PMC251 系列
数据手册

IO 类型、采用 FPPA™ 技术
双核心 8 位单片机

第 0.12 版
2014 年 5 月 20 日

Copyright © 2014 by PADAUK Technology Co., Ltd., all rights reserved

10F-2, No. 1, Sec. 2, Dong-Da Road, Hsin-Chu 300, Taiwan, R.O.C.

TEL: 886-3-532-7598 www.padauk.com.tw

重要声明

应广科技保留权利在任何时候变更或终止产品，建议客户在使用或下单前与应广科技或代理商联系以取得最新、最正确的产品信息。

应广科技不担保本产品适用于保障生命安全或紧急安全的应用，应广科技不为此类应用产品承担任何责任。关键应用产品包括，但不仅限于，可能涉及的潜在风险的死亡，人身伤害，火灾或严重财产损失。

应广科技不承担任何责任来自于因客户的产品设计所造成的任何损失。在应广科技所保障的规格范围内，客户应设计和验证他们的产品。为了尽量减少风险，客户设计产品时，应保留适当的产品工作范围安全保障。

提供本文档的中文简体版是为了便于了解，请勿忽视文中英文的部份，因为其中提供有关产品性能以及产品使用的有用信息，应广科技暨代理商对于文中可能存在的差错不承担任何责任，建议参考本文件英文版。

©Copyright 2014, PADAUK Technology Co. Ltd

目　录

1. 单片机特点 7
1.1 高性能 RISC CPU 架构 7
1.2 系统功能 7
2. 系统概述和方框图 8
3. 引脚框图和引脚功能说明 9
4. 器件电气特性 11
4.1 直流交流电气特性 11
4.2 绝对最大值 12
4.3 IHRC 频率与 VDD 关系曲线图 13
4.4 ILRC 频率与 VDD 关系曲线图 13
4.5 IHRC 频率与温度关系曲线图 14
4.6 ILRC 频率与温度关系曲线图 14
4.7 工作电流与 VDD、系统时钟 CLK=IHRC/n 曲线图 15
4.8 工作电流与 VDD、系统时钟 CLK=ILRC/n 曲线图 15
4.9 最低工作电流与 VDD、系统时钟 CLK=ILRC/n 曲线图 16
4.10 工作电流与 VDD、系统时钟 CLK=32KHz EOSC/n 曲线图 17
4.11 工作电流与 VDD、系统时钟 CLK=1MHz EOSC/n 曲线图 18
4.12 工作电流与 VDD、系统时钟 CLK=4MHz EOSC/n 曲线图 19
4.13 引脚拉高电阻曲线图 20
4.14 引脚输出驱电流(Ioh)与灌电流(Iol) 曲线图 20
4.15 引脚输出输入高电压与低电压(V_{IH} / V_{IL}) 曲线图 20
5. 功能概述 21
5.1 处理单元 21
5.1.1 程序计数器 22
5.1.2 堆栈指针 22
5.1.3 一个处理单元工作模式 23
5.2 OTP 程序存储器 24
5.2.1 程序存储器分配 24
5.2.2 两个处理单元工作模式下程序存储器分配例子 24
5.2.3 一个处理单元工作模式下程序存储器分配例子 25
5.3 程序结构 26
5.3.1 两个处理单元工作模式下程序结构 26
5.3.2 一个处理单元工作模式下程序结构 26
5.4 开机流程 27
5.5 数据存储器 - SRAM 28
5.6 算术和逻辑单元 29
5.7 振荡器和时钟 29
5.7.1 内部高频振荡器和内部低频振荡 29
5.7.2 芯片校准 29
5.7.3 IHRC 校准 30

5.7.4 外部晶体振荡器 31
5.7.5 系统时钟和 LVD 基准位 32
5.7.6 系统时钟切换 33
5.8 16 位定时器 (Timer16) 34
5.9 看门狗定时器 35
5.10 中断 36
5.11 省电与掉电 38
5.11.1 省电模式（stopexe） 38
5.11.2 掉电模式（stopsys） 39
5.11.3 唤醒 40
5.12 IO 引脚 41
5.13 复位和 LVD 42
5.13.1 复位 42
5.13.2 LVD 复位 42
6. IO 寄存器 43
6.1 标志寄存器（*flag*），IO 地址 = 0x00 43
6.2 FPP 单元允许寄存器（*fppen*），IO 地址 = 0x01 43
6.3 堆栈指针寄存器（*sp*），IO 地址 = 0x02 43
6.4 时钟控制寄存器（*clkmd*），IO 地址 = 0x03 43
6.5 中断允许寄存器（*inten*），IO 地址 = 0x04 44
6.6 中断请求寄存器（*intrq*），IO 地址 = 0x05 44
6.7 Timer16 控制寄存器（*t16m*），IO 地址 = 0x06 44
6.8 通用数据输入/输出寄存器（*gdio*），IO 地址 = 0x07 45
6.9 外部晶体振荡器控制寄存器（*eoscr*，只写），IO 地址 = 0x0a 45
6.10 内部高频 RC 振荡器控制寄存器（*ihrcr*，只写），IO 地址 = 0x0b 45
6.11 中断沿选择寄存器（*integs*，只写），IO 地址 = 0x0c 45
6.12 端口 A 数字输入启用寄存器（*padier*），IO 地址 = 0x0d 46
6.13 端口 B 数字输入启用寄存器（*pbdier*），IO 地址 = 0x0e 47
6.14 端口 A 数据寄存器（*pa*），IO 地址 = 0x10 48
6.15 端口 A 控制寄存器（*pac*），IO 地址 = 0x11 48
6.16 端口 A 上拉控制寄存器（*paph*），IO 地址 = 0x12 48
6.17 端口 B 数据寄存器（*pb*），IO 地址 = 0x14 48
6.18 端口 B 控制寄存器（*pbc*），IO 地址 = 0x15 48
6.19 端口 B 上拉控制寄存器（*pbph*），IO 地址 = 0x16 48
6.20 杂项寄存器（*misc*），IO 地址 = 0x3b 49
7. 指令 50
7.1 数据传输类指令 50
7.2 算术运算类指令 54
7.3 移位运算类指令 55
7.4 逻辑运算类指令 56
7.5 位运算类指令 59
7.6 条件运算类指令 60
7.7 系统控制类指令 62
7.8 指令执行周期综述 63

©Copyright 2014, PADAUK Technology Co. Ltd

7.9 指令影响标志的综述 64
8. 特别注意事项 65
8.1. 使用 IC 时 65
8.1.1. IO 使用与设定 65
8.1.2. 中断 66
8.1.3. 切换系统时钟 66
8.1.4. 掉电模式、唤醒以及看门狗 67
8.1.5. TIMER 溢出时间 67
8.1.6. 延时指令 67
8.1.7. LVR 68
8.1.8. 单/双核模式下指令周期差异 68
8.1.9. 烧录方法 68
8.2. 使用 IDE 时 69
8.2.1. PMC251 系列于仿真器 PDK3S-I-001/002/003 上仿真时: 69
8.2.2. 使用 PDK3S-I-001/002/003 仿真 PMC251 系列功能時注意事項: 69

修订历史:

修订	日期	描述
0.02	2012/5/1	初版
0.03	2012/6/25	1. 修正 4.5 IHRC 频率与温度关系曲线图 2. 修正 4.6 ILRC 频率与温度关系曲线图 3. 修正 clkmd 寄存器描述
0.04	2012/7/25	1. 删除 *delay* 指令
0.05	2012/8/2	1. 修正 clkmd 寄存器：当内部低频 RC 振荡器功能禁用时，看门狗定时器功能同时被关闭。
0.06	2012/8/8	1. 增加仿真器使用 padier, pbdier 寄存器注意事项
0.07	2012/9/7	1. 修正 PAPH 和 PBPH 寄存器都是可以读写 2. 增加购买信息：PMC251 – S14 3. 修正 LVD 规格 4. 修正图 9 系统时钟选项 5. 修正看门狗定时器超时溢出时间 6. 修正 5.4. 开机流程描述 7. 修正 misc.4 描述：从 LVD 复位后，单片机开机时间 8. 编排 5.7.4.范例 9. 修正 16m 寄存器位[7:5]，以保持与 ICE 的一致性 10. 删除 rstst 寄存器，以保持与 ICE 的一致性 11. 修正 5.13. 关于 rstst 寄存器描述，以保持与 ICE 的一致性 12. 增加 padier 和 pbdier 寄存器注意事项，确保程序在 ICE 仿真和实际芯片是同一个 13. 修正量测图 4.3 和 4.4 – IHRC/ILRC 频率和 VDD 关系图 14. 修正 t_{WUP} 系统唤醒时间和 5-11-3 章节，删除图.15 15. 修正 5.7.2 芯片校准, **p2** =16～18;
0.08	2012/10/3	1. 修正量测图 4.3, 4.5, 4.6 2. 修正 misc 寄存器位 4 3. 删除 Bandgap 规格 4. 修正图 10，删除 PA4 输入 5. 修正 f_{IHRC} 规格 6. 增加 5-11-1 使用注意事项：(A) 在下 stopexe 命令前要关闭看门狗时钟 (B) stopexe 命令下，用 Timer16 唤醒系统
0.09	2012/11/23	1. 增加章节 5-7-6：系统频率切换 2. 增加 DIP14 封装信息 3. 修正 t_{WUP} 系统唤醒时间规格 4. 修正低电压侦测电压 V_{BRD} 为 V_{LVD}
0.10	2013/3/31	1. 用 $I_{power\ down}$ 取代 $I_{stand\ by}$ 2. 于脚位图增加 DIP14 封装 3. Page 39 : 当 EOSC 被选用当系统时钟后，快速换醒就自动关闭 4. 移除封装打印信息
0.11	2013/10/30	1. 增加输入电压(Input voltage) 与脚位的引入电流(Injected current on pin) 于直流/交流电气特性表中 2. 修正 7.8 指令执行周其综述
0.12	2014/5/20	1. 增加节章 8 特别注意事项

©Copyright 2014, PADAUK Technology Co. Ltd

1. 单片机特点

1.1 高性能 RISC CPU 架构

- 工作模式：2 个 FPPA™ 平行工作模式或传统式单片机工作模式
- 1KW OTP 程序存储器
- 60 字节数据存储器
- 提供 99 条指令
- 绝大部分指令都是单周期（1T）指令
- 可程序设定的堆栈深度
- 提供数据与指令的直接、间接寻址模式
- 提供位（Bit）处理指令
- 所有的数据存储器都可当数据指针（index pointer）
- 程序代码保护功能
- 独立的 IO 地址以及存储地址方便程序开发

1.2 系统功能

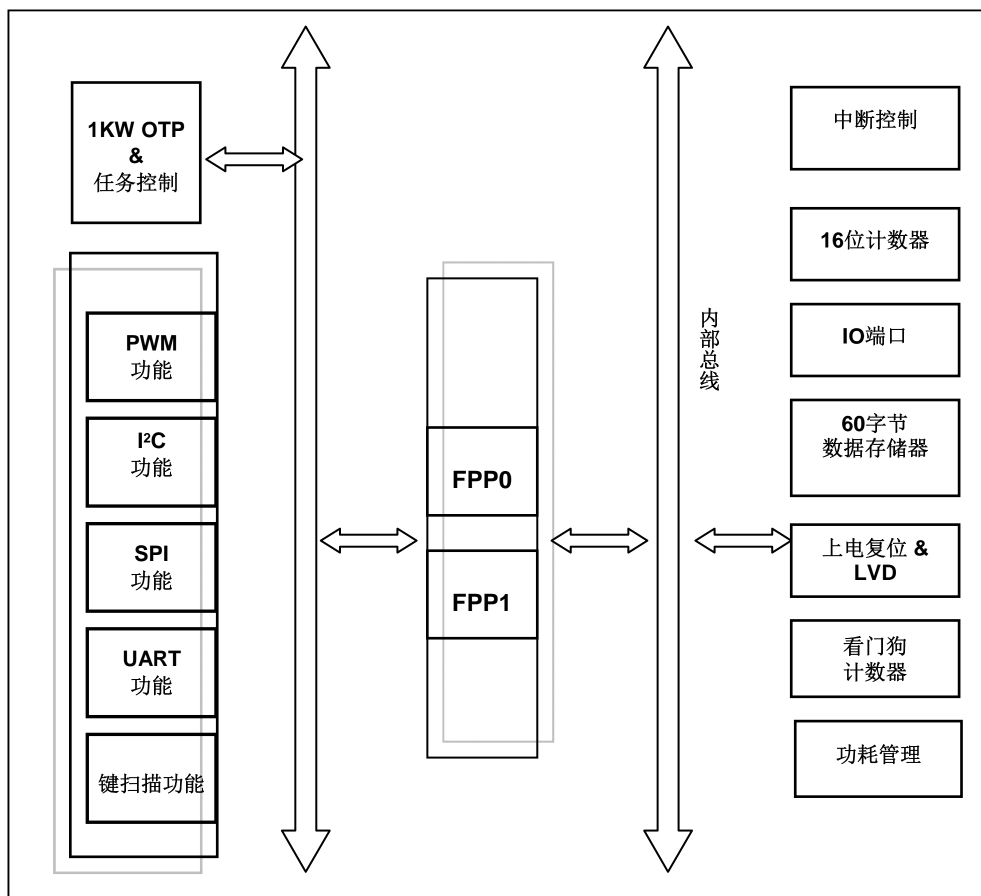
- 时钟模式：内部高频振荡器、内部低频振荡器、外部晶振
- 内置高频 RC 振荡器（IHRC）
- 内置 Band-gap 硬件模块输出 1.20V 参考电压
- 硬件 16 位计数器
- 快速唤醒功能
- 8 段 LVD 复位设定~ 4.1V, 3.6V, 3.1V, 2.8V, 2.5V, 2.2V, 2.0V, 1.8V
- 12 个 IO 引脚
- 2 个外部中断输入引脚
- IO 引脚具有 10mA 电流驱动能力
- 每个引脚都可弹性设定唤醒功能
- 工作频率（合并 2 个 FPP 处理单元）
  0 ~ 8MHz@VDD≧3.3V; 0 ~ 4MHz@VDD≧2.5V; 0 ~ 2MHz@VDD≧2.2V;
- 工作电压：2.2V ~ 5.5V
- 工作温度：-40°C ~ 85°C
- 功耗特性：
  $I_{operating}$ ~ 1.7mA@1MIPS, VDD=5.0V; $I_{operating}$ ~ 8uA@12KHz, VDD=3.3V
  $I_{power\ down}$ ~ 0.7uA@VDD=5.0V; $I_{power\ down}$ ~ 0.4uA@VDD=3.3V
- 购买信息：
  PMC251 - S14: SOP14 (150mil); PMC251 - D14: DIP14 (300mil);

©Copyright 2014, PADAUK Technology Co. Ltd

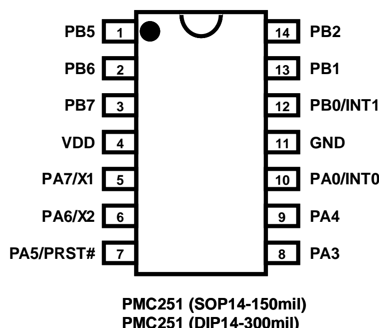
2. 系统概述和方框图

PMC251 是一个 IO 类型、并行处理、完全静态，以 OTP 为程序存储基础的处理器，此处理器具有两个处理单元可并行执行两个程序。另外它还提供和传统单片机一样的单核处理模式。它在 RISC 架构的基础上获得专利（Field Programmable Processor Array 现场可编程处理器阵列）技术。PMC251 可以采用 FPPA™ 两核心平行处理架构工作或传统单核心架构工作。大部分的指令执行时间都是一个指令周期，只有少部分指令是需要两个指令周期。

PMC251 内置 1KW OTP 程序存储器以及 60 字节数据存储器，供两个 FPP 单元工作使用；另外，PMC251 还提供一个 16 位的硬件计数器。常用的周边组件功能，例如：UART、PWM 都可以轻易的利用 FPPA™架构来实现。



3. 引脚框图和引脚功能说明



PMC251 引脚功能说明：

引脚名称	引脚&缓冲器类型	功能描述
PA7/X1	IO ST / CMOS / 模拟	此引脚可用做： （1）当端口 A 位 7，并可编程设定为输入或输出，弱上拉电阻模式。 （2）使用晶体振荡器时，作 X1 (input)用。 假如这个引脚被选来当晶体振荡器使用，寄存器 *padier* 位 7 必须设置为”0”以避免漏电流。这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *padier* 位 7 为”0”时，唤醒功能是被关闭的。
PA6/X2	IO ST / CMOS /模拟	此引脚可用做： （1）当端口 A 位 6，并可编程设定为输入/输出，弱上拉电阻模式。 （2）使用晶体振荡器时，作 X2 (output)用。 假如这个引脚被选来当晶体振荡器使用，寄存器 *padier* 位 6 必须设置为”0”以避免漏电流。这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *padier* 位 6 为”0”时，唤醒功能是被关闭的。
PA5/PRST#	IO ST / CMOS	此引脚可用做： （1）当单片机的外部复位。 （2）当端口 A 位 5，此引脚可以设定为输入口或开漏输出（open drain）模式。 请注意此引脚没有上拉或下拉电阻。这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *padier* 位 5 为”0”时，唤醒功能是被关闭的。另外，当此引脚设定成输入时，对于需要高抗干扰能力的系统，请串接 33Ω 电阻。
PA4	IO ST / CMOS	此引脚可用做端口 A 位 4，并可编程设定为输入或输出，弱上拉电阻模式。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *padier* 位 4 为”0”时，唤醒功能是被关闭的。
PA3	IO ST / CMOS	此引脚可用做端口 A 位 3，并可编程设定为输入或输出，弱上拉电阻模式。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *padier* 位 3 为”0”时，唤醒功能是被关闭的。

©Copyright 2014, PADAUK Technology Co. Ltd

引脚名称	引脚&缓冲器类型	功能描述
PA0/INT0	IO ST / CMOS	此引脚可用做： （1）当端口 A 位 0，并可编程设定为输入或输出，弱上拉电阻模式。 （2）外部引脚中断 0，中断服务可发生在上升沿或下降沿。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *padier* 位 0 为"0"时，唤醒功能是被关闭的。
PB7	IO ST / CMOS	此引脚可用做端口 B 位 7，并可编程设定为输入或输出，弱上拉电阻模式。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *pbdier* 位 7 为"0"时，唤醒功能是被关闭的。
PB6	IO ST / CMOS	此引脚可用做当端口 B 位 6，并可编程设定为输入或输出，弱上拉电阻模式。这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *pbdier* 位 6 为"0"时，唤醒功能是被关闭的。
PB5	IO ST / CMOS	此引脚可用做当端口 B 位 5，并可编程设定为输入或输出，弱上拉电阻模式。这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *pbdier* 位 5 为"0"时，唤醒功能是被关闭的。
PB2	IO ST / CMOS	此引脚可用做当端口 B 位 2，并可编程设定为输入或输出，弱上拉电阻模式。这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *pbdier* 位 2 为"0"时，唤醒功能是被关闭的。
PB1	IO ST / CMOS	此引脚可用做当端口 B 位 1，并可编程设定为输入或输出，弱上拉电阻模式。这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *pbdier* 位 1 为"0"时，唤醒功能是被关闭的。
PB0/INT1	IO ST / CMOS	此引脚可用做： （1）当端口 B 位 0，并可编程设定为输入或输出，弱上拉电阻模式。 （2）外部引脚中断 1，中断服务可发生在上升沿或下降沿 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *pbdier* 位 0 为"0"时，唤醒功能是被关闭的。
VDD		正电源
GND		地

4. 器件电气特性

4.1 直流交流电气特性

下列所有数据除特别列明外，皆于 Ta = -40 °C ~ 85 °C, Vdd=5.0V, f_{SYS}=2MHz 之条件下获得。

符 号	特 性	最小值	典型值	最大值	单位	条 件
V_{DD}	工作电压	2.2	5.0	5.5	V	
f_{SYS}	系统时钟* = IHRC/2 IHRC/4 或外部晶体振荡器 IHRC/8 或外部晶体振荡器 ILRC	 0 0 0 	 24K	 8M 4M 2M 	Hz	Under_20ms_Vdd_ok**=Y/N V_{DD}≧3.0V / V_{DD}≧3.3V V_{DD}≧2.5V / V_{DD}≧2.8V V_{DD}≧2.2V / V_{DD}≧2.2V V_{DD} = 5.0V
I_{OP}	工作电流		1.7 8		mA uA	f_{SYS}=1MIPS@5.0V f_{SYS}=ILRC~12KHz@3.3V
I_{PD}	掉电模式消耗电流 (用 *stopsys* 命令)		0.7 0.4		uA uA	f_{SYS}= 0Hz,VDD=5.0V f_{SYS}= 0Hz,VDD=3.3V
I_{PS}	省电模式消耗电流 (用 *stopexe* 命令)		0.4		mA	VDD=5.0V; Band-gap, LVD, IHRC, ILRC, Timer16 硬件模块开启.
V_{IL}	输入低电压	0		0.3V_{DD}	V	
V_{IH}	输入高电压	0.7 V_{DD}		V_{DD}	V	
I_{OL}	IO 引脚输出灌电流	7	10	13	mA	V_{DD}=5.0V, V_{OL}=0.5V
I_{OH}	IO 引脚输出驱动电流	-5	-7	-9	mA	V_{DD}=5.0V, V_{OH}=4.5V
V_{IN}	输入电压	-0.3		VDD+0.3	V	
$I_{INJ (PIN)}$	脚位的引入电流			1	mA	VDD+0.3≧V_{IN}≧ -0.3
R_{PH}	上拉电阻		62 100 210		KΩ	V_{DD}=5.0V V_{DD}=3.3V V_{DD}=2.2V
V_{LVD}	低电压侦测电压 *	3.86 3.35 2.84 2.61 2.37 2.04 1.86 1.67	4.15 3.60 3.05 2.80 2.55 2.20 2.00 1.80	4.44 3.85 3.26 3.00 2.73 2.35 2.14 1.93	V	
f_{IHRC}	IHRC 输出频率(校准后) *	15.68*	16*	16.32*	MHz	@25°C
		14.72*	16*	17.28*	MHz	VDD=2.2V~5.5V, -40°C <Ta<85°C*

