PMS160

6 触摸键 OTP 类型单片机

数据手册

第 0.01 版

2022 年 2 月 16 日

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。

目录

修订历史:

修订	日期	描述
0.00	2021/11/12	初版
0.01	2022/02/16	1. 补充第 1.2 节 LDO 模块及低功耗唤醒功能 2. 更新第 4.1 节直流交流电气特性：fSYS、VPOR 3. 补充第 5.4 节对 NILRC 振荡器的相关介绍 4. 修正第 5.14 节 11bit LPWMG 部分图文错误 5. 大幅修改与完善第 5.15 节关于触摸功能的相关介绍、注意事项以及使用范例 6. 修改第 6.24 节、第 6.33-6.36 节各寄存器 7. 补充第 6.37-6.39 节触摸相关寄存器 8. 删除第 8 章 LPWM_Source、GPC_LPWM 两个 Code Options，改由 GPC2PWM 寄存器分别控制，并修改其他关联处内容 9. 补充第 8 章 Interrupt_Src0、ICE_LPWM_INTR 两个 Code Options，并补充其他关联处内容 10. 补充第 9.3 节使用 ICE 及相关仿真注意事项 11. 其他已知细节错误修正

1. 功能

1.1. 特性

- 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°C ~ 70°C

1.2. 系统特性

- 1.5KW OTP 程序内存
- 96 字节数据存储器
- 最多可选择 6 个 IO 引脚作为触摸按键
- 一个硬件 16 位计数器
- 两个 8 位硬件 PWM 生成器 Timer2/Timer3，Timer2/Timer3 还配置 NILRC 振荡器，它的频率比 ILRC 更慢，适合做更省电的唤醒时钟
- 一组可设三路 11 位 SuLED (Super LED)PWM 生成器和计数器 LPWMG0/LPWMG1/LPWMG2
- 提供一个硬件比较器
- 6 个 IO 引脚并带有上拉/下拉电阻选项
- Bandgap 电路提供 1.2V Bandgap 电压
- 时钟源：内部高频 RC 振荡器(IHRC)，内部低频 RC 振荡器(ILRC)
- 14 段 LVR 复位设定：4.5V, 4.0V, 3.75V, 3.5V, 3.3V, 3.15V, 3.0V, 2.7V, 2.5V, 2.4V, 2.3V, 2.2V, 2.1V, 2.0V
- 两个可选的外部中断引脚
- 内部 LDO 可防止触摸噪声
- 支持低功耗(NILRC)定时唤醒 stopsys

1.3. CPU 特性

- 单一处理单元工作模式
- 提供 82 个有效指令
- 大部分都是 1T（单周期）指令
- 可程序设定的堆栈指针和堆栈深度（使用 2 bytes SRAM 作为一层堆栈）
- 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

- PMS160-S08A: SOP8 (150mil)
- PMS160-S08B: SOP8 (150mil)
- PMS160-2N08A: DFN (2*2mm)
- PMS160-2N08B: DFN (2*2mm)
- PMS160-2N06: DFN (2*2mm)
- PMS160-U06A: SOT23-6 (60mil)

2. 系统概述和方框图

PMS160 系列是一款完全静态的，以 OTP 为程序存储基础的 CMOS 8-bit 微处理器。它运用 RISC 的架构且大部分的指令执行都是一个指令周期的，只有少部分处理间接寻址指令需要两个指令周期。

PMS160 包含一个最多 6 键的电容式触摸控制电路。另外，PMS160 还包含 1.5KW OTP 程序内存以及 96 字节数据存储器，一个 16 位的硬件计数器，两个 8 位 Timer2/Timer3 计数器和一组新的三路 11 位计数器带 SuLED PWM 生成器(LPWMG0/1/2)。

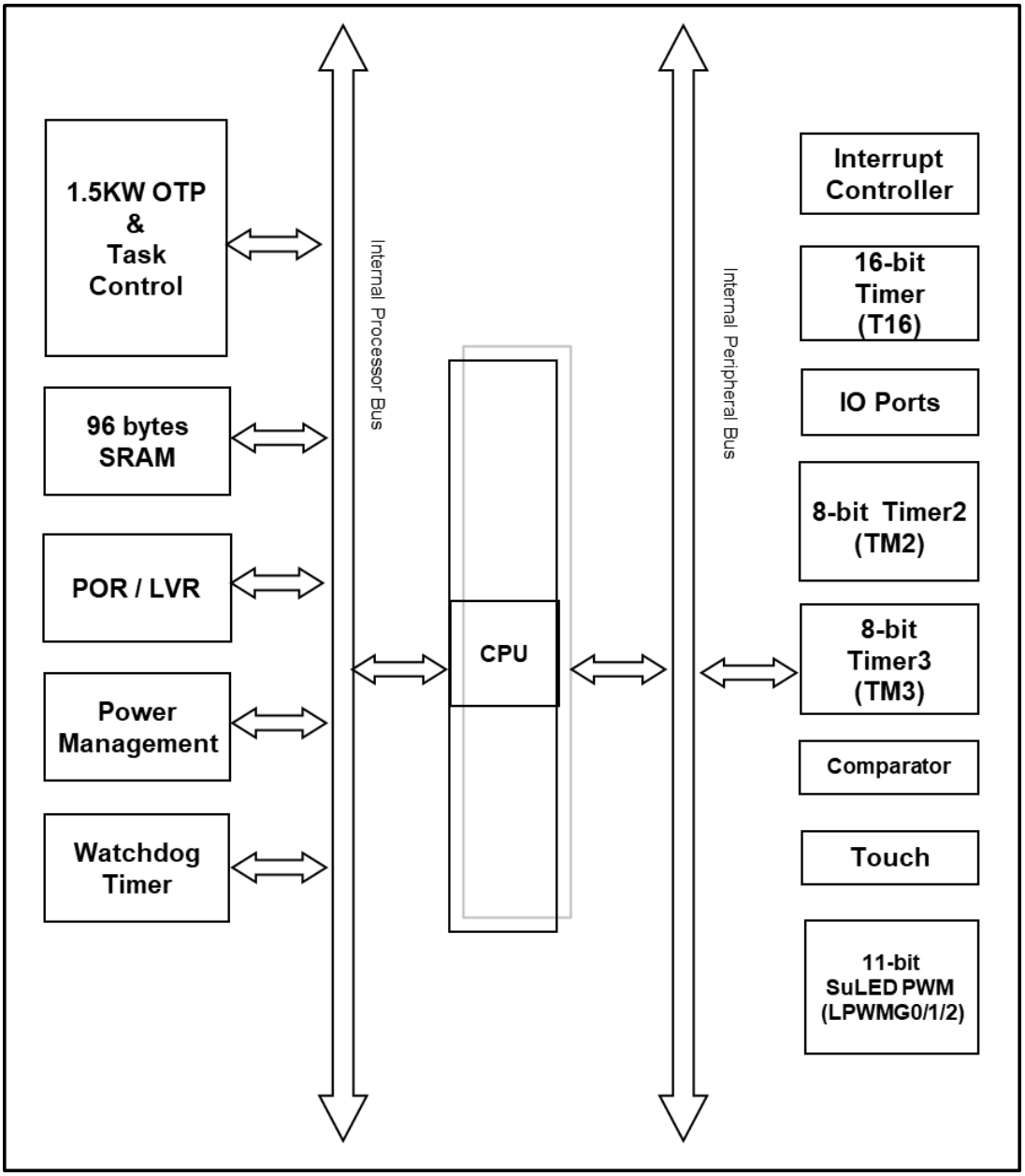
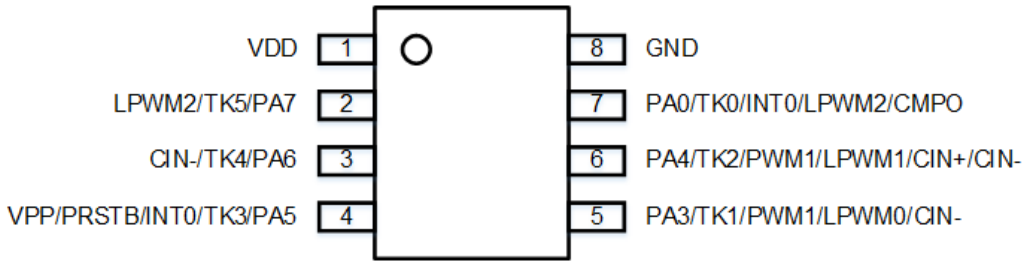
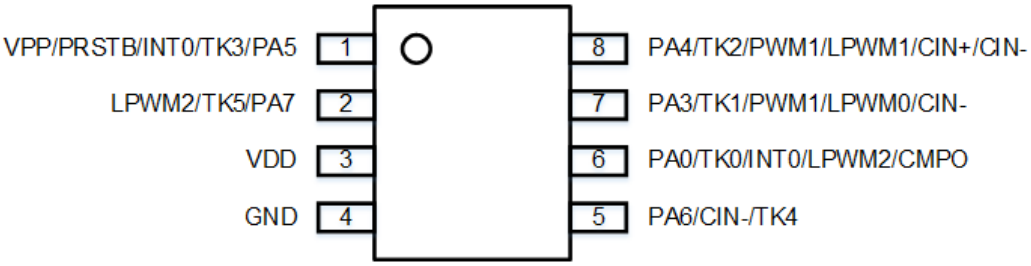


图 1：PMS160 系统方框图

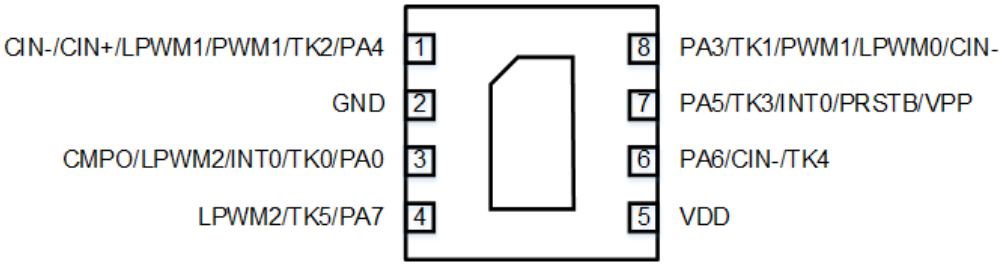
3. 引脚功能说明



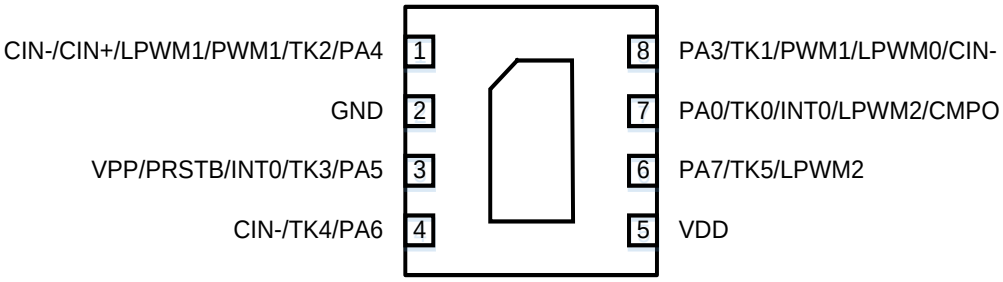
PMS160-S08A: SOP8 (150mil)



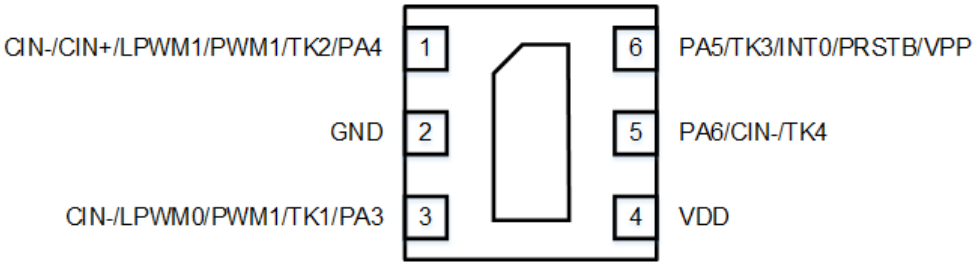
PMS160-S08B: SOP8 (150mil)



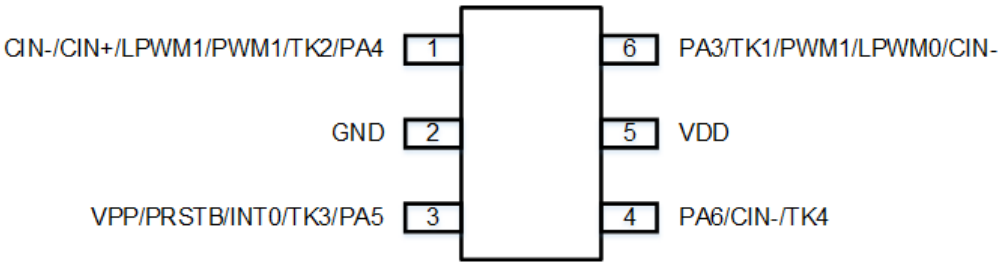
PMS160-2N08A: DFN (2*2mm)



PMS160-2N08B: DFN (2*2mm)



PMS160-2N06: DFN (2*2mm)



PMS160-U06A: SOT23-6 (60mil)

引脚名称	引脚& 缓存器类型	描述
PA6 / TK4 / CIN-	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 A 位 6。可程序设计设定为输入或输出，弱上拉/下拉电阻模式。 (2) 触摸按键 4。 (3) 比较器的负输入端口。 该引脚可以配置为模拟输入，为减少漏电流，请用 ***padier*** 寄存器位 6 关闭其数字输入功能。
PA5 / TK3 / INT0 / PRSTB / VPP	IO ST / CMOS	此引脚可以用作： (1) 端口 A 位 5。可程序设计设定为输入或输出，弱上拉/下拉电阻模式。 (2) 触摸按键 3。 (3) 外部中断源 0。上升沿和下降沿都可触发中断。 (4) 外部复位引脚。 (5) 烧录时作为 VPP 引脚。 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 / TK2 / PWM1 / LPWM1 / CIN+ / CIN-	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 A 位 4。可程序设计设定为输入或输出，弱上拉/下拉电阻模式。 (2) 触摸按键 2。 (3) Timer2 的 PWM 输出。 (4) LPWM 输出通道 1 (5) 比较器的正输入源。 (6) 比较器的负输入源。 该引脚可以配置为模拟输入，为减少漏电流，请用 ***padier*** 寄存器位 4 关闭其数字输入功能。

引脚名称	引脚& 缓存器类型	描述
PA3 / TK1 / PWM1 / LPWM0 / CIN-	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 A 位 3。可程序设计设定为输入或输出，弱上拉/下拉电阻模式。 (2) 触摸按键 1 (3) Timer2 的 PWM 输出。 (4) LPWM 输出通道 0。 (5) 比较器的负输入源。 该引脚可以配置为模拟输入，为减少漏电流，请用 ***padier*** 寄存器位 3 关闭其数字输入功能。
PA0 / TK0 / INT0 / LPWM2 / CMPO	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 A 位 0。可程序设计设定为输入或输出，弱上拉/下拉电阻模式。 (2) 触摸按键 0。 (3) 外部中断源 0。上升沿和下降沿都可触发中断。 (4) LPWM 输出通道 2。 (5) 比较器输出。 该引脚可以配置为模拟输入，为减少漏电流，请用 ***padier*** 寄存器位 0 关闭其数字输入功能。
PA7 / TK5 / LPWM2	IO ST / CMOS	此引脚可以用作： (1) 端口 A 位 7。可程序设计设定为输入或输出，弱上拉/下拉电阻模式。 (2) 触摸按键 5。 (3) LPWM 输出通道 2。 该引脚可以配置为模拟输入，为减少漏电流，请用 ***padier*** 寄存器位 7 关闭其数字输入功能。
VDD	VDD	VDD：数字正电源
GND	GND	GND：数字负电源
注意：IO：输入/输出；**ST**：施密特触发器输入；**Analog**：模拟输入引脚；**CMOS**：CMOS 电压基准位		

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 V_{DD}=5V，f_{SYS}=2MHz 之条件下获得。.

符号	描述	最小值	典型值	最大值	单位	条件
V_{DD}	工作电压	2.0#		5.5	V	# 受限于 LVR 公差
LVR%	低电压复位公差	-5		5	%	
f_{SYS}	系统时钟 (CLK)* = IHRC/2 IHRC/4 IHRC/8 ILRC	 0 0 0	 46K	 8M 4M 2M	 Hz	 V_{DD} ≧ 3.0V，No IFC Touch V_{DD} ≧ 2.5V，No IFC Touch V_{DD} ≧ 2.0V，No IFC Touch V_{DD} =5V
	IHRC/16	0		1M	Hz	IFC Touch 转换最大系统频率
V_{POR}	上电复位电压		1.8		V	
I_{OP}	工作电流		0.6 37		mA uA	f_{SYS}=IHRC/16=1MIPS@5.0V f_{SYS}=ILRC=46KHz@5.0V
I_{PD}	掉电模式消耗电流 （使用 ***stopsys*** 命令）		0.2 0.12		uA	V_{DD} =5V V_{DD} =3.3V
I_{PS}	省电模式消耗电流 （使用 ***stopexe*** 命令） *停用 IHRC		2.3		uA	V_{DD} =5V
V_{IL}	输入低电压	0		0.1 V_{DD}	V	
V_{IH}	输入高电压	0.7 V_{DD}		V_{DD}	V	
I_{OL}	IO 输出灌电流（正常输出）		18		mA	V_{DD}=5.0V, V_{OL}=0.5V
I_{OH}	IO 输出驱动电流（正常输出）		13		mA	V_{DD}=5.0V, V_{OH}=4.5V
V_{IN}	输入电压	-0.3		V_{DD}+0.3	V	
$I_{INJ\ (PIN)}$	引脚注入电流		1		uA	V_{DD} +0.3≧V_{IN}≧ -0.3
R_{PH}	上拉电阻		71		KΩ	V_{DD}=5.0V
R_{PL}	下拉电阻		64		KΩ	V_{DD}=5.0V
f_{IHRC}	校准后 IHRC 频率 *	15.76*	16*	16.24*	MHz	25°C, V_{DD} =2.2V~5.5V
		15.20*	16*	16.80*		V_{DD} =2.2V~5.5V, -20°C <Ta<70°C*
		14.60*	16*	17.40*		V_{DD} =2.0V~5.5V, -20°C <Ta<70°C
f_{ILRC}	ILRC 频率 *		46		KHz	V_{DD} =5V
f_{NILRC}	NILRC 频率*		14		KHz	V_{DD} = 5.0V
t_{INT}	中断脉冲宽度	30			ns	V_{DD} = 5.0V
t_{WDT}	看门狗超时溢出时间		8192		ILRC clock period	misc[1:0]=00（默认）
			16384			misc[1:0]=01
			65536			misc[1:0]=10
			262144			misc[1:0]=11

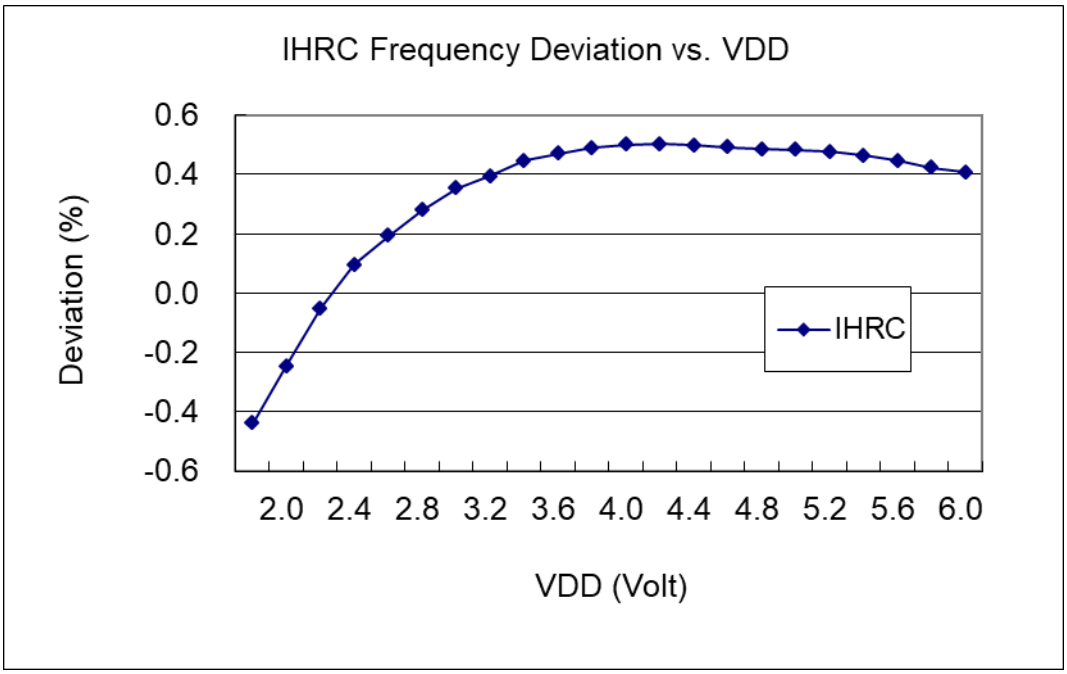
符号	描述	最小值	典型值	最大值	单位	条件
t_{WUP}	快速唤醒时间		8		T_{ILRC}	T_{ILRC} 是 ILRC 时钟周期
	正常唤醒时间		16			
t_{SBP}	系统正常开机时间		43		ms	@ V_{DD} =5V
	系统快速时间		720		us	@ V_{DD} =5V
t_{RST}	外部复位脉冲宽度	120			us	@ V_{DD} =5V

*这些参数是设计参考值，并不是每个芯片测试。
*特性图是实际测量值。考虑到生产飘移等因素的影响，表格中的数据是在实际测量值的安全范围内。

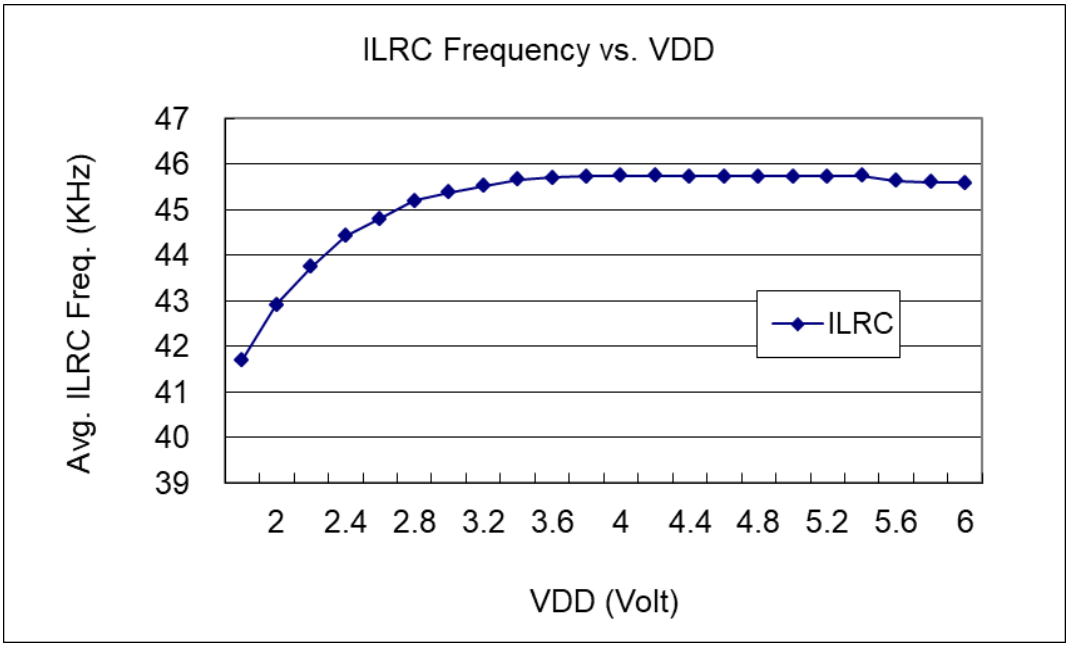
4.2. 绝对最大值范围

- 电源电压 ..... 2.0V ~ 5.5V（最大值：5.5V）
  *最大电压不能超过 5.5V，否则会损坏 IC。
- 输入电压 ..... -0.3V ~ V_{DD} + 0.3V
- 工作温度 ..... -20°C ~ 70°C
- 存储温度..... -50°C ~ 125°C
- 结点温度 ..... 150°C

