PFC232

工业级- 8 位 MTP 型单片机（FPPA™）
带 12 位增强型 ADC

数据手册

第 0.06 版

2025 年 12 月 10 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	2024/08/29	更新特性的叙述
0.06	2025/12/10	更新第 15 章节的叙述

使用警告

在使用 IC 前，请务必认真阅读 PFC232 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/tw/product/show.aspx?num=110&kw=PFC232

（下列图示仅供参考，依官网为主。）

◆◆ PFC232 ◆◆

- 高抗干擾High EFT 系列
- 工作溫度範圍： -40°C ~ 85°C

Feature Documents Software &Tools Application Note

內容	說明	中文下載	英文下載
APN001	ADC模擬信號源輸出阻抗應用須知		
APN002	過壓保護應用須知		
APN003	IO輸出引腳連接長導線時的應用須知		
APN004	半自動燒錄機台使用需知		
APN005	過電壓輸入對ADC的影響使用須知		
APN007	設置LVR時的使用須知		
APN011	半自動燒錄機台提高燒錄穩定性		
APN013	晶振使用須知		
APN017	提升IC在電源插拔測試下的抗干擾能力		
APN019	E-PAD 產品的PCB佈局指南		

1. 单片机特点

1.1. 系统特性

- 高抗干扰（High EFT）系列
- 工作温度范围：-40°C ~ 85°C
- ESD > 8 KV

1.2. 系统功能

- 2KW MTP 程序空间供两个 FPPA 单元使用（可编程 1000 次）
- 128 Bytes 数据空间供两个 FPPA 单元使用
- 一个硬件 16 位定时器
- 两个 8 位硬件 PWM 生成器
- 三个 11 位硬件 PWM 生成器
- 提供一个硬件比较器
- 提供一个运算放大器（OPA）
- 提供 1T 8×8 硬件乘法器
- 14 个 IO 引脚，有可选的上拉/下拉电阻
- 每个 IO 引脚都具有系统唤醒功能
- 对于每个设定唤醒功能的 IO，有两种可选择的唤醒速度：正常唤醒和快速唤醒
- 内部 Bandgap 电路提供 1.2V 参考电压
- 最大 12 信道 12 位 ADC，其中一个通道来自于内部 bandgap 参考电压或 0.25*VDD
- 提供 ADC 参考高电压选项：外部输入，内部 VDD，bandgap（1.2V），4V，3V，2V
- 时钟模式：外部晶体震荡器、内部高频振荡器、内部低频振荡器
- 内建 VDD/2 偏置电压产生器，可支持最大 5×9 点阵的 LCD 屏
- 8 段 LVR 复位设定，从 1.8V 到 4.5V
- 4 个外部中断输入脚

1.3. CPU 特点

- 工作模式：两个 FPPA™ 处理单元运作模式或单一处理单元运作模式
- 89 条高效的指令
- 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，用数据存储器即可当作间接寻址模式的数据指针(index pointer)
- 寄存器地址空间、数据存储空间、MTP 程序空间三者互相独立

1.4. 订购/封装信息

- PFC232-S08: SOP8 (150mil);
- PFC232-D08: DIP8 (300mil);
- PFC232-S14: SOP14 (150mil);
- PFC232-D14: DIP14 (300mil);
- PFC232-S16: SOP16 (150mil);
- PFC232-D16: DIP16 (300mil);

- 封装尺寸信息请参考官网文件：”封装信息“

2. 系统概述和方框图

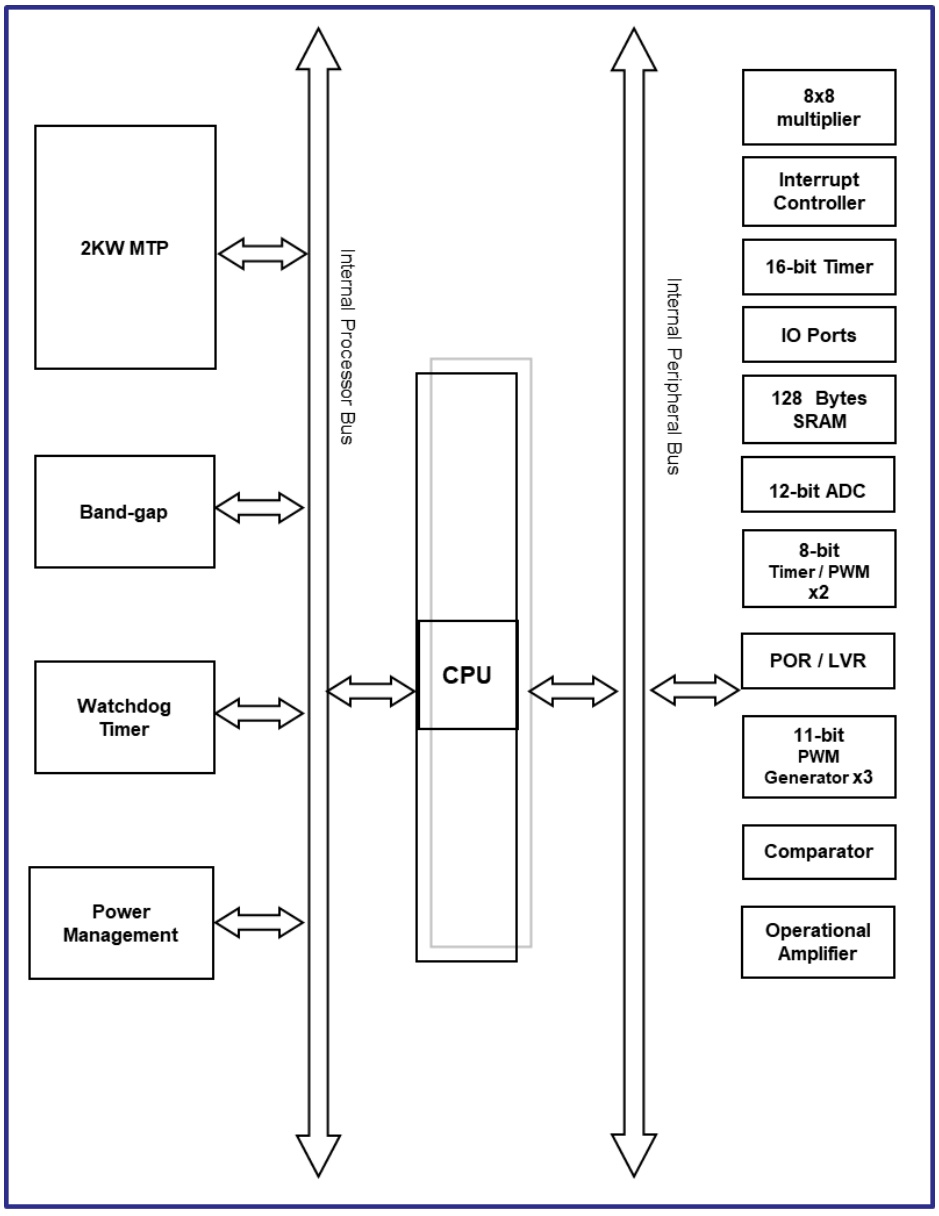
PFC232 是一个带 ADC、并行处理、完全静态，以 MTP 为程序存储基础的处理器，此处理器具有两个处理单元。它基于 RISC 架构基础，获得 （Field Programmable Processor Array 现场可编程处理器阵列）技术专利。大部分指令的执行周期都是一个指令周期，只有少部分间接寻址的指令需要两个指令周期。

PFC232 内置 2KW MTP 程序存储器以及 128 字节数据存储器，供两个 FPPA 单元工作使用。

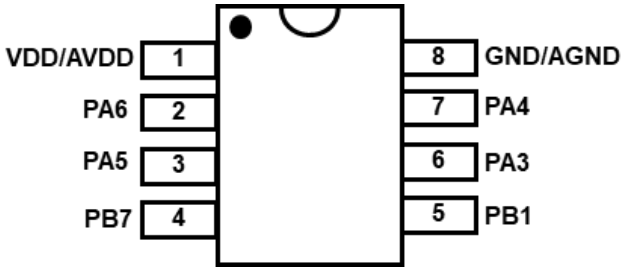
PFC232 内置 12 通道 12 位分辨率 A/D 转换器，其中一通道为内部 Bandgap 参考电压或 0.25*VDD。

PFC232 提供一个 16 位的硬件计数器(Timer16)、两个 8 位计数器(Timer2、Timer3)和 3 个 11 位计数器(PWMG0、PWMG1、PWMG2)。除 Timer16 之外，其余计数器都能产生 PWM 波形。

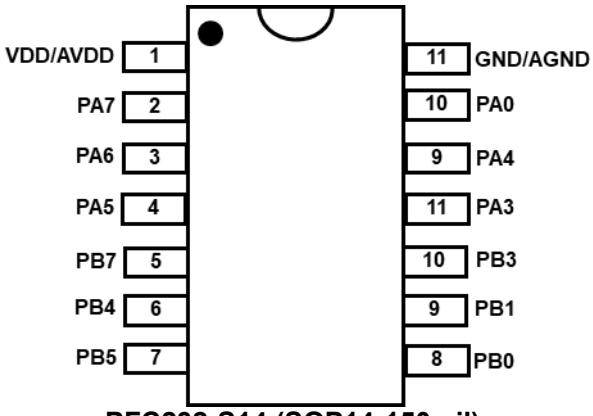
PFC232 还提供一个运算放大器（OPA）、硬件比较器、驱动 LCD 的 VDD/2 偏置电压生成器以及加强硬件运算功能的 8×8 硬件乘法器。



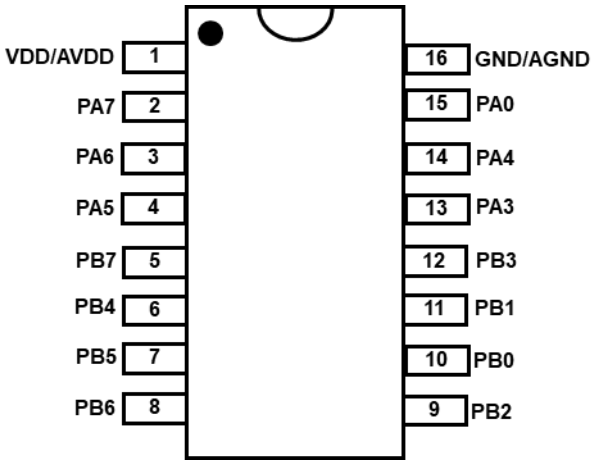
3. 引脚分配及功能说明



PFC232-S08 (SOP8-150mil)
PFC232-D08 (DIP8-300mil)



PFC232-S14 (SOP14-150mil)
PFC232-D14 (DIP14-300mil)



PFC232-S16 (SOP16-150mil)
PFC232-D16 (DIP16-300mil)

注意：**PFC232** 的引脚排列，与 **PMC232/PMS232** 互不相容。

引脚名称	输入/输出				特殊功能								
	I / O	上拉	下拉	唤醒	晶振	比较器	PWM	VDD/2	ADC	OPA	外部中断	外部复位	烧录
PA0	√	√	√	√		CO CIN- CIN+	PG0PWM	COM2	AD10	OPO	INT0		
PA3	√	√	√	√		CIN-	TM2PWM PG2PWM	COM4	AD8	OPIN-			√
PA4	√	√	√	√		CIN+ CIN-	PG1PWM	COM3	AD9	OPIN+ OPIN-	INT1A		
PA5	√	√	√	√			PG2PWM					√	√
PA6	√	√	√	√	√								√
PA7	√	√	√	√	√								
PB0	√	√	√	√				COM1	AD0		INT1		
PB1	√	√	√	√					AD1 Vref				
PB2	√	√	√	√			TM2PWM PG2PWM		AD2				
PB3	√	√	√	√			PG2PWM	COM5	AD3				
PB4	√	√	√	√			TM2PWM PG0PWM		AD4				
PB5	√	√	√	√			TM3PWM PG0PWM		AD5		INT0A		
PB6	√	√	√	√		CIN-	TM3PWM PG1PWM		AD6	OPIN-			
PB7	√	√	√	√		CIN-	TM3PWM PG1PWM		AD7	OPIN-			
VDD AVDD													√
GND AGND													√

注意

1. 所有 I/O 引脚都具有：施密特触发器输入；CMOS 电压基准位。
2. 当某引脚作为 PWM 输出端口时，其 IO 功能自动停用。
3. 当 PA5 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。
4. 仿真器不支持 PG2PWM 输出到 PA5。
5. VDD 是 IC 电源，AVDD 为模拟正电源。在 IC 内部，AVDD 与 VDD 连在一起(double bonding)，而外部为相同引脚。
6. GND 是 IC 接地引脚，而 AGND 是模拟接地引脚。在 IC 内部，AGND 与 GND 连在一起(double bonding)，而外部为相同引脚。

4. 中央处理器 (CPU)

4.1. 功能描述

PFC232 内有两个处理单元：FPPA0 和 FPPA1，在每一个处理单元中包括：

a. 其本身的程序计数器来控制程序执行的顺序
b. 自己的堆栈指针用来存储或恢复程序计数器的程序执行
c. 自己的累加器
d. 状态标志以记录程序执行的状态。

每一个 FPPA 都有自己的程序计数器和累加器用以执行程序，标志寄存器以记录程序状态，堆栈指针做为跳跃操作。基于这样的架构，FPPA0 和 FPPA1 可以独立执行自己程序，达到并行处理效能。

4.1.1. 两个处理单元工作模式

FPPA0 和 FPPA1 共享 2K words MTP 程序存储器，128 bytes 数据 SRAM 以及所有的 IO 口，这两个 FPPA 单元是各自独立运作在相斥的时钟周期，以避免干扰。芯片内部有一个工作切换硬件模块以决定 FPPA0 和 FPPA1 相对应的周期。图 1 所示为 FPPA0 和 FPPA1 硬件框图以及基本时序图。对于 FPPA0 而言，其程序将按顺序每两个系统时钟执行一次，如图：FPPA0 在第（M-1），第 M 和第（M+1）时钟周期执行程序。对于 FPPA1 而言，其程序将按顺序每两个系统时钟执行一次，如图：FPPA1 在第（N-1），第 N 和第（N+1）时钟周期执行程序。

