PMB180 / PMB180B

8 位 OTP 型单片机带充电

数据手册

第 0.06 版

2024 年 11 月 19 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。

应广科技为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
0.05	2024/09/25	1. 更新特性的描述 2. 更新 6.35 与 9.45 的章节描述 3. 新增 9.1.10 章节 4. 其他已知细节错误修正
0.06	2024/11/19	1. 新增 PMB180B 2. 新增 PMB180 和 PMB180B 差异表 3. 更新寄存器 chg_temp 位 1 功能说明

使用警告

在使用 IC 前，请务必认真阅读 PMB180(B)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=175&kw=PMB180(B)

（下列图示仅供参考，依官网为主。）

- 通用系列
- 不建议使用于AC 阻容降压供电或有高EFT要求之应用
- 工作温度范围： -40°C ~ 85°C

Feature Documents Software &Tools Application Note

内容	說明	中文下载	英文下载
APN002	过压保护应用须知		
APN003	IO输出引脚连接长导线时的应用须知		
APN004	半自动烧录机台使用需知		
APN007	设置LVR时的使用须知		
APN011	半自动烧录机台提高烧录稳定性		
APN019	E-PAD 产品的PCB布局指南		

PMB180 和 PMB180B 主要差异表

项目	功能	PMB180	PMB180B
1	寄存器 chg_temp 位 1	OTP_100 (新版 IDE 不支持使用)	新增 充电完成指示位 (充电电流小于 1/10，关闭充电)
2	Vcc_Pin 浮空微漏电	Vcc_Pin 不可浮空要加电阻电容 (请参考 APN-021)	无此问题 Vcc_Pin 漏电问题已修正

1. 单片机特点

1.1. 特性

- 通用系列
- 工作温度范围：-40°C ~ 85°C

1.2. 系统特性

- 1.25KW OTP 程序内存
- 64Bytes 数据存储器
- 一个硬件 16 位计数器
- 一个 8 位硬件定时器（可以 PWM 模式输出，PWM 分辨率可以位 6/7/8 位）
- 一组 3 连套 11 位硬件 PWM 生成器
- 一个硬件比较器
- 7 个 IO 引脚并带有上拉/下拉电阻选项
- 每个 IO 引脚都具有系统唤醒功能
- 时钟源：内部高频 RC 振荡器(IHRC)，内部低频 RC 振荡器(ILRC)
- 对所有带有唤醒功能的 IO，都支持两种可选择的唤醒速度：正常唤醒和快速唤醒
- LVR 复位电压：从 1.8V 到 4.5V
- 2 个外部中断引脚：PA0, PA4
- Bandgap 电路提供 1.20V 参考电压
- 支持低功耗(NILRC)定时唤醒 stopsys
- VCC 输入电压范围：4.3V ~ 6.5V
- 0.5A 最大充电电流； 内部可设定 50mA/100mA/200mA/250mA/300mA/350mA/400mA/500mA
- 无需外接 MOSFET，检测电阻，反向二级管等组件
- 恒流/恒压操作模式且具有热保护功能
- 具有 1% 精度的默认充电电压
- 自动再充电
- C/10 充电中止
- 2.9V 涓流充电电压
- 充电模式待机功耗 57uA (VCC)

1.3. CPU 特性

- 单一处理单元工作模式
- 提供 86 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. 订购/封装信息

- PMB180(B)-ES08: ESOP8 (150mil)
- PMB180(B)-EY10: ESSOP10 (150mil)

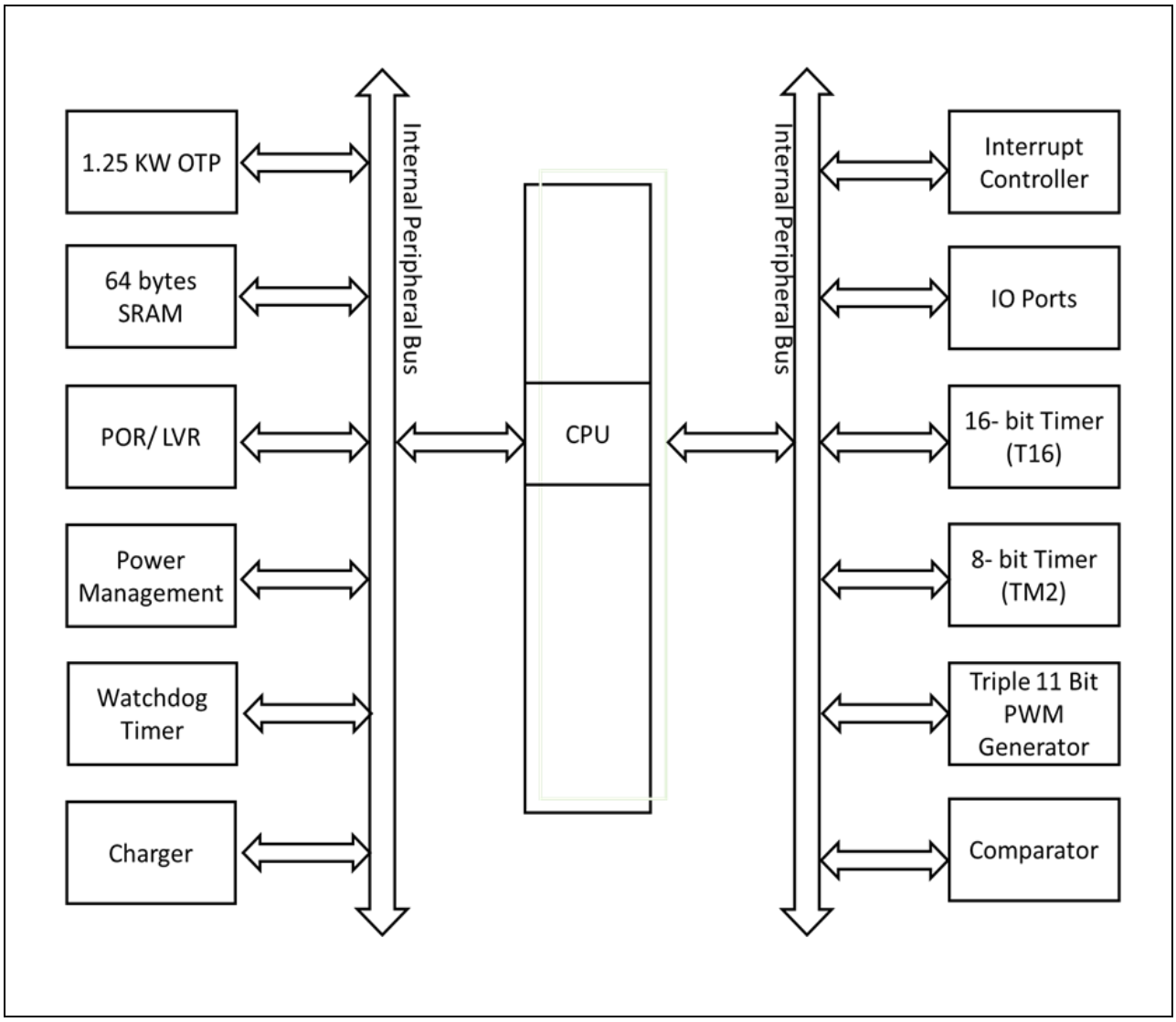
- 有关封装尺寸的信息，请参阅官方网站文件：“封装信息”

2. 系统概述和方框图

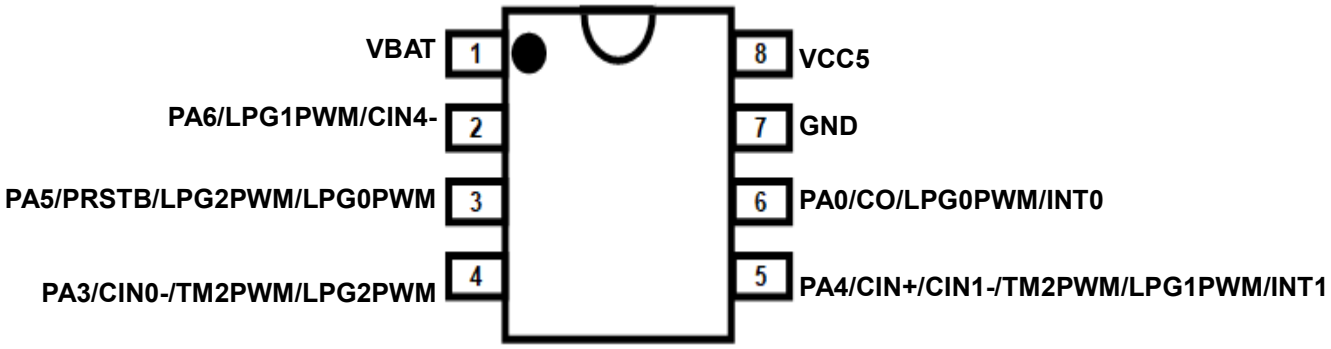
PMB180(B)系列是一款单 IO 型，完全静态的，以 OTP 为程序基础的 CMOS 8-bit 微处理器。它运用 RISC 的架构并大部分的指令执行都是一个指令周期的，只有少部分处理间接寻址指令需要两个指令周期。

PMB180(B)内置 1.25KW OTP 数据存储器以及 64 字节数据存储器，一个硬件比较器，用于比较两个引脚之间的信号或内部参考电压 Vinternal-R 或内部带隙参考电压 Bandgap。PMB180(B)还提供三个硬件定时器：一个 16 位定时器、一个 8 位定时器（可以 PWM 模式输出），和一组 3 连套 11 位 PWM 定时器/生成器（LPWMG0、LPWMG1 和 LPWMG2）。

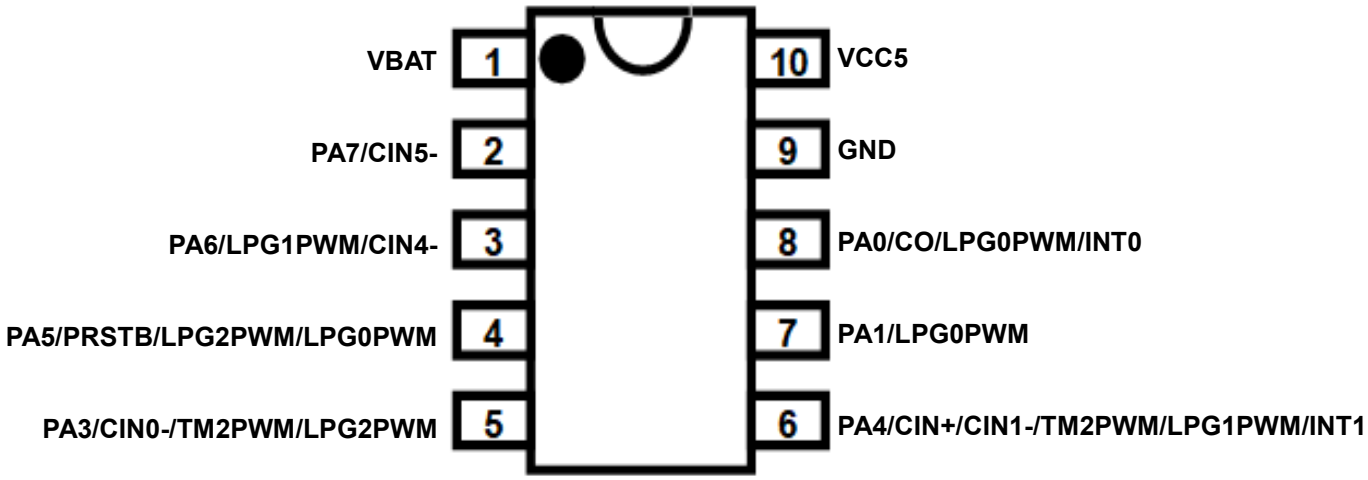
PMB180(B)中的充电器是一种基于单一锂离子电池的恒流/恒压充电器，设计用于 USB 电源规格。当结温升高时，内部模块调节电流，以在高功率或高环境温度下运行时保护设备。充电电压固定在 4.2V，充电电流可通过寄存器编程限制，无需外部电阻器，电流可达 500mA。当流向电池的电流低于编程值的 1/10 时，充电循环自动终止。如果卸下外部适配器，充电器将关闭，电池到设备的电流将小于 2μA。



3. 引脚分配及功能说明



PMB180(B)-ES08 (ESOP8-150mil)



PMB180(B)-EY10 (ESSOP10-150mil)

引脚名称	引脚& 缓存器类型	描述
PA7 / CIN5-	IO ST / CMOS	此引脚可以用作： (1) 端口 A 位 7，并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) 比较器负输入源 5。 如果此引脚用于晶体振荡器，则必须将 ***padier*** 位 7 设为“0”，以避免漏电流。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 7 为“0”时，唤醒功能是被关闭的。
PA6 / LPG1PWM / CIN4-	IO ST / CMOS	此引脚可以用作： (1) 端口 A 位 6，并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) 11 位 PWM 生成器 PWMG1 的输出。 (3) 比较器负输入源 4。 如果此引脚用于晶体振荡器，则必须将 ***padier*** 位 6 设为“0”，以避免漏电流。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 6 为“0”时，唤醒功能是被关闭的。
PA5 / PRSTB / LPG2PWM / LPG0PWM	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输出，弱上拉/下拉电阻模式。 (2) 硬件复位。 (3) 11 位 PWM 生成器 PWMG2 的输出。 (4) 11 位 PWM 生成器 PWMG0 的输出。 如果此引脚用于晶体振荡器，则必须将 ***padier*** 位 5 设为“0”，以避免漏电流。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+ / CIN1- / TM2PWM/ PG1PWM/ INT1	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 A 位 4。可程序设计设定为输入或输出，弱上拉/下拉电阻模式。 (2) 比较器的正输入源。 (3) 比较器的负输入源 1。 (4) Timer2 的 PWM 输出。 (5) 11 位 PWM 生成器 PWMG1 的输出。 (6) 外部中断源 1，通过寄存器可以设置上升沿和下降沿响应中断服务请求。 当用做模拟输入功能时，为减少漏电流，请用 ***padier*** 寄存器位 4 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& 缓存器类型	描述
PA3 / CIN0- / TM2PWM / LPG2PWM	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 A 位 3。可程序设计设定为输入或输出，弱上拉/下拉电阻模式。 (2) 比较器的负输入源 0。 (3) Timer2 的 PWM 输出。 (4) 11 位 PWM 生成器 PWMG2 的输出。 当用做模拟输入功能时，为减少漏电流，请用 ***padier*** 寄存器位 3 关闭数字输入功能。 ***padier*** 寄存器位 3 可以设置为“0”关闭其数字输入；通过切换此引脚禁用断电唤醒。
PA1 / LPG0PWM	IO ST / CMOS	此引脚可以用作： (1) 端口 A 位 0。可程序设计设定为输入或输出，弱上拉/下拉电阻模式。 (2) 11 位 PWM 生成器 PWMG0 的输出。 ***padier*** 寄存器的位 0 可以设为“0” 停用睡眠中唤醒系统的功能。 当用做模拟输入功能时，为减少漏电流，请用 ***padier*** 寄存器位 3 关闭其数字输入功能。 可将 ***padier*** 寄存器的位 1 设置为“0”关闭数字输入；通过切换此引脚禁用断电唤醒。
PA0 / CO / LPG0PWM / INT0	IO ST / CMOS	此引脚可以用作： (1) 端口 A 位 0，并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) 比较器输出。 (3) 11 位 PWM 生成器 LPWMG0 的输出。 (4) 外部中断源 0。通过寄存器可以设置上升沿和下降沿响应中断服务请求。 ***padier*** 寄存器的位 0 可以设为“0”，通过切换此引脚禁用断电唤醒。
VBAT	VBAT	电池的正电源。MCU 的正电源。
VCC5	VCC	当连接 VCC 时，可以为电池充电。
GND	GND	接地。
注意：IO：输入/输出；**ST**：施密特触发器输入；**OD**：开漏；**Analog**：模拟输入引脚； **CMOS**：CMOS 电压基准位		

4. 器件电器特性

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 = -40°C ~ 85°C，V_{BAT} =5.0V，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(Ta=25°C)
V_{BAT}	工作电压	1.8#	5.0	5.5	V	# 受限于 LVR 公差
LVR%	低电压复位公差	-5		5	%	
f_{SYS}	系统时钟 （CLK）* = IHRC/2 IHRC/4 IHRC/8 ILRC	 0 0 0	 100K	 8M 4M 2M	Hz	 V_{BAT} ≧ 2.7V V_{BAT}≧ 2.2V V_{BAT} ≧ 1.8V V_{BAT} =3.0V
V_{POR}	复位电压		1.8*		V	# 受限于 LVR 公差
I_{OP}	工作电流		0.55 85		mA uA	f_{SYS}=IHRC/16=1MIPS@5.0V f_{SYS}=ILRC=90KHz@5.0V
I_{PD}	掉电模式消耗电流 （使用 ***stopsys*** 命令）		1 0.6		uA uA	f_{SYS}= 0Hz, V_{BAT} =5.0V f_{SYS}= 0Hz, V_{BAT} =3.3V
I_{PS}	省电模式消耗电流 （使用 ***stopexe*** 命令） *停用 IHRC		3		uA	V_{BAT} = 5.0V; f_{SYS} = ILRC 仅使用 ILRC 模式条件下
V_{IL}	输入低电压	0		0.2 V_{BAT}	V	
V_{IH}	输入高电压	0.7 V_{BAT}		V_{BAT}	V	
I_{OL}	IO 输出灌电流					
	全部 IO		18		mA	V_{BAT} = 5.0V, V_{OL} = 4.5V
I_{OH}	IO 输出驱动电流					
	全部 IO		-16		mA	V_{BAT} = 5.0V, V_{OH} = 4.5V
V_{IN}	输入电压	-0.3		V_{BAT}+0.3	V	
$I_{INJ (PIN)}$	引脚输入电流			1	mA	V_{BAT} + 0.3≧V_{IN}≧ -0.3
R_{PH}	上拉电阻		91		KΩ	V_{BAT} = 5.0V
R_{PL}	下拉电阻		91		KΩ	V_{BAT} = 5.0V
V_{BG}	Bandgap 参考电压	1.145*	1.20*	1.255*	V	V_{BAT} = 2.2V ~ 5.5V -40°C < Ta <85°C*
f_{IHRC}	校准后 IHRC 频率 *	15.76*	16*	16.24*	MHz	25oC, VBAT =2.2V~5.5V
		15.20*	16*	16.80*		VBAT =2.2V~5.5V, -40oC <Ta<85oC*
		13.60*	16*	18.40*		VBAT =2.0V~5.5V, -40oC <Ta<85oC
f_{ILRC}	ILRC 频率 *		100		KHz	V_{BAT} = 5.0V
f_{NILRC}	NILRC 频率*		17		KHz	V_{BAT} = 5.0V
t_{INT}	中断脉冲宽度	30			ns	V_{BAT} = 5.0V

符号	描述	最小值	典型值	最大值	单位	条件(Ta=25°C)
V_{DR}	RAM 数据保持电压*	1.5			V	in stop mode
t_{WDT}	看门狗超时溢出时间		8k		T_{ILRC}	misc[1:0]=00（默认）
			16k			misc[1:0]=01
			64k			misc[1:0]=10
			256k			misc[1:0]=11
t_{WUP}	快速唤醒时间		45		T_{ILRC}	T_{ILRC} 是 ILRC 时钟周期
	慢速唤醒时间		3000			
t_{SBP}	系统开机时间		30		ms	V_{BAT} =5V
t_{RST}	外部复位脉冲宽度	120			us	@ V_{BAT} =5V
CPos	比较器偏移*		±10	±20	mV	
CPcm	比较器输入普通模式*	0		V_{BAT}-1.5	V	
CPspt	比较器响应时间		100	500	ns	上升和下降
CPmc	改变比较器模式的稳定时间		2.5	7.5	us	
CPcs	比较器消耗电流		20		uA	V_{BAT} = 3.3V
VCC	充电器输入电源电压	4.3	5	6.5	V	
I_{VCC}	充电器输入电源电流		200	500	μA	充电模式
			57		μA	待机模式
			38		μA	关机模式
			0		μA	休眠模式
I_{CCM}	恒流模式充电电流	-15%	50	+15%	mA	@VCC=5V
			100		mA	
			200		mA	
			250		mA	
			300		mA	
			350		mA	
			400		mA	
			500		mA	
I_{TRKL}	涓流充电电流		1/10		I_{CCI}	V_{BAT} <V_{TRKL}
V_{FLOAT}	浮动电压	-1%	4.2	+1%	V	@VCC=5V
V_{TRKL}	涓流充电阈值电压		2.9		V	V_{BAT} rising

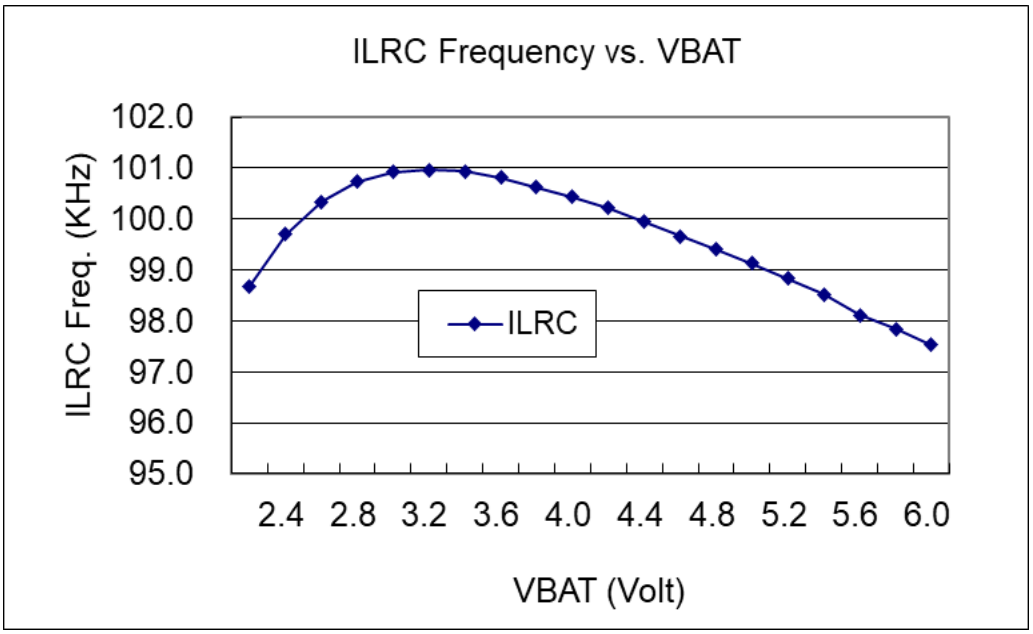
符号	描述	最小值	典型值	最大值	单位	条件(Ta=25°C)
V_{TRHYS}	涓流电压滞后电压		100		mV	
V_{UV}	欠压锁定阈值		3.7		V	VCC rising
V_{UVHYS}	欠压锁定滞后		200		mV	
V_{ASD}	锁定阈值电压		100		mV	VCC rising
			30		mV	VCC falling
t_{RECHA}	充电比较器过滤器时间		2		mS	V_{BAT} 高到低
t_{TERM}	终端比较器滤波器时间		1		mS	I_{CCM} 少于 1/10
I_{TERM}	C/10 终止电流阈值		0.1		mA	
$\triangle V_{RECHA}$	重新充电蓄电池阈值电压		150		mV	
T_{LIM}	恒温模式下的结点温度		90		℃	

* 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。

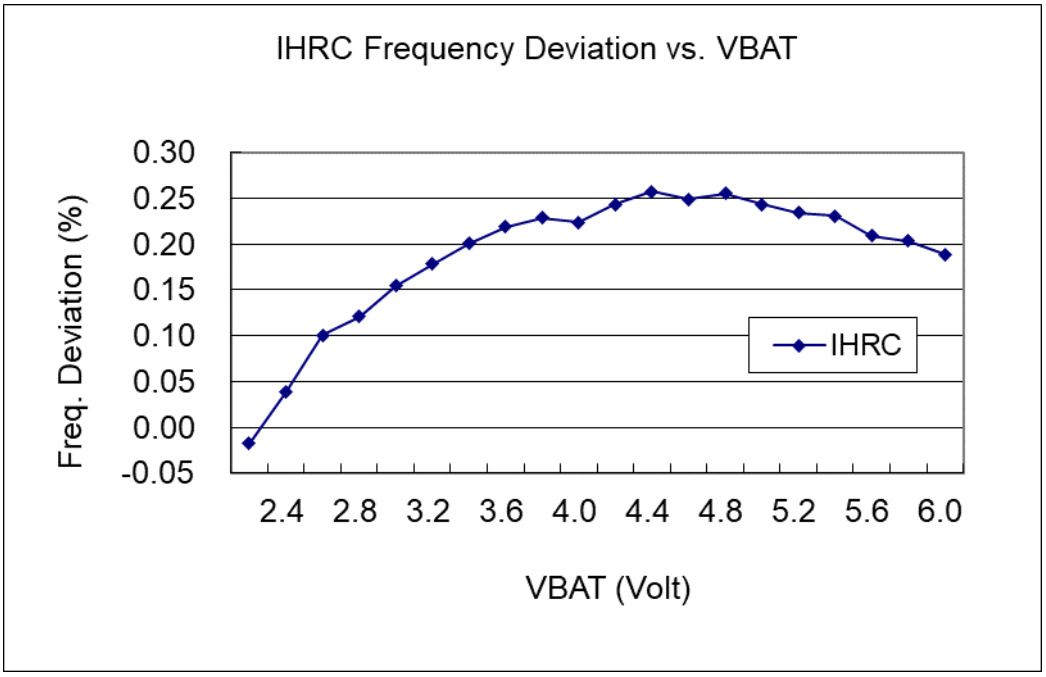
4.2. 绝对最大值范围

- 电源电压 ..... 1.8V ~ 5.5V（最大值：5.5V）
  *最大电压不能超过 5.5V，否则会损坏 IC。
- 输入电压 ..... -0.3V ~ V_{BAT} + 0.3V
- 工作温度 ..... -40°C ~ 85°C
- 存储温度 ..... -50°C ~ 125°C
- 结点温度 ..... 150°C