符 号	特 性	最小值	典型值	最大值	单位	条 件
f_{ILRC}	ILRC 输出频率 *	20.4*	24*	27.6*	KHz	VDD=5.0V, Ta=25°C
		15.6*	24*	32.4*		VDD=5.0V, -40°C <Ta<85°C*
		10.2*	12*	13.8*		VDD=3.3V, Ta=25°C
		7.8*	12*	16.2*		VDD=3.3V, -40°C <Ta<85°C*
t_{INT}	中断脉冲宽度	ˇ30			ns	VDD=5V
V_{DR}	数据存储器数据保存电压*	1.5			V	掉电模式下
t_{WDT}	看门狗定时器超时溢出时间		2048		T_{ILRC}	misc[1:0]=00 (默认)
			4096			misc[1:0]=01
			16384			misc[1:0]=10
			256			misc[1:0]=11
t_{SBP}	系统开机时间(从开启电源算起)		1024		T_{ILRC}	T_{ILRC} 是 ILRC 的时钟周期
t_{WUP}	系统唤醒时间					
	STOPEXE 省电模式下，切换 IO 引脚的快速唤醒		128		T_{SYS}	T_{SYS} 是系统时钟周期
	STOPSYS 掉电模式下，切换 IO 引脚的快速唤醒；IHRC 是系统时钟		128 T_{SYS} + T_{SIHRC}			T_{SIHRC} 是 IHRC 从上电后的稳定时间，在 5V 下约 5us.
	STOPSYS 掉电模式下，切换 IO 引脚的快速唤醒；ILRC 是系统时钟		128 T_{SYS} + T_{SILRC}			T_{SILRC} 是 ILRC 从上电后的稳定时间，在 5V 下约 43ms
	STOPEXE 省电模式和 STOPSYS 掉电模式下，切换 IO 引脚的普通唤醒		1024		T_{ILRC}	T_{ILRC} 是 ILRC 时钟周期
t_{RST}	外部复位脉冲宽度	120			us	@VDD=5V

*这些参数是设计参考值，并不是每个芯片测试。

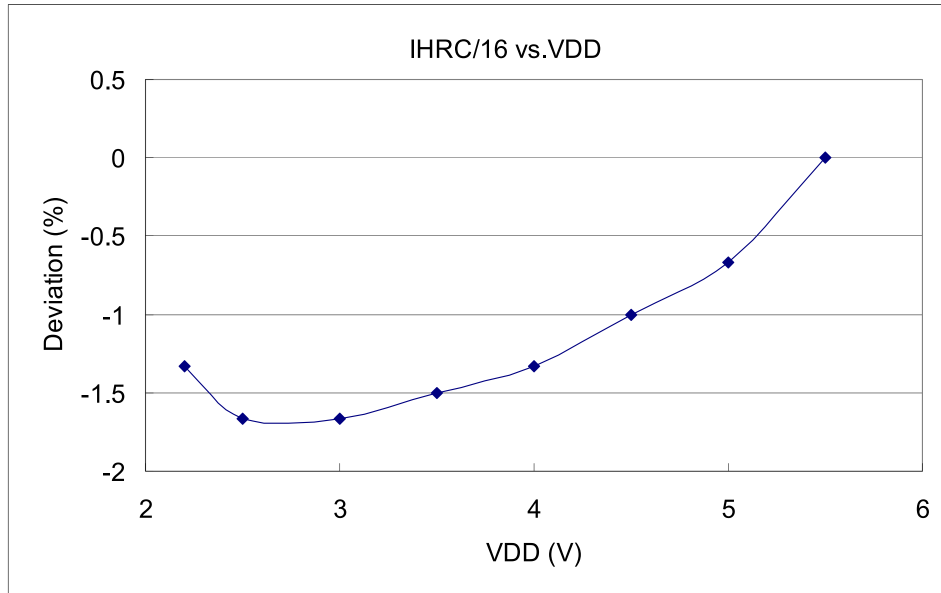
** Under_20ms_Vdd_Ok 为对 VDD 能否于 20ms 内从 0V 上升到指定电压的一个检查条件

4.2 绝对最大值

- 电源电压 ………………………………...... 2.2V ~ 5.5V
- 输入电压 …………………………………... -0.3V ~ VDD + 0.3V
- 工作温度 ………………...………………… -40°C ~ 85°C
- 储藏温度 …………………………...……… -50°C ~ 125°C

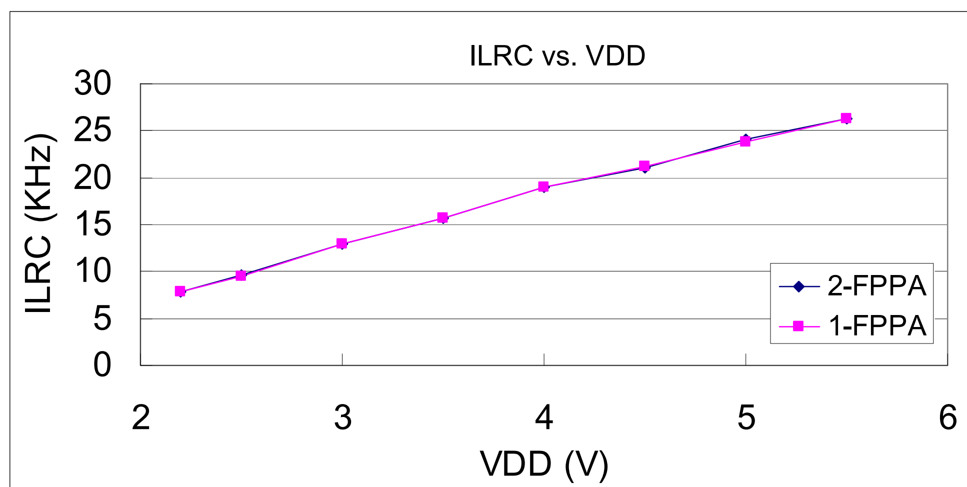
©Copyright 2014, PADAUK Technology Co. Ltd

4.3 IHRC 频率与 VDD 关系曲线图

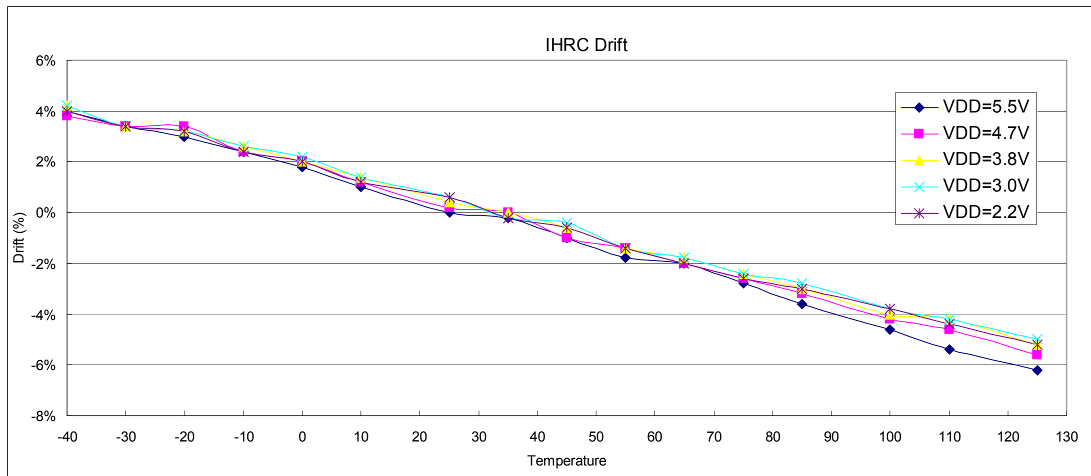


请注意：在不同的系统时钟下，频率的漂移会稍微不同

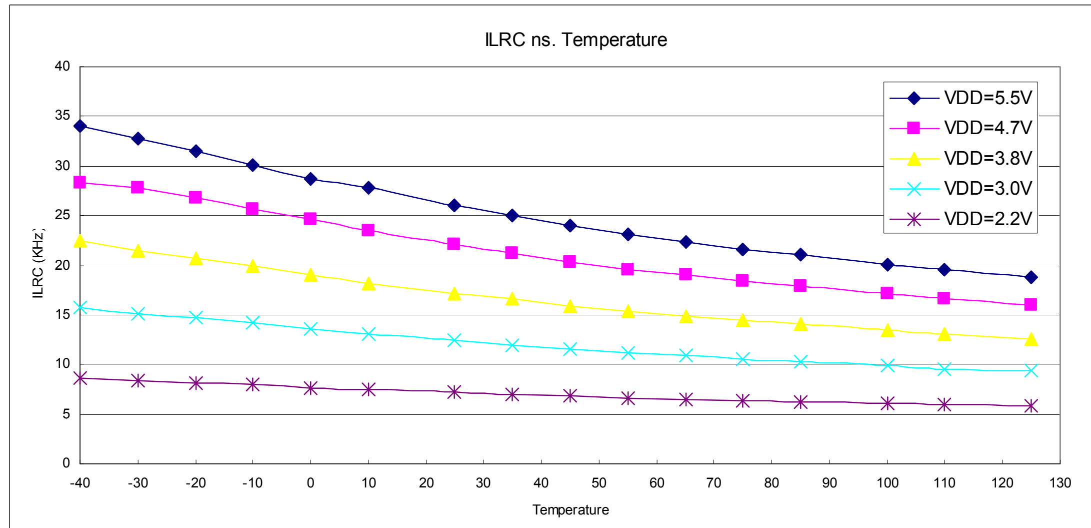
4.4 ILRC 频率与 VDD 关系曲线图



4.5 IHRC 频率与温度关系曲线图



4.6 ILRC 频率与温度关系曲线图

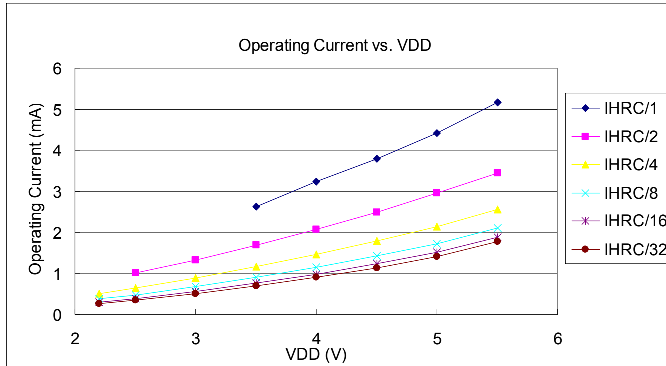


©Copyright 2014, PADAUK Technology Co. Ltd

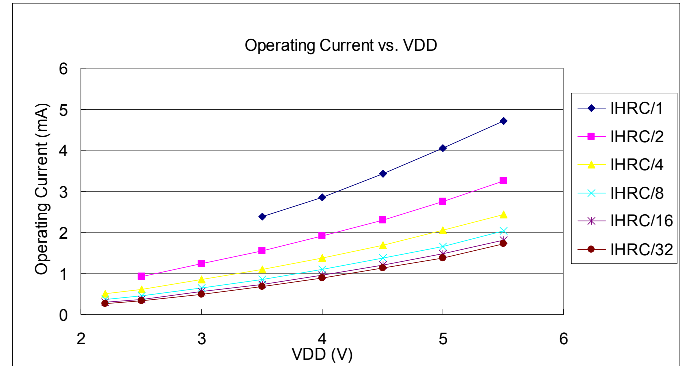
4.7 工作电流与 VDD、系统时钟 CLK=IHRC/n 曲线图

条件：开启的硬件模块：Band-gap, LVD, IHRC, T16；关闭的硬件模块：ILRC, EOSC;
IO 引脚：PA0 以 0.5Hz 频率高低电压交换输出，无负载；其它引脚：设为输入而且不浮接

两个 FPP 单元工作模式



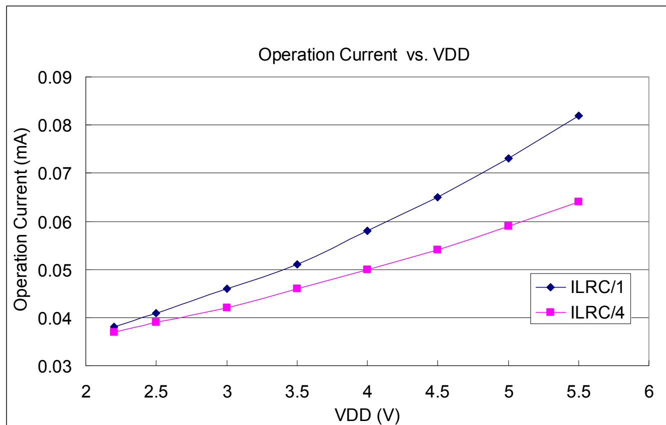
单一 FPP 单元工作模式



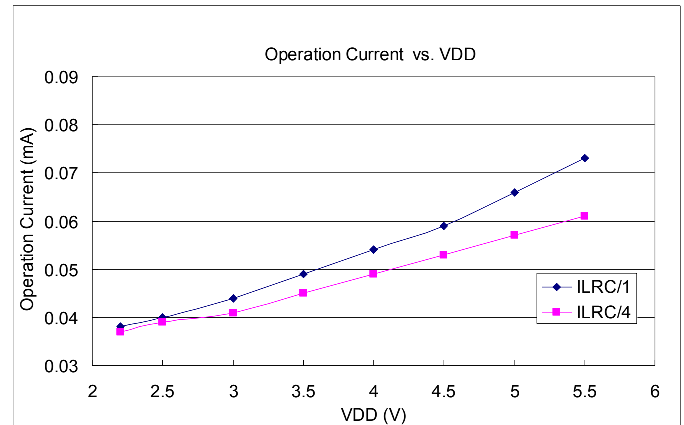
4.8 工作电流与 VDD、系统时钟 CLK=ILRC/n 曲线图

条件：开启的硬件模块：Band-gap, LVD, ILRC, T16；关闭的硬件模块：IHRC, EOSC;
IO 引脚：PA0 以 0.5Hz 频率高低电压交换输出，无负载；其它引脚：设为输入而且不浮接

两个 FPP 单元工作模式



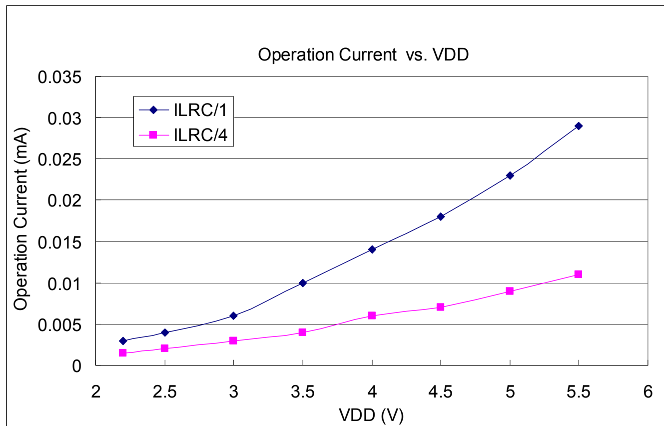
单一 FPP 单元工作模式



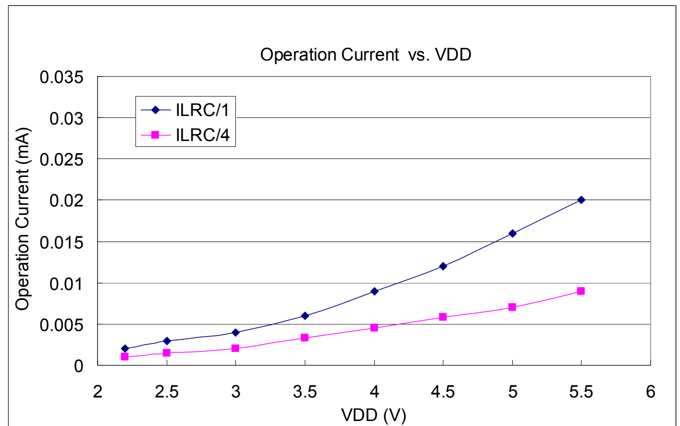
4.9 最低工作电流与 VDD、系统时钟 CLK=ILRC/n 曲线图

条件：开启的硬件模块：ILRC, T16；关闭的硬件模块：IHRC, Band-gap, LVD, EOSC;
IO 引脚：PA0 以 0.5Hz 频率高低电压交换输出，无负载；其它引脚：设为输入而且不浮接

两个 FPP 单元工作模式



单一 FPP 单元工作模式

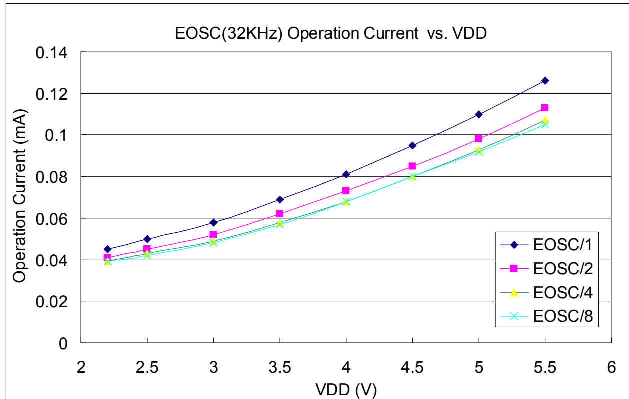


©Copyright 2014, PADAUK Technology Co. Ltd

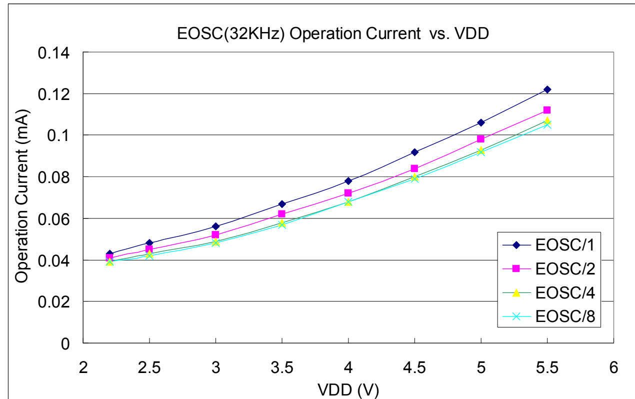
4.10 工作电流与 VDD、系统时钟 CLK=32KHz EOSC/n 曲线图

条件：开启的硬件模块：EOSC, Band-gap, LVD, T16；关闭的硬件模块：IHRC, ILRC;
IO 引脚：PA0 以 0.5Hz 频率高低电压交换输出，无负载；其它引脚：设为输入而且不浮接
EOSC: 高驱动电流

两个 FPP 单元工作模式

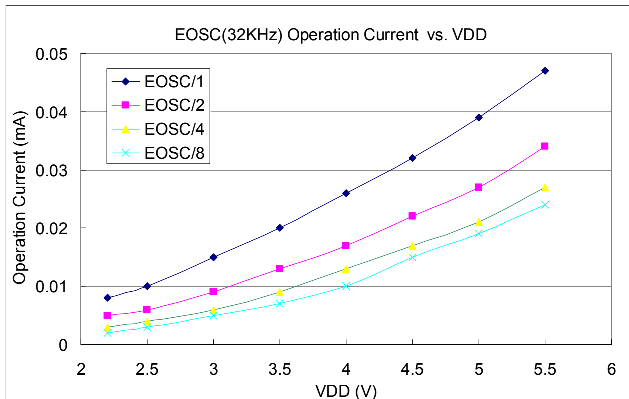


单一 FPP 单元工作模式

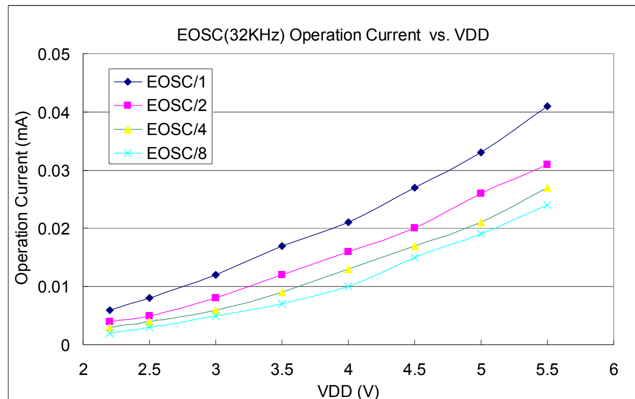


条件：开启的硬件模块：EOSC, T16；关闭的硬件模块：IHRC, ILRC, Band-gap, LVD;
IO 引脚：PA0 以 0.5Hz 频率高低电压交换输出，无负载；其它引脚：设为输入而且不浮接
EOSC: 低驱动电流

