PMS150C 系列

8 位 IO 类型单片机

数据手册

第 0.02 版

2016 年 11 月 3 日

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

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

重要声明

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

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

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

PMS150C 不适用于交流供电阻容降压或者电源纹波大，EFT 要求高的应用，请注意不要将 PMS150C 用于这种特殊要求的产品中。

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

目 录

修订历史：

修 订	日 期	描 述
0.01	2016/3/17	初版
0.02	2016/11/3	1. 修改第二章方框图。 2. 修改第 4.1 节直流交流电气特性。 3. 增加第 5.5 节比较器的描述。 4. 增加第 5.7 节 8 位计数器(Timer2)的描述。

PMS150B 和 PMS150C 主要差异表

PMS150B 与 PMS150C 主要差异列举如下：

项目	功能	PMS150B	PMS150C
1	ILRC 频率	110KHz@5.0V，25°C	62KHz@5.0V，25°C （VDD 变化对 ILRC 有影响）
2	LVR	2.8V,2.2V,2.0V	4.0V,3.5V,3.0V,2.75V 2.5V,2.2V,2.0V,1.8V
3	RAM	60 bytes	64 bytes
4	PA5 口输入模式上拉电阻	没有	有
5	工作温度	0°C ~70°C	-20°C ~70°C
6	省电模式功耗（stopexe）	40uA@3.3V	3 uA@3.3V
7	IO 输出电流	17mA/-7mA@5.0V	普通模式：14.5mA/-10.5mA@5.0V 低驱动模式：5mA/-3.5mA@5.0V
8	看门狗定时器溢时	4096,16384,65536 ILRC 时钟周期	8192,16384,65536,262144 ILRC 时钟周期
9	唤醒时间	快速模式：1024 T_{IHRC} 普通模式：1024 T_{ILRC}	快速模式：32 T_{ILRC} 普通模式：2048 T_{ILRC}
10	开机时间	快速模式：2048 T_{IHRC} 普通模式：1024 T_{ILRC}	快速模式：32 T_{ILRC} 普通模式：2048 T_{ILRC}
11	系统保留 OTP 区	0x3F8~0x3FF(8 word)	0x3F0~0x3FF(16 word)
12	ILRC 做系统时钟源	ILRC,ILRC/4	ILRC,ILRC/4,ILRC/16
13	支持 ICE 类型	PDK3S-I-001/002/003, 5S-I-S01	5S-I-S01
14	8 位计数器 2(含 PWM)	没有	有
15	比较器	没有	有

1. 单片机特点

1.1. 系统功能

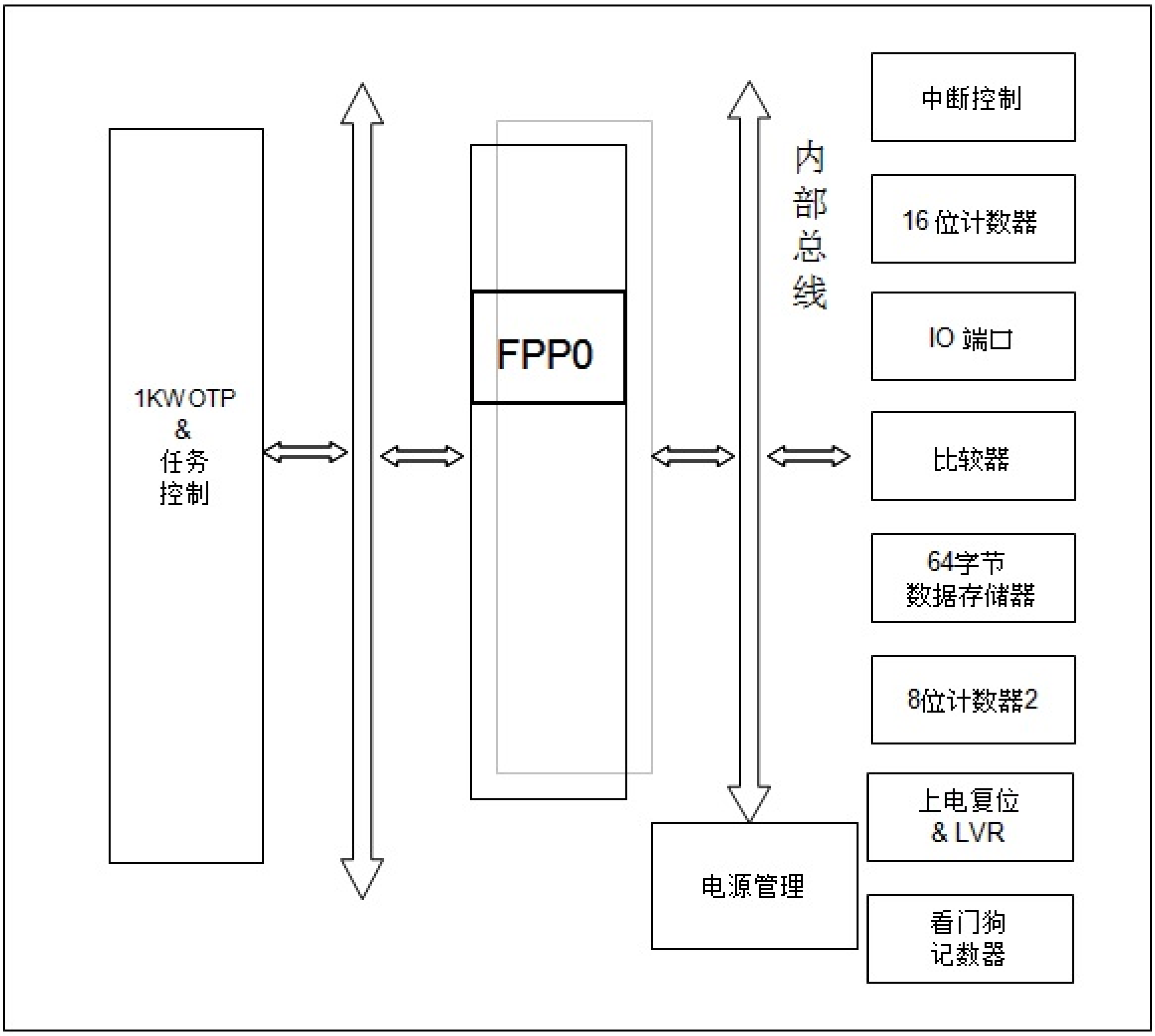
- 时钟模式：内部高频振荡器、内部低频振荡器
- 硬件 16 位定时器
- 1 个 8 位硬件 PWM 生成器
- 1 个通用比较器
- 快速唤醒功能
- 6 个带输入上拉电阻 IO 引脚，且做输出时具有可选的电流驱动能力
- 1 个外部中断输入引脚
- 每个引脚都可弹性设定唤醒功能
- 8 级 LVR 可选
- 工作频率
  0 ~ 8MHz@VDD≧3V; 0 ~ 4MHz@VDD≧2.2V; 0 ~ 2MHz@VDD≧2.0V;
- 工作电压：2.0V ~ 5.5V
- 工作温度：-20°C ~70°C
- 功耗特性：
  $I_{operating}$ ~ 0.3mA@1MIPS, VDD=3.3V
  $I_{operating}$ ~ 13uA@ILRC=62KHz, VDD=3.3V
  $I_{powerdown}$ ~ 0.5uA@VDD=3.3V

1.2. CPU 特点

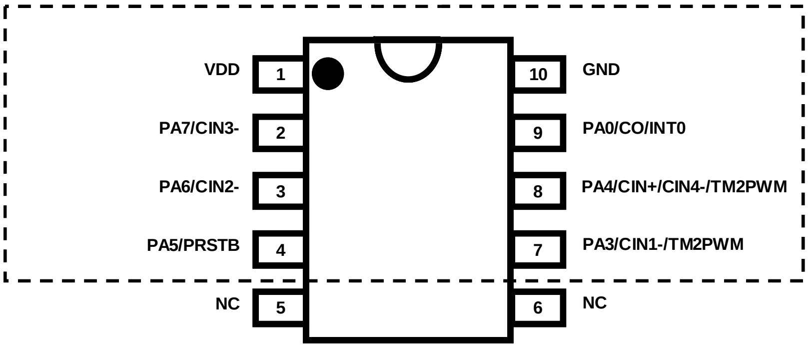
- 工作模式：单一处理单元的工作模式
- 1KW OTP 程序内存
- 64 字节数据存储器
- 提供 79 条指令
- 绝大部分指令都是单周期（1T）指令
- 可程序设定的堆栈深度
- 所有的数据存储器都可当数据指针（index pointer）
- 独立的 IO 地址以及存储地址方便程序开发

2. 系统概述和方框图

PMS150C 是一个 IO 类型、完全静态，以 OTP 为程序存储基础的单片机。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。PMS150C 内置 1KW OTP 程序内存以及 64 字节数据存储器；另外，PMS150C 还提供一个 16 位的硬件计数器、一个 8 位的硬件 PWM 生成器和一个通用比较器。



3. 引脚功能说明



引脚名称	引脚&缓冲器类型	功能描述
PA7	IO ST / CMOS	当端口 A 位 7，此引脚可编程设定为输入或输出，弱上拉电阻模式。 此外，亦可设定在睡眠中唤醒系统的功能；但是，当寄存器 *padier* 位 7 为”0”时，唤醒功能是被关闭的。
PA6	IO ST / CMOS	当端口 A 位 6，此引脚可编程设定为输入/输出，弱上拉电阻模式。 此外，亦可设定在睡眠中唤醒系统的功能；但是，当寄存器 *padier* 位 6 为”0”时，唤醒功能是被关闭的
PA5/PRST#	IO ST / CMOS	此引脚可用做： （1）当单片机的外部复位。 （2）当端口 A 位 5，此引脚可以设定为输入口或开漏输出（open drain）模式。 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。<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	IO ST / CMOS	此引脚可用做端口 A 位 4，并可编程设定为输入或输出，弱上拉电阻模式。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *padier* 位 4 为”0”时，唤醒功能是被关闭的。
PA3	IO ST / CMOS	此引脚可用做端口 A 位 3，并可编程设定为输入或输出，弱上拉电阻模式。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *padier* 位 3 为”0”时，唤醒功能是被关闭的。
PA0/INT0	IO ST / CMOS	此引脚可用做： （1）当端口 A 位 0，并可编程设定为输入或输出，弱上拉电阻模式。 （2）外部引脚中断 0，中断服务可发生在上升沿或下降沿。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *padier* 位 0 为”0”时，唤醒功能是被关闭的。
VDD		正电源
GND		地
注意 ：IO：输入/输出 ； ST：施密特触发器输入 ； 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标明外，都基于 VDD=5.0V, f_{SYS}=2MHz 的条件下获得。

符 号	特 性	最小值	典型值	最大值	单位	条 件
V_{DD}	工作电压	2.0		5.5	V	
f_{SYS}	系统时钟(CLK)* = IHRC/2 IHRC/4 IHRC/8 ILRC	 0 0 0 	 62K	 8M 4M 2M 	Hz	 V_{DD} ≧ 3.0V V_{DD} ≧ 2.2V V_{DD} ≧ 2.0V V_{DD} =5.0V
I_{OP}	工作电流		0.3 13		mA uA	f_{SYS}=IHRC/16=1MIPS@3.3V f_{SYS}=ILRC=62kHz@3.3V
I_{PD}	掉电模式消耗电流 (用 *stopsys* 命令)		0.5		uA	f_{SYS}= 0Hz, V_{DD} =3.3V
I_{PS}	省电模式消耗电流 (用 *stopexe* 命令)		3		uA	V_{DD} =3.3V; Band-gap, LVR, IHRC 关闭,ILRC 打开
V_{IL}	输入低电压	0		0.2V_{DD}	V	
V_{IH}	输入高电压	0.6 V_{DD}		V_{DD}	V	
I_{OL}	IO 引脚输出灌电流 普通模式 低驱动模式	 10 3.5	 14.5 5.0	 19 6.5	mA	V_{DD}=5.0V, V_{OL}=0.5V
I_{OH}	IO 引脚输出驱动电流 普通模式 低驱动模式	 -7.5 -2.6	 -10.5 -3.5	 -13.5 -4.4	mA	V_{DD}=5.0V, V_{OH}=4.5V
V_{IN}	输入电压	-0.3		VDD+0.3	V	
$I_{INJ\ (PIN)}$	脚位的引入电流			1	mA	V_{DD} +0.3≧V_{IN}≧ -0.3
R_{PH}	上拉电阻		10 220		KΩ	V_{DD}=5.0V V_{DD}=3.3V
f_{IHRC}	IHRC 输出频率(校准后)*	15.76*	16*	16.24*	MHz	@25°C
		15.20*	16*	16.80*		V_{DD}=2V~5.5V,-20°C <Ta<70°C*
f_{ILRC}	ILRC 输出频率*		62*		KHz	V_{DD} =5.0V, -20°C <Ta<70°C*
t_{INT}	中断脉冲宽度	30			ns	V_{DD} = 5.0V
V_{DR}	数据存储器数据保存电压*	1.5			V	关机模式