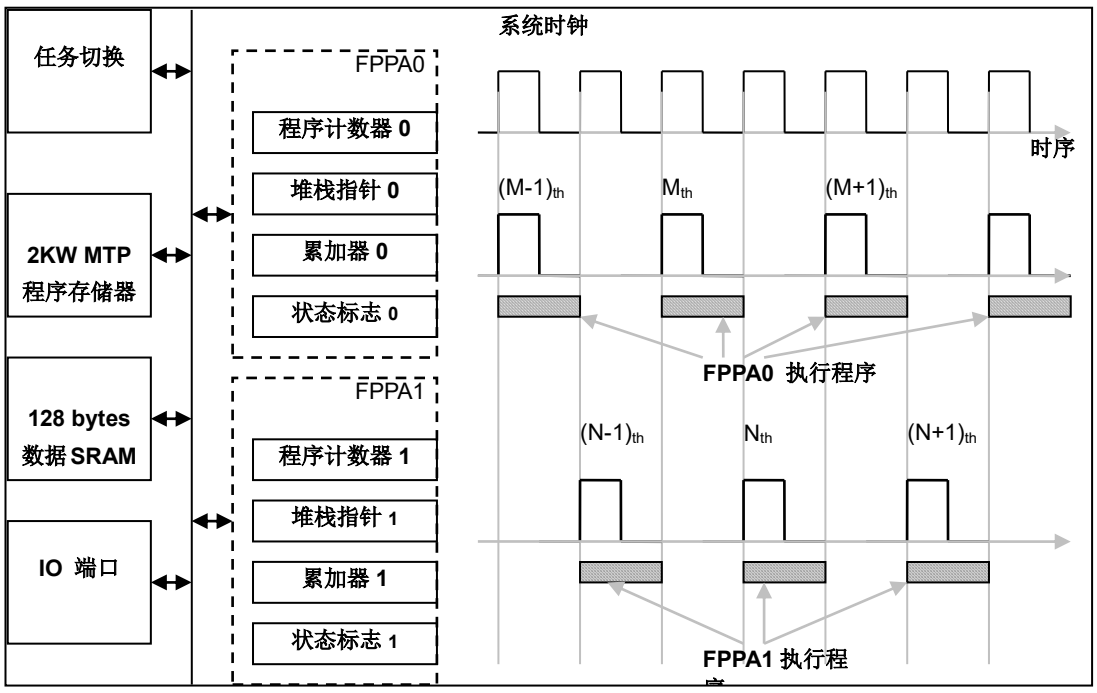


图 1：FPPA 单元架构以及基本时序

每个 FPPA 单元具有整个系统一半的计算能力，例如，如果系统时钟为 8MHz，FPPA0 和 FPPA1 将分别在 4MHz 时钟下工作。FPPA 单元可以通过允许寄存器编程来启用或停用；上电复位后，只有 FPPA0 是被启用的。系统初始化将从 FPPA0 开始，FPPA1 可以由用户的程序来决定是否启用。FPPA0 和 FPPA1 可以被 FPPA0 或 FPPA1 中任一个停用，包括停用本身这一 FPPA 单元。

4.1.2. 单一处理单元工作模式

传统的单片机使用者如果不需要有并行处理能力的单片机，PFC232 还提供单一处理单元工作模式，它的表现与传统单片机一致。当一个处理单元工作模式被选中后，FPPA1 始终停用，只有 FPPA0 是使能的。图 2 显示了每个 FPPA 单元的时序图，FPPA1 总是停用，只 FPPA0 活跃。

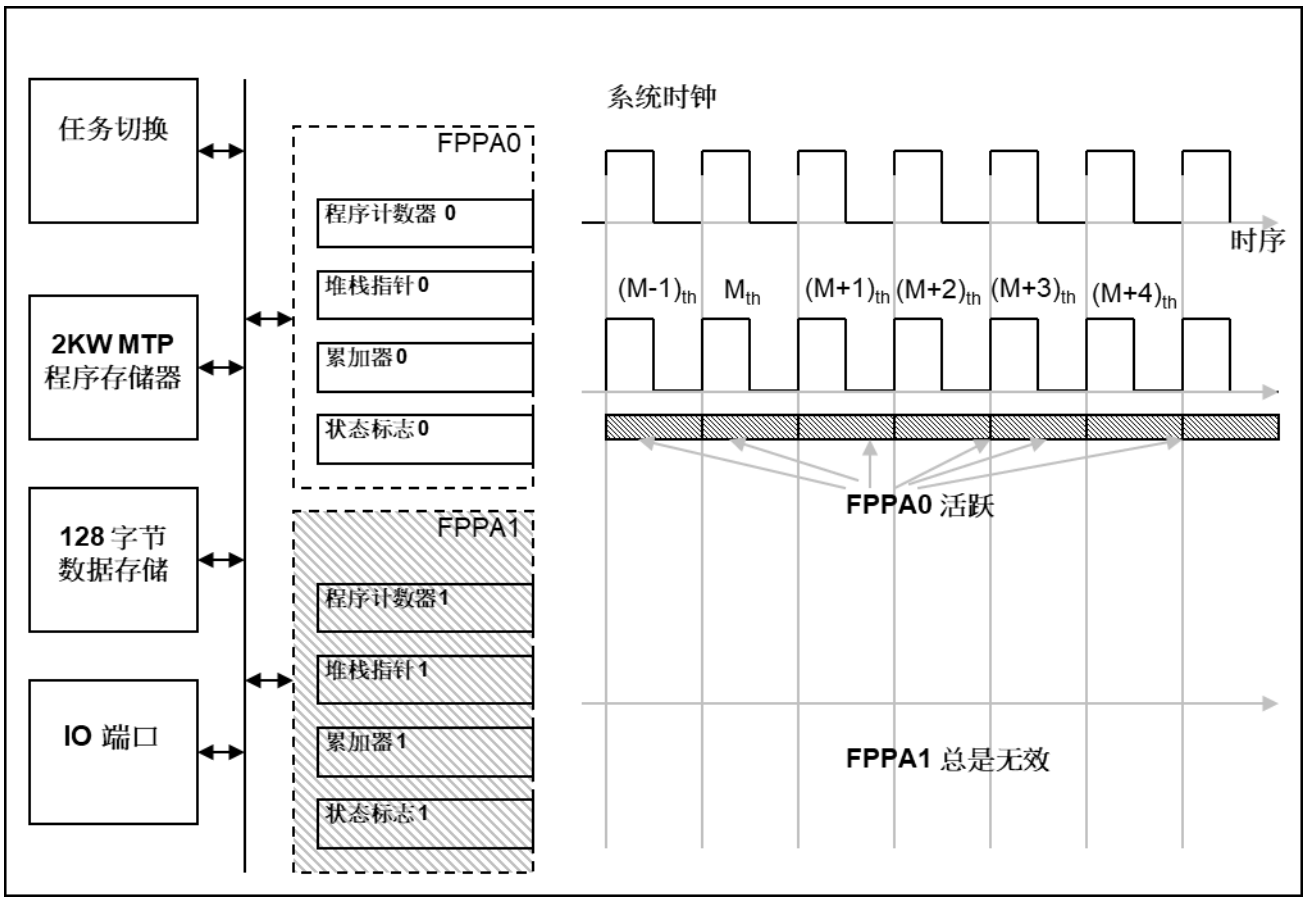


图 2：一个处理单元工作模式下的时序

4.1.3. 程序计数器

程序计数器（PC）记录下一个执行指令的地址，在每个指令周期后程序计数器会自动递增，以便指令码按顺序从程序存储器取出。某些指令，如分支指令和子程序调用都会改变顺序并放入一个新值到程序计数器。PFC232 程序计数器的位长度是 11 。在硬件复位后，FPPA0 的程序计数器为 0、FPPA1 为 1。当中断发生时，程序计数器会跳转到 0x10 的中断服务程序处。FPPA0 和 FPPA1 都具有各自独立的程序计数器来控制其程序执行顺序。

4.1.4. 程序结构

两个处理单元工作模式下程序结构

开机后，FPPA0 和 FPPA1 的程序开始地址分别是 0x000 和 0x001。中断服务程序的入口地址是 0x010，而且只有 FPPA0 才能接受中断服务。 PFC232 的基本软件结构如图 3 所示。两个 FPPA 的处理单元的程序代码是被放置在同一个程序空间。除了初始地址和中断入口地址外，处理单元的程序代码可以放在程序存储器任何位置，并没有在特定的地址。开机后，将首先执行 FPPA0Boot，其中将包括系统初始化和启用其它 FPPA 的单元。

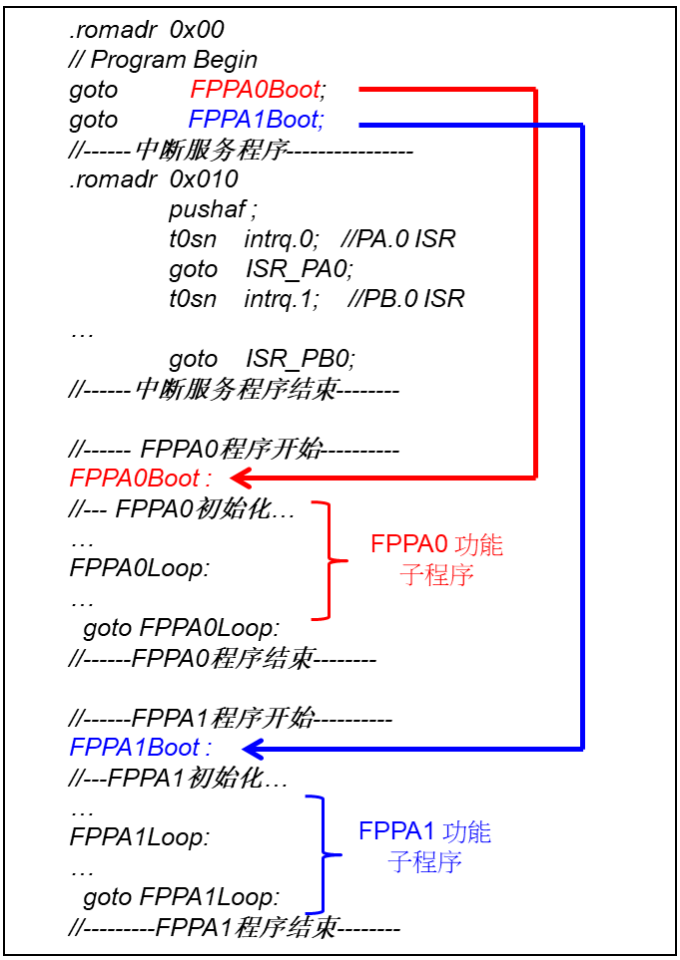


图 3：两个处理单元工作模式之程序结构

一个处理单元工作模式下程序结构

开机后，FPPA0 的程序开始地址是 0x000，中断服务程序的入口地址是 0x010，一个处理单元工作模式下的程序结构与传统的单片机软件结构相同，开机后，程序将从地址 0x000 开始依序执行。

4.1.5. 算术和逻辑单元

算术和逻辑单元（ALU）是用来作整数算术、逻辑、移位和其它特殊运算的单元。运算的数据来源可以从指令、累加器或 SRAM 数据存储器，计算结果可写入累加器或 SRAM。FPPA0 和 FPPA1 在其相应的操作周期分享 ALU 的使用。

4.2. 存储器

4.2.1. 程序存储器 (ROM)

PFC232 的程序存储器记忆体是 MTP（可多次编程），用来存放数据（包含：数据、表格和中断入口）和要执行的程序指令。PFC232 的程序存储器容量为 2K words，如表 1 所示。

复位之后，FPPA0 的初始地址是 0x000，FPPA1 的初始地址是 0x001，中断入口在 0x010，只有 FPPA0 能使用中断功能。

MTP 存储器从地址“0x7E0 to 0x7FF”供系统使用，从“0x001 ~ 0x00F”和“0x011~0x7DF”地址空间是用户的程序空间。地址 0x001 为两个 FPPA 单元模式的 FPPA1 初始地址，为单个 FPPA 单元模式的用户程序地址。

MTP 程序存储器最后 32 个地址空间是被保留给系统使用，如：校验码，序列号等。

地址	功能
0x000	FPPA0 起始地址 – *goto* 指令
0x001	FPPA1 起始地址 – *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
•	•
0x7D7	用户程序区
0x7E0	系统使用
•	•
0x7FF	系统使用

表 1：PFC232 程序存储器结构

两个处理单元工作模式下程序存储器分配例子

表 2 显示了一个例子，使用两个处理单元工作模式下，程序存储器分配情形：

地址	功能
0x000	FPPA0 起始地址 – *goto* 指令(*goto* 0x020)
0x001	FPPA1 程序开始
•	•
0x00F	*goto* 0x1A1 继续 FPPA1 程序
0x010	中断入口地址(只给 FPPA0)
•	•
0x01F	中断程序结束(取决于用户程序大小)
0x020	FPPA0 程序开始
•	•
0x1A0	FPPA0 程序结束
0x1A1	继续 FPPA1 程序
•	•
0x7DF	FPPA1 程序结束
0x7E0	系统使用
•	•
0x7FF	系统使用

表 2：两个处理单元工作模式之程序存储器分配案例

一个处理单元工作模式下程序存储器分配例子

当使用一个 FPPA 工作模式时，整个用户程序存储区都可以被分配到 FPPA0，表 3 显示程序存储器分配情形。

地址	功能
0x000	FPPA0 起始地址
0x001	FPPA0 程序开始
0x002	用户程序区
•	•
0x00F	*goto* 指令(*goto* 0x020)
0x010	中断入口地址
0x011	中断程序
•	•
0x01F	中断程序结束(取决于用户程序大小)
0x020	用户程序区
•	•
•	•
0x7DF	用户程序区
0x7E0	系统使用
•	•
0x7FF	系统使用

表 3：一个处理单元工作模式之程序存储器分配案例

4.2.2. 数据存储器 (SRAM)

图 4 显示了 PFC232 数据存储器的结构以及使用，所有的 SRAM 数据存储器可以透过 FPPA0 和 FPPA1 在 1 个时钟周期内直接读取或写入。存取方式可以是字节或位操作。此外 SRAM 数据存储器还充当间接访问方法的数据指针和 FPPA0、FPPA1 的堆栈记忆体。

FPPA0 和 FPPA1 的堆栈记忆体使用是独立互不影响的，并定义在数据存储器中。FPPA0 和 FPPA1 的堆栈指针通过堆栈指针寄存器各自定义，FPPA0 和 FPPA1 所需要的堆栈深度是由使用者来定。堆栈记忆体的调整可完全灵活安排，可以由用户动态调整。

对于间接存取指令而言，数据存储器用作数据指针来当数据地址，所有的数据存储器都可以当做数据指针，这对于间接存取指令是相当灵活和有用的。由于数据宽度为 8 位，PFC232 的 128 个字节数据存储器都可以利用间接存取指令来存取。

