



PMC131/PMS131/PMS130 系列
带 12 位 ADC、8 位单片机
数据手册

第 1.07 版
2019 年 12 月 26 日

Copyright © 2019 by PADAUK Technology Co., Ltd., all rights reserved

6F-6, No.1, Sec. 3, Gongdao 5th Rd., Hsinchu City 30069, Taiwan, R.O.C.

TEL: 886-3-572-8688  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，建议参考本档英文版。

目 录

1. 单片机特点	7
1.1. 系列特点	7
1.2. 系统功能	7
1.3. 高性能 RISC CPU 架构	8
1.4. 封装信息	8
2. 系统概述和方框图	9
3. PMC131/PMS131/PMS130 系列引脚功能描述	10
4. 器件电气特性	15
4.1. 直流/交流特性	15
4.2. 最大范围	17
4.3. ILRC 频率与 VDD 关系的曲线图	18
4.4. IHRC 频率与 VDD 关系的曲线图 (校准至 16MHz)	18
4.5. ILRC 频率与温度关系曲线图	19
4.6. IHRC 频率与温度关系曲线图 (校准至 16MHz)	19
4.7. 工作电流量测值 @系统时钟= ILRC÷N	20
4.8. 工作电流量测值 @系统时钟= IHRC÷N	20
4.9. 工作电流量测值 @系统时钟= 4MHz 晶振 EOSC÷N	21
4.10. 工作电流量测值 @系统时钟= 32kHz 晶振 EOSC÷N	21
4.11. 工作电流量测值 @系统时钟= 1MHz 晶振 EOSC÷N	22
4.12. IO 引脚输出驱动电流(I _{OH})和灌电流(I _{OL})曲线图	22
4.13. 测量的 IO 输入阈值电压(V _{IH} /V _{IL})	23
4.14. IO 引脚拉高阻抗曲线图	23
4.15. 省电模式和掉电模式消耗电流	24
5. 功能概述	25
5.1. OTP 程序存储器	25
5.2. 启动程序	25
5.2.1. 复位时序图	26
5.3. 数据存储	27
5.4. 振荡器和时钟	27
5.4.1. 内部高频振荡器 (IHRC)和低频振荡器 (ILRC)	27
5.4.2. 单片机校准	28
5.4.3. IHRC 频率校准和系统时钟	28
5.4.4. 外部晶体振荡器	29
5.4.5. 系统时钟和 LVR 水平	31
5.4.6. 系统时钟切换	32
5.5. 16 位计数器 (TIMER16)	33
5.6. 8 位 PWM 计数器(TIMER2/TIMER3)	35
5.6.1. 使用 Timer2 产生定期波形	36
5.6.2. 使用 Timer2 产生 8 位 PWM 波形	37

5.6.3.	使用 Timer2 产生 6 位 PWM 波形	39
5.7.	看门狗计数器	40
5.8.	中断	41
5.9.	省电与掉电模式	43
5.9.1.	省电模式 (stopexe)	44
5.9.2.	掉电模式 (stopsys)	45
5.9.3.	唤醒	46
5.10.	IO 设置	47
5.11.	复位和 LVR	48
5.11.1.	复位	48
5.11.2.	LVR 复位	48
5.12.	数字转换(ADC)模块	49
5.12.1.	AD 转换的输入要求	50
5.12.2.	ADC 分辨率选择	51
5.12.3.	选择参考高电压	51
5.12.4.	ADC 时钟选择	51
5.12.5.	AD 转换	51
5.12.6.	模拟引脚的配置	52
5.12.7.	使用 ADC	52
5.13.	乘法器	53
6.	IO 寄存器	54
6.1.	算术逻辑状态寄存器 (FLAG), IO 地址 = 0x00	54
6.2.	堆栈指针寄存器 (SP), IO 地址 = 0x02	54
6.3.	时钟控制寄存器 (CLKMD), IO 地址 = 0x03	54
6.4.	中断允许寄存器 (INTEN), IO 地址 = 0x04	55
6.5.	中断请求寄存器 (INTRQ), IO 地址 = 0x05	55
6.6.	乘法器运算域寄存器 (MULOP), IO 地址 = 0x08	55
6.7.	乘法器结果高字节寄存器 (MULRH), IO 地址 = 0x09	55
6.8.	TIMER16 控制寄存器 (T16M), IO 地址 = 0x06	56
6.9.	外部晶体振荡器控制寄存器 (EOSCR), IO 地址 = 0x0A	56
6.10.	中断边沿选择寄存器 (INTEGS), IO 地址 = 0x0C	57
6.11.	端口 A 数字输入使能寄存器 (PADIER), IO 地址 = 0x0D	57
6.12.	端口 B 数字输入使能寄存器 (PBDIER), IO 地址 = 0x0E	58
6.13.	端口 A 数据寄存器 (PA), IO 地址 = 0x10	58
6.14.	端口 A 控制寄存器 (PAC), IO 地址 = 0x11	58
6.15.	端口 A 上拉控制寄存器 (PAPH), IO 地址 = 0x12	58
6.16.	端口 B 数据寄存器 (PB), IO 地址 = 0x14	58
6.17.	端口 B 控制寄存器 (PBC), IO 地址 = 0x15	58
6.18.	端口 B 上拉控制寄存器 (PBPH), IO 地址 = 0x16	58
6.19.	ADC 控制寄存器 (ADCC), IO 地址 = 0x20	59
6.20.	ADC 调节控制寄存器 (ADCRGC), IO 地址 = 0x1C	59
6.21.	ADC 模式控制寄存器 (ADCM), IO 地址 = 0x21	60
6.22.	ADC 数据高位寄存器 (ADCRH), IO 地址 = 0x22	60
6.23.	ADC 数据低位寄存器 (ADCRL), IO 地址 = 0x23	60
6.24.	杂项寄存器 (MISC), IO 地址 = 0x1B	61
6.25.	TIMER2 控制寄存器 (TM2C), IO 地址 = 0x3C	62
6.26.	TIMER2 计数寄存器 (TM2CT), IO 地址 = 0x3D	62

6.27. TIMER2 分频器寄存器 (TM2S), IO 地址 = 0x37.....	62
6.28. TIMER2 上限寄存器 (TM2B), IO 地址= 0x09.....	63
6.29. TIMER3 控制寄存器 (TM3C), IO 地址= 0x2E.....	63
6.30. TIMER3 控制寄存器 (TM3CT), IO 地址 = 0x2F.....	63
6.31. TIMER3 分频寄存器 (TM3S), IO 地址 = 0x39.....	64
6.32. TIMER3 上限寄存器 (TM3B), IO 地址 = 0x23.....	64
6.33. 复位状态寄存器 (RSTST), IO 地址 = 0x25.....	64
7. 指令.....	65
7.1. 数据传输类指令.....	66
7.2. 算术运算类指令.....	68
7.3. 移位元运算类指令.....	70
7.4. 逻辑运算类指令.....	71
7.5. 位运算类指令.....	73
7.6. 条件运算类指令.....	74
7.7. 系统控制类指令.....	75
7.8. 指令执行周期综述.....	76
7.9. 指令影响标志的综述.....	77
7.10. RAM 定义限制.....	77
8. 代码选项(CODE OPTIONS).....	78
9. 特别注意事项.....	79
9.1. 警告.....	79
9.2. 使用 IC 时.....	79
9.2.1. IO 使用与设定.....	79
9.2.2. 中断.....	81
9.2.3. 系统时钟.....	81
9.2.4. 掉电模式、唤醒以及看门狗.....	82
9.2.5. TIMER16 溢出时间.....	83
9.2.6. IHRC.....	83
9.2.7. LVR.....	83
9.2.8. 烧录方法.....	83
9.3. 使用 ICE 时.....	84

修订历史:

修订	日期	描述
0.01	2013/12/02	初版
0.02	2014/02/18	增加章节 8 特别注意事项
0.03	2014/06/10	增加章节 8.1.1 之(7)关于 PB3 IO 口使用
0.04	2014/12/22	修正 PMS131/PMS130 工作温度
0.05	2015/06/17	1. 修正 PMS131/PMS130 工作温度为-20°C ~ 70°C 2. 修正 4.1 Band-gap 参考电压
1.06	2018/11/07	1. 更新应广科技股份有限公司地址与电话 2. 修改第 1.1 节系列特点 3. 修改 1.4.封装信息 4. 增加第 3 章引脚功能说明图和描述: PMC131-S08A, PMC131-D08A, PMC131-S08B, PMC131-D08B, PMC131-U06, PMS131-S08, PMS130-S08, PMS130-D08, PMS130-U06 5. 第 3 章新增 AVDD 和 AGND 6. 修改 4.7.工作电流量测值 @系统时钟= ILRC÷n 7. 修改 4.8.工作电流量测值 @系统时钟= IHRC÷n 8. 修改 4.9.工作电流量测值 @系统时钟= 4MHz 晶振 EOSC÷n 9. 修改 4.10.工作电流量测值 @系统时钟= 32kHz 晶振 EOSC÷n 10. 修改 4.11.工作电流量测值 @系统时钟= 1MHz 晶振 EOSC÷n 11. 修改 4.12. IO 引脚输出驱动电流(I _{OH})和灌电流(I _{OL})曲线图 12. 修改 4.13.测量的 IO 输入阈值电压(V _{IH} /V _{IL}) 13. 修改 4.14 .IO 引脚拉高阻抗曲线图 14. 新增 4.15.省电模式和掉电模式消耗电流 15. 修改表 2: PMC131/PMS131/PMS130 内置 3 个振荡器电路 16. 修改 5.4.5.系统时钟和 LVR 水平 17. 修改 5.5. 16 位计数器 18. 修改 5.6. 8 位 PWM 计数器 19. 修改 5.8.中断 20. 修改 5.9.1.省电模式 21. 修改 5.9.2.掉电模式 22. 修改 5.11.1.复位 23. 修改图 13: 模拟输入模型 24. 修正 5.12.7.使用 ADC 25. 修改 6.11.端口 A 数字输入使能寄存器 26. 修改 6.12. 端口 B 数字输入使能寄存器 27. 增加第 8 章代码选项 28. 增加 9.1. 警告 29. 修改 9.2.1 IO 使用与设定 30. 增加 9.2.6. IHRC 31. 修改 9.2.7. LVR 32. 9.2.9. RAM 定义限制移至 7.10 33. 修改 9.2.9.烧录方法 34. 修改 9.3.使用 ICE 时
1.07	2019/12/26	1. 修改 1.2、4.3、4.4、4.5、4.9、4.10、4.11、5.6、5.6.1、5.6.2、5.6.3、5.9.1、5.9.3、5.12.7、6.3、6.11、6.12、6.33、7.8、9.2.8 2. 修改 4.1: V _{IL} 3. 删除 4.16 并新增 5.2.1 4. 修改图 10: 掉电模式和省电模式在唤醒源的差异 5. 修改第 3 章、第 8 章 6. 删除 9.2.8 新增功能

1. 单片机特点

1.1. 系列特点

- ◆ PMC131 系列:
 - ◇ 高抗干扰 (High EFT)系列
 - ◇ 工作温度范围: $-40^{\circ}\text{C} \sim 85^{\circ}\text{C}$
- ◆ PMS131, PMS130 系列:
 - ◇ 通用系列
 - ◇ 不建议使用于 AC 阻容降压供电或有高 EFT 要求的应用。应广不对使用于此类应用而不达安规要求负责
 - ◇ 工作温度范围: $-20^{\circ}\text{C} \sim 70^{\circ}\text{C}$

1.2. 系统功能

- ◆ 1.5K OTP 程序存储器
- ◆ 88 Bytes 数据存储器
- ◆ 内置一个硬件 16 位计数器
- ◆ 内置两个硬件 8 位计数器并可提供 PWM 模式输出
- ◆ 提供单周期 (1T) 8x8 硬件乘法器
- ◆ 提供快速唤醒模式
- ◆ 14 个 IO 引脚
- ◆ 每一 IO 引脚都可以单独设置系统唤醒功能
- ◆ 内置 Band-gap 硬件模块输出 1.20V 参考电压
- ◆ 内置一个 12 通道 12 位分辨率 A/D 转换器, 其中一个通道来自于内部 band-gap 1.2V 或者 $0.24 \cdot V_{\text{DD}}$
- ◆ 工作电压: 2.2V ~ 5.5V
- ◆ 工作频率与电压 (晶体模式与 IHRC 内部高频 RC 振荡器模式相同)
DC ~ 8MHz@ $V_{\text{DD}} \geq 3.1\text{V}$ DC ~ 4MHz@ $V_{\text{DD}} \geq 2.5\text{V}$ DC ~ 2MHz@ $V_{\text{DD}} \geq 2.2\text{V}$
- ◆ 提供 ADC 参考高电压: 外部输入, 内部 V_{DD} , Band-gap 1.20V, 4V, 3V, 2V
- ◆ 时钟源: 内部高频 RC 振荡器(IHRC)、内部低频 RC 振荡器(ILRC)、外部晶振
- ◆ 8 段 LVR 设定: 4.1V, 3.6V, 3.1V, 2.8V, 2.5V, 2.2V, 2.0V, 1.8V
- ◆ 四个可选择的外部中断引脚

1.3. 高性能 RISC CPU 架构

- ◆ 工作模式：单一处理单元运作模式
- ◆ 提供 86 条指令
- ◆ 大部份指令都是单周期(1T)指令
- ◆ 弹性化的堆栈深度，可程序设定
- ◆ 提供数据与指令的直接、间接寻址模式
- ◆ 数据存取支持直接和间接寻址模式，用数据存储器即可当作间接寻址模式的数据指针(index pointer)
- ◆ IO 地址以及存储地址空间互相独立

1.4. 封装信息

◆ PMC131 系列

- ◇ PMC131-S16A: SOP16 (150mil);
- ◇ PMC131-S16B: SOP16 (150mil);
- ◇ PMC131-D16A: DIP16 (300mil);
- ◇ PMC131-D16B: DIP16 (300mil);
- ◇ PMC131-S14: SOP14 (150mil);
- ◇ PMC131-D14: DIP14 (300mil);
- ◇ PMC131-M10: MSOP10 (118mil);
- ◇ PMC131-S08A: SOP8 (150mil);
- ◇ PMC131-S08B: SOP8 (150mil);
- ◇ PMC131-D08A: DIP8 (300mil);
- ◇ PMC131-D08B: DIP8 (300mil);
- ◇ PMC131-U06: SOT23-6 (60mil)

◆ PMS131 系列

- ◇ PMS131-S16: SOP16 (150mil);
- ◇ PMS131-D16: DIP16 (300mil);
- ◇ PMS131-S14: SOP14 (150mil);
- ◇ PMS131-D14: DIP14 (300mil);
- ◇ PMS131-S08: SOP8 (150mil)

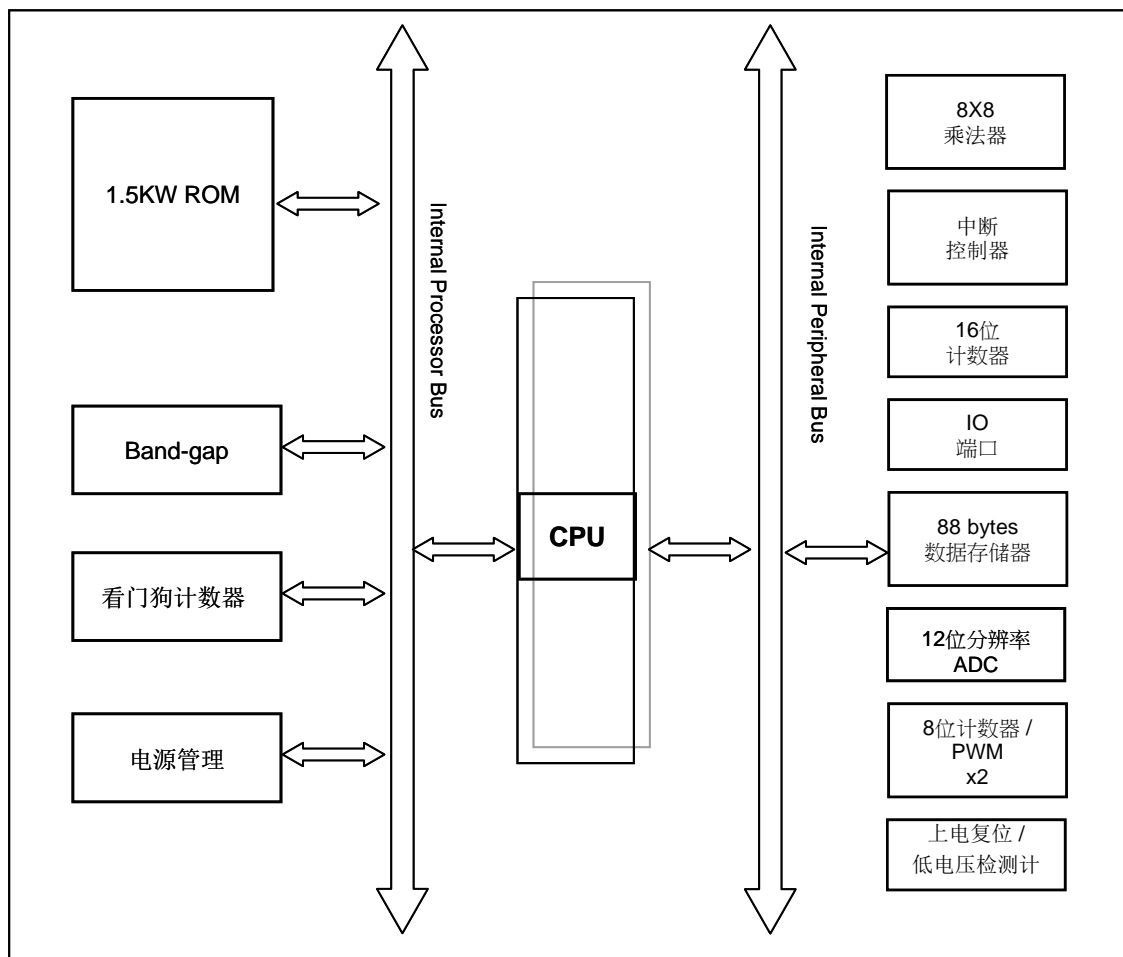
◆ PMS130 系列

- ◇ PMS130-M10: MSOP10 (118mil);
- ◇ PMS130-S08: SOP8 (150mil);
- ◇ PMS130-D08: DIP8 (300mil);
- ◇ PMS130-U06: SOT23-6 (60mil)

2. 系统概述和方框图

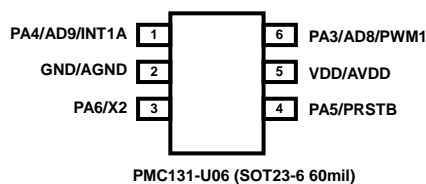
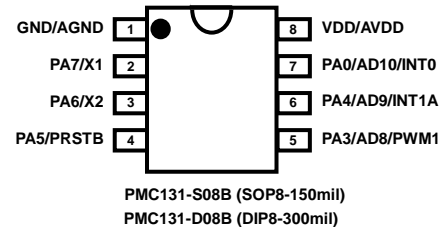
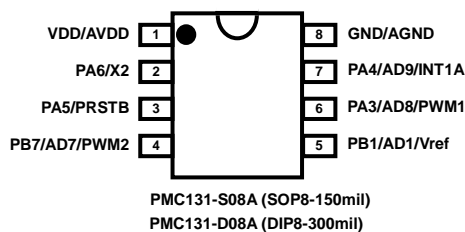
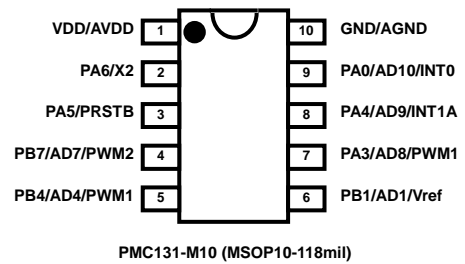
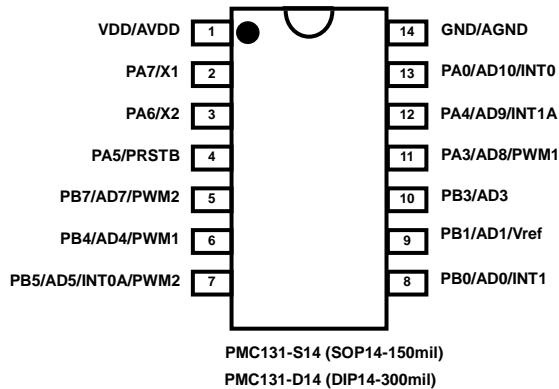
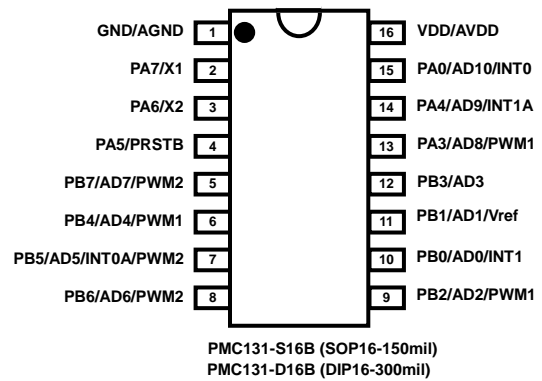
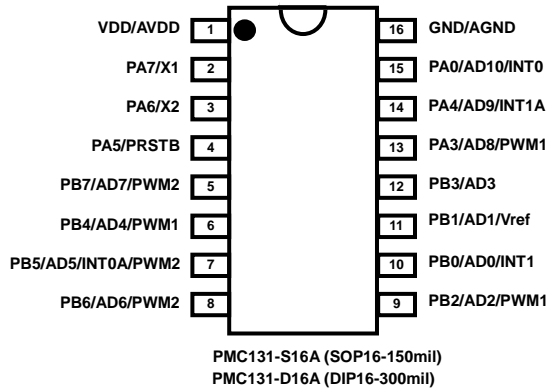
PMC131/PMS131/PMS130 系列是一个带 12 位 ADC，以 OTP 为程序存储基础的 CMOS 8 位微处理器。它运用 RISC 的架构基础使大部分的指令执行时间都是一个指令周期，只有少部分指令是需要两个指令周期。

在 PMC131/PMS131/PMS130 内部有 1.5Kx14 bit OTP 程序存储器以及 88 Bytes 数据存储器，芯片内部还设置有 12 通道 12 位分辨率 A/D 转换器，其中 1 通道为内置的 Band-gap 参考电压生成器，它可以提供 1.2V 电压供测量；另外，PMC131/PMS131/PMS130 提供 3 组硬件计数器 (Timer)，一个为 16 位计数器，另外两个为 8 位计数器并且可产生 PWM 波形。



3. PMC131/PMS131/PMS130 系列引脚功能描述

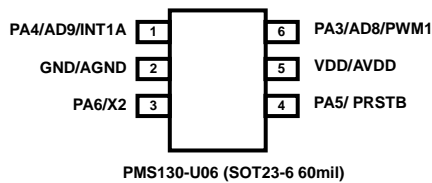
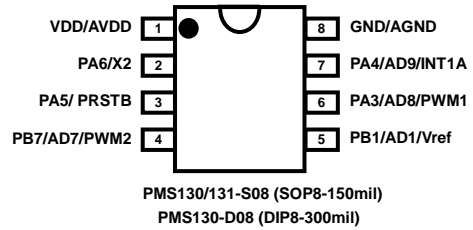
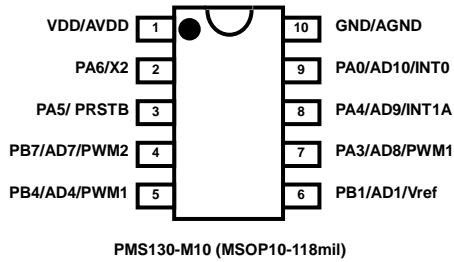
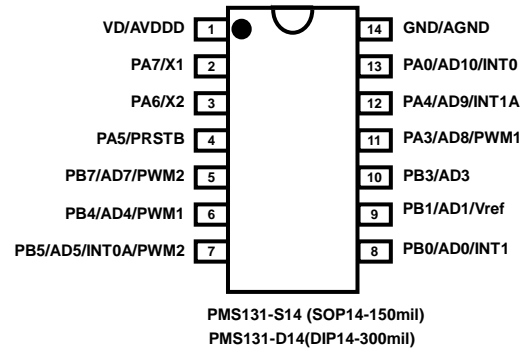
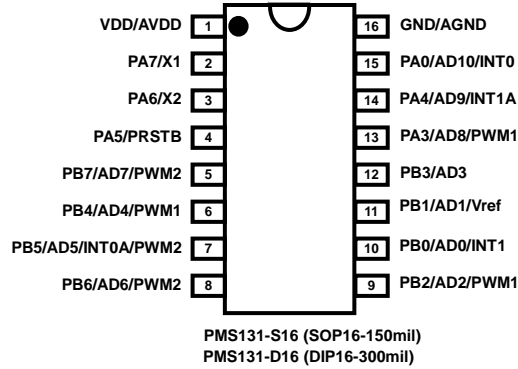
◆ PMC131 系列



