} \times 1024}{\text{VDD Voltage}}$$

16.2.1. ADC 输入通道

模拟多路器(AMUX)选择输入口给 ADC, 允许端口 1(P1)的任何一个口成为被测量的单节点模式。通过 ADCON 寄存的 CHS.2~0 位选择进入 ADC 测量的通道（见图 17-1）。对被选择的引脚测量的是对地（GND）电压。

16.2.2. 开始转换

在使用 ACD 功能之前，用户应：

1) 设置 ADCEN 位启动 ADC 硬件；
2) 设置 SPEED1 和 SPEED0 位设定转换速度；
3) 设置 CHS2、CHS1 和 CHS0 选择输入通道；
4) 设置 P1M0 和 P1M1 寄存器将所选引脚设定成只输入模式；
5) 设置 ADRJ 位配置 ADC 转换结果输出形式。

现在，用户就可以置位 ADCS 来启动 AD 转换了。转换时间取决于 SPEED1 和 SPEED0 位的设置。一旦转换结束，硬件自动清除 ADCS 位，设置中断标志 ADCI，并将转换结果按照 ADRJ 的设置存入 ADCH 和 ADCL。

如上所述，中断标志 ADCI，由硬件设置以表明一次转换完成。因此，有两种方法检测 AD 转换是否完成：
(1) 软件检测 ADCI 中断标志；
(2) 设置 EXIE1 寄存器 EADC 位和 IE 寄存器 EA 位使能 ADC 中断。这样，转换结束就会跳入中断服务进程。
无论(1) 或 (2)， ADCI 标志都必须在下次转换前用软件清零。

16.2.3. ADC 示例代码

```
start:
        ;...
        ;...

        MOV     ADCON,#0E2h      ;ADCEN=1, turn on ADC hardware
                                     ;(SPEED1,SPEED0)=(1,1), Conv. Time= 60 clock cycles
                                     ;select AIN0 (P1.2) as analog input

        ORL     P1M0,#00000100B  ;P1M0,bit2=1 ;configure P1.2 as Input-Only Mode
        ANL     P1M1,#11111011B   ;P1M1,bit2=0 ;

        ANL     AUXR0,#11111011B  ;ADRJ=0: ADCH contains B9~B2; ADCL contains B1,B0

        ;now, suppose the analog input is ready on AIN2 (P1.2)

        ORL     ADCON,#00001000B ;ADCS=1 ➔Start A-to-D conversion

wait_loop:
        MOV     ACC,ADCON
        JNB     ACC.4,wait_loop  ;wait until ADCI=1 ➔conversion completed

        ;now, the 10-bit ADC result is in the ADCH and ADCL.

        ;...
        ;...
```

16.2.4. ADC 转换时间

用户可以根据输入的模拟信号频率选择合适的转换速度。例如，若 SYSCLK =12MHz，转换速度设为 60 个时钟周期，则输入的模拟信号频率不应超过 200KHz,以保证转换精度。(转换时间 = 1/12MHz x60 = 5us，所以转换速率 = 1/5us = 200KHz.)

16.2.5. I/O 口用于 ADC 转换

用作 A/D 转换的模拟输入引脚也可以保持其数字 I/O 输入输出功能。为了获得最佳转换效果，用作 ADC 的引脚应当禁止其数字输出和输入，可以按照引脚配置一节中的描述将引脚设为只输入模式。

16.2.6. 空闲和掉电模式

在待机和掉电模式下，ADC 将无法使用，若 A/D 功能打开，它将消耗一部分的电流。因此，为了降低待机和掉电模式下的功耗，可以在进入掉电和空闲模式前关闭 ADC 硬件 (ADCON=0)。

16.3. ADC 寄存器

ADCON: ADC 控制寄存器

SFR 页 =全部

SFR 地址 = 0xC5 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: 关闭 ADC 模块
1: 开启 ADC 模块

Bit 6~5: SPEED1、SPEED0, ADC 转换速度控制

SPEED[1:0]	转换时钟选择
0 0	SYSCLK/60
0 1	SYSCLK/120
1 0	SYSCLK/180
1 1	SYSCLK/240

Bit 4: ADCI, ADC 中断标志
0: 该标志必须软件清零
1: 一次 A/D 转换完成时该标志置 1，若中断允许则还会产生一个中断

Bit 3: ADCS. ADC 转换启动
0: ADCS 不能被软件清零，
1: 软件置此位启动 A/D 转换。转换完成，ADC 硬件会自动清除 ADCS 并设置 ADCI。ADCS 或 ADCI 为 1 时将不会开始新的 A/D 转换。

Bit 2~0: CHS2 ~ CHS1, 多路器 ADC 输入通道选择位

CHS[2:0]	被选择的 ADC 通道
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)

AUXR0: 辅助寄存器 0

SFR 页 =全部

SFR 地址 = 0x8E 复位值 = 0000-000X

7	6	5	4	3	2	1	0
P60FC1	P60FC0	P60FD	P34FD	MOVXFD	ADRJ	EXTRAM	--
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R

Bit 2: ADRJ, ADC 结果正确放置选择

0: 转换结果高 8 位存入 ADCH[7:0], 低 2 位存入 ADCL[1:0]

1: 转换结果高 2 位存入 ADCH[1:0], 低 8 位存入 ADCL[7:0]

若 ADRJ = 0

ADCH: ADC 结果高字节寄存器

SFR 页 =全部

SFR 地址 = 0xC6 复位值 = xxxx-xxxx

7	6	5	4	3	2	1	0
(B9)	(B8)	(B7)	(B6)	(B5)	(B4)	(B3)	(B2)
R	R	R	R	R	R	R	R

ADCL: ADC 结果低字节寄存器

SFR 页 =全部

SFR 地址 = 0xBE 复位值 = xxxx-xxxx

7	6	5	4	3	2	1	0
--	--	--	--	--	--	(B1)	(B0)
R	R	R	R	R	R	R	R

若 ADRJ = 1

7	6	5	4	3	2	1	0
--	--	--	--	--	--	(B8)	(B9)
R	R	R	R	R	R	R	R

7	6	5	4	3	2	1	0
(B7)	(B6)	(B5)	(B4)	(B3)	(B2)	(B1)	(B0)
R	R	R	R	R	R	R	R

16.4. ADC 示例代码

(1). 功能需求: ADC 示例代码为系统时钟 SYSCLK=24MHz， 模拟输入是 P1.0/P1.1/P1.2 且 SPEED[1:0]=SYSCLK/270 为 88.9KHz 转换率。

Assembly Code Example:

```
CHS0        EQU             01h
CHS1        EQU             02h
ADCS        EQU             08h
ADCI        EQU             10h
SPEED0          EQU             20h
SPEED1          EQU             40h
ADCON           EQU             80h

INITIAL_ADC_PIN:
    ORL     P1M0, #00000111B            ; P1.0, P1.1, P1.2 =  仅输入模式
    ANL     P1M1,#11111000B

    MOV     ADCTL,#ADCON                ; 使能  ADC 模块
    ; delay 5us
    ; call ....

Get_P10:
    MOV     ADCTL, #(ADCON + SPEED1 + SPEED0)
                                        ; 使能 ADC 模块和启动转换
                                                    ; 转换速度  114.3k @ 24MHz,  选择
P1.0 为 ADC 输出引脚
    CALL    delay_5us
    ORL     ADCTL, #ADCS

    MOV     A, ADCTL                    ; 检测是否转换完成?
    JNB     ACC.4,$-3
    ANL     ADCTL,#(0FFh - ADCI - ADCS)     ; clear ADCI & ADCS
    MOV     AIN0_data_V,ADCV            ; 保存 P1.0 ADC  数据
    ; to do ...

Get_P11:
    MOV     ADCTL,#(ADCON + SPEED1 + SPEED0 + CHS0)   ; 选择 P1.1
```

```
        CALL    delay_5us
        ORL     ADCTL, #ADCS

        MOV     A, ADCTL                        ; 检测是否转换完成?
        JNB     ACC.4,$-3
        ANL     ADCTL,# (0FFh - ADCI - ADCS)    ; 清除 ADCI & ADCS
        MOV     AIN1_data_V,ADCV
       ; to do ...

Get_P12:
        MOV     ADCTL,#(ADCON + SPEED1 + SPEED0 + CHS1)  ; 选择 P1.2
        CALL    delay_5us
        ORL     ADCTL, #ADCS

        MOV     ACC,ADCTL                       ; 检测是否转换完成?
        JNB     ACC.4,$-3
        ANL     ADCTL,# (0FFh - ADCI - ADCS)    ; 清除 ADCI & ADCS
        MOV     AIN2_data_V,ADCV

       ; to do ...

        RET
```

C 语言代码范例:

```
#define     CHS0        0x01
#define     CHS1        0x02
#define     ADCS        0x08
#define     ADCI        0x10
#define     SPEED0          0x20
#define     SPEED1          0x40
#define     ADCON           0x80


void main(void)
{
    unsigned char AIN0_data_V, AIN1_data_V, AIN2_data_V;

    P1M0 |= 0x07;                       // P1.0, P1.1, P1.2 =  仅输入模式
```

```
P1M1 &= ~0x07;

ADCTL = ADCON;                          //使能 ADC 模块
// delay 5us
// ...

// select P1.0
ADCTL = (ADCON | SPEED1 | SPEED0);
                                        //使能 ADC 模块和启动转换
                                        // 转换速度为 114.3k @ 24MHz, 选择 P1.0 为 ADC 输入脚
Delay_5us();
ADCTL |= ADCS;

while ((ADCTL & ADCI) == 0x00);                 //等待完成
ADCTL &= ~(ADCI | ADCS);
AIN0_data_V = ADCV;

// to do ...

// select P1.1
ADCTL = (ADCON | SPEED1 | SPEED0 | CHS0);   // 选择 P1.1
Delay_5us();
ADCTL |= ADCS;


while ((ADCTL & ADCI) == 0x00);                 //等待完成
ADCTL &= ~(ADCI | ADCS);
AIN1_data_V = ADCV;

// to do ...

// select P1.2
ADCTL = (ADCON | SPEED1 | SPEED0 | CHS1);   // 选择 P1.2
Delay_5us();
ADCTL |= ADCS;

while ((ADCTL & ADCI) == 0x00);                 //等待完成
ADCTL &= ~(ADCI | ADCS);
```

