PMS164

12 触摸键单片机

数据手册

Version 0.00 – March 25, 2020

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

目 录

修订历史:

修订	日期	描述
0.00	2020/03/25	初版

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

- 1.75KW OTP 程序内存
- 128 字节数据存储器
- 一个硬件 16 位计数器
- 两个硬件 8 位计数器和 6/7/8 位 PWM 生成器
- 一个硬件比较器
- 14 个 IO 引脚并带有上拉电阻选项
- 12 个 IO 引脚可被选作为独立的触摸引脚
- Bandgap 电路提供 1.20V Bandgap 电压
- 时钟源：内部高频 RC 振荡器，内部低频 RC 振荡器
- 8 段 LVR 复位设定：4.0V，3.5V，3.0V，2.75V，2.5V，2.2V，2.0V，1.8V
- 三个可选的外部中断引脚

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

- PMS164-2N06: DFN (2*2mm)
- PMS164-U06: SOT23-6 (60mil)
- PMS164-S08A: SOP8 (150mil)
- PMS164-S08B: SOP8 (150mil)
- PMS164-2N08: DFN (2*2mm)
- PMS164-EY10B: ESSOP10 (150mil)
- PMS164-S16: SOP16 (150mil)

2. 系统概述和方框图

PMS164 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 的架构并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。

PMS164 包含一个最多 12 键的电容式触摸控制电路。另外，PMS164 还包含 1.75KW OTP 程序内存以及 128 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（伴有 PWM 生成器功能）。

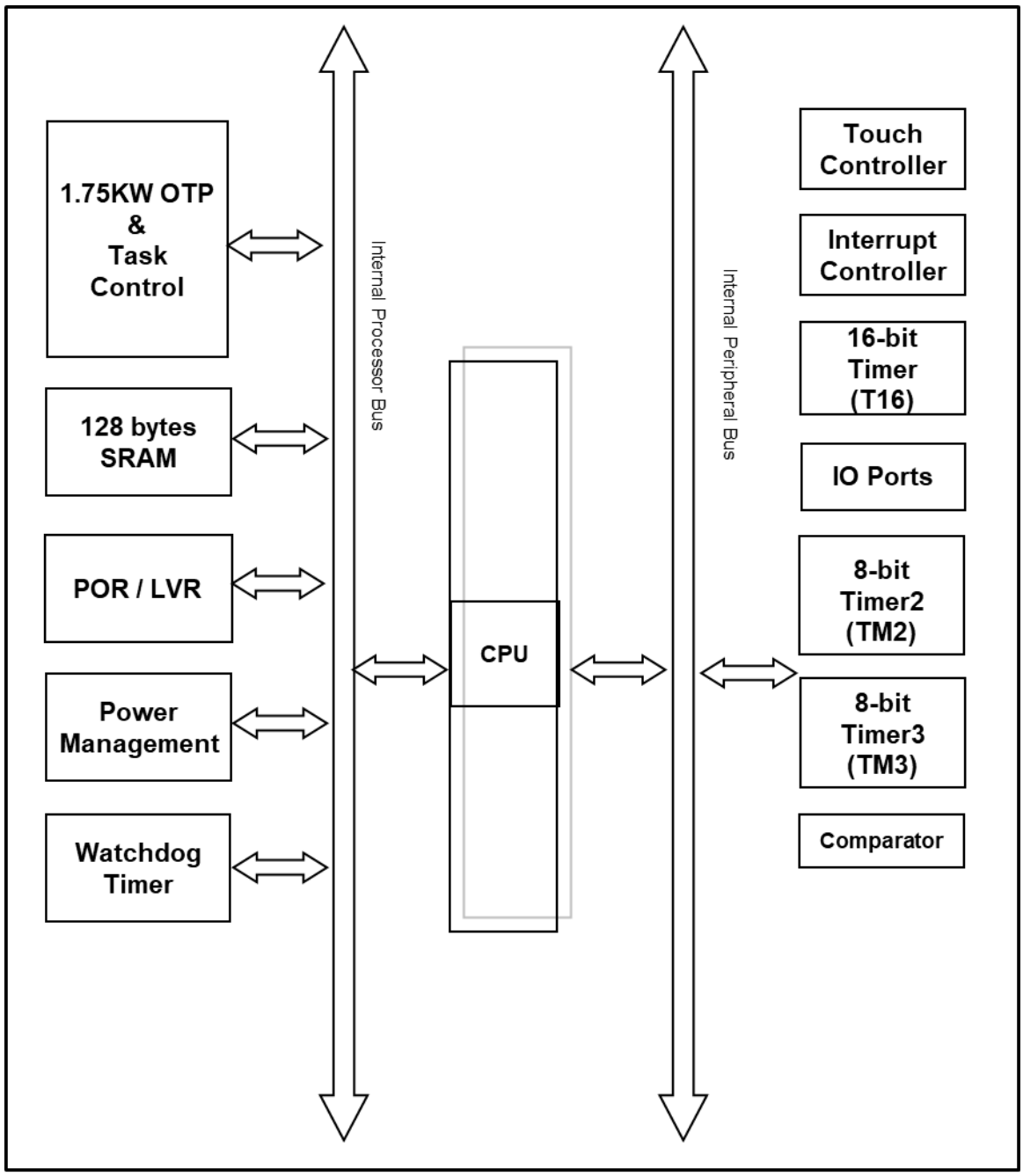
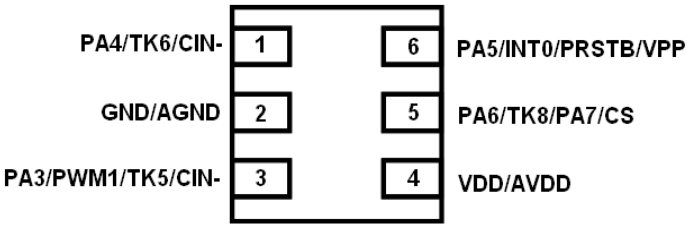
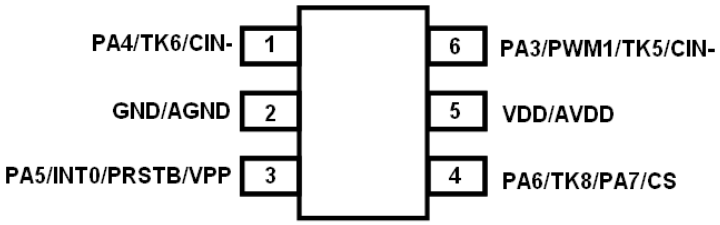


图 1：PMS164 系统方框图

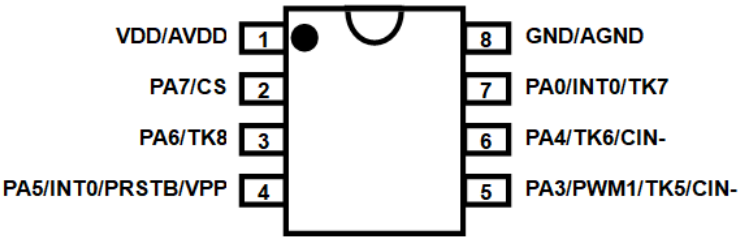
3. 引脚功能说明



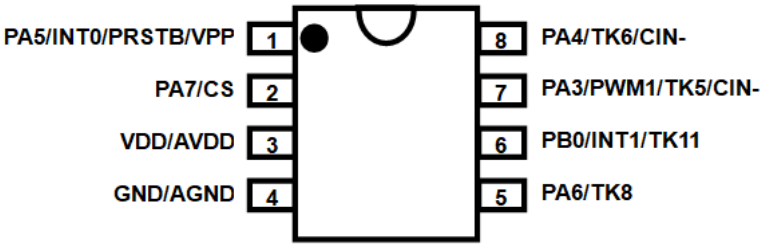
PMS164-2N06 (DFN-2*2mm)



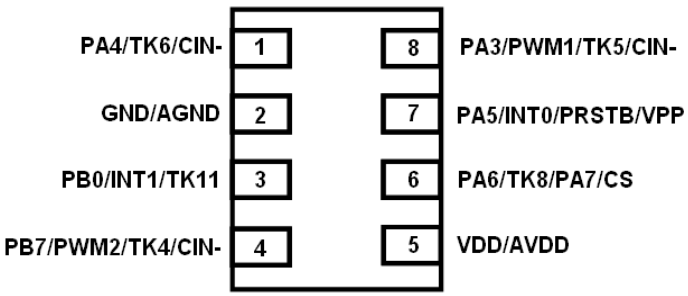
PMS164-U06 (SOT23-6 60mil)



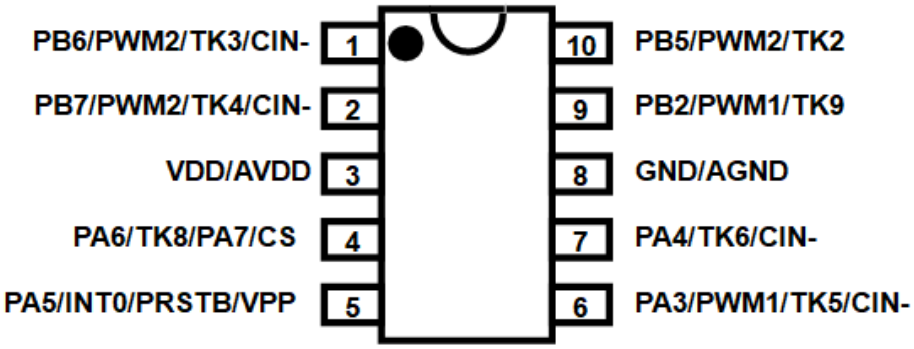
PMS164-S08A (SOP8-150mil)



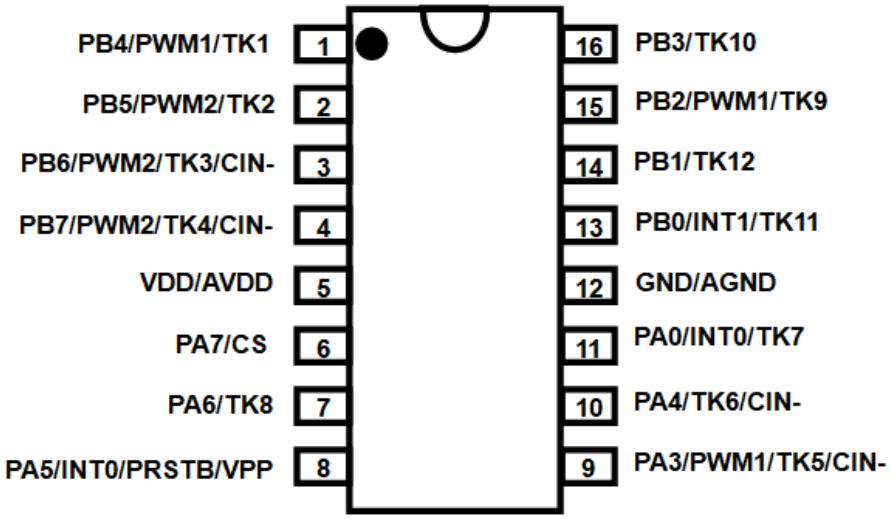
PMS164-S08B (SOP8-150mil)



PMS164-2N08 (DFN-2*2mm)



PMS164-EY10B (ESSOP10-150mil)



PMS164-S16 (SOP16-150mil)

引脚名称	引脚 & 缓存器类型	描述
PA6 / TK8	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，弱上拉电阻模式。 (2) 触摸按键 8。 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 / INT0 / PRSTB / VPP	IO ST / CMOS	此引脚可以用作： (1) 端口 A 位 5。此引脚可以设定为输入或开漏输出(open drain)，弱上拉电阻模式。 (2) 外部中断源 0。上升沿和下降沿都可触发中断。 (3) 外部复位引脚。 (4) 刻录时作为 VPP 引脚。
PA4 / TK6 / 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，弱上拉电阻模式。 (2) 触摸按键 6。 (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 4 关闭其数字输入功能。
PA3 / PWM1 / TK5 / 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，弱上拉电阻模式。 (2) Timer2 的 PWM 输出口。 (3) 触摸按键 5。 (4) 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 3 关闭其数字输入功能。
PA0 / INT0 / TK7	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，弱上拉电阻模式。 (2) 外部中断源 0。上升沿和下降沿都可触发中断。 (3) 触摸按键 7。 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。

引脚名称	引脚 & 缓存器类型	描述
PB0 / INT1 / TK11	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 B 位 0。可程序设计设定为输入或输出，弱上拉电阻模式。 (2) 外部中断源 1。上升沿和下降沿都可触发中断。 (3) 触摸按键 11 该引脚可以配置为模拟输入，为减少漏电流，请用 pbdier 寄存器位 0 关闭其数字输入功能。
PB1 / TK12	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 B 位 1。可程序设计设定为输入或输出，弱上拉电阻模式。 (2) 触摸按键 12 该引脚可以配置为模拟输入，为减少漏电流，请用 pbdier 寄存器位 1 关闭其数字输入功能。
PB2 / PWM1 / TK9	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 B 位 2。可程序设计设定为输入或输出，弱上拉电阻模式。 (2) Timer2 的 PWM 输出。 (3) 触摸按键 9。 该引脚可以配置为模拟输入，为减少漏电流，请用 pbdier 寄存器位 2 关闭其数字输入功能。
PB3 / TK10	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 B 位 3。可程序设计设定为输入或输出，弱上拉电阻模式。 (2) 触摸按键 10。 该引脚可以配置为模拟输入，为减少漏电流，请用 pbdier 寄存器位 3 关闭其数字输入功能。
PB4 / PWM1 / TK1	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 B 位 4。可程序设计设定为输入或输出，弱上拉电阻模式。 (2) Timer2 的 PWM 输出。 (3) 触摸按键 1。 该引脚可以配置为模拟输入，为减少漏电流，请用 pbdier 寄存器位 4 关闭其数字输入功能。
PB5 / PWM2 / TK2	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 B 位 5。可程序设计设定为输入或输出，弱上拉电阻模式。 (2) Timer3 的 PWM 输出。 (3) 触摸按键 2。 该引脚可以配置为模拟输入，为减少漏电流，请用 pbdier 寄存器位 5 关闭其数字输入功能。