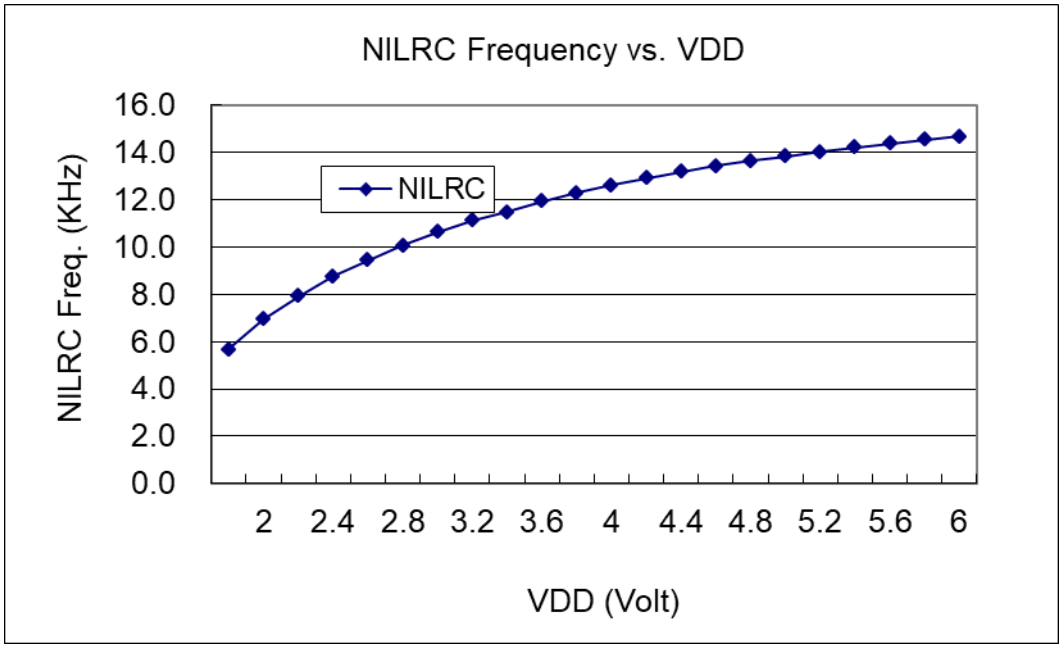
4.3. IHRC 频率与 VDD 关系曲线图（校准到 16MHz）



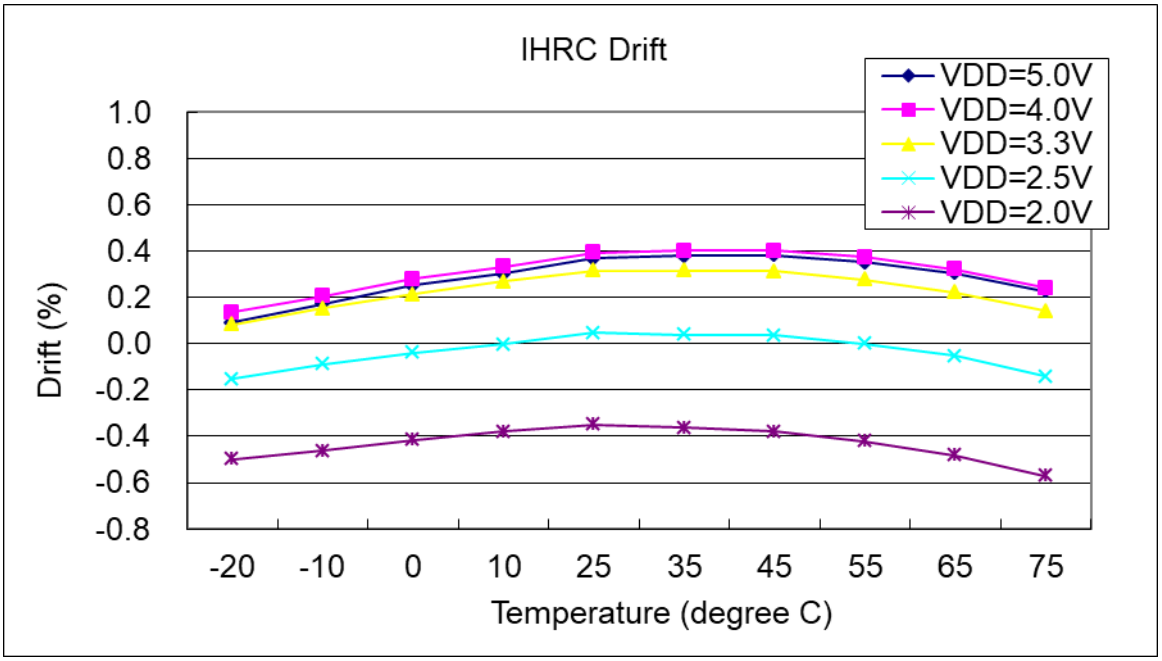
4.4. ILRC 频率与 VDD 关系曲线图



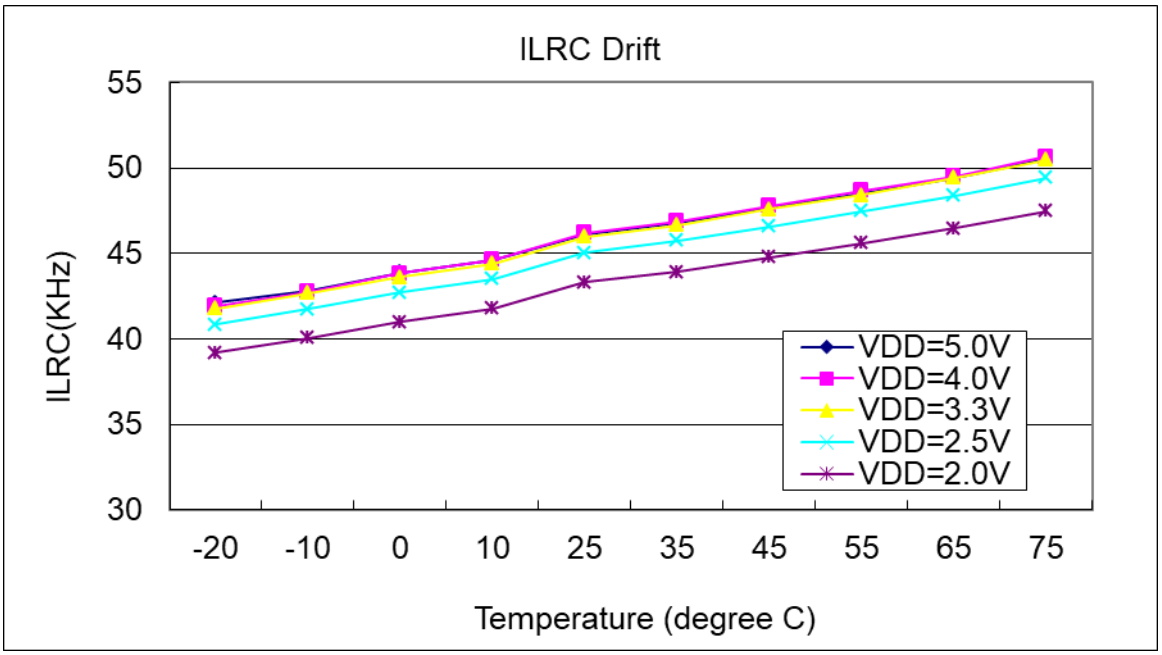
4.5. NILRC 频率与 VDD 关系曲线图



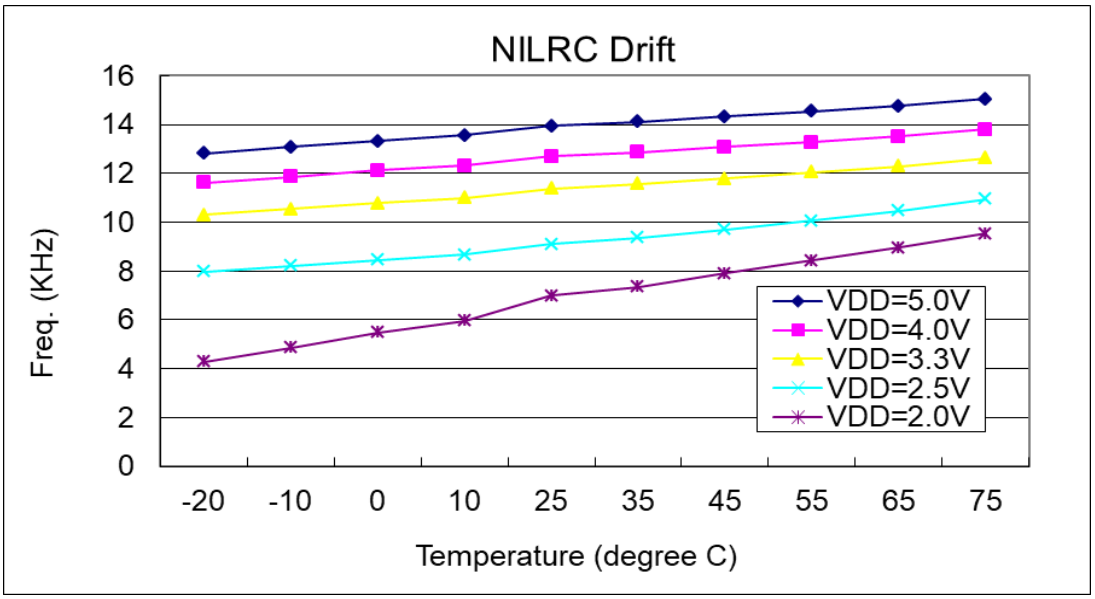
4.6. IHRC 频率与温度关系曲线图（校准到 16MHz）



4.7. ILRC 频率与温度关系曲线图



4.8. NILRC 频率与温度关系曲线图

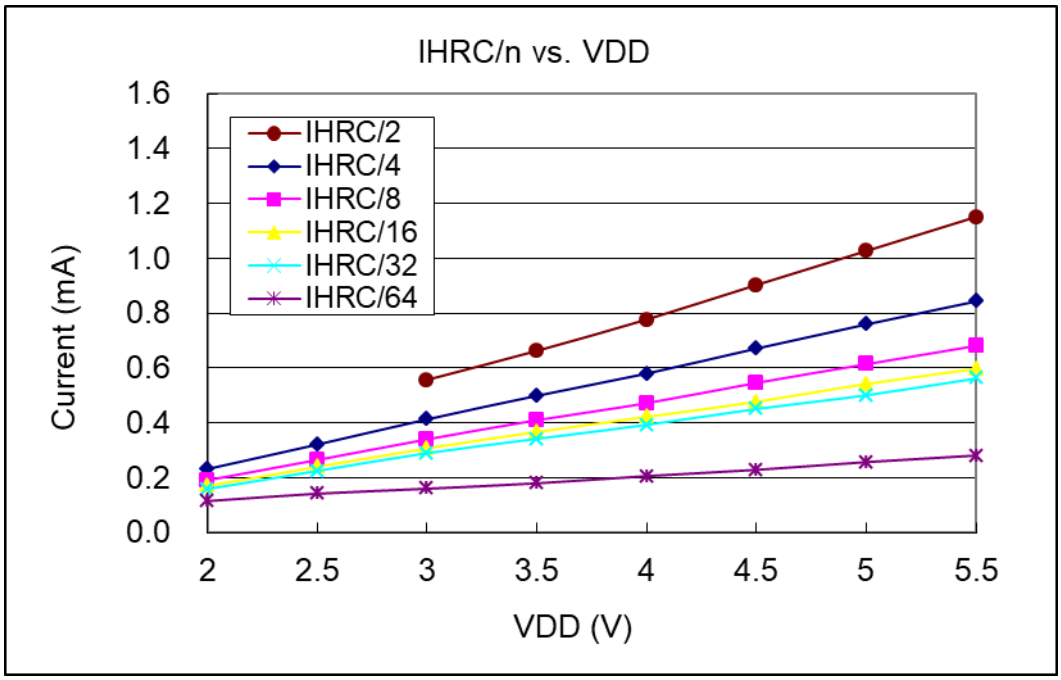


4.9. 工作电流 vs. VDD 与系统时钟 = IHRC/n 关系曲线图

- 条件：

  pa0 间隔(1s)高低电平翻转。**ON**: Bandgap, LVR, IHRC

  停用：t16 定时器，中断，ILRC，**触摸功能**，且 IO 引脚不悬空。

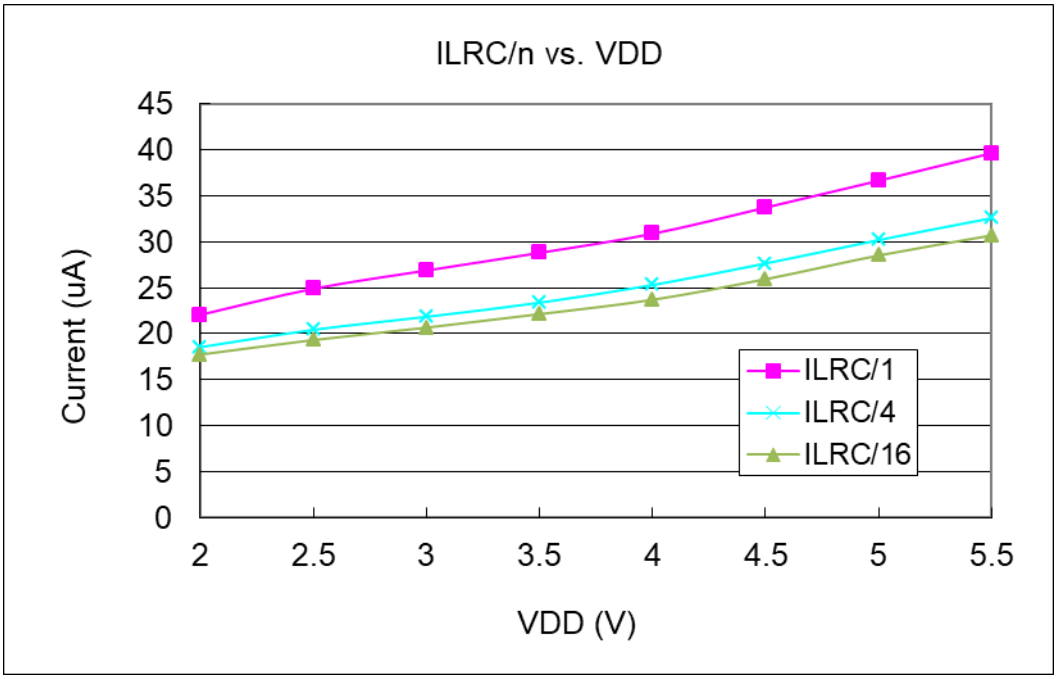


4.10. 工作电流 vs. VDD 与系统时钟 = ILRC/n 关系曲线图

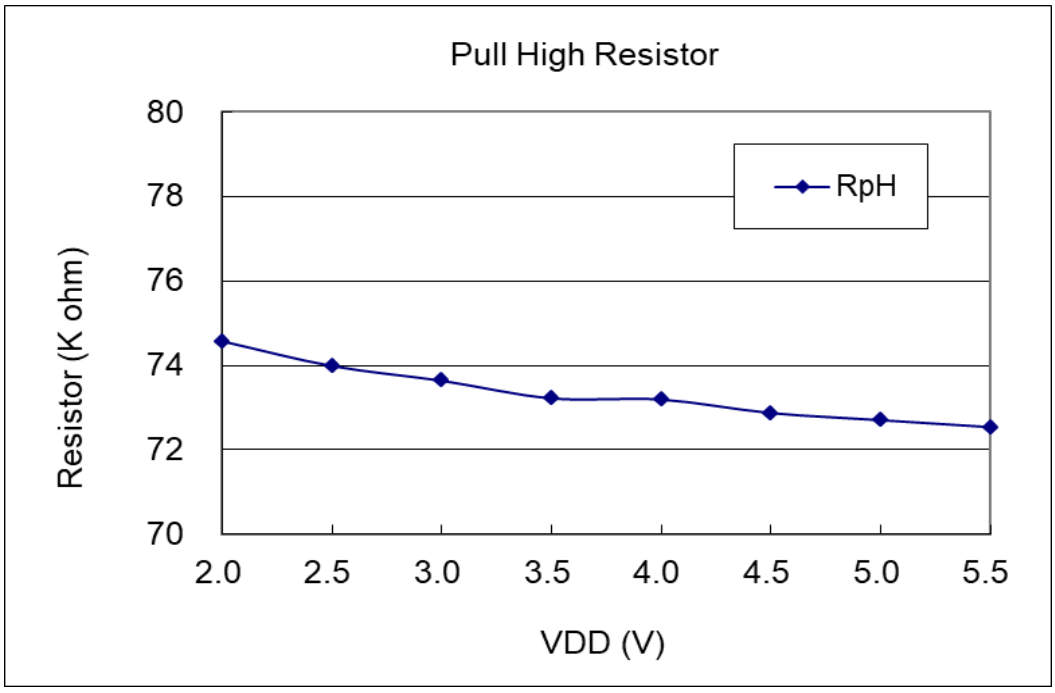
- 条件：

  pa0 间隔(1s)高低电平翻转。**启用**：Bandgap，LVR，IHRC。

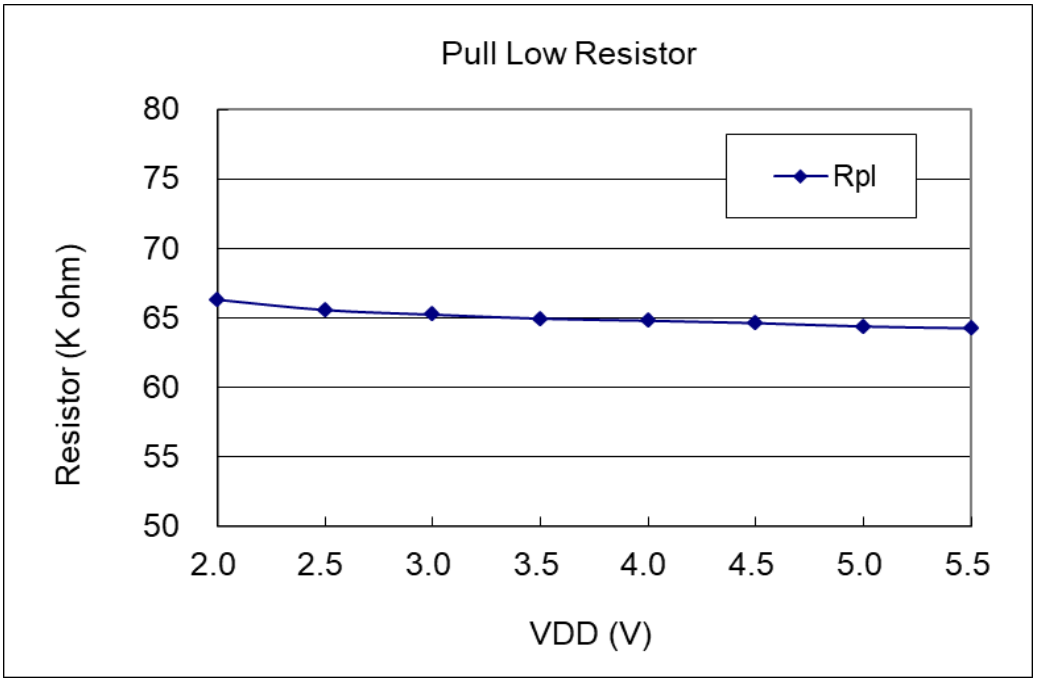
  停用：t16 定时器，中断，ILRC，**触摸功能**，且 IO 引脚不悬空。