两个 FPP 单元工作模式



单一 FPP 单元工作模式

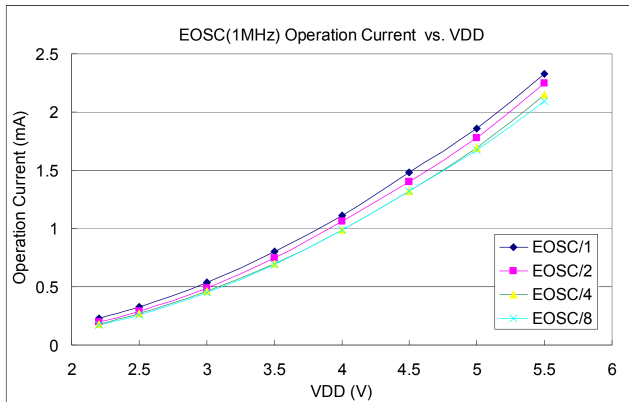


©Copyright 2014, PADAUK Technology Co. Ltd

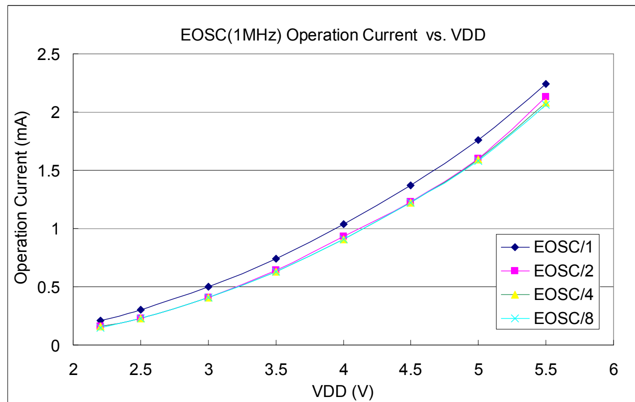
4.11 工作电流与 VDD、系统时钟 CLK=1MHz EOSC/n 曲线图

条件：开启的硬件模块：EOSC, Band-gap, LVD, T16; 关闭的硬件模块：IHRC, ILRC;
IO 引脚：PA0 以 0.5Hz 频率高低电压交换输出，无负载；其它引脚：设为输入而且不浮接
EOSC: 高驱动电流

两个 FPP 单元工作模式

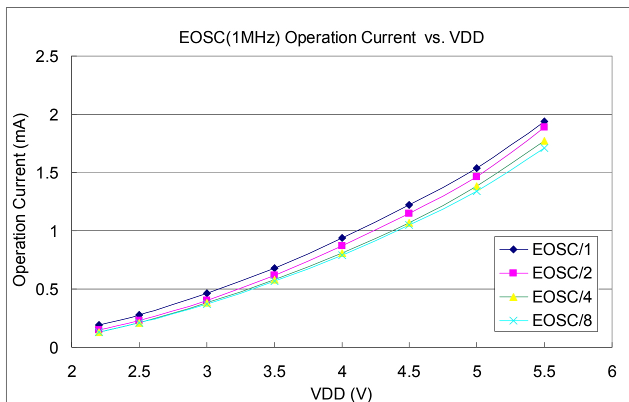


单一 FPP 单元工作模式

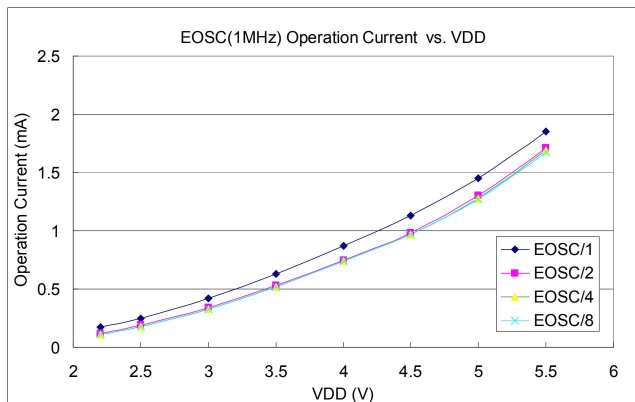


条件：开启的硬件模块：EOSC, T16; 关闭的硬件模块：IHRC, ILRC, Band-gap, LVD;
IO 引脚：PA0 以 0.5Hz 频率高低电压交换输出，无负载；其它引脚：设为输入而且不浮接
EOSC: 低驱动电流

两个 FPP 单元工作模式



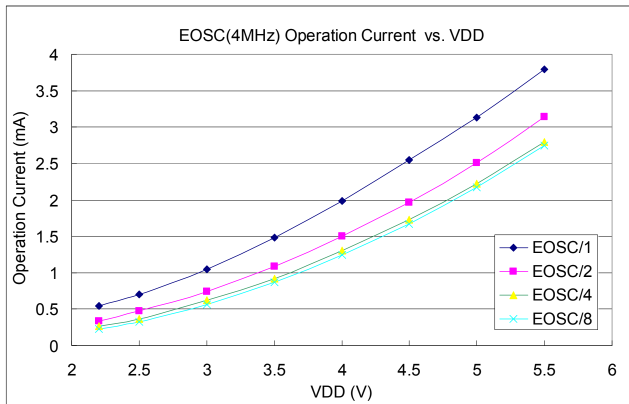
单一 FPP 单元工作模式



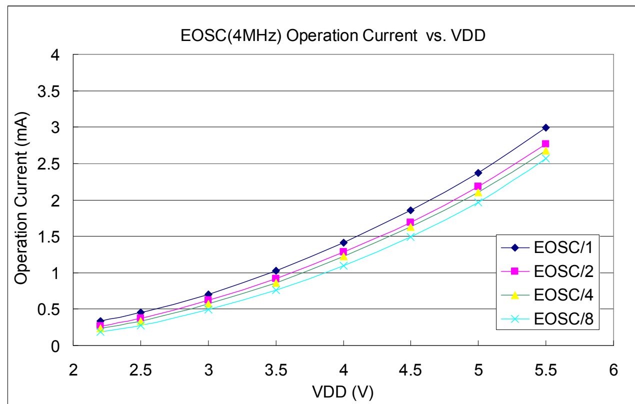
4.12 工作电流与 VDD、系统时钟 CLK=4MHz EOSC/n 曲线图

条件：开启的硬件模块：EOSC, Band-gap, LVD, T16; 关闭的硬件模块：IHRC, ILRC;
IO 引脚：PA0 以 0.5Hz 频率高低电压交换输出，无负载；其它引脚：设为输入而且不浮接
EOSC: 高驱动电流

两个 FPP 单元工作模式

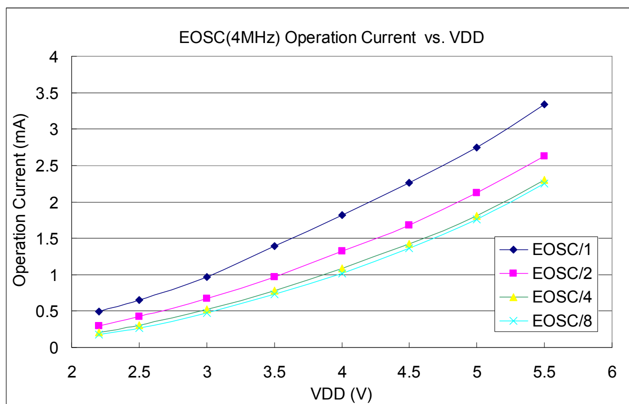


单一 FPP 单元工作模式

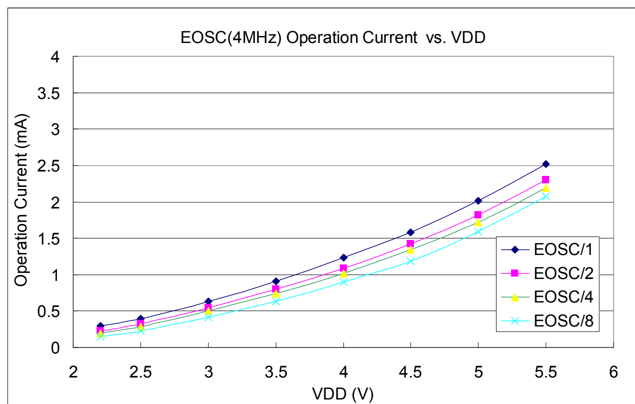


条件：开启的硬件模块：EOSC, T16; 关闭的硬件模块：IHRC, ILRC, Band-gap, LVD;
IO 引脚：PA0 以 0.5Hz 频率高低电压交换输出，无负载；其它引脚：设为输入而且不浮接
EOSC: 低驱动电流

两个 FPP 单元工作模式

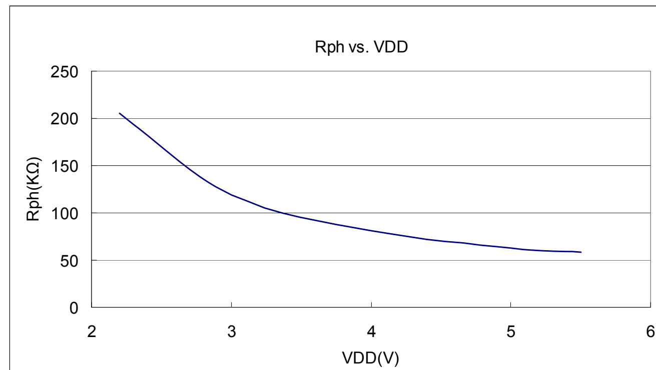


单一 FPP 单元工作模式

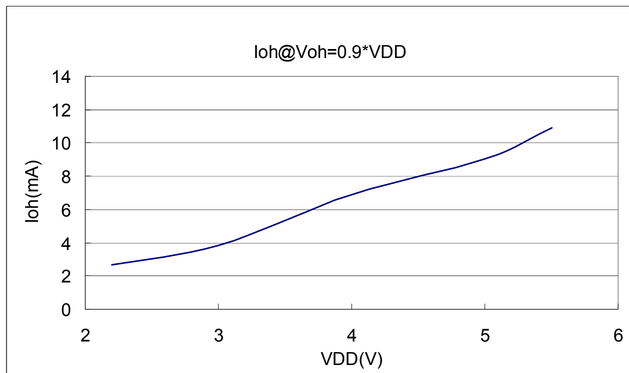


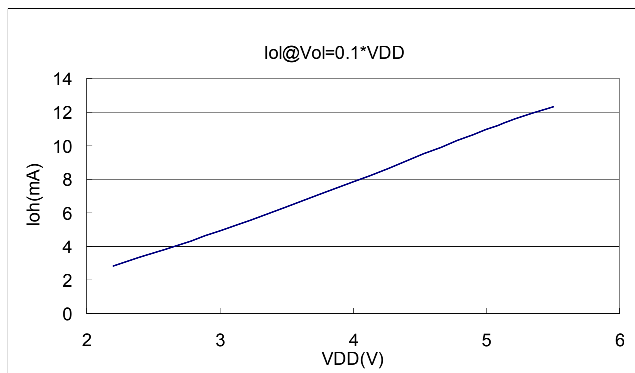
©Copyright 2014, PADAUK Technology Co. Ltd

4.13 引脚拉高电阻曲线图

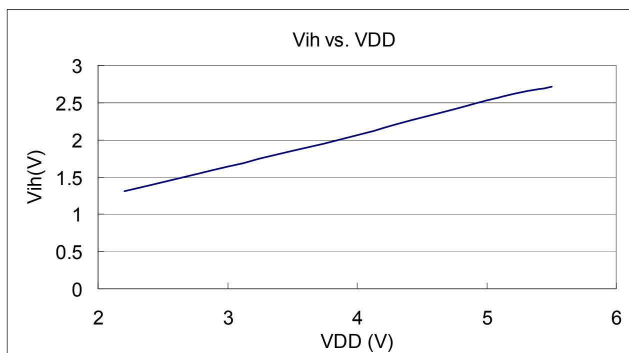


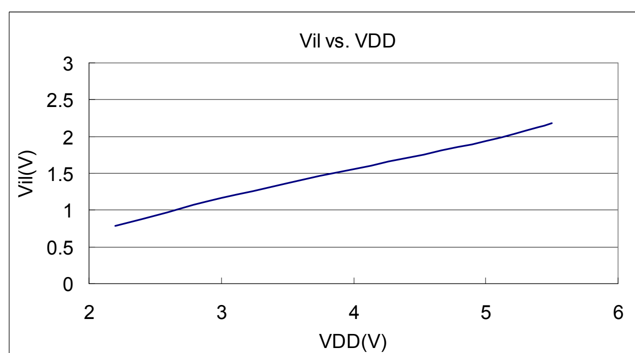
4.14 引脚输出驱电流(Ioh)与灌电流(Iol) 曲线图





4.15 引脚输出输入高电压与低电压(V_{IH} / V_{IL}) 曲线图





5. 功能概述

5.1 处理单元

PMC251 内有两个处理单元：FPP0 和 FPP1，每一个 FPP 单元分享一半的系统性能。在每一个处理单元中包括：（1）其本身的程序计数器来控制程序执行的顺序（2）自己的堆栈指针用来存储或恢复程序计数器的程序执行（3）自己的累加器（4）标志寄存器以记录程序执行的状态。基于这样的架构，FPP 单元可以独立执行自己的程序，当两个 FPP 单元都独立执行自己的程序，就是达到并行处理的结果。每一个 FPP 单元可以通过设置 FPP 单元允许寄存器来允许或禁止执行，在上电复位后只有 FPP0 是启用的。系统初始化将从 FPP0 开始，而 FPP1 可以由使用者的程序来决定是否使用。

PMC251 的两个处理单元共享相同 1Kx16 位 OTP 程序存储器、60 字节的数据存储器和所有的 IO 端口，这两个处理单元是在相互交错的时钟周期下工作，以避免干扰。芯片内部有一个任务切换硬件模块，用来决定该时间周期是要让那一个相应的 FPP 单元执行。硬件框图和基本时序图如图 1 所示，FPP0 单元在每两系统时钟执行一次程序，执行属于 FPP0 单元的程序，图中显示为第(M-1)个，第 M 个和第(M +1)个指令，相同的，FPP1 单元在每两系统时钟执行一次程序，执行属于 FPP1 单元的程序，图 1.中显示为第(N-1)个，第 N 个和第(N +1)个指令。

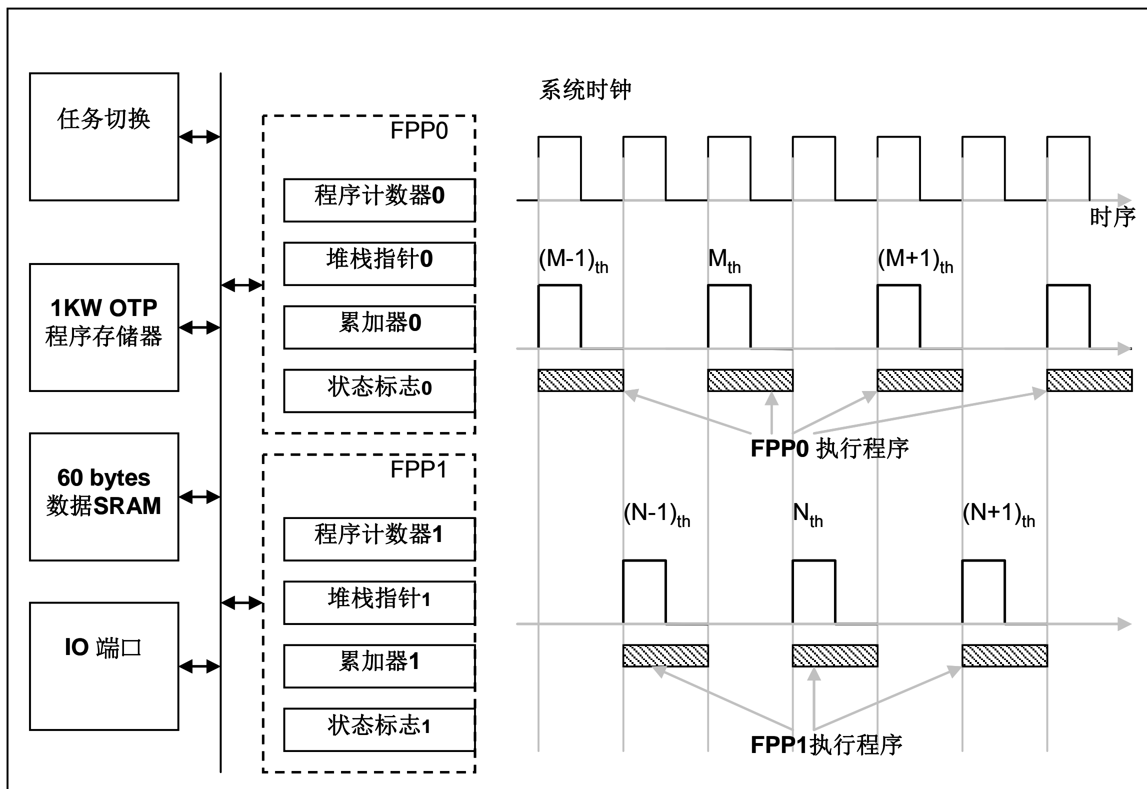


图 1：硬件 FPP 内部硬件和时序图

每个 FPP 单元有一半整个系统的计算能力;例如，如果系统时钟为 8MHz，则 FPP0 和 FPP1 同时在 4MHz 系统时钟下同时工作。 FPP 单元可以利用寄存器来启用或禁用，但是在电源复位后，只有 FPP0 被启用。系统初始化将从 FPP0 开始，如果需要的话，再用 FPP0 来启用 FPP1 单元。FPP0 和 FPP1 可以用来启用或禁用任何一个 FPP 单元。

©Copyright 2014, PADAUK Technology Co. Ltd

5.1.1 程序计数器

程序计数器（PC）记录下一个执行指令的地址，在每个指令周期后程序计数器会自动递增，以便指令码顺序从程序存储器取出。某些指令，如分支指令和子程序调用都会改变顺序并放入一个新值到程序计数器。PMC251 程序计数器的位长度是 10 位。在硬件复位后，FPP0 的程序计数器为 0、FPP1 为 1。当中断发生时，程序计数器会跳转到' h10 的中断服务程序处只有 FPP0 接受中断，只有 FPP0 才会接受中断。FPP0 和 FPP1 都具有各自独立的程序计数器来控制其程序执行顺序。

5.1.2 堆栈指针

在每个处理单元的堆栈指针是用来指引堆栈存储器的顶部，该处是用来存储子程序的局部变量和参数的地方；堆栈指针寄存器（SP）的地址是 IO 0x02h。如果堆栈是一个“满”的堆栈状态，该堆栈指针指向最后入栈的位置；否则，如果它是一个“空”的堆栈状态，该堆栈指针指向第一个空的位置，在那里将是下一个压栈的位置。堆栈指针位数字是 8 位;栈内存取不能超过 64 个字节，并应在从 0x00h 地址定义 64 字节。PMC251 的堆栈存储器中，每个 FPP 单元都可以由使用者通过堆栈指针寄存器做编程分配，每个 FPP 单元堆栈指针的深度可调，以利于优化系统性能。下面例子就是如何再汇编(ASM)下定义堆栈：

```
        . ROMADR            0
        GOTO                FPPA0
        GOTO                FPPA1
        ...
        . RAMADR            0    // 地址必须小于 0x100
        WORD                Stack0 [1]
        WORD                Stack1 [2]
        ...
    FPPA0:
        SP =  Stack0;       // 指派 Stack0 给 FPPA0, 需 1 层呼叫因为 Stack0[1]
        ...
        call   function1
        ...
    FPPA1:
        SP =  Stack1;       //指派 Stack1 给 FPPA1, 需 2 层呼叫因为 Stack1[2]
        ...
        call   function2
        ...
```

由于堆栈使用常容易有错，建议使用者多利用 Mini-C；在 Mini-C，堆栈的计算是由系统软件（IDE）自动完成，使用者不需要考虑堆栈的计算，例子如下所示：

```
void     FPPA0 (void)
{
...
}
```

在 IDE 中，如下图的窗口，使用者可以检查堆栈的分配，图 2.显示出在 FPP0 执行前，堆栈的状态；系统软件已计算出所需的堆栈空间，并已预留给程序。

©Copyright 2014, PADAUK Technology Co. Ltd

```
20:
21:   void     FPPA0 (void)
00000000    C028            GOTO  0x28
```

```
21:   void     FPPA0 (void)
22:   {
00000028    0030            WDRESET
00000029    1F00            MOV  A  #0x0
0000002A    0082            MOV  SP  A
```

图 2：在 Mini-C 程序的堆栈分配

5.1.3 一个处理单元工作模式

传统的单片机使用者如果不需要有并行处理能力的单片机，PMC251 除了具有平行处理能力的双处理单元工作外，并提供使用者可以选择的一个处理单元工作模式，它的表现就如同传统的单片机。当一个处理单元工作模式被选中后，FPP1 始终禁用，只有 FPP0 是使能的。图 3 显示了每个 FPP 单元的时序图，FPP1 总是禁用，只 FPP0 活跃。请注意在一个处理单元工作模式下，是不支持等待(*wait*)指令。