位寻址只能定义在 RAM 区的 0x00 到 0x3F 空间。

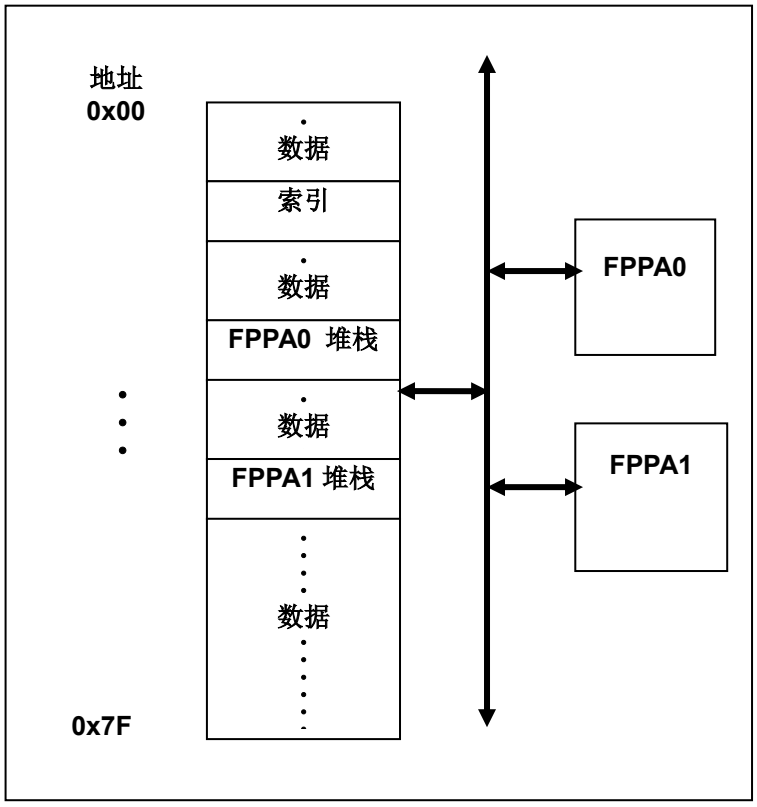


图 4：数据存储器结构和使用

4.2.3. 系统寄存器

PFC232 的寄存器地址空间与数据存储空间、MTP 程序空间三者互相独立。

以下是 PFC232 的各寄存器存放地址及简要描述：

	+0	+1	+2	+3	+4	+5	+6	+7
0x00	*FLAG*	*FPPEN*	*SP*	*CLKMD*	*INTEN*	*INTRQ*	*T16M*	*OPAOFS*
0x08	*MULOP*	*MULRH / TM2B*	*EOSCR*	-	*INTEGS*	*PADIER*	*PBDIER*	-
0x10	*PA*	*PAC*	*PAPH*	*PAPL*	*PB*	*PBC*	*PBPH*	*PBPL*
0x18	*GPCC*	*GPCS*	*OPAC*	-	*TM2C*	*TM2CT*	*TM2S*	*MISC*
0x20	*PWMG0C*	*PWMG0S*	*PWMG0-DTH*	*PWMG0-DTL*	*PWMG0-CUBH*	*PWMG0-CUBL*	*PWMG1C*	*PWMG1S*
0x28	*PWMG1DTH*	*PWMG1DTL*	*PWMG1-CUBH*	*PWMG1-CUBL*	*PWMG2C*	*PWMG2S*	*PWMG2-DTH*	*PWMG2-DTL*
0x30	*PWMG2CUBH*	*PWMG2CUBL*	*TM3C*	*TM3CT*	*TM3S*	*ADCC*	*ADCM*	*ADCRH*
0x38	*TM3B / ADCRL*	*ADCRGC*	-	-	-	-	-	-

FLAG：标志寄存器

FPPEN：FPP 单元允许寄存器

SP：堆栈指针寄存器

CLKMD：时钟控制寄存器

INTEN：中断允许寄存器

INTRQ：中断请求寄存器

T16M：Timer16 控制寄存器

OPAOFS：OPA 失调寄存器

MULOP：乘法器运算对象寄存器

MOLRH：乘法器结果高字节寄存器

TM2B / *TM3B*：Timer2 / Timer3 上限寄存器

EOSCR：外部晶体振荡器控制寄存器

INTEGS：中断缘选择寄存器

PADIER：端口 A 数字输入启用寄存器

PBDIER：端口 B 数字输入启用寄存器

PA：端口 A 数据寄存器

PAC：端口 A 控制寄存器

PAPH：端口 A 上拉控制寄存器

PAPL：端口 A 下拉控制寄存器

PB：端口 B 数据寄存器

PBC：端口 B 控制寄存器

PBPH：端口 B 上拉控制寄存器

PBPL：端口 B 下拉控制寄存器

GPCC：比较器控制寄存器

GPCS：比较器选择寄存器

OPAC：OPA 控制寄存器

TM2C / *TM3C*：Timer2 / Timer3 控制寄存器

TM2CT / *TM3CT*：Timer2 / Timer3 计数寄存器

TM2S / *TM3S*：Timer2 / Timer3 分频寄存器

MISC：杂项寄存器

PWMG0C / *PWMG1C* / *PWMG2C*：
PWMG0 / PWMG1 / PWMG2 控制寄存器

PWMG0S / *PWMG1S* / *PWMG2S*：
PWMG0 / PWMG1 / PWMG2 分频寄存器

PWMG0DTH / *PWMG1DTH* / *PWMG2DTH*：
PWMG0 / PWMG1 / PWMG2 计数上限高位寄存器

PWMG0DTL / *PWMG1DTL* / *PWMG2DTL*：
PWMG0 / PWMG1 / PWMG2 计数上限低位寄存器

PWMG0CUBH / *PWMG1CUBH* / *PWMG2CUBH*：
PWMG0 / PWMG1 / PWMG2 占空比高位寄存器

PWMG0CUBL / *PWMG1CUBL* / *PWMG2CUBL*：
PWMG0 / PWMG1 / PWMG2 占空比低位寄存器

ADCC：ADC 控制寄存器

ADCM：ADC 模式控制寄存器

ADCRH / ADCRL：ADC 数据 高/低 位寄存器

ADCRGC：ADC 调节控制寄存器

4.2.3.1. 标志寄存器(*FLAG*)，地址 = 0x00

位	初始值	读/写	描 述
7 – 4	-	-	保留。这 4 个位读值为“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。

4.2.3.2. FPPA 单元允许寄存器(*FPPEN*)，地址 = 0x01

位	初始值	读/写	描 述
7 – 2	-	-	保留。
1	0	读/写	FPPA1 启用。此位是用来启用 FPPA1。 0/1：停用/启用
0	1	读/写	FPPA0 启用。此位是用来启用 FPPA0。 0/1：停用/启用

4.2.3.3. 杂项寄存器(*MISC*)，地址 = 0x1F

位	初始值	读/写	描 述
7 – 6	-	-	保留。请保持为 0。
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 VDD/2 偏置电压产生器。 0 / 1：停用 / 启用（仿真器不可以动态切换）
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

4.3. 堆栈

在每个处理单元的堆栈指针是用来指引堆栈存储器的顶部，该处是用来存储子程序的局部变量和参数的地方；堆栈指针寄存器（*SP*）的地址是 0x02。堆栈指针的位数是 8 位，堆栈存储器是与数据 SRAM 共享，所以堆栈存储器的使用从地址 0x00 开始，并在 128 字节以内，不可以超过 128 字节。FPPA0 和 FPPA1 使用的堆栈存储器都可以由用户通过指定堆栈指针寄存器来调整，意味着 FPPA0 和 FPPA1 的堆栈指针单位深度是可调的，以优化系统性能。下面的示例显示了如何在 ASM 汇编语言下定义堆栈：

```
    .ROMADR  0
    GOTO   FPPA0
    GOTO   FPPA1
    ...
    .RAMADR  0          // 地址必需小于 0x100
    WORD   Stack0 [1]       // 1 个 WORD
    WORD   Stack1 [2]       // 2 个 WORD
    ...
FPPA0:
    SP   =   Stack0;        // 指定 Stack0 给 FPPA0 使用,
                            // 只能有一层呼叫，因为 Stack0[1]
    ...
    call function1
    ...
FPPA1:
    SP   =   Stack1;        // 指定 Stack1 给 FPPA1 使用,
                            // 可以有 2 层呼叫，因为 Stack1[2]
    ...
    call function2
    ...
```

在使用 Mini-C 汇编语言下，由系统软件计算堆栈的深度，使用者不需特别花时间计算，主程序如下：

```
void        FPPA0 (void)
{
        ...
}
```

用户可以在程序分解的窗口里检查堆栈的设定，图 5 表示在 FPPA0 执行前的堆栈状态，系统计算出所需的堆栈空间，并保留该空间给程序使用。

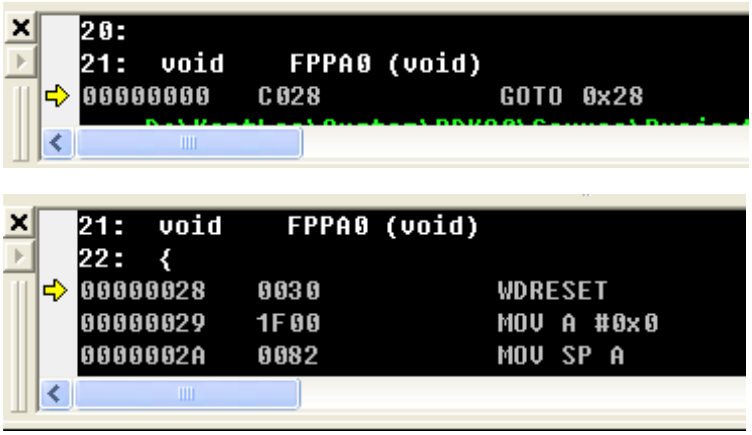


图 5：使用 Mini-C 的堆栈设定

4.3.1. 堆栈指针寄存器(*SP*)，地址 = 0x02

位	初始值	读/写	描 述
7 – 0	-	读/写	堆栈指针寄存器。读出当前堆栈指针，或写入以改变堆栈指针。请注意 0 位必须维持为 0。因程序计数器是 16 位。

4.4. 程序选项 Code Options

选项	选择	说明
Security	Enable	MTP 内容加密 7 / 8 words
	Disable（默认）	MTP 内容不加密，程序可以被读取
LVR	4.0V	选择 LVR = 4.0V
	3.75V	选择 LVR = 3.75V
	3.0V	选择 LVR = 3.0V
	2.7V	选择 LVR = 2.7V
	2.5V（默认）	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_P_In	**PA.4（默认）**	比较器正输入来自于 PA.4
	PA.0	比较器正输入来自于 PA.0
GPC_PWM	**Disable（默认）**	GPC / PWM 互相独立
	Enable	比较器输出结果控制 PWM（仿真器不支持）

选项	选择	说明
OPA_PWM	**Disable（默认）**	OPA / PWM 互相独立
	Enable	OPA 输出结果控制 PWM（仿真器不支持）
Comparator_Edge	**All_Edge（默认）**	在上升和下降沿比较器都触发中断
	Rising_Edge	在上升沿比较器触发中断
	Falling_Edge	在下降沿比较器触发中断
Interrupt Src0	**PA.0（默认）**	*INTEN/ INTRQ*.位 0 来自于 PA.0
	PB.5	*INTEN/ INTRQ*.位 0 来自于 PB.5
Interrupt Src1	**PB.0（默认）**	*INTEN/ INTRQ*.位 1 来自于 PB.0
	PA.4	*INTEN/ INTRQ*.位 1 来自于 PA.4
PB4_PB7_Drive	Normal	PB4 & PB7 驱动电流/灌电流 正常
	Strong（默认）	PB4 & PB7 驱动电流/灌电流 强
PWM_Source	**16MHz（默认）**	当 *PWMG0C*.0 = 1,PWMG0 时钟源 = 16 Mhz 当 *PWMG1C*.0 = 1,PWMG1 时钟源 = 16 Mhz 当 *PWMG2C*.0 = 1,PWMG2 时钟源 = 16 Mhz
	32MHz	当 *PWMG0C*.0 = 1,PWMG0 时钟源 = 32 Mhz 当 *PWMG1C*.0 = 1,PWMG1 时钟源 = 32 Mhz 当 *PWMG2C*.0 = 1,PWMG2 时钟源 = 32 Mhz （仿真器不支持）
TMx_Source	**16MHz（默认）**	当 *TM2C*[7:4] = 0010,TM2 时钟源 = 16 MHz 当 *TM3C*[7:4] = 0010,TM3 时钟源 = 16 MHz
	32MHz	当 *TM2C*[7:4] = 0010,TM2 时钟源 = 32 MHz 当 *TM3C*[7:4] = 0010,TM3 时钟源 = 32 MHz （仿真器不支持）
TMx_Bit	**6 Bit（默认）**	当 *TM2S*.7 = 1,TM2 PWM 分辨率为 6 位 当 *TM3S*.7 = 1,TM3 PWM 分辨率为 6 位
	7 Bit	当 *TM2S*.7 = 1,TM2 PWM 分辨率为 7 位 当 *TM3S*.7 = 1,TM3 PWM 分辨率为 7 位 （仿真器不支持）
EMI	Disable	停用 EMI 优化选项
	Enable（默认）	系统时钟会轻微调整以获得更好的 EMI 性能
FPPA	**1-FPPA（默认）**	单一 FPPA 处理单元模式
	2-FPPA	双 FPPA 处理单元模式

5. 振荡器和系统时钟

PFC232 提供 3 个振荡器电路：外部晶体振荡器(EOSC)、内部高频 RC 振荡器(IHRC)、内部低频 RC 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 4：PFC232 提供 3 个振荡器电路

5.1. 内部高频振荡器和内部低频振荡

IHRC、ILRC 的频率会因工厂生产、电源电压和温度的变化而变化，请参阅 IHRC、ILRC 频率和 VDD、温度的测量图表。

PFC232 烧录工具提供 IHRC 频率校准（通常校准到 16MHz）功能，以此来消除工厂生产引起的频率漂移。ILRC 没有校准操作，对于需要精准定时的应用请不要使用 ILRC 的时钟当作参考时间。

5.2. 外部晶体振荡器

外部晶体振荡器的工作频率范围可以从 32KHz 至 4MHz，PFC232 不支持频率在 4MHz 以上的振荡器。图 6 显示了使用外部晶体振荡器的的硬件连接。

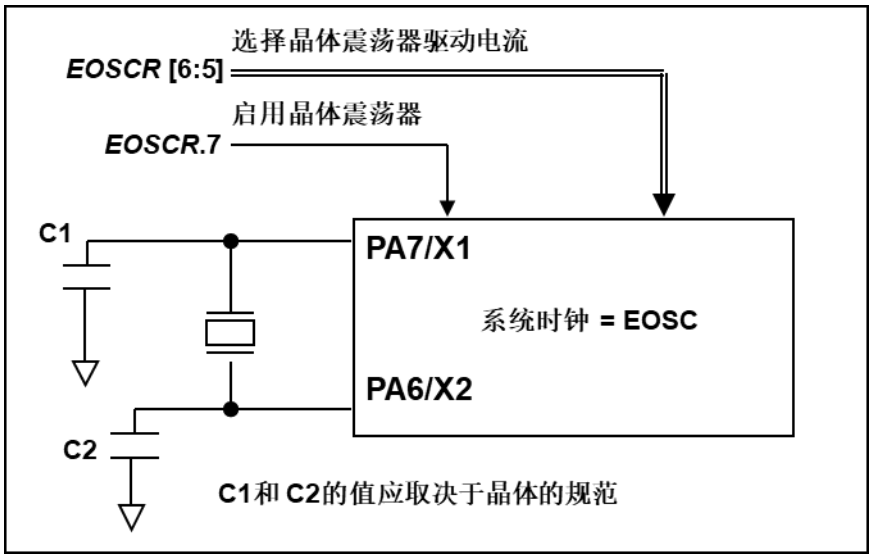


图 6：外部晶体振荡器的的硬件连接

5.2.1. 外部晶体振荡器控制寄存器(*EOSCR*)，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/使能
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 – 0	-	-	保留。请设为 0。