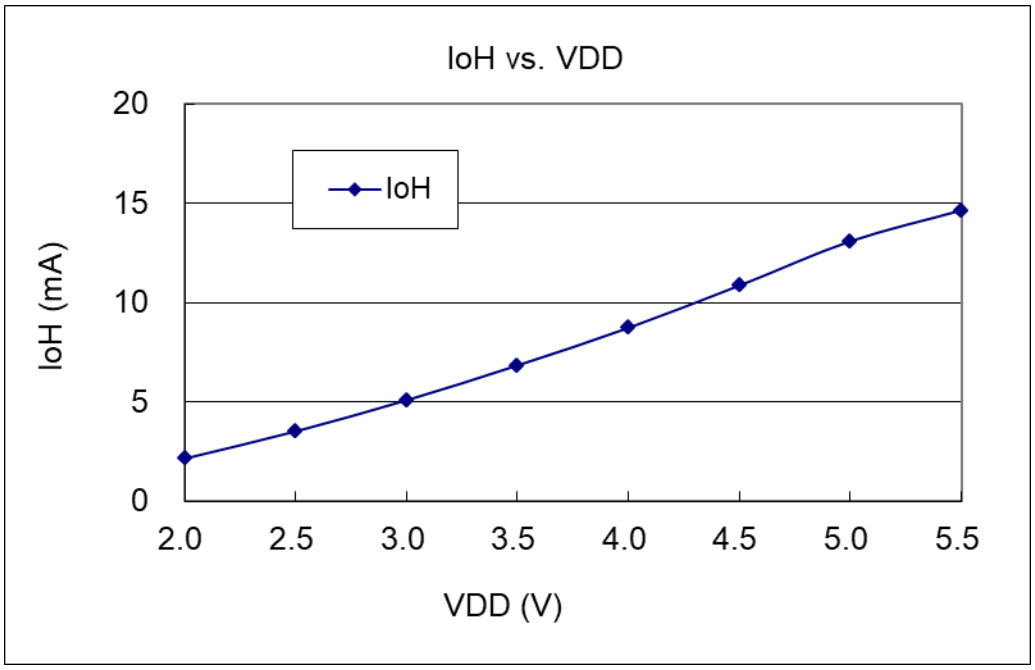
4.11. IO 引脚上拉阻抗曲线图

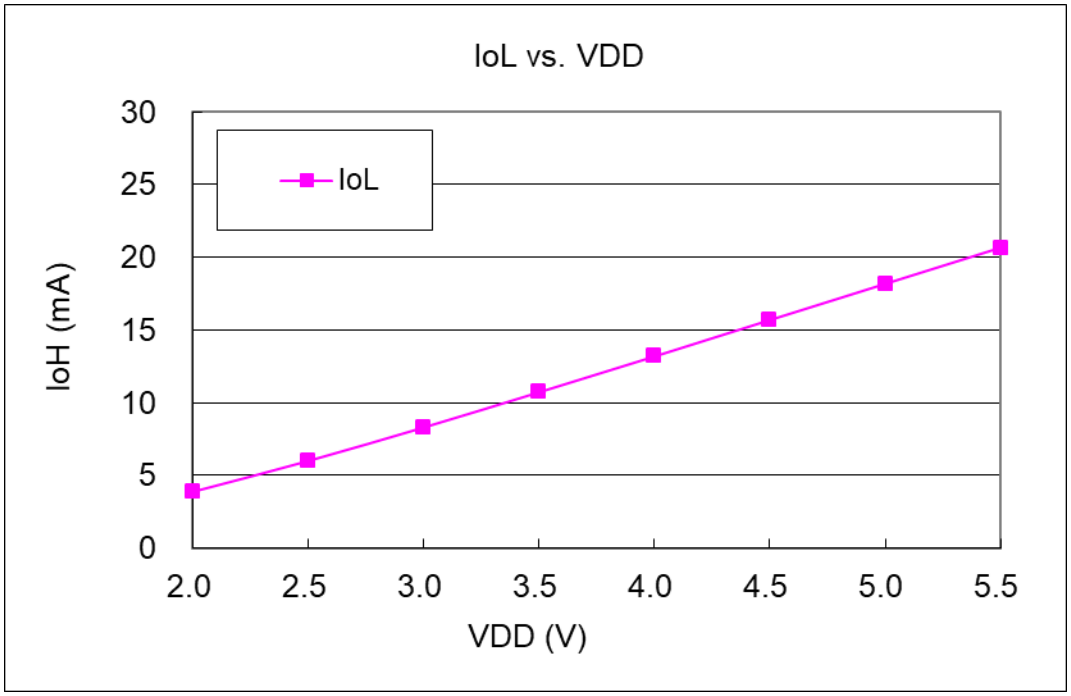


4.12. IO 引脚下拉阻抗曲线图

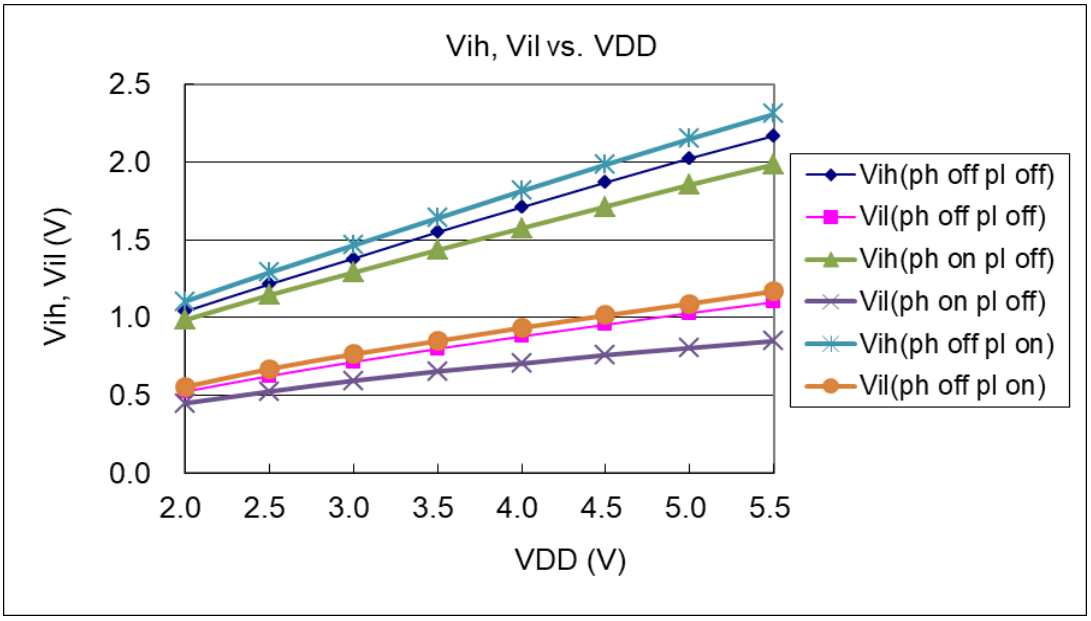


4.13. IO 引脚输出的驱动电流(I_{OH})与灌电流(I_{OL})曲线图
（VOH=0.9*VDD, VOL=0.1*VDD）

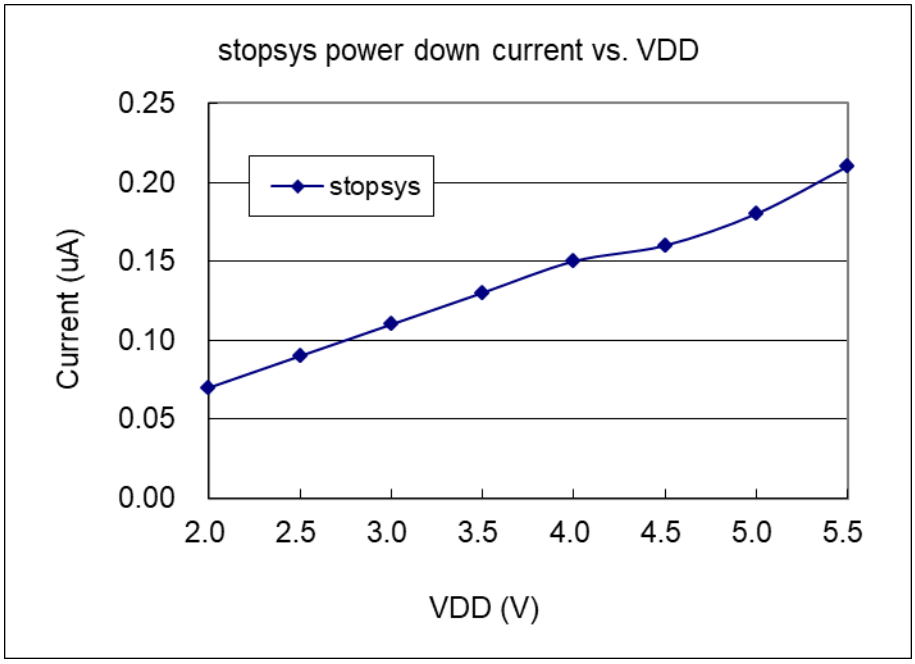


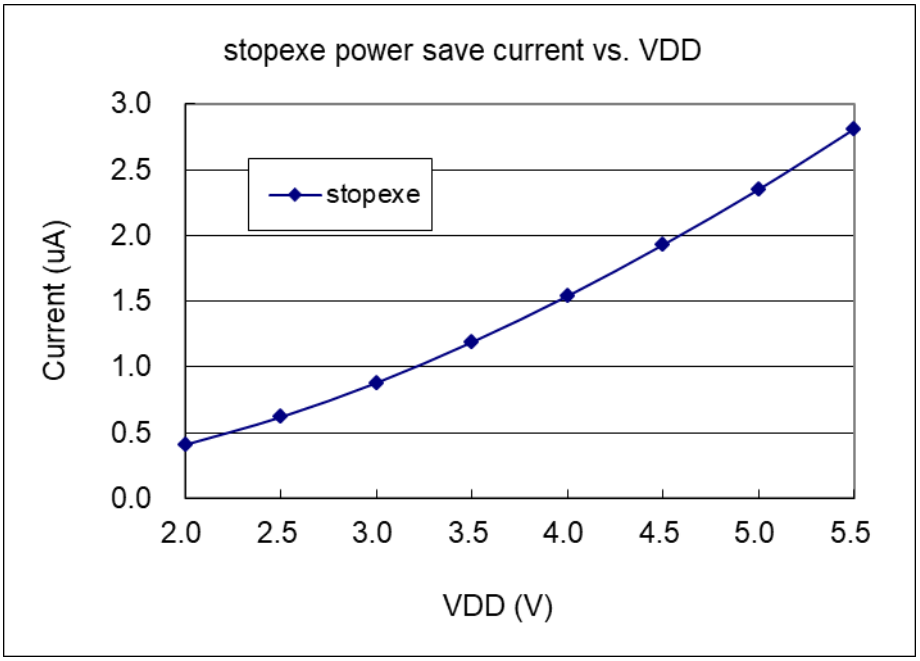


4.14. IO 引脚输入高/低阈值电压(V_{IH}/V_{IL})曲线图



4.15. 掉电电流(I_{PD})和省电电流(I_{PS})





5. 功能概述

5.1. 程序内存 – OTP

OTP（一次性可程序设计）程序内存用来存放要执行的程序指令。OTP 程序内存可以储存数据，包含：数据，表格和中断入口。复位之后，FPP0 的初始地址为 0x000 保留给系统使用，中断入口是 0x010。PMS160 的 OTP 程序内存容量为 1.5KW 如表 1 所示。OTP 内存从地址“0x5F0 ~0x5FF”供系统使用，从 0x001 到 0x00F 和从 0x011 到 0x5EF 地址空间是用户的程序空间。

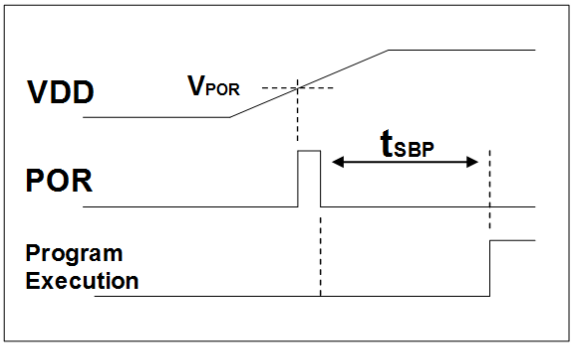
地址	功能
0x000	用于 FPP0 复位，goto 主程序
0x001	用户程序区
•	•
•	•
0x00F	用户程序区
0x010	中断入口地址
0x011	用户程序区
•	•
0x5EF	用户程序区
0x5F0	系统使用
•	•
0x5FF	系统使用

表 1：程序内存结构

5.2. 启动程序

开机时，POR（上电复位）是用于复位 PMS160。开机时间可选快速开机或正常开机。不管用哪种开机模式，用户在使用时都必须确保上电后电源电压稳定。开机时序如图 2 所示，其中 t_{SBP} 是开机时间。

注意，上电复位(Power-On Reset)时，V_{DD} 必须先超过 V_{POR} 电压，MCU 才会进入开机状态。



Boot up from Power-On Reset

图 2：上电时序

5.3. 数据存储器 – SRAM

数据存储可以是字节或位操作。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，以及堆栈内存。

堆栈定义在数据存储器里面，堆栈指针定义在堆栈指针寄存器，用户可在使用时自行定义堆栈深度，堆栈内存对堆栈的排列是非常灵活的，用户可以动态调整堆栈。

对于间接存储指令而言，数据存储器可以用作数据指针来当作数据地址。所有的数据存储器都可以当作资料指针，这对于间接存储指令是相当灵活和有效的。由于数据宽度是 8 位，PMS160 的所有 96 字节的数据存储器都可以利用间接存取指令做存取。

5.4. 振荡器和时钟

PMS160 有三个振荡器电路：内部高频 RC 振荡器(IHRC) 、内部低频振荡器(ILRC)和内部超低频振荡器(NILRC)。前两个振荡器可以分别通过寄存器 clkmd.4 和 clkmd.2 来启用或停用，NILRC 振荡器保持默认开启的状态。用户可以选择不同的振荡器（IHRC 或 ILRC）作为系统时钟源，同时可以通过设置 clkmd 寄存器来满足不同的应用要求。

振荡器模块	启用/停用
IHRC	**clkmd**.4
ILRC	**clkmd**.2
NILRC	**tm3c.0**

表 2：振荡器模块

5.4.1. 内部高频 RC 振荡器和内部低频 RC 振荡器

开机后，IHRC、ILRC 以及 NILRC 振荡器是自动启用的。IHRC 频率能通过 ***ihrcr*** 寄存器校准，通常校准到 16 MHz。校准后的频率偏差通常在 1%以内；且校准后 IHRC 的频率仍然会因电源电压和工作温度而略有漂移。请参阅 IHRC 频率和 VDD、温度的测量图表。

ILRC 的频率会因生产工艺，使用的电源电压和温度的差异而产生漂移，请参考直流电气特性规格数据，建议不要应用在要求精准时序的产品上。

NILRC 振荡器是比 ILRC 更慢的时钟，用来做更省电的唤醒时钟。NILRC 和 ILRC 都可通过 IHRC 估算频率，但 NILRC 的误差更大，所以需要先估算频率才可使用。若需相关 demo，请洽 FAE。

5.4.2. IHRC 校准

在芯片生产制造时，每颗芯片的 IHRC 频率都有可能稍微不同，PMS160 提供 IHRC 频率校准来消除这些差异，校准功能可以被用户的程序选择并编译，同时这个命令会自动嵌入用户的程序里面。

校准命令如下所示：

.ADJUST_IC SYSCLK=IHRC/(**p1**), IHRC=(**p2**)MHz, V_{DD}=(**p3**)V

p1=4, 8, 16, 32; 用以提供不同的系统时钟。

p2=14 ~ 18; 用以校准芯片到不同的频率，16MHz 是通用的选择。

p3=2.3 ~ 5.5; 用以在不同的工作电压下校准频率。

5.4.3. IHRC 频率校准和系统时钟

在用户编译程序时，IHRC 频率校准和系统时钟的选项如表 3 所示：

SYSCLK	CLKMD	IHRCR	Description
○ Set IHRC / 4	= 14h (IHRC / 4)	有校准	IHRC 校准到 16MHz, CLK=4MHz (IHRC/4)
○ Set IHRC / 8	= 3Ch (IHRC / 8)	有校准	IHRC 校准到 16MHz, CLK=2MHz (IHRC/8)
○ Set IHRC / 16	= 1Ch (IHRC / 16)	有校准	IHRC 校准到 16MHz, CLK=1MHz (IHRC/16)
○ Set IHRC / 32	= 7Ch (IHRC / 32)	有校准	IHRC 校准到 16MHz, CLK=0.5MHz (IHRC/32)
○ Set ILRC	= E4h (ILRC / 1)	有校准	IHRC 校准到 16MHz, CLK=ILRC
○ Disable	不改变	没改变	IHRC 不校准，CLK 不改变

表 3：IHRC 频率校准选项

通常，.ADJUST_IC 是开机后第一条指令，以便系统开机后能设定系统频率，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。如果用户选择了不同的频率校准选项，PMS160 的系统状态在开机后也会不同。以下所示为不同的选项开机后，PMS160 执行此命令后的状态：

(1) .ADJUST_ICSYSCLK=IHRC/4, IHRC=16MHz, VDD=3.3V

开机后，CLKMD = 0x14：

- IHRC 频率在 V_{DD}=3.3V 时校准到 16MHz，并且 IHRC 模块是启用的
- 系统时钟= IHRC/4 = 4MHz
- 看门狗计数器停用，ILRC 启用，PA5 引脚是输入模式

(2) .ADJUST_IC SYSCLK=IHRC/8, IHRC=16MHz, V_{DD}=2.5V

开机后，CLKMD = 0x3C：

- IHRC 频率在 V_{DD}=2.5V 时校准到 16MHz，并且 IHRC 模块是启用的
- 系统时钟= IHRC/8 = 2MHz
- 看门狗计数器停用，ILRC 启用，PA5 引脚是输入模式

(3) .ADJUST_ICSYSCLK=IHRC/16, IHRC=16MHz, V_{DD}=2.3V

开机后，CLKMD = 0x1C：

- IHRC 频率在 V_{DD}=2.3V 时校准到 16MHz，并且 IHRC 模块是启用的
- 系统时钟= IHRC/16 = 1MHz
- 看门狗计数器停用，ILRC 启用，PA5 引脚是输入模式

(4) .ADJUST_ICSYSCLK=IHRC/32, IHRC=16MHz, V_{DD}=5V

开机后，CLKMD = 0x7C：

- IHRC 频率在 V_{DD}=5V 时校准到 16MHz，并且 IHRC 模块是启用的
- 系统时钟= IHRC/32 = 500kHz
- 看门狗计数器停用，ILRC 启用，PA5 引脚是输入模式

(5) .ADJUST_ICSYSCLK=ILRC, IHRC=16MHz, V_{DD}=5V

开机后，CLKMD = 0XE4：

- IHRC 频率在 V_{DD}=5V 时校准到 16MHz，并且 IHRC 模块是停用的
- 系统时钟 = ILRC
- 看门狗计数器停用，ILRC 启用，PA5 引脚是输入模式

(6) .ADJUST_IC DISABLE

开机后，CLKMD 寄存器没有改变（没任何动作）：

- IHRC 没有校准并且 IHRC 模块启用或停用可通过 Code Option 中 Boot-up_Time 选择
- 系统频率= ILRC 或 IHRC/6（通过 Code Option 中 Boot-up_Time 选择）
- 看门狗计数器启用，ILRC 启用，PA5 引脚是输入模式。

5.4.4. 系统时钟和 LVR 基准位

系统时钟来自 IHRC 或者 ILRC，PMS160 的时钟系统的硬件框图，如图 3 所示：

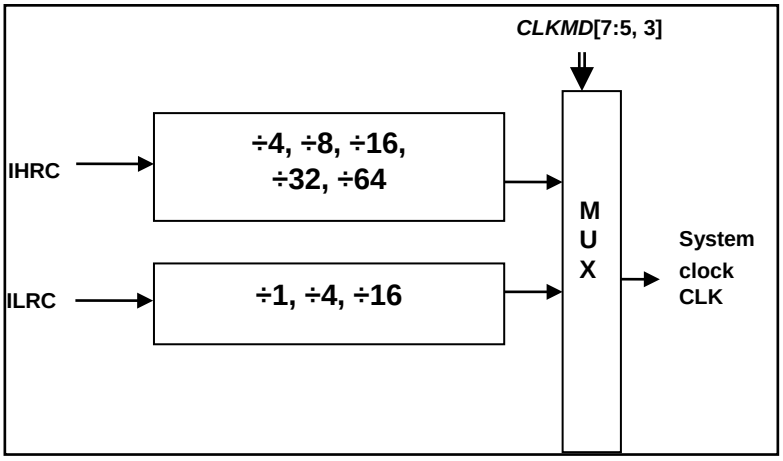


图 3：系统时钟选项

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 LVR 的基准位结合起来才能使系统稳定。LVR 的基准位是在编译过程中选择，不同系统时钟对应的 LVR 设定，请参考章节 4.1 中系统时钟的最低工作电压。

5.4.5. 系统时钟切换

IHRC 校准后，用户可能要求切换系统时钟到新的频率或者可能会随时切换系统时钟来优化系统性能及功耗。基本上，PMS160 的系统时钟能够随时通过设定寄存器 ***clkmd*** 在 IHRC 和 ILRC 之间切换。在设定寄存器 ***clkmd*** 之后，系统时钟立即转换成新的频率。**请注意，在下命令给 *clkmd* 寄存器时，不能同时关闭原来的时钟模块。**请参考以下例子。

例 1： 系统时钟从 ILRC 切换到 IHRC/8

```
...                             //  系统时钟是 ILRC
CLKMD.4     =   1;              //  先打开 IHRC，可以提高抗干扰能力
CLKMD       =   0x3C;           //  切换到 IHRC/8，ILRC 不能在这里停用
//  CLKMD.2 =   0;              //  假如需要，ILRC 可以在这里停用
...
```

例 2： 系统时钟从 IHRC/8 切换到 ILRC

```
...                             //  系统时钟是 IHRC/8
CLKMD       =   0xF4;           //  切换到 ILRC，IHRC 不能在这里停用
CLKMD.4     =   0;              //  IHRC 可以在这里停用
...
```

例 3： 系统时钟从 IHRC/8 切换到 IHRC/32

```
...                             //  系统时钟是 IHRC/8，ILRC 在这里是启用的
CLKMD       =   0X7C;           //  切换到 IHRC/32
...
```

例 4： 如果同时切换系统时钟关闭原来的振荡器，系统会当机

```
...                             //  系统时钟是 ILRC
CLKMD       =   0x30;           //  不能从 ILRC 切换到 IHRC/8 同时关闭 ILRC 振荡器
```

5.5. 比较器

PMS160 内置一个硬件比较器，如图 4 所示比较器硬件原理框图。它可以比较两个引脚之间的信号或者与内部参考电压 $V_{internal\ R}$ 或者与内置 bandgap (1.2v)做比较。两个信号进行比较，一个是正输入，另一个是负输入。比较器的负输入可以是 PA3，PA4，内置 bandgap (1.2v)，PA6 或者内部参考电压 $V_{internal\ R}$，并由寄存器 gpcc 的[3:1]位来选择。比较器的正输入可以是 PA4 或者 $V_{internal\ R}$，并由 gpcc 寄存器的位 0 来选择。

比较器输出的结果可以用 gpcs.7 选择性的送到 PA0，此时无论 PA0 是输入还是输出状态，比较器结果都会被强制输出；输出结果信号可以是直接输出，或是通过 Time2 从定时器时钟模块(TM2_CLK)采样。另外，信号是否反极性也可由 gpcc.4 选择。比较输出结果可以用来产生中断信号或通过 gpcc.6 读取出来。

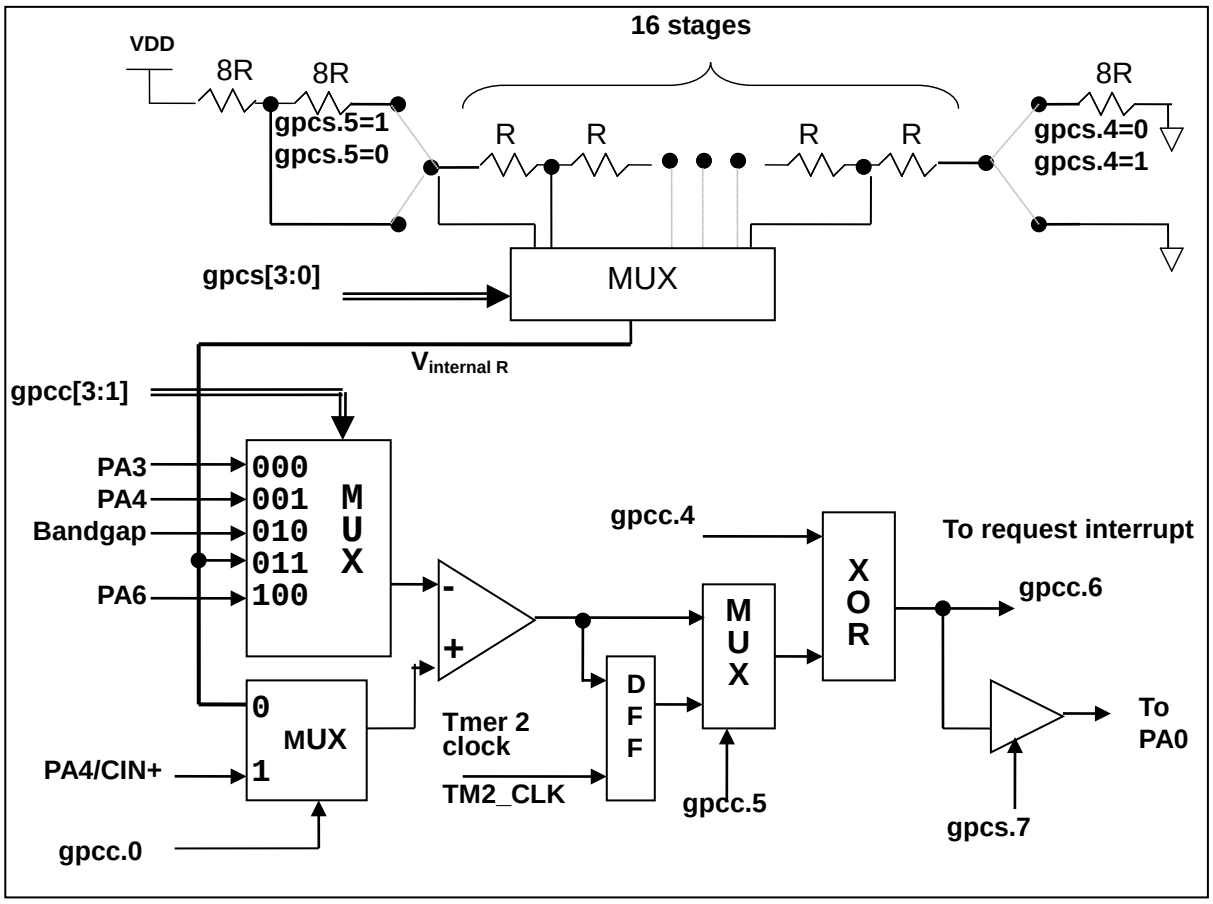


图 4：比较器硬件原理框图

5.5.1. 内部参考电压 ($V_{internal\ R}$)

