凌阳科技

SPMC65 系列单片机编程指南

V1.1 – 2005.6.1

凌阳单片机技术资料

http://www.sunplusmcu.com

版权声明

目 录

修订记录

日期	版本	编写及修订说明
2005/05/01	1.0	初始版本
2005/06/01	1.1	第二版本

1 简介

1.1 简述

SPMC65X 系列是由凌阳公司设计开发的 8 位微控制器。每款芯片都独具特色，同时凌阳公司还开发了一款仿真芯片 ECMC653，专门用于 SPMC65X 系列的仿真。

采用 SPMC65 CPU 核，凌阳公司新开发了功能强大的 8 位 SPMC65 系列 CPU。该系列 CPU 具有可编程的通用 I/O 端口、不同大小的 ROM 和 RAM 区、8 位/16 位定时/计数器、强大的 CCP (Capture/Compare/PWM)功能模块和看门狗复位电路等。并采用先进的微米制造工艺，保证了产品高的电磁兼容性和可靠性。

除此之外，部分 SPMC65X 系列芯片具备高吸入电流和慢速输出的端口、丰富的外部中断源、低电压复位、ADC、PWM、标准通讯接口和多种时钟选择。SPMC65X 系列芯片适用于通用工控场合、计算机外围控制和家电等。

ECMC653 采用 8 位 SPMC65 CPU 核，具有 928 字节的 RAM 和 16k 字节的 ROM。同时还集成了 1 个时基、1 个看门狗定时器、6 个 16 位定时/计数器和 9 通道的 ADC。为了降低整个仿真板的成本，该芯片还配有一个 OTP ROM 的串行可编程接口。此外，为了帮助用户加快程序的调试，并发现程序中隐藏的错误，该芯片内部专门有一 RAM 区域用于记录程序最近一段时间执行的指令，用户可以从中了解到程序是否正确执行。

1.2 特性

下面基于 ECMC653 来介绍 SPMC65x 系列芯片的特性.

存储空间

- 16K 字节的 RAM 来模拟程序空间 ROM
- 928 字节的数据空间 RAM

SPMC65 CPU 核

- 支持 182 条指令
- CPU 最高频率 8MHz
- 支持位操作指令

I/O 端口

- 具有复用功能的双向 IO 口
- 可设置为上拉/下拉输入口，或者输出口

中断

- 外部中断（6 通道）：非屏蔽中断 NMI 或可屏蔽中断 IRQ
- 内部中断

复位

- 增强的复位系统

时钟（Clock）

- 3 种时钟源：RC 振荡器、晶体或陶瓷振荡器、外部时钟输入。
- 具备时钟频率输出功能

省电模式

- 2 种省电模式：Stop、Halt

外围模拟电路

- 9 通道 10 位 AD 转换器（采用内部参考电压），或 8 通道带一个外部参考电压的 10 位 AD 转换器。
- 4.0V 和 2.5V 可选的低电压复位系统
- 1 通道 10 位 DA 转换器，最大输出电流为 3.3mA
- 2 个电压比较器（30mv 滞环）

3 个 16 位定时/计数器(type I)

- 定时、计数功能
- 捕获功能(8 位脉宽/周期测量，16 位脉宽测量)
- 16 位比较输出
- 8 位 PWM 输出

2 个 16 位定时/计数器(type II)

- 定时、计数功能
- 捕获功能(8 位脉宽/周期测量，16 位脉宽测量)
- 16 位比较输出
- 12 位 PWM 输出

1 个 16 位定时/计数器(type III)

- 定时、计数功能
- 捕获功能(16 位脉宽/周期测量)
- 16 位比较输出
- 16 位 PWM 输出

时基

- 频率选择：1Hz ~ 62.5kHz @8MHz

峰鸣器输出

- 频率：1kHz to 2MHz @8MHz

可编程看门狗定时器

串行总线接口

- SPI 总线
- UART 总线
- IIC 总线

1.3 应用范围

小家电（电饭煲、电磁炉、微波炉、冰箱、洗衣机、空调、充电器）…
带有 IO 控制和 AD 转换功能的 MCU 应用的工控场合
工业设备控制
计算机外围控制

1.4 设计参考

无

1.5 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了丰富的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题	**应用例系列号**
编程方法（Coding method）	AN_O0328.DOC

库函数

标题	**应用例系列号**
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

1.6 修订记录

版本号	**日期**	**备注**
V0.1	03/31/2005	第一版

2 结构总述

2.1 简述

SPMC65X 芯片架构基于冯·诺伊曼模型：程序空间（ROM）和数据空间（RAM）共享总线。其结构分为内核、外围和特殊功能三部分。其中内核是芯片的基本部分，例如 CPU、存储器等；外围包括定时/计数器、捕获器、脉宽调制等控制；特殊功能可以帮助系统提高可靠性、降低功耗，例如低电压复位、省电模式等。SPMC65X 芯片结构概览如图 2-1 所示。

2.2 芯片结构概览

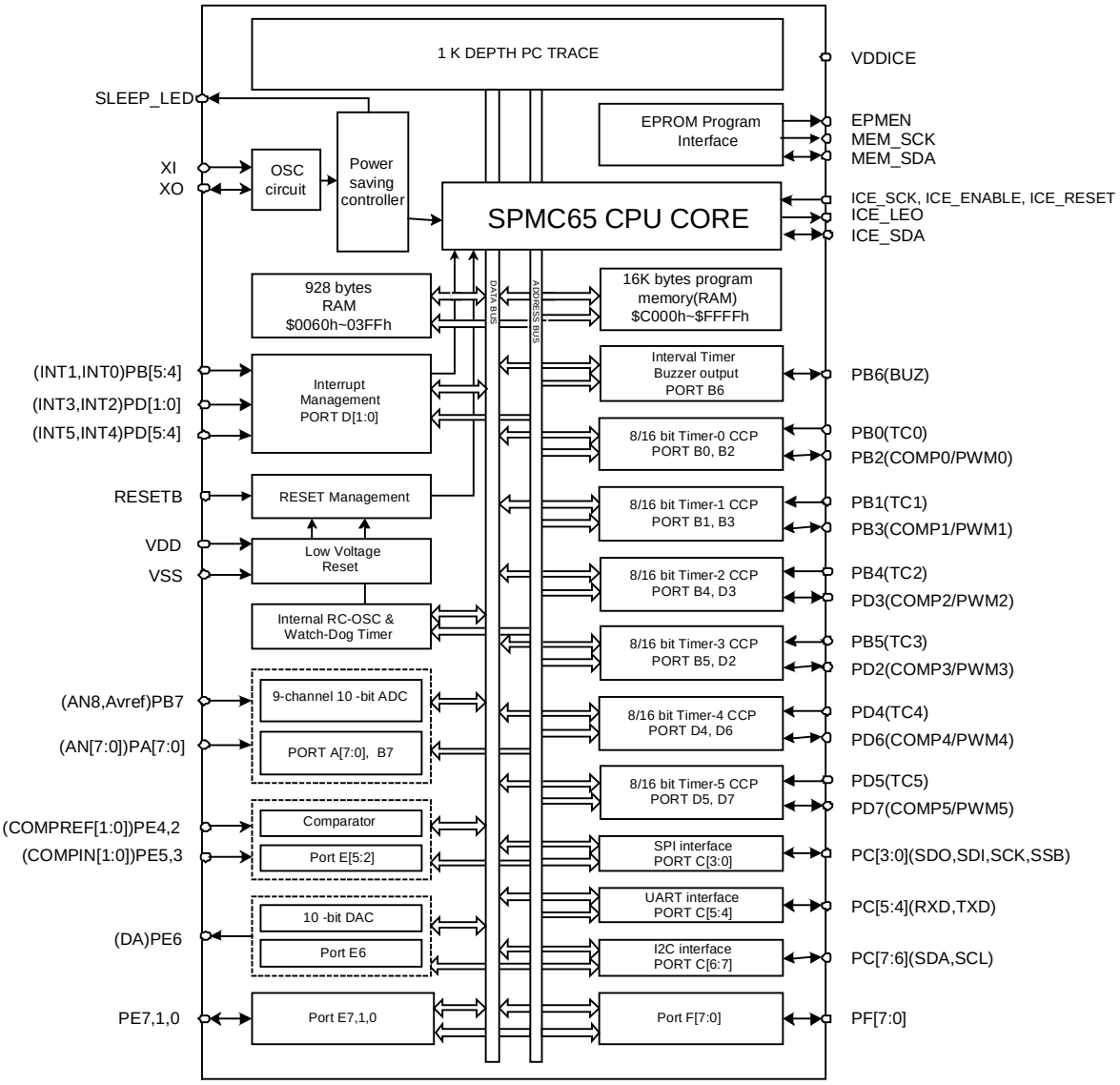
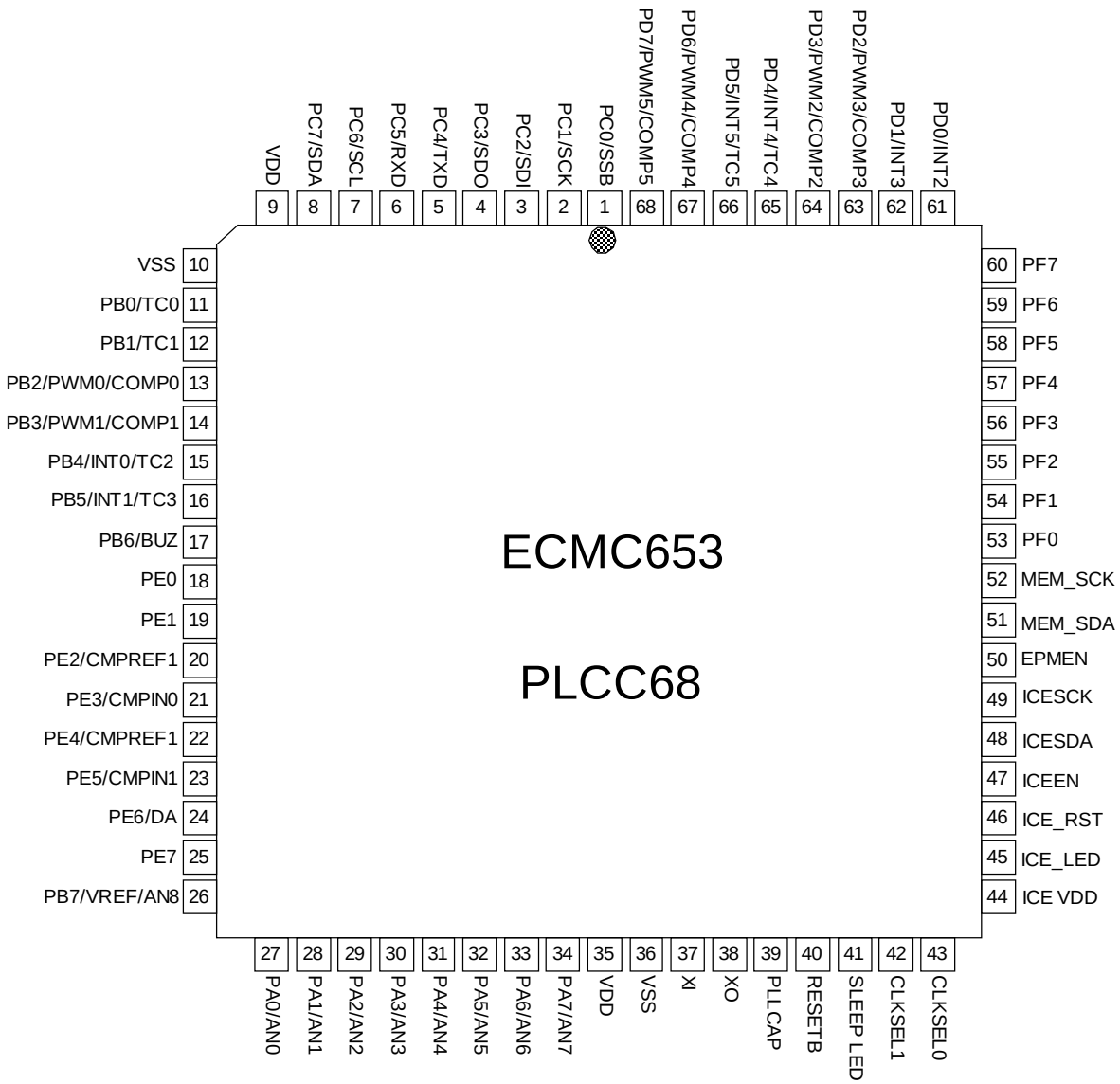


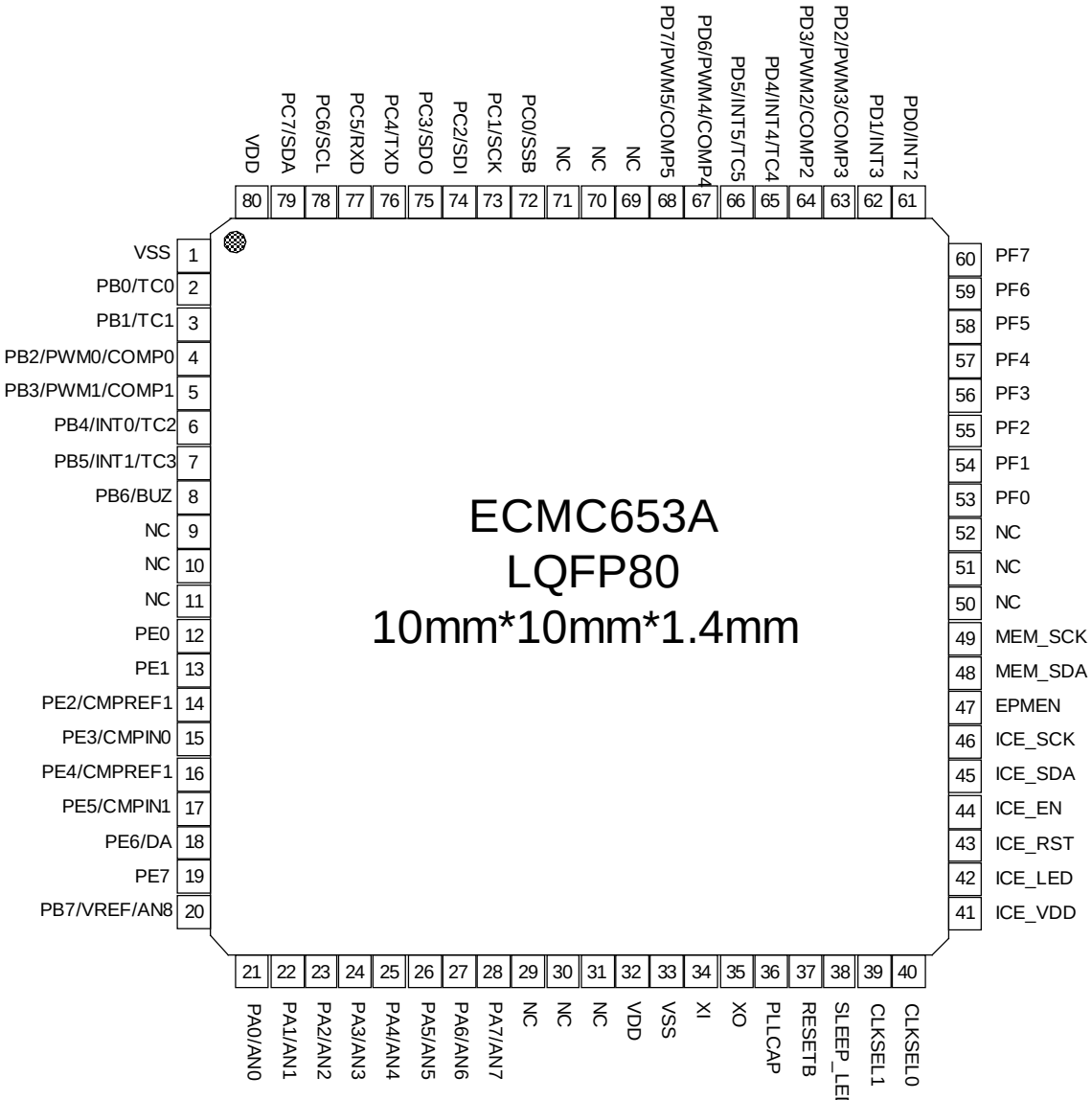
图 2-1 SPMC65X 原理框图

2.3 管脚分配

2.3.1 PLCC68



2.3.2 LQFP80



2.4 管脚描述

类型：I = 输入， O = 输出， S = 电源

输入电平：A = 专用的模拟电压输入

输出电平：HS = 20 毫安的高吸入电流

输入/输出电平：C = CMOS 0.3VDD/0.7VDD

端口控制配置：

— 输入：float = 悬浮，WPD = 下拉，WPU = 上拉，Int = 中断，ana = 模拟

— 输出：OD = 开漏输出（open drain）, PP =推挽

2.4.1 PLCC68

管脚号	管脚名称	类型	电平		端口/控制							主要功能（复位后）	功能
					输入					输出			
			输入	输出	Float	WPD	WPU	Int	ana	OD	PP		
9,35	VDD	S										主电源供电	
10,36	VSS	S										接地	
37	XI/R	I										*晶体/陶瓷振荡器输入端、RC 振荡器输入端或外部时钟输入端：*在 RC 振荡模式下，外接一个上拉电阻，可以和芯片内部的 OSC 电路产生内部时钟，作为 CPU 的时钟信号； 或者在晶体/陶瓷振荡器模式下，直接和外部晶体/陶瓷振荡器的一个管脚连接，由晶体和芯片内部电路共同产生时钟信号	

管脚号	管脚名称	类型	电平		端口/控制							主要功能（复位后）	功能
					输入					输出			
			输入	输出	Float	WPD	WPU	Int	ana	OD	PP		
38	XO	O										*振荡器频率输出端：*在晶体/陶瓷振荡器振荡器模式下，直接和外部晶体/陶瓷振荡器的另一个管脚连接，由晶体和芯片内部电路共同产生时钟信号； 也可以做为 RC 振荡器的频率输出管脚	
39	NC												
34	PA7/AN7	I/O	C					X		X	X	Port A7	ADC 模拟电压输入
33	PA6/AN6	I/O	C					X		X	X	Port A6	ADC 模拟电压输入
32	PA5/AN5	I/O	C					X		X	X	Port A5	ADC 模拟电压输入
31	PA4/AN4	I/O	C					X		X	X	Port A4	ADC 模拟电压输入

管脚号	管脚名称	类型	电平		端口/控制							主要功能（复位后）	功能
			输入	输出	输入					输出			
					Float	WPD	WPU	Int	ana	OD	PP		
30	PA3/AN3	I/O	C					X		X	X	Port A3	ADC 模拟电压输入
29	PA2/AN2	I/O	C					X		X	X	Port A2	ADC 模拟电压输入
28	PA1/AN1	I/O	C					X		X	X	Port A1	ADC 模拟电压输入
27	PA0/AN0	I/O	C					X		X	X	Port A0	ADC 模拟电压输入
26	PB7/VREF/AN8	I/O	C					X		X	X	Port B7	ADC 模拟电压输入或 ADC 外部参考电压输入
17	PB6/BUZ	I/O	C					X	X	X	X	Port B6	蜂鸣器输出

管脚号	管脚名称	类型	电平		端口/控制							主要功能（复位后）	功能
			输入	输出	输入					输出			
					Float	WPD	WPU	Int	ana	OD	PP		
16	PB5/INT1/TC3	I/O	C					INT1	X	X	X	Port B5	外部中断 1 输入/定时/计数器 3 的捕获输入/用定时/计数器 3 计数的外部时钟输入
15	PB4/INT0/TC2	I/O	C					INT0	X	X	X	Port B4	外部中断 0 输入/定时/计数器 2 的捕获事件输入/用定时/计数器 2 计数的外部时钟输入
14	PB3/PWM1/COMP1	I/O	C					X	X	X	X	Port B3	定时/计数器 1 比较输出 / PWM 输出

管脚号	管脚名称	类型	电平		端口/控制							主要功能（复位后）	功能
			输入	输出	输入					输出			
					Float	WPD	WPU	Int	ana	OD	PP		
13	PB2/PWM0/COMP0	I/O	C					X	X	X	X	Port B2	定时/计数器 0 比较输出 / PWM0 输出
12	PB1/TC1	I/O	C					X	X	X	X	Port B1	定时/计数器 1 的捕获事件输入/用定时/计数器 1 计数的外部时钟输入
11	PB0/TC0	I/O	C					X	X	X	X	Port B0	定时/计数器 0 的捕获事件输入/用定时/计数器 0 计数的外部时钟输入
8	PC7/SDA	I/O	C					X	X	X	X	Port C7	IIC 数据线

管脚号	管脚名称	类型	电平		端口/控制							主要功能（复位后）	功能
			输入	输出	输入					输出			
					Float	WPD	WPU	Int	ana	OD	PP		
7	PC6/SCL	I/O	C					X	X	X	X	Port C6	IIC 时钟线
6	PC5/RXD	I/O	C					X	X	X	X	Port C5	UART 信号接收端口
5	PC4/TXD	I/O	C					X	X	X	X	Port C4	UART 信号发送端口
4	PC3/SDO	I/O	C					X	X	X	X	Port C3	SPI 数据输出端口
3	PC2/SDI	I/O	C					X	X	X	X	Port C2	SPI 数据输入端口
2	PC1/SCK	I/O	C					X	X	X	X	Port C1	SPI 时钟输出/ 时钟输入
1	PC0/SSB	I/O	C					X	X	X	X	Port C0	SPI 片选

管脚号	管脚名称	类型	电平		端口/控制							主要功能（复位后）	功能
			输入	输出	输入					输出			
					Float	WPD	WPU	Int	ana	OD	PP		
68	PD7/PWM5/COMP5	I/O	C					X	X	X	X	Port D7	定时/计数器 5 比较输出 /PWM 5 输出
67	PD6/PWM4/COMP4	I/O	C					X	X	X	X	Port D6	定时/计数器 4 比较输出 /PWM4 输出
66	PD5/INT5/ TC5	I/O	C					INT5	X	X	X	Port D5	外部中断 5 输入/定时/计数器 5 的捕获事件输入/用定时/计数器 5 计数的外部时钟输入

管脚号	管脚名称	类型	电平		端口/控制							主要功能（复位后）	功能
			输入	输出	输入					输出			
					Float	WPD	WPU	Int	ana	OD	PP		
65	PD4/INT4/ TC4	I/O	C					INT4	X	X	X	Port D4	外部中断 4 输入/定时/计数器 4 的捕获事件输入/用定时/计数器 4 计数的外部时钟输入
64	PD3/PWM2/COMP2	I/O	C					X	X	X	X	Port D3	定时/计数器 2 比较输出/PWM 2 输出
63	PD2/PWM3/COMP3	I/O	C					X	X	X	X	Port D2	定时/计数器 3 比较输出 / PWM3 输出
62	PD1/INT3	I/O	C					INT3	X	X	X	Port D1	外部中断 3 输入

管脚号	管脚名称	类型	电平		端口/控制							主要功能（复位后）	功能
			输入	输出	输入					输出			
					Float	WPD	WPU	Int	ana	OD	PP		
61	PD0/INT2	I/O	C					INT2	X	X	X	Port D0	外部中断 2 输入
25	PE7	I/O	C					X	X	X	X	Port E7	
24	PE6/ DA	I/O	C					X	X	X	X	Port E6	DA 转换电流输出
23	PE5/ CMPIN1	I/O	C					X	X	X	X	Port E5	比较器 1 比较电压输入
22	PE4/ CMPREF1	I/O	C					X	X	X	X	Port E4	比较器 1 参考电压输入
21	PE3/CMPIN0	I/O	C					X	X	X	X	Port E3	比较器 0 比较电压输入
20	PE2/CMPREF0	I/O	C					X	X	X	X	Port E2	比较器 0 参考电压输入
19	PE1	I/O	C					X	X	X	X	Port E1	
18	PE0	I/O	C					X	X	X	X	Port E0	

管脚号	管脚名称	类型	电平		端口/控制							主要功能（复位后）	功能
					输入					输出			
			输入	输出	Float	WPD	WPU	Int	ana	OD	PP		
60	PF7	I/O	C					X	X	X	X	Port F7	
59	PF6	I/O	C					X	X	X	X	Port F6	
58	PF5	I/O	C					X	X	X	X	Port F5	
57	PF4	I/O	C					X	X	X	X	Port F4	
56	PF3	I/O	C					X	X	X	X	Port F3	
55	PF2	I/O	C					X	X	X	X	Port F2	
54	PF1	I/O	C					X	X	X	X	Port F1	
53	PF0	I/O	C					X	X	X	X	Port F0	
40	RESET	I/O	C		X	X						外部复位	
50	EPMEN	O										EPROM 模式输出管脚。系统开发人员用它打开 13 v /6v.	
52	MEM_SCK	I	C									访问外部 EPROM 时，时钟信号输入	

管脚号	管脚名称	类型	电平		端口/控制							主要功能（复位后）	功能
			输入	输出	输入					输出			
					Float	WPD	WPU	Int	ana	OD	PP		
51	MEM_SDA	I/O	C	C								访问外部 EPROM 时，数据串行输入/输出	
44	ICE_VDD	S										在线仿真电路 ICE 的接口电压，典型值为 3.3V。	
49	ICE_SCK	I										ICE 时钟输入.	
48	ICE_SDA	I/O										ICE 数据输入/输出	
47	ICE_EN	I										ICE 使能	
46	ICE_RST	I										ICE 复位	
45	ICE_LED	O										ICE 工作时该管脚可外接 LED 作为 指示	
41	SLEEP_LED	O										睡眠模式指示	
42	CLKSEL1	I										时钟选择 1	
43	CLKSEL0	I										时钟选择 0	

注意: CLKSEL[1:0]: 00:晶体和陶瓷振荡器

01:RC 振荡器

10:外部时钟输入

11:保留

总计：68 管脚。

2.4.2 LQFP80

Pin No	Pin Name	Type	Level		Port / Control							Main Function (after reset)	Alternate Function
					Input					Output			
			Input	Output	Float	WPD	WPU	Int	ana	OD	PP		
32,80	VDD	S										主电源供电	
1,33	VSS	S										接地	
34	XI/R	I										*晶体/陶瓷振荡器输入端、RC 振荡器输入端或外部时钟输入端：* 在 RC 振荡模式下，外接一个上拉电阻，可以和芯片内部的 OSC 电路产生内部时钟，作为 CPU 的时钟信号； 或者在晶体/陶瓷振荡器模式下，直接和外部晶体/陶瓷振荡器的一个管脚连接，由晶体/陶瓷振荡器和芯片内部电路共同产生时钟信号	
35	XO	O										*振荡器频率输出端：*在晶体/陶瓷振荡器模式下，直接和外部晶体/陶瓷振荡器的另一个管脚连接，由晶体/陶瓷振荡器和芯片内部电路共同产生时钟信号 也可以做为 RC 振荡器的频率输出管脚	
36	NC												
28	PA7/AN7	I/O	C					X		X	X	Port A7	ADC 模拟电压输入
27	PA6/AN6	I/O	C					X		X	X	Port A6	ADC 模拟电压输入
26	PA5/AN5	I/O	C					X		X	X	Port A5	ADC 模拟电压输入
25	PA4/AN4	I/O	C					X		X	X	Port A4	ADC 模拟电压输入

24	PA3/AN3	I/O	C				X		X	X	Port A3	ADC 模拟电压输入
23	PA2/AN2	I/O	C				X		X	X	Port A2	ADC 模拟电压输入
22	PA1/AN1	I/O	C				X		X	X	Port A1	ADC 模拟电压输入
21	PA0/AN0	I/O	C				X		X	X	Port A0	ADC 模拟电压输入
20	PB7/VREF/AN8	I/O	C				X		X	X	Port B7	ADC 模拟电压输入 或 ADC 外部参考电压输入
8	PB6/BUZ	I/O	C				X	X	X	X	Port B6	蜂鸣器输出
7	PB5/INT1/TC3	I/O	C				INT1	X	X	X	Port B5	外部中断 1 输入/定时/计数器 3 的捕获输入/用定时/计数器 3 计数的外部时钟输入
6	PB4/INT0/TC2	I/O	C				INT0	X	X	X	Port B4	外部中断 0 输入/定时/计数器 2 的捕获事件输入/用定时/计数器 2 计数的外部时钟输入
5	PB3/PWM1/COMP1	I/O	C				X	X	X	X	Port B3	定时/计数器 1 比较输出 / PWM1 输出
4	PB2/PWM0/COMP0	I/O	C				X	X	X	X	Port B2	定时/计数器 0 比较输出 / PWM0 输出
3	PB1/TC1	I/O	C				X	X	X	X	Port B1	定时/计数器 1 的捕获事件输入/用定时/计数器 1 计数的外部时钟输入
2	PB0/TC0	I/O	C				X	X	X	X	Port B0	定时/计数器 0 的捕获事件输入/用定时/计数器 0 计数的外部时钟输入

79	PC7/SDA	I/O	C				X	X	X	X	Port C7	IIC 总线数据输入/输出
78	PC6/SCL	I/O	C				X	X	X	X	Port C6	IIC 总线时钟输入
77	PC5/RXD	I/O	C				X	X	X	X	Port C5	UART 信号接收端口
76	PC4/TXD	I/O	C				X	X	X	X	Port C4	UART 信号发送端口
75	PC3/SDO	I/O	C				X	X	X	X	Port C3	SPI 数据输出端口
74	PC2/SDI	I/O	C				X	X	X	X	Port C2	SPI 数据输入端口
73	PC1/SCK	I/O	C				X	X	X	X	Port C1	SPI 时钟输出/时钟输入
72	PC0/SSB	I/O	C				X	X	X	X	Port C0	SPI 片选
68	PD7/PWM5/COMP5	I/O	C				X	X	X	X	Port D7	定时/计数器 5 比较输出/PWM5 输出
67	PD6/PWM4/COMP4	I/O	C				X	X	X	X	Port D6	定时/计数器 4 比较输出/PWM 输出
66	PD5/INT5/ TC5	I/O	C				INT5	X	X	X	Port D5	外部中断 5 输入/定时/计数器 5 的捕获事件输入/用定时/计数器 5 计数的外部时钟输入
65	PD4/INT4/ TC4	I/O	C				INT4	X	X	X	Port D4	外部中断 4 输入/定时/计数器 4 的捕获事件输入/用定时/计数器 4 计数的外部时钟输入
64	PD3/PWM2/COMP2	I/O	C				X	X	X	X	Port D3	定时/计数器 2 比较输出/PWM2 输出
63	PD2/PWM3/COMP3	I/O	C				X	X	X	X	Port D2	定时/计数器 3 比较输出 / PWM3 输出

62	PD1/INT3	I/O	C				INT3	X	X	X	Port D1	外部中断 3 输入
61	PD0/INT2	I/O	C				INT2	X	X	X	Port D0	外部中断 2 输入
19	PE7	I/O	C				X	X	X	X	Port E7	
18	PE6/ DA	I/O	C				X	X	X	X	Port E6	D/A 输出
17	PE5/ CMPIN1	I/O	C				X	X	X	X	Port E5	比较器 1 比较电压输入
16	PE4/ CMPREF1	I/O	C				X	X	X	X	Port E4	比较器 1 参考电压输入
15	PE3/CMPIN0	I/O	C				X	X	X	X	Port E3	比较器 0 比较电压输入
14	PE2/CMPREF0	I/O	C				X	X	X	X	Port E2	比较器 0 参考电压输入
13	PE1	I/O	C				X	X	X	X	Port E1	
12	PE0	I/O	C				X	X	X	X	Port E0	
60	PF7	I/O	C				X	X	X	X	Port F7	
59	PF6	I/O	C				X	X	X	X	Port F6	
58	PF5	I/O	C				X	X	X	X	Port F5	
57	PF4	I/O	C				X	X	X	X	Port F4	
56	PF3	I/O	C				X	X	X	X	Port F3	
55	PF2	I/O	C				X	X	X	X	Port F2	
54	PF1	I/O	C				X	X	X	X	Port F1	
53	PF0	I/O	C				X	X	X	X	Port F0	
37	RESET	I/O	C		X	X					外部复位	
47	EPMEN	O									EPROM 模式输出管脚。系统开发人员用它打开 13V/6V.	

49	MEM_SCK	I	C									访问外部 EPROM 时，时钟信号输入
48	MEM_SDA	I/O	C	C								访问外部 EPROM 时，数据串行输入/输出
41	ICE_VDD	S										在线仿真电路 ICE 的接口电压，典型值为 3.3V。
46	ICE_SCK	I										ICE 时钟输入.
45	ICE_SDA	I/O										ICE 数据输入/输出
44	ICE_EN	I										ICE 使能
43	ICE_RST	I										ICE 复位
42	ICE_LED	O										ICE 工作时该管脚可外接 LED 作为指示
38	SLEEP_LED	O										睡眠模式指示
39	CLKSEL1	I										时钟选择 1
40	CLKSEL0	I										时钟选择 0

注意: CLKSEL[1:0]: 00:晶体或者陶瓷振荡器

01:RC 振荡器

10:外部时钟输入

11:保留

总计：80 管脚。

2.5 设计参考

无

2.6 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题 **应用例系列号**

本章无相关应用例

库函数

标题	应用例系列号
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

2.7 Revision History

版本号	日期	备注
V0.1	03/31/2005	第一版

3 CPU

3.1 简述

SPMC65X 系列芯片采用高性能的微处理，全静态 CMOS 工艺设计，它有 6 个内部寄存器：累加器（A）、程序指针（PC）、X 寄存器、Y 寄存器、堆栈指针（SP）和状态寄存器（P）。支持 182 条指令。系统最高运行时钟（F_{SYS}）可以达到 8MHz。CPU 内部结构见图 3-1 所示。

3.2 CPU 内部结构图

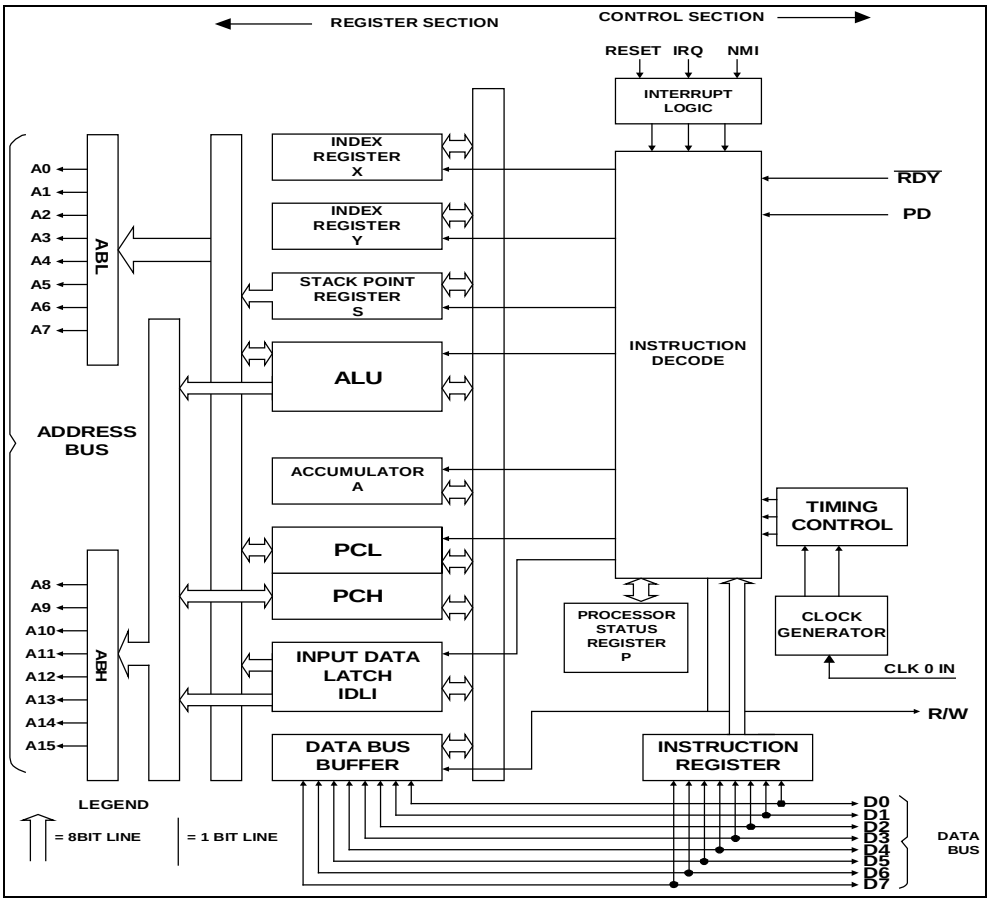


图 3-1 CPU 内部结构图

3.3 CPU 寄存器

SPMC65X 系列芯片有 6 个寄存器：程序指针（PC）、累加器(A)、X 寄存器、Y 寄存器，堆栈指针(SP)、状态寄存器（P）。其中程序指针（PC）是一个 16 位寄存器，其它都是 8 位寄存器，见下表。

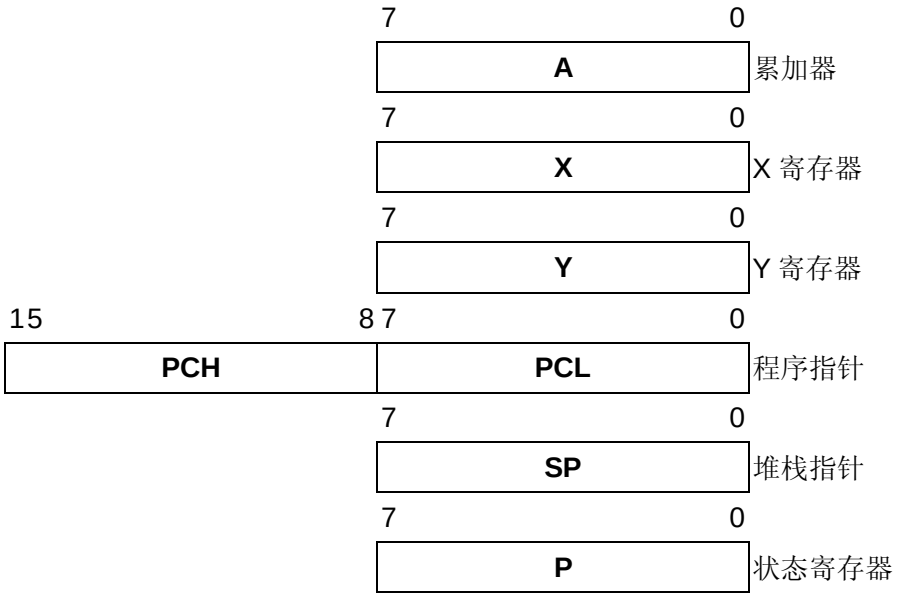


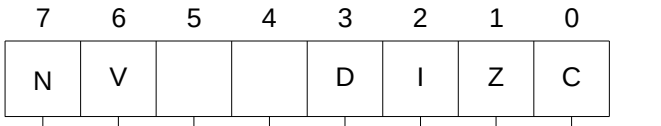
表 1 CPU 寄存器

CPU 寄存器详细描述见下表.

表 2 CPU 寄存器

寄存器	大小	描述
累加器 (A)	8 Bit	累加器是唯一的可以用于算术或逻辑操作的寄存器，如加、减、与、或、异或运算，并且可以存储计算结果。
X 寄存器	8 Bit	X 寄存器可以用作数据存储、偏移量或者计数器
Y 寄存器	8 Bit	Y 寄存器可以用作数据存储、偏移量或者计数器
程序指针 (PC)	16 Bit	程序计数器(PC)是一个 16 位寄存器，它指向 CPU 即将执行的下一条指令的地址 当 CPU 执行完一条指令，PC 的值加 1，指向下一条指令的地址，如果发生指令跳转，则 PC 指向跳转地址。
堆栈指针 (SP)	8 Bit	堆栈指针(SP)是一个 8 位寄存器，通常它用于指示堆栈中即将被操作的字节的地址，也可以表明当前堆栈的使用情况。(相关指令 Push / Pop)
状态寄存器 (P)	8 Bit	状态寄存器可以提供 CPU 执行完上一条指令后的状态信息。

3.4 状态寄存器



进位标志：
有进位，C=1；无进位，C＝0

零标志：
数学或逻辑运算结果为 0，Z=1；
结果为 1，Z＝0

中断屏蔽标志：
I=1，CPU 忽略中断请求信号（非屏蔽中断源 NMI 除外）；
I＝0，CPU 接收中断请求信号

十进制模式标志：
D=1，当前运算模式为十进制；
D=0，当前运算模式为二进制

保留

保留

溢出标志：
算术运算过程中发生溢出，V=1；否则，V=0

负标志：
算术或逻辑运算结果为负，N=1；否则，N＝0

注意：并非所有的指令执行都会影响状态寄存器的值。在 25 章中将对每个指令作详细描述。

X、Y 寄存器：在寻址模式下，X、Y 寄存器用作地址索引指针，可以很方便的进行数据的存取。同时，X、Y 寄存器还可以进行加 1、减 1、比较和数据传送的操作。

累加器（A）：累加器是一个 8 位通用寄存器，可用于数据传送、数据暂存和条件判断等操作。

堆栈指针（SP）：它是一个 8 位寄存器，用于指示堆栈中即将被操作的字节的地址，也可以表明当前堆栈的使用情况。

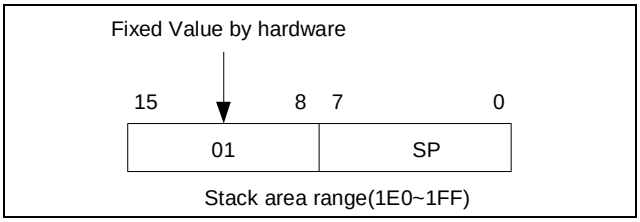


图 3-2 堆栈指针寄存器

【例 3 1】：堆栈指针初始化

```
ldx     #C_STACK_BOTTOM     ;堆栈指针初始值为$FF
txs
```

当调用子程序或中断发生时，堆栈指针会自动更新。但是如果堆栈指针的值超出了其允许的堆栈空间（$1E0~$1FF），系统就会发生非法地址复位。

程序计数器(PC)：它一个 16 位寄存器，存放 CPU 即将执行的下一条指令的 16 位地址。该 16 位地址的高 8 位和低 8 位分别存放在寄存器 PCH 和 PCL 中。复位后，PC 中存储的是程序运行的起始地址$FFFCH。

状态寄存器(P)：它是一个 8 位寄存器，包括一个中断屏蔽位和 6 个独立的状态标志位，这 6 个状态标志位可以反映程序执行完上一条指令后的状态信息。该 8 位寄存器也可以由指令 PHP 和 PLP 来进行保存和取出。同时，也有相应的特殊指令来对这个寄存器进行位操作，详细说明如下。

负标志位（N）：该位指示一个数据或者运算结果的 bit7 的状态。用户可以通过该位进行条件跳转。

溢出标志位(V)：该位指示在算术运算过程是是否发生溢出。如果加法运算结果大于 127 或者减法运算结果小于-128，该位置 1。

十进制模式标志位(D)：在 SPMC65 系列芯片中有两种算术运算模式：二进制模式和十进制

模式，该位标明了当前的运算模式。用户可以通过相应的指令来选择一种运算模式。

中断屏蔽位(I)：该位用于使能/禁止除“非屏蔽中断源（NMI）”以外的所有中断源。将该位置 1，CPU 将忽略中断请求，置 0，CPU 将接受中断请求。

零标志位（Z）：数据和算术运算结果标志位。当数据或算术运算结果为 0 时，该位被置 1；为其它值时，该位被置 0。

进位标志（C）
当加法操作中产生进位或减法操作中没有借位时，该位被置 1。此外，移位或循环指令也会改变该位的值。

3.5 寻址方式

SPMC65X 系列芯片支持 11 种寻址方式：

立即数寻址
绝对寻址
绝对变址寻址
零页寻址
零页变址寻址
隐含寻址
累加器寻址
变址间接寻址
间接跳转寻址
间接变址寻址
相对寻址

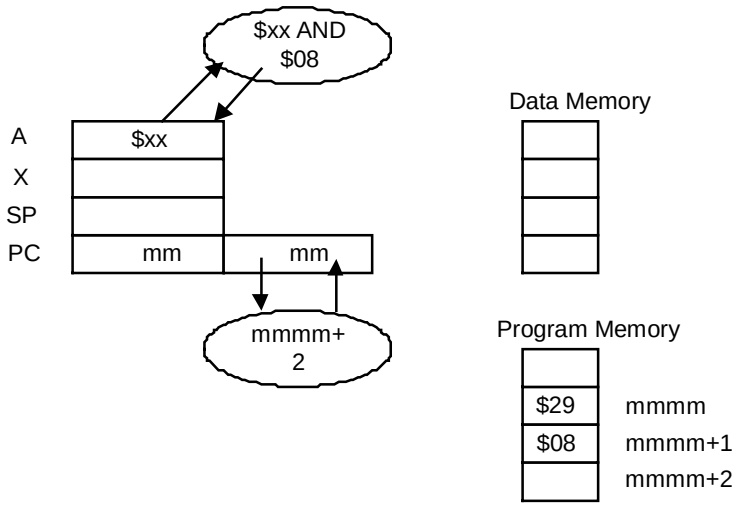
3.5.1 立即数寻址

立即数为 1 字节。

操作：　操作码　#dd

【例 3 2】:立即数寻址

```
AND     #$08
```



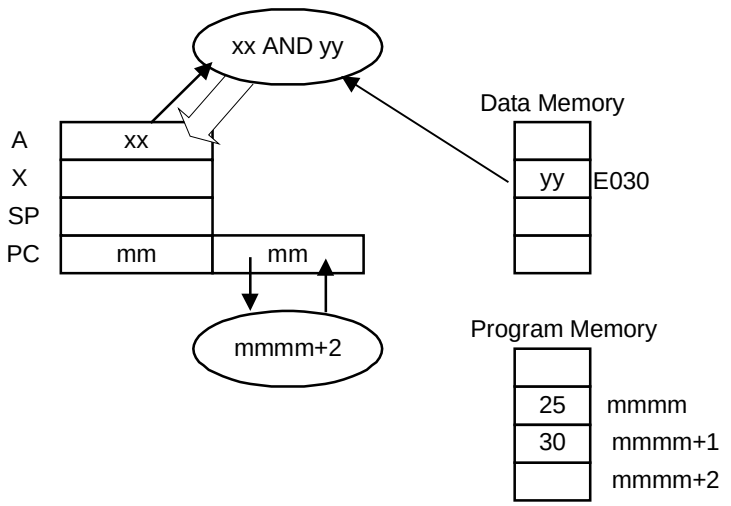
3.5.2 绝对寻址

绝对寻址用两个字节(adr 16)来指明目标操作数地址。这两个字节(adr 16)可以是一个数据的地址，也可以是即将执行的下一条指令的起始地址。

操作： 操作码 Adr16

【例 3 3】: 绝对寻址

```
AND     $E030
```



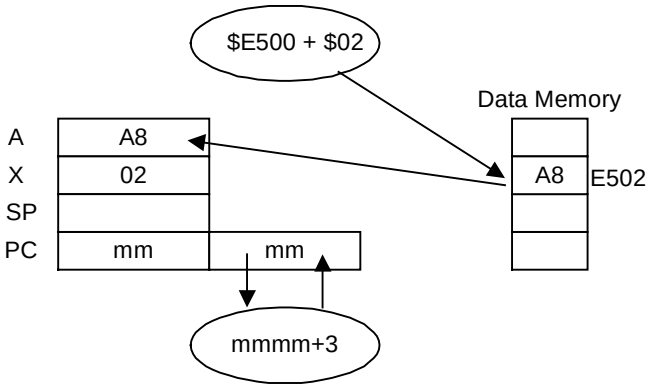
3.5.3 绝对变址寻址

绝对变址寻址使用一个基址（双字节）和一个指针（X）来指明目标操作数地址。

操作： 操作码 **Adr 16, X**

【例 3 4】：绝对变址寻址

```
LDA     $E502,X
```



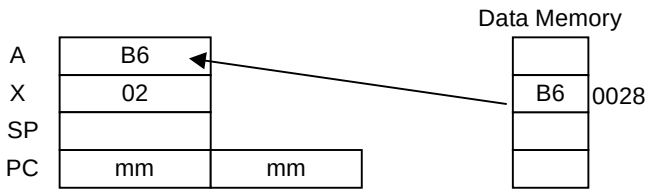
3.5.4 零页寻址

零页寻址用零页的低位地址来定义目标地址（高位地址默认为零）。

操作：　操作码　Adr 08

【例 3 5】：零页寻址

```
LDA     $28
```



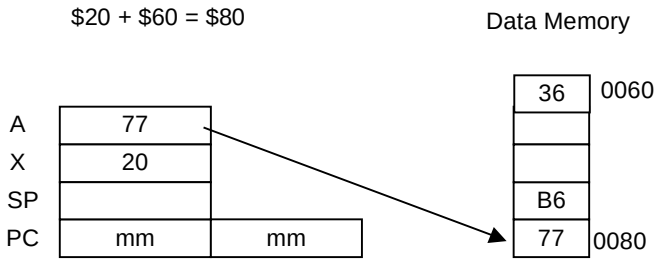
3.5.5 零页变址寻址

零页变址寻址用使用一个基址（零页的低位地址）和一个索引指针（X）来指明目标操作数地址。

操作：　操作码　Adr 08, X

【例 3 6】：零页变址寻址

```
LDX     #$20
LDA     #$77
STA     $60, X
```



3.5.6 隐含寻址

隐含是特殊指令对应的特殊寻址方式，源操作数和目标操作数的地址由指令默认。

操作： 操作码

【例 3 7】: 隐含寻址

```
TAX     ；将累加器内容传送到 X 寄存器中
CLC     ；清除进位标志
```

3.5.7 累加器寻址

累加器寻址中所有的数据操作均在累加器 A 中进行。

操作： 操作码

【例 3 8】: 累加器寻址

```
ROL     ；带进位标志(C)循环左移
ROR     ；带进位标志(C)循环右移
```

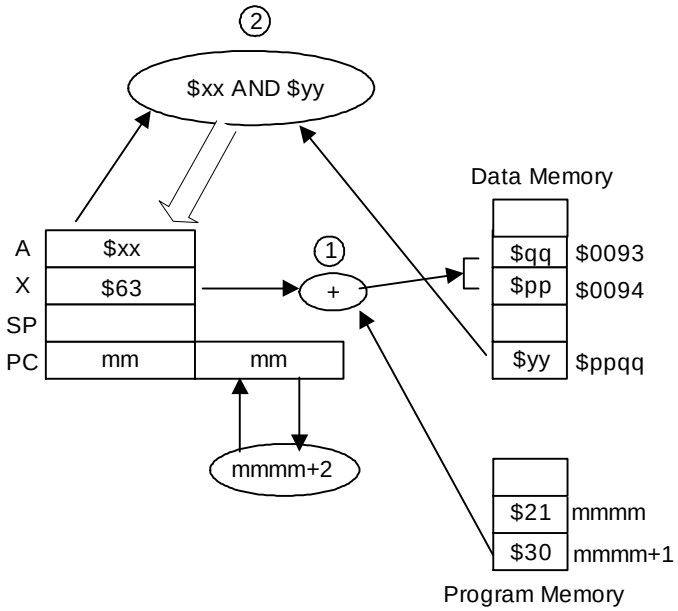
3.5.8 变址间接寻址

变址间接寻址采用（8 位地址＋索引指针 X）来指明目标操作数地址。，在此寻址模式下，只能采用 X 寄存器作为索引指针。该寻址方式仅在零页有效。

操作： 操作码 (Adr 08, X)

【例 3 9】: 变址间接寻址

```
AND     ($30, X)
```



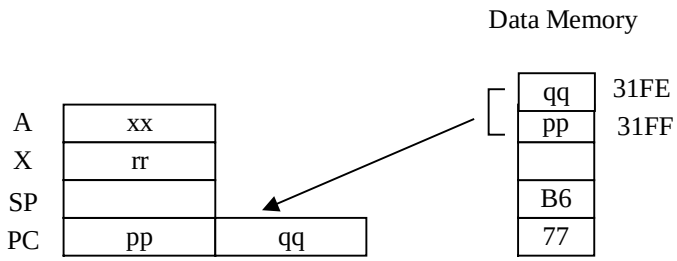
3.5.9 间接跳转寻址

间接跳转寻址只能用在 JMP 指令中。

操作： 操作码 JMP (Adr)

【例 3 10】: 间接跳转寻址

JMP ($31FE)



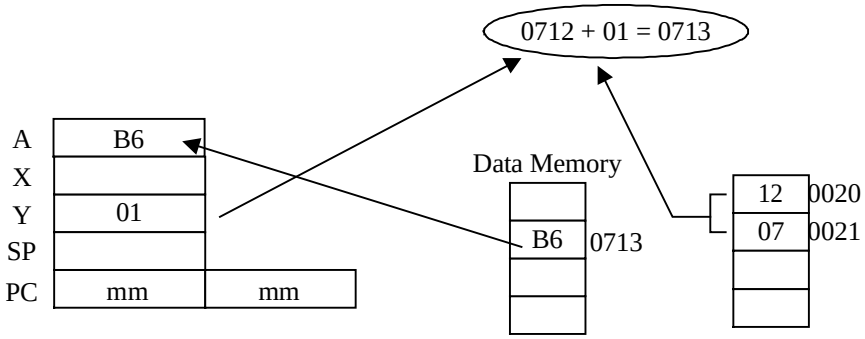
3.5.10 间接变址寻址

间接变址寻址只能用 Y 寄存器作为索引指针。

操作： 操作码 (aa), Y

【例 3 11】:间接变址寻址模式

LDA ($20), Y



3.5.11 相对寻址

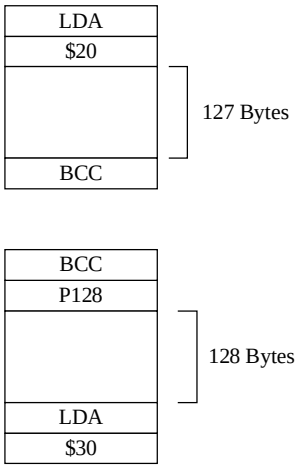
相对寻址使用一个 8 位地址来定义目标操作数的地址，该寻址方式仅用在分支跳转指令中，最大的向前跨度为 127 字节，向后跨度为 128 字节。

操作： 操作码 Adr 08

【例 3 12】：相对寻址模式

```
        BCC     M127
        BCC     P128
M127:   LDA     #$20
        .
        .
        .
        BCC     M127

        BCC     P128
        .
        .
        .
P128:   LDA     #$30
```



3.6 设计参考

无

3.7 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题	应用例系列号
本章无相关应用例	

库函数

标题	应用例系列号
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

3.8 修订记录

版本号	日期	备注
V0.1	03/31/2005	第一版

4 存储器

4.1 简述

SPMC65X 系列芯片有独立的程序存储区和数据存储区。程序存储区为只读，最大为 16K 字节。数据存储区可读可写，包含了各种控制寄存器和和用户 RAM。其中用户 RAM 最大为 928 字节（包括堆栈区）。SPMC65X 系列芯片集成的外围通过地址映像到存储区的 0~5FH 范围内。CPU 访问外围即在内存中寻访到相应的外围控制寄存器。注意不要访问有“保留”字样的内存区，否则可能导致非法地址复位。

4.2 存储器分配图

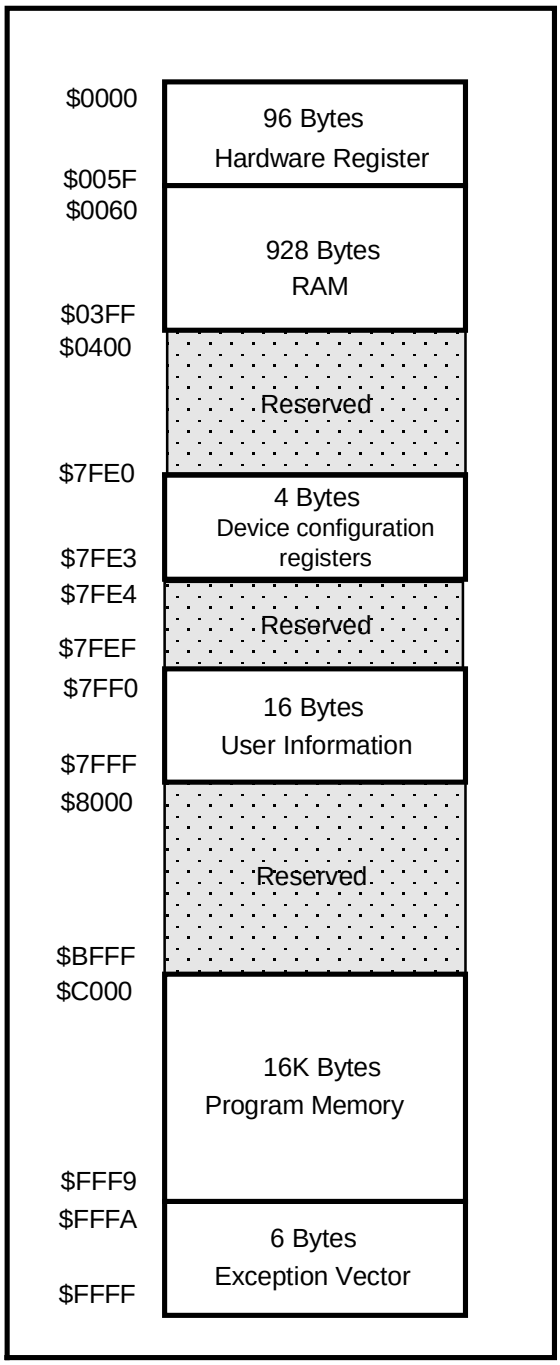


图 4-1 SPMC65X 存储器分配图

4.3 程序存储器

SPMC65X 系列的程序存储区最大为 16K 字节，所以使用一个 16 位的程序计数器（最大寻址 64K 字节）便可以访问所有的区域，它的内存映像见图 4-2。程序存储区由一次性可烧录型只读存储器 OTP ROM 组成，烧录时可以设置“保密位”对程序进行保护。当“保密位”设置为 0 时，用户无法读取这 16K 的 OTP ROM 中的内容，相反，置 1 时，程序可以从芯

片中读出。设置了保密后，应用选项和用户信息仍然是可读的。复位后，CPU 从地址$FFFC 和$FFFD 中读出复位后程序的入口地址，然后跳转到该程序的入口去执行。中断发生时，CPU 通过中断向量跳转到相应的中断服务子程序中继续执行。存储器的$FFFA~$FFFB 作为非屏蔽中断 NMI 的向量地址，$FFFE~$FFFF 作为可屏蔽中断 IRQ 的向量地址。这些中断向量的地址应在用户程序中做出正确的定义。

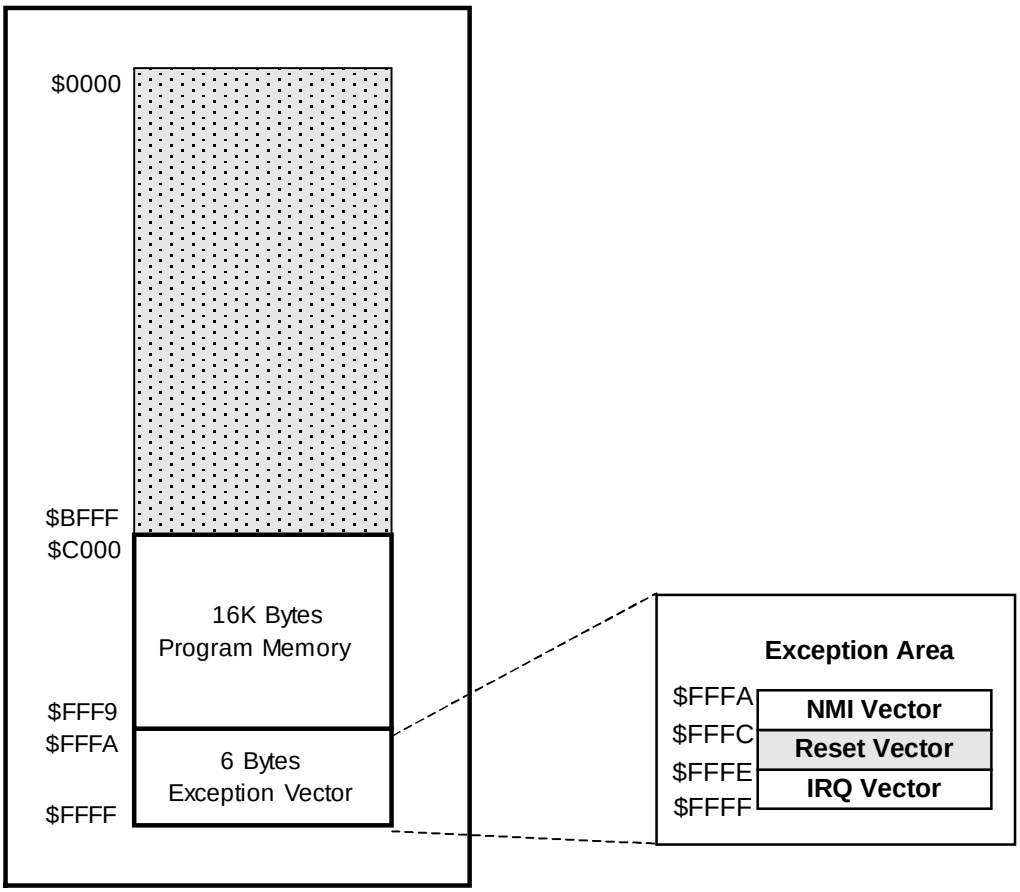


图 4-2 SPMC65X 内存映像

【例 4 1】：程序在 ROM 区对中断向量表的定义

```
VECTOR:     .SECTION
    DW      V_NMI
    DW      V_Reset
    DW      V_IRQ
```

4.4 数据存储区

数据存储区如图 4-3 所示分为 4 部分：控制寄存器、用户 RAM(包含堆栈空间)、芯片配置寄存器和用户信息。$0~$5F 空间作为控制寄存器，$0060~$03FF 作为用户使用的 RAM 空间。堆栈区为$01E0~$01FF，生长方向从栈底$01FF 开始，不断递减，当堆栈指针超过了$01E0

时，CPU 复位。$7FE0～$7FE3 这 4 个字节为芯片配置寄存器，用户可以设置一些特殊功能。SPMC65X 系列预留了$7FF0～$7FFF 区域内的用户信息块，可以由用户写入序列号或版本控制信息等。

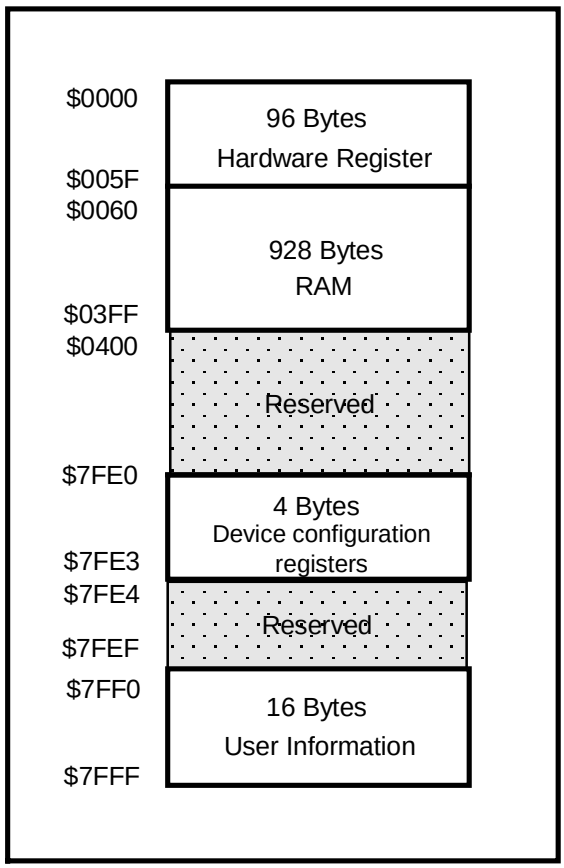


图 4-3 SPMC65X 数据存储区

4.4.1 硬件控制寄存器

CPU 通过设置硬件控制寄存器来控制其外围的工作。硬件控制寄存器位于存储器的 0000h ~005fh 空间内，共有 96 字节。它的作用是对中断系统、定时/计数器、AD 转换器和 IO 端口进行设置和状态标识。概要介绍见 4.1.1 节，详细介绍请见各章节。

4.4.1.1 概述

地址	$0000	$0001	$0002	$0003	$0004	$0005	$0006	$0007	$0008	$0009	$000A	$000B	$000C	$000D	$000E	$000F
	端口 A~D 数据&设置												中断设置&标志			
地址	$0010	$0011	$0012	$0013	$0014	$0015	$0016	$0017	$0018	$0019	$001A	$001B	$001C	$001D	$001E	$001F

	WTD 清零	定时/计数器/PWM/捕获 0~1 设置&数据							定时/计数器/PWM/捕获 2~3 设置&数据							
地址	$0020	$0021	$0022	$0023	$0024	$0025	$0026	$0027	$0028	$0029	$002A	$002B	$002C	$002D	$002E	$002F
	定时/计数器/PWM/捕获 4~5 设置&数据						中断设置&标志		ADC 设置&转换结果					蜂鸣器	比较器 0~1 设置	
地址	$0030	$0031	$0032	$0033	$0034	$0035	$0036	$0037	$0038	$0039	$003A	$003B	$003C	$003D	$003E	$003F
	系统设置&看门狗设置								SPI 设置&数据							
地址	$0040	$0041	$0042	$0043	$0044	$0045	$0046	$0047	$0048	$0049	$004A	$004B	$004C	$004D	$004E	$004F
	I O 端口 E~F 数据&设置						UART 设置&数据				IIC 总线设置&数据					
地址	$0050	$0051	$0052	$0053	$0054	$0055	$0056	$0057	$0058	$0059	$005A	$005B	$005C	$005D	$005E	$005F
						DAC 设置&数据			Cap Ctrl	端口 A~F 数据锁存缓冲器						定时/计数器 5 数据

4.4.1.2 控制寄存器列表

$0000~000F:

地址	功能	上电复位值	复位值	R/W	Bit 7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
$0000	端口 A 数据寄存器 (P_IOA_Data)		00h	R	端口 A 的管脚状态							
				W	端口 A 的输出数据(写入$0059)							
$0001	端口 B 数据寄存器 (P_IOB_Data)		00h	R	端口 B 的管脚状态							
				W	端口 B 的输出数据(写入$005A)							

地址	功能	上电复位值	复位值	R/W	Bit 7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
$0002	端口 C 数据寄存器 (P_IOC_Data)		00h	R	端口 C 的管脚状态							
				W	端口 C 的输出数据 (写入$005B)							
$0003	端口 D 数据寄存器 (P_IOD_Data)		00h	R	端口 D 的管脚状态							
				W	端口 D 的输出数据 (写入$005C)							
$0004	端口 A 方向寄存器 (P_IOA_Dir)		00h	R	0＝输入 1＝输出							
				W	0＝输入 1＝输出							
$0005	端口 B 方向寄存器 (P_IOB_Dir)		00h	R	0＝输入 1＝输出							
				W	0＝输入 1＝输出							
$0006	端口 C 方向寄存器 (P_IOC_Dir)		00h	R	0＝输入 1＝输出							
				W	0＝输入 1＝输出							
$0007	端口 D 方向寄存器 (P_IOD_Dir)		00h	R	0＝输入 1＝输出							
				W	0＝输入 1＝输出							
$0008	端口 A 属性寄存器 (P_IOA_Attrib)		00h	R	端口 A 属性							
				W	端口 A 属性							
$0009	端口 B 属性寄存器 (P_IOB_Attrib)		00h	R	端口 B 属性							
				W	端口 B 属性							
$000A	端口 C 属性寄存器 (P_IOC_Attrib)		00h	R	端口 C 属性							
				W	端口 C 属性							
$000B	端口 D 属性寄存器 (P_IOD_Attrib)		00h	R	端口 D 属性							
				W	端口 D 属性							
$000C	中断标志 0 (P_INT_Flag0)		00h	R	ADIF	WDIF	IRQ5IF/CAP5IF	IRQ4IF/CAP4IF	IRQ3IF	IRQ2IF	IRQ1IF/CAP3IF	IRQ0IF/CAP2IF
				W	写“1”清除相应中断标志位							

地址	功能	上电复位值	复位值	R/W	Bit 7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
$000D	中断控制 0 (P_INT_Ctrl0)		00h	R	ADIE	WDIE	IRQ5IE/CAP5IE	IRQ4IE/CAP4IE	IRQ3IE	IRQ2IE	IRQ1IE/CAP3IE	IRQ0IE/CAP2IE
				W	ADIE	WDIE	IRQ5IE/CAP5IE	IRQ4IE/CAP4IE	IRQ3IE	IRQ2IE	IRQ1IE/CAP3IE	IRQ0IE/CAP2IE
$000E	中断标志 1 (P_INT_Flag1)		00h	R	CAP1IF	CAP0IF	T5OIF	T4OIF	T3OIF	T2OIF	T1OIF	T0OIF
				W	写“1”清除相应中断标志位							
$000F	中断控制 1 (P_INT_Ctrl1)		00h	R	CAP1IE	CAP0IE	T5OIE	T4OIE	T3OIE	T2OIE	T1OIE	T0OIE
				W	CAP1IE	CAP0IE	T5OIE	T4OIE	T3OIE	T2OIE	T1OIE	T0OIE

$0011~$001F：Timer/PWM 设置&数据寄存器

地址	功能	上电复位值	复位值	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
$0010	看门狗清除 (P_WDT_Clr)		00h	R	0	0	0	0	0	0	0	0
				W	写 #55h 清除看门狗(C_WDT_Clr = $55)							
$0011	定时/计数器 0-1 控制寄存器 0		00h	R	0	定时/计数器 1 功能选择			0	定时/计数器 0 功能选择		

地址	功能	上电复位值	复位值	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
	(P_TMR0_1_Ctrl0)			W	-	定时/计数器 1 功能选择			-	定时/计数器 0 功能选择		
$0012	定时/计数器 0-1 控制寄存器 1 (P_TMR0_1_Ctrl1)		00h	R	0	定时/计数器 1 预分频选择			0	定时/计数器 0 预分频选择		
				W	-	定时/计数器 1 预分频选择			-	定时/计数器 0 预分频选择		
$0013	定时/计数器 0 计数值的低字节 (P_TMR0_Count)		00h	R	8 位或 16 位（低字节）定时/计数器 0 的计数值							
	定时/计数器 0 重载寄存器 (P_TMR0_Preload)			W	8 位或 16 位（低字节）定时/计数器 0 的重载计数值							
	比较器 0 计数值的低字节 (P_TMR0_Comp)		00h	R	8 位或 16 位（低字节）比较器 0 的计数值							
				W	8 位或 16 位（低字节）比较器 0 的比较值							
	捕获器 0 捕获值的低字节 (P_TMR0_Cap)		00h	R	8 位或 16 位（低字节）捕获器 0 的捕获脉宽值							
				W	8 位或 16 位（低字节）捕获器 0 的重载值							
	PWM0 周期值 (P_TMR0_PWMPeriod)		00h	R	8 位 PWM0 的周期							
				W	8 位 PWM0 的周期							
$0014	定时/计数器 0 计数值的高字节 (P_TMR0_CountHi)		00h	R	16 位定时/计数器 0 计数值的高字节							
	定时/计数器 0 重载寄存器 (P_TMR0_PreloadHi)			W	16 位定时/计数器 0 重载计数值的高字节							
	比较器 0 计数值的高字节 (P_TMR0_CompHi)		00h	R	16 位比较器 0 计数值的高字节							
				W	16 位比较器 0 比较值的高字节							
	捕获器 0 捕获值的高字节		00h	R	16 位捕获器 0 捕获脉宽值的高字节							

地址	功能	上电复位值	复位值	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
	(P_TMR0_CapHi)			W	16 位捕获器 0 重载值的高字节							
	捕获器 0 周期值 (P_TMR0_CapCycle8)		00h	R	8 位捕获器 0 捕获的周期值							
				W	8 位捕获器 0 的重载值							
	PWM0 占空比值 (P_TMR0_PWMDuty)		00h	R	8 位 PWM0 的占空比值							
				W	8 位 PWM0 的占空比值							
$0015	定时/计数器 1 计数值的低字节 (P_TMR1_Count)		00h	R	8 位或 16 位（低字节）定时/计数器 1 的计数值							
	定时/计数器 1 重载寄存器 (P_TMR1_Preload)			W	8 位或 16 位（低字节）定时/计数器 1 的重载计数值							
	比较器 1 计数值的低字节 (P_TMR1_Comp)		00h	R	8 位或 16 位（低字节）比较器 1 的计数值							
				W	8 位或 16 位（低字节）比较器 1 的比较值							
	捕获器 1 捕获值的低字节 (P_TMR1_Cap)		00h	R	8 位或 16 位（低字节）捕获器 1 的捕获脉宽值							
				W	8 位或 16 位（低字节）捕获器 1 的重载值							
	PWM1 低字节周期值 (P_TMR1_PWMPeriod)		00h	R	12 位 PWM 1 低字节的周期值							
				W	12 位 PWM 1 低字节的周期值							
$0016	定时/计数器 1 计数值的高字节 (P_TMR1_CountHi)		00h	R	16 位定时/计数器 1 计数值的高字节							
	定时/计数器 1 重载寄存器 (P_TMR1_PreloadHi)			W	16 位定时/计数器 1 重载计数值的高字节							
	比较器 1 计数值的高字节 (P_TMR1_CompHi)		00h	R	16 位比较器 1 计数值的高字节							
				W	16 位比较器 1 比较值的高字节							
	捕获器 1 捕获值的高字节		00h	R	16 位捕获器 1 捕获脉宽值的高字节							

地址	功能	上电复位值	复位值	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
	(P_TMR1_CapHi)			W	16 位捕获器 1 重载值的高字节							
	捕获器 1 周期值 (P_TMR1_CapCycle8)		00h	R	8 位捕获器 1 捕获的周期值							
				W	8 位捕获器 1 的重载值							
	PWM1 占空比和周期的高字节 (P_TMR1_DutyPeriod)		00h	R	12 位定时/计数器(高字节) PWM1 占空比值				12 位定时/计数器 1(高字节) PWM1 周期值			
				W	12 位定时/计数器(高字节) PWM1 占空比值				12 位定时/计数器 1(高字节) PWM1 周期值			
$0017	PWM1 低字节占空比 (P_TMR1_PWMDuty)		00h	R	12 位定时/计数器(低字节) PWM1 的占空比值							
				W	12 位定时/计数器(低字节)PWM1 的占空比值							
$0018	定时/计数器 2-3 控制器 0 (P_TMR2_3_Ctrl0)		00h	R	0	定时/计数器 3 功能选择			0	定时/计数器 2 功能选择		
				W	-	定时/计数器 3 功能选择			-	定时/计数器 2 功能选择		
$0019	定时/计数器 2-3 控制器 1 (P_TMR2_3_Ctrl1)		00h	R	0	定时/计数器 3 预分频选择			0	定时/计数器 2 预分频选择		
				W	-	定时/计数器 3 预分频选择			-	定时/计数器 2 预分频选择		
$001A	定时/计数器 2 计数值的低字节 (P_TMR2_Count)		00h	R	8 位或 16 位（低字节）定时/计数器 2 的计数值							
	定时/计数器 2 重载寄存器 (P_TMR2_Preload)			W	8 位或 16 位（低字节）定时/计数器 0 的重载计数值							
	比较器 2 计数值的低字节 (P_TMR2_Comp)		00h	R	8 位或 16 位（低字节）比较器 2 的计数值							
				W	8 位或 16 位（低字节）比较器 2 的比较值							
	捕获器 2 捕获值的低字节		00h	R	8 位或 16 位（低字节）捕获器 2 的捕获脉宽值							

地址	功能	上电复位值	复位值	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
	(P_TMR2_Cap)			W	8 位或 16 位（低字节）捕获器 2 的重载值							
	PWM2 周期值 (P_TMR2_PWMPeriod)		00h	R	8 位 PWM2 的周期							
				W	8 位 PWM2 的周期							
$001B	定时/计数器 2 的高字节 (P_TMR2_CountHi)		00h	R	16 位定时/计数器 2 计数值的高字节							
				W	16 位定时/计数器 2 重载计数值的高字节							
	比较 2 计数值的高字节 (P_TMR2_CompHi)		00h	R	16 位比较器 2 计数值的高字节							
				W	16 位比较器 2 比较值的高字节							
	捕获器 2 捕获值的高字节 (P_TMR2_CapHi)		00h	R	16 位捕获器 2 捕获脉宽值的高字节							
				W	16 位捕获器 2 重载值的高字节							
	捕获器 2 周期值 (P_TMR2_CapCycle8)		00h	R	8 位捕获器 2 捕获的周期值							
				W	8 位捕获器 2 的重载值							
	PWM2 占空比 (P_TMR2_PWMDuty)		00h	R	PWM2 的占空比值							
				W	PWM2 的占空比值							
$001C	定时/计数器 3 计数值的低字节 (P_TMR3_Count)		00h	R	8 位或 16 位（低字节）定时/计数器 3 的计数值							
	定时/计数器 3 重载寄存器 (P_TMR3_Preload)			W	8 位或 16 位（低字节）定时/计数器 3 的重载计数值							
	比较器 3 计数值的低字节 (P_TMR3_Comp)		00h	R	8 位或 16 位（低字节）比较器 3 的计数值							
				W	8 位或 16 位（低字节）比较器 3 的比较值							
	捕获器 3 捕获值的低字节 (P_TMR3_Cap)		00h	R	8 位或 16 位（低字节）捕获器 3 的捕获脉宽值							
				W	8 位或 16 位（低字节）捕获器 3 的重载值							
	PWM3 周期值的低字节		00h	R	12 位 PWM3 周期值的低字节							

地址	功能	上电复位值	复位值	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
	(P_TMR3_PWMPeriod)			W	12 位 PWM3 周期值的低字节							
$001D	定时/计数器 3 计数值的高字节 (P_TMR3_CountHi)		00h	R	16 位定时/计数器 3 计数值的高字节							
	定时/计数器 3 重载寄存器 (P_TMR3_PreloadHi)			W	16 位定时/计数器 3 重载计数值的高字节							
	比较器 3 计数值的高字节 (P_TMR3_CompHi)		00h	R	16 位比较器 3 计数值的高字节							
				W	16 位比较器 3 比较值的高字节							
	捕获器 3 捕获值的高字节 (P_TMR3_CapHi)		00h	R	16 位捕获器 3 捕获脉宽值的高字节							
				W	16 位捕获器 3 重载值的高字节							
	捕获器 3 周期值 (P_TMR3_CapCycle8)		00h	R	8 位捕获器 3 捕获的周期值							
				W	8 位捕获器 3 的重载值							
	PWM3 占空比/周期值 (P_TMR3_DutyPeriod)		00h	R	12 位 PWM3 占空比高字节值				12 位 PWM3 周期高字节值			
				W	12 位 PWM3 占空比高字节值				12 位 PWM3 周期高字节值			
$001E	PWM3 低字节占空比 (P_TMR3_PWMDuty)		00h	R	12 位 PWM3 低字节的占空比值							
				W	12 位 PWM3 低字节的占空比值							
$001F	定时/计数器 4-5 控制器 0 (P_TMR4_5_Ctrl0)		00h	R	0	定时/计数器 5 功能选择			0	定时/计数器 4 功能选择		
				W	-	定时/计数器 5 功能选择			-	定时/计数器 4 功能选择		

$0020~002F:定时/计数器 4~5 和 ADC 控制寄存器

地址	功能	上电复位值	复位值	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

地址	功能	上电复位值	复位值	R/W	Bit 7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
$0020	定时/计数器 4-5 控制寄存器 (P_TMR4_5_Ctrl1)		00h	R	0	定时/计数器 5 预分频选择			0	定时/计数器 4 预分频选择		
				W	-	定时/计数器 5 预分频选择			-	定时/计数器 4 预分频选择		
$0021	定时/计数器 4 计数值的低字节 (P_TMR4_Count)		00h	R	8 位或 16 位（低字节）定时/计数器 4 的计数值							
	定时/计数器 4 重载寄存器低字节 (P_TMR4_Preload)			W	8 位或 16 位（低字节）定时/计数器 4 的重载计数值							
	比较器 4 计数值的低字节 (P_TMR4_Comp)		00h	R	8 位或 16 位（低字节）比较器 4 的计数值							
				W	8 位或 16 位（低字节）比较器 4 的比较值							
	捕获器 4 捕获值的低字节 (P_TMR4_Cap)		00h	R	8 位或 16 位（低字节）捕获器 4 的捕获脉宽值							
				W	8 位或 16 位（低字节）捕获器 4 的重载值							
	PWM4 周期值 (P_TMR4_PWMPeriod)		00h	R	8 位 PWM4 的周期值							
				W	8 位 PWM4 的周期值							

地址	功能	上电复位值	复位值	R/W	Bit 7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
$0022	定时/计数器 4 计数值的高字节 (P_TMR4_CountHi)		00h	R	16 位定时/计数器 4 计数值的高字节							
	定时/计数器 4 重载寄存器 (P_TMR4_PreloadHi)			W	16 位定时/计数器 4 重载计数值的高字节							
	比较 4 计数值的高字节 (P_TMR4_CompHi)		00h	R	16 位比较器 4 计数值的高字节							
				W	16 位比较器 4 比较值的高字节							
	捕获器捕获值高字节 (P_TMR4_CapHi)		00h	R	16 位捕获器 4 捕获脉宽值的高字节							
				W	16 位捕获器 4 重载值的高字节							
	捕获器 4 周期值 (P_TMR4_CapCycle8)		00h	R	8 位捕获器 4 捕获的周期值							
				W	8 位捕获器 4 的重载值							
	PWM4 占空比 (P_TMR4_PWMDuty)		00h	R	8 位 PWM4 的占空比值							
				W	8 位 PWM4 的占空比值							
$0023	定时/计数器 5 计数值的低字节 (P_TMR5_Count)		00h	R	8 位或 16 位（低字节）定时/计数器 5 的计数值							

地址	功能	上电复位值	复位值	R/W	Bit 7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
	定时/计数器 5 重载寄存器 (P_TMR5_Preload)			W	8 位或 16 位（低字节）定时/计数器 5 的重载计数值							
	比较器 5 计数值低字节 (P_TMR5_Comp)		00h	R	8 位或 16 位（低字节）比较器 5 的计数值							
				W	8 位或 16 位（低字节）比较器 5 的比较值							
	捕获器 5 捕获值低字节 (P_TMR5_Cap)		00h	R	8 位或 16 位（低字节）捕获器 5 的捕获脉宽值							
				W	8 位或 16 位（低字节）捕获器 5 的重载值							
	PWM5 周期值的低字节 (P_TMR5_PWMPeriodLo)		00h	R	16 位 PWM5 低字节的周期值							
				W	16 位 PWM5 低字节的周期值							
$0024	定时/计数器 5 的高字节 (P_TMR5_CountHi)		00h	R	16 位定时/计数器 5 计数值的高字节							
	定时/计数器 5 重载寄存器高字节 (P_TMR5_PreloadHi)			W	16 位定时/计数器 5 重载计数值的高字节							
	比较 5 计数值的高字节 (P_TMR5_CompHi)		00h	R	16 位比较器 5 计数值的高字节							
				W	16 位比较器 5 比较值的高字节							
	捕获器 5 捕获		00h	R	16 位捕获器 5 捕获脉宽值的高字节							

地址	功能	上电复位值	复位值	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
	值高字节 (P_TMR5_CapHi)			W	16 位捕获器 5 重载值的高字节							
	捕获器 5 周期值 (P_TMR5_CapCycle8)		00h	R	8 位捕获器 5 捕获的周期值							
				W	8 位捕获器 5 的重载值							
	PWM5 周期值的高字节 (P_TMR5_PWMPeriodHi)		00h	R	16 位 PWM5 的高字节的周期值							
				W	16 位 PWM5 的高字节的周期值							
$0025	捕获 5 捕获值低字节 (P_TMR5_CapCycleLo)		00h	R	16 位捕获器 5 捕获的低字节周期值							
				W	-	-	-	-	-	-	-	-
	PWM5 Low 占空比 (P_TMR5_PWMDutyLo)		00h	R	16 位 PWM5 的低字节的占空比值							
				W	16 位 PWM5 的低字节的占空比值							
$0026	中断标志 2 (P_INT_Flag2)		00h	R	-	-	ITVALIF	IICIF	UARTIF	SPIIF	CMP1IF	CMP0IF
				W			‘1’：清除	只读	只读	只读	‘1’：清除	‘1’：清除
$0027	中断控制 2 (P_INT_Ctrl2)		00h	R	-	-	ITVALIE	-	-	-	CMP1IE	CMP0IE
				W	-	-	ITVALIE	-	-	-	CMP1IE	CMP0IE
$0028	A/D 控制 0 (P_AD_Ctrl0		0Ch	R	ADEN	ADVRT	0	0	AD 时钟选择			ADRDY

地址	功能	上电复位值	复位值	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
	)			W	ADEN	ADVRT	-	-	AD 时钟选择			ADSTART
$0029	A/D Input 通道 Ctrl 1 (P_AD_Ctrl1)		0Ch	R	PCFG7	PCFG6	PCFG5	PCFG4	PCFG3	PCFG2	PCFG1	PCFG0
				W	PCFG7	PCFG6	PCFG5	PCFG4	PCFG3	PCFG2	PCFG1	PCFG0
$002A	A/D 控制 2 (P_AD_Ctrl2)		00h	R	ADCE	ADC 通道选择				0	0	Pin8
				W	ADCE	ADC 通道选择				-	-	Pin8
$002B	A/D 转换值的高字节 (P_AD_DataHi)		00h	R	10 位 A/D 高字节值							
				W	-							
$002C	A/D 转换值的低字节 (P_AD_DataLo)		00h	R	10 位 A/D 低字节值		0	0	0	0	0	0
				W	-							
$002D	蜂鸣器控制 (P_BUZ_Ctrl)		00h	R	时基时钟选择				蜂鸣器时钟选择			
				W	时基时钟选择				蜂鸣器时钟选择			
$002E	比较器或控制 (P_CMP_Ctrl)		00h	R	CMP1EN	CMP1RS	CMP1LOG	CMP1OUT	CMP0EN	CMP0RS	CMP0LOG	CMP0OUT
				W	CMP1EN	CMP1RS	CMP1LOG	CMP1OUT	CMP0EN	CMP0RS	CMP0LOG	CMP0OUT
$002F					保留							

$0030~$003F：系统控制和 SPI 控制寄存器

地址	功能	上电复位值	复位值	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
$0030	系统控制 (P_SYS_Ctrl)	80h	C0h	R	POR	ERST	LVR	-	WDTR	IAR	0	0
				W	写“1”清除相应复位标志						-	-
$0031	模式控制 (P_MODE_Ctrl)		00h	R	0	0	0	0	0	0	0	0
				W	写 #$5A 进入 STOP 模式 (C_MODE_STOP = $5A) 写 #$A5 进入 HALT 模式 (C_MODE_HALT = $A5) 写 #$66 复位除 CPU 以外的所有内部模块 (C_MODE_复位 = $66)							
$0032	看门狗控制 (P_WDT_Ctrl)		F2h	R	SCKEN	看门狗时钟选择			0	0	0	0
				W	SCKEN	看门狗时钟选择			-	-	-	-
$0033	IRQ 和 Capture 设置寄存器 0 (P_IRQ_Opt0)		00h	R					IRQ5ES / CAP5ES	IRQM5	IRQ4ES / CAP4ES	IRQM4
				W					IRQ5ES / CAP5ES	IRQM5	IRQ4S / CAP4ES	IRQM4
$0034	IRQ 和 Capture 设置寄存器 1 (P_IRQ_Opt1)		00h	R	IRQ3ES	IRQM3	IRQ2ES	IRQM2	IRQ1ES / CAP3ES	IRQM1	IRQ0ES / CAP2ES	IRQM0
				W	IRQ3ES	IRQM3	IRQ2ES	IRQM2	IRQ1ES / CAP3ES	IRQM1	IRQ0ES / CAP2ES	IRQM0
$0035	慢速输出功能控制		00h	R	0	0	0	0	0	0	0	SLOWE

地址	功能	上电复位值	复位值	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
	(P_IO_Opt)			W	-	-	-	-	-	-	-	SLOWE
$0036	低电压复位选项 (P_LVR_Opt)		00h	R	0	0	0	0	0	0	0	LVRV40
				W	-	-	-	-	-	-	-	LVRV40
$0037					保留							
$0038	SPI 控制寄存器 0 (P_SPI_Ctrl0)		00h	R	SPIEN	MOD	SCKPHA	SCKPOL	SMS	SPI CPU 时钟选择		
				W	SPIEN	MOD	SCKPHA	SCKPOL	SMS	SPI CPU 时钟选择		
$0039	SPI 控制寄存器 1 (P_SPI_Ctrl1)		00h	R	SMSEN	SWRST	-	-	-	-	SPI 采样时钟选择	
				W	SMSEN	SWRST	-	-	-	-	SPI 采样时钟选择	
$003A	SPI TX/RX 状态 (P_SPI_Status)		00h	R	SPIIF	SPIIEN	TXBF	0	0	-	-	BUFFull
				W	‘1’:清除	SPIIEN	-	-	-	-	-	-
$003B	SPI TX 数据 (P_SPI_Tx 数据)		00h	R	0	0	0	0	0	0	0	0
				W	SPI 发送数据到缓冲器中							
$003C	SPI RX 数据 (P_SPI_Rx 数据		00h	R	SPI 从缓冲器中接收数据							
				W	-	-	-	-	-	-	-	-
$003D					保留							

地址	功能	上电复位值	复位值	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
$003E					保留							
$003F					保留							

$0040~$004F：端口 E/F 和 UART/IIC 控制寄存器

地址	功能	上电复位值	复位值	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
$0040	端口 E 电平状态 (P_IOE_Data)		00h	R	端口 E 管脚状态							
				W	端口 E 输出数据(写入$005D)							
$0041	端口 F 电平状态 (P_IOF_Data)		00h	R	端口 F 管脚状态							
				W	端口 F 输出数据(写入$005E)							
$0042	端口 E 方向 (P_IOE_Dir)		00h	R	端口 E 数据方向寄存器。0=IN, 1=OUT.							
				W	端口 E 数据方向寄存器。0=IN, 1=OUT.							
$0043	端口 F 方向 (P_IOF_Dir)		00h	R	端口 F 数据方向寄存器。0=IN, 1=OUT.							
				W	端口 F 数据方向寄存器。0=IN, 1=OUT.							
$0044	端口 E 属性 (P_IOE_Attrib)		00h	R	端口 E 属性寄存器							
				W	端口 E 属性寄存器							
$0045	端口 F 属性 (P_IOF_Attrib)		00h	R	端口 F 属性寄存器							
				W	端口 F 属性寄存器							
$0046	UART 控制 (P_UART_Ctrl		00h	R	RXIE	TXIE	RXEN	TXEN	SOFTRST	STOPSEL	PSEL	PEN

地址	功能	上电复位值	复位值	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
	)			W	RXIE	TXIE	RXEN	TXEN	SOFTRST	STOPSEL	PSEL	PEN
$0047	UART 波特率分频寄存器 (P_UART_Baud)		00h	R	UART 波特率分频寄存器							
				W	UART 波特率分频寄存器							
$0048	UART 状态 (P_UART_Status)		00h	R	RXIF	TXIF	BUSY	0	0	OERR	PERR	FERR
				W	-	-	-	-	-	OERR	PERR	FERR
$0049	UART 数据 (P_UART_Data)		00h	R	UART 接收数据							
				W	UART 发送数据							
$004A	IIC Bus 控制 (P_IIC_Ctrl)		00h	R	IICEN		TXRXEN	INTEN	ACKEN	IIC 时钟选择		
				W	IICEN		TXRXEN	INTEN	ACKEN	IIC 时钟选择		
$004B	IIC 总线状态 (P_IIC_Status)		00h	R	MXT	TXRXSEL	BUSY/SIGGEN	IICIF	ARBITRAT	AAS	AZERO	LRB
				W	MXT	TXRXSEL	BUSY/SIGGEN	IICIF	ARBITRAT	AAS	AZERO	LRB
$004C	IIC 总线数据 (P_IIC_Data)		00h	R	IIC 总线串行数据寄存器							
				W	IIC 总线串行数据寄存器							
$004	IIC 总线地址		00h	R	IIC 总线串行地址寄存器							

地址	功能	上电复位值	复位值	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
D	(P_IIC_Address)			W	IIC 总线串行地址寄存器							
$004E					保留							
$004F					保留							

$0058~$005F：DAC 寄存器和端口数据寄存器

地址	功能	上电复位值	复位值	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
$0050~54					保留							
$0055	DAC 控制 (P_DA_Ctrl)		00h	R	DAEN	0	0	0	0	0	0	0
				W	DAEN	-	-	-	-	-	-	-
$0056	DAC 数据低字节 (P_DA_DataLo)		00h	R	10 位 D/A 转换值的低 2 位		0	0	0	0	0	0
				W	10 位 D/A 转换值的低 2 位		-	-	-	-	-	-
$0057	DAC 数据高字节 (P_DA_DataHi)		00h	R	10 位 D/A 转换值的高 8 位							
				W	10 位 D/A 转换值的高 8 位							

地址	功能	上电复位值	复位值	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
$0058	捕获控制 (P_CAP_Ctrl)		00h	R	CAPOPT	CAPIP4	CAPIP3	CAPIP2	CAPIP1	CAPIP0	CAP1ES	CAP0ES
				W	CAPOPT	CAPIP4	CAPIP3	CAPIP2	CAPIP1	CAPIP0	CAP1ES	CAP0ES
$0059	端口 A 锁存缓冲器 (P_IOA_Buf)		00h	R	端口 A 数据锁存							
				W	端口 A 数据锁存							
$005A	端口 B 锁存缓冲器 (P_IOB_Buf)		00h	R	端口 B 数据锁存							
				W	端口 B 数据锁存							
$005B	端口 C 锁存缓冲器 (P_IOC_Buf)		00h	R	端口 C 数据锁存							
				W	端口 C 数据锁存							
$005C	端口 D 锁存缓冲器 (P_IOD_Buf)		00h	R	端口 D 数据锁存							
				W	端口 D 数据锁存							
$005D	端口 E 锁存缓冲器 (P_IOE_Buf)		00h	R	端口 E 数据锁存							
				W	端口 E 数据锁存							
$005E	端口 F 锁存缓冲器 (P_IOF_Buf)		00h	R	端口 F 数据锁存							
				W	端口 F 数据锁存							
$005F	捕获器5 高		00h	R	16 位捕获器 5 捕获周期值的高字节							

地址	功能	上电复位值	复位值	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
	字节 (P_TMR5_CapCycleHi)			W	-	-	-	-	-	-	-	-
	PWM5 占空比值的高字节 (P_TMR5_PWMDutyHi)		00h	R	16 位 PWM 占空比值的高字节							
				W	16 位 PWM 占空比值的高字节							

4.4.2 用户 RAM 区

SPMC65X 系列芯片的用户 RAM 区位于$0060 ~ $03FF 的地址空间内，可以用作堆栈，可读可写。

4.4.3 堆栈空间

堆栈区域为$01E0~$01FF 共 32 字节，初始状态堆栈指针（SP）指向$01FF，每压入一个字节的数据，指针减一，一旦超出$01E0，CPU 复位。

当中断发生时，中断返回地址和状态寄存器的值被压入堆栈。中断服务子程序执行完毕，执行指令 RTI 后，中断返回地址和状态寄存器的值被弹出堆栈，程序返回。

同样，当调用子程序时，子程序的返回地址也会被压入堆栈。子程序执行完毕，执行指令 RTS 后，子程序的返回地址被弹出堆栈，程序返回。

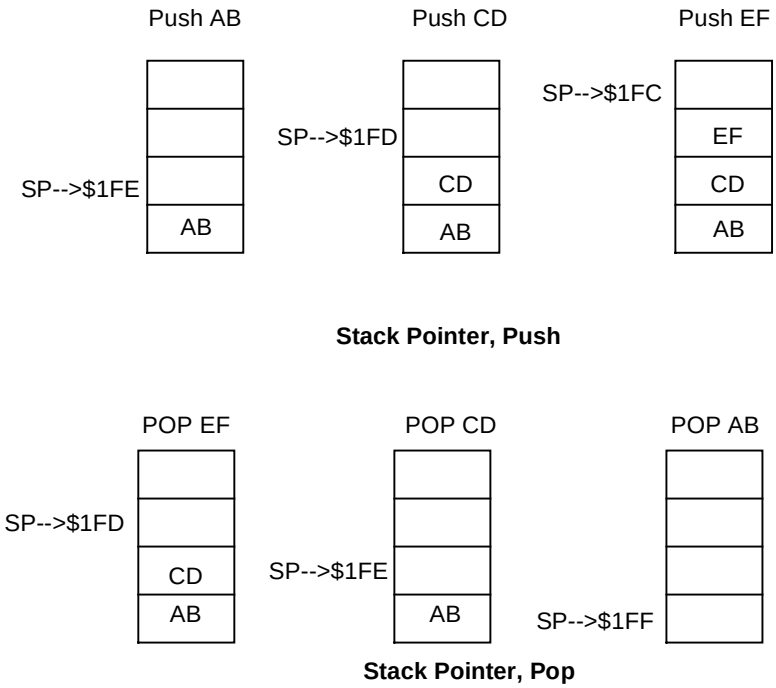


图 4-4 SPMC65X 堆栈操作

堆栈指针（SP）用于指示当前的堆栈状况，每当压入一个数据，指针会自动减 1，每当弹出一个数据，指针又会自动加 1，所以堆栈指针（SP）总是指向堆栈的顶端。详细操作见图 4-4。

4.4.4 芯片配置寄存器

$7FE0h ~ $7FE3h 为芯片配置寄存器，用户可以设置一些特殊功能，以满足不用应用的需要，如低电压复位、看门狗复位、非屏蔽中断源和时钟源的选择等。在进行芯片烧录的时候，这些设置会通过烧录器写入芯片。如果使用凌阳公司的集成开发环境 FortisIDE，可以用图 4-5 所示的方法设置这些寄存器。操作步骤：打开[Project **à** Setting **à** Mask Option]，然后在对话框里进行特殊功能的选择。

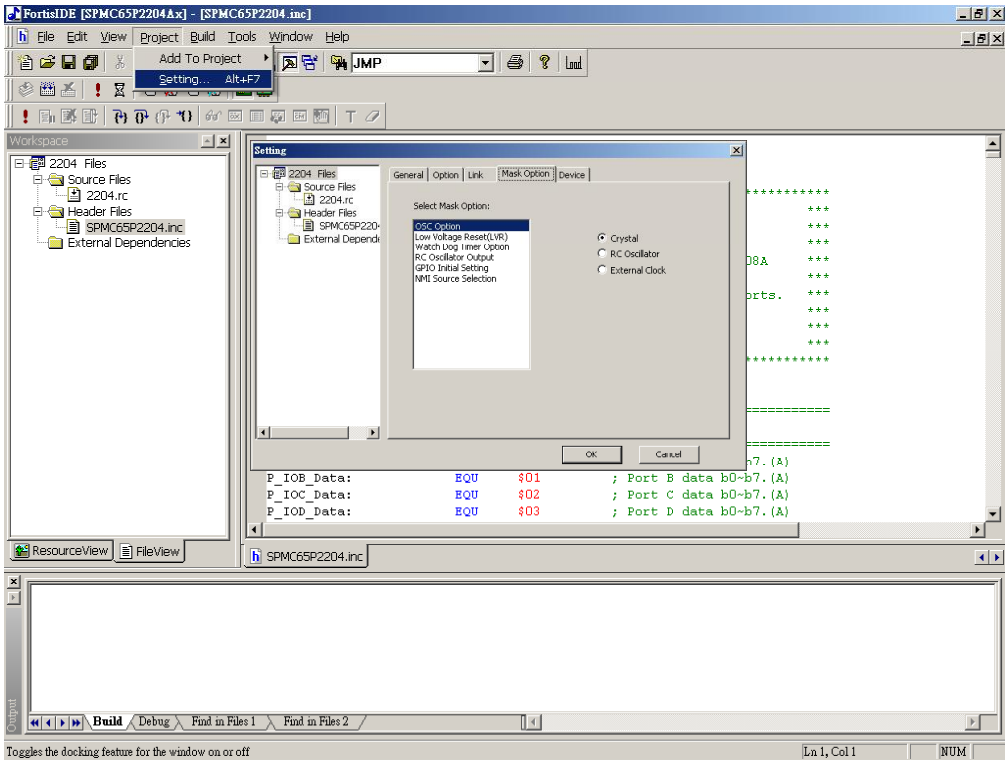


图 4-5 芯片配置寄存器

在[OSC Option]选项中，有三种时钟源可选，即[Crystal]、[RC Oscillator] 和 [External Clock]，可以在 Fortis IDE 中进行设置，请参考第 7 章，作出适当的选择。

在[Low Voltage Reset (LVR)]选项中，可以选择[Enabled] 或 [Disabled]。如果选择[Enabled]，那么系统将会启动低电压复位功能，请参考第 5 章，作出适当的选择。

在[Watch Dog Timer Option]选项中，可以选择[Enabled]或[Disabled]。请参考第 5 章，作出适当的选择。

在[RC Oscillator Output]选项中，可以选择[Clock Output]或[No Output]。请参考第 7 章，作出适当的选择。

在[GPIO Initial Setting]选项中，可以选择[All float] 或 [All pull low]。请参考第 8 章，作出适当的选择。

在[NMI Source Selection]选项中，可以选择[Disabled]、[PB4(INT0)]、[PB5(INT1)]、[PD0(INT2)]、[PD1(INT3)]、[PD4(INT4)]或[PD5(INT5)]。请参考第 6 章，作出适当的选择。

4.4.5 用户信息

SPMC65X 系列预留了$7FF0 ~ $7FFF 区域内作为用户信息块，如系列号或版本控制信息。凌阳公司将这一区域定义为系列号和产品信息区，用户可以写入任意信息。关于用户信息块的详细设置见第 23 章。

4.5 特殊控制寄存器

在 SPMC65X 系列芯片中，位于内存$30~$36 之间的寄存器是与系统运行相关的寄存器，需要慎重配置。因此访问$30~$36 之间的寄存器时，必须进行连续两次的写操作才能设置成功，目的是为了保证写入控制寄存器的内容正确并且可以防止数据或地址总线上的噪声干扰。

【例 4 2】：清除复位标志

```
lda     #$FF                    ; 清除复位标志
sta     P_SYS_Ctrl
sta     P_SYS_Ctrl
```

【例 4 3】：复位所有 IO 端口

```
lda     #C_MODE_Reset           ; 复位所有 IO 端口
sta     P_MODE_Ctrl
sta     P_MODE_Ctrl
```

【例 4 4】:看门狗中断频率为 1.5Hz

```
lda     #C_WDT_Div_16384        ; WDI= Fslow(25KHz)/16384= 1.5Hz
sta     P_WDT_Ctrl
sta     P_WDT_Ctrl
```

【例 4 5】: INT0 为上升沿触发，其它外部中断为下降沿触发

```
lda     #C_IRQOpt1_IRQ0ES       ; INT0 为上升沿触发，其它外部中断为下降沿触发
sta     P_IRQ_Opt1
sta     P_IRQ_Opt1
```

【例 4 6】:设置低电压复位值为 4.0V

```
lda     # C_LVR_V40             ; 设置低电压复位值为 4.0V
sta     P_LVR_Opt
sta     P_LVR_Opt
```

4.6 设计参考

无

4.7 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题	**应用例系列号**
本章无相关应用例	

库函数

标题	**应用例系列号**
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

4.8 修订记录

版本号	日期	备注
V0.1	03/31/2005	第一版

5 复位

5.1 简述

系统中共有 5 种外部和内部的复位源。外部复位来自电源线或外部触发事件；内部复位来自程序异常或程序设置的复位事件。这 5 个复位分别是：

(1) 上电复位(POR)

(2) 外部复位(RESET)

(3) 低电压复位 (LVR)

(4) 看门狗复位 (WDTR)

(5) 非法地址复位 (IAR)

复位时序见图 5-1：

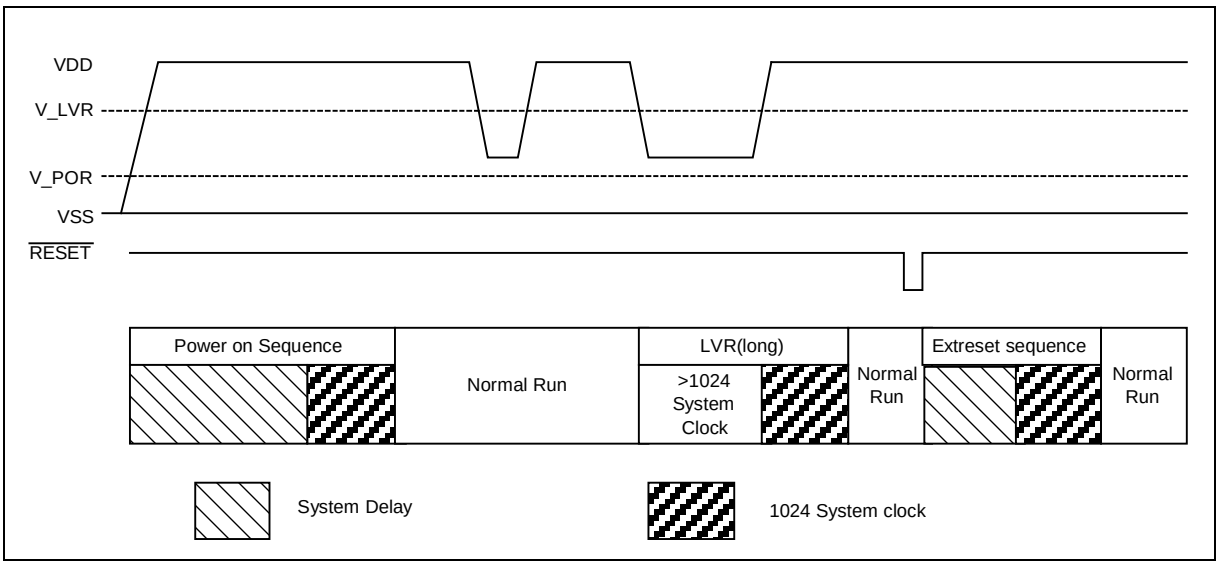


图 5-1 复位时序

按照复位结果可以分为系统复位和 CPU 复位。系统复位即复位所有的内部模块包括 CPU 在内；CPU 复位只会复位 CPU，程序重新从头开始执行。有些事件导致系统复位，有些只是导致 CPU 复位，具体信息见下表 5－1 所示。

表 5-1 复位源列表

复位源	外部/内部.	复位功能	注意
上电复位 (POR)	外部	复位所有模块，包括 CPU	
外部复位(RESET)	外部	复位所有模块，包括 CPU	
低电压复位(LVR)	外部	复位所有模块，包括 CPU	复位电压可选 （Case of>T） T=1024 系统时钟

复位源	外部/内部.	复位功能	注意
看门狗复位(WDR)	内部	仅复位 CPU 和看门狗电路	复位时间长度可选
非法地址复位(IAR)	内部	仅复位 CPU 和看门狗电路	

5.2 控制寄存器

5.2.1 系统控制寄存器(P_SYS_Ctrl, $30)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
POR	ERST	LVR	-	WDR	IAR	-	-
R/W	R/W	R/W	-	R/W	R/W	-	-
1	0	0	0	0	0	0	0

Bit 7 **POR**:上电复位标志

0 = 无上电复位发生

1 = 上电复位发生

Bit 6 **ERST**:外部复位标志

0 = 无外部复位发生

1 = 外部复位发生

Bit 5 **LVR**：低电压复位标志

0 = 无低电压复位发生

1 = 低电压复位发生

Bit 4 保留

Bit 3 **WDR**: 看门狗复位标志

0 = 无看门狗复位发生

1 = 看门狗复位发生

Bit 2 **IAR**：非法地址复位标志

0 = 无非法地址复位发生

1 = 非法地址复位发生

Bit [1:0] 保留

注意：

向相应位写“1”清除该标志。

对上述位进行设置时需要连续写两次才能将值写入。

5.2.2 看门狗控制寄存器(P_WDT_Ctrl, $32)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
WDTES	WDS2	WDS1	WDS0	-	-	-	-
R/W	R/W	R/W	R/W				
1	1	1	1	0	0	0	0

Bit 7 **WDTES**：STOP 模式下看门狗使能位

0 = STOP 模式下看门狗使能

1 = STOP 模式下看门狗禁止

Bit [6:4] 看门狗中断频率选择位

000 = f_{slow}/128

001 = f_{slow}/256

010 = f_{slow}/512

011 = f_{slow}/1024

100 = f_{slow}/2048

101 = f_{slow}/4096

110 = f_{slow}/8192

111 = f_{slow}/16384

f_{slow}:内置的 RC 振荡电路，振荡频率通常为 25kHz。

WDS[2:0]	25kHz 分频	看门狗中断频率 (Hz)	看门狗复位频率(Hz)
000	f_{slow}/128	195	195/8
001	f_{slow}/256	97	97/8
010	f_{slow}/512	48	48/8
011	f_{slow}/1024	24	24/8
100	f_{slow}/2048	12	12/8
101	f_{slow}/4096	6	6/8
110	f_{slow}/8192	3	3/8
111	f_{slow}/16384	1.5	1.5/8

注意：对上述位进行设置时需要连续写两次才能将值写入。

5.2.3 低电压复位控制寄存器(P_LVR_Opt, $36)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	-	-	-	-	-	-	LVRV40
-	-	-	-	-	-	-	R/W
0	0	0	0	0	0	0	0

Bit [7:1] 保留

Bit 0 **LVRV40**:复位电压选择位

0 =低于 2.5V

1 =低于 4.0V

注意：

芯片工作后，该选项只能设置一次，只有上电复位才能将其清除。

对上述位进行设置时需要连续写两次才能将值写入。

5.3 操作

系统中共有 5 种复位源。分别是：上电复位(POR)、外部复位(ERST)、低电压复位(LVR)、看门狗复位 (WDR)和非法地址复位(IAR)。当有系统复位时，用户可以通过系统控制寄存器(P_SYS_Ctrl)查询到它的复位源，参见【例 5 1】。关于复位源操作的详细信息将在下面介绍。

【例 5 1】:保存 P_SYS_Ctrl 的值，然后再清除。

```
G_MWorkReg1     DS              1
    lda         P_SYS_Ctrl      ;将复位标志读出
    sta         G_MWorkReg1
    lda         #$FF            ;清除复位标志
    sta         P_SYS_Ctrl
    sta         P_SYS_Ctrl
```

5.3.1 上电复位

当芯片的电源电压 VDD 上升到 1.45V 左右，产生上电复位。上电复位将会复位整个芯片和所有的寄存器。通常情况下，RESETB 管脚需要外接 RC 电路，这样，上电复位发生后，经过一段延迟，RESETB 管脚才能变为高电平。对于 SPMC65X 系列芯片，其优点为：RESETB 管脚只需直接拉高（或接一个上拉电阻）到 VDD，就可立即复位，不需外接 RC 电路，不会造成上电延迟。复位后，所有的寄存器都初始化为默认值。如图 5-2 所示为上电复位的时序。

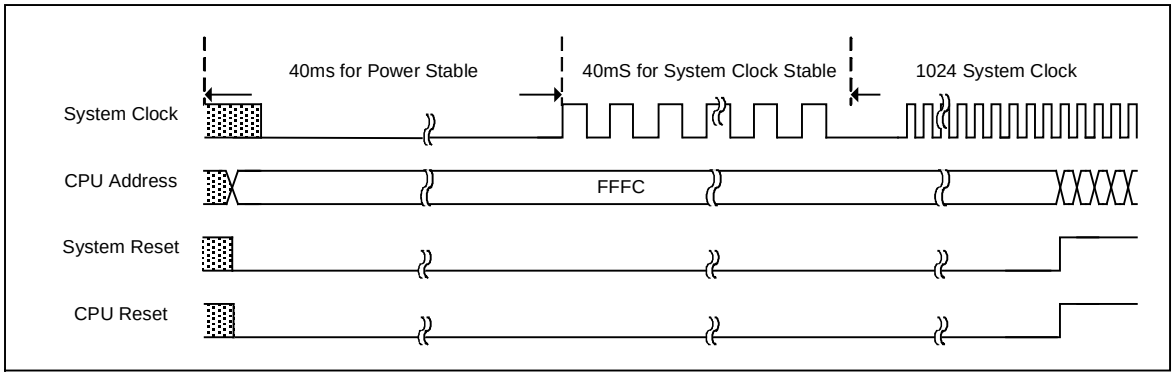


图 5-2 上电复位时序

5.3.2 外部复位

SPMC65X 系列芯片提供了一个外部复位接口，可手动强制系统复位，即把 RESETB 管脚连接到一个外部 RC 电路中，如图 5-3 所示，当 VDD 电源下降时，二极管 D1 帮助电容迅速放电。推荐采用 R1< 40KΩ，保证 R 的分压不违反芯片的电气指标，如果采用 R4= 33Ω ~1KΩ，将会限制通过 RESETB 的电流，提高抗干扰能力（ESD）。

