PMS154C 系列

8bit OTP IO 型单片机

数据手册

第 1.06 版

2023 年 6 月 29 日

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

TEL: 886-3-572-8688 www.padauk.com.tw

重要声明

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

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

应广科技為服务客户所提供之任何编程软件，皆为服务与参考性质，不具备任何软件漏洞责任，应广科技不承担任何责任来自于因客户的产品设计所造成的任何损失。在应广科技所保障的规格范围内，客户应设计和验证他们的产品。为了尽量减少风险，客户设计产品时，应保留适当的产品工作范围安全保障。

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

目　录

修订历史

修订	日期	描述
1.05	2020/06/09	1. PRST#改为 PRSTB 2. 修改第 3 章、第 8 章、表 6、表 7 3. 修改第 4.1 节：LVR%、t_{SBP}、t_{WUP} 4. 修改第 5.8.1 节、第 5.8.3 节、第 5.13 节、第 5.13.1 节、第 5.13.3 节、第 5.13.3 节、第 5.14 节、第 5.14.3 节、第 6.9 节、第 6.10 节、第 6.28 节、第 6.34 节、第 6.36 节、第 6.38 节、第 6.40 节、第 6.42 节、第 6.44 节、第 9.3 节 5. 新增第 5.4.6 节、第 5.14.4 节、图 19、表 8
1.06	2023/06/29	1. 更新”重要声明” 2. 更新 I_{OH} 特性（第 4.1 节） 3. 修改 5.8.1、5.10.1、5.13、6.7、6.9、6.26、6.27、9.1.1、9.1.4、9.1.7、9.2 4. 修改图 3、表 5、表 6、图 14、图 17 5. 其他已知细节错误修正

使用警告

在使用 IC 前，请务必认真阅读 PMS154C 相关的 APN（应用注意事项）。
请至官网下载查看与之关联的最新 APN 资讯。

http://www.padauk.com.tw/cn/product/show.aspx?num=15&kw=PMS154C

（以下附图仅供参考。）

PMS154B 和 PMS154C 主要差异表

项目	功能	PMS154B	PMS154C
1	工作电压范围	2.2V~5.5V	1.8V~5.5V
2	11-bit PWM	一组： PWMG0	三组： PWMG0, PWMG1 & PWMG2
3	Comparator_Edge	未开放	Code Option

1. 单片机特点

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. 系统功能

- 2KW OTP 程序存储器
- 128 字节数据存储器
- 一个硬件 16 位定时器
- 两个 8 位定时器（可作为 PWM 生成器）
- 三个 11 位硬件 PWM 生成器(PWMG0，PWMG1 & PWMG2)
- 提供一个硬件比较器
- 14 个 IO 引脚，有可选的上拉电阻
- 3 组不同的驱动电流 IO，可应对不同的应用需求
- 可选择的 IO 驱动能力（普通或低选项）
- 每个 IO 引脚都可设定为唤醒功能
- 内建 1/2 V_{DD} LCD 偏置电压生成器，可支持最大 4X10 点阵的 LCD 屏
- 时钟模式：内部高频振荡器(IHRC)，内部低频振荡器(ILRC)，外部晶体震荡(EOSC)
- 每个能唤醒的 IO：支持两种可选的唤醒速度：正常和快速
- 8 段 LVR 复位设定：4.0V, 3.5V, 3.0V, 2.75V, 2.5V, 2.2V, 2.0V, 1.8V
- 两个外部中断输入引脚

1.3. CPU 特点

- 工作模式：单一处理单元的工作模式
- 86 个强大指令
- 绝大部分指令都是单周期 (1T)指令
- 可程序设定的堆栈指针和堆栈深度
- 数据存取支持直接和间接寻址模式，用数据存储器即可當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

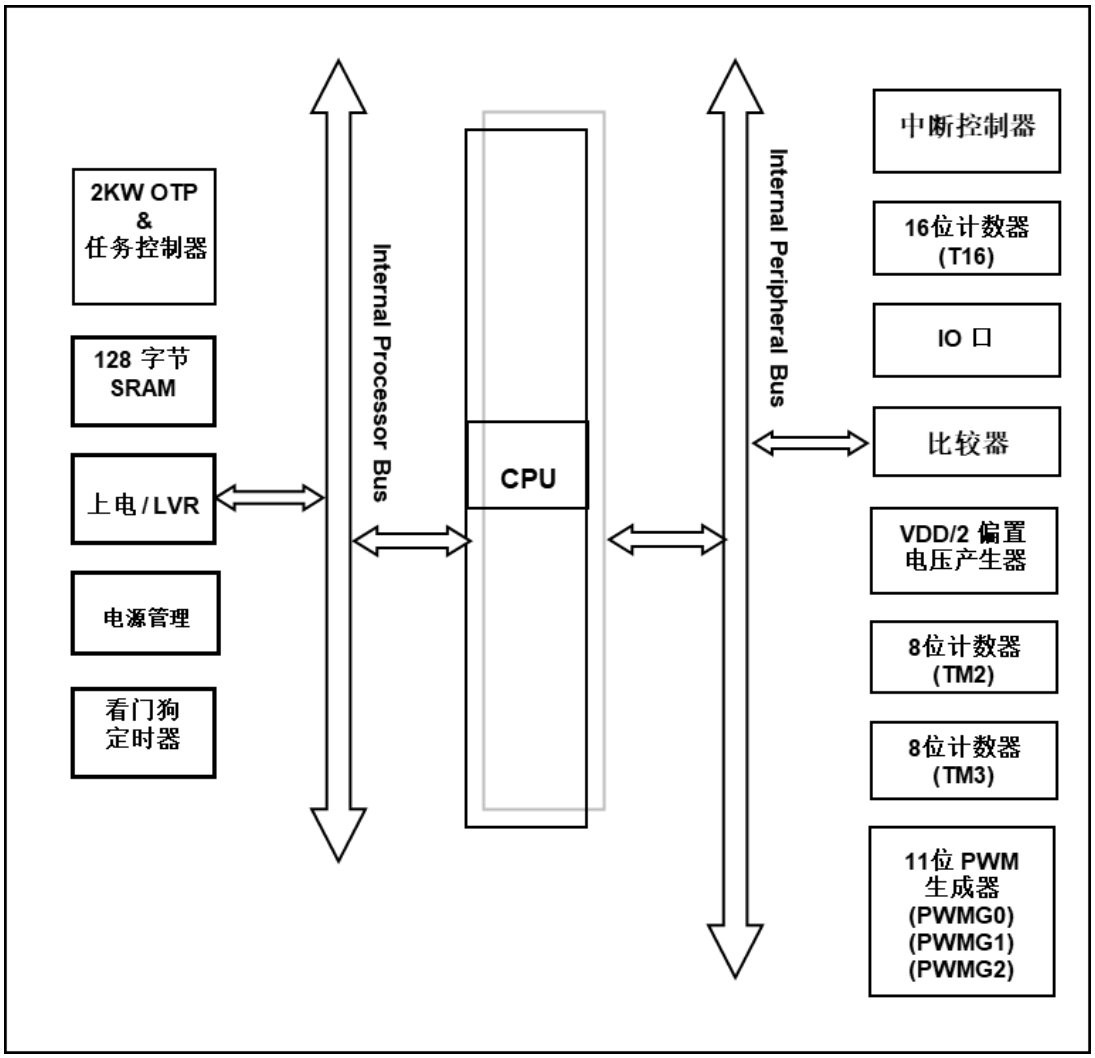
1.4. 订购/封装信息

- PMS154C-U06: SOT23-6 (60mil)
- PMS154C-M10: MSOP10 (118mil)
- PMS154C-S16: SOP16 (150mil)
- PMS154C-1J16A: QFN3*3-16pin (0.5pitch)
- PMS154C-S08: SOP8 (150mil)
- PMS154C-S14: SOP14 (150mil)
- PMS154C-D16: DIP16 (300mil)

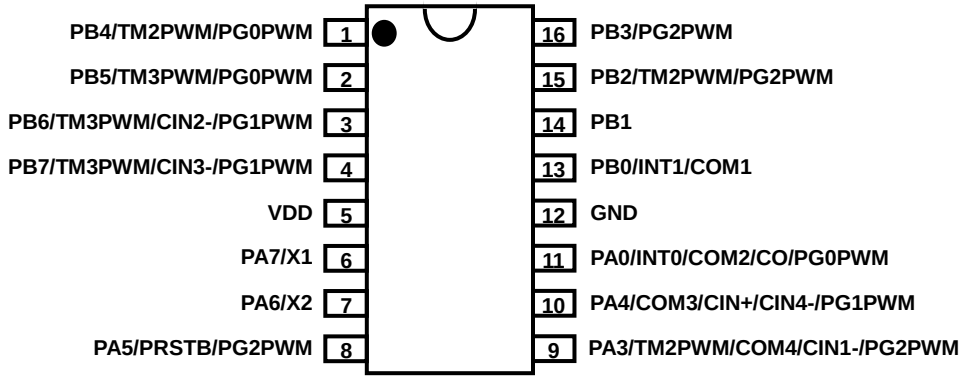
- 有关封装尺寸的信息，请参阅官方网站文件：“封装信息”

2. 系统概述和方框图

PMS154C 是一个 IO 类型，静态 OTP 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，只有少部分间接地址访问的指令是需要两个指令周期。 PMS154C 内置 2KW OTP 程序存储器以及 128 字节数据存储器；另外，PMS154C 提供一个 16 位的硬件计数器，还有两个 8 位计数器（Timer2、Timer3）和三个 11 位计数器（PWMG0、PWMG1 及 PWMG2）都能产生 PWM，另外 PMS154C 还提供一个硬件比较器和驱动 LCD 的 $V_{DD}/2$ 偏置电压生成器。

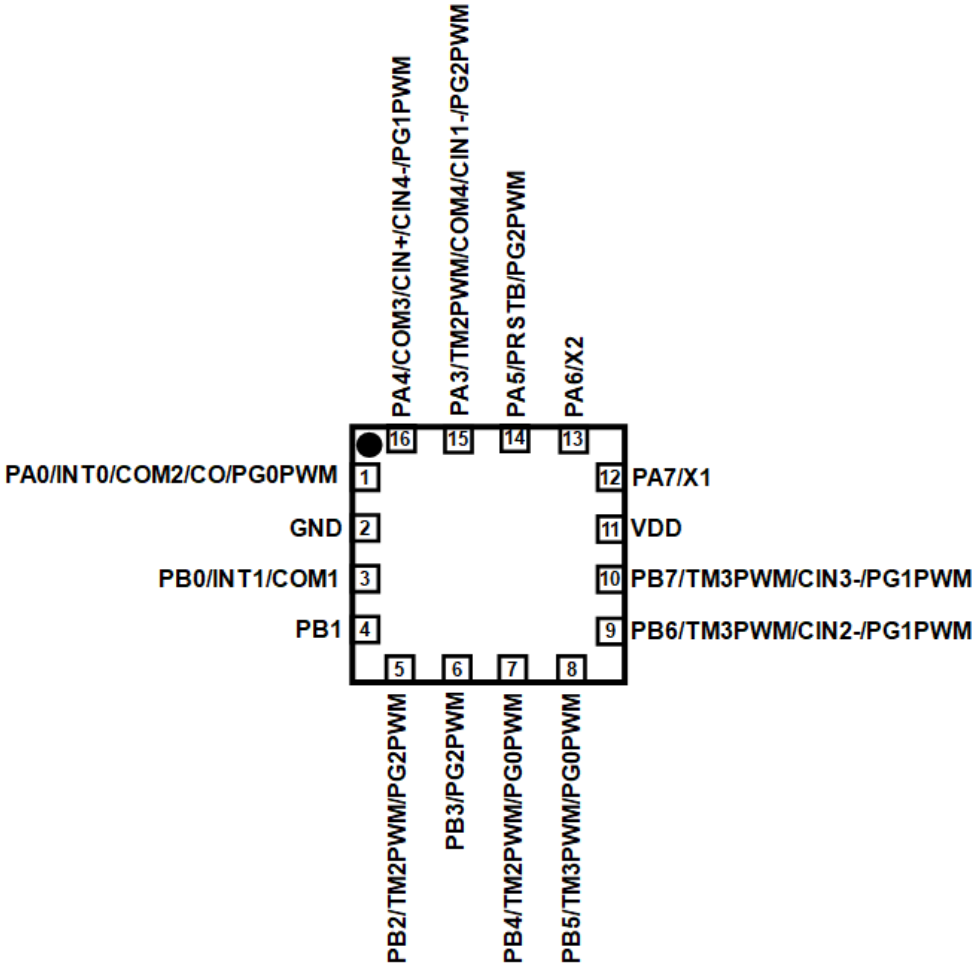


3. 引脚功能说明

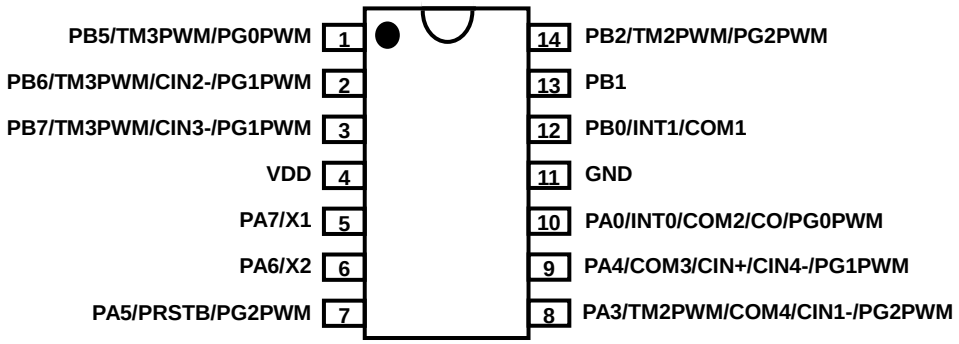


PMS154C-S16:SOP16 (150mil)

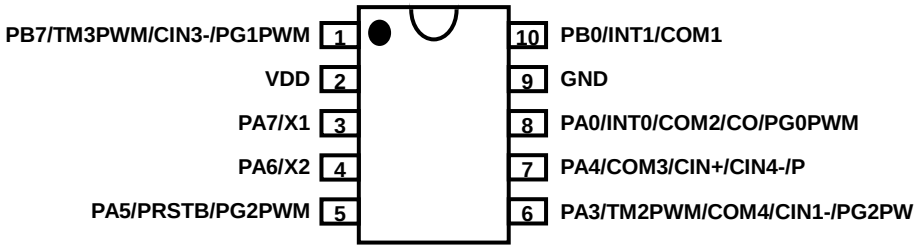
PMS154C-D16:DIP16 (300mil)



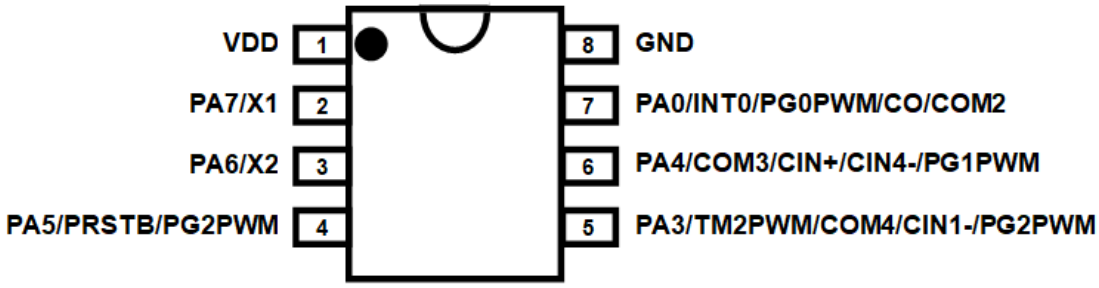
PMS154C-1J16A: QFN3*3-16pin (0.5pitch)



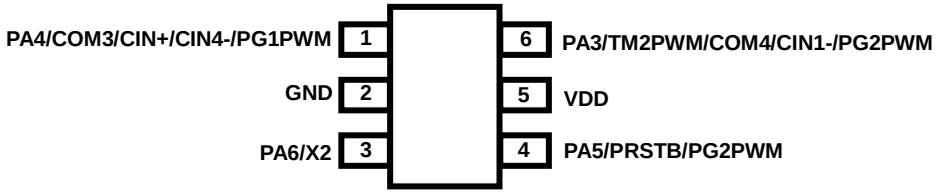
PMS154C-S14:SOP14 (150mil)



PMS154C-M10: MSOP10 (118mil)



PMS154C-S08: SOP8 (150mil)



PMS154C-U06: SOT23-6 (60mil)

引脚名称	引脚&缓冲器类型	功能描述
PA7 / X1	IO ST / CMOS	此引脚可当： (1) 端口 A 位 7，并可编程设定为输入或输出，弱上拉电阻模式。 (2) 当使用外部晶体振荡器时，做为 X1 引脚。 当用做晶体振荡器的功能时，为减少漏电流，请用 padier 寄存器位 7 关闭其数字输入功能，这个引脚可以设定在睡眠中唤醒系统的功能；但当寄存器 *padier* 位 7 为”0”时，唤醒功能是被关闭的。
PA6 / X2	IO ST / CMOS	此引脚可当： (1) 端口 A 位 6，并可编程设定为输入或输出，弱上拉电阻模式。 (2) 当使用外部晶体振荡器时，做为 X2 引脚。 当用做晶体振荡器的功能时，为减少漏电流，请用 padier 寄存器位 6 关闭其数字输入功能，这个引脚可以设定在睡眠中唤醒系统的功能；但当寄存器 *padier* 位 6 为”0”时，唤醒功能是被关闭的。
PA5 / PRSTB / PG2PWM	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或开漏输出(open drain) ，弱上拉电阻模式。 (3) 11 位计数器 PWMG2 的输出。（仿真器不支持） 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *padier* 位 5 为”0”时，唤醒功能是被关闭的。 另外，当此引脚设定成输入时，对于需要高抗干扰能力的系统，请串接 33Ω 电阻。
PA4 / CIN+ / COM3 / CIN4- / PG1PWM	IO ST / CMOS	此引脚可用做： (1) 端口 A 位 4，并可编程设定为输入或输出，弱上拉电阻模式。 (2) COM3 口，提供 1/2 V_{DD} 驱动 LCD 显示。 (3) 比较器的正输入源。 (4) 比较器的第 4 负输入源。 (5) 11 位计数器 PWMG1 的输出。 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。

引脚名称	引脚&缓冲器类型	功能描述
PA3 / TM2PWM / COM4 / CIN1- / PG2PWM	IO ST / CMOS	此引脚可用做： (1) 端口 A 位 3，并可编程设定为输入或输出，弱上拉电阻模式。 (2) 8 位计数器 Timer2 的输出。 (3) 比较器的第 1 负输入源。 (4) COM4 口，提供 1/2 V_{DD} 驱动 LCD 显示。 (5) 11 位计数器 PWMG2 的输出。 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 / PG0PWM / CO / COM2	IO ST / CMOS	此引脚可用做： (1) 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) 外部中断源 0，上升沿和下降沿都可触发中断。 (3) 比较器的输出。 (4) 11 位计数器 PWMG0 的输出。 (5) COM2 口，提供 1/2 V_{DD} 驱动 LCD 显示。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *padier* 位 0 为”0”时，唤醒功能是被关闭的。
PB7 / TM3PWM / CIN3- / PG1PWM	IO ST / CMOS	此引脚可用做： (1) 端口 B 位 7，并可编程设定为输入或输出，弱上拉电阻模式。 (2) 比较器的第 3 负输入源。 (3) 8 位计数器 Timer3 的输出。 (4) 11 位计数器 PWMG1 的输出。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *pbdier* 位 7 为”0”时，唤醒功能是被关闭的。
PB6 / TM3PWM / CIN2- / PG1PWM	IO ST / CMOS	此引脚可用做： (1) 端口 B 位 6，并可编程设定为输入或输出，弱上拉电阻模式。 (2) 比较器的第 2 负输入源。 (3) 8 位计数器 Timer3 的输出。 (4) 11 位计数器 PWMG1 的输出。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *pbdier* 位 6 为”0”时，唤醒功能是被关闭的。

