MA82G5B32/ MA82G5B16/ MA82G5B08 说明书

版本: 1.03

特性

- 高速 1-T 结构 80C51 内核
- **MA82G5B32/MA82G5B16/MA82G5B08** 32K/16K/8K 字节 flash 程序存储器空间
  - ISP 空间可以选择为 1KB/1.5KB~4KB
  - 灵活的 IAP 大小空间由软件设置
  - 密码保护程序区访问
  - Flash 写/擦 次数：10,000 次
  - Flash 数据保留时间：100 年在 25℃环境下
  - **MA82G5B32 出厂默认空间设置**
    - AP Flash 程序空间默认设置(29.5KB, 0000h~75FFh)
    - IAP Flash 数据空间默认设置(1KB, 7600h~79FFh)
    - ISP Flash 引导码空间默认设置(1.5KB, 7A00h~7FFFh), ISP 引导码
  - **MA82G5B16 出厂默认空间设置**
    - AP Flash 程序空间默认设置(13.5KB, 0000h~35FFh)
    - IAP Flash 数据空间默认设置(1KB, 3600h~39FFh)
    - ISP Flash 引导码空间默认设置(1.5KB, 3A00h~3FFFh), ISP 引导码
  - **MA82G5B08 出厂默认空间设置**
    - AP Flash 程序空间默认设置(5.5KB, 0000h~15FFh)
    - IAP Flash 数据空间默认设置(1KB, 1600h~19FFh)
    - ISP Flash 引导码空间默认设置(1.5KB, 1A00h~1FFFh), ISP 引导码
- 数据存储器 XRAM
  - 内部 256 字节数据存储器
  - **MA82G5B32** 1792 字节扩展数据存储器(XRAM)
  - **MA82G5B16** 768 字节扩展数据存储器(XRAM)
  - **MA82G5B08** 256 字节扩展数据存储器(XRAM)
- 两个数据指针
- 中断控制
  - 16 中断源，4 个优先级
  - 四个带抗干扰滤波器的外部中断， nINT0，nINT1， nINT2 和 nINT3
  - 所有的外部中断支持高/低电平或上升/下降沿触发
- 三个 16-位 定时/计数， 定时器 0，定时器 1 和定时器 2
  - T0CKO 在 P34, T1CKO 在 P35 和 T2CKO 在 P10
  - T0/T1/T2 可以选择 X12 模式
  - S1BRG 通过定时器 1 级联到一个 16/24 位的定时/计数器
- 有 8 个比较/俘获单元的可编程 16 位定时/计数阵列(PCA)
  - 可编程的 16 位基准计数器
  - 来自片内时钟倍频器(CKM)的时钟源高达 100MHz
  - 俘获模式，16 位软件时钟模式和高速输出模式
  - 8/10/12/16-位具有相移功能的脉宽调制(PWM)模式，**高达 8 通道的脉宽调制(PWM)**
  - 脉宽调制(PWM)模块具有空载时间控制和中心对齐选项
- 键盘中断
- **10-位 ADC**
  - 可编程的转换率高达 200 千次采样每秒(ksps)
  - 最多可到 8 通道单一输入
- 增强型 UART (S0)
  - 帧误差侦测

- 自动地址识别
- 速度增强机制(X2/X4 模式)
- SPI 主机工作在模式 4

- 第二个 UART (S1)
  - 专用的波特率产生器分享到 S0 或设置为 8 位定时器
  - SPI 主机工作在模式 4
- 一个主/从 SPI 串口接口(SPI)
  - 最大 SPI 时钟频率高达 12MHz
  - 高达 3 个 SPI 主机包括 S0 和 S1 工作在模式 4
- 三个两线串口接口：TWI0, TWI1 和 STWI(SID)
  - 2 个主机/从机硬件引擎：TWI0 和 TWI1
  - 3 个装置地址识别在 TWI0/TWI1 从机模式下
  - 两线串口接口开始/停止(Start/Stop)侦测(SID)支持软件 TWI 从机接口
- 片上调试接口 (OCD)
- 可编程看门狗定时器，时钟来源于 ILRCO
  - 通过 CPU 或上电复位一次性使能
  - 看门狗(WDT)溢出会中断或复位 CPU
  - 掉电模式(watch 模式)支持看门狗(WDT)功能
- 实时时钟模块
  - 0.5S ~ 64S 可编程中断周期
  - 21-位长系统定时器
- 报警器功能
- 在 32-脚封装中最大 29 个通用输入输出口(GPIO)
  - P3 可以设置成准双向口模式，推挽输出模式，开漏集输出模式和仅输入模式
  - P1, P2, P4 and P6 可以设置为推挽输出模式，开漏集输出模式
  - P6.0, P6.1 和 P4.7 公用 XTAL2, XTAL1 和 RST
- 多种功耗控制模式：掉电模式， 空闲模式，慢频模式，副频模式，RTC 模式，watch 模式和 monitor 模式
  - 所有的中断能唤醒空闲(IDLE)模式
  - 11 中源能唤醒掉电模式
  - 慢频模式和副频模式支持低速 MCU 运转
  - RTC 模式在掉电模式下支持实时时钟(RTC)复位 CPU
  - Watch 模式在掉电模式下支持看门狗(WDT)复位 CPU
  - Monitor 模式在掉电模式下支持 BOD1 复位 CPU
- 两个低电压检测
  - BOD0：检测 1.7V
  - BOD1：选择检测电压为 4.2V/3.7V/2.4V/2.0V
  - 中断 CPU 或复位 CPU
  - 在掉电模式下唤醒 CPU
- 工作电压范围：1.8V – 5.5V 芯片子系列
  - flash 写操作(ISP/IAP/ICP)的最低电压为 1.7V
- 工作频率范围：25MHz(最高)
  - 外部晶振模式，0 – 12MHz 在 1.8V – 5.5V 和 0 – 25MHz 在 2.7V – 5.5V
  - CPU 工作频率可达 12MHz 在 1.8V – 5.5V 和 25MHz 在 2.2V – 5.5V
- 时针源
  - 内部 **12MHz/11.059MHz** 振荡器 (IHRCO) ： 工厂条件下测误差 ±1%，典型值
  - 外部晶振模式，支持 32.768KHz 振荡和时钟丢失检测(MCD)
  - 内部低功耗 32KHz RC 振荡器(ILRCO)
  - 外部时针输入(ECKI) 在 P6.0/XTAL2

  - 内部振荡输出在 P6.0/XTAL2
  - 内建时钟倍频器(CKM)用来提供高速时钟源
- 工作温度
  - 工业级(-40°C 到+85°C)*
- 16-字节唯一的 ID 代码
- 封装类型
  - LQFP32 (7mm x 7mm): MA82G5B32AD32
  - LQFP32 (7mm x 7mm): MA82G5B16AD32
  - SOP28 : MA82G5B16AS28
  - SOP20 : MA82G5B16AS20, MA82G5B08AS20
  - SOP16 : MA82G5B08AS16

*:抽样检测

目录

1. 概述

MA82G5BXX 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 时钟信号 (比标准的8051快 6~7 倍), 与标准8051指令集兼容。因此在与标准8051有同样的处理能力的情况下，**MA82G5BXX**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。

MA82G5BXX有32K/16K/8K字节的内置Flash存储器用于保存代码。Flash存储器可以通过串行模式编程 (ICP, 在电路编程) 或者 ISP模式进行编程的能力。同时，也提供在应用编程(IAP)的能力。ISP和ICP让使用者无需从产品中取下微控制器就可以下载新的代码；IAP意味着应用程序正在运行时，微控制器能够在Flash中写入非易失数据。这些功能都由内建的电荷泵提供编程用的高压。

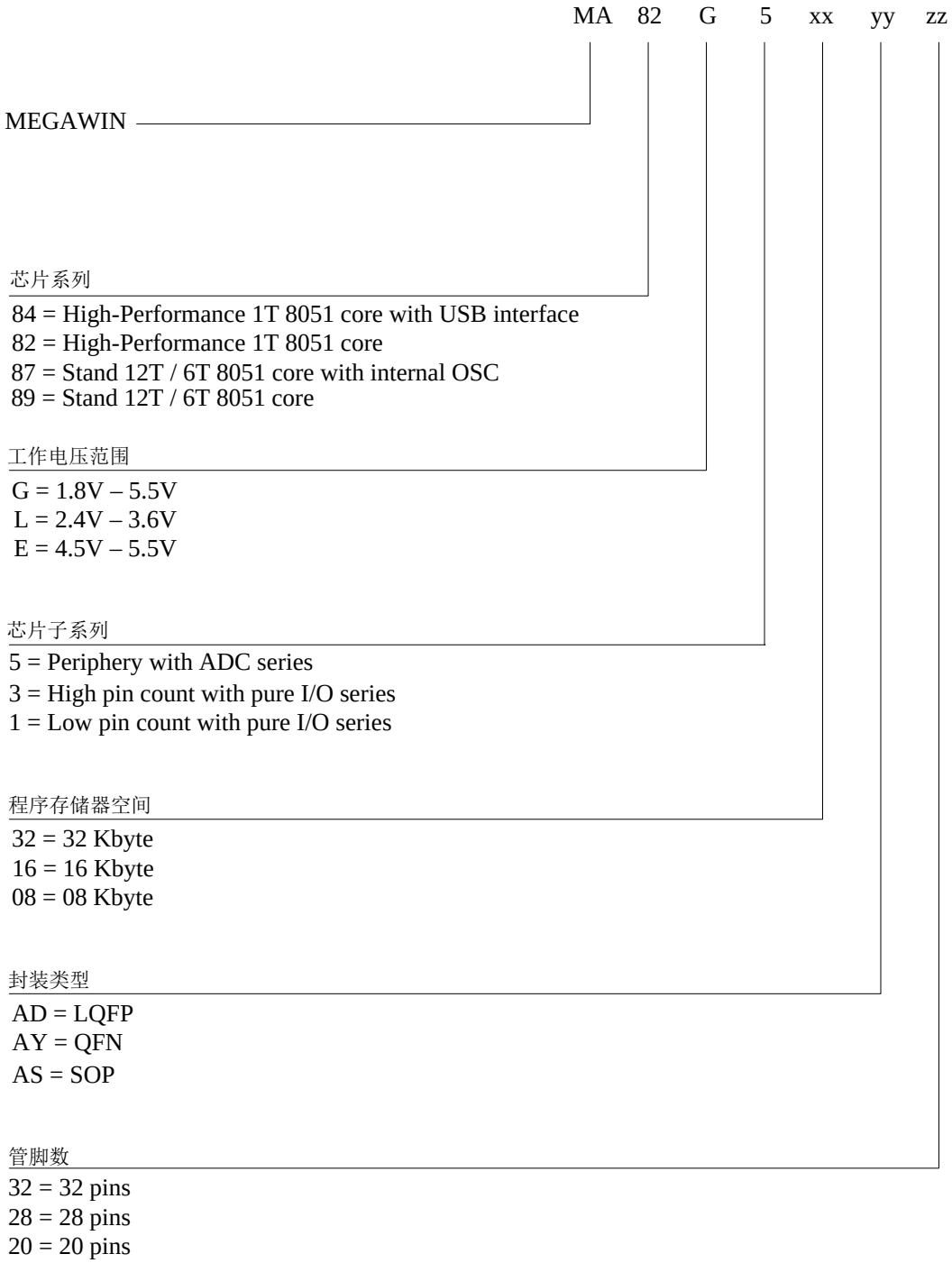
MA82G5BXX除了80C52 MCU的标准功能（例如 256 字节的随机存储器，三个8位I/O口，二个带有高/低触发选项的外部中断，一个多源4级中断控制，一个串口(UART0)和三个定时/计数器)外，**MA82G5BXX**有三个额外的 I/O 端口引脚(P4.5, P4.4, P4.1, P4.0, P6.1和P6.0)，**MA82G5BXX**有1792/768/256字节外部数据存储器（XRAM），两个额外的带高/低触发选项的外部中断，10位ADC，一个8通道PCA带有空载时间控制的脉宽调制(PWM)，一个SPI，二个TWI(TWI0和TWI1)，第二串口（UART1），键盘中断，一次性使能的看门狗定时器，实时时钟(RTC)模块，两个低电压检测器，一个晶振(与 P6.0 和 P6.1共用)， 一个高精度的内部振荡(IHRCO)，一个片内时钟倍频器(CKM)来产生高速时钟源，一个低速的内部 RC 振荡器 (ILRCO) 和一个增型的串口(UART0)用来促进多处理器的通讯及一个速度增强设备 (X2/X4 模式)。

MA82G5BXX有多种工作模式可以减少耗电量：空闲模式，掉电模式，慢频模式，副频模式，RTC模式，watch 模式和 monitor 模式。 在空闲模式下，CPU被冻结而外围模块和中断系统依然活动。在掉电模式下，随机存储器RAM和特殊功能寄存器SFR的值被保存，而其他所有功能被终止。最重要的是，在掉电模式下的微控制器可以被多种中断或复位唤醒。在慢频模式， 使用者可以通过8位的系统时钟分频器减慢系统速度以减少耗电量。选择副频模式系统时钟来自内部低速振荡器CPU 用一个特别慢的速度在运行。实时时钟(RTC)模式支持所有模式下的实时时钟功能，watch 模式, 在掉电模式或空闲模式下保持WDT正常运行来唤醒CPU。Monitor 模式，在掉电模式检测电压，当电压特别低的时候会复位。

另外，**MA82G5BXX** 装配有笙泉独家的（OCD）接口可以用于在线仿真（ICE），OCD 接口提供在片内和在系统不干扰调试并且不占用任何资源。 支持 ICE 应用中的几个必须的操作例如： 复位、全速、停止、单步、全速到光标和断点设置。软件开发期间使用者不需要使用任何的开发板或者传统的 ICE 上应用的插头转接器，使用者只需要连接好 OCD 接口， 这强有力的接口使得开发非常容易。

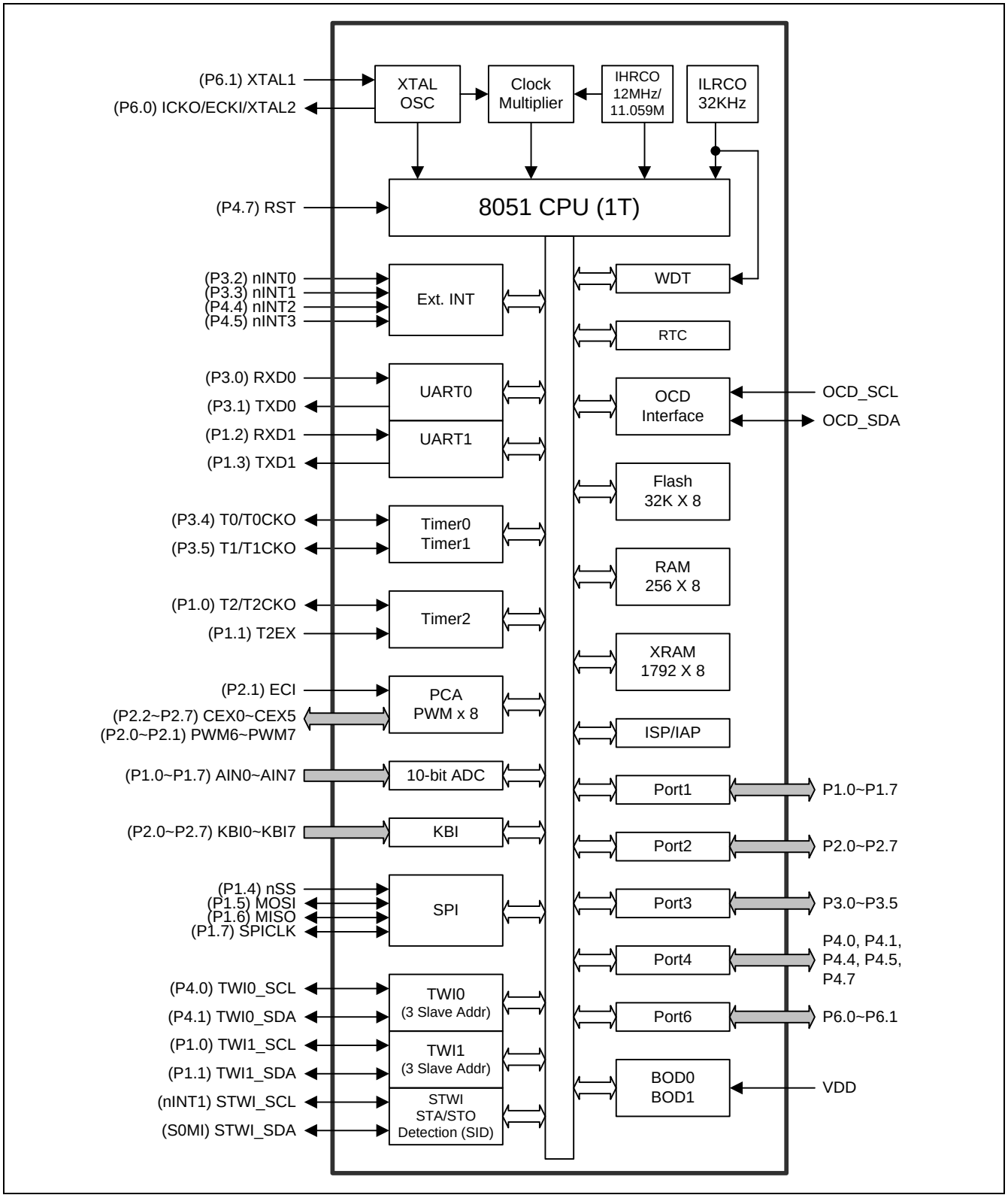
2. 采购信息

图 2–1. 采购信息



3. 方框图

图 3–1. 方框图


(P6.1) XTAL1
(P6.0) ICKO/ECKI/XTAL2
XTAL OSC
Clock Multiplier
IHRCO 12MHz/ 11.059M
ILRCO 32KHz
(P4.7) RST
8051 CPU (1T)
(P3.2) nINT0
(P3.3) nINT1
(P4.4) nINT2
(P4.5) nINT3
Ext. INT
WDT
RTC
(P3.0) RXD0
(P3.1) TXD0
UART0
(P1.2) RXD1
(P1.3) TXD1
UART1
OCD Interface
OCD_SCL
OCD_SDA
Flash 32K X 8
(P3.4) T0/T0CKO
(P3.5) T1/T1CKO
Timer0 Timer1
RAM 256 X 8
(P1.0) T2/T2CKO
(P1.1) T2EX
Timer2
XRAM 1792 X 8
(P2.1) ECI
(P2.2~P2.7) CEX0~CEX5
(P2.0~P2.1) PWM6~PWM7
PCA PWM x 8
ISP/IAP
(P1.0~P1.7) AIN0~AIN7
10-bit ADC
Port1
P1.0~P1.7
(P2.0~P2.7) KBI0~KBI7
KBI
Port2
P2.0~P2.7
(P1.4) nSS
(P1.5) MOSI
(P1.6) MISO
(P1.7) SPICLK
SPI
Port3
P3.0~P3.5
Port4
P4.0, P4.1, P4.4, P4.5, P4.7
(P4.0) TWI0_SCL
(P4.1) TWI0_SDA
TWI0 (3 Slave Addr)
Port6
P6.0~P6.1
(P1.0) TWI1_SCL
(P1.1) TWI1_SDA
TWI1 (3 Slave Addr)
(nINT1) STWI_SCL
(S0MI) STWI_SDA
STWI STA/STO Detection (SID)
BOD0 BOD1
VDD

4. 特殊功能寄存器

4.1. SFR 图(页 0~F)

表 4–1. SFR 图(页 0~F)

	页索引	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8	0*	P6	CH	CCAP0H*	CCAP1H*	CCAP2H*	CCAP3H*	CCAP4H*	CCAP5H*
	1*			CCAP6H*	CCAP7H*	--	--	--	--
F0	0*	B	PAOE	PCAPWM0*	PCAPWM1*	PCAPWM2*	PCAPWM3*	PCAPWM4*	PCAPWM5*
	1*			PCAPWM6*	PCAPWM7*	--	--	--	--
E8	0*	P4	CL	CCAP0L*	CCAP1L*	CCAP2L*	CCAP3L*	CCAP4L*	CCAP5L*
	1*			CCAP6L*	CCAP7L*	--	--	--	--
E0	0	ACC	WDTCR	IFD	IFADRH	IFADRL	IFMT	SCMD	ISPCR
	1								
D8	0*	CCON	CMOD	CCAPM0*	CCAPM1*	CCAPM2*	CCAPM3*	CCAPM4*	CCAPM5*
	1*			CCAPM7*	CCAPM7*	--	--	--	--
D0	0*	PSW	SIADR*	SIDAT*	SISTA*	SICON*	KBPATN	KBCON	KBMASK
	1*		SI1ADR*	SI1DAT*	SI1STA*	SI1CON*			
C8	0	T2CON	T2MOD	RCAP2L	RCAP2H	TL2	TH2	CLRL	CHRL
	1								
C0	0	XICON	XICFG	--	ADCFG0	ADCON0	ADCDL	ADCDH	CKCON0
	1								
B8	0	IP0L	SADEN	--	ADCFG1	PWMCR	PDTCR	RTCCR	CKCON1
	1								
B0	0*	P3	P3M0	P3M1	P4M0	PUCON0*	--	RTCTM	IP0H
	1*					PUCON1*	P6M0*		
A8	0	IE	SADDR	--	--	SFRPI*	EIE1	EIP1L	EIP1H
	1								
A0	0	P2	AUXR0	AUXR1	AUXR2	AUXR3	--	--	--
	1								
98	0*	S0CON*	S0BUF*	--	--	S0CFG*	--	--	--
	1*	S1CON*	S1BUF*	S1BRT*	S1BRC*	S1CFG*	--*		
	2*								
90	0	P1	P1M0	P1AIO	--	--	P2M0	BOREV	PCON1
	1								
88	0	TCON	TMOD	TL0	TL1	TH0	TH1	SFIE	--
	1								
80	0	--	SP	DPL	DPH	SPSTAT	SPCON	SPDAT	PCON0
	1								
		0/8	**1/9**	**2/A**	**3/B**	**4/C**	**5/D**	**6/E**	**7/F**

***:用户需要设置 SFRPI 为 SFRPI=0x00~0x02 作为 SFR 页访问**
（进入中断 MCU 不会保持 SFRPI 的值。用户需要使用软件来保持 SFRPI 的值）

SFRPI: SFR 页索引寄存器

SFR 页 = 0~F

SFR 地址 = 0xAC 复位值 = xxxx-0000

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

Bit 7~4: 保留。当 SFRPI 写时，这些位软件必须写“0”。

Bit 3~0: SFR 页索引。可用的页仅是页**“0”和“1”。**

PIDX[3:0]	可选页
0000	Page 0
0001	Page 1
0010	Page 2
0011	Page 3
……	……
……	……
……	……
1111	Page F

4.2. SFR 位分配 (页 0~F)

表 4–2. SFR 位分配 (页 0~F)

符号	描述	地址	位地址及符号								复位值
			位-7	位-6	位-5	位-4	位-3	位-2	位-1	位-0	
SP	堆栈指针	81H									00000111
DPL	数据指针低 8 位	82H									00000000
DPH	数据指针高 8 位	83H									00000000
SPSTAT	SPI 状态寄存器	84H	SPIF	WCOL	THRF	SPIBSY	MODF	--	--	SPR2	00000xx0
SPCON	SPI 控制寄存器	85H	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	00000100
SPDAT	SPI 数据寄存器	86H	SPDAT.7	SPDAT.6	SPDAT.5	SPDAT.4	SPDAT.3	SPDAT.2	SPDAT.1	SPDAT.0	00000000
PCON0	电源控制寄存器 0	87H	SMOD1	SMOD0	GF	POF0	GF1	GF0	PD	IDL	00010000
TCON	*定时器控制寄存器*	*88H*	*TF1*	*TR1*	*TF0*	*TR0*	*IE1*	*IT1*	*IE0*	*IT0*	*00000000*
TMOD	定时器模式寄存器	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	00000000
TL0	定时器 0 低 8 位	8AH									00000000
TL1	定时器 1 低 8 位	8BH									00000000
TH0	定时器 0 高 8 位	8CH									00000000
TH1	定时器 1 高 8 位	8DH									00000000
SFIE	系统标志中断使能	8EH	SIDFIE	MCDRE	MCDFIE	RTCFIE	--	BOF1IE	BOF0IE	WDTFIE	xxxxx000
P1	*端口 1*	*90H*	*P1.7*	*P1.6*	*P1.5*	*P1.4*	*P1.3*	*P1.2*	*P1.1*	*P1.0*	*11111111*
P1M0	P1 模式寄存器 0	91H	P1M0.7	P1M0.6	P1M0.5	P1M0.4	P1M0.3	P1M0.2	P1M0.1	P1M0.0	00000000
P1AIO	P1 仅模拟输入寄存器	92H	P17AIO	P16AIO	P15AIO	P14AIO	P13AIO	P12AIO	P11AIO	P10AIO	00000000
P2M0	P2 模式寄存器 0	95H	P2M0.7	P2M0.6	P2M0.5	P2M0.4	P2M0.3	P2M0.2	P2M0.1	P2M0.0	00000000
BOREV	**位序反转**	**96H**									00000000
PCON1	电源控制寄存器 1	97H	SWRF	EXRF	**MCDF**	**RTCF**	--	BOF1	BOF0	WDTF	0000x000
S0CON	*串行口 0 控制寄存器*	*98H*	*SM00 /FE*	*SM10*	*SM20*	*REN0*	*TB80*	*RB80*	*TI0*	*RI0*	*00000000*
S1CON	*串行口 1 控制寄存器*	*98H*	*SM01*	*SM11*	*SM21*	*REN1*	*TB81*	*RB81*	*TI1*	*RI1*	*00000000*
----	----	*98H*	--	--	--	--	--	--	--	--	----
S0BUF	串行口 0 缓冲寄存器	99H									xxxxxxxx
S1BUF	串行口 1 缓冲寄存器	99H									xxxxxxxx
----	----	99H	--	--	--	--	--	--	--	--	----
----	----	9AH	--	--	--	--	--	--	--	--	----
S1BRT	串行口 1 波特率定时器	9AH									00000000
----	----	9AH	--	--	--	--	--	--	--	--	----
----	----	9BH	--	--	--	--	--	--	--	--	----
S1BRC	串行口 1 波特率计数器	9BH									00000000
----	----	9BH	--	--	--	--	--	--	--	--	----
S0CFG	串行口 0 配置	9CH	URTS	SMOD2	URM0X3	SM30	S0DOR	BTI	UTIE	--	0000100x
S1CFG	串行口 1 配置	9CH	SM31	S1EVPS	S1DOR	S1TR	S1MOD1	S1TX12	S1CKOE	S1TME	00100000
----	----	9CH	--	--	--	--	--	--	--	--	----
----	----	9DH	--	--	--	--	--	--	--	--	----
----	----	9DH	--	--	--	--	--	--	--	--	----
----	----	9DH	--	--	--	--	--	--	--	--	----
P2	*端口 2*	*A0H*	*P2.7*	*P2.6*	*P2.5*	*P2.4*	*P2.3*	*P2.2*	*P2.1*	*P2.0*	*11111111*
AUXR0	辅助寄存器 0	A1H	P60OC1	P60OC0	P60FD	T0XL	P4FS1	P4FS0	INT1H	INT0H	00000000
AUXR1	辅助寄存器 1	A2H	P1KBIH	P3KBIL	P4SPI	P3S1	P3S1MI	P6TWI0	P3CEX	DPS	00000000
AUXR2	辅助寄存器 2	A3H	INT3IS1	INT3IS0	INT2IS1	INT2IS0	T1X12	T0X12	T1CKOE	T0CKOE	00000000
AUXR3	辅助寄存器 3	A4H	STAF	STOF	BPOC1	BPOC0	GF	P1S0MI	P3ECI	P3TWI1	00000000
IE	中断使能	A8H	EA	GF4	ET2	ES0	ET1	EX1	ET0	EX0	00000000
SADDR	从机地址	A9H									00000000
SFRPI	SFR 页索引	ACH	--	--	--	--	IDX3	IDX2	IDX1	IDX0	xxxx0000
EIE1	扩展中断使能 1	ADH	ETWI1	ETWI0	EKB	ES1	ESF	EPCA	EADC	ESPI	00000000
EIP1L	扩展中断优先级 1 低	AEH	PTWI1L	PTWI0L	PKBL	PS1L	PSFL	PPCAL	PADCL	PSPIL	00000000
EIP1H	扩展中断优先级 1 高	AFH	PTWI1H	PTWI0H	PKBH	PS1H	PSFH	PPCAH	PADCH	PSPIH	00000000
P3	*端口 3*	*B0H*	--	--	*P3.5*	*P3.4*	*P3.3*	*P3.2*	*P3.1*	*P3.0*	*xx111111*
P3M0	P3 模式寄存器 0	B1H	--	--	P3M0.5	P3M0.4	P3M0.3	P3M0.2	P3M0.1	P3M0.0	00000000
P3M1	P3 模式寄存器 1	B2H	--	--	P3M1.5	P3M1.4	P3M1.3	P3M1.2	P3M1.1	P3M1.0	00000000
P4M0	P4 模式寄存器 0	B3H	P4M0.7	--	P4M0.5	P4M0.4	--	--	P4M0.1	P4M0.0	0x00xx00
PUCON0	上拉控制寄存器 0	B4H	P4PU1	P4PU0	P2PU1	P2PU0	P1PU1	P1PU0	--	--	000000xx
PUCON1	上拉控制寄存器 1	B4H	--	--	--	--	--	P6PU0	--	--	xxxxx0xx
----	----	B5H	--	--	--	--	--	--	--	--	----
P6M0	P6 模式寄存器 0	B5H	--	--	--	--	--	--	P6M0.1	P6M0.0	xxxxxx00
IP0H	中断优先级 0 高	B7H	PX3H	PX2H	PT2H	PSH	PT1H	PX1H	PT0H	PX0H	00000000

IP0L	中断优先级 0 低	*B8H*	*PX3L*	*PX2L*	*PT2L*	*PSL*	*PT1L*	*PX1L*	*PT0L*	*PX0L*	*00000000*
SADEN	从机地址屏蔽	B9H									00000000
ADCFG1	ADC 配置 1	BBH	--	VRS2	VRS1	SIGN	AOS.3	AOS.2	AOS.1	AOS.0	x0000000
PWMCR	PWM 控制	BCH	PCAE	EXDT	--	PFCF	PFCM	PFCS2	PFCS1	PFCS0	00x00000
PDTCR	PWM 空载时间控制	BDH	DTPS1	DTPS0	DT.5	DT.4	DT.3	DT.2	DT.1	DT.0	00000000
RTCCR	RTC 控制寄存器	BEH	RTCE	RTCO	RTCRL5	RTCRL4	RTCRL3	RTCRL2	RTCRL1	RTCRL0	00111111
CKCON1	时钟控制寄存器 1	BFH	--	--	XCKS5	XCKS4	XCKS3	XCKS2	XCKS1	XCKS0	xx000000
XICON	外部中断控制	C0H	INT3H	EX3	IE3	IT3	INT2H	EX2	IE2	IT2	00000000
XICFG	外部中断配置	C1H	INT1IS1	INT1IS0	INT0IS1	INT0IS0	X3FLT	X2FLT	X1FLT	X0FLT	00000000
ADCFG0	ADC 配置 0	C3H	ADCKS2	ADCKS1	ADCKS0	ADRJ	ADPS	VRS0	ADTM1	ADTM0	00000000
ADCON0	ADC 控制 0	C4H	ADCEN	--	CH3	ADCI	ADCS	CHS2	CHS1	CHS0	0x000000
ADCDL	ADC 数据低	C5H	ADCV.1	ADCV.0	--	--	--	--	--	--	00xxxxxx
ADCDH	ADC 数据高	C6H	ADCV.9	ADCV.8	ADCV.7	ADCV.6	ADCV.5	ADCV.4	ADCV.3	ADCV.2	00000000
CKCON0	时钟控制寄存器 0	C7H	AFS	ENCKM	CKMIS1	CKMIS0	CCKS	SCKS2	SCKS1	SCKS0	00010000
T2CON	*定时器 2 控制 I*	*C8H*	*TF2*	*EXF2*	*RCLK*	*TCLK*	*EXEN2*	*TR2*	*C/T2*	*CP/RL2*	*00000000*
T2MOD	定时器 2 模式	C9H	--	--	T2EXH	T2X12	--	--	T2OE	DCEN2	xx00xx00
RCAP2L	定时器 2 捕获低字节	CAH									00000000
RCAP2H	定时器 2 捕获高字节	CBH									00000000
TL2	定时器 2 低字节	CCH									00000000
TH2	定时器 2 高字节	CDH									00000000
CLRL	CL 重载寄存器	CEH									00000000
CHRL	CH 重载寄存器	CFH									00000000
PSW	*程序状态字*	*D0H*	*CY*	*AC*	*F0*	*RS1*	*RS0*	*OV*	*F1*	*P*	*00000000*
SIADR	TWI0 地址寄存器	D1H								GC	00000000
SI1ADR	TWI1 地址寄存器	D1H								GC1	00000000
SIDAT	TWI0 数据寄存器	D2H									00000000
SI1DAT	TWI1 数据寄存器	D2H									00000000
SISTA	TWI0 状态寄存器	D3H									11111000
SIS1TA	TWI1 状态寄存器	D3H									11111000
SICON	TWI0 控制寄存器	D4H	CR2	ENSI	STA	STO	SI	AA	CR1	CR0	00000000
SI1CON	TWI1 控制寄存器	D4H	CR21	ENSI1	STA1	STO1	SI1	AA1	CR11	CR01	00000000
KBPATN	键盘模式	D5H									11111111
KBCON	键盘控制	D6H	--	--	--	--	--	--	PATNS	KBIF	xxxxxx01
KBMASK	键盘中断掩码	D7H									00000000
CCON	*PCA 控制寄存器*	*D8H*	*CF*	*CR*	*CCF5*	*CCF4*	*CCF3*	*CCF2*	*CCF1*	*CCF0*	*00000000*
CMOD	PCA 模式寄存器	D9H	CIDL	BME4	BME2	BME0	CPS2	CPS1	CPS0	ECF	00000000
CCAPM0	PCA 模块 0 模式	DAH	DTE0	ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0	00000000
CCAPM6	PCA 模块 6 模式	DAH	BME6						PWM6		0xxxxxx0x
CCAPM1	PCA 模块 1 模式	DBH	--	ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1	x0000000
CCAPM7	PCA 模块 7 模式	DBH							PWM7		xxxxxxx0x
CCAPM2	PCA 模块 2 模式	DCH	DTE2	ECOM2	CAPP2	CAPN2	MAT2	TOG2	PWM2	ECCF2	00000000
----	----	DCH	--	--	--	--	--	--	--	--	----
CCAPM3	PCA 模块 3 模式	DDH	--	ECOM3	CAPP3	CAPN3	MAT3	TOG3	PWM3	ECCF3	x0000000
----	----	DDH	--	--	--	--	--	--	--	--	----
CCAPM4	PCA 模块 4 模式	DEH	DTE4	ECOM4	CAPP4	CAPN4	MAT4	TOG4	PWM4	ECCF4	00000000
----	----	DEH	--	--	--	--	--	--	--	--	----
CCAPM5	PCA 模块 5 模式	DFH	--	ECOM5	CAPP5	CAPN5	MAT5	TOG5	PWM5	ECCF5	x0000000
----	----	DFH	--	--	--	--	--	--	--	--	----
ACC	*累加器*	*E0H*									*00000000*
WDTCR	看门狗控制寄存器	E1H	WREN	NSW	ENW	CLRW	WIDL	PS2	PS1	PS0	00000000
IFD	ISP Flash 数据	E2H									11111111
IFADRH	ISP Flash 地址高 8 位	E3H									00000000
IFADRL	ISP Flash 地址低 8 位	E4H									00000000
IFMT	ISP 模式表	E5H	GF	GF	GF	GF	GF	MS.2	MS.1	MS.0	00000000
SCMD	ISP 系列命令	E6H									xxxxxxxx
ISPCR	ISP 控制寄存器	E7H	ISPEN	SWBS	SWRST	CFAIL	--	DATM2	DATM1	DATM0	0000x000
P4	*端口 4*	*E8H*	*P4.7*	*--*	*P4.5*	*P4.4*	*--*	*--*	*P4.1*	*P4.0*	*1x11xx11*
CL	PCA 基准定时器低	E9H									00000000
CCAP0L	PCA 模块 0 捕获低 8 位	EAH									00000000
CCAP6L	PCA 模块 6 捕获低 8 位	EAH									00000000
CCAP1L	PCA 模块 1 捕获低 8 位	EBH									00000000

CCAP7L	PCA 模块 7 捕获低 8 位	EBH									00000000
CCAP2L	PCA 模块 2 捕获低 8 位	ECH									00000000
----	----	ECH	--	--	--	--	--	--	--	--	----
CCAP3L	PCA 模块 3 捕获低 8 位	EDH									00000000
----	----	EDH	--	--	--	--	--	--	--	--	----
CCAP4L	PCA 模块 4 捕获低 8 位	EEH									00000000
----	----	EEH	--	--	--	--	--	--	--	--	----
CCAP5L	PCA 模块 5 捕获低 8 位	EFH									00000000
----	----	EFH	--	--	--	--	--	--	--	--	----
B	*B 寄存器*	*F0H*									*00000000*
PCAPWM0	PCA PWM0 模式	F2H	P0RS1	P0RS0	P0PS2	P0PS1	P0PS0	P0INV	EPC0H	EPC0L	00000000
PCAPWM6	PCA PWM6 模式	F2H	P6RS1	P6RS0	P6PS2	P6PS1	P6PS0	P6INV	EPC6H	EPC6L	00000000
PCAPWM1	PCA PWM1 模式	F3H	P1RS1	P1RS0	P1PS2	P1PS1	P1PS0	P1INV	EPC1H	EPC1L	00000000
PCAPWM7	PCA PWM7 模式	F3H	P7RS1	P7RS0	P7PS2	P7PS1	P7PS0	P7INV	EPC7H	EPC7L	00000000
PCAPWM2	PCA PWM2 模式	F4H	P2RS1	P2RS0	P2PS2	P2PS1	P2PS0	P2INV	EPC2H	EPC2L	00000000
----	----	F4H	--	--	--	--	--	--	--	--	----
PCAPWM3	PCA PWM3 模式	F5H	P3RS1	P3RS0	P3PS2	P3PS1	P3PS0	P3INV	EPC3H	EPC3L	00000000
----	----	F5H	--	--	--	--	--	--	--	--	----
PCAPWM4	PCA PWM4 模式	F6H	P4RS1	P4RS0	P4PS2	P4PS1	P4PS0	P4INV	EPC4H	EPC4L	00000000
----	----	F6H	--	--	--	--	--	--	--	--	----
PCAPWM5	PCA PWM5 模式	F7H	P5RS1	P5RS0	P5PS2	P5PS1	P5PS0	P5INV	EPC5H	EPC5L	00000000
----	----	F7H	--	--	--	--	--	--	--	--	----
P6	*端口 6*	*F8H*	*P6.7*	*P6.6*	*P6.5*	*P6.4*	*P6.3*	*P6.2*	*P6.1*	*P6.0*	*11111111*
CH	PCA 基准定时器高	F9H									00000000
CCAP0H	PCA 模块 0 捕获高 8 位	FAH									00000000
CCAP6H	PCA 模块 6 捕获高 8 位	FAH									00000000
CCAP1H	PCA 模块 1 捕获高 8 位	FBH									00000000
CCAP7H	PCA 模块 7 捕获高 8 位	FBH									00000000
CCAP2H	PCA 模块 2 捕获高 8 位	FCH									00000000
----	----	FCH	--	--	--	--	--	--	--	--	----
CCAP3H	PCA 模块 3 捕获高 8 位	FDH									00000000
----	----	FDH	--	--	--	--	--	--	--	--	----
CCAP4H	PCA 模块 4 捕获高 8 位	FEH									00000000
----	----	FEH	--	--	--	--	--	--	--	--	----
CCAP5H	PCA 模块 5 捕获高 8 位	FFH									00000000
----	----	FFH	--	--	--	--	--	--	--	--	----

4.3. 辅助 SFR 图 (P 页)

MA82G5BXX 特殊功能寄存器（SFR）有一个辅助索引 P 页，它写的方法跟标准的 8051 特殊功能寄存器的不一样。象访问 ISP/IAP 一样通过设置 IFMT 和 SCMD 来访问这个辅助的特殊功能寄存器。P 页有 256 字节有用到的为 6 个物理字节地址和 8 个逻辑字节地址。6 个物理字节地址包括 IAPLB, CKCON2, CKCON3, PCON2, SPCON0 和 DCON0。8 个逻辑字节地址包括 PCON0, PCON1, RTCCR, CKCON0, CKCON1, WDTCR, P4 和 P6。在 0~F 写这 8 个逻辑地址会得到相同的特殊功能（SFR）值。更多详细的信息请参考章节 “27 P 页 SFR 访问”

表 4–3. 辅助 SFR 图(P 页)

		0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8		P6	--	--	--	--	--	--	--
F0		--	--	--	--	--	--	--	--
E8		P4	--	--	--	--	--	--	--
E0		--	WDTCR	--	--	--	--	--	--
D8		--	--	--	--	--	--	--	--
D0		--	--	--	--	--	--	--	--
C8		--	--	--	--	--	--	--	--
C0		--	--	--	--	--	--	--	CKCON0
B8		--	--	--	--	--	--	RTCCR	CKCON1
B0		--	--	--	--	--	--	--	--
A8		--	--	--	--	--	--	--	--
A0		--	--	--	--	--	--	--	--
98		--	--	--	--	--	--	--	--
90		--	--	--	--	--	--	--	PCON1
88		--	--	--	--	--	--	--	
80		--	--	--	--	--	--	--	PCON0
78		--	--	--	--	--	--	--	--
70		--	--	--	--	--	--	--	--
68		--	--	--	--	--	--	--	--
60		--	--	--	--	--	--	--	--
58		--	--	--	--	--	--	--	--
50		--	--	--	--	--	--	--	--
48		SPCON0	--	--	--	DCON0	--	--	--
40		CKCON2	CKCON3	--	--	PCON2		--	--
38		--	--	--	--	--	--	--	--
30		--	--	--	--	--	--	--	--
28		--	--	--	--	--	--	--	--
20		--	--	--	--	--	--	--	--
18		--	--	--	--	--	--	--	--
10		--	--	--	--	--	--	--	--
08		--	--	--	--	--	--	--	--
00		--	--	--	IAPLB	--	--	--	--
		0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F

4.4. 辅助特殊功能寄存器位分配 (P 页)

表 4–4. 辅助 SFR 位分配(P 页)

符号	描述	地址	位地址及符号								复位值
			Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0	
物理字节											
IAPLB	IAP 低边界	03H	IAPLB6	IAPLB5	IAPLB4	IAPLB3	IAPLB2	IAPLB1	IAPLB0	0	01110110
CKCON2	时钟控制 2	40H	XTGS1	XTGS0	XTALE	IHRCOE	MCKS1	MCKS0	OSCS1	OSCS0	01010000
CKCON3	时钟控制 3	41H	--	--	FWKP	--	MCKD1	MCKD0	MCDS1	MCDS0	xx0x0010
PCON2	电源控制 2	44H	AWBOD1	0	BO1S1	BO1S0	BO1RE	EBOD1	BO0RE	1	00000101
SPCON0	SFR 页控制 0	48H	RTCCTL	P6CTL	P4CTL	WRCTL	CKCTL1	CKCTL0	PWCTL1	PWCTL0	00000000
DCON0	设备控制 0	4CH	0	IAPO	0	0	0	IORCTL	RSTIO	OCDE	00000011
逻辑字节											
PCON0	电源控制 0	87H	SMOD1	SMOD0	GF	POF0	GF1	GF0	PD	IDL	00010000
PCON1	电源控制 1	97H	SWRF	EXRF	MCDF	RTCF	--	BOF1	BOF0	WDTF	0000x000
RTCCR	RTC 控制寄存器	BEH	RTCE	RTCO	RTCRL.5	RTCRL.4	RTCRL.3	RTCRL.2	RTCRL.1	RTCRL.0	00111111
CKCON1	时钟控制 1	BFH	--	--	XCKS5	XCKS4	XCKS3	XCKS2	XCKS1	XCKS0	xx000000
CKCON0	时钟控制 0	C7H	AFS	ENCKM	CKMIS1	CKMIS0	CCKS	SCKS2	SCKS1	SCKS0	00010000
WDTCR	看门狗控制寄存器	E1H	WREN	NSW	ENW	CLRW	WIDL	PS2	PS1	PS0	00000000
P4	端口 4	E8H	P4.7	--	P4.5	P4.4	--	--	P4.1	P4.0	1x11xx11
P6	端口 6	F8H	--	--	--	--	--	--	P6.1	P6.0	xxxxxx11

写入 P 页 SFR 示例代码：

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

5. 引脚结构

5.1. 封装指南

图 5–1. MA82G5BXXAD32 顶视图

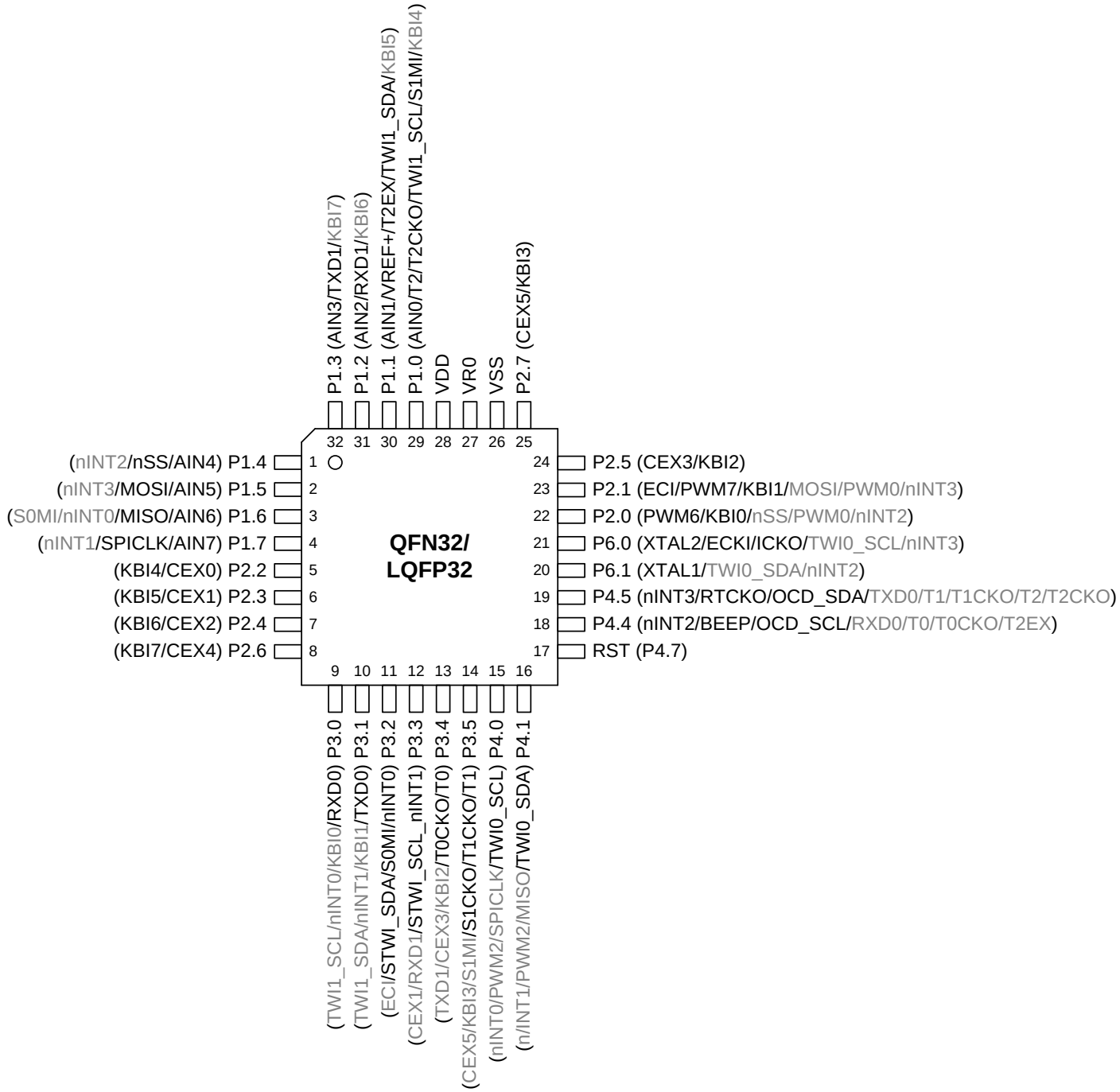


图 5–2. MA82G5BxxAS28 顶视图

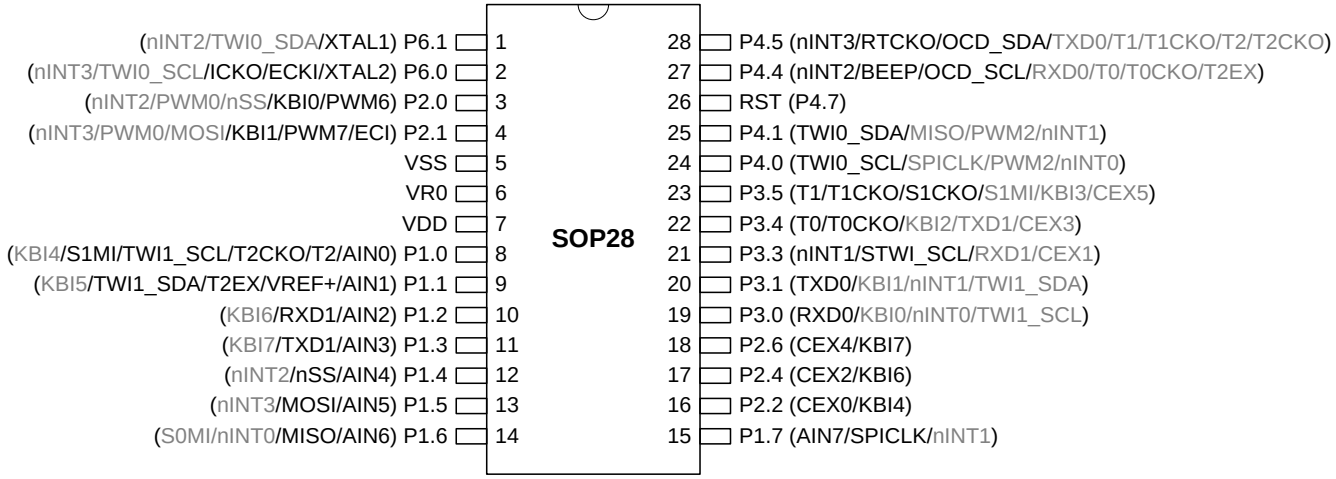


图 5–3. MA82G5BxxAS20 顶视图

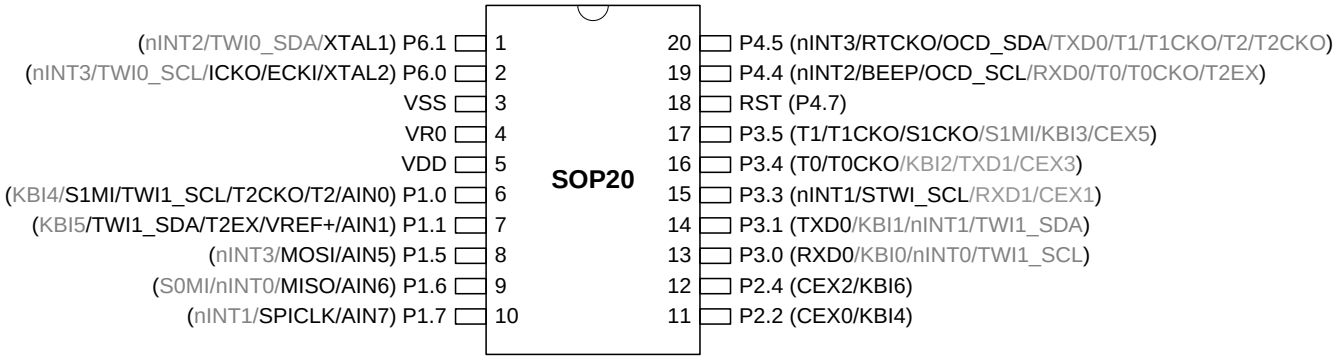
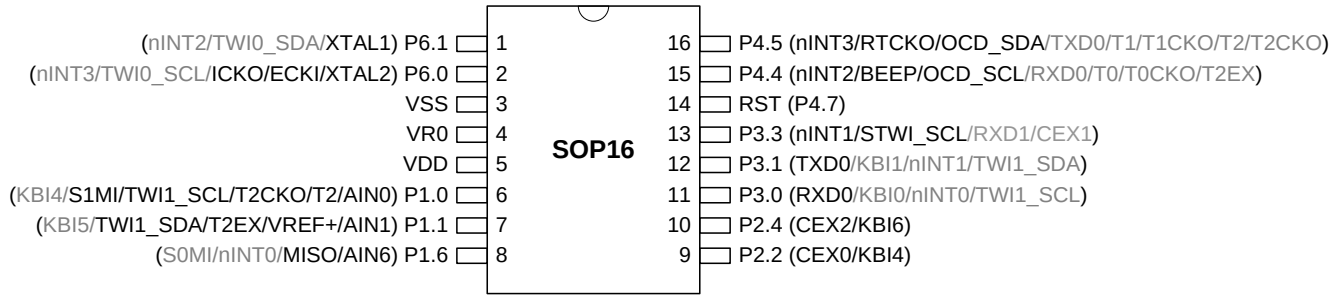


图 5–4. MA82G5BxxAS16 顶视图



5.2. 引脚定义

表 5–1. 引脚定义

助记符	引脚号				I/O 类型	描述
	32-Pin LQFP	28-Pin SOP	20-Pin SOP	16-Pin SOP		
P1.0 (AIN0) (T2) (T2CKO) (TWI1_SCL) (S1MI)	29	8	6	6	I/O	* 端口 1.0. * AIN0: ADC 通道 0 模拟输入 * T2: 定时器/计数器 2 外部时钟输入 * T2CKO: 定时器 2 可编程时钟输出 * TWI1_SCL: TWI1 串行时钟 * S1MI: 在 UART1 模式下 SPI 主机输入
P1.1 (AIN1) (T2EX) (TWI1_SDA)	30	9	7	7	I/O	* 端口 1.1. * AIN1: ADC 通道 1 模拟输入 * T2EX: 定时器/计数器 2 外部控制输入 * TWI1_SDA: TWI1 串行数据.
P1.2 (AIN2) (RXD1)	31	10			I/O	* 端口 1.2. * AIN2: ADC 通道 2 模拟输入 * RXD1: UART1 串行输入口
P1.3 (AIN3) (TXD1)	32	11			I/O	*端口 1.3. * AIN3: ADC 通道 3 模拟输入 * TXD1: UART1 串行输出口
P1.4 (AIN4) (nSS)	1	12			I/O	*端口 1.4. * AIN4: ADC 通道 4 模拟输入 * nSS: SPI 从机选择
P1.5 (AIN5) (MOSI)	2	13	8		I/O	*端口 1.5. * AIN5: ADC 通道 5 模拟输入 * MOSI: SPI 主机输出及从机输入
P1.6 (AIN6) (MISO)	3	14	9	8	I/O	*端口 1.6. * AIN6: ADC 通道 6 模拟输入 * MISO: SPI 主机输入及从机输出
P1.7 (AIN7) (SPICLK)	4	15	10		I/O	*端口 1.7. * AIN7: ADC 通道 7 模拟输入 * SPICLK: SPI 时钟,主机时钟输出和从机时钟输入
P2.0 (PWM6) (KBI0)	22	3			I/O	*端口 2.0. * PWM6: PCA 模块 6 PWM6 输出 * KBI0: 键盘输入 0
P2.1 (ECI) (PWM7) (KBI1)	23	4			I/O	*端口 2.1. * ECI: PCA 外部时钟输入 * PWM7: PCA 模块 7 PWM7 输出 * KBI1: 键盘输入 1
P2.2 (CEX0) (KBI4)	5	16	11	9	I/O	*端口 2.2. * CEX0: PCA 模块 0 外部输入/输出 * KBI4: 键盘输入 4
P2.3 (CEX1) (KBI5)	6				I/O	*端口 2.3. * CEX1: PCA 模块 1 外部输入/输出 * KBI5: 键盘输入 5
P2.4 (CEX2) (KBI6)	7	17	12	10	I/O	*端口 2.4. * CEX2: PCA 模块 2 外部输入/输出 * KBI6: 键盘输入 6
P2.5 (CEX3) (KBI2)	24				I/O	*端口 2.5. * CEX3: PCA 模块 3 外部输入/输出 * KBI2: 键盘输入 2
P2.6 (CEX4) (KBI7)	8	18			I/O	*端口 2.6. * CEX4: PCA 模块 4 外部输入/输出 * KBI7: 键盘输入 7
P2.7 (CEX5) (KBI3)	25				I/O	*端口 2.7. * CEX5: PCA 模块 5 外部输入/输出 * KBI3: 键盘输入 3

P3.0 (RXD0)	9	19	13	11	I/O	*端口 3.0. * RXD0: UART0 串行输入口
P3.1 (TXD0)	10	20	14	12	I/O	*端口 3.1. * TXD0: UART0 串行输出口
P3.2 (nINT0) (S0MI)	11				I/O	*端口 3.2. * nINT0: 外部中断 0 输入 * S0MI: 在 UART0 模式下 SPI 主机输入
P3.3 (nINT1)	12	21	15	13	I/O	*端口 3.3. * nINT1: 外部中断 1 输入
P3.4 (T0) (T0CKO)	13	22	16		I/O	*端口 3.4. * T0: 定时器/计数器 0 外部时钟输入 * T0CKO: 定时器 0 可编程时钟输出
P3.5 (T1) (T1CKO) (S1CKO)	14	23	17		I/O	*端口 3.5. * T1: 定时器/计数器 1 外部时钟输入 * T1CKO: 定时器 1 可编程时钟输出 * S1CKO: S1BRT 可编程时钟输出
P4.0 (TWI0_SCL)	15	24			I/O	*端口 4.0. * TWI0_SCL: TWI0 串行时钟
P4.1 (TWI0_SDA)	16	25			I/O	*端口 4.1. * TWI0_SDA: TWI0 串行数据
P4.4 (nINT2) (BEEP) (OCD_SCL)	18	27	19	15	I/O	*端口 4.4. * nINT2: 外部中断 2 输入 * BEEP: 报警器输出 * OCD_SCL: OCD 接口, 串行时钟
P4.5 (nINT3) (RTCKO) (OCD_SDA)	19	28	20	16	I/O	*端口 4.5. * nINT3: 外部中断 3 输入 * RTCKO: RTC 可编程时钟输出 * OCD_SDA: OCD 接口, 串行数据
P6.0 (XTAL2) (ECKI) (ICKO)	21	2	2	2	I/O O I O	*端口 6.0. * XTAL2: 片上晶振振荡电路输出 * ECKI: 外部时钟输入模式,为时钟输入脚 * ICKO: 使能 IHRCO/ILRCO 输出
P6.1 (XTAL1)	20	1	1	1	I/O I	*端口 6.1. * XTAL1: 片上晶振振荡电路输入
RST (P4.7)	17	26	18	14	I	* RST: 外部复位(RESET)输入,高电平有效 *端口 4.7.
VR0	27	6	4	4	I/O	* VR0. 电压参考点 0。接 0.1uF 电容和 4.7uF 电容到 VSS
VDD	28	7	5	5	P	电源供应输入
VSS	26	5	3	3	G	地, 0V 参考电压

5.3. 功能复用

许多 I/O 口，除了普通的 I/O 口功能之外，还能复用其他内部功能。如：键盘中断(KBI)、PCA、SPI、UART0、UART1、TWI0，TWI1 和外部中断(nINT0~3)，Port 1, Port 2 , Port 3，Port 4 和 Port 6 默认的是 I/O 口功能，但是，使用者可以设置 Port 4 和 Port 5 为其它复用功能通过设置 AUXR1 寄存器中的位 P4KB, P4PCA, P5SPI 和 P4S1，使用者注意任意时间只能设置一个位（4 个位中），特别注意当封装大于 40 个脚时。

AUXR0: 辅助寄存器 0

SFR 页 = 0~F

SFR 地址 = 0xA1 复位值 = 000X-0000

7	6	5	4	3	2	1	0
P60FC1	P60FC0	P60FD	T0XL	P4FS1	P4FS0	INT1H	INT0H
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7~6: P6.0 功能配置控制位 1 和位 0，这两位仅仅当内部 RC 振荡（IHRCO 或 ILRCO）被选择为系统时钟源时有效。这种情况， XTAL2 和 XTAL1 改变功能作 P6.0 和 P6.1，当外部时钟输入模式，P6.0 专用于时钟输入。在内部振荡模式， P6.0 为普通 I/O 或时钟源发生器提供下列选项， 当 P60OC[1:0] 索引为非 P6.0 GPIO 功能时，P6.0 将驱动内部 RC 振荡器输出为其它设备提供时钟源。

P60OC[1:0]	P60 function	I/O mode
00	P60	By P6M0.0
01	MCK	By P6M0.0
10	MCK/2	By P6M0.0
11	MCK/4	By P6M0.0

了解详情，请参考“9 系统时钟” P6.0 作为时钟输出功能时, 建议设置 P6M0.0 为“1” 来选着 P6.0 为推挽输出模式。

Bit 5: P60FD, P6.0 快速驱动标志。

0: P6.0 默认驱动输出。

1: P6.0 快速驱动输出使能。若 P6.0 被配置为时钟输出， 当 P4.0 输出频率大于 12MHz （5V） 或者大于 6MHz （3V） 时使能此位。

Bit 3~2: P4.4 和 P4.5 复用功能选项。

P4FS[1:0]	P4.4	P4.5
00	P4.4	P4.5
01	RXD0	TXD0
10	T0/T0CKO	T1/T1CKO
11	T2EX	T2/T2CKO

AUXR1: 辅助寄存器 1

SFR 页 = 0~F

SFR 地址 = 0xA2 复位值 = 0000-0000

7	6	5	4	3	2	1	0
P1KBIH	P3KBIL	P4SPI	P3S1	P3S1MI	P6TWI	P3CEX	DPS
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7: P1KBIH, 键盘中断(KBI)高 4 位端口选择在 P1.3, P1.2, P1.1 和 P1.0

P1KBIH	KBI.7~4
0	P2.6, P2.4, P2.3, P2.2
1	P1.3, P1.2, P1.1, P1.0

Bit 6: P3KBIL, 键盘中断(KBI)低 4 位端口选择在 P3.5, P3.4, P3.1 和 P3.0

P3KBIL	KBI.3~0
0	P2.7, P2.5, P2.1, P2.0
1	P3.5, P3.4, P3.1, P3.0

Bit 5: P4SPI, SPI 接口在 P4.1~P4.0 和 P2.1~P2.0

P4SPI	nSS	MOSI	MISO	SPICLK
0	P1.4	P1.5	P1.6	P1.7
1	P2.0	P2.1	P4.1	P4.0

Bit 4: P3S1,串行口 1 (UART1) 功能在 P3.3 和 P3.4，如果 P3CEX (AUXR1.1)是禁止

P3S1	RXD1	TXD1
0	P1.2	P1.3
1	P3.3	P3.4

Bit 3: P3S1MI, S1MI 功能在 P3.5。S1MI 是 S1 模式 4 的 SPI 串行输入(SPI 主机)

P3S1MI	S1MI
0	P1.0
1	P3.5

Bit 2: P6TWI0, TWI0 功能在 P6。当 P60OC[1:0]等于“00”此功能有效

P6TWI0	TWI0_SCL	TWI0_SDA
0	P4.0	P4.1
1	P6.0	P6.1

Bit 1: P3CEX, CEX5, CEX3 和 CEX1 的功能在 P3.5, P3.4 和 P3.3.

P3CEX	CEX5	CEX3	CEX1
0	P2.7	P2.5	P2.3
1	P3.5	P3.4	P3.3

AUXR2: 辅助功能寄存器 2

SFR 页 = 0~F

SFR 地址 = 0xA3 复位值 = 0000-0000

7	6	5	4	3	2	1	0
INT3IS1	INT3IS0	INT2IS1	INT2IS0	T1X12	T0X12	T1CKOE	T0CKOE
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~6: INT3IS1~0, nINT3 输入功能选择位定义如下表

INT3IS1~0	nINT3
00	P4.5
01	P2.1
10	P1.5
11	P6.0

Bit 5~4: INT2IS1~0, nINT2 输入功能选择位定义如下表

INT2IS1~0	nINT2
00	P4.4
01	P2.0
10	P1.4
11	P6.1

AUXR3: 辅助功能寄存器 3

SFR 页 = 0~F

SFR 地址 = 0xA4 复位值 = 0000-0000

7	6	5	4	3	2	1	0
STAF	STOF	BPOC1	BPOC0	GF	P1S0MI	P3ECI	P3TWI1
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 2: P1S0MI, S0MI 功能在 P1.6

P1S0MI	S0MI
0	P3.2
1	P1.6

Bit 1: P3ECI, ECI 功能在 P3.2

P3ECI	ECI
0	P2.1
1	P3.2

Bit 0: P3TWI1, TWI1 功能在 P3

P3TWI1	TWI1_SCL	TWI1_SDA
0	P1.0	P1.1
1	P3.0	P3.1

6. 8051 CPU 功能描述

6.1. CPU 寄存器

PSW: 程序状态字

SFR 页 = 0~F

SFR 地址 = 0xD0 复位值 = 0000-0000

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

CY: 进位标志
AC: 辅助进位标志
F0: 用户可设定的标志位 0
RS1: 寄存器组选择位 1
RS0: 寄存器组选择位 0
OV: 溢出标志
F1: 用户可设定的标志位 1
P: 奇偶标志

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

进位标志，不仅有算术运算的进位功能，也充当许多布尔运算的“累加器”。

RS0 和 RS1 被用来选择 4 组中的任意一组寄存器组，祥见章节“7.2 片内数据存储器 RAM”。

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

SP: 堆栈指针

SFR 页 = 0~F

SFR 地址 = 0x81 复位值 = 0000-0111

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

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

DPL: 数据指针低字节

SFR 页 = 0~F

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

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

DPH: 数据指针高字节

SFR 页 = 0~F

SFR 地址 = 0x83 复位值 = 0000-0000

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

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

ACC: 累加器

SFR 页 = 0~F

SFR 地址 = 0xE0 复位值 = 0000-0000

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

算术运算的累加器。

B: B 寄存器

SFR 页 = 0~F

SFR 地址 = 0xF0 复位值 = 0000-0000

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

另一个算术运算的累加器。

6.2. CPU 时序

MA82G5BXX 是基于 80C51 的高效 1-T 结构的单芯片微处理器，与 8051 指令集兼容，每条指令需要 1~7 个时钟信号(比标准 8051 快 6~7 倍)。使用流线型结构同标准的 8051 结构比较大大增加了指令完成的速度，指令的时序也和标准的 8051 不同。

多数8051执行指令，一个区别是建立在机器周期和时钟周期之间，机器周期来自2到12个时钟周期长度。然而，1-T结构的80C51执行指令是基于单独的时钟周期时序。所有指令时序被指定在时钟周期期间。关于1T-80C51指令更详细的说明，请参考“32 指令集”，这里有每一条指令的助记符、字节数、时钟周期数。

6.3. CPU 寻址模式

直接寻址*(DIR)*

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

间接寻址*(IND)*

间接寻址时指令用一个包含操作数地址的寄存器表示，内部和外部存储器均可间接寻址。
8 位地址的地址寄存器可以是选中区的 R0 或 R1 或堆栈指针，16 位地址的地址寄存器只能是 16 位的“数据指针”寄存器，DPTR。

寄存器操作（寻址）*(REG)*

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

特殊寄存器寻址（寄存器间接寻址）

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

立即寻址*(IMM)*

常量的数值可以在程序存储器中跟随操作码。

索引寻址

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

7. 存储器组织

像所有的 80C51 一样，**MA82G5BXX** 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 **32K/16K/8K** 字节。在 **MA82G5BXX** 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。

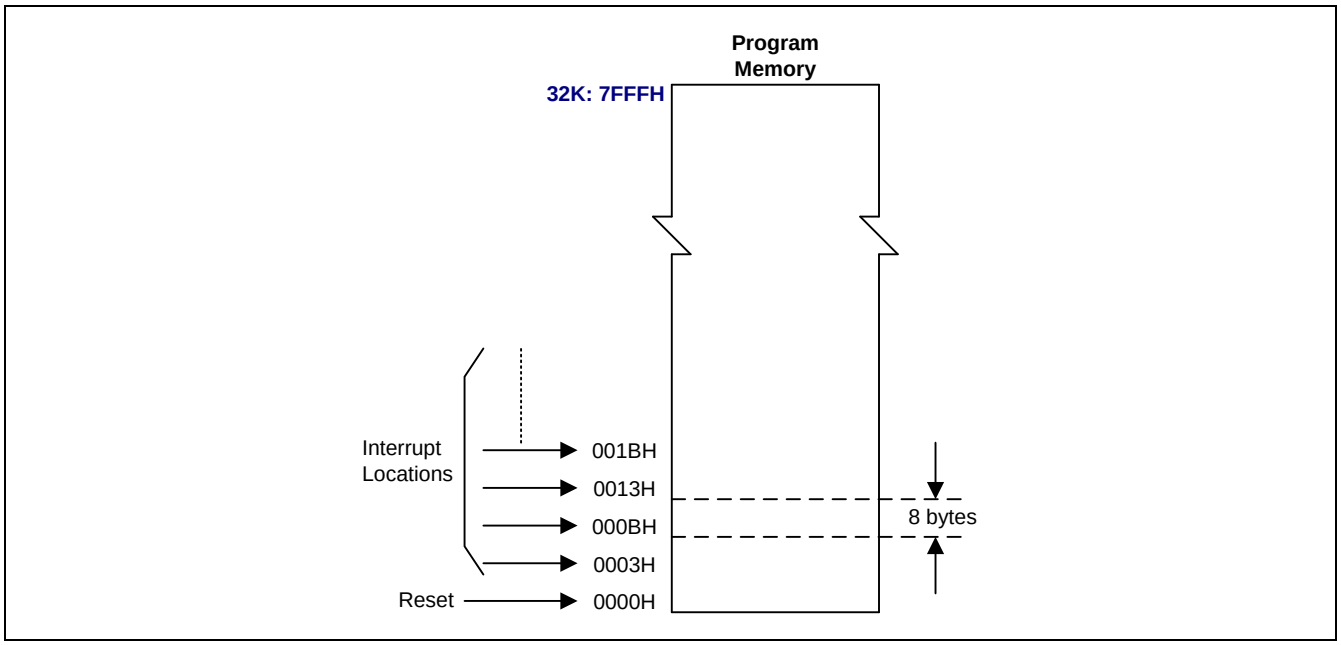
数据存储器使用与程序存储器不同的地址空间。**MA82G5BXX** 只有 256 字节的内部和 **1792/768/256** 字节的片上扩展存储器（XRAM）。

7.1. 片内程序存储器 Flash

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。如果其他的中断也被使用的话，较长的中断服务程序可以通过一条跳转指令越过后面的中断服务起始地址。

图 7–1. 程序存储器



7.2. 片内数据存储器 RAM

图 7–2 向 **MA82G5BXX** 使用者展示了内部和外部数据存储器的空间划分。内部数据存储器被划分为三部分，通常被称为低 128 字节 RAM，高 128 字节 RAM 和 128 字节 SFR 空间。内部数据存储器的地址线只有 8 位宽，因此地址空间只有 256 字节。SFR 空间的地址高于 7FH，用直接地址访问；而用间接访问的方法访问高 128 字节的 RAM。这样虽然 SFR 和高 128 字节 RAM 占用相同的地址空间（80H—FFH），但他们实际上是分开的。

如图 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个地址同时支持位寻址和直接寻址。可以位寻址的 SFR 的地址末位是0H或8H。

图 7–2. MA82G5B32 数据存储器

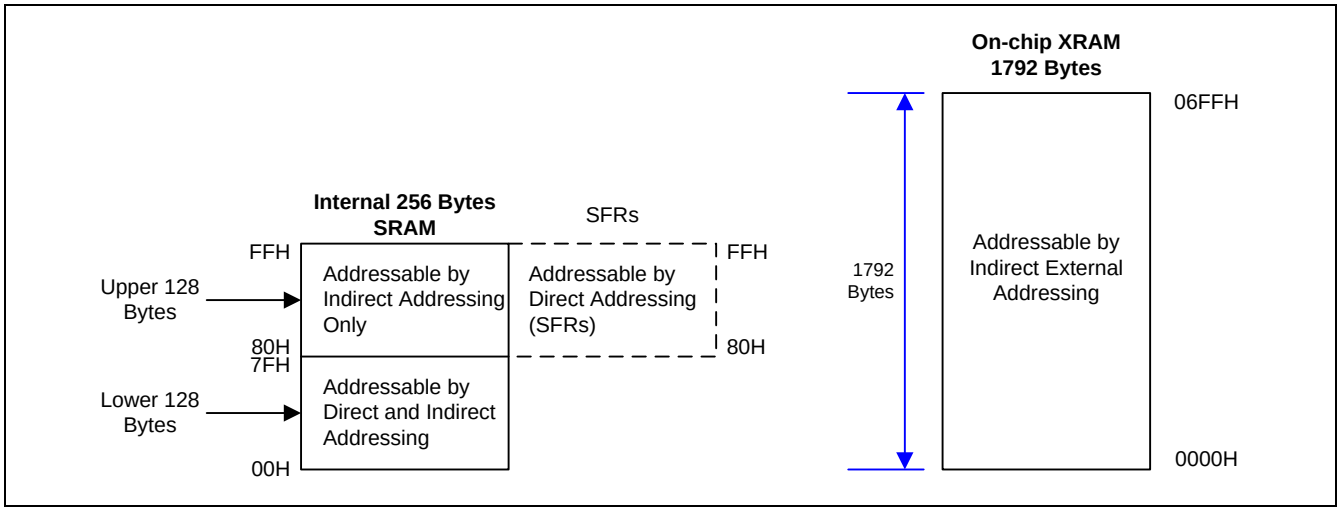


图 7–3. 内部 RAM 的低 128 字节

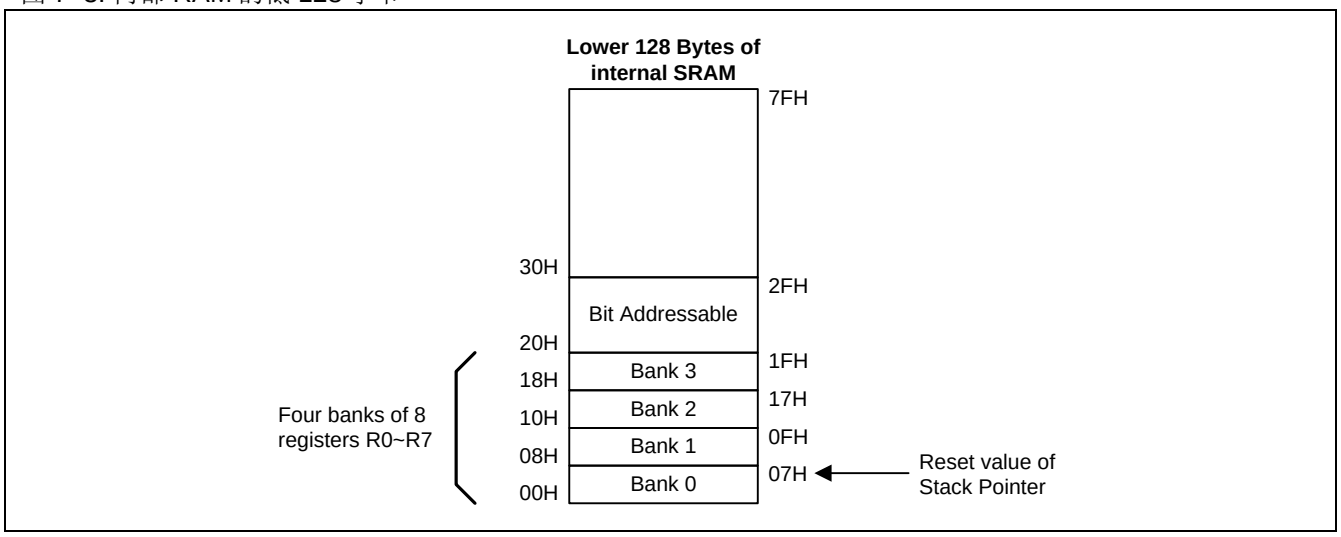
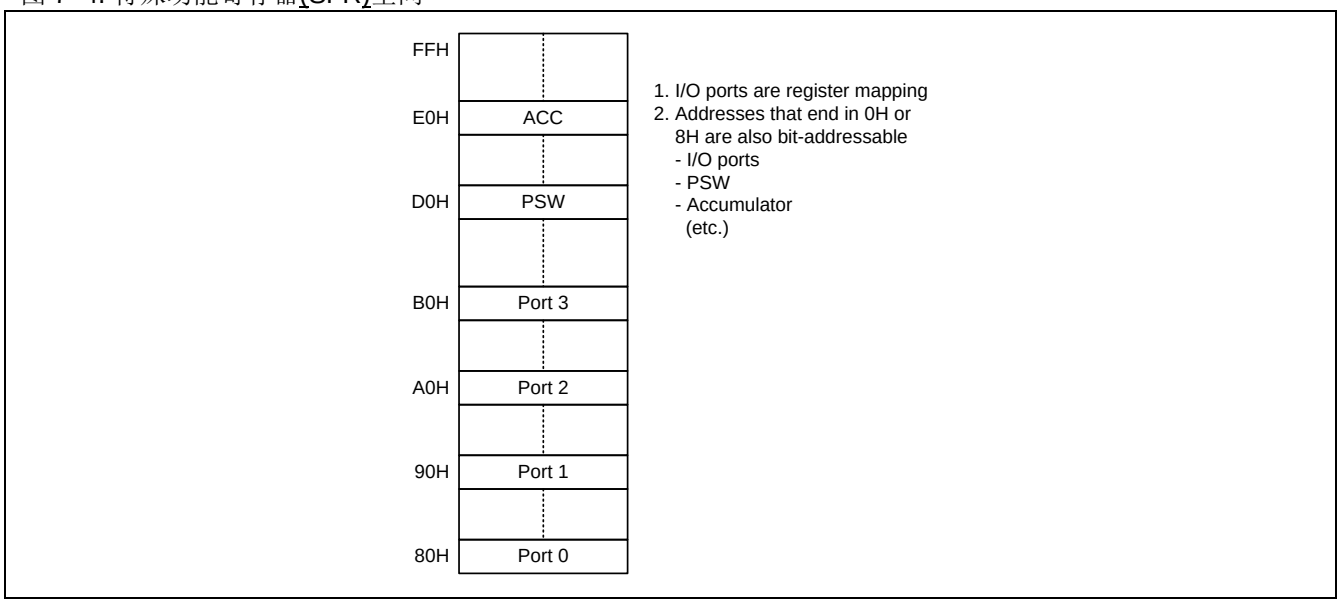


图 7–4. 特殊功能寄存器(SFR)空间



7.3. 片上扩展 RAM (XRAM)

访问片内扩展 RAM (XRAM)，参考图 7–2，这 **1792** 字节的 XRAM (0000H to **06FFH**) 可以被外部移动指令“MOVX @Ri” 和 “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 **MA82G5B32** 硬体才能正确访问 XRAM。

7.4. 关于 C51 编译器的声明标识符

C51 编译器的声明识别符与 **MA82G5BXX** 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。
MA82G5BXX 有 **1792/768/256** 字节的片上 xdata 存储空间。

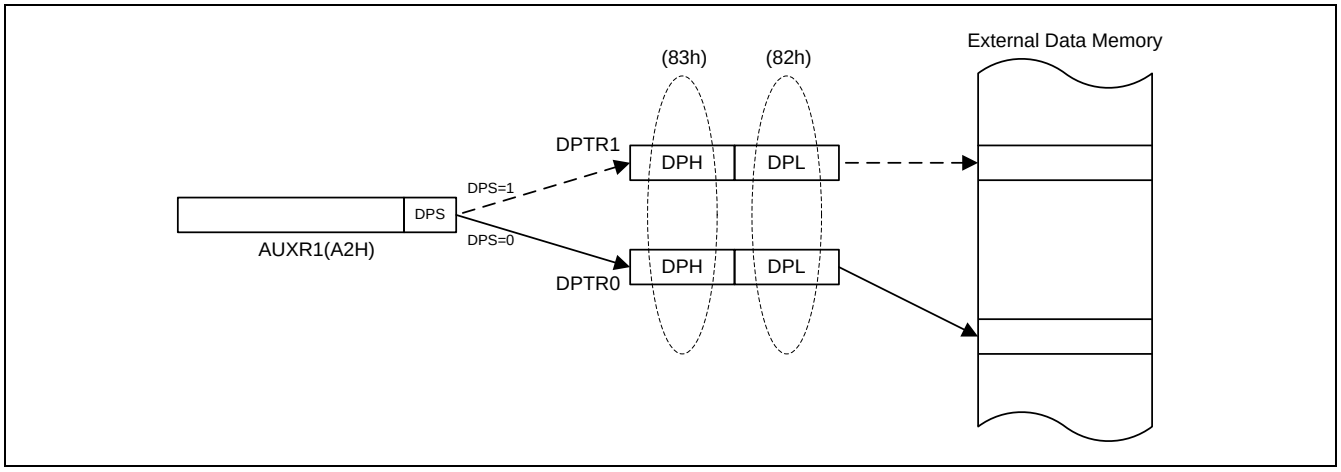
pdata
分页的外部数据(256 字节) 或片上的扩展 RAM（XRAM）：重叠的 256 字节的存储器地址通过“MOVX @Ri”指令访问。**MA82G5BXX** 有 256 字节片上 pdata 存储器它与片上 xdata 存储器共享。

code
32K/16K/8K 程序存储空间。通过“MOVC @A+DTPR”访问，作为程序部分被读取。**MA82G5BXX** 有 **32K/16K/8K** 字节的片上程序存储器。