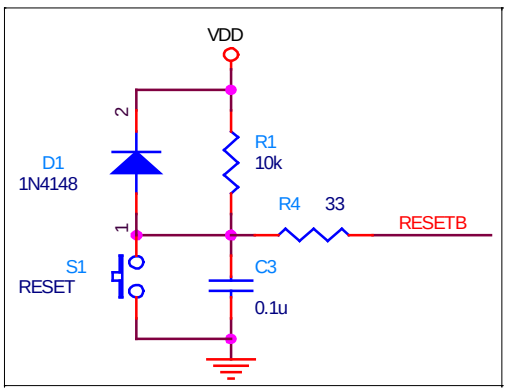


图 5-3 复位电路

由于 RESETB 为低有效，所以当电压低于 0.3 * VDD 时，系统复位。如果电容再充电的电压大于系统可以正常工作的电压，系统在 PWRT 延时之后完成整个复位。外部复位时序见图 5-4。

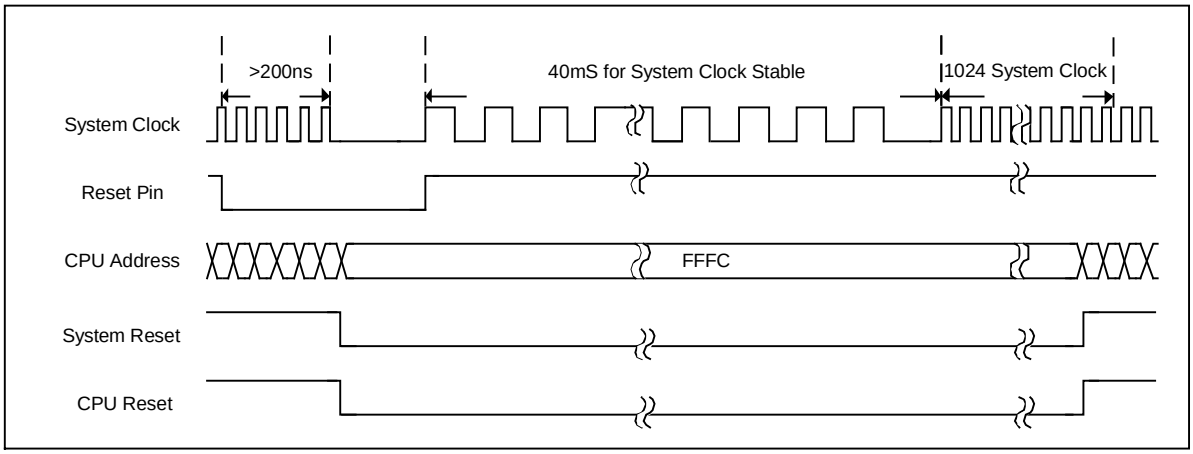


图 5-4 外部复位序列

5.3.3 低电压复位

低电压复位即当 MCU 的电源电压低于设定的复位电压后，芯片复位，保证了 MCU 不会在非正常的电压范围内连续工作。通过寄存器可以禁止/使能 LVR 功能。如图 5-5 所示，使能 LVR 功能后，便开始对电源电压进行检测，如果监测到电源电压低于设定的电压达到一定的时间长度（1024 个系统时钟周期），系统便会复位。在低电压回升到要求的的高度之后，再经过 1024 个系统时钟周期，系统恢复正常，开始正常运行。

另外，低电压复位在上电复位周期内或外部复位（RESETB）周期内无效。如图 5-5 所示为低电压复位时序。

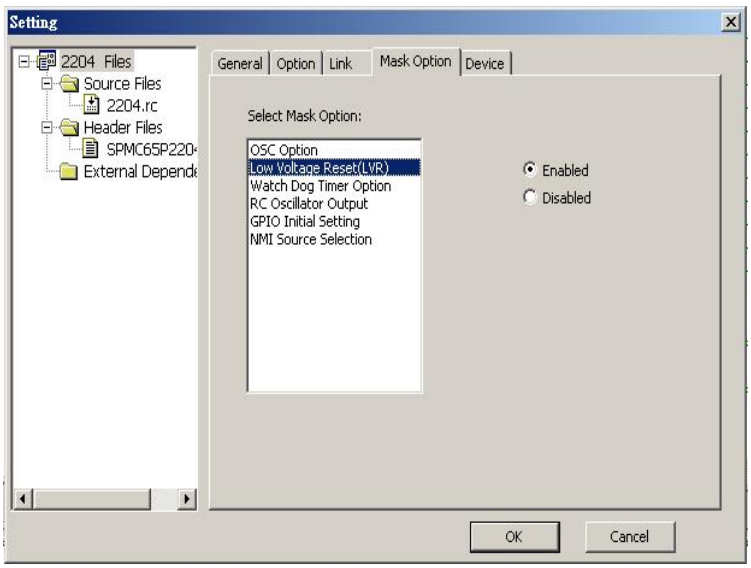


图 5-5 LVR 设置窗口

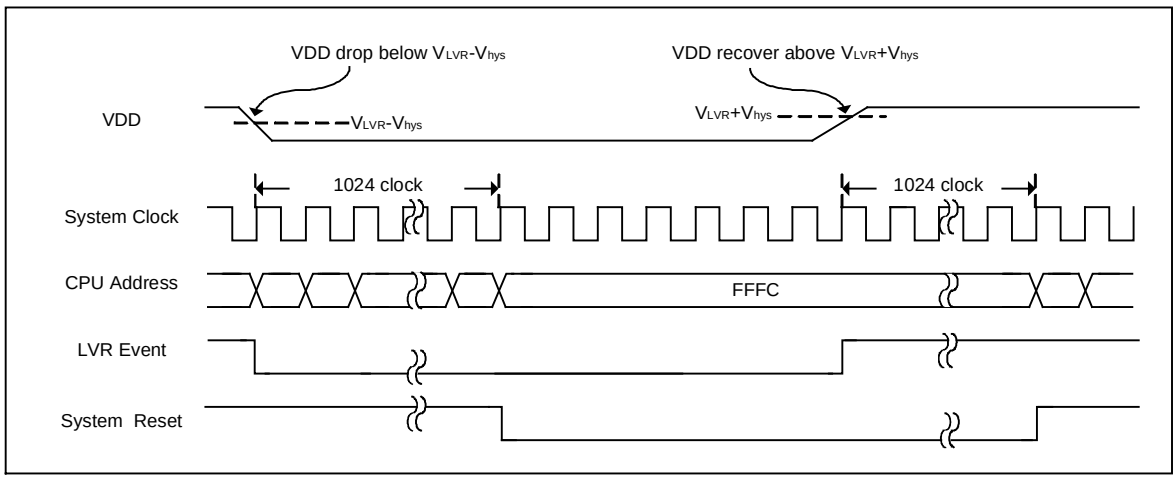


图 5-6 LVR 复位时序

【例 5 2】：电压低于 4V 时，使能电压复位。

```
lda     #C_LVR_V40          ; 设置 LVR= 4.0V
sta     P_LVR_Opt
sta     P_LVR_Opt
```

5.3.4 看门狗复位

当程序运行异常（跑飞或进入死循环等），没有在规定的时间内清除看门狗，看门狗电路就会产生内部复位信号，将 CPU 复位，保证了 MCU 不会在异常情况下连续工作。

看门狗功能可通过寄存器使能和禁止。如图 5-7 所示。看门狗定时器开始计时，在规定的时间内向 P_WDT_Clr 写入“#$55”，可以清除看门狗定时器，否则，看门狗定时器溢出，会产生看门狗复位信号，使 CPU 复位。看门狗电路内置独立的 RC 振荡电路，通常振荡频率为 25kHz。通过寄存器 P_WDT_Ctrl[6:4]的设置，看门狗定时器中断频率可以有 8 种选择。CPU 复位时会清除看门狗定时器并重新开始定时，但是不能清除看门狗定时器的复位标志。如图 5-8 所示为看门狗复位时序。详细操作见 17 章。

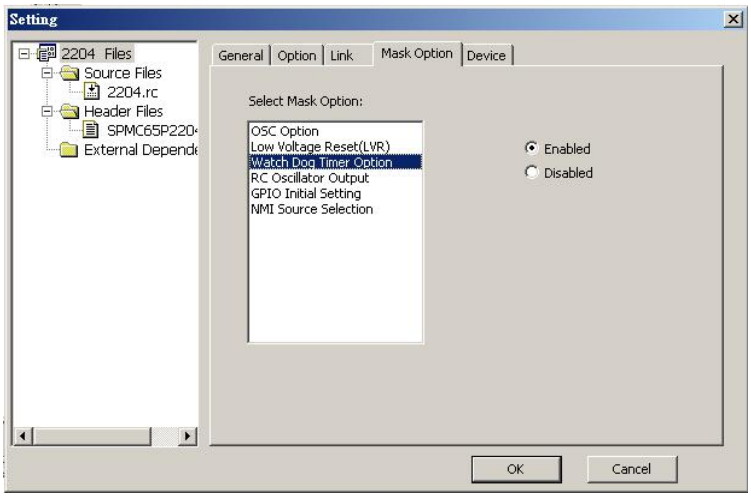


图 5-7 WDT 设置窗口

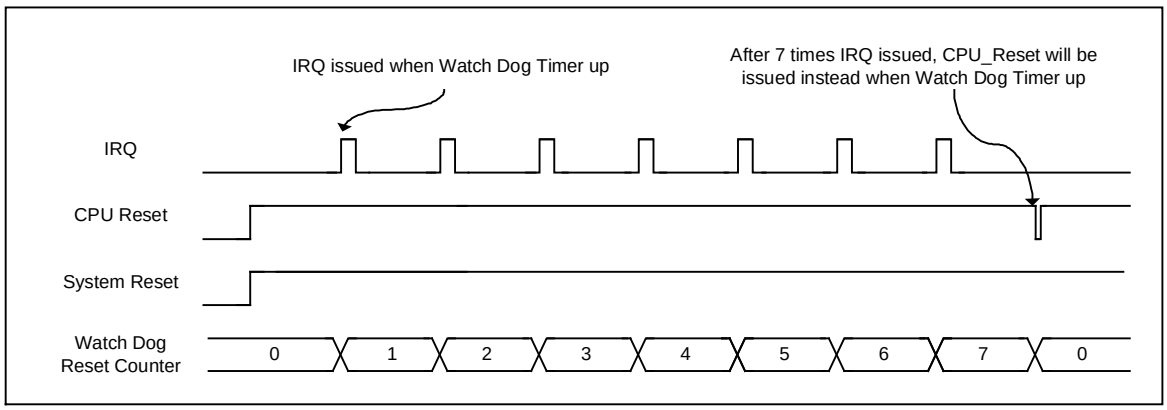


图 5-8 看门狗复位时序

5.3.5 非法地址复位

非法地址复位(IAR)是内部复位，可以防止系统进入非法地址。当程序对某个地址进行读写操作时，这个地址既不在工作区域，也不在堆栈区，便会产生非法地址复位信号，从而使 CPU 复位。如图 5-9 所示为非法地址复位时序。

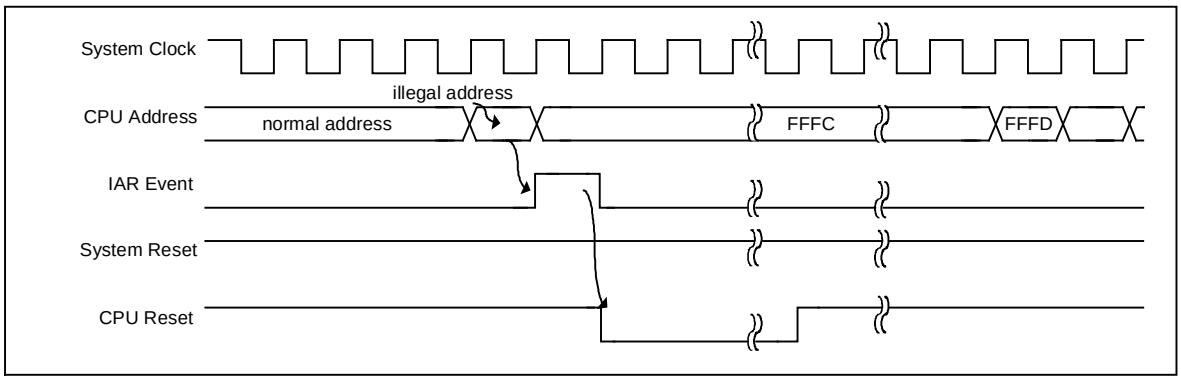


图 5-9 非法地址复位时序

5.4 设计参考

【例 5 3】：以芯片 SPMC65P2404A 为例，使能低电压复位功能，复位电压为 4.0V，具体范例如下：

```
.SYNTAX     6502                ; process standard 6502 addressing syntax
.LINKLIST                       ; generate linklist information
.SYMBOLS                        ; generate symbolic debug information

.INCLUDE                        spmc65p2404a.inc

    .PAGE0                      ; define values in the range from $00 to $FF
;
    .DATA                       ; define data storage section;
    .CODE
;=====================================
;=                                  =
;= Power on Reset Process - Main Program  =
;=                                  =
;=====================================
V_Reset:
    sei                         ; 关闭总的中断开关
    ldx     #C_STACK_BOTTOM     ; 将堆栈指针初始化为$00FF 位置上
    txs                         ; 将 X 寄存器的值传送到堆栈指针(SP)中

    jsr     F_InitIOPort        ; 初始化 IO 端口

    lda     P_SYS_Ctrl
    sta     P_IOA_Data          ; 复位标志通过 IOA 显示
```

```
    set         P_SYS_Ctrl, CB_SCR_LVR1
    set         P_SYS_Ctrl, CB_SCR_LVR1         ; 清除 LVR 复位标志

    set          P_LVR_Opt,CB_LVR_V40
    set          P_LVR_Opt,CB_LVR_V40           ; 设置低电压复位值为 4.0V

L_Main:
    nop
    jmp         L_Main
;
;====================================
;=                                  =
;=       Interrupt Vector Table     =
;=                                  =
;====================================
VECTOR:             .SECTION
    DW      V_NMI               ; may download program emulated either
    DW      V_Reset             ; in internal memory or external memory
    DW      V_IRQ               ; dw define two bytes interrupt vector
;
;====================================
;=                                  =
;=       End Of Interrupt Vector Table  =
;=                                  =
;====================================
    .END                            ;
```

5.5 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题 **应用例系列号**

本章无相关应用例

库函数

标题	应用例系列号
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

5.6 Revision History

版本号	日期	备注
V0.1	03/31/2005	第一版

6 中断

6.1 简述

SPMC65X 系列芯片共有 7 种中断源：定时/计数器溢出中断、捕获中断、外部中断、ADC 中断、通讯中断、看门狗中断和时基中断。每个中断都有各自的标志位和控制寄存器，用于标志中断是否发生和设置中断的禁止/使能。当设定了相应的控制寄存器使中断使能，还必须执行指令“CLI”打开总的中断开关，中断才可以产生。通常，当相关中断控制位设置为使能后，一旦中断事件发生，便会产生中断请求信号，向 CPU 发出中断请求，并且相应的中断标志位也会置 1。在 CPU 进入了中断服务程序进行中断处理时，必须清除中断标志位，否则，CPU 会循环进入中断服务程序。

各中断源见表 6-1。注意：带有*标记的中断源(IRQ0 和 Capture2、IRQ1 和 Capture3、IRQ4 和 Capture4、IRQ5 和 Capture5)共享中断控制寄存器和中断状态寄存器。

表 6-1 中断源列表

中断源		中断状态寄存器	中断控制寄存器	中断源		中断状态寄存器	中断控制寄存器
定时/计数器溢出中断	Timer0	T0OIF ($0E.0)	T0OIE ($0F.0)	外部中断	IRQ0*	IRQ0IF ($0C.0)	IRQ0IE ($0D.0)
	Timer1	T1OIF ($0E.1)	T1OIE ($0F.1)		IRQ1*	IRQ1IF ($0C.1)	IRQ1IE ($0D.1)
	Timer2	T2OIF ($0E.2)	T2OIE ($0F.2)		IRQ2	IRQ2IF ($0C.2)	IRQ2IE ($0D.2)
	Timer3	T3OIF ($0E.3)	T3OIE ($0F.3)		IRQ3	IRQ3IF ($0C.3)	IRQ3IE ($0D.3)
	Timer4	T4OIF ($0E.4)	T4OIE ($0F.4)		IRQ4*	IRQ4IF ($0C.4)	IRQ4IE ($0D.4)
	Timer5	T5OIF ($0E.5)	T5OIE ($0F.5)		IRQ5*	IRQ5IF ($0C.5)	IRQ5IE ($0D.5)
捕获中断	Capture0	CAP0IF ($0E.6)	CAP0IE ($0F.6)	ADC 中断	ADC	ADIF ($0C.7)	ADIE ($0D.7)
	Capture1	CAP1IF ($0E.7)	CAP1IE ($0F.7)		Comparator0	CMP0IF ($26.0)	CMP0IE ($27.0)

中断源		中断状态寄存器	中断控制寄存器	中断源		中断状态寄存器	中断控制寄存器
	Capture2*	CAP2IF ($0C.0)	CAP2IE ($0D.0)		Comparator1	CMP1IF ($26.1)	CMP1IE ($27.1)
	Capture3*	CAP3IF ($0C.1)	CAP3IE ($0D.1)	通讯中断	SPI	SPIIF ($26.2)(R) SPIIF ($3A.7)(R/W)	SPIIE ($3A.6)
	Capture4*	CAP4IF ($0C.4)	CAP4IE ($0D.4)		UART	UARTIF ($26.3)(R) RXIF ($48.7)(R) TXIF ($48.6)(R)	RXIE ($46.7) TXIE ($46.6)
	Capture5*	CAP5IF ($0C.5)	CAP5IE ($0D.5)		IIC	IICIF($26.4) (R) IICIF($4B.4)(R/W)	IICIE ($4A.4)
看门狗中断		WDIF ($0C.6)	WDIE ($0D.6)	时基中断		ITVALIF ($26.5)	ITVALIE ($27.5)

外部中断 INT0~5 共有 6 个通道，每个通道的中断都有独立的控制位和中断标志位。控制位用于设置外部中断的触发模式：沿触发或电平触发。注意：非屏蔽中断 NMI 不可以设置为电平触发，也就是说 NMI 只有上升沿或者下降沿触发。中断标志位用于标志中断是否发生。一旦中断发生，中断标志便会被置 1，直到在程序中将其清除。

当外部中断被设置为沿触发模式时，一个有效的边沿输入便会触发中断。如果设置为电平触发，一个持续有效的电平会令中断产生，直到该电平被去掉。

为了增强抗干扰能力，防止外部中断被错误的设置，中断相关寄存器（包括 P_IRQ_Opt0 和 P_IRQ_Opt1）需要被写两次才能将设置值写入，以便更好的保护。

SPMC65X 系列芯片最多可有 6 个外部中断源，每款芯片具备的外部中断源数量各不相同。例如 SPMC65P2404A 只有 4 个外部中断源，INT0~INT3。6 个外部中断中只能将其中一个设置为非屏蔽中断(NMI)，在芯片烧录的时候会被写入芯片的 OTP ROM 中，并且不会改变。“NMI”的选择方法见图 6-1(基于 SPMC65P2404A)。如果设置通用的外部中断，选择“Disabled”项，如果设置 NMI，选择相应的中断。

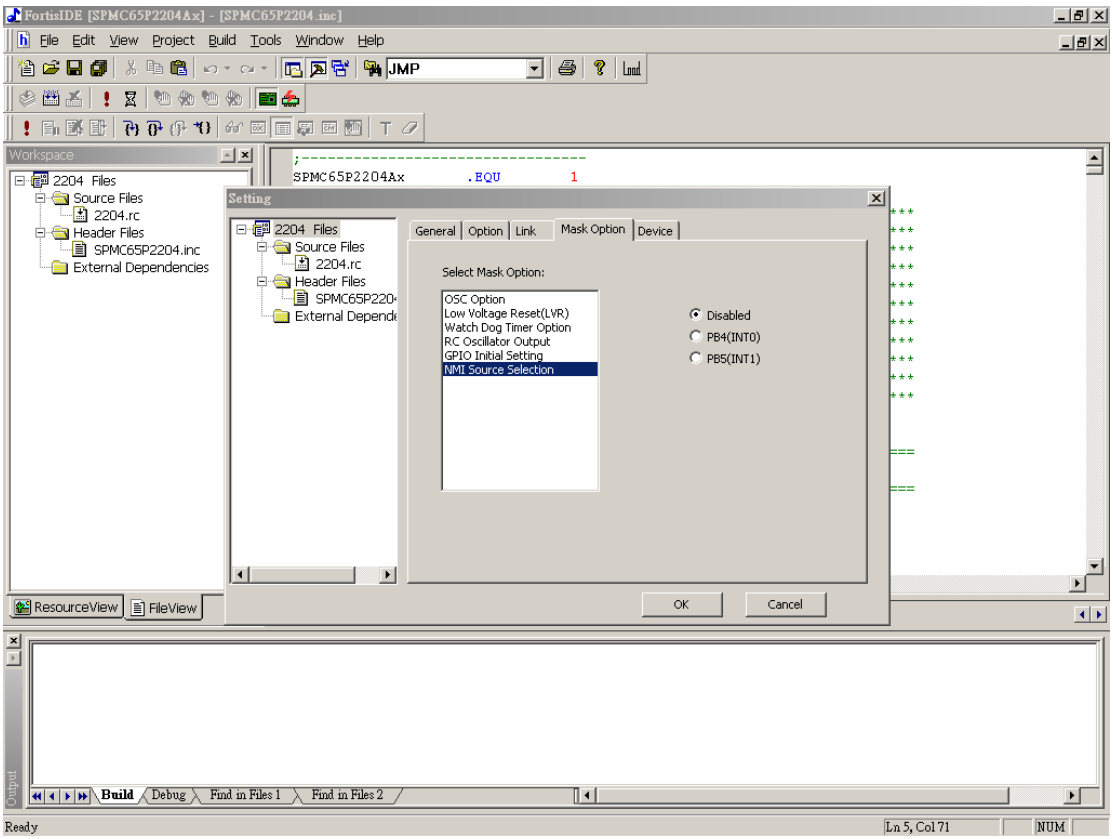


图 6-1 基于 SPMC65P2404A 的 NMI 中断源选择

定时/计数器溢出中断、时基中断、看门狗中断、ADC 中断、捕获中断、比较中断和通讯中断在后续章节都有详细描述。

6.2 控制寄存器

6.2.1 中断标志寄存器 0 (P_INT_Flag0, $0C) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADIF	WDIF	IRQ5IF/ CAP5IF	IRQ4IF/ CAP4IF	IRQ3IF	IRQ2IF	IRQ1IF/ CAP3IF	IRQ0IF/ CAP2IF
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 **ADIF**: ADC 中断标志

0 = ADC 中断没有发生

1 = ADC 中断发生

Bit 6 **WDIF**:看门狗中断标志

0 = 看门狗中断没有发生

1 = 看门狗中断发生

Bit 5 **IRQ5F/CAP5F**: IRQ5/CAP5 中断标志

0 = IRQ5/CAP5 中断没有发生

1 = IRQ5/CAP5 中断发生

Bit 4 **IRQ4F/CAP4F**: IRQ4/CAP4 中断标志

0 = IRQ4/CAP4 中断没有发生

1 = IRQ4/CAP4 中断发生

Bit 3 **IRQ3IF**: IRQ3 中断标志

0 = IRQ3 中断没有发生

1 = IRQ3 中断发生

Bit 2 **IRQ2IF**: IRQ2 中断标志

0 = IRQ2 中断没有发生

1 = IRQ2 中断发生

Bit 1 **IRQ1IF/CAP3IF**: IRQ1/CAP3 中断标志

0 = IRQ1/CAP3 中断没有发生

1 = IRQ1/CAP3 中断发生

Bit 0 **IRQ0IF/CAP2IF**: IRQ0/CAP2 中断标志

0 = IRQ0/CAP2 中断没有发生

1 = IRQ0/CAP2 中断发生

注意：

写入“1”清除相应标志

IRQ5、IRQ4 分别和 CAP5、CAP4 共享中断标志位

IRQ1、IRQ0 分别和 CAP3、CAP2 共享中断标志位

6.2.2 中断控制寄存器 0 (P_INT_Ctrl0, $0D) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADIE	WDIE	IRQ5E/ CAP5E	IRQ4E/ CAP4E	IRQ3IE	IRQ2IE	IRQ1IE/ CAP3IE	IRQ0IE/ CAP2IE
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 **ADIE**: AD 转换中断使能位

0 =禁止

1 =使能

Bit 6 **WDIE**: 看门狗中断使能位

0 = 禁止

1 = 使能

Bit 5 **IRQ5IE/CAP5IE**: IRQ5/CAP5 使能位

0 = 禁止

1 = 使能

Bit 4 **IRQ4IE/CAP4IE**: IRQ4/CAP4 使能位

0 = 禁止

1 = 使能

Bit 3 **IRQ3IE**: IRQ3 使能位

0 = 禁止

1 = 使能

Bit 2 **IRQ2IE**: IRQ2 使能位

0 = 禁止

1 = 使能

Bit 1 **IRQ1IE/ CAP3IE**: IRQ1/CAP3 使能位

0 = 禁止

1 = 使能

Bit 0 **IRQ0IE/ CAP2IE**: IRQ0/CAP2 使能位

0 = 禁止

1 = 使能

注意：

IRQ5、IRQ4 分别和 CAP5、CAP4 共享中断控制位

IRQ1、IRQ0 分别和 CAP3、CAP2 共享中断控制位

6.2.3 中断标志寄存器 1 (P_INT_Flag1, $0E) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
CAP1IF	CAP0IF	T5OIF	T4OIF	T3OIF	T2OIF	T1OIF	T0OIF
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 **CAP1IF**:捕获 1 中断标志

0 =中断没有发生

1 =中断发生

Bit 6 **CAP0IF**: 捕获 0 中断标志

0 =中断没有发生

1 =中断发生

Bit 5 **T5OIF**: 定时/计数器 5 溢出中断标志

0 =中断没有发生

1 =中断发生

Bit 4 **T4OIF**: 定时/计数器 4 溢出中断标志

0 =中断没有发生

1 =中断发生

Bit 3 **T3OIF**: 定时/计数器 3 溢出中断标志

0 =中断没有发生

1 =中断发生

Bit 2 **T2OIF**: 定时/计数器 2 溢出中断标志

0 =中断没有发生

1 =中断发生

Bit 1 **T1OIF**: 定时/计数器 1 溢出中断标志

0 =中断没有发生

1 =中断发生

Bit 0 **T0OIF**: 定时/计数器 0 溢出中断标志

0 =中断没有发生

1 =中断发生

注意：向相应位写“1”清除该标志。

6.2.4 中断控制寄存器 1 (P_INT_Ctrl1, $0F) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
CAP1IE	CAP0IE	T5OIE	T4OIE	T3OIE	T2OIE	T1OIE	T0OIE
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 **CAP1IE**: 捕获 1 中断使能位

0 = 禁止

1 = 使能

Bit 6 **CAP0IE**: 捕获 0 中断使能位

0 = 禁止

1 = 使能

Bit 5 **T5OIE**: 定时/计数器 5 溢出中断使能位

0 = 禁止

1 = 使能

Bit 4 **T4OIE**: 定时/计数器 4 溢出中断使能位

0 = 禁止

1 = 使能

Bit 3 **T3OIE**: 定时/计数器 3 溢出中断使能位

0 = 禁止

1 = 使能

Bit 2 **T2OIE**: 定时/计数器 2 溢出中断使能位

0 = 禁止

1 = 使能

Bit 1 **T1OIE**: 定时/计数器 1 溢出中断使能位

0 = 禁止

1 = 使能

Bit 0 **T0OIE**: 定时/计数器 0 溢出中断使能位

0 = 禁止

1 = 使能

6.2.5 中断标志寄存器 2 (P_INT_Flag2, $26) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	-	ITVALIF	IIC _IF	UARTIF	SPIIF	CMP1IF	CMP0IF
-	-	R/W	R	R	R	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:6] 保留

Bit 5 **ITVALIF**: 时基中断标志

0 =中断没有发生

1 =中断发生

Bit 4 IIC _IF:IIC 中断标志

0 = 中断没有发生

1 = 中断发生

Bit 3 **UARTIF**: UART 中断标志

0 =中断没有发生

1 =中断发生

Bit 2 **SPIIF**: SPI 中断标志

0 =中断没有发生

1 =中断发生

Bit 1 **CMP1IF**: 比较器 1 中断标志

0 =中断没有发生

1 =中断发生

Bit 0 **CMP0IF**: 比较器 0 中断标志

0 =中断没有发生

1 =中断发生

注意：向相应位写“1”清除该标志。但 SPIIF 标志不能清除，它要通过寄存器 P_SPI_Status 中的 SPIIF 位进行清除。

6.2.6 中断控制寄存器 2 (P_INT_Ctrl2, $27) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

-	-	ITVALIE	-	-	-	CMP1IE	CMP0IE
-	-	R/W	-	-	-	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:6] 保留

Bit 5 **ITVALIE**: 时基中断使能位

0 = 禁止

1 = 使能

Bit [4:2] 保留

Bit 1 **CMP1IE**: 比较器 1 使能位

0 = 禁止

1 = 使能

Bit 0 **CMP0IE**: 比较器 0 使能位

0 = 禁止

1 = 使能

6.2.7 IRQ 和 Capture 设置寄存器 0 (P_IRQ_Opt0, $33) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	-	-	-	IRQ5ES/ CAP5ES	IRQM5	IRQ4ES/ CAP4ES	IRQM4
-	-	-	-	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:4] 保留

Bit 3 **IRQ5ES/CAP5ES**: 外部中断 5 极性控制位或捕获器 5 的计数触发沿极性选择位，其中 IRQ5ES 需要和 Bit2（IRQM5）共同设置才能决定外部中断 5 的触发极性，见 Bit2 描述。

CAP5ES＝1：下降沿触发计数（下降沿清除计数器）

CAP5ES＝0：上升沿触发计数（上升沿清除计数器）

Bit 2 **IRQM5**: 外部中断 5 触发模式选择位

1=电平触发

0=边沿触发

	IRQM5=1 电平触发	IRQM5=0 边沿触发
RQ5ES＝1	高电平触发	上升沿触发
IRQ5ES＝0	低电平触发	下降沿触发

Bit 1 IRQ4ES/CAP4ES:外部中断 4 极性控制位或捕获器 4 的计数触发沿极性选择位，其中 IRQ4ES 需要和 Bit0（IRQM4）共同设置才能决定外部中断 4 的触发极性，见 Bit0 描述。

CAP4ES＝1：下降沿触发计数（下降沿清除计数器）

CAP4ES＝0：上升沿触发计数（上升沿清除计数器）

Bit 0 IRQM4: 外部中断 4 触发模式选择位

1=电平触发

0=边沿触发

	IRQM4=1 电平触发	IRQM4=0 边沿触发
RQ4ES＝1	高电平触发	上升沿触发
IRQ4ES＝0	低电平触发	下降沿触发

注意：以上所有位都需要写两次才能设置成功

6.2.8 IRQ 和 Capture 设置寄存器 1 (P_IRQ_Opt1, $34) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IRQ3ES	IRQM3	IRQ2ES	IRQM2	IRQ1ES/ CAP3ES	IRQM1	IRQ0ES/ CAP2ES	IRQM0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 **IRQ3ES**: 外部中断 3 极性控制位

	IRQM3=1 电平触发	IRQM3=0 边沿触发
IRQ3ES＝1	高电平触发	上升沿触发
IRQ3ES＝0	低电平触发	下降沿触发

Bit 6 **IRQM3**: 外部中断 3 功能选择位

0 = 沿触发功能选择位

1 = 电平触发

Bit 5 **IRQ2ES**:外部中断 2 极性控制位

	IRQM2=1 电平触发	IRQM2=0 边沿触发
IRQ2ES＝1	高电平触发	上升沿触发
IRQ2ES＝0	低电平触发	下降沿触发

Bit 4 **IRQM2**: 外部中断 2 触发模式选择位

0 = 边沿触发

1 = 电平触发

Bit 3 **IRQ1ES/CAP3ES**: 外部中断 1 极性控制位或捕获器 3 的计数触发沿极性选择位，其中 IRQ1ES 需要和 Bit2（IRQM1）共同设置才能决定外部中断 1 的触发极性，见 Bit2 描述。

CAP3ES＝1：下降沿触发计数（下降沿清除计数器）

CAP3ES＝0：上升沿触发计数（上升沿清除计数器)

Bit 2 **IRQM1**: 外部中断 1 触发模式选择位

1 = 电平触发

0 = 边沿触发

	IRQM1=1 电平触发	IRQM1=0 边沿触发
IRQ1ES＝1	高电平触发	上升沿触发
IRQ1ES＝0	低电平触发	下降沿触发

Bit 1 **IRQ0ES/CAP2ES**:外部中断 0 极性控制位或捕获器 2 的计数触发沿极性选择位，其中 IRQ0ES 需要和 Bit0（IRQM0）共同设置才能决定外部中断 0 的触发极性，见 Bit0 描述。

CAP2ES＝1：下降沿触发计数（下降沿清除计数器）

CAP2ES＝0：上升沿触发计数（上升沿清除计数器)

Bit 0 **IRQM0**: 外部中断 0 触发模式选择位

1 = 电平触发

0 = 边沿触发

	IRQM0=1 电平触发	IRQM0=0 边沿触发
RQ0ES＝1	高电平触发	上升沿触发
IRQ0ES＝0	低电平触发	下降沿触发

注意：以上所有位都需要写两次才能设置成功

6.3 操作

1. 在 Fortis IDE’s 的屏蔽选项中选择 NMI 中断源或者普通外部中断源。
2. SEI ”指令关闭总的中断开关
3. 向中断控制寄存器的相应位写入“1”使能中断
4. 用“CLI”指令打开总的中断开关
5. 等待中断产生

发生中断时，系统自动将程序返回地址和状态寄存器（P）压入堆栈。如果用户还希望保存关键的寄存器（例如：累加器 A、X 寄存器、Y 寄存器），则需要软件完成。

进入中断服务程序，通过查询中断标志位便可以知道是哪个中断源发生了中断。中断在重新使能前必须由软件清除以避免中断程序的重复调用。执行中断返回指令[RTI]中断返回。

【例 6 1】: Enable IRQ1 interrupt

```
    lda     #$FF
    sta     P_INT_Flag0         ; 清除中断请求标志
    lda     #C_INT_IRQ1E        ; 使能 IRQ1 INT (INT1).
    sta     P_INT_Ctrl0
    cli                         ; 使能 INT
```

6.4 设计参考

【例 6 2】：下面以 SPMC65P2404A 为例，使能 IRQ0 中断，上升沿触发。具体步骤如下：

1、在 Fortis IDE 中，操作步骤如下：将[Fortis à Project à Setting à Mask option à NMI Source Selection]设置成“Disabled”。

2、将 PB4 设置为上拉输入，作为 IRQ0 的中断输入

3、设置 IRQ0 为上升沿触发

4、使能 IRQ0 中断

```
.SYNTAX     6502                ; process standard 6502 addressing syntax
.LINKLIST                       ; generate linklist information
.SYMBOLS                        ; generate symbolic debug information

.INCLUDE        spmc65p2404a.inc

    .PAGE0                      ; define values in the range from $00 to $FF
;
    .DATA                       ; define data storage section
;
    .CODE
;=====================================
;=                                   =
;= Power on Reset Process - Main Program  =
;=                                   =
;=====================================
V_Reset:
    sei                             ; 关闭总的中断开关
```

```
    ldx     #C_STACK_BOTTOM     ; 将堆栈指针初始化为$00FF 位置上
    txs                         ; 将 X 寄存器的值传送到堆栈指针(SP)中
;
    lda     #$FF                ; 清除复位标志
    sta     P_SYS_Ctrl
    sta     P_SYS_Ctrl
;
    jsr     F_InitIOPort        ; 初始化 IO 端口
;
    lda     #$00
    sta     P_IOB_Attrib

    lda     #00010000B
    sta     P_IOB_Data

    lda     #11101111B
    sta     P_IOB_Dir           ; 将 PB4 设置为上拉输入，作为 IRQ0 的中断输入
;
    lda     #$00
    sta     P_IRQ_Opt1
    sta     P_IRQ_Opt1          ; IRQ0 为下降沿触发
;
    lda     #$FF                ; 清除中断请求标志
    sta     P_INT_Flag0
    sta     P_INT_Flag1
    lda     #C_INT_IRQ0E        ; 使能 IRQ0 中断
    sta     P_INT_Ctrl0
    cli                         ; 打开总的中断开关
;
L_Main:
    nop
    jmp     L_Main
;
;=====================================
;=                                   =
;=       IRQ Interrupt Service Routine       =
;=                                   =
```

```
;===================================
V_IRQ:
    pha                             ; 将累加器 A 的值压入堆栈
    txa                             ; 将 X 寄存器的值送入累加器 A 中
    pha                             ; 将累加器 A 的值压入堆栈(即 push X)
;
; Timer 1 溢出中断?
;
    lda     P_INT_Flag0
    and     #C_INT_T0OIF
    beq     L_exit_irq
;
    lda     #C_INT_T0OIF
    sta     P_INT_Flag0
;
    lda     P_IOA_Data
    eor     #$FF
    sta     P_IOA_Data              ; PA 口输出数据取反
;
L_exit_irq:
    pla                             ; 将堆栈内容弹出给 A
    tax                             ; 将累加器 A 的值送入 X 寄存器中
    pla                             ; 将堆栈内容弹出给 A
;
    rti

;===================================
;=                                  =
;=       Interrupt Vector Table     =
;=                                  =
;===================================
VECTOR:             .SECTION
    DW      V_NMI                   ; may download program emulated either
    DW      V_Reset                 ; in internal memory or external memory
    DW      V_IRQ                   ; dw define two bytes interrupt vector
;
;===================================
```

```
;=                                          =
;=        End Of Interrupt Vector Table     =
;=                                          =
;==========================================
    .END                                    ; end of program
```

【例 6 3】：以 SPMC65P2404A 为例，使能 NMI 中断，下降沿触发。具体步骤如下：

在 Fortis IDE 中，操作步骤如下：

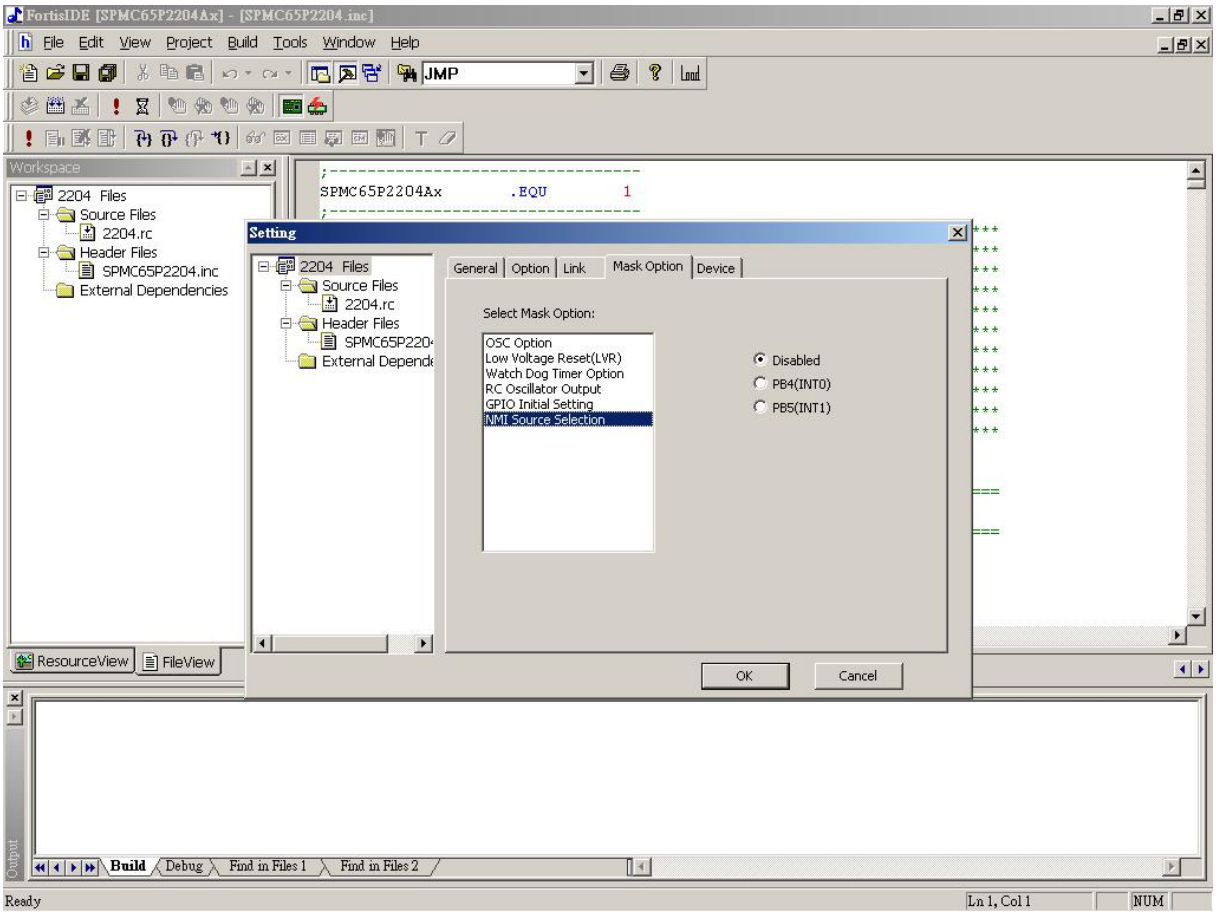
（1） 将[Fortis **à** Project **à** Setting **à** Mask option **à** NMI Source Selection]设置成“PB4(INT0)”。

（2） 将 PB4 设置为下拉输入，作为 IRQ0 的中断输入

（3） 设置 IRQ0 为下降沿触发

（4） 使能 IRQ0 中断

在下图的环境中进行 NMI 中断源的选择。



```
.SYNTAX     6502                ; process standard 6502 addressing syntax
.LINKLIST                       ; generate linklist information
.SYMBOLS                        ; generate symbolic debug information

.INCLUDE        spmc65p2404a.inc

    .PAGE0                      ; define values in the range from $00 to $FF
;
    .DATA                       ; define data storage section
;
    .CODE
;=====================================
;=                                   =
;= Power on Reset Process - Main Program  =
;=                                   =
;=====================================
V_Reset:
    sei                         ; 关闭总的中断开关
    ldx     #C_STACK_BOTTOM     ; 将堆栈指针初始化为$01FF 位置上
    txs                         ; 将 X 寄存器的值传送到堆栈指针(SP)中
;
    lda     #$FF                ; 清除复位标志
    sta     P_SYS_Ctrl
    sta     P_SYS_Ctrl
;
    jsr     F_InitIOPort        ; 初始化 IO 端口
;
    lda     #$00
    sta     P_IOB_Attrib
    sta     P_IOB_Data

    lda     #11101111B
    sta     P_IOB_Dir           ; PB4 设置为带下拉电阻输入，用于 IRQ0 中断
;
    lda     #C_IRQOpt1_IRQ0ES
    sta     P_IRQ_Opt1
    sta     P_IRQ_Opt1          ; 设置为上升沿触发
```

```
;
    lda     #$FF                ; 清除中断请求标志
    sta     P_INT_Flag0
    sta     P_INT_Flag1
    lda     #C_INT_IRQ0E        ; 使能 IRQ0 中断
    sta     P_INT_Ctrl0
    cli                         ; 打开总的中断开关
;
L_Main:
    nop
    jmp     L_Main
;
;====================================
;=                                  =
;= NMI Interrupt Service Routine        =
;=                                  =
;====================================
V_NMI:
    pha                         ; 将累加器 A 的值压入堆栈
    txa                         ; 将 X 寄存器的值送入累加器 A 中
    pha                         ; 将累加器 A 的值压入堆栈 (即 push X)
;
; IRQ0 中断 ?
;
    lda     P_INT_Flag0
    and     #C_INT_IRQ0F
    beq     L_exit_nmi
;
    lda     #C_INT_IRQ0F
    sta     P_INT_Flag0
;
    lda     P_IOA_Data
    eor     #$FF
    sta     P_IOA_Data          ; PA 口输出数据取反
;
L_exit_nmi:
    pla                         ; 将堆栈内容弹出给 A
```

```
    tax                             ; 将累加器 A 的值送入 X 寄存器中
    pla                             ; 将堆栈内容弹出给 A
;
    rti

;=====================================
;=                                   =
;=       Interrupt Vector Table      =
;=                                   =
;=====================================
VECTOR:             .SECTION
    DW      V_NMI               ; may download program emulated either
    DW      V_Reset             ; in internal memory or external memory
    DW      V_IRQ               ; dw define two bytes interrupt vector
;
;=====================================
;=                                   =
;=       End Of Interrupt Vector Table =
;=                                   =
;=====================================
    .END                            ; end of program
```

6.5 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题	**应用例系列号**
红外接收（IR）	AN_O0320.DOC
无线通讯（RF）	AN_O0321.DOC
过零检测	AN_O0322.DOC

库函数

标题	**应用例系列号**

SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

6.6 修订记录

版本号	日期	备注
V0.1	03/31/2005	第一版

7 时钟源

7.1 简述

SPMC65X 系列芯片支持三种时钟输入：晶体或者陶瓷振荡器、RC 振荡器和外部时钟输入。时钟输入经过分频作为系统时钟(F_{SYS})，提供给 CPU，供其正常工作。

注意：晶体或者陶瓷振荡频率和外部时钟输入频率（即 F_{OSC}）最大为 16Mhz，经过二分频作为系统频率 F_{SYS}，所以系统频率最大为 8MHz。

7.2 操作

通过对芯片配置寄存器的设置来选择不同类型的时钟源。在凌阳 Fortis IDE 环境下，如图 7-1 所示，可以运用如下步骤进行设置：[Project **à** Setting **à** Mask Option **à** OSC Option]。

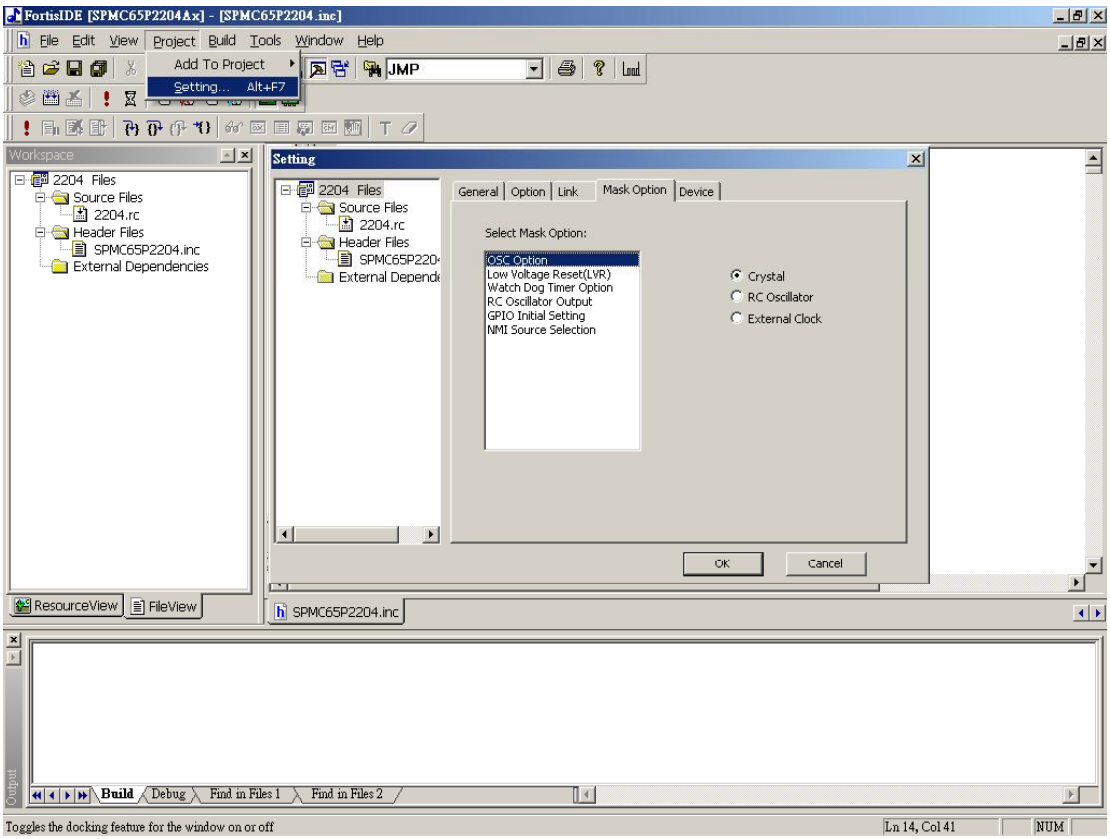


图 7-1 时钟源选择窗口

具体应用电路的设计应参考供应商提供的规格。典型的的 X'TAL/ROSC 电路应用如图 7-2 所示：

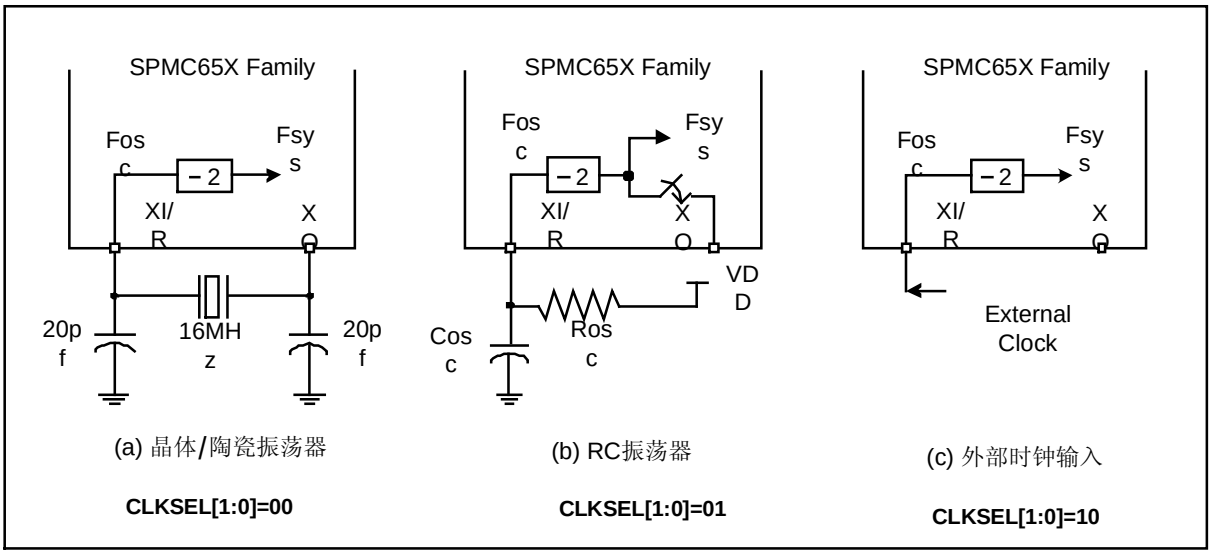


图 7-2 三种时钟源的典型应用电路

时钟模式如下：

a、晶体或陶瓷振荡器

b、RC 振荡器

c、外部时钟输入

在晶振或陶瓷振荡模式下

如图 7-1 所示，在 Fortis IDE 的[OSC option]中选择[Crystal]。

如图 7-2 (a).所示，在 XI 和 XO 管脚上连接一个振荡器和两个 20 pF 的电容。

如果振荡频率为 F_{OSC}，那么系统时钟为 F_{OSC} /2。比如，当 F_{OSC} = 16MHz，那么 F_{SYS} = 8MHz。

在 RC 振荡模式下：

如图 7-1 所示，在 Fortis IDE 的[OSC option]中选择[RC Oscillator]。

如图 7-2 (b)所示，在 XI 管脚上连接一个电阻和电容。

如图 7-3 所示，芯片配置寄存器可以这样设置：[Mask Option **à** RC Oscillator Output] as [Clock Output]。这样，XO 管脚可以输出运行时钟。RC 电阻和时钟频率的关系见表 7-1。

在外部时钟模式下：

如图 7-1 所示，在 Fortis IDE 的[OSC option]中选择[External Clock]。

如图 7-2 (c)所示，在 XI 管脚上连接一个外部时钟源。

如果外部时钟频率为 F_{OSC}，那么系统时钟频率(F_{SYS})为 F_{OSC} /2。

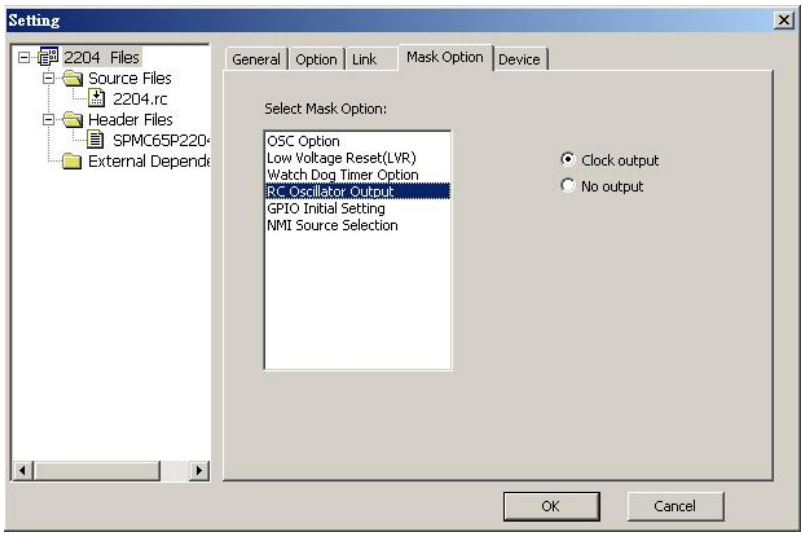


图 7-3 RC 振荡输出

Frequency (Hz)	Resistor (Ω), C=50PF
F_{SYS} = 135K	75k
F_{SYS} = 195K	51k
F_{SYS} = 480K	20k
F_{SYS} = 0.9M	10k
F_{SYS} = 1.6M	5.1k
F_{SYS} = 2.25M	3.3k

表 7-1 RC 电阻与振荡频率的对应关系

7.3 设计参考

时钟频率可能有±15%的差异，详见电气特性。

7.4 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题	应用例系列号
晶振类型选择	AN_O0330.DOC

库函数

标题	应用例系列号
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC

SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

7.5 修订记

版本号	日期	备注
V0.1	03/31/2005	第一版

8 IO 端口

8.1 简述

SPMC65X 系列芯片共有 6 个 IO 端口：端口 A、端口 B、端口 C、端口 D、端口 E 和端口 F。这些端口的各个管脚还可能有一些复用功能。在初始化状态时，都作为通用输入端口。

SPMC65X 提供了位控制结构的 I/O 端口，每一位都可以被单独定义用于输入或输出数据。通常，对某一位的设定包括以下 3 个基本项：数据向量 Data、属性向量 Attribution 和方向控制向量 Direction。3 个向量内每个对应的位组合在一起，形成一个控制字，用来定义相应 I/O 口位的输入输出状态和方式。8.3 节详细介绍了如何进行 IO 端口设置。

这三个基本项作用如下：
方向控制向量 Direction 将管脚设置为输入或输出
属性向量 Attribution 将管脚设置为悬浮或不悬浮
当管脚作为输入时，数据向量 Data 将其设置为上拉或下拉；当管脚作为输出时，往数据向量 Data 里面写入数值便可以将其输出。
控制寄存器的设置见表 8-1：
注意：N 极开漏输出(ODN)可通过设定(Dir Attrib Data)为 110 然后 100，或 100 然后 110 实现。
P 极开漏输出(ODP)可通过设定(Dir Attrib Data)为 111 然后 101，或 101 然后 111 实现。

表 8-1: IO 端口控制寄存器的设置

方向（Dir）	属性（Attrib）	数据（Data）	功能	描述
0	0	0	下拉	带下拉电阻的输入管脚
0	0	1	上拉	带上拉电阻的输入管脚
1	0	1	高电平输出	高电平输出
1	0	0	低电平输出	低电平输出
X	1	X	悬浮	悬浮式输入管脚
其它			保留	

每个 IO 端口都有 8 个可编程管脚，并且有相应的控制寄存器来进行设置。

这些控制寄存器为：数据寄存器 P_IOX_Data、方向寄存器 P_IOX_Dir、属性寄存器 P_IOX_Attrib 和数据锁存寄存器 P_IOX_Buf (X=A～F)。

P_IOX_Data 用于访问端口，读 P_IOX_Data 寄存器将得到管脚的电平状态，向 P_IOX_Data 写入数据，其值将会存入 P_IOX_Buf 中。P_IOX_Buf 是一个专门用于存储端口(PX)数据的寄存器，并用来对端口各管脚进行位操作，而不直接用 P_IOX_Data 寄存器来操作。
例如，当端口做为输出，向寄存器 P_IOX_Buf 或者 P_IOX_Data 中写数据时，效果相同。

但是，当使用位操作指令 SET, TST,CLR 或者 INV 对寄存器 P_IOX_Data 进行位操作时，寄存器 P_IOX_Buf 的值可能会发生错误。因此，强烈建议只对寄存器 P_IOX_ Buf (X=A,B,C,D,E,F)进行位操作。每个端口的 8 个管脚 PX[7:0]内部都有上拉/下拉电阻，可以通过寄存器设置为上拉或下拉。图 8-1 为典型的 IO 内部结构图，图 8-2 为 IDE 环境下对 IO 口的初始化设置选项。如果选择 “All Float”，上电复位
后 IO 口的状态为悬浮。如果选择 “All pull low”，则为下拉。

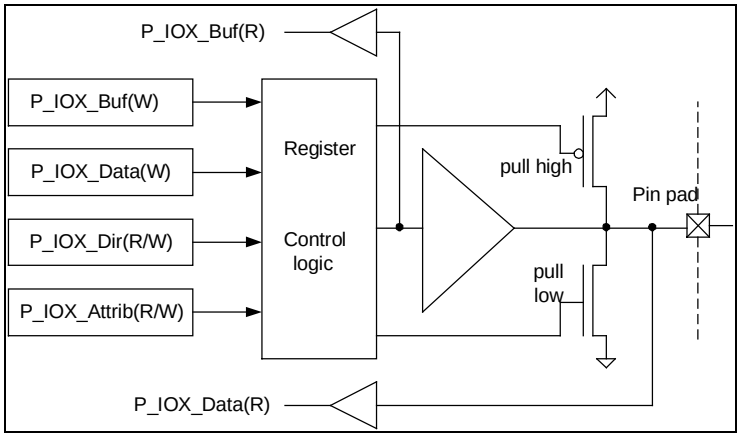


图 8-1 典型的 IO 内部结构图

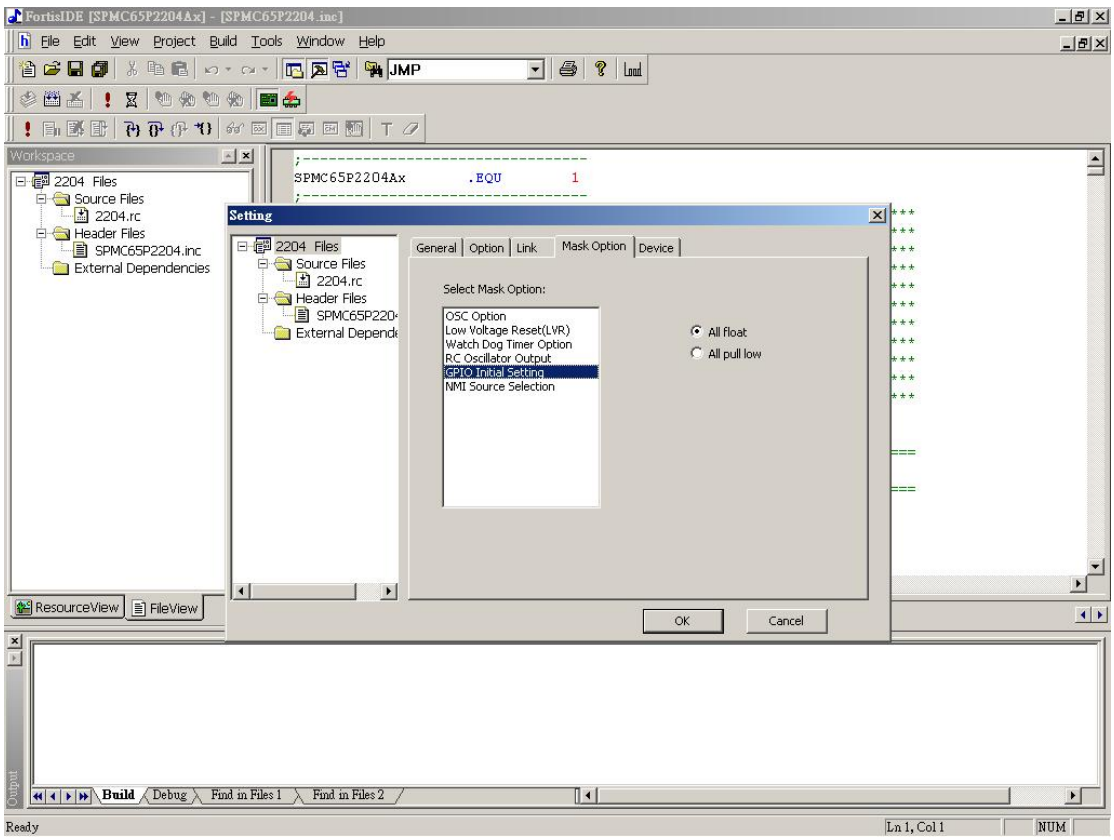


图 8-2 IO 口的初始化设置

8.1.1 端口 A

端口 A 的控制寄存器为：数据寄存器 P_IOA_Data、方向寄存器 P_IOA_Dir、属性寄存器

P_IOA_Attrib 和数据锁存寄存器 P_IOA_Buf。除了作为通用的 IO 口之外，端口 A 还可以作为 AD 转换的输入通道

表 8-2: 端口 A 的特殊功能

管脚	Rp	输出	输出	特殊功能
PA7	100K 上拉/下拉	施密特触发器	4/10mA	AD7 模拟输入
PA6	100K 上拉/下拉	施密特触发器	4/10mA	AD6 模拟输入
PA5	100K 上拉/下拉	施密特触发器	4/10mA	AD5 模拟输入
PA4	100K 上拉/下拉	施密特触发器	4/10mA	AD4 模拟输入
PA3	100K 上拉/下拉	施密特触发器	4/10mA	AD3 模拟输入
PA2	100K 上拉/下拉	施密特触发器	4/10mA	AD2 模拟输入
PA1	100K 上拉/下拉	施密特触发器	4/10mA	AD1 模拟输入
PA0	100K 上拉/下拉	施密特触发器	4/10mA	AD0 模拟输入

注意：Rp 为上拉/下拉电阻。

8.1.2 端口 B

端口 B 的控制寄存器为：数据寄存器 P_IOB_Data、方向寄存器 P_IOB_Dir、属性寄存器 P_IOB_Attrib 和数据锁存寄存器 P_IOB_Buf。除了作为通用的 IO 口之外，端口 B 的每个管脚都有相应复用的特殊功能，例如：PB7 即可以作为 AD 转换的输入通道，也可以作为 AD 转换的外部参考电压输入；PB6 可以驱动蜂鸣片；PB[5:4]可以作为外部时钟输入或中断输入管脚；PB[3:2]可以作为比较输出或 PWM 输出管脚；PB[1:0]可以为捕获输入或外部时钟输入管脚。具体情况见表 9.3。

以慢速输出功能为例，PB[7:6]通常情况下作为普通 IO 口，但是如果将寄存器 P_IO_Opt ($35) 的 bit0 设置为 1，便会打开 PB[7:6]的慢速输出功能，当 PB[7:6]输出从高电平变为低电平时，将会延迟 250ns ± 20%，相当于 CPU 在 2MHz 的速度下，该管脚带有 50pf 的电容。

端口 B 各个管脚复用的特殊功能见表 8-3，详细描述见相应的章节。

表 8-3: 端口特殊功能 B 端口 B 的特殊功能

管脚	Rp	IN	OUT	特殊功能
PB7	100K 上拉/下拉	施密特触发器	4/20mA	AD8 模拟输入或 ADC 外部参考电压输入
PB6	100K 上拉/下拉	施密特触发器	4/20mA	蜂鸣器输出
PB5	100K 上拉/下拉	施密特触发器	4/10mA	外部中断 1/定时/计数器 3 外部时钟输入

PB4	100K 上拉/下拉	施密特触发器	4/10mA	外部中断 0/定时/计数器 2 外部时钟输入
PB3	100K 上拉/下拉	施密特触发器	4/10mA	Timer1 比较输出/PWM 输出
PB2	100K 上拉/下拉	施密特触发器	4/10mA	Timer0 比较输出/ PWM 输出
PB1	100K 上拉/下拉	施密特触发器	4/10mA	捕获器 1 输入/定时/计数器 1 外部时钟输入
PB0	100K 上拉/下拉	施密特触发器	4/10mA	捕获器 0 输入/定时/计数器 0 外部时钟输入

8.1.3 端口 C

端口 C 的控制寄存器为：数据寄存器 P_IOC_Data、方向寄存器 P_IOC_Dir、属性寄存器 P_IOC_Attrib 和数据锁存寄存器 P_IOC_Buf。除了作为通用的 IO 口之外，端口 C 的每个管脚都有相应复用的特殊功能，例如：PC[7:6]可以用作 IIC 总线、PC[5:4]可用于 UART 通讯接口、PC[3:0]用作 SPI 通讯接口。端口 C 各个管脚复用的特殊功能见表 8-4，详细书机描述见相应的章节。

表 8-4: 端口 C 的特殊功能

管脚	Rp	输入	输出	特殊功能
PC7	100K 上拉/下拉	施密特触发器	4/10mA	IIC 总线数据输入/输出
PC6	100K 上拉/下拉	施密特触发器	4/10mA	IIC 总线时钟输入
PC5	100K 上拉/下拉	施密特触发器	4/10mA	UART 接收信号
PC4	100K 上拉/下拉	施密特触发器	4/10mA	UART 发送信号
PC3	100K 上拉/下拉	施密特触发器	4/10mA	SPI 数据输出
PC2	100K 上拉/下拉	施密特触发器	4/10mA	SPI 数据输入
PC1	100K 上拉/下拉	施密特触发器	4/10mA	SPI 时钟输出/时钟输入
PC0	100K 上拉/下拉	施密特触发器	4/10mA	SPI 从机模式选择

8.1.4 端口 D

端口 D 的控制寄存器为：数据寄存器 P_IOD_Data、方向寄存器 P_IOD_Dir、属性寄存器 P_IOD_Attrib 和数据锁存寄存器 P_IOD_Buf。除了作为通用的 IO 口之外，端口 D 的每个管脚都有相应复用的特殊功能，例如：PD[7:6]可作为比较输出或 PWM 输出管脚、PD[5:4]可作为外部中断输入或外部时钟输入管脚、PD[3:2]可用于比较输出或 PWM 输出管脚、PD[1:0]可用作外部中断输入管脚。端口 D 各个管脚复用的特殊功能见表 8-5，详细描述见相应的章节。

表 8-5: 端口 D 特殊功能

PIN	Rp	IN	OUT	特殊功能

PD7	100K 上拉/下拉	施密特触发器	4/10mA	定时/计数器 5 比较输出/PWM 输出
PD6	100K 上拉/下拉	施密特触发器	4/10mA	定时/计数器 4 比较输出/PWM 输出
PD5	100K 上拉/下拉	施密特触发器	4/10mA	外部中断 5 输入/定时/计数器 5 外部时钟输入
PD4	100K 上拉/下拉	施密特触发器	4/10mA	外部中断 4 输入/定时/计数器 4 外部时钟输入
PD3	100K 上拉/下拉	施密特触发器	4/10mA	定时/计数器 2 比较输出/PWM 输出
PD2	100K 上拉/下拉	施密特触发器	4/10mA	定时/计数器 3 比较输出/PWM 输出
PD1	100K 上拉/下拉	施密特触发器	4/10mA	外部中断 3 输入
PD0	100K 上拉/下拉	施密特触发器	4/10mA	外部中断 2 输入

8.1.5 端口 E

端口 E 的控制寄存器为：数据寄存器 P_IOE_Data、方向寄存器 P_IOE_Dir、属性寄存器 P_IOE_Attrib 和数据锁存寄存器 P_IOE_Buf。除了作为通用的 IO 口之外，端口 E 有的管脚有相应复用的特殊功能，例如：PE6 可以作为 D/A 转换的输出管脚，PE[5:2]可以用于比较器功能。端口 E 各个管脚复用的特殊功能见表 8-6，详细描述见相应的章节。

表 8-6: 端口特殊功能 E

PIN	Rp	IN	OUT	特殊功能
PE7	100K 上拉/下拉	施密特触发器	4/10mA	
PE6	100K 上拉/下拉	施密特触发器	4/10mA	D/A 输出
PE5	100K 上拉/下拉	施密特触发器	4/10mA	比较器 1 输入
PE4	100K 上拉/下拉	施密特触发器	4/10mA	比较器 1 参考输入
PE3	100K 上拉/下拉	施密特触发器	4/10mA	比较器 0 输入
PE2	100K 上拉/下拉	施密特触发器	4/10mA	比较器 0 参考输入
PE1	100K 上拉/下拉	施密特触发器	4/10mA	
PE0	100K 上拉/下拉	施密特触发器	4/10mA	

8.1.6 端口 F

端口 F 的控制寄存器为：数据寄存器 P_IOF_Data、方向寄存器 P_IOF_Dir、属性寄存器 P_IOF_Attrib 和数据锁存寄存器 P_IOF_Buf。没有复用的特殊功能，管脚分配见表 8-7，详细描述见相应的章节。

表 8-7: 端口特殊功能 F

PIN	Rp	IN	OUT	特殊功能
PF7	100K 上拉/下拉	施密特触发器	4/10mA	

PF6	100K 上拉/下拉	施密特触发器	4/10mA	
PF5	100K 上拉/下拉	施密特触发器	4/10mA	
PF4	100K 上拉/下拉	施密特触发器	4/10mA	
PF3	100K 上拉/下拉	施密特触发器	4/10mA	
PF2	100K 上拉/下拉	施密特触发器	4/10mA	
PF1	100K 上拉/下拉	施密特触发器	4/10mA	
PF0	100K 上拉/下拉	施密特触发器	4/10mA	

8.1.7 慢速输出功能控制

SPMC65X 系列芯片支持慢速输出功能，PB[7:6]通常情况下作为普通 IO 口，但是如果将寄存器 P_IO_Opt ($35)的 bit0 设置为 1，便会打开 PB[7:6]的慢速输出功能。当 PB[7:6]输出从高电平变为低电平时，将会延迟 250ns 左右，具体延迟时间由系统时钟（F_{SYS}）决定。当 MCU 在进行远距离通信时，慢速输出功能可以防止产生电磁干扰。相关寄存器的详细设置和举例见后续章节。

8.2 端口控制寄存器

8.2.1 端口 A 数据寄存器 (P_IOA_Data、$00)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOA_Data							
R/W							
00h							

Bit [7:0] P_IOA_Data：端口 A 数据寄存器
读：读取端口 A 外部管脚上的电平状态值
写：写入数据到端口 A 的数据锁存器中($0059)

8.2.2 端口 A 方向寄存器 (P_IOA_Dir, $04)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOA_Dir							
R/W							
00h							

Bit [7:0] P_IOA_Dir：端口 A 方向寄存器
0 = 输入
1 = 输出

8.2.3 端口 A 属性寄存器 (P_IOA_Attrib, $08)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOA_Attrib							
R/W							
00h							

Bit [7:0] P_IOA_Attrib：端口 A 属性寄存器
0 = 不悬浮

1 = 悬浮式输入

8.2.4 端口 A 数据锁存器 (P_IOA_Buf, $59) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOA_Buf							
R/W							
00h							

Bit [7:0] P_IOA_Buf：端口 A 数据锁存器

注意：当使用位指令对该 IO 口进行操作时，必须对 P_IOA_Buf 来做操作，而不应该对 P_IOA_Data 进行操作，其它 IO 口与此相同。

8.2.5 端口 B 数据寄存器 (P_IOB_Data、$01)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOB_Data							
R/W							
00h							

Bit [7:0] P_IOB_Data：端口 B 数据寄存器

读：读取端口 B 外部管脚上的电平状态值

写：写入数据到端口 B 的数据锁存器中($005A)

8.2.6 端口 B 方向寄存器 (P_IOB_Dir, $05)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOB_Dir							
R/W							
00h							

Bit [7:0] P_IOB_Dir：端口 B 方向寄存器

0 = 输入

1 = 输出

8.2.7 端口 B 属性寄存器 (P_IOB_Attrib, $0009)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOB_Attrib							
R/W							
00h							

Bit [7:0] P_IOB_Attrib：端口 B 的属性寄存器

0 = 不悬浮

1 = 悬浮输入

8.2.8 端口 B 数据锁存器 (P_IOB_Buf, $5A) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOB_Buf							
R/W							
00h							

Bit [7:0] P_IOB_Buf：端口 B 数据锁存器

注意：当使用位指令对该 IO 口进行操作时，必须对 P_IOB_Buf 来做操作，而不应该对 P_IOB_Data 进行操作，其它 IO 口与此相同。

8.2.9 端口 C 数据寄存器 (P_IOC_Data、$02)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOC_Data							
R/W							
00h							

Bit [7:0] P_IOC_Data：端口 C 数据寄存器

读：读取端口 C 外部管脚上的电平状态值

写：写入数据到端口 C 的数据锁存器中($005B)

8.2.10 端口 C 方向寄存器(P_IOC_Dir, $06)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOC_Dir							
R/W							
00h							

Bit [7:0] P_IOC_Dir：端口 C 方向寄存器

0 = 输入

1 = 输出

8.2.11 端口 C 属性寄存器 (P_IOC_Attrib, $0A)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOC_Attrib							
R/W							
00h							

Bit [7:0] P_IOC_Attrib：端口 C 的属性寄存器

0 = 不悬浮

1 = 悬浮输入

8.2.12 端口 C 数据锁存器 (P_IOC_Buf, $5B) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOC_Buf							
R/W							
00h							

Bit [7:0] P_IOC_Buf：端口 C 数据锁存器

注意：当使用位指令对该 IO 口进行操作时，必须对 P_IOC_Buf 来做操作，而不应该对 P_IOC_Data 进行操作，其它 IO 口与此相同。

8.2.13 端口 D 数据寄存器 (P_IOD_Data、$03)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOD_Data							
R/W							
00h							

Bit [7:0] P_IOD_Data：端口 D 数据寄存器

读：读取端口 D 外部管脚上的电平状态值

写：写入数据到端口 D 的数据锁存器中($005C)

8.2.14 端口 D 方向寄存器 (P_IOD_Dir, $07)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOD_Dir							
R/W							
00h							

Bit [7:0] P_IOD_Dir：端口 D 方向寄存器

0 = 输入

1 = 输出

8.2.15 端口 D 属性寄存器 (P_IOD_Attrib, $0B)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOD_Attrib							
R/W							
00h							

Bit [7:0] P_IOD_Attrib：端口 D 属性寄存器

0 = 不悬浮

1 = 悬浮输入

8.2.16 端口 D 数据锁存器 (P_IOD_Buf, $5C) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOD_Buf							
R/W							
00h							

Bit [7:0] P_IOD_Buf：端口 D 数据锁存器

注意： 当使用位指令对该 IO 口进行操作时，必须对 P_IOD_Buf 来做操作，而不应该对 P_IOD_Data 进行操作，其它 IO 口与此相同。

8.2.17 端口 E 数据寄存器 (P_IOE_Data，$40) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOE_Data							
R/W							
00h							

Bit [7:0] P_IOE_Data：端口 E 数据寄存器
读：读取端口 E 外部管脚上的电平状态值
写：写入数据到端口 E 的数据锁存器中($005D)

8.2.18 端口 E 方向寄存器 (P_IOE_Dir, $42) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOE_Dir							
R/W							
00h							

Bit [7:0] P_IOE_Dir：端口 E 方向寄存器
0 = 输入
1 = 输出

8.2.19 端口 E 属性寄存器 (P_IOE_Attrib, $44h) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOE_Attrib							
R/W							
00h							

Bit [7:0] P_IOE_Attrib：端口 E 属性寄存器
0 = 不悬浮
1 = 悬浮输入

8.2.20 端口 E 数据锁存器 (P_IOE_Buf, $5D) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOE_Buf							
R/W							
00h							

Bit [7:0] P_IOE_Buf：端口 E 数据锁存器

注意： 当使用位指令对该 IO 口进行操作时，必须对 P_IOE_Buf 来做操作，而不应该对 P_IOE_Data 进行操作，其它 IO 口与此相同。

8.2.21 端口 F 数据寄存器 (P_IOF_Data、$41) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOF_Data							
R/W							
00h							

Bit [7:0] P_IOF_Data：端口 F 数据寄存器
读：读取端口 E 外部管脚上的电平状态值
写：写入数据到端口 F 的数据锁存器中($005E)

8.2.22 端口 F 方向寄存器 (P_IOF_Dir, $43) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOF_Dir							
R/W							
00h							

Bit [7:0] P_IOF_Dir：端口 F 方向寄存器

0 = 输入

1 = 输出

8.2.23 端口 F 属性寄存器 (P_IOF_Attrib, $45) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOF_Attrib							
R/W							
00h							

Bit [7:0] P_IOF_Attrib：端口 F 属性寄存器

0 = 不悬浮

1 = 悬浮输入

8.2.24 端口 F 数据锁存器 (P_IOF_Buf, $5E) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P_IOF_Buf							
R/W							
00h							

Bit [7:0] P_IOF_Buf：端口 F 数据锁存器

注意：当使用位指令对该 IO 口进行操作时，必须对 P_IOF_Buf 来做操作，而不应该对 P_IOF_Data 进行操作，其它 IO 口与此相同。

8.2.25 慢速输出功能控制寄存器 (P_IO_Opt, $35)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	-	-	-	-	-	-	SLOWE
-	-	-	-	-	-	-	R/W
0	0	0	0	0	0	0	0

注意：该寄存器必须连续写两次才能将值写入。

Bit [7:1] 保留

Bit 0 慢速输出功能使能位

1=PB[7：6]开启慢速输出功能

0=PB[7：6]关闭慢速输出功能

8.3 操作

(1) 确定端口 A~F 中的哪几位用于输入或输出，然后设置相应的方向寄存器（P_IOX_Dir）。

(2) 通过属性寄存器(P_IOX_Attrib)设置管脚属性，如是否悬浮。

(3) 数据寄存器的设置会影响管脚的初始值。当管脚作为输入时，数据寄存器可以将其设置为上拉或下拉；

(4) 当管脚作为输出时，往数据寄存器里面写入数值便可以将其输出。

(5) 注意除 PB[6:7]最大驱动电流为 20mA 外，其余每个管脚最大驱动电流为 10mA。

【例 8 1】:将端口 X(X=A~F)[7:0]设置为输出

```
lda     #$FF                ; 将$ FF 写入累加器 A
sta     P_IOX_Dir           ; 将累加器 A 的值送入 P_IOX_Dir
lda     #$00
sta     P_IOX_Attrib
sta     P_IOX_Data
```

【例 8 2】:将端口 X(X=A~F)[7:0]设置为下拉输入

```
lda     #$00                ; 将$00 送入累加器 A 中
sta     P_IOX_Dir           ; 将累加器 A 的值赋给 P_IOX_Dir
sta     P_IOX_Attrib
sta     P_IOX_Data;
```

【例 8 3】:将端口 X(X=A~F) [7:4]设置为上拉输入，将端口 X(X=A~F) [3:0]设置为输出低电平。

```
lda     #$0F                ; 将$0F 送入累加器 A 中
sta     P_IOX_Dir
lda     #$00
sta     P_IOX_Attrib
lda     #$F0
sta     P_IOX_Data
```

【例 8 4】:将端口 X(X=A~F) [7:0]设置为悬浮输入。

```
lda     #$FF                ; 将$FF 送入累加器 A 中
sta     P_IOX_Attrib
```

[例 8-5]: 将端口 Port X[3:2]置高（对寄存器 P_IOX_Buf 进行位操作而不是 P_IOX_Data）。

```
set     P_IOX_Buf, 3        ; 将 bit3 置高
set      P_IOX_Buf, 2       ; 将 bit2 置高
```

[例 8-6]: 将端口 Port X[3:2]置低（对寄存器 P_IOX_Buf 进行位操作而不是 P_IOX_Data）。

```
clr     P_IOX_Buf, 3        ; 将 bit3 置低
clr     P_IOX_Buf, 2        ; 将 bit2 置低
```

[例 8-7]: 将端口 Port X[2:1]的输出电平取反（对寄存器 P_IOX_Buf 进行位操作而不是 P_IOX_Data）。

```
    inv     P_IOX_Buf, 2        ; 将 bit2 取反
    inv     P_IOX_Buf, 1        ; 将 bit1 取反
```

8.4 设计参考

【例 8 5】 以 SPMC65P2404A 芯片为例，将将端口 A~D 设置为输入。

```
.SYNTAX     6502                ; process standard 6502 addressing syntax
.LINKLIST                       ; generate linklist information
.SYMBOLS                        ; generate symbolic debug information

.INCLUDE        spmc65p2404a.inc

    .PAGE0                      ; define values in the range from $00 to $FF
;
    .DATA                       ; define data storage section
;
    .CODE
;=====================================
;=                                   =
;= Power on Reset Process - Main Program  =
;=                                   =
;=====================================
V_Reset:
    sei                         ; 关闭总的中断开关
    ldx     #C_STACK_BOTTOM     ; 将堆栈指针初始化为$00FF 位置上
    txs                         ; 将 X 寄存器的值传送到堆栈指针(SP)中
;
    lda     #$FF                ; 清除复位标志
    sta     P_SYS_Ctrl
    sta     P_SYS_Ctrl
;
    ldx     #$00                    ; X = $00

; IO 口初始化
    lda     #$00
    sta     P_IOA_Dir           ; 端口 A 设置为输入
    sta     P_IOB_Dir           ; 端口 B 设置为输入
    sta     P_IOC_Dir           ; 端口 C 设置为输入
    sta     P_IOD_Dir           ; 端口 D 设置为输入

    sta     P_IOA_Attrib
```

```
    sta     P_IOB_Attrib
    sta     P_IOC_Attrib
    sta     P_IOD_Attrib
;
    lda     #$FF
    sta     P_IOA_Data          ; 设置 IOA 为上拉
    sta     P_IOB_Data          ; 设置 IOB 为上拉
    sta     P_IOC_Data          ; 设置 IOC 为上拉
    sta     P_IOD_Data          ; 设置 IOD 为上拉
;
L_Main:
    nop
    jmp     L_Main
;
;=====================================
;=                                   =
;=       Interrupt Vector Table      =
;=                                   =
;=====================================
VECTOR:             .SECTION
    DW      V_NMI               ; may download program emulated either
    DW      V_Reset             ; in internal memory or external memory
    DW      V_IRQ               ; dw define two bytes interrupt vector
;
;=====================================
;=                                   =
;=       End Of Interrupt Vector Table =
;=                                   =
;=====================================
    .END                        ; end of program
```

8.5 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题	**应用例系列号**
按键(1 x 4)扫描	AN_O0301.DOC
拨码开关(x 4) + 按键(4 x 4)扫描	AN_O0304.DOC

7 段数码管(x 4)显示＋按键(4 x 4) 扫描	AN_O0306.DOC
7 段数码管(x 4)显示 + 拨码开关 (x 4) + 按键(4 x 4)扫描	AN_O0307.DOC
7 段数码管显示	AN_O0308.DOC
普通 I/O 口驱动 LCD 显示	AN_O0309.DOC
I/O 模拟 SPI 通讯	AN_O0310.DOC
I/O 模拟 IIC 通讯	AN_O0311.DOC
I/O 模拟 UART 通讯	AN_O0312.DOC
I/O 访问 EEPROM (93c46)	AN_O0313.DOC
I/O 访问 EEPROM (24c01)	AN_O0314.DOC
I/O 扩展	AN_O0315.DOC
I/O 模拟 A/D 转换	AN_O0316.DOC
I/O 模拟 D/A 转换	AN_O0317.DOC
I/O 模拟正弦波输出(R2R)	AN_O0318.DOC
I/O 模拟音乐输出	AN_O0319.DOC
I/O 设置介绍	AN_O0329.DOC
库函数	
标题	**应用例系列号**
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

8.6 修订记录

版本号	日期	备注
V0.1	03/31/2005	第一版

9 定时/计数器

9.1 简述

SPMC65X 系列芯片共有 6 个定时/计数器，每个定时/计数器都分别具备 CCP(Capture/Compare/PWM)功能，即捕获、比较和 PWM 输出功能。定时/计数器有 4 种工作模式：8 位/16 位定时/计数、捕获、比较输出和 PWM 输出。另外，除了 PWM 输出模式外，其它三种模式都可以以 8 位或 16 位的方式工作。所有功能通过相应控制寄存器进行设置，见下表简介：

表 9-1 SPMC65X 系列芯片定时/计数器功能简介

	定时/计数器		捕获器		比较器		PWM
	8 bit	16 bit	8 bit	16 bit	8 bit	16 bit	
Timer 0	是	是	脉宽/周期	脉宽	是	是	8 bit
Timer 1	是	是	脉宽/周期	脉宽	是	是	12 bit
Timer 2	是	是	脉宽/周期	脉宽	是	是	8 bit
Timer 3	是	是	脉宽/周期	脉宽	是	是	12 bit
Timer 4	是	是	脉宽/周期	脉宽	是	是	8 bit
Timer 5	是	是	脉宽/周期	脉宽/周期	是	是	16 bit

这 6 种定时器共有以下三种不同的组合，可以灵活选用具体介绍见下：

1. 组合 I：(定时器 0、定时器 2、定时器 4)
   8 位或 16 位定时/计数器
   8 位或 16 位捕获模式(8 位脉宽/周期测量，16 位脉宽测量)
   8 位或 16 位比较模式
   8 位 PWM 模式
2. 组合 II：(定时器 1、定时器 3)
   8 位或 16 位定时/计数器
   8 位或 16 位捕获模式(8 位脉宽/周期测量，16 位脉宽测量)
   8 位或 16 位比较模式
   12 位 PWM 模式
3. 组合 III：(定时器 5)
   8 位或 16 位定时/计数器
   8 位或 16 位捕获模式(8 位/16 位脉宽/周期测量)
   8 位或 16 位比较模式
   16 位 PWM 模式

这些功能通过控制寄存器 P_TMRx_Ctrl0 (x=0~5)设置，本章首先介绍了定时/计数器功能，其它功能在后续章节介绍。

9.2 控制寄存器

9.2.1 定时/计数器 0-1 控制寄存器 0 (P_TMR0_1_Ctrl0, $11) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T1FC2	T1FC1	T1FC0	-	T0FC2	T0FC1	T0FC0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留

Bit [6:4] **T1FC**[2：0]: 定时器 1 功能设置位

111 = 12 位 PWM

110 = 16 位脉宽捕获

101 = 16 位比较器

100 = 16 位定时/计数器

011 = 8 位脉宽/周期捕获

010 = 8 位比较器

001 = 8 位定时/计数器

000 = 禁止

Bit 3 保留

Bit [2:0] **T0FC**[2：0]: 定时器 0 功能设置位

111 = 8-bit PWM

110 = 16 位脉宽捕获

101 = 16 位比较器

100 = 16 位定时/计数器

011 = 8 脉宽/周期捕获

010 = 8 位比较器

001 = 8 位定时/计数器

000 = 禁止

9.2.2 定时/计数器 0-1 控制寄存器 1(P_TMR0_1_Ctrl1, $12) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T1PSS2	T1PSS1	T1PSS0	-	T0PSS2	T0PSS1	T0PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留

Bit [6:4] **T1PSS**[2：0]: 定时器 1 预分频设置位

111 = 外部时钟输入

110 = F_{SYS} ÷ 512

101 = F_{SYS} ÷ 128

100 = F_{SYS} ÷ 32

011 = F_{SYS} ÷ 8

010 = F_{SYS} ÷ 4

001 = F_{SYS} ÷ 2

000 = F_{SYS}

Bit 3　　保留

Bit [2:0]　**T0PSS**[2：0]: 定时器 0 预分频设置位

111 = 外部时钟输入

110 = F_{SYS} ÷ 512

101 = F_{SYS} ÷ 128

100 = F_{SYS} ÷ 32

011 = F_{SYS} ÷ 8

010 = F_{SYS} ÷ 4

001 = F_{SYS} ÷ 2

000 = F_{SYS}

F_{SYS}：系统时钟频率

9.2.3 定时/计数器 0 计数值的低字节(P_TMR0_Count, $13) (R)

9.2.4 定时/计数器 0 重载寄存器的低字节(P_TMR0_Preload, $13) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T0PL7	T0PL6	T0PL5	T0PL4	T0PL3	T0PL2	T0PL1	T0PL0
R	T0CN7	T0CN6	T0CN5	T0CN4	T0CN3	T0CN2	T0CN1	T0CN0
	0	0	0	0	0	0	0	0

Bit [7:0]　写：**T0PL** [7:0]　定时/计数器 0 重载值的低字节

Bit [7:0]　读：**T0CN** [7:0]　定时/计数器 0 计数值的低字节

注意：写入数据顺序要按照从高字节到低字节的顺序写。例如：要设置 16 位定时器模式先设置位于高字节的 P_TMR0_PreloadHi 然后设置位于低字节的 P_TMR0_Preload。

9.2.5 定时/计数器 0 计数值的高字节(P_TMR0_CountHi, $14) (R)

9.2.6 定时/计数器 0 重载寄存器高字节(P_TMR0_PreloadHi, $14) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T0PL15	T0PL14	T0PL13	T0PL12	T0PL11	T0PL10	T0PL9	T0PL8
R	T0CN15	T0CN14	T0CN13	T0CN12	T0CN11	T0CN10	T0CN9	T0CN8
	0	0	0	0	0	0	0	0

Bit [7:0]　写 **T0PL** [15:8]：定时/计数器 0 重载值的高字节

Bit [7:0]　读 **T0CN** [15:8]：定时/计数器 0 计数值的高字节

9.2.7 定时/计数器 1 计数值的低字节(P_TMR1_Count, $15) (R)

9.2.8 定时/计数器 1 重载寄存器低字节(P_TMR1_Preload, $15) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T1PL7	T1PL6	T1PL5	T1PL4	T1PL3	T1PL2	T1PL1	T1PL0
R	T1CN7	T1CN6	T1CN5	T1CN4	T1CN3	T1CN2	T1CN1	T1CN0
	0	0	0	0	0	0	0	0

Bit [7:0]　写：**T1PL** [7:0]　定时/计数器 1 重载值的低字节

Bit [7:0]　读：**T1CN** [7:0]　定时/计数器 1 计数值的低字节

注意：写入数据顺序要按照从高字节到低字节的顺序写。例如：要设置 16 位定时器模式先设置位于高字节的 P_TMR0_PreloadHi，然后设置位于低字节的 P_TMR0_Preload。

9.2.9 定时/计数器 1 计数值的高字节(P_TMR1_CountHi, $16) (R)

9.2.10 定时/计数器 1 重载寄存器高字节(P_TMR1_PreloadHi, $16) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T1PL15	T1PL14	T1PL13	T1PL12	T1PL11	T1PL10	T1PL9	T1PL8
R	T1CN15	T1CN14	T1CN13	T1CN12	T1CN11	T1CN10	T1CN9	T1CN8
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T1PL**[15:8] 定时/计数器 1 重载值的高字节

Bit [7:0] 读：**T1CN** [15:8] 定时/计数器 1 计数值的高字节

9.2.11 定时/计数器 2-3 控制器 0(P_TMR2_3_Ctrl0, $18) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T3FS2	T3FS1	T3FS0	-	T2FS2	T2FS1	T2FS0
	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留

Bit [6:4] **T3FS** [2:0]: 定时/计数器 3 功能设置位

111 = 12 位 PWM
110 = 16 位脉宽捕获
101 = 16 位比较器
100 = 16 位定时/计数器
011 = 8 脉宽/周期捕获
010 = 8 位比较器
001 = 8 位定时/计数器
000 = 禁止

Bit [3] 保留

Bit [2:0] **T2FS**[2:0]: 定时/计数器 2 功能设置位

111 = 8-位 PWM
110 = 16 位脉宽捕获
101 = 16 位比较器
100 = 16 位定时/计数器
011 = 8 脉宽/周期捕获
010 = 8 位比较器
001 = 8 位定时/计数器
000 = 禁止

9.2.12 定时/计数器 2-3 控制器 1(P_TMR2_3_Ctrl1, $19) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T3PSS2	T3PSS1	T3PSS0	-	T2PSS2	T2PSS1	T2PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7　　保留

Bit [6:4]　**T3PSS**[2：0]: 定时/计数器 3 预分频设置位

111 = 外部时钟输入

110 = F_{SYS} ÷ 512

101 = F_{SYS} ÷ 128

100 = F_{SYS} ÷ 32

011 = F_{SYS} ÷ 8

010 = F_{SYS} ÷ 4

001 = F_{SYS} ÷ 2

000 = F_{SYS}

Bit 3　　保留

Bit [2:0]　**T2PSS**[2：0]: 定时/计数器 2 预分频设置位

111 = 外部时钟输入

110 = F_{SYS} ÷ 512

101 = F_{SYS} ÷ 128

100 = F_{SYS} ÷ 32

011 = F_{SYS} ÷ 8

010 = F_{SYS} ÷ 4

001 = F_{SYS} ÷ 2

000 = F_{SYS}

F_{SYS}：系统时钟频率

9.2.13 定时/计数器 2 计数值的低字节(P_TMR2_Count, $1A) (R)

9.2.14 定时/计数器 2 重载寄存器低字节(P_TMR2_Preload, $1A) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T2PL7	T2PL6	T2PL5	T2PL4	T2PL3	T2PL2	T2PL1	T2PL0
R	T2CN7	T2CN6	T2CN5	T2CN4	T2CN3	T2CN2	T2CN1	T2CN0
	0	0	0	0	0	0	0	0

Bit [7:0]　写：**T2PL** [7:0]　定时/计数器 2 重载值的低字节

Bit [7:0]　读：**T2CN** [7:0]　定时/计数器 2 计数值的低字节

注意：写入数据顺序要按照从高字节到低字节的顺序写。例如：要设置 16 位定时器模式先设置位于高字节的 P_TMR2_PreloadHi，然后设置位于低字节的 P_TMR2_Preload。

9.2.15 定时/计数器 2 计数值的高字节(P_TMR2_CountHi, $1B) (R)

9.2.16 定时/计数器 2 重载寄存器高字节(P_TMR2_PreloadHi, $1B) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T2PL15	T2PL14	T2PL13	T2PL12	T2PL11	T2PL10	T2PL9	T2PL8
R	T2CN15	T2CN14	T2CN13	T2CN12	T2CN11	T2CN10	T2CN9	T2CN8
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T2PL** [15:8] 定时/计数器 2 重载值的高字节

Bit [7:0] 读：**T2CN** [15:8] 定时/计数器 2 计数值的高字节

9.2.17 定时/计数器 3 计数值的低字节(P_TMR3_Count, $1C) (R)

9.2.18 定时/计数器 3 重载寄存器低字节(P_TMR3_Preload, $1C) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T3PL7	T3PL6	T3PL5	T3PL4	T3PL3	T3PL2	T3PL1	T3PL0
R	T3CN7	T3CN6	T3CN5	T3CN4	T3CN3	T3CN2	T3CN1	T3CN0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T3PL** [7:0] 定时/计数器 3 重载值的低字节

Bit [7:0] 读：**T3CN** [7:0] 定时/计数器 3 计数值的低字节

注意：写入数据顺序要按照从高字节到低字节的顺序写。例如：要设置 16 位定时器模式先设置位于高字节的 P_TMR3_PreloadHi，然后设置位于低字节的 P_TMR3_Preload。

9.2.19 定时/计数器 3 计数值的高字节(P_TMR3_CountHi, $1D) (R)

9.2.20 定时/计数器 3 重载寄存器高字节(P_TMR3_PreloadHi, $1D) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T3PL15	T3PL14	T3PL13	T3PL12	T3PL11	T3PL10	T3PL9	T3PL8
R	T3CN15	T3CN14	T3CN13	T3CN12	T3CN11	T3CN10	T3CN9	T3CN8
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T3PL** [15:8] 定时/计数器 3 重载值的高字节

Bit [7:0] 读：**T3CN** [15:8] 定时/计数器 3 计数值的高字节

9.2.21 定时/计数器 4-5 控制器 0(P_TMR4_5_Ctrl0, $1F) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T5FS2	T5FS1	T5FS0	-	T4FS2	T4FS1	T4FS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留

Bit [6:4] **T5FS** [2:0]: 定时/计数器 5 功能设置位

111 = 16 位 PWM

110 = 16 位脉宽捕获

101 = 16 位比较器

100 = 16 位定时/计数器
011 = 8 脉宽/周期捕获
010 = 8 位比较器
001 = 8 位定时/计数器
000 = 禁止

Bit [3] 保留

Bit [2:0] **T4FS**[2：0]: 定时/计数器 4 功能设置位
111 = 8 位 PWM
110 = 16 位脉宽捕获
101 = 16 位比较器
100 = 16 位定时/计数器
011 = 8 脉宽/周期捕获
010 = 8 位比较器
001 = 8 位定时/计数器
000 = 禁止

9.2.22 定时/计数器 4-5 控制器 1(P_TMR4_5_Ctrl1, $20) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T5PSS2	T5PSS1	T5PSS0	-	T4PSS2	T4PSS1	T4PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留

Bit [6:4] **T5PSS**[2：0]: 定时/计数器 5 预分频设置位
111 = 外部时钟输入
110 = F_{SYS} ÷ 512
101 = F_{SYS} ÷ 128
100 = F_{SYS} ÷ 32
011 = F_{SYS} ÷ 8
010 = F_{SYS} ÷ 4
001 = F_{SYS} ÷ 2
000 = F_{SYS}

Bit 3 保留

Bit [2:0] **T4PSS**[2：0]: 定时/计数器 4 预分频设置位
111 = 外部时钟输入
110 = F_{SYS} ÷ 512
101 = F_{SYS} ÷ 128
100 = F_{SYS} ÷ 32
011 = F_{SYS} ÷ 8
010 = F_{SYS} ÷ 4
001 = F_{SYS} ÷ 2
000 = F_{SYS}
F_{SYS}：系统时钟频率

9.2.23 定时/计数器 4 计数值的低字节(P_TMR4_Count, $21) (R)

9.2.24 定时/计数器 4 重载寄存器低字节(P_TMR4_Preload, $21) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T4PL7	T4PL6	T4PL5	T4PL4	T4PL3	T4PL2	T4PL1	T4PL0
R	T4CN7	T4CN6	T4CN5	T4CN4	T4CN3	T4CN2	T4CN1	T4CN0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T4PL** [7:0] 定时/计数器 4 重载值的低字节

Bit [7:0] 读：**T4CN** [7:0] 定时/计数器 4 计数值的低字节

注意：写入数据顺序要按照从高字节到低字节的顺序写。例如：要设置 16 位定时器模式先设置位于高字节的 P_TMR4_PreloadHi，然后设置位于低字节的 P_TMR4_Preload。

9.2.25 定时/计数器 4 计数值的高字节(P_TMR4_CountHi, $22) (R)

9.2.26 定时/计数器 4 重载寄存器高字节(P_TMR4_PreloadHi, $22) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T4PL15	T4PL14	T4PL13	T4PL12	T4PL11	T4PL10	T4PL9	T4PL8
R	T4CN15	T4CN14	T4CN13	T4CN12	T4CN11	T4CN10	T4CN9	T4CN8
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T4PL** [15:8] 定时/计数器 4 重载值的高字节

Bit [7:0] 读：**T4CN** [15:8] 定时/计数器 4 计数值的高字节

9.2.27 定时/计数器 5 计数值的低字节(P_TMR5_Count, $23) (R)

9.2.28 定时/计数器 5 重载寄存器低字节(P_TMR5_Preload, $23) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T5PL7	T5PL6	T5PL5	T5PL4	T5PL3	T5PL2	T5PL1	T5PL0
R	T5CN7	T5CN6	T5CN5	T5CN4	T5CN3	T5CN2	T5CN1	T5CN0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T5PL** [7:0] 定时/计数器 5 重载值的低字节

Bit [7:0] 读：**T5CN** [7:0] 定时/计数器 5 计数值的低字节

注意：写入数据顺序要按照从高字节到低字节的顺序写。例如：要设置 16 位定时器模式先设置位于高字节的 P_TMR5_PreloadHi 然后设置位于低字节的 P_TMR5_Preload。

9.2.29 定时/计数器 5 计数值的高字节(P_TMR5_CountHi, $24) (R)

9.2.30 定时/计数器 5 重载寄存器高字节(P_TMR5_PreloadHi, $24) (W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T5PL15	T5PL14	T5PL13	T5PL12	T5PL11	T5PL10	T5PL9	T5PL8
R	T5CN15	T5CN14	T5CN13	T5CN12	T5CN11	T5CN10	T5CN9	T5CN8
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T5PL** [15:8] 定时/计数器 5 重载值的高字节

Bit [7:0]　　读：**T5CN** [15:8]　　定时/计数器 5 计数值的高字节

9.3 操作

SPMC65X 系列芯片最多有 6 个 8 位定时/计数器(Timer0~5)，都是可重载的。在作定时器时，需要设置相关寄存器，例如：定时器功能选择、8 位预分频寄存器和 16 位重载寄存器。

通过寄存器 8 位预分频寄存器的设置，定时器的时钟频率共有 8 种选择。其中 7 种时钟频率由系统时钟的 1～512 分频得到，因而频率范围非常广。8 位/16 位重载寄存器的作用是在定时/计数器开始工作或者溢出后重新赋初值。

定时器的 8 位/16 位计数器寄存器是可读的，当它每收到一个外部或内部时钟输入后加一，就这样不断地计数。以一个 8 位定时器为例，当其计数到 255 时，这时再来一个时钟信号，计数器便会溢出。此时，如果定时器的溢出中断使能被打开，便会产生溢出中断。同时，在计数器从 255 即将变为 0 时，重载寄存器中的值会被取出，载入计数器，重新从载入的初值开始计数。定时/计数器的计数操作见表 9-2。发生定时器溢出中断的详细步骤，见 9.4 章。如图 9-1 所示为定时/计数器结构图。

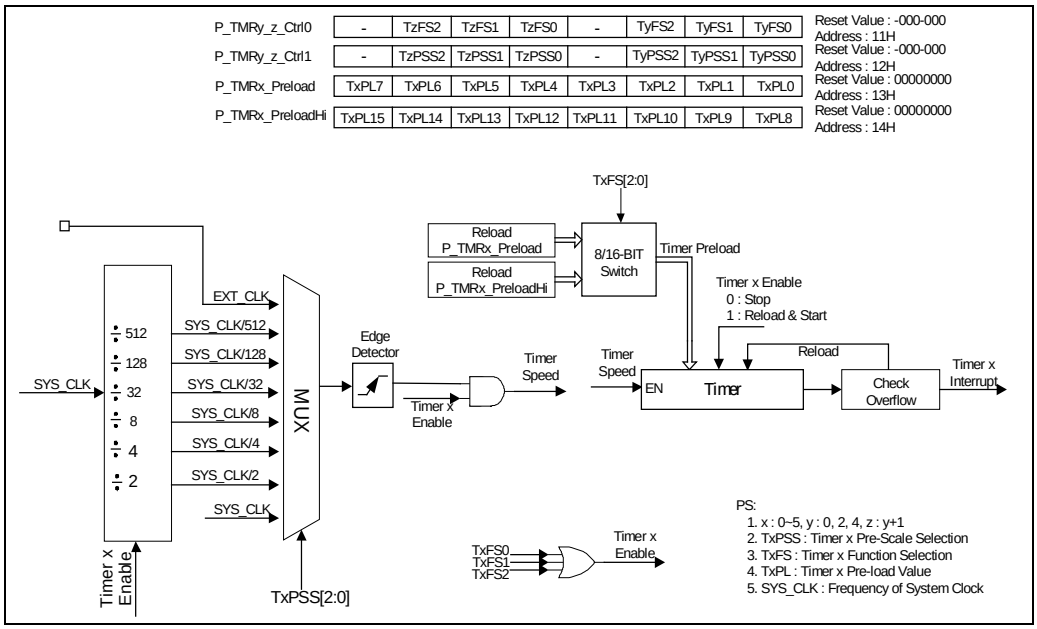


图 9-1 定时/计数器结构图

表 9-2 定时/计数器的计数操作

8 位定时/计数器数据 (Hex)	中断周期(* T_TMRx)		16 位定时/计数器(Hex)	中断周期(* T_TMRx)	
	Hex	Decimal		Hex	Decimal
$00	$100	256	$0000	$10000	65536
$01	$FF	255	$0001	$FFFF	65535
……	……	……	……	……	……
$FE	$02	2	$FFFE	$0002	2
$FF	$01	1	$FFFF	$0001	1

9.3.1 8 位定时/计数器

寄存器 P_TMRy_z_Ctrl0 (y=0, 2, 4; z=1, 3, 5)用于设置每个多功能定时/计数器的具体功能。每个 8 位定时器都有自己的计数器和控制器。每收到一个内部或外部时钟输入，计数器加一，当其计数到 255 时，这时再来一个时钟信号，计数器便会溢出。此时，如果定时器的溢出中断使能被打开，便会产生溢出中断。同时，在计数器从 255 即将变为 0 时，重载寄存器中的值会被取出，载入计数器，重新从载入的初值开始计数。通过公式 9-1 可以计算出 8 位定时器的定时长度。下面以一个 8 位定时/计数器——Timer0 为例分析。

8 位定时/计数器定时长度的计算见下面公式：

公式 9-1：

8 位定时/计数器

DATA = P_TMRx_Preload

T_TMRx = Timer x 时钟分频后的周期, x = 0~5

中断周期 = {$100 - DATA} * T_TMRx

以 8 位定时/计数器 0 为例分析，如图 9-2 所示为 8 位定时/计数器的内部结构。

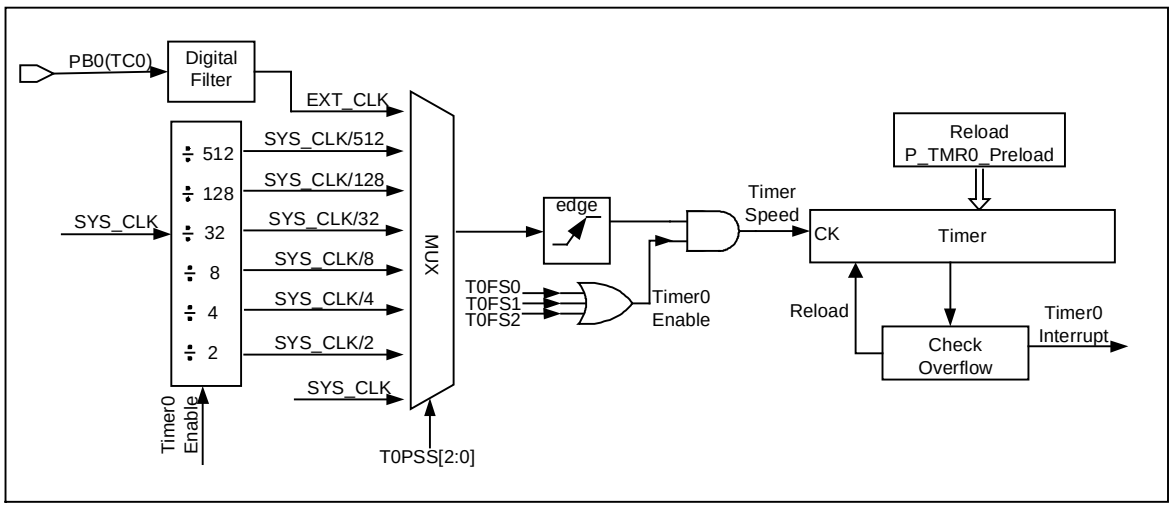


图 9-2 8 位定时/计数器内部结构

如图 9-3 所示为定时/计数器溢出和中断发生的时间

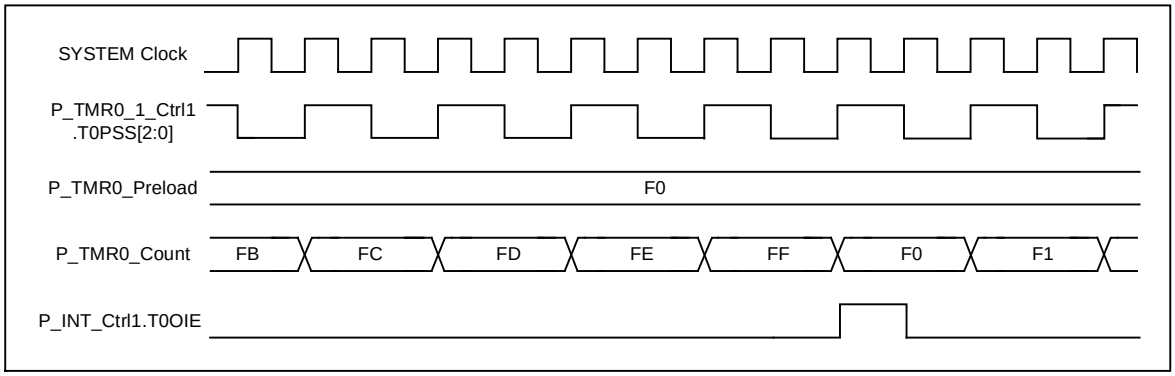


图 9-3 8 位定时/计数器溢出

如果 P_TMR0_1_Ctrl0.T0FS [2:0]=001 且 PB0 设置为输入，那么 PB0 就可以作为定时/计数器 0 的外部时钟输入端，进行外部脉冲计数。定时/计数器 0 的低字节寄存器 P_TMR0_Count ($0013)可以自动装入初值并负责计数，在其计数的过程中，可以从中读取当前的计数值。一旦计数溢出，如果该溢出中断使能，则会产生溢出中断，同时，计数寄存器会自动的被载入初值。如图 9-4 所示，为 8 位计数器计数情况和中断发生的关系。关于寄存器的详细设置见本章前半部分的描述。

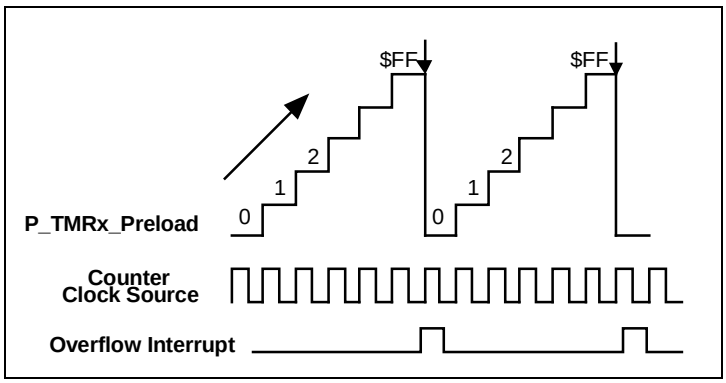


图 9-4 8 位计数器工作状态和中断之间的关系

【例 9 1】：将定时/计数器 0 设置成 8 位定时/计数器模式

```
    lda     #6                  ; 启动 Timer0 之前,首先设置 Timer0 的计数初值（重载值）
    sta     P_TMR0_Preload          ;
    lda     #C_T0FCS_Div_32     ; 时钟计数频率为 Fsys/32
    sta     P_TMR0_1_Ctrl1
    lda     #C_T08B_Timer       ; 设置 Timer0 为 8 位定时/计数器
    sta     P_TMR0_1_Ctrl0
    lda     #6                  ; 设置 Timer0 的重载值= 256-6= 250
    sta     P_TMR0_Preload       ; Fsys(8MHz)/32/250= 1KHz(1ms)
```

9.3.2 16 位定时/计数器

一个 8 位 CPU，总线宽度为 8 位，通常无法直接访问 16 位的数据。为了克服这样的局限，

在 SPMC65X 系列芯片中设计了一个数据寄存器 Timer x MSB，用于 16 位数据高 8 位的读写操作，16 位的读写操作如下图 9-5 所示，它有专门的读/写缓冲器。在读取 16 位数据时，用户需要先读出低字节，与此同时，高字节被锁存在缓冲器里，低字节读取结束，然后再读取高字节。直到 16 位数据全部读出，缓冲器的值才会改变。相反的，在写 16 位数据时，用户需要先写入高字节并被缓冲器自动保存，然后再写入低字节，直到 16 位数据全部写入，缓冲器的值才会改变，这样缓冲器中的高字节和低字节将会同时载入定时/计数器 0 中去。