5.2.2. 外部晶体振荡器的使用及注意事项

除了晶振的选择外，外部电容器和寄存器 *EOSCR* 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。

表 5 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 5：晶体振荡器 C1 和 C2 推荐值

使用晶振时 PA7 和 PA6 的配置：

(1) PA7 和 PA6 设定为输入；

(2) PA7 和 PA6 内部上拉电阻设为关闭；

(3) 用 *PADIER* 寄存器将 PA6 和 PA7 设为模拟输入，防止漏电。

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。

使用晶体振荡器时，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 = 0b111_00000;
        $   T16M    EOSC, /1, BIT13;   // T16M.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;      // 从 0x0000 算到 0x2000，然后设置 INTRQ.T16
        clkmd=0xB4;                      // 切换系统时钟到 EOSC;
    clkmd.4 = 0;                         // 关闭 IHRC
        ...
}
```

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

5.3. 系统时钟与 IHRC 频率校准

5.3.1. 系统时钟

系统时钟的时钟源有 EOSC，IHRC 和 ILRC，PFC232 的时钟系统的硬件框图如图 7 所示。

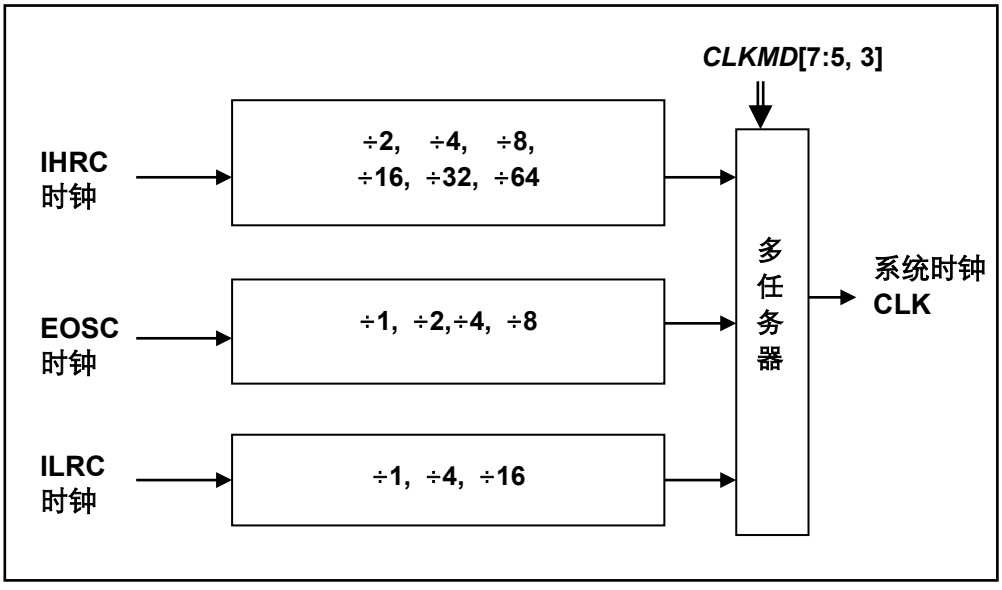


图 7：系统时钟源选择

5.3.1.1. 时钟控制寄存器(CLKMD)，地址 = 0x03

位	初始值	读/写	描 述	
7 – 5	111	读/写	系统时钟选择	
			类型 0，*CLKMD*[3]=0	类型 1，*CLKMD*[3]=1
			000：IHRC÷4 001：IHRC/2 010：保留 011：EOSC/4 100：EOSC/2 101：EOSC 110：ILRC/4 111：ILRC（默认）	000：IHRC/16 001：IHRC/8 010：ILRC/16（仿真器不支持） 011：IHRC/32 100：IHRC/64 101：EOSC/8 其他：保留
4	1	读/写	内部高频 RC 振荡器功能。 0/1：停用/启用	
3	0	读/写	时钟类型选择。这个位是用来选择位 7~位 5 的时钟类型。 0 / 1：类型 0 /类型 1	
2	1	读/写	内部低频 RC 振荡器功能。0/1：停用/启用 当内部低频 RC 振荡器功能停用时，看门狗功能同时被关闭。	
1	1	读/写	看门狗功能。 0/1：停用/启用	
0	0	读/写	引脚 PA5/PRSTB 功能。0 / 1：PA5 / PRSTB	

5.3.2. 频率校准

IHRC 校准的功能是在用户编译程序时序做选择，校准命令以及校准选项将自动插入到用户程序中。校准命令如下所示：

.ADJUST_IC SYSCLK=IHRC/(**p1**), IHRC=(**p2**)MHz, VDD=(**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 =2.2～5.5；根据不同的电源电压校准芯片。

通常情况下，ADJUST_IC 是开机后的第一个命令，用以设定系统的工作频率。IHRC 频率校准的程序只在将程序代码写入 MTP 存储器的时候执行一次，此后不会再被执行。

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。IHRC 频率校准以及系统时钟的选项，如表 6 所示：

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

表 6：IHRC 频率校准选项

下面显示在不同的选项下，PFC232 不同的状态：

(1) .ADJUST_IC SYSCLK=IHRC/4, IHRC=16MHz, V_{DD}=5V

开机后，*CLKMD* = 0x14：

a. IHRC 的校准频率为 16MHz@V_{DD}=5V，启用 IHRC 的硬件模块
b. 系统时钟 = IHRC/4 = 4MHz
c. 看门狗被停用，启用 ILRC，PA5 是在输入模式

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

开机后，*CLKMD* = 0x3C：

a. IHRC 的校准频率为 16MHz@VDD=2.5V，启用 IHRC 的硬件模块
b. 系统时钟 = IHRC/8 = 2MHz
c. 看门狗被停用，启用 ILRC，PA5 是在输入模式

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

开机后，*CLKMD* = 0xE4：

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

(4) .ADJUST_IC DISABLE

开机后，*CLKMD* 寄存器没有改变（没任何动作）：

a. IHRC 不校准并且 IHRC 模块停用
b. 系统时钟 = ILRC 或 IHRC/64
c. 看门狗被启用，启用 ILRC，PA5 是在输入模式

5.3.2.1. 特别声明

(1) IHRC 的校正操作是在 IC 烧录时进行的。

(2) IC 塑封材料（不论是封装用还是 COB 用的黑胶）的特性会对 IHRC 的频率有一定影响。如果用户在 IC 盖上塑封材料前进行烧录，然后再封上塑封材料，则可能造成 IHRC 的特性偏移超出规格的现象，正常情况下频率会变慢一些。

(3) 上述问题通常发生在用户使用 COB 封装或是委托我司进行晶圆代烧(QTP)时。此情况下我司将不对频率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。

5.3.3. 系统时钟切换

IHRC 校准后，透过 *CLKMD* 寄存器的设定，PFC232 系统时钟可以随意在 IHRC，ILRC 和 EOSC 之间切换。**但必须注意，不可在切换系统时钟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，应该先把系统时钟源切换到 B，然后再关闭 A 时钟源。请参阅 IDE：“使用手册”→ “IC 介绍” → “绶存器介绍” → “*CLKMD*”。

例 1: 系统时钟从 ILRC 切换到 IHRC/4

```
    ...                         //  系统时钟为 ILRC
    CLKMD.4     =   1;          //  先打开 IHRC，可以提高抗干扰能力
CLKMD   =   0x14;   //  切换为 IHRC/4，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/4 切换到 EOSC

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

例 3: 系统时钟从 IHRC/8 切换到 IHRC/4

```
...                         //  系统时钟为 IHRC/8，ILRC 为启用
CLKMD   =   0X14;   //  切换为 IHRC/4
...
```

例 4: 系统可能当机，如果同时切换时钟和关闭原来的振荡器

```
...                         //  系统时钟为 ILRC
CLKMD   =   0x10;   //  不能从 ILRC 切换到 IHRC/4，同时又关闭 ILRC 振荡器
...
```

6. 复位

引起 PFC232 复位的原因有四种：上电复位、LVR 复位、看门狗超时溢出复位和 PRSTB 引脚复位。发生复位后，系统会重新启动，程序计数器会跳跃到地址 0x000，PFC232 的所有寄存器将被设置为默认值。

6.1. 上电复位(POR)

开机时，POR(Power On Reset)是用于复位 PFC232，开机时间大约 3000 ILRC，其时序图如图 8 所示。用户必须确保上电后电源电压稳定。

发生上电复位时，PFC232 数据存储器的值处于不确定的状态。

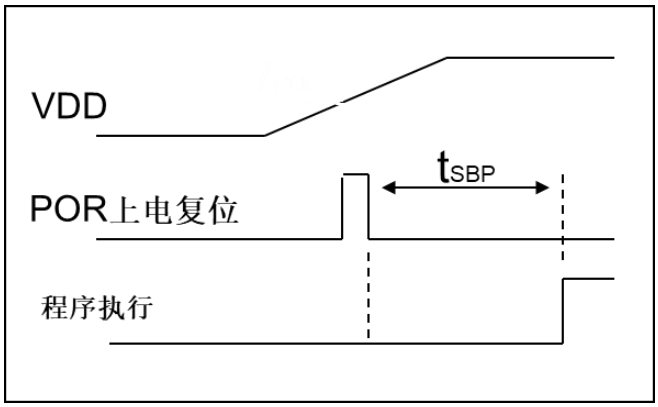


图 8：上电复位时序图

图 9 显示的是典型开机流程。请注意，上电复位后 FPPA1 是停用，建议不要在 FPPA0 以及系统初始化完成前，启用 FPPA1。

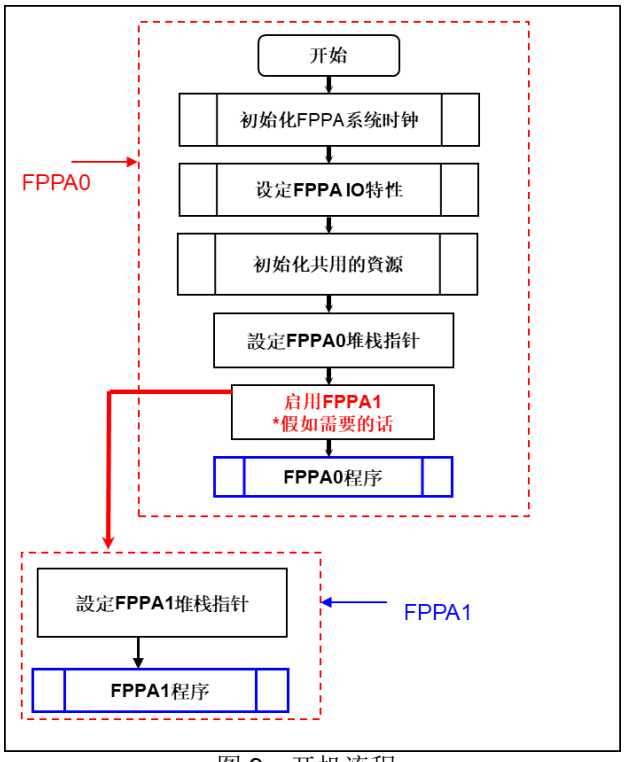


图 9：开机流程

6.2. 低电压复位(LVR)

若 VDD 下降到低于 LVR(Low Voltage Reset)电压水平，系统会发生 LVR 复位，其时序图如图 10。当 LVR 复位时，若 VDD 大于 V_{DR}（数据存储器数据保存电压），数据存储器的值将会被保留，但若在重新上电后 SRAM 被清除，则数据无法保留；若 VDD 小于 V_{DR}，数据存储器的值是在不确定的状态。

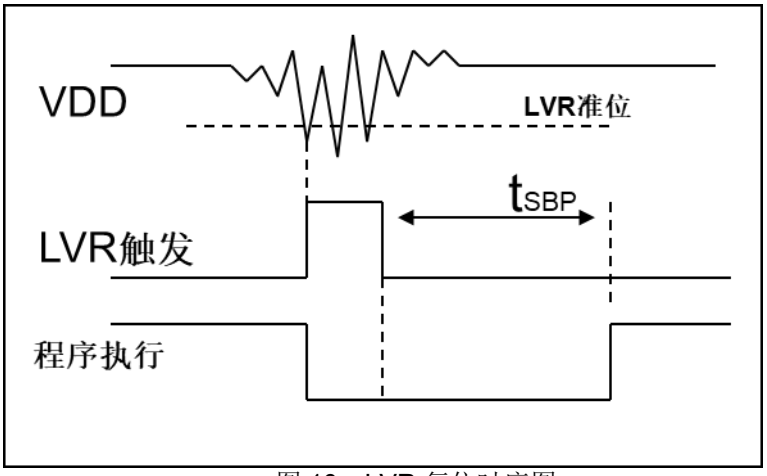


图 10：LVR 复位时序图

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
8MHz	≧ 3.75V	3.75V
4MHz	≧ 2.5V	2.5V
2MHz	≧ 2.2V	2.2V

表 7：LVR 设置参考，与系统频率、VDD 之间的关系

(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。

杂项寄存器(*MISC*)，地址 = 0x08			
位	初始值	读/写	描 述
7 – 5	-	-	保留。请保持为 0。
4	0	只写	使能 VDD/2 偏置电压产生器。0 / 1：停用 / 启用
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**

6.3. 看门狗超时溢出复位

看门狗（WDT）是一个计数器，其时钟源来自 ILRC，所以当 ILRC 关闭时，看门狗也会失效。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。

另外，在复位或唤醒事件后，看门狗的周期也会比预期的短。建议在这些事件之后通过 *wdreset* 指令清除 WDT，以确保在 WDT 超时之前有足够的时钟周期。

为确保看门狗在超时溢出之前被清零，在安全时间内，可以用指令 *wdreset* 清零看门狗。在上电复位(POR)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，PFC232 将复位并重新运行程序，其复位时序图如图 5 所示。发生 WDT 复位时，PFC232 数据存储器的值将被保留。