内部参考电压 $V_{internal\ R}$ 由一连串电阻所组成，可以产生不同层次的参考电压，***gpcs*** 寄存器的位 4 和位 5 是用来选择 $V_{internal\ R}$ 的最高和最低值，位[3:0]用于选择所要的电压水平，这电压水平是由 $V_{internal\ R}$ 的最高和最低值均分 16 等份，由位[3:0]选择出来。图 5 ~ 图 8 显示四个条件下有不同的参考电压 $V_{internal\ R}$。内部参考电压 $V_{internal\ R}$ 可以通过 gpcs 寄存器来设置，范围从(1/32)*V_{DD} 到(3/4)*V_{DD}。

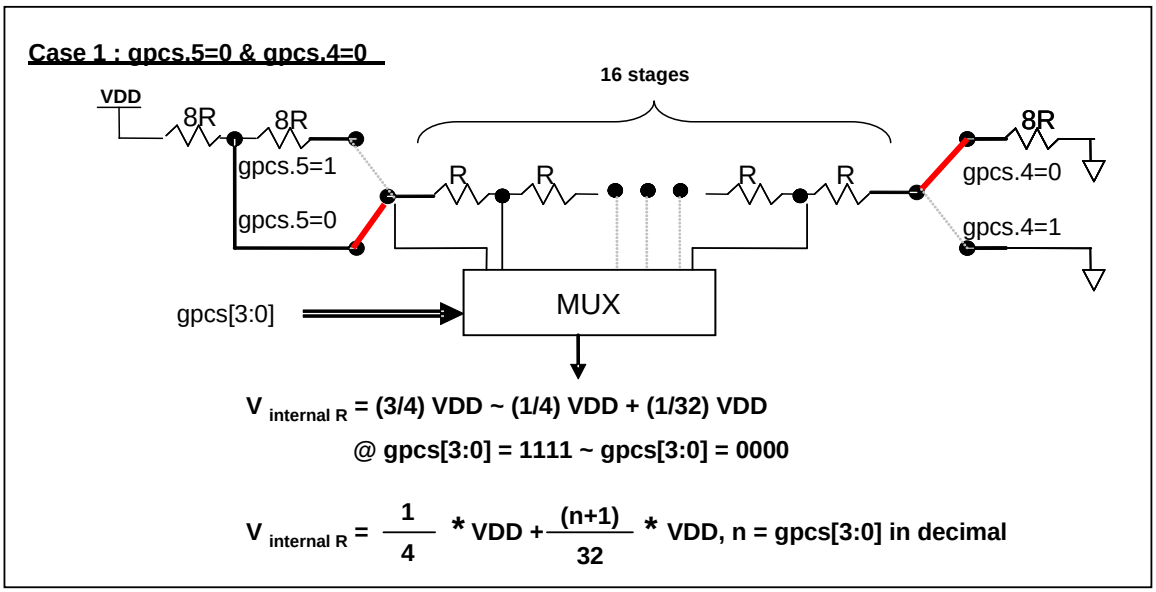


图 5：$V_{internal\ R}$ 硬件接法 (gpcs.5=0 & gpcs.4=0)

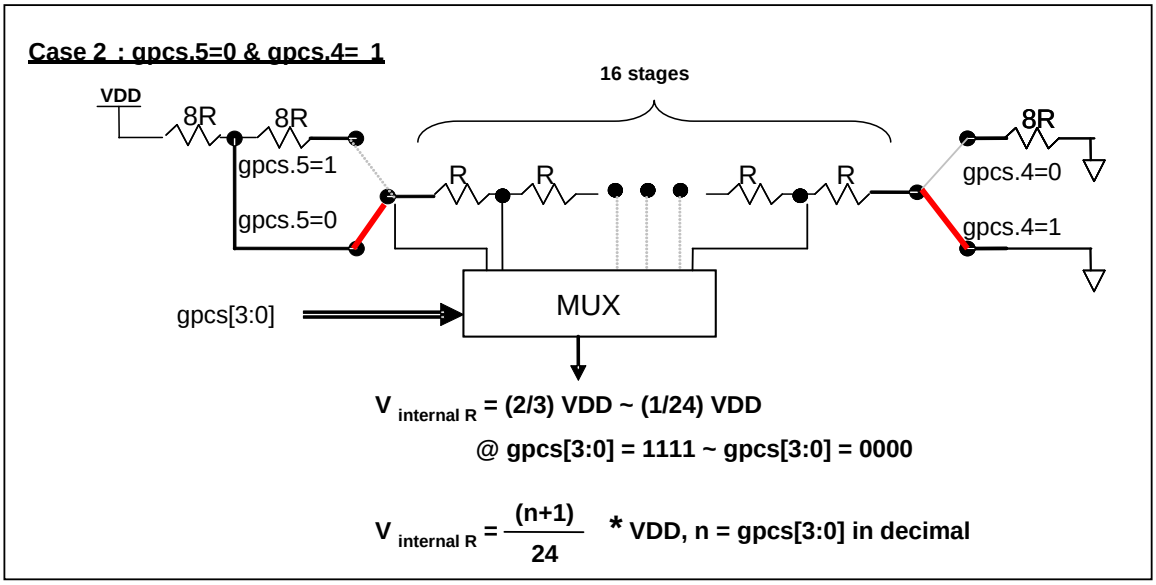


图 6：$V_{internal\ R}$ 硬件接法 (gpcs.5=0 & gpcs.4=1)

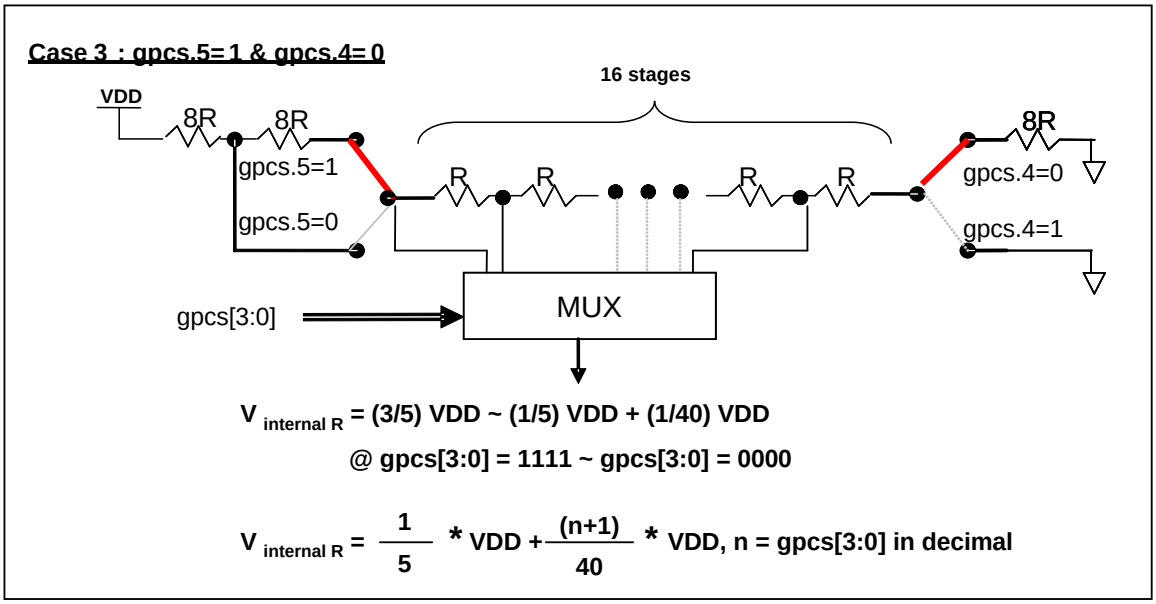


图 7：$V_{internal\ R}$ 硬件接法 (gpcs.5=1 & gpcs.4=0)

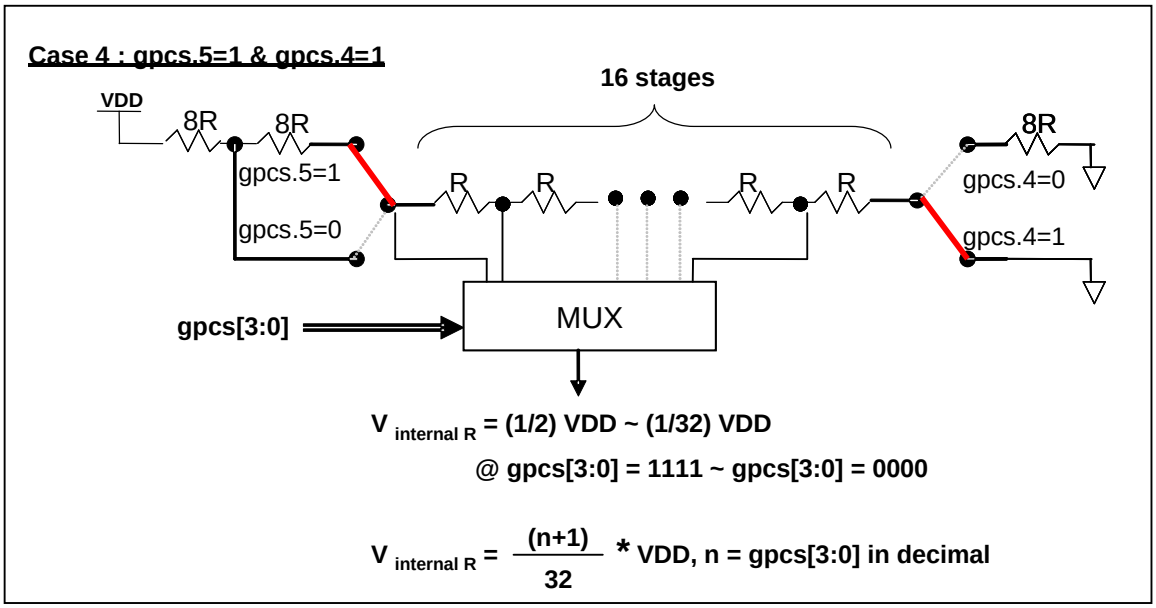


图 8：$V_{internal\ R}$ 硬件接法 (gpcs.5=1 & gpcs.4=1)

5.5.2. 使用比较器

例 1：

选择 PA3 为负输入和 Vinternal R 的电压为(18/32)*VDD 作为正输入。Vinternal R 选择上图 gpcs[5:4] = 2b'00 的配置方式，gpcs [3:0] = 4b'1001 (n=9)以得到 Vinternal R = (1/4)*VDD + [(9+1)/32]*VDD = [(9+9)/32]*VDD = (18/32)*VDD 的参考电压。

```
gpcs    = 0b1_0_00_1001;        // V_internal R = V_DD*(18/32)
gpcc    = 0b1_0_0_0_000_0;      // 启用比较器，负输入：PA3，正输入：V_internal R
padier = 0bxxxx_0_xxx;          // 停用 PA3 数字输入防止漏电（x：由客户自定）
```

或

```
$ GPCS     V_DD*18/32;
$ GPCC Enable, N_PA3, P_R;      //  N_xx 是负输入，P_R 代表正输入是内部参考电压
PADIER = 0bxxxx_0_xxx;
```

例 2：

选择 Vinternal R 为负输入，Vinternal R 的电压为(22/40)*VDD，选择 PA4 为正输入，比较器的结果将反极性并输出到 PA0。Vinternal R 选择上图的配置方式 "gpcs[5:4] = 2b'10" 和 gpcs[3:0] = 4b'1101 (n=13) 得到 Vinternal R = (1/5)*VDD + [(13+1)/40]*VDD = [(13+9)/40]*VDD = (22/40)*VDD。

```
gpcs    = 0b1_0_1_0_1101;       // 输出到 PA0，V_internal R = V_DD*(22/40)
gpcc    = 0b1_0_0_1_011_1;      // 反极性输出，负输入: V_internal R，正输入：PA4
padier = 0bxxx_0_xxxx;          // 停用 PA4 数字输入防止漏电（x：由客户自定）
```

或

```
$ GPCS     Output, V_DD*22/40;
$ GPCC Enable, Inverse, N_R, P_PA4;   // N_R 代表负输入是内部参考电压，P_xx 是正输入
PADIER = 0bxxx_0_xxxx;
```

5.5.3. 使用比较器和 Bandgap 1.20V

内部 Bandgap 参考电压生成器可以提供 1.20V，它可以测量外部电源电压水平。该 Bandgap 参考电压可以选做负输入去和正输入 Vinternal R 比较。Vinternal R 的电源是 VDD，利用调整 Vinternal R 电压水平和 Bandgap 参考电压比较，就可以知道 VDD 的电压。如果 N（**gpcs**[3:0]十进制）是让 Vinternal R 最接近 1.20V，那么 VDD 的电压就可以透过下列公式计算：

对于 Case 1 而言：V_{DD} = [32 / (N+9)] * 1.20 volt；

对于 Case 2 而言：V_{DD} = [24 / (N+1)] * 1.20 volt；

对于 Case 3 而言：V_{DD} = [40 / (N+9)] * 1.20 volt；

对于 Case 4 而言：V_{DD} = [32 / (N+1)] * 1.20 volt；

例一：

```
$ GPCS    VDD*12/40;                  //  4.0V * 12/40 = 1.2V
$ GPCC    Enable, BANDGAP, P_R;       //  BANDGAP 是负输入，P_R 代表正输入是内部参考电压
...
if   (GPC_Out)                        // 或写成 GPCC.6
{                                     //  当 VDD 大于 4V 时
}
else
{                                     //  当 VDD 小于 4V 时
}
```

5.6. 16 位计数器 (Timer16)

PMS160 内置一个 16 位硬件计数器 (Timer16)，计数器时钟可来自于系统时钟 (CLK)，内部高频振荡时钟 (IHRC)，内部低频振荡时钟 (ILRC)，PA4 和 PA0。在送到 16 位计数器之前，1 个可软件程序设计的预分频器提供÷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。

16 位计数器的中断请求可以通过 16 位计数器的位[15:8]来选择，中断类型可以上升沿触发或下降沿触发，定义在寄存器 ***integs.4***。Timer16 模块框图如图 9 所示。

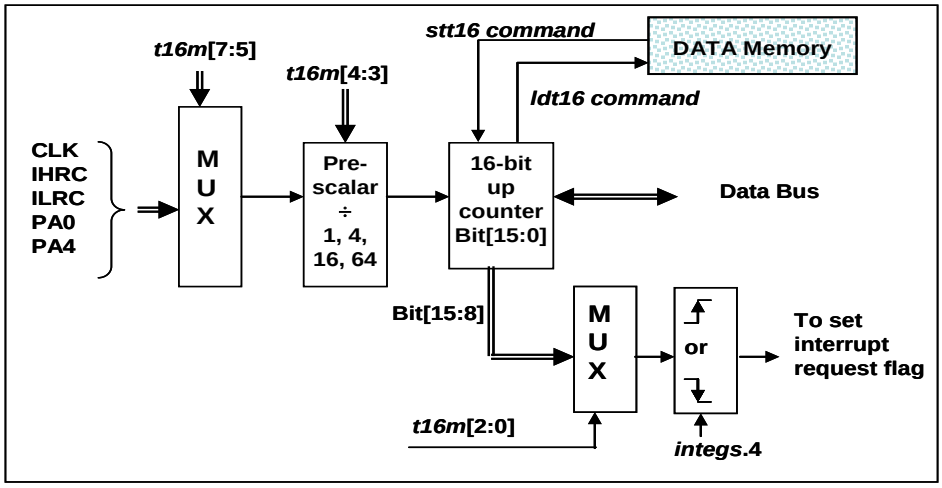


图 9：Timer16 模块框图

当使用 Timer16 时，Timer16 的语法定义在.inc 文件中。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，最后一个参数是定义中断源。详细如下：

```
T16M    IO_RW    0x06
$ 7~5:      STOP, SYSCLK, X, PA4_F, IHRC, X, ILRC, PA0_F      //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
```

用户可以依照系统的要求来定义 T16M 参数，例子如下，更多例子请参考 IDE 软件“说明→ 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
     // 如果系统时钟 System Clock = IHRC / 2 = 8 MHz
     // 则 SYSCLK/64 = 8 MHz/64 = 125kHz(8us)，约每 524 mS 产生一次 INTRQ.2=1

$    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 时钟周期产生一次 INTRQ.2=1

$    T16M    STOP;
     // 停止 Timer16 计数
```

5.7. 看门狗计数器

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

- 当 misc[1:0]=00（默认）时：8k ILRC 时钟周期
- 当 misc[1:0]=01 时：16k ILRC 时钟周期
- 当 misc[1:0]=10 时：64k ILRC 时钟周期
- 当 misc[1:0]=11 时：256k 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。由于在系统重启或者唤醒之后，看门狗计数周期会比预计要短，为防止看门狗计数溢出导致复位，建议在系统重启或唤醒之后使用立即 wdreset 指令清零看门狗计数。

当看门狗超时溢出时，PMS160 将复位并重新运行程序。看门狗时序图如图 10 所示。

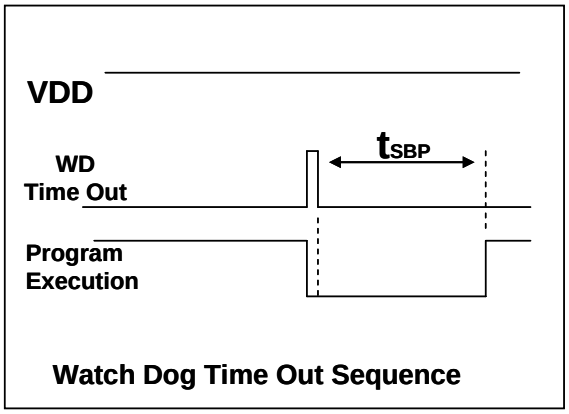


图 10：看门狗超时溢出时序图

5.8. 中断

PMS160 有 6 个中断源：

- 外部中断源 PA0 / PA5
- Timer16 中断
- Timer2 中断
- Timer3 中断
- GPC 中断
- LPWM 中断

每个中断请求源都有自己的中断控制位来启用或停用。中断功能的硬件框图如图 11 所示。所有的中断请求标志位是由硬件置位并且通过软件写寄存器 ***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（停用全局中断）停用它。

中断堆栈与数据存储器共享，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。

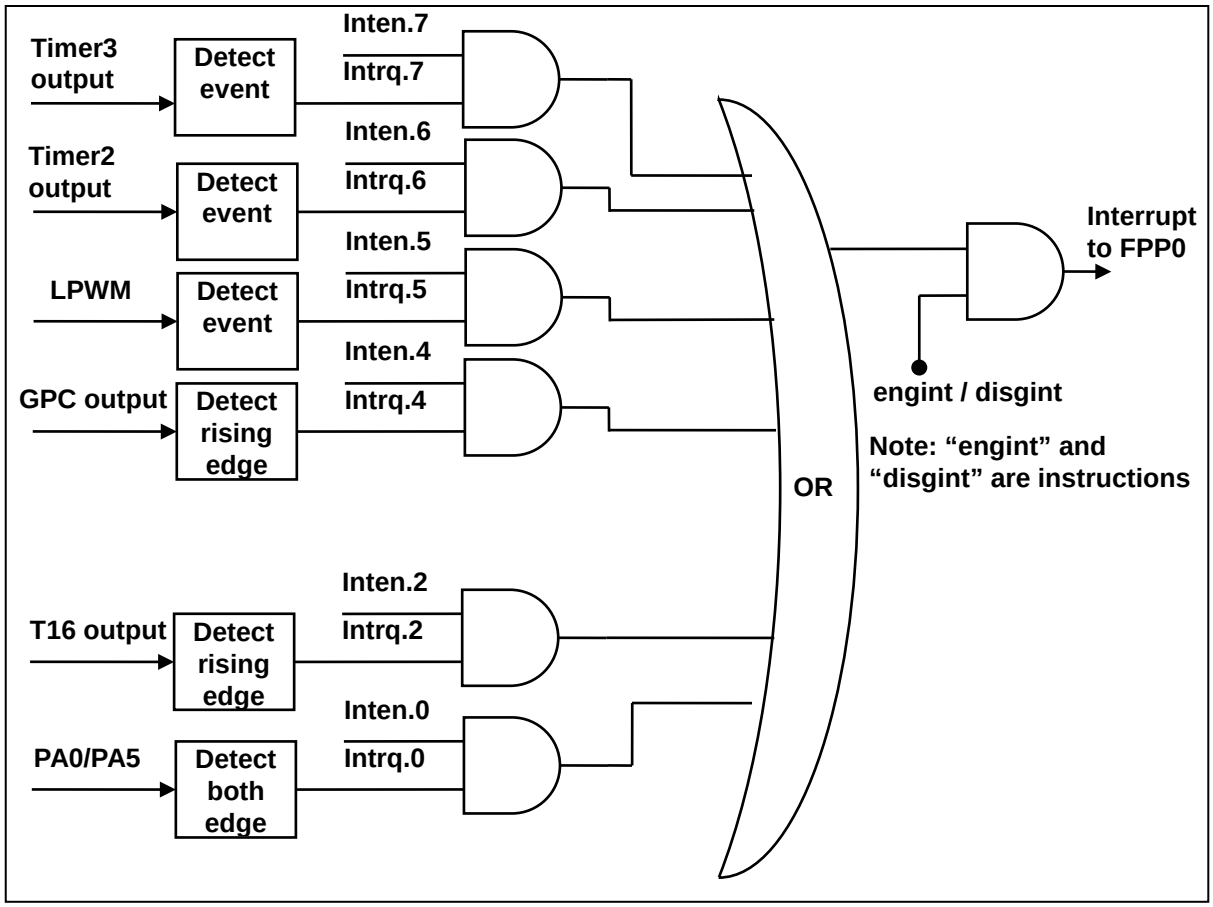


图 11：中断控制器硬件框图

一旦发生中断，其具体工作流程将是：

- 程序计数器将自动存储到 ***sp*** 寄存器指定的堆栈内存。
- 新的 ***sp*** 将被更新为 ***sp+2***。
- 全局中断将被自动停用。
- 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* 寄存器指定的堆栈内存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。

用户必须预留足够的堆栈内存以存中断向量，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，处理一级中断和 pushaf 总共需要四个字节堆栈内存。