引脚名称	引脚 & 缓存器类型	描述
PB6 / PWM2 / TK3 / CIN-	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 B 位 6。可程序设计设定为输入或输出，弱上拉电阻模式。 (2) Timer3 的 PWM 输出。 (3) 触摸按键 3。 (4) 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 pbdier 寄存器位 6 关闭其数字输入功能。
PB7 / PWM2 / TK4 / CIN-	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 B 位 7。可程序设计设定为输入或输出，弱上拉电阻模式。 (2) Timer3 的 PWM 输出。 (3) 触摸按键 4。 (4) 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 pbdier 寄存器位 7 关闭其数字输入功能。
PA7 / CS	IO ST / CMOS / Analog	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，弱上拉电阻模式。 (2) 外部电容脚。 该引脚可以配置为 CS，为减少漏电流，请用 pbdier 寄存器位 7 关闭其数字输入功能。
VDD / AVDD	VDD / AVDD	VDD：数字正电源 AVDD：模拟正电源 VDD 是 IC 电源，而 AVDD 是模拟正电源专用电源。在 IC 内部，AVDD 与 VDD 连在一起(double bonding)，而外部为相同引脚。
GND / AGND	GND / AGND	GND：数字负电源 AGND：模拟负电源 GND 是 IC 接地引脚，而 AGND 是模拟负电源接地引脚。在 IC 内部，AGND 与 GND 连在一起(double bonding)，而外部为相同引脚。
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 的精度
LVR%	低电压复位精度	-5		5	%	
f_{SYS}	系统时钟 (CLK)* = IHRC/2 IHRC/4 IHRC/8 ILRC	 0 0 0	 61K	 8M 4M 2M	Hz	 V_{DD} ≧ 3.5V V_{DD} ≧ 2.5V V_{DD} ≧ 2.0V V_{DD} =3V
I_{OP}	工作电流		0.5 50		mA uA	f_{SYS}=IHRC/16=1MIPS@5.0V f_{SYS}=ILRC=55KHz@5.0V
I_{PD}	掉电电流（通过 *stopsys* 命令设置）		1.4 1.0		uA	V_{DD} =5V V_{DD} =3.3V
I_{PS}	省电电流（通过 *stopexe* 命令设置） *停用 IHRC		5		uA	V_{DD} =3.3V
V_{IL}	输入低电压	0		0.1 V_{DD}	V	
V_{IH}	输入高电压	0.8 V_{DD} 0.7 V_{DD}		V_{DD} V_{DD}	V	PA5 其它 IO 口
I_{OL}	IO 输出灌电流（正常输出） PB4/PB5 PA7 PA5 Others		 42 26 19 14		mA	V_{DD}=5.0V, V_{OL}=0.5V
	IO 输出灌电流（低输出） PB4/PB5 PA7 PA5 Others		 8 9 19 5			
I_{OH}	IO 输出驱动电流（正常输出） PB5 PA7 PA5 Others		 30 19 0 10		mA	V_{DD}=5.0V, V_{OH}=4.5V
	IO 输出驱动电流(低输出） PA7 PA5 Others		 5 0 3			
V_{IN}	输入电压	-0.3		V_{DD}+0.3	V	
$I_{INJ\ (PIN)}$	引脚输入电流		1		uA	V_{DD} +0.3≧V_{IN}≧ -0.3
R_{PH}	上拉电阻		110 200		KΩ	V_{DD}=5.0V V_{DD}=3.3V

符号	描述	最小值.	典型值	最大值.	单位	条件
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*
		13.60*	16*	18.40*		V_{DD} =2.0V~5.5V, -20°C <Ta<70°C
f_{ILRC}	ILRC 频率 *		55		KHz	
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
t_{SBP}	系统开机时间		50		ms	@ V_{DD} =5V
t_{RST}	外部复位脉冲宽度	120			us	@ V_{DD} =5V

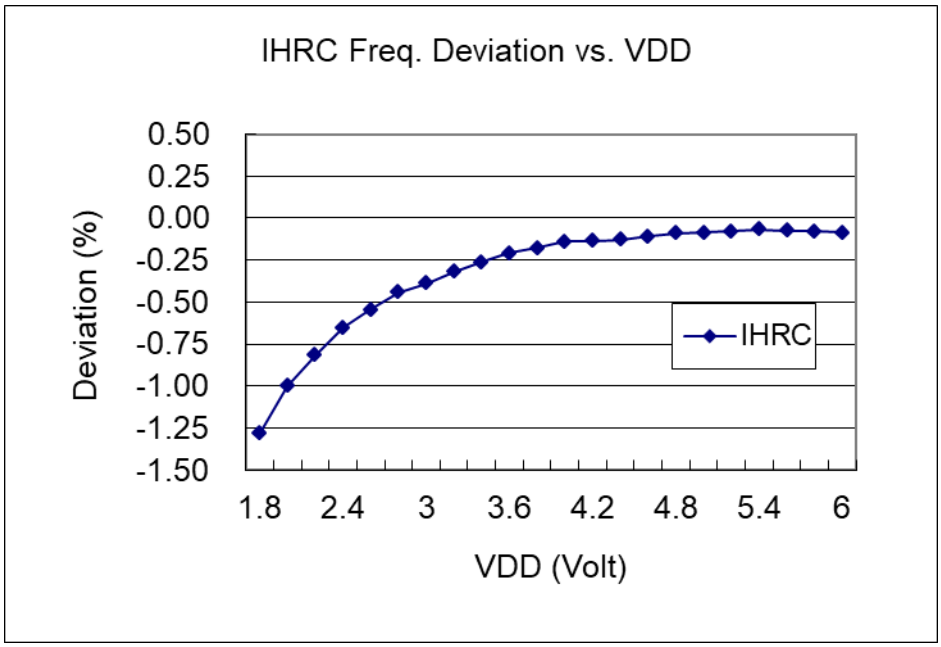
***** 这些参数是设计参考值，并不是每个芯片测试。

* 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。

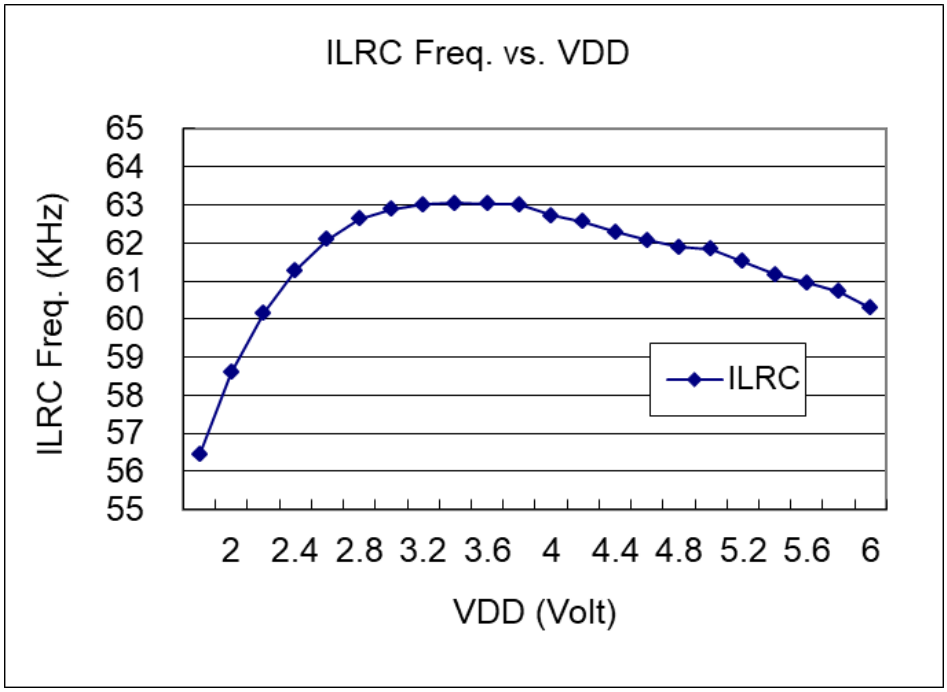
4.2. 绝对最大值范围

- 电源电压……..... 2.0V ~ 5.5V
  * 最大电压不能超过 5.5V，否则可能永久性的损坏 IC。
- 输入电压……..... -0.3V ~ V_{DD} + 0.3V
- 工作温度..... -20°C ~ 70°C
- 节点温度……..... 150°C
- 存储温度……..... -50°C ~ 125°C