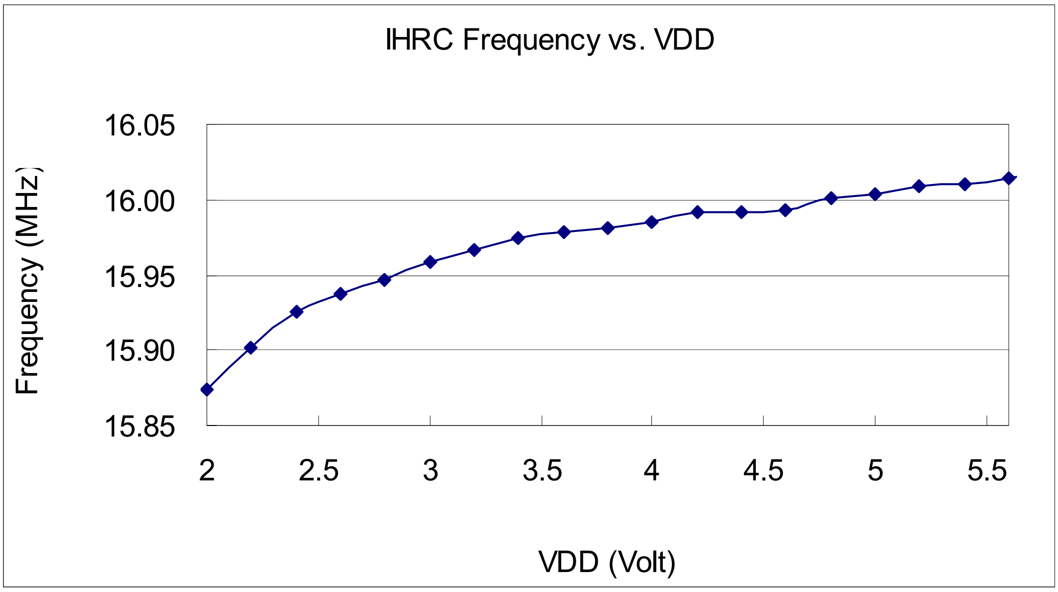
符 号	特 性	最小值	典型值	最大值	单位	条 件
t_{WDT}	看门狗超时溢出时间		8192		ILRC 时钟周期	misc[1:0]=00 (默认)
			16384			misc[1:0]=01
			65536			misc[1:0]=10
			262144			misc[1:0]=11
t_{SBP}	系统开机时间(快速开机)		780 780		us	@ V_{DD} =5V @ V_{DD} =2.5V
	系统开机时间(正常开机)		47 47		ms	@ V_{DD} =5V @ V_{DD} =2.5V
t_{WUP}	快速唤醒下的唤醒时间(misc.5=1)		32		T_{ILRC}	T_{ILRC} 为 ILRC 振荡周期
	普通唤醒下的唤醒时间(misc.5=0)		2048		T_{ILRC}	T_{ILRC} 为 ILRC 振荡周期
t_{RST}	外部复位脉冲宽度	120			us	@ V_{DD} =5V
CPos	比较器偏压*		±10	±20	mV	
CPcm	比较器共模输入电压*	0		VDD+1.5	V	
CPspt	比较器响应时间*		100	500	ns	上升沿和下降沿一样
CPmc	比较器模式改变稳定时间		2.5	7.5	us	
CPcs	比较器电流消耗		20		uA	V_{DD} = 3.3V

***** 这些参数是设计参考值，并不是每个芯片测试结果。

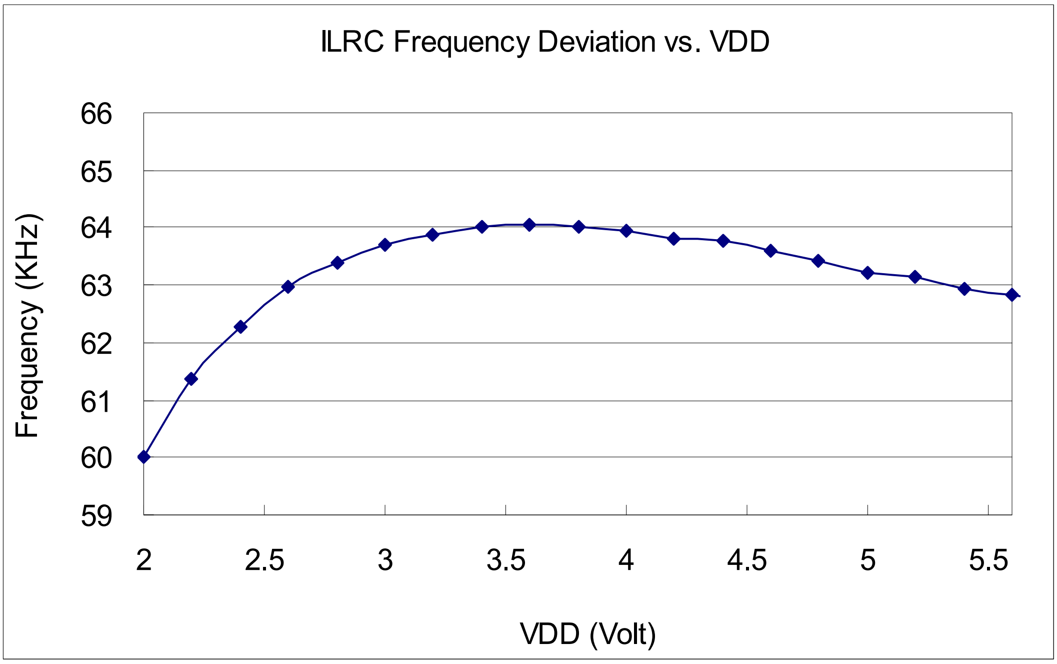
4.2 工作范围

- 电源电压 …………………………………...... 2.0V ~ 5.5V（最高 5.5V）
  如果输入电压高过 5.5V，可能造成 IC 损坏
- 输入电压 ………………………………….... -0.3V ~ VDD + 0.3V
- 工作温度 ……………………………………. -20°C ~ 70°C
- 储藏温度 ……………………………………. -50°C ~ 125°C
- 结点温度 ……………………………………. 150°C