```
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             // 停用全局中断

    ...
}
```

```
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 一直在使能状态，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. 省电和掉电

PMS160 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。表 4 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	NILRC
STOPSYS	停止	停止	没改变
STOPEXE	没改变	没改变	没改变

表 4：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*”)

用 ***stopexe*** 指令进入省电模式，只有系统时钟被停用，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，然而，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 或者 ILRC），或者使用 NILRC 作时钟源的 TM2C/TM3C 唤醒（需设定 TM3C.0=1 为 1 开启 NILRC）或比较器唤醒（需同时设定 GPCC.7 为 1 与 GPCS 为 1 来启用比较器唤醒功能）。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 振荡器模块：没改变，如果被启用，则仍然保持运行状态。
- 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：停用，因此 CPU 停止运行。
- OTP 内存关闭。
- 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，LPWMG0/1/2，下同）。
- 唤醒源：
  a. IO Toggle 唤醒：IO 在数字输入模式下的电平变换（*PxC* 位是 0，*PxDIER* 位是 1）。
  b. Timer 计数器唤醒：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。
  c. TM2C/TM3C 唤醒（使用 NILRC 作时钟源）：需设定 TM3C.0=1 为 1 开启 NILRC，同时 Timer2/Timer3 的时钟源选择 NILRC。
  d. 比较器唤醒：使用比较器唤醒时，需同时设定 GPCC.7 为 1 与 GPCS.6 为 1 来启用比较器唤醒功能。但请注意：内部 1.20V Bandgap 参考电压不适用于比较器唤醒功能。

在使用 “***stopexe***” 命令前，须关闭看门狗指令，举例如下：

```
CLKMD.En_WatchDog    =    0;          // 关闭看门狗
stopexe;
....                                  // 进入省电模式
Wdreset;
CLKMD.En_WatchDog    =    1;          // 唤醒后再使能看门狗
```

再举例使用 Timer16 唤醒省电模式 “***stopexe***”：

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

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。通过使用“***stopsys***” 指令，芯片会直接进入掉电模式。在下达 stopsys 指令之前建议将 GPCC.7 设为 0 来关闭比较器。下面显示发出 ***stopsys*** 命令后，PMS160 内部详细的状态：

- 所有的振荡器模块被关闭
- OTP 内存被关闭
- SRAM 和寄存器内容保持不变
- 唤醒源：
  a. 设定为数字模式（PxDIER 对应位为 1）的 IO 切换。
  b. TM2C/TM3C 唤醒（使用 NILRC 作时钟源）：需设定 TM3C.0=1 为 1 开启 NILRC，同时 Timer2/Timer3 的时钟源选择 NILRC。

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 停用
…
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   =   0x3C;       //  系统时钟从 ILRC 变为 IHRC/8
```

5.9.3. 唤醒

进入掉电或省电模式后，PMS160 可以通过切换 IO 引脚或 Tm3c.NILRC 唤醒恢复正常工作；而 Timer16/Timer2 唤醒只适用于省电模式。表 5 显示 ***stopsys*** 掉电模式和 ***stopexe*** 省电模式在唤醒源的差异。

掉电模式(stopsys)和省电模式(stopexe)在唤醒源的差异				
	IO 引脚切换	使用 NILRC 作时钟源的 TM2C/TM3C 唤醒	Timer16 唤醒	比较器唤醒
STOPSYS	是	是	否	否
STOPEXE	是	是	是	是

表 5：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 PMS160，***padier*** 寄存器和 ***pbdier*** 寄存器应对每一个相应的引脚正确设置“使能唤醒功能”。从唤醒事件发生后开始计数，正常的唤醒时间大约是 16 个 ILRC 时钟周期，另外，PMS160 提供快速唤醒功能，透过 ***misc***.5 寄存器选择快速唤醒大约 8 个 ILRC 时钟周期。

休眠模式	唤醒模式	切换 IO 引脚的唤醒时间(t_{WUP})
STOPEXE 省电模式 STOPSYS 掉电模式	快速唤醒	8 * T_{ILRC}, 这里的 T_{ILRC} 是指 ILRC 时钟周期
STOPEXE 省电模式 STOPSYS 掉电模式	正常唤醒	16 * T_{ILRC}, 这里的 T_{ILRC} 是指 ILRC 时钟周期

表 6：休眠模式/唤醒模式 /IO 唤醒时间

请注意，当代码选项(Code Option)设置为快速开机时，不管 MISC.5 写多少，都会强行设定为快速唤醒模式。只有在普通开机模式下，唤醒模式才由 MISC.5 决定。

5.10. IO 引脚

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）并且仍是通过 ***paph***.5 设置上拉。当 PMS160 进入掉电或省电模式时，每个引脚都可以通过切换其状态来唤醒系统。因此，唤醒系统所需的引脚必须设置为输入模式，并将寄存器 ***padier*** 的相应位设置为高。同样地，当 PA0 作为外部中断引脚时，应将 ***padier.***0 设置为高电平。

对于被选择为模拟功能的引脚，必须在寄存器 padier 相应位设置为低，以防止漏电流。

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。表 7 为端口 PA0 位的设定配置表。图 12 显示了 IO 缓冲区硬件图。

pa.0	*pac.0*	*paph.0*	*papl.0*	描述
X	0	0	0	输入，没有弱上拉/下拉电阻
X	0	1	0	输入，有弱上拉电阻
X	0	0	1	输入，有弱下拉电阻
X	0	1	1	输入，有弱上拉/下拉电阻
0	1	X	X	输出低电位，没有弱上拉/下拉电阻
1	1	X	X	输出高电位，没有弱上拉/下拉电阻

表 7：PA0 设定配置表

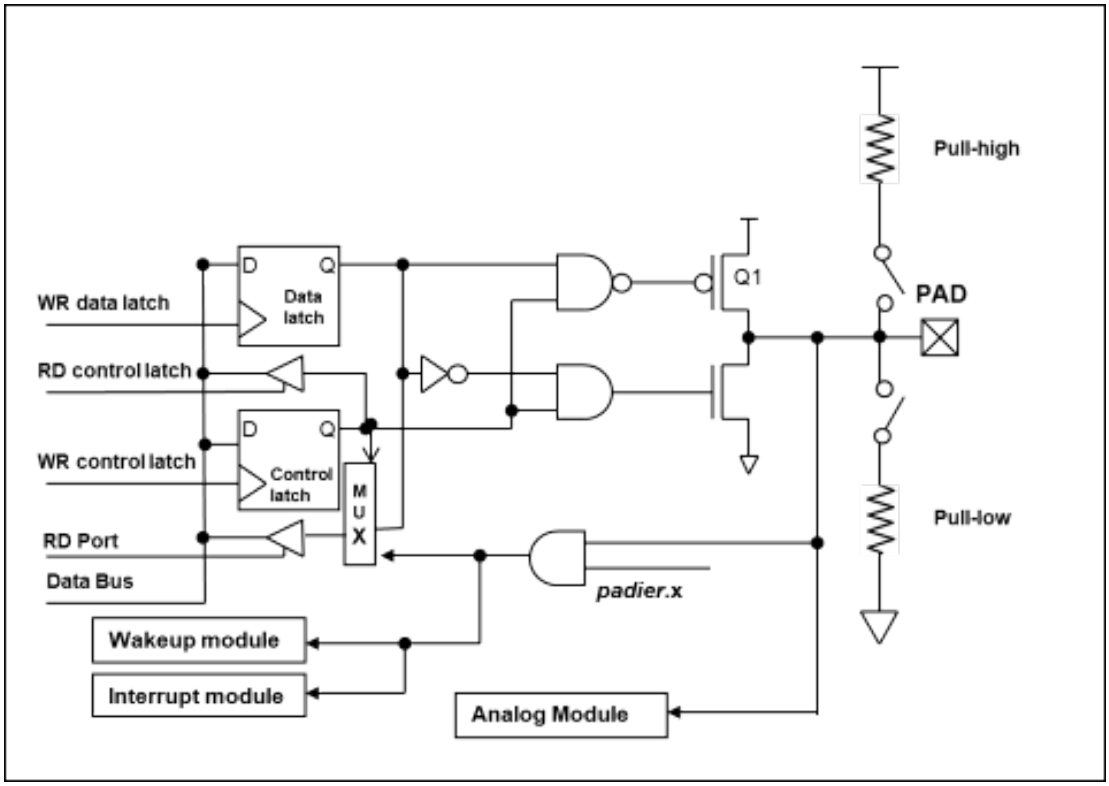


图 12：IO 引脚缓冲区硬件图

5.11. 复位

引起 PMS160 复位的原因很多，一旦复位发生，PMS160 的所有寄存器将被设置为默认值，系统会重新启动，程序计数器会跳跃地址 0x0。发生上电复位或 LVR 复位后，若 VDD 大于 V_{DR}（数据保存电压），数据存储器的值将会被保留，但若在重新上电后 SRAM 被清除，则数据无法保留；若 VDD 小于 V_{DR}，数据存储器的值是在不确定的状态。若是复位是因为 PRSTB 引脚或 WDT 超时溢位，数据存储器的值将被保留。

5.12. 8-bit Timer (Timer2)

PMS160 内置了 1 个 8 位硬件计数器 (Timer2)。Timer2 计数器的时钟源可以来自系统时钟 (CLK)，内部高频 RC 振荡器时钟 (IHRC)，内部低频 RC 振荡器时钟 (ILRC/NILRC)，PA0，PA4 和比较器。寄存器 tm2c 的位[7:4]用来选择 Timer2 的时钟。利用软件程序设计寄存器 ***tm2s*** 位[6:5]，时钟预分频模块提供÷1，÷4，÷16 和÷64 的选择，另外，利用软件程序设计寄存器 ***tm2s*** 位[4:0]，时钟分频器的模块提供了÷1~÷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 位定时器只能执行 8 位上升计数操作，经由寄存器 tm2ct，定时器的值可以设置或读取。当 8 位定时器计数值达到上限寄存器设定的范围时，定时器将自动清除为零，上限寄存器用来定义定时器产生波形的周期。Timer2 定时器有一个工作模式：周期模式；周期模式用于输出固定周期波形或中断事件。图 14 显示出 Timer2 周期模式的时序图。

寄存器 TM2C/TM3C 的位[7:4]可将时钟源选择为 NILRC，以支持更低功耗定时唤醒“stopexe”和“stopsys”。NILRC 振荡器是比 ILRC 更慢的时钟，用来做更省电的唤醒时钟。NILRC 和 ILRC 都可通过 IHRC 估算频率，但 NILRC 的误差更大，所以需要先估算频率才可使用。若需相关 demo，请恰 FAE。

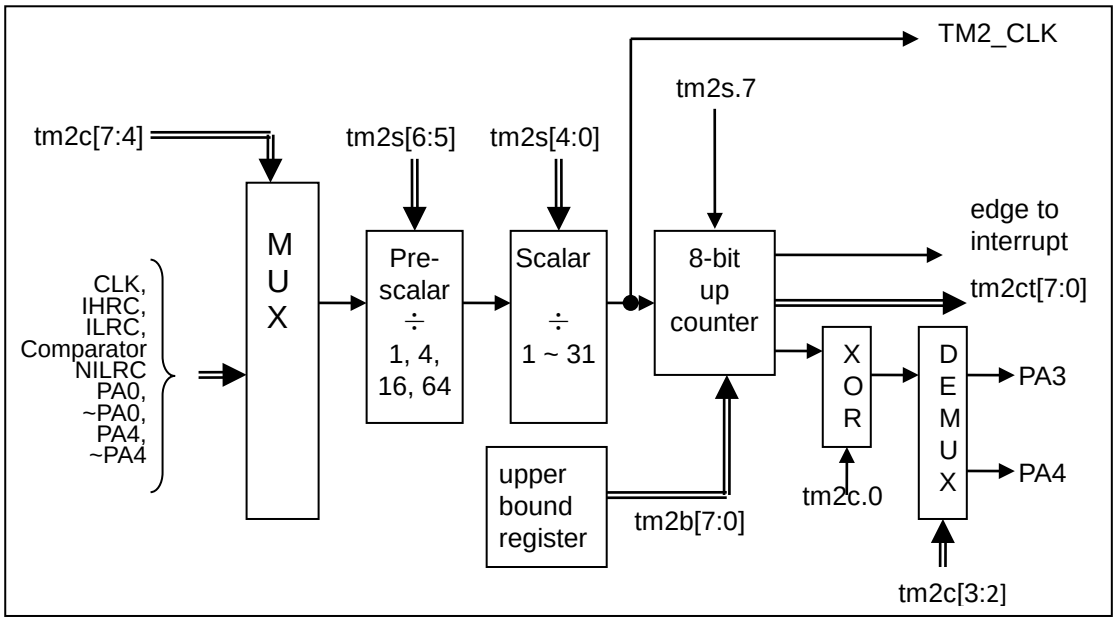


图 13：Timer2 硬件框图

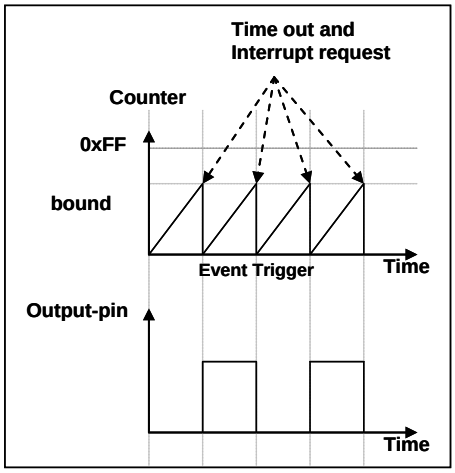


图 14：Timer2 周期模式的时序图

5.12.1. 使用 Timer2 产生周期波形

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：

输出频率= Y ÷ [2 × (K+1) × S1 × (S2+1)]

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 = 0b0000_00000, S1=1, S2=0
➔ 输出频率= 8MHz ÷ [2 × (127＋1) × 1 × (0＋1)] = 31.25kHz

例 2：

tm2c = 0b0001_1000, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s[7:0] = 0b0111_11111, S1=64 , S2 = 31
➔ 输出频率= 8MHz ÷ (2 × (127＋1) × 64 × (31＋1)) =15.25Hz

例 3：

tm2c = 0b0001_1000, Y=8MHz
tm2b = 0b0000_1111, K=15
tm2s = 0b0000_00000, S1=1, S2=0
➔ 输出频率= 8MHz ÷ (2 × (15＋1) × 1 × (0＋1)) = 250kHz

例 4：

tm2c = 0b0001_1000, Y=8MHz
tm2b = 0b0000_0001, K=1
tm2s = 0b0000_00000, S1=1, S2=0
➔ 输出频率= 8MHz ÷ (2 × (1＋1) × 1 × (0＋1)) =2MHz

使用 Timer2 定时器从 PA3 引脚产生周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-bit 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;
       }
}
```

5.13. 8 位计数器 (Timer3)

PMS160 内置了 1 个 8 位硬件计数器 (Timer3)。Timer3 计数器的时钟源可以来自系统时钟 (CLK)，内部高频 RC 振荡器时钟 (IHRC)，内部低频 RC 振荡器时钟 (ILRC/NILRC)，比较器和 IFC。寄存器 tm3c 的位[6:4]用来选择 Timer3 的时钟。利用软件程序设计寄存器 ***tm3s*** 位[6:5]，时钟预分频模块提供÷1，÷4，÷16 和÷64 的选择，另外，利用软件程序设计寄存器 ***tm3s*** 位[1:0]，时钟分频器的模块提供了÷1~÷31 的功能。在结合预分频器以及分频器，Timer3 时钟(TM3_CLK)频率可以广泛和灵活，以提供不同产品应用。

8 位定时器只能执行 8 位上升计数操作，经由寄存器 tm3ct，定时器的值可以设置或读取。当 8 位定时器计数值达到上限寄存器设定的范围时，定时器将自动清除为零，上限寄存器用来定义定时器产生波形的周期。Timer3 定时器有一个工作模式：周期模式；周期模式用于输出固定周期波形或中断事件。图 16 显示出 Timer3 周期模式的时序图。

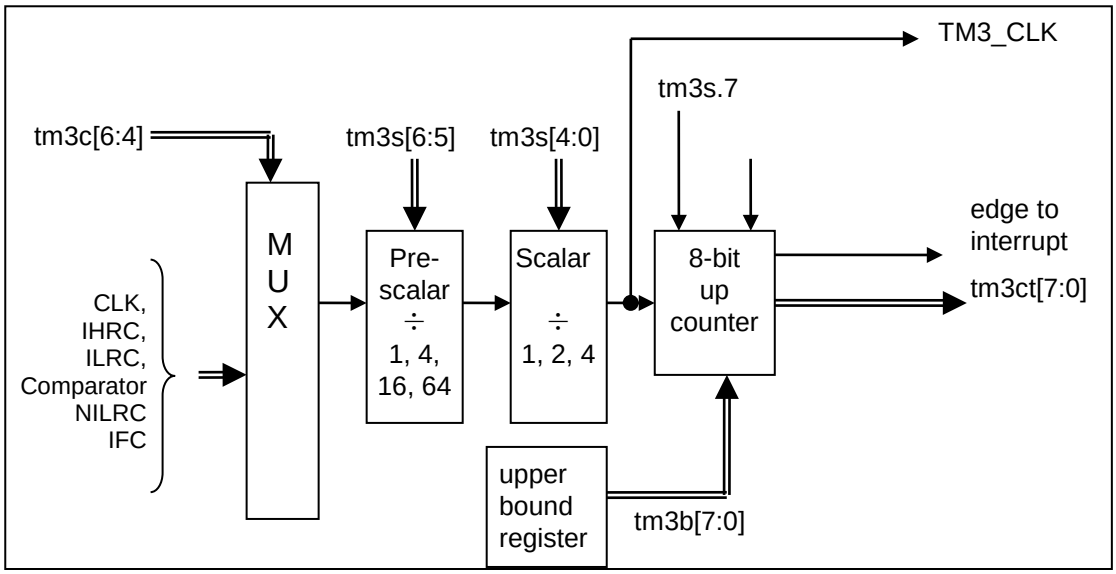


图 15：Timer3 硬件框图

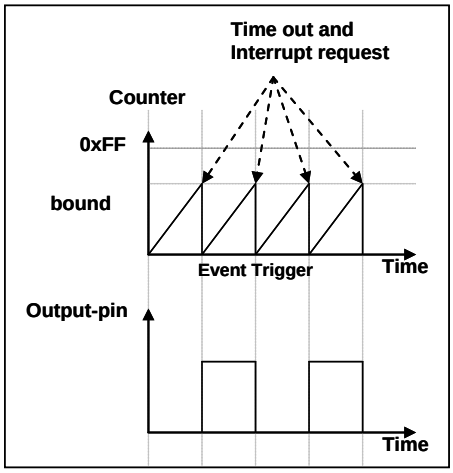


图 16：Timer3 周期模式的时序图

5.14.11 位 SuLED LPWM 计数器 (LPWMG0/1/2)

PMS160 内置一组三路 11 位 SuLED (Super LED)硬件 LPWM 生成器(LPWMG0、LPWMG1 和 LPWMG2)。各路输出端口如下：

- LPWMG0 – PA3
- LPWMG1 – PA4
- LPWMG2 – PA0, PA7

5.14.1. LPWM 波形

LPWM 输出波形（图 17）有一个时基（T_{Period} = 时间周期）和一个周期里输出高电平的时间（占空比）。LPWM 输出的频率取决于时基($f_{LPWM} = 1/T_{Period}$)。

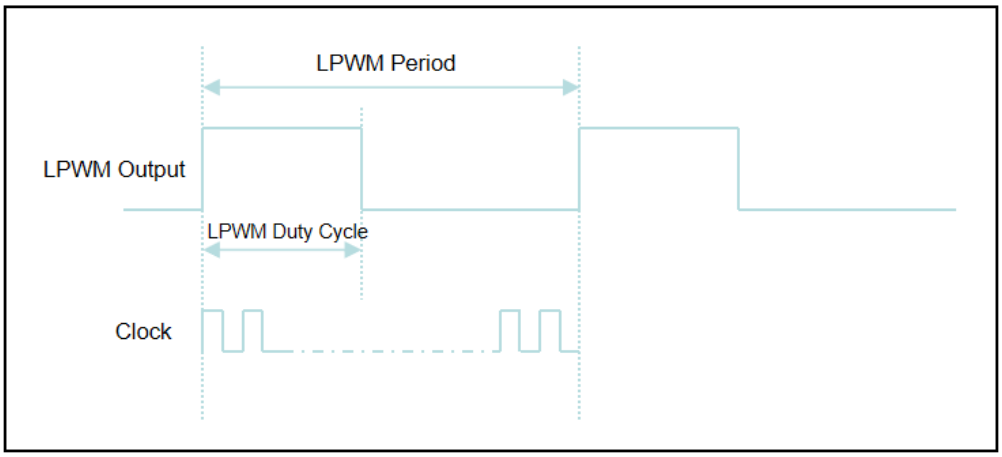


图 17：LPWM 输出波形

5.14.2. 硬件框图

图 18 所示是整组 SuLED 11 位 LPWM 生成器的硬件方框图。这三组 LPWM 生成器使用共同的 Up-Counter 和时钟源选择开关来产生时基，所以 LPWM 周期的起点（上升沿）是同步的，时钟源可以是 IHRC 或者系统时钟。LPWM 信号输出引脚通过 LPWMGxC 寄存器来选择。LPWM 波形的周期由 LPWM 上限高和低寄存器决定，各路 LPWM 波形的占空比由各路 LPWM 占空比高和低寄存器决定。