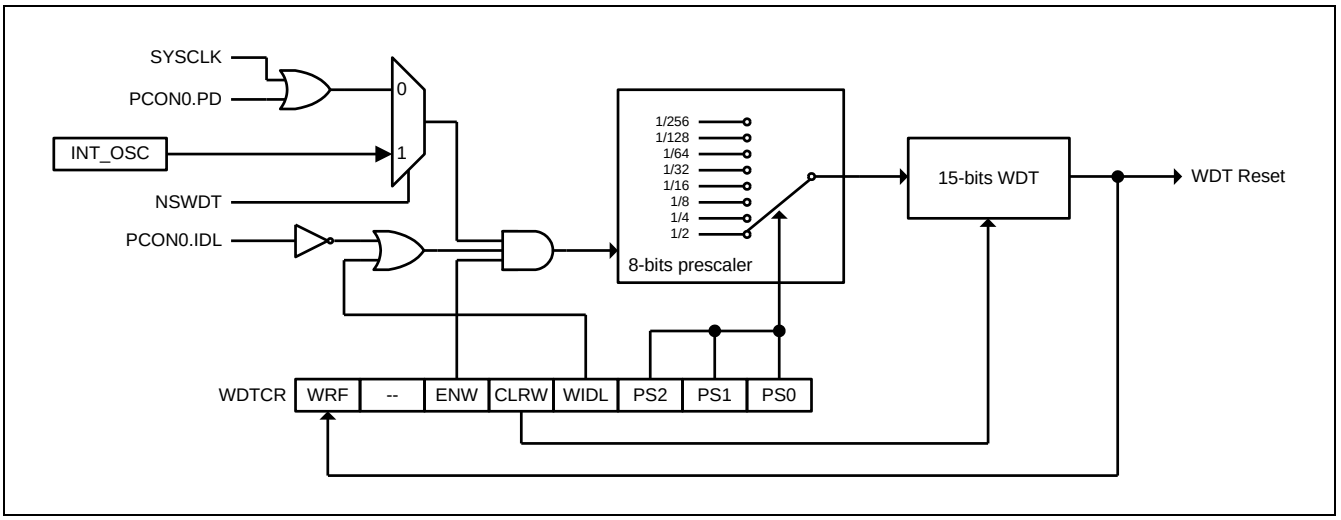
```
        AIN2_data_V = ADCV;

        // to do ...

        while (1);

}
```

17. 看门狗定时器 (WDT)

看门狗定时器 (WDT) 用来使程序从跑飞或死机状态(诸如电源噪音/波动、静电干扰等使 CPU 运行出现混乱或死机)恢复的一个手段。软件跑飞时，WDT 使系统复位来防止系统执行错误的代码。WDT 由一个 15 位独立定时器、一个 8 分频器和一个控制寄存器(WDTCR)组成，图 18-1 显示了 WDT 方框图。

17.1. WDT 结构

图 18-1. WDT 方框图



17.2. WDT 寄存器

WDTCR: WDT 控制寄存器

SFR 页 =全部

SFR 地址 = 0xE1 POR = 0X00-0000

7	6	5	4	3	2	1	0
WRF	--	ENW	CLRW	WIDL	PS2	PS1	PS0
R/W	R	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7: WRF, WDT 复位标志
0: 此位应由软件清零
1: WDT 溢出时，这一位被硬件置位指示是 WDT 复位发生

Bit 6: 保留。当对 WDTCR 进行写操作的时候此位必须写“0”

Bit 5: ENW. 使能 WDT
0: ENW 不能被软件清零
1: 使能 WDT（即开启 WDT）

Bit 4: CLRW. 清零 WDT 计数器
0: 硬件自动清零此位
1: 清 WDT 以重新开始计数

Bit 3: WIDL. WDT 空闲模式控制

0: MCU 在空闲模式时停止 WDT 计数
1: MCU 在空闲模式时保持（继续）WDT 计数

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]	分频值
0 0 0	2
0 0 1	4
0 1 0	8
0 1 1	16
1 0 0	32
1 0 1	64
1 1 0	128
1 1 1	256

17.3. WDT 示例代码

(1)功能需求: 使能 WDT 并且选择 WDT 预分频为 1/32

汇编语言代码范例:

```
PS0             EQU     01h
PS1             EQU     02h
PS2             EQU     04h
WIDL            EQU     08h
CLRW                    EQU             10h
ENW             EQU     20h
WRF             EQU     80h

    ANL     WDTCR,#(0FFh - WRF)                 ; 清除 WRF 标志(写“0”)
    MOV     WDTCR,#(ENW + CLRW + PS2)   ; 使能 WDT 并且选择 WDT 预分频为 1/32
```

C 语言代码范例:

```
#define     PS0     0x01
#define     PS1     0x02
#define     PS2     0x04
#define     WIDL    0x08
#define     CLRW            0x10
#define     ENW     0x20
#define     WRF     0x80

    WDTCR &= ~WRF;                          //清除 WRF 标志(写“0”)
    WDTCR = (ENW | CLRW | PS2);             //使能 WDT 并且选择 WDT 预分频为 1/32
                                            // PS[2:0]   | WDT  预分频器选项
                                            //     0     | 1/2
                                            //     1     | 1/4
                                            //     2     | 1/8
                                            //     3     | 1/16
                                            //     4     | 1/32
                                            //     5     | 1/64
                                            //     6     | 1/128
                                            //     7     | 1/256
```

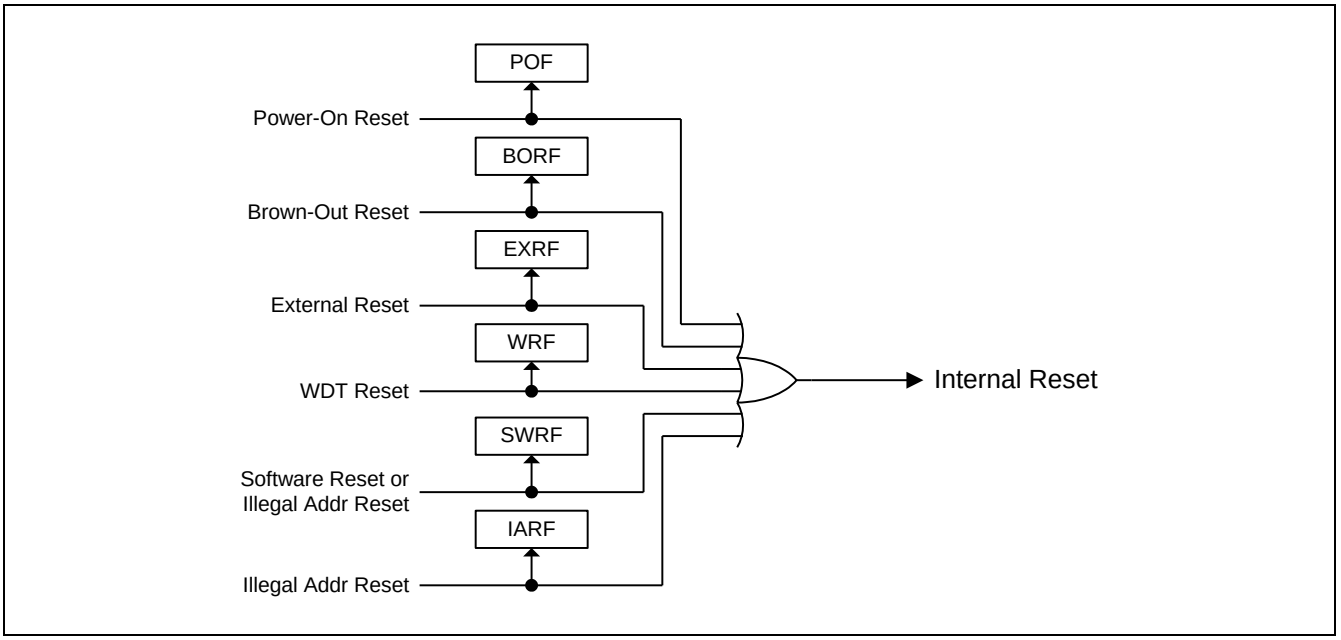
18. 复位

复位期间，所有的 I/O 寄存器设置为它们的初始值，端口引脚微上拉到 VDD，程序从复位地址 0000H 或通过硬件选项设置从 ISP 起始地址开始执行。MA805-64 有六个复位源：上电复位、WDT 复位、软件复位、外部（RST）复位、掉电检测器（brown-out）复位及非法地址复位。

18.1. 复位源

MA805-64 有六个复位源，它们产生一个内部复位去初始化 CPU 和寄存器。图 19-1 所示为复位源图示。

图 19-1. 复位源图示



18.2. 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 VPOR (POR 开始电压)电压之前将保持复位状态。 VDD 电压降到 VPOR 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 VPOR 之下。

PCON0：电源控制寄存器 0

SFR 页 =全部

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, 上电标志

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0V上升到正常电压时硬件置位。POF也能由软件置位

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、软件复位(ISPCR.5)和 WDT 复位）的影响 。它帮助用户检测 CPU 是否是从上电状态开始运行。

18.3. WDT 复位

当 WDT 溢出，将引起一个系统热复位并置位 WRF 标志以指示一个 WDT 复位发生。

WDTCR：WDT 控制寄存器

SFR 页 =全部

SFR 地址 = 0xE1 上电 = 0x00-0000

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

Bit 7: WRF, WDT 复位标志

0: 此位应该由软件清零

1: 当 WDT 溢出时，此位由硬件置位以指示一个 WDT 复位发生

18.4. 软件复位

软件通过对 ISPCR 寄存器的 SWRST 位写“1”触发一个系统热复位。软件复位完成后，硬件置位 PCON1 寄存器的 SWRF 标志指示一个软件复位发生。

ISPCR：ISP 控制寄存器

SFR 页 =全部

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**	**R**	**R**	**R**	**R**

Bit 5: SWRST, 软件复位触发控制位

0: 没有操作

1: 产生软件系统复位。它将被硬件自动清零

PCON1：电源控制寄存器 1

SFR 页 =全部

SFR 地址 = 0x97 POR = 0000-xxx0

7	6	5	4	3	2	1	0
SWRF	EXRF	BORF	IARF	--	--	--	BOD
R/W	**R/W**	**R/W**	**R/W**	**R**	**R**	**R**	**R/W**

Bit 7: SWRF, 软件复位标志

0: 这位必须通过软件清零

1: 若一个软件复位发生时硬件置位此位

18.5. 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，为确保微控制器的正常工作，必须在 RST 引脚上连接可靠的硬件复位电路。外部复位完成后，硬件置位 PCON1 寄存器的 EXRF 标志位以指示一个外部复位发生。

PCON1：电源控制寄存器 1

SFR 页 =全部

SFR 地址 = 0x97 POR = 0000-XXX0

7	6	5	4	3	2	1	0
SWRF	**EXRF**	BORF	IARF	--	--	--	BOD
R/W	R/W	R/W	R/W	R	R	R	R/W

Bit 6: EXRF, 外部复位标志
0: 这位必须通过软件清零
1: 若外部复位发生则被硬件置位

18.6. 掉电检测器（Brown-Out）复位

MA805-64 中，如果 VDD 电压低于 4.2V 则置位 PCON1 寄存器的 BOD 标志位。如果 AUXRA 寄存器中的 BORE 位被使能，BOD 事件将触发一个 CPU 复位并置位 BORF 标志以指示一个掉电检测器（Brown-Out）复位发生。