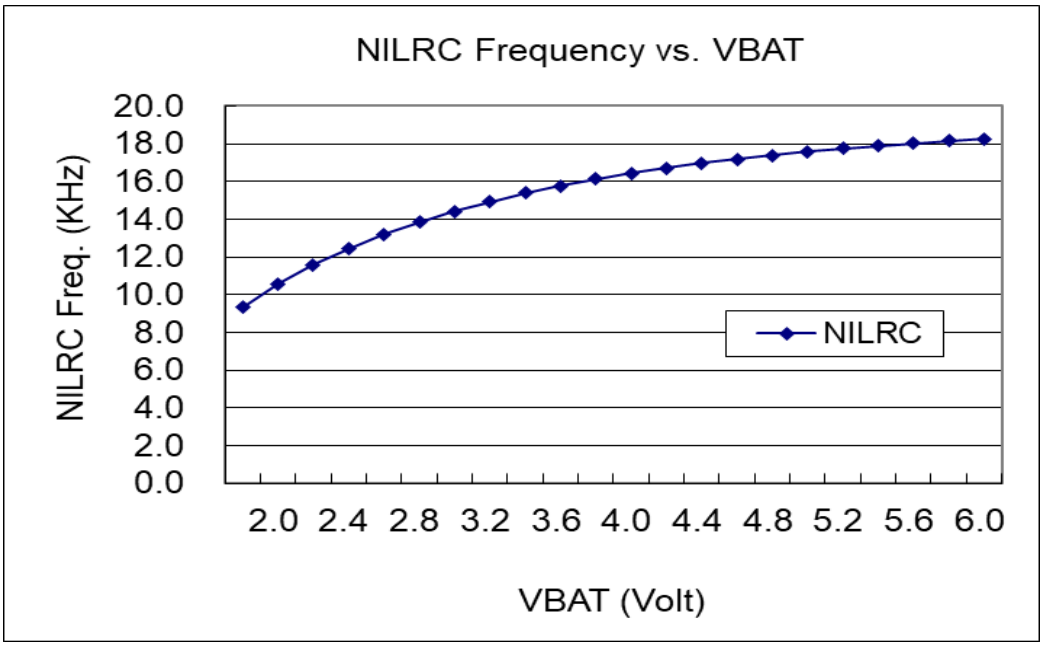
4.3. ILRC 频率与 V_{BAT} 关系曲线图



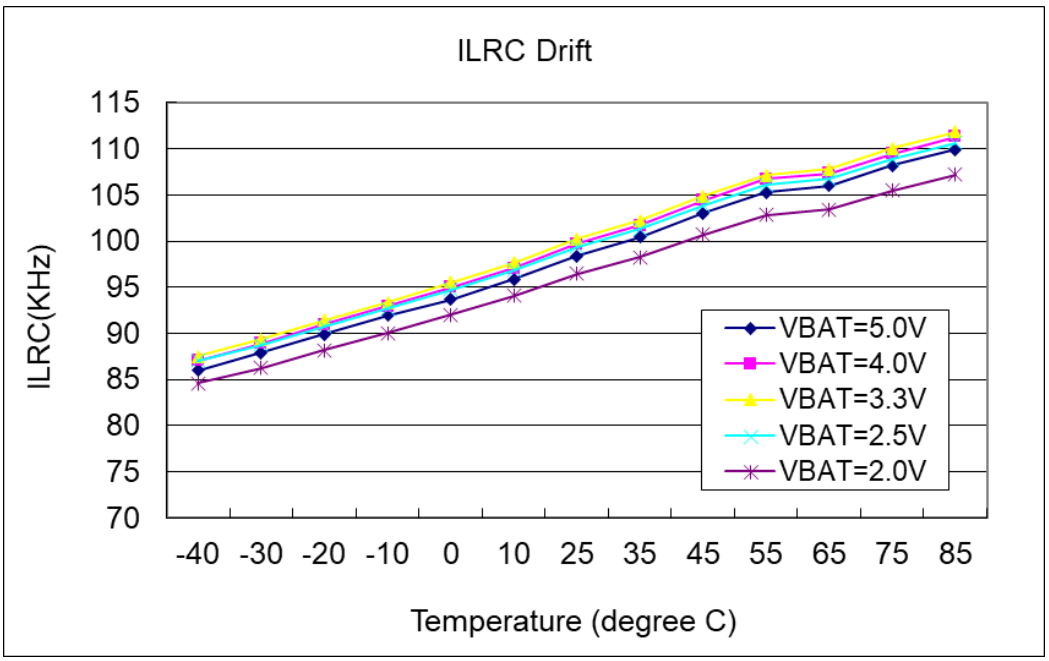
4.4. IHRC 频率与 V_{BAT} 关系曲线图（校准到 16MHz）



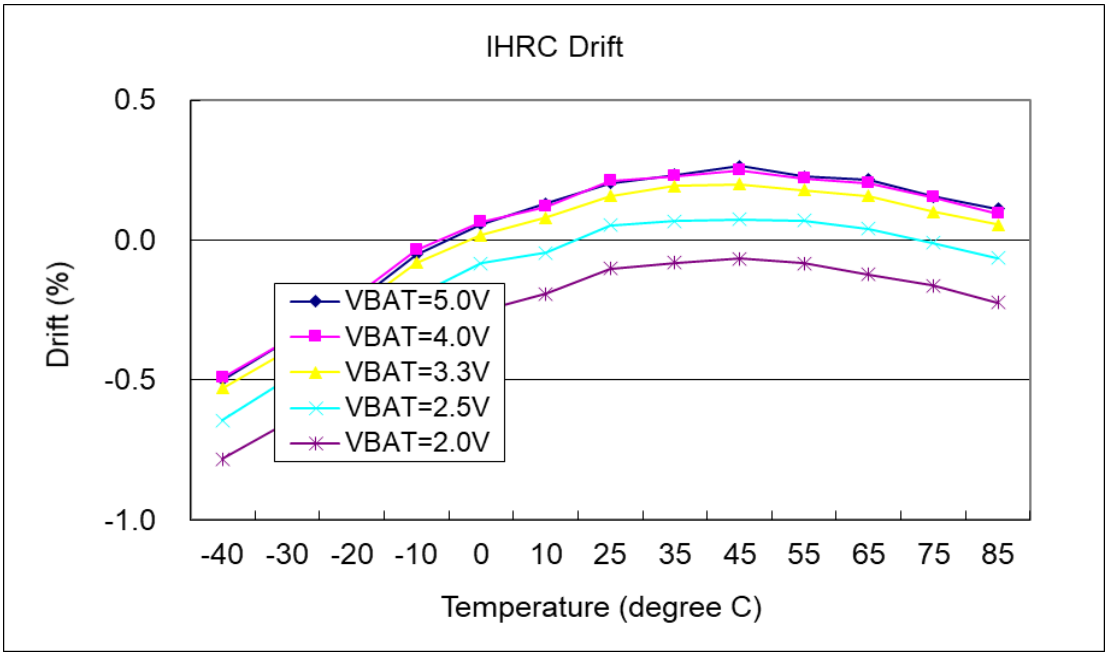
4.5. NILRC 频率与 V_{BAT} 关系曲线图



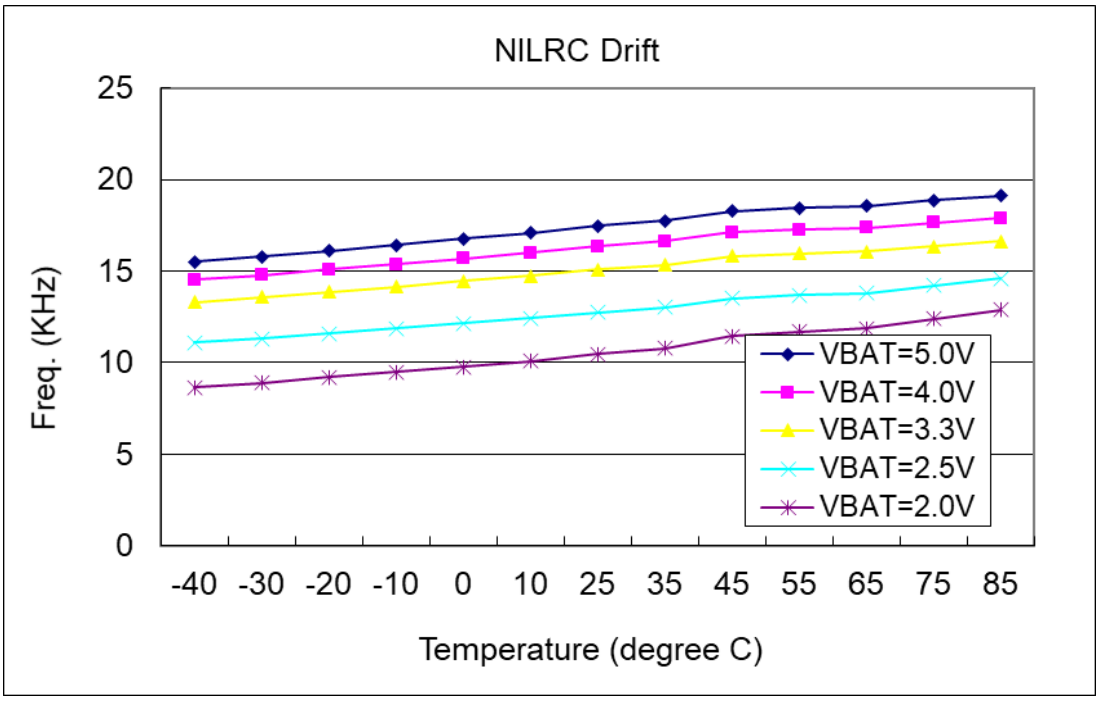
4.6. ILRC 频率与温度关系曲线图



4.7. IHRC 频率与温度关系曲线图（校准到 16MHz）



4.8. NILRC 频率与温度关系曲线图

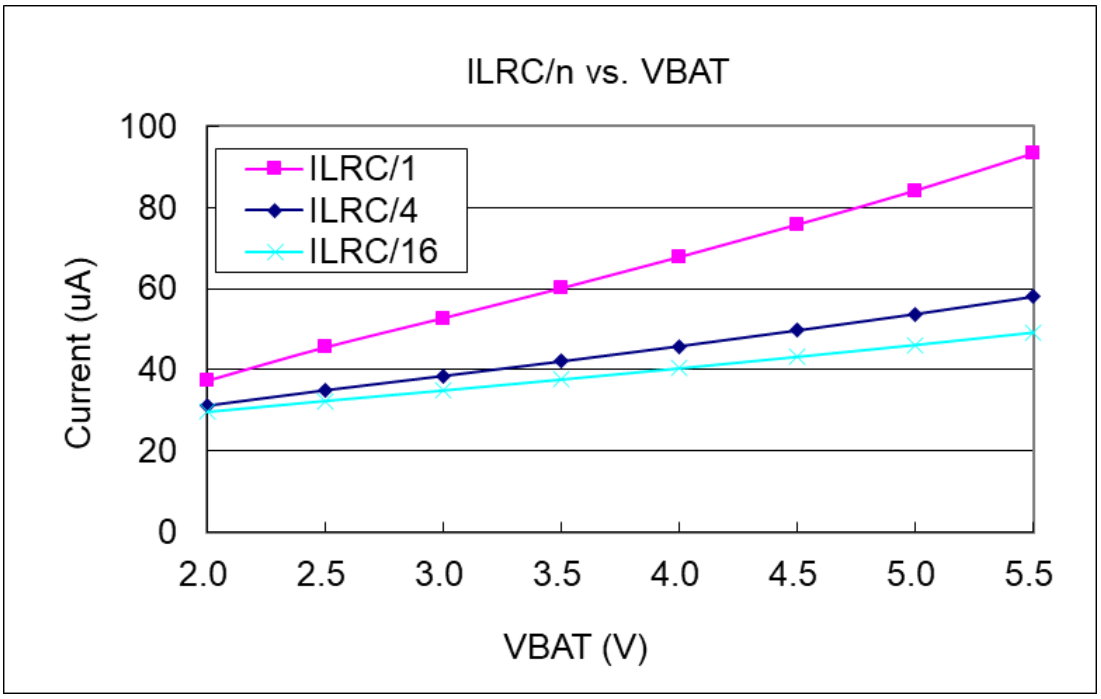


4.9. 工作电流 vs. V_{BAT} 与系统时钟 = ILRC/n 关系曲线图

条件：

ON: Bandgap，LVR，ILRC; OFF: IHRC, EOSC, T16, TM2;

IO: PA0:0.5Hz 输出翻转不悬空，其他：输入且 IO 引脚不悬空。

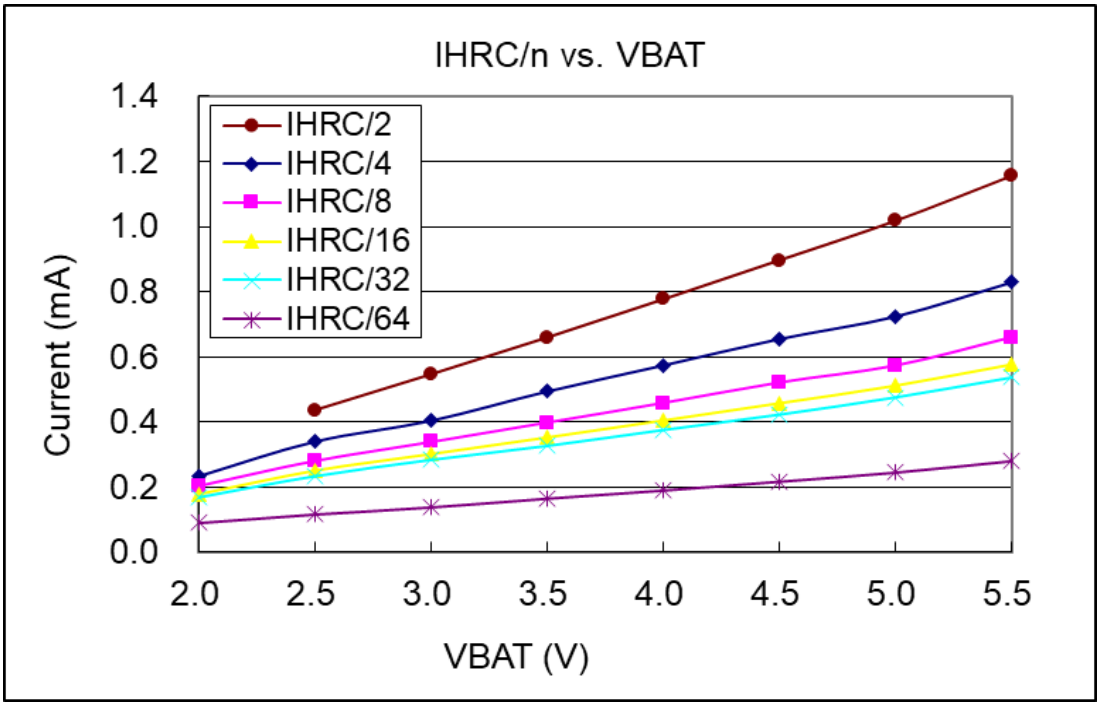


4.10. 工作电流 vs. V_{BAT} 与系统时钟 = IHRC/n 关系曲线图

条件：

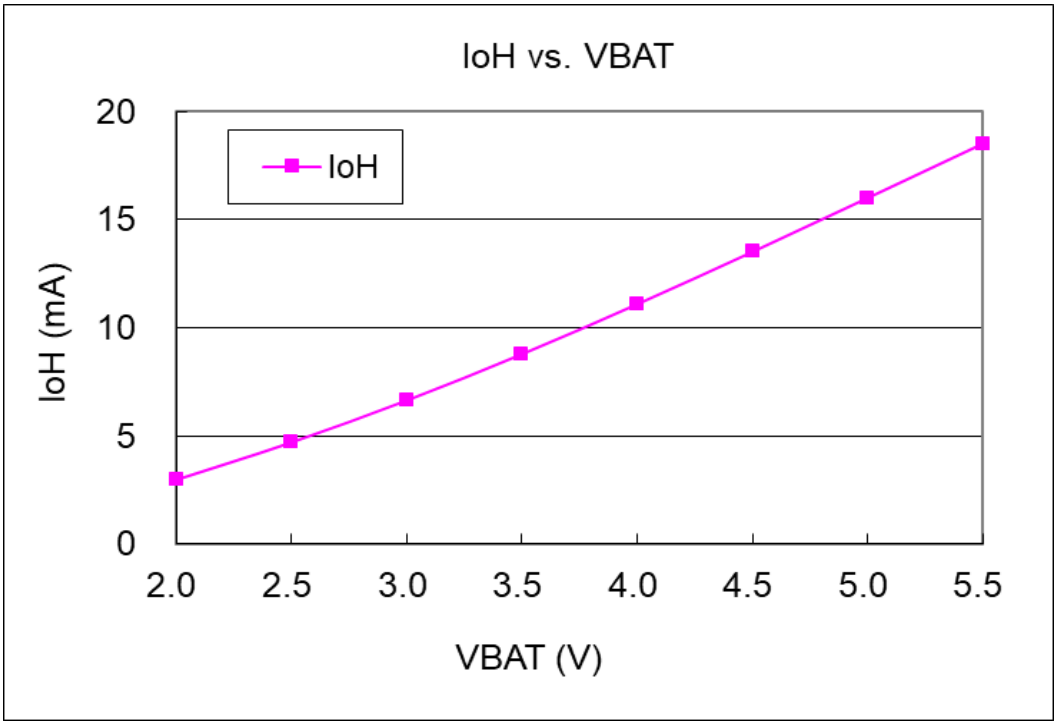
ON: Bandgap，LVR，IHRC; OFF: ILRC, EOSC, T16, TM2;

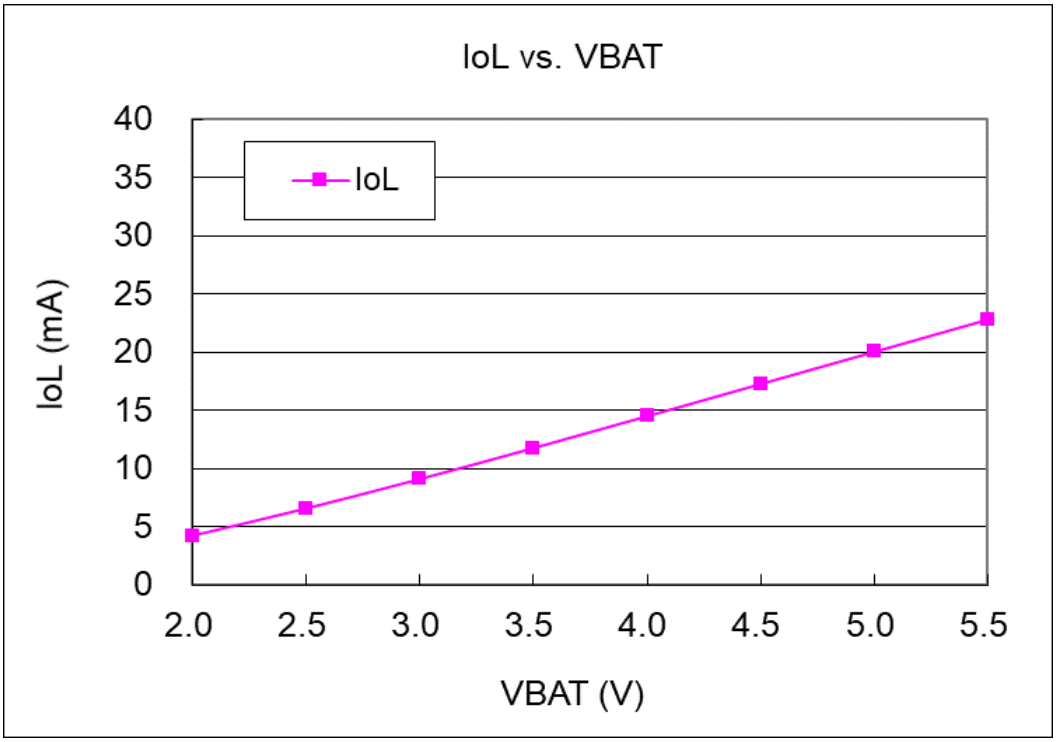
IO: PA0:0.5Hz 输出翻转不悬空，其他：输入且 IO 引脚不悬空。



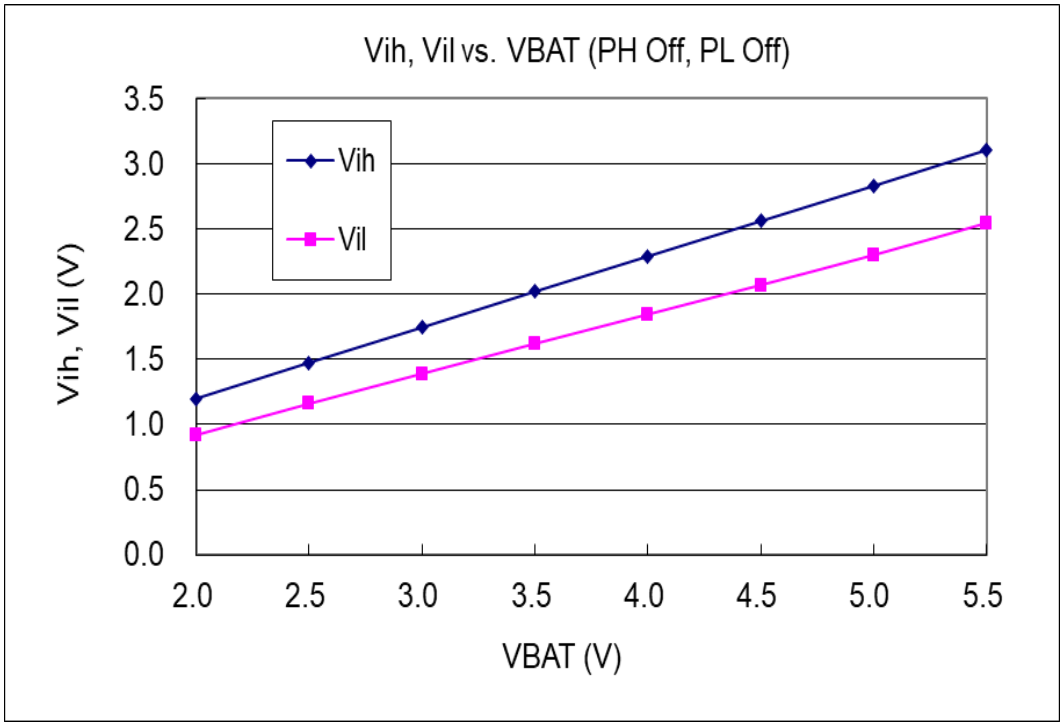
4.11. IO 引脚输出的驱动电流(I_{OH})与灌电流(I_{OL})曲线图

(VOH=0.9* V_{BAT}, VOL=0.1* V_{BAT})