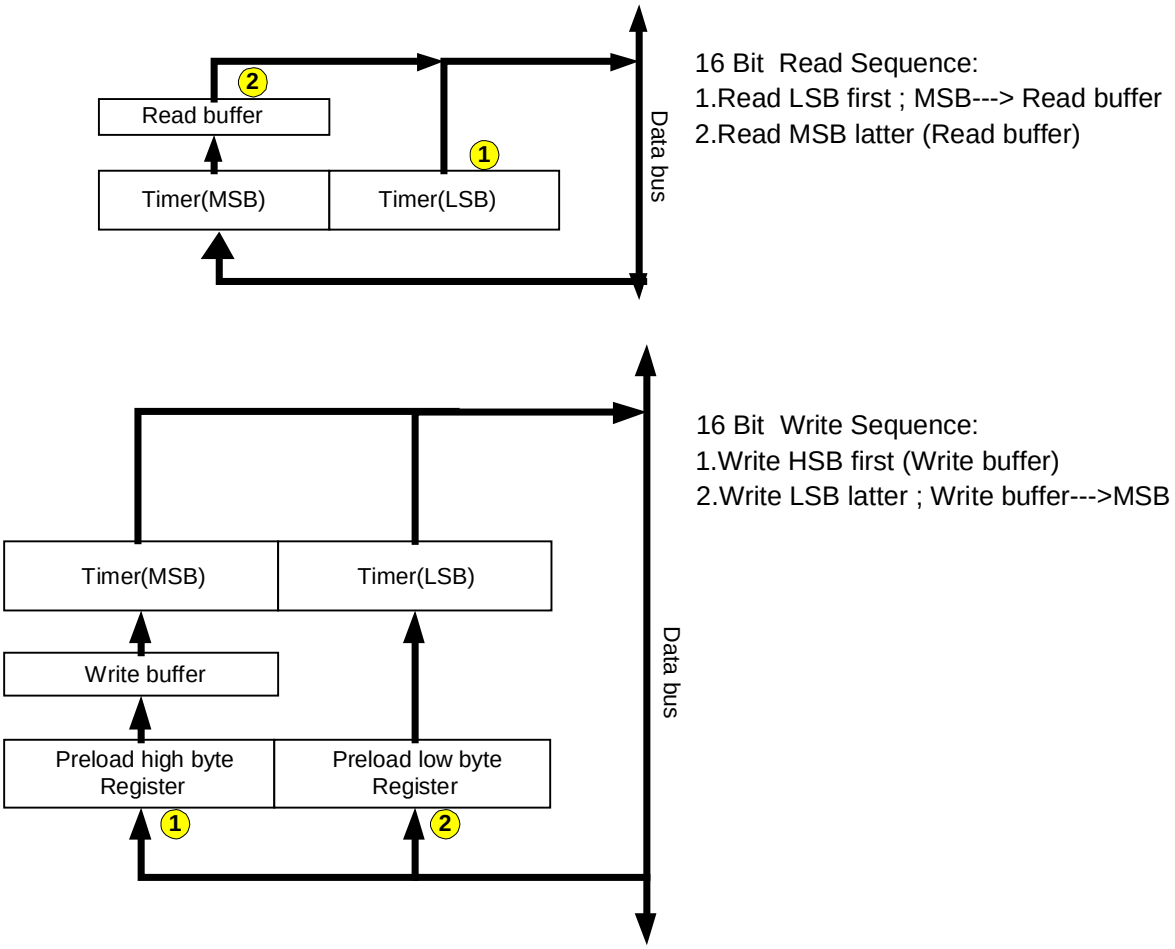


图 9-5 16 位定时/计数器读写顺序框图

中断周期的计算见公式 9-2

公式 9-2 16 位定时/计数器中断周期计算

16 位定时/计数器

DATA={ P_TMRx_PreloadHi, P_TMRx_Preload }

T_TMRx = Timer x 时钟分频后的周期, x = 0~5

中断周期 = {$10000 - DATA} * T_TMRx

下面以一个 16 位定时/计数器——Timer0 为例分析。它的详细内部结构见图 9-6。

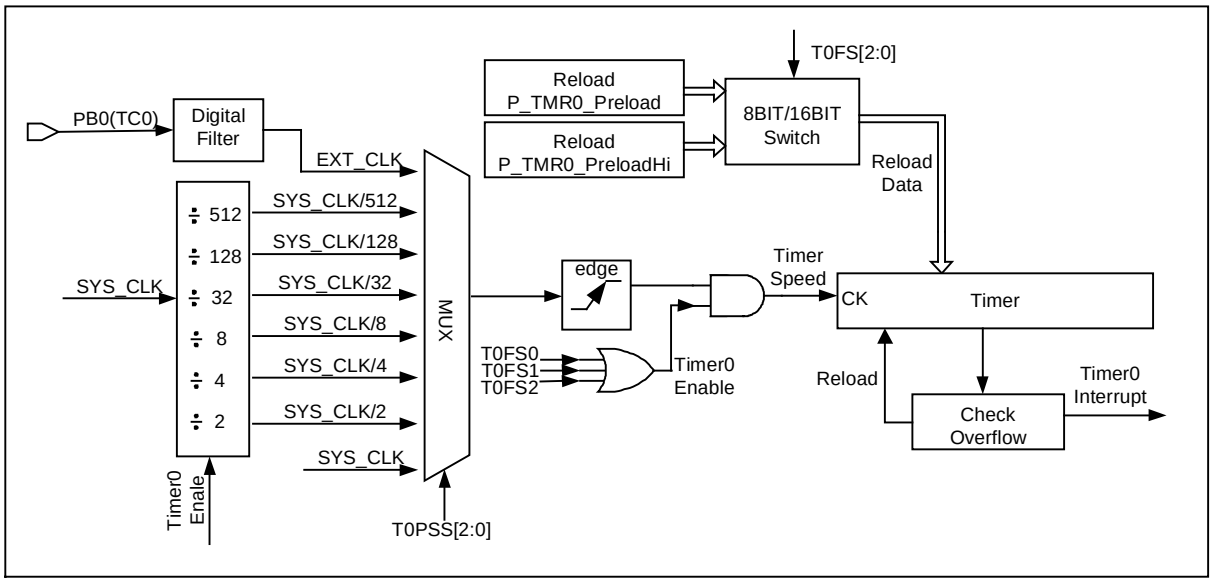


图 9-6 16 位定时器内部结构图

如图 9-7 所示为定时/计数器溢出和中断发生的时序

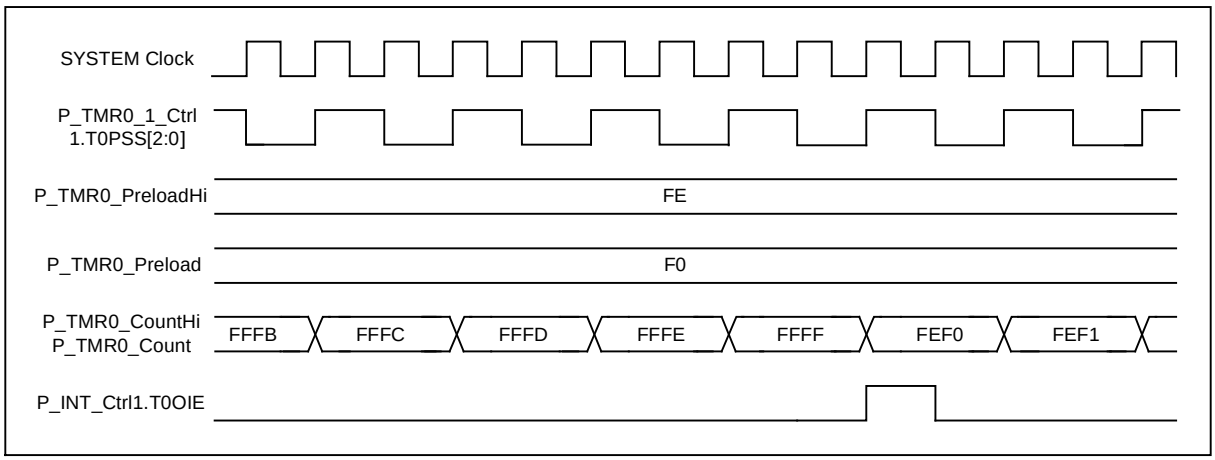


图 9-7 **16 位定时器溢出**

如果 P_TMR0_1_Ctrl0.T0FS [2:0]=100 且 PB0 设置为输入，那么 PB0 就可以作为定时/计数器 0 的时钟输入端，进行外部脉冲计数。定时/计数器 0 有两个计数寄存器：高字节寄存器 P_TMR0_CountHi ($0014)和低字节寄存器 P_TMR0_Count ($0013)，这两个计数器可以自动的载入初值进行计数，并且是可读的，在计数的过程中随时可以读取当前的计数值。当该 16 位定时/计数器 0 溢出时，如果溢出中断使能，则会产生溢出中断，同时定时/计数器 0 会自动重载初值继续计数。如图 9-8 所示，为 16 位计数器计数情况和中断发生的关系。关于寄存器的详细设置见本章前半部分的描述。

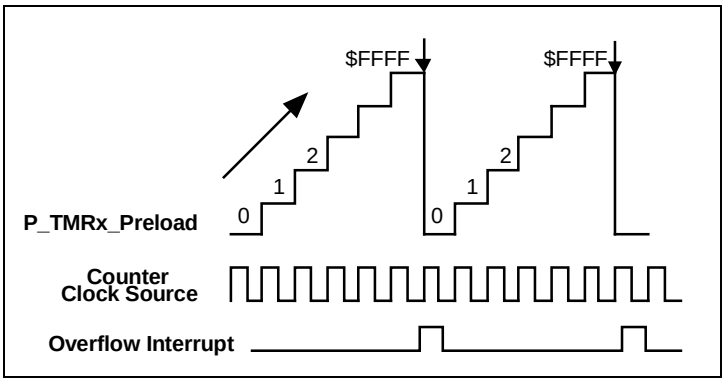


图 9-8 16 位计数器工作状态和中断之间的关系

【例 9 2】:将定时/计数器 0 设置为 16 位工作方式

```
    lda     #253                ; 启动 Timer0 之前，首先设置 Timer0 的计数初值
    sta     P_TMR0_PreloadHi    ; 首先设置高字节初始值（重载值）
    lda     #143
    sta     P_TMR0_Preload      ; 然后再设置低字节初始值（重载值）
    lda     #C_T0FCS_Div_128    ; 设定 Timer0 时钟频率为 Fsys/128
    sta     P_TMR0_1_Ctrl1
    lda     #C_T016B_Timer      ; 将 Timer0 设置为 16 位工作方式
    sta     P_TMR0_1_Ctrl0
    lda     #253                ; 首先设定计数重载值高字节为(256-254)x 256=512
    sta     P_TMR0_PreloadHi
    lda     #143                ; 然后设定计数重载值低字节为 256-143= 113。
    sta     P_TMR0_Preload      ; 设定 Fsys(8Mhz)/128/625= 100Hz(10ms)
```

9.4 定时/计数器中断

定时/计数器有以下五种中断源：

定时/计数器 0 溢出中断
定时/计数器 1 溢出中断
定时/计数器 2 溢出中断
定时/计数器 3 溢出中断
定时/计数器 4 溢出中断
定时/计数器 5 溢出中断

下面是设置定时/计数器溢出中断的几个步骤：

1、执行指令“SEI ”关闭总的中断开关；
2、决定采用 8 位还是 16 位的定时/计数方式并选择一个合适的定时/计数器；
3、设置所选的定时/寄存器的相关寄存器；
4、在寄存器 P_INT_Ctrl1($000F)中使能该定时器中断；
5、执行指令”CLI”打开总的中断开关；
6、设置完毕，等待中断产生。

【例 9 3】:将定时/计数器 0 设置为 8 位工作方式并产生 1ms 的溢出中断。

```
    lda     #6
    sta     P_TMR0_Preload          ; 启动 Timer0 之前，首先设置计数初值（重载值）
    lda     #C_T0FCS_Div_32         ; 设定 Timer0 时钟频率为 Fsys/32
    sta     P_TMR0_1_Ctrl1
    lda     #C_T08B_Timer           ; 将 Timer0 设置为 8 位工作方式
    sta     P_TMR0_1_Ctrl0

    lda     #6                      ; 设定 Timer0 计数重载值为 256-6= 250
    sta     P_TMR0_Preload          ; 设定 Fsys(8MHz)/32/250= 1KHz(1ms)
    lda     #$FF                    ; 清除所有的中断标志位
    sta     P_INT_Flag1
    set     P_INT_Ctrl1, CB_INT_T0OIE   ; 使能 Timer0 溢出中断
    cli                             ; 打开总的中断开关
```

【例 9 4】：将定时/计数器 1 设置为 16 位工作方式并产生 10ms 的周期溢出中断

```
    lda     #253                    ; 启动 Timer1 之前，首先设置计数初值（重载值）
    sta     P_TMR1_PreloadHi        ; 首先设置高字节初始值（重载值）
    lda     #143
    sta     P_TMR1_Preload          ; 然后再设置低字节初始值（重载值）
    lda     #C_T116B_Timer          ; 将 Timer1 设置为 16 位工作方式
    sta     P_TMR0_1_Ctrl0
    lda     #C_T1FCS_Div_128        ; 设定 Timer1 时钟频率为 Fsys/128
    sta     P_TMR0_1_Ctrl1
    lda     #253                    ; 首先设定计数重载值高字节为(256-254)x 256=512
    sta     P_TMR1_PreloadHi
    lda     #143                    ; 然后设定计数重载值低字节为 256-143= 113
    sta     P_TMR1_Preload          ; 设定 Fsys(8Mhz)/128/625= 100Hz(10ms)
    lda     #$FF                    ; 清除所有的中断标志位
    sta     P_INT_Flag1
    set     P_INT_Ctrl1, CB_INT_T1OIE       ; 使能 Timer1 溢出中断
    cli                             ; 打开总的中断开关
```

9.5 设计参考

【例 9 5】：以 SPMC65P2404A 为例

- 将 Timer1 设置为 16 位工作方式
- 设定 Timer1 时钟源为 Fsys/128
- 通过 Timer1 溢出产生周期 262msec 的中断

```
.SYNTAX     6502                ; process standard 6502 addressing syntax
.LINKLIST                       ; generate linklist information
```

```
.SYMBOLS                        ; generate symbolic debug information

.INCLUDE        spmc65p2404a.inc

    .PAGE0                      ; define values in the range from $00 to $FF
;
    .DATA                       ; define data storage section
;
    .CODE
;===================================
;=                                 =
;= Power on Reset Process - Main Program  =
;=                                 =
;===================================
V_Reset:
    sei                         ; 关闭总的中断开关
    ldx     #C_STACK_BOTTOM     ; 将堆栈指针初始化为$00FF 位置上
    txs                         ; 将 X 寄存器的值传送到堆栈指针(SP)中
;
    lda     #$FF                ; 清除复位标志
    sta     P_SYS_Ctrl
    sta     P_SYS_Ctrl
;
    jsr     F_InitIOPort        ; 初始化 IO 端口
;
    lda     #$C0                ; ($FFFF - $C000) x 128 / 8M ~= 262 ms
    sta     P_TMR1_PreloadHi
    lda     #$00
    sta     P_TMR1_Preload
;
    lda     #C_T1FCS_Div_128    ; 设置 Timer1 的时钟频率为 Fsys/128
    sta     P_TMR0_1_Ctrl1
    lda     #C_T116B_Timer      ; 设置 Timer1 为 16 位定时/计数器
    sta     P_TMR0_1_Ctrl0
;
    lda     #$FF                ; 清除中断请求标志
    sta     P_INT_Flag0
    sta     P_INT_Flag1
    lda     #C_INT_T1OIE        ; 使能 Timer1 溢出中断
    sta     P_INT_Ctrl1
    cli                         ; 打开总的中断开关
;
```

```
L_Main:
    nop
    jmp     L_Main
;
;====================================
;=                                  =
;=       IRQ Interrupt Service Routine      =
;=                                  =
;====================================
V_IRQ:
    pha                         ; 将累加器 A 的值压入堆栈
    txa                         ; 将 X 寄存器的值送入累加器 A 中
    pha                         ; 将累加器 A 的值压入堆栈(即 push X)
;
; Timer 1 溢出中断?
;
    lda     P_INT_Flag1
    and     #C_INT_T1OIF
    beq     L_exit_irq
;
    lda     #C_INT_T1OIF
    sta     P_INT_Flag1
;
    lda     P_IOA_Data
    eor     #$FF
    sta     P_IOA_Data          ; PA 口输出数据取反
;
L_exit_irq:
    pla                         ; 将堆栈内容弹出给 A
    tax                         ; 将累加器 A 的值送入 X 寄存器中
    pla                         ; 将堆栈内容弹出给 A
;
    rti

;====================================
;=                                  =
;=       Interrupt Vector Table          =
;=                                  =
;====================================
VECTOR:             .SECTION
    DW      V_NMI               ; may download program emulated either
    DW      V_Reset             ; in internal memory or external memory
```

```
    DW      V_IRQ                   ; dw define two bytes interrupt vector
;
;=====================================
;=                                   =
;=       End Of Interrupt Vector Table       =
;=                                   =
;=====================================
    .END                            ; end of program
```

9.6 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题	**应用例系列号**
LED (x 4) + 按键扫描(4 x 4)	AN_O0306.DOC
LED (x 4) + 拨码开关 (x 4) + 按键扫描\(4 x 4)	AN_O0307.DOC
7 段数码管显示	AN_O0308.DOC
普通 I/O 口驱动 LCD 显示	AN_O0309.DOC
I/O 模拟 SPI 通讯	AN_O0310.DOC
I/O 模拟 IIC 通讯	AN_O0311.DOC
I/O 模拟 UART 通讯	AN_O0312.DOC
I/O 访问 EEPROM (93c46)	AN_O0313.DOC
I/O 访问 EEPROM (24c01)	AN_O0314.DOC
I/O 模拟 A/D 转换	AN_O0316.DOC
I/O 模拟 D/A 转换	AN_O0317.DOC
I/O 模拟正弦波输出(R2R)	AN_O0318.DOC
I/O 模拟音乐输出	AN_O0319.DOC
定时/计数器 0 CCP 功能使用说明	AN_O0331.DOC
定时/计数器 1 CCP 功能使用说明	AN_O0332.DOC
定时/计数器 2 CCP 功能使用说明	AN_O0333.DOC
定时/计数器 3 CCP 功能使用说明	AN_O0334.DOC
定时/计数器 4 CCP 功能使用说明	AN_O0335.DOC

定时/计数器 5 CCP 功能使用说明	AN_O0336.DOC
库函数	
标题	**应用例系列号**
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

9.7 修订记录

版本号	日期	备注
V0.1	03/31/2005	第一版

10 捕获器

10.1 简述

SPMC65X 系列芯片共有 6 个多功能定时/计数器，并都具有 8 位/16 位捕获功能，支持两种触发模式：上升沿触发和下降沿触发；在 8 位捕获模式下，可以进行脉宽和周期的测量。在 16 位捕获模式下，只能进行脉宽测量，定时/计数器 5 除外。所有功能通过相应控制寄存器进行配置，见下表简介：

表 11.1 SPMC65X 系列芯片捕获功能简表

	定时器/计数器		捕获器		比较器		PWM
	8 bit	16 bit	8 bit	16 bit	8 bit	16 bit	
定时/计数器 0	YES	YES	脉宽/周期	脉宽	YES	YES	8 bit
定时/计数器 1	YES	YES	脉宽/周期	脉宽	YES	YES	12 bit
定时/计数器 2	YES	YES	脉宽/周期	脉宽	YES	YES	8 bit
定时/计数器 3	YES	YES	脉宽/周期	脉宽	YES	YES	12 bit
定时/计数器 4	YES	YES	脉宽/周期	脉宽	YES	YES	8 bit
定时/计数器 5	YES	YES	脉宽/周期	脉宽/周期	YES	YES	16 bit

这 6 种定时器共有以下三种不同的组合，可以实现不同的功能，具体介绍见下：

1. **type I（定时/计数器 0、2、4）**
   - 8 位或 16 位定时、计数功能
   - **8 位或 16 位捕获模式(8 位脉宽/周期测量，16 位脉宽测量)**
   - 8 位或 16 位比较输出
   - 8 位 PWM 输出

2. **type II（定时/计数器 1、3）**
   - 8 位或 16 位定时、计数功能
   - **8 位或 16 位捕获模式(8 位脉宽/周期测量，16 位脉宽测量)**
   - 8 位或 16 位比较输出
   - 12 位 PWM 输出

3. **组合 III (定时/计数器 5)**
   - 8 位或 16 位定时、计数功能
   - **8 位或 16 位捕获(8 位脉宽/周期测量，16 位脉宽测量)**
   - 8 位或 16 位比较输出
   - 16 位 PWM 输出

10.2 控制寄存器

10.2.1 定时/计数器 0-1 控制寄存器 0 (P_TMR0_1_Ctrl0, $11) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T1FC2	T1FC1	T1FC0	-	T0FC2	T0FC1	T0FC0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留

Bit [6:4] **T1FC**[2：0]: 定时/计数器 1 功能设置位

111 = 12 位 PWM
110 = 16 位脉宽捕获值
101 = 16 位比较器
100 = 16 位定时/计数器
011 = 8 位脉宽/周期捕获
010= 8 位比较器
001 = 8 位定时/计数器
000 = 禁止

Bit 3 保留

Bit [2:0] **T0FC**[2：0]: 定时/计数器 0 功能设置位

111 = 8 位 PWM
110 = 16 位脉宽捕获值
101 = 16 位比较器
100 = 16 位定时/计数器
011 = 8 位脉宽/周期捕获
010= 8 位比较器
001 = 8 位定时/计数器
000 = 禁止

10.2.2 定时/计数器 0-1 控制寄存器 1 (P_TMR0_1_Ctrl1, $12) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T1PSS2	T1PSS1	T1PSS0	-	T0PSS2	T0PSS1	T0PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留

Bit [6:4] **T1PSS**[2：0]: 定时/计数器 1 预分频设置位

111 = 外部时钟输入
110 = F_{SYS} ÷ 512
101 = F_{SYS} ÷ 128
100 = F_{SYS} ÷ 32
011 = F_{SYS} ÷ 8

010 = F_{SYS} ÷ 4
001 = F_{SYS} ÷ 2
000 = F_{SYS}

Bit 3 保留

Bit [2:0] **T0PSS**[2：0]: 定时/计数器 0 预分频设置位
111 = 外部时钟输入
110 = F_{SYS} ÷ 512
101 = F_{SYS} ÷ 128
100 = F_{SYS} ÷ 32
011 = F_{SYS} ÷ 8
010 = F_{SYS} ÷ 4
001 = F_{SYS} ÷ 2
000 = F_{SYS}
FSYS：系统时钟源

10.2.3 定时/计数器 0 脉宽捕获值的低字节(P_TMR0_Cap, $13) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T0PL7	T0PL6	T0PL5	T0PL4	T0PL3	T0PL2	T0PL1	T0PL0
R	T0CW7	T0CW6	T0CW5	T0CW4	T0CW3	T0CW2	T0CW1	T0CW0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：T0PL [7:0] 定时/计数器 0 重载值的低字节
Bit [7:0] 读：T0CW [7:0] 定时/计数器 0 脉宽捕获值的低字节

注意：写入数据顺序要按照从高字节到低字节的顺序写。例如：要设置 16 位定时器模式，先设置位于高字节的 P_TMR0_CapHi，然后设置位于低字节的 P_TMR0_ Cap。

10.2.4 定时/计数器 0 脉宽捕获值的高字节(P_TMR0_CapHi, $14) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T0PL15	T0PL14	T0PL13	T0PL12	T0PL11	T0PL10	T0PL9	T0PL8
R	T0CW15	T0CW14	T0CW13	T0CW12	T0CW11	T0CW10	T0CW9	T0CW8
	0	0	0	0	0	0	0	0

Bit [7:0] 写：T0PL [15:8] 定时/计数器 0 重载值的高字节
Bit [7:0] 读：T0CW [15:8] 定时/计数器 0 脉宽捕获值的高字节

10.2.5 定时/计数器 0 周期捕获值(P_TMR0_CapCycle8, $14) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T0PL15	T0PL14	T0PL13	T0PL12	T0PL11	T0PL10	T0PL9	T0PL8
R	T0CC7	T0CC6	T0CC5	T0CC4	T0CC3	T0CC2	T0CC1	T0CC0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：T0PL [15:8] 定时/计数器 0 重载值

Bit [7:0] 读：T0CC [7:0] 定时/计数器 0 周期捕获值

10.2.6 定时/计数器 1 脉宽捕获值的低字节(P_TMR1_Cap, $15) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T1PL7	T1PL6	T1PL5	T1PL4	T1PL3	T1PL2	T1PL1	T1PL0
R	T1CW7	T1CW6	T1CW5	T1CW4	T1CW3	T1CW2	T1CW1	T1CW0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：T1PL [15:8] 定时/计数器 1 重载值的低字节
Bit [7:0] 读：T1CW [7:0] 定时/计数器 1 脉宽捕获值的低字节

注意：写入数据顺序要按照从高字节到低字节的顺序写。例如：要设置 16 位定时器模式，先设置位于高字节的 P_TMR1_ CapHi，然后设置位于低字节的 P_TMR1_ Cap。

10.2.7 定时/计数器 1 脉宽捕获值的高字节(P_TMR1_CapHi, $16) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T1PL15	T1PL14	T1PL13	T1PL12	T1PL11	T1PL10	T1PL9	T1PL8
R	T1CW15	T1CW14	T1CW13	T1CW12	T1CW11	T1CW10	T1CW9	T1CW8
	0	0	0	0	0	0	0	0

Bit [7:0] 写：T1PL [15:8] 定时/计数器 1 重载值的高字节
Bit [7:0] 读：T1CW [15:8] 定时/计数器 1 脉宽捕获值的高字节

10.2.8 定时/计数器 1 周期捕获值(P_TMR1_CapCycle8, $16) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T1PL15	T1PL14	T1PL13	T1PL12	T1PL11	T1PL10	T1PL9	T1PL8
R	T1CC7	T1CC6	T1CC5	T1CC4	T1CC3	T1CC2	T1CC1	T1CC0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T1PL** [15:8] 定时/计数器 1 重载值
Bit [7:0] 读：**T1CC** [7:0] 定时/计数器 1 周期捕获值

10.2.9 定时/计数器 2-3 控制寄存器 0(P_TMR2_3_Ctrl0, $18) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T3FC2	T3FC1	T3FC0	-	T2FC2	T2FC1	T2FC0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留
Bit [6:4] **T3FC**[2：0]: 定时/计数器 3 预分频设置位
111 = 12 位 PWM
110 = 16 位脉宽捕获值

101 = 16 位比较器
100 = 16 位定时/计数器
011 = 8 位脉宽/周期捕获
010= 8 位比较器
001 = 8 位定时/计数器
000 = 禁止

Bit 3 保留

Bit [2:0] **T2FC**[2：0]: 定时/计数器 2 预分频设置位
111 = 8 位 PWM
110 = 16 位脉宽捕获值
101 = 16 位比较器
100 = 16 位定时/计数器
011 = 8 位脉宽/周期捕获
010= 8 位比较器
001 = 8 位定时/计数器
000 = 禁止
FSYS：系统时钟源

10.2.10 定时/计数器 2-3 控制寄存器 1(P_TMR2_3_Ctrl1, $19) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T3PSS2	T3PSS1	T3PSS0	-	T2PSS2	T2PSS1	T2PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留

Bit [6:4] **T3PSS**[2：0]: 定时/计数器 3 预分频设置位
111 = 外部时钟输入
110 = F_{SYS} ÷ 512
101 = F_{SYS} ÷ 128
100 = F_{SYS} ÷ 32
011 = F_{SYS} ÷ 8
010 = F_{SYS} ÷ 4
001 = F_{SYS} ÷ 2
000 = F_{SYS}

Bit 3 保留

Bit [2:0] **T2PSS**[2：0]: 定时/计数器 2 预分频设置位
111 = 外部时钟输入
110 = F_{SYS} ÷ 512
101 = F_{SYS} ÷ 128
100 = F_{SYS} ÷ 32
011 = F_{SYS} ÷ 8
010 = F_{SYS} ÷ 4

001 = F_{SYS} ÷ 2
000 = F_{SYS}
FSYS：系统时钟源

10.2.11 定时/计数器 2 脉宽捕获值的低字节(P_TMR2_Cap, $1A) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T2PL7	T2PL6	T2PL5	T2PL4	T2PL3	T2PL2	T2PL1	T2PL0
R	T2CW7	T2CW6	T2CW5	T2CW4	T2CW3	T2CW2	T2CW1	T2CW0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T2PL** [15:8] 定时/计数器 2 重载值的低字节
Bit [7:0] 读：**T2CW** [7:0] 定时/计数器 2 脉宽捕获值的低字节

注意：写入数据顺序要按照从高字节到低字节的顺序写。例如：要设置 16 位定时器模式先设置位于高字节的 P_TMR2_ CapHi 然后设置位于低字节的 P_TMR2_ Cap。

10.2.12 定时/计数器 2 脉宽捕获值的高字节(P_TMR2_CapHi, $1B) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T2PL15	T2PL14	T2PL13	T2PL12	T2PL11	T2PL10	T2PL9	T2PL8
R	T2CW15	T2CW14	T2CW13	T2CW12	T2CW11	T2CW10	T2CW9	T2CW8
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T2PL** [15:8] 定时/计数器 2 重载值的高字节
Bit [7:0] 读：**T2CW** [15:8] 定时/计数器 2 脉宽捕获值的高字节

10.2.13 定时/计数器 2 周期捕获值(P_TMR2_CapCycle8, $1B) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T2PL15	T2PL14	T2PL13	T2PL12	T2PL11	T2PL10	T2PL9	T2PL8
R	T2CC7	T2CC6	T2CC5	T2CC4	T2CC3	T2CC2	T2CC1	T2CC0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T2PL** [15:8] 定时/计数器 2 重载值的高字节
Bit [7:0] 读：**T2CC** [7:0] 定时/计数器 2 周期捕获值

10.2.14 定时/计数器 3 脉宽捕获值的低字节(P_TMR3_Cap, $1C) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T3PL7	T3PL6	T3PL5	T3PL4	T3PL3	T3PL2	T3PL1	T3PL0
R	T3CW7	T3CW6	T3CW5	T3CW4	T3CW3	T3CW2	T3CW1	T3CW0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：T3PL [7：0] 定时/计数器 3 重载值的低字节

Bit [7:0] 读：T3CW [7:0] 定时/计数器 3 脉宽捕获值的低字节

注意：写入数据顺序要按照从高字节到低字节的顺序写。例如：要设置 16 位定时器模式先设置位于高字节的 P_TMR3_ CapHi 然后设置位于低字节的 P_TMR3_ Cap。

10.2.15 定时/计数器 3 脉宽捕获值的高字节(P_TMR3_CapHi, $1D) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T3PL15	T3PL14	T3PL13	T3PL12	T3PL11	T3PL10	T3PL9	T3PL8
R	T3CW15	T3CW14	T3CW13	T3CW12	T3CW11	T3CW10	T3CW9	T3CW8
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T3PL** [15:8] 定时/计数器 3 重载值的高字节

Bit [7:0] 读：**T3CW** [15:8] 定时/计数器 3 脉宽捕获值的高字节

10.2.16 定时/计数器 3 周期捕获值(P_TMR3_CapCycle8, $1D) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T3PL15	T3PL14	T3PL13	T3PL12	T3PL11	T3PL10	T3PL9	T3PL8
R	T3CC7	T3CC6	T3CC5	T3CC4	T3CC3	T3CC2	T3CC1	T3CC0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T3PL** [15:8] 定时/计数器 3 重载值的高字节

Bit [7:0] 读：**T3CC** [7:0] 定时/计数器 3 周期捕获值

10.2.17 定时/计数器 4-5 控制器 0(P_TMR4_5_Ctrl0, $1F) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T5FC2	T5FC1	T5FC0	-	T4FC2	T4FC1	T4FC0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留

Bit [6:4] **T5FC**[2:0]: 定时/计数器 5 功能设置位

111 = 16 位 PWM

110 = 16 位脉宽/周期捕获值

101 = 16 位比较器

100 = 16 位定时/计数器

011 = 8 位脉宽/周期捕获

010= 8 位比较器

001 = 8 位定时/计数器

000 = 禁止

Bit 3 保留

Bit [2:0] **T4FC**[2：0]: 定时/计数器 4 功能设置位

111 = 8 位 PWM
110 = 16 位脉宽捕获值
101 = 16 位比较器
100 = 16 位定时/计数器
011 = 8 位脉宽/周期捕获
010= 8 位比较器
001 = 8 位定时/计数器
000 = 禁止
FSYS：系统时钟源

10.2.18 定时/计数器 4-5 控制器 1(P_TMR4_5_Ctrl1, $20) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T5PSS2	T5PSS1	T5PSS0	-	T4PSS2	T4PSS1	T4PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留

Bit [6:4] **T5PSS**[2：0]: 定时/计数器 5 预分频设置位
111 = 外部时钟输入
110 = F_{SYS} ÷ 512
101 = F_{SYS} ÷ 128
100 = F_{SYS} ÷ 32
011 = F_{SYS} ÷ 8
010 = F_{SYS} ÷ 4
001 = F_{SYS} ÷ 2
000 = F_{SYS}

Bit 3 保留

Bit [2:0] **T4PSS**[2：0]: 定时/计数器 4 预分频设置位
111 = 外部时钟输入
110 = F_{SYS} ÷ 512
101 = F_{SYS} ÷ 128
100 = F_{SYS} ÷ 32
011 = F_{SYS} ÷ 8
010 = F_{SYS} ÷ 4
001 = F_{SYS} ÷ 2
000 = F_{SYS}
FSYS：系统时钟源

10.2.19 定时/计数器 4 脉宽捕获值的低字节(P_TMR4_Cap, $21) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T4PL7	T4PL6	T4PL5	T4PL4	T4PL3	T4PL2	T4PL1	T4PL0
R	T4CW7	T4CW6	T4CW5	T4CW4	T4CW3	T4CW2	T4CW1	T4CW0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T4PL** [7：0] 定时/计数器 4 重载值的低字节 T3PL [15:8] (W)

Bit [7:0] 读：**T4CW** [7:0] 定时/计数器 4 脉宽捕获值的低字节 T3CW [7:0] (R)

注意：写入数据顺序要按照从高字节到低字节的顺序写。例如：要设置 16 位定时器模式先设置位于高字节的 P_TMR4_ CapHi 然后设置位于低字节的 P_TMR4_ Cap。

10.2.20 定时/计数器 4 脉宽捕获值的高字节(P_TMR4_CapHi, $22) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T4PL15	T4PL14	T4PL13	T4PL12	T4PL11	T4PL10	T4PL9	T4PL8
R	T4CW15	T4CW14	T4CW13	T4CW12	T4CW11	T4CW10	T4CW9	T4CW8
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T4PL** [15:8] 定时/计数器 4 重载值的高字节

Bit [7:0] 读：**T4CW** [15:8] 定时/计数器 4 脉宽捕获值的高字节

10.2.21 定时/计数器 4 周期捕获值(P_TMR4_CapCycle8, $22) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T4PL15	T4PL14	T4PL13	T4PL12	T4PL11	T4PL10	T4PL9	T4PL8
R	T4CC7	T4CC6	T4CC5	T4CC4	T4CC3	T4CC2	T4CC1	T4CC0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T4PL** [15:8] 定时/计数器 4 重载值的高字节

Bit [7:0] 读：**T4CC** [7:0] 定时/计数器 4 周期捕获值

10.2.22 定时/计数器 5 脉宽捕获值的低字节(P_TMR5_Cap, $23) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T5PL7	T5PL6	T5PL5	T5PL4	T5PL3	T5PL2	T5PL1	T5PL0
R	T5CW7	T5CW6	T5CW5	T5CW4	T5CW3	T5CW2	T5CW1	T5CW0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T5PL** [7:0] 定时/计数器 5 重载值的低字节

Bit [7:0] 读：**T5CW** [7:0]定时/计数器 5 脉宽捕获值的低字节

注意：写入数据顺序要按照从高字节到低字节的顺序写。例如：要设置 16 位定时器模式先设置位于高字节的 P_TMR5_ CapHi 然后设置位于低字节的 P_TMR5_ Cap。

10.2.23 定时/计数器 5 脉宽捕获值的高字节(P_TMR5_CapHi, $24) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T5PL15	T5PL14	T5PL13	T5PL12	T5PL11	T5PL10	T5PL9	T5PL8
R	T5CW15	T5CW14	T5CW13	T5CW12	T5CW11	T5CW10	T5CW9	T5CW8
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T5PL** [15:8] 定时/计数器 5 重载值的高字节

Bit [7:0] 读：**T5CW** [15:8] 定时/计数器 5 脉宽捕获值的高字节

10.2.24 定时/计数器 5 周期捕获值(P_TMR5_CapCycle8, $24) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T5PL15	T5PL14	T5PL13	T5PL12	T5PL11	T5PL10	T5PL9	T5PL8
R	T5CC7	T5CC6	T5CC5	T5CC4	T5CC3	T5CC2	T5CC1	T5CC0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T5PL** [15:8] 定时/计数器 5 重载值的高字节

Bit [7:0] 读：**T5CC** [7:0] 定时/计数器 5 周期捕获值

10.2.25 定时/计数器 5 周期捕获值的低字节(P_TMR5_CapCycleLo, $25) (R)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T5CC7	T5CC6	T5CC5	T5CC4	T5CC3	T5CC2	T5CC1	T5CC0
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Bit [7:0] 读：**T5CC** [15:8] 定时/计数器 5 周期捕获值的低字节

10.2.26 定时/计数器 5 周期捕获值的高字节(P_TMR5_CapCycleHi, $5F) (R)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T5CC15	T5CC14	T5CC13	T5CC12	T5CC11	T5CC10	T5CC9	T5CC8
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Bit [7:0] 读：**T5CC** [15:8] 定时/计数器 5 周期捕获值的高字节

10.2.27 捕获控制寄存器(P_CAP_Ctrl, $58) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
CAPOPT	CAPIP4	CAPIP3	CAP1P2	CAPIP1	CAP1P0	CAP1ES	CAP0ES
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 CAPOPT: 捕获数据保持选择位

1= 即使接收到新的捕获数据，也不更新，保持原来的捕获数据，除非对捕获数据寄存器进行读操作。

0=若接收到新数据，立即更新捕获数据

Bit 6 CAPIP4: 捕获器 4 中断触发沿极性选择位

1=与捕获器 4 计数触发沿极性相同

0=与捕获器 4 计数触发沿极性相反

Bit 5 CAPIP3: 捕获器 3 中断触发沿极性选择位

1=与捕获器 3 计数触发沿极性相同

0=与捕获器 3 计数触发沿极性相反

Bit 4 CAPIP2: 捕获器 2 中断触发沿极性选择位

1=与捕获器 2 计数触发沿极性相同

0=与捕获器 2 计数触发沿极性相反

Bit 3 CAPIP1: 捕获器 1 中断触发沿极性选择位

1=与捕获器 1 计数触发沿极性相同

0=与捕获器 1 计数触发沿极性相反

Bit 2 CAPIP0: 捕获器 0 中断触发沿极性选择位

1=与捕获器 0 计数触发沿极性相同

0=与捕获器 0 计数触发沿极性相反

Bit 1 CAP1ES:捕获器 1 的计数触发沿极性选择位

1=下降沿触发计数（下降沿清除计数器）

0=上升沿触发计数（上升沿清除计数器）

Bit 0 CAP0ES:捕获器 0 的计数触发沿极性选择位

1=下降沿触发计数（下降沿清除计数器）

0=上升沿触发计数（上升沿清除计数器）

10.3 操作

当定时/计数器用于 8 位或 16 位的捕获模式下，捕获输入管脚(例如定时/计数器 0 的 PB0)必须设置成输入，才能进行事件捕获。为了保证捕获值的正确，必须选择选择正确的分频率。当需要得到脉宽值或周期值时，捕获器的计数清除极性和捕获中断触发极性也必须设置正确。捕获器操作相关的寄存器如下：P_CAP_Ctrl ($58)、P_IRQ_Opt0 ($33) 和 P_IRQ_Opt1 ($34)。如图 10-1 所示为捕获器原理框图。关于捕获中断的详细步骤见 10-4 节。

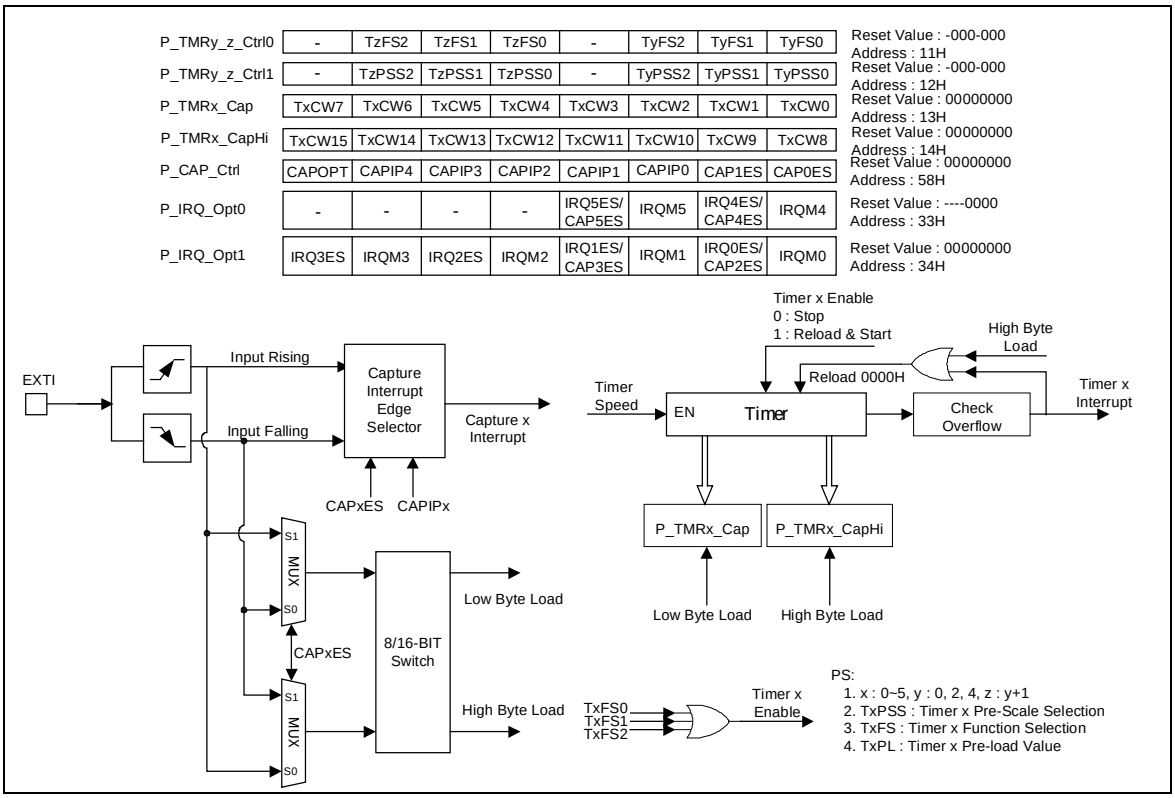


图 10-1 捕获器原理框图

10.3.1 8 位捕获器

当定时/计数器处于 8 位捕获模式下时，建议将中断触发沿极性与捕获器的计数清除极性设置成相同的极性。比如，如果捕获器的计数清除设置为下降沿(CAPxES=1)清除，中断触发也是下降沿(CAPIPx = 1)，并且 CAPOPT=0(收到新数据后更新捕获数据)，这样，一旦捕获器输入管脚上有下降沿输入时，计数器马上清除。在接下来的一个上升沿来临时，计数器的值即脉宽值被立即送入寄存器 P_TMRx_Cap 中，然后接着继续计数。在下一个下降沿到来时，计数器的值即周期值又被送入寄存器 P_TMRx_CapHi 中，同时中断标志 CAPxIF 置位，然后计数器清零又重新开始计数，进行下一个捕获操作。如果 CAPOPT=1，并且没有对存放脉宽值或周期值的寄存器进行读操作，那么即使有新数据，这两个寄存器的值也不会被更新，除非进行读操作。

另外，如果捕获脉宽大于定时/计数器所能表示的最大值时，可以用定时/计数器的溢出中断来辅助计数。例如：如果捕获的脉宽信号大于定时/计数器 1 最大计数值($FF)，那么，当定时/计数器计数到$FF 时，发生溢出中断，中断次数加 1，同时，定时/计数器继续计数。当计数完毕，可以根据中断次数和当前计数值计算出正确的捕获值。

8 位捕获模式和 8 位/16 位定时/计数器模式有所不同。在 16 位定时/计数模式下，除了定时/计数器 5，其它定时/计数器只能进行脉宽测量，因为他们必须占用两个字节的 8 位寄存器来保存捕获的数据。但是在 8 位定时/计数模式下，即可以进行脉宽测量页可以进行周期测量。8 位捕获器操作时序见图 10-2。

捕获器模块还为用户提供了一次输入信号保持功能。将 P_CAP_Ctrl.CAPOPT 位设置为 1

就可以进行一次性输入信号捕获，捕获器中一直会保持这个捕获信号的值，直到将该信号读出。然后再进行第二次捕获、保持、读出，如此循环。可以利用该功能进行单一脉冲的测量。8 位脉宽捕获和保持的操作时序见图 10-3。

8 位的脉宽捕获和周期计算方法见公式 10-2。

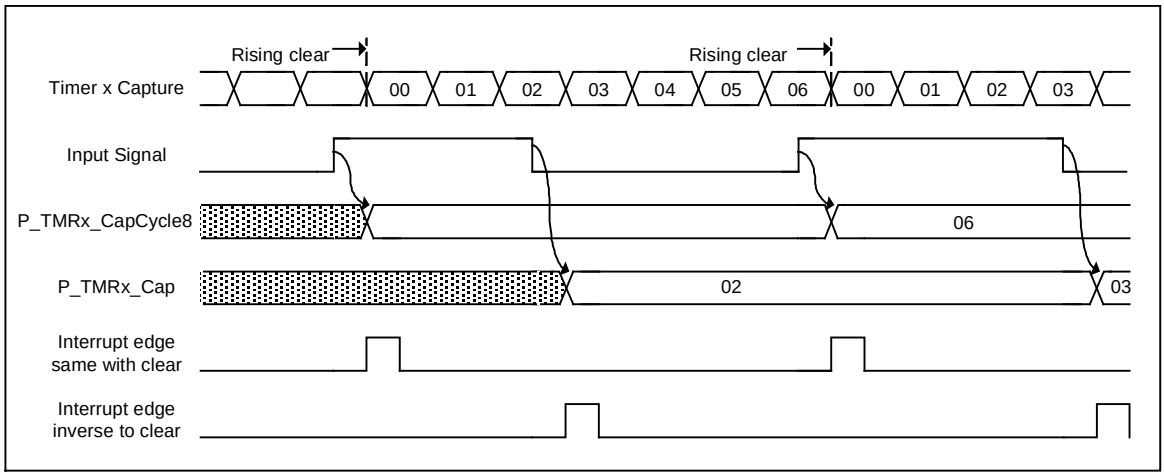


图 10-2 8 位捕获操作

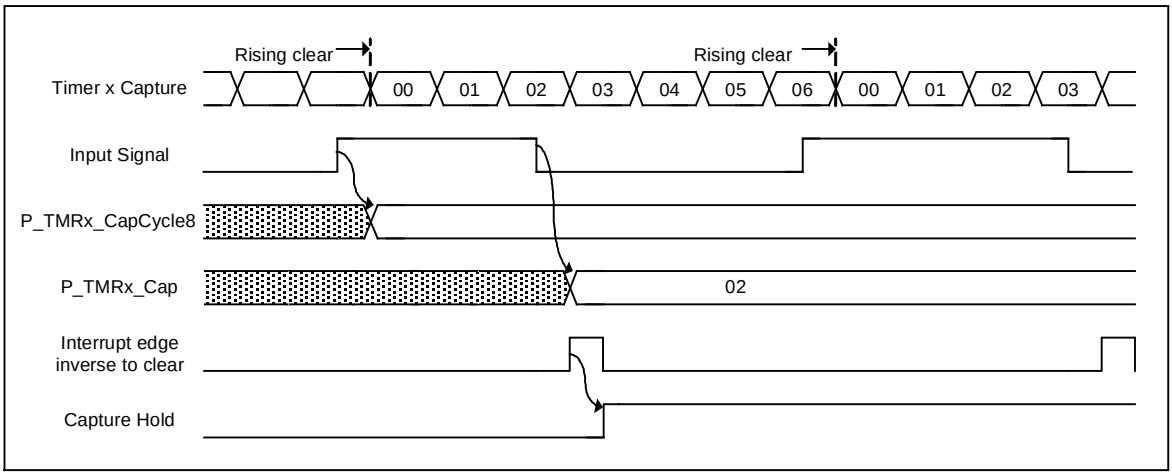


图 10-3 8 位脉宽捕获和保持操作

公式 10-2: 8 位脉宽捕获和周期捕获的计算公式

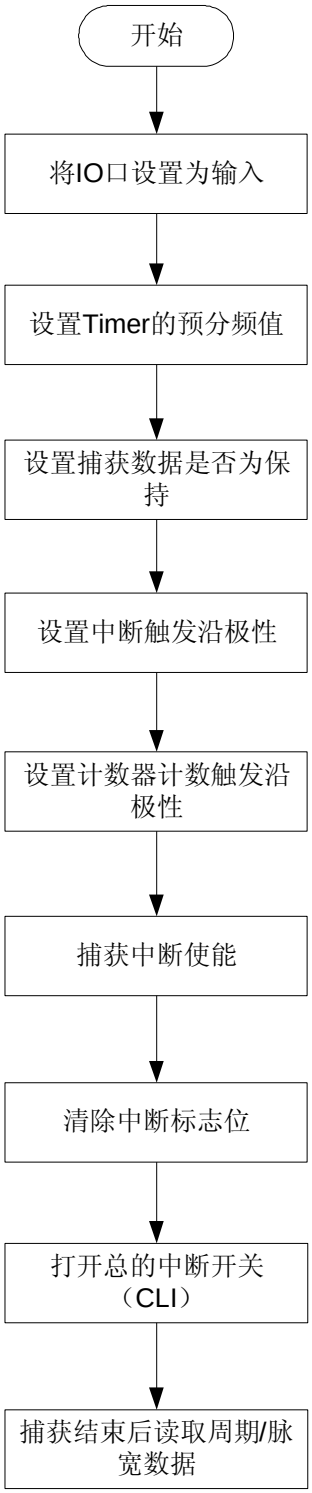
T_TMRx = 定时/计数器 x 分频后的时钟周期, x = 0 ~ 5

8 位定时/计数器

捕获脉宽值 = [P_TMRx_Cap + 1] * T_TMRx

捕获周期值 = [P_TMRx_CapHi + 1] * T_TMRx

注意：捕获的脉宽/周期值允许有一个时钟周期的误差。



[1] 将捕获端口设置为输入

[2] 设置 P_TMRx_x_Ctrl1.TxPSC[2:0]

[3] 设置 P_CAP_Ctrl.CAPOPT

[4] 设置 P_CAP_Ctrl.CAPIPx

[5] 设置 P_CAP_Ctrl.CAPxES或者
P_IRQ_Opt1.CAPxES或者
P_IRQ_Opt0.CAPxES

[6] 设置 P_INT_Ctrlx

[7] 相应的捕获中断标志位清零

[8] CPU 准备接收中断事件

[9] 读取 P_TMRx_CapHi, P_TMRx_Cap

图 10-4 捕获流程图

【例 10 1】:将定时/计数器 0 设置为 8 位捕获模式

```
lda     #00000000B
sta     P_IOB_Data
sta     P_IOB_Attrib
lda     #11111110B              ; 将 PB0 设置为输入，作为捕获器 0 的输入管脚
sta     P_IOB_Dir
lda     #C_T0FCS_Div_512        ; 定时/计数器 0 分频率 Fsys/512 (15.6KHz~61Hz)
sta     P_TMR0_1_Ctrl1
lda     #C_T08B_CAP             ; 将定时/计数器 0 设置为 8 位捕获模式
sta     P_TMR0_1_Ctrl0

lda     #(C_CAP_IP0+C_CAP0_ES)  ; 设置为下降沿采数据，捕获 0 中断为下降沿触发
sta     P_CAP_Ctrl              ; 捕获 PB0 管脚输入的低电平脉宽
```

10.3.2 16 位捕获器

当定时/计数器处于 16 位捕获模式下时，建议将中断触发沿极性与捕获器计数清除极性设置成相反。例如：如果捕获器计数清除设置为下降沿清除(CAPxES=1)，那么捕获中断就设置成上升沿触发(CAPIPx = 0)，并且 CAPOPT=0(收到新数据后更新捕获数据)，这样，一旦捕获器输入管脚上有下降沿输入时，计数器马上清除。在接下来的一个上升沿来临时，计数器的值即脉宽值被立即送入寄存器 P_TMRx_Cap 和 P_TMRx_CapHi 中，并且产生捕获中断，然后接着继续计数。在下一个下降沿到来时，计数器清零又重新开始计数，进行下一个捕获操作。如果 CAPOPT=1，并且没有对存放捕获值的寄存器进行读操作，那么即使有新数据，寄存器的值也不会被更新，除非进行读操作。

另外，如果捕获脉宽大于定时/计数器所能表示的最大值时，可以用定时/计数器的溢出中断来辅助计数。例如：如果捕获的脉宽信号大于定时/计数器 1 最大计数值($FFFF)，那么，当定时/计数器计数到$FFFF 时，发生溢出中断，中断次数加 1，同时，定时/计数器继续计数。当计数完毕，可以根据中断次数和当前计数值计算出正确的脉宽值。由于 16 位捕获器只能进行脉宽捕获（捕获 5 除外），所以，当需要测量周期时，可由软件处理来实现。即：在捕获中断的服务子程序中改变中断的触发极性，此时计数器会在脉宽值地基础上继续计数，在下一次捕获中断到来的时候，计数值便是信号的周期值，在中断服务子程序中直接读取即可，同时，改变中断的触发极性，进行下一次捕获。如图 10-5 所示为 16 位捕获器的操作原理，16 位脉宽捕获计算方法见公式 10-3。

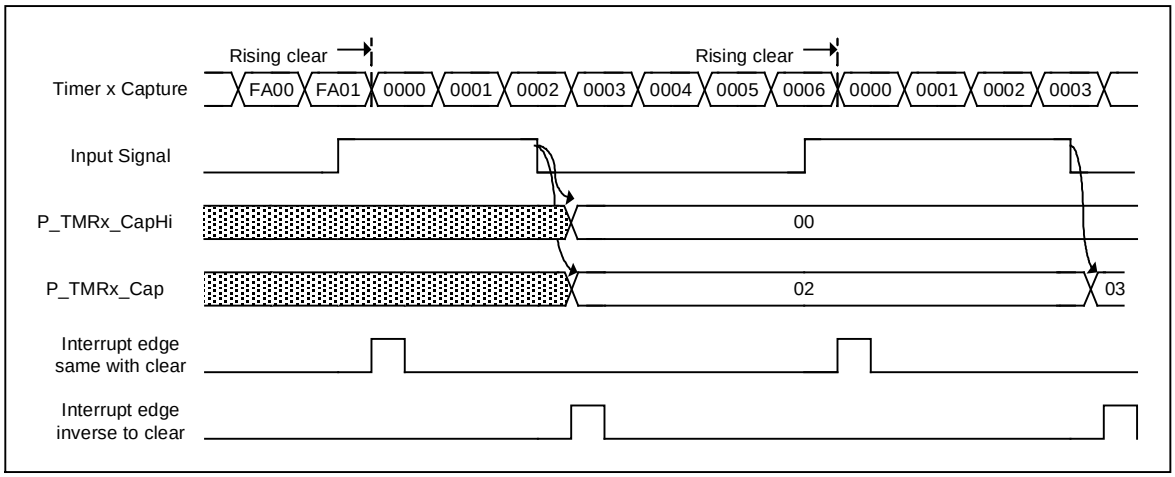


图 10-5 16 位捕获操作

数据寄存器和捕获脉宽/周期之间的关系可用如下公式计算：

公式 10-3: 16 位脉宽捕获的计算公式

T_TMRx = 定时/计数器 x 分频后的时钟周期, x = 0 ~ 5

16 位定时/计数器

捕获脉宽值=[{ P_TMRx_CapHi, P_TMRx_Cap }+1]* T_TMRx

注意：捕获的脉宽/周期值允许有一个时钟周期的误差。

【例 10 2】:将定时/计数器 1 设置为 16 位捕获模式

```
    lda     #00000000B
    sta     P_IOB_Data
    sta     P_IOB_Attrib
    lda     #11111101B              ; 将 PB1 设置为输入，作为捕获器 1 的输入管脚
    sta     P_IOB_Dir
    lda     #C_T1FCS_Div_512        ; 定时/计数器 1 分频率 Fsys/128 (62.5KHz~1Hz)
    sta     P_TMR0_1_Ctrl1
    lda     #C_T116B_CAP            ; 将定时/计数器 1 设置为 16 位捕获模式
    sta     P_TMR0_1_Ctrl0
    lda     #(C_CAP_IP1+C_CAP1_ES)  ; 设置为下降沿采数据，捕获 1 中断为下降沿触发
    sta     P_CAP_Ctrl              ; 测量 PB1 管脚输入的低电平脉宽
```

10.4 捕获中断

捕获器相关中断如下：

- 定时/计数器 0 捕获中断
- 定时/计数器 1 捕获中断
- 定时/计数器 2 捕获中断
- 定时/计数器 3 捕获中断

- 定时/计数器 4 捕获中断
- 定时/计数器 5 捕获中断

下面是设置定时/计数器捕获中断的几个步骤：

1、执行指令“SEI ”关闭总的中断开关；
2、决定采用 8 位还是 16 位的捕获方式并选择一个合适的定时/计数器 x (x=0~5)；
3、设置所选的定时/计数器的相关寄存器；
4、在寄存器 P_IRQ_Opt0($33)、P_IRQ_Opt1($34)中使能该定时/计数器的捕获中断；
5、执行指令”CLI”打开总的中断开关；
6、设置完毕，等待捕获中断产生。

【例 10 3】:将定时/计数器 1 设置为 16 位捕获模式并使能捕获中断。

```
    lda     #00000000B
    sta     P_IOB_Data
    sta     P_IOB_Attrib
    lda     #11111101B              ; 将 PB1 设置为输入，作为捕获器 1 的输入管脚
    sta     P_IOB_Dir
    lda     #C_T1FCS_Div_128        ; 定时/计数器 1 分频率 Fsys/128 (62.5KHz~1Hz)
    sta     P_TMR0_1_Ctrl1
    lda     #C_T116B_CAP            ; 将定时/计数器 1 设置为 16 位捕获模式
    sta     P_TMR0_1_Ctrl0
    lda     #(C_CAP_IP1+C_CAP1_ES)  ; 设置为下降沿采数据，捕获 1 中断为下降沿触发
    sta     P_CAP_Ctrl              ; 测量从 PB1 输入的低电平脉宽
L_TestT1L61:
    lda     P_INT_Flag1
    and     # C_INT_CAP1IF          ; 捕获 1 中断? (捕获数据就绪)
    beq     L_TestT1L61             ; 否
    lda     # C_INT_CAP1IF
    sta     P_INT_Flag1             ; 清除捕获 1 中断标志
    lda     P_TMR1_Cap              ; 将脉宽低字节值送入累加器 A
    ldx     P_TMR1_CapHi            ; 将脉宽高字节值送入 X 寄存器
    jmp     L_TestT1L61
```

10.5 设计参考

【例 10 4】：以 SPMC65P2404A 为例，将 Timer0 初始化为捕获功能，具体步骤如下

- 采用 Timer0 的 8 位定时/计数器功能
- 将 Timer0 时钟源设为 Fsys/512
- 将定时/计数器 0 设置为 8 位捕获模式
- 下降沿清除，并触发中断

```
.SYNTAX     6502                ; process standard 6502 addressing syntax
.LINKLIST                       ; generate linklist information
.SYMBOLS                        ; generate symbolic debug information

.INCLUDE        spmc65p2404a.inc

    .PAGE0                      ; define values in the range from $00 to $FF
;
    .DATA                       ; define data storage section
;
    .CODE
;=====================================
;=                                   =
;= Power on Reset Process - Main Program  =
;=                                   =
;=====================================
V_Reset:
    sei                         ; 关闭总的中断开关
    ldx     #C_STACK_BOTTOM     ; 将堆栈指针初始化为$00FF 位置上
    txs                         ; 将 X 寄存器的值传送到堆栈指针(SP)中
;
    lda     #$FF                ; 清除复位标志
    sta     P_SYS_Ctrl
    sta     P_SYS_Ctrl
;
    jsr     F_InitIOPort            ; 初始化 IO 端口
;
    lda     #$00
    sta     P_IOB_Data
    sta     P_IOB_Attrib

    lda     #11111110B
    sta     P_IOB_Dir           ; 将 PB0 设置为输入，作为捕获器 0 的输入管脚
;
    lda     #C_T0FCS_Div_512    ; 设置 Timer0 的时钟频率为 Fsys/512
    sta     P_TMR0_1_Ctrl1

    lda     #C_T08B_CAP         ; 设置 Timer0 为 8 位脉宽捕获模式
    sta     P_TMR0_1_Ctrl0
;
    lda     #(C_CAP0_ES + C_CAP_IP0); 下降沿清除并触发中断
    sta     P_CAP_Ctrl          ;测量 PB0 输入的低电平脉宽
```

```
;
    lda     #$FF                ; 清除中断请求标志
    sta     P_INT_Flag0
    sta     P_INT_Flag1
    lda     #C_INT_CAP0IE       ; 使能 Timer0 捕获中断
    sta     P_INT_Ctrl1
    cli                         ; 打开总的中断开关

;
L_Main:
    nop
    jmp     L_Main
;
F_InitIOPort:
;
;       ///     初始化  Port A          ///
;
        lda     #00000000B
        sta     P_IOA_Data
        lda     #00000000B
        sta     P_IOA_Attrib
        lda     #11111111B
        sta     P_IOA_Dir
;
        rts
;=====================================
;=                                   =
;=       IRQ Interrupt Service Routine       =
;=                                   =
;=====================================
V_IRQ:
    pha                             ; 将累加器 A 的值压入堆栈
    txa                             ; 将 X 寄存器的值送入累加器 A 中
    pha                             ; 将累加器 A 的值压入堆栈(即 push X)
;
; Timer 0  捕获中断  ?
;
    lda     P_INT_Flag1
    and     #C_INT_CAP0IF
    beq     L_exit_irq
;
    lda     #C_INT_CAP0IF
```

```
    sta     P_INT_Flag1
;
    lda     P_TMR0_Cap
    sta     G_CapWidthBuf0          ; 如果输入频率 = 100 Hz, 那么 G_CapWidthBuf0 ~= 78
;
    lda     P_IOA_Data
    eor     #$FF
    sta     P_IOA_Data              ; PA 口输出数据取反
;
L_exit_irq:
    pla                             ; 将堆栈内容弹出给 A
    tax                             ; 将累加器 A 的值送入 X 寄存器中
    pla                             ; 将堆栈内容弹出给 A
;
    rti

;=====================================
;=                                   =
;=       Interrupt Vector Table      =
;=                                   =
;=====================================
VECTOR:             .SECTION
    DW      V_NMI                   ; may download program emulated either
    DW      V_Reset                 ; in internal memory or external memory
    DW      V_IRQ                   ; dw define two bytes interrupt vector
;
;=====================================
;=                                   =
;=       End Of Interrupt Vector Table =
;=                                   =
;=====================================
    .END                            ; end of program
```

10.6 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题 **应用例系列号**

红外接收（IR）	AN_O0320.DOC
过零检测	AN_O0322.DOC
定时/计数器 0 CCP 功能使用说明	AN_O0331.DOC
定时/计数器 1 CCP 功能使用说明	AN_O0332.DOC
定时/计数器 2 CCP 功能使用说明	AN_O0333.DOC
定时/计数器 3 CCP 功能使用说明	AN_O0334.DOC
定时/计数器 4 CCP 功能使用说明	AN_O0335.DOC
定时/计数器 5 CCP 功能使用说明	AN_O0336.DOC
库函数	
标题	**应用例系列号**
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

10.7 修订记录

版本号	日期	备注
V0.1	03/31/2005	第一版

11 比较

11.1 简述

SPMC65X 系列芯片共有 6 个定时/计数器，每个定时/计数器都可以通过相应控制寄存器的设置，具备比较(Compare)功能。即当某一个定时/计数器被设置为比较输出时，相应的管脚就会输出占空比为 50%的方波。详细的比较输出波形信息如图 11-1 所示。SPMC65X 系列芯片的比较功能简介见表 11.1。关于比较功能寄存器的详细设置见 11.2 节。

表 11.1 SPMC65X 系列芯片的比较功能简介

	定时/计数器		捕获器		比较器		PWM
	8 bit	16 bit	8 bit	16 bit	8 bit	16 bit	
定时/计数器 0	YES	YES	脉宽/周期	脉宽	YES	YES	8 bit
定时/计数器 1	YES	YES	脉宽/周期	脉宽	YES	YES	12 bit
定时/计数器 2	YES	YES	脉宽/周期	脉宽	YES	YES	8 bit
定时/计数器 3	YES	YES	脉宽/周期	脉宽	YES	YES	12 bit
定时/计数器 4	YES	YES	脉宽/周期	脉宽	YES	YES	8 bit
定时/计数器 5	YES	YES	脉宽/周期	脉宽/周期	YES	YES	16 bit

这 6 种定时器共有以下三种不同的组合，可以灵活选用具体介绍见下：

1. 组合 I：(定时器 0、定时器 2、定时器 4)
   8 位或 16 位定时/计数器
   8 位或 16 位捕获模式(8 位脉宽/周期测量，16 位脉宽测量)
   8 位或 16 位比较模式
   8 位 PWM 模式

2. 组合 II：(定时器 1、定时器 3)
   8 位或 16 位定时/计数器
   8 位或 16 位捕获模式(8 位脉宽/周期测量，16 位脉宽测量)
   8 位或 16 位比较模式
   12 位 PWM 模式

3. 组合 III：(定时器 5)
   8 位或 16 位定时/计数器
   8 位或 16 位捕获模式(8 位/16 位脉宽/周期测量
   8 位或 16 位比较模式
   16 位 PWM 模式

11.2 控制寄存器

11.2.1 定时/计数器 0-1 控制寄存器 0 (P_TMR0_1_Ctrl0, $11) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T1FS2	T1FS1	T1FS0	-	T0FS2	T0FS1	T0FS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留

Bit [6:4] **T1FS**[2：0]: 定时/计数器 1 功能设置位

111 = 12 位 PWM

110 = 16 位脉宽捕获值

101 = 16 位比较器

100 = 16 位定时/计数器

011 = 8 位脉宽/周期捕获

010= 8 位比较器

001 = 8 位定时/计数器

000 = 禁止

Bit 3 保留

Bit [2:0] **T0FS**[2：0]: 定时/计数器 0 功能设置位

111 = 8 位 PWM

110 = 16 位脉宽捕获值

101 = 16 位比较器

100 = 16 位定时/计数器

011 = 8 位脉宽/周期捕获

010= 8 位比较器

001 = 8 位定时/计数器

000 = 禁止

11.2.2 定时/计数器 0-1 控制寄存器 1 (P_TMR0_1_Ctrl1, $12) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T1PSS2	T1PSS1	T1PSS0	-	T0PSS2	T0PSS1	T0PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留

Bit [6:4] **T1PSS**[2：0]: 定时/计数器 1 预分频设置位

111 = 外部时钟输入

110 = F_{SYS} ÷ 512

101 = F_{SYS} ÷ 128

100 = F_{SYS} ÷ 32

011 = F_{SYS} ÷ 8

010 = F_{SYS} ÷ 4

001 = F_{SYS} ÷ 2
000 = F_{SYS}

Bit 3 保留

Bit [2:0] **T0PSS**[2：0]: 定时/计数器 0 预分频设置位
111 = 外部时钟输入
110 = F_{SYS} ÷ 512
101 = F_{SYS} ÷ 128
100 = F_{SYS} ÷ 32
011 = F_{SYS} ÷ 8
010 = F_{SYS} ÷ 4
001 = F_{SYS} ÷ 2
000 = F_{SYS}
FSYS：系统时钟源

11.2.3 比较器 0 比较值的低字节(P_TMR0_Comp, $13) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T0CP7	T0CP6	T0CP5	T0CP4	T0CP3	T0CP2	T0CP1	T0CP0
R	T0CN7	T0CN6	T0CN5	T0CN4	T0CN3	T0CN2	T0CN1	T0CN0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T0CP** [7:0] 定时/计数器 0 比较值的低字节
Bit [7:0] 读：**T0CN** [7:0] 定时/计数器 0 计数值的低字节

注意：写入数据顺序要按照从高字节到低字节的顺序写。例如：要设置 16 位定时器模式先设置位于高字节的 P_TMR0_PreloadHi 然后设置位于低字节的 P_TMR0_Preload。

11.2.4 比较器 0 比较值的高字节(P_TMR0_CompHi, $14) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T0CP15	T0CP14	T0CP13	T0CP12	T0CP11	T0CP10	T0CP9	T0CP8
R	T0CN15	T0CN14	T0CN13	T0CN12	T0CN11	T0CN10	T0CN9	T0CN8
	0	0	0	0	0	0	0	0

Bit [7:0]写：**T0CP** [15:8] 定时/计数器 0 比较值的高字节
Bit [7:0]读：**T0CN** [15:8] 定时/计数器 0 计数值的高字节

11.2.5 比较器 1 比较值的低字节(P_TMR1_Comp, $15) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T1CP7	T1CP6	T1CP5	T1CP4	T1CP3	T1CP2	T1CP1	T1CP0
R	T1CN7	T1CN6	T1CN5	T1CN4	T1CN3	T1CN2	T1CN1	T1CN0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T1CP** [7:0] 定时/计数器 1 比较值的低字节

Bit [7:0] 读：**T1CN** [7:0] 定时/计数器 1 计数值的低字节

注意：写入数据顺序要按照从高字节到低字节的顺序写。例如：要设置 16 位定时器模式先设置位于高字节的 P_TMR1_PreloadHi 然后设置位于低字节的 P_TMR1_Preload。

11.2.6 比较器 1 比较值的高字节(P_TMR1_CompHi, $16) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T1CP15	T1CP14	T1CP13	T1CP12	T1CP11	T1CP10	T1CP9	T1CP8
R	T1CN15	T1CN14	T1CN13	T1CN12	T1CN11	T1CN10	T1CN9	T1CN8
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T1CP** [15:8] 定时/计数器 1 比较值的高字节
Bit [7:0] 读：**T1CN** [15:8] 定时/计数器 1 计数值的高字节

11.2.7 定时/计数器 2-3 控制器 0 (P_TMR2_3_Ctrl0, $18) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T3FS2	T3FS1	T3FS0	-	T2FS2	T2FS1	T2FS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留
Bit [6:4] **T3FS**[2：0]: 定时/计数器 3 功能设置位
111 = 12 位 PWM
110 = 16 位脉宽捕获值
101 = 16 位比较器
100 = 16 位定时/计数器
011 = 8 位脉宽/周期捕获
010= 8 位比较器
001 = 8 位定时/计数器
000 = 禁止
Bit 3 保留
Bit [2:0] **T2FS**[2：0]: 定时/计数器 2 功能设置位
111 = 8 位 PWM
110 = 16 位脉宽捕获值
101 = 16 位比较器
100 = 16 位定时/计数器
011 = 8 位脉宽/周期捕获
010= 8 位比较器
001 = 8 位定时/计数器
000 = 禁止

11.2.8 定时/计数器 2-3 控制器 1(P_TMR2_3_Ctrl1, $19) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T3PSS2	T3PSS1	T3PSS0	-	T2PSS2	T2PSS1	T2PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留

Bit [6:4] **T3PSS** [2：0]: 定时/计数器 3 预分频设置位

111 = 外部时钟输入

110 = F_{SYS} ÷ 512

101 = F_{SYS} ÷ 128

100 = F_{SYS} ÷ 32

011 = F_{SYS} ÷ 8

010 = F_{SYS} ÷ 4

001 = F_{SYS} ÷ 2

000 = F_{SYS}

Bit 3 保留

Bit [2:0] **T2PSS** [2：0]: 定时/计数器 2 预分频设置位

111 = 外部时钟输入

110 = F_{SYS} ÷ 512

101 = F_{SYS} ÷ 128

100 = F_{SYS} ÷ 32

011 = F_{SYS} ÷ 8

010 = F_{SYS} ÷ 4

001 = F_{SYS} ÷ 2

000 = F_{SYS}

FSYS：系统时钟源

11.2.9 比较器 2 比较值的低字节(P_TMR2_Comp, $1A) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T2CP7	T2CP6	T2CP5	T2CP4	T2CP3	T2CP2	T2CP1	T2CP0
R	T2CN7	T2CN6	T2CN5	T2CN4	T2CN3	T2CN2	T2CN1	T2CN0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T2CP** [7:0] 定时/计数器 2 比较值的低字节

Bit [7:0] 读：**T2CN** [7:0] 定时/计数器 2 计数值的低字节

注意：写入数据顺序要按照从高字节到低字节的顺序写。例如：要设置 16 位定时器模式先设置位于高字节的 P_TMR2_PreloadHi 然后设置位于低字节的 P_TMR2_Preload。

11.2.10 比较器 2 比较值的高字节(P_TMR2_CompHi, $1B) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T2CP15	T2CP14	T2CP13	T2CP12	T2CP11	T2CP10	T2CP9	T2CP8
R	T2CN15	T2CN14	T2CN13	T2CN12	T2CN11	T2CN10	T2CN9	T2CN8
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T2CP** [15:8] 定时/计数器 2 比较值的高字节

Bit [7:0] 读：**T2CN** [15:8] 定时/计数器 2 计数值的高字节

11.2.11 比较器 3 比较值的低字节(P_TMR3_Comp, $1C) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T3CP7	T3CP6	T3CP5	T3CP4	T3CP3	T3CP2	T3CP1	T3CP0
R	T3CN7	T3CN6	T3CN5	T3CN4	T3CN3	T3CN2	T3CN1	T3CN0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T3CP** [7:0] 定时/计数器 3 比较值的低字节

Bit [7:0] 读：**T3CN** [7:0] 定时/计数器 3 计数值的低字节

注意：写入数据顺序要按照从高字节到低字节的顺序写。例如：要设置 16 位定时器模式先设置位于高字节的 P_TMR3_PreloadHi 然后设置位于低字节的 P_TMR3_Preload。

11.2.12 比较器 3 比较值的高字节(P_TMR3_CompHi, $1D) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T3CP15	T3CP14	T3CP13	T3CP12	T3CP11	T3CP10	T3CP9	T3CP8
R	T3CN15	T3CN14	T3CN13	T3CN12	T3CN11	T3CN10	T3CN9	T3CN8
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T3CP** [15:8] 定时/计数器 3 比较值的高字节

Bit [7:0] 读：**T3CN** [15:8] 定时/计数器 3 计数值的高字节

11.2.13 定时/计数器 4-5 控制器 0 (P_TMR4_5_Ctrl0, $1F) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T5FS2	T5FS1	T5FS0	-	T4FS2	T4FS1	T4FS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留

Bit [6:4] **T5FS**[2：0]: 定时/计数器 5 功能设置位

111 = 16 位 PWM

110 = 16 位脉宽捕获值

101 = 16 位比较器

100 = 16 位定时/计数器
011 = 8 位脉宽/周期捕获
010= 8 位比较器
001 = 8 位定时/计数器
000 = 禁止

Bit 3 保留
Bit [2:0] **T4FS**[2：0]: 定时/计数器 4 功能设置位
111 = 8 位 PWM
110 = 16 位脉宽捕获值
101 = 16 位比较器
100 = 16 位定时/计数器
011 = 8 位脉宽/周期捕获
010 = 8 位比较器
001 = 8 位定时/计数器
000 = 禁止

11.2.14 定时/计数器 4-5 控制器 1(P_TMR4_5_Ctrl1, $20) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T5PSS2	T5PSS1	T5PSS0	-	T4PSS2	T4PSS1	T4PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留
Bit [6:4] **T5PSS** [2：0]: 定时/计数器 5 预分频设置位
111 = 外部时钟输入
110 = F_{SYS} ÷ 512
101 = F_{SYS} ÷ 128
100 = F_{SYS} ÷ 32
011 = F_{SYS} ÷ 8
010 = F_{SYS} ÷ 4
001 = F_{SYS} ÷ 2
000 = F_{SYS}
Bit 3 保留
Bit [2:0] **T4PSS** [2：0]: 定时/计数器 4 预分频设置位
111 = 外部时钟输入
110 = F_{SYS} ÷ 512
101 = F_{SYS} ÷ 128
100 = F_{SYS} ÷ 32
011 = F_{SYS} ÷ 8
010 = F_{SYS} ÷ 4
001 = F_{SYS} ÷ 2

000 = F_{SYS}
FSYS：系统时钟源

11.2.15 比较器 4 比较值的低字节(P_TMR4_Comp, $21) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T4CP7	T4CP6	T4CP5	T4CP4	T4CP3	T4CP2	T4CP1	T4CP0
R	T4CN7	T4CN6	T4CN5	T4CN4	T4CN3	T4CN2	T4CN1	T4CN0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T4CP** [7:0] 定时/计数器 4 比较值的低字节
Bit [7:0] 读：**T4CN** [7:0] 定时/计数器 4 计数值的低字节

注意：写入数据顺序要按照从高字节到低字节的顺序写。例如：要设置 16 位定时器模式先设置位于高字节的 P_TMR4_PreloadHi 然后设置位于低字节的 P_TMR4_Preload。

11.2.16 比较器 4 比较值的高字节(P_TMR4_CompHi, $22) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T4CP15	T4CP14	T4CP13	T4CP12	T4CP11	T4CP10	T4CP9	T4CP8
R	T4CN15	T4CN14	T4CN13	T4CN12	T4CN11	T4CN10	T4CN9	T4CN8
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T4CP** [15:8] 定时/计数器 4 比较值的高字节
Bit [7:0] 读：**T4CN** [15:8] 定时/计数器 4 计数值的高字节

11.2.17 比较器 5 比较值的低字节(P_TMR5_Comp, $23) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T5CP7	T5CP6	T5CP5	T5CP4	T5CP3	T5CP2	T5CP1	T5CP0
R	T5CN7	T5CN6	T5CN5	T5CN4	T5CN3	T5CN2	T5CN1	T5CN0
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T5CP** [7:0] 定时/计数器 5 比较值的低字节
Bit [7:0] 读：**T5CN** [7:0] 定时/计数器 5 计数值的低字节

注意：写入数据顺序要按照从高字节到低字节的顺序写。例如：要设置 16 位定时器模式先设置位于高字节的 P_TMR5_PreloadHi 然后设置位于低字节的 P_TMR5_Preload。

11.2.18 比较器 5 比较值的高字节(P_TMR5_CompHi, $24) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
W	T5CP15	T5CP14	T5CP13	T5CP12	T5CP11	T5CP10	T5CP9	T5CP8
R	T5CN15	T5CN14	T5CN13	T5CN12	T5CN11	T5CN10	T5CN9	T5CN8
	0	0	0	0	0	0	0	0

Bit [7:0] 写：**T5CP** [15:8] 定时/计数器 5 比较值的高字节

Bit [7:0] 读：**T5CN** [15:8] 定时/计数器 5 计数值的高字节

11.3 操作

当定时/计数器 x 作为 8/16 位比较输出时，相应的管脚(如定时/计数器 0 的 PB2)便会输出精度为 8/16 位的方波，并且可以进行 7 级分频选择。比较功能的计数初值在定时/计数器 x 的比较寄存器 P_TMRx_Comp 和 P_TMRxCompHi(16 位模式中有)中设置。比较输出功能使能后，计数器从设置的初始值开始计数，当计数器发生第一次溢出时，在相应比较管脚上输出一个高电平，然后载入初始值重新开始计数，当计数器发生第二次溢出时，又在相应比较管脚上输出一个低电平，第三次溢出会再次输出一个高电平，依次不断循环。也就是说，相应管脚会一直输出占空比为 50%的方波。

11.3.1 8 位比较输出

8 位比较器输出时序见图 11-1。下面以定时/计数器 0 的 8 位比较输出为例介绍：首先将寄存器 P_TMR0_1_Ctrl0 ($12)的 T0FS[2:0]位设置为“010”并在寄存器 P_TMR0_Comp ($13)中设置好计数初值；比较输出功能使能后，计数器从设置的初始值开始计数，当计数器发生第一次溢出时，PB2 管脚上输出一个高电平。如果定时/计数器 0 的溢出中断使能，此时会产生溢出中断。然后载入初始值重新开始计数，当计数器发生第二次溢出时，PB2 管脚上输出一个低电平，第三次溢出会再次输出一个高电平，依次不断循环。也就是说，PB2 管脚会一直输出占空比为 50%的方波关于比较输出波形的详细描述见表 11-1。比较输出频率计算见公式 11-1。产生比较溢出中断的详细步骤见 11.4 节的描述。

公式 11-1: 8 位比较输出频率计算

$$f_{Comp} = \frac{F_{sys}}{Timer\_prescaler \times (256 - Timer\_Comparevalue)} \times \frac{1}{2}$$

f_{Comp}： 比较输出频率

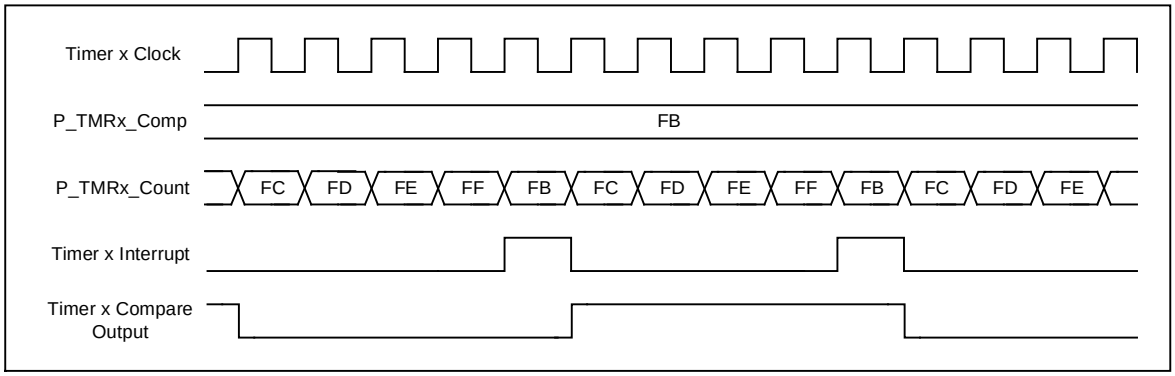


图 11-1 8 位比较器操作时序

【例 11-1】：将定时/计数器 0 设置为 8 位比较模式

```
    lda     #156                ; 启动 Timer0 之前，首先设置 Timer0 的计数初值
    sta     P_TMR0_Preload      ;
    lda     #C_T1FCS_Div_128    ; 设置 Timer0 时钟源为 Fsys/128
    sta     P_TMR0_1_Ctrl1
    lda     #C_T08B_COMP        ; 将 Timer0 设置为 8 位比较输出
    sta     P_TMR0_1_Ctrl0
    lda     #156                ; 设置 Timer0 重载值为 256-156= 100
    sta     P_TMR0_Preload      ; 设置管脚 PB2 的比较频率为 Fsys(8MHz)/128/100= 625Hz
```

11.3.2 16 位比较器

16 位比较输出与 8 位相似，唯一不同的是 16 位比较输出有两个计数寄存器，比较输出时可以进行 16 位计数。下面以定时/计数器 1 的比较输出为例介绍：首先将计数初值的高 8 位放入寄存器 P_TMR1_CompHi ($16)中，低 8 位放入将寄存器 P_TMR1_Comp ($15)，再将寄存器 P_TMR0_1_Ctrl0 ($12)中的 T1FS[2：0]位设置为“101”。计数器开始计数，当计数器发生第一次溢出时，PB3 管脚上输出一个高电平。如果定时/计数器 1 的溢出中断使能，此时会产生溢出中断。 然后载入初始值重新开始计数，当计数器发生第二次溢出时，PB3 管脚上输出一个低电平，第三次溢出会再次输出一个高电平，依次不断循环。也就是说，PB3 管脚会一直输出占空比为 50%的方波。16 位比较输出频率计算见公式 11-2。详细的比较输出波形见图 11-3

$$f_{Comp} = \frac{F_{sys}}{Timer\_prescaler \times (65536 - Timer\_Compare\_value)} \times \frac{1}{2}$$

f_{Comp}： **比较输出频率**

公式 11-2: 16 位比较器的计算公式

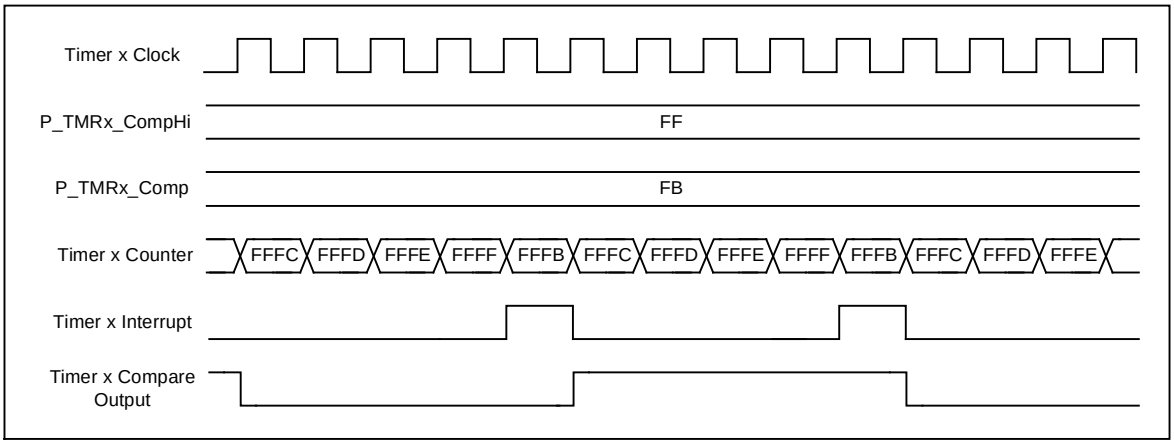


图 11-2 16 位比较输出波形

【例 11-2】:将定时/计数器 1 设置为 16 位比较输出功能并产生输出波形

```
    lda     #$FD                ; 在启动 Timer1 之前，首先设置 Timer1 的计数初值.
    sta     P_TMR1_PreloadHi    ; 设置高字节
    lda     #143                ;
    sta     P_TMR1_Preload      ; 设置低字节
    lda     #C_T1FCS_Div_128    ; 设置定时/计数器 1 时钟源为 Fsys/128
    sta     P_TMR0_1_Ctrl1
    lda     #C_T116B_COMP       ; 将定时/计数器 1 设置为 16 位比较输出
    sta     P_TMR0_1_Ctrl0
    lda     #$FD                ; 首先设置 Timer1 的高字节计数初值为 2x 256= 512
    sta     P_TMR1_PreloadHi
    lda     #143                ; 再设置 Timer1 的低字节计数初值为 256-143= 113
    sta     P_TMR1_Preload      ; 设置管脚 PB3 的比较频率为 Fsys(8Mhz)/128/625= 100Hz
```

11.4 比较器中断

与比较器相关的中断有 5 个，如下：

- 定时/计数器 0 比较中断
- 定时/计数器 1 比较中断
- 定时/计数器 2 比较中断
- 定时/计数器 3 比较中断
- 定时/计数器 4 比较中断
- 定时/计数器 5 比较中断

下面是设置定时/计数器比较中断的几个步骤：

- 执行指令“SEI ”关闭总的中断开关；
- 决定采用 8 位还是 16 位的比较输出方式并选择一个合适的定时/计数器 x (x=0~5)；
- 设置所选的定时/寄存器的相关寄存器；
- 在寄存器 P_INT_Ctrl1($0F)中使能该定时/计数器的比较中断；
- 执行指令”CLI”打开总的中断开关；
- 设置完毕，等待比较中断产生。

11.5 设计参考

【例 11 1：以 SPMC65P2404A 为例，将 Timer1 初始化为比较功能，具体步骤如下

- **n** 采用 Timer1 做为 16 位的比较输出功能
- **n** 将 Timer1 时钟源设为 Fsys/2
- **n** 在 PB3 输出频率为 63Hz 的方波

下降沿清除，并触发中断

```
.SYNTAX     6502                    ; process standard 6502 addressing syntax
.LINKLIST                           ; generate linklist information
.SYMBOLS                            ; generate symbolic debug information

.INCLUDE        spmc65p2404a.inc

    .PAGE0                          ; define values in the range from $00 to $FF
;
    .DATA                           ; define data storage section
;
    .CODE
;=====================================
;=                                   =
;= Power on Reset Process - Main Program  =
;=                                   =
;=====================================
V_Reset:
    sei                             ; 关闭总的中断开关
    ldx     #C_STACK_BOTTOM         ; 将堆栈指针初始化为$00FF 位置上
    txs                             ; 将 X 寄存器的值传送到堆栈指针(SP)中
;
    lda     #$FF                    ; 清除复位标志
    sta     P_SYS_Ctrl
    sta     P_SYS_Ctrl
;
    jsr     F_InitIOPort            ; 初始化 IO 端口
    lda     #$80                     ; Fsys (8MHz) / (2 x ($FFFF - $8000)) = 61 Hz
    sta     P_TMR1_CompHi
    lda     #00
    sta     P_TMR1_Comp
;
    lda     #C_T1FCS_Div_2          ; 设置 Timer1 时钟频率 Fcs/2
    sta     P_TMR0_1_Ctrl1
    lda     #C_T116B_COMP           ; 设置 Timer1 为 16 位的比较输出
```

```
    sta     P_TMR0_1_Ctrl0
;
    lda     #$80
    sta     P_TMR1_CompHi       ; Fsys (8MHz) / (2 x ($FFFF - $8000)) = 61 Hz
    lda     #00
    sta     P_TMR1_Comp
;
L_Main:
    nop
    jmp     L_Main
;
F_InitIOPort:
;
;       ///     初始化  Port A          ///
;
    lda     #00000000B
    sta     P_IOA_Data
    lda     #00000000B
    sta     P_IOA_Attrib
    lda     #11111111B
    sta     P_IOA_Dir
;
    rts
;*************************************************
;*                                               *
;*          Interrupt Vector Table               *
;*                                               *
;*************************************************
VECTOR:             .SECTION
    DW      V_NMI               ; may download program emulated either
    DW      V_Reset             ; in internal memory or external memory
    DW      V_IRQ               ; dw define two bytes interrupt vector
;
;*************************************************
;*                                               *
;*          End Of Interrupt Vector Table        *
;*                                               *
;*************************************************
    .END                        ; end of program
```

11.6 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数

和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题	应用例系列号
定时/计数器 0 CCP 功能使用说明	AN_O0331.DOC
定时/计数器 1 CCP 功能使用说明	AN_O0332.DOC
定时/计数器 2 CCP 功能使用说明	AN_O0333.DOC
定时/计数器 3 CCP 功能使用说明	AN_O0334.DOC
定时/计数器 4 CCP 功能使用说明	AN_O0335.DOC
定时/计数器 5 CCP 功能使用说明	AN_O0336.DOC

库函数

标题	应用例系列号
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

11.7 Revision Histoy

版本号	日期	备注
V0.1	03/31/2005	第一版

12 PWM

12.1 简述

SPMC65X 系列芯片共有 6 个定时/计数器，每个定时/计数器都可以通过相应控制寄存器设置成 PWM 输出功能。定时/计数器作为 PWM 输出使用有三种类型。比如，定时/计数器 0/2/4 只能作为 8 位 PWM 结果输出；定时/计数器 1/3 可以输出 12 位 PWM 结果；定时/计数器 5 可以输出 16 位 PWM 结果。SPMC65X 系列芯片的 PWM 功能简介见表 12-1。

PWM 输出即输出频率、占空比均可调的方波。如表 13.2 所示为经过分频后的 PWM 输出频率。图 12-4 所示为的 PWM 输出波形图。关于寄存器的详细设置见 12.2 章。

表 12-1 SPMC65X 系列芯片 PWM 功能简表

	定时器/事件计数器		捕获器		比较器		PWM
	8 bit	16 bit	8 bit	16 bit	8 bit	16 bit	
定时/计数器 0	YES	YES	脉宽/周期	脉宽	YES	YES	8 bit
定时/计数器 1	YES	YES	脉宽/周期	脉宽	YES	YES	12 bit
定时/计数器 2	YES	YES	脉宽/周期	脉宽	YES	YES	8 bit
定时/计数器 3	YES	YES	脉宽/周期	脉宽	YES	YES	12 bit
定时/计数器 4	YES	YES	脉宽/周期	脉宽	YES	YES	8 bit
定时/计数器 5	YES	YES	脉宽/周期	脉宽/周期	YES	YES	16 bit

这 6 种定时器共有以下三种不同的组合，可以灵活选用具体介绍见下：

1. 组合 I：(定时器 0、定时器 2、定时器 4)
   8 位或 16 位定时/计数器
   8 位或 16 位捕获模式(8 位脉宽/周期测量，16 位脉宽测量)
   8 位或 16 位比较模式
   8 位 PWM 模式

2. 组合 II：(定时器 1、定时器 3)
   8 位或 16 位定时/计数器
   8 位或 16 位捕获模式(8 位脉宽/周期测量，16 位脉宽测量)
   8 位或 16 位比较模式
   12 位 PWM 模式

3. 组合 III：(定时器 5)
   8 位或 16 位定时/计数器
   8 位或 16 位捕获模式(8 位/16 位脉宽/周期测量
   8 位或 16 位比较模式
   16 位 PWM 模式

表 12-3 分频后的 PWM 几种输出频率

结果	PWM 频率			
	预分频 = F_{sys} ÷ 1	预分频 = F_{sys} ÷ 2	预分频 = F_{sys} ÷ 4	预分频 = F_{sys} ÷ 8
16 位 PWM	122HZ	61HZ	30.5 HZ	15.25 HZ
12 位 PWM	1.95KHZ	975HZ	487HZ	243.5HZ
11 位 PWM	3.9 KHZ	1.95KHZ	975HZ	487HZ
10 位 PWM	7.8 KHZ	3.9 KHZ	1.95KHZ	975HZ
9 位 PWM	15.6 KHZ	7.8 KHZ	3.9 KHZ	1.95KHZ
8 位 PWM	31.2 KHZ	15.6 KHZ	7.8 KHZ	3.9 KHZ

注意：系统时钟为 8MHz。

12.2 控制寄存器

12.2.1 定时/计数器 0-1 控制寄存器 0 (P_TMR0_1_Ctrl0, $11) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T1FS2	T1FS1	T1FS0	-	T0FS2	T0FS1	T0FS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留

Bit [6:4] **T1FS**[2：0]: 定时/计数器 1 功能设置位

111 = 12 位 PWM
110 = 16 位脉宽捕获
101 = 16 位比较器
100 = 16 位定时/计数器
011 = 8 位脉宽/周期捕获
010 = 8 位比较器
001 = 8 位定时/计数器
000 = 禁止

Bit 3 保留

Bit [2:0] **T0FS**[2：0]: 定时/计数器 0 功能设置位

111 = 8 位 PWM
110 = 16 位脉宽捕获
101 = 16 位比较器
100 = 16 位定时/计数器
011 = 8 位脉宽/周期捕获
010 = 8 位比较器
001 = 8 位定时/计数器
000 = 禁止

12.2.2 定时/计数器 0-1 控制寄存器 1 (P_TMR0_1_Ctrl1, $12) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T1PSS2	T1PSS1	T1PSS0	-	T0PSS2	T0PSS1	T0PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留

Bit [6:4] **T1PSS**[2：0]: 定时/计数器 1 预分频设置位

111 = 外部时钟输入

110 = F_{SYS} ÷ 512

101 = F_{SYS} ÷ 128

100 = F_{SYS} ÷ 32

011 = F_{SYS} ÷ 8

010 = F_{SYS} ÷ 4

001 = F_{SYS} ÷ 2

000 = F_{SYS}

Bit 3 保留

Bit [2:0] **T0PSS**[2：0]: 定时/计数器 0 预分频设置位

111 = 外部时钟输入

110 = F_{SYS} ÷ 512

101 = F_{SYS} ÷ 128

100 = F_{SYS} ÷ 32

011 = F_{SYS} ÷ 8

010 = F_{SYS} ÷ 4

001 = F_{SYS} ÷ 2

000 = F_{SYS}

FSYS：系统时钟源

12.2.3 定时/计数器 0 PWM 周期值(P_TMR0_PWMPeriod, $13) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T0PP7	T0PP6	T0PP5	T0PP4	T0PP3	T0PP2	T0PP1	T0PP0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0]: T0PP [7:0]: 定时/计数器 0 PWM 周期值 (8 位 PWM 模式)

12.2.4 定时/计数器 0 PWM 占空比值(P_TMR0_PWMDuty, $14) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T0PD7	T0PD6	T0PD5	T0PD4	T0PD3	T0PD2	T0PD1	T0PD0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] T0PD [7:0]: 定时/计数器 0 PWM 占空比值 (8 位 PWM 模式)

8 位 PWM 模式：

使能8位PWM模式时，先写寄存器P_TMR0_PWMDuty，再写寄存器P_TMR0_PWMPeriod

12.2.5 定时/计数器 1 PWM 周期值的低 8 位(P_TMR1_PWMPeriod, $15) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T1PP7	T1PP6	T1PP5	T1PP4	T1PP3	T1PP2	T1PP1	T1PP0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] T1PP [7:0]: 定时/计数器 1 PWM 周期值的低 8 位 (12 位 PWM 模式)

注意：写入数据顺序要按照从高字节到低字节的顺序写。

12 位 PWM 模式

更新周期值时，先写 P_TMR1_DutyPeriod，然后再写 P_TMR1_PWMPeriod.