PCON1：电源控制寄存器 1

SFR 页 =全部

SFR 地址 = 0x97 POR = 0000--xxx0

7	6	5	4	3	2	1	0
SWRF	EXRF	**BORF**	IARF	--	--	--	**BOD**
R/W	R/W	R/W	R/W	R	R	R	R/W

Bit 5: BORF, Brown-Out 复位标志
0: 这位必须通过软件清零
1: 若 brown-out 复位发生则被硬件置位

Bit 0: BOD, Brown-Out 侦察标志
0: 这位必须通过软件清零
1: 如果工作电压匹配 Brown-Out 侦察器的侦察电压则被硬件置位

AUXRA：辅助寄存器 A

SFR 地址 = IFMT POR = 0010--0100

7	6	5	4	3	2	1	0
DBOD	**BORE**	OCDE	ILRCOE	XTALE	IHRCOE	OSCS1	OSCS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 6: BORE, Brown-Out 复位使能
0: 禁止当 BOD 发生时激活复位
1: 使能当 BOD 发生时激活复位

18.7. 非法地址复位

MA805-64 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 PCON1 寄存器中的 IARF 标志以指示一个非法地址复位发生。

PCON1：电源控制寄存器 1

SFR 页 =全部

SFR 地址 = 0x97 上电+ = 0000-xxx0

7	6	5	4	3	2	1	0
SWRF	EXRF	BORF	**IARF**	--	--	--	BOD
R/W	**R/W**	**R/W**	**R/W**	**R**	**R**	**R**	**R/W**

Bit 4: IARF, 非法地址复位标志

0: 这位必须通过软件清零

1: 发生 PC 进入非法地址而复位时被硬件置位

18.8. 复位示例代码

(1) 功能需求: 触发一个软件复位

汇编语言代码范例:

```
SWRST           EQU             20h

    ORL     ISPCR,#SWRST                ; 触发软件复位
```

C 语言代码范例:

```
#define     SWRST               0x20

    ISPCR |= SWRST;                     // 触发软件复位
```

19. 电源管理

MA805-64 支持一个电源监测模块、Brown-Out 侦察器、和两个电源节能模式：空闲模式（IDLE）和掉电模式（Power-Down）。通过对 PCON0 和 PCON1 寄存器来访问这些模式操作芯片的电源事件。

19.1. 节能模式

19.1.1. 空闲模式（IDLE）

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 端口也保持当前逻辑。有两种方式是设备从空闲模式唤醒，首先，将“复位”脚连接到高电平来产生一个内部硬件复位可以唤醒空闲模式中的设备，其次任何处于激活状态的中断源都将会清除 PCON.0 而使设备终止空闲模式，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。空闲模式下定时器 0、定时器 1、定时器 2、SPI、KBI、ADC、UART0、及 UART1 仍然处于工作状态，PCA 定时器和看门狗有条件的设置在 IDLE 时是否唤醒 CPU。在空闲或掉电模式时若 ADC 被禁用则 ADC 的输出通道应该设置为输出“0”或准双向口。

19.1.2. 掉电模式（Power-down）

可以通过软件的方式置 PCON.PD 位，使设备进入掉电模式。在掉电模式下，振荡器被停止，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 或使能的没有停止的看门狗定时器能使系统退出掉电模式。
系统复位或刚退出掉电模式后至少要等 4 微秒后才能进入或再次进入掉电模式。

19.1.3. 中断唤醒

外部中断 nINT0 (P3.2), nINT1 (P3.3), nINT2 (P4.3), nINT3 (P4.2)四个外部中断可以配置为使系统退出掉电模式，但这些中断必须在进入掉电模式前使能并配置成低电平敏感。
唤醒由内部时钟控制，中断口的下降沿系统退出掉电模式，振荡器开始振荡，内部时钟开始计数，但 CPU 要等到内部时钟计数满时才开始执行指令，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系统稳定（设置高电平或上升沿时则保持高电平足够长的时间）。

19.1.4. 复位唤醒

外部复位脚 RST 唤醒有点类似于中断，复位脚 RST 有上升沿电平时系统退出掉电模式，振荡器开始振荡，内部时钟开始计数，但 CPU 要等到内部时钟计数满时才开始执行指令。复位脚 RST 必须保持足够长时间的高电平以保证系统完全复位，复位脚 RST 变成低电平时便开始执行程序。
值得指出的是当 IDLE 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）。

19.1.5. 键盘（KBI）唤醒

MA805-64 的键盘中断, P2.7 ~ P2.0 具有唤醒能力，可以通过 KBI 模块的控制寄存器进行使能。

通过使能KBI唤醒以从掉电模式唤醒有点类似中断唤醒。在使能的KBI参数匹配条件产生及KBI中断被使能(EIE1.5, EKB)时，掉电模式退出，振荡器开始振荡，内部定时器开始计数。但CPU要等到内部时钟计数满时才开始执行指令。计数溢出后，CPU将响应KBI中断并执行其中断服务程序。

19.2. 电源监控模块

MA805-64有一个片上Brown-Out侦察器(BOD)通过比较固定的触发电压去监控芯片工作期间的VCC电压。触发电压为4.2V，当VDD 降到这个触发电压以下时，PCON1中的BOD将被置1并且若EIE1中的EBD被使能则产生一个中断请求，软件进入中断服务程序以响应BOD事件。直到VDD退出BOD电平，硬件将阻碍任何对片上Flash存储器写操作。

当AUXRA 中的BORE被使能，则BOD有能力使MCU复位。当VDD降到BOD电平以下时，Brown-Out 复位被激活。当VDD上升到触发电压时，芯片从0000H处重新开始执行代码。并且硬件置位Brown-Out复位标志PAOCN1.BORF以指示brown-out 复位正好发生。

19.3. 电源控制寄存器

PCON0：电源控制寄存器0

SFR 页 =全部

SFR 地址 = 0x87 上电 = 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 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 =全部

SFR 地址 = 0x97 POR = 0000-xxx0

7	6	5	4	3	2	1	0
SWRF	EXRF	BORF	IARF	--	--	--	BOD
R/W	R/W	R/W	R/W	R	R	R	R/W

Bit 7: SWRF, 软件复位标志
0: 必须由软件清零
1: 软件复位发生时置位此位

Bit 6: EXRF, 外部复位标志
0: 必须由软件清零
1: 外部复位发生时置位此位

Bit 5: BORF, Brown-Out 复位标志
0: 必须由软件清零
1: Brown-Out复位发生时置位此位

Bit 4: IARF,非法地址复位标志
0: 必须由软件清零
1: 非法地址复位发生时置位此位

Bit 3~1: 保留

Bit 0: BOD, Brown-Out 侦察标志
0: 必须由软件清零
1: 若工作电压匹配 Brown-Out 侦察器侦察电压时置位此位

19.4. 电源控制示例代码

(1) 功能需求: 选择系统时钟分频为 OSCin/128 的空闲模式 (默认为 OSCin/1)

汇编语言代码范例:

```
CKS0        EQU     01h
CKS1        EQU     02h
CKS2        EQU     04h

    ORL     PCON2,#( CKS2 + CKS1 + CKS0)    ; 设置 CKS[2:0] = “111” 选择 OSCin/128
```

C 语言代码范例:

```
#define     CKS0            0x01
#define     CKS1            0x02
#define     CKS2            0x04

    PCON2 |= (CKS2 | CKS1 | CKS0);          // 系统时钟分频 /128
                                            // CKS[2:0], 系统时钟分频
                                            //     0    | OSCin/1
                                            //     1    | OSCin/2
                                            //     2    | OSCin/4
                                            //     3    | OSCin/8
                                            //     4    | OSCin/16
                                            //     5    | OSCin/32
                                            //     6    | OSCin/64
                                            //     7    | OSCin/128
```

20. 系统时钟

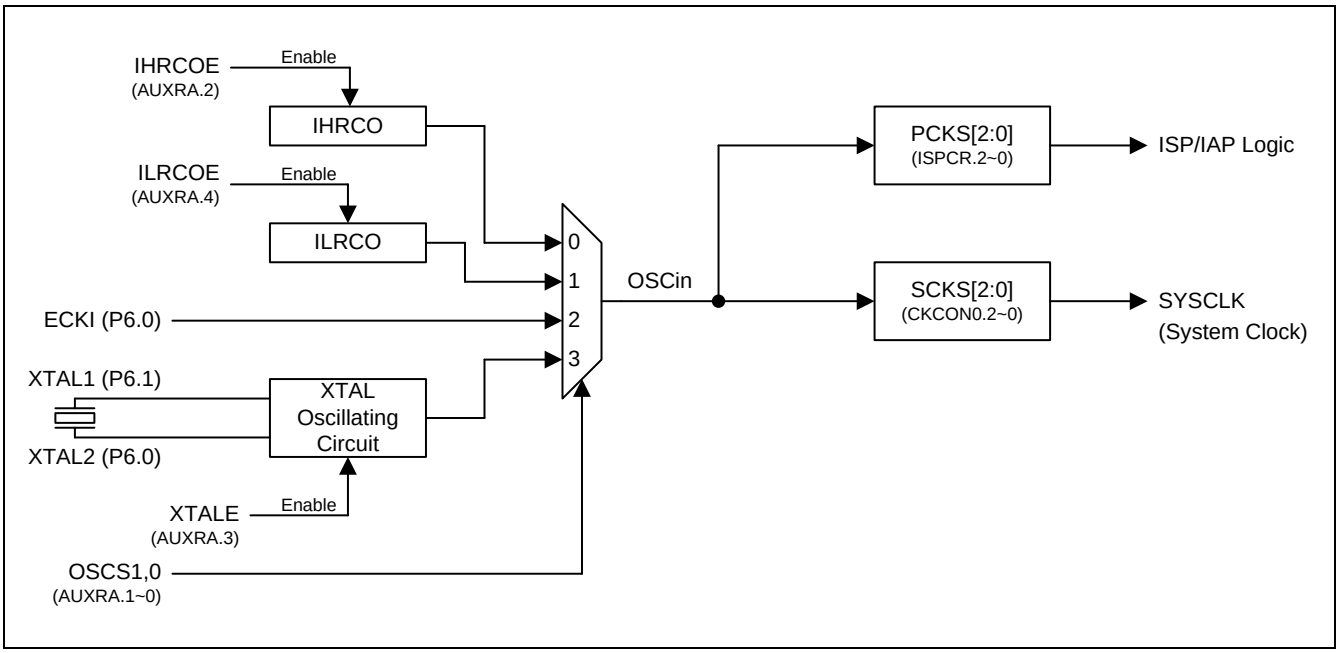
20.1. 时钟结构

MA805-64 有四个时钟源提供给系统时钟： 内部高频 RC 振荡器(IHRCO)、内部低频 RC 振荡器(ILRCO)、外部时钟输入及外部晶体振荡器。系统时钟 SYSCLK,将从这四种时钟源的一种中通过分频器而获得。见图 21-1。用户可以编程分频器控制位 SCKD2~SCKD0 (在 PCON2 寄存中)以得到想要的系统时钟。