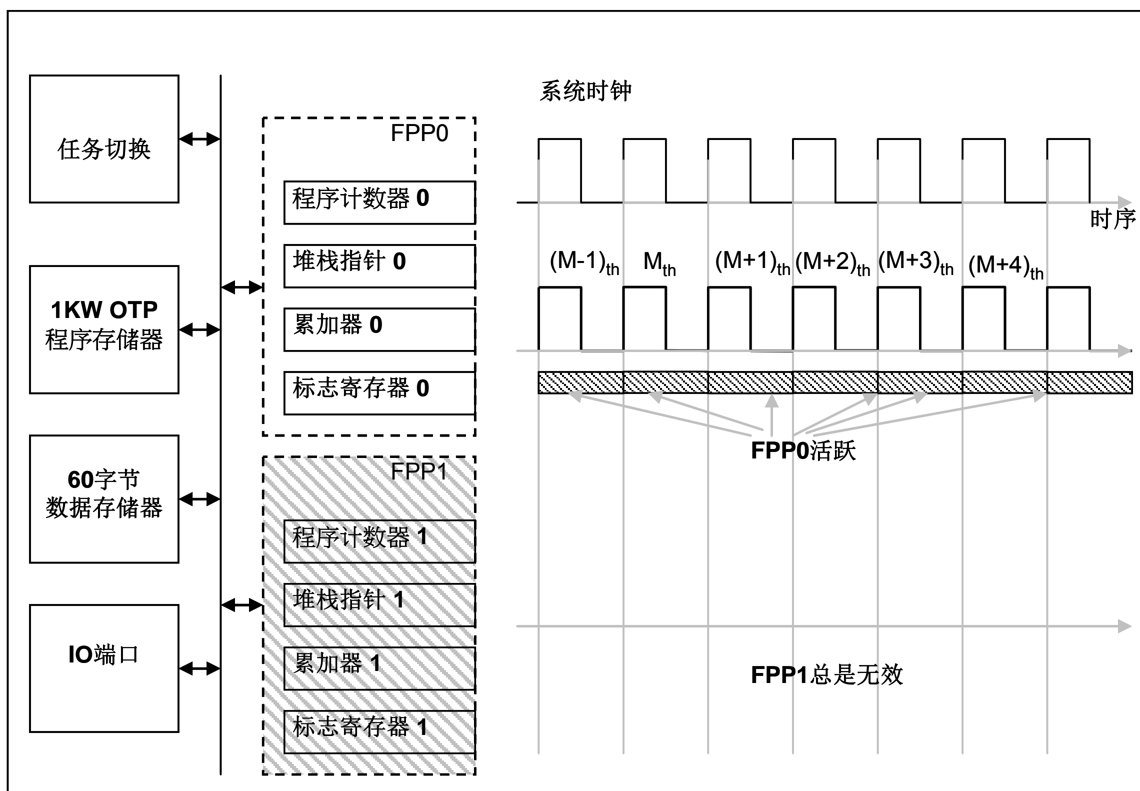


图 3：一个处理单元工作模式下的时序

©Copyright 2014, PADAUK Technology Co. Ltd

5.2 OTP 程序存储器

5.2.1 程序存储器分配

OTP（一次性可编程）程序存储器用来存放要执行的程序指令。FPP0 和 FPP1 的所有程序代码都存储在这个 OTP 存储器。OTP 程序存储器可以储存数据，包含：数据，表格和中断入口。复位之后，FPP0 的初始地址为'h0，FPP1 的初始地址为'h1。中断入口是'h10，只有 FPP0 能使用中断功；OTP 程序存储器最后 8 个地址空间是被保留给系统使用，如：校验，序列号等。PMC251 的 OTP 程序存储器结构是 1Kx16 位，如表 1 所示。OTP 存储器从地址 “h3F8~h3FF”供系统使用，从 “h002 ~ h00F”和 “h011~h3F7”地址空间是使用者的程序空间。地址'h001 是 FPP1 的初始地址；另外，两个处理单元工作模式或一个处理单元工作模式，FPP0 的初始地址都是'h000。

地址	功能
'h000	FPP0 起始地址 – *goto* 指令
'h001	FPP1 起始地址 – *goto* 指令
'h002	使用者程序区
•	•
'h00F	使用者程序区
'h010	中断入口地址
'h011	使用者程序区
•	•
'h3F7	使用者程序区
'h3F8	系统使用
•	•
'h3FF	系统使用

表 1：PMC251 程序存储器结构

5.2.2 两个处理单元工作模式下程序存储器分配例子

表 2 显示了一个例子，使用两个处理单元工作模式下，程序存储器分配情形：

地址	功能
000	FPP0 起始地址 – *goto* 指令(goto 'h020)
001	FPP1 程序开始
•	•
00F	goto 'h1A1 继续 FPP1 程序
010	中断入口地址(只给 FPP0)
•	•
01F	中断程序结束
020	FPP0 程序开始
•	•
1A0	FPP0 程序结束
1A1	继续 FPP1 程序
•	•
3F7	FPP1 程序结束
3F8	系统使用
•	•
3FF	系统使用

表 2：两个处理单元工作模式之程序存储器分配案例

©Copyright 2014, PADAUK Technology Co. Ltd

5.2.3 一个处理单元工作模式下程序存储器分配例子

表 3 显示了一个例子，使用一个处理单元工作模式下，程序存储器分配情形，整个使用者程序存储器都可以被分配到 FPP0。

地址	功能
000	FPP0 起始地址
001	FPP0 程序开始
002	使用者程序区
•	•
00F	Goto 指令(goto ‘h020)
010	中断入口地址
011	中断程序
•	•
01F	中断程序结束
020	使用者程序区
•	•
•	•
3F7	使用者程序区
3F8	系统使用
•	•
3FF	系统使用

表 3：一个处理单元工作模式之程序存储器分配案例

©Copyright 2014, PADAUK Technology Co. Ltd

5.3 程序结构

5.3.1 两个处理单元工作模式下程序结构

开机后，FPP0 和 FPP1 的程序开始地址分别是’h000 和’h001。中断服务程序的入口地址是’h010，而且只有 FPP0 才能接受中断服务。 PMC251 的基本软件结构如图 4 所示。两个 FPP 的处理单元的程序代码是被放置在同一个程序空间。除了初始地址和中断入口地址外，处理单元的程序代码可以放在程序存储器任何位置，并没有在特定的地址;开机后，将首先执行 fpp0Boot，其中将包括系统初始化和启用其它 FPP 的单元。

```
.romadr  0x00
// Program Begin
goto          fpp0Boot;
goto          fpp1Boot;
//------中断服务程序----------------
.romadr  0x010
              pushaf ;
              t0sn    intrq.0;   //PA.0 ISR
              goto    ISR_PA0;
              t0sn    intrq.1;   //PB.0 ISR
…
              goto    ISR_PB0;
//------中断服务程序结束--------
//------ FPP0程序开始----------
fpp0Boot :
//--- FPP0初始化…
…
fpp0Loop:
…
  goto fpp0Loop:
//------FPP0程序结束--------
//------FPP1程序开始----------
fpp1Boot :
//---FPP1初始化…
…
fpp1Loop:
…
  goto fpp1Loop:
//---------FPP1程序结束--------
```

图 4：两个处理单元工作模式之程序结构

5.3.2 一个处理单元工作模式下程序结构

开机后，FPP0 的程序开始地址是’h000，中断服务程序的入口地址是’h010，一个处理单元工作模式下的程序结构与传统的单片机软件结构相同，开机后，程序将从地址’h000 然后继续程序的顺序。

©Copyright 2014, PADAUK Technology Co. Ltd

5.4 开机流程

开机时，POR（上电复位）是用于复位 PMC251；但是，上电后电源电压可能不太稳定，为确保单片机是工作在电压稳定的状态，在执行第一条指令之前，PMC251 会延迟 1024 个 ILRC 时钟周期，这时间就是 t_{SBP}，如图 5 所示。

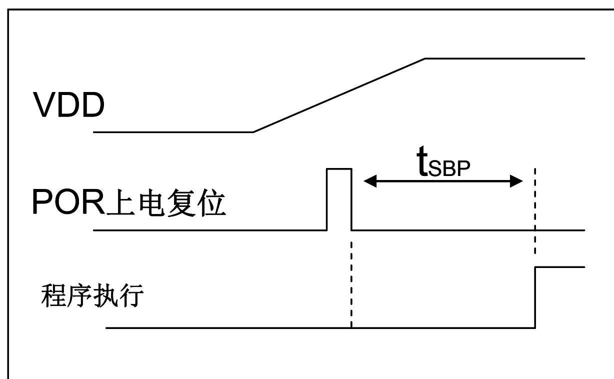


图 5：上电复位时序

图 6 显示了开机后的典型程序流程，特别注意到的是，在上电复位后，FPP1 是被禁用的，建议在 FPP0 完成系统初始化之前，不要使用 FPP1。

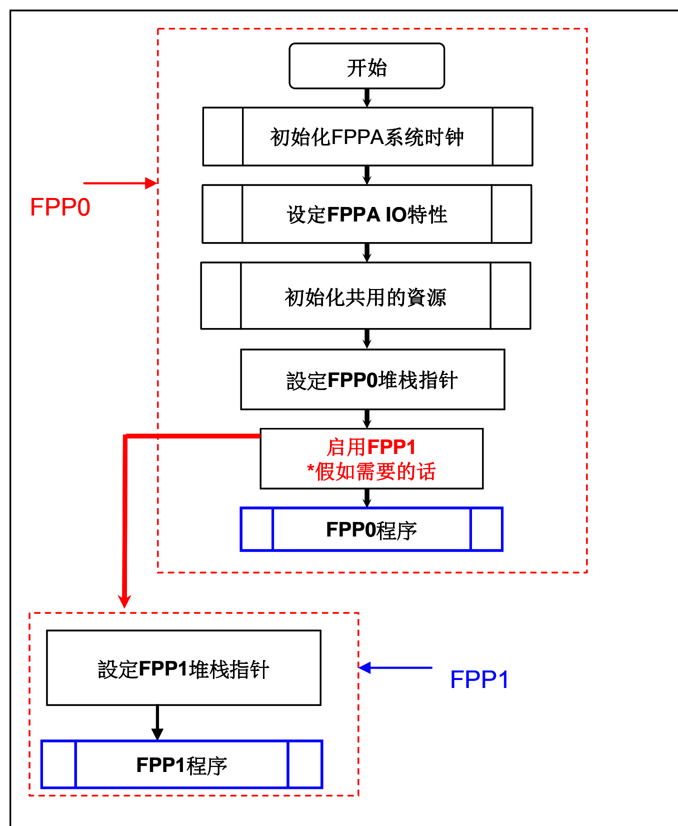


图 6：开机流程

5.5 数据存储器 - SRAM

图 7 显示 PMC251 数据存储器(SRAM)的架构，所有的数据存储器都可以任由 FPP0 和 FPP1 存取，绝大部分数据存储器的存取都在 1T 时钟周期完成，数据存取可以是字节或位的操作。除了存储数据外，数据存储器还可以担任间接存取方式的数据指针，以及所有 FPP 处理单元的堆栈存储器。

每个 FPP 处理单元的堆栈存储器是独立不相干的，堆栈存储器的堆栈指针是定义在堆栈指针寄存器；使用者可以依其程序需求来订定每一个 FPP 处理单元所需要堆栈存储器的大小，以保持最大的弹性。

数据存储器的间接存取方式，是以数据存储器当作数据指针来存取数据字节。所有的数据存储器，都可以拿来当作数据指针，这可以让单片机的资源做最大的使用。由于数据宽度为 8 位，数据存储器的间接存取范围必须在 256 字节以内，由于 PMC251 的数据存储器只有 60 字节，所以全部都可以用间接方式来存取。

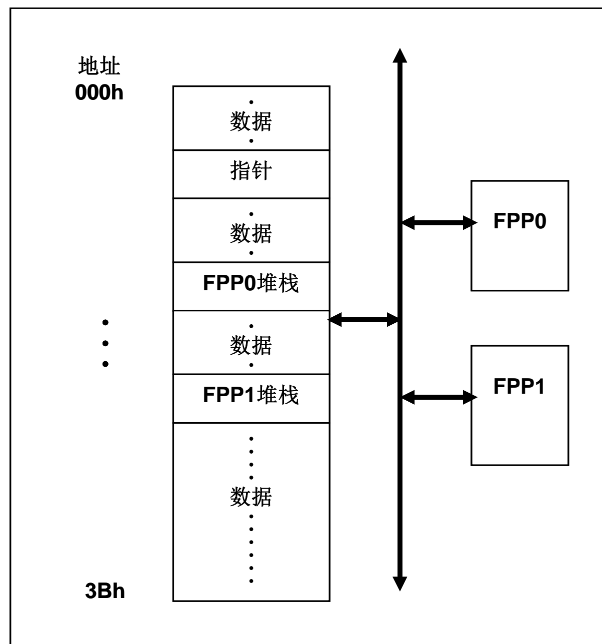


图 7：数据存储器架构

©Copyright 2014, PADAUK Technology Co. Ltd

5.6 算术和逻辑单元

算术和逻辑单元（ALU）是用来作整数算术、逻辑、转移和其它特殊运算的单元。运算的数据来源可以从指令、累加器或 SRAM 数据存储器，计算结果可写入累加器或 SRAM。FPP0 和 FPP1 两个处理单元共享算术和逻辑单元。

5.7 振荡器和时钟

PMC251 提供 3 个振荡器电路：外部晶体振荡器（EOSC）、内部高频振荡器（IHRC）、内部低频振荡器（ILRC ）。这三个振荡器可以分别用寄存器 eoscr.7、clkmd.4、clkmd.2 启用或禁用，使用者可以选择这三个振荡器之一作为系统时钟源，并透过 *clkmd* 寄存器来改变系统时钟频率，以满足不同的系统应用。

振荡器硬件	启用或禁用选择	开机后默认值
EOSC	**eoscr**.7	禁用
IHRC	**clkmd**.4	启用
ILRC	**clkmd**.2	启用

5.7.1 内部高频振荡器和内部低频振荡

开机后，IHRC 和 ILRC 振荡器都是被启用的，PMC251 烧录工具提供 IHRC 频率校准，透过 *ihrcr* 寄存器来消除工厂生产引起的频率漂移，IHRC 振荡器通常被校准到 16MHz，通常校准后的频率偏差都在 2％以内；除了工厂生产造成 IHRC 频率漂移外，IHRC 的频率仍然会因电源电压和工作温度而略有漂移；在 VDD =2.2V～5.5V，40℃～85℃的条件下，漂移率约为±6％，请参阅 IHRC 频率和 VDD、温度的测量图表。ILRC 的频率是 20 kHz 左右，但是，其频率会因工厂生产、电源电压和温度而变化，PMC251 不提供 ILRC 频率校准功能，请参阅 ILRC 频率和 VDD、温度的测量图表，需要精确定时的应用请不要使用 ILRC 的时钟当作参考时间。

5.7.2 芯片校准

IHRC 的输出频率可能因工厂制造变化而有所差异，PMC251 提供 IHRC 输出频率校准，来消除工厂生产时引起的变化。这个功能是在编译使用者的程序时序做选择，校准命令以及选项将自动插入到用使用者的程序，校准命令如下所示：

ADJUST_IC   SYSCLK=IHRC/(**p1**), IHRC=(**p2**)MHz, VDD=(**p3**)V;

这里：

p1 =2，4，8，16，32；以提供不同的系统时钟。

p2 =16～18; 校准芯片到不同的频率，通常选择 16MHz。

p3 =2.5～5.5; 根据不同的电源电压校准芯片。

5.7.3 IHRC 校准

使用者在程序编译期间，IHRC 频率校准以及系统时钟的选项，如表 4 所示：

SYSCLK	CLKMD	IHRCR	描述
○ Set IHRC / 2	= 34h (IHRC / 2)	Calibrated	IHRC 校准到 16MHz, CLK=8MHz (IHRC/2)
○ Set IHRC / 4	= 14h (IHRC / 4)	Calibrated	IHRC 校准到 16MHz, CLK=4MHz (IHRC/4)
○ Set IHRC / 8	= 3Ch (IHRC / 8)	Calibrated	IHRC 校准到 16MHz, CLK=2MHz (IHRC/8)
○ Set IHRC / 16	= 1Ch (IHRC / 16)	Calibrated	IHRC 校准到 16MHz, CLK=1MHz (IHRC/16)
○ Set IHRC / 32	= 7Ch (IHRC / 32)	Calibrated	IHRC 校准到 16MHz, CLK=0.5MHz (IHRC/32)
○ Set ILRC	= E4h (ILRC / 1)	Calibrated	IHRC 校准到 16MHz, CLK=ILRC
○ Disable	No change	No Change	IHRC 不校准, CLK 没改变

表 4：IHRC 频率校准选项

通常情况下，ADJUST_IC 将是开机后的第一个命令，以设定系统的工作频率。IHRC 频率校准的程序只会执行一次，是发生在要将程序代码在写入 OTP 存储器的时候，以后，它就不会再被执行了。如果 IHRC 校准选择不同的选项，开机后的系统状态也是不同的。下面显示在不同的选项下，PMC251 不同的状态：

(1) .ADJUST_IC SYSCLK=IHRC/2, IHRC=16MHz, VDD=5V

开机后, CLKMD = 0x34：

- IHRC 的校准频率为 16MHz@VDD=5V，启用 IHRC 的硬件模块
- 系统时钟 CLK = IHRC/2 = 8MHz
- 看门狗定时器被禁止，启用 ILRC, PA5 是在输入模式

(2) .ADJUST_IC SYSCLK=IHRC/4, IHRC=16MHz, VDD=3.3V

开机后, CLKMD = 0x14：

- IHRC 的校准频率为 16MHz@VDD=3.3V，启用 IHRC 的硬件模块
- 系统时钟 CLK = IHRC/4 = 4MHz
- 看门狗定时器被禁止，启用 ILRC, PA5 是在输入模式

(3) .ADJUST_IC SYSCLK=IHRC/8, IHRC=16MHz, VDD=2.5V

开机后, CLKMD = 0x3C：

- IHRC 的校准频率为 16MHz@VDD=2.5V，启用 IHRC 的硬件模块
- 系统时钟 CLK = IHRC/8 = 2MHz
- 看门狗定时器被禁止，启用 ILRC, PA5 是在输入模式

(4) .ADJUST_IC SYSCLK=IHRC/16, IHRC=16MHz, VDD=2.5V

开机后, CLKMD = 0x1C：

- IHRC 的校准频率为 16MHz@VDD=2.5V，启用 IHRC 的硬件模块
- 系统时钟 CLK = IHRC/16 = 1MHz
- 看门狗定时器被禁止，启用 ILRC, PA5 是在输入模式

(5) .ADJUST_IC SYSCLK=IHRC/32, IHRC=16MHz, VDD=5V

开机后, CLKMD = 0x7C：

- IHRC 的校准频率为 16MHz@VDD=5V，启用 IHRC 的硬件模块
- 系统时钟 CLK = IHRC/32 = 500KHz
- 看门狗定时器被禁止，启用 ILRC, PA5 是在输入模式

©Copyright 2014, PADAUK Technology Co. Ltd

(6) .ADJUST_IC   SYSCLK=ILRC, IHRC=16MHz, VDD=5V

开机后, CLKMD = 0XE4：

- IHRC 的校准频率为 16MHz@VDD=5V，启用 IHRC 的硬件模块
- 系统时钟 CLK = ILRC
- 看门狗定时器被~~启用~~禁止，启用 ILRC, PA5 是在输入模式

(7) .ADJUST_IC   DISABLE

开机后, CLKMD is not changed (Do nothing)：

- IHRC 不校准，禁用 IHRC 的硬件模块
- 系统时钟 CLK = ILRC
- 看门狗定时器被启用，启用 ILRC, PA5 是在输入模式

5.7.4 外部晶体振荡器

如果要使用晶体振荡器， 就需要再在 X1 和 X2 之间放置晶体或谐振器。图 8 显示了使用晶体振荡器的的硬件连接；晶体振荡器的工作频率范围可以从 32 千赫至 4 兆赫，取决于放置的晶体，PMC251 不支持比 4 兆赫更高的频率振荡器。

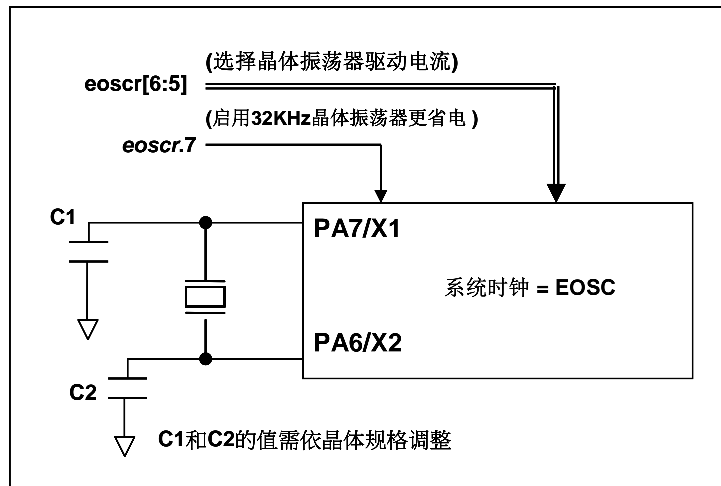


图 8：晶体振荡器的的硬件连接

除了晶振的选择外，外部电容器和 PMC251 寄存器 *eoscr*（0x0b）相关选项也应该适度调整以求得有良好的正弦波。 *eoscr*.7 是用开启晶体振荡器硬件模块，*eoscr*.6 和 *eoscr*.5 用于设置振荡器不同的驱动电流，以满足晶体振荡器不同频率的的要求：

eoscr[6:5]= 01：驱动电流低，适用于较低的频率，例如：32KHz 晶体振荡器

eoscr[6:5]= 10：中度驱动电流，适用于中间的频率，例如：1MHz 的晶体振荡器

eoscr[6:5]= 11：驱动电流高，适用于较高的频率，例如：4MHz 晶体振荡器

表 5 显示了不同的晶体振荡器 C1 和 C2 的推荐值，同时也显示其对应的条件下测量的起振时间。由于晶体或谐振器有其自身的特点，不同类型的晶体或谐振器的启动时间可能会略有不同，请参考其规格并选择恰当的 C1 和 C2 电容值。