更新占空比时，先写 P_TMR1_DutyPeriod，然后再写 P_TMR1_PWMDuty

12.2.6 定时/计数器 1 PWM 占空比/周期的高 4 位(P_TMR1_DutyPeriod, $16) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T1PD11	T1PD10	T1PD9	T1PD8	T1PP11	T1PP10	T1PP9	T1PP8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:4] T1PD [11:8]: 定时/计数器 1 PWM 占空比的高 4 位 (12 位 PWM 模式)

Bit [3:0] T1PP [11:8]: 定时/计数器 1 PWM 周期值的高 4 位 (12 位 PWM 模式)

12.2.7 定时/计数器 1 PWM 占空比的低字节(P_TMR1_PWMDuty, $17) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T1PD7	T1PD6	T1PD5	T1PD4	T1PD3	T1PD2	T1PD1	T1PD0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] T1PD [7:0]: 定时/计数器 1 PWM 占空比的低字节(12 位 PWM 模式)

12.2.8 定时/计数器 2-3 控制器 0 (P_TMR2_3_Ctrl0, $18) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T3FS2	T3FS1	T3FS0	-	T2FS2	T2FS1	T2FS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留

Bit [6:4] **T3FS**[2：0]: 定时/计数器 3 功能设置位

111 = 12 位 PWM

110 = 16 位脉宽捕获
101 = 16 位比较器
100 = 16 位定时/计数器
011 = 8 位脉宽/周期捕获
010 = 8 位比较器
001 = 8 位定时/计数器
000 = 禁止

Bit 3 保留

Bit [2:0] **T2FS**[2：0]: 定时/计数器 2 功能设置位
111 = 8 位 PWM
110 = 16 位脉宽捕获
101 = 16 位比较器
100 = 16 位定时/计数器
011 = 8 位脉宽/周期捕获
010 = 8 位比较器
001 = 8 位定时/计数器
000 = 禁止

12.2.9 定时/计数器 2-3 控制器 1(P_TMR2_3_Ctrl1, $19) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T3PSS2	T3PSS1	T3PSS0	-	T2PSS2	T2PSS1	T2PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留

Bit [6:4] **T3PSC**[2：0]: 定时/计数器 3 预分频设置位
111 = 外部时钟输入
110 = F_{SYS} ÷ 512
101 = F_{SYS} ÷ 128
100 = F_{SYS} ÷ 32
011 = F_{SYS} ÷ 8
010 = F_{SYS} ÷ 4
001 = F_{SYS} ÷ 2
000 = F_{SYS}

Bit 3 保留

Bit [2:0] **T2PSC**[2：0]: 定时/计数器 2 预分频设置位
111 =外部时钟输入
110 = F_{SYS} ÷ 512
101 = F_{SYS} ÷ 128
100 = F_{SYS} ÷ 32
011 = F_{SYS} ÷ 8
010 = F_{SYS} ÷ 4

001 = F_{SYS} ÷ 2
000 = F_{SYS}
FSYS：系统时钟源

12.2.10 定时/计数器 2 PWM 周期(P_TMR2_PWMPeriod, $1A) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T2PP7	T2PP6	T2PP5	T2PP4	T2PP3	T2PP2	T2PP1	T2PP0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] T2PP [7:0]: 定时/计数器 2 PWM 周期值 (8 位 PWM 模式)

12.2.11 定时/计数器 2 PWM 占空比(P_TMR2_PWMDuty, $1B) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T2PD7	T2PD6	T2PD5	T2PD4	T2PD3	T2PD2	T2PD1	T2PD0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] T2PD [7:0]: 定时/计数器 2 PWM 占空比(8 位 PWM 模式)

8 位 PWM 模式：
使能 8 位 PWM 模式时，先写寄存器 P_TMR2_PWMDuty，再写寄存器 P_TMR2_PWMPeriod

12.2.12 定时/计数器 3 PWM 周期的低 8 位(P_TMR3_PWMPeriod, $1C) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T3PP7	T3PP6	T3PP5	T3PP4	T3PP3	T3PP2	T3PP1	T3PP0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] T3PP [7:0]: 定时/计数器 3 PWM 周期的低 8 位(12 位 PWM 模式)

注意：写入数据顺序要按照从高字节到低字节的顺序写
12 位 PWM 模式
更新周期值时，先写 P_TMR3_DutyPeriod，然后再写 P_TMR3_PWMPeriod.
更新占空比时，先写 P_TMR3_DutyPeriod，然后再写 P_TMR3_PWMDuty

12.2.13 定时/计数器 3 PWM 占空比/周期的高 4 位(P_TMR3_DutyPeriod, $1D) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T3PD11	T3PD10	T3PD9	T3PD8	T3PP11	T3PP10	T3PP9	T3PP8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:4] T3PD [11:8]: 定时/计数器 3 PWM 占空比 (12 位 PWM 模式)
Bit [3:0] T3PP [11:8]: 定时/计数器 3 PWM 周期值 (12 位 PWM 模式)

12.2.14 定时/计数器 3 PWM 占空比的低 8 位(P_TMR3_PWMDuty, $1E) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T3PD7	T3PD6	T3PD5	T3PD4	T3PD3	T3PD2	T3PD1	T3PD0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] **T3PD** [7:0]: 定时/计数器 3 PWM 占空比的低 8 位(12 位 PWM 模式)

12.2.15 定时/计数器 4-5 控制器 0 (P_TMR4_5_Ctrl0, $1F) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T5FS2	T5FS1	T5FS0	-	T4FS2	T4FS1	T4FS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留
Bit [6:4] **T5FS**[2：0]: 定时/计数器 5 功能设置位
111 = 16-bit PWM
110 = 16 位脉宽捕获
101 = 16 位比较器
100 = 16 位定时/计数器
011 = 8 位脉宽/周期捕获
010 = 8 位比较器
001 = 8 位定时/计数器
000 = 禁止
Bit 3 保留
Bit [2:0] **T4FS**[2：0]: 定时/计数器 4 功能设置位
111 = 8 位 PWM
110 = 16 位脉宽捕获
101 = 16 位比较器
100 = 16 位定时/计数器
011 = 8 位脉宽/周期捕获
010 = 8 位比较器
001 = 8 位定时/计数器
000 = 禁止

12.2.16 定时/计数器 4-5 控制器 1 (P_TMR4_5_Ctrl1, $20) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	T5PSS2	T5PSS1	T5PSS0	-	T4PSS2	T4PSS1	T4PSS0
-	R/W	R/W	R/W	-	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 保留

Bit [6:4] **T5PSS**[2：0]: 定时/计数器 5 预分频设置位

111 = 外部时钟输入

110 = F_{SYS} ÷ 512

101 = F_{SYS} ÷ 128

100 = F_{SYS} ÷ 32

011 = F_{SYS} ÷ 8

010 = F_{SYS} ÷ 4

001 = F_{SYS} ÷ 2

000 = F_{SYS}

Bit 3 保留

Bit [2:0] **T4PSS**[2：0]: 定时/计数器 4 预分频设置位

111 = 外部时钟输入

110 = F_{SYS} ÷ 512

101 = F_{SYS} ÷ 128

100 = F_{SYS} ÷ 32

011 = F_{SYS} ÷ 8

010 = F_{SYS} ÷ 4

001 = F_{SYS} ÷ 2

000 = F_{SYS}

FSYS：系统时钟源

12.2.17 定时/计数器 4 PWM 周期(P_TMR4_PWMPeriod, $21) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T4PP7	T4PP6	T4PP5	T4PP4	T4PP3	T4PP2	T4PP1	T4PP0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] **T4PP** [7:0]: 定时/计数器 4 PWM 周期值(8 位 PWM 模式)

12.2.18 定时/计数器 4 PWM 占空比(P_TMR4_PWMDuty, $22) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T4PD7	T4PD6	T4PD5	T4PD4	T4PD3	T4PD2	T4PD1	T4PD0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] T4PD [7:0]: 定时/计数器 4 PWM 占空比(8 位 PWM 模式)

8 位 PWM 模式：

使能8位PWM模式时，先写寄存器P_TMR4_PWMDuty，再写寄存器P_TMR4_PWMPeriod

12.2.19 定时/计数器 5 PWM 周期的低 8 位(P_TMR5_PWMPeriodLo, $23) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T5PP7	T5PP6	T5PP5	T5PP4	T5PP3	T5PP2	T5PP1	T5PP0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] T5PP [7:0]: 定时/计数器 5 PWM 周期的低 8 位 (16-bit PWM mode)

注意：写入数据顺序要按照从高字节到低字节的顺序写

16 位 PWM 模式

更新周期值时，先写 P_TMR5_DutyPeriod ，然后再写 P_TMR5_PWMPeriodLo

更新占空比时，先写 P_TMR5_DutyPeriod，然后再写 P_TMR5_PWMDutyLo

12.2.20 定时/计数器 5 PWM 周期的高 8 位(P_TMR5_PWMPeriodHi, $24) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T5PP11	T5PP10	T5PP9	T5PP8	T5PP11	T5PP10	T5PP9	T5PP8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] **T5PP** [15:8]: 定时/计数器 5 PWM 周期值(16-bit PWM 模式)

12.2.21 定时/计数器 5 PWM 占空比的低 8 位(P_TMR5_PWMDutyLo, $25) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T5PD7	T5PD6	T5PD5	T5PD4	T5PD3	T5PD2	T5PD1	T5PD0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] T5PD [7:0]: 定时/计数器 5 PWM 占空比的低 8 位(16-bit PWM 模式)

12.2.22 定时/计数器 5 PWM 占空比的高 8 位(P_TMR5_PWMDutyHi, $5F) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T5PD15	T5PD14	T5PD13	T5PD12	T5PD11	T5PD10	T5PD9	T5PD8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] T5PD [15:8]: 定时/计数器 5 PWM 占空比的高 8 位(16-bit PWM 模式)

12.3 操作

12.3.1 8-bit PWM (组合 I)

SPMC65X 系列芯片中，定时器 0/2/4 属于第一种(组合 I)—8 位 PWM。当 TxFS[2:0]设置为 111 后，定时/计数器 x 就成为 8 位 PWM 发生器，相应的管脚(例如，定时/计数器 0 对应 PB2)自动成为 PWM 输出管脚。PWM 周期在寄存器 P_TMRx_PWMPeriod 中设置；占空比在寄存器 P_TMRx_PWMDuty 中设置；在 8 位 PWM 模式下，使能 8 位 PWM 功能后用户必须首先写寄存器 P_TMRx_PWMDuty，然后再写寄存器 P_TMRx_PWMPeriod。时钟的分频系数由 TxPSS[2:0]位选择。当 PWM 使能后，计数器载入计数初值（P_TMRx_PWMPeriod 的值），开始计数，同时，相应的管脚输出低电平。在计数过程中，计数值不断的和占空比值（P_TMRx_PWMDuty 的值）进行比较，相等时，PWM 管脚输出电平翻转，为高电平。计数器继续计数，直到溢出，此时一个周期的方波输出完成。接着，PWM 管脚再次翻转，输出低电平，计数器重新载入计数初值开始计数，产生下一个 PWM 输出，依次不断循环。如图 12-1 所示为 8 位 PWM 输出原理框图。PWM 周期和占空比计算见公式 12-1。关于发生 PWM 周期中断的详细步骤见 12.4 节。

公式 12-1：Duty 8 位 PWM 周期和占空比的计算

If
PRDREG= P_T0DATA0[7:0]
DUTYREG=T0DATA1[7:0]
T_TMRx= Timer x 分频后的时钟周期
Then
PWM 周期=[$100-PRDREG]*T_TMRx
PWM 占空比=[$FF-DUTYREG]*T_TMRx
注意： x = 0, 2, 4

寄存器的设置值和实际值之间的关系详见表 12-5。

表 12-5 8 位 PWM 操作中寄存器的设置值和实际值之间的关系。

PRDREG (Hex)	PWM PERIOD (* T_TMRx)		DUTYREG (Hex)	PWM DUTY(* T_TMRx)	
	Hex	Decimal		Hex	Decimal
$00	$100	256	$00	$FF	255
$01	$FF	255	$01	$FE	254
……	……	……	……	……	……
$FE	$02	2	$FE	$01	1
$FF	$01	1	$FF	$00	0

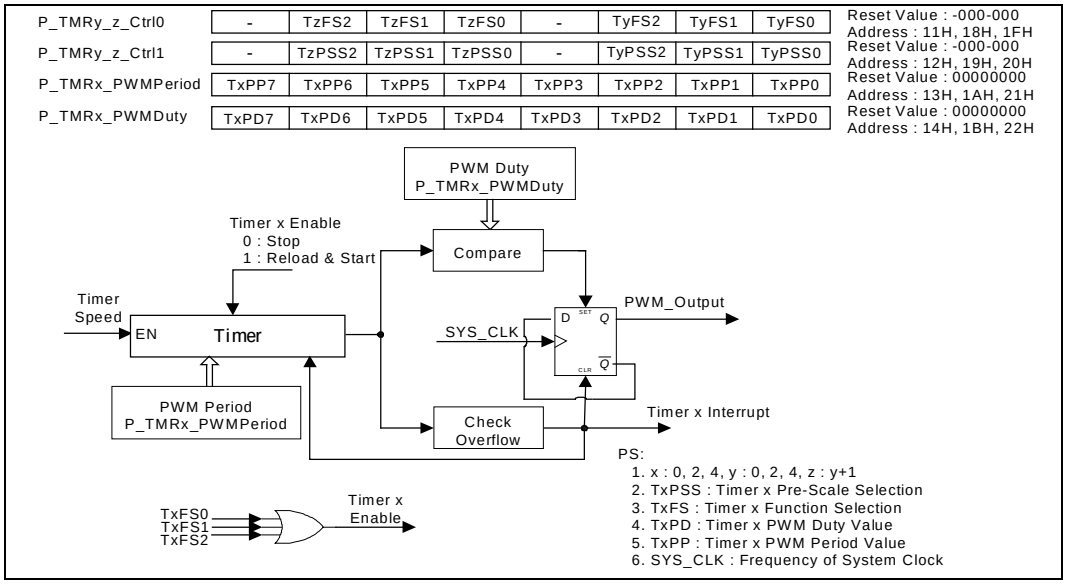


图 12-1 8 位 PWM 原理框图

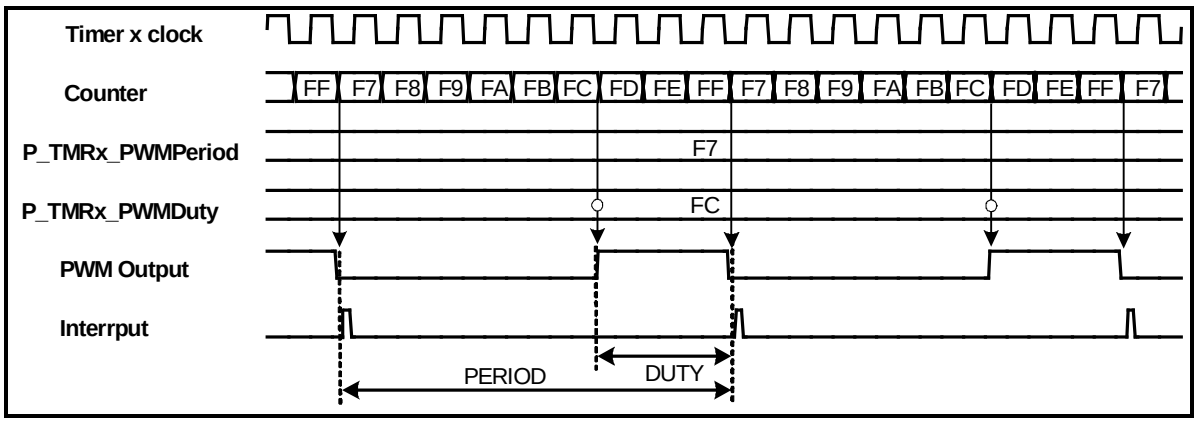


图 12-2 8 位 PWM 操作时序

【例 12 1】:将定时/计数器 0 设置为 8 位 PWM 模式

```
lda     #178                    ;启动 Timer0 之前，首先设置占空比和周期寄存器
sta     P_TMR0_PWMDuty          ;设置占空比寄存器
lda     #$00
sta     P_TMR0_PWMPeriod        ;设置周期寄存器
lda     #C_T0FCS_Div_8          ; 8 设置定时/计数器 0 时钟源为 Fcs/8
sta     P_TMR0_1_Ctrl1
lda     #C_T08B_PWM             ; 将定时/计数器 0 设置为 8 位 PWM 模式
sta     P_TMR0_1_Ctrl0
lda     #178
```

```
sta     P_TMR0_PWMDuty          ; 在 PB2 管脚上输出占空比(255 - 178) / 255 ~= 30%
lda     #$00
sta     P_TMR0_PWMPeriod        ; PWM 频率 = 8.0MHz / (255 x 8) = 3.9 KHz
```

12.3.2 12 位 PWM (组合 II)

SPMC65X 系列芯片中，定时器 1/3 属于第二种(组合 II)—12 位 PWM。当 TxFS[2:0]设置为 111 后，定时/计数器 x 就成为 12 位 PWM 发生器，相应的管脚(例如：Timer1 对应 PB3)自动成为 12 位 PWM 输出管脚。PWM 周期在寄存器 P_TMRx_PWMPeriod 和 P_TMRx_DutyPeriod [3:0]中设置；占空比在寄存器 P_TMRx_DutyPeriod [7:4]和 P_TMRx_PWMDuty.中设置；时钟的分频系数由 TxPSS[2:0]位选择。当 PWM 使能后，计数器载入计数初值（P_TMRx_PWMPeriod 的值），开始计数，同时，相应的管脚(即 Timer1 = PB3)输出低电平。在计数过程中，计数值不断的和占空比值{P_TMRx_DutyPeriod [7:4], P_TMRx_PWMDuty [7:0]}进行比较，相等时，PWM 管脚输出电平翻转，为高电平。计数器继续计数，直到溢出，此时一个周期的方波输出完成。接着，PWM 管脚再次翻转，输出低电平，计数器重新载入计数初值开始计数，产生下一个 PWM 输出，依次不断循环。

一个 8 位 CPU，总线宽度为 8 位，通常无法直接访问 16 位的数据。因此，在 12 位 PWM 模式下，用户需要先写入高字节寄存器 P_TMRx_DutyPeriod (MSB byte)并被缓冲器自动保存，然后再写入低字节 P_TMRx_PWMPeriod (period) / P_TMRx_PWMDuty (duty)，直到 12 位数据全部写入，缓冲器的值才会改变，这样缓冲器中的高字节和低字节将会同时载入定时/计数器 x 中去。如图 12-3 所示为 12 位 PWM 输出原理框图。PWM 周期和占空比计算见公式 12-2。

P_TMRy_z_Ctrl0	-	TzFS2	TzFS1	TzFS0	-	TyFS2	TyFS1	TyFS0	Reset Value : -000-000 Address : 11H, 18H
P_TMRy_z_Ctrl1	-	TzPSS2	TzPSS1	TzPSS0	-	TyPSS2	TyPSS1	TyPSS0	Reset Value : -000-000 Address : 12H, 19H
P_TMRx_PWMPeriod	TxPP7	TxPP6	TxPP5	TxPP4	TxPP3	TxPP2	TxPP1	TxPP0	Reset Value : 00000000 Address : 15H, 1CH
P_TMRx_DutyPeriod	TxPD11	TxPD10	TxPD9	TxPD8	TxPP11	TxPP10	TxPP9	TxPP8	Reset Value : 00000000 Address : 16H, 1DH
P_TMRx_PWMDuty	TxPD7	TxPD6	TxPD5	TxPD4	TxPD3	TxPD2	TxPD1	TxPD0	Reset Value : 00000000 Address : 17H, 1EH

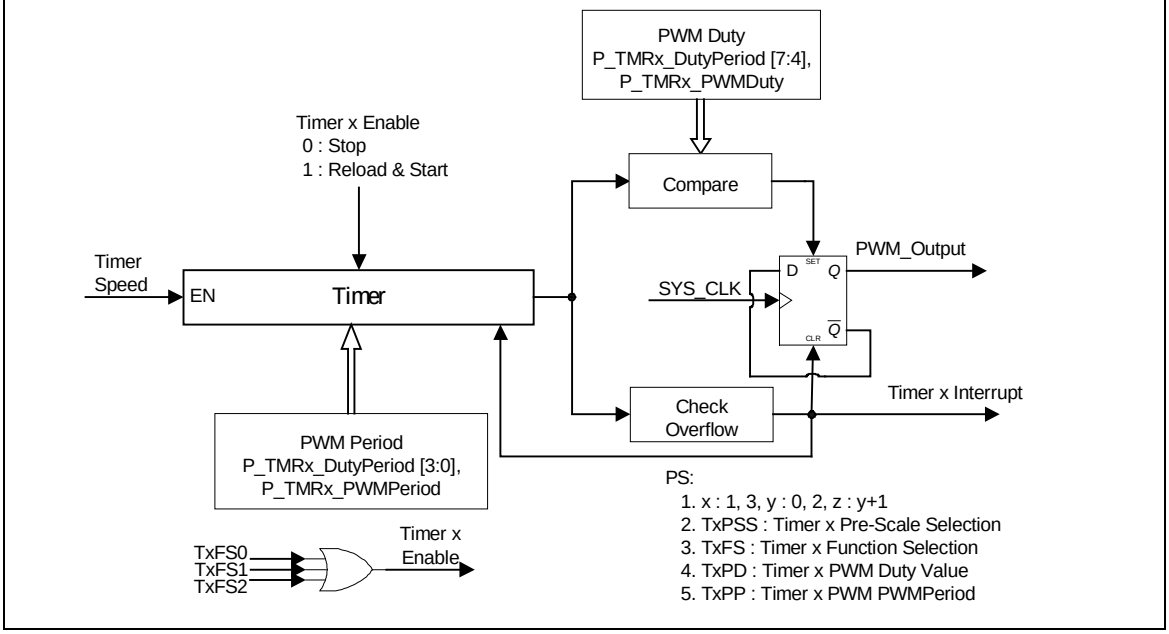


图 12-3 12 位 PWM 原理框图

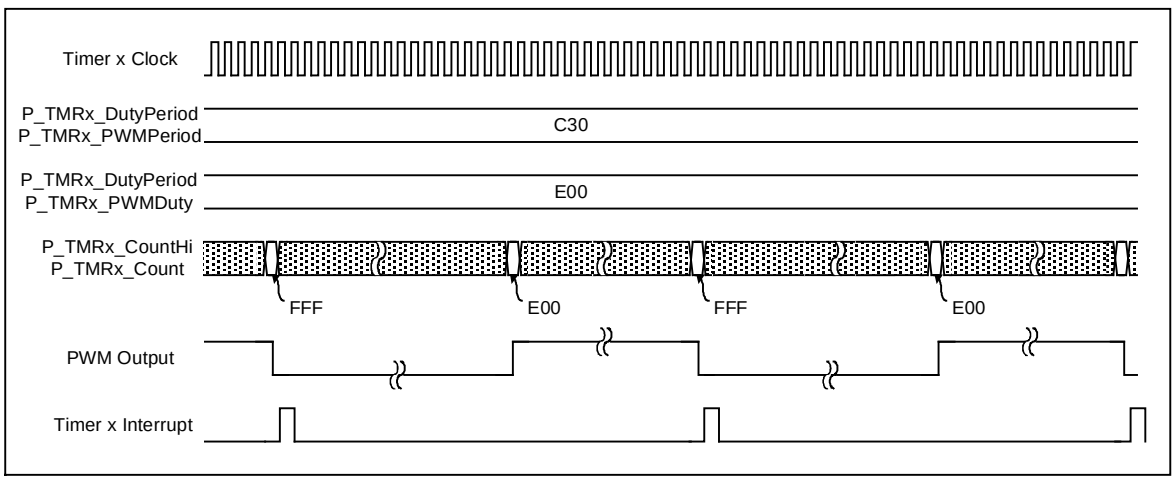


图 12-4 12 位 PWM 操作时序

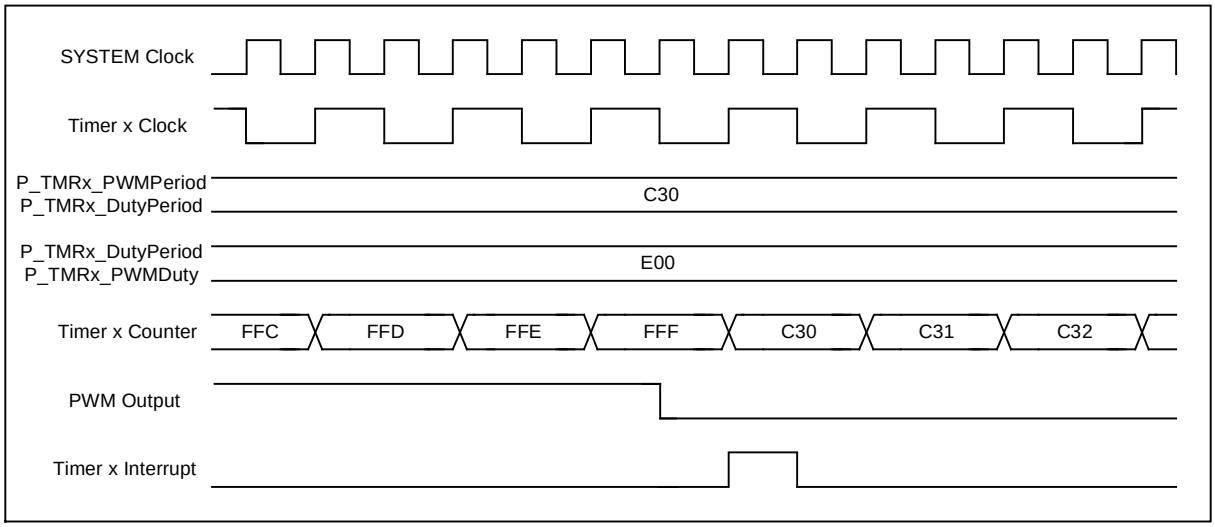


图 **12-5 12** 位 **PWM** 计数寄存器重载时序图

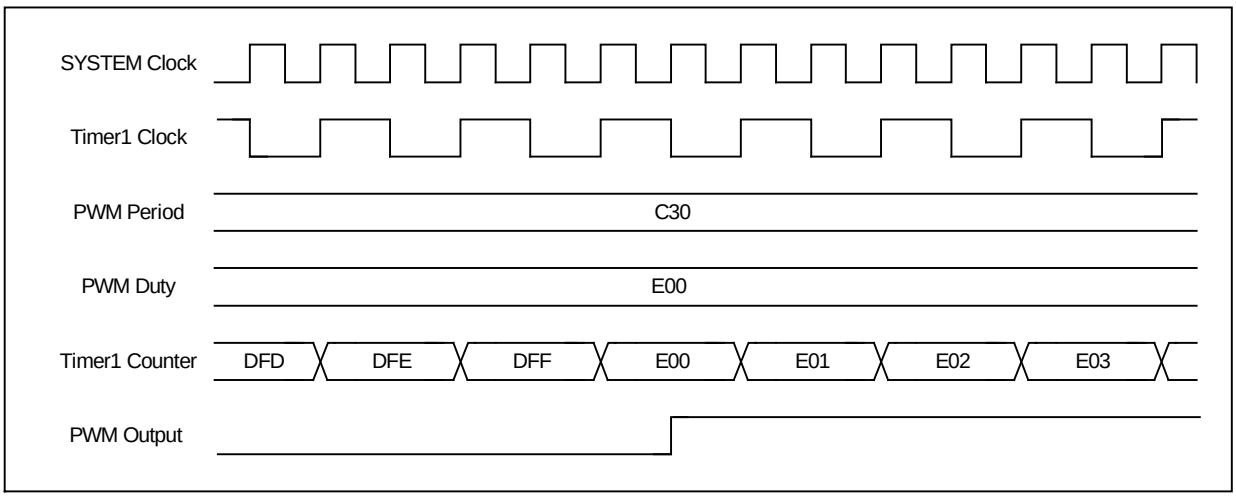


图 12-6 12 位 PWM 波形输出翻转时序图

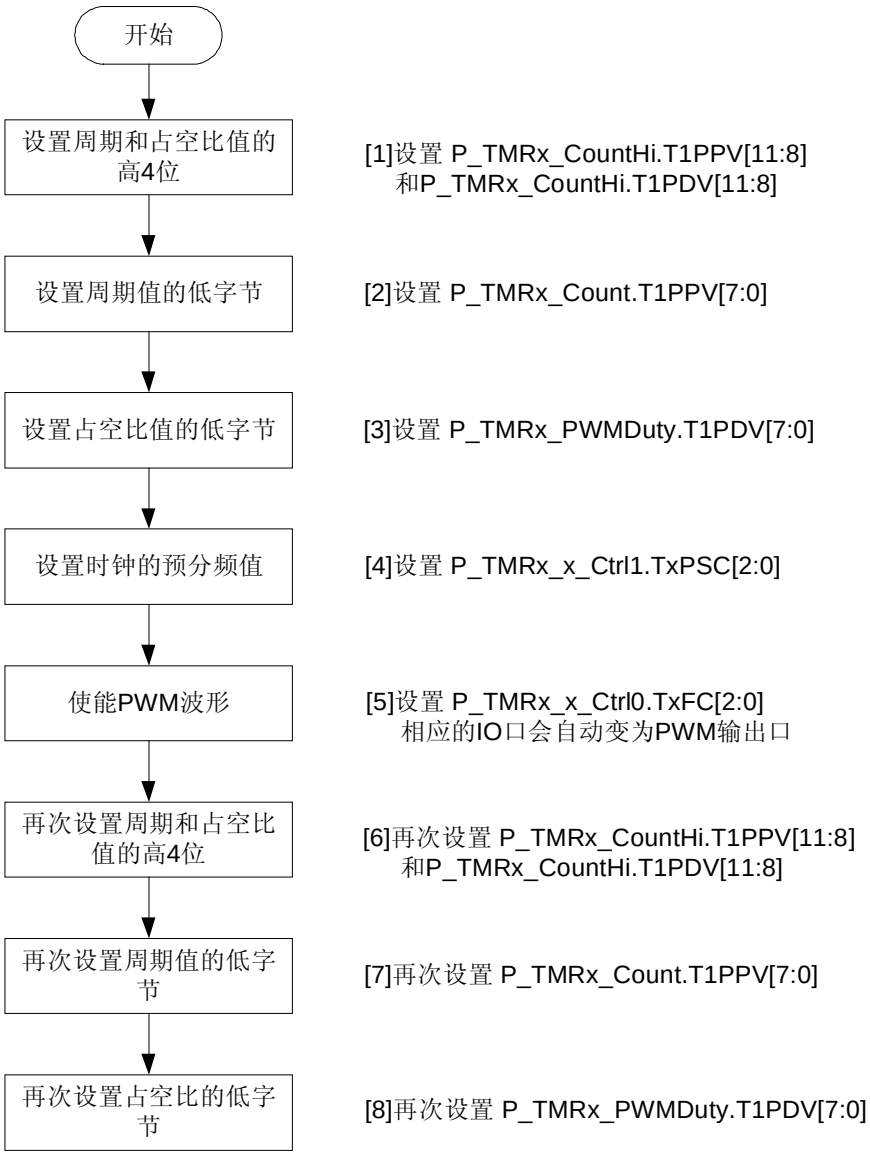


图 12-7 12 位 PWM 设置流程图

公式 12-2: 12 位 PWM 周期和占空比的计算

PRDREG={P_TMRx_DuyPeriod [3:0], P_TMRx_PWMPeriod [7:0]}
DUTYREG={P_TMRx_DuyPeriod [7:4], P_TMRx_PWMDuy [7:0]}
T_TMRx = Timer x Clock Period after pre-scaling

PWM PERIOD=[$1000-PRDREG]* T_TMRx
PWM DUTY=[$FFF-DUTYREG]* T_TMRx
注意： x = 1, 3

寄存器的设置值和实际值之间的关系表 12-7。

表 12-7 12 位 PWM 操作中寄存器的设置值和实际值之间的关系。

PRDREG (Hex)	PWM 周期 (*T_TMRx)		DUTYREG (Hex)	PWM 占空比 (*T_TMRx)	
	Hex	Decimal		Hex	Decimal
$000	$1000	4096	$000	$FFF	4095
$001	$FFF	4095	$001	$FFE	4094
……	……	……	……	……	……
$FFE	$002	2	$FFE	$001	1
$FFF	$001	1	$FFF	$000	0

【例 12 2】:将定时/计数器 1 设置为 12 位 PWM 模式

```
 lda     #$70                ; 启动 Timer1 之前，首先设置占空比和周期寄存器
 sta     P_TMR1_DutyPeriod   ; 设置周期和占空比的高位
 lda     #$00
 sta     P_TMR1_PWMPeriod    ; 设置周期寄存器
 lda     #$FF
 sta     P_TMR1_PWMDuty      ;
 lda     #C_T1FCS_Div_8      ; 设置定时/计数器 1 时钟源为 Fsys/8
 sta     P_TMR0_1_Ctrl1
 lda     #C_T112B_PWM        ; 将定时/计数器 1 设置为 12 位 PWM 模式
 sta     P_TMR0_1_Ctrl0
 lda     #$70
 sta     P_TMR1_DutyPeriod   ; PWM 占空比 $7FF，PWM 周期$1000
 lda     #$00
 sta     P_TMR1_PWMPeriod
 lda     #$FF
 sta     P_TMR1_PWMDuty      ; 在 PB3 管脚产生 244Hz 50%占空比的 PWM 输出
```

12.3.3 16 位 PWM (组合 III)

SPMC65X 系列芯片中，定时器 5 属于第三种(组合 III)—16 位 PWM。当 T5FC [2:0]设置为 111 后，定时/计数器 5 就成为 16 位 PWM 发生器，相应的管脚 PD7 自动成为 16 位 PWM 输出管脚。PWM 周期在寄存器 P_TMR5_PWMPeriodLo ($0023)和 P_TMR5_PWMPeriodHi ($0024)中设置；占空比在寄存器 P_TMR5_PWMDutyLo ($0025)和 P_TMR5_PWMDutyHi ($005F)中设置；时钟的分频系数由 T5PSS [2:0]位选择。当 PWM 使能后，计数器载入计数初值{ P_TMR5_PWMPeriodHi, P_TMR5_PWMPeriodLo}，开始计数。同时，管脚 PD7 输出低电平。在计数过程中，计数值不断的和占空比值{ P_TMR5_PWMDutyHi, P_TMR5_PWMDutyLo}进行比较，相等时，PWM 管脚输出电平翻转，为高电平。计数器继续计数，直到溢出，此时一个周期的方波输出完成。接着，PWM 管脚再次翻转，输出低电平，计数器重新载入计数初值开始计数，产生下一个 PWM 输出，依次不断循环。

一个 8 位 CPU，总线宽度为 8 位，通常无法直接访问 16 位的数据。因此，在 16 位 PWM 模式下，设置 PWM 周期时，需要先写入高字节寄存器 P_TMR5_PWMPeriodHi (MSB byte)并被缓冲器自动保存，然后再写入低字节 P_TMR5_PWMPeriod（LSB byte），直到 16 位数据全部写入，缓冲器的值才会改变，这样缓冲器中的高字节和低字节将会同时载入定时/计数器 5 的相关寄存器中去；同样，在 16 位 PWM 模式下，设置 PWM 占空比时，需要先写入高字节寄存器 P_TMR5_PWMDutyHi (MSB byte)并被缓冲器自动保存，然后再写入低字节 P_TMR5_PWMDutyLo（LSB byte），直到 16 位数据全部写入，缓冲器的值才会改变，这样缓冲器中的高字节和低字节将会同时载入定时/计数器 5 的相关寄存器中去。如图 12-8 所示为 16 位 PWM 输出原理框图。PWM 周期和占空比计算见公式 12-3。

公式 12-3: 16 位 PWM 周期和占空比的计算

```
PRDREG={P_TMR5_PWMPeriodHi, P_TMR5_PWMPeriodLo}
DUTYREG={P_TMR5_PWMDutyHi, P_TMR5_PWMDutyLo}
T_TMR5= Timer5 Clock Period after pre-scaling

PWM 周期=[$10000-PRDREG]* T_TMR5
PWM 占空比=[$FFFF-DUTYREG]* T_TMR5
```

寄存器的设置值和实际值之间的关系见表 12-9。

表 12-9 16PWM 操作中寄存器的设置值和实际值之间的关系。

PRDREG (Hex)	PWM 周期 (*T_TMR5)		DUTYREG (Hex)	PWM 占空比 (*T_MR5)	
	Hex	Decimal		Hex	Decimal
$0000	$10000	65536	$0000	$FFFF	65535
$0001	$FFFF	65535	$0001	$FFFE	65534
……	……	……	……	……	……

$FFFE	$0002	2	$FFFE	$0001	1
$FFFF	$0001	1	$FFFF	$0000	0

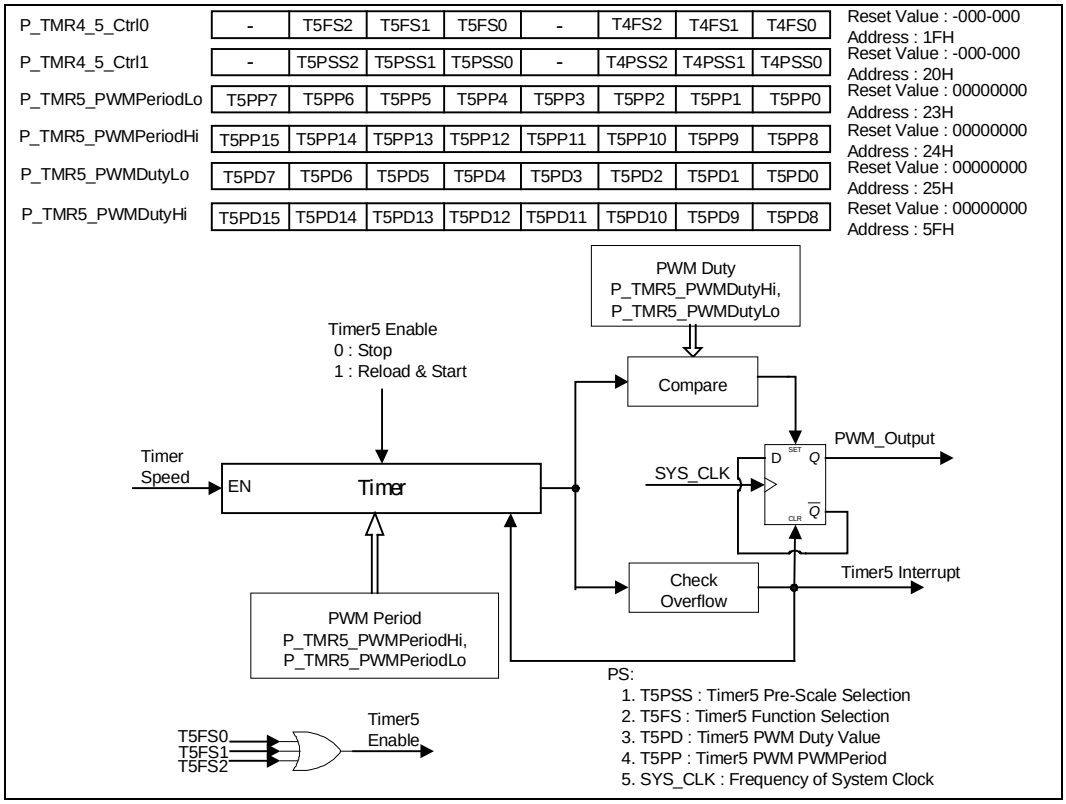


图 12-8 16 位 PWM 结构框图

【例 12 3】:将定时/计数器 5 设置为 16 位 PWM 模式

```
    lda     #$A0                ; 启动 Timer5 之前，首先设置占空比和周期寄存器
    sta     P_TMR5_PWMDutyHi    ; 设置占空比的高位
    lda     #$00
    sta     P_TMR5_PWMDutyLo    ; 设置占空比的低位
    lda     #$80
    sta     P_TMR5_PWMPeriodHi  ; 设置周期的高位
    lda     #$00
    sta     P_TMR5_PWMPeriodLo  ; 设置周期的低位
    lda     #C_T5FCS_Div_1      ; 将 Timer5 时钟源设置为 Fcs/1
    sta     P_TMR4_5_Ctrl1
    lda     #C_T516B_PWM        ; 将定时/计数器 5 设置为 16 位 PWM 功能
    sta     P_TMR4_5_Ctrl0
    lda     #$A0
    sta     P_TMR5_PWMDutyHi
    lda     #$00
    sta     P_TMR5_PWMDutyLo    ; 在 PD7 输出 PWM 波形，占空比 = 75%
                                ;( $FFFF-$A000)/( $FFFF-$8000)~= 75%
```

```
    lda     #$80
    sta     P_TMR5_PWMPeriodHi
    lda     #$00
    sta     P_TMR5_PWMPeriodLo    ; PWM 频率= 8.0MHz / ($FFFF - $8000) ~= 244 Hz
```

12.4 PWM 中断

与 PWM 操作相关的中断如下：

- 定时/计数器 0 周期中断
- 定时/计数器 1 周期中断
- 定时/计数器 2 周期中断
- 定时/计数器 3 周期中断
- 定时/计数器 4 周期中断
- 定时/计数器 5 周期中断

下面是设置 PWM 中断的几个步骤：

- 执行指令“SEI ”关闭总的中断开关；
- 决定采用 8 位还是 12 位或是 16 位的 PWM 方式并选择一个合适的定时/计数器；
- 设置与 PWM 操作相关寄存器；
- 在寄存器 P_INT_Ctrl1($0F)中使能该定时/计数器的 PWM 中断；
- 执行指令”CLI”打开总的中断开关；
- 设置完毕，等待 PWM 中断产生。

12.5 设计参考

【例 12 4】：以 SPMC65P2404A 为例，将 Timer1 初始化为 12 位 PWM 功能，具体步骤如下

- 采用 Timer1 的 12 位 PWM 功能
- 将 Timer1 时钟源设为 Fsys/8
- PWM 输出频率为 244 Hz
- PWM 在 PB3 管脚上输出占空比为 75%

```
.SYNTAX     6502                ; process standard 6502 addressing syntax
.LINKLIST                       ; generate linklist information
.SYMBOLS                        ; generate symbolic debug information

.INCLUDE        spmc65p2404a.inc

    .PAGE0                      ; define values in the range from $00 to $FF
;
    .DATA                       ; define data storage section
;
    .CODE
;=================================
;=                               =
```

```
;= Power on Reset Process - Main Program  =
;=                                        =
;==========================================
V_Reset:
    sei                             ; 关闭总的中断开关
    ldx     #C_STACK_BOTTOM         ; 将堆栈指针初始化为$00FF 位置上
    txs                             ; 将 X 寄存器的值传送到堆栈指针(SP)中
;
    lda     #$FF                    ; 清除复位标志
    sta     P_SYS_Ctrl
    sta     P_SYS_Ctrl

    jsr     F_InitIOPort            ; 初始化 IO 端口;
    lda     #$30                    ; 12-bit PWM 的高字节
    sta     P_TMR1_DutyPeriod
    lda     #$00
    sta     P_TMR1_PWMPeriod        ; PWM  频率= 8.0MHz / (4095 x 8) = 244 Hz;
    lda     #$FF
    sta     P_TMR1_PWMDuty          ; PWM  占空比= (4095 - 1023) / 4095 ~= 75%（PB3）
    lda     #C_T1FCS_Div_8          ; 设置 Timer1 的时钟频率为  Fcs/8
    sta     P_TMR0_1_Ctrl1
    lda     #C_T112B_PWM            ; 将 Timer1 设为 12 位 PWM 模式
    sta     P_TMR0_1_Ctrl0

    lda     #$30                    ; 12-bit PWM 的高字节
    sta     P_TMR1_DutyPeriod
    lda     #$00
    sta     P_TMR1_PWMPeriod        ; PWM  频率= 8.0MHz / (4095 x 8) = 244 Hz;
    lda     #$FF
    sta     P_TMR1_PWMDuty          ; PWM 占空比  = (4095 - 1023) / 4095 ~= 75%
    lda     #$FF                    ; 清除中断请求标志
    sta     P_INT_Flag0
    sta     P_INT_Flag1
    lda     #C_INT_T1OIE            ; 使能 Timer1 溢出中断
    sta     P_INT_Ctrl1
    cli                             ; 打开总的中断开关

L_Main:
    nop
    jmp     L_Main
;
F_InitIOPort:
```

```
;
;          ///       Initial Port A              ///
;
           lda       #00000000B
           sta       P_IOA_Data
           lda       #00000000B
           sta       P_IOA_Attrib
           lda       #11111111B
           sta       P_IOA_Dir
;
           rts
;====================================
;=                                  =
;=         IRQ Interrupt Service Routine        =
;=                                  =
;====================================
V_IRQ:
     pha                           ; 将累加器 A 的值压入堆栈
     txa                           ; 将 X 寄存器的值送入累加器 A 中
     pha                           ; 将累加器 A 的值压入堆栈(即 push X)
;
; Timer 1 溢出中断?
;
     lda       P_INT_Flag1
     and       #C_INT_T1OIF
     beq       L_exit_irq
;
     set       P_INT_Flag1, CB_INT_T1OIF
      lda      #$30                 ; 再次设置周期和占空比的高位
     sta       P_TMR1_DutyPeriod
     lda       #$00                 ; 再次设置周期的低位
     sta       P_TMR1_PWMPeriod     ; PWM  频率= 8.0MHz / (4095 x 8) = 244 Hz
     lda       #$FF                 ; 再次设置占空比的低位
     sta       P_TMR1_PWMDuty       ; 占空比= (4095 - 1023) / 4095 ~= 75%
     lda       P_IOA_Data
     eor       #$FF
     sta       P_IOA_Data           ; PA 口输出数据取反
;
L_exit_irq:
     pla                           ; 将堆栈内容弹出给 A
     tax                           ; 将累加器 A 的值送入 X 寄存器中
     pla                           ; 将堆栈内容弹出给 A
```

```
;
    rti
;
;====================================
;=                                  =
;=       Interrupt Vector Table     =
;=                                  =
;====================================
VECTOR:             .SECTION
    DW      V_NMI               ; may download program emulated either
    DW      V_Reset             ; in internal memory or external memory
    DW      V_IRQ               ; dw define two bytes interrupt vector
;
;====================================
;=                                  =
;=       End Of Interrupt Vector Table  =
;=                                  =
;====================================
    .END                        ; end of program
```

12.6 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题	**应用例系列号**
I/O 模拟音乐输出	AN_O0319.DOC
定时/计数器 0 CCP 功能使用说明	AN_O0331.DOC
定时/计数器 1 CCP 功能使用说明	AN_O0332.DOC
定时/计数器 2 CCP 功能使用说明	AN_O0333.DOC
定时/计数器 3 CCP 功能使用说明	AN_O0334.DOC
定时/计数器 4 CCP 功能使用说明	AN_O0335.DOC
定时/计数器 5 CCP 功能使用说明	AN_O0336.DOC

库函数

标题	**应用例系列号**

SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

12.7 修订记录

版本号	日期	备注
V0.1	03/31/2005	第一版

13 A/D 转换

13.1 简述

SPMC65X 系列芯片内置了一个 10 位 9 通道 ADC，其中一路 ADC 与外部参考电压输入端共用 IO 口。ADC 可以应用于触摸屏、电池电量检测等很多场合。ADC 输入通道和端口 A[7:0]、PB7 复用，并且 PB7 还可以用作外部参考电压的输入管脚。如图 13-1 所示为 AD 转换的原理框图。

可以通过将寄存器 P_AD_Ctrl0 的 ADVRT 位置 0 选择内部电源电压 VDD 作为 AD 转换的参考电压，也可以将 ADVRT 位置 1 从而选择外部参考电压，PB7 为外部参考电压输入管脚。ADC 模块包含了三个控制寄存器：P_AD_Ctrl0、P_AD_Ctrl1 和 P_AD_Ctrl2。寄存器 P_AD_Ctrl0 控制着 AD 转换的使能和转换时钟频率的选择，为满足 AD 硬件速度的需要，AD 转换的时钟频率必须小于 1.4 MHZ。当 P_AD_Ctrl0 中的 STARTB 位被置为“0”时，AD 转换开始。寄存器 P_AD_Ctrl1 负责将端口 PA[7:0]设置为 AD 转换的模拟量输入通道。P_AD_Ctrl2 用于选择当前的 AD 转换通道。AD 转换中断的详细步骤请参看 16.3 节。

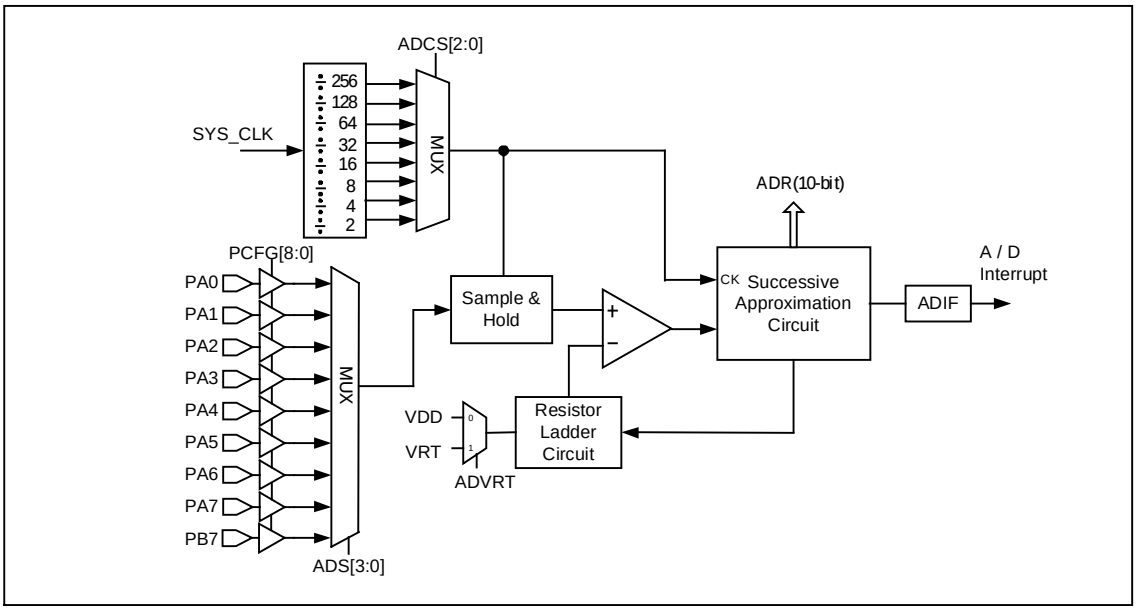


图 13-1 AD 转换原理框图

13.2 控制寄存器

13.2.1 ADC 控制寄存器 0 (P_AD_Ctrl0, $28) (R/W)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
R	ADEN	ADVRT	-	-	ADCS2	ADCS1	ADCS0	ADRDY
W	ADEN	ADVRT	-	-	ADCS2	ADCS1	ADCS0	STARTB
	0	0	0	0	0	0	0	0

Bit 7 ADEN: ADC 使能位

0 = 禁止 AD 转换功能

1 = 允许 AD 转换功能

Bit 6 ADVRT: ADC 参考电压选择位

1 =外部电压(PB7)作为参考电压

0 =内部电源电压(Vdd)作为参考电压

Bit [5:4] 保留

Bit [3:1] ADCS[2:0] : ADC 时钟选择位。

注意：AD 转换的时钟频率不能超过 1.4MHz，以保证转换结果的正确性。

111= F_{SYS} ÷ 256

110= F_{SYS} ÷ 128

101= F_{SYS} ÷ 64

100= F_{SYS} ÷ 32

011= F_{SYS} ÷ 16

010= F_{SYS} ÷ 8

001= F_{SYS} ÷ 4

000= F_{SYS} ÷ 2

F_{SYS}：系统时钟

Bit 0 ADRDY/STARTB: AD 转换状态标志或启动位

读操作：AD 转换的状态

1: AD 转换结束，等待下次转换开始

0: 正在进行数据转换

写操作：AD 转换启动位

1: 无效

0: 启动 AD 转换

13.2.2 ADC 控制寄存器 1 (P_AD_Ctrl1, $29) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PCFG7	PCFG6	PCFG5	PCFG4	PCFG3	PCFG2	PCFG1	PCFG0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] PCFG[7:0]: ADC 通道设置位

这些位用于将 AD 通道的数字管脚设置为模拟管脚。

PCFG7:

1= 模拟输入(AN7)

0= 数字输入(PA7)

PCFG6:

1= 模拟输入(AN6)

0= 数字输入(PA6)

PCFG5:

1= 模拟输入(AN5)

0= 数字输入(PA5)

PCFG4:

1= 模拟输入(AN4)

0= 数字输入(PA4)

PCFG3:

1= 模拟输入(AN3)

0= 数字输入(PA3)

PCFG2:

1= 模拟输入(AN2)

0= 数字输入(PA2)

PCFG1:

1= 模拟输入(AN1)

0= 数字输入(PA1)

PCFG0:

1= 模拟输入(AN0)

0= 数字输入(PA0)

13.2.3 ADC 控制寄存器 2 (P_AD_Ctrl2, $2A) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADCE	ADS3	ADS2	ADS1	ADS0	-	-	PCFG8
R/W	R/W	R/W	R/W	R/W	-	-	R/W
0	0	0	0	0	0	0	0

Bit 7 ADCE: AD 转换电源控制位。当不进行 AD 转换时，关闭 ADC 电源可以省电。

1= 打开 AD 转换电源

0= 关闭 AD 转换电源

Bit [6:3] ADS[3:0]:ADC 当前通道选择位

0000= 选择通道 0 (AN0)

0001= 选择通道 1 (AN1)

0010= 选择通道 2 (AN2)

0011= 选择通道 3 (AN3)

0100= 选择通道 4 (AN4)

0101= 选择通道 5 (AN5)

0110= 选择通道 6 (AN6)

0111= 选择通道 7 (AN7)

1000= 选择通道 8 (AN8)

其余 = 保留

Bit [2:1] 保留

Bit 0 PCFG8:

1= 模拟输入(AN8/Avref)

0= 数字输入(PB7)

注意：当设置这些位时，用户必须注意 PB7 是作为 ADC 的 AN8 通道还是作为外部参考电压的输入端。

13.2.4 A/D 转换结果的高 8 位(P_AD_DataHi, $2B)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADData9	ADData8	ADData7	ADData6	ADData5	ADData4	ADData3	ADData2
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] ADData [9:2]:10 位 AD 转换结果的高 8 位数据。

13.2.5 A/D 转换结果的低 2 位 (P_AD_DataLo, $2C) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADData1	ADData0						
R/W	R/W	-	-	-	-	-	-
0	0	0	0	0	0	0	0

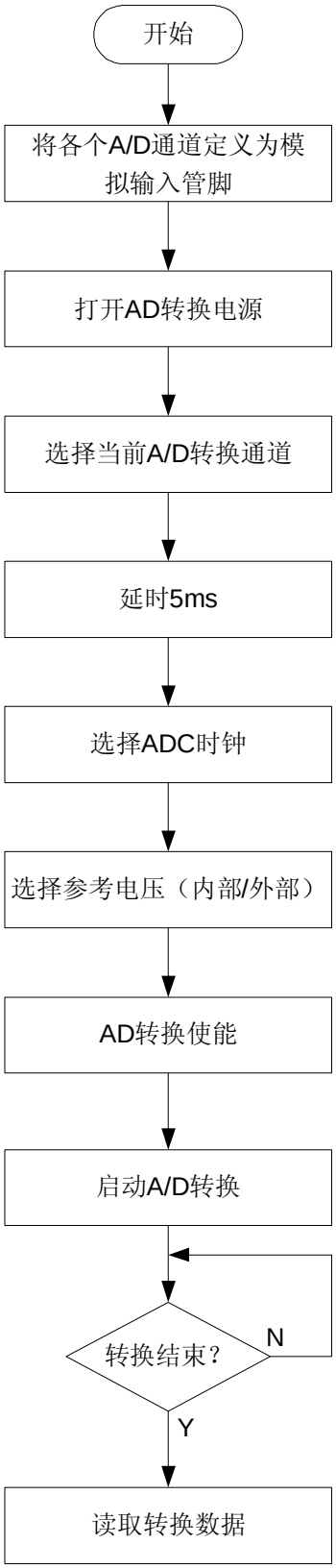
Bit [7:0] ADData [1:0]: 10 位 AD 转换结果的低 2 位数据，bit7 和 bit6 有效。

13.3 ADC 操作

A/D 转换结束后，转换结果保存在寄存器 P_AD_DataHi 和 P_AD_DataLo 中。AD 转换结束有两种判断方式：查询式和中断式，无论应用哪种方式，其结果都会保存在以上这两个寄存器当中，但是注意：在查询方式中，要等到 ARDY 置 1 后转换结果才会存入结果寄存器。

如果 P_INT_Ctrl0 寄存器中的 A/D 转换中断使能位 ADIE 置 1，在 A/D 转换完成后，ADRDY 位和 A/D 中断标志位 ADIF 会置 1，随即发生 A/D 转换中断。

AD 转换时每完成一位所花的时间定义为 T_{AD}，在 SPMC65X 系列芯片中，进行 10 位 A/D 转换总共需要 14 T_{AD}。A/D 转换时钟频率共有 8 种选择，可以通过 P_AD_Ctrl0 寄存器中的 ADCS[2:0]位设定，为了保证转换结果的正确，A/D 转换的时钟频率(T_{AD})应低于 1.4MHZ，以配合 AD 转换硬件部分的要求。注意：初次使用 ADC 功能，在打开 ADC 电源后需要等待 5ms，使其电源稳定下来。如图 13-2 所示，为 AD 转换流程。AD 转换中断的详细步骤见 13.4 节。



[1] 设置 P_AD_Ctrl1.PCFGx

[2] 设置 P_AD_Ctrl2.ADCE=#%1

[3] 设置 P_AD_Ctrl2.ADS[2:0]

[4] 等待ADC电源稳定

[5] 设置 P_AD_Ctrl0.ADCS[2:0]

[6] 设置 P_AD_Ctrl0.ADVRT

[7] 设置 P_AD_Ctrl0.ADEN=#%1

[8] 设置 P_AD_Ctrl0.STARTB=#%0

[9]读 {P_AD_DataHi, P_AD_DataLo}

图 13-2 AD 转换流程

【例 13 1】：使能 AD 转换

```
lda     #C_AD_Pin7                  ; 设置 PC[7] 为模拟输入通道 AN[7]
sta     P_AD_Ctrl1
lda     #(C_AD_CE+ C_AD_Ch7)        ; 选择模拟通道 7 AN7
sta     P_AD_Ctrl2
jsr     Delay5ms                    ;延时 5ms
lda     #(C_AD_EN+C_AD_CS_8+C_AD_RDY)
                                    ; 使能 ADC 功能，ADCclock= Fsys(8MHz)/8= 1MHz(最高)
sta     P_AD_Ctrl0
lda     P_AD_Ctrl0
and     #11111110B                  ; 启动转换
sta     P_AD_Ctrl0
```

13.4 ADC 中断

与 A/D 转换有关的一个中断源如下：

- A/D 转换中断

A/D 转换中断的设置步骤如下：

- 用“SEI”指令关闭总的中断开关；
- 设置与 A/D 转换相关的寄存器；
- 启动 A/D 转换；
- 在 P_INT_Ctrl0 寄存器中将 A/D 转换中断使能位置 1
- 用“CLI”指令打开总的中断开关；
- 等待 AD 转换结束后中断发生。

【例 13 2】:以中断方式进行 AD 转换

```
lda     #C_AD_CE                    ; 使能 ADC 电源
sta     P_AD_Ctrl2
lda     #(C_AD_EN+C_AD_CS_8+C_AD_RDY)
                                    ;使能 ADC 功能，ADCclock= Fsys(8MHz)/8= 1MHz(最高频率)
sta     P_AD_Ctrl0
jsr     Delay5ms                    ;延时 5ms
lda     #(C_AD_Pin0+C_AD_Pin1+C_AD_Pin2+C_AD_Pin3
        +C_AD_Pin4+C_AD_Pin5+C_AD_Pin6+C_AD_Pin7)
                                    ; 将 PC[7:0]设置为模拟通道 AN[7:0]
sta     P_AD_Ctrl1
lda     #(C_AD_CE+C_AD_Ch1) ; 选择模拟通道 1 AN1
```

```
    sta     P_AD_Ctrl2
    lda     P_AD_Ctrl0
    and     #11111110B              ; 启动转换
    sta     P_AD_Ctrl0
    lda     #$FF
    sta     P_INT_Flag0             ;清除中断标志
    lda     #C_INT_ADIE             ; 使能 AD 中断.
    sta     P_INT_Ctrl0
    cli                             ;打开总中断开关
```

13.5 设计参考

【例 13 3] :以芯片 SPMC65P2404A 为例，下面介绍了 AD 转换功能的设置以及转换结果的读取。

- **n** 采用查询式读取转换结果
- **n** AD 转换的时钟频率为 Fcs/8
- **n** PA0 做为模拟电压的输入

```
.SYNTAX     6502                    ; process standard 6502 addressing syntax
.LINKLIST                           ; generate linklist information
.SYMBOLS                            ; generate symbolic debug information

.INCLUDE        spmc65p2404a.inc

    .PAGE0                          ; define values in the range from $00 to $FF
G_AD0_Buf           DS      2       ; AN0 ADC buffer
G_Tempbuf1           DS     1
;
    .DATA                           ; define data storage section
;
    .CODE
;=====================================
;=                                   =
;= Power on Reset Process - Main Program  =
;=                                   =
;=====================================
V_Reset:
    sei                             ; 关闭总的中断开关
    ldx     #C_STACK_BOTTOM         ; 将堆栈指针初始化为$00FF 位置上
    txs                             ; 将 X 寄存器的值传送到堆栈指针(SP)中
```

```
;
      lda       #$FF                ;清除复位标志
      sta       P_SYS_Ctrl
      sta       P_SYS_Ctrl
;
       jsr      F_InitIOPort        ; IO 口初始化


      lda       #C_AD_Pin0
      sta       P_AD_Ctrl1          ; PA0 作为模拟电压的输入
       lda      #(C_AD_CE+ C_AD_Ch0)
      sta       P_AD_Ctrl2          ; 选择通道 0 （AN0）
;
      lda       #(C_AD_EN + C_AD_CS_8)
      sta       P_AD_Ctrl0          ; ADC 使能，ADC 时钟频率＝Fsys / 8

      jsr         Delay5ms          ; 延时 5ms
      lda       P_AD_Ctrl0
      and       #11111110B          ; 开始转换
      sta       P_AD_Ctrl0
;
L_Main:
L_PollADC:
      lda       P_INT_Flag0
      and       #C_INT_ADIF
      beq       L_PollADC
;
      lda       #C_INT_ADIF
      sta       P_INT_Flag0

      lda       P_AD_DataLo
      sta       G_AD0_Buf

      lda       P_AD_DataHi
      sta       G_AD0_Buf + 1
;
      lda       P_AD_Ctrl0
      and       #11111110B          ; 重新开始转换
      sta       P_AD_Ctrl0
      jmp       L_Main

VECTOR:             .SECTION
```

```
    DW      V_NMI                   ; may download program emulated either
    DW      V_Reset                 ; in internal memory or external memory
    DW      V_IRQ                   ; dw define two bytes interrupt vector
;
;=====================================
;=                                   =
;=       End Of Interrupt Vector Table       =
;=                                   =
;=====================================
    .END                                ; end of program
```

13.6 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题	**应用例系列号**
I/O 模拟 A/D 转换	AN_O0316.DOC
ADC 功能使用说明	AN_O0341.DOC

库函数

标题	**应用例系列号**
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

13.7 修订记录

版本号	日期	备注
V0.1	03/31/2005	第一版

14 D/A 转换

14.1 简述

SPMC65X 系列芯片内置了一个 10 位的数字-模拟转换器(DAC)，输出的转换电流最大为 3.3mA，转换完成时间是 80 微秒。当数据写入 DAC 数据寄存器 P_DA_DataHi 和 P_DA_DataLo 中时，DAC 模块自动将其转换为相应的的模拟电流，通过 PE6 管脚输出。向 P_DA_DataHi 寄存器写入“11111111”，向 P_DA_DataLo 寄存器写入“11000000”，PE6 输出最大转换电流 3.3mA。用户可以在 PE6 管脚上串连一个大约 500 欧姆的电阻，将输出电流转换为电压，这种电路可以应用于很多场合，比如家电控制。如图 14-1 所示为 DAC 的原理框图。

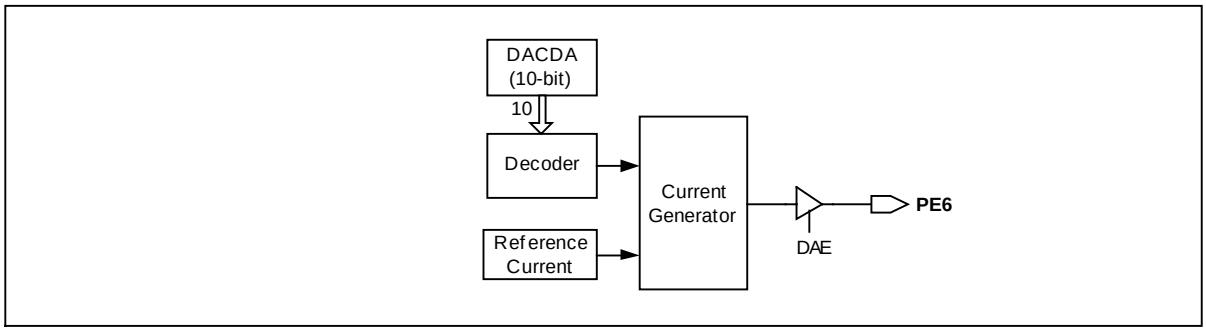


图 14-1 DAC 原理框图

14.2 控制寄存器

14.2.1 DAC 控制寄存器(P_DA_Ctrl, $55) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
DAEN	-	-	-	-	-	-	-
R/W	-	-	-	-	-	-	-
0	0	0	0	0	0	0	0

Bit 7 DAEN: DAC 使能启动位
1=使能并启动 DAC 功能
0=禁止 DAC 功能

Bit [6:0] 保留

14.2.2 DAC 转换数据低 2 位(P_DA_DataLo, $56) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
DAData1	DAData0	-	-	-	-	-	-
R/W	R/W	-	-	-	-	-	-
0	0	0	0	0	0	0	0

Bit [7:6] DAData[1:0]: 10 位 DA 转换结果的低 2 位数据 bit7 和 bit6 有效。

Bit [5:0] 保留

14.2.3 DAC 转换数据高 8 位(P_DA_DataHi, $57) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
DAData9	DAData8	DAData7	DAData6	DAData5	DAData4	DAData3	DAData2
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0]　DAData[9:2]: 10 位 DA 转换结果的高 8 位数据。

14.3 DAC 操作

进行 DA 转换时，需要将寄存器 P_DA_Ctrl 中的 DAEN 位置 1，随即输出转换电流。电流大小由 DAC 数据寄存器中数值大小决定。DAC 数据寄存器包括两个字节：寄存器 P_DA_DataHi 存储转换数据的高 8 位，寄存器 P_DA_DataLo 存储转换数据的低 2 位。DAC 转换完成需要约 80 微秒，用户必须等待本次转换完成才能进行下一次转换以保证 DAC 转换的正确性。

【例 14 1】: DA 转换

```
    lda     #$C0            ; 将转换数据的低 2 位值送入寄存器 P_DA_DataLo 中
    sta     P_DA_DataLo
    lda     #$FF            ; 将转换数据的高 8 位值送入寄存器 P_DA_DataHi 中
    sta     P_DA_DataHi
    lda     #C_DA_EN        ; 使能 DAC 功能
    sta     P_DA_Ctrl
```

14.4 DAC 中断

DAC 转换无中断源。

14.5 设计参考

【例 14 2】：利用 DA 转换产生锯齿波。

```
.SYNTAX   6502                          ; process standard 6502 addressing syntax
.LINKLIST                               ; generate linklist information
.SYMBOLS                                ; generate symbolic debug information

;****************************************
;*                                      *
;*        System Register Define        *
;*                                      *
;****************************************
.INCLUDE        Spmc65.inc
        .PAGE0
G_DAC_Data      DS      2               ; DAC 转换数据，范围：  0 ~ 1023
```

```
G_Temp          DS      1
;
        .DATA                           ; define data storage section
;
        .CODE
;=======================================
;=                                      =
;= Power on Reset Process - Main Program =
;=                                      =
;=======================================
V_Reset:
        sei                             ; 关闭总的中断开关
        ldx     #C_STACK_BOTTOM         ; 将堆栈指针初始化为$00FF 位置上
        txs                             ; 将 X 寄存器的值传送到堆栈指针(SP)中
;
        lda     #$FF                    ; 清除复位标志
        sta     P_SYS_Ctrl
        sta     P_SYS_Ctrl
;
        jsr     F_InitIOPort            ; 初始化 IO 端口
;
        lda     #C_DA_EN
        sta     P_DA_Ctrl               ; 使能 DAC 功能
;
        lda     #$00
        sta     G_DAC_Data
        sta     G_DAC_Data + 1
;
L_Main:
        inc     G_DAC_Data
        beq     L_inc_dachigh
        jmp     L_updata_DAC
;
L_inc_dachigh:
        inc     G_DAC_Data + 1
        lda     G_DAC_Data + 1
        and     #$03
        sta     G_DAC_Data + 1          ; 高位 (G_DAC_Data) = 0 ~ $03
        jmp     L_updata_DAC

L_updata_DAC:                           ; DA 转换数据：从 0 到 1023
        lda     G_DAC_Data
```

```
        asl         A
        asl         A
        asl         A
        asl         A
        asl         A
        asl         A
        sta         P_DA_DataLo                 ; DAData[1:0]
;
        lda         G_DAC_Data
        ror         A
        ror         A
        and         #$3F                        ; DAData[2:7]
        sta         G_Temp
;
        lda         G_DAC_Data + 1
        asl         A
        asl         A
        asl         A
        asl         A
        asl         A
        asl         A                           ; DAData[9:8]

        adc         G_Temp                      ; DAData[9:2]
        sta         P_DA_DataHi
;
        jmp         L_Main
; F_InitIOPort:
;
;           ///       Initial Port A            ///
;
            lda       #00000000B
            sta       P_IOA_Data
            lda       #00000000B
            sta       P_IOA_Attrib
            lda       #11111111B
            sta       P_IOA_Dir
;
            rts
;*****************************************
;*                                       *
;*         Interrupt Vector Table        *
;*                                       *
```

```
;****************************************
VECTOR:                 .SECTION
        DW          V_NMI                       ; may download program emulated either
        DW          V_Reset                     ; in internal memory or external memory
        DW          V_IRQ                       ; dw define two bytes interrupt vector
;
;****************************************
;*                                      *
;*    End Of Interrupt Vector Table     *
;*                                      *
;****************************************
                .END                            ; end of program
```

14.6 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题	**应用例系列号**
I/O 模拟 D/A 转换	AN_O0317.DOC

库函数

标题	**应用例系列号**
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

14.7 修订记录

版本号	日期	备注
V0.1	03/31/2005	第一版

15 比较器

15.1 简述

SPMC65X 系列芯片内置两个电压比较器，这两个比较电压输入端 CMPIN1/CMPIN0 通过比较器控制寄存器的设置(见 15.2 节的描述)与普通 IO 管脚复用，它们可以和来自 PE4/PE2 管脚上的外部参考电压或内部参考电压(1.2V)进行比较。选定了参考电压源和比较电压输入端后就可以使能比较功能，并可以选择打开比较中断。输入电压范围 0.2V~(VDD-0.2) V。如图 15-1 为比较器原理框图。

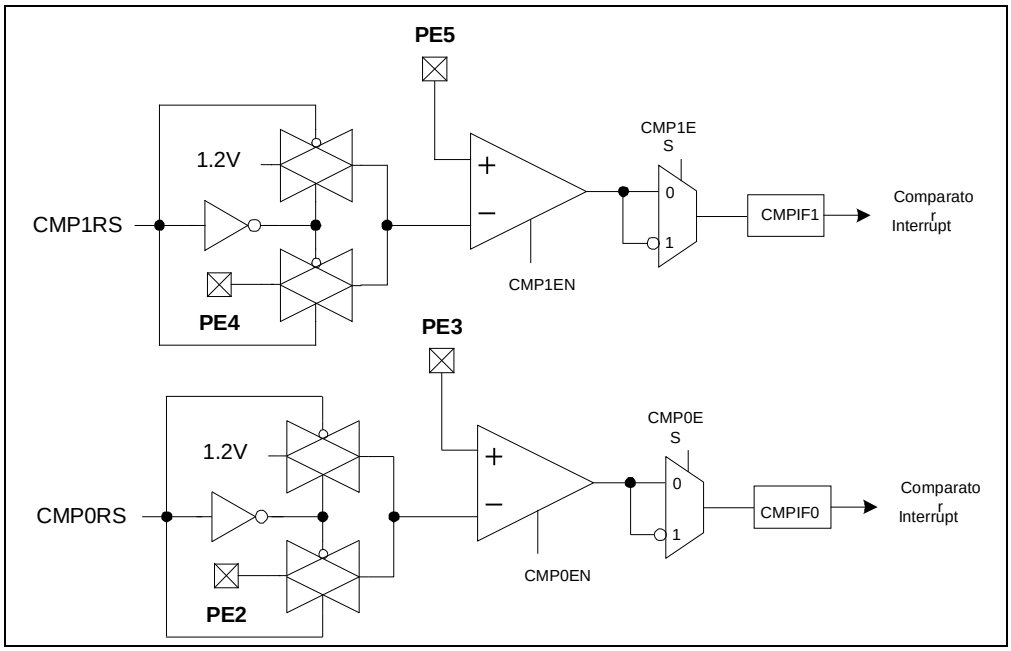


图 15-1 比较器原理框图

15.2 控制寄存器

15.2.1 比较器控制寄存器(P_CMP_Ctrl, $2E) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
CMP1EN	CMP1RS	CMP1LOG	CMP1OUT	CMP0EN	CMP0RS	CMP0LOG	CMP0OUT
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 **CMP1EN**: 比较器 1 使能位
1=使能比较器 1
0=禁止比较器 1

Bit 6 **CMP1RS**:比较器 1 参考电压源选择位
1=外部参考电压(PE4)
0=内部参考电压(1.2v)

Bit 5 **CMP1LOG**:比较器 1 的事件逻辑选择位
1=输入电压 > 参考电压，标志位置 1

0=输入电压 < 参考电压，标志位置 1

Bit 4 **CMP1OUT**: 比较器 1 结果指示位
1=输入电压 > 参考电压
0=输入电压 < 参考电压

Bit 3 **CMP0EN**: **比较器** 0 使能位.
1=使能比较器 0
0=禁止比较器 0

Bit 2 **CMP0RS**: :比较器 0 参考电压源选择位.
1=外部参考电压(PE2)
0=内部参考电压(1.2v)

Bit 1 **CMP0LOG**:比较器 0 的事件逻辑选择位
1=参考电压 > 参考电压，标志位置 1
0=参考电压 < 参考电压，标志位置 1

Bit 0 **CMP0OUT**: 比较器 0 结果指示位
1=参考电压 > 参考电压
0=参考电压 < 参考电压

15.3 操作

这两个比较器都有自己的比较器模块，并且设有独立的控制位：在比较器控制寄存器 P_CMP_Ctrl 中，Bit7~Bit4 分配给比较器 1 使用，Bit3~Bit0 分配给比较器 0 使用。当使能比较器时，CMPxEN(x=0~1)需要置 1。另外，比较器模块还允许选择不同的比较参考电压源：外部参考电压源(来自 PE4/PE2)和内部参考电压源(1.2V)，通过 P_CMP_Ctrl 寄存器种的 CMPxRS(x=0~1)位进行选择。

电压比较时还可以产生中断。CMPxLOG(x=0~1)可以设置在什么情况下发生中断。举个例子：如果 P_INT_Ctrl2 寄存器中的比较器中断使能位 COMxIE(x=0~1)和事件逻辑选择位都被置 1，则当输入电压大于参考电压时，会发生比较器中断，同时 CMPxOUT(x=0~1)位置 1，表明比较的结果。产生比较器中断的详细步骤见 16.4 节的描述。

15.4 比较器中断

比较器中断有两个中断源

- 比较器 0 中断
- 比较器 1 中断

比较器中断设置的步骤如下：

- 用“SEI”指令关闭总的中断开关
- 设置与比较器操作相关的寄存器
- 在寄存器 P_INT_Ctrl2 中打开比较器中断使能位
- 用“CLI”指令打开总的中断开关
- 等待比较结束，中断发生。

15.5 设计参考

【例 15 1】:中断方式初始化电压比较器，步骤如下；

- 使能电压比较器 0
- 使用内部参考电压 1.2V
- 当输入电压大于参考电压时，发生中断

```
.SYNTAX   6502                                  ; process standard 6502 addressing syntax
.LINKLIST                                       ; generate linklist information
.SYMBOLS                                        ; generate symbolic debug information

.INCLUDE        Spmc65.inc


        .PAGE0                                  ; define values in the range from $00 to $FF
;
        .DATA                                   ; define data storage section
;
        .CODE
;=======================================
;=                                             =
;= Power on Reset Process - Main Program       =
;=                                             =
;=======================================
V_Reset:
        sei                                     ; 关闭总的中断开关
        ldx         #C_STACK_BOTTOM             ; 将堆栈指针初始化为$00FF 位置上
        txs                                     ; 将 X 寄存器的值传送到堆栈指针(SP)中
;
        lda         #$FF                        ; 清除复位标志
        sta         P_SYS_Ctrl
        sta         P_SYS_Ctrl
;
        jsr         F_InitIOPort                ; 初始化 IO 端口

        ;lda        #(C_CMP0_EN + C_CMP0_LOG)
        lda         #(C_CMP0_EN)
        sta         P_CMP_Ctrl                  ; 比较器 0 使能，采用内部参考电压 1.2V
                                                ; 当输入电压大于参考电压时，比较器结果指示位置 1

        lda         #C_INT_CMP0IE
        sta         P_INT_Ctrl2                 ; 比较器 0 中断使能
;
        cli                                     ; 打开总的中断开关
;
```

```
L_Main:
        nop
        jmp     L_Main
;
;=======================================
;=                                     =
;= IRQ Interrupt Service Routine       =
;=                                     =
;=======================================
V_IRQ:
        pha                             ; 将累加器 A 的值压入堆栈
        txa                             ; 将 X 寄存器的值送入累加器 A 中
        pha                             ; 将累加器 A 的值压入堆栈(即 push X)
; 发生时基溢出中断?
                                        ;
        lda     P_INT_Flag2
        and     #C_INT_ITVALIF
        beq     L_exit_irq
                                        ;
        lda     #C_INT_ITVALIF
        sta     P_INT_Flag2
                                        ;
        lda     P_IOA_Data
        eor     #$FF
        sta     P_IOA_Data              ; PA 口输出数据取反
                                       ;
L_exit_irq:
        pla                             ; 将堆栈内容弹出给 A
        tax                             ; 将累加器 A 的值送入 X 寄存器中
        pla                             ; 将堆栈内容弹出给 A
        rti
;****************************************
;*                                      *
;*        Interrupt Vector Table        *
;*                                      *
;****************************************
VECTOR:             .SECTION
            DW      V_NMI                   ; may download program emulated either
            DW      V_Reset                 ; in internal memory or external memory
             DW      V_IRQ                   ; dw define two bytes interrupt vector
;****************************************
;*                                      *
```

```
;*    End Of Interrupt Vector Table *
;*                                   *
;*************************************
        .END                                ; end of program
```

15.6 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题	**应用例系列号**
本章无相关应用例	

库函数

标题	**应用例系列号**
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

15.7 修订记录

版本号	日期	备注
V0.1	03/31/2005	第一版

16 时基和蜂鸣器

16.1 简述

SPMC65X 系列芯片配置了一个时基定时器，通过控制寄存器的设置，可以产生从 16us～1s 共 15 种不同的时基，以供编程需要。每当系统时钟(FSYS)输入一个脉冲，8 位时基计数器就加一，当计数器的值与用户设定的时基周期匹配时，发生时基中断。

SPMC65X 系列芯片还有一个蜂鸣器输出，可以输出频率可调、占空比为 50%的方波来驱动蜂鸣器。蜂鸣器输出频率可以通过寄存器 P_BUZ_Ctrl 的 BZFS[3:0]来设定。关于时基和蜂鸣器寄存器的详细设置见 16.2 节。

16.2 控制寄存器

16.2.1 蜂鸣器控制寄存器(P_BUZ_Ctrl, $2D) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
INTIMS3	INTIMS2	INTIMS1	INTIMS0	BZFS3	BZFS2	BZFS1	BZFS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:4] **INTIMS**[3:0]:时基定时长度选择位

0000 = 功能禁止
0001 = F_{SYS} /128
0010 = F_{SYS} /256
0011 = F_{SYS} /512
0100 = F_{SYS} /1024
0001 = F_{SYS} /2048
0110 = F_{SYS} /4096
0111 = F_{SYS} /8192
1000 = F_{SYS} /16384
1001 = F_{SYS} /32768
1010 = F_{SYS} /65535
1011 = F_{SYS} /131072
1100 = F_{SYS} /262144
1101 = F_{SYS} /524288
1110 = F_{SYS} /2^{21}
1111 = F_{SYS} /2^{23}

Bit [3:0] **BZFS**[3:0]:蜂鸣器时钟频率选择位

0000 = 功能禁止
0001 = F_{SYS} /64
0010 = F_{SYS} /128
0011 = F_{SYS} /256
0100 = F_{SYS} /512
0101 = F_{SYS} /1024

0110 = F_{SYS} /2048
0111 = F_{SYS} /4096
1000 = F_{SYS} /8192
1001 = F_{SYS} /4
1010 = F_{SYS} /8
1011 = F_{SYS} /16
1100 = F_{SYS} /32
1101 = F_{SYS} /32
1110 = F_{SYS} /32
1111 = F_{SYS} /32

16.3 操作

16.3.1 时基定时器

时基定时器可以产生 15 种不同的时基供编程需要，通过设置 INTIMS [3:0]可以选择所需时基。详细设置见下表 16.1。如图 16-1 所示为时基定时器运行时序

表 16.1 时基列表

INTIMS [3:0]	时基周期	
	Divisor	F_{T0}= F_{SYS} /1*
0000	-	功能禁止
0001	2^7=128	16us
0010	2^8=256	32us
0011	2^9=512	64us
0100	2^{10}=1024	128us
0101	2^{11}=2048	256us
0110	2^{12}=4096	512us
0111	2^{13}=8192	1.024ms
1000	2^{14}=16384	2.048ms
1001	2^{15}=32768	4.096ms
1010	2^{16}=65535	8.192ms
1011	2^{17}=131072	16.384ms
1100	2^{18}=262144	32.768ms
1101	2^{19}=524288	65.535ms
1110	2^{21}	262.144ms
1111	2^{23}	1s

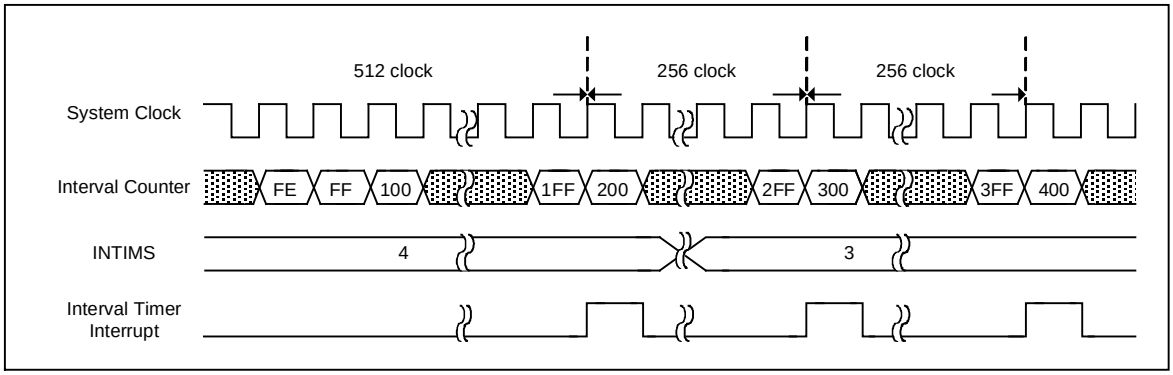


图 16-1 时基定时器运行时序

【例 16 1】：用时基定时器产生周期 512us 的中断

```
lda     #C_TBASE_Div_4k      ；将立即数$60 放入 P_BUZ_Ctrl 寄存器
sta     P_BUZ_Ctrl
```

16.3.2 蜂鸣器

当蜂鸣器输出功能使能时，可以输出频率可调、占空比为 50％的方波来驱动蜂鸣器。蜂鸣器的频率可以通过寄存器 P_BUZ_Ctrl ($2D)的 BZFS[3:0]来进行设置，频率范围 976Hz～2MHz，如表 16-3 所列。

表 16-3 蜂鸣器输出频率列表

BZFS [3:0]	BZO (Buzzer output) rate **(F_{SYS} = 8.0MHz)** 蜂鸣器输出频率	
	分频	F_{T0}= **F_{SYS}** /1*
0000		该功能禁止
0001	64	8us
0010	128	16us
0011	256	32us
0100	512	64us
0101	1024	128us
0110	2048	256us
0111	4096	512us
1000	8192	1.024ms
1001	4	0.5us
1010	8	1us
1011	16	2us
1100	32	4us
1101	32	4us
1110	32	4us
1111	32	4us

用户只需设置寄存器 P_BUZ_Ctrl 便可以产生时基或峰鸣器输出，其中寄存器 P_BUZ_Ctrl 的高四位用于设置时基，低四位用于产生峰鸣器输出。时基定时器可以产生中断，在将寄存器 P_INT_Ctrl2 中的时基使能位(ITVALIE)置 1 并对时基寄存器 P_BUZ_Ctrl 设置好后，便可以等待时基中断的产生。

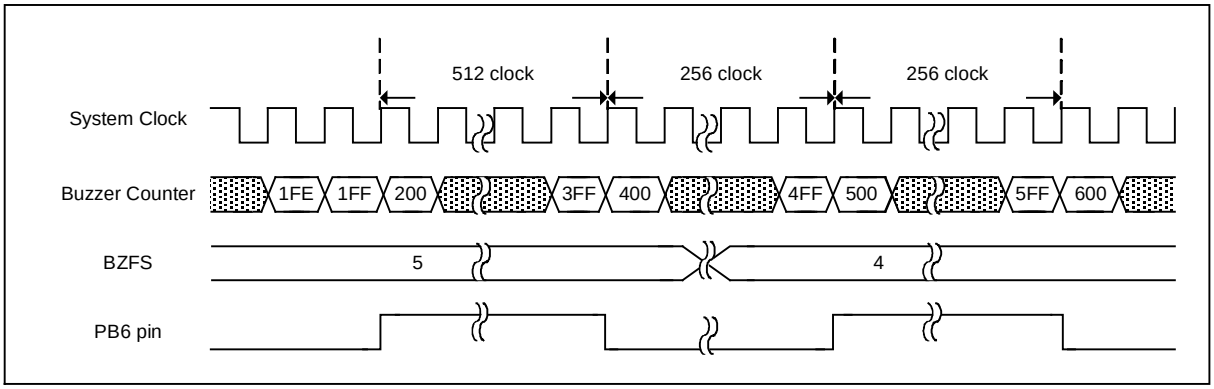


图 16-2 蜂鸣器运行时序

【例 16 2】：用蜂鸣器功能产生脉冲输出，周期 128us，占空比 50%

```
    lda     #C_BUZ_Div_1k           ; 将立即数$05 放入累加器
    sta     P_BUZ_Ctrl
```

16.4 电压比较器模式中断

使用时基定时器产生 512us 的中断

```
    lda     #C_TBASE_Div_4k         ;寄存器 P_BUZ_Ctrl 中存入$60
    sta     P_BUZ_Ctrl
    set     P_INT_Ctrl2, CB_INT_ITVALIE   ; 使能时基中断
    cli                             ;打开总的中断开关
```

16.5 设计参考

【例 16 3】:以 SPMC65P2404A 为例，初始化时基定时器，步骤如下

- 使能时基定时器
- 设置时基定时器为 1.953 KHz

```
.SYNTAX   6502                      ; Process standard 6502 addressing syntax
.LINKLIST                           ; Generate linklist information
.SYMBOLS                            ; Generate symbolic debug information

;****************************************
;*                                      *
;*        System Register Define        *
;*                                      *
;****************************************
.INCLUDE          SPMC65P2404A.inc      ; Define all hardware,Registers and ports.


.PAGE0                                  ; define values in the range from $00 to $FF
```

```
.DATA                                   ; define data storage section

;****************************************
;*                                      *
;*          Program Area                *
;*                                      *
;****************************************
.CODE


.PUBLIC     V_Reset
V_Reset:
        sei                             ; 关闭总的中断开关
        ldx     #C_STACK_BOTTOM         ; 将堆栈指针初始化为$00FF 位置上
        txs                             ; 将 X 寄存器的值传送到堆栈指针(SP)中
;
        lda     #$FF                    ; 清除复位标志
        sta     P_SYS_Ctrl
        sta     P_SYS_Ctrl
;
        jsr     F_InitIOPort            ; 初始化 IO 端口
;
        lda     #C_TBASE_Div_4k
        sta     P_BUZ_Ctrl              ; 时基频率＝8.0MHz / 4096 = 1.953 KHz
        lda     #C_INT_ITVALIE
        sta     P_INT_Ctrl2             ; 使能时基中断
        cli
;
L_MainLoop:
        nop
        jmp   L_MainLoop
;
F_InitIOPort:
                                        ;
;       ///     初始化 Port A           ///
                                        ;
        lda     #00000000B
        sta     P_IOA_Data
        lda     #00000000B
        sta     P_IOA_Attrib
        lda     #11111111B          ;
```

```
        sta     P_IOA_Dir
;
        rts
;****************************************
;*                                      *
;* IRQ Interrupt Service Routine        *
;*                                      *
;****************************************
V_IRQ:
        pha                     ; 将累加器 A 的值压入堆栈
        txa                     ; 将 X 寄存器的值送入累加器 A 中
        pha                     ; 将累加器 A 的值压入堆栈(即 push X)
;
;时基中断?
;
        lda     P_INT_Flag2
        and     #C_INT_ITVALIF
        beq     L_exit_irq
;
        lda     #C_INT_ITVALIF
        sta     P_INT_Flag2
;
        lda     P_IOA_Data
        eor     #$FF
        sta     P_IOA_Data      ; PA 口输出数据取反
;
L_exit_irq:
        pla                     ; 将堆栈内容弹出给 A
        tax                     ; 将累加器 A 的值送入 X 寄存器中
        pla                     ; 将堆栈内容弹出给 A
;
        rti
;****************************************
;*                                      *
;*        Interrupt Vector Table        *
;*                                      *
;****************************************
VECTOR          .SECTION
        DW      V_NMI           ; non-mask interrupt vector(no use)
        DW      V_Reset         ; Reset vector
        DW      V_IRQ           ; IRQ interrupt vector
```

```
;*****************************************
;*                                       *
;* End Of Interrupt Vector Table     *
;*                                       *
;*****************************************
.END                                    ; end of program
```

16.6 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题	**应用例系列号**
时基的使用说明	AN_O0337.DOC

库函数

标题	**应用例系列号**
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

16.7 修订记录

版本号	日期	备注
V0.1	03/31/2005	第一版

17 看门狗定时器

17.1 简述

看门狗定时器的作用是监测程序是否运行正常。如果程序运行正常，在规定的时间内由软件进行一次看门狗清除。倘若程序发生异常，没有在规定的时间内清除看门狗计数器，且看门狗中断为使能的情况下，CPU 发生看门狗中断。如果看门狗中断连续发生 8 次，CPU 会认为程序已经运行异常（跑飞或进入死循环等），从而将整个系统复位并从头开始执行程序。由于软件没有在规定时间内清除看门狗，导致的看门狗定时器中断所引发的 CPU 复位可以保护系统不会执行错误的代码。SPMC65X 系列芯片的看门狗功能可以在 Fortis IDE 中进行设定。看门狗原理图见图 17-1：

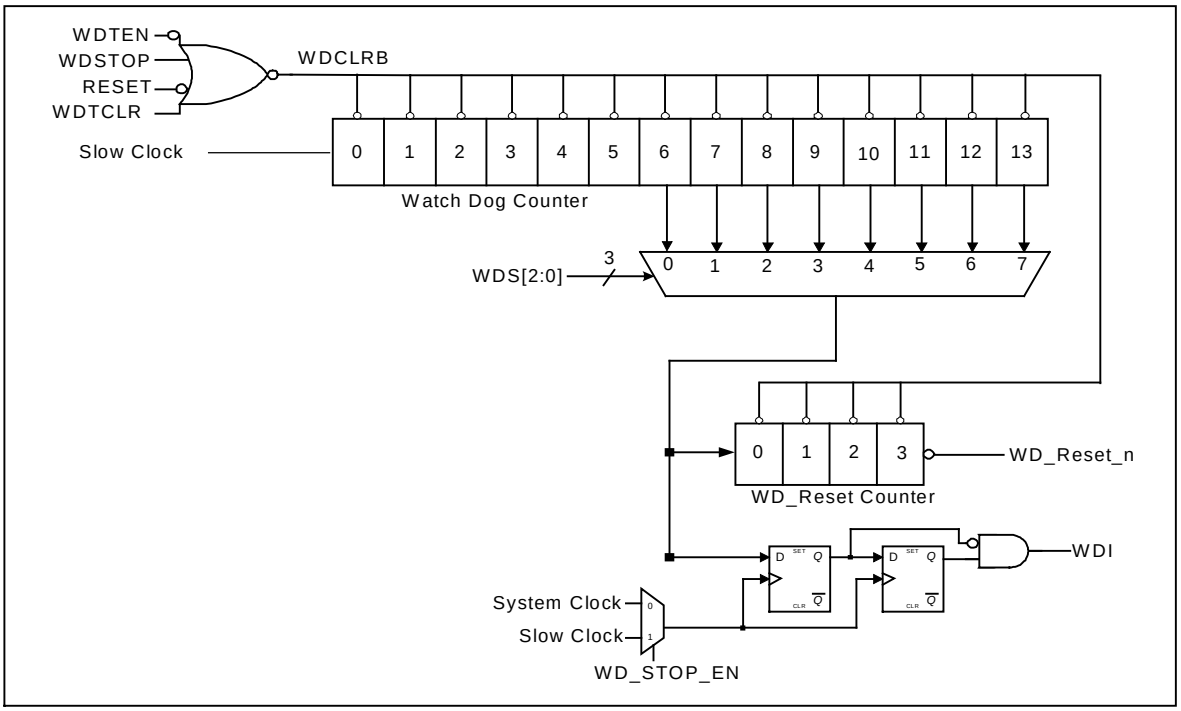


图 17-1 看门狗定时器原理框图

17.2 控制寄存器

17.2.1 看门狗控制寄存器(P_WDT_Ctrl, $32) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
WDTEN	WDCS2	WDCS1	WDCS0	-	-	-	-
R/W	R/W	R/W	R/W	-	-	-	-
1	1	1	1	-	-	-	-

Bit 7 SCKEN: STOP 模式下的看门狗内置 RC 振荡器使能位

1= 使能

0= 禁止

Bit [6:4] WDCS: 看门狗中断时钟选择位

000 = f_slow/128
001 = f_slow/256
010 = f_slow/512
011 = f_slow/1024
100 = f_slow/2048
101 = f_slow/4096
110 = f_slow/8192
111 = f_slow/16384

f_slow: 内置 RC 振荡器，振荡频率为 25kHz

WDS[2:0]	看门狗复位频率(Hz)	看门狗中断频率(Hz)
000	195/8	195
001	97/8	97
010	48/8	48
011	24/8	24
100	12/8	12
101	6/8	6
110	3/8	3
111	1.5/8	1.5

Bit [3:0] 保留

注意：对上述位进行设置时需要连续写两次才能将值写入。

17.2.2 看门狗清除寄存器(P_WDT_Clr, $10) (W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	-	-	-	-	-	-	-
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] WDT_CLR[7:0]:看门狗清除命令

写入：

#$55= 清除看门狗计数器值
其它 = 无效

读出：

总是#$00

注意：

使用看门狗定时器时，用户可以通过寄存器$32(P_WDT_Ctrl)选择适合的频率。

通常，向 P_WDT_Clr 寄存器写入“#$55”可以清除看门狗计数器值。

17.3 操作

看门狗功能可以通过芯片的配置寄存器使能或禁止。使能时，看门狗的定时器溢出就会产生系统复位。用户通过对 P_WDT_Ctrl.WDCS[2:0]的设定，规定溢出的时间即选择看门狗中断

的周期，看门狗计数器必须在这个规定的时间内，用向 P_WDT_Clr 寄存器写 C_WDT_Clr (#$55)的方法清除看门狗计数器，否则，若看门狗中断使能(寄存器 P_INT_Flag0 的 WDIF 位置 1)，就发生了看门狗中断。如果看门狗中断连续发生 8 次，CPU 自动复位，从头开始执行程序。看门狗的时钟源来自芯片中独立的 RC 振荡电路，无需外接组件，其典型频率为 25Khz。注意，使能 RC 振荡器时，在 STOP 模式下可能会引起一些电源消耗，详见第 21 章省电模式。

关于产生看门狗中断的详细步骤见 17.4 节。

【例 17 1】:设置看门狗定时器工作频率为 1.5Hz

```
lda     #C_WDT_Div_16384        ; WDI= Fslow(25KHz)/16384= 1.5Hz
sta     P_WDT_Ctrl
sta     P_WDT_Ctrl
```

17.4 看门狗中断

与看门狗操作有关的中断只有一个：.

- 看门狗中断

使能看门狗中断的设置步骤如下：

- 在 Fortis IDE 中使能看门狗中断；
- 用“SEI”指令关闭总的中断开关；
- 设置与看门狗操作相关的寄存器，如 P_INT_Ctrl0 中的 WDT 使能位和 P_WDT_Ctrl 中的 WDT 周期；
- 用“CLI”指令打开总的中断开关；
- 等待中断产生。

【例 17 2】:使能看门狗中断

```
lda     #$FF
sta     P_INT_Flag0         ; 清除中断标志
lda     #C_INT_WDIE         ; 使能看门狗中断
sta     P_INT_Ctrl0
cli                         ; 打开总的中断开关
```

【例 17 3】:为了尽量节省耗电，在 stop 模式下禁止看门狗时钟(25Khz)。

```
clr     P_WDT_Ctrl,7
clr     P_WDT_Ctrl,7            ;
```

17.5 设计参考

【例 17 4】:.以 SPMC65P2404A 为例，初始化看门狗，设置看门狗中断频率为 F_slow/512 (48.828 Hz)

```
.SYNTAX     6502                ; process standard 6502 addressing syntax
.LINKLIST                       ; generate linklist information
.SYMBOLS                        ; generate symbolic debug information

.INCLUDE        spmc65p2404a.inc

    .PAGE0                      ; define values in the range from $00 to $FF
;
    .DATA                       ; define data storage section
;
    .CODE
;=====================================
;=                                   =
;= Power on Reset Process - Main Program  =
;=                                   =
;=====================================
V_Reset:
    sei                         ; 关闭总的中断开关
    ldx     #C_STACK_BOTTOM     ; 将堆栈指针初始化为$00FF 位置上
    txs                         ; 将 X 寄存器的值传送到堆栈指针(SP)中
;
    lda     #$FF                ; 清除复位标志清除复位标志
    sta     P_SYS_Ctrl
    sta     P_SYS_Ctrl
;
    jsr     F_InitIOPort        ; 初始化 IO 端口初始化 IO 口
;
    lda     #C_WDT_Div_512      ; 看门狗中断频率＝F_slow/512 = 48.828 Hz
    sta     P_WDT_Ctrl
    sta     P_WDT_Ctrl

    lda     #$FF
    sta     P_INT_Flag0         ; 清除中断标志
    sta     P_INT_Flag1
    lda     #C_INT_WDIE         ; 使能看门狗中断
    sta     P_INT_Ctrl0
    cli                         ; 打开总的中断开关
;
L_Main:
    nop
    jmp     L_Main
;
```

```
F_InitIOPort:
;
;          ///       初始化  Port A              ///
;
           lda       #00000000B
           sta       P_IOA_Data
           lda       #00000000B
           sta       P_IOA_Attrib
           lda       #11111111B
           sta       P_IOA_Dir
;
           rts
;====================================
;=                                  =
;=         IRQ Interrupt Service Routine        =
;=                                  =
;====================================
V_IRQ:
     pha                           ; 将累加器 A 的值压入堆栈
     txa                           ; 将 X 寄存器的值送入累加器 A 中
     pha                           ; 将累加器 A 的值压入堆栈(即 push X)
;
; WDT 溢出中断?
;
     lda       P_INT_Flag0
     and       #C_INT_WDIF
     beq       L_exit_irq
;
     lda       #C_INT_WDIF
     sta       P_INT_Flag0
;
     lda       P_IOA_Data
     eor       #$FF
     sta       P_IOA_Data          ; PA 口输出数据取反
;
L_exit_irq:
     pla                           ; 将堆栈内容弹出给 A
     tax                           ; 将累加器 A 的值送入 X 寄存器中
     pla                           ; 将堆栈内容弹出给 A
;
     rti
;
```

```
;=====================================
;=                                   =
;=        Interrupt Vector Table     =
;=                                   =
;=====================================
VECTOR:             .SECTION
    DW      V_NMI               ; may download program emulated either
    DW      V_Reset             ; in internal memory or external memory
    DW      V_IRQ               ; dw define two bytes interrupt vector
;
;=====================================
;=                                   =
;=        End Of Interrupt Vector Table  =
;=                                   =
;=====================================
    .END                        ; end of program
```

17.6 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题	**应用例系列号**
看门狗定时器使用说明	AN_O0340.DOC

库函数

标题	**应用例系列号**
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

17.7 修订记录

版本号	日期	备注
V0.1	03/31/2005	第一版

18 SPI

18.1 简述

SPI(Serial Peripheral Interface)即串行外设接口，它是一种高速同步的串行接口，可以串行接收或发送数据，支持主从机之间的全双工同步传输，传输速率可编程设定。在 SPMC65X 系列芯片中，SPI 模块有 4 个管脚，每款芯片同时支持主模式和从模式。许多与 SPI 有关的参数都可编程设定，如运行模式、工作时钟频率、时钟相位和极性等。SPI 模块具有以下特性：

- 四个接口管脚

  SDO：数据输出管脚 (与 PC3 复用)

  SDI：数据输入管脚 (与 PC2 复用)

  SCK：时钟输入/输出管脚 (与 PC1 复用)

  SSB：从机选择管脚 (与 PC0 复用)
- 支持全双工同步传输
- 两种工作模式：主模式、从模式
- 波特率：8 种可编程传输速率，CPU 时钟在 8 MHz 时，最大可达 2Mbps
- 每次发送或接收的数据长度：8 位
- 时钟相位和极性的可编程设定
- 数据采样时刻选择：可在数据输出中或数据输出末尾进行采样
- SPI 接收/发送缓冲器大小为 1 个字节

下图是 SPI 模块功能框图。

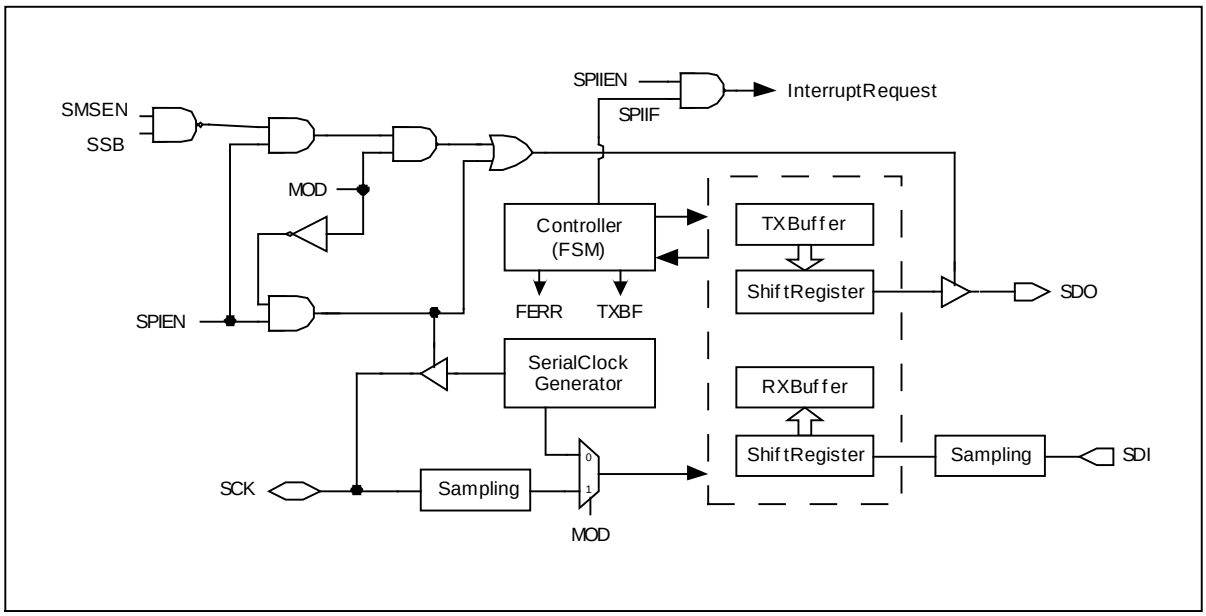


图 18-1 SPI 模块功能框图

18.2 应用电路

SPI 接口经常用于和 EEPROM 进行通讯。如图 18-2 所示为 SPMC65X 系列芯片与 93C46 通讯的应用电路。

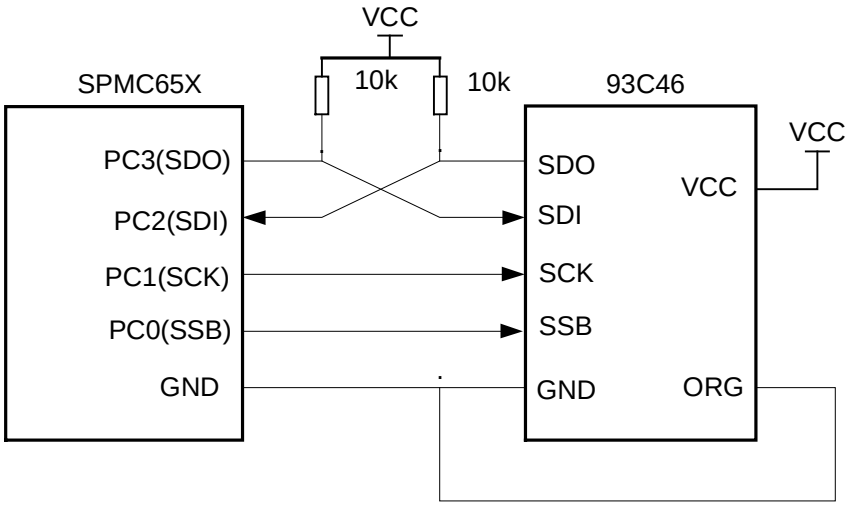


图 18-2 SPI 工作的应用电路

18.3 控制寄存器

18.3.1 SPI 控制寄存器 0 (P_SPI_Ctrl0, $38)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SPIEN	MOD	SCKPHA	SCKPOL	SMS	SCKSEL2	SCKSEL1	SCKSEL0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 **SPIEN**: SPI 使能位。该位置 1 后 PC[3:0]即作为 SPI 通讯接口。
1= 使能 SPI 功能
0= 禁止 SPI 功能

Bit 6 **MOD**: SPI 运行模式
1= 从机模式
0= 主机模式

Bit 5 **SCKPHA**: SPI 时钟相位选择，见 SPI 主模式时序。

Bit 4 **SCKPOL**:SPI 时钟极性选择，见 SPI 主模式时序。

Bit 3 **SMS**: 主机采样模式选择
1=输入数据输出末尾进行采样
0=输入数据输出中进行采样

Bit [2:0] **CKSEL** [2:0]: 主模式时钟选择位
111= F_{SYS} ÷ 128
110= F_{SYS} ÷ 128
101= F_{SYS} ÷ 128
100= F_{SYS} ÷ 64
011= F_{SYS} ÷ 32

010= F_{SYS} ÷ 16
001= F_{SYS} ÷ 8
000= F_{SYS} ÷ 4
FSYS：系统时钟频率

18.3.2 SPI SPI 控制寄存器 1(P_SPI_Ctrl1, $39)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SMSEN	SWRST	-	-	-	-	SPISPCLK1	SPISPCLK0
R/W	R/W	-	-	-	-	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 **SMSEN**: SPI 从模式选择输入
1= PC0 作为从模式选择位，低有效
0= PC0 作为通用 IO 管脚

Bit 6 **SWRST**: SPI 软件复位

写：
1=产生脉冲复位 SPI 模块（寄存器设置除外）
0=无效
读：总为 0
Bit [5:2] 保留
Bit [1:0] **SPISPCLK** [1:0]: 采样时钟选择位
11= F_{SYS} ÷ 4
10= F_{SYS} ÷ 2
01= F_{SYS}
00=不采样
F_{SYS}：系统时钟频率
注意：采样时钟的目的是为了防止在接收数据时尖脉冲的干扰，但是低的采样率会影响通信速度。

18.3.3 SPI 状态寄存器(P_SPI_Status, $3A)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SPIIF	SPIIEN	TXBF	0	0	0	0	BUFFull
R/W	R/W	R	-	-	-	-	R
0	0	0	0	0	0	0	0

Bit 7 SPIIF: SPI 中断标志
读：
1=SPI 中断标志置 1
0=SPI 中断标志为 0
写：

1=清除标志
0=无效

Bit 6 SPIIEN: SPI 中断使能位
1=使能
0=禁止

Bit 5 TXBF: 传输缓冲器满标志位
1= 传输缓冲器满
0= 传输缓冲器空

Bit [4:1] 保留

Bit 0 BUFFull: 缓冲器满标志位，若满，新数据覆盖缓冲器里的值
1= 覆盖
0= 缓冲器工作正常

18.3.4 SPI 发送缓冲器 0 (P_SPI_TxData, $3B)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SPITXDATA7	SPITXDATA6	SPITXDATA5	SPITXDATA4	SPITXDATA3	SPITXDATA2	SPITXDATA1	SPITXDATA0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] SPITXDATA: SPI 发送数据
读：总为#0
写：发送数据

18.3.5 SPI 接收缓冲器 0 (P_SPI_RxData, $3C)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SPIRXDATA7	SPIRXDATA6	SPIRXDATA5	SPIRXDATA4	SPIRDATA3	SPIRXDATA2	SPIRXDATA1	SPIRXDATA0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] SPIRXDATA: SPI 接收数据
读：SPI 接收数据
写：无效果

18.4 SPI 运行

18.4.1 主模式

在主模式下，移位时钟(SCK)由 SPI 模块产生。在寄存器 P_SPI_Ctrl0 中有两个控制时钟相位(SCKPHA)和极性(SCKPOL)的控制位。当向 P_SPI_TxData 寄存器写入数据时，发送立即开始。另外如果用户想要接收从机数据，SDI 管脚(PC2)可以设定为悬浮输入。SPMC65X 系列芯片支持 SPI 模块的接收和发送中断。在寄存器 P_SPI_Status 中将 SPIIEN 位置 1 可同时使能收发中断。同样，在收发中断服务程序结束后需要清除 SPIIF 标志位。

在软件向寄存器 P_SPI_TxData 写入一个字节之后，数据被锁存到内部发送缓冲中。如果此时移位寄存器是空的，该数据将被载入到移位寄存器中并在下一个 SCK 相位时开始传输。如果移位寄存器正在执行数据移位(由 P_SPI_TxStatus 寄存器中的 TXBF 标志得知)，新数据会等待当前的数据移出后才可以载入进行移位。

SPI 通过 SDO 管脚将数据从最高有效位(MSB)开始移出，直到最低有效位(LSB)也被移出。8 位数据在 8 个 SCK 周期后全部移出。同时，接收的数据也通过 SDI 引脚移入。当每组 8 位发送完成后，P_SPI_Status 寄存器中的 SPIIF 置位；此外，如果寄存器 P_SPI_Status 中的 SPIIEN 位被设置为‘1’，会产生一个 SPI 发送中断。

相反的，当 SPI 接口成功地接收了一组 8 位数据时，接收到的数据将被锁存到接收缓冲器中。此时，P_SPI_Status 寄存器中的 SPIIF 位将被设置为‘1’，如果 P_SPI_Status 寄存器中的 SPIIEN 位被设置为‘1’，会产生一个 SPI 接收中断。

下图给出了 SPI 主机模式下不同运行类型的时序(极性控制位等于“1”或“0”，相位控制位等于“1”或“0”，取样控制位等于“1”或“0”)

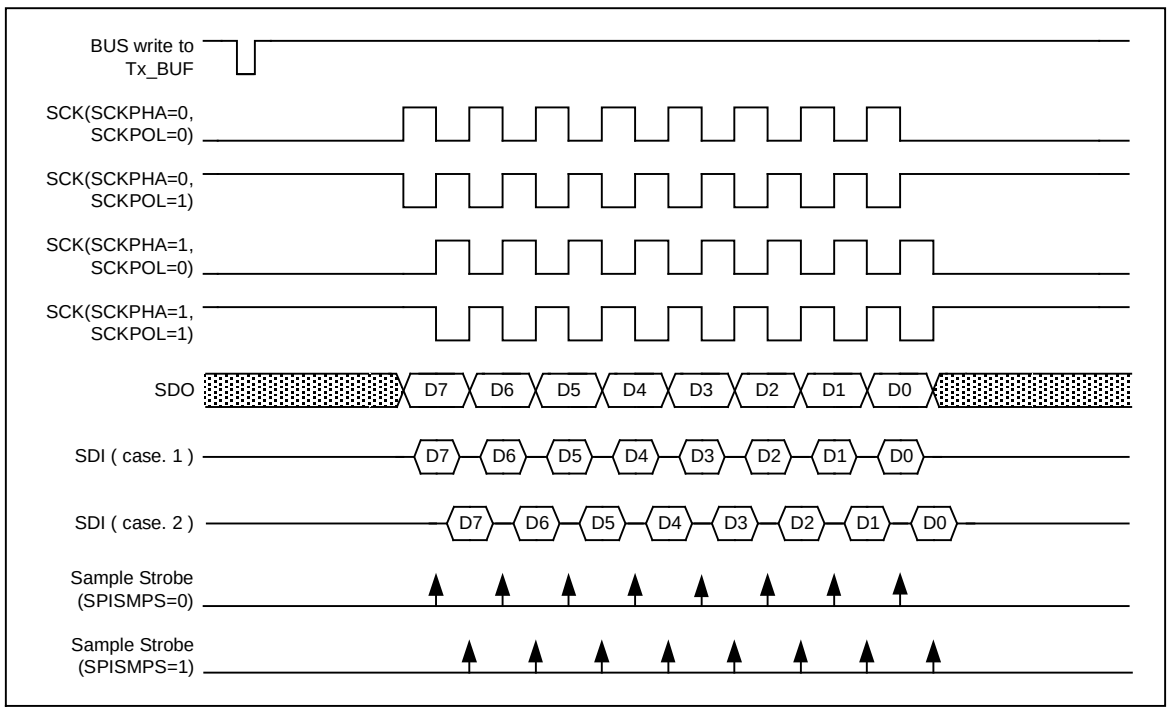


图 18-3 SPI 模式时序图，主机模式

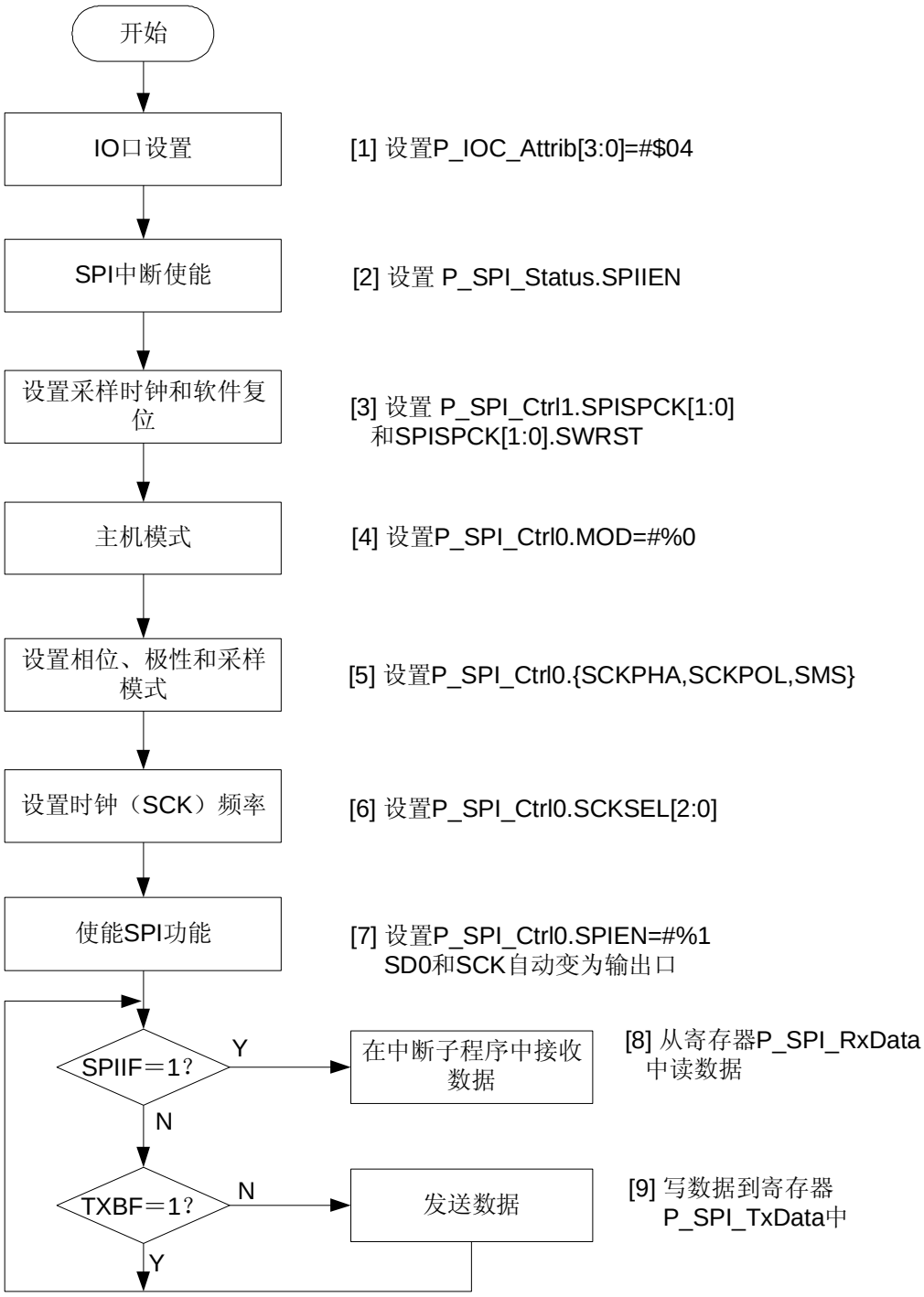


图 18-4 SPI 主机模式软件流程图

【例 18 1】：SPI 在主模式下运行(初始化)

```
lda     #00000000B
sta     P_IOC_Data
lda     #00000100B                  ; PC2 设置为悬浮式输入(SDI)
sta     P_IOC_Attrib
lda     #11111011B
sta     P_IOC_Dir

lda     #(C_SPI_INTEN + C_SPI_INTIF)
sta     P_SPI_Status                ; 使能 SPI 中断，清除中断标志位

lda     #(C_SPI_SWRST + C_SPISPC_Div_4)
sta     P_SPI_Ctrl1                 ; 采样时钟 = Fsys/1, 软件复位

lda     #(C_SPI_EN + C_SPI_SCKPHA + C_SPICS_Div_32 + C_SPI_SPISMPS)
sta     P_SPI_Ctrl0                 ; Fsys/32, PHA = 1, 主机模式
```

18.4.2 从模式

在从机模式下，移位时钟 SCK 来自外部 SPI 主设备，所以第一个外部时钟周期开始传输。发送前，软件应在第一个来自主设备的 SCK 之前向其发送缓冲写入数据。主设备与从设备都必须按相同的 SCK 相位和极性运行，以进行数据的发送与接收。

如果时钟相位(SCKPHA)为“1”，只要向 P_SPI_TxData 寄存器写入数据，就开始移出的第一个数据位。如果时钟相位(SPIPHA)为“0”，则在第一个 SCK 边沿后才开始移出的第一个数据位。

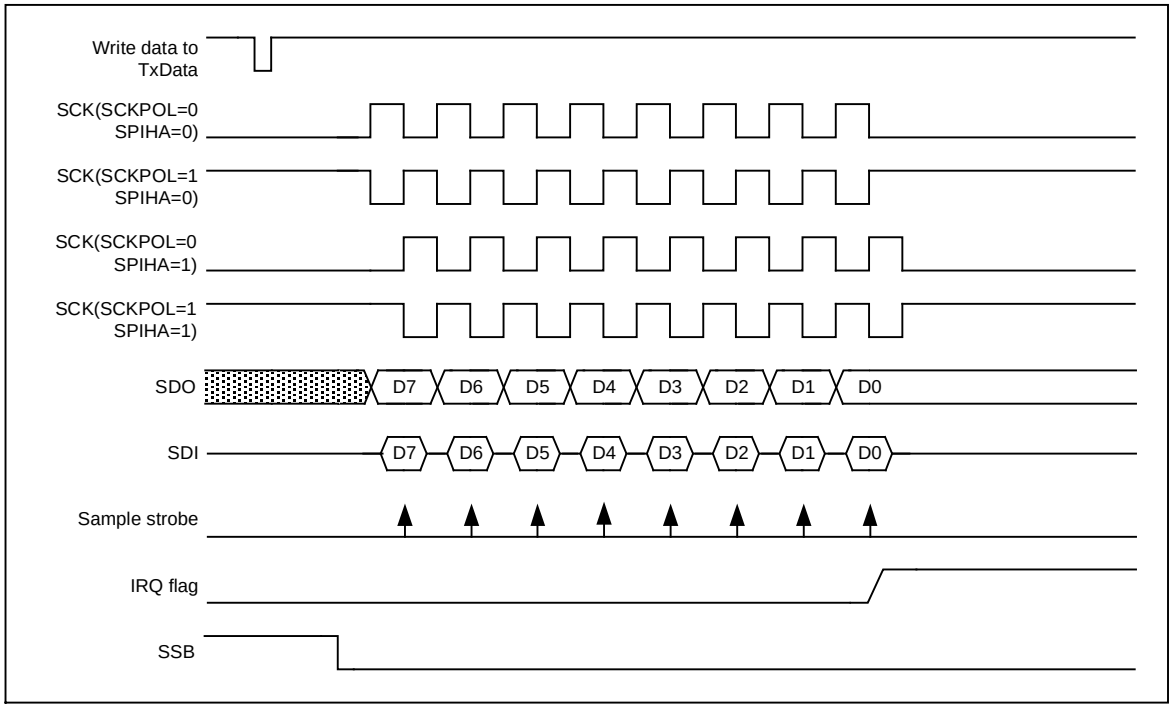


图 18-5 SPI 模式时序图，从机模式

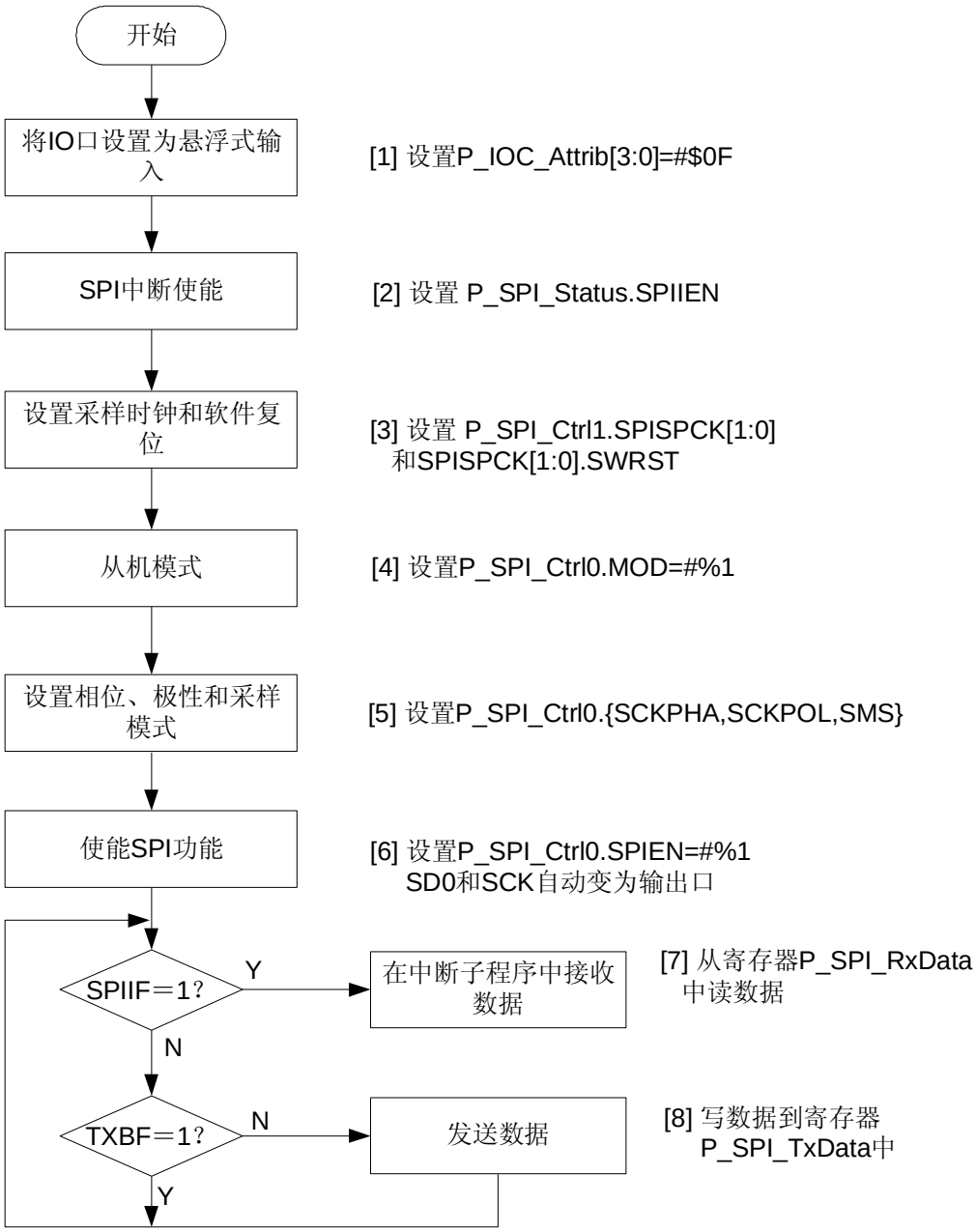


图 18-6 SPI 从机模式软件流程图

【例 18 2】: SPI 在从模式下运行(初始)

```
lda     #00000000B
sta     P_IOC_Data
lda     #00001111B          ; PC0~PC3 设置为悬浮式输入
sta     P_IOC_Attrib
lda     #11110000B          ;
sta     P_IOC_Dir
```

```
    lda     #(C_SPI_INTEN + C_SPI_INTIF)
    sta     P_SPI_Status                ;使能 SPI 中断，清除中断标志位

    lda     #(C_SPI_SWRST + C_SPISPC_Div_4)
    sta     P_SPI_Ctrl1                 ; 采样时钟 = Fsys/1, 软件复位

    lda     #(C_SPI_EN+C_SPI_MOD+C_SPI_SCKPHA+C_SPICS_Div_32+C_SPI_SPISMPS)
    sta     P_SPI_Ctrl0                 ; Fsys/32, PHA = 1, 从机模式
```

18.5 SPI 中断

与 SPI 相关的中断有两个：SPI 发送中断和接收中断。因为 SPI 为全双工通讯，因此这两个中断同时发生，并且共享一个中断标志位。

- SPI 发送中断
- SPI 接收中断

设置 SPI 中断需要以下步骤：

- 用“SEI”指令关闭总的中断开关；
- 使能 SPI 模块，配置相关寄存器；
- 在 P_SPI_Status($3A)寄存器中将 SPIIEN 位置 1，使能 SPI 中断；
- 用“CLI”指令打开总的中断开关；
- 等待中断产生

18.6 设计参考

【例 13-1】SPI 采样时钟选择和 IO 设置公式

1. SPI 时钟计算公式

用下面的公式计算采样时钟，以保证获得正确的数据。

采样时钟> 4 X SPI 时钟（主机模式）

系统时钟> 4 x SPI 时钟 x SPISPCLK (从机模式)