上电后 IHRCO 被使能并设置为默认的系统时钟源。 软件能使能其它振荡器电路和通过编程 AUXRA 寄存器对它们进行切换。譬如用户选择外部晶体振荡器作系统时钟，软件必须使能外部晶体振荡器电路并等到它稳定，然后编程 OSCS[1:0]去切换时钟源到外部晶体振荡器。软件切换选择时钟源前必须小心的确认所选时钟源已准备好并且已稳定。否则将导致系统故障或 CPU 挂起。选好时钟后，软件可能要禁止未用的振荡器电路以减少能耗。

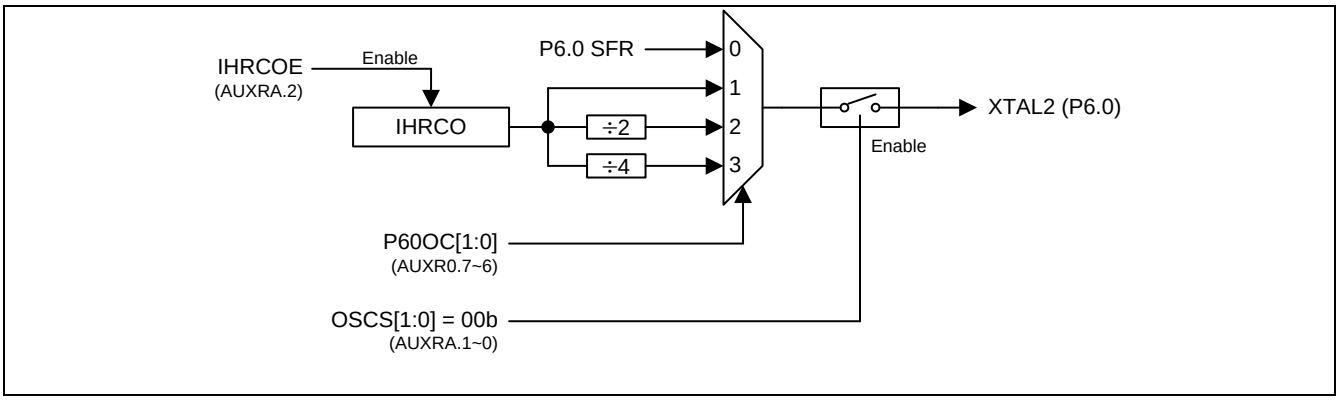
芯片在出厂时默认选择 22.1184MHz 的 IHRCO 作为系统时钟。

图 21-1. 系统时钟方框图



在 IHRCO 模式,XTAL2 和 XTAL1 作普通 I/O(GPIO)功能 P6.0 和 P6.1。这里的 P6.0 能被编程为输出分频过的 IHRCO 时钟，软件可以选择 AUXR1 寄存器中的 P60OC[1:0] 位选择 1、2 或 4 分频。图 21-2 为 IHRCO 输出方框图。

图 21-2. IHRCO 时钟输出方框图



20.2. 时钟控制寄存器

PCON2: 时钟控制寄存器 2

SFR 页 =全部

SFR 地址 = 0xC7 复位值 = xxxx-x000

7	6	5	4	3	2	1	0
OSCDR	-	-	-	-	SCKS2	SCKS1	SCKS0
R/W	R	R	R	R	R/W	R/W	R/W

Bit 7: OSCDR, OSC 驱动力控制寄存器。默认值为 OSCDN (在硬件选项里)，并且软件能够读/写它。
0: 振荡器的驱动能力足够振荡到 25MHz
1: 振荡器的驱动能力被减少。这有利于减少 EMI。关于不需要高频率时钟的应用推荐使用此模式。

Bit 6~3: 保留

Bit 2~0: SCKS2 ~ SCKS0, 可编程系统时钟选择

SCKS[2:0]	系统时钟(Fosc)
0 0 0	OSCin (默认)
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	COSin /128

AUXR0: 辅助寄存器 0

SFR 页 =全部

SFR 地址 = 0x8E 复位值 = 0000-000x

7	6	5	4	3	2	1	0
P60OC1	P60OC0	P60FD	P34FD	MOVXFD	ADRJ	EXTRAM	--
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R

Bit 7~6: P60 输出配置控制位 1 和位 0。这两位仅仅当 IHRCO 被选择作系统时钟源时有效。这种情况， XTAL2 和 XTAL1 改变功能作 P60 和 P61。P60 为 GPIO 或时钟源发生器提供下列选项。当 P60OC[1:0] 索引为非 P60 功能时，XTAL2 将驱动内部高频 RC 振荡器输出为其它设备提供时钟源。

P60OC[1:0]	XTAL2 功能
00	P60 (默认)

01	IHRCO
10	IHRCO/2
11	IHRCO/2

Bit 5: P60FD, P6.0 快速驱动
0: P6.0 默认驱动力输出
1: P6.0 快速驱动力输出使能。若 P6.0 被配置为时钟输出，当 P6.0 输出频率大于 12MHz(5V)或大于 6MHz(3V)时使能此位。

AUXRA: 辅助寄存器 A

SFR 地址 = IFMT 复位值 = 0010--0100

7	6	5	4	3	2	1	0
DBOD	BORE	OCDE	ILRCOE	XTALE	IHRCOE	OSCS1	OSCS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 4: LIRCOE, 内部低频 RC 振荡器使能
0: 禁用内部低频 RC 振荡器
1: 使能内部低频 RC 振荡器。大约是 125 KHz。在使能 ILRCOE 后需要 **50 微秒**才能有稳定的输出。

Bit 3: XTALE, 外部晶体(XTAL)使能
0: 禁止 XTAL 振荡电路。这时，XTAL2 和 XTAL1 作 P6.0 和 P 6.1
1: 使能 XTAL 振荡电路。如过此位被软件置 1 使能 XTALE 后需要 **5 毫秒**才能有稳定的输出。

Bit 2: IHRCOE, 内部高频 RC 振荡器使能
0: 禁用内部高频 RC 振荡器
1: 使能内部高频 RC 振荡器。如过此位被软件置 1 使能 IHRCOE 后需要 **50 微秒**才能有稳定的输出。

Bit 1~0: OSC 输入选择

OSCS[1:0]	OSCin 源	P6.0 功能	P6.1 功能
00	IHRCO (默认)	P6.0 或 IHRCO 输出	P6.1
01	ILRCO	P6.0	P6.1
10	外部时钟输入	时钟输入	P6.1
11	外部晶体振荡	XTAL2	XTAL1

20.3. 例程：内部晶振切换为外部晶振

```
;*******************************************************************************
; 以下例程针对 Megawin MA805-xx 内部晶振切换为外部晶振.
; 将以下例程植入到用户程序的开始
;*******************************************************************************
$INCLUDE (REG_MA805-xx.INC) ;for MA805-xx SFR definition
ISP_StandBy     EQU  00h
AUXRA_Wr   EQU  06h
AUXRA_Rd   EQU  07h
;===============================================================================
     CSEG AT   0000h
     JMP  start
;===============================================================================
code_main  SEGMENT   CODE
     USING 0
start:
     MOV  SP,#stack_space-1
```

```
    CLR   TR0
    ORL   TMOD, #01h
    MOV   TH0, #0DCh            ;使用定时器0来做5ms的定时延迟
    MOV   TL0, #00h
    CLR   TF0

    ORL   ISPCR, #80h           ;使能ISP/IAP 控制
    MOV   IFMT, #AUXRA_Rd       ;设置AUXRA寄存器读取
    MOV   SCMD, #46h
    MOV   SCMD, #0B9h
    MOV   A, IFD
    ORL   A, #08h               ;设置使能外置晶振震荡 XTALE
    MOV   IFD, A
    MOV   IFMT, #AUXRA_Wr       ;设置AUXRA寄存器写入
    MOV   SCMD, #46h
    MOV   SCMD, #0B9h

    SETB  TR0                   ;定时器0 开始定时
    JNB   TF0, $                ;等待定时器0 的5ms定时延迟，让外部晶振震荡稳定
    CLR   TF0
    CLR   TR0                   ;定时器0 停止定时

    MOV   IFMT, #AUXRA_Rd       ;设置AUXRA寄存器读取
    MOV   SCMD, #46h
    MOV   SCMD, #0B9h
    MOV   A, IFD
    ORL   A, #03h               ;设置外部晶振为系统时钟
    MOV   IFD, A
    MOV   IFMT, #AUXRA_Wr       ;设置AUXRA寄存器写入
    MOV   SCMD, #46h
    MOV   SCMD, #0B9h

    MOV   IFMT, #AUXRA_Rd       ;设置AUXRA寄存器读取
    MOV   SCMD, #46h
    MOV   SCMD, #0B9h
    MOV   A, IFD
    ANL   A, #0FBh              ;设置将内置晶振停止
    MOV   IFD, A
    MOV   IFMT, #AUXRA_Wr       ;设置AUXRA寄存器写入
    MOV   SCMD, #46h
    MOV   SCMD, #0B9h
;===============================================================================
;     用户代码开始……..
```

21. 在系统编程序(ISP)

MA805-64 的 ISP 使用户无需将 MCU 从应用系统中取下修改程序(AP-空间)和非易失性数据(IAP-空间)成为可能. 这个功能非常适用于在现场需要更新应用程序的应用 (注意 ISP 功能引导预编程在 ISP-空间中的程序) 。总的来说，用户不需要知道 ISP 如何工作的，因为笙泉已经提供标准的 ISP 工具并且芯片在出厂的时候已经嵌入 ISP 代码。

21.1. 自己的 ISP 代码开发

ISP 代码开发将在以后版本的数据手册里提供。

21.2. ISP (IAP) 控制寄存器

下面的特殊功能寄存跟 ISP 操作有关。用户应用程序能访问所有的这些寄存器。

IFD: ISP/IAP Flash 数据寄存器

SFR 页 =全部

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 为 ISP/IAP 操作的数据寄存器，ISP/IAP 进行读写操作时，IFD 作为数据缓冲区。当用于访问 IAPLB 、AUXRA 或 AUXRB 时，IFD 为 IAPLB、AUXRA 或 AUXRB 的值。