©Copyright 2014, PADAUK Technology Co. Ltd

频率	C1	C2	起振时间	条件
4MHz	4.7pF	4.7pF	6ms	(*eoscr*[6:5]=11, *misc*.6=0)
1MHz	10pF	10pF	11ms	(*eoscr*[6:5]=10, *misc*.6=0)
32KHz	22pF	22pF	450ms	(*eoscr*[6:5]=01, *misc*.6=0)

表 5：晶体振荡器 C1 和 C2 推荐值

当使用晶体振荡器，使用者必须特别注意振荡器的稳定时间，稳定时间将取决于振荡器频率、晶型、外部电容和电源电压。在系统时钟切换到晶体振荡器之前，使用者必须确保晶体振荡器是稳定的，相关参考程序如下所示：

```
void  FPPA0 (void)
{
      .ADJUST IC      DISABLE              // IHRC 没校准, WDT 是启用
      ...
      $     EOSCR     Enable, 4Mhz;        // EOSCR = 0b110_00000;
      $     T16M      EOSC, /1, BIT13;     // T16 收到 2^14=16384 个晶体振荡时钟,
                                           // Intrq.T16 =>1,  晶体振荡器已经稳定
      WORD      count     =     0;
      stt16     count;
      Intrq.T16  =    0;
      wait1     Intrq.T16;                 //  从 0x0000 算到 0x2000,  然后设置 INTRQ.T16
      clkmd     =     0xA4;                //  切换系统时钟到 EOSC;
      ...
}
```

请注意，在进入掉电模式之前，晶体振荡器应关闭以避免不预期的唤醒事件发生。假如系统选用 32KHz 的晶体振荡器而且系统又需要极度省电，在振荡器起振后，可以设置寄存器 *misc* 位 6 为 1 来降低电流。

5.7.5 系统时钟和 LVD 基准位

系统时钟的时钟源从 EOSC，IHRC 和 ILRC，PMC251 的时钟系统的硬件框图如图 9 所示。

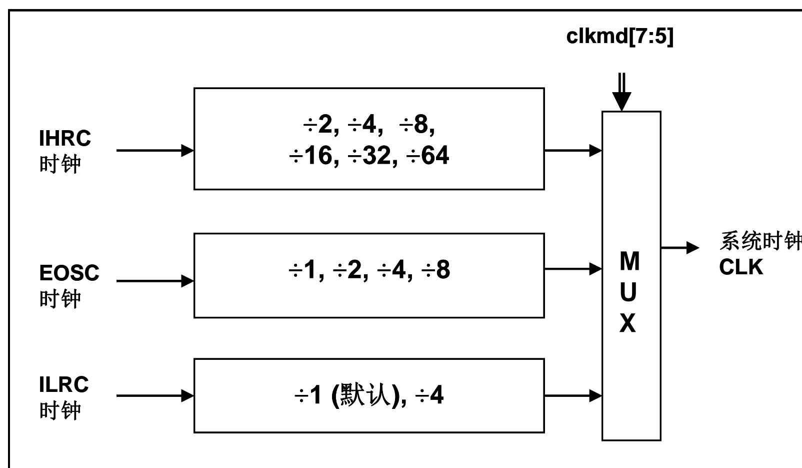


图 9：系统时钟源选择

©Copyright 2014, PADAUK Technology Co. Ltd

使用者可以在不同的需求下选择不同的系统时钟，选定的系统时钟应与电源电压和 LVD 的水平结合，才能使系统稳定。 LVD 的水平是在在编译过程中选择，下面是工作频率和 LVD 水平设定的建议：

- 系统时钟为 8MHz 时，LVD=3.1V
- 系统时钟为 4MHz 时，LVD=2.5V
- 系统时钟为 2MHz 时，LVD=2.2V

5.7.6 系统时钟切换

IHRC 校准后，用户可能希望系统时钟切换到新的频率或可随时切换系统时钟来优化系统性能和功耗。基本上，PMC251 系统时钟可以随意在 IHRC, ILRC 和 EOSC 之间切换，只要透过 ***clkmd*** 寄存器设定，;系统时钟可以立即的转换成新的频率。请注意，在下命令给寄存器 ***clkmd*** 切换频率，不能同时关闭原来的时钟模块。下面这些例子显示更多时钟切换需知道的信息，请参阅“求助”→“使用手册”→ “IC 介绍”→ “缓存器介绍”→ “CLKMD”。

例 1: 系统时钟从 ILRC 切换到 IHRC/2

```
...                      //  系统时钟为 ILRC
CLKMD     =   0x34;      //  切换为 IHRC/2，ILRC 不能在这里禁用
CLKMD.2   =   0;         //  ILRC 可以在这里禁用
...
```

例 2: 系统时钟从 ILRC 切换到 EOSC

```
...                      //  系统时钟为 ILRC
CLKMD     =   0xA6;      //  切换为 IHRC，ILRC 不能在这里禁用
CLKMD.2   =   0;         //  ILRC 可以在这里禁用
...
```

例 3: 系统时钟从 IHRC/2 切换到 ILRC

```
...                      //  系统时钟为 IHRC/2
CLKMD     =   0xF4;      //  切换为 ILRC，IHRC 不能在这里禁用
CLKMD.4   =   0;         //  IHRC 可以在这里禁用
...
```

例 4: 系统时钟从 IHRC/2 切换到 EOSC

```
...                      //  系统时钟为 IHRC/2
CLKMD     =   0XB0;      //  切换为 EOSC，IHRC 不能在这里禁用
CLKMD.4   =   0;         //  IHRC 可以在这里禁用
...
```

例 5: 系统时钟从 IHRC/2 切换到 IHRC/4

```
...                      //  系统时钟为 IHRC/2, ILRC 为启用
CLKMD     =   0X14;      //  切换为 IHRC/4
...
```

例 6: 系统可能当机，如果同时切换时钟和关闭原来的振荡器

```
...                      //  系统时钟为 ILRC
CLKMD     =   0x30;      //  不能从 ILRC 切换到 IHRC/2，
                         //  同时又关闭 ILRC 振荡器
...
```

©Copyright 2014, PADAUK Technology Co. Ltd

5.8 16 位定时器 (Timer16)

PMC251 内置一个 16 位硬件定时器，定时器时钟可来自于系统时钟（CLK）、外部晶振时钟（EOSC）、内部高频振荡时钟（IHRC）、内部低频振荡时钟（ILRC）或 PA0， 1 个多任务器用于选择时钟来源，在送到时钟的 16 位计数器（counter16）之前，1 个可软件编程的预分频器提供÷1、÷4、÷16、÷64 选择，让计数范围更大。

16 位计数器只能向上计数，计数器初始值可以使用 stt16 指令来设定，而计数器的数值也可以利用 ldt16 指令存储到 SRAM 数据存储器。可软件编程的选择器用于选择 Timer16 的中断条件，当计数器溢出时， Timer16 可以触发中断。中断源是来自 16 位定时器的位 8 到位 15，中断类型可以上升沿触发或下降沿触发，是经由寄存器 *integs*.4 选择。Timer16 模块框图如图 10。

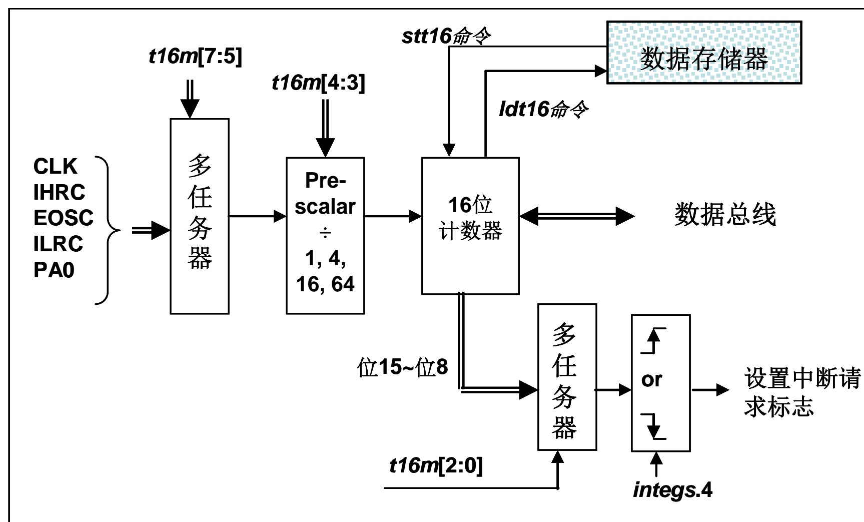


图 10：Timer16 模块框图

使用 Timer16 时，Timer16 的语法定义在.inc 文件中。共有三个参数来定义 Timer16 的使用，第一个参数是用来定义 Timer16 的时钟源，第二个参数是用来定义预分频器，第三个参数是确定中断源。

```
T16M      IO_RW     0x06
    $ 7~5:    STOP, SYSCLK, X, X, PA4, IHRC, EOSC, ILRC, PA0              // 第一个参数
    $ 4~3:    /1, /4, /16, /64                                            // 第二个参数
    $ 2~0:    BIT8, BIT9, BIT10, BIT11, BIT12, BIT13, BIT14, BIT15        // 第三个参数
```

使用者可以依照系统的要求来定义 T16M 参数，例子如下：

```
$    T16M      SYSCLK, /64, BIT15;
     // 选择(SYSCLK/64) 当 Timer16 时钟源,每 2^16 个时钟周期产生一次 INTRQ.2=1
     // 假如用 .ADJUST_OTP_IHRCR 8MIPS，系统时钟 System Clock = IHRC / 2 = 8 MHz
     // SYSCLK/64 = 8 MHz/64 = 8 uS,约每 524 mS 产生一次 INTRQ.2=1

$    T16M      EOSC, /1, BIT13;
     // 选择(EOSC/1) 当 Timer16 时钟源,每 2^14 个时钟周期产生一次 INTRQ.2=1
     // 假如 EOSC=32768 Hz, 32768 Hz/(2^14) = 2Hz, 约每 0.5S 产生一次 INTRQ.2=1
```

©Copyright 2014, PADAUK Technology Co. Ltd

$ T16M PA0, /1, BIT8;

// 选择 PA0 当 Timer16 时钟源，每 2^9 个时钟周期产生一次 INTRQ.2=1
// 每接收 512 个 PA0 个时钟周期产生一次 INTRQ.2=1

$ T16M STOP;
// 停止 Timer16 计数

假如 Timer16 是不受干扰的自由运行，中断发生的频率可以用下列式子描述:

$$F_{INTRQ_T16M} = F_{clock\ source} \div P \div 2^{n+1}$$

这里,
F 是 Timer16 的时钟源频率，
P 是寄存器 ***t16m*** [4:3]的选择(可以为 1, 4, 16, 64)，
N 是中断要求所选择的位，例如：选择位 10，n=10.

5.9 看门狗定时器

看门狗定时器是一个计数器，其时钟源来自内部低频振荡器（ILRC），频率大约是 24KHz。利用 *misc* 寄存器的选择，可以设定四种不同的看门狗定时器超时时间，它是：

- 当 misc[1:0]=11 时：256 个 ILRC 时钟周期
- 当 misc[1:0]=10 时：16384 ILRC 时钟周期
- 当 misc[1:0]=01 时：4096 ILRC 时钟周期
- 当 misc[1:0]=01 时：2048 ILRC 时钟周期

为确保看门狗定时器在超时溢出周期之前被清零，在安全时间内，用指令“wdreset”清零看门狗定时器。在上电复位或任何时候使用 wdreset 指令，看门狗定时器都会被清零。当看门狗定时器超时溢出时， PMC251 将复位并重新运行程序。请特别注意，由于生产制程会引起 ILRC 频率相当大的漂移，上面的数据仅供设计参考用，还是需要以各个单片机测量到的数据为准。

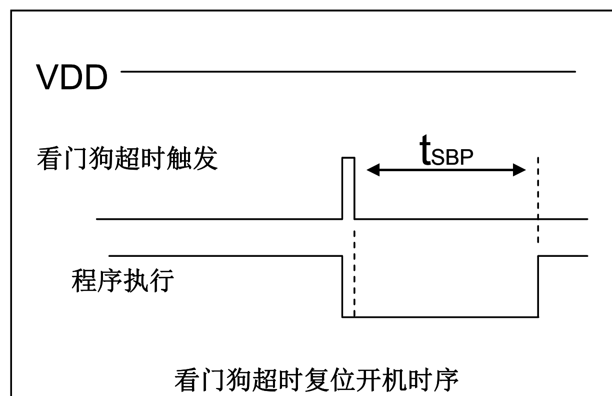


图 11：看门狗定时器超时溢出的相关时序

©Copyright 2014, PADAUK Technology Co. Ltd

5.10 中断

PMC251 有 3 个中断源：两个外部中断源（PA0，PB0，边缘触发形态可以由 *integs* 寄存器选择）和 Timer16 中断源。每个中断请求源都有自己的中断控制位启用或禁用它。硬件框图请参考图 12，所有的中断请求标志位是由硬件置位并且并通过软件写寄存器 INTRQ 清零。中断请求标志设置点可以是上升沿或下降沿或两者兼而有之，这取决于对寄存器 *integs* 的设置。所有的中断请求源最后都需由 *engint* 指令控制（启用全局中断）使中断运行，以及使用 *disgint* 指令（禁用全局中断）停用它。只 FPP0 可以接受中断请求，其它的 FPP 的单元不会受到中断干扰。中断堆栈是共享数据存储器，其地址由堆栈寄存器 *sp* 指定。由于程序计数器是 16 位宽度，堆栈寄存器 *sp* 位 0 应保持 0。此外，用户可以使用 *pushaf* 指令存储 ACC 和标志寄存器的值到堆栈，以及使用 *popaf* 指令将值从堆栈恢复到 ACC 和标志寄存器中。

当 PMC251 执行到中断入口地址处，全局中断会自动停止；到 *reti* 指令被执行时自动恢复启用。中断请求可以在任何时候接受，包括在中断服务程序被执行的过程中，中断嵌套的深度是由软件编程所决定的，因为每个 FPP 单位的 8 位堆栈指针寄存器都是可读写的。由于堆栈是共享数据存储器，使用者应仔细使用，通过软件编程调整栈点在存储器的位置，每个 FPP 单元堆栈指针的深度可以完全由用户指定，以实现最大的系统弹性。中断服务程序的入口地址都是 0x10，只属于 FPP0。

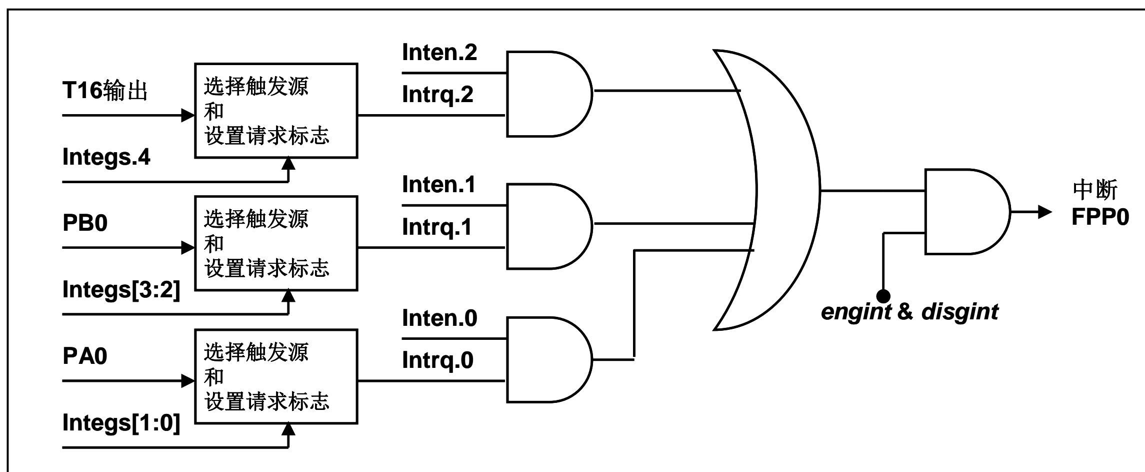


图 12：中断硬件框图

一旦发生中断，工作流程将是：

- 程序计数器将自动存储到 *sp* 寄存器指定的堆栈存储器。
- 新的 *sp* 将被更新为 *sp*+2。
- 全局中断将自动被禁用。
- 将从地址’h010 获取下一条指令。

在中断服务程序中，可以通过读寄存器 *intrq* 知道中断发生源。

中断服务程序完成后，发出 reti 指令返回既有的程序，其具体工作流程将是：

- 从 *sp* 寄存器指定的堆栈存储器自动恢复程序计数器。
- 新的 *sp* 将被更新为 *sp*-2。
- 全局中断将自动启用。
- 下一条指令将是中断前原来的指令。

©Copyright 2014, PADAUK Technology Co. Ltd

使用者必须预留足够的堆栈存储器以存中断向量，一级中断需要两个字节，两级中断需要 4 个字节。下面的示例程序演示了如何处理中断，请注意，处理中断和 *pushaf* 是需要四个字节堆栈存储器。

```
void        FPPA0    (void)
{
    ...
    $   INTEN   PA0;        // INTEN =1;当 PA0 准位改变，产生中断请求
    INTRQ   =   0;          // 清除 INTRQ
    ENGINT                  // 启用全局中断
    ...
    DISGINT                 // 禁用全局中断
    ...
}

void        Interrupt  (void)           // 中断程序
{
    PUSHAF                          //   存储 ALU 和 FLAG 寄存器
    If (INTRQ.0)
    {                               //   PA0 的中断程序
      INTRQ.0   =     0;
      ...
    }
    ...
    POPAF                           //    回复 ALU 和 FLAG 寄存器
 }
```

5.11 省电与掉电

PMC251 有三个由硬件定义的操作模式，分别为：正常工作模式，电源省电模式和掉电模式。正常工作模式是所有功能都正常运行的状态，省电模式（*stopexe*）是在降低工作电流而且 CPU 保持在随时可以继续工作的状态，掉电模式（*stopsys*）是用来深度的节省电力。因此，省电模式适合在偶尔需要唤醒的系统工作，掉电模式是在非常低消耗功率且很少需要唤醒的系统中使用。图 13 显示省电模式（*stopexe*）和掉电模式（*stopsys*）之间在振荡器模块的差异，没改变就是维持原状态。

STOPSYS 和 STOPEXE 模式下在振荡器的差异			
	IHRC	ILRC	EOSC
STOPSYS	停止	停止	停止
STOPEXE	没改变	没改变	没改变

图 13：省电模式和掉电模式在振荡器模块的差异

5.11.1 省电模式（stopexe）

使用 *stopexe* 指令进入省电模式，只有系统时钟被禁用，其余所有的振荡器模块都仍继续工作。所以只有 CPU 是停止执行指令，对 Timer16 计数器而言，如果它的时钟源不是系统时钟，那 Timer16 仍然会保持计数。*stopexe* 的省电模式下，唤醒源可以是 IO 的切换，或者 Timer16 计数到设定值时(假如 Timer16 的时钟源是 IHRC、ILRC 或 EOSC 模块)。假如系统唤醒是因输入引脚切换，那可以视为单片机继续正常的运行，在 *stopexe* 指令之后最好加个 ***nop*** 指令，省电模式的详细信息如下所示：

- IHRC、ILRC 和 EOSC 振荡器模块：没有变化。如果它被启用，它仍然继续保持活跃。
- 系统时钟禁用。因此，CPU 停止执行。
- OTP 存储器被关闭。
- Timer16：停止计数，如果选择系统时钟或相应的振荡器模块被禁止，否则，仍然保持计数。
- 唤醒来源：IO 的切换或 Timer16

请注意在下“***stopexe***”命令前，必须先关闭看门狗时钟以避免发生复位，例子如下：

```
CLKMD.En_WatchDog   =   0;        // 关闭看门狗时钟
stopexe;
nop;
....                              // 省电中
Wdreset;
CLKMD.En_WatchDog   =   1;        // 开启看门狗时钟
```

另一个例子是利用 Timer16 来唤醒系统因 ***stopexe*** 的省电模式:

```
$ T16M    IHRC, /1, BIT8          // Timer16 setting
…
WORD      count     =    0;
STT16     count;
stopexe;
nop;
…
```

Timer16 的初始值为 0，在 Timer16 计数了 256 个 IHRC 时钟后，系统将被唤醒。.