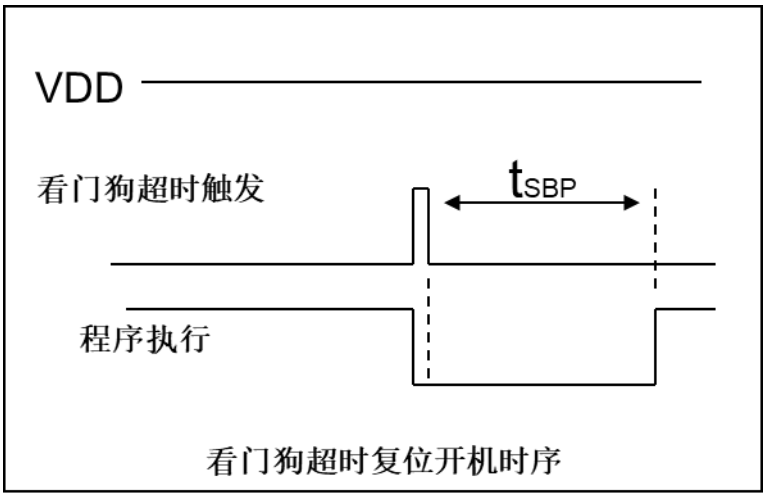


图 11：看门狗超时溢出的相关时序

利用寄存器 *MISC*[1:0]可选择四种不同的看门狗超时时间，利用 *CLKMD*.1 可以选择将看门狗功能停用。

时钟控制寄存器(*CLKMD*)，地址 = 0x03			
位	初始值	读/写	描 述
7 - 5	111	读/写	系统时钟选择
4	1	读/写	内部高频 RC 振荡器功能。 0/1：停用/启用
3	0	读/写	时钟类型选择
2	**1**	**读/写**	**内部低频 RC 振荡器功能。 0/1：停用/启用** **当 ILRC 关闭时，看门狗也会失效**
1	**1**	**读/写**	**看门狗功能。 0/1：停用/启用**
0	**0**	**读/写**	**引脚 PA5/PRSTB 功能。0 / 1：PA5 / PRSTB**

6.4. 外部复位(PRSTB)

PFC232 支持外部复位功能，其外部复位引脚与 PA5 共享同一个 IO 端口。使用外部复位功能需要：

(1) 设定 PA5 为输入；

(2) 设定 *CLKMD*.0=1，使 PA5 为外部 PRSTB 输入脚位。

在外部复位引脚为高电平时，系统处于正常工作状态；一旦复位引脚检测到低电平，系统即发生复位。PRSTB 复位时序图如图 12 所示。

当发生 PRSTB 复位时，PFC232 数据存储器的值将被保留。

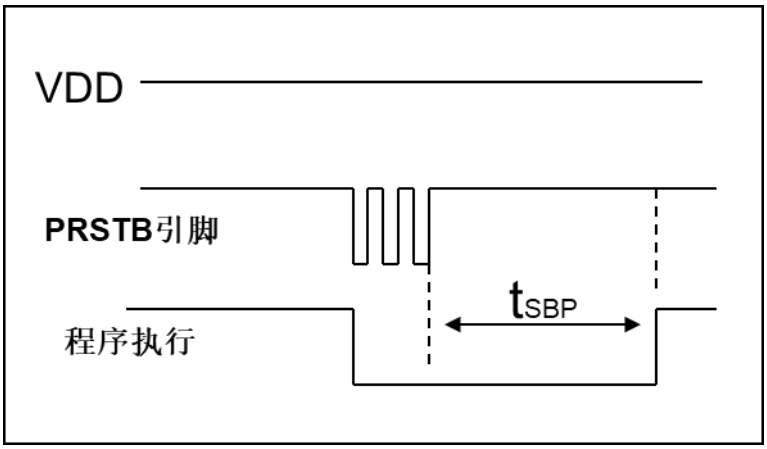


图 12：外部引脚复位的相关时序

7. 系统工作模式

PFC232 有三个由硬件定义的操作模式，分别为：

(1) 正常工作模式

(2) 电源省电模式

(3) 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。

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。

7.1. 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继续工作。所以只有 CPU 是停止执行指令。输入引脚切换引起的系统唤醒可视为单片机继续正常的运行。

省电模式的详细信息如下所示：

(1) IHRC 和振荡器模块：没有变化。如果它被启用，它仍然继续保持活跃。

(2) ILRC 振荡器模块：必须保持启用，唤醒时需要靠 ILRC 启动。

(3) 系统时钟停用，因此，CPU 停止执行。

(4) MTP 存储器被关闭。

(5) 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）

(6) 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 设置
...
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。
```

7.2. 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* 指令可以使芯片直接进入掉电模式。在下达 *stopsys* 指令之前建议将 *GPCC*.7 设为 0 来关闭比较器。

输入引脚的唤醒可视为程序正常运行，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。

下面显示发出 *stopsys* 命令后，PFC232 内部详细的状态：

(1) 所有的振荡器模块被关闭。

(2) MTP 存储器被关闭。

(3) SRAM 和寄存器内容保持不变。

(4) 唤醒源：IO 在数字输入模式下电平变换（*PxDIER* 位是 1）。

掉电模式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;       // 进入掉电模式
        if (...) break;// 假如发生唤醒而且检查 OK，就返回正常工作
                       // 否则，停留在掉电模式。
}
CLKMD = 0x14;          // 系统时钟从 ILRC 变为 IHRC/4
```

7.3. 唤醒

进入掉电或省电模式后，PFC232 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 8 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			
	切换 IO 引脚	计时器唤醒	比较器唤醒
stopsys	是	否	否
stopexe	是	是	是

表 8：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 PFC232，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，正常唤醒时间大约是 3000 ILRC 时钟周期。 通过 *MISC* 寄存器可以选择快速唤醒来减少唤醒时间，切换 IO 引脚的快速唤醒时间约为 45 个 ILRC 时钟。

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

8. 中断

PFC232 有 8 个中断源：

- 外部中断源 PA0/PB5
- 外部中断源 PB0/PA4
- ADC 中断
- 比较器中断
- Timer16 中断
- Timer2 中断
- Timer3 中断
- PWMG0 中断

每个中断请求源都有自己的中断控制位启用或停用它。中断硬件框图请参考图 13。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。

中断堆栈是共享数据存储器，其地址由堆栈寄存器 *SP* 指定。由于程序计数器是 16 位宽度，堆栈寄存器 *SP* 位 0 应保持 0。此外，用户可以使用 *pushaf* 指令存储 ACC 和标志寄存器的值到堆栈，以及使用 *popaf* 指令将值从堆栈恢复到 ACC 和标志寄存器中。由于堆栈与数据存储器共享，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。

在中断服务程序中，可以通过读取寄存器 *INTRQ* 知道中断发生源。

注：可在 Code Option Interrupt Src0 或 Interrupt Src1 中切换外部中断源。

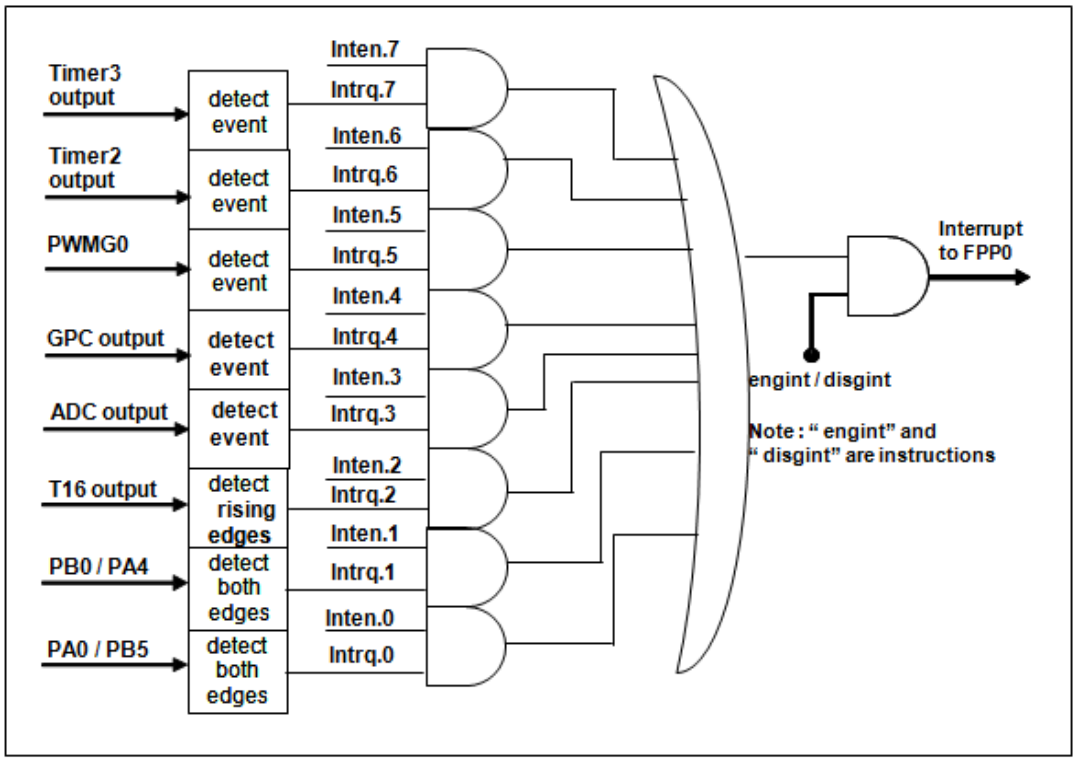


图 13：中断硬件框图

8.1. 中断允许寄存器(*INTEN*)，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	启用从比较器的中断。0/1：停用/启用
3	-	读/写	启用从 ADC 的中断。0/1：停用/启用
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/PA4 的溢出中断。0/1：停用/启用 （由 code option Interrupt Src1 决定）
0	-	读/写	启用从 PA0/PB5 的中断。 0/1：停用/启用 （由 code option Interrupt Src0 决定）

8.2. 中断请求寄存器(INTRQ)，地址 = 0x05

位	初始值	读/写	描 述
7	-	读/写	Timer3 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
6	-	读/写	Timer2 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
5	-	读/写	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	ADC 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
2	-	读/写	Timer16 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
1	-	读/写	PB0/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	PA0/PB5 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

8.3. 中断缘选择寄存器 (*INTEGS*), 地址 = 0x0C

位	初始值	读/写	描 述
7 – 5	-	只写	保留。
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/PA4 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/PB5 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

注意：

(1) *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。即使 *INTEN* 为 0，*INTRQ* 还是会被中断发生源触发。

(2) PA4 and PB5 可以被用作外部中断引脚。当使用 PA4 作为外部中断引脚时，寄存器 *INTEN* / *INTRQ* / *INTEGS* 的设置与 PB0 一致，唯一的区别是在 Code Options 中选择 PB0 或 PA4 作为 interrupt_Src1 中断源。同样，当使用 PB5 作为外部中断引脚时，*INTEN* / *INTRQ* / *INTEGS* 寄存器的设置方法与 PA0 相同，唯一的区别是在 Code Options 中选择 PA0 或 PB5 作为 interrupt_Src0 中断源。

8.4. 中断工作流程

一旦发生中断，其具体工作流程如下：

(1) 程序计数器将自动存储到 *SP* 寄存器指定的堆栈存储器。
(2) 新的 *SP* 将被更新为 SP+2。
(3) 全局中断将自动被停用。
(4) 将从地址 0x010 获取下一条指令。

中断完成后，发出 *reti* 指令返回既有的程序，其具体工作流程如下：

(1) 从 *SP* 寄存器指定的堆栈存储器自动恢复程序计数器。
(2) 新的 *SP* 将被更新为 SP-2。
(3) 全局中断将自动启用。
(4) 下一条指令将是中断前原来的指令。

8.5. 中断的一般步骤

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（启用全局中断）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。

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
```

* 在主程序中，可使用 *disgint* 指令关闭所有中断。

8.6. 使用中断举例

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