PMC131/PMS131/PMS130 系列

帶 12 位 ADC、8 位单片机

◆ PMS131/PMS130 系列



引脚功能说明

引脚名称	电器型态	功能描述
PA7 / X1	IO ST / CMOS	此引脚可用作： (1) Port A 位 7，并可编程设定为数字输入或输出，弱上拉电阻模式。 (2) 使用晶体振荡器时，作 X1 用。 当此引脚设定为晶体振荡功能时，请用寄存器 <i>padier</i> 位 7 关闭("0")此引脚的数字输入以减少漏电流。此外，亦可设定在睡眠中唤醒系统的功能；但是，当寄存器 <i>padier</i> 位 7 为"0"时，唤醒功能是被关闭的。
PA6 / X2	IO ST / CMOS	此引脚可用作： (1) Port A 位 6，可编程设定为数字输入或输出，弱上拉电阻模式。 (2) 使用晶体振荡器时，作 X2 用。 当此引脚设定为晶体振荡功能时，请用寄存器 <i>padier</i> 位 6 关闭("0")此引脚的数字输入以减少漏电流。此外，亦可设定在睡眠中唤醒系统的功能；但是，当寄存器 <i>padier</i> 位 6 为"0"时，唤醒功能是被关闭的。
PA5 / PRSTB	IO (OD) ST / CMOS	此引脚可用作： (1) 当单片机的硬件外部复位。 (2) 当 Port A 位 5；此引脚可以设定为输入或开漏输出(open drain)模式。 <u>请注意 PA5 没有上拉电阻。</u> 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 <i>padier</i> 位 5 为"0"时，唤醒功能是被关闭的。 <u>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。</u>
PA4 / AD9 / INT1A	IO ST / CMOS / Analog	此引脚可用作： (1) Port A 位 4，这个引脚可编程设定为数字输入或输出，弱上拉电阻模式。 (2) ADC 模拟输入通道 9。 (3) 外部中断服务。外部中断可发生在上升沿和下降沿。 当此引脚设定为模拟输入时，请用寄存器 <i>pbdier</i> 位 4 关闭("0")此引脚的数字输入以减少漏电流。当此引脚设定禁用数字输入，在掉电模式的唤醒功能将同时被禁用。
PA3 / AD8 / PWM1	IO ST / CMOS / Analog	此引脚可用作： (1) Port A 位 3，这个引脚可编程设定为数字输入或输出，弱上拉电阻模式。 (2) ADC 模拟输入通道 8。 (3) Timer2 的 PWM 输出。 当此引脚设定为模拟输入时，请用寄存器 <i>padier</i> 位 3 关闭("0")此引脚的数字输入以减少漏电流。当此引脚设定禁用数字输入，在掉电模式的唤醒功能将同时被禁用。
PA0 / AD10 / INT0	IO ST / CMOS / Analog	此引脚可用作： (1) Port A 位 0，这个引脚可编程设定为数字输入或输出，弱上拉电阻模式。 (2) ADC 模拟输入通道 10。 (3) 外部中断服务。外部中断可发生在上升沿和下降沿。 当此引脚设定为模拟输入时，请用寄存器 <i>padier</i> 位 0 关闭("0")此引脚的数字输入以减少漏电流。当此引脚设定禁用数字输入，在掉电模式的唤醒功能将同时被禁用。

引脚名称	电器型态	功能描述
PB7 / AD7 / PWM2	IO ST / CMOS / Analog	<p>此引脚可用作：</p> <p>(1) Port B 位 7，这个引脚可编程设定为数字输入或输出，弱上拉电阻模式。</p> <p>(2) ADC 模拟输入通道 7。</p> <p>(3) Timer3 的 PWM 输出。</p> <p>当此引脚设定为模拟输入时，请用寄存器 <i>pbdier</i> 位 7 关闭（"0"）此引脚的数字输入以减少漏电流。当此引脚设定禁用数字输入，在掉电模式的唤醒功能将同时被禁用。</p>
PB6 / AD6 / PWM2	IO ST / CMOS / Analog	<p>此引脚可用作：</p> <p>(1) Port B 位 6，这个引脚可编程设定为数字输入或输出，弱上拉电阻模式。</p> <p>(2) ADC 模拟输入通道 6。</p> <p>(3) Timer3 的 PWM 输出。</p> <p>当此引脚设定为模拟输入时，请用寄存器 <i>pbdier</i> 位 6 关闭（"0"）此引脚的数字输入以减少漏电流。当此引脚设定禁用数字输入，在掉电模式的唤醒功能将同时被禁用。</p>
PB5 / AD5 / INT0A / PWM2	IO ST / CMOS / Analog	<p>此引脚可用作：</p> <p>(1) Port B 位 5，这个引脚可编程设定为数字输入或输出，弱上拉电阻模式。</p> <p>(2) ADC 模拟输入通道 5。</p> <p>(3) 外部中断服务。外部中断可发生在上升沿和下降沿。</p> <p>(4) Timer3 的 PWM 输出。</p> <p>当此引脚设定为模拟输入时，请用寄存器 <i>pbdier</i> 位 5 关闭（"0"）此引脚的数字输入以减少漏电流。当此引脚设定禁用数字输入，在掉电模式的唤醒功能将同时被禁用。</p>
PB4 / AD4 / PWM1	IO ST / CMOS / Analog	<p>此引脚可用作：</p> <p>(1) Port B 位 4，这个引脚可编程设定为数字输入或输出，弱上拉电阻模式。</p> <p>(2) ADC 模拟输入通道 4。</p> <p>(3) Timer2 的 PWM 输出。</p> <p>当此引脚设定为模拟输入时，请用寄存器 <i>pbdier</i> 位 4 关闭（"0"）此引脚的数字输入以减少漏电流。当此引脚设定禁用数字输入，在掉电模式的唤醒功能将同时被禁用。</p>
PB3 / AD3 /	IO ST / CMOS / Analog	<p>此引脚可用作：</p> <p>(1) Port B 位 3，这个引脚可编程设定为数字输入或输出，弱上拉电阻模式。</p> <p>(2) ADC 模拟输入通道 3。</p> <p>当此引脚设定为模拟输入时，请用寄存器 <i>pbdier</i> 位 3 关闭（"0"）此引脚的数字输入以减少漏电流。当此引脚设定禁用数字输入，在掉电模式的唤醒功能将同时被禁用。</p>
PB2 / AD2 / PWM1	IO ST / CMOS / Analog	<p>此引脚可用作：</p> <p>(1) Port B 位 2，这个引脚可编程设定为数字输入或输出，弱上拉电阻模式。</p> <p>(2) ADC 模拟输入通道 2。</p> <p>(3) Timer2 的 PWM 输出。</p> <p>当此引脚设定为模拟输入时，请用寄存器 <i>pbdier</i> 位 2 关闭（"0"）此引脚的数字输入以减少漏电流。当此引脚设定禁用数字输入，在掉电模式的唤醒功能将同时被禁用。</p>

引脚名称	电器型态	功能描述
PB1 / AD1 / Vref	IO ST / CMOS / Analog	<p>此引脚可用作：</p> <p>(1) Port B 位 1，这个引脚可编程设定为数字输入或输出，弱上拉电阻模式。</p> <p>(2) ADC 模拟输入通道 1。</p> <p>(3) ADC 的外部参考高电压。</p> <p>当此引脚设定为模拟输入时，请用寄存器 <i>pbdir</i> 位 1 关闭 (“0”) 此引脚的数字输入以减少漏电流。当此引脚设定禁用数字输入，在掉电模式的唤醒功能将同时被禁用。</p>
PB0 / AD0 / INT1	IO ST / CMOS / Analog	<p>此引脚可用作：</p> <p>(1) Port B 位 0，这个引脚可编程设定为数字输入或输出，弱上拉电阻模式。</p> <p>(2) ADC 模拟输入通道 0。</p> <p>(3) 外部中断服务。外部中断可发生在上升沿和下降沿。</p> <p>当此引脚设定为模拟输入时，请用寄存器 <i>pbdir</i> 位 0 关闭 (“0”) 此引脚的数字输入以减少漏电流。当此引脚设定禁用数字输入，在掉电模式的唤醒功能将同时被禁用。</p>
VDD / AVDD	VDD / AVDD	<p>VDD: 数字正电源</p> <p>AVDD: 模拟正电源</p> <p>VDD 是 IC 电源，而 AVDD 是 ADC 专用电源。在 IC 内部，AVDD 与 VDD 连在一起(double bonding)，而外部为相同引脚。</p>
GND / AGND	GND / AGND	<p>GND: 数字负电源</p> <p>AGND: 模拟负电源</p> <p>GND 是 IC 接地引脚，而 AGND 是 ADC 接地引脚。在 IC 内部，AGND 与 GND 连在一起(double bonding)，而外部为相同引脚。</p>
<p>注意： IO: 输入/输出； ST: 施密特触发； OD: 开漏输出； Analog: 模拟输入； CMOS: CMOS 电压准位</p>		

4. 器件电气特性

4.1. 直流/交流特性

下列所有数据除特别列明外，皆于 $V_{DD}=5.0V$, $f_{SYS}=2MHz$ 之条件下获得。

符号	特性	最小值	典型值	最大值	单位	条件 (Ta=25°C)
V_{DD}	工作电压	2.2	5.0	5.5	V	
f_{SYS}	系统时钟(CLK)* = IHRC/2 IHRC/4 IHRC/8 ILRC	0 0 0	37K	8M 4M 2M	Hz	Under_20ms_V _{DD} _ok** = Y/N $V_{DD} \geq 2.5V / V_{DD} \geq 3.1V$ $V_{DD} \geq 2.2V / V_{DD} \geq 2.5V$ $V_{DD} \geq 2.2V / V_{DD} \geq 2.2V$ $V_{DD} = 5.0V$
I_{OP}	工作电流		1.7 15		mA uA	$f_{SYS}=IHRC/16=1MIPS@5.0V$ $f_{SYS}=ILRC=21kHz@3.3V$
I_{PD}	掉电电流 (使用 stopsys 指令)		2.0 1.0		uA uA	$f_{SYS}= 0Hz, V_{DD}=5.0V$ $f_{SYS}= 0Hz, V_{DD}=3.3V$
I_{PS}	省电电流 (使用 stopexe 指令)		0.3		mA	$V_{DD}=5.0V$; Band-gap, LVR, IHRC, ILRC, Timer16 硬件模块启用
V_{IL}	输入低电压	0		$0.2 V_{DD}$	V	
V_{IH}	输入高电压	$0.7 V_{DD}$		V_{DD}	V	
I_{OL}	IO 引脚灌电流 限于 PA5	8 3	11 4	14 5.5	mA	$V_{DD} =5.0V, V_{OL}=0.5V$
I_{OH}	IO 引脚驱动电流	-6	-8	-10	mA	$V_{DD} =5.0V, V_{OH}=4.5V$
V_{IN}	输入电压	-0.3		$V_{DD}+0.3$	V	
$I_{INJ}(PIN)$	脚位的引入电流			1	mA	$V_{DD}+0.3 \geq V_{IN} \geq -0.3$
R_{PH}	上拉电阻		62 100 210		KΩ	$V_{DD} =5.0V$ $V_{DD} =3.3V$ $V_{DD} =2.2V$
V_{LVR}	低电压复位电压	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	
V_{BG}	Band-gap 参考电压 (校准前)	1.11	1.20	1.29	V	$V_{DD}=5V, 25^{\circ}C$
	Band-gap 参考电压 (校准后)	1.140* 1.145*	1.200* 1.200*	1.260* 1.255*		$V_{DD}=2.2V \sim 5.5V,$ $-40^{\circ}C < Ta < 85^{\circ}C^*$ $-20^{\circ}C < Ta < 70^{\circ}C^*$

符号	特性	最小值	典型值	最大值	单位	条件 (Ta=25°C)
f _{IHRC}	IHRC 频率* (校准后)	15.76*	16*	16.24*	MHz	25°C, V _{DD} =2.2V~5.5V
		15.04*	16*	16.96*		V _{DD} =2.2V~5.5V, -40°C <Ta<85°C*
		15.20*	16*	16.80*		-20°C <Ta<70°C*
f _{ILRC}	ILRC 频率 *	31.3*	37*	41.9*	kHz	V _{DD} =5.0V, Ta=25°C
		24.0*	37*	50.0*		V _{DD} =5.0V, -40°C <Ta<85°C*
		25.9*	37*	48.1*		V _{DD} =5.0V, -20°C <Ta<70°C*
		18.3*	21*	24.5*		V _{DD} =3.3V, Ta=25°C
		14.0*	21*	29.0*		V _{DD} =3.3V, -40°C <Ta<85°C*
		14.7*	21*	27.3*		V _{DD} =3.3V, -20°C <Ta<70°C*
t _{INT}	中断脉冲宽度	30			ns	V _{DD} = 5.0V
V _{ADC}	ADC 的可工作电压	2.5		5.0	V	
V _{AD}	AD 输入电压	0		V _{DD}	V	
ADrs	ADC 分辨率			12	bit	
ADcs	ADC 消耗电流		0.9 0.8		mA	@5V @3V
ADclk	ADC 工作时钟周期		2		us	2.5V ~ 5.5V
t _{ADCONV}	ADC 转换时间 (T _{ADCLK} 是选定 AD 转换时钟周期)		13 14 15 16 17		T _{ADCLK}	8 位分辨率 9 位分辨率 10 位分辨率 11 位分辨率 12 位分辨率
AD DNL	AD 微分非线性		±2*		LSB	
AD INL	AD 积分非线性		±4*		LSB	
ADos	AD 失调电压 (offset)		3		mV	@V _{DD} =3V
V _{REFH}	ADC 参考高电压					@V _{DD} =5V
	4V	3.90	4	4.10		
	3V	2.93	3	3.07		
	2V	1.95	2	2.05		
V _{DR}	数据存储器数据维持电压*	1.5			V	在待机模式下
t _{WDT}	看门狗计数器超时溢出时间		2048		T _{ILRC}	misc[1:0]=00 (默认)
			4096			misc[1:0]=01
			16384			misc[1:0]=10
			256			misc[1:0]=11

符号	特性	最小值	典型值	最大值	单位	条件 (Ta=25°C)
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。系统时钟为 IHRC。		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。系统时钟为 ILRC。		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 _{SBP}	系统开机时间 (从开启电源算起)		1024		T _{ILRC}	T _{ILRC} 是 ILRC 时钟周期
t _{RST}	外部复位脉冲宽度	120			us	@V _{DD} =5V

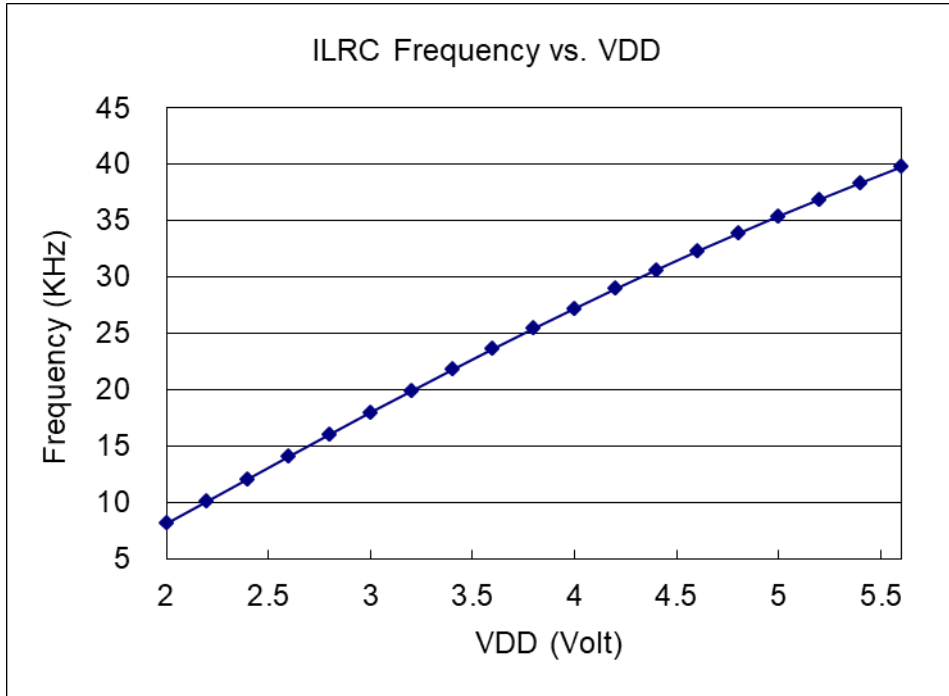
*这些参数是设计参考值, 并不是每个芯片测试。

** Under_20ms_V_{DD}_Ok 为对 V_{DD} 能否于 20ms 内从 0V 上升到指定电压的一个检查条件。

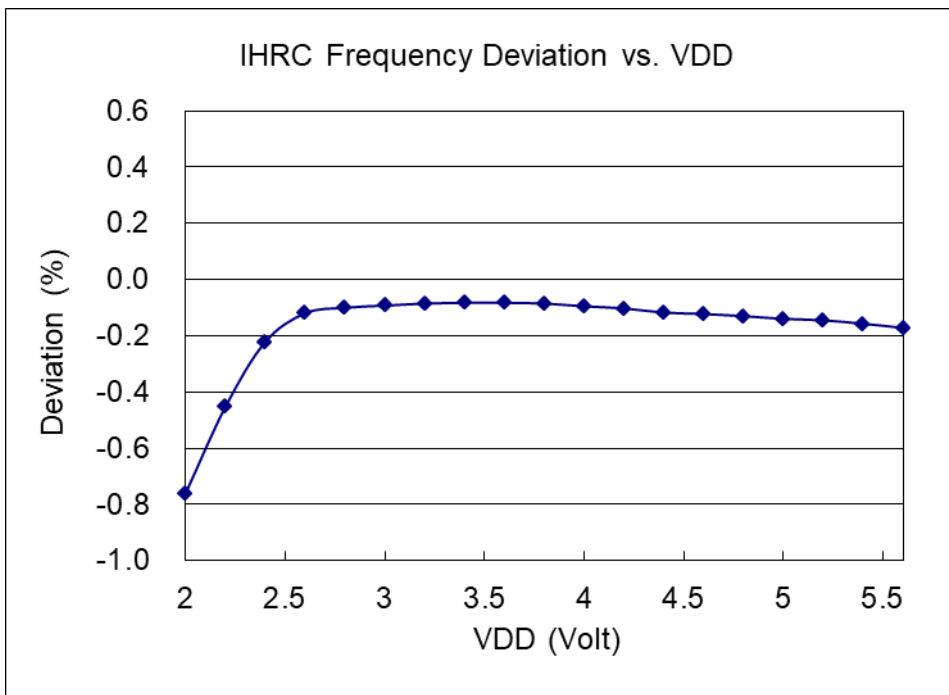
4.2. 最大范围

- 电源电压 2.2V ~ 5.5V
- 输入电压..... -0.3V ~ V_{DD} + 0.3V
- 工作温度 PMC131 系列: -40°C ~ 85°C
PMS131, PMS130 系列: -20°C ~ 70°C
- 节点温度..... 150°C
- 储藏温度 -50°C ~ 125°C

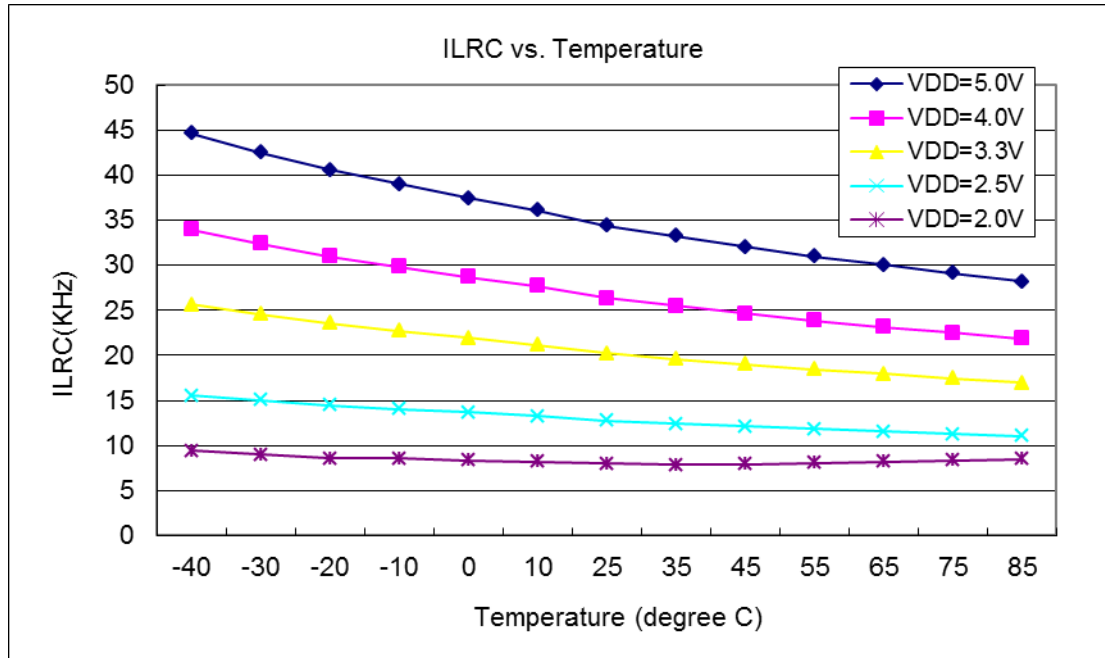
4.3. ILRC 频率与 VDD 关系的曲线图



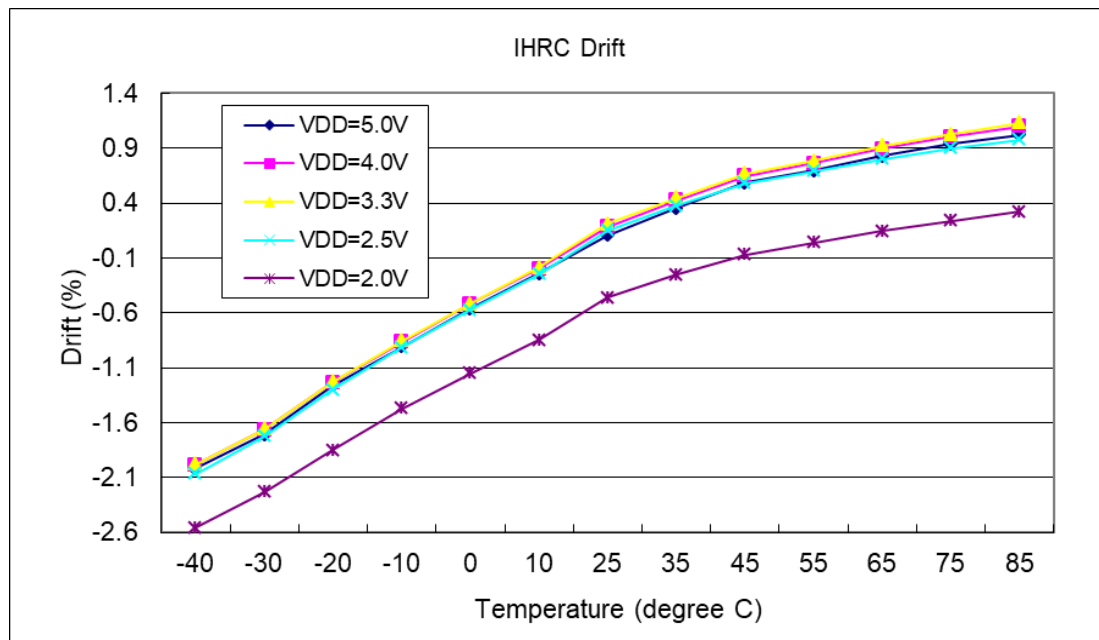
4.4. IHRC 频率与 VDD 关系的曲线图（校准至 16MHz）



4.5. ILRC 频率与温度关系曲线图



4.6. IHRC 频率与温度关系曲线图（校准至 16MHz）



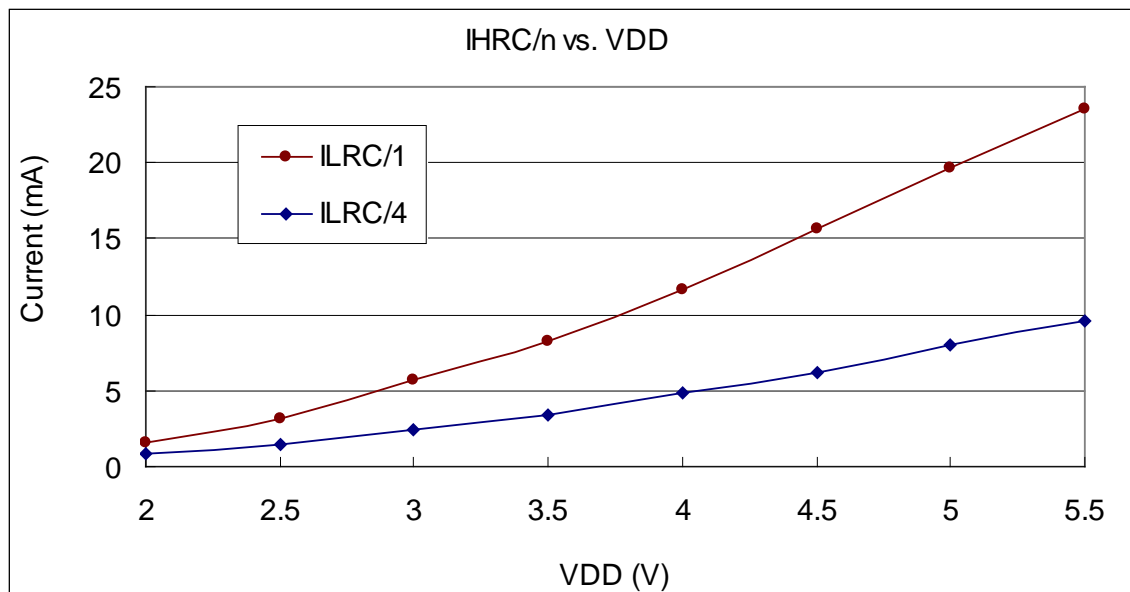
4.7. 工作电流量测值 @系统时钟= ILRC÷n

量测条件:

启用: ILRC;

禁用: Band-gap, LVR, IHRC, EOSC, T16, TM2, TM3, ADC 等模块;

IO 引脚: PA0:0.5Hz 输出切换而且没负载, 其它脚位: 输入而且不浮接。



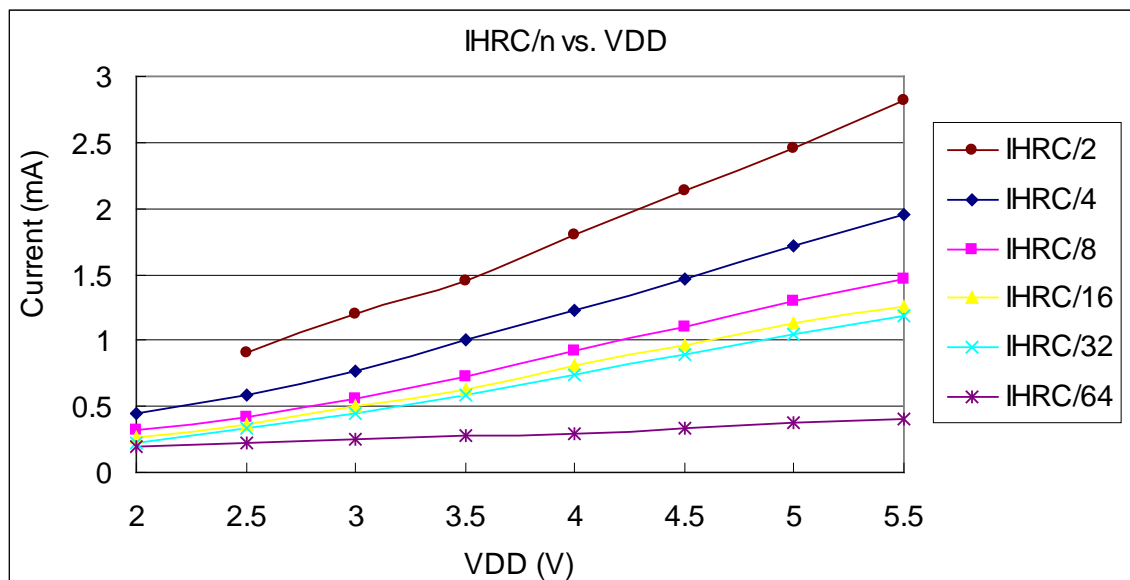
4.8. 工作电流量测值 @系统时钟= IHRC÷n

量测条件:

启用: Band-gap, LVR, IHRC;

禁用: ILRC, EOSC, T16, TM2, TM3, ADC 等模块;

IO 引脚: PA0:0.5Hz 输出切换而且没负载, 其它脚位: 输入而且不浮接。



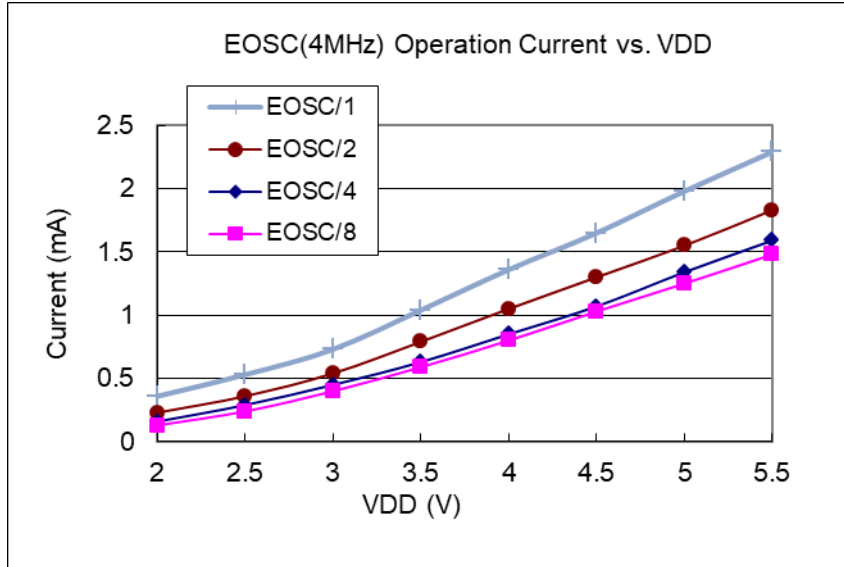
4.9. 工作电流量测值 @系统时钟= 4MHz 晶振 EOSC÷n

量测条件:

启用: EOSC, MISC.6 = 1;

禁用: Band-gap, LVR, IHRC, ILRC, T16, TM2, TM3, ADC,等模块;

IO 引脚: PA0:0.5Hz 输出切换而且没负载, 其它脚位: 输入而且不浮接。



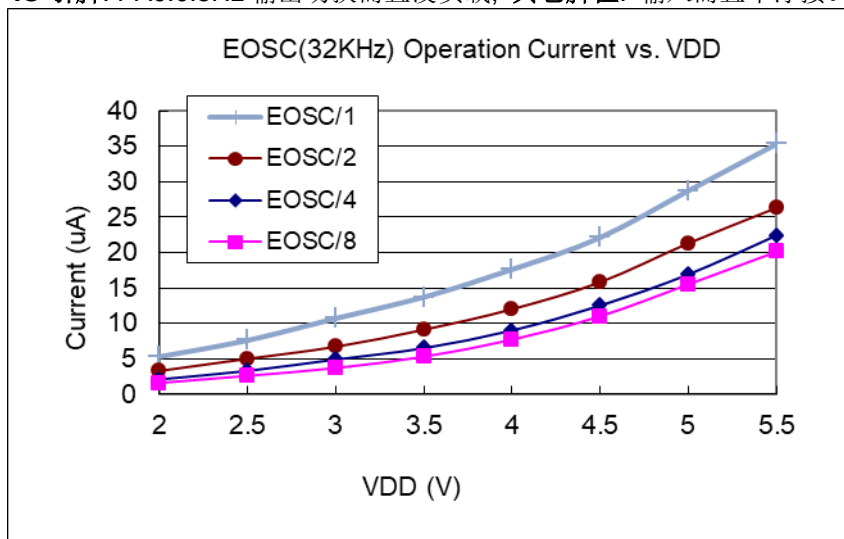
4.10. 工作电流量测值 @系统时钟= 32kHz 晶振 EOSC÷n

量测条件:

启用: EOSC, MISC.6 = 1;

禁用: Band-gap, LVR, IHRC, ILRC, T16, TM2, TM3, ADC 等模块;

IO 引脚: PA0:0.5Hz 输出切换而且没负载, 其它脚位: 输入而且不浮接。



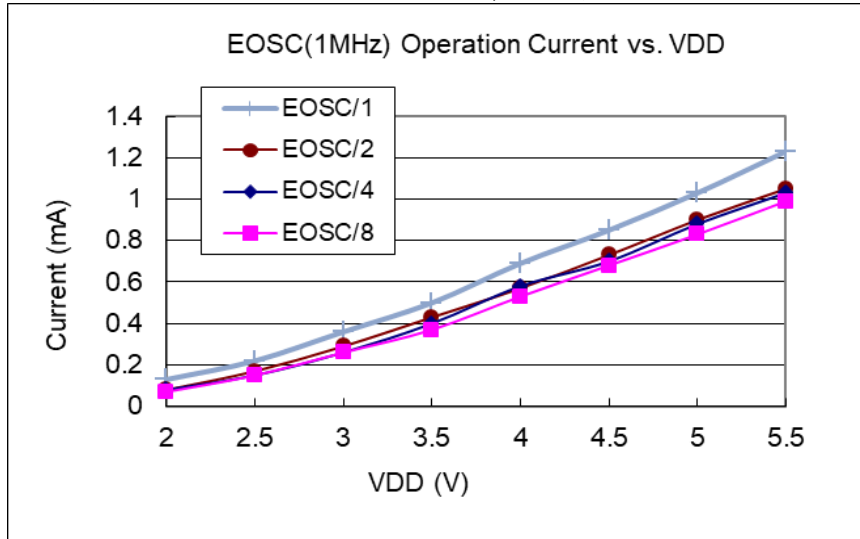
4.11. 工作电流量测值 @系统时钟= 1MHz 晶振 EOSC=n

量测条件:

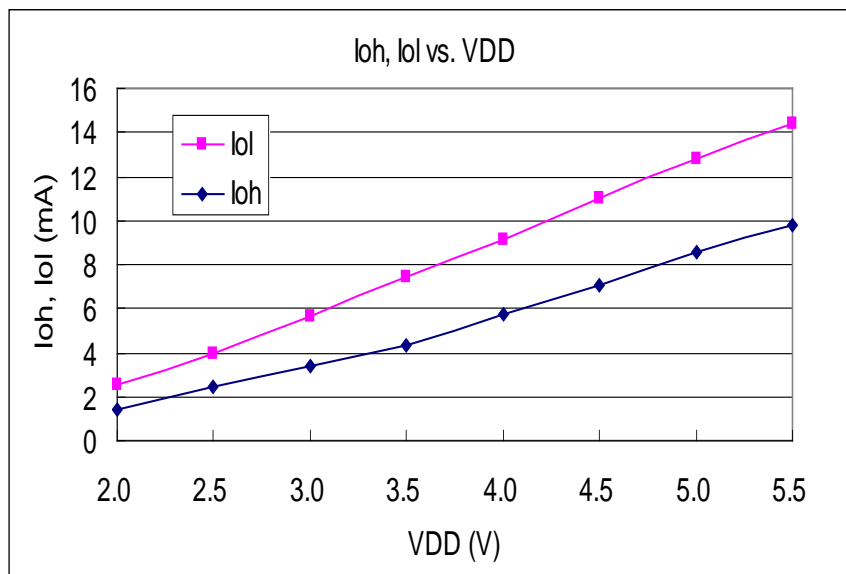
启用: EOSC, MISC.6 = 1;

禁用: Band-gap, LVR, IHRC, ILRC, T16, TM2, TM3, ADC 等模块;

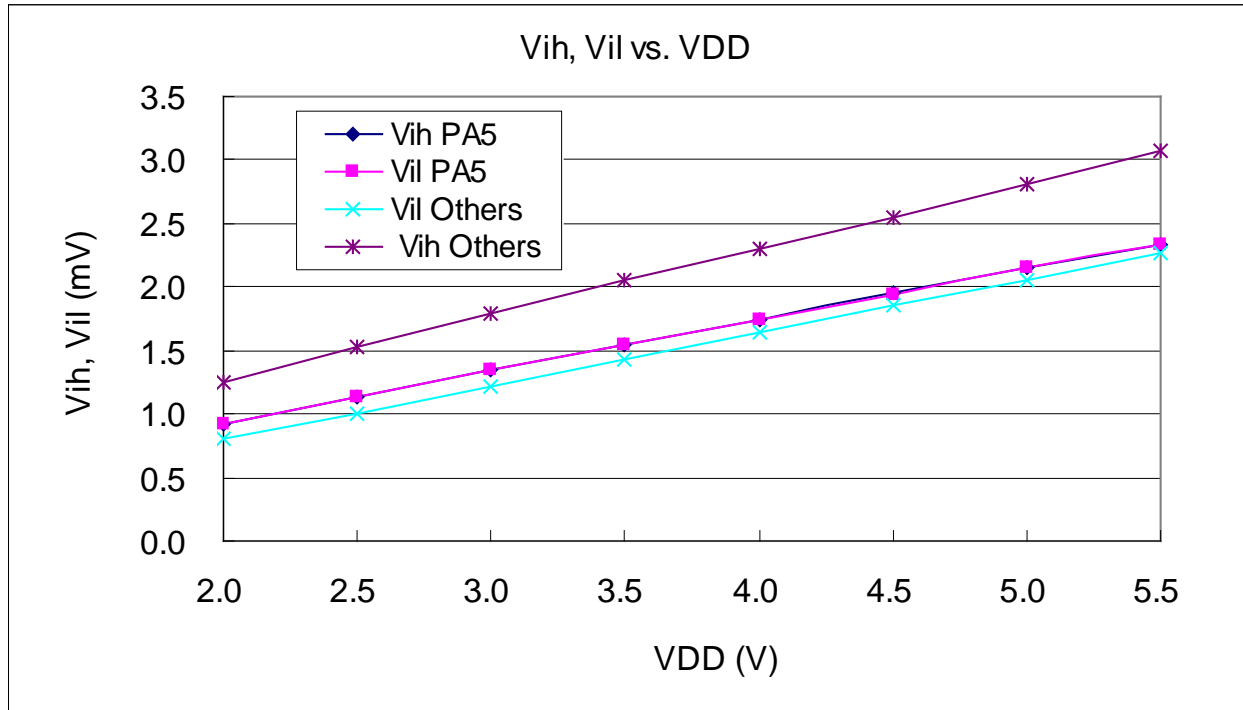
IO 引脚: PA0:0.5Hz 输出切换而且没负载, 其它脚位: 输入而且不浮接。



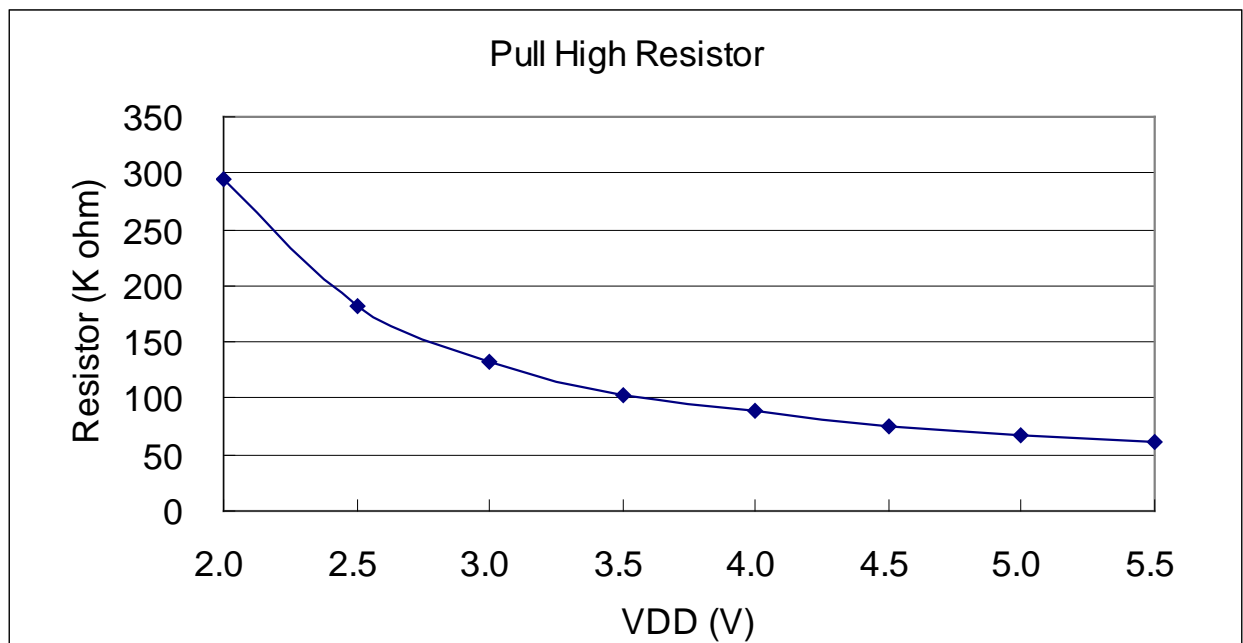
4.12. IO 引脚输出驱动电流(I_{OH})和灌电流(I_{OL})曲线图



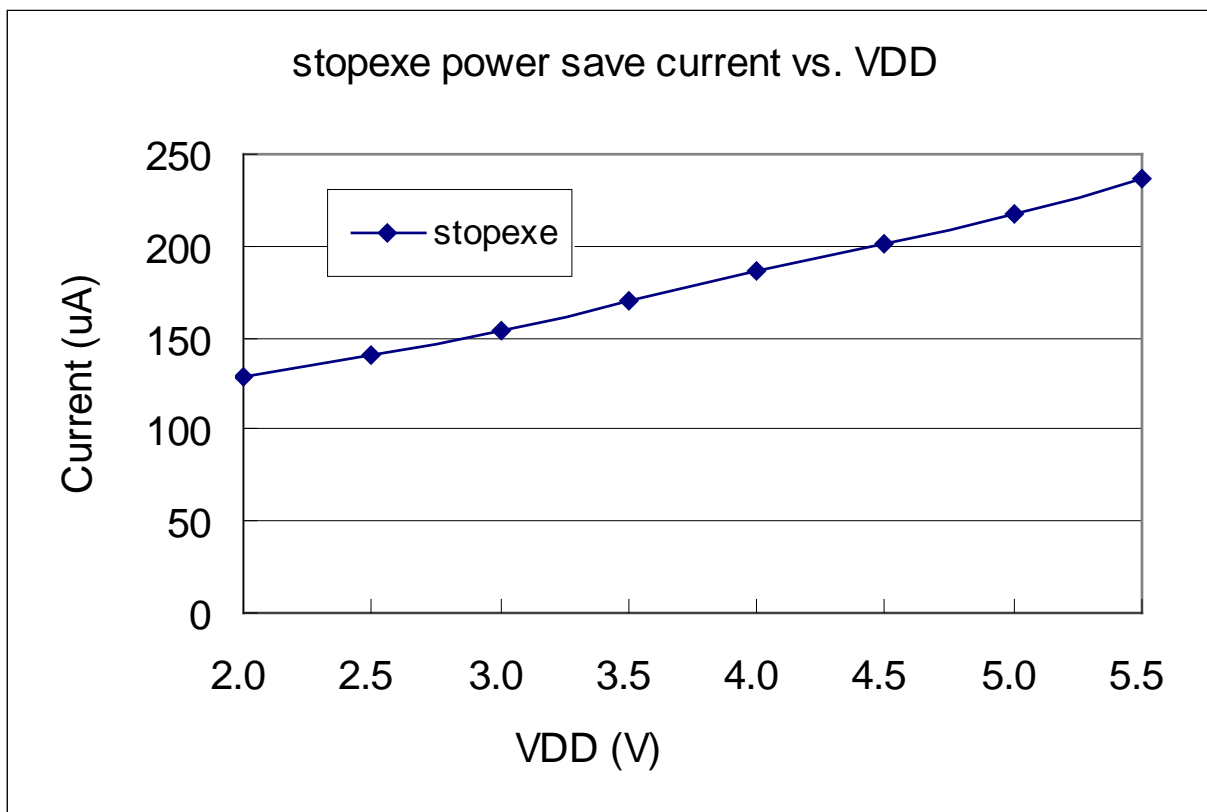
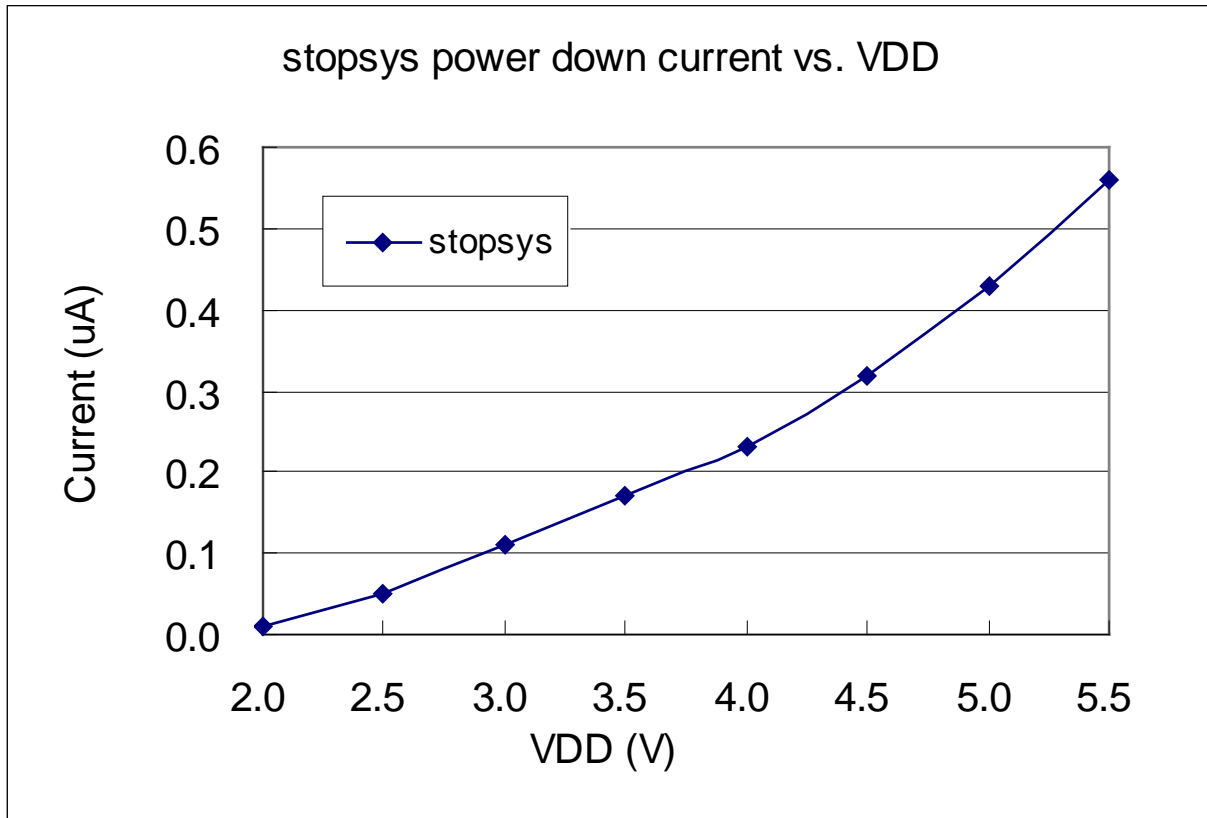
4.13. 测量的 IO 输入阈值电压(V_{IH}/V_{IL})