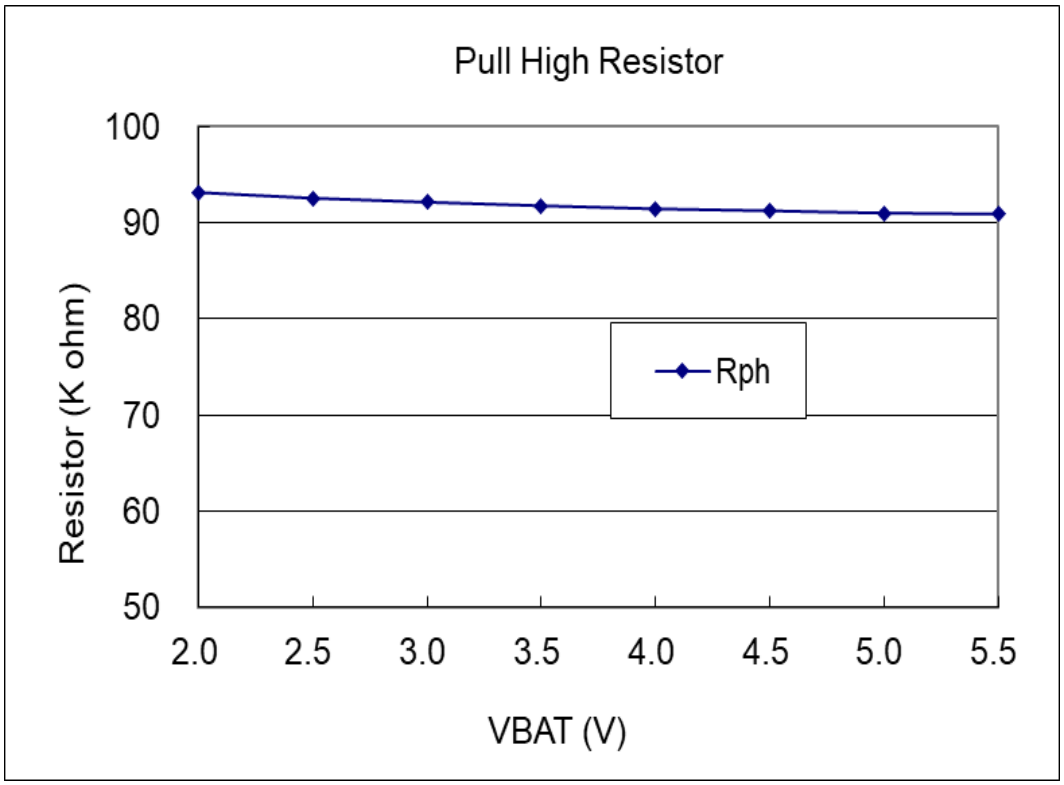


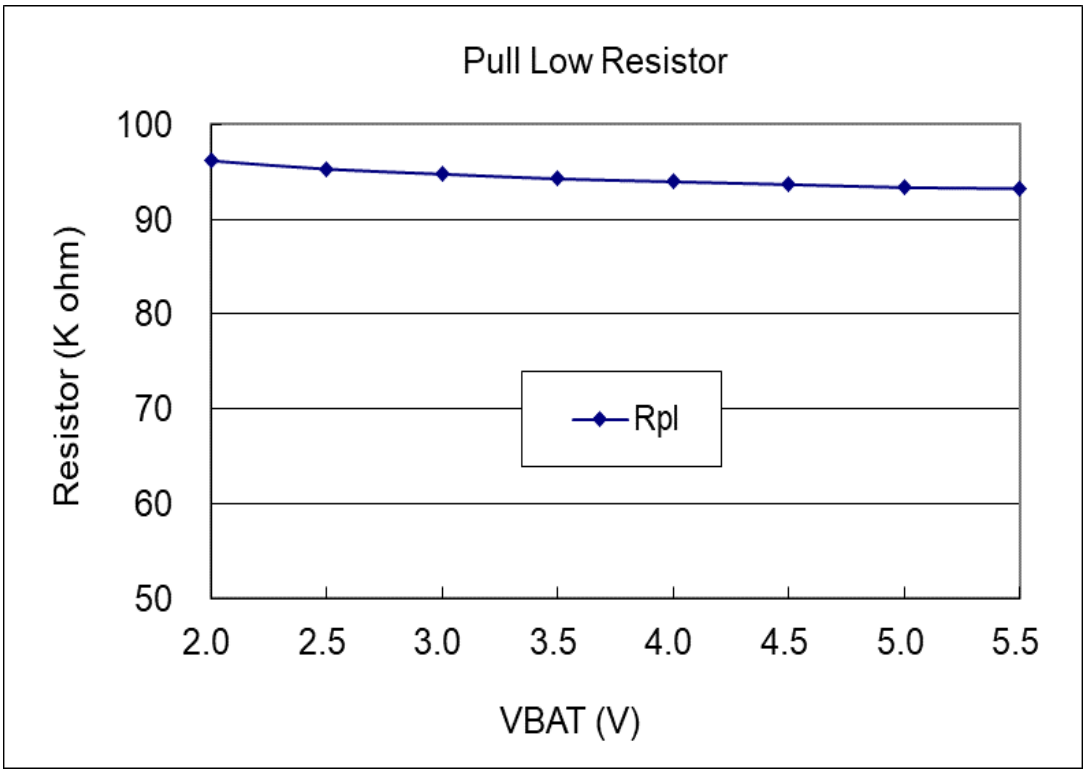


4.12. IO 引脚输入高/低阈值电压曲线图 (V_{IH}/V_{IL})

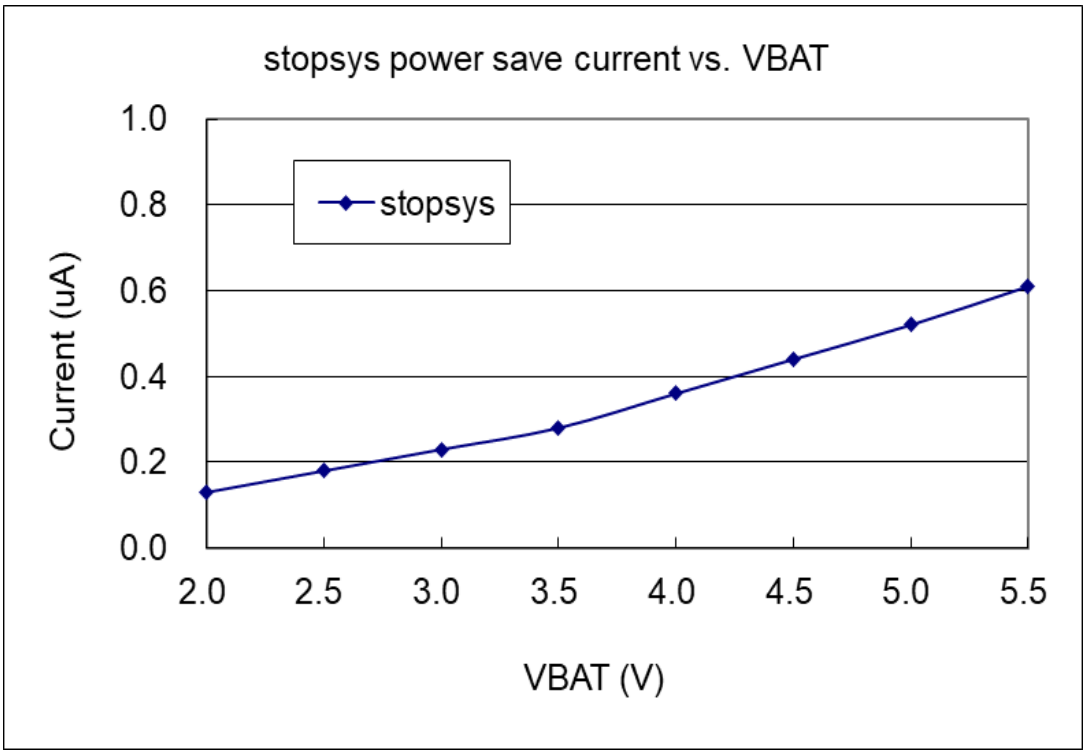


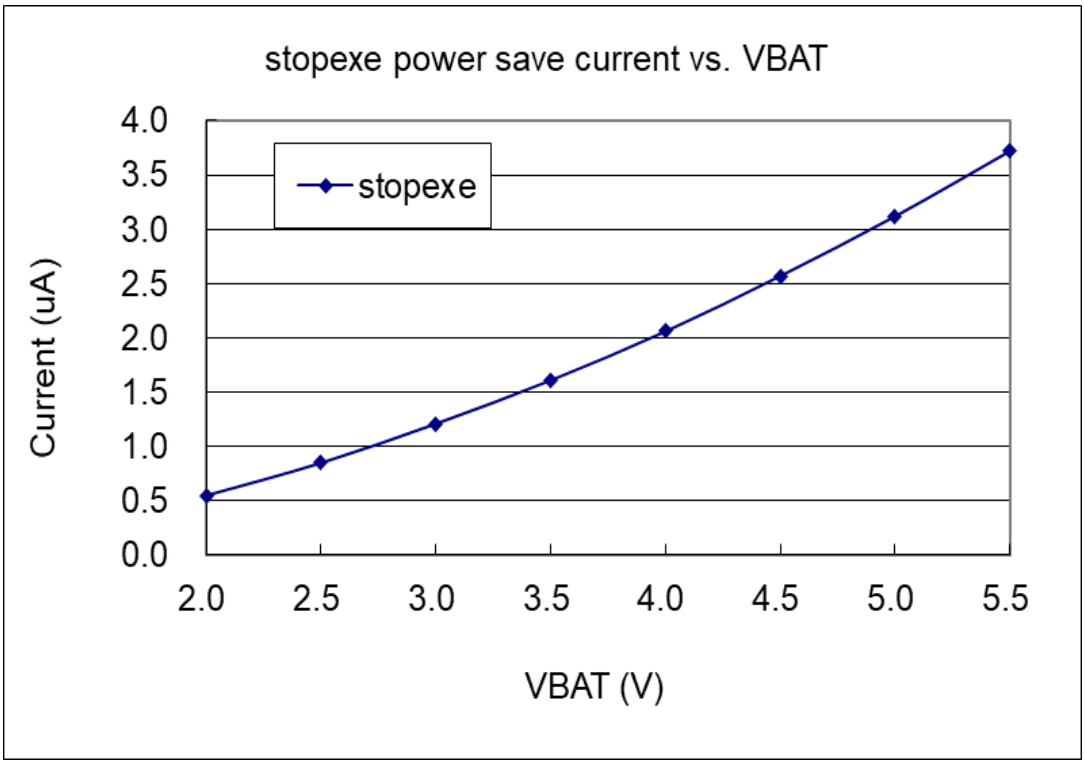
4.13. IO 引脚上拉/下拉阻抗曲线图





4.14. 掉电消耗电流(I_{PD})与省电消耗电流(I_{PS})关系曲线图





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。PMB180(B)的 OTP 程序内存容量为 1.25KW 如表 1 所示。OTP 内存从地址“0x4F0 ~0x4FF”供系统使用，从 0x001 到 0x00F 和从 0x011 到 0x4EF 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
•	•
0x4EF	用户程序区
0x4F0	系统使用
•	•
0x4FF	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 PMB180(B)，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 1 所示，其中 t_{SBP} 是开机时间。

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

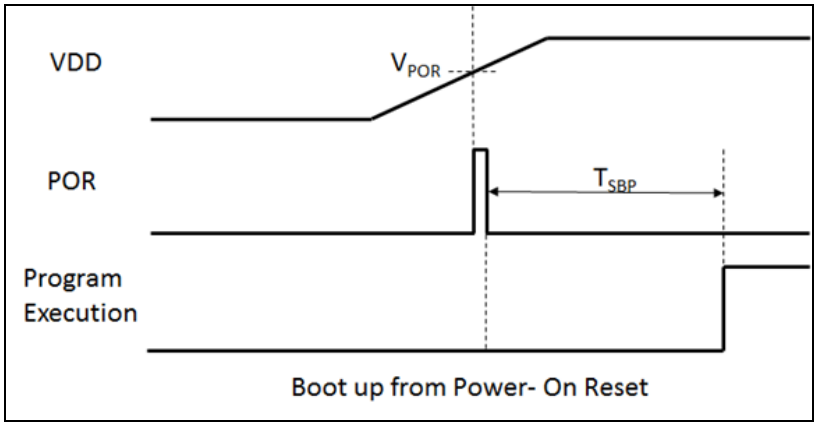
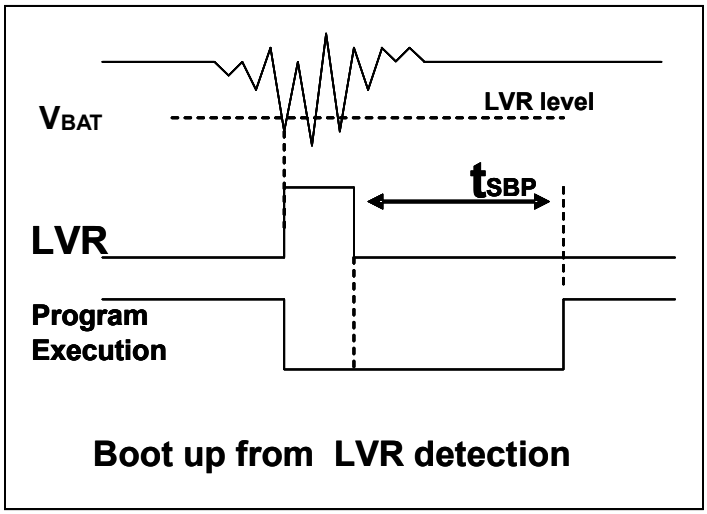
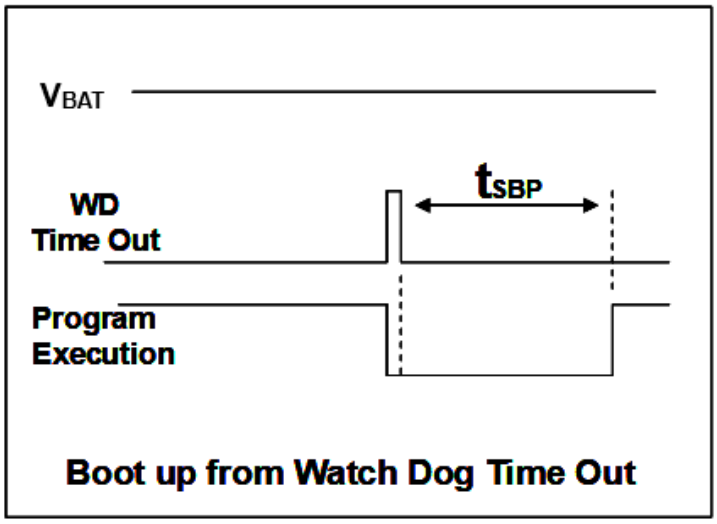
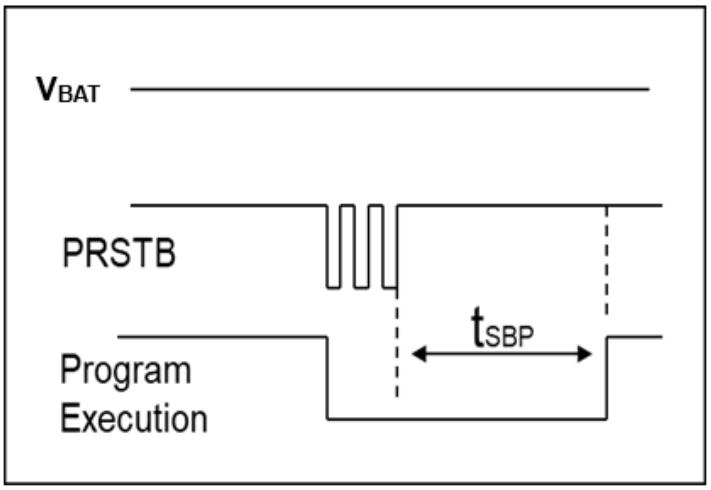


图 1：上电时序

5.2.1. 复位时序图







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，PMB180(B)的所有 64 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

PMB180(B)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_{BAT}、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，PMB180(B) 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_{BAT} =(**p3**)V

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	Description
○ 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，PMB180(B)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，PMB180(B)执行此命令后的状态：

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

开机后，CLKMD = 0x34：

- IIHRC 频率在 VDD =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, VDD=3.3V

开机后，CLKMD = 0x14：

- IHRC 频率在 VDD=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, VDD=2.5V

开机后，CLKMD = 0x3C：

- IHRC 频率在 VDD =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, VDD=2.5V

开机后，CLKMD = 0x1C：

- IHRC 频率在 VDD=2.5V 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, VDD =5V

开机后，CLKMD = 0x7C：

- IHRC 频率在 VDD=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, VDD=5V

开机后，CLKMD = 0XE4：

- IHRC 频率在 VDD=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，PMB180(B)的时钟系统的硬件框图，如图 2 所示：

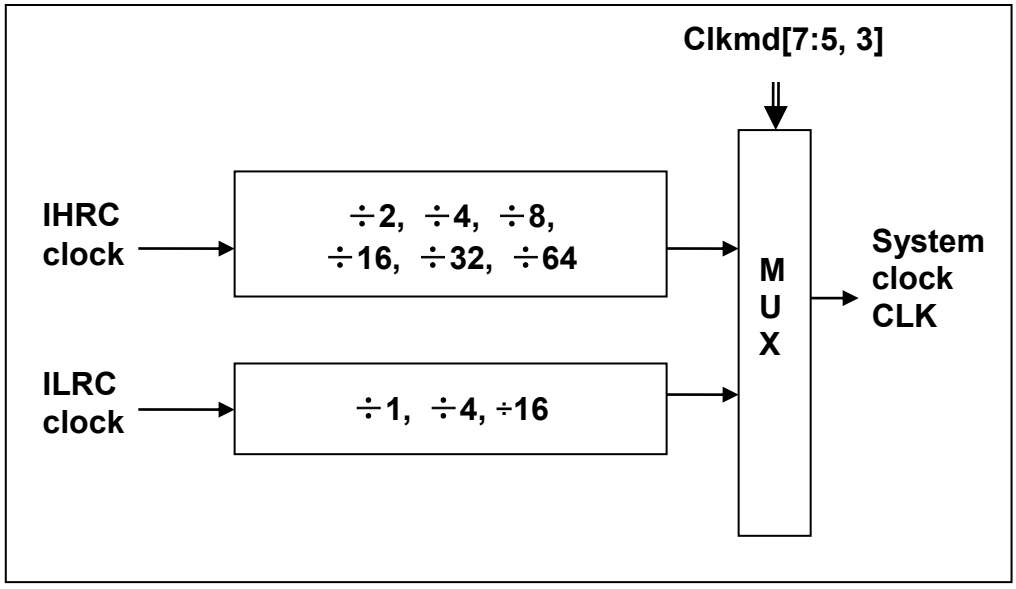


图 2：系统时钟选项

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，PMB180(B)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.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 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. 充电器

PMB180(B)内置一个硬件充电器。此充电器为完全恒流/恒压线性充电，可用于单节锂离子电池充电管理。由于内部 MOSFET 结构，不需要外部感测电阻器，也不需要阻断二极管，最大充电电流可达 500mA。充电器充电功能为供电即自动工作，不需 MCU 程序做启始设置即可采用硬件默认状态对电池做充电管理。

使用者可透过 CHG_CTRL[7:5]这三个 Bits 做充电电流的设置或是调整。共有 8 组充电电流可设置，最大 500mA，最小 50mA。

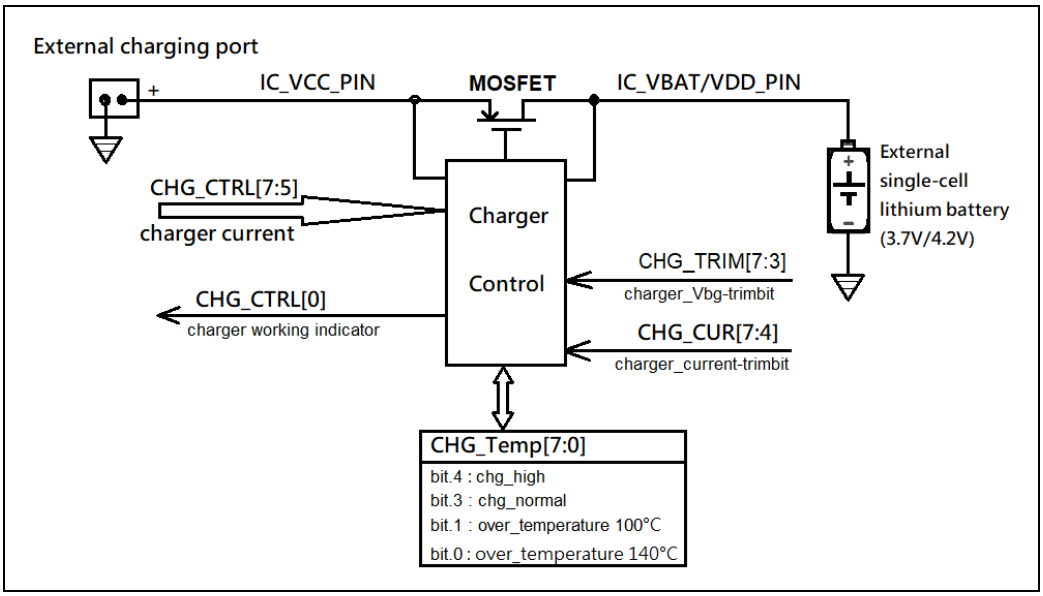
MCU 程序可读取寄存器 CHG_CTRL[0]来判断充电器工作状态。读取寄存器 CHG_Temp[4]可用于判断充电 Vcc 电压是否大于 Vbat。读取寄存器 CHG_Temp[3]可判别充电 Vcc 电压是否过低至使充电器自动关闭。当 CHG_CTRL[4:3] = 0b11 时可判读为充电接口已接入，且充电电压正常。

充电器也集成一个充电过温保护电路，充电过温保护有 100°C 度及 140°C 两种选项。可由 CHG_Temp[1:0]的写入做充电过温保护控制，CHG_Temp[1:0]的读取可判断充电过温是否触发。

PMB180(B)充电器的充电电压及电流已于出厂时做精准度校正，校正值会写入在系统参数保护区内。当 MCU 开机成功且在执行 “.Adjust_IC” 的宏指令时会将充电器的电压 / 电流出厂校正值回填写入 CHG_Trim / CHG_CUR 两组寄存器内，此时对充电器的充电电压及电流做量测方为出厂校准值。MCU OTP 为空白无程序

或是开机失败时对充电器所量测到的充电电压及电流皆是未校正的状态。CHG_Trim / CHG_CUR 两组寄存器不建议用户自行改写。使用 “.Adjust_IC Disable” 或是写汇编代码时可与 FAE 联络讨论。用户若想微调充电器的充电电压及电流可以使用”ReLoad_VbatBGTRIM”及”ReLoad_ChargerCURTRIM” 这两个宏指令。

PMB180(B)对于电池电压 Vbat 的侦测方式可使用内置比较器的 Bandgap 1.20V 与 VintR 分压做比较量测。由此可以计算出 Vbat 电池的电压。使用方式可参考比较器章节说明。



充电器充电电压固定在 4.2V，充电电流可通过寄存器 CHG_CTRL[7:5]设置。当达到最终恒定充电电压后，充电电流会自动降至寄存器 CHG_CTRL[7:5]所设定编程值的 1/10 时，充电器自动停止当次的充电循环。

卸下输入电源（墙壁插座或 USB 电源）后，PMB180(B)充电器 IP 自动进入低电流状态，使电池漏电流降至低于 2uA。

充电器在设计上也俱备其他功能如 欠压锁定、自动充电和涓流充电模式。

5.5.1. 热限值

充电器具备热反馈调节充电电流以限制在高功率操作或高环境温度。当 PMB180(B) IC 内部温度上升超过约 90°C 的预设值，内部热反馈回路将减少编程所设定的充电电流。此功能可保护 PMB180(B)不受过高温度的影响，并允许用户在不损坏 PMB180(B)的情况下推动给定电路板的功率处理能力极限。充电电流可根据典型（非最坏情况）环境温度进行设置，确保充电器在最坏情况下自动降低电流。

1.5.2. 功耗

PMB180(B)通过热反馈降低充电电流的条件可以参考 IC 中功耗来近似。几乎所有的功耗都是由内部 MOSFET 产生的，这大约为：

$P_D = (VCC – VBAT) • I_{VBAT}$

其中 PD 是功耗，V_{VCC} 是输入电源电压，V_{VBAT} 是电池电压，I_{VBAT} 是充电电流。热反馈开始保护 IC 的环境温度约为：

$T_A = 120°C – P_D\theta_{JA}$
$T_A = 120°C – (V_{VCC} – V_{VBAT}) • I_{VBAT} • \theta_{JA}$

示例：PMB180(B)由 5V USB 供电，通过编程向放电锂离子电池提供 400mA 满标度电流，电压为 3.75V。假设 θ_{JA} 为 150°C/W（参见 PCB 布局注意事项），PMB180(B)开始降低充电电流的环境温度约为：

$T_A = 90°C – (5V – 3.75V) • (400mA) • 150°C/W$
$T_A = 90°C – 0.5W • 150°C/W = 90°C – 75°C$
$T_A = 15°C$

PMB180(B)可在环境温度高于 15°C 时使用，但充电电流将从 400mA 降低。给定环境温度下的近似电流约为：

$$I_{VBAT} = \frac{90°C - T_A}{(V_{VCC} - V_{VBAT}) \cdot \theta_{JA}}$$

使用环境温度为 60°C 的前一个示例，充电电流将减少至约为：

$$I_{VBAT} = \frac{90°C - 60°C}{(5V - 3.75V) \cdot 150\,°C/W} = \frac{30°C}{187.5\,°C/A}$$

$$I_{VBAT} = 160mA$$

注意，应用 PMB180(B)时不要在最热条件下进行设计，因为当结温达到约 90°C 时，IC 将自动降低功耗。

1.5.3. 热温条件

由于封装尺寸较小，因此使用良好的 PCB 布局来散热使充电电流达到最大非常重要。IC 产生热量的路径是从芯片到引脚，通过封装线（尤其是接地线）到 PCB 铜箔。PCB 铜箔等于散热器。铜焊盘占地区域应尽可能大并扩展到更大的铺铜区域，将热量扩散并消散到周围环境。穿过内部或背面铜层的过孔也有助于改善充电器的整体热性能。在设计 PCB 布局时，还必须考虑板上与充电器无关的其他热源，因为它们会影响整体温升和最大充电电流。

下表列出了几种不同板尺寸和铜区域的热阻。所有测量均在 3/32" FR-4 板上的静止空气中进行，器件放置在顶部。

覆铜面积		电路板面积	结与环境间的热阻
上部	底部		
2500mm^2	2500mm^2	2500mm^2	125°C/W
1000mm^2	2500mm^2	2500mm^2	125°C/W
225mm^2	2500mm^2	2500mm^2	130°C/W
100mm^2	2500mm^2	2500mm^2	135°C/W
50mm^2	2500mm^2	2500mm^2	150°C/W

表 4：测量的热阻（双层板）

5.5.4. EPAD

PCB 布局走线时需确保封装引脚 GND 及 EPAD 具有足够数量的热通路，以使热通路有效。

5.6. 比较器

PMB180(B)内置一个硬件比较器，如图 3 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 做比较。两个信号进行比较，一个是正输入，另一个是负输入。比较器的负输入可以是 PA3，PA4，内置 bandgap (1.2v)，PA6，PA7，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。输出结果信号可以由 PA0 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 选择，比较输出结果可以用来产生中断信号。