8. 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
```

AUXR1: 辅助控制寄存器 1

SFR 页 = 0~F

SFR 地址 = 0xA2　　复位值 = 0000-0000

7	6	5	4	3	2	1	0
P1KBIH	P3KBIL	P4SPI	P3S1	P3S1MI	P6TWI	P3CEX	**DPS**
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**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	选择 DPTR
0	DPTR0
1	DPTR1

9. 系统时钟

系统时钟有 4 个时钟源：内部快频 RC 震荡器 (IHRCO)，外部晶振 ，内部慢频 RC 震荡器(ILRCO) 和外部频率输入。如图 9–1 所示 **MA82G5BXX** 系统时钟结构。

MA82G5BXX 默认值是 IHRCO **12MHz** 并保留晶振脚 P6.0/P6.1 普通 I/O 口的特性。软件可以根据应用要求自由切换 4 种时钟的任意一种作为系统时钟， 但必须等时钟稳定后才能切换。如果软件选择外部时钟模式， 脚 P6.0 和 P6.1 分配给 XTAL2 和 XTAL1，并且 P6.0/P6.1 普通 I/O 功能失效。 在外部时钟输入模式 (ECKI)， 时钟源来自 P6.0 ， P6.1 仍然是普通 I/O 口。

在设置 XTALE (CKCON2.5)使能外部晶振振荡之后，XTOR (CKCON1.7)将由硬件置位表明晶振振荡是稳定的软件可以切换到 OSCin 。XTOR 仅读。MCU 在切换晶振振荡器作为系统时钟源之前必须使能此位 XTALE (CKCON2.5)。

内建 IHRCO 提供两种软件可选的频率。其中频率 **11.059MHz** 通过软件置位 AFS(CKCON0.7)选择。IHRCO 的 12 MHz 以及 11.059 MHz 都是高精度的系统时钟源。详细的 IHRCO 性能请参考章节“31.4 IHRCO 特性”。 在 IHRCO 模式， P6.0 可以作为内部 *MCK* 或 2 分频时钟（*MCK/2*）输出或 4 分频时钟（*MCK/4*）输出给其他系统时钟源应用。

内置 ILRCO 提供的低功耗和低转速频率约 32 KHz 对 WDT 和系统时钟源。MCU 可以选择对系统时钟源 ILRCO 的低功率运行的软件。若要查找详细的 IHRCO 性能，请参阅章节“31.5 ILRCO 特性”)。在 ILRCO 模式下，可以将 P6.0 配置为内部 MCK 输出或 MCK2 和 MCK4 为系统中的应用。

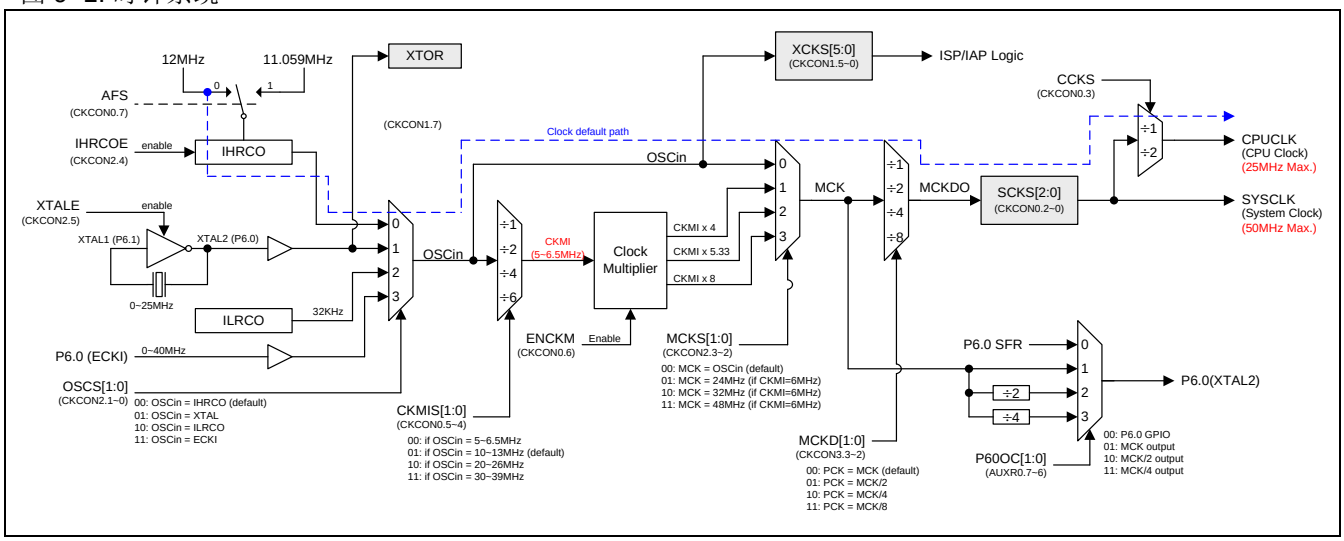
MA82G5BXX 包含一个时钟倍频器(CKM)来产生高速的系统时钟源。**MA82G5BXX** 时钟倍频器(CKM)的应用见图 9–1 并且典型输入频率是大约 6MHz 。在使能时钟倍频器(CKM)之前，软件必须配置 CKMIS1~0 (CKCON.5~4)获取合理的 CKMI 频率作为时钟倍频器(CKM)的输入源。CKM 能产生 4/5.33/8 倍的 CKMI 频率并且设置 MCKS1~0 (CKCON2.3~2)选择不同的 CKM 输出在 MCK 来提供给 MCU 高速运行而不需要高速的时钟源。详细的 CKM 性能请参阅章节“31.6 CKM 特性”。

时钟分配器分配 4 种时钟源的一种为系统时钟 *SYSCLK*， 如图 9–1.所示。 用户能通过设置 SCKS2~SCKS0 位 (CKCON0 寄存器) 来获得理性的系统时钟。**MA82G5BXX** 的最大系统时钟(SYSCLK)是 50MHz。软件必须配置 CPU 时钟。

9.1. 时钟结构

图 9–1 展示了 **MA82G5BXX** 的主要时钟系统。系统时钟来自于外部振荡电路或内部振荡器。

图 9–1. 时钟系统



9.2. 时钟寄存器

CKCON0: 时钟控制寄存器 0

SFR 页 = **0~F & P**

SFR 地址 = 0xC7 复位值 = 0001-0000

7	6	5	4	3	2	1	0
AFS	ENCKM	CKMIS1	CKMIS0	CCKS	SCKS2	SCKS1	SCKS0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: AFS, 交错频率选择
0: 选择 IHRCO 为 12MHz
1: 选择 IHRCO 为 11.059MHz.

Bit 6: ENCKM, 使能时钟倍频器(X8)
0: 禁止时钟倍频器(X8)
1: 使能时钟倍频器(X8)

Bit 5~4: CKMIS1 ~ CKMIS0, 时钟倍频器输入选择

CKMIS[1:0]	时钟倍频器输入选择
0 0	OSCin/1 (当 OSCin 为 5 ~ 6.5MHz)
0 1	OSCin/2 (当 OSCin 为 10 ~ 13MHz)
1 0	OSCin/4 (当 OSCin 为 20 ~ 26MHz)
1 1	OSCin/6 (当 OSCin 为 30 ~ 39MHz)

Bit 3: CCKS, CPU 时钟选择
0: 选择 CPU 时钟为 SYSCLK
1: 选择 CPU 时钟为 SYSCLK/2.

Bit 2~0: SCKS2 ~ SCKS0, 可编程系统时钟选择

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

CKCON1: 时钟控制寄存器 1

SFR 页 = **0~F & P**

SFR 地址 = 0xBF 复位值 = 0x00-1011

7	6	5	4	3	2	1	0
XTOR	--	XCKS5	XCKS4	XCKS3	XCKS2	XCKS1	XCKS0
R	**W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: XTOR, 晶振振荡准备好。仅读
0: 晶振振荡没有准备好
1: 晶振振荡准备好。当 XTALE 使能，XTOR 报告晶振振荡器达到开始计数的状态。

Bit 6: 保留位。 写 CKCON1 时，此位必须写“0”

Bit 5~0: 根据 OSCin 频率值设置写 ISP/IAP 时基，依照 OSCin 这 6 位写恰当值参考如下。

[XCKS5~XCKS0] = OSCin – 1, 当 *OSCin=1~40 (MHz).*

例如
(1) 若 OSCin=12MHz，那就[XCKS5~XCKS0]写 11，也就是 00-1011B。
(2) 若 OSCin=6MHz，那就[XCKS5~XCKS0]写 5 ，也就是 00-0101B。

OSCin	XCKS[4:0]
1MHz	00-0000
2MHz	00-0001
3MHz	00-0010
4MHz	00-0011
……	……
……	……
38MHz	10-0101
39MHz	10-0110
40MHz	10-0111

缺省值 XCKS= 00-1011 且 OSCin= 12MHz。

CKCON2: 时钟控制寄存器 2
SFR 页 = P
SFR 地址 = 0x40 复位值 = 0101-0000

7	6	5	4	3	2	1	0
XTGS1	XTGS0	XTALE	IHRCOE	MCKS1	MCKS0	OSCS1	OSCS0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~6: XTGS1~XTGS0, OSC 驱动控制寄存器

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

Bit 5: XTALE, 外部晶振 (XTAL)使能
0: 禁止 XTAL 振荡电路，在这种情况下， XTAL2 和 XTAL1 表现为 Port 6.0 和 Port 6.1。
1: 使能 XTAL 振荡电路，如果 CPU 软件设置这个位，软件检测到 **XTOR** (CKCON1.7)为"1"表明晶振振荡器准备好作为 OSCin 时钟选择。

Bit 4: IHRCOE, 内部高频 RC 震荡使能位
0: 禁止内部高频 RC 震荡电路
1: 使能内部快频 RC 震荡电路。如果软件设置这个位，在 IHRCOE 位使能后，必须等待 **32 us** IHRCOE 才能稳定输出

Bit 3~2: MCKS[1:0], MCK 时钟源选择

MCKS[1:0]	MCK 时钟源选择	OSCin =12MHz CKMIS = [01]	OSCin =11.059MHz CKMIS = [01]
0 0	OSCin	12MHz	11.059MHz
0 1	CKMI x 4 (ENCKM =1)	24MHz	22.118MHz
1 0	CKMI x 5.33 (ENCKM =1)	32MHz	29.491MHz
1 1	CKMI x 8 (ENCKM =1)	48MHz	44.236MHz

Bit 1~0: OSC[1:0], OSCin 时钟源选择

CKMIS[1:0]	OSCin 时钟源选择
0 0	IHRCO
0 1	XTAL
1 0	ILRCO
1 1	ECKI, 外部时钟输入 (P6.0) 作为 OSCin

CKCON3: 时钟控制寄存器 3

SFR 页 = P

SFR 地址 = 0x41 复位值 = 0000-0010

7	6	5	4	3	2	1	0
0	0	FWKP	0	MCKD1	MCKD0	MCDS1	MCDS0
W	**W**	**R/W**	**W**	**R/W**	**R/W**	**R/W**	**R/W**

AUXR0: 辅助寄存器 0

SFR 页 = 0~F

SFR 地址 = 0xA1 复位值 = 0000-0000

7	6	5	4	3	2	1	0
P60OC1	P60OC0	P60FD	T0XL	P4FS1	P4FS0	INT1H	INT0H
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~6: P60 输出配置控制位 1 和 0，这两位仅仅当内部振荡（IHRCO 或 ILRCO）被选择为系统时钟源时有效。这种情况，在晶振模式， XTAL2 和 XTAL1 改变功能作 P6.0 和 P6.1，在外部时钟输入模式，P6.0 专用于时钟输入口。在内部振荡条件下， P6.0 为普通 I/O 或时钟源发生器提供下列选项， 当 P60OC[1:0] 索引为非 P6.0 GPIO 功能时，P6.0 将驱动内部 RC 振荡器输出为其它设备提供时钟源。

P60OC[1:0]	P6.0 功能	I/O 模式
00	P60	By P6M0.0
01	MCK/1	By P6M0.0
10	MCK/2	By P6M0.0
11	MCK/4	By P6M0.0

P6.0 作为时钟输出功能时, 建议设置 P6M0.0 为“1” 来选着 P6.0 为推挽输出模式。

Bit 5: P60FD, P6.0 快速驱动

0: P6.0 默认驱动输出

1: P6.0 快速驱动输出使能。若 P6.0 被配置为时钟输出，当 P6.0 输出频率大于 12MHz （5V） 或者大于 6MHz （3V） 的应用时使能此位。

9.3. 系统时钟示例代码

(1)规定功能: IHRCO 从 12MHz 更改到 11.0592MHz

汇编语言代码范例:

```
ORL     CKCON0,#(AFS)              ; 选择 IHRCO 输出 11.0592MHz，AFS=1
```

C 语言代码范例:

```
CKCON0 |= AFS;                     //选择 IHRCO 输出 11.0592MHz，AFS=1
```

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

汇编语言代码范例:

```
ANL     CKCON0,#~( SCKS2 | SCKS1)          ; 设置 SCKS[2:0] = 1 来选择系统时钟(SYSCLK)为 OSCin/2
ORL     CKCON0,#( SCKS0)                   ;
```

C 语言代码范例:

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

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

汇编语言代码范例:

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

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

check_XTOR:                        ; 检测外部晶振(XTAL)振荡准备好
   MOV    A,CKCON1
   JNB    ACC.7,check_XTOR          ; 等待 XTOR(CKCON1.7)为 1

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

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

C 语言代码范例:

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

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

   while(CKCON1 & XTOR == 0x00);          //检测外部晶振(XTAL)振荡准备好
                                  //等待 XTOR(CKCON1.7)为 1

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

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

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

汇编语言代码范例:

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

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

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

C 语言代码范例:

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

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

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

(5). 规定功能: IHRCO 频率输出在 P6.0

汇编语言代码范例:

```
MOV    SFRPI,#01h                ; 设置 SFRPI=1
MOV    P6M0,#P6M00               ; 设置 P6.0 为推挽输出模式
MOV    SFRPI,#00h                ; 设置 SFRPI=0
ANL    AUXR0,#~(P60OC1|P60OC0)     ; P6.0 更改为通用输入输出口(GPIO)功能
ORL    AUXR0,#(P60OC0|P6FD)        ; P6.0 = IHRCO 频率 + 引脚快速驱动
                                   ; P60OC[1:0] |  P6.0
                                   ; 00         |  GPIO
                                   ; 01         |  IHRCO/1
                                   ; 10         |  IHRCO/2
                                   ; 11         |  IHRCO/4
```

C 语言代码范例:

```
SFRPI = 0x01;                      //设置 SFR page =1
P6M0 |= P6M00;                      //设置 P6.0 为推挽输出模式
SFRPI = 0x00;                      //设置 SFR page =0
AUXR0 &= ~(P60OC0 | P60OC1);       // P6.0 更改为通用输入输出口(GPIO)功能
AUXR0 |= (P60OC0 | P6FD);           // P6.0 输出 IHRCO/1
//   AUXR0 = P60OC1|P6FD;          // P6.0 输出 IHROC/2
//   AUXR0 = P60OC1|P60OC0|P6FD;       // P6.0 输出 IHRCO/4
```

10. 看门狗定时器 (WDT)

10.1. WDT 结构

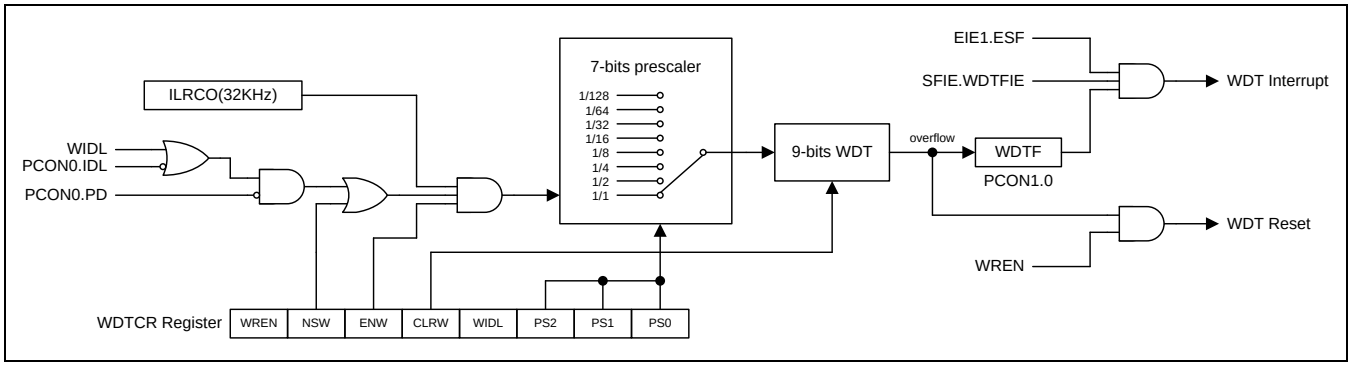
看门狗定时器 (WDT) 用来使程序从跑飞或死机状态恢复的一个手段。WDT 由一个 9 位独立定时器、一个 7 分频器和一个控制寄存器(WDTCR)组成。图 10–1 显示 **MA82G5BXX** WDT 结构框图。

当 WDT 使能，时钟源来自 32KHz ILRCO。WDT 溢出会设置位 WDTF PCON1.0， 也能产生中断通过使能位 WDTFIE (SFIE.0) 和 ESF (EIE1.3)， 溢出也能触发系统复位通过设置位 WREN (WDTCR.7) 。软件可以在溢出之前在 CLRW 位 (WDTCR.4)上写“1” 来清除它，可以阻止 WDT 溢出。

一旦 WDT 使能通过设置位 ENW， 将没有办法使之失效除非上电复位或在 page-p SFR 覆盖 ENW，能清除位 ENW。 WDTCR 会保持以前的值不会改变在硬件(RST-pin)复位、软件复位和 WDT 复位后。

WREN, NSW 和 ENW 都是一次性使能生效，写“1”使能。在 Page-P 中写“0”到位 WDTCR.7~5 禁止 WREN, NSW 和 ENW 使用。祥见章节“10.3 WDT 寄存器”和章节“27 P 页 SFR 访问”。

图 10–1.看门狗定时器



10.2. WDT 在掉电模式和空闲模式期间

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

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

10.3. WDT 寄存器

WDTCR: 看门狗控制寄存器

SFR 页 = 0~F & P

SFR 地址 = 0xE1 POR = XXX0-XXXX (0000-0111)

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

Bit 7: WREN, WDT 复位使能，初始值随硬件选项 WRENO

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

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

Bit 6: NSW. 不停止 WDT。 初始值随硬件选项 NSWDT

0: WDT 在掉电模式停止计数 MCU

1: WDT 在 MCU 进入掉电模式(Watch Mode)或空闲模式下总是保持计数。 一旦 NSW 已经设置， 不能用软件在 0~F 页中清除， **但在 P 页中, 软件能修改其值 “0” 或 “1”。**

Bit 5: ENW. 使能 WDT

0: 禁止 WDT 运行。此位仅仅能被 POR 清除

1: 使能 WDT 。 一旦 ENW 位被设置，不能用软件在 0~F 页中清除， **但在 P 页中, 软件能修改其值 “0” 或 “1”。**

Bit 4: CLRW. WDT 清零位

0: 写 “0”到此位 WDT 没有任何操作

1: 写 “1”到此位会清除 9 位 WDT 计数器到 000H。注意此位不需要清除请写“0”

Bit 3: WIDL. WDT 空闲模式控制位

0: WDT 在 MCU 的空闲模式下停止计数

1: WDT 在 MCU 的空闲模式下保持计数

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	WDT 时间
0 0 0	1	15 ms
0 0 1	2	31 ms
0 1 0	4	62 ms
0 1 1	8	124 ms
1 0 0	16	248 ms
1 0 1	32	496 ms
1 1 0	64	992 ms
1 1 1	128	1.984 S

PCON1: 电源控制寄存器 1

SFR 页 = 0~F & P

SFR 地址 = 0x97 POR = 0010-x000

7	6	5	4	3	2	1	0
SWRF	EXRF	MCDF	RTCF	--	BOF1	BOF0	**WDTF**
R/W	**R/W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**	**R/W**

Bit 1: WDTF, WDT 溢出标志

0: 必须由软件写**“1”** 清除， 软件写“0” 不操作

1: 当 WDT 溢出时硬件置位此位，写 **“1”** 清除

10.4. WDT 硬件选项

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

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

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

WRENO:
☑:使能: 置位 WDTCR.WREN 以使能 WDTF 系统复位功能
☐:禁止: 清除 WDTCR.WREN 以禁止 WDTF 系统复位功能

NSWDT: 不停止的 WDT
☑:使能：使能 WDT 在掉电模式(watch 模式) 下保持运行
☐:禁止: 禁止 WDT 在掉电模式 watch 模式)下运行

HWENW: 硬件载入 WDTCR 的“ENW”
☑:使能： 上电时自动硬件使能看门狗定时器，并且自动加载 WRENO, NSWDT, HWWIDL 和 HWPS2~0 的值到 WDTCR 中
☐:禁止： 上电时看门狗定时器（WDT）不自动使能

HWWIDL, HWPS2, HWPS1, HWPS0:
当 HWENW 被使能，上电复位时，这四个保险丝位将被载入到特殊功能寄存器 WDTCR 中。

WDSFWP:
☑:使能： 特殊功能寄存器 WDTCR 中的 WREN, NSW, WIDL, PS2, PS1 和 PS0 软件写保护
☐:禁止： 特殊功能寄存器 WDTCR 中的 WREN, NSW, WIDL, PS2, PS1 和 PS0 可被软件改写

10.5. WDT 示例代码

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

汇编语言代码范例:

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

C 语言代码范例:

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

(2) 规定功能: 如何禁止 WDT

汇编语言代码范例:

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

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

C 语言代码范例:

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

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

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

汇编语言代码范例:

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

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

C 语言代码范例:

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

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

(4). 规定功能:使能 WDTCR 的写保护

汇编语言代码范例:

```
ORL     PCON1,#(WDTF)               ; 清除 WDTF 标志(写“1”)
```

```
MOV    WDTCR,#(ENW | CLRW | PS2)  ；使能 WDT 计数器并且设置 WDT 周期为 248 毫秒(ms)

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

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

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

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

C 语言代码范例:

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

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

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

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

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

11. 实时时钟(RTC)/系统时间

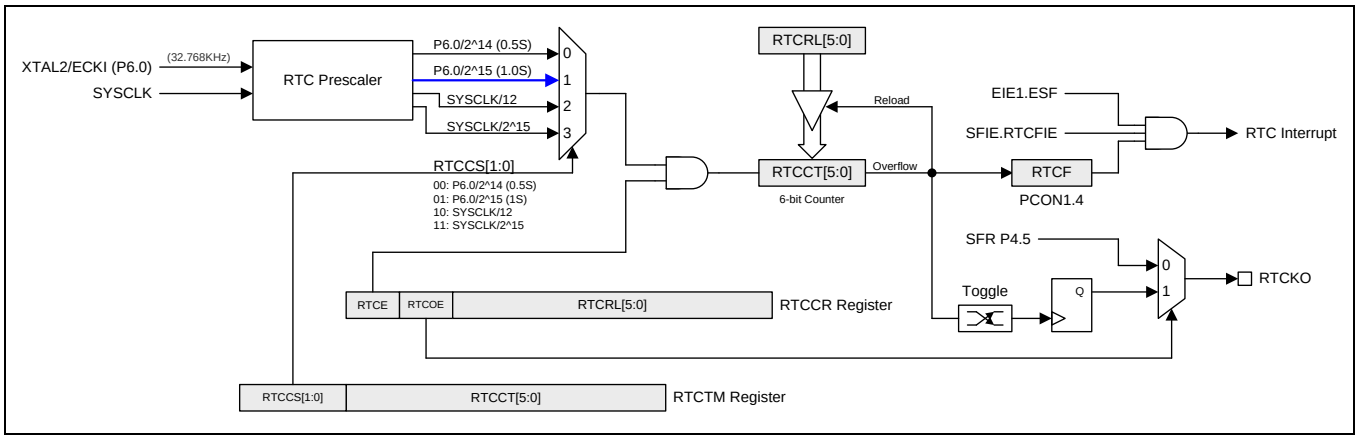
11.1. RTC 结构

MA82G5BXX 有一个简单的实时时钟允许使用者不停的记一个准确的时间在其它设备在掉电模式下。实时时钟能用于唤醒或中断源。实时时钟是一个 21 位的计数器包含 14/15 位的一个预分频器和一个 6 位的重载计数器。当其溢出，这个计数器会被重新加载并且 RTCF 旗标被设置。预分频器的时钟源来自内部系统时钟(SYSCLK)或者 XTAL 震荡器，条件是 XTAL 震荡器不可以作为系统时钟。图 11–1 显示 **MA82G5BXX** 的 RTC 结构。

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

如果 XTAL 震荡器被用于系统时钟，P6.0 仍然作为 RTC 时钟输入源。只有上电复位会重置 RTC 和它相应的特殊功能寄存器为默认值

图 11–1. 实时时钟计数器



11.2. RTC 寄存器

RTCCR: 实时时钟控制寄存器

SFR 页 = 普通和 P 页

SFR 地址 = 0xBE POR = 0011-1111

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

Bit 7: RTCE, RTC 使能

0: 停止 RTC 计数器，RTCCT

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

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

0: 禁止 RTCKO 输出

1: 使能 RTCKO 输出在 P4.5

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

RTCTM: 实时时钟定时器寄存器

SFR 页 = 普通

SFR 地址 = 0xB6 POR = 0111-1111

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

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

RTCCS[1:0]	时钟源	RTC 中断周期	最小周期
0 0	P6.0/2^14	0.5S ~ 32S 当 P6.0 = 32768Hz	0.5S
0 1	P6.0/2^15	1S ~ 64S 当 P6.0 = 32768Hz	1S
1 0	SYSCLK/12	1us ~ 64us 当 SYSCLK = 12MHz	1us
1 1	SYSCLK/2^15	2.73ms ~ 174.72ms 当 SYSCLK = 12MHz	2.73ms

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

PCON1: 电源控制寄存器 1

SFR 页 = 0~F & P

SFR 地址 = 0x97 POR = 0000-x000

7	6	5	4	3	2	1	0
SWRF	EXRF	MCDF	**RTCF**	--	BOF1	BOF0	WDTF
R/W	R/W	R/W	R/W	W	R/W	R/W	R/W

Bit 4: RTCF, RTC 溢出标志

0: 这位必须通过软件写**“1”**清除，软件写“0”不操作

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

SFIE: 系统旗标中断使能寄存器

SFR 页 = 0~F

SFR 地址 = 0x8E POR = 0110-x000

7	6	5	4	3	2	1	0
SIDFIE	MCDRE	MCDFIE	**RTCFIE**	--	BOF1IE	BOF0IE	WDTFIE
R/W	**R/W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**	**R/W**

Bit 4: RTCFIE, 使能 RTCF (PCON1.4) 中断

0: 禁止 RTCF 中断

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

11.3. RTC 实时时钟示例代码

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

汇编语言代码范例:

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

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

check_XTOR_0:                       ; 检测外部振荡(XTAL) 振荡准备好
   MOV    A,CKCON1
   JNB    ACC.7,check_XTOR_0        ; 等待 XTOR(CKCON1.7) 为 1
```

C 语言代码范例:

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

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

   while( CKCON1&XTOR == 0x00 );    //检测外部振荡(XTAL) 振荡准备好
                                    //等待 XTOR(CKCON1.7) 为 1
```

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

汇编语言代码范例:

```
   ORG    0005Bh
SystemFlag_ISR:
   ANL    PCON1,#(RTCF)             ; 清除 RTC 标志 (写“1”)
   RETI

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

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

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

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

C 语言代码范例:

```
void SystemFlag_ISR (void) interrupt 11
{
   PCON1 &= RTCF;                   //清除 RTC 标志 (写“1”)
}

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

   RTCTM = RTCCS1 | RTCCS0;         //选择 SYSCLK/2^15 作为 RTC 计数器时钟源
```

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

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

(3). 规定功能: 使能 RTCKO 输出 SYSCLK/12/2

汇编语言代码范例:

```
    ORL     P4M0,#20H                   ; 设置 RTCKO (P4.5)为推挽输出模式

    MOV     RTCTM,#0BFH                 ; RTC 时钟选择 SYSCLK/12 并且设置 RTCCT[5:0] = 3Fh

    MOV     RTCCR,#03FH                 ; 设置 RTCRL[5:0] = 3Fh

    ORL     RTCCR,#(RTCE|RTCOE)  ; 使能 RTC 计数器并且 RTCKO 输出
```

C 语言代码范例:

```
    P4M0 |= 0x20;                   //设置 RTCKO (P4.5)为推挽输出模式

    RTCTM = 0xBF;                   // RTC 时钟选择 SYSCLK/12 并且设置 RTCCT[5:0] = 3Fh

    RTCCR |= 0x3F;                  // 设置 RTCRL[5:0] = 3Fh

    RTCCR |= (RTCE | RTCOE);        //使能 RTC 计数器并且 RTCKO 输出
```

12. 系统复位

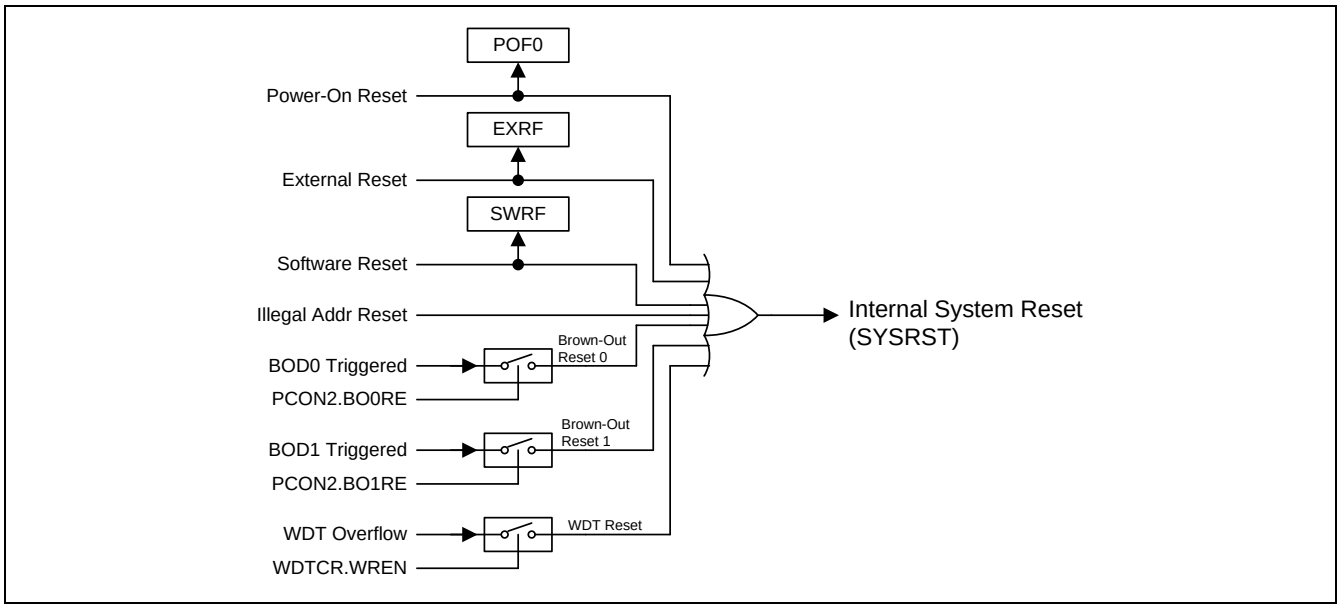
复位期间， 所有的 I/O 寄存器都设置为初始值，程序会根据 OR 设置选择从复位向量的 0000H 开始运行，或者从 ISP 地址开始运行。**MA82G5BXX** 有 7 种复位源： 上电复位，外部复位，软件复位，非法地址复位，掉电检测 0 复位，掉电检测 1 复位和 WDT 复位。 如图 12–1 所示系统复位源(**MA82G5BXX**)。

下面的选项描述复位产生源及其相应的控制寄存器和指示标志。

12.1. 复位源

图 12–1 显示 **MA82G5BXX** 复位系统和所有的复位源

图 12–1. 系统复位源



12.2. 上电复位

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

PCON0: 电源控制寄存器 0

SFR 页 = 0~F & P

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

7	6	5	4	3	2	1	0
SMOD1	SMOD0	GF	**POF0**	GF1	GF0	PD	IDL
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 4: POF0, 上电复位标志 0
0: 这标志必须通过软件清零以便认出下一个复位类型
1: 当VDD从0 伏上升到正常电压时硬件置位，POF 也能有软件置位

上电标志 POF0 在上电过程中由硬件置“1” 或当 VDD 电压降到 V_{POR} 电压之下时由硬件置“1”。它能通过软件来清除但不受任何热复位（譬如：外部 RST 引脚复位、掉电检测器 Brown-Out 复位、软件(ISPCR.5)复位和 WDT 复位）的影响。它帮助用户检测 CPU 是否从上电开始运行。注意：POF0 必须由软件清除。

12.3. 外部复位

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

PCON1: 电源控制寄存器 1

SFR 页 = 0~F 及 P 页

SFR 地址 = 0x97 POR = 0000-X000

7	6	5	4	3	2	1	0
SWRF	**EXRF**	MCDF	RTCF	--	BOF1	BOF0	WDTF
R/W	**R/W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**	**R/W**

Bit 6: EXRF, 外部复位标志
0: 这位必须通过软件清零，写**"1"** 清零，写 "0" 无效
1: 若外部复位产生则被硬件置位，写**"1"** 清零

12.4. 软件复位

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

ISPCR: ISP 控制寄存器

SFR 页 = 0~F

SFR 地址 = 0xE7 复位值 = 0000-X000

7	6	5	4	3	2	1	0
ISPEN	**SWBS**	**SWRST**	CFAIL	--	DATM2	DATM1	DATM0
R/W	**R/W**	**R/W**	**R/W**	**W**	**W**	**W**	**W**

Bit 6: SWBS, 软件执行起始选择控制
0: 复位软件从 AP 存储区开始执行
1: 复位软件从 ISP 存储区开始执行

Bit 5: SWRST, 软件复位触发控制
0: 写"0"无操作
1: 写"1" 产生软件系统复位，它将被硬件自动清除

PCON1: 电源控制寄存器 1

SFR 页 = 0~F 及 P 页

SFR 地址 = 0x97 POR = 0000-X000

7	6	5	4	3	2	1	0
SWRF	EXRF	MCDF	RTCF	--	BOF1	BOF0	WDTF
R/W	**R/W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**	**R/W**

Bit 7: SWRF, 软件复位标志
0: 这位必须通过软件清零 ， 写"1" 清零，写 "0"无操作
1: 软件复位产生时硬件置位此位，写"1" 清零

12.5. 掉电检测(Brown-Out)复位

MA82G5BXX 中，有两个掉电检测器(BOD0& BOD1)检测电源电压（VDD），掉电检测器(BOD0) 的检测固定点为 VDD=1.7V，掉电检测器(BOD1)的检测固定点可以被软件选择为 VDD=4.2V, 3.7V, 2.4V 或 2.0V，如果 VDD 电压低于 BOD0 或 BOD1 检测点， 则置位相关联的 BOF0 和 BOF1 标志，如果 BO0RE (PCON2.1) 被使能， BOD0 事件将触发一个 CPU 复位并置位 BOF0 指示一个掉点检测器（BOD0）复位发生；如果 BO1RE (PCON2.3) 被使能， BOD1 事件将触发一个 CPU 复位并置位 BOF1 指示一个掉点检测器（BOD1）复位发生。

PCON1: 电源控制寄存器 1

SFR 页 = 0~F 及 P 页

SFR 地址 = 0x97 POR = 0000-X000

7	6	5	4	3	2	1	0
SWRF	EXRF	MCDF	RTCF	--	**BOF1**	**BOF0**	WDTF
R/W	**R/W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**	**R/W**

Bit 2: BOF1, BOF1 (复位) 标志

0: 这位必须通过软件清零 ， 写“1” 清零，写 “0”无操作

1: 当 VDD 电压碰到 BOD1 检测点时，硬件置位此位，写“1” 清零。如果 BO1RE (PCON2.3) 被使能，BOD1 事件将触发一个 CPU 复位并置位 BOF1 指示一个掉点检测器（BOD1）复位发生

Bit 1: BOF0, BOF0(复位) 标志

0: 这位必须通过软件清零 ， 写“1” 清零，写 “0”无操作

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

12.6. WDT 复位

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

PCON1: 电源控制寄存器 1

SFR 页 = 0~F 及 P 页

SFR 地址 = 0x97 POR = 0010-X000

7	6	5	4	3	2	1	0
SWRF	EXRF	MCDF	RTCF	--	BOF1	BOF0	**WDTF**
R/W	**R/W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**	**R/W**

Bit 0: WDTF, WDT 溢出/复位 标志

0: 这位必须通过软件清零 ， 写“1”清零，写“0”无操作

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

12.7. 非法地址复位

MA82G5BXX 如果软件程序运行到非法的地址比如超出程序空间(ROM)范围，将触发 CPU 复位

12.8. 复位示例代码

(1) 规定功能: 触发一个软件复位

汇编语言代码范例:

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

C 语言代码范例:

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

(2). 规定功能: 使能 BOD0 复位

汇编语言代码范例:

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

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

C 语言代码范例:

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

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

13. 电源管理

MA82G5BXX 支持两个电源监测模块（掉电侦察器(BOD0 和 BOD1)模块），和 6 种电源节能模式：空闲模式（IDLE）、掉电模式（Power-Down）、慢频模式、 副频模式、Watch 模式和 Monitor 模式。

BOD0 和 BOD1 通过 BOF0 和 BOF1 标志位报告电源状态，软件可以通过这个状态产生中断或复位。6 种电源节能模式提供不同的节能应用， 通过对 CKCON0, CKCON2, PCON0, PCON1, PCON2, PCON3 和 WDTCR 寄存器的访问来操作这些电源事件。

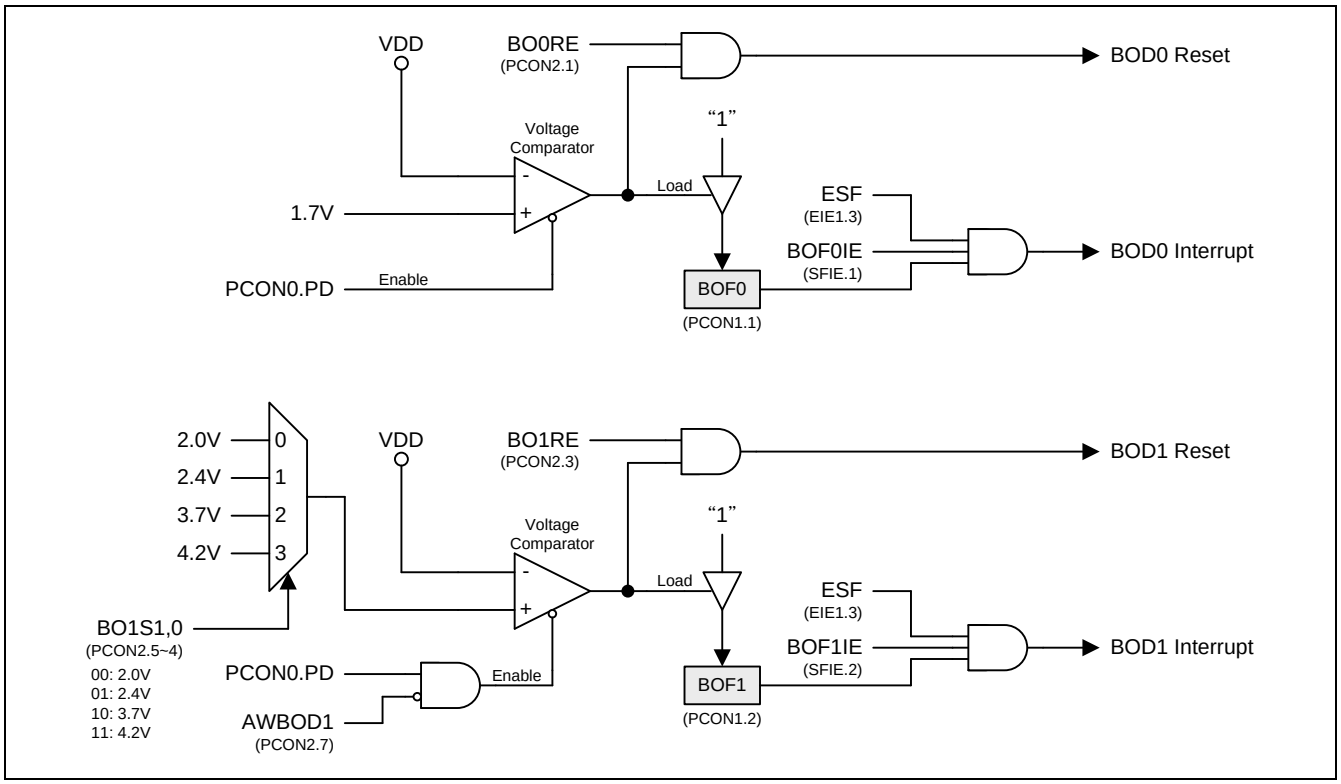
13.1. 电源监控模块

MA82G5BXX 有两个片上检测器 (BOD0& BOD1)通过比较固定的触发电压来检测芯片电压，图 13–1 是 BOD0 和 BOD1 功能逻辑图， BOD0 检测固定触发电压为 VDD=1.7V 和 BOD1 检测固定触发电压为 VDD=(4.2V/3.7V/2.4V/2.0V) 。 当 VDD 降到触发电压以下时， BOF0 (PCON1.1)标志被置位， 如果 ESF (EIE1.3) 和 BOF0IE (SFIE.1) 被使能， 不管是普通模式或空闲模式都能产生一个中断请求以响应(BOD0)事件，BOD1 有同样的标志 BOF1，也有同样的中断功能， 如果 AWBOD1 (PCON2.7)使能，这个中断（BOD1）也能唤醒掉电模式。

当 BO0RE (PCON2.1) 被使能， BOD0 事件产生一个系统复位并硬件置位 BOF0 指示一个 BOD0 复位事件已经产生。在普通模式和空闲模式下 BOD0 事件能重新启动 CPU，BOD1 也有同样的复位功能设置相关的控制位 BO1RE (PCON2.3)，如果 AWBOD1 (PCON2.7)位被使能，BOD1 也能重新启动掉电模式。

如果 BOD1 在应用中没有使用，为了节省功耗可以通过软件清除 EBOD1 (PCON2.2)来禁止 BOD1

图 13–1. 电源监控检测器 0/1



13.2. 电源节省模式

13.2.1. 慢频模式

程序设置位 SCKS2~SCKS0(CKCON0 寄存器, 参考章节“9 系统时钟”)为非 0/0/0 值，可以减慢 MCU 的工作速度达到节能的目的， 使用者考量在特殊的程序段使用合适的慢速度，原则上不应该影响系统的其他功能。而且，应该在普通的程序段恢复到正常的速度。

13.2.2. 副频模式

设置 OSCS1~0 选择 OSCS1~0 作为系统时钟，MCU 的工作频率会慢下来， 32KHz ILRCO 系统频率使 MCU 工作在特别慢的速度和功耗下， 另外设置 SCKS2~SCKS0 位 (CKCON0 寄存器, 参考章节“9 系统时钟”) 使用者可以使 MCU 的速度最低到 250HZ。

13.2.3. RTC 模式

MA82G5BXX 有一个简单的 RTC 模块允许用户在设备部分掉电时继续运行准确的定时器。在 RTC 模式，RTC 模块作为一个“时钟”功能并且能在 RTC 溢出时唤醒芯片的掉电模式。详细描述请参考章节“11 实时时钟(RTC)/系统时间”。

13.2.4. Watch 模式

如果看门狗被使能并且位 NSW 被设置， 看门狗在掉电模式保持运行，这个在 **MA82G5BXX** 应用中叫 Watch 模式。 当 WDT 溢出， 软件选择中断或系统复位来唤醒 CPU 并硬件置位 WDTF 。 通过定义 WDT 预分频最大唤醒时间能到 2 秒， 更详细信息请参考章节“10 看门狗定时器 (WDT)”和章节“15 中断”。

13.2.5. Monitor 模式

如果 AWBOD1 (PCON2.7) 被设置， BOD1 即使在掉电模式下，掉电检测功能 BOD1 会有效，这就是 **MA82G5BXX** 应用中的 Monitor 模式。 当 BOD1 触发到检测电压， 软件选择中断或系统复位来唤醒 CPU 并硬件置位 BOF1 ，更详细信息请参考章节“13.1 电源监控模块”和章节“15 中断”。

13.2.6. 空闲模式

可以通过软件的方式置 PCON.IDL 位，使设备进入空闲模式。 在空闲模式下，系统不会给 CPU 提供时钟 CPU 状态、RAM、SP、PC、PSW、ACC 被保护起来。I/O 端口也保持当前的逻辑状态。空闲模式保持外部设置当有中断来时能唤醒 CPU， 空闲模式下定时器 0、定时器 1、定时器 2、nINT0~nINT3、UART0、 UART1、SPI、TWI0、TWI1 、KBI、 ADC、SID 、RTC 、BOD0 和 BOD1 仍然处于工作状态。在空闲模式下 PCA 和 WDT 唤醒 CPU 有条件制约。 任何使能的中断源或复位都能终止空闲模式， 一个中断会退出空闲模式， 并同时进入中断服务程序，只有在中断返回后才会开始执行进入空闲模式指令之后的程序。

ADC 输入通道必须在 **P1AIO** SFR 设置为“ 仅模拟输入”当 MCU 在空闲模式和掉电模式。

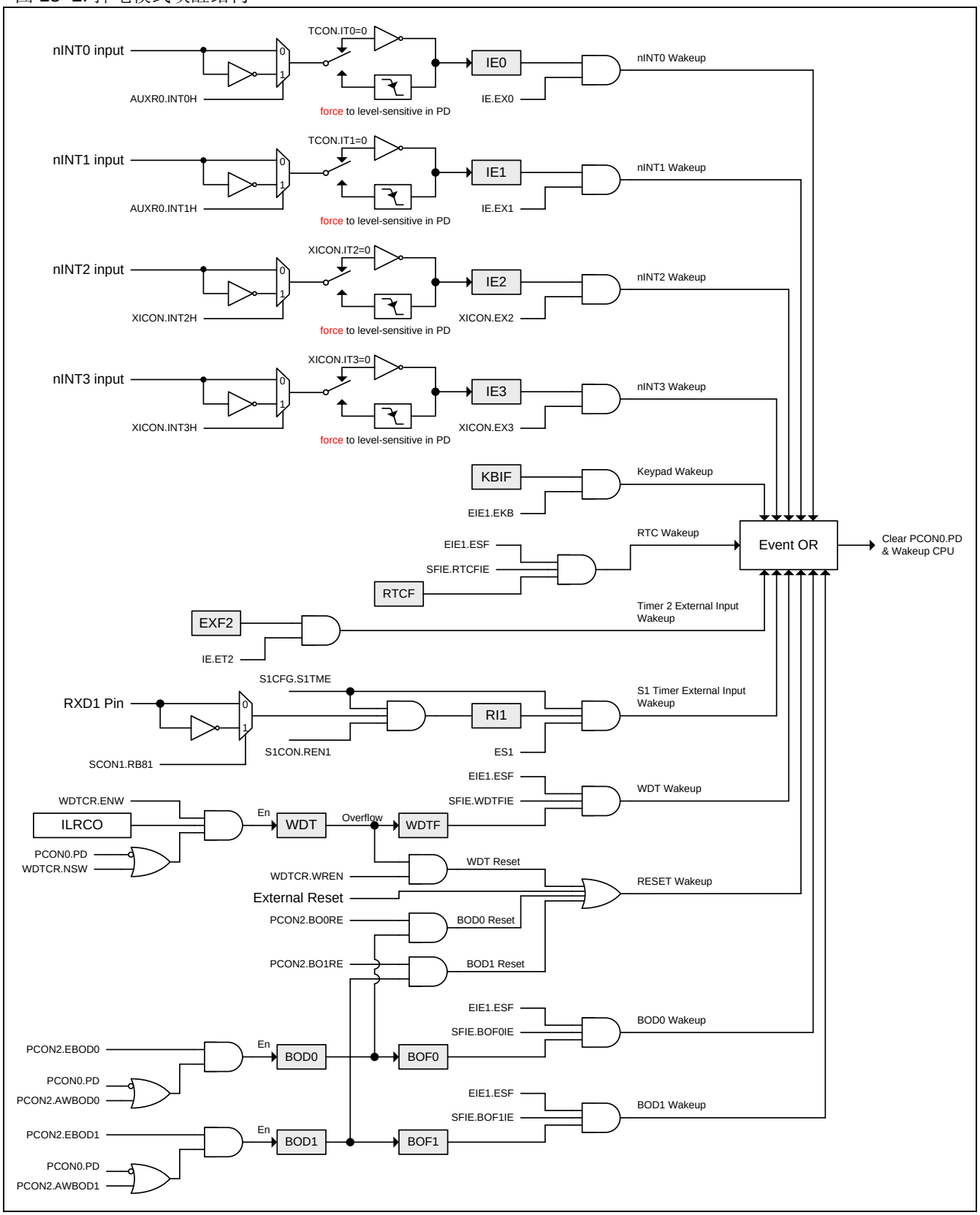
13.2.7. 掉电模式

可以通过软件的方法置位 PCON0. PD 使设备进入掉电模式， 掉电模式下，震荡器停止震荡， Flash 存储器掉电以节约电能， 只有上电电路继续刷新电源，在减少 VDD 的时候 RAM 的内容仍然会被保持；但如果电源电压低于芯片工作电压，特殊功能寄存器 SFR 的内容就不一定能保持住。外部复位、上电复位、使能的外部中断、使能的 KBI、使能的 RTC (RTC 模式) 、使能的 BOD1(monitor 模式)或使能的不停止的 WDT 能是系统退出掉电模式。

如果有下列情况发生，使用者至少要等 4 微秒后才能进入或再次进入掉电模式： 刚开始运行代码(任何形式的复位后面)，或者刚刚退出掉电模式。 为了在掉电模式达到最小功耗， 软件必须设置所有的 I/O 为非悬浮状态， 包含封装中没有露出来的 I/O。为了确保在掉电模式处于最小功耗, 软件必须设置所有的 I/O 为非悬浮状态, 包含封装中没有露出来的 I/O。

图 13–2 展示了 **MA82G5BXX** 掉电模式唤醒结构

图 13–2. 掉电模式唤醒结构



13.2.8. 中断唤醒掉电模式

四个外部中断都能终止掉电模式， 外部中断 nINT0、 nINT1、 nINT2 和 nINT3 能退出掉电模式。为了能唤醒掉电模式， 中断 nINT0, nINT1, nINT2 或 nINT3 必须使能并且设置为电平触发操作，如果外部中断使能且设置是边沿触发（上升或下降），他们会被硬件强置为电平触发（低电平或高电平）。

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

13.2.9. 复位唤醒掉电模式

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

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

13.2.10.KBI 键盘唤醒掉电模式

MA82G5BXX 中 P2.7 ~ P2.0 具有键盘中断唤醒功能，通过使能 KBI 模块的控制寄存器。软件可以设置 P1KBIH(AUXR1.7)交换 KBI 功能的高 4 位到端口 1(Port 1)和设置 P3KBIL (AUXR1.6)交换 KBI 功能的低 4 位到端口 3(Port 3)。更详细的 AUXR1 信息请参考章节“4 辅助特殊功能寄存器”。

通过使能 KBI 唤醒掉电模式有点类似中断唤醒， 在 KBI 模式下且已经使能 KBI 中断(EIE1.5, EKB)，系统退出掉电模式， 震荡重新启动， 且一个内部计数器开始计数， 在内部计数器没有计满之前内部时钟不允许被应用 CPU 也不能运行指令。计数溢出后，CPU 会响应 KBI 中断并执行中断服务程序。

13.3. 电源控制寄存器

PCON0: 电源控制寄存器 0

SFR 页 = 0~F 及 P 页

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

7	6	5	4	3	2	1	0
SMOD1	SMOD0	GF	POF0	GF1	GF0	PD	IDL
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 4: POF0, 上电标志 0
0: 这位必须由软件清零，写“0”清零
1: 当上电复位产生时硬件置位此位

Bit 1: PD, 掉电控制位
0: CPU 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: CPU 清零或任何一个退出掉电模式的事件发生时硬件清零
1: 置位则激活空闲操作（即进入空闲模式）

PCON1: 电源控制寄存器 1

SFR 页 = 0~F 及 P 页

SFR 地址 = 0x97 POR = 0010-X000

7	6	5	4	3	2	1	0
SWRF	EXRF	MCDF	RTCF	--	BOF1	BOF0	WDTF
R/W	**R/W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**	**R/W**

Bit 7: SWRF, 软件复位标志
0: 这位必须由软件清零，写“1”清零
1: 当软件复位产生时硬件置位此位

Bit 6: EXRF, 外部复位标志
0: 这位必须由软件清零，写“1”清零
1: 当外部复位产生时硬件置位此位

Bit 5: MCDF. (验证中)

Bit 4: RTCF, RTC 溢出标志
0: 这位必须由软件清零，写“1”清零。写“0”无效
1: 当 RTCCT 溢出时此位仅由硬件置位。写“1”则清除 RTCF

Bit 3: 保留位. 当写 PCON1 寄存器时软件必须在这位写 “0”

Bit 2: BOF1, 电源监控（Brown-Out）标志 1
0: 这位必须由软件清零，写“1”清零
1: 当电源电压触及到电源监测器 1 电压(4.2V/3.7/2.4/2.0)时，硬件置位此位

Bit 1: BOF0, 电源监控（Brown-Out）标志 0
0: 这位必须由软件清零，写“1”清零
1: 当电源电压触及到电源监测器 0 电压(1.7V)时，硬件置位此位

Bit 0: WDTF, WDT 溢出标志
0: 这位必须由软件清零，写“1”清零
1: 当 WDT 复位产生时硬件置位此位

PCON2: 电源控制寄存器 2

SFR 页 = **P**

SFR 地址 = 0x44 POR = 0011-0101

7	6	5	4	3	2	1	0
AWBOD1	**0**	BO1S1	BO1S0	BO1RE	EBOD1	BO0RE	**1**
R/W	**W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**W**

Bit 7: AWBOD1, 掉电模式(PD)下 BOD1 的唤醒
0: 掉电模式(PD)下禁止 BOD1
1: 掉电模式(PD)下保持 BOD1

Bit 6: 保留位. 当写 PCON2 寄存器时软件必须在这位写 “0”

Bit 5~4: BO1S[1:0]. 电源监测器 1 监测电压选择

BO1S[1:0]	BOD1 监测电压
0 0	2.0V
0 1	2.4V
1 0	3.7V
1 1	4.2V

Bit 3: BO1RE, BOD1 复位使能
0: 当 BOF1 已经设置，禁止电源监控（BOD1）系统复位
1: 当 BOF1 已经设置，使能电源监控（BOD1）系统复位

Bit 2: EBOD1, 使能 BOD1 监测 VDD 下降到 BO1S1~0 设置的固定值
0: 禁止 BOD1 监测电源电压降低芯片功耗
1: 使能 BOD1 监测电源电压 VDD

Bit 1: BO0RE, BOD0 复位使能
0: 当 BOF0 已经设置，禁止电源监控（BOD0）系统复位
1: 当 BOF0 已经设置，使能电源监控（BOD0）系统复位（VDD 触到 1.7V）

Bit 0: 保留位. 当写 PCON2 寄存器时软件必须在这位写 “1”

13.4. 电源管理示例代码

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

汇编语言代码范例:

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

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

C 语言代码范例:

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

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

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

汇编语言代码范例:

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

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

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

MOV     A,CKCON0                    ; 选择系统时钟为 OSCin/1
ANL     A,#~(SCKS2|SCKS1|SCKS0)
MOV     CKCON0,A
```

C 语言代码范例:

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

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

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

ACC = CKCON0;                       // 选择系统时钟为 OSCin/1
ACC &= ~(SCKS2 | SCKS1 | SCKS0);
CKCON0 = ACC;
```

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

汇编语言代码范例:

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

ANL     IFD,#~((XTGS1 | XTGS0)   ; 对 32768Hz 的外部晶振(XTAL), 设置 XTGS1 = XTGS0 = 0
ORL     IFD,#(XTALE)                ; 使能外部振荡(XTAL)振荡
CALL    _page_p_sfr_write           ; 写数据到 CKCON2
```

```
check_XTOR:                          ; 检测外部晶振(XTAL)振荡准备好
   MOV    A,CKCON1
   JNB    ACC.7,check_XTOR           ; 等待 XTOR(CKCON1.7)为 1

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

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

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

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

C 语言代码范例:

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

   IFD &= ~(XTGS1 | XTGS0);          //设置对 32768Hz 的外部晶振(XTAL), 设置 XTGS1 = XTGS0 = 0
   IFD |= XTALE;                     //使能外部振荡(XTAL)振荡
   page_p_sfr_write();               // 写数据到 CKCON2

   while(CKCON1 & XTOR == 0x00);          //检测外部晶振(XTAL)振荡准备好
                                          //等待 XTOR(CKCON1.7)为 1

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

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

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

(4). 规定功能: 使能 2 秒(s)周期的 Watch 模式

汇编语言代码范例:

```
   ORG    0005Bh
SystemFlag_ISR:
   ORL    PCON1,#(WDTF)              ; 清除 WDT 标志(写“1”)
   RETI

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

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

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

;  MCU 等待唤醒
```

C 语言代码范例:

```
void SystemFlag_ISR (void) interrupt 11
{
   PCON1 |= WDTF;                    //清除 WDT 标志(写“1”)
}

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

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

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

//   MCU 等待唤醒
}
```

(5). 规定功能: Monitor 模式

汇编语言代码范例:

```
   ORG    0005Bh
SystemFlag_ISR:
   ORL    PCON1,#(BOF1)              ; 清除 BOD1 标志(写“1”)
   RETI

main:
   MOV    IFADRL,#PCON2              ; 索引 P 页地址为 PCON2
   CALL   _page_p_sfr_read           ; 读取 PCON3 数据

   ORL    IFD,#AWBOD1                ; 在掉电模式使能 BOD1 工作
   CALL   _page_p_sfr_write          ; 写数据到 PCON2

   ORL    SFIE,#BOF1IE               ; 使能 BOF1 中断
   ORL    EIE1,#ESF                  ; 使能系统标志中断
   SETB   EA                         ; 使能全局中断

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

;  MCU 等待唤醒
```

C 语言代码范例:

```
void SystemFlag_ISR() interrupt 11
{
   PCON1 |= BOF1;                    // 清除 BOD1 标志(写“1”)
}

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

   IFD |= AWBOD1;                    //在掉电模式使能 BOD1 工作
   page_p_sfr_write();               // 写数据到 PCON2

   SFIE |= BOF1IE;                   //使能 BOF1 中断
   EIE1 |= ESF;                      //使能系统标志中断
   EA = 1;                           //使能全局中断