例：

SPI时钟频率=1MHz(PC1), F_{SYS}=8MHz

采样时钟频率 >= 4X1MHz = 4MHz

采样时钟必须大于 4Mhz 才能保证从机数据接收正确。

因此，采样时钟选择位(SPISPCLK [1:0])宜设置为 2

采样时钟频率=8MHz/2=4Mhz

2. SPI 运行在主模式下端口 C 的设置

PC0(SSB):输出

PC1(SCK): 输出

PC2(SDI): 悬浮式输入

PC3(SDO): 输出

3. SPI 运行在从模式下端口 C 的设置

```
PC0(SSB): 悬浮式输入
PC1(SCK): 悬浮式输入
PC2(SDI): 悬浮式输入
PC3(SDO): 悬浮式输入
```

【例 18 3】：以 SPMC65P2404A 为例，将 SPI 初始化为主机模式，采用查询方式，步骤如下：

- **n** 将 SPI 设置成主机模式
- **n** 通过查询，发送数据

```
.SYNTAX   6502                          ; process standard 6502 addressing syntax
.LINKLIST                               ; generate linklist information
.SYMBOLS                                ; generate symbolic debug information

.INCLUDE        Spmc65.inc


        .PAGE0                          ; define values in the range from $00 to $FF
G_Tempbuf1      DS      1       ;
G_Tempbuf2      DS      1
G_MWorkReg1     DS      1
G_MWorkReg2     DS      1       ;
G_TXbuf         DS      7       ;
;
        .DATA                           ; define data storage section
;
        .CODE
;****************************************
;*                                      *
;*        Program Area                  *
;*                                      *
;****************************************

V_Reset:
        sei                             ; 关闭总的中断开关
        ldx     #C_STACK_BOTTOM ; 将堆栈指针初始化为$00FF 位置上
        txs                             ; 将 X 寄存器的值传送到堆栈指针(SP)中
;
        lda     #$FF                    ; 清除复位标志清除中断标志位
        sta     P_SYS_Ctrl
        sta     P_SYS_Ctrl
;
;       ///     初始化 Port C           ///
```

```
;
        lda             #00000000B
        sta             P_IOC_Data
        lda             #00000100B                  ; PC2  设置为悬浮式输入(SDI)
        sta             P_IOC_Attrib
        lda             #11111011B
        sta             P_IOC_Dir
        ldy             #00

        lda             #C_SPI_INTIF
        sta             P_SPI_Status                ; 禁止 SPI 中断,清除中断标志位

        lda             #(C_SPI_SWRST + C_SPISPC_Div_4)
        sta             P_SPI_Ctrl1                 ; 采样时钟 = Fsys/1

        lda             #(C_SPI_EN + C_SPI_SCKPHA + C_SPICS_Div_32 + C_SPI_SPISMPS)
        sta             P_SPI_Ctrl0                 ; Fsys/32, PHA = 1, 主机模式
                ;
L_Main:
L_TestSPI3LInit:
        ldy         #0

L_TestSPI3L2:
        lda         #$00                            ; 片选使能
        sta         P_IOC_Data
        lda         #C_SPI_INTIF                    ; 清除 SPI 中断标志位
        sta         P_SPI_Status
        tya
        sta         P_SPI_TxData                    ; 发送数据
L_TestSPI3L3:
        lda         P_SPI_Status
        and         #C_SPI_INTIF                    ; 发送结束?
        beq         L_TestSPI3L3                    ; 否
        lda         P_SPI_RxData                    ; 读取数据
        sta         G_TXbuf
        sta         P_IOB_Data
        lda         P_IOC_Data                      ; 片选禁止
        eor         #$01
        sta         P_IOC_Data
        cpy         #0                               ; 第一个数据?
        beq         L_TestSPI3L4                     ; 是
        pla
```

```
        cmp     G_TXbuf                 ; 接受的数据 = 上一次发送的数据 ?
        beq     L_TestSPI3L4            ; 是
        lda     #$AA                    ; 错误
        sta     P_IOA_Data
        jmp     L_TestSPI3LInit
L_TestSPI3L4:
        tya
        pha
        jsr     F_Delay100ms
        iny
        cpy     #00                     ; 结束?
        bne     L_TestSPI3L2            ; 否
        pla
        jmp     L_TestSPI3LInit
;

F_Delay100ms:
        lda     #C_WDT_Clr              ; 2c 清除 WDT
        sta     P_WDT_Clr               ; 3c
F_Delay100ms1:
        lda     #$FF                    ; 2c (224ms)
        sta     G_Tempbuf1              ; 3c
        jmp     L_Delay1S_L3

F_Delay1sec:
        lda     #C_WDT_Clr              ; 2c  清除 WDT
        sta     P_WDT_Clr               ; 3c
        lda     #$F9                    ; 2c (1.55s)
        sta     G_Tempbuf1              ; 3c
L_Delay1S_L3:
        ldx     #0
L_Delay1S_L2:
        clc                             ; 2c
        lda     #0                      ; 2c
L_Delay1S_L1:
        Nop                             ; 2c
        nop                             ; 2c
        adc     #1                      ; 2c
        bne     L_Delay1S_L1            ; 3c ((9*256+ 9)*256+ 10)*7+ 5
        inx                             ; 2c
        bne     L_Delay1S_L2            ; 2c
        inc     G_Tempbuf1              ; 5c
```

```
        bne     L_Delay1S_L3                    ; 2c
        rts

F_Delay50us:
        ldx     #200                            ; 2c
L_Delay50_L1:
        nop                                     ; 2c
        nop                                     ; 2c
        inx                                     ; 2c
        bne     L_Delay50_L1                    ; 2c
        rts
;*****************************************
;*                                       *
;*        Interrupt Vector Table         *
;*                                       *
;*****************************************
VECTOR:         .SECTION
            DW      V_NMI                   ; may download program emulated either
            DW      V_Reset                 ; in internal memory or external memory
            DW      V_IRQ                   ; dw define two bytes interrupt vector
;*****************************************
;*                                       *
;*    End Of Interrupt Vector Table *
;*                                       *
;*****************************************
        .END                            ; end of program
```

18.7 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题	**应用例系列号**
I/O 模拟 SPI 通讯	AN_O0310.DOC
SPI 模块使用说明	AN_O0342.DOC

库函数

标题	**应用例系列号**
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

18.8 修订记录

版本号	日期	备注
V0.1	03/31/2005	第一版

19 UART

19.1 简述

UART 即通用异步收发器，可配置成全双工异步通讯方式，与 PC 等通讯；或配置成半双工同步模式与其它周边通讯，如 A/D 或 D/A。SPMC65X 系列芯片都内置了 UART 模块，它的作用是：将外部设备串行数据转换为并行数据接收；将内部并行数据转换为串行数据发送。UART 内部结构见图 19-1。产生 UART 中断的详细步骤见 19.5 节的。UART 模块特点如下：

两个接口管脚

- RXD：数据接收管脚 (与 PC5 复用)
- TXD：数据发送管脚 (与 PC4 复用)

提供标准的异步全双工通讯
可编程的收发波特率
可进行偶校验、奇校验或禁止校验
停止位可设置为 1 位或 2 位
支持发送中断
支持接收中断
高抗噪声能力的数据接收(接收中间连续进行 3 次采样，并对结果进行多数决策)
在接收中进行帧校验和奇偶校验
溢出侦测
CPU 工作频率为 8MHz 时，波特率可在 2400 bps~38400 bps 之间编程设定

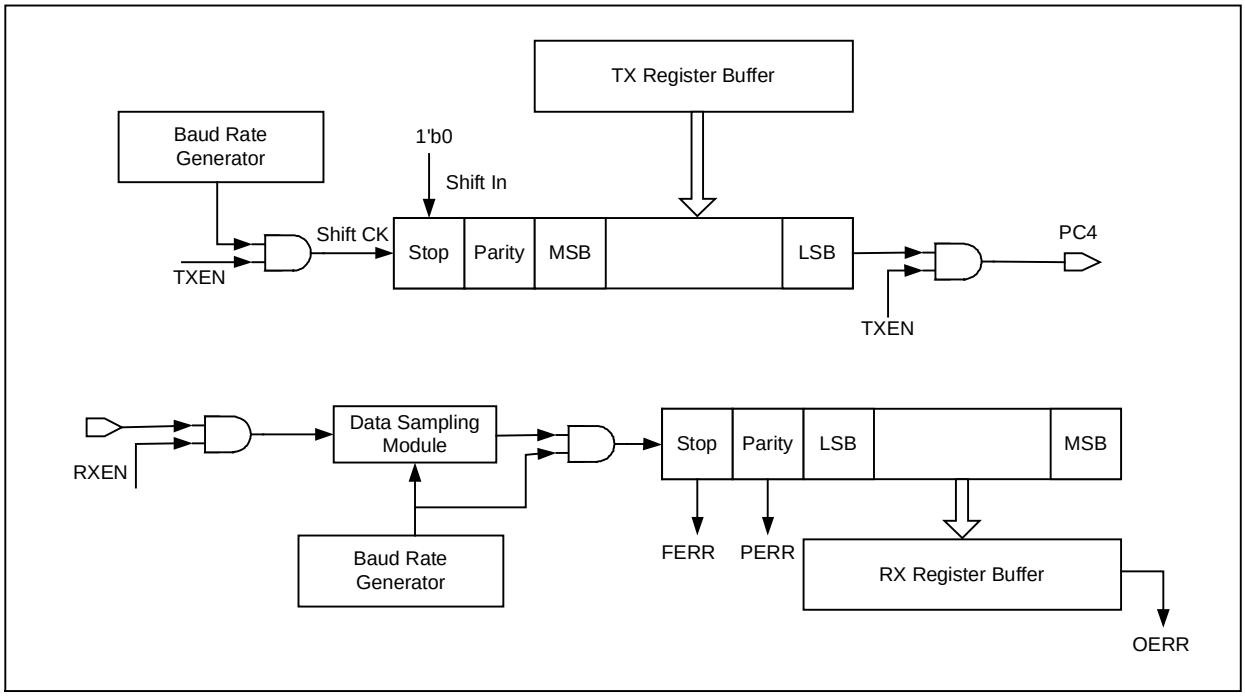


图 19-1 UART 内部结构

UART 数据格式如下图所示：

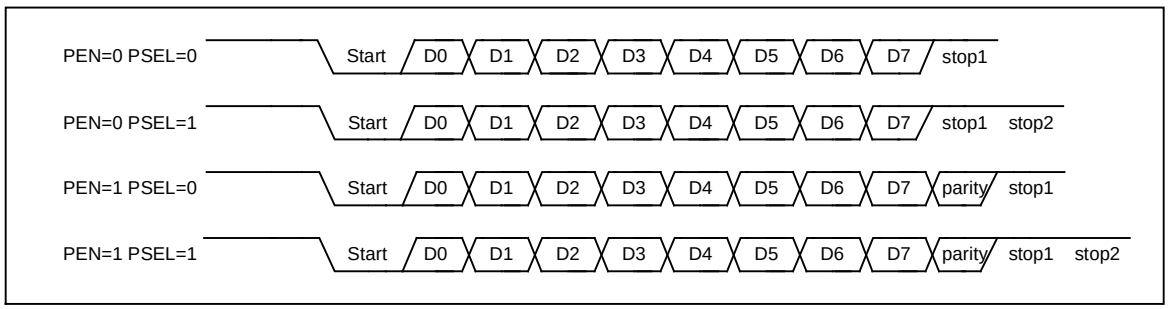


图 19-2 UART 的数据格式

19.2 应用电路

UART 接口通常用于与 PC 机通讯。图 19-3 给出了在 PC 机通过 ICL232 芯片与 SPMC65X 系列芯片通讯的应用电路。

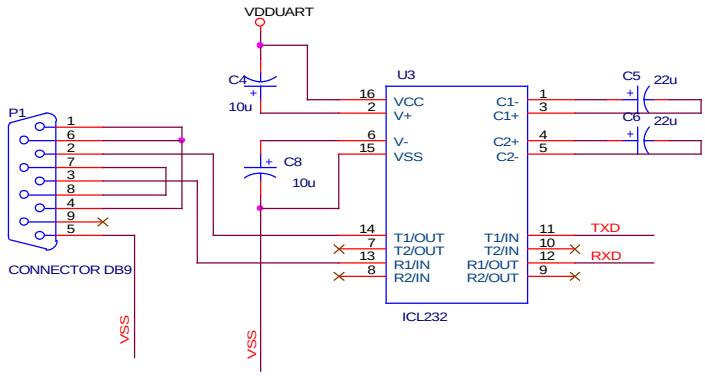


图 19-3 UART 应用电路的接

19.3 控制寄存器

19.3.1 UART 控制寄存器(P_UART_Ctrl, $46) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
RXIE	TXIE	RXEN	TXEN	SOFTRST	STOPSEL	PSEL	PEN
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 **RXIE**: 接收中断使能位
1=使能
0=禁止

Bit 6 **TXIE**: 发送中断使能位
1=使能
0=禁止

Bit 5 **RXEN**: UART 接收功能使能位
1=使能
0=禁止

Bit 4 **TXEN**: UART 发送功能使能位
1=使能
0=禁止
Bit 3 **SOFTRST**: 软件复位
写：
1=复位所有 UART 模块（包括所有的 UART 控制寄存器）
0= 无效
Bit 2 **STOPSEL**:停止位长度选择位
1= 2 位停止位
0= 1 位停止位
Bit 1 **PSEL**: 校验类型选择位
1=偶校验
0=奇校验
Bit 0 **PEN**: 校验使能
1=使能
0=禁止

19.3.2 UART 波特率分频寄存器(P_UART_Baud, $47) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
BaudRate7	BaudRate6	BaudRate5	BaudRate4	BaudRate3	BaudRate2	BaudRate1	BaudRate0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] BaudRate[7:0]: UART 波特率分频器
Baud Rate= F_{SYS} / [16 X (256 – UARTBAUD)]

按下面的公式计算 P_UART_BaudRate 寄存器的值
UARTBAUD=256 - F_{SYS} / (16 X Baud Rate)

波特率 (bps)	建议值@16MHz
2400	#$30
4800	#$98
9600	#$CC
19200	#$E6
38400	#$F3

注意：波特率计算结果不能大于 **38400**。

19.3.3 UART 状态寄存器(P_UART_Status, $48) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
RXIF	TXIF	BUSY	-	-	OERR	PERR	FERR
R	R	R	R	R	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 RXIF: 接收中断标志
1=接收数据准备好(满)
0=接收数据未准备好
Bit 6 TXIF: 发送中断标志
1=发送数据准备好(空)
0=发送数据未准备好
Bit 5 BUSY: UART 正在进行发送标志
1=发送正在进行
0=发送结束，等待下一次发送
Bit [4:3] 保留
Bit 2 OERR:越界错误标志
读：
1=发生越界错误（上一个数据未被读取，新数据覆盖旧数据）
0=未发生界越错误

Bit 1 PERR:奇偶校验错误标志
读：
1=有奇偶校验错误
0=无奇偶校验错误

Bit 0 FERR:帧错误标志
读：
1=发生帧错误（数据信息不全，停止位丢失）
0=无帧错误

19.3.4 UART 数据寄存器(P_UART_Data, $49) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
DATA7	DATA6	DATA5	DATA4	DATA3	DATA2	DATA1	DATA0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] **UARTDATA** [7:0]: UART 发送/接收数据寄存器
读：接收数据，并清除 RXIF 标志
写：将发送数据写入寄存器，并清除 TXIF 标志

19.4 UART 的运行

UART 发送/接收状态从寄存器 P_UART_Status 中读出。当 UART 模块成功的接收到一个数据后，接收数据会被锁存到接收缓冲器中，此时，寄存器 P_UART_Status 中的 RXIF（接收中断标志）便会被置 1，如果寄存器 P_UART_Ctrl 中的 RXIE（接收中断使能位）被置 1，中断便会发生。当读取接收数据后，寄存器 P_UART_Status 中的 RXIF 将会自动的被清除。当发送结束后，寄存器 P_UART_Ctrl 中的 TXIF（发送中断标志）便会被置 1，如果寄存器 P_UART_Ctrl 中的 TXIE（发送中断使能位）被置 1，中断便会发生。当向寄存器 P_UART_Data 中写入数据后，寄存器 P_UART_Status 中的 TXIF 将会自动的被清除。

寄存器 P_UART_Ctrl 中的 BUSY（UART 正在进行发送标志）是个只读标志位，当该位为 1 时，表明 CPU 正在发送数据，发送结束，该标志将会被清除。
寄存器 P_UART_Ctrl 中的 OERR, PERR 和 FERR 为错误指示位，如果下次通讯中没有错误发生，这些错误标志位将会被自动清除。
注意：必须在从 P_UART_Status 读出相应错误标志位之前从 P_UART_Data 读出接收数据。由于状态寄存器 P_UART_Status 只在对控制寄存器 P_UART_Data 进行读取操作时更新，所以此读取顺序不能颠倒。

波特率是由一个波特率寄存器和一个 8 位定时/计数产生的。每次定时/计数器计到最大计数值($FFFF)后，再加 1 时，一个时钟信号就会被发送到波特率产生电路，在该电路中，时钟信号会通过一个 16 分频的计数器，然后产生波特率。定时/计数器溢出后，会自动重新载入波特率寄存器中的值。
波特率= F_{SYS} / [16 X (256 – UARTBAUD)]

波特率寄存器中的内容为 8 位的无符号数。使用下面的公式可从一个已知的波特率导出所需的波特率寄存器值：
UARTBAUD=256 - FSYS / (16 X Baud Rate)

在软件向寄存器 P_UART_Data 写入数据后，UART 开始发送。UART 在 TX 引脚上发送数据按照以下顺序：起始位、8 位数据 (低位在前)、奇偶校验位(奇偶校验使能的情况下有效)、停止位。发送停止位后再经过两个 CPUCLK 周期，寄存器 P_UART_Status 中的 TXIF 位被置 1。

当寄存器 P_UART_Ctrl 中的 RXEN 位使能接收功能后，如果 RX 引脚接收到一个开始位的下降沿，标志着接收数据开始。在任何波特率下 RX 的每一位都要采样 16 次。当检测到开始位的下降沿时，产生波特率的 16 分频计数器会被复位，复位后与接收位边沿对齐，开始接收数据。

为抑制噪声，串口通过对每位接收期间进行 3 次连续采样，并通过多数决议来确定每个接收位的值，此方法尤其适用于起始位。如果 RX 上的下降沿没有通过连续 3 次采样的逻辑低电平多数决议校验，串口则停止接收并等待 RX 管脚上下一个下降沿的产生。

在停止位时间周期内，串口检查以下条件：
如果满足 RXIF = 0 且 RXIE = 1 的条件，寄存器串口向 P_UART_Data 写入接收字节并置 RXIF 位为 1；不满足，则丢失接收数据，不会载入到寄存器 P_UART_Data 中。在停止位时间过了一半之后，串口开始等待 RX 引脚上的下一个下降沿。

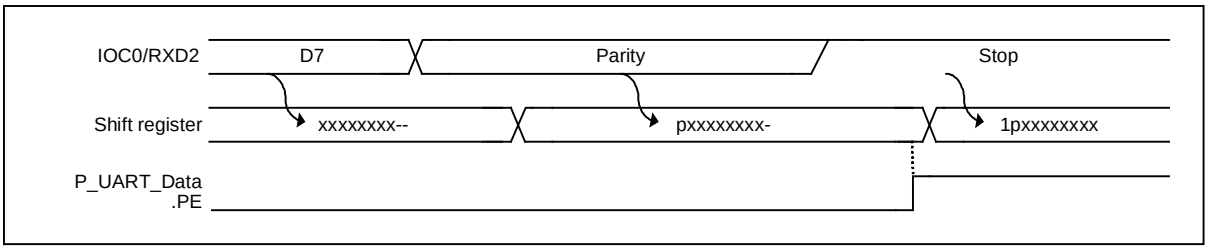


图 19-4 奇偶校验错误时序

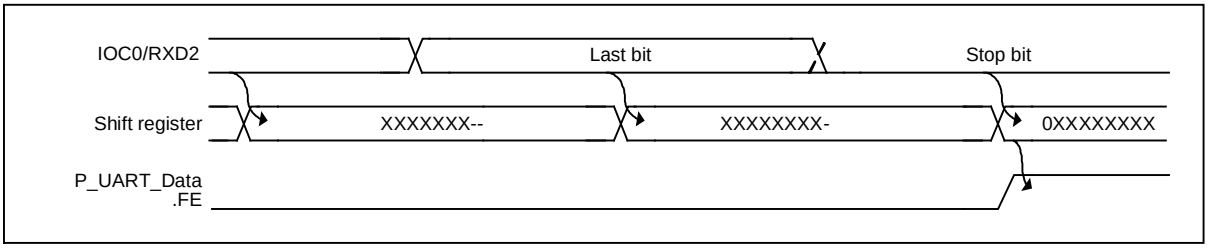


图 19-5 帧错误时序

19.5 UART 中断

与 UART 有关的两个中断：

- 发送中断
- 接收中断

产生 UART 中断需要以下步骤：

- 用“SEI”指令关闭总的中断开关；
- 设置 UART 相关寄存器；
- 在寄存器 P_UART_Ctrl($46h)中使能发送/接收中断；
- 用“CLI”指令打开总的中断开关；
- 等待中断产生。

【例 19 1】UART 初始化，使用中断方式（收/发）

```
    lda     #C_UART_SOFTRST
    sta     P_UART_Ctrl                 ; 对 UART 进行软件复位
    lda     #(C_UART_TXEN + C_UART_TXIE)
    sta     P_UART_Ctrl                 ; UART 发送使能，发送中断 TXINT 使能
    lda     #(C_UART_OERR + C_UART_PERR + C_UART_FERR)
    sta     P_UART_Status               ; 清除错误标志
```

19.6 设计参考

【例 19 2】：以 SPMC65P2408A 为例，使用查询方式发送数据。

```
.SYNTAX   6502                          ; process standard 6502 addressing syntax
.LINKLIST                               ; generate linklist information
.SYMBOLS                                ; generate symbolic debug information

.INCLUDE        Spmc65.inc

        .PAGE0                          ; define values in the range from $00 to $FF
;
        .DATA                           ; define data storage section
```

```
;
        .CODE
;=======================================
;=                                     =
;= Power on Reset Process - Main Program =
;=                                     =
;=======================================
V_Reset:
        sei                             ; 关闭总的中断开关
        ldx     #C_STACK_BOTTOM         ; 将堆栈指针初始化为$00FF 位置上
        txs                             ; 将 X 寄存器的值传送到堆栈指针(SP)中
;
        lda     #$FF                    ; 清除复位标志 清除复位标志
        sta     P_SYS_Ctrl
        sta     P_SYS_Ctrl
;
        jsr     F_InitIOPort            ; 初始化 IO 端口  初始化 IO 口
;
        lda     #$00
        sta     P_IOC_Data
        sta     P_IOC_Attrib

        lda     #11011111B              ; PC5(Rx)设置为输入，PC4(Tx)设置为输出
        sta     P_IOC_Dir
;
        lda     #C_UART_SOFTRST
        sta     P_UART_Ctrl             ; UART 软件复位
        lda     #C_UART_TXEN
        sta     P_UART_Ctrl             ; UART 发送使能

        lda     #(C_UART_OERR + C_UART_PERR + C_UART_FERR)
        sta     P_UART_Status           ; 清除 UART 错误标志
;
        lda     #$CC
        sta     P_UART_Baud             ; P_UART_Baud $CC -> 9600 bps
;
        ldx     #0
;
L_TxReady:
        lda     P_UART_Status
        and     #C_UART_TXIF            ; 检查发送是否准备就绪
        beq     L_TxReady
```

```
;
        lda     Hello, X
        beq     L_exitTx
;
        sta     P_UART_Data
        inx
        lda     #(C_UART_OERR + C_UART_PERR + C_UART_FERR)
        sta     P_UART_Status                   ; 清除 UART 错误标志位
        jmp     L_TxReady                       ; 再次循环
;
L_exitTx:
L_Main:
        nop
        jmp     L_Main
;
Hello:          .BYTE       'Hello World!', 0Ah, 0Dh, 00h

;****************************************
;*                                      *
;*      Interrupt Vector Table          *
;*                                      *
;****************************************
VECTOR:             .SECTION
        DW      V_NMI                   ; may download program emulated either
        DW      V_Reset                 ; in internal memory or external memory
        DW      V_IRQ                   ; dw define two bytes interrupt vector
;****************************************
;*                                      *
;*      End Of Interrupt Vector Table   *
;*                                      *
;****************************************
        .END
```

19.7 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题 **应用例系列号**

I/O 模拟 UART 通讯	AN_O0312.DOC
UART 模块使用说明	AN_O0344.DOC
库函数	
标题	**应用例系列号**
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

19.8 修订记录

版本号	日期	备注
V0.1	03/31/2005	第一版

20 IIC

20.1 简述

IIC(Inter-Integrated Circuit)功能即内部集成电路总线。它支持多主机(multi-master)通讯机制，主机之间相互通信或与其它外围通信时使用两根信号线：一个串行数据线(SDA)和一个串行时钟线(SCL)。为了避免数据混淆、丢失或阻塞，主从机之间通讯必须有特定的协议。在多主机的 IIC 总线中，主机可以向多个从机发送/接收串行数据。数据传送由主机发起，也由主机结束。在一个主机系统中，也可以有多个主机和多个从机。如果多个主机同时企图控制总线，仲裁器会决定那个主机拥有最高优先权。总线上最多可连接的设备数由总线上允许的最大电容值 400 pF 决定，并且还受 IIC 总线最大寻址空间 16 K 限制。

两个外部管脚

SDA：数据输入/输出管脚(与 PC7 复用)

SCL：时钟管脚(与 PC6 复用)

支持主机发送/接收模式

支持从机发送/接收模式

侦测总线仲裁失败

中断发生

可编程的应答信号（**ACK**）

主模式下可编程的时钟频率

20.2 应用电路

IIC 接口通常用于和 EEPROM 进行通讯。如图 20-1 所示为 24C01A 和 SPMC65X 系列 MCU 芯片的应用电路。

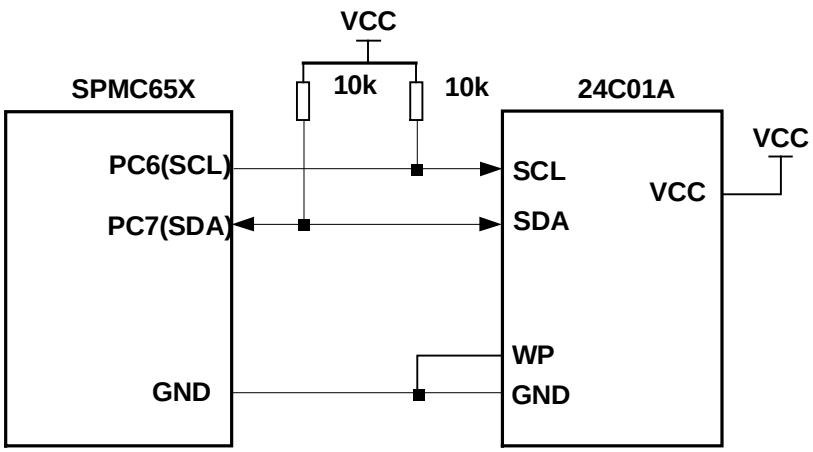


图 20-1 IIC 通信应用电路

20.3 控制寄存器

20.3.1 IIC 总线控制寄存器(P_IIC_Ctrl, $4A) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IICEN	-	TXRXEN	IICIEN	ACKEN	SCK2	SCK1	SCK0
R/W	-	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 **IICEN**: IIC 串行接口使能位
1=使能 IIC 串行接口功能
0= 禁止 IIC 串行接口功能

Bit 6 保留

Bit 5 **TXRXEN**: IIC 总线数据输出使能位
1=使能
0=禁止

Bit 4 **IICIEN**: IIC 中断使能位
1=使能
0=禁止

Bit 3 **ACKEN**: IIC 应答使能位
1=使能应答信号发生
0=禁止应答信号发生

Bit [2:0] **SCK[2:0]**: IIC 时钟频率选择位
111= F_{SYS} ÷ 1024
110= F_{SYS} ÷ 512
101= F_{SYS} ÷ 256
100= F_{SYS} ÷ 128
011= F_{SYS} ÷ 64
010= F_{SYS} ÷ 32
001= F_{SYS} ÷ 16
000= F_{SYS} ÷ 8
FSYS：系统时钟频率

20.3.2 IIC 总线状态寄存器(P_IIC_Status, $4B) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
MXT	TXRXSEL	BUSY/SIGEN	IICIF	ARBITRAT	AAS	AZERO	LRB
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit 7 **MXT**: IIC 主/从模式选择位
1= 主模式
0= 从模式

Bit 6 **TXRXSEL**:发送/接收选择位
1= 发送模式
0= 接收模式

Bit 5 **BUSY/SIGEN**:总线正忙或启动/停止信号发生位

读：

1=总线忙

0=总线就绪

写：

1=启动信号

0=停止信号

注意：

向 MXT 和 **TXRXSEL** 写数据时，保持该位为 0

当 IICF =1，清除 INTIF 标志时，芯片自动忽略该位的任何设置，

Bit 4 **IICIF**: IIC 中断标志

1= IIC 中断发生

0= IIC 中断未发生

Bit 3 **ARBITRAT**: IIC 仲裁器状态标志位

1=总线仲裁器在通讯中仲裁失败

0=总线仲裁器良好

Bit 2 **AAS**: IIC 从机寻址状态位

1=接收到的地址和 IIC 地址寄存器里的值相同

0=启动信号/停止信号已经发生

Bit 1 **AZERO**: IIC 零地址状态位

1=接收从地址为"$00"

0=启动/停止动作已经发生

Bit 0 **LRB**: IIC 最后接收位状态标志

1=最后接收位为 1(应答信号 ACK 未收到)

0=最后接收位为 0(应答信号 ACK 已收到)

20.3.3 IIC 总线数据寄存器(P_IIC_Data, $4C) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
DATA7	DATA6	DATA5	DATA4	DATA3	DATA2	DATA1	DATA0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] **IICDATA** [7:0]: IIC 总线数据寄存器

读： 接收数据

写： 发送数据

20.3.4 IIC 总线地址寄存器(P_IIC_Address, $4D) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADDR7	ADDR6	ADDR 5	ADDR 4	ADDR 3	ADDR 2	ADDR 1	ADDR 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] **IICADDRESS** [7:0]: IIC 总线地址寄存器

从机模式：从机的地址

20.4 操作

IIC 总线协议对其连接的每个通讯设备都定义了一个地址，当主机启动数据传送时，首先要将它欲与之通讯的那个从机的地址发到总线上，此时所有的从机都监听这个地址信息。在这个地址中，有一位用来定义主机是从从机中读数据还是向从机写数据的。主机和从机在进行数据传送的时候的状态永远是互补的(发送/接收)。他们的关系必为以下二者之一：

主机发送，从机接收

从机发送，主机接收

在这两种情况下，时钟信号都是由主机产生的。

为了在总线上与其它设备“线与”(wired-AND)连接，时钟线(SCL)和数据线(SDA)是开漏输出。外部的上拉电阻用于保证当没有设备拉低总线时，总线保持高电平状态。

如下的协议通常用于 24C01 中：

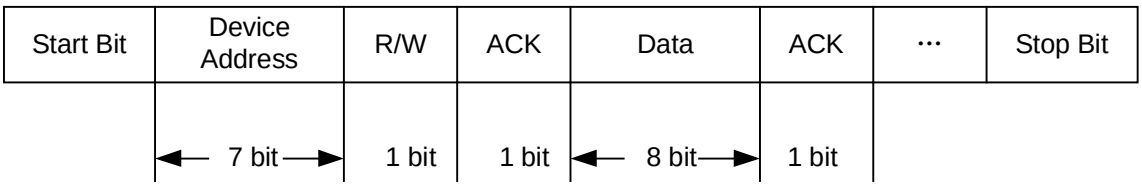


图 20-2 24C01 协议

20.4.1 启动和停止数据传送

当总线上无数据传送时(空闲时间)，时钟线(SCL)和数据线(SDA)通过外部上拉电阻被拉高。

启动信号和停止信号决定数据传送的开始和停止，定义如下：

启动信号：SCL 为高电平时，SDA 由高电平向低电平跳变，开始传送数据；

停止信号：SCL 为高电平时，SDA 由低电平向高电平跳变，结束传送数据。

如图 16-3 所示为开始信号和停止信号的具体定义。主机产生这两个信号作为数据传送的开始和结束。根据启动信号和停止信号的定义，当数据正在传送时，SDA 只能在 SCL 拉低后才能改变状态，传送新的数据位。

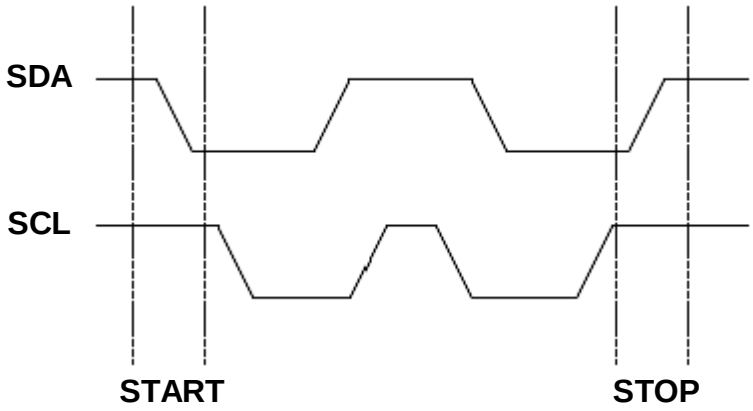


图 20-3 开始和停止信号

20.4.2 地址格式的定义

SPMC65x 芯片仅支持 7 位地址格式外加 1 位读/写标志位。7 位地址格式如图 20-4 所示。

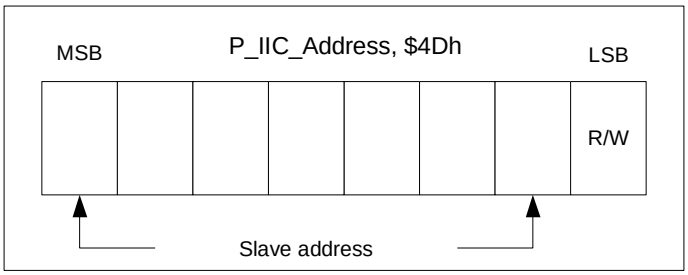


图 20-4 7 位地址格式

20.4.3 传送应答

所有数据都是一个字节一个字节的发送，每次发送的字节数不做限定。从机每收到一个字节后，都会产生一个应答信号(ACK)。

如果主机没有收到应答信号，则必须停止传送。此时从机必须释放 SDA，使其恢复为高电平，让主机可以产生停止信号。如图 20-5 所示为 IIC 总线上的应答信号。

如果主机正在接收数据(主机接收模式)，除了最后接收的那个字节外，每接收到一个字节都会产生一个应答信号。最后一个字节不产生应答信号，其实也是一种信息，就是为了告诉从机发送结束。然后从机释放 SDA，让主机产生停止信号。主机也可以在应答信号的脉冲期间，与时钟信号 SCL 组合，产生停止信号，从而结束数据传送。

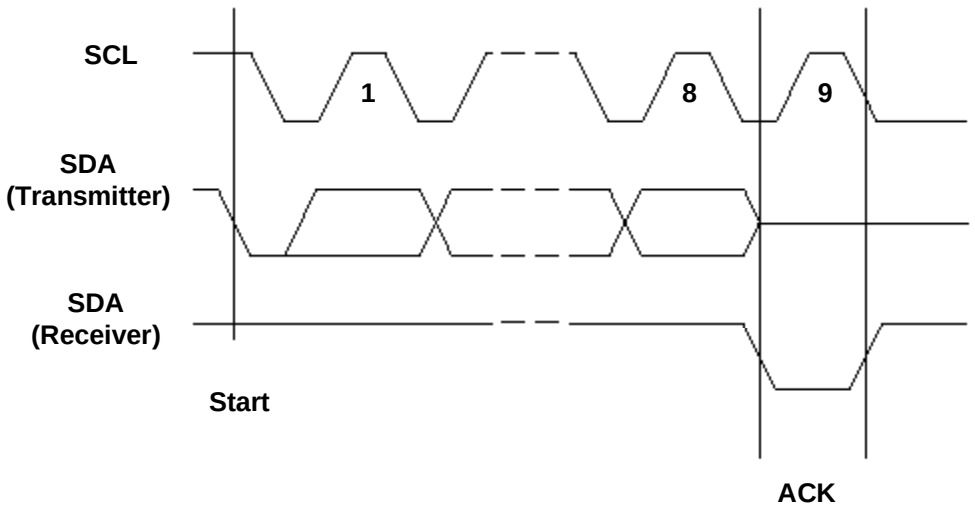


图 20-5 IIC 总线应答信号

20.4.4 通用地址

IIC 总线上寻址的过程是这样的：在启动信号后传送的第一个字节即主机要访问的从机的地址信息。不同的是：当主机发送的第一个字节为通用地址时，则可以访问所有的从机。理论

上讲，发送通用地址后，所有的从机都应该向主机发送一个应答信号。

IIC 协议规定了一个通用地址，它的 7 个地址位和一个 R/W 位均为 0，用于呼叫所有的从机。通用地址可以被自动识别。此时，将 AZERO 位置 1。

20.4.5 总线仲裁

多主机模式的系统是通过总线仲裁来协调工作的。当主机 A 输出高电平到 SDA 管脚上时，同时，另一主机 B 输出低电平到 SDA 管脚，将 SDA 线拉低，这种情况下，主机 A 从 SDA 管脚读回的信号为低，与输出不一致，此时，ARBITRAT 位置 1，表明总线被另一主机 B 占用。

20.4.6 从机模式

在从机模式下，从机的 SCL (PC6)和 SDA (PC7)管脚必须设置为输入。当需要作为 IIC 输出(slave–transmitter)时,IIC 模块会主动将 IO 口设置为输出。
当地址值与寄存器 P_IIC_Address 里的值相等或者 IIC 模块自动将 ACKEN 位置 1，产生应答信号，然后将接收到的数据存入 P_IIC_Data 寄存器。
从机的设置步骤如下：

- 将 PC6 和 PC7 置为悬浮输入
- 使能 IIC 模块(IICEN= 1)
- 使能 IIC 数据输出(TXRXEN= 1)
- 设置从机模式(MXT= 0, TXRXSEL= 0)

典型的接收数据步骤如下：

- 在 P_IIC_Data 寄存器中设置从机地址
- 通过查询 IICIF 位，得知一个字节的数据接收完毕
- 当主机在写模式下(TXRXSEL=0)，从寄存器 P_IIC_Data 中获取主机要写入的数据。
- 当主机在读模式下(TXRXSEL=1)，从机将要发送的数据载入寄存器 P_IIC_Data 中。
- 清除 IICIF 位

20.4.7 主机模式

在主机模式下，SCL (PC6)和 SDA (PC7)管脚必须配置为输入。当需要作为 IIC 输出(master–transmitter)时,IIC 模块会主动将 IO 口设置为输出。
IIC 的时钟(SCL)由主机产生，SPMC65x 系列芯片支持 8 种时钟频率(通过 SCK[2:0]选择)，最高达 1MHz@ 8MHz。设置主机模式的步骤如下：

- 将 PC6 和 PC7 置为悬浮输入
- 使能 IIC 模块(IICEN= 1)
- 使能 IIC 数据输出(TXRXEN= 1)
- 设置主机模式(MXT= 1, TXRXSEL= 1)

典型的数据发射步骤如下：

- 在寄存器 P_IIC_Data 中设置从机地址
- 通过查询 BUSY 位，得知总线是否就绪可用
- 设置 SIGGEN 位，产生启动信号，开始发送寄存器 P_IIC_Data 中的数据。

- 通过查询 IICIF 位，得知数据已发送完成
- 将下一个要发送的数据载入 P_IIC_Data
- 清除 IICIF 标志位
- 设置 SIGGEN 位，产生停止信号

【例 20 1】:设置 IIC 主机模式

```
    lda     #(C_IIC_IICEN+C_IIC_INTEN+C_IIC_TXRXEN+C_IIC_ACKEN+ C_IIC_SCK_128)
    sta     P_IIC_Ctrl                  ; IIC 使能，应答和数据输出使能
                                        ; IIC   clock = Fsys / 128
    lda     #(C_IIC_MXT + C_IIC_TXRXSEL + C_IIC_INTIF)
    sta     P_IIC_Status                ; 主机发送模式，清除 IICIF 标志
```

20.5 IIC 中断

以下事件导致发生 IIC 中断，并置中断标志位 IICIF。

- 发送一个字节的数据
- 接收一个字节的数据

设置 IIC 中断的步骤如下：

- 用“SEI ”指令关闭总的中断开关
- 设置与 IIC 操作相关的寄存器
- 在 P_IIC_Ctrl($4Ah)寄存器中将 IIC 中断使能位置 1
- 用“CLI”指令打开总的中断开关
- 等待中断产生

20.6 设计参考

【例 20 3】 :以 24C02 为例，进行 IIC 通信，范例如下：

- 查询方式发送数据
- 发送($AA,$A5,$5A)

```
.SYNTAX   6502                                  ; process standard 6502 addressing syntax
.LINKLIST                                       ; generate linklist information
.SYMBOLS                                        ; generate symbolic debug information
.INCLUDE        Spmc65.inc
    .PAGE0                                      ; define values in the range from $00 to $FF
G_Tempbuf1      DS      1               ;
G_Tempbuf2      DS      1
G_TXbuf         DS      7               ;
;
        .DATA                                   ; define data storage section
;
        .CODE
;=======================================
```

```
;=                                                =
;= Power on Reset Process - Main Program          =
;=                                                =
;==================================================
V_Reset:
            sei                                   ; 关闭总的中断开关
            ldx         #C_STACK_BOTTOM           ; 将堆栈指针初始化为$00FF 位置上
            txs                                   ; 将 X 寄存器的值传送到堆栈指针(SP)中
;
            lda         #$FF                      ; 清除复位标志清除复位标志
            sta         P_SYS_Ctrl
            sta         P_SYS_Ctrl
;
          ; IIC   主机模式设置
            lda         #00000000B
            sta         P_IOC_Data
            lda         #11000000B
            sta         P_IOC_Attrib
            lda         #00000000B                ; PC6(SDL), PC7(SDA)设置为悬浮式输入
            sta         P_IOC_Dir

            lda         #(C_IIC_IICEN + C_IIC_TXRXEN + C_IIC_ACKEN + C_IIC_SCK_128)
            sta         P_IIC_Ctrl                ; IIC  使能，数据输出和应答信号使能
                                                  ; IIC    clock = Fsys / 128

            lda         #(C_IIC_MXT + C_IIC_TXRXSEL + C_IIC_INTIF)
            sta         P_IIC_Status              ; 主机发送模式，清除 IICIF 标志
;
;
L_Main:
L_I2C_Func:
            lda         #%10100000
            sta         P_IIC_Data                ; 发送从机地址

L_TestI2CL20:
            lda         P_IIC_Status
            and         #C_IIC_BUSY               ; 总线繁忙?
            bne         L_TestI2CL20              ; 是
;
            lda         P_IIC_Status
            ora         #%00100000
            sta         P_IIC_Status              ; 产生'START'信号
```

```
        ldx     #0

L_TestI2CL21:
        lda     P_IIC_Status
        and     #C_IIC_INTIF                    ; INTIF 置 1?
        beq     L_TestI2CL21                    ; 否
;
L_TestI2CL22:
        lda     T_I2C_Data, X                   ; 发送数据
        sta     P_IIC_Data

        lda     P_IIC_Status
        ora     #C_IIC_INTIF                    ; 清除 INTIF(发送)
        sta     P_IIC_Status
        inx
        cpx     #4                              ; 4 bytes 发送结束?
        bne     L_TestI2CL21                    ; 否
;
L_TestI2CL3:
        lda     P_IIC_Status
        and     #C_IIC_INTIF                    ; INTIF 置 1?
        beq     L_TestI2CL3                     ; 否

L_TestI2CL31:
        lda     #%11000000                      ; 产生'STOP'信号
        sta     P_IIC_Status
        jsr     F_Delay100ms                    ; 等待 E2RPOM 写周期(10ms)


        lda     #(C_IIC_IICEN + C_IIC_TXRXEN + C_IIC_ACKEN + C_IIC_SCK_128)
        sta     P_IIC_Ctrl

        lda     #(C_IIC_MXT + C_IIC_TXRXSEL + C_IIC_INTIF)
        sta     P_IIC_Status

        lda       #$A0                          ; 从机地址
        sta     P_IIC_Data

        lda     #%11100000                      ; 写起始信号
        sta     P_IIC_Status
        nop
```

```
L_TestI2CL4:
        lda     P_IIC_Status
        and     #C_IIC_INTIF                    ; INTIF 置 1?
        beq     L_TestI2CL4                     ; 否
;
        lda     T_I2C_Data                      ; IIC 地址
        sta     P_IIC_Data
        lda     P_IIC_Status
        ora     #C_IIC_INTIF                    ; 清除 INTIF 标志(发送)
        sta     P_IIC_Status
L_TestI2CL41:
        lda     P_IIC_Status
        and     #C_IIC_INTIF                    ; INTIF 置 1?
        beq     L_TestI2CL41                    ; 否


        lda     #%10100000                      ; IIC 从机地址
        sta     P_IIC_Data

        lda     #%10100000                      ; 读开始信号
        sta     P_IIC_Status

L_TestI2CL42:
        lda     P_IIC_Status
        and     #C_IIC_INTIF                    ; INTIF 置 1?
        beq     L_TestI2CL42                    ; 否
;
        lda     P_IIC_Status
        ora     #C_IIC_INTIF                    ; 清除 INTIF 标志(发送)
        sta     P_IIC_Status

        ldx     #0
L_TestI2CL43:
        lda     P_IIC_Status
        and     #C_IIC_INTIF                    ; INTIF 置 1?
        beq     L_TestI2CL43                    ; 否
;
        lda     P_IIC_Data
        sta     G_TXbuf,X
;
        inx
```

```
            cpx     #3                                  ; 接收结束?
            beq     L_TestI2CL45                        ; 是
            cpx     #2                                  ; 接收最后一个数据?
            bne     L_TestI2CL44                        ; 否
            lda     #(C_IIC_IICEN + C_IIC_TXRXEN)
            sta     P_IIC_Ctrl                          ; I2C 使能,无应答 ACK,输出使能
;
L_TestI2CL44:
            lda     P_IIC_Status
            ora     #C_IIC_INTIF                        ; 清除 INTIF 标志(发送)
            sta     P_IIC_Status
            jmp     L_TestI2CL43

L_TestI2CL45:
            lda     #%10000000                          ; 停止信号
            sta     P_IIC_Status
;
            jmp     L_I2C_Func
;
T_I2C_Data:
            DB      #$52                                ; Word address
            DB      #$AA                                ; data 1
            DB      #$A5                                ; data 2
            DB      #$5A                                ; data 3
;****************************************
;*                                      *
;*          Interrupt Vector Table      *
;*                                      *
;****************************************
VECTOR:                 .SECTION
                DW          V_NMI                   ; may download program emulated either
                DW          V_Reset                 ; in internal memory or external memory
                DW          V_IRQ                   ; dw define two bytes interrupt vector
;****************************************
;*                                      *
;*          End Of Interrupt Vector Table *
;*                                      *
;****************************************
            .END                                    ; end of program
;
```

20.7 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题	应用例系列号
I/O 模拟 IIC 通讯	AN_O0311.DOC
IIC 模块使用说明	AN_O0343.DOC

库函数

标题	应用例系列号
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

20.8 修订记录

版本号	日期	备注
V0.1	03/31/2005	第一版

21 省电模式

21.1 简述

在 SPMC65X 系列中，芯片有三种工作模式：正常(NORMAL)模式、STOP 模式和 HALT 模式。如图 21-1 是这三种模式的关系图，芯片上电后直接进入 NORMAL 模式。此时，用户可以将 NORMAL 模式转换为 STOP 模式或 HALT 模式，但进入 STOP 模式或 HALT 模式后，只能返回到 NORMAL 模式。也就是说，用户无法直接在 HALT 模式与 STOP 模式之间进行转换，必须先转换成 NORMAL 模式后再进行转换，反之亦然。

NOMAL 模式下功耗最大，所有的外围均可用。选择这两种省电模式，可以极大的降低整个芯片的功耗。它们的唤醒时间不同，用户可以根据应用需要选择适当的运行模式。从 NORMAL 模式转换成 STOP 或 HALT 模式时，需要对模式控制控制寄存器连续写两次才能有效。关于省电模式的详细描述见接下来的章节。

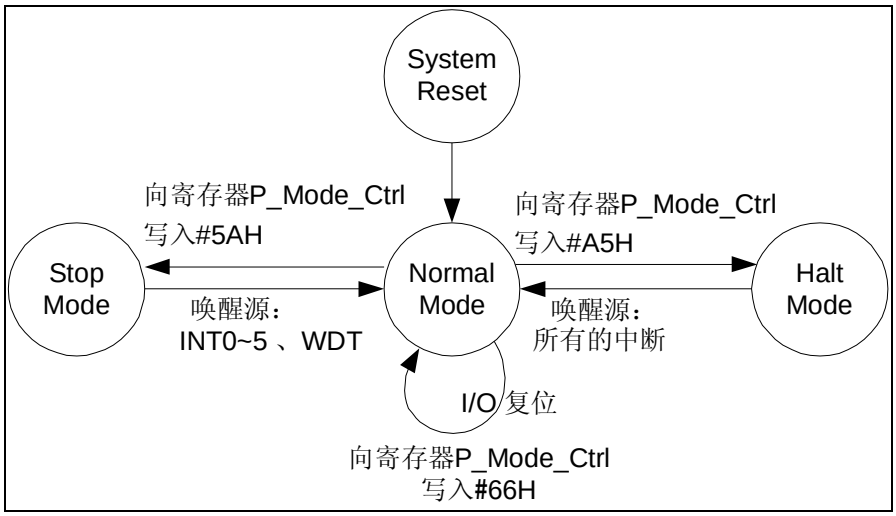


图 21-1 三种工作模式的关系图

表 21 1 三种模式下 CPU 和外围硬件的运行情况

运行模式	控制寄存器 ($31)	CPU	外围
Stop	#$5A	OFF	OFF
Halt	#$A5	OFF	ON
High Speed (Normal)	-	ON	ON

当系统进入 STOP 模式下，系统时钟将会停止，CPU 所有的工作也随之停止，只有来自 IO 管脚的外部中断和看门狗中断才能唤醒系统重新进入 NORMAL 模式。

采用中断将系统唤醒后，时钟的稳定需要一定的时间，一般这个时间值大约为 40ms，范围在 20ms~ 60ms 之间。时钟稳定后，如果总中断开关打开(用 CLI 命令)，程序就会立即跳转

到中断服务子程序中执行；若总中断开关关闭，则系统唤醒后不产生中断，不会进入中断子程序，接着原程序（进入 STOP 模式的命令程序之后）继续执行。为确保外部中断可以唤醒系统，在进入 STOP 模式之前必须打开相应的中断使能位。

若使用看门狗进行唤醒，则必须在进入 STOP 模式之前先打开看门狗定时器。但是，使能看门狗定时器后，芯片的功耗会有所增加，因此如果不需要用看门狗定时中断作为唤醒源，则从降低功耗的角度上考虑，应该关闭 RC 振荡器，方法是将寄存器 P_WDT_Ctrl 中的 SCKEN 清零。详细做法见本章的设计参考。另外，在使用看门狗进行唤醒的时候，。必须设置恰当的看门狗中断频率，以保证系统时钟有足够的稳定时间，防止发生系统复位。在 HALT 模式下，系统时钟停止，CPU 的电源被关闭，但是其外围正常工作，所以 HALT 模式下功耗大于 STOP 模式下的功耗。HALT 模式下，所有的中断源都可以作为唤醒源。

HALT 模式下，所有的中断源都可以作为唤醒源，使系统回到 NORMAL 模式下。如果总中断开关打开(用 CLI 命令)，在系统被中断唤醒后，程序就会立即跳转到中断服务子程序中执行；若总中断开关关闭，则系统唤醒后不产生中断，不会进入中断子程序，接着原程序（进入 HALT 模式的命令程序之后）继续执行。从 HALT 模式回到 NORMAL 模式无时间延迟。

21.2 控制寄存器

21.2.1 省电模式控制寄存器 (P_MODE_Ctrl, $31) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Mode_Ctrl7	Mode_Ctrl6	Mode_Ctrl5	Mode_Ctrl4	Mode_Ctrl3	Mode_Ctrl2	Mode_Ctrl1	Mode_Ctrl0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit [7:0] **Mode_Ctrl**[7:0]:省电模式选择位

读：

数据总为$00

写(两次)：

$5A=进入 Stop 模式 (C_MODE_STOP = $5A)

$A5=进入 HALT 模式(C_MODE_HALT = $A5)

$66=复位除 CPU 的所有内部模块(C_MODE_Reset = $66)

注意：该字节必须连续写两次才能设置生效。

21.2.2 看门狗控制寄存器(P_WDT_Ctrl, $32) (R/W)

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SCKEN	WDCS2	WDCS1	WDCS0	-	-	-	-
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	1	-	-	-	-

Bit 7 SCKEN: STOP 模式下看门狗内置 RC 振荡器使能位

1= 使能

0= 禁止

Bit [6:4] WDCS: 看门狗中断频率选择位

000 = f_slow/128

001 = f_slow/256

010 = f_slow/512

011 = f_slow/1024

100 = f_slow/2048

101 = f_slow/4096

110 = f_slow/8192

111 = f_slow/16384

f_slow:内置 RC 振荡器，典型频率为 25KHz

Bit [3:0] 保留

注意：该字节必须连续写两次才能设置生效。

21.3 操作

21.3.1 STOP 模式

进入 STOP 模式后，片上的振荡器停止，但是 RAM 区的数据包括控制寄存器的设置仍然保持不变。IO 端口管脚状态也保持与进入 STOP 模式之前的状态相同。

如图 21-2 所示为进入 STOP 模式的时序图。通过向寄存器 P_MODE_Ctrl 写入“STOP”命令(写入“C_MODE_STOP”，其值为 “$5a”)，系统进入 STOP 模式。在执行进入 STOP 模式命令（向寄存器 P_Mode_Ctrl 写入“C_MODE_STOP”）之前，用户必须清除所有的中断标志位，并使能唤醒中断源。否则，若在中断标志位为 1 的情况下执行进入 STOP 模式命令，程序不会进入 STOP 模式。因此，必须清除中断标志位(P_INT_Flag0、P_INT_Flag1、P_INT_Flag2)。另外，需要在进入 STOP 模式的指令之后插入至少两条 NOP 指令，保证程序的正确性。注意，如果用户不需要使能看门狗定时器，则应当将寄存器 P_WDT_Ctrl 中的 WDTEN 清零，停止片上振荡电路，以降低功耗。

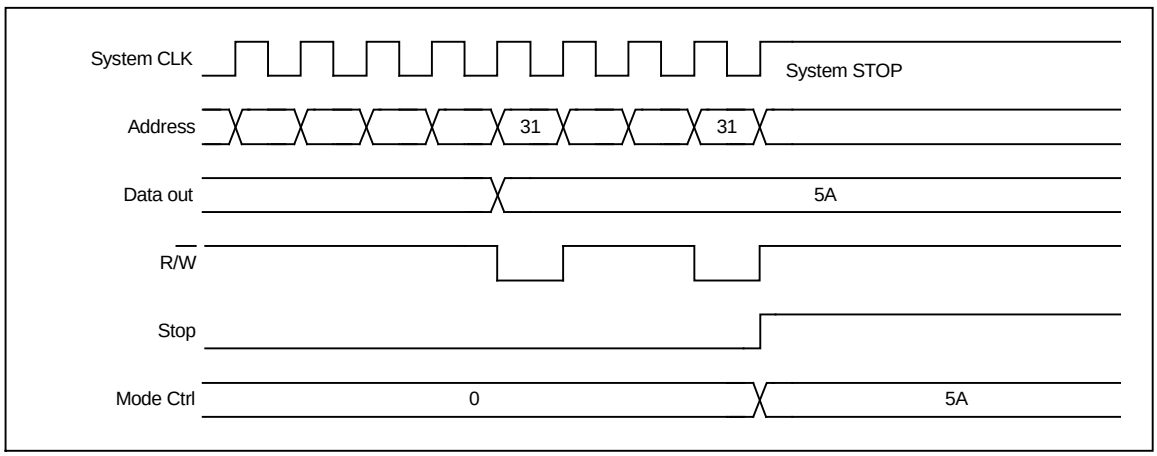


图 21-2 进入 STOP 模式的时序

【例 21 1】：设置芯片的 STOP 模式

```
lda     #$FF
sta     P_INT_Flag0         ; 清除中断标志位
sta     P_INT_Flag1
lda     # C_INT_IRQ0E       ; 使能 IRQ0 作为唤醒源
sta     P_INT_Ctrl0
cli                         ; 打开总的中断开关
lda     #C_MODE_STOP        ; STOP 命令
sta     P_MODE_Ctrl
sta     P_MODE_Ctrl
nop                         ; 等待 2 个 NOP 指令周期
nop
```

21.3.2 唤醒 STOP 模式

来自 IO 口的外部中断、看门狗中断可以将系统从 STOP 模式中唤醒系统重新进入 NORMAL 模式。使用硬件复位也可以使系统重新进入 NORMAL 模式，但是硬件复位会重新定义控制寄存器的值，并不改变用户 RAM 的值；通过外部中断唤醒后，控制寄存器和用户 RAM 的值都不会改变。为了使设定的中断能够将系统唤醒，需要在进入 STOP 模式之前，必须使能唤醒中断源。

当使用中断将 CPU 唤醒后，程序会产生分支：如果状态寄存器 P 中的中断屏蔽位（I）为 1，主程序会继续执行下去，不会产生跳转。但是如果（I）为 0，主程序会暂停，跳转到中断服务子程序去执行。具体流程如图 21-3 所示。

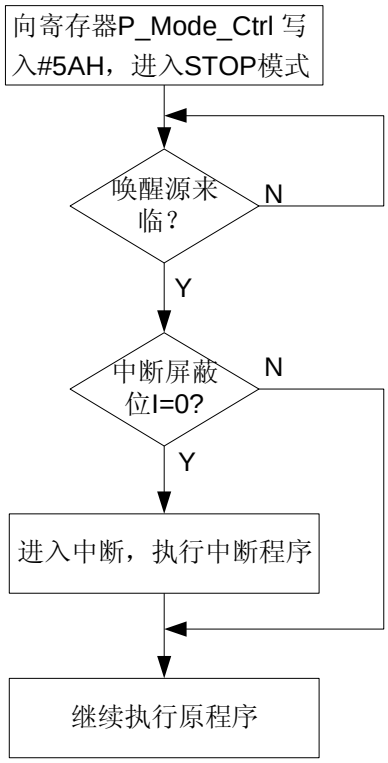


图 21-3 STOP 模式的运行流程

当 STOP 模式被外部中断唤醒时，系统振荡器需要一点时间来重新稳定下来，这个时间值在 20ms~ 60ms 之间，一般为 40ms。如图 21-4 所示为 STOP 模式释放时序。

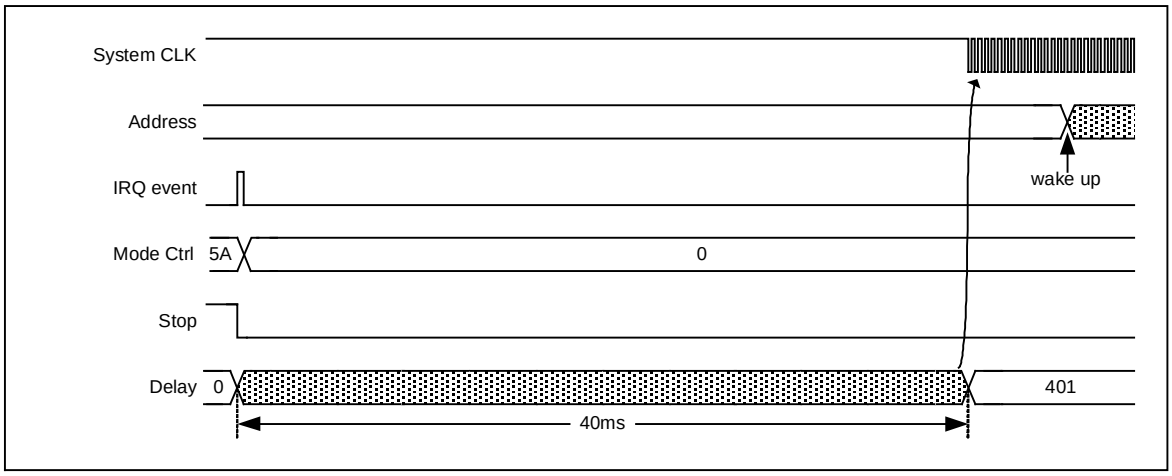


图 21-4 STOP 模式释放时序

如果用户采用看门狗定时中断作为系统的唤醒源，那么必须在系统进入 STOP 模式之前，使能 RC 振荡时钟。另外，在使用看门狗进行唤醒的时候，必须设置恰当的看门狗中断频率，以保证系统时钟有足够的稳定时间，防止发生系统复位。见图 21-5。

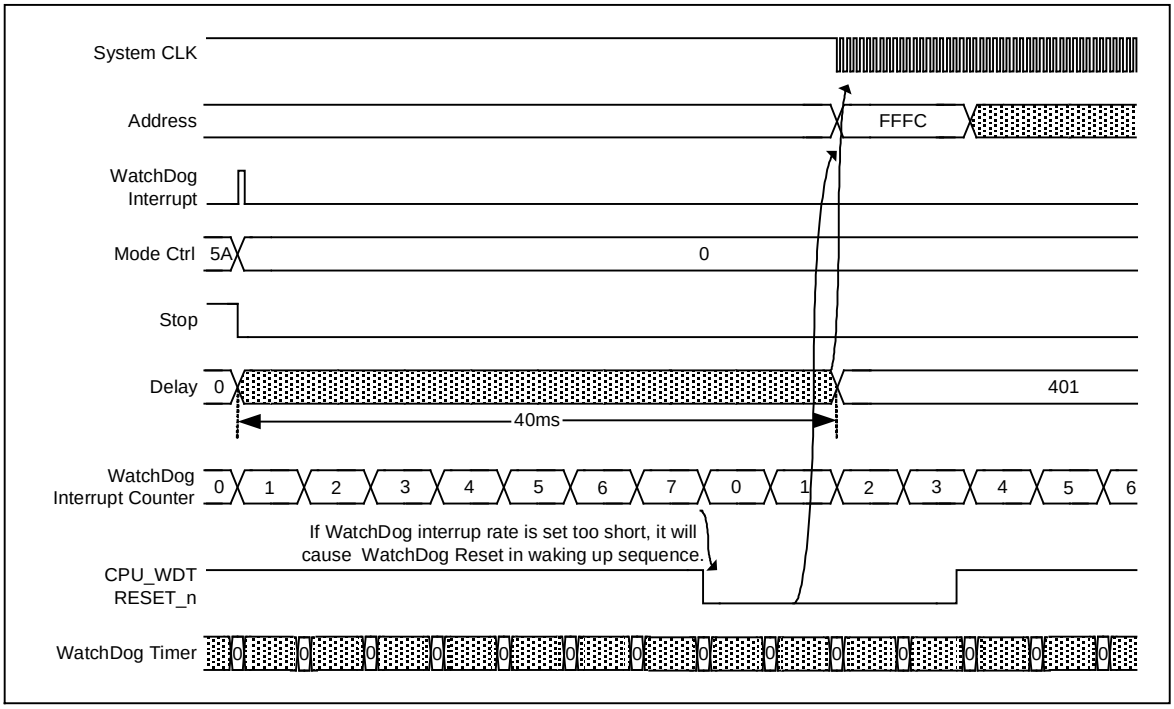


图 21-5 看门狗时钟频率设置不当导致在唤醒期间发生系统复位。

21.3.3 Halt 模式

如图 21-6 所示，为进入 HALT 模式的时序。在 HALT 模式下，所有的外围保持原来状态，正常工作。通过向寄存器 P_MODE_Ctrl 写入“HALT”命令(写入“C_MODE_HALT”，其值为“$a5”)进入 HALT 模式。在执行进入“HALT”模式命令（向寄存器 P_MODE_Ctrl 写入“C_MODE_HALT”）之前，用户必须清除所有的中断标志位，并使能唤醒中断源。否则，若在中断标志位为 1 的情况下执行进入 HALT 模式命令，程序不会进入 HALT 模式。因此，必须清除中断标志位(P_INT_Flag0、P_INT_Flag1、P_INT_Flag2)。另外，需要在进入 HALT 模式的指令之后插入至少两条 NOP 指令，保证程序的正确性。

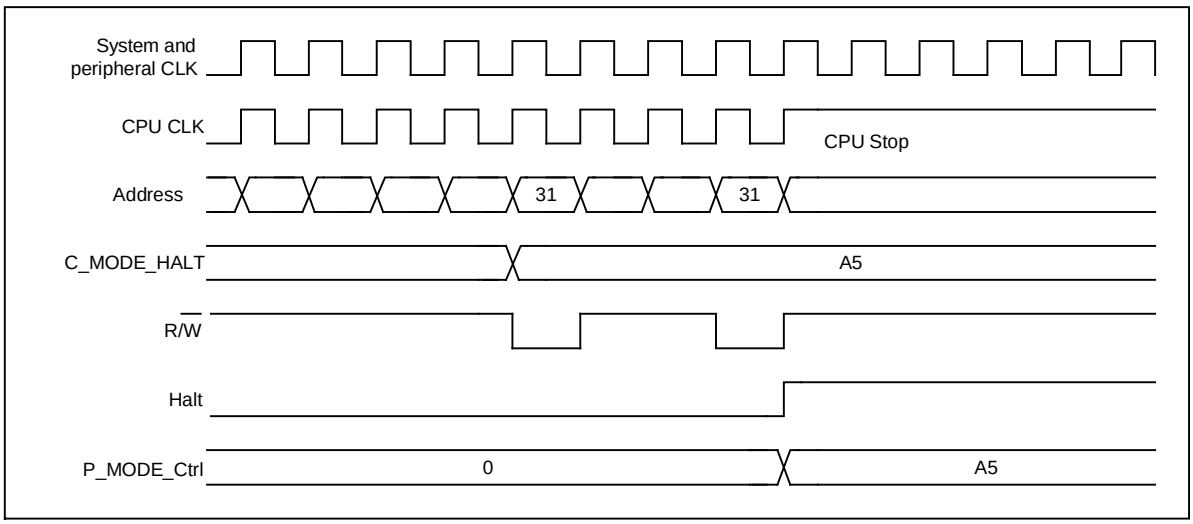


图 21-6 进入 HALT 模式

关于如何设置进入 HALT 模式，请参考下面的范例。

【例 21 2】：令 MCU 进入 HALT 模式

```
lda     #$FF                ; 清除中断标志位
sta     P_INT_Flag0
sta     P_INT_Flag1
lda     #C_INT_IRQ0E        ; 使能 IRQ0 中断
sta     P_INT_Ctrl0
cli                         ; 打开总的中断开关;
lda     #C_MODE_HALT
sta     P_MODE_Ctrl
sta     P_MODE_Ctrl         ; 进入 HALT 模式
nop
nop                         ; 等待 2 个 NOP 指令周期
```

21.3.4 唤醒 Halt 模式

所有的中断都可以作为 HALT 模式的唤醒源，例如：外部中断、定时/计数器溢出中断、看门狗中断、ADC 中断、捕获中断、SPI 中断和 UART 通讯中断等。或者通过硬件复位唤醒。硬件复位唤醒会重新定义控制寄存器的值但并不改变 RAM 的值；通过外部中断则二者的值都不会改变。为了使设定的中断能够将系统唤醒，在进入 HALT 模式之前，必须使能唤醒中断源。

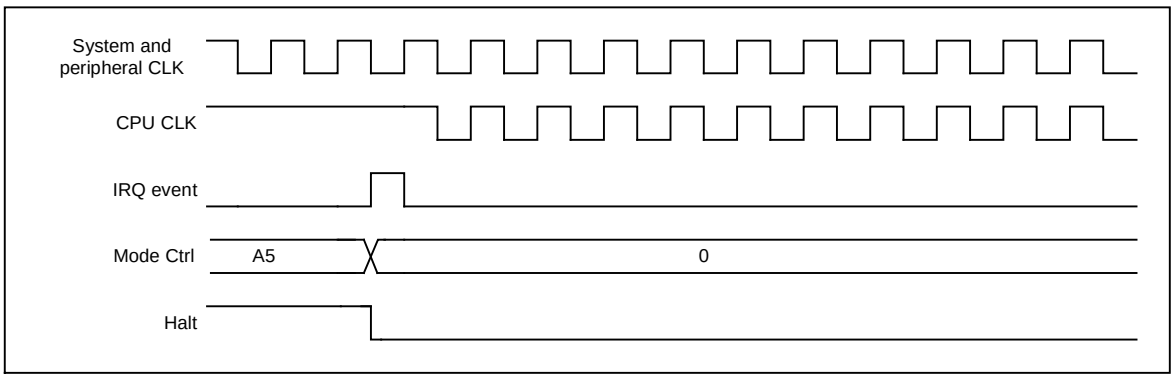


图 21-7 HALT 模式唤醒时序

HALT 模式唤醒时序见图 21-7。当使用中断将 CPU 唤醒后，程序会产生分支：如果状态寄存器 P 中的中断屏蔽位（I）为 1，主程序会继续执行下去，不会产生跳转。但是如果（I）为 0，主程序会暂停，跳转到中断服务子程序去执行。具体流程如图 21-8 所示。当 HALT 模式被外部中断唤醒时，需要一点时间来重新建立和稳定系统振荡器。

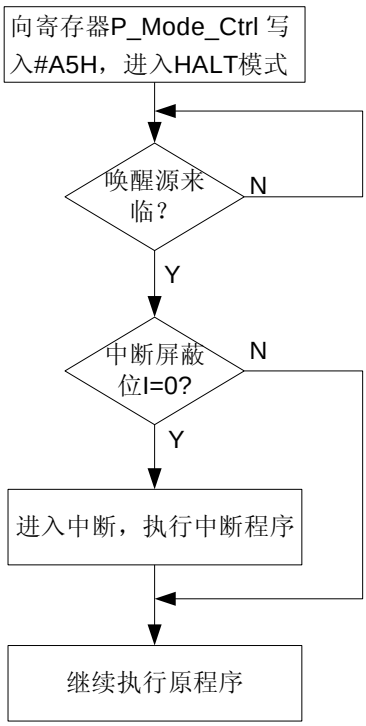


图 21-8 HALT 模式的运行流程

21.3.5 降低功耗

为了降低功耗，芯片采用了 STOP 模式的工作方式。如果暂时用不到一些外围的情况下，可以通过设置，关闭它们的驱动，使没有电流输出。首先，将端口设置成输入口，并确保端口和外围电路的连接中没有电流流过。作为输入端，这些管脚的阻抗从外部看来又非常高，因此也就不会产生电流。正常输入电压应该为 Vss 或 Vdd，但要注意如果输入电压不是 Vss 或 Vdd，就有可能产生微小的电流（2V 时电流最大为 1mA）。因此，考虑到和外围电路的关系，如果将端口设置为输入不合适，那么可以将其设置为输出。输出电平为高电平还是低电平由外围电路决定。例如：如果管脚外接有上拉电阻，可以将其设置为输出高电平。如果外接下拉电阻，可以将其设置为输出低电平。如图 21-9 所示为关于没有用到的端口在输出模式下的连接方式。

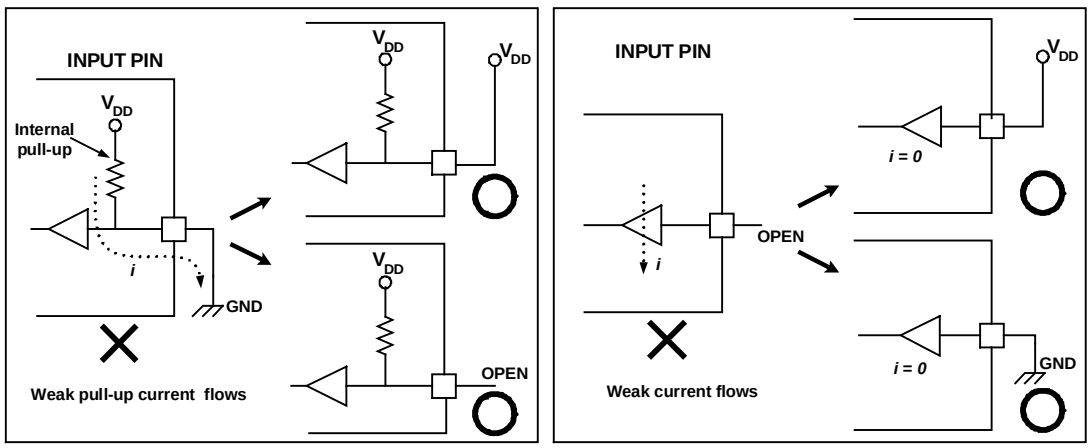


图 21-9 暂时不用端口的应用电路

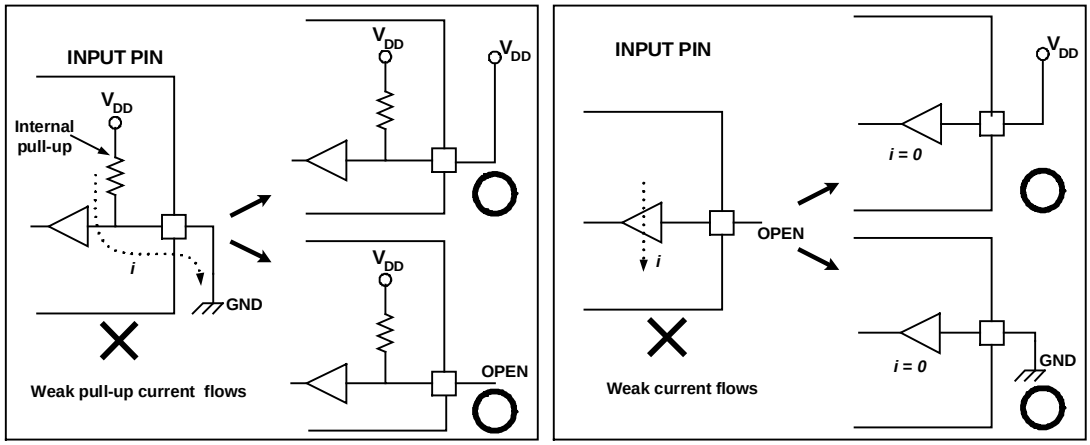


图 21-10 暂时不用端口的应用电路

21.4 省电模式中断

省电模式中无相应中断。

21.5 设计参考

【例 21 3】:以 SPMC65P2404A 为例，令芯片进入 STOP 模式并将 IRQ0 设置为唤醒源，步骤如下：

- 使能 IRQ0 作为中断唤醒源
- 关闭省电时钟功能
- MCU 进入 STOP 模式

```
.SYNTAX     6502                ; process standard 6502 addressing syntax
.LINKLIST                       ; generate linklist information
.SYMBOLS                        ; generate symbolic debug information

.INCLUDE        spmc65p2404a.inc

    .PAGE0                      ; define values in the range from $00 to $FF
;
    .DATA                       ; define data storage section
;
    .CODE
;=====================================
;=                                   =
;= Power on Reset Process - Main Program  =
;=                                   =
;=====================================
V_Reset:
    sei                         ; 关闭总的中断开关
    ldx     #C_STACK_BOTTOM     ; 将堆栈指针初始化为$00FF 位置上
```

```
    txs                         ; 将 X 寄存器的值传送到堆栈指针(SP)中
;
    lda     #$FF                ; 清除复位标志清除复位标志
    sta     P_SYS_Ctrl
    sta     P_SYS_Ctrl
;
    lda     #$00
    sta     P_IOB_Attrib

    lda     #00010000B
    sta     P_IOB_Data

    lda     #11101111B
    sta     P_IOB_Dir           ; 将 PB4 设置为上拉输入，作为 IRQ0 的中断输入
;
     clr    P_WDT_Ctrl,7        ; 关闭看门狗定时器的片内 RC 振荡，节省功耗。
    clr     P_WDT_Ctrl,7
    lda     #$00
    sta     P_IRQ_Opt1
    sta     P_IRQ_Opt1          ; IRQ0 为下降沿触发
;
    lda     #$FF                ; 清除中断请求标志
    sta     P_INT_Flag0
    sta     P_INT_Flag1
    lda     #C_INT_IRQ0E        ; 使能 IRQ0 中断
    sta     P_INT_Ctrl0
    cli                         ; 打开总的中断开关
;
L_Main:
    lda     #C_MODE_STOP
    sta     P_MODE_Ctrl
    sta     P_MODE_Ctrl         ; 执行进入 STOP 模式的命令
    nop
    nop                         ; 必须等待两个 NOPs

    jmp     L_Main
;
;=====================================
;=                                   =
;=   IRQ Interrupt Service Routine          =
;=                                   =
;=====================================
```

```
V_IRQ:
    pha                         ; 将累加器 A 的值压入堆栈
    txa                         ; 将 X 寄存器的值送入累加器 A 中
    pha                         ; 将累加器 A 的值压入堆栈(即 push X)
;
; IRQ0 中断 ?
;
    lda     P_INT_Flag0
    and     #C_INT_IRQ0F
    beq     L_exit_irq
;
    lda     #C_INT_IRQ0F
    sta     P_INT_Flag0
;
L_exit_irq:
    pla                         ; 将堆栈内容弹出给 A
    tax                         ; 将累加器 A 的值送入 X 寄存器中
    pla                         ; 将堆栈内容弹出给 A
;
    rti
;
;=====================================
;=                                   =
;=       Interrupt Vector Table      =
;=                                   =
;=====================================
VECTOR:             .SECTION
    DW      V_NMI               ; may download program emulated either
    DW      V_Reset             ; in internal memory or external memory
    DW      V_IRQ               ; dw define two bytes interrupt vector
;
;=====================================
;=                                   =
;=       End Of Interrupt Vector Table =
;=                                   =
;=====================================
    .END                        ; end of program
```

【例 21 4】:以 SPMC65P2404A 为例，令芯片进入 STOP 模式，将看门狗作为唤醒源，步骤如下：

- 使能看门狗作为唤醒源
- MCU 进入 STOP 模式

```
.SYNTAX     6502                    ; process standard 6502 addressing syntax
.LINKLIST                           ; generate linklist information
.SYMBOLS                            ; generate symbolic debug information

.INCLUDE        spmc65p2404a.inc

    .PAGE0                          ; define values in the range from $00 to $FF
;
    .DATA                           ; define data storage section
;
    .CODE
;=====================================
;=                                   =
;= Power on Reset Process - Main Program  =
;=                                   =
;=====================================
V_Reset:
    sei                             ; 关闭总的中断开关
    ldx     #C_STACK_BOTTOM         ; 将堆栈指针初始化为$00FF 位置上
    txs                             ; 将 X 寄存器的值传送到堆栈指针(SP)中
;
    lda     #$FF                    ; 清除复位标志
    sta     P_SYS_Ctrl
    sta     P_SYS_Ctrl
;
    lda         #C_WDT_Div_512      ; 看门狗中断频率＝F_slow/512 = 48.828 Hz
     sta         P_WDT_Ctrl
     sta         P_WDT_Ctrl
;
    lda     #$FF
     sta     P_INT_Flag0            ; 清除中断请求标志
     sta     P_INT_Flag1
     lda     #C_INT_WDIE            ; 使能看门狗中断
     sta     P_INT_Ctrl0
L_Main:
    lda     #C_MODE_STOP
    sta     P_MODE_Ctrl
    sta     P_MODE_Ctrl             ; 进入 STOP 模式
    nop
    nop                             ; 必须等待两个 NOPs

    jmp     L_Main
```

```
;
;====================================
;=                                  =
;=   IRQ Interrupt Service Routine          =
;=                                  =
;====================================
V_IRQ:
      pha                                ; 将累加器 A 的值压入堆栈
      txa                                ; 将 X 寄存器的值送入累加器 A 中
      pha                                ; 将累加器 A 的值压入堆栈(即 push X)
;
; 看门狗定时器溢出中断 ?
;
    lda     P_INT_Flag0
    and     #C_INT_WDIF
    beq     L_exit_irq
;
    lda     #C_INT_WDIF
    sta     P_INT_Flag0
;
    lda     P_IOA_Data
    eor     #$FF
    sta     P_IOA_Data          ; PA 口输出数据取反
;
L_exit_irq:
      pla                               ; 将堆栈内容弹出给 A
      tax                               ; 将累加器 A 的值送入 X 寄存器中
      pla                               ; 将堆栈内容弹出给 A
;
      rti;
;====================================
;=                                      =
;=        Interrupt Vector Table          =
;=                                      =
;====================================
VECTOR:             .SECTION
    DW      V_NMI               ; may download program emulated either
    DW      V_Reset             ; in internal memory or external memory
    DW      V_IRQ               ; dw define two bytes interrupt vector
;
;====================================
;=                                      =
```

```
;=          End Of Interrupt Vector Table          =
;=                                                 =
;===================================
    .END                                  ; end of program
```

21.6 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题	**应用例系列号**
省电模式	AN_O0346

库函数

标题	**应用例系列号**
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

21.7 修订记录

版本号	日期	备注
V0.1	03/31/2005	第一版

22 开发工具

22.1 说明

Sunplus 致力于为产品开发工程师提供高可靠性和界面友好的开发环境，该开发环境以 SPMC65X 系列微控制器为核心。本章将向用户介绍 SPMC65X 系列芯片的开发工具 FortisIDE™ 和在线仿真器(ICE)的用法。

22.2 Fortis IDE

FortisIDE™ 向用户提供了一个友好和方便的 SPMC65 CPU 内核的集成开发环境，该开发环境用于 SPMC65X 系列芯片的程序的开发。它具有工程管理、文本编辑、程序编译和调试等多种功能，还具备用户界面、下拉菜单、快捷方式和快速访问命令列表等，用户能够方便地编辑、调试程序，且通过其内部集成的仿真芯片 ECMC653A 仿真多种处理器的功能，大大提高了开发效率。

22.2.1 特性

- 内置汇编器和链接器
- 集成的程序编辑器
- 工程管理
- 资源文件管理
- 在当前文件内或磁盘上进行文本的查找 / 替换
- 内置在线仿真器 (ICE)
- 可设置软件断点和硬件断点
- 调试功能(step into、step over、step out、run to cursor)
- 寄存器 / 标志位状态显示
- 存储器的状态显示
- 反汇编功能
- 环境设置

22.2.2 安装

1、系统要求

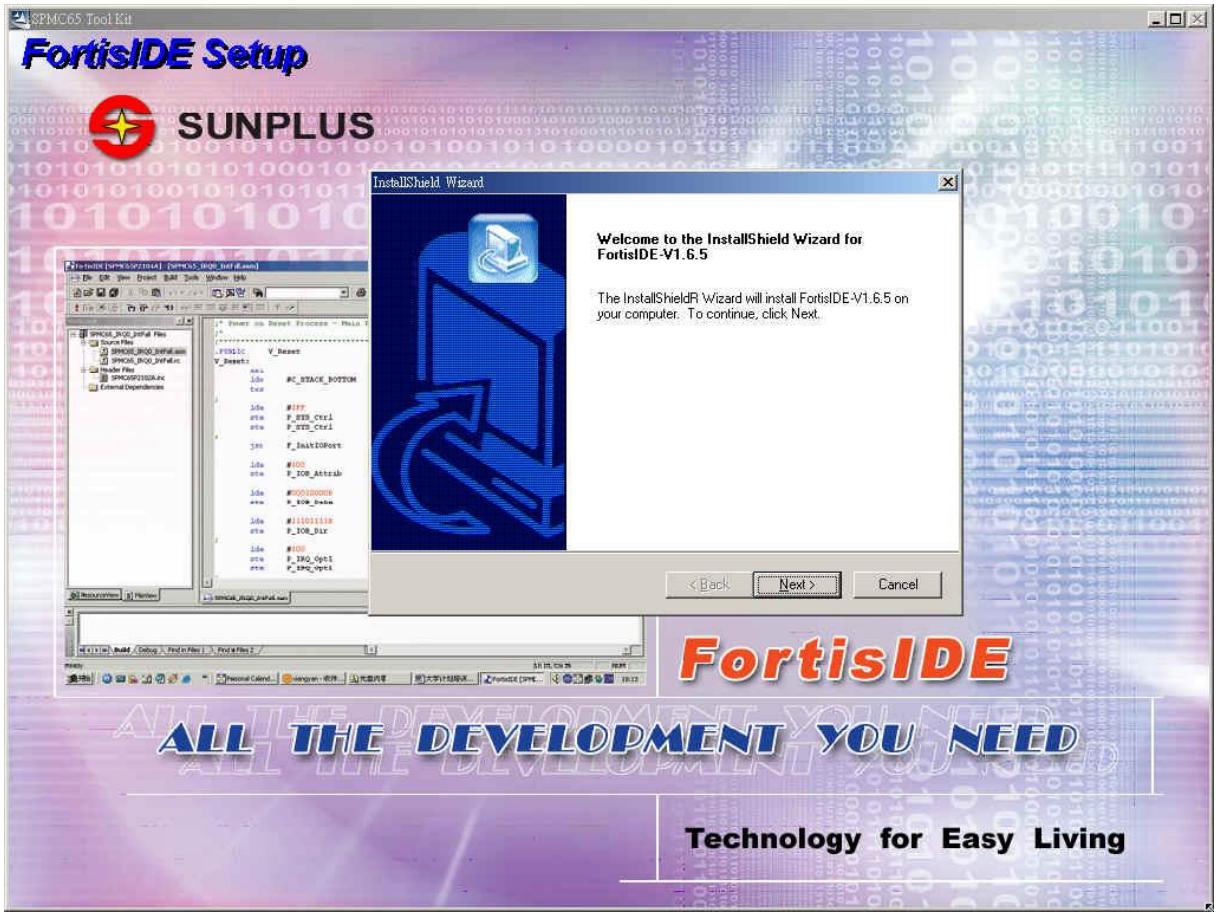
FortisIDE™运行的软件平台为 Windows 98® / 2000®/XP®。

2、安装

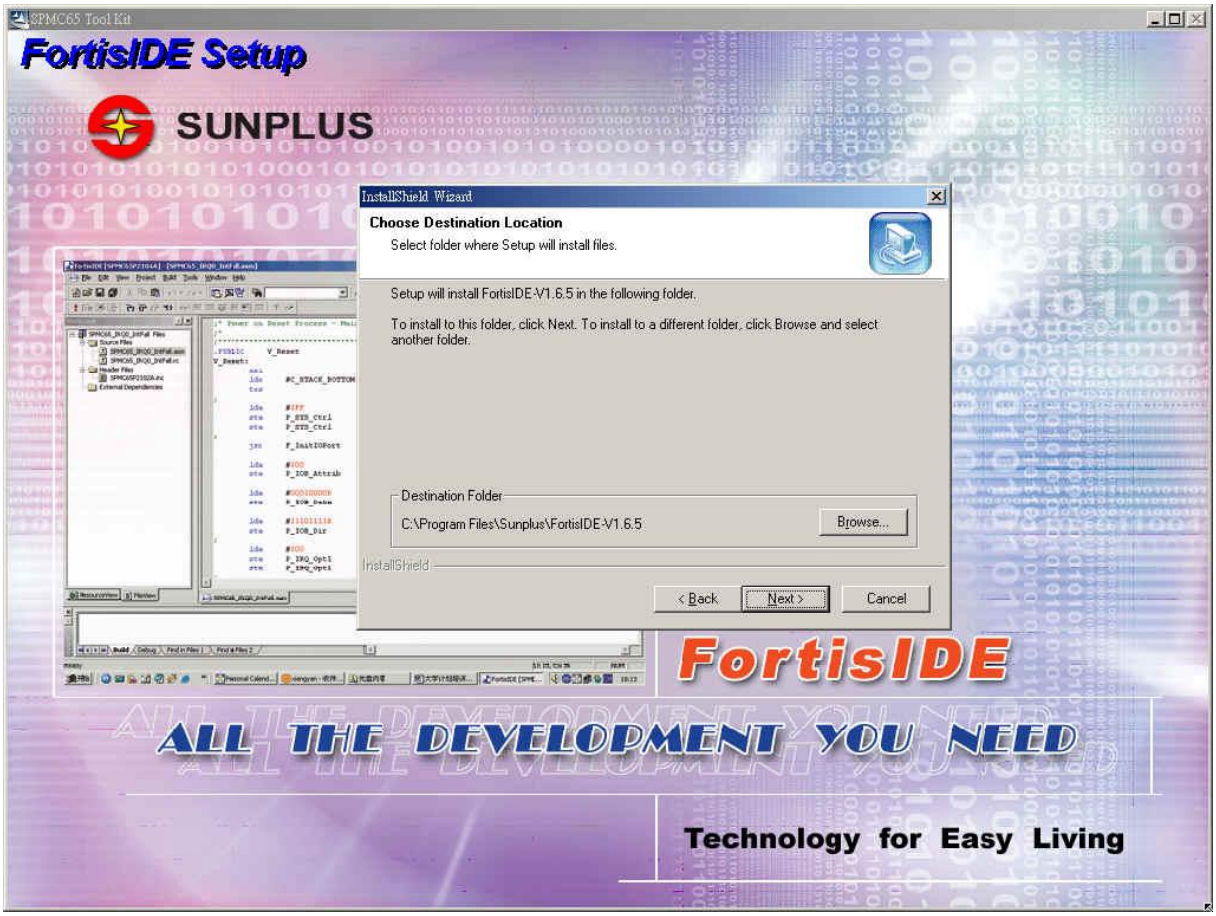
FortisIDE™安装文件会跟随开发系统一起提供给用户。 在本章节里， 将向用户说明如何在 WINDOWS 环境下安装 FortisIDE™。

（1）在 CD-ROM 或者凌阳科技网站获取 Fortis IDE 安装包。

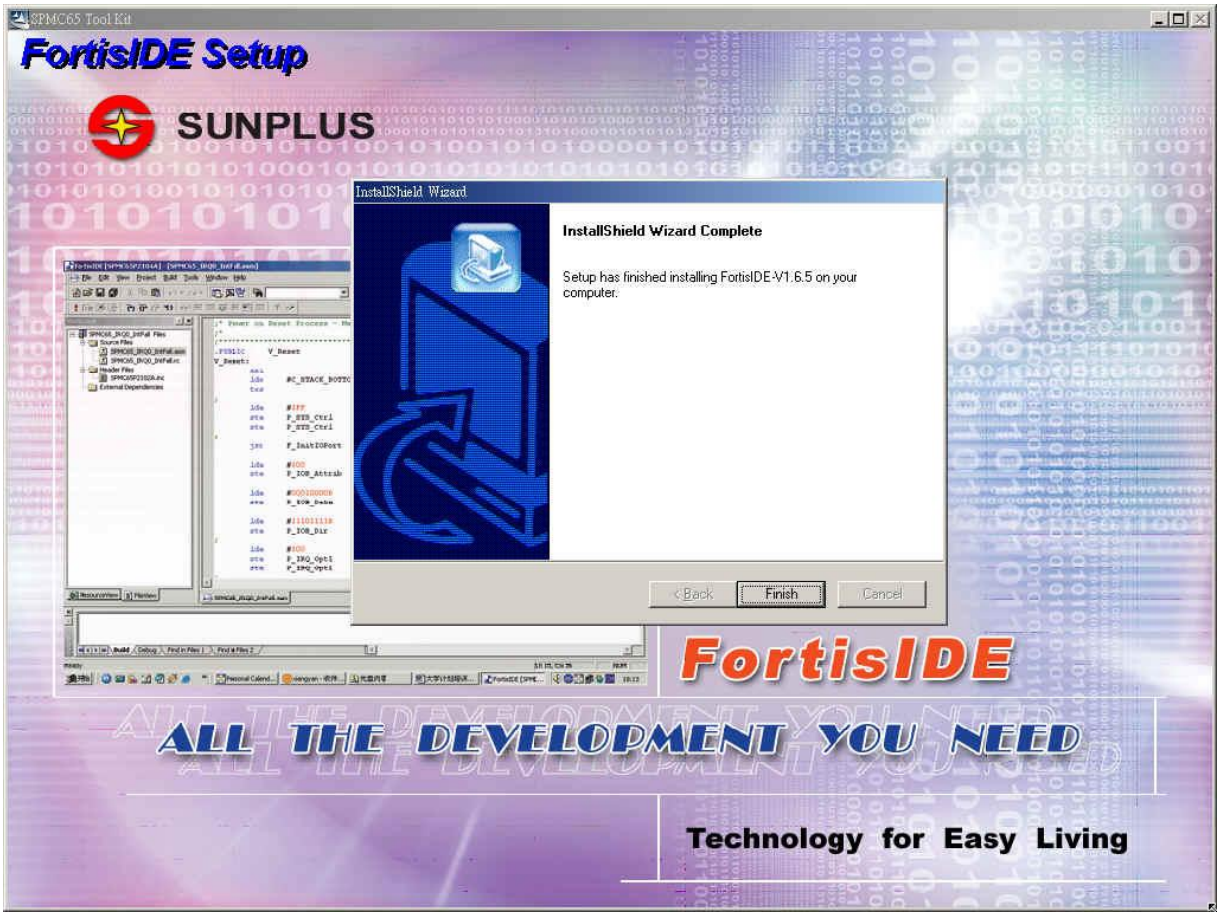
（2）运行 SPMC65 Tool Kit_v1.0.0.exe，屏幕上出现一个如下的画面。



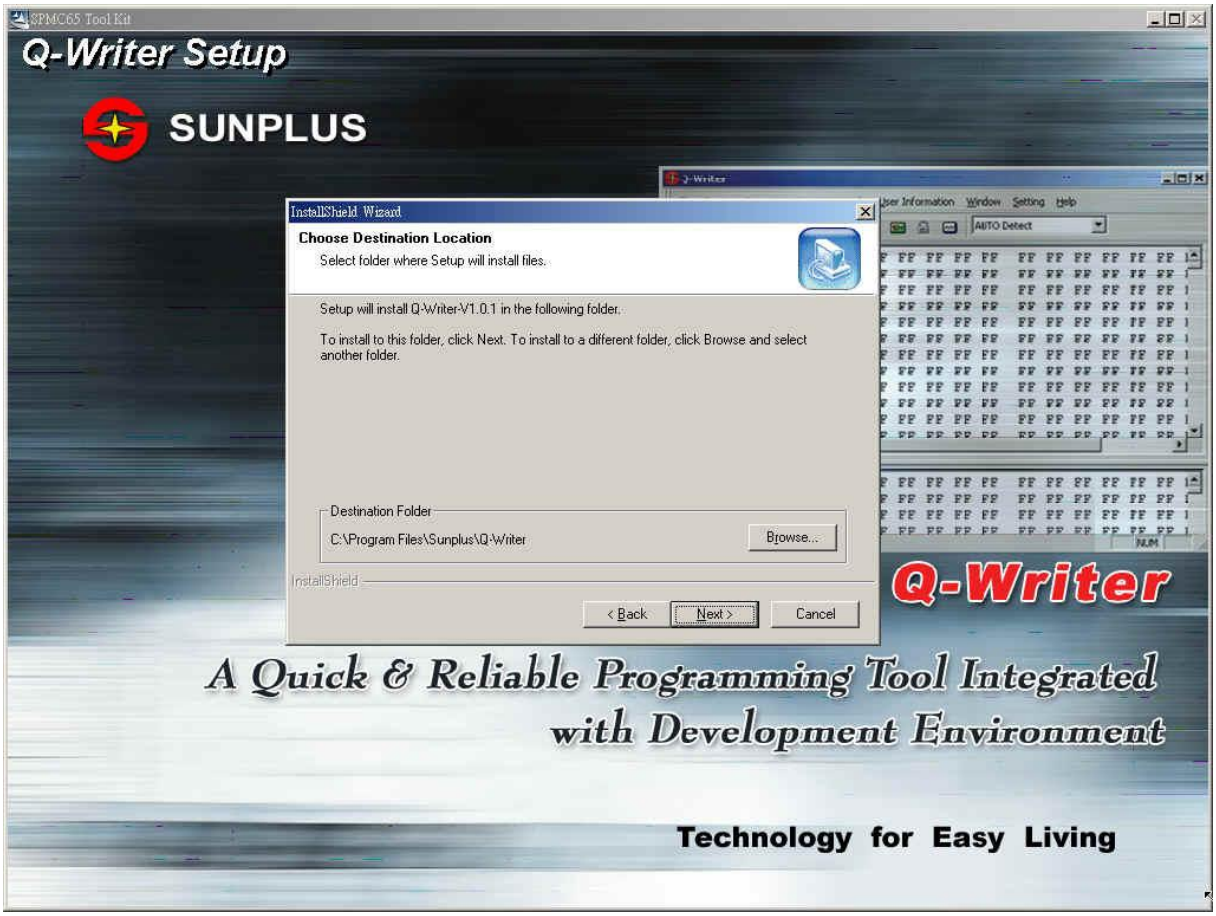
（3）首先，按照下面的提示安装“Fortis IDE Body”。



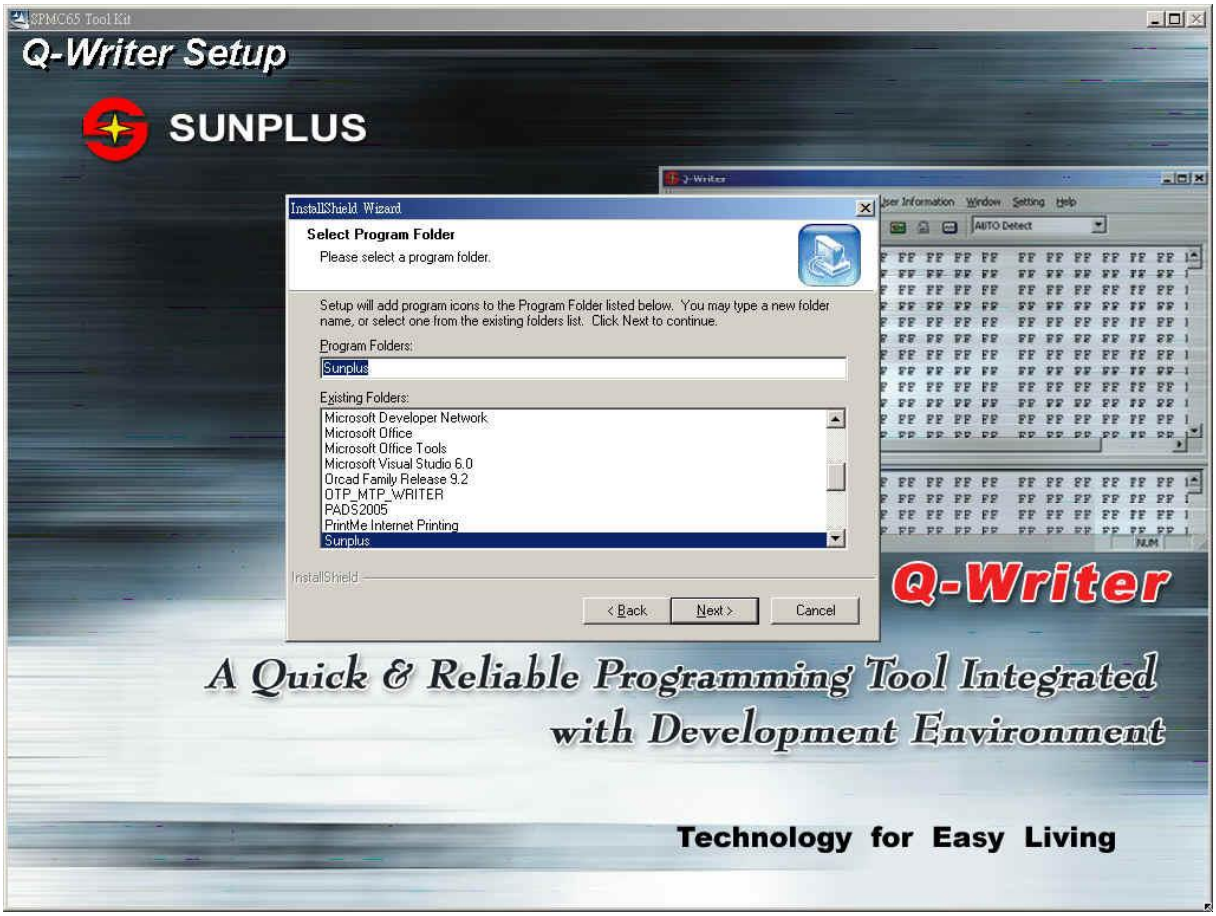
（4）FortisIDE 安装完毕，然后等待大约 5 秒钟，继续安装 FortisIDE Body。


SPMC65 Tool Kit
FortisIDE Setup
SUNPLUS
InstallShield Wizard
InstallShield Wizard Complete
Setup has finished installing FortisIDE-V1.6.5 on your computer.
< Back
Finish
Cancel
FortisIDE
ALL THE DEVELOPMENT YOU NEED
Technology for Easy Living

（5）最后，安装 Q-Writer。



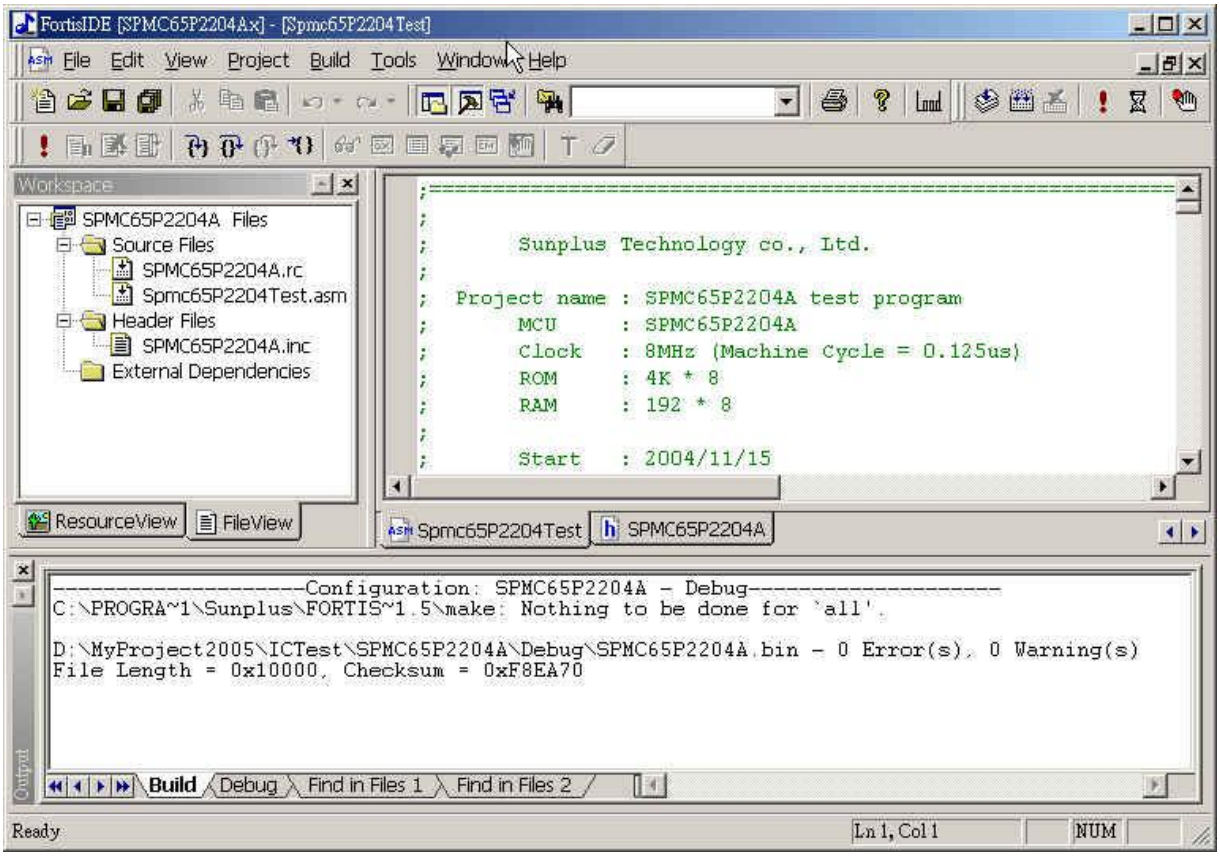
(6)下一步进入安装路径，强烈建议不要改变安装路径。



（7）单击[开始] → [程序] → [Sunplus] → [FortisIDE] → [FortisIDE]，运行 FortisIDE™。

22.2.3 主界面

FortisIDE 包含一系列的用于程序开发的工具，它能让用户方便地在一个集成开发环境内进行程序的编辑、编译、链接和调试。用户能同时打开多个文档，且能在它们之间方便地进行切换。 在主界面上，用户将看到 3 个主要的窗口：Workspace、Output、Editor，用鼠标单击不同的窗口即能够进行窗口的切换。



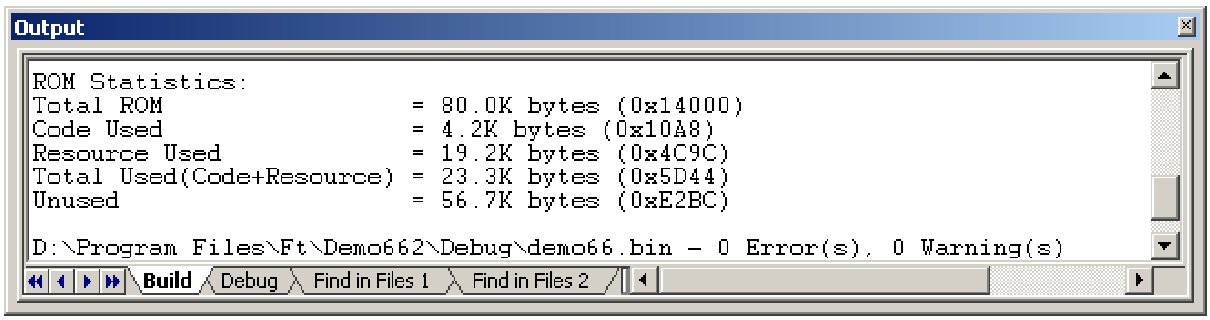
File View

FileView 显示所有的自动生成的文件夹和用户定义的文件夹以及各个文件夹内的文件，并借助于树形目录表明了工程内的所有文件和文件夹之间的逻辑关系。注意，当前的目录结构不表明文件在硬盘上的存储位置。

利用鼠标右键点击最顶端的工程文件名/文件夹名/文件名，可以激活相应的用于方便用户操作的快捷菜单。

Output 窗口

Output 窗口向用户提供了一个方便的浏览编译、程序调试和文本查找的状态的途径。用户单击 Build、Debug 和 Find in Files 等面板名称，可以激活相应的面板。



1. Build:

显示编译和链接过程中产生的信息，例如程序编译过程中产生的错误和警告信息等。如果编译中没有产生错误或警告信息，表示程序成功地通过编译。在报错信息行或警告信息行双击鼠标，即可将光标定位到产生错误和警告的源代码行。

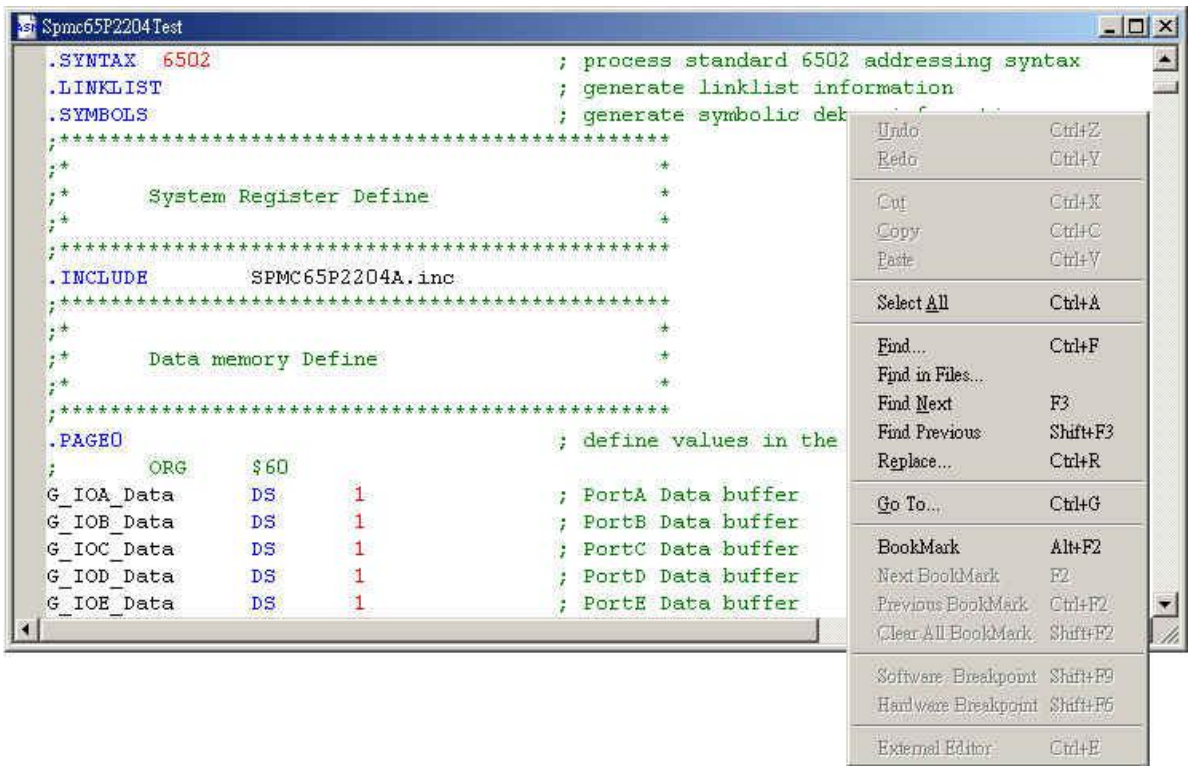
2. Debug:

显示调试过程中的各项信息。

3. Find in Files:

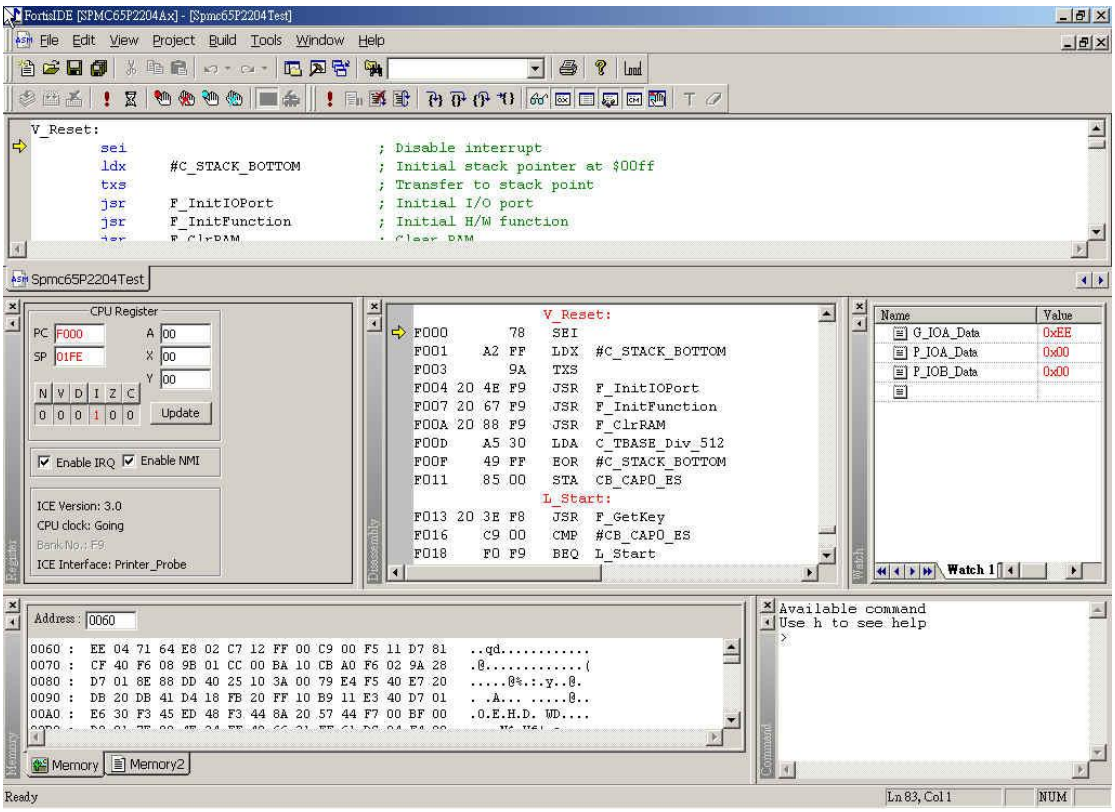
显示查找的结果。 查找结果自动显示在 Find in Files1 面板内。如需显示到 Find in Files2 面板内，用户需点选 Find in Files2。

文本编辑器



用户能够通过文本编辑器窗口开发程序代码。当打开一个文件的时候，其内容就会显示在文本编辑器里。当新建一个工程之后，用户能够通过 Load program 的方法（点选[File] → [Load Program]）向工程之中加载目标文件，也可通过文本编辑器建立新的文件后加载到当前工程内（新建文件时点选 Add to project）。从这个窗口的任一行上单击鼠标右键，都能够弹出如上图所示的快捷菜单。