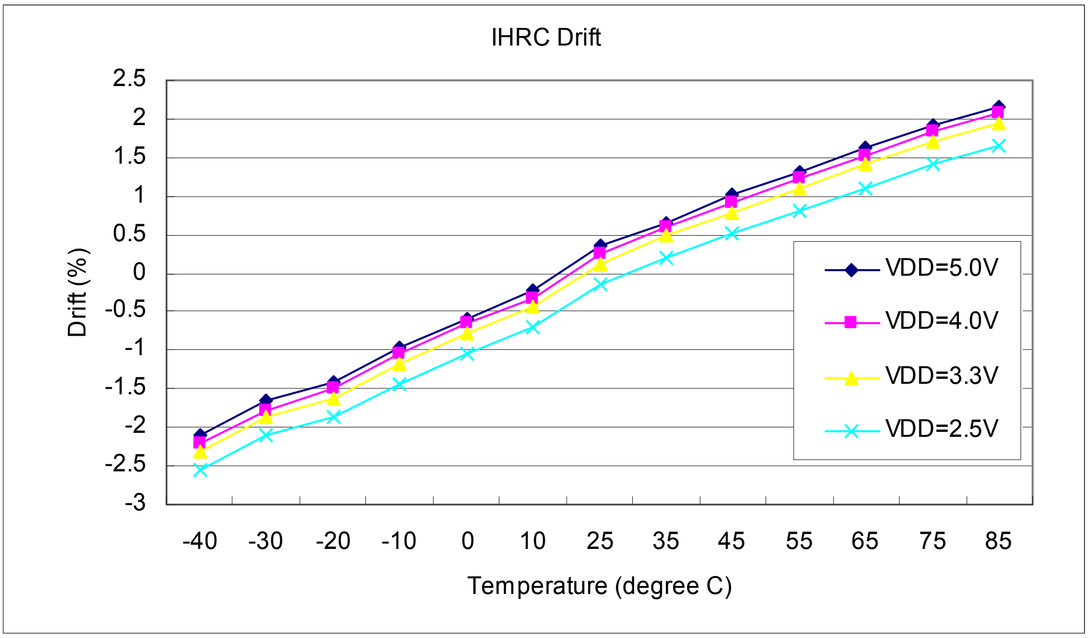
4.3 IHRC 频率与 VDD 关系曲线图



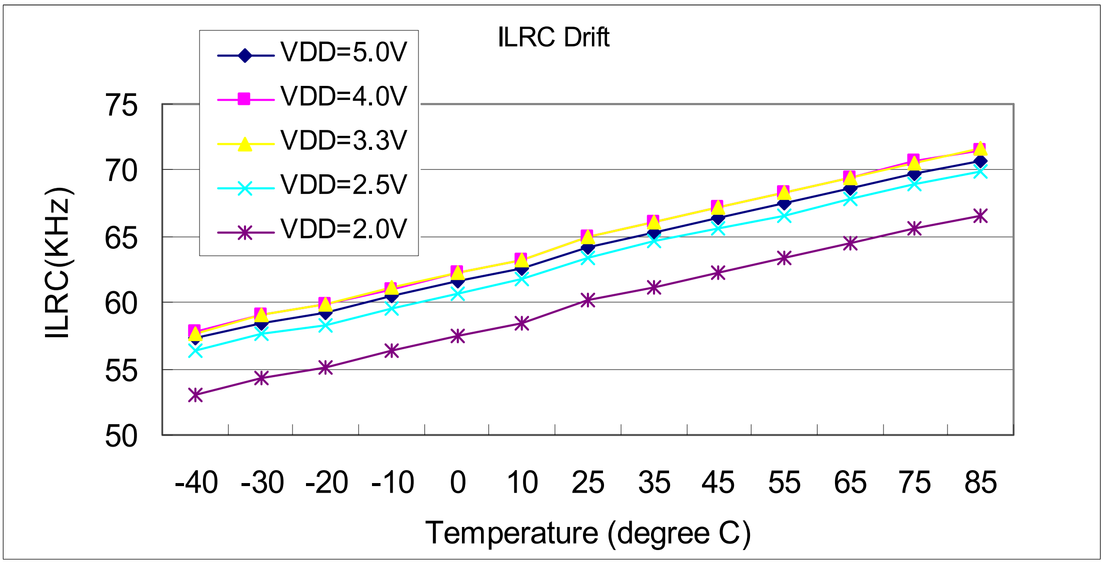
4.4 ILRC 频率与 VDD 关系曲线图



4.5 IHRC 频率与温度关系曲线图（校准到 16MHz）



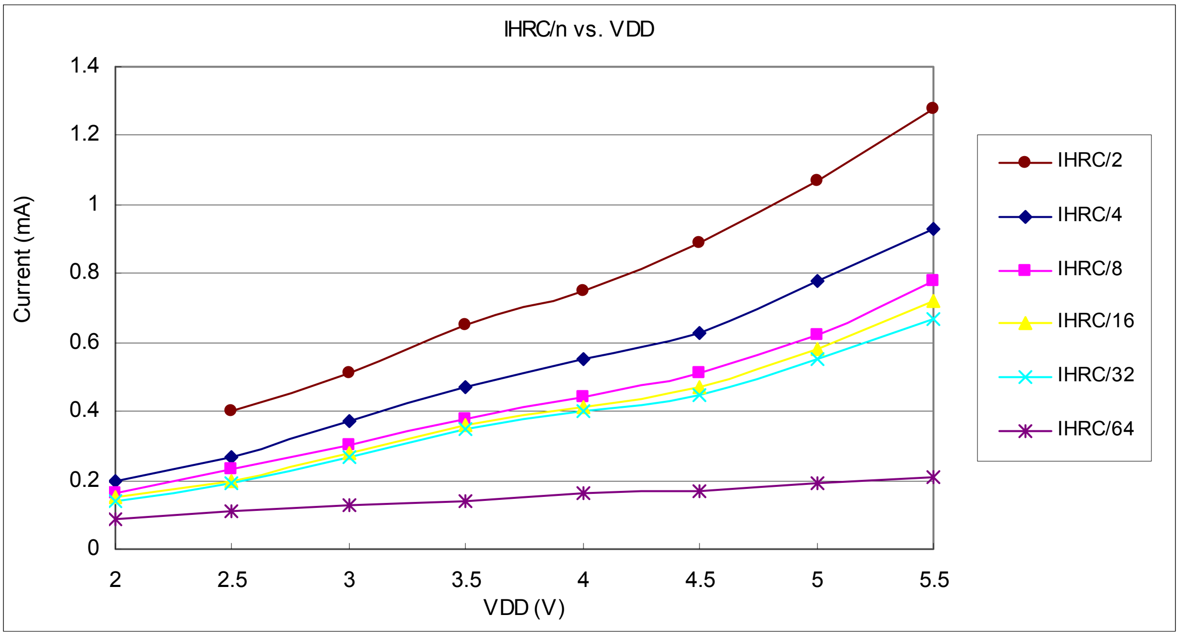
4.6 ILRC 频率与温度关系曲线图



4.7 工作电流与 VDD、系统时钟 CLK=IHRC/n 曲线图

条件=>开启的硬件模块：Band-gap, LVR, IHRC, T16; 关闭的硬件模块：ILRC;

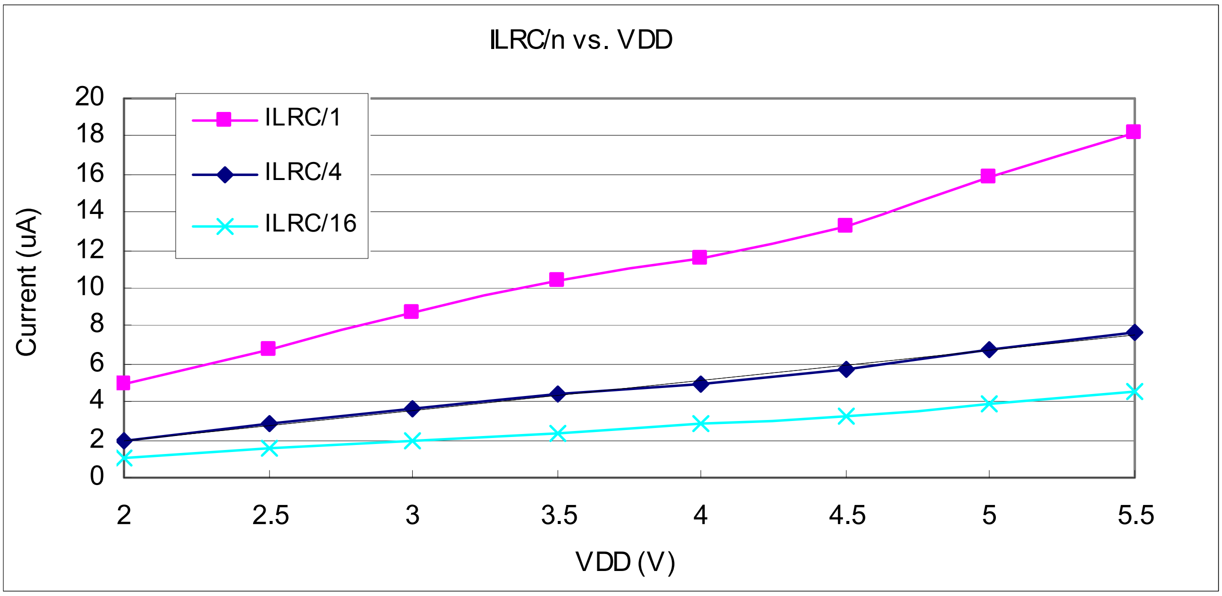
IO 引脚：PA0 输出 0.5Hz 频率的 PWM，无负载；其它引脚：设为输入且不悬空



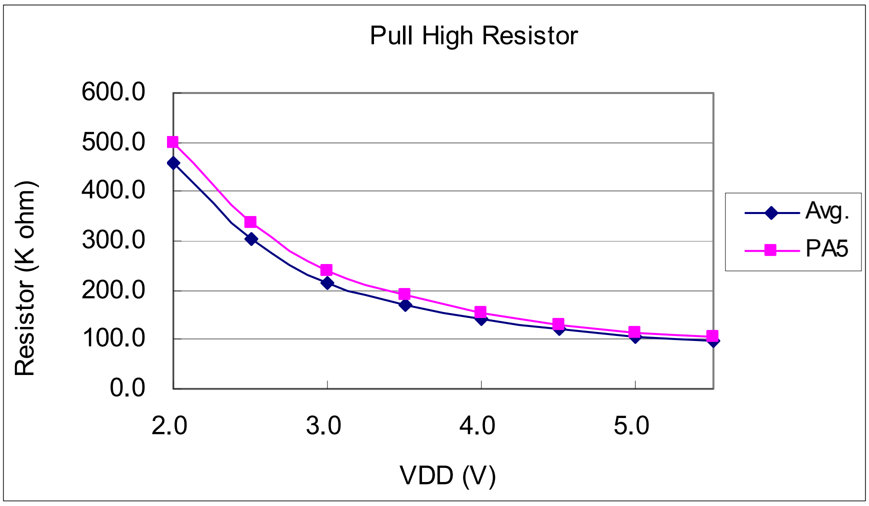
4.8 工作电流与 VDD、系统时钟 CLK=ILRC/n 曲线图

条件=>开启的硬件模块：T16; 关闭的硬件模块：Band-gap, LVR, ILRC, IHRC;

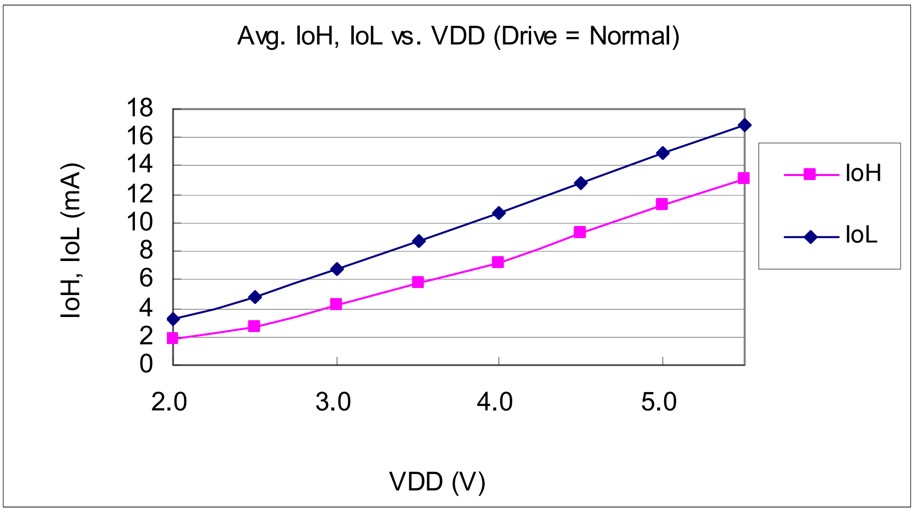
IO 引脚：PA0 以 0.5Hz 频率的 PWM，无负载；其它引脚：设为输入且不悬空

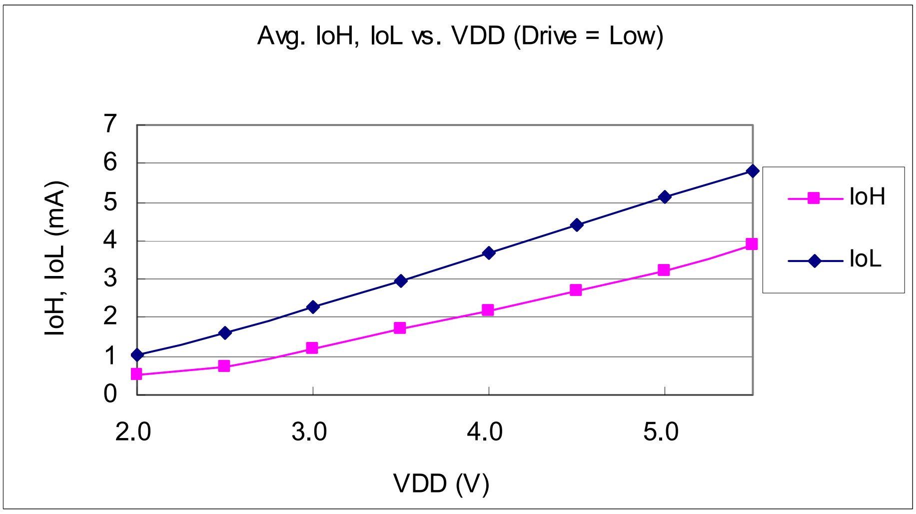


4.9 引脚上拉电阻曲线图

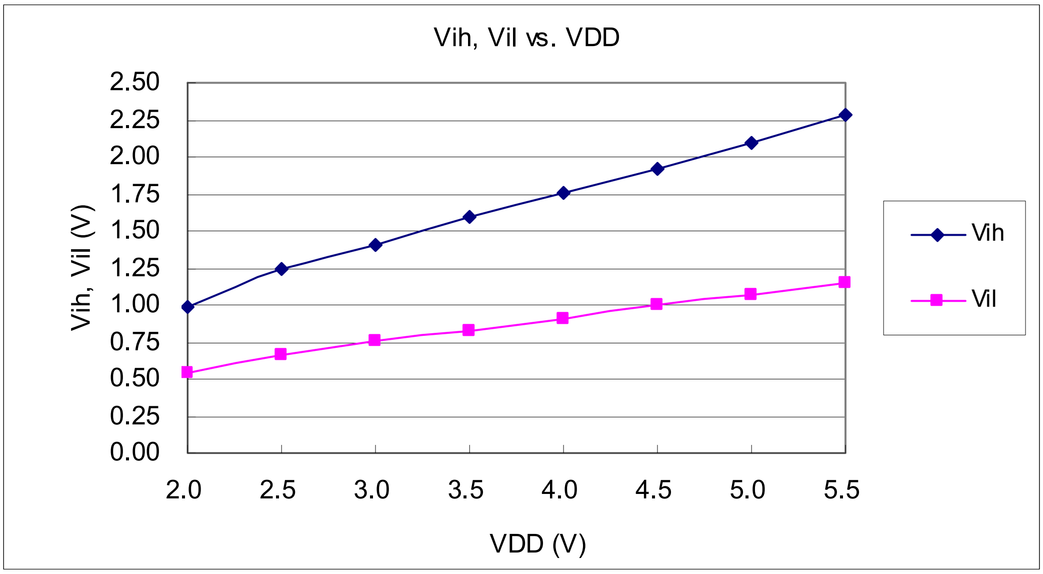


4.10 引脚输出驱电流(Ioh)与灌电流(Iol) 曲线图





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



5. 功能概述

5.1 程序内存—OTP

OTP（一次性可编程）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。中断入口是 0X010；OTP 程序内存最后 16 个地址空间是被保留给系统使用，如：校验，序列号等。PMS150C 的 OTP 程序内存容量为 1KW，如表 1 所示。 OTP 内存从地址"0x3F0 to 0x3FF"供系统使用，从"0x001~0x00F"和"0x011~0x3EF"地址空间是用户的程序空间。