IFADRH: ISP/IAP 地址高

SFR 页 =全部

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 存放 ISP/IAP 操作的目标地址的高位字节。

IFADRL: ISP/IAP 地址低

SFR 页 =全部

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 存放 ISP/IAP 操作的目标地址的低位字节，在进行页擦除时，IFADRL 的值被忽略。

IFMT: ISP/IAP Flash 模式表寄存器

SFR 页 =全部

SFR 地址 = 0xE5 复位值 = xxx0-0000

7	6	5	4	3	2	1	0
--	--	--	MS[4]	MS[3]	MS[2]	MS[1]	MS[0]

R	R	R	R	R/W	R/W	R/W	R/W

Bit 7~4: 保留

Bit 4~0: ISP/IAP 操作模式选择。IFMT 被用来选择实现各种 ISP/IAP 功能的 Flash 操作模式或用来访问保护的特殊功能寄存器 SFRs。

Bit[4:0]					模式
0	0	0	0	0	待命
0	0	0	0	1	AP 存储器读
0	0	0	1	0	AP 存储器写
0	0	0	1	1	AP 存储器页擦除
0	0	1	0	0	IAPLB 写(保护的 SFR)
0	0	1	0	1	IAPLB 读(保护的 SFR)
0	0	1	1	0	AUXRA 写(保护的 SFR)
0	0	1	1	1	AUXRA 读(保护的 SFR)
1	0	0	0	0	AUXRB 写(保护的 SFR)
1	0	0	0	1	AUXRB 读(保护的 SFR)
其它					**为测试而保留，一定不要编程它们。**

SCMD: 顺序命令寄存器

SFR 页 =全部

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

ISP/IAP/IAPLB 的操作都需要用 SCMD 寄存器来触发，当 ISPCR.7 为“1”且 SCMD 顺序写入命令“0x46 0xB9”时，ISP 操作被触发。

ISPCR: ISP 控制寄存器

SFR 页 =全部

SFR 地址 = 0xE7 复位值 = 0000-x000

7	6	5	4	3	2	1	0
ISPEN	SWBS	SWRST	CFAIL	--	PCKS2	PCKS1	PCKS0
R/W	R/W	R/W	R/W	R	R/W	R/W	R/W

Bit 7: ISPEN, ISP/IAP 操作使能位
0: 全局禁用所有的 ISP/IAP 读/写/擦除功能
1: 使能 ISP/IAP 读/写/擦除功能

Bit 6: SWBS, 软件引导选项控制
0: 复位后引导从主存储器开始
1: 复位后引导从 ISP 存储器开始

Bit 5: SWRST, 软件复位触发控制
0: 无操作
1: 产生软件系统复位。此位被硬件自动清零

Bit 4: CFAIL, 指示 ISP/IAP 操作命令是否执行失败

0: 最后的 ISP/IAP 命令已经操作成功
1: 最后的 ISP/IAP 命令已经操作失败。它会导致后续的 Flash 存储器的操作被抑制。

Bit 3: 保留。当对 ISPCR 进行写操作的时候此位必须写“0”

Bit 2~0: PCKS2~0, ISP/IAP 编程时钟源选择

PCKS[2:0]	OSCin 频率 (MHz)
0 0 0	> 24MHz
0 0 1	20 ~ 24
0 1 0	12 ~ 20
0 1 1	6~ 12
1 0 0	3 ~ 6
1 0 1	2 ~ 3
1 1 0	1 ~ 2
1 1 1	< 1

IAPLB: IAP 低边界

SFR 地址 =间接地址 复位值 = 1111-111x

7	6	5	4	3	2	1	0
IAPLB							--
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R

Bit 7~0: IAPLB 决定 IAP 存储器的低边界。由于 Flash 的页面大小为 512 字节，所以 IAPLB 的值必须为偶数。

读 IAPLB，MCU 需要定义 IMFT 为读取 IAPLB 并置位 ISPCR.ISPEN，然后顺序的写 0x46h 、0xB9h 到 SCMD，此时 IFD 中保存的即为 IAPLB 的值。如果写 IAPLB， MCU 先将 IAPLB 的预设置值写入 IFD 中，然后设置 IMFT 为写 IAPLB 并置位 ISPCR.ISPEN，接着顺序的写 0x46h 、0xB9h 到 SCMD，IAPLB 中的内容已经完成更新序列。
具体流程如下：
读 IAPLB 的方法：

```
IFMT = 0x05;
ISPCR = 0x80;
SCMD = 0x46;
SCMD = 0xB9;
        //此时 IFD 中保存的即为 IAPLB 的值
```

设置 IAPLB 的方法：

```
IFD = ??;  //将 IAPLB 的预设值写入 IFD 中
IFMT = 0x04;
ISPCR = 0x80;
SCMD = 0x46;
SCMD = 0xB9;
```

MA805-64 中高于 AP 边界处有 2.5K 字节的 IAP 空间。如果 IAP 大小不够用并且 AP 空间有多余一页未用的 Flash，此时软件可以通过修改 IAPLB 把这些 AP 设置为 IAP 应用存储器。
IAP 区域由 IAPLB 和 ISP 起始地址共同决定：
IAP 低边界 = IAPLB * 256
IAP 高边界 = ISP 起始地址 - 1

例如：在 MA805-64 中，如果 IAPLB=0xF0 及 ISP 起始地址是 FA00H，则 IAP 存储器范围位于 F000H ~ F9FFH。

特别注意：IAP 低边界地址必须不要高于 ISP 起始地址或非设备定义的空间。否则，可能会导致 Flash 存储器中数据内容遭到破坏。

22. 在应用程序编程(IAP)

MA805-64 可编程序存储器（AP-memory）大小被限制为 60K。IAPLB 与 ISP 起始地址之间的 Flash 可以定义为数据闪存 IAP，可供应用程序通过 ISP 操作存取，IAP 可以通过 IAPLB 来改变其大小。

22.1. ISP/IAP 示例代码

(1). 功能需求: ISP/IAP Flash 读的子程序

汇编语言代码范例:

```
IxP_Flash_Read    EQU       01h
ISPEN             EQU       80h

_ixp_read:
ixp_read:

    MOV     ISPCR,#ISPEN                    ; 功能使能
    MOV     IFMT,# IxP_Flash_Read           ; ixp_read=0x01

    MOV     IFADRH,??                       ; 填写 [IFADRH,IFADRL] 字节地址
    MOV     IFADRL,??

    MOV     SCMD,#046h                      ;
    MOV     SCMD,#0B9h                      ;

    MOV     A,IFD                           ; 现在读出来的数据存在于 IFD 中

    MOV     IFMT,#000h                      ; Flash_Standby=0x00
    ANL     ISPCR,#(0FFh – ISPEN)           ; 禁止功能

    RET
```

C 语言代码范例:

```
#define     Flash_Standby           0x00
#define     IxP_Flash_Read          0x01
#define     ISPEN                   0x80


unsigned char ixp_read (void)
{
    unsigned char arg;
    ISPCR = ISPEN;                      //功能使能
    IFMT = IxP_Flash_Read;              // IxP_Read=0x01
```

```
    IFADRH = ??
    IFADRL = ??

    SCMD = 0x46;                                    //
    SCMD = 0xB9;                                    //

    arg = IFD;

    IFMT = Flash_Standby;                           // Flash_Standby=0x00
    ISPCR &= ~ISPEN;

    return arg;
}
```

(2). 功能需求: ISP/IAP Flash 檫除的子程序

汇编语言代码范例:

```
IxP_Flash_ Erase    EQU             03h
ISPEN               EQU             80h

_ixp_erase:
ixp_erase:

    MOV     ISPCR,#ISPEN                        ; 功能使能
    MOV     IFMT,# IxP_Flash_Erase          ; ixp_erase=0x03

    MOV     IFADRH,??                           ; 填写 [IFADRH,IFADRL] 字节地址
    MOV     IFADRL,??

    MOV     SCMD,#046h                  ;
    MOV     SCMD,#0B9h                  ;

    MOV     IFMT,#000h                          ; Flash_Standby=0x00
    ANL     ISPCR,#(0FFh – ISPEN)           ; 禁止功能

    RET
```

C 语言代码范例:

```
#define     Flash_Standby           0x00
#define     IxP_Flash_Erase         0x03
#define     ISPEN                   0x80


void ixp_erase (unsigned char Addr_H, unsigned char Addr_L)
{
    ISPCR = ISPEN;                          //功能使能
    IFMT = IxP_Flash_Erase;                 // IxP_Erase=0x03

    IFADRH = Addr_H;
    IFADRL = Addr_L;

    SCMD = 0x46;                            //
```

```
    SCMD = 0xB9;                                        //


    IFMT = Flash_Standby;                       // Flash_Standby=0x00
    ISPCR &= ~ISPEN;
}
```

(3). 功能需求: ISP/IAP Flash 写的子程序

汇编语言代码范例:

```
IxP_Flash_Program EQU     02h
ISPEN             EQU     80h

_ixp_program:
ixp_program:

    MOV     ISPCR,#ISPEN                    ; 功能使能
    MOV     IFMT,# IxP_Flash_Program        ; ixp_program=0x03

    MOV     IFADRH,??                       ; 填写[IFADRH,IFADRL] 字节地址
    MOV     IFADRL,??
    MOV     IFD, A                          ; 现在,要写的数据存在于 A 累加器中

    MOV     SCMD,#046h                      ;
    MOV     SCMD,#0B9h                      ;

    MOV     IFMT,#000h                      ; Flash_Standby=0x00
    ANL     ISPCR,#(0FFh – ISPEN)           ; 禁止功能

    RET
```

C 语言代码范例:

```
#define     Flash_Standby           0x00
#define     IxP_Flash_Program       0x02
#define     ISPEN                   0x80

void ixp_program(unsigned char Addr_H, unsigned char Addr_L, unsigned char dta)
{
    ISPCR = ISPEN;                          //功能使能
    IFMT = IxP_Flash_Program;               // IxP_Program=0x02

    IFADRH = Addr_H;
    IFADRL = Addr_L;
    IFD = dta;
```

```
    SCMD = 0x46;                                        //
    SCMD = 0xB9;                                        //

    IFMT = Flash_Standby;                       // Flash_Standby=0x00
    ISPCR &= ~ISPEN;


}
```

23. 辅助特殊功能寄存器

AUXR0: 辅助寄存器 0

SFR 页 =全部

SFR 地址 = 0x8E 复位值 = 0000-000x

7	6	5	4	3	2	1	0
P60FC1	P60FC0	P60FD	P34FD	MOVXFD	ADRJ	EXTRAM	--
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R