4.14. IO 引脚拉高阻抗曲线图



4.15. 省电模式和掉电模式消耗电流



5. 功能概述

5.1. OTP 程序存储器

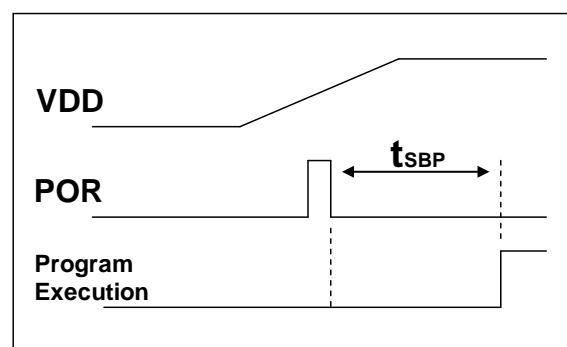
OTP（一次性可编程）程序存储器用来存放要执行的程序指令。OTP 程序存储器可以储存数据，包含：数据，表格和中断入口。复位之后，初始地址为 0x00，中断入口是 0x10；OTP 程序存储器最后 8 个地址空间是被保留给系统使用，如：校验，序列号等。PMC131/PMS131/PMS130 的 OTP 程序存储器结构是 1.5Kx14 位，如表 1 所示。OTP 存储器从地址“0x5F8 ~0x5FF”供系统使用，从“0x001~ 0x00F”和“0x011 ~0x5F7”地址空间是用户的程序空间。

地址	功能
0x000	起始地址 – goto 指令
0x001	用户程序区
0x002	用户程序区
•	•
0x00F	用户程序区
0x010	中断入口地址
0x011	用户程序区
•	•
0x5F7	用户程序区
0x5F8	系统使用
•	•
0x5FF	系统使用

表 1: 程序存储器结构

5.2. 启动程序

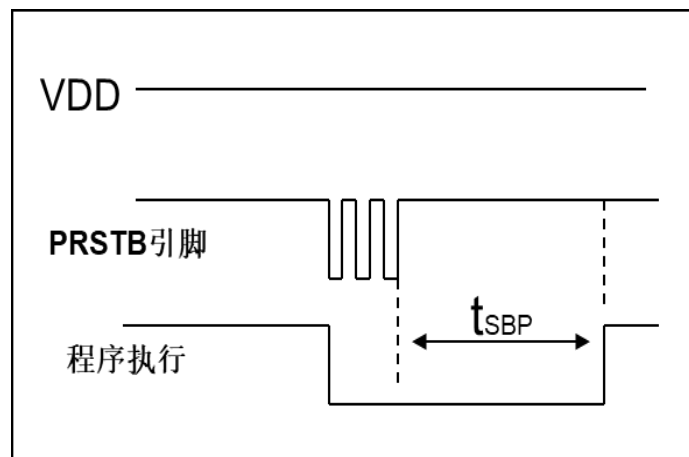
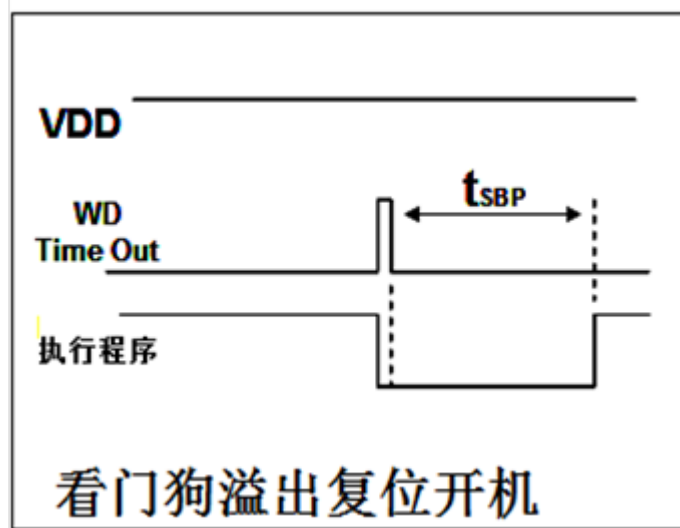
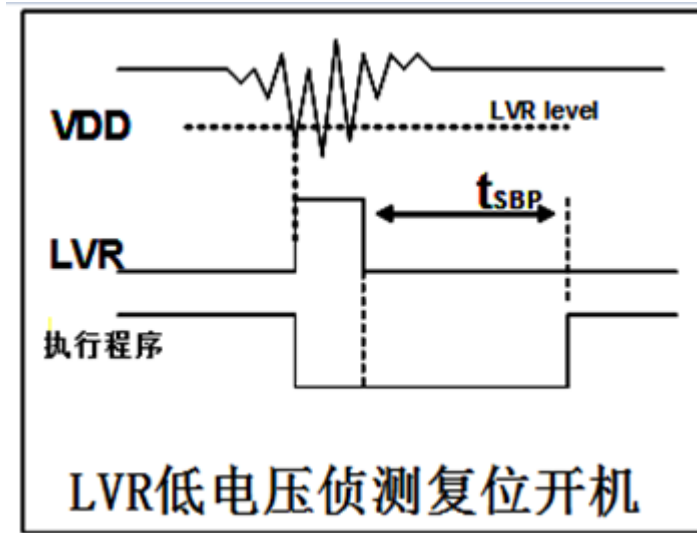
开机时，POR（上电复位）是用于复位 PMC131/PMS131/PMS130；但是，上电后电源电压可能不太稳定，为确保单片机是工作在电压稳定的状态，在执行第一条指令之前，PMC131/PMS131/PMS130 会延迟 1024 个 ILRC 时钟周期，这时间就是 t_{SBP} ，如图 1 所示。开机后，系统的预设时钟是 ILRC，使用者必需在启动后的时间里确保电源稳定。



Boot up from Power-On Reset

图 1: 上电复位时序

5.2.1.复位时序图



5.3. 数据存储

数据存取可以是字节或位的操作。除了存储资料外，SRAM 数据存储器还可以担任间接存取方式的资料指针，以及所有处理单元的堆栈存储器。

对于间接存取指令而言，数据存储器用作数据指针来当数据地址，所有的数据存储器都可以当做资料指针，这对于间接存取指令是相当灵活和有用的。由于数据宽度为 8 位，PMC131/PMS131/PMS130 内置的 88 个位组数据存储器都可以利用间接存取指令来存取。

5.4. 振荡器和时钟

PMC131/PMS131/PMS130 内置 3 个振荡器电路：外部晶体振荡器 (EOSC)、内部高频 RC 振荡器(IHRC) 和内部低频 RC 振荡器(ILRC)，这 3 个振荡器电路可以分别透过寄存器 `eoscr.7`、`clkmd.4` 以及 `clkmd.2` 来启用或禁用。使用者可以选择不同的振荡器以及 `clkmd` 寄存器产生不同的系统频率，以满足不同的应用。

振荡器硬件模块	启用或禁用
EOSC	<code>eoscr.7</code>
IHRC	<code>clkmd.4</code>
ILRC	<code>clkmd.2</code>

表 2: PMC131/PMS131/PMS130 内置 3 个振荡器电路

5.4.1. 内部高频振荡器 (IHRC)和低频振荡器 (ILRC)

开机后，IHRC 是自动被启用的，IHRC 的频率是可以透过 `ihrcr` 寄存器校准，通常它被校准至 16MHz 以消除工艺生产所产生的变化，校准后的频率偏差，正常情况下可在 1% 以内。IHRC 的频率会因电源电压和工作温度而漂移，详细请参考 IHRC 频率与 VDD 及温度关系的曲线图。

ILRC 的频率会因工厂生产的过程会有所不同，使用时电源电压和温度的差异等因素，都可能影响频率漂移。请参考直流电气特性规格数据。

5.4.2. 单片机校准

在芯片生产制造时，每一颗的 IHRC 频率和 Band-gap 参考电位可能都有稍微的不同，PMC131/PMS131/PMS130 提供了 IHRC 频率校准以及 Band-gap 参考电压校准，以消除芯片生产制造时的漂移，校准功能选项是在用户程序编译时选择，IDE 软件在编译用户的程序时会自动插入用户程序，校准的命令如下：

`.ADJUST_IC SYSCLK=IHRC/(p1), IHRC=(p2)MHz, VDD=(p3)V, Band-gap=(p4);`

这里，**p1=2, 4, 8, 16, 32;** 提供系统时钟不同的频率。

p2=14 ~ 18; 提供芯片 IHRC 校准到不同的频率，通常选 16MHz。

p3=2.5 ~ 5.5; 提供芯片在不同的电压校准。

p4= On 或 Off; Band-gap 参考电压校准是 On 或 Off。

5.4.3. IHRC 频率校准和系统时钟

IHRC 频率校准选项是在用户程序编译时选择，IDE 软件在编译用户的程序时会自动插入用户程序，提供的选项是如表 3 所示：

SYSCLK	CLKMD	IHRCR	描述
<input type="radio"/> Set IHRC / 2	= 34h (IHRC / 2)	有校准	IHRC 校准到 16MHz, 系统时钟 CLK=8MHz (IHRC/2)
<input type="radio"/> Set IHRC / 4	= 14h (IHRC / 4)	有校准	IHRC 校准到 16MHz, 系统时钟 CLK=4MHz (IHRC/4)
<input type="radio"/> Set IHRC / 8	= 3Ch (IHRC / 8)	有校准	IHRC 校准到 16MHz, 系统时钟 CLK=2MHz (IHRC/8)
<input type="radio"/> Set IHRC / 16	= 1Ch (IHRC / 16)	有校准	IHRC 校准到 16MHz, 系统时钟 CLK=1MHz (IHRC/16)
<input type="radio"/> Set IHRC / 32	= 7Ch (IHRC / 32)	有校准	IHRC 校准到 16MHz, 系统时钟 CLK=0.5MHz (IHRC/32)
<input type="radio"/> Set ILRC	= E4h (ILRC / 1)	有校准	IHRC 校准到 16MHz, 系统时钟 CLK=ILRC
<input type="radio"/> Disable	没改变	没改变	IHRC 没有校准, 系统时钟 CLK 也没有改变, Band-gap 没有校准

表 3: IHRC 频率校准选项

通常，`.ADJUST_IC` 命令是摆在程序启动后的第 1 个动作，以便开机后能够设立所要的工作频率。程序代码在写入 OTP 的时候，IHRC 频率校准的程序会执行一次，以后，它就不会再被执行了。假如使用者选择不同的频率校准选项，PMC131/PMS131/PMS130 在开机后的状态也将不同，下面所示为不同选项在开机后，PMC131/PMS131/PMS130 执行此命令后的状态：

(1) `.ADJUST_IC SYSCLK=IHRC/2, IHRC=16MHz, VDD=5V, Band-gap=On`

开机后，CLKMD = 0x34:

- ◆ IHRC 频率在 V_{DD}=5V 下，校准到 16MHz 并且是启用的
- ◆ 系统时钟 CLK = IHRC/2 = 8MHz
- ◆ 看门狗计数器是停用，ILRC 是启用的，PA5 引脚设为输入，Band-gap 校准到 1.2V

(2) `.ADJUST_IC SYSCLK=IHRC/4, IHRC=16MHz, VDD=3.3V, Band-gap=On`

开机后，CLKMD = 0x14:

- ◆ IHRC 频率在 V_{DD}=3.3V 下，校准到 16MHz 并且是启用的
- ◆ 系统时钟 CLK = IHRC/4 = 4MHz
- ◆ 看门狗计数器是停用，ILRC 是启用的，PA5 引脚设为输入，Band-gap 校准到 1.2V

(3) .ADJUST_IC SYSCLK=IHRC/8, IHRC=16MHz, $V_{DD}=2.5V$, Band-gap=On

开机后, CLKMD = 0x3C:

- ◆ IHRC 频率在 $V_{DD}=2.5V$ 下, 校准到 16MHz 并且是启用的
- ◆ 系统时钟 CLK = IHRC/8 = 2MHz
- ◆ 看门狗计数器是停用, ILRC 是启用的, PA5 引脚设为输入, Band-gap 校准到 1.2V

(4) .ADJUST_IC SYSCLK=IHRC/16, IHRC=16MHz, $V_{DD}=2.5V$, Band-gap=On

开机后, CLKMD = 0x1C:

- ◆ IHRC 频率在 $V_{DD}=2.5V$ 下, 校准到 16MHz 并且是启用的
- ◆ 系统时钟 CLK = IHRC/16 = 1MHz
- ◆ 看门狗计数器是停用, ILRC 是启用的, PA5 引脚设为输入, Band-gap 校准到 1.2V

(5) .ADJUST_IC SYSCLK=IHRC/32, IHRC=16MHz, $V_{DD}=5V$, Band-gap=Off

开机后, CLKMD = 0x7C:

- ◆ IHRC 频率在 $V_{DD}=5V$ 下, 校准到 16MHz 并且是启用的
- ◆ 系统时钟 CLK = IHRC/32 = 500kHz
- ◆ 看门狗计数器是停用, ILRC 是启用的, A5 引脚设为输入, Band-gap 没校准

(6) .ADJUST_IC SYSCLK=ILRC, IHRC=16MHz, $V_{DD}=5V$, Band-gap=Off

开机后, CLKMD = 0xE4:

- ◆ IHRC 频率在 $V_{DD}=5V$ 下, 校准到 16MHz 并且是禁用的
- ◆ 系统时钟 CLK = ILRC
- ◆ 看门狗计数器是停用, ILRC 是启用的, PA5 引脚设为输入

(7) .ADJUST_IC DISABLE

开机后, CLKMD 寄存器没被改变 (没有任何动作):

- ◆ IHRC 频率没有校准并且是停用的, Band-gap 没有校准
- ◆ 系统时钟 CLK = ILRC 或 IHRC/64
- ◆ 看门狗计数器是启用, ILRC 是启用的, PA5 引脚设为输入

5.4.4. 外部晶体振荡器

如果使用晶体振荡器, X1 和 X2 之间需要晶体或谐振器。其应用线路如图 2 所示; 晶体振荡器的工作频率可以从 32kHz 到 4MHz, 超过 4MHz 是不支持的。

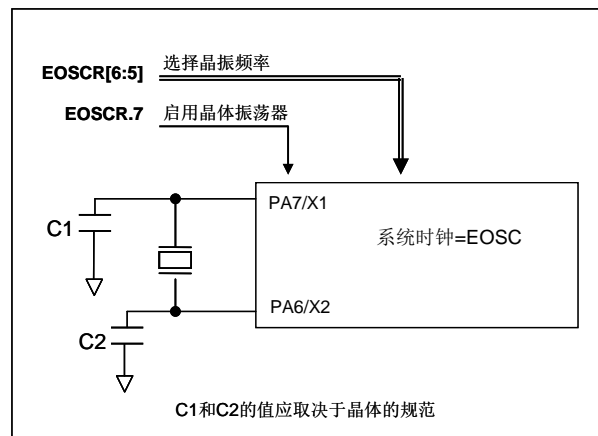


图 2: 晶体振荡器使用接法

寄存器 **eoscr** (0x0b) 位 7 是用来启用晶体振荡器，另外，寄存器 **eoscr** (0x0b) 位 6-5 提供不同的驱动电流能力，以配合不同的晶体振荡器频率：

- ◆ **eoscr**[6:5]=01：低驱动电流，适用于较低频率，例如：32kHz 晶体振荡器
- ◆ **eoscr**[6:5]=10：中驱动电流，适用于中间频率，例如：1MHz 晶体振荡器
- ◆ **eoscr**[6:5]=11：高驱动电流，适用于较高频率，例如：4MHz 晶体振荡器

为了得到良好的正弦波形，外部电容 **C1** 和 **C2** 也需调整，表 4 显示不同的晶体或谐振器，**C1** 和 **C2** 的建议值以及在对对应条件下所测量到的起振时间。因为晶体或谐振器都有其不同的特性，所需要的 **C1**、**C2** 以及起振时间也会因不同的晶体或谐振器而有些差异，使用时请参考晶体或谐振器规格并选择合适的 **C1** 和 **C2**。

频率	C1	C2	测量起振时间	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 4: 不同的晶体或谐振器所需 **C1** 和 **C2** 的建议值

另外，使用晶体振荡器要特别注意启用之后所需要的稳定时间，它会依频率、晶体或谐振器型号、外部电容、工作电压而不同，在将系统时钟源切换成晶体振荡器之前，必需确保晶体振荡器已经稳定，参考程序如下：

```

void FPPA0 (void)
{
    . ADJUST_IC  SYSCLK=IHRC/16, IHRC=16MHz, VDD=5V, Band-gap=On
    // 如果 Band-gap 不需要校准，可以写成 “. ADJUST_IC DISABLE” ...
    ...
    $ EOSENable, 4MHz; // EOSCR = 0b110_00000;
    $ T16M  EOSC, /1, BIT13; // T16 收到 2^14=16384 个晶体振荡器时钟,
        // Intrq.T16 =>1, 晶体振荡器已经稳定

    WORD count = 0;
    stt16  count;
    Intrq.T16 = 0;
    do
    {
        nop;
    }while(!Intrq.T16); // 计数从 0x0000 到 0x2000,然后设置 INTRQ.T16
    clkmd = 0xB4; // 将系统时钟切换成 EOSC;
    clkmd.4=0; //关闭 IHRC
    ...
}

```

进入掉电模式之前，请先将晶体振荡器关闭以避免不可预期的唤醒发生；假如使用 32kHz 晶体振荡器而且又需要非常的省电，当晶体振荡器稳定后，设置 **misc**.6 为 1 以降低电流。

5.4.5. 系统时钟和 LVR 水平

系统时钟可以来自 EOSC，IHRC 和 ILRC，图 3 显示为 PMC131/PMS131/PMS130 中的系统时钟选项的硬件框图。

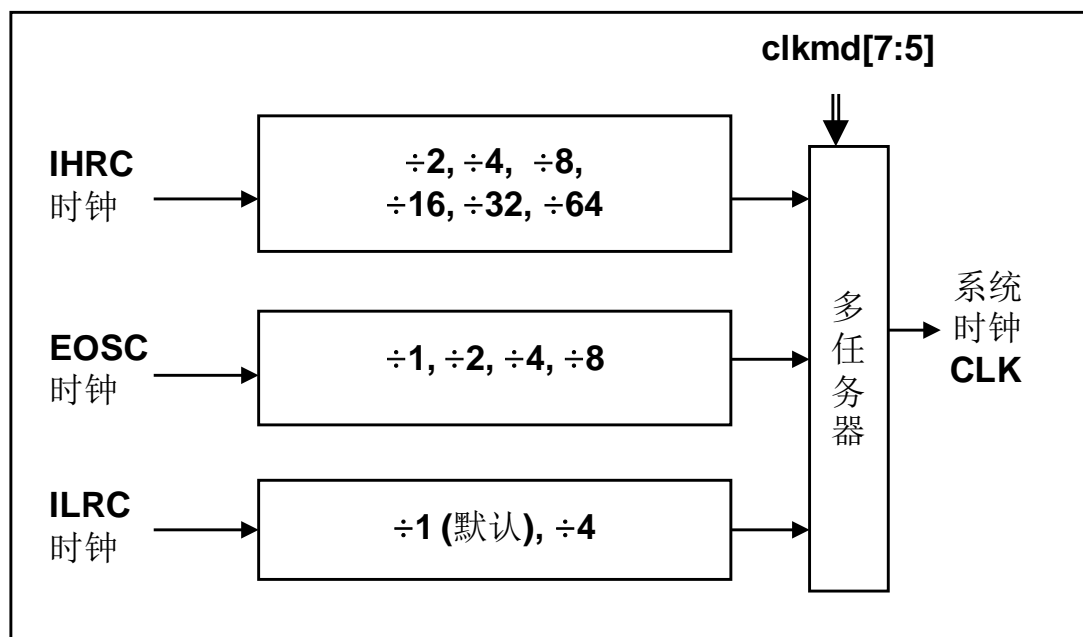


图 3: 系统时钟选项

使用者可以依照不同的需求选择不同的工作系统时钟，选定的工作系统时钟应与电源电压和 LVR 水平结合起来，才能使系统稳定运作。LVR 的水平是在编译过程中选择，请参考第 4.1 节直流交流电气特性的工作频率和 LVR 水平设定建议。

5.4.6. 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，PMC131/PMS131/PMS130 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* 切换频率时，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
...
```


5.5. 16 位计数器 (Timer16)

PMC131/PMS131/PMS130 内含一个 16 位计数器，计数器时钟可来自于系统时钟 (CLK)、外部晶体振荡器时钟、内部高频振荡时钟 (IHRC)、内部低频振荡时钟 (ILRC)，或 PA0、PA4，1 个多任务器用于选择时钟输出的时钟来源，在送到 16 位计数器之前，1 个可软件编程的预分频器提供 $\div 1$ 、 $\div 4$ 、 $\div 16$ 、 $\div 64$ 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。Timer16 模块框图如图 4。中断源是来自 16 位计数器的位 8 到位 15，中断类型可以上升沿触发或下降沿触发，定义在 **intgs.4** 寄存器位 4 (IO 地址 0x0C)。

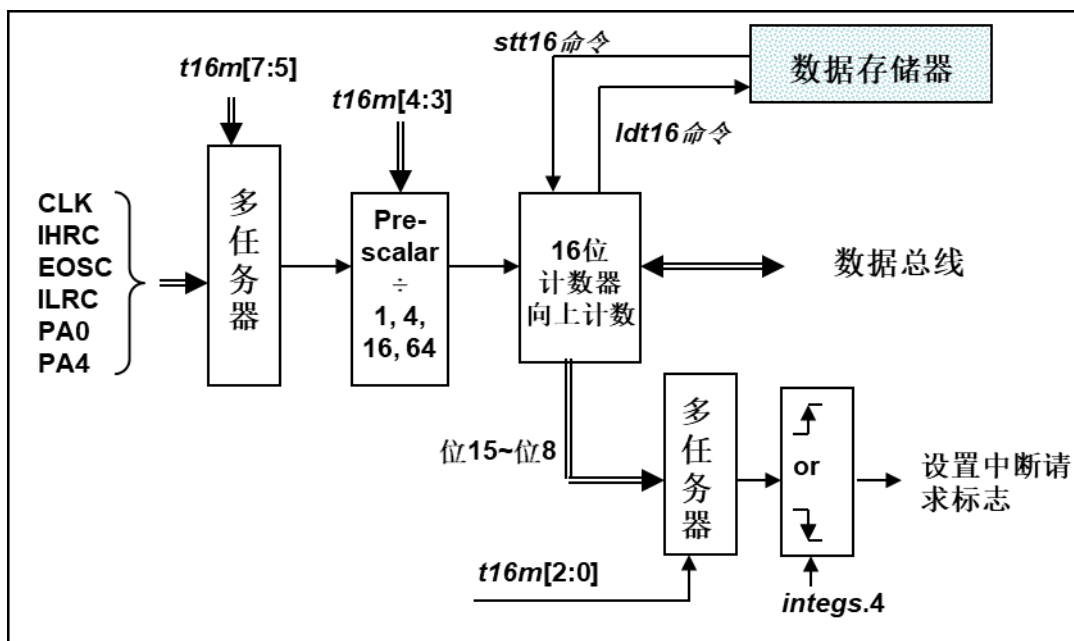


图 4: Timer16 模块框图

当使用 Timer16 时，Timer16 的使用语法已定义在 .INC 文件中。有三个参数来定义 Timer16 的使用；第一参数是用来定义 Timer16 时钟源，第二参数是用来定义预分频器，最后一个定义中断源。详细如下：

```

T16M IO_RW 0x06
$ 7~5: STOP, SYSCLK, X, PA4_F, IHRC, EOSC, ILRC, PA0_F // 第一参数
$ 4~3: /1, /4, /16, /64 // 第二参数
$ 2~0: BIT8, BIT9, BIT10, BIT11, BIT12, BIT13, BIT14, BIT15 // 第三参数
    