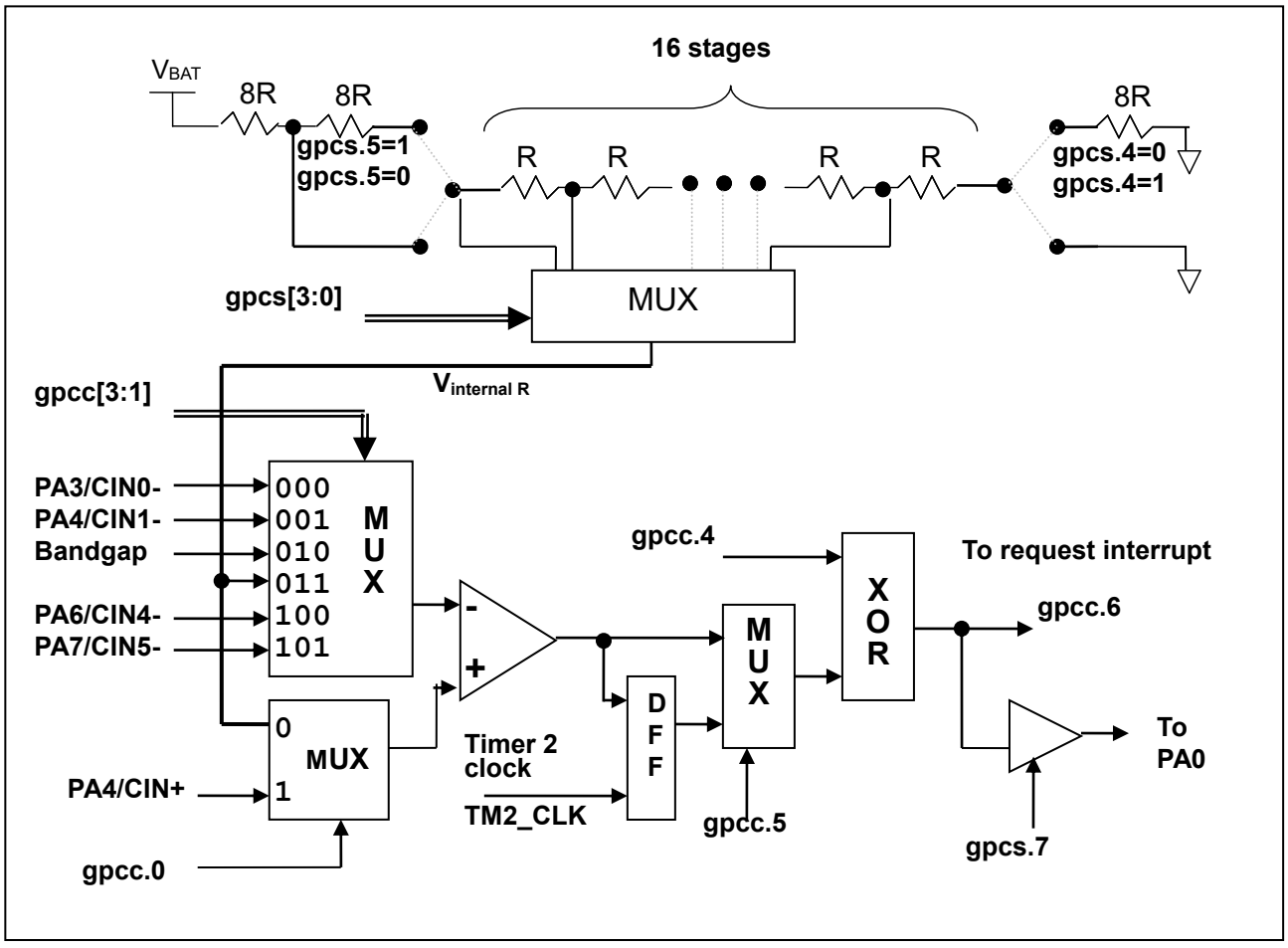


图 3：比较器硬件原理框图

5.6.1. 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 4 ~ 图 7 显示四个条件下有不同的参考电压 $V_{internal\ R}$。内部参考电压 $V_{internal\ R}$ 可以通过 gpcs 寄存器来设置，范围从(1/32)*V_{BAT} 到(3/4)*V_{BAT}。

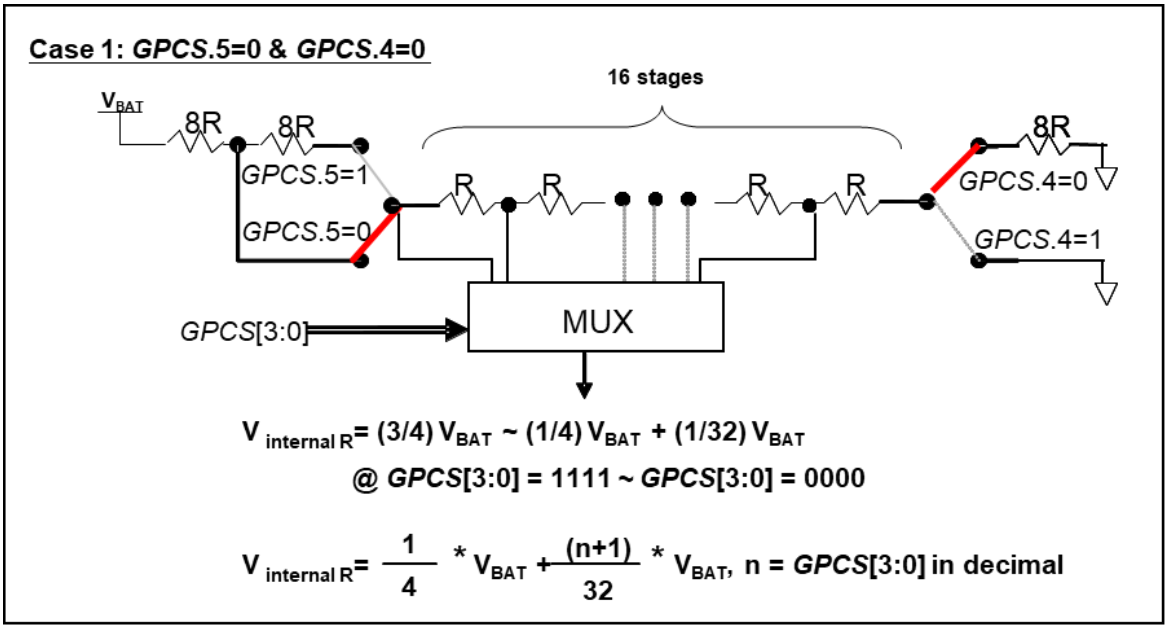


图 4：$V_{internal\ R}$ 硬件接法 (gpcs.5=0 & gpcs.4=0)

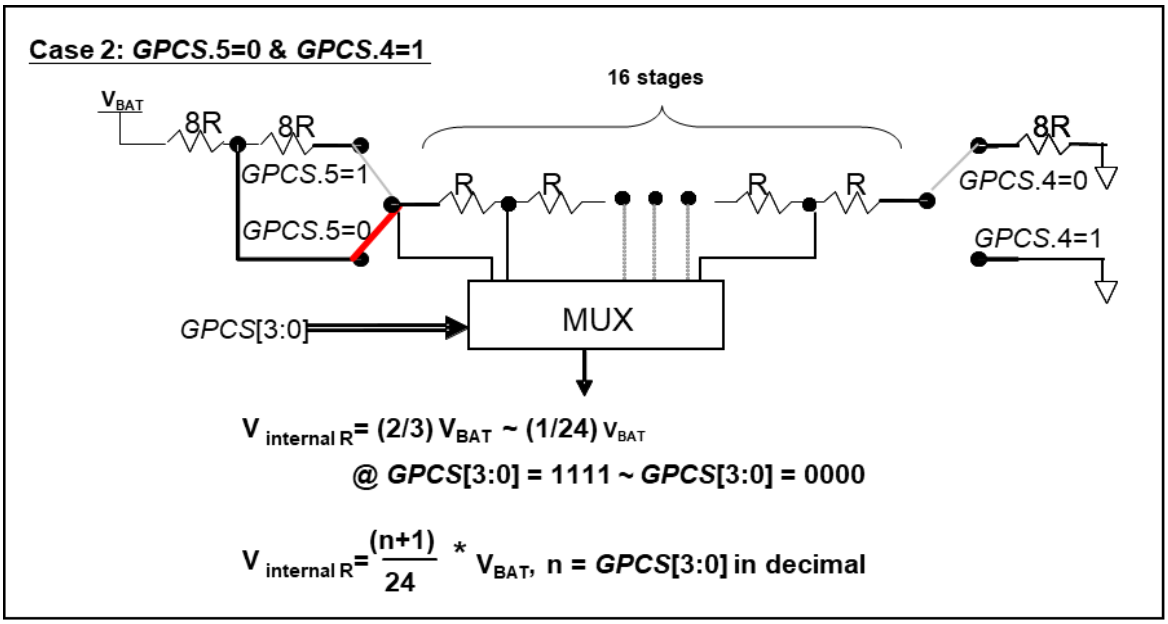


图 5：$V_{internal\ R}$ 硬件接法 (gpcs.5=0 & gpcs.4=1)

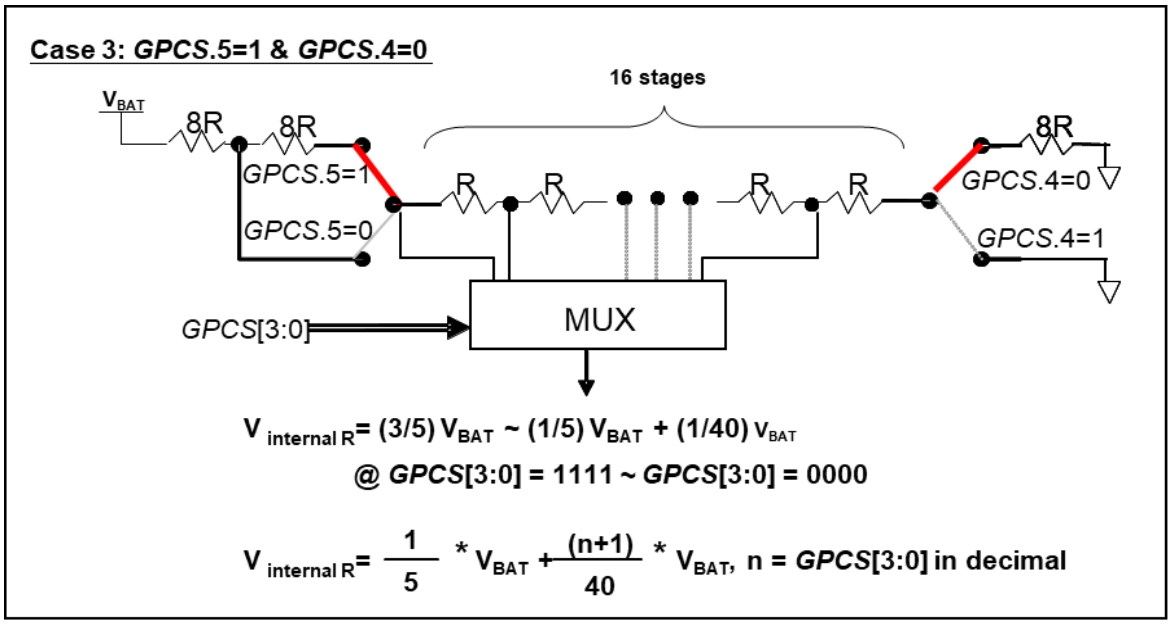


图 6：$V_{internal\ R}$ 硬件接法 (gpcs.5=1 & gpcs.4=0)

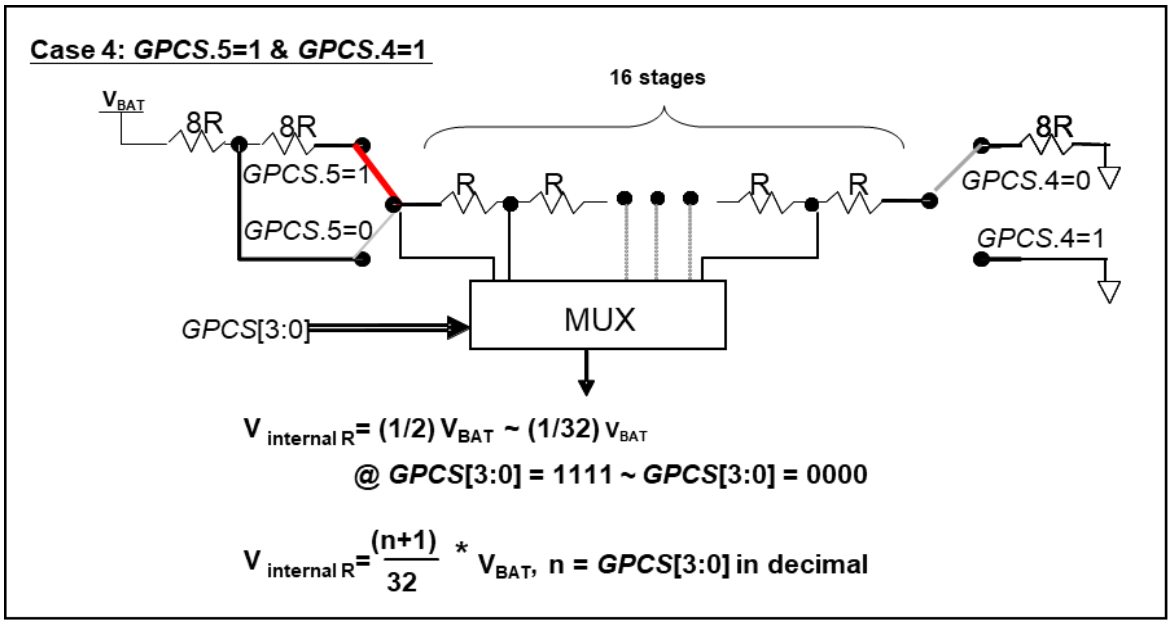


图 7：$V_{internal\ R}$ 硬件接法 (gpcs.5=1 & gpcs.4=1)

5.6.2. 使用比较器

例 1：

选择 PA3 为负输入和 $V_{internal\ R}$ 的电压为(18/32)* V_{BAT} 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_{BAT} + [(9+1)/32]* V_{BAT} = [(9+9)/32]* V_{BAT} = (18/32)* V_{BAT} 的参考电压。

```
gpcs    = 0b0_0_00_1001;          // Vinternal R = VBAT *(18/32)
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：由客户自定）
```

或

```
$ GPCS     VBAT*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_{BAT} ，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_{BAT} + [(13+1)/40]* V_{BAT} = [(13+9)/40]* V_{BAT} = (22/40)* V_{BAT}.

```
gpcs    = 0b1_0_1_0_1101;         // 输出到 PA0，Vinternal R = VBAT *(22/40)
gpcc    = 0b1_0_0_1_011_1;        // 反极性输出，负输入: Vinternal 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, VBAT *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。

5.6.3. 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_{BAT}，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_{BAT} 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_{BAT} 的电压就可以透过下列公式计算：

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

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

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

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

例一：

```
$ GPCS    VBAT *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 VBAT 大于 4V 时
}
else
{                                   // 当 VBAT 小于 4V 时
}
```

5.7. 16 位计数器 (Timer16)

PMB180(B) 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。选择器用于选择 Timer16 的中断条件，无论何时发生溢出，都可以触发 Timer16 中断。计时器 16 的硬件框图如图 8 所示。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***。

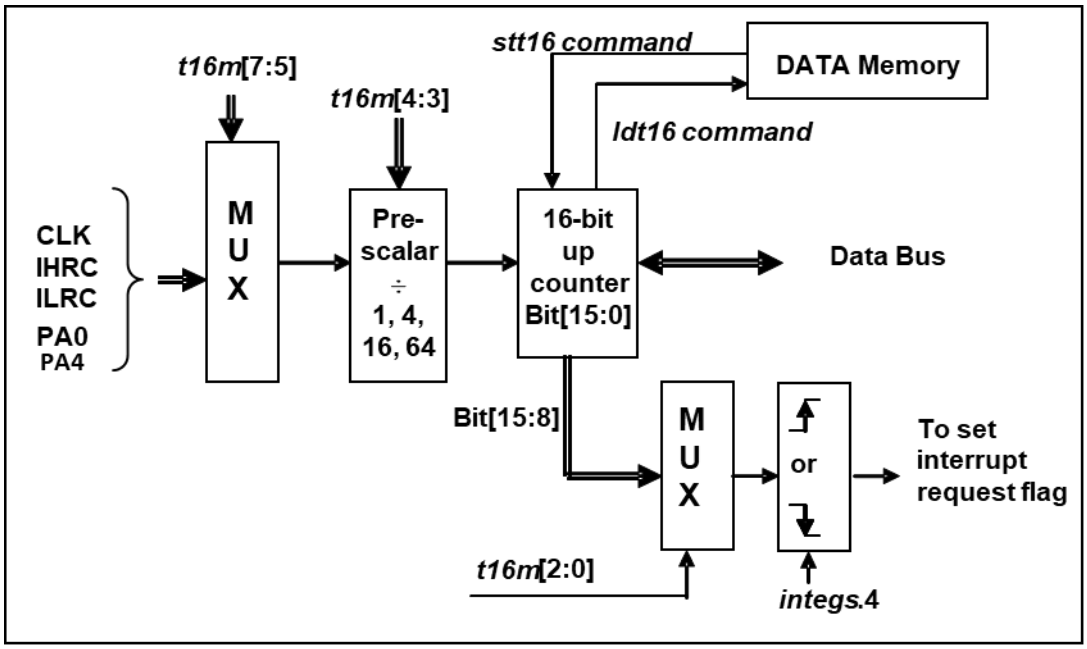


图 8：Timer16 模块框图

当使用 Timer16 时，Timer16 的语法定义在.INC 文件中。有三个参数来定义 Timer16 的使用。第一个参数是用来定义 Timer16 的时钟源，第二个参数是用来定义预分频器，最后一个参数是定义中断源。详细如下：

```
T16M        IO_RW           0x06
$ 7~5:      STOP, SYSCLK, X, PA4_F, IHRC, X, ILRC, PA0_F      // 第一个参数.
$ 4~3:      /1, /4, /16, /64                                  // 第二个参数.
$ 2~0:      BIT8, BIT9, BIT10, BIT11, BIT12, BIT13, BIT14, BIT15   // 第三个参数.
```

用户可以依照系统的要求来定义 T16M 参数，例子如下，更多例子请参考 IDE 软件“说明→ 使用手册→ IC 介绍 → 缓存器介绍 → T16M”：

```
$    T16M    SYSCLK, /64, BIT15;
     // 选择(SYSCLK/64)当 Timer16 时钟源，每 2^16 个时钟周期产生一次 INTRQ.2=1
     // 如果系统时钟 System Clock = IHRC / 2 = 8 MHz
     // 则 SYSCLK/64 = 8 MHz/64 = 125kHz(8us)，约每 524 mS 产生一次 INTRQ.2=1


$    T16M    PA0_F, /1, BIT8;
     // 选择 PA0 当 Timer16 时钟源，每 2^9 个时钟周期产生一次 INTRQ.2=1
     // 每接收 512 个 PA0 时钟周期产生一次 INTRQ.2=1

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

如果 Timer16 在自由运行状态下运行，中断频率如下所示：

$$F_{INTRQ\_T16M} = F_{clock\ source} \div P \div 2^{n+1}$$

式中，F 为所选时钟源是 Timer16 的频率；

P 是 t16m[4:3]的选择；(1, 4, 16, 64)

N 是选择请求中断服务的第 N 位，例如：如果选择了位 10，则 N＝10。

5.8. 8 位定时器(Timer2) / PWM 生成器

PMB180(B)内置一个 8 位硬件计数器(Timer2)。请参考 Timer2 的硬件框图，Timer2 计数器的时钟源可以来自系统时钟(CLK)、内部高频振荡器 (IHRC)、内部低频振荡器 (ILRC)、NILRC、PA0、PA4 和比较器。寄存器 ***tm2c*** 的位[7:4]用于选择 Timer2 的时钟。如果 Timer2 时钟源选择 IHRC，当 ICE 暂停时，Timer2 的时钟仍继续运行。Timer2 的输出可以发送到引脚 PA3 或 PA4，具体取决于 ***tm2c[3:2]***。时钟预分频模块提供÷1，÷4，÷16 和÷64 的选项，由 ***tm2s[6:5]***控制；还提供了一个÷1～÷32 的分频模块，并由 ***tm2s[4:0]***控制。结合预分频功能和分频功能，Timer2 时钟（TM2_CLK）频率可以广泛和灵活。

Timer2 计数器只能执行 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 位定时器计数值在周期模式下达到上限寄存器设定的范围时，定时器将自动清零。上限寄存器用来定义 PWM 的定时器周期或占空比。Timer2 有两种工作模式：周期模式和 PWM 模式；周期模式用于生成周期性输出波形或中断事件；PWM 模式用于生成具有可选 6 位、7 位或 8 位 PWM 分辨率的 PWM 输出波形，图 10 显示了周期模式和 PWM 模式下 Timer2 的时序图。

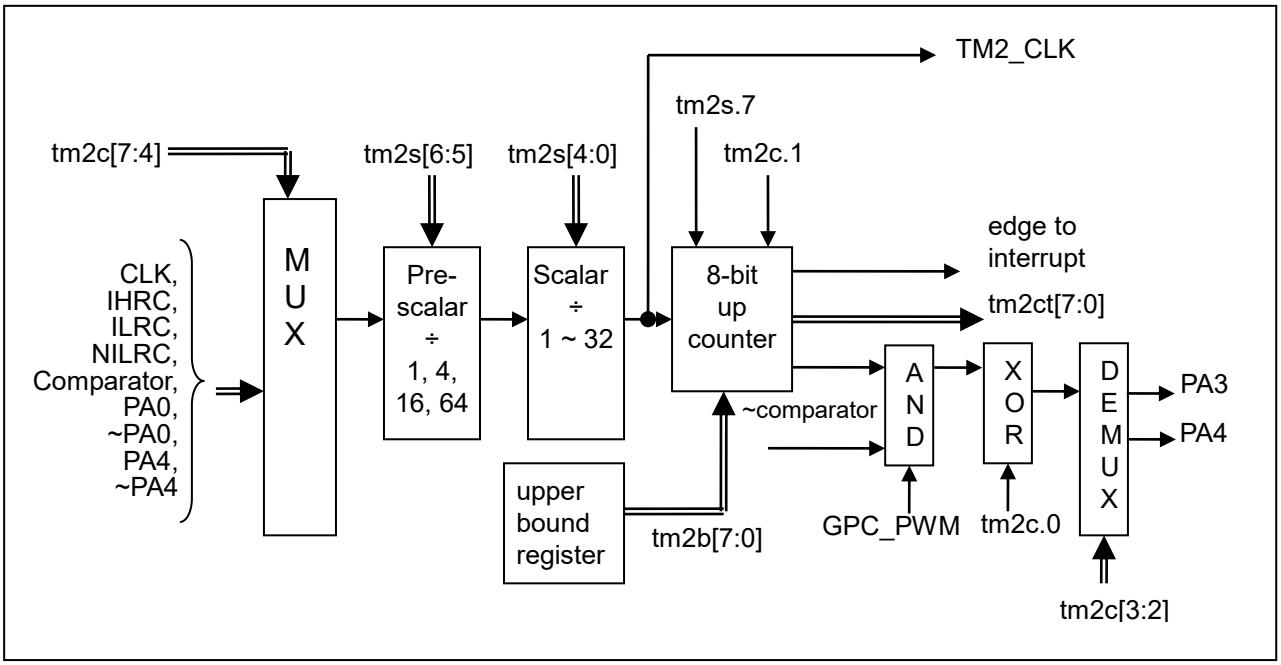


图 9：Timer2 硬件框图

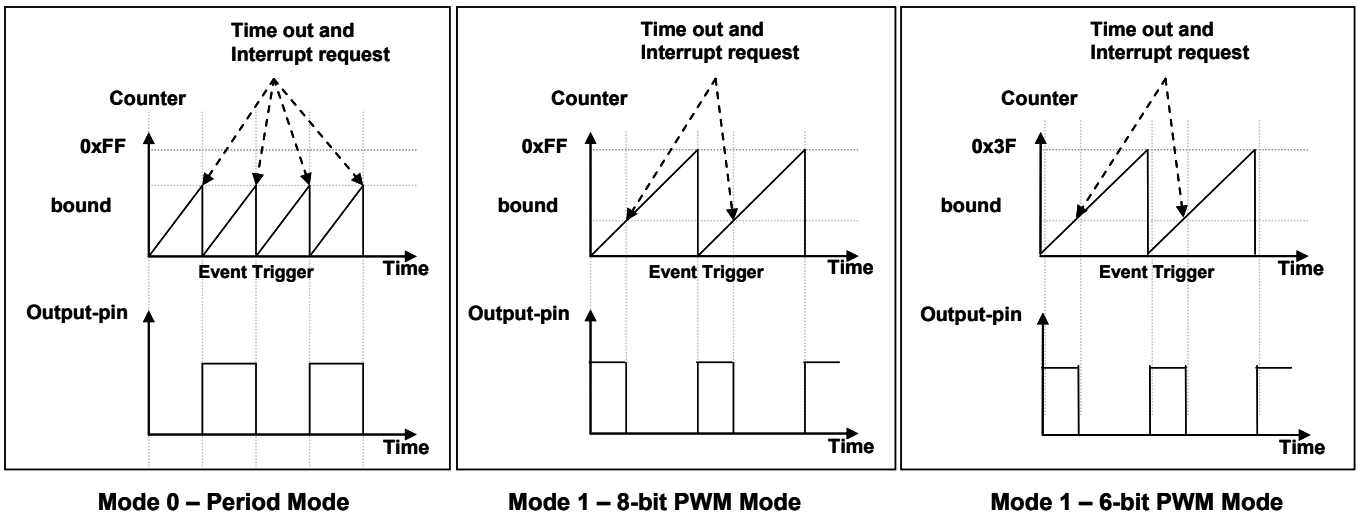


图 10：Timer2 在周期模式和 PWM 模式下的时序图（tm2c.1=1）

Code Option GPC_PWM 适用于需要由比较器结果控制生成的 PWM 波形的应用。如果选择了 Code Option GPC_PWM，则比较器输出为 1 时 PWM 输出关闭，比较器输出返回 0 时 PWM 输出启用，如图 11 所示。

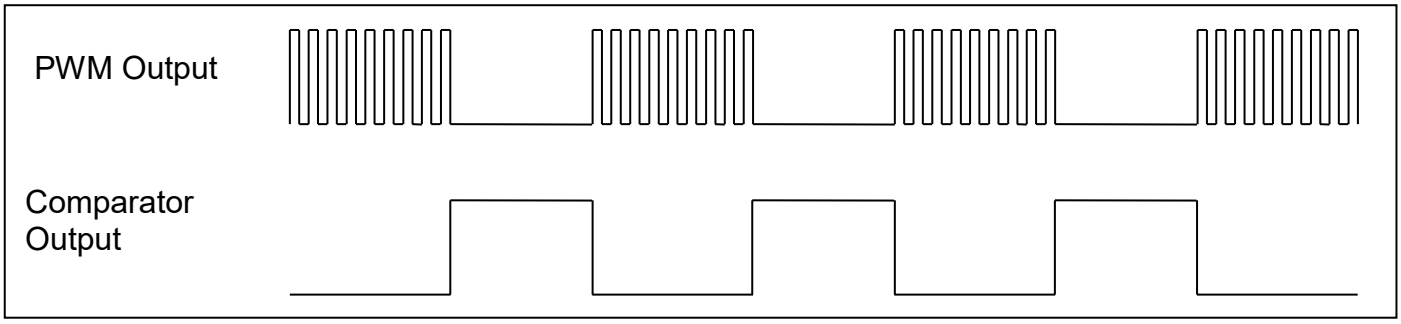


图 11：比较器控制 PWM 波形的输出

5.8.1. 使用 Timer2 生成周期性波形

如果选择周期模式，输出占空比始终为 50%；其频率可概括如下：

输出频率= Y ÷ [2 × (K+1) × S1 × (S2+1)]

式中，
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, VBAT =5V
    …
    tm2ct = 0x00;
    tm2b = 0x7f;
    tm2s = 0b0_00_00001;          //   8-bit PWM, pre-scalar = 1, scalar = 2
    tm2c = 0b0001_10_0_0;         //   system clock, output=PA3, period mode
    while(1)
    {
        nop;
    }
}
```