在 LPWMG0 通道的那两个附带的 OR 和 XOR 逻辑门是用于产生互补非重叠并有死区的开关控制波形的。

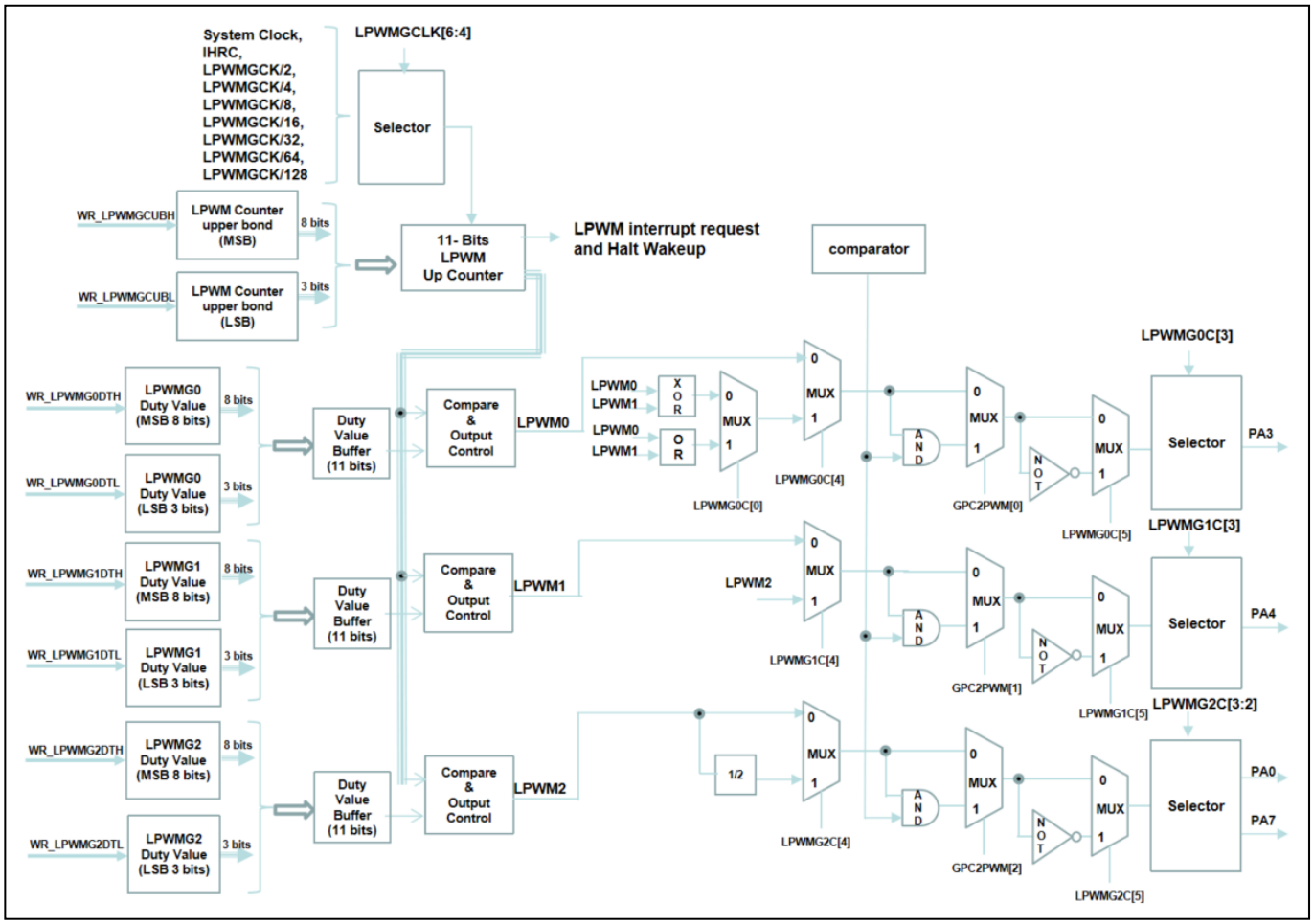


图 18：整组 SuLED 三路 11 位 LPWM 生成器硬件框图

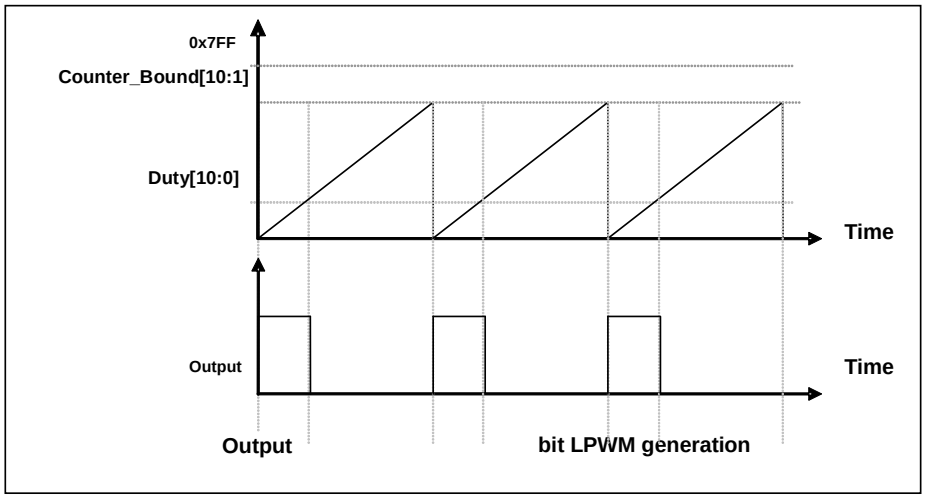


图 19：11 位 LPWM 生成器输出时序图

寄存器 GPC2PWM 的位[2:0]可以选择由比较器分别控制 LPWMG0/1/2 的 PWM 输出功能，用户可根据需求选择是否启用。启用后，此时当比较器输出是 1 时，LPWM 停止输出；而比较器输出是 0 时，LPWM 恢复输出，如图 20 所示。

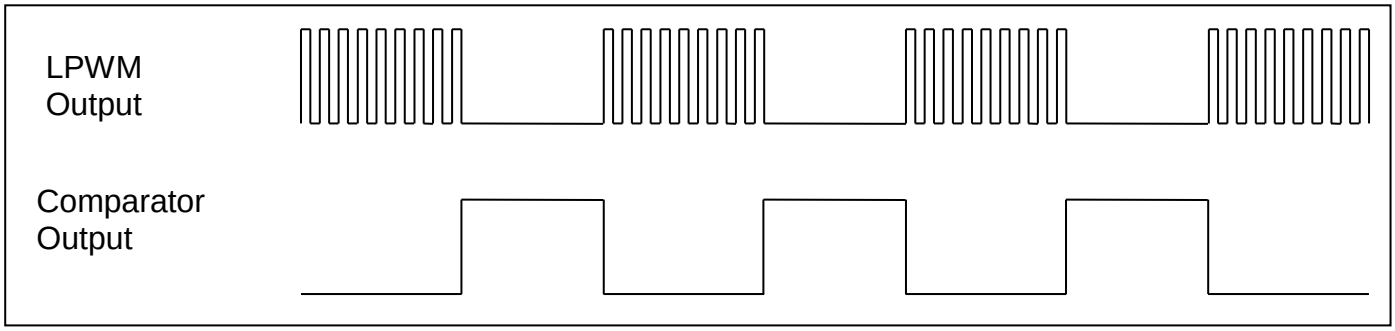


图 20：比较器控制 LPWM 波形的输出

5.14.3. 11 位 LPWM 生成器计算公式

LPWM 输出频率 F_{LPWM} = F clock source ÷ [P × (CB10_1 + 1)]

LPWM 占空比（时间） = (1 / F_{LPWM}) × (DB10_1 + DB0 × 0.5 + 0.5) ÷ (CB10_1 + 1)

LPWM 占空比（百分比）= (DB10_1 + DB0 × 0.5 + 0.5) ÷ (CB10_1 + 1) × 100%

这里,

P=LPWMGCLK[6:4]; 预分频 **P**=1,2,4,8,16,32,64,128

DB10_1 = Duty_Bound[10:1] = {LPWMGxDTH [7:0],LPWMGxDTL[7:6]},（x=0/1/2） 占空比

DB0 = Duty_Bound[0] = LPWMGxDTL[5] （x=0/1/2）

CB10_1 = Counter_Bound[10:1] = {LPWMGCUBH[7:0],LPWMGCUBL[7:6]}, 计数器

5.14.4. 带互补死区的 LPWM 波形范例

基于 PMS160 独特的 11 bit SuLED LPWM 结构，在此采用 LPWM2 输出、LPWM0 与 LPWM1 异或后通过 LPWM0 反极性输出，来获得两路互补带死区 LPWM 波形。示例如下：

（注：该功能暂不支持仿真。）

```
#define   dead_zone              10          //   死区时间 = 10% * (1/LPWM_Frequency) us
#define   LPWM_Pulse     50            //   该互补死区 LPWM 占空比为 50%

#define   LPWM_Pulse_1   35            //   该互补死区 LPWM 占空比为 35%
#define   LPWM_Pulse_2   60            //   该互补死区 LPWM 占空比为 60%
#define   switch_time    400*2     //   切换占空比时，用于调整切换时间
//注：为防止杂波产生，switch_time 应为 LPWM 周期的倍数。此例 LPWM 周期：1/2.5KHz = 400 us，故切
//换时间为 400*2 us


void    FPPA0 (void)
{
.ADJUST_IC SYSCLK=IHRC/16, IHRC=16MHz, VDD=5V;
//******* 产生固定占空比 ************************
//------- 设置计数上限及占空比 --------------------
LPWMG0DTL     =    0x00;
LPWMG0DTH     =    LPWM_Pulse + dead_zone;

LPWMG1DTL     =    0x00;
LPWMG1DTH     =    dead_zone;       // LPWMG0 与 LPWMG1 异或后，LPWM 占空比
                                    //为 LPWM_Pulse%
LPWMG2DTL       =  0x00;
LPWMG2DTH     =    LPWM_Pulse + dead_zone*2;

LPWMGCUBL     =    0x00;
LPWMGCUBH     =    100;
//---- 统一配置 LPWM 时钟及分频 ----------------
$ LPWMGCLK     Enable, /1, sysclk;
//------- 输出控制 ----------------------------
$ LPWMG0C  Inverse,LPWM_Gen,PA3,gen_xor;  //   LPWMG0 与 LPWMG1 异或后，从
                                          //   PA3 脚反极性输出
$ LPWMG1C  LPWMG1,disable;                //    LPWMG1 不输出
$ LPWMG2C      PA0;                       //    LPWMG2 PA0 输出

while(1)
{
   //******** 切换占空比 *********************************
```

```
    // 切换占空比时，为避免可能出现的瞬间死区消失，应遵循如下顺序。
    // 占空比由大变小：50%/60%  →  35%
  LPWMG0DTL  =  0x00;
  LPWMG0DTH  =  LPWM_Pulse_1 + dead_zone;
  LPWMG2DTL  =  0x00;
  LPWMG2DTH  =  LPWM_Pulse_1 + dead_zone*2;
  .delay     switch_time

    //占空比由小变大：35% → 60%
    LPWMG2DTL     =     0x00;
  LPWMG2DTH  =  LPWM_Pulse_2 + dead_zone*2;
  LPWMG0DTL  =  0x00;
  LPWMG0DTH  =  LPWM_Pulse_2 + dead_zone;
  .delay     switch_time
}
}
```

上述程序中，固定占空比时对应的 LPWM0/LPWM2 波形如图 21 所示。

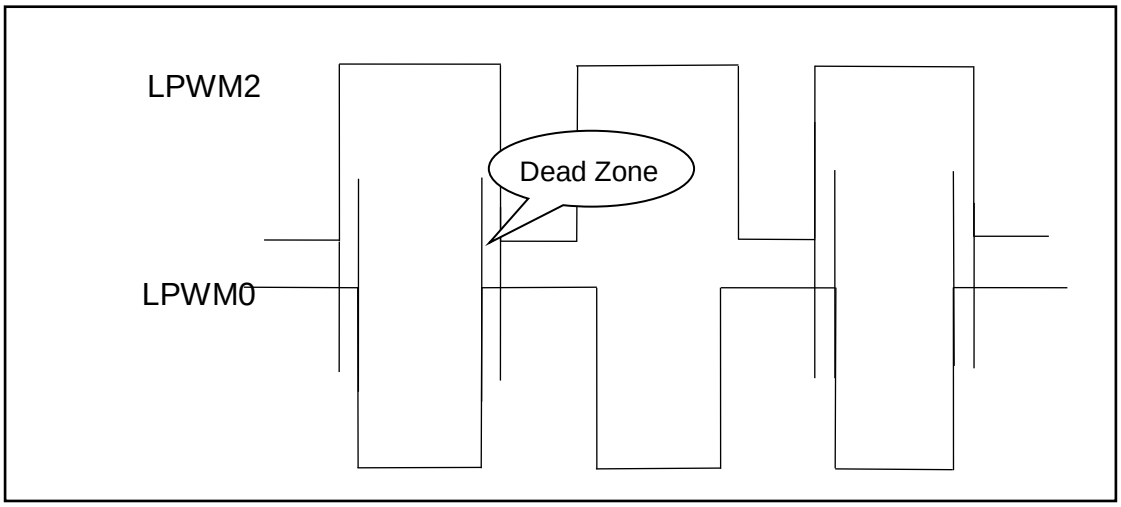


图 21：两路互补 LPWM 波形

切换占空比时对应的 LPWM0/LPWM2 波形如图 22。

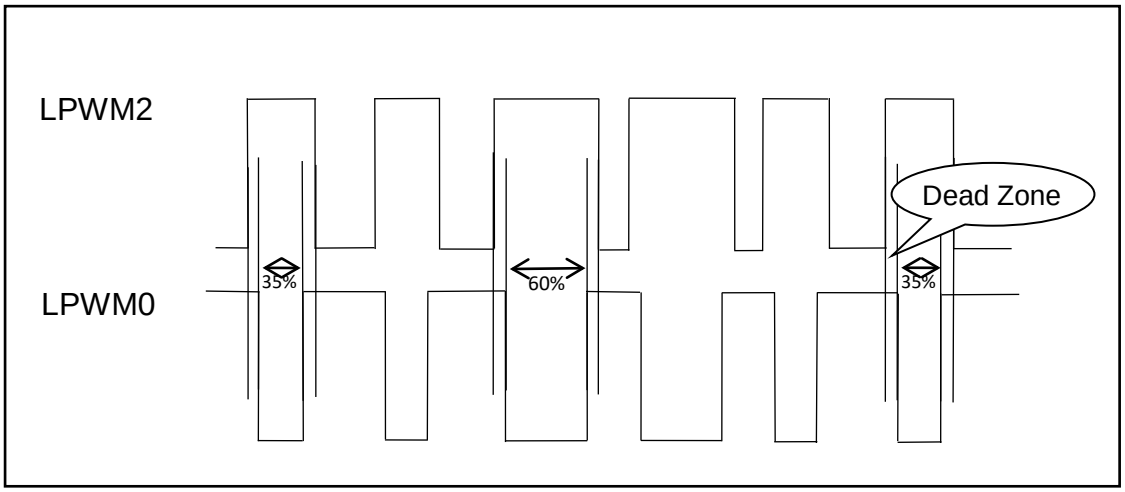


图 22：两路互补 LPWM 波形

可以发现，上述例程所得波形，其死区是两组 LPWM 同时为高。若用户需要 LPWM 同时为低的死区，仅需改变各自输出控制的 Inverse 即可。如：

```
$ LPWMG0C  LPWM_Gen,PA3,gen_xor;
$ LPWMG2C   Inverse, PA0;
```

5.15. 触摸功能

PMS160 内含一个 IFC 触摸检测电路，图 23 为其功能方框图：

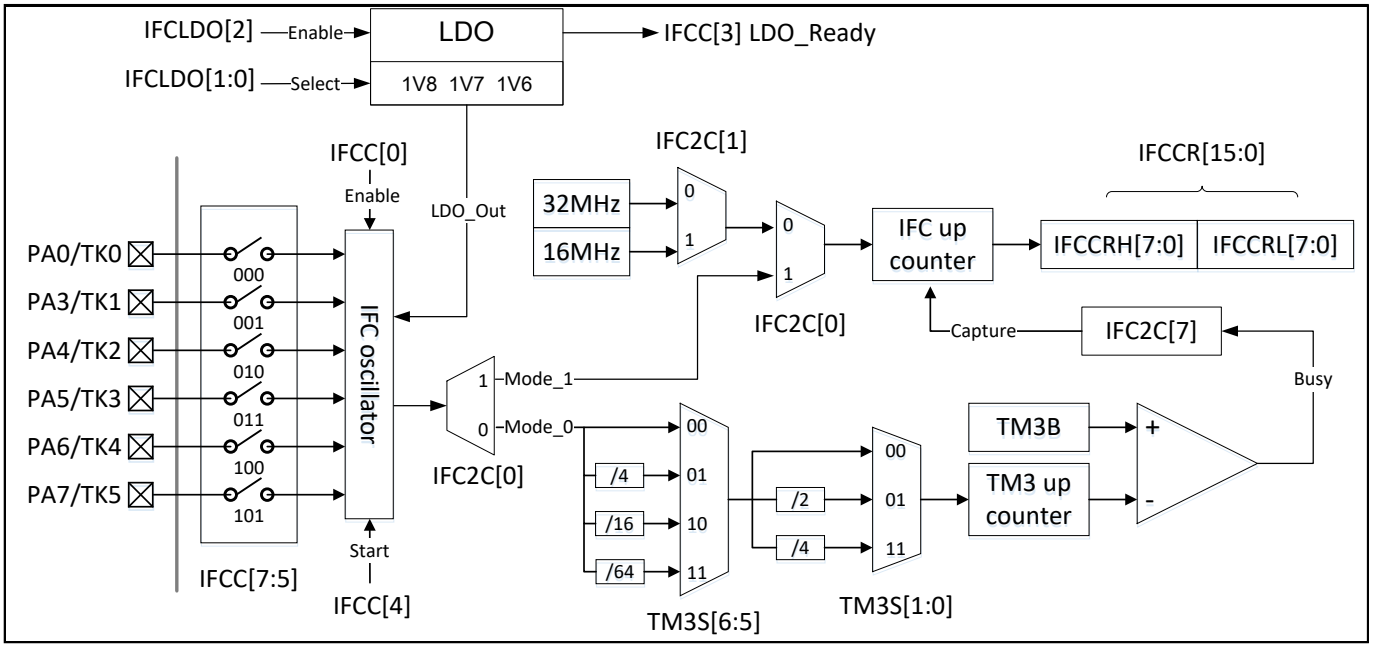


图 23：触摸检测电路的功能方框图

PMS160 中的触摸检测电路应用电容式感应的方法，检测手指的虚拟地面效应电容或感应极片之间的电容。该触摸模块内置有 IFC 振荡电路以及硬件 LDO 电压基准，且 LDO 另有 1.8V、1.7V、1.6V 三种基准电压可以选用。PMS160 内建二种触摸转换模式，Mode_0 及 Mode_1，以下将会详细介绍。

通过读取 IFC 触摸计数器 IFCCRH 和 IFCCRL 的值，用户可监测感应极片上的电容量变化。读取到的值与 CP（电容触摸传感器引脚）有关，而 CP 表示因用户手指触摸而变化的 PCB、导线和触摸板的组合的总电容。一旦 CP 值被改变，触摸计数器 IFCCRH 和 IFCCRL 的值也将跟着变化，由变化即可判断是否有触摸动作。

Mode_0 的动作方式为使用 TM3 计数 IFC 的振荡脉波数。当 TM3_Counter 计数值达到计数上限寄存器 TM3B 所设定的值，将会停止 IFC 计时，通过读取 IFC 计数器（IFCCRH 和 IFCCRL）的值可得到当下触摸通道的触摸计数值。藉由 IFC 计数值的变化差异即可判断触摸板是否被按下或释放。

当给定 IFCC.4 start 信号后，IFC2C.7 的 busy 标志将被拉高为“1”，开始做触摸检测及转换，直至达到 TM3B 计数上限时 IFC2C.7 busy 信号被清零，表示转换完成，然后读取触摸计数器 IFCCRH 和 IFCCRL 的值即可。

Mode_1 是将 IFC 振荡器的输出设成 IFC 触摸计数器的时钟源。由 IFC 触摸计数器来直接计数 IFC 振荡器输出的频率。用户透过程序的其他计时方式来决定 IFC 触摸计数器的工作时间。

注意：

1. 在执行 IFC Touch 检测及转换流程时，系统时钟 SYSCLK 最高不可超过 1MHz。执行前请注意切换系统时钟至 1MHz 或以下，待 IFC Touch 转换完成后才可切换至更高频率。

```
CLKMD_Save=CLKMD;
$ CLKMD IHRC/16,En_IHRC,En_ILRC;
……
……
CLKMD=CLKMD_Save;
```

建议： Project 中带有 IFC 转换程序时建议系统时钟直接使用 1MHz，可免除频繁切换系统时钟的困扰。而且系统时钟切换有可能带来计时不准、中断运行时间变长等现象。

2. LDO 启用前必须先确保 ILRC 低频振荡处于启用状态。LDO 关闭后再重启的时间间隔必须大于 2 个 ILRC 时钟周期或是超过 50uS。

```
$ IFCLDO;                   // LDO Off
CLKMD.EN_ILRC = 1;          // 启用前确认 ILRC 在开启状态
.delay 50;                  // 启用前 Delay 50us ( @sysclk=1MHz )
$ IFCC ;                    // 清零 IFCC，清零 LDO ready 标志
$ IFCLDO Enable, 1V8        // 启用 LDO
while (!IFCC.LDO_Ready) NOP;
EXCAP =  0x08;              // 默认写 0x08
CHDIS =  0x40;              // 默认写 0x40
```

3. 配置 IFCC 寄存器时，TK 通道必须先设置使能，且延时 1uS 之后才可将 IFC 振荡器使能。在 IFC 振荡器使能后需延时 20uS 等待振荡稳定才可将 IFC 触摸计数使能。

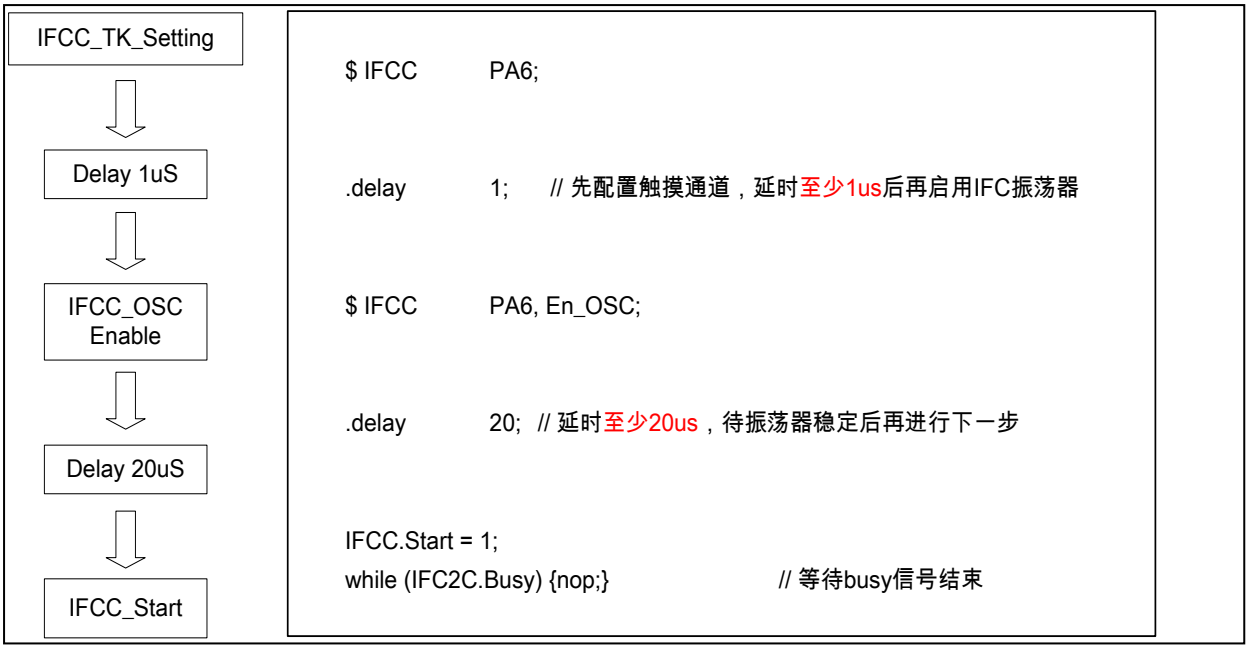


图 24：IFCC 寄存器配置时序图

5.15.1. 触摸检测程序使用范例

在 MCU_IDE_0.92C7 后可支持 PMS160 IFC Touch 转换的宏指令：

Touch_TKProcess 参数 0，参数 1，参数 2，参数 3

参数 0：IFC Touch 通道设置，0 ~ 5。 设 0 = TK0, 1 = TK1。

参数 1：IFC Touch 计数值。(16Bits)