Bit 7~6: P60 输出配置控制位 1 和位 0。这两位仅仅当 IHRC0 被选择作系统时钟源时有效。这种情况， XTAL2 和 XTAL1 改变功能作 P60 和 P61。P60 为 GPIO 或时钟源发生器提供下列选项。当 P60OC[1:0] 索引为非 P60 功能时，XTAL2 将驱动内部高频 RC 振荡器输出为其它设备提供时钟源。

P60OC[1:0]	XTAL2 功能
00	P60 (默认)
01	IHRC0
10	IHRC0/2
11	IHRC0/2

Bit 5: P60FD, P6.0 快速驱动
0: P6.0 默认驱动力输出
1: P6.0 快速驱动力输出使能。若 P6.0 被配置为时钟输出，当 P6.0 输出频率大于 12MHz(5V)或大于 6MHz(3V)时使能此位。

Bit 4: P34FD, P3.4 快速驱动
0: 3.4 默认驱动力输出
1: P3.4 快速驱动力输出使能。若 P3.4 被配置为 T0CK0，当 P3.4 输出频率大于 12MHz(5V)或大于 6MHz(3V)时使能此位。

Bit 3: MOVXFD, MOVX 输出信号快速驱动使能
0: MOVX 输出信号为默认速度
1: MOVX 输出信号设为快速驱动。如果有外部存储器，MOVX@DPTR 或 MOVX@Ri，MOVX 输出信号需要快速驱动以延长 ALE/RD/WR 脉冲频率超过 12MHz @5V 或 6MHz @3.3V。

Bit 2: ADRJ, ADC 结果正确放置选择
0: 转换结果高 8 位存入 ADCH[7:0], 低 2 位存入 ADCL[1:0]
1: 转换结果高 2 位存入 ADCH[1:0], 低 8 位存入 ADCL[7:0]

Bit 1: EXTRAM, 外部数据 RAM 使能
0: 使能片上扩展数据存储器(XRAM 1024 字节)
1: 禁止片上扩展数据存储器

Bit 0: 保留。当对 AUXR0 进行写的时候必须对这些位写“0”

AUXR1：辅助控制寄存器 1

SFR 页　　=全部

SFR 地址　　= 0xA2　　复位值 = 0000-xxx0

7	6	5	4	3	2	1	0
P4KBI	P4PCA	P5SPI	P4S1	--	--	--	DPS
R/W	R/W	R/W	R/W	R	R	R	R/W

Bit 7: P4KBI, KBI 功能映像到 P4/P5
0: 禁止 KBI 功能映像到 P4/P5。
1: 设置 KBI 功能映像到 P4/P5，作如下定义：
P2.0 上的‘KBI0’功能映像到 P4.0
P2.1 上的‘KBI1’功能映像到 P4.1
P2.2 上的‘KBI2’功能映像到 P4.2
P2.3 上的‘KBI3’功能映像到 P4.3
P2.4 上的‘KBI4’功能映像到 P5.1
P2.5 上的‘KBI5’功能映像到 P5.0
P2.6 上的‘KBI6’功能映像到 P5.2
P2.7 上的‘KBI7’功能映像到 P5.3

Bit 6: P4PCA, PCA 功能映像到 P4/P5
0: 禁止 PCAI 功能映像到 P4/P5。
1: 设置 PCA 功能映像到 P4/P5，作如下定义：
P1.1 上的‘ECI’功能映像到 P4.2
P1.2 上的‘CEX0’功能映像到 P4.0
P1.3 上的‘CEX1’功能映像到 P4.1
P1.4 上的‘CEX2’功能映像到 P5.0
P1.5 上的‘CEX3’功能映像到 P5.1
P1.6 上的‘CEX4’功能映像到 P5.2
P1.7 上的‘CEX5’功能映像到 P5.3

Bit 5: P5SPI, SPI 接口使能/禁止
0: 禁止 SPI 功能映像到 P5
1: 使能 SPI 功能映像到 P5，作如下定义：
P1.4 上的‘/SS’功能映像到 P5.0
P1.5 上的‘MOSI’功能映像到 P5.1
P1.6 上的‘MISO’功能映像到 P5.2
P1.7 上的‘SPICLK’功能映像到 P5.3

Bit 4: P4S1, 串口 1 (UART1) 功能映像到 P4.0/P4.1.
0: 禁止 UART1 功能映像到 P4。
1: 设置 UART1 功能映像到 P4，作如下定义：
P1.2 上的‘RXD1’功能映像到 P4.0
P1.3 上的‘TXD1’功能映像到 P4.1

Bit 3~1: 保留。当对 AUXR1 进行写的时候必须对这些位写“0”

Bit 0: DPS，双 DPTR 选择位
0: 选择 DPTR0.
1: 选择 DPTR1.

AUXR2：辅助控制寄存器 2

SFR 页　　=全部

SFR 地址 = 0xA6 复位值 = 00xx-xx00

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

Bit 7: T0X12，当 C/T=0 时，定时器 0 的时钟源选择。
0: 清零选择 SYSCLK/12 作为时钟源。
1: 置位选择 SYSCLK 作为系统时钟源。

Bit 6: T1X12, 当 C/T=0 时，定时器 1 的时钟源选择。
0: 清零选择 SYSCLK/12 作为时钟源。
1: 置位选择 SYSCLK 作为系统时钟源。

Bit 5~2: 保留

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 时钟从 P3.5 输出

Bit 0: T0CKOE, 定时器 0 时钟输出使能
0: 禁止定时器 1 时钟输出
1: 使能定时器 1 时钟从 P3.4 输出

SFRPI：SFR 页索引寄存器

SFR 页 =全部
SFR 地址 = 0xAC 复位值 = xxxx-0000

7	6	5	4	3	2	1	0
--	--	--	--	PIDX3	PIDX2	PIDX1	PIDX0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7~4: 保留。当对 SFRPI 进行写的时候必须对这些位写“0”

Bit 3~0: SFR 页索引。有用到的页仅有页 “0”, “1” 及 “F”
有四个寄存器仅在页 0：T2CON(C8H), SCON0(98H), SBUF0(99H) 和 SCFG(9AH)
三个寄存在页 1：SCON0(98H), SBUF0(99H) 和 SCFG(9AH)
一个寄存在页 F：P6(C8H).
其它寄存器能被所有页访问。

PIDX[3:0]	被选择的页
0000	页 0
0001	页 1
0010	页 2
0011	页 3
……	……
……	……
……	……
1111	页 F

AUXRA：辅助控制寄存器 A

SFR 地址 = IFMT POR = 0010-0100

7	6	5	4	3	2	1	0

DBOD	BORE	OCDE	ILRCOE	XTALE	IHRCOE	OSCS1	OSCS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7: DBOD, 禁用 BOD
0: 使能 BOD（默认状态是使能的）
1: 禁用 BOD
在程序流程中如果软件重新使能 BOD，必须等待 50 微秒以上的 BOD 电路启动时间。此期间，软件必须禁止 BOD 中断并且 BORE 滤掉假的 BOD 事件。

Bit 6: BORE, Brown-Out 复位使能
0: 禁止当 BOD 发生时激活复位
1: 使能当 BOD 发生时激活复位

Bit 5: OCDE, OCD 使能。初始值在上电复位时从 OR 选项中载入
0: 禁止 P4.4 和 P4.5 作 OCD 接口
1: 使能 P4.4 和 P4.5 作 OCD 接口

Bit 4: LIRCOE, 内部低频 RC 振荡器使能
0: 禁用内部低频 RC 振荡器
1: 使能内部低频 RC 振荡器。大约是 125 KHz。在使能 ILRCOE 后需要 50 微秒才能有稳定的输出。

Bit 3: XTALE, 外部晶体(XTAL)使能
0: 禁止 XTAL 振荡电路。这时，XTAL2 和 XTAL1 作 P6.0 和 P 6.1
1: 使能 XTAL 振荡电路。如过此位被软件置 1 使能 XTALE 后需要 5 毫秒才能有稳定的输出。

Bit 2: IHRCOE, 内部高频 RC 振荡器使能
0: 禁用内部高频 RC 振荡器
1: 使能内部高频 RC 振荡器。如过此位被软件置 1 使能 IHRCOE 后需要 50 微秒才能有稳定的输出。

Bit 1~0: OSC 输入选择

OSCS[1:0]	OSCin 源	P6.0 功能	P6.1 功能
00	IHRCO (默认)	P6.0 或 IHRCO 输出	P6.1
01	ILRCO	P6.0	P6.1
10	外部时钟输入	时钟输入	P6.1
11	外部晶体振荡	XTAL2	XTAL1

AUXRB: 辅助控制寄存器 B

SFR 地址 = IFMT 复位值 = xxx0-00x0

7	6	5	4	3	2	1	0
--	--	--	IAPO	LPM3	LPM2	--	LPM0
R	R	R	R/W	R/W	R/W	R	R/W

Bit 7~5: 保留。当对 AUXRB 进行写的时候必须对这些位写“0”

Bit 4: IAPO, 仅 IAP 功能
0: 当 Flash 区域低于 IAPLB 定义的 AP 边界时，维持 IAP 区域作 IAP 功能及能运行程序（也就是 IAP 区域中能执行程序）。
1: 禁止程序在 IAP 区域执行，IAP 区域只能作 IAP 功能。

Bit 3, 2, 0: LPM3, 2, 0. 低电压模式操作控制位
如果 OSCin 和 SYSCLK 频率低于 10MHz，软件在这些位上写“1”以降低操作电流。否则，软件必须在这些位写“0”

以维持高速运行。在这些位上写入其它的值是不被允许的。

Bit 1: 保留。当对 AUXRB 进行写的时候必须对些位写“0”

24. 极限参数

MA805-64

参数	额定值	单位
环境温度偏差	-40 ~ +85	° C
存储温度	-65 ~ + 150	° C
IO 口和复位脚的对地电压	-0.5 ~ VDD + 0.5	V
VDD 脚的对地电压	-0.5 ~ +6.0	V
芯片总电流	400	mA
IO 口的最大吸收电流	40	mA

注意：实际参数超过上述各项“绝对最大额定值”可能会对设备造成永久性损坏。这些参数是一个设备进行正常功能操作的最大额定值，任何超过上述各项的条件都不被建议，否则可能会影响设备运行的稳定性。