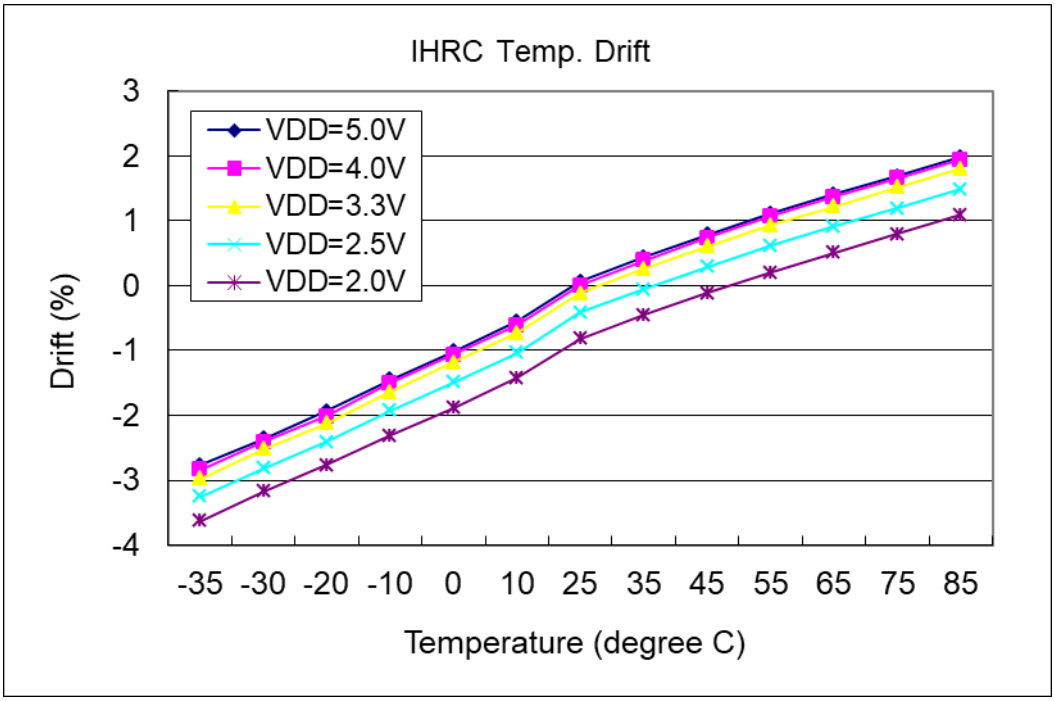
4.3. IHRC 频率与 VDD 关系曲线图（校准到 16MHz）



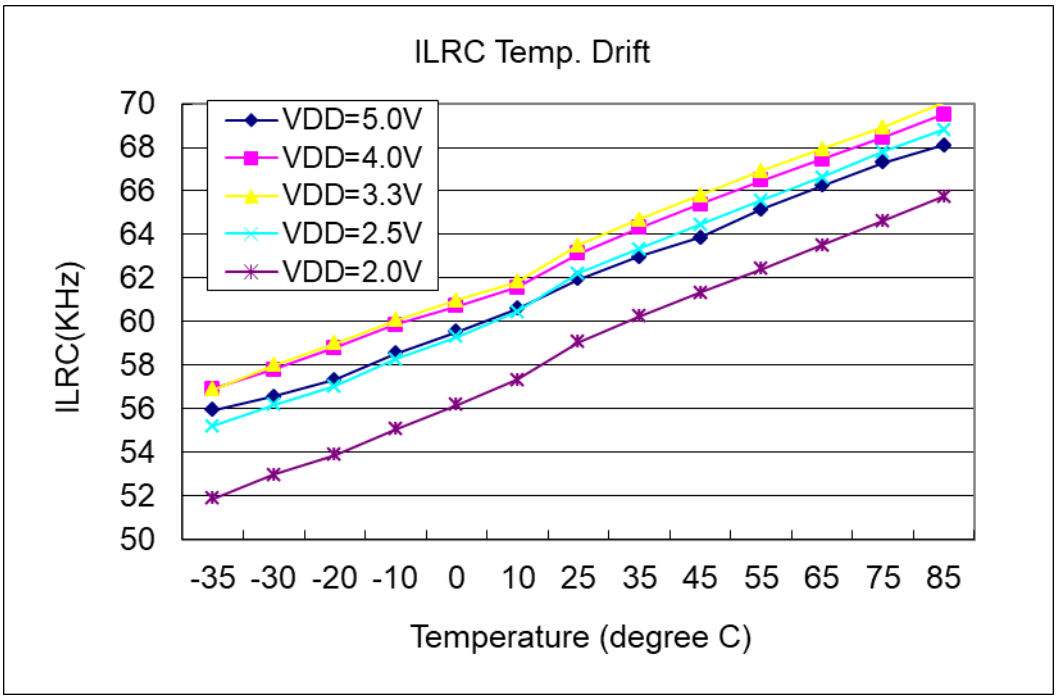
4.4. ILRC 频率与 VDD 关系曲线图



4.5. 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



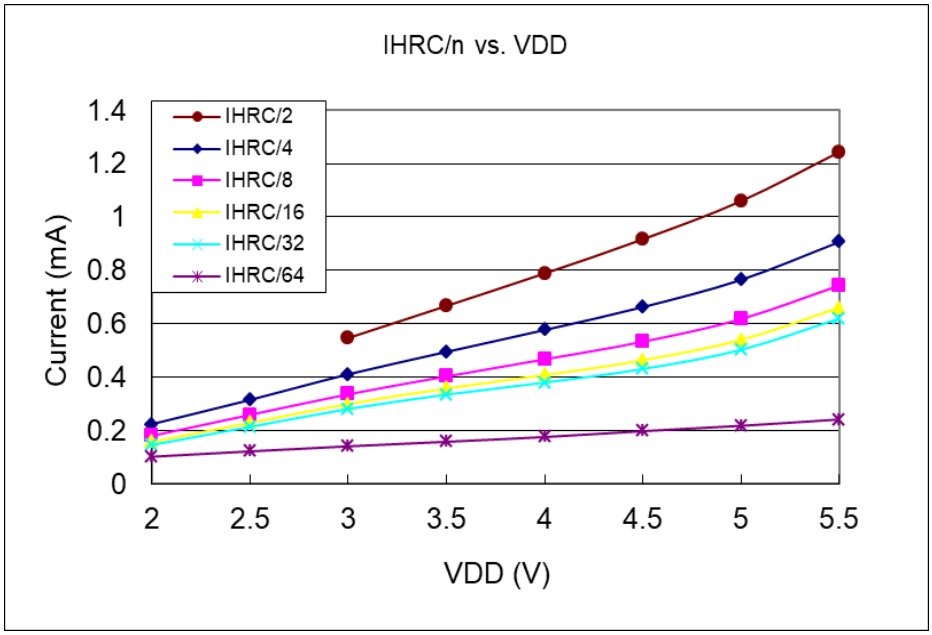
4.7. 工作电流 vs. VDD 与系统时钟 = IHRC/n 关系曲线图

➢ 测量条件：

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。



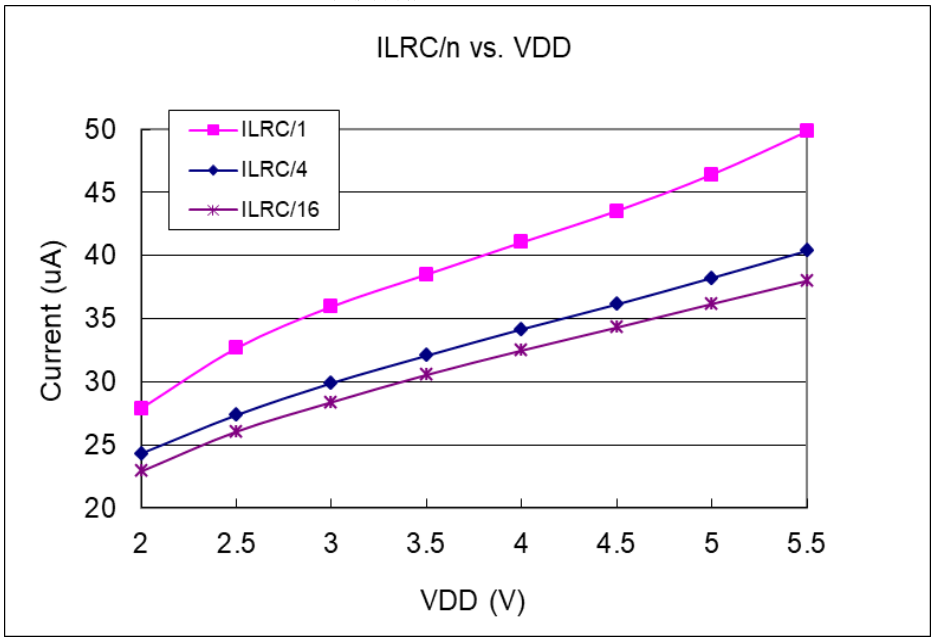
4.8. 工作电流 vs. VDD 与系统时钟 = ILRC/n 关系曲线图

➢ 测量条件：

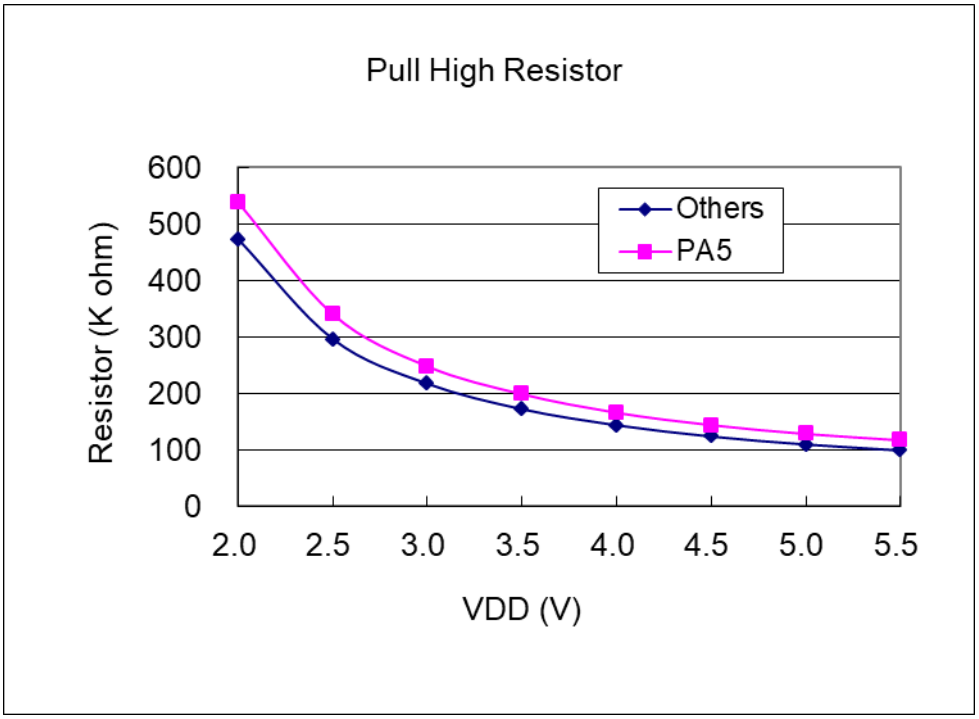
pa0 间隔(1s)高低电平翻转。

启用：Bandgap，LVR，IHRC。

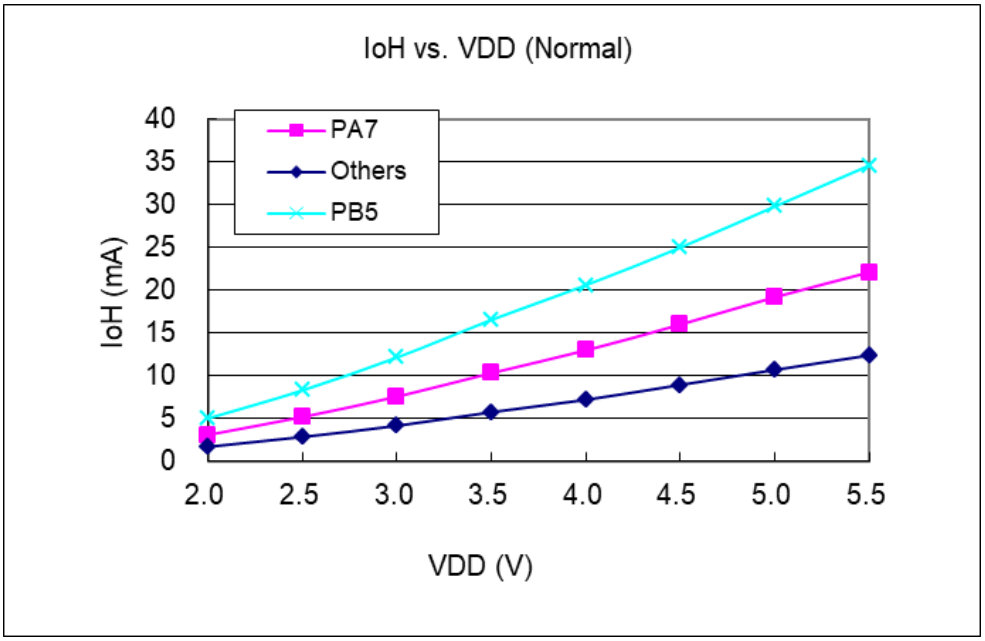
停用：t16 定时器，中断，ILRC，**触摸功能**，且 IO 引脚不悬空。

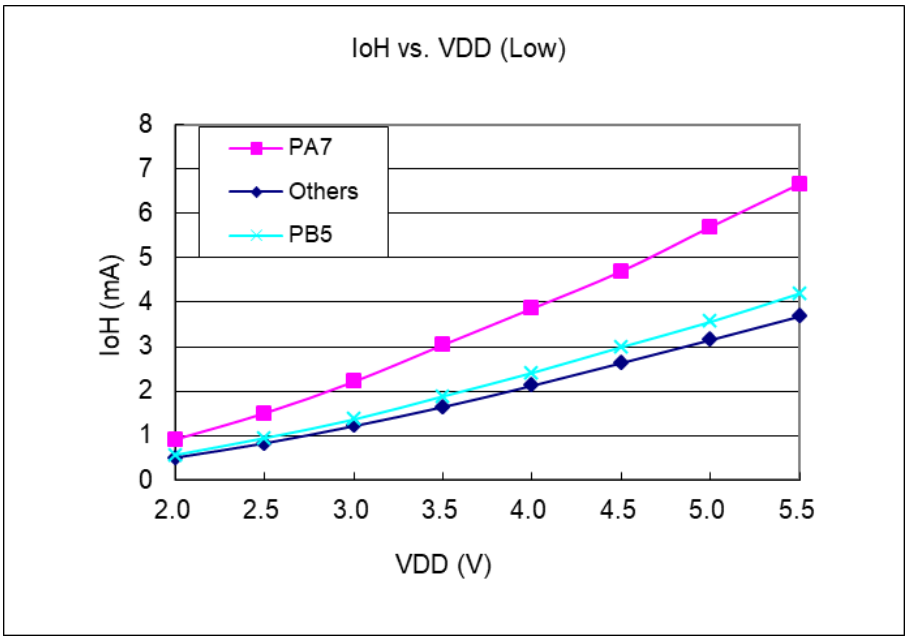


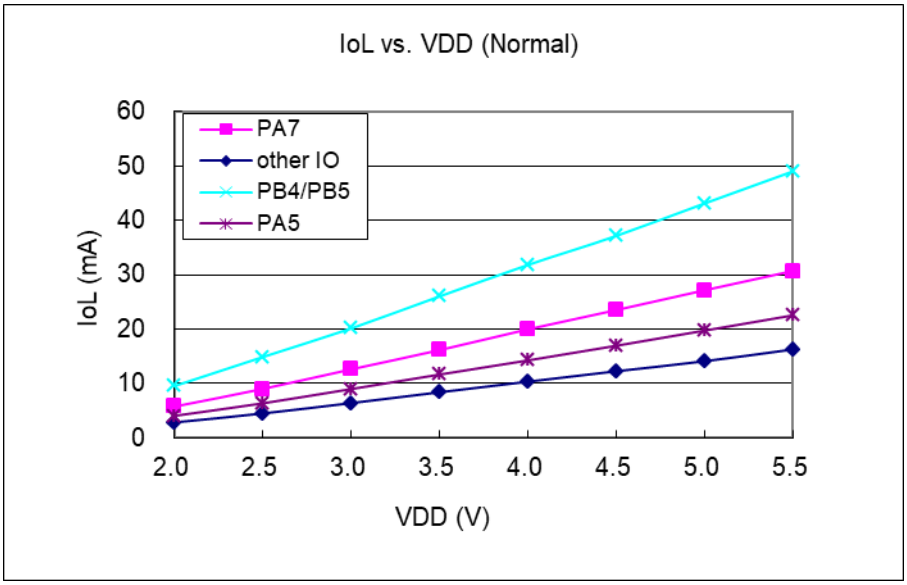
4.9. IO 引脚上拉阻抗曲线图

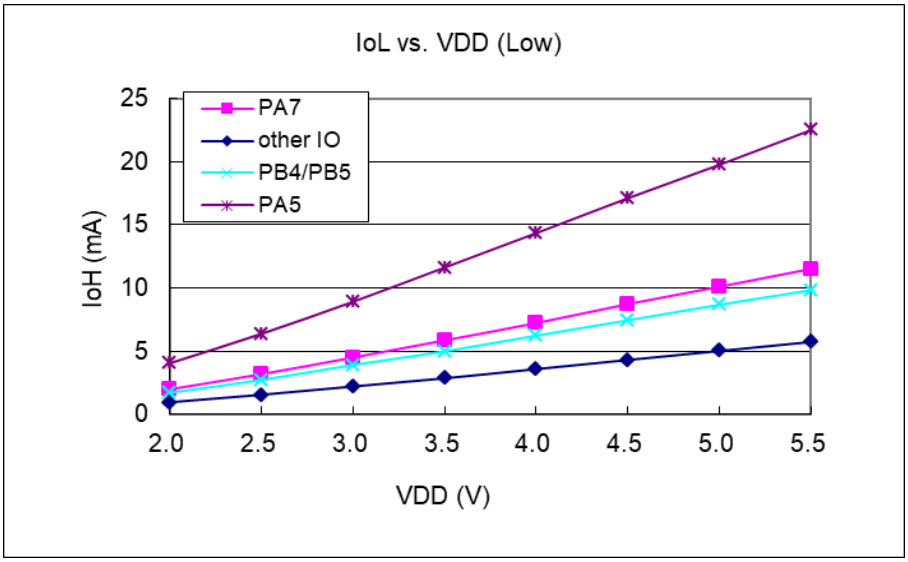


4.10. IO 引脚输出的驱动电流(I_{OH})与灌电流(I_{OL})曲线图

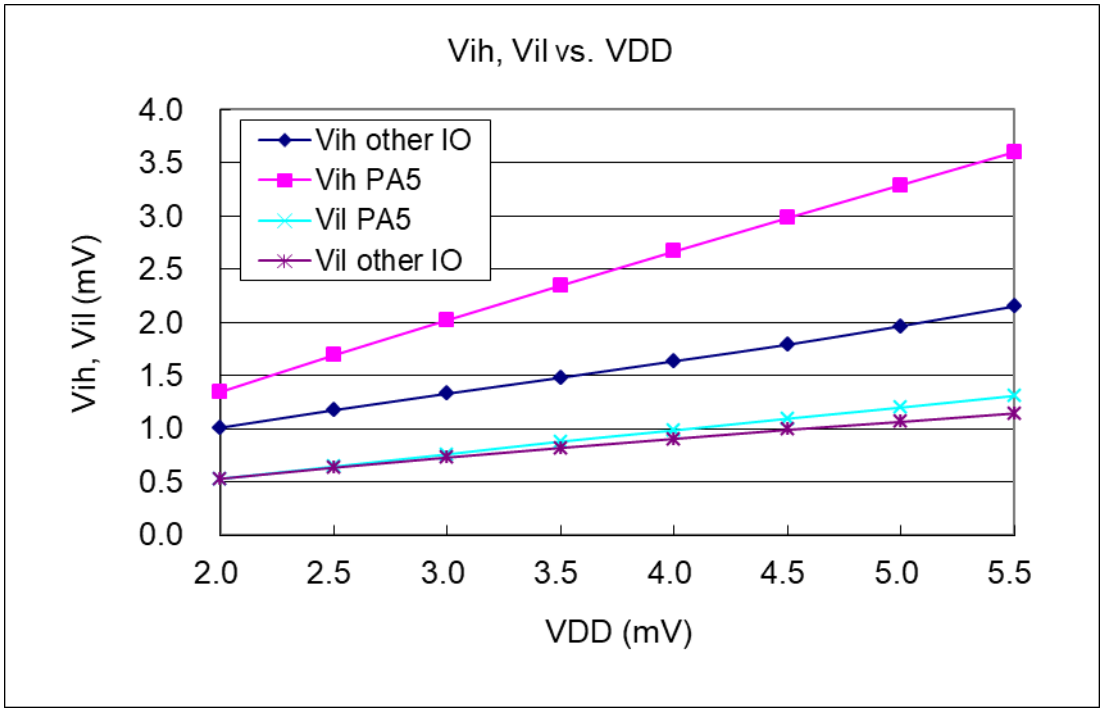



IoH vs. VDD (Low)
PA7
Others
PB5
IoH (mA)
VDD (V)
0
1
2
3
4
5
6
7
8
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5