```

使用者可以依照系统的要求来定义 T16M 参数，例子如下，更多例子请参阅 IDE 软件中的“求助”→“使用手册”→“IC 介绍”→“缓存器介绍”→“T16M”：

\$ 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
// 假如系统时钟 System Clock = IHRC / 2 = 8 MHz
// $\text{SYSCLK}/64 = 8 \text{ MHz}/64 = 125\text{kHz}$, 约每 512 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 $\text{EOSC}=32768 \text{ Hz}$, $32768 \text{ Hz}/(2^{14}) = 2\text{Hz}$, 约每 0.5S 产生一次 INTRQ.2=1

\$ T16M PA0_F, /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:

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

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.

5.6. 8 位 PWM 计数器(Timer2/Timer3)

PMC131/PMS131/PMS130 内置两个 8 位 PWM 硬件计数器，硬件框图请参考图 5，以下叙述只提 Timer2 (Timer 3 和 Timer 2 是一样的)。图 4 显示出 Timer2 的模块框图。计数器的时钟源可能来自系统时钟(CLK)，内部高频 RC 振荡器时钟(IHRC)，内部低频 RC 振荡器时钟(ILRC)，PA0, PB0, PA4, 寄存器 *tm2c* 的位[7:4] 用来选择计数器时钟。请注意，外部晶体振荡器是不能当做 Timer2 的时钟，因为它可能有突波。另外，在执行仿真器(ICE)时，若内部高频 RC 振荡器时钟(IHRC)被选择当做 Timer2 的时钟，当仿真器停住时，IHRC 时钟仍继续送到 Timer2，所以 Timer2 在仿真器停住时仍然会继续计数。依据寄存器 *tm2c* 位 [3:2] 的设定，Timer2 的输出可以是 PB2, PA3 或 PB4。利用软件编程寄存器 *tm2s* 位[6:5]，时钟预分频器的模块提供了 $\div 1$, $\div 4$, $\div 16$ 和 $\div 64$ 的选择，由寄存器 *tm2s* 的位[6:5] 控制。另外，利用软件编程寄存器 *tm2s* 位[4:0]，时钟分频器的模块提供了 $\div 1 \sim \div 31$ 的功能。在结合预分频器以及分频器，Timer2 时钟 (TM2_CLK) 频率可以广泛和灵活，以提供不同产品应用。

8 位 PWM 计数器只能执行 8 位上升计数操作，经由寄存器 *tm2ct*，计数器的值可以设置或读取。当 8 位计数器计数值达到上限寄存器设定的范围时，计数器将自动清除为零，上限寄存器用来定义计数器产生波形的周期或 PWM 占空比。8 位 PWM 计数器有两个工作模式：周期模式和 PWM 模式；周期模式用于输出固定周期波形或中断事件；PWM 模式是用来产生 PWM 输出波形，PWM 分辨率可以为 6 位或 8 位。图 6 显示出 Timer2 周期模式和 PWM 模式的时序图。

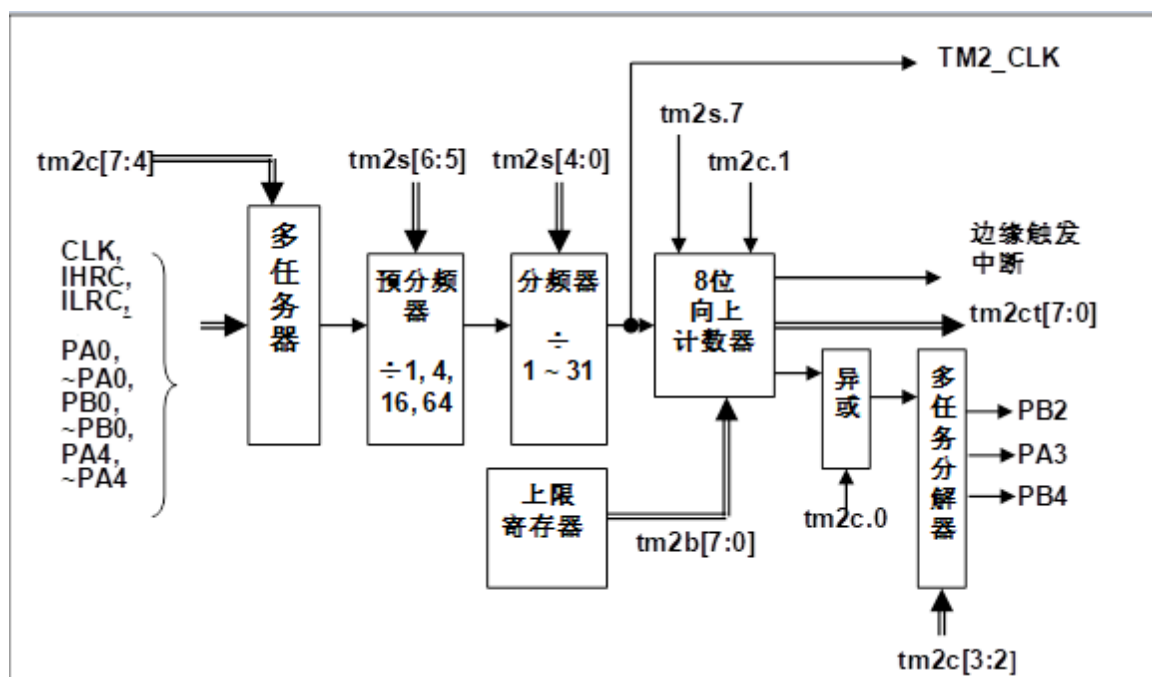


图 5: Timer2 模块框图

Timer3 的输出可以是 PB5, PB6 或 PB7。

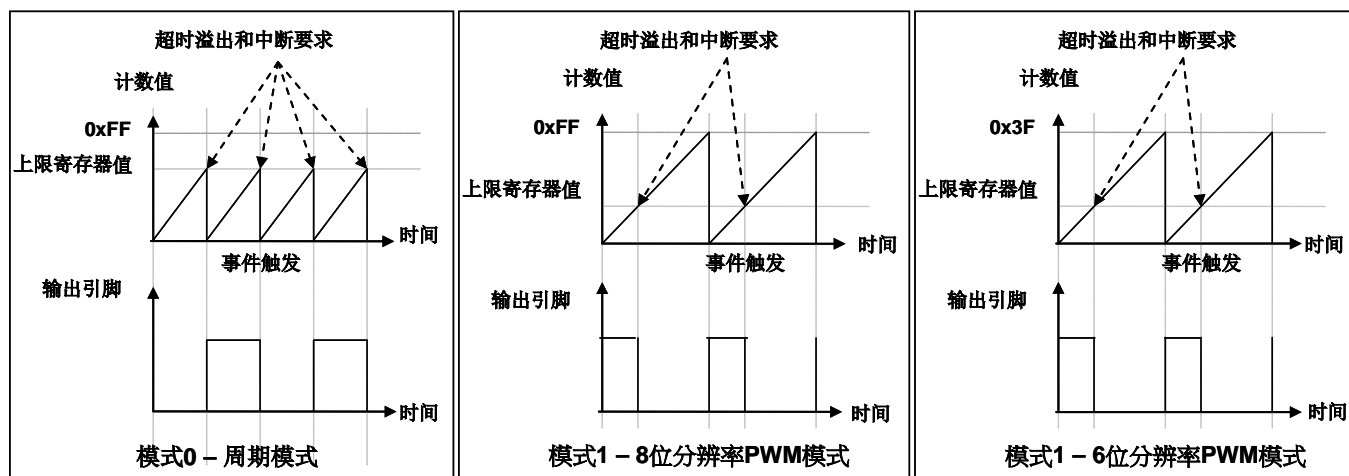


图 6: Timer2 周期模式和 PWM 模式的时序图 (tm2c.1=1)

5.6.1. 使用 Timer2 产生定期波形

如果选择周期模式的输出，输出波形的占空比总是 50%，其输出频率与寄存器设定，可以概括如下：

$$\text{输出信号频率} = Y \div [2 \times (K+1) \times S1 \times (S2+1)]$$

这里，

Y = tm2c[7:4] : Timer2 所选择的时钟源频率

K = tm2b[7:0] : 上限寄存器设定的值 (十进制)

S1 = tm2s[6:5] : 预分频器设定值 (S1= 1, 4, 16, 64)

S2 = tm2s[4:0] : 分频器值 (十进制, S2= 0 ~ 31)

例 1:

tm2c = 0b0001_1000, Y=8MHz

tm2b = 0b0111_1111, K=127

tm2s = 0b0_00_00000, S1=1, S2=0

→ 输出信号频率 = 8MHz ÷ [2 × (127+1) × 1 × (0+1)] = 31.25kHz

例 2:

tm2c = 0b0001_1000, Y=8MHz

tm2b = 0b0111_1111, K=127

tm2s[7:0] = 0b0_11_11111, S1=64, S2 = 31

→ 输出信号频率 = 8MHz ÷ (2 × (127+1) × 64 × (31+1)) = 15.25Hz

例 3:

tm2c = 0b0001_1000, Y=8MHz

tm2b = 0b0000_1111, K=15

tm2s = 0b0_00_00000, S1=1, S2=0

→ 输出信号频率 = 8MHz ÷ (2 × (15+1) × 1 × (0+1)) = 250kHz

例 4:

```
tm2c = 0b0001_1000, Y=8MHz
tm2b = 0b0000_0001, K=1
tm2s = 0b0_00_00000, S1=1, S2=0
➔ 输出信号频率 = 8MHz ÷ ( 2 × (1+1) × 1 × (0+1) ) =2MHz
```

使用 Timer2 计数器产生定期波形的示例程序如下所示:

```
void FPPA0(void)
{
    . ADJUST_IC SYSCLK=IHRC/2, IHRC=16MHz, VDD=5V
    ...
    tm2ct = 0x00;
    tm2b = 0x7f;
    tm2s = 0b0_00_00001; // 8 位 pwm, 预分频 = 1, 分频 = 2
    tm2c = 0b0001_10_0_0; // 系统时钟, 输出 =PA3, 周期模式
    while(1)
    {
        nop;
    }
}
```

5.6.2. 使用 Timer2 产生 8 位 PWM 波形

如果选择 8 位 PWM 的模式, 应设立 $tm2c[1] = 1$, $tm2s[7] = 0$, 输出波形的频率和占空比可以概括如下:

$$\text{输出频率} = Y \div [256 \times S1 \times (S2+1)]$$

$$\text{输出空占比} = [(K+1) \div 256] \times 100\%$$

这里,

Y = $tm2c[7:4]$: Timer2 所选择的时钟源频率
 K = $tm2b[7:0]$: 上限寄存器设定的值 (十进制)
 S1 = $tm2s[6:5]$: 预分频器设定值 (S1= 1, 4, 16, 64)
 S2 = $tm2s[4:0]$: 分频器值 (十进制, S2= 0 ~ 31)

例 1:

```
tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s = 0b0_00_00000, S1=1, S2=0
➔ 输出频率 = 8MHz ÷ ( 256 × 1 × (0+1) ) = 31.25kHz
➔ 输出空占比 = [(127+1) ÷ 256] × 100% = 50%
```

例 2:

tm2c = 0b0001_1010, Y=8MHz

tm2b = 0b0111_1111, K=127

tm2s = 0b0_11_11111, S1=64, S2=31

→ 输出频率 = $8\text{MHz} \div (256 \times 64 \times (31+1)) = 15.25\text{Hz}$

→ 输出空占比 = $[(127+1) \div 256] \times 100\% = 50\%$

例 3:

tm2c = 0b0001_1010, Y=8MHz

tm2b = 0b1111_1111, K=255

tm2s = 0b0_00_00000, S1=1, S2=0

→ 输出频率 = $8\text{MHz} \div (256 \times 1 \times (0+1)) = 31.25\text{KHz}$

→ 输出空占比 = $[(255+1) \div 256] \times 100\% = 100\%$

例 4:

tm2c = 0b0001_1010, Y=8MHz

tm2b = 0b0000_1001, K = 9

tm2s = 0b0_00_00000, S1=1, S2=0

→ 输出频率 = $8\text{MHz} \div (256 \times 1 \times (0+1)) = 31.25\text{kHz}$

→ 输出空占比 = $[(9+1) \div 256] \times 100\% = 3.9\%$

使用 Timer2 计数器从 PA2 产生 PWM 波形的示例程序如下所示:

```
void FPPA0 (void)
{
    .ADJUST_IC SYSCLK=IHRC/2, IHRC=16MHz, VDD=5V
    wdreset;
    tm2ct = 0x00;
    tm2b = 0x7f;
    tm2s = 0b0_00_00001; // 8 位 pwm, 预分频 = 1, 分频 = 2
    tm2c = 0b0001_10_1_0; // 系统时钟, 输出 = PA3, PWM 模式
    while(1)
    {
        nop;
    }
}
```

5.6.3. 使用 Timer2 产生 6 位 PWM 波形

如果选择 6 位 PWM 的模式，应设立 $tm2c[1] = 1$ ， $tm2s[7] = 1$ ，输出波形的频率和占空比可以概括如下：

$$\text{输出频率} = Y \div [64 \times S1 \times (S2+1)]$$

$$\text{输出空占比} = [(K+1) \div 64] \times 100\%$$

这里，

$Y = tm2c[7:4]$: Timer2 所选择的时钟源频率

$K = tm2b[7:0]$: 上限寄存器设定的值（十进制）

$S1 = tm2s[6:5]$: 预分频器设定值 ($S1 = 1, 4, 16, 64$)

$S2 = tm2s[4:0]$: 分频器值（十进制， $S2 = 0 \sim 31$ ）

例 1:

$tm2c = 0b0001_1010$, $Y=8MHz$

$tm2b = 0b0001_1111$, $K=31$

$tm2s = 0b1_00_00000$, $S1=1$, $S2=0$

→ 输出频率 = $8MHz \div (64 \times 1 \times (0+1)) = 125kHz$

→ 输出空占比 = $[(31+1) \div 64] \times 100\% = 50\%$

例 2:

$tm2c = 0b0001_1010$, $Y=8MHz$

$tm2b = 0b0001_1111$, $K=31$

$tm2s = 0b1_11_11111$, $S1=64$, $S2=31$

→ 输出频率 = $8MHz \div (64 \times 64 \times (31+1)) = 61.03 Hz$

→ 输出空占比 = $[(31+1) \div 64] \times 100\% = 50\%$

例 3:

$tm2c = 0b0001_1010$, $Y=8MHz$

$tm2b = 0b0011_1111$, $K=63$

$tm2s = 0b1_00_00000$, $S1=1$, $S2=0$

→ 输出频率 = $8MHz \div (64 \times 1 \times (0+1)) = 125KHz$

→ 输出空占比 = $[(63+1) \div 64] \times 100\% = 100\%$

例 4:

$tm2c = 0b0001_1010$, $Y=8MHz$

$tm2b = 0b0000_0000$, $K=0$

$tm2s = 0b1_00_00000$, $S1=1$, $S2=0$

→ 输出频率 = $8MHz \div (64 \times 1 \times (0+1)) = 125kHz$

→ 输出空占比 = $[(0+1) \div 64] \times 100\% = 1.5\%$

5.7. 看门狗计数器

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

- ◆ 当 *misc*[1:0]=11 时：256 个 ILRC 时钟周期
- ◆ 当 *misc*[1:0]=00 时（默认）：2048 ILRC 时钟周期
- ◆ 当 *misc*[1:0]=01 时：4096 ILRC 时钟周期
- ◆ 当 *misc*[1:0]=10 时：16384 ILRC 时钟周期

ILRC 的频率有可能因为工厂制造的变化，电源电压和工作温度而漂移很多；使用者必须预留安全操作范围。为确保看门狗计数器在超时溢出周期之前被清零，在安全时间内，用指令“*wdreset*”清零看门狗计数器。在上电复位或任何时候使用 *wdreset* 指令，看门狗计数器都会被清零。当看门狗计数器超时溢出时，PMC131/PMS131/PMS130 将复位并重新运行程序。

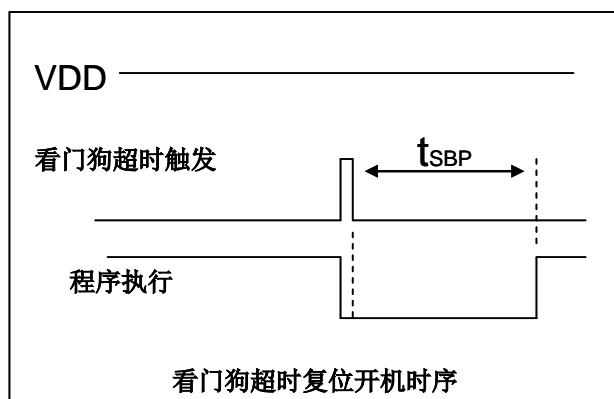


图 7：看门狗计数器超时溢出的相关时序

5.8. 中断

PMC131/PMS131/PMS130 有 8 个中断源：四个外部中断源(PA0/PB5, PB0/PA4), ADC 中断源, Timer16 中断源, Timer2 中断源, Timer3 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 8, 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。

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。

由于堆栈与数据存储器共享, 在 Mini-C 模式, 堆栈位置与深度由编译程序安排。在汇编模式或自行定义堆栈深度时, 用户应仔细安排位置, 以防地址冲突。

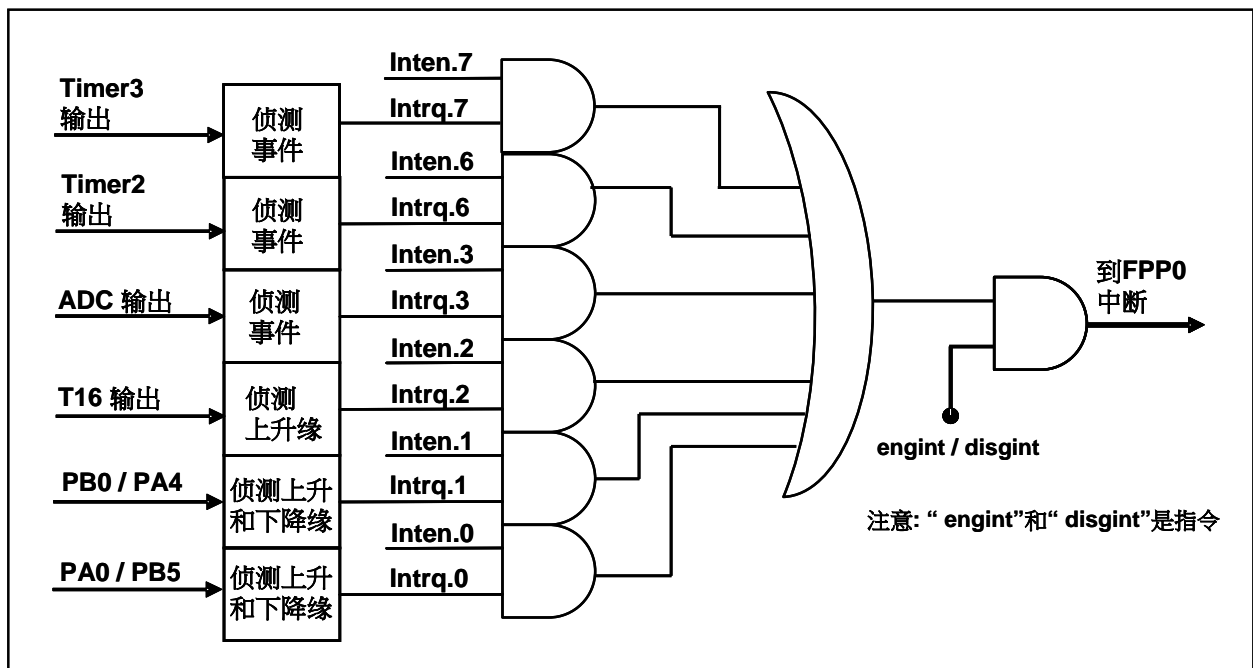


图 8: 中断硬件框图

一旦发生中断, 其工作流程将是:

- ◆ 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 0x010 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。

注意: 即使 *INTEN* 为 0, *INTRQ* 还是会被中断发生源触发。

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。

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。依此类推，另外 *pushaf* 也需要两个字节。下面的示例程序演示了如何处理中断，请注意，处理一级中断和 *pushaf* 总共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

    // 如果 INTEN.PA0 在主程序会动态开和关, 则表达式中可以判断 INTEN.PA0 是否为 1。
    // 例如: If (INTEN.PA0 && INTRQ.PA0) {...}

    // 如果 INTEN.PA0 一直在致能状态, 就可以省略判断 INTEN.PA0, 以加速中断执行。

    If (INTRQ.PA0)
    {
        // PA0 的中断程序
        INTRQ.PA0 = 0; // 只须清除相对应的位 (PA0)
        ...
    }
    ...
    // X: INTRQ = 0; // 不建议在中断程序最后, 才使用 INTRQ = 0 一次全部清除
    // 因为它可能会把刚发生而尚未处理的中断, 意外清除掉
    POPAF                // 回复 ALU 和 FLAG 寄存器
}
```

5.9. 省电与掉电模式

PMC131/PMS131/PMS130 有三个由硬件定义的工作模式，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 9 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 9：省电模式和掉电模式在振荡器模块的差异

5.9.1. 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)。系统唤醒后，单片机将继续正常的运行，省电模式的详细信息如下所示：

- ◆ IHRC 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。
- ◆ ILRC 振荡器模块：必须保持启用，唤醒时需要靠 ILRC 启动。
- ◆ 系统时钟禁用。因此，CPU 停止执行。
- ◆ OTP 存储器被关闭。
- ◆ Timer 计数器：若 Timer 计数器的时钟源是系统时钟或其相应的时钟振荡器模块被停用，则 Timer 停止计数；否则，仍然保持计数。（其中，Timer 包含 Timer16，TM2，TM3）
- ◆ 唤醒来源：
 - a. IO Toggle 唤醒：IO 在数字输入模式下的电平变换（PAC 位是 0，PADIER 位是 1）
 - b. Timer 唤醒：如果计数器 (Timer)的时钟源不是系统时钟，则当计数到设定值时，系统会被唤醒。

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：

```
$ 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。

5.9.2. 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 PMC131/PMS131/PMS130 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 (0x03)`寄存器的位 2 必须设置为 1。下面显示发出 `stopsys` 命令后， PMC131/PMS131/PMS130 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：设定为数字模式（PxDIER 对应位为 1）的 IO 切换。

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：