引脚名称	引脚&缓冲器类型	功能描述
PB5 / TM3PWM / PG0PWM	IO ST / CMOS	此引脚可用做： (1) 端口 B 位 5，并可编程设定为输入或输出，弱上拉电阻模式。 (2) 11 位计数器 PWMG0 的输出。 (3) 8 位计数器 Timer3 的输出。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *pbdier* 位 5 为”0”时，唤醒功能是被关闭的。
PB4 / TM2PWM / PG0PWM	IO ST / CMOS	此引脚可用做： (1) 端口 B 位 4，并可编程设定为输入或输出，弱上拉电阻模式。 (2) 11 位计数器 PWMG0 的输出。 (3) 8 位计数器 Timer2 的输出。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *pbdier* 位 4 为”0”时，唤醒功能是被关闭的。
PB3 / PG2PWM	IO ST / CMOS	此引脚可用做： (1) 端口 B 位 3，并可编程设定为输入或输出，弱上拉电阻模式。 (2) 11 位计数器 PWMG2 的输出。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *pbdier* 位 3 为”0”时，唤醒功能是被关闭的。
PB2 / TM2PWM / PG2PWM	IO ST / CMOS	此引脚可用做： (1) 端口 B 位 2，并可编程设定为输入或输出，弱上拉电阻模式。 (2) 8 位计数器 Timer2 的输出。 (3) 11 位计数器 PWMG2 的输出。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *pbdier* 位 2 为”0”时，唤醒功能是被关闭的。
PB1	IO ST / CMOS	此引脚可用做端口 B 位 1，并可编程设定为输入或输出，弱上拉电阻模式。这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *pbdier* 位 1 为”0”时，唤醒功能是被关闭的。
PB0 / INT1 / COM1	IO ST / CMOS	此引脚可用做： (1) 当端口 B 位 0，并可编程设定为输入或输出，弱上拉电阻模式。 (2) 外部引脚中断 1，中断服务可发生在上升沿或下降沿。 (3) COM1 口，提供 1/2 V_{DD} 驱动 LCD 显示。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 *pbdier* 位 0 为”0”时，唤醒功能是被关闭的。
VDD		正电源
GND		地
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列明外，皆于 Ta = -20ºC ~ 70 ºC, V_{DD}=3.3V, f_{SYS}=2MHz 之条件下获得。

符 号	特 性	最小值	典型值	最大值	单位	条 件
V_{DD}	工作电压	1.8#		5.5	V	#受限于 LVR 公差
LVR%	低电压复位公差	-5		5	%	
f_{SYS}	系统时钟*= IHRC/2 IHRC/4 IHRC/8 ILRC	 0 0 0	 70K	 8M 4M 2M	Hz	 V_{DD}≧ 3.5V V_{DD}≧ 2.5V V_{DD}≧ 1.8V V_{DD} = 3V
V_{POR}	上电复位电压	1.9	2.0	2.1	V	
I_{OP}	工作电流		0.3 12 10		mA uA uA	f_{SYS}=IHRC/16=1MIPS@3V f_{SYS}=ILRC=70KHz@3V f_{SYS}=EOSC=32KHz@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，关闭 IHRC）		5		uA	V_{DD}=3.3V
V_{IL}	输入低电压	0 0		0.2 V_{DD} 0.1 V_{DD}	V	PA5 其他 IO 口
V_{IH}	输入高电压	0.7 V_{DD}		V_{DD}	V	
I_{OL}	IO 输出灌电流（正常, normal）					
	*PA0,PA3,PA4,PB2,PB5,PB6		10		mA	V_{DD}=3.3V, V_{OL}=0.33V
	*PA6,PA7,PB0,PB1,PB3,PB4,PB7		6			
	*PA5		5			
	IO 输出灌电流（低, low）					
	*PA5		5		mA	V_{DD}=3.3V, V_{OL}=0.33V
	*其他 IO		2			
I_{OH}	PA5 其他 IO 输出驱动电流（正常, normal）		0 -5		mA	V_{DD}=3.3V, V_{OH}=2.97V
	PA5 其他 IO 输出驱动电流（低, low）		0 -1.6			
V_{IN}	输入电压	-0.3		V_{DD}+0.3	V	
$I_{INJ\ (PIN)}$	脚位的引入电流			1	mA	V_{DD}+0.3 ≧ V_{IN} ≧ -0.3
R_{PH}	上拉电阻		200		KΩ	V_{DD}=3.3V
f_{IHRC}	IHRC 输出频率（校准后）*	15.84*	16*	16.16*	MHz	V_{DD}=5V, Ta=25ºC
		15.20*		16.80*		V_{DD} =2.0V~5.5V, -20ºC <Ta<70ºC*
		13.60*		18.40*		V_{DD} =1.8V~5.5V, -20ºC <Ta<70ºC*

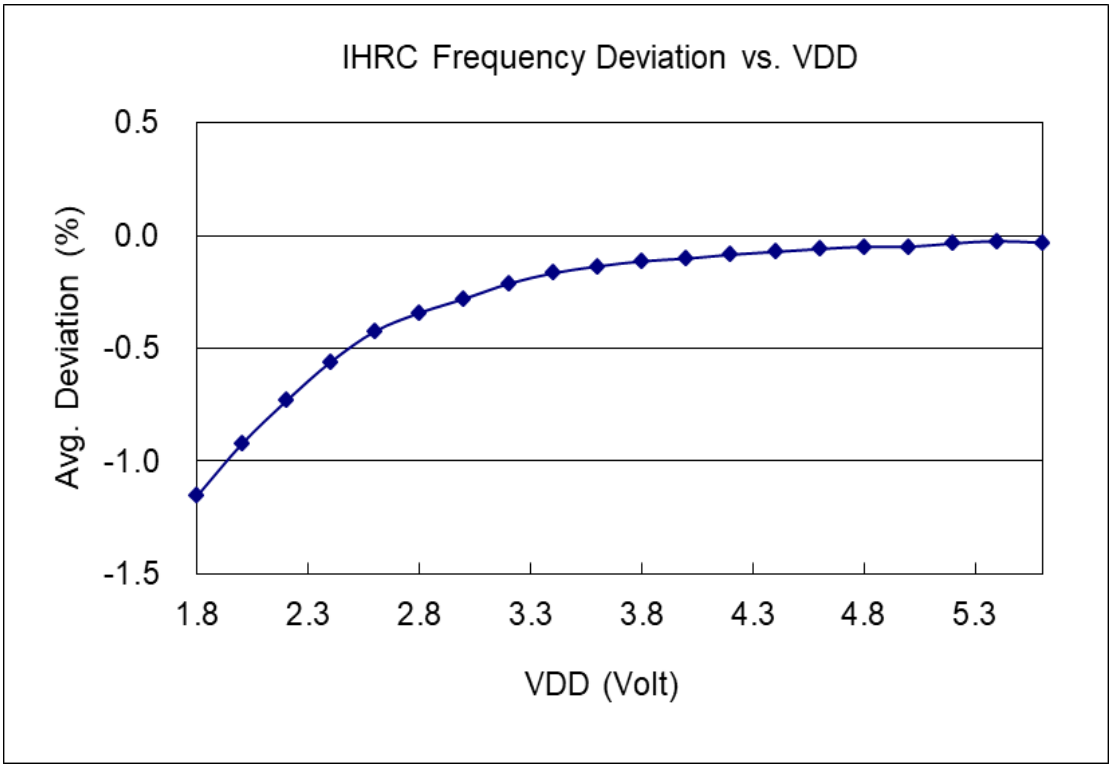
符 号	特 性	最小值	典型值	最大值	单位	条 件
t_{INT}	中断脉冲宽度	30			ns	V_{DD}=3.3V
V_{DR}	数据存储器数据保存电压*	1.5			V	掉电模式下
t_{WDT}	看门狗超时溢出时间		8192		T_{ILRC}	misc[1:0]=00（默认）
			16384			misc[1:0]=01
			65536			misc[1:0]=10
			262144			misc[1:0]=11
t_{SBP}	系统上电开机时间（慢开机）		47		ms	@V_{DD}=5V
	系统上电开机时间（快开机）		780		us	
t_{WUP}	快速唤醒时间（misc.5=1）或者快速开机		45		T_{ILRC}	T_{IHRC} 是 IHRC 时钟周期
	正常唤醒时间（misc.5=0）或者慢开机		3000			
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		V_{DD}-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。

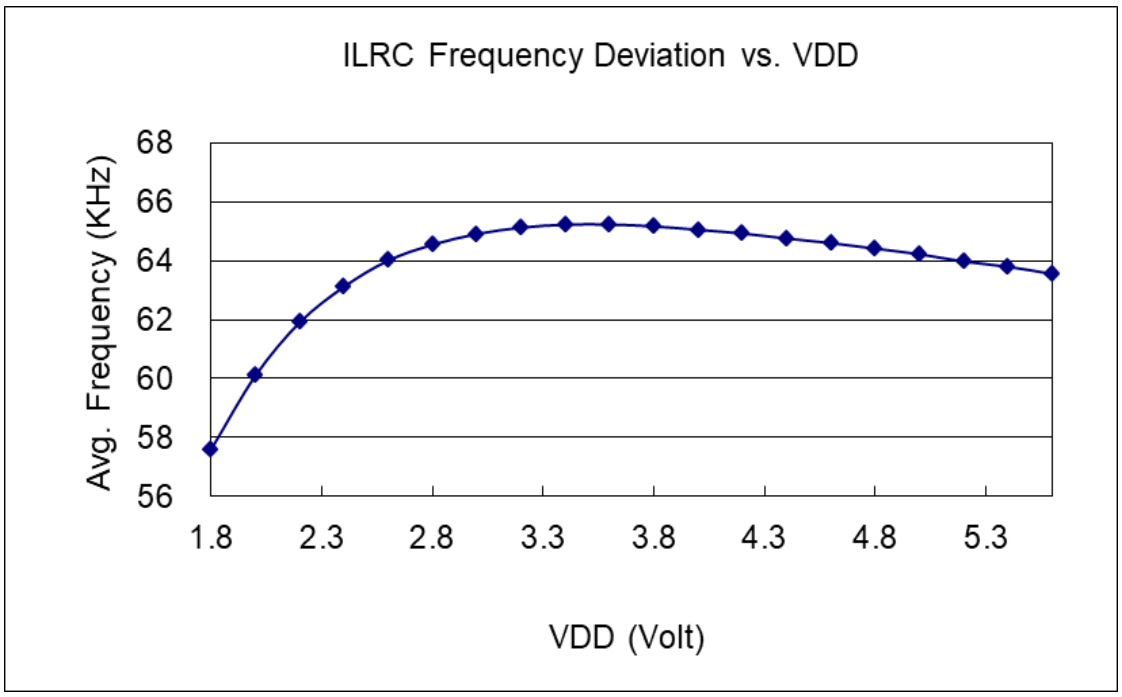
4.2. 绝对最大值

- 电源电压 ..... 1.8V ~ 5.5V
  * 最大电压不能超过 5.5V，否则可能损坏 IC。
- 输入电压 ..... -0.3V ~ V_{DD} + 0.3V
- 工作温度 ..... -20°C ~ 70°C
- 储藏温度 ..... -50°C ~ 125°C
- 节点温度 ..... 150°C

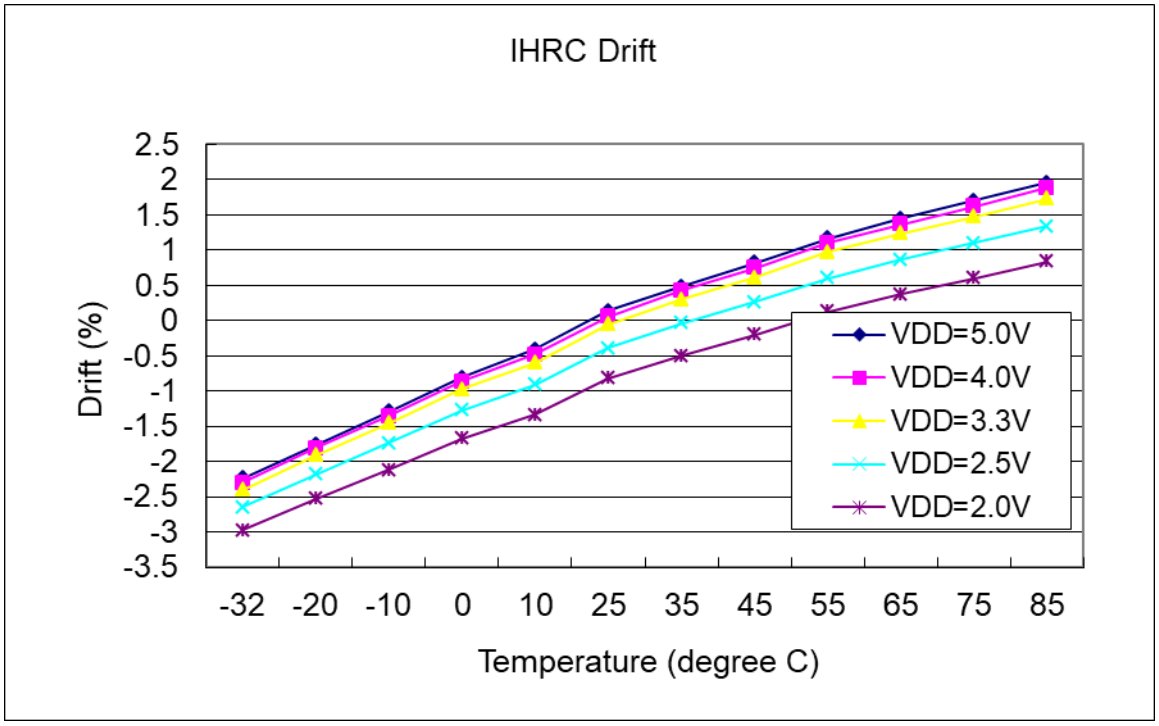
4.3. IHRC 频率与 VDD 关系曲线图（校准到 16MHz）



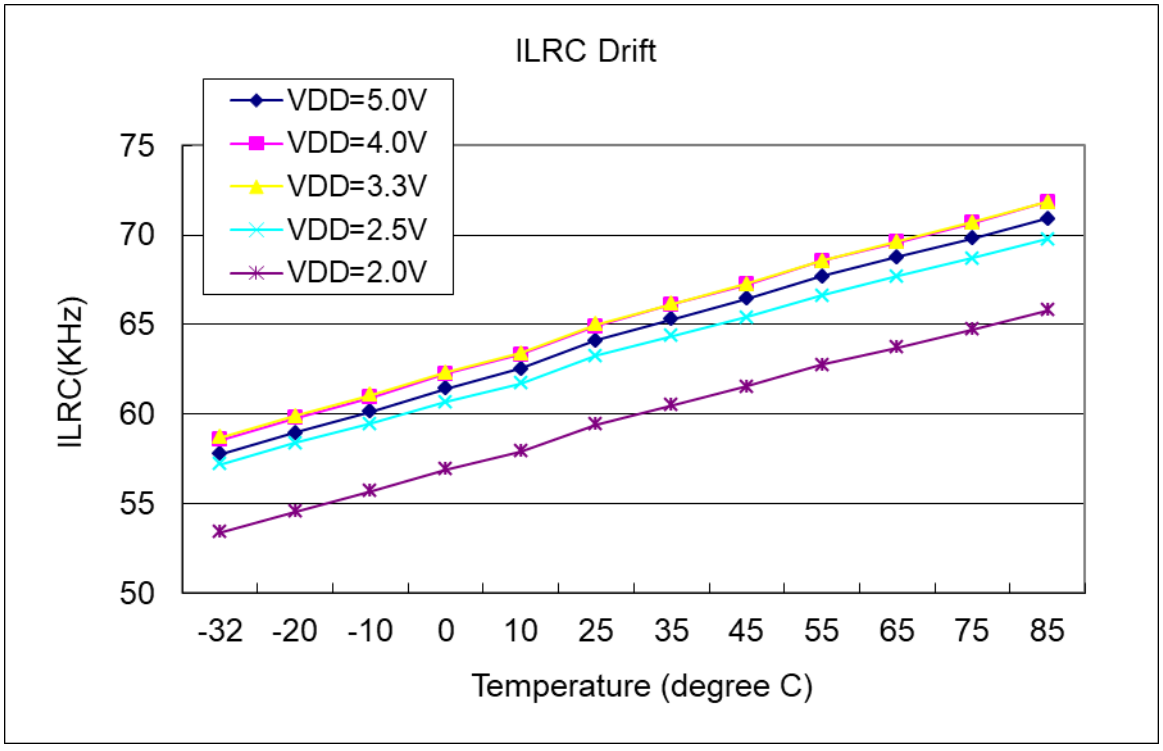
4.4. ILRC 频率与 VDD 关系曲线图



4.5. IHRC 频率与温度关系曲线图（校准到 16MHz）



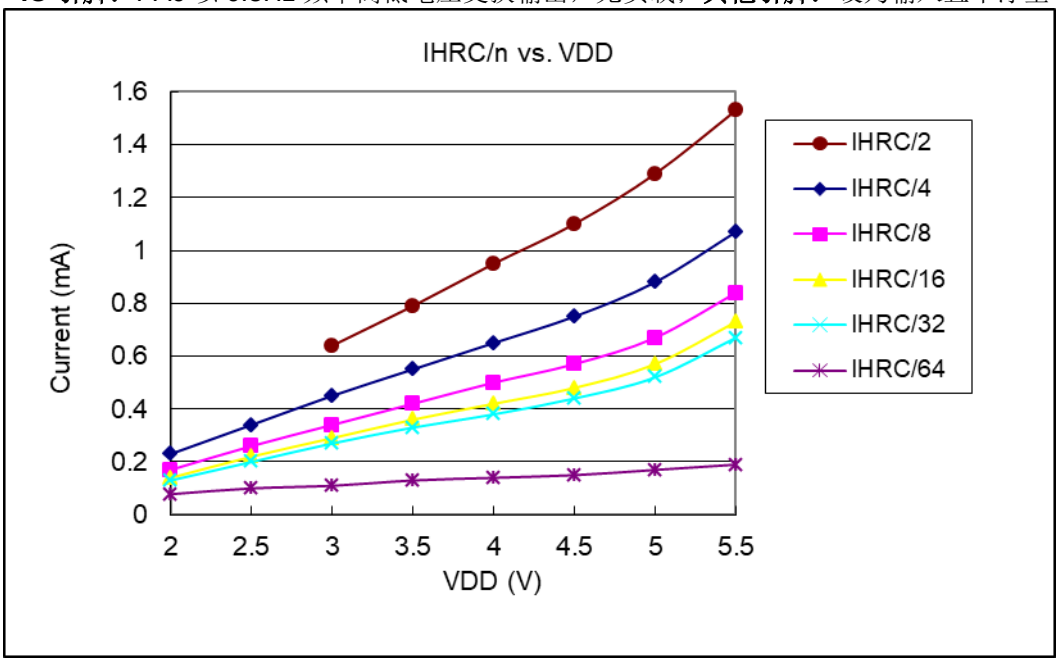
4.6. ILRC 频率与温度关系曲线图



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**：IHRC；**关闭的硬件模块**：ILRC，Bandgap，LVR，T16