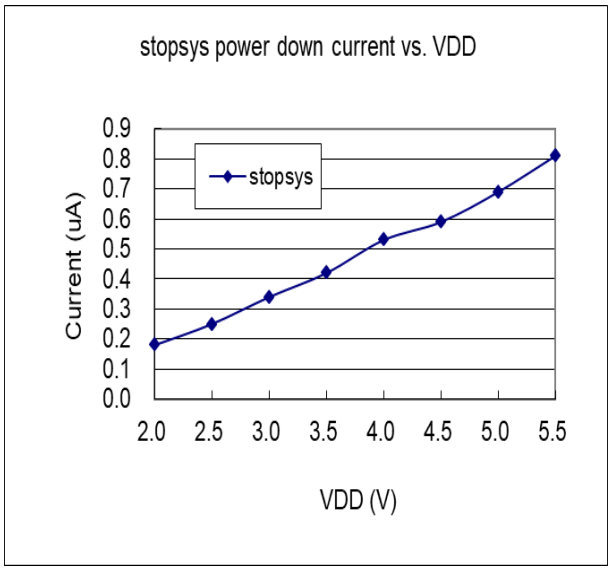

IoL vs. VDD (Normal)
PA7
other IO
PB4/PB5
PA5
IoL (mA)
VDD (V)
0
10
20
30
40
50
60
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5

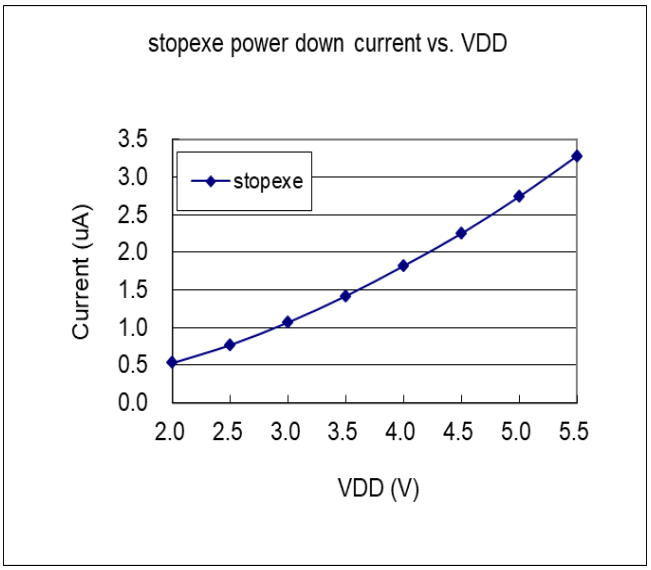

IoL vs. VDD (Low)
PA7
other IO
PB4/PB5
PA5
IoH (mA)
VDD (V)
0
5
10
15
20
25
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5

4.11. IO 引脚输入高/低阀值电压(V_{IH}/V_{IL})曲线图



4.12. 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。PMS164 的 OTP 程序内存容量为 1.75KW 如表 1 所示。OTP 内存从地址“0x700 ~0x7FF”供系统使用，从 0x001 到 0x00F 和从 0x011 到 0x6FF 地址空间是用户的程序空间。

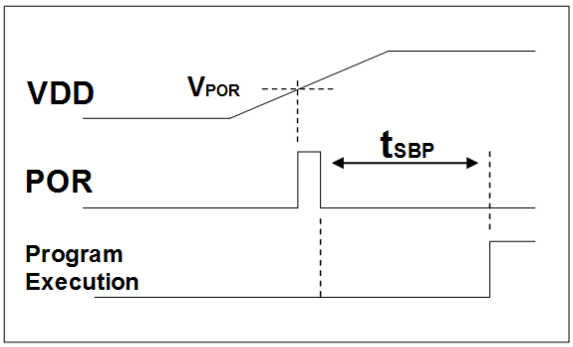
地址	功能
0x000	用于 FPP0 复位，goto 主程序
0x001	用户程序区
•	•
•	•
0x00F	用户程序区
0x010	中断入口地址
0x011	用户程序区
•	•
0x6FF	用户程序区
0x700	系统使用
•	•
0x7FF	系统使用

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 PMS164，正常开机的开机时间是 3000 个 ILRC 时钟周期。用户在使用时，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。

请注意：上电时，VDD 电压必须先到达及超过 V_{POR}，MCU 才会起动。



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，PMS164 的所有 128 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

PMS164 有两个振荡器电路:内部高频 RC 振荡器(IHRC) 和内部低频振荡器(ILRC)，这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。用户可以选择不同的振荡器作为系统时钟源，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

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 和 ILRC 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 V_{DD}、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。

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，PMS164 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**)

p1=2, 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	描述
○ Set IHRC / 2	= 34h (IHRC / 2)	有校准	IHRC 校准到 16MHz, CLK=8MHz (IHRC/2)
○ 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，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，PMS164 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，PMS164 执行此命令后的状态：

(1) .ADJUST_IC SYSCLK=IHRC/2, IHRC=16MHz, V_{DD}=5V

开机后，CLKMD = 0x34：

- IIHRC 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/2 = 8MHz
- 看门狗计数器停用，ILRC 启用，PA5 引脚是输入模式

(2) .ADJUST_IC SYSCLK=IHRC/4, IHRC=16MHz, V_{DD}=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

(3) .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

(4) .ADJUST_IC SYSCLK=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

(5) .ADJUST_IC SYSCLK=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

(6) .ADJUST_IC SYSCLK=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

(7) .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 模块是停用的。
- 系统频率= ILRC
- 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，PMS164 的时钟系统的硬件框图，如图 3 所示：

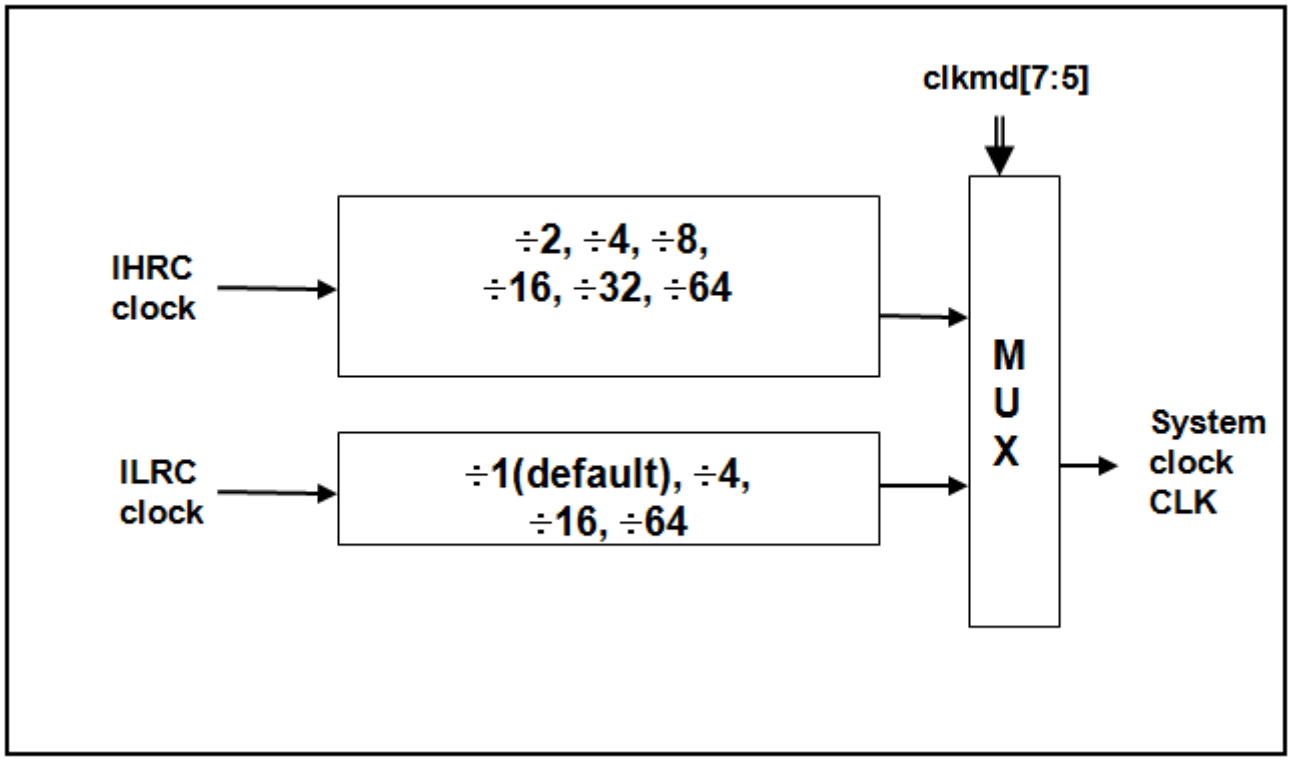


图 3：系统时钟选项

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，PMS164 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**，下面这些例子显示更多时钟切换需知道的信息，请参阅 IDE 工具“求助” -> “使用手册” -> “IC 介绍” -> “缓存器介绍” -> CLKMD” 。

例 1： 系统时钟从 ILRC 切换到 IHRC/2

```
...                         //  系统时钟是 ILRC
CLKMD       =   0x34;       //  切换到 IHRC/2，ILRC 不能在这里停用
CLKMD.2     =   0;          //  ILRC 可以在这里停用
...
```

例 2： 系统时钟从 IHRC/2 切换到 ILRC

```
...                         //  系统时钟是 IHRC/2
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/2 切换到 IHRC/4

```
...                         //  系统时钟是 IHRC/2，ILRC 在这里是启用的
CLKMD       =   0X14;       //  切换到 IHRC/4
...
```

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/2 同时关闭 ILRC 振荡器
```

5.5. 比较器

PMS164 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)，PB6，PB7，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。