```
CLKMD = 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
```

5.9.3. 唤醒

进入掉电或省电模式后，PMC131/PMS131/PMS130 可以通过切换 IO 引脚恢复正常工作；而 Timer 唤醒的唤醒只适用于省电模式。图 10 显示 *stopsys* 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	计时器唤醒
<i>stopsys</i>	是	否
<i>stopexe</i>	是	是

图 10: 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 PMC131/PMS131/PMS130，寄存器 *padier* 和 *pbdir* 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，PMC131/PMS131/PMS130 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。特别注意，当 EOSC 被选用当系统时钟后，快速唤醒就自动关闭。

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
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
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计数器，等系统被唤醒后，在关闭快速唤醒之后再打开看门狗计数器。

5.10. IO 设置

对于数字功能而言，PMC131/PMS131/PMS130 所有的双向输入/输出端口都可以使用数据寄存器(pa, pb)，控制寄存器(pac, pbc)和弱上拉电阻(paph, pbph)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。当这些引脚设定为输出低电位或开漏模式下，弱上拉电阻会自动关闭。如果要读取 IO 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 11 显示了 IO 缓冲区硬件图。

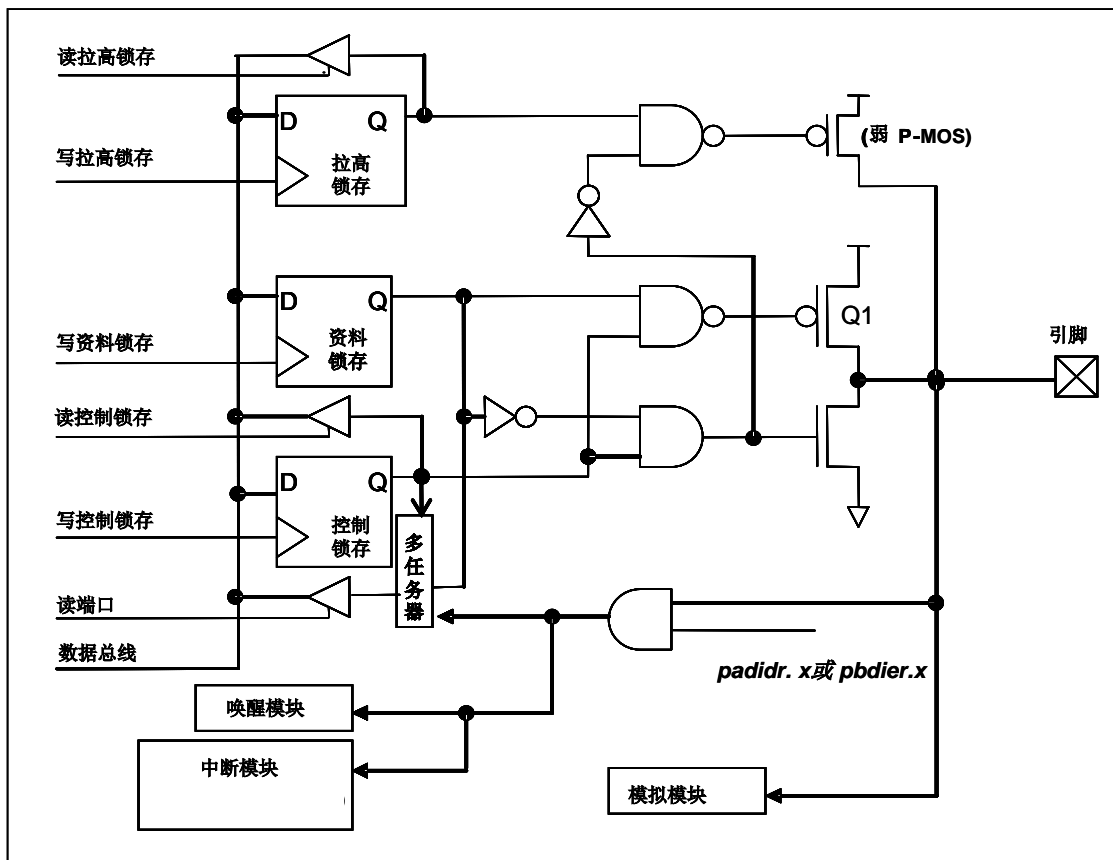


图 11: IO 缓冲区硬件图

使用 PA0 为例，表 5 显示了端口 A 位 0 的配置表。

pa.0	pac.0	paph.0	功能描述
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 5: 端口 A 位 0 的配置表

除 PA5 外所有其它的 IO 口具有相同的结构。PA5 的输出只能是漏极开路模式（没有 Q1）。对于被选择为仿真功能的引脚，必须在寄存器 padier 以及 pbdier 相应位设置为低，以防止漏电流。当 PMC131/PMS131/PMS130 在掉电模式或者省电模式，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 IO 口用来作为外部中断引脚时，PADIER/PBDIER 相应的位应设置为高。

5.11. 复位和 LVR

5.11.1. 复位

复位 PMC131/PMS131/PMS130 有很多原因，一旦发生复位，大部份 PMC131/PMS131/PMS130 的寄存器将被设置为上电初始值。系统在出现异常情况应当重新启动，或跳跃程序计数器到 0x0 来解决的。当复位来自上电或 LVR 时，数据存储器处在不确定的状态，但若来自 PRSTB 引脚复位或 WDT 超时溢位复位，内容将保持不变。

5.11.2. LVR 复位

程序编译时，用户可以选择 8 个不同级别的 LVR：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 LVR 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。

5.12. 数字转换(ADC)模块

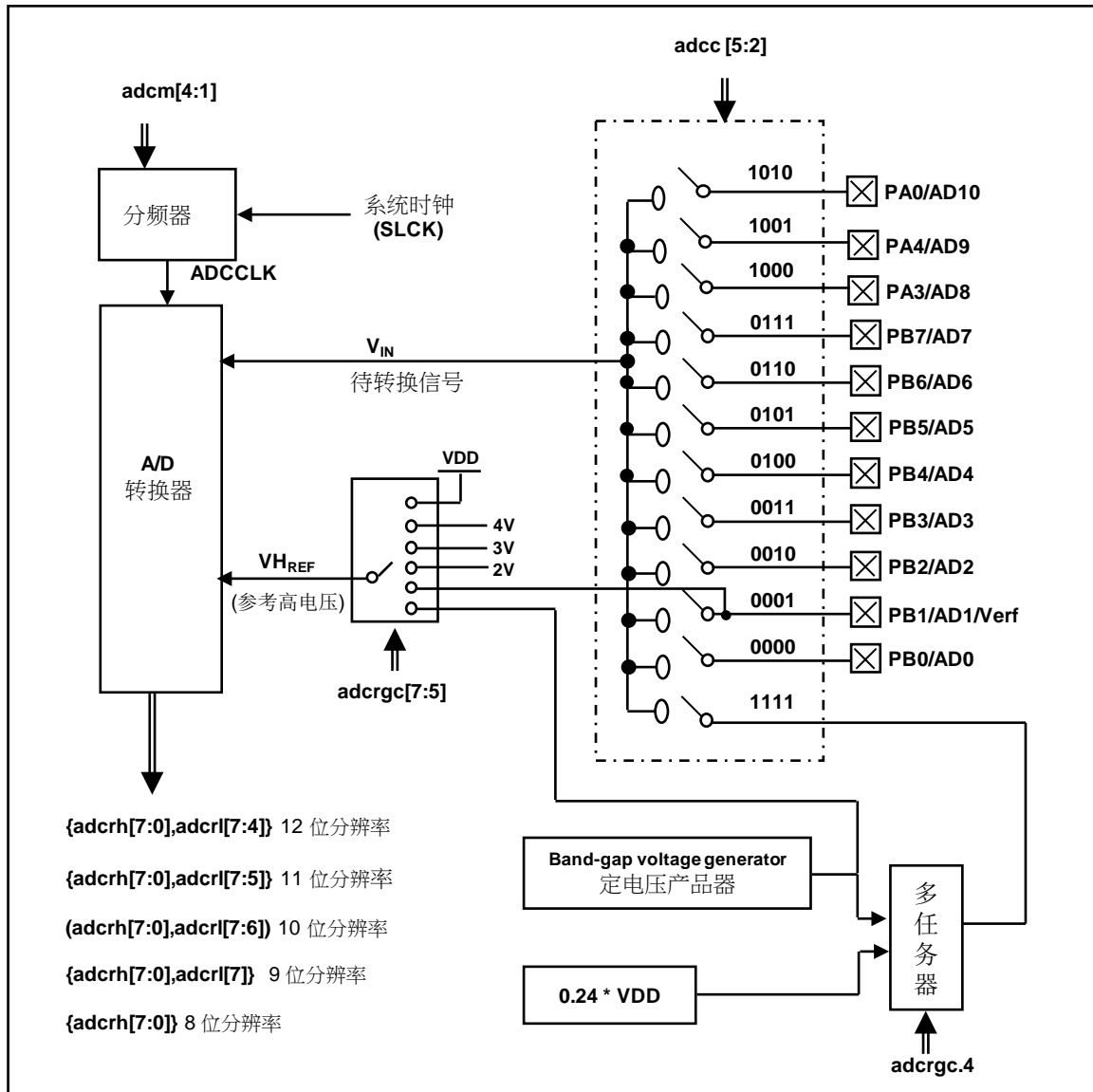


图 12: ADC 模块框图

ADC 模块有 5 个寄存器，分别是：

- ◆ ADC 控制寄存器 (*adcc*)
- ◆ ADC 调节控制寄存器(*adcrhc*)
- ◆ ADC 模式控制寄存器 (*adcm*)
- ◆ ADC 数据高位/低位寄存器(*adcrh, adcl*)
- ◆ 端口 A/B 数字输入禁用寄存器 (*padier, pbdier*)

做 AD 转换建议使用者遵守下面的步骤：

(1) ADC 模块的配置与设定：

- ◆ 利用 **adcrhc** 寄存器编程设置高电压参考
- ◆ 利用 **adcc** 寄存器选择 ADC 输入通道
- ◆ 利用 **adcm** 寄存器配置 ADC 分辨率
- ◆ 利用 **adcm** 寄存器配置 ADC 转换时钟
- ◆ 利用 **padier, pbdier** 寄存器配置所选定的引脚作为模拟输入
- ◆ 利用 **adcc** 寄存器启用 ADC 模块

(2) 配置 ADC 模块的中断：（如果需要）

- ◆ 清零 **intrq** 寄存器位 3 的 ADC 中断请求标志
- ◆ 启用 **inten** 寄存器位 3 的 ADC 中断请求
- ◆ 利用 **engint** 指令启用全局中断

(3) 启动 ADC 转换：

- ◆ 利用 **adcc** 寄存器置位 ADC 转换过程控制位启动转换(**set1 adcc.6**)

(4) 等待完成 AD 转换标志位置位，方法可以用如下的任一种：

- ◆ 检查 **adcc.6** 状态来等待完成的标志位置；或
- ◆ 等待 ADC 的中断

(5) 读取 ADC 的数据寄存器

- ◆ 读取 **adcrh, adcrh** 数据寄存器

(6) 下一个转换，依要求转到步骤 1 或第 2 步。

5.12.1. AD 转换的输入要求

为了满足 AD 转换的准确性，电荷保持电容 (C_{HOLD}) 必须完全充电到参考高电压以及放电到参考低电压的水平。模拟输入电路模型图如图 13 所示，信号驱动源阻抗 (R_s) 和内部采样开关阻抗 (R_{ss}) 将影响到电荷保持电容 C_{HOLD} 充电所需要的时间。内部采样开关阻抗可能会因 ADC 的电源电压 V_{DD} 有所变化，信号驱动源阻抗会影响到模拟输入信号的精度。用户必须确保在被测信号稳定时采样，因此，信号驱动源最大阻抗是与待量测信号频率有高度相关。建议在 500kHz 输入频率和 10 位精确度条件下，模拟信号源的最高阻抗为 10K Ω ；在 500Hz 输入频率和 10 位精确度条件下，为 10M Ω 。

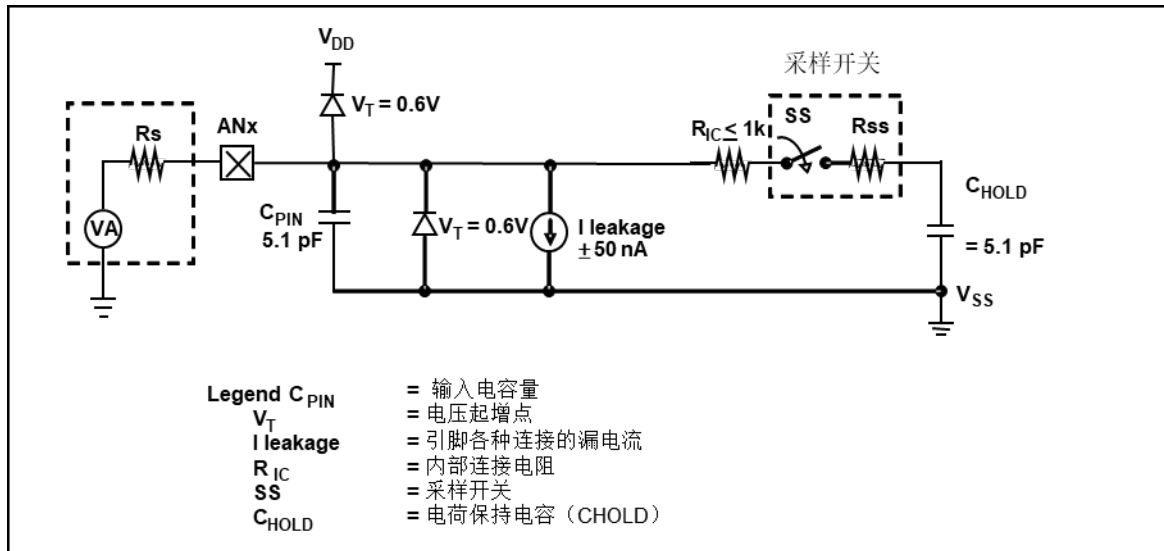


图 13: 模拟输入模型

在 AD 转换开始之前，必须确认所选模拟输入的信号采集时间应符合要求。ADCLK 的选择必须满足最短信号采集时间。

5.12.2. ADC 分辨率选择

ADC 的位分辨率也可选择从 8 位到 12 位，这取决于客户的应用要求。高的分辨率可以检测小信号的变化，但是，它需要更多的时间把模拟信号转换为数字信号。位分辨率的选择可以通过 **adcm** 寄存器设置。ADC 的位分辨率设定应在 AD 转换开始之前配置。

5.12.3. 选择参考高电压

ADC 的参考高电压可透过 **adcrhc** 寄存器位[7:5] 来选择。它的选择有 VDD、4V、3V、2V、1.20V band-gap 参考电压或来自外部引脚的 PB1。

5.12.4. ADC 时钟选择

ADC 模块的时钟 (ADCLK) 可以由设置 **adcm** 寄存器来选择，ADCLK 有 8 个可能的选择：从 $CLK \div 1$ 到 $CLK \div 128$ (CLK 是系统时钟)。由于信号采集时间 T_{ACQ} 是一个 ADCLK 时钟周期，所以该 ADCLK 必须满足这一要求。建议 ADC 模块时钟周期是 2us。

5.12.5. AD 转换

AD 转换的过程，从设置 START/DONE (adcc 位 6) 为高开始，START/DONE 的标志位内部将会自动清零，然后转换模拟信号将会一位一位的转换，当 AD 转换完成时，START/DONE 将自动置高表示完成转换。当 ADCLK 被选定后，ADCLK 的周期是 T_{ADCLK} ，而 AD 转换的时间将是如下：

- ◆ 8 位分辨率：AD 转换时间 = 13 T_{ADCLK}
- ◆ 9 位分辨率：AD 转换时间 = 14 T_{ADCLK}
- ◆ 10 位分辨率：AD 转换时间 = 15 T_{ADCLK}
- ◆ 11 位分辨率：AD 转换时间 = 16 T_{ADCLK}
- ◆ 12 位分辨率：AD 转换时间 = 17 T_{ADCLK}

5.12.6. 模拟引脚的配置

ADC 共有 12 个模拟信号可供选择: 11 个来自外部引脚的模拟输入信号以及内部信号 1.2V band-gap 参考电压或 $0.24 \times V_{DD}$ 。以外部引脚而言, 模数转换器的 12 个模拟输入信号与端口 Port A[0], Port A[3], Port A[4], and Port B[7:0] 共享引脚, 为了避免在设置为数字电路时发生漏电流, 这些引脚在当模拟输入时一定要利用寄存器 *padier*, *pbdier* 设置为模拟输入。对于那些定义为模拟输入的引脚, 当读 port A、B 时, 其值将为 0。

ADC 的测量信号属于小信号, 为避免测试信号在测量期间被干扰, 被选定的引脚应该 (1) 被设置为输入模式 (2) 关闭弱上拉电阻高 (3) 利用 *padier*, *pbdier* 寄存器配置所选定的引脚作为模拟输入。

5.12.7. 使用 ADC

下面的示例演示使用 PB0~PB3 来当 ADC 输入引脚。

首先, 定义所选择的引脚:

```
PBC           = 0B_XXXX_0000;    // PB0 ~ PB3 当输入
PBPH         = 0B_XXXX_0000;    // PB0 ~ PB3 没有弱上拉电阻
$ PBDIER     = 0B_XXXX_0000;    // PB0 ~ PB3 数据输入禁用
```

下一步, 设定 **ADCC** 寄存器, 示例如下:

```
$ ADCC Enable, PB3;    // 设定 PB3 当 ADC 输入
$ ADCC Enable, PB2;    // 设定 PB2 当 ADC 输入
$ ADCC Enable, PB0;    // 设定 PB0 当 ADC 输入
```

下一步, 设定 **ADCM** 和 **ADCRGC** 寄存器, 示例如下:

```
$ ADCM 12BIT, /16;    // 建议 /16 @系统时钟=8MHz
$ ADCM 12BIT, /8;     // 建议 /8 @系统时钟=4MHz
$ ADCRGC VDD;
```

接着, 开始 ADC 转换:

```
AD_START = 1;        // 开始 ADC 转换
do
{
    Nop;
}while(!AD_DONE);    // 等待 ADC 转换结果
```

最后, 当 **AD_DONE** 高电位时读取 ADC 结果:

```
WORD Data;          // 2 字节结果: ADCRH 和 ADCRL
Data = (ADCRH << 8) | ADCRL;
```

ADC 也可以利用下面方法禁用:

```
$ ADCC Disable;
```

或

```
ADCC = 0;
```

5.13. 乘法器

芯片上有一个 8x8 乘法器用来强化硬件的运算功能，它的乘法运算方式是一个 8x8 无符号运算并且能在一个时钟周期内完成。在未下达 `mul` 指令前，被乘数和乘数两者都必须放在 ACC 和 `mulop (0x08)` 寄存器上。待下达 `mul` 指令后，高字节结果将放在 `mulrh (0x09)` 寄存器上而低字节结果会放在 ACC 上。乘法器硬件图如图 14 显示：

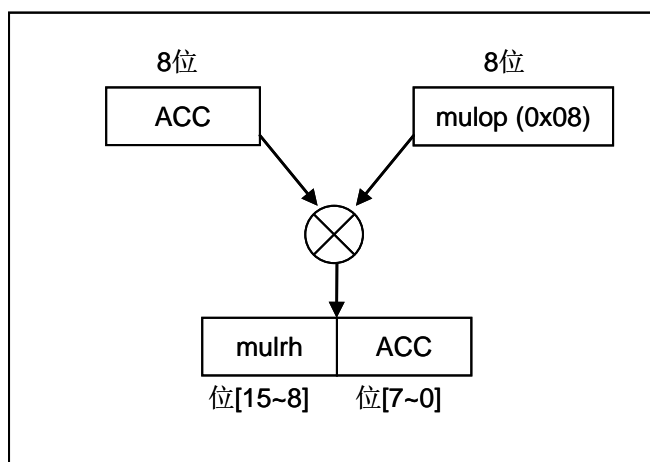


图 14: 乘法器硬件图

6. IO 寄存器

6.1. 算术逻辑状态寄存器 (*flag*), IO 地址 = 0x00

位	初始值	读/写	描述
7-4	-	-	保留。请不要使用。
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) 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 <i>shift</i> 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. 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。

6.3. 时钟控制寄存器 (*clkmd*), IO 地址 = 0x03

位	初始值	读/写	描述		
系统时钟选择					
		类型 0, <i>clkmd</i> [3]=0			
		类型 1, <i>clkmd</i> [3]=1			
7-5	111	读/写	<table style="width: 100%; border: none;"> <tr> <td style="width: 50%; border: none;"> 000: IHRC ÷ 4 001: IHRC ÷ 2 010: 保留 011: EOSC ÷ 4 100: EOSC ÷ 2 101: EOSC 110: ILRC ÷ 4 111: ILRC (默认) </td> <td style="width: 50%; border: none;"> 000: IHRC ÷ 16 001: IHRC ÷ 8 010: 保留 011: IHRC ÷ 32 100: IHRC ÷ 64 101: EOSC ÷ 8 11X: 保留 </td> </tr> </table>	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
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: 停用/启用		
3	0	读/写	时钟类型选择。此位用来选择开机后寄存器 <i>clkmd</i> 位[7:5]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: 停用/启用 如果 ILRC 停用时, 看门狗计数器也会被停用		
1	1	读/写	看门狗计数器功能。0/1: 停用/启用		
0	0	读/写	引脚 PA5/PRSTB 功能。0/1: PA5 / PRSTB		

6.4. 中断允许寄存器 (*inten*), IO 地址 = 0x04

位	初始值	读/写	描述
7	0	读/写	启用从 Timer3 的溢出中断。0/1: 停用/启用
6	0	读/写	启用从 Timer2 的溢出中断。0/1: 停用/启用
5:4	-	-	保留。
3	0	读/写	启用从模数转换器的中断。0/1: 停用/启用
2	0	读/写	启用从 Timer16 的溢出中断。0/1: 停用/启用
1	0	读/写	启用从 PB0/ PA4 的中断。0/1: 停用/启用
0	0	读/写	启用从 PA0/ PB5 的中断。0/1: 停用/启用

6.5. 中断请求寄存器 (*intrq*), IO 地址 = 0x05

位	初始值	读/写	描述
7	-	读/写	Timer3 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	-	读/写	Timer2 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
5:4	-	-	保留。
3	-	读/写	模拟数字转换器的中断请求, 此位是由硬件置位并由软件清零。0/1: 不要求/请求
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/PA4 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/PB5 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

6.6. 乘法器运算域寄存器 (*mulop*), IO 地址 = 0x08

位	初始值	读/写	描述
7-0	-	读/写	硬件乘法运算的运算域

6.7. 乘法器结果高字节寄存器 (*mulrh*), IO 地址= 0x09

位	初始值	读/写	描述
7-0	-	只读	乘法运算的高字节结果 (只读)

6.8. 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: 禁用 001: 系统时钟 010: 保留 011: PA4 下降缘 (来自外部引脚) 100: IHRC 101: EOSC 110: ILRC 111: PA0 下降缘 (来自外部引脚)
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。当选择位改变时, 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

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

位	初始值	读/写	描述
7	0	只写	启用外部晶体振荡器。 0/1: 停用/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。将来的兼容性, 请保留 0。
0	0	只写	将 Band-gap 与 LVR 硬件模块掉电处理。0/1: 正常 / 掉电

6.10. 中断边沿选择寄存器 (*integs*), IO 地址 = 0x0c

位	初始值	读/写	描述
7-5	-	-	保留。
4	0	只写	Timer16 中断沿选择。 0: 被选定的位上升沿触发中断 1: 被选定的位下降沿触发中断
3-2	00	只写	PB0/PA4 中断沿选择。 00: 被选定的位上升沿和下降沿都触发中断 01: 被选定的位上升沿触发中断 10: 被选定的位下降沿触发中断 11: 保留
1-0	00	只写	PA0/PB5 中断沿选择。 00: 被选定的位上升沿和下降沿都触发中断 01: 被选定的位上升沿触发中断 10: 被选定的位下降沿触发中断 11: 保留

6.11. 端口 A 数字输入使能寄存器 (*padier*), IO 地址 = 0x0d

位	初始值	读/写	描述
7	1	只写	使能 PA7 数字输入以及唤醒功能。 1/0: 启用/停用 当使用外部晶振时, 此位应设为 0 用以防止漏电流。 当此位设为 0 时, PA7 不能用来唤醒系统。
6	1	只写	使能 PA6 数字输入以及唤醒功能。 1/0: 启用/停用 当使用外部晶振时, 此位应设为 0 用以防止漏电流。 当此位设为 0 时, PA6 不能用来唤醒系统。
5	1	只写	使能 PA5 唤醒功能。 1/0: 启用/停用 此位设为 0 用来禁用 PA5 的唤醒功能。
4	1	只写	使能 PA4 数字输入、唤醒功能与中断请求。 1/0: 启用/停用 此位设为 0 用来防止引脚当模拟输入时漏电。 当此位设为 0 时, PA4 不能用来唤醒系统和外部中断。
3	1	只写	使能 PA3 数字输入以及唤醒功能。 1/0: 启用/停用 此位设为 0 用来防止引脚当模拟输入时漏电。 当此位设为 0 时, PA3 不能用来唤醒系统。
2-0	1	只写	保留。
0	1	只写	使能 PA0 数字输入、唤醒功能与中断请求。 1/0: 启用/停用 此位设为 0 用来防止引脚当模拟输入时漏电, 以及禁用 PA0 的唤醒功能和外部中断。

6.12. 端口 B 数字输入使能寄存器 (*pbdier*), IO 地址 = 0x0e

位	初始值	读/写	描述
7-6	11	只写	启用 PB7~ PB6 数字输入和系统唤醒。 1/0 : 启用 / 停用 当这些位设为 0 时, PB7~ PB6 无法用来唤醒系统。
5	1	只写	启用 PB5 数字输入、系统唤醒和中断请求。 1/0 : 启用 / 停用 当这个位设为 0 时, PB5 无法用来唤醒系统和中断请求。
4-1	1111	只写	启用 PB4~ PB1 数字输入和系统唤醒。 1/0 : 启用 / 停用 当这些位设为 0 时, PB4~ PB1 无法用来唤醒系统。
0	1	只写	启用 PB0 数字输入、系统唤醒和中断请求。 1/0 : 启用 / 停用 当这个位设为 0 时, PB0 无法用来唤醒系统和中断请求。

6.13. 端口 A 数据寄存器 (*pa*), IO 地址 = 0x10

位	初始值	读/写	描述
7-0	0x00	读/写	端口 A 资料寄存器

6.14. 端口 A 控制寄存器 (*pac*), IO 地址 = 0x11

位	初始值	读/写	描述
7-0	0x00	读/写	端口 A 控制寄存器。这个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 请注意, PA5 可以当输入或输出低; 当 PA5 设为输出高时, 为 OC 输出。

6.15. 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	0x00	读/写	端口 A 上拉控制寄存器。这个寄存器是用来控制端口 A 每个相应引脚的内部上拉功能。此上拉功能只有在输入模式时有效。 0/1: 停用/启用 请注意 PA5 没有上拉电阻。

6.16. 端口 B 数据寄存器 (*pb*), IO 地址 = 0x14

位	初始值	读/写	描述
7-0	0x00	读/写	端口 B 资料寄存器。

6.17. 端口 B 控制寄存器 (*pbc*), IO 地址 = 0x15

位	初始值	读/写	描述
7-0	0x00	读/写	端口 B 控制寄存器。这个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

6.18. 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	0x00	读/写	端口 B 上拉控制寄存器。这个寄存器是用来控制端口 B 每个相应引脚的内部上拉功能。 0/1: 停用/启用

6.19. ADC 控制寄存器 (*adcc*), IO 地址 = 0x20

位	初始值	读/写	描述
7	0	读/写	启用的 ADC 功能。0/1: 停用/启用
6	0	读/写	模数转换器过程控制位。 写“1”开始 AD 转换，同时自动清零完成标志。 读到“1”表示完成 AD 的转换。读到“0”表示进行中。
5-2	0000	读/写	通道选择器。这 4 个位用于选择 AD 转换的输入信号。 0000: PB0/AD0, 0001: PB1/AD1, 0010: PB2/AD2, 0011: PB3/AD3, 0100: PB4/AD4, 0101: PB5/AD5, 0110: PB6/AD6, 0111: PB7/AD7, 1000: PA3/AD8 1001: PA4/AD9 1010: PA0/AD10, 1111: (通道 F) Band-gap 1.20 volt 参考电位或者 $0.24 \cdot V_{DD}$ 其它: 保留。
0-1	-	-	保留。(将来兼容性, 请保留 0)

6.20. ADC 调节控制寄存器 (*adcr gc*), IO 地址 = 0x1c

位	初始值	读/写	Description
7-5	000	只写	这 3 个位用于选择 AD 参考电压。 000: V_{DD} , 001: 2V, 010: 3V, 011: 4V, 100: PB1, 101: Band-gap 1.20 volt 参考电位 其它: 保留。
4	0	只写	ADC 通道 F 选择器 0: Band-gap 1.20 volt 参考电位 1: $0.24 \cdot V_{DD}$ 。一般而言, 电压偏移约在 $\pm 0.01 \cdot V_{DD}$ 以内。
3-0	-	-	保留。请设为 0。

6.21. ADC 模式控制寄存器 (*adcm*), IO 地址 = 0x21

位	初始值	读/写	描述
7-5	000	只写	位分辨率。 000: 8-bit, AD 8-bit result [7:0] = <i>adcrh</i> [7:0]. 001: 9-bit, AD 9-bit result [8:0] = { <i>adcrh</i> [7:0], <i>adcr</i> [7] }. 010: 10-bit, AD 10-bit result [9:0] = { <i>adcrh</i> [7:0], <i>adcr</i> [7:6] }. 011: 11-bit, AD 11-bit result [10:0] = { <i>adcrh</i> [7:0], <i>adcr</i> [7:5] }. 100: 12-bit, AD 12-bit result [11:0] = { <i>adcrh</i> [7:0], <i>adcr</i> [7:4] }. 其它: 保留。
4	-	-	保留。将来的兼容性, 请保留 0。
3-1	000	只写	ADC 时钟源的选择。 000: CLK (系统时钟) ÷1, 001: CLK (系统时钟) ÷2, 010: CLK (系统时钟) ÷4, 011: CLK (系统时钟) ÷8, 100: CLK (系统时钟) ÷16, 101: CLK (系统时钟) ÷32, 110: CLK (系统时钟) ÷64, 111: CLK (系统时钟) ÷128
0	-	-	保留。

6.22. ADC 数据高位寄存器 (*adcrh*), IO 地址 = 0x22

位	初始值	读/写	描述
7-0	-	只读	这 8 个只读位是 AD 转换的结果的位[11: 4]。这个寄存器的位 7 是 ADC 转换结果的最高位。

6.23. ADC 数据低位寄存器 (*adcr*), IO 地址 = 0x23

位	初始值	读/写	描述
7-4	-	只读	这 4 个只读位是 AD 转换的结果的位[3: 0]。
3-0	-	-	保留。

6.24. 杂项寄存器 (misc), IO 地址 = 0X1b

位	初始值	读/写	描述
7	-	-	保留。将来的兼容性，请保留 0。
6	0	只写	校正后，起振 32kHz 晶体振荡器，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	只写	启动快速唤醒功能。快速唤醒功能并不能启动 EOSC。 0: 正常唤醒。唤醒时间为 1024 ILRC 时钟。 1: 快速唤醒。 唤醒时间为 128 个系统时钟+ 震荡稳定时间。 如果从 STOPEXE 唤醒，震荡稳定时间为 0 如果从 STOPSYS 唤醒，震荡稳定时间为 IHRC 或 ILRC 的上电震荡稳定时间 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计数器，等系统被唤醒后，在关闭快速唤醒之后再打开看门狗计数器。
4	-	-	保留。将来的兼容性，请保留 0。
3	0	只写	LVR 复原时间。 0: 正常。LVR 开机时间为 1024 ILRC 时钟周期。 1: 快速。LVR 开机时间为 64ILRC。
2	0	只写	禁用 LVR 功能。 0/1 : 启用/停用
1-0	00	只写	看门狗溢出时间设置。 00: 2048 ILRC 时钟周期 01: 4096 ILRC 时钟周期 10: 16384 ILRC 时钟周期 11: 256 ILRC 时钟周期

6.25. Timer2 控制寄存器 (*tm2c*), IO 地址 = 0x3c

位	初始值	读/写	描述
7-4	0000	读/写	Timer2 时钟选择。 0000: 禁用 0001: CLK (系统时钟) 0010: IHRC 时钟 0011: 保留 0100: ILRC 时钟 0101: 保留 011X: 保留 1000: PA0 (上升沿) 1001: ~PA0 (下降沿) 1010: PB0 (上升沿) 1011: ~PB0 (下降沿) 1100: PA4 (上升沿) 1101: ~PA4 (下降沿) 注意: 在 ICE 模式且 IHRC 被选为 Timer2 计数器时钟, 当 ICE 停下时, 发送到计数器的时钟是不停止, 计数器仍然继续计数。
3-2	00	读/写	Timer2 输出选择。 00: 停用 01: PB2 10: PA3 11: PB4
1	0	读/写	Timer2 模式选择。 0: 周期模式 1: PWM 模式
0	0	读/写	启用 Timer2 反极性输出。 0/1: 停用/启用。

6.26. Timer2 计数寄存器 (*tm2ct*), IO 地址 = 0x3d

位	初始值	读/写	描述
7-0	0x00	读/写	Timer2 计数寄存器位[7:0]。

6.27. Timer2 分频器寄存器 (*tm2s*), IO 地址 = 0x37

位	初始值	读/写	描述
7	0	只写	PWM 分辨率选择。 0: 8 位 1: 6 位
6-5	00	只写	Timer2 时钟预分频器。 00: ÷ 1 01: ÷ 4 10: ÷ 16 11: ÷ 64
4-0	00000	只写	Timer2 时钟分频器。

6.28.Timer2 上限寄存器 (*tm2b*), IO 地址= 0x09

位	初始值	读/写	描述
7-0	0x00	只写	Timer2 上限寄存器。

6.29.Timer3 控制寄存器 (*tm3c*), IO 地址= 0x2e

位	初始值	读/写	描述
7-4	0000	读/写	Timer3 时钟选择。 0000 : 停用 0001 : CLK (系统时钟) 0010 : IHRC 0011 : 保留 0100 : ILRC 0101 : 保留 011x : 保留 1000 : PA0 (上升沿) 1001 : ~PA0 (下降沿) 1010 : PB0 (上升沿) 1011 : ~PB0 (下降沿) 1100 : PA4 (上升沿) 1101 : ~PA4 (下降沿) 注意: 在 ICE 模式且 IHRC 被选为 Timer3 计数器时钟, 当 ICE 停下时, 发送到计数器的时钟是不停止, 计数器仍然继续计数。
3-2	00	读/写	Timer3 输出选择 00 : 停用 01 : PB5 10 : PB6 11 : PB7
1	0	读/写	Timer3 模式选择 0 / 1 : 周期模式/ PWM 模式
0	0	读/写	启用 Timer3 反极性输出。 0 / 1: disable / enable.

6.30.Timer3 控制寄存器 (*tm3ct*), IO 地址 = 0x2f

位	初始值	读/写	描述
7-0	0x00	只写	Timer3 计数寄存器位[7:0]

6.31.Timer3 分频寄存器 (*tm3s*), IO 地址 = 0x39

位	初始值	读/写	描述
7	0	只读	PWM 分辨率选择。 0: 8 位 1: 6 位
6-5	00	只读	Timer3 时钟预分频器。 00: ÷ 1 01: ÷ 4 10: ÷ 16 11: ÷ 64
4-0	00000	只读	Timer3 时钟分频器。

6.32.Timer3 上限寄存器 (*tm3b*), IO 地址 = 0x23

位	初始值	读/写	描述
7-0	0x00	只写	Timer3 上限寄存器

6.33.复位状态寄存器 (*rstst*), IO 地址 = 0x25

位	初始值	读/写	描述
7-4	-	-	保留。
3	-	读/写	外部复位引脚 (PA5) 的复位标志。当发生外部引脚复位时, 该位为 1。 此位写入 0 即清除该标志。
2	-	读/写	VDD 低于 4V 标志。当 VDD 电压低于 4V 时, 该位为 1。 此位写入 0 即清除该标志。 请注意, 当 VDD 上电较缓慢时, 此位亦会自动置 1。如有需要, 建议客户在程序初始化阶段将此位清零。
1	-	读/写	VDD 低于 3V 标志。当 VDD 电压低于 3V 时, 该位为 1。 此位写入 0 即清除该标志。 请注意, 当 VDD 上电较缓慢时, 此位亦会自动置 1。如有需要, 建议客户在程序初始化阶段将此位清零。
0	-	读/写	VDD 低于 2V 标志。当 VDD 电压低于 2V 时, 该位为 1。 此位写入 0 即清除该标志。 请注意, 当 VDD 上电较缓慢时, 此位亦会自动置 1, 如有需要, 建议客户在程序初始化阶段将此位清零。

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 是程序使用累加器的符号）
sp	堆栈指针
flag	累加器状态标志寄存器
 	实时数据
&	逻辑 AND
 	逻辑 OR
←	移动
^	异或 OR
+	加
-	减
~	按位取反（逻辑补码，1 补码）
⌋	负数（2 补码）
OV	溢出（2 补码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	CPU 的程序计数器
word	只允许寻址在 address 0~0x1F (0~31) 的位置
M.n	只允许寻址在 address 0~0xF (0~15) 的位置

7.1. 数据传输类指令

<i>mov</i> a, l	<p>移动实时数据到累加器 例如: <i>mov</i> a, 0x0f; 结果: $a \leftarrow 0fh$; 受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』</p>
<i>mov</i> M, a	<p>移动数据由累加器到存储器 例如: <i>mov</i> MEM, a; 结果: $MEM \leftarrow a$ 受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』</p>
<i>mov</i> a, M	<p>移动数据由存储器到累加器 例如: <i>mov</i> a, MEM; 结果: $a \leftarrow MEM$; 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』</p>
<i>mov</i> a, IO	<p>移动数据由 IO 到累加器 例如: <i>mov</i> a, pa; 结果: $a \leftarrow pa$; 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』</p>
<i>mov</i> IO, a	<p>移动数据由累加器到 IO 例如: <i>mov</i> pb, a; 结果: $pb \leftarrow a$; 受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』</p>
<i>ldt16</i> word	<p>将 Timer16 的 16 位计算值复制到 RAM。 例如: <i>ldt16</i> word; 结果: $word \leftarrow 16\text{-bit timer}$ 受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』 应用范例:</p> <pre> ----- 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 ----- </pre>
<i>stt16</i> word	<p>将放在 word 的 16 位 RAM 复制到 Timer16。 例如: <i>stt16</i> word; 结果: $16\text{-bit timer} \leftarrow word$ 受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』 应用范例:</p> <pre> ----- 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 ... ----- </pre>

<i>idxm</i> a, index	<p>使用索引作为 RAM 的地址并将 RAM 的数据读取并加载到累加器。需要 2T 时间执行这一指令。 例如: <i>idxm</i> a, index; 结果: $a \leftarrow [\text{index}]$, index 是用 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:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin: 0;">word RAMIndex ; // 定义一个 RAM 指标 ... 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), mov hb@RAMIndex, a ; // 将指针存到 RAM (MSB) ... idxm a, RAMIndex ; // 将 RAM 地址为 0x5B 的数据读取并加载累加器</pre> <hr style="border-top: 1px dashed black;"/>
<i>ldxm</i> index, a	<p>使用索引作为 RAM 的地址并将累加器的数据读取并加载到 RAM。需要 2T 时间执行这一指令。 例如: <i>ldxm</i> index, a; 结果: $[\text{index}] \leftarrow a$; 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:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin: 0;">word RAMIndex ; // 定义一个 RAM 指标 ... 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) 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加载地址为 0x5B 的 RAM</pre> <hr style="border-top: 1px dashed black;"/>
<i>xch</i> M	<p>累加器与 RAM 之间交换数据 例如: <i>xch</i> MEM ; 结果: $\text{MEM} \leftarrow a, a \leftarrow \text{MEM}$ 受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』</p> <hr style="border-top: 1px dashed black;"/>
<i>pushaf</i>	<p>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: <i>pushaf</i>; 结果: $[\text{sp}] \leftarrow \{\text{flag}, \text{ACC}\}$; $\text{sp} \leftarrow \text{sp} + 2$; 受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』 应用范例:</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin: 0;">.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数据存到堆栈存储器 ... // 中断服务程序 ... // 中断服务程序 popaf ; // 将堆栈存储器的数据回存到累加器和算术逻辑状态寄存器 reti ;</pre> <hr style="border-top: 1px dashed black;"/>

<i>popaf</i>	<p>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</p> <p>例如: <code>popaf</code>;</p> <p>结果: <code>sp ← sp - 2 ;</code> <code>{Flag, ACC} ← [sp];</code></p> <p>受影响的标志位: Z: 『受影响』, C: 『受影响』, AC: 『受影响』, OV: 『受影响』</p>
--------------	--

7.2. 算术运算类指令

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

<i>sub</i> M, a	RAM 减累加器，然后把结果放入 RAM 例如: <i>sub</i> MEM, a; 结果: $MEM \leftarrow MEM - a$ (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』
<i>subc</i> a, M	累加器减 RAM，再减进位，然后把结果放入累加器 例如: <i>subc</i> a, MEM; 结果: $a \leftarrow a - MEM - C$ 受影响的标志位: Z: 『受影响』, C: 『受影响』, AC: 『受影响』, OV: 『受影响』
<i>subc</i> M, a	RAM 减累加器，再减进位，然后把结果放入 RAM 例如: <i>subc</i> MEM, a; 结果: $MEM \leftarrow MEM - a - C$ 受影响的标志位: Z: 『受影响』, C: 『受影响』, AC: 『受影响』, OV: 『受影响』
<i>subc</i> a	累加器减进位，然后把结果放入累加器 例如: <i>subc</i> a; 结果: $a \leftarrow a - C$ 受影响的标志位: Z: 『受影响』, C: 『受影响』, AC: 『受影响』, OV: 『受影响』
<i>subc</i> M	RAM 减进位，然后把结果放入 RAM 例如: <i>subc</i> MEM; 结果: $MEM \leftarrow MEM - C$ 受影响的标志位: Z: 『受影响』, C: 『受影响』, AC: 『受影响』, OV: 『受影响』
<i>inc</i> M	RAM 加 1 例如: <i>inc</i> MEM; 结果: $MEM \leftarrow MEM + 1$ 受影响的标志位: Z: 『受影响』, C: 『受影响』, AC: 『受影响』, OV: 『受影响』
<i>dec</i> M	RAM 减 1 例如: <i>dec</i> MEM; 结果: $MEM \leftarrow MEM - 1$ 受影响的标志位: Z: 『受影响』, C: 『受影响』, AC: 『受影响』, OV: 『受影响』
<i>clear</i> M	清除 RAM 为 0 例如: <i>clear</i> MEM; 结果: $MEM \leftarrow 0$ 受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』
<i>mul</i>	乘法运算，执行 8x8 无符号乘法 例如: <i>mul</i> ; 结果: {MulRH,ACC} \leftarrow ACC * MulOp 受影响的标志位: 『不变』 Z 『不变』 C 『不变』 AC 『不变』 OV 应用范例: ... mov a, 0x5a ; mov mulop, a ; mov a, 0xa5 ; mul // 0x5A * 0xA5 = 3A02 (mulrh + ACC) mov ram0, a ; // LSB, ram0=0x02 mov a, mulrh ; // MSB, ACC=0X3A ...

7.3. 移位元运算类指令

<i>sr a</i>	累加器的位右移，位 7 移入值为 0 例如： <i>sr a</i> ； 结果： $a(0,b7,b6,b5,b4,b3,b2,b1) \leftarrow a(b7,b6,b5,b4,b3,b2,b1,b0)$, $C \leftarrow a(b0)$ 受影响的标志位：Z: 『不变』, C: 『受影响』, AC: 『不变』, OV: 『不变』
<i>src a</i>	累加器的位右移，位 7 移入进位标志位 例如： <i>src a</i> ； 结果： $a(c,b7,b6,b5,b4,b3,b2,b1) \leftarrow a(b7,b6,b5,b4,b3,b2,b1,b0)$, $C \leftarrow a(b0)$ 受影响的标志位：Z: 『不变』, C: 『受影响』, AC: 『不变』, OV: 『不变』
<i>sr M</i>	RAM 的位右移，位 7 移入值为 0 例如： <i>sr MEM</i> ； 结果： $MEM(0,b7,b6,b5,b4,b3,b2,b1) \leftarrow MEM(b7,b6,b5,b4,b3,b2,b1,b0)$, $C \leftarrow MEM(b0)$ 受影响的标志位：Z: 『不变』, C: 『受影响』, AC: 『不变』, OV: 『不变』
<i>src M</i>	RAM 的位右移，位 7 移入进位标志位 例如： <i>src MEM</i> ； 结果： $MEM(c,b7,b6,b5,b4,b3,b2,b1) \leftarrow MEM(b7,b6,b5,b4,b3,b2,b1,b0)$, $C \leftarrow MEM(b0)$ 受影响的标志位：Z: 『不变』, C: 『受影响』, AC: 『不变』, OV: 『不变』
<i>sl a</i>	累加器的位左移，位 0 移入值为 0 例如： <i>sl a</i> ； 结果： $a(b6,b5,b4,b3,b2,b1,b0,0) \leftarrow a(b7,b6,b5,b4,b3,b2,b1,b0)$, $C \leftarrow a(b7)$ 受影响的标志位：Z: 『不变』, C: 『受影响』, AC: 『不变』, OV: 『不变』
<i>slc a</i>	累加器的位左移，位 0 移入进位标志位 例如： <i>slc a</i> ； 结果： $a(b6,b5,b4,b3,b2,b1,b0,c) \leftarrow a(b7,b6,b5,b4,b3,b2,b1,b0)$, $C \leftarrow a(b7)$ 受影响的标志位：Z: 『不变』, C: 『受影响』, AC: 『不变』, OV: 『不变』
<i>sl M</i>	RAM 的位左移，位 0 移入值为 0 例如： <i>sl MEM</i> ； 结果： $MEM(b6,b5,b4,b3,b2,b1,b0,0) \leftarrow MEM(b7,b6,b5,b4,b3,b2,b1,b0)$, $C \leftarrow MEM(b7)$ 受影响的标志位：Z: 『不变』, C: 『受影响』, AC: 『不变』, OV: 『不变』
<i>slc M</i>	RAM 的位左移，位 0 移入进位标志位 例如： <i>slc MEM</i> ； 结果： $MEM(b6,b5,b4,b3,b2,b1,b0,C) \leftarrow MEM(b7,b6,b5,b4,b3,b2,b1,b0)$, $C \leftarrow MEM(b7)$ 受影响的标志位：Z: 『不变』, C: 『受影响』, AC: 『不变』, OV: 『不变』
<i>swap a</i>	累加器的高 4 位与低 4 位互换 例如： <i>swap a</i> ； 结果： $a(b3,b2,b1,b0,b7,b6,b5,b4) \leftarrow a(b7,b6,b5,b4,b3,b2,b1,b0)$ 受影响的标志位：Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』

7.4. 逻辑运算类指令

<i>and</i> a, l	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： <i>and</i> a, 0x0f; 结果： $a \leftarrow a \& 0fh$ 受影响的标志位：Z: [受影响], C: [不变], AC: [不变], OV: [不变]
<i>and</i> 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： <i>and</i> a, RAM10; 结果： $a \leftarrow a \& RAM10$ 受影响的标志位：Z: [受影响], C: [不变], AC: [不变], OV: [不变]
<i>and</i> M, a	累加器和 RAM 执行逻辑 AND，然后把结果保存到 RAM 例如： <i>and</i> MEM, a; 结果： $MEM \leftarrow a \& MEM$ 受影响的标志位：Z: [受影响], C: [不变], AC: [不变], OV: [不变]
<i>or</i> a, l	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： <i>or</i> a, 0x0f; 结果： $a \leftarrow a 0fh$ 受影响的标志位：Z: [受影响], C: [不变], AC: [不变], OV: [不变]
<i>or</i> 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： <i>or</i> a, MEM; 结果： $a \leftarrow a MEM$ 受影响的标志位：Z: [受影响], C: [不变], AC: [不变], OV: [不变]
<i>or</i> M, a	累加器和 RAM 执行逻辑 OR，然后把结果保存到 RAM 例如： <i>or</i> MEM, a; 结果： $MEM \leftarrow a MEM$ 受影响的标志位：Z: [受影响], C: [不变], AC: [不变], OV: [不变]
<i>xor</i> a, l	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： <i>xor</i> a, 0x0f; 结果： $a \leftarrow a \wedge 0fh$ 受影响的标志位：Z: [受影响], C: [不变], AC: [不变], OV: [不变]
<i>xor</i> a, M	累加器和 RAM 执行逻辑 XOR，然后把结果保存到累加器 例如： <i>xor</i> a, MEM; 结果： $a \leftarrow a \wedge RAM10$ 受影响的标志位：Z: [受影响], C: [不变], AC: [不变], OV: [不变]
<i>xor</i> M, a	累加器和 RAM 执行逻辑 XOR，然后把结果保存到 RAM 例如： <i>xor</i> MEM, a; 结果： $MEM \leftarrow a \wedge MEM$ 受影响的标志位：Z: [受影响], C: [不变], AC: [不变], OV: [不变]
<i>not</i> a	累加器执行 1 补码运算，结果放在累加器 例如： <i>not</i> a; 结果： $a \leftarrow \sim a$ 受影响的标志位：Z: [受影响], C: [不变], AC: [不变], OV: [不变] 应用范例： ----- <pre>mov a, 0x38; // ACC=0X38 not a; // ACC=0XC7</pre> -----

<i>not</i> M	<p>RAM 执行 1 补码运算，结果放在 RAM 例如： <i>not</i> 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：</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin-left: 40px;"> mov a, 0x38 ; mov mem, a ; // mem = 0x38 not mem ; // mem = 0xC7 </pre> <hr style="border-top: 1px dashed black;"/>
<i>neg</i> a	<p>累加器执行 2 补码运算，结果放在累加器 例如： <i>neg</i> 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：</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin-left: 40px;"> mov a, 0x38 ; // ACC=0X38 neg a ; // ACC=0XC8 </pre> <hr style="border-top: 1px dashed black;"/>
<i>neg</i> M	<p>RAM 执行 2 补码运算，结果放在 RAM 例如： <i>neg</i> 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：</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin-left: 40px;"> mov a, 0x38 ; mov mem, a ; // mem = 0x38 neg mem ; // mem = 0xC8 </pre> <hr style="border-top: 1px dashed black;"/>
<i>comp</i> a, M	<p>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： <i>comp</i> 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：</p> <hr style="border-top: 1px dashed black;"/> <pre style="margin-left: 40px;"> 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 </pre> <hr style="border-top: 1px dashed black;"/>
<i>comp</i> M, a	<p>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： <i>comp</i> 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』</p>

7.5. 位运算类指令

<i>set0</i> IO.n	<p>IO 的位 N 设为 0</p> <p>例如: <i>set0</i> pa.5 ;</p> <p>结果: PA5=0</p> <p>受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』</p>
<i>set1</i> IO.n	<p>IO 的位 N 设为 1</p> <p>例如: <i>set1</i> pb.5 ;</p> <p>结果: PB5=1</p> <p>受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』</p>
<i>swapc</i> IO.n	<p>IO 的第 n 位与进位标志位互换</p> <p>例如: <i>swapc</i> IO.0 ;</p> <p>结果: C ← IO.0 , IO.0 ← C</p> <p>当 IO.0 是输出脚位, 进位标志 C 将被送到 IO.0 脚</p> <p>当 IO.0 是输入脚位, IO.0 脚的状态将被送到进位标志 C</p> <p>受影响的标志位: Z: 『不变』, C: 『受影响』, AC: 『不变』, OV: 『不变』</p>
	<p>应用范例 1: (串行输出) :</p> <p>-----</p> <pre> ... 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 ... </pre> <p>应用范例 2: (串行输入)</p> <p>-----</p> <pre> ... 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 ... </pre> <p>-----</p>
<i>set0</i> M.n	<p>RAM 的位 N 设为 0</p> <p>例如: <i>set0</i> MEM.5 ;</p> <p>结果: MEM 位 5 为 0</p> <p>受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』</p>
<i>set1</i> M.n	<p>RAM 的位 N 设为 1</p> <p>例如: <i>set1</i> MEM.5 ;</p> <p>结果: MEM 位 5 为 1</p> <p>受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』</p>

7.6. 条件运算类指令

<i>ceqsn a, l</i>	<p>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 $(a \leftarrow a - l)$ 相同</p> <p>例如：<code>ceqsn a, 0x55;</code> <code>inc MEM;</code> <code>goto error;</code></p> <p>结果：假如 $a=0x55$，then “goto error”；否则，“inc MEM”。</p> <p>受影响的标志位： Z: 『受影响』， C: 『受影响』， AC: 『受影响』， OV: 『受影响』</p>
<i>ceqsn a, M</i>	<p>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 $(a \leftarrow a - M)$ 相同</p> <p>例如：<code>ceqsn a, MEM;</code></p> <p>结果：假如 $a=MEM$，跳过下一个指令</p> <p>受影响的标志位： Z: 『受影响』， C: 『受影响』， AC: 『受影响』， OV: 『受影响』</p>
<i>cneqsn a, M</i>	<p>比较累加器与 RAM，如果是不相同的，即跳过下一指令。标志位改变与 $(a \leftarrow a - M)$ 相同</p> <p>例如：<code>cneqsn a, MEM;</code></p> <p>结果：假如 $a \neq MEM$，跳过下一个指令</p> <p>受影响的标志位： Z: 『受影响』， C: 『受影响』， AC: 『受影响』， OV: 『受影响』</p>
<i>cneqsn a, l</i>	<p>比较累加器与立即数据，如果是不相同的，即跳过下一指令。标志位的改变与 $(a \leftarrow a - l)$ 相同</p> <p>例如：<code>cneqsn a, 0x55;</code> <code>inc MEM;</code> <code>goto error;</code></p> <p>结果：假如 $a \neq 0x55$，然后 “goto error”；否则，“inc MEM”。</p> <p>受影响的标志位： Z: 『受影响』， C: 『受影响』， AC: 『受影响』， OV: 『受影响』</p>
<i>t0sn IO.n</i>	<p>如果 IO 的指定位是 0，跳过下一个指令。</p> <p>例如：<code>t0sn pa.5;</code></p> <p>结果：如果 PA5 是 0，跳过下一个指令。</p> <p>受影响的标志位： Z: 『不变』， C: 『不变』， AC: 『不变』， OV: 『不变』</p>
<i>t1sn IO.n</i>	<p>如果 IO 的指定位是 1，跳过下一个指令。</p> <p>例如：<code>t1sn pa.5;</code></p> <p>结果：如果 PA5 是 1，跳过下一个指令。</p> <p>受影响的标志位： Z: 『不变』， C: 『不变』， AC: 『不变』， OV: 『不变』</p>
<i>t0sn M.n</i>	<p>如果 RAM 的指定位是 0，跳过下一个指令。</p> <p>例如：<code>t0sn MEM.5;</code></p> <p>结果：如果 MEM 的位 5 是 0，跳过下一个指令。</p> <p>受影响的标志位： Z: 『不变』， C: 『不变』， AC: 『不变』， OV: 『不变』</p>
<i>t1sn M.n</i>	<p>如果 RAM 的指定位是 1，跳过下一个指令。</p> <p>例如：<code>t1sn MEM.5;</code></p> <p>结果：如果 MEM 的位 5 是 1，跳过下一个指令。</p> <p>受影响的标志位： Z: 『不变』， C: 『不变』， AC: 『不变』， OV: 『不变』</p>
<i>izsn a</i>	<p>累加器加 1，若累加器新值是 0，跳过下一个指令。</p> <p>例如：<code>izsn a;</code></p> <p>结果：$a \leftarrow a + 1$，若 $a=0$，跳过下一个指令。</p> <p>受影响的标志位： Z: 『受影响』， C: 『受影响』， AC: 『受影响』， OV: 『受影响』</p>
<i>dzsn a</i>	<p>累加器减 1，若累加器新值是 0，跳过下一个指令。</p> <p>例如：<code>dzsn a;</code></p> <p>结果：$a \leftarrow a - 1$，若 $a=0$，跳过下一个指令。</p> <p>受影响的标志位： Z: 『受影响』， C: 『受影响』， AC: 『受影响』， OV: 『受影响』</p>

<i>izsn</i> M	RAM 加 1, 若 RAM 新值是 0, 跳过下一个指令。 例如: <code>izsn MEM;</code> 结果: $MEM \leftarrow MEM + 1$, 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』
<i>dzsn</i> M	RAM 减 1, 若 RAM 新值是 0, 跳过下一个指令。 例如: <code>dzsn MEM;</code> 结果: $MEM \leftarrow MEM - 1$, 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』

7.7. 系统控制类指令

<i>call</i> label	函数调用, 地址可以是全部空间的任一地址 例如: <code>call function1;</code> 结果: $[sp] \leftarrow pc + 1$ $pc \leftarrow function1$ $sp \leftarrow sp + 2$ 受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』
<i>goto</i> label	转到指定的地址, 地址可以是全部空间的任一地址 例如: <code>goto error;</code> 结果: 跳到 <code>error</code> 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』
<i>ret</i> l	将立即数据复制到累加器, 然后返回 例如: <code>ret 0x55;</code> 结果: $A \leftarrow 55h$ <code>ret;</code> 受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』
<i>ret</i>	从函数调用中返回原程序 例如: <code>ret;</code> 结果: $sp \leftarrow sp - 2$ $pc \leftarrow [sp]$ 受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』
<i>reti</i>	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: <code>reti;</code> 受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』
<i>nop</i>	没任何动作 例如: <code>nop;</code> 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』
<i>pcadd</i> a	目前的程序计数器加累加器成为下一个程序计数器。 例如: <code>pcadd a;</code> 结果: $pc \leftarrow pc + a$ 受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』 应用范例: ----- <pre>... mov a, 0x02 ; pcadd a ; // PC <- PC+2 goto err1 ;</pre>

	<pre> goto correct ; // 跳到这里 goto err2 ; goto err3 ; ... correct: // 跳到这里 ... </pre>
<i>engint</i>	<p>允许全部中断。</p> <p>例如: <i>engint</i>;</p> <p>结果: 中断要求可送到 CPU, 以便进行中断服务</p> <p>受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』</p>
<i>disgint</i>	<p>禁止全部中断。</p> <p>例如: <i>disgint</i>;</p> <p>结果: 送到 CPU 的中断要求全部被挡住, 无法进行中断服务</p> <p>受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』</p>
<i>stopsys</i>	<p>系统停止。</p> <p>例如: <i>stopsys</i>;</p> <p>结果: 停止系统时钟和关闭系统</p> <p>受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』</p>
<i>stopexe</i>	<p>CPU 停止。</p> <p>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。</p> <p>例如: <i>stopexe</i>;</p> <p>结果: 停住系统时钟, 但是仍保持震荡器模块工作</p> <p>受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』</p>
<i>reset</i>	<p>复位整个单片机, 其运行将与硬件复位相同。</p> <p>例如: <i>reset</i>;</p> <p>结果: 复位整个单片机</p> <p>受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』</p>
<i>wdreset</i>	<p>复位看门狗计数器。</p> <p>例如: <i>wdreset</i>;</p> <p>结果: 复位看门狗计数器</p> <p>受影响的标志位: Z: 『不变』, C: 『不变』, AC: 『不变』, OV: 『不变』</p>

7.8. 指令执行周期综述

2 个周期		<i>goto, call, pcadd, ret, reti, idxm</i>
2 个周期	条件满足	<i>ceqsn, cneqsn, t0sn, t1sn, dzsn, izsn</i>
1 个周期	条件不满足	
1 个周期		其他

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
<i>mov a, l</i>	-	-	-	-	<i>mov M, a</i>	-	-	-	-	<i>mov a, M</i>	Y	-	-	-
<i>mov a, IO</i>	Y	-	-	-	<i>mov IO, a</i>	-	-	-	-	<i>ldt16 word</i>	-	-	-	-
<i>stt16 word</i>	-	-	-	-	<i>idxm a, index</i>	-	-	-	-	<i>idxm index, a</i>	-	-	-	-
<i>xch M</i>	-	-	-	-	<i>pushaf</i>	-	-	-	-	<i>popaf</i>	Y	Y	Y	Y
<i>add a, l</i>	Y	Y	Y	Y	<i>add a, M</i>	Y	Y	Y	Y	<i>add M, a</i>	Y	Y	Y	Y
<i>addc a, M</i>	Y	Y	Y	Y	<i>addc M, a</i>	Y	Y	Y	Y	<i>addc a</i>	Y	Y	Y	Y
<i>addc M</i>	Y	Y	Y	Y	<i>nadd a, M</i>	Y	Y	Y	Y	<i>nadd M, a</i>	Y	Y	Y	Y
<i>sub a, l</i>	Y	Y	Y	Y	<i>sub a, M</i>	Y	Y	Y	Y	<i>sub M, a</i>	Y	Y	Y	Y
<i>subc a, M</i>	Y	Y	Y	Y	<i>subc M, a</i>	Y	Y	Y	Y	<i>subc a</i>	Y	Y	Y	Y
<i>subc M</i>	Y	Y	Y	Y	<i>inc M</i>	Y	Y	Y	Y	<i>dec M</i>	Y	Y	Y	Y
<i>clear M</i>	-	-	-	-	<i>mul</i>	-	-	-	-	<i>sr a</i>	-	Y	-	-
<i>src a</i>	-	Y	-	-	<i>sr M</i>	-	Y	-	-	<i>src M</i>	-	Y	-	-
<i>sl a</i>	-	Y	-	-	<i>slc a</i>	-	Y	-	-	<i>sl M</i>	-	Y	-	-
<i>slc M</i>	-	Y	-	-	<i>swap a</i>	-	-	-	-	<i>and a, l</i>	Y	-	-	-
<i>and a, M</i>	Y	-	-	-	<i>and M, a</i>	Y	-	-	-	<i>or a, l</i>	Y	-	-	-
<i>or a, M</i>	Y	-	-	-	<i>or M, a</i>	Y	-	-	-	<i>xor a, l</i>	Y	-	-	-
<i>wdreset</i>	-	-	-	-	<i>xor a, M</i>	Y	-	-	-	<i>xor M, a</i>	Y	-	-	-
<i>not a</i>	Y	-	-	-	<i>not M</i>	Y	-	-	-	<i>neg a</i>	Y	-	-	-
<i>neg M</i>	Y	-	-	-	<i>comp a, M</i>	Y	Y	Y	Y	<i>comp M, a</i>	Y	Y	Y	Y
<i>set0 IO.n</i>	-	-	-	-	<i>set1 IO.n</i>	-	-	-	-	<i>set0 M.n</i>	-	-	-	-
<i>set1 M.n</i>	-	-	-	-	<i>swapc IO.n</i>		Y			<i>ceqsn a, l</i>	Y	Y	Y	Y
<i>ceqsn a, M</i>	Y	Y	Y	Y	<i>cneqsn a, M</i>	Y	Y	Y	Y	<i>cneqsn a, l</i>	Y	Y	Y	Y
<i>t0sn IO.n</i>	-	-	-	-	<i>t1sn IO.n</i>	-	-	-	-	<i>t0sn M.n</i>	-	-	-	-
<i>t1sn M.n</i>	-	-	-	-	<i>izsn a</i>	Y	Y	Y	Y	<i>dzsn a</i>	Y	Y	Y	Y
<i>izsn M</i>	Y	Y	Y	Y	<i>dzsn M</i>	Y	Y	Y	Y	<i>call label</i>	-	-	-	-
<i>goto label</i>	-	-	-	-	<i>ret l</i>	-	-	-		<i>ret</i>	-	-	-	-
<i>reti</i>	-	-	-	-	<i>nop</i>	-	-	-	-	<i>pcadd a</i>	-	-	-	-
<i>engint</i>	-	-	-	-	<i>disgint</i>	-	-	-	-	<i>stopsys</i>	-	-	-	-
<i>stopexe</i>	-	-	-	-	<i>reset</i>	-	-	-						

7.10. RAM 定义限制

位寻址只能定义在 RAM 区的 0x00 到 0x3F 空间。

8. 代码选项(Code Options)

配置	选项	描述
Security	Enable	OTP 内容加密，程序不允许被读取
	Disable	OTP 内容不加密，程序可以被读取
LVR	4.1V	选择 LVR = 4.1V
	3.6V	选择 LVR = 3.6V
	3.1V	选择 LVR = 3.1V
	2.8V	选择 LVR = 2.8V
	2.5V	选择 LVR = 2.5V
	2.2V	选择 LVR = 2.2V
	2.0V	选择 LVR = 2.0V
	1.8V	选择 LVR = 1.8V
Under_20mS_VDD_OK	Yes	可以在 20 mS 内，迅速达到正常的工作电压
	No	无法在 20 mS 内，达到正常的工作电压
Interrupt Src0	PA.0	INTEN/INTRQ.0 源自 PA.0 中断
	PB.5	INTEN/INTRQ.0 源自 PB.5 中断
Interrupt Src1	PB.0	INTEN/INTRQ.1 源自 PB.0 中断
	PA.4	INTEN/INTRQ.1 源自 PA.4 中断

9. 特别注意事项

此章节是提醒使用者在使用 PMC131/PMS131/PMS130 时避免一些常犯的错误。

9.1. 警告

用户必须仔细阅读所有与此 IC 有关的 APN，才能使用此 IC。有关此 IC 的 APN 请于以下网站查阅：

<http://www.padauk.com.tw/tw/technical/index.aspx>

9.2. 使用 IC 时

9.2.1. IO 使用与设定

(1) IO 作为数字输入时

- ◆ IO 作为数字输入时， V_{ih} 与 V_{il} 准位，会随着电压与温度变化，请遵守 V_{ih} 的最小值， V_{il} 的最大值规范。
- ◆ 内部上拉电阻值将随着电压、温度与引脚电压而变动，并非为固定值。

(2) IO 作为数字输入和打开唤醒功能

- ◆ 将 IO 设为输入。
- ◆ 用 PADIER 和 PBDIER 寄存器，将对应的位设为 1。

(3) PA5 作为输出

- ◆ PA5 只能做 Open Drain 输出，输出高需要外加上拉电阻。

(4) PA5 作为 PRSTB 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 PRSTB 输入脚位。

(5) 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。**

(6) PA7 和 PA6 作为外部晶体振荡器

- ◆ 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

注意：请务必仔细阅读 PMC-APN013 之内容，并据此合理使用晶体振荡器。如因用户的晶体振荡器的质量不佳、使用条件不合理、PCB 清洁剂残留漏电、或是 PCB 板布局不合理等等用户原因，造成的慢起振或不起振情况，我司不对此负责。

(7) 以 PB1 当交流电 AC 过零检测 IO 口时，连接 PB1 的串接电阻阻值至少要 1M 欧姆(含)以上。

(8) 使用 PB3 IO 口注意事项：

使用 PB3 来当数字输入时，如果同时有使用 Port B 的其它 I/O 口来当输出，在对这些输出口设定输出准位时，请勿使用 set1/set0 指令，请改用对 Port B 寄存器 pb 整个写入的方式，来对 Port B 输出 I/O 口设定输出准位，或将 PB.3 改当输出 I/O 口来使用。

例如：使用 PB3 当数字输入 I/O 口，PB7 当数字输出 I/O 口