```
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
 }
```

9. I/O 端口

9.1. IO 相关寄存器

9.1.1. 端口 A 数字输入启用寄存器(*PADIER*)，地址 = 0x0D

位	初始值	读/写	描 述
7 – 6	11	只写	启用 PA7~PA6 数字输入和系统唤醒。 1 / 0：启用 / 停用 当使用外部晶体振荡器时，这两个位应该设置为低，以防止漏电。
5	1	只写	启用 PA5 数字输入和系统唤醒。 1 / 0：启用 / 停用
4	1	只写	启用 PA4 数字输入、系统唤醒和中断请求。 1 / 0：启用 / 停用
3	1	只写	启用 PA3 数字输入和系统唤醒。 1 / 0：启用 / 停用
2 – 1	-	只写	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

9.1.2. 端口 B 数字输入启用寄存器(*PBDIER*)，地址 = 0x0E

位	初始值	读/写	描 述
7 – 6	11	只写	启用 PB7~PB6 数字输入和系统唤醒。 1 / 0：启用 / 停用
5	1	只写	启用 PB5 数字输入、系统唤醒和中断请求。 1 / 0：启用 / 停用
4 - 1	1111	只写	启用 PB4~PB1 数字输入和系统唤醒。 1 / 0：启用 / 停用
0	1	只写	启用 PB0 数字输入、系统唤醒和中断请求。 1 / 0：启用 / 停用

9.1.3. 端口 A 数据寄存器(*PA*)，地址 = 0x10

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

9.1.4. 端口 A 控制寄存器(*PAC*)，地址 = 0x11

位	初始值	读/写	描 述
7 – 0	0x00	读/写	端口 A 控制寄存器。这些寄存器是用来定义端口 A 每个相应的引脚的输入模式或输出模式。0 / 1：输入/输出

9.1.5. 端口 A 上拉控制寄存器(*PAPH*)，地址 = 0x12

位	初始值	读/写	描 述
7 – 3	00000	读/写	端口 PA7~PA3 上拉控制寄存器。用来控制端口上拉电阻的使能。 0/1：停用/启用
2 – 1	-	-	保留。请保持为 0。
0	0	读/写	端口 PA0 上拉控制寄存器。用来控制端口上拉电阻的使能。 0/1：停用/启用

9.1.6. 端口 A 下拉控制寄存器(*PAPL*)，地址 = 0x13

位	初始值	读/写	描 述
7 – 0	0x00	读/写	端口 A 下拉控制寄存器。用来控制端口下拉电阻的使能，且仅在输入状态下有效。 0/1：停用/启用

9.1.7. 端口 B 数据寄存器(*PB*)，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。

9.1.8. 端口 B 控制寄存器(*PBC*)，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

9.1.9. 端口 B 上拉控制寄存器(*PBPH*)，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端口 B 每个相应引脚上拉电阻的使能。 0/1：停用/启用

9.1.10. 端口 B 下拉控制寄存器(*PBPL*)，地址 = 0x17

位	初始值	读/写	描 述
7 – 0	0x00	读/写	端口 B 下拉控制寄存器。用来控制端口下拉电阻的使能，且仅在输入状态下有效。 0/1：停用/启用

9.2. IO 结构及功能

9.2.1. IO 引脚的结构

PFC232 的所有 IO 引脚都具有相同的结构，如下图 14。

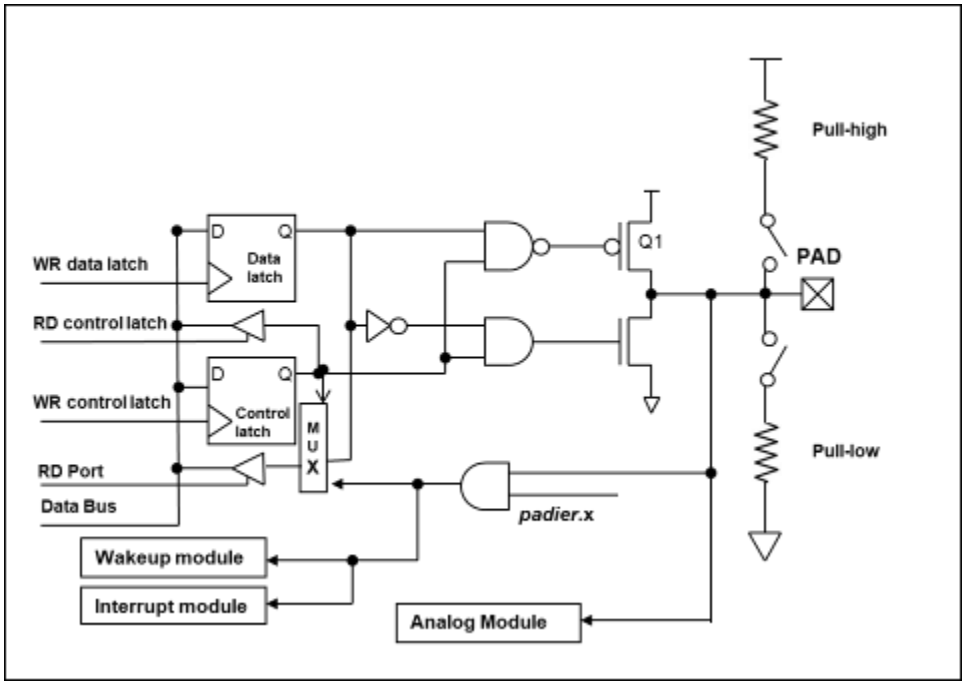


图 14：引脚缓冲区硬件图

9.2.2. IO 引脚的一般功能

(1) 输入、输出功能：

PFC232 所有 IO 引脚都可编程设定为数字输入或模拟输入、低输出或高输出。

透过数据寄存器(*PA/PB*)，控制寄存器(*PAC/PBC*)，上拉控制寄存器(*PAPH/PBPH*)和下拉控制寄存器(*PAPL/PBPL*)的设定，每一 IO 引脚都可以独立配置成不同的功能。

当引脚被用做模拟输入功能时，为减少漏电流，请关闭 *PxDIER* 寄存器相应位的数字输入功能。当引脚为输出低电位时，弱上拉/下拉电阻会自动关闭。

如果要读取端口上的电位状态，一定要先将端口设置成输入模式；在输出模式下，读取到的数据是数据寄存器的值。表 9 为端口 PA0 的设定配置表。

PA.0	*PAC*.0	*PAPH*.0	*PAPL*.0	描述
X	0	0	0	输入，没有弱上拉/下拉电阻
X	0	1	0	输入，有弱上拉电阻
X	0	0	1	输入，有弱下拉电阻
X	0	1	1	输入，有弱上拉/下拉电阻
0	1	X	X	输出低电位，没有弱上拉/下拉电阻
1	1	X	X	输出高电位，没有弱上拉/下拉电阻

表 9：PA0 设定配置表

(2) 睡眠唤醒功能：

当 PFC232 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* 相应位为高。

(3) 外部中断功能：

当 IO 作为外部中断引脚时，*PxDIER* 相应位应设置高。例，当 PA0 用来作为外部中断引脚时，*PADIER*.0 应设置高。

(4) 驱动能力可选：

PB4 和 PB7 可通过程序选项 PB4_PB7_Drive 来调整驱动电流和灌电流。

9.2.3. IO 使用与设定

(1) IO 作为数字输入

- 将 IO 设为数字输入时，Vih 与 Vil 的准位，会随着工作电压和温度有变动。请参考 Vih 最小值和 Vil 最大值。
- 内部上拉电阻值也将随着电压、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

- 用 *PxC* 寄存器，将 IO 设为输入。
- 用 *PxDIER* 寄存器，将对应的位设为 1 以启用数字输入。
- 为了防止 PA 中没有用到的 IO 口漏电，*PADIER*[1:2]需要常设为 0。

(3) 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。

(4) PA5 作为输入并通过长导线连接至按键或者开关

- 必需在 PA5 与长导线中间串接 >33 Ω 电阻。
- 应尽量避免使用 PA5 作为输入。

10. Timer / PWM 计数器

10.1. 16 位计数器 (Timer16)

10.1.1. Timer16 介绍

PFC232 内置一个 16 位硬件计数器 Timer16，其模块框图如图 15。

计数器时钟源由寄存器 *T16M*[7:5]来选择，在时钟送到 16 位计数器(counter16)之前，*T16M*[4:3]可对时钟进行预分频处理，有÷1、÷4、÷16、÷64 等四种选项，让计数范围更大。

T16M[2:0]用于选择 Timer16 的中断源，其来自于 16 位计数器的位 8 到位 15。当计数器溢出时，Timer16 就触发中断。经由寄存器 *INTEGS*.4，可选择中断类型是上升沿触发或下降沿触发。

16 位计数器只能向上计数，计数器初始值可以用 *stt16* 指令设定，计数器的数值可以用 *ldt16* 指令存储到数据存储器。

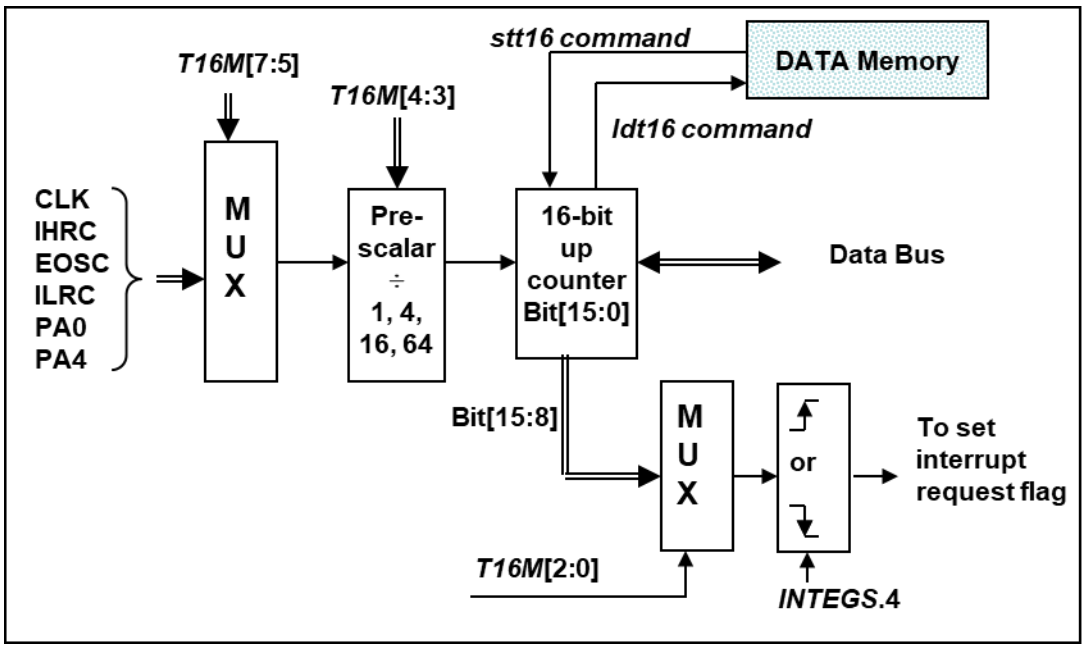


图 15：Timer16 模块框图

Timer16 的语法定义在.inc 文件中。*T16M* 共有三个配置参数，第一个参数用来定义 Timer16 的时钟源，第二个参数用来定义预分频器，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 / 4 = 4 MHz
// 则 SYSCLK/64 = 4 MHz/64 = 16 uS，约每 1 S 产生一次 INTRQ.2=1

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

$T16M      STOP;
// 停止 Timer16 计数
```

假如 Timer16 是不受干扰自由运行，中断发生的频率可以用下列式子描述：

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

其中，F 是 Timer16 的时钟源频率；

P 是 *T16M* [4:3]的选项(比如 1, 4, 16, 64)；

N 是中断要求选择的位，例如：选择位 10，那么 n=10。

10.1.2. 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。

10.1.3. Timer16 控制寄存器(*T16M*)，地址 = 0x06

位	初始值	读/写	描 述
7 – 5	000	读/写	Timer16 时钟选择： 000：停用 Timer16 001：CLK 系统时钟 010：保留 011：PA4 下降沿（外部事件） 100：IHRC 101：EOSC 110：ILRC 111：PA0 下降沿（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由高变低时，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

10.2. 8 位 PWM 计数器(Timer2,Timer3)

PFC232 内置 2 个 8 位 PWM 硬件定时器(Timer2/TM2,Timer3/TM3)，两个计数器的原理一样，以下以 Timer2 来说明，TM2 硬件框图请参考图 16。

寄存器 *TM2C*[7:4]用来选择定时器时钟；*TM2C*[3:2]用来选择 Timer2 的输出。寄存器 *TM2S*[6:0]用于选择时钟分频处理。寄存器 *TM2B* 用来控制定时器的计数上限，当计数值达到 *TM2B* 设定的上限时，定时器将自动清零。寄存器 *TM2C*T 用于设置或读取定时器的计数值。

8 位 PWM 定时器的工作模式有周期模式和 PWM 模式两种。周期模式用于输出固定周期波形；PWM 模式是用来产生 PWM 输出波形，PWM 分辨率可以为 6~8 位。

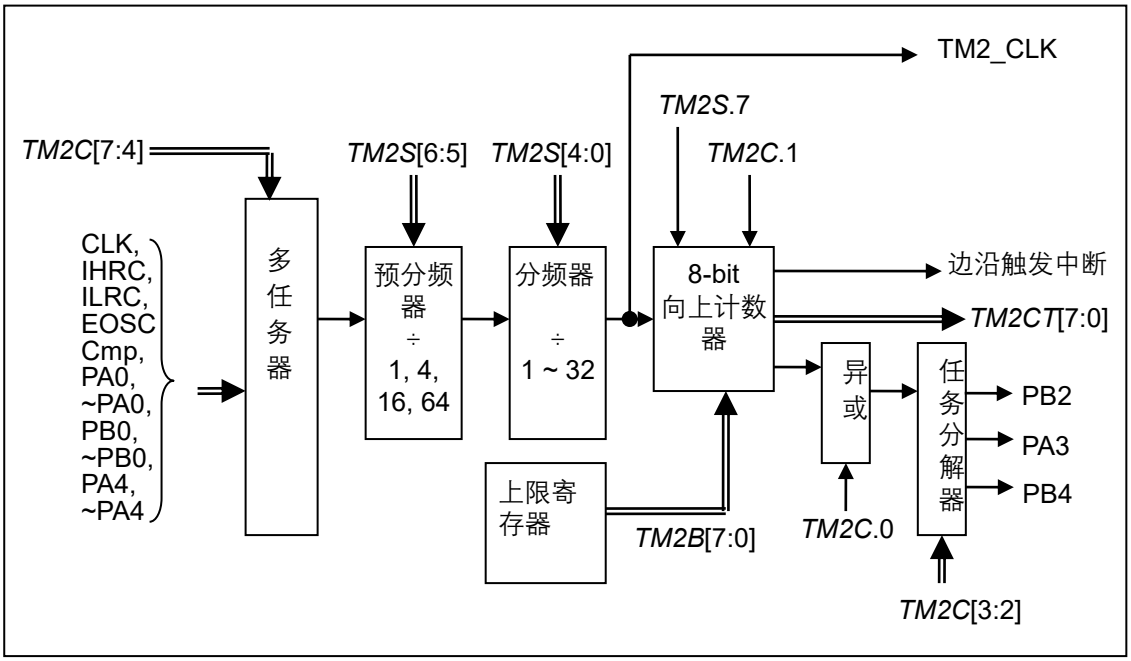


图 16：Timer2 模块框图

Timer3 的计数输出可选择为 PB5，PB6 或 PB7。

图 17 显示出 Timer2 周期模式和 PWM 模式的时序图：

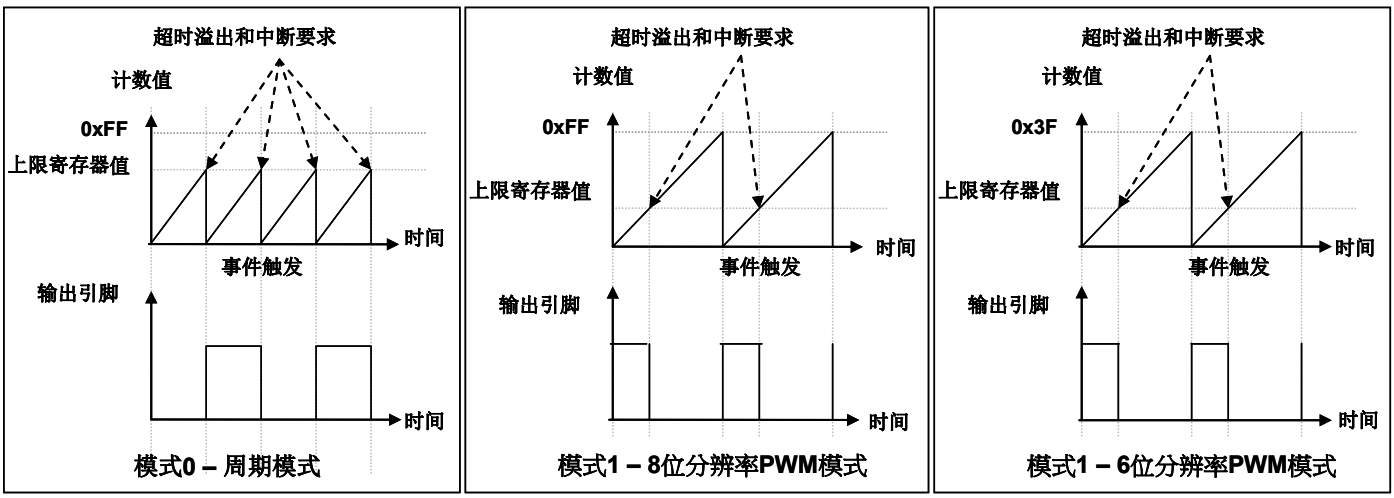


图 17：Timer2 周期模式和 PWM 模式的时序图

程序选项“GPC_PWM”和“OPA_PWM”是指由比较结果控制 PWM 波形。选用此功能后，当比较结果输出为 1 时，PWM 停止输出；比较结果输出为 0 时，PWM 恢复输出。如图 18 所示。