参数 2：IFC Mode，0 = Mode_0；1 = Mode_1

参数 3：/2；/4；/8；/16…。IFC Touch 转换后系统主频切换频率(IHRC)。

关于 Touch_TKProcess 宏指令的具体细节，请参考 IDE 中的 PMS160.INC 档案。

参考例程如下：

```
#include        "extern.h"
Debug_Printf        => 1
Debug_CMD           => 1
Div                 => 8
word TK_Data[6],TK_Avg[6];

void      FPPA0 (void)
{
    .ADJUST_IC      SYSCLK=IHRC/Div, IHRC=16MHz, VDD=4V,INIT_RAM;
    .if Debug_CMD==1
    .DBG_CMD        LOG_CREATE      "log.bin";
    .DBG_CMD        SAVE_BIN        WAV, "P13:TK1:P13:TK2:P13:TK3:P13:TK4:P13:TK5"
    .endif

    $ TM2S /64, /25;
    TM2B = 200;
    $ TM2C IHRC;

    while (1)
    {
        //              wdreset;
        if (INTRQ.TM2)
        {
```

```
        INTRQ.TM2       =       0;
        word temp;
        .for N,<1,2,3,4,5>
            Touch_TKProcess TK#N#,TK_Data[N],0,/Div
            temp        =       TK_Avg[N] << 2;
            temp        =       temp - TK_Avg[N];
            temp        =       temp + TK_Data[N];
            TK_Avg[N]=      temp >> 2;
        .endm

        .if Debug_CMD==1
        .for N,<1,2,3,4,5>
            A=TK_Data[N]$0;
            .DBG_CMD    SAVE_ALU;   //将 A 数据存 Log 并传送 Touch Count WReviewdows
            A=TK_Data[N]$1;
            .DBG_CMD    SAVE_ALU;   //将 A 数据存 Log 并传送 Touch Count WReviewdows
        .endm
        .endif

.if Debug_Printf==1
.printf(rep,"TK1:%d TK2:%d TK3:%d TK4:%d TK5:%d\n", TK_Data[1], TK_Data[2], TK_Data[3], TK_Data[4],
TK_Data[5]);
.endif
            }
    }
}
```

6. IO 寄存器

6.1. ACC 状态标志寄存器 (*flag*)，地址 =0x00

位	初始值	读/写	描述
7 - 4	-	-	保留。这 4 个位读是“1”。
3	-	读/写	OV（溢出标志）。溢出时置 1。
2	-	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	-	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	-	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*)，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。请注意 0 位必须维持为 0 因程序计数器是 16 位。

6.3. 时钟模式寄存器 (*clkmd*)，地址 =0x03

<table>
<tr><th>位</th><th>初始值</th><th>读/写</th><th colspan="2">描述</th></tr>
<tr><td rowspan="10">7 - 5</td><td rowspan="10">111</td><td rowspan="10">读/写</td><td colspan="2">系统时钟 (CLK)选择：</td></tr>
<tr><td>类型 0，clkmd[3]=0</td><td>类型 1，clkmd[3]=1</td></tr>
<tr><td>000：IHRC/4</td><td>000：IHRC/16</td></tr>
<tr><td>001：保留</td><td>001：IHRC/8</td></tr>
<tr><td>01x：保留</td><td>010：ILRC/16</td></tr>
<tr><td>100：保留</td><td>011：IHRC/32</td></tr>
<tr><td>101：保留</td><td>100：IHRC/64</td></tr>
<tr><td>110：ILRC/4</td><td>110：保留</td></tr>
<tr><td>111：ILRC（默认）</td><td>1x1：保留</td></tr>
<tr><td>4</td><td>0</td><td>读/写</td><td colspan="2">内部高频 RC 振荡器功能。 0/1：停用/启用</td></tr>
<tr><td>3</td><td>0</td><td>读/写</td><td colspan="2">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</td></tr>
<tr><td>2</td><td>1</td><td>读/写</td><td colspan="2">内部低频 RC 振荡器功能。0/1：停用/启用
当内部低频 RC 振荡器功能停用时，看门狗功能同时被关闭。</td></tr>
<tr><td>1</td><td>1</td><td>读/写</td><td colspan="2">看门狗功能。0/1：停用/启用</td></tr>
<tr><td>0</td><td>0</td><td>读/写</td><td colspan="2">引脚 PA5/PRSTB 功能。0/1：PA5 / PRSTB</td></tr>
</table>

6.4. 中断允许寄存器 (*inten*)，地址 =0x04

位	初始值	读/写	描述
7	0	读/写	使能 Timer3 中断。0/1：停用/启用
6	0	读/写	使能 Timer2 中断。0/1：停用/启用
5	0	读/写	使能 LPWM 中断。0/1：停用/启用
4	0	读/写	使能比较器中断。0/1：停用/启用
3	-	-	保留
2	0	读/写	使能 Timer16 溢出中断。0/1：停用/启用
1	-	-	保留
0	0	读/写	使能 PA0/PA5 中断。0/1：停用/启用

6.5. 中断请求寄存器 (*intrq*)，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	-	读/写	LPWM 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
4	-	读/写	比较器的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
3	-	-	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	-	-	保留
0	-	读/写	引脚 PA0/PA5 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. Timer16 控制寄存器 (*t16m*)，地址 =0x06

位	初始值	读/写	描述
7 - 5	000	读/写	Timer16 时钟选择： 000：Timer16 停用 001：CLK（系统时钟） 010：保留 011：PA4 下降沿（从外部引脚） 100：IHRC 101：保留 110：ILRC 111：PA0 下降沿（从外部引脚）
4 - 3	00	读/写	Timer16 时钟分频： 00：÷1 01：÷4 10：÷16 11：÷64
2 - 0	000	读/写	中断源选择。当所选择的状态位变化时，中断事件发生。 0：bit 8 of Timer16 1：bit 9 of Timer16 2：bit 10 of Timer16 3：bit 11 of Timer16 4：bit 12 of Timer16 5：bit 13 of Timer16 6：bit 14 of Timer16 7：bit 15 of Timer16

6.7. 中断边缘选择寄存器 (*integs*)，地址 =0x0c

位	初始值	读/写	描述
7 - 6	00	只写	GPC 中断边缘选择。 00：上升缘和下降缘都请求中断 01：上升缘请求中断 10：下降缘请求中断 11：保留
5	-	-	保留。写 0。
4	0	只写	Timer16 中断边缘选择： 0：上升缘请求中断。 1：下降缘请求中断。
3 - 2	-	-	保留。写 00。
1 - 0	00	只写	PA0/PA5 中断边缘选择。 00：上升缘和下降缘都请求中断 01：上升缘请求中断 10：下降缘请求中断 11：保留

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

位	初始值	读/写	描述
7 - 6	11	只写	使能 PA7~PA6 数字输入和唤醒事件。1/0：启用/ 停用 如果 PA7~PA6 位设 0 可停用唤醒。
5	1	只写	使能 PA5 数字输入、唤醒事件和中断请求。1/0：启用/ 停用 如果这个位设为 0，PA5 则不能用来唤醒系统，并且停用中断请求。
4 - 3	11	只写	使能 PA4~PA3 数字输入和唤醒事件。1/0：启用/ 停用 如果 PA4~PA3 位设 0 可停用唤醒。
2 - 1	-	-	保留，建议写 00。
0	1	只写	使能 PA0 数字输入、唤醒事件和中断请求。1/0：启用/ 停用 如果这个位设为 0，PA0 则不能用来唤醒系统，并且停用中断请求。

6.9. 端口 A 数据寄存器 (*pa*)，地址 =0x10

位	初始值	读/写	描述
7 - 0	0x00	读/写	数据寄存器的端口 A。

6.10. 端口 A 控制寄存器 (*pac*)，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。

6.11. 端口 A 上拉控制寄存器 (*paph*)，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。这些寄存器是用来控制上拉高端口 A 每个相应的引脚。 0/1：停用/启用

6.12. 端口 A 下拉控制寄存器 (*papl*)，地址 =0x13

位	初始值	读/写	描述
7 - 0	0x00	读/写	端口 A 下拉控制寄存器。这些寄存器是用来控制下拉高端口 A 每个相应的引脚。 0/1：停用/启用

6.13. 杂项寄存器 (*misc*), 地址 =0x1b

位	初始值	读/写	描述
7 - 6	-	-	保留（写 0）。
5	0	只写	快唤醒功能。快速唤醒功能 EOSC 模式下不支持。 0：正常唤醒 唤醒时间是 16 个 ILRC 时钟（不适用快速开机） 1：快速唤醒 唤醒时间为 8 个 ILRC 时钟
4 - 3	-	-	保留（写 0）。
2	0	只写	停用 LVR 功能： 0 / 1：启用 / 停用
1 - 0	00	只写	看门狗时钟超时时间设定： 00：8k ILRC 时钟周期 01：16k ILRC 时钟周期 10：64k ILRC 时钟周期 11：256k ILRC 时钟周期

6.14. 比较器控制寄存器 (*gpcc*)，地址 =0x1a

位	初始值	读/写	描述
7	0	读/写	启用比较器。0/1：停用/启用 当此位被设置为启用，请同时设置相应的模拟输入引脚是数字停用，以防止漏电。
6	-	只读	比较器结果。 0：正输入 < 负输入 1：正输入 > 负输入
5	0	读/写	选择比较器的结果是否由 TM2_CLK 采样输出。 0：比较器的结果没有 TM2_CLK 采样输出 1：比较器的结果是由 TM2_CLK 采样输出
4	0	读/写	选择比较器输出的结果是否反极性。 0：比较器输出的结果没有反极性 1：比较器输出的结果是反极性
3 - 1	000	读/写	选择比较器负输入的来源。 000：PA3 001：PA4 010：内部 1.20 V bandgap 参考电压 011：$V_{internal\ R}$ 100：PA6 101：保留 11X：保留
0	0	读/写	选择比较器正输入的来源。 0：$V_{internal\ R}$ 1：PA4

6.15. 比较器选择寄存器 (*gpcs*)，地址 =0x1e

位	初始值	读/写	描述
7	0	只写	比较器输出启用（到 PA0）。 0/1：停用/启用
6	0	只写	比较器唤醒启用。（gpcc.6 发生电平变化时才可唤醒） 0/1：停用/启用
5	0	只写	选择比较器参考电压 $V_{internal\ R}$ 最高的范围。
4	0	只写	选择比较器参考电压 $V_{internal\ R}$ 最低的范围。
3 - 0	0000	只写	选择比较器参考电压 $V_{internal\ R}$。 0000（最低）~ 1111（最高）

6.16. Timer2 控制寄存器 (*tm2c*)，地址 =0x1c

位	初始值	读/写	描述
7 - 4	0000	读/写	Timer2 时钟源选择： 0000：停用 0001：CLK（系统时钟） 0010：IHRC 0011：保留 0100：ILRC 0101：比较器输出 011x：保留 1000：PA0（上升沿） 1001：~PA0（下降沿） 101x：保留 1100：PA4（上升沿） 1101：~PA4（下降沿）
3 - 2	00	读/写	Timer2 输出选择： 00：停用 01：保留 10：PA3 11：PA4
1	0	读/写	保留
0	0	读/写	启用 Timer2 反极性输出： 0/1：停用/启用

6.17. Timer2 计数寄存器 (*tm2ct*)，地址 =0x1d

位	初始值	读/写	描述
7 - 0	0x00	读/写	Timer2 定时器位[7:0]。

请注意：Timer2 只设计了周期模式，因此不要读 tm2ct 寄存器。

6.18. Timer2 分频寄存器 (*tm2s*)，地址 = 0x17

位	初始值	读/写	描述
7	0	只写	保留
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。

6.19. Timer2 上限寄存器 (*tm2b*)，地址 = 0x09

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

6.20. Timer3 控制寄存器 (*tm3c*)，地址 = 0x2c

位	初始值	读/写	描述
7	-	-	保留
6 - 4	000	读/写	Timer3 时钟源选择： 000：停用 001：SYSCLK（系统时钟） 010：IHRC 011：保留 100：ILRC 101：比较器输出 110：NILRC 111：IFC
3 - 1	-	-	保留
0	0	读/写	启用 NILRC 0 / 1: 停用/启用

6.21. Timer3 计数寄存器 (*tm3ct*)，地址 = 0x2d

位	初始值	读/写	描述
7 - 0	0x00	读/写	Timer3 定时器位[7:0]。

请注意：Timer3 只设计了周期模式，因此不要读 tm3ct 寄存器。

6.22.Timer3 分频寄存器 (*tm3s*), 地址= 0x2e

位	初始值	读/写	描述
7	0	只写	保留
6 - 5	00	只写	Timer3 时钟预分频器： 00：÷ 1 01：÷ 4 10：÷ 16 11：÷ 64
4 - 2	-	只写	保留
1 - 0	00	只写	Timer3 时钟分频 00：÷ 1 01：÷ 2 10：保留 11：÷ 4

6.23.Timer3 上限寄存器 (*tm3b*), 地址 = 0x2f

位	初始值	读/写	描述
7 - 0	0x00	只写	Timer3 上限寄存器。

6.24.LPWM 控制寄存器(*GPC2PWM*), 地址 = 0x33

位	初始值	读/写	描述
7 - 4	-	-	保留
3	-	只写	LPWMG 时钟源选择 0：IHRC = 16MHz 1：IHRC*2 = 32MHz
2	-	只写	使能比较器控制 LPWMG2 输出 0/1：停用/启用
1	-	只写	使能比较器控制 LPWMG1 输出 0/1：停用/启用
0	-	只写	使能比较器控制 LPWMG0 输出 0/1：停用/启用

6.25.LPWMG0 控制寄存器(*LPWMG0C*)，地址= 0x34

位	初始值	读/写	描述
7	-	-	保留。
6	-	只读	LPWMG0 生成器输出状态。
5	0	只写	选择 LPWMG0 的输出的结果是否反极性： 0/1：停用/启用
4	0	只写	LPWMG0 输出选择。 0：LPWMG0 输出 1：LPWMG0 XOR LPWMG1 或者 LPWMG0 OR LPWMG1（通过 LPWMG0C.0 位来选择）
3	0	读/写	LPWMG0 输出端口选择。 0：输出停用 1：PA3
2 - 1	-	-	保留。
0	0	读/写	LPWMG0 输出预选择。 0：LPWMG0 XOR LPWMG1 1：LPWMG0 OR LPWMG1

6.26.LPWMG1 控制寄存器 (*LPWMG1C*)，地址= 0x35

位	初始值	读/写	描述
7	-	-	保留。
6	-	只读	LPWMG1 生成器输出状态。
5	0	读/写	选择 LPWMG1 的输出的结果是否反极性： 0/1：停用/启用。
4	0	读/写	LPWMG1 输出选择： 0：LPWMG1 1：LPWMG2
3	0	读/写	LPWMG1 输出端口选择： 0：输出停用 1：PA4
2 - 0	-	读/写	保留。

6.27.LPWMG2 控制寄存器(*LPWMG2C*)，地址 = 0x36

位	初始值	读/写	描述
7	-	-	保留。
6	-	只读	LPWMG2 生成器输出状态。
5	0	读/写	选择 LPWMG2 的输出的结果是否反极性： 0/1：停用/启用
4	0	读/写	LPWMG2 输出选择: 0：LPWMG2 1：LPWMG2 ÷2
3 - 2	00	读/写	LPWMG2 输出端口选择： 00：输出停用 01：PA0 10：PA7 11：保留
1 - 0	-	读/写	保留

6.28.LPWMG 时钟寄存器(*LPWMGCLK*)，地址 = 0x37

位	初始值	读/写	描述
7	0	只读	LPWMG 停用/ 启用。 0：LPWMG 停用 1：LPWMG 启用
6 - 4	000	只读	LPWMG 时钟预分频。 000：÷1 001：÷2 010：÷4 011：÷8 100：÷16 101：÷32 110：÷64 111：÷128
3 - 1	-	-	保留。
0	0	只读	LPWMG 时钟源选择。 0：系统时钟 1：IHRC 或者 IHRC*2（由 GPC2PWM.3 决定）

6.29.LPWMG 计数上限高位寄存器(*LPWMGCUBH*)，地址 = 0x38

位	初始值	读/写	描述
7 - 0	-	只读	LPWMG 上限寄存器。位[10:3]。

6.30.LPWMG 计数上限低位寄存器(*LPWMGCUBL*)，地址= 0x39

位	初始值	读/写	描述
7 - 6	-	只读	LPWMG 上限寄存器。位[2:1]。
5 - 0	-	-	保留。

6.31.LPWMG0/1/2 占空比高位寄存器(*LPWMGxDTH*, x=0/1/2)，地址 = 0x3A/0x3C/0x3E

位	初始值	读/写	描述
7 - 0	-	只读	LPWMG0/LPWMG1/LPWMG2 占空比值。位[10:3]。

6.32.LPWMG0/1/2 占空比低位寄存器(*LPWMGxDTL*, x=0/1/2)，地址 = 0x3B/0x3D/0x3F

位	初始值	读/写	描述
7 - 5	-	只读	LPWMG0/LPWMG1/LPWMG2 占空比值。位[2:0]。
4 - 0	-	-	保留

注意：必须先写入 LPWMGx 占空比低位寄存器，再写入 LPWMGx 占空比高位寄存器。（x=0/1/2）

6.33.触摸控制寄存器 2(*IFC2C*)，地址 = 0x20

位	初始值	读/写	描述
7	0	读/写	IFC 忙碌标志
6 - 2	-	-	保留
1	0	读/写	IFC 计数时钟选择： 0：32MHz 1：16MHz
0	0	读/写	0: mode_0。建议写 0。（默认 mode_0） 1: mode_1。

6.34. 触摸控制寄存器 (*IFCC*)，地址 = 0x21

位	初始值	读/写	描述	
7 - 5	-	读/写	数据	描述
			000	使能 PA0/TK0。0/1：停用/启用
			001	使能 PA3/TK1。0/1：停用/启用
			010	使能 PA4/TK2。0/1：停用/启用
			011	使能 PA5/TK3。0/1：停用/启用
			100	使能 PA6/TK4。0/1：停用/启用
			101	使能 PA7/TK5。0/1：停用/启用
			11x	停用 IFC 功能
4	0	只写	IFC 触摸计数使能，在转换结束时会自动清除	
3	0	读/写	LDO_Ready 标志，启用 LDO 前应先清零此位。硬件设置，软件清零。	
2	0	只读	IFC 触摸计数溢出标志	
1	-	-	保留。建议写 0。	
0	0	读/写	IFC 振荡器使能	

注意：TK 通道必须先设置使能，且延时 1uS 之后才可将 IFC 振荡器使能。在 IFC 振荡器使能后需延时 20uS 等待振荡稳定才可将 IFC 触摸计数使能。

6.35. 触摸按键充电计数高位寄存器 (*IFCCRH*)，地址 = 0x22

位	初始值	读/写	描述
7 - 0	-	只读	触摸按键充电计数的 IFCCR[15:8]

6.36. 触摸按键充电计数低位寄存器 (*IFCCRL*)，地址 = 0x23

位	初始值	读/写	描述
7 - 0	-	只读	触摸按键充电计数的 IFCCR[7:0]

6.37. LDO 控制寄存器 (*IFCLDO*)，地址 = 0x24

位	初始值	读/写	描述
7 - 3	-	-	保留
2	0	读/写	LDO 模块使能。启用前应先清零 IFCC.3 的 LDO_Ready 标志。 0：关闭；1：启用。
1 - 0	00	读/写	LDO 基准电压选择： 00：1V8, 1.8V 01：保留 10：1V7, 1.7V 11：1V6, 1.6V

注意：LDO 启用前必须先启用 ILRC 低频振荡。LDO 关闭后再重启的时间间隔必须大于 2 个 ILRC 或是 50uS。

6.38. 触摸电容充放电设置寄存器 (*CHDIS*)，地址 = 0x25

位	初始值	读/写	描述
7 - 4	-	读/写	0100：程序默认写入值
3 - 0	-	读/写	保留，程序填 0。

注意：在每一次 LDO Enable 且 LDO Ready 后必须重新设置 CHDIS = 0x40。

6.39. 触摸电容控制寄存器 (*EXCAP*)，地址 = 0x26

位	初始值	读/写	描述
7	-	读/写	保留，程序填 0。
6 - 4	-	读/写	保留，程序填 0。
3 - 0	00	读/写	IFC 触摸转换电容参考系数。设置范围：0x01 ~ 0x0F 0000：不可填 0001：触摸值大 (Mode_0) 1000：程序默认写入值 1111：触摸值小 (Mode_0) 其他：按需配置

注意：在每一次 LDO Enable 且 LDO Ready 后必须重新设置 excap 寄存器的值，默认填 0x08。

7. 指令

符号	描述
ACC	累加器（Accumulator 的缩写）
a	累加器（Accumulator 在程序里的代表符号）
sp	堆栈指针
flag	ACC 标志寄存器
I	立即数据
&	逻辑与
|	逻辑或
←	移动
^	异或
+	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)
M.n	只允许寻址在地址 0~0x3F (0~63) 的位置

7.1. 数据传输类指令

mov a, I	移动实时数据到累加器 例如： *mov* a, 0x0f; 结果： a ← 0fh 受影响标志位： Z:『不变』， C:『不变』， AC:『不变』， OV:『不变』
mov M, a	移动数据由累加器到内存 例如： *mov* MEM, a; 结果： MEM ← a 受影响标志位： Z:『不变』， C:『不变』， AC:『不变』， OV:『不变』
mov a, M	移动数据由内存到累加器 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 受影响标志位：Z:『不变』， C:『不变』， AC:『不变』， OV:『不变』

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 ; // 停用 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 的数据读取并加载到累加器。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：『不变』， C：『不变』， AC：『不变』， OV：『不变』
	应用范例：
	`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)，在 PMS160 要为 0`
	`mov hb@RAMIndex, a ; // 将指针存到 RAM (MSB)`
	`…`
	`idxm a, RAMIndex ; // 将 RAM 地址为 0x5B 的数据读取并加载累加器`

Idxm index, a	使用索引作为 RAM 的地址并将累加器的数据读取并加载到 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)，在 PMS160 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加载地址为 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内存
	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数据存到堆栈内存
	… // 中断服务程序
	… // 中断服务程序
	popaf ; // 将堆栈内存的数据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内存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

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』
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』
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』
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 例如： *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』

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 IO, a	累加器和 IO 寄存器执行逻辑 XOR，然把结果存到 IO 寄存器 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 例如：*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
	not mem ; // mem = 0xC8
	--- 	

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* pa.5 ; 结果：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	受影响的标志位：『不变』Z 『受影响』C 『不变』AC 『不变』OV
	应用范例 1（连续输出）：
	...
	set1 pac.0 ; // 设置 PA.0 作为输出
	...
	set0 flag.1 ; // C=0
	swapc pa.0 ; // 送 C 给 PA.0（位操作），PA.0=0
	set1 flag.1 ; // C=1
	swapc pa.0 ; // 送 C 给 PA.0（位操作），PA.0=1
	...
	应用范例 2（连续输入）：
	...
	set0 pac.0 ; // 设置 PA.0 作为输入
	...
	swapc pa.0 ; // 读 PA.0 的值给 C（位操作）
	src a ; // 把 C 移位给 ACC 的位 7
	swapc pa.0 ; // 读 PA.0 的值给 C（位操作）
	src a ; // 把新进 C 移位给 ACC 的位 7，上一个 PA.0 的值给 ACC 的位 6
	...
	---------- 	