地址	功能
0x000	FPP0 起始地址 – *goto* 指令
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
•	•
0x3EF	用户程序区
0x3F0	系统使用
•	•
0x3FF	系统使用

表 1：PMS150C 程序内存结构

5.2 开机流程

开机时，POR（上电复位）是用于复位 PMS150C；开机时间可以设置为快速模式或者普通模式，快速模式开机时间是 32 ILRC，普通模式开机时间是 2048 ILRC。用户必须确保开机时的电源稳定，开机流程如图 1 所示。

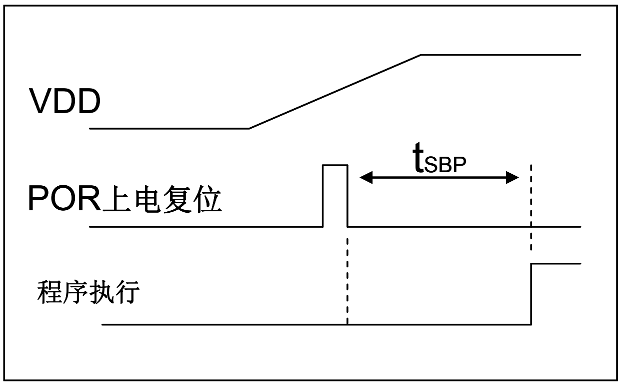


图 1：上电复位时序

5.3 数据存储器 – SRAM

数据存取可以是字节或位的操作。除了存储数据外，数据存储器还可以担任间接存取方式的资料指针，以及所有处理单元的堆栈内存。
堆栈内存是定义在数据储存器里。堆栈内存的堆栈指针是定义在堆栈指针寄存器；而每个处理单元的堆栈内存深度是由使用者定义的。用户可以依其程序需求来订定所需要堆栈内存的大小，以保持最大的弹性。

数据存储器的间接存取方式，是以数据存储器当作数据指针来存取数据字节。所有的数据存储器，都可以拿来当作数据指针，这可以让单片机的资源做最大的使用。由于 PMS150C 的数据存储器只有 64 字节，所以全部都可以用间接方式来存取。

5.4 振荡器和时钟

PMS150C 提供 2 个振荡器电路：内部高频振荡器（IHRC）与内部低频振荡器（ILRC）。这二个振荡器可以分别用寄存器 clkmd.4 与 clkmd.2 启用或禁用，使用者可以选择这二个振荡器之一作为系统时钟源，并透过 *clkmd* 寄存器来改变系统时钟频率，以满足不同的系统应用。

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

5.4.1 内部高频振荡器和内部低频振荡

开机后，IHRC 和 ILRC 振荡器都是被启用的，PMS150C 烧录工具提供 IHRC 频率校准，透过 *ihrcr* 寄存器来消除工厂生产引起的频率漂移，IHRC 振荡器通常被校准到 16MHz，通常校准后的频率偏差都在 2％以内；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；在 VDD =2.0V～5.5V，-20℃～70℃的条件下，总漂移率约为±5％，请参阅 IHRC 频率和 VDD、温度的测量图表。

5V 时 ILRC 的频率是 62KHz 左右，但是，其频率会因工厂生产、电源电压和温度而变化，请参阅 DC 规格书。需要精确定时的应用时请不要使用 ILRC 的时钟当作参考时间。

5.4.2 芯片校准

IHRC 的输出频率可能因工厂制造变化而有所差异，PMS150C 提供 IHRC 输出频率校准，来消除工厂生产时引起的变化。这个功能是在编译用户的程序时序做选择，校准命令以及选项将自动插入到用用户的程序，校准命令如下所示：

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

  这里：

  p1 =2，4，8，16，32；以提供不同的系统时钟。

  p2 =14～18; 校准芯片到不同的频率，通常选择 16MHz。

  p3 =2.2～5.5; 根据不同的电源电压校准芯片。

5.4.3 IHRC 频率校准与系统时钟

用户在程序编译期间，IHRC 频率校准以及系统时钟的选项，如表 2 所示：

SYSCLK	CLKMD	IHRCR	描述
○设置 IHRC / 2	= 34h (IHRC / 2)	被校准	IHRC 校准到 16MHz, CLK=8MHz (IHRC/2)
○设置 IHRC / 4	= 14h (IHRC / 4)	被校准	IHRC 校准到 16MHz, CLK=4MHz (IHRC/4)
○设置 IHRC / 8	= 3Ch (IHRC / 8)	被校准	IHRC 校准到 16MHz, CLK=2MHz (IHRC/8)
○设置 IHRC / 16	= 1Ch (IHRC / 16)	被校准	IHRC 校准到 16MHz, CLK=1MHz (IHRC/16)
○设置 IHRC / 32	= 7Ch (IHRC / 32)	被校准	IHRC 校准到 16MHz, CLK=0.5MHz (IHRC/32)
○设置 ILRC	= E4h (ILRC / 1)	被校准	IHRC 校准到 16MHz, CLK=ILRC
○Disable	不改变	不改变	IHRC 不校准, CLK 没改变

表 2：IHRC 频率校准选项

通常情况下，ADJUST_IC 将是开机后的第一个命令，以设定系统的工作频率。IHRC 频率校准的程序只会执行一次，是发生在要将程序代码在写入 OTP 内存的时候，以后，它就不会再被执行了。如果 IHRC 校准选择不同的选项，开机后的系统状态也是不同的。下面显示在不同的选项下，PMS150C 不同的状态：

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

开机后, CLKMD = 0x34：

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

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

开机后, CLKMD = 0x14：

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

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

开机后, CLKMD = 0x3C：

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

(4) .ADJUST_IC　　SYSCLK=IHRC/16, IHRC=16MHz, VDD=2.2V

开机后, CLKMD = 0x1C：

- IHRC 的校准频率为 16MHz@VDD=2.2V，启用 IHRC 的硬件模块
- 系统时钟 CLK = IHRC/16 = 1MHz
- 看门狗定时器被禁止，启用 ILRC, PA5 是在输入模式

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

开机后, CLKMD = 0x7C：

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

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

开机后, CLKMD = 0XE4：

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

5.4.4 系统时钟和 LVR 基准位

系统时钟的时钟源基于 IHRC 或 ILRC，PMS150C 的时钟系统的硬件框图如图 2 所示。

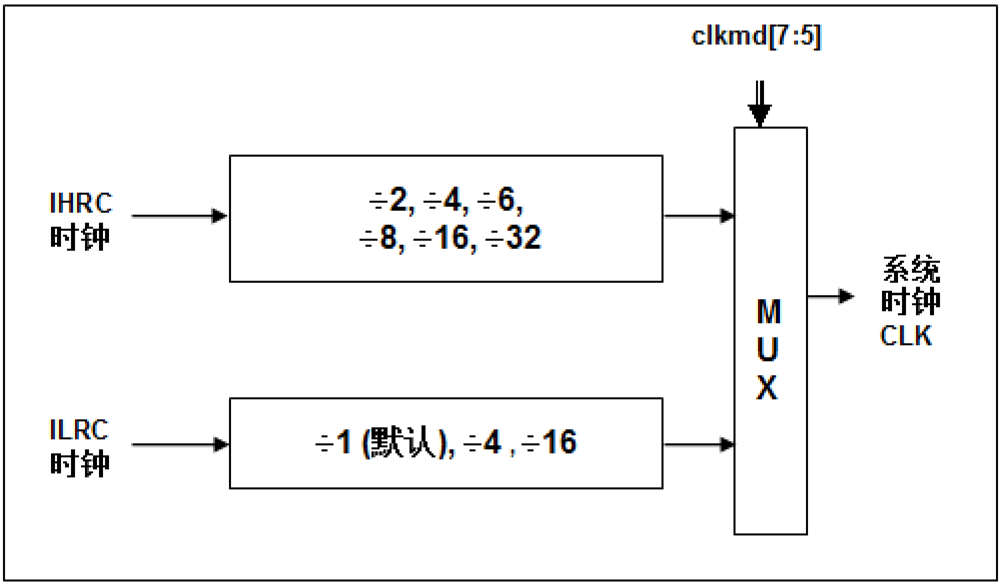


图 2：系统时钟源选择

- 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 的电压结合，才能使系统稳定。 LVR 的电压是在在编译过程中选择的。

5.5 比较器

PMS150C 内部内置了一个比较器，图 3 显示了它的硬件框图。它可以比较两个引脚之间的信号或与内部参考电压 $V_{internal\ R}$ 的信号或者 1.2V Band-gap 电压进行比较。进行比较的两个信号，一个是正输入，另一个是负输入。负输入可以是 PA3，PA4，PB6，PB7，band-gap 参考电压 1.20V，或 $V_{internal\ R}$，并由 ***gpcc*** 寄存器的位[3:1]来选择；正输入可以 PA4 或 $V_{internal\ R}$，由 ***gpcc*** 寄存器位 0 选择。比较器输出的结果可以选择性的送到 PA0；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，信号是否反极性也是可选的。比较输出结果可以用来产生中断信号或通过 gpcc 寄存器读取出来。

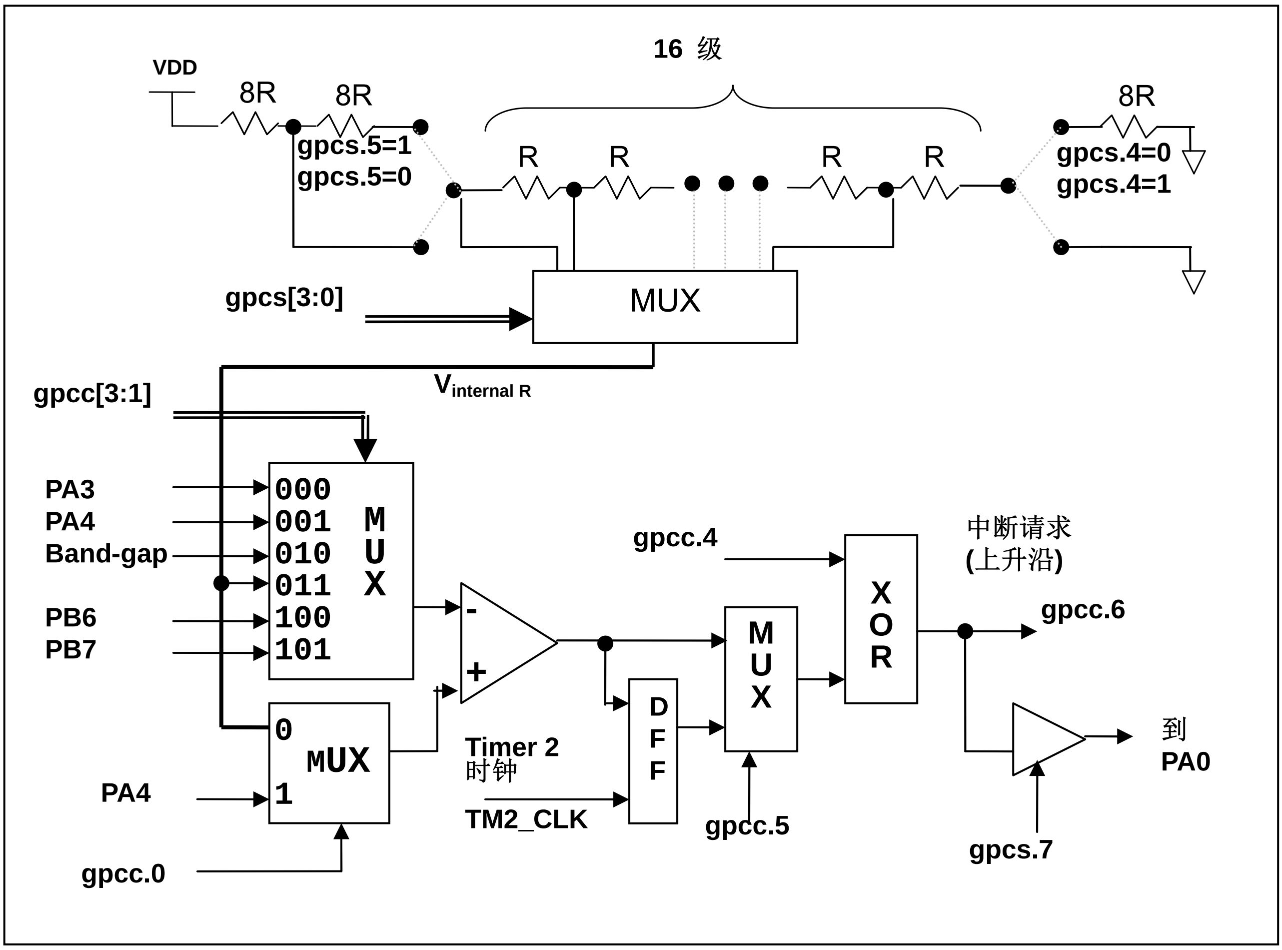
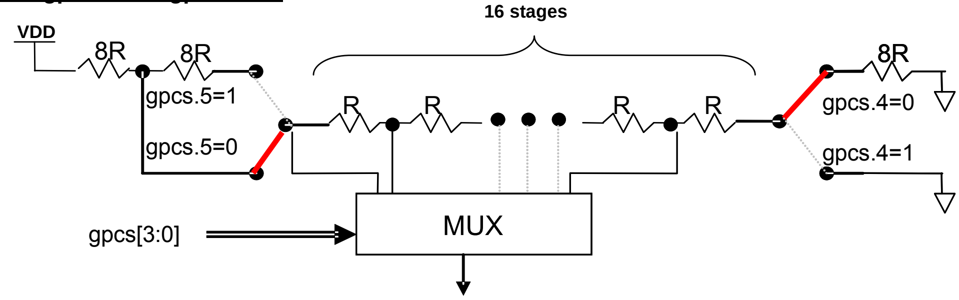