25. 电器特性

25.1. 直流特性

VSS = 0V, TA = 25 ℃, VDD = 5.0V 和为每个机器周期执行 NOP，除非另外说明

符号	参数	测试条件	界限			单位
			最小	典型	最大	
V_{IH1}	Input High voltage (all I/O Ports)		2.0			V
V_{IH2}	Input High voltage (RESET)		3.5			V
V_{IL1}	Input Low voltage (all I/O Ports)				0.8	V
V_{IL2}	Input Low voltage (RESET)				1.6	V
I_{IH}	Input High Leakage current (all I/O Ports)	V_{PIN} = VDD		0	10	uA
I_{IL1}	Logic 0 input current (all quasi-I/O Ports)	V_{PIN} = 0.4V		20	50	uA
I_{IL2}	Logic 0 input current (all Input only or open-drain Ports)	V_{PIN} = 0.4V		0	10	uA
I_{H2L}	Logic 1 to 0 input transition current (all quasi-I/O Ports)	V_{PIN} =1.8V		250	500	uA
I_{OH1}	Output High current (all quasi-I/O Ports)	V_{PIN} =2.4V	150	220		uA
I_{OH2}	Output High current (all push-pull output ports)	V_{PIN} =2.4V	12			mA
I_{OL1}	Output Low current (all I/O Ports)	V_{PIN} =0.4V	12			mA
I_{OP}	Operating current	F_{OSC} = 24MHz		22	30	mA
I_{IDLE}	Idle mode current	F_{OSC} = 20MHz		12	20	mA
I_{PD}	Power down current			1	10	uA
R_{RST}	Internal reset pull-down resistance			100		Kohm

VSS = 0V, TA = 25 ℃, VDD = 3.3V 和为每个机器周期执行 NOP，除非另外说明

符号	参数	测试条件	界限			单位
			最小	典型	最大	
V_{IH1}	Input High voltage (all I/O Ports)		2.0			V
V_{IH2}	Input High voltage (RESET)		2.8			V
V_{IL1}	Input Low voltage (all I/O Ports)				0.8	V
V_{IL2}	Input Low voltage (RESET)				1.5	V
I_{IH}	Input High Leakage current (all I/O Ports)	V_{PIN} = VDD		0	10	uA
I_{IL1}	Logic 0 input current (all quasi-I/O Ports)	V_{PIN} = 0.4V		7	30	uA
I_{IL2}	Logic 0 input current (all Input only or open-drain Ports)	V_{PIN} = 0.4V		0	10	uA
I_{H2L}	Logic 1 to 0 input transition current (all quasi-I/O Ports)	V_{PIN} =1.8V		100	250	uA
I_{OH1}	Output High current (all quasi-I/O Ports)	V_{PIN} =2.4V	40	70		uA
I_{OH2}	Output High current (all push-pull output ports)	V_{PIN} =2.4V	4			mA

I_{OL1}	Output Low current (all I/O Ports)	V_{PIN} =0.4V	8			mA
I_{OP}	Operating current	F_{OSC} = 24MHz		20	25	mA
I_{IDLE}	Idle mode current	F_{OSC} = 20MHz		9	15	mA
I_{PD}	Power down current			1	5	uA
R_{RST}	Internal reset pull-down resistance			200		Kohm

25.2. 交流特性

26. 指令集

助记符	描述	字节	执行周期
数据传送			
MOV A,Rn	Move register to Acc	1	1
MOV A,direct	Move direct byte o Acc	2	2
MOV A,@Ri	Move indirect RAM to Acc	1	2
MOV A,#data	Move immediate data to Acc	2	2
MOV Rn,A	Move Acc to register	1	2
MOV Rn,direct	Move direct byte to register	2	4
MOV Rn,#data	Move immediate data to register	2	2
MOV direct,A	Move Acc to direct byte	2	3
MOV direct,Rn	Move register to direct byte	2	3
MOV direct,direct	Move direct byte to direct byte	3	4
MOV direct,@Ri	Move indirect RAM to direct byte	2	4
MOV direct,#data	Move immediate data to direct byte	3	3
MOV @Ri,A	Move Acc to indirect RAM	1	3
MOV @Ri,direct	Move direct byte to indirect RAM	2	3
MOV @Ri,#data	Move immediate data to indirect RAM	2	3
MOV DPTR,#data16	Load DPTR with a 16-bit constant	3	3
MOVC A,@A+DPTR	Move code byte relative to DPTR to Acc	1	4
MOVC A,@A+PC	Move code byte relative to PC to Acc	1	4
MOVX A,@Ri	Move on-chip auxiliary RAM(8-bit address) to Acc	1	3
MOVX A,@DPTR	Move on-chip auxiliary RAM(16-bit address) to Acc	1	3
MOVX @Ri,A	Move Acc to on-chip auxiliary RAM(8-bit address)	1	3
MOVX @DPTR,A	Move Acc to on-chip auxiliary RAM(16-bit address)	1	3
MOVX A,@Ri	Move external RAM(8-bit address) to Acc	1	3 ~ 20*Note1
MOVX A,@DPTR	Move external RAM(16-bit address) to Acc	1	3 ~ 20*Note1
MOVX @Ri,A	Move Acc to external RAM(8-bit address)	1	3 ~ 20*Note1
MOVX @DPTR,A	Move Acc to external RAM(16-bit address)	1	3 ~ 20*Note1
PUSH direct	Push direct byte onto Stack	2	4
POP direct	Pop direct byte from Stack	2	3
XCH A,Rn	Exchange register with Acc	1	3
XCH A,direct	Exchange direct byte with Acc	2	4
XCH A,@Ri	Exchange indirect RAM with Acc	1	4
XCHD A,@Ri	Exchange low-order digit indirect RAM with Acc	1	4
算术运算			
ADD A,Rn	Add register to Acc	1	2
ADD A,direct	Add direct byte to Acc	2	3
ADD A,@Ri	Add indirect RAM to Acc	1	3
ADD A,#data	Add immediate data to Acc	2	2
ADDC A,Rn	Add register to Acc with Carry	1	2
ADDC A,direct	Add direct byte to Acc with Carry	2	3
ADDC A,@Ri	Add indirect RAM to Acc with Carry	1	3
ADDC A,#data	Add immediate data to Acc with Carry	2	2

SUBB A,Rn	Subtract register from Acc with borrow	1	2
SUBB A,direct	Subtract direct byte from Acc with borrow	2	3
SUBB A,@Ri	Subtract indirect RAM from Acc with borrow	1	3
SUBB A,#data	Subtract immediate data from Acc with borrow	2	2
INC A	Increment Acc	1	2
INC Rn	Increment register	1	3
INC direct	Increment direct byte	2	4
INC @Ri	Increment indirect RAM	1	4
DEC A	Decrement Acc	1	2
DEC Rn	Decrement register	1	3
DEC direct	Decrement direct byte	2	4
DEC @Ri	Decrement indirect RAM	1	4
INC DPTR	Increment DPTR	1	1
MUL AB	Multiply A and B	1	4
DIV AB	Divide A by B	1	5
DA A	Decimal Adjust Acc	1	4
逻辑运算			
ANL A,Rn	AND register to Acc	1	2
ANL A,direct	AND direct byte to Acc	2	3
ANL A,@Ri	AND indirect RAM to Acc	1	3
ANL A,#data	AND immediate data to Acc	2	2
ANL direct,A	AND Acc to direct byte	2	4
ANL direct,#data	AND immediate data to direct byte	3	4
ORL A,Rn	OR register to Acc	1	2
ORL A,direct	OR direct byte to Acc	2	3
ORL A,@Ri	OR indirect RAM to Acc	1	3
ORL A,#data	OR immediate data to Acc	2	2
ORL direct,A	OR Acc to direct byte	2	4
ORL direct,#data	OR immediate data to direct byte	3	4
XRL A,Rn	Exclusive-OR register to Acc	1	2
XRL A,direct	Exclusive-OR direct byte to Acc	2	3
XRL A,@Ri	Exclusive-OR indirect RAM to Acc	1	3
XRL A,#data	Exclusive-OR immediate data to Acc	2	2
XRL direct,A	Exclusive-OR Acc to direct byte	2	4
XRL direct,#data	Exclusive-OR immediate data to direct byte	3	4
CLR A	Clear Acc	1	1
CPL A	Complement Acc	1	2
RL A	Rotate Acc Left	1	1
RLC A	Rotate Acc Left through the Carry	1	1
RR A	Rotate Acc Right	1	1
RRC A	Rotate Acc Right through the Carry	1	1
SWAP A	Swap nibbles within the Acc	1	1
布尔变量操作			
CLR C	Clear Carry	1	1
CLR bit	Clear direct bit	2	4
SETB C	Set Carry	1	1
SETB bit	Set direct bit	2	4
CPL C	Complement Carry	1	1
CPL bit	Complement direct bit	2	4

ANL C,bit	AND direct bit to Carry	2	3
ANL C,/bit	AND complement of direct bit to Carry	2	3
ORL C,bit	OR direct bit to Carry	2	3
ORL C,/bit	OR complement of direct bit to Carry	2	3
MOV C,bit	Move direct bit to Carry	2	3
MOV bit,C	Move Carry to direct bit	2	4
JC rel	Jump if Carry is set	2	3
JNC rel	Jump if Carry not set	2	3
JB bit,rel	Jump if direct bit is set	3	4
JNB bit,rel	Jump if direct bit not set	3	4
JBC bit,rel	Jump if direct bit is set and then clear bit	3	5
程序分支			
ACall addr11	Absolute subroutine call	2	6
LCall addr16	Long subroutine call	3	6
RET	Return from subroutine	1	4
RETI	Return from interrupt subroutine	1	4
AJMP addr11	Absolute jump	2	3
LJMP addr16	Long jump	3	4
SJMP rel	Short jump	2	3
JMP @A+DPTR	Jump indirect relative to DPTR	1	3
JZ rel	Jump if Acc is zero	2	3
JNZ rel	Jump if Acc not zero	2	3
CJNE A,direct,rel	Compare direct byte to Acc and jump if not equal	3	5
CJNE A,#data,rel	Compare immediate data to Acc and jump if not equal	3	4
CJNE Rn,#data,rel	Compare immediate data to register and jump if not equal	3	4
CJNE @Ri,#data,rel	Compare immediate data to indirect RAM and jump if not equal	3	5
DJNZ Rn,rel	Decrement register and jump if not equal	2	4
DJNZ direct,rel	Decrement direct byte and jump if not equal	3	5
NOP	No Operation	1	1

Note 1: 访问外部辅助 RAM 的周期时间是：

EMAI[1:0] = 00: 5 + 2 x ALE_Stretch + RW_Stretch + 2 x RWSH; (5~20)

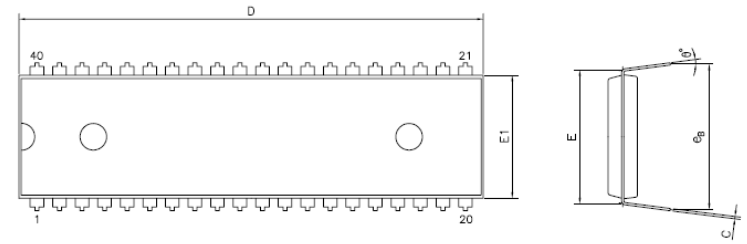
EMAI[1:0] = 01: 3 + RW_Stretch + 2 x RWSH; (3~12)