5.8.2. 使用 Timer2 产生 8 位 PWM 波形

如果选择 8 位 PWM 模式，应将 tm2c[1]设置为 1，tm2s[7]设置为 0，输出波形的频率和占空比总结如下：

输出频率 = Y ÷ [256 × S1 × (S2+1)]

占空比 = [(K+1) ÷ 256]×100%

式中，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

➔ 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

➔ 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
➔ PWM 输出持续为高
➔ 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
➔ 占空比 = [(9+1) ÷ 256] × 100% = 3.9%

使用 Timer2 从 PA3 生成 PWM 波形的示例程序如下所示：

```
void    FPPA0 (void)
{
    .ADJUST_IC    SYSCLK=IHRC/2, IHRC=16MHz, VBAT =5V
    wdreset;
    tm2ct = 0x00;
    tm2b = 0x7f;
    tm2s = 0b0_00_00001;            //    8-bit PWM, pre-scalar = 1, scalar = 2
    tm2c = 0b0001_10_1_0;       //    system clock, output=PA3, PWM mode
    while(1)
    {
        nop;
    }
}
```

5.8.3. 使用 Timer2 产生 6 位 PWM 波形

如果选择 6 位 PWM 模式，应将 tm2c[1]设置为 1，tm2s[7]设置为 0，输出波形的频率和占空比总结如下：

输出频率= Y ÷ [64 × S1 × (S2+1)]

占空比 = [(K＋1) ÷ 64] × 100%

式中，
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）

用户可以使用 TM2_bit 代码选项将 Timer2 设置为 7 位 PWM 模式，而不是 6 位模式。此时，上述方程的计算因子变为 128 而不是 64。

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
➔ 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
➔ duty of output = [(31+1) ÷ 64] × 100% = 50%

例 3:
tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0011_1111, K=63
tm2s = 0b1000_00000, S1=1, S2=0
➔ PWM 输出持续为高
➔ 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
➔ 占空比 = [(0+1) ÷ 64] × 100% =1.5%

5.9. 11-bit PWM 生成器

PMB180(B)内置一组三路 11 位 SuLED（Super LED）硬件 PWM 发生器。它由三个 PWM 发生器（LPWMG0、LPWMG1 和 LPWMG2）组成。各路输出端口如下：

- LPWMG0 – PA0, PA1, PA5
- LPWMG1 – PA4, PA6
- LPWMG2 – PA3, PA5

注：5S-I-S01/2(B)不支持 11 位 SuLED 硬件 PWM 发生器的功能。

5.9.1. PWM 波形

PWM 输出波形（图 12）有一个时基（T_{Period}=周期时间）和一个周期里输出高电平的时间（占空比）。PWM 输出的频率取决于时基($f_{PWM} = 1/T_{Period}$)。

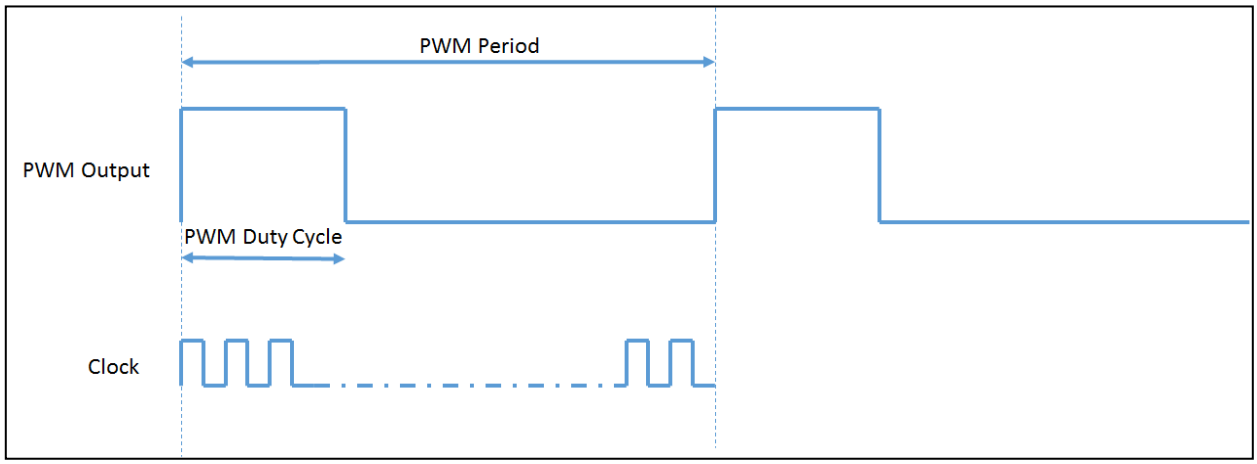


图 12：PWM 输出波形

5.9.2. 硬件框图

图 13 显示了整组 SuLED 11 位硬件 PWM 发生器的硬件方框图。这三组 PWM 生成器使用共同的 Up-Counter 和时钟源选择开关来产生时基，所以 PWM 周期的起点（上升沿）是同步的，时钟源可以是 IHRC 或系统时钟。PWM 信号输出引脚通过寄存器 ***lpwmgxc[3:1]*** 选择的。PWM 输出 Enable 由 GPC 控制，可通过 ***lpwmgxc.7***. 选择。PWM 波形的周期由公共 PWM 上限高和低寄存器决定，各路 PWM 波形的占空比由各路 PWM 占空比高和低压寄存器决定。

在 LPWMG0 通道的那两个附带的 OR 和 XOR 逻辑门是用于产生互补非重叠并有死区的开关控制波形的。选择代码选项 GPC_TMx_LPWM 也可以通过比较器结果来控制所生成的 PWM 波形。

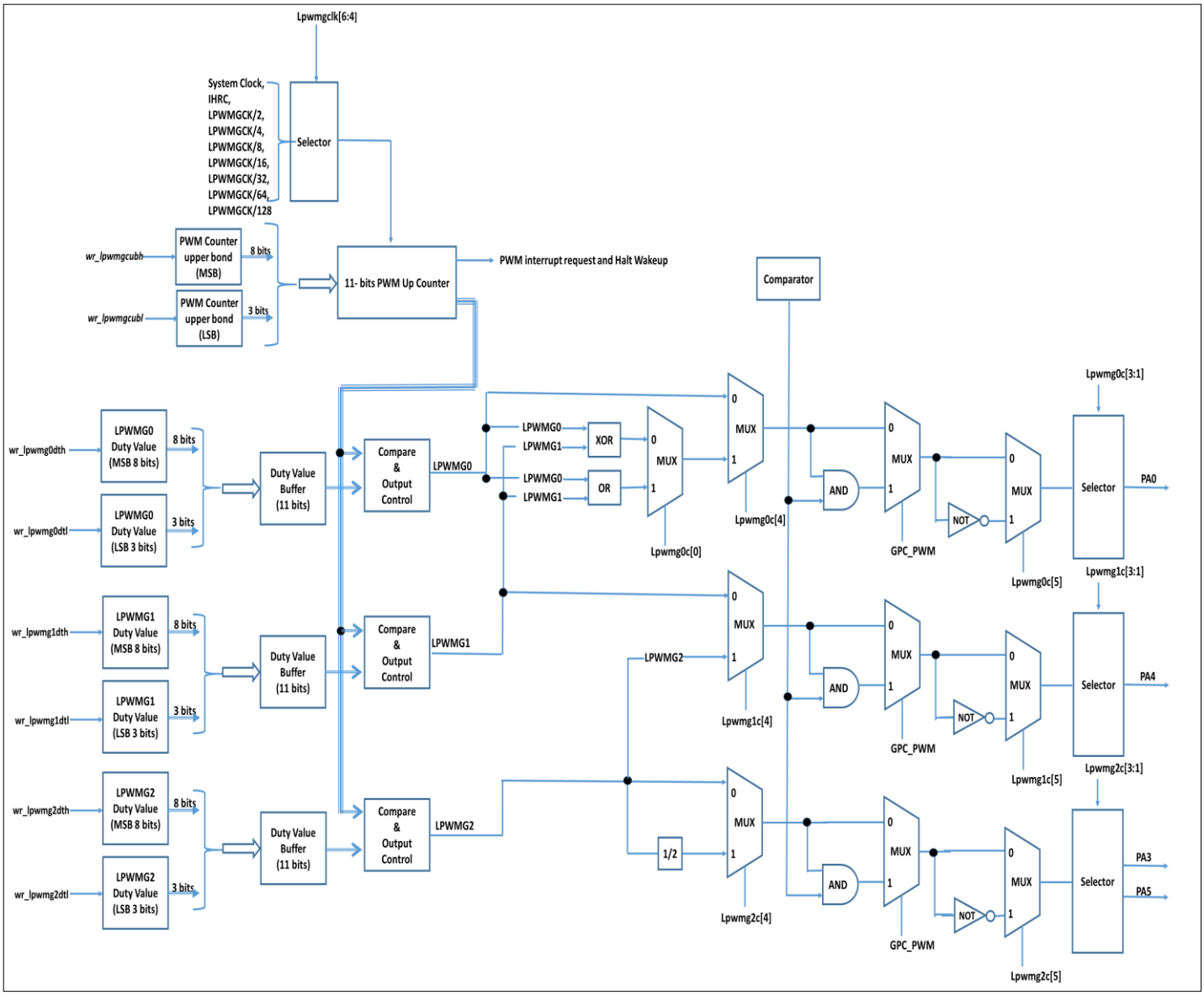


图 13：整组 SuLED 三路 11 位 PWM 生成器的硬件框图

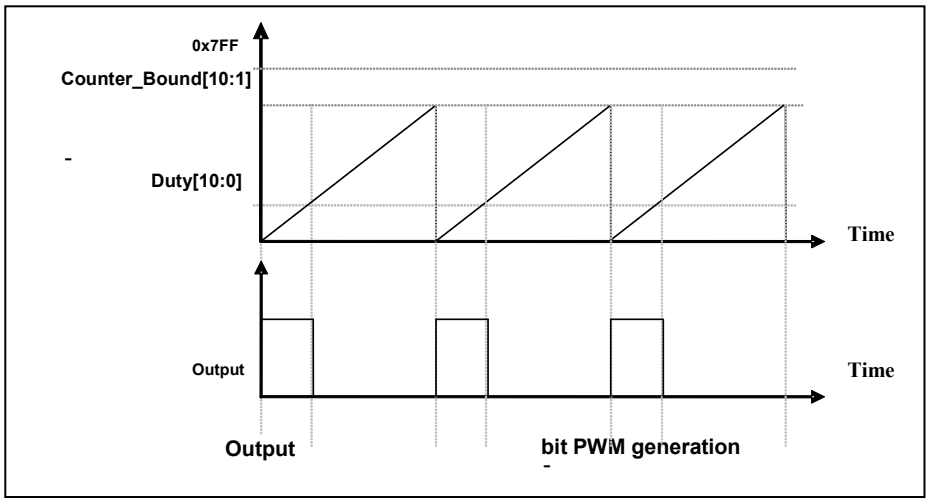


图 14：11 位 PWM 生成器输出时序图

5.9.3. 11 位 PWM 生成器计算公式

PWM 输出频率 $F_{PWM} = F_{clock\ source} \div [\ P \times (CB10\_1 + 1)\ ]$

PWM 占空比(时间) = $(1 / F_{PWM}) \times (\ DB10\_1 + DB0 \times 0.5 + 0.5) \div (CB10\_1 + 1)$

PWM 占空比(百分比) = $(\ DB10\_1 + DB0 \times 0.5 + 0.5) \div (CB10\_1 + 1) \times 100\%$

式中,

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

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

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

CB10_1 = Counter_Bound[10:1] = {*LPWMGCUBH[7:0], LPWMGCUBL[7:6]*}， 占空比

5.9.4. 带互补死区的 PWM 波形范例

基于 PMB180(B)独特的 11 bit PWM 架构，在此采用 PWM2 输出、PWM0 与 PWM1 异或后通过 PWM0 反极性输出，来获得两路互补带死区的 PWM 波形。

示例如下:

```
#define   dead_zone      10          //   死区时间 = 10% * (1/PWM_Frequency) us
#define   PWM_Pulse      50          //   该互补死区 PWM 占空比为 50%

#define   PWM_Pulse_1    35          //   该互补死区 PWM 占空比为 35%
#define   PWM_Pulse_2    60          //   该互补死区 PWM 占空比为 60%
#define   switch_time    400*2       //   切换占空比时，用于调整切换时间

// 注：为防止杂波产生，switch_time 应为 PWM 周期的倍数。此例 PWM 周期 = 400us
// 故切换时间为 400*2 us
```

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

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

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

while(1)
{
   //******** 切换占空比 *********************************
      // 切换占空比时，为避免可能出现的瞬间死区消失，应遵循如下顺序。
      // 占空比由大变小：  50%/60%   →   35%
   LPWMG0DTL   =     0x00;
   LPWMG0DTH   =     PWM_Pulse_1 + dead_zone;
   LPWMG2DTL   =     0x00;
   LPWMG2DTH   =     PWM_Pulse_1 + dead_zone*2;
   .delay      switch_time

      // 占空比由小变大：  35%  →  60%
   LPWMG2DTL   =     0x00;
   LPWMG2DTH   =     PWM_Pulse_2 + dead_zone*2;
   LPWMG0DTL   =     0x00;
   LPWMG0DTH   =     PWM_Pulse_2 + dead_zone;
   .delay      switch_time
}
}
```

下图显示了不同条件下的波形。

1. 固定占空比中的 PWM 波形：

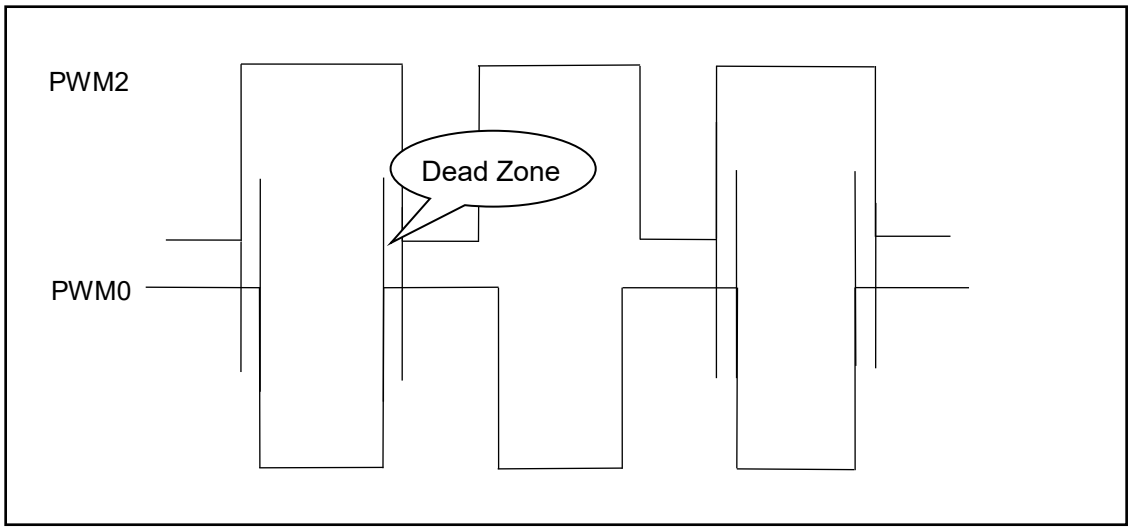


图 15：互补死区 PWM 波形

2. 切换两个占空比时的 PWM 波形：

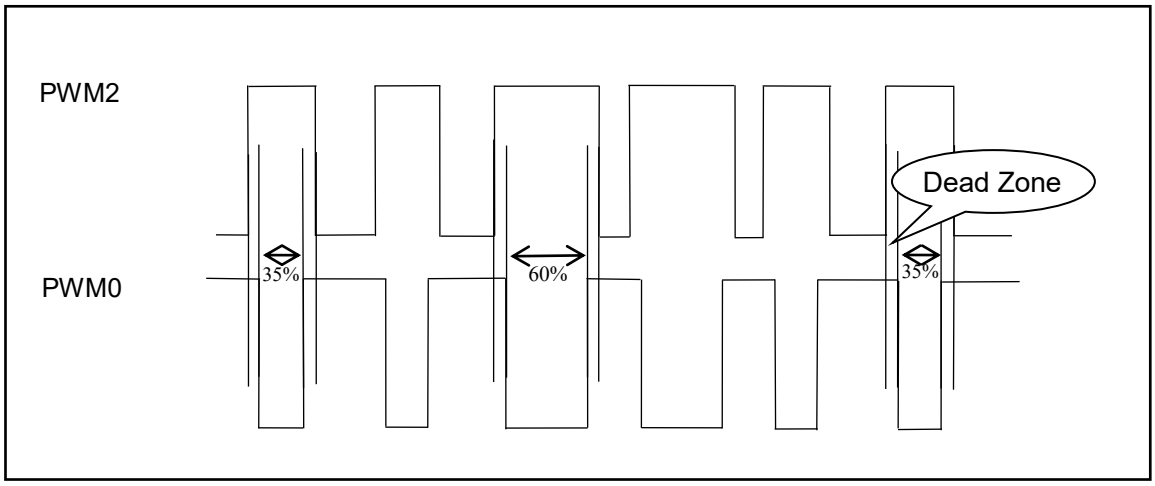


图 16：互补死区的 PWM 波形

可以发现，上述例程所得波形，其死区两组 PWM 同时为高。若用户需要 PWM 同时为低的死区，仅需要改变各自输出控制的 Inverse 即可。如：

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

5.10. 看门狗计数器

看门狗 (WDT)是一个计数器，其时钟源来自内部低频振荡器 (ILRC)。利用 *misc* 寄存器的选择，可以设定四种不同的看门狗超时时间，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，PMB180(B)将复位并重新运行程序。看门狗时序图如图 17 所示。

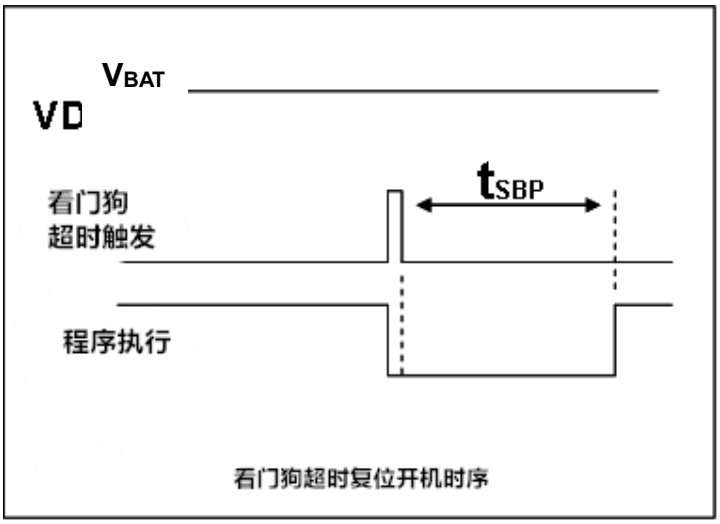


图 17：看门狗超时溢出时序图

5.11. 中断

PMB180(B)有 6 个中断源：

- 外部中断源 PA0
- 外部中断源 PA4
- Timer16 中断
- GPC / PWMG1 中断
- LPWMG 中断
- Timer2 中断

每个中断请求源都有自己的中断控制位来启用或停用。中断功能的硬件框图如图 18 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。

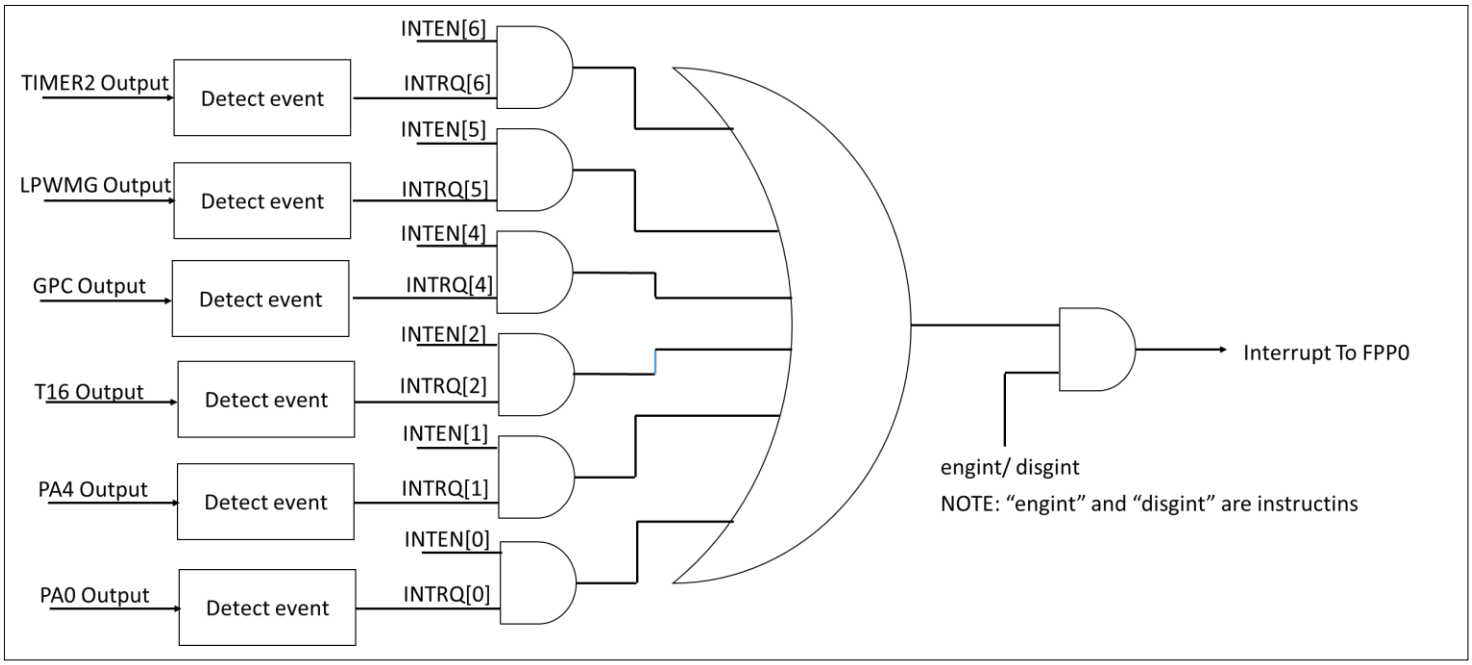


图 18：中断控制器硬件框图

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
    INTRQ2   =   0;             // 清除 INTRQ2
    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
    }
```

5.12. 省电和掉电

PMB180(B)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	NILRC
STOPSYS	停止	停止	没改变
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

5.12.1. 省电模式 (“*stopexe*”)

用 ***stopexe*** 指令进入省电模式，只有系统时钟被停用，其余所有的振荡器模块都仍继续工作。所以只有 CPU 是停止执行指令，然而，对 Timer16 计数器而言，如果它的时钟源不是系统时钟，那 Timer16 仍然会保持计数。***stopexe*** 的省电模式下，唤醒源可以是 IO 的切换，或者 Timer16 计数到设定值时（假如 Timer16 的时钟源是 IHRC 或者 ILRC），或者使用 NILRC 作时钟源的 TM2C 唤醒或比较器唤醒（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 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，LPWMG0，LPWMG1，LPWMG2）。
- 唤醒源：
  a. IO Toggle 唤醒：IO 在数字输入模式下的电平变换（*PAC* 位是 0，*PADIER* 位是 1）。
  b. Timer 唤醒：如果计数器 (Timer)的时钟源不是系统时钟，则当计数到设定值时，系统会被唤醒。
  c. TM2C 唤醒（使用 NILRC 作时钟源）
  d. 比较器唤醒：使用比较器唤醒时，需同时设定 GPCC.7 为 1 与 GPCS.6 为 1 来启用比较器唤醒功能。但请注意：内部 1.20V Bandgap 参考电压不适用于比较器唤醒功能。