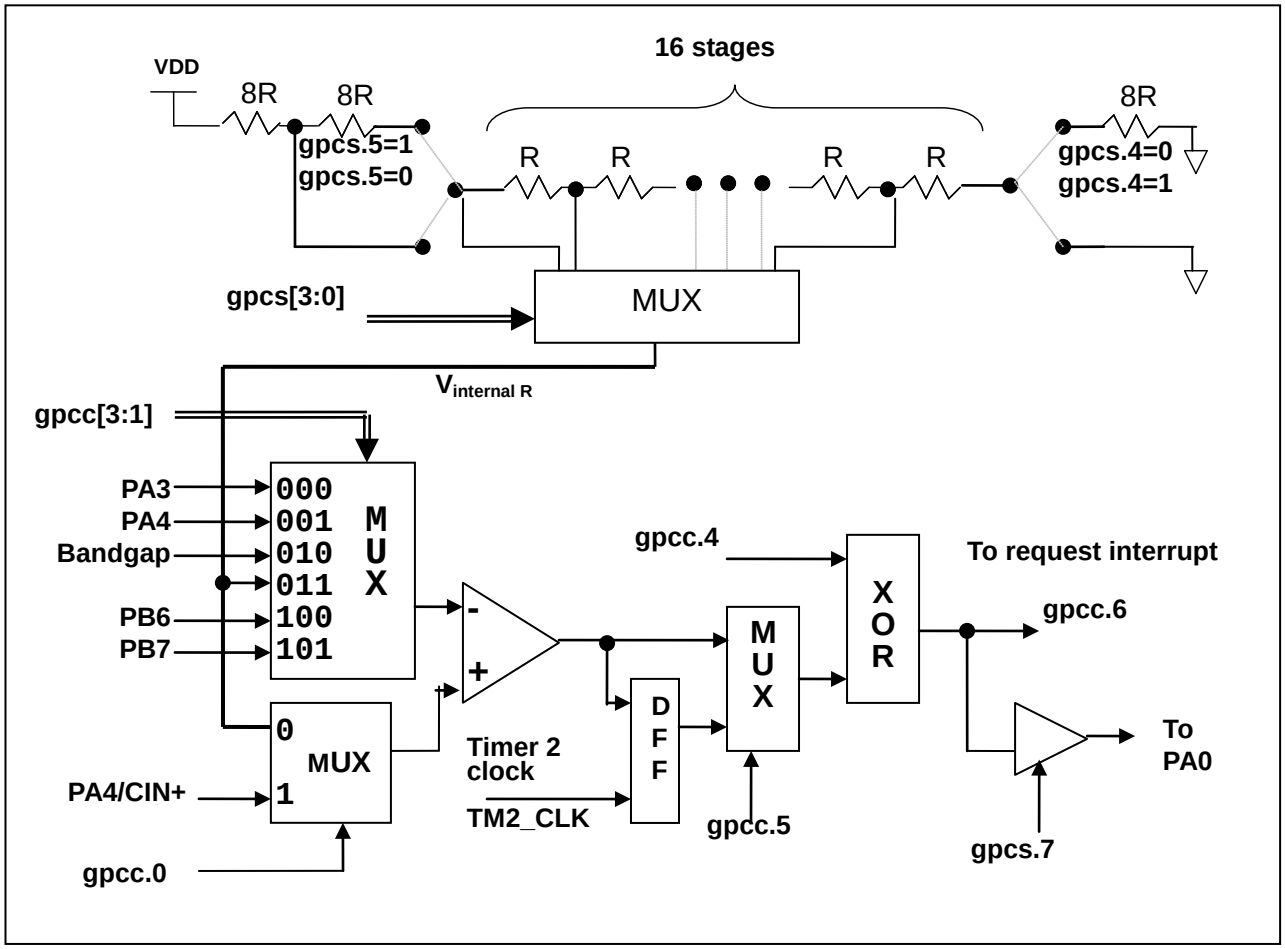


图 4：比较器硬件原理框图

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}。

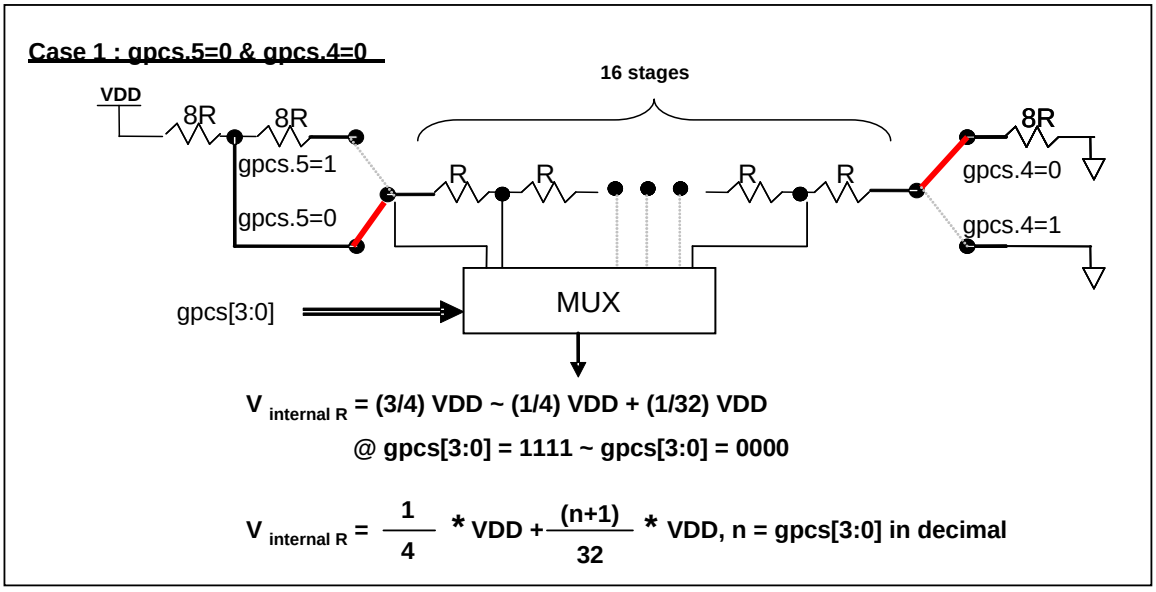


图 5：$V_{internal\ R}$ 硬件接法 (gpcs.5=0 & gpcs.4=0)

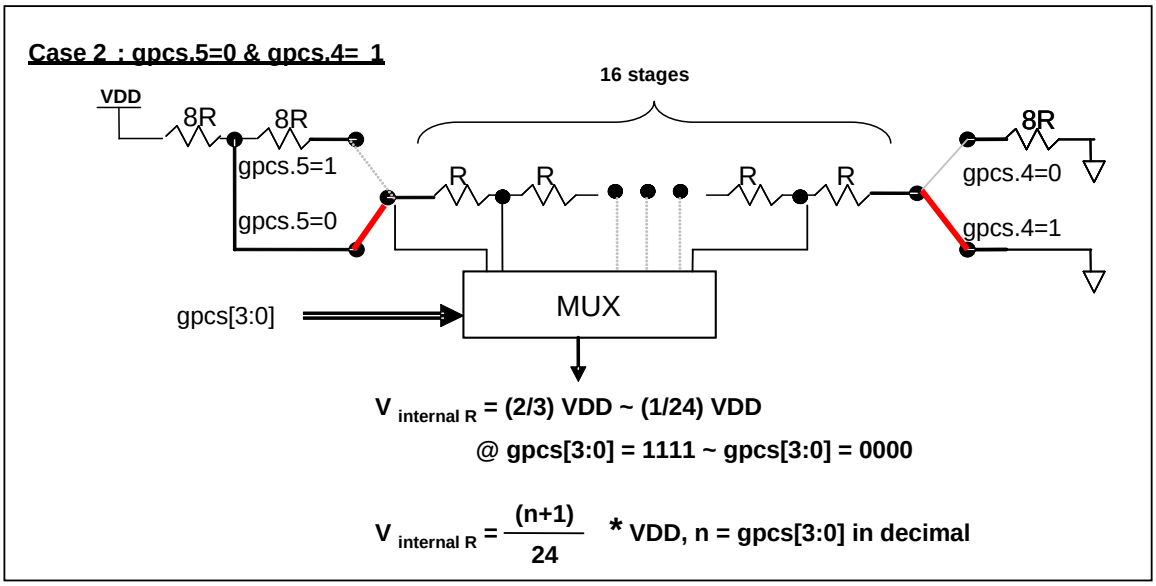


图 6：$V_{internal\ R}$ 硬件接法 (gpcs.5=0 & gpcs.4=1)

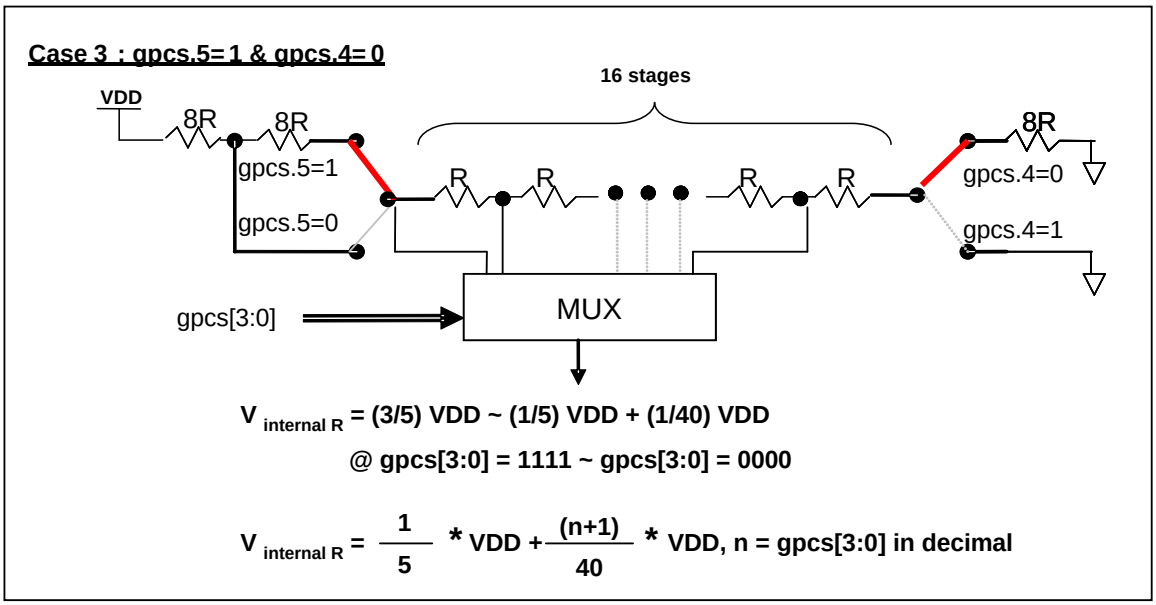


图 7：$V_{internal\ R}$ 硬件接法 (gpcs.5=1 & gpcs.4=0)

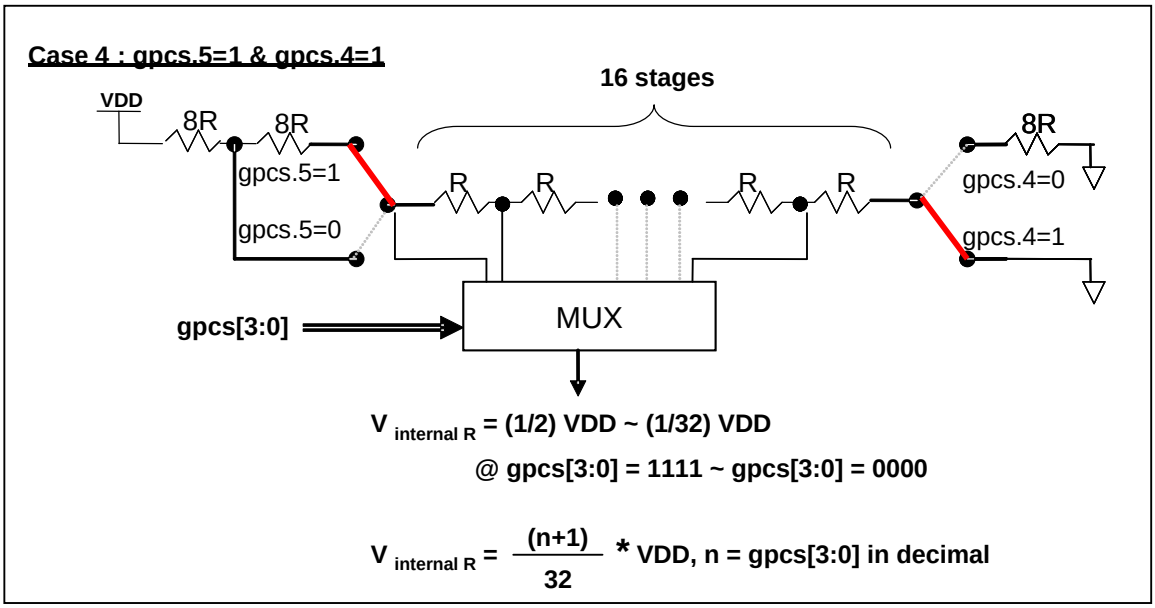


图 8：$V_{internal\ R}$ 硬件接法 (gpcs.5=1 & gpcs.4=1)

5.5.2. 使用比较器

例 I：

选择 PA3 为负输入和 $V_{internal\ R}$ 的电压为(18/32)*V_{DD} 作为正输入。 $V_{internal\ R}$ 选择上图 gpcs[5:4] = 2b'00 的配置方式，gpcs [3:0] = 4b'1001 (n=9)以得到 $V_{internal\ R}$ = (1/4)*V_{DD} + [(9+1)/32]*V_{DD} = [(9+9)/32]*V_{DD} = (18/32)*V_{DD} 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 $V_{internal\ R}$ 为负输入，$V_{internal\ R}$ 的电压为(22/40)*V_{DD}，选择 PA4 为正输入，比较器的结果将反极性并输出到 PA0。$V_{internal\ R}$ 选择上图的配置方式 "gpcs[5:4] = 2b'10" 和 gpcs[3:0] = 4b'1101 (n=13) 得到 $V_{internal\ R}$ = (1/5)*V_{DD} + [(13+1)/40]*V_{DD} = [(13+9)/40]*V_{DD} = (22/40)*V_{DD}。

```
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;
```

注意：当 GPCS 选择 Output 到 PA0 输出时，仿真器的 PA3 输出功能会受影响，但 IC 是正确的，所以模拟时请注意避开这错误。

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 $V_{internal\ R}$ 比较。$V_{internal\ R}$ 的电源是 V_{DD}，利用调整 $V_{internal\ R}$ 电压水平和 Bandgap 参考电压比较，就可以知道 V_{DD} 的电压。如果 N（***gpcs***[3:0]十进制）是让 $V_{internal\ R}$ 最接近 1.20V，那么 V_{DD} 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)