图 3：比较器硬件图框

5.5.1 内部参考电压（$V_{internal\ R}$）

内部参考电压 $V_{internal\ R}$ 是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 4 ~ 图 7 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)*VDD 到(3/4)*VDD。

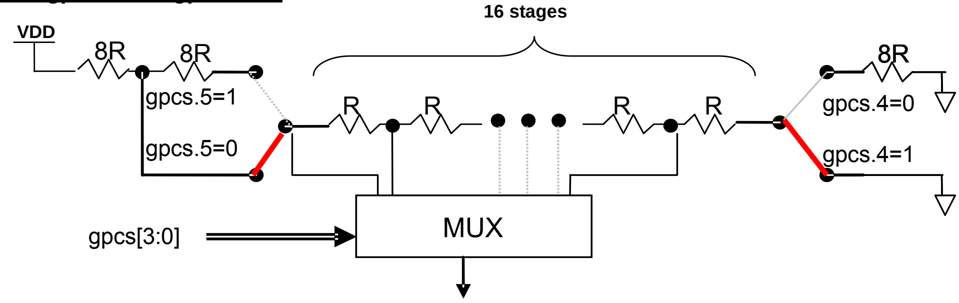


$$V_{internal\ R} = (3/4)\ VDD \sim (1/4)\ VDD + (1/32)\ VDD$$

@ gpcs[3:0] = 1111 ~ gpcs[3:0] = 0000

$$V_{internal\ R} = \frac{1}{4} * VDD + \frac{(n+1)}{32} * VDD,\ n = gpcs[3:0] \text{ in decimal}$$

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



$$V_{internal\ R} = (2/3)\ VDD \sim (1/24)\ VDD$$

@ gpcs[3:0] = 1111 ~ gpcs[3:0] = 0000

$$V_{internal\ R} = \frac{(n+1)}{24} * VDD,\ n = gpcs[3:0] \text{ in decimal}$$

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

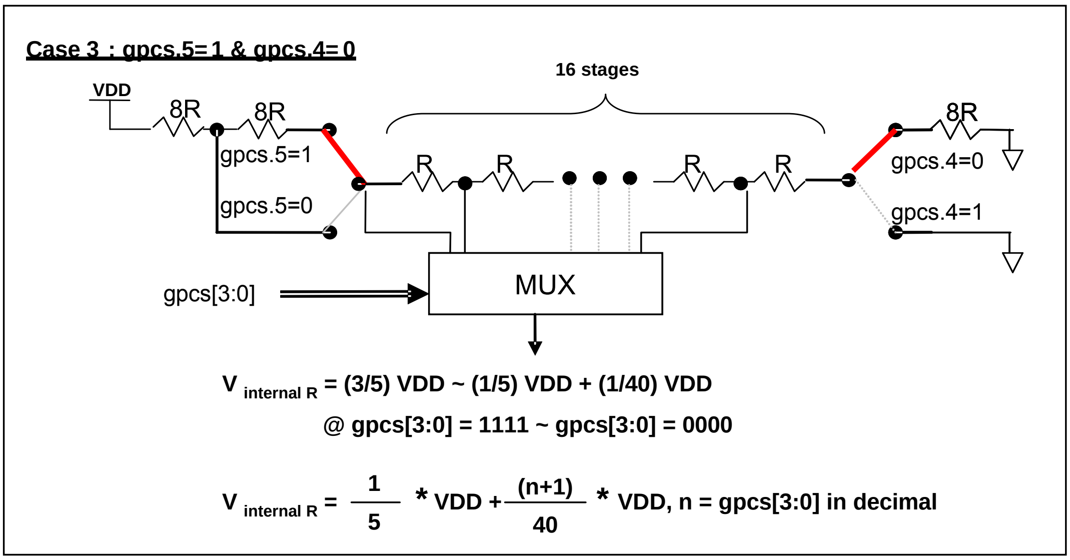


图 6：$V_{internal\ R}$ 硬件接法（gpcs.5=1 & gpcs.4=0）

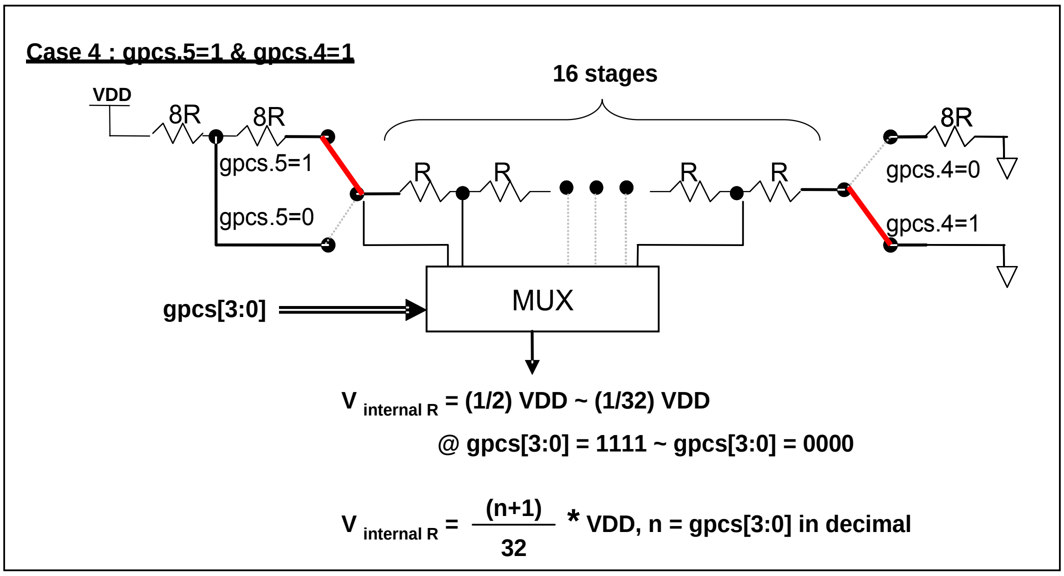


图 7：$V_{internal\ R}$ 硬件接法（gpcs.5=1 & gpcs.4=1）

5.5.2 使用比较器

例一:

选择 PA3 为负输入和 $V_{internal\ R}$ 为正输入，$V_{internal\ R}$ 的电压为(18/32)*VDD，比较器的结果将输出到 PA0。$V_{internal\ R}$ 选择图 4 的配置方式，gpcs [3:0] = 4b'1001 (n=9) 以得到 $V_{internal\ R}$ = (1/4)*VDD + [(9+1)/32]*VDD = (18/32)*VDD 的参考电压。

```
gpcs        = 0b1_0_0_0_1001;      // 输出到 PA0, V_internal R = (18/32)*VDD
gpcc        = 0b1_0_0_0_000_0;     // 启用比较器, 负输入=PA3-, 正输入=V_internal R
$ padier      0b0000_1_000;        //禁用 PA3 数字输入使能
```

例二:

选择 $V_{internal\ R}$ 为负输入，$V_{internal\ R}$ 的电压为(14/32)*VDD 和 PA4 为正输入，比较器的结果将反极性并输出到 PA0。$V_{internal\ R}$ 的电压为(14/32)*VDD。$V_{internal\ R}$ 选择图 7 的配置方式，gpcs [3:0] = 4b'1101 (n=13) 以得到 $V_{internal\ R}$ = [(13+1)/32]*VDD = (14/32)*VDD。

```
gpcs        = 0b1_1_1_1_1101;      //  V_internal R = VDD*(14/32)
gpcc        = 0b1_0_0_1_011_1;     //  输出反极性, 负输入= V_internal R, 正输入=PA4
$ padier      0b000_1_0000;        //禁用 PA4 数字输入使能
```

5.5.3 使用比较器和 band-gap 参考电压生成器

内部 Band-gap 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 Band-gap 参考电压可以选做负输入去和正输入 $V_{internal\ R}$ 比较。$V_{internal\ R}$ 的电源是 VDD，利用调整 $V_{internal\ R}$ 电压水平和 Band-gap 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 $V_{internal\ R}$ 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：VDD = [32 / (N+9)] * 1.20 volt;
对于 Case 2 而言：VDD = [24 / (N+1)] * 1.20 volt;
对于 Case 3 而言：VDD = [40 / (N+9)] * 1.20 volt;
对于 Case 4 而言：VDD = [32 / (N+1)] * 1.20 volt;

更多的讯息以及参考程序，请参考 IDE 软件。

5.6 16 位定时器 (Timer16)

PMS150C 内置一个 16 位硬件定时器，定时器时钟可来自于系统时钟（CLK）、内部高频振荡时钟（IHRC）、内部低频振荡时钟（ILRC）或 PA0/PA4，在送到时钟的 16 位计数器（counter16）之前，1 个可软件编程的预分频器提供÷1、÷4、÷16、÷64 选择，让计数范围更大。16 位计数器只能向上计数，计数器初始值可以使用 stt16 指令来设定，而计数器的数值也可以利用 ldt16 指令存储到 SRAM 数据存储器。可软件编程的选择器用于选择 Timer16 的中断条件，当计数器溢出时， Timer16 可以触发中断。中断源是来自 16 位定时器的位 8 到位 15，中断类型可以上升沿触发或下降沿触发，是经由寄存器 *integs*.4 选择。Timer16 模块框图如图 8。