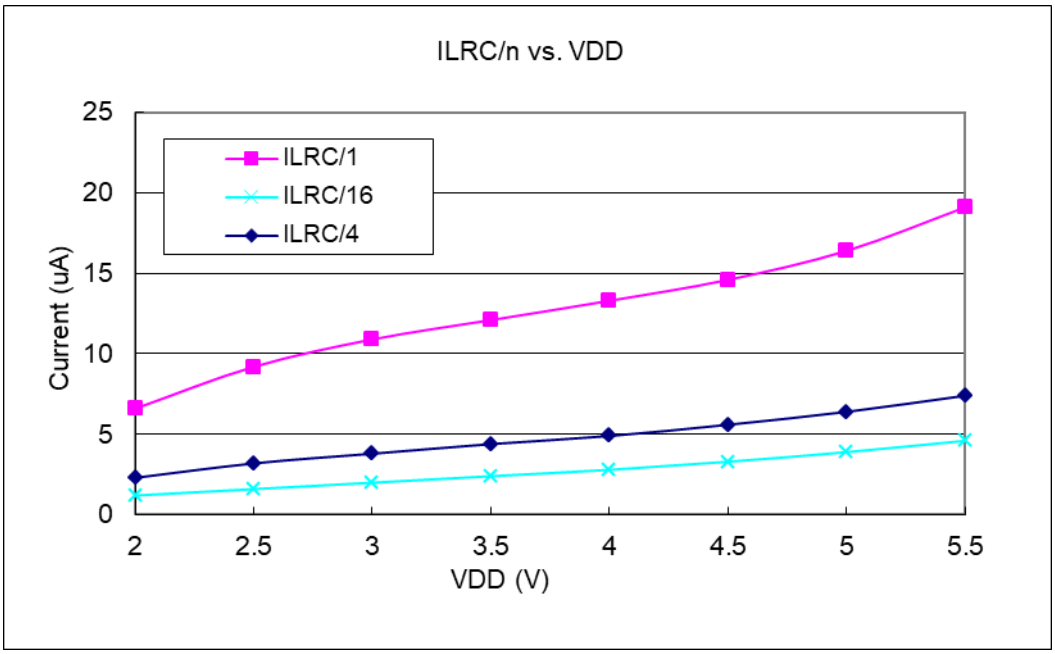
IO 引脚：PA0 以 0.5Hz 频率高低电压交换输出，无负载；**其他引脚**：设为输入且不浮空



4.8. 工作电流与 VDD、系统时钟 CLK=ILRC/n 曲线图

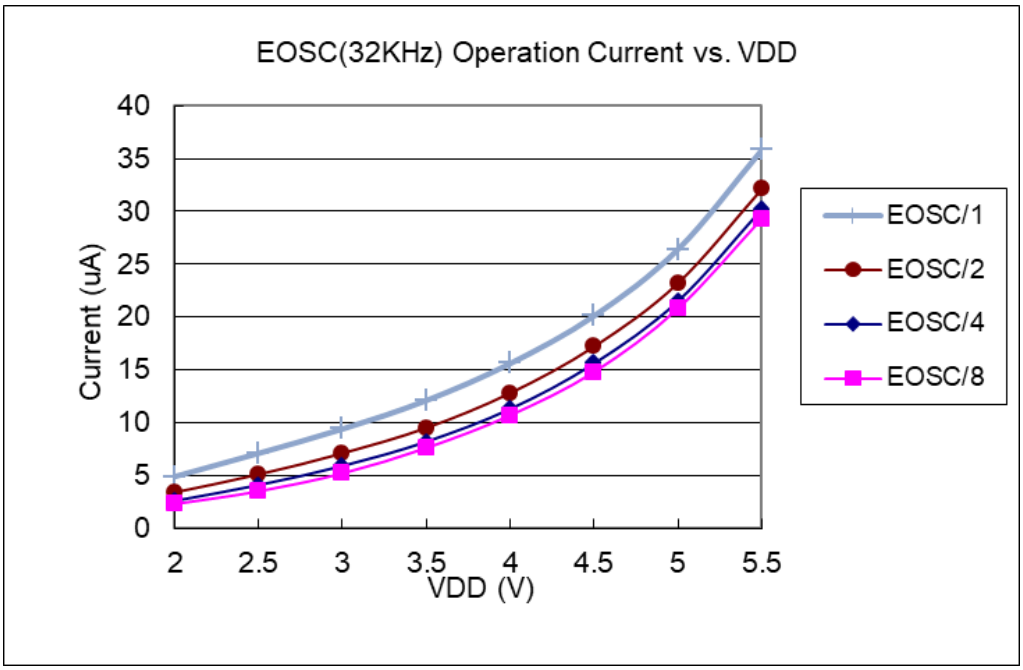
条件：**开启的硬件模块**：ILRC；**关闭的硬件模块**：IHRC，T16，Bandgap，LVR；

IO 引脚：PA0 以 0.5Hz 频率高低电压交换输出，无负载；**其他引脚**：设为输入且不浮空



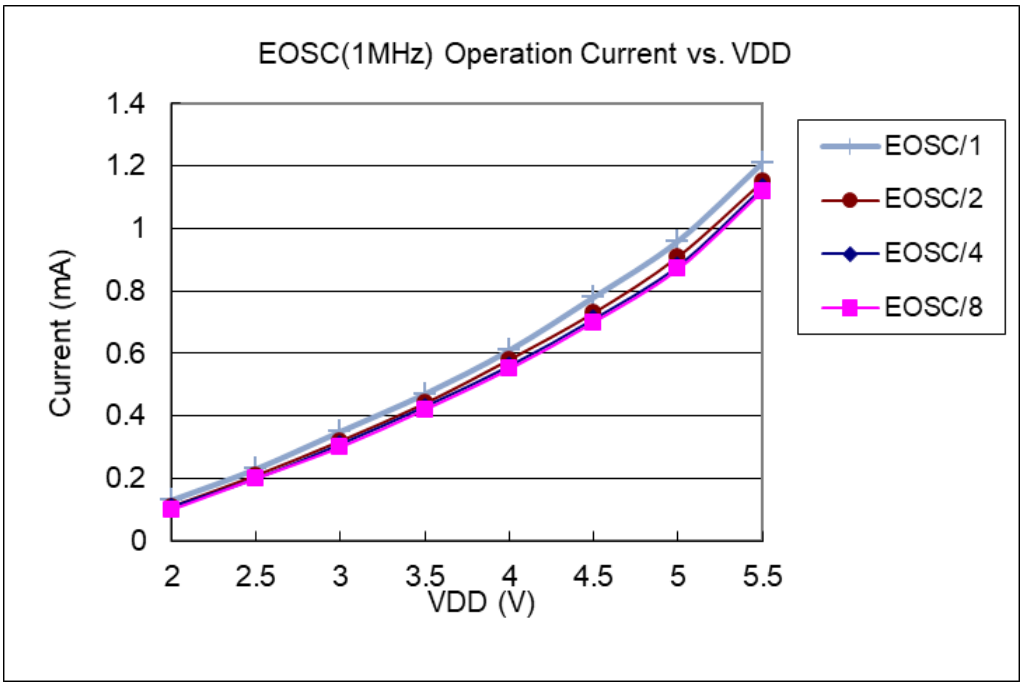
4.9. 工作电流与 VDD、系统时钟 CLK=32KHz EOSC/n 曲线图

条件：**开启的硬件模块**：EOSC；**关闭的硬件模块**：IHRC，T16，Bandgap，LVR，ILRC；
IO 引脚：PA0 以 0.5Hz 频率高低电压交换输出，无负载；**其他引脚**：设为输入且不浮空



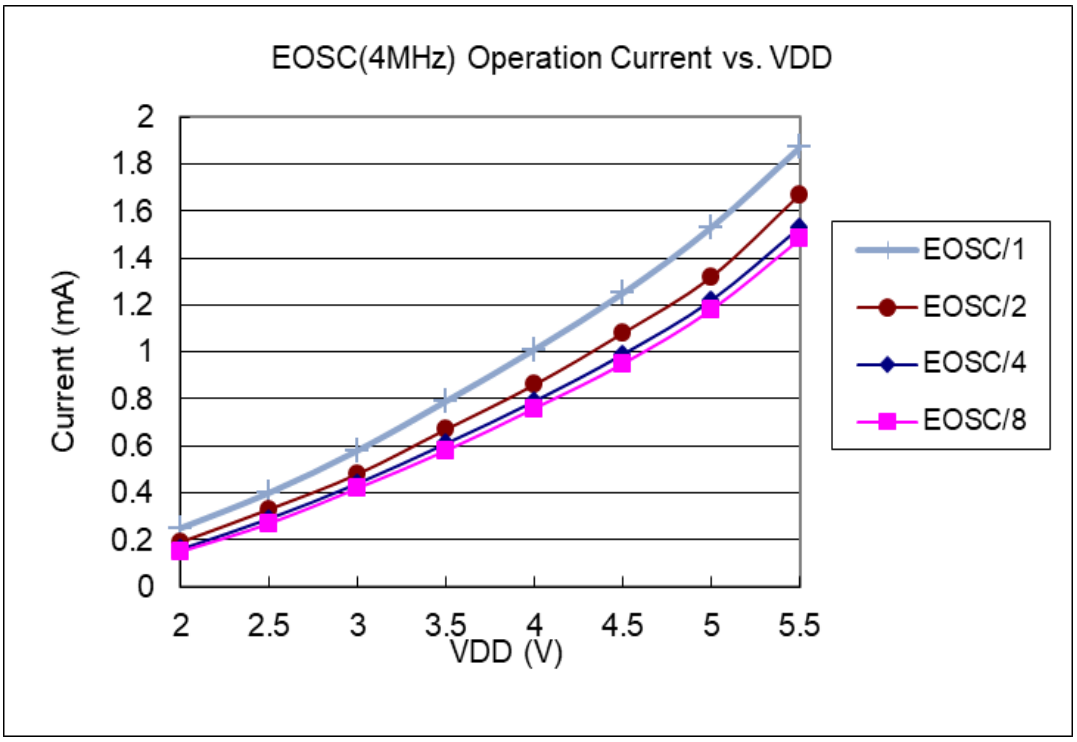
4.10. 工作电流与 VDD、系统时钟 CLK=1MHz EOSC/n 曲线图

条件：**开启的硬件模块**：EOSC；**关闭的硬件模块**：IHRC，T16，Bandgap，LVR，ILRC；
IO 引脚：PA0 以 0.5Hz 频率高低电压交换输出，无负载；**其他引脚**：设为输入且不浮空

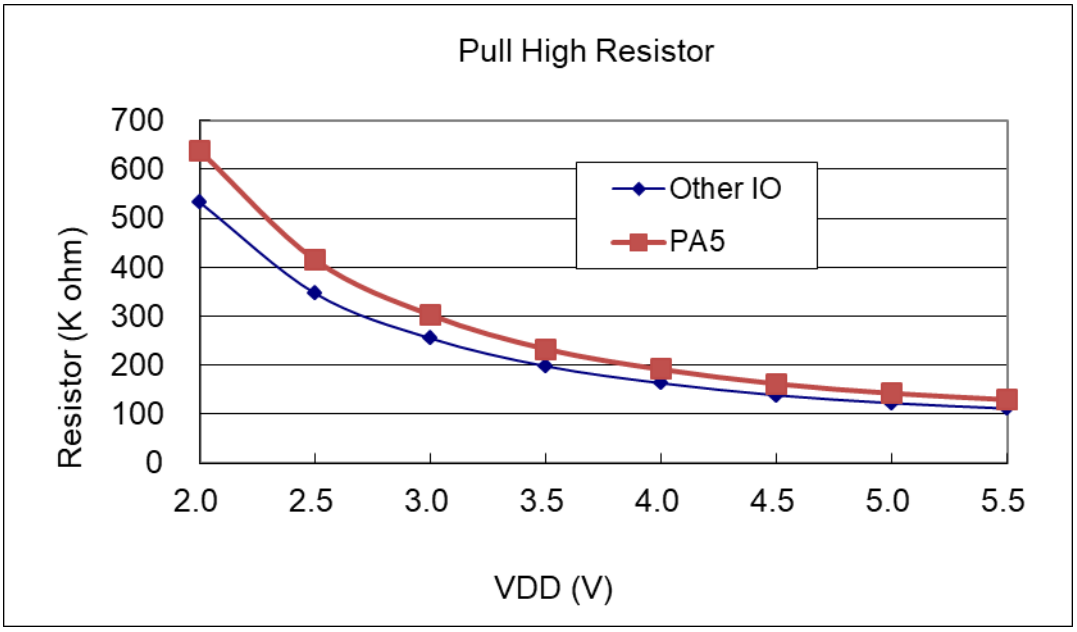


4.11. 工作电流与 VDD、系统时钟 CLK=4MHz EOSC/n 曲线图

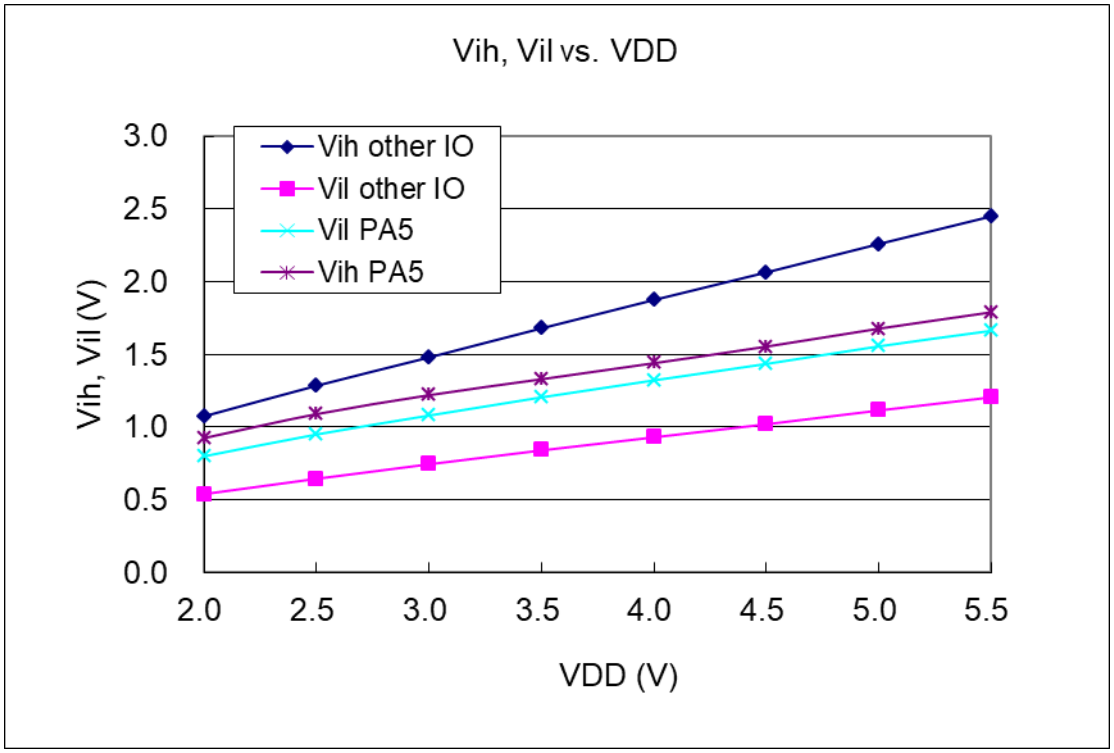
条件：**开启的硬件模块**：EOSC；**关闭的硬件模块**：IHRC，T16，Bandgap，LVR，ILRC；
IO 引脚：PA0 以 0.5Hz 频率高低电压交换输出，无负载；**其他引脚**：设为输入且不浮空



4.12. 引脚上拉电阻曲线图

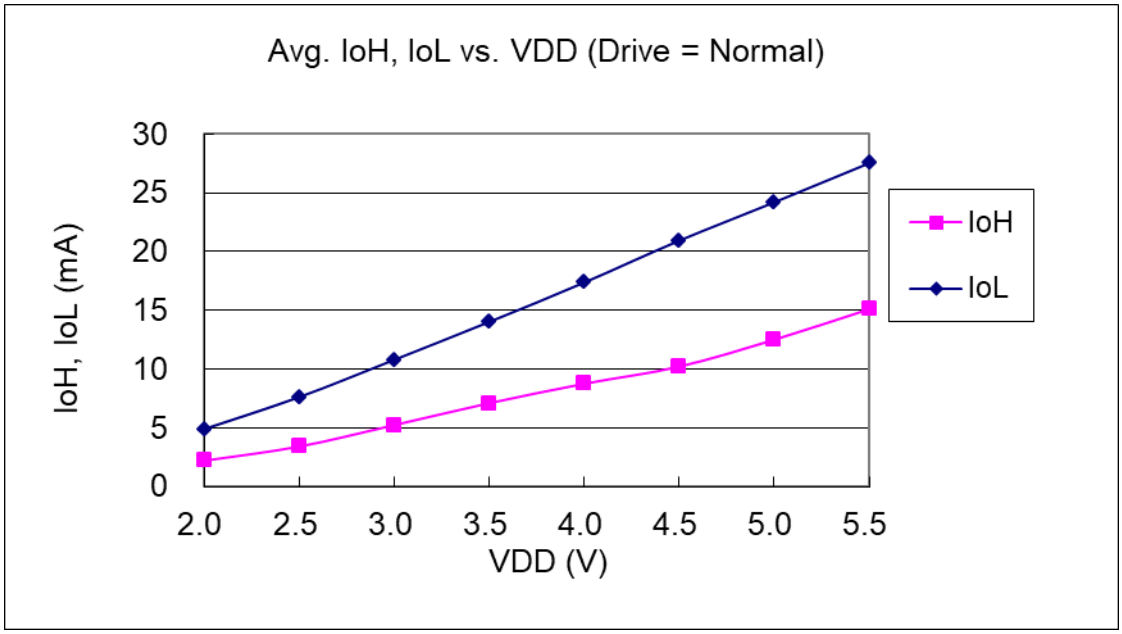


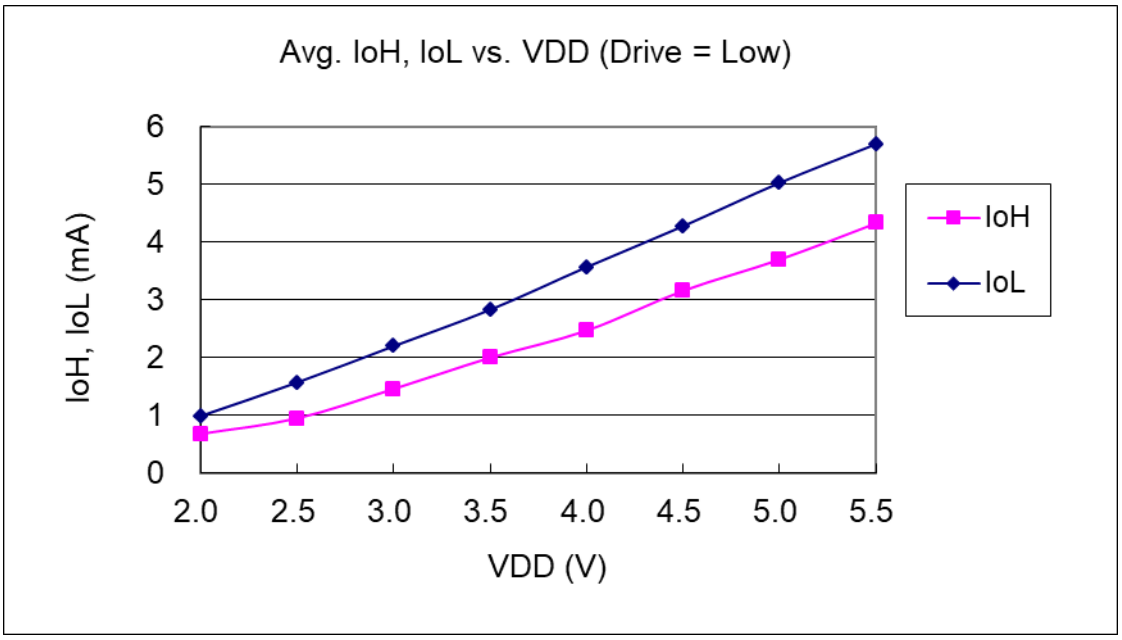
4.13. 引脚输入高电压与低电压(V_{IH} / V_{IL}) 曲线图



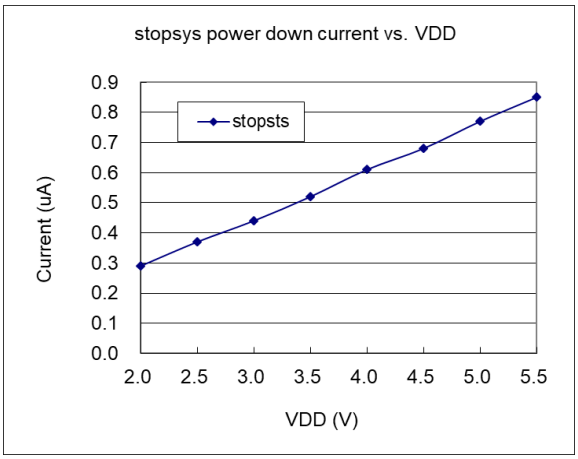
4.14. 引脚输出驱电流(Ioh)与灌电流(Iol) 曲线图

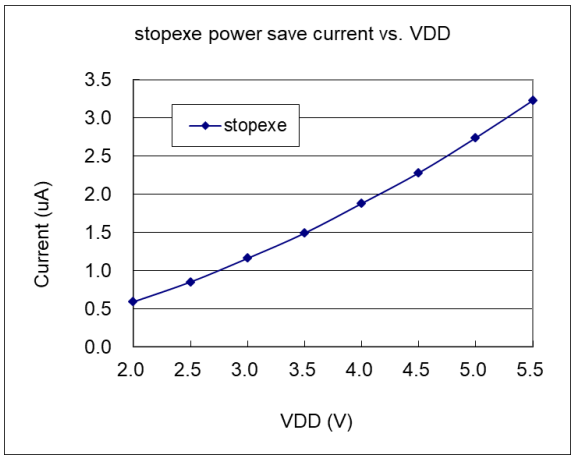
(VOH=0.9*VDD, VOL=0.1*VDD)