Debug 窗口



Disassembly 窗口:
显示内存中的反汇编数据。

Register 窗口:
显示通用寄存器和状态寄存器的内容。

Memory 窗口:
显示内存数据。

Watch 窗口:
显示各个变量的值和地址。

Command 窗口:
显示所有的命令。

Breakpoint 窗口:
显示断点信息。

Trace buffer 窗口:
显示已执行过的指令、状态和内存中的数据。

22.2.4 快速启动

1. 运行 FortisIDE。
2. 单击[File] **à** [Open Project]以打开一个工程。 或单击[File] **à** [New]，按屏幕上的提示操作，建立一个新的工程。
3. 工程窗口在主界面的左半边。在这个窗口内用户能看到当前工程所包含的所有文件。双击 File view 内的一个文件名，即可打开相应的文件。
4. 单击[Build] **à** [Rebuild All]对所有的源文件进行编译和链接。如果在编译过程中没有产生错误，继续第 5 步。
5. 单击[Build] **à** [Start Debug] **à** [Download]以便把程序下载到仿真芯片中。然后，利用 Debug 菜单内所提供的调试命令来调试和运行程序。或者，单击[Build] **à** [Start Debug] **à** [Go]，直接让程序全速执行。

22.2.5 创建工程

工程包括创建一个特定程序所要的各种信息。新建立的工程应包含 7 种文件：*.spj、*.rc、*.set、*.env*、prog.lik、*.cmd、*.inc。 一个工程生成后，FortisIDE 自动将这 7 种文件添加到工程内。

***.spj, *.rc, *.set, *.env** (与工程文件同名)：系统文件，不允许用户改变其存储其路径。

Prog.lik：链接器根据它来决定如何链接目标文件，利用外部的编辑器能够对该文件的内容进行改动。

***.cmd** (与工程文件同名)：根据它来决定如何向仿真芯片中下载二进制文件(.tsk/.bin)，即决定 ICE/仿真板上的文件映象。如果该文件有变化，下载的进度不会达到 100%，原因在于下载进度取决于用户选择的 ROM 段。

***.inc**：在*.inc 文件中，对每一个 I/O 端口和硬件寄存器进行了定义；它必须被包含在源代码内。

步骤：

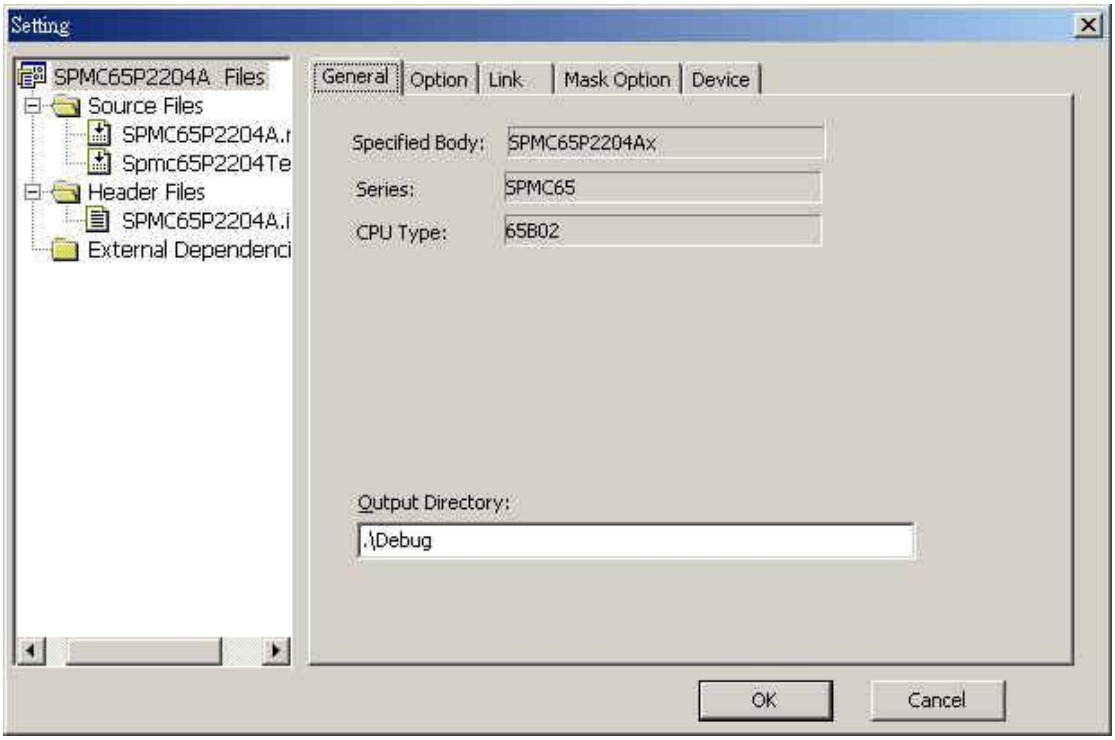
1. 单击[File] **à** [New] **à** [Project]，弹出一个 New 对话框。
2. 在 File 区域输入新工程的名称。
3. 在 Location 区域输入或选取工程文件的保存路径。
4. 单击[Next]，选取芯片的类型。
5. 单击[Finish]。

22.2.6 设定工程

6. 打开一个工程；
7. 选择 [Project]→[Setting]，打开 Setting 对话框；
8. 选择对话框上部的各个属性标签。

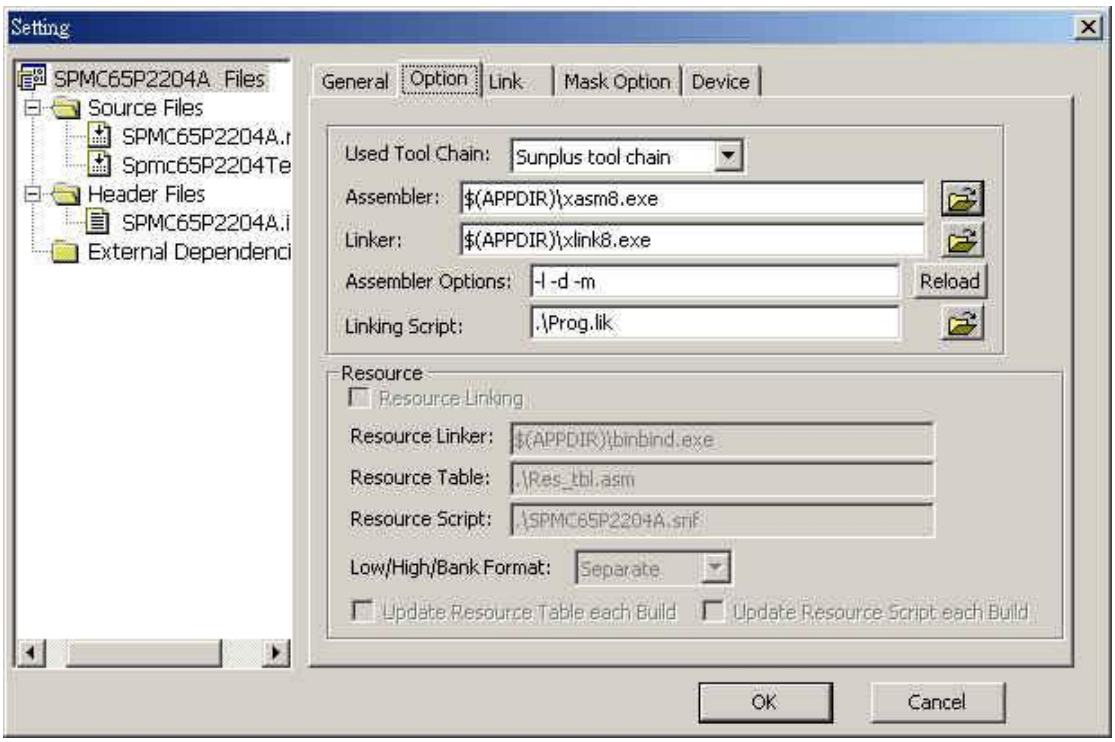
General 页

在当前页里，用户能看到当前工程所用的芯片类型和相应的系列名、CPU 的类型，用户还可为编译过程中生成的目标文件和临时文件设定存储的路径。



Option 页

在 Option 页里，用户能对工具链和资源进行各种设定。



在 Used Tool Chain 里，用户能够选用 Sunplus tool chain 或 Customized tool chain 即外部的汇编器、编译器和链接器。

(1) 汇编器与链接器:

Assembler option 和 linking script 可以在选项框分别进行设置：

Assembler Options：汇编器相关内容的设定，包含 4 个设置参数：

-l: 生成列表文件。

-d: 生成源代码级的调试信息。

-m: 在调试信息生成宏。

-z: 向 DS 寄存器写入 0。

对以上的选项进行设定后，如需要恢复原有的设定，请单击旁边的 Reload 按钮。

Linking Script：设定 Linking Script，用来规定文件映象中各个段的属性。

(2) Resource:

根据用户选用的芯片类型，Resource 会自动激活或屏蔽。

如果资源文件可以被链，用户能够通过 Resource Linking 复选框控制是否需要链接资源。

Resource linking:

如已选 Resource Linking，在编译的过程中，资源链接器会被调用。

Resource Table (Res_tbl.asm):

为每个资源文件保留 3 个字节的空间（地址低位/高位/空），用户须将其包含到源代码内。如用户不需要让 IDE 在每次编译工程时自动更新 **Res_tbl.asm** 的内容，则可以取消选项 Update Resource Table each Build。如需要随时更新 **Res_tbl.asm** 的内容，在 Workspace 窗口的 ResourceView 页的顶级结点上单击鼠标右键且点选 Update Resource Table (Res_tbl.asm)。

'Address low/high/bank'的两种格式都支持，一种是连续的，另一种是独立的。连续的格式是指对于所有的资源文件，'Address low/high/bank'被连续的定位。独立的格式是指对于每一个资源文件，'Address low/high/bank'被单独定位。默认得格式为连续的，可以通过 Low/High/Bank Format'复合框进行重新选择，来改变默认格式。

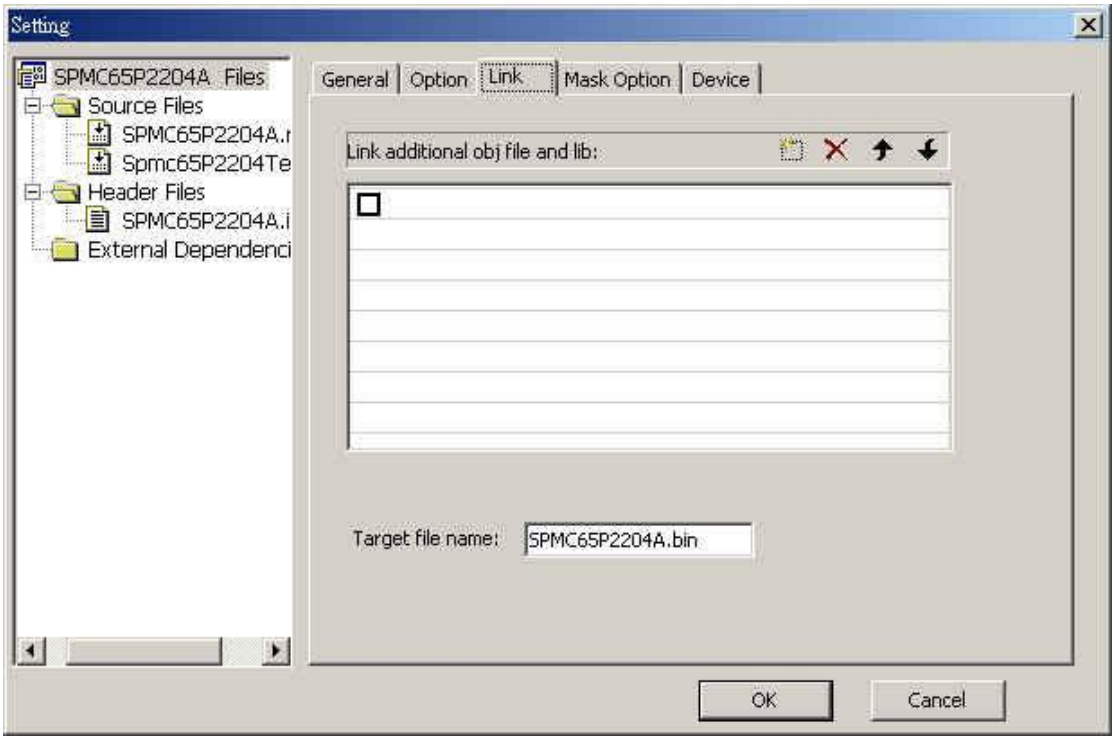
Resource Script (.snf):

sunplus 资源链接器 (binbind.exe)的输入文件。在编译工程的过程中，如果 Resource 被激活，IDE 自动将该文件传递给资源链接器。用户如不需要 IDE 在每次编译工程时均自动更新资源脚本，则取消对 Update Resource Script each Build 的选择。在任意时刻如需更新资源脚本的内容，可点选 Workspace 窗口的 ResourceView 子窗口，再在顶级资源上单击鼠标右键且在弹出的菜单内选取 Update Resource Script (*.snf)项。

对于某些特殊的应用，要用到资源链接器的增强功能。例如带有扩展 ROM 的芯片类型。为配合这种应用，用户必须手动修改.snf 文件且不选中 Update Resource Script each Build 复选框。如需要资源脚本的具体规格，请阅读 binbind 用户说明书。

Link 页

在 Link 页内，用户能够选择需要链接到当前工程的外部的目标文件 (*.obj) 和库文件 (*.lib)。在 Target file name 文本框内，用户能指定所需链接的目标文件的名称和路径。



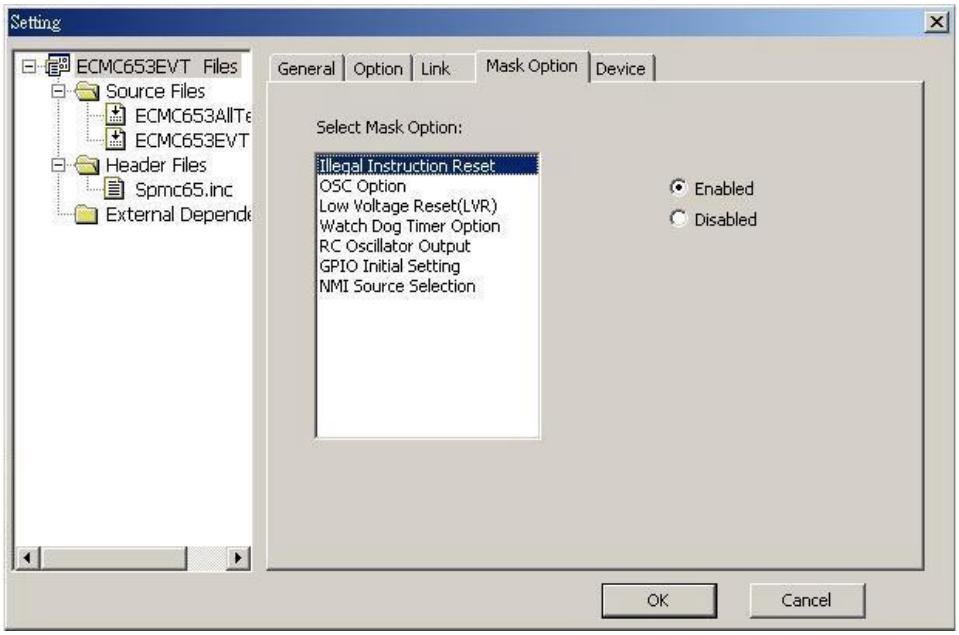
Address space of code crossing bank: (如果无跨 bank 存储的代码段，可忽略这一段的文字)跨 bank 的代码段有两种文件映象地址空间，用于分配代码的存储空间，它取决于链接器的输出文件.s37。其中一个是连续的地址，另一个是不连续的地址。

如果用户在 Address space of code crossing bank 复选框内选取 Sequence address 项，则生成的.s37 映象文件是连续的地址；如果选取 Non-Sequence address 项，则生成的.s37 映象文件不是连续的地址；

对于不连续的文件映象地址，用户须通过运行安装目录内的 binbind.exe / N [BodyName]命令来获得资源链接器的联机帮助。

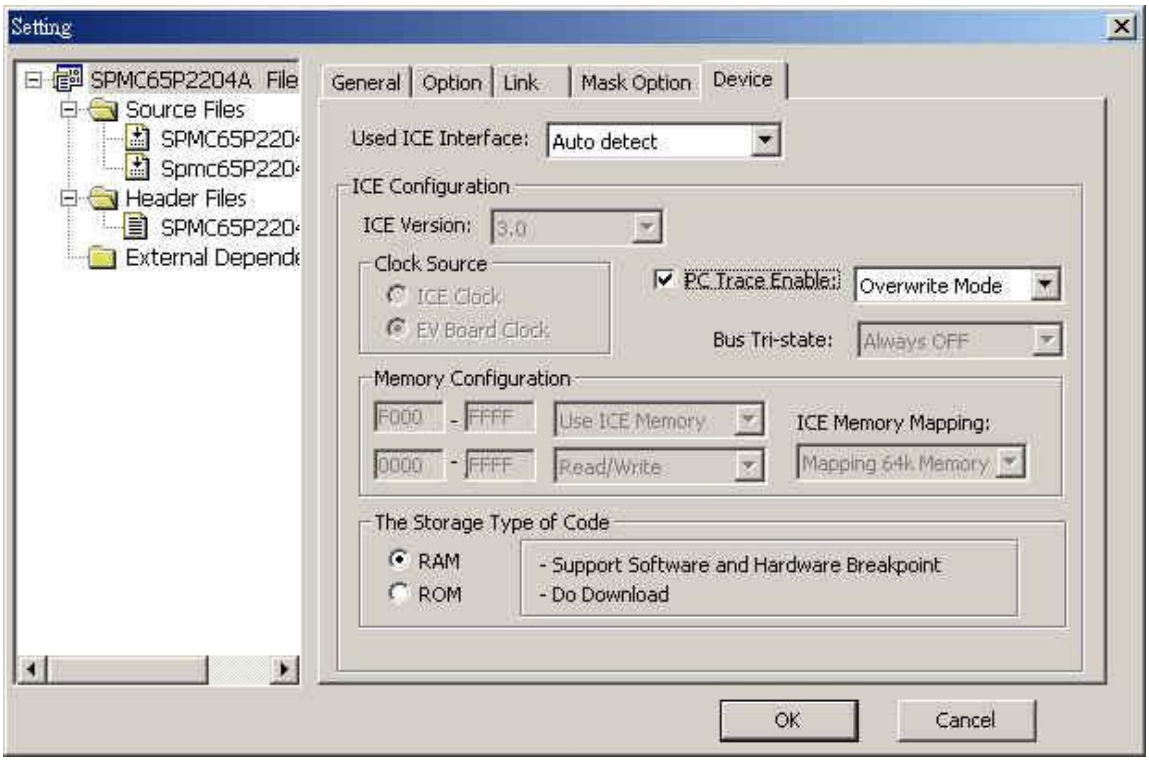
Mask Option 页

编译工程后，FortisIDE 会向目标文件的固定位置写入指定的值。SPMC65X 系列微控制器的当前的硬件设定包括：OSC Option、Low Voltage Reset (LVR)、Watch Dog Timer Option、RC Osillator Output、GPIO Initial Setting 和 NMI Source Setting。用户需根据应用的需要设定合适的选项。



Device 页

选取一个同 ICE 相连的 ICE 接口。 将鼠标定位到 Used ICE Interface 列表框中的一个选项上后，在相应选项的右侧用户会发现它所支持的 ICE 的版本号。如选取 Auto detect，IDE 将自动检查当前所连接的 ICE 接口和 ICE。



ICE Configuration

包含 ICE 的各种信息：ICE 版本号、时钟源、PC trace、总线的三态、存储属性、代码的类型等。在 SPMC65x 系列中，只有一部分是有效的。

ICE 接口

仿真板和电脑通过两种方式进行连接：USB probe 和并口 probe。在烧录环境下，系统会自动检测用户所使用的 probe，用户也可以指定所用的 probe。

PC Trace 使能

（A） PC Trace Buffer 说明

当用户编写的程序不能正确执行时，可以通过 PC Trace Buffer 来进行错误追踪。PC Trace Buffer 用于记录当前最近一段时间程序的运行路经，当程序执行出现问题时，可以通过 PC Trace Buffer 来检查程序在何处发生错误。

（B） PC Trace 特点

PC Trace 是用于捕获 CPU 总线和 IO 口动作的高速 RAM，有以下两个特点：

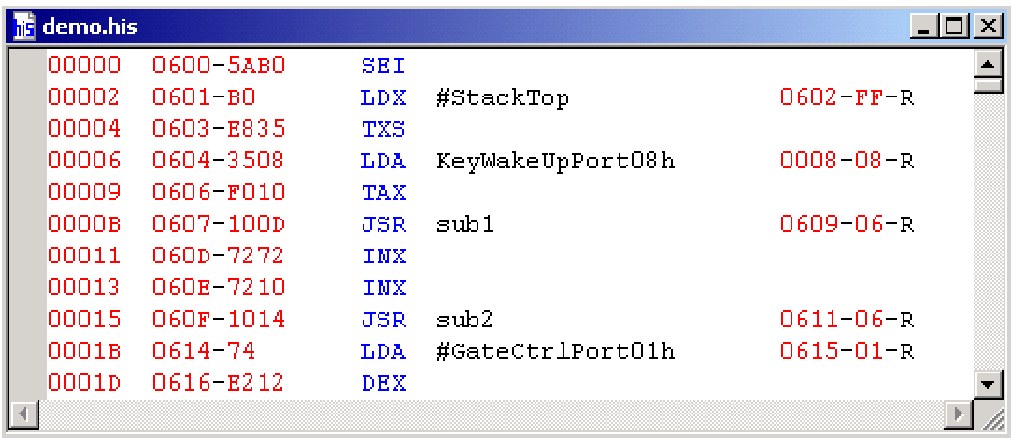
–记录 CPU 的读/写地址和数据，通过 IDE 为程序调试和问题追踪传送数据；

–在 ECMC653A 中 PC Trace buffer 大小为内置的 1K SRAM，用户不需外接任何的扩展芯片；

– PC Trace 有 1K 的可重复记录区域。

（C） 如何使用 PC Trace

首先在 IDE 环境中选中 PC Trace 选项，将其使能。然后，在程序中可能出现问题的地方，用户可以设置两个硬件断点或软件断点，在运行到第二个断点时，在程序调试环境下，点击工具栏中的T，查看 PC Trace 记录，如下图所示，为 PC Trace 记录程序的执行路径。



PC Trace 功能有两种工作模式：Overwrite 模式和 Extended 模式。

1. Overwrite 模式：用于 ICE 3.0 and ICE 2.0，可以记录 1k 的指令周期，可以进行数据的连续记录，当缓冲器填满后，新数据会不断覆盖旧数据。

Extended 模式：用于 ICE 3. 0，可以进行数据的连续记录，但是和 Overwrite 模式有所不同，如果缓冲器被填满，调试系统将会停止，自动的将缓冲器中的数据保存到 PC 中，然后连续填充接下来的 1K 空间。在这种模式下，用于数据记录的空间很大，但是，由于硬件方面的原因，可能会影响到定时/计数器的中断频率，因而，按键检测或者声音播放时可能会出现

误差。所以，建议在实时性要求比较高的场合不要使用该功能。

代码的存储类型

1. 利用 RAM:

ICE 3.0 同时能支持软、硬件两种断点，ICE 2.0 只支持硬件断点。程序中无论是否有断点，下载不受任何影响。

2. 利用 ROM:

ICE 2.0 和 ICE 3.0 只支持硬件断点。下载的过程里，下载的过程里，同样不受断点影响。

22.2.7 工程的管理

向工程内添加文件

方法一：

1. 单击[Project] → [Add File to Project] → [File]，激活 Add File 对话框。
2. 在 File Name 列表之中选取需要添加到工程里的文件的名称后，单击[Add]进行文件的添加。

当 Add Files 对话框被关闭后，FortisIDE™ 通过文件的扩展名自动判断其类型并将其添加到相应的文件夹内。

方法二：

在 Workspace 窗口的某一个文件夹上单击鼠标右键，选中[Add File to Folder]且在相应的对话框之中选择需要添加的文件的名称。

向工程添加文件即将文件名称添加到工程的文件列表内，而并非将添加文件移到或复制到当前的工程中。

删除文件

在 Workspace 窗口内单击某一文件名，按键盘的 DEL 键，即删除相应的文件。

用户还可以通过在 Workspace 窗口内用鼠标右键单击某一文件名，再从弹出的快捷菜单内点选[Remove]删除该文件。

移动文件

通过拖曳的方法，用户能根据自己的需要移动文件。

22.2.8 编译工程

编译

Fortis IDE 的编译功能能够帮助用户编译和重新编译整个工程/工程里的单个源文件。如在 Build 菜单内选中 Build 子菜单，Fortis IDE 将对自前次编译后被改动的源文件进行编译；如选 Rebuild，将编译工程内的所有的文件。

1、单击[Build] → [Compile]，对当前打开的源文件进行编译(它等同于在 Workspace 窗口的*.asm 文件名称上单击鼠标右键后选取[Compile])。

2、[Build] → [Build]，对当前工程进行编译。该功能用于对自前次编译后被改动的源文件进行编译。

3、单击[Build] → [Rebuild]，重新编译当前工程内的所有源文件。

4、单击[Build] → [Stop Build] 用于停止编译。

用户通过 Output 窗口的 Build 面板可以查看编译信息。如果在 Output 窗口内不显示错误信息，说明程序已被编译成功。

Note!

在每个文件中，程序结束后，必须回车，预留一个空行，否则，编译后会出现如下错误。

```
--------------------Configuration: TestIstruct - Debug--------------------
C:\PROGRA~1\Sunplus\FORTIS~1.5\xasm8.exe -Tb -s -l -d -m -z -o ".\Debug\TestInst.obj" "E:\SPMC653 Series\
Sunplus 6502 Assembler - Version 1.6.8 , Copyright(c) by Sunplus.
Apply for 65b02(with bit operation) instruction set
Error: Spmc65.inc: no newline at end of file. Please add <Enter> to the end of file.
1 error(s), 0 warning(s).
```

编译过程里产生的各项信息均显示在 Output 窗口的 Build 页内。包括程序编制过程里产生的一些错误和警告信息。如不显示错误，说明通过了编译和链接。在等待文件编制的过程中，可利用开发环境进行其它操作。不过，此时有些菜单命令和工具栏中的按钮不起作用。

运行

程序被编译成功后，如果电脑与 ICE 板连接正常，便可以直接下载进行程序。

步骤：

1. 单击[Build] → [Start Debug] → [Download]，下载程序到 ICE 板。
2. 单击[Build] → [Start Debug] → [Go]，运行程序。

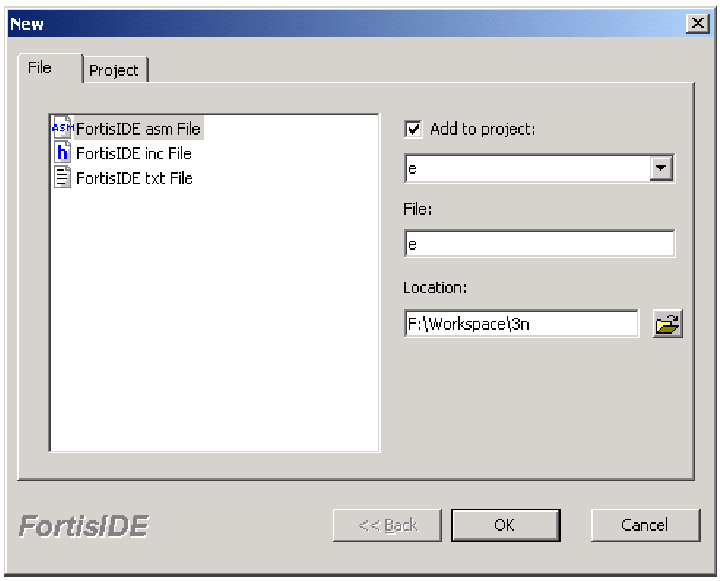
用户还可以利用 [File] → [Load Program]向工程内加载纯代码后再编译、运行。

22.2.9 文本编辑器

1、建立一个新的文件

FortisIDE™ 支持的文件类型包含：asm (汇编文件)、inc (包含文件)和 txt (文本文件)。

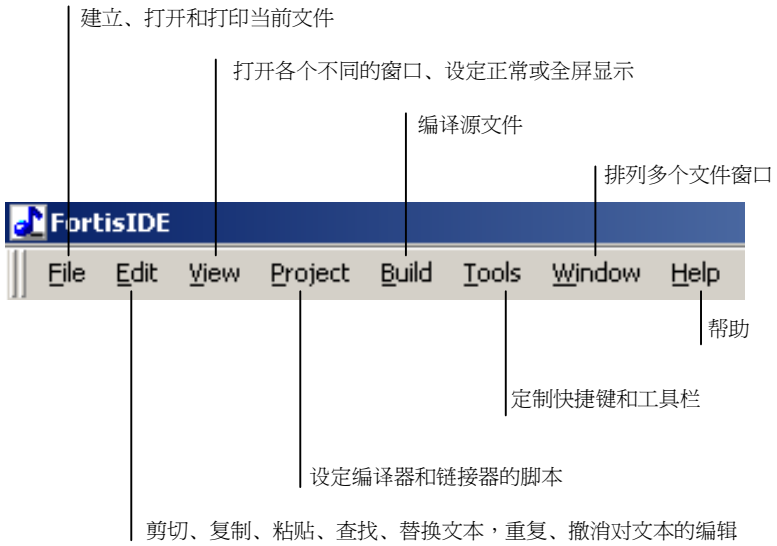
单击[File] → [New]，弹出一个 New 对话框，如图。



1. 在左部的文件类型列表内选取所要建立的文件的类型。
2. 如需将该文件添加到某个工程内，还需要点选 Add to project 复选框，且在列表内选取一

个工程名。
3. 在 File 和 Location 文本框内分别输入文件的名称和存储的路径。
4. 单击[OK]。

对于所有的 FortisIDE™ 编辑窗口，菜单都是通用的。FortisIDE™ 会根据当前编辑窗口的类型自动显示不同的菜单项。在下图中，用户能看到提供给文本编辑器的各个菜单项为一些基本的菜单说明。



2、打开一个文件

FortisIDE™ 支持多文档机制，在编辑器内，用户能同时打开多个文档。当窗口被设定成最大化时，文件的名称显示在屏幕顶部的标题栏内。按 Ctrl + F6/单击 Window 菜单内的文件名称，可以方便地进行文件的切换。

FortisIDE 提供了两种打开文件的方法：

1. 单击[File] → [Open]，弹出 Open 对话框，在对话框内选取所需的文件。
2. 单击[File] → [Recent Files]，在当前的 Recent File 列表内选取一个文件后，该文件自动被打开。 Recent Files 菜单最多列出 8 个近期内被打开过的文件的名称。

3、多文档窗口布局

集成开发环境有两种不同的显示方式：常规方式、全屏方式。当需要切换到全屏方式时，单击[View] → [Full Screen]。这时，标题栏和状态栏都自动隐藏。如需返回到常规的显示方式，按[Esc]或单击浮动的 Full Screen 按钮。

如需在不同的文件之间进行切换，单击 Window 菜单且在文件名列表内选取目标文件名。

用户还可以将多个打开的文件按不同的方法进行排列：

1. 单击[Window] → [Cascade]/[Tile]，以层叠或平铺的方式排列。
2. 单击[Window] → [New Window]，生成一个当前文件的副本且在另一个编辑窗口内打开该文件(在每个窗口内都有各自独立的滚动条和光标)，再单击[Window] → [Tile]，按平铺的方法进行排列。 该副本的文字与源文件的文字保持同步的变化。

4、保存文件

当正在被编辑的文件的标题栏上文件名称后面有“*”标号时，表示本文件正在被编辑且最新的修改还未被保存。这时，用三种方法保存文件：

1. 单击[File] → [Save]，将文件按当前的名称进行保存。
2. 单击[File] → [Save As]，将文件以不同的名称进行保存。
3. 单击[File] → [Save All]，保存当前所有打开的文件。

保存后，“*”标号会自动消失。如果没有保存，当关闭该文件时，会弹出一个对话框提示你对文件进行保存。

5、书签

在文本编辑器内，可以在程序的任意行插入书签作为标志，以便能快速定位。

书签的设定和删除操作都非常简单。

设置书签

(1) 把光标移到指定的行，选择 [Edit]→[Bookmark]或按 Alt + F2，这时，在当前行的左侧会出现一个蓝色的圆圈，表示书签已经被设置。

(2) 选择 [Edit]→[Previous Bookmark]，查找到前一个书签所在行；

(3) 选择 [Edit]→[Next Bookmark]，查找到后一个书签所在行。

删除书签

(1) 把光标移到书签所在行处，选择[Edit]→[Bookmark]或按 Alt + F2 以便删除当前行上的标签。

(2) 选择 [Edit]→[Clear All Bookmark]或按 Shift + F2，清除所有的书签。

(3) 关闭当前文件，也可以清除所有的书签。

6、查找文本

文本编辑器提供了 3 种文本查找的方法：

1. 在当前文件内查找文本。
2. 在当前文件内对文本进行替换。
3. 在磁盘的所有文件内查找文本。

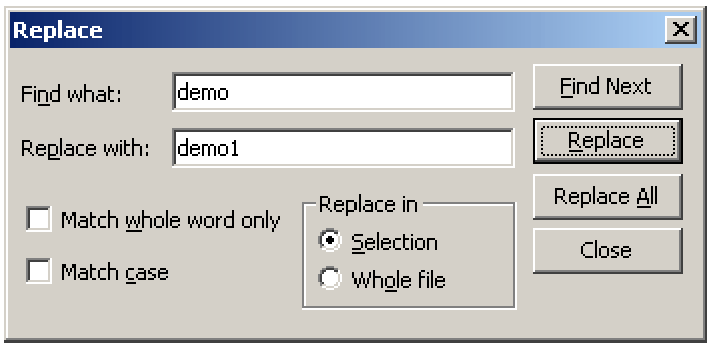
在当前文件内查找文本

1. 单击[Edit] → [Find]，弹出 Find 对话框。

2. 在 Find What 区域，输入要查找的字符串。
3. 选择匹配的条件 Match Case 或 Match Whole Word Only 即区分大小写或全字匹配。
4. 选择查找的方向 Up 或 Down 即向上或向下。
5. 单击[Mark All]，用书签标明所有匹配的字符串。单击[Unmark All]，取消所有匹配字符串的书签标识。
6. 单击[Find Next]，光标指向下一个匹配的字符串。

在当前文件内对文本进行替换。

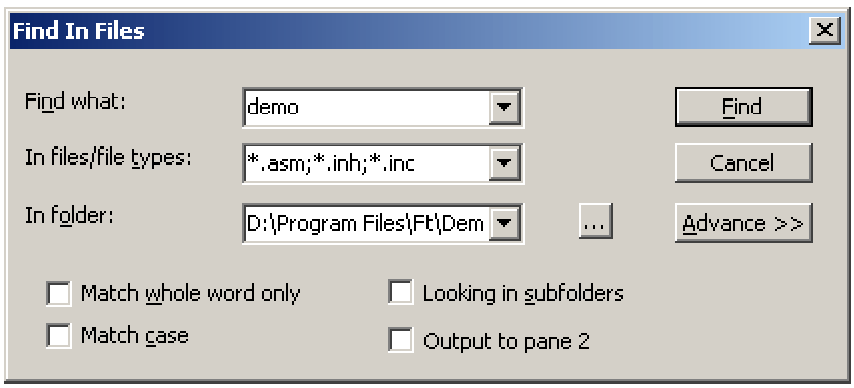
1. 单击[Edit] → [Replace] ，弹出 Replace 对话框。



2. 在 Find what 区域输入需要被替换的字符串。
3. 在 Replace with 区域输入替换后的字符串。
4. 选择匹配的条件 Match Case 或 Match Whole Word Only 即区分大小写或全字匹配。
5. 在 Replace in 内，单击 Selection 或 Whole file 项，以指定查找的范围为当前所选的内容或整个文件。
6. 单击[Find Next]，光标指向下一个匹配的字符串。
7. 单击[Replace]/[Replace All]，替换当前找到的或所有的匹配的字符串。

在磁盘的所有文件内查找文本。

1. 单击[Edit] → [Find In File]，弹出 Find In File 对话框。



2. 在 Find What 区域，输入要查找的字符串。

3. 默认的范围即当前工程文件夹，用户还可通过利用浏览按钮选择路径。
4. 在 In files/file types 列表内选取文件的类型。
5. 选择匹配的条件 Match Case 或 Match Whole Word Only 即区分大小写或全字匹配。
6. 选择 Looking in subfolders，在指定文件夹及其内所有子文件夹中查找。
7. 选择 Output to pane 2，查找的详细结果在窗口 Find in File2 内显示，在 Find in File2 中双击查找到的结果，光标会直接调转到程序中该结果所在的位置。如果取消选中 Output to pane 2，默认的结果会在 Find in File1 面板内显示。可以将两个不同字符串的查找结果分别放在这两个面板内。
8. 单击[Advance]，设定更多的查找路径。
9. 单击[Find]。

22.2.10 应用调试器

应用调试器，用户能非常轻松地找到程序中的逻辑和语法错误，并且可以查看变量及寄存器的值，显示内存的状态等，进而来监视程序的运行。同时还可以连续或单步执行程序、设置断点等。

1、控制程序运行

控制程序运行的目的是为迅速查找程序中存在的错误。用户可以通过单步执行程序或查看变量等各种调试方法来发现程序中的错误。

步骤：

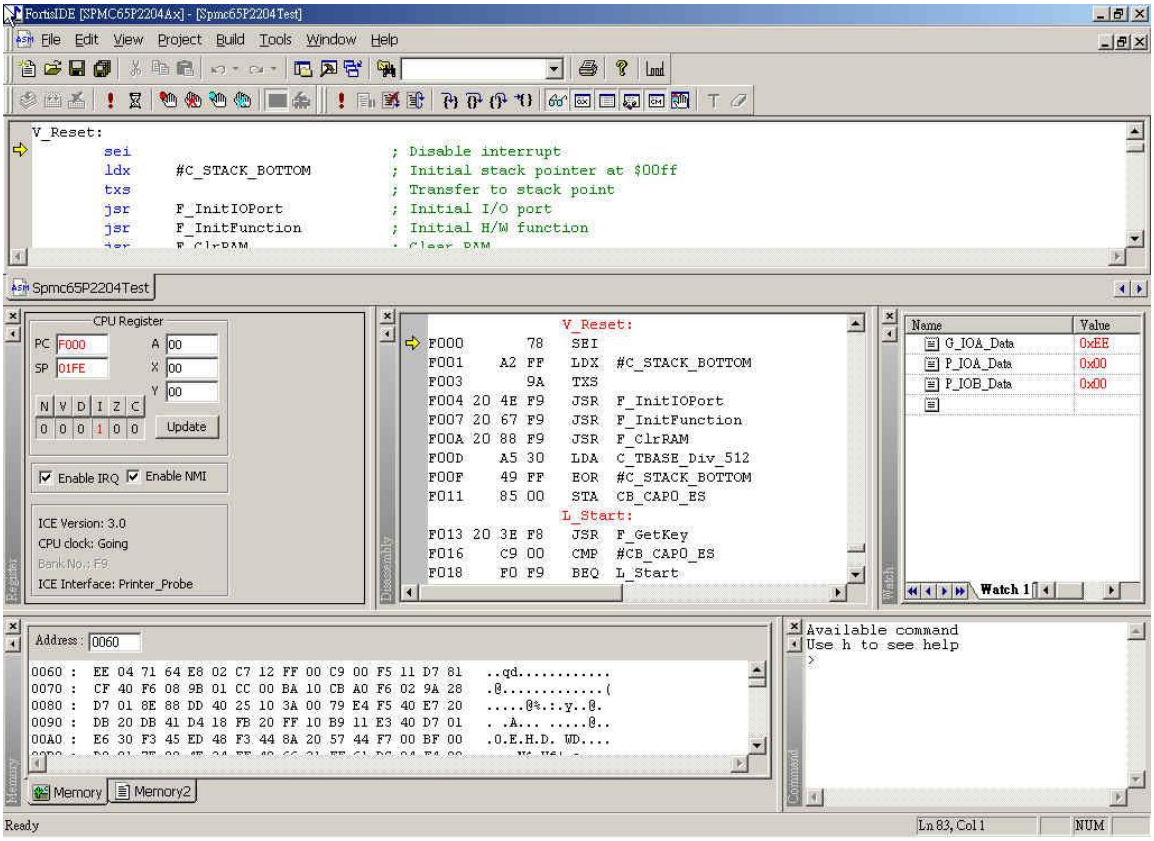
1. 首先，选择 [Build]→[Start Debug]→[Download]，把用户程序的可执行代码载入到内存。

2. 进入调试过程后，利用 Build 菜单里的 Start Debug 子菜单内的各项菜单进行调试。Start Debug 子菜单内各命令和对程序运行的控制操作列在这里：

Go (F5)	从当前光标所在位置开始全速运行程序，直到运行结束或遇到断点才停止运行。
Restart	复位 CPU 和程序计数器，从起始地址重新运行程序。系统会自动保存所有调试状态的设置，以便重复利用。
Stop Debug	停止程序的运行，返回到编辑状态。
Break	暂停运行。
Step Into	单步运行程序，如其后的一条指令为子程序调用，会进入子程序继续单步执行。
Step Over	单步运行程序，如其后的一条指令子程序调用，会执行完整个子程序，然后指向子程序调用的下一条指令。
Step Out	单步运行程序，如果该指令在一个子程序中，会将该子程序执行完毕，返回子程序调用的下一条指令。
Run to Cursor	从当前指令行执行到光标所在的指令。

2、Debug 窗口

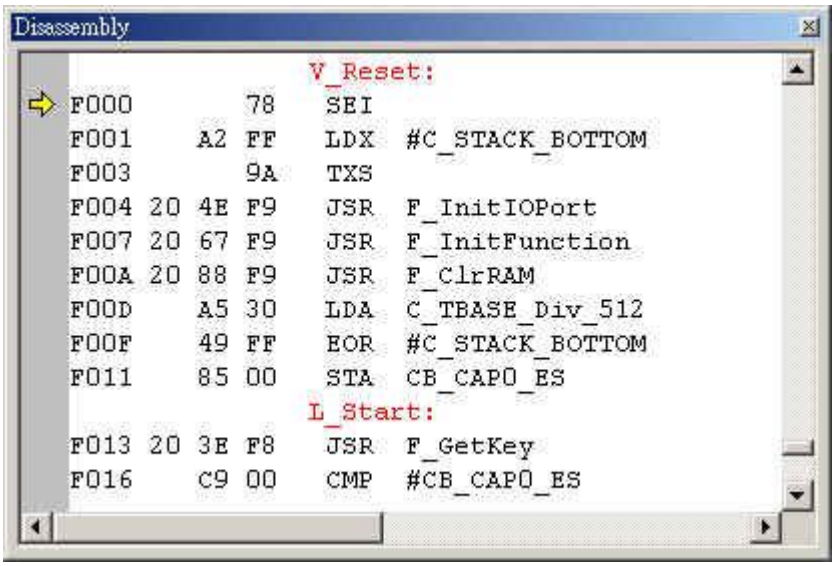
进入调试状态后，用户用[View] → [Debug Windows] 或工具栏上的相应图标能够激活各个调试窗口。调试窗口可用于查看、修改 CPU 寄存器、内存、IO 端口的信息。



注意：如果用户需要同时浏览多个调试窗口，可以根据自己的需要随时调整各个窗口的尺寸、改变各窗口的位置。

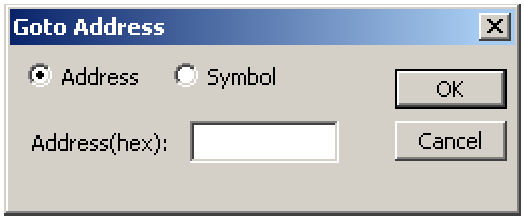
1. Disassembly 窗口

Disassembly: [View] → [Debug Windows] → [Disassembly]



显示内存中程序的反汇编代码，Disassembly 窗口的内容会根据程序计数器里的地址变化而变化。单击鼠标右键，能够激活相应的快捷菜单。

Goto：激活 Goto Address 对话框，快速跳转到指定地址的反汇编行，从指定的地址行开始显现反汇编指令。

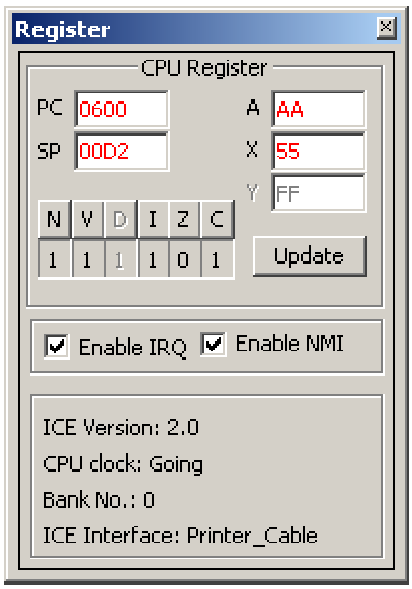


Return to PC：恢复显示当前程序计数器指向的地址单元的数据。

Show Code Byte：显示每条指令的机器码。

2. Register 窗口

Register: [View] → [Debug Windows] → [Register]



Register 窗口显示各个通用寄存器和 CPU 状态寄存器的值、ICE 版本、ICE 接口、CPU 的时钟的状态、bank 数目 (bank 数目仅在运行在 ICE3. 0 时才会显示)。用户可以对其中一些信息进行修改，改动后的内容会被立即传递给 ICE 或仿真板。

PC：程序计数器

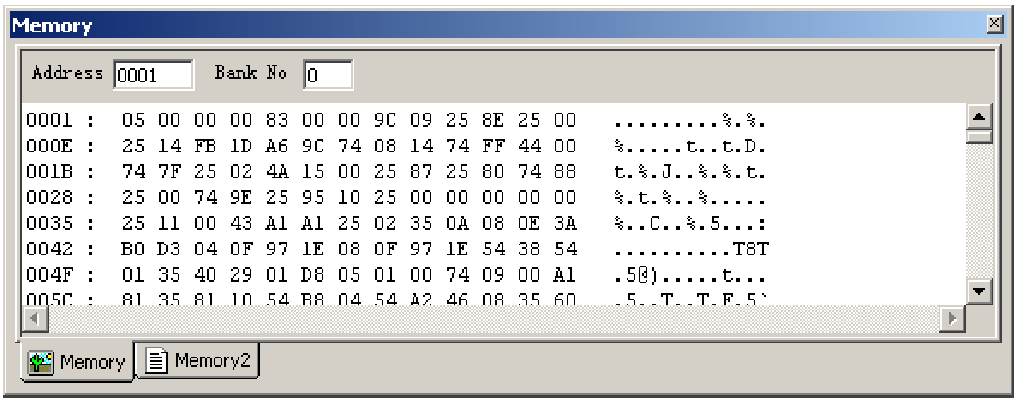
SP：堆栈指针
A, X, Y：通用寄存器
N, V, D, I, Z, C：负标志、溢出标志、十进制模式标志、中断标志、零标志、进位标志
Enable IRQ/NMI：激活/屏蔽 IRQ/NMI 中断
ICE Version：当前所用 ICE 的版本号
CPU clock：CPU 时钟的状态(停止/运行)
Bank No.：bank 号
ICE Interface：当前所用的连接到 ICE 的接口
[Update]：将各项设定数据传递给 ICE 仿真板

3. Memory 窗口

Memory: [View] → [Debug Windows] → [Memory]
Memory 窗口包括两个子窗口：Memory 和 Memory2，用于显示存储器内的数据。

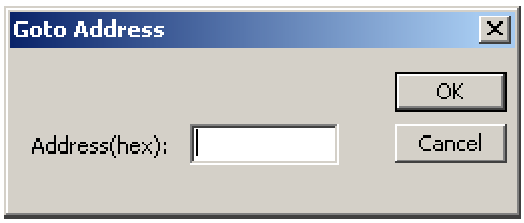
Memory

Memory 窗口以十六进制的形式显示存储器内的数据。如需在 Memory 窗口内查看指定存储单元的数据，可以在 Address 和 Bank No 区域内输入起始地址和 bank 信息，Memory 窗口会自动显示从该起始地址开始的存储器数据。利用窗口内的滚动条，可以查看任一内存单元的值。而且，用户能直接在点击某一需要变动的内存数据后进行编辑。



在当前窗口内单击鼠标右键后，用户能够选取 Goto 和 Dump 选项来启动相应的功能。
(1) Goto
Goto 功能让用户能够快速地查看指定存储单元的数据。

具体的作法：
在 Memory 窗口的编辑区内单击鼠标右键，且在弹出的快捷菜单内选取 Goto，弹出 Goto Address 对话框。用户输入一个地址后，Memory 窗口显示从当前指定的地址单元开始的连续的多个存储单元的数据。

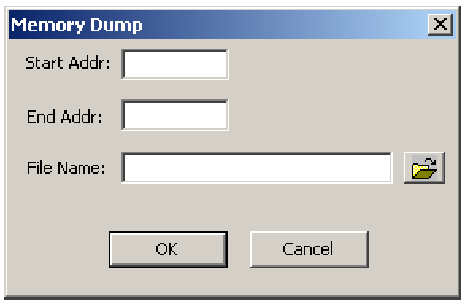


(2) Dump

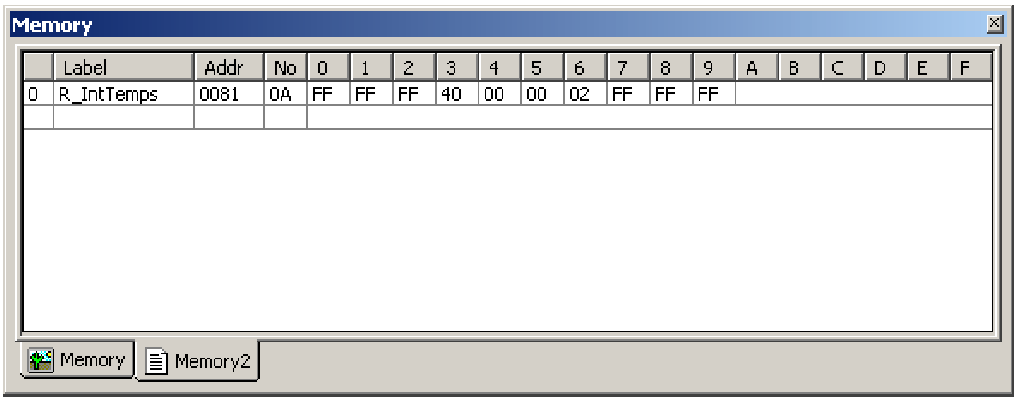
IDE 提供了转存功能，可以将指定地址范围内的内存数据存储到文件内。

具体的作法：

在 Memory 窗口的编辑区内单击鼠标右键，且在弹出的快捷菜单内选取 Dump，弹出 Memory Dump 对话框。用户在窗口内指定地址范围及数据的存储路径且单击[OK]即可。



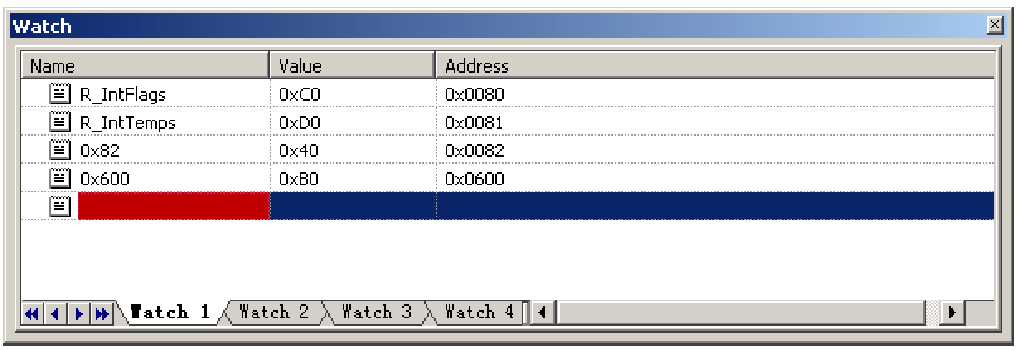
Memory2



Memory2 窗口用于显示连续的多个字节的值。在 Label 栏内输入变量名称，该变量的第一个字节地址会自动被默认为起始地址，或在 Address 栏内直接输入起始地址，No 栏里输入所要显示的字节数，Memory2 窗口便会显示从起始地址开始连续的多字节值。

4. Watch 窗口

Watch: [View] → [Debug Windows] → [Watch]



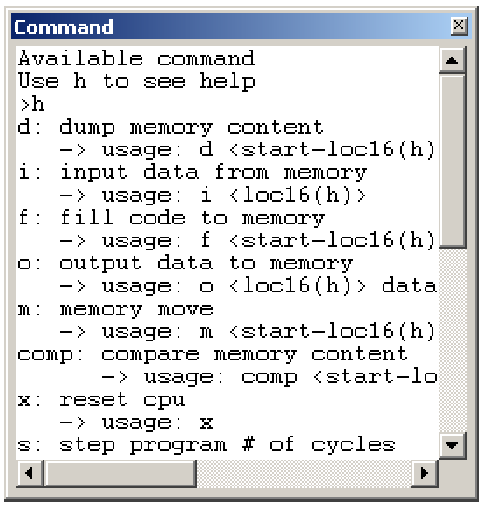
在进行程序的调试时，用户能够利用 Watch 窗口查看和更改变量的值。
Watch 窗口由 Watch1、Watch2、Watch3 和 Watch4 组成。每一个视窗利用一个数据表来显示变量的地址和值。
向 Name 栏内输入变量名称后，Value 栏和 Address 栏会自动显示该变量的值和地址。用户还可以根据自己的需要对 Value 栏里的变量值进行修改，方法非常简单：将光标定位在一个 Value 上，再输入新的数据。如果某个变量的数据显示为红色，表明它刚刚被修改过。

在 Watch 窗口单击鼠标右键，可以激活相应的热键菜单。利用 Paste、Show decimal/hexadecimal、Hide、Add、Remove 和 Remove all，用户可改变数据的显示格式、添加和删除数据、控制 Watch 窗口的显隐。

Paste：从编辑窗口剪切或复制一个变量名并粘贴到 Name 栏。
Show Decimal/Hexadecimal：设定变量值的显示格式为十进制或十六进制。
Hide：隐藏 Watch 窗口。
Remove：删除当前光标所在的行。
Remove All：删除所有的行。

在这里，用户还可以调整每个栏位的宽度以便能让信息显示得更全。

5. Command 窗口
Command: [View] → [Debug Windows] → [Command]



在'>'后键入命令 H，用户能够获得所有命令及其相应的用法说明。
命令:

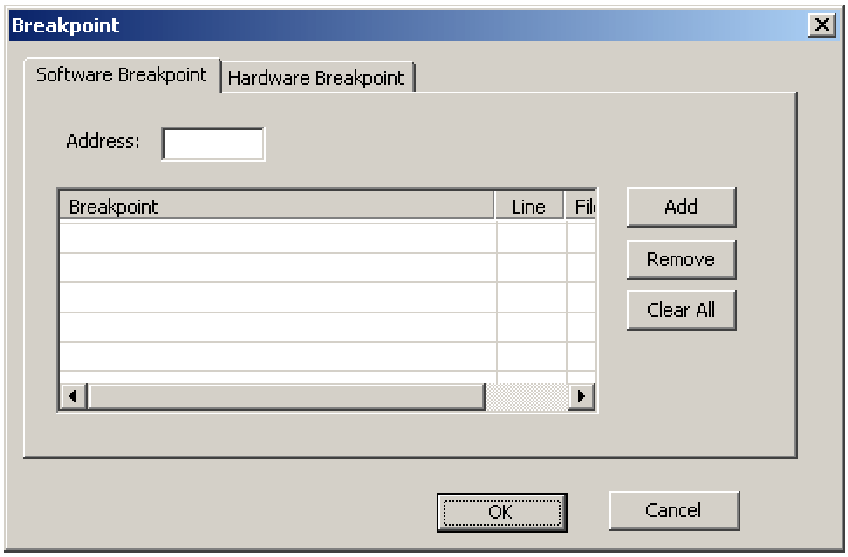
d：转存内存中的数据
l：从内存读出数据
f：向内存写入数据
o：从内存中输出数据
m：内存中数据搬移
comp：比较内存中的内容
x：复位 CPU
s：执行设定数目的指令
g：全速执行程序
ft：追踪程序的执行过程（仅用于 ICE2.0)
clk：设定 CPU 时钟源
ms：设定存储块的属性
cls：删除在窗口内所列的命令

6. Breakpoint 窗口

Breakpoint 窗口包含两部分：Software Breakpoint 窗口和 Hardware Breakpoint 窗口。在相应窗口里，用户能够任意添加、删除、激活、屏蔽、查看、改变断点。断点信息是工程的不可缺少的内容。设定一个硬件 / 软件断点后，在指令行前会出现蓝点 / 红点。

Software Breakpoint 窗口

单击[Edit] → [Software Breakpoint]。



描述:

1、软件断点只能支持指令中断。即断点只能设在指令行。
2、软件断点的数量没有限制。
3、ICE3. 0 支持软件断点，它使用 RAM 区来存储程序代码(参考代码的存储类型)。

用法：

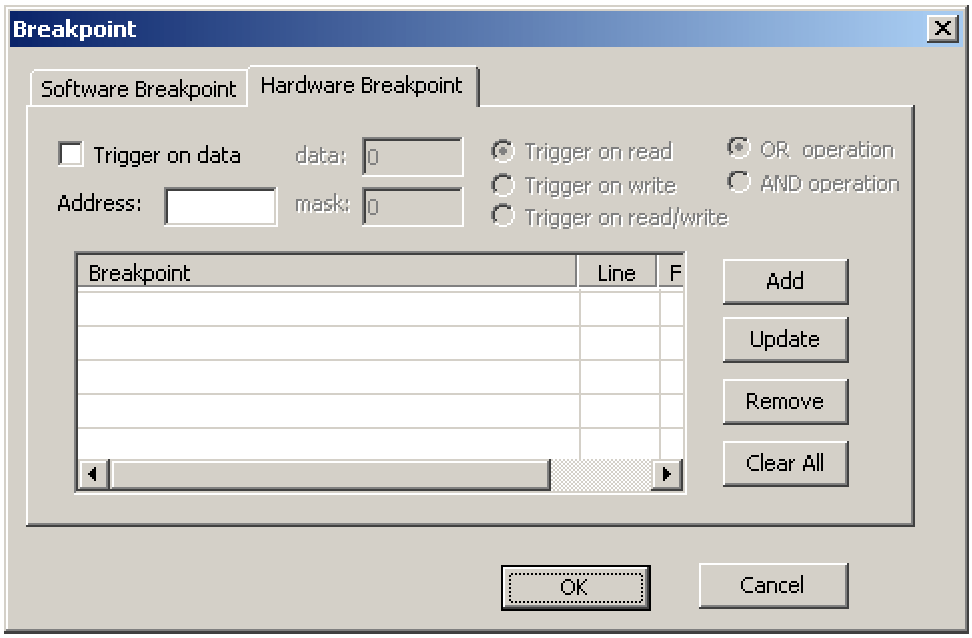
1、Addr：断点地址。
2、[Add]：增加一个软件断点。
3、[Remove]：删除一个断点。
4、[Clear All]：删除所有的断点。

用户还可以通过其他方法设置软件断点：

1、在源代码窗口或 Disassembly 窗口内，将鼠标置于需要设定软件断点的命令行内，按[F9]。
2、在源代码窗口或 Disassembly 窗口内，将鼠标置于需要设定软件断点的命令行内，再单击工具栏图标  。

Hardware Breakpoint 窗口

单击[Edit] → [Hardware Breakpoint]。



1、硬件断点包含指令中断和条件中断。
2、硬件断点的个数：2。
3、ICE2.0 和 ICE3. 0 均支持硬件断点。

用法：

1. Trigger on data：选中即启动条件中断，否则，启动指令中断。
2. Address：断点地址。
3. data：设置符合条件的值。
4. mask：用于屏蔽断点地址的数据的某些位；
5. Trigger on read/Trigger on write/Trigger on read-write：当对断点地址的数据进行只读/只写/读写操作时触发中断。
6. OR operation/AND operation：当已经设定好两个条件断点后，如选用 OR operation，程序的执行过程中有一个条件符合便停下来；如选用 AND operation，程序仅在两个条

件都符合时才停下来。

7. [Add]：增加一个硬件断点。
8. [Update]：更新对断点的设定。
9. [Remove]：删除一个断点。
10. [Clear All]：删除所有断点。

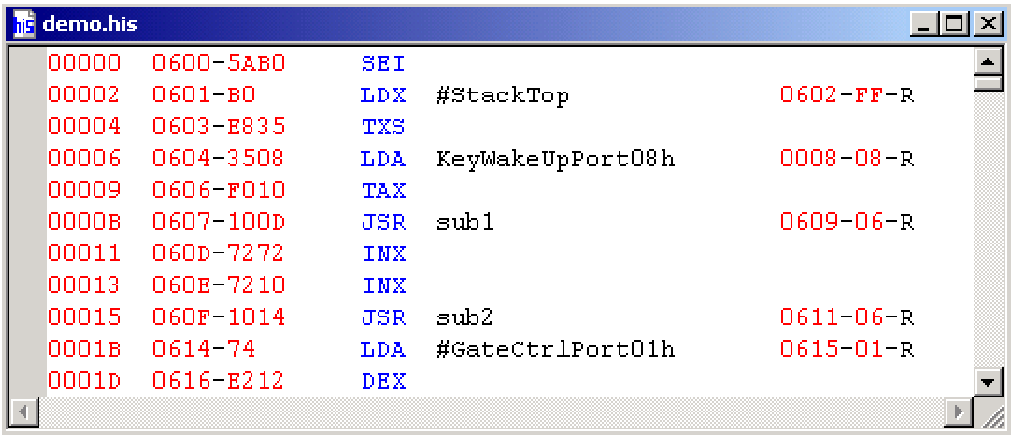
用户还可以通过其他方法设置硬件断点：

1、在源代码窗口或 Disassembly 窗口内，将鼠标置于需要设定硬件断点的命令行内，按[F6]。

2、在源代码窗口或 Disassembly 窗口内，将鼠标置于需要设定硬件断点的命令行内，单击  。

Trace Buffer 窗口

单击 Project 出现 Setting 窗口，在 Setting 窗口的 Device 页内选中 PC Trace Enable，能够启动 Trace Buffer，它记录已执行过的指令、状态和存储区的内容。



用户在调试工具栏内单击  后，相应的跟踪记录会在 Trace Buffer 窗口显示出。

用户在调试工具栏内单击  ，可以删除跟踪记录，同时，Trace buffer 窗口被关闭。

注意：删除跟踪记录功能仅在 ICE3. 0 上支持。

22.2.11 Toolbar


显示/隐藏 **Workspace** 窗口。


显示/隐藏 **Output** 窗口。


管理所有已经打开的文件。


在多个文件内查找文本。


加载目标文件。


编辑源文件。


编译工程。


停止编译。


从当前光标所在位置开始全速运行程序，直到运行结束或遇到断点才停止运行。


将程序下载到 **ICE**/仿真板。


插入/删除软件断点。


删除所有的软件断点。


插入/删除硬件断点。


删除所有的硬件断点。


中断程序的运行。


停止调试。


程序复位。


单步运行程序，如其后的一条指令为子程序调用，会进入子程序继续单步执行。


单步运行程序，如其后的一条指令子程序调用，会执行完整个子程序，然后指向子程序调用的下一条指令。


单步运行程序，如果该指令在一个子程序中，会将该子程序执行完毕，返回子程序调用的下一条指令。


运行程序到当前光标所在行。


显示/隐藏 **Watch** 窗口。


显示/隐藏 **Register** 窗口。


显示/隐藏 **Memory** 窗口。


显示/隐藏 **Disassembly** 窗口。


显示/隐藏 **Command** 窗口。


显示跟踪缓存。


删除跟踪缓存。

22.3 在线仿真器

ECMC653A 是一个功能强大的仿真芯片，它集成了 SPMC65X 系列芯片的所有功能以及在线仿真电路（ICE）。以 ECMC653A 为核心的 SPMC65X 系列仿真器用于仿真 SPMC65X 系列的所有芯片，它将该芯片的 IO 口用排线引出，用户可以很方便的进行各种硬件系统开发。如图 22-1 所示。同时，仿真板还提供了在线编程器（烧录器）ICP，用户可以直接在线编程、调试、仿真以及直接烧录芯片。SPMC65X 系列仿真板还配有一个功能强大的软件开发环境 FortisIDE™，用于和仿真板配合使用。

图 22-1 SPMC65X 系列仿真器

22.3.1 仿真板的硬件结构

SPMC65X 系列仿真板的硬件框图如图 22-2。该仿真板由仿真芯片 ECMC653A、电源、ICE Probe 连接器、振荡电路、在线编程电路和 IO 扩展槽等组成。

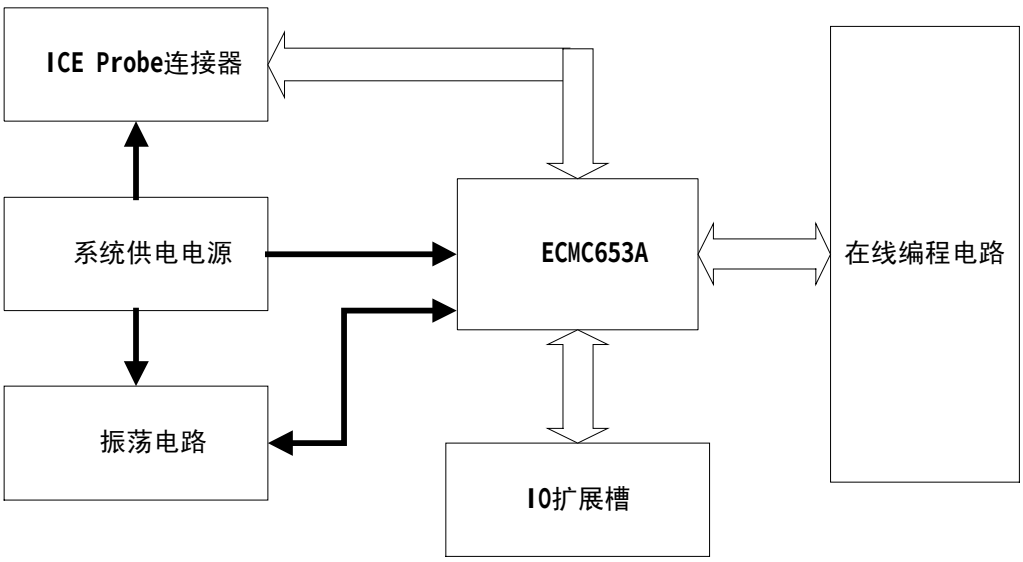


图 22-2 SPMC65X 系列仿真板的硬件框图

(1) ECMC653A

在仿真板上提供了 PLCC68 和 LQFP80 封装。Sunplus 提供了多种仿真板，其仿真芯片有 PLCC68 封装的，也有 LQFP80 封装的，用户必须根据成本、方便性等因素选用恰当的仿真板。

(2) 电源

本系统为 ECMC653A、ICEProbe 和扩展连接器提供了两种电压（5V 和 3.3V）的电源。默认的工作电源为 5V 直流电源（由输入的 9V 电源稳压得到）；同时，仿真板也可以给用户的目标板提供电源电压或者从用户的目标板获取电源电压。

图 22-3 9V 电源输入和 USB 接口

(3) ICE Probe 连接器

共有 3 种 ICEProbe：USB Probe、并口 Probe 和简易 Probe。SPMC65X 系列仿真板支持简易 Probe，它使用通用的 USB 接口将仿真板和电脑连接。简易 Probe 是 Sunplus 新近开发的一种 Probe，具有成本低和应用方便等优点。

(4) 振荡电路

ECMC653A 支持 3 种系统时钟：晶体或陶瓷振荡器、RC 振荡器和外部时钟源。 SPMC65X 系列仿真板支持晶体或者 RC 振荡器做为时钟源，默认情况下使用 16Mhz 的晶体振荡器。用户能任选一种振荡器用于不同的场合。

(5)在线编程电路

凌阳科技在 ECMC653A 上集成了实时串行编程接口，它同外部电路配合，将由 FortisIDE™ 产生的二进制代码写入空白的 SPMC65X 系列微控制器。 另外，Sunplus 开发的应用程序 Q-Write 能够用于对 ICP 进行操作。 Q-Writer 是一个带有友好和便利的用户界面的烧录程序，功能同 Sunplus OTP_MTP_Writer 相近。 用户即可以独立地启动该工具，也能通过 FortisIDE™ 提供的菜单或按钮来调用该工具。但用户需注意，在同一时刻，仅能对 FortisIDE™/Q-Writer 这二者之一进行操作。

(6) IO 扩展槽

SPMC65X 系列仿真板提供了一个 IO 扩展槽，可以通过配套的连接板将 IO 端口引出，连接

到用户的硬件电路板。SPMC65X 系列芯片有不同的 IO 端口，该 IO 扩展槽兼容全部的 IO 端口。

图 22-4 IO 扩展槽和电源开关

22.3.2 操作说明

为让用户对 SPMC65X 系列仿真板有一个全面的了解且能正确地应用，本章节向用户说明如何设定硬件的属性。

n 电源

SPMC65X 系列仿真器支持 2 种电源： 9V 直流电源和通过用户的目标板输入的电源。

图 22-5 DC-9V 电源输入

图 22-5 中右侧的开关用于选择电源。当开关“O”端按下，SPMC65X 系列仿真器的电源和用户目标板的电源独立，二者地线相连。当开关“I”端按下，仿真器的电源、地线和目标板的电源、地线相连，仿真器和目标板可以互相供电，当仿真器由目标板供电的时候，目标板提供的电源电压超过的 6V 后，仿真器便会启动过压保护电路，防止毁坏内部芯片。

n 振荡电路

为保证 ECMC653A 的正常运行，SPMC65X 系列仿真板向其提供了 2 种时钟源：晶体/陶瓷振荡器或者 RC 振荡器做为时钟源。默认为晶体振荡器。

图 22-6 晶体振荡器

如果用户想选择 RC 振荡器，必须首先打开仿真器的铁盒，然后按照下面的步骤进行设置。选择合适的电阻电容，如下图所示，分别焊接在 R37 和 C36 的位置。
按照下图改变拨码开关的位置。

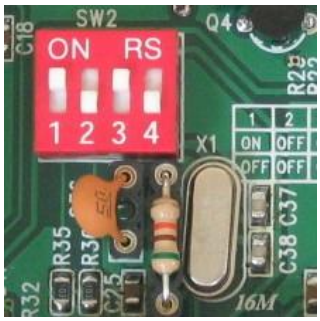


图 22-7 RC 振荡器

n ICE Probe

SPMC65X 系列仿真器提供简易 Probe 用于程序的在线调试，接上 Probe 和电源后，便可以在 FortisIDE™
环境下运行程序。Probe 和电脑通过 USB 线进行连接。

n IO 扩展槽

仿真器上有一个 IO 扩展槽，用于连接 IO 转接板，IO 转接板的输出管脚完全按照芯片实际管脚的排列，可以直接嵌入用户的目标板，进行在线仿真，不同封装的芯片有不同的转接板。用户可以通过转接板向仿真器提供电源电压。

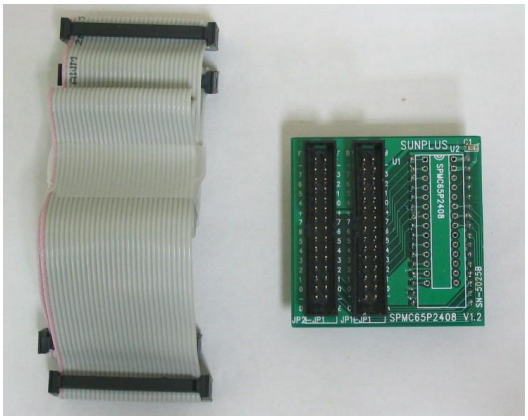


图 22-8 IO 转接板

22.3.3 实时串行编程

Sunplus 向用户提供一个方便和易用的面向 SPMC65X 系列的实时编程工具 Q-Writer。用户在 FortisIDE™ 之上编写好程序后，利用 Q-Writer 将工程文件的代码写入所选的 SPMC65X 微控制器内，且能在用户的硬件电路板上对代码的执行结果进行验证。它能够提高程序开发的方便性和可靠性。具体的功能在后面的章节内有详细的介绍。

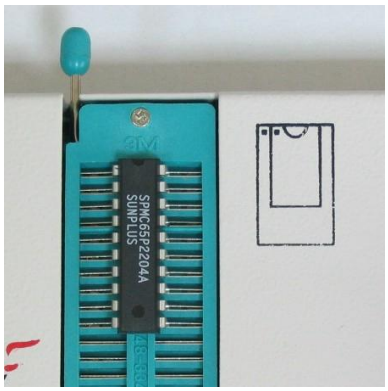


图 22-9 烧录器上芯片的摆放

22.4 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题	应用例系列号
本章无相关应用例	

库函数

标题	应用例系列号
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

22.5 Revision History

版本号	日期	备注
V0.1	03/31/2005	第一版

23 在线烧录器

23.1 简介

本章将为您介绍三种烧录工具及其使用方法。Sunplus 公司一直以来，致力于研发可靠易用的烧录工具。目前，Sunplus 公司已研发出两款烧录工具，一款是最新研发的 Q-Writer，它是配合 SPMC65X 系列的 EMU 板进行芯片烧录的工具；另一款是 Sunplus 公司早期研发的 OTP/MTP Writer。此外，Sunplus 公司委托第三方厂商研发了功能强大的 gang-writer，其专门为产品量产而设计。本章还将为您介绍利用 Sunplus 公司的烧录工具如何进行产品序列号、产品信息以及芯片加密功能的设定。

23.2 Q-Writer

23.2.1 简介

Q-Writer 是 Sunplus 公司最新研发的专门用于烧录 SPMC65X 系列 EMU 的一款烧录工具，其软件部分集成在 FortisIDE 之中。用户可以在 SPMC65X 系列开发环境中完成所有的设计流程。Q-Writer 具有友好的操作界面和简捷的硬件结构，而其可靠易用，价格低廉等特点更为用户带来诸多便利。

23.2.2 硬件集成设计

Q-Writer 的硬件部分附属在 SPMC65X 系列 EMU 板上，它利用 ICE 接口和外部电源来实现芯片的串行烧录。其中，ECMC653A 控制着串行烧录信号和外部电源的开启。为进入 EPM 模式，必须同时提供 VDD 和 Vpp 两种外部电源。

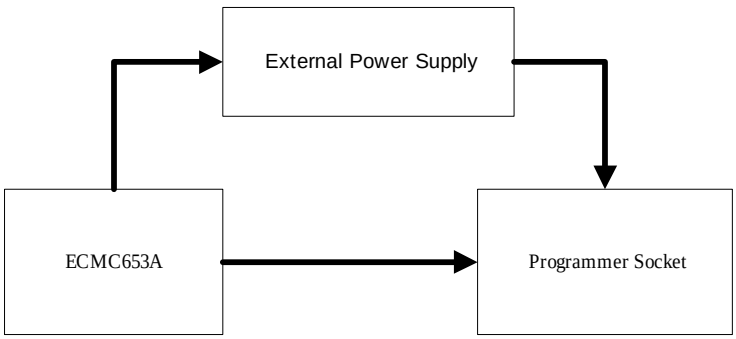


图 23-1 ICP 框图

通常情况下，OTP 芯片处于 MCU 模式；而当 VDD 为 6V，Vpp 为 13V 时，OTP 芯片会进入 EPM 模式，进行芯片烧录。

23.2.3 安装和启动

FortisIDE 做为新一代的芯片开发工具，包括两部分，一个用于程序的开发，另一个用于芯片管理。如下图所示：

凌阳科技提供了一个软件安装包，同时涵盖了 FortisIDE, FortisIDE Body, 和 Q-Writer，用户可以将其一并安装。当然，对这三个软件也有独立的安装包，便于以后升级。

新版 FortisIDE 已将 Q-Writer 的软件部分整合到其中，因此，从 V1.6.5 版起，FortisIDE 将开始支持 Q-Writer 的功能。Q-Writer 可以运行于 Windows95® 、Windows98®、Windows 2000®、Windows NT®[1] 和 Windows XP®操作系统下。

安装

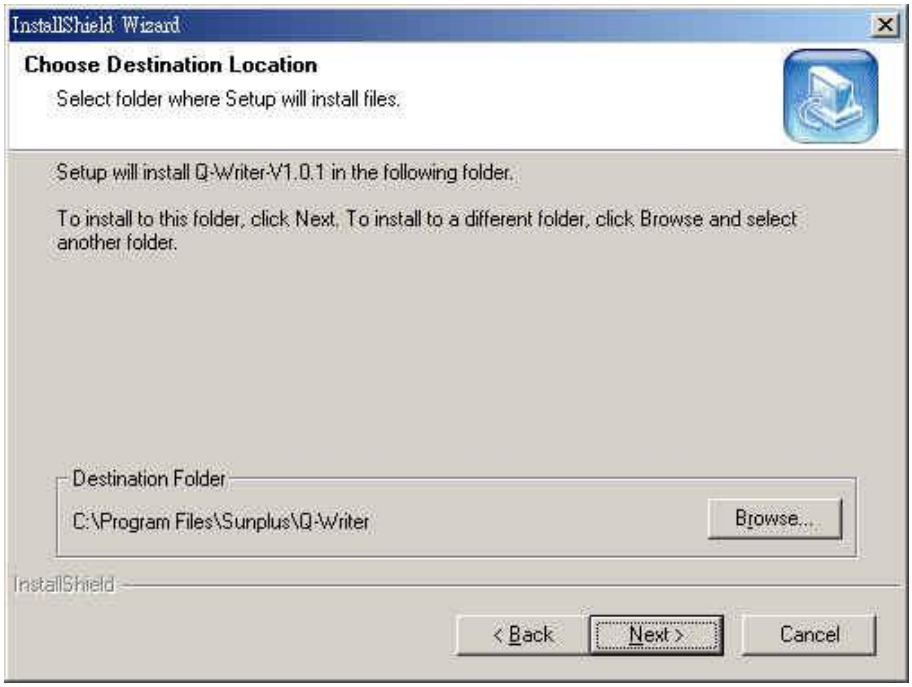
按照下列步骤进行 Q-Writer 的安装

1. 首先运行 Q-Writer V1.0.1.exe 文件，安装界面显示如下，然后点击“Next”。

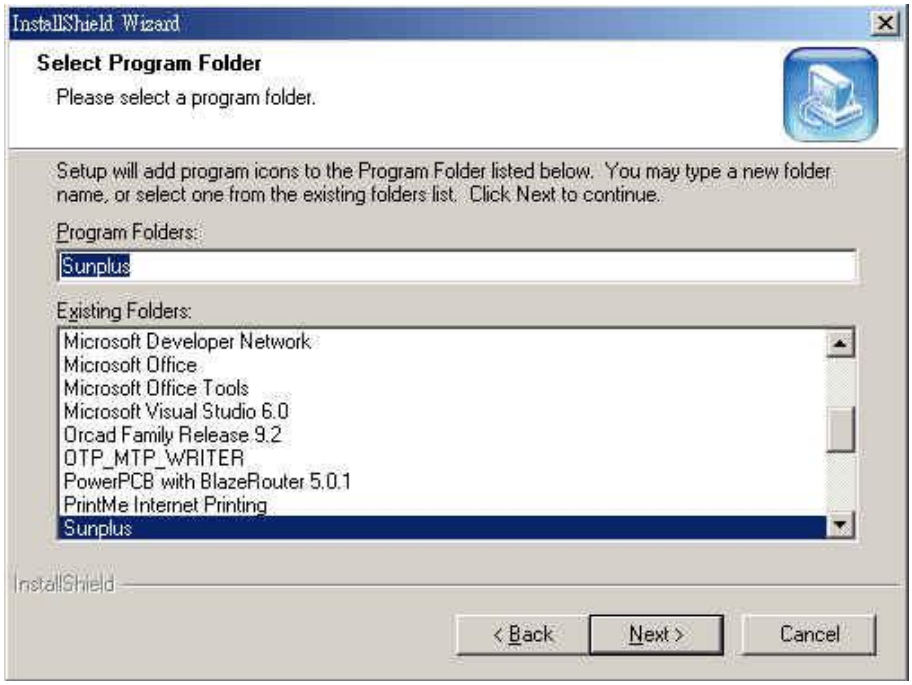
注意：V1.0.1 为软件的版本号。

2. 根据屏幕上的提示进行操作。

3. 选择程序的安装路径和安装文件夹（强烈推荐用户使用默认选项，单击“Next”即可）。



4. 点击“Next”，开始安装。



5. 选择[Start Menu] → [Program] → [Sunplus] → [Q-Writer] → [Q-Writer-V1.0.1.exe]，启动 Q-Writer 应用程序。

开始

首先选择合适的 probe 将 SPMC65X 系列的仿真板与电脑连接好，之后再启动 Q-Writer。您可以选择以下两种方式启动 Q-Writer。

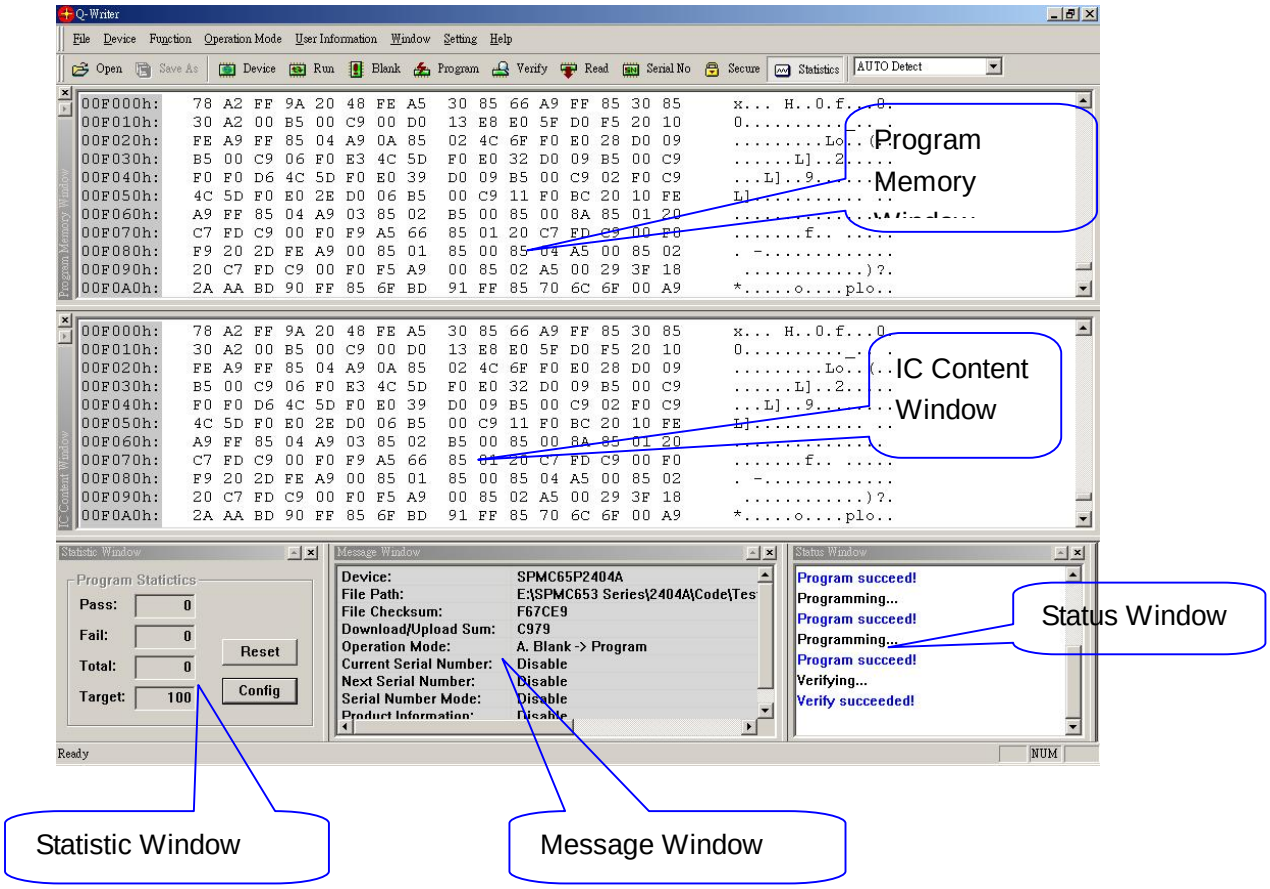
1. 首先启动 FortisIDE，之后点击工具栏中的  图标，或者选择[Tools]→[Q-Writer]来启动 Q-Writer。
2. 直接运行 Q-Writer-V1.0.1.exe 文件。

Note!

在 FortisIDE 中启动 Q-Writer 后，便不能进入 debug 模式。即使在 debug 模式下，一旦启动了 Q-Writer，FortisIDE 也会退出 debug 模式，回到正常编辑状态。

23.2.4 如何使用 Q-Writer

下图所示的 Q-Writer 界面由以下部分组成：menu，toolbar， program memory window，IC content window，statistic window，message window 和 status window 等。



23.2.4.3 Menu

Q-Writer 的菜单包括 File，Device，Function，Operation Mode，User Information，Window，

Setting 和 Help。下面我们分别来描述它们的功能和用法。

– File

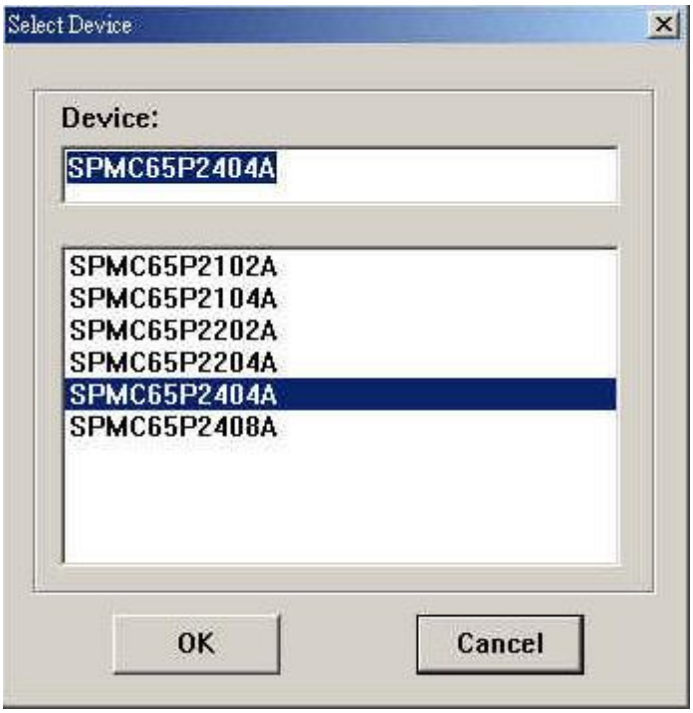
Ø Open: 打开*.bin 或者 *.tsk 类型的文件并将其下载到“program memory window”窗口。

Ø Recent Files: 记录最新打开的 4 个文件。

Ø Exit: 退出 Q-Writer。关闭 Q-Writer 后，FortisIDE 便可以进入 debug 模式。

– Device

Ø Select: 选择所需要烧录的芯片类型。具体方法：点击 Device-> Select，出现下图所示窗口“Select Device”，选择所要烧录的芯片类型即可。安装了 Q-Writer 后，首次运行时，必须选择芯片类型。当下次启动 Q-Writer 后，如果烧录的芯片和上次相同，可以不用进行选择，系统会默认为上一次选择的芯片类型。



– Function

Q-Writer 可以支持如下六种操作。在往芯片中烧录程序前，请先在该环境下打开你所需要烧录的*.bin 或者 *.tsk 文件，否则，将不能进行芯片烧录操作。

Ø Auto Run: 此命令会顺序执行连续的烧录动作。为了让用户更方便的烧录芯片，Q-Writer 烧录共有三种烧录模式（后面详述），用户点击 Auto Run 便可以进行直接烧录。首次运行 Q-Writer，默认为第一种烧录模式。

Ø Blank Check: 此命令是检查烧录器上的芯片是否为空白芯片。同时，Q-Writer 会读出芯片中的内容，显示在“IC content window”窗口。如果检查到该芯片不是空白的，系统会在“IC content window”窗口中自动显示非空白区域，并将其内容用红色高亮标记出来。

Ø Program: 此命令是将“Program Memory Window”窗口的内容烧录到 OTP 芯片上，同时，将芯片中的内容读出，显示在“IC Content Window”窗口中。如果烧录失败，系统会跳到“IC content window”窗口中出错的位置，将其内容用红色高亮显示。烧录成功之后，

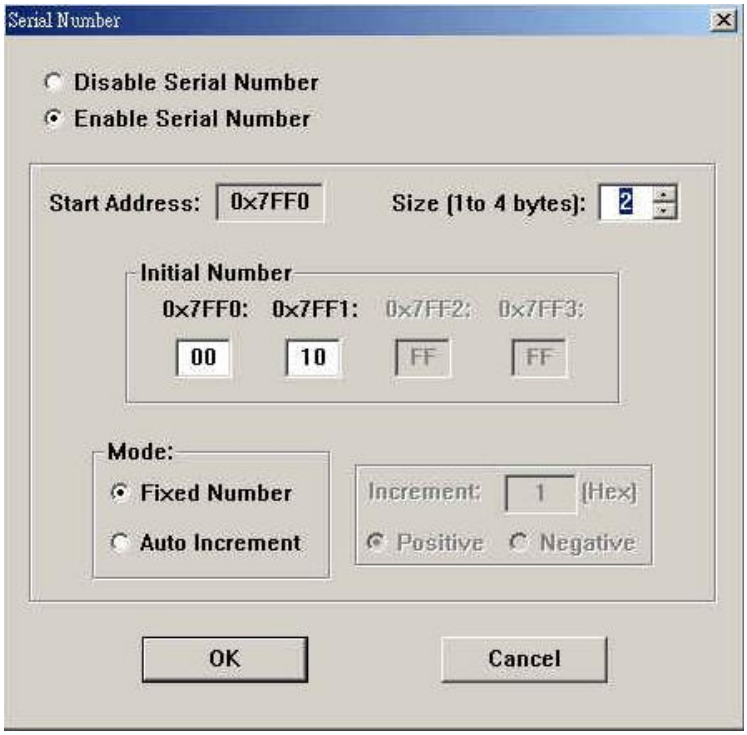
还需执行 Verify 功能确保烧录的可靠性。

Ø Verify: 此命令用来验证烧录到 OTP 芯片中的内容是否与源文件中的内容一致。启动 Verify 功能，Q-Writer 会将芯片中的内容读出，显示在“IC content window”窗口，然后和“Program Memory Window”窗口中的内容（源文件）进行比较，发现有不同之处，系统会提示出错，并跳到“IC content window”窗口中出错的位置，将其内容用红色高亮显示。
Ø Read: 将芯片中内容读出，显示在“IC content window”窗口。

Ø Secure: 此命令用来设定芯片的加密功能。一旦加密，芯片中的大部分内容将禁止读出。因此，在对芯片进行加密前，用户必须完成 program 和 verify 的操作。加密后，SPMC65X 系列的微控器仅允许读出芯片中的部分内容，包括芯片设置选项，用户信息以及芯片存储器中最后 16 个字节($FFF0~$FFFF)中的信息。其它地址的内容均显示为$00。
在对芯片加密前，Q-Writer 必须执行 Blank Check 功能，以提高芯片烧录的可靠性，避免错误的发生。若 OTP 芯片为空，系统将不能对其进行加密操作。

– Operation Mode
芯片的三种烧录模式
为了用户烧录方便，Q-Writer 提供了三种烧录模式，可以进行连续的烧录动作，如下 A、B、C 三种模式所示。例如，倘若选择 C 模式，用户在启动 Blank check 功能后，系统会自动对芯片进行 Blank check、Program、 Verify 和 Secure 的连续操作，完成全部烧录过程。如果在某一个环节发生错误，烧录动作都会立即停止，提示错误信息。
Ø A. Blank à Program
Ø B. Blank à Program à Verify
Ø C. Blank à Program à Verify à Secure

– User Information
此菜单用来设定产品信息。Q-Writer 提供了两项功能来在设置产品信息。
Ø Serial Number
Q-Writer 指定了 4 个字节（地址为$7FF0～$7FF3）作为序列号码的写入空间。



v Start Address: $7FF0 为序列号码的起始地址。其中$7FF0 为四个地址中的最低地址，$7FF3 为最高地址。

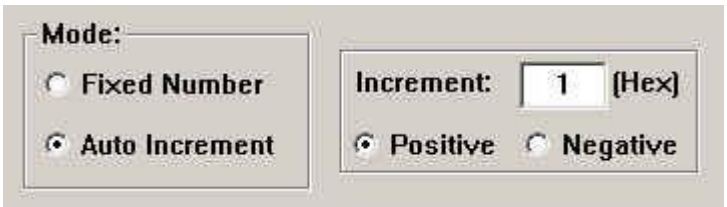
v Size (1 ～ 4 bytes):用户可以设定写入序列号码的字节数。

v Initial Number:此项用来写入芯片的初始序列号。序列号写入芯片后，“Program Memory Window”窗口中相同地址的内容也将随之更新。其中，未用到的字节其默认值为$FF。比如，用户上一次写入 3 个字节的序列号，之后，用户将其改为 2 个字节，那么，未用到的字节将被置为$FF。

Previous â 007FF0h: 00 10 20 FF FF FF FF FF

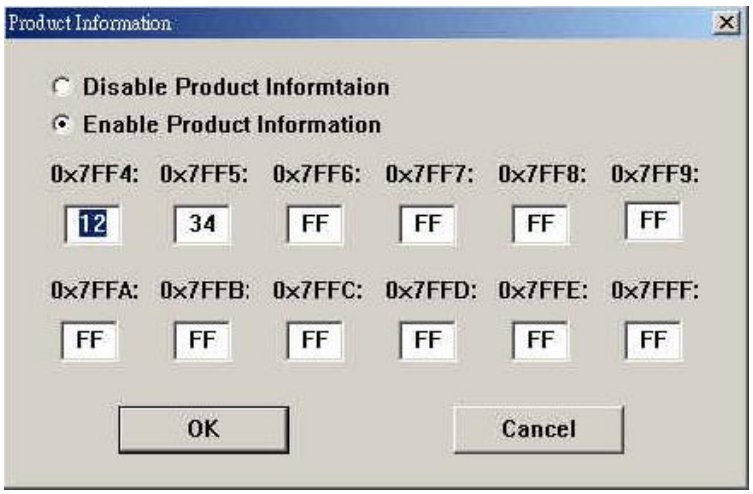
Next â 007FF0h: 00 10 FF FF FF FF FF FF

v Mode:此项用来设置序列号码的生成模式。系统为用户提供了两种模式：固定模式和自动模式。固定模式下，*increment* 项无效，在每一个芯片进行烧录时，烧入的序列号码均为初始设置值。自动模式下，用户可以设定序列号码的递增或递减变化方式，以及设定相应变化的增量值或减量值。设置完毕后，烧入芯片的序列号码将从初始设置值开始递增或递减。



Ø Product Info:

用户可以在$7FF4~$7FFF 这 12 个字节写入任意的产品信息，比如生产日期和生产厂商等。这些字节的默认值为$FF。



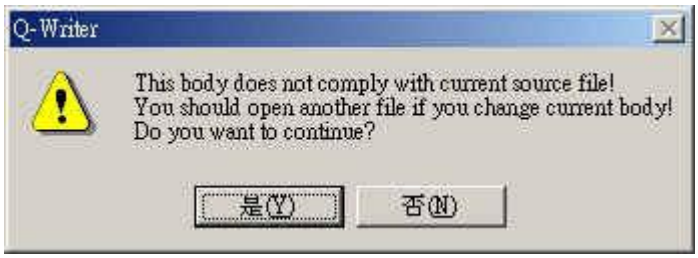
– Window

在菜单中点击 Window，会出现 5 个选项，分别对应于 5 个操作窗口。用户可以在 window 菜单中勾选任意选项来打开或关闭相应的窗口。下面分别介绍。

Ø Program Memory Window: 该窗口显示下载到 Q-Writer 中的二进制文件即源程序。

Note!

如果用户从 FortisIDE 中启动 Q-Writer，系统便会自动下载当前文件的二进制代码到 program memory window 窗口，这样使用起来更方便。但是注意：如果用户改变了芯片类型而没有重新下载二进制代码，系统会出现如下警告提示用户。如果 FortisIDE 中选择的芯片和 Q-Writer 中的不同，program memory window 窗口中的内容没有更新，也会出现如下警告。



右击鼠标，弹出如下菜单，点击便可以使用这些功能：

C *Goto Address*: 点击 *Goto* Address，在出现的对话框中输入一个地址，便可以跳转到相应的位置（可以在$0000 ~$FFFF 范围内任意跳转）。

Ø IC Content Window: 该窗口显示从 OTP 芯片的存储器中读出的内容。在操作过程中如有错误发生，该窗口将显示出错地址的内容。

Ø Statistic Window: 该窗口用来统计烧录芯片的数量。

I. Editbox

z *Pass*:烧录成功的芯片数量。

z *Fail*:烧录失败的芯片数量。

z *Total*:.烧录成功和失败的芯片总数。

z *Target*:设置预期烧录芯片的数目。当 total 中的总数达到 target 中的设定值时，系统会弹出一个信息提示框询问用户是否清除此设定值。

II. Push button

z Reset:清除“target number.”以外的所有统计数据。按下此按钮后，系统弹出下图所示的信息提示框，要求用户确认是否执行该操作。

z Config:设置预期烧录的程序数目。

Ø Message Window: 在 window 菜单中选中或取消该选项可以打开或关闭该窗口。Message Window 窗口的信息显示如下：

X *Device*:显示所选的芯片名称。

Note!

安装了 Q-Writer 软件之后，用户第一次使用时，该选项为空，需要选择芯片类型。一般而言，下次使用时，该项总是默认为最近一次选择的芯片类型。如果用户直接从 FortisIDE 环境中直接打开 Q-Writer，则 Q-Writer 中默认的芯片类型与 FortisIDE 中的保持一致。

X *File Path*:显示下载文件（*.bin）的路径。

X *File Checksum*:显示加载文件的校验码，其计算范围从$00 到 $FFFF 共 64k 字节。File checksum 和 FortisIDE 中计算的校验码相同。

X *Download/Upload Sum*:显示“IC content window”窗口内容的校验码，由程序寄存器，应用选项和用户信息组成，其计算范围从$00 到 $FFFF 共 64k 字节。

Note!

编程者在程序中经常使用‘DB’, ‘DS’,和‘DW’来定义变量空间，如果使用‘DB’和‘DW’ 来定义，编译器会在 RAM 区将变量指定为$01 或$00，如果使用‘DS’，则指定为$FF。为了保持 file checksum 的值，强烈建议用户使用“.DS”来定义变量

X *Operation Mode*:显示芯片的烧录模式。

X *Current Serial Number*:以十六进制的格式显示当前序列号。如果用户没有启动此项功能，该选项将显示“Disable”。

X *Next Serial Number*:以十六进制的格式显示下一个序列号。如果用户没有启动此项功能，该选项将显示“Disable”。

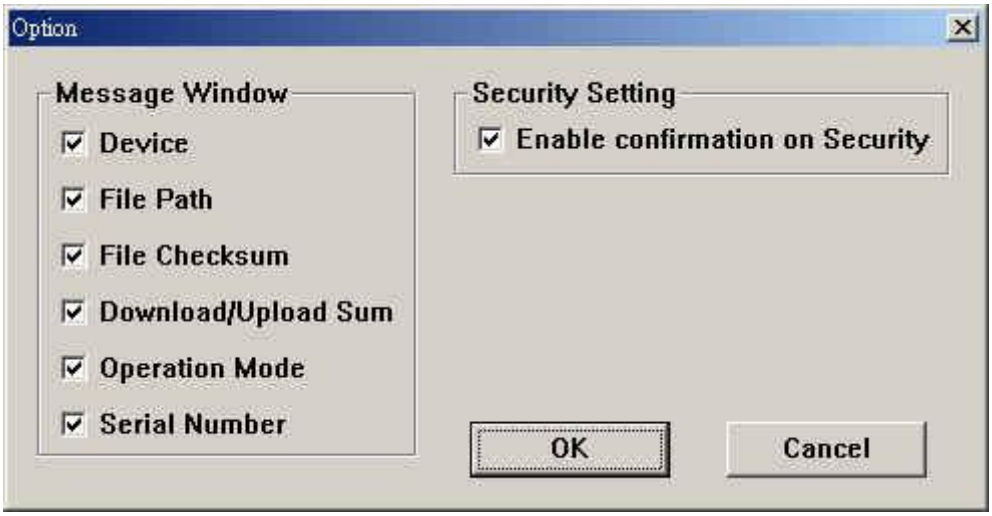
X *Serial Number Mode*:.显示序列号的变化方式。如果用户没有启动此项功能，该选项将显示“Disable”。

X *Product Information*:显示产品信息的状态：“enable”或“disable”。

Ø此窗口显示了 Q-Writer 的所有操作结果。用户可以清晰的看到整个烧录过程。

– Setting

Ø选择该菜单，系统会弹出如下图所示的信息管理窗口。如果用户未选中”Security Setting”选项，系统在为芯片加密时，将不会再显示加密确认的窗口。



Ø Program Statistic: 此功能用来.激活/取消统计功能，与工具栏中  按钮的作用相同。

Ø 16 Bytes per Row: 此功能用来控制每一行显示的字节数。选中该功能后，“Program Memory Window”窗口内中的每一行都将显示 16 个字节的数据。

Ø Show ASCII Code: 此功能用来显示“Program Memory Window”窗口内二进制数据的 ASCII 码。

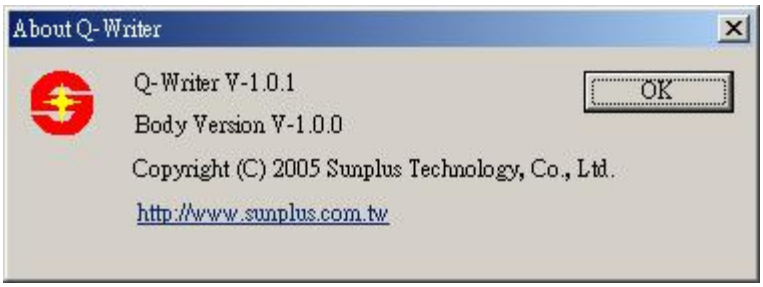
```
00F010h:   01 A9 FF 85 30 85 30 20  84 FD C9 00 F0 F9 A9 00     ....0.0 ........
00F020h:   85 01 A5 00 85 02 20 84  FD C9 00 F0 F5 A9 00 85     ...... ........
00F030h:   02 A5 00 29 3F 18 2A AA  BD 90 FF 85 76 BD 91 FF     ...)?.*.....v...
00F040h:   85 77 6C 76 00 A9 00 AA  8A A8 98 48 68 85 01 20     .wlv.......Hh..
00F050h:   84 FD C9 00 F0 F9 A9 55  AA 8A A8 98 48 68 85 01     .......U....Hh..
00F060h:   20 84 FD C9 00 F0 F9 A9  AA AA 8A A8 98 48 68 85      ............Hh.
00F070h:   01 20 84 FD C9 00 F0 F9  A9 FF AA 8A A8 98 48 68     . ............Hh
00F080h:   85 01 20 84 FD C9 00 F0  F9 38 A9 55 E9 55 D0 44     .. ......8.U.U.D
00F090h:   A9 01 85 01 20 84 FD C9  00 F0 F9 38 A9 55 E9 56     .... ......8.U.V
00F0A0h:   10 32 A9 02 85 01 20 84  FD C9 00 F0 F9 18 A9 FF     .2.... ........
00F0B0h:   69 02 90 20 A9 04 85 01  20 84 FD C9 00 F0 F9 18     i.. .... .......
```

– Help

Ø Help Topics (F1): Q-Writer 为用户提供了在线帮助，可以查找相关主题来获得帮助。

Ø About Q-Writer: 该选项为用户提供 Q-Writer 软件的版本信息及凌阳公司的网址。其中版

本信息包括 Q-Writer 软件的版本号和烧录芯片的版本号。用户可以在凌阳公司的网站上下载到最新版本的软件。



23.2.4.4 Toolbar


打开二进制格式（*.bin 或*.tsk）的源文件。用户必须将需要烧录的源文件下载到 Q-writer 中，以激活其所有功能。


将“program memory window”窗口中的内容保存到另一路径下。


使用 Q-Writer 烧录程序前，用户必须选择烧录芯片的型号。


依据烧录模式的设定执行连续动作


检测芯片是否为空白


将程序烧录到芯片中。


验证烧录程序是否有被正确地烧录进芯片


从芯片的存储器中读取数据


打开序列号码的设置窗口。


芯片加密


激活/取消芯片烧录数量统计功能。

.选择 probe。系统为用户提供了三种方式来选择合适的 probe。

23.2.5 系统信息

r All data will be cleaned to 0, do you want to continue?

所有内容将被清除，是否继续？

Ú 用户按下“Statistic Window”中的“Reset”按钮后，系统将弹出一个信息提示框，要求用户确认是否清除 *Target*.以外的所有内容。

r Can not detect any probe!

未检测到任何 Probe！

系统未检测到与 EMU 板连接的 probe，用户需要检测 probe 是否与 EMU 板连接良好或者 EMU 板是否上电。在使用的过程中，如果 USB probe 被拔下，便会出现如下警告：

凌阳科技公司对 IC 进行了精细的设计，防止在芯片烧录的过程中突然出现各种意外，即使 USB probe 被拔掉，在重新插上后，芯片仍然可以继续烧录。但是，在重新插上 USB probe 后，用户需要在烧录环境下再次选择 probe 类型。

r EMU board is not ready. Please check if EMU board is set or connected properly.

未检测到 EMU 板。请用户检查 EMU 板是否被正确连接。

该信息表明用户选择的 probe 与实际使用的 probe 不一致，或者 EMU 板有问题。如果用户选择“Auto Detect”probe 选项，系统将会自动检测 probe。

r File cannot be opened to save!

当前文件无法保存！

该信息表明用户不能保存当前的文件。可能是用户的电脑或者开启的文件出错。

r IC has been secured already!

芯片已加密！

启动加密功能前，系统会检测芯片是否已加密。如果已经加密，系统会显示提示信息并停止对其进行加密。

r Incorrect file format!

文件格式不正确！

用户仅可以在 Q-Writer 中加载.bin 或.tsk 格式的文件。此项信息表明用户下载的文件格式不正确。

r No body is selected!

没有选择芯片型号

用户首次启动 Q-Writer 后，若未选择芯片型号，系统将会显示警告信息并自动弹出“Select Device”窗口。

r Q-Writer is already opened!