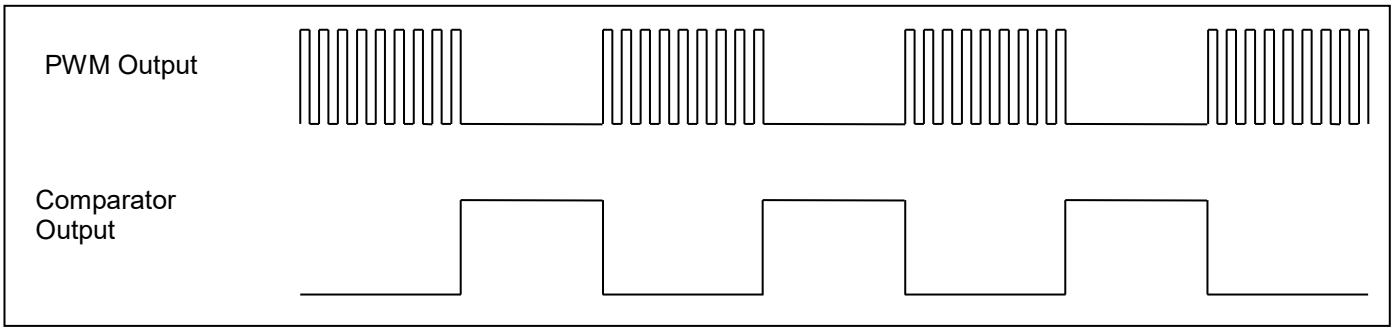


图 18：比较器控制 PWM 输出

10.2.1. Timer2、Timer3 相关寄存器

10.2.1.1. Timer2 上限寄存器(*TM2B*)，地址 = 0x09

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

10.2.1.2. Timer2 计数寄存器(*TM2CT*)，地址 = 0x1D

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

10.2.1.3. Timer2 分频寄存器(*TM2S*)，地址 = 0x1E

位	初始值	读/写	描 述
7	0	只写	PWM 分辨率选择。 0：8 位 1：6 位或者 7 位（由程序选项 TMx_bit 控制）
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。

10.2.1.4. Timer2 控制寄存器(*TM2C*)，地址 = 0x1C

位	初始值	读/写	描 述
7 – 4	0000	读/写	Timer2 时钟源选择。 0000：停用 0001：CLK 0010：IHRC 或者 IHRC*2 （由程序选项 TMx_source 控制） 0011：EOSC 0100：ILRC 0101：比较器输出 0110 : OPA（比较器模式）比较输出 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：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

10.2.1.5. Timer3 计数寄存器(*TM3CT*)，地址 = 0x33

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

10.2.1.6. Timer3 分频寄存器(TM3S)，地址= 0x34

位	初始值	读/写	描 述
7	0	只写	PWM 分辨率选择。 0：8 位 1：6 位或 7 位（由程序选项 TMx_bit 控制）
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。

10.2.1.7. Timer3 上限寄存器(*TM3B*)，地址 = 0x35

位	初始值	读/写	描 述
7 – 0	0x00	只写	Timer3 上限寄存器。

10.2.1.8. Timer3 控制寄存器(*TM3C*)，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 0010：IHRC 或者 IHRC*2 （由程序选项 TMx_source 控制） 0011：EOSC 0100：ILRC 0101：比较器输出 0110 : OPA（比较器模式）比较输出 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 Timer3 定时器时钟，当 ICE 停下时，发送到定时器的时钟是不停止，定时器仍然继续计数。
3 – 2	00	读/写	Timer3 输出选择。 00：停用 01：PB5 10：PB6 11：PB7
1	0	读/写	Timer3 模式选择。 0：周期模式 1：PWM 模式
0	0	读/写	启用 Timer3 反极性输出。 0 / 1：停用/启用。

10.2.2. 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=4MHz
TM2B = 0b0111_1111, K=127
TM2S = 0b0_00_00000, S1=1, S2=0
➔ 输出频率 = 4MHz ÷ [2 × (127＋1) × 1 × (0＋1)] = 15.625KHz

例 2：

TM2C = 0b0001_1100, Y=4MHz
TM2B = 0b0000_0001, K=1
TM2S = 0b0_00_00000, S1=1, S2=0
➔ 输出频率 = 4MHz ÷ [2 × (1＋1) × 1 × (0＋1)] =1MHz

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/4, 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;
    }
}
```

10.2.3. 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=4MHz
TM2B = 0b0111_1111, K=127
TM2S = 0b0_00_00000, S1=1, S2=0
➔ 输出频率 = 4MHz ÷ (256 × 1 × (0+1)) = 15.625KHz
➔ 输出空占比 = [(127+1) ÷ 256] × 100% = 50%

例 2：

TM2C = 0b0001_1110, Y=4MHz
TM2B = 0b0000_1001, K = 9
TM2S = 0b0_00_00000, S1=1, S2=0
➔ 输出频率 = 4MHz ÷ (256 × 1 × (0+1)) = 15.625KHz
➔ 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/4, 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_1_0;       //系统时钟，输出 = PA3，PWM 模式
    while(1)
    {
        nop;
    }
}
```

10.2.4. 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=4MHz

TM2B = 0b0011_1111, K=63

TM2S = 0b1_00_00000, S1=1, S2=0

➔ 输出频率 = 4MHz ÷ (64 × 1 × (0+1)) = 62.5KHz

➔ 输出空占比 = [(63+1) ÷ 64] × 100% = 100%

例 2：

TM2C = 0b0001_1110, Y=4MHz

TM2B = 0b0000_0000, K=0

TM2S = 0b1_00_00000, S1=1, S2=0

➔ 输出频率 = 4MHz ÷ (64 × 1 × (0+1)) = 62.5KHz

➔ 输出空占比 = [(0+1) ÷ 64] × 100% =1.5%

10.3. 11 位 PWM 计数器

PFC232 有三个 11 位的 PWM 生成器(PWMG0, PWMG1 & PWMG2)。以 PWMG0 为例说明 11 位 PWM 的用法，其他两路用法类似。每路 PWM 输出 IO 可选如下：

(1) PWMG0 – PA0, PB4, PB5
(2) PWMG1 – PA4, PB6, PB7
(3) PWMG2 – PA3, PB2, PB3, PA5（仿真器不支持 PA5）

10.3.1. PWM 波形

PWM 波形（图 19）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, $2^N \times T_{clock} = T_{Period}$）。

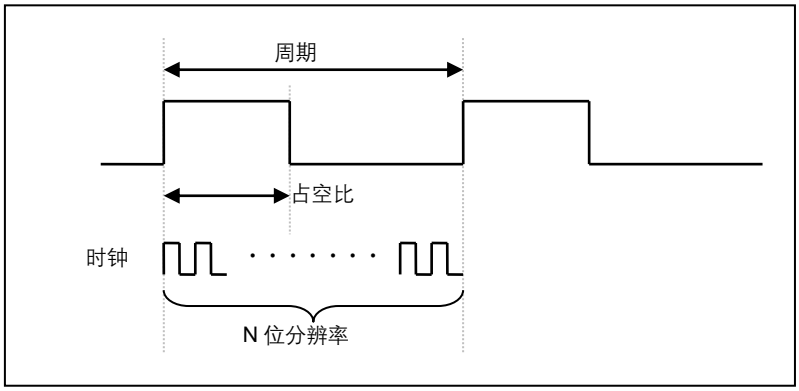


图 19：PWM 输出波形

10.3.2. 硬件和时钟框图

图 20 是 11 位计数器 PWMG0 的硬件框图。这个计数器的时钟源可以是 IHRC 或者系统时钟。寄存器 *PWMG0C* 用来选择其 PWM 的输出端口。PWM 的周期由寄存器 *PWMG0CUBH* 和 *PWMG0CUBL* 决定，PWM 的占空比由寄存器 *PWMG0DTH* 和 *PWMG0DTL* 决定。

程序选项 GPC_PWM，可选择由比较器结果控制 PWM 波形的输出。参考 Timer2 章节。

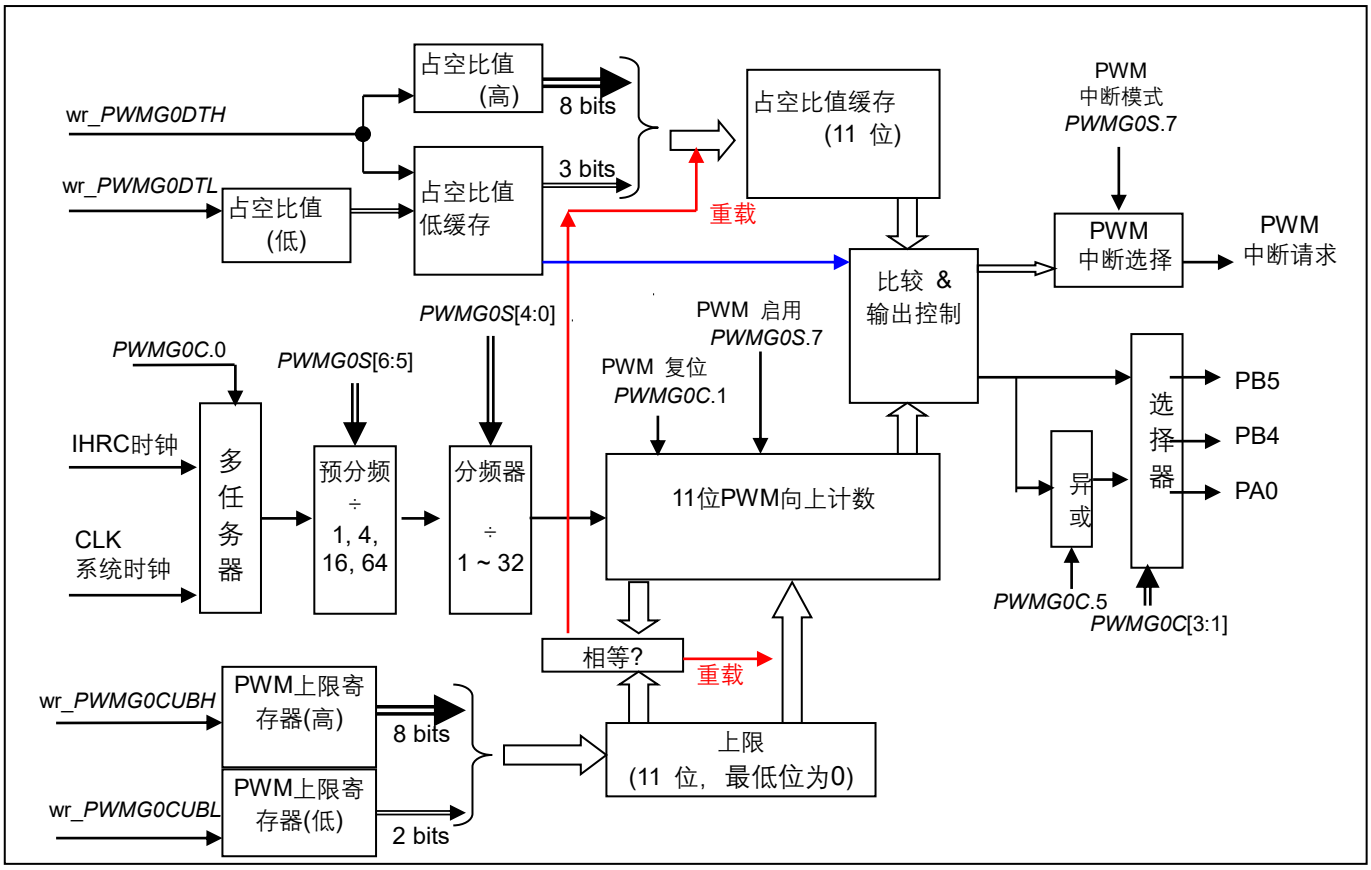


图 20：11 位 PWM 生成器（PWMG0）硬件框图

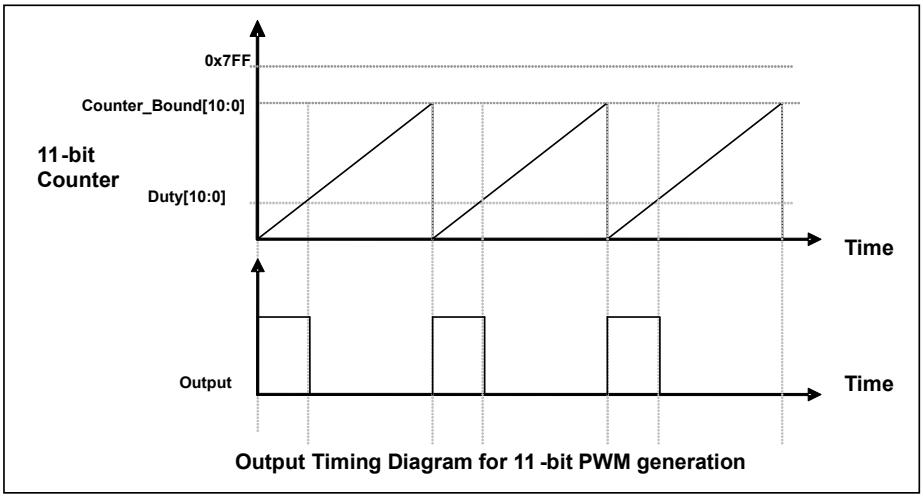


图 21 ： 11 位 PWM 生成器(PWMG0)的输出时序图

10.3.3. 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

10.3.4. 11bit PWM 计数器相关寄存器

10.3.4.1. PWMG0 控制寄存器(*PWMG0C*)，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： 000：不输出 001：PB5 011：PA0 100：PB4 其他：保留
0	0	读/写	PWMG0 时钟源。0：CLK，1：IHRC 或者 IHRC*2 （由程序选项 PWM_source 控制）

10.3.4.2. PWMG0 分频寄存器(*PWMG0S*)，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。

10.3.4.3. PWMG0 占空比高位寄存器(*PWMG0DTH*)，地址 = 0x22

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

10.3.4.4. PWMG0 占空比低位寄存器(*PWMG0DTL*)，地址 = 0x23

位	初始值	读/写	描 述
7 – 5	-	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*。

10.3.4.5. PWMG0 计数上限高位寄存器(*PWMG0CUBH*)，地址= 0x24

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

10.3.4.6. PWMG0 计数上限低位寄存器(*PWMG0CUBL*)，地址= 0x25

位	初始值	读/写	描 述
7 – 6	-	只写	PWMG0 上限寄存器位[2:1] 。
5	-	只写	PWMG0 上限寄存器位[0]。
4 – 0	-	-	保留。