4.15. 掉电模式消耗电流电流(I_{PD})与省电模式消耗电流(I_{PS}) 曲线图





5. 功能概述

5.1. OTP 程序存储器

OTP（一次性可编程）程序存储器用来存放要执行的程序指令。OTP 程序存储器可以储存数据，包含：数据，表格和中断入口。复位之后，FPP0 的初始地址 0x000 为系统保留，所以程序从 0x001 开始（通常用 GOTO FPPA0），中断入口是 0X010；OTP 程序存储器最后 16 个地址空间是被保留给系统使用，如：校验，序列号等。PMS154C 的 OTP 程序存储器容量为 2KW，如表 1 所示。OTP 存储器从地址“0x7E8 to 0x7FF”供系统使用，从“0x002 ~ 0x00F”和“0x011~0x7E7”地址空间是用户的程序空间。

地址	功能
0x000	系统使用
0x001	*GOTO* 指令
0x002	用户程序区
•	•
•	•
0x00F	用户程序区
0x010	中断入口地址
0x011	用户程序区
•	•
0x7E7	用户程序区
0x7E8	系统使用
•	•
0x7FF	系统使用

表 1：PMS154C 程序存储器结构

5.2. 开机流程

开机时，POR（上电复位）是用于复位 PMS154C；开机时间可选快开机或者普通模式。快速开机的时间是 45 个 ILRC 时钟周期，正常开机的开机时间是 3000 个 ILRC 时钟周期。不管哪种开机模式，用户必须确保上电后电源电压稳定，开机时间 t_{SBP}，如图 1 所示。

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

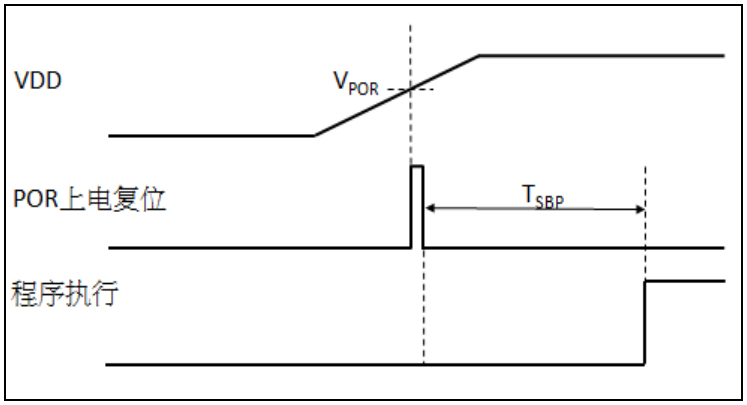
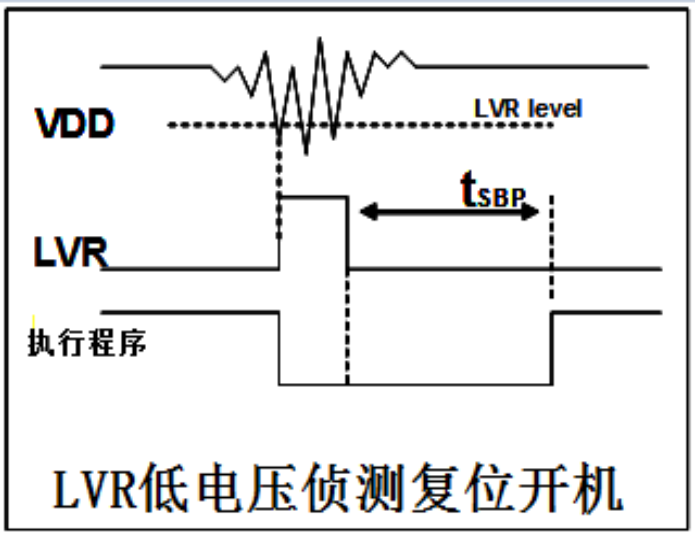
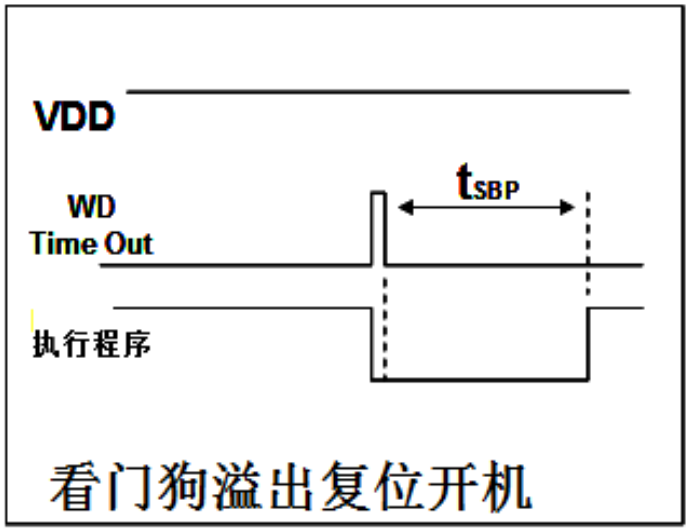
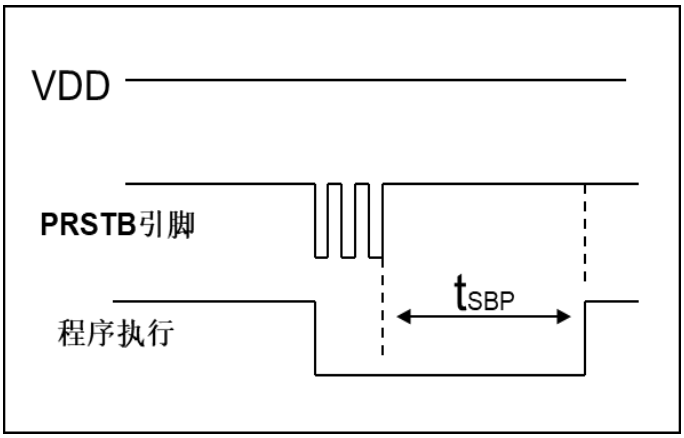


图 1：上电复位时序

5.2.1.复位时序图


VDD
LVR level
tSBP
LVR
执行程序
LVR低电压侦测复位开机


VDD
WD
Time Out
tSBP
执行程序
看门狗溢出复位开机


VDD
PRSTB引脚
tSBP
程序执行

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数据指针，以及堆栈存储器。

堆栈存储器是定义在数据储存器里。堆栈存储器的堆栈指针是定义在堆栈指针寄存器；堆栈存储器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存储器的大小，以保持最大的弹性。

数据存储器的间接存取方式，是以数据存储器当作数据指针来存取数据字节。所有的数据存储器，都可以拿来当作数据指针，这可以让单片机的资源利用率最大化。PMS154C 的数据存储器 128 字节全部都可以用间接方式来存取。

5.4. 振荡器和时钟

PMS154C 提供 3 个振荡器电路：外部晶体振荡器（EOSC），内部高频振荡器（IHRC）与内部低频振荡器（ILRC）。这 3 个振荡器可以分别用寄存器 eoscr.7，clkmd.4 与 clkmd.2 启用或停用，使用者可以选择这 3 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	启用或停用选择
EOSC	eoscr.7
IHRC	clkmd.4
ILRC	clkmd.2

表 2 : PMS154C 提供 3 个振荡器电路

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，PMS154C 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，详情请参阅 IHRC 频率和 V_{DD}、温度的测量图表。

ILRC 的频率会因工厂生产、电源电压和温度而变化，请参阅 DC 规格表。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，PMS154C 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户的程序，校准命令如下所示：

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

p1 =2，4，8，16，32；以提供不同的系统时钟。

p2 =16～18；校准芯片到不同的频率，通常选择 16MHz。

p3 =1.8～5.5；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 3 所示：

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

表 3：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。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 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，PMS154C 不同的状态：

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

开机后，CLKMD = 0x34：

- IHRC 的校准频率为 16MHz@V_{DD}=5V，启用 IHRC 的硬件模块
- 系统时钟 CLK = IHRC/2 = 8MHz
- 看门狗被停止，启用 ILRC，PA5 是在输入模式

(2) .ADJUST_IC SYSCLK=IHRC/4，IHRC=16MHz，V_{DD}=3.3V

开机后，CLKMD = 0x14：

- IHRC 的校准频率为 16MHz@V_{DD}=3.3V，启用 IHRC 的硬件模块
- 系统时钟 CLK = IHRC/4 = 4MHz
- 看门狗被停止，启用 ILRC，PA5 是在输入模式

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

开机后，CLKMD = 0x3C：

- IHRC 的校准频率为 16MHz@V_{DD}=2.5V，启用 IHRC 的硬件模块
- 系统时钟 CLK = IHRC/8 = 2MHz
- 看门狗被停止，启用 ILRC，PA5 是在输入模式

(4) .ADJUST_IC　SYSCLK=IHRC/16，IHRC=16MHz，V_{DD}=2.2V

开机后，CLKMD = 0x1C：

- IHRC 的校准频率为 16MHz@V_{DD}=2.2V，启用 IHRC 的硬件模块
- 系统时钟 CLK = IHRC/16 = 1MHz
- 看门狗被停止，启用 ILRC，PA5 是在输入模式

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

开机后，CLKMD = 0x7C：

- IHRC 的校准频率为 16MHz@V_{DD}=5V，启用 IHRC 的硬件模块
- 系统时钟 CLK = IHRC/32 = 500kHz
- 看门狗被停止，启用 ILRC，PA5 是在输入模式

(6) .ADJUST_IC　SYSCLK=ILRC，IHRC=16MHz，V_{DD}=5V

开机后，CLKMD = 0XE4：

- IHRC 的校准频率为 16MHz@V_{DD}=5V，停用 IHRC 的硬件模块
- 系统时钟 CLK = ILRC
- 看门狗被停止，启用 ILRC，PA5 是在输入模式

(7) .ADJUST_IC　DISABLE

开机后，CLKMD 没改变（沒动作）：

- IHRC 不校准，停用 IHRC 的硬件模块
- 系统时钟 CLK = ILRC 或 IHRC/64（由 Boot-up_Time 决定）
- 看门狗被启用，启用 ILRC，PA5 是在输入模式

5.4.4 外部晶体振荡器

如果要使用晶体振荡器，就需要再在 X1 和 X2 之间放置晶体或谐振器。图 2 显示了使用晶体振荡器的硬件连接；晶体振荡器的工作频率范围可以从 32KHz 至 4MHz，取决于放置的晶体，PMS154C 不支持比 4MHz 更高的频率振荡器。

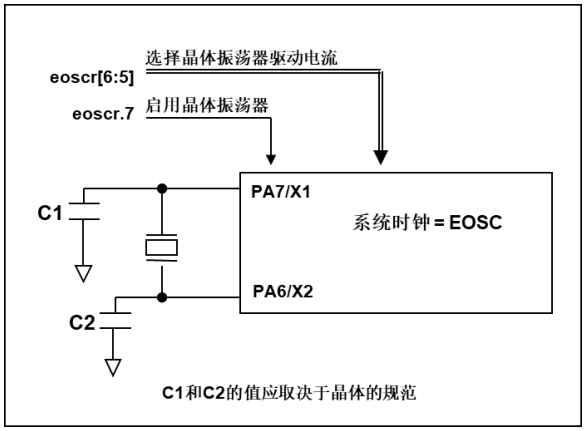


图 2：晶体振荡器的的硬件连接

除了晶振的选择外，外部电容器和 PMS154C 寄存器 eoscr (0x0a)相关选项也应该适度调整以求得有良好的正弦波。*eoscr*.7 是用开启晶体振荡器硬件模块，*eoscr*.6 和 *eoscr*.5 用于设置振荡器不同的驱动电流，以满足晶体振荡器不同频率的的要求：

- *eoscr*[6:5]= 01：驱动电流低，适用于较低的频率，例如：32KHz 晶体振荡器
- *eoscr*[6:5]= 10：中度驱动电流，适用于中间的频率，例如：1MHz 的晶体振荡器
- *eoscr*[6:5]= 11：驱动电流高，适用于较高的频率，例如：4MHz 晶体振荡器

表 4 显示了不同的晶体振荡器 C1 和 C2 的推荐值，同时也显示其对应的条件下测量的起振时间。由于晶体或谐振器有其自身的特点，不同类型的晶体或谐振器的启动时间可能会略有不同，请参考其规格并选择恰当的 C1 和 C2 电容值。

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

表 4：晶体振荡器 C1 和 C2 推荐值

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

```
void    FPPA0 (void)
{
    .ADJUST_IC   SYSCLK=IHRC/16, IHRC=16MHz, VDD=5V
    ...
    $   EOSCR   Enable, 4Mhz;       // EOSCR = 0b110_00000;
    $ T16M   EOSC，  /1，  BIT13;   // T16.Bit13 由 0->1 时，Intrq.T16 => 1
                                    // 假设此时晶体振荡器已稳定
    WORD    count    =    0;
    stt16   count;
    Intrq.T16  =    0;
    while (! Intrq.T16)   NULL;     // 计数从 0x0000 to 0x2000，然后 INTRQ.T16 触发
    clkmd   =    0xB4;              // 切换系统时钟到 EOSC，此时不关闭 IHRC

    clkmd.4 = 0;                    //关闭 IHRC
    ...
}
```

需要注意，在进入掉电模式前，为保证不会被误唤醒，要确保外部晶体振荡器已完全关闭。

5.4.5 系统时钟和 LVR 基准位

系统时钟的时钟源有 EOSC，IHRC 和 ILRC，PMS154C 的时钟系统的硬件框图如图 3 所示。

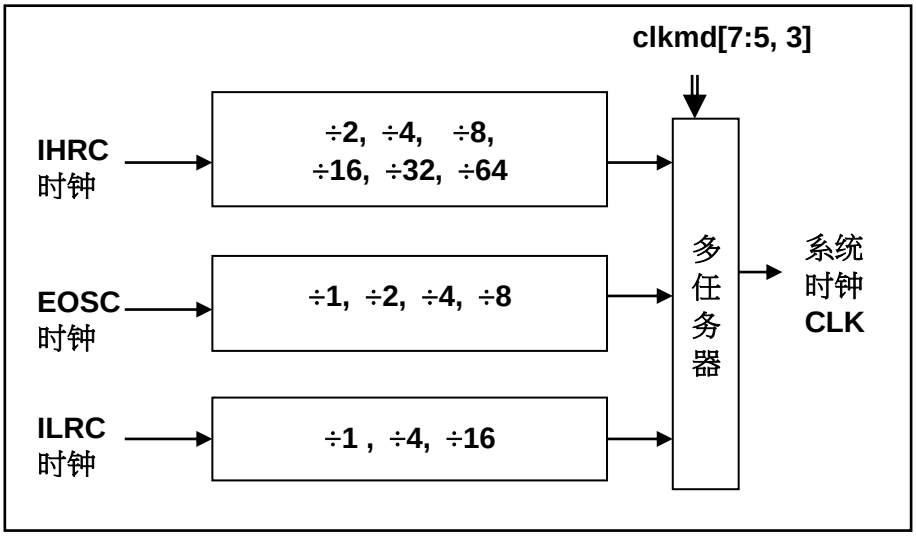


图 3：系统时钟源选择

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 的水平结合，才能使系统稳定。LVR 的水平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。

5.4.6. 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，PMS154C 的系统时钟能够随时通过设定寄存器 ***clkmd*** 在 IHRC、ILRC 和 EOSC 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.4     =   1;              //  先打开 IHRC，可以提高抗干扰能力
CLKMD       =   0x34;           //  切换到 IHRC/2，ILRC 不能在这里停用
// CLKMD.2  =   0;              //  假如需要，ILRC 可以在这里停用
...
```

例 2: 系统时钟从 ILRC 切换到 EOSC

```
...                             //  系统时钟是 ILRC
CLKMD       =   0xA6；          //  切换到 IHRC，ILRC 不能在这里停用
CLKMD.2     =   0；             //  ILRC 可以在这里停用
...
```

例 3: 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 4: 系统时钟从 IHRC/2 切换到 EOSC

```
...                         //  系统时钟是 IHRC/2
CLKMD       =   0XB0 ;      //  切换到 EOSC，IHRC 不能在这里停用
CLKMD.4     =   0 ;         //  IHRC 可以在这里停用
...
```

例 5: 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 6: 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. 16 位计数器 (Timer16)

PMS154C 内置一个 16 位硬件计数器，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）、外部晶体振荡（EOSC）或 PA0 和 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 位计数器的位 8 到位 15，中断类型可以上升沿触发或下降沿触发，是经由寄存器 *integs*.4 选择。Timer16 模块框图如图 4。

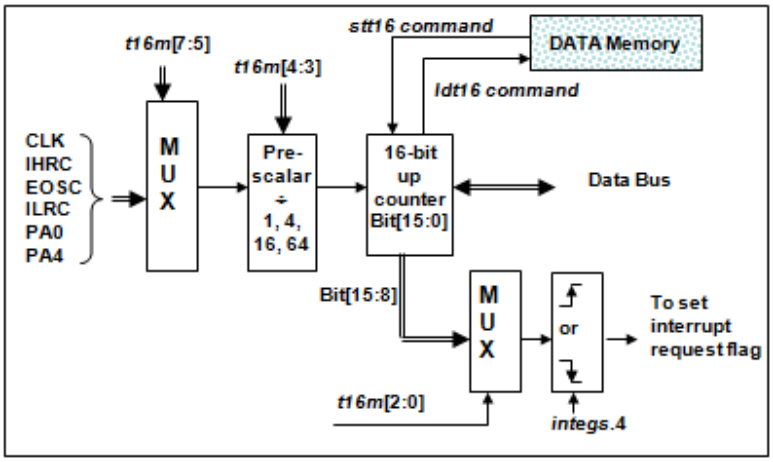


图 4：Timer16 模块框图

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, PA4_F, IHRC, EOSC, 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
// 如果系统时钟 System Clock = IHRC / 2 = 8 MHz
// 则 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
```

5.6. 看门狗

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

- 当 misc[1:0]=00（默认）时：8K 个 ILRC 时钟周期
- 当 misc[1:0]=01 时：16K 个 ILRC 时钟周期
- 当 misc[1:0]=10 时：64K 个 ILRC 时钟周期
- 当 misc[1:0]=11 时：256K 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。在上电复位或任何时候使用 wdreset 指令，看门狗都会被清零。当看门狗超时溢出时，PMS154C 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。

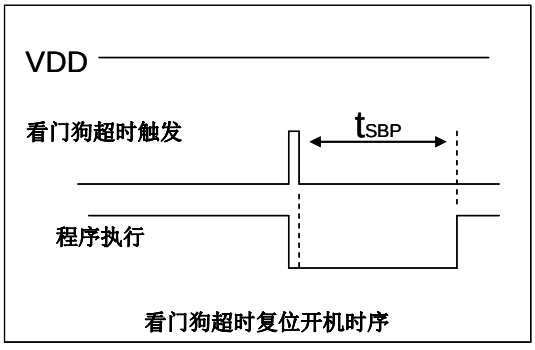


图 5：看门狗超时溢出的相关时序

5.7. 中断

PMS154C 有 7 个中断源：外部中断源 PA0 和 PB0，计数器中断源 Timer16，比较器，Timer2，Timer3，PWM 发生器 0。每个中断请求源都有自己的中断控制位启用或停用它。硬件框图请参考图 6，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。