举例使用 Timer16 唤醒省电模式 “***stopexe***”：

```
$ 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 IHRC 时钟后，系统将被唤醒。

5.12.2. 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，PMB180(B)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（PADIER 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
```

5.12.3. 唤醒

进入掉电或省电模式后，PMB180(B)可以通过切换 IO 引脚或 TM2C 唤醒（使用 NILRC 作时钟源）恢复正常工作；而 Timer16 唤醒只适用于省电模式。表 6 显示 ***stopsys*** 掉电模式和 ***stopexe*** 省电模式在唤醒源的差异。

掉电模式(stopsys)和省电模式(stopexe)在唤醒源的差异			
	IO 引脚切换	使用 NILRC 作时钟源的 TM2C 唤醒	Timer16 唤醒 / 比较器唤醒
STOPSYS	是	是	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 PMB180(B)，***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 3000 个 ILRC 时钟周期，另外，PMB180(B)提供快速唤醒功能，透过 ***misc*** 寄存器选择快速唤醒大约 45 个 ILRC 时钟周期。

休眠模式	唤醒模式	切换 IO 引脚的唤醒时间(t_{WUP})
STOPEXE 省电模式 STOPSYS 掉电模式	快速唤醒	45 * T_{ILRC}, 这里的 T_{ILRC} 是指 ILRC 时钟周期
STOPEXE 省电模式 STOPSYS 掉电模式	正常唤醒	3000 * T_{ILRC}, 这里的 T_{ILRC} 是指 ILRC 时钟周期

表 7：快速/正常唤醒之间的唤醒时间差异

请注意：当使用快速开机模式时，不管寄存器 ***misc***.5 是否选择了唤醒模式，都会强制使用快速唤醒模式。如果选择正常开机模式，即由寄存器 ***misc***.5 来选择唤醒模式。

5.13. IO 引脚

通过配置数据寄存器（*pa*）、控制寄存器（*pac*）和拉高/拉低电阻（*paph/papl*），所有引脚可以独立设置为两种状态输出或输入。所有这些引脚都具有具有 CMOS 电平的施密特触发器输入缓冲器和输出驱动器。当它设置为低输出时，拉高电阻会自动关闭。如果用户想读取 pin 状态，请注意在读取数据端口之前应将其设置为输入模式；如果用户在设置为输出模式时读取数据端口，则读取数据来自数据寄存器，而不是来自 IO 板。例如，表 7 显示了端口 A 的位 0 的配置表。IO 缓冲器的硬件图也如图 19 所示。

所有 IO 引脚都具有相同的结构。当 PMB180(B)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 8 为端口 PA0 位的设定配置表。图 19 显示了 IO 缓冲区硬件图。

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

表 8：PA0 设定配置表

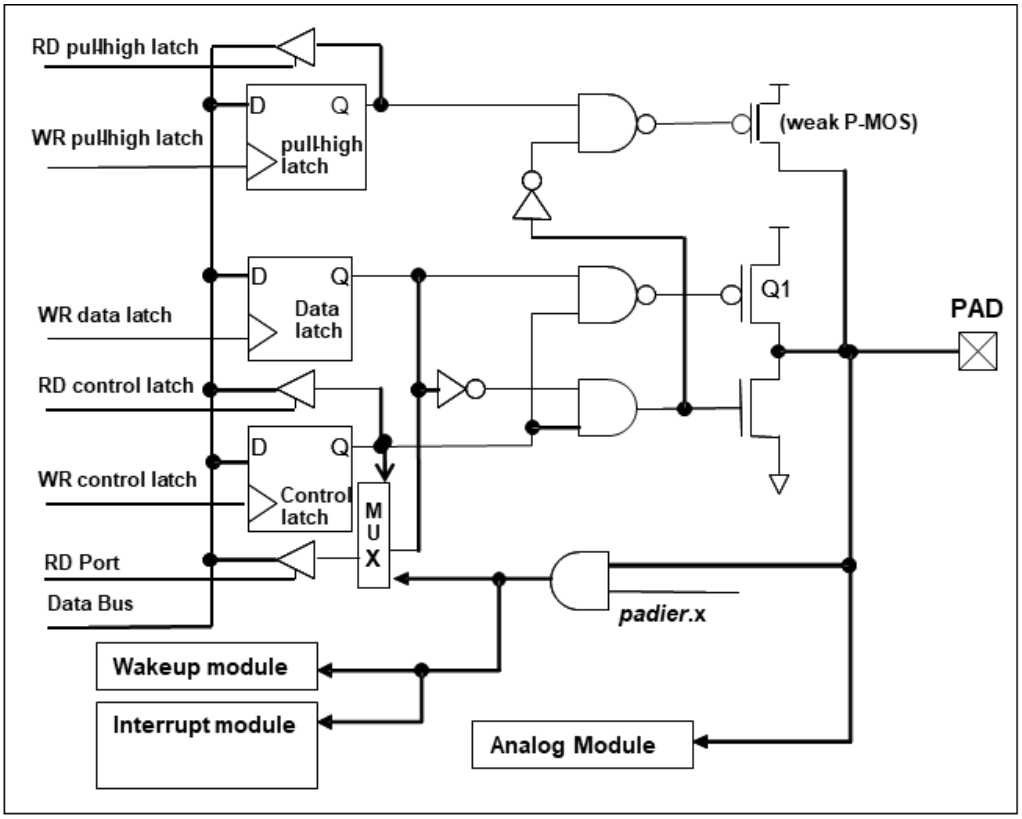


图 19：IO 引脚缓冲区硬件图

所有的 IO 引脚具有相同的结构。对于被选择为模拟功能的引脚，必须在寄存器 ***padier*** 相应位设置为低，以防止漏电流。当 PMB180(B)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。

5.14. 复位, LVR 复位及 LVD

5.14.1. 复位

引起 PMB180(B)复位的原因很多，一旦复位发生，PMB180(B)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 0x00。

发生上电复位或 LVR 复位后，若 VDD 大于 VDR（数据保存电压），数据存储器的值将会被保留；若 VDD 小于 VDR，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。

5.5.1 LVR 复位

根据代码选项，从 1.8V 到 4.0V 有 8 个不同级别的 LVR 用于复位。通常，用户根据工作频率和电源电压选择 LVR 复位电平。

5.14.2. LVD

通过代码选项，用户可以使用 LVDC[7:2]从 1.85V 到 5V 选择不同的 LVD 电平，LVD 可以为用户提供更准确的电压以确认电压电平。

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

<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 (CLK)选择：</td></tr>
<tr><td>类型 0，clkmd[3]=0</td><td>类型 1，clkmd[3]=1</td></tr>
<tr><td>000：IHRC/4
001：IHRC/2
01x：保留
100：保留
101：保留
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
110：保留
11X：保留</td></tr>
<tr><td>4</td><td>0</td><td>读/写</td><td colspan="2">内部高频 RC 振荡器功能。 0/1：停用/启用</td></tr>
<tr><td>3</td><td>0</td><td>读/写</td><td colspan="2">时钟类型选择。这个位是用来选择位 7~位 5 的时钟类型。
0/1：类型 0 /类型 1</td></tr>
<tr><td>2</td><td>1</td><td>读/写</td><td colspan="2">内部低频 RC 振荡器功能。0/1：停用/启用
当内部低频 RC 振荡器功能停用时，看门狗功能同时被关闭。</td></tr>
<tr><td>1</td><td>1</td><td>读/写</td><td colspan="2">看门狗功能。0/1：停用/启用</td></tr>
<tr><td>0</td><td>0</td><td>读/写</td><td colspan="2">引脚 PA5/PRSTB 功能。0/1：PA5 / PRSTB</td></tr>
</table>

6.4. 中断允许寄存器（*inten*），IO 地址 = 0x04

位	初始值	读/写	描述
7	0	读/写	保留
6	0	读/写	使能 Timer2 中断。0/1：停用/启用
5	0	读/写	使能 LPWMG 中断。0/1：停用/启用
4	0	读/写	使能比较器中断。0/1：停用/启用
3	0	读/写	保留
2	0	读/写	使能 Timer16 溢出中断。0/1：停用/启用
1	0	读/写	使能 PA4 中断。0/1：停用/启用
0	0	读/写	使能 PA0 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	保留
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	LPWMG 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	引脚 PA4 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
0	-	读/写	引脚 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 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. MISC 杂项寄存器 (*misc*), IO address = 0x08

位	初始值	读/写	描述
7 - 6	-	-	保留（写 0）。
5	0	只写	快唤醒功能。 0：正常唤醒 唤醒时间是 3000 个 ILRC 时钟（不适用快速开机） 1：快速唤醒 唤醒时间为 45 个 ILRC 时钟
4 - 3	-	-	保留（写 0）。
2	0	只写	停用 LVR 功能： 0 / 1：启用 / 停用
1 - 0	00	只写	看门狗时钟超时时间设定： 00：8k ILRC 时钟周期 01：16k ILRC 时钟周期 10：64k ILRC 时钟周期 11：256k ILRC 时钟周期

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

位	初始值	读/写	描述
7 - 1	-	-	保留
0	0	只写	将 Bandgap 和 LVR/LVD 硬件模块断电。0/1：正常/断电。 **注**：如果关闭 Bandgap，则只能使用 ILRC/T16/TM2 和 I/O 功能。

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	PA4 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 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	0	只写	使能 PA7 数字输入和唤醒事件。1/0：启用/ 停用 如果 PA7 位设 0 可停用唤醒。
6	0	只写	使能 PA6 数字输入和唤醒事件。1/0：启用/ 停用 如果 PA6 位设 0 可停用唤醒。
5	0	只写	使能 PA5 数字输入、唤醒事件和中断请求。1 / 0：启用/ 停用。 当 PA5 作为 AD 输入时，该位设为 0 可以防止耗电。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	00	只写	使能 PA4-PA3 数字输入和唤醒事件。1/0：启用/ 停用 当 PA3 作为比较器输入时，该位设为 0 以防止漏电流。如果这个位设为 0，PA4-PA3 作为模拟输入，则不能用来唤醒系统。
2	-	-	保留（建议写 00）
1 - 0	00	只写	使能 PA0 数字输入、唤醒事件和中断请求。 1 / 0：启用/ 停用。 该位可以设置为低，以禁用 PA0 触发的唤醒和来自该引脚的中断请求。如果该位设置为低，PA0-PA1 为模拟输入，不能用于唤醒系统，则该引脚的中断也被禁用。

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 A 下拉控制寄存器 (*papl*)，IO 地址 =0x0E

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

6.15.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（不适用于比较器唤醒功能） 011：$V_{internal\ R}$ 100：PA6 101：PA7 11X：保留
0	0	读/写	选择比较器正输入的来源。 0：$V_{internal\ R}$ 1：PA4

6.16.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 （请避免这种情况：在 ICE 中选择 PA0 输出时，GPCS 将影响 PA3 输出功能。）
6	0	只写	比较器唤醒启用。（gpcc.6 发生电平变化时才可唤醒） 0/1：停用/启用
5	0	只写	选择比较器参考电压 $V_{internal\ R}$ 最高的范围。
4	0	只写	选择比较器参考电压 $V_{internal\ R}$ 最低的范围。
3 - 0	0000	只写	选择比较器参考电压 $V_{internal\ R}$。 0000（最低）~ 1111（最高）

6.17.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 或 IHRC *2（通过 code option TM2_source）（ICE 不支持 IHRC *2） 0011：保留 0100：ILRC 0101：比较器输出 0110：NILRC 0111：保留 1000：PA0（上升沿） 1001：~PA0（下降沿） 1010：保留 1011：保留 1100：PA4（上升沿） 1101：~PA4（下降沿） **注意：**在 ICE 模式下，如果为 Timer2 时钟选择了 IHRC，则发送给 Timer2 的时钟不会停止，当 ICE 处于停止状态时，Timer2 将继续计数。
3 - 2	00	读/写	Timer2 输出选择： 00：停用 01：保留 10：PA3 11：PA4
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。

6.18.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（通过 code option TM2_bit）（ICE 不支持 7 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。

6.19.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 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。

6.21.低电压检测控制寄存器 (*lvdc*), IO 地址 = 0x1e

位	初始值	读/写	描述
7 - 2	000000	WO	设置 LVD 电平：在 1.85~5V 范围内，增加 0.05V
1	-	-	保留
0	0	只读	LVD & V_{BAT} 之间的检测结果 0：V_{BAT} > LVD level 1：V_{BAT} < LVD level

6.22.LPWMG0 控制寄存器(*lpwmg0c*), IO 地址 = 0x20

位	初始值	读/写	描述
7	0	只写	GPC 控制 LPWMG0 输出：0/1：停用/启用。
6	-	只读	LPWMG0 生成器输出状态。
5	0	只写	选择 LPWMG0 的输出的结果是否反极性。0/1：停用/启用。
4	0	只写	LPWMG0 计数器清零。 0：LPWMG0 输出 1：LPWMG0 XOR LPWMG1 或 LPWMG0 OR LPWMG2（通过 lpwmg0c.0）
3 - 1	0	只写	选择 LPWMG0 输出： 000：不输出 001：PA1 010：PA5 011：PA0 1xx：保留
0	0	只写	LPWMG0 时钟源。 0：LPWMG0 XOR LPWMG1 1：LPWMG0 OR LPWMG1

6.23. LPWMG 时钟寄存器 (l*pwmgclk*), IO 地址 = 0x21

位	初始值	读/写	描述
7	0	只写	LPWMG 开关。 0：LPWMG 关闭 1：LPWMG 打开
6 - 5	0	只写	LPWMG 时钟预分频 000: ÷1 001: ÷2 010: ÷4 011: ÷8 100: ÷16 101: ÷32 110: ÷64 111: ÷128
3 - 1	-	-	保留
0	0	只写	LPWMG 时钟分频 0：系统时钟 1：IHRC or IHRC*2（由 code option：PWM_Source 决定）

6.24.LPWMG0 占空比高位寄存器(*lpwmg0dth*), IO 地址 = 0x22

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

6.25.LPWMG0 占空比低位寄存器(*lpwmg0dtl*), IO 地址 = 0x23

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

注意：LPWMG0 占空比低位寄存器的值必须写在 LPWMG0 占空比高位寄存器之前。

6.26.LPWMG0 计数上限高位寄存器 (*lpwmgcubh*), IO 地址 = 0x24

位	初始值	读/写	描述
7 - 0	-	只写	PWMG0 上限寄存器 Bit[10:3]。

6.27.LPWMG0 计数上限低位寄存器 (*lpwmgcubl*), IO 地址 = 0x25

位	初始值	读/写	描述
7 - 6	00	只写	LPWMG0 上限寄存器 Bit[2:1]。
5 - 0	-	-	保留。

6.28.LPWMG1 控制寄存器 (*lpwmg1c*), IO 地址 = 0x26

位	初始值	读/写	描述
7	0	读/写	GPC 控制 LPWMG1 输出： 0/1：停用/启用。
6	-	只读	LPWMG1 生成器输出状态。
5	0	只写	选择 LPWMG1 的输出的结果是否反极性： 0/1：停用/启用。
4	0	读/写	LPWMG1 输出选择 0: LPWMG1 1: LPWMG2
3 - 1	0	读/写	选择 LPWMG1 输出： 000：不输出 001：PA6 010：保留 011：PA4 1xx：保留
0	0	读/写	保留

6.29.LPWMG1 占空比高位寄存器(*lpwmg1dth*), IO 地址 = 0x28

位	初始值	读/写	描述
7 - 0	0x00	只写	LPWMG1 上限寄存器 Bit[10:3]。

6.30.LPWMG1 占空比低位寄存器 (*lpwmg1dtl*), IO 地址 = 0x29

位	初始值	读/写	描述
7 - 5	00	只写	LPWMG1 上限寄存器 Bit[2:0]。
4 - 0	-	-	保留

注意：LPWMG1 占空比低位寄存器的值必须写在 LPWMG1 占空比高位寄存器之前。

6.31.LPWMG2 占空比高位寄存器 (*lpwmg2dth*), IO 地址 = 0x2E

位	初始值	读/写	描述
7 - 0	0x00	只写	LPWMG2 占空比值 bit[10:3]。

6.32. LPWMG2 占空比低位寄存器 (*lpwmg2dtl*), IO 地址 = 0x2F

位	初始值	读/写	描述
7 - 5	000	只写	LPWMG2 占空比值 bit [2:0]。
4 - 0	-	-	保留。

注意：LPWMG2 占空比低位寄存器的值必须写在 LPWMG2 占空比高位寄存器之前。

6.33. LPWMG2 控制寄存器 (*lpwmg2c*), IO 地址 = 0x2C

位	初始值	读/写	描述
7	0	读/写	GPC 控制 LPWMG2 输出： 0/1：停用/启用。
6	-	只读	LPWMG2 生成器输出状态。
5	0	读/写	选择 LPWMG2 的输出的结果是否反极性： 0/1：停用/启用。
4	0	读/写	LPWMG2 输出选择。 0：LPWMG2 1：LPWMG2 ÷2
3 - 1	0	读/写	选择 LPWMG2 输出： 000：停用 001：保留 010：保留 011：PA3 100：保留 101：PA5 1xx：保留
0	0	读/写	保留

6.34. 充电电流控制寄存器 (*chg_ctrl*), IO 地址 = 0x34

位	初始值	读/写	描述
7 - 5	011	读/写	000：500mA 001：400mA 010：350mA 011：300mA 100：250mA 101：200mA 110：100mA 111：50mA
4 - 1	-	-	保留
0	0	只读	充电工作指示位，当读到 0，充电器开始充电。（PMB180(B)_不支持） 0：开始充电或是充电中 1：充电完成或是停止充电

6.35. 充电状态寄存器 (*chg_temp*), IO 地址 = 0x35

位	初始值	读/写	描述
7 - 5	-	-	保留
4	-	只读	Vcc 电压源的状态指示位，可用于判断充电 Vcc 的状态 1：Vcc > V_{BAT} 0：Vcc < V_{BAT}
3	-	只读	充电动作指示位 1：Vcc 电压正常 0：Vcc 电压过低关闭充电器
2	-	-	保留
1	-	读/写	PMB180： 保留，只能設成 0。(新版 IDE 不再支援 OTP_100 設定) PMB180B： 充电完成指示位 (此功能限用于 PMB180B) (当充电电流降低于 1/10，关闭充电，chg_temp.1 自动设为 1) 读"1" ：充电完成指示灯。(充电电流小于 1/10，关闭充电) 读"0" ：充电进行中。
0	-	读/写	过温充电保护位（温度不超过 140） **写 1。**读到"0"触发过温保护，读"1"正常

6.36. 充电器充电电压校准寄存器 (*chg_trim*), IO 地址 = 0x32

位	初始值	读/写	描述
7 - 3	0xF	只写	充电器充电电压校准位。 0b_1111_1:Vbat=max ； 0b_0000_0:Vbat=min 在执行完 “.Adust_IC” 宏指令会自动填入校正参数值，不建议用户自行任意改动。
2	0	只写	充电器 CV Mode Vbat 电压输出，0/1：停用/启用。 (仅供验证测试使用，VCC >= +5V，不建议用户自行改动。)
1	0	只写	保留，写入值需为 0
0	0	只读	保留，写入值需为 0

注意：

(1) 用户若想在程序中动态微调整充电器的充电电压，或是将 **chg_trim** 重新写入校正参数值，可以使用 "ReLoad_VbatBGTRIM” 这一个宏指令。

(2) 使用宏指令”**Charger_CVMode”**，充电器将切换至 CV Mode。CV Mode 仅供 VBAT 引脚在空载未接锂电池时量测最终恒压充电电压值。

(3) 使用宏指指令 ”**Charger_CCMode”**或是”**ReLoad_VbatBGTRIM”**，充电器将切换回 CC Mode。

(4) 宏指令 ”.Adjust_IC” 支持对充电器充电电压校正参数微调整，只需加入下列命令 “REG:CHG_TRIM = -0x08”;

//chg_cur[7:0] = CP_TrimParameter – 0x08;

Example:

.ADJUST_IC SYSCLK=ILRC (IHRC/16), IHRC=16MHz, VDD=4.2V O_WDRST, X_FIRST, REG:CHG_CUR = -0x08;

(5) 充电器校正值(chg_trim)若被用户改动，在烧录器下载烧录檔时将会出现 “The charger trim value of this PDK file has been modified”的提示讯息弹窗。

6.37. 充电器充电电流校准寄存器 (*chg_cur*), IO 地址 = 0x33

位	初始值	读/写	描述
7 - 4	0x8	只写	充电器充电电流校准位。 0b_1111:Vchg_cur = min ； 0b_0000:Vchg_cur = max 在执行完 “.Adust_IC” 宏指令会自动填入校正参数值，不建议用户自行任意改动。
3 - 0	0	只写	保留，写入值需为 0

注意：

(1) 用户若想在程序中动态微调整充电器的充电电流，或是将 **chg_cur** 重新写入校正参数值，可以使用”ReLoad_ChargerCURTRIM” 这一个宏指令。

(2) 宏指令”.Adjust_IC”支持对充电器充电电流校正参数微调整，只需加入下列命令 “REG:CHG_CUR = -0x10”;

//chg_cur[7:0] = CP_TrimParameter – 0x10;

Example:

.ADJUST_IC SYSCLK=ILRC (IHRC/16), IHRC=16MHz, VDD=4.2V O_WDRST, X_FIRST, REG:CHG_CUR = -0x10;

(3) 充电器校正值(chg_cur)若被用户改动，在烧录器下载烧录檔时将会出现 “The charger trim value of this PDK file has been modified”的提示讯息弹窗。

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（逻辑补数，1 补数）
∓	负数（2 补码）
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)
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	移动即时数据到累加器。 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存储器。 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* pa, a; 结果: pa ← 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 PMB180(B)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 PMB180(B)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. 算数运算类指令

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	将累加器的负逻辑（2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: Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
nadd M, a	将RAM的负逻辑（2补码）与累加器相加，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: Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
sub a, I	累加器减立即数据，然后把结果放入累加器。 例如： *sub* a, 0x0f; 结果： a ← a - 0fh (a + [2’s complement of 0fh]) 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
sub a, M	累加器减 RAM，然后把结果放入累加器。 例如： *sub* a, MEM ; 结果： a ← a - MEM (a + [2’s complement of M]) 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』

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

7.3. 移位运算类指令

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

7.4. 逻辑运算类指令

and a, I	累加器和立即数据执行逻辑 AND，然后把结果保存到累加器。 例如：*and* a, 0x0f ; 结果：a ← a & 0fh 受影响的标志位： Z:『受影响』， C:『不变』， AC:『不变』， OV:『不变』
and a, M	累加器和 RAM 执行逻辑 AND，然后把结果保存到累加器。 例如：*and* a, RAM10 ; 结果：a ← a & RAM10 受影响的标志位： Z:『受影响』， C:『不变』， AC:『不变』， OV:『不变』
and M, a	累加器和 RAM 执行逻辑 AND，然后把结果保存到 RAM。 例如： *and* MEM, a ; 结果： MEM ← a & MEM 受影响的标志位： Z:『受影响』， C:『不变』， AC:『不变』， OV:『不变』
or a, I	累加器和立即数据执行逻辑 OR，然后把结果保存到累加器。 例如： *or* a, 0x0f ; 结果： a ← a | 0fh 受影响的标志位： Z:『受影响』， C:『不变』， AC:『不变』， OV:『不变』
or a, M	累加器和 RAM 执行逻辑 OR，然后把结果保存到累加器。 例如：*or* a, MEM ; 结果：a ← a | MEM 受影响的标志位： Z:『受影响』， C:『不变』， AC:『不变』， OV:『不变』
or M, a	累加器和立即数据执行逻辑 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』
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 的内容。 例如： *comp a, MEM;* 结果： 等效于(a - MEM)，并改变标志位 Flag。 受影响的标志位: Z:『受影响』, C:『受影响』, AC:『受影响』, OV:『受影响』 应用范例： --- *mov a, 0x38 ;* *mov mem, a ;* *comp a, mem ; // Z 标志被设为 1* *mov a, 0x42 ;* *mov mem, a ;* *mov a, 0x38 ;* *comp a, mem ; // C 标志被设为 1* ---
comp M, a	比较累加器和 RAM 的内容。 例如： *comp MEM, a;* 结果：等效于(MEM - a)，并改变标志位 Flag。 受影响的标志位： Z:『受影响』, C:『受影响』, AC:『受影响』, OV:『受影响』