Q-Writer 已经打开！
Q-Writer 仅能同时打开一次。

r The source file of program memory has been modified. Do you want to reload it?
“Program Memory Window”窗口的源文件已经改变，是否要重新加载它？
当下载到“Program Memory Window”窗口的源文件已经改变并保存为另一文件时，系统便会显示此项信息来提醒用户。

r USB Probe detected but is used by other application!
检测到 USB Probe 正在为其它设备所使用！
用户需要检查 USB Probe 是否正在为另一应用程序所使用。

r You are going to add Security on IC, which is a non-reversible action. Are you sure to do this?
您将启动加密功能，该动作不能撤消。您确定要启动吗？
该信息表明，系统即将执行加密功能，用户必须确认芯片是否已准备就绪。此外，您还可以通过选中 / 不选中该窗口中的选项来选择是否每次加密前都要显示此窗口。

r You have reached the target amount! Do you want to reset Statistics Window?
烧录芯片的数目已达到 *Target* 的设定值，您要清除“Statistics Window”窗口中的数据吗？
当 total 中的数目达到 target 中的设定值时，系统会弹出一个信息提示框，询问用户是否清除此设定值。用户可以选择“Yes”清除掉该窗口中的内容，也可以选择“No”，此时系统将忽略此提示信息并保留该窗口中的内容。

r Your IC is blank! Securing is prohibited!
芯片空白，您不能对芯片进行加密！
芯片内容为空，系统不允许对空白芯片进行加密。

r Your Serial Number could have been used again!
r 这个序列号码可能已被使用过！
Ú 序列号码在“Auto Increment”模式下会自动进行递增或递减。当序列号码的值递增或递减产生溢出后，可能会产生与以前相同的序列号，如果用户继续执行烧录动作，那么多个芯片就可能会被烧入相同的序列号码。

23.3 SUNPLUS OTP/MTP Writer

23.3.1 简介

OTP/MTP Writer 是 SUNPLUS 公司研制的一款用于烧录 SUNPLUS MTP (Multi-Time Programmable IC)和 OTP (One-Time Programmable IC)的烧录工具。OTP/MTP Writer 包括两个基本部分：硬件部分和烧录软件。下面我们来分别介绍。

23.3.2 硬件

硬件部分包括四种部件：OTP/MTP 烧录器，COM 口电缆，DC-18V 电源和一个转接板（针

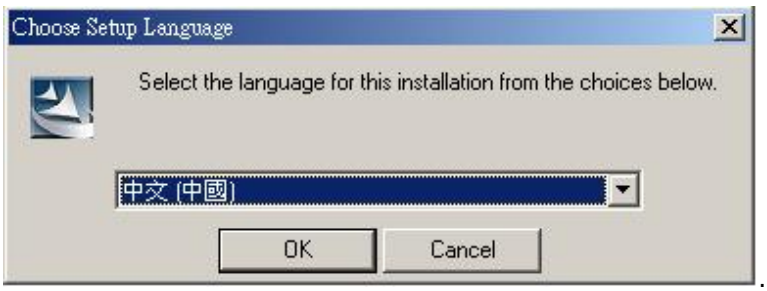
对不同的 IC 而异）。
按照以下步骤来连接硬件部分。
— 选择一个转接板，并插到 OTP/MTP Writer 上。
— 用一根 COM 口电缆将 OTP/MTP Writer 连接到 PC 机。
— 用一个 DC-18V 电源为 OTP/MTP Writer 供电。

23.3.3 转接板

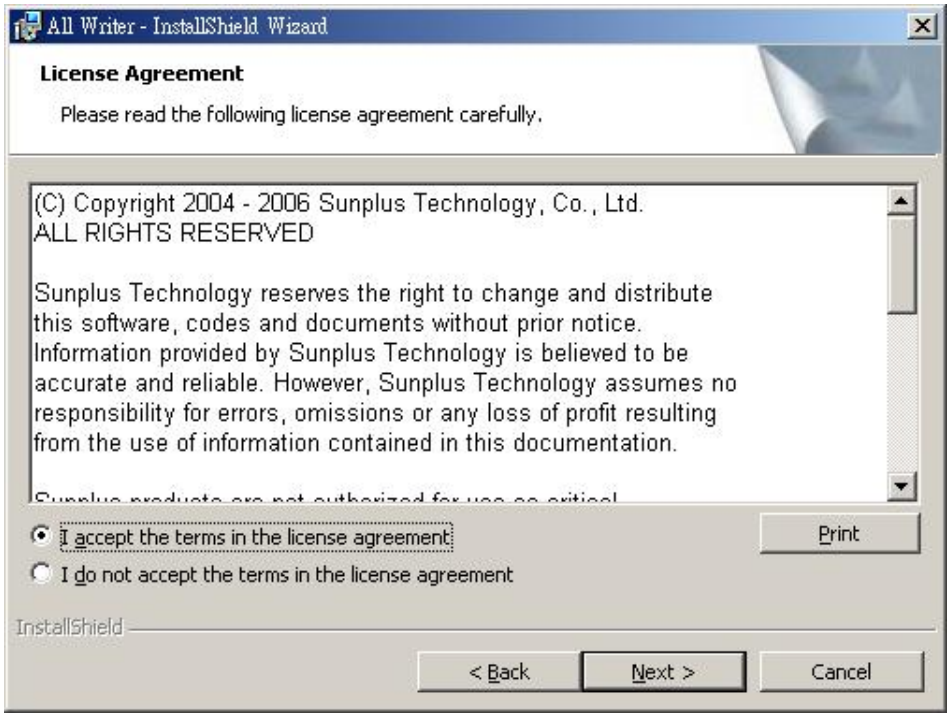
不同的芯片系列，所需的转接板也不相同。由于 Sunplus 公司为 SPMC65X 系列芯片设计了统一的烧录管脚位置，因此，所有 SPMC65X 系列芯片可以使用同一个转接板， 烧录更加方便。

23.3.4 烧录软件

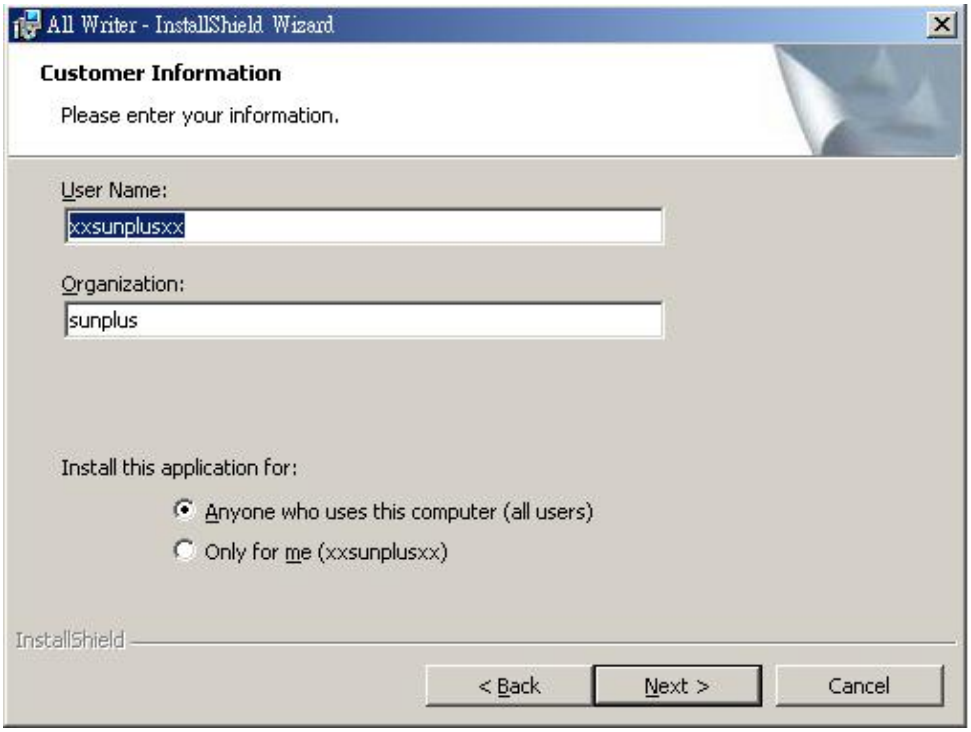
OTP/MTP writer 的烧录软件适用于 SUNPLUS 绝大部分的 OTP/MTP 芯片，可以运行于 Windows95®, Windows98®，Windows 2000®，Windows NT®1 和 Windows XP®操作系统下。
安装和启动
按照以下步骤进行安装：
1、单击 Start " Settings " Control Panel， 然后双击 Add/Remove Programs， 打开 Install/Uninstall 选项，之后在已安装的程序列表中选择 OTP_MTP_WRITER 并单击 Add/Remove 按钮。
2、运行“S+ OTP_MTP ”安装程序，选择语言。



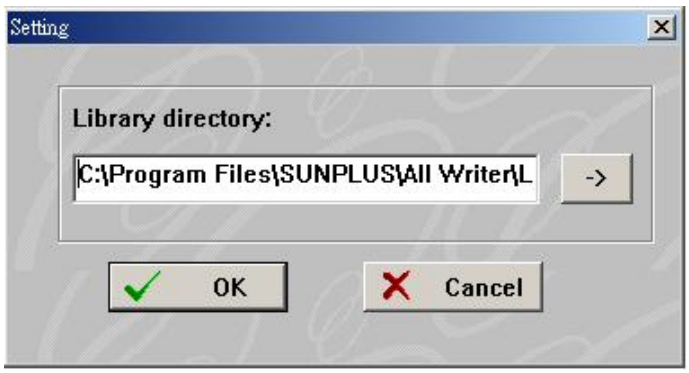
Ø 按照屏幕提示信息安装 Sunplus OTP/MTP Writer 烧录软件。



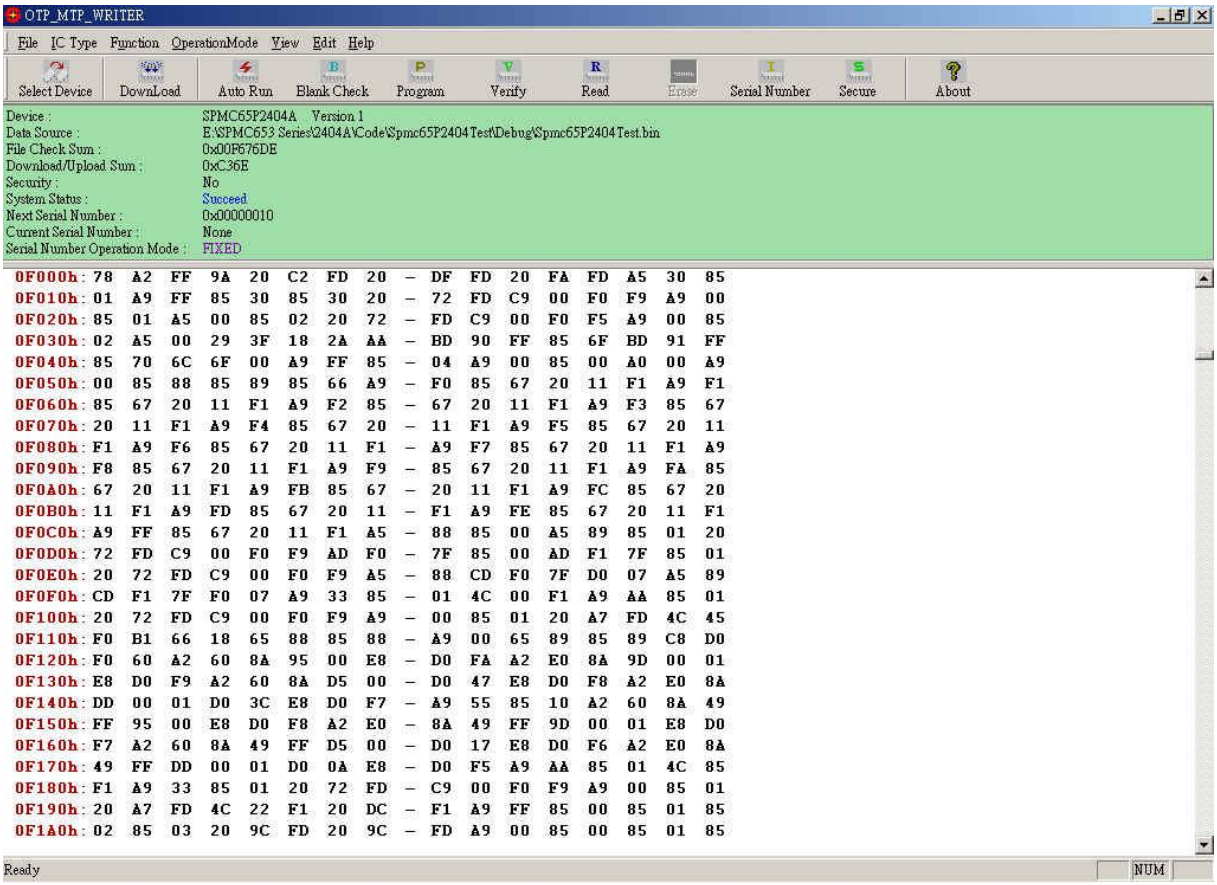
Ø 键入你的姓名和公司名称。



Ø 在 ICType->Setting 的功能列表中检查“OTP_MTP_WRITER.ini”的安装路径，本文件应安装在 C:\Program Files\Sunplus\OTP MTP Writer\Library 路径下。



23.3.5 功能描述



File

- **Download (Ctrl + D):**打开一个指定的烧录程序，将程序下载到烧录器中。
- **Save (Ctrl + S):**将烧录程序保存到电脑中。
- **Save As:**将烧录程序存为另一文件名。
- **Exit:**退出烧录环境。

IC Type

- **Select Device:** 此命令用于选择烧录芯片的类型（使用合适的转接板）并将相应的资源文件(*.das)下载到烧录器中。资源文件(*.das)用于执行烧录功能，随着转接板的不同而变化。如果 device 选择正确，message window 窗口中会显

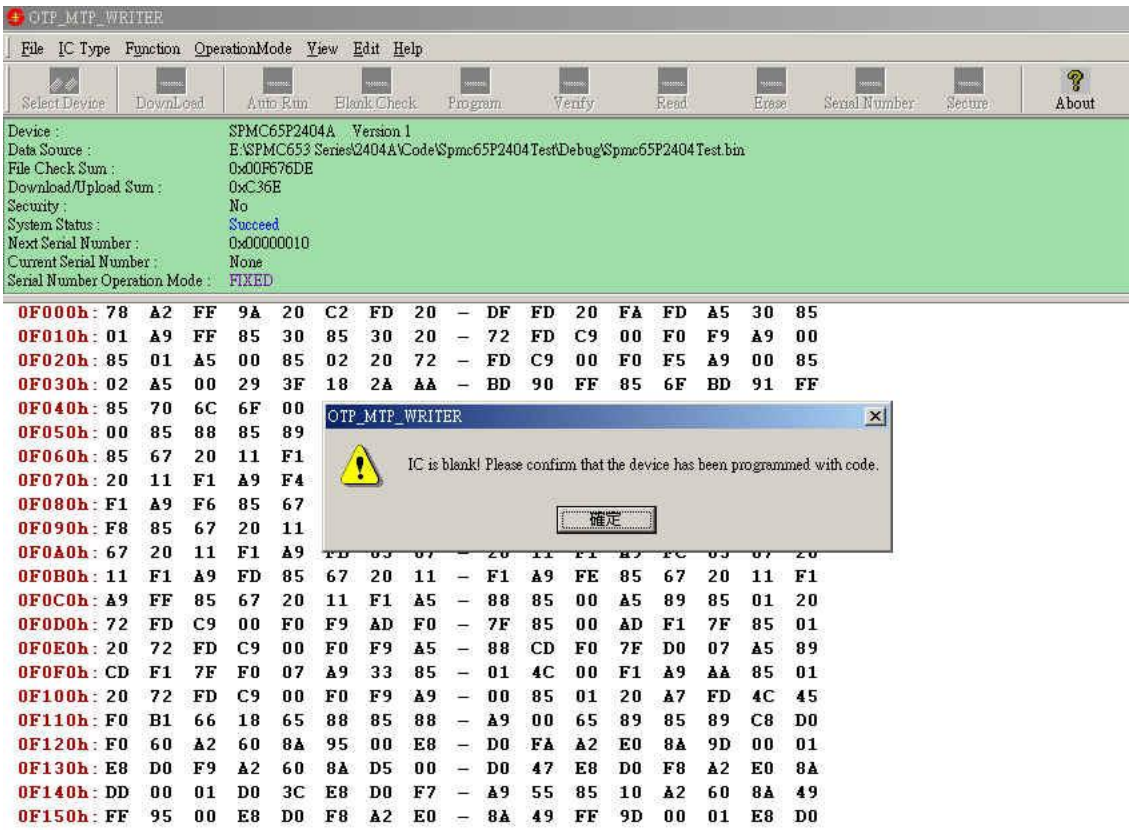
示出相应的芯片类型；如果 device 选择错误，message window 中会显示 "Not Selected" 信息，OTP/MTP Writer 便无法进行芯片烧录。

Setting: 设置资源文件路径：默认路径是 “C:\Program Files\Sunplus\OTP_MTP_Writer\library "。用户在安装了 Sunplus OTP/MTP Writer AP 并首次启动后，系统可能会要求用户指定其库文件的安装路径。请根据安装软件所在的位置为其指定正确的路径。

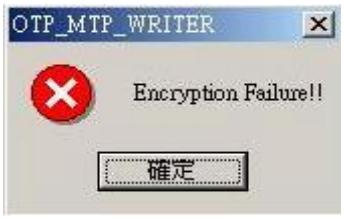
Function

- **Auto Run (F5):** 依据烧录模式的设定执行连续的动作，将“memory window”窗口中的内容烧录到 OTP/MTP 芯片中。
- **Blank Check:** 检查芯片是否为空白。
- **Program:** 将“memory window”窗口中的内容烧录到 OTP/MTP 芯片中。
- **Verify:** 验证烧录程序是否有被正确地烧录进芯片。
- **Read:** 从芯片的存储器中读取数据并在“Program Memory Window”窗口显示。
- **Erase:** 擦除 MTP 芯片中的内容，将其变为空白片。
- **Serial Number:** 用户可以为量产的产品指定序列号码。
- **Security:** 此命令用来设定或取消芯片的加密功能。芯片一旦被加密，用户将不能再读出其中的内容。因此，在对芯片进行加密前，用户必须完成 program 和 verify 的操作。下面将详细描述此项操作。

1. 在执行加密操作之前，用户必须完成对芯片的烧录。否则，若在芯片为空白的情况下执行加密操作，系统会显示一个错误信息，禁止对空白芯片进行加密。



2. 如果已经完成对芯片的烧录动作，将继续执行加密操作，加密完成后，会弹出相应的信息窗口来提示用户是否加密成功。



Write Mode

- **Blank Check->Program:** 首先检测芯片是否为空白，如果为空白，执行烧录操作，否则，弹出检测失败的提示信息。
- **Blank Check->Program->Verify:** 首先检测芯片是否为空白，如果为空白，执行烧录操作，烧录完毕，验证烧录是否成功。期间，如有一个环节发生错误，均会停止当前动作，弹出错误提示信息。
- **Blank Check->Program->Verify->Secure:** 首先检测芯片是否为空白，如果为空白，执行烧录操作，烧录完毕，验证烧录是否成功。若烧录成功，继续对芯片进行加密。期间，若有一个环节发生错误，均会停止当前动作，弹出错误提示信息。此项操作简便易用，可以完成芯片烧录的全部功能。而且，还可以清楚的看到每一步的操作过程。

View

- **Toolbar:** 显示/不显示工具栏。
- **Status Bar:** 显示/不显示状态栏。
- **Sunplus ID:** .运用 Sunplus 算法计算出上传/下载文件内容的校验码。
- **File Checksum:** 显示指定文件内容的校验码。
- **Goto Address:** 跳转到指定地址，显示从指定地址开始的数据。
- **Edit:** 对“memory window”窗口中的内容进行编辑。

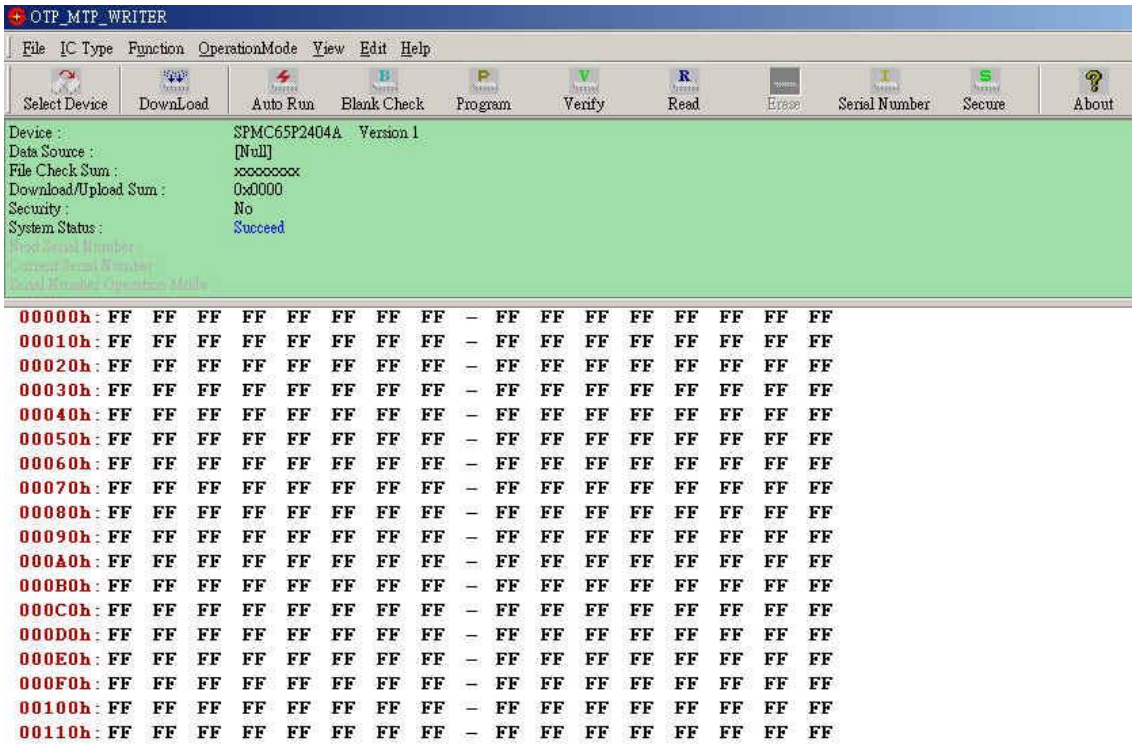
Help

- **About OTP/MTP Writer:** 显示 OTP/MTP Writer 软件的版本信息。

23.3.6 Serial Number 功能

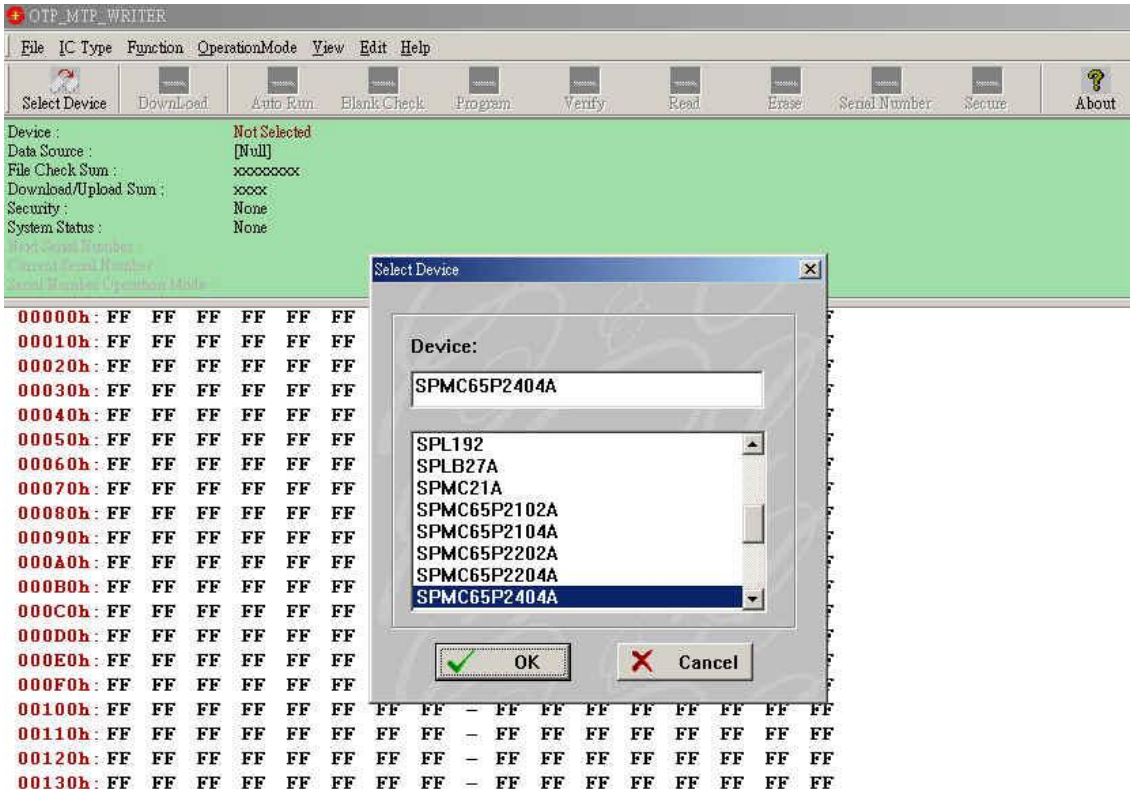
为对量产产品进行管理，用户可以使用 Serial Number 功能在指定地址为产品设置序列号码。比如像遥控器这样的产品，需要使用此功能来为每一个 OTP 芯片设置不同的序列号码。Sunplus OTP/MTP Writer 软件为用户提供了一个简单易用的界面来写入产品的序列号，有助于提高产品的效率。SPMC65X 系列的微控制器指定 4 个字节（地址为$7FF0～$7FF3）来写入序列号码。其中$7FF0 为四个地址中的最低地址，$7FF3 为最高地址。如果用户不想设置序列号码，这 4 个字节的默认值为$FF。

普通状态



操作步骤

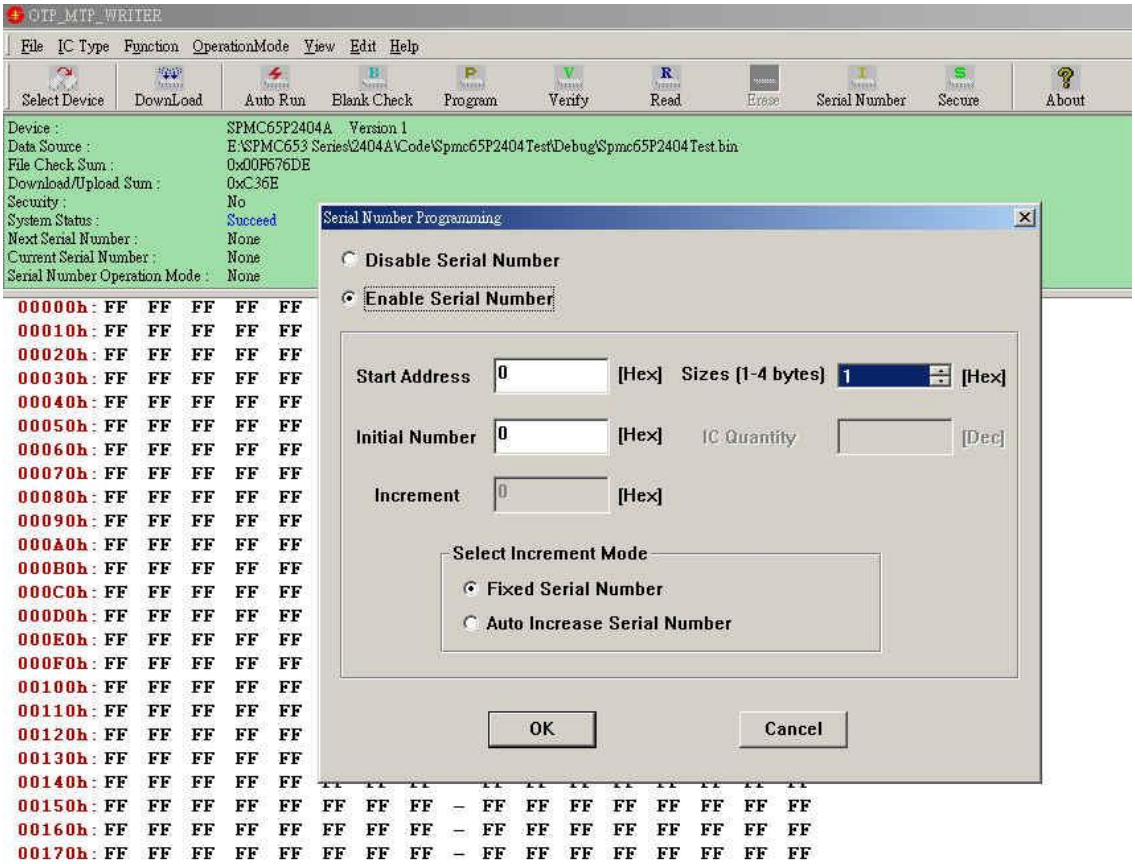
n 选择 OTP 芯片型号



n 下载源程序

n 启动 serial number 功能

从 Function 菜单中选择 Serial Number 或者点击 Serial Number 图标来启动此功能。所有设置值均默认为十六进制，大小写均可。



Start Address

默认的起始地址是$7FF0。

Sizes (1-4 bytes)

选择序列号的写入字节数。最多可以写入 4 个字节。

Initial Number

写入初始序列号。

IC Quantity

本版此项功能无效。

Increment

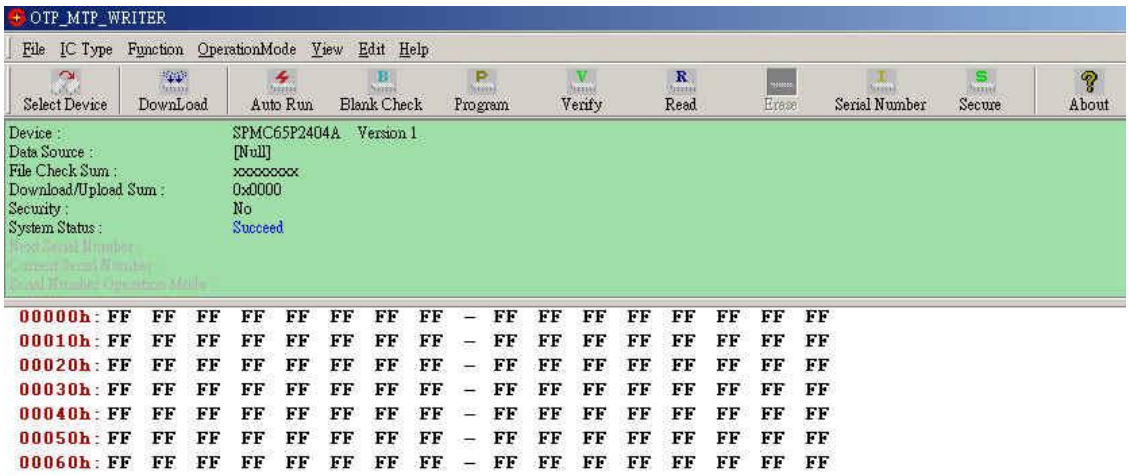
设置增量值。选择 auto 模式时有效。

Mode Selection

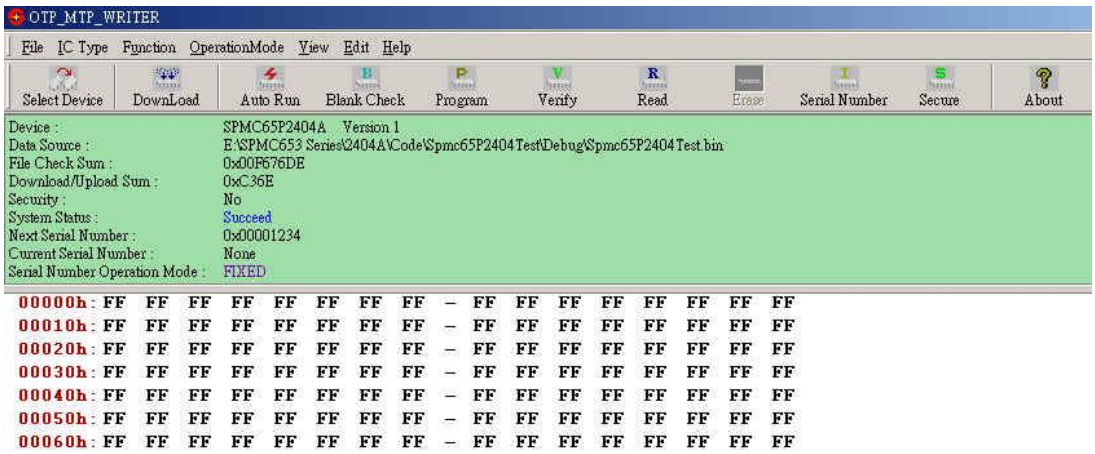
系统为用户提供了两种模式来设置序列号码：固定模式和自动模式。固定模式下，*increment* 项无效，所有芯片的序列号码为固定值。自动模式下，用户可以设定序列号码的增量值，烧录芯片的序列号将从设定值开始递增。

n Message Window

如下图绿色区域所示，“message window”显示了 Sunplus OTP/MTP Writer 烧录界面的所有操作信息。



在“message window”中，关于序列号有三项显示内容。用户下载源程序前，显示内容无效（显示为灰色）。在前面几张图中可以看到，用户设置序列号前，有关序列号码的内容显示为“None”。



用户设置了序列号后，“message window”中便可以看到相关信息。

Next Serial Number

下一个要烧录芯片的序列号码，此号码尚未写入任何 OTP 芯片中。

Current Serial Number

上一次烧录芯片的序列号码。此项首次显示为“None”。

Serial Number Operation Mode

此项显示用户选择的序列号设置方式是“Fixed”模式还是“Auto”模式。

n 举例

输入信息

Start Address:$7FF0

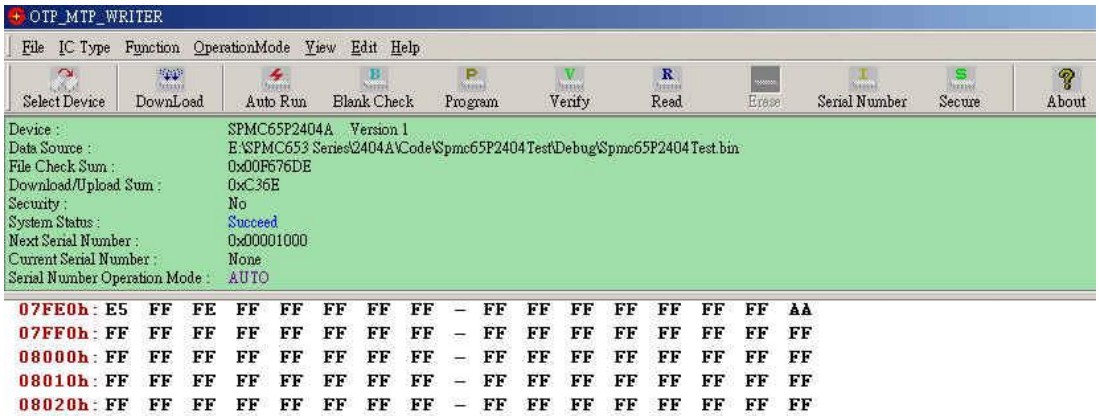
sizes: 2

Initial Number: $1000

Increment: $02

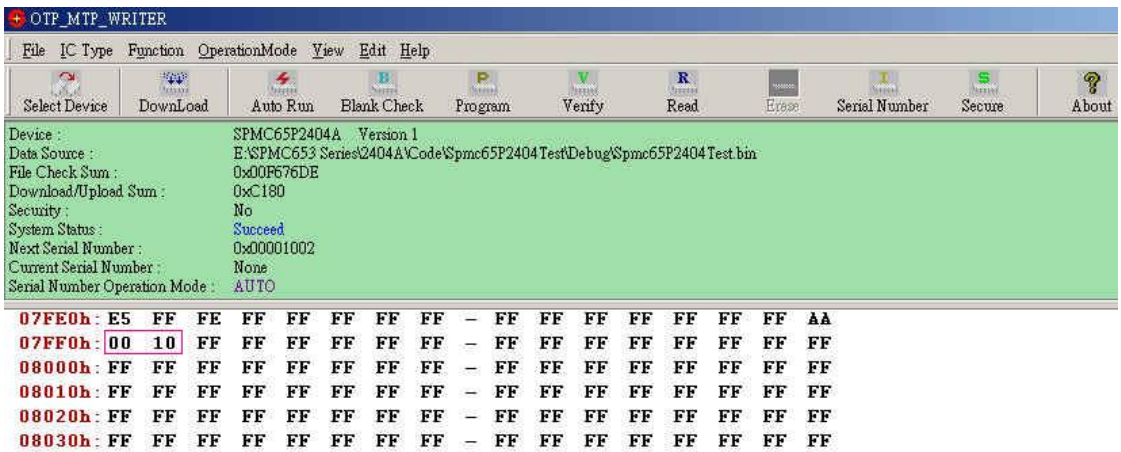
Mode: Auto

初始序列号设置为$1000，其地址为$7FF0 和$7FF1（$7FF0 为低地址）。相关信息设置完成后，其状态显示如下。此时，当前的序列号码仍然显示为“None”。



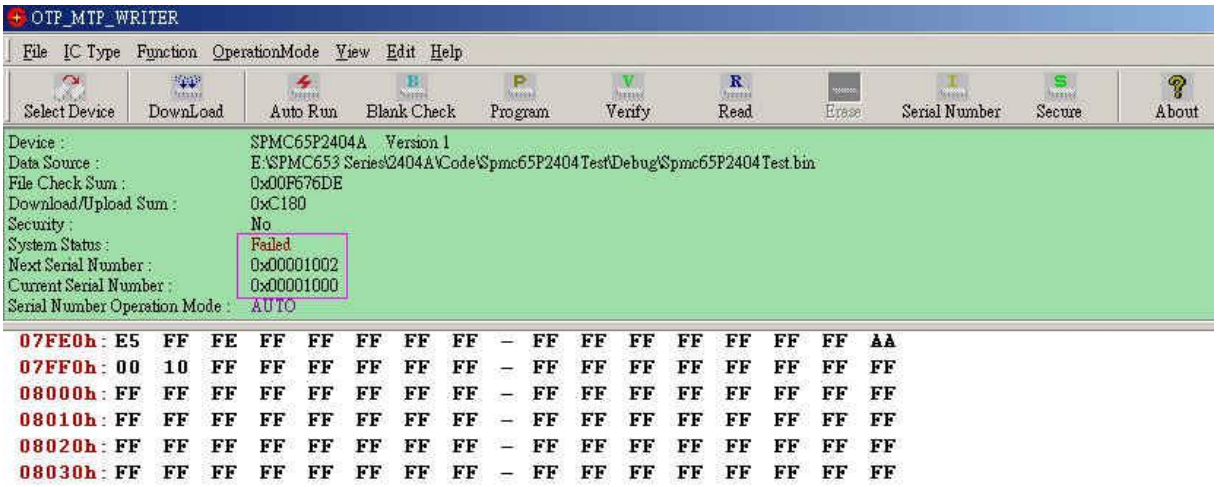
点击“Program”图标

程序烧录完成后，“Current Serial Number”显示为$1000，“Next Serial Number”显示为$1002。同时，校验码也被重新计算。如果程序烧录成功，序列号码和校验码的值将被连续累加。



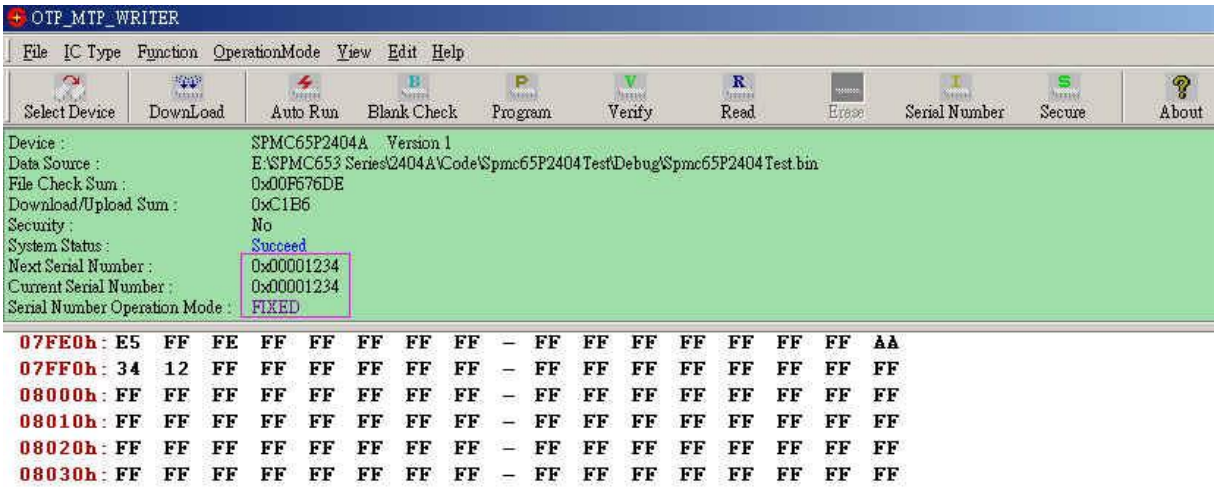
发生错误

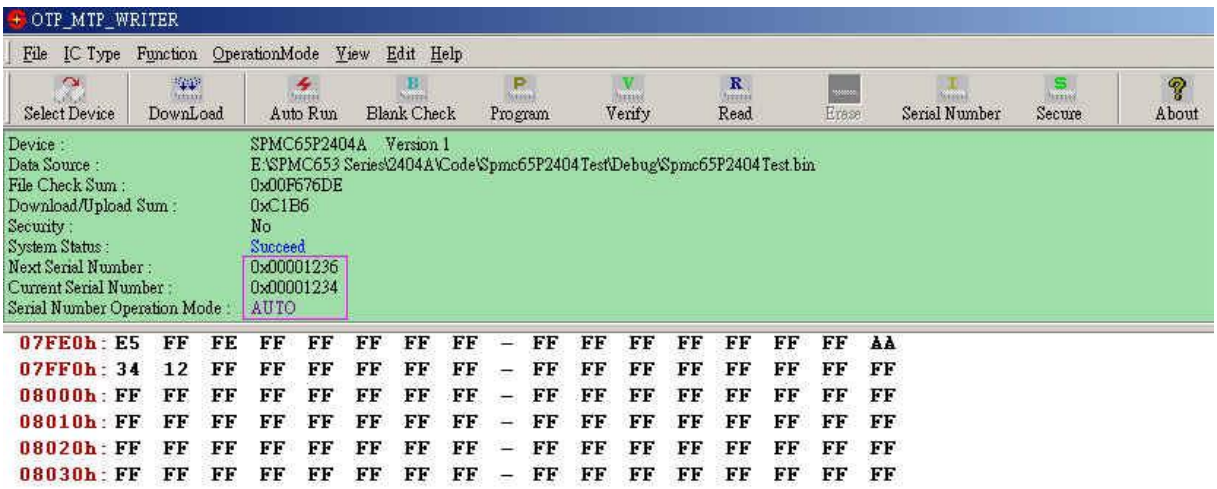
烧录过程中，如有错误发生，系统将恢复至上一次的操作状态。比如，在上例中，系统准备烧录的下一个芯片的序列号码为$1002，但在烧录中，有错误发生，无法完成烧录操作，这时，“message window”窗口将显示“Failed”信息，序列号码维持上一次的设置值。



读出 OTP 芯片中的内容

有时，用户可能需要读出 OTP 芯片中的内容，之后再进行程序烧录。如果操作模式为固定模式，下一个要烧录芯片的序列号码将与当前序列号码相同；如果操作模式为自动模式，下一个要烧录芯片的序列号码为读出的序列号码（当前的序列号码）与增量值之和。





23.3.7 重要注意事项

- 在未用到的芯片内存中写入$FF，以减少程序的烧录时间。
- 为了配合第三方生产厂商的校验和，凌阳 OTP_MTP_Writer 采用了不同的校验和计算方法。对于 SPMC65x 系列芯片而言，下载/上传文件的总和（校验和）包括程序存储区的内容、芯片配置寄存器的值和用户信息。OTP_MTP_Writer 下载/上传的总和与 Hilosystems 编程器的校验和相同，也与 Q-Writer 的下载/上传文件的总和相同。
- OTP_MTP_Writer 的文件校验码与 FortisIDE 中的相同。
- 所有的烧录动作都可以在“Status Window”显示。
- 当使用“auto run”功能进行烧录时，执行时间因操作模式的不同而不同。此间，一旦有错误发生，烧录动作将会停止，并弹出错误提示信息。
- 擦除和加密功能并非对所有芯片都适用。
- 系统可以保存上一次烧录的芯片类型。因此，下次烧录时，系统将询问用户是否选择上次使用的芯片类型，如果选择是，系统会自动下载相应的资源文件。
- 系统可以保存上次选择的自动烧录模式，因此，在下次进行烧录时，系统将默认为上次设置的自动烧录模式.
- 除非烧录器处于空闲状态，否则用户不能将转接板与烧录器断开。如果更换了转接板，必须重新选择芯片类型。
- 若“message window”窗口显示“Device: Not Selected”，表明芯片类型不正确。需要重新选择芯片类型。
- 一旦选择了芯片类型，烧录器 RAM 中的代码（*.bin, *.tsk）将被清除。当按下烧录器上的复位键，芯片的类型必须重新选择。
- 若只是按下烧录器上的 write 按键，烧录器开始对芯片进行烧录，这个过程将不会在电脑中显示出来，而且“auto run”功能无效。不过，用户在程序下载完毕后，断开 COM-Port 和电脑的连接，仍然可以进行芯片烧录。
- 打开或者编辑一个文件后，文件内容将自动下载到烧录器中。

Note!

启动加密功能后，烧录器仅对芯片的加密地址进行烧录。芯片一旦加密，用户将不能再读出其中的内容。因此，在对 OTP 芯片进行加密前，用户必须完成 program 和 verify 的操作。

23.4 第三方生产的烧录器

为了提高芯片的烧录效率和多种烧录器选择，Sunplus 公司委托第三方制造商研发烧录器，支持 SPMC65x 系列芯片的烧录 。由 Hilo systems 公司生产的新一代 ALL-100 支持多种芯片的烧录，同时，还可以做为 gang-wtiter 使用，进行 8 颗 OTP 芯片的同步烧录，并且用户最多可以将 8 个 ALL-100 与电脑连接，来同时控制 64 片 OTP 芯片的烧录。ALL-100 的功能特点如下所示：

23.4.1 产品介绍

ALL-100 支持对多种类型和封装 IC 进行烧录，IC 类型涵盖 EPROM、EEPROM、Serial PROM、FLASH、PLD/CPLD/FPGA、MPU/MCU、NAND Flash、RS-MMC/MMC/SD Card 等，大多数的封装类型如 DIP、SDIP、SOP、SSOP、TSOP、PLCC、QFP、BGA 等都可直接使用 DIP48 烧录座进行烧录，或者通过适当的转换座来进行烧录。

ALL-100 的烧录管脚接口采用专业的设计技术，其高速性、高可靠性与高扩展性为用户提供了卓越的烧录品质。ALL-100 强大的功能可以充分满足当今各种芯片的烧录或者测试的需要，对于最新投放市场的低电压芯片，也不例外。

ALL-100 采用高速的 USB 接口可以连接绝大部分的 Windows 98 SE/Me/2000 SP4/XP SP1/Server 2003 操作系统的笔记本或台式机，而且一台电脑可以和多达 8 台同种类型的烧录机同时连接，进行同步烧录，提高烧录效率。ALL-100 是功能强大的单插座型万用烧录器，适合于不同烧录时序和封装的芯片，帮助工程师解决各种烧录问题。此外，用户可以选择多达 8 插座的 Gang 烧录模块，为日后中小型生产线的扩充节省成本。您可以参考 ALL-100/GANG opt.了解更为详细的信息。

23.4.2 功能特色

1. 性能卓越：每个烧录接脚都具备高弹性化的接口线路，噪声低而烧录速度快，并具有过载保护、自我诊断等功能，以确保 ALL-100 的卓越性能。

2. IC 支持层面广：支持多达 9000 多种可烧录 IC，范围涵盖 EPROM、EEPROM、Serial PROM、FLASH、PLD/CPLD/FPGA、MPU/MCU、NAND Flash、RS-MMC/MMC/SD Card 等类型，及 DIP、SDIP、SOP、SSOP、TSOP、PLCC、QFP、BGA 等封装。

3. 承接 ALL 列的转接座(ADAPTER)／转换座(CONVERTER)：ALL-100 的单座烧录模块可承接原 ALL 系列超过 500 种的转接座(ADAPTER)与 150 种的转换座(CONVERTER)，以确保旧资产仍能持续创造新价值。

4. 适合工程单位及量产单位：ALL-100 可烧录的 ICs 涵盖范围广泛、扩充性灵活，满足用户对其上市时机的要求。
5. 可进行多台烧录：透过 USB 可将多达 8 台同型机连接，进行同步烧录，可让每个作业人员的烧录产量达到最高峰。
6. 使用方便：使用 Windows 界面，操作人性化，在从 PC 机上载入烧录程序后，用户只须从屏幕菜单中选择 Blank check, Program, Auto 等功能，然后按烧录器的 YES 键即可开始烧录。操作非常简单。
7. 每周可从网上获得最新器件支持：免费享有器件支持 S/W。

23.4.3 规格

支持组件
管脚数：从 8 pins 到 300 pins 以上的可烧录 IC 种类：涵盖 EPROM、EEPROM、Serial PROM、FLASH、PLD/CPLD/FPGA、MPU/MCU、NAND Flash、RS-MMC/MMC/SD Card 等。
芯片封装
基本：多数 DIP 封装可直接使用标准配备 DIP48 pin 万用型连接器 其它：多数 SOP、TSOP、PLCC、QFP、MLF、SDIP 等封装可透过适当的转接座(ADAPTER)或转换座(CONVERTER)由 DIP48 转换。部分 SOP、TSOP、BGA、MLF 等封装需使用单座烧录模块进行烧录，少数 BGA 封装需再另行添购 TOP。
最大烧录数目
多达 8 插座的万用 Gang 烧录模块可供用户选择。
控制器
采用大容量的 16 位高速 FPGA & CPLD 控制器。
接口
1 个 USB 接口
数据传输速率
USB 1.1 : 12 Mb/S USB 2.0 : 480 Mb/S
最多连接台数
透过 USB 接口可同时连接 8 台 ALL-100
功能
Load file, Read Master, Program, Verify, Auto, ID Check, Checksum, Blank Check, Erase, Secure, Protect/Unprotect, Edit, Function Configuration, Self-Diagnostics.
PC 系统需求
Intel Pentium 或兼容机； 64MB 以上内存； 硬盘(有 20 MB 以上可用空间)；

	操作系统：Windows 98 SE/ Me/ 2000 SP4/ XP SP1/ Server 2003。
电源	
	AC 电压:100-240 VAC 频率：50-60 Hz 消耗电力：50W
尺寸	
	L x W x H - 260mm x 150mm x 100mm.
重量	
	4 公斤
操作温度	
	0-40°C(32-105 °F)
安规	
	获得 CE 认证。

24 汇编工具

24.1 简述

凌阳汇编工具箱包含以下工具：汇编器(xasm8.exe)、链接器(xlink8.exe)和库文件生成器(xlib8.exe)。汇编器(xasm8.exe)可以将汇编程序进行编译生成以二进制码，产生目标文件。链接器(xlink8.exe)可以将多个*.obj 或*.lib 文件整合成一个可执行文件。库文件生成器(xlib8.exe)可以将程序以另一种形式（*.lib 文件）储存，用于不同的场合。通常，大多数基于凌阳 8 位 MCU 的工程开发都可依靠该汇编工具箱完成。每个工具的具体用法将在下面详细介绍。在 Fortis IDE 工具中集成了包括 xasm8.exe, xlink8.exe,xlib8.exe 在内的所有可执行文件。用户可以用 Fortis IDE 完成编译、链接和库文件生成等多项工作。
注意本手册只着重介绍如何使用这些工具，关于汇编指令集请参考第 25 章。

警告：凌阳汇编工具仅适用于凌阳在线仿真器(ICE)和凌阳8位MCU芯片。

24.1.1 汇编器(xasm8.exe)

Xasm8 是凌阳 8 位微处理器的汇编器，负责将凌阳程序代码转换为目标代码。
伪指令和宏
Xasm8 工具包含了丰富的伪指令和宏指令用于汇编编程。
可重定位目标代码
Xasm8 可将汇编代码转换为可重定位的目标代码。
条件汇编功能
汇编器可以将任意段的程序进行条件汇编，并且可以嵌套任意多层。同时，进行条件的配对检查，并显示当前目标代码域的条件汇编，这样可以避免出错。

列表文件
屏幕上以列表的形式列出所有错误的类型和出错的地方。

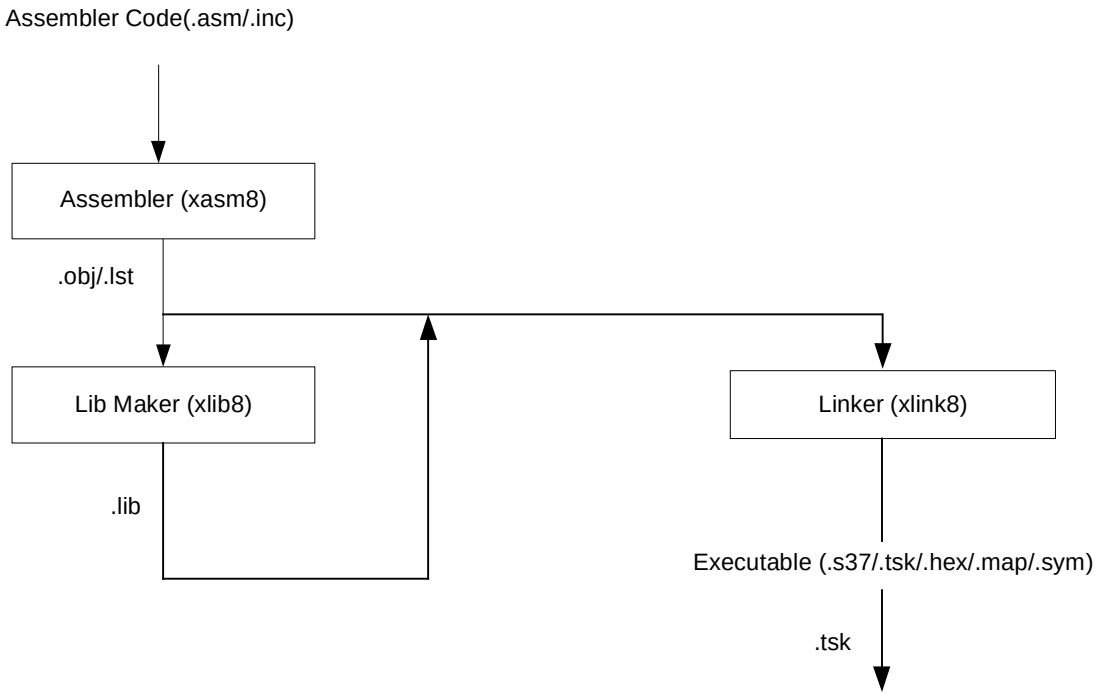
24.1.2 链接器(xlink8.exe)

链接器将代码定位到编程人员指定的执行地址上，并且按照此地址格式生成代码文件。链接器必须不断运行，即使程序在正常运行并且没有外界引用，也是如此。

24.1.3 库文件管理器(xlib8.exe)

库文件生成器用于管理目标文件，生成库文件。生成的库文件可以被链接到任何程序段。

24.2 编码流程



24.3 汇编器

24.3.1 文件扩展名

如下表所示，没有特殊定义时，文件默认的扩展名

汇编器	
.asm	汇编文件
.obj	目标文件
.lst	列表文件

24.3.2 操作符

除了求幂运算用指数算法外，以下操作符所实现的运算都用整数算法。所有的比较操作结果为真，返回 1；为假，返回 0。操作码所代表的运算功能如下表所示。

操作符	描述
&& !! ||	逻辑与 逻辑非 逻辑或
& or .AND. ~ or ! or .NOT. ^ or .OR. .XOR.	位与 位非 位或 位异或
+ -	指定一个正操作数 指定一个负操作数.

**	无符号数求幂
* /	无符号数求积 无符号数求商
+ -	加 减
>> or .SHR. << or .SHL. .MOD.	数据位右移，左移入数据位为 0 紧接着可以定义移位的数目 数据位左移，右移入数据位为 0 紧接着可以定义移位的数目 求余

以下操作数必须由一个圆点、空格或者 tab 键来开始或停止

操作符	**描述**
< or .LOW.	地址的 bits 0 – 7(用于可重定位字节地址)
> or .HIGH.	地址的 bits 8 – 15(用于可重定位字节地址)
.LOW16.	32 位地址的 bits 8 – 15(用于可重定位字节地址)
.HIGH16.	32 位地址的 bits 16 – 31(用于可重定位字节地址)
^ or .HIGH8.	24 位地址的 bits 16 – 23(用于可重定位字节地址)

以下操作符用于比较指令

操作符	**描述**
!= or .NE.	不等
> or .GT.	大于
< or .LT.	小于
>= or .GE.	大于或等于
<= or .LE.	小于或等于
.UGT.	无符号数大与
.ULT.	无符号数小于

操作符的优先级如下所示，第一行的优先级最高，依次降低。同一行的具有相同的优先级，但使用括号时，括号里的优先级最高。

'&&' or '||' or '!!'
'^' or .OR.
.XOR.
'&' or .AND.
.EQ., .NE.
.GE., .GT., .LE., .LT., .UGT., .ULT.
'>', .HIGH., '<', .LOW., .LOW16., .HIGH16., '^', .HIGH8.
'>>', .SHR., '<<', .SHL.

'+', '-'　　；加、减
'*', '/', .MOD.
'**'
'+', '-'　　；单目操作符，正、负号
'! ' or .NOT., '~'

n　高字节和低字节

要得到 16 位地址的高字节，用.HIGH.命令，可以定位一个地址的高 8 位。要得到 16 位数据的低字节，用. LOW.命令，可以定位一个地址的低 8 位。

n　结构和过程介绍

Ø　介绍

STRUCT
类别：　定义
功能;　定义一个结构体的开头
语法:　label:　.STRUCT
注意 ：　结构体定义
举例:　test1:　.STRUCT

ENDST
类别:　定义
功能:　定义一个结构的结束
语法:　.ENDST
注意:　结构定义结束

PROC
类别:　定义
功能:　定义一个过程的开头
语法:　label:　.PROC
注意:　过程开头定义
举例:　test1:　.PROC

ENDP
类别:　定义
功能:　定义一个过程的结束
语法:　.ENDP
注意:　过程定义结束

Ø　结构定义

语法:
Struct_name:　**.STRUCT**
　　子项描述
　　.ENDST

举例:

```
test1:          .STRUCT
        class:          .DB   10
        name:           .DW   'apple pie'
        count:          .DD   01000h
            .ENDST
; 在这个例子中，定义了一个结构体'test1'
; 它包含 3 个子项'class', 'name', and 'count'
; 子项的初始值分别为 10, 'apple pie', 01000h
```

Ø **结构变量定义**

语法:

```
Struct_variable:   .struct_name [expression_list]
        ; 这个表达式列表用于向结构体 struct_variable 存储子项值
```

举例:

```
        Stru_var1:   test1 [7, 'cake',200h]
        Stru_var2:   test1 [6,,500h]      ; 不把值存入第 2 个子项.
                                          ; 因此保持初始值
```

Ø **结构变量参考**

语法:

```
Struct_variable@chiled_field
```

举例:

```
            LDA   Stru_var1@class
        ; “stru_var1”是一个结构体变量，“class”是这个结构体的一个子项
```

Ø **过程定义**

语法:

```
Proc_name:      .PROC
                    Instruction_list
                    RTS
                .ENDP
```

Ø **过程引用**

语法:

```
        CALL        proc_name
```

举例:

```
        CALL        sub1            ; sub1 是一个之前已经定义的过
```

程。

n 段

Ø 段的声明

SECTION

功能: 创建一个用户定义段

语法: *label:* .SECTION options

注意: 创建一个用户定义段，有如下选项：

AUTO_STACK COMMON PAGE0 REF_ONLY

举例:

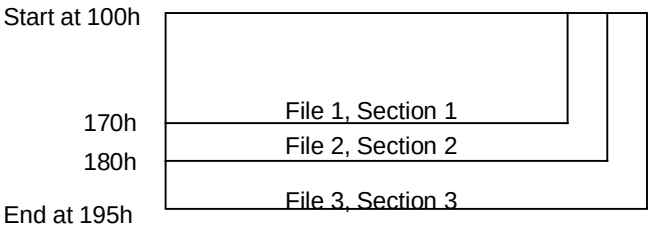
```
MySec1:  .SECTION  COMMON
         LDA #11
```

AUTO-STACK

各个段默认为 AUTO-STACK 格式，即：相同段名的段分配地址时会自动的连接起来。

COMMON

在 RAM 区使用，将不同文件中不同名称的段以相同的起始地址链接起来。因为该功能并不能自动指定重定位地址，因此链接器会提示需要一个偏移地址。偏移地址是这样计算的：从加载映像图中得到起始地址，加上最长段的跨度，就得到了偏移地址。



在上例中，“file1, section 1”的地址 100h 即初始地址，段长度为 70h，从而算出该段的尾地址是 170h；在同样的起始地址迭加“file3, section 3”，段长度 95h，尾地址变为 195h；在同样的起始地址迭加“file2, section 2”，段长度 80h，由于没有超过该链接段的最大长度，因此尾地址保持不变。第一个占用了 COMMON 段的文件不可以声明为 COMMON，注意：除了 RAM 区的地址，其它地方 COMMON 段只能定义一次，否则，不同的段将会被链接到相同的地址，造成互相覆盖，发生错误。

PAGE0

可以在预定义的 PAGE0 段(00h~FFh)定义数据。在 PAGE0

段中声明的外部符号(用 PAGE0 的修改器)会拥有 PAGE0 的赋值。

```
.PAGE0
var1        .DB     2
.EXTERNAL  PAGE0 var2,PAGE0 var3
```

可以用 PAGE0 的修改器产生用户自定义的 PAGE0 段，并指向它们所声明的名字。

```
Mysec1: .SECTION    PAGE0
var4    .DB 3
```

REF_ONLY

仅作为引用的段链接.

```
MySec2: .SECTION    ref_only
        INX
        LDA     #11
```

Ø **段摘要表**

链接器生成了一个段摘要表，作为它载入的 map 文件的一部分。在这个表里，以段的创建顺序，列出了所有要链接段的名字。对于一段来说，表里只包含每个实际用到的段的名字和起始结束地址。起始地址即链接器所需要连接段的起始地址。如果段 1 被两个文件连接，定位方法如下：

```
For file 1:  section1:     from 100h to 180h
For file 2:  section1:     from 230h to 400h
```

段摘要表将会显示：

```
Section Name   Start        End
Section1       000100       000400
```

n 宏举例

Ø 数字比较

```
Mac1:    .MACRO          arg1

         .IFNMA      1
            .EXIT        “Mac1 needs an argument”
         .MACEXIT
         .ENDIF

         .IF         1=arg1
month:   .DB         1
         .MACEXIT
         .ENDIF
```

```
            .IF 2=arg1
month:      .DB             2
            .MACEXIT
            .ENDIF

            .IF             3=arg1
month:      .DB             3
            .MACEXIT
            .ENDIF
            .IF4=arg1

month:      .DB             4
            .MACEXIT
            .ENDIF
            .IF5=arg1
month:      .DB             5
            .MACEXIT
            .ENDIF
            .IF             6=arg1
month:      .DB             6
            .MACEXIT
            .ENDIF
            .ENDM
```

Ø 字符串比较

```
Mac2: .MACRO        arg1
            .IFNMA      1
            .EXIT       "Mac2 needs an argument"
            .MACEXIT
            .ENDIF

            .IFSAME         "Monday",arg1
day         .BYTE       1
            .MACEXIT
            .ENDIF

            .IFSAME         "Tuesday",arg1
day         .BYTE       2
            .MACEXIT
            .ENDIF
            .IFSAME         "Wednesday",arg1
day         .BYTE       3
```

```
                .MACEXIT
                .ENDIF
                .IFSAME         "Thursday",arg1
                .ENDIF
day             .BYTE       4
                .MACEXIT
                .ENDIF
                .IFSAME         "Friday",arg1
day             .BYTE       5
                .MACEXIT
                .ENDIF
                .IFSAME         "Saturday",arg1
day             .BYTE       6
                .MACEXIT
                .ENDIF
                .IFSAME         "Sunday",arg1
day             .BYTE       0
                .MACEXIT
                .ENDIF
                .EXIT           "Argument error "
                .ENDM
```

Ø 传递标号

```
Mac2: .MACRO                arg1
                .ASCII          arg1
                .ENDM
; 按如下的方式调用宏
                mac2                label1
;  扩展宏如下：
                .ASCII          label1
```

Ø 向操作域传递参数

```
good:           .MACRO              arg1,arg2,arg3
name:       .DB             arg1
company:        .ASCII          arg2
quantity:       .DD             arg3
                .ENDM
;  按如下的方式调用宏：
                    good                "apple pie", UCLA,17425
; 扩展宏如下：
name:       .DB             "apple pie"
company:        .ASCII          UCLA
quantity:       .DD             17425
```

Ø 向标号域传递参数

作用是改变程序结构

```
good:       .MACRO      arg1,arg2,arg3
arg1:       .DS         25
arg2:       .DS         20
arg3:       .DD         0
            .ENDM
;   按如下的方式调用宏：
good        name, company, quantity
; 扩展宏如下：
name:       .DS         25
company:    .DS         20
quantity:   .DD         0
```

Ø 宏参数

宏的参数之间通常用逗号(,)来隔开，然后将其传到宏里。也可以用{ }代替逗号作为分隔符，每个参数必须由成对的{ }分隔才是合法的。

```
mac1:       .MACRO arg1,arg2,arg3
            .DB arg1
            .DB arg2 arg3
            .ENDM
;宏的调用:
            mac1        23h,{14h},{,62h}
;编译后结果如下:
            .DB 23h
            .DB 14h,62h
```

Ø 宏的递归调用

在宏的递归调用，,变量 **arg1** (计数)控制递归层数。在每次递归调用中，保留一个字节的值放入变量 **arg2.**中。每递归调用完成一次，**arg1** (计数)就减一。

```
mac3:       .MACRO              arg1,arg2
count:      .VAR                arg1
                .IF             count = 0
                .MACEXIT
                .ENDIF
count:      .VAR                count - 1
                .DB             arg2
                mac3            count,arg2
                .ENDM
; 调用宏：
```

```
                mac3            5,1ah
; 产生结果：
count:    .VAR          5
                .IF             count = 0
                .MACEXIT
                .ENDIF
count:    .VAR          count - 1 ;
                .DB             1ah
                mac3            count,0ah
                . . .
count:    .VAR          count - 1
                .IF             count = 0
                .MACEXIT
                .ENDM
                . . .
                .ENDM
```

宏的递归调用中可以进行嵌套，嵌套层数不受限制。在退出宏之前无需保持 IF/ENDIF 的配对，因为 xasm8 会自动将其配对(伪指令 MACEXIT 将程序返回到调用该宏指令的下一条指令)

24.3.3 汇编语言

n 数制

汇编器默认数制为十进制。下表列出了各种数制的表达。应用时，将其与数值组合即可。

二进制	**B 为**后缀或者%为前缀
八进制	**O** 或 **Q 为**后缀
十进制	**D 为**后缀或无后缀
十六进制	**H 为**后缀或以**$, 0X / 0x** 为前缀
ASCII 字符串	用双/单引号表示(Example:“car” or ‘car’)

n 数制定义

一个数字默认的数制为 10 进制。如果要表示成其它数制，需要用上表所示的方法，给数字加上后缀或前缀。

n 字符串定义

字符串必须放在双引号(“”)或单引号(‘ ’)内表示，除非语法上另有所指。

n 空格

在 XASM8 中，可以在操作数之间或操作数和操作码之间加入空格。

```
                .IFTRUE A.AND..NOT. B
        or      .IFTRUE A .AND. .NOT. B
```

n 标号(Labels)

标号的命名区分大小写。一个全局的标号可以取任意数字和字母的组合，总长度不能大于

32 个字符。所有的标号必须以字母或下滑线(_)开头，非数字和字母型的标号不合法，但下滑线或问号(?)命名的标号除外。宏名可以字母、下滑线或百分号(%)开头。标号的后缀可以是字母、下滑线或井号(#)。

n 局部标号(Local Labels)

局部标号只能在当前汇编文件中使用。汇编程序用标号作为标识，可以令它们被全局引用。利用局部标号的局部引用特性，可以在各个局部区域内进行重复利用这些标号所定义的代码。

```
LABEL1:                 or      LABEL1:
?x:     INX                     x?:     INX
?y:     JMP   ?x                y?:     JMP   x?
        JMP   ?y ;                      JMP   y?


LABEL2:                 or      LABEL2:
?x:     INX                     x?:     INX
?y:     JMP   ?x                y?:     JMP   x?
        JMP   ?y                        JMP   y?

LABEL3:                 or      LABEL3:
?x:     INX                     x?:     INX
?y:     JMP   ?x                y?:     JMP   x?
        JMP   ?y                        JMP   y?
```

说明:

汇编器通常将局部标号定义为以问号(?)作为前缀或后缀

局部编号必须以字母或问号开头。

当同一个局部标号在不同的区域中定义时，意义不同。

标号“?x” 不等同于“x?”。

局部标号长度最大为 32 个字符。禁止在局部标号中使用操作符(例如+)。建议所有标号的命名都遵循全局标号的命名规则。

n 区分大小写

汇编伪指令是不区分大小写的。可以大写、小写或大小写混用。但是所有的标号、宏名、结构名、结构变量名、段名和过程名都是区分大小写的。

n 程序计数器标志

可以用特殊的字符如($)或(*)表示程序计数器。

举例:

```
Smart_Branch: .macro    Target
        .if (($+2)-Target) <= 127)
                BEQ             Target
        .else
                BNE     $+5
                JMP     Target
        .endif
```

```
 .endm
.Code
Cond1:
        ...
Cond2:
        ...
        LDA Index
        CMP #60
        Smart_Branch Cond1
        ...
```

n 外部标号

尽管可以声明多个外部标号(见 **EXTERNAL** 介绍)，但是汇编器并不支持数学或逻辑运算中包含超过一个以上的外部标号。

下面的声明是合法的：

```
.EXTERNAL  VAR1,VAR2,PAGE0 VAR3,VAR4
```

下面的声明是非法的：

```
LDA#(VAR1 .AND. VAR2 .OR. VAR4)
```

n 预定义段

XASM8 包括预定义段、代码段(默认的)、数据段和零页段。如果使用预定义段，它会先于其它用户定义段而首先得到链接。

CODE, DATA, PAGE0

这个链接顺序不能改变。要获得更灵活的链接方式，可以使用用户定义段代替预定义段。

n 用户定义段

用户可以自定义段，段的名字不能超过 32 个字符，定义的段的数量最多为 4096 个。选项域控制着段的连接途径。选项可以选择任意链接顺序或以字母为序；用逗号将各选项分隔。

n 注释

注释为可选项，必须以分号(;)开头或；COMMENT 开始。

24.3.4 伪指令

伪指令可以被编译，但它区别于真正的汇编指令。伪指令以圆点开头，能够在程序中的任意位置调用。括号区域或参数是可选项。如果参数是用双层括号括起来的，那么该参数为可选，内层括号是其语法本身的定义。Example:*[[variable1]]*是可选参数，但应用时必须输入*[variable1]*。

n 设置伪指令

ABSOLUTE

功能: 切换到绝对定位模式

语法: *.ABSOLUTE*

注意:汇编指令一般处于相对模式(汇编器会产生可重定位的目标代码)。用该指令切换到绝对定位模式。

举例:

```
.ABSOLUTE
test1:      .DS 1
test2:      .DS 1
test3:      .DB test1&test2
```

LINKLIST
功能: 引导链接器定位汇编列表文件。
语法: .LINKLIST
注意:为了定位列表文件，伪指令 LINKLIST 必须和 SYMBOLS 连用。

举例: .LINKLIST

```
.SYMBOLS
.CODE
LDA #17
```

RELATIVE
功能: 切换到相对模式(JI KE 重定位)模式
语法: .RELATIVE
注意:切换方向为从绝对模式到相对模式的切换
符号(SYMBOLS)
功能:为目标文件输出符号信息
语法: .SYMBOLS
注意:为了定位列表文件，伪指令 SYMBOLS 必须和 LINKLIST 连用。

举例: .LINKLIST

```
.SYMBOLS
.CODE
INX
```

n 定义伪指令
ASK
功能:在编译执行时提示输入关键字并等待一个字符应答。
语法: [label:] *.ASK* *prompt*
注意:在终端上输出一个提示符并等待一个字符应答。从输入的字符减去 30h,将减去 30h 的结果设成 *label* 赋值。这样做的目的是在程序中引入一个 0/1 标志。直接输入字符即可，不需回车。在 pass 2 下，输出一个提示符和应答。

举例:

```
ver_num:    .ASK    AP code version for 2.0 (=1) or 1.0 (=0) ? :
```

CODE

功能: 切换到预定义代码段

语法: .CODE

注意:所有储存型变量和指令都写在这个段中。如果没有段被指定，CODE 段将是默认的。当编译可执行指令时，汇编器总是在相对模式下。

举例:

```
.CODE
Reset:      LDA     #7AH
```

COMMENT

功能: 使用该指令可以进行多行注释

语法: .COMMENT char

String char

注意:使用该指令就可以输入多行注释而不用每一行都用分号开始。汇编器会 COMMENT 定义的多行字符将都视为注释。这也是一个暂时屏蔽一部分代码的办法。注释字符可以任意长，但必须在该指令及定义的字符后面开始输入注释。

举例:

```
                .COMMENT @
                …
…           @
```

DATA

功能: 切换到预定义数据段

语法: .DATA

注意:所有的变量都在该段中定义，但指令不允许写在在该段。

举例:

```
.DATA
Var1:       .DB     12H
```

DEFL / VAR / SET

功能: 标号赋值

语法:

```
label:      .DEFL       value
label:      .VAR        value
label:      .SET        value
```

注意:可以在程序中任意改变标号所赋的值，但是不能用 DEFL 重新定义一个标号。

举例:

```
Count:      .VAR        10
.IF Count
    INX
    Count       .VAR        Count - 1
```

```
        .ENDIF
```

END

功能: 定义程序的结束
语法: .END
注意:定义了程序或一个包含文件的结束。

举例: .END

ENDM / MACEND

类别: Definition
功能: 定义宏的结束
语法: .ENDM
        .MACEND
注意:宏的结束定义

举例:
```
Mac1        .MACRO      arg1
        LDA     arg1
        .ENDM
```

ENDP

功能: 定义过程的结束
语法: .ENDP
注意:结束一个以伪指令 PROC 为开始的过程。

举例:
```
        test1:      .PROC
                    LDA#IO_PORT1
                AND     Mask_Value
                STA Setting
                RTS
                .ENDP
```

ENDST

功能: 定义结构体的结束
语法: .ENDST
注意:结束一个结构体的定义

举例:
```
        .PAGE0
        Body1:  .STRUCT
                Index:  .DB     10
```

```
        name:   .DB     'Drive1'
        value:  .DD     0ffcH
Body1:  .ENDST
MySt1:  Body1[20,'Car',7dh]
.CODE
        LDA MySt1@Index
```

EQU / EQUAL

功能: 标号赋值

语法: *label:* .EQUAL *value*

label: .EQU value

注意:给标号赋值，这个值可能会是一个表达式或其它象征性的符号。该表达式可以是由变量或立即数组成，因此变量必须事先声明。否则会编译出错。

举例:
```
label1:  .EQUAL  3Ah
label2:  .EQU    20
```

EXTERN / EXTERNAL / XREF

功能: 声明一个在其它文件中已经定义的标号。

语法:
```
.EXTERN     label[,label][,...]
.EXTERNAL   label[,label][,...]
.XREF       label[,label][,...]
```

注意:声明每一个在其它文件中定义了的标号，标号之间用逗号分隔。在数学或逻辑运算操作中，汇编器不支持两个或两个以上的外部标号引用。

举例:
```
.EXTERNAL   sn_lee, sn_ert
.EXTERN     var1, PAGE0 var2    ; var2 在 PAGE0 段
```

GLOBAL / PUBLIC / XDEF

功能: 声明其它文件中已经用到的标号。

语法:
```
.GLOBAL     label[,label][,...]
.PUBLIC     label[,label][,...]
.XDEF       label[,label][,...]
```

注意:将每个标号定义为全局标号，这样在其它文件中便可以引用。标号间用逗号隔开。链接器将会连接所有的全局变量和外部变量。

举例:
```
.GLOBAL   var1          ; 声明标号 var1 在其它文件中可用
.GLOBAL   var2,var3     ; 在同一行中多个声明是合法的
                        ; 当用逗号分开时，空格会被忽略
```

INCLUDE

功能: 在汇编程序中包含文件

语法: .INCLUDE *filename*

.INCLUDE “filename” ;for long filename format

注意:引导汇编器将指定的文件包含到程序中进行编译。Filename 可以是一个路径，但必须是要包含文件的完全展开的路径。每包含一个文件，都需要执行一条这样的伪指令。长文件名需要将文件名加上双引号。

举例: .INCLUDE src\lib\sub1.asm
.INCLUDE "C:\Program Files\凌阳\Fortis IDE\Hardware.inc”

MACEXIT

功能: 从宏中退出
语法: .MACEXIT
注意:MACEXIT 为立即并无条件退出的指令。执行 MACEXIT 将会恢复宏调用前的所有状态。

```
举例: mac1:    .MACRO          arg1,arg2
                .IFSAME         arg1,arg2
                .MACEXIT
                .ENDIF
                .INY
                .ENDM
```

MACRO

功能: 定义宏的开始
语法: label: .MACRO args
注意:一个宏的定义的开始。
举例: mac1: .MACRO arg1,arg2,arg3

MESSAGE / MESSG

功能: 在 pass2 阶段显示一屏信息
语法: .MESSAGE use_message
.MESSG use_message
注意:在汇编 pass2 阶段，屏幕上输出用户定义的信息
举例:Output: .MESSAGE Compiling AP code…

PAGE0

功能: 地址为 0h ～ 0FFh 的变量定义区
语法: .PAGE0
注意:零页数据即定义在地址 0h~0FFh 之间的变量，只能在这个范围内进行重定位。该指令允许汇编器用 8 位偏移量和最优化的错误校验方式。

```
举例: .PAGE0
        Var1:       .DB     21h
```

PROC

功能: 定义一个过程的开始

语法: *label:* .PROC

注意: 过程开始的定义

举例: 参考 ENDP

SECTION

功能: 创建一个用户定义段

语法: *label:* .SECTION options

注意:创建一个 用户定义段，其中可选项为：

AUTO_STACK PAGE0 REF_ONLY COMMON

举例: MySec1: .SECTION COMMON

LDA #11

STRUCT

功能: 定义一个结构体的开始

语法: *label*: .STRUCT

注意: 结构体定义开始

举例: 参考 ENDST

n Storage Directives

ASCII

功能: 存储 ASCII 字符串

语法: [label:] *.ASCII string*

注意: 将每个 ASCII 字符放入存储器中

举例:

test1: .ASCII John

test2: .ASCII “Mary”

BLKB

功能: 定义一个连续的字节变量，并给字节赋特定的值

语法: [label:] .BLKB bytes[,value]

注意: 连续变量的字节数以十进制的形式存储

举例:

Label1: .BLKB 17, 3AH ；定义 17 字节空间并在每个字节空间中存入 3AH

Label2: .BLKB 7,0 ；定义 7 个字节空间并在每个字节空间中存入 0

Label3: .BLKB 8 ；定义 8 个字节空间并在每个字节空间中存入 0

BLKL

功能: 定义一个连续的长字（32bits）变量，并给长字赋特定的值。

语法: [label:] .BLKL words[,value]

注意: 连续变量的长字数以十进制的形式存储

举例:

```
Label1:  .BLKL 17,43218765H
                          ;定义 17 个长字空间并在每个长字空间中存入 43218765H
         Label2:  .BLKL 7,0        ;定义 7 个长字空间并在每个长字空间中存入零
         Label3:  .BLKL 8          ;定义 8 个长字空间并在每个长字空间中存入零
```

BLKW

功能: 定义一个连续的字（16bits）变量，并给字赋特定的值。

语法: [label:] .BLKW words[,value]

注意: 连续变量的字数以十进制的形式存储

举例:

```
Label1:  .BLKW 17,4321H        ;定义 17 个字空间并在每个字空间中存入 4321H
         Label2:  .BLKW 7,0    ;定义 7 个字空间并在每个字空间中存入零
         Label3:  .BLKW 8      ;定义 8 个字空间并在每个字空间中存入零
```

BYTE / DB / DEFB / STRING

功能: 在 8 位存储器中保存数值

语法:
```
[label:]  .BYTE     [value][,value][,...]
[label:]  .DB       [value][,value][,...]
[label:]  .DEFB     [value][,value][,...]
[label:]  .STRING   [value][,value][,...]
```

注意:在连续的存储空间保存数值。数值之间用逗号分隔(数值可以为不同进制)。将 ASCII 字符串用单引号括起来(字符里有单引号，再用一层单引号)。

举例:

```
label1:   .BYTE   0              ; 定义 1 个字节空间
label2:   .DB     6              ; 定义 1 个字节空间，存入 6
label3:   .DEFB   5,2,8          ; 定义 3 个字节空间，依次存入 5、2、8
label4:   .STRING     'Kitty'    ; 在连续的空间内以 ASCII 形式存储字符“Kitty”。
label5:   .BYTE   'Kitty',0DH    ; 与之前一样，结束后，带有返回值
; 操作数之前的空格将被忽略，但需要在之前加一个逗号
label6:   .DB     'XASM''8'      ; 字符串中含有引号
label7:   .DB       “Xasm’8”
```

DD / LONG

功能: 在 32 位空间内存储数值

语法: [label:] .DD [[count]][value][,...]

[label:] .LONG [[count]][value][,...]

注意:在一个 32 位连续的空间内存储数值，数值之间用逗号分隔。数值可以为任意进制，但是以十六进制形式存储。

举例:

```
label1:  .DD     0A10H   ; 存入 0A10H
label2:  .LONG   'High'  ; 在 32 位空间内，存入 48H, 69H, 67H, 68H
label3:  .LONG   [5]31A2H    ; 在 32 位空间内存入 31A2H
```

DEFS / DS

功能: 保存 n 个字节空间（n 为十进制）

语法: label: .DS bytes

label: .DEFS bytes

注意: 如果零标志('-z' flag)是禁止的，数值($FF)存入定义字节。

举例:

```
label1:  .DS    16      ; 定义 16 个字节空间并存入$FF
label2:  .DEFS  25      ; 定义 25 个字节空间并存入$FF
```

DEFW / DW / WORD

功能: 在 16 位存储器内保存数值

语法: [label:] .DEFW [[count]][value][,...]

[label:] .DW [[count]][value][,...]

[label:] .WORD [[count]][value][,…]

注意:在连续的存储区内以十六进制形式保存十进制数值。[count]表示保存字的数量。数值可以为任意类型，用逗号分隔。如果存储 ASCII 字符串，用单引号将 ASCII 字符串括起来，如果 ASCII 字符串中已经包含单引号，则使用双引号将其括起来。

举例:

```
label1:  .WORD  0           ; 定义 1 个字空间，值为零
label2:  .WORD  10          ; 定义 1 个字空间，存入十进制的 10
label3:  .DEFW  1,2,3       ; 定义 3 个字空间，依次存入 1、2、3
label4:  .DW    'Hello', 0DH
; 以 ASCII 形式在连续的空间内存储字符串“Hello”，并带返回值返回，操作数之前的空格将被忽略，但需要在之前加一个逗号
label5:  .DW [4]31E2H       ; 定义 4 个字空间，均存入 31E2H
```

DOUBLE

功能: 将数值变为双精度

语法: label: .DOUBLE value[,value][,...]

注意:将数值转换为 IEEE 格式双精度浮点数。数值在必须事先定义为浮点数，数值之间用逗号分隔。

举例:

```
label1:   .DOUBLE        345.65   ; 存入 66H,66H,66H,66H,66H,9AH,75H,40H
label2:   .DOUBLE        70.0     ; 存入 00H,00H,00H,00H,00H,80H,51H,40H
```

DUP

功能: 协同其它存储指令保存数值

语法 1: [label:] .DB number DUP (value)
注意 1：定义一个连续字节的空间，并在每个字节内存入相同的数据
语法 2: [label:] .DW number DUP (value)
注意 2：定义一个连续字（16bits）的空间，并在每个字内存入相同的数据
语法 3: [label:] .FLOAT number DUP (value)
注意 3：定义一个连续浮点数（32 位）的空间，并在每个浮点数（32 位）空间内存入相同的数据
语法 4: [label:] .LONG number DUP (value)
注意 4：定义一个连续长字（32 位）的空间，并在每个长字内存入相同的数据
语法 5: [label:] .DOUBLE number DUP (value)
注意 5：定义一个连续浮点数（64 位）的空间，并在每个浮点数（64 位）的空间内存入相同的数据
举例:

```
.PAGE0
label1:   .DB        20    DUP (0)          ;定义 20 个字节的连续空间，并存入 0。
label2:   .DW        20    DUP ($0FF)       ;定义 20 个字的连续空间,并存入$0FF。
label3:   .DW        11Q   DUP (20)         ;定义 9 个字的连续空间，并存入存入 20。
label4:   .DW        11    DUP ($20)        ;定义 11 个字的连续空间，并存入存入$20。
; 浮点数和双精度数的表达格式依照 IEEE 格式
label5:   .FLOAT     20    DUP (10.982)  ; 定义 20 个浮点数的连续空间，并存入 10.982
lable6:   .DOUBLE    5     DUP (5223.29) ;定义 5 个双精度数的连续空间，并存入存入 5223.29
label7:   .DD        20    DUP (0)          ;定义 20 个长字的连续空间，并存入 0。
label8:   .DD        20    DUP ($12345678)  ;定义 20 个长字空间,并存入$12345678
label9:   .DW        5     DUP (?)          ;定义 5 个字的连续空间，并存入 0
label10:  .DB        3     DUP(17,20,12)    ;定义 9 个字节的连续空间
```

FLOAT

功能: 浮点数声明
语法: label: .FLOAT value[,value][,...]
注意:将数值转换为 IEEE 格式单精度浮点数。数值在必须事先定义为浮点数，数值之间用逗号分隔。

```
举例: label1:   .FLOAT  134.125    ; 存入 00H,20H,06H,43H
      label2:   .FLOAT  102.0      ; 存入 00H,00H,CCH,42H
```

ORG / ORGIN

功能: 定义段的起始地址

语法: .ORG *value*

. ORG value

注意:*value* 是一个段的地址偏移量，可以是标号也可以是数字。如果在链接时，会将段的起始地址加上偏移量作为重定位地址。

举例: .CODE

.ORG 600H

- 条件伪指令

ELSE

功能: 如果前面的条件为假则编译下面的指令

语法: .ELSE

注意:如果前面的条件不成立，则编译下面的指令

举例：

```
.IF
…
.ELSE
…
.ENDIF
```

ENDC / ENDIF

功能: 定义条件汇编程序段的结束。

语法: .ENDIF

.ENDC

注意:定义条件汇编程序段的结尾。没有配对的 IF – ENDIF 会产生错误信息。

举例: .IF

INX

.ENDIF

EXIT

功能: 结束汇编

语法: .EXIT *message*

注意: EXIT 不应单独存在，用于条件判断中，条件成立，结束汇编，输出用户定义的信息(最大 500 个字符)。如果条件为真，执行退出；如果条件为假，继续执行汇编。因为 EXIT 将汇编停止在 Pass 1，因此不会有列表文件。

举例: Count: .DB 25H

.IFNDEF Count

```
        .EXIT           ;没有被定义
        .ENDIF
```

IF / IFN / IFNFALSE/ IFNZ / IFTRUE

功能: 条件为真则进行汇编

语法:
```
        .IF             value
        .IFN            value
        .IFNFALSE       value
        .IFNZ           value
        .IFTRUE         value
```

注意:如果 *value* 不为零，汇编继续执行。该条件可嵌套任意多层。*value* 可以为数学运算表达式、标号或者字符串。

举例:
```
.IF     label1
        …
        .ENDIF
```

IFABS / IFNREL

功能: 标号是绝对地址时，汇编接着继续执行

语法:
```
.IFABS      label
        .IFNREL     label
```

注意:搜索 *label.*。如果 *label.*为绝对地址，汇编继续执行。如果需要重定位，则忽略 ELSE 或 ENDIF 之前的所有语句。如果没有找到 *label* 则产生错误信息。

举例:
```
.IFABS      label1
        …
        .ENDIF
```

IFCLEAR

功能: 在宏的递归调用里保证条件指令的配对平衡。

语法: .IFCLEAR

注意:在宏的递归调用里， IFCLEAR 保持条件指令 的配对平衡(例如 IF-ENDIF)，保持程序流始终在完整的条件控制下工作，因为有些 IF 条件总是导致宏的提前退出。当宏中包含 MACEXIT，又当宏提前退出时，可以使用指令 IFCLEAR 来保证宏的平衡

举例:
```
%DeclareTimers   .MACRO   Count
        .IFZ      Count
         .IFCLEAR
         .MACEXIT
        .ENDIF
```

```
 .IFDEF    D_Timer|<Order>
       %RAM_ALLOC R_Timer|<Order>,1
 .ENDIF
 Order:    .VAR       Order+1
    %DeclareTimers    Count-1
.ENDM
```

IFDEF

功能: 如果标号已经定义则顺序执行

语法: .IFDEF *label*

注意:搜索符号表里的标号 *label*，如果 *label* 已经定义则继续执行。如果没有定义，则忽略 ELSE 或 ENDIF 之前的语句。

举例:
```
var1:       .DEFL    14H
            .IFDEF   var1
            INX
            .ENDIF
```

IFDIFF / IFNSAME

功能: 如果两个字符串不同则顺序执行

语法:
```
.IFDIFF       string1,string2
.IFNSAME      string1,string2
```

注意:比较两个字符串。如果不相同，汇编继续执行。相同，则忽略下一个 ELSE 或 ENDIF 之前的所有语句。字符串有两种类型：有空格的和没有空格的。将空格放入单引号里，并用再套一层单引号表示空格。无空格字符串不需要套单引号。只能相同类型两个字符串进行比较，以逗号分隔。IFDIFF 指令对于比较宏参数很有用。

举例:
```
.IFNSAME   'my str1','my str1'
.IFDIFF    'Xasm"8','Xasm"8'
.IFDIFF      "Kitty","Kitty"
.IFDIFF    arg1,"January"
```

IFE / IFFALSE / IFNTRUE / IFZ

功能: 条件为假则顺序执行

语法:
```
.IFE          value
.IFFALSE      value
.IFNTRUE      value
.IFZ          value
```

注意:如果条件为真，则忽略下一个 ELSE 或 ENDIF 之前的所有语句。*Value* 可以是算术表达式，其它标号或者字符串。

举例:
```
.IFZ      label1
          …
          .ENDIF
```

IFEXT

功能: 如果为外部标号则顺序执行

语法: .IFEXT *label*

注意:搜索符号表里的标号 *label* 如果 *label* 已经定义则继续执行;如果没有定义,则忽略 ELSE 或 ENDIF 之前的语句; 如果没有找到标号, 会产生错误信息。

举例: .IFEXT label1

IFMA

功能: 如果宏参数存在则顺序执行

语法: .IFMA *argument#*

注意:该指令在宏的内部使用, 在宏的调用行扫描指定的参数。如果发现参数, 汇编继续执行。如果没有发现, 则忽略 ELSE 或 ENDIF 之前的语句。但是如果 argument# 设置为 0, IFMA 不会找到任何参数, 但汇编仍会继续执行。

举例:
```
.IFMA       2
    ECO_Flag    Var  1
    .ENDIF
```

IFNABS / IFREL

功能: 如果标号是可重定位的则顺序执行

语法:
```
.IFNABS     label
.IFREL      label
```

注意:搜索符号表里的标号 *label*, 如果 *label* 是可重定位的, 汇编继续执行。如果没有定义, 标号为绝对地址, 则忽略 ELSE 或 ENDIF 之前的语句。如果没有找到标号, 会产生错误信息

举例: .IFNABS label1

IFNDEF

功能: 如果标号还没有定义则顺序执行

语法: .IFNDEF *label*

注意:搜索符号表里的标号 *label*, 如果 *label* 没有定义则继续执行下面的语句。否则, 则忽略 ELSE 或 ENDIF 之前的语句。

举例: .IFNDEF label1

IFNDIFF / IFSAME

功能: 如果两个字符串相等则顺序执行

语法:
```
.IFNDIFF        string1,string2
.IFSAME         string1,string2
```

注意:比较两个字符串。如果相同，汇编继续执行；如果不同，则忽略下一个 ELSE 或 ENDIF 之前的所有语句。字符串有两种类型：含有空格的和不含空格的。将空格放入单引号里，并用再套一层单引号表示空格。无空格字符串不需要套单引号。只能相同类型两个字符串进行比较，以逗号分隔。IFSAME 指令对于比较宏参数很有用。

举例:
```
.IFSAME     'my str1','my str1'
…
.ENDIF
.IFSAME     'Xasm"8','Xasm"8'
…
.ENDIF
.IFSAME     "Teach","Teach"
…
.ENDIF
.IFNDIFF    arg1,"January"
…
.ENDIF
```

IFNEXT

功能: 如果表不是外部标号则顺序执行

语法: .IFNEXT *label*

注意:搜索符号表里的标号 *label*，如果 *label* 不是外部的，继续执行后面的语句；如果是外部标号，则忽略 ELSE 或 ENDIF 之前的语句；如果没有找到标号，会产生错误信息。

举例: .IFNEXT var1

IFNMA

功能: 如果该宏参数不存在则顺序执行

语法: .IFNMA *argument#*

注意: 该指令在宏的内部使用，在宏的调用行扫描指定的参数。如果没有发现参数，汇编继续执行到 ELSE 或 ENDIF。否则，则忽略 ELSE 或 ENDIF 之前的语句。但是如果 argument# 设置为 0，IFMA 将会查找所有的参数，如果至少存在一个，汇编仍会继续执行。

举例: .IFNMA 2

IFNPAGE0

功能: 如果该标号不是零页变量则顺序执行

语法: .IFNPAGE0 *label*

注意: 搜索符号表里的标号 *label*，如果 *label* 没有定义为零页变量，则继续执行；否则，忽略 ELSE 或 ENDIF 之前的语句；如果没有找到标号，会产生错误信息。

举例: .IFNPAGE0 label1

IFPAGE0

功能: 如果该标号是零页变量则顺序执行

语法: .IFPAGE0 *label*

注意:搜索符号表里的标号 *label*，如果 *label* 没有定义为零页变量，则继续执行；否则，则忽略 ELSE 或 ENDIF 之前的语句；如果没有找到标号，会产生错误信息。

举例: .IFPAGE0 label1

IFSSEQ

功能: 如果第一个字符串是第二个字符串的子串，则顺序执行

语法: .IFSSEQ n,string1,string2

注意:将第一个字符串与第二个字符串的第 n 个字符以后的部分相比较。如果相同，汇编继续执行；如果不同，则忽略下一个 ELSE 或 ENDIF 之前的所有语句。

n 指定了的是要比较的第二个字符串第几个字符以后的部分，1<n<第二个字符串长度。

举例: .IFSSEQ 4,‘car’,’The car is mine’

IFSSNEQ

功能: 如果第一个字符串不是第二个字符串的子串，则顺序执行

语法: .IFSSNEQ n,string1,string2

注意:将第一个字符串与第二个字符串的第 n 个字符以后的部分相比较。如果不相同，汇编继续执行；如果不同，则忽略下一个 ELSE 或 ENDIF 之前的所有语句。

n 指定了的是要比较的第二个字符串第几个字符以后的部分，1<n<第二个字符串长度。

举例: .IFSSNEQ 4,‘car’,’The car is mine’

24.3.5 宏

n 定义

宏是一系列的源代码行，可以使用一行源程序对其调用。必须在应用之前定义宏。在 pass 1 阶段，汇编器将宏的定义存储，当在某一行源程序中被调用时，该宏的所有代码便会映像到此行，作为一段源程序被执行。宏定义可以包含参数，在程序的任何地方都可以调用。空参数不包括空格。宏必须以指令 MACRO 作为起始，在结尾处用 ENDM (MACEND)结束。定义后的宏名被纳入标号域。

n 宏的调用

在宏调用时，参数类型有：直接型、间接型、字符串型或寄存器型。只有用双引号标注的 ASCII 字符串才可以包含空格。虚参数也可以作为参数被传递到宏中，即使宏处于嵌套中。只要物理存储空间足够，嵌套层数可以任意大。参数之间用逗号分隔。开头的空格和 tab 键被忽略。如果只有一个逗号，则代表参数缺省。

n 参数分隔符

在宏中，参数分隔符如下：

, { }

举例:

```
Math_Add:    .MACRO var1,var2,sum
        LDA var1
        CLC
        ADC var2
        STA sum
.ENDM

; 调用宏：
Math_Add S1,S2,S3
        Math_Add S1,{Speech_Table,X},S3
Math_Add S1,{Speech_Table,Y},S3
; 产生如下结果：
Math_Add S1,S2,S3
  LDA S1
  CLC
  ADC S2
  STA    S3
Math_Add S1,{Speech_Table,X},S3
  LDA S1
  CLC
  ADC Speech_Table,X
  STA    S3
Math_Add S1,{Speech_Table,Y},S3
  LDA S1
  CLC
  ADC Speech_Table,Y
  STA    S3
```

n 默认标号(Implicit Labels)

宏定义中可以包含外部的(用户定义)或隐含的(自动定义)标号。汇编器不会改变用户定义的外部标号。在外部标号后加一个(#)后缀，就会将其变为隐含标号，并通知汇编器自动将#替换为(_)和 6-digit 扩展数字。标号和它的扩展数字不能超过 32 个字符。

```
mac1:          .MACRO          arg1,arg2,arg3
                arg1
    var#:          .DB          arg2
    wed#:          .DB          arg3
               .ENDM
; 调用宏：
               mac1             RTS,19,”House”
; 产生如下结果：
```

```
                RTS
 var1_1568:        .DB              19
wed1_1568:      .DB              “House”
```

n 字符串连结

竖杠(“|”) (hex 7C)是字符串连结操作符。只能在宏的内部做字符串的连结。

n 数字连结

可以用竖杠(“|”)将一个字符串和数字连结起来(连结符之间不能有空格)，数字用方括号括起来。

```
mac2     .MACRO     arg
value:          .VAR          value+1
arg|<value>     .EQU          1
          .ENDM
```

引用宏前要将数值初始化，避免标号在 pass 1 和 pass 2 上的值不同。

```
value:          .VAR          0
```

宏的调用：

```
          mac2          label
```

扩展宏如下：

```
label1:         .EQU          1
```

24.4 链接器

24.4.1 简述

链接器的作用是将汇编程序、文件、模块和段整合为一个文件。在链接器里决定了外部引用，地址重定位和修改列表，指定了运行地址和最终操作码。链接器生成各种通用格式的文件，这些文件可以在 RAM 上执行。在存储空间允许的情况下，链接的文件没有大小限制。

24.4.2 文件扩展名

下表列出各文件扩展名的具体含义：

.obj	链接器的输入文件
.lib	库文件
.tsk	可执行目标代码
.s37	Motorola 格式的 s37 文件
.hex	Intex Hex / Extended Intel Hex
.map	资源占用列表文件
.sym	符号表(Microtek/AD High Level)

24.4.3 载入/运行地址

可以在链接的时候指定一个链接段地址进行链接，但只能将这个段定义为引用。因此，尽管链接器包含链接段的所有信息，但是该链接段不会出现在输出文件中。段中的代码可以由不同的载入和运行地址被间接的连接，将 ROM 中的代码在运行周期内载入 RAM 运行。

24.4.4 搜索顺序

搜索库文件的全局标号顺序如下：
1.输入数据
2.库文件

24.4.5 地址重定位

进行地址重定位时，链接器将偏移量地址加到汇编器产生的地址上。如果在汇编文件中用 ORG 伪指令，重定位的地址即 ORG。汇编器中有一个向量表，包含程序中所用到的每一个标号。

1．指令前的标号即重定位

2．如果一个标号用伪指令 EQU 来定义，它的操作数类型决定了它是否可以重定位。如果参数值包含一个可重定位的标记，则该标号是可重定位的；如果参数值中不包括可重定位的标记，则这个标号是不可重定位的。但参数中不能包含超过一个以上的标记，否则汇编器会产生错误信息。如下例所示：

```
SIZE: .EQU     $10              ; 绝对定位
      LDASIZE                       ;将$10 送入寄存器 A

Label:
      LDA..
      NOP
SIZE: .EQU     Lable+10         ; 可重定位，在链接阶段，Label 才会被赋值
  LDASIZE                       ; 在编程阶段，SIZE 的值未知。
```

3．如果标号被定义在绝对地址模式下，那么是不可重定位的。

24.4.6 符号表输出格式

链接器可以输出 Microtek 或 ADHighLevel 符号表和 s37 格式、tsk 格式、及扩展的 Intel hex 格式文件。

24.4.7 链接器特性

可以通过交互命令、命令行和链接器描述文件启动链接器工作。载入的关系图、按字母排序的全局标号交叉引用列表以及所有的链接错误都可以保存在磁盘中。在链接器描述文件模式下，可以采用下列方式对任意文件进行链接。

自动堆栈段
每个文件的每段自动链接到之前具有相同名字的段的顶端。

间接链接
如果程序存储在 ROM 中，使用间接链接很有用。但是这些资料有初始值，在程序运行时会被改变。需要一个启动程序，将这些初始值从 ROM 搬移到 RAM 中。这些初始值一般存在 ROM 中，运行的时候会将其链接。
偏移量段
将段链接到指定地址。如果汇编文件中用 ORG 伪指令，重定位的地址即 ORG。

堆栈段
来自不同文件具有相同名字的段都会被链接成一个段。
在任何模式下，按 <Ctrl>C 都会终止链接退回到操作系统。

24.4.8 交互模式

运行链接器，键入 **xlink8** 并按回车
输入
会看到提示 Input Filename：
键入输入的名字并按回车(文件默认扩展名是**.obj**.)。链接器打开文件，又出现提示 **Input Filename:**
如果没有要输入的文件，直接按回车
输出
会看到提示 Output Filename:
键入输出文件名并按回车
如果只按了回车，那么链接器就会输出与第一个输出文件相同名字的文件。文件扩展名为用户指定。

库文件
下一个提示是 Library Filename:
键入库文件名并按回车。可以不输入扩展名，链接器会自动搜索扩展名为.lib 的文件。如无库文件，单击回车即可。

选项
链接器经常出现这样的提示选项：
Options (D, C, M, N, X, 3, E, H, Z, I <CR>= Default):
键入选项，按回车。这些选项可以设置载入的映像目标、代码的文件格式和符号的文件格式（如果使用“Z”选项，则在 tsk 文件里，未用到的区域填入“0”）。
偏移量
你会看到如下提示：
Enter Hex ROM Offset For “(section name)” in “(input file name)”:

Enter the offset, and press **<cr>**.

Enter Hex RAM Offset For “(section name)”:
为保证提示段载入地址与它的运行地址相同，需要按回车键。如果输入文件和库文件有多个段，链接器会为每个段的偏移量输出偏移量提示。

24.4.9 交互模式操作

选项域创建了一个载入映像图，指定了目的地址并选择了输出文件和符号表格式。
可选项如下：
Options (D, C, M, N, X, 3, E, H, Z, I <CR>= Default): (见下表所示的交互模式选项)
这些选项不区分大小写。默认的设置为：

载入映像图	**none**
符号&代码文件	processor’s defaults

- 一个载入的映像目标 (**D**).
- 一个符号的文件格式 (**C/ M/N**).
- 一个代码的文件格式 (**X/3/ E or H**).

如果在一个类型里输入多个选项，只有最后的那个是有效的

交互模式选项表	
选项	**描述**
-D	创建一个包含所有链接错误，全局符号表和载入映像图的文件。文件名与链接器输出文件名相同，扩展名为**.map**。
-C	创建一个 AD High Level 符号表
-M	创建一个 16 位地址的 Microtek 符号表
-N	创建一个 24 位地址的 Microtek 符号表
-X	生成纯二进制可执行输出文件
-3	生成 Motorola s37 格式的输出文件
-E	生成 extend Intel hex 格式的文件
-H	生成 Intel hex 格式的文件
-Z	Tsk 文件间隔由$00 填充
-I	生成 64K 字节纯二进制可执行输出文件。
<cr>	生成处理器默认的输出格式。

24.4.10 命令行模式

可以通过命令行启动链接器。命令行格式如下(加方括号的参数为可选项)

Xlink8 -c *file1* [-l*offset*] [file2 [...]] [-o*file*] [-L*file*] [...] [-*options*]

-c

在输入文件名之前输入，使链接器进入命令行模式。

file1

至少指定一个输入文件名。

-l

(小写 L)在输入文件的每个段的可选择的偏移量偏移量量之前加**-l** (Example:如果有 12 个段，每一个段的偏移量量之前都应该加**-l** 偏移量)，**Example:**“**–l-nnnn**”是一个有效的偏移量量设定。

file2

第二个输入文件。用同样的语法，可以链接任意多个文件和对应第一个输入文件的段偏移量。

-o

加在输出文件名之前。如果没有指定输出文件名，链接器会默认与输入文件名相同，其扩展名根据输出文件格式确定。

-L

(大写 L)加在每个包含在链接中的库文件之前。不需要指定文件扩展名，链接器会自动搜索

扩展名为.lib 的文件。

-

选项前缀(不加空格)。如果命令行没有指定选项，链接器会输出选项提示。

24.4.11 链接描述文件模式

链接描述文件模式是对链接器进行操作的主要方法，其中的关键字设置使得它比交互模式具有更高的灵活性。可以直接或间接的链接到用户指定的地址：定义段地址、以任意顺序指定选项并随意链接各段(但在上一个段还没有定位时不能对下一个段进行定位)

n 命令

关键字不区分大小写

命令	修改器
Version:	*[Pseudo-parameter]*
[Options:]	**tsk, map, adhighlevel, microtek, microtek24, m37, ehex, hex, zero, img**
Input: or file:	*Filename [, filename...]*
Output:	*Filename*
Library:	*Filename [, filename...]*
Locate:	**after** *section* **at** *address* **linkafter** *section* **linkat** *address* **stack** **common** **reference**

Version

它必须是文件中的第一个关键字

功能: 通知连接器使用 Linker Script file 模式.

语法: version: [pseudo-parameter]

举例: version: 1.1.5 dated 06/26/01

注释可以在 Version 命令之后或之前写，也可以在不同行写。但 Version 必须是文件中的第一个关键字。可以在 Version 后加入数字、日期或其它信息作为参考。

Options

功能: 控制着创建和定位载入映像、输出符号格式和代码文件。

语法: options: option [, option...]

举例: options: tsk, map

下表列出了可选参数的详细信息。

参数选项链接
Load Map

Map	创建**.map** 磁盘文件，包含了链接错误、段信息和载入映像
Symbol Table	
Adhighlevel	AD High Level 符号表
Microtek	Microtek 16 位地址符号表
Microtek24	Microtek 24 位地址符号表
Code File	
Tsk	可执行文件(纯二进制)
M37	Motorola 格式的 s37 文件
Ehex	扩展 Intex Hex 文件
Hex	Intel 格式的 Hex 文件
Zero	Tsk 文件中空格将会被填充为 0（必须使用 Tsk 选项）
Img	生成 64K-byte 的纯二进制的可执行输出文件

Input or File

为被链接的输入文件命名。默认扩展名是.obj，但也可以自定义。

语法: Input: “filename”[,” filename”...]

举例： Input: “mytest”, “yourtest”

File: “mytest.obj”, “yourtest.obj”

Output

功能: 为输出文件命名。处理器会自动加上默认的代码文件格式的扩展名，也可以自定义。

语法: output: “filename”

举例： output: “ourtest.tsk”

Library

功能: 为库文件命名。默认扩展名是.lib.

语法: library: “libfile”[,” libfile”...]

举例： library: “Cmacro.lib”, “Printfl.ib”

Locate

功能: 在文件中或库中定义段的地址，或者将该段定义为另一个段之后

语法: locate: section at address, after section name

[linkat address, linkafter section name]

[stack, common, reference]

举例： locate : section1 at 1234h

locate : section2 after section1

locate : section3 at 8000

locate : section4 at 9000

locate : section5 at 1000h linkafter section2 stack

locate : section6 after section4

说明: 使用 linkat 可以直接链接在指定的地址；或者使用 linkafter 链接在某一段之后。
* 无论在哪种模式下，默认的基地址都是十六进制，不能改变。

24.4.12 偏移量

在所有模式下，链接器都需要一个偏移量为每个文件的每个段进行链接。载入的地址即偏移量(假设没有 ORG 伪指令)。如果哪个段没有给出的偏移量地址，链接器会认为它的偏移量地址是$0000，并把该段放入$0000 上。如果所有第一个文件都没有给出偏移量地址，链接器会把第一段放入$0000，其它段首尾相接依次放入这个段之后。

SectionD
SectionC
SectionB
SectionA

ß 0000H

如果所有的第二个或之后的文件都没有给出偏移量，链接器自动以相同的名字将这些段从低向上连续放置。

File2/SectionD
File1/SectionD
File2/SectionC
File1/SectionC
File2/SectionB
File1/SectionA

ß $0000

24.4.13 间接链接

存储在 ROM 中的数据只有移到 RAM 中才能修改。间接链接能通过 ROM 中的一个加载地址，将数据移到 RAM 中，并将其链接到 RAM 的运行地址。

n 直接链接

用于 ROM 中的程序，包含了完整的查找表和常量，只读不可写。

n 间接链接

程序存储在 ROM 中，但是这些程序有初始值，在运行时会被改变，因此，需要将这些初始值从 ROM 中搬移到 RAM 中，在运行时会被间接链接。如下表所示：

Direct		**Indirect**	
Load	**Run-Time**	**Load**	**Run-Time**
(ROM) CODE DATA	(ROM) CODE DATA	(ROM) CODE DATA	(ROM) CODE
  (RAM)	 (RAM)	 (RAM)	DATA (RAM)

举例：

```
Start     .Section  .CODE
Label1:
          LDA       #10
```

在 link script file 中如下定义:

```
Locate  Start at $800 link at $70 ;  间接
          LDA       Label1          ; A ß $70
```

;BIN 代码起始段存在$800, 但是程序的入口在$70.