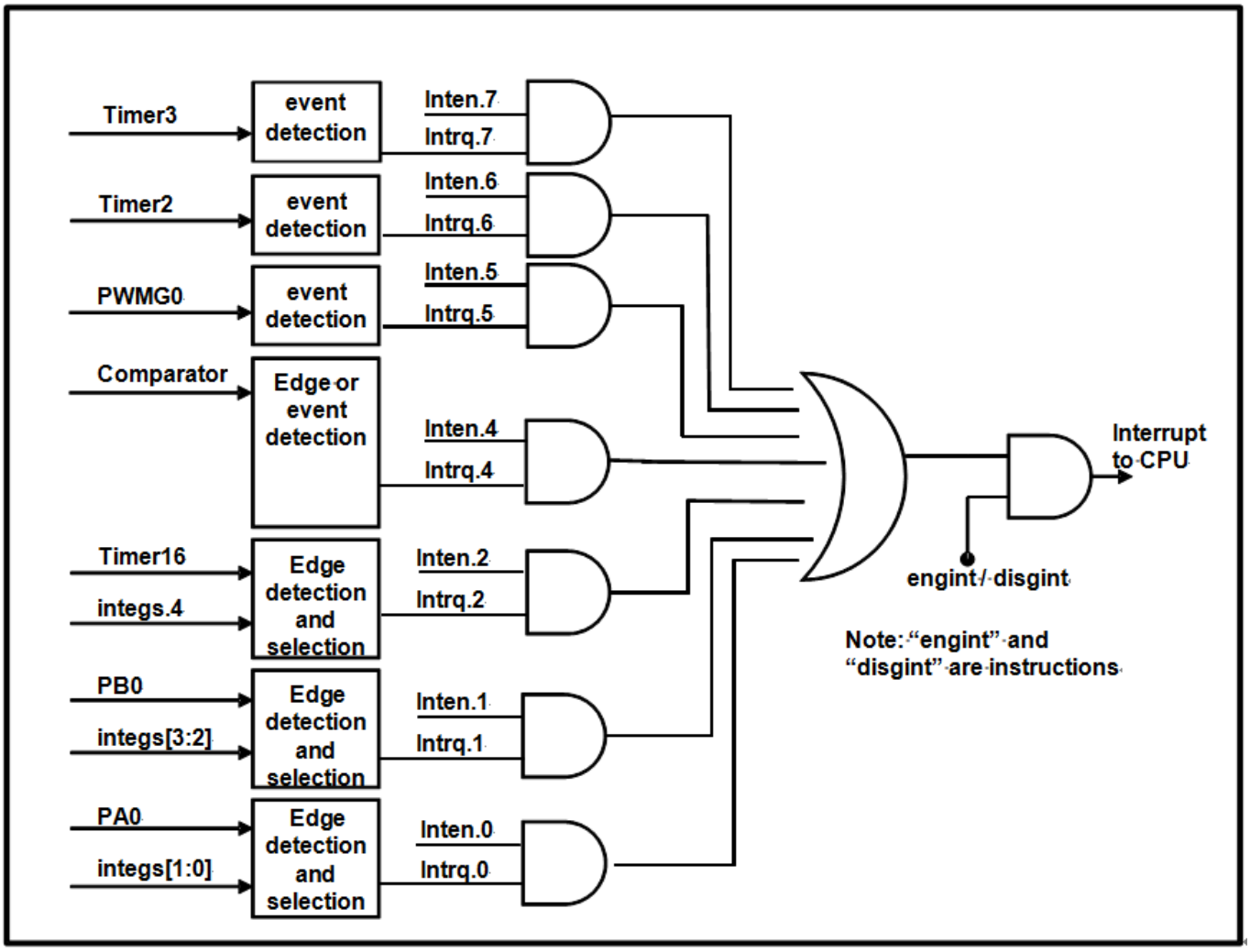


图 6：中断硬件框图

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。

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

使用者必须预留足够的堆栈存储器以存中断向量，一级中断需要两个字节，两级中断需要 4 个字节。依此类推，另外 *pushaf* 也需要两个字节。下面的示例程序演示了如何处理中断，请注意，处理一级中断和 *pushaf* 总共需要四个字节堆栈存储器。

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

5.8. 省电与掉电

PMS154C 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 5 显示省电模式（*stopexe*）和掉电模式（*stopsys*）之间在振荡器模块的差异，没改变就是维持原状态。

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

表 5 ：省电模式和掉电模式在振荡器模块的差异

5.8.1 省电模式（stopexe）

使用 *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.6 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 和 EOSC 振荡器模块：没有变化。如果它被启用，它仍然继续保持活跃。
- ILRC 振荡器模块：必须保持启用，唤醒时需要靠 ILRC 启动。
- 系统时钟停用。因此，CPU 停止执行。
- OTP 存储器被关闭。
- Timer 计数器：若 Timer 计数器的时钟源是系统时钟或其相应的时钟振荡器模块被停用，则 Timer 停止计数；否则，仍然保持计数。（其中，Timer 包含 Timer16，TM2，TM3，PWMG0，PWMG1，PWMG2）
- 唤醒来源：
  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。但请注意：内部 1.20V Bandgap 参考电压不适用于比较器唤醒功能。

以下例子是利用 Timer16 来唤醒系统因 ***stopexe*** 的省电模式：

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

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

5.8.2 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 PMS154C 芯片直接进入掉电模式。在进入掉电模式之前，必须启用内部低频振荡器（ILRC）以便唤醒系统时使用，也就是说在发出 *stopsys* 命令之前，*clkmd* 寄存器的位 2 必须设置为 1，同时建议将 GPCC.7 设为 0 来关闭比较器。下面显示发出 *stopsys* 命令后，PMS154C 内部详细的状态：

- 所有的振荡器模块被关闭。
- OTP 存储器被关闭。
- SRAM 和寄存器内容保持不变。
- 唤醒源：设定为数字模式（PxDIER 对应位为 1）的 IO 切换。

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

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

5.8.3 唤醒

进入掉电或省电模式后，PMS154C 可以通过切换 IO 引脚恢复正常工作；而 Timer 的唤醒只适用于省电模式。表 6 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	计时器唤醒	比较器唤醒
stopsys	是	否	否
stopexe	是	是	是

表 6：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 PMS154C，寄存器 *pxdier* 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 3000 ILRC 时钟周期；另外，PMS154C 提供快速唤醒功能，透过 *misc* 寄存器选择快速唤醒大约 45 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 STOPSYS 掉电模式	快速唤醒	45 * $T_{ILRC,}$ 这里 $T_{ILRC,}$是 ILRC 时钟周期
STOPEXE 省电模式 STOPSYS 掉电模式	普通唤醒	3000 * $T_{ILRC,}$ 这里 $T_{ILRC,}$是 ILRC 时钟周期

表 7：切换 IO 引脚的唤醒时间(t_{WUP})

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

5.9. IO 引脚

除了 PA5，PMS154C 所有 IO 引脚都可以设定成输入或输出，透过数据寄存器（*pa, pb*），控制寄存器（*pac, pbc*）和弱上拉电阻（*paph, pbph*）设定，每一 IO 引脚都可以独立配置成不同的功能；所有这些引脚设置有施密特触发输入缓冲器和 CMOS 输出驱动电位水平。当这些引脚为输出低电位时，弱上拉电阻会自动关闭。如果要读取端口上的电位状态，一定要先设置成输入模式；在输出模式下，读取到的数据是数据寄存器的值。图 7 显示了 IO 缓冲区硬件图，表 8 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 8：PA0 设定配置表

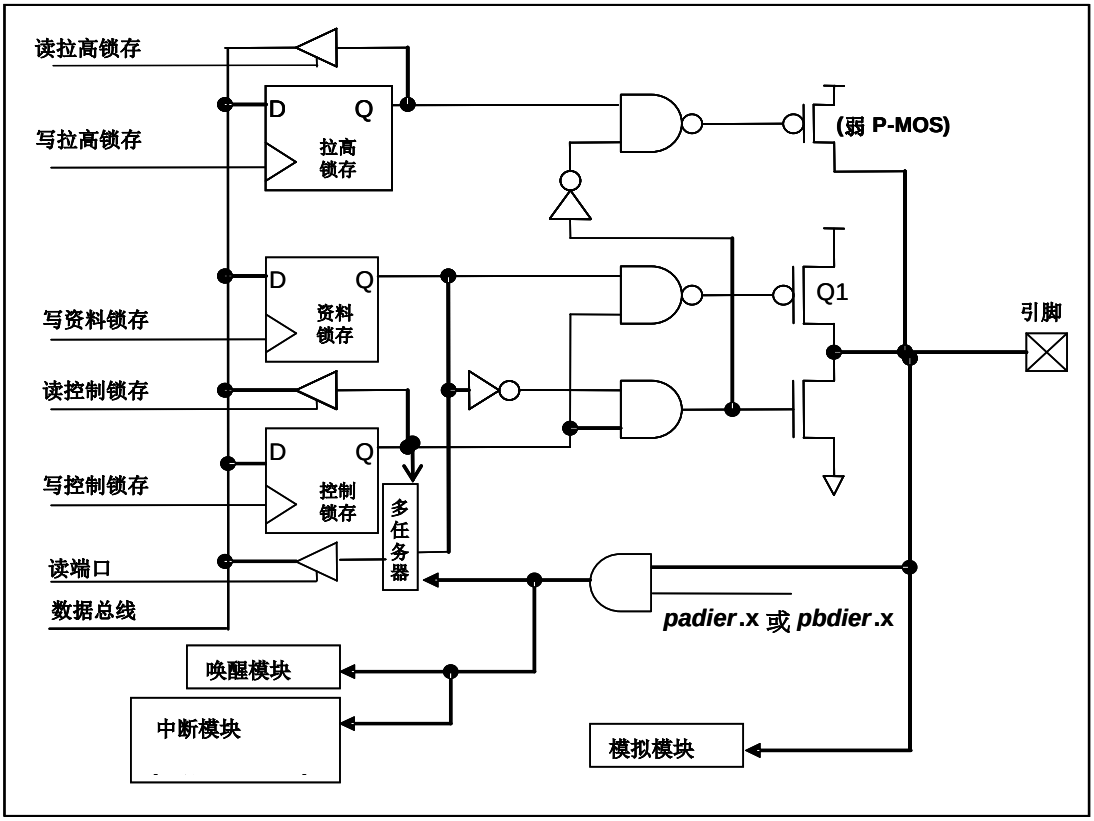


图 7：引脚缓冲区硬件图

透过代码选项(Code Option) ***Drive*** 大多数 IO 可以被调整其驱动(drive)或灌(sink)电流能力（正常或低电流）。

除了 PA5 外，所有的 IO 引脚具有相同的结构；PA5 的输出只能是漏极开路模式（没有 Q1）。对于被选择为模拟功能的引脚，必须在寄存器 *pxdier* 相应位设置为低，以防止漏电流。当 PMS154C 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 *pxdier* 相应为高。同样的原因，当 PA0 或 PB0 用来作为外部中断引脚时，*padier.0* 或 *pbdier*.0 应设置高。

5.10. 复位和 LVR

5.10.1 复位

引起 PMS154C 复位的原因有很多，一旦复位发生，PMS154C 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。

发生上电复位或 LVR 复位后，若 VDD 大于 V_{DR}（数据保存电压），数据存储器的值将会被保留；若 VDD 小于 V_{DR}，数据存储器的值将转为未知的状态。

发生复位，且程序中有加清除 SRAM 的指令或语法，则先前的数据将会在程序初始化中被清除，无法保留。

若复位是因为 PRSTB 引脚或 WDT 超时溢位，数据存储器的值将被保留。

5.10.2 LVR 复位

程序编译时，有很多不同级别的 LVR 复位电压可供选择；通常情况下，使用者在选择 LVR 复位水平时，必须结合单片机工作频率和电源电压，以便让单片机稳定工作。

5.11. 半电位偏置电压

这项功能是用来产生半电位(VDD/2)，以做为驱动液晶显示器的功能，它并不合适在需要极度省电的应用产品上。该功能可以透过 misc.4 和代码选项 LCD2 去设置和启用， 要使用此功能，用户必须为 LCD2 选择 PB0_A034，并在程序中将 misc.4 设置为 1，PA4、PA3、PA0、PB0 这四支引脚可以输出半电位，以做为驱动液晶显示器时 COM 的功能。当被选定的引脚希望有输出半电位的功能时，用户只需要将相对应的引脚设为输入模式，PMS154C 将自动在该引脚产生半电位。如果使用者想要输出高电位、半电位、GND 三个层次，只要设置 *misc* 寄存器位 4，然后输出高电位（V_{DD}）、输入（VDD/2）、输出低电位（GND）即可产生三种相对应的电位，图 8 显示了如何使用此功能。

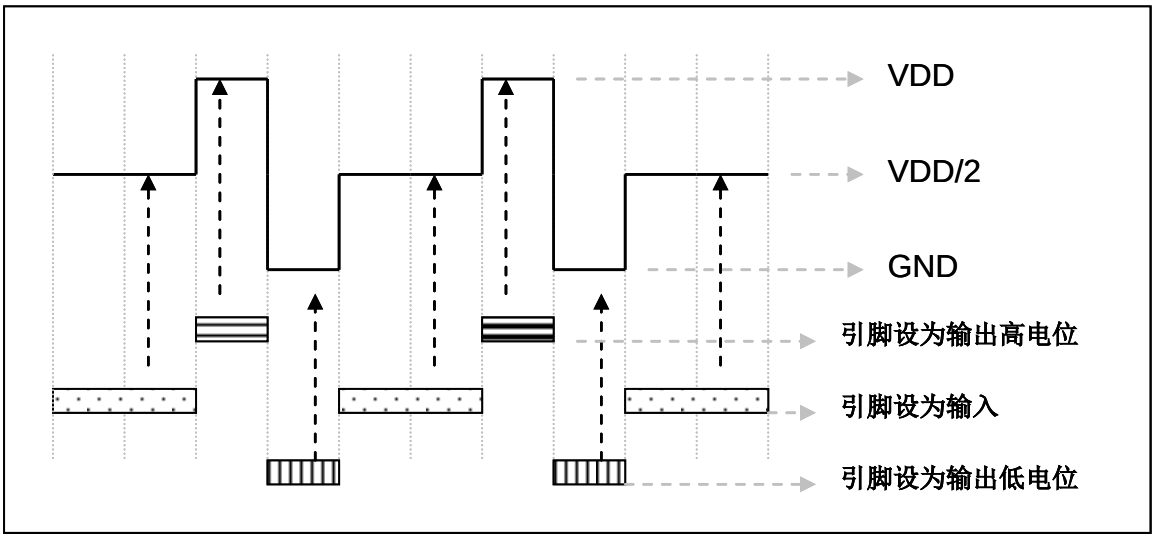


图 8：使用半电位偏置电压

5.12. 比较器

PMS154C 内部内置了一个比较器，图 9 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 Bandgap 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，bandgap 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。

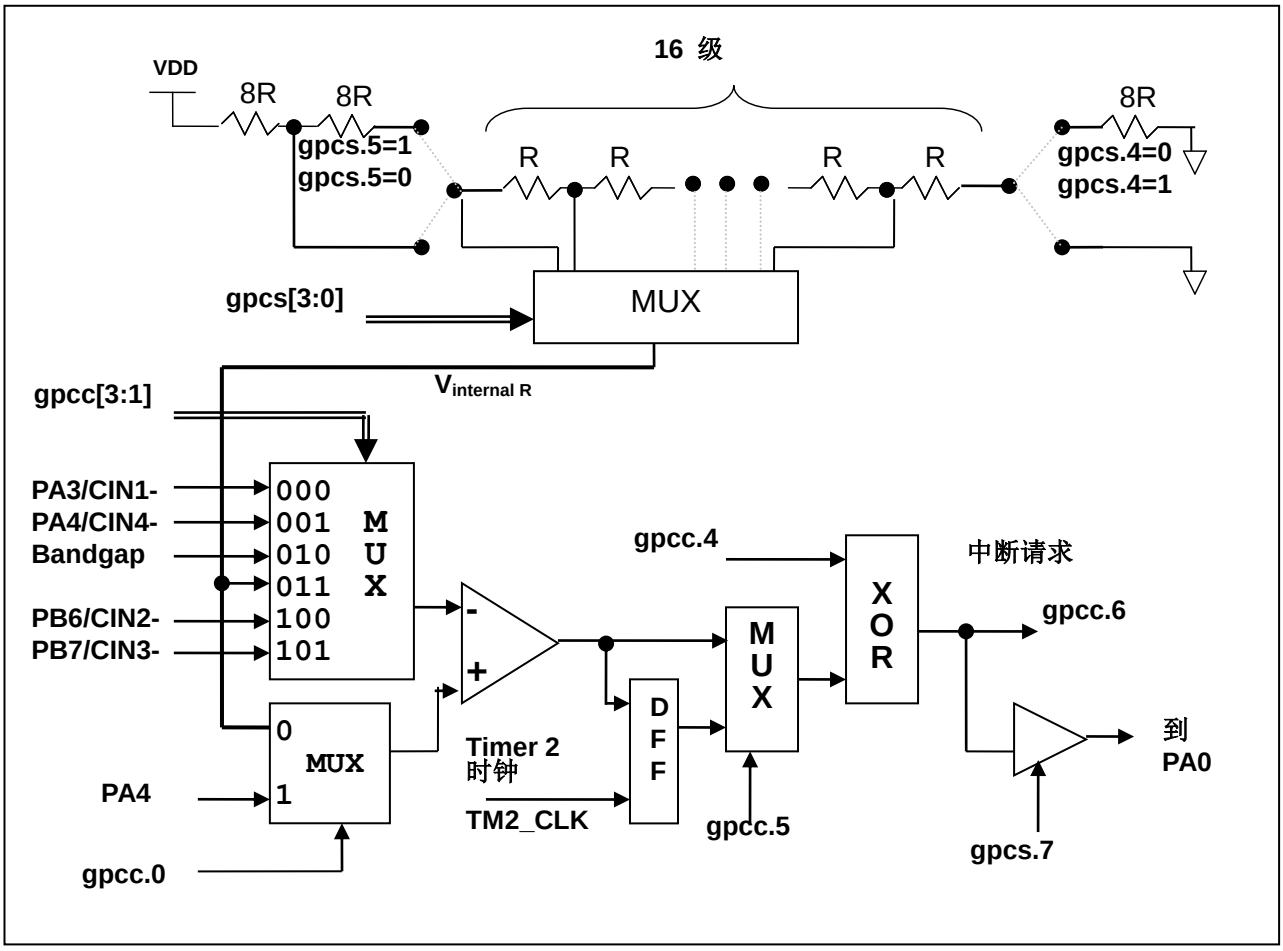


图 9：比较器硬件图框

5.12.1 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 10 ~ 图 13 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}。

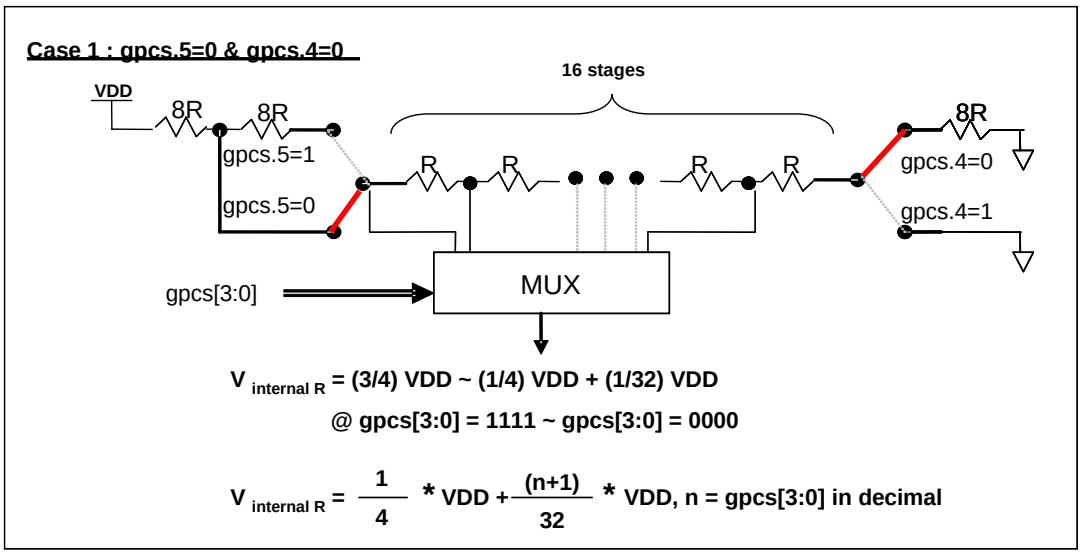


图 10：$V_{internal\ R}$ 硬件接法 (gpcs.5=0 & gpcs.4=0)