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』
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
...
```

set0 M.n	RAM 的位 N 设为 0。 例如：*set0* MEM.5 ; 结果： MEM 位 5 为 0 受影响的标志位： Z:『不变』， C:『不变』， AC:『不变』， OV:『不变』
set1 M.n	RAM 的位 N 设为 1。 例如： *set1* MEM.5 ; 结果： MEM 位 5 为 1 受影响的标志位： Z:『不变』， C:『不变』， AC:『不变』， OV:『不变』

7.6. 条件运算类指令

ceqsn a, I	比较累加器与立即数据，如果是相同的，即跳过下一指令。标志位的改变与 (a ← a - I)相同 例如： *ceqsn* a, 0x55 ; *inc* MEM ; *goto* error ; 结果： 假如 a=0x55, then “goto error”; 否则, “inc MEM”. 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
ceqsn a, M	比较累加器与 RAM，如果是相同的，即跳过下一指令。标志位改变与 (a ← a - M)相同 例如： *ceqsn* a, MEM; 结果： 假如 a=MEM，跳过下一个指令 受影响的标志位： Z:『受影响』， C:『受影响』， AC:『受影响』， OV:『受影响』
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	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, pcadd, ret, reti , idxm*
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. 位定义

位寻址只能定义在 RAM 区地址的 0x00 to 0x3F。

8. 程序选项

选项	选择	描述
保护	启用	OTP 内容加密，程序不允许被读取
	禁用	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
	2.7V	选择 LVR = 2.7V
	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
GPC_TM2	启用	GPC 不控制 TM2 输出
	禁用	GPC 控制 TM2 输出 （ICE 不支持）
LPWM_源	16MHZ	当 Lpwmgclk.0= 1, LPWMG 时钟源= IHRC = 16MHZ
	32MHZ	当 Lpwmgclk.0= 1, LPWMG 时钟源= IHRC*2 = 32MHZ （ICE 不支持）
TM2_源	16MHZ	tm2c[7:4]= 0010, TM2 时钟源= IHRC = 16MHZ
	32MHZ	tm2c[7:4]= 0010, TM2 时钟源= IHRC*2 = 32MHZ （ICE 不支持）
TM2_Bit	6 Bit	tm2s.7=1, TM2 PWM 分辨率 6 位
	7 Bit	tm2s.7=1, TM2 PWM 分辨率 7 位 （ICE 不支持）
比较器_边缘	全部边缘	比较器在上升沿或下降沿触发中断
	上升边缘	比较器在上升沿触发中断
	下降边缘	比较器在下降沿触发中断

9. 特别注意事项

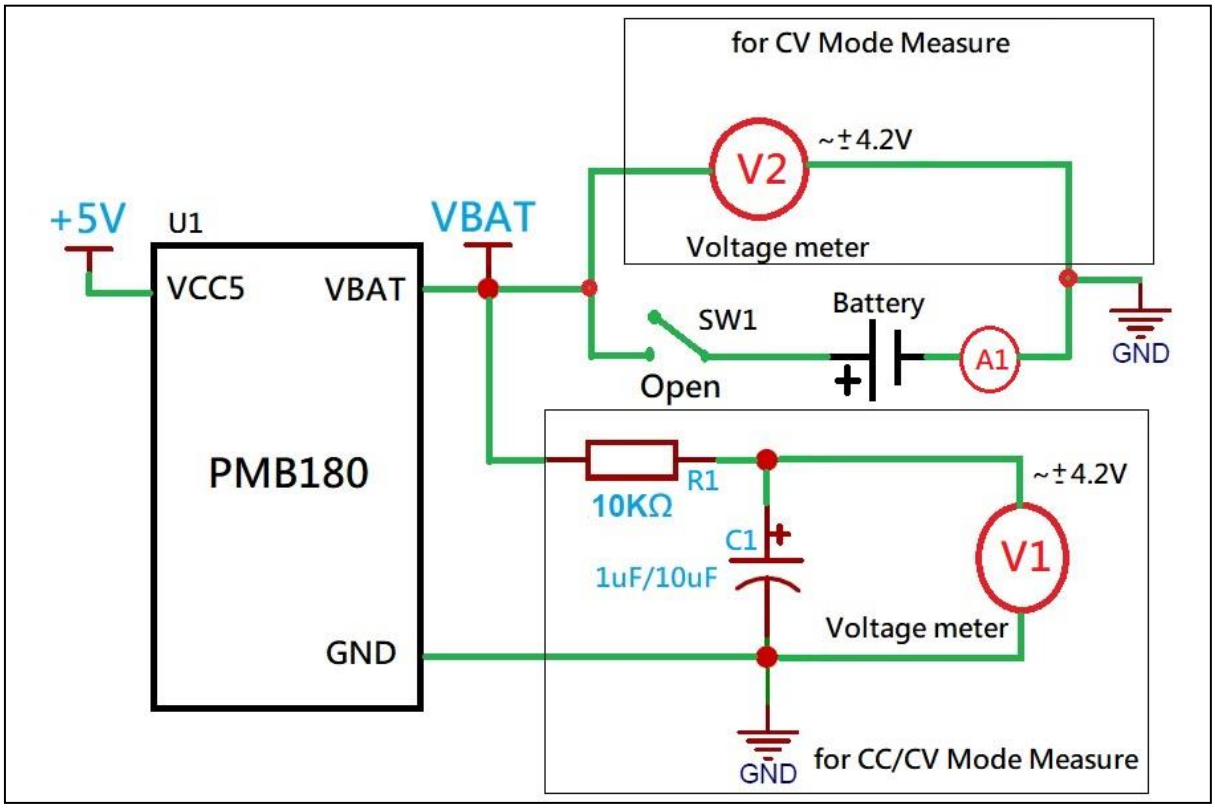
此章节提醒用户在使用 PMB180(B)系列 IC 时避免常犯的一些错误。

9.1. 使用 IC

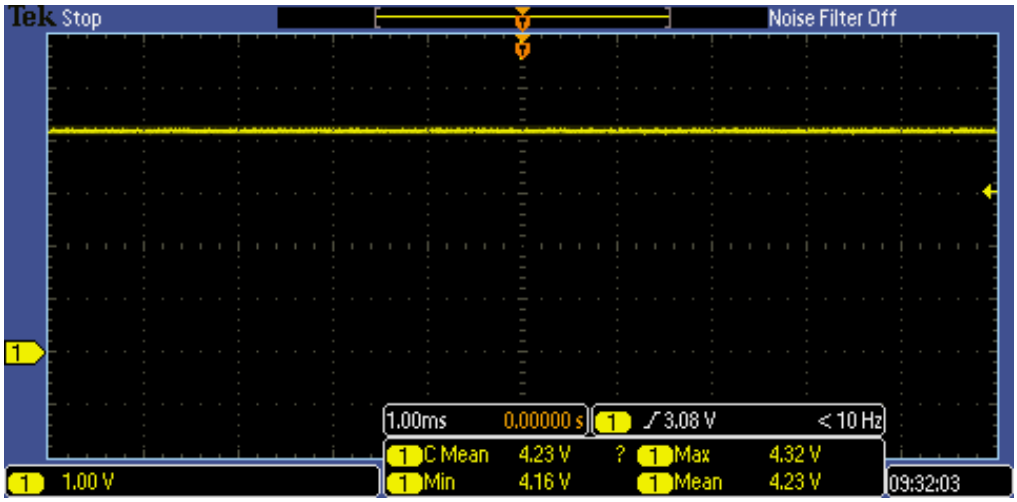
9.1.1. 充电器使用与设定

(1) 充电电压及电流量测

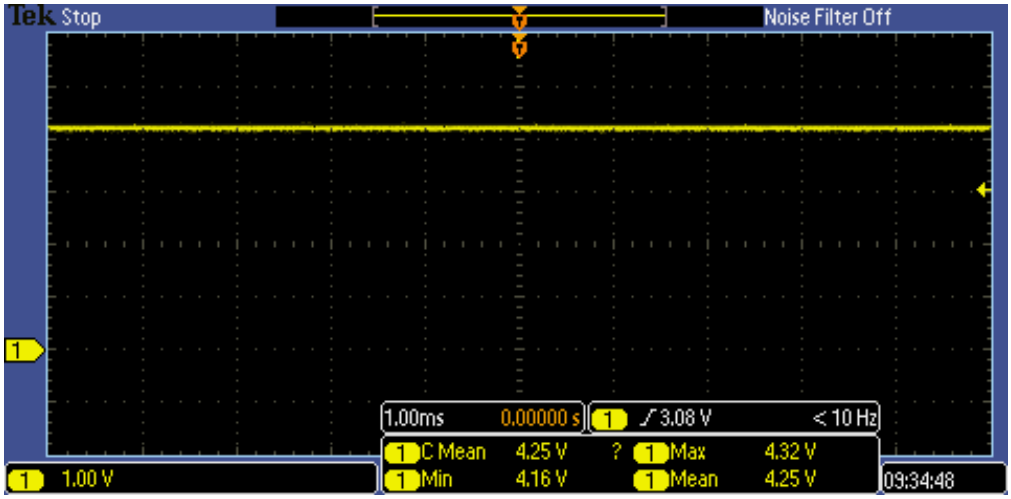
- PMB180(B) 充电器与 MCU 执行程序是个别独立工作, MCU 复位并不影响到充电动作。
- 对未写入程序的 PMB180(B)芯片做充电电压及电流的量测，将会得到未被校准过的充电电压及充电电流。(因为充电器的电压及电流校准寄存器未被填入精准的校正参数)
- PMB180(B)在程序正常启动工作后将会为充电器写入校正参数，此时可对 PMB180(B)的充电器做电性量测。
- 量测 PMB180(B)充电器的充电电压接线图如下示意：在 CC Mode 下需在 VBAT 引脚串联 RC 电路(仿真电池内阻等效电路)，电压表 V1 并接于电容器 C1 上，在 CV Mode 下可将电压表 V2 并联接于 VBAT 引脚。



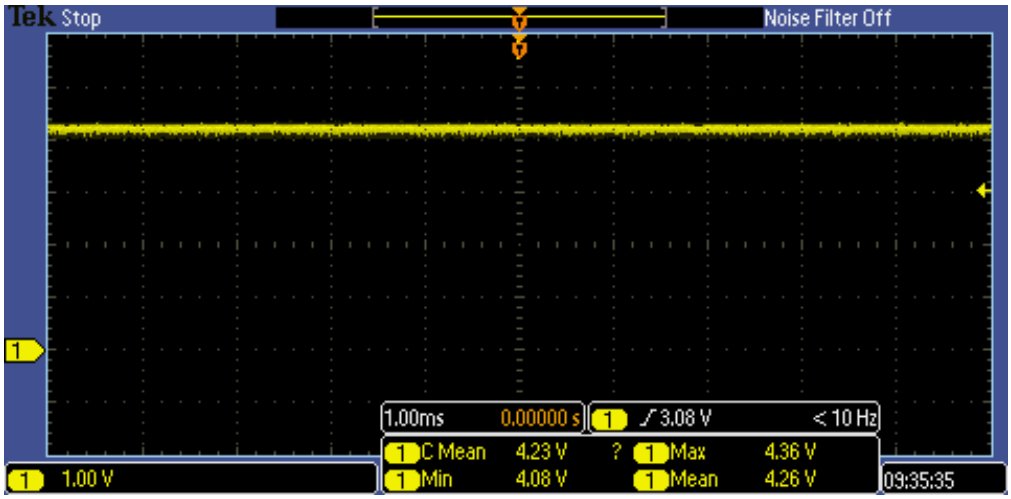
V1 电压波形：(CC Mode, R1 = 10Kohm, C1 = 1uF)



V1 电压波形：(CV Mode, R1 = 10Kohm, C1 = 1uF)

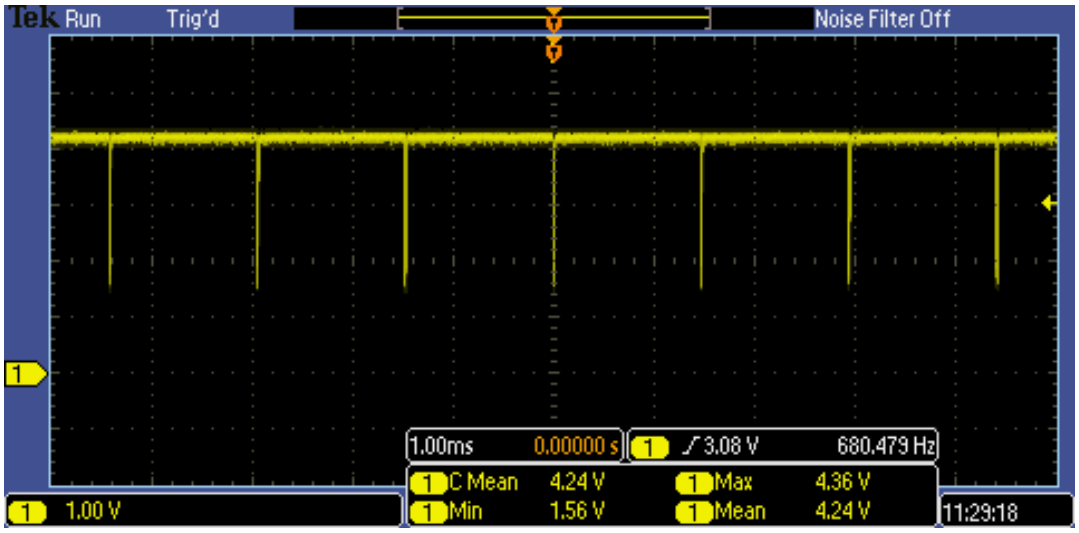


V2 电压波形：(CV Mode, No R1 and C1)

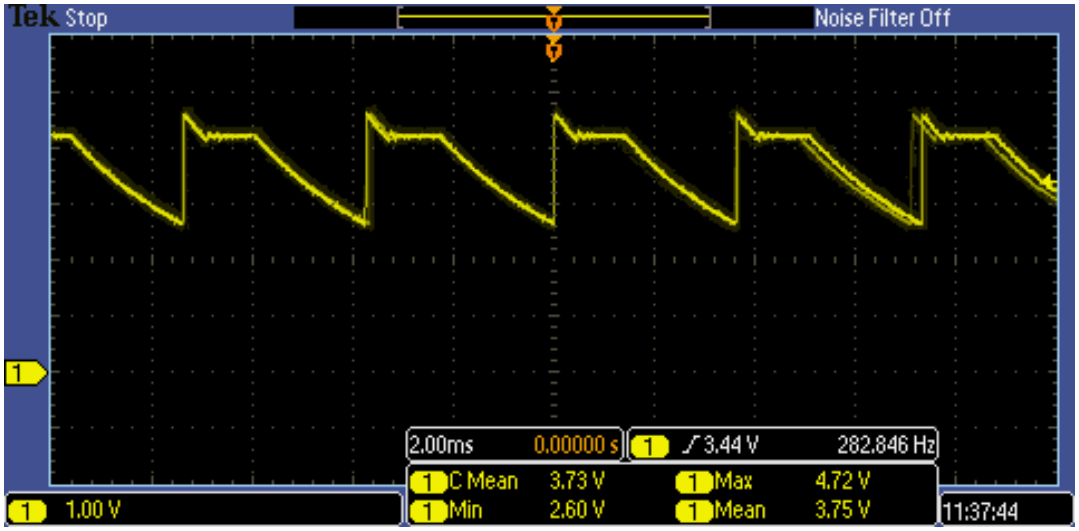


◆ 在 CC Mode 下若 VBAT 引脚空接或是只接上滤波电容 / 电解电容时所量到的电压可能会因为充电器对外部电容的充放电效应，使得平均电压偏低。充电器在充电至 4.2V 满电后的间隔循环充电时间约在 1~3ms，此时若 VBAT 引脚未接电池或是只接电容，将会因为放电时间导致电压表量到的平均值变低。例如使用电压表直接量测 VBAT-Pin 电压时，且 VBAT 引脚只有并联 1uF 电容反而会造成量测错误(远低于目标值)，因为放电时间会变长，导致电压表量到较低的电压值。Vbat 电压波形如下图波形。

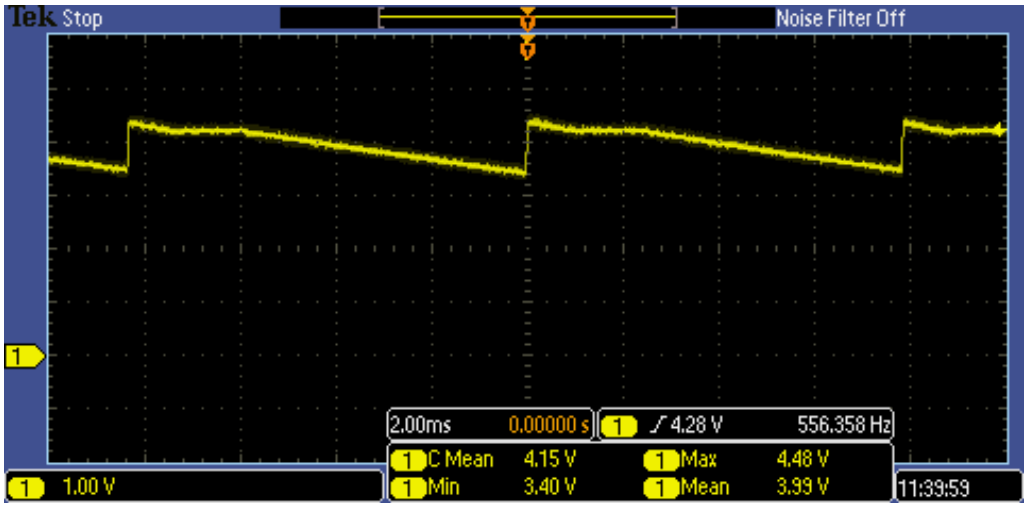
V2 电压波形：(CC Mode, No R1 and C1)



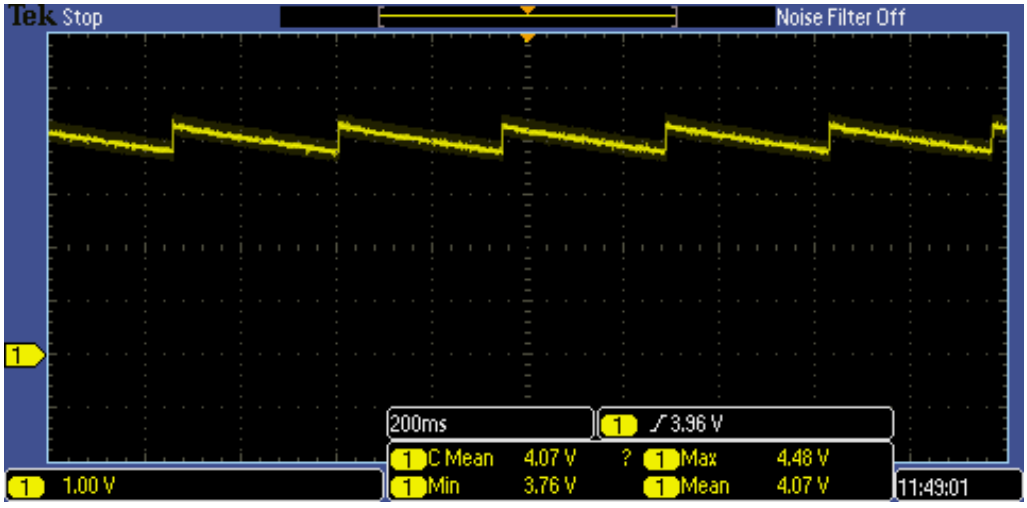
V2 电压波形：(CC Mode, R1 = 0R , C1 = 1uF)



V2 电压波形：(CC Mode, R1 = 0R , C1 = 4.7uF)



V2 电压波形：(CC Mode, R1 = 0R , C1 = 470uF)



- IDE 0.99D8 之后的版本有提供数个针对充电器校正参数修改的进阶宏参数指令，一般不建议用户随意更改。有相关疑虑可咨询原厂或是代理商的相关技术工程人员。充电器校正值若被用户改动，在烧录器下载烧录檔时将会出现 “The charger trim value of this PDK file has been modified” 的提示讯息弹窗。

9.1.2. IO 引脚的使用和设定

(1) IO 作为数字输入时

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

(2) IO 作为数字输入和打开唤醒功能

- 设置 IO 为输入
- 用 PADIER 寄存器，将对应的位设为 1
- 对于未使用的 PA 引脚，应将 PADIER[1:2]设置为低，以防止其泄漏。

(2) 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

(3) 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 的电阻
- 应尽量避免使用 PA5 作为输入

9.1.3. 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 和 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。

9.1.4. 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
```

9.1.5. 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。

9.1.6. 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。

9.1.7. IHRC

(1) IHRC 频率会在使用烧录器烧录程序时校准。

(2) 校准 IHRC 时，不管是封装片机台刻录还是 COB 在线刻录，EMC 的干扰都会对 IHRC 的精度有影响。如果在封装前校准了 IHRC，那么在封装后 IHRC 的实际频率可能会出现偏差并超出规格范围。通常封装后频率会变慢一点。

(3) 频率偏离较大的情况一般发生在 COB 刻录或者 QTP 时。应广科技对这种情况不担负责任。

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

9.1.8. 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	V_{BAT}	LVR
2MHz	≧1.8V	≧ 1.8V
4MHz	≧ 2.2V	≧ 2.2V
8MHz	≧ 2.7V	≧ 2.7V

表 9：LVR 设置参考

(1) 只有当 IC 正常起动后，设定 LVR（1.8V ~ 4.5V）才会有效。

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

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

9.1.9. 烧录方法

PMB180(B)的烧录脚为 PA4，PA6，V_{BAT} 和 GND 这 4 个引脚。

请使用 5S-P-003x 或以后的版本烧录 PMB180(B)实际芯片（3S-P-002 或之前的版本皆以不支持烧录该芯片）。

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

(1) V_{BAT} 可能高于 9.5V，而最大供给电流最高可达约 20mA。

(2) 其他烧录引脚（GND 除外）的电位与 V_{BAT} 相同。

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

重要注意事项：

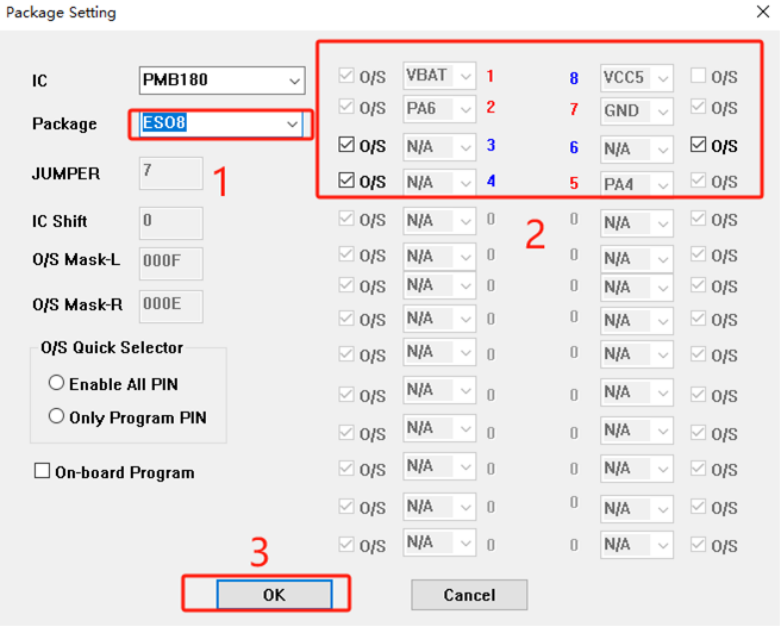
- **您必须按照 APN004 和 APN011 上的说明在处理器上编程 IC。**
- **在处理器端口的 V_{BAT} 和 GND 之间连接 0.01uF 电容器到 IC 有利于抑制干扰。但请不要连接＞0.01uF 的电容器，否则，编程可能会失败。**

9.1.9.1. 5S-P-003 烧录 PMB180(B)方法

使用 5S-P-003 烧录 PMB180(B)，使用 Jumper7 转接程序信号。信号的连接取决于 IC 封装。请参阅 Writer 用户手册的第 5 章，为目标 IC 封装制作 Jumper7 转接板。用户可以从以下网页链接获取用户手册。

http://www.padauk.com.tw/en/technical/index.aspx?kind=27

从 GUI 加载 PDK，插入 JP7，然后在插座上插入 IC，无需移位。LCDM 显示 IC ready 后，可以烧录。



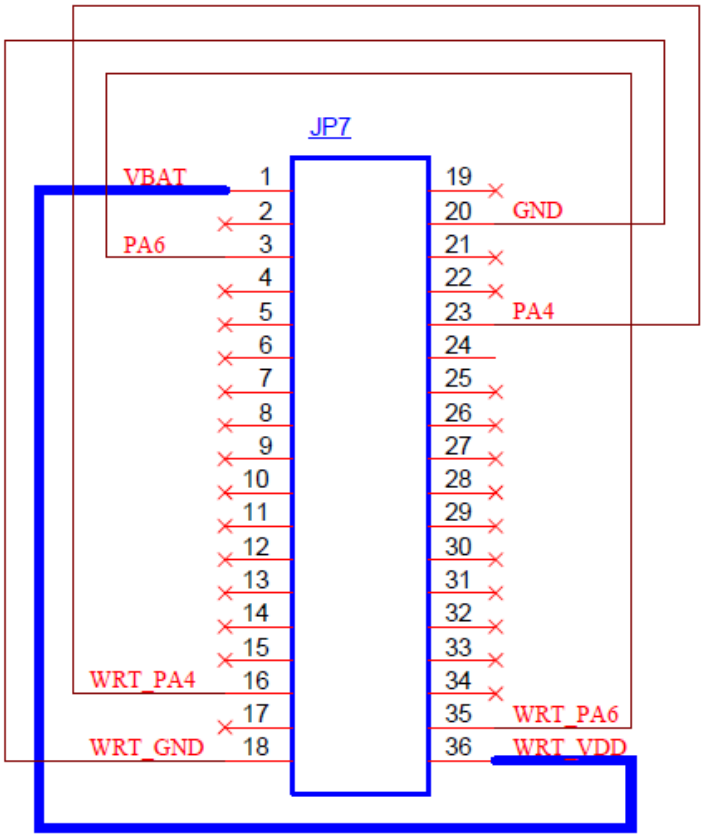
注意：烧录器上不支持 VCC5 脚位的 O/S Test

另封装脚位（自）定义亦可直接于程序中定义：

```
////////////////////////////
// program pin for ESOP10
////////////////////////////
//   VBAT  1    10  VCC5
//         2    9   GND
//   PA6   3    8
//         4    7
//         5    6   PA4
////////////////////////////

// package      pin_cnt   vbat pa0 pa3 pa4 pa5 pa6 pa7 gnd vcc5  agnd   mask1  mask2  shift  option
.writer package 10,         1,  0,  0,  6,  0,  3,  0,  9,  10,   0,     0x1F,  0x1E,  0,     0
```