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』
cneqsn *a, M*	比较累加器和 RAM 的值，如果不相等就跳到下一条指令。标志改变与(a ← a - M)相同 例如：*cneqsn* *a, MEM;* 结果：如果 a≠MEM, 跳到下一条指令 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*	比较累加器和立即数的值，如果不相等就跳到下一条指令。标志改变与(a ← a - I) 例如： *cneqsn* *a,0x55 ;* *inc* *MEM ;* *goto* *error ;* 结果： 如果 a≠0x55，然后 “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』
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 例如：*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』

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』

	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
	…
	---------- 	

engint	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』
disgint	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』
stopsys	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』
stopexe	CPU 停止。所有震荡器模块仍然继续工作并输出：但是系统时钟是被停用以节省功耗 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』
reset	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』
wdreset	复位看门狗 例如： *wdreset* ; 结果： 复位看门狗 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

7.8. 指令执行周期综述

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

7.9. 指令影响标志综述

指令	Z	C	AC	OV	指令	Z	C	AC	OV	指令	Z	C	AC	OV
mov a, I	-	-	-	-	*mov* M, a	-	-	-	-	*mov* a, M	Y	-	-	-
mov a, IO	Y	-	-	-	*mov* IO, a	-	-	-	-	*ldt16* word	-	-	-	-
stt16 word	-	-	-	-	*idxm* a, index	-	-	-	-	*idxm* index, a	-	-	-	-
xch M	-	-	-	-	*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	*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	-	-	-	-	*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 IO, a	-	-	-	-	*xor* a, M	Y	-	-	-	*xor* M, a	Y	-	-	-
not a	Y	-	-	-	*not* M	Y	-	-	-	*neg* a	Y	-	-	-
neg M	Y	-	-	-	*set0* IO.n	-	-	-	-	*set1* IO.n	-	-	-	-
set0 M.n	-	-	-	-	*set1* M.n	-	-	-	-	*ceqsn* a, I	Y	Y	Y	Y
ceqsn a, M	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
call label	-	-	-	-	*goto* label	-	-	-	-	*ret* I	-	-	-	-
ret	-	-	-	-	*reti*	-	-	-	-	*nop*	-	-	-	-
pcadd a	-	-	-	-	*engint*	-	-	-	-	*disgint*	-	-	-	-
stopsys	-	-	-	-	*stopexe*	-	-	-	-	*reset*	-	-	-	-
wdreset	-	-	-	-	*swapc IO.n*	-	Y	-	-	*ceqsn a, I*	Y	Y	Y	Y
cneqsn a, M	Y	Y	Y	Y										

7.10. BIT 定义

位寻址只能定义在 RAM 区地址的 0x00 到 0x3F。

8. 代码选项(Code Options)

选项	选择	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
EMI	Disable	停用 EMI 优化选项
	Enable	系统时钟会轻微调整以获得更好的 EMI 性能
LVR	14 阶	4.5V, 4.0V, 3.75V, 3.5V, 3.3V, 3.15V, 3.0V, 2.7V, 2.5V, 2.4V, 2.3V, 2.2V, 2.1V, 2.0V
Boot-up_Time	Slow	请参考第 4.1 节 t_{WUP} 和 t_{SBP}
	Fast	请参考第 4.1 节 t_{WUP} 和 t_{SBP}
Interrupt_Src0	PA.0	Inten.0 / intrq.0 用于 PA.0 中断
	PA.5	Inten.0 / intrq.0 用于 PA.5 中断
ICE_LPWM_INTR （仅仿真时用）	As_G01	仿真时 LPWM 中断源来自 LPWMG0/1，不影响实际 IC
	As_G2	仿真时 LPWM 中断源来自 LPWMG2，不影响实际 IC

表 8：代码选项

9. 特别注意事项

此章节提醒用户在使用 PMS160 系列 IC 时避免常犯的一些错误。

9.1. 警告

用户必须详细阅读所有与此 IC 有关的 APN，才能使用此 IC。有关下载此 IC 的 APN，请访问公司网站：http://www.padauk.com.tw/cn/technical/index.aspx

9.2. 使用 IC

9.2.1. IO 引脚的使用和设定

(1) IO 作为数字输入时

- IO 作为数字输入时，Vih 与 Vil 的准位，会随着电压与温度变化，请遵守 Vih 的最小值，Vil 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

- 设置 IO 为输入
- 用 PADIER 寄存器，将对应的位设为 1

(3) PA5 设置为 PRSTB 输入引脚

- 设定 PA5 作输入
- 设定 CLKMD.0=1 来启用 PA5 作为 PRSTB 输入引脚

(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 >33Ω
- 应尽量避免使用 PA5 作为输入

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 寄存器资料，并在 RETI 之前，使用 POPAF 指令复原，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。

9.2.3. 系统时钟选择

利用 CLKMD 寄存器可切换系统时钟源。**请注意，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，然后再通过 CLKMD 寄存器关闭 A 时钟振荡源。

◆ 例一：系统时钟从 ILRC 切换到 IHRC/8

```
CLKMD   =   0x3C;           // 切到 IHRC，但 ILRC 不要停用
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 会死机
```

9.2.4. 掉电模式、唤醒和看门狗

当 ILRC 关闭时，看门狗也会失效。

9.2.5. TIMER 溢出

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) IHRC 频率会在使用烧录器烧录程序时校准。

(2) 校准 IHRC 时，不管是封装片机台刻录还是 COB 在线刻录，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) 频率偏离较大的情况一般发生在 COB 刻录或者 QTP 时。应广科技对这种情况不担负责任。

(4) 客户可根据自己的经验做一些调整，例如，可以将 IHRC 频率预先设置高 0.5%到 1%以让最终的实际频率更符合原来的期望。

9.2.7. LVR

LVR 水平的选择在程序编译时进行。使用者必须结合单片机工作频率和电源电压来选择 LVR，才能让单片机稳定工作。

下面是工作频率、电源电压和 LVR 水平设定的建议：

SYSCLK	VDD	LVR
2MHz	≧ 2.0V	≧ 2.0V
4MHz	≧ 2.5V	≧ 2.5V
8MHz	≧ 3.0V	≧ 3.0V

表 9：LVR 设置参考

(1) 只有当 IC 正常起动后，设定 LVR（2.0V ~ 4.5V）才会有效。

(2) 可以设定寄存器 MISC.2 为 1 将 LVR 关闭，但此时应确保 V_{DD} 在最低工作电压以上，否则 IC 可能工作不正常。

(3) 在省电模式 stopexe 和掉电模式 stopsys 下，LVR 功能无效。

9.2.8. 烧录方法

PMS160 的烧录脚为 PA3，PA4，PA5，PA6，VDD 和 GND 这 6 个引脚。

请使用 PDK5S-P-003 或以后的版本烧录 PMS161 实际芯片（PDK3S-P-002 或之前的版本皆以不支持烧录该芯片）。

- 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 （VPP）可能高于 6.5V。
  (2) VDD 可能高于 9.8V，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 VDD 相同。

请用户自行确认在使用本产品于合封或在板烧录时，周边元件及电路不会被上述电压破坏，也不会钳制上述电压。

9.2.8.1. PDK5S-P-003 烧录 PMS160 方法

使用 PDK5S-P-003 烧录 PMS160 的所有封装都需要特殊转档，接下来以 PMS160-S08B 的烧录为例，其他封装需对应修改“package setting”界面和 Jumper7 跳线方法即可。

1. PDK 转档

IDE 连接烧录器后点击 convert to package，打开待烧 PDK 进入 package setting 页面，在 package 选项选择带有[P003]后缀的封装，勾选 O/S test only program pin，确认勾选 VDD/PA5 swap 选项，确认 IC 脚位信息，保存并使用新生成的 PDK 文件。步骤参考图 25、图 26。

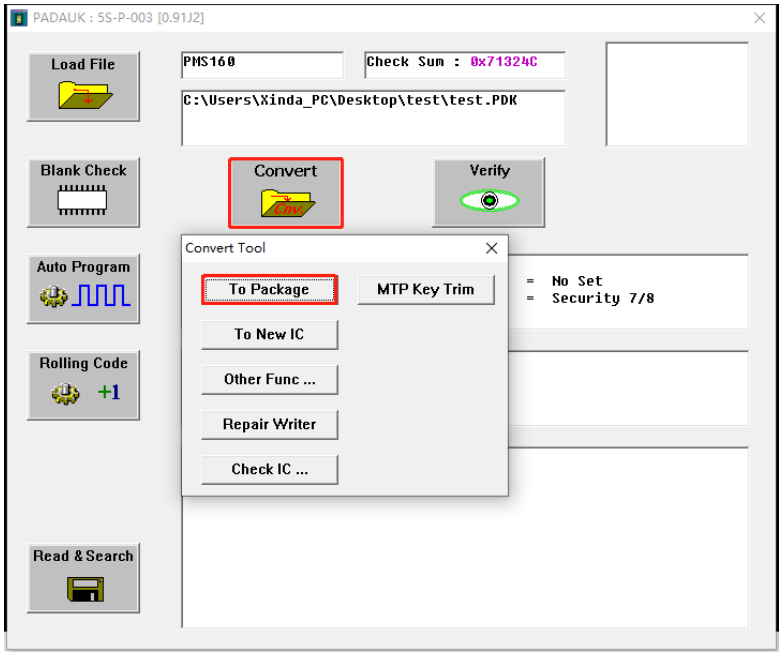


图 25：Convert PDK

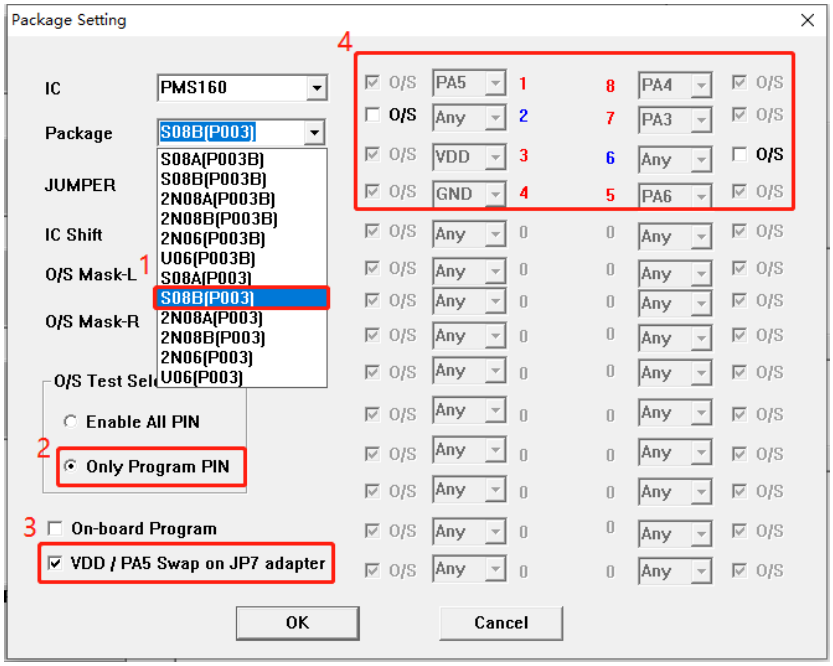


图 26：PMS160-S08B 在 P003 转档配置

2. 烧录器背面 JP7 跳线

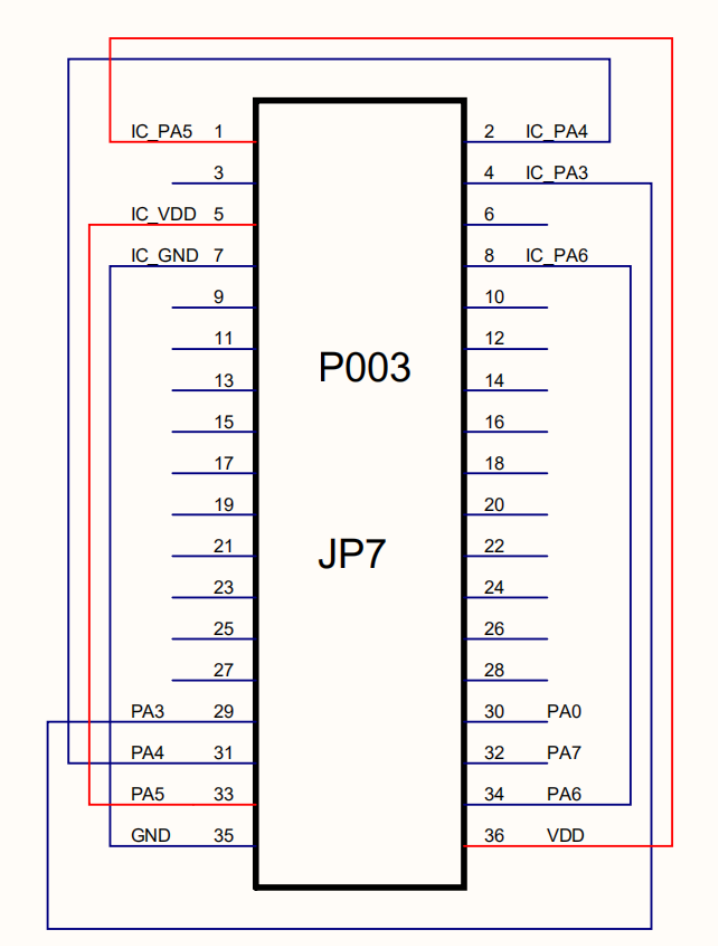


图 27：使用 P003 时，PMS160-S08B JP7 跳线原理图

注：在 P003 烧录跳线时，VDD 和 PA5 需要互换，即烧录器 VDD 引脚连接到 IC PA5 引脚，烧录器 PA5 引脚连接到 IC VDD 引脚。

3. 芯片顶格放入正面插座，提示 IC ready 后即可烧录。

9.2.8.2. PDK5S-P-003B 烧录 PMS160 方法

1. 使用 PDK5S-P-003B 烧录 PMS160 时，只有 S08A 封装是烧录器背面 jumper 插 JP2 位置，正面烧录插座往下空移 4 格，即可烧录。而其他封装都需要转档，使用 Jumper7 跳线。接下来以 PMS160-S08B 的烧录为例，其他封装需对应修改“package setting”界面和 Jumper7 跳线方法即可。封装设置如图 28 所示：

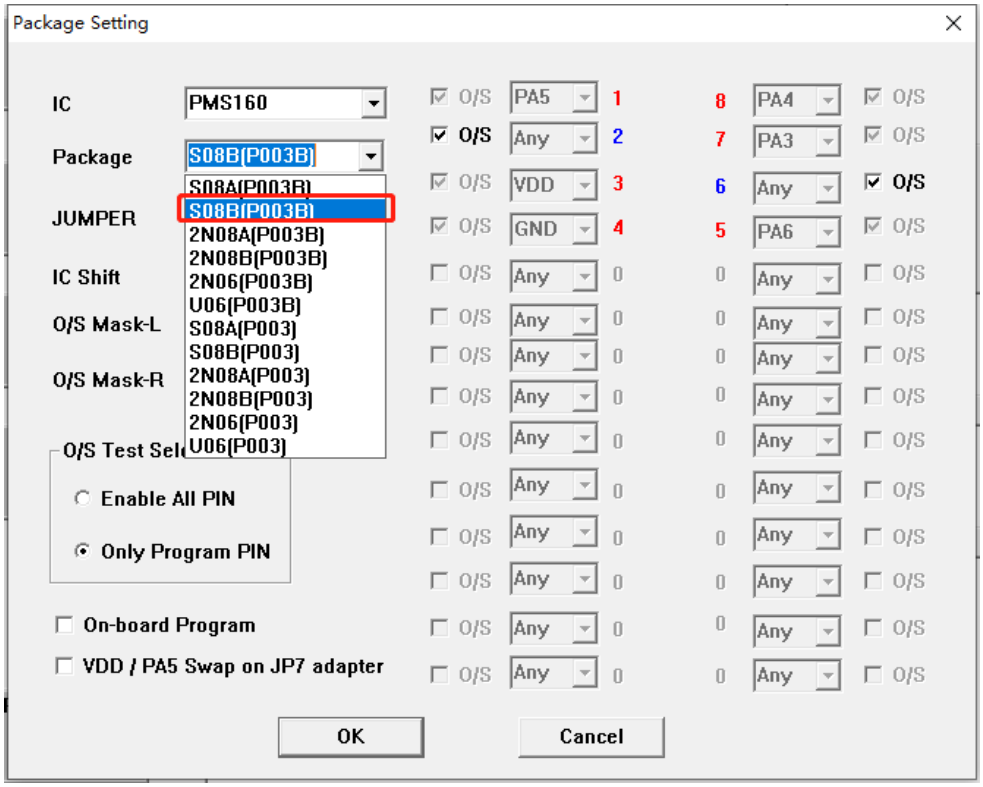


图 28：PMS160-S08B 在 P003B 转档配置

2. 烧录器背面 JP7 跳线，如图 29 所示：

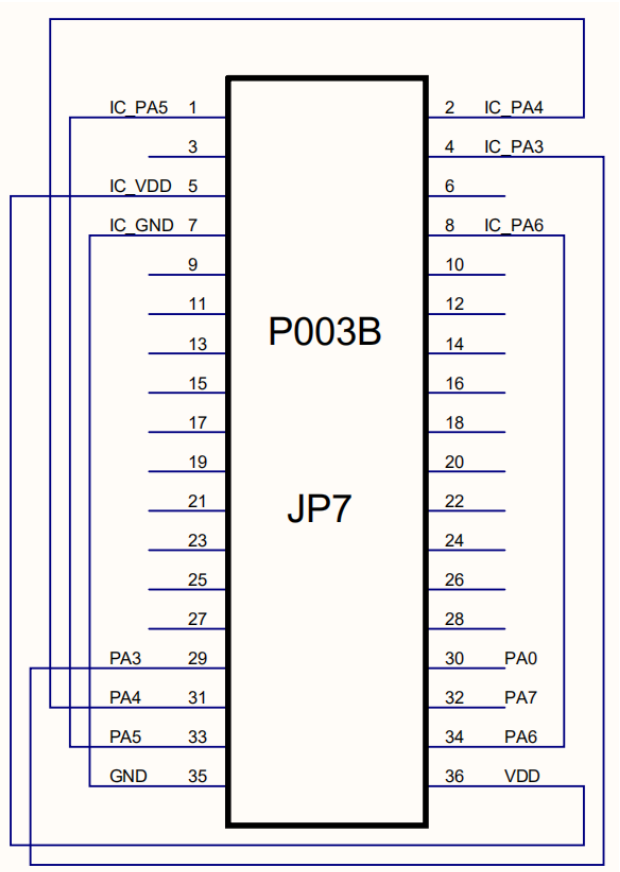


图 29：使用 P003B 时，PMS160-S08B JP7 跳线原理图

注：使用 P003B 烧录器跳线时，VDD 和 PA5 不需要互换。

3. 芯片顶格放入正面插座，提示 IC ready 后即可烧录。

9.3. 使用 ICE

请使用 PDK5S-I-S01/2(B)外挂触摸板 5S-I-TB002（又名 PMS160_ICE_Touch kit）对 PMS160 进行仿真。参考如下图片：

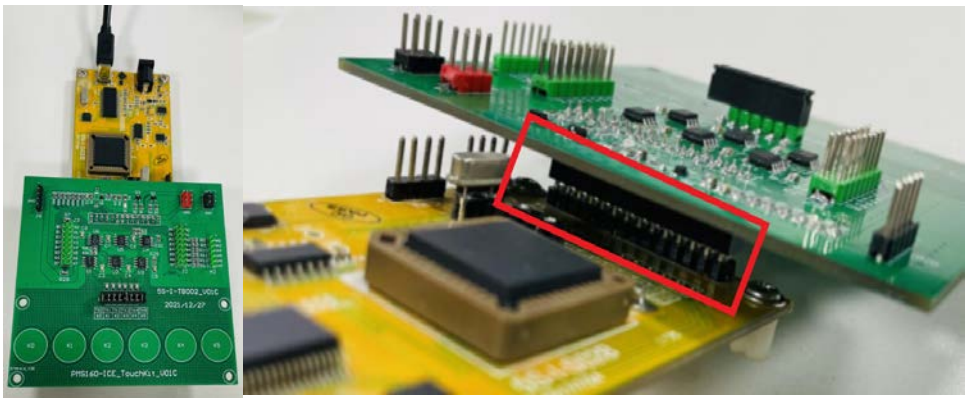


图 30：PMS160 仿真工具连接示意图

仿真时请注意以下事项：

(1) 关于仿真通讯

该触摸仿真板采用外挂 PMS160 实际 IC 的方式仿真，各功能模块的仿真与实际 IC 更吻合，但因与主仿真器做通讯，仿真速度将会略微减慢，因此也不建议过于快速且频繁的进入中断（仿真进中断的频率建议不超过 1KHz）。实际 IC 则不受此限。

受仿真通讯影响，IHRC/ILRC 的振荡器模块将始终保持振荡状态，在程序中设定关闭 IHRC/ILRC 也将只会对实际 IC 有效，故在使用 IHRC/ILRC 作为各 Timer 时钟源时尤其要注意此差异。

(2) 关于仿真电压

该触摸板的仿真电压范围限制在 3.5V ~ 4.8V，若搭配 PDK5S-I-S01 标准仿真器且仿真器采用由外部供电的方式仿真时，则外供电源不建议超过 5V，否则可能会因仿真器与触摸板电压不匹配导致无法仿真或仿真结果异常。

(3) 关于各 Timer 计数器（Timer2/3/16、LPWMG0/1/2）

不支持 Timer3、LPWMG0/1/2 选择 SYSCLK 作为计数器时钟源。

不支持 Timer2、Timer3 选择 GPCRS 作为计数器时钟源。

不支持 Timer2 选择 NILRC 作为计数器时钟源。

LPWMG2 与 LPWMG0/1 分属两颗不同的 IC 进行仿真，仿真的频率/相位会略有差异，且不支持带互补死区的 LPWM 功能仿真；当使用 LPWM 中断时，仿真器也会额外增加一条 Code Option：ICE_LPWM_INTR，供选择仿真时的 LPWMG 中断来源：来自 LPWMG0/1 或来自 LPWMG2。

当仿真输出 LPWM 时，其设定的输出脚位会被强制切换为输入状态（透过 pac 寄存器强制切换，不影响 padier、paph 等寄存器的原本设定值）。关闭 LPWM 输出后，相应端口的输入/输出配置（pac 寄存器）需要用户自行切换。对于实际 IC 则无需此操作。

(4) 关于省电和掉电：stopexe/stopsys

仿真时，省电/掉电模式的唤醒时间与实际 IC 不一致，且和系统时钟频率快慢有关。以系统时钟 1MHz 为例，stopexe 的仿真唤醒时间预估在 30us~1ms 之间；stopsys 唤醒则大约需要 700us。若有相关唤醒时间要求，建议采用实际 IC 测量。

考虑到仿真通讯对 IHRC/ILRC 的影响，为避免 Timer 误唤醒，建议客户使用 stopexe/stopsys 前关闭所有不需要用来唤醒的 Timer 模块，而非关闭 Timer 时钟源。以 Timer2 为例，建议程序写法：“$ TM2C STOP”。