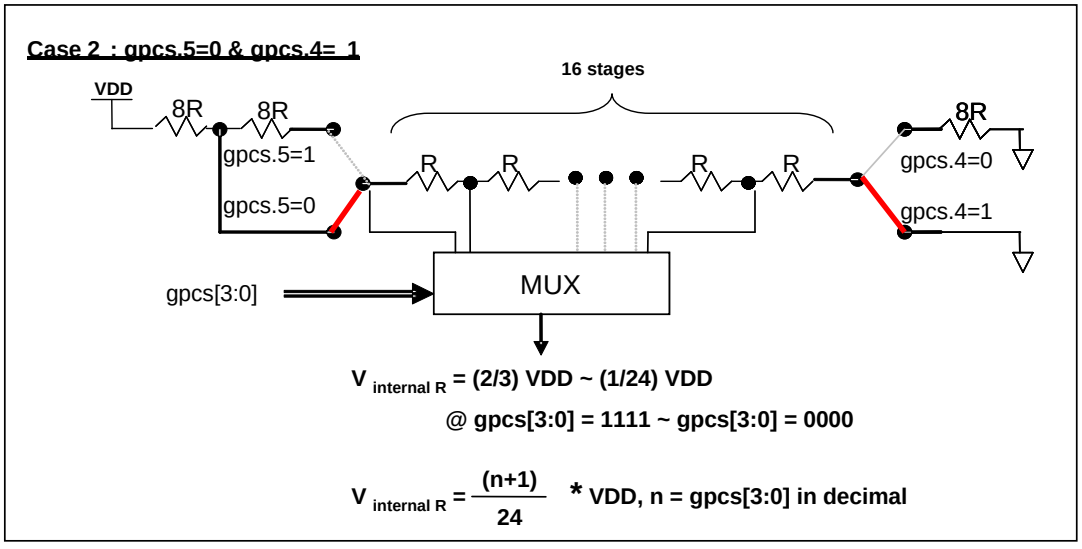


图 11：$V_{internal\ R}$ 硬件接法 (gpcs.5=0 & gpcs.4=1)

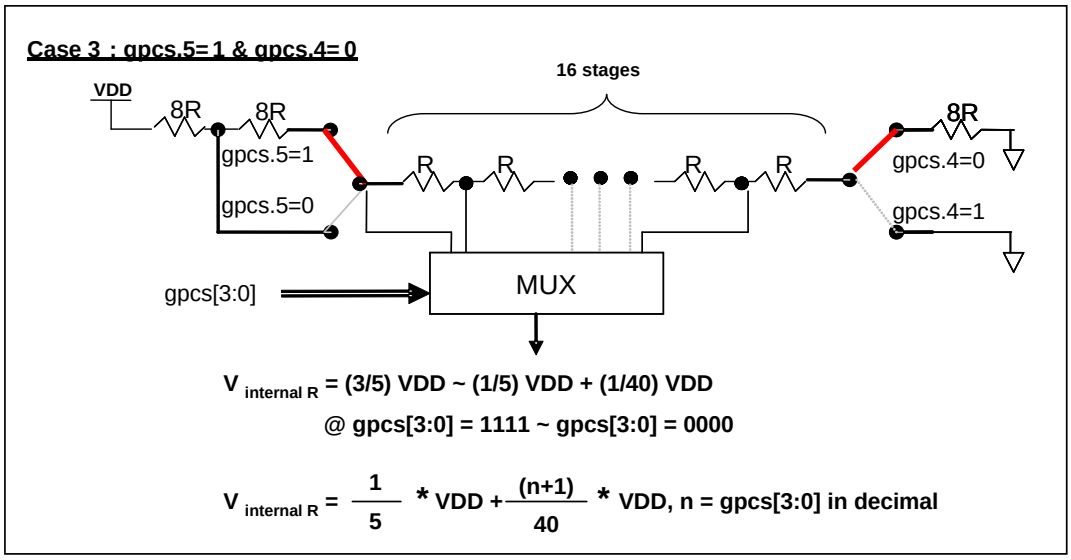


图 12：$V_{internal\ R}$ 硬件接法 (gpcs.5=1 & gpcs.4=0)

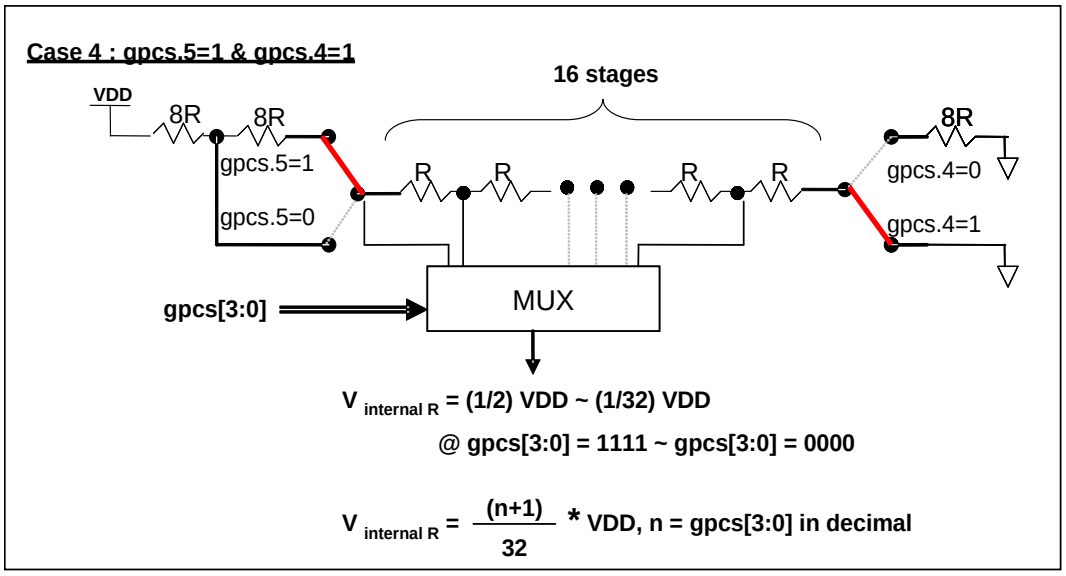


图 13：$V_{internal\ R}$ 硬件接法 (gpcs.5=1 & gpcs.4=1)

5.12.2 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)*V_{DD}。$V_{internal\ R}$ 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    = 0b0_0_00_1001;      // Vinternal R = (18/32)*VDD
gpcc    = 0b1_0_0_0_000_0;    // 启用比较器，负输入=PA3-，正输入=Vinternal R
padier  = 0bxxxx_0_xxx;       // 停用 PA3 数字输入防止漏电  (x 表示用户自定)
```

或

```
$ GPCS    VDD*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;
```

例二:

选择 $V_{internal\ R}$ 为负输入，$V_{internal\ R}$ 的电压为(22/40)*V_{DD} 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)*V_{DD}。$V_{internal\ R}$ 选择上图 gpcs[5:4] = 2b'10 的配置方式，gpcs [3:0] = 4b'1101 (n=13) 以得到 $V_{internal\ R}$ =(1/5)*V_{DD} + [(13+1)/40]*V_{DD} = [(13+9)/40]*V_{DD} = (22/40)*V_{DD}。

```
gpcs    = 0b1_0_10_1101;      //  输出到 PA0，Vinternal R = VDD*(22/40)
gpcc    = 0b1_0_0_1_011_1;    // 输出反极性，负输入= Vinternal R，正输入=PA4
padier  = 0bxxxx_0_xxx;       // 停用 PA4 数字输入防止漏电  (x 表示用户自定)
```

或

```
$ GPCS    Output, VDD*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，所以仿真时请注意避开这错误。

5.12.3. 使用比较器和 bandgap 参考电压生成器

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

5.13 8 位 PWM 计数器(Timer2,Timer3)

PMS154C 内置 2 个 8 位 PWM 硬件定时器（Timer2/TM2,Timer3/TM3），硬件框图请参考图 14，两个计数器的原理一样，以下以 Timer2 来说明。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），外部晶体震荡（EOSC），PA0，PA4，PB0 或者比较器的输出 。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[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~÷32 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 位或 8 位。图 15 显示出 Timer2 周期模式和 PWM 模式的时序图。

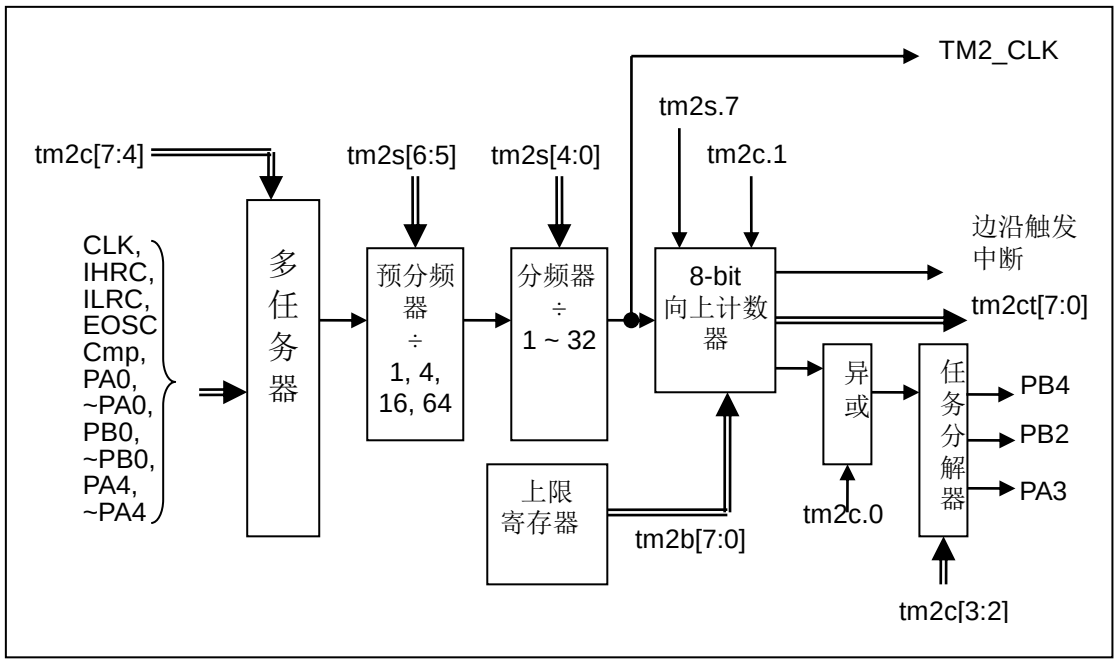


图 14：Timer2 模块框图

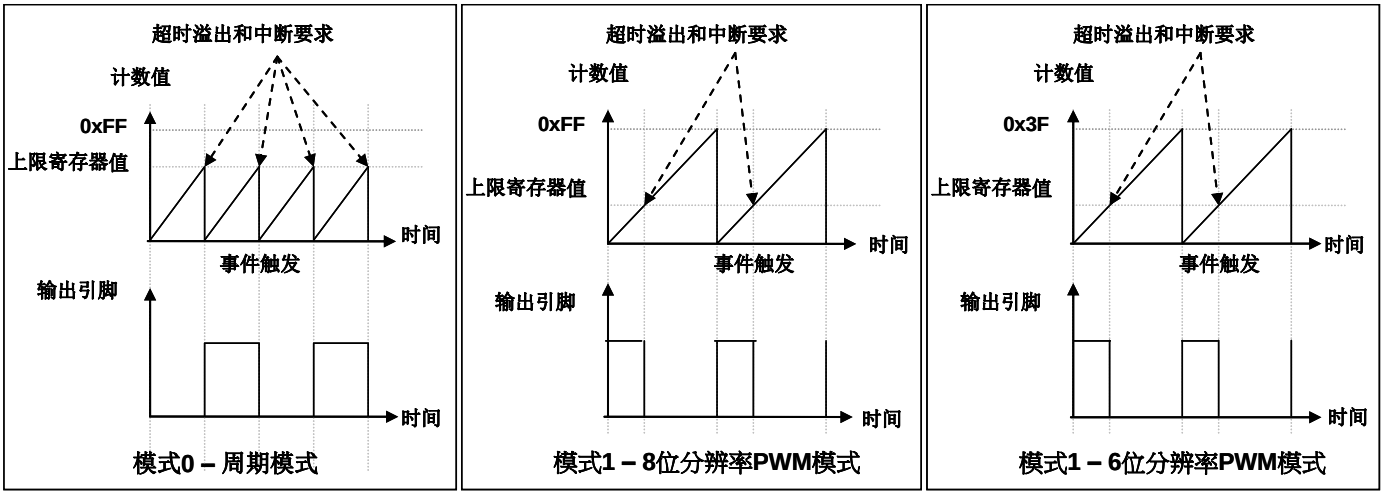


图 15：Timer2 周期模式和 PWM 模式的时序图

5.13.1 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(S1= 1，4，16，64)
S2 = tm2s[4:0]：分频器值(十进制，S2= 0 ~ 31)

例 1：

tm2c = 0b0001_1100，Y=8MHz
tm2b = 0b0111_1111，K=127
tm2s = 0b0_00_00000，S1=1，S2=0
➔ 输出频率 = 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 = 0x00;
    tm2b = 0x7f;
    tm2s = 0b0_00_00001;      // 8 位 pwm，预分频 = 1，分频 = 2
    tm2c = 0b0001_10_0_0;     //系统时钟，输出 =PA3，周期模式
    while(1)
    {
        nop;
    }
}
```

5.13.2 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]×100%

这里，

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

例 1：

tm2c = 0b0001_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 = 0x00;
    tm2b = 0x7f;
    tm2s = 0b0_00_00001;          // 8 位 pwm，预分频 = 1，分频 = 2
    tm2c = 0b0001_10_1_0;         //系统时钟，输出 = PA3，PWM 模式
    while(1)
    {
        nop;
    }
}
```

5.13.3 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]×100%

这里，

Y = Tm2c[7:4]：Timer2 所选择的时钟源频率
K = tm2b[7:0]：上限寄存器设定的值(十进制)
S1 = tm2s[6:5]：预分频器设定值(S1= 1，4，16，64)
S2 = tm2s[4:0]：分频器值(十进制，S2= 0 ~ 31)

例 1：

tm2c = 0b0001_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%

5.14 11 位 PWM 计数器

在 PMS154 中执行了三个 11 位 PWM 生成器（PWMG0、PWMG1 和 PWMG2）。以 PWMG0 作为示例来描述其功能，因为它们几乎相同。各路输出端口如下：

- PWMG0 – PA0, PB4, PB5
- PWMG1 – PA4, PB6, PB7
- PWMG2 – PA3, PB2, PB3, PA5（注：PA5 只有开漏输出，使用时需打开内部上拉或外加上拉电阻，且仿真器不支持 PA5 PWM 功能）

5.14.1 PWM 波形

PWM 波形（图 16）有一个时基（T_{Period} =时间周期）和一个周期里输出高的时间（占空比）。PWM 的频率取决于时基($f_{PWM} = 1/T_{Period}$)，PWM 的分辨率取决于一个时基里的计数个数(N 位分辨率, $2^N \times T_{clock} = T_{Period}$)。

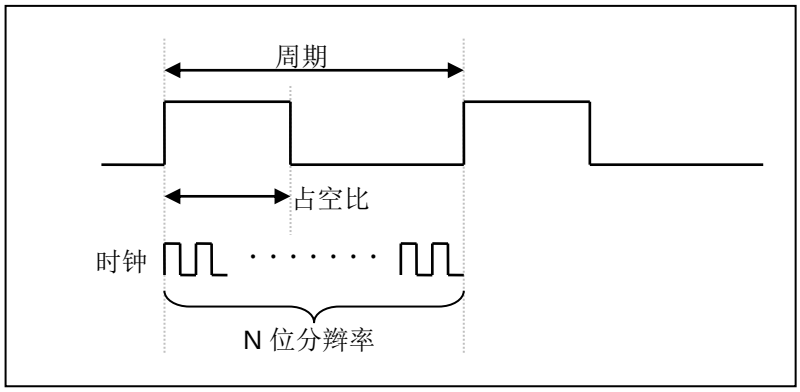


图 16：PWM 输出波形

5.14.2 硬件和时钟框图

图 17 是 11 位计数器的硬件框图。这个计数器的时钟源可以是 IHRC 或者系统时钟。依据寄存器 ***PWMC*** 的设定，使用者可以选择性将 PWM 输出到 PA0，PB4 或者 PB5。此时无论 PX.x 是输入还是输出的状态，PWM 信号都会被强制输出。PWM 的周期由 PWM 上限高和低寄存器决定，PWM 的占空比由 PWM 占空比高和低寄存器决定。

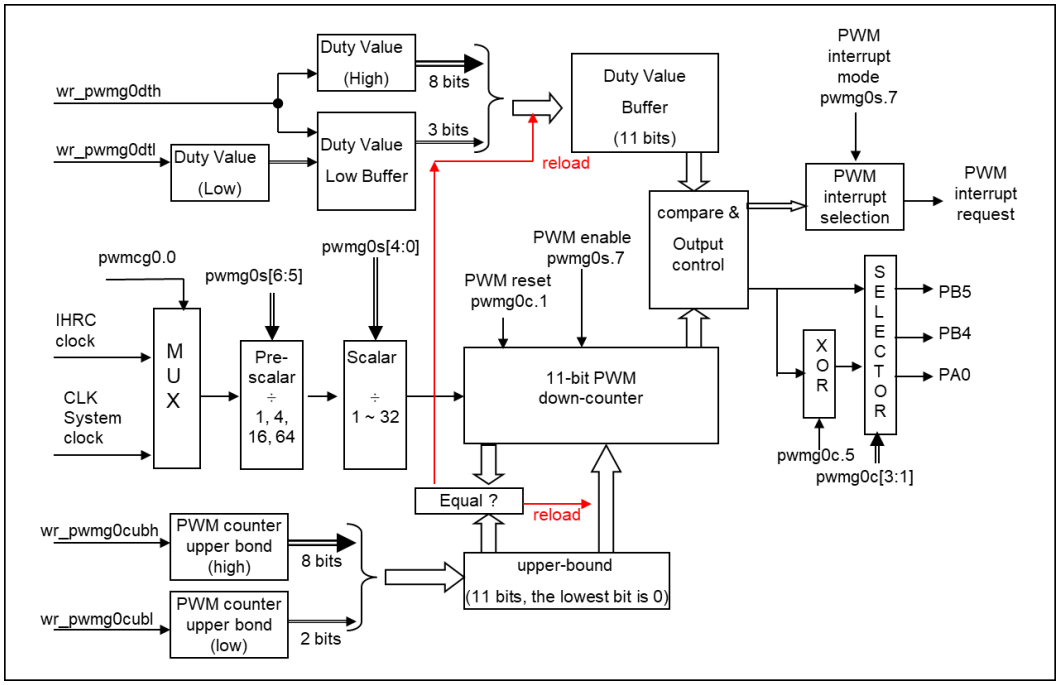


图 17：11 位 PWM 生成器（PWMG0）硬件框图

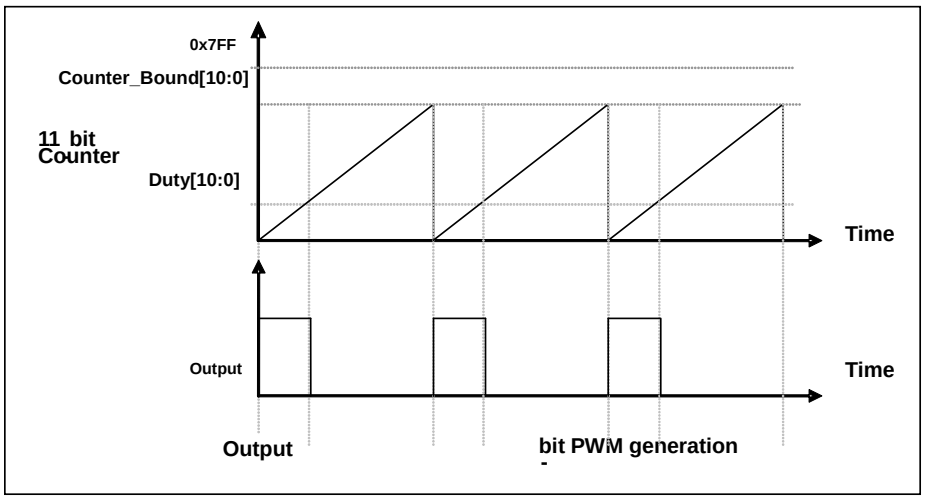


图 18：11 位 PWM 生成器输出时序图

5.14.3 11 位 PWM 生成器计算公式

11bit PWM 的频率和占空比可由下公式得出：

PWM 输出频率 F_{PWM} = $F_{clock\ source}$ ÷ [P × (K + 1) × (CB10_1 + 1)]