以 ESSOP-10 为例，转接板接线如下：



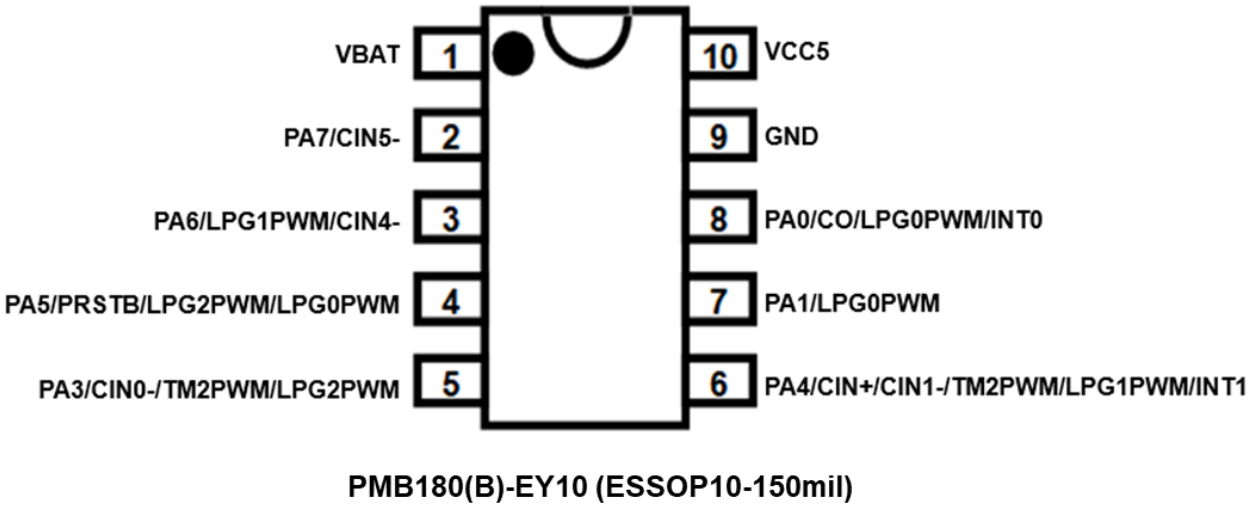


图 20：JP7 跳线原理图 for P003B

芯片顶格放入正面插座，不要移位，LCDM 显示 IC ready 就绪后，可以烧录。

9.1.9.2. 5S-P-003 烧录 PMB180(B)方法

5S-P-003 和 5S-P-003B 以类似的方法烧录 PMB180(B)，但用户应注意以下事项。

1. PDK 转档

IDE 连接烧录器后点击 convert to package，打开待烧 PDK 进入 package setting 页面，在 package 选项选择带有[P003]后缀的封装，确认勾选 V_{BAT} /PA5 swap 选项，确认 IC 脚位信息，保存并使用新生成的 PDK 文件。步骤参考图 21、图 22。

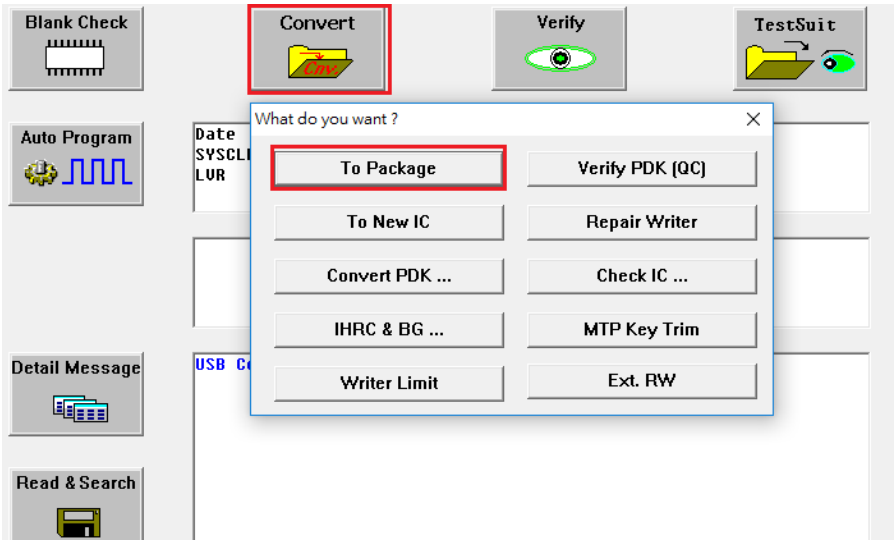


图 21：Convert PDK

注意：烧录器上不支持 VCC5 脚位的 O/S 测试

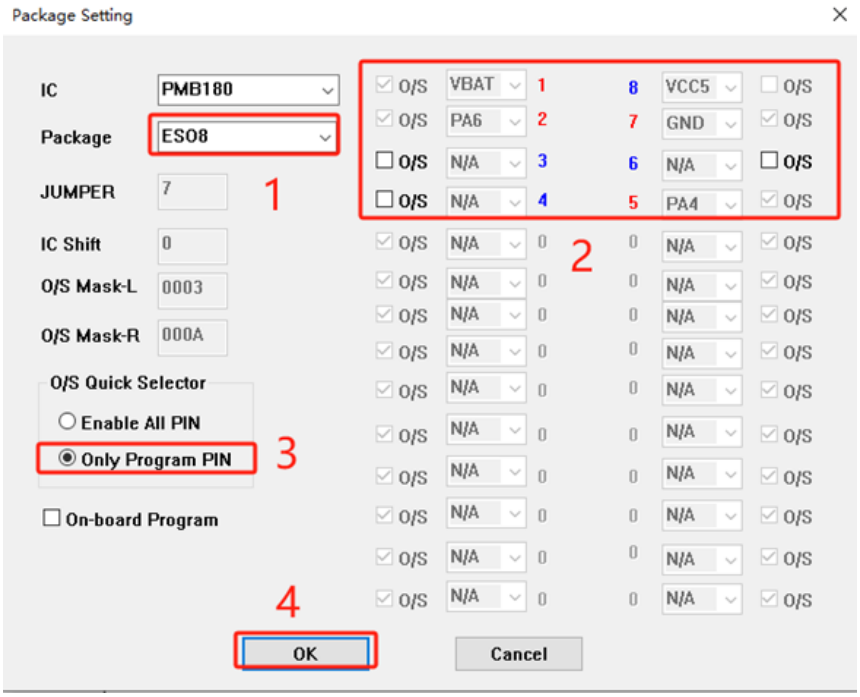


图 22：PMB180(B)-ESOP10 在 P003 转档配置

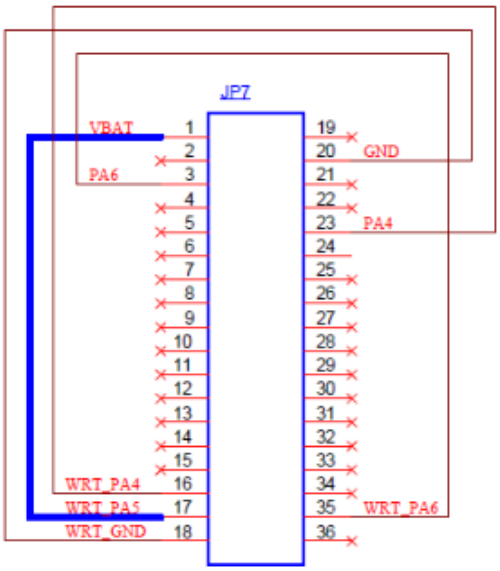
另 PDK 转文件信息亦可直接定义于程序中，如下：

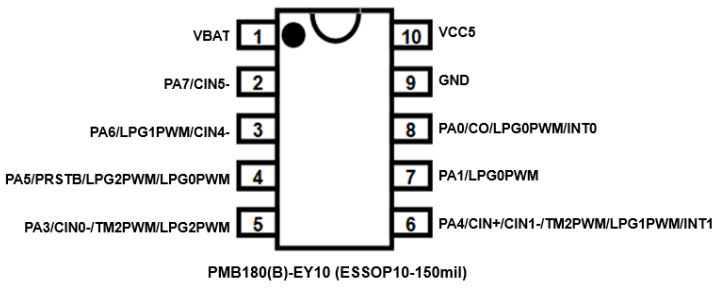
```
//////////////////////////////
// program pin for ESOP10
//////////////////////////////
//   VBAT  1    10  VCC5
//         2    9   GND
//   PA6   3    8
//         4    7
//         5    6   PA4
//////////////////////////////

// package      pin_cnt   vbat pa0 pa3 pa4 pa5 pa6 pa7 gnd vcc5  agnd   mask1  mask2  shift  option
.writer package 10,         1,  0,  0,  6,  0,  3,  0,  9,  10,   0,     0x05,  0x12,  0,     0x10

// option
//    bit2 - onboard
//    bit4 - vbat/vpp swap
//  others - reserved
```

2. 烧录器背面 JP7 跳线





注：烧录脚 PA5 必须连接 IC-VBAT

图 23：JP7 跳线原理图 for P003

3. 芯片顶格放入正面插座，不要移位，LCDM 显示 IC ready 就绪后，可以烧录。

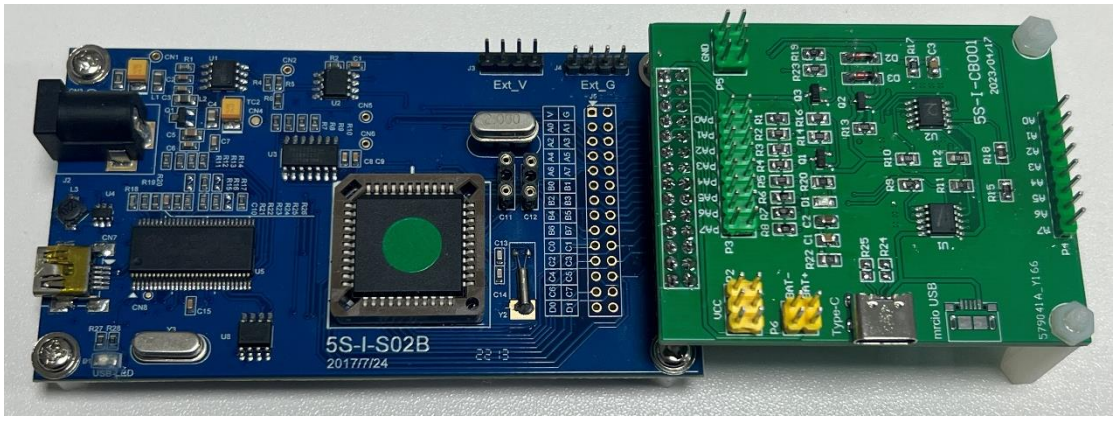
5S-P-003 以相同的方式烧录 PMB180(B)其他 IC 封装，并且都使用 V_{BAT}/PA5 swap。原理和步骤类似于 5S-P-003B 烧录 ESOP10（P003B）。请注意更改 To Package 设置和 jumper7 连接方法以与其他封装相对应，此处不再重复。

9.1.10. 应用手册

关于充电器更多的使用注意事项，请参阅 APN-021 文件说明。

9.2. 使用 ICE

5S-I-CB001 为应广科技针对 PMB180(B) 所推出的仿真工具，需要搭配应广科技的 IDE 软件做联机仿真，5S-I-CB001 使用时需搭配 5S-I-S01/2(B)使用，参考如下图片：



5S-I-CB001 仿真注意事项：

(1) 5S-I-CB001 必须搭配 5S-I-S01/2(B)系列仿真器方可做 PAPL/LPWM/CHARGE 等功能仿真

(2) 5S-I-CB001 电源由 5S-I-S01/2(B) 为了稳定电源请将 5S-I-S01/S02(B) 接上 DC9V 的电源调适器。

(3) IDE 版本 0.97E4 开始支持 5S-I-CB001 的仿真。

(4) 仿真充电、LPWM 和 PxPL 相关功能时，5S-I-S01/2(B)会和仿真板 5S-I-CB001 进行通信，所以仿真时许会慢略慢于实际 IC，主要影响以下寄存器配置 INTRQ/PAPL/LVDC/CHGC/CHGS/LPWMGCLK/LPWMGCUBH/LPWMGCUBL/LPWMGxC,LPWMGxDTH,LPWMGxDTL 等。

(5) 仿真时，ICE 和 5S-I-CB001 通信会使中断延后触发，延后时间约 0~335us。

(6) 使用 T16/TM2/TM3 等中断定时，中断定时不准。ICE 和 5S-I-CB001 通信会切换系统时钟，且会使中断延后触发，此现象常出现在定时器时钟源选择 SYSCLK 时或定时时间较短时。故，建议仿真时单个定时中断周期≥ 10ms。

(7) 仿真时，ICE 和 5S-I-CB001 通信需使用中断，因此不支持重烧 7 次做功能仿真。

(8) 仿真时，ICE 和 5S-I-CB001 不支持充电器 CV_Mode 切换、充电电压校正参数及充电电流校正参数动态调整功能。

使用 5S-I-S01/2(B)仿真器模拟时请注意以下几点：

(1) 5S-I-S01/2(B)仿真器不支持 11 位 SuLED 硬件生成器的功能。

(2) 5S-I-S01/2(B)仿真器不支持 PMB180(B)的 NADDD/COMP 指令。

(3) 5S-I-S01/2(B)仿真器不支持 PMB180(B)的 SYSCLK=ILRC/16。

(4) 5S-I-S01/2(B)仿真器不支持 PMB180(B)的 Tm2C.*gpcrs，PA4*。

(5) 5S-I-S01/2(B)不支持 EOSCR 内置电容器。

(6) 5S-I-S01/2(B)不支持 GPCC.N_PA6/N_PA7。

(7) 5S-I-S01/2(B)不支持 LVDC 和 OPR3。

(8) 5S-I-S01/2(B) 仿真器不支持 TM2 的 NILRC 时钟源仿真。

(9) 当 GPCS 选择输出到 PA0 输出时，PA3 输出功能将受到影响。

(10) 仿真 PWM 波形时，请在程序运行期间检查波形。当 ICE 暂停或单步运行时，其波形可能与实际情况不符。

(11) 5S-I-S01/2(B)仿真器的 ILRC 频率与实际 IC 不同，且未校准，频率范围约为 34K~38KHz。

(12) 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。

(13) 快速唤醒的时间有差异：5S-I-S01/2(B)：128 系统时钟，PMB180(B)：45 ILRC 周期。

(14) 看门狗溢出的时间和仿真器 5S-I-S01/2 有不同：

WDT 周期	5S-I-S01/2(B)	PMB180(B)
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}

9.3. 典型应用

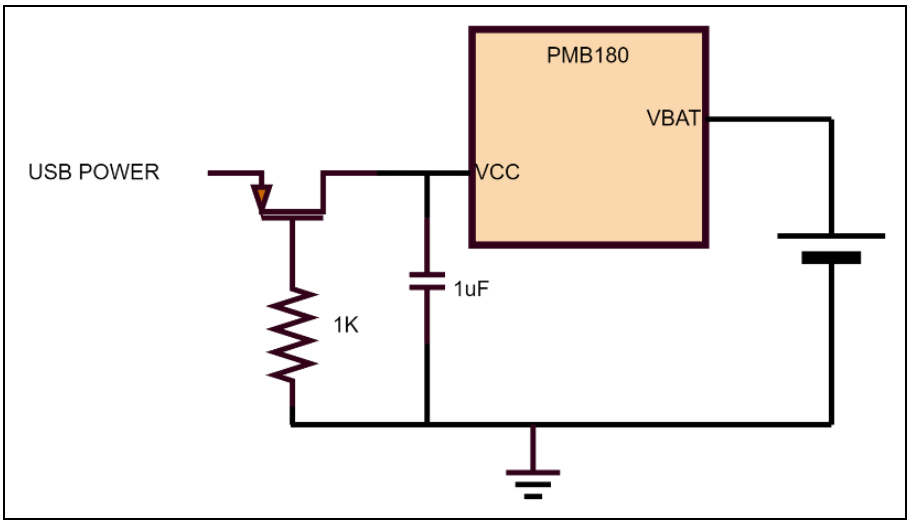


图 24：带反极性输入保护的基本锂离子充电器

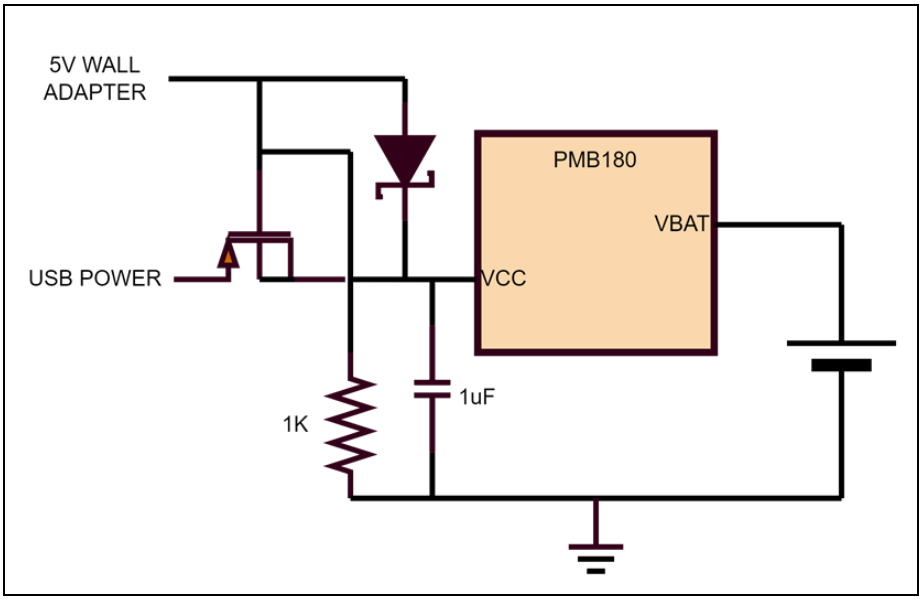


图 25：USB/墙壁适配器电源锂离子充电器

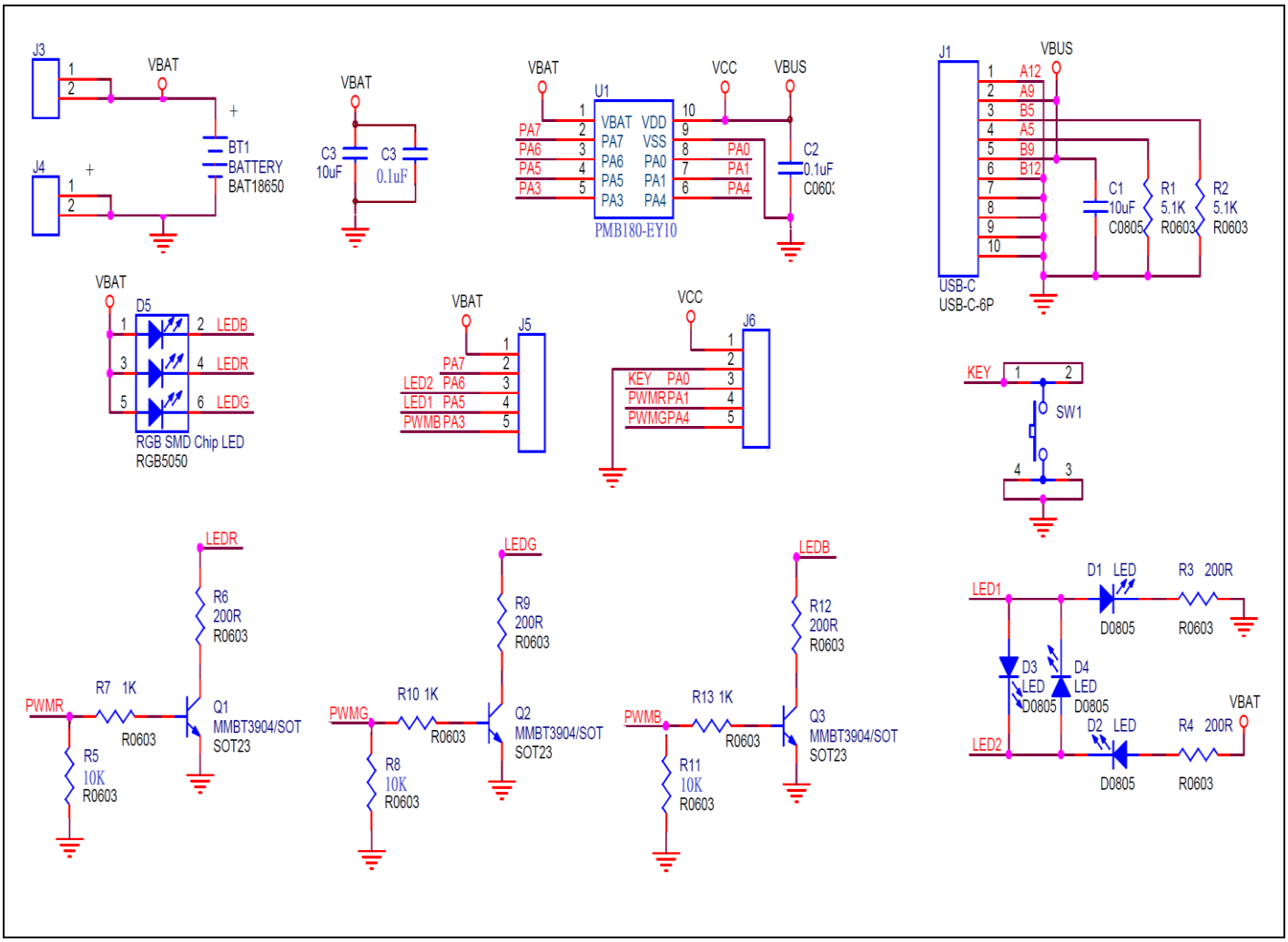


图 26：带锂离子充电灯光控制原理图

9.4. 提高 PMB180(B)芯片上电开机的稳定度

PMB180(B)应用板在生厂过程中常会有锂电池电极片与应用板需要透过点焊加工做连接。在此状况会造成应用板上的电源跳动不稳定，使得 PMB180(B)上电波形出现抖动干扰。此一过程有可能会持续数十至数百毫秒，这对于芯片上电开机是一严苛的挑战，有可能造成芯片出现死机、程序跑飞、功能异常、系统参数错乱…等状况。针对此一状况可在程序依照下列几点设置可提升 PMB180(B) IC 在电源上电及电源抖动时的稳定性。

(1) CodeOption 设置：

- CodeOption 开机速度选择 Slow。(若芯片有支持此功能)

(2) 开机程序在.Adjust_IC 前优先做 IO 输出低及延时设置：

- 上电开机程序以 ILRC 优先执行 IO 输出低 (在 .Adjust_IC 前)。
- IO 输出低后，以 ILRC 系统频率执行延时 (在 .Adjust_IC 前)。
- .Adjust_IC 的参数里加 x_first。
- 建议上电开机延时 100ms ~ 200ms，才进入 Stopexe Mode。

(3) 系统频率：

- 建议在 IC 上电开机后保持系统频率为 ILRC 工作，直到进入 Stopexe / Stopsys 模式。
- 建议 Stopexe / Stopsys 唤醒后系统频率才切换至高频工作。
- 建议系统时钟频率设定在 1MHz (含)以下对于稳定性提升有效果。
- 建议ILRC 可以永远保持 Enable 状态，从高频切低频时可以直接切换。
- 系统主频由 ILRC 切换至 IHRC 时，必须先 Enable IHRC，并且等后 2 个以上的指令的运行时间才可以切换。避免一行指令做完 Enable IHRC 及切换频率的动作。

```
//-------------------------------------------------------
// Macro Name: PowerOn_Delay
// Parameter: none
//
//-------------------------------------------------------
byte pndt;
PowerOn_Delay  macro              // delay 2048 ILRC period = 20.48ms
  PA = 0x00;
  PAC = 0xFF;                     // change to Output Mode
  pndt = 0xFF;
  do
   {     nop; nop; nop; nop; nop;   }
  while(pndt--);
  PAC = 0x00;                     // change to Intput Mode
endm
```

```
void  FPPA0 (void)
{
 #if  _SYS(AT_EV)
    $ MISC WDT_64K;                    // 64K*ILRC = 640ms
 #endif
  PowerOn_Delay     //delay 20ms
  .ADJUST_IC  SYSCLK=ILRC (IHRC/16), IHRC=16MHz, VDD=4.2V, O_WDRST, X_FIRST;
  .wdreset;
  .delay 7000    // delay 70ms for VDD = 5V
 //---- ReLoad_All_Param
  ReLoad_IHRC                          //Reload IHRCR Parameter
  ReLoad_ChargerCURTRIM    //Reload Charger Current Trim Bits
  ReLoad_VbatBGTRIM          //Reload Charger Vbat Trim Bits
  $ MISC WDT_64K;              // 64K*ILRC = 640ms
  .wdreset;
  $ CLKMD IHRC/16, En_IHRC, En_ILRC, En_WatchDog;

  while(1)
  {
     .delay 10000;
     $ PA.6 Out, High;
     .delay 10000;
     $ PA.6 Out, Low;
  }
}
```

(4) 看门狗：

- 保持看门狗为 Enable 的状态，不关闭看门狗。在.Adjust_IC 宏指令中加入”**O_WDTRST”字符串参数。**
- 进入 Stopexe Mode 时不必关闭看门狗，芯片硬件会自动关闭看门狗，并且在唤醒后自动重新启用看门狗，看门狗计数器也将一并会被清除。
- 看门狗清零指令执行周期建议不要加的太密集。建议在主程序中执行一次即可
- 若有使用中断，则不建议在中断程序中加清看门狗指令。
- 切换系统频率时需注意要保持看门狗开启状态，避免误关看门狗。
- 建议看门狗时间设成 64K 个 ILRC，约 640ms。考虑 ILRC 的频漂误差，清看门狗周期建议约 320ms。

(5) Stopexe/Stopsys 唤醒：

- 建议在 stopexe/Stopsys 唤醒后可执行 **ReLoad_IHRC / ReLoad_ChargerCURTRIM / ReLoad_VbatBGTRIM** 这三个宏指令，将系统校正参数寄存器重新回写入一次。