©Copyright 2014, PADAUK Technology Co. Ltd

5.11.2 掉电模式（stopsys）

掉电模式是深度省电的状态，所有的振荡器模块都会被关闭。使用 *stopsys* 指令就可以使 PMC251 芯片直接进入掉电模式。在进入掉电模式之前，必须启用内部低频振荡器（ILRC）以便唤醒系统时使用，也就是说在发出 *stopsys* 命令之前，*clkmd* 寄存器的位 2 必须设置为 1。下面显示发出 *stopsys* 命令后，PMC251 内部详细的状态：

- 所有的振荡器模块被关闭。
- 启用内部低频振荡器（设置寄存器 *clkmd* 位 2）。
- OTP 存储器被关闭。
- SRAM 和寄存器内容保持不变。
- 唤醒源：任何 IO 切换。
- 如果 PA 或 PB 是输入模式，并由 *padier* 或 *pbdier* 寄存器设置为模拟输入，那该引脚是不能被用来唤醒系统。

输入引脚的唤醒可以被视为正常运行的延续，为了降低功耗，进入掉电模式之前，所有的 I/O 引脚应仔细检查，避免悬空而漏电。断电参考示例程序如下所示：

```
CMLKMD =     0xF4;      // 系统时钟从 IHRC 变为 ILRC，关闭看门狗时钟
CLKMD.4 =    0;         //  IHRC 禁用
…
while (1)
{
        STOPSYS;        // 进入断电模式
        if  (…)  break; // 假如发生唤醒而且检查 OK，就返回正常工作
                        // 否则，停留在断电模式。
}
CLKMD   =    0x34;      // 系统时钟从 ILRC 变为 IHRC/2
```

5.11.3 唤醒

进入掉电或省电模式后，PMC251 可以通过切换 IO 引脚恢复正常工作；而 Timer16 中断的唤醒只适用于省电模式。图 14 显示 *stopsysy* 掉电模式和 *stopexe* 省电模式在唤醒源的差异。

掉电模式和省电模式在唤醒源的差异		
	切换 IO 引脚	T16 中断
stopsys	是	否
stopexe	是	是

图 14：掉电模式和省电模式在唤醒源的差异

当使用 IO 引脚来唤醒 PMC251，寄存器 *padier* 和 *pbdier* 应正确设置，使每一个相应的引脚可以有唤醒功能。从唤醒事件发生后开始计数，正常的唤醒时间大约是 1024 个 ILRC 时钟周期；另外，PMC251 提供快速唤醒功能，透过 *misc* 寄存器选择快速唤醒可以降低唤醒时间。对快速唤醒而言，假如是在 *stopexe* 省电模式下，切换 IO 引脚的快速唤醒时间为 128 个系统时钟周期；假如是在 *stopsys* 掉电模式下，切换 IO 引脚的快速唤醒时间为 128 个系统时钟周期加上上电后振荡器(IHRC 或 ILRC)的稳定时间。振荡器的稳定时间是从上电后开始算起，视系统时钟是选择 IHRC 或 ILRC 而定。当 EOSC 被选用当系统时钟后，快速换醒就自动关闭。

模式	唤醒模式	系统时钟源	切换 IO 引脚的唤醒时间(t_{WUP})
STOPEXE 省电模式	快速唤醒	IHRC 或 ILRC	128 * $T_{SYS,}$；这里 T_{SYS} 是系统时钟周期
STOPSYS 掉电模式	快速唤醒	IHRC	128 T_{SYS} + T_{SIHRC}； 这里 T_{SIHRC} 是 IHRC 从上电到稳定的时间，在 5V 下约 5us
STOPSYS 掉电模式	快速唤醒	ILRC	128 T_{SYS} + T_{SILRC}； 这里 T_{SILRC} 是 ILRC 从上电到稳定的时间，在 5V 下约 43ms
STOPSYS 或 STOPEXE 模式	快速唤醒	EOSC	1024 * T_{ILRC}；这里 T_{ILRC} 是 ILRC 时钟周期
STOPEXE 省电模式	普通唤醒	任一	1024 * T_{ILRC}；这里 T_{ILRC} 是 ILRC 时钟周期
STOPSYS 掉电模式	普通唤醒	任一	1024 * T_{ILRC}；这里 T_{ILRC} 是 ILRC 时钟周期

请注意：当启用快速唤醒时，看门狗时钟源会切换到系统时钟(例如：4MHz)，所以，建议要进入掉电模式前，打开快速唤醒之前要关闭看门狗定时器，等系统被唤醒后，在关闭快速唤醒之后再打开看门狗定时器。

©Copyright 2014, PADAUK Technology Co. Ltd

5.12 IO 引脚

除了 PA5，PMC251 所有 IO 引脚都可以设定成输入或输出，透过数据寄存器（*pa, pb*），控制寄存器（*pac, pbc*）和弱上拉电阻（*paph, pbph*）设定，每一 IO 引脚都可以独立配置成不同的功能；所有这些引脚设置有施密特触发输入缓冲器和 CMOS 输出驱动电位水平。当这些引脚为输出低电位时，弱上拉电阻会自动关闭。如果要读取端口上的电位状态，一定要先设置成输入模式；在输出模式下，读取到的数据是数据寄存器的值。图 16 显示了 IO 缓冲区硬件图，除了 PA5 外，所有的 IO 口具有相同的结构，当 PMC251 进入掉电模式，所有的 IO 引脚都可唤醒系统。表 6 端口 PA0 位的设定配置表。

pa.0	*pac.0*	*paph.0*	描述
X	0	0	输入，没有弱上拉电阻
X	0	1	输入，有弱上拉电阻
0	1	X	输出低电位，没有弱上拉电阻（弱上拉电阻自动关闭）
1	1	0	输出高电位，没有弱上拉电阻
1	1	1	输出高电位，有弱上拉电阻

表 6：PA0 设定配置表

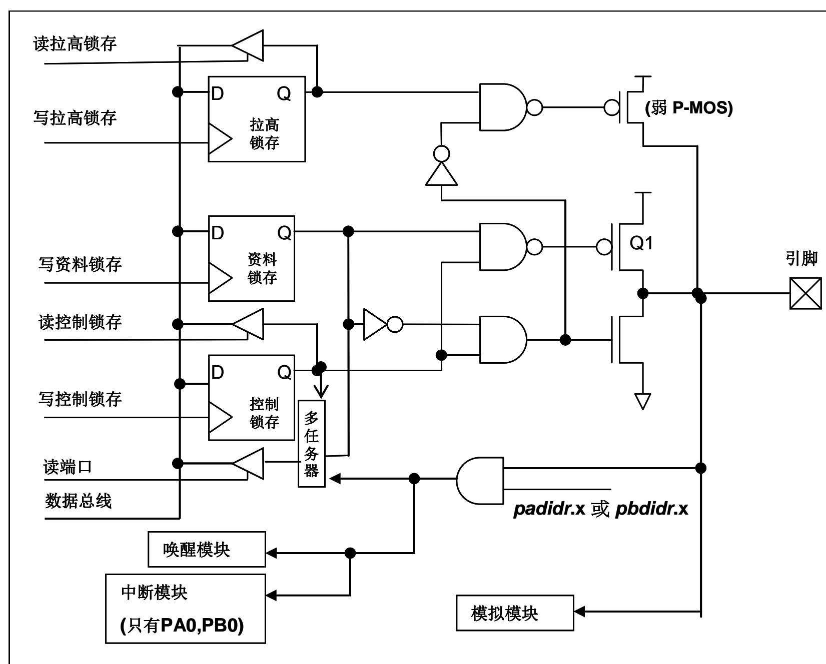


图 16：引脚缓冲区硬件图

除了 PA5 外，所有的 IO 引脚具有相同的结构；PA5 的输出只能是漏极开路模式（没有 Q1）。对于被选择为模拟功能的引脚，必须在寄存器 *padier* 以及 *pbdier* 相应位设置为低，以防止漏电流。当 PMC251 在掉电或省电模式，每一个引脚都可以切换其状态来唤醒系统。对于需用来唤醒系统的引脚，必须设置为输入模式以及寄存器 *padier* 以及 *pbdier* 相应为高。同样的原因，当 PA0 或 PB0 用来作为外部中断引脚时，*padier.0* 或 *pbdier.0* 应设置高。

©Copyright 2014, PADAUK Technology Co. Ltd

5.13 复位和 LVD

5.13.1 复位

引起 PMC251 复位的原因有很多，一旦复位发生，PMC251 的大部分寄存器将被设置为默认值，只有 *gdio* 寄存器（IO 地址为 0x7）仍然保持相同的值；发生复位后，系统会重新启动，程序计数器会跳跃地址'h0。当发生上电复位或 LVD 复位，数据存储器的值是在不确定的状态；然而，若是复位是因为 PRST# 引脚或 WDT 超时溢位，数据存储器的值将被保留。

5.13.2 LVD 复位

程序编译时，使用者可以选择 8 个不同级别的 LVD ~ 4.1V, 3.6V, 3.1V, 2.8V, 2.5V, 2.2V, 2.0V, 1.8V，通常情况下，使用者在选择 LVD 复位水平时，必须结合单片机工作频率和电源电压，以便让单片机稳定工作。

©Copyright 2014, PADAUK Technology Co. Ltd

6. IO 寄存器

6.1 标志寄存器（*flag*），IO 地址 = 0x00

位	初始值	读/写	描 述
7 - 4	-	-	保留。这 4 个位读值为“1”。
3	0	读/写	OV（溢出标志）。当数学运算溢出时，这一位会设置为 1。
2	0	读/写	AC（辅助进位标志）。两个条件下，此位设置为 1：（1）是进行低半字节加法运算产生进位（2）减法运算时，低半字节向高半字节借位。
1	0	读/写	C（进位标志）。有两个条件下，此位设置为 1：（1）加法运算产生进位（2）减法运算有借位。进位标志还受带进位标志的 shift 指令影响。
0	0	读/写	Z（零）。此位将被设置为 1，当算术或逻辑运算的结果是 0；否则将被清零。

6.2 FPP 单元允许寄存器（*fppen*），IO 地址 = 0x01

位	初始值	读/写	描 述
7 – 2	-	-	保留。
1	0	读/写	FPP1 启用。此位是用来启用 FPP1。 0/1：禁用/启用
0	1	读/写	FPP0 启用。此位是用来启用 FPP0。 0/1：禁用/启用

6.3 堆栈指针寄存器（*sp*），IO 地址 = 0x02

位	初始值	读/写	描 述
7 - 0	-	读/写	堆栈指针寄存器。读出当前堆栈指针，或写入以改变堆栈指针。

6.4 时钟控制寄存器（*clkmd*），IO 地址 = 0x03

位	初始值	读/写	描 述	
7 – 5	111	读/写	系统时钟选择	
			类型 0, clkmd[3]=0	类型 1, clkmd[3]=1
			000: IHRC÷4 001: IHRC÷2 010: 保留 011: EOSC÷4 100: EOSC÷2 101: EOSC 110: ILRC÷4 111: ILRC (默认)	000: IHRC÷16 001: IHRC÷8 010: 保留 011: IHRC÷32 100: IHRC÷64 101: EOSC÷8 11x: 保留
4	1	读/写	内部高频 RC 振荡器功能。 0/1：禁用/启用	
3	0	读/写	时钟类型选择。这个位是用来选择位 7~位 5 的时钟类型。 0 / 1: 类型 0 /类型 1	
2	1	读/写	内部低频 RC 振荡器功能。0/1：禁用/启用 当内部低频 RC 振荡器功能禁用时，看门狗定时器功能同时被关闭。	
1	1	读/写	看门狗定时器功能。 0/1：禁用/启用	
0	0	读/写	引脚 PA5/PRST# 功能. 0 / 1: PA5 / PRST#.	

©Copyright 2014, PADAUK Technology Co. Ltd

6.5 中断允许寄存器（*inten*），IO 地址 = 0x04

位	初始值	读/写	描 述
7 – 3	-	读/写	保留。
2	-	读/写	启用从 Timer16 的溢出中断。0/1：禁用/启用
1	-	读/写	启用从 pb0 的中断。 0/1：禁用/启用
0	-	读/写	启用从 pa0 的中断。 0/1：禁用/启用

6.6 中断请求寄存器（*intrq*），IO 地址 = 0x05

位	初始值	读/写	描 述
7 - 3	-	读/写	保留。
2	-	读/写	Timer16 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
1	-	读/写	PB0 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
0	-	读/写	PA0 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求

6.7 Timer16 控制寄存器（*t16m*），IO 地址 = 0x06

位	初始值	读/写	描 述
7 - 5	000	读/写	Timer16 时钟选择 000: Timer16 禁用 001：CLK 系统时钟 010: 保留 011：保留 100：IHRC 101：EOSC 110：ILRC 111：PA0（外部事件）
4 – 3	00	读/写	Timer16 内部的时钟分频器 00： ÷1 01： ÷4 10： ÷16 11： ÷64
2 – 0	000	读/写	中断源选择。当选择位由低变高时，发生中断事件。 0 ：Timer16 位 8 1 ：Timer16 位 9 2 ：Timer16 位 10 3 ：Timer16 位 11 4 ：Timer16 位 12 5 ：Timer16 位 13 6 ：Timer16 位 14 7 ：Timer16 位 15

©Copyright 2014, PADAUK Technology Co. Ltd

6.8 通用数据输入/输出寄存器（*gdio*），IO 地址 = 0x07

位	初始值	读/写	描 述
7 – 0	00	读/写	这个端口是 IO 空间的数据缓冲区，它只有在 POR、LVD 或引脚 PRST＃发生复位时被清零。它是在 IO 空间进行操作，如 wait0 gdio.x，wait1 gdio.x 和 tog gdio.x 用以取代记忆空间的指令（例如：wait1 mem; wait0 mem; tog mem）。

6.9 外部晶体振荡器控制寄存器（*eoscr*，只写），IO 地址 = 0x0a

位	初始值	读/写	描 述
7	0	只写	启用晶体振荡器。 0/1：禁用/启用
6 – 5	00	只写	晶体振荡器的选择。 00：保留 01：低驱动电流。适用于较低频率晶体，例如：32KHz 10：中驱动电流。适用于中等频率晶体，例如：1MHz 11：高驱动电流。适用于较高频率晶体，例如：4MHz
4 – 1	-	-	保留
0	0	只写	将 Band-gap 和 LVD 硬件模块断电。 0 / 1：正常/ 断电

6.10 内部高频 RC 振荡器控制寄存器（*ihrcr*，只写），IO 地址 = 0x0b

位	初始值	读/写	描 述
7 – 0	00	只写	内部高频 RC 振荡器的速度校准的位[7：0]。 这个寄存器是给系统频率校准使用，使用者请勿自行填入值。

6.11 中断沿选择寄存器（*integs*，只写），IO 地址 = 0x0c

位	初始值	读/写	描 述
7 – 5	-	-	保留。请设为 0。
4	0	只写	Timer16 中断沿选择。 0：上升沿请求中断。 1：下降沿请求中断。
3 – 2	00	只写	PB0 中断沿选择。 00：上升沿和下降沿都请求中断。 01：上升沿请求中断。 10：下降沿请求中断。 11：保留。
1 – 0	00	只写	PA0 中断沿选择。 00：上升沿和下降沿都请求中断。 01：上升沿请求中断。 10：下降沿请求中断。 11：保留。

©Copyright 2014, PADAUK Technology Co. Ltd

6.12 端口 A 数字输入启用寄存器（*padier*），IO 地址 = 0x0d

位	初始值	读/写	描 述
7	1	只写	启用 PA7 数字输入和系统唤醒。 1 / 0 : 启用 / 禁用 当选择外部晶体振荡器时，这个位要设为 0，以避免漏电流。当这个位设为 0 时，PA7 无法用来唤醒系统。 注意：使用仿真器时，当此位为 1 时，功能是被禁用的；0 才是启用
6	1	只写	启用 PA6 数字输入和系统唤醒。 1 / 0 : 启用 / 禁用 当选择外部晶体振荡器时，这个位要设为 0，以避免漏电流。当这个位设为 0 时，PA7 无法用来唤醒系统。注意：使用仿真器时，当此位为 1 时，功能是被禁用的；0 才是启用
5	1	只写	启用 PA5 系统唤醒。 1 / 0 : 启用 / 禁用 当这个位设为 0 时，PA5 无法用来唤醒系统。 注意：使用仿真器时，当此位为 1 时，功能是被禁用的；0 才是启用
4	1	只写	启用 PA4 系统唤醒。 1 / 0 : 启用 / 禁用 当这个位设为 0 时，PA4 无法用来唤醒系统。 注意：使用仿真器时，当此位为 1 时，功能是被禁用的；0 才是启用
3	1	只写	启用 PA3 系统唤醒。 1 / 0 : 启用 / 禁用 当这个位设为 0 时，PA3 无法用来唤醒系统。 注意：使用仿真器时，当此位为 1 时，功能是被禁用的；0 才是启用
2 – 1	-	-	保留。
0	1	只写	启用 PA0 系统唤醒和中断请求。 1 / 0 : 启用 / 禁用 当这个位设为 0 时，PA0 无法用来唤醒系统以及中断请求。 注意：使用仿真器时，当此位为 1 时，功能是被禁用的；0 才是启用

请注意：在仿真器模拟时和实际芯片，这个寄存器的控制正好相反，为了在仿真器模拟和实际芯片能是相同的一个程序，请使用下面的命令来写入这个寄存器：

“$ PADIER 0xhh” ;

例如：

$ PADIER 0xF0;

上面命令用来在仿真器模拟和实际芯片时，都能自动且正确开启端口 A 数字输入启用寄存器位[7:4]的数字输入和唤醒功能。

6.13 端口 B 数字输入启用寄存器（*pbdier*），IO 地址 = 0x0e

位	初始值	读/写	描 述
7	1	只写	启用 PB7 系统唤醒。 1 / 0 : 启用 / 禁用 当这个位设为 0 时，PB7 无法用来唤醒系统。 注意：使用仿真器时，当此位为 1 时，功能是被禁用的；0 才是启用
6	1	只写	启用 PB6 系统唤醒。 1 / 0 : 启用 / 禁用 当这个位设为 0 时，PB6 无法用来唤醒系统。 注意：使用仿真器时，当此位为 1 时，功能是被禁用的；0 才是启用
5	1	只写	启用 PB5 系统唤醒。 1 / 0 : 启用 / 禁用 当这个位设为 0 时，PB5 无法用来唤醒系统。 注意：使用仿真器时，当此位为 1 时，功能是被禁用的；0 才是启用
4 – 3	-	-	保留。
2	1	只写	启用 PB2 系统唤醒。 1 / 0 : 启用 / 禁用 当这个位设为 0 时，PB2 无法用来唤醒系统。 注意：使用仿真器时，当此位为 1 时，功能是被禁用的；0 才是启用
1	1	只写	启用 PB1 系统唤醒。 1 / 0 : 启用 / 禁用 当这个位设为 0 时，PB1 无法用来唤醒系统。 注意：使用仿真器时，当此位为 1 时，功能是被禁用的；0 才是启用
0	1	只写	启用 PB0 系统唤醒和中断请求。 1 / 0 : 启用 / 禁用 当这个位设为 0 时，PB0 无法用来唤醒系统和中断请求。 注意：使用仿真器时，当此位为 1 时，功能是被禁用的；0 才是启用

请注意：在仿真器模拟时和实际芯片，这个寄存器的控制正好相反，为了在仿真器模拟和实际芯片能是相同的一个程序，请使用下面的命令来写入这个寄存器：

“$ PBDIER 0xhh” ;

例如：

$ PBDIER 0xF0;

上面命令用来在仿真器模拟和实际芯片时，都能自动且正确开启端口 A 数字输入启用寄存器位[7:4]的数字输入和唤醒功能。

6.14 端口 A 数据寄存器（*pa*），IO 地址 = 0x10

位	初始值	读/写	描 述
7 – 0	8’h00	读/写	数据寄存器的端口 A。

6.15 端口 A 控制寄存器（*pac*），IO 地址 = 0x11

位	初始值	读/写	描 述
7 - 0	8’h00	读/写	端口 A 控制寄存器。这些寄存器是用来定义端口 A 每个相应的引脚的输入模式或输出模式。 0/1：输入/输出

6.16 端口 A 上拉控制寄存器（*paph*），IO 地址 = 0x12

位	初始值	读/写	描 述
7 - 0	8’h00	读/写	端口 A 上拉控制寄存器。这些寄存器是用来控制上拉高端口 A 每个相应的引脚。 0/1：禁用/启用 请注意：端口 A 位 5（PA5）没有上拉电阻。

6.17 端口 B 数据寄存器（*pb*），IO 地址 = 0x14

位	初始值	读/写	描 述
7 – 0	8’h00	读/写	数据寄存器的端口 B。

6.18 端口 B 控制寄存器（*pbc*），IO 地址 = 0x15

位	初始值	读/写	描 述
7 - 0	8’h00	读/写	端口 B 控制寄存器。这些寄存器是用来定义端口 B 每个相应的引脚的输入模式或输出模式。 0/1：输入/输出

6.19 端口 B 上拉控制寄存器（*pbph*），IO 地址 = 0x16

位	初始值	读/写	描 述
7 - 0	8’h00	读/写	端口 B 上拉控制寄存器。这些寄存器是用来控制上拉高端口 B 每个相应的引脚。 0/1：禁用/启用

©Copyright 2014, PADAUK Technology Co. Ltd

6.20 杂项寄存器（*misc*），IO 地址 = 0x3b

位	初始值	读/写	描 述
7	0	-	保留。
6	0	WO	32KHz 晶体振荡器起振后，进入振荡器省电模式。 0: 普通模式。 1: 32KHz 晶体振荡器省电模式
5	0	WO	快唤醒功能。 0: 正常唤醒。唤醒时间为 1024 ILRC 时钟。 1: 快唤醒。 假如系统时钟用 IHRC：唤醒时间为 128 个系统时钟。 假如系统时钟用晶体振荡器：唤醒时间为 1024 个系统时钟+晶体振荡器稳定时间 请注意：当启用快速唤醒时，看门狗时钟源会切换到系统时钟(例如：4MHz) , 所以, 建议要进入掉电模式前，打开快速唤醒之前要关闭看门狗定时器，等系统被唤醒后，在关闭快速唤醒之后再打开看门狗定时器。
4	0	WO	保留。
3	0	WO	从 LVD 复位后，单片机开机时间： 0：正常。从 LVD 复位后，单片机开机时间约为 1024 个 ILRC。 1：快速。从 LVD 复位后，单片机开机时间约为 64 个 ILRC。
2	0	WO	禁用 LVD 功能： 0 / 1 : 启用 / 禁用
1 – 0	00	WO	看门狗时钟超时时间设定： 00: 2048 个 ILRC 时钟周期 01: 4096 个 ILRC 时钟周期 10: 16384 个 ILRC 时钟周期 11: 256 个 ILRC 时钟周期