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

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

这里,

P = *PWMGxS* [6:5]：预分频（**P** = 1, 4, 16, 64)

K = *PWMGxS* [4:0]：分频器值（十进制，**K** = 0 ~ 31）

DB10_1 = Duty_Bound[10:1] = {*PWMGxDTH*[7:0],*PWMGxDTL*[7:6]}, 占空比

DB0 = Duty_Bound[0] = *PWMGxDTL*[5]

CB10_1 = Counter_Bound[10:1] = {*PWMGxCUBH*[7:0],*PWMGxCUBL*[7:6]}, 计数器

5.14.4 带互补死区的 PWM 波形范例

用户可以用两个 11bit PWM 生成器输出两路互补带死区的 PWM 波形。以 PWMG0 输出 PWM0 及 PWMG1 输出 PWM1 为例，（Timer2 及 Timer3 也可输出两路带互补死区的 8bit PWM 波形，其原理与此类似，不再详细描述），程序参考如下：

```
#define dead_zone_R   2          //    用于调控 PWM1 上升沿之前的死区时间，可修改
#define dead_zone_F   3          //    用于调控 PWM1 下降沿之后的死区时间，可修改

void    FPPA0 (void)
{
  .ADJUST_IC    SYSCLK=IHRC/16, IHRC=16MHz, VDD=5V;
  //.....
  Byte duty     =    60;                 //    代表 PWM0 的占空比
  Byte _duty    =    100 - duty;         //    代表 PWM1 的占空比

  //************** 设置计数上限及占空比 **************
  PWMG0DTL    =    0x00;
  PWMG0DTH    =    _duty;
  PWMG0CUBL   =    0x00;
  PWMG0CUBH   =    100;
```

```
    PWMG1DTL    =    0x00;
    PWMG1DTH    =    _duty - dead_zone_F;     //    用 duty 调节 PWM1 下降沿之后的死区时间
    PWMG1CUBL   =    0x00;
    PWMG1CUBH   =    100;
    //    以上放在开 PWM 之前赋值

    //******************** 输出控制 **********************
    $ PWMG0C    Enable,Inverse,PA0,SYSCLK;       //    PWMG0 输出 PWM0 波形到 PA0
    $ PWMG0S    INTR_AT_DUTY,/1,/1;

    .delay      dead_zone_R;              //    用 delay 的方式调节 PWM1 上升沿之前的死区时间

    $ PWMG1C    Enable, PA4, SYSCLK;              //    PWMG1 输出 PWM1 波形到 PA4
    $ PWMG1S    INTR_AT_DUTY, /1, /1;

    //***** 注意：针对输出控制部分的程序，代码顺序不能动 *********//

    While(1)
    {    nop;    }
}
```

以上程序得到的 PWM0 / PWM1 波形如图 19 所示。

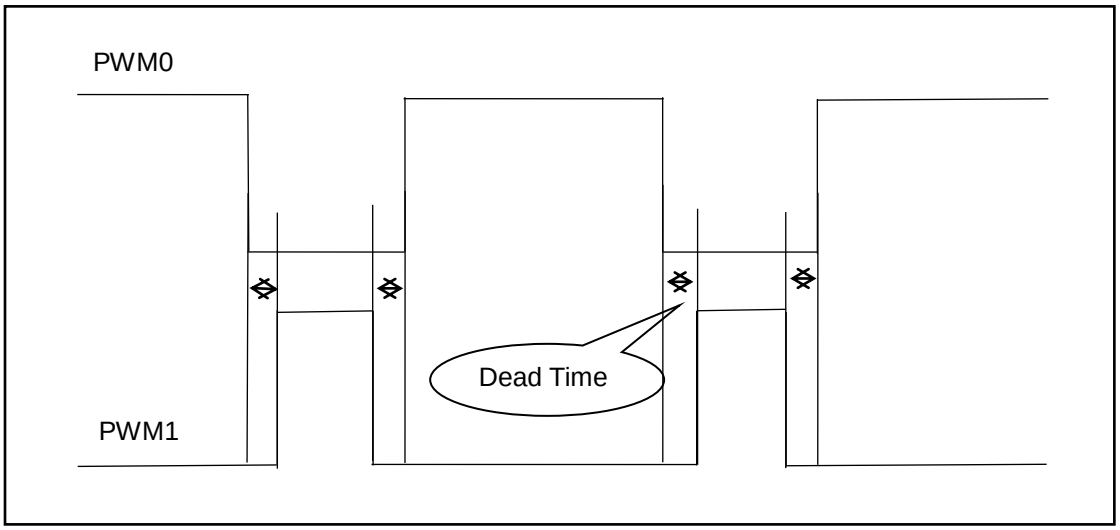


图 19：两路互补 PWM 波形

用户可以修改程序中 **dead_zone_R** 和 **dead_zone_F** 的数值来调节 PWM1 波形前/后死区时间的长短。表 8 提供几组不同死区时间对应的数据，供用户参考。其中，若 dead time = 4us，则 PWM1 高电平前/后各有 4us 的死区。

dead Time (us)	dead_zone_R	dead_zone_F
4（最小值）	0	2
6	2	3
8	4	4
10	6	5
12	8	6
14	10	7

表 8：死区时间参考数值

dead_zone_R 和 **dead_zone_F** 需要共同配合才能得到理想的死区时间。若用户想要调整其他死区时间，请注意 **dead_zone_R** 和 **dead_zone_F** 需要符合以下条件：

dead_zone_R	dead_zone_F
1 / 2 / 3	> 1
4 / 5 / 6 / 7	> 2
8 / 9	> 3
...	...

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
010：保留
011：EOSC/4
100：EOSC/2
101：EOSC
110：ILRC/4
111：ILRC（默认）</td><td>000：IHRC/16
001：IHRC/8
010：ILRC/16（仿真器不支持）
011：IHRC/32
100：IHRC/64
101：EOSC/8
其他：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：停用/启用</td></tr>
<tr><td>3</td><td>0</td><td>读/写</td><td colspan="2">时钟类型选择。这个位是用来选择位 7~位 5 的时钟类型。
0/1：类型 0/类型 1</td></tr>
<tr><td>2</td><td>1</td><td>读/写</td><td colspan="2">内部低频 RC 振荡器功能。0/1：停用/启用
当内部低频 RC 振荡器功能停用时，看门狗功能同时被关闭。</td></tr>
<tr><td>1</td><td>1</td><td>读/写</td><td colspan="2">看门狗功能。0/1：停用/启用</td></tr>
<tr><td>0</td><td>0</td><td>读/写</td><td colspan="2">引脚 PA5/PRSTB 功能。0/1：PA5/PRSTB</td></tr>
</table>

6.4. 中断允许寄存器(*inten*)，IO 地址 = 0x04

位	初始值	读/写	描 述
7	-	读/写	启用从 Timer3 的溢出中断。0/1：停用/启用
6	-	读/写	启用从 Timer2 的溢出中断。0/1：停用/启用
5	-	读/写	启用从 PWMG0 的溢出中断。0/1：停用/启用
4	-	读/写	启用从比较器的溢出中断。0/1：停用/启用
3	-	读/写	保留。
2	-	读/写	启用从 Timer16 的溢出中断。0/1：停用/启用
1	-	读/写	启用从 PB0 的溢出中断。0/1：停用/启用
0	-	读/写	启用从 PA0 的中断。 0/1：停用/启用

6.5. 中断请求寄存器(*intrq*)，IO 地址 = 0x05

位	初始值	读/写	描 述
7	-	读/写	Timer3 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
6	-	读/写	Timer2 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
5	-	读/写	PWMG0 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
4	-	读/写	比较器的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
3	-	读/写	保留。
2	-	读/写	Timer16 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
1	-	读/写	PB0 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
0	-	读/写	PA0 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求

6.6. 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：停用 Timer16 001：CLK 系统时钟 010：保留 011：PA4（外部事件） 100：IHRC 101：EOSC 110：ILRC 111：PA0（外部事件）
4 – 3	00	读/写	Timer16 内部的时钟分频器。 00：÷1 01：÷4 10：÷16 11：÷64
2 – 0	000	读/写	中断源选择。当选择位由低变高或高变低时，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	0	只写	使能外部晶体振荡器。0/1：停用/启用
6 – 5	00	只写	晶体振荡器的选择。 00：保留 01：低驱动电流。适用于较低频率晶体，例如：32KHz 10：中驱动电流。适用于中等频率晶体，例如：1MHz 11：高驱动电流。适用于较高频率晶体，例如：4MHz
4 – 1	-	-	保留。请设为 0。
0	0	只写	将 Bandgap 和 LVR 硬件模块断电。0/1：正常/ 断电 请注意：Bandgap 关闭后将仅 ILRC/T16/TM2/TM3 及 I/O 功能可用。

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	PB0 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。
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	11111	只写	启用 PA7~PA3 数字输入和系统唤醒。1/0：启用/停用 当这个位设为 0 时，PA7~PA3 无法用来唤醒系统。
2 – 1	-	-	保留。（建议写 00）
0	1	只写	启用 PA0 数字输入、系统唤醒和中断请求。1/0：启用/停用 当这个位设为 0 时，PA0 无法用来唤醒系统以及中断请求。

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

位	初始值	读/写	描 述
7 – 1	0xFF	只写	启用 PB7~PB1 数字输入和系统唤醒。1/0：启用/停用 当这个位设为 0 时，PB7~PB1 无法用来唤醒系统。
0		只写	启用 PB0 数字输入、系统唤醒和中断请求。1/0：启用/停用 当这个位设为 0 时，PB0 无法用来唤醒系统以及中断请求。

6.11. 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	数据寄存器的端口 A。

6.12. 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

6.13. 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：停用/启用

6.14. 端口 B 数据寄存器(*pb*)，IO 地址 = 0x14

位	初始值	读/写	描 述
7 – 0	0x00	读/写	数据寄存器的端口 B。

6.15. 端口 B 控制寄存器(*pbc*)，IO 地址 = 0x15

位	初始值	读/写	描 述
7 – 0	0x00	读/写	端口 B 控制寄存器。这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

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

位	初始值	读/写	描 述
7 – 0	0x00	读/写	端口 B 上拉控制寄存器。这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

6.17. 杂项寄存器(*misc*)，IO 地址 = 0x08

位	初始值	读/写	描 述
7 – 6	-	-	保留。
5	0	只写	快唤醒功能。EOSC 使能时，不支持快唤醒。 0：正常唤醒。唤醒时间为 3000 ILRC 时钟 1：快唤醒。唤醒时间为 45 ILRC 时钟
4	0	只写	使能 LCD 显示 VDD/2 功能。0/1：停用/启用（ICE 仿真时无法动态切换） 如果 Code Option 有选择 LCD 输出，但 MISC.4 没有设为 1，则在 IC 上还是无法输出 VDD/2 bias，但仿真器则恒可以，此处两边现象不同。
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 个 ILRC 时钟周期 01：16K 个 ILRC 时钟周期 10：64K 个 ILRC 时钟周期 11：256K 个 ILRC 时钟周期

6.18. 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：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） 其他：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：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

6.19. 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]。

6.20. 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。

6.21. 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。

6.22 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：停用 0001：CLK（系统时钟） 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） 其他：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	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

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

位	初始值	读/写	描 述
7 – 0	0x00	读/写	Timer3 定时器位[7:0]。

6.24 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
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。

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

位	初始值	读/写	描 述
7 – 0	0x00	只写	Timer2 上限寄存器。

6.26 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（不适用于比较器唤醒功能） 011：$V_{internal\ R}$ 100：PB6 101：PB7 11X：保留
0	0	读/写	选择比较器正输入的来源。 0：$V_{internal\ R}$ 1：PA4

6.27 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 （在仿真器上，输出到 PA0 也会造成 PA3 输出不良，请避开此问题）
6	-	只写	比较器唤醒启用。（gpcc.6 发生电平变化时才可唤醒） 0/1：停用/启用
5	0	只写	选择比较器参考电压 $V_{internal\ R}$ 最高的范围。
4	0	只写	选择比较器参考电压 $V_{internal\ R}$ 最低的范围。
3 – 0	0000	只写	选择比较器参考电压 $V_{internal\ R}$。 0000（最低）~ 1111（最高）

6.28 PWMG0 控制寄存器(*pwmg0c*)，IO 地址 = 0x20

位	初始值	读/写	描 述
7	0	读/写	启用 PWMG0。0/1：停用/启用
6	-	只读	PWMG0 生成器输出状态。
5	0	读/写	选择 PWMG0 的输出的结果是否反极性。 0/1：停用/启用
4	0	读/写	PWMG0 计数器清零。 写“1”清零 PWMG0 计数，清零 PWMG0 计数后，这个位会自动归 0。
3 – 1	0	读/写	选择 PWMG0 输出引脚： 000：不输出 001：PB5 011：PA0 100：PB4 其他：保留
0	0	读/写	PWMG0 时钟源。0：SYSCLK，1：IHRC

6.29 PWMG0 分频寄存器 (*pwmg0s*)，IO 地址 = 0x21

位	初始值	读/写	描 述
7	0	只写	PWMG0 中断模式。 0：当计数为设定的占空比时产生中断。 1：当计数为 0 产生中断。
6 – 5	0	只写	PWMG0 预分频。 00：÷1 01：÷4 10：÷16 11：÷64
4 – 0	0	只写	PWMG0 分频。

6.30 PWMG0 计数上限高位寄存器 (*pwmg0cubh*), 地址= 0x24

位	初始值	读/写	描 述
7 – 0	0x00	只写	PWMG0 上限寄存器。位[10:3]。

6.31 PWMG0 计数上限低位寄存器(*pwmg0cubl*), 地址= 0x25

位	初始值	读/写	描 述
7 – 6	000	只写	PWMG0 上限寄存器。位[2:1]。
5 – 0	-	-	保留。

6.32 PWMG0 占空比高位寄存器 (*pwmg0dth*), 地址 = 0x22

位	初始值	读/写	描 述
7 – 0	0x00	只写	PWMG0 占空比值。位[10:3]。

6.33 PWMG0 占空比低位寄存器(*pwmg0dtl*), 地址 = 0x23

位	初始值	读/写	描 述
7 – 5	000	只写	PWMG0 占空比值。位[2:0]。
4 – 0	-	-	保留。

注意：PWMG0 占空比寄存器的设置，要先写 pwmg0dtl，后写 pwmg0dth。

6.34 PWMG1 控制寄存器(*pwmg1c*)，IO 地址 = 0x26

位	初始值	读/写	描 述
7	0	读/写	启用 PWMG1。0/1：停用/启用
6	-	只读	PWMG1 生成器输出状态。
5	0	读/写	选择 PWMG1 的输出的结果是否反极性。 0/1：停用/启用
4	0	读/写	PWMG1 计数器清零。 写“1”清零 PWMG0 计数，清零 PWMG1 计数后，这个位会自动归 0。
3 – 1	0	读/写	选择 PWMG1 输出引脚： 000：不输出 001：PB6 011：PA4 100：PB7 其他：保留
0	0	读/写	PWMG1 时钟源。0：SYSCLK，1：IHRC

6.35 PWMG1 分频寄存器(*pwmg1s*), 地址 = 0x27

位	初始值	读/写	描 述
7	0	只写	PWMG1 中断模式。 0：当计数为设定的占空比时产生中断 1：当计数为 0 产生中断
6 – 5	0	只写	PWMG1 时钟预分频。 00：÷1 01：÷4 10：÷16 11：÷64
4 – 0	0	只写	PWMG1 时钟分频。

6.36 PWMG1 计数上限高位寄存器 (*pwmg1cubh*), IO 地址= 0x2a

位	初始值	读/写	描 述
7 - 0	0x00	只写	PWMG1 上限寄存器。位[10:3]。

6.37 PWMG1 计数上限低位寄存器(*pwmg1cubl*), IO 地址= 0x2b

位	初始值	读/写	描述
7 – 6	000	只写	Bit[2:0] PWMG1 上限寄存器。位[2:1]。
5 – 0	-	-	保留。

6.38 PWMG1 占空比高位寄存器(*pwmg1dth*), IO 地址= 0x28

位	初始值	读/写	描述
7 – 0	0x00	只写	PWMG1 占空比值。位[10:3]。

6.39 PWMG1 占空比低位寄存器(*pwmg1dtl*), IO 地址= 0x29

位	初始值	读/写	描述
7 – 5	000	只写	PWMG1 占空比值。位[2:0]。
4 – 0	-	-	保留。

注意：PWMG1 占空比寄存器的设置，要先写 pwmg1dtl，后写 pwmg1dth。

6.40 PWMG2 时钟寄存器(*pwmg2c*), IO 地址 = 0x2c

位	初始值	读/写	描述
7	0	读/写	启用 PWMG2。0 / 1：停用 / 启用
6	-	只读	PWMG2 生成器输出状态
5	0	读/写	选择 PWMG2 的输出的结果是否反极性。 0/1：停用/启用
4	0	读/写	PWMG2 计数器清零。 写“1”清零 PWMG2 计数，清零 PWMG2 计数后，这个位会自动归 0。
3 – 1	0	读/写	选择 PWMG2 输出引脚： 000：不输出 001：PB3 011：PA3 100：PB2 101：PA5（仿真器不支持） Others：保留。
0	0	读/写	PWMG2 时钟源。0：SYSCLK，1：IHRC

6.41 PWMG2 分频寄存器(*pwmg2s*), IO 地址= 0x2d

位	初始值	读/写	描述
7	0	只写	PWMG2 中断模式。 0：当计数为设定的占空比时产生中断 1：当计数为 0 产生中断
6 – 5	0	只写	PWMG2 时钟预分频。 00：÷1 01：÷4 10：÷16 11：÷64
4 – 0	0	只写	PWMG2 时钟分频。

6.42 PWMG2 计数上限高位寄存器(*pwmg2cubh*), IO 地址 = 0x30

位	初始值	读/写	描述
7 - 0	0x00	只写	PWMG2 上限寄存器。位[10:3]。

6.43 PWMG2 计数上限低位寄存器(*pwmg2cubl*), IO 地址= 0x31

位	初始值	读/写	描述
7 – 6	000	只写	PWMG2 占空比值。位[2:1]。
5 – 0	-	-	保留。

6.44 PWMG2 占空比高位寄存器(*pwmg2dth*), IO 地址 = 0x2e

位	初始值	读/写	描述
7 – 0	0x00	只写	PWMG2 占空比值。位[10:3]。

6.45 PWMG2 占空比低位寄存器(*pwmg2dtl*), IO 地址= 0x2f

位	初始值	读/写	描述
7 – 5	000	只写	PWMG2 占空比值。位[2:0]。
4 – 0	-	-	保留。

注意：PWMG2 占空比寄存器的设置，要先写 pwmg2dtl，后写 pwm2dth。

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	即时数据
&	逻辑 AND
|	逻辑 OR
←	移动
^	异或 OR
+	加
–	减
~	NOT（逻辑补数，1 补数）
∓	2 补数
OV	溢出（2 补数系统的运算结果超出范围）
Z	零（如果零运算单元操作的结果是 0，这位设置为 1）
C	进位（Carry）
AC	辅助进位标志（Auxiliary Carry）
IO.n	寄存器的位
M.n	只允许寻址在 address 0~0x3F (0~63)的位置