EMAI[1:0] = 10: 3 + RW_Stretch + 2 x RWSH; (3~12)

EMAI[1:0] = 11: 未定义

27.封装尺寸

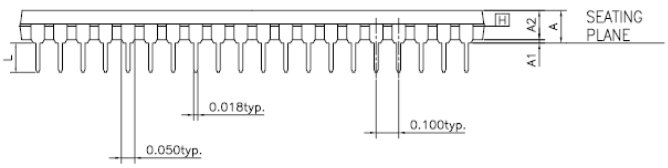
PDIP-40 （PDIP-40 快速验证功能用，不推荐批量生产）



	SYMBOLS	MIN.	NOR.	MAX.
⚠	A	–	–	0.190
	A1	0.015	–	–
	A2	0.150	0.155	0.160
⚠	C	0.008	–	0.015
	D	2.055	2.060	2.070
	E		0.600 BSC	
	E1	0.540	0.545	0.550
⚠	L	0.120	0.130	0.140
	e_B	0.630	0.650	0.670
	θ	0	7	15

UNIT : INCH

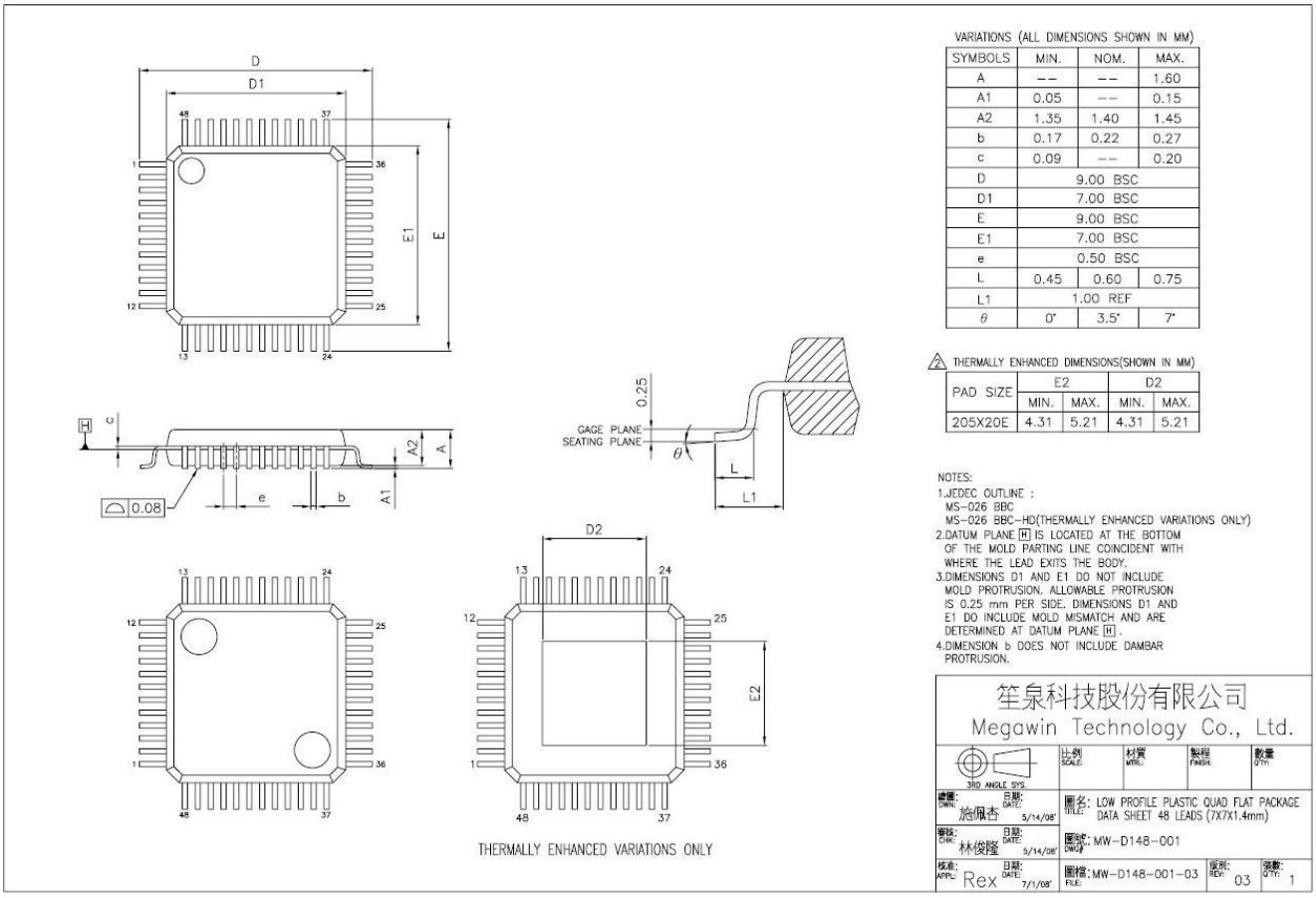
NOTE:
1.JEDEC OUTLINE : MS-011 AC



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 Q'TY:
繪圖: DWN: 施佩杏 日期: DATE: 9/30/05'	圖名: TITLE: DUAL INLINE PLASTIC DATA SHEET P-DIP 40 LEADS (600mil)		
審核: CHK: 蔡大猷 日期: DATE: 10/1/05'	圖號: DWG#: MW-E140-001		
核准: APPL: Rex 日期: DATE: 7/1/07'	圖檔: FILE: MW-E140-001-03	版別: DATE: 03	張數: DATE: 1

LQFP-48



VARIATIONS (ALL DIMENSIONS SHOWN IN MM)

SYMBOLS	MIN.	NOM.	MAX.
A	--	--	1.60
A1	0.05	--	0.15
A2	1.35	1.40	1.45
b	0.17	0.22	0.27
c	0.09	--	0.20
D	9.00 BSC		
D1	7.00 BSC		
E	9.00 BSC		
E1	7.00 BSC		
e	0.50 BSC		
L	0.45	0.60	0.75
L1	1.00 REF		
θ	0°	3.5°	7°

2 THERMALLY ENHANCED DIMENSIONS(SHOWN IN MM)

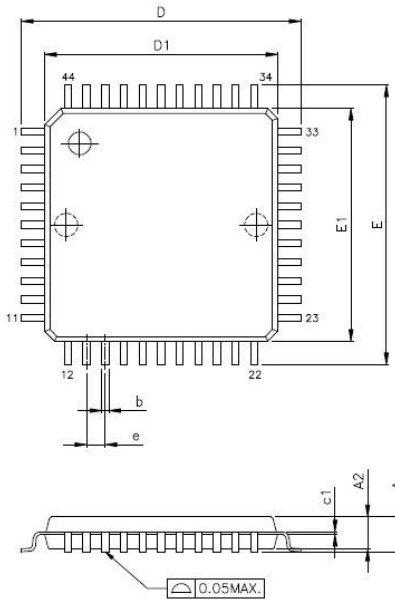
PAD SIZE	E2		D2	
	MIN.	MAX.	MIN.	MAX.
205X20E	4.31	5.21	4.31	5.21

NOTES:
1.JEDEC OUTLINE :
MS-026 BBC
MS-026 BBC-HD(THERMALLY ENHANCED VARIATIONS ONLY)
2.DATUM PLANE [H] IS LOCATED AT THE BOTTOM OF THE MOLD PARTING LINE COINCIDENT WITH WHERE THE LEAD EXITS THE BODY.
3.DIMENSIONS D1 AND E1 DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE PROTRUSION IS 0.25 mm PER SIDE. DIMENSIONS D1 AND E1 DO INCLUDE MOLD MISMATCH AND ARE DETERMINED AT DATUM PLANE [H].
4.DIMENSION b DOES NOT INCLUDE DAMBAR PROTRUSION.

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: 施佩杰	日期: DATE: 5/14/08'	圖名: TITLE: LOW PROFILE PLASTIC QUAD FLAT PACKAGE DATA SHEET 48 LEADS (7X7X1.4mm)		
審核: CHK: 林俊隆	日期: DATE: 5/14/08'	圖號: DWG#: MW-D148-001		
核准: APPL: Rex	日期: DATE: 7/1/08'	圖檔: FILE: MW-D148-001-03	版別: REV: 03	張數: QTY: 1

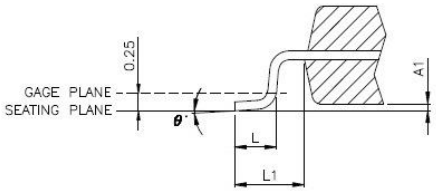
LQFP-44



VARIATIONS (ALL DIMENSIONS SHOWN IN MM)

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

NOTES:
1. JEDEC OUTLINE:MS-026 BCB
2. DIMENSIONS D1 AND E1 DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE PROTRUSION IS 0.25mm PER SIDE. D1 AND E1 ARE MAXIMUM PLASTIC BODY SIZE DIMENSIONS IMCLUDING MOLD MISMATCH.
3. DIMENSION b DOES NOT INCLUDE DAMBAR PROTRUSION.ALLOWABLE DAMBAR PROTRUSION SHALL NOT CAUSE THE LEAD WIDTH TO EXCEED THE MAXIMUM b DIMENSION BY MORE THAN 0.08mm.



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: 陳曉玲 日期: 2/20/08	圖名: LOW PROFILE PLASTIC QUAD FLAT PACKAGE DATA SHEET 44 LEADS			
審核: Erik 日期: 2/20/08	圖號: MW-AD44-001			
核准: Rex 日期: 11/10/11	檔案: MW-AD44-001-03	版別: 03	張數: 1	

28. 版本历史

版本	描述	日期
v0.10	1. 首版	2010/06/29
	2. 编辑	
v0.11	第 114 页 增加例程说明用户如何切换到外部晶振 第 98 页，增加 ADC 操作限制	2010/09/13
v0.12	增加 LQFP-48 封装以及相关管脚定义	2011/01/05
v0.13	修改 PCON2 錯誤描述 修改 ISPCR 錯誤描述	2011/06/20
v0.14	修改錯誤描述	2011/06/23
v0.15	新增 LQFP-44 封装描述,移除 PQFP-44 封装	2012/06/13
v0.16	修改 SFR IE 錯誤描述	2013/07/09
A1.0	编排修改，增加例程	2014/02/25
A1.01	修改系统时钟方框图及 IHRCO 时钟输出方框图	2015/09/25

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。