7. 指令

符 号	描 述
ACC	累加器(ACC 是累加器的简写，用来避免与程序的 a 混淆)
a	累加器(a 是程序使用的累加器)
sp	堆栈指针
Flag	标志寄存器
I	即时数据
&	逻辑 AND
|	逻辑 OR
←	移动
^	异或 OR
+	加
－	减
～	按位取反（逻辑补数，1 补数）
〒	负数 (2's complement)
OV	溢出（2 补数系统的运算结果超出范围）
Z	零（如果零运算单元操作的结果是 0，这位设置为 1）
C	进位（Carry）
AC	辅助进位标志（Auxiliary Carry）。
pc0	FPP0 的程序计数器
pc1	FPP1 的程序计数器

7.1 数据传输类指令

指令	说明
mov **a, I**	移动即时数据到累加器 例如： *mov* a, 0x0f; 结果： a ← 0fh; 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
mov **M, a**	移动数据由累加器到存储器 例如： *mov* MEM, a; 结果： MEM ← a 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
mov **a, M**	移动数据由存储器到累加器 例如： *mov* a, MEM ; 结果： a ← MEM; 当 MEM 为零时，标志位 Z 会被置位。 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
mov **a, IO**	移动数据由 IO 到累加器 例如： *mov* a, pa ; 结果： a ← pa; 当 pa 为零时，标志位 Z 会被置位。 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』

mov **IO, a**	移动数据由累加器到 IO 例如： *mov* pb, a; 结果： pb ← a ; 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

nmov **M, a**	取累加器的负逻辑（2 补数）并复制到存储器。
	例如： *mov* MEM, a;
	结果： MEM ← a 的 2 补码
	受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
	应用范例：
	-- 	
	mov a, 0xf5 ; // ACC=0xf5
	nmov ram9, a; // ram9=0x0b，ACC=0xf5
	-- 	

nmov **a, M**	取存储器的负逻辑（2 补数）并复制到累加器。
	例如： *mov* a, MEM ;
	结果： a ←MEM 的 2 补码; 当 MEM 的 2 补码为零时，标志位 Z 会被置位。
	受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
	应用范例：
	-- 	
	mov a, 0xf5 ;
	mov ram9, a ; // ram9=0xf5
	nmov a, ram9 ; // ram9=0xf5, ACC=0x0b
	-- 	

ldtabh **index**	使用索引作为 OTP 的地址将 OTP 程序存储器的高字节数据读取并载入到累加器。它需要 2T 时间执行这一指令。
	例如： *ldtabh* index;
	结果： a ← {bit 15~8 of OTP [index]};
	受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
	应用范例：
	-- 	
	word ROMptr ; // 在 RAM 定义 OTP 的指针
	…
	mov a, la@TableA ; // 指定 OTP TableA 指针 （LSB）
	mov lb@ROMptr, a ; // 将指针存到 RAM（LSB）
	mov a, ha@TableA ; // 指定 OTP TableA 指针(MSB)
	mov hb@ROMptr, a ; // 将指针存到 RAM (MSB)
	…
	ldtabh ROMptr ; // 读取数据并载入到累加器 (ACC=0X02)
	….
	TableA : dc 0x0234, 0x0042, 0x0024, 0x0018 ;
	-- 	

ldtabl **index**	使用索引作为 OTP 的地址并将 OTP 程序存储器的低字节数据读取并载入到累加器。它需要 2T 时间执行这一指令。 例如： *ldtabl* index; 结果： a ← {bit7~0 of OTP [index]}; 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

©Copyright 2014, PADAUK Technology Co. Ltd

	应用范例：
	--- 	
	word ROMptr ; // 在 RAM 定义 OTP 的指针
	…
	mov a, la@TableA ; // 指定 OTP TableA 指针（LSB）
	mov lb@ROMptr, a ; // 将指针存到 RAM （LSB）
	mov a, ha@TableA ; // 指定 OTP TableA 指针 （MSB）
	mov hb@ROMptr, a ; // 将指针存到 RAM （MSB）
	…
	ldtabl ROMptr ; // 读取数据并载入到累加器 （ACC=0x34）
	….
	TableA : dc 0x0234, 0x0042, 0x0024, 0x0018 ;
	--- 	

***ldt16* word**	将 Timer16 的 16 位计算值复制到 RAM。
	例如： *ldt16* word;
	结果： word ← 16-bit timer
	受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
	应用范例：
	--- 	
	word T16val ; // 定义一个 RAM word
	…
	clear lb@T16val ; // 清零 T16val（LSB）
	clear hb@T16val ; // 清零 T16val（MSB）
	stt16 T16val ; // 设定 Timer16 的起始值为 0
	…
	set1 t16m.5 ; // 启用 Timer16
	…
	set0 t16m.5 ; // 禁用 Timer16
	ldt16 T16val ; // 将 Timer16 的 16 位计算值复制到 RAM T16val
	….
	--- 	

***stt16* word**	将放在 word 的 16 位 RAM 复制到 Timer16。
	例如： *stt16* word;
	结果： 16-bit timer ← word
	受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
	应用范例：
	--- 	
	word T16val ; // 定义一个 RAM word
	…
	mov a, 0x34 ;
	mov lb@T16val, a ; // 将 0x34 搬到 T16val（LSB）
	mov a, 0x12 ;
	mov hb@T16val, a ; // 将 0x12 搬到 T16val（MSB）
	stt16 T16val ; // Timer16 初始化 0x1234
	…
	--- 	

***idxm* a, index**	使用索引作为 RAM 的地址并将 RAM 的数据读取并载入到累加器。它需要 2T 时间执行这一指令。 例如： *idxm* a, index; 结果： a ← [index], index 是用 word 定义。 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』

©Copyright 2014, PADAUK Technology Co. Ltd

	应用范例：
	-- 	
	word RAMIndex ; // 定义一个 RAM 指针
	…
	mov a, 0x5B ; // 指定指针地址（LSB）
	mov lb@RAMIndex, a ; // 将指针存到 RAM（LSB）
	mov a, 0x00 ; // 指定指针地址为 0x00（MSB），在 PMC251 要为 0
	mov hb@RAMIndex, a ; // 将指针存到 RAM（MSB）
	…
	idxm a, RAMIndex ; // 将 RAM 地址为 0x5B 的数据读取并载入累加器
	-- 	

Idxm **index, a**	使用索引作为 RAM 的地址并将累加器的数据读取并载入到 RAM。它需要 2T 时间执行这一指令。
	例如：*idxm* index, a;
	结果： [index] ← a; index 是以 word 定义.
	受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
	应用范例：
	-- 	
	word RAMIndex ; // 定义一个 RAM 指针
	…
	mov a, 0x5B ; // 指定指针地址（LSB）
	mov lb@RAMIndex, a ; // 将指针存到 RAM（LSB）
	mov a, 0x00 ; //指定指针地址为 0x00（MSB），在 PMC251 要为 0
	mov hb@RAMIndex, a ; // 将指针存到 RAM（MSB）
	…
	mov a, 0xA5 ;
	idxm RAMIndex, a ; // 将累加器数据读取并载入地址为 0x5B 的 RAM
	-- 	

xch **M**	累加器与 RAM 之间交换数据 例如： *xch* MEM ; 结果： MEM ← a , a ← MEM 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

pushaf	将累加器和算术逻辑状态寄存器的数据存到堆栈指针指定的堆栈存储器
	例如：*pushaf*;
	结果： [sp] ← {flag, ACC};
	sp ← sp + 2 ;
	受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
	应用范例：
	-- 	
	.romadr 0x10 ; // 中断服务程序入口地址
	pushaf ; // 将累加器和算术逻辑状态寄存器的资料存到堆栈存储器
	… // 中断服务程序
	… // 中断服务程序
	popaf ; // 将堆栈存储器的资料回存到累加器和算术逻辑状态寄存器
	reti ;
	-- 	

popaf	将堆栈指针指定的堆栈存储器的数据回传到累加器和算术逻辑状态寄存器 例如：*popaf*; 结果： sp ← sp - 2 ; {Flag, ACC} ← [sp] ; 受影响的标志位：Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』

7.2 算术运算类指令

指令	说明
add **a, I**	将立即数据与累加器相加，然后把结果放入累加器 例如： *add* a, 0x0f ; 结果： a ← a + 0fh 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
add **a, M**	将 RAM 与累加器相加，然后把结果放入累加器 例如： *add* a, MEM ; 结果： a ← a + MEM 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
add **M, a**	将 RAM 与累加器相加，然后把结果放入 RAM 例如： *add* MEM, a; 结果： MEM ← a + MEM 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
addc **a, M**	将 RAM、累加器以及进位相加，然后把结果放入累加器 例如： *addc* a, MEM ; 结果： a ← a + MEM + C 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
addc **M, a**	将 RAM、累加器以及进位相加，然后把结果放入 RAM 例如： *addc* MEM, a ; 结果： MEM ← a + MEM + C 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
addc **a**	将累加器与进位相加，然后把结果放入累加器 例如： *addc* a ; 结果： a ← a + C 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
addc **M**	将 RAM 与进位相加，然后把结果放入 RAM 例如： *addc* MEM ; 结果： MEM ← MEM + C 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
nadd **a, M**	将累加器的负逻辑（2 补码）与 RAM 相加，然后把结果放入累加器 例如： *nadd* a, MEM ; 结果： a ←a 的 2 补码 + MEM 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
nadd **M, a**	将 RAM 的负逻辑（2 补码）与累加器相加，然后把结果放入 RAM 例如： *nadd* MEM, a ; 结果： MEM ← MEM 的 2 补码 + a 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
sub **a, I**	累加器减立即数据，然后把结果放入累加器 例如： *sub* a, 0x0f; 结果： a ← a - 0fh (a + [2’s complement of 0fh]) 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
sub **a, M**	累加器减 RAM，然后把结果放入累加器 例如： *sub* a, MEM ; 结果： a ← a - MEM (a + [2’s complement of M]) 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
sub **M, a**	RAM 减累加器，然后把结果放入 RAM 例如： *sub* MEM, a; 结果： MEM ← MEM - a (MEM + [2’s complement of a]) 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』

©Copyright 2014, PADAUK Technology Co. Ltd

subc **a, M**	累加器减 RAM，再减进位，然后把结果放入累加器 例如： *subc* a, MEM; 结果： a ← a － MEM - C 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
subc **M, a**	RAM 减累加器，再减进位，然后把结果放入 RAM 例如： *subc* MEM, a ; 结果： MEM ← MEM － a - C 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
subc **a**	累加器减进位，然后把结果放入累加器 例如： *subc* a; 结果： a ← a - C 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
subc **M**	RAM 减进位，然后把结果放入 RAM 例如： *subc* MEM; 结果： MEM ← MEM - C 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
inc **M**	RAM 加 1 例如： *inc* MEM ; 结果： MEM ← MEM + 1 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
dec **M**	RAM 减 1 例如： *dec* MEM; 结果： MEM ← MEM - 1 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
clear **M**	清除 RAM 为 0 例如： *clear* MEM ; 结果： MEM ← 0 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

7.3 移位运算类指令

sr **a**	累加器的位右移，位 7 移入值为 0 例如： *sr* a ; 结果： a (0,b7,b6,b5,b4,b3,b2,b1) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a(b0) 受影响的标志位： Z：『不变』， C：『受影响』， AC：『不变』， OV：『不变』
src **a**	累加器的位右移，位 7 移入进位标志位 例如： *src* a ; 结果： a (c,b7,b6,b5,b4,b3,b2,b1) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a(b0) 受影响的标志位： Z：『不变』， C：『受影响』， AC：『不变』， OV：『不变』
sr **M**	RAM 的位右移，位 7 移入值为 0 例如： *sr* MEM ; 结果： MEM(0,b7,b6,b5,b4,b3,b2,b1) ← MEM(b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM(b0) 受影响的标志位： Z：『不变』， C：『受影响』， AC：『不变』， OV：『不变』
src **M**	RAM 的位右移，位 7 移入进位标志位 例如： *src* MEM ; 结果： MEM(c,b7,b6,b5,b4,b3,b2,b1) ← MEM (b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM(b0) 受影响的标志位： Z：『不变』， C：『受影响』， AC：『不变』， OV：『不变』

©Copyright 2014, PADAUK Technology Co. Ltd

sl **a**	累加器的位左移，位 0 移入值为 0 例如： *sl* a ; 结果： a (b6,b5,b4,b3,b2,b1,b0,0) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a (b7) 受影响的标志位： Z：『不变』， C：『受影响』， AC：『不变』， OV：『不变』
slc **a**	累加器的位左移，位 0 移入进位标志位 例如： *slc* a ; 结果： a (b6,b5,b4,b3,b2,b1,b0,c) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a(b7) 受影响的标志位： Z：『不变』， C：『受影响』， AC：『不变』， OV：『不变』
sl **M**	RAM 的位左移，位 0 移入值为 0 例如： *sl* MEM ; 结果：MEM (b6,b5,b4,b3,b2,b1,b0,0) ← MEM (b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM(b7) 受影响的标志位： Z：『不变』， C：『受影响』， AC：『不变』， OV：『不变』
slc **M**	RAM 的位左移，位 0 移入进位标志位 Example: *slc* MEM ; 结果： MEM (b6,b5,b4,b3,b2,b1,b0,C) ← MEM (b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM (b7) 受影响的标志位： Z：『不变』， C：『受影响』， AC：『不变』， OV：『不变』
swap **a**	累加器的高 4 位与低 4 位互换 例如：*swap* a ; 结果： a (b3,b2,b1,b0,b7,b6,b5,b4) ← a (b7,b6,b5,b4,b3,b2,b1,b0) 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
swap **M**	RAM 的高 4 位与低 4 位互换 例如：*swap* MEM ; 结果： MEM (b3,b2,b1,b0,b7,b6,b5,b4) ← MEM (b7,b6,b5,b4,b3,b2,b1,b0) 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

7.4 逻辑运算类指令

and **a, I**	累加器和立即数据执行逻辑 AND，然后把结果保存到累加器 例如：*and* a, 0x0f ; 结果： a ← a & 0fh 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
and **a, M**	累加器和 RAM 执行逻辑 AND，然后把结果保存到累加器 例如：*and* a, RAM10 ; 结果： a ← a & RAM10 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
and **M, a**	累加器和 RAM 执行逻辑 AND，然后把结果保存到 RAM 例如： *and* MEM, a ; 结果： MEM ← a & MEM 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』

or **a, I**	累加器和立即数据执行逻辑 OR，然后把结果保存到累加器 例如： *or* a, 0x0f ; 结果： a ← a | 0fh 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
or **a, M**	累加器和 RAM 执行逻辑 OR，然后把结果保存到累加器 例如：*or* a, MEM ; 结果： a ← a | MEM 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
or **M, a**	累加器和 RAM 执行逻辑 OR，然后把结果保存到 RAM 例如： *or* MEM, a ; 结果： MEM ← a | MEM 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
xor **a, I**	累加器和立即数据执行逻辑 XOR，然后把结果保存到累加器 例如： *xor* a, 0x0f ; 结果： a ← a ^ 0fh 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』

xor **a, IO**	累加器和 IO 寄存器执行逻辑 XOR，然后把结果保存到累加器
	例如： *xor a, pa ;*
	结果： a ← a ^ pa ; // pa 是 port A 资料寄存器
	受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』

xor **IO, a**	Perform logic XOR on ACC and IO register, then put result into IO register
	例如： *xor pa, a ;*
	结果： pa ← a ^ pa ; // pa 是 port A 资料寄存器
	受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

xor **a, M**	累加器和 RAM 执行逻辑 XOR，然后把结果保存到累加器 Example: *xor* a, MEM ; 结果： a ← a ^ RAM10 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
xor **M, a**	累加器和 RAM 执行逻辑 XOR，然后把结果保存到 RAM 例如： *xor* MEM, a ; 结果： MEM ← a ^ MEM 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』

not **a**	累加器执行 1 补码运算，结果放在累加器
	例如： *not* a ;
	结果： a ← ～a
	受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
	应用范例：
	-- 	
	mov a, 0x38 ; // ACC=0X38
	not a ; // ACC=0XC7
	-- 	

not **M**	RAM 执行 1 补码运算，结果放在 RAM
	例如： *not* MEM ;
	结果： MEM ← ～MEM
	受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
	应用范例：
	-- 	
	mov a, 0x38 ;
	mov mem, a ; // mem = 0x38
	not mem ; // mem = 0xC7
	-- 	

neg **a**	累加器执行 2 补码运算，结果放在累加器
	例如： *neg* a;
	结果： a ←a 的 2 补码
	受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
	应用范例：
	-- 	
	mov a, 0x38 ; // ACC=0X38
	neg a ; // ACC=0XC8
	-- 	

neg **M**	RAM 执行 2 补码运算，结果放在 RAM
	例如： *neg* MEM;
	结果： MEM ← MEM 的 2 补码
	受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
	应用范例：
	-- 	
	mov a, 0x38 ;
	mov mem, a ; // mem = 0x38
	neg mem ; // mem = 0xC8
	-- 	

comp **a, I**	累加器和立即数据比较运算，影响的是标志，标志的改变与（a - I）运算相同
	例如： *comp* a, 0x55;
	结果： 标志的改变与 （a - 0x55）运算相同
	受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
	应用范例：
	-- 	
	mov a, 0x38 ;
	comp a, 0x38 ; // Z 标志位被设置为 1
	comp a, 0x42 ; // C 标志位被设置为 1
	comp a, 0x24 ; // C, Z 标志位被清除为 0
	comp a, 0x6a ; // C, AC 标志位被设置为 1
	-- 	

comp **a, M**	累加器和 RAM 比较运算，影响的是标志位，标志位的改变与（a - MEM）运算相同
	例如： *comp* a, MEM;
	结果：标志位的改变与 (a - MEM) 运算相同
	受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
	应用范例：
	-- 	
	mov a, 0x38 ;
	mov mem, a ;
	comp a, mem ; // Z 标志位被设置为 1
	mov a, 0x42 ;
	mov mem, a ;
	mov a, 0x38 ;
	comp a, mem ; // C 标志位被设置为 1
	-- 	

comp **M, a**	累加器和 RAM 比较运算，影响的是标志位，标志位的改变与（MEM - a）运算相同 例如： *comp* MEM, a; 结果：标志位的改变与 (MEM - a) 运算相同 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』

7.5 位运算类指令

set0 **IO.n**	IO 口的位 N 拉低电位 例如： *set0* pa.5 ; 结果： PA5=0 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
set1 **IO.n**	IO 口的位 N 拉高电位 例如：*set1* pb.5 ; 结果： PB5=1 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
tog **IO.n**	IO 口的位 N 反向 例如： *tog* pa.5 ; 结果： PA5=>1 假如 PA5=0 ; PA5=>0 假如 PA5=1 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
set0 M.n	RAM 的位 N 设为 0 例如：*set0* MEM.5 ; 结果： MEM 位 5 为 0 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
set1 **M.n**	RAM 的位 N 设为 1 例如： *set1* MEM.5 ; 结果： MEM 位 5 为 1 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

swapc **IO.n**	IO 口的第 N 位与进位标志位互换
	例如：*swapc* IO.0;
	结果： C ← IO.0 , IO.0 ← C
	当 IO.0 是输出脚位，进位标志 C 将被送到 IO.0 脚
	当 IO.0 是输入脚位，IO.0 脚的状态将被送到进位标志 C
	
	受影响的标志位： Z：『不变』， C：『受影响』， AC：『不变』， OV：『不变』
	应用范例 1： （串行输出） ：
	--- 	
	…
	set1 pac.0 ; // PA.0 设为输出
	…
	set0 flag.1 ; // C=0
	swapc pa.0 ; // 将 C 传送到 PA.0， PA.0=0
	set1 flag.1 ; // C=1
	swapc pa.0 ; //将 C 传送到 PA.0， PA.0=1
	…
	--- 	
	应用范例 2： （串行输入）
	--- 	
	…
	set0 pac.0 ; // PA.0 设为输入
	…
	swapc pa.0 ; // 把 PA.0 读到 C
	src a ; // 将 C 移到累加器的位 7
	swapc pa.0 ; //把 PA.0 读到 C
	src a ; //将新的 C 移到累加器的位 7
	…
	--- 	

7.6 条件运算类指令

ceqsn **a, I**	比较累加器与立即数据，如果是相同的，即跳过下一指令。标志位的改变与 (a ← a - I)相同 例如： *ceqsn* a, 0x55 ; *inc* MEM ; *goto* error ; 结果： 假如 a=0x55, then “goto error”; 否则, “inc MEM”. 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
ceqsn **a, M**	比较累加器与 RAM，如果是相同的，即跳过下一指令。标志位改变与 (a ← a - M)相同 例如：*ceqsn* a, MEM; 结果： 假如 a=MEM，跳过下一个指令 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
ceqsn **M, a**	比较累加器与 RAM，如果是相同的，即跳过下一指令。标志位改变与 (M ← M - a)相同 例如： *ceqsn* MEM, a; 结果： 假如 a=MEM，跳过下一个指令 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
cneqsn **a, M**	比较累加器与 RAM，如果是不相同，即跳过下一指令。标志位改变与 (a ← a - M)相同 例如： *ceqsn* a, MEM; 结果： 假如 a≠MEM，跳过下一个指令 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』