24.4.14 链接描述文件模式

1、在链接中至少第一段必须被直接链接，链接器可以基于此固定地址计算出其它段的链接地址。

2、间接链接的段如同直接链阶段一样按顺序存放。

3、可以将一个段用关键字 **linkat** 和 **linkafter** 进行间接链接。

24.4.15 符号表

两种选择：

1.AD High Level

2.MicroTek

24.4.16 可执行代码文件格式

n Image File

链接器选项 **X**

默认输出文件扩展名 **.tsk**

可执行代码文件即纯二进制操作数和操作码，默认的起始地址是 0000H。链接器将每个段和文件的结尾处 填充空格，下一个段和文件的开始地址是 0FFh(除非 zero 标志在 TSK 格式下已经使能)。还可以用 ORG 指令设置连续的段和文件的地址。

n Motorola S37

链接器选项： **3**

默认输出文件扩展名 **.s37**

n Intel Hex

链接器选项： **H**

默认输出文件扩展名 **.hex**

24.4.17 映像文件

n 内存映像

如果在命令行模式和交互模式下选择了“D”选项或者在链接器描述文件模式选择了“map”选项，链接器就会创建一个映像文件(即 *filename*.**map**)。映像文件中包含了每个链接文件的符号定义和交叉引用。SECTION SUMMARY 列出了目标文件中包含的所有段和他们的载入/运行时间地址(如果是间接链接段，会不同)。

链接器可以重定位列表文件。重定位后，符号地址/数值被修改，从而表示运行时间地址/数值。下面是一个典型的映像文件，后面是所有元素的描述。

Wednesday, June 20, 10:16:55, 2001

Sunplus 6502 Linker - Version 1.1.5

Global Symbol Name	Global Value	Global Filename
St2	B	Test2.obj
St1	100	Test2.obj
St3	101	Test2.obj
FSub1	72E	Test1.obj
FSub2	737	Test1.obj
FSt1	D	Test1.obj
F_DIV	742	DIV
F_MUL	74E	MUL
F_ADD	753	ADD

****

* S E C T I O N S U M M A R Y *

****

* Section Name	Start	End	Description
* PAGE0	00000000	00000011	*
* DATA	00000100	00000124	*
* CODE	00000600	00000759	*
* MySec1	00000000	0000000D	*
* MySec2	0000000E	0000001B	*

****

```
**********************************************************************************************************
*********
*                                    L O A D      M A P
 *
**********************************************************************************************************
*********
*       Section Name                 Starting Address     Ending Address       Size
 *
**********************************************************************************************************
*********
* Test2.obj
 *
*       PAGE0                        00000000             0000000C
0000000D      *
*       DATA                         00000100             0000011C
0000001D      *
*       CODE                         00000600             0000072D
0000012E      *
*       MySec1                       00000000             0000000D
0000000E      *
* Test1.obj
 *
*       CODE                         0000072E             00000741
00000014      *
*       DATA                         0000011D             00000124
00000008      *
*       PAGE0                        0000000D             0000000F
00000003      *
*       MySec2                       0000000E             0000001B
0000000E      *
* DIV
 *
*       CODE                         00000742             0000074D
0000000C      *
*       PAGE0                        00000010             00000011
00000002      *
* MUL
 *
*       CODE                         0000074E             00000752
00000005      *
* ADD
 *
```

```
*      CODE                              00000753              00000759
00000007      *
*************************************************************************************************
*********
                        Linker Output Filename : Test2.tsk
                        Disk Mapping Filename : Test2.map
                        Symbol Table FileName : Test2.sym
                                        Format : Microtek

                        Linker Errors :              0
                        Output Format :               tsk
```

24.5 库文件管理器

24.6 错误信息

24.6.1 编译错误

该列表包含了 XASM8 汇编器编译的错误信息。

A0000: syntax error
指令或表达式的格式错误

A0001: endm expected before end of file
缺少 ENDM。在程序中，MACRO 和 ENDM 配对出现，缺一不可。

A0002: unmatched '{'"
“{“配对不平衡

A0003: symbol already defined
标号已经被定义，不能重复定义。

A0004: macro name used in expression
在表达式中，宏的名称被做为标号或者变量名使用，这是不允许的。

A0005: section size over 64k
段的大小不能超过 64K。

A0006: this chip no support such instruction addressing mode
芯片不支持该指令格式，或者芯片型号选择错误。

A0007: branch too far

跳转范围过大，跳转的有效距离为–128 ~ +127 字节。

A0008: branch into different section
使用跳转指令时，不能从一个段调转到另一个段。

A0009: file not found
包含文件没有被找到。

A0010: macro definition inside another macro definition
不能在一个宏的内部定义另一个宏。

A0011: incorrect operand
指令缺少操作数。

A0012: macro name expected
定义的宏缺少名称。

A0013: undefined Symbol
不能引用没有定义的符号。

A0014: constant expression expected
缺少常量表达式。某些指令需要使用常量。

A0015: number too large
数据太大，超出范围。

A0016: endif expected before end of file
缺少 ENDIF。在程序中，IF 和 ENDIF 配对出现，缺一不可。

A0017: local symbol already defined
局部标号已经被定义，局部标号在全局标号内有效。

A0018: ifma should be used inside macro
指令 IFMA 只能在宏中使用。

A0019: section name expected
定义的用户段缺少段名。

A0020: incorrect section option
段的选项错误。

A0021: section name used in expression

在表达式中，段名不能做为一个标号使用。

A0022: add two relative values
不能将两个标号相加。

A0023: subtract two symbols of different sections
不能将处于不同段的符号进行相减。

A0024: subtract constant with relative value
不能用标号减去常量。
A0025: constant expected
缺少常量。某些指令需要使用常量。

A0026: divided by zero
除数不能为 0。

A0027: symbol already defined other type
该符号已经被定义，不能再次定义。

A0028: illegal forward reference
不能使用没有定义的变量。

A0029: global symbol expected
应该使用全局标号

A0030: operand too large
操作数太大，超出指令寻址范围。

A0031: address expected
缺少地址，伪指令.HIGH., .LOW., .HIGH8., .HIGH16. 和 .LOW16 可以做为一个地址使用。

A0032: incorrect operand
操作数错误

A0033: zero page address expected
超出零页地址。

A0034: bad use of relative address
一个表达式里最多只能包含一个标号。

A0035: can't push two-byte operand into one-byte place
不能将两个字节的操作数放到一个字节的空间里面。

A0037: can't export such kind of symbol
当使用 VAR 或 SET 定义一个标号后，不能使用 PUBLIC 去声明。

A0038: segment or offset operators should be the outmost operator on an expression
在表达式中，段或偏移量应该处于算式的最外面。

A0039: keyword used as section name
指令不能用作段名。

A0040: keyword used as symboll
指令不能用作标号。

A0041: too many sections defined
段定义的数目过多，不能超过 4096。

A0042: incorrect use of external symbol
外部标号使用错误。

A0043: directive DW expected
需要使用指令 DW 进行定义。当一个 16 位的地址值被当作 8 位进行操作时，发生错误。用户可以使用.LOW. (低 8 位地址)或.HIGH（高 8 位地址）做为前缀分别对 16 位地址的高低位进行操作。也可以使用指令 DW 定义一个 16 位的存储空间。

A0044: directive DB expected
This occurs when an 8-bit address is used in a 16-bit storage instruction.
需要使用指令 DB 定义。当一个 8 位的地址值被当作 16 位进行操作时，发生错误。

A0045: symbol address expected
在定义一个公共符号之前，该符号应该首先被定义为标号、变量或者结构变量。

A0046: incorrect conditional assembly
条件表达式错误。

A0047: redefine a macro
宏不能被重复定义。

A0048: zero or positive number expected
在一条指令里面不能使用负数，或者可能是正数超出了指令的范围。

A0049: can't change section inside procedure
过程必须在段里面被定义，在过程里面不能改变段的定义。

A0050: can't nest procedure
过程不能被嵌套定义。即：过程 A 和过程 B，定义过程 A 时，在其内部不能定义过程 B。

A0051: unexpected end of file before proc end
在定义一个过程的时候，缺少结束符。

A0052: proc and endp unmatched
PROC 和 ENDP 配对不平衡。一般而言，ENDP 的数目少于 PROC。

A0053: not defined a struct name
结构名称没有被定义。

A0054: error in define 'struct' structure
在定义结构体的时候不能使用指令。

A0055: the 'struct ' name are not match
结构体名前后不一致

A0056: redefine a struct
结构体不能重复定义。

A0057: error in define procedure
定义过程时发生错误。

A0058: error in define struct variable name
定义结构体的变量名时发生错误。

A0059: not define such a struct field
不能引用一个没有定义的结构区域。用户必须使用符号将逻辑运算符、关系运算符、和位操作分开。

A0060: not defined as a struct variable
将一个标号（非结构体变量）引用为结构体变量。

A0061: not defined struct variable name
结构体的变量名没有被定义。

A0062: 'end' used inside procedure
在过程里面，不能使用 END。

A0063: use address expression with directive float/double
不能将伪指令 FLOAT/DOUBLE 用于寻址表达式中。

A0064: error use directive float/double define string
不能使用伪指令 FLOAT/DOUBLE 定义字符串。

A0065: not defined procedure name
缺少过程名的定义。

A0066: incorrect parameter
指令中操作数错误。

A0067: struct and endst number not equal
在源文件中，STRUCT 和 ENDST 必须配对平衡。一般而言，ENDST 和 STRUCT 的数目相等。

A0068: can't define a local label field inside struct define
在一个结构体里面，不能定义局部标号区域。

A0069: allocate memory failed for macro expansion
扩展宏的存储器分配失败。

A0070: macro parameter length too long
宏参数长度过大。

A0071: value expected
缺少常量值。

A0072: hex and symbol are identical
符号名称和 16 进制数不明确。

A0073: macro level too deep
宏的嵌套引用不能超过 500 个字节。

A0074: operate two symbol of different section
用户不能同时使用不同段的两个标号。

A0075: not provide this command
该命令不支持。

A0076: local symbol can't be declared as public.
局部标号不能使用 PUBLIC 进行声明。

A0077: line size over 512 bytes.
行的长度不能超过 512 字节。

A0078: only support 'add' and 'sub' operation for external symbol.
外部标号只能进行加减运算。

A0079: nested conditional command unbalance detected.
当一个宏使用指令 MACEXIT 提前退出时，必须使用指令 IFCLEAR 进行配对平衡。

A0080: no section directive command.
在程序中，段必须被指定。

24.6.2 编译警告

A2001: incorrect dummy parameter
在宏的定义中，虚参有误。

A2002: incorrect real parameter
引用宏时，实参有错误，或者和虚参的类型不符。

A2003: keyword used as symbol
汇编指令和伪指令不能用作标号。

A2004: unexpected end of line
当引用字符串的时候，应该使用“ ”或‘ ’将其引起来。

A2005: unexpected macexit
MACEXIT 使用错误，不能在宏定义的外部使用 MACEXIT

A2006: unmatched endm
在源文件中，MACRO 和 ENDM 配对不平衡。一般而言，二者数目相等。

A2007: pass dependence
不能在一个变量定义的前面去引用它。

A2008: operand could be too large
操作数过大，超出了指令范围。

A2009: unmatched endp
在源文件中，PROC 和 ENDP 配对不平衡。一般而言，二者数目相等。

A2010: unexpected else
ELSE 不能在条件汇编程序的外部使用。

A2011: unexpected endif
没有与 ENDIF 指令相匹配的 IF 指令

A2012: too much parameter
在宏的引用里面，实参数目过多，超过宏定义的虚参数目。

A2013: need an operate number or expression
存储区域声明需指定区域大小

A2014: symbol name and hex value are identical
符号名称和 16 进制数不明确。

A2015: zero page address expected
超出零页地址。

A2016: subtract constant with relative value
不能用相对值减去常量。

A2017: constant expected, expression mixed with relative value
需要使用常量

24.6.3 链接错误

L0001: Fatal Error: MFC initialization failed
重大错误：MFC 初始化失败。Xlink8 在当前平台上不能运行，当 MFC 库初始化时便会发生错误。

L0002: No object filename
缺少目标文件名。在命令行模式下，必须输入目标文件名。

L0003: Cannot identify the word…
xlink8 不能识别输入信息。

L0004: Cannot open the file…
指定的文件无法访问或者打开。

L0005: The library file … is not Sunplus Library
指定的文件不是凌阳格式的库文件。

L0006: No address after the word 'At'
当定位一个段时，“At’”后面必须接一个地址。

L0007: No section name after the word 'After'
当定位一个段时，“After’”后面必须接一个段名。

L0008: No section name after the word 'Linkafter'
当定位一个段时，“Linkafter’”后面必须接一个段名。

L0009: The section … has not been located
在 Linker Script file 模式下，段必须被定位，否则，它将会被定位在 00H。

L0010: The external symbol …has not a public definition
用户使用了一个局部标号，该标号在其它文件中被引用为外部标号，所以在该局部标号的定义文件中，需要将其声明为公共属性。

L0011: The section … has not been located before use the section …after it
在 Linker Script file 模式下，一个段在用于定位另一个段之前，必须已经被定位。

L0012: List file destroyed. Skip update the list file.
列表文件毁坏，不能更新。

L0013: The linkafter section … has not been defined at any obj file.
在 Linker Script file 模式下，用户引用的段在任何文件中都没有被定义。

L0014: No content in obj file.
目标文件中没有指令。

24.6.4 库文件生成器错误

L0500: MFC initialization failed
MFC 初始化失败。Xlink8 在当前平台上不能运行，当 MFC 库初始化时便会发生错误。

L0510: The second argument must be a lib file
第二个参数必须是凌阳格式的库文件，并且库文件的扩展名必须是‘.lib’。

L0511: No FIND argument
在命令行模式下，FIND 缺少参数。

L0512: No ADD argument
在命令行模式下，ADD 缺少参数。

L0513: No REP argument
在命令行模式下，REP 缺少参数。

L0514: No DEL argument
在命令行模式下，DEL 缺少参数。

L0515: Can't identify the command.
在命令行模式下，不能使用未知命令。

L0520: Read Lib Error
在读取指定的库文件时发生错误。

L0522: Can't find the module …
在库文件中找不到指定的模块。

L0541: Obj Type is not Sunplus !
指定的文件不是凌阳格式的 obj 文件。

24.7 常见错误

(1) 指令不完整

```
 .IF …
…
.ELSE IF
…
.ENDIF
```

在这段代码中，“ELSE IF”是不完整的指令

```
 .DB        C_Speech,S_19,
```

在这段代码中，包含了多余的逗号

(2) 注释前必须加分号(;)

错误的：

```
 LDA      #20       Setting IO
 LDA      GB_RTDataBank   =1000H-(ClockRate/24/1024/1024)
 DB       00,01,02,03,04,05,06,07,08          00,01
```

正确的：

```
 LDA      #20                    ; Setting IO
 LDA      GB_RTDataBank          ; =1000H-(ClockRate/24/1024/1024)
 DB       00,01,02,03,04,05,06,07,08      ; 00,01
```

(3) 在宏连接中，将表达式放在<>中是非法的。

```
 JSR       F_ServiceTask<_TASK_COUNT_-1>
```

应该这样书写：

```
 EXPR_VALUE       VAR      _TASK_COUNT_-1
 JSR       F_ServiceTask<EXPR_VALE>
```

(4) 用保留字作为符号名字是非法的

```
 LONG    EQU          1
```

```
 HIGH:    LDA      #11
```

(5) PAGE0 的 ORG 值超过 255 是非法的

```
.PAGE0
.....
.ORG 600H
```

(6) 非法的符号定义

```
 KEY_@:
LDA       #GRAD$
L_#Compare?:
```

(7) 字符串必须由单引号/双引号括起来。如果字符串中包括单引号这个字符，则必须要用双引号；如果字符串中包括双引号这个字符，则必须要用单引号。

```
 v_temp10: .db “XASM”S”  ; 非法
Rewrite as:
 .db 'XASM"S'
 Or   .db "XASM'S"
```

(8) 不支持 UNIX/MAC 文件格式。需要进行预处理才能识别。

(9) 存储器和累加器进行异或指令的指令是“EOR”，而不是“XOR”。

(10) Xasm8 除了减法或处于绝对地址模式(非可重定位模式)外，不支持包括两个或更多符号的逻辑或数学运算。

```
 LDA      Label1+Label2              ; 非法
 LDA      .LOW. (Label1 & Label2)    ; 非法
 LDA      Label1+10                  ; 正确
 LDA      Label1-10                  ; 正确
 LDA      100 - Label1               ; 正确
 LDA      Label2-Label1              ; 正确
```

(11) 数值超过 255 或者符号不在零页段却进行立即寻址都是非法的。

```
.PAGE0
 Var1:          .DS 1
.DATA
 Var2:          .DS 1
 Count:         EQU          156
 Price:         EQU          371
.CODE
 LDA      #Var1    ; 正确
 LDA      #Count   ; 正确
```

```
LDA     #Var2    ; 非法，应改为： LDA        #.LOW.Var2
LDA     #Price   ; 非法数值超过 255
```

(12). 如果标号后面没有冒号，那么必须从第一列开头。然而，如果指令设置放在第一列后，意欲作为标号，但编译器会自动识别指令。因此所有的保留字不能作为标号使用。

(13).不支持向前引用。符号必须在应用之前定义。

ex.

错误:

```
D_EndAddress:    EQU    (D_StartAddress+D_Segment*D_Common/4)
D_Segment:       EQU    160
D_Common:        EQU    D_Const
D_Const:         EQU    120
```

正确:

```
D_Const:         EQU    120
D_Segment:       EQU    160
D_Common:        EQU    D_Const
D_EndAddress     EQU    (D_StartAddress+D_Segment*D_Common/4)
```

(14). 不支持重复定义符号。除了“VAR”命令外，任何符号在一个文件中只能定义一次。

ex.

```
My_ADD .MACRO sum,p1,p2
LDA     p1
CLC
ADC     p2
STA     sum
.ENDM
My_ADD:
LDA     V1
CLC
ADC     V2
ADC     V3
 ...
```

(15).除了以下几种形式外，在复杂的表达式操作中不允许使用相对寻址标号。因为最终的地址不是不是编译阶段而是在链接阶段决定的。

Form 1: Label1-Label2

Form 2: Label1+immediate value

Form 3: Label1-immediate value

ex.

```
Tab1:
...
Tab2:
...
    .DW (Tab1*Tab2/300) ; 非法
    .DW (Tab1+Tab2)     ; 非法
    .DW (Tab1/2)        ; 非法
    .DW (Tab2-Tab1)     ; 非法
    .DW (Tab1+17)       ; 非法
    .DW (Tab2-9)        ; 非法
```

标号在绝对寻址模式下没有这样的限制，因为地址是在编译阶段指定的。

```
.Absolute
.ORG 600H
Tab1:
    ...
Tab2:
    ...
.DW (Tab1*Tab2/300) ; 非法
.DW (Tab1+Tab2)     ; 非法
.DW (Tab1/2)        ; 非法
.DW (Tab2-Tab1)     ; 非法
.DW (Tab1+17)       ; 非法
.DW (Tab2-9)        ; 非法
```

(16). 段或偏移量操作数应该是表达式最外层的操作数。

```
 ex.
 Row:       .EQU     12
 Column:    .EQU     5
 .DW .HIGH8.Label1+$10000        ; 非法
.DW .LOW.Label2+Row*Column       ; 非法
LDA .LOW.Label1+.HIGH.Label2     ; 非法
.DW .HIGH8.(Tab1+$1)             ; 非法
.DW .LOW.(Label2+Row*Column)     ; 非法
```

(17).局部标号只在规定的范围内有效。

ex.

```
%FG_PutStr:     .MACRO    STRPTR
%FP_SaveRegs
.IFSSEQ 1,'#',STRPTR
 jmp          ?stringgo#                ; 非法
string#:          .VAR        $
.DB       STRPTR,0
```

```
    ?stringgo#:
        mov2i     RG_StringAddr,string#
    exit#:
  .ENDIF
  ...
.ENDM
```

Label "?stringgo#" is only valid between "string#" with "exit#".

(18). Xlink8 不支持类似“group” 和 “range”这样的命令。

24.8 相关的应用例和库函数

为了帮助用户加快开发速度并提供相关方面的参考资料，凌阳公司开发了功能强大的库函数和应用例。其中库函数包括基本功能函数、数学运算函数以及数据通信等。以下列举了相关的的库函数和应用例：

应用例

标题	**应用例系列号**
本章无相关应用例	

库函数

标题	**应用例系列号**
SPMC65x 软件基本功能函数库说明书	AN_O0100.DOC
SPMC65x 数据处理函数库说明书	AN_O0101.DOC
SPMC65x 浮点运算函数库说明书	AN_O0102.DOC

24.9 修订记录

版本号	日期	备注
V0.1	03/31/2005	第一版

25 指令集

25.1 简述

本章意在使用户掌握指令集的用法。所有的指令均按字母顺序介绍。注意本章介绍的是65B02 指令集，它比通用 6502 指令集增加入了一些位操作指令，如 CLR, INV, SET 和 TST。

25.2 汇编指令格式

汇编指令包括四部分，格式如下：
[标号:] 操作码 [操作数] [; 注释]
[ ]中的内容为可选项

标号　　标号标识了一条指令的位置，可以使用标号作为访问该指令的地址，标号的格式应遵循以下规则：
- 必须从第一列开始或者以：结束
- 不能用操作码或寄存器的名字命名
- 标号长度为 1~32 个字符
- 避免使用特殊符号
- 以字母开头，注意：局部标号格式为：?L_xx　全局标号格式为：L_xx

操作码　　指令区域，用于写指令

操作数　　操作数可以是程序中的数据或地址。当操作码为单字节时，无操作数。当进行立即数寻址时，操作数为一个字节数据，使用一个符号来标识其所在位置。当操作数作为程序地址时，其实就是一个标号。

注释　　注释可以提高程序的可读性。在注释语句的前面应该加一个分号(;)。例如：
例如：

```
LDA    #00        ; 将数据 00 放入累加器 A
STA    Counter    ; 将累加器 A 中的数放入 Counter 中
```

注意：操作数和操作码之间需要加空格。

25.3 指令集

n ADC

累加器 A 带进位加法，指令操作：(A+M+C) **à** A, C

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Immediate	ADC #dd	69H	2	2
Zero Page	ADC aa	65H	2	3
Zero Page, X	ADC aa, X	75H	2	4
Absolute	ADC aaaa	6DH	3	4
Absolute, X	ADC aaaa, X	7DH	3	4
Absolute, Y	ADC aaaa, Y	79H	3	4
(Indirect, X)	ADC (aa, X)	61H	2	6
(Indirect), Y	ADC (aa), Y	71H	2	6

*如果数据寻址时超出了一页的范围，需要再加一个指令周期

N	V			D	I	Z	C
!	!			*	-	!	!

N：如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）

V：运算过程中发生溢出置 1

Z：结果为 0 置 1

C：最高位产生进位时，该位被置 1

D：*置 1 表示选择十进制加法模式

n AND

累加器 A 与数据进行“与”操作，指令操作：(A^M) **à** A

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
ImmediateImmediate	AND #dd	29H	2	2
Zero Page	AND aa	25H	2	3
Zero Page, X	AND aa, X	35H	2	4
Absolute	AND aaaa	2DH	3	4
Absolute, X	AND aaaa, X	3DH	3	4
Absolute, Y	AND aaaa, Y	39H	3	4
(Indirect, X)	AND (aa, X)	21H	2	6
(Indirect), Y	AND (aa), Y	31H	2	6

*如果数据寻址时超出了一页的范围，需要再加一个指令周期

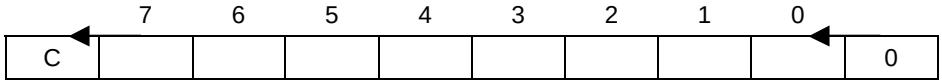
N	V			D	I	Z	C
!	-			-	-	!	-

N：如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）

Z：结果为 0 置 1

n ASL

算术左移



寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Accumulator	ASL A	0AH	1	2
Zero Page	ASL aa	06H	2	5
Zero Page, X	ASL aa, X	16H	2	6
Absolute	ASL aaaa	0EH	3	6
Absolute, X	ASL aaaa, X	1EH	3	6

*如果数据寻址时超出了一页的范围，需要再加一个指令周期

N	V			D	I	Z	C
!	-			-	-	!	!

N：如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）

Z：结果为 0 置 1

C:最高位 bit7 移出的值赋给 C

n BCC/BCS/BEQ/BMI/BNE/BPL/BVC/BVS

如果条件为真，指令跳转到 aa 执行

相对跳转范围：最大的向前跨度为 127 字节，向后跨度为 128 字节。

汇编指令格式	Condition	操作码	字节数	指令周期（Cycles）
BCC aa	C=0	90H	2	2*
BCS aa	C=1	B0H	2	2*
BEQ aa	Z=1	F0H	2	2*
BMI aa	N=1	30H	2	2*
BNE aa	Z=0	D0H	2	2*
BPL aa	N=0	10H	2	2*
BVC aa	V=0	50H	2	2*
BVS aa	V=1	70H	2	2*

*发生跳转时，需要再加一个指令周期

N	V			D	I	Z	C
-	-			-	-	-	-

n BIT

用累加器 A 进行位测试

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Zero Page	BIT aa	24H	2	3
Absolute	BIT aaaa	2CH	3	4

X: Not available.

N	V			D	I	Z	C
!	!			-	-	!	-

N:将 bit7 的值赋给“N”

V: 将 bit6 的值赋给“V”

Z：将变量 aa/aaaa 和累加器 A 相“与”，若结果为 0，则置标志“Z”为 1，否则置 0。

n CLC

清除进位标志

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	CLC	18H	1	2

N	V			D	I	Z	C
-	-			-	-	-	!

C: 无条件清除进位标志

n CLD

退出十进制运算模式

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	CLD	D8H	1	2

N	V			D	I	Z	C
-	-			!	-	-	-

D：无条件清除

n CLI

清除中断屏蔽位，打开总的中断开关

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	CLI	58H	1	2

N	V			D	I	Z	C

-	-			-	!	-	-

I: 无条件清除

- **CLR**

位清除。

将$aa 的 bit n 清除为“0”。

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Zero Page	CLR aa, 0	0FH	2	5
Zero Page	CLR aa, 1	1FH	2	5
Zero Page	CLR aa, 2	2FH	2	5
Zero Page	CLR aa, 3	3FH	2	5
Zero Page	CLR aa, 4	4FH	2	5
Zero Page	CLR aa, 5	5FH	2	5
Zero Page	CLR aa, 6	6FH	2	5
Zero Page	CLR aa, 7	7FH	2	5

N	V			D	I	Z	C
-	-			-	-	-	-

- **CLV**

清除溢出标志

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	CLV	B8H	1	2

N	V			D	I	Z	C
-	!			-	-	-	-

V: 无条件清除

- **CMP**

将数据与累加器 A 的值进行比较，指令操作：A - M

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Immediate	CMP #dd	C9H	2	2
Zero Page	CMP aa	C5H	2	3
Zero Page, X	CMP aa, X	D5H	2	4
Absolute	CMP aaaa	CDH	3	4
Absolute, X	CMP aaaa, X	DDH	3	4
Absolute, Y	CMP aaaa, Y	D9H	3	4
(Indirect, X)	CMP (aa, X)	C1H	2	6
(Indirect), Y	CMP (aa), Y	D1H	2	6

*如果数据寻址时超出了一页的范围，需要再加一个指令周期

N	V			D	I	Z	C
!	-			-	-	!	!

N：如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）

Z：结果为 0 置 1

C：A > = M，置 1；A< M，置 0

- **CPX**

将数据与 X 寄存器的值比较，指令操作：X - data

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Immediate	CPX #dd	E0H	2	2
Zero Page	CPX aa	E4H	2	3
Absolute	CPX aaaa	ECH	3	4

N	V			D	I	Z	C
!	-			-	-	!	!

- **CPY**

将数据与 Y 寄存器的值比较，指令操作：Y- data

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Immediate	CPY #dd	C0H	2	2
Zero Page	CPY aa	C4H	2	3
Absolute	CPY aaaa	CCH	3	4

N	V			D	I	Z	C
!	-			-	-	!	!

N：如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）

Z：结果为 0 置 1

C: Y > = data，置 1；Y < data，置 0

- **DEC**

减一指令

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Zero Page	DEC aa	C6H	2	5
Zero Page, X	DEC aa, X	D6H	2	6
Absolute	DEC aaaa	CEH	3	6
Absolute, X	DEC aaaa, X	DEH	3	6

N	V			D	I	Z	C
!	-			-	-	!	-

N：如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）

Z：结果为 0，Z 置 1

- **DEX**

X 寄存器减一指令

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	DEX	CAH	1	2

N	V			D	I	Z	C
!	-			-	-	!	-

N：如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）

Z：结果为 0 置 1

- **DEY**

Y 寄存器减一指令

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	DEY	88H	1	2

N	V			D	I	Z	C
!	-			-	-	!	-

N：如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）

Z：结果为 0 置 1

- **EOR**

异或指令，指令操作：A ß A XOR memory

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Immediate	EOR #dd	49H	2	2
Zero Page	EOR aa	45H	2	3
Zero Page, X	EOR aa, X	55H	2	4
Absolute	EOR aaaa	4DH	3	4
Absolute, X	EOR aaaa, X	5DH	3	4
Absolute, Y	EOR aaaa, Y	59H	3	4
(Indirect, X)	EOR (aa, X)	41H	2	6
(Indirect), Y	EOR (aa), Y	51H	2	6

*如果数据寻址时超出了一页的范围，需要再加一个指令周期

N	V			D	I	Z	C
!	-			-	-	!	-

N：如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）

Z：结果为 0 置 1

- **INC**

加一指令

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Zero Page	INC aa	E6H	2	5
Zero Page, X	INC aa, X	F6H	2	6
Absolute	INC aaaa	EEH	3	6
Absolute, X	INC aaaa, X	FEH	3	6

N	V			D	I	Z	C
!	-			-	-	!	-

N：如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）

Z：结果为 0 置 1

- **INV**

取反指令

将$aa 的 bit n 取反

寻址方式	汇编指令格式	操作码	字节数	可执行的指令 & 指令周期（Cycles）
Zero Page	INV aa, 0	87H	2	5
Zero Page	INV aa, 1	97H	2	5
Zero Page	INV aa, 2	A7H	2	5
Zero Page	INV aa, 3	B7H	2	5
Zero Page	INV aa, 4	C7H	2	5
Zero Page	INV aa, 5	D7H	2	5
Zero Page	INV aa, 6	E7H	2	5
Zero Page	INV aa, 7	F7H	2	5

N	V			D	I	Z	C
-	-			-	-	-	-

- **INX**

X 寄存器加一指令

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	INX	E8H	1	2

N	V			D	I	Z	C
!	-			-	-	!	-

N：如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）
Z：结果为 0 置 1

- **INY**

Y 寄存器加一指令

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	INY	C8H	1	2

N	V			D	I	Z	C
!	-			-	-	!	-

N：如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）
Z：结果为 0 置 1

- **JMP**

跳转到指定地址

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Absolute	JMP aaaa	4CH	3	3
Indirect	JMP (aaaa)	6CH	3	5

N	V			D	I	Z	C
-	-			-	-	-	-

- **JSR**

调用子程序

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Absolute	JSR aaaa	20H	3	6

执行指令 JSR aaaa，调用子程序，在跳转之前，硬件自动将子程序的返回地址压入堆栈，然后再跳转到子程序处执行，子程序执行完毕，执行指令 RTS，硬件自动从堆栈中弹出返回地址后返回。

N	V			D	I	Z	C
-	-			-	-	-	-

- **LDA**

将数据送入累加器 A 中。指令操作：A ß data

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Immediate	LDA #dd	A9H	2	2

Zero Page	LDA aa	A5H	2	3
Zero Page, X	LDA aa, X	B5H	2	4
Absolute	LDA aaaa	ADH	3	4
Absolute, X	LDA aaaa, X	BDH	3	4
Absolute, Y	LDA aaaa, Y	B9H	3	4
(Indirect, X)	LDA (aa, X)	A1H	2	6
(Indirect), Y	LDA (aa), Y	B1H	2	6

* 如果数据寻址时超出了一页的范围，需要再加一个指令周期

N	V			D	I	Z	C
!	-			-	-	!	-

N：如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）
Z：结果为 0 置 1

- **LDX**

将数据送入 X 寄存器中。指令操作：X ß data

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Immediate	LDX #dd	A2H	2	2
Zero Page	LDX aa	A6H	2	3
Zero Page, Y	LDX aa, Y	B6H	2	4
Absolute	LDX aaaa	AEH	3	4
Absolute, Y	LDX aaaa, Y	BEH	3	4

* 如果数据寻址时超出了一页的范围，需要再加一个指令周期

N	V			D	I	Z	C
!	-			-	-	!	-

N：如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）
Z：结果为 0 置 1

- **LDY**

将数据传送到 Y 寄存器中。指令操作：Y ß data

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Immediate	LDY #dd	A0H	2	2
Zero Page	LDY aa	A4H	2	3
Zero Page, X	LDY aa, X	B4H	2	4
Absolute	LDY aaaa	ACH	3	4
Absolute, X	LDY aaaa, X	BCH	3	4

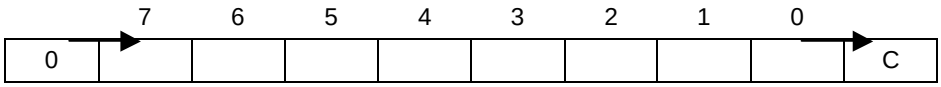
*如果数据寻址时超出了一页的范围，需要再加一个指令周期

N	V			D	I	Z	C
!	-			-	-	!	-

N：如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）
Z：结果为 0 置 1

- **LSR**

逻辑右移



寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Accumulator	LSR A	4AH	1	2
Zero Page	LSR aa	46H	2	5
Zero Page, X	LSR aa, X	56H	2	6
Absolute	LSR aaaa	4EH	3	6
Absolute, X	LSR aaaa, X	5EH	3	6

*如果数据寻址时超出了一页的范围，需要再加一个指令周期

N	V			D	I	Z	C
!	-			-	-	!	!

N：如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）
Z：结果为 0 置 1
C:将最低位 bit0 移出的数据赋给 C

- **NOP**

空操作

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	NOP	EAH	1	2

N	V			D	I	Z	C
-	-			-	-	-	-

- **ORA**

数据与累加器 A 的值进行“或”操作，指令操作：A ß A OR memory

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Immediate	ORA #dd	09H	2	2
Zero Page	ORA aa	05H	2	3
Zero Page, X	ORA aa, X	15H	2	4
Absolute	ORA aaaa	0DH	3	4

Absolute, X	ORA aaaa, X	1DH	3	4
Absolute, Y	ORA aaaa, Y	19H	3	4
(Indirect, X)	ORA (aa, X)	01H	2	6
(Indirect), Y	ORA (aa), Y	11H	2	6

*如果数据寻址时超出了一页的范围，需要再加一个指令周期

N	V			D	I	Z	C
!	-			-	-	!	-

N：如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）
Z：结果为 0 置 1

- **PHA**

入栈指令，将累加器 A 的值压入堆栈

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	PHA	48H	1	3

N	V			D	I	Z	C
-	-			-	-	-	-

- **PHP**

将状态标志寄存器的值压入堆栈

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	PHP	08H	1	3

N	V			D	I	Z	C
-	-			-	-	-	-

- **PLA**

出栈指令，将堆栈内容弹出给 A

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	PLA	68H	1	4

N	V			D	I	Z	C
!	-			-	-	!	-

- **PLP**

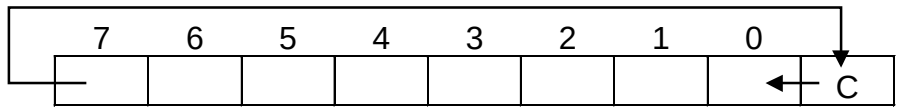
出栈指令，将堆栈内容弹出给状态寄存器 P

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	PLP	28H	1	4

N	V			D	I	Z	C
!	!			!	!	!	!

- **ROL**

循环左移



寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Accumulator	ROL A	2AH	1	2
Zero Page	ROL aa	26H	2	5
Zero Page, X	ROL aa, X	36H	2	6
Absolute	ROL aaaa	2EH	3	6
Absolute, X	ROL aaaa, X	3EH	3	6

*如果数据寻址时超出了一页的范围，需要再加一个指令周期

N	V			D	I	Z	C
!	-			-	-	!	!

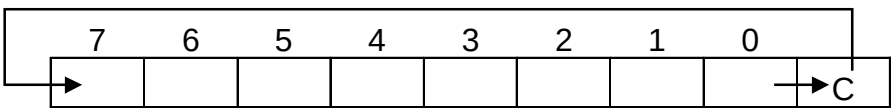
N：如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）

Z：结果为 0 置 1

C:将最高位移出的数据 bit7 赋给 C

- **ROR**

循环右移



寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Accumulator	ROR A	6AH	1	2
Zero Page	ROR aa	66H	2	5
Zero Page, X	ROR aa, X	76H	2	6
Absolute	ROR aaaa	6EH	3	6
Absolute, X	ROR aaaa, X	7EH	3	6

*如果数据寻址时超出了一页的范围，需要再加一个指令周期

N	V			D	I	Z	C
!	-			-	-	!	!

N：如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）

Z：结果为 0 置 1
C:将最低位移出的数据赋给 C

- **RTI**

中断返回指令

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	RTI	40H	1	6

N	V			D	I	Z	C

N、V、D、Z、C：执行指令 RTI 后，状态寄存器 P 的值从堆栈中弹出，将当前值覆盖，所以 N、V、D、Z、C 均恢复中断前的状态。

- **RTS**

子程序返回指令

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	RTS	60H	1	6

N	V			D	I	Z	C
-	-			-	-	-	-

- **SBC**

带借位的减法，指令操作：(A-M - $\overline{C}$) **à** A, C

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Immediate	SBC #dd	E9H	2	2
Zero Page	SBC aa	E5H	2	3
Zero Page, X	SBC aa, X	F5H	2	4
Absolute	SBC aaaa	EDH	3	4
Absolute, X	SBC aaaa, X	FDH	3	4
Absolute, Y	SBC aaaa, Y	F9H	3	4
(Indirect, X)	SBC (aa, X)	E1H	2	6
(Indirect), Y	SBC (aa), Y	F1H	2	6

*如果数据寻址时超出了一页的范围，需要再加一个指令周期

N	V			D	I	Z	C
!	!			*	-	!	!

N：如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）
V：运算过程中发生溢出置 1

Z：结果为 0 置 1
C:无借位，置 1(A >= M)；否则置 0
D:如果 D 置 1，带借位的减法以十进制方式操作

- **SEC**

进位标志置 1，指令操作：C ß1

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	SEC	38H	1	2

N	V			D	I	Z	C
-	-			-	-	-	!

C 无条件置 1

- **SED**

设置十进制运算模式，指令操作：D ß1

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	SED	F8H	1	2

N	V			D	I	Z	C
-	-			!	-	-	-

D：无条件置 1

- **SEI**

中断屏蔽位置 1，关闭总的中断开关，指令操作：I ß1

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	SEI	78H	1	2

N	V			D	I	Z	C
-	-			-	!	-	-

I：无条件置 1

- **SET**

位置 1
将$aa 的 bit n 位置 1。

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Zero Page	SET aa, 0	8FH	2	5
Zero Page	SET aa, 1	9FH	2	5
Zero Page	SET aa, 2	AFH	2	5

Zero Page	SET aa, 3	BFH	2	5
Zero Page	SET aa, 4	CFH	2	5
Zero Page	SET aa, 5	DFH	2	5
Zero Page	SET aa, 6	EFH	2	5
Zero Page	SET aa, 7	FFH	2	5

N	V			D	I	Z	C
-	-			-	-	-	-

- **STA**

将累加器 A 的值赋给存储器 M，指令操作：M ß A

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Zero Page	STA aa	85H	2	3
Zero Page, X	STA aa, X	95H	2	4
Absolute	STA aaaa	8DH	3	4
Absolute, X	STA aaaa, X	9DH	3	4
Absolute, Y	STA aaaa, Y	99H	3	4
(Indirect, X)	STA (aa, X)	81H	2	6
(Indirect), Y	STA (aa), Y	91H	2	6

N	V			D	I	Z	C
-	-			-	-	-	-

- **STX**

将寄存器 X 的值赋给存储器 M，指令操作：M ß X

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Zero Page	STX aa	86H	2	3
Zero Page, Y	STX aa, Y	96H	2	4
Absolute	STX aaaa	8EH	3	4

N	V			D	I	Z	C
-	-			-	-	-	-

- **STY**

将 Y 的值赋给存储器 M，指令操作：M ß Y

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Zero Page	STY aa	84H	2	3
Zero Page, X	STY aa, X	94H	2	4
Absolute	STY aaaa	8CH	3	4

N	V			D	I	Z	C
-	-			-	-	-	-

- **TAX**

将累加器 A 值送入 X 寄存器中，指令操作：X ß A

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	TAX	AAH	1	2

N	V			D	I	Z	C
!	-			-	-	!	-

N:如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）
Z:运算结果为零置 1

- **TAY**

将累加器 A 值送入 Y 寄存器中，指令操作：Y ß A

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	TAY	A8H	1	2

N	V			D	I	Z	C
!	-			-	-	!	-

N:如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）
Z: 运算结果为零置 1

- **TST**

位测试指令
判断$aa 的 bit n

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Zero Page	TST aa, 0	07H	2	3
Zero Page	TST aa, 1	17H	2	3
Zero Page	TST aa, 2	27H	2	3
Zero Page	TST aa, 3	37H	2	3
Zero Page	TST aa, 4	47H	2	3
Zero Page	TST aa, 5	57H	2	3
Zero Page	TST aa, 6	67H	2	3
Zero Page	TST aa, 7	77H	2	3

N	V			D	I	Z	C
-	-			-	-	!	-

Z:若$aa 的 bit n 为 0，则置标志位“z”为 1，否则置 0。

- **TSX**

将堆栈指针(SP)中的值送入 X 寄存器中，指令操作：X ß S。

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	TSX	BAH	1	2

N	V			D	I	Z	C
!	-			-	-	!	-

N:如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）
Z:运算结果为零置 1

- **TXA**

将 X 寄存器的值送入累加器 A 中，指令操作：A ßX。

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	TXA	8AH	1	2

N	V			D	I	Z	C
!	-			-	-	!	-

N:如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）
Z: 运算结果为零置 1

- **TXS**

将 X 寄存器的值传送到堆栈指针(SP)中，指令操作：S ß X

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	TXS	9AH	1	2

N	V			D	I	Z	C
-	-			-	-	-	-

- **TYA**

将 Y 寄存器的值传送到累加器 A 中，指令操作：A ß Y

寻址方式	汇编指令格式	操作码	字节数	指令周期（Cycles）
Implied	TYA	98H	1	2

N	V			D	I	Z	C
!	-			-	-	!	-

N:如果运算结果为负，该位置 1（即将运算结果的 bit7 的值赋给 N）
Z:运算结果为零置 1

25.4 指令简表

Assembly Form 指令格式	寻址方式 寻址方式	操作码 操作码	字节数 占用字节数	No. Cycles 执行时间 (Cycles)	标志							
					N	V			D	I	Z	C
ADC #dd	Immediate	69H	2	2	!	!			*	-	!	!
ADC aa	Zero Page	65H	2	3	!	!			*	-	!	!
ADC aa, X	Zero Page, X	75H	2	4	!	!			*	-	!	!
ADC aaaa	Absolute	6DH	3	4	!	!			*	-	!	!
ADC aaaa, X	Absolute, X	7DH	3	4	!	!			*	-	!	!
ADC aaaa, Y	Absolute, Y	79H	3	4	!	!			*	-	!	!
ADC (aa, X)	(Indirect, X)	61H	2	6	!	!			*	-	!	!
ADC (aa), Y	(Indirect), Y	71H	2	6	!	!			*	-	!	!
AND #dd	Immediate	29H	2	2	!	-			-	-	!	-
AND aa	Zero Page	25H	2	3	!	-			-	-	!	-
AND aa, X	Zero Page, X	35H	2	4	!	-			-	-	!	-
AND aaaa	Absolute	2DH	3	4	!	-			-	-	!	-
AND aaaa, X	Absolute, X	3DH	3	4	!	-			-	-	!	-
AND aaaa, Y	Absolute, Y	39H	3	4	!	-			-	-	!	-
AND (aa, X)	(Indirect, X)	21H	2	6	!	-			-	-	!	-
AND (aa), Y	(Indirect), Y	31H	2	6	!	-			-	-	!	-
ASL A	Accumulator	0AH	1	2	!	-			-	-	!	!
ASL aa	Zero Page	06H	2	5	!	-			-	-	!	!
ASL aa, X	Zero Page, X	16H	2	6	!	-			-	-	!	!
ASL aaaa	Absolute	0EH	3	6	!	-			-	-	!	!
ASL aaaa, X	Absolute, X	1EH	3	6	!	-			-	-	!	!
BCC aa	Relative	90H	2	2*	-	-			-	-	-	-
BCS aa	Relative	B0H	2	2*	-	-			-	-	-	-
BEQ aa	Relative	F0H	2	2*	-	-			-	-	-	-
BMI aa	Relative	30H	2	2*	-	-			-	-	-	-
BNE aa	Relative	D0H	2	2*	-	-			-	-	-	-
BPL aa	Relative	10H	2	2*	-	-			-	-	-	-
BVC aa	Relative	50H	2	2*	-	-			-	-	-	-
BVS aa	Relative	70H	2	2*	-	-			-	-	-	-
BIT aa	Zero Page	24H	2	3	!	!			-	-	!	-
BIT aaaa	Absolute	2CH	3	4	!	!			-	-	!	-
CLC	Implied	18H	1	2	-	-			-	-	-	!
CLD	Implied	D8H	1	2	-	-			!	-	-	-
CLI	Implied	58H	1	2	-	-			-	!	-	-
CLR aa, 0	Zero Page	0FH	2	5	-	-			-	-	-	-
CLR aa, 1	Zero Page	1FH	2	5	-	-			-	-	-	-
CLR aa, 2	Zero Page	2FH	2	5	-	-			-	-	-	-
CLR aa, 3	Zero Page	3FH	2	5	-	-			-	-	-	-

CLR aa, 4	Zero Page	4FH	2	5	-	-			-	-	-	-
CLR aa, 5	Zero Page	5FH	2	5	-	-			-	-	-	-
CLR aa, 6	Zero Page	6FH	2	5	-	-			-	-	-	-
CLR aa, 7	Zero Page	7FH	2	5	-	-			-	-	-	-
CLV	Implied	B8H	1	2	-	!			-	-	-	-
CMP #dd	Immediate	C9H	2	2	!	-			-	-	!	!
CMP aa	Zero Page	C5H	2	3	!	-			-	-	!	!
CMP aa, X	Zero Page, X	D5H	2	4	!	-			-	-	!	!
CMP aaaa	Absolute	CDH	3	4	!	-			-	-	!	!
CMP aaaa, X	Absolute, X	DDH	3	4	!	-			-	-	!	!
CMP aaaa, Y	Absolute, Y	D9H	3	4	!	-			-	-	!	!
CMP (aa, X)	(Indirect, X)	C1H	2	6	!	-			-	-	!	!
CMP (aa), Y	(Indirect), Y	D1H	2	6	!	-			-	-	!	!
CPX #dd	Immediate	E0H	2	2	!	-			-	-	!	!
CPX aa	Zero Page	E4H	2	3	!	-			-	-	!	!
CPX aaaa	Absolute	ECH	3	4	!	-			-	-	!	!
CPY #dd	Immediate	C0H	2	2	!	-			-	-	!	!
CPY aa	Zero Page	C4H	2	3	!	-			-	-	!	!
CPY aaaa	Absolute	CCH	3	4	!	-			-	-	!	!
DEC aa	Zero Page	C6H	2	5	!	-			-	-	!	-
DEC aa, X	Zero Page, X	D6H	2	6	!	-			-	-	!	-
DEC aaaa	Absolute	CEH	3	6	!	-			-	-	!	-
DEC aaaa, X	Absolute, X	DEH	3	6	!	-			-	-	!	-
DEX	Implied	CAH	1	2	!	-			-	-	!	-
DEY	Implied	88H	1	2	!	-			-	-	!	-
EOR #dd	Immediate	49H	2	2	!	-			-	-	!	-
EOR aa	Zero Page	45H	2	3	!	-			-	-	!	-
EOR aa, X	Zero Page, X	55H	2	4	!	-			-	-	!	-
EOR aaaa	Absolute	4DH	3	4	!	-			-	-	!	-
EOR aaaa, X	Absolute, X	5DH	3	4	!	-			-	-	!	-
EOR aaaa, Y	Absolute, Y	59H	3	4	!	-			-	-	!	-
EOR (aa, X)	(Indirect, X)	41H	2	6	!	-			-	-	!	-
EOR (aa), Y	(Indirect), Y	51H	2	6	!	-			-	-	!	-
INC aa	Zero Page	E6H	2	5	!	-			-	-	!	-
INC aa, X	Zero Page, X	F6H	2	6	!	-			-	-	!	-
INC aaaa	Absolute	EEH	3	6	!	-			-	-	!	-
INC aaaa, X	Absolute, X	FEH	3	6	!	-			-	-	!	-
INV aa, 0	Zero Page	87H	2	5	-	-			-	-	-	-
INV aa, 1	Zero Page	97H	2	5	-	-			-	-	-	-
INV aa, 2	Zero Page	A7H	2	5	-	-			-	-	-	-
INV aa, 3	Zero Page	B7H	2	5	-	-			-	-	-	-

INV aa, 4	Zero Page	C7H	2	5	-	-			-	-	-	-
INV aa, 5	Zero Page	D7H	2	5	-	-			-	-	-	-
INV aa, 6	Zero Page	E7H	2	5	-	-			-	-	-	-
INV aa, 7	Zero Page	F7H	2	5	-	-			-	-	-	-
INX	Implied	E8H	1	2	!	-			-	-	!	-
INY	Implied	C8H	1	2	!	-			-	-	!	-
JMP aaaa	Absolute	4CH	3	3	-	-			-	-	-	-
JMP (aaaa)	Indirect	6CH	3	5	-	-			-	-	-	-
JSR aaaa	Absolute	20H	3	6	-	-			-	-	-	-
LDA #dd	Immediate	A9H	2	2	!	-			-	-	!	-
LDA aa	Zero Page	A5H	2	3	!	-			-	-	!	-
LDA aa, X	Zero Page, X	B5H	2	4	!	-			-	-	!	-
LDA aaaa	Absolute	ADH	3	4	!	-			-	-	!	-
LDA aaaa, X	Absolute, X	BDH	3	4	!	-			-	-	!	-
LDA aaaa, Y	Absolute, Y	B9H	3	4	!	-			-	-	!	-
LDA (aa, X)	(Indirect, X)	A1H	2	6	!	-			-	-	!	-
LDA (aa), Y	(Indirect), Y	B1H	2	6	!	-			-	-	!	-
LDX #dd	Immediate	A2H	2	2	!	-			-	-	!	-
LDX aa	Zero Page	A6H	2	3	!	-			-	-	!	-
LDX aa, Y	Zero Page, Y	B6H	2	4	!	-			-	-	!	-
LDX aaaa	Absolute	AEH	3	4	!	-			-	-	!	-
LDX aaaa, Y	Absolute, Y	BEH	3	4	!	-			-	-	!	-
LDY #dd	Immediate	A0H	2	2	!	-			-	-	!	-
LDY aa	Zero Page	A4H	2	3	!	-			-	-	!	-
LDY aa, X	Zero Page, X	B4H	2	4	!	-			-	-	!	-
LDY aaaa	Absolute	ACH	3	4	!	-			-	-	!	-
LDY aaaa, X	Absolute, X	BCH	3	4	!	-			-	-	!	-
LSR A	Accumulator	4AH	1	2	!	-			-	-	!	!
LSR aa	Zero Page	46H	2	5	!	-			-	-	!	!
LSR aa, X	Zero Page, X	56H	2	6	!	-			-	-	!	!
LSR aaaa	Absolute	4EH	3	6	!	-			-	-	!	!
LSR aaaa, X	Absolute, X	5EH	3	6	!	-			-	-	!	!
NOP	Implied	EAH	1	2	-	-			-	-	-	-
ORA #dd	Immediate	09H	2	2	!	-			-	-	!	-
ORA aa	Zero Page	05H	2	3	!	-			-	-	!	-
ORA aa, X	Zero Page, X	15H	2	4	!	-			-	-	!	-
ORA aaaa	Absolute	0DH	3	4	!	-			-	-	!	-
ORA aaaa, X	Absolute, X	1DH	3	4	!	-			-	-	!	-
ORA aaaa, Y	Absolute, Y	19H	3	4	!	-			-	-	!	-
ORA (aa, X)	(Indirect, X)	01H	2	6	!	-			-	-	!	-
ORA (aa), Y	(Indirect), Y	11H	2	6	!	-			-	-	!	-

PHA	Implied	48H	1	3	-	-			-	-	-	-
PHP	Implied	08H	1	3	-	-			-	-	-	-
PLA	Implied	68H	1	4	!	-			-	-	!	-
PLP	Implied	28H	1	4	!	!			!	!	!	!
ROL A	Accumulator	2AH	1	2	!	-			-	-	!	!
ROL aa	Zero Page	26H	2	5	!	-			-	-	!	!
ROL aa, X	Zero Page, X	36H	2	6	!	-			-	-	!	!
ROL aaaa	Absolute	2EH	3	6	!	-			-	-	!	!
ROL aaaa, X	Absolute, X	3EH	3	6	!	-			-	-	!	!
ROR A	Accumulator	6AH	1	2	!	-			-	-	!	!
ROR aa	Zero Page	66H	2	5	!	-			-	-	!	!
ROR aa, X	Zero Page, X	76H	2	6	!	-			-	-	!	!
ROR aaaa	Absolute	6EH	3	6	!	-			-	-	!	!
ROR aaaa, X	Absolute, X	7EH	3	6	!	-			-	-	!	!
RTI	Implied	40H	1	6	-	-			-	-	-	-
RTS	Implied	60H	1	6	-	-			-	-	-	-
SBC #dd	Immediate	E9H	2	2	!	!			*	-	!	!
SBC aa	Zero Page	E5H	2	3	!	!			*	-	!	!
SBC aa, X	Zero Page, X	F5H	2	4	!	!			*	-	!	!
SBC aaaa	Absolute	EDH	3	4	!	!			*	-	!	!
SBC aaaa, X	Absolute, X	FDH	3	4	!	!			*	-	!	!
SBC aaaa, Y	Absolute, Y	F9H	3	4	!	!			*	-	!	!
SBC (aa, X)	(Indirect, X)	E1H	2	6	!	!			*	-	!	!
SBC (aa), Y	(Indirect), Y	F1H	2	6	!	!			*	-	!	!
SEC	Implied	38H	1	2	-	-			-	-	-	!
SED	Implied	F8H	1	2	-	-			!	-	-	-
SEI	Implied	78H	1	2	-	-			-	!	-	-
SET aa, 0	Zero Page	8FH	2	5	-	-			-	-	-	-
SET aa, 1	Zero Page	9FH	2	5	-	-			-	-	-	-
SET aa, 2	Zero Page	AFH	2	5	-	-			-	-	-	-
SET aa, 3	Zero Page	BFH	2	5	-	-			-	-	-	-
SET aa, 4	Zero Page	CFH	2	5	-	-			-	-	-	-
SET aa, 5	Zero Page	DFH	2	5	-	-			-	-	-	-
SET aa, 6	Zero Page	EFH	2	5	-	-			-	-	-	-
SET aa, 7	Zero Page	FFH	2	5	-	-			-	-	-	-
STA aa	Zero Page	85H	2	3	-	-			-	-	-	-
STA aa, X	Zero Page, X	95H	2	4	-	-			-	-	-	-
STA aaaa	Absolute	8DH	3	4	-	-			-	-	-	-
STA aaaa, X	Absolute, X	9DH	3	4	-	-			-	-	-	-
STA aaaa, Y	Absolute, Y	99H	3	4	-	-			-	-	-	-
STA (aa, X)	(Indirect, X)	81H	2	6	-	-			-	-	-	-

STA (aa), Y	(Indirect), Y	91H	2	6	-	-			-	-	-	-
STX aa	Zero Page	86H	2	3	-	-			-	-	-	-
STX aa, Y	Zero Page, Y	96H	2	4	-	-			-	-	-	-
STX aaaa	Absolute	8EH	3	4	-	-			-	-	-	-
STY aa	Zero Page	84H	2	3	-	-			-	-	-	-
STY aa, X	Zero Page, X	94H	2	4	-	-			-	-	-	-
STY aaaa	Absolute	8CH	3	4	-	-			-	-	-	-
TAX	Implied	AAH	1	2	!	-			-	-	!	-
TAY	Implied	A8H	1	2	!	-			-	-	!	-
TST aa, 0	Zero Page	07H	2	3	-	-			-	-	!	-
TST aa, 1	Zero Page	17H	2	3	-	-			-	-	!	-
TST aa, 2	Zero Page	27H	2	3	-	-			-	-	!	-
TST aa, 3	Zero Page	37H	2	3	-	-			-	-	!	-
TST aa, 4	Zero Page	47H	2	3	-	-			-	-	!	-
TST aa, 5	Zero Page	57H	2	3	-	-			-	-	!	-
TST aa, 6	Zero Page	67H	2	3	-	-			-	-	!	-
TST aa, 7	Zero Page	77H	2	3	-	-			-	-	!	-
TSX	Implied	BAH	1	2	!	-			-	-	!	-
TXA	Implied	8AH	1	2	!	-			-	-	!	-
TXS	Implied	9AH	1	2	-	-			-	-	-	-
TYA	Implied	98H	1	2	!	-			-	-	!	-

25.5 修订记录

版本号	日期	备注
V0.1	03/31/2005	第一版

26 Include File

26.1 描述

本章列出了 SPMC65X 系列所有的硬件寄存器的定义，地址从$00 到 $5F。对于 SPMC65X 系列的每一种芯片，它们都有自己的‘SPMC65PxxxxA.inc’文件，在该文件中，对此种芯片所有的硬件寄存器和相关常量进行了定义，其中使用 C_XXX_XXX 来代替具体的数字量做为常量，目的是为了增加程序的可读性，使程序更加规范。。当用户新建工程，选择了芯片类型之后，工程下面会自动生成‘SPMC65PxxxxA.inc’文件。用户可以参考前述的例子，以便迅速掌握。

26.2 Include 文件

```
;; ***************************************************************************
;; ***      Copyright 2005    Sunplus Technology Co., Ltd. All right reserved.     ***
;; ***    Header Name   : ECMC653.inc                                    ***
;; ***    Applied Body    : ECMC653A                                  ***
;; ***    Data                : 2005/07/04                                ***
;; ***    Revision          : V1.0.0A                                          ***
;; ***************************************************************************
;=============================================================================
;ECMC653 Input/Output Ports and Data Direction Registers
;=============================================================================
P_IOA_Data:         EQU         $00         ; Port A data b0~b7.(A)
P_IOB_Data:         EQU         $01         ; Port B data b0~b7.(A)
P_IOC_Data:         EQU         $02         ; Port C data b0~b7.(A)
P_IOD_Data:         EQU         $03         ; Port D data b0~b7.(A)
P_IOA_Dir:              EQU         $04          ; Port A direction control    b0~b7.(W), 0=In,
1=Out
P_IOB_Dir:              EQU         $05          ; Port B direction control    b0~b7.(W)
P_IOC_Dir:              EQU         $06          ; Port C direction control    b0~b7.(W)
P_IOD_Dir:              EQU         $07          ; Port D direction control    b0~b7.(W)
P_IOA_Attrib:           EQU         $08          ; Port A attribute register b0~b7.(W)
P_IOB_Attrib:           EQU         $09          ; Port B attribute register b0~b7.(W)
P_IOC_Attrib:           EQU         $0A          ; Port C attribute register b0~b7.(W)
P_IOD_Attrib:           EQU         $0B          ; Port D attribute register b0~b7.(W)
;
;-----------------------------------------------------------------------------------------------
P_INT_Flag0:            EQU         $0C          ; Interrupt Flag 0.(A)
    C_INT_ADIF:         EQU         %10000000        ; A/D INT flag bit.(A)
    C_INT_WDIF:         EQU         %01000000        ; WDT INT flag bit.(A)
    C_INT_IRQ5IF:       EQU         %00100000          IRQ5 INT flag bit.(A)
    C_INT_IRQ4IF:       EQU         %00010000        ; IRQ4 INT flag bit.(A)
    C_INT_IRQ3IF:       EQU         %00001000        ; IRQ3 INT flag bit.(A)
```

```
    C_INT_IRQ2IF:       EQU         %00000100       ; IRQ2 INT flag bit.(A)
    C_INT_IRQ1IF:       EQU         %00000010       ; IRQ1 INT flag bit.(A)
    C_INT_IRQ0IF:       EQU         %00000001       ; IRQ0 INT flag bit.(A)
    C_INT_CAP5IF:       EQU         %00100000       ; CAP5 INT flag bit.(A)
    C_INT_CAP4IF:       EQU         %00010000       ; CAP4 INT flag bit.(A)
    C_INT_CAP3IF:       EQU         %00000010       ; CAP3 INT flag bit.(A)
    C_INT_CAP2IF:       EQU         %00000001       ; CAP2 INT flag bit.(A)

    CB_INT_ADIF:        EQU         7           ; A/D INT flag bit for bit mode.(A)
    CB_INT_WDIF:        EQU         6           ; WDT INT flag bit for bit mode.(A)
    CB_INT_IRQ5IF:      EQU         5           ; IRQ5 INT flag bit for bit mode.(A)
    CB_INT_IRQ4IF:      EQU         4           ; IRQ4 INT flag bit for bit mode.(A)
    CB_INT_IRQ3IF:      EQU         3           ; IRQ3 INT flag bit for bit mode.(A)
    CB_INT_IRQ2IF:      EQU         2           ; IRQ2 INT flag bit for bit mode.(A)
    CB_INT_IRQ1IF:      EQU         1           ; IRQ1 INT flag bit for bit mode.(A)
    CB_INT_IRQ0IF:      EQU         0           ; IRQ0 INT flag bit for bit mode.(A)
    CB_INT_CAP5IF:      EQU         5           ; CAP5 INT flag bit for bit mode.(A)
    CB_INT_CAP4IF:      EQU         4           ; CAP4 INT flag bit for bit mode.(A)
    CB_INT_CAP3IF:      EQU         1           ; CAP3 INT flag bit for bit mode.(A)
    CB_INT_CAP2IF:      EQU         0           ; CAP2 INT flag bit for bit mode.(A)
;
P_INT_Ctrl0:        EQU         $0D         ; Interrupt control 0.(A)
    C_INT_ADIE:         EQU         %10000000       ; A/D INT enable bit.(A)
    C_INT_WDIE:         EQU         %01000000       ; WDT INT enable bit.(A)
    C_INT_IRQ5IE:       EQU         %00100000       ; IRQ5 INT enable bit.(A)
    C_INT_IRQ4IE:       EQU         %00010000       ; IRQ4 INT enable bit.(A)
    C_INT_IRQ3IE:       EQU         %00001000       ; IRQ3 INT enable bit.(A)
    C_INT_IRQ2IE:       EQU         %00000100       ; IRQ2 INT enable bit.(A)
    C_INT_IRQ1IE:       EQU         %00000010       ; IRQ1 INT enable bit.(A)
    C_INT_IRQ0IE:       EQU         %00000001       ; IRQ0 INT enable bit.(A)
    C_INT_CAP5IE:       EQU         %00100000       ; CAP5 INT enable bit.(A)
    C_INT_CAP4IE:       EQU         %00010000       ; CAP4 INT enable bit.(A)
    C_INT_CAP3IE:       EQU         %00000010       ; CAP3 INT enable bit.(A)
    C_INT_CAP2IE:       EQU         %00000001       ; CAP2 INT enable bit.(A)

    CB_INT_ADIE:        EQU         7           ; A/D INT enable bit for bit mode.(A)
    CB_INT_WDIE:        EQU         6           ; WDT INT enable bit for bit mode.(A)
    CB_INT_IRQ5IE:      EQU         5           ; IRQ5 INT enable bit for bit mode.(A)
    CB_INT_IRQ4IE:      EQU         4           ; IRQ4 INT enable bit for bit mode.(A)
    CB_INT_IRQ3IE:      EQU         3           ; IRQ3 INT enable bit for bit mode.(A)
    CB_INT_IRQ2IE:      EQU         2           ; IRQ2 INT enable bit for bit mode.(A)
    CB_INT_IRQ1IE:      EQU         1           ; IRQ1 INT enable bit for bit mode.(A)
```

```
    CB_INT_IRQ0IE:      EQU         0           ; IRQ0 INT enable bit for bit mode.(A)
    CB_INT_CAP5IE:      EQU         5           ; CAP5 INT enable bit for bit mode.(A)
    CB_INT_CAP4IE:      EQU         4           ; CAP4 INT enable bit for bit mode.(A)
    CB_INT_CAP3IE:      EQU         1           ; CAP3 INT enable bit for bit mode.(A)
    CB_INT_CAP2IE:      EQU         0           ; CAP2 INT enable bit for bit mode.(A)
;
P_INT_Flag1:            EQU         $0E         ; Interrupt flag 1.
    C_INT_CAP1IF:       EQU         %10000000       ; CAP1 INT flag bit.(A)
    C_INT_CAP0IF:       EQU         %01000000       ; CAP0 INT flag bit.(A)
    C_INT_T5OIF:        EQU         %00100000       ; Timer5 overflow INT flag bit.(A)
    C_INT_T4OIF:        EQU         %00010000       ; Timer4 overflow INT flag bit.(A)
    C_INT_T3OIF:        EQU         %00001000       ; Timer3 overflow INT flag bit.(A)
    C_INT_T2OIF:        EQU         %00000100       ; Timer2 overflow INT flag bit.(A)
    C_INT_T1OIF:        EQU         %00000010       ; Timer1 overflow INT flag bit.(A)
    C_INT_T0OIF:        EQU         %00000001       ; Timer0 overflow INT flag bit.(A)

    CB_INT_CAP1IF:      EQU         7           ; CAP1 INT flag bit for bit mode.(A)
    CB_INT_CAP0IF:      EQU         6           ; CAP0 INT flag bit for bit mode.(A)
    CB_INT_T5OIF:       EQU         5           ; Timer5 overflow INT flag bit for bit
mode.(A)
    CB_INT_T4OIF:       EQU         4           ; Timer4 overflow INT flag bit for bit
mode.(A)
    CB_INT_T3OIF:       EQU         3           ; Timer3 overflow INT flag bit for bit
mode.(A)
    CB_INT_T2OIF:       EQU         2           ; Timer2 overflow INT flag bit for bit
mode.(A)
    CB_INT_T1OIF:       EQU         1           ; Timer1 overflow INT flag bit for bit
mode.(A)
    CB_INT_T0OIF:       EQU         0           ; Timer0 overflow INT flag bit for bit
mode.(A)
;
P_INT_Ctrl1:            EQU         $0F         ; Interrupt control 1.
    C_INT_CAP1IE:       EQU         %10000000       ; CAP1 INT enable bit.(A)
    C_INT_CAP0IE:       EQU         %01000000       ; CAP0 INT enable bit.(A)
    C_INT_T5OIE:        EQU         %00100000       ; Timer5 overflow INT enable bit.(A)
    C_INT_T4OIE:        EQU         %00010000       ; Timer4 overflow INT enable bit.(A)
    C_INT_T3OIE:        EQU         %00001000       ; Timer3 overflow INT enable bit.(A)
    C_INT_T2OIE:        EQU         %00000100       ; Timer2 overflow INT enable bit.(A)
    C_INT_T1OIE:        EQU         %00000010       ; Timer1 overflow INT enable bit.(A)
    C_INT_T0OIE:        EQU         %00000001       ; Timer0 overflow INT enable bit.(A)

    CB_INT_CAP1IE:      EQU         7           ; CAP1 INT enable bit for bit mode.(A)
```

```
    CB_INT_CAP0IE:    EQU    6          ; CAP0 INT enable bit for bit mode.(A)
    CB_INT_T5OIE:     EQU    5          ; Timer5 overflow INT enable bit for bit
mode.(A)
    CB_INT_T4OIE:     EQU    4          ; Timer4 overflow INT enable bit for bit
mode.(A)
    CB_INT_T3OIE:     EQU    3          ; Timer3 overflow INT enable bit for bit
mode.(A)
    CB_INT_T2OIE:     EQU    2          ; Timer2 overflow INT enable bit for bit
mode.(A)
    CB_INT_T1OIE:     EQU    1          ; Timer1 overflow INT enable bit for bit
mode.(A)
    CB_INT_T0OIE:     EQU    0          ; Timer0 overflow INT enable bit for bit
mode.(A)
;
P_INT_Flag2:          EQU    $26        ; Interrupt flag 2.
    C_INT_ITVALIF:    EQU    %00100000      ; Timer Base interrupt flag.
    C_INT_IICIF:      EQU    %00010000  ; IIC interrupt flag.
    C_INT_UARTIF:     EQU    %00001000      ; UART interrupt flag.
    C_INT_SPIIF:      EQU    %00000100      ; SPI interrupt flag.
    C_INT_CMP1IF:     EQU    %00000010      ; Comparator 1 interrupt flag.
    C_INT_CMP0IF:     EQU    %00000001      ; Comparator 0 interrupt flag.

    CB_INT_ITVALIF:   EQU    5          ; Timer Base interrupt flag for bit mode.
    CB_INT_IICIF:     EQU    4          ; IIC interrupt flag for bit mode.
    CB_INT_UARTIF:    EQU    3          ; UART interrupt flag for bit mode.
    CB_INT_SPIIF:     EQU    2          ; SPI interrupt flag for bit mode.
    CB_INT_CMP1IF:    EQU    1          ; Comparator 1 interrupt flag for bit mode.
    CB_INT_CMP0IF:    EQU    0          ; Comparator 0 interrupt flag for bit mode.
;
P_INT_Ctrl2:          EQU    $27        ; Interrupt control 2.
    C_INT_ITVALIE:    EQU    %00100000  ; Timer Base interrupt enable bit.
    C_INT_CMP1IE:     EQU    %00000010      ; Comparator 1 interrupt enable bit.
    C_INT_CMP0IE:     EQU    %00000001      ; Comparator 0 interrupt enable bit.

    CB_INT_ITVALIE:   EQU    5          ; Timer Base interrupt enable bit for bit
mode.
    CB_INT_CMP1IE:    EQU    1          ; Comparator 1 interrupt enable bit for bit
mode.
    CB_INT_CMP0IE:    EQU    0          ; Comparator 0 interrupt enable bit for bit
mode.
;
P_WDT_Clr:            EQU    $10        ; Watchdog clear register.(W), $55= clear
```

```
    C_WDT_Clr:      EQU         $55             ; Write '55' to clear this register.
;
;-------------------------------------------------------------------------------------------------
P_TMR0_1_Ctrl0:             EQU         $11         ; Timer0/1 control 0.
    C_T112B_PWM:    EQU         %01110000       ; Timer1 Function as 12 Bit PWM.
    C_T116B_CAP:    EQU         %01100000       ; Timer1 Function as 16 Bit
Capture(Width).
    C_T116B_COMP:   EQU         %01010000       ; Timer1 Function as 16 Bit Compare.
    C_T116B_Timer:      EQU         %01000000       ; Timer1 Function as 16 Bit Timer.
    C_T18B_CAP:     EQU         %00110000       ; Timer1 Function as  8  Bit
Capture(Width,Cycle).
    C_T18B_COMP:    EQU         %00100000       ; Timer1 Function as  8  Bit
Compare.
    C_T18B_Timer:   EQU         %00010000       ; Timer1 Function as  8 Bit Timer.
    C_T08B_PWM:     EQU         %00000111       ; Timer0 Function as  8 Bit PWM.
    C_T016B_CAP:    EQU         %00000110       ; Timer0 Function as 16 Bit
Capture(Width).
    C_T016B_COMP:   EQU         %00000101       ; Timer0 Function as 16 Bit Compare.
    C_T016B_Timer:  EQU         %00000100       ; Timer0 Function as 16 Bit Timer.
    C_T08B_CAP:     EQU         %00000011       ; Timer0 Function as  8 Bit
Capture(Width,Cycle).
    C_T08B_COMP:    EQU         %00000010       ; Timer0 Function as  8  Bit
Compare.
    C_T08B_Timer:   EQU         %00000001       ; Timer0 Function as  8 Bit Timer.

P_TMR0_1_Ctrl1:             EQU         $12         ; Timer0/1 control 1.
    C_T1EXT_EN:     EQU         %01110000       ; External Event
    C_T1FCS_Div_512:    EQU         %01100000       ; Timer1 Clock= FCS/512.
    C_T1FCS_Div_128:    EQU         %01010000       ; Timer1 Clock= FCS/128.
    C_T1FCS_Div_32:     EQU         %01000000       ; Timer1 Clock= FCS/32.
    C_T1FCS_Div_8:      EQU         %00110000       ; Timer1 Clock= FCS/8.
    C_T1FCS_Div_4:      EQU         %00100000       ; Timer1 Clock= FCS/4.
    C_T1FCS_Div_2:      EQU         %00010000       ; Timer1 Clock= FCS/2.
    C_T1FCS_Div_1:      EQU         %00000000       ; Timer1 Clock= FCS/1.
    C_T0EXT_EN:     EQU         %00000111       ; External Event
    C_T0FCS_Div_512:    EQU         %00000110       ; Timer0 Clock= FCS/512.
    C_T0FCS_Div_128:    EQU         %00000101       ; Timer0 Clock= FCS/128.
    C_T0FCS_Div_32:     EQU         %00000100       ; Timer0 Clock= FCS/32.
    C_T0FCS_Div_8:      EQU         %00000011       ; Timer0 Clock= FCS/8.
    C_T0FCS_Div_4:      EQU         %00000010       ; Timer0 Clock= FCS/4.
    C_T0FCS_Div_2:      EQU         %00000001       ; Timer0 Clock= FCS/2.
    C_T0FCS_Div_1:      EQU         %00000000       ; Timer0 Clock= FCS/1.
```

```
P_TMR0_Count:       EQU     $13     ; Timer0 8/16-bit counter register.(R)
P_TMR0_Preload:     EQU     $13     ; Timer0 8/16-bit preload register.(W)
P_TMR0_Comp:        EQU     $13     ; Timer0 8/16-bit compare low byte value.
P_TMR0_Cap:         EQU     $13     ; Timer0 8/16-bit capture low byte width
value.
P_TMR0_PWMPeriod:       EQU     $13     ; Timer0 8 bit PWM period value.
;
P_TMR0_CountHi:     EQU     $14     ; Timer0 16-bit counter register.(R)
P_TMR0_PreloadHi:   EQU     $14     ; Timer0 16-bit preload register.(W)
P_TMR0_CompHi:      EQU     $14     ; Timer0 16-bit compare high byte value.
P_TMR0_CapHi:       EQU     $14     ; Timer0 16-bit capture high byte width
value.
P_TMR0_CapCycle8:   EQU     $14     ; Timer0 8-bit capture cycle value.(R)
P_TMR0_PWMDuty:     EQU     $14     ; Timer0 8-bit PWM duty value.
;
P_TMR1_Count:       EQU     $15     ; Timer1 8/16-bit Counter register.(R)
P_TMR1_Preload:     EQU     $15     ; Timer1 8/16-bit preload register.(W)
P_TMR1_Comp:        EQU     $15     ; Timer1 8/16-bit compare low byte value.
P_TMR1_Cap:         EQU     $15     ; Timer1 8/16-bit capture low byte width
value.
P_TMR1_PWMPeriod:       EQU     $15     ; Timer1 12-bit PWM peroid low byte
register.
;
P_TMR1_CountHi:     EQU     $16     ; Timer1 16-bit Counter register.(R)
P_TMR1_PreloadHi:   EQU     $16     ; Timer1 16-bit preload register.(W)
P_TMR1_CompHi:      EQU     $16     ; Timer1 16-bit compare high byte value.
P_TMR1_CapHi:       EQU     $16     ; Timer1 16-bit capture high byte width
value.
P_TMR1_CapCycle8:   EQU     $16     ; Timer1 8-bit capture cycle value.(R)
P_TMR1_DutyPeriod:  EQU     $16     ; Timer1 12-bit PWM high byte register.
;
P_TMR1_PWMDuty:     EQU     $17     ; Timer1 12-bit PWM duty low byte register.
;
P_TMR2_3_Ctrl0:         EQU     $18     ; Timer2/3 control 0.
    C_T312B_PWM:    EQU     %01110000   ; Timer3 Function as 12 Bit PWM.
    C_T316B_CAP:    EQU     %01100000   ; Timer3 Function as 16 Bit
Capture(Width).
    C_T316B_COMP:   EQU     %01010000   ; Timer3 Function as 16 Bit Compare.
    C_T316B_Timer:  EQU     %01000000   ; Timer3 Function as 16 Bit Timer.
    C_T38B_CAP:     EQU     %00110000   ; Timer3 Function as  8  Bit
Capture(Width,Cycle).
```

```
    C_T38B_COMP:    EQU     %00100000    ; Timer3 Function as  8   Bit
Compare.
    C_T38B_Timer:   EQU     %00010000    ; Timer3 Function as  8 Bit Timer.
    C_T28B_PWM:     EQU     %00000111    ; Timer2 Function as  8 Bit PWM.
    C_T216B_CAP:    EQU     %00000110    ; Timer2 Function as 16 Bit
Capture(Width).
    C_T216B_COMP:   EQU     %00000101    ; Timer2 Function as 16 Bit Compare.
    C_T216B_Timer:  EQU     %00000100    ; Timer2 Function as 16 Bit Timer.
    C_T28B_CAP:     EQU     %00000011    ; Timer2 Function as  8 Bit
Capture(Width,Cycle).
    C_T28B_COMP:    EQU     %00000010    ; Timer2 Function as  8   Bit
Compare.
    C_T28B_Timer:       EQU     %00000001       ; Timer2 Function as  8 Bit Timer.
;
P_TMR2_3_Ctrl1:     EQU     $19         ; Timer2/3 control 1.
    C_T3EXT_EN:     EQU     %01110000   ; External Event.
    C_T3FCS_Div_512: EQU    %01100000   ; Timer3 Clock= FCS/512.
    C_T3FCS_Div_128: EQU    %01010000   ; Timer3 Clock= FCS/128.
    C_T3FCS_Div_32: EQU     %01000000   ; Timer3 Clock= FCS/32.
    C_T3FCS_Div_8:  EQU     %00110000   ; Timer3 Clock= FCS/8.
    C_T3FCS_Div_4:  EQU     %00100000   ; Timer3 Clock= FCS/4.
    C_T3FCS_Div_2:  EQU     %00010000   ; Timer3 Clock= FCS/2.
    C_T3FCS_Div_1:  EQU     %00000000   ; Timer3 Clock= FCS/1.
    C_T2EXT_EN:     EQU     %00000111   ; External Event.
    C_T2FCS_Div_512: EQU    %00000110   ; Timer2 Clock= FCS/512.
    C_T2FCS_Div_128: EQU    %00000101   ; Timer2 Clock= FCS/128.
    C_T2FCS_Div_32: EQU     %00000100   ; Timer2 Clock= FCS/32.
    C_T2FCS_Div_8:  EQU     %00000011   ; Timer2 Clock= FCS/8.
    C_T2FCS_Div_4:  EQU     %00000010   ; Timer2 Clock= FCS/4.
    C_T2FCS_Div_2:  EQU     %00000001   ; Timer2 Clock= FCS/2.
    C_T2FCS_Div_1:  EQU     %00000000   ; Timer2 Clock= FCS/1.
;
P_TMR2_Count:       EQU     $1A         ; Timer2 8/16-bit counter register.(R)
P_TMR2_Preload:     EQU     $1A         ; Timer2 8/16-bit preload register.(W)
P_TMR2_Comp:        EQU     $1A         ; Timer2 8/16-bit compare low byte value.
P_TMR2_Cap:         EQU     $1A         ; Timer2 8/16-bit capture low byte width
value.
P_TMR2_PWMPeriod:       EQU     $1A         ; Timer2 8-bit PWM period value.
;
P_TMR2_CountHi:     EQU     $1B         ; Timer2 16-bit counter register.(R)
P_TMR2_PreloadHi:   EQU     $1B         ; Timer2 16-bit preload register.(W)
P_TMR2_CompHi:      EQU     $1B         ; Timer2 16-bit compare high byte value.
```

```
P_TMR2_CapHi:       EQU     $1B     ; Timer2 16-bit capture high byte width
value.
P_TMR2_CapCycle8:   EQU     $1B     ; Timer2 8-bit capture cycle value.(R)
P_TMR2_PWMDuty:     EQU     $1B     ; Timer2 8-bit PWM duty value.
;
P_TMR3_Count:       EQU     $1C     ; Timer3 8/16-bit Counter register.(R)
P_TMR3_Preload:     EQU     $1C     ; Timer3 8/16-bit preload register.(W)
P_TMR3_Comp:        EQU     $1C     ; Timer3 8/16-bit compare low byte value.
P_TMR3_Cap:         EQU     $1C     ; Timer3 8-bit capture low byte width value.
P_TMR3_PWMPeriod:       EQU     $1C     ; Timer3 12-bit PWM peroid low byte
register.
;
P_TMR3_CountHi:     EQU     $1D     ; Timer3 16-bit Counter register.(R)
P_TMR3_PreloadHi:   EQU     $1D     ; Timer3 16-bit preload register.(W)
P_TMR3_CompHi:      EQU     $1D     ; Timer3 16-bit compare high byte value.
P_TMR3_CapHi:       EQU     $1D     ; Timer3 16-bit capture high byte width
value.
P_TMR3_CapCycle8:   EQU     $1D     ; Timer3 8-bit capture cycle value.(R)
P_TMR3_DutyPeriod:  EQU     $1D     ; Timer3 12-bit PWM high byte register.
;
P_TMR3_PWMDuty:     EQU     $1E     ; Timer3 12-bit PWM duty low byte register.
;
P_TMR4_5_Ctrl0:         EQU     $1F     ; Timer4/5 control 0.
    C_T516B_PWM:    EQU     %01110000   ; Timer5 Function as 16 Bit PWM.
    C_T516B_CAP:    EQU     %01100000   ; Timer5 Function as 16 Bit
Capture(Width,Cycle).
    C_T516B_COMP:   EQU     %01010000   ; Timer5 Function as 16 Bit Compare.
    C_T516B_Timer:  EQU     %01000000   ; Timer5 Function as 16 Bit Timer.
    C_T58B_CAP:     EQU     %00110000   ; Timer5 Function as  8  Bit
Capture(Width,Cycle).
    C_T58B_COMP:    EQU     %00100000   ; Timer5 Function as  8  Bit
Compare.
    C_T58B_Timer:   EQU     %00010000   ; Timer5 Function as  8 Bit Timer.
    C_T48B_PWM:     EQU     %00000111   ; Timer4 Function as  8 Bit PWM.
    C_T416B_CAP:    EQU     %00000110   ; Timer4 Function as 16 Bit
Capture(Width).
    C_T416B_COMP:   EQU     %00000101   ; Timer4 Function as 16 Bit Compare.
    C_T416B_Timer:  EQU     %00000100   ; Timer4 Function as 16 Bit Timer.
    C_T48B_CAP:     EQU     %00000011   ; Timer4 Function as  8 Bit
Capture(Width,Cycle).
    C_T48B_COMP:    EQU     %00000010   ; Timer4 Function as  8  Bit
Compare.
```

```
    C_T48B_Timer:       EQU         %00000001       ; Timer4 Function as    8 Bit Timer.
;
P_TMR4_5_Ctrl1:             EQU         $20         ; Timer4/5 control 1.
    C_T5EXT_EN:         EQU         %01110000       ; External Event.
    C_T5FCS_Div_512:    EQU         %01100000       ; Timer5 Clock= FCS/512.
    C_T5FCS_Div_128:    EQU         %01010000       ; Timer5 Clock= FCS/128.
    C_T5FCS_Div_32:     EQU         %01000000       ; Timer5 Clock= FCS/32.
    C_T5FCS_Div_8:      EQU         %00110000       ; Timer5 Clock= FCS/8.
    C_T5FCS_Div_4:      EQU         %00100000       ; Timer5 Clock= FCS/4.
    C_T5FCS_Div_2:      EQU         %00010000       ; Timer5 Clock= FCS/2.
    C_T5FCS_Div_1:      EQU         %00000000       ; Timer5 Clock= FCS/1.
    C_T4EXT_EN:         EQU         %00000111       ; External Event.
    C_T4FCS_Div_512:    EQU         %00000110       ; Timer4 Clock= FCS/512.
    C_T4FCS_Div_128:    EQU         %00000101       ; Timer4 Clock= FCS/128.
    C_T4FCS_Div_32:     EQU         %00000100       ; Timer4 Clock= FCS/32.
    C_T4FCS_Div_8:      EQU         %00000011       ; Timer4 Clock= FCS/8.
    C_T4FCS_Div_4:      EQU         %00000010       ; Timer4 Clock= FCS/4.
    C_T4FCS_Div_2:      EQU         %00000001       ; Timer4 Clock= FCS/2.
    C_T4FCS_Div_1:      EQU         %00000000       ; Timer4 Clock= FCS/1.
;
P_TMR4_Count:           EQU         $21         ; Timer4 8/16-bit counter register.(R)
P_TMR4_Preload:         EQU         $21         ; Timer4 8/16-bit preload register.(W)
P_TMR4_Comp:            EQU         $21         ; Timer4 8/16-bit compare low byte value.
P_TMR4_Cap:             EQU         $21         ; Timer4 8/16-bit capture low byte width
value.
P_TMR4_PWMPeriod:           EQU         $21         ; Timer4 8-bit PWM period value.
;
P_TMR4_CountHi:         EQU         $22         ; Timer4 16-bit counter register.(R)
P_TMR4_PreloadHi:       EQU         $22         ; Timer4 16-bit preload register.(W)
P_TMR4_CompHi:          EQU         $22         ; Timer4 16-bit compare high byte value.
P_TMR4_CapHi:           EQU         $22         ; Timer4 16-bit capture high byte width
value.
P_TMR4_CapCycle8:       EQU         $22         ; Timer4 8-bit capture cycle value.(R)
P_TMR4_PWMDuty:         EQU         $22         ; Timer4 8-bit PWM duty value.
;
P_TMR5_Count:           EQU         $23         ; Timer5 8/16-bit Counter register.(R)
P_TMR5_Preload:         EQU         $23         ; Timer5 8/16-bit preload register.(W)
P_TMR5_Comp:            EQU         $23         ; Timer5 8/16-bit compare low byte value.
P_TMR5_Cap:             EQU         $23         ; Timer5 8/16-bit capture low byte width
value.
P_TMR5_PWMPeriod:           EQU         $23         ; Timer5 16-bit PWM peroid low byte
register.
```

```
;
P_TMR5_CountHi:         EQU         $24         ; Timer5 16-bit Counter register.(R)
P_TMR5_PreloadHi:       EQU         $24         ; Timer5 16-bit preload register.(W)
P_TMR5_CompHi:          EQU         $24         ; Timer5 16-bit compare high byte value.
P_TMR5_CapHi:           EQU         $24         ; Timer5 16-bit capture high byte width
value.
P_TMR5_CapCycle8:       EQU         $24         ; Timer5 8-bit capture cycle value.(R)
P_TMR5_DutyPeriod:      EQU         $24         ; Timer5 16-bit PWM period high byte
register.
;
P_TMR5_CapCycleLo:          EQU         $25         ; Timer5 16-bit capture cycle high byte
value.(R)
P_TMR5_PWMDuty:         EQU         $25         ; Timer5 16-bit PWM duty low byte register.
;
P_TMR5_CapCycleHi:          EQU         $5F         ; Timer5 16-bit capture cycle high byte
value.(R)
P_TMR5_PWMDutyHi:           EQU         $5F         ; Timer5 16-bit PWM duty high byte
register.
;
;-------------------------------------------------------------------------------------------
P_AD_Ctrl0:             EQU         $28         ; A/D converter control 0.
    C_AD_EN:            EQU         %10000000       ; ADC enable control.(A)
    C_AD_VRT:       EQU         %01000000       ; ADC bottom voltage source select bit.(A)

    C_AD_CS_2:          EQU         %00000000       ; 000=Fcpu/2.
    C_AD_CS_4:          EQU         %00000010       ; 001=Fcpu/4.
    C_AD_CS_8:          EQU         %00000100       ; 010=Fcpu/8.
    C_AD_CS_16:         EQU         %00000110       ; 011=Fcpu/16.
    C_AD_CS_32:         EQU         %00001000       ; 100=Fcpu/32.
    C_AD_CS_64:         EQU         %00001010       ; 101=Fcpu/64.
    C_AD_CS_128:        EQU         %00001100       ; 110=Fcpu/128.
    C_AD_CS_256:        EQU         %00001110       ; 111=Fcpu/256.
    C_AD_RDY:       EQU         %00000001       ; ADC status bit.(R)
    C_AD_Start:     EQU         %00000000       ; ADC conversion start bit.(W)

    CB_AD_EN:       EQU         7           ; ADC enable control for bit mode.(A)
    CB_AD_VRT:          EQU         6           ; ADC top voltage source select bit for bit
mode.(A)
    CB_AD_RDY:          EQU         0           ; ADC status bit for bit mode.(R)
    CB_AD_Start:        EQU         0           ; ADC conversion start bit for bit mode.(W)

;
```

```
P_AD_Ctrl1:         EQU         $29             ; A/D converter control 1.
    C_AD_Pin0:      EQU         %00000001       ; Analog input Port Configuration: channel
0.
    C_AD_Pin1:      EQU         %00000010       ; Analog input Port Configuration: channel
1.
    C_AD_Pin2:      EQU         %00000100       ; Analog input Port Configuration: channel
2.
    C_AD_Pin3:      EQU         %00001000       ; Analog input Port Configuration: channel
3.
    C_AD_Pin4:      EQU         %00010000       ; Analog input Port Configuration: channel
4.
    C_AD_Pin5:      EQU         %00100000       ; Analog input Port Configuration: channel
5.
    C_AD_Pin6:      EQU         %01000000       ; Analog input Port Configuration: channel
6.
    C_AD_Pin7:      EQU         %10000000       ; Analog input Port Configuration: channel
7.

    CB_AD_Pin0:         EQU         0           ; Analog input Configuration: channel 0 for
bit mode.
    CB_AD_Pin1:         EQU         1           ; Analog input Configuration: channel 1 for
bit mode.
    CB_AD_Pin2:         EQU         2           ; Analog input Configuration: channel 2 for
bit mode.
    CB_AD_Pin3:         EQU         3           ; Analog input Configuration: channel 3 for
bit mode.
    CB_AD_Pin4:         EQU         4           ; Analog input Configuration: channel 4 for
bit mode.
    CB_AD_Pin5:         EQU         5           ; Analog input Configuration: channel 5 for
bit mode.
    CB_AD_Pin6:         EQU         6           ; Analog input Configuration: channel 6 for
bit mode.
    CB_AD_Pin7:         EQU         7           ; Analog input Configuration: channel 7 for
bit mode.
;
P_AD_Ctrl2:         EQU         $2A             ; A/D converter control 2.
    C_AD_CE:            EQU         %10000000       ; ADC power control bit.(A)
    C_AD_Ch0:           EQU         %00000000       ; 0000:channel 0.
    C_AD_Ch1:           EQU         %00001000       ; 0001:channel 1.
    C_AD_Ch2:           EQU         %00010000       ; 0010:channel 2.
    C_AD_Ch3:           EQU         %00011000       ; 0011:channel 3.
    C_AD_Ch4:           EQU         %00100000       ; 0100:channel 4.
```

```
    C_AD_Ch5:       EQU     %00101000       ; 0101:channel 5.
    C_AD_Ch6:       EQU     %00110000       ; 0110:channel 6.
    C_AD_Ch7:       EQU     %00111000       ; 0111:channel 7.
    C_AD_Ch8:       EQU     %01000000       ; 1000:channel 8.
    C_AD_Pin8:      EQU     %00000001       ; Analog input : channel 8.

    CB_AD_CE:   EQU     7           ; ADC power control bit for bit mode.(A)
;
P_AD_DataHi:        EQU     $2B         ; Converted A/D data[9:2] hi.(R)
P_AD_DataLo:        EQU     $2C         ; Converted A/D data[1:0] low.(R)
;
;-------------------------------------------------------------------------------------------
P_BUZ_Ctrl:         EQU     $2D         ; Buzzer & Timer base Control.
                                        ; Interval Timer Period selecttion bits.
    C_TBASE_Dis:    EQU     %00000000       ; Time Base disable
    C_TBASE_Div_128:EQU     %00010000       ; Time Base Clk: Fto/2^7.
    C_TBASE_Div_256:EQU     %00100000       ; Time Base Clk: Fto/2^8.
    C_TBASE_Div_512:    EQU     %00110000       ; Time Base Clk: Fto/2^9.
    C_TBASE_Div_1k: EQU     %01000000       ; Time Base Clk: Fto/2^10.
    C_TBASE_Div_2k: EQU     %01010000       ; Time Base Clk: Fto/2^11.
    C_TBASE_Div_4k: EQU     %01100000       ; Time Base Clk: Fto/2^12.
    C_TBASE_Div_8k: EQU     %01110000       ; Time Base Clk: Fto/2^13.
    C_TBASE_Div_16k:EQU     %10000000       ; Time Base Clk: Fto/2^14.
    C_TBASE_Div_32k:EQU     %10010000       ; Time Base Clk: Fto/2^15.
    C_TBASE_Div_64k:EQU     %10100000       ; Time Base Clk: Fto/2^16.
    C_TBASE_Div_128k:EQU    %10110000       ; Time Base Clk: Fto/2^17.
    C_TBASE_Div_256k:EQU    %11000000       ; Time Base Clk: Fto/2^18.
    C_TBASE_Div_512k:EQU    %11010000       ; Time Base Clk: Fto/2^19.
    C_TBASE_Div_2M: EQU     %11100000       ; Time Base Clk: Fto/2^21.
    C_TBASE_Div_8M: EQU     %11110000       ; Time Base Clk: Fto/2^23.
                                        ; Buzzer frequency selecttion bits.
    C_BUZ_Dis:      EQU     %00000000       ; Buzzer disable
    C_BUZ_Div_64:       EQU     %00000001       ; Buzzer:Fto/2^6.
    C_BUZ_Div_128:      EQU     %00000010       ; Buzzer:Fto/2^7.
    C_BUZ_Div_256:      EQU     %00000011       ; Buzzer:Fto/2^8.
    C_BUZ_Div_512:      EQU     %00000100       ; Buzzer:Fto/2^9.
    C_BUZ_Div_1k:       EQU     %00000101       ; Buzzer:Fto/2^10.
    C_BUZ_Div_2k:       EQU     %00000110       ; Buzzer:Fto/2^11.
    C_BUZ_Div_4k:       EQU     %00000111       ; Buzzer:Fto/2^12.
    C_BUZ_Div_8k:       EQU     %00001000       ; Buzzer:Fto/2^13.
    C_BUZ_Div_4:        EQU     %00001001       ; Buzzer:Fto/2^2.
    C_BUZ_Div_8:        EQU     %00001010       ; Buzzer:Fto/2^3.
```

```
    C_BUZ_Div_16:       EQU         %00001011       ; Buzzer:Fto/2^4.
    C_BUZ_Div_32:       EQU         %00001100       ; Buzzer:Fto/2^5.
;
;-------------------------------------------------------------------------------------------
P_CMP_Ctrl:         EQU         $2E             ; Comparator control register.
    C_CMP1_EN:      EQU         %10000000       ; comparator1 function enable.
    C_CMP1_RS:      EQU         %01000000       ; comparator1 reference source
selection.
    C_CMP1_LOG:     EQU         %00100000       ; comparator1 logic event direction
selection.
    C_CMP1_OUT:     EQU         %00010000       ; comparator1 result bit.
    C_CMP0_EN:      EQU         %00001000       ; comparator0 function enable.
    C_CMP0_RS:      EQU         %00000100       ; comparator0 reference source
selection.
    C_CMP0_LOG:     EQU         %00000010       ; comparator0 logic event direction
selection.
    C_CMP0_OUT:     EQU         %00000001       ; comparator0 result bit.

    CB_CMP1_EN:     EQU         7               ; comparator1 function enable for bit mode.
    CB_CMP1_RS:     EQU         6               ; comparator1 reference
                                                ; source selection for bit mode.
    CB_CMP1_LOG:    EQU         5               ; comparator1 logic event direction
                                                ; selection for bit mode.
    CB_CMP1_OUT:    EQU         4               ; comparator1 result bit for bit mode.
    CB_CMP0_EN:     EQU         3               ; comparator0 function enable for bit mode.
    CB_CMP0_RS:     EQU         2               ; comparator0 reference
                                                ; source selection for bit mode.
    CB_CMP0_LOG:    EQU         1               ; comparator0 logic event
                                                ; direction selection for bit mode.
    CB_CMP0_OUT:    EQU         0               ; comparator0 result bit for bit mode.
;
;-------------------------------------------------------------------------------------------
; Double Write Register
P_SYS_Ctrl:         EQU         $30             ; System control.
    C_SCR_POR:      EQU         %10000000       ; Power On Reset Flag.(A)
    C_SCR_ERST:     EQU         %01000000       ; External Reset Flag.(A)
    C_SCR_LVR:      EQU         %00100000       ; Low Voltage Reset Flag.(A)
    C_SCR_WDTR:     EQU         %00001000       ; WDT Reset Flag.(A)
    C_SCR_IAR:      EQU         %00000100       ; Illegal Address Reset Flag.(A)
    C_SCR_IIR:      EQU         %00000001       ; Illegal instruction reset(A)

    CB_SCR_POR:     EQU         7               ; Power On Reset Flag for bit mode.(A)
```

```
    CB_SCR_ERST:    EQU     6       ; External Reset Flag for bit mode.(A)
    CB_SCR_LVR:     EQU     5       ; Low Voltage Reset Flag for bit mode.(A)
    CB_SCR_WDTR:    EQU     3       ; WDT Reset Flag for bit mode.(A)
    CB_SCR_IAR:     EQU     2       ; Illegal Address Reset Flag for bit mode.(A)
    CB_SCR_IIR:     EQU     0       ; Illegal instruction reset for bit mode.(A)
;
P_MODE_Ctrl:        EQU     $31     ; Operation mode control register.(W)
    C_MODE_STOP:    EQU     $5A     ; Enter STOP mode.(W)
    C_MODE_HALT:    EQU     $A5     ; Enter HALT mode.(W)
    C_MODE_Reset:   EQU     $66     ; Reset all of internal modules except
CPU.(W)
;
P_WDT_Ctrl:         EQU     $32     ; Watchdog control register.
    C_WDT_SCKEN:    EQU     %10000000   ; Slow Clock enable bit in stop
mode.(A)
                                    ; Selection bits of watchdog interrupt rate.(A)
    C_WDT_RTO:      EQU     %00000000   ; RTO from Timer0 is selected.
    C_WDT_Div_256:  EQU     %00010000   ; WDI clock = /256.
    C_WDT_Div_512:  EQU     %00100000   ; WDI clock = /512.
    C_WDT_Div_1024: EQU     %00110000   ; WDI clock = /1024.
    C_WDT_Div_2048: EQU     %01000000   ; WDI clock = /2048.
    C_WDT_Div_4096: EQU     %01010000   ; WDI clock = /4096.
    C_WDT_Div_8192: EQU     %01100000   ; WDI clock = /8192.
    C_WDT_Div_16384:EQU     %01110000   ; WDI clock = /16384.

    CB_WDT_EN:      EQU     7       ; Watchdog enable bit in stop mode for bit
mode.(A)
;
P_IRQ_Opt0:         EQU     $33     ; IRQ Option 0 register.
    C_IRQOpt0_IRQ5ES: EQU   %00001000   ; Polarity control of INT5.(A)
    C_IRQOpt0_IRQM5:  EQU   %00000100   ; INT5 trigger mode selection.(A)
    C_IRQOpt0_IRQ4ES: EQU   %00000010   ; Polarity control of INT4.(A)
    C_IRQOpt0_IRQM4:  EQU   %00000001   ; INT4 trigger mode selection.(A)
    C_IRQOpt0_CAP5ES:   EQU     %00001000   ; Polarity control of CAP5.(A)
    C_IRQOpt0_CAP4ES:   EQU     %00000010   ; Polarity control of CAP4.(A)

    CB_IRQOpt0_IRQ5ES:  EQU     3       ; Polarity control of INT5 for bit
mode.(A)
    CB_IRQOpt0_IRQM5:EQU    2       ; INT5 trigger mode selection for bit
mode.(A)
    CB_IRQOpt0_IRQ4ES:  EQU     1       ; Polarity control of INT4 for bit
mode.(A)
```

```
    CB_IRQOpt0_IRQM4:EQU        0           ; INT4 trigger mode selection for bit
mode.(A)
    CB_IRQOpt0_CAP5ES:  EQU     3           ; Polarity control of CAP5 for bit
mode.(A)
    CB_IRQOpt0_CAP4ES:  EQU     1           ; Polarity control of CAP4 for bit
mode.(A)
;
P_IRQ_Opt1:             EQU     $34         ; IRQ Option 1 register.
    C_IRQOpt1_IRQ3ES: EQU       %10000000       ; Polarity control of INT3.(A)
    C_IRQOpt1_IRQM3:  EQU       %01000000       ; INT3 trigger mode selection.(A)
    C_IRQOpt1_IRQ2ES: EQU       %00100000       ; Polarity control of INT2.(A)
    C_IRQOpt1_IRQM2:  EQU       %00010000       ; INT2 trigger mode selection.(A)
    C_IRQOpt1_IRQ1ES: EQU       %00001000       ; Polarity control of INT1.(A)
    C_IRQOpt1_IRQM1:  EQU       %00000100       ; INT1 trigger mode selection.(A)
    C_IRQOpt1_IRQ0ES: EQU       %00000010       ; Polarity control of INT0.(A)
    C_IRQOpt1_IRQM0:  EQU       %00000001       ; INT0 trigger mode selection.(A)
    C_IRQOpt1_CAP3ES:   EQU     %00001000       ; Polarity control of CAP3.(A)
    C_IRQOpt1_CAP2ES:   EQU     %00000010       ; Polarity control of CAP2.(A)

    CB_IRQOpt1_IRQ3ES:  EQU     7           ; Polarity control of INT3 for bit
mode.(A)
    CB_IRQOpt1_IRQM3:EQU        6           ; INT3 trigger mode selection for bit
mode.(A)
    CB_IRQOpt1_IRQ2ES:  EQU     5           ; Polarity control of INT2 for bit
mode.(A)
    CB_IRQOpt1_IRQM2:EQU        4           ; INT2 trigger mode selection for bit
mode.(A)
    CB_IRQOpt1_IRQ1ES:  EQU     3           ; Polarity control of INT1 for bit
mode.(A)
    CB_IRQOpt1_IRQM1:EQU        2           ; INT1 trigger mode selection for bit
mode.(A)
    CB_IRQOpt1_IRQ0ES:  EQU     1           ; Polarity control of INT0 for bit
mode.(A)
    CB_IRQOpt1_IRQM0:EQU        0           ; INT0 trigger mode selection for bit
mode.(A)
    CB_IRQOpt1_CAP3ES:  EQU     3           ; Polarity control of CAP3 for bit
mode.(A)
    CB_IRQOpt1_CAP2ES:  EQU     1           ; Polarity control of CAP2 for bit
mode.(A)
;
P_IO_Opt:               EQU     $35         ; I/O slew rate control register.
    C_IO_SLOWE:         EQU     %00000001       ; PB[7:6] slew rate enable selection.(A)
```

```
    CB_IO_SLOWE:    EQU     0           ; PB[7:6] slew rate enable selection for bit
mode.(A)
;
P_LVR_Opt:          EQU     $36         ; LVR Option
    C_LVR_V40:      EQU     %00000001       ; LVR level select bit.(A)

    CB_LVR_V40:     EQU     0           ; LVR level select bit for bit mode.(A)
;
;-------------------------------------------------------------------------------------------
P_SPI_Ctrl0:        EQU     $38         ; SPI control register 0.
    C_SPI_EN:       EQU     %10000000       ; enable Control bit.
    C_SPI_MOD:      EQU     %01000000       ; operation Mode Master/Slave Mode.
    C_SPI_SCKPHA:   EQU     %00100000       ; clock phase.
    C_SPI_SCKPOL:   EQU     %00010000       ; clock polarity.
    C_SPI_SPISMPS:  EQU     %00001000       ; sample mode selection bit for master
mode.
    C_SPICS_Div_128:EQU     %00000101       ; CPU Clock Selection/128.
    C_SPICS_Div_64: EQU     %00000100       ; CPU clock Selection/64.
    C_SPICS_Div_32: EQU     %00000011       ; CPU clock Selection/32.
    C_SPICS_Div_16: EQU     %00000010       ; CPU clock Selection/16.
    C_SPICS_Div_8:  EQU     %00000001       ; CPU clock Selection/8.
    C_SPICS_Div_4:  EQU     %00000000       ; CPU clock Selection/4.

    CB_SPI_EN:      EQU     7           ; enable Control bit for bit mode.
    CB_SPI_MOD:     EQU     6           ; operation Mode Master/Slave Mode for bit
mode.
    CB_SPI_SCKPHA:  EQU     5           ; clock phase for bit mode.
    CB_SPI_SCKPOL:  EQU     4           ; clock polarity for bit mode.
    CB_SPI_SPISMPS: EQU     3           ; sample mode selection bit for master
mode for bit mode.
;
P_SPI_Ctrl1:        EQU     $39         ; SPI control register 1.
    C_SPI_SMSEN:    EQU     %10000000       ; SPI Slave Mode Selection enable bit.
    C_SPI_SWRST:    EQU     %01000000       ; software reset bit.
    C_SPISPC_Div_4: EQU     %00000011       ; sampling clock Div/4.
    C_SPISPC_Div_2: EQU     %00000010       ; sampling clock Div/2.
    C_SPISPC_Div_1: EQU     %00000001       ; sampling clock Div/1.
    C_SPISPC_Dis:   EQU     %00000000       ; no sampling.

    CB_SPI_SMSEN:   EQU     7           ; SPI Slave Mode Selection enable bit for
bit mode.
```

```
    CB_SPI_SWRST:    EQU      6          ; software reset bit for bit mode.
;
P_SPI_Status:        EQU      $3A        ; SPI status register.
    C_SPI_INTIF:     EQU      %10000000      ; SPI interrupt flag active.
    C_SPI_INTEN:     EQU      %01000000      ; SPI interrupt enable/diable.
    C_SPI_TXBF:      EQU      %00100000      ; transmission buffer full flag.
    C_SPI_BUFFull:   EQU      %00000001      ; buffer full and overwrite.

    CB_SPI_INTIF:    EQU      7          ; SPI interrupt flag active for bit mode.
    CB_SPI_INTEN:    EQU      6          ; SPI interrupt enable/diable for bit mode.
    CB_SPI_TXBF:     EQU      5          ; transmission buffer full flag for bit mode.
    CB_SPI_BUFFull:      EQU      0          ; buffer full and overwrite for bit mode.
;
P_SPI_TxData:        EQU      $3B        ; SPI Transmit data buffer.
P_SPI_RxData:        EQU      $3C        ; SPI Receive data buffer.
;
;-------------------------------------------------------------------------------------------
P_IOE_Data:      EQU      $40        ; Port E data b0~b7.(A)
P_IOF_Data:      EQU      $41        ; Port F data b0~b7.(A)
P_IOE_Dir:           EQU      $42        ; Port E direction control   b0~b7.(W)
P_IOF_Dir:           EQU      $43        ; Port F direction control   b0~b7.(W)
P_IOE_Attrib:        EQU      $44        ; Port E attribute register b0~b7.(W)
P_IOF_Attrib:        EQU      $45        ; Port F attribute register b0~b7.(W)
;
;-------------------------------------------------------------------------------------------
P_UART_Ctrl:         EQU      $46        ; UART control register.
    C_UART_RXIE:     EQU      %10000000      ; receive interrupt enable bit.
    C_UART_TXIE:     EQU      %01000000      ; transmit interrupt enable bit.
    C_UART_RXEN:     EQU      %00100000      ; receive function enable bit.
    C_UART_TXEN:     EQU      %00010000      ; transmit function enable bit.
    C_UART_SOFTRST:  EQU      %00001000      ; software reset.
    C_UART_STOPSEL:  EQU      %00000100      ; stop bit length selection bit.
    C_UART_PSEL:     EQU      %00000010      ; parity type selection bit.
    C_UART_PEN:      EQU      %00000001      ; parity check or generation enable bit.

    CB_UART_RXIE:    EQU      7          ; receive interrupt enable bit for bit mode.
    CB_UART_TXIE:    EQU      6          ; transmit interrupt enable bit for bit mode.
    CB_UART_RXEN:    EQU      5          ; receive function enable bit for bit mode.
    CB_UART_TXEN:    EQU      4          ; transmit function enable bit for bit mode.
    CB_UART_SOFTRST:     EQU      3          ; software reset for bit mode.
    CB_UART_STOPSEL:     EQU      2          ; stop bit length selection bit for bit
mode.
```

```
    CB_UART_PSEL:   EQU     1       ; parity type selection bit for bit mode.
    CB_UART_PEN:    EQU     0       ; parity check or generation enable bit for
bit mode.
;
P_UART_Baud:        EQU     $47     ; UART baud rate divisor.
;
P_UART_Status:      EQU     $48     ; UART staus register.
    C_UART_RXIF:    EQU     %10000000       ; receive interrupt flag.
    C_UART_TXIF:    EQU     %01000000       ; transmit interrupt flag.
    C_UART_BUSY:    EQU     %00100000       ; transmission in progress flag bit.
    C_UART_OERR:    EQU     %00000100       ; overrun error flag bit.
    C_UART_PERR:    EQU     %00000010       ; parity error flag bit.
    C_UART_FERR:    EQU     %00000001       ; frame error flag bit.

    CB_UART_RXIF:   EQU     7       ; receive interrupt flag for bit mode.
    CB_UART_TXIF:   EQU     6       ; transmit interrupt flag for bit mode.
    CB_UART_BUSY:   EQU     5       ; transmission in progress flag bit for bit
mode.
    CB_UART_OERR:   EQU     2       ; overrun error flag bit for bit mode.
    CB_UART_PERR:   EQU     1       ; parity error flag bit for bit mode.
    CB_UART_FERR:   EQU     0       ; frame error flag bit for bit mode.
;
P_UART_Data:        EQU     $49     ; UART data register.
;
;-------------------------------------------------------------------------------------------
P_IIC_Ctrl:         EQU     $4A     ; IIC series control register.
    C_IIC_IICEN:    EQU     %10000000       ; IIC series interface enable bit.
    C_IIC_TXRXEN:   EQU     %00100000       ; IIC bus data output enable bit.
    C_IIC_INTEN:    EQU     %00010000       ; IIC interface interrupt enable bit.
    C_IIC_ACKEN:    EQU     %00001000       ; IIC acknowledge enable bit.
    C_IIC_SCK_1024: EQU     %00000111       ; interface clock rate. Fcs/1024.
    C_IIC_SCK_512:  EQU     %00000110       ; interface clock rate. Fcs/512.
    C_IIC_SCK_256:  EQU     %00000101       ; interface clock rate. Fcs/256.
    C_IIC_SCK_128:  EQU     %00000100       ; interface clock rate. Fcs/128.
    C_IIC_SCK_64:   EQU     %00000011       ; interface clock rate. Fcs/64.
    C_IIC_SCK_32:   EQU     %00000010       ; interface clock rate. Fcs/32.
    C_IIC_SCK_16:   EQU     %00000001       ; interface clock rate. Fcs/16.
    C_IIC_SCK_8:    EQU     %00000001       ; interface clock rate. Fcs/8.

    CB_IIC_IICEN:   EQU     7       ; IIC series interface enable bit for bit mode.
    CB_IIC_TXRXEN:  EQU     5       ; IIC bus data output enable bit for bit mode.
    CB_IIC_INTEN:   EQU     4       ; IIC interface interrupt enable bit for bit
```

```
mode.
    CB_IIC_ACKEN:    EQU    3          ; IIC acknowledge enable bit for bit mode.
;
P_IIC_Status:        EQU    $4B        ; IIC series status register.
    C_IIC_MXT:       EQU    %10000000  ; IIC master/slave mode selection.
    C_IIC_TXRXSEL:   EQU    %01000000      ; Tx/Rx selection bit.
    C_IIC_BUSY:      EQU    %00100000      ; bus busy signal.
    C_IIC_SIGGEN:    EQU    %00100000      ; IIC start/stop signal generation.
    C_IIC_INTIF:     EQU    %00010000  ; IIC series interrupt flag bit.
    C_IIC_ARBITRAT:  EQU    %00001000      ; IIC bus arbitration status.
    C_IIC_AAS:       EQU    %00000100  ; I2C bus address-as-slave status flag data
foramt bit0.
    C_IIC_AZERO:     EQU    %00000010      ; IIC address zero status flag.
    C_IIC_LRB:       EQU    %00000001      ; IIC last-received bit status flag.
    CB_IIC_MXT:      EQU    7          ; IIC master/slave mode selection for bit
mode.
    CB_IIC_TXRXSEL:  EQU    6          ; Tx/Rx selection bit for bit mode.
    CB_IIC_BUSY:     EQU    5          ; bus busy signal for bit mode.
    CB_IIC_SIGGEN:   EQU    5          ; IIC start/stop signal generation for bit
mode.
    CB_IIC_INTIF:    EQU    4          ; IIC series interrupt flag bit for bit mode.
    CB_IIC_ARBITRAT: EQU    3          ; IIC bus arbitration status for bit mode.
    CB_IIC_AAS:      EQU    2          ; I2C bus address-as-slave status flag data
foramt bit0 for bit mode.
    CB_IIC_AZERO:    EQU    1          ; IIC address zero status flag for bit mode.
    CB_IIC_LRB:      EQU    0          ; IIC last-received bit status flag for bit
mode.
;
P_IIC_Data:          EQU    $4C        ; IIC series data register.
P_IIC_Address:       EQU    $4D        ; IIC series address bits.
;
;-------------------------------------------------------------------------------------------
P_DA_Ctrl:           EQU    $55        ; DA converter control register.
    C_DA_EN:         EQU    %10000000      ; DA converter enable bit.

    CB_DA_EN:        EQU    7          ; DA converter enable bit for bit mode.
;
P_DA_DataLo:         EQU    $56        ; Converted D/A data[1:0] low.(W)
P_DA_DataHi:         EQU    $57        ; Converted D/A data[9:2] hi.(W)
;
;-------------------------------------------------------------------------------------------
P_CAP_Ctrl:          EQU    $58        ; Capture control.
```

```
    C_CAP_OPT:      EQU     %10000000       ; Capture option control bit.(A)
    C_CAP_IP4:      EQU     %01000000       ; CAP4 interrupt evoke polarity.
    C_CAP_IP3:      EQU     %00100000       ; CAP3 interrupt evoke polarity.
    C_CAP_IP2:      EQU     %00010000       ; CAP2 interrupt evoke polarity.
    C_CAP_IP1:      EQU     %00001000       ; CAP1 interrupt evoke polarity.
    C_CAP_IP0:      EQU     %00000100       ; CAP0 interrupt evoke polarity.
    C_CAP1_ES:      EQU     %00000010       ; Polarity control of capture1 interrupt.
    C_CAP0_ES:      EQU     %00000001       ; Polarity control of capture0 interrupt.

    CB_CAP_OPT:     EQU     7               ; Capture option control bit for bit mode.(A)
    CB_CAP_IP4:     EQU     6               ; CAP4 interrupt evoke polarity for bit
mode.
    CB_CAP_IP3:     EQU     5               ; CAP3 interrupt evoke polarity for bit
mode.
    CB_CAP_IP2:     EQU     4               ; CAP2 interrupt evoke polarity for bit
mode.
    CB_CAP_IP1:     EQU     3               ; CAP1 interrupt evoke polarity for bit
mode.
    CB_CAP_IP0:     EQU     2               ; CAP0 interrupt evoke polarity for bit
mode.
    CB_CAP1_ES:     EQU     1               ; Polarity control of capture1 interrupt for bit
mode.
    CB_CAP0_ES:     EQU     0               ; Polarity control of capture0 interrupt for bit
mode.
;
;-------------------------------------------------------------------------------------------
P_IOA_Buf:          EQU     $59
    C_IOA_Buf7:     EQU     %10000000       ; Output data Latch of PORTA bit7.
    C_IOA_Buf6:     EQU     %01000000       ; Output data Latch of PORTA bit6.
    C_IOA_Buf5:     EQU     %00100000       ; Output data Latch of PORTA bit5.
    C_IOA_Buf4:     EQU     %00010000       ; Output data Latch of PORTA bit4.
    C_IOA_Buf3:     EQU     %00001000       ; Output data Latch of PORTA bit3.
    C_IOA_Buf2:     EQU     %00000100       ; Output data Latch of PORTA bit2.
    C_IOA_Buf1:     EQU     %00000010       ; Output data Latch of PORTA bit1.
    C_IOA_Buf0:     EQU     %00000001       ; Output data Latch of PORTA bit0.
P_IOB_Buf:          EQU     $5A
    C_IOB_Buf7:     EQU     %10000000       ; Output data Latch of PORTB bit7.
    C_IOB_Buf6:     EQU     %01000000       ; Output data Latch of PORTB bit6.
    C_IOB_Buf5:     EQU     %00100000       ; Output data Latch of PORTB bit5.
    C_IOB_Buf4:     EQU     %00010000       ; Output data Latch of PORTB bit4.
    C_IOB_Buf3:     EQU     %00001000       ; Output data Latch of PORTB bit3.
    C_IOB_Buf2:     EQU     %00000100       ; Output data Latch of PORTB bit2.
```

```
    C_IOB_Buf1:     EQU     %00000010       ; Output data Latch of PORTB bit1.
    C_IOB_Buf0:     EQU     %00000001       ; Output data Latch of PORTB bit0.
P_IOC_Buf:          EQU     $5B
    C_IOC_Buf7:     EQU     %10000000       ; Output data Latch of PORTC bit7.
    C_IOC_Buf6:     EQU     %01000000       ; Output data Latch of PORTC bit6.
    C_IOC_Buf5:     EQU     %00100000       ; Output data Latch of PORTC bit5.
    C_IOC_Buf4:     EQU     %00010000       ; Output data Latch of PORTC bit4.
    C_IOC_Buf3:     EQU     %00001000       ; Output data Latch of PORTC bit3.
    C_IOC_Buf2:     EQU     %00000100       ; Output data Latch of PORTC bit2.
    C_IOC_Buf1:     EQU     %00000010       ; Output data Latch of PORTC bit1.
    C_IOC_Buf0:     EQU     %00000001       ; Output data Latch of PORTC bit0.
P_IOD_Buf:          EQU     $5C
    C_IOD_Buf7:     EQU     %10000000       ; Output data Latch of PORTD bit7.
    C_IOD_Buf6:     EQU     %01000000       ; Output data Latch of PORTD bit6.
    C_IOD_Buf5:     EQU     %00100000       ; Output data Latch of PORTD bit5.
    C_IOD_Buf4:     EQU     %00010000       ; Output data Latch of PORTD bit4.
    C_IOD_Buf3:     EQU     %00001000       ; Output data Latch of PORTD bit3.
    C_IOD_Buf2:     EQU     %00000100       ; Output data Latch of PORTD bit2.
    C_IOD_Buf1:     EQU     %00000010       ; Output data Latch of PORTD bit1.
    C_IOD_Buf0:     EQU     %00000001       ; Output data Latch of PORTD bit0.
P_IOE_Buf:          EQU     $5D
    C_IOE_Buf7:     EQU     %10000000       ; Output data Latch of PORTE bit7.
    C_IOE_Buf6:     EQU     %01000000       ; Output data Latch of PORTE bit6.
    C_IOE_Buf5:     EQU     %00100000       ; Output data Latch of PORTE bit5.
    C_IOE_Buf4:     EQU     %00010000       ; Output data Latch of PORTE bit4.
    C_IOE_Buf3:     EQU     %00001000       ; Output data Latch of PORTE bit3.
    C_IOE_Buf2:     EQU     %00000100       ; Output data Latch of PORTE bit2.
    C_IOE_Buf1:     EQU     %00000010       ; Output data Latch of PORTE bit1.
    C_IOE_Buf0:     EQU     %00000001       ; Output data Latch of PORTE bit0.
P_IOF_Buf:          EQU     $5E
    C_IOF_Buf7:     EQU     %10000000       ; Output data Latch of PORTF bit7.
    C_IOF_Buf6:     EQU     %01000000       ; Output data Latch of PORTF bit6.
    C_IOF_Buf5:     EQU     %00100000       ; Output data Latch of PORTF bit5.
    C_IOF_Buf4:     EQU     %00010000       ; Output data Latch of PORTF bit4.
    C_IOF_Buf3:     EQU     %00001000       ; Output data Latch of PORTF bit3.
    C_IOF_Buf2:     EQU     %00000100       ; Output data Latch of PORTF bit2.
    C_IOF_Buf1:     EQU     %00000010       ; Output data Latch of PORTF bit1.
    C_IOF_Buf0:     EQU     %00000001       ; Output data Latch of PORTF bit0.
C_RAM_ADDR:         EQU     $60             ; RAM Start Address.
C_STACK_BOTTOM:     EQU     $0FF                ; Stack Bottom Address.

CB_Bit7             EQU     7               ; for bit mode
```

```
CB_Bit6         EQU     6       ;
CB_Bit5         EQU     5       ;
CB_Bit4         EQU     4       ;
CB_Bit3         EQU     3       ;
CB_Bit2         EQU     2       ;
CB_Bit1         EQU     1       ;
CB_Bit0         EQU     0       ;
```