   PCON0 |= PD;                      //设置 MCU 为掉电模式
//   MCU 等待唤醒
}
```

14. 输入输出口配置

MA82G5BXX 有下列 I/O 端口：P1.0~P1.7, P2.0~P2.7, P3.0~P3.5, P4.0~P4.1, P4.4, P4.5, P4.7,P6.0 和 P6.1。**RST** 引脚可以设置为 **P4.7**。如果选择外部振荡做系统时钟输入，则 Port 6.0 和 Port 6.1 被配置为 XTAL2 和 XTAL1。准确的可用 I/O 引脚数量由封装类型决定。见表 14–1。

表 14–1. 可用引脚数量

封装类型	I/O 引脚	引脚数量
32-pin LQFP	P1.0~P1.7, P2.0~P2.7, P3.0~P3.5, P4.0, P4.1, P4.4, P4.5, P4.7(RST), P6.0 (ECKI/XTAL2), P6.1 (XTAL1)	29 or 28 (RST 被选择) 或 27 (RST & ECKI 被选择) 或 26 (RST & XTAL 被选择)
28-pin SOP	P1.0~P1.7, P2.2, P2.4, P2.6, P3.0~P3.1, P3.3~P3.5, P4.0, P4.1, P4.4, P4.5, P4.7(RST), P6.0 (ECKI/XTAL2), P6.1 (XTAL1)	25 or 24 (RST 被选择) 或 23 (RST & ECKI 被选择) 或 22 (RST & XTAL 被选择)
20-pin SOP	P1.0~P1.1, P1.5~P1.7, P2.2, P2.4, P3.0~P3.1, P3.3~P3.5, P4.4, P4.5, P4.7(RST), P6.0 (ECKI/XTAL2), P6.1 (XTAL1)	17 or 16 (RST 被选择) 或 15 (RST & ECKI 被选择) 或 14 (RST & XTAL 被选择)

14.1. 输入输出口结构

MA82G5BXX 输入输出口分成两个配置类型。第一类仅仅是端口 3 有四种模式，这四种模式有：准双向口(标准 8051 的 I/O 端口)、推挽输出、集电极开漏输出和输入(高阻抗输入)。 缺省值是弱上拉的准双向口模式。

其它口属于第二类，这些口有两种模式分别是推挽输出和上拉电阻的集电极开漏输出。缺省是集电极开漏输出高，也就意味着带有高阻状态的输入模式。

下面描述这四种类型的 I/O 模式的配置。

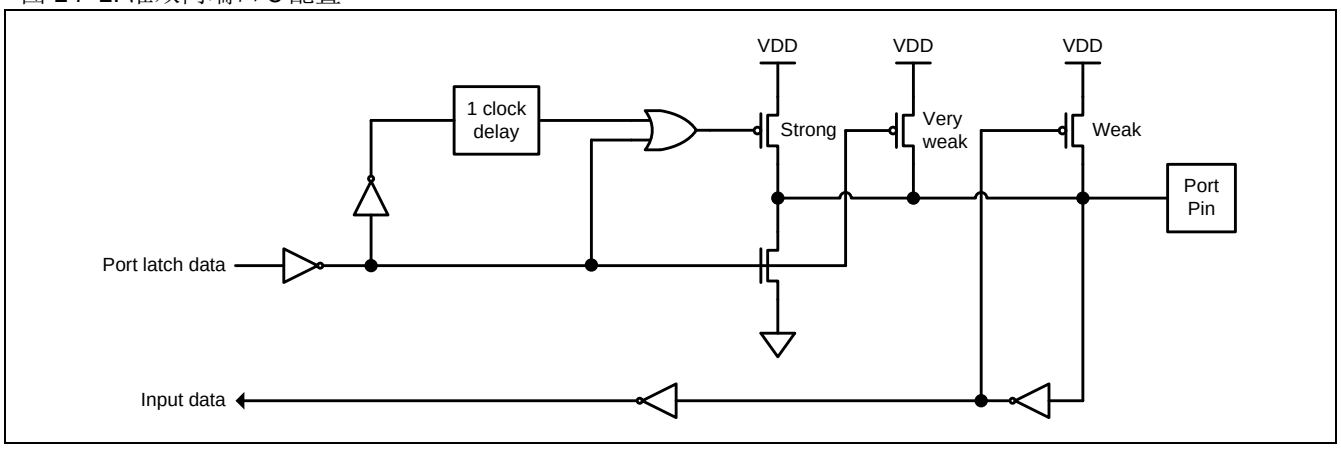
14.1.1. 端口 3 准双向口

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

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

准双向端口 3 配置如图 14–1 所示。

图 14–1. 准双向端口 3 配置

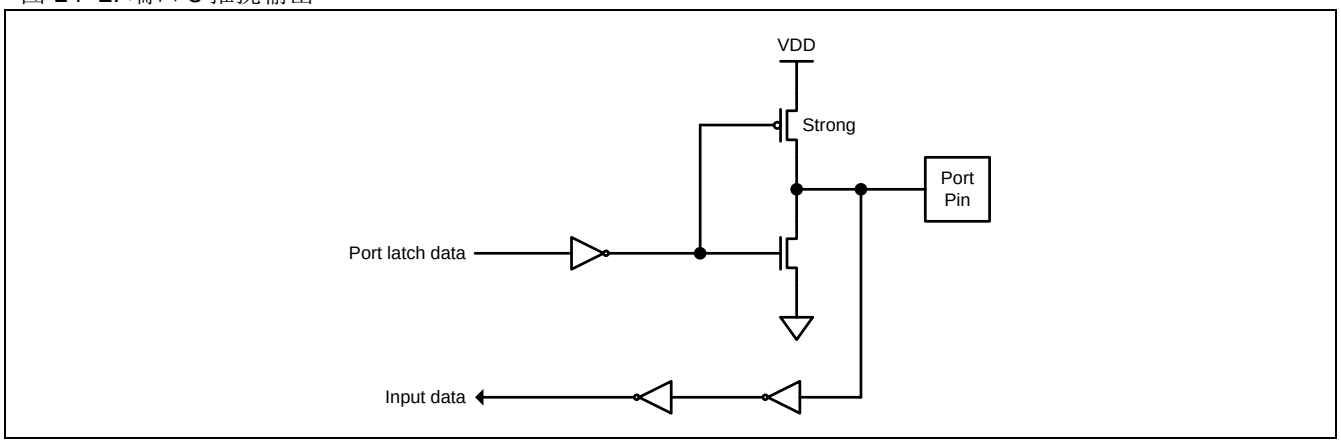


14.1.2. 端口 3 推挽输出

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

端口 3 推挽输出配置见图 14–2。

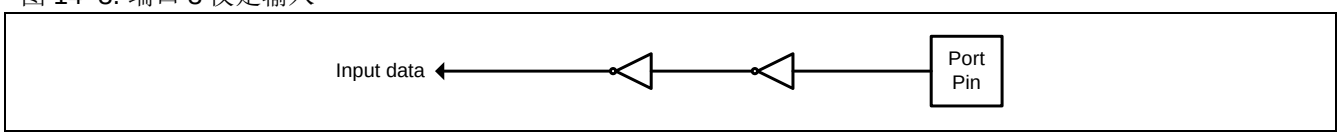
图 14–2. 端口 3 推挽输出



14.1.3. 端口 3 仅是输入（高阻抗输入）模式

仅输入配置在引脚上没有任何上拉电阻，如下图 14–3 所示。

图 14–3. 端口 3 仅是输入

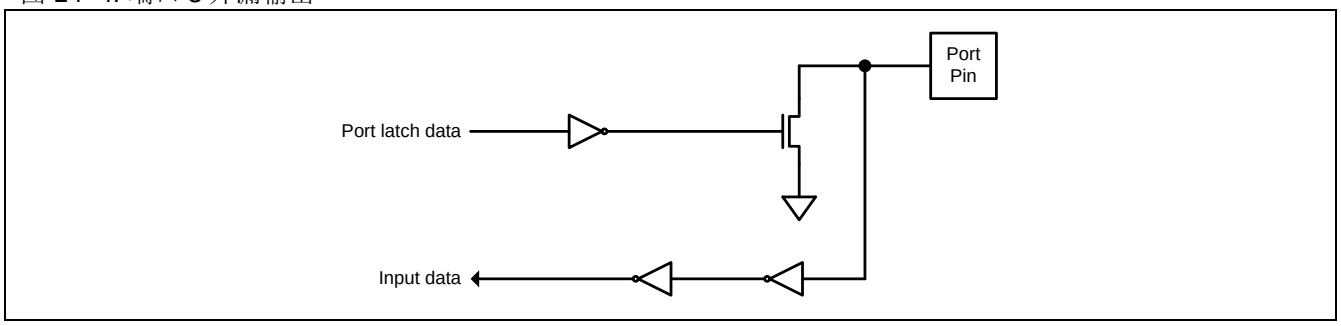


14.1.4. 端口 3 开漏输出

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

开漏输出端口 3 配置如图 14–4 所示。

图 14–4. 端口 3 开漏输出

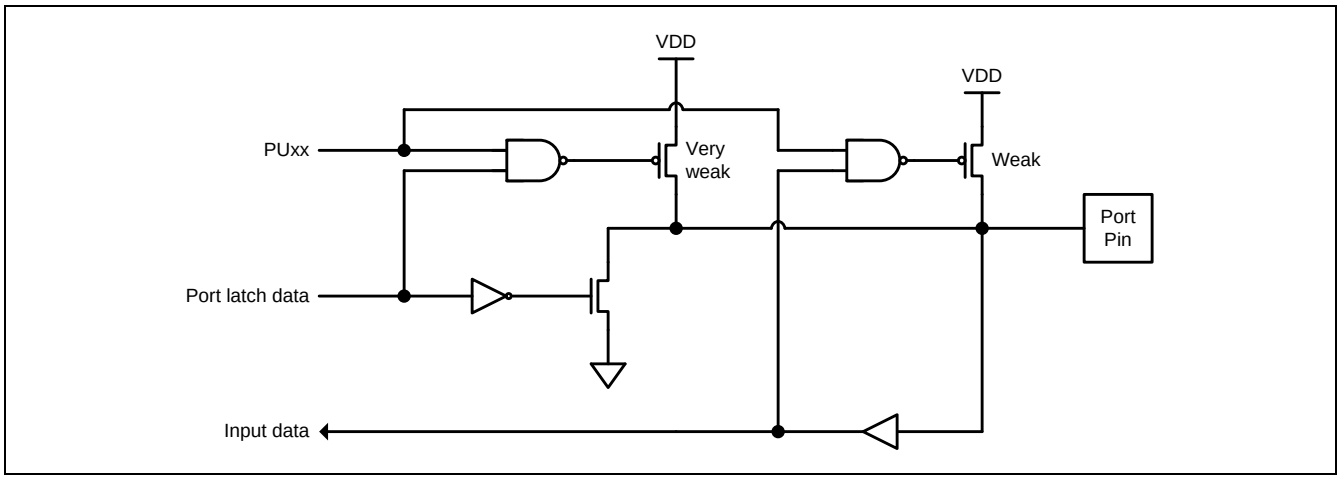


14.1.5. 通用端口集电极开漏输出结构

当端口数据寄存器写”0”时，通用端口的集电极开漏输出仅是驱动端口的下拉晶体管。在应用中使用这个配置，端口引脚可以选择外部上拉或 PUCON0 和 PUCON1 置位使能片内的内部上拉。

通用端口集电极开漏结构如图 14–5 所示。

图 14–5. 通用端口集电极开漏输出

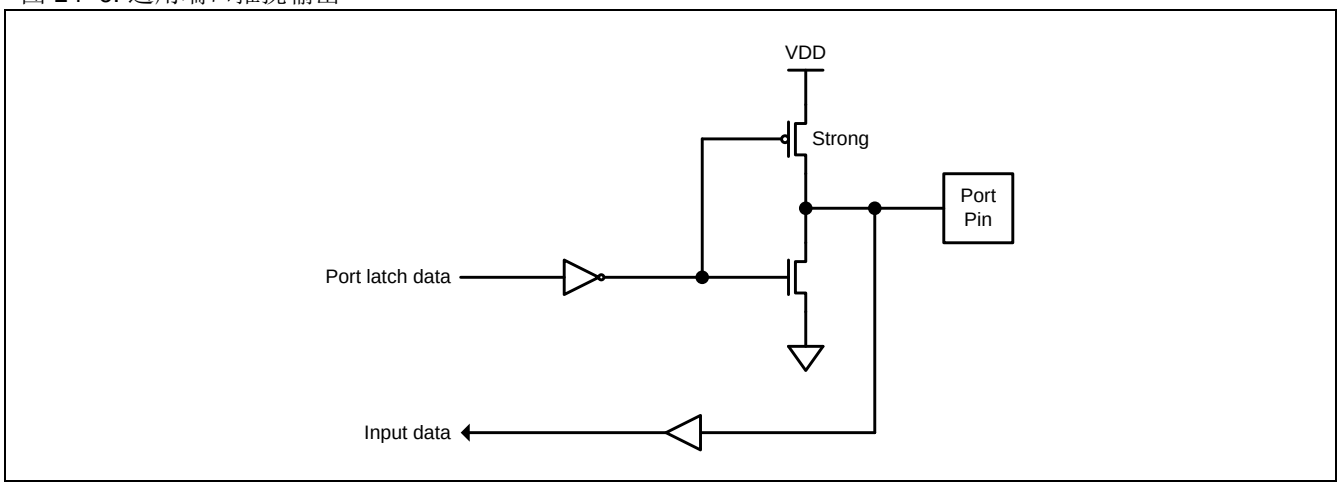


14.1.6. 通用端口推挽输出结构

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

通用端口推挽输出结构如图 14–6 所示。

图 14–6. 通用端口推挽输出



14.1.7. 通用端口输入配置

一个端口通过设置为“开路输出”的输出模式，并写逻辑“1”到向对应的端口数据寄存器位，可以配置为数字输入. 例如, P1.1 通过设置 P1M0.1 为逻辑“0”且 P1.1 为逻辑“1”来配置为数字输入.

14.2. 输入输出口寄存器

MA82G5BXX 所有的端口可通过软件个别的、独立的配置其工作模式，仅仅端口 3 有 4 种工作模式，如表 14–2。两个寄存器用于选择每个端口 3 引脚的输出类型。

表 14–2. 端口 3 配置设定

P3M0.y	P3M1.y	端口模式
0	0	准双向
0	1	推挽输出
1	0	输入口 (高阻抗输入)
1	1	集电极开漏输出

这里 y=0~5(端口引脚号)。寄存器 P3M0 和 P3M1 列举了每个引脚的描述。

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

表 14–3. 通用端口配置设定

PxM0.y	端口模式
0	集电极开漏输出
1	推挽输出

这里 x= **1, 2, 4, 6** (端口)，y=0~7(端口引脚号)。寄存器 PxM0 列举了每个引脚的描述。

14.2.1. 端口 1 寄存器

P1: 端口 1 寄存器

SFR 页 = 0~F

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 CPU 置位/清零

P1M0: 端口 1 模式寄存器 0

SFR 页 = 0~F

SFR 地址 = 0x91 复位值 = 0000-0000

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

0: 端口定义为集电极开漏输出
1: 端口定义为推挽输出

P1AIO: 端口 1 仅模拟输入

SFR 页 = 0~F

SFR 地址 = **0x92** 复位值 = 0000-0000

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

0: 端口有数字和模拟输入的能力
1: 只能是模拟输入给 ADC 输入应用，当这位被设置相应的端口总是读到“0”

14.2.2. 端口 2 寄存器

P2: 端口 2 寄存器

SFR 页 = 0~F

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 CPU 置位/清零

P2M0: 端口 2 模式寄存器 0

SFR 页 = 0~F

SFR 地址 = 0x95 复位值 = 0000-0000

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

0: 端口定义为集电极开漏输出
1: 端口定义为推挽输出

14.2.3. 端口 3 寄存器

P3: 端口 3 寄存器

SFR 页 = 0~F

SFR 地址 = 0xB0 复位值 = XX11-1111

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

Bit 7~6: 保留位。当 P3 被写时这些位软件必须写“1”。

Bit 5~0: P3.5~P3.0 通过 CPU 置位/清零

P3M0: 端口 3 模式寄存器 0

SFR 页 = 0~F

SFR 地址 = 0xB1 复位值 = XX00-0000

7	6	5	4	3	2	1	0
--	--	P3M0.5	P3M0.4	P3M0.3	P3M0.2	P3M0.1	P3M0.0
W	**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 = 0~F

SFR 地址 = 0xB2 复位值 = XX00-0000

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

14.2.4. 端口 4 寄存器

P4: 端口 4 寄存器

SFR 页 = 0~F

SFR 地址 = 0xE8 复位值 = 1X11-XX11

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

Bit 7, 5, 4, 1, 0: P4.7, P4.5, P4.4, P4.1 和 P4.0 通过 CPU 置位/清零

Bit 6, 3, 2: 保留位。当 P4 被写时这些位软件必须写“1”

P4M0: 端口 4 模式寄存器 0

SFR 页 = 0~F

SFR Address = 0xB3 复位值 = 0X00-XX00

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

Bit 7, 5, 4, 1, 0:
0: 端口定义为集电极开漏输出
1: 端口定义为推挽输出

Bit 6, 3, 2: 保留位。当 P4M0 被写时这些位软件必须写“0”

14.2.5. 端口 6 寄存器

P6: 端口 6 寄存器

SFR 页 = 0~F

SFR 地址 = 0xF8 复位值 = XXXX-XX11

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

Bit 7~2: 保留位。当 P6 被写时这些位软件必须写“1”

Bit 1~0: P6.1~P6.0 通过 CPU 置位/清零
P6.1 和 P6.0 复用晶体振荡电路功能，XTAL1 和 XTAL2

P6M0: 端口 6 模式寄存器 0

SFR 页 = **1**

SFR 地址 = **0xB5** 复位值 = XXXX-XX00

7	6	5	4	3	2	1	0
--	--	--	--	--	--	P6M0.1	P6M0.0
W	**W**	**W**	**W**	**W**	**W**	**R/W**	**R/W**

Bit 7~2: 保留位。当 P6M0 被写时这些位软件必须写“0”

Bit 1~0:
0: 端口定义为集电极开漏输出
1: 端口定义为推挽输出

14.2.6. 上拉控制寄存器

PUCON0: 端口上拉控制寄存器 0

SFR 页 = **0**

SFR 地址 = **0xB4** 复位值 = 0000-00XX

7	6	5	4	3	2	1	0
P4PU1	P4PU0	P2PU1	P2PU0	P1PU1	P1PU0	--	--
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**W**	**W**

Bit 7: 端口 4(P4) 高四位上拉使能控制位
0: 在集电极开漏输出模式禁止 P4.7, P4.5, P4.4 上拉
1: 在集电极开漏输出模式使能 P4.7, P4.5, P4.4 上拉

Bit 6: 端口 4(P4) 低四位上拉使能控制位
0: 在集电极开漏输出模式禁止 P4.1, P4.0 上拉
1: 在集电极开漏输出模式使能 P4.1, P4.0 上拉

Bit 5: 端口 2(P2) 高四位上拉使能控制位
0: 在集电极开漏输出模式禁止 P2.7 ~ P2.4 上拉
1: 在集电极开漏输出模式使能 P2.7 ~ P2.4 上拉

Bit 4: 端口 2(P2) 低四位上拉使能控制位
0: 在集电极开漏输出模式禁止 P2.3 ~ P2.0 上拉
1: 在集电极开漏输出模式使能 P2.3 ~ P2.0 上拉

Bit 3: 端口 1(P1) 高四位上拉使能控制位
0: 在集电极开漏输出模式禁止 P1.7 ~ P1.4 上拉
1: 在集电极开漏输出模式使能 P1.7 ~ P1.4 上拉

Bit 2: 端口 1(P1) 低四位上拉使能控制位
0: 在集电极开漏输出模式禁止 P1.3 ~ P1.0 上拉
1: 在集电极开漏输出模式使能 P1.3 ~ P1.0 上拉

Bit 1~0: 保留位。当 PUCON0 被写时这些位软件必须写“0”

PUCON1: 端口上拉控制寄存器 1

SFR 页 = **1**

SFR 地址 = **0xB4** 复位值 = XXXX-00XX

7	6	5	4	3	2	1	0
--	--	--	--	--	P6PU0	--	--
W	**W**	**W**	**W**	**W**	**R/W**	**W**	**W**

Bit 7 ~ 3: 保留位。当 PUCON1 被写时这些位软件必须写“0”

Bit 2: 端口 6(P6) 低四位上拉使能控制位
0: 在集电极开漏输出模式禁止 P6.3 ~ P6.0 上拉
1: 在集电极开漏输出模式使能 P6.3 ~ P6.0 上拉

Bit 1 ~ 0: 保留位。当 PUCON1 被写时这些位软件必须写“0”

14.3. 输入输出口示例代码

(1). 规定功能: 设置 P1.0 为片内上拉电阻使能的输入模式

汇编语言代码范例:

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

C 语言代码范例:

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

15. 中断

MA82G5BXX 有 **16** 个中断源。这些中断源有几个特殊功能寄存器 SFR 与设定四个级别的中断优先级相关。这些特殊功能寄存器分别是 IE, IP0L, IP0H, EIE1, EIP1L, EIP1H, EIE2, EIP2L, EIP2H 和 XICON。IP0H（中断优先级 0 高字节），EIP1H（扩展中断优先级 1 高字节）和 EIP2H（扩展中断优先级 2 高字节）寄存器使四个级别的中断结构合理分配。四个级别的中断优先级在处理这些中断源时更加灵活。

15.1. 中断结构

表 15–1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位或清零，这和硬件置位或清零结果相同。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。

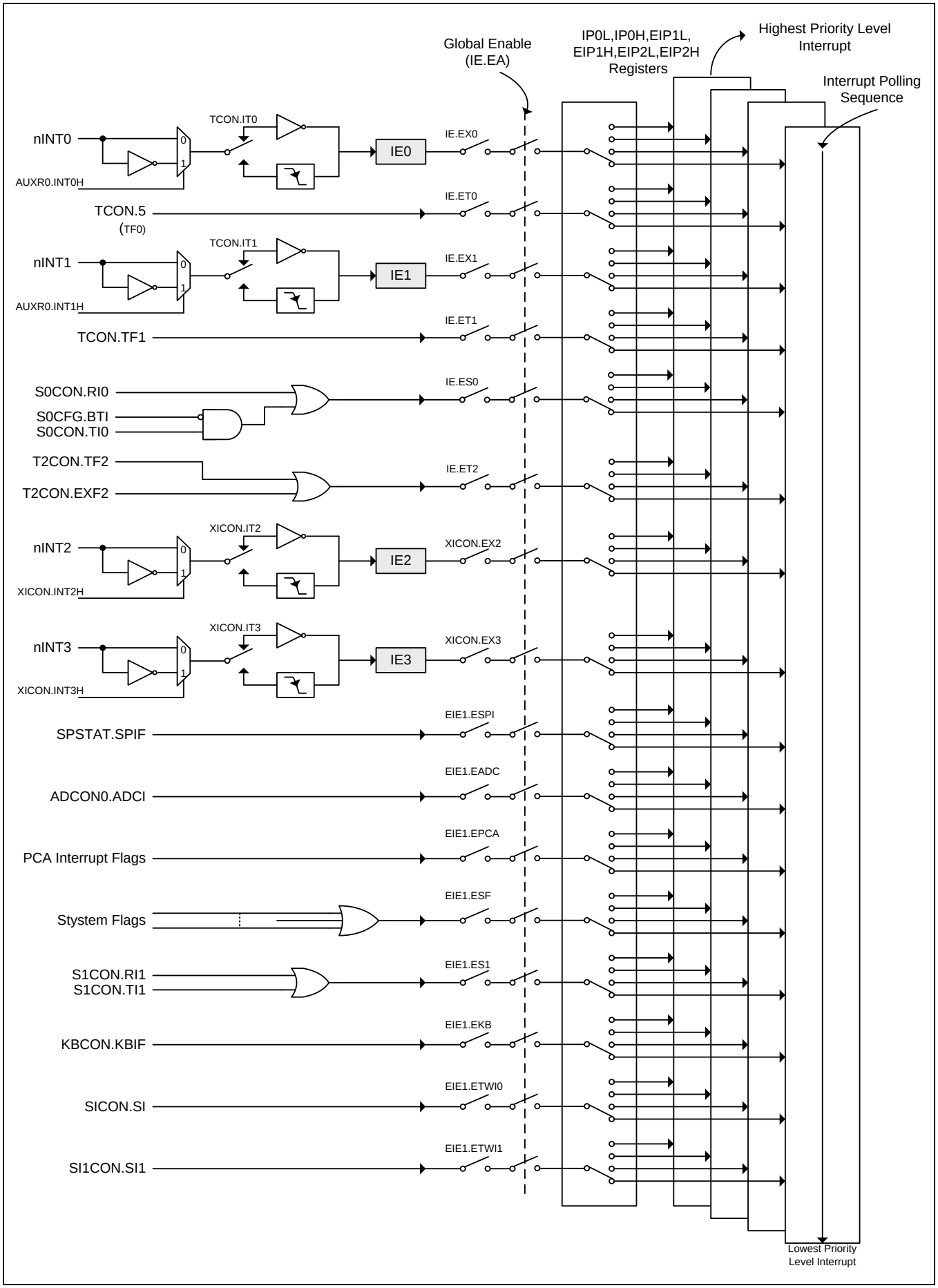
图 15–1展示了整个中断系统。每一个中断将在下面部分做简单的描述。

表 15–1. 中断源

序号	中断名称	使能位	请求位	优先级位	优先级	向量地址
#1	外部中断 0, nINT0	EX0	IE0	[PX0H, PX0L]	(最高)	0003H
#2	定时器 0	ET0	TF0	[PT0H, PT0L]	...	000Bh
#3	外部中断 1, nINT1	EX1	IE1	[PX1H, PX1L]	...	0013H
#4	定时器 1	ET1	TF1	[PT1H, PT1L]	...	001BH
#5	串行口 0	ES0	RI0, TI0	[PS0H, PS0L]	...	0023H
#6	定时器 2	ET2	TF2, EXF2	[PT2H, PT2L]	...	002Bh
#7	外部中断 2, nINT2	EX2	IE2	[PX2H, PX2L]	...	0033H
#8	外部中断 3, nINT3	EX3	IE3	[PX3H, PX3L]	...	003BH
#9	SPI	ESPI	SPIF	[PSPIH, PSPIL]	...	0043H
#10	ADC	EADC	ADCI	[PADCH, PADCL]	...	004Bh
#11	PCA	EPCA	CF, CCFn (n=0~5)	[PPCAH, PPCAL]	...	0053H
#12	系统标志	ESF	(Note 1)	[PSFH, PSFL]	...	005BH
#13	串行口 1	ES1	RI1, TI1	[PS1H, PS1L]	...	0063H
#14	键盘中断	EKB	KBIF	[PKBH, PKBL]	...	006BH
#15	TWI0	ETWI0	SI	[PTWI0H, PTWI0L]	...	0073H
#16	TWI1	ETWI1	SI1	[PTWI1H, PTWI1L]	(最低	007BH

注意1: 系统标志中断标志位包括：PCON1寄存器的WDTF, BOF0, BOF1, RTCF和MCDF；S0CON寄存器的TI0 ；AUXR3寄存器的STAF和STOF。

图 15–1. 中断系统


Global Enable
(IE.EA)
IP0L,IP0H,EIP1L,
EIP1H,EIP2L,EIP2H
Registers
Highest Priority Level
Interrupt
Interrupt Polling
Sequence
nINT0
AUXR0.INT0H
TCON.IT0
IE0
IE.EX0
TCON.5
(TF0)
IE.ET0
nINT1
AUXR0.INT1H
TCON.IT1
IE1
IE.EX1
TCON.TF1
IE.ET1
S0CON.RI0
S0CFG.BTI
S0CON.TI0
IE.ES0
T2CON.TF2
T2CON.EXF2
IE.ET2
nINT2
XICON.INT2H
XICON.IT2
IE2
XICON.EX2
nINT3
XICON.INT3H
XICON.IT3
IE3
XICON.EX3
SPSTAT.SPIF
EIE1.ESPI
ADCON0.ADCI
EIE1.EADC
PCA Interrupt Flags
EIE1.EPCA
Stystem Flags
EIE1.ESF
S1CON.RI1
S1CON.TI1
EIE1.ES1
KBCON.KBIF
EIE1.EKB
SICON.SI
EIE1.ETWI0
SI1CON.SI1
EIE1.ETWI1
Lowest Priority
Level Interrupt

15.2. 中断源

表 15–2. 中断源标志

序号	中断名称	请求位	位的位置
#1	外部中断 0,nINT0	IE0	TCON.1
#2	定时器 0	TF0	TCON.5
#3	外部中断 1,nINT1	IE1	TCON.3
#4	定时器 1	TF1	TCON.7
#5	串行口 0	RI0, TI0	S0CON.0 S0CON.1
#6	定时器 2	TF2, EXF2	T2CON.7 T2CON.6
#7	外部中断 2,nINT2	IE2	XICON.1
#8	外部中断 3,nINT3	IE3	XICON.5
#9	SPI	SPIF	SPSTAT.7
#10	ADC	ADCI	ADCON0.4
#11	PCA	CF, CCFn (n=0~5)	CCON.7 CCON.5~0
#12	系统标志	WDTF, BOF1, BOF0, RTCF, MCDF, STAF, STOF, (TI0)	PCON1.0 PCON1.1 PCON1.2 PCON1.4 PCON1.5 AUXR3.7 AUXR3.6 S0CON.1
#13	串行口 1	RI1, TI1	S1CON.0 S1CON.1
#14	键盘中断	KBIF	KBCON.0
#15	TWI0	SI	SICON.3
#16	TWI1	SI1	SI1CON.3

通过 TCON 寄存器的位 IT0 和 IT1 及 XICON 寄存器的位 IT2 和 IT3 可以设定外部中断 0（INT0）和外部中断 1（INT1）及外部中断 2（nINT2） 和外部中断 3（nINT3）为电平触发或边沿触发。中断被触发后将置位 TCON 的 IE0 或 IE1，XICON 的 IE2 或 E3。如果中断被激活这些标志在进入中断服务程序后被硬件清除，那么外部请求源控制中断请求而不是片内硬件。

定时器 0 和定时器 1 寄存器产生溢出时则置位 TCON 寄存器中的溢出标志位 TF0、TF1。如果中断被激活这些标志在进入中断服务程序后被硬件清除。

串行口 0 中断由位 RI0 和位 TI0 的逻辑或产生。执行中断服务程序后不会被硬件清除须由软件清零，可以在中断服务程序中查询 RI0 和 TI0 判断是接收中断还是发送中断。

定时器/计数器 2 中断由两个标志位 TF2 和 EXF2 产生。跟串行口一样，执行中断服务程序后不会被硬件清除须由软件清零，可以在中断服务程序中查询 TF2 和 EXF2 判断执行哪个中断服务子程序。

SPI 中断由 SPSTAT 寄存器的位 SPIF 来产生，SPIF 在 SPI 传输结束由 SPI 引擎置位。执行中断服务程序后不会被硬件清除。

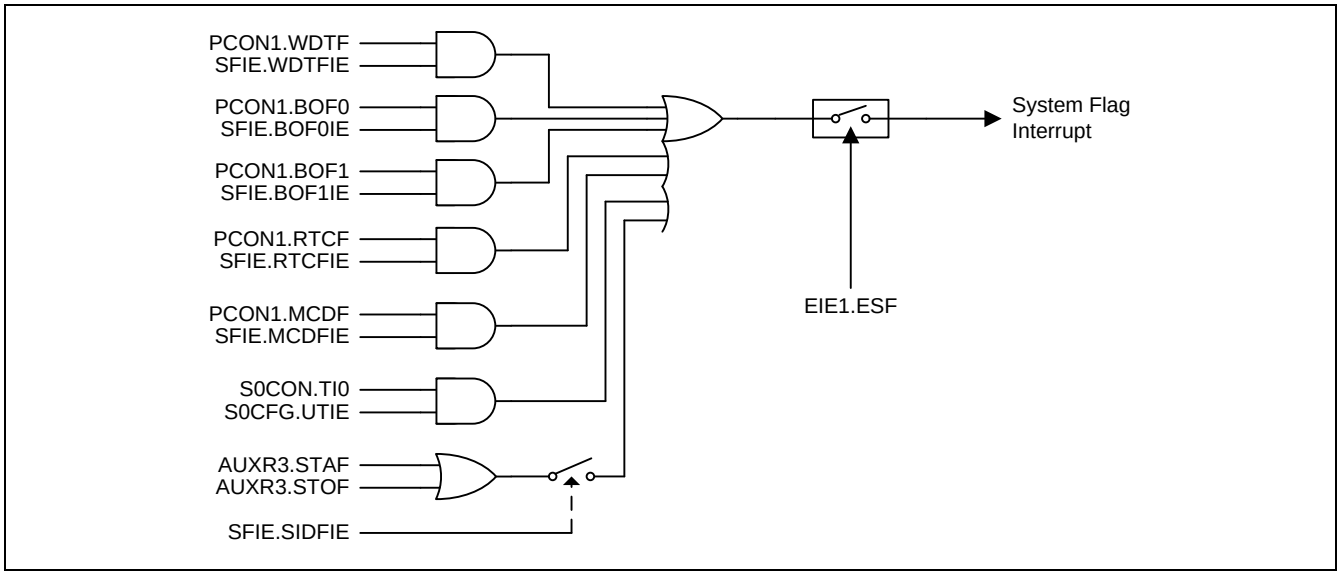
ADC 中断由 ADCON0 寄存器的位 ADCI 来产生。执行中断服务程序后不会被硬件清除。

PCA 中断由 CCON 寄存器位 CF, CCF5, CCF4, CCF3, CCF2, CCF1 和 CCF0 的逻辑或来产生。这些位执行中断服务程序后不会被硬件清除须由软件清零，可以在中断服务程序中查询这些位判断执行哪个中断服务子程序。

系统标志中断由 MCDF, RTCF, BOF1, BOF0, WDTF, TI0, STAF 和 STOF 标志位来产生。AUXR3 的 STAF 和 STOF 由串行接口侦测来置位。串行口 TI 标志位通过 UTIE 置位可选择中断向量共享系统标志中断。剩余的标志位

都在 PCON1。MCDF 由 **MCD** 激活置位。RTCF 由 RTC 计数器溢出置位。BOF1 和 BOF0 由片内低电压检测器 (BOD1 和 BOD01)遇到低电压置位。WDTF 由看门狗溢出置位。执行中断服务程序后不会被硬件清除。图 15–2 展示了系统标志中断结构。

图 15–2. 系统标志中断结构



串行口 1 中断由位 RI1 和位 TI1 的逻辑或产生。执行中断服务程序后不会被硬件清除须由软件清零，可以在中断服务程序中查询 RI1 和 TI1 判断是接收中断还是发送中断。

键盘中断由 KBCON 寄存器的位 KBIF 来产生，KBIF 由键盘模块遇到键输入来置位。执行中断服务程序后不会被硬件清除。

TWI0 中断由 SICON 寄存器的位 SI 来产生，SI 由 TWI0 引擎检测到一个新的总线状态来置位。执行中断服务程序后不会被硬件清除。

TWI1 中断由 SI1CON 寄存器的位 **SI1** 来产生，**SI1** 由 TWI1 引擎检测到一个新的总线状态来置位。执行中断服务程序后不会被硬件清除。

所有这些中断标志都能被软件置位或清零，跟硬件置位或清零的结果是一样的。也就是说，中断能通过软件来产生也可以软件来取消。

15.3. 中断使能

表 15–3. 中断使能

序号	中断名称	使能位	位的位置
#1	外部中断 0,nINT0	EX0	IE.0
#2	定时器 0	ET0	IE.1
#3	外部中断 1,nINT1	EX1	IE.2
#4	定时器 1	ET1	IE.3
#5	串行口 0	ES0	IE.4
#6	定时器 2	ET2	IE.5
#7	外部中断 2,nINT2	EX2	XICON.2
#8	外部中断 3,nINT3	EX3	XICON.3
#9	SPI	ESPI	EIE1.0
#10	ADC	EADC	EIE1.1
#11	PCA	EPCA	EIE1.2
#12	系统标志	ESF	EIE1.3
#13	串行口 1	ES1	EIE1.4

#14	键盘中断	EKB	EIE1.5
#15	TWI0	ETWI0	EIE1.6
#16	TWI1	ETWI1	EIE1.7

MA82G5BXX 有 **16** 个中断源可用。每个中断源可以通过 IE, EIE1 和 XICON 寄存器的中断使能位置位或清零各自中断使能或禁止。IE 也提供一个全局中断使能位（EA），此位清零可以立刻禁止所有中断。如果此位置位中断由相应的中断使能位各自使能或禁止。如果此位清零则所有中断被禁止。

15.4. 中断优先级

服务中断的优先级除了有 4 个级别比 80C51 多 2 个之外跟 80C51 一样。优先级位决定每个中断的优先级（见表 15–1）。IP0L, IP0H, EIP1L 和 EIP1H 跟 4 个级别优先级中断相关。表 15–4 显示位的值和优先级的关系。

表 15–4. 中断优先级

{IPnH.x , IPnL.x}	优先级
11	1 (最高)
10	2
01	3
00	4

每个中断源都有两个中断优先级相关位。一个位在 IPnH 寄存器另一个在 IPnL 寄存器。高优先级中断不会被低优先级中断打断。如果两个不同优先级的中断请求同时出现，较高优先级将被执行。如果相同优先级的中断请求同时出现，则按照内部优先级排序执行。表 15–2 显示了同一优先级的内部优先级排序和中断向量地址。

15.5. 中断处理

每一个系统时钟周期将采样每一个中断标志。在下一个系统时钟采样成功。如果其中一个标志在第一个周期置位，第二个周期找到并且只要没有被下列条件阻止则中断系统产生一个硬件调用（LCALL）相应的中断服务程序。

阻止条件：

- 进行中已经有一个同级或更高级优先级的中断
- 进行中当前周期（中断获得周期）不是指令执行结束周期
- 指令进行是 RETI 或 IE, IP0L, IP0H, EIE1, EIP1L 和 EIP1H 寄存器的写操作

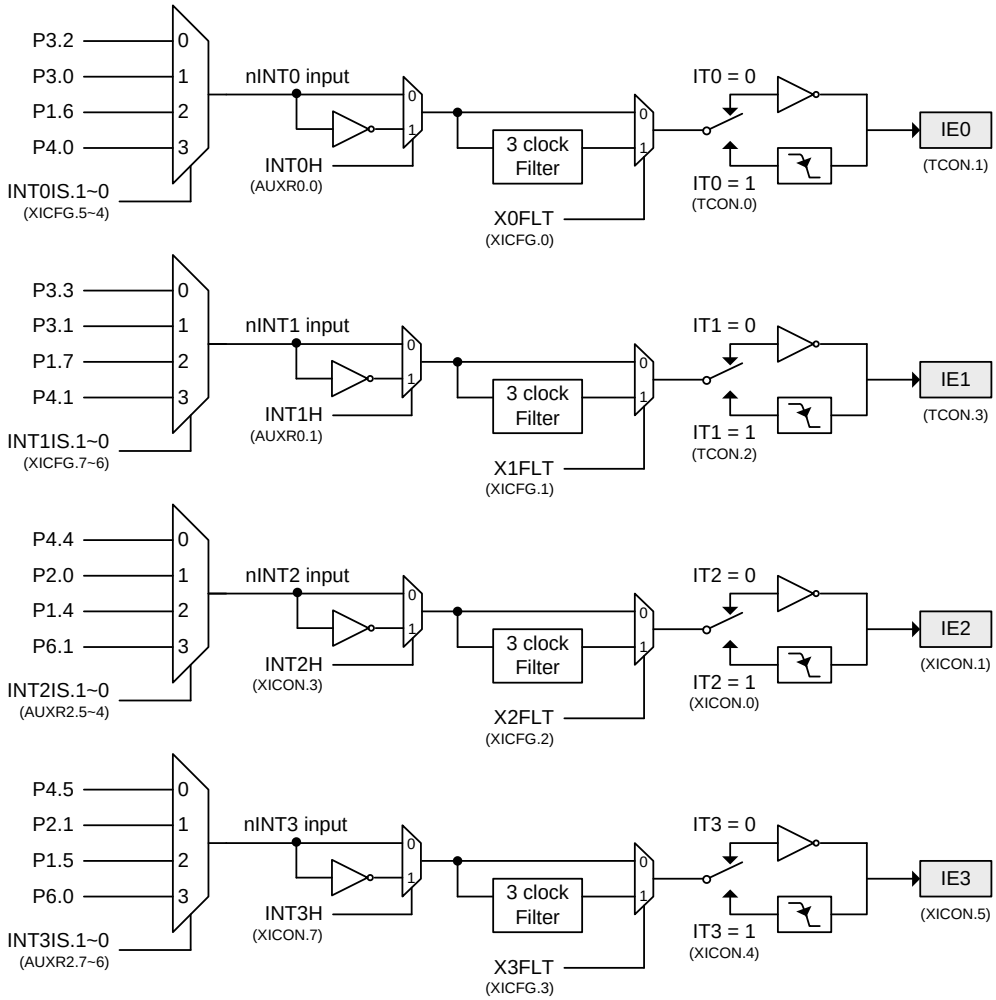
上述三个条件中的任意一个将阻止硬件中断调用（LCALL）去中断服务程序。条件 2 确保中断进入任意一个服务程序之前指令执行完毕。条件 3 确保如果在 RETI 执行或 IE, IP0L, IP0H, EIE1, EIP1L 及 EIP1H 的访问之后，进入中断服务程序之前至少一个或更多指令被执行。

15.6. nINTi 输入源选择和输入滤波器 (i=0~3)

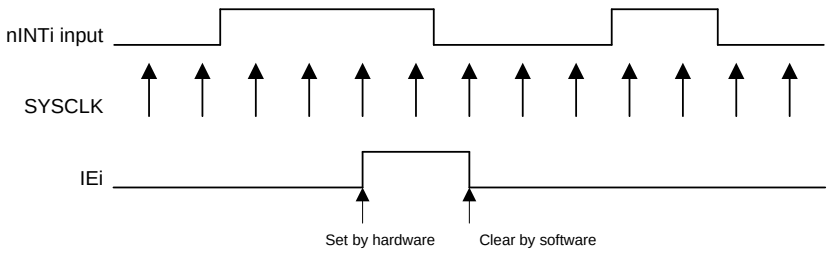
MA82G5BXX 提供灵活的 nINT0, nINT1, nINT2 和 nINT3 输入源选择为了共享片内串行接口的端口引脚输入。支持掉电模式通讯外设的额外远程唤醒功能。nINTi 输入可以设置成捕获端口变化的接口引脚并且设置他们的中断输入事件来唤醒 MCU。INT0H (AUXR0.0), INT1H (AUXR0.1), INT2H (XICON.3)和 INT3H (XICON.7)配置端口改变检测电平是低/下降沿或高/上升沿触发。在 MCU 掉电模式外部中断的下降沿或上升沿都被强制成电平检测操作。

每个外部中断输入都有一个滤波器通过 3 个系统时钟周期(SYSCLK)来识别外部中断信号来增强噪声的抑制。图 15–3 展示了外部中断结构和滤波器特性。

图 15–3. nINTi 端口引脚选择的结构和输入滤波器



Operating Example of nINTi Input Filter (XiFLT=1, i = 0~3)



15.7. 中断寄存器

TCON: 定时器/计数器控制寄存器

SFR 页 = 0~F

SFR 地址 = 0x88 复位值 = 0000-0000

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

Bit 3: IE1, 外部中断 1 请求标志
0: 如果是边沿触发的中断则在进入中断向量后硬件清零
1: 外部中断 1 由边沿或电平触发（由 IT1 设置）硬件置标志

Bit 2: IT1: 外部中断 1 类型控制位
0: 软件选择低电平触发外部中断 1 。如果 INT1H (AUXR0.1)置位，则高电平触发外部中断 1(nINT1)
1: 软件选择下降沿触发外部中断 1 。如果 INT1H (AUXR0.1)置位，则上升沿触发外部中断 1(nINT1)

Bit 1: IE0, 外部中断 0 请求标志
0: 如果是边沿触发的中断则在进入中断向量后硬件清零
1: 外部中断 0 由边沿或电平触发（由 IT0 设置）硬件置标志

Bit 0: IT0: 外部中断 0 类型控制位
0: 软件选择低电平触发外部中断 0 。如果 INT0H (AUXR0.0)置位，则高电平触发外部中断 0
1: 软件选择下降沿触发外部中断 0 。如果 INT0H (AUXR0.0)置位，则上升沿触发外部中断 0

IE: 中断使能寄存器

SFR 页 = 0~F

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	**W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: EA, 总中断使能位
0: 禁止所有中断
1: 使能所有中断

Bit 6: 保留。当 IE 写入时，此位软件必须写”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, 串行口 0 中断使能
0: 禁止串行口 0 中断
1: 使能串行口 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 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 使能
0: 禁止外部中断 0
1: 使能外部中断 0

XICON: 扩展中断控制寄存器

SFR 页 = 0~F

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,外部中断 3(nINT3)触发方式控制位
0:被选择端口引脚输入的下降沿/低电平触发 nINT3.
1:被选择端口引脚输入的上升沿/高电平触发 nINT3

Bit 6: EX3, 外部中断 3 使能位
0: 禁止外部中断 3
1: 使能外部中断 3
当 CPU 在空闲或掉电模式，如果 EX3 使能 nINT3 事件触发 IE3 则可以唤醒 CPU；如果 EX3 禁止，nINT3 事件触发 IE3 则不能唤醒 CPU。

Bit 5: IE3, 外部中断 3 中断标志
0: 如果是边沿触发的中断则在进入中断向量后硬件清零，也可以 CPU 清零
1: 外部中断 3 由边沿或电平触发（由 IT3 设置）硬件置位，也可以 CPU 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外部中断 3 。如果 INT3H 置位，则高电平触发外部中断 3
1: 软件选择下降沿触发外部中断 3 。如果 INT3H 置位，则上升沿触发外部中断 3

Bit 3: INT2H, 外部中断 2 触发方式控制位
0:被选择端口引脚输入的下降沿/低电平触发 nINT2
1:被选择端口引脚输入的上升沿/高电平触发 nINT2

Bit 2: EX2, 外部中断 2 使能位
0: 禁止外部中断 2
1: 使能外部中断 2
当 CPU 在空闲或掉电模式，如果 EX2 使能 nINT2 事件触发 IE2 则可以唤醒 CPU；如果 EX2 禁止，nINT2 事件触发 IE2 则不能唤醒 CPU。

Bit 1: IE2, 外部中断 2 中断标志
0: 如果是边沿触发的中断则在进入中断向量后硬件清零，也可以软件清零
1: 外部中断 2 由边沿或电平触发（由 IT2 设置）硬件置位，也可以软件置位

Bit 0: IT2, 外部中断 2 类型控制位
0: 软件选择低电平触发外部中断 2 。如果 INT2H 置位，则高电平触发外部中断 2
1: 软件选择下降沿触发外部中断 2 。如果 INT2H 置位，则上升沿触发外部中断 2

EIE1: 扩展中断使能 1 寄存器

SFR 页 = 0~F

SFR 地址 = 0xAD 复位值 = 0000-0000

7	6	5	4	3	2	1	0
ETWI1	ETWI0	EKBI	ES1	ESF	EPCA	EADC	ESPI
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: ETWI1, TWI1 中断使能
0: 禁止 TWI1 中断
1: 使能 TWI1 中断

Bit 6: ETWI0, TWI0 中断使能
0: 禁止 TWI0 中断
1: 使能 TWI0 中断

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

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

Bit 3: ESF, 系统标志中断使能
0:当 PCON1 的位{ MCDF, RTCF, BOF1, BOF0, WDTF }，AUXR3 的位{STAF, STOF}，UTIE 的位{TI0}任一置位时禁止中断
1:当 SFIE 寄存器的相关系统标志中断使能并且 PCON1 的位{ MCDF, RTCF, BOF1, BOF0, WDTF }，AUXR3 的位{STAF, STOF}，UTIE 的位{TI0}任一置位时使能中断

Bit 2: EPCA, PCA 中断使能
0: 禁止 PCA 中断
1: 使能 PCA 中断

Bit 1: EADC, ADC 中断使能
0: 当 ADC 模块的 ADCON0.ADCI 置位禁止中断
1: 当 ADC 模块的 ADCON0.ADCI 置位使能中断

Bit 0: ESPI, SPI 中断使能
0: 当 SPI 模块的 SPSTAT.SPIF 置位禁止中断
1: 当 SPI 模块的 SPSTAT.SPIF 置位使能中断

SFIE: 系统标志中断使能寄存器

SFR 页 = 0~F

SFR 地址 = 0x8E 复位值 = 0110-X000

7	6	5	4	3	2	1	0
SIDFIE	MCDRE	MCDFIE	RTCFIE	--	BOF1IE	BOF0IE	WDTFIE
R/W	**R/W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**	**R/W**

Bit 7: SIDFIE, 串行接口(STWI)侦测标志中断使能
0: 禁止 SIDF(STAF 或 STOF)中断
1: 使能 SIDF(STAF 或 STOF)中断共享系统标志中断

Bit 6: MCDRE (校验中)

Bit 5: MCDFIE (校验中)

Bit 4: RTCFIE, RTCF (PCON1.4)中断使能
0: 禁止 RTCF 中断
1: 使能 RTCF 中断

Bit 3: 保留。当 SFIE 写入时，这些位软件必须写”0”。

Bit 2: BOF1IE, BOF1 (PCON1.2)中断使能
0: 禁止 BOF1 中断
1: 使能 BOF1 中断

Bit 1: BOF0IE, BOF0 (PCON1.1) 中断使能
0: 禁止 BOF0 中断
1: 使能 BOF0 中断

Bit 0: WDTFIE, WDTF (PCON1.0) 中断使能
0: 禁止 WDTF 中断
1: 使能 WDTF 中断

IP0L: 中断优先级 0 低字节寄存器

SFR 页 = 0~F
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 中断优先级低位
Bit 6: PX2L, 外部中断 2 中断优先级低位
Bit 5: PT2L, 定时器 2 中断优先级低位
Bit 4: PSL, UART0 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 中断优先级低位
Bit 1: PT0L, 定时器 0 中断优先级低位
Bit 0: PX0L, 外部中断 0 中断优先级低位

IP0H: 中断优先级 0 高字节寄存器

SFR 页 = 0~F
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 中断优先级高位
Bit 6: PX2H, 外部中断 2 中断优先级高位
Bit 5: PT2H, 定时器 2 中断优先级高位
Bit 4: PSH, UART0 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 中断优先级高位
Bit 1: PT0H, 定时器 0 中断优先级高位
Bit 0: PX0H, 外部中断 0 中断优先级高位

EIP1L: 扩展中断优先级 1 低字节寄存器

SFR 页 = 0~F
SFR 地址 = 0xAE 复位值 = X000-0000

7	6	5	4	3	2	1	0
PTWI1L	PTWI0L	PKBL	PS1L	PSFL	PPCAL	PADCL	PSPIL
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: PTWI1L, TWI1 中断优先级低位
Bit 6: PTWI0L, TWI0 中断优先级低位
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: PSFL, 系统标志中断优先级低位
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 高字节寄存器

SFR 页 = 0~F
SFR 地址 = 0xAF 复位值 = X000-0000

7	6	5	4	3	2	1	0
PTWI1H	PTWIH	PKBH	PS1H	PSFH	PPCAH	PADCH	PSPIH
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: PTWI1H, TWI1 中断优先级高位
Bit 6: PTWI0H, TWI0 中断优先级高位
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: PSFH, 系统标志中断优先级高位
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

AUXR0: 辅助寄存器 0

SFR 页 = 0~F
SFR 地址 = 0xA1 复位值 = 000X-0000

7	6	5	4	3	2	1	0
P60OC1	P60OC0	P60FD	T0XL	P4FS1	P4FS0	INT1H	INT0H
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 1: INT1H,外部中断 1(INT1)触发方式控制位
0:被选择端口引脚输入的下降沿/低电平触发 nINT1
1:被选择端口引脚输入的上升沿/高电平触发 nINT1

Bit 0: INT0H, 外部中断 0(INT0)触发方式控制位
0: 被选择端口引脚输入的下降沿/低电平触发 nINT0
1: 被选择端口引脚输入的上升沿/高电平触发 nINT0

AUXR2: 辅助寄存器 2

SFR 页 = 0~F
SFR 地址 = **0xA3** 复位值 = 0000-0000

7	6	5	4	3	2	1	0
INT3IS1	INT3IS0	INT2IS1	INT2IS0	T1X12	T0X12	T1CKOE	T0CKOE
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~6: INT3IS1~0, nINT3 的输入选择位功能定义如下表

INT3IS1~0	nINT3 的输入选择引脚
00	P4.5
01	P2.1
10	P1.5
11	P6.0

Bit 5~4: INT2IS1~0, nINT2 的输入选择位功能定义如下表

INT2IS1~0	nINT2 的输入选择引脚
00	P4.4
01	P2.0
10	P1.4
11	P6.1

XICFG:外部中断配置寄存器

SFR 页 = 0~F

SFR 地址 = 0xC1 复位值 = 0000-0000

7	6	5	4	3	2	1	0
INT1IS.1	INT1IS.0	INT0IS.1	INT0IS.0	X3FLT	X2FLT	X1FLT	X0FLT
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~6: INT1IS.1~0, nINT3 的输入选择位功能定义如下表

INT1IS.1~0	nINT1 的输入选择引脚
00	P3.3
01	P3.1
10	P1.7
11	P4.1

Bit 5~4: INT0IS.1~0, nINT0 的输入选择位功能定义如下表

INT0IS.1~0	nINT0 的输入选择引脚
00	P3.2
01	P3.0
10	P1.6
11	P4.0

Bit 3: X3FLT, INT3 的滤波器使能
0: 禁止 INT3 输入的 3 个系统时钟滤波器。INT3 的默认功能
1: 使能 INT3 输入的 3 个系统时钟滤波器

Bit 2: X2FLT, INT2 的滤波器使能
0: 禁止 INT2 输入的 3 个系统时钟滤波器。INT2 的默认功能
1: 使能 INT2 输入的 3 个系统时钟滤波器

Bit 1: X1FLT, INT1 的滤波器使能
0: 禁止 INT1 输入的 3 个系统时钟滤波器。INT1 的默认功能
1: 使能 INT1 输入的 3 个系统时钟滤波器

Bit 0: X0FLT, INT0 的滤波器使能
0: 禁止 INT0 输入的 3 个系统时钟滤波器。INT0 的默认功能
1: 使能 INT0 输入的 3 个系统时钟滤波器

PCON1:电源控制寄存器 1

SFR 页 = 0~F 及 P 页

SFR 地址 = 0x97 POR = 0010-X000

7	6	5	4	3	2	1	0
SWRF	EXRF	MCDF	RTCF	--	BOF1	BOF0	WDTF
R/W	**R/W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**	**R/W**

Bit 7: SWRF, 软件复位标志
0: 此位软件写**"1"**清零
1: 如果软件复位出现此位由硬件置位

Bit 6: EXRF, 外部复位标志
0: 此位软件写**"1"**清零
1: 如果外部复位出现此位由硬件置位

Bit 5: MCDF. (校验中)

Bit 4: RTCF, RTC 溢出标志

0: 此位软件写**“1”**清零。软件写**“0”**无效
1: 如果 RTCCT 溢出此位由硬件置位。写**“1”** RTCF 清零

Bit 3: 保留。当 PCON1 写入时，这些位软件必须写”0”

Bit 2: BOF1,掉电侦测标志 1
0: 此位软件写**“1”**清零
1: 如果工作电压达到掉电侦测标志 1 的侦测电压 1 (4.2V/3.7/2.4/2.0)，此位由硬件置位

Bit 1: BOF0, 掉电侦测标志 0
0: 此位软件写**“1”**清零
1: 如果工作电压达到掉电侦测标志 0 的侦测电压(1.7V)，此位由硬件置位

Bit 0: WDTF, WDT 溢出标志
0: 此位软件写**“1”**清零
1: 如果 WDT 溢出此位由硬件置位

AUXR3: 辅助寄存器 3
SFR 页 = 0~F
SFR 地址 = 0xA4 复位值 = 0000-0000

7	6	5	4	3	2	1	0
STAF	STOF	BPOC1	BPOC0	GF	P1S0MI	P3ECI	P3TWI1
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: STAF, STWI 的开始侦测标志
0: 此位软件写**“0”**清零
1: 硬件置位表明 STWI 总线出现 START

Bit 6: STOF, STWI 的停止侦测标志
0: 此位软件写**“0”**清零
1: 硬件置位表明 STWI 总线出现 STOP

15.8. 中断示例代码

(1). 规定功能: 在掉电模式下设置 INT0 高电平唤醒 MCU

汇编语言代码范例:

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

main:

   SETB    P32                    ;

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

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

   JB      P32,$                  ; 确认 P3.2 输入低

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

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

C 语言代码范例:

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

void main(void)
{
   P32 = 1;

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

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

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

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

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

16. 定时器/计数器

MA82G5BXX 有三个 16 位定时器/计数器：定时器 0，定时器 1 和定时器 2。所有这些操作既可配置为定时器或事件记数器。

定时器功能，定时器预分频是每 12 个时钟周期加 1。换句话说，定时器是标准 C51 机器周期计数一次。AUXR2.T0X12, AUXR2.T1X12 和 T2MOD.T2X12 可以设置定时器每个时钟周期计数一次。这样就是标准 C51 定时器 12 倍的速度。结合 AUXR0.T0XL 和 T0X12 定时器 0 时钟输入可选择额外的预分频 SYSCLK/48 和 SYSCLK/192

计数器功能，下降沿时寄存器加1，根据相应的外部输入引脚T0，T1或T2。在这些功能中，每个定时器时钟周期对外部输入信号进行采样。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。

16.1. 定时器 0 和 1

16.1.1. 定时器 0/1 模式 0

定时器寄存器配置为一个 PWM 产生器。计数器所有位从全 1 翻转到全 0，置位定时器中断标志位 TFx。当 TRx=1 且 GATE=0 或/INTx=1，定时器使能输入计数。定时器 0 和 1 的模式 0 操作是一样的。定时器 0/1 的 PWM 功能结构图见图 16–1 和图 16–2

图 16–1. 定时器 0 模式 0 结构

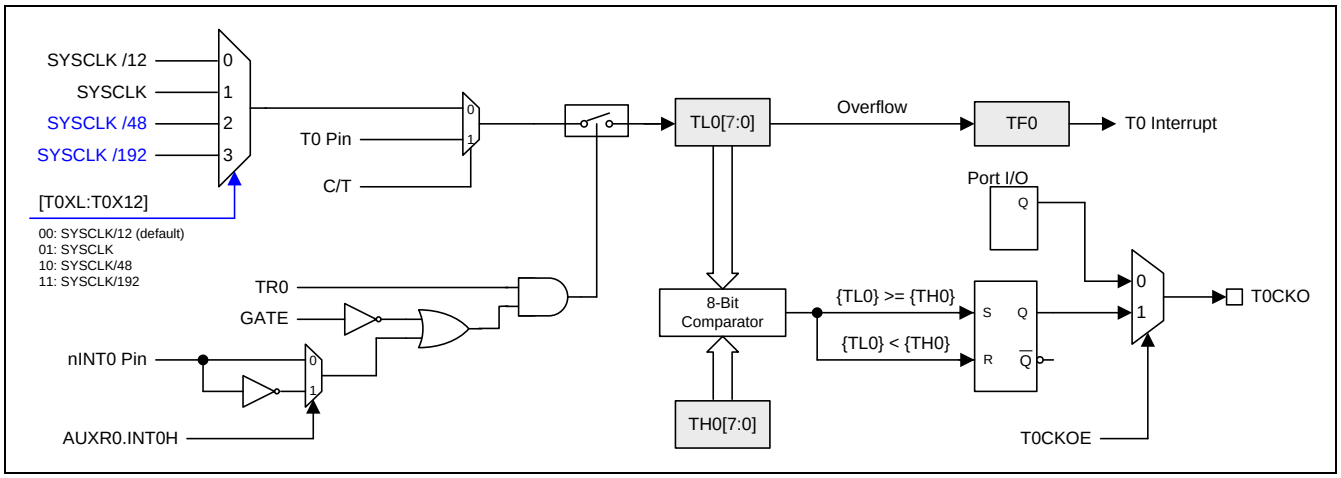
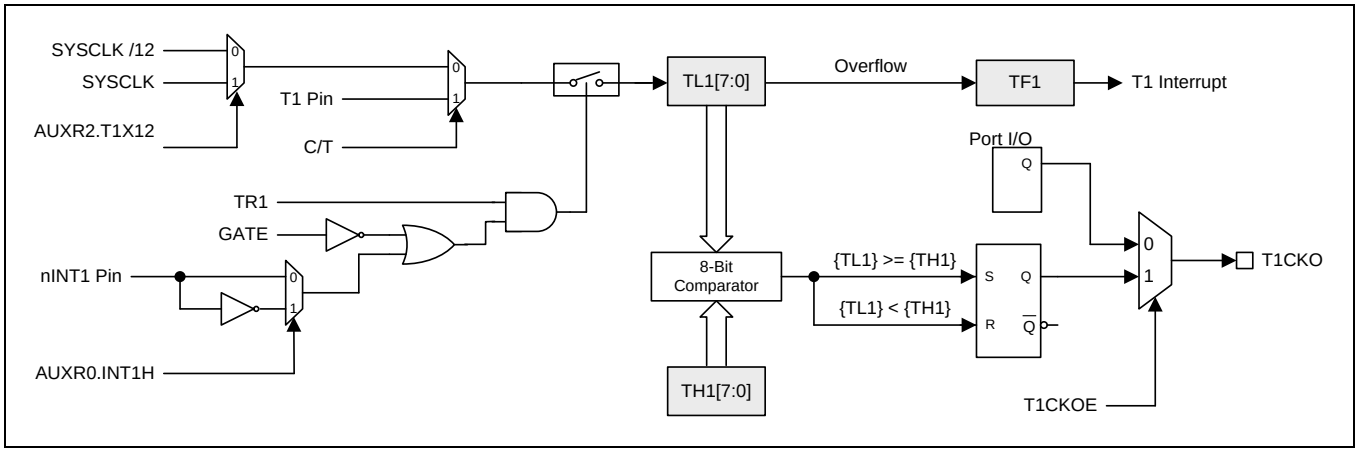


图 16–2. 定时器 1 模式 0 结构



16.1.2. 定时器 0/1 模式 1

除了定时器寄存器是 16 位之外，模式 1 跟模式 0 一样。定时器 0/1 模式 1 的结构图见图 16–3 和图 16–4

图 16–3. 定时器 0 模式 1 结构

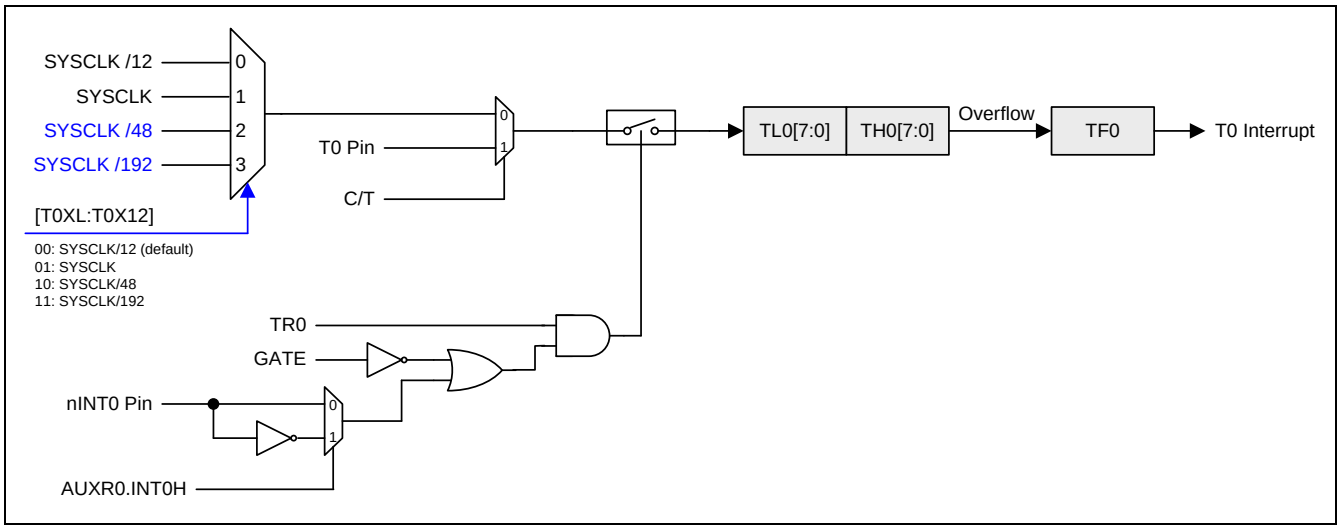
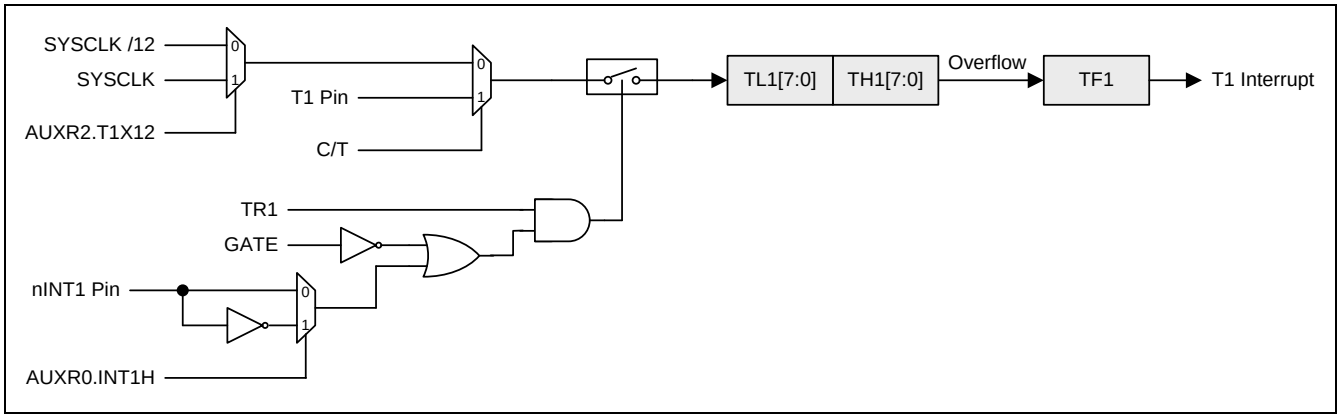


图 16–4. 定时器 1 模式 1 结构



16.1.3. 定时器 0/1 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 1 的模式 2 操作是一样的。定时器 0/1 模式 2 的结构图见图 16–5 和图 16–6

图 16–5. 定时器 0 模式 2 结构

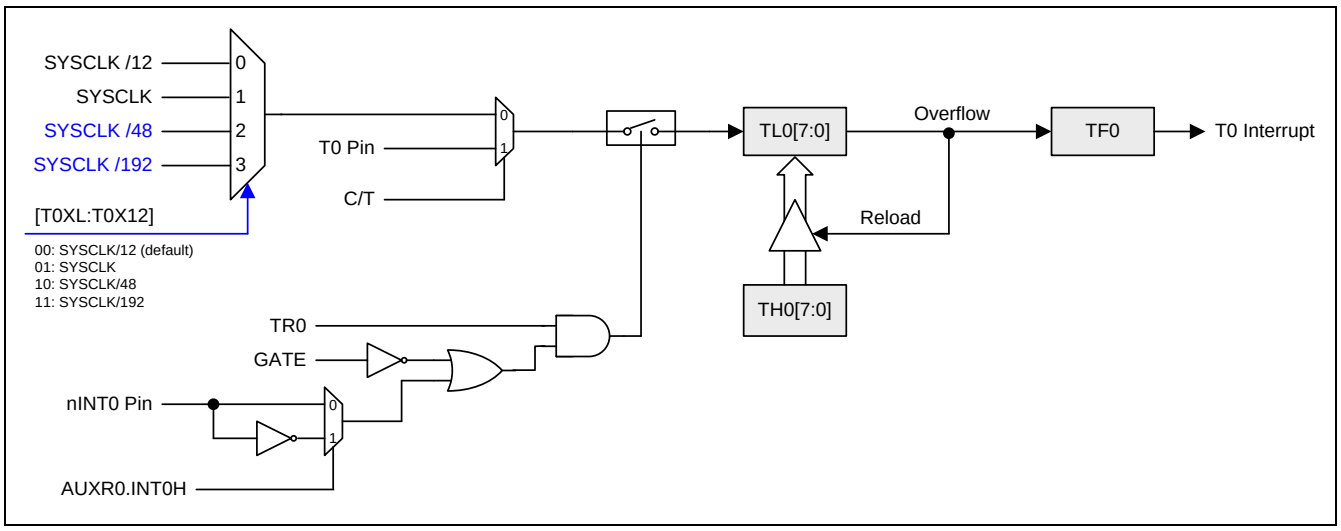
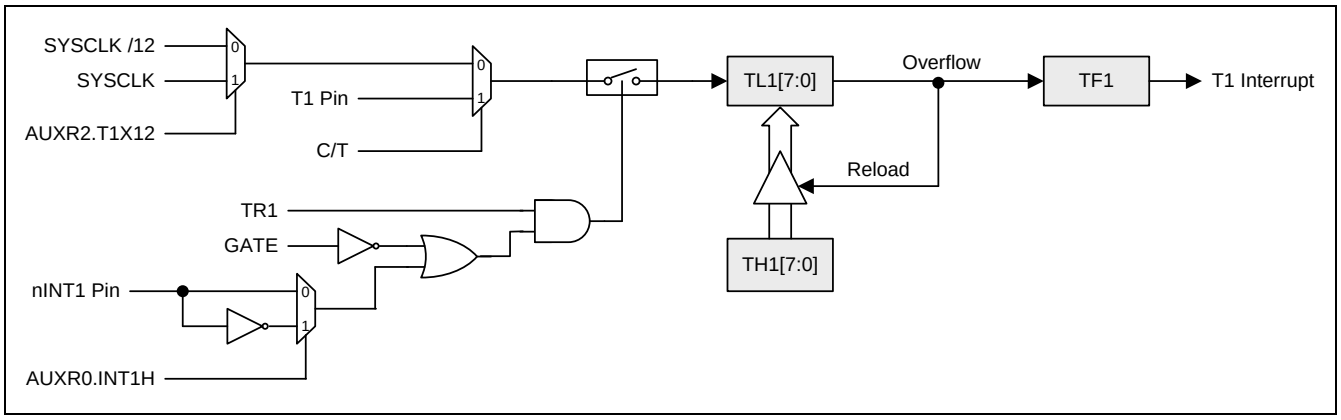


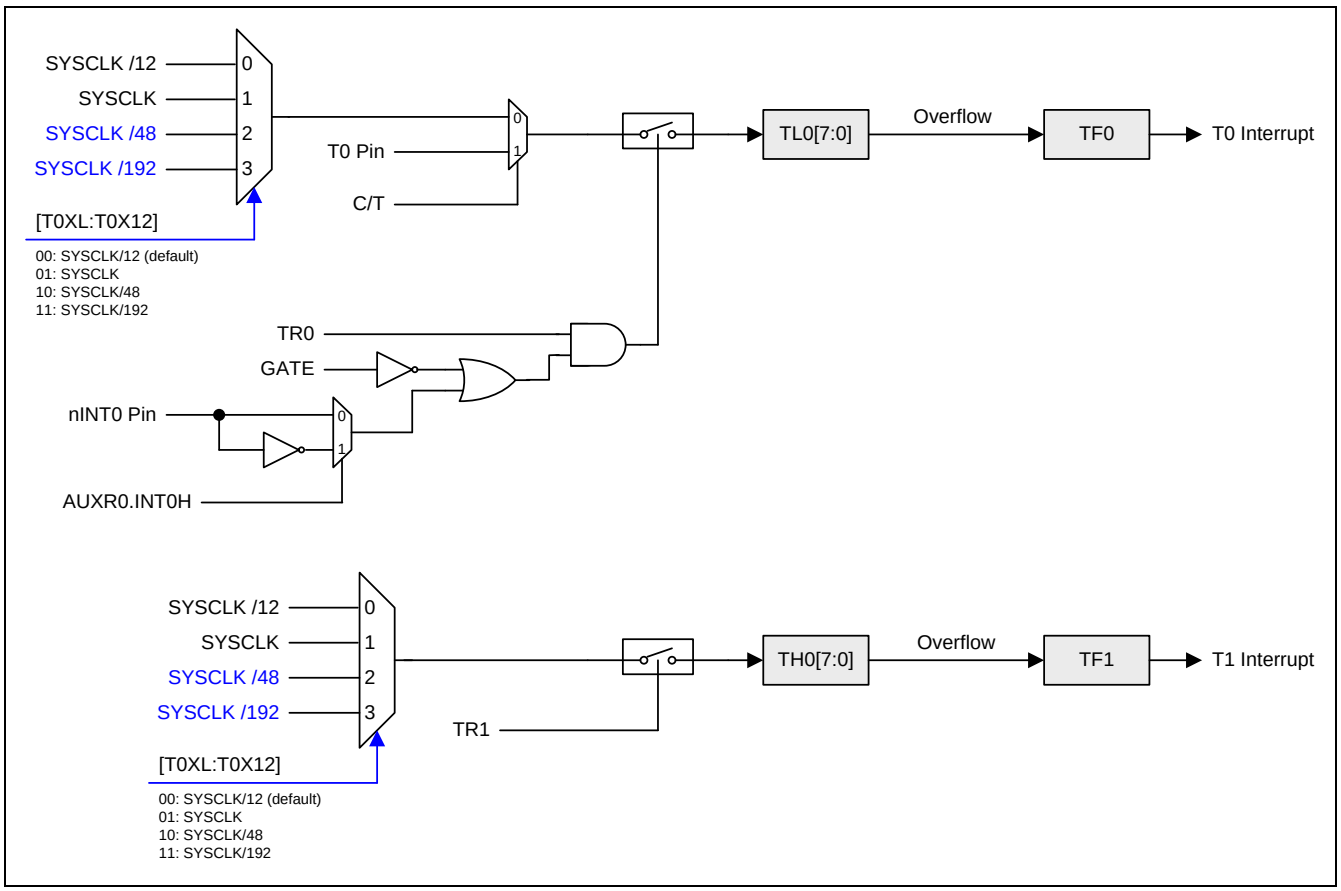
图 16–6. 定时器 1 模式 2 结构



16.1.4. 定时器 0/1 模式 3

定时器 1 在模式 3 保持计数值，效果和设置 TR1=1 一样。定时器 0 在模式 3 建立 TL0 和 TH0 两个独立的计数器。TL0 使用定时器 0 控制位：C/T、GATE、TR0、/INT0 和 TF0。TH0 锁定为定时器功能(每个机器周期计数)且接替定时器 1 来使用 TR1 和 TF1，因从 TH0 控制定时器 1 中断。定时器 0 模式 3 的结构图见图 16–7

图 16–7. 定时器 0 模式 3 结构



16.1.5. 定时器 0/1 可编程时钟输出

定时器 0 和 1 有一个时钟输出模式（当 C/Tx=0 和 TxCKOE=1）。此模式下，定时器 0 或 1 操作在 8 位自动重载占空比为 1:1 的可编程时钟发生器。产生的时钟在 P3.4 (T0CKO)和 P3.5 (T1CKO)独立输出。8 位定时器(TL0)每个输入时钟(SYSCLK/12,SYSCLK,SYSCLK/48 或 SYSCLK/192)加一,在定时器 0 模块。8 位定时器(TL1)每个输入时钟(SYSCLK/12 或 SYSCLK)加一,在定时器 1 模块。定时器从载入值到溢出重复计数。一旦溢出，(TH0,TH1)的值被载入到(TL0, TL1)同时计数。图 16–8 和图 16–9 给出了定时器 0/1 时钟输出频率公式。图 16–10 和图 16–11 给出了定时器 0/1 时钟输出结构

图 16–8. 定时器 0 时钟输出公式

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

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

图 16–9. 定时器 1 时钟输出公式

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

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

注意:

(1) 定时器 0/1 溢出标志 TF0/1,在定时器 0/1 溢出时置位但是不产生中断

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

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

(4) 当 SYSCLK=12MHz, T0X12=0 及 T0XL=1,定时器 0 可编程输出频率范围从 488Hz 到 125KHz.

(5) 当 SYSCLK=12MHz, TxX12=1 及 T0XL=1,定时器 0 可编程输出频率范围从 122Hz 到 31.25KHz.

图 16–10.定时器 0 的时钟输出模式

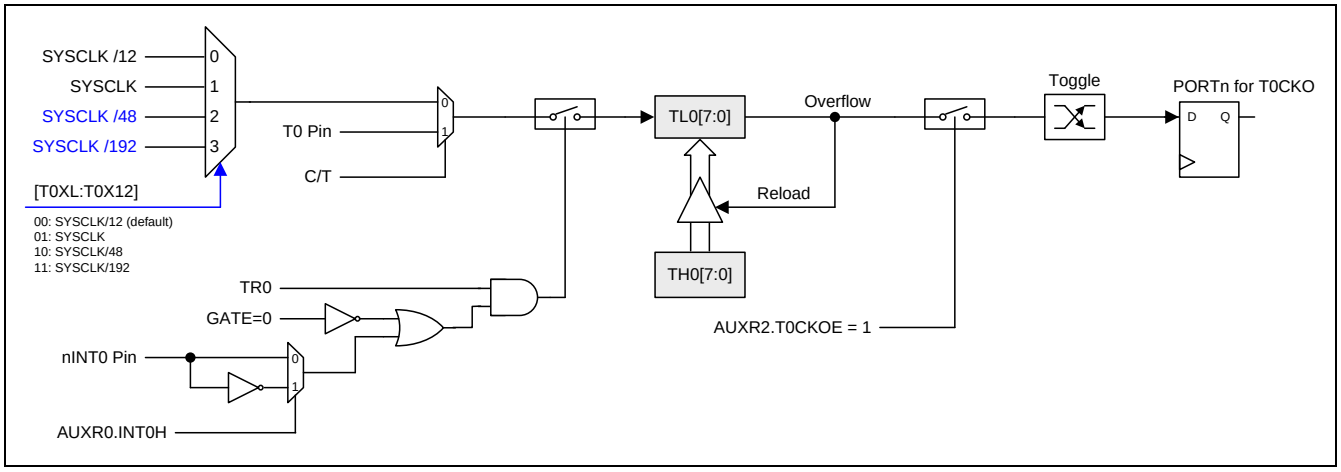
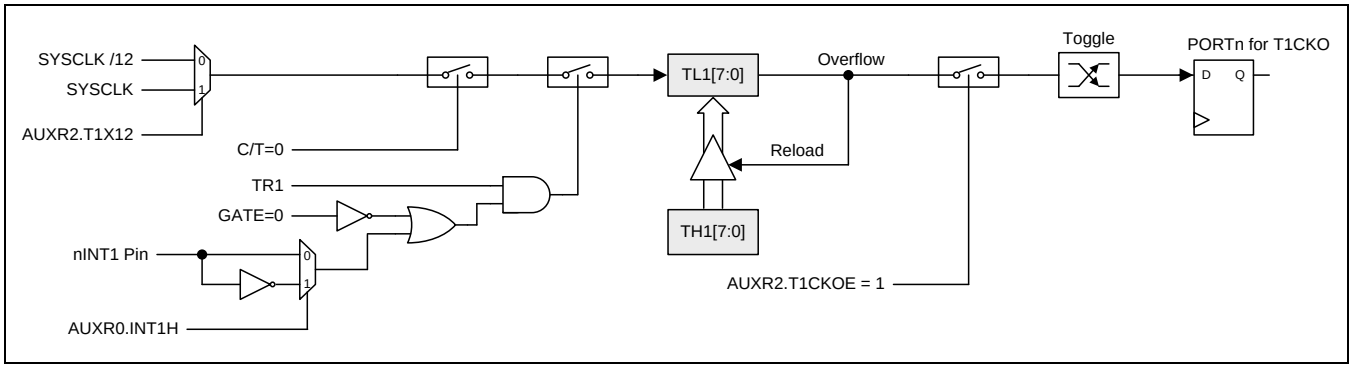


图 16–11. 定时器 1 的时钟输出模式



定时器 0/1 时钟输出模式如何编程

- AUXR2.T0X12 和 AUXR0.T0XL 来选择定时器 0 时钟源。AUXR2.T1X12 来选择定时器 1 时钟源
- AUXR2 寄存器的 T0CKOE/T1CKOE 置位
- TMOD 寄存器的 C/T 清零
- 从公式计算出 8 位自动加载值并输入到 TH0/TH1 寄存器
- 在 TL0/TL1 寄存器输入一个跟自动加载值相同 8 位初始值
- 通过设置 TCON 寄存器的 TR0/TR1 位启动定时器

时钟输出模式，定时器 0/1 溢出不会中断。这跟定时器 1 被用作波特率发生器相似。定时器 1 即可用作波特率发生器也可同时用作时钟发生器。注意，波特率和时钟输出频率都是相同的定时器 1 溢出率。

16.1.6. 定时器 0/1 寄存器

TCON: 定时器/计数器控制寄存器

SFR 页 = 0~F

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
0: 处理器进入中断向量程序由硬件清零或软件清零
1: 定时器/计数器 1 溢出时由硬件置位或软件置位

Bit 6: TR1, 定时器 1 运行控制位
0: 软件清零关闭定时器/计数器 1
1: 软件置位开启定时器/计数器 1

Bit 5: TF0, 定时器 0 溢出标志
0: 处理器进入中断向量程序由硬件清零或软件清零
1: 定时器/计数器 0 溢出时由硬件置位或软件置位

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

TMOD: 定时器/计数器模式控制寄存器

SFR 页 = 0~F

SFR 地址 = 0x89 复位值 = 0000-0000

7	6	5	4	3	2	1	0
GATE	C/T	M1	M0	GATE	C/T	M1	M0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**
|←---------- Timer1 ----------→|				|←---------- Timer0 ----------→|		

Bit 7/3: Gate, 定时器 1/0 门控制位
0: 禁止定时器 1/0 门控制
1: 使能定时器 1/0 门控制。当门控制位置位时，只有在/INT0 或/INT1 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 T1 或 T0 引脚输入)

Bit 5~4/1~0: 操作模式选择

M1	**M0**	**操作模式**
0	0	定时器 0/1 的 8 位 PWM 产生器
0	1	定时器 0/1 工作在 16 位定时器/计数器模式
1	0	定时器 0/1 工作在 8 位自动装载定时器/计数器模式
1	1 (定时器 0)	TL0 是 8 位定时器/计数器，TH0 锁定 8 位定时器
1	1 (定时器 1)	定时器/计数器停止

TL0: 定时器 0 低字节寄存器

SFR 页 = 0~F

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 高字节寄存器

SFR 页 = 0~F

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 低字节寄存器

SFR 页 = 0~F

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 高字节寄存器

SFR 页 = 0~F

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 = 0~F

SFR 地址 = 0xA3 复位值 = 0000-0000

7	6	5	4	3	2	1	0
INT3IS1	INT3IS0	INT2IS1	INT2IS0	T1X12	T0X12	T1CKOE	T0CKOE
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 3: T1X12, 当 C/T=0 时，定时器 1 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 2: T0X12, 当 C/T=0 时，定时器 0 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

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

Bit 1: T1CKOE, 定时器 1 时钟输出使能
0: 禁止定时器 1 时钟输出
1: 禁止定时器 1 时钟输出在 P3.5

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

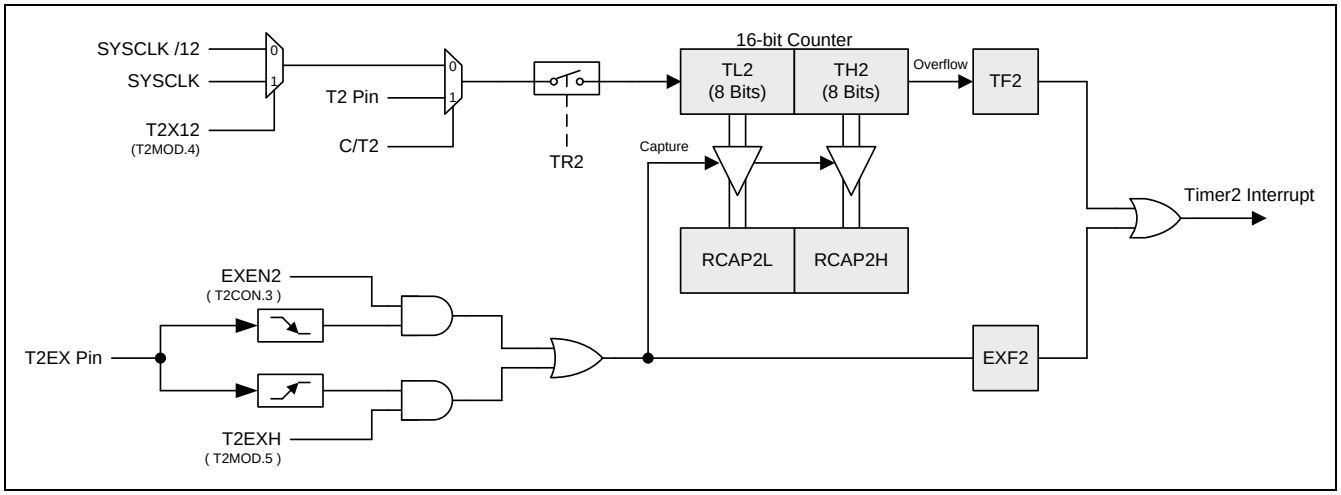
16.2. 定时器 2

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。

16.2.1. 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)。捕获模式如图 16–12 所示。

图 16–12. 定时器 2 捕获模式框图



16.2.2. 自动重载模式 (AR)

图 16–13 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。T2EXH 跟 EXEN2 一样的功能只是它侦测的是输入 T2EX 的正跳变(0-to-1)。

图 16–13. 定时器 2 自动重载模式(DCEN=0)

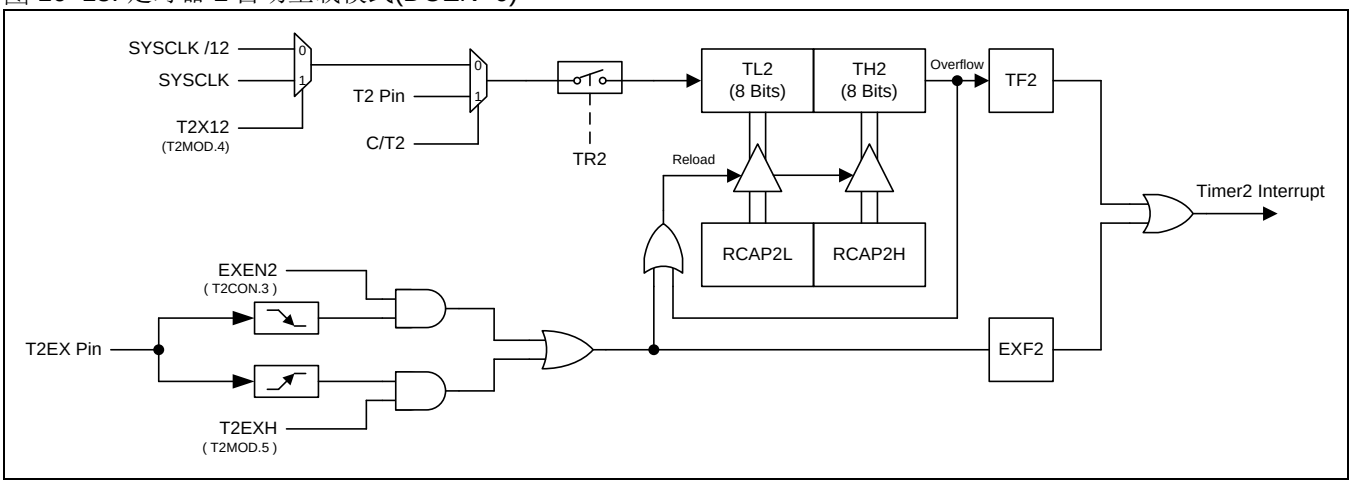
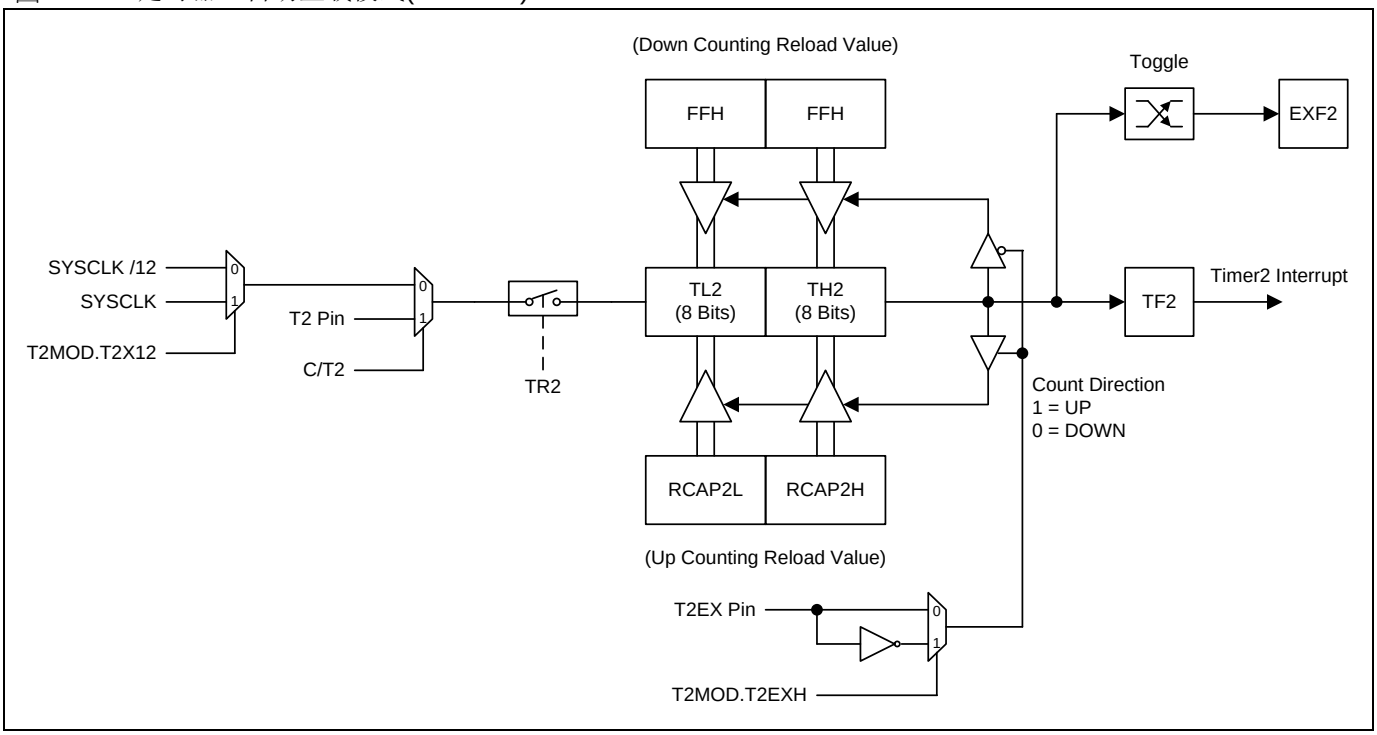


图 16–14 示 DCEN=1，使能定时器 2 向上或向下计数。这种模式下允许 T2EX 引脚控制计数方向。当 T2EX 的引脚为逻辑 1 时定时器 2 向上计数。定时器 2 在 0FFFFH 时溢出并置位 TF2 标志位，如果中断使能将产生中断。溢出也将引起 RCAP2L 和 RCAP2H 的 16 位值加载到定时器的寄存器 TL2 和 TH2。当 T2EX 的引脚为逻辑 0 时定时器 2 向下计数。当 TL2 和 TH2 和存储在 RCAP2L 和 RCAP2H 的值相等时将产生下溢。下溢将置位 TF2 标志位并将 0FFFFH 加载到定时器的寄存器 TL2 和 TH2。此模式下，T2EX 控制极性通过 T2EXH 来反转。

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 16–14. 定时器 2 自动重载模式(DCEN=1)



16.2.3. 波特率发生器模式(BRG)

T2CON 寄存器的 RCLK 和 TCLK 位允许串行口发送和接收波特率源可选择定时器 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，串行口可以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 16–15 所示定时器 2 在波特率发生器模式 UART 引擎产生 RX 和 TX 时钟（见图 17–6.）。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。

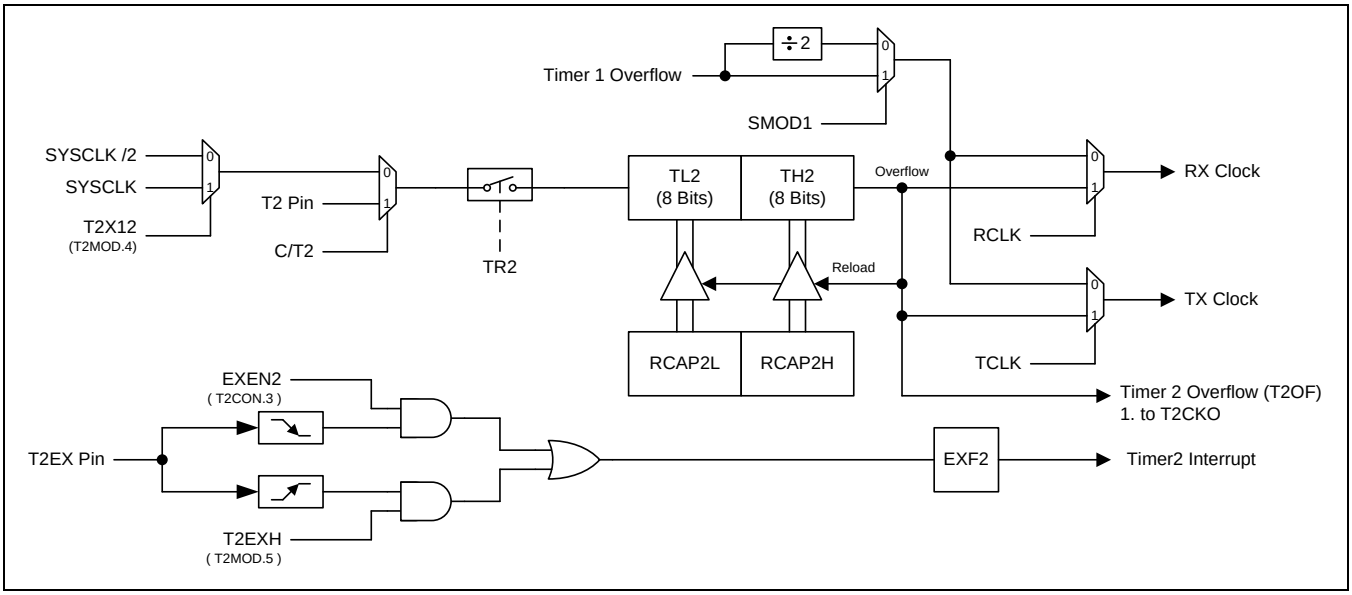
定时器 2 作为波特率发生器只有在 T2CON 寄存器的位 RCLK=1 和/或 TCLK=1 时有效。注意 TH2 翻转不会置位 TF2，也不会产生中断。因而，当定进器 2 在波特率发生器模式时定时器中断不需要禁止。如果 EXEN2(T2 外部中断使能位)置位，T2EX(定时器/计数器 2 触发输入)的负跳变将置位 EXF2(T2 外部标志位)，但是不会引起从(RCAP2H，RCAP2L)到(TH2，TL2 的)重载。因此，当定时器 2 作为波特率发生器时，如果需要的话，T2EX 也可以作为传统的外部中断。

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。在访问定时器 2 或 RCAP2 寄存器之前定时器必须关闭(清零 TR2)。

注意:

当定时器 2 用作波特率发生器时，参考章节"17.7.3 模式 1 和 3 波特率"波特率模式 1 和 3 获取波特率设定值

图 16–15. 定时器 2 波特率发生器模式



16.2.4. 定时器 2 可编程时钟输出

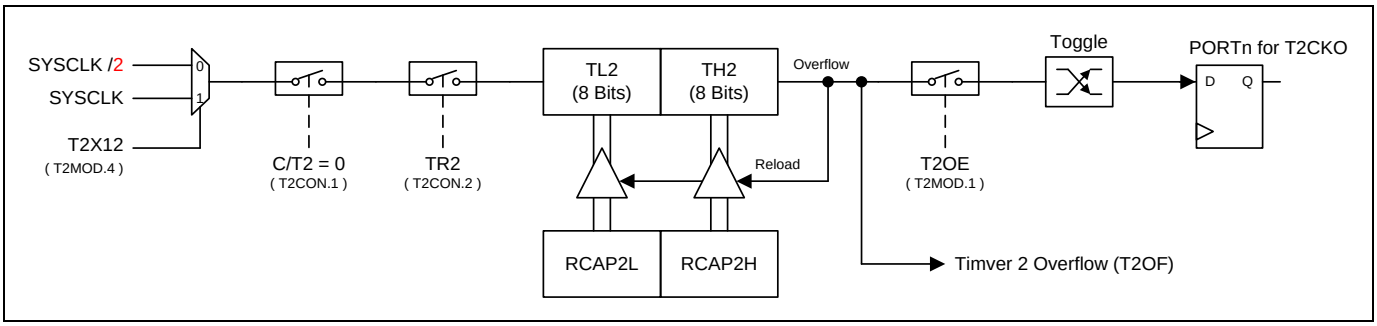
当 CP/RL2=0 和 T2OE=1 时定时器 2 工作在时钟输出模式。使用定时器 2 的可编程时钟输出模式，则从引脚 T2CKO(P1.0)输出占空比为 50%的时钟周期。输入时钟(SYSCLK/12 或 SYSCLK)使 16 位定时器(TH2, TL2)加一。定时器从载入值到溢出重复计数。一旦溢出，(RCAP2H, RCAP2L))的值被载入到(TH2, TL2)同时计数。图 16–16 给出了定时器 2 时钟输出频率计算公式，图 16–17 所示定时器 2 的时钟输出结构。

图 16–16. 定时器 2 时钟输出频率计算公式

$$\text{T2 Clock-out Frequency} = \frac{\text{SYSCLK Frequency}}{n \times (65536 - (\text{RCAP2H, RCAP2L}))} \quad \begin{array}{l} ; n=4, \text{ if T2X12=0} \\ ; n=2, \text{ if T2X12=1} \end{array}$$

注意:
(1) 定时器 2 溢出标志 TF2, 在定时器 2 溢出时置位但是不产生中断
(2) 当 SYSCLK=12MHz 及 T2X12=0, 定时器 2 可编程输出频率范围从 45.7Hz 到 3MHz.
(3) 当 SYSCLK=12MHz 及 T2X12=1, 定时器 2 可编程输出频率范围从 91.5Hz 到 6MHz.

图 16–17. 定时器 2 的时钟输出模式



定时器 2 时钟输出模式如何编程

- T2MOD 寄存器的位 T2X12 来选择定时器 2 时钟源
- T2MOD 寄存器的 T2OE 置位
- T2CON 寄存器的 C/T2 清零
- 从公式计算出 16 位自动加载值并输入到 RCAP2H 和 RCAP2L 寄存器
- 在 TH2 和 TL2 寄存器输入一个跟自动加载值相同的初始值
- 通过设置 T2CON 寄存器的 TR2 位启动定时器

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。

16.2.5. 定时器 2 寄存器

T2CON: 定时器 2 控制寄存器

SFR 页 = 0~F

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必须软件清零
1: 定时器2溢出TF2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
0: EXF2必须软件清零
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: 定时器1溢出用作接串行口接收时钟
1: 定时器2溢出用作接串行口模式1和3接收时钟

Bit 4: TCLK, 发送时钟控制位
0: 定时器1溢出用作接串行口发送时钟
1: 定时器2溢出用作接串行口模式1和3发送时钟

Bit 3: EXEN2, 定时器2外部使能位在T2EX引脚的负跳变
0: 定时器2忽略T2EX引脚的负跳变事件
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。如果定时器2配置为串行口0的时钟，T2EX保持外部信号侦测并产生 EXF2 旗标响应中断

Bit 2: TR2, 定时器2运行控制位
0: 定时器2停止运行
1: 定时器2开启运行

Bit 1: C/T2, 定时器或计数器选择位
0: 定时器2选择内部定时器功能
1: 定时器2选择外部事件计数器(下降沿触发)

Bit 0: CP/-RL2, 捕获/重载控制位
0: 如果EXEN2=1，定时器2溢出或T2EX上有负跳变时将产生自动重载
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溢出时强制重载

T2MOD: 定时器 2 模式寄存器

SFR 页 = 0~F

SFR 地址 = 0xC9 复位值= XX00-XX00

7	6	5	4	3	2	1	0
--	--	T2EXH	T2X12	--	--	T2OE	DCEN2
W	**W**	**R/W**	**R/W**	**W**	**W**	**R/W**	**R/W**

Bit 7~6: 保留。当 T2MOD 写入时，这两位软件必须写”0”

Bit 5: T2EXH, 定时器2检测T2EX 输入正跳变使能
0: 定时器2忽略T2EX引脚的正跳变事件

1: 如果定时器2没有用作串行口时钟，在T2EX的正跳变时捕获或加载并作为结果。如果定时器2配置为串行口0的时钟，T2EX保持外部信号侦测并产生 EXF2 旗标响应中断

Bit 4: T2X12, 定时器 2 时钟源选择
0: 当 T2CON.C/T2 = 0 选择 SYSCLK/12 作为捕获和自动重载模式定时器 2 的时钟源。当 T2CON.C/T2 = 0 如果在波特率发生器模式则 SYSCLK/2 作为定时器 2 的时钟源
1: 当 T2CON.C/T2 = 0 选择 SYSCLK 作为捕获和自动重载模式定时器 2 的时钟源。当 T2CON.C/T2 = 0 如果在波特率发生器模式则 SYSCLK 作为定时器 2 的时钟源

Bit 3~2: 保留。当 T2MOD 写入时，这两位软件必须写”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: DCEN2, 定时器 2 向下计数使能位
0: 定时器 2 总是保持向上计数
1: 定时器 2 向下计数使能

当DCEN2 =0时，定时器/计数器2的功能与标准8052单片机一样（总是向上计数）。当DCEN2 =1时，定时器/计数器2 能根据T2EX脚(P1.1)的逻辑电平向上或向下计数。定时器2操作模式如表 16–1

表 16–1. T2 模式

TR2	T2OE	RCLK + TCLK	CP/-RL2	DCEN2	模式
0	0	x	x	x	(关闭)
1	**1**	0	0	0	定时器 2 时钟输出(C/T2=0)
1	0	1	**0**	0	波特率发生器
1	**1**	1	**0**	0	时钟输出和波特率发生器 (C/T2=0)
1	0	0	1	0	16 位捕获
1	0	0	0	0	16 位自动重载 (仅向上计数)
1	0	0	0	**1**	16 位自动重载(向上或向下计数)

TL2: 定时器 2 低字节寄存器

SFR 页 = 0~F

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 高字节寄存器

SFR 页 = 0~F

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 捕获低字节寄存器

SFR 页 = 0~F

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 捕获高字节寄存器

SFR 页 = 0~F

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**

16.3. 定时器示例代码

(1). 规定功能:系统时钟 SYSCLK = ILRCO 时定时器 T0 以 320Hz 的频率唤醒空闲模式

汇编语言代码范例:

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

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

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

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

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

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

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

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

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

   SETB   TR0                       ; 启动定时器 0 运行

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

C 语言代码范例:

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

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

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

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

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

    IFD &= ~HSE;                    //当系统时钟 SYSCLK ≤ 6MHz 为了省电禁止 HSE
    page_p_sfr_write();             // 写数据到 PCON2

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

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

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

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

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

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

(2). 规定功能: SYSCLK/48 时钟源设置定时器 0 的时钟输出

汇编语言代码范例:

```
    CLR     TR0                     ;