10.3.4.7. PWMG1 控制寄存器(*PWMG1C*)，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 PWMG1 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： 000：不输出 001：PB6 011：PA4 100：PB7 其他：保留
0	0	读/写	PWMG1 时钟源。0：CLK，1：IHRC 或者 IHRC*2 （由程序选项 PWM_source 控制）

10.3.4.8. 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 预分频。 00：÷1 01：÷4 10：÷16 11：÷64
4 – 0	0	只写	PWMG1 分频。

10.3.4.9. PWMG1 占空比高位寄存器(*PWMG1DTH*)，地址 = 0x28

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

10.3.4.10. PWMG1 占空比低位寄存器(*PWMG1DTL*)，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*。

10.3.4.11. PWMG1 计数上限高位寄存器(*PWMG1CUBH*)，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] 。

10.3.4.12. PWMG1 计数上限低位寄存器(*PWMG1CUBL*)，地址= 0x2B

位	初始值	读/写	描 述
7 – 6	00	只写	PWMG1 上限寄存器位[2:1] 。
5	0	只写	PWMG1 上限寄存器位[0]。
4 – 0	-	-	保留。

10.3.4.13. PWMG2 控制寄存器(*PWMG2C*)，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 2。
3 – 1	0	读/写	选择 PWMG2 输出： 000：不输出 001：PB3 011：PA3 100：PB2 101：PA5（仿真器不支持） 其他：保留
0	0	读/写	PWMG2 时钟源。0：CLK，1：IHRC 或者 IHRC*2 （由程序选项 PWM_source 控制）

10.3.4.14. PWMG2 分频寄存器(*PWMG2S*)，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 预分频。 00：÷1 01：÷4 10：÷16 11：÷64
4 – 0	0	只写	PWMG2 分频。

10.3.4.15. PWMG2 占空比高位寄存器(*PWMG2DTH*)，地址 = 0x2E

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

10.3.4.16. PWMG2 占空比低位寄存器(*PWMG2DTL*)，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 *PWMG2DTH*。

10.3.4.17. PWMG2 计数上限高位寄存器(*PWMG2CUBH*)，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] 。

10.3.4.18. PWMG2 计数上限低位寄存器(*PWMG2CUBL*)，地址= 0x31

位	初始值	读/写	描 述
7 – 6	00	只写	PWMG2 上限寄存器位[2:1] 。
5	0	只写	PWMG2 上限寄存器位[0]。
4 – 0	-	-	保留。

10.3.5. 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 22 所示。

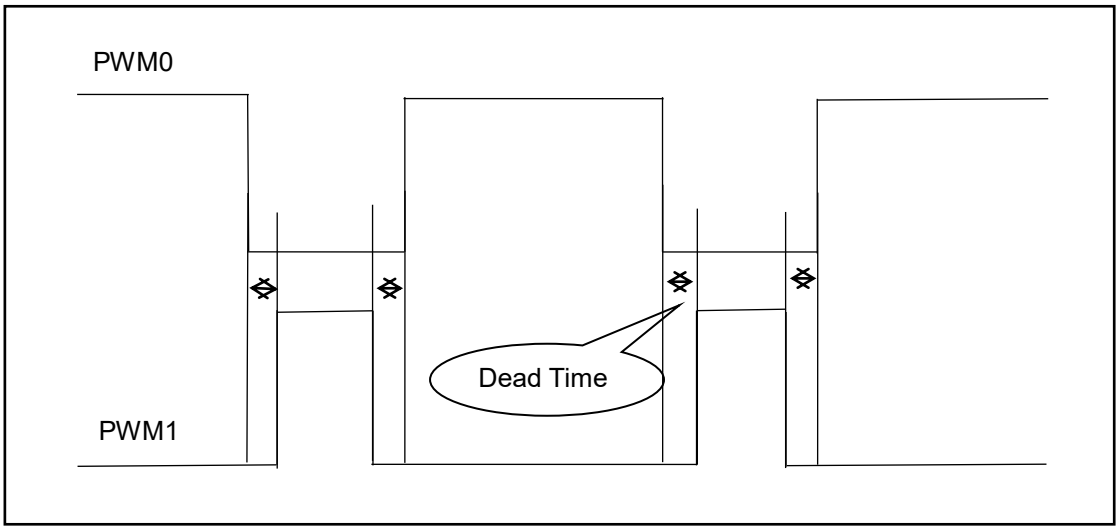


图 22 ：两路互补 PWM 波形

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 10 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 10：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
...	...

11. 特殊功能

11.1. 比较器

PFC232 内置一个硬件比较器，硬件框图如图 23。它可以比较两个输入端之间的信号大小。进行比较的两个信号，一个是正输入，另一个是负输入。正输入由寄存器 *GPCC*.0 选择；负输入由 *GPCC*[3:1] 选择。

比较器输出的结果可以：

(1) 由 *GPCC*.6 读取出来；

(2) 由 *GPCC*.4 选择输出信号是否反极性；

(3) 由 *GPCC*.5 选择是否由 Time2(TM2_CLK)采样输出；

(4) 由 *GPCS*.7 选择是否输出到 PA0；

(5) 产生中断信号。

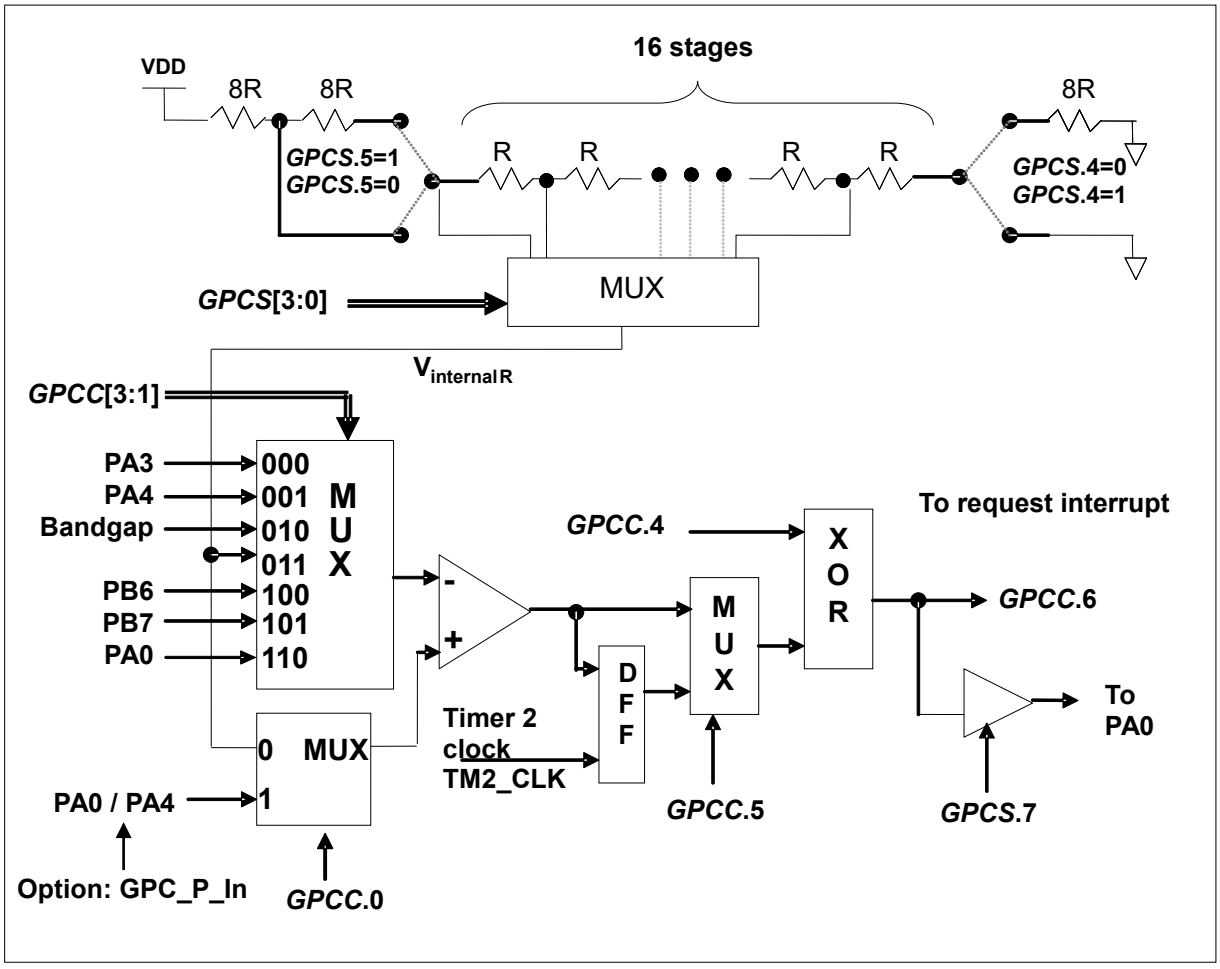


图 23 ： 比较器硬件图框

11.1.1. 比较器控制寄存器(*GPCC*)，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：PB6 101：PB7 110：PA0 111：保留
0	0	读/写	选择比较器正输入的来源。 0：$V_{internal\ R}$ 1：PA4 或 PA0（由程序选项 GPC_P_In 决定）

11.1.2. 比较器选择寄存器(*GPCS*)，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。 在设置这一位启用比较器输出到 PA0 之前，请确保 OPA 没有启用，以避免信号冲突。
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）

11.1.3. 内部参考电压 ($V_{internal\ R}$)

内部参考电压 $V_{internal\ R}$ 由一连串电阻组成，可以通过寄存器 *GPCS*[5:0]来设置具体数值，范围从 (1/32)*VDD 到 (3/4)*VDD。寄存器 *GPCS* 的位 4 和位 5 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。

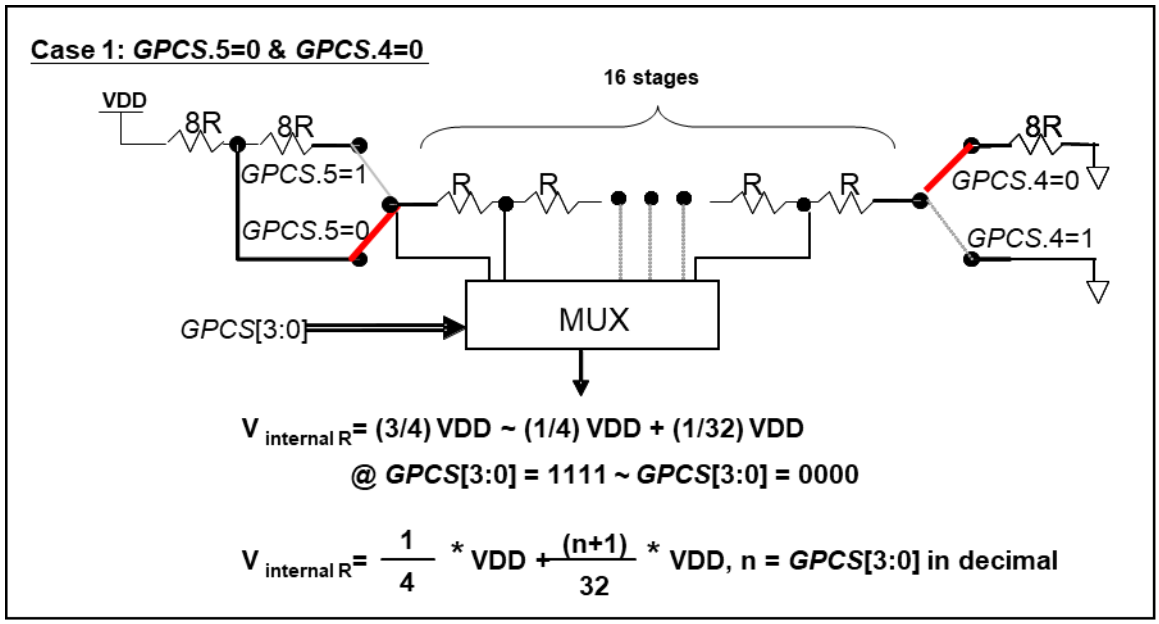


图 24 ： $V_{internal\ R}$ 硬件接法 (*GPCS*.5=0 & *GPCS*.4=0)

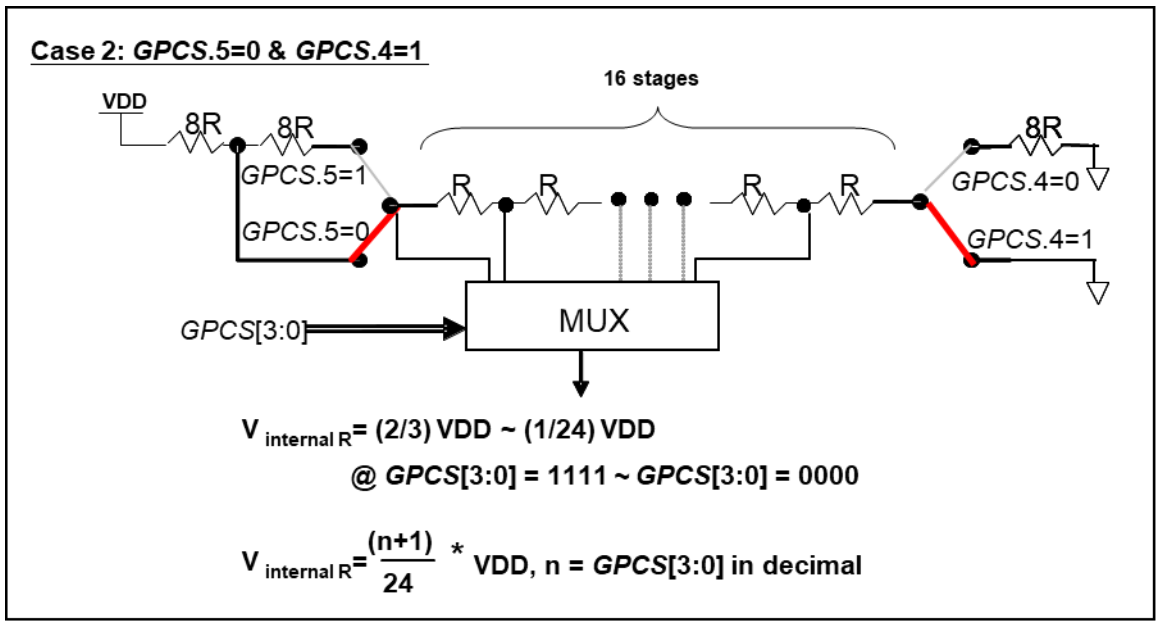


图 25： $V_{internal\ R}$ 硬件接法 (*GPCS*.5=0 & *GPCS*.4=1)