7.1. 数据传输类指令

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

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 ; // 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
	受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
	应用范例：
	word T16val ; // 定义一个 RAM word
	…
	mov a, 0x34 ;
	mov lb@T16val, a ; // 将 0x34 搬到 T16val（LSB）
	mov a, 0x12 ;
	mov hb@T16val, a ; // 将 0x12 搬到 T16val（MSB）
	stt16 T16val ; // Timer16 初始化 0x1234
	…

idxm **a, index**	使用索引作为 RAM 的地址并将 RAM 的数据读取并载入到累加器。它需要 2T 时间执行这一指令。
	例如： *idxm* a, index;
	结果： a ← [index], index 是用 word 定义。
	受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
	应用范例：
	word RAMIndex ; // 定义一个 RAM 指针
	…
	mov a, 0x5B ; // 指定指针地址（LSB）
	mov lb@RAMIndex, a ; // 将指针存到 RAM（LSB）
	mov a, 0x00 ; // 指定指针地址为 0x00（MSB），在 PMS154C 要为 0
	mov hb@RAMIndex, a ; // 将指针存到 RAM（MSB）
	…
	idxm a, RAMIndex ; // 将 RAM 地址为 0x5B 的数据读取并载入累加器

Idxm **index, a**	使用索引作为 RAM 的地址并将累加器的数据读取并载入到 RAM。它需要 2T 时间执行这一指令。
	例如：*idxm* index, a;
	结果：[index] ← a; index 是以 word 定义.
	受影响的标志位： Z:『不变』， C:『不变』， AC:『不变』， OV:『不变』
	应用范例：
	word RAMIndex ; // 定义一个 RAM 指针
	…
	mov a, 0x5B ; // 指定指针地址（LSB）
	mov lb@RAMIndex, a ; // 将指针存到 RAM（LSB）
	mov a, 0x00 ; //指定指针地址为 0x00（MSB），在 PMS154C 要为 0
	mov hb@RAMIndex, a ; // 将指针存到 RAM（MSB）
	…
	mov a, 0Xa5 ;
	idxm RAMIndex, a ; // 将累加器数据读取并载入地址为 0x5B 的 RAM

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

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

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

7.2. 算术运算类指令

add **a, I**	将立即数据与累加器相加，然后把结果放入累加器。 例如： *add* a, 0x0f ; 结果： a ← a + 0fh 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
add **a, M**	将 RAM 与累加器相加，然后把结果放入累加器。 例如： *add* a, MEM ; 结果： a ← a + MEM 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
add **M, a**	将 RAM 与累加器相加，然后把结果放入 RAM。 例如： *add* MEM, a; 结果： MEM ← a + MEM 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
addc **a, M**	将 RAM、累加器以及进位相加，然后把结果放入累加器。 例如： *addc* a, MEM ; 结果： a ← a + MEM + C 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
addc **M, a**	将 RAM、累加器以及进位相加，然后把结果放入 RAM。 例如： *addc* MEM, a ; 结果： MEM ← a + MEM + C 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
addc **a**	将累加器与进位相加，然后把结果放入累加器。 例如： *addc* a ; 结果： a ← a + C 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
addc **M**	将 RAM 与进位相加，然后把结果放入 RAM。 例如： *addc* MEM ; 结果： MEM ← MEM + C 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
nadd a, M	将累加器的二补码与 RAM 相加，然后把结果放入累加器。 例如： *nadd* a, MEM ; 结果： a ← ∓a + MEM 受影响的标志位： 『Y』Z 『Y』C 『Y』AC 『Y』OV （ICE 不支持）
nadd M, a	将累加器与 RAM 的二补码相加，然后把结果放入 RAM。 例如： *nadd* MEM, a ; 结果： MEM ← ∓MEM + a 受影响的标志位： 『Y』Z 『Y』C 『Y』AC 『Y』OV （ICE 不支持）

sub **a, I**	累加器减立即数据，然后把结果放入累加器。 例如： *sub* a, 0x0f; 结果： a ← a - 0fh (a + [2's complement of 0fh]) 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
sub **a, M**	累加器减 RAM，然后把结果放入累加器。 例如： *sub* a, MEM ; 结果： a ← a - MEM (a + [2's complement of M]) 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
sub **M, a**	RAM 减累加器，然后把结果放入 RAM。 例如： *sub* MEM, a; 结果： MEM ← MEM - a (MEM + [2's complement of a]) 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
subc **a, M**	累加器减 RAM，再减进位，然后把结果放入累加器。 例如： *subc* a, MEM; 结果： a ← a – MEM - C 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
subc **M, a**	RAM 减累加器，再减进位，然后把结果放入 RAM。 例如： *subc* MEM, a ; 结果： MEM ← MEM – a - C 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
subc **a**	累加器减进位，然后把结果放入累加器。 例如： *subc* a; 结果： a ← a - C 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
subc **M**	RAM 减进位，然后把结果放入 RAM。 例如： *subc* MEM; 结果： MEM ← MEM - C 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
inc **M**	RAM 加 1。 例如： *inc* MEM ; 结果： MEM ← MEM + 1 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
dec **M**	RAM 减 1。 例如： *dec* MEM; 结果： MEM ← MEM - 1 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
clear **M**	清除 RAM 为 0。 例如： *clear* MEM ; 结果： MEM ← 0 受影响的标志位： Z:『不变』， C:『不变』， AC:『不变』， OV:『不变』

7.3. 移位运算类指令

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

7.4. 逻辑运算类指令

and **a, I**	累加器和立即数据执行逻辑 AND，然后把结果保存到累加器。 例如：*and* a, 0x0f ; 结果：a ← a & 0fh 受影响的标志位： Z:『受影响』， C:『不变』， AC:『不变』， OV:『不变』
and **a, M**	累加器和 RAM 执行逻辑 AND，然后把结果保存到累加器。 例如：*and* a, RAM10 ; 结果：a ← a & RAM10 受影响的标志位： Z:『受影响』， C:『不变』， AC:『不变』， OV:『不变』
and **M, a**	累加器和 RAM 执行逻辑 AND，然后把结果保存到 RAM。 例如： *and* MEM, a ; 结果： MEM ← a & MEM 受影响的标志位： Z:『受影响』， C:『不变』， AC:『不变』， OV:『不变』
or **a, I**	累加器和立即数据执行逻辑 OR，然后把结果保存到累加器。 例如： *or* a, 0x0f ; 结果： a ← a | 0fh 受影响的标志位： Z:『受影响』， C:『不变』， AC:『不变』， OV:『不变』
or **a, M**	累加器和 RAM 执行逻辑 OR，然后把结果保存到累加器。 例如：*or* a, MEM ; 结果：a ← a | MEM 受影响的标志位： Z:『受影响』， C:『不变』， AC:『不变』， OV:『不变』
or **M, a**	累加器和 RAM 执行逻辑 OR，然后把结果保存到 RAM。 例如： *or* MEM, a ; 结果： MEM ← a | MEM 受影响的标志位： Z:『受影响』， C:『不变』， AC:『不变』， OV:『不变』
xor **a, I**	累加器和立即数据执行逻辑 XOR，然后把结果保存到累加器。 例如： *xor* a, 0x0f ; 结果： a ← a ^ 0fh 受影响的标志位： Z:『受影响』， C:『不变』， AC:『不变』， OV:『不变』
xor **IO, a**	累加器和 IO 寄存器执行逻辑 XOR，然把结果存到 IO 寄存器。 例如： *xor pa, a ;* 结果： pa ← a ^ pa ； // pa 是 port A 资料寄存器 受影响的标志位： Z:『不变』， C:『不变』， AC:『不变』， OV:『不变』
xor **a, M**	累加器和 RAM 执行逻辑 XOR，然后把结果保存到累加器。 例如： *xor* a, MEM ; 结果： a ← a ^ RAM10 受影响的标志位： Z:『受影响』， C:『不变』， AC:『不变』， OV:『不变』
xor **M, a**	累加器和 RAM 执行逻辑 XOR，然后把结果保存到 RAM。 例如： *xor* MEM, a ; 结果： MEM ← a ^ MEM 受影响的标志位： Z:『受影响』， C:『不变』， AC:『不变』， OV:『不变』

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

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

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

neg **M**	RAM 执行 2 补码运算，结果放在 RAM。
	例如： *neg* MEM;
	结果： MEM ← MEM 的 2 补码
	受影响的标志位： Z:『受影响』， C:『不变』， AC:『不变』， OV:『不变』
	应用范例：
	--- 	
	mov a, 0x38 ;
	mov mem, a ; // mem = 0x38
	not mem ; // mem = 0xC8
	--- 	

comp a, M	比较累加器和 RAM。
	例如： *comp* a,MEM;
	结果： 影响标志位
	受影响的标志位： Z:『受影响』， C:『不变』， AC:『不变』， OV:『不变』
	应用范例：
	--- 	
	mov a, 0x38 ;
	mov mem, a ;
	comp a, mem ; // Z = 1
	mov a, 0x42 ;
	mov mem, a ;
	mov a, 0x38 ;
	comp a, mem ; // C = 1
	--- 	
	（ICE 不支持）

comp M, a	比较 RAM 和累加器。 例如： *comp* MEM,a; 结果： 影响标志位 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV （ICE 不支持）

7.5. 位运算类指令

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

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

7.6. 条件运算类指令

ceqsn **a, I**	比较累加器与立即数据，如果是相同的，即跳过下一指令。标志位的改变与 (a ← a - I)相同 例如： *ceqsn* a, 0x55 ; *inc* MEM ; *goto* error ; 结果： 假如 a=0x55, then “goto error”; 否则, “inc MEM”. 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
ceqsn **a, M**	比较累加器与 RAM，如果是相同的，即跳过下一指令。标志位改变与 (a ← a - M)相同 例如：*ceqsn* a, MEM; 结果：假如 a=MEM，跳过下一个指令 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
cneqsn **a, M**	比较累加器与 RAM，如果是不相同的，即跳过下一指令。标志位改变与 (a ← a - M)相同 例如：*cneqsn* 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, I**	比较累加器与立即数据，如果是不相同的，即跳过下一指令。标志位的改变与 (a ← a - I)相同 例如： *cneqsn* 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』
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	停止全部中断。 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, idxm, pcadd, ret, reti*
2 个周期	条件成立	*ceqsn, cneqsn,t0sn, t1sn, dzsn, izsn*
1 个周期	条件不成立	
1 个周期		其他

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	-	-	-	-	*nadd M, a*	Y	Y	Y	Y	*cneqsn a, I*	Y	Y	Y	Y
cneqsn a, M	Y	Y	Y	Y	*comp a, M*	Y	Y	Y	Y	*nadd a, M*	Y	Y	Y	Y
comp M, a	Y	Y	Y	Y	*swapc IO.n*	-	Y	-	-					

7.10. BIT 定义限制

位寻址只能定义在 RAM 区的 0X00 到 0X3F 空间。

8. 代码选项(Code Options)

选项	选择	描述
Security	**Enable**	**OTP 内容加密，程序不允许被读取**
	Disable	OTP 内容不加密，程序可以被读取
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
Boot-up_Time	**Slow**	**请参考第 4.1 节 t_{WUP} 和 t_{SBP}**
	Fast	请参考第 4.1 节 t_{WUP} 和 t_{SBP}
Drive	Low	IO 低驱动和灌电流
	Normal	**IO 正常驱动和灌电流**
LCD2 （请参考 MISC.4）	**Disable**	**VDD/2 偏置电压生成器停用, PB0 PA[0,3,4]是正常 IO 脚**
	PB0_A034	VDD/2 偏置电压生成器启用，如果为输入模式, PB0 PA[0,3,4]为 VDD/2
Comparator_Edge	**All_Edge**	**比较器在上升/下降缘都会触发中断**
	Rising_Edge	比较器在上升缘触发中断
	Falling_Edge	比较器在下降缘触发中断

注：使用黑粗字体的为预置选项（default options）。

9. 特别注意事项

此章节是提醒使用者在使用 PMS154C 时避免一些常犯的错误。

9.1. 使用 IC 时

9.1.1. IO 使用与设定

(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。

◆ 将 IO 设为输入。

◆ 用 PXDIER 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。

(3) PA5 作为输出。

◆ PA5 只能做 Open Drain 输出，输出高时需要外加上拉电阻。

(4) PA5 作为 PRSTB 输入。

◆ 设定 PA5 为输入。

◆ 设定 CLKMD.0=1，使 PA5 为外部 PRSTB 输入脚位。

(5) PA5 作为输入并通过长导线连接至按键或者开关。

◆ 必需在 PA5 与长导线中间串接 >33Ω 电阻。

◆ 应尽量避免使用 PA5 作为输入。

(6) PA7 和 PA6 作为外部晶体振荡器

◆ PA7 和 PA6 设定为输入。

◆ PA7 和 PA6 内部上拉电阻设为关闭。

◆ 用 PADIER 寄存器将 PA6 和 PA7 设为模拟输入。

◆ EOSCR 寄存器位[6:5]选择对应的晶体振荡器频率：

✧ 01 ： 低频，例如：32KHz

✧ 10 ： 中频，例如：455KHz、1MHz

✧ 11 ： 高频，例如：4MHz

◆ 设置 EOSCR.7 =1 启用晶体振荡器。

◆ 从 IHRC 或 ILRC 切换到 EOSC，要先确认 EOSC 已经稳定振荡。

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

9.1.2. 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。

9.1.3. 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。
```

9.1.4. 看门狗

看门狗默认为开，但程序执行 ADJUST_IC 时，会将看门狗关闭，若要使用看门狗，需重新配置打开。当 ILRC 关闭时，看门狗也会失效。

9.1.5. TIMER16 溢出时间

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

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

9.1.6. IHRC 校准

(1) IHRC 的校正操作是于使用 writer 烧录时进行的。

(2) 因为 IC 的塑封材料（不论是封装用的或 COB 用的黑胶）的特牲，是会对 IHRC 的频率有一定影响。所以如果用户是在 IC 盖上塑封材料前，就对 IC 进行烧录，及后再封上盖上塑封材料的，则可能造成 IHRC 的特性偏移超出规格的情况。正常情况是频率会变慢一些。

(3) 此种情况通常发生在用户是使用 COB 封装，或者是委托我司进行晶圆代烧（QTP）时。此情况下我司将不对频率的超出规格的情况负责。

(4) 用户可按自身经验进行一些补偿性调整，例如把 IHRC 的目标频率调高 0.5%-1%左右，令封装后 IC 的 IHRC 频率更接近目标值。

9.1.7. LVR

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

下面是工作频率、电源电压和 LVR 水平设定的建议：

系统时钟	VDD	LVR
2MHz	≧ 1.8V	≧ 1.8V
4MHz	≧ 2.5V	≧ 2.5V
8MHz	≧ 3.5V	≧ 3.5V

表 9：LVR 设置参考

(1) 只有当 IC 正常起动后，设定 LVR（1.8V ~ 4.0V）才会有效。

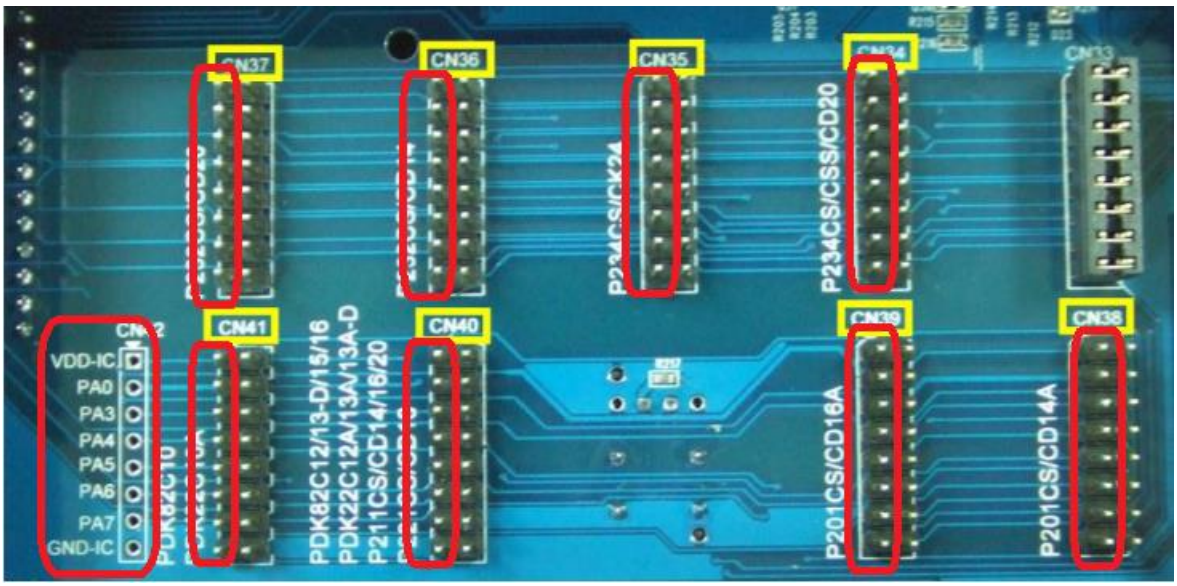
(2) 可以设定寄存器 MISC.2 为 1 将 LVR 关闭，但此时应确保 V_{DD} 在 chip 最低工作电压以上，否则 IC 可能工作不正常。

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

9.1.8. 烧录方法

PMS154C 的烧录脚为 PA3，PA4，PA5，PA6，V_{DD}，GND 这 6 只引脚。

在 3S-P-002 烧录器上，可以使用烧录器背后的 jumper 放在 CN39 的位置。如果为 16 Pin 包装，正面的接脚不用移位；14 Pin 包装，正面的接脚往下空移一格；10 Pin 包装，往下空移三格；8 Pin 包装，往下空移四格。如为其他包装，可以自行跳接烧录接脚，烧录器背后的 Jumper，所有左侧的讯号都是一致的，就如左下角说明文字一样，分别为 V_{DD}，PA0（不需要），PA3，PA4，PA5，PA6，PA7（不需要），GND。



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

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

(1) PA5 （V_{PP}）可能高于 11V。

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

(3) 其他烧录引脚（GND 除外）的电位与 V_{DD} 相同。

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

9.2. 使用 ICE 时

仿真 PMS154C 功能时注意事项：

(1) 建议使用 5S-I-S01/2(B)仿真器。

(2) 用 5S-I-S01/2(B)仿真时，请注意以下几点：

- 用 5S-I-S01/2(B)仿真时，不支持 NADD/COMP 指令
- 用 5S-I-S01/2(B)仿真时，不支持系统时钟 SYSCLK=ILRC/16
- 用 5S-I-S01/2(B)仿真时，不支持 misc.4 动态设定（只能固定为 0 或 1）
- 用 5S-I-S01/2(B)仿真时，不支持 TM2 和 TM3 的 GPCRS 功能
- 用 5S-I-S01/2(B)仿真时，当 GPCS 选择 Output 到 PA0 输出时，PA3 输出功能也会受影响
- 5S-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
- 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。
- 用 5S-I-S01/2(B)仿真时，在 timer2/timer3 定周期模式下，改变 tm2ct/tm3ct 的值会影响占空比，对于实际 IC 则不会。
- 用 5S-I-S01/2(B)仿真时，当快速唤醒被使能，看门狗溢出复位时间会变得很短。实际 IC 没有影响。
- 掉电指令 Stopsys 不支持比较器唤醒功能，使用 5S-I-S01/2(B)仿真时，在进掉电模式前需注意比较器使能应设为关闭状态，若使能状态为开启，将有可能发生比较器误唤醒。
- 快速唤醒时间和使用 5S-I-S01/2(B)仿真不同（ICE：128 SysClk，PMS154C：45 ILRC）
- 看门狗溢出时间和使用 5S-I-S01/2(B)仿真不同，如下：

WDT 溢出时间	5S-I-S01/2(B)	PMS154C
misc[1:0]=00	2048 * T_{ILRC}	8K * T_{ILRC}
misc[1:0]=01	4096 * T_{ILRC}	16K * T_{ILRC}
misc[1:0]=10	16384 * T_{ILRC}	64K * T_{ILRC}
misc[1:0]=11	256 * T_{ILRC}	256K * T_{ILRC}