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

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

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

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

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

C 语言代码范例:

```
    TR0 = 0;

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

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

    TH0 = TL0 = 0xFF;

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

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

(3). 规定功能: SYSCLK 时钟源设置定时器 1 的时钟输出

汇编语言代码范例:

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

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

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

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

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

C 语言代码范例:

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

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

TH1 = TL1 = 0xFF;

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

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

17. 串行口 0 (UART0)

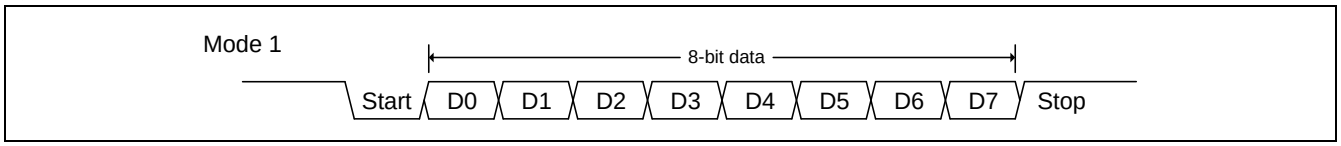
MA82G5BXX 支持一个全双工的串行口，意思是可以同时发送和接收数据。它有一个接收缓冲，意味着在前一个接收到的字节没有从寄存器读出前，就可以开始接收第二个字节。但是，如果第一个字节在第二个字节接收完成前仍然没有被读出，则其中的一个字节将会丢失。串行口的接收和发送寄存器都通过特殊寄存器 S0BUF 来访问。写到 S0BUF 加载到传送寄存器，当从 S0BUF 读时是一个物理上独立分离的接收寄存器。

串行口可以工作在 **5** 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。UART0 的模式 4 支持 SPI 主机工作，数率设置跟模式 0 一样。

模式 0:8 位数据(低位先出)通过 RXD0 传送和接收。TXD0 总是作为输出移位时钟。波特率可通过 S0CFG 寄存器的 URM0X3 位选择为系统时钟频率的 1/12 或 1/4。**MA82G5BXX** 串行口模式 0 的时钟极性也可以软件选择。在串行数据移入或移出之前它由 P3.1 的状态决定。图 17–4 和图 17–5 所示模式 0 的时钟极性波形。

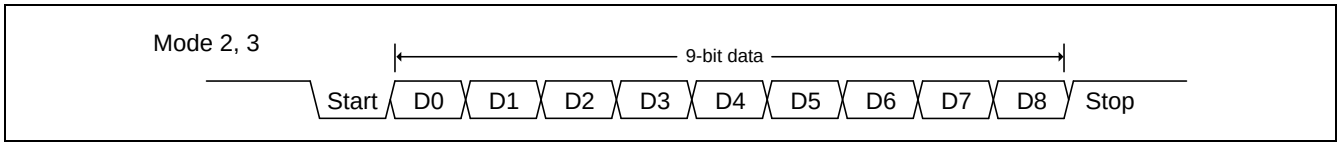
模式 1: 10 位通过 TXD0 传送或通过 RXD0 接收，数据帧包括一个起始位(0)，8 个数据位(低位优先)，和一个停止位(1)（如图 17–1 所示）。在接收时，停止位进入到专用寄存器(S0CON)的 RB80。波特率是可变的。

图 17–1. 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) （如图 17–2 所示）。在传送时，第 9 个数据位(TB80 在 S0CON 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 S0CON 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 SYSCLK/64 或 SYSCLK/32。

图 17–2. 模式 2, 3 数据帧



模式 3: 除了波特率是可变之外模式 3 与模式 2 一样。

在四种模式中，使用S0BUF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。

17.1. 串行口 0 模式 0

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S0CFG寄存器中的URM0X3选择为系统时钟的 1/12或1/4。图 17–3显示了串口模式 0 的简化功能框图。

使用 **S0BUF** 作为一个目的寄存器可通过任何指令来启动传输。“写到 **S0BUF** ”信号触发 UART0 引擎开始发送。**S0BUF** 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。模式 0 发送时序见图 17–4。

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 TXD0（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。模式 0 接收时序见图 17–5。

当 TXD0 被分配在 P3.1，在串行传送移位之前，时钟极性可以通过 P3.1 数据锁存的软件设置来选择。如果 P3.1 设置为逻辑高，时钟极性跟标准 8051 一样。如果 P3.1 数据锁存为逻辑低，时钟极性跟标准 8051 UART 模式 0 相反。

图 17–3. 串行口 0 模式 0

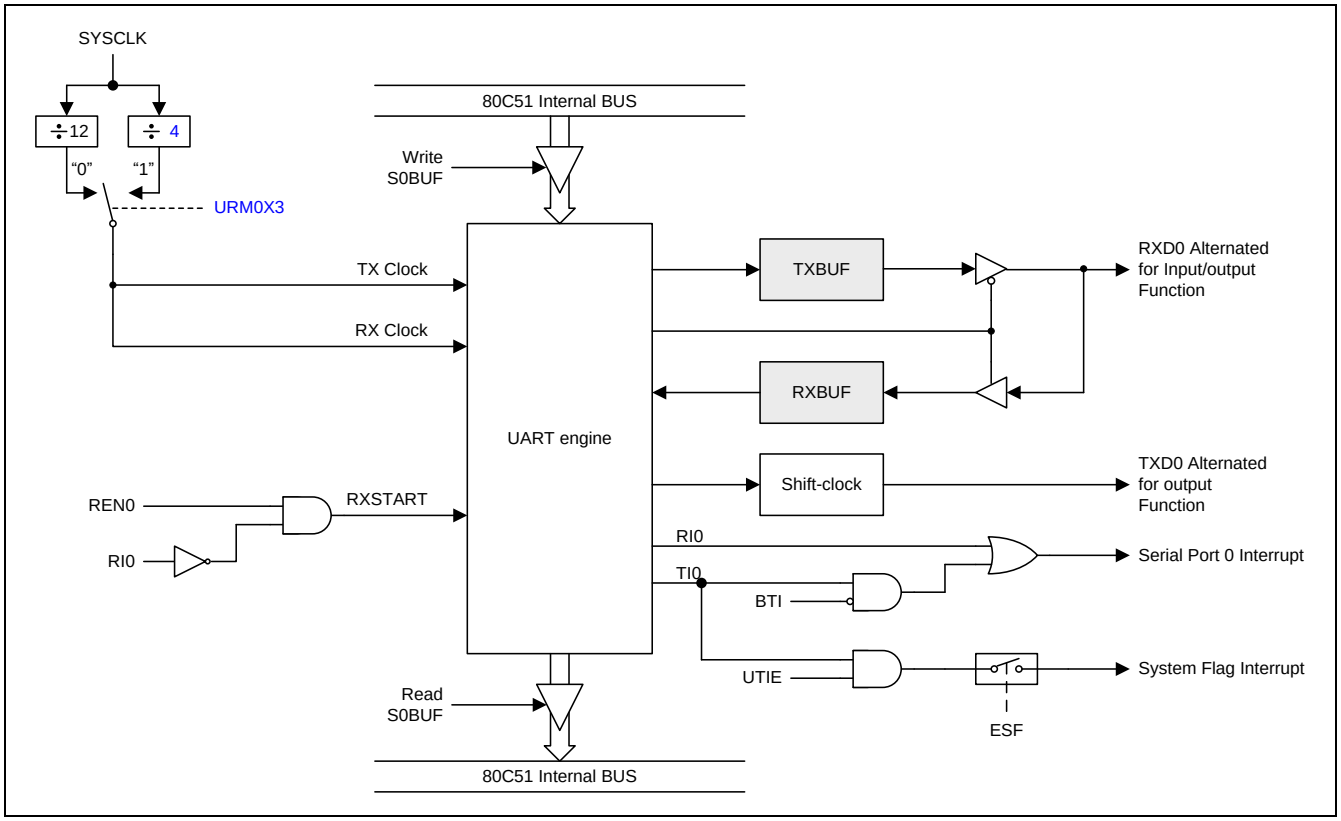


图 17–4. 模式 0 发送波形

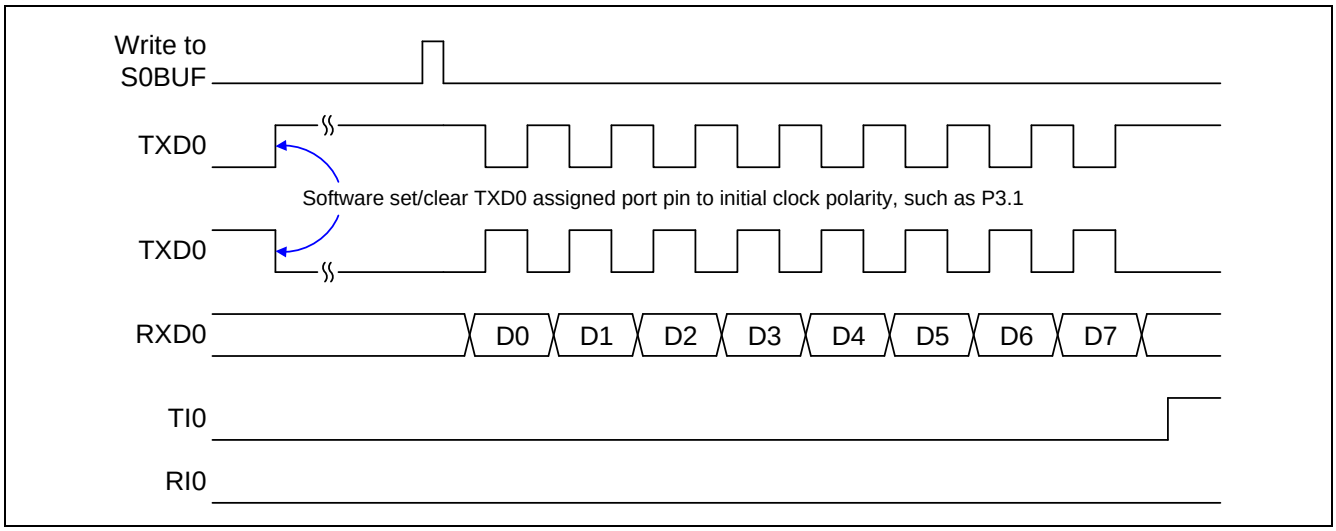
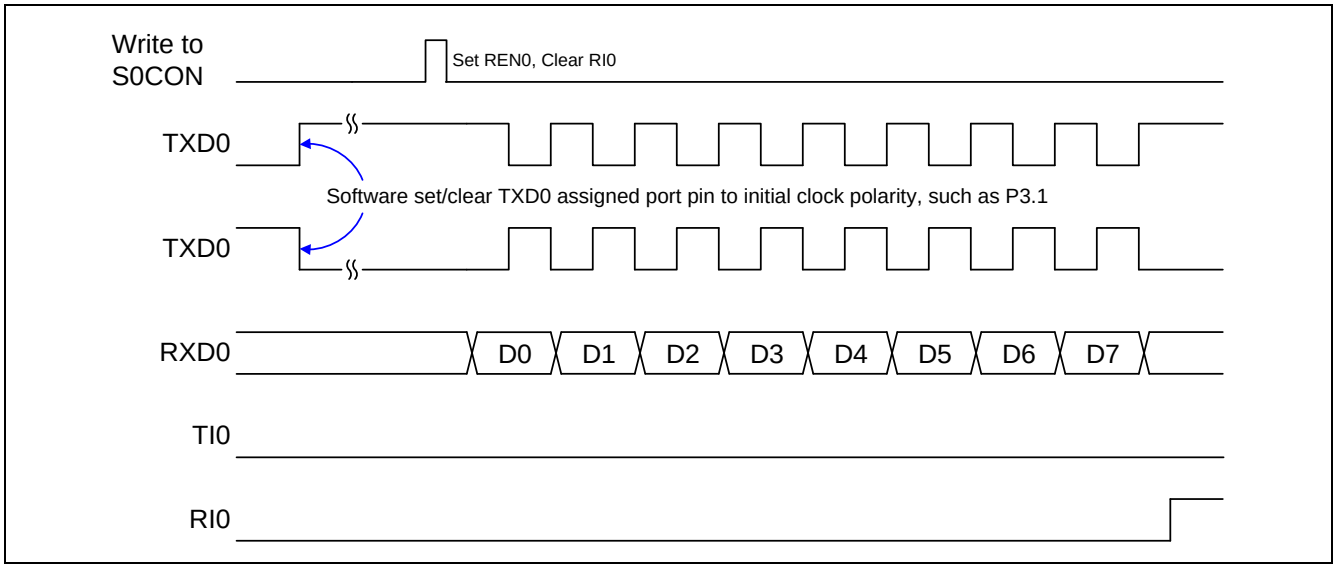


图 17–5. 模式 0 接收波形



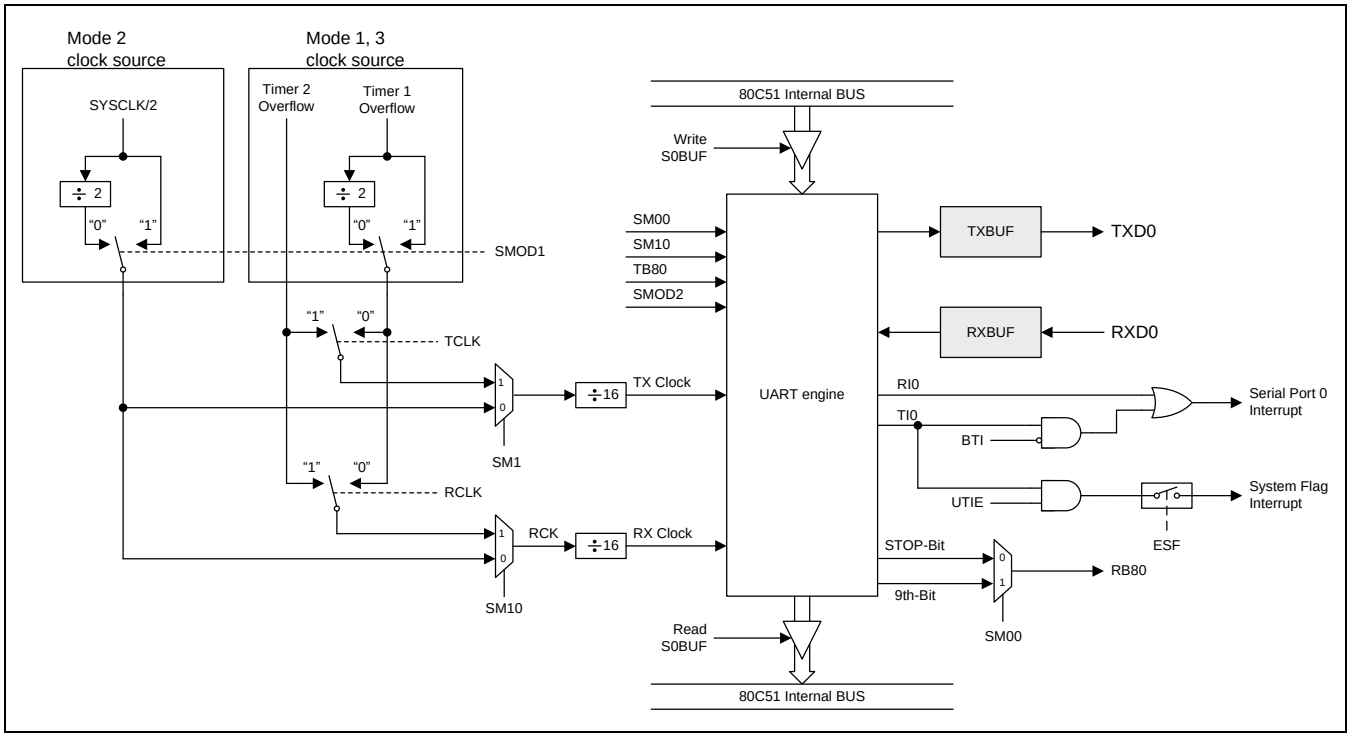
17.2. 串行口 0 模式 1

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 S0CON 的 RB80，波特率由定时器 1 或定时器 2 的溢出速率来决定。模式 1 数据帧时序如图 17–1 所示并且模式 1 的简化功能框图如图 17–6 所示

使用 **S0BUF** 作为目的寄存器的任何指令来启动传输。写到 **S0BUF** 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。**S0BUF** 中的数据从 TXD0 引脚串行输出，数据帧如图 17–1 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 S0CON 寄存器的 RB80。

图 17–6. 串行口模式 1, 2, 3



17.3. 串行口 0 模式 2 和模式 3

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 S0CON 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。

模式 2 和 3 数据帧如图 17–2 所示，简化功能框图如图 17–6 所示。接收部分和模式 1 相同。与模式 1 传送部分不同的仅仅是传送移位寄存器的第 9 位。

写到 S0BUF 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。S0BUF 中的数据从 TXD0 引脚串行输出，数据帧如图 17–2 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。

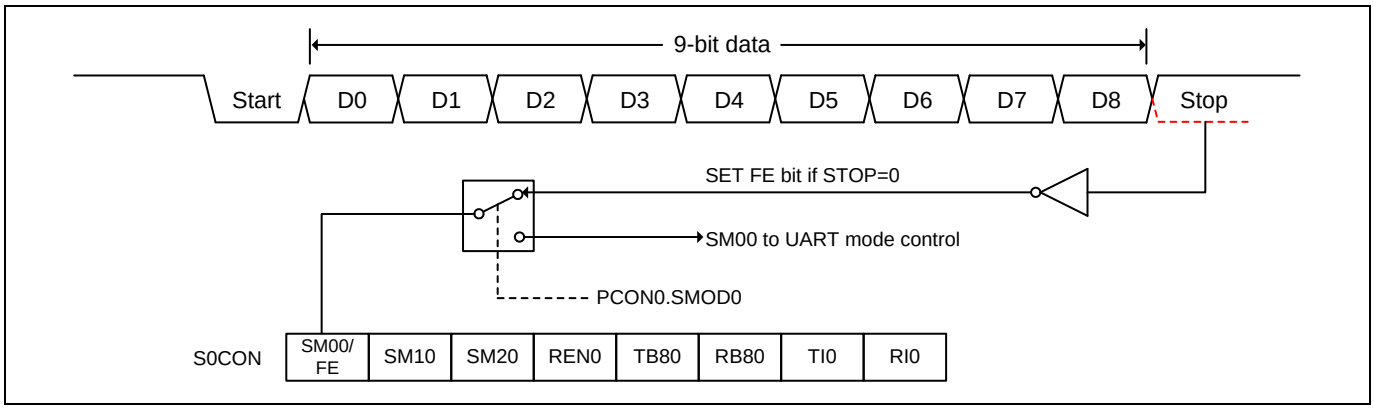
当串行口 0 控制器在 RCK 采样时钟下检测到在 RXD0 有 1 到 0 跳变的起始位时接收开始。在 RXD0 引脚上的数据将被串行口 0 的位侦查器采样。当收数据接收完后，硬件置位 RI0 表示接收结束并把第 9 位加载到 S0CON 寄存器的 RB80。

在四种模式中，使用 S0BUF 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。

17.4. 帧错误侦测

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S0CON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S0CON.7究竟代表哪个标志，如果SMOD0位（PCON0.6）置位则S0CON.7就是FE标志，SMOD0位清零则S0CON.7就是SM00标志。当S0CON.7代表FE时，只能软件清零。参考图 17–7。

图 17–7. UART0 帧错误侦测



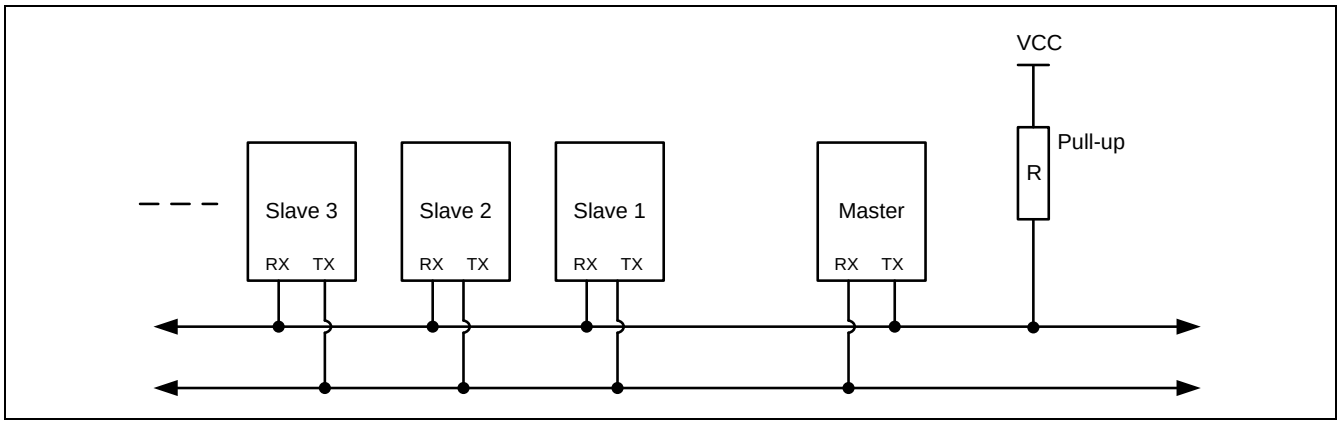
17.5. 多处理器通讯

模式 2 和 3 在用作多处理器通讯时有特殊的规定如图 17–8 所示。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 S0CON 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 17–8. UART0 多处理器通讯



17.6. 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 S0CON 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 17–9。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：

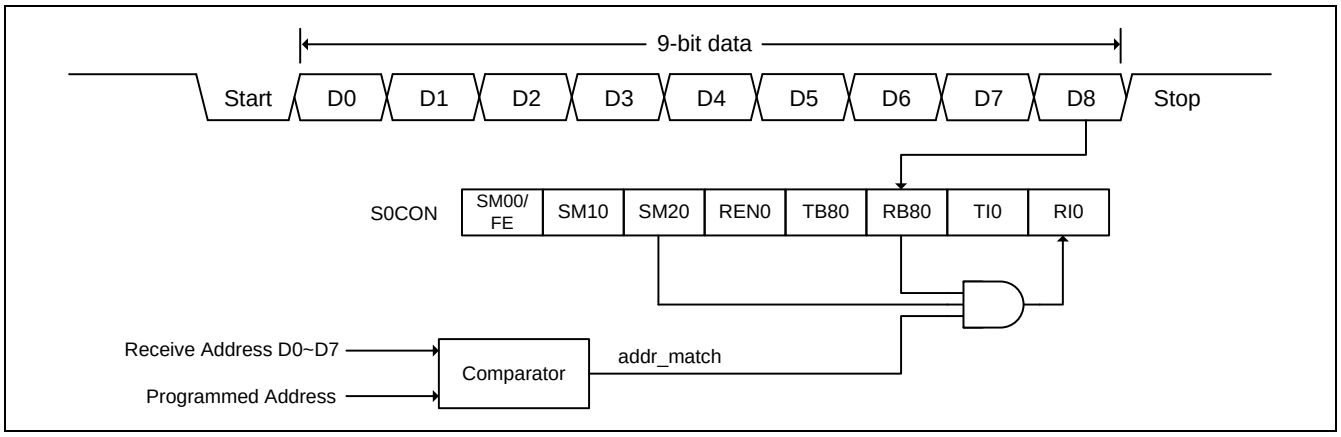
从机 0	从机 1	从机 2
SADDR = 1100 0000	SADDR = 1110 0000	SADDR = 1110 0000
SADEN = 1111 1001	SADEN = 1111 1010	SADEN = 1111 1100
Given = 1100 0XX0	Given = 1110 0X0X	Given = 1110 00XX

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的创建都是通过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 17–9. 自动地址识别



注意:

(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

17.7. 波特率设置

位T2X12 (T2MOD.4), T1X12 (AUXR2.3), URM0X3 (S0CFG.5)和SMOD2 (S0CFG.6)提供一个新的波特率选项设置，如下所列：

17.7.1. 模式 0 波特率

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

注意:

如果 URM0X3=0, 波特率公式跟标准 8051 一样

17.7.2. 模式 2 波特率

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

注意:

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。表 17–1 定义了模式 2 波特率发生器由 SMOD2 因数决定的波特率设置。

表 17–1. SMOD2 在模式 2 中应用标准

SMOD2	SMOD1	波特率	备注	推荐的最大接收误差 (%)
0	0	缺省波特率	标准功能	± 3%
0	1	双倍波特率	标准功能	± 3%
1	0	双倍波特率 **X2**	增强型功能	± 2%
1	1	双倍波特率 **X4**	增强型功能	± 1%

17.7.3. 模式 1 和 3 波特率

16.7.3.1 使用定时器 1 作为波特率发生器

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

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

注意:

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。表 17–2 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。

表 17–2. SMOD2 在模式 1 & 3 用定时器 1 时的应用标准

SMOD2	SMOD1	波特率	备注	推荐的最大接收误差 (%)
0	0	缺省波特率	标准功能	± 3%
0	1	双倍波特率	标准功能	± 3%
1	0	双倍波特率 **X2**	增强型功能	± 2%
1	1	双倍波特率 **X4**	增强型功能	± 1%

表 17–3 ~ 表 17–10 列出 8 位自动加载模式的定时器 1 中各种常用的波特率和怎样从中获得。

表 17–3. 定时器 1 产生常用的波特率@ F_{SYSCLK}=11.0592MHz

波特率	**TH1**, 重载值					
	T1X12=0 & SMOD2=0			**T1X12=1 & SMOD2=0**		
	SMOD1=0	SMOD1=1	误差	SMOD1=0	SMOD1=1	误差
1200	232	208	0.0%	--	--	--
2400	244	232	0.0%	112	--	0.0%
4800	250	244	0.0%	184	112	0.0%
9600	253	250	0.0%	220	184	0.0%
14400	254	252	0.0%	232	208	0.0%
19200	--	253	0.0%	238	220	0.0%
28800	255	254	0.0%	244	232	0.0%
38400	--	--	--	247	238	0.0%
57600	--	255	0.0%	250	244	0.0%
115200	--	--	--	253	250	0.0%
230400	--	--	--	--	253	0.0%

表 17–4. 定时器 1 产生高波特率@ F_{SYSCLK} =11.0592MHz

波特率	**TH1**, 重载值					
	T1X12=0 & SMOD2=1			**T1X12=1 & SMOD2=1**		
	SMOD1=0	SMOD1=1	误差	SMOD1=0	SMOD1=1	误差
230.4K	--	255	0.0%	250	244	0.0%
460.8K	--	--	--	253	250	0.0%
691.2K	--	--	--	254	252	0.0%
921.6K	--	--	--	--	253	0.0%
1.3824M	--	--	--	255	254	0.0%
2.7648M	--	--	--	--	255	0.0%

表 17–5. 定时器 1 产生常用的波特率@ F_{SYSCLK}=22.1184MHz

波特率	**TH1**, 重载值					
	T1X12=0 & SMOD2=0			**T1X12=1 & SMOD2=0**		
	SMOD1=0	SMOD1=1	误差	SMOD1=0	SMOD1=1	误差
1200	208	160	0.0%	--	--	--
2400	232	208	0.0%	--	--	0.0%
4800	244	232	0.0%	112	--	0.0%
9600	250	244	0.0%	184	112	0.0%
14400	252	248	0.0%	208	160	0.0%
19200	253	250	0.0%	220	184	0.0%
28800	254	252	0.0%	232	208	0.0%
38400	--	253	0.0%	238	220	0.0%
57600	255	254	0.0%	244	232	0.0%
115200	--	255	0.0%	250	244	0.0%
230400	--	--	--	253	250	0.0%
460800	--	--	--	--	253	0.0%

表 17–6. 定时器 1 产生高波特率@ F_{SYSCLK}=22.1184MHz

波特率	**TH1**, 重载值					
	T1X12=0 & SMOD2=1			**T1X12=1 & SMOD2=1**		
	SMOD1=0	SMOD1=1	误差	SMOD1=0	SMOD1=1	误差
460.8K	--	255	0.0%	250	244	0.0%
691.2K	--	--	--	252	248	0.0%
921.6K	--	--	--	253	250	0.0%
1.3824M	--	--	--	254	252	0.0%
1.8432M				--	253	0.0%
2.7648M	--	--	--	255	254	0.0%
5.5296M	--	--	--	--	255	0.0%

表 17–7. 定时器 1 产生常用的波特率@ F_{SYSCLK}=12.0MHz

波特率	**TH1**, 重载值					
	T1X12=0 & SMOD2=0			**T1X12=1 & SMOD2=0**		
	SMOD1=0	SMOD1=1	误差	SMOD1=0	SMOD1=1	误差
1200	230	204	0.16%	--	--	--
2400	243	230	0.16%	100	--	0.16%
4800	--	243	0.16%	178	100	0.16%
9600	--	--	--	217	178	0.16%
14400	--	--	--	230	204	0.16%
19200	--	--	--	--	217	0.16%
28800	--	--	--	243	230	0.16%
38400	--	--	--	246	236	2.34%
57600	--	--	--	--	243	0.16%
115200	--	--	--	--	--	--

表 17–8. 定时器 1 产生高波特率@ F_{SYSCLK}=12.0MHz

波特率	**TH1**, 重载值					
	T1X12=0 & SMOD2=1			**T1X12=1 & SMOD2=1**		
	SMOD1=0	SMOD1=1	误差	SMOD1=0	SMOD1=1	误差
115.2K	--	--	--	243	230	0.16%
230.4K	--	--	--	--	243	0.16%
460.8K	--	--	--	--	--	--

表 17–9. 定时器 1 产生常用的波特率@ F_{SYSCLK}=24.0MHz

波特率	**TH1**, 重载值					
	T1X12=0 & SMOD2=0			**T1X12=1 & SMOD2=0**		
	SMOD1=0	SMOD1=1	误差	SMOD1=0	SMOD1=1	误差
1200	204	152	0.16%	--	--	--
2400	230	204	0.16%	--	--	--
4800	243	230	0.16%	100	--	0.16%
9600	--	243	0.16%	178	100	0.16%
14400	--	--	--	204	152	0.16%
19200	--	--	--	217	178	0.16%
28800	--	--	--	230	204	0.16%
38400	--	--	--	--	217	0.16%
57600	--	--	--	243	230	0.16%
115200	--	--	--	--	243	0.16%

表 17–10. 定时器 1 产生高波特率@ F_{SYSCLK}=24.0MHz

波特率	**TH1**, 重载值					
	T1X12=0 & SMOD2=1			**T1X12=1 & SMOD2=1**		
	SMOD1=0	SMOD1=1	误差	SMOD1=0	SMOD1=1	误差
230.4K	--	--	--	243	230	0.16%
460.8K	--	--	--	--	243	0.16%
691.2K	--	--	--	--	--	--
921.6K	--	--	--	--	--	--

16.7.3.2 使用定时器 2 作为波特率发生器

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：

$$\text{Mode 1, 3 Baud Rate} = \frac{2^{\text{SMOD2 X (SMOD1 + 1)}} \times F_{SYSCLK}}{32 \times (65536 - (RCAP2H, RCAP2L))} \quad ; \text{T2X12=0}$$

$$\text{or} = \frac{2^{\text{SMOD2 X (SMOD1 + 1)}} \times F_{SYSCLK}}{16 \times (65536 - (RCAP2H, RCAP2L))} \quad ; \text{T2X12=1}$$

注意:

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。表 17–11 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。

表 17–11. SMOD2 在定时器 2 模式 1 & 3 中应用条件

SMOD2	SMOD1	波特率	备注	推荐的最大接收误差 (%)
0	X	缺省波特率	标准功能	± 3%
1	0	双倍波特率	增强型功能	± 3%
1	1	双倍波特率 **X2**	增强型功能	± 2%

表 17–12 ~ 表 17–19 列出定时器 2 中波特率产生器模式中的各种通用的波特率和怎样获得

表 17–12. 定时器 2 产生普通的波特率@ F_{SYSCLK}=11.0592MHz

波特率	**[RCAP2H, RCAP2L]**, 重载值					
	T2X12=0 & SMOD2=0			**T2X12=1 & SMOD2=0**		
	SMOD1=0	SMOD1=1	误差	SMOD1=0	SMOD1=1	误差
1200	65248	65248	0.0%	64960	64960	0.0%
2400	65392	65392	0.0%	65248	65248	0.0%
4800	65464	65464	0.0%	65392	65392	0.0%
9600	65500	65500	0.0%	65464	65464	0.0%
14400	65512	65512	0.0%	65488	65488	0.0%
19200	65518	65518	0.0%	65500	65500	0.0%
28800	65524	65524	0.0%	65512	65512	0.0%
38400	65527	65527	0.0%	65518	65518	0.0%
57600	65530	65530	0.0%	65524	65524	0.0%
115200	65533	65533	0.0%	65530	65530	0.0%
230400	--	--	--	65533	65533	0.0%

表 17–13. 定时器 2 产生高的波特率@ F_{SYSCLK}=11.0592MHz

波特率	**[RCAP2H, RCAP2L]**, 重载值					
	T2X12=0 & SMOD2=1			**T2X12=1 & SMOD2=1**		
	SMOD1=0	SMOD1=1	误差	SMOD1=0	SMOD1=1	误差
230.4K	65533	65530	0.0%	65530	65524	0.0%
460.8K	--	65533	0.0%	65533	65530	0.0%
691.2K	65535	65534	0.0%	65534	65532	0.0%
921.6K	--	--	--	--	65533	0.0%
1.3824M	--	65535	0.0%	65535	65534	0.0%
2.7648M	--	--	--	--	65535	0.0%

表 17–14. 定时器 2 产生普通的波特率@ F_{SYSCLK}=22.1184MHz

波特率	**[RCAP2H, RCAP2L]**, 重载值					
	T2X12=0 & SMOD2=0			**T2X12=1 & SMOD2=0**		
	SMOD1=0	SMOD1=1	误差	SMOD1=0	SMOD1=1	误差
1200	64960	64960	0.0%	64384	64384	0.0%
2400	65248	65248	0.0%	64960	64960	0.0%
4800	65392	65392	0.0%	65248	65248	0.0%
9600	65464	65464	0.0%	65392	65392	0.0%
14400	65488	65488	0.0%	65440	65440	0.0%
19200	65500	65500	0.0%	65464	65464	0.0%
28800	65512	65512	0.0%	65488	65488	0.0%
38400	65518	65518	0.0%	65500	65500	0.0%
57600	65524	65524	0.0%	65512	65512	0.0%
115200	65530	65530	0.0%	65524	65524	0.0%
230400	65533	65533	0.0%	65530	65530	0.0%
460800	--	--	--	65533	65533	0.0%

表 17–15. 定时器 2 产生高的波特率@ F_{SYSCLK}=22.1184MHz

波特率	**[RCAP2H, RCAP2L]**, 重载值					
	T2X12=0 & SMOD2=1			**T2X12=1 & SMOD2=1**		
	SMOD1=0	SMOD1=1	误差	SMOD1=0	SMOD1=1	误差
460.8K	65533	65530	0.0%	65530	65524	0.0%
691.2K	65534	65532	0.0%	65532	65528	0.0%
921.6K	--	65533	0.0%	65533	65530	0.0%
1.3824M	65535	65534	0.0%	65534	65532	0.0%
1.8432M	--	--	--	--	65533	0.0%
2.7648M	--	65535	0.0%	65535	65534	0.0%
5.5296M	--	--	--	--	65535	0.0%

表 17–16. 定时器 2 产生普通的波特率@ F_{SYSCLK}=12.0MHz

波特率	**[RCAP2H, RCAP2L]**, 重载值					
	T2X12=0 & SMOD2=0			**T2X12=1 & SMOD2=0**		
	SMOD1=0	SMOD1=1	误差	SMOD1=0	SMOD1=1	误差
1200	65224	65224	0.16%	64912	64912	0.16%
2400	65380	65380	0.16%	65224	65224	0.16%
4800	65458	65458	0.16%	65380	65380	0.16%
9600	65497	65497	0.16%	65458	65458	0.16%
14400	65510	65510	0.16%	65484	65484	0.16%
19200	65516	65516	2.34%	65497	65497	0.16%
28800	65523	65523	0.16%	65510	65510	0.16%
38400	--	--	--	65516	65516	2.34%
57600	--	--	--	65523	65523	0.16%
115200	--	--	--	--	--	--

表 17–17. 定时器 2 产生高的波特率@ F_{SYSCLK}=12.0MHz

波特率	**[RCAP2H, RCAP2L]**, 重载值					
	T2X12=0 & SMOD2=1			**T2X12=1 & SMOD2=1**		
	SMOD1=0	SMOD1=1	误差	SMOD1=0	SMOD1=1	误差
115.2K	--	65523	0.16%	65523	65510	0.16%
230.4K	--	--	--	--	65523	0.16%
460.8K	--	--	--	--	--	--

表 17–18. 定时器 2 产生普通的波特率@ F_{SYSCLK}=24.0MHz

波特率	**[RCAP2H, RCAP2L]**, 重载值					
	T2X12=0 & SMOD2=0			**T2X12=1 & SMOD2=0**		
	SMOD1=0	SMOD1=1	误差	SMOD1=0	SMOD1=1	误差
1200	64912	64912	0.16%	64288	64288	0.16%
2400	65224	65224	0.16%	64912	64912	0.16%
4800	65380	65380	0.16%	65224	65224	0.16%
9600	65458	65458	0.16%	65380	65380	0.16%
14400	65484	65484	0.16%	65432	65432	0.16%
19200	65497	65497	0.16%	65458	65458	0.16%
28800	65510	65510	0.16%	65484	65484	0.16%
38400	65516	65516	2.34%	65497	65497	0.16%
57600	65523	65523	0.16%	65510	65510	0.16%
115200	--	--	--	65523	65523	0.16%

表 17–19. 定时器 2 产生高的波特率@ F_{SYSCLK}=24.0MHz

波特率	**[RCAP2H, RCAP2L]**, 重载值					
	T2X12=0 & SMOD2=1			**T2X12=1 & SMOD2=1**		
	SMOD1=0	SMOD1=1	误差	SMOD1=0	SMOD1=1	误差
230.4K	--	65523	0.16%	65523	65510	0.16%
460.8K	--	--	--	--	65523	0.16%
691.2K	--	--	--	--	--	--
921.6K	--	--	--	--	--	--

16.7.3.3 使用串行口 1 波特率定时器作为波特率发生器

MA82G5BXX 第二串行口 UART (S1)有一个独立的波特率发生器。串行口 0 可以置位 URTS (S0CFG.7)来选择 S1BRT 作为模式 1 和 3 的定时器源。详细的描述见“18.6 串行口 0(S0)的波特率定时器来自串行口 1(S1)”

17.8. 串行口 0 模式 4 (SPI 主机)

MA82G5BXX 串行口嵌入了一个额外的模式 4 支持 SPI 主机引擎。模式 4 由 SM3, SM0 和 SM1 选择。表 17–20 展示了 **MA82G5BXX** 的串行口模式定义。

表 17–20.串行口 0 模式选择

SM30	SM00	SM10	模式	描述	波特率
0	0	0	0	移位寄存器	SYSCLK/12 or SYSCLK/4
0	0	1	1	8-bit UART	可变
0	1	0	2	9-bit UART	SYSCLK/64, /32
0	1	1	3	9-bit UART	可变
1	0	0	4	**SPI 主机**	SYSCLK/12 or SYSCLK/4
1	0	1	5	保留	保留
1	1	0	6	保留	保留
1	1	1	7	保留	保留

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

MA82G5BXX 的 SPI 主机使用 TXD 作为 SPICLK, RXD 作为 MOSI,以及 S0MI 作为 MISO。而 nSS 由 MCU 软件选择在其它端口引脚。图 17–10 展示了 SPI 连接。他支持多从机通讯架构见图 17–11.

图 17–10.串行口 0 模式 4,单主机和单从机架构(n = 0)

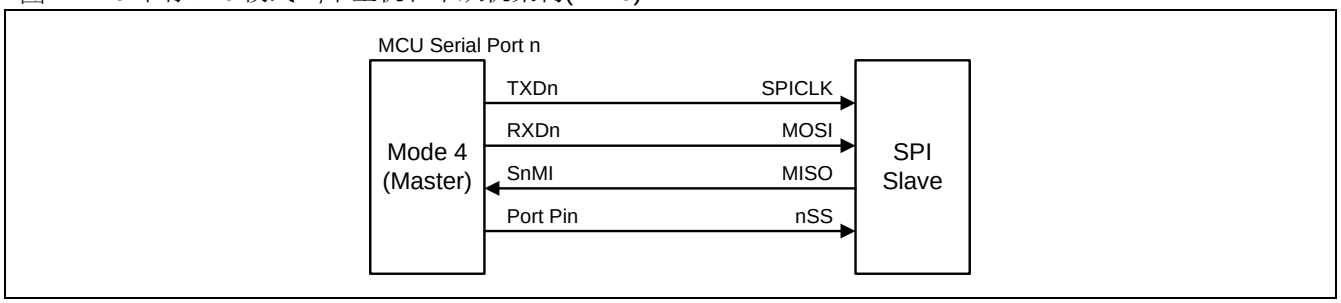
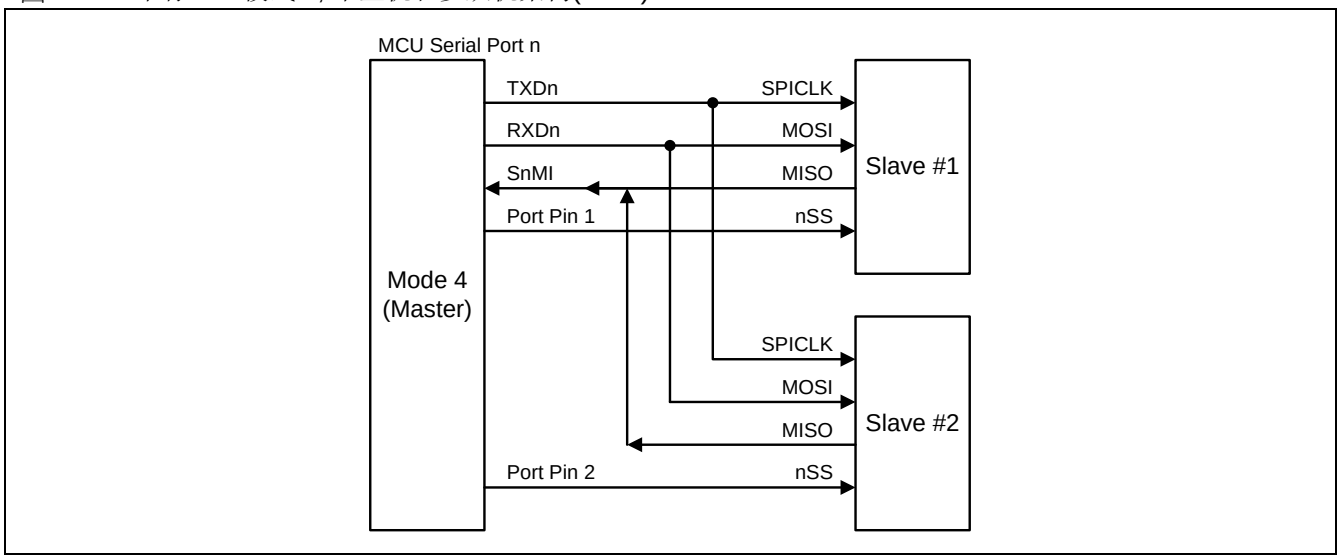


图 17–11.串行口 0 模式 4, 单主机和多从机架构(n = 0)



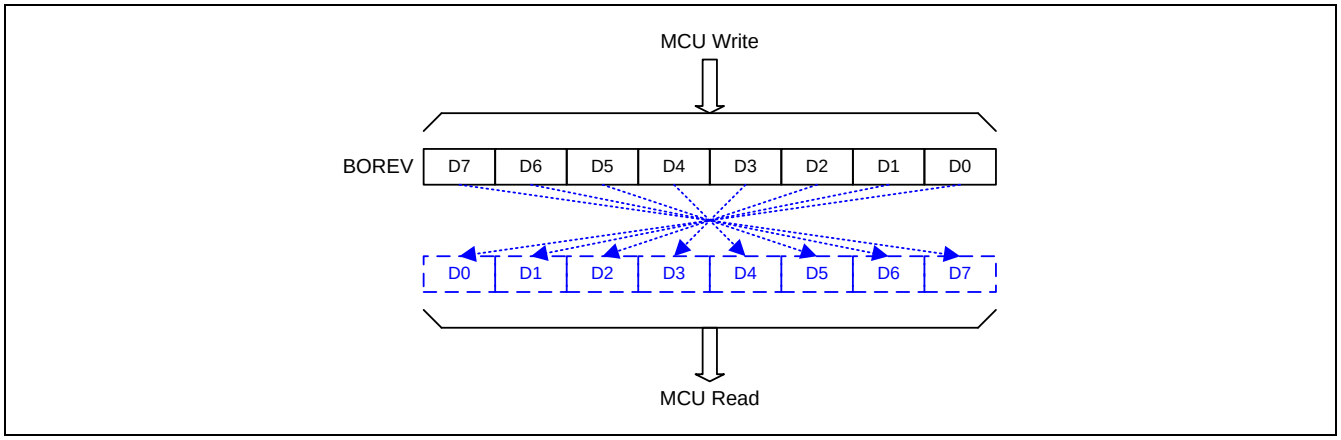
SPI 主机能满足笙泉 MA82/84 系列 MCU（由 CPOL, CPHA 和 DORD 选择）的全功能 SPI 模块的传输。在 CPOL 和 CPHA 条件下, **MA82G5BXX** 很容易初始化 SPI 的时钟(TXD0, P3.1/P4.5)极性去适合他们使用 。表 17–21 展示了串行口模式 4 的 4 个 SPI 工作模式。

表 17–21.串行口 0 模式 4 的 SPI 模式结构

SPI 模式	CPOL	CPHA	当 TXD0 在 P3.1 脚 **MA82G5BXX** 的结构
0	0	0	清除 P3.1 为“0”
1	0	1	清除 P3.1 为“0”
2	1	0	设置 P3.1 为“1”
3	1	1	设置 P3.1 为“1”

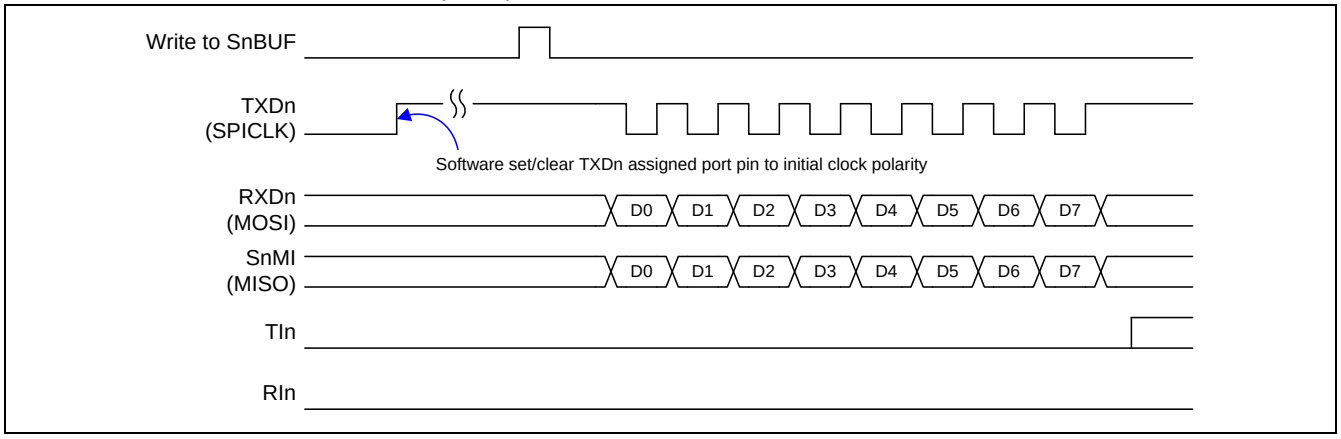
SPI 系列传输的位序控制(DORD)，**MA82G5BXX** 提供了一个软件编程可以相反的位序控制（SFR, BOREV）。 在 MCU 写一个数据格式（MSB）到 BOREV ， MCU 通过读 BOREV 得到一个数据格式（LSB）回来。SPI 主机引擎在串行口模式 4 与串行口模式 0 一样都是（LSB）传输。为了支持 SPI（MSB）移位传输, MCU 必须使用（BOREV）写/读取操作来翻转 SPI 输入/输出传送的数据位序。图 17–12 展示了 BOREV 结构。

图 17–12. SFR BOREV 读取/写结构



通过指令使用 SBUF 作为目标寄存器初始化传送。“写数据到 SBUF”触发 UART 引擎开始传送。SBUF 的数据被移位到作为 MOSI 串口数据的 RXD 引脚。SPI 移位时钟由作为 SPICLK 输出的 TXD 引脚输出。在 8 个移位时钟的上升沿之后，TI 被硬件声明传送结束。同时 S0MI 引脚的内容被采样并且移位到移位寄存器。然后“读取 SBUF”能获取 SPI 的移入数据。 图 17–13 展示了模式 4 传送波形。RI 在模式 4 不被声明。

图 17–13. 串行口 0 模式 4 传送波形(n = 0)



17.9. 串行口 0 寄存器

串行口的四种操作模式除波特率的设定之外都与标准的 8051 相同。此三个寄存器 PCON, AUXR2 和 S0CFG 是与波特率的设定有关

S0CON: 串行口 0 控制寄存器

SFR 页 = **0**

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 必须置位才能访问 FE 位
0: FE 位不会被有效的帧清零，它应当被软件清零
1: 当检测到一个无效的停止位时，该位被接收器置位

Bit 7: 串行口 0 模式位 0， (SMOD0 必须 = 0 才能访问位 SM0)
Bit 6: 串行口 0 模式位 1

SM30	SM00	SM10	模式	描述	波特率
0	0	0	0	移位寄存器	SYSCLK/12 or SYSCLK/4
0	0	1	1	8-bit UART	可变的
0	1	0	2	9-bit UART	SYSCLK/64, /32, /16 or /8
0	1	1	3	9-bit UART	可变的
1	0	0	4	SPI 主机	SYSCLK/12 or SYSCLK/4
1	0	1	5	保留	保留
1	1	0	6	保留	保留
1	1	1	7	保留	保留

Bit 5: 串行口 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: 软件置位使能接收

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

S0BUF: 串行口 0 缓冲寄存器

SFR 页 = **0**

SFR Address = 0x99 复位值 = XXXX-XXXX

7	6	5	4	3	2	1	0
S0BUF.7	S0BUF.6	S0BUF.5	S0BUF.4	S0BUF.3	S0BUF.2	S0BUF.1	S0BUF.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 = 0~F

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 = 0~F

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 = 0~F

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

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

Bit 7: SMOD1, 双倍波特率控制位
0: 禁止 UART 双倍波特率
1: 使能 UART 双倍波特率(模式 1, 2,或 3)

Bit 6: SMOD0, 帧错误选择
0: S0CON.7 作 SM0 功能
1: S0CON.7 作 FE 功能。注：当帧错误后不管 SMOD0 什么状态 FE 都将置位

S0CFG: 串行口 0 配置寄存器

SFR 页 = **0**

SFR 地址 = 0x9C 复位值 = 0000-100X

7	6	5	4	3	2	1	0
URTS	SMOD2	URM0X3	SM30	S0DOR	BTI	UTIE	--
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**W**

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。（参考章节“17.7.3 模式 1 和 3 波特率”）

Bit 6: SMOD2, UART0 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: URM0X3, 串行口模式 0 和模式 4 波特率选择
0: 清零选择 SYSCLK/12 作 UART 模式 0 和模式 4 波特率
1: 置位选择 SYSCLK/4 作 UART 模式 0 和模式 4 波特率

Bit 4: SM30, 串行口模式控制位 3
0: 禁止串行口模式 4
1: 使能 SM30 去控制串行口模式 4，SPI 主机。更多 S0 模式选择信息参考 S0CON 描述

Bit 3: S0DOR,串行口 0 模式 4 数据序位控制
0: 数据字节高位在先(MSB)传送
1: 数据字节低位在先(LSB)传送。默认是 S0DOR 为“1”

Bit 2: BTI,在串行口 0 中断阻止 TI0
0: 保留 TI0 作为一个串行口 0 中断源
1: 阻止 TI0 作为一个串行口 0 中断源

Bit 1: UTIE,在系统标志中断里使能 S0 TI0
0: 禁止在系统标志中断里中断向量共享给 TI0
1: 设置 TI0 标志将与系统标志中断共享中断向量

Bit 0: 保留。当 S0CFG 写时，软件必须写“0”

AUXR2: 辅助寄存器 2

SFR 页 = 0~F

SFR 地址 = 0xA3 复位值 = 0000-0000

7	6	5	4	3	2	1	0
INT3IS1	INT3IS0	INT2IS1	INT2IS0	**T1X12**	T0X12	T1CKOE	T0CKOE
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 3: T1X12, 当 C/T=0 时，定时器 1 时钟源选择
0: 清零选择 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

18. 串行口 1 (UART1)

MA82G5BXX 装备有第 2 个 UART（以后就称作 UART1），和第 1 个 UART 一样，也有 5 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(S1BRG)
(3) UART1 使用端口 TXD1 (P1.3/P3.4)和 RXD1 (P1.2/P3.3) 分别作为接收和发送端口
(4) 波特率发生器提供 S1CKO 切换时钟源和外设时钟
(5) S1 + S1BRG 在端口改变检测下可以被配置成一个 8 位自动重载定时器
(6) 模式 0 和模式 4，UART1 的 S1TX12 与 UART0 的 URM0X3 一样的功能

MA82G5BXX 的 UART1 和 UART0 可以不同或相同模式、不同或相同通讯速率同时工作。

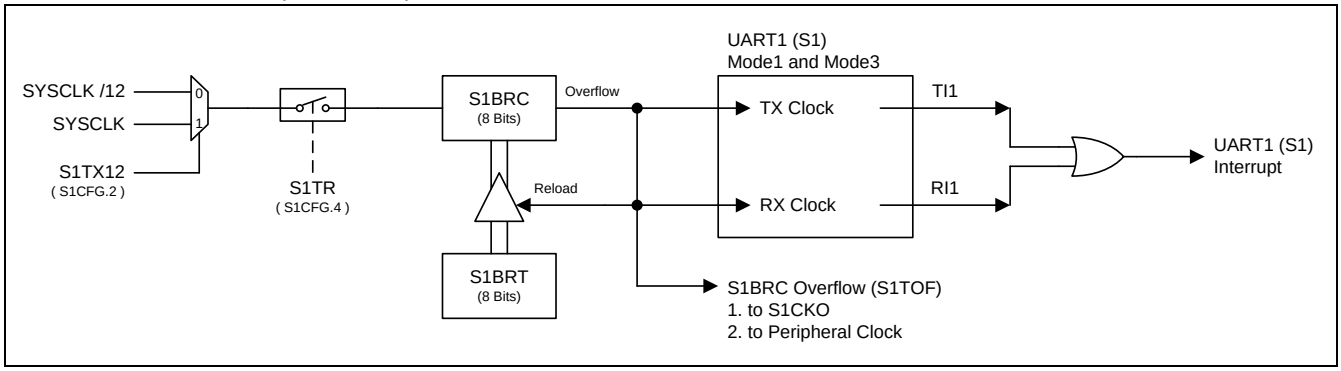
18.1. 串行口 1 波特率发生器(S1BRG)

MA82G5BXX 在模式 1 和模式 3 有一个嵌入式波特率发生器给串行口 1 提供 UART 时钟。它由一个 8 位的向上计数器(S1BRC)和一个自动装载寄存器(S1BRT)构成。S1BRC 的溢出(S1TOF)是 UART1 在模式 1 和模式 3 串行引擎的时间基准并且触发 S1BRT 的值重载到 S1BRC 继续计数。

此波特率发生器通过软件配置也能给串行口 0 提供时基。这里有一个 S1BRC 溢出率 2 分频(S1TOF/2)的额外时钟输出（S1CKO）。S1TOF 也支持 UART0, PCA, SPI, TWI0, TWI1 和 ADC 时钟输入源切换。无论 S1 运行还是待运行，S1BRG 总是服务于这些外设的时间基准功能。

串行口 1 波特率发生器框图见图 18–1。

图 18–1. S1BRG 框图(S1TME=0)



18.2. 串行口 1 波特率设定

18.2.1. 模式 0 波特率

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

; n=12, if S1TX12=0
; n=4, if S1TX12=1

18.2.2. 模式 2 波特率

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

18.2.3. 模式 1 和 3 波特率

$$\text{S1 Mode 1, 3 Baud Rate} = \frac{2^{S1MOD1}}{32} \times \frac{F_{SYSCLK}}{12 \times (256 - S1BRT)} \text{; S1TX12=0}$$

$$\text{or} = \frac{2^{S1MOD1}}{32} \times \frac{F_{SYSCLK}}{1 \times (256 - S1BRT)} \text{; S1TX12=1}$$

表 17-1 ~ 表 17-4 列出 S1BRG（串行端口 1 波特率产生器）各种通用的波特率和怎样获得

表 18–1. S1BRG 产生普通波特率@ F_{SYSCLK}=11.0592MHz

波特率	**S1BRT**, S1BRG 重载值					
	S1TX12=0			**S1TX12=1**		
	S1MOD1=0	S1MOD1=1	误差	S1MOD1=0	S1MOD1=1	误差
1200	232	208	0.0%	--	--	--
2400	244	232	0.0%	112	--	0.0%
4800	250	244	0.0%	184	112	0.0%
9600	253	250	0.0%	220	184	0.0%
14400	254	252	0.0%	232	208	0.0%
19200	--	253	0.0%	238	220	0.0%
28800	255	254	0.0%	244	232	0.0%
38400	--	--	--	247	238	0.0%
57600	--	255	0.0%	250	244	0.0%
115200	--	--	--	253	250	0.0%
230400	--	--	--	--	253	0.0%

表 18–2. S1BRG 产生普通波特率@ F_{SYSCLK}=22.1184MHz

波特率	**S1BRT**, S1BRG 重载值					
	S1TX12=0			**S1TX12=1**		
	S1MOD1=0	S1MOD1=1	误差	S1MOD1=0	S1MOD1=1	误差
1200	208	160	0.0%	--	--	--
2400	232	208	0.0%	--	--	0.0%
4800	244	232	0.0%	112	--	0.0%
9600	250	244	0.0%	184	112	0.0%
14400	252	248	0.0%	208	160	0.0%
19200	253	250	0.0%	220	184	0.0%
28800	254	252	0.0%	232	208	0.0%
38400	--	253	0.0%	238	220	0.0%
57600	255	254	0.0%	244	232	0.0%
115200	--	255	0.0%	250	244	0.0%
230400	--	--	--	253	250	0.0%
460800	--	--	--	--	253	0.0%

表 18–3. S1BRG 产生普通波特率@ F_{SYSCLK}=12.0MHz

波特率	**S1BRT**, S1BRG 重载值					
	S1TX12=0			**S1TX12=1**		
	S1MOD1=0	S1MOD1=1	误差	S1MOD1=0	S1MOD1=1	误差
1200	230	204	0.16%	--	--	--
2400	243	230	0.16%	100	--	0.16%
4800	--	243	0.16%	178	100	0.16%
9600	--	--	--	217	178	0.16%
14400	--	--	--	230	204	0.16%
19200	--	--	--	--	217	0.16%
28800	--	--	--	243	230	0.16%
38400	--	--	--	246	236	2.34%
57600	--	--	--	--	243	0.16%
115200	--	--	--	--	--	--

表 18–4. S1BRG 产生普通波特率@ F_{SYSCLK}=24.0MHz

波特率	**S1BRT**, S1BRG 重载值					
	S1TX12=0			**S1TX12=1**		
	S1MOD1=0	S1MOD1=1	误差	S1MOD1=0	S1MOD1=1	误差
1200	204	152	0.16%	--	--	--
2400	230	204	0.16%	--	--	--
4800	243	230	0.16%	100	--	0.16%
9600	--	243	0.16%	178	100	0.16%
14400	--	--	--	204	152	0.16%
19200	--	--	--	217	178	0.16%
28800	--	--	--	230	204	0.16%
38400	--	--	--	--	217	0.16%
57600	--	--	--	243	230	0.16%
115200	--	--	--	--	243	0.16%

18.3. 串行口 1 模式 4 (SPI 主机)

MA82G5BXX 串行口嵌入了一个模式 4 支持 SPI 主机引擎。模式 4 由 SM31, SM01 和 SM11 选择。表 18–5 展示了 **MA82G5BXX** 的串行口模式定义。

表 18–5.串行口 1 模式选择

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

S1TX12 也是控制 SPI 的传输速度。如果 S1TX12 = 1,则 SPI 的时钟频率是 SYSCLK/4。如果 S1TX12 = 0, 则 SPI 的时钟频率是 SYSCLK/12。

MA82G5BXX 的 SPI 主机使用 TXD1 作为 SPICLK, RXD1 作为 MOSI,以及 S1MI 作为 MISO。而 nSS 由 MCU 软件选择在其它端口引脚。图 18–2 展示了 SPI 连接。他支持多从机通讯架构见图 18–3

图 18–2.串行口 1 模式 4,单主机和单从机架构(n = 1)

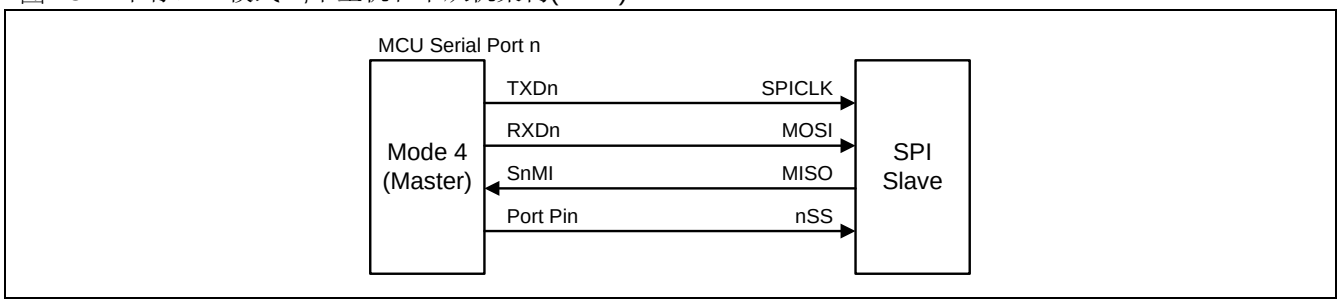
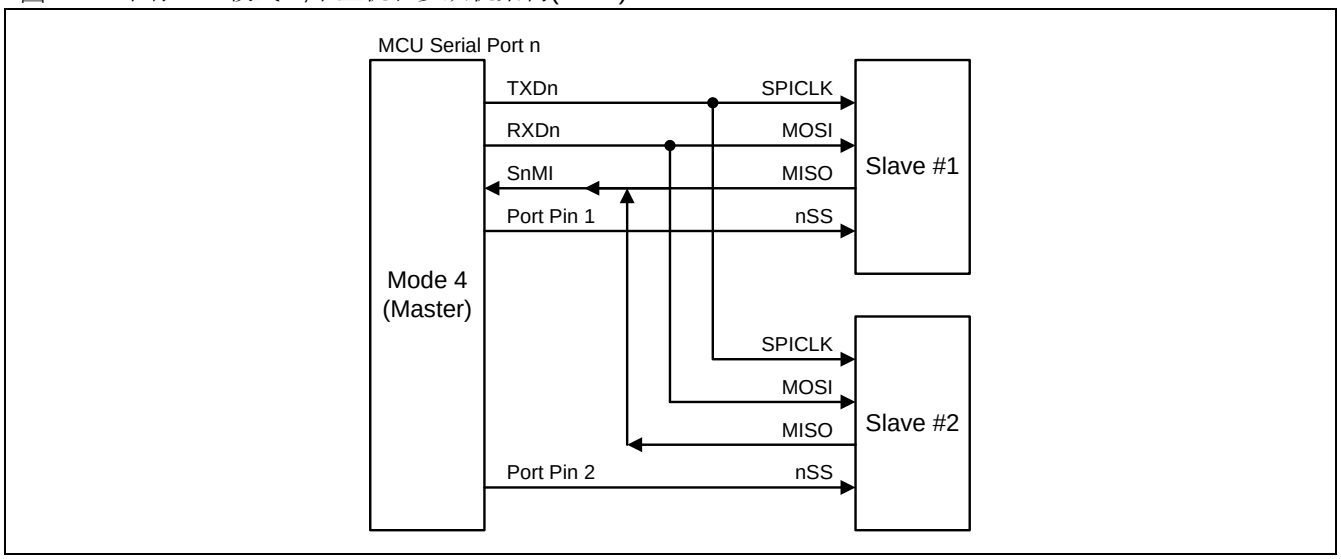


图 18–3.串行口 1 模式 4,单主机和多从机架构(n = 1)



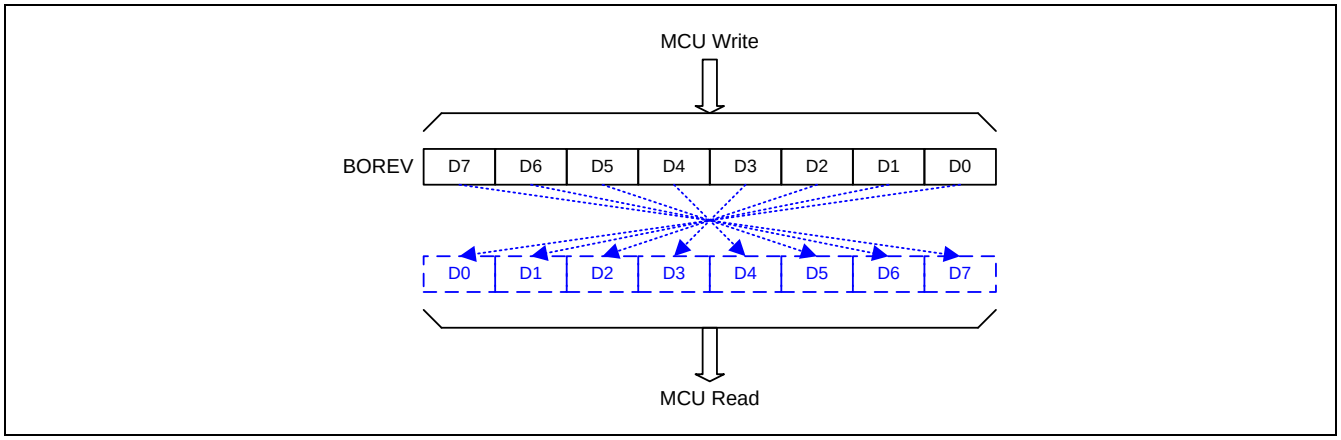
SPI 主机能满足笙泉 MA82/84 系列 MCU（由 CPOL, CPHA 和 DORD 选择）的全功能 SPI 模块的传输。在 CPOL 和 CPHA 条件下, **MA82G5BXX** 很容易初始化 SPI 的时钟(TXD1, P1.3/P3.4)极性去适合他们使用 。表 17–21 展示了串行口模式 4 的 4 个 SPI 工作模式。

表 18–6.串行口 1 模式 4 的 SPI 模式结构

SPI Mode	CPOL	CPHA	当 TXD1 在 P1.3 脚 **MA82G5BXX** 的结构
0	0	0	清除 P1.3 为“0”
1	0	1	清除 P1.3 为“0”
2	1	0	设置 P1.3 为“1”
3	1	1	设置 P1.3 为“1”

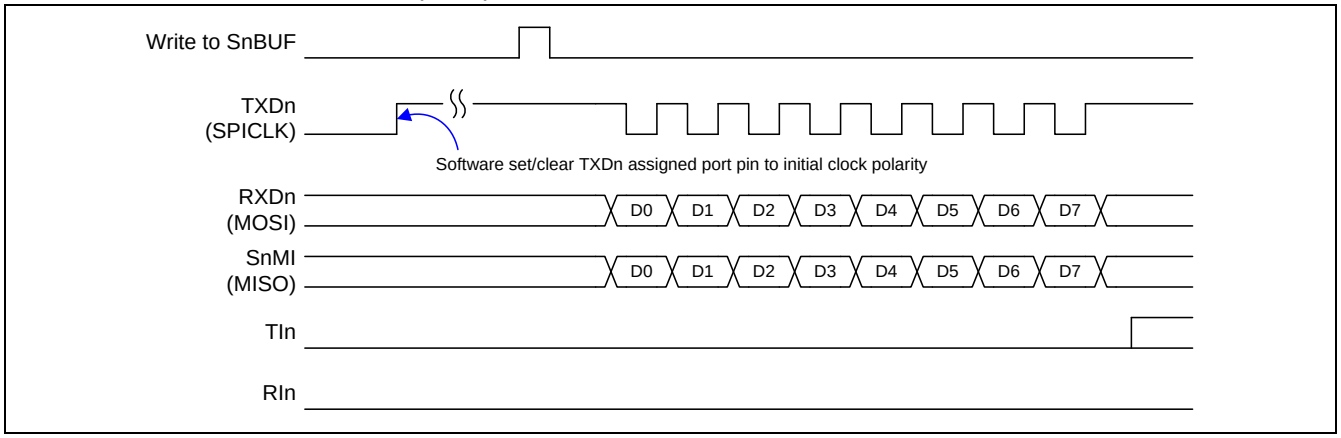
SPI 系列传输的位序控制(DORD)，**MA82G5BXX** 提供了一个软件编程可以相反的位序控制（SFR, BOREV）。 在 MCU 写一个数据格式（MSB）到 BOREV ， MCU 通过读 BOREV 得到一个数据格式（LSB）回来。SPI 主机引擎在串行口 1 模式 4 与串行口 1 模式 0 一样都是（LSB）传输。为了支持 SPI（MSB）移位传输, MCU 必须使用（BOREV）写/读取操作来翻转 SPI 输入/输出传送的数据位序。图 18–4 展示了 BOREV 结构。

图 18–4. SFR BOREV 读取/写结构



通过指令使用 S1BUF 作为目标寄存器初始化传送。“写数据到 S1BUF”触发 UART 引擎开始传送。S1BUF 的数据被移位到作为 MOSI 串口数据的 RXD1 引脚。SPI 移位时钟由作为 SPICLK 输出的 TXD1 引脚输出。在 8 个移位时钟的上升沿之后，TI1 被硬件声明传送结束。同时 S0MI 引脚的内容被采样并且移位到移位寄存器。然后“读取 S1BUF”能获取 SPI 的移入数据。图 18–5 展示了模式 4 传送波形。RI1 在模式 4 不被声明。

图 18–5.串行口 1 模式 4 传送波形(n = 1)



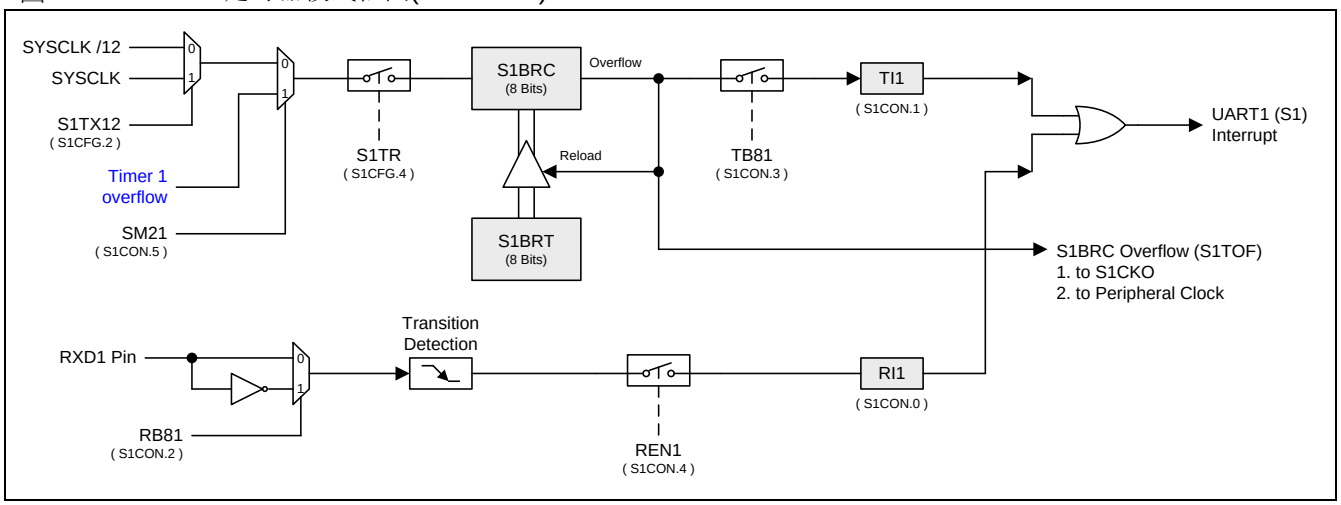
18.4. S1BRG 纯定时器模式

如果 UART1 不用或软件搁置，**MA82G5BXX** 设定 S1TME=1 串行口 1 波特率发生器(S1BRG)提供纯定时器操作模式。此定时器操作在一个 8 位自动重载定时器并且提供溢出标志（即 S1CON.1 的 TI1 置位）。S1CON.0 的 RI1 服务于 RXD1 端口引脚的端口变化检测器。在此模式下 TI1 和 RI1 保持 UART1 中断源的中断能力并且有一个独立中断使能控制(TB81 和 REN1)。RB81 选择 RXD1 端口输入的 RI1 检测电平。如果 RB81=0，RI1 由 REN1=1 并且检测到 RXD1 引脚下降沿置位；否则 RI1 将检测 RXD1 端口引脚的下降沿。在 MCU 掉电模式，RI1 被强制成电平传感操作并且如果 UART1 中断使能可以唤醒 CPU。

此纯定时器模式有一个来自定时器 1 溢出的时钟输入选项，是一个级联计数器工作在一个 16 位定时器。当 S1BRC 溢出，它将是 UART0, PCA, ADC, SPI, TWI0 和 TWI1 的时钟源或切换端口引脚输出。S1CKOE=1 使能 S1CKO 输出在端口引脚并且屏蔽 RI1 中断。

S1BRG 纯定时器模式框图见图 18–6。

图 18–6. S1BRG 定时器模式框图(S1TME=1)



18.5. S1BRT 可编程时钟输出

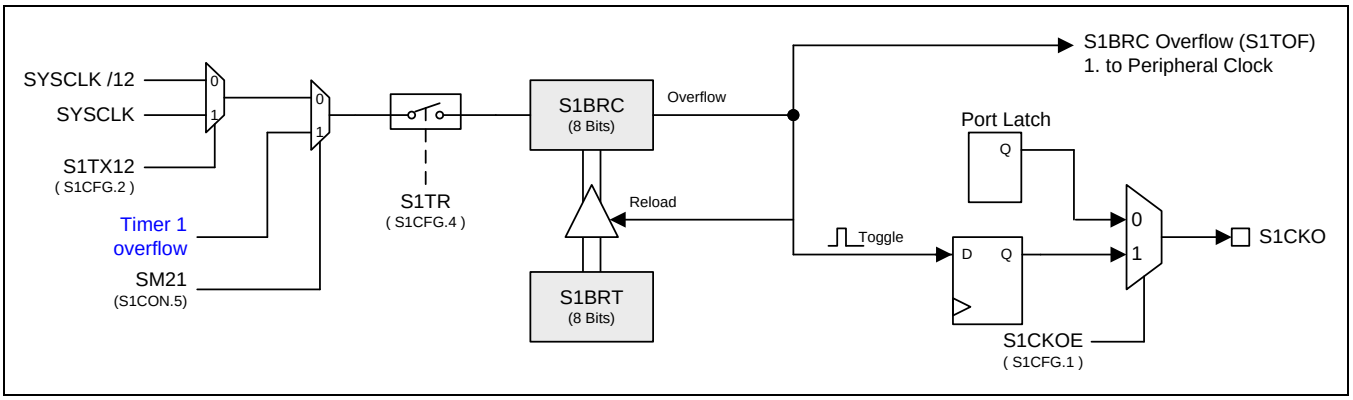
当 S1BRC 溢出时，溢出标志 S1TOF 提供 S1CKO 切换时钟源和外设时钟。输入时钟（SYSCLK/12 或 SYSCLK）作为 8 位定时器(S1BRC)时钟源。定时器重复计数到被载入值溢出。一旦溢出，S1BRC 载入 S1BRT 的值继续计数。图 18–7 是串行口 1 波特率产生器的方框图。时钟输出频率如下公式：

$$\text{S1T Clock-out Frequency} = \frac{\text{SYSCLK Frequency}}{n \times (256 - \text{S1BRT})}$$

; n=24, if S1TX12=0
; n=2, if S1TX12=1

注意:
(1) 如 SYSCLK=12MHz 和 S1TX12=0, S1BRG 可编程输出频率范围从 1.95KHz 到 500KHz.
(2) 如 SYSCLK=12MHz 和 S1TX12=1, S1BRG 可编程输出频率范围从 23.43KHz 到 6MHz.

图 18–7. S1BRG 在时钟输出模式



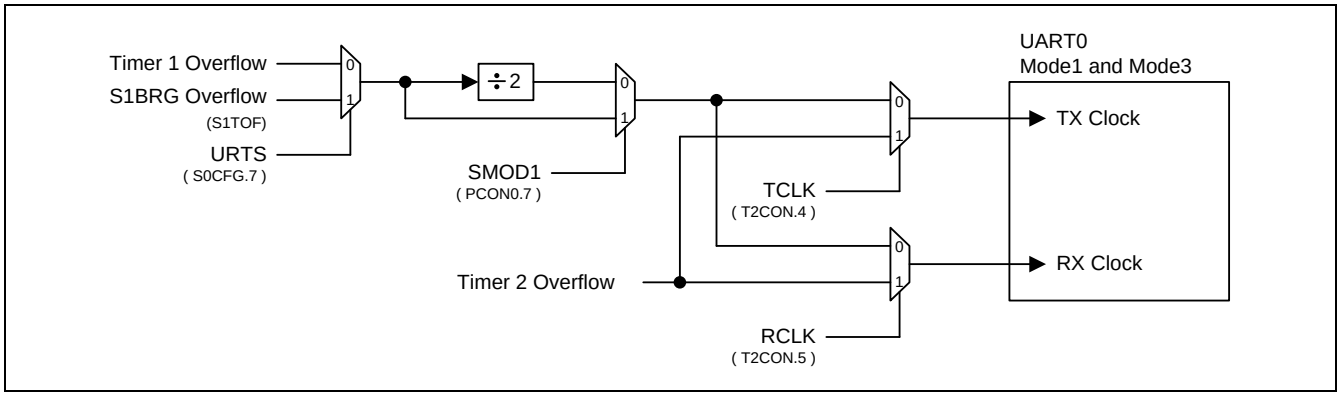
在时钟输出模式如何编程 8 位 S1BRG

- S1CFG.S1TX12 和 S1CON.SM21 选择 S1BRG 时钟源
- 由公式计算 8 位重装载值并且存入 S1BRT 和 S1BRC 寄存器
- S1CFG 寄存器的 S1CKOE 位置位
- S1TR 置位去启动 S1BRC 定时器

18.6. 串行口 0(S0)的波特率定时器来自串行口 1(S1)

串行口 0（UART0）的模式 1 和 3 操作，软件通过 T2CON 寄存器的位 TCLK 和 RCLK 清零可以选择定时器 1 作为波特率发生器。同时，如果 URTS(在 S1CFG 寄存器)置位，定时器 1 溢出信号通过 UART1 波特率定时器溢出信号被替代。换句话说，只要 RCLK=0, TCLK=0 和 URTS=1 用户可以采用 UART1 波特率定时器作为 UART0 模式 1 或 3 的波特率发生器。这种情况下，定时器 1 也可以用作其他应用。当然，如果 UART1 的模式 1 或 3 也同时操作，则两个 UART 具有相同的波特率。

图 18–8. UART0 额外波特率源



18.7. 串行口 1 寄存器

下面的特殊功能寄存器与 UART1 有关：

S1CON: 串行口 1 控制寄存器

SFR 页 = **1**

SFR 地址 = 0x98 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

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

Bit 5: 串行口 0 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: 软件置位使能接收

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，RB81 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

S1BUF: 串行口 1 缓冲寄存器

SFR 页 = **1**

SFR 地址 = 0x99 复位值 = XXXX-XXXX

7	6	5	4	3	2	1	0
S1BUF.7	S1BUF.6	S1BUF.5	S1BUF.4	S1BUF.3	S1BUF.2	S1BUF.1	S1BUF.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 = **1**

SFR 地址 = 0x9A 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

S1BRC:串行口 1 波特率计数寄存器

SFR 页 = **1**

SFR 地址 = 0x9B 复位值 = 0000-0000

7	6	5	4	3	2	1	0
S1BRC.7	S1BRC.6	S1BRC.5	S1BRC.4	S1BRC.3	S1BRC.2	S1BRC.1	S1BRC.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~0: 用作波特率定时器发生器重载寄存器，用法与定时器 1 相似。此寄存器总是可以软件读写。如果 S1CFG1.S1TME = 0，软件写数据到 S1BRT 同时数据存入 S1BRT 和 S1BRC

S1CFG: 串行口 1 配置寄存器

SFR 页 = **1**

SFR 地址 = 0x9C 复位值 = 0010-0000

7	6	5	4	3	2	1	0
SM31	0	1	S1TR	S1MOD1	S1TX12	S1CKOE	S1TME
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: SM31, 串行口 1 模式控制位。参考 S1CON 描述

Bit 6~5: 保留。当 S1CFG 写入时，这些位软件必须写”01”。

Bit 4: S1TR, UART1 波特率定时器控制位
0: 清零关闭 S1BRG
1: 置位开启 S1BRG

Bit 3: S1MOD1, UART1 双倍波特率选择使能位
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:清零选择 SYSCLK/12 作 S1BRG 的时钟源
1:置位选择 SYSCLK 作 S1BRG 的时钟源

Bit 1: S1CKOE,串行口 1 波特率定时器时钟输出使能
0:禁止 S1CKO 在端口引脚输出
1:使能 S1CKO 在端口引脚输出

Bit 0: S1TME,串行口 1 波特率(BRG)定时器模式使能
0:保持 S1BRG 服务串行口 1 (UART1)
1:禁止串行口 1 功能并且 S1BRG 作为一个 8 位自动装载的定时器。这个模式下，这是一个 RXD1 端口引脚变化检测器的额外功能

AUXR1:辅助控制寄存器 1

SFR 页 = 0~F

SFR 地址 = 0xA2 复位值 = 0000-0000

7	6	5	4	3	2	1	0
P1KBIH	P3KBIL	P4SPI	**P3S1**	**P3S1MI**	P6TWI	P3CEX	DPS
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 4: P3S1, 如果 P3CEX (AUXR1.1)是禁止的，串行口 1 (UART1)的功能在 P3.3 和 P3.4

P3S1	RXD1	TXD1
0	P1.2	P1.3
1	P3.3	P3.4

Bit 3: P3S1MI, S1MI 功能在 P3.5。S1MI 是串行口 1(S1)模式 4(SPI 主机)的串行数据输入

P3S1MI	S1MI
0	P1.0
1	P3.5

18.8. 串行口示例代码

(1). 规定功能:串行口输入 RI 唤醒空闲模式

汇编语言代码范例:

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

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

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

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

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

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

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

C 语言代码范例:

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

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

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

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

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

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

19. 可编程计数器阵列(PCA)

MA82G5BXX 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。

19.1. PCA 概述

PCA 由一个专用定时/计数器作为一个 **8** 组比较/捕获模块的时间基础，图 19–1 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，那么该功能就和相应的端口 2 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。

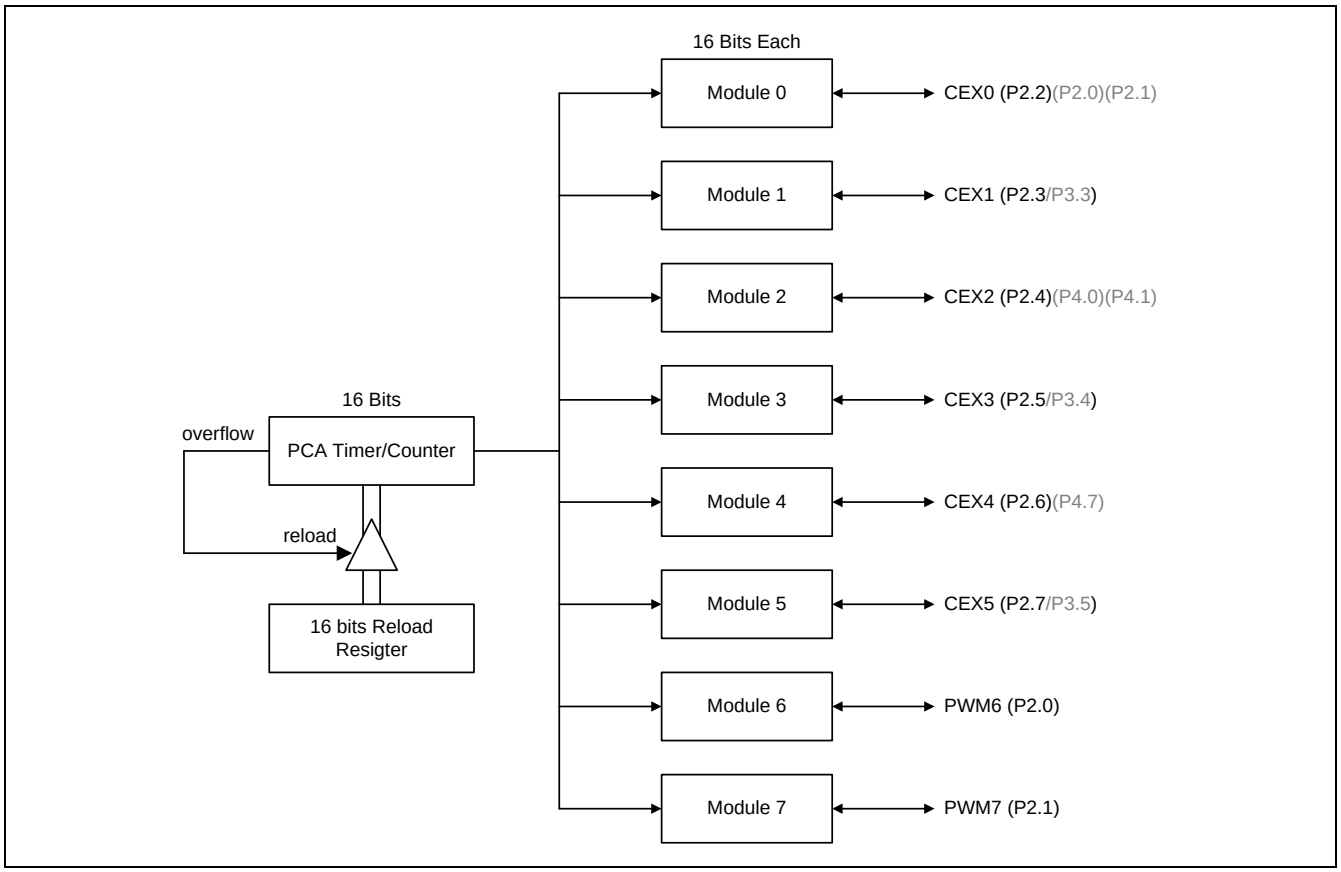
模块 0~5 都可以编程为如下任意模式：

-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

模块 6 和模块 7 仅支持 PWM 输出模式

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 19–1. PCA 方框图



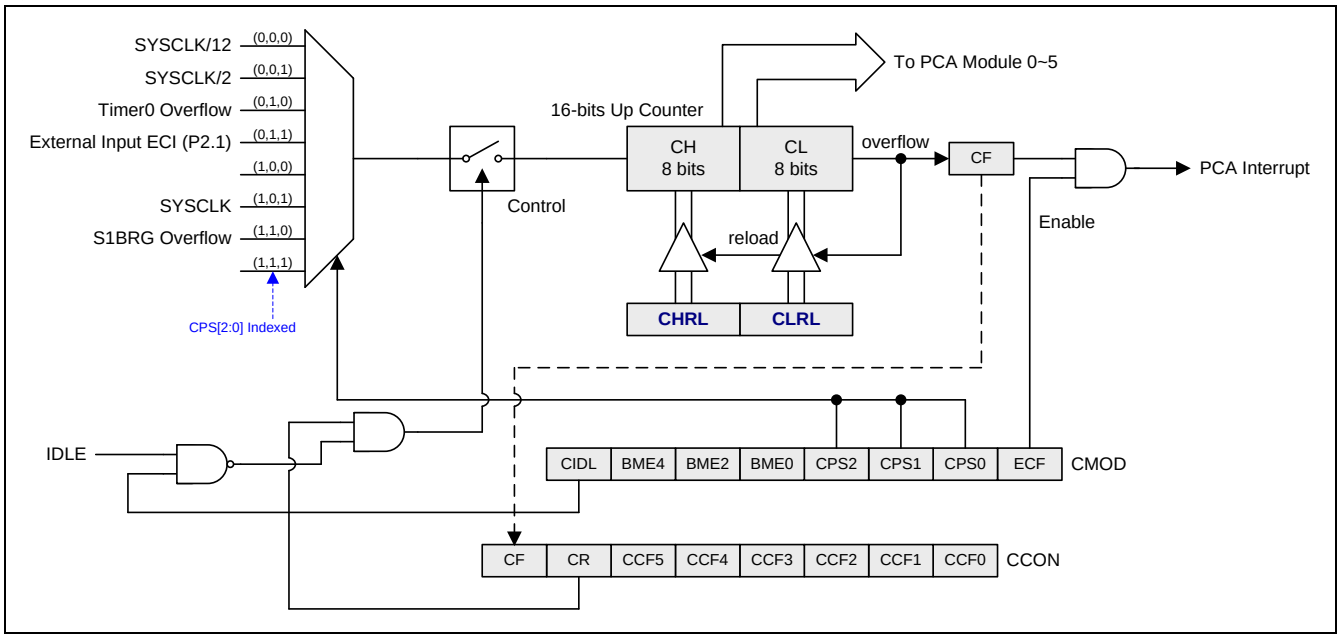
19.2. PCA 定时器/计数器

PCA 的定时器/计数器由一个可以自动重载的 16 位定时器由寄存器 CH 和 CL(计数值的高低字节)，CHRL 和 CLRL(重载寄存器的高低字节)组成，如图 19–2 所示。{CH + CL}计数器每一次溢出 CHRL 和 CLRL 被重载入 CH 和 CL，这样可以改变 PCA 周期时间提供可变的 PWM 周期，比如 7 位或 9 位的 PWM。

{CH + CL}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(P2.1/P3.2)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 (CPS2,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 19–2. PCA 定时器/计数器



CMOD: PCA 计数器模式寄存器

SFR 页 = 0~F

SFR 地址 = 0xD9　　复位值 = 0000-0000

7	6	5	4	3	2	1	0
CIDL	BME4	BME2	BME0	CPS2	CPS1	CPS0	ECF
R/W	R/W	R/W	R/W	R/W	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 6~4: BME4/2/0.保留为测试模式。当 CMOD 写入时，这些位软件必须写"0"

Bit 3~1: CPS2-CPS0, PCA 计数器时钟源选择位

CPS2	CPS1	CPS0	PCA 时钟源
0	0	0	内部时钟, 系统时钟 (system clock)/12
0	0	1	内部时钟, 系统时钟(system clock)/2
0	1	0	定时器 0 溢出
0	1	1	来自 ECI 引脚的外部时钟
1	0	0	保留
1	0	1	系统时钟(SYSCLK)
1	1	0	S1BRG 溢出
1	1	1	保留

Bit 0: ECF,使能 PCA 计数器溢出中断
0:当 CF 位（CCON 寄存器中）置位时禁止中断
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为（CCON.6）必须软件置位，要关闭PCA，可以清除该位。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 19–3所示。

CCON: PCA 计数器控制寄存器

SFR 页 = 0~F

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 计数器运行控制位
0: 软件清零关闭 PCA 计数器
1:软件置位启动 PCA 计数器

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

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

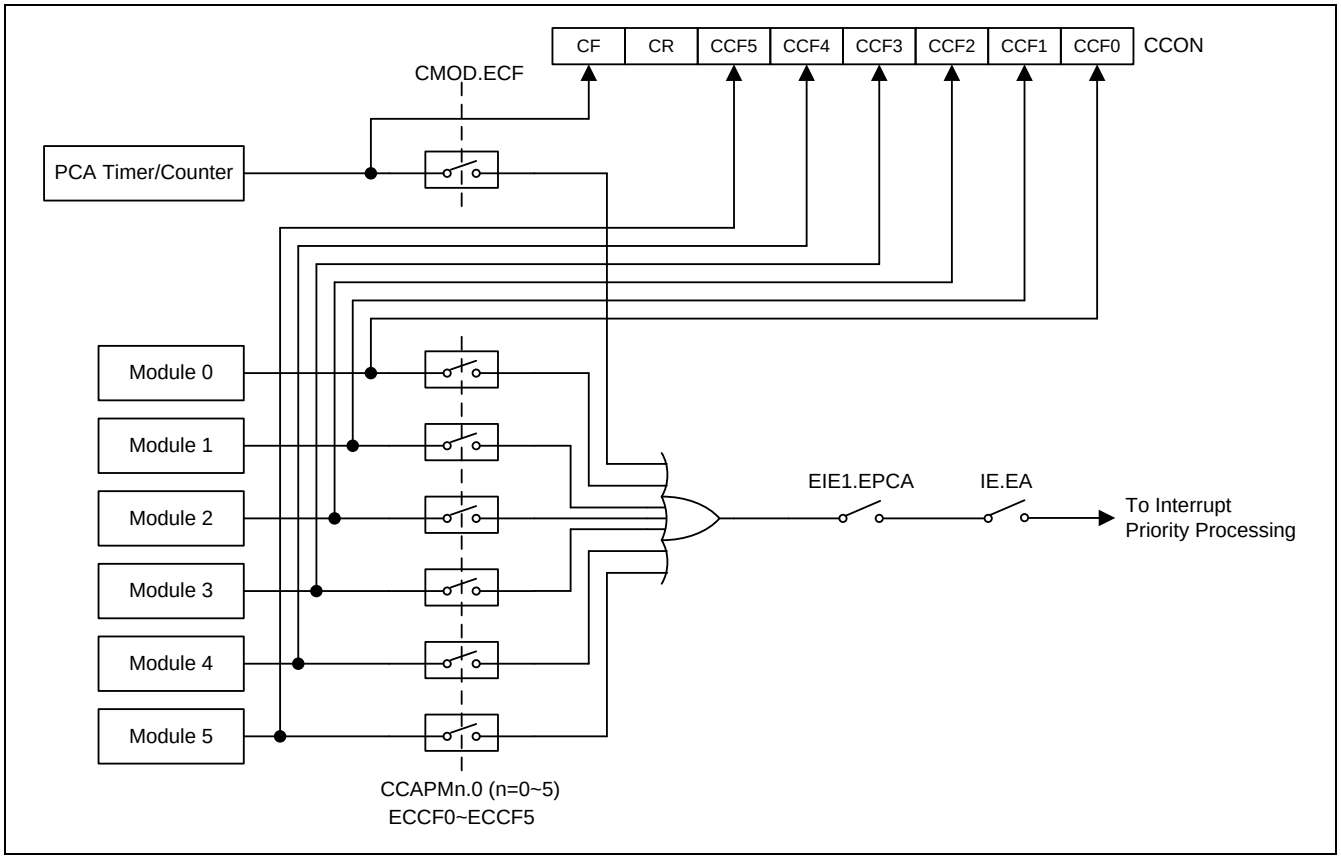
Bit 3: CCF3, PCA 模块 3 中断标志
0: 必须软件清零
1: 当发生一个匹配或捕获事件时，硬件置位

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

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

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

图 19–3. PCA 中断系统



19.3. 比较/捕获模块

比较/捕获模块 0~5 中的每一组都有一个模式寄存器，叫做 CCAPMn（n = 0，1，2，3，4 或 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 = **0**

SFR 地址 = 0xDA~0xDF 复位值 = 0000-0000

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

Bit 7: DETn, 保留为测试模式。当 CCAPMn 写入时，此位软件必须写”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翻转 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, CCFn 中断使能
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

注意：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

模块 6~7 仅有 PWM 功能由 CCAPMn.PWMn (n = 6 或 7)的设置来使能。这两个模块没有中断标志。

CCAPMn: PCA 模块比较/捕获寄存器, n=6~7
SFR 页 = **1**
SFR 地址 = 0xDA~0xDB 复位值 = 0xxx-xx0x

7	6	5	4	3	2	1	0
BME6	--	--	--	--	--	PWMn	--
R/W	**W**	**W**	**W**	**W**	**W**	**R/W**	**W**

Bit 7~2: 保留。当 CCAPM6 或 CCAPM7 写入时，这些位软件必须写”0”

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 PWM6 或 PWM7 引脚用作脉宽调制输出引脚

Bit 0: 保留。当 CCAPM6 或 CCAPM7 写入时，这位软件必须写”0”

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。

CCAPnH: PCA 模块 n 捕获高寄存器, n=0~5
SFR 页 = **0**
SFR 地址 = 0xFA~0xFF 复位值 = 0000-0000

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

CCAPnH: PCA 模块 n 捕获高寄存器, n=6~7

SFR 页 = **1**

SFR 地址 = 0xFA~0xFB 复位值 = 0000-0000

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

CCAPnL: PCA 模块 n 捕获低寄存器, n=0~5

SFR 页 = **0**

SFR 地址 = 0xEA~0xEF 复位值 = 0000-0000

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

CCAPnL: PCA 模块 n 捕获低寄存器, n=6~7

SFR 页 = **1**

SFR 地址 = 0xEA~0xEB 复位值 = 0000-0000

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

PCAPWMn: PWM 模式辅助寄存器, n=0~5

SFR 页 = **0**

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**

Bit 1: ECAPnH, 扩展第 9 位(MSB), 联合 CCAPnH 形成 9 位寄存器用于 PWM 模式。

Bit 0: ECAPnL,扩展第 9 位(MSB)，联合 CCAPnL 形成 9 位寄存器用于 PWM 模式。

PCAPWMn: PWM 模式辅助寄存器, n=6~7

SFR 页 = **1**

SFR 地址 = 0xF2~0xF3 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 1: ECAPnH,扩展第 9 位(MSB), 联合 CCAPnH 形成 9 位寄存器用于 PWM 模式

Bit 0: ECAPnL,扩展第 9 位(MSB)，联合 CCAPnL 形成 9 位寄存器用于 PWM 模式

19.4. PCA 操作模式

表 19–1 显示了不同 PCA 功能对应的 CCAPMn 寄存器设置。

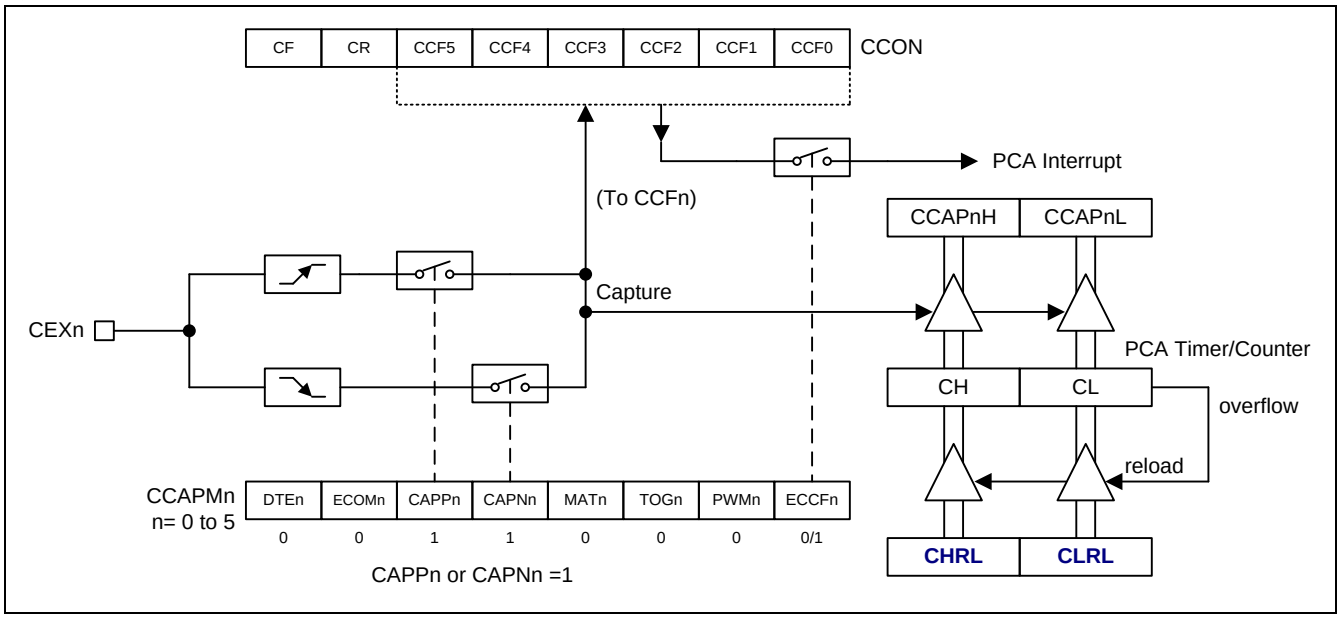
表 19–1. PCA 模块模式

ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	模块功能
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	脉宽调制器(PWM)

19.4.1. 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。

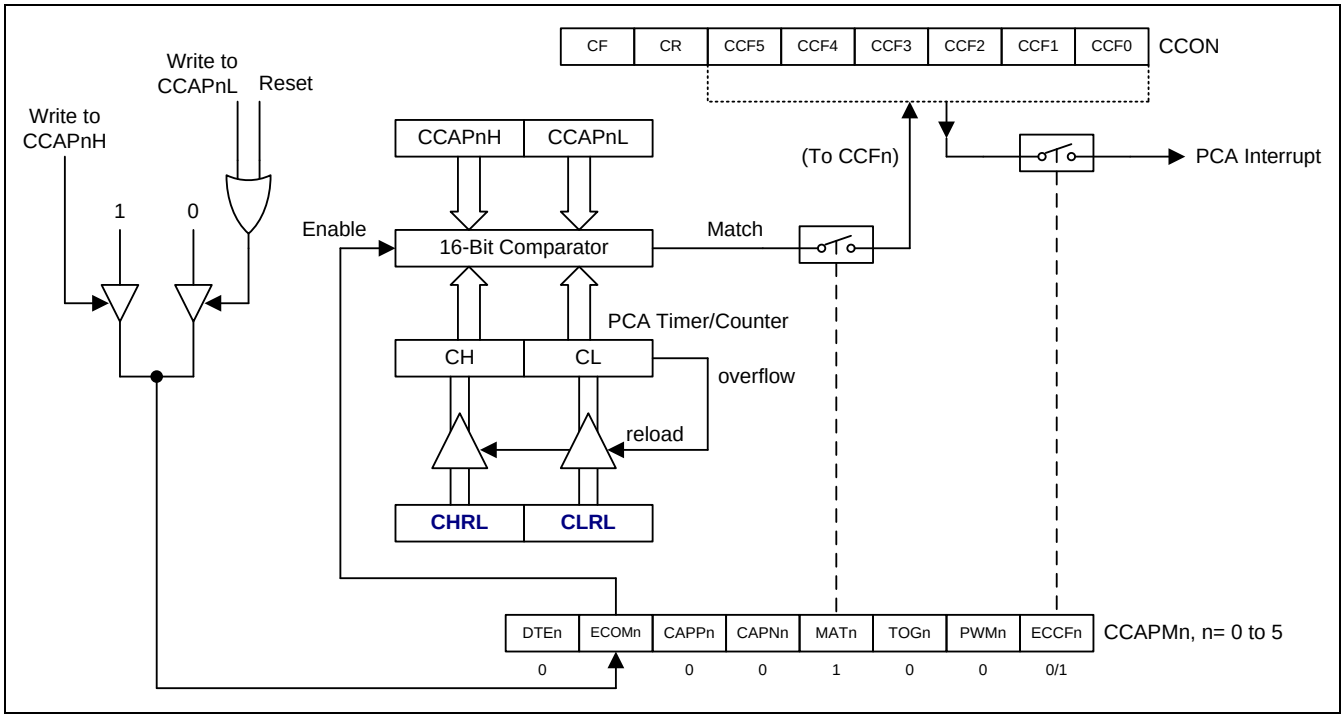
图 19–4. PCA 捕获模式



19.4.2. 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。

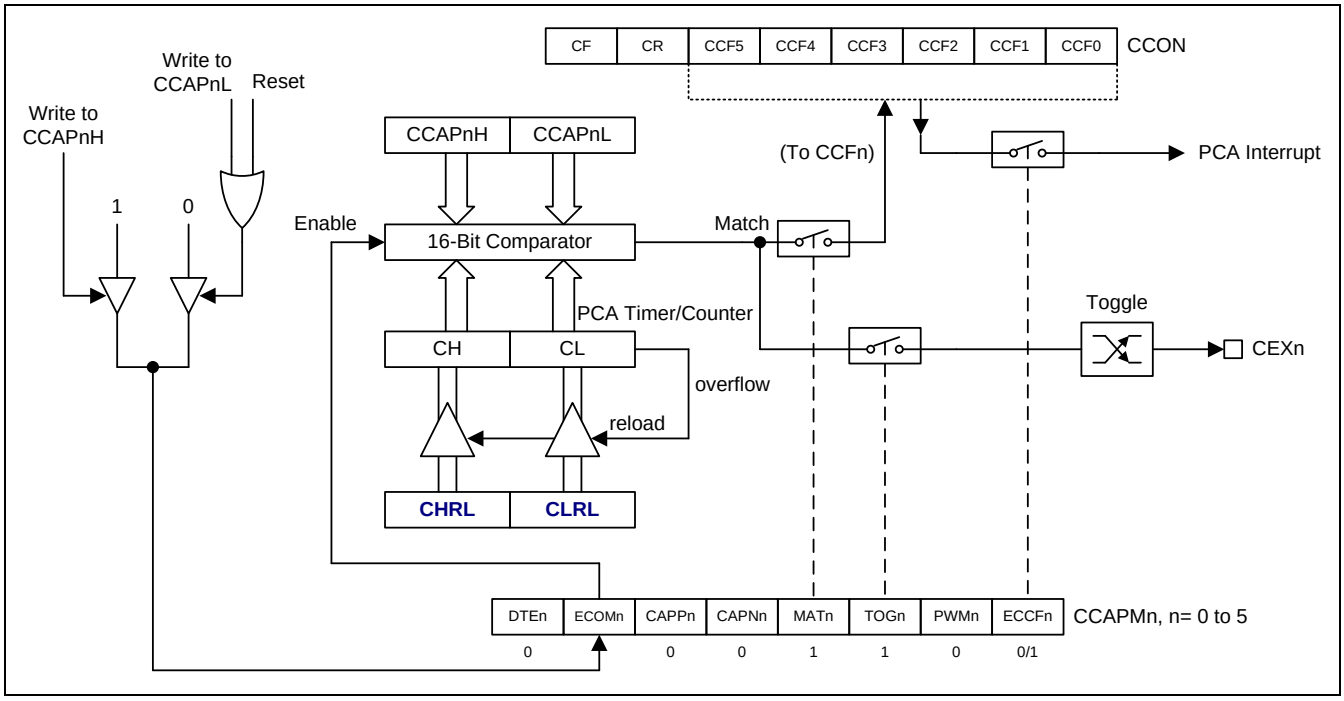
图 19–5. PCA 软件定时器模式



19.4.3. 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 19–6. PCA 高速输出模式



19.4.4. 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。

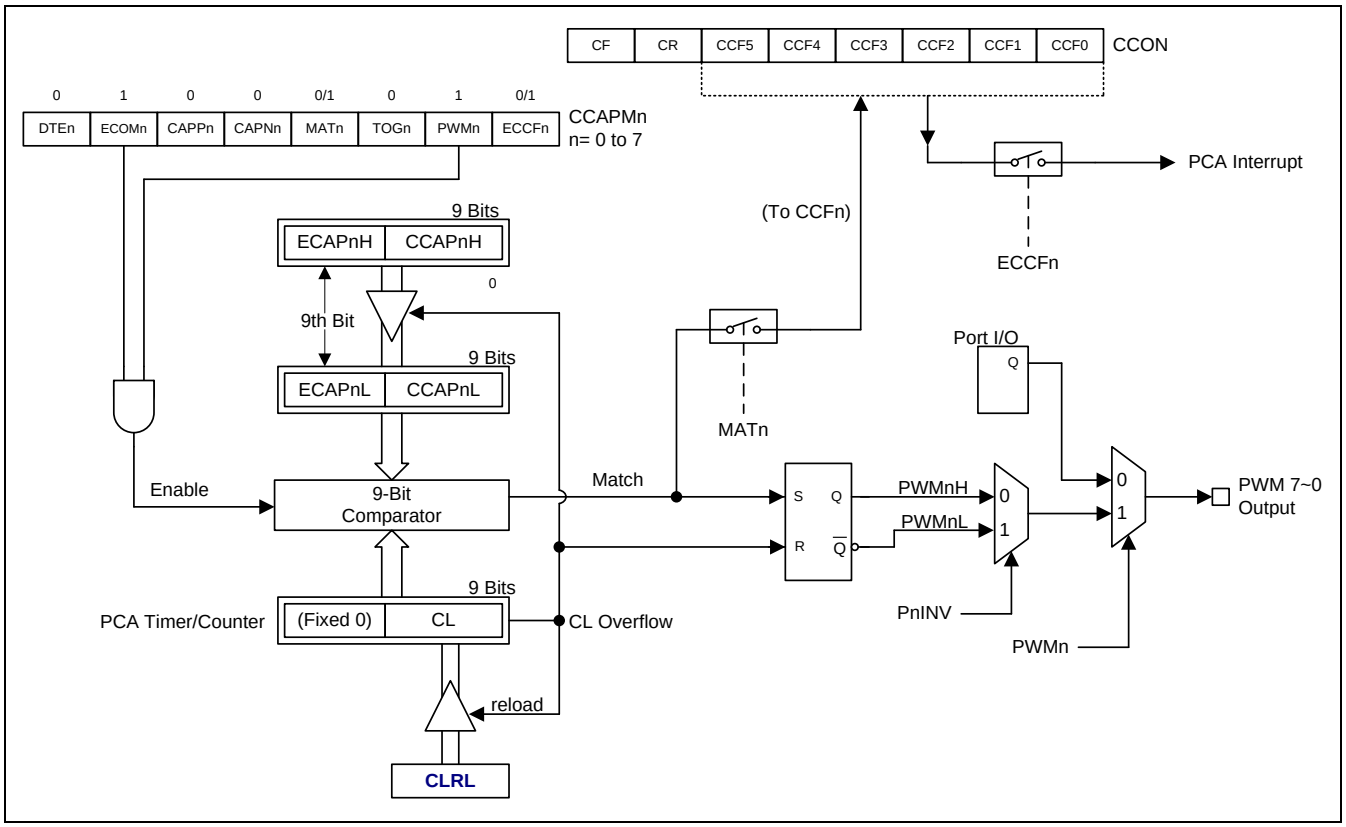
使用9位比较，输出的占空比可以真正实现从0%到100%可调。占空比计算公式如下：

占空比= 1 – { ECAPnH, [CCAPnH] } / 256.

这里， [CCAPnH] 是CCAPnH 寄存器的8位值， 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), 占空比为100%.
b. 若ECAPnH=0且CCAPnH=0x40 (即9位值, 0x040), 占空比为75%.
c. 若ECAPnH=0且CCAPnH=0xC0 (即9位值, 0x0C0), 占空比为25%.
d. 若ECAPnH=1且CCAPnH=0x00 (即9位值, 0x100), 占空比为0%.

图 19–7. PCA PWM 模式



19.4.5. 增强型 PWM 模式

PCA 提供不同的 PWM 模式增强 PWM 应用中的控制能力。这里有增加的 10/12/16 位 PWM 分配给每一个通道并且每个 PWM 通道拥有可以同时工作的不同的分别率和不同的相位延时。

图 19–8. 10/12/16 位 PWM 模式的 PCA 增强型 PWM

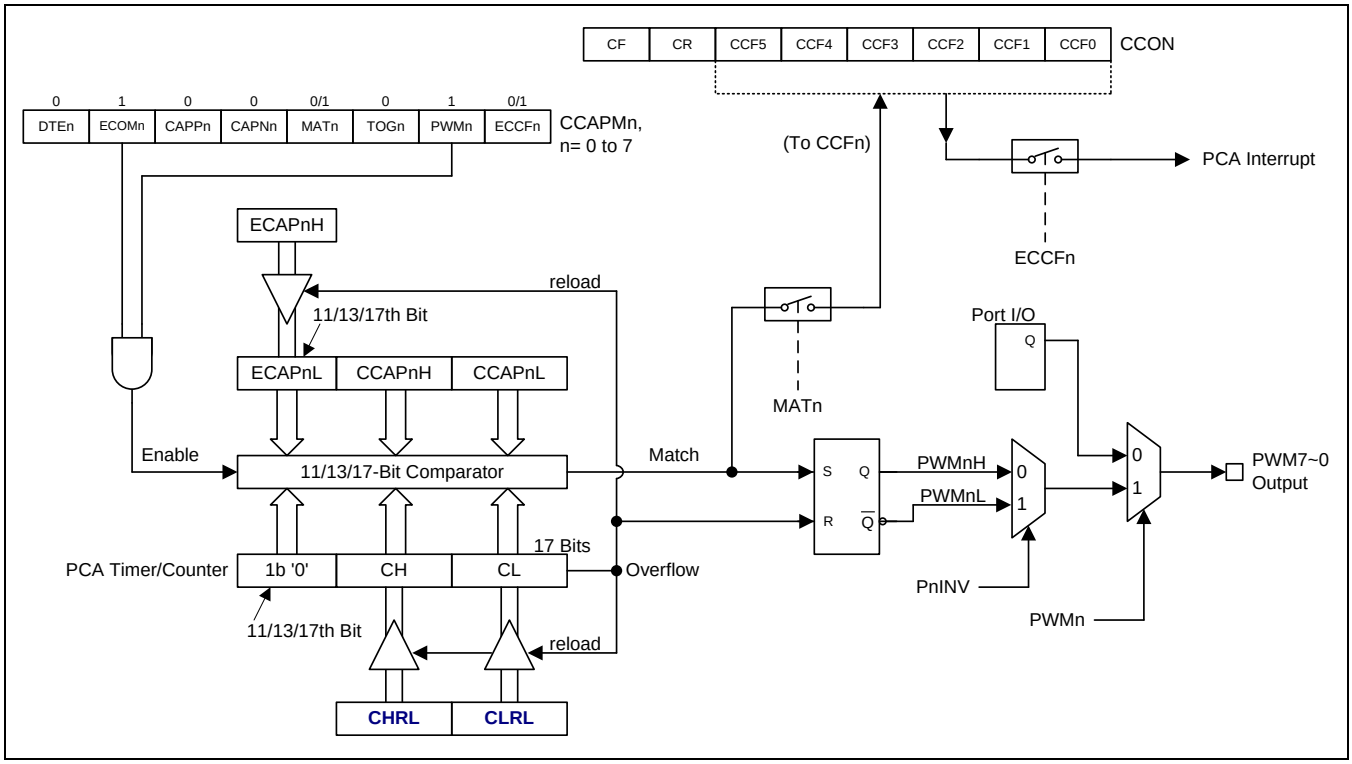
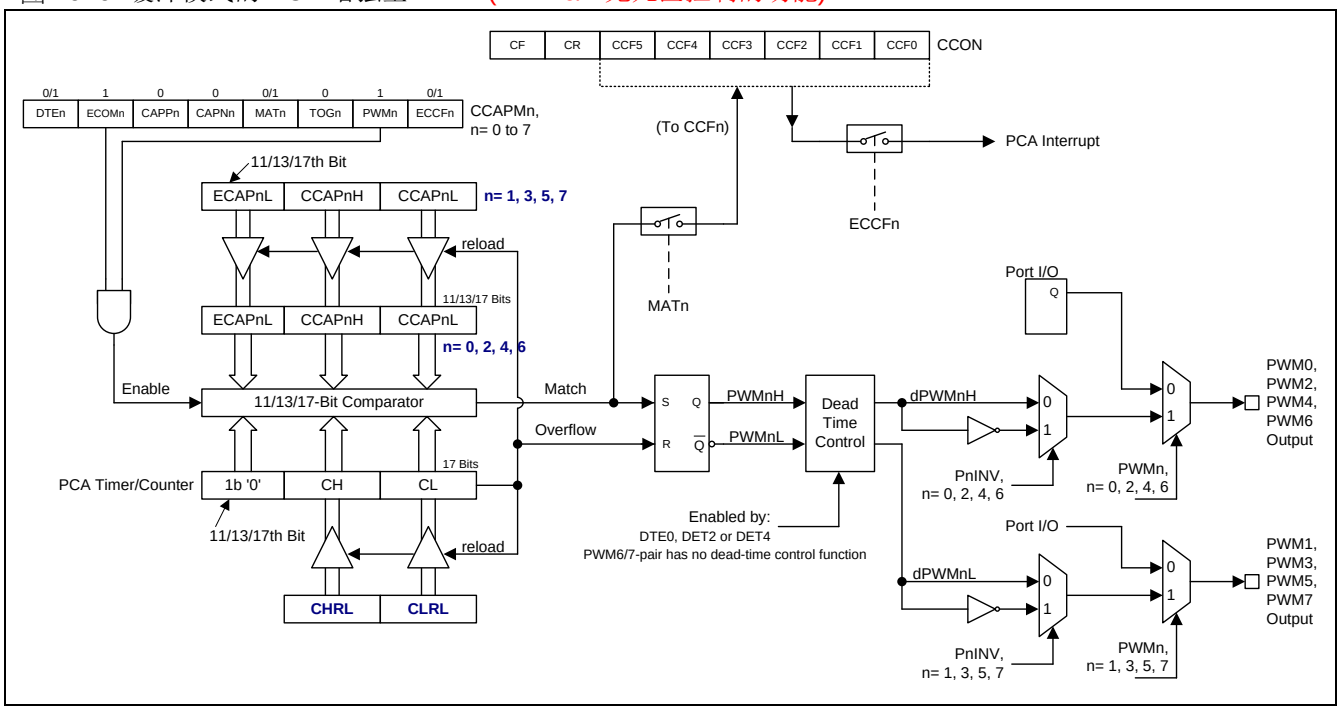


图 19–9. 缓冲模式的 PCA 增强型 PWM (PWM6/7 无死区控制的功能)



PAOE: PWM 额外输出使能寄存器

SFR 页 = 0~F

SFR 地址 = 0xF1 复位值 = 0001-1001

7	6	5	4	3	2	1	0
P47OP4	P41OP2	P40OP2	P24OP2	P26OP4	P21OP0	P20OP0	P22OP0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7: P47OP4, P47 使能输出 PWM4.
0: 禁止 P47 输出 PWM4. 默认是禁止.
1: 使能 P47 输出 PWM4.

Bit 6: P41OP2, P41 使能输出 PWM2.
0: 禁止 P41 输出 t PWM2. 默认是禁止.
1: 使能 P41 输出 PWM2.

Bit 5: P40OP2, P40 使能输出 PWM2.
0: 禁止 P40 输出 PWM2. 默认是禁止.
1: 使能 P40 输出 PWM2.

Bit 4: P24OP2, P24 使能输出 PWM2.
0: 禁止 P24 输出 PWM2.
1: 使能 P24 输出 PWM2. 默认是使能.

Bit 3: P26OP4, P26 使能输出 PWM4.
0: 禁止 P26 输出 PWM4.
1: 使能 P26 输出 PWM4. 默认是使能.

Bit 2: P21OP0, P21 使能输出 PWM0.
0: 禁止 P21 输出 PWM0. 默认是禁止.
1: 使能 P21 输出 PWM0.

Bit 1: P20OP0, P20 使能输出 PWM0.
0: 禁止 P20 输出 PWM0. 默认是禁止.
1: 使能 P20 输出 PWM0.

Bit 0: P22OP0, P22 使能输出 PWM0.
0: 禁止 P22 输出 PWM0.
1: 使能 P22 输出 PWM0. 默认是使能.

PCAPWMn: PWM 模式辅助寄存器, n=0~7

SFR 页 = 0 if n = 0~1
SFR 页 = 1 if n = 6~7
SFR 页 = 0~F if n = 2~5
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**

Bit 7~6: PWMn 分别率设置位 1~0
00: 8 位 PWMn,当 [CH, CL] 计数 XXXX-XXXX-1111-1111 → XXXX-XXXX-0000-0000 时溢出激活
01: 10 位 PWMn,当[CH, CL] 计数 XXXX-XX11-1111-1111 → XXXX-XX00-0000-0000 时溢出激活
10: 12 位 PWMn,当[CH, CL] 计数 XXXX-1111-1111-111 → XXXX-0000-0000-0000 时溢出激活
11: 16 位 PWMn,当[CH, CL] 计数 1111-1111-1111-1111→ 0000-0000-0000-0000 时溢出激活

Bit 5~3: PWMn 相位设置位 2~0
000:使能 PWM 通道从 0 度开始
001:使能 PWM 通道从 90 度开始
010:使能 PWM 通道从 180 度开始
011:使能 PWM 通道从 270 度开始
100:使能 PWM 通道从 120 度开始
101:使能 PWM 通道从 240 度开始
110:使能 PWM 通道从 60 度开始
111:使能 PWM 通道从 300 度开始

缺省的 PCA PWM 模式，所有的 PWM 输出在 CL 溢出时清零。所有 PWM 同时输出低并且通过各自的 CCAPnL 设置和 CL 计数器相匹配来输出高。此模式下 PWM 表现为同相位 PWM 应为 PWM 输出总是同一时间启动。PCA 增强型 PWM 模式提供每个 PWM 通道拥有可以同时工作的不同的分别率和不同的相位延时。 下表标示了如果计数器值与比较器结果相匹配则清除 PWM 输出。PWM 输出设置条件通过{CCFnH, CCFnL}和{CH, CL}保持最初的匹配事件。如此在设定相位延时参数之后，软件仅仅当心 PWM END 计数值(PWM 输出设置)来执行不同的相位延时。

相位	0°/360°	90°	180°	270°	120°	240°	60°	300°
PWM8	00	40	80	C0	55	AA	2A	D5
PWM10	{00}00	{01}00	{10}00	{11}00	{01}55	{10}AA	{00}AA	{11}55
PWM12	000	400	800	C00	555	AAA	2AA	D55
PWM16	0000	4000	8000	C000	5555	AAAA	2AAA	D555

Bit 2: CEXn 的 PWM 反转输出。
0: PWM 不反转输出
1: PWM 反转输出

Bit 1: ECAPnH: 扩展的 MSB 位，在 8 位 PWM 模式下与 CCAPnH 一起成为一个 9 位的寄存器。同理在 10/12/16 位 PWM 模式下，他将是一个 11th/13th/17th 位寄存器。

Bit 0: ECAPnL: 扩展的 MSB 位，在 8 位 PWM 模式下与 CCAPnL 一起成为一个 9 位的寄存器。同理在 10/12/16 位 PWM 模式下，他将是一个 11th/13th/17th 位寄存器。

CMOD: PCA 计数器模式寄存器

SFR 页 = 0~F

SFR 地址 = 0xD9 复位值 = 0000-0000

7	6	5	4	3	2	1	0
CIDL	BME4	BME2	BME0	CPS2	CPS1	CPS0	ECF
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 6: BME4, PCA 模块 4/5 的缓冲模式使能。他在 PCA 模块 4/5 捕捉模式或 PWM 模式有效。
0: 禁止 PCA 模块 4/5 的缓冲模式
1: 使能 PCA 模块 4/5 的缓冲模式

Bit 5: BME2, PCA 模块 2/3 的缓冲模式使能。他在 PCA 模块 2/3 捕捉模式或 PWM 模式有效。
0: 禁止 PCA 模块 2/3 的缓冲模式
1: 使能 PCA 模块 2/3 的缓冲模式

Bit 4: BME0, PCA 模块 0/1 的缓冲模式使能。他在 PCA 模块 0/1 捕捉模式或 PWM 模式有效。
0: 禁止 PCA 模块 0/1 的缓冲模式
1: 使能 PCA 模块 0/1 的缓冲模式

PWMCR: PWM 控制寄存器

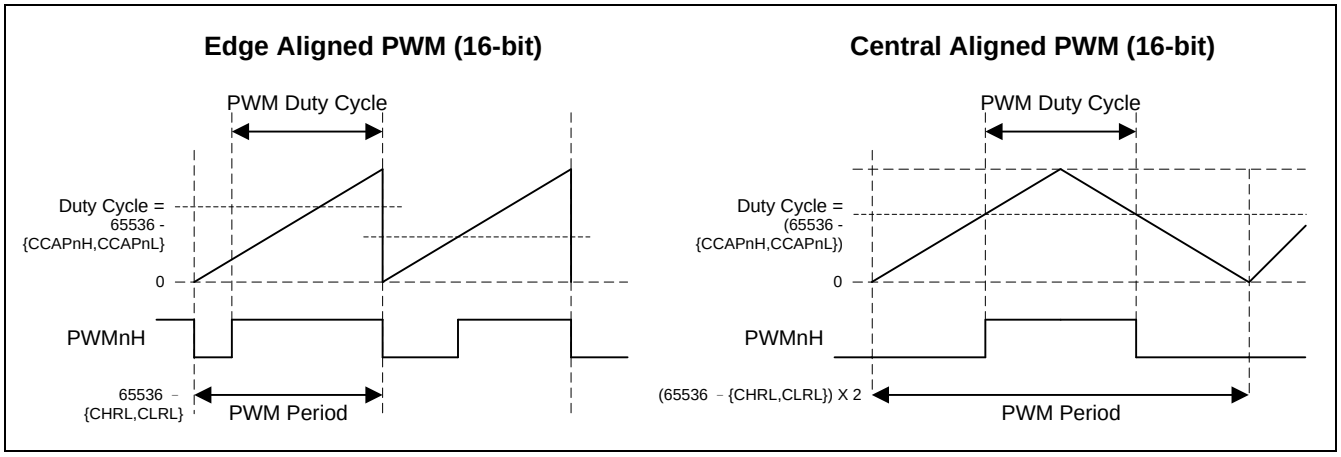
SFR 页 = 0~F

SFR 地址 = 0xBC 复位值 = 00x0-0000

7	6	5	4	3	2	1	0
PCAE	EXDT	--	PBKF	PBKM	PBKS2	PBKS1	PBKS0
R/W	**R/W**	**W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: PCAE, PWM 中心排列使能。PCAE 控制使能的 PWM 通道为中心排列调制包括 PWM 的缓冲模式或非缓冲模式。在这个 PWM 模式下，PWM 频率是半边沿排列模式。此功能仅在 PWMO0~5 激活。
0: 设置 PWM 功能为边沿排列调制
1: 使能 PWM 功能为中心排列调制

图 19–10. 边沿排列和中心排列的 PWM 波形



Bit 6: EXDT: PWM 时间的扩展死区。此功能使能将使非 PWM 通道功能改变为 PWM 通道。比如捕捉模式，软件定时器模式和高速输出模式。EXDT 控制位的波形见图 1-7。
0: 禁止 M + **2P**.
1: 使能 M + **2P** 并且使能 PWM 通道

Bit 5: 保留位。当 PWMCR 写时软件必须写“0”。

Bit 4: PBKF, PWM 终止事件标志。此位设置由 PBKS2~0 控制和软件编程。如果此位设置，则使能的 PWM 通道 0~5 将被锁住并且输出引脚保持最初的 GPIO 状态。
0: 没有 PWM 终止事件出现。仅由软件清零
1: PWM 终止事件出现或软件触发一个 PWM 终止

Bit 3: PBKM, PWM 终止模式选择
0: 锁存模式
1: 逐周期模式

图 19–11. PWM 终止控制锁存模式波形

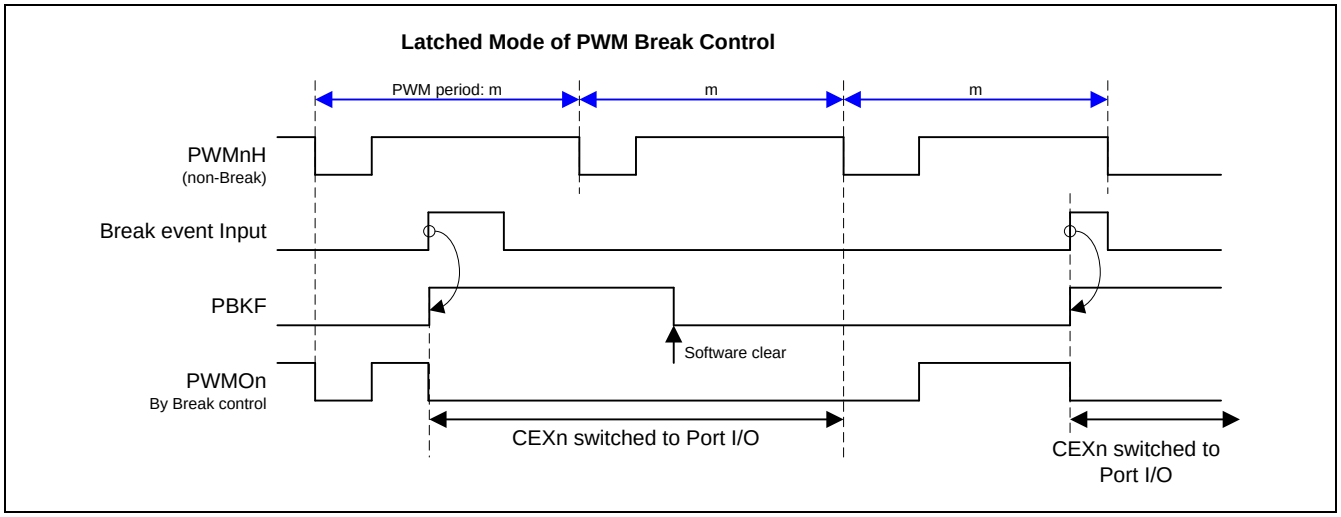
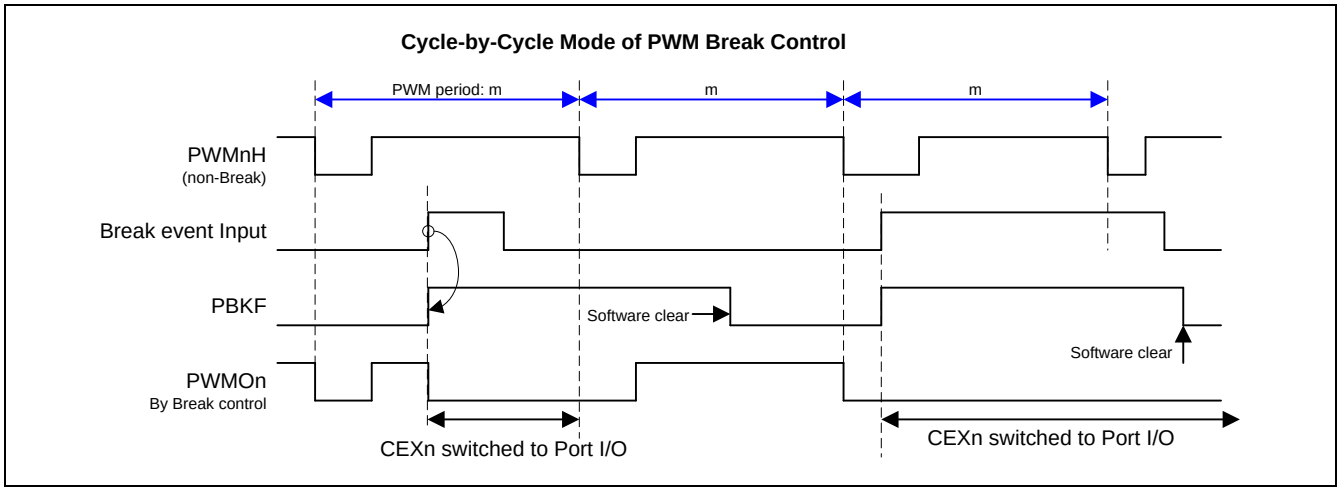


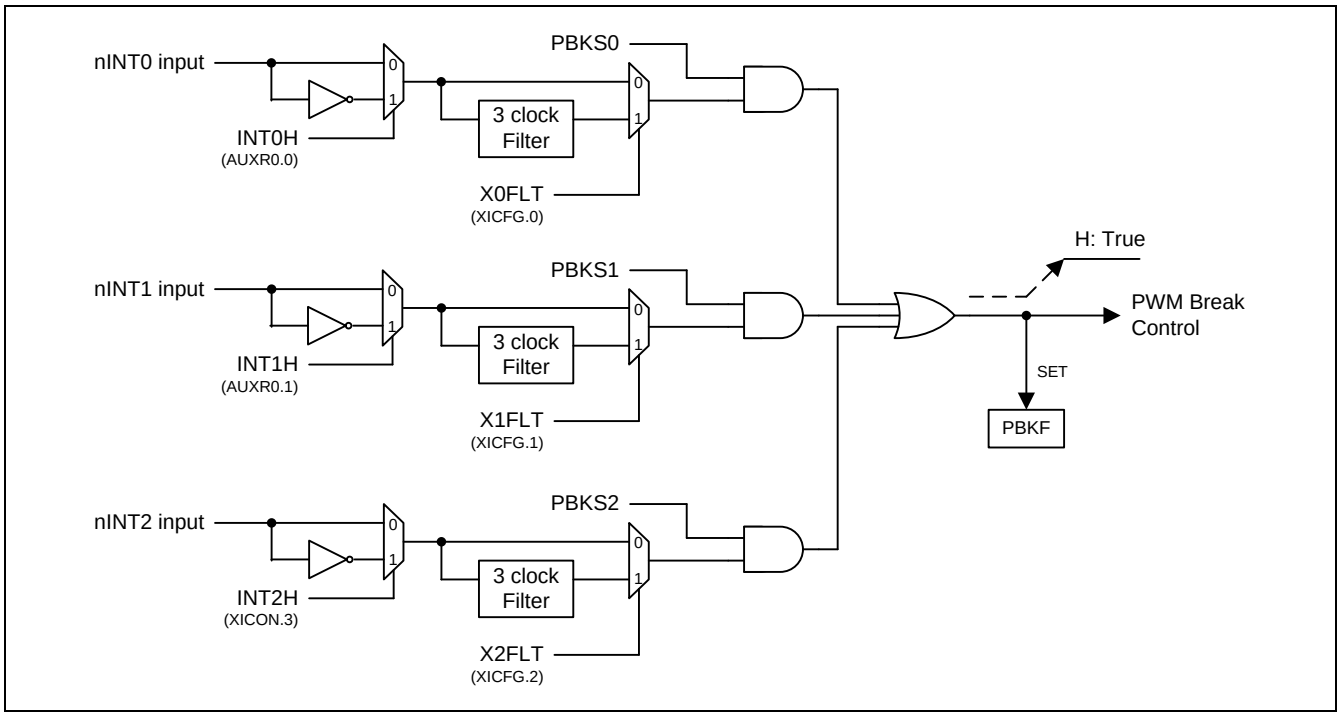
图 19–12. PWM 终止控制逐周期模式波形



Bit 2~0: PBKS2~0, PWM 终止源选择。PWM 终止功能仅在 PWMO0~5 激活。

PFCS[2:0]	PWM 终止事件源
000	禁止
001	nINT0 激活
010	nINT1 激活
011	nINT0 激活和 nINT1 激活
100	nINT2 激活
101	nINT0 激活和 nINT2 激活
110	nINT1 激活和 nINT2 激活
111	nINT0 激活和 nINT1 激活和 nINT2 激活

图 19–13. PWM 终止源



PDTCR: PWM 死区控制寄存器

SFR 页 = 0~F

SFR 地址 = 0xBD 复位值 = 0000-0000

7	6	5	4	3	2	1	0
DTPS1	DTPS0	DT5	DT4	DT3	DT2	DT1	DT0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~6: DTPS1~0,死区计数器的时钟预分频

DTPS[1:0]	预分频选择
00	SYSCLK
01	SYSCLK/2
10	SYSCLK/4
11	SYSCLK/8

Bit 5~0: DT5~0, 死区时间控制位

DT[5:0]	死区时间
00000	禁止死区
00001	预分频器 Clock X 1
00010	预分频器 Clock X 2
00011	预分频器 Clock X 3
…..	……
11110	预分频器 Clock X 30
11111	预分频器 Clock X 31

CCAPMn: PCA 模块比较/捕捉寄存器，n=0~5

SFR 页 =仅 **0**

SFR 地址 = 0xDA~0xDF 复位值 = 0000-0000

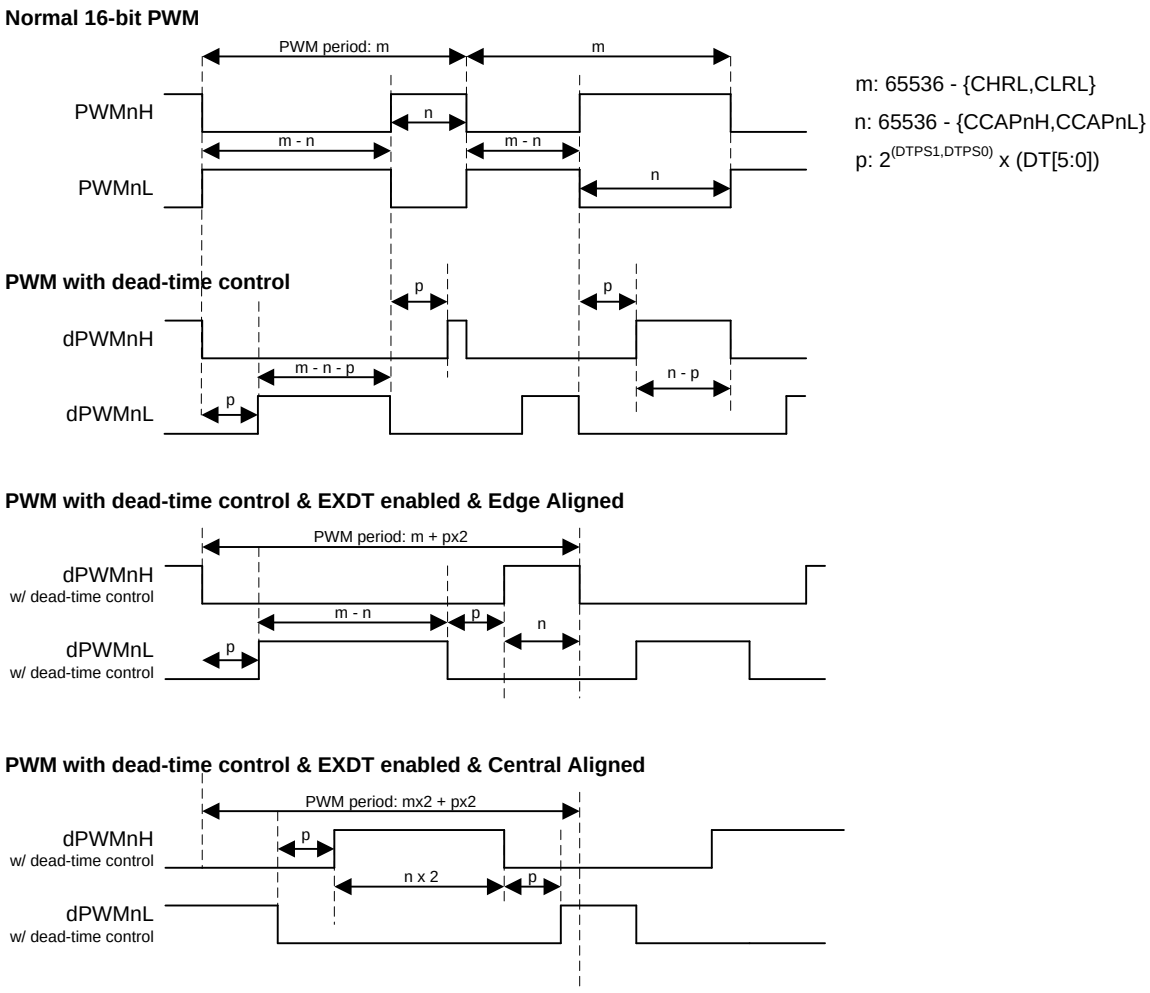
7	6	5	4	3	2	1	0
DTEn	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: DTEn. 在 PWMHn/PWMLn 执行输出对时使能死区。在 PWM 通道工作在缓冲模式下 n= 0, 2 和 4 并且死区功能激活时此位有效。通道缓冲模式由 CMOD 的 BME0, BME2 或 BME4 使能。

0: 在 PWMn/PWMn+1 的输出上(n= 0, 2, 4)禁止死区控制

1: 在 PWMn/PWMn+1 的输出上(n= 0, 2, 4)使能死区控制

图 19–14. PWM 死区控制波形



19.5. PCA 示例代码

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

汇编语言代码范例:

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

    MOV     CH,#00H                 ; 状态初始化
    MOV     CL,#00H
    MOV     CHRL,#00H               ; 重载初始化
    MOV     CLRL,#00H
    ;
    MOV     CCAPM2,#( ECOM2 + PWM2)     ; 使能 PCA 模块 2 (PWM 模式)
    MOV     CCAP2H,#0C0H            ; 25%
    MOV     CCAP2L,#0C0H

    MOV     CCAPM3,#( ECOM3 + PWM3)     ; 使能 PCA 模块 3 (PWM 模式)
    MOV     CCAP3H,#40H             ; 75%
    MOV     CCAP3L,#40H
    ;
    MOV     P2M0,#00110000B         ; 使能 P2.5 和 P2.4 上拉输出
    SETB    CR                      ; 启动 PCA 的 PWM 输出
```