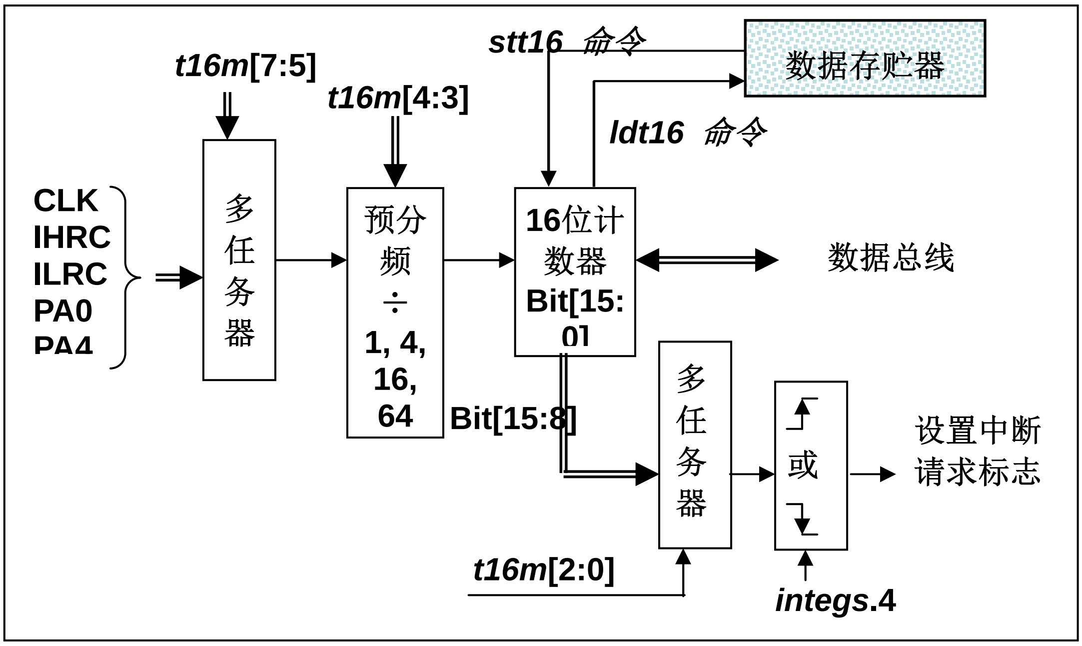


图 8：Timer16 模块框图

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

```
T16M    IO_RW    0x06
    $ 7~5:   STOP, SYSCLK, X, X, 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
```

使用者可以依照系统的要求来定义 T16M 参数，例子如下：

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

$    T16M    PA0, /1, BIT8;
     // 选择 PA0 当 Timer16 时钟源，每 2^9 个时钟周期产生一次 INTRQ.2=1
     // 每接收 512 个 PA0 个时钟周期产生一次 INTRQ.2=1

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

5.7 8 位 PWM 计数器(Timer2)

PMS150C 内置 1 个 8 位 PWM 硬件定时器(Timer2/TM2)，硬件框图请参考图 9。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 或 PA4 的输出 。寄存器 ***tm2c*** 的位[7：4]用来选择定时器时钟。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 的设定，Timer2 的输出可以是 PA3 或 PA4。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÷1，÷4，÷16 和÷64 的选择，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÷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。TM2_CLK 也可以被选定为系统时钟，以提供特殊的系统时钟频率，请参阅 ***clkmd*** 寄存器。

8 位 PWM 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或 PWM 占空比。8 位 PWM 定时器有两个工作模式：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 10 显示出 Timer2 周期模式和 PWM 模式的时序图。

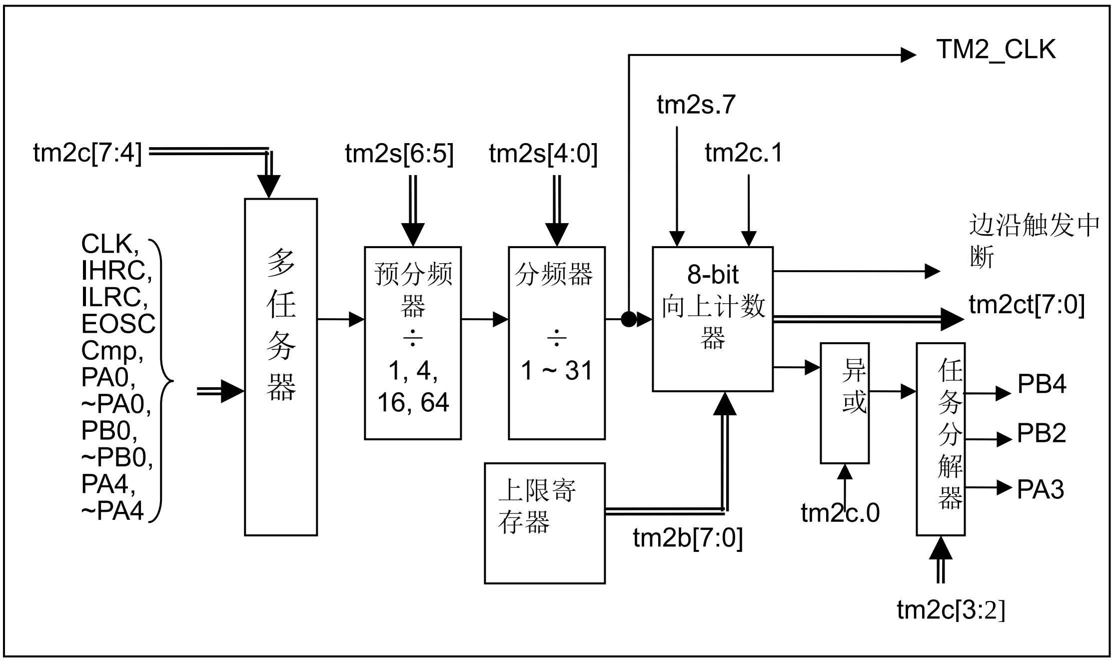


图 9. Timer2 模块框图

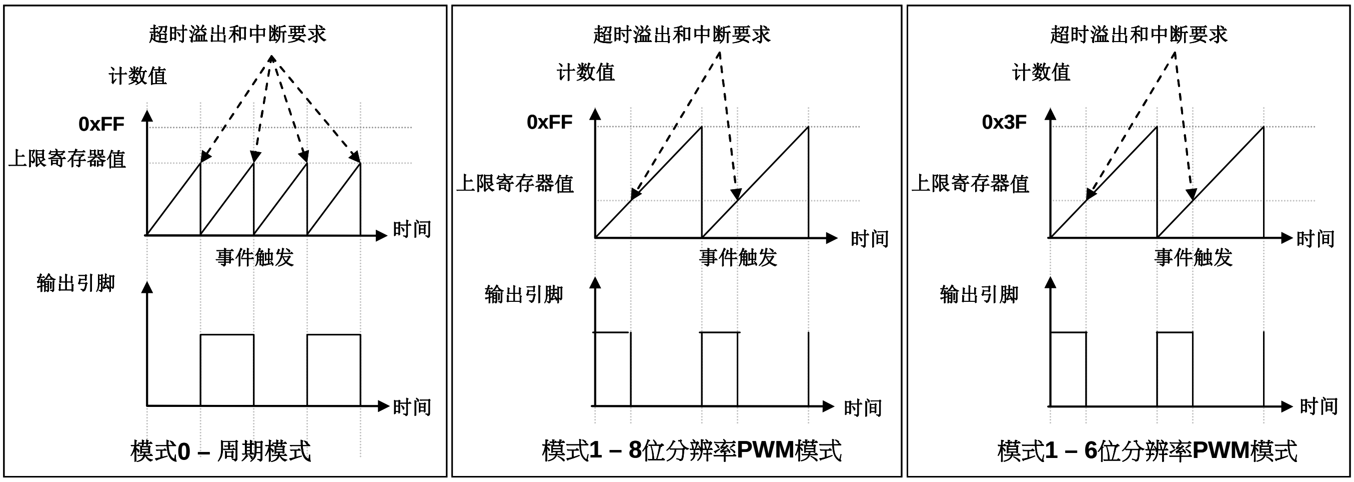


图 10. Timer2 周期模式和 PWM 模式的时序图

5.7.1 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：

输出信号频率 = Y ÷ [2 × (K+1) × S1 × (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(1，4，16，64)

S2 = tm2s[4:0]：分频器值(十进制，1 ~ 31)

例 1：

tm2c = 0b0001_1100，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_1100，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_1100，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_1100，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 定时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 = 0x0;
        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;
        }
}
```

5.7.2 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：

输出频率 = Y ÷ [256 × S1 × (S2+1)]

输出空占比 = (K＋1) ÷ 256

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(1，4，16，64)
S2 = tm2s[4:0]：分频器值(十进制，1 ~ 31)

例 1：

tm2c = 0b0001_1110，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_1110，Y=8MHz
tm2b = 0b0111_1111，K=127
tm2s = 0b0_11_11111，S1=64, S2=31
➔ 输出频率 = 8MHz ÷ (256 × 64 × (31+1)) = 15.25Hz
➔ 输出空占比 = [(127+1) ÷ 256] × 100% = 50%

例 3：

tm2c = 0b0001_1110，Y=8MHz
tm2b = 0b1111_1111，K=255
tm2s = 0b0_00_00000，S1=1, S2=0
➔ 输出频率 = 8MHz ÷ (256 × 1 × (0+1)) = 31.25KHz
➔ 输出空占比 = [(255+1) ÷ 256] × 100% = 100%

例 4：

tm2c = 0b0001_1110，Y=8MHz
tm2b = 0b0000_1001，K = 9
tm2s = 0b0_00_00000，S1=1, S2=0
➔ 输出频率 = 8MHz ÷ (256 × 1 × (0+1)) = 31.25KHz
➔ 输出空占比 = [(9+1) ÷ 256] × 100% = 3.9%

使用 Timer2 定时器产生 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 = 0x0;
  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;
  }
}
```

5.7.3 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：

输出频率 = Y ÷ [64 × S1 × (S2+1)]

输出空占比 = (K＋1) ÷ 64

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(1，4，16，64)
S2 = tm2s[4:0]：分频器值(十进制，1 ~ 31)

例 1：

tm2c = 0b0001_1110，Y=8MHz
tm2b = 0b0001_1111，K=31
tm2s = 0b1_00_00000，S1=1，S2=0
➔ 输出频率 = 8MHz ÷ (64 × 1 × (0+1)) = 125KHz
➔ 输出空占比 = [(31+1) ÷ 64] × 100% = 50%

例 2：

tm2c = 0b0001_1110，Y=8MHz
tm2b = 0b0001_1111，K=31
tm2s = 0b1_11_11111，S1=64，S2=31
➔ 输出频率 = 8MHz ÷ (64 × 64 × (31+1)) = 61.03Hz
➔ 输出空占比 = [(31+1) ÷ 64] × 100% = 50%

例 3：

tm2c = 0b0001_1110，Y=8MHz
tm2b = 0b0011_1111，K=63
tm2s = 0b1_00_00000，S1=1, S2=0
➔ 输出频率 = 8MHz ÷ (64 × 1 × (0+1)) = 125KHz
➔ 输出空占比 = [(63+1) ÷ 64] × 100% = 100%

例 4：

tm2c = 0b0001_1110，Y=8MHz
tm2b = 0b0000_0000，K=0
tm2s = 0b1_00_00000，S1=1, S2=0
➔ 输出频率 = 8MHz ÷ (64 × 1 × (0+1)) = 125KHz
➔ 输出空占比 = [(0+1) ÷ 64] × 100% =1.5%

5.8 看门狗定时器

看门狗定时器是一个计数器，其时钟源来自内部低频振荡器（ILRC），频率大约是 62KHz@5V。利用 *mis* 寄存器的选择，可以设定三种不同的看门狗定时器超时时间，它是：

- 当 misc[1:0]=11 时：262144 个 ILRC 时钟周期
- 当 misc[1:0]=10 时：65536 个 ILRC 时钟周期
- 当 misc[1:0]=01 时：16384 个 ILRC 时钟周期
- 当 misc[1:0]=00(默认)时：8192 个 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。为确保看门狗定时器在超时溢出周期之前被清零，在安全时间内，用指令"wdreset"清零看门狗定时器。在上电复位或任何时候使用 wdreset 指令，看门狗定时器都会被清零。当看门狗定时器超时溢出时，PMS150C 将复位并重新运行程序。请特别注意，由于生产制程会引起 ILRC 频率相当大的漂移，上面的数据仅供设计参考用，还是需要以各个单片机测量到的数据为准。