©Copyright 2014, PADAUK Technology Co. Ltd

cneqsn **M, a**	比较累加器与 RAM，如果是不相同，即跳过下一指令。标志位改变与 (M← M-a) 相同 例如： *ceqsn* MEM, a; 结果： 假如 a≠MEM，跳过下一个指令 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
cneqsn **a, I**	比较累加器与立即数据，如果是不相同，即跳过下一指令。标志位改变与 (a ← a - I)相同 例如： *ceqsn* a, *0x55*; *inc* *MEM ;* *goto* *error* ***;*** 结果： 假如 a≠0x55，将执行“goto error”; 否则执行 “inc MEM” 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
t0sn **IO.n**	如果 IO 的指定位是 0，跳过下一个指令。 例如：*t0sn* pa.5; 结果： 如果 PA5 是 0，跳过下一个指令。 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
t1sn **IO.n**	如果 IO 的指定位是 1，跳过下一个指令。 Example: *t1sn* pa.5 ; 结果：如果 PA5 是 1，跳过下一个指令。 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
t0sn **M.n**	如果 RAM 的指定位是 0，跳过下一个指令。 例如： *t0sn* MEM.5 ; 结果： 如果 MEM 的位 5 是 0，跳过下一个指令。 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
t1sn **M.n**	如果 RAM 的指定位是 1，跳过下一个指令。 例如： *t1sn* MEM.5 ; 结果： 如果 MEM 的位 5 是 1，跳过下一个指令。 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
izsn **a**	累加器加 1，若累加器新值是 0，跳过下一个指令。 例如： *izsn* a; 结果： a ← a + 1，若 a=0，跳过下一个指令。 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
dzsn **a**	累加器减 1，若累加器新值是 0，跳过下一个指令。 例如： *dzsn* a; 结果： a ← a － 1，若 a=0，跳过下一个指令。 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
izsn **M**	RAM 加 1，若 RAM 新值是 0，跳过下一个指令。 例如： *izsn* MEM; 结果： MEM ← MEM + 1，若 MEM=0，跳过下一个指令。 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
dzsn **M**	RAM 减 1，若 RAM 新值是 0，跳过下一个指令。 例如： *dzsn* MEM; 结果： MEM ← MEM – 1，若 MEM=0，跳过下一个指令。 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』

©Copyright 2014, PADAUK Technology Co. Ltd

***wait0* IO.n**	直到 IO 口的位 N 为 0，才转到下一个指令；否则，在这里等候。 例如： *wait0* pa.5; 结果： 等候 PA5=0 才转到下一个指令 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
***wait1* IO.n**	直到 IO 口的位 N 为 1，才转到下一个指令；否则，在这里等候。 例如： *wait1* pa.5; 结果： 等候 PA5=0 才转到下一个指令 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

7.7 系统控制类指令

***call* label**	函数调用，地址可以是全部空间的任一地址 例如： *call* function1; 结果： [sp] ← pc + 1 pc ← function1 sp ← sp + 2 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
***goto* label**	转到指定的地址，地址可以是全部空间的任一地址 例如： *goto* error; 结果： 跳到 error 并继续执行程序 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
***ret* I**	将立即数据复制到累加器，然后返回 例如： *ret* 0x55; 结果： A ← 55h ret ; 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
ret	从函数调用中返回原程序 例如： *ret;* 结果： sp ← sp – 2 pc ← [sp] 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
reti	从中断服务程序返回到原程序。在这指令执行之后，全部中断将自动启用。 例如： *reti*; 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
nop	没任何动作 例如： *nop*; 结果： 没任何改变 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
***pcadd* a**	目前的程序计数器加累加器是下一个程序计数器。 例如： *pcadd a*; 结果： pc ← pc + a 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

©Copyright 2014, PADAUK Technology Co. Ltd

<table>
<tr><td></td><td>应用范例：

    …
    mov   a, 0x02 ;
    pcadd   a ;   // PC <- PC+2
    goto   err1 ;
    goto   correct ;   // 跳到这里
    goto   err2 ;
    goto   err3 ;
    …
correct: // 跳到这里
    …
--</td></tr>
<tr><td>engint</td><td>允许全部中断。
例如： engint;
结果：中断要求可送到 FPP0，以便进行中断服务
受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』</td></tr>
<tr><td>disgint</td><td>禁止全部中断。
例如： disgint ;
结果：送到 FPP0 的中断要求全部被挡住，无法进行中断服务
受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』</td></tr>
<tr><td>stopsys</td><td>系统停止。
例如： stopsys;
结果：停止系统时钟和关闭系统
受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』</td></tr>
<tr><td>stopexe</td><td>CPU 停止。所有震荡器模块仍然继续工作并输出：但是系统时钟是被禁用以节省功耗。
例如： stopexe;
结果： 停住系统时钟，但是仍保持震荡器模块工作
受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』</td></tr>
<tr><td>reset</td><td>复位整个单片机，其运行将与硬件复位相同。
例如： reset;
结果：复位整个单片机
受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』</td></tr>
<tr><td>wdreset</td><td>复位看门狗定时器
例如： wdreset ;
结果： 复位看门狗定时器
受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』</td></tr>
</table>

7.8 指令执行周期综述

2 个周期	ldtabh, ldtabl, idxm
1 个周期	其它

7.9 指令影响标志的综述

Instruction	Z	C	AC	OV	Instruction	Z	C	AC	OV	Instruction	Z	C	AC	OV
mov a, I	-	-	-	-	*mov* M, a	-	-	-	-	*mov* a, M	Y	-	-	-
mov a, IO	Y	-	-	-	*mov* IO, a	-	-	-	-	*nmov* M, a	-	-	-	-
nmov a, M	Y	-	-	-	*ldtabh* index	-	-	-	-	*ldtabl* index	-	-	-	-
ldt16 index	-	-	-	-	*stt16* index					*xch* M	-	-	-	-
idxm a, index	-	-	-	-	*idxm* index, a	-	-	-	-	*wdreset*	-	-	-	-
pushaf	-	-	-	-	*popaf*	Y	Y	Y	Y	*add* a, I	Y	Y	Y	Y
add a, M	Y	Y	Y	Y	*add* M, a	Y	Y	Y	Y	*addc* a, M	Y	Y	Y	Y
addc M, a	Y	Y	Y	Y	*addc* a	Y	Y	Y	Y	*addc* M	Y	Y	Y	Y
nadd a, M	Y	Y	Y	Y	*nadd* M, a	Y	Y	Y	Y	*sub* a, I	Y	Y	Y	Y
sub a, M	Y	Y	Y	Y	*sub* M, a	Y	Y	Y	Y	*subc* a, M	Y	Y	Y	Y
subc M, a	Y	Y	Y	Y	*subc* a	Y	Y	Y	Y	*subc* M	Y	Y	Y	Y
inc M	Y	Y	Y	Y	*dec* M	Y	Y	Y	Y	*clear* M	-	-	-	-
sr a	-	Y	-	-	*src* a	-	Y	-	-	*sr* M	-	Y	-	-
src M	-	Y	-	-	*sl* a	-	Y	-	-	*slc* a	-	Y	-	-
sl M	-	Y	-	-	*slc* M	-	Y	-	-	*swap* a	-	-	-	-
swap M	-	-	-	-	*and* a, I	Y	-	-	-	*and* a, M	Y	-	-	-
and M, a	Y	-	-	-	*or* a, I	Y	-	-	-	*or* a, M	Y	-	-	-
or M, a	Y	-	-	-	*xor* a, I	Y	-	-	-	*xor* a, M	Y	-	-	-
xor M, a	Y	-	-	-	*xor* a, IO	Y	-	-	-	*xor* IO, a	-	-	-	-
not a	Y	-	-	-	*not* M	Y	-	-	-					
neg a	Y	-	-	-	*neg* M	Y	-	-	-	*comp* a, I	Y	Y	Y	Y
comp a, M	Y	Y	Y	Y	*comp* M, a	Y	Y	Y	Y	*set0* IO.n	-	-	-	-
set1 IO.n	-	-	-	-	*tog* IO.n	-	-	-	-	*set0* M.n	-	-	-	-
set1 M.n	-	-	-	-	*swapc* IO.n	-	Y	-	-	*ceqsn* a, I	Y	Y	Y	Y
ceqsn a, M	Y	Y	Y	Y	*ceqsn* M, a	Y	Y	Y	Y	*cneqsn* a, I	Y	Y	Y	Y
cneqsn a, M	Y	Y	Y	Y	*cneqsn* M, a	Y	Y	Y	Y	*t0sn* IO.n	-	-	-	-
t1sn IO.n	-	-	-	-	*t0sn* M.n	-	-	-	-	*t1sn* M.n	-	-	-	-
izsn a	Y	Y	Y	Y	*dzsn* a	Y	Y	Y	Y	*izsn* M	Y	Y	Y	Y
dzsn M	Y	Y	Y	Y	*wait0* IO.n	-	-	-	-	*wait1* IO.n	-	-	-	-
call label	-	-	-	-	*goto* label	-	-	-	-	*ret* I	-	-	-	-
ret					*reti*	-	-	-	-	*nop*	-	-	-	-
pcadd a					*engint*	-	-	-	-	*disgint*	-	-	-	-
stopsys					*reset*	-	-	-	-	*stopexe*	-	-	-	-

©Copyright 2014, PADAUK Technology Co. Ltd

8. 特别注意事项

此章节是提醒使用者在使用 PMC251 时避免一些常犯的错误。

8.1. 使用 IC 时

8.1.1. IO 使用与设定

(1) IO 作为模拟输入

- 将 IO 设为输入。
- 用 PADIER 和 PBDIER 寄存器，将对应的 IO 设为模拟输入。
- 用 PAPH 和 PBPH 寄存器，将对应的 IO 上拉电阻设为关闭。
- PMC251 芯片的 PADIER 与 PBDIER 寄存器，与 ICE 的功能极性是相反的，为了 ICE 仿真和 PMC251 芯片的程序能够一致，请用下列方法来编写程序：

```
$ PADIER 0xF0;
$ PBDIER 0x07;
```

(2) PA5 作为输出

PA5 只能做为 Open Drain 输出，高输出需要外加上拉电阻。

(3) PA5 做作为为 PRST#输入

- PA5 没有内部上拉电阻的功能。
- 设定 PA5 为输入。
- 设定 CLKMD.0=1，使 PA5 为外部 PRST#输入脚位。

(4) PA5 作为输入并通过长导线连接至按键或者开关。

- 必需在 PA5 与长导线中间串接 >10 欧电阻。
- 应尽量避免使用 PA5 作为输入。

(5) PA7 和 PA6 作为外部晶体振荡器引脚

- PA7 和 PA6 设定为输入。
- PA7 和 PA6 内部上拉电阻设为关闭。
- 用 PADIER 寄存器将 PA6 和 PA7 设为模拟输入。
- EOSCR 寄存器位[6:5]选择对应的晶体振荡器频率：
  - 01：低频，例如：32kHz。
  - 10：中频，例如：455kHz、1MHz。
  - 11：高频，例如：4MHz。
- EOSCR.7 设为 1，使能晶体振荡器。
- 从 IHRC 或 ILRC 切换到 EOSC，要先确认 EOSC 已经稳定振荡，参考 8.1.3.(2)。

©Copyright 2014, PADAUK Technology Co. Ltd

8.1.2. 中断

(1) 只有 FPPA0 能使用中断，也就是只有 FPPA0 才能使用 ENGINT 和 DISGINT 这两条指令。

使用中断功能的一般步骤如下：

步骤 1：设定 INTEN 寄存器，开启需要的中断的控制位。

步骤 2：清除 INTRQ 寄存器。

步骤 3：主程序中，使用 ENGINT 指令允许 FPPA0 的中断功能。

步骤 4：等待中断。中断发生后，跳入中断子程序。

步骤 5：当中断子程序执行完毕，返回主程序。

* 在主程序中，可使用 DISGINT 指令关闭所有中断。

* 跳入中断子程序处理时，可使用 PUSHAF 指令来保存 ALU 和 FLAG 寄存器数据，并在 RETI 之前，使用 POPAF 指令复原。一般步骤如下：

```
void Interrupt (void)    // 中断发生后，跳入中断子程序，
{                        // 自动进入 DISGINT 的状态，FPP0 不会再接受中断
        PUSHAF;
        …
        POPAF;
}   // 系统自动填入 RETI，直到执行 RETI 完毕才自动恢复到 ENGINT 的状态
```

(2) FPPA1 完全不受中断影响。

(3) INTEN，INTRQ 没有初始值，所以要使用中断前，一定要根据需要设定数值。

8.1.3. 切换系统时钟

(1) 利用 CLKMD 寄存器可切换系统时钟源。但必须注意，不可在切换系统时钟源的同时把原时钟源关闭。例如: 从 A 时钟源切换到 B 时钟源时，应该先用 CLKMD 寄存器切换系统时钟源，然后再透过 CLKMD 寄存器关闭 A 时钟源振荡器。

◆ 例一：系统时钟从 ILRC 切换到 IHRC/2

```
CLKMD   =   0x36;      // 切到 IHRC，但 ILRC 不要 disable。
CLKMD.2 =   0;         // 此时才可关闭 ILRC。
```

◆ 例二：系统时钟从 ILRC 切换到 EOSC

```
CLKMD   =   0xA6;      // 切到 EOSC，但 ILRC 不要 disable。
CLKMD.2 =   0;         // 此时才可关闭 ILRC。
```

◆ 错误的写法，ILRC 切换到 IHRC，同时关闭 ILRC

```
CLKMD   =   0x50;      // MCU 会当机。
```

©Copyright 2014, PADAUK Technology Co. Ltd

(2) 系统时钟从 ILRC 或 IHRC 切换到 EOSC 时，另一个重点是要先确认 EOSC 已经稳定振荡。MCU 并没有检查晶体振荡器是否已经稳定的功能，所以在程序中，透过设定 EOSCR 寄存器让 EOSC 起振后，需要延迟一段时间，等待 EOSC 稳定振荡后，才可以将系统时钟切换到 EOSC，否则会造成 MCU 当机。以开机后，系统时钟从 ILRC 切换到 4MHz EOSC 为例：

```
.ADJUST_IC     DISABLE
CLKMD.1 =      0;                    // 关闭 WDT，让后面 delay 指令不会 timeout
$   EOSCR      Enable, 4MHz;         // 4MHz EOSC 开始振荡。
//延迟 (Delay)一段时间等待 EOSC 稳定
$ T16M EOSC,/1,BIT10
Word Count = 0;
Stt16 Count;
Intrq.T16 = 0;
wait1       Intrq.T16;
CLKMD   =    0xA4;                   // ILRC -> EOSC;
CLKMD.2 =      0;                    // 关闭 ILRC，但不一定需要
```

延迟(Delay)等待时间需依照晶体震荡器以及板子的特性调整。如使用示波器测量晶体震荡器信号，请把示波器的探棒切到 x10 档，并从 PA6(X2)测量，避免影响震荡器。

8.1.4. 掉电模式、唤醒以及看门狗

(1) 当 ILRC 关闭时，看门狗也会失效。

(2) 在执行 STOPSYS 或 STOPEXE 命令之前，一定要关闭看门狗时钟，否则可能会因看门狗时钟溢位而让 IC 复位。在 ICE 模拟也有相同的问题。

(3) 当快速唤醒功能关闭时，看门狗的时钟源是 ILRC；当快速唤醒功能被使能时，看门狗的时钟源会自动切换成系统时钟，所以看门狗的溢位复位时间也因时钟源是系统时钟而变得很短。建议使用快速唤醒的步骤为：系统要进入 STOPSYS 之前，先将看门狗关闭，再打开快速唤醒功能；等系统从掉电模式中被唤醒，先关闭快速唤醒功能，再打开看门狗。这样可以避免系统被唤醒后，因看门狗时钟源是系统时钟而快速的复位。

8.1.5. TIMER 溢出时间

如果设定 T16M 计数器 BIT8 为 1 时产生中断，则第一次中断是在计数到 0x100 时发生（BIT8 从 0 变 1），第二次中断在计数到 0x300 时发生（BIT8 从 0 变 1）.所以设定 BIT8 为 1 时产生中断，就是计数每 512 次才中断一次。请注意，如果对 T16M 计数器重新设值，则下一次中断也将在 BIT8 从 0 变 1 时发生。

8.1.6. 延时指令

PMC251 不支持"delay"指令，需使用循环指令作延时。但编程时请注意，条件转移指令（如 ceqsn…）在判断条件成立与不成立时所耗费的指令周期是不同的，计算总体延时长度时，请考虑此差异并参考 8.1.8 的列表。另外，也可以使用 ".delay"(前面有加".")代替 "delay"。

©Copyright 2014, PADAUK Technology Co. Ltd

8.1.7. LVR

(1) Power On 时，VDD 必须超过 2.2V 才能成功启动，否则 IC 不会工作。

(2) 只有当 IC 正常启动后，LVR 设定(1.8V, 2.0V, 2.2V 等) 才会生效。

8.1.8. 单/双核模式下指令周期差异

PMC251 系列指令周期差异

指令	条件	单核心	双核心
goto, call		**2T**	1T
ceqsn, cneqsn, t0sn, t1sn, dzsn, izsn	判断条件成立	**2T**	1T
	判断条件不成立	1T	1T
ldtabh, ldtabl, idxm		2T	2T
Others		1T	1T

8.1.9. 烧录方法

PMC251 8pin 封装 IC : 把 Jumper 插在 P201CS14A 的位置，并把 IC 放置在 Socket 往后退 3 pin 的位置。

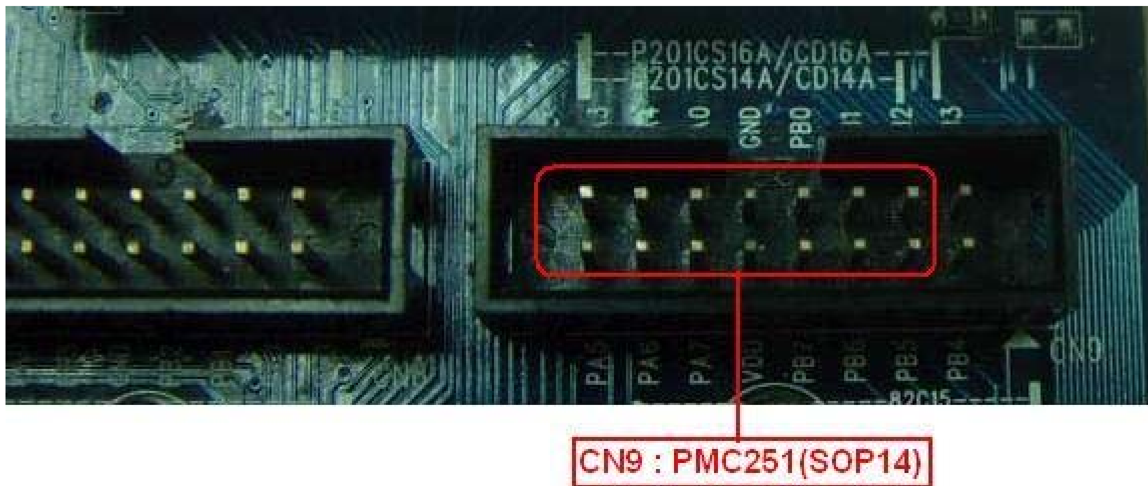
PMC251 其它封装 IC : 把 Jumper 插在 P201-14PIN 的位置，并把 IC 放置在 Socket 最前端的位置。

©Copyright 2014, PADAUK Technology Co. Ltd

8.2. 使用 IDE 时

8.2.1. PMC251 系列于仿真器 PDK3S-I-001/002/003 上仿真时:

PMC251 系列 I/O 脚位定义兼容于 PDK201 系列,以 PDK3S-I-001/002/003 仿真 PMC251 系列时，使用卷标标示为 CN9：P201CS14A/CD14A 的 Cable 连接于 PDK3S-I-001/002/003 仿真器上的 CN9 连接座上。接法如下图示：



8.2.2. 使用 PDK3S-I-001/002/003 仿真 PMC251 系列功能時注意事項:

下列为 PDK3S-I-001/002/003 **不支持**仿真 PMC251 系列的功能,而必需以 PMC251 Real Chip 来测试。请注意：为避免仿真程序与 Real chip 程序不同，在支持这些功能的仿真器没提供之前，datasheet 将会暂时移除，以避免仿真程序与 Real Chip 程序不同的困扰。下面这些功能在 PMC251 Real Chip 都是存在而且正常的，若客户要使用这些功能，请客户自行审慎处理仿真与 Real Chip 的程序问题。

(a) PMC251 系列 LVD 设定,最低可到 1.8V,共支持 8 阶 4.0V, 3.5V, 3.0V, 2.75V, 2.5V, 2.2V, 2.0V, 1.8V 功能
(b) PMC251 系列 LVD 功能可由缓存器(misc.2)来关闭功能
(c) PMC251 系列可由缓存器(misc.3)设定来支持 LVD 复位(reset)快速回复(Fast recover)功能
(d) PMC251 系列可设定为单核心工作模式功能
(e) PMC251 系列支持 VDD 低于 4V, 3V, 2V 电压准位侦测并将侦测结果存在内部缓存器(rstst)功能
(f) PMC251 系列支持复位(Reset)来源侦测功能
(g) PMC251 系列 Timer 16 Clock Source 支持外部 PA4 讯号输入功能
(h) Watch-dog 溢出时间，或由 misc[1:0]来选择

©Copyright 2014, PADAUK Technology Co. Ltd