PMS164 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。

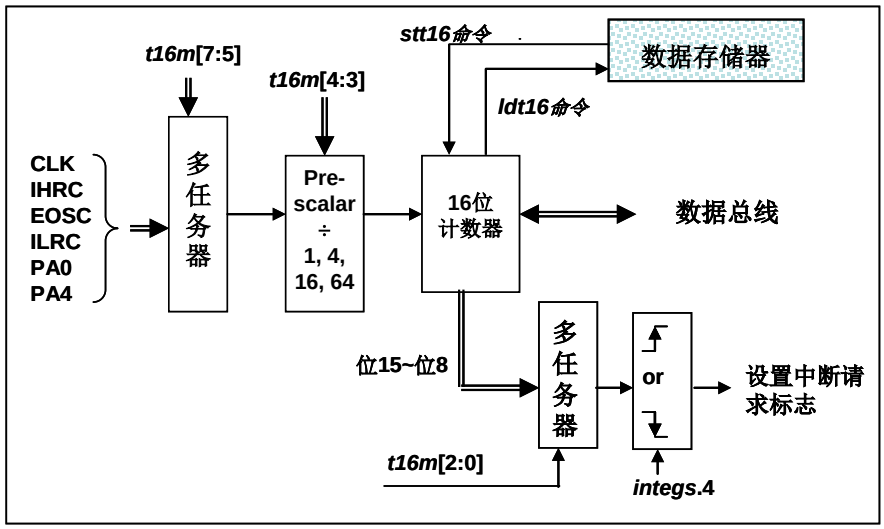


图 9：Timer16 模块框图

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      // 1st par.
$ 4~3:      /1, /4, /16, /64                                  // 2nd par.
$ 2~0:      BIT8, BIT9, BIT10, BIT11, BIT12, BIT13, BIT14, BIT15   // 3rd par.
```

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，PMS164 将复位并重新运行程序。看门狗时序图如图 10 所示。

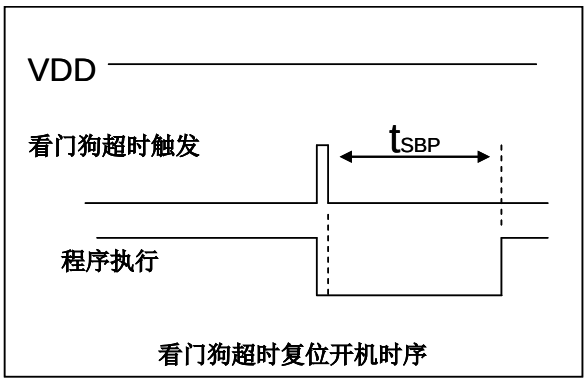


图 10：看门狗超时溢出时序图

5.8. 中断

PMS164 有 8 个中断源：

- 外部中断源 PA0 / PA5
- 外部中断源 PB0
- Timer16 中断
- Timer2 中断
- Timer3 中断
- GPC 中断
- 两个触摸按键中断 (TK_OV and TK_END)

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（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。

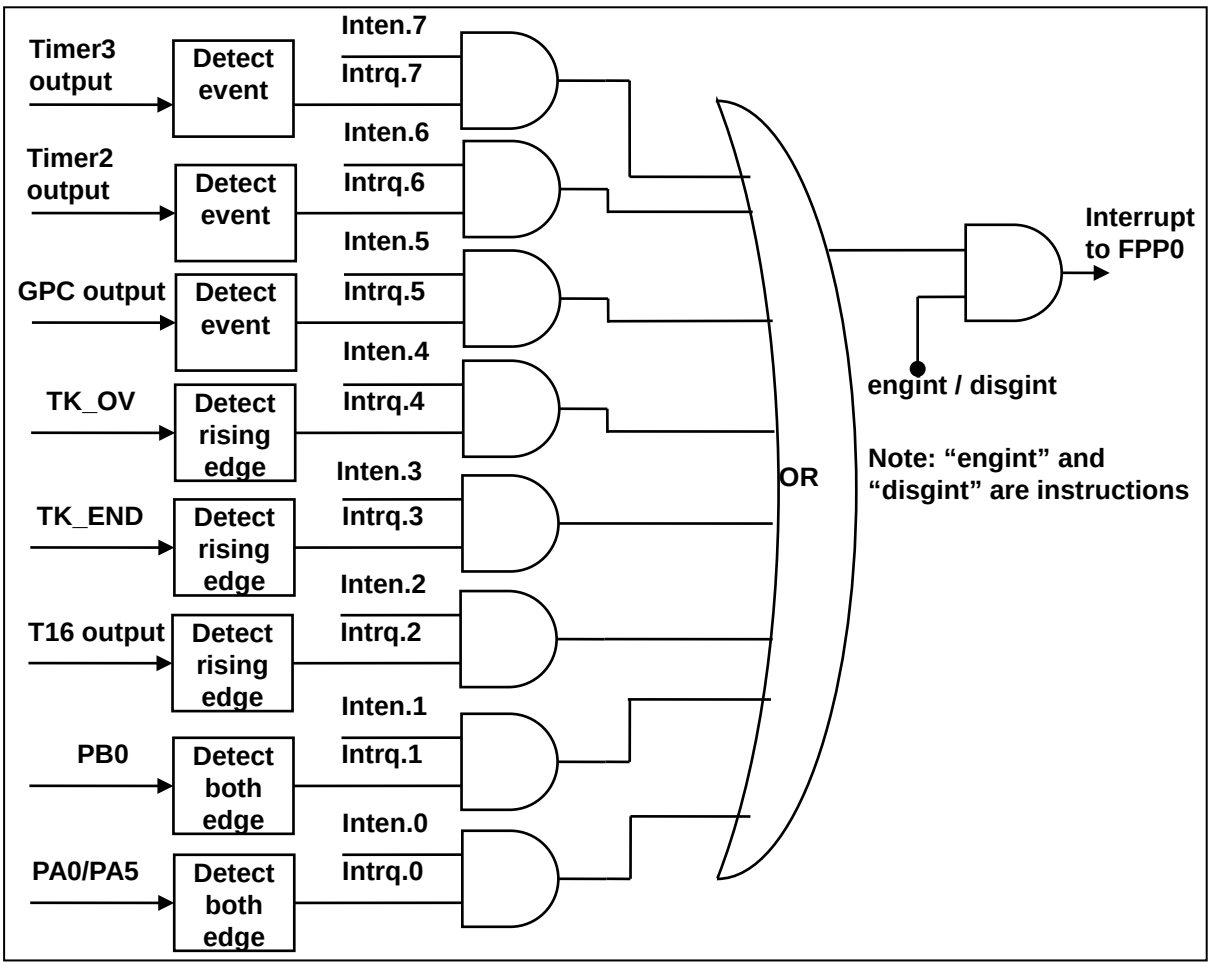


图 11：中断控制器硬件框图

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

PMS164 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
STOPSYS	停止	停止
STOPEXE	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）。
- 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 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. 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。

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，PMS164 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：设定为数字模式（PxDIER 对应位为 1）的 IO 切换。

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

```
CLKMD    =    0xF4;          //    系统时钟从 IHRC 变为 ILRC，关闭看门狗时钟
CLKMD.4  =    0;             //    IHRC 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    =    0x34;          //    系统时钟从 ILRC 变为 IHRC/2
```

5.9.3. 唤醒

进入掉电或省电模式后，PMS164 可以通过切换 IO 引脚恢复正常工作；而 Timer 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	定时器唤醒
STOPSYS	是	否
STOPEXE	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 PMS164, ***padier*** 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 2048 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 模式	正常唤醒	2048 * T_{ILRC}， 这里 T_{ILRC} 是指 ILRC 时钟周期
STOPSYS 模式	正常唤醒	2048 * T_{ILRC}， 这里 T_{ILRC} 是指 ILRC 时钟周期

表 6：休眠模式/唤醒模式 /IO 唤醒时间

5.10. IO 引脚

除了 PA5 外，所有 IO 引脚都具有相同的结构，PA5 只有开漏输出（图 12 中没有 Q1），但通过设置 ***paph.***5，其上拉电阻仍然有效。当 PMS164 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。

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 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***	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 7：PA0 设定配置表

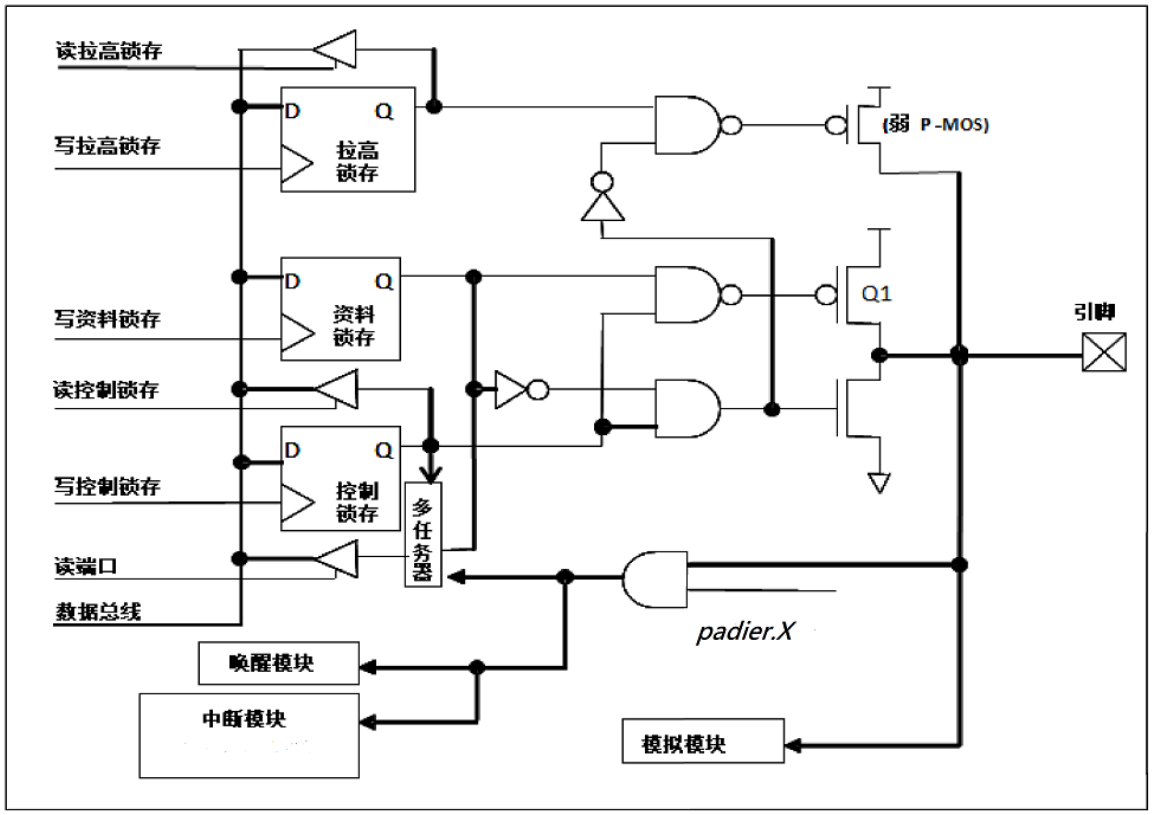


图 12：IO 引脚缓冲区硬件图

除了 PA5 外，所有的 IO 引脚具有相同的结构；PA5 的输出只能是漏极开路模式（没有 Q1）。对于被选择为仿真功能的引脚，必须在寄存器 ***padier*** 相应位设置为低，以防止漏电流。当 PMS164 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 用作外部中断引脚时，***padier***.0 应设置为高。

5.11. 复位

引起 PMS164 复位的原因很多，一旦复位发生，PMS164 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。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 PRSTB 引脚或 WDT 超时溢位，数据存储器的值将被保留。

5.12. 8 位 PWM 计数器 (Timer2/Timer3)

PMS164 内置 2 个 8 位硬件 PWM 计数器 (Timer2/Timer3)。以下描述只以 Timer2 为例，因为 Timer3 和 Timer2 结构是一样的。图 13 为 Timer2 硬件框图，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)，PA0，PB0，PA4 和比较器。寄存器 tm2c 的位[7:4]用来选择 Timer2 的时钟。如果 IHRC 作为 Timer2 的时钟源,当仿真器停住时，IHRC 时钟仍然会送到 Timer2，所以 Timer2 仍然会计数。依据寄存器 tm2c[3:2]的设定，Timer2 的输出可以选择性输出到 PB2，PA3 或 PB4(Timer3 的计数输出可选择为 PB5，PB6 或 PB7)。此时无论 PX.x 是输入还是输出的状态，Timer2（或 Timer3）的信号都会被强制输出。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 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 位到 8 位。图 14 显示出 Timer2 周期模式和 PWM 模式的时序图。

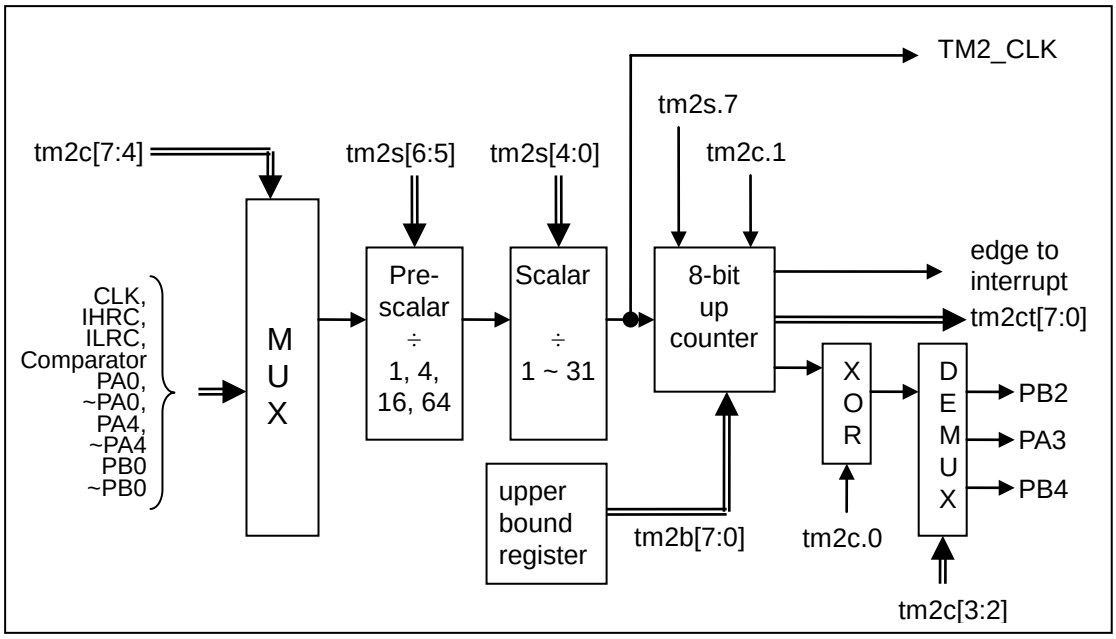


图 13：Timer2 硬件框图

Timer3 的输出是 PB5，PB6 或 PB7。

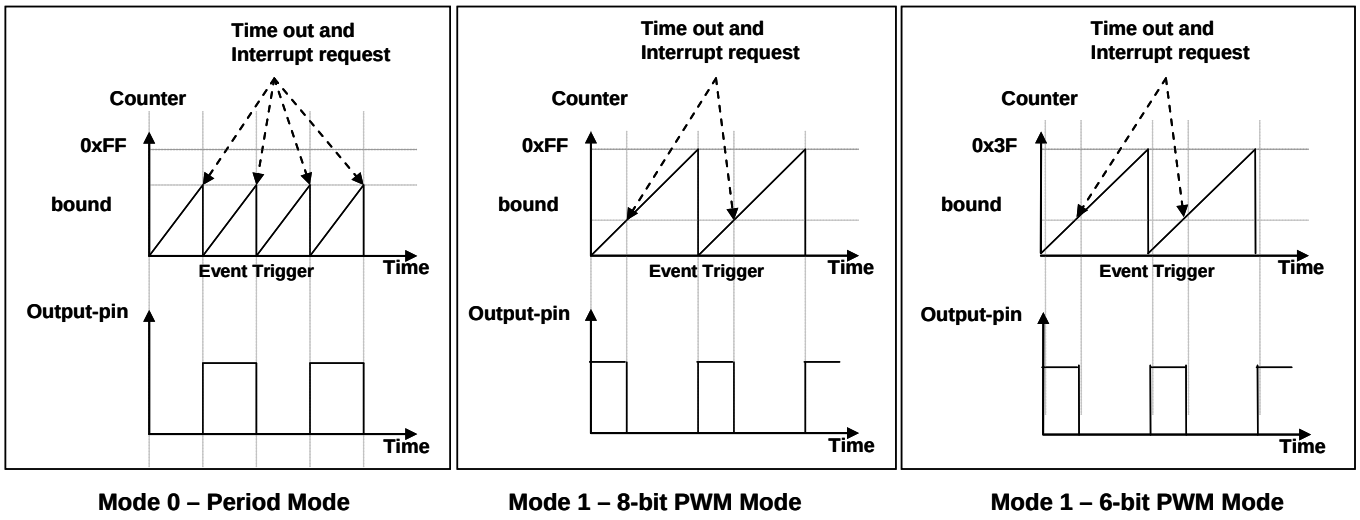


图 14：Timer2 周期模式和 PWM 模式的时序图(tm2c.1=1)

程序选项 ”GPC_PWM“ 是指根据需求由比较器结果控制生成 PWM 波形的功能。如果程序选项 “GPC_PWM”被选中后，此时当比较器输出是 1 时，PWM 停止输出；而比较器输出是 0 时，PWM 恢复输出，如图 15 所示。

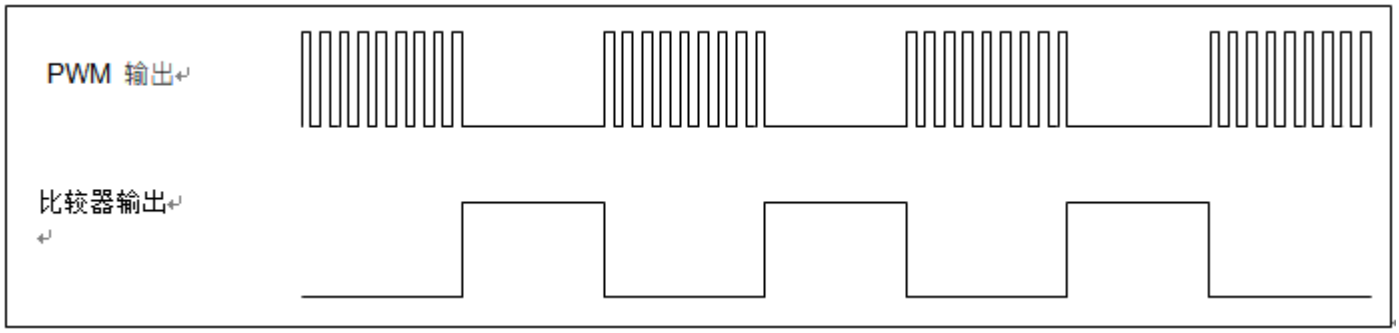


图 15：比较器控制 PWM 输出

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;
    }
}
```