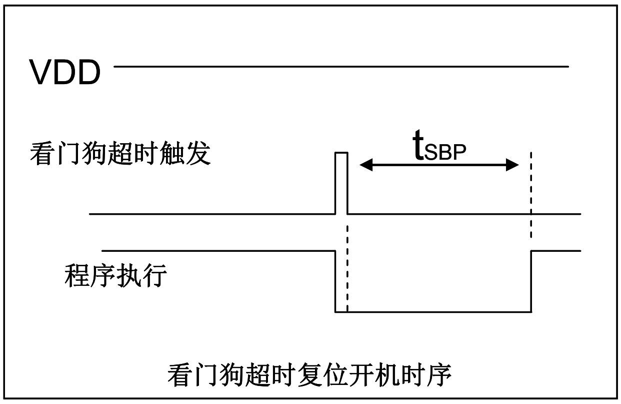


图 11：看门狗定时器超时溢出的相关时序

5.9 中断

PMS150C 有二个中断源：

1. 外部中断源 PA0
2. Timer16 中断源

每个中断请求源都有自己的中断控制位启用或禁用它。硬件框图请参考图 12，所有的中断请求标志位是由硬件置位并且并通过软件写寄存器 intrq 清零。中断请求标志设置点可以是上升沿或下降沿或两者兼而有之，这取决于对寄存器 *integs* 的设置。所有的中断请求源最后都需由 *engint* 指令控制（启用全局中断）使中断运行，以及使用 *disgint* 指令（禁用全局中断）停用它。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。

由于堆栈是共享数据存储器，使用者应仔细使用，通过软件编程调整栈点在内存的位置，每个堆栈指针的深度可以完全由用户指定，以实现最大的系统弹性。

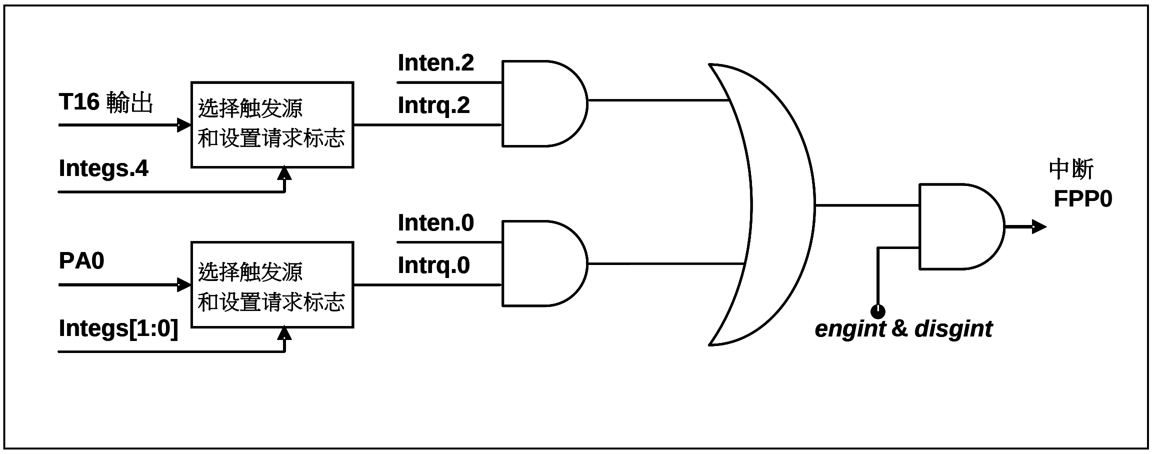


图 12：中断硬件框图

一旦发生中断，工作流程是：

- 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。

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。

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

```
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                         // 禁用全局中断
        ...
}

void        Interrupt  (void)           // 中断程序
{
         PUSHAF                         // 存储 ALU 和 FLAG 寄存器
         If      (INTRQ.0)
         {                              // PA0 的中断程序
                 // INTRQ.0 =   0;      // 这条指令不能使用
                   INTRQ   =   0;       // 请使用这条指令
                 ...
         }
         ...
         POPAF                          // 回复 ALU 和 FLAG 寄存器
 }
```

5.10 省电与掉电

PMS150C 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 3 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
STOPSYS	停止	停止
STOPEXE	没改变	没改变

表 3：省电模式和掉电模式在振荡器模块的差异

5.10.1 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，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 IHR/ILRC)。假如系统唤醒是因输入引脚切换，那可以视为单片机继续正常的运行，在 *stopexe* 指令之后最好加个 ***nop*** 指令，省电模式的详细信息如下所示：

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

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

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

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

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

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

5.10.2 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 PMS150C 芯片直接进入掉电模式。下面显示发出 *stopsys* 命令后，PMS150C 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：任何 IO 切换。
- 如果 PA 是输入模式，并由 *padier* 寄存器设置为模拟输入，
  那该引脚是不能被用来唤醒系统。

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

```
.CMKMD   =    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
```

5.10.3 唤醒

进入掉电或省电模式后，PMS150C 可以通过切换 IO 引脚恢复正常工作；而 Timer16 中断的唤醒只适用于省电模式。表 4 显示 *stopsysy* 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		
	切换 IO 引脚	T16 中断
stopsys	是	否
stopexe	是	是

表 4：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 PMS150C，寄存器 *padier* 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 2048 ILRC 时钟周期；另外，PMS150C 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。对快速开机而言，假如是在 *stopexe* 省电模式下，切换 IO 引脚的快速唤醒时间为 32 ILRC 时钟周期。

模式	唤醒模式	切换 IO 引脚的唤醒时间(t_{WUP})
STOPEXE 省电模式	快速唤醒	32 * T_{ILRC}； 这里 T_{ILRC} 是 IHRC 时钟周期
STOPSYS 掉电模式	快速唤醒	32 * T_{ILRC}； 这里 T_{ILRC} 是 IHRC 时钟周期
STOPEXE 省电模式	普通唤醒	2048 * T_{ILRC}； 这里 T_{ILRC} 是 ILRC 时钟周期
STOPSYS 掉电模式	普通唤醒	2048 * T_{ILRC}； 这里 T_{ILRC} 是 ILRC 时钟周期

5.11 IO 引脚

除了 PA5，PMS150C 所有 IO 引脚都可以透过数据寄存器（*pa*），控制寄存器（*pac*）和弱上拉电阻（*paph*）设定成输入或输出，每一 IO 引脚都可以独立配置成不同的功能；所有这些引脚设置有施密特触发输入缓冲器和 CMOS 输出驱动电位水平。当这些引脚为输出低电位时，弱上拉电阻会自动关闭。如果要读取端口上的电位状态，一定要先设置成输入模式；在输出模式下，读取到的数据是数据寄存器的值。表 5 显示了 IO 缓冲区硬件图，表 2 为端口 PA0 位的设定配置表。

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

表 5: PA0 设定配置表

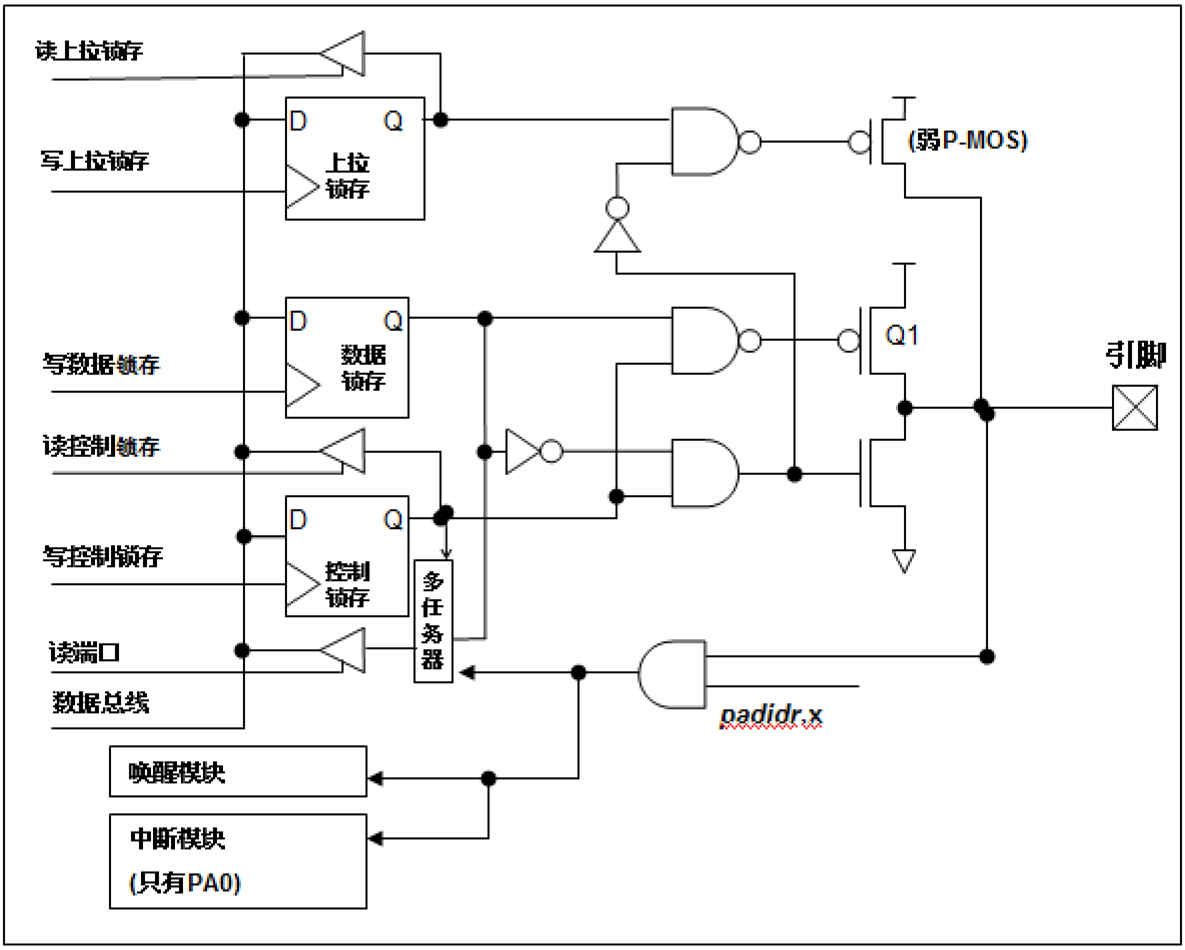


图 13：引脚缓冲区硬件图

除了 PA5，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）。当 PMS150C 在掉电或省电模式，每一个引脚都可以切换其状态来唤醒系统。对于需用来唤醒系统的引脚，必须设置为输入模式以及寄存器 *padier* 相应为高。同样的原因，当 PA0 用来作为外部中断引脚时，*padier.0* 应设置高。

5.12 复位和 LVR

5.12.1 复位

引起 PMS150C 复位的原因有很多，一旦复位发生，PMS150C 的所有寄存器将被设置为默认值; 发生复位后，系统会重新启动，程序计数器会跳跃地址 0x00。当发生上电复位或 LVR 复位，数据存储器的值是在不确定的状态；然而，若是复位是因为 PRST＃引脚或 WDT 超时溢位，数据存储器的值将被保留。

6. IO 寄存器

6.1 标志寄存器（*flag*），IO 地址 = 0x00

位	初始值	读/写	描 述
7 – 4	-	-	保留。这 4 个位读值为“1”。
3	-	读/写	OV（溢出标志）。当数学运算溢出时，这一位会设置为 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），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。请注意 0 位必须维持为 0 因程序计数器是 16 位。

6.3 时钟控制寄存器（clkmd），IO 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="3">7 – 5</td><td rowspan="3">111</td><td rowspan="3">读/写</td><td colspan="2">系统时钟选择</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
001: IHRC÷2
01x: 保留
10x: 保留
110: ILRC÷4
111: ILRC (默认)</td><td>000: IHRC÷16
001: IHRC÷8
010: ILRC÷16
011: IHRC÷32
100: IHRC÷64
1xx: 保留</td></tr>
<tr><td>4</td><td>1</td><td>读/写</td><td colspan="2">内部高频 RC 振荡器功能。 0/1：禁用/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：禁用/启用
当内部低频 RC 振荡器功能禁用时，看门狗定时器功能同时被关闭。</td></tr>
<tr><td>1</td><td>1</td><td>读/写</td><td colspan="2">看门狗定时器功能。 0/1：禁用/启用</td></tr>
<tr><td>0</td><td>0</td><td>读/写</td><td colspan="2">引脚 PA5/PRST# 功能. 0 / 1: PA5 / PRST#.</td></tr>
</table>