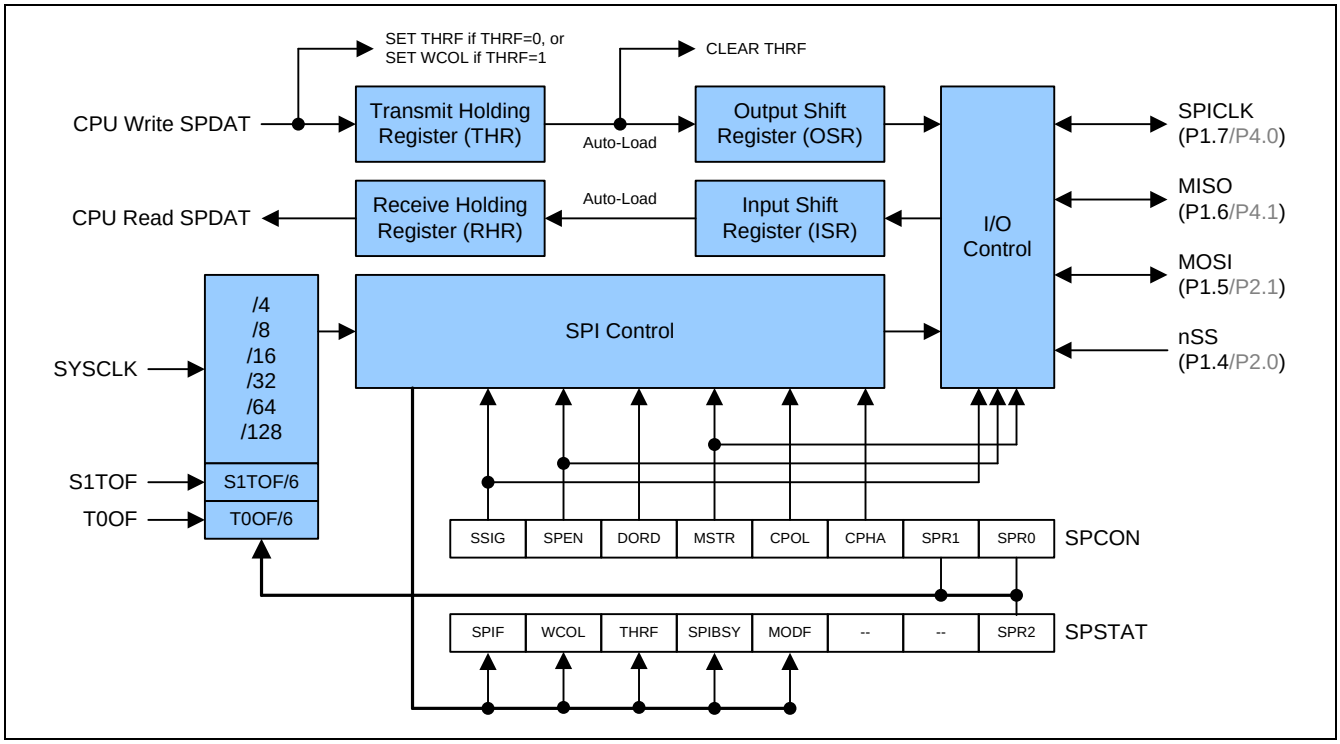
C 语言代码范例:

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

    CL = 0x00; CH = 0x00;
    CHRL = 0x00; CLRL = 0x00;       // PCA 计数器范围
    //---------------------------------------------
    CCAPM2 = ECOM2 + PWM2;          //使能 PCA 模块 2 (PWM 模式)
    CCAP2H = 0xC0; CCAP2L = 0xC0;   // 25%

    CCAPM3 = ECOM3 + PWM3;          //使能 PCA 模块 3 (PWM 模式)
    CCAP3H = 0x40; CCAP3L = 0x40;   // 75 %
    //---------------------------------------------
    P2M0 = 0x30;
    CR = 1;                         // 启动 PCA 的 PWM 输出

    while(1);
}
```

20. 串行外设接口(SPI)

MA82G5BXX 提供了一个高速串行外设接口（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。在 48MHz 的系统时钟下主机模式支持高达 12MHz 速率。在 SPI 状态寄存器(SPSTAT)里有三个标志传送完成标志(SPIF)，写冲突标志(WCOL)和模式缺陷标志(MODF)。与传统的 SPI 相比较，一个经过特别设计的发送保持寄存器（THR）显著改善了传输效率。SPI 工作下忙状态由只读标志 SPIBSY 指示。

图 20–1. SPI 模块框图



SPI 接口有 4 个引脚： MISO (P1.6), MOSI (P1.5), SPICLK (P1.7)及/SS (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。

• /SS 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/SS 引脚确定自己是否被选中。下面条件下/SS 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 (/SS) 被配置成输出
-若/SS 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:引脚输出选项见引脚功能控制寄存器 AUXR1 参考章节"5.3 功能复用"。

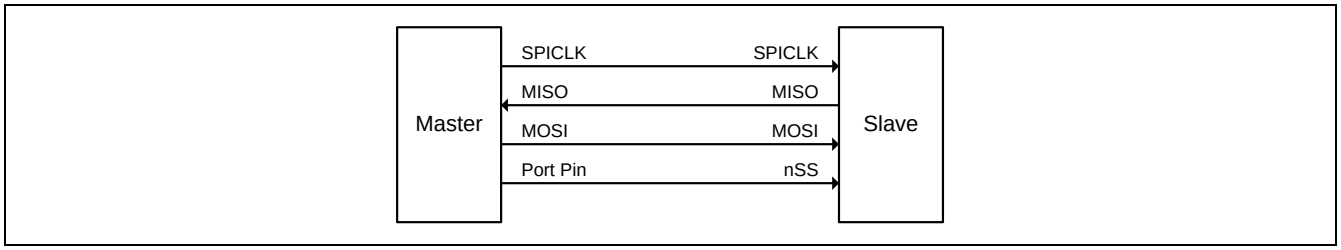
注意，即使 SPI 被配置成主机运行(MSTR=1)， 它仍然可以被/SS 引脚的低电平拉成从机(若 SSIG=0)，一旦发生这种情况， SPIF 位(SPSTAT.7)置位。 (参考章节"20.2.3 nSS 引脚的模式改变") 。

20.1. 典型 SPI 配置

20.1.1. 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。

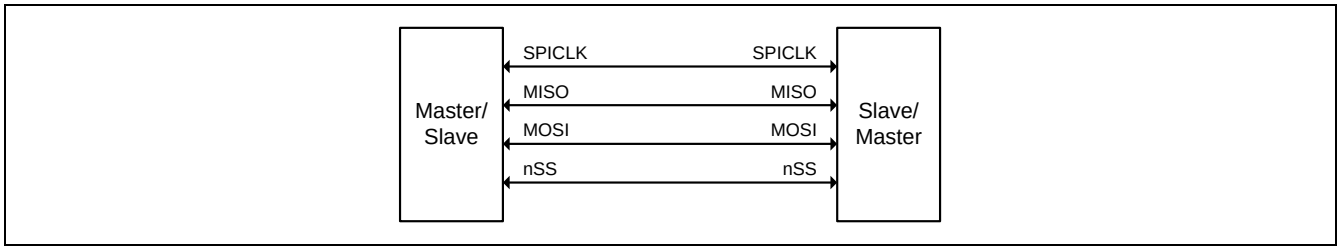
图 20–2. SPI 单主机和单从机框图



20.1.2. 双驱动器，既是主机也是从机

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。(参考“20.2.3 nSS 引脚的模式改变”) 。

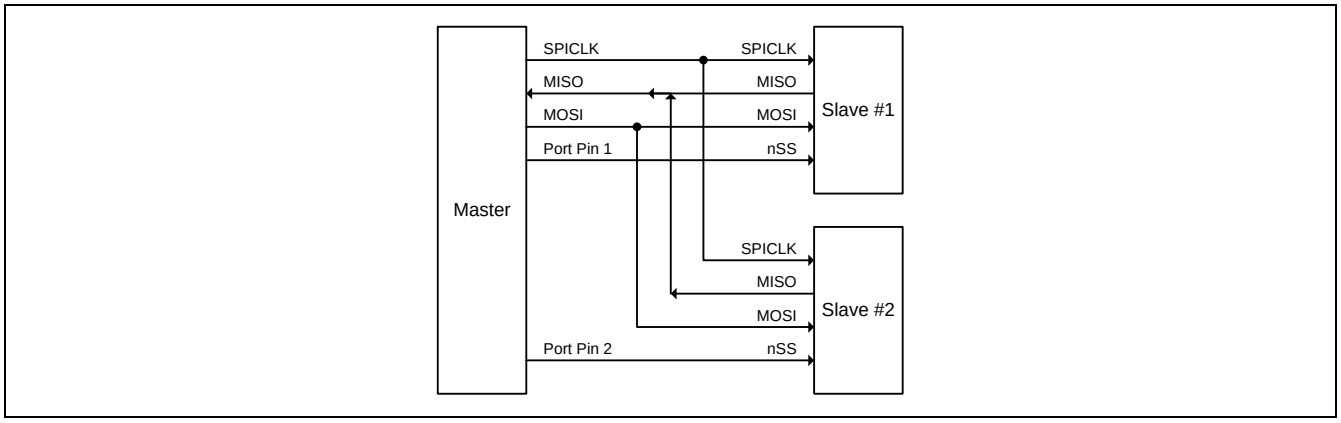
图 20–3. SPI 双驱动器框图，既是主机也是从机



20.1.3. 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/SS 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 20–4. SPI 单主机多从机框图



20.2. 配置 SPI

表 20–1 既是主机/从机配置又是模式用法和方向。

表 20–1. SPI 主机和从机选择

SPEN (SPCTL.6)	SSIG (SPCTL.7)	/SS -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 (激活)		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”.

20.2.1. 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。

20.2.2. 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。

20.2.3. nSS 引脚的模式改变

若 SPEN=1， SSIG=0， MSTR=1 且 /SS 引脚=1， SPI 使能在主机模式，这种情况下，其他主机可以将/SS 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。

20.2.4. 发送保持寄存器非空标志

为了提高 SPI 发送速度一个特殊设计保持寄存器(THR)可以减短 CPU 数据移动字节与字节传送的延迟时间。THRF 置位表明 THR 的数据是有效的并且等待发送。如果 THR 是空的(THRF=0)，软件写一个字节数据到 SPDAT 数据将存储在 THR 并且 THRF 置位。如果输出移位寄存器(OSR)是空的，硬件立刻将 THR 数据移到 OSR 并且 THRF 清零。在 SPI 主机模式，OSR 有效数据将触发 SPI 发送。在 SPI 从机模式，OSR 有效数据等待另一个 SPI 主机移出数据。如果 THR 是非空(THRF=1)，软件写一个字节数据到写冲突标志 WCOL (SPSTAT.6)将置位。

20.2.5. 写冲突

MA82G5BXX 的 SPI 在发送方向和接收方向是双缓冲数据器。发送数据在 THR 空时才能写入到缓冲器 THR。只读标志 THRF 表示 THR 是空或非空。在 THRF 为”1”时数据寄存器被写入数据冲突标志 WCOL (SPSTAT.6)将置位。这种情况下，SPDAT 写入操作将被忽略。

主机或从机检测到写冲突时，主机异常是主机传输过程中有非空控制；从机是在主机初始化传输没有控制结束时出现冲突 。

WCOL 软件写”1”清零。

20.2.6. SPI 时钟速率选择

SPI 时钟频率选择（主机模式）使用 SPCON 寄存器的 SPR1 和 SPR0 位及 SPSTAT 寄存器的 SPR2 来设置，如表 20–2 所示。

表 20–2. SPI 串行时钟速率

SPR2	SPR1	SPR0	SPI 时钟选择	SPI 时钟速率@ SYSCLK=12MHz	SPI 时钟速率@ SYSCLK=48MHz
0	0	0	SYSCLK/4	3 MHz	12 MHz
0	0	1	SYSCLK/8	1.5 MHz	6 MHz
0	1	0	SYSCLK/16	750 KHz	3 MHz
0	1	1	SYSCLK/32	375 KHz	1.5 MHz
1	0	0	SYSCLK/64	187.5 KHz	750 KHz
1	0	1	SYSCLK/128	93.75 KHz	375 KHz
1	1	0	**S1TOF/6**	可变的	可变的
1	1	1	**T0OF/6**	可变的	可变的

注意:

1. *SYSCLK 是系统时钟*
2. *S1TOF 是 UART1 波特率产生器溢出*
3. *T0OF 是定时器 0 溢出*

20.3. 数据模式

时钟相位(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 20–5. SPI 从机传送格式在(CPHA=0)

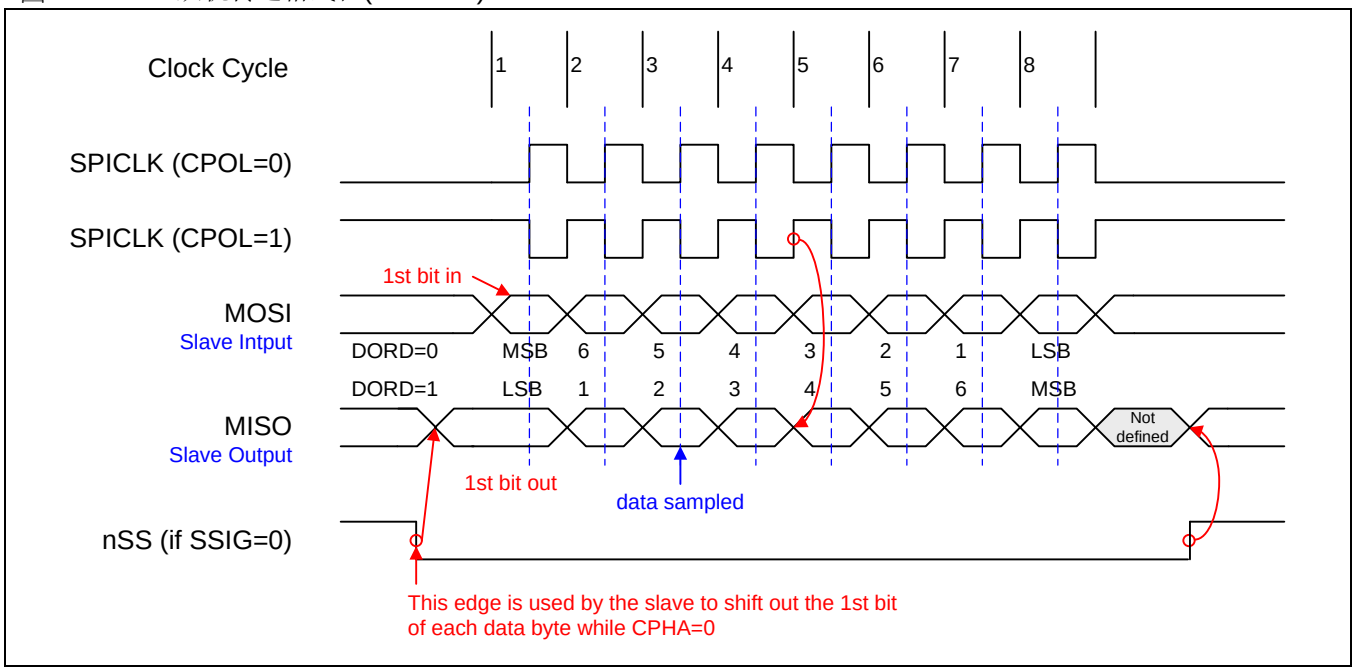


图 20–6. 从机传送格式在(CPHA=1)

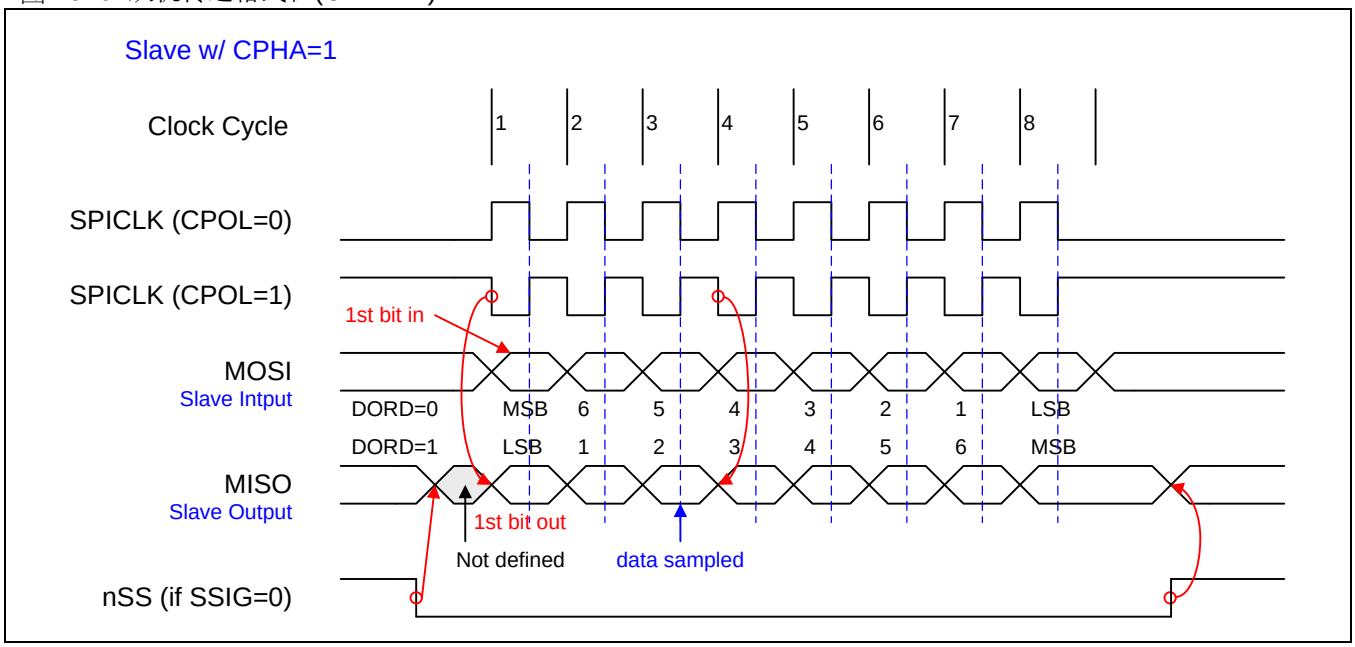


图 20–7. SPI 主机传送格式在(CPHA=0)

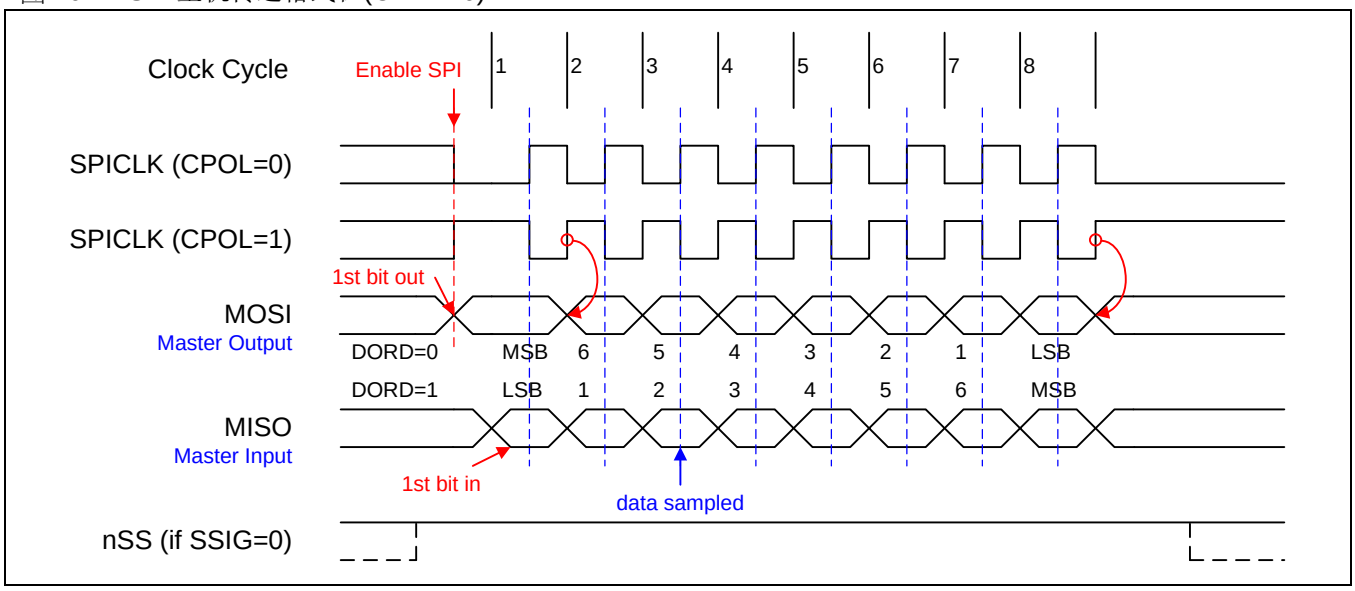
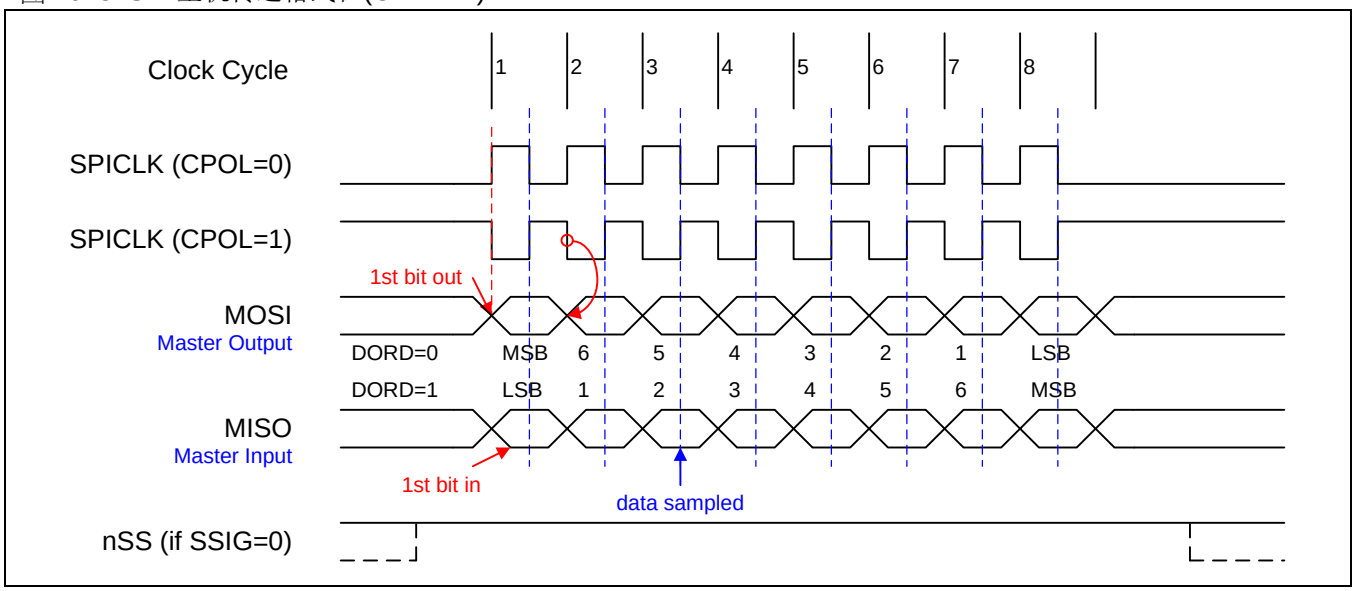


图 20–8. SPI 主机传送格式在(CPHA=1)



20.4. SPI 寄存器

下面是 SPI 操作的相关特殊功能寄存器：

SPCON: SPI 控制寄存器

SFR 页 = 0~F

SFR 地址 = 0x85 复位值= 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, 忽略 nSS
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 使能

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: SPI 从机模式
1: SPI 主机模式

Bit 3: CPOL, SPI 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: /SS 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

Bit 1~0: SPR1-SPR0, SPI 时钟速率选择位 0 和 1 (主机模式，与 SPR2 配合使用)

SPR2	SPR1	SPR0	SPI 时钟选择	SPI 时钟速率@ SYSCLK=12MHz	SPI 时钟速率@ SYSCLK=48MHz
0	0	0	SYSCLK/4	3 MHz	12 MHz
0	0	1	SYSCLK/8	1.5 MHz	6 MHz
0	1	0	SYSCLK/16	750 KHz	3 MHz
0	1	1	SYSCLK/32	375 KHz	1.5 MHz
1	0	0	SYSCLK/64	187.5 KHz	750 KHz
1	0	1	SYSCLK/128	93.75 KHz	375 KHz
1	1	0	**S1TOF/6**	可变的	可变的
1	1	1	**T0OF/6**	可变的	可变的

注意:

1. SYSCLK 是系统时钟
2. S1TOF 是 UART1 波特率定时器溢出
3. T0OF 是定时器 0 溢出

SPSTAT: SPI 状态寄存器

SFR 页 = 0~F

SFR 地址 = 0x84 复位值= 0000-0XX0

7	6	5	4	3	2	1	0
SPIF	WCOL	THRF	SPIBSY	MODF	--	--	SPR2
R/W	**R/W**	**R**	**R**	**W**	**W**	**W**	**R/W**

Bit 7: SPIF, SPI 传输完成标志

0:**软件写"1"此位清零**

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:软件写"1"此位清零

1: SPI 数据寄存器（SPDAT）在数据传输过程中被写入此位置位(见章节"**錯誤! 找不到參照來源。** 写冲突")

Bit 5: THRF,发送保持寄存器（THR）非空标志。只读

0:表明 THR 是"空的"。当 THR 为空时此位被硬件清零，这意味着 THR 中的数据已经被装入移位输出寄存器进行发送，而现在用户可以向 SPDAT 写下一个要发送的数据

1:表明 THR 是"非空"。当软件向 SPDAT 写数据时由硬件置位

Bit 4, SPIBSY, SPI 忙标志。只读

0:表示 SPI 是待机状态并且所有的移位寄存器是空的

1: 置位表示 SPI 传输进行中（主机或从机）

Bit 3: 模式缺陷标志(校验中)

Bit 2~1: 保留。当 SPSTAT 写入时，这些位软件必须写"0"

Bit 0: SPR2, SPI 时钟速率选择 2(与 SPR1 和 SPR2 配合使用)

SPDAT: SPI 数据寄存器

SFR 页 = 0~F

SFR 地址 = 0x86 复位值= 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发送和接收过程中各自独立写入和读取。

AUXR1:辅助控制寄存器 1

SFR 页 = 0~F

SFR 地址 = 0xA2 复位值 = 0000-0000

7	6	5	4	3	2	1	0
P1KBIH	P3KBIL	**P4SPI**	P3S1	P3S1MI	P6TWI	P3CEX	DPS
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 5: P4SPI, SPI 接口在 P4.1~P4.0 和 P2.1~P2.0

P4SPI	nSS	MOSI	MISO	SPICLK
0	P1.4	P1.5	P1.6	P1.7
1	P2.0	P2.1	P4.1	P4.0

20.5. SPI 示例代码

(1). Required Function: Set SPI Master write/read

汇编语言代码范例:

```
        MOV         SPCON,#( SPEN | SSIG | MSTR)  ;enable SPI and set sampling data at rising edge,
                                                  ;SPICLK is sysclk/ 4.
        MOV         P1M0,#0B0H                    ; set P14 to push-pull
    CLR     P14                 ; enable slave device select
        MOV         SPDAT,#55H                    ; SPI send Addr=0x55 to slave
        MOV         a,#20H
check_THRF_0:
    ANL     a,SPSTAT
    JNZ     check_THRF_0

        MOV         SPDAT,#0AAH                   ; SPI send Data=0xAA to slave;
        MOV         a,#10H
check_SPIBSY_0:
    ANL     a,SPSTAT
    JNZ     check_SPIBSY_0
    SETB    P14                 ; disable slave device select

    CLR     P14                 ; enable slave device select
        MOV         SPDAT,#55H                    ; SPI send Addr=0x55 to slave
        MOV         a,#20H
check_THRF_0:
    ANL     a,SPSTAT
    JNZ     check_THRF_0

        MOV         SPDAT,#0FFH                   ; SPI send Data=0xff dummy data, and read back data
        MOV         a,#10H
check_SPIBSY_0:
    ANL     a,SPSTAT
    JNZ     check_SPIBSY_0
    SETB    P14                 ; disable slave device select

   MOV      A,SPDAT
   ;SPDAT=read back Data
```

C 语言代码范例:

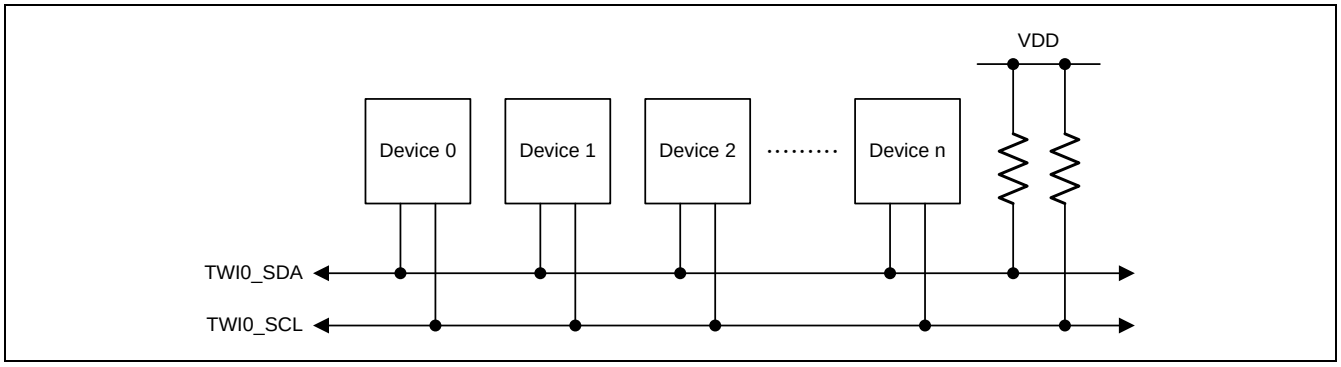
```
#define  nCS                                    P14
void main(void)
{
    Unsigned char SPI_read_Data;

        SPCON = ( SPEN | SSIG | MSTR);          //enable SPI and set sampling data at rising edge, SPICLK is sysclk
/ 4.
        P1M0 = 0xB0;                                    //set P14 to push-pull
        nCS = 0;                    //enable slave device select
        SPDAT = 0x55;                   // SPI send Addr=0x55 to slave;
        while(SPSTAT & THRF);
        SPDAT = 0xAA;                    //SPI send Data=0xAA to slave;
        while(SPSTAT & SPIBSY);
        nCS = 1;                    //disable slave device select
    //;
        nCS = 0;                    //enable slave device select
        SPDAT = 0x55;                    // SPI send Addr=0x55 to slave;
        while(SPSTAT & THRF);
        SPDAT = 0xFF;                    // SPI send Data=0xff dummy data, and read back data
        while(SPSTAT & SPIBSY);
        nCS = 1;                    //disable slave device select

        SPI_read_Data = SPDAT;

        while (1);
}
```

21. 双线串行接口(TWI0 和 TWI1)

双线串行接口是一个双线双向总线。双线串行接口(TWI0)很适合于典型的处理器应用。**MA82G5BXX** 锲入了两个独立的双线串行接口硬件引擎(TWI0 和 TWI1)。TWI1 是跟 TWI0 一样的设计除了不同的 SFR 访问页和不同的端口引脚之外具有全兼容的控制流。所有 TWI0 的 SFR 都在 SFR 0 页并且接口引脚是 TWI0_SCL 和 TWI0_SDA。所有 TWI1 的 SFR 都在 SFR 1 页并且接口引脚是 TWI1_SCL 和 TWI1_SDA。

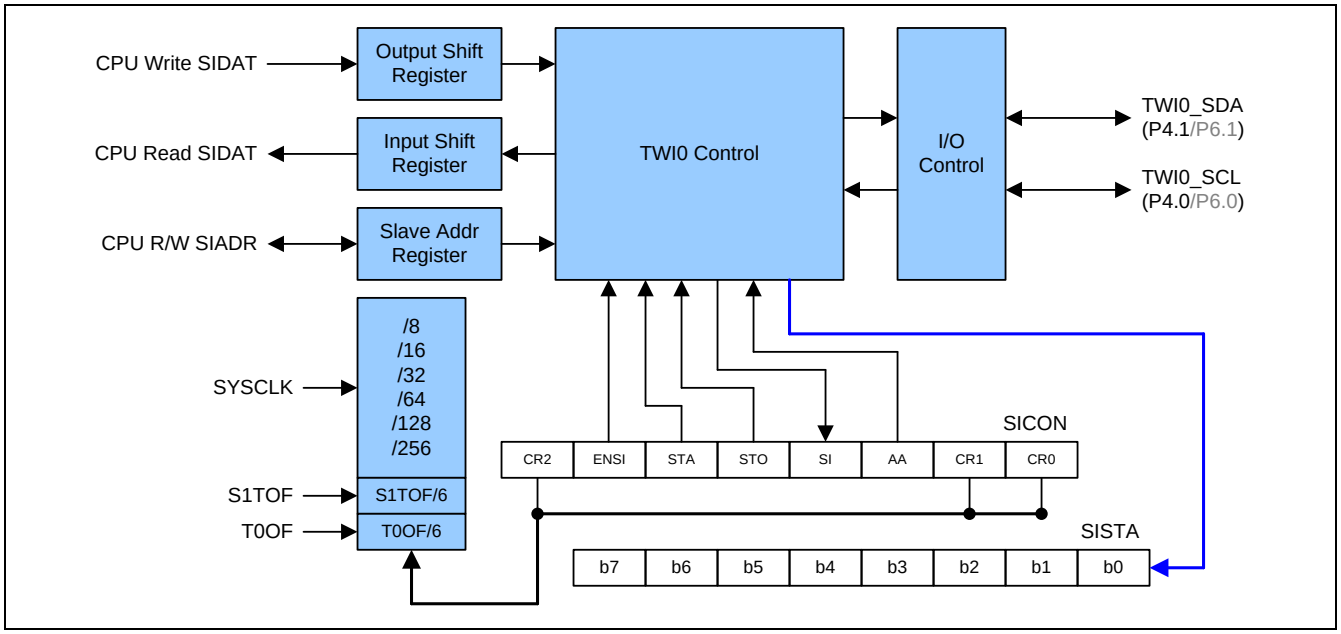
TWI0 协议允许系统设计人员只用两根双向传输线来连接多达 128 个不同的设备，一根用于时钟（TWI0_SCL），一根用于数据（TWI0_SDA）。双线串行接口(TWI0)由 TWI0_SDA（串行数据，P4.1）TWI0_SCL（串行时钟，P4.0）控制产生和同步，仲裁逻辑以及起始/停止（START/STOP）。唯一需要的外部硬件就是在 TWI0 的每根传输线上添加一个上拉电阻。所有连接到总线的设备都有自己的地址，而且 TWI0 协议解决了总线仲裁的问题。

图 21–1. TWI0 总线互联框图



TWI0 总线可以操作在主机或从机也可以是多主机。CPU 通过 SIADR（串行接口从机地址寄存器）、SIDAT（串行接口数据寄存器，用于发送和接收 TWI0 数据）、SICON（串行接口控制寄存器）、SISTA（串行接口状态寄存器）这四个特殊功能寄存器与 TWI0 相连。TWI0 硬件通过两根数据线与串行总线相连：TWI0_SDA（串行数据线）、TWI0_SCL（串行时钟线）。

图 21–2. TWI0 功能框图



21.1. 操作模式

TWI0 有 4 种操作模式： 1) 主机/发送模式， 2)主机/接收模式， 3)从机/发送模式 4)从机/接收模式。SI 软件清零之后 SICON 寄存器的位 STA, STO 和 AA 决定 TWI0 硬件下一个执行的是哪一个操作。当下一个操作完成，SISTA 寄存器将更新一个新状态同时 SI 也会硬件置位。现在，中断服务程序会被调用（如果 TWI0 中断使能），软件可以通过新的状态区分需要调用哪一个子程序。

21.1.1. 主机发送模式

在主机发送模式，一定数量的字节数据可以发送到一个从机接收器。在进入主机发送模式前，SICON 必须作如下设置：

SICON

7	6	5	4	3	2	1	0
CR2	ENSI	STA	STO	SI	AA	CR1	CR0
Bit rate	1	0	0	0	x	Bit rate	

CR0、 CR1和CR2定义了串行位速率。ENSI必须设置为逻辑1来使能TWI0。如果AA位复位，在其它设备成为总线的主机时，TWI0将不会应答它自身的从机地址或广播地址。也就是说，如果AA复位，TWI0不能进入从机模式。STA、STO与SI必须复位。

置位STA也许可以立即进入主机发送模式。TWI0逻辑将检测串行总线并且在总线空闲时产生一个START信号。发送完START信号后，串行中断标志（SI）将被置位，并且状态寄存器（SISTA）中的状态编码将为08H。这个状态编码必须用于指示一个中断服务程序加载从机地址和数据方向位（SLA+W）到SIDAT。SICON的SI位必须清零，串行传输才能继续进行。

当从机地址与方向位发送完，并且接收到一个应答位后，串行中断标志（SI）会再次被置位。SISTA可能为以下的编码：在主机模式为18H，20H或38H，如果从机模式使能（AA=1），也可以为68H，78H或B0H。在这些状态编码下对应的操作将在随后的工作流程图中详细叙述。在一个REPEATED START信号后（状态编码10H），TWI0可以通过向SIDAT写入SLA+R进入主机接收模式。

21.1.2. 主机接收模式

在主机接收模式，可以从从机发送器接收一定数量的字节数据。SICON也必须如主机发送模式一样初始化。开始信号发送后，中断服务程序必须向SIDAT写入7位从机地址与数据方向位（SLA+R）。SICON的SI位必须清零，串行传输才能继续进行。

在从机地址与数据方向位发送完并且接收到应答位后，串行中断标志（SI）重新置位。SISTA可能为以下的编码：在主机模式为40H，48H或38H，如果从机模式使能（AA=1），也可以为68H，78H或B0H。这些状态编码下对应的操作将在随后的工作流程图中详细叙述。在一个REPEATED START信号后（状态编码10H），TWI0可以通过向SIDAT写入SLA+W进入主机接收模式。

21.1.3. 从机发送模式

在从机发送模式下，许多数据发送给主机接收。SIADR 和 SICON 必须如下初始化从机发送模式：

SIADR

7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	GC

|<---------- Own Slave Address ---------->|

高 7 位是响应被主机寻址的 TWI0 地址。如果 LSB (GC)置位，TWI0 将应答广播地址(00H)；否则将忽略广播地址。

SICON

7	6	5	4	3	2	1	0
CR2	ENSI	STA	STO	SI	AA	CR1	CR0
x	1	0	0	0	1	x	x

在从机模式下 CR0, CR1 和 CR2 不影响 TWI0 。ENSI 必须置位去使能 TWI0。AA 必须置位去使能 TWI0 应答自己的从机地址或广播地址。STA, STO 和 SI 必须清零。

当SIADR和SICON初始化之后，TWI0会等待直到其从机地址被寻址并且数据方向为”1”（R），TWI0将工作于从机发送模式。在接收到自身的从机地址以及”R”位后，串行中断标志（SI）置位，并且可以从SISTA读出一个可用的状态编码。这些状态编码可以用作指示一个中断服务程序，在这些状态编码下对应的操作将在随后的工作流程图详细叙述。当TWI0处于主机模式时，如果仲裁失败也可能进入从机发送模式（参考B0H状态）。

如果在一次传输的过程中 AA 位复位，TWI0 将发送完当前字节的数据后进入 C0H 或 C8H 状态。TWI0 会转换到未被寻址从机模式，如果主机继续传输，TW0I 将会忽略主机接收器，因此主机总是接收到”1”。当 AA 复位时，TWI0 不会回应其从机地址或广播地址，但是会继续监听串行总线。在任何时候可以通过置位 AA 恢复，这意味着 AA 位可用于暂时从总线中隔离 TWI0。

21.1.4. 从机接收模式

在从机接收模式，会从主机发送器接收一定数量的字节数据。数据传送的初始化与从机发送模式一样。

SIADR与SICON初始化后，TWI0会等待直到其从机地址被寻址并且数据方向为”0”（W），TWI0将工作于从机接收模式。在接收到其从机地址与”W”位后，串行中断标志（SI）置位，并且可以从SISTA读出一个可用的状态编码。这些状态编码可以用作指示一个中断服务程序，在这些状态编码下对应的操作将在随后的工作流程图详细叙述。当TWI0处于主机模式时，如果仲裁失败可能进入从机接收模式（参考状态68H和78H）。

如果在一次传输的过程中AA位被复位，TWI0会在接收到下一个字节后回复NACK（逻辑1）。当AA复位时，TWI0不会响应自身的从机地址或广播地址，但是会继续监听串行总线。在任何时候可以通过置位AA恢复，这意味着AA位可用于暂时从总线中隔离TWI0。

21.2. 混合状态

有两个SISTA编码没有与已经定义的TWI0硬件状态对应，描述如下：

S1STA = F8H:
这个状态编码表明还没有相应的信息可用，因为串行中断标志（SI）还没有置位。这种情况发生在状态转换之间和TWI0未涉及串行传输时。

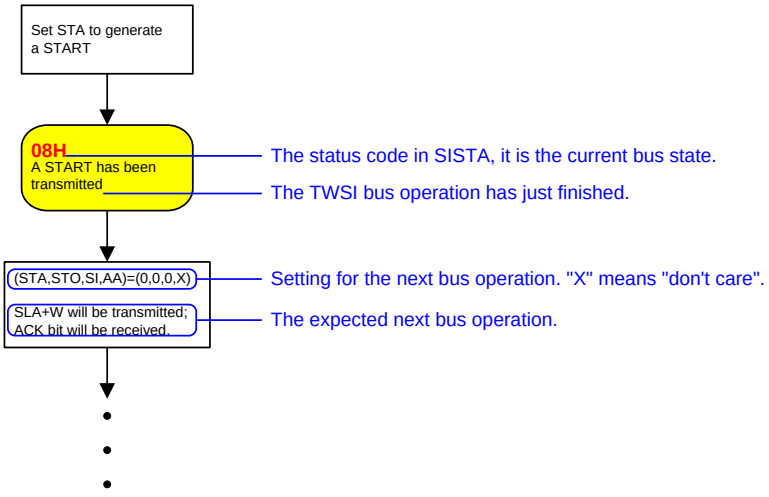
S1STA = 00H:
这个状态编码表明在一个TWI0串行传输过程中发生总线错误。当一个START或STOP信号在一帧的不合法位置发送时，总线错误就会发生。例如：在传输一个字节地址、数据时，或者在应答位。总线错误也会在外界干扰扰乱内部TWI0信号时发生。当总线错误发生时，SI被置位，STO标志必须置位并且SI必须软件清零用来从总线错误中恢复。这会使TWI0进入“未被寻址”（not-addressed）从机状态（已定义的状态）并且清除STO标志（SICON的其它位不受影响）。TWI0_SDA与TWI0_SCL线将被释放（不会发送STOP信号）。

21.3. 使用 TWI0

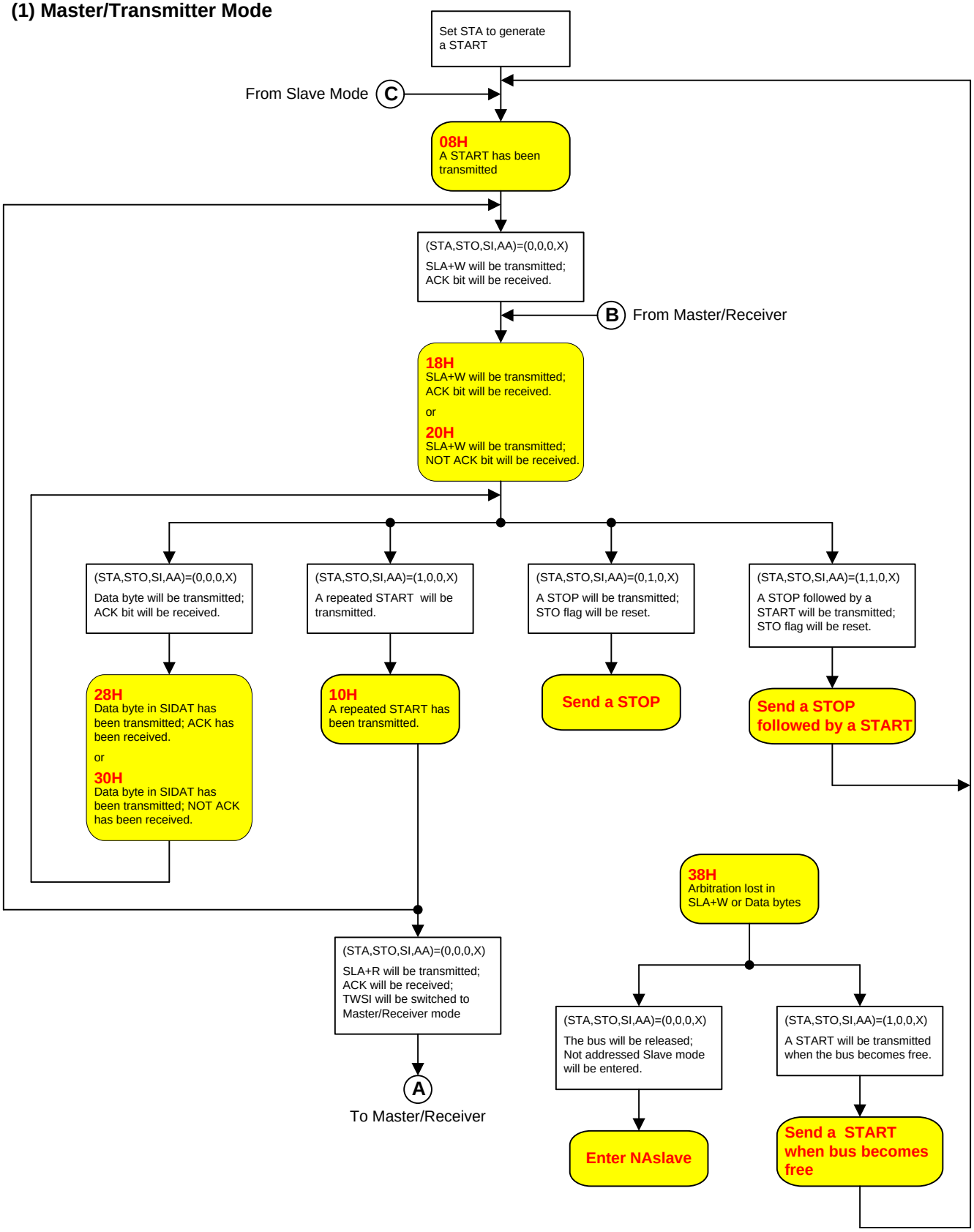
TWI0 是面向字节并且基于中断的。中断会在所有总线事件后发生，例如接收到一字节数据或发送 START 信号。因为 TWI0 是基于中断的，应用软件可以自由的在一个 TWI0 字节发送的过程中处理其它工作。注意，TWI0 中断使能（EIE1.6）与 EA 位允许应用程序选择在 SI 标志出现时是否产生中断请求。当 SI 标志出现时，表明 TWI0 已经完成一个操作并且等待程序响应。此时状态寄存器 SISTA 保存的状态编码表明 TWI0 总线的当前状态。用户程序可以通过对 STA，STO 和 AA 位（在 SICON 中）进行适当的编程来决定接下来 TWI0 总线将如何运行。

下面的操作流程图将指导用户通过“状态到状态”（state-by-state）的操作来使用 TWI0。首先，用户应该向 SIADR 写入自身的从机地址（参考前面对 SIADR 的描述）。作为主机时，在初始 SICON 后，第一步为置位“STA”来向总线产生一个 START 信号。作为从机时，在初始化 SICON 后，TWI0 等待直到被寻址。然后参考操作流程图对 SICON 的 STA，STO，SI，AA 位进行适当的编程来进行后续动作。当 SI 清零后 TWI0 硬件就会进行下一步动作，因此推荐使用如下两个步骤：先对 STA，STO 与 AA 编程，然后清零 SI 位（可以使用“CLR SI”指令）来进行可靠的操作。

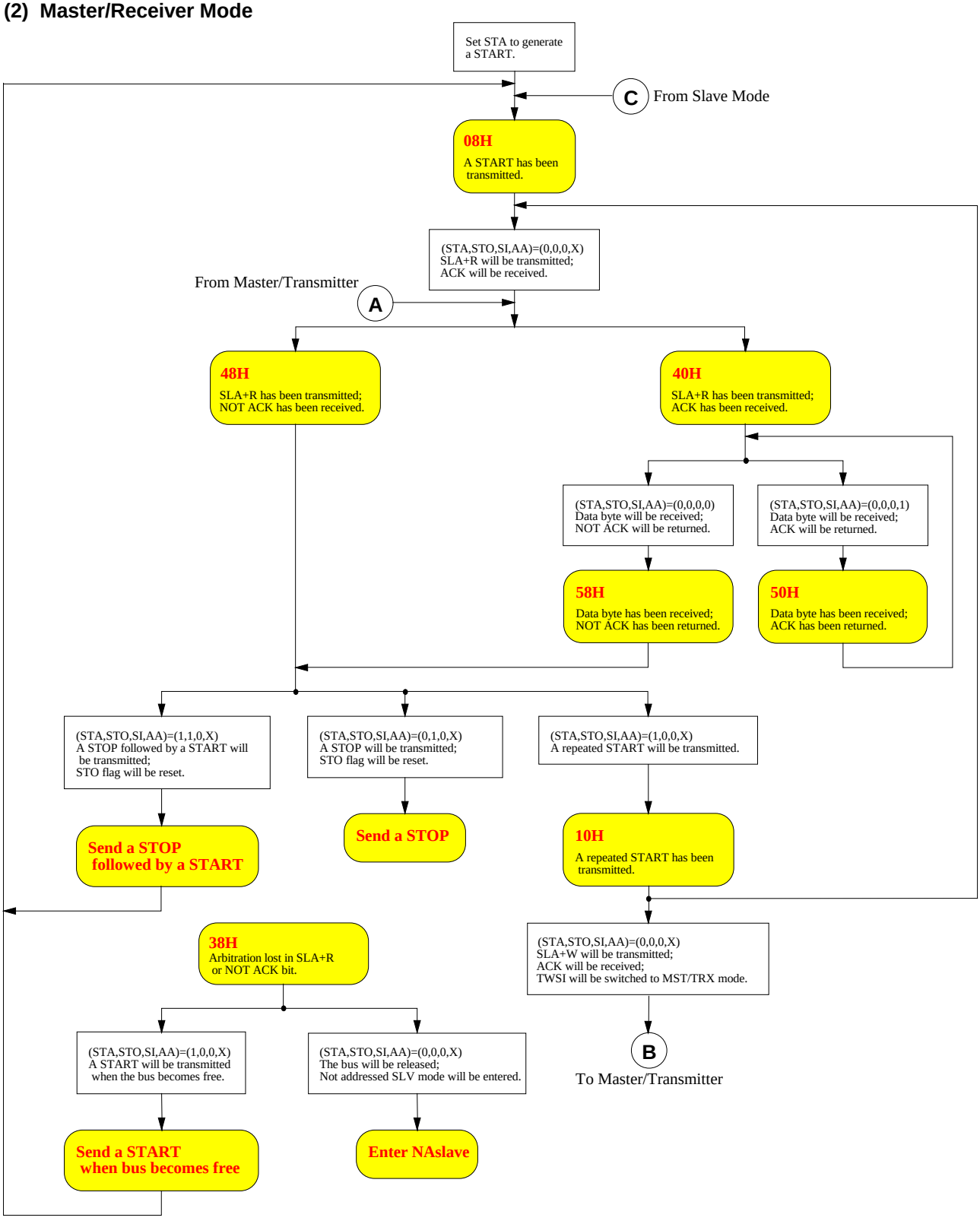
下面的图指出如何阅读流程图。



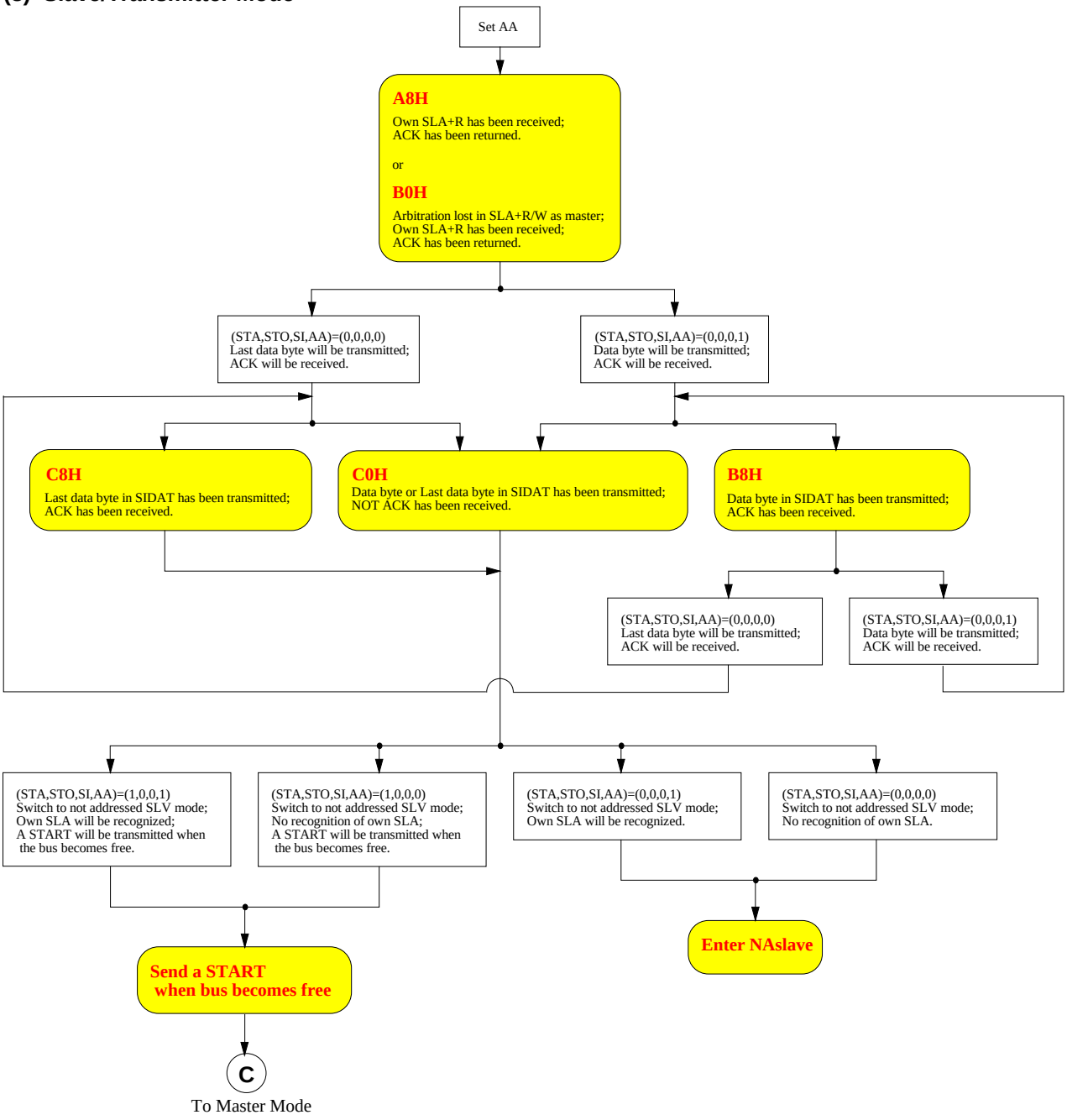
(1) Master/Transmitter Mode



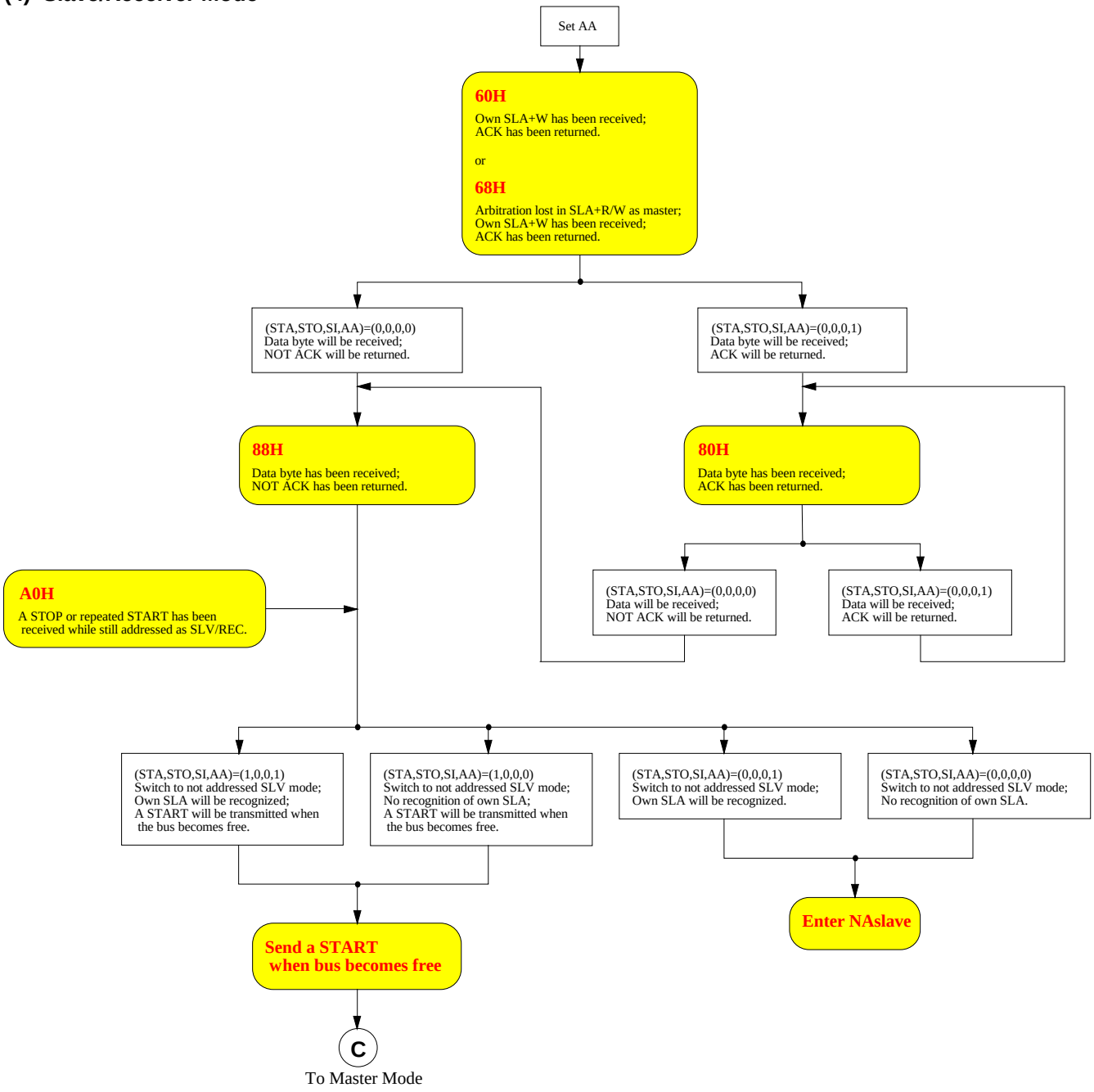
(2) Master/Receiver Mode



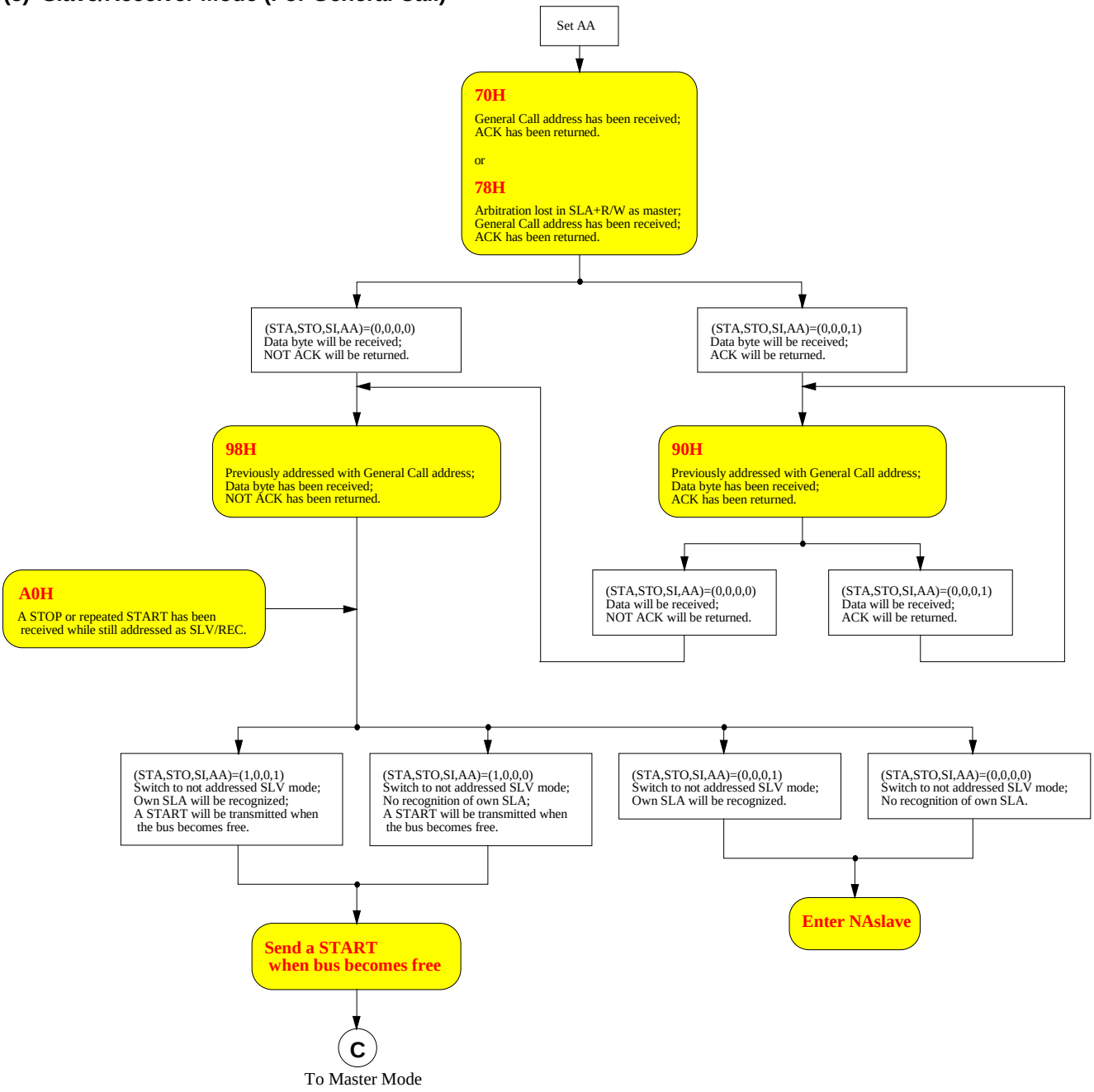
(3) Slave/Transmitter Mode



(4) Slave/Receiver Mode



(5) Slave/Receiver Mode (For General Call)



21.4. TWI0 寄存器

SIADR:双线串行接口地址寄存器

SFR 页 = **0**

SFR 地址 = 0xD1 复位值= 0000-0000

7	6	5	4	3	2	1	0
A6	A5	A4	A3	A2	A1	A0	GC
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

CPU 可以直接对此寄存器进行读写。SIADR 不受 TWI0 硬件的影响。当 TWI0 处于主机模式时此寄存器的值会被忽略。当处于从机模式时，寄存的高七位必须被用于本机的从机地址，并且当最低位（GC）置位时，广播地址（00H）会被识别，否则忽略。在 START 状态后，最高位与从 TWI0 总线上收到的首位相对应。

SIDAT: 双线串行接口数据寄存器

SFR 页 = **0**

SFR 地址 = 0xD2 复位值= 0000-0000

7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

此寄存器保存着一字节将要发送或者刚接收到的数据。在没有进行移位工作时，CPU 可以直接对此寄存器进行读写。这种情况发生在 TWI0 正处于一个被定义的状态并且串行中断标志位（SI）置位。只要 SI 被置位，SIDAT 中的数据总是保持稳定的。在数据被移出时，总线上的数据同时移入，SIDAT 总保存着总线上出现的最后一个字节数据。因此在仲裁失败时，主机切换为从机的过程会在 SIDAT 中产生一个正确的数据。

SIDAT 与 ACK 标志位组成一个 9 位的移位寄存器，可以在移入或移出一个 8 位的数据后，跟随一个应答位。ACK 标志由 TWI0 硬件控制，CPU 访问不到。串行数据在 TWI0_SCL 的上升沿移入 SIDAT 寄存器。当一字节的数据完全移入 SIDAT 后，SIDAT 中的数据将是可以用的，并且控制逻辑会在第 9 个时钟周期返回一个应答位。串行数据在 TWI0_SCL 的下降沿从 SIDAT 寄存器移出。

CPU 向 SIDAT 写入数据后，SD7 位将首先出现在 TWI0_SDA 线上。9 个时钟周期后，SIDAT 中的 8 位数据将被发送完成，并且通过应答位返回 ACK 标志。注意发送出去的 8 位数据会移回 SIDAT。

SICON: 双线串行接口控制寄存器

SFR 页 = **0**

SFR 地址 = 0xD4 复位值= 0000-0000

7	6	5	4	3	2	1	0
CR2	ENSI	STA	STO	SI	AA	CR1	CR0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

CPU 可以直接读写此寄存器。其中两个位会受 TWI0 硬件的影响：SI 位会在串行中断请求时置位，STO 位会在总线出现 STOP 状态时清零。STO 位也会在 ENS1=0 时清零。

Bit 7: CR2, TWI0 时钟速率选择位 2 (与 CR1 和 CR0 一起使用)

Bit 6: ENSI, TWI0 硬件使能位
ENSI 为 "0"时，TWI0_SDA 与 TWI0_SCL 输出为高阻态，TW0I_SDA 与 TWI0_SCL 输入信号被忽略，TWI0 处于被寻址（not-addressed）从机状态，SICON 的 STO 位被强制置为”0”，但不影响其它位。TWI0_SDA 与 TWI0_SCL 可用作通用 I/O 引脚。ENSI 为 "1"时，TWI0 使能，TWI0_SDA 和 TWI0_SCL 端口锁存器(比如 P4.1 和 P4.0)必须设置为逻辑 1 并且 I/O 模式必须配置成开漏模式以用于接下来的串行通讯。

Bit 5: STA, 开始(START)标志
当 STA 位被置位进入主机模式时，TWI0 硬件将检查串行总线的状态，若总线空闲将产生一个开始信号。若总线忙，TWI0 将等待 STOP 信号出现并且在一个延迟后产生 START 信号。如果 STA 在 TWI0 已经是处于主机模式并且一个或多个字节已被发送或接收的情况下置位，TWI0 会发送一个 REPEATED START 信号。STA 可以在任何

时候置位，也可以在 TWI0 是一个被寻址的从机时置位。当 STA 位复位时，无 START 或 REPEATED START 信号产生。

Bit 4: STO, 停止(STOP)标志
当 TWI0 处于主机模式时，置位 STO 会向串行总线发送一个 STOP 信号。当在总线上检测到 STOP 信号时，TWI0 硬件清除 STO 标志。在从机模式时，置位 STO 标志可从总线错误状态恢复。在这种情况下不会向总线发送 STOP 信号，但是 TWI0 硬件表现就像已经接收到一个 STOP 信号，并且转换到未被寻址的从机接收模式。STOP 标志自动被硬件清零。如果 STA 与 STO 位同时置位，若 TWI0 处于主机模式将产生一个 STOP 信号（当处于从机模式时将产生一个内部的 STOP 信号，但不发送），接着发送一个 START 信号。

Bit 3: SI, 串行中断标志
当一个新的 TWI0 状态出现在 SISTA 寄存器时，SI 标志会被硬件置位。如果 TWI0 中断允许，中断服务程序将会运行。唯一不会使 SI 置位的状态是指出没有相关状态信息可以获得的 F8H。当 SI 置位时，TWI0_SCL 线上的低电平会延长，并且串行传输暂停。TWI0_SCL 线上的高电平不受 SI 标志影响。SI 必须由软件清零。SI 标志复位时不会产生中断请求，TWI0_SCL 线上的时钟也不会延长。

Bit 2: AA, 确实应答标志
如果 AA 标志设为 “1”，一个 ACK（TWI0_SDA 低电平）将在 TWI0_SCL 的应答时钟周期内回复，当：
1)接收到本机的从机地址
2) TWI0 处于主机/接收模式时，接收到一字节的数据
3) TWI0 处于已被寻址的从机/接收模式时，接收到一字节的数据

如果 AA 标志设为“0”，一个 NACK（ TWI0_SDA)高电平）将在 TWI0_SCL 的应答时钟周期内回复，当：
1) TWI0 处于主机/接收模式时，接收到一字节的数据
2) TWI0 处于已被寻址的从机/接收模式时，接收到一字节的数据

Bit 7, 1~0: CR2, CR1 和 CR0 时钟速率选择位
TWI0 处于主机模式时，这三个位决定串行时钟频率。当 TWI0 处于从机模式时，时钟频率并不重要，因为 TWI0 会自动同步任何主机的时钟频率，高达 100 KHz。表 21–1 给出不同的时钟速率设置：

表 21–1. TWI0 串行时钟速率

CR2	CR1	CR0	TWI0 时钟选择	TWI0 时钟速率@ SYSCLK=12MHz
0	0	0	SYSCLK/8	1.5 MHz
0	0	1	SYSCLK/16	750 KHz
0	1	0	SYSCLK/32	375 KHz
0	1	1	SYSCLK/64	187.5 KHz
1	0	0	SYSCLK/128	93.75 KHz
1	0	1	SYSCLK/256	46.875 KHz
1	1	0	**S1TOF/6**	可变的
1	1	1	**T0OF/6**	可变的

注意:
1. SYSCLK 是系统时钟
2. S1TOF 是 UART1 波特率定时器溢出
3. T0OF 定时器 0 溢出

SISTA:双线串行接口状态寄存器
SFR 页 = **0**
SFR 地址 = 0xD3 复位值= 1111-1000

7	6	5	4	3	2	1	0
SIS7	SIS6	SIS5	SIS4	SIS3	SIS2	SIS1	SIS0
R	**R**	**R**	**R**	**R**	**R**	**R**	**R**

SISTA 是一个 8 位的只读寄存器。低三位总是为 0，高五位保存状态编码，可以组成多个可能的状态编码。当 SISTA 为 F8H 时，没有串行中断请求。SISTA 的其它值用于定义相应的 TWI0 状态。当进入这些状态的一种时，会请求串行中断（SI=1）。在 SI 硬件置位时，一个有效的状态编码会存于 SISTA 中。

另外，状态 00H 表示总线错误。当一个 START 或 STOP 信号在不符合规定的位置发送时会产生总线错误，如一个地址/数据的内部或者刚好在应答位上。

AUXR1:辅助控制寄存器 1

SFR 页 = 0~F

SFR 地址 = 0xA2 复位值 = 0000-0000

7	6	5	4	3	2	1	0
P1KBIH	P3KBIL	P4SPI	P3S1	P3S1MI	**P6TWI0**	P3CEX	DPS
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 2: P6TWI0, TWI0 功能在 P6。当 P60OC[1:0] = “00”时此功能有效

P6TWI0	TWI0_SCL	TWI0_SDA
0	P4.0	P4.1
1	P6.0	P6.1

21.5. TWI1 寄存器

SI1ADR:双线串行接口 1 地址寄存器

SFR 页 = **1**

SFR 地址 = 0xD1 复位值= 0000-0000

7	6	5	4	3	2	1	0
A61	A51	A41	A31	A21	A11	A01	GC1
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

CPU 可以直接对此寄存器进行读写。SI1ADR 不受 TWI1 硬件的影响。当 TWI1 处于主机模式时此寄存器的值会被忽略。当处于从机模式时，寄存的高七位必须被用于本机的从机地址，并且当最低位（GC1）置位时，广播地址（00H）会被识别，否则忽略。在 START 状态后，最高位与从 TWI1 总线上收到的首位相对应。

SI1DAT: 双线串行接口 1 数据寄存器

SFR 页 = **1**

SFR 地址 = 0xD2 复位值= 0000-0000

7	6	5	4	3	2	1	0
D71	D61	D51	D41	D31	D21	D11	D01
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

此寄存器保存着一字节将要发送或者刚接收到的数据。在没有进行移位工作时，CPU 可以直接对此寄存器进行读写。这种情况发生在 TWI1 正处于一个被定义的状态并且串行中断标志位（SI1）置位。只要 SI 被置位，SI1DAT 中的数据总是保持稳定的。在数据被移出时，总线上的数据同时移入，SI1DAT 总保存着总线上出现的最后一个字节数据。因此在仲裁失败时，主机切换为从机的过程会在 SI1DAT 中产生一个正确的数据。

SI1DAT 与 ACK 标志位组成一个 9 位的移位寄存器，可以在移入或移出一个 8 位的数据后，跟随一个应答位。ACK 标志由 TWI1 硬件控制，CPU 访问不到。串行数据在 TWI1_SCL 的上升沿移入 SI1DAT 寄存器。当一字节的数据完全移入 SI1DAT 后，SI1DAT 中的数据将是可以用的，并且控制逻辑会在第 9 个时钟周期返回一个应答位。串行数据在 TWI1_SCL 的下降沿从 SI1DAT 寄存器移出。

CPU 向 SI1DAT 写入数据后，D71 位将首先出现在 TWI1_SDA 线上。9 个时钟周期后，SI1DAT 中的 8 位数据将被发送完成，并且通过应答位返回 ACK 标志。注意发送出去的 8 位数据会移回 SI1DAT。

SI1CON: 双线串行接口 1 控制寄存器

SFR 页 = **1**

SFR 地址 = 0xD4 复位值= 0000-0000

7	6	5	4	3	2	1	0
CR21	ENSI1	STA1	STO1	SI1	AA1	CR11	CR01
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

CPU 可以直接读写此寄存器。其中两个位会受 TWI1 硬件的影响：SI1 位会在串行中断请求时置位，STO1 位会在总线出现 STOP 状态时清零。STO1 位也会在 ENS1=0 时清零。

Bit 7: CR21, TWI1 时钟速率选择位 2 (与 CR11 和 CR01 一起使用)

Bit 6: ENSI1, TWI1
ENSI 为 "0"时，TWI1_SDA 与 SCL 输出为高阻态，TWI1_SDA 与 TWI1_SCL 输入信号被忽略，TWI1 处于被寻址（not-addressed）从机状态，SI1CON 的 STO1 位被强制置为”0”，但不影响其它位。TWI1_SDA 与 TWI1_SCL 可用作通用 I/O 引脚。ENSI 为 "1"时，TWI1 使能，TWI1_SDA 和 TWI1_SCL 端口锁存器(比如 P1.1 和 P1.0)必须设置为逻辑 1 并且 I/O 模式必须配置成开漏模式以用于接下来的串行通讯。

Bit 5: STA1, 开始(START)标志
当 STA1 位被置位进入主机模式时，TWI1 硬件将检查串行总线的状态，若总线空闲将产生一个开始信号。若总线忙，TWI1 将等待 STOP 信号出现并且在一个延迟后产生 START 信号。如果 STA1 在 TWI1 已经是处于主机模式并且一个或多个字节已被发送或接收的情况下置位，TWI1 会发送一个 REPEATED START 信号。STA1 可以在任何时候置位，也可以在 TWI1 是一个被寻址的从机时置位。当 STA1 位复位时，无 START 或 REPEATED START 信号产生。

Bit 4: STO1, 停止(STOP)标志
当 TWI1 处于主机模式时，置位 STO1 会向串行总线发送一个 STOP 信号。当在总线上检测到 STOP 信号时，TWI1 硬件清除 STO1 标志。在从机模式时，置位 STO1 标志可从总线错误状态恢复。在这种情况下不会向总线发送 STOP 信号，但是 TWI1 硬件表现就像已经接收到一个 STOP 信号，并且转换到未被寻址的从机接收模式。STOP 标志自动被硬件清零。如果 STA1 与 STO1 位同时置位，若 TWI1 处于主机模式将产生一个 STOP 信号（当处于从机模式时将产生一个内部的 STOP 信号，但不发送），接着发送一个 START 信号。

Bit 3: SI1,串行接口 1 中断标志
当一个新的 TWI1 状态出现在 SI1STA 寄存器时，SI1 标志会被硬件置位。如果 TWI1 中断允许，中断服务程序将会运行。唯一不会使 SI1 置位的状态是指出没有相关状态信息可以获得的 F8H。当 SI1 置位时，TWI1_SCL 线上的低电平会延长，并且串行传输暂停。TWI1_SCL 线上的高电平不受 SI1 标志影响。SI1 必须由软件清零。SI1 标志复位时不会产生中断请求，TWI1_SCL 线上的时钟也不会延长。

Bit 2: AA1, 确实应答标志
如果 AA1 标志设为 “1”，一个 ACK（TWI1_SDA 低电平）将在 TWI1_SCL 的应答时钟周期内回复，当：
1) 接收到本机的从机地址
2) TWI1 处于主机/接收模式时，接收到一字节的数据
3) TWI1 处于已被寻址的从机/接收模式时，接收到一字节的数据

如果 AA1 标志设为“0”，一个 NACK（ TWI1_SDA 高电平）将在 TWI1_SCL 的应答时钟周期内回复，当：
1) TWI1 处于主机/接收模式时，接收到一字节的数据
2) TWI1 处于已被寻址的从机/接收模式时，接收到一字节的数据

Bit 7, 1~0: CR21, CR11 和 CR01 时钟速率选择位
TWI1 处于主机模式时，这三个位决定串行时钟频率。当 TWI1 处于从机模式时，时钟频率并不重要，因为 TWI1 会自动同步任何主机的时钟频率，高达 100 KHz。表 21–1 给出不同的时钟速率设置：

表 21–2. TWI1 串行时钟速率

CR21	CR11	CR01	TWI1 时钟选择	TWI1 时钟速率@ SYSCLK=12MHz
0	0	0	SYSCLK/8	1.5 MHz
0	0	1	SYSCLK/16	750 KHz
0	1	0	SYSCLK/32	375 KHz
0	1	1	SYSCLK/64	187.5 KHz
1	0	0	SYSCLK/128	93.75 KHz
1	0	1	SYSCLK/256	46.875 KHz
1	1	0	**S1TOF/6**	可变的
1	1	1	**T0OF/6**	可变的

注意:

1. SYSCLK 是系统时钟
2. S1TOF 是 UART1 波特率定时器溢出
3. T0OF 定时器 0 溢出

SI1STA: 双线串行接口 1 状态寄存器

SFR 页 = **1**

SFR 地址 = 0xD3 复位值= 1111-1000

7	6	5	4	3	2	1	0
SIS71	SIS61	SIS51	SIS41	SIS31	SIS21	SIS11	SIS01
R	**R**	**R**	**R**	**R**	**R**	**R**	**R**

SI1STA 是一个 8 位的只读寄存器。低三位总是为 0，高五位保存状态编码，可以组成多个可能的状态编码。当 SI1STA 为 F8H 时，没有串行中断请求。SI1STA 的其它值用于定义相应的 TWI1 状态。当进入这些状态的一种时，会请求串行中断（SI1=1）。在 SI1 硬件置位时，一个有效的状态编码会存于 SI1STA 中。

另外，状态 00H 表示总线错误。当一个 START 或 STOP 信号在不符合规定的位置发送时会产生总线错误，如一个地址/数据的内部或者刚好在应答位上。

AUXR3:辅助寄存器 3

SFR 页 = 0~F

SFR 地址 = 0xA4 复位值 = 0000-0000

7	6	5	4	3	2	1	0
STAF	STOF	BPOC1	BPOC0	GF	P1S0MI	P3ECI	**P3TWI1**
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 0: P3TWI1, TWI1 功能在 P3

P3TWI1	TWI1_SCL	TWI1_SDA
0	P1.0	P1.1
1	P3.0	P3.1

21.6. 双线串行接口示例代码

(1). Required Function: Set TWI0 Master write/read

汇编语言代码范例:

建构中...

C 语言代码范例:

```
uCHAR I2C_Read(uCHAR Dev_Addr, uCHAR Reg_Addr)
 {
        uCHAR  usData = 0;

        SICON |= STA;
        SICON &= ~SI;
        while(( SICON & SI ) != SI );
        SICON &= ~STA;

        SIDAT = Dev_Addr;                       // send device address
        SICON &= ~SI;
        while(( SICON & SI ) != SI );

        SIDAT = Reg_Addr;                       // send register address
        SICON &= ~SI;
        while(( SICON & SI ) != SI );


        SICON |= STA;                           // restart
        SICON &= ~SI;
        while(( SICON & SI ) != SI );
        SICON &= ~STA;

        SIDAT = Dev_Addr | 0x01;                // send device address
        SICON &= ~SI;
        while(( SICON & SI ) != SI );

        SICON &= ~SI;
        while(( SICON & SI ) != SI );
        usData = SIDAT;

       SICON |= STO;
       SICON &= ~SI;
       while(( SICON & STO ) == STO );

        return usData;
 }


void I2C_Write(uCHAR Dev_Addr, uCHAR Reg_Addr, uCHAR ucData)
{
        SICON |= STA;
        SICON &= ~SI;
        while(( SICON & SI ) != SI );
        SICON &= ~STA;

        SIDAT = Dev_Addr;                       // send device address
        SICON &= ~SI;
        while(( SICON & SI ) != SI );

        SIDAT = Reg_Addr;                       // send register address
        SICON &= ~SI;
        while(( SICON & SI ) != SI );

        SIDAT = ucData;                         // send data
        SICON &= ~SI;
        while(( SICON & SI ) != SI );

       SICON |= STO;
       SICON &= ~SI;
```

```
        while(( SICON & STO ) == STO );
}

void main()
{
        SICON |= ENSI;                                  //enable TWI0 and clock source is 1.5M @MCU run at 12MHz.

        I2C_Write(0xA0, 0x30, 0x55);
        delay_ms(10);
        P0 = I2C_Read(0xA0, 0x30);

        while(1);
}
```

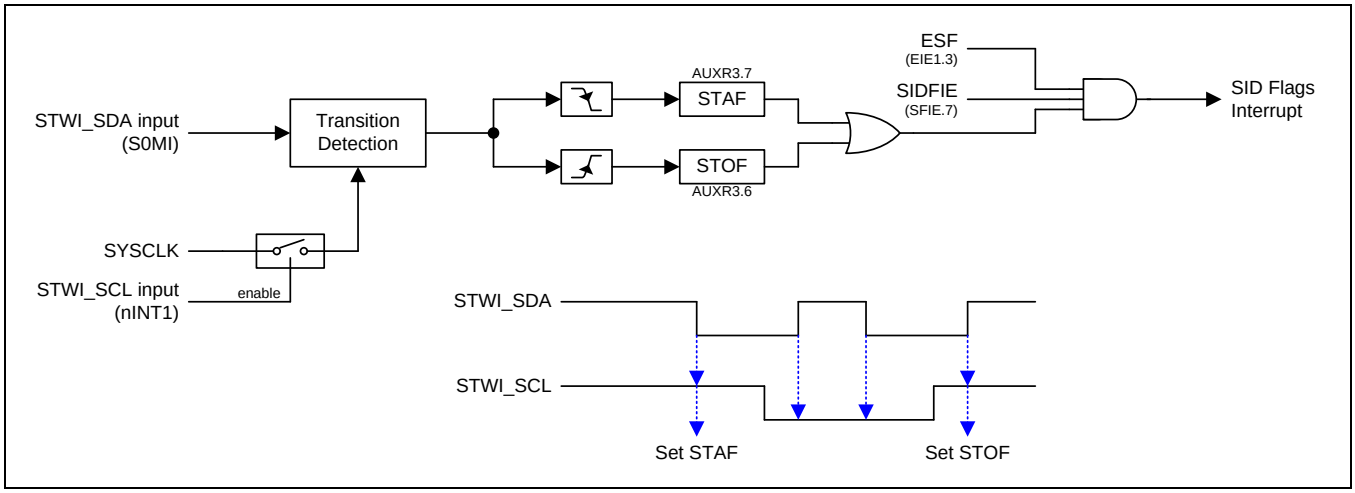
22. 串行接口侦测(SID/STWI)

串行接口侦测模块总是监控**软件**双线串行接口(STWI)的“Start”和“Stop”状态。STWI_SCL 是串行时钟信号和 STWI_SDA 是串行数据信号。如果任何匹配条件被侦测到，硬件设置 STAF 和 STOF 标志位。软件可以决定这两个标志或设置 SIDFIE (SFIE.7) 与系统标志共享中断向量。并且 STWI_SCL 位于 nINT1 将帮助 MCU 通过 nINT1 中断来判断串行数据。软件可以说使用这些资源来实施一个可变的 TWI 从机设备。

22.1. SID 结构

图 22–1 展示了 STAF 和 STOF 侦测的结构，中断结构和事件侦测波形。

图 22–1.串行接口侦测结构



22.2. SID 寄存器

AUXR3:辅助寄存器 3

SFR 页 = 0~F

SFR 地址 = 0xA4 复位值 = 0000-0000

7	6	5	4	3	2	1	0
STAF	**STOF**	BPOC1	BPOC0	GF	P1S0MI	P3ECI	P3TWI1
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: STAF, STWI 启动(Start)标志侦测
0:写“0”软件清零
1:硬件这位表明启动(START)条件出现在 STWI 总线上

Bit 6: STOF, STWI 停止(Stop) 标志侦测
0: 写“0”软件清零
1: 硬件这位表明启动(STOP)条件出现在 STWI 总线上

SFIE:系统标志中断使能寄存器

SFR 页 = 0~F

SFR 地址 = 0x8E 复位值 = XXXX-X000

7	6	5	4	3	2	1	0
SIDFIE	MCDRE	MCDFIE	RTCFIE	--	BOF1IE	BOF0IE	WDTFIE
R/W	**R/W**	**R/W**	**R/W**	**W**	**R/W**	**R/W**	**R/W**

Bit 7: SIDFIE,串行接口(STWI)侦测标志中断使能
0:禁止 SID 标志(STAF 或 STOF)中断
1:使能 SID 标志(STAF 或 STOF)中断

22.3. SIDF 串行接口侦测示例代码

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

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

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

汇编语言代码范例:

```
$INCLUDE (REG_MA82G5B32.INC)

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

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

;-------------------------------------------------------------------------
; TWSI 引脚定义
;-------------------------------------------------------------------------
SDA                 EQU     P3.2
SCL                 EQU     P3.3

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

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

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

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

        CSEG    AT  0013H                   ; EX1 中断服务(ISR)地址
        JMP     SCL_DETECT_ISR

        CSEG    AT  005BH               ; 检测 STAF 或 STOF 中断服务(ISR)地址
```

```
        JMP     SystemFlag_ISR

TWSI_CS SEGMENT CODE
        RSEG    TWSI_CS
        USING   0

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

MAIN_LOOP:
        ; to do ...

        MOV     ACC,Stage
        XRL     A,#I2C_Disable
        JZ      MAIN_LOOP

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

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

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

SUBROUTINE_I2C_SLA_with_R:
        CJNE    A,#I2C_SLA_with_R,SUBROUTINE_I2C_SL_W_ACK

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

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

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

        CLR     completeAByte           ; 清除事件标志位
        JMP     MAIN_LOOP

SUBROUTINE_I2C_SL_R_ACK:
        CJNE    A,#I2C_SL_R_ACK,SUBROUTINE_I2C_SL_R_NAK

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

        MOV     IICByte,@R1             ; 从数据缓冲区准备数据
        MOV     ACC,@R1
        RLC     A
        MOV     SDA,C                   ; SDA = MSB 高位在先

        CLR     completeAByte           ; 清除事件标志位
        JMP     MAIN_LOOP
```

```
SUBROUTINE_I2C_SL_R_NAK:
        CJNE    A,#I2C_SL_R_NAK,MAIN_LOOP

        SETB    SDA                     ; 无响应事件
        CLR     completeAByte
        JMP     MAIN_LOOP


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

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

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

        ; P33 和 P32 是漏极开路模式在 TWSI
        MOV     P3M0,#0CH
        MOV     P3M1,#0CH

        ; 边缘侦测
        SETB    IT1
        ORL     AUXR0,#INT1H

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

        RET

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

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

EXIT_FLAG_ISR:
        POP     PSW
        POP     ACC
        RETI

STAF_ROUTINE:                           ; TWSI 启动

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

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

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

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

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

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

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

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

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

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

```
        JNB     firstByte,REPEAT_RECEIVE_MODE
        CLR     firstByte

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

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

        MOV     Stage,#I2C_SLA_with_R
        JMP     EXIT_SCL_DETECT_ISR

SET_IN_SLAW_MODE:
        MOV     Stage,#I2C_SLA_with_W
        JMP     EXIT_SCL_DETECT_ISR

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

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

SLAVE_READ_WAIT_FOR_ACK:
        SETB    SDA
        JNZ     COMPLETE_READ_ONE_BYTE
        JMP     EXIT_SCL_DETECT_ISR

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

SET_I2C_SLAVE_READ_ACK:
        MOV     Stage,#I2C_SL_R_ACK
        JMP     EXIT_SCL_DETECT_ISR

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

C 语言代码范例:

```
#include <REG_MA82G5B32.H>
```

```
#include <intrins.h>

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

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

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

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

//-------------------------------------------------------------------------
// TWSI 引脚定义
//-------------------------------------------------------------------------
sbit SDA = P3^2;
sbit SCL = P3^3;

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

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

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

            // 在 TWSI 下 P33 和 P32 是漏极开路模式
            P3M0 = 0x0C;
            P3M1 = 0x0C;

            IT1 = 1;
            AUXR0 |= INT1H;

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

```
}

//---------------------------------------------------------------------------
// main()
//---------------------------------------------------------------------------
void main(void)
{
        unsigned char BufferIndex;
        unsigned char ReceiveString         [DATA_LENGTH];

        CKCON0 &= ~0x07;                                        // 系统时钟 SYSCLK/ 1
        INITIAL_TWSI ();                                        // 初始化中断和优先级

        while (1) {
                if (twsi.Stage != I2C_Disable) {
                        if (twsi.completeAByte == 1) {
                                switch (twsi.Stage)         {
                                case I2C_SLA_with_W:
                                        BufferIndex = 0;            //初始化缓冲区索引
                                        twsi.completeAByte = 0;
                                        twsi.Stage = I2C_SlaveStandby;
                                        break;

                                case I2C_SLA_with_R:
                                        // prepare MSB on SDA pin
                                        twsi.IICByte = ReceiveString [0];
                                        SDA = twsi.IICByte & 0x80;

                                        BufferIndex = 0;            //初始化缓冲区索引
                                        twsi.completeAByte = 0;
                                        twsi.Stage = I2C_SlaveStandby;
                                        break;

                                case I2C_SL_W_ACK:
                                        ReceiveString [BufferIndex] = twsi.IICByte;
                                        twsi.completeAByte = 0;

                                        BufferIndex       ++;               //限定缓冲区索引 0~31
                                        BufferIndex       &= 0x1F;
                                        twsi.Stage = I2C_SlaveStandby;

                                        break;

                                case I2C_SL_R_ACK:
                                        BufferIndex       ++;               //限定缓冲区索引 0~31
                                        BufferIndex       &= 0x1F;

                                        twsi.IICByte = ReceiveString [BufferIndex];
                                        SDA = twsi.IICByte & 0x80;
                                        twsi.completeAByte = 0;
                                        twsi.Stage = I2C_SlaveStandby;
                                        break;

                                case I2C_SL_R_NAK:
                                        SDA = 1;
                                        twsi.completeAByte = 0;
                                        twsi.Stage = I2C_SlaveStandby;
                                        break;

                                }
                        }

                }

                // to do ...
        }

}
```

```
//-------------------------------------------------------------------------
// 初始化 TWSI 的 SDA (STAF 和 STOF)边缘侦测
//-------------------------------------------------------------------------
void SystemFlag_ISR       (void) interrupt 11
{
        unsigned char tempReg;

        tempReg = AUXR1;
        AUXR1 &= ~(STAF+STOF);                          // 清除 STAF 和 STOF 标志位

        if (tempReg       & STOF){
                EX1 = 0;
                twsi.Stage = I2C_Disable;

        } else if (tempReg & STAF){
                AUXR0 |= INT1H;                         // SCL 上升缘侦测
                _nop_ ();
                IE1 = 0;
                SDA = 1;

                EX1 = 1;                                // 使能外部中断 EX1
                position = 0xFF;
                twsi.Slave_RW = 0;                      // 清除接收一个自己或地址

                twsi.Stage = I2C_SlaveStandby;
                firstByte = 1;
        }
}

//-------------------------------------------------------------------------
// access SDA by EX1 interrupt

//-------------------------------------------------------------------------
void TWSI_EX1_ISR (void) interrupt 2
{
        position ++;

        if ((twsi.Slave_RW) == 0) {
                if (position < 8) {                     // 0~7th 位
                        tempByte = tempByte       << 1; // 6th
                        tempByte |= SDA;

                        if (position == 7) {            // 侦测下降沿
                                AUXR0 &= ~INT1H;
                                _nop_ ();
                                IE1 = 0;
                                return;
                        } else {
                                IE1 = 0;
                                return;
                        }

                } else if (position == 8){              // 第 9 位(ACK)

                        if (firstByte) {
                                if ((tempByte >> 1) == twsi.ADDR) {
                                        SDA = 0;
                                } else {
                                        EX1 = 0;
                                        twsi.Stage = I2C_Disable;
                                }

                        } else {
                                SDA = 0;
                        }
```

```
                } else {
                        position = 0xFF;                                //复位位置
                        AUXR0 |= INT1H;                                 // 下降沿侦测复位 SCL 中断
                        _nop_ ();
                        IE1 = 0;
                        SDA = 1;

                        if (firstByte) {
                                firstByte = 0;

                                if ((tempByte &     0x01) == 0x01) {
                                        twsi.Slave_RW = 1;
                                        twsi.Stage = I2C_SLA_with_R;

                                        // for SCL falling edge        detection
                                        AUXR0 &= ~INT1H;
                                        _nop_ ();
                                        IE1 = 0;

                                } else {
                                        twsi.Slave_RW = 0;
                                        twsi.Stage = I2C_SLA_with_W;
                                }
                        } else {
                                twsi.Stage = I2C_SL_W_ACK;
                        }

                        twsi.IICByte = tempByte;
                        twsi.completeAByte = 1;                 // 当发送一个字节时置'1'

                }
        } else {
                if (position < 7) {
                        twsi.IICByte = twsi.IICByte << 1;       // 发送 6~0th 位到 SDA
                        SDA = twsi.IICByte & 0x80;
                } else if (position == 8) {
                        twsi.completeAByte = 1;                 //当发送一个字节时置'1'
                        position = 0xFF;                                // 复位位置

                        if (SDA) {
                                twsi.Stage = I2C_SL_R_NAK;
                        } else {
                                twsi.Stage = I2C_SL_R_ACK;
                        }
                        return;
                } else {                                                // 应答(ACK)/无应答(NAK)位
                        SDA = 1;
                }

        }
}
```

(2). 规定功能: STWI 从机在系统时钟 SYSCLK=24MHz 时的突发模式:

汇编语言代码范例:

```
$INCLUDE (REG_MA82G5B32.INC)

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

;-------------------------------------------------------------------
; TWSI 状态定义
;-------------------------------------------------------------------
I2C_SlaveStandby    EQU         0x00
I2C_SLA_with_W      EQU         0x01
I2C_SLA_with_R      EQU         0x02
I2C_Disable         EQU         0x03
```

```
I2C_SL_W_ACK        EQU         0x04
I2C_SL_R_ACK        EQU         0x05
I2C_SL_R_NAK        EQU         0x06

;-------------------------------------------------------------------
; TWSI 引脚定义
;-------------------------------------------------------------------
SDA                             EQU         P3.2
SCL                             EQU         P3.3

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

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

BITDATA SEGMENT BIT
             RSEG BITDATA
firstByte:          DBIT        1                       ; 接收 SLA+R/W 标志位
completeAByte:      DBIT        1                       ; 当发送/接收一个字节设置完成标志
Slave_RW:                   DBIT        1                       ; 清除 Slave_RW 设置传送
DisableTWSI:        DBIT        1                       ; 清除禁止 TWSI 而激活 TWSI 发送接收
StartTWSI:                  DBIT        1


;-------------------------------------------------------------------
; 程序代码区
;-------------------------------------------------------------------
        CSEG    AT 0000H                                ;复位地址= 0x0000
        JMP             ASSEMBLY_MAIN

        CSEG    AT 0013H                                ; 外部中断 EX1 中断服务(ISR)地址
        JMP             SCL_DETECT_ISR

        CSEG    AT 005BH                                ; 侦测 STAF 或 STOF 中断服务(ISR)地址
        JMP             SystemFlag_ISR

TWSI_CS SEGMENT CODE
        RSEG    TWSI_CS
        USING   0

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

MAIN_LOOP:
        ; to do    ...

        MOV             ACC,Stage
        XRL             A,#I2C_Disable
        JZ              MAIN_LOOP

        JNB             completeAByte,MAIN_LOOP         ; 是否有事件?

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

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

SUBROUTINE_I2C_SLA_with_R:
        CJNE            A,#I2C_SLA_with_R,SUBROUTINE_I2C_SL_W_ACK

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

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

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

        CLR             completeAByte                           ; 清除事件标志位
        JMP             MAIN_LOOP

SUBROUTINE_I2C_SL_R_ACK:
        CJNE            A,#I2C_SL_R_ACK,SUBROUTINE_I2C_SL_R_NAK

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

        MOV             IICByte,@R1                             ; 从数据缓冲区准备数据
        MOV             ACC,@R1
        RLC             A
        MOV             SDA,C                                   ; SDA = MSB 高位在先

        CLR             completeAByte                           ; 清除事件标志位
        JMP             MAIN_LOOP

SUBROUTINE_I2C_SL_R_NAK:
        CJNE            A,#I2C_SL_R_NAK,MAIN_LOOP

        SETB            SDA                                             ; 无应答(NAK)事件
        CLR             completeAByte
        JMP             MAIN_LOOP


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

        ; 外部中断 EX1 一般优先级
        ORL             IP0H,#00000100B
        ANL             IP0L,#11111011B

        ; 使能 ETWSI
        ORL             EIE1,#ESF
        ORL             SFIE,#SDIFIE
```

```
        SETB    EA

        ; 在 TWSI 中 P33 和 P32 是漏极开路模式
        MOV             P3M0,#0CH
        MOV             P3M1,#0CH

        ; 边缘侦测
        SETB            IT1
        ORL             AUXR0,#INT1H

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

        RET

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

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

EXIT_FLAG_ISR:
        POP             PSW
        POP             ACC
        RETI

STAF_ROUTINE:                                               ; 启动 TWSI
        ORL             AUXR0, #INT1H  ; 初始化外部中断 EX1 为上升沿侦测和使能 EX1 中断
        NOP
        CLR             IE1
        SETB            SDA
        SETB            EX1

        CLR             DisableTWSI                         ; 清除禁止 TWSI 标志为 0 (=激活 TWSI)
        ANL             AUXR1, #~STAF                       ; 清除 STAF 标志

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

STOF_ROUTINE:                                               ; 停止 TSWI
        CLR             EX1                                         ; 禁止外部中断 EX0 中断服务程序
        ANL             AUXR1,#~STOF                        ; 清除 STOF 标志
        SETB            DisableTWSI                         ; 禁止 TWSI (=TWSI 无效)
        MOV             Stage,#I2C_Disable
        JMP             EXIT_FLAG_ISR

;-------------------------------------------------------------------------
;  通过外部中断 EX1 访问 SDA
;-------------------------------------------------------------------------
SCL_DETECT_ISR:
        PUSH    ACC
        PUSH    PSW

        JNB             Slave_RW, SLAVE_WRITE
        JMP             SLAVE_READ
        ;-------------------------------------------------------------------------
```

```
SLAVE_WRITE:
        JNB         StartTWSI,$+5
        CLR         StartTWSI
        MOV         C, SDA                          ; MSB 高位在先–位 7
        MOV         A, tempByte                     ; 左移 SDA 到 tempByte.0
        RLC         A
        MOV         tempByte, A

        CLR         IE1                                     ; 等待 IE1
        JB          IE1, $+6
        JNB         DisableTWSI, $-3                ; 避免 STOF 事件

        JNB         StartTWSI,$+6
        LJMP        EXIT_SCL_DETECT_ISR
        MOV         C, SDA                          ; 位 6
        MOV         A, tempByte
        RLC         A
        MOV         tempByte, A
        CLR         IE1
        JB          IE1, $+6
        JNB         DisableTWSI, $-3

        JNB         StartTWSI,$+6
        LJMP        EXIT_SCL_DETECT_ISR
        MOV         C, SDA                          ; 位 5
        MOV         A, tempByte
        RLC         A
        MOV         tempByte, A
        CLR         IE1
        JB          IE1, $+6
        JNB         DisableTWSI, $-3

        JNB         StartTWSI,$+6
        LJMP        EXIT_SCL_DETECT_ISR
        MOV         C, SDA                          ; 位 4
        MOV         A, tempByte
        RLC         A
        MOV         tempByte, A
        CLR         IE1
        JB          IE1, $+6
        JNB         DisableTWSI, $-3

        JNB         StartTWSI,$+6
        LJMP        EXIT_SCL_DETECT_ISR
        MOV         C, SDA                          ; 位 3
        MOV         A, tempByte
        RLC         A
        MOV         tempByte, A
        CLR         IE1
        JB          IE1, $+6
        JNB         DisableTWSI, $-3

        JNB         StartTWSI,$+6
        LJMP        EXIT_SCL_DETECT_ISR
        MOV         C, SDA                          ; 位 2
        MOV         A, tempByte
        RLC         A
        MOV         tempByte, A
        CLR         IE1
        JB          IE1, $+6
        JNB         DisableTWSI, $-3

        JNB         StartTWSI,$+6
        LJMP        EXIT_SCL_DETECT_ISR
        MOV         C, SDA                          ; 位 1
        MOV         A, tempByte
        RLC         A
```

```
        MOV             tempByte, A
        CLR             IE1
        JB              IE1, $+6
        JNB             DisableTWSI, $-3

        JNB             StartTWSI,$+6
        LJMP            EXIT_SCL_DETECT_ISR
        MOV             C, SDA                          ; 位 0
        MOV             A, tempByte
        RLC             A
        MOV             tempByte, A
        CLR             IE1

        JB              DisableTWSI, EXIT_WITHOUT_COMPLETE_FLAG

        JNB             StartTWSI,$+6
        LJMP            EXIT_SCL_DETECT_ISR

        ANL             AUXR0, #~INT1H                  ; 设置外部中断 EX1 为下降沿侦测
        NOP
        CLR             IE1
        JB              IE1, $+6
        JNB             DisableTWSI, $-3

        JNB             firstByte,SLAVE_WRITE_RESPONSE_ACK

        MOV             ACC,tempByte
        CLR             C
        RRC             A
        CJNE            A,ADDR,NOT_SLAVE_ADDR
SLAVE_WRITE_RESPONSE_ACK:
        CLR             SDA
        JMP             COMPLETE_WRITE_ONE_BYTE

NOT_SLAVE_ADDR:
        CLR             EX1
        MOV             Stage,#I2C_Disable
        JMP             EXIT_SCL_DETECT_ISR

COMPLETE_WRITE_ONE_BYTE:                        ; 第 9 位下降沿
        CLR             IE1
        JB              IE1, $+6                        ; 等待第 9 位
        JNB             DisableTWSI, $-3

        SETB        SDA                                 ; 设置 SDA 为输入

        ORL             AUXR0, #INT1H                   ; 设置外部中断 EX1 为边缘侦测
        NOP
        CLR             IE1

        SETB    completeAByte                           ; 当接收一个字节时置'1'
        JNB             firstByte,REPEAT_RECEIVE_MODE
        CLR             firstByte

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

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

        MOV             Stage,#I2C_SLA_with_R
        JMP             EXIT_SCL_DETECT_ISR
```

```
SET_IN_SLAW_MODE:
        MOV             Stage,#I2C_SLA_with_W
        JMP             EXIT_SCL_DETECT_ISR

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

EXIT_WITHOUT_COMPLETE_FLAG:
        CLR             completeAByte
        JMP             EXIT_SCL_DETECT_ISR
        ;-------------------------------------------------------------------------
SLAVE_READ:
        ; 必须发送(tempByte.7)在主程序中
        ; 并且设置外部中断 EX1(SCL) 为下降沿侦测
        JNB             StartTWSI,$+5
        CLR             StartTWSI

        MOV             A,IICByte                       ; 发送 IICByte.6
        RL              A
        MOV             IICByte,A
        RLC             A
        MOV             SDA,C

        JNB             StartTWSI,$+6
        LJMP            EXIT_SCL_DETECT_ISR
        CLR             IE1                             ; 等待 IE1
        JB              IE1, $+6
        JNB             DisableTWSI, $-3                ; 避免 STOF 事件

        JNB             StartTWSI,$+6
        LJMP            EXIT_SCL_DETECT_ISR
        MOV             A,IICByte                       ; 发送 IICByte.5
        RL              A
        MOV             IICByte,A
        RLC             A
        MOV             SDA,C
        CLR             IE1
        JB              IE1, $+6
        JNB             DisableTWSI, $-3

        JNB             StartTWSI,$+6
        LJMP            EXIT_SCL_DETECT_ISR
        MOV             A,IICByte                       ; 发送 IICByte.4
        RL              A
        MOV             IICByte,A
        RLC             A
        MOV             SDA,C
        CLR             IE1
        JB              IE1, $+6
        JNB             DisableTWSI, $-3

        JNB             StartTWSI,$+6
        LJMP            EXIT_SCL_DETECT_ISR
        MOV             A,IICByte                       ; 发送 IICByte.3
        RL              A
        MOV             IICByte,A
        RLC             A
        MOV             SDA,C
        CLR             IE1
        JB              IE1, $+6
        JNB             DisableTWSI, $-3
```

```
        JNB     StartTWSI,$+6
        LJMP    EXIT_SCL_DETECT_ISR
        MOV     A,IICByte                                   ; 发送 IICByte.2
        RL      A
        MOV     IICByte,A
        RLC     A
        MOV     SDA,C
        CLR     IE1
        JB      IE1, $+6
        JNB     DisableTWSI, $-3

        JNB     StartTWSI,$+6
        LJMP    EXIT_SCL_DETECT_ISR
        MOV     A,IICByte                                   ; 发送 IICByte.1
        RL      A
        MOV     IICByte,A
        RLC     A
        MOV     SDA,C
        CLR     IE1
        JB      IE1, $+6
        JNB     DisableTWSI, $-3

        JNB     StartTWSI,$+6
        LJMP    EXIT_SCL_DETECT_ISR
        MOV     A,IICByte                                   ; 发送 IICByte.0
        RL      A
        MOV     IICByte,A
        RLC     A
        MOV     SDA,C
        CLR     IE1
        JB      IE1, $+6
        JNB     DisableTWSI, $-3

        JNB     StartTWSI,$+6
        LJMP    EXIT_SCL_DETECT_ISR

        SETB    SDA                                         ; 第 9 位-应答(ACK)/无应答(NAK)
        CLR     IE1
        JB      IE1, $+6
        JNB     DisableTWSI, $-3

        JNB     SDA,SET_I2C_SLAVE_READ_ACK
        MOV     Stage,#I2C_SL_R_NAK
        JMP     COMPLETE_READ_ONE_BYTE

SET_I2C_SLAVE_READ_ACK:
        MOV     Stage,#I2C_SL_R_ACK

COMPLETE_READ_ONE_BYTE:
        CLR     IE1
        SETB    completeAByte                               ; 当发送一个字节只'1'

        JMP     EXIT_SCL_DETECT_ISR

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

C 语言代码范例:

```
#include <REG_MA82G5B32.H>
#include <intrins.h>

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

//-------------------------------------------------------------------------
// I2C 定义
//-------------------------------------------------------------------------
```

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

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

_TWSI twsi;
unsigned char tempByte;
bit firstByte,DisableTWSI,StartTWSI;

//-----------------------------------------------------------------------------
// TWSI 引脚定义
//-----------------------------------------------------------------------------
sbit SDA = P3^2;
sbit SCL = P3^3;

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

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

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

        //在 TWSI 中 P33 和 P32 是漏极开路模式
        P3M0 = 0x0C;
        P3M1 = 0x0C;

        IT1 = 1;
        AUXR0 |= INT1H;

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

//-----------------------------------------------------------------------------
// main()
//-----------------------------------------------------------------------------
void main(void)
{
        unsigned char BufferIndex;
        unsigned char ReceiveString         [DATA_LENGTH];
```

```
        CKCON0 &= ~0x07;                                        // 系统时钟 SYSCLK / 1
        INITIAL_TWSI ();                                        // 初始化中断和优先级

        while (1) {
                if (twsi.Stage != I2C_Disable) {
                        if (twsi.completeAByte == 1) {
                                switch (twsi.Stage)          {
                                case I2C_SLA_with_W:
                                        BufferIndex = 0;            // 初始化缓冲区索引
                                        twsi.completeAByte = 0;
                                        twsi.Stage = I2C_SlaveStandby;
                                        break;

                                case I2C_SLA_with_R:
                                        // prepare MSB on SDA pin
                                        twsi.IICByte = ReceiveString [0];
                                        SDA = twsi.IICByte & 0x80;

                                        BufferIndex = 0;            //初始化缓冲区索引
                                        twsi.completeAByte = 0;
                                        twsi.Stage = I2C_SlaveStandby;
                                        break;

                                case I2C_SL_W_ACK:
                                        ReceiveString [BufferIndex] = twsi.IICByte;
                                        twsi.completeAByte = 0;

                                        BufferIndex       ++;               //限定缓冲区索引 0~31
                                        BufferIndex       &= 0x1F;
                                        twsi.Stage = I2C_SlaveStandby;

                                        break;

                                case I2C_SL_R_ACK:
                                        BufferIndex       ++;               //限定缓冲区索引 0~31
                                        BufferIndex       &= 0x1F;

                                        twsi.IICByte = ReceiveString [BufferIndex];
                                        SDA = twsi.IICByte & 0x80;
                                        twsi.completeAByte = 0;
                                        twsi.Stage = I2C_SlaveStandby;
                                        break;

                                case I2C_SL_R_NAK:
                                        SDA = 1;
                                        twsi.completeAByte = 0;
                                        twsi.Stage = I2C_SlaveStandby;
                                        break;

                                }
                        }

                }

                // to do ...
        }

}

//-----------------------------------------------------------------------
// 初始化 TWSI 的 SDA (STAF 和 STOF)边缘侦测
//-----------------------------------------------------------------------
void SystemFlag_ISR (void) interrupt 11
{
   unsigned char tempReg;

   tempReg = AUXR1;
```

```
    AUXR1 &= ~(STAF+STOF);                          // 清除 STAF 和 STOF 标志
    if (tempReg & STOF) {
        EX1 = 0;
        DisableTWSI = 1;
        twsi.Stage = I2C_Disable;
    } else if (tempReg & STAF){
        AUXR0 |= INT1H;
        _nop_ ();
        IE1 = 0;
        SDA = 1;
        EX1 = 1;
        DisableTWSI = 0;                                    // 避免误解

        twsi.Slave_RW = 0;                                  // 接收一个字节或地址而清除

        twsi.Stage = I2C_SlaveStandby;
        firstByte = 1;
        StartTWSI = 1;
    }
}

// ----------------------------------------------------------------------
//   通过外部中断 EX1 访问 SDA
// ----------------------------------------------------------------------
void TWSI_EX1_ISR(void) interrupt 2
{
    if (twsi.Slave_RW == 0) {
        if (StartTWSI) {
            StartTWSI = 0;
        }
        tempByte = tempByte << 1;               // 位 7
        tempByte |= SDA;
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);
                                                //位 6
        if (StartTWSI) return;
        tempByte = tempByte << 1;
        tempByte |= SDA;
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        if (StartTWSI) return;
        tempByte = tempByte << 1;               //位 5
        tempByte |= SDA;
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        if (StartTWSI) return;
        tempByte = tempByte << 1;               //位 4
        tempByte |= SDA;
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);
        if (StartTWSI) return;
        tempByte = tempByte << 1;               //位 3
        tempByte |= SDA;
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        if (StartTWSI) return;
        tempByte = tempByte << 1;               //位 2
        tempByte |= SDA;
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        if (StartTWSI) return;
        tempByte = tempByte << 1;               //位 1
        tempByte |= SDA;
```

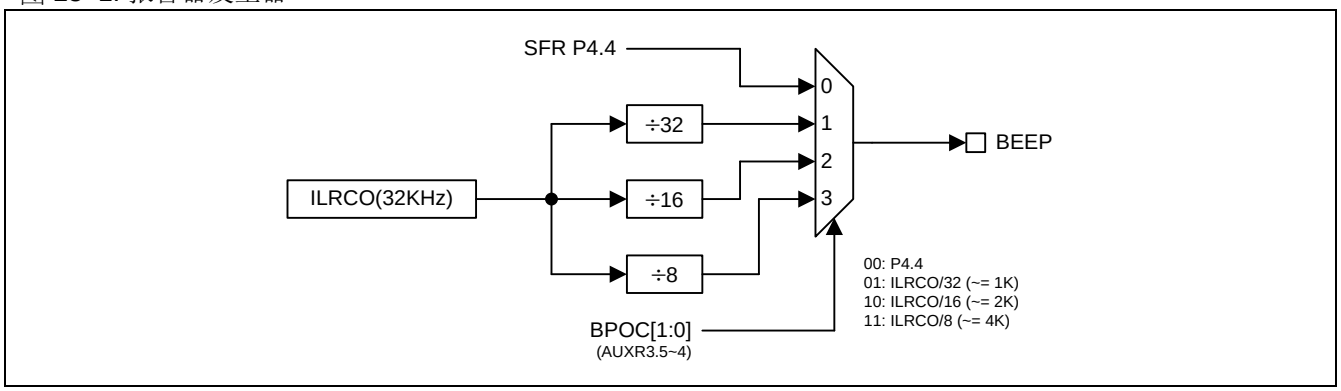
```
    IE1 = 0;
    while ((IE1 | DisableTWSI) == 0);

    if (StartTWSI) return;
    tempByte = tempByte << 1;                   //位 0
    tempByte |= SDA;

    AUXR0 &= ~INT1H;                            // SCL 的边缘侦测
    _nop_ ();
    IE1 = 0;
    while ((IE1 | DisableTWSI) == 0);                       // 第 0 个下降沿

    if (StartTWSI) return;
    if (DisableTWSI) return;

    if (firstByte) {
       if ((tempByte >> 1) == SLAVE_DEV_ADDR) {
          SDA = 0;
       } else {
          EX1 = 0;
          twsi.Stage = I2C_Disable;
       }
    } else {
       SDA = 0;
    }

    IE1 = 0;
    while ((IE1 | DisableTWSI) == 0);
    SDA = 1;

    AUXR0 |= INT1H;                             // SCL 的上升缘侦测
    _nop_ ();
    IE1 = 0;

    if (firstByte) {
       firstByte = 0;

       twsi.Slave_RW = (tempByte & 0x01);
       if (tempByte & 0x01) {
          twsi.Slave_RW = 1;
          twsi.Stage = I2C_SLA_with_R;
          // for SCL falling edge detection
          AUXR0 &= ~INT1H;
          _nop_ ();
          IE1 = 0;
       } else {
          twsi.Slave_RW = 0;
          twsi.Stage = I2C_SLA_with_W;
       }
    } else {
       twsi.Stage = I2C_SL_W_ACK;
    }
    twsi.IICByte = tempByte;
    if (DisableTWSI) return;
    twsi.completeAByte = 1;                                 // 当发送一个字节置'1'
    P35 = 1;
 } else {
    if (StartTWSI) {
       StartTWSI = 0;
    }
    twsi.IICByte = twsi.IICByte << 1;
    SDA = twsi.IICByte & 0x80;                  //位 6
    IE1 = 0;
    while ((IE1 | DisableTWSI) == 0);

    if (StartTWSI) return;
    twsi.IICByte = twsi.IICByte << 1;
```

```
        SDA = twsi.IICByte & 0x80;              //位 5
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        if (StartTWSI) return;
        twsi.IICByte = twsi.IICByte << 1;
        SDA = twsi.IICByte & 0x80;              //位 4
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        if (StartTWSI) return;
        twsi.IICByte = twsi.IICByte << 1;
        SDA = twsi.IICByte & 0x80;              //位 3
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        if (StartTWSI) return;
        twsi.IICByte = twsi.IICByte << 1;
        SDA = twsi.IICByte & 0x80;              //位 2
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        if (StartTWSI) return;
        twsi.IICByte = twsi.IICByte << 1;
        SDA = twsi.IICByte & 0x80;              //位 1
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        if (StartTWSI) return;
        twsi.IICByte = twsi.IICByte << 1;
        SDA = twsi.IICByte & 0x80;              //位 0
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);

        SDA = 1;                                // 应答(ACK)
        IE1 = 0;
        while ((IE1 | DisableTWSI) == 0);
        IE1 = 0;

        if (DisableTWSI) return;
        if (SDA) {
            twsi.Stage = I2C_SL_R_NAK;
        } else {
            twsi.Stage = I2C_SL_R_ACK;
        }
        twsi.completeAByte = 1;
    }
}
```

23. 报警器

报警器功能输出信号产生声音在 BEEP 引脚。信号来自 ILRCO 的分频，频率范围大约在 1, 2 或 4 kHz 。图 23–1 所示报警器发生器电路。但是 ILRCO 不是精确的时钟源。更详细的 ILRCO 频率偏差范围请参考章节"31.5 ILRCO 特性"。

图 23–1. 报警器发生器



23.1. 报警器寄存器

AUXR3: 辅助寄存器 3

SFR 页 = 0~F

SFR 地址 = 0xA4 复位值 = 0000-0000

7	6	5	4	3	2	1	0
STAF	STOF	**BPOC1**	**BPOC0**	GF	P1S0MI	P3ECI	P3TWI1
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 5~4: BPOC1~0,报警器输出控制位

BPOC[1:0]	P4.4 功能	I/O 模式
00	P4.4	By P4M0.4
01	ILRCO/32	By P4M0.4
10	ILRCO/16	By P4M0.4
11	ILRCO/8	By P4M0.4

报警器功能在 P4.4，推荐设置 P4M0.4 为"1"使 P4.4 工作在推挽输出模式。

23.2. 报警器示例代码

(1). 规定功能: 设置蜂鸣器输出 1KHz

汇编语言代码范例:

```
ORL     P4M0,#10H                   ; 设置 P4.4 为推挽输出模式

ANL     AUXR3,#~(BPOC1|BPOC0)       ; 设置 P4.4 为通用输入输出(GPIO)功能
ORL     AUXR3,#BPOC0                ; BEEP = ILRCO/32 ~= 1KHz 蜂鸣器频率约为 1KHz
```

C 语言代码范例:

```
P4M0 = P4M0 | 0x10;                 //设置 P4.4 为推挽输出模式

AUXR3 &= ~(BPOC1 | BPOC0);          //设置 P4.4 为通用输入输出(GPIO)功能
AUXR3 |= BPOC0;                     // BEEP = ILRCO/32 ~= 1KHz  蜂鸣器频率约为 1KHz
```

24. 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。键盘输入可以通过 P1KBIH 和 P3KBIL, AUXR1.7~6 从端口 1 和 3 来选择。默认下端口 2(Port 2)是键盘输入。

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 寄存器定义的相应端口，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。

24.1. 键盘寄存器

下面是键盘中断(KBI)操作相关的特殊功能寄存器：

KBPATN: 键盘模式寄存器

SFR 页 = 0~F

SFR 地址 = 0xD5 复位值= 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 。复位值是 0xFF

KBCON: 键盘控制寄存器

SFR 页 = 0~F

SFR 地址 = 0xD6 复位值= XXXX-XX01

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

Bit 7~2: 保留。当 KBCON 写入时，这些位软件必须写”0”

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
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。KBIF 缺省值是“1”
0:必须由软件写入‘0’来清零
1:键盘端口值匹配用户定义的 KBPATN、KBMASK 和 PATN_SEL 设置条件时置位

KBMASK: 键盘中断掩码寄存器

SFR 页 = 0~F

SFR 地址 = 0xD7 复位值= 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 P2.6 或 P1.3 作为键盘中断源 (KBI7) , 并由 AUXR1.7 选择 P2.6 或 P1.3
KBMASK.6: 置位时，使能 P2.4 或 P1.2 作为键盘中断源 (KBI6) , 并由 AUXR1.7 选择 P2.4 或 P1.2
KBMASK.5: 置位时，使能 P2.3 或 P1.1 作为键盘中断源 (KBI5) , 并由 AUXR1.7 选择 P2.3 或 P1.1
KBMASK.4: 置位时，使能 P2.2 或 P1.0 作为键盘中断源 (KBI4) , 并由 AUXR1.7 选择 P2.2 或 P1.0
KBMASK.3: 置位时，使能 P2.7 或 P3.5 作为键盘中断源 (KBI3) , 并由 AUXR1.6 选择 P2.7 或 P3.5
KBMASK.2: 置位时，使能 P2.5 或 P3.4 作为键盘中断源 (KBI2) , 并由 AUXR1.6 选择 P2.5 或 P3.4
KBMASK.1: 置位时，使能 P2.1 或 P3.1 作为键盘中断源 (KBI1) , 并由 AUXR1.6 选择 P2.1 或 P3.1
KBMASK.0: 置位时，使能 P2.0 或 P3.0 作为键盘中断源 (KBI0) , 并由 AUXR1.6 选择 P2.0 或 P3.0

AUXR1: 辅助控制寄存器 1

SFR 页 = 0~F

SFR 地址 = 0xA2 复位值 = 0000-0000

7	6	5	4	3	2	1	0
P1KBIH	P3KBIL	P4SPI	P3S1	P3S1MI	P6TWI	P3CEX	DPS
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: P1KBIH, KBI 高 4 位端口选择在 P1.3, P1.2, P1.1 和 P1.0

P1KBIH	KBI.7~4
0	P2.6, P2.4, P2.3, P2.2
1	P1.3, P1.2, P1.1, P1.0

Bit 6: P3KBIL, KBI 低 4 位端口选择在 P3.5, P3.4, P3.1 和 P3.0

P3KBIL	KBI.3~0
0	P2.7, P2.5, P2.1, P2.0
1	P3.5, P3.4, P3.1, P3.0

24.2. KB 键盘中断示例代码

(1). Required Function: Implement a KBI function on P2

Assembly Code Example:

```
   ORG    0003Bh
KBI_INT:
        MOV     KBCON, #00h                     ;Clear KP Interrupt Flag
        MOV     KBMASK, #00h                    ;Will Disable KP Interrupt

        RETI

main:
        MOV     PUCON0, #3Fh                    ;enable P0, P1 P2 internal pull high
        ORL     EIE1, #20h
        SETB    EA

        delay_ms        5

        MOV     KBPATN, #0FFh
        MOV     KBCON, #00h
        MOV     KBMASK, #0FFh                   ;Will Enable KP Interrupt

        CLR     P1.0
        ORL     PCON0, #02h                     ;into power down

        CLR     P1.1                            ;pull low any P0.x will wake up MCU.

Loop:
        JMP     Loop
```

C Code Example:

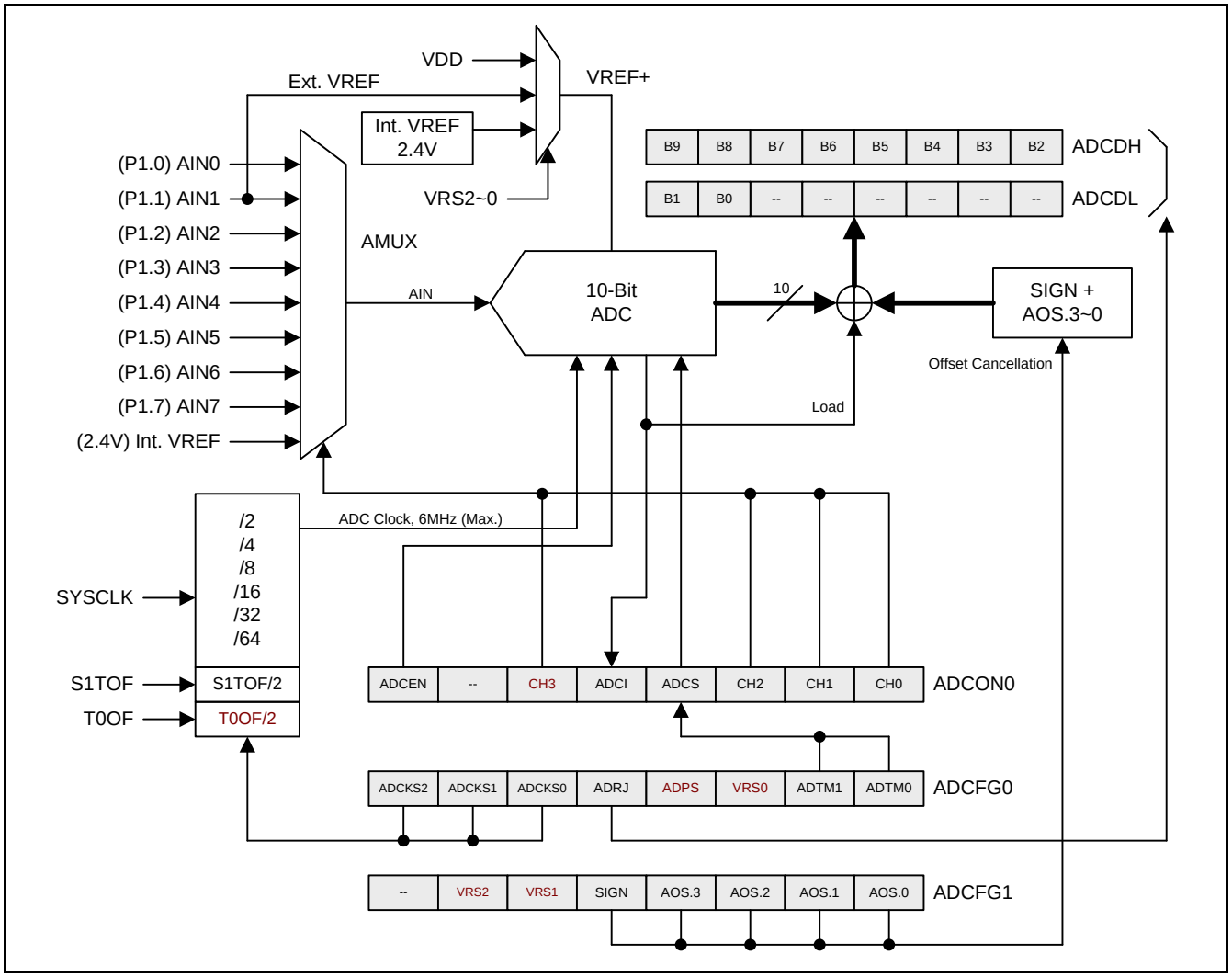
```
void KBI_ISR(void) interrupt 7
{
   KBCON=0;
   KBMASK=0;
}

void main(void)
{
   PUCON0 = 0x3F;                  // Enable P0 ~P2 on-chip pull-up resistor
   EIE1 |= EKB;                    // Enable KBI interrupt
   EA = 1;                         // Enable global interrupt

   Delay_5mS();

   KBPATN=0xFF;
   KBCON=0;
   KBMASK=0xFF;
   P10=0;

   PCON0 |= PD;                    // Set MCU into power-down mode
   P11=0;
   While(1);

}
```

25. 10 位模数转换器(ADC)

MA82G5BXX 的 ADC 子系统由一个模拟多路器（AMUX）和一个 **200** ksps、**10** 位逐次逼近型模数转换器组成。多路器(AMUX)可以通过特殊功能寄存器进行配置（如图 25–1）运行在单一节点模式，并且可以配置测量端口 1 的任何一个口或内部参考点。仅当 ADC 控制寄存器(ADCON0) 的 ADCEN 位被置逻辑 1 的时候 ADC 子系统被使能，ADCEN 设置为逻辑 0 的话 ADC 子系统低电关闭。

25.1. ADC 结构

图 25–1. ADC 结构框图



25.2. ADC 操作

ADC 最大转换速度可以达到 200 ksps，ADC 转换时钟由 ADCFG0 寄存器的 ADCKS2~0 位决定的系统时钟分频或 S1BRG 和定时器 0 的计时器溢出而来。ADC 转换时钟不能超过 6 MHz。

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：

$$\text{ADC Result} = \frac{V_{IN} \bullet x\,1024}{\text{VDD Voltage}}$$

25.2.1. 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 ADCON0 寄存器的 CHS.3~0 位选择进入 ADC 测量的通道（见图 25–1）。对被选择的引脚测量的是对地（GND）电压。

25.2.2. 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) 通过 ADCMS 来配置 ADC 的模式是单节点模式
3) 通过 ADCKS2, ADCKS1 和 ADCKS0 位配置 ADC 转换速度
4) 设置 CHS3, CHS2,CHS1 和 CHS0 选择输入通道
5) 设置 ADC 电压参考源
6) 设置 P1M0 和 P1AIO 寄存器将所选引脚设定成只模拟输入模式
7) 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 ADCKS2, ADCKS1 和 ADCKS0 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 **10** 位的转换结果按照 ADRJ 的设置存入 ADCH 和 ADCL。如果用户设置 ADCS 并且选择 ADC 的触发模式是 S1BRG/定时器 0 溢出或全速运行，这样 ADC 保持不断转换知道 ADCEN 清零或 ADC 配置成手动模式。

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EIE1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。

25.2.3. 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。ADC 的最大输入时钟是 6MHz 并且操作在固定的 30 个 ADC 转换时钟。用户可以通过 ADCFG0 寄存器的 ADCKS2~0 来配置转换速率。例如，若 SYSCLK =12MHz 并且 ADCKS = SYSCLK/2，则输入的模拟信号频率不应超过 200KHz，以保证转换精度。(转换速率= 12MHz/2/30 = 200KHz) 。

25.2.4. I/O 引脚用于 ADC 功能

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，可以按照引脚配置一节中的描述将引脚设为只输入模式。当 ADCI7~0 引脚作为模拟信号输入时并且数字信号输入不需要时，软件可以在 P1AIO 寄存器设置相应的引脚仅为模拟输入以便降低数字输入缓冲器的功耗。模拟输入功能的端口配置描述参考章节“14.2.1 端口 1 寄存器”

25.2.5. 空闲和掉电模式

在空闲和掉电模式下，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 (ADCEN =0)。

在掉电模式下，ADC 不工作。如果软件触发 ADC 操作在空闲模式下，ADC 将完成转换，并设置 ADC 中断标志，ADCI。当设置 ADC 中断启用 (EIE1.1 EADC)时，ADC 中断醒来 CPU 从空闲模式。

25.3. ADC 寄存器

ADCON0: ADC 控制寄存器 0

SFR 页 = 0~F

SFR 地址 = 0xC4 复位值 = 0000-0000

7	6	5	4	3	2	1	0
ADCEN	--	CHS3	ADCI	ADCS	CHS2	CHS1	CHS0
R/W	**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。在 ADCS 置位之前至少需要 5us 的 ADC 使能时间

Bit 6:保留。当 ADCON0 写时，此位软件必须写“0”

Bit 5: CHS3. 结合 CH2~0 选择 ADC 输入通道

Bit 4: ADCI, ADC 中断标志
0:此标志必须软件清零
1:一次 A/D 转换完成时此标志置位，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 并且 ADCI 置位。ADCS 或 ADCI 为”1”时将不会开始新的 A/D 转换。

Bit 2~0: CHS2 ~ CHS0, ADC 模拟多路器输入通道选择位

单节点模式:

CHS3~0	通道选择
0 0 0 0	AIN0 (P1.0)
0 0 0 1	AIN1 (P1.1)
0 0 1 0	AIN2 (P1.2)
0 0 1 1	AIN3 (P1.3)
0 1 0 0	AIN4 (P1.4)
0 1 0 1	AIN5 (P1.5)
0 1 1 0	AIN6 (P1.6)
0 1 1 1	AIN7 (P1.7)
1 0 0 0	保留
1 0 0 1	保留
1 0 1 0	保留
1 0 1 1	保留
1 1 0 0	保留
1 1 0 1	保留
1 1 1 0	GNDA
1 1 1 1	内部参考电压 (2.4V)

ADCFG0: ADC 配置寄存器 0

SFR 页 = 0~F

SFR 地址 = 0xC3 复位值 = 0000-0000

7	6	5	4	3	2	1	0
ADCKS2	ADCKS1	ADCKS0	ADRJ	ADPS	VRS0	ADTM1	ADTM0

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

Bit 7~5: ADC 转换时钟选择位

ADCKS[1:0]	ADC 时钟选择
0 0 0	SYSCLK/2
0 0 1	SYSCLK/4
0 1 0	SYSCLK/8
0 1 1	SYSCLK/16
1 0 0	SYSCLK/32
1 0 1	SYSCLK/64
1 1 0	S1TOF/2
1 1 1	T0OF/2

注意:

1. *SYSCLK 是系统时钟*
2. *S1TOF 是 UART1 波特率定时器溢出*
3. *T0OF 是定时器 0 溢出*

Bit 4: ADRJ, ADC 结果向右对齐选择
0:高 8 位转换结果存在 ADCH[7:0]，低 2 位转换结果存在 ADCL[7:6]
1:高 2 位转换结果存在 ADCH[1:0]，低 8 位转换结果存在 ADCL[7:0]

If ADRJ = 0

ADCDH: ADC 数据高字节寄存器

SFR 页 = 0~F

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 数据低字节寄存器

SFR 页 = 0~F

SFR 地址 = **0xC5** 复位值 = xxxx-xxxx

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

If ADRJ = 1

ADCDH：ADC 数据高字节寄存器

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

ADCDL：ADC 数据低字节寄存器

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

在单节点模式下，转换结果是 10 位的整形数。输入的测量值从”0”到 VREF x 1023/1024。向右对齐和向左对齐数据见下表示例。ADCDH 和 ADCDL 寄存器没有用到的位都是”0”。

输入电压 (单节点模式)	ADCDH:ADCDL (ADRJ = 0)	ADCDH:ADCDL (ADRJ = 1)
VREF+ x 1023/1024	0xFFC0	0x03FF
VREF+ x 512/1024	0x8000	0x0200
VREF+ x 256/1024	0x4000	0x0100
VREF+ x 128/1024	0x2000	0x0080
0	0x0000	0x0000

Bit 3: ADPS.保留为测试模式。当 ADCFG0 写入时，这位软件必须写”0”

Bit 2: VRS0. 结合 VRS2, VRS1 选择 ADC 电压参考源(VREF+).

Bit 1~0: ADC 触发模式选择.

ADTM[1:0]	ADC 转换启动选择
0 0	ADCS 置位
0 1	定时器 0 溢出
1 0	全速模式
1 1	S1 BRG 溢出

ADCFG1: ADC 配置寄存器 1

SFR 页 = 0~F

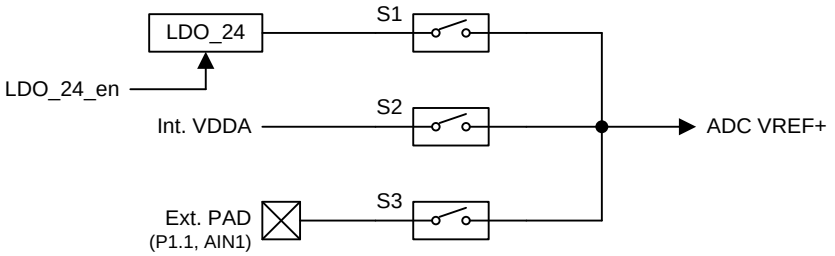
SFR 地址 = 0xBB 复位值 = xxx0-0000

7	6	5	4	3	2	1	0
--	VRS2	VRS1	SIGN	AOS.3	AOS.2	AOS.1	AOS.0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7:保留。当 ADCFG1 写入时，这位软件必须写”0”

Bit 6~5: VRS2~VRS1. 结合 VRS2, VRS1, VRS0 选择 ADC 电压参考源(VREF+).

VRS[2:0]	ADC 电压参考源 VREF+选择	S3, S2, S1	LDO_24_en
0 0 0	ADC VREF+ = VDDA	0 1 0	Dis
0 0 1	ADC VREF+ = AIN1 external VREF input	1 0 0	Dis
0 1 0	ADC VREF+ = VDDA, LDO24 for AIN15	0 1 0	En
0 1 1	ADC VREF+ = LDO_24 with Ext. pad	1 0 1	En
1 0 0	ADC VREF+ = VDDA	0 1 0	Dis
1 0 1	ADC VREF+ = VDDA	0 1 0	Dis
1 1 0	ADC VREF+ = VDDA with Ext. pad	1 1 0	Dis
1 1 1	ADC VREF+ = LDO_24	0 0 1	En



Bit 4~0: SIGN 和 AOS.3~0。寄存器值校正 ADC 结果{ADCH, ADCL}的偏移量消除。

{Sign, AOS.[3:0]}	{ADCDH, ADCDL}的值
0_1111	ADC 转换结果 + 15
0_1110	ADC 转换结果 + 14
……	……
0_0010	ADC 转换结果 + 2
0_0001	ADC 转换结果 + 1
0_0000	ADC 转换结果 + 0
1_1111	ADC 转换结果 – 1
1_1110	ADC 转换结果 – 2
……	……
1_0001	ADC 转换结果 – 15
1_0000	ADC 转换结果 – 16

P1AIO: 端口 1 仅是模拟输入

SFR 页 = 0~F

SFR 地址 = 0x92 复位值 = 0000-0000

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

0:端口引脚数字和模拟输入共享
1:端口引脚仅是模拟输入。此位置位相应端口引脚寄存器的位读出来总是"0"

25.4. ADC 转换示例代码

26. ISP 和 IAP

MA82G5BXX 的 Flash 存储空间划分为 AP 存储空间，IAP 存储空间和 ISP 存储空间。AP 存储空间用来存储用户应用程序；IAP 存储空间用来存储非易失性数据；ISP 存储空间用来存储在线编程的引导码。系统在 ISP 空间运行时，MCU 可以修改 AP 和 IAP 来更新软件。如果 MCU 运行在 AP 空间，那么 MCU 就仅能修改 IAP 存储的数据。

26.1. MA82G5B32 Flash 存储空间配置

MA82G5B32 总共有 32K 字节的 Flash，图 26–1 显示了 **MA82G5B32** 的 Flash 配置。ISP 存储空间可以被禁止或由硬件选项配置最大 4K 字节。IAP 存储空间大小由 IAP 低边界和高边界决定。IAP 低边界由 IAPLB 寄存器的值决定。IAP 高边界与 ISP 的起始地址相关，ISP 存储空间由硬件选项决定。IAPLB 寄存器值由硬件选项配置或 AP 软件编程设定。所有 AP, IAP 和 ISP 存储空间共享总 32K 字节的存储空间。

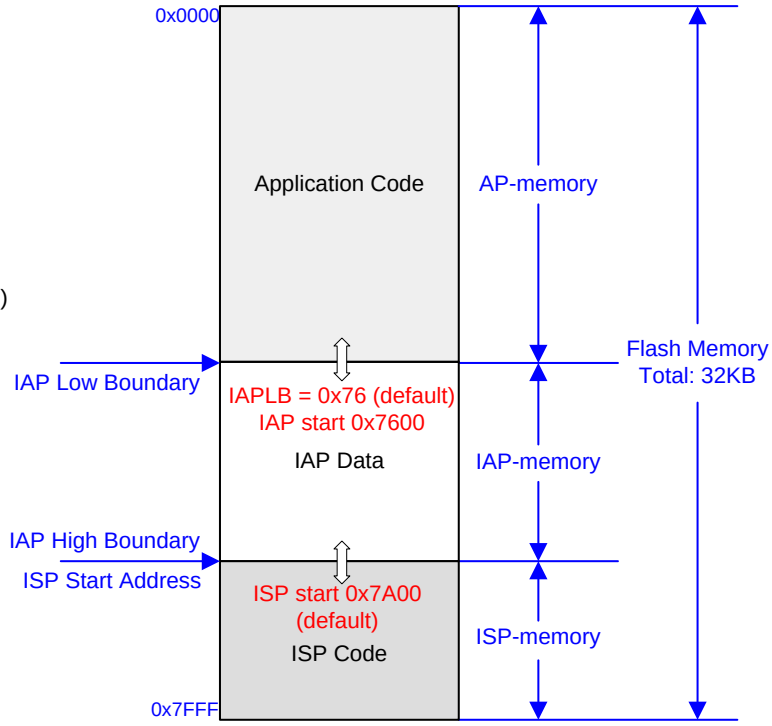
图 26–1. **MA82G5B32** Flash 存储空间配置

Note:
(1) ISP Start Address:
0x7000 if ISP Size = 4KB
0x7200 if ISP Size = 3.5KB
0x7400 if ISP Size = 3KB
0x7600 if ISP Size = 2.5KB
0x7800 if ISP Size = 2KB
0x7A00 if ISP Size = 1.5KB
0x7C00 if ISP Size = 1KB

(2) IAP Size :
IAPLB = IAP Low Boundary (ROM High-Byte address)
IAP Start Address = { IAPLB, 00H }
IAP Size = ISP Start Address – IAP Start Address
Set LAPLB = Change IAP Size

(3) If ISP is enabled:
IAP High Boundary = ISP Start Address – 1
IAP Low Boundary = ISP Start Address – IAP Size

(4) If ISP is disabled:
IAP High Boundary = 0x7FFF
IAP Low Boundary = 0x7FFF – IAP Size + 1



注意:

*笙泉公司 **MA82G5B32** 的默认设置是：1.5K ISP, 1K IAP 和加密。1.5K ISP 区域是镶入有笙泉专利的 COMBO ISP 代码通过一条线就能在线下载的 1-线 ISP 协议及串口(COM)ISP 协议。1K IAP 大小可以通过应用程序软件来重新配置。*

26.2. MA82G5BXX Flash 在 ISP/IAP 的访问

MA82G5BXX 给 ISP 和 IAP 应用提供三种 flash 访问模式：页擦除模式，编程模式及读取模式。MCU 软件使用这三种模式去更新 Flash 的数据和获取 Flash 的数据。本章展示了不同 Flash 模式的流程图和范例代码。

在执行 ISP/IAP 操作之前,用户需要在 CKCON1 寄存器的 XCKS5~XCKS0 填入正确值。(参考章节"9.2 时钟寄存器")

页擦除 (512 字节每页)

步骤1: 在ISPCR 寄存器上设置MS[2:0]=[0,1,1]选择页擦除模式
步骤2: 填入页地址到IFADRH和IFADRL寄存器
步骤3: 顺序地在SCMD 寄存器写入0x46h 然后0xB9h 触发一个ISP处理

字节编程

步骤1:在ISPCR 寄存器上设置MS[2:0]=[0,1,0]选择字节编程模式
步骤2: 填入字节地址到IFADRH和IFADRL寄存器
步骤3: 填入被编程数据到IFD寄存器
步骤4: 顺序地在SCMD 寄存器写入0x46h 然后0xB9h 触发一个ISP处理

字节读取

步骤1: 在ISPCR 寄存器上设置MS[2:0]=[0,0,1]选择字节读取模式
步骤2: 填入字节地址到IFADRH和IFADRL寄存器
步骤3: 顺序地在SCMD 寄存器写入0x46h 然后0xB9h 触发一个ISP处理
步骤4: 现在,Flash 数据在IFD 寄存器

MA82G5BXX的页擦除，字节编程和读取的详细描述见下面章节：

26.2.1. ISP/IAP Flash 页擦除模式

MA82G5BXX 的 flash 数据任何一位只能编程为“0”。如果用户需要写“1”到 flash 数据，flash 需要擦除。但是在 **MA82G5BXX** 的 ISP/IAP 操作中的 flash 擦除只支持”页擦除”模式，一页擦除将写“1”到一页的所有数据位。**MA82G5BXX** 的一页有 512 个字节并且页的起始地址排列到 A8~A0 = 0x000。目标 flash 地址由 IFADRH 和 IFADRL 决定。这样，在 flash 页擦除模式， IFADRH.0(A8)和 IFADRL.7~0(A7~A0)必须写 “0”选择正确的页地址。图 26–2 展示了在 ISP/IAP 操作的 flash 页擦除流程。

图 26–2. ISP/IAP 页擦除流程

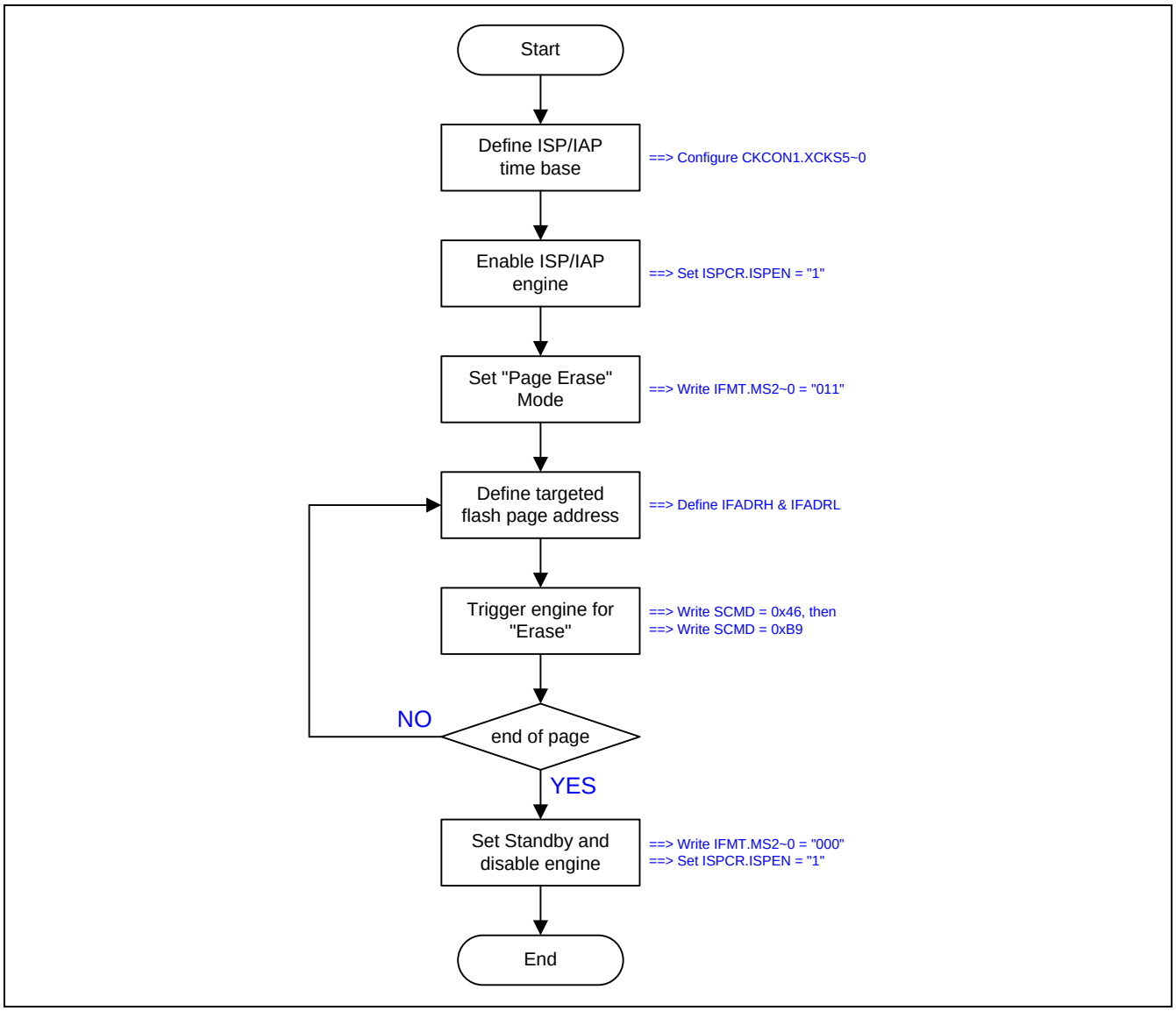


图 26–3 所示 ISP/IAP 页擦除操作的示例代码

图 26–3. ISP/IAP 页擦除操作的示例代码

```
MOV    ISPCR,#00001011b ; XCKS5~0 = 11(十进制) 当 OSCin = 12MHz 时

MOV    ISPCR,#10000000b ; ISPCR.7 = 1, 使能 ISP

MOV    IFMT,#03h        ; 选择页擦除模式

MOV    IFADRH,??        ; 页地址填到[IFADRH,IFADRL]
MOV    IFADRL,??        ;

MOV    SCMD,#46h        ; 触发 ISP/IAP 处理
MOV    SCMD,#0B9h       ;

; MCU 等待处理完成

MOV    IFMT,#00h        ; 选择备用模式
MOV    ISPCR,#00000000b ; ISPCR.7 = 0, 禁止 ISP
```

26.2.2. ISP/IAP Flash 编程模式

MA82G5BXX 编程模式提供 Flash 存储空间的字节写操作来更新数据。IFADRH 和 IFADRL 指向 Flash 的物理字节地址。IFD 存储编程到 Flash 的内容。图 26–4 展示了 ISP/IAP 操作的 Flash 字节编程流程。

图 26–4. ISP/IAP 字节编程流程

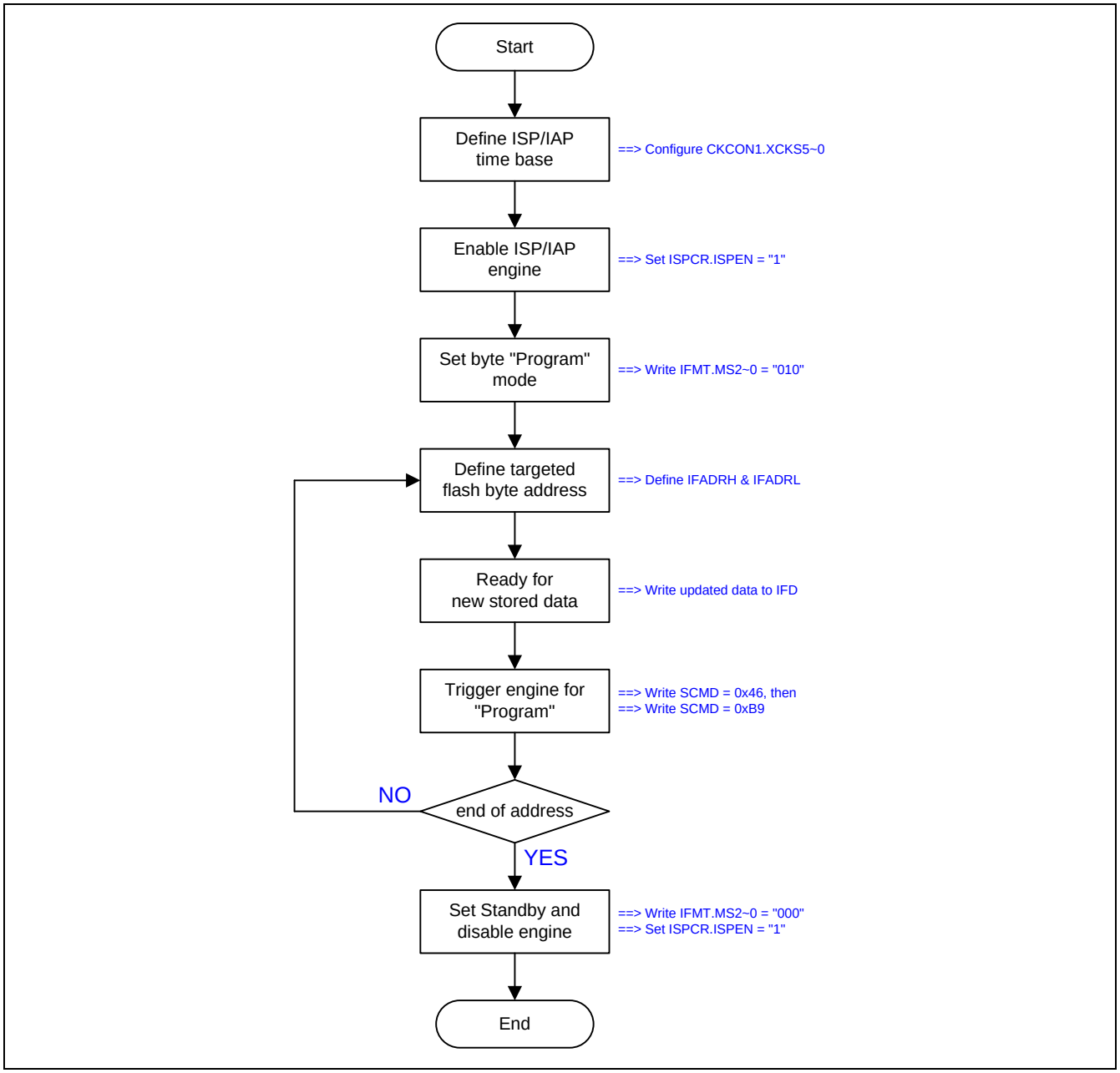


图 26–5 所示 ISP/IAP 字节编程操作的示例代码

图 26–5. ISP/IAP 字节编程的示例代码

```
MOV    ISPCR,#00001011b ; XCKS5~0 = 11(十进制) 当 OSCin = 12MHz 时

MOV    ISPCR,#10000000b ; ISPCR.7=1, 使能 ISP

MOV    IFMT,#02h        ; 选择编程模式

MOV    IFADRH,??        ; 字节地址填到[IFADRH,IFADRL]
MOV    IFADRL,??        ;

MOV    IFD,??           ; 编程数据填到 IFD

MOV    SCMD,#46h        ; 触发 ISP/IAP 处理
MOV    SCMD,#0B9h       ;

; MCU 等待处理完成

MOV    IFMT,#00h        ; 选择备用模式
MOV    ISPCR,#00000000b ; ISPCR.7 = 0, 禁止 ISP
```

26.2.3. ISP/IAP Flash 读取模式

MA82G5BXX 读取模式提供从 Flash 存储空间获取已存储数据的字节读取操作。IFADRH 和 IFADRL 指向 Flash 的物理字节地址。IFD 存储从 Flash 读取到的内容。建议在数据编程或页擦除之后通过读取模式核对 Flash 数据。图 26–6 所示 ISP/IAP 操作下的 Flash 字节读取流程。

图 26–6. ISP/IAP 字节读取流程

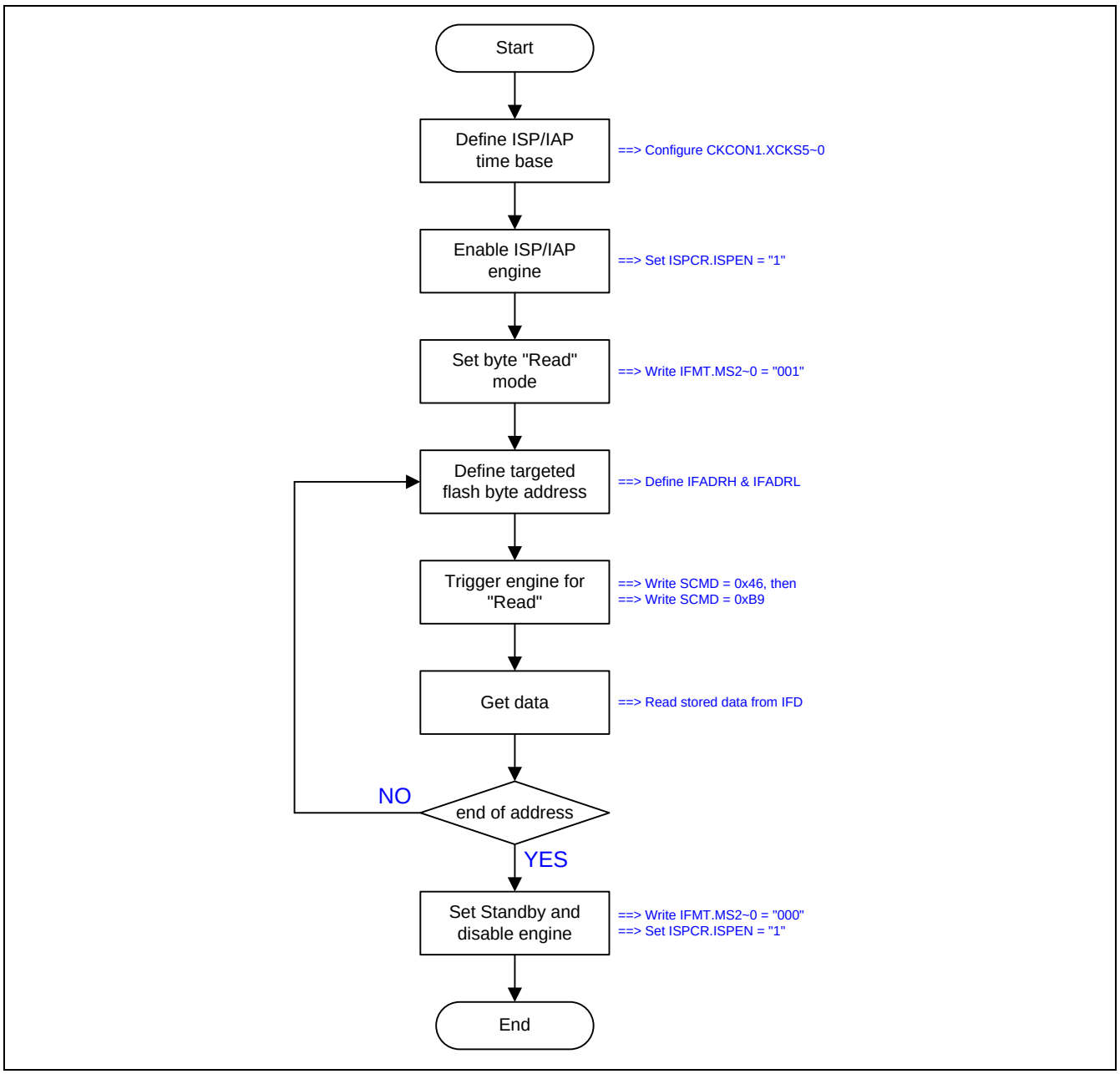


图 26–7 所示 ISP/IAP 字节读取操作的范例代码

图 26–7. ISP/IAP 字节读取的范例代码

```
MOV    ISPCR,#00001011b ; XCKS5~0 = 11(十进制) 当 OSCin = 12MHz 时

MOV    ISPCR,#10000000b ; ISPCR.7=1, 使能 ISP

MOV    IFMT,#01h        ; 选择读取模式

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

MOV    SCMD,#46h        ; 触发 ISP/IAP 处理
MOV    SCMD,#0B9h       ;

; MCU 等待处理完成

MOV    A,IFD            ; 读取到的数据存在 IFD

MOV    IFMT,#00h        ; 选择备用模式
MOV    ISPCR,#00000000b ; ISPCR.7 = 0, 禁止 ISP
```

26.3. ISP 操作

ISP 意指在系统可编程，不需要在实际的终端产品上移除 MCU 芯片就可以更新用户的应用程序(AP 存储空间)和非易失性应用数据(IAP 存储空间)。这个可使用性就有一个宽的现场应用范围。ISP 模式使用引导程序来编程 AP 存储空间和 IAP 存储空间。

注意:

(1) 在用 ISP 功能之前，使用者必须先配置 ISP-存储器空间并用通用烧写器或笙泉的烧写器插入 ISP 文件到 ISP-存储器中。

(2) ISP-存储器中的 ISP 文件 code 只能下载 AP-存储器和非易失的 IAP-存储器

在 ISP 操作完成之后,软件写 “001”到 ISPCR.7 ~ ISPCR.5 这样会触发一个软件**复位(**RESET**)**并且使 CPU 再启动到应用程序存储空间(AP)的 0x0000 地址。

如我们所知,ISP 代码的作用就是编程 AP 存储空间和 IAP 存储空间。因此，***MCU 为了执行 ISP 代码必须从 ISP 存储空间启动***。根据 MCU 如何从 ISP 存储空间启动，有两种方法执行在系统可编程。

26.3.1. 硬件访问 ISP

在上电复位时为了使 MCU 直接从 ISP 存储空间启动，MCU 的硬件选项 *HWBS* 和 *ISP 存储空间*必须使能。硬件选项的 ISP 进入方法叫做硬件访问。一旦 *HWBS* 和 *ISP 存储空间*使能,当上电复位时 MCU 总是从 ISP 存储空间启动去执行 ISP 代码(引导程序)。ISP 代码做的第一件事是核对是否有 ISP 请求。如果没有 ISP 请求,ISP 代码触发软件复位(设置 ISPCR.7~5 为“101”)使 MCU 在启动到 AP 存储空间去运行用户应用程序。

如果额外的硬件选项 HWBS2 与 HWBS 及 ISP 存储空间一起使能, MCU 在上电复位或外部复位结束之后总从 ISP 存储空间启动 。通过外部复位信号提供另外一个硬件访问进入 ISP 模式。第一上电复位之后，**MA82G5BXX** 通过外部复位触发而执行 ISP 操作并且不用等待下一次的上电复位，这适合不断电系统去应用硬件访问 ISP 功能。

26.3.2. 软件访问 ISP

当 MCU 运行在 AP 存储空间时，软件访问 ISP 通过触发软件复位使 MCU 从 ISP 存储空间启动。这种情况，HWBS 或 HWBS2 不用使能。仅有的方法是当 MCU 运行在 AP 存储空间时同时设置 ISPCR.7~5 为“111”触发软件复位 MCU 从 ISP 存储空间启动。注意： ISP 存储空间必须通过硬件选项配置一个有效空间来保留 ISP 模式给软件访问 ISP 应用。

26.3.3. ISP 注意事项

ISP 代码开发

尽管 ISP 存储空间的 ISP 代码是可编程的，ISP 存储空间在 MCU 的 Flash 中有一个 *ISP 起始地址*(**MA82G5B32** 见图 26–1), 但是并不意味着你需要在你的源代码中加入这个偏移量 (*ISP 起始地址*)。代码偏移量硬件自动处理。用户只需像在 AP 存储空间开发应用程序一样开发。

ISP 期间中断

在触发 ISP/IAP flash 处理之后，内部 ISP 处理时 MCU 将停止一会儿直到处理完成。此时，如果中断已使能则中断事件将排队等待服务。一旦 ISP/IAP flash 处理完成，MCU 继续运行并且如果中断标志仍然有效则排队中的中断将立即服务。不过用户需要意识到下列事项：

(1) 当 MCU 停止在 ISP 处理时，中断不能实时服务
(2) 低/高电平触发外部中断 nINTx,必须保持到 ISP 处理完成，否则将被忽略

ISP 和空闲模式

MA82G5BXX 不使用空闲模式执行 ISP 功能。反而 ISP/IAP 引擎操作 Flash 存储空间将冻结 CPU 的运行。一旦 ISP/IAP 运行结束，CPU 将继续并且推进紧跟着 ISP/AP 激活的指令。

ISP 的访问目标

如前所述, ISP 用来编程 AP 存储空间和 IAP 存储空间。一旦访问目标地址超出 IAP 存储空间的最后一个字节之外, 硬件将自动忽略 ISP 处理的触发。这样 ISP 触发是无效的并且硬件不做任何事情。

ISP 的 Flash 持久期

内置 Flash 的持久期是 10,000 写周期，换句话说写周期不能超过 10,000 次。这样用户必须注意应用中需要频繁更新 AP 存储空间和 IAP 存储空间这一点。

26.4. IAP 操作

MA82G5BXX 内建一个在应用可编程(IAP)功能，当应用程序运行时在 Flash 存储空间里允许一些区域被应用成非易失性数据存储区。这个有用特点能使用在断电后还需要保存数据的应用中。这样不需要使用外部的串行 EEPROM (比如 93C46, 24C01, .., 等等)来保存非易失性的数据。

事实上，IAP的操作除了Flash存储空间被划分在不同的区域之外与ISP一样。ISP操作的可编程Flash范围在AP存储空间和IAP存储空间，而IAP操作的范围只在IAP存储空间。

注意:

*(1) **MA82G5BXX** 的IAP 特点，软件通过写SFR P页的IAPLB寄存器声明IAP 存储空间。IAP 存储空间也可以通过通用的烧入器/编程器或笙泉专利的烧入器/编程器来配置IAPLB 的初始值。*

(2)执行IAP 的程序代码是在AP存储空间并且仅能编程IAP存储空间而不能编程ISP存储空间

26.4.1. IAP-存储空间边界/范围

如果ISP 存储空间被声明，IAP存储空间范围由IAP和ISP起始地址决定如下列表：

IAP高边界= ISP起始地址 –1
*IAP低边界 = ISP起始地址 – IAP*空间

如果ISP 存储空间没有被声明，IAP存储空间范围由下列公式决定：

IAP高边界 = 0x7FFF.
*IAP高边界 = 0x7FFF – IAP*空间 *+ 1.*

例如，如果ISP 存储空间是**1K**字节，这样ISP 的起始地址是0x7C00，并且IAP 存储空间是**1K**字节，此时IAP 存储空间的范围就在0x7800 ~ 0x7BFF 。**MA82G5B32**的IAP 低边界由IAPLB 寄存器决定，IAPLB 寄存器可以在用户AP程序里用软件修改来调整IAP大小。

26.4.2. IAP-存储空间更新数据

ISP/IAP 相关的特殊功能寄存器见章节“26.5 ISP/IAP寄存器 “

由于 IAP 存储空间是 Flash 存储空间的一部分，Flash 擦除仅提供页擦除，没有字节擦除。为了在 IAP 存储空间更新”一个字节”, 用户不能直接编程一个新数据到那个字节。正确的步骤如下：

步骤 1: 保存整页 flash 数据(512 字节)到包含被更新数据的 XRAM 缓冲区
步骤 2:擦除此页 ***(使用 ISP/IAP Flash 页擦除模式)***
步骤 3:在 XRAM 缓冲区修改新数据字节
步骤 4: 编程 XRAM 缓冲区的被更新数据到此页***(使用 ISP/IAP Flash 编程模式)***

为了读取 IAP 存储空间数据，用户可以使用 ***ISP/IAP Flash 读取模式***获取目标数据。

26.4.3. IAP 注意事项

IAP 期间中断

在触发 ISP/IAP flash 处理之后，内部 IAP 处理时 MCU 将停止一会儿直到处理完成。此时，如果中断已使能则中断事件将排队等待服务。一旦 ISP/IAP flash 处理完成，MCU 继续运行并且如果中断标志仍然有效则排队中的中断将立即服务。不过用户需要意识到下列事项：

(1) 当 MCU 停止在 IAP 处理时，中断不能实时服务
(2) 低/高电平触发外部中断 nINTx,必须保持到 IAP 处理完成，否则将被忽略

IAP 和空闲模式

MA82G5BXX 不使用空闲模式执行 IAP 功能。反而 ISP/IAP 引擎操作 Flash 存储空间将冻结 CPU 的运行。一旦 ISP/IAP 运行结束，CPU 将继续并且推进紧跟着 ISP/AP 激活的指令。

IAP 的访问目标

如前所述, IAP 用来编程 IAP 存储空间。一旦访问目标地址不在 IAP 存储空间之内, 硬件将自动忽略 ISP 处理的触发。这样 IAP 触发是无效的并且硬件不做任何事情。

读取 IAP 数据的另一种方法

IAP 存储空间读取 Flash 数据，除了使用 Flash 的读取模式之外，另一个方法是使用“MOVC A,@A+DPTR”指令。这里，DPTR 和 ACC 各自填入想要的地址和偏移量。并且访问目标必须在 IAP 存储空间内，否则读取的数据将不确定。注意使用‘MOVC’指令比使用 Flash 的读取模式更快。

IAP 的 Flash 持久期

内置 Flash 的持久期是 10,000 擦除/写周期，换句话说擦除再写周期不能超过 10,000 次。这样用户必须注意应用中需要频繁更新 IAP 存储空间这一点。

26.5. ISP/IAP 寄存器

下面专门描述ISP，IAP和P页相关的特殊功能寄存器：

IFD: ISP/IAP Flash 数据寄存器

SFR 页 = 0~F

SFR 地址 = 0xE2 复位值 = 1111-1111

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

IFD 是 ISP/IAP/P 页操作的数据端口寄存器。 在 ISP/IAP/P 页写操作时 IFD 的数据将被写入到期望的地址，在 ISP/IAP/P 页读操作时 IFD 的值是读到期望地址的数据。

IFADRH: ISP/IAP 高 8 位地址

SFR 页 = 0~F

SFR 地址 = 0xE3 复位值 = 0000-0000

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

IFADRH 是所有 ISP/IAP 模式下的高 8 位地址。在 P 页模式下没有定义

IFADRL: ISP/IAP 低 8 位地址

SFR 页 = 0~F

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

IFADRH 是所有 ISP/IAP/P 页模式下的低 8 位地址。在闪存页擦除时，IFADRL 可以不用理会。

IFMT: ISP/IAP Flash 模式表

SFR 页 = 0~F

SFR 地址 = 0xE5 复位值 = xxxx-x000

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

Bit 7~4: 保留。当 IFMT 改写时这些位必须写入” 0000_0”

Bit 3~0: ISP/IAP/P 页工作模式选择

MS.2~0	模式
0 0 0	备用
0 0 1	AP/IAP-存储器读
0 1 0	AP/IAP-存储器编程
0 1 1	AP/IAP-存储器页擦除
1 0 0	**P 页 SFR 写**
1 0 1	**P 页 SFR 读**
其它	**保留**

IFMT 是用来选择闪存是用执行众多的 ISP/IAP 功能还是选择 P 页寄存器的访问。

SCMD: 连续命令数据寄存器

SFR 页 = 0~F

SFR 地址 = 0xE6 复位值 = xxxx-xxxx

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

SCMD 是激活 ISP/IAP/P 页 的命令口。如果 SCMD 连续填入 0x46h, 0xB9h 并且 ISPCR.7 = 1，ISP/IAP/P 页被激活。

ISPCR: ISP 控制寄存器

SFR 页 = 0~F

SFR 地址 = 0xE7 复位值 = 0000-xxxx

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

Bit 7: ISPEN, ISP/IAP/P 页操作使能
0:所有的 ISP/IAP/P 页编程/擦除/读都是被禁止的
1:使能 ISP/IAP/P 页编程/擦除/读功能

Bit 6: SWBS, 软件执行起始选择控制
0:复位软件从主存储区开始执行
1:复位软件从 ISP 存储区开始执行

Bit 5: SWRST, 软件复位触发控制.
0: 没有操作
1: 产生软件系统复位，硬件自动清零

Bit 4: CFAIL, ISP/IAP 操作命令失败指示
0:最后一次 ISP/IAP 命令成功
1:最后一次 ISP/IAP 命令失败。失败的原因是闪存访问被阻止

Bit 3~0: 保留。当 ISPCR 被写时这些位软件必须写入”0”

CKCON1:时钟控制寄存器 1

SFR 页 = **0~F 及 P**

SFR 地址 = 0xBF 复位值 = 0x00-0000

7	6	5	4	3	2	1	0
XTOR	--	XCKS5	XCKS4	XCKS3	XCKS2	XCKS1	XCKS0
R	**W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 5~0: 这是设置 OSCin 频率值决定 ISP/IAP 编程的时间基准。根据 OSCin 填入正确的值，如下所示。

[XCKS5~XCKS0] = OSCin – 1, 当 *OSCin=1~40 (MHz).*

例如,
(1) 如果 OSCin=12MHz, 然后 11 填入 [XCKS5~XCKS0], 即 00-1011B.
(2) 如果 OSCin=6MHz, 然后 5 填入[XCKS5~XCKS0], 即 00-0101B.

OSCin	XCKS[4:0]
1MHz	00-0000
2MHz	00-0001
3MHz	00-0010
4MHz	00-0011
……	……

……	……
38MHz	10-0101
39MHz	10-0110
40MHz	10-0111

IAPLB: IAP 低边界

SFR 页 = **P**

SFR 地址 = 0x03 复位值 = 0111-0000

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

Bit 7~0: IAPLB 决定 IAP 存储区的最低边界。因为一个闪存页是 512 字节，所以 IAPLB 必须是偶数。
为了读取 IAPLB，MCU 需要在 P 页里定义 IFADRL 地址，IMFT 模式选择 P 页读及 ISPCR.ISPEN 置位。并且 在 SCMD 依次写入 0x46h 和 0xB9h，这样 IAPLB 的值就会出现在 IFD。写 IAPLB ， 首先 MCU 把新的 IAPLB 设定值写入 IFD ； 其次索引 IFADRL ， 选择 IMFT ， 使能 ISPCR.ISPEN ； 然后设置 SCMD 。这样 IAPLB 就会更新到最新的顺序。

由 IAPLB 及 ISP 起始地址决定的 IAP 存储区见下列表。

IAP 低边界 = IAPLB[7:0] x 256, 和
IAP 高边界 = ISP 起始地址 – 1.

例如， IAPLB=0x60 及 ISP 起始地址是 0x7000, 那么 IAP 存储区就是 0x6000 ~ 0x6FFF。

另外要注意一点，IAP 的低边界地址不能大于 ISP 的起始地址。

26.6. ISP 示例代码

图 26–8 所示 ISP 操作的示例代码

图 26–8. ISP 示例代码

```
;*******************************************************************************************
; ISP 范例程序
;*******************************************************************************************
IFD      DATA   0E2h
IFADRH   DATA   0E3h
IFADRL   DATA   0E4h
IFMT     DATA   0E5h
SCMD     DATA   0E6h
ISPCR    DATA   0E7h
;
       MOV    ISPCR,#10000000b  ;ISPCR.7=1, 使能 ISP

;=================================================================================
; 1. 页擦除模式 (512 字节每页)
;=================================================================================
       ORL    IFMT,#03h     ;MS[2:0]=[0,1,1], 选择页擦除模式
       MOV    IFADRH,??     ; 页地址填写到 IFADRH 及 IFADRL
       MOV    IFADRL,??     ;
       MOV    SCMD,#46h   ;触发 ISP 处理
       MOV    SCMD,#0B9h  ;
       ;Now in processing...( CPU 等待处理完成)

;=================================================================================
; 2. 字节编程模式
;=================================================================================
       ORL    IFMT,#02h     ;MS[2:0]=[0,1,0], 选择字节编程模式
       ANL    ISPCR,#0FAh  ;
       MOV    IFADRH,??     ; 字节地址填写到 IFADRH 及 IFADRL
       MOV    IFADRL,??     ;
       MOV    IFD,??        ; 被编程数据填写到 IFD
       MOV    SCMD,#46h   ; 触发 ISP 处理
       MOV    SCMD,#0B9h  ;
       ;Now in processing...( CPU 等待处理完成)

;=================================================================================
; 3.使用读取模式校验
;=================================================================================

       ANL    IFMT,#0F9h    ;MS1[2:0]=[0,0,1], 选择字节读取模式
       ORL    IFMT,#01h     ;
       MOV    IFADRH,??     ; 字节地址填写到 IFADRH 及 IFADRL
       MOV    IFADRL,??     ;
       MOV    SCMD,#46h   ; 触发 ISP 处理
       MOV    SCMD,#0B9h  ;
       ;Now in processing...( CPU 等待处理完成)
       MOV    A,IFD       ; 数据存在 IFD
       CJNE   A,wanted,ISP_error ; 比较想要的数值
       ...
ISP_error:
       ...
;
```

27. P 页 SFR 访问

MA82G5BXX 内建一个特别的 P 页寄存器 (P 页) 用来存储 MCU 操作的控制寄存器。这些特殊功能寄存器在不同 IFMT 下通过 ISP/IAP 操作来访问。在 P 页访问时，IFADRH 必须设置为”00”及 IFADRL 索引 P 页内特殊功能寄存器地址。如果 IFMT= 04H 则 P 页写操作，在 SCMD 激活之后 IFD 的数据会被载入到 IFADRL 索引的特殊功能寄存器。如果 IFMT= 05H 则 P 页读操作，在 SCMD 激活之后 IFD 的数据将是 IFADRL 索引的特殊功能寄存器(SFR)的值。

下面描述的是 P 页里的特殊功能寄存器(SFR):

IAPLB: IAP 低边界地址

SFR 页 = **P**

SFR 地址 = 0x03 复位值 = 0111-0000

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

Bit 7~0: IAPLB 决定 IAP 存储区的最低边界。因为一个闪存页是 512 字节，所以 IAPLB 必须是偶数
为了读取 IAPLB，MCU 需要在 P 页里定义 IFADRL 地址，IMFT 模式选择 P 页读及 ISPCR.ISPEN 置位。并且 在 SCMD 依次写入 0x46h 和 0xB9h，这样 IAPLB 的值就会出现在 IFD。写 IAPLB ， 首先 MCU 把新的 IAPLB 设定值写入 IFD ； 其次索引 IFADRL ， 选择 IMFT ， 使能 ISPCR.ISPEN ； 然后设置 SCMD 。这样 IAPLB 就会更新到最新的顺序。

由 IAPLB 及 ISP 起始地址决定的 IAP 存储区见下列表。

IAP 低边界= IAPLBx256,和

IAP 高边界 = ISP 其实地址 – 1.

例如， IAPLB=*0x60* 及 ISP 起始地址是 *0x7000*, 那么 IAP 存储区就是 *0x6000* ~ 0x6FFF。

另外要注意一点，IAP 的低边界地址不能大于 ISP 的起始地址。

CKCON2: 时钟控制寄存器 2

SFR 页 = **P**

SFR 地址 = 0x40 复位值 = 0101-0000

7	6	5	4	3	2	1	0
XTGS1	XTGS0	XTALE	IHRCOE	MCKS1	MCKS0	OSCS1	OSCS0
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~6: XTGS1~XTGS0, XTAL 振荡器增益控制寄存器。这两位软件必须写 “01”

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

Bit 5: XTALE,外部晶振(XTAL)使能
0:禁止（XTAL）振荡电路。 此时 XTAL2 及 XTAL1 当做 P6.0 及 P6.1
1:使能（XTAL）振荡电路。如果此位是通过 CPU 软件来设置的话，硬件置 **XTOR** (CKCON1.7)为”1”时表明作为 OSCin 时钟选择的晶振振荡器准备好

Bit 4: IHRCOE, 内部高频 RC 振荡使能
0:禁止内部高频 RC 振荡
1:使能内部高频 RC 振荡。如果此位是通过 CPU 软件来设置的话，则在 IHRCOE 使能之后需要 32 微秒才能稳定输出

Bit 3~2: MCKS[1:0], MCK 时钟源选择

MCKS[1:0]	MCK 时钟源选择	OSCin =12MHz CKMIS = [01]	OSCin =11.059MHz CKMIS = [01]
0 0	OSCin	12MHz	11.059MHz
0 1	CKMI x 4 (ENCKM =1)	24MHz	22.118MHz
1 0	CKMI x 5.33 (ENCKM =1)	32MHz	29.491MHz
1 1	CKMI x 8 (ENCKM =1)	48MHz	44.236MHz

Bit 1~0: OSC[1:0], OSCin 时钟源选择

CKMIS[1:0]	时钟源选择
0 0	IHRCO
0 1	XTAL
1 0	ILRCO
1 1	ECKI, 外部时钟输入(P6.0)作为 OSCin

CKCON3: 时钟控制寄存器 3

SFR 页 = P

SFR 地址 = 0x41 复位值 = 0000-0010

7	6	5	4	3	2	1	0
0	0	FWKP	0	MCKD1	MCKD0	MCDS1	MCDS0
W	**W**	**R/W**	**W**	**R/W**	**R/W**	**R/W**	**R/W**

PCON2: 电源控制寄存器 2

SFR 页 = **P**

SFR 地址 = 0x44 POR = 0011-0101

7	6	5	4	3	2	1	0
AWBOD1	**0**	BO1S1	BO1S0	BO1RE	EBOD1	BO0RE	**1**
R/W	**W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**W**

Bit 7: AWBOD1, 在掉电模式下 BOD1 唤醒
0:在掉电模式下禁止 BOD1 唤醒
1:在掉电模式下 BOD1 保持运行

Bit 6:保留。当 PCON2 写时，这位软件必须写"0"

Bit 5~4: BO1S[1:0].低电压检测 1 监视电压选择。这两位的初始值由 OR1.BO1S1O 和 OR1.BO1S0O 决定

BO1S[1:0]	BOD1 侦测电压
0 0	2.0V
0 1	2.4V
1 0	3.7V
1 1	4.2V

Bit 3: BO1RE, BOD1 复位使能
0:当 BOF1 置位禁止 BOD1 触发系统复位
1:当 BOF1 置位使能 BOD1 触发系统复位

Bit 2: EBOD1,使能 BOD1 监测 VDD 电压下降，监测电压由 BO1S1~0 指定
0:禁止 BOD1 降低芯片的功耗
1:使能 BOD1 监测 VDD 电压下降

Bit 1: BO0RE, BOD0 复位使能
0: 当 BOF0 置位禁止 BOD0 触发系统复位
1: 当 BOF0 置位使能 BOD0 触发系统复位(VDD 遇到 1.7V).

Bit 0: 保留。当 PCON2 写入时，此位软件必须写"1"

SPCON0: 特殊功能寄存器(SFR)页控制 0

SFR 页 = **P**

SFR 地址 = 0x48 POR = 0000-0000

7	6	5	4	3	2	1	0
RTCCTL	P6CTL	P4CTL	WRCTL	CKCTL1	CKCTL0	PWCTL1	PWCTL0
W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: RTCCTL. RTCCR SFR 访问控制
如果 RTCCTL 置位，将禁止在普通页修改 RTCCR SFR。RTCCR 在普通页仅保持 SFR 读取功能。但是在 SFR P 页软件拥有改写权利。

Bit 6: P6CTL. P6 SFR 访问控制
如果 P6CTL 置位，则 P6 禁止在 0~F 页改写。P6 在 0~F 页保持读取。但是在 SFR P 页软件拥有改写权利。

Bit 5: P4CTL. P4 SFR 访问控制
如果 P4CTL 置位，则 P4 禁止在 0~F 页改写。P4 在 0~F 页保持读取。但是在 SFR P 页软件拥有改写权利。

Bit 4: WRCTL. WDTCR SFR 访问控制
如果 WRCTL 置位，则 WRCTL 禁止在 0~F 页改写。WRCTL 在 0~F 页保持读取。但是在 SFR P 页软件拥有改写权利。

Bit 3: CKCTL1. CKCON1 SFR 访问控制
如果 CKCTL1 置位，则 CKCON1 禁止在 0~F 页改写。CKCON1 在 0~F 页保持读取。但是在 SFR P 页软件拥有改写权利。

Bit 2: CKCTL0. CKCON0 SFR 访问控制
如果 CKCTL0 置位，则 CKCON0 禁止在 0~F 页改写。CKCON0 在 0~F 页保持读取。但是在 SFR P 页软件拥有改写权利。

Bit 1: PWCTL1. PCON1 SFR 访问控制
如果 PWCTL1 置位，则 PCON1 禁止在 0~F 页改写。PCON1 在 0~F 页保持读取。但是在 SFR P 页软件拥有改写权利。

Bit 0: PWCTL0. PCON0 SFR 访问控制
如果 PWCTL0 置位，则 PCON0 禁止在 0~F 页改写。PCON0 在 0~F 页保持读取。但是在 SFR P 页软件拥有改写权利。

DCON0:装置控制 0

SFR 页 = P

SFR 地址 = 0x4C 复位值 = 0000-0011

7	6	5	4	3	2	1	0
0	IAPO	0	0	0	IORCTL	RSTIO	OCDE
W	**R/W**	**W**	**W**	**W**	**R/W**	**R/W**	**W**

Bit 7:保留。当 DCON0 写入时，此位软件必须写“**0**”

Bit 6: IAPO, 仅 IAP 功能
0:保留 IAP 区服务于 IAP 功能和程序代码执行。
1: IAP 区禁止程序代码执行并且仅服务于 IAP 功能

Bit 4~3: 保留。当 DCON0 写入时，这些位软件必须写“**0**”

Bit 2: IORCTL, GPIO 复位控制
0:端口 6(Port 6)所有复位事件下保持复位
1:如果此位置位，端口 6(Port 6)仅通过 POR/Ext-Reset/BOR0/BOR1 (如果 BO0RE 或 BO1RE 是使能的)复位

Bit 1: RSTIO, RST 功能为 I/O
0:选择 I/O 引脚功能为 P47
1:选择 I/O 引脚功能为外部复位输入(RST)

Bit 0: OCDE, OCD 使能
0:在 P4.4 和 P4.5 禁止 OCD 接口
1:在 P4.4 和 P4.5 使能 OCD 接口

27.1. P 页示例代码

(1). 规定功能: P 页特殊功能寄存器(SFR)写的通用功能子程序

汇编语言代码范例:

```
_page_p_sfr_write:
page_p_sfr_write:
   MOV    IFADRH,000h              ;

   MOV    ISPCR,#ISPEN             ; 使能 IAP/ISP 功能
   MOV    IFMT,#MS2               ; P 页写，IFMT =0x04

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

   MOV    IFMT,#000h              ; IAP/ISP 备用模式，IFMT =0x00
   ANL    ISPCR,#~ISPEN            ; 禁止 IAP/ISP 功能
   RET
```

C 语言代码范例:

```
void page_p_sfr_write (void)
{
   IFADRH = 0x00;

   ISPCR = ISPEN;                  //使能 IAP/ISP 功能
   IFMT = MS2;                    // P 页写，IFMT =0x04

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

   IFMT = Flash_Standby;           // IAP/ISP 备用模式，IFMT =0x00
   ISPCR &= ~ISPEN;
}
```

(2). 规定功能: 使能 PWCTL0 在 P 页控制 SPCON0

汇编语言代码范例:

```
   MOV    IFADRL,#SPCON0           ;

   ORL    IFD,#PWCTL0              ; 设置 PWCTL0
   CALL   page_p_sfr_write         ;

   MOV    IFD,PCON0                ; 设置 PCON0

   ORL    IFD,#PD                  ; 写 PCON0 并且掉电
   MOV    IFADRL,#PCON0_P          ;
   CALL   page_p_sfr_write         ;
```

C 语言代码范例:

```
   IFADRL = SPCON0;                //

   IFD |= PWCTL0;                  // 设置 PWCTL0
   page_p_sfr_write();             //

   IFD = PCON0;                   // 读取 PCON0

   IFD |= PD;                     // 写 PCON0
   IFADRL = PCON0_P;               //
   page_p_sfr_write();             //
```

28. 辅助特殊功能寄存器

AUXR0: 辅助寄存器 0

SFR 页 = 0 ~ F

SFR 地址 = 0xA1 复位值 = 0000-0000

7	6	5	4	3	2	1	0
P60OC1	P60OC0	P60FD	T0XL	P4FS1	P4FS0	INT1H	INT0H
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7~6: P6.0 输出设定控制位 1 和 0。当选择内部 RC 震荡(IHRCO or ILRCO)作为系统时钟时这两个位才起作用。在此晶振模式下，P6.0 和 P6.1 可以设定为 XTAL2 和 XTAL1。在外部时钟输入模式下，P6.0 仅是时钟输入引脚。在内部震荡模式下，P6.0 提供下表设定选择作为通用输入输出口或时钟源产生器。当 P60OC[1:0] 设定为非 P6.0 时，P6.0 将驱动片内 RC 振荡器(IHRCO 或 ILRCO)输出作为其它设备的时钟源。

P60OC[1:0]	P60 功能	I/O 模式
00	P60	By P6M0.0
01	MCK/1	By P6M0.0
10	MCK/2	By P6M0.0
11	MCK/4	By P6M0.0

详细的时钟信息请参考章节“9 系统时钟”。P6.0 用于时钟输出时，建议置位 P6M0 为 1，设置 P6.0 为推挽输出模式。

Bit 5: P60FD, P6.0 高速驱动
0: P6.0 作为缺省驱动输出
1: P6.0 高速驱动输出使能。如果 P6.0 定义为时钟输出，使能此位 P6.0 的输出频率会大于 12MHz 在 Vdd=5V 或大于 6MHz 在 Vdd=3V

Bit 3~2: P4.4 和 P4.5 交错功能选择

P4FS[1:0]	P4.4	P4.5
00	P4.4	P4.5
01	RXD0	TXD0
10	T0/T0CKO	T1/T1CKO
11	T2EX	T2/T2CKO

Bit 1: INT1H, INT1 高电平/上升沿触发使能
0:保留 nINT1 端口引脚的低电平或下降沿作为 INT1 触发
1:设置 nINT1 端口引脚的高电平或上升沿作为 INT1 触发

Bit 0: INT0H, INT0 高电平/上升沿触发使能
0: 保留 nINT0 端口引脚的低电平或下降沿作为 INT0 触发
1: 设置 nINT0 端口引脚的高电平或上升沿作为 INT0 触发.