5.12.2. 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)]

输出占空比= [(K＋1) ÷ 256]× 100%

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

例 1：

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s = 0b0000_00000, S1=1, S2=0
➔ 输出频率 = 8MHz ÷ (256 × 1 × (0+1)) = 31.25kHz
➔ 输出占空比 = [(127+1) ÷ 256] × 100% = 50%

例 2：

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s = 0b0111_11111, S1=64, S2=31
➔ 输出频率 = 8MHz ÷ (256 × 64 × (31+1)) = 15.25Hz
➔ 输出占空比 = [(127+1) ÷ 256] × 100% = 50%

例 3：

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b1111_1111, K=255
tm2s = 0b0000_00000, S1=1, S2=0
➔ 输出频率 = 8MHz ÷ (256 × 1 × (0+1)) = 31.25kHz
➔ 输出占空比 = [(255+1) ÷ 256] × 100% = 100%

例 4：

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0000_1001, K = 9
tm2s = 0b0000_00000, S1=1, S2=0
➔ 输出频率 = 8MHz ÷ (256 × 1 × (0+1)) = 31.25kHz
➔ 输出占空比 = [(9+1) ÷ 256] × 100% = 3.9%

使用 Timer2 定时器从 PA3 产生 PWM 波形的示例程序如下所示：

```
void    FPPA0   (void)
{
    .ADJUST_IC    SYSCLK=IHRC/2, IHRC=16MHz, VDD=5V
    wdreset;
    tm2ct = 0x00;
    tm2b = 0x7f;
    tm2s = 0b0_00_00001;            //   8-bit 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;
    }
}
```

5.12.3. 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)]

输出占空比 = [(K＋1) ÷ 64] × 100%

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

例 1：

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0001_1111, K=31
tm2s = 0b1000_00000, S1=1, S2=0
➔ 输出频率 = 8MHz ÷ (64 × 1 × (0+1)) = 125kHz
➔ 输出占空比 = [(31+1) ÷ 64] × 100% = 50%

例 2：

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0001_1111, K=31
tm2s = 0b1111_11111, S1=64, S2=31
➔ 输出频率= 8MHz ÷ (64 × 64 × (31+1)) = 61.03 Hz
➔ 输出占空比 = [(31+1) ÷ 64] × 100% = 50%

例 3：

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0011_1111, K=63
tm2s = 0b1000_00000, S1=1, S2=0
➔ 输出频率 = 8MHz ÷ (64 × 1 × (0+1)) = 125kHz
➔ 输出占空比 = [(63+1) ÷ 64] × 100% = 100%

例 4：

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0000_0000, K=0
tm2s = 0b1000_00000, S1=1, S2=0
➔ 输出频率= 8MHz ÷ (64 × 1 × (0+1)) = 125kHz
➔ 输出占空比 = [(0+1) ÷ 64] × 100% =1.5%

5.13. 触摸功能

PMS164 内含一个触摸检测电路，图 16 为其功能方框图：

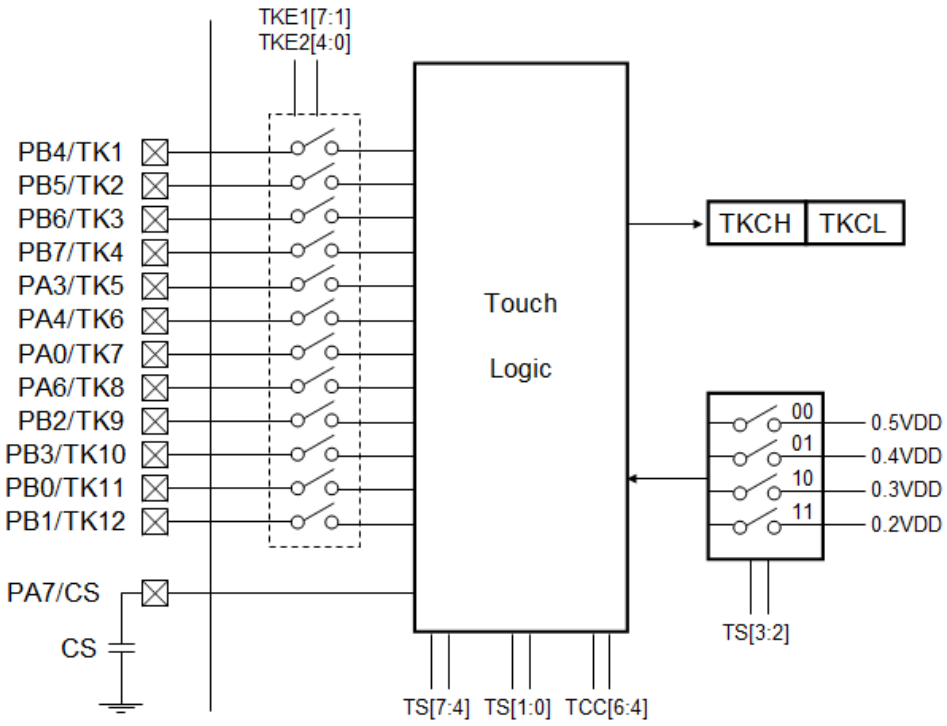


图 16：触摸检测电路的功能方框图

PMS164 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。

使用该功能时，需要在 PA7/CS 引脚和 GND 之间连接一颗精确而低漏电率的外部电容器 CS。同时，用户应将代码选项 PA7_Sel 设置为 As_CS，将其配置为 CS 引脚，而不是 PA7。

要开动触摸检测，使用者应跟从以下步骤：

1. 用户通过设置 TKE1 和 TKE2 寄存器来选择要测量的感应极片（引脚）。每次应只选择一个感应极片。
2. 用户可通过将“0x10”写入 TCC 寄存器以发出 Touch START 命令。电容 CS 首先被完全放电到 VSS，放电时间可以透过 ***TS[1:0]***从 32，64 和 128 个触摸电路时钟周期中选择。
3. 电容值越大，将电容器完全放电到 VSS 所需的放电时间就越长。然而有些情况下，128 个触控时钟仍不足以把 CS 电容完全放电，这时用户应通过将“0x30” 而不是“0x10”写入 TCC 寄存器来启动此手动放电过程。在由用户控制的一定放电时间之后，用户可以发出 Touch START (0x10)命令来继续此触摸转换进程， 或者使用者也可以通过将“0x00”写入 TCC 寄存器中止转换进程。
4. 在放电之后，CS 会在每个触摸时钟周期（TK_CLK）朝着 VCC 充电。充电速度是由所选感应极片的电容值决定。

5. 当其电压达到内部产生的阈电压 VREF 时，充电进程将自动停止。程序可以透过读取 TCC[6:4] 或是 INTRQ[3] 来判断充电过程是否停止。VREF 电压可透过 ***TS[3:2]***，在 0.2*VCC，0.3*VCC，0.4*VCC 和 0.5*VCC 间选择。
6. 经过读取触摸计数器 TKCH 和 TKCL 的值，用户可监测感应极片上的电容量变化。 读取到的值与 CS 和 CP 的比例有关，而 CP 表示电容可以通过因用户手指的触摸而变化的 PCB，导线和触摸板的组合的总电容。一旦 CP 值被改变，将 CS 充电到 VREF 所需的时间缩短。通过计数时钟周期的差异，电路可以决定触摸板是否启用。

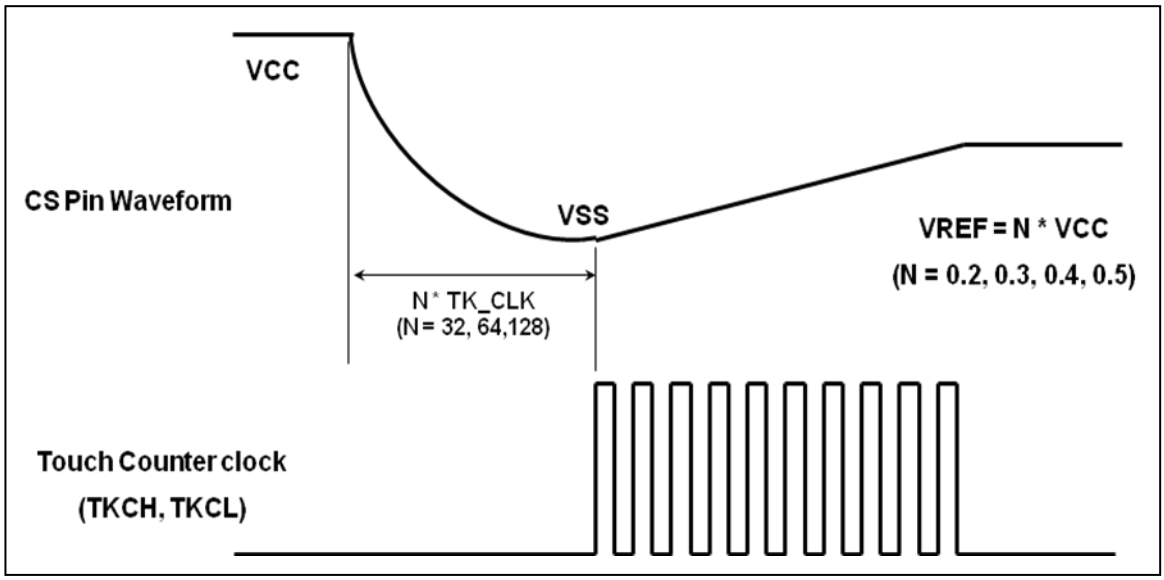


图 17：触摸转换的时序图

注意： 当 VREF 电压首次设置或者中途切换参考电压选项时，请舍弃在此之后读取到的第一笔 *TKCH* 和 *TKCL* 的数据。

6. IO 寄存器

6.1. ACC 状态标志寄存器 (*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*)，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. 时钟模式寄存器 (*clkmd*)，IO 地址 =0x03

位	初始值	读/写	描述	
7 - 5	111	读/写	系统时钟 (CLK)选择：	
			类型 0，clkmd[3]=0	类型 1，clkmd[3]=1
			000：IHRC/4 001：IHRC/2 01x：保留 100：保留 101：保留 110：ILRC/4 111：ILRC（默认）	000：IHRC/16 001：IHRC/8 010：ILRC/16（仿真器不支持） 011：IHRC/32 100：IHRC/64 110：ILRC/64（仿真器不支持） 1x1：保留
4	0	读/写	内部高频 RC 振荡器功能。 0/1：停用/启用	
3	0	读/写	时钟类型选择。这个位是用来选择位 7~位 5 的时钟类型。 0/1：类型 0 /类型 1	
2	1	读/写	内部低频 RC 振荡器功能。0/1：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。	
1	1	读/写	看门狗功能。0/1：停用/启用	
0	0	读/写	引脚 PA5/PRSTB 功能。0/1：PA5 / PRSTB	

6.4. 中断允许寄存器 (*inten*)，IO 地址 =0x04

位	初始值	读/写	描述
7	0	读/写	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	使能比较器中断。0/1：停用/启用
4	0	读/写	使能触摸按键中断 TK_OV。0/1：停用/启用
3	0	读/写	使能触摸按键中断 TK_END。0/1：停用/启用
2	0	读/写	使能 Timer16 溢出中断。0/1：停用/启用
1	0	读/写	使能 PB0 中断。0/1：停用/启用
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*)，IO 地址 =0x05

位	初始值	读/写	描述
7	-	读/写	Timer3 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	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
4	-	读/写	触摸按键 TK_OV 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	-	读/写	触摸按键 TK_END 的中断请求，此位是由硬件置位并由软件清零。0/1：不要求/请求
2	-	读/写	Timer16 的中断请求，此位是由硬件置位并由软件清零。0/1：不要求/请求
1	-	读/写	引脚 PB0 的中断请求，此位是由硬件置位并由软件清零。0/1：不要求/请求
0	-	读/写	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*)，IO 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. 杂项寄存器 (*misc*)，IO 地址 = 0x08

位	初始值	读/写	描述
7 - 3	-	-	保留
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

6.8. 外部晶体振荡寄存器（*eoscr*，只写），IO 地址 =0x0a

位	初始值	读/写	描述
7 - 1	-	-	保留。为 0。
0	0	只写	将 Bandgap 和 LVR 硬件模块断电。0/1：正常/断电

6.9. 中断边缘选择寄存器（*integs*），IO 地址 =0x0c

位	初始值	读/写	描述
7 - 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	00	只写	PB0 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
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

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

位	初始值	读/写	描述
7 – 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。
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。

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

位	初始值	读/写	描述
7 - 1	0xFF	只写	使能 PB7 ~ PB1 数字输入和唤醒事件。1 / 0：启用/ 停用 如果这些位设为 0，PB7 ~ PB1 则不能用来唤醒系统。
0		只写	使能 PB0 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，PB0 则不能用来唤醒系统，并且停用中断请求。

6.12. 端口 A 数据寄存器 (*pa*)，IO 地址 =0x10

位	初始值	读/写	描述
7 - 0	8'h00	读/写	资料寄存器的端口 A。

6.13. 端口 A 控制寄存器 (*pac*)，IO 地址 =0x11

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

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

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

6.15. 端口 B 数据寄存器 (*pb*)，IO 地址 =0x14

位	初始值	读/写	描述
7 - 0	0'h00	读/写	资料寄存器的端口 B。

6.16. 端口 B 控制寄存器 (*pbc*)，IO 地址 =0x15

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

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

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

6.18. 比较器控制寄存器 (*gpcc*)，IO 地址 =0x18

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：PB6（不适用 EV5） 101：PB7（不适用 EV5） 11X：保留
0	0	读/写	选择比较器正输入的来源。 0：$V_{internal\ R}$ 1：PA4