6.4 中断允许寄存器（inten），IO 地址 = 0x04

位	初始值	读/写	描 述
7 – 3	-	读/写	保留。
2	-	读/写	启用从 Timer16 的溢出中断。0/1：禁用/启用
1	-	-	保留
0	-	读/写	启用从 pa0 的中断。 0/1：禁用/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 – 3	-	读/写	保留。
2	-	读/写	Timer16 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
1	-	读/写	保留
0	-	读/写	PA0 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求

6.6 Timer16 控制寄存器（t16m），IO 地址 = 0x06

位	初始值	读/写	描 述
7 – 5	000	读/写	Timer16 时钟选择 000：Timer16 禁用 001：CLK 系统时钟 010：保留 011：PA4（外部事件） 100：IHRC 101：保留 110：ILRC 111：PA0（外部事件）
4 – 3	00	读/写	Timer16 内部的时钟分频器 00：÷1 ，01：÷4 ，10：÷16 ，11：÷64
2 – 0	000	读/写	中断源选择。当选择位由低变高或高变低时，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

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

位	初始值	读/写	描 述
7 – 1	-	-	保留. 请设为 0。
0	0	只写	LVR 硬件模块断电。 0 / 1：正常/ 断电

6.8 中断缘选择寄存器 (integs), IO 地址 = 0x0c

位	初始值	读/写	描 述
7 – 5	-	-	保留。请设为 0。
4	0	只写	Timer16 中断缘选择。 0 : 上升缘请求中断。 1 : 下降缘请求中断。
3 – 2	00	只写	保留。
1 – 0	00	只写	PA0 中断缘选择。 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。

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

位	初始值	读/写	描 述
7– 3	0x1F	只写	启用 PA7~PA3 系统唤醒。 1 / 0 : 启用 / 禁用 当这个位设为 0 时，PA7~PA3 无法用来唤醒系统。
2 – 1	-	-	保留。
0	1	只写	启用 PA0 系统唤醒和中断请求。 1 / 0 : 启用 / 禁用 当这个位设为 0 时，PA0 无法用来唤醒系统以及中断请求。

6.10 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	资料寄存器的端口 A。

6.11 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

6.12 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。这些寄存器是用来控制上拉高端口 A 每个相应的引脚。 0/1：禁用/启用

6.13 杂项寄存器(misc), IO 地址 = 0x3b

位	初始值	读/写	描 述
7 – 6	-	-	保留。
5	0	WO	快唤醒功能。 0: 正常唤醒。 如果是普通开机模式，唤醒时间为 2048 ILRC 时钟周期 如果是快速开机模式，唤醒时间为 32 ILRC 时钟 1: 快唤醒。 唤醒时间为 32 ILRC 时钟
4	0	-	保留。
3	0	-	保留。
2	0	WO	禁用 LVR 功能： 0 / 1 : 启用 / 禁用
1 – 0	00	WO	看门狗时钟超时时间设定： 00: 8192 个 ILRC 时钟周期 01: 16384 个 ILRC 时钟周期 10: 65536 个 ILRC 时钟周期 11: 262144 个 ILRC 时钟周期

6.14 比较器控制寄存器(gpcc)，IO 地址 = 0x1A

位	初始值	读/写	描 述
7	0	读/写	启用比较器。0/1：禁用/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禁用，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 band-gap 参考电压 011：$V_{internal\ R}$ 100：PB6 (不适用 EV5 仿真) 101：PB7 (不适用 EV5 仿真) 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)，IO 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：禁用/启用
6	-	-	保留 。
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 $V_{internal\ R}$。 00 最低) ~ 1111 (最高)

6.16 Timer2 控制寄存器(tm2c)，IO 地址 = 0x1C

位	初始值	读/写	描 述
7 – 4	0000	读/写	Timer2 时钟源选择： 0000：禁用 0001：CLK 0010：IHRC 0011：EOSC 0100：ILRC 0101：比较器输出 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) 其它：保留 注意：在 ICE 模式且 IHRC 被选为 Timer2 定时器时钟，当 ICE 停下时，发送到定时器的时钟是不停止，定时器仍然继续计数。
3 – 2	00	读/写	Timer2 输出选择： 00：禁用 01：PB2 10：PA3 11：PB4
1	0	读/写	Timer2 模式选择： 0/1：定周期模式/PWM 模式
0	0	读/写	启用 Timer2 反极性输出。 0/1：禁用/启用

6.17 Timer2 计数寄存器(tm2ct)，IO 地址 = 0x1D

位	初始值	读/写	描 述
7 – 0	0x00	读/写	Timer2 定时器位[7:0]。

6.18 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。

6.19 Timer2 分频寄存器(tm2s)，IO 地址 = 0x17

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。

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	标志寄存器
I	实时数据
&	逻辑 AND
|	逻辑 OR
←	移动
^	**异或 OR**
+	加
－	减
-～	**NOT**（逻辑补码，1 补码）
〒	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	FPP0 的程序计数器
word	只允许寻址在 address 0~0x1F (0~31) 的位置
M.n，IO.n	只允许寻址在 address 0~0xF (0~15) 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; 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
mov **M, a**	移动数据由累加器到内存 例如： *mov* MEM, a; 结果： MEM ← a 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
mov **a, M**	移动数据由内存到累加器 例如： *mov* a, MEM ; 结果： a ← MEM; 当 MEM 为零时，标志位 Z 会被置位。 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
mov **a, IO**	移动数据由 IO 到累加器 例如： *mov* a, pa ; 结果： a ← pa; 当 pa 为零时，标志位 Z 会被置位。 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
mov **IO, a**	移动数据由累加器到 IO 例如： *mov* pa, a; 结果： pa ← a ; 受影响的标志位： 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
	受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
	应用范例：
	-- 	
	word T16val ; // 定义一个 RAM word
	…
	clear lb@T16val ; // 清零 T16val（LSB）
	clear hb@T16val ; // 清零 T16val（MSB）
	stt16 T16val ; // 设定 Timer16 的起始值为 0
	…
	set1 t16m.5 ; // 启用 Timer16
	…
	set0 t16m.5 ; // 禁用 Timer16
	ldt16 T16val ; // 将 Timer16 的 16 位计算值复制到 RAM T16val
	….
	-- 	

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

***idxm* a, index**	使用索引作为 RAM 的地址并将 RAM 的数据读取并加载到累加器。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 PMS150C 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 PMS150C 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 移位元元运算类指令

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 Example: *slc* MEM ; 结果： MEM (b6,b5,b4,b3,b2,b1,b0,C) ← MEM (b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM (b7) 受影响的标志位： Z：『不变』， C：『受影响』， AC：『不变』， OV：『不变』
swap **a**	累加器的高 4 位与低 4 位互换 例如：*swap* a ; 结果： a (b3,b2,b1,b0,b7,b6,b5,b4) ← a (b7,b6,b5,b4,b3,b2,b1,b0) 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

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

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

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

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

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

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』
t0sn **IO.n**	如果 IO 的指定位是 0，跳过下一个指令。 例如：*t0sn* pa.5; 结果： 如果 PA5 是 0，跳过下一个指令。 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
t1sn **IO.n**	如果 IO 的指定位是 1，跳过下一个指令。 Example: *t1sn* pa.5 ; 结果：如果 PA5 是 1，跳过下一个指令。 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

t0sn **M.n**	如果 RAM 的指定位是 0，跳过下一个指令。 例如： *t0sn* MEM.5 ; 结果： 如果 MEM 的位 5 是 0，跳过下一个指令。 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
t1sn **M.n**	如果 RAM 的指定位是 1，跳过下一个指令。 例如： *t1sn* MEM.5 ; 结果： 如果 MEM 的位 5 是 1，跳过下一个指令。 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
izsn **a**	累加器加 1，若累加器新值是 0，跳过下一个指令。 例如： *izsn* a; 结果： a ← a + 1，若 a=0，跳过下一个指令。 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
dzsn **a**	累加器减 1，若累加器新值是 0，跳过下一个指令。 例如： *dzsn* a; 结果： a ← a － 1，若 a=0，跳过下一个指令。 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
izsn **M**	RAM 加 1，若 RAM 新值是 0，跳过下一个指令。 例如： *izsn* MEM; 结果： MEM ← MEM + 1，若 MEM=0，跳过下一个指令。 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
dzsn **M**	RAM 减 1，若 RAM 新值是 0，跳过下一个指令。 例如： *dzsn* MEM; 结果： MEM ← MEM - 1，若 MEM=0，跳过下一个指令。 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』

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：但是系统时钟是被禁用以节省功耗。 例如： *stopexe*; 结果： 停住系统时钟，但是仍保持震荡器模块工作 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
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	复位看门狗定时器 例如： *wdreset* ; 结果： 复位看门狗定时器 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

7.8 指令执行周期综述

2 个周期	*goto, call, pcadd, ret, reti, idxm*
1 个周期/2 个周期	ceqsn, cneqsn, *t0sn, t1sn, dzsn, izsn*
1 个周期	其它指令

7.9 指令影响标志的综述

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	-	-	-	-										

8. 特别注意事项

此章节是提醒使用者在使用 PMS150C 时避免一些常犯的错误。

8.1. 警告

在使用 IC 前，请务必认真阅读 PMS150C 相关的 APN（应用注意事项）。APN 下载地址为：
http://www.padauk.com.tw/technical-application.php

8.2. 使用 IC 时

8.2.1. IO 使用与设定

(1) IO 作为数字输入和打开唤醒功能
- 将 IO 设为输入。
- 用 PADIER 寄存器，将对应的位设为 1。
- 为了防止 PA 中那些没有用到的 IO 口漏电，PADIER[1：2]需要常设为 0。

(2) PA5 作为输出
- PA5 只能做 Open Drain 输出，输出高需要外加上拉电阻。

(3) PA5 作为 PRST#输入
- 设定 PA5 为输入。
- 设定 CLKMD.0=1，使 PA5 为外部 PRST#输入脚位。

(4) PA5 作为输入并通过长导线连接至按键或者开关
- 必需在 PA5 与长导线中间串接 >10 欧电阻。
- 应尽量避免使用 PA5 作为输入。

8.2.2. 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。

8.2.3. 切换系统时钟

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

```
.CLKMD   =    0x36;        // 切到 IHRC，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。
```

8.2.4. 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。

8.2.5. TIMER16 溢出时间

如果设定 T16M 计数器 BIT8 为 1 时产生中断，则第一次中断是在计数到 0x100 时发生（BIT8 从 0 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。

8.2.6. LVR

(1) Power On 时, VDD 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 才会有效。
(3) 可以设定寄存器 EOSCR.0 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 可能工作不正常。

8.2.7. 指令

(1) PMS150C 支持 79 个指令。
(2) PMS150C 指令周期如下表所示 ：

指令	条件	指令周期
goto, call，pcadd,ret,reti,idxm		**2T**
ceqsn, cneqsn, t0sn, t1sn, dzsn, izsn	判断条件成立	**2T**
	判断条件不成立	1T
Others		1T

8.2.8. RAM 定义限制

(1) 位寻址只能定义在 RAM 区的 0X00 到 0X0F 空间。
(2) Word 变量只能定义在 RAM 区的 0X00 到 0X1E 空间。

8.2.9. 烧录方法

若客户自行 SOP8 / DIP8 封装, IC 摆放位置在烧录器上 Socket 后退 3 PIN, 烧录器背后 Jumper 插在 CN38 (P201CS/CD14A) 的位置。

8.3. 使用 ICE 时

仿真 PMS150C 系列功能时注意事项：

1. 请使用 5S-I-S01 仿真器，其它仿真器不支持 PMS150C。
2. 用户请在仿真器上找到 PMS150C 对应的引脚，并自行用排线或杜邦线正确地连接到目标板上。