```
pbc    = 0b_1111_0111;
pb     = 0b_0000_0000;
pbph   = 0b_1000_1000; // PB3 设为内部上拉
$ pbdier 0b_1111_1111;
```

请勿使用 set1/set0 设定 PB7 输出口输出准位

```
if ( pb.3 )
{
    set1 pb.7; // 也请勿使用 pb.7 = 1;
}
else
{
    set0 pb.7; // 也请勿使用 pb.7 = 0;
}
```

请改用对 Port B 寄存器 pb 整个写入的方式，来对 PB7 输出 I/O 口设定输出准位

```
if ( pb.3 )
{
    pb = 0b_1000_0000; // 其中 bit 3 必需为 0
}
else
{
    pb = 0b_0000_0000; // 其中 bit 3 必需为 0
}
```


9.2.2. 中断

(1) 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 CPU 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，CPU 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) 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。

(3) 外部中断源新增 PA4 及 PB5。使用 PA4 当外部中断源，在 inten/intrq/integs 寄存器的设定与 PB0 相同，但是以 PADAUK_CODE_OPTION 中的 Interrupt_Src1 来设定是 PB0 还是 PA4 使能。使用 PB5 当外部中断源，在 inten/intrq/integs 寄存器的设定与 PA0 相同，但是以 PADAUK_CODE_OPTION 中的 Interrupt_Src0 来设定是 PA0 还是 PB5 致能。此外外部中断源只能在 PADAUK_CODE_OPTION 中设定，不能以程序来设定。

9.2.3. 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：系统时钟从 ILRC 切换到 IHRC/2。

◆ 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。
```

- (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;
do
{ nop; }while(!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。

9.2.4. 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) 在下 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。
- (4) 如果程序中使用看门狗，并且想快速唤醒，范例程序如下：

```
CLKMD.En_WatchDog =    0;        // disable watchdog timer
$ MISC    Fast_Wake_Up;
stopexe;
nop;
$ MISC    WT_xx;                // 重新设定 Watchdog time 并设为 normal wake-up
Wdreset;
CLKMD.En_WatchDog =    1;        // enable watchdog timer
```

9.2.5. TIMER16 溢出时间

当设定 \$ INTEGS BIT_R 时（这是 IC 默认值），且设定 T16M 计数器 BIT8 产生中断，若 T16 计数从 0 开始，则第一次中断是在计数到 0x100 时发生（BIT8 从 0 到 1），第二次中断在计数到 0x300 时发生（BIT8 从 0 到 1）。所以设定 BIT8 是计数 512 次才中断。请注意，如果在中断中重新给 T16M 计数器设值，则下一次中断也将在 BIT8 从 0 变 1 时发生。

如果设定 \$ INTEGS BIT_F（BIT 从 1 到 0 触发）而且设定 T16M 计数器 BIT8 产生中断，则 T16 计数改为每次数到 0x200/0x400/0x600/...时发生中断。两种设定 INTEGS 的方法各有好处，也请注意其中差异。

9.2.6. IHRC

- (1) 当 IC 在烧录器烧录时，会校准 IHRC 频率。
- (2) 校准 IHRC 时，不管是封装片机台烧录还是 COB 在线烧录，EMC 的干扰都会对 IHRC 的精度有影响。如果在封装前校准了 IHRC，那么在封装后 IHRC 的实际频率可能会出现偏差并超出规格范围。通常封装后频率会变慢一点。
- (3) 通常在 COB 封胶或 QTP 时会发生如上描述的情况，应广科技不负任何责任。
- (4) 用户可以根据使用经验来做频率补偿，例如，用户可以在使用时调高 IHRC 频率约 0.5%~1%，以便得到比 IC 封塑之后更好的 IHRC 频率。

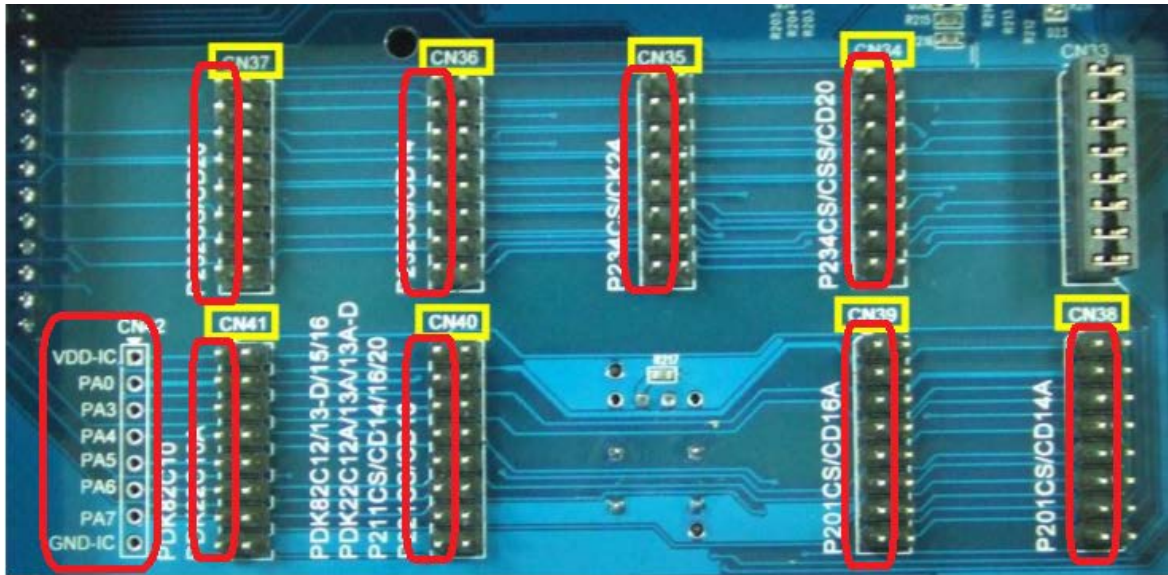
9.2.7. LVR

- (1) Power On 时， V_{DD} 需要到达或超过 2.0V 左右，IC 才能成功起动，否则 IC 不能工作。
- (2) 只有当 IC 正常起动后，设定 LVR=1.8V, 2.0V, 2.2V 才有作用。
- (3) 可以设定寄存器 MISC.2 为 1 将 LVR 关闭，但此时应确保 VDD 在 chip 最低工作电压以上，否则 IC 不能工作。

9.2.8. 烧录方法

PMC131/PMS131/PMS130 的烧录脚为 PA3, PA4, PA5, PA6, V_{DD} , GND 这 6 只引脚。

在 PDK3S-P-002 烧录器上，可以使用 CN38 跳线，并在烧录插座上把 IC 往下空 3 格，就可以烧录 S16/D16, S16A/D16A, S14/D14 这些封装；如为 S08/D08 及其他封装，可以自行跳接烧录接脚。烧录器背后的跳线，所有左侧的讯号都是一致的，就如左下角说明文字一样，分别为 V_{DD} , PA0（不需要），PA3, PA4, PA5, PA6, PA7（不需要），GND。



如使用 PDK5S-P-003 或以上进行烧录，并依照烧录器软件上说明，连接 jumper 即可。

◆ 合封（MCP）或在板烧录（On-Board Writing）时的有关电压和电流的注意事项

- (1) PA5 (V_{PP}) 可能高于 11V。
- (2) V_{DD} 可能高于 7V，而最大供给电流最高可达约 20mA。
- (3) 其他烧录引脚（GND 除外）的电位与 V_{DD} 相同。

请用户自行确认在使用本产品于合封或在板烧录时，周边电路及元件不会被上述电压破坏，也不会限制上述电压。

9.3. 使用 ICE 时

(1) 请使用 PDK5S-I-S01/2(B)仿真器。仿真时请注意以下几点：

- (a) 仿真时的快速唤醒时间和 IC 不一样；仿真器：128 个系统时钟； IC：45 ILRC 时钟
- (b) 当使用 ADCRGC 里的 PB1 时，PA1 必须浮空。