6.19. 比较器选择寄存器 (*gpcs*)，IO 地址 =0x19

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 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）

6.20. 状态复位寄存器 (*rstst*)，IO 地址 =0x1b

位	初始值 (只有 POR)	读/写	描述
7	0	读/写	看门狗溢出复位标志。当发生看门狗复位时，该位为 1。 此位写入 0 或上电复位（POR）后清除该标志。
6	0	读/写	无效代码复位标志。当 MCU 从无效代码复位时，该位为 1。 此位写入 0 或上电复位（POR）后清除该标志。
5	0	-	保留（写 0）。
4	-	-	保留（写 1）。
3	-	读/写	外部复位引脚（PA5）的复位标志。当发生外部引脚复位时，该位为 1。 此位写入 0 即清除该标志。
2	-	读/写	VDD 低于 4V 标志。当 VDD 电压低于 4V 时，该位为 1。 此位写入 0 即清除该标志。 请注意，当 VDD 上电较缓慢时，此位亦会自动置 1。如有需要，建议客户在程序初始化阶段将此位清零。
1	-	读/写	VDD 低于 3V 标志。当 VDD 电压低于 3V 时，该位为 1。 此位写入 0 即清除该标志。 请注意，当 VDD 上电较缓慢时，此位亦会自动置 1。如有需要，建议客户在程序初始化阶段将此位清零。
0	-	读/写	VDD 低于 2V 标志。当 VDD 电压低于 2V 时，该位为 1。 此位写入 0 即清除该标志。 请注意，当 VDD 上电较缓慢时，此位亦会自动置 1，如有需要，建议客户在程序初始化阶段将此位清零。

6.21. 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：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） 1010：PB0（上升沿） 1011：~PB0（下降沿） 1100：PA4（上升沿） 1101：~PA4（下降沿） 注意：在 ICE 模式且 IHRC 被选为 Timer2 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：停用 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：停用/启用

6.22. Timer2 计数寄存器 (*tm2ct*)，IO 地址 =0x1d

位	初始值	读/写	描述
7 - 0	0'h00	只读	Timer2 定时器位[7:0]。

请注意：Timer2 同时设计了 PWM 模式和周期模式，因此不要读 tm2ct 寄存器。

6.23. 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 位或 7 位（由 code option TMx_Bit 决定）
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。

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

位	初始值	读/写	描述
7 - 0	0'h00	只写	Timer2 上限寄存器。

6.25. Timer3 控制寄存器 (*tm3c*)，IO 地址 = 0x32

位	初始值	读/写	描述
7 - 4	0000	读/写	Timer3 时钟选择。 0000：disable 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） 1010：PB0（上升沿） 1011：~PB0（下降沿） 1100：PA4（上升沿） 1101：~PA4（下降沿） 注意：在 ICE 模式且 IHRC 被选为 Timer3 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	Timer3 输出选择。 00：停用 01：PB5 10：PB6 11：PB7
1	0	读/写	Timer3 模式选择。 0：周期模式 1：PWM 模式

0	0	读/写	启用 Timer3 反极性输出。 0 / 1 ： 停用/启用

6.26. Timer3 计数寄存器 (*tm3ct*)，IO 地址 = 0x33

位	初始值	读/写	描述
7 - 0	0'h00	读/写	Timer3 定时器位[7:0]。

6.27. Timer3 分频寄存器 (*tm3s*)，IO 地址 = 0x34

位	初始值	读/写	描述
7	0	只写	PWM 分辨率选择。 0：8 位 1：6 位或 7 位（由 code option TMx_Bit 决定）
6 - 5	00	只写	Timer3 时钟预分频器。 00：÷ 1 01：÷ 4 10：÷ 16 11：÷ 64
4 - 0	00000	只写	Timer3 时钟分频器。

6.28. Timer3 上限寄存器 (*tm3b*)，IO 地址 = 0x35

位	初始值	读/写	描述
7 - 0	0'h00	只写	Timer3 上限寄存器。

6.29. 触摸选项寄存器 (*ts*)，IO 地址 = 0x20

位	初始值	读/写	描述
7 - 4	-	读/写	触摸时钟选择 (TK_CLK) 0010：IHRC/4 0011：IHRC/8 0100：IHRC/16 0101：IHRC/32 0110：IHRC/64 0111：IHRC/128 1000：ILRC 其他：保留
3 - 2	00	读/写	触摸 VREF 选择 00：0.5 * VCC 01：0.4 * VCC 10：0.3 * VCC 11：0.2 * VCC
1 - 0	-	读/写	在开始触摸功能前选择放电时间 (TK_DISCHG) 00：保留 01：32 * CLK 10：64 * CLK 11：128 * CLK

6.30. 触摸充电控制寄存器 (*tcc*)，IO 地址 = 0x21

位	初始值	读/写	描述		
7	-	-	保留		
6 - 4	-	读/写	触摸控制和状态		
			数据	**命令(W)**	**状态(R)**
			000	TK_STOP （触摸模块掉电）	准备中/ 结束
			001	TK_RUN	运行中
			011	放电 （CS 电容放电）	放电中
			其他	保留	保留
3 - 0	-	-	保留		

6.31. 触摸按键使能 2 寄存器 (*tke2*)，IO 地址 =0x22

位	初始值	读/写	描述
7 - 5	-	-	保留
4	0	读/写	使能 PB1/TK12。0/1：停用/启用
3	0	读/写	使能 PB0/TK11。0/1：停用/启用
2	0	读/写	使能 PB3/TK10。0/1：停用/启用
1	0	读/写	使能 PB2/TK9。0/1：停用/启用
0	0	读/写	使能 PA6/TK8。0/1：停用/启用

6.32. 触摸按键使能 1 寄存器 (*tke1*)，IO 地址 =0x24

位	初始值	读/写	描述
7	0	读/写	使能 PA0/TK7。0/1：停用/启用
6	0	读/写	使能 PA4/TK6。0/1：停用/启用
5	0	读/写	使能 PA3/TK5。0/1：停用/启用
4	0	读/写	使能 PB7/TK4。0/1：停用/启用
3	0	读/写	使能 PB6/TK3。0/1：停用/启用
2	0	读/写	使能 PB5/TK2。0/1：停用/启用
1	0	读/写	使能 PB4/TK1。0/1：停用/启用
0	-	-	保留

6.33. 触摸按键充电计数高位寄存器(*tkch*)，IO 地址= 0x2B

位	初始值	读/写	描述
7 - 4	-	-	保留
3 - 0	-	只读	触摸按键充电计数的 tkc [11:8]

6.34. 触摸按键充电计数低位寄存器((*tkcl*)，IO 地址= 0x2C

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

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 PMS164 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 PMS164 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. 程序选项表

选项	选择	描述
Security	启用	OTP 内容加密，程序不允许被读取
	停用	OTP 内容不加密，程序可以被读取
LVR	4.0V	选择 LVR = 4.0V
	3.5V	选择 LVR = 3.5V
	3.0V	选择 LVR = 3.0V
	2.75V	选择 LVR = 2.75V
	2.5V	选择 LVR = 2.5V
	2.2V	选择 LVR = 2.2V
	2.0V	选择 LVR = 2.0V
	1.8V	选择 LVR = 1.8V
Drive	Low	低 IO 驱动电流/灌电流
	Normal	正常 IO 驱动电流/灌电流
TMx_Source	16MHZ	当 tm2c[7:4]= 0010, TM2 时钟源= IHRC = 16MHZ 当 tm3c[7:4]= 0010, TM3 时钟源= IHRC = 16MHZ
	32MHZ	当 tm2c[7:4]= 0010, TM2 时钟源 = IHRC*2 = 32MHZ 当 tm3c[7:4]= 0010, TM3 时钟源 = IHRC*2 = 32MHZ （仿真器不支持）
TMx_Bit	6 Bit	当 tm2s.7=1, TM2 是 6 位 PWM 当 tm3s.7=1, TM3 是 6 位 PWM
	7 Bit	当 tm2s.7=1, TM2 是 7 位 PWM 当 tm3s.7=1, TM3 是 7 位 PWM （仿真器不支持）
Comparator Edge	All Edge	上升缘和下降缘都触发中断
	Rising_Edge	仅上升缘触发中断
	Falling_Edge	仅下降缘触发中断
GPC_PWM	Disable	比较器和 PWM 相互独立
	Enable	比较器输出控制 PWM 输出（仿真器不支持）
Interrupt Src0	PA.0	选择 INTEN/INTRQ.Bit0 为 PA.0
	PA.5	选择 INTEN/INTRQ.Bit0 为 PA.5（仿真器不支持）
PA7_Sel	As_CS	配置引脚 PA7/CS 为触摸的 CS 脚
	As_IO	配置引脚 PA7/CS 为 PA7 IO 引脚
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

表 8：程序选项

9. 特别注意事项

此章节提醒用户在使用 PMS164 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。

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 和 PBDIER 寄存器，将对应的位设为 1

(3) PA5 设置为输出引脚

- PA5 只能做 Open Drain 输出，输出高需要外加上拉电阻

(4) 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

(5) PA5 作为输入并通过长导线连接至按键或者开关

- 必需在 PA5 与长导线中间串接 >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/2

```
CLKMD   =   0x36;          // 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 频率会在使用刻录器刻录程序时校准。

(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。

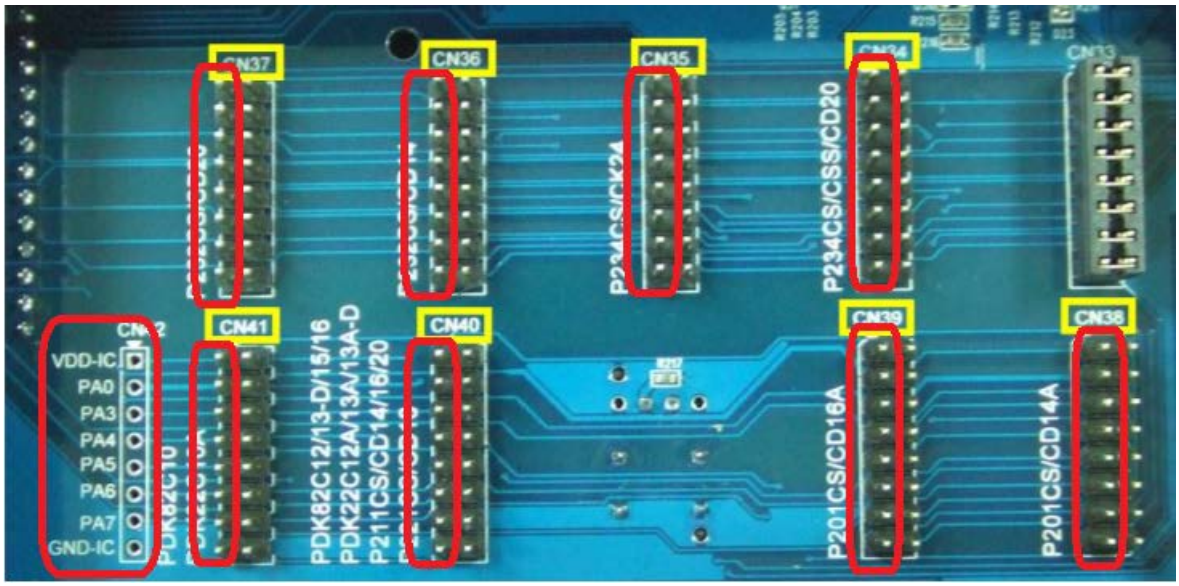
9.2.7 LVR

可以设定寄存器 MISC.2 为 1 将 LVR 关闭，但此时应确保 VDD 在 IC 的最低工作电压以上，否则 IC 不能工作。

9.2.8 PMS164 的刻录方法

PMS164 的刻录信号为 PA3，PA4，PA5，PA6，VDD，GND 这 6 只引脚。

在 3S-P-002 刻录器上，请把 jumper 放置在刻录器背后的 CN39 的位置。如果是刻录 S16 包装的 IC，请把 IC 放在正面的 Textool 的最高位置，接脚不用移位；而在 S08A 包装时，则需往下空移四格。如刻录其他包装的 IC，用户可以自行跳接刻录接脚。刻录器背后的所有 Jumper 的左侧引脚的讯号都是一致的，就如左下角 CN42 的说明文字一样，分别为 VDD，PA0（不需要），PA3，PA4，PA5，PA6，PA7（不需要），GND。



如使用 5S-P-003 或以上进行刻录，并依照刻录器软件上说明，连接 jumper 即可。

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

(1) PA5 （V_{PP}）可能高于 11V。

(2) V_{DD} 可能高于 6.5V，而最大供给电流最高可达约 20mA。

(3) 其他刻录引脚（GND 除外）的电位与 V_{DD} 相同。

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

9.3 使用 ICE

(1) 请使用 5S-I-S01/2(B) 仿真器。模拟时请注意以下几点：

- 5S-I-S01/2(B)不支持系统时钟为 ILRC/16 和 ILRC/64 的模拟
- 5S-I-S01/2(B)不支持 PA5 当中断源
- 5S-I-S01/2(B)不支持 GPC_PWM，TMx_source，TMx_bit 等 code option
- 5S-I-S01/2(B)不支持所有触摸功能
- 用 5S-I-S01/2(B)模拟时，当 GPCS 选择 Output 到 PA0 输出时，PA3 输出功能也会受影响
- 仿真 PWM 波形时，建议用户在程序运行期间查看波形，当仿真器暂停或单步运行时波形可能会与实际不符
- PDK5S-I-S01/2(B)仿真器的 ILRC 频率与实际 IC 不同，且未经校准，其频率范围大约在 34K~38KHz。
- 模拟时的快速唤醒时间和 IC 不一样；仿真器：128 个系统时钟； IC：45 ILRC 时钟
- 看门狗溢出时间仿真器和 IC 不一样

看门狗溢出设置	5S-I-S01/2(B)	PMS164
misc[1:0]=00	2048 * T_{ILRC}	8192 * T_{ILRC}
misc[1:0]=01	4096 * T_{ILRC}	16384 * T_{ILRC}
misc[1:0]=10	16384 * T_{ILRC}	65536 * T_{ILRC}
misc[1:0]=11	256 * T_{ILRC}	262144 * T_{ILRC}