AUXR1: 辅助控制寄存器 1

SFR 页 = 0~F

SFR 地址 = 0xA2 复位值 = 0000-0000

7	6	5	4	3	2	1	0
P1KBIH	P3KBIL	P4SPI	P3S1	P3S1MI	P6TWI0	P3CEX	DPS
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7: P1KBIH, KBI 高 4 位端口选择在 P1.3, P1.2, P1.1 和 P1.0

P1KBIH	KBI.7~4
0	P2.6, P2.4, P2.3, P2.2
1	P1.3, P1.2, P1.1, P1.0

Bit 6: P3KBIL, KBI 低 4 位端口选择在 P3.5, P3.4, P3.1 和 P3.0

P3KBIL	KBI.3~0
0	P2.7, P2.5, P2.1, P2.0
1	P3.5, P3.4, P3.1, P3.0

Bit 5: P4SPI, SPI 接口在 P4.1~P4.0 和 P2.1~P2.0

P4SPI	nSS	MOSI	MISO	SPICLK
0	P1.4	P1.5	P1.6	P1.7
1	P2.0	P2.1	P4.1	P4.0

Bit 4: P3S1, 如果 P3CEX 是禁止的，串行口 1(UART1)功能在 P3.3 和 P3.4

P3S1	RXD1	TXD1
0	P1.2	P1.3
1	P3.3	P3.4

Bit 3: P3S1MI, S1MI 功能在 P3.5

P3S1MI	S1MI
0	P1.0
1	P3.5

Bit 2: P6TWI0, TWI0 功能在 P6。当 P60OC[1:0] = “00”时，此功能有效

P6TWI0	TWI0_SCL	TWI0_SDA
0	P4.0	P4.1
1	P6.0	P6.1

Bit 1: P3CEX, CEX5, CEX3 和 CEX1 功能在 P3.5, P3.4 和 P3.3

P3CEX	CEX5	CEX3	CEX1
0	P2.7	P2.5	P2.3
1	P3.5	P3.4	P3.3

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

AUXR2: 辅助寄存器 2

SFR 页 = 0~F
SFR 地址 = 0xA3 复位值 = 0000-0000

7	6	5	4	3	2	1	0
INT3IS1	INT3IS0	INT2IS1	INT2IS0	T1X12	T0X12	T1CKOE	T0CKOE
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7~6: INT3IS1~0, nINT3 输入选择，定义如下表

INT3IS1~0	nINT3
00	P4.5
01	P2.1
10	P1.5
11	P6.0

Bit 5~4: INT2IS1~0, nINT2 输入选择，定义如下表

INT2IS1~0	nINT2
00	P4.4
01	P2.0
10	P1.4
11	P6.1

Bit 3: T1X12,当 C/T=0 时，定时器 1 时钟源选择
0:清零是选择 SYSCLK/12 作为定时器 1 时钟源
1:置位是选择 SYSCLK 作为定时器 1 时钟源

Bit 2: T0X12,当 C/T=0 时，定时器 0 时钟源选择
0: 清零是选择 SYSCLK/12 作为定时器 0 时钟源
1: 置位是选择 SYSCLK 作为定时器 0 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 时钟在 P3.5 输出

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

AUXR3: 辅助寄存器 3

SFR 页 = 0~F

SFR 地址 = 0xA4 复位值 = 0000-0000

7	6	5	4	3	2	1	0
STAF	STOF	BPOC1	BPOC0	GF	P1S0MI	P3ECI	P3TWI1
R/W	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**	**R/W**

Bit 7: STAF, STWI 的开始侦测标志
0: 此位软件写**“0”**清零
1: 硬件置位表明 STWI 总线出现 START

Bit 6: STOF, STWI 的停止侦测标志
0: 此位软件写**“0”**清零
1: 硬件置位表明 STWI 总线出现 STOP

Bit 3:保留。当 AUXR3 写时，此位软件必须写“0”

Bit 5~4: BPOC1~0,报警器输出控制位

BPOC[1:0]	P4.4 功能	I/O 模式
00	P4.4	By P4M0.4
01	ILRCO/64	By P4M0.4
10	ILRCO/32	By P4M0.4
11	ILRCO/16	By P4M0.4

报警器功能在 P4.4，推荐设置 P4M0.4 为“1”使 P4.4 工作在推挽输出模式。

Bit 2: P1S0MI, S0MI 功能在 P1.6

P1S0MI	S0MI
0	P3.2
1	P1.6

Bit 1: P3ECI, ECI 功能在 P3.2

P3ECI	ECI
0	P2.1
1	P3.2

Bit 0: P3TWI1, TWI1 功能在 P3

P3TWI1	TWI1_SCL	TWI1_SDA
0	P1.0	P1.1
1	P3.0	P3.1

SFRPI: SFR 页索引寄存器

SFR 页 = **0~F 及 P**

SFR 地址 = 0xAC 复位值 = xxxx-0000

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

Bit 7~4:保留位。当改写 SFRPI 时，这些位必须软件写“0”

Bit 3~0: SFR 页索引。可用的页仅是页**“0”和“1”**

PIDX[3:0]	选择页
0000	Page 0
0001	Page 1
0010	Page 2
0011	Page 3
……	……
……	……
……	……
1111	Page F

29. 硬件选项

MCU 的硬件选项定义了器件的性能，它不能由软件编程和控制。硬件选项仅能由通用编程器，“Megawin 8051 Writer U1” 或 “Megawin 8051 ICE Adapter”(这个 ICE 也支持 ICP 编程功能。参考章节“30.5 在芯片编程功能”)来编程。整片擦除后，所有的硬件选项被设置成“禁止”状态，没有配置 ISP 空间和 IAP 空间 。**MA82G5BXX** 有下列的硬件选项

LOCK:

☑:使能. 加密上锁，使得用通用编程器读取代码锁定为 0xFF

☐:禁止. 没有上锁

ISP 存储空间

由其指定 ISP 空间的起始地址。它的高边界由 Flash 的结束地址限定，例如：0x7FFF。下表列举了 ISP 空间选项。默认设定， **MA82G5B32** ISP 空间被配置为 1.5K， 并嵌入了 Megawin ISP 引导码来执行 ISP 在线设备 FW 更新。

ISP 空间大小	ISP 起始地址
4K 字节	0x7000
3.5K 字节	0x7200
3K 字节	0x7400
2.5K 字节	0x7600
2K 字节	0x7800
1.5K 字节	0x7A00
1K 字节	0x7C00
无 ISP 空间	--

HWBS:

☑:使能. 上电时，如果 ISP 空间有配置，则 MCU 从 ISP 空间启动

☐:禁止. MCU 总是从 AP 空间启动

HWBS2:

☑:使能. 如果 ISP 空间有配置，不仅上电，而且所有复位都是从 ISP 空间启动

☐:禁止. 由 HWBS 决定 MCU 从哪里启动

IAP-储存区空间:

IAP 存储空间指定用户定义的 IAP 空间。IAP 存储空间可以由硬件选项或者 MCU 软件修改 IAPLB 来配置。默认，它被配置为 1KB

BO1S1O, BO1S0O:

☑,☑:选择 BOD1 检测电压 2.0V.

☑,☐:选择 BOD1 检测电压 2.4V.

☐,☑:选择 BOD1 检测电压 3.7V.

☐,☐:选择 BOD1 检测电压 4.2V.

BO0REO:

☑:使能. BOD0 将触发复位事件使得 CPU 从 AP 程序起始地址允许(1.7V)

☐:禁止. BOD0 不能触发 CPU 复位

BO1REO:

☑:使能. BOD1 (4.2V, 3.7V, 2.4V 或 2.0V)将触发复位事件使得 CPU 从 AP 程序起始地址允许

☐:禁止. BOD1 不能触发 CPU 复位

WRENO:

☑:使能. 置位 WDTCR.WREN 使能 WDTF 产生一个系统复位
☐:禁止. 清零 WDTCR.WREN 禁止 WDTF 产生一个系统复位

NSWDT: 不停止 WDT
☑:使能. 置位 WDTCR.NSW 在掉电模式下使能 WDT 运行(watch 模式)
☐:禁止. 清零 WDTCR.NSW 在掉电模式下禁止 WDT 允许(禁止 Watch 模式)

HWENW:硬件加载“ENW”到 WDTC
☑: 使能。上电后使能 WDT 并且加载 WRENO, NSWDT, HWWIDL 和 HWPS2~0 的内容到 WDTCR
☐: 禁止。上电后 WDT 不会自动使能

HWWIDL, HWPS2, HWPS1, HWPS0:
当 HWENW 使能，上电后这 4 个熔丝位的内容将被加载到 WDTCR

WDSFWP:
☑:使能. WDT 特殊寄存器，WDTCR 的 WREN, NSW, WIDL, PS2, PS1 和 PS0 位,将被写保护
☐:禁止. WDT 特殊寄存器，WDTCR 的 WREN, NSW, WIDL, PS2, PS1 和 PS0 位,由软件自由写

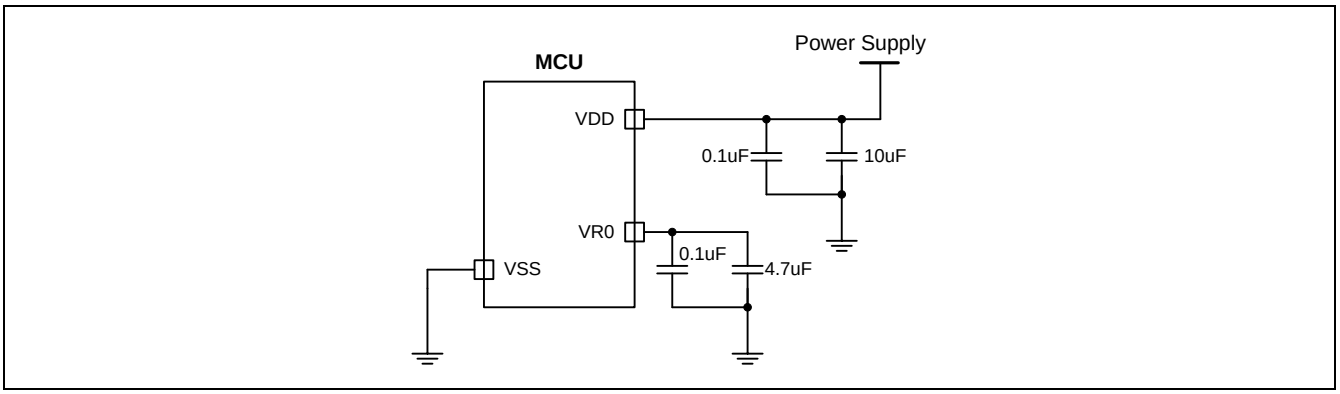
P47EN:
☑:使能. 使能 P4.7 口功能在复位 RST 管脚上
☐:禁止. 禁止 P4.7 口功能，保持复位 RST 管脚复位功能

30. 应用说明

30.1. 电源电路

MA82G5BXX 的工作电源变化可以从 2.0V 到 5.5V 但是增加一些外部去耦和滤波电容是必须的，如图 30–1 所示。

图 30–1. 电源电路



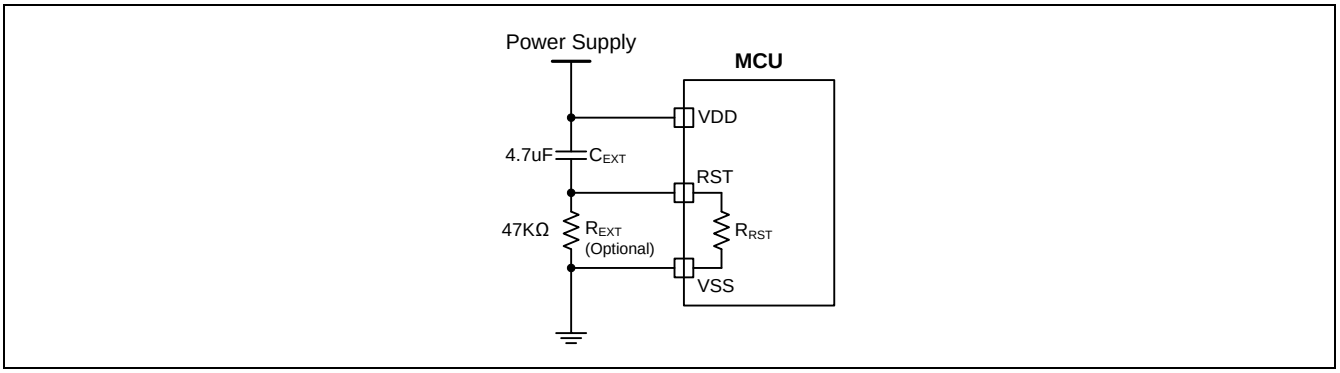
30.2. 复位电路

通常，上电可以成功产生上电复位，然而，为了上电时 MCU 产生一个可靠的复位，有必要加外部复位。外部复位电路如图 30–2 所示，它由一个连接到 VDD（电源）的电容 C_{EXT} 和一个连接到 VSS(地)的电阻组成。

一般的, R_{EXT} 是可选的，因为 RST 引脚有一个内部下拉电阻(R_{RST}). 这个对 VSS 的内部扩散电阻在仅使用一个外部对 VDD 的电容 C_{EXT} 时也可产生一个上电复位

R_{RST} 的值见章节“31.2 直流特性”。

图 30–2. 复位电路



30.3. 外部晶振(XTAL)振荡电路

为了能成功起振 (最大到 24MHz), 电容 C1 和 C2 是必须的，如图 30–3 所示。 通常， C1 和 C2 使用相同的值。表 30–1 列举了 C1 & C2 在不同晶振下的值。

图 30–3. XTAL 振荡电路

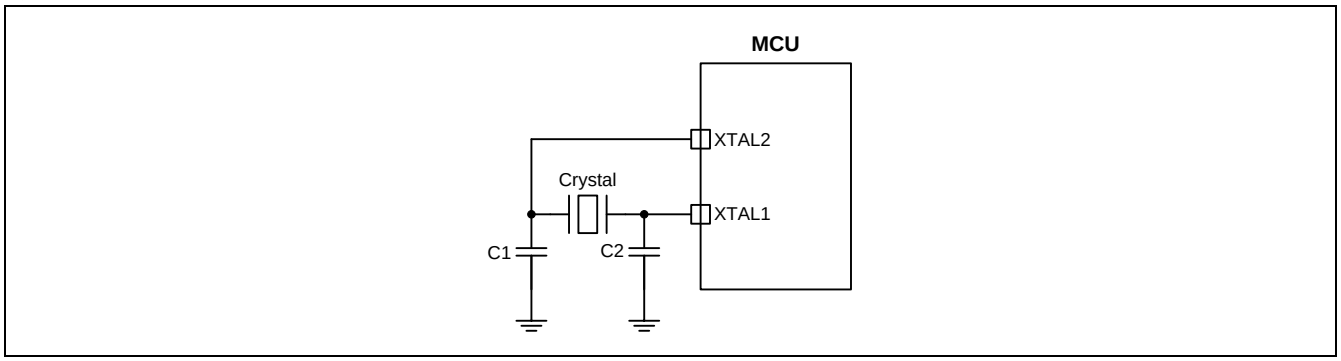


表 30–1. 振荡电路的电容 C1 及 C2 参照表

晶振	C1, C2 电容
16MHz ~ 25MHz	10pF
6MHz ~ 16MHz	15pF
2MHz ~ 6MHz	33pF

30.4. ICP 和 OCD 接口电路

MA82G5BXX 包含一个笙泉专有的在芯片调试接口，它允许在元器件已经安装在产品上在芯片编程(ICP)和在线调试(OCD)。ICP 和 OCD 共享同样的接口使用一个时钟线（ICP_SCL/OCD_SCL）和一个个双向数据线(ICP_SDA/OCD_SDA)完成主机与设备之间的数据传送。

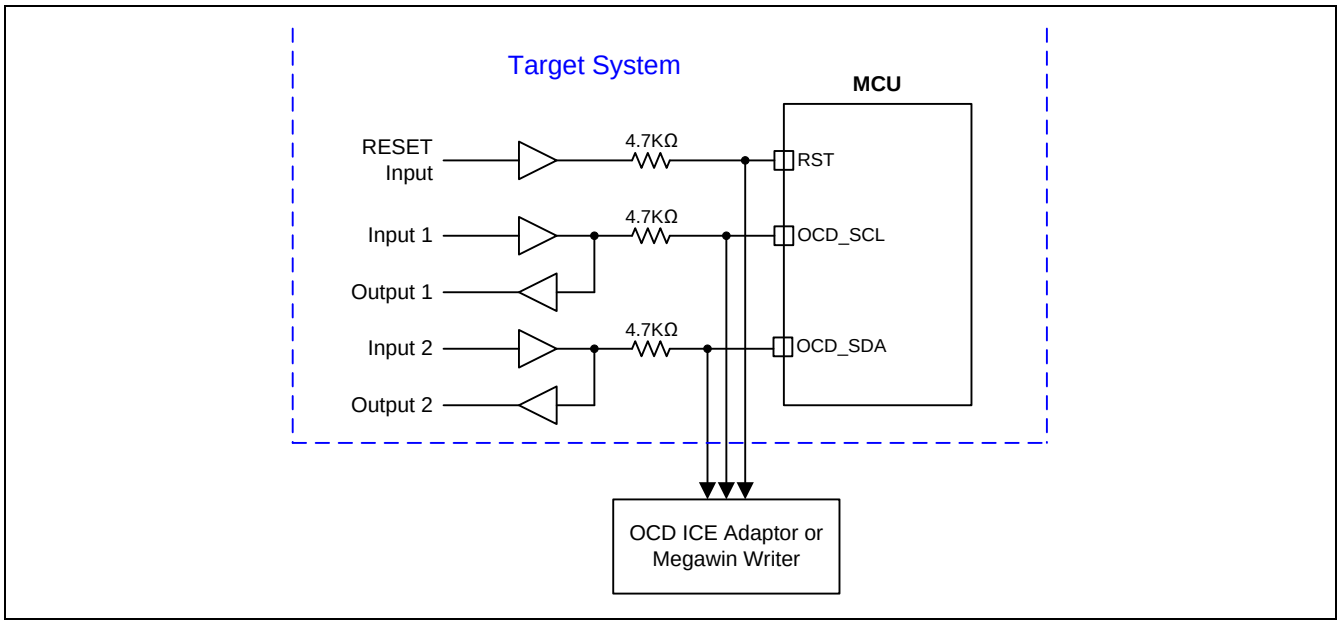
ICP 接口允许的 ICP_SCL/ICP_SDA 引脚与用户应用共享，使得可以实现在芯片 FLASH 编程。这是可行的，因为当芯片在 Halt 状态时执行 ICP 通信，此时芯片上的外围设备和用户软件都是失效的。在 halt 状态，ICP 接口能够安全的“借用”ICP_SCL (P4.4)和 ICP_SDA (P4.5) 引脚。 在大多应用中，必须用外部电阻来隔开 ICP 电路和用户应用电路。图 30–4 展示了一种典型的隔离方法。

强烈建议在目标系统建立 ICP 接口电路。它保留了整个软件编程和硬件选项配置的能力。

上电后，**MA82G5BXX** 的 P4.4 和 P4.5 被配置成 OCD_SCL/OCD_SDA 用于在线调试功能。 这是可行的，因为 OCD 通信是在 CPU Halt 状态下执行，此时用户软件是无效的。 在 halt 状态，OCD 接口可以安全的使用 OCD_SCL(P4.4)和 OSC_SDA(P4.5)引脚。就像上面提到的隔离 ICP 接口，如图 30–4,用外部电阻来隔开 ICP 电路和用户应用电路。

如果用户放弃 OCD 功能，软件可以通过清零 DCON0 的位 0 (OCDE) 来配置 OCD_SCL 和 OCD_SDA 引脚作为 P4.4 口和 P4.5 口。当用户想重新使用 OCD 功能， 用户可以置 OCDE 为 1 来切换 P4.4 和 P4.5 到 OCD_SCL 和 OCD_SDA。或者用 ICP“擦除”在芯片 FLASH 清除用户软件来停止端口的却换。

图 30–4. ICP 和 OCD 接口电路



30.5. 在芯片编程功能

ICP，就像传统的并行编程方式，可以编程 MCU 的任何区域，包括 FLASH 和 MCU 的硬件选项。并且，得益于它专用的串行接口（经由在线调试通道），使得 ICP 可以更新 MCU 而不用从用户的产品上卸下 MCU，就像 ISP 做的那样。

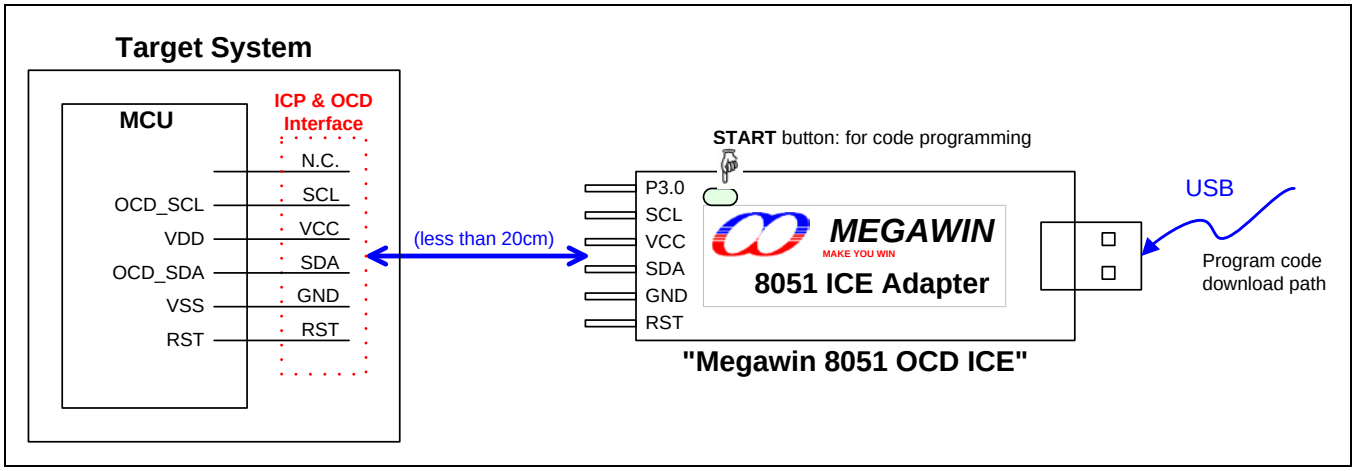
专用的 6 脚 “Megawin 8051 ICE Adapter” 可以支持 **MA82G5BXX** 在线路编程。 “Megawin 8051 ICE Adapter” 有在系统的存储器来存储用户的程序和器件选项。因此，该工具可以完成一个便携的，独立的编程，而不用连线主机，如连接该工具到 PC。下面列举了 ICP 功能的特点：

特点

- 不必在目标芯片上预编程一个引导程序
- 专用串行接口;不占用 IO 口
- 目标芯片不必在运行状态；仅需电源
- 便携，独立的工作，而无需主机的干预

以上特点使得 ICP 非常有利于用户。特别的，在编程数据下载后的便携独立工作，尤其有利于没有 PC 的地方使用。图 30–5 显示了 ICP 独立编程的系统框图。ICP 接口仅需 **5** 个引脚： SDA 线和 SCL 线是串行数据和串行时钟，用来从 6 脚 “Megawin 8051 ICE Adapter”传送编程数据到目标 MCU； RST 线用来暂停 MCU； VCC & GND 是 6 脚 “Megawin 8051 ICE Adapter” 用于便携编程应用的电源输入。 USB 连接器可以直接的插入 PC 的 USB 端口，用来从 PC 下载编程数据到 6-pin “Megawin 8051 ICE Adapter”。

图 30–5. 经由 ICP 的独立编程



30.6. 在线调试功能

MA82G5BXX 预备了一个用于在线仿真(ICE)的 Megawin 专用的在线调试（OCD）接口。这个 OCD 接口提供在芯片和系统不干扰的调试，且不占用任何的目标系统资源。支持 ICE 的几种必要操作，如复位，运行，停止，单步运行，运行到光标和断点设置。

使用 OCD 技术，Megawin 提供 “Megawin 8051 OCD ICE” 给用户，如图 30–6 所示. 用户在开发过程中不必准备任何的开发板，或者用在传统 ICE 探头的转换座。所有这些，用户仅需在系统上保留一个 6-脚的连接器用于专用的 OCD 接口： P3.0, RST, VCC, OCD_SDA, OCD_SCL 和 GND，如图 30–6 所示

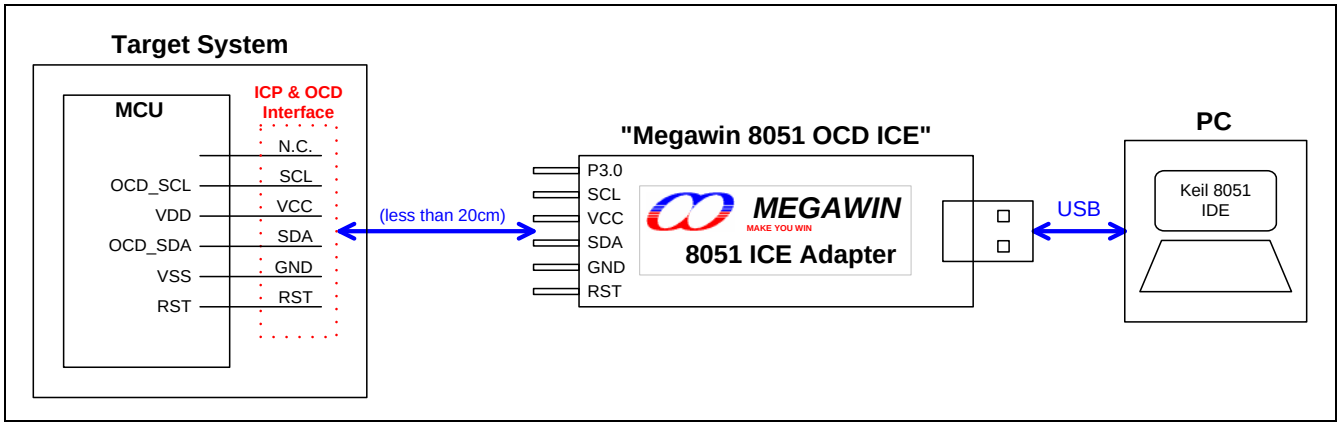
另外，最有力的功能是，它可以直接让用户的系统连接到 Keil 8051 IDE 软件进行仿真,它直接利用 Keil IDE’s dScope-Debugger 功能。当然，所有的特点都基于你使用的 Keil 8051 IDE 软件。

注: “Keil” 是 “Keil Elektronik GmbH and Keil Software, Inc.”的注册商标.

特点

- 笙泉科技专用的 OCD (在芯片调试) 技术
- 在芯片和在系统实时调试
- 用于 OCD 的 5-引脚专用串行接口, 不占用目标资源
- 直接连接 Keil IDE 软件的调试功能
- USB 连接目标板与主机（PC）
- 有用的调试动作：复位，运行，停止，单步运行和运行到光标
- 可编程断点，可在仿真中插入 4 个断点
- 数个帮助调试串口：寄存器/反汇编/监视/存储区窗口
- 源代码级(汇编或 C 语言)调试能力

图 30–6. ICE 系统框图



注: 更多有关 OCD ICE 的详细信息，请联系笙泉。

30.7. 唯一 ID 读的示例代码

MA82G5BXX 每颗芯片锲入了一个唯一 ID 代码(128 位)

唯一 ID 读的示例代码

```
;*************************************************************************************
; ASM 汇编示例程序
;*************************************************************************************
'
            MOV     ISPCR, #80h
            MOV     IFMT, #09h
            MOV     IFADRH, #00h
            MOV     IFADRL, #0F0h

Read_1Byte_unique_ID:
            MOV     SCMD, #046h
            MOV     SCMD, #0B9h
            MOV     A, IFD
             ; A = 唯一 ID 1 字节数据
            INC     IFADRL
           ;重复读 1 字节的唯一 ID 16 次将得到 16 字节的唯一 ID
```

```
;=========================================================================
; C 语言示例代码
;=========================================================================
Unsigned char ID[16];

          ISPCR = 0x80;
          IFMT = 0x09;
          IFADRH = 0;
          IFADRL = 0xF0;

          For (i=0;i<16;i++)
          {
                SCMD = 0x46;
                SCMD = 0xB9;
                ID[i]=IFD;
                ++IFADRL;
          }
```

31. 电气特性

31.1. 最大绝对额定值

参数	范围	单位
环境温度	-40 ~ +85	°C
存储温度	-65 ~ + 150	°C
任意 GPIO 口或 RST 对地电压	-0.5 ~ VDD + 0.5	V
VDD 对地电压	-0.5 ~ +6.0	V
VDD 到地的最大电流	200	mA
任意引脚最大灌电流	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应力额定值，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。

31.2. 直流特性

VDD = 5.0V±10%, VSS = 0V, T_A = 25 ℃且 CPU 空运行，除非另外说明

标号	参数	测试环境	极限			单位
			最小	典型	最大	
输入/输出特性						
V_{IH1}	输入高电平(所有 I/O 口)	除 P6.0, P6.1	0.6			VDD
V_{IH2}	输入高电平(RST, P6.0, P6.1)		0.75			VDD
V_{IL1}	输入低电平(所有 I/O 口)	除 P6.0, P6.1			0.15	VDD
V_{IL2}	输入低电平(RST, P6.0, P6.1)				0.2	VDD
I_{IH}	输入高漏电流 (所有 I/O 口)	V_{PIN} = VDD		0	10	uA
I_{IL1}	逻辑 0 输入电流(P3 在准双向口模式或片内上拉电阻的输入端口)	V_{PIN} = 0.4V		20	50	uA
I_{IL2}	逻辑 0 输入电流(所有仅输入或开漏输出口)	V_{PIN} = 0.4V		0	10	uA
I_{H2L}	逻辑 1 到 0 输入转变电流 (P3 在准双向口模式或片内上拉电阻的输入端口)	V_{PIN} =1.8V		330	500	uA
I_{OH1}	输出高电流(P3 在准双向口模式或片内上拉电阻与其它开漏输出端口)	V_{PIN} =2.4V	150	200		uA
I_{OH2}	输出高电流(所有推挽输出口)	V_{PIN} =2.4V	12			mA
I_{OL1}	输出低电流(所有 I/O 口)	V_{PIN} =0.4V	12			mA
R_{RST}	内部复位下拉电阻			85		Kohm
功耗						
I_{OP2}	一般模式工作电流	SYSCLK = 24MHz @ IHRCO with PLL		4.3		mA
I_{OP3}		SYSCLK = 12MHz @ IHRCO		2.5		mA
I_{OP4}		SYSCLK = 12MHz @ IHRCO with ADC		3.3		mA
I_{OP5}		SYSCLK = 24MHz @ XTAL		5		mA
I_{OP6}		SYSCLK = 12MHz @ XTAL		3.2		mA
I_{OP7}		SYSCLK = 6MHz @ XTAL		2.6		mA
I_{OP8}		SYSCLK = 2MHz @ XTAL		1.9		mA
I_{OPS1}	低速模式工作电流	SYSCLK = 12MHz/128 @ IHRCO		1.04		mA
I_{OPS2}		SYSCLK = 12MHz/128 @ XTAL		1.8		mA
I_{IDLE1}	空闲模式工作电流	SYSCLK = 12MHz @ IHRCO		1.3		mA
I_{IDLE2}		SYSCLK = 12MHz @ XTAL		2		mA
I_{IDLE3}		SYSCLK = 12MHz/128 @ IHRCO		0.85		mA
I_{IDLE4}		SYSCLK = 12MHz/128 @ XTAL		1.6		mA
I_{IDLE5}		SYSCLK = 32KHz @ ILRCO		100		uA
I_{IDLE6}		SYSCLK = 32KHz/128 @ ILRCO, BOD1 disabled		100		uA

I_{SUB1}	副频模式工作电流	SYSCLK = 32KHz @ ILRCO, BOD1 禁止		170		uA
I_{SUB2}		SYSCLK = 32KHz/128 @ ILRCO, BOD1 禁止		100		uA
I_{WAT}	Watch 模式工作电流	WDT = 32KHz @ ILRCO 掉电模式		6		uA
I_{MON1}	Monitor 模式工作电流	掉电模式 BOD1 使能		120		uA
I_{RTC1}	RTC 模式工作电流	RTC operating in PD mode, VDD = 5.0V		10.5		uA
		RTC operating in PD mode, VDD = 3.0V		4.8		
I_{PD1}	掉电模式电流			4		uA
BOD0/BOD1 特性						
V_{BOD0}	BOD0 检测电平	T_A = -40℃ to +85℃	1.6(1)	1.7	1.85(1)	V
V_{BOD10}	BOD1 检测电平在 2.0V	T_A = -40℃ to +85℃	1.85(1)	2.0	2.15(1)	V
V_{BOD11}	BOD1 检测电平在 2.4V	T_A = -40℃ to +85℃	2.25(1)	2.4	2.55(1)	V
V_{BOD12}	BOD1 检测电平在 3.7V	T_A = -40℃ to +85℃	3.55(1)	3.7	3.85(1)	V
V_{BOD13}	BOD1 检测电平在 4.2V	T_A = -40℃ to +85℃	4.05(1)	4.2	4.35(1)	V
I_{BOD1}	BOD1 功耗	T_A = +25℃, VDD=5.0V		110		uA
		T_A = +25℃, VDD=3.3V		95		
工作环境						
V_{PSR}	上电边沿速率	T_A = -40℃ to +85℃	0.05			V/ms
V_{POR1}	Power-on Reset Valid Voltage	T_A = -40℃ to +85℃			0.1	V
V_{OP1}	XTAL 工作速度 0–24MHz	T_A = -40℃ to +85℃	2.7		5.5	V
V_{OP2}	XTAL 工作速度 0-12MHz	T_A = -40℃ to +85℃	2.0		5.5	V
V_{OP4}	CPU 工作速度 0-24MHz	T_A = -40℃ to +85℃	2.4		5.5	V
V_{OP5}	CPU 工作速度 0-12MHz	T_A = -40℃ to +85℃	2.0		5.5	V

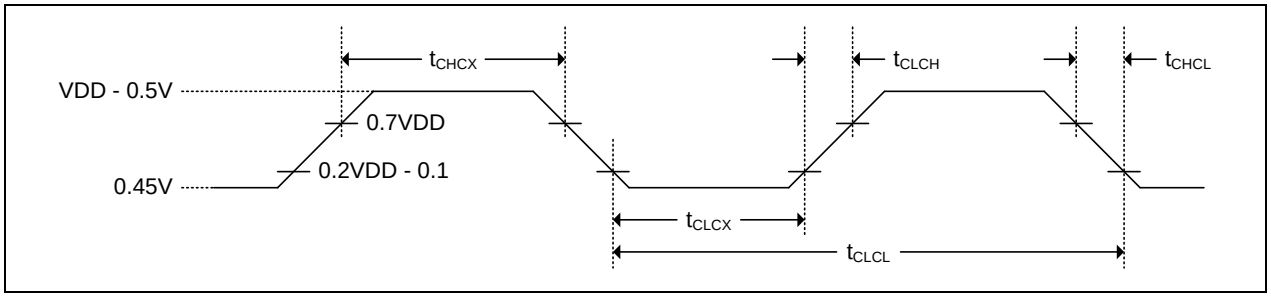
(1) 数据基于特性所得, 非产品测试.

31.3. 外部时钟特性

VDD = 2.0V ~ 5.5V, VSS = 0V, T_A = -40℃ to +85℃, 除非其它说明

标号	参数	振荡				单位
		晶振模式		ECKI 模式		
		最小	最大	最小	最大	
$1/t_{CLCL}$	Oscillator Frequency (VDD = 2.7V ~ 5.5V)	0.032	25	0	25	MHz
$1/t_{CLCL}$	Oscillator Frequency	0.032	12	0	12	MHz
t_{CLCL}	Clock Period	41.6		27.7		ns
t_{CHCX}	High Time	0.4T	0.6T	0.4T	0.6T	t_{CLCL}
t_{CLCX}	Low Time	0.4T	0.6T	0.4T	0.6T	t_{CLCL}
t_{CLCH}	Rise Time		5		5	ns
t_{CHCL}	Fall Time		5		5	ns

图 31–1. 外部时钟驱动波形



31.4. IHRCO 特性

参数	测试环境	极限			单位
		最小	典型	最大	
电源电压		2.0		5.5	V
IHRCO 频率	TA = +25℃, AFS = 0		12		MHz
	TA = +25℃, AFS = 1		11.059		MHz
IHRCO 频率误差	TA = +25℃	-1.0		+1.0	%
(工厂校对)	TA = -40℃ to +85℃	-2.5(1)		+2.5(1)	%
IHRCO 启动时间	TA = -40℃ to +85℃			32(1)	us
IHRCO 功耗	TA = +25℃, VDD=5.0V		600		uA

(1) 数据基于特性所得, 非产品测试

31.5. ILRCO 特性

参数	测试环境	极限			单位
		最小	典型	最大	
电源电压		2.0		5.5	V
ILRCO 频率	TA = +25℃		32		KHz
ILRCO 频率误差	TA = +25℃	-20(1)		+20(1)	%
	TA = -40℃ to +85℃	-40(1)		+40(1)	%

(1) 数据基于特性结果, 非产品测试

31.6. CKM 特性

参数	测试环境	极限			单位
		最小	典型	最大	
电源电压	TA = -40℃ to +85℃	2.4		5.5	V
时钟输入范围	TA = -40℃ to +85℃	5[1]	6	6.5[1]	MHz
CKM 启动时间	TA = -40℃ to +85℃	30[2]		100[2]	us
CKM 功耗	TA = +25℃, VDD=5.0V		400		uA

[1] 数据由设计保证，非产品测试
[2] 数据基于特性所得, 非产品测试

31.7. Flash 特性

参数	测试环境	极限			单位
		最小	典型	最大	
电源电压	TA = -40℃ to +85℃	2.0		5.5	V
Flash 写 (擦除/编程)电压	TA = -40℃ to +85℃	2.2		5.5	V
Flash 擦除/编程 周期	TA = -40℃ to +85℃	20,000			次
Flash 数据保留	TA = +25℃	100			年

31.8. ADC 特性

VDD=5.0V, VREF+=5.0, VREF+ <= VDD, T_A= -40℃ ~ +85℃除非其他说明

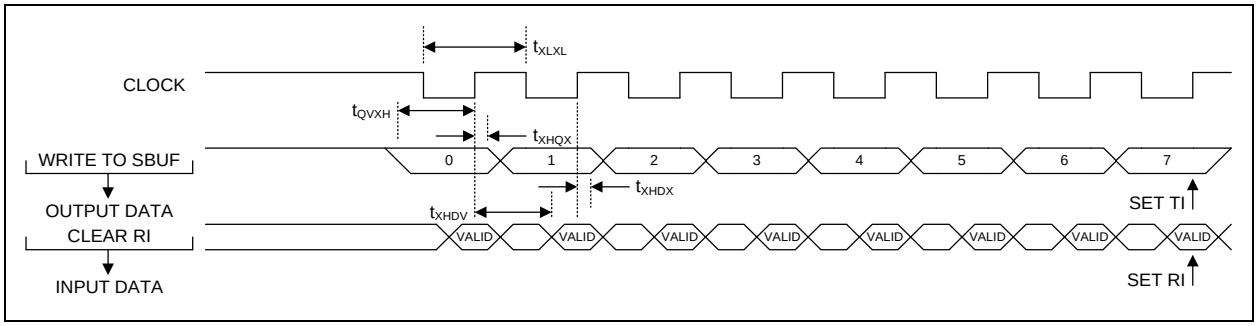
参数	测试环境	极限			单位
		最小	典型	最大	
电源范围					
电源电压		2.4		5.5	V
DC 精度					
分辨率			10		bits
整体非线性		1	2	3	LSB
差分非线性		1	1.5	2	LSB
偏移错误		2	4	6	LSB
转换率					
SAR 转换时钟				6	MHz
在 SAR 时钟里的转换时间			30		clocks
吞吐率				200	ksps
模拟输入					
ADC 输入电压范围		0		VREF+ /VDD	V
输入电容			2		pF
功耗					
电源电流	工作模式, 200 ksps	0.6	0.8	1	mA

31.9. 串行口时序特性

VDD = 5.0V±10%, VSS = 0V, T_A = -40℃ to +85℃, 除非其它说明

标号	参数	URM0X3 = 0		URM0X3 = 1		单位
		最小	最大	最小	最大	
t_{XLXL}	串行口时钟周期	12T		4T		T_{SYSCLK}
t_{QVXH}	设置输出数据到时钟上升沿	10T-20		T-20		ns
t_{XHQX}	上升沿后保持输出数据	T-10		T-10		ns
t_{XHDX}	上升沿后保持输入数据	0		0		ns
t_{XHDV}	时钟上升沿到输入数据有效		10T-20		2T-20	ns

图 31–2. 移位寄存器模式时序图



31.10. SPI 时序特性

VDD = 5.0V±10%, VSS = 0V, T_A = -40℃ to +85℃, 除非其它说明

标号	参数	最小	最大	单位
主模式时序				
t_{MCKH}	SPICLK 高时间	2T		T_{SYSCLK}
t_{MCKL}	SPICLK 低时间	2T		T_{SYSCLK}
t_{MIS}	MISO 有效到 SPICLK 转变边沿	2T+20		ns
t_{MIH}	SPICLK 转变边沿到 MISO 变化	0		ns
t_{MOH}	SPICLK 转变边沿到 MOSI 变化		10	ns
从模式时序				
t_{SE}	nSS 下降沿到第一个 SPICLK 边沿	2T		T_{SYSCLK}
t_{SD}	最后一个 SPICLK 边沿到 nSS 上升沿	2T		T_{SYSCLK}
t_{SEZ}	nSS 下降沿到 MISO 有效		4T	T_{SYSCLK}
t_{SDZ}	nSS 上升沿到 MISO 高阻		4T	T_{SYSCLK}
t_{CKH}	SPICLK 高时间	4T		T_{SYSCLK}
t_{CKL}	SPICLK 低时间	4T		T_{SYSCLK}
t_{SIS}	MOSI 有效到 SPICLK 采样边沿	2T		T_{SYSCLK}
t_{SIH}	SPICLK 采样边沿到 MOSI 变化	2T		T_{SYSCLK}
t_{SOH}	SPICLK 移位边沿到 MISO 变化		4T	T_{SYSCLK}
t_{SLH}	最后的 SPICLK 边沿到 MISO 变化 (仅 CPHA = 1)	1T	2T	T_{SYSCLK}

图 31–3. SPI 主机传送波形 CPHA=0

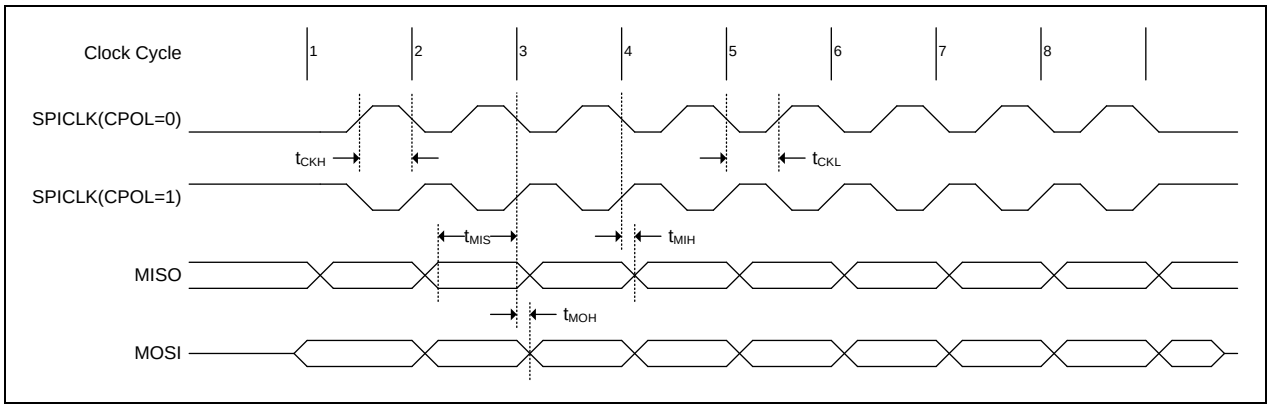


图 31–4. SPI 主机传送波形 CPHA=1

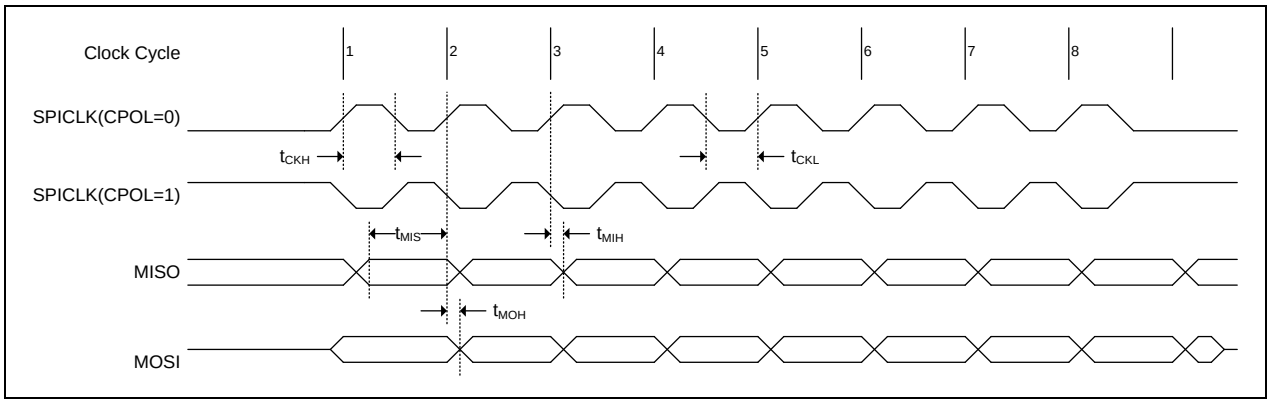


图 31–5. SPI 从机传送波形 CPHA=0

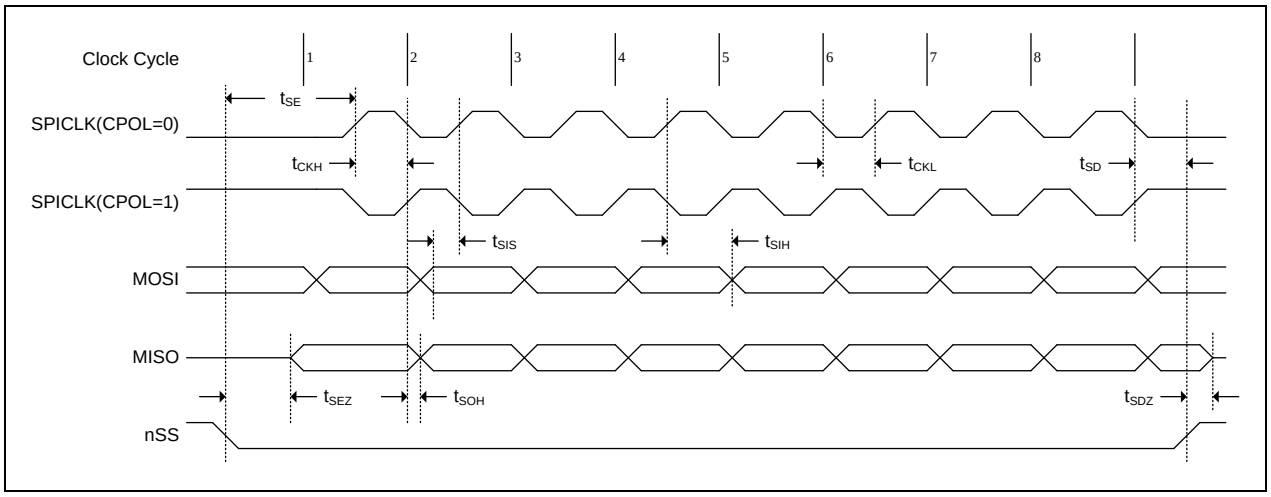
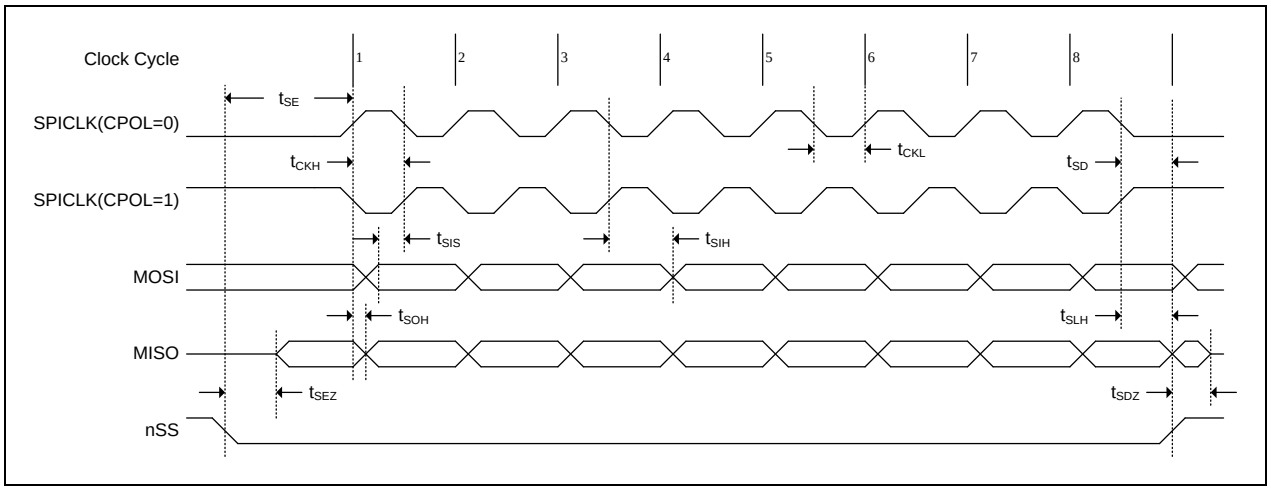


图 31–6. SPI 从机传送波形 CPHA=1



32. 指令集

表 32–1. 指令集

助记符	描述	字节	周期 时钟
数据传送			
MOV A,Rn	寄存器Rn中的内容送到累加器中	1	1
MOV A,direct	直接地址单元中的内容送到累加器中	2	2
MOV A,@Ri	工作寄存器Ri指向的地址单元中的内容送到累加器中	1	2
MOV A,#data	立即数送到累加器中	2	2
MOV Rn,A	累加器中内容送到寄存器Rn中	1	2
MOV Rn,direct	直接寻址单元中的内容送到寄存器Rn中	2	4
MOV Rn,#data	立即数直接送到寄存器Rn中	2	2
MOV direct,A	累加器送到直接地址单元	2	3
MOV direct,Rn	寄存器Rn中的内容送到直接地址单元	2	3
MOV direct,direct	直接地址单元中的内容送到另一个直接地址单元	3	4
MOV direct,@Ri	工作寄存器Ri指向的地址单元中的内容送到直接地址单元	2	4
MOV direct,#data	立即数送到直接地址单元	3	3
MOV @Ri,A	累加器送到以工作寄存器Ri指向的地址单元中	1	3
MOV @Ri,direct	直接地址单元中内容送到以工作寄存器Ri指向的地址单元	2	3
MOV @Ri,#data	立即数送到以工作寄存器Ri指向的地址单元中	2	3
MOV DPTR,#data16	16位常数的高8位送到DPH，低8位送到DPL	3	3
MOVC A,@A+DPTR	以DPTR为基地址变址寻址单元中的内容送到累加器中	1	4
MOVC A,@A+PC	以PC为基地址变址寻址单元中的内容送到累加器中	1	4
MOVX A,@Ri	内置RAM（8位地址）的数据送入累加器中	1	Not Support
MOVX A,@DPTR	寄存器Ri指向扩展RAM地址(8位地址)中的内容送到ACC	1	Not Support
MOVX @Ri,A	数据指针指向扩展RAM地址(16位地址)中的内容送到ACC	1	Not Support
MOVX @DPTR,A	累加器中的内容送到寄存器Ri指向的扩展RAM地址（8位	1	Not Support
MOVX A,@Ri	累加器中的内容送到寄存器Ri指向的扩展RAM地址（16	1	Not Support
MOVX A,@DPTR	外部RAM（16位地址）的数据送入累加器中	1	Not Support
MOVX @Ri,A	寄存器Ri指向片外RAM地址中的内容送到ACC中	1	Not Support
MOVX @DPTR,A	数据指针指向片外RAM地址（16位地址）中内容送到	1	Not Support
PUSH direct	直接地址单元中的数据压入堆栈中	2	4
POP direct	出栈数据送到直接地址单元中	2	3
XCH A,Rn	累加器与寄存器Rn中的内容互换	1	3
XCH A,direct	累加器与直接地址单元中的内容互换	2	4
XCH A,@Ri	累加器与工作寄存器Ri指向的地址单元中内容互换	1	4
XCHD A,@Ri	累加器与工作寄存器Ri指向的地址单元中内容低半字节互	1	4
算术运算			
ADD A,Rn	将寄存器Rn中的内容加到累加器中	1	2
ADD A,direct	直接地址单元中的内容加到累加器中	2	3
ADD A,@Ri	寄存器工作寄存器Ri指向的地址单元中的内容加到累加器	1	3
ADD A,#data	立即数加到累加器中	2	2
ADDC A,Rn	累加器与工作寄存器Rn中的内容、连同进位位相加，结	1	2
ADDC A,direct	累加器与直接地址单元的内容、连同进位位相加，结果存	2	3
ADDC A,@Ri	累加器与工作寄存器Ri指向的地址单元中的内容、连同进	1	3
ADDC A,#data	累加器与立即数、连同进位位相加，结果存在累加器中	2	2
SUBB A,Rn	累加器与工作寄存器中的内容、连同借位位相减，结果存	1	2
SUBB A,direct	累加器与直接地址单元中的内容、连同借位位相减，结果	2	3

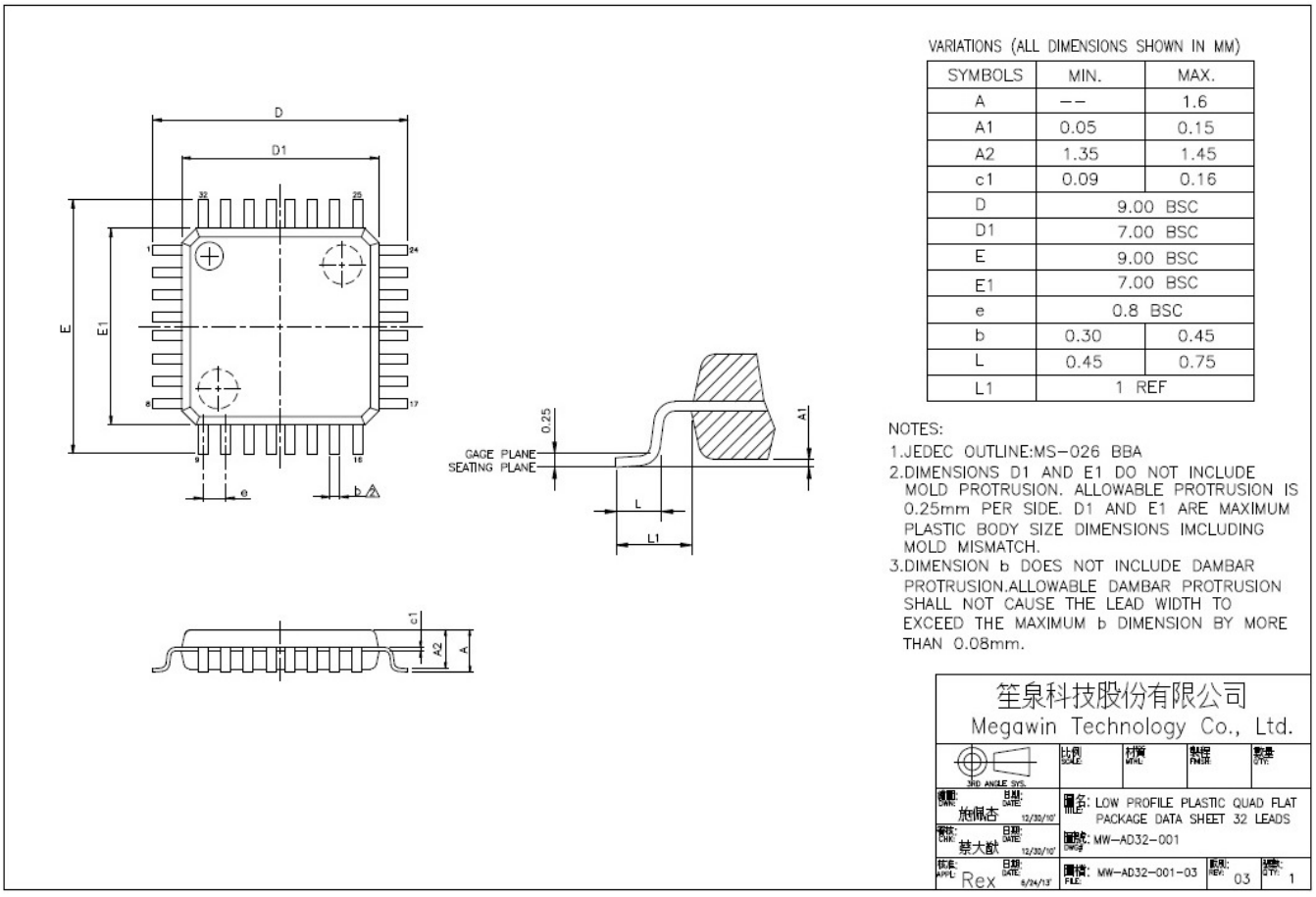
SUBB A,@Ri	累加器与工作寄存器Ri指向的地址单元中内容、连同借位	1	3
SUBB A,#data	累加器与立即数、连同借位位相减，结果存在累加器中	2	2
INC A	累加器中的内容加1	1	2
INC Rn	寄存器Rn的内容加1	1	3
INC direct	直接地址单元中的内容加1	2	4
INC @Ri	工作寄存器Ri指向的地址单元中的内容加1	1	4
DEC A	数据指针DPTR的内容加1	1	2
DEC Rn	累加器中的内容减1	1	3
DEC direct	寄存器Rn中的内容减1	2	4
DEC @Ri	直接地址单元中的内容减1	1	4
INC DPTR	工作寄存器Ri指向的地址单元中的内容减1	1	1
MUL AB	ACC中内容与寄存器B中内容相乘，其结果低位存在ACC	1	4
DIV AB	ACC中内容除以寄存器B中内容，商存在ACC，而余数存	1	5
DA A	ACC十进制调整	1	4
逻辑运算			
ANL A,Rn	累加器和寄存器Rn中的内容相“与”	1	2
ANL A,direct	累加器和直接地址单元中的内容相“与”	2	3
ANL A,@Ri	累加器和工作寄存器Ri指向的地址单元中的内容相“与”	1	3
ANL A,#data	累加器和立即数相“与”	2	2
ANL direct,A	直接地址单元中的内容和累加器相“与”	2	4
ANL direct,#data	直接地址单元中的内容和立即数相“与”	3	4
ORL A,Rn	累加器和寄存器Rn中的内容相“或”	1	2
ORL A,direct	累加器和直接地址单元中的内容相“或”	2	3
ORL A,@Ri	累加器和工作寄存器Ri指向的地址单元中的内容相“或”	1	3
ORL A,#data	累加器和立即数相“或”	2	2
ORL direct,A	直接地址单元中的内容和累加器相“或”	2	4
ORL direct,#data	直接地址单元中的内容和立即数相“或”	3	4
XRL A,Rn	累加器和寄存器Rn中的内容相“异或”	1	2
XRL A,direct	累加器和直接地址单元中的内容相“异或”	2	3
XRL A,@Ri	累加器和工作寄存器Ri指向的地址单元中的内容相“异或”	1	3
XRL A,#data	累加器和立即数相“异或”	2	2
XRL direct,A	直接地址单元中的内容和累加器相“异或”	2	4
XRL direct,#data	直接地址单元中的内容和立即数相“异或”	3	4
CLR A	累加器内容清“0”	1	1
CPL A	累加器按位取反	1	2
RL A	累加器循环左移一位	1	1
RLC A	累加器连同进位位CY循环左移一位	1	1
RR A	累加器循环右移一位	1	1
RRC A	累加器连同进位位CY循环右移一位	1	1
SWAP A	累加器高低半字节互换	1	1
位逻辑运算			
CLR C	清“0”进位位	1	1
CLR bit	清“0”直接地址位	2	4
SETB C	置“1”进位位	1	1
SETB bit	置“1”直接地址位	2	4
CPL C	进位位求反	1	1
CPL bit	直接地址位求反	2	4
ANL C,bit	进位位和直接地址位相“与”	2	3
ANL C,/bit	进位位和直接地址位的反码相“与”	2	3
ORL C,bit	进位位和直接地址位相“或”	2	3

ORL C,/bit	进位位和直接地址位的反码相“或”	2	3
MOV C,bit	直接地址位数据送入进位位	2	3
MOV bit,C	进位位数据送入直接地址位	2	4
位逻辑跳转			
JC rel	进位位为“1”则转移	2	3
JNC rel	进位位为“0”则转移	2	3
JB bit,rel	直接地址位为“1”则转移	3	4
JNB bit,rel	直接地址位为“0”则转移	3	4
JBC bit,rel	直接地址位为“1”则转移，且清“0”该位	3	5
程序跳转			
ACALL addr11	绝对短调用子程序，2K字节（页内）空间限制	2	6
LCALL addr16	绝对长调用子程序，64K字节空间限制	3	6
RET	子程序返回	1	4
RETI	中断子程序返回	1	4
AJMP addr11	绝对短转移，2K字节（页内）空间限制	2	3
LJMP addr16	绝对长转移，64K字节空间限制	3	4
SJMP rel	相对转移	2	3
JMP @A+DPTR	转移到DPTR加ACC所指间接地址	1	3
JZ rel	累加器为“0”则转移	2	3
JNZ rel	累加器不为“0”则转移	2	3
CJNE A,direct,rel	累加器中的内容不等于直接地址单元的内容，则转移到偏	3	5
CJNE A,#data,rel	累加器中的内容不等于立即数，则转移到偏移量所指向的	3	4
CJNE Rn,#data,rel	寄存器Rn中的内容不等于立即数，则转移到偏移量所指	3	4
CJNE @Ri,#data,rel	工作寄存器Ri指向的地址单元中的内容不等于立即数，则	3	5
DJNZ Rn,rel	寄存器Rn中的内容减1，如不等于0，则转移到偏移量所	2	4
DJNZ direct,rel	直接地址单元中的内容减1，如不等于0，则转移到偏移	3	5
NOP	空操作指令	1	1

33. 封装尺寸

33.1. LQFP-32 (7mm X 7mm)

图 33–1. LQFP-32 (7mm X 7mm)



VARIATIONS (ALL DIMENSIONS SHOWN IN MM)

SYMBOLS	MIN.	MAX.
A	--	1.6
A1	0.05	0.15
A2	1.35	1.45
c1	0.09	0.16
D	9.00 BSC	
D1	7.00 BSC	
E	9.00 BSC	
E1	7.00 BSC	
e	0.8 BSC	
b	0.30	0.45
L	0.45	0.75
L1	1 REF	

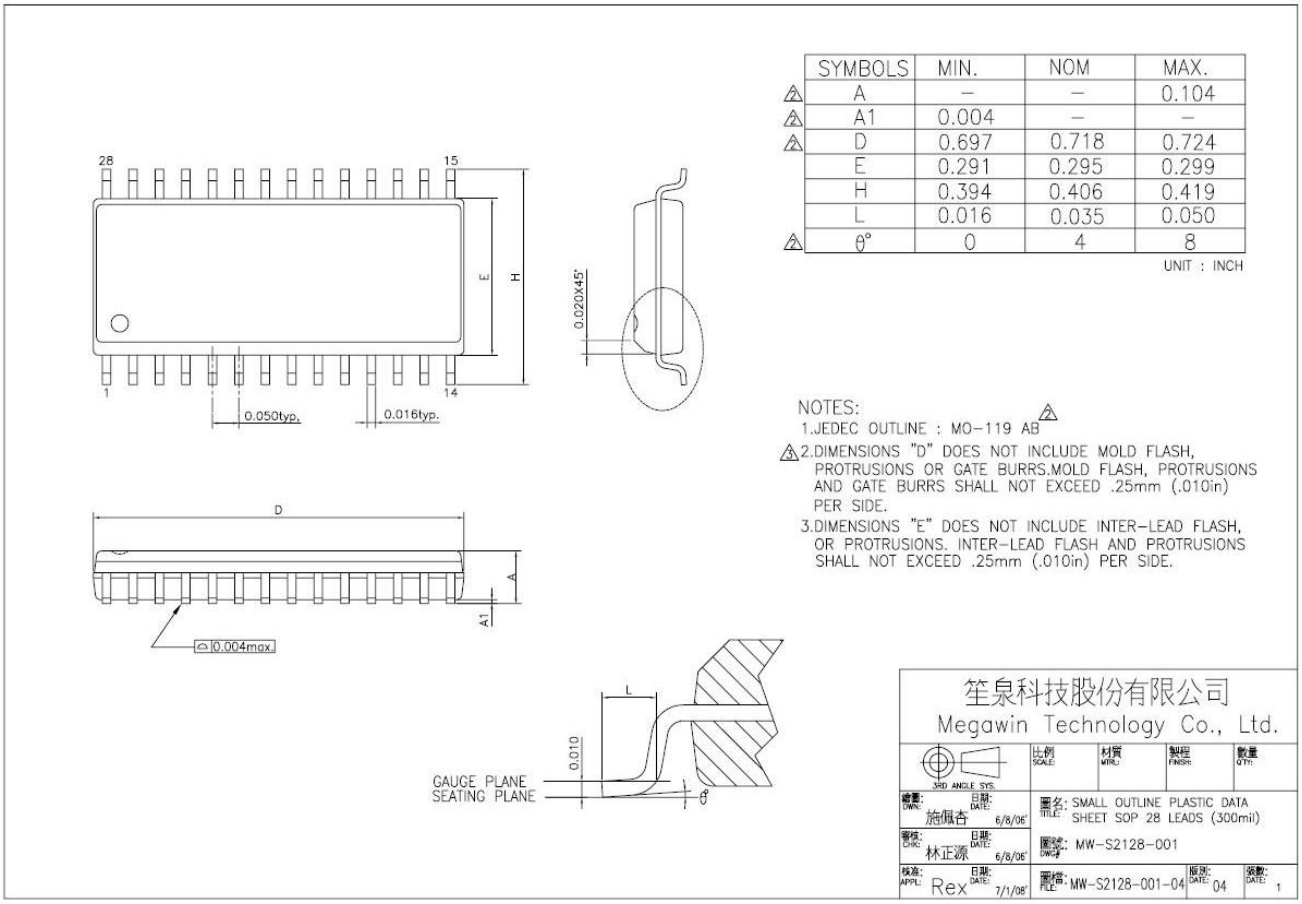
NOTES:
1. JEDEC OUTLINE:MS-026 BBA
2. DIMENSIONS D1 AND E1 DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE PROTRUSION IS 0.25mm PER SIDE. D1 AND E1 ARE MAXIMUM PLASTIC BODY SIZE DIMENSIONS IMCLUDING MOLD MISMATCH.
3. DIMENSION b DOES NOT INCLUDE DAMBAR PROTRUSION.ALLOWABLE DAMBAR PROTRUSION SHALL NOT CAUSE 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 DWN: 旅国杏 日期 DATE: 12/30/'10	圖名 TITLE: LOW PROFILE PLASTIC QUAD FLAT PACKAGE DATA SHEET 32 LEADS			
審核 CHK: 蔡大猷 日期 DATE: 12/30/'10	圖號 DWG#: MW-AD32-001			
核准 APVL: Rex 日期 DATE: 6/24/'13	圖檔 FILE: MW-AD32-001-03	版次 REV: 03	總數 QTY: 1	

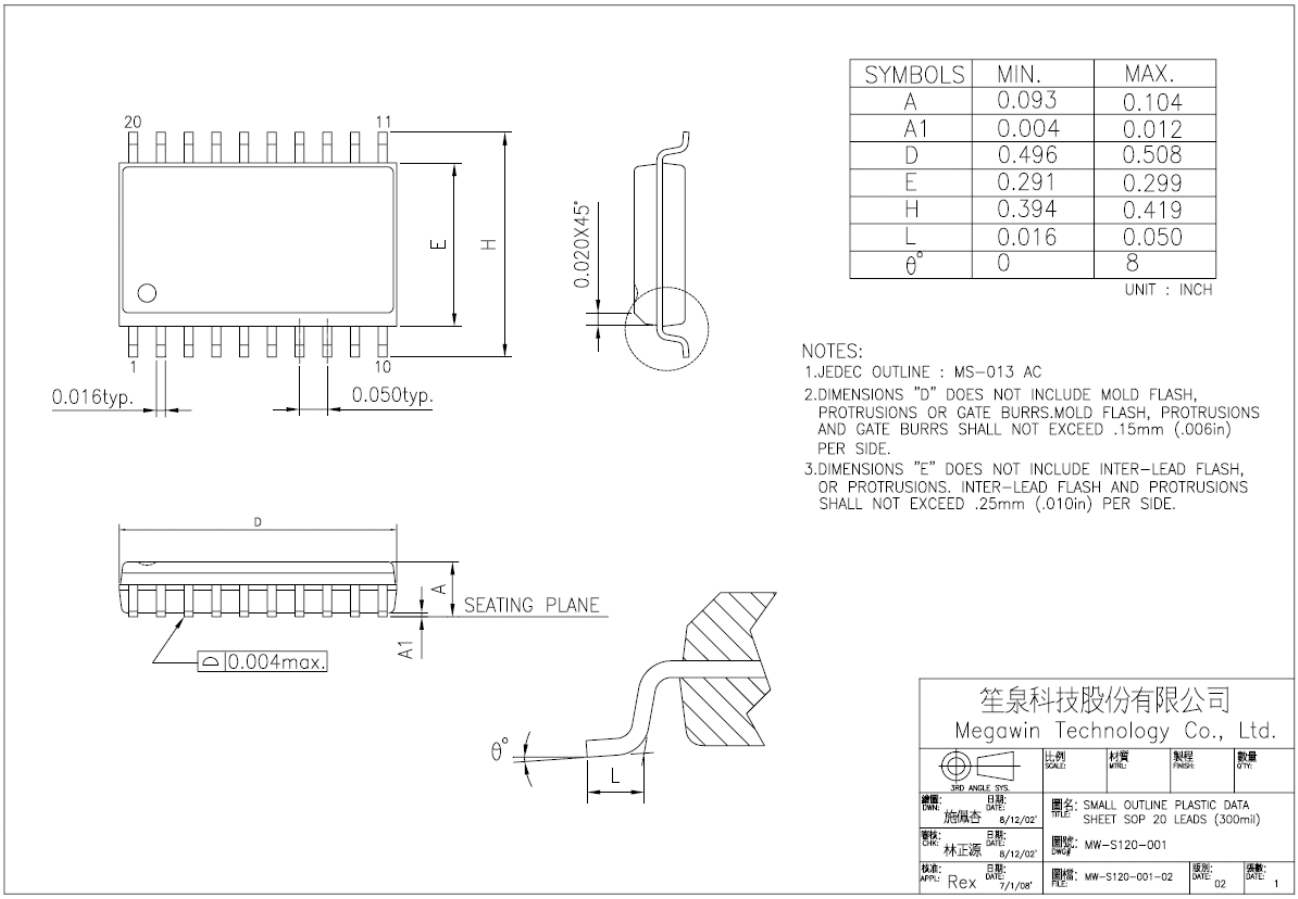
33.2. SOP-28

图 33–2. SOP-28



33.3. SOP-20

图 33–3. SOP-20



SYMBOLS	MIN.	MAX.
A	0.093	0.104
A1	0.004	0.012
D	0.496	0.508
E	0.291	0.299
H	0.394	0.419
L	0.016	0.050
θ°	0	8

UNIT : INCH

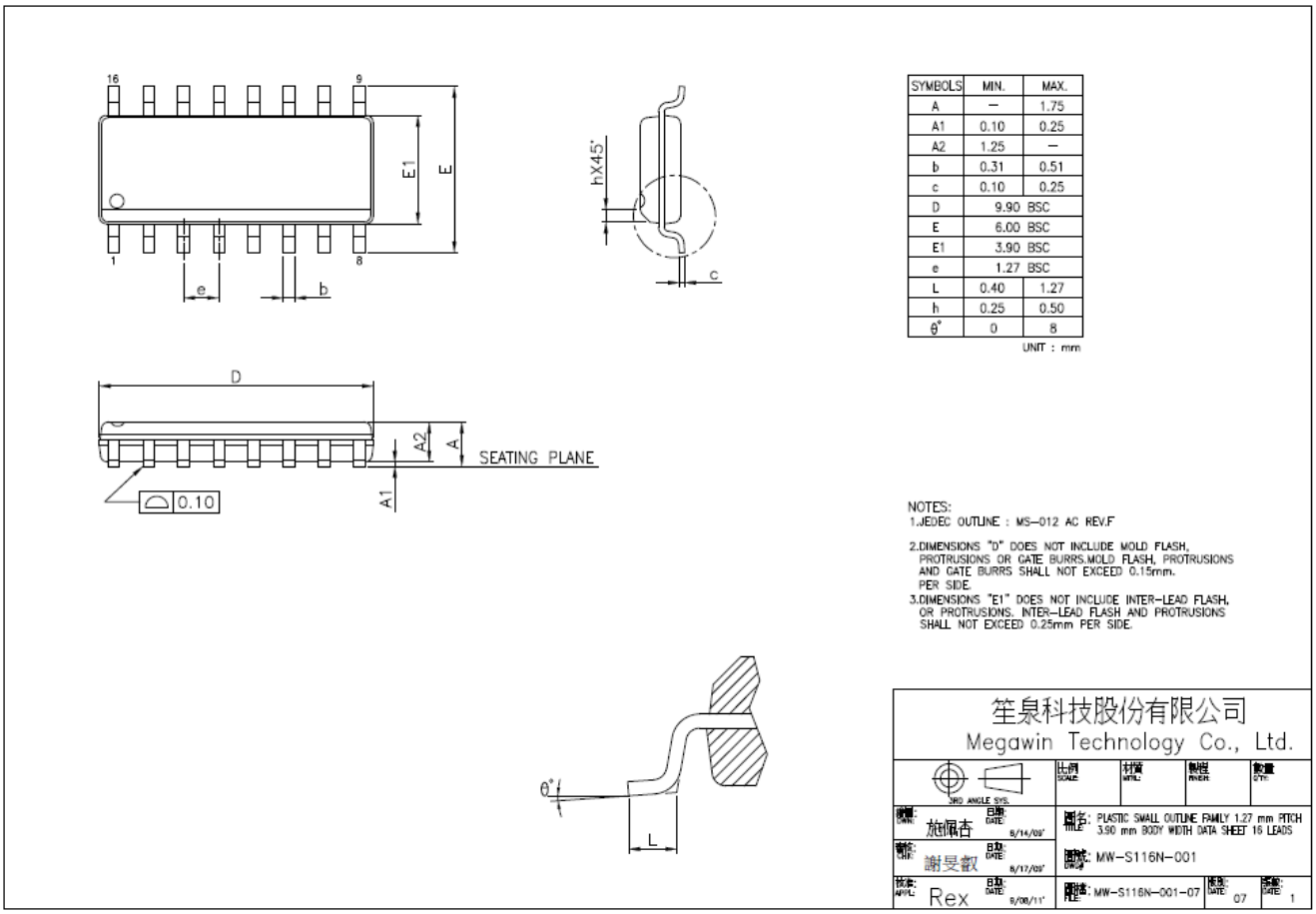
NOTES:
1.JEDEC OUTLINE : MS-013 AC
2.DIMENSIONS "D" DOES NOT INCLUDE MOLD FLASH, PROTRUSIONS OR GATE BURRS.MOLD FLASH, PROTRUSIONS AND GATE BURRS SHALL NOT EXCEED .15mm (.006in) PER SIDE.
3.DIMENSIONS "E" DOES NOT INCLUDE INTER-LEAD FLASH, OR PROTRUSIONS. INTER-LEAD FLASH AND PROTRUSIONS SHALL NOT EXCEED .25mm (.010in) PER SIDE.

笙泉科技股份有限公司
Megawin Technology Co., Ltd.

3RD ANGLE SYS.	比例 SCALE:	材質 MTRL:	製程 FINISH:	數量 QTY:
繪圖: 旭福杏 日期: 8/12/02'	圖名: SMALL OUTLINE PLASTIC DATA SHEET SOP 20 LEADS (300mil)			
審核: 林正源 日期: 8/12/02'	圖號: MW-S120-001			
核准: Rex 日期: 7/1/08'	圖檔: MW-S120-001-02	版次: 02	張數: 1	

33.4. SOP-16

图 33–4. SOP-16(150mil)



SYMBOLS	MIN.	MAX.
A	—	1.75
A1	0.10	0.25
A2	1.25	—
b	0.31	0.51
c	0.10	0.25
D	9.90 BSC	
E	6.00 BSC	
E1	3.90 BSC	
e	1.27 BSC	
L	0.40	1.27
h	0.25	0.50
θ°	0	8

UNIT : mm

NOTES:
1.JEDEC OUTLINE : MS-012 AC REV.F
2.DIMENSIONS "D" DOES NOT INCLUDE MOLD FLASH, PROTRUSIONS OR GATE BURRS.MOLD FLASH, PROTRUSIONS AND GATE BURRS SHALL NOT EXCEED 0.15mm. PER SIDE.
3.DIMENSIONS "E1" DOES NOT INCLUDE INTER-LEAD FLASH, OR PROTRUSIONS. INTER-LEAD FLASH AND PROTRUSIONS SHALL NOT EXCEED 0.25mm PER SIDE.

笙泉科技股份有限公司
Megawin Technology Co., Ltd.

3RD ANGLE SYS.	比例 SCALE	材質 MTRL	製程 PROC.	數量 QTY
設計: 施佩杏	日期: 8/14/09'	圖名: PLASTIC SMALL OUTLINE FAMILY 1.27 mm PITCH 3.90 mm BODY WIDTH DATA SHEET 16 LEADS		
審核: 謝旻叡	日期: 8/17/09'	圖號: MW-S116N-001		
核准: Rex	日期: 9/08/11'	檔名: MW-S116N-001-07	版次: 07	張次: 1

34. 版本历史

表 34–1. 版本历史

版本	描述	日期
v0.55	初版开放	2013/10/14
v0.56	1. 增加例程 2. 補充 DC/AC 数据	2013/10/23
v0.58	1. 修正图 25-1 的 Flash 空间配置 2. 补充封装尺寸图	2013/11/01
v0.59	1. 删除 SOP-28 封装 2. 补充 ADC 规格数据	2013/11/27
v0.60	1. 修正错误 PCON3 为-- 软件可以通过清零 **DCON0** 的位 0 (OCDE)（239 页） 2. ADC 章节补充 VRS2~VRS0 ADC 电压参考源选择说明	2014/02/24
v0.61	补充 PWM 模式增强 PWM 的控制说明	2014/03/31
v0.62	增加特别版本，新增 SOP-28, SOP-20, SOP16 封装 修正 OPTION 错误说明	2015/01/21
v1.00	增加 PAOE 控制说明	2015/04/28
v1.01	1. 修正 SOP16 封装格式 2. 修正 DCON0 初始默认值 3. 修正 TWSI 示例代码 4. 更改 TWSI 与 TWI2 的名称为 TWI0 与 STWI	2015/06/01
v1.02	修正 S1CFG 内容说明	2015/09/25
v1.03	1. 增加采购信息章节 2. 修正 ISP/IAP 范例代码	2015/11/13

免责声明

在此，笙泉（Megawin）代表“*Megawin Technology Co., Ltd.*”

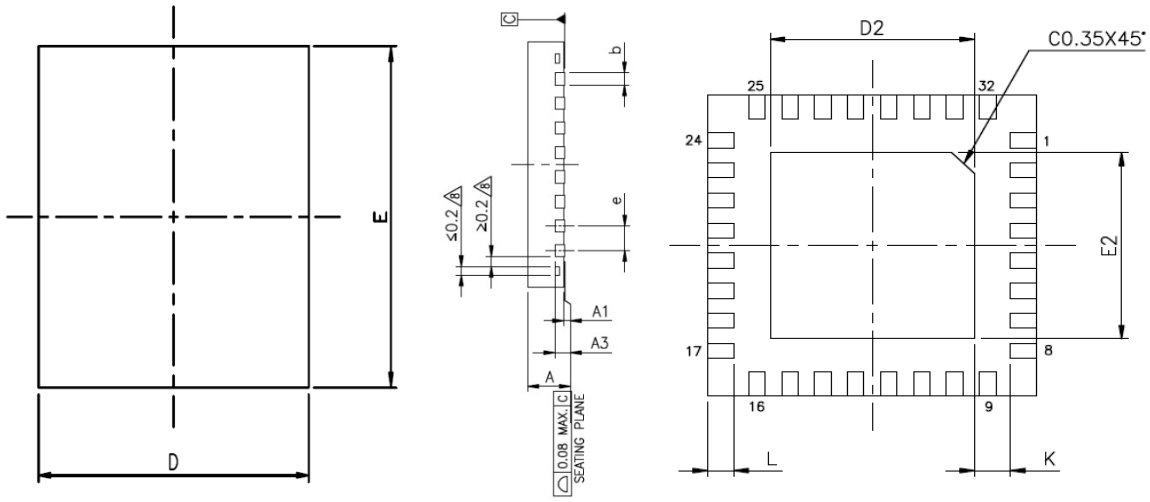
生命支援

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。

QFN 32P package dimension



Symbol	Dimensions in mm		
JEDEC	MO-220		
PKG	WQFN(X532)		
Symbol	Min.	Nom.	Max.
A	0.70	0.75	0.80
A1	0.00	0.02	0.05
A3	0.203 REF.		
b	0.18	0.25	0.30
D	5.00 BSC		
E	5.00 BSC		
e	0.50 BSC		
L	0.35	0.40	0.45
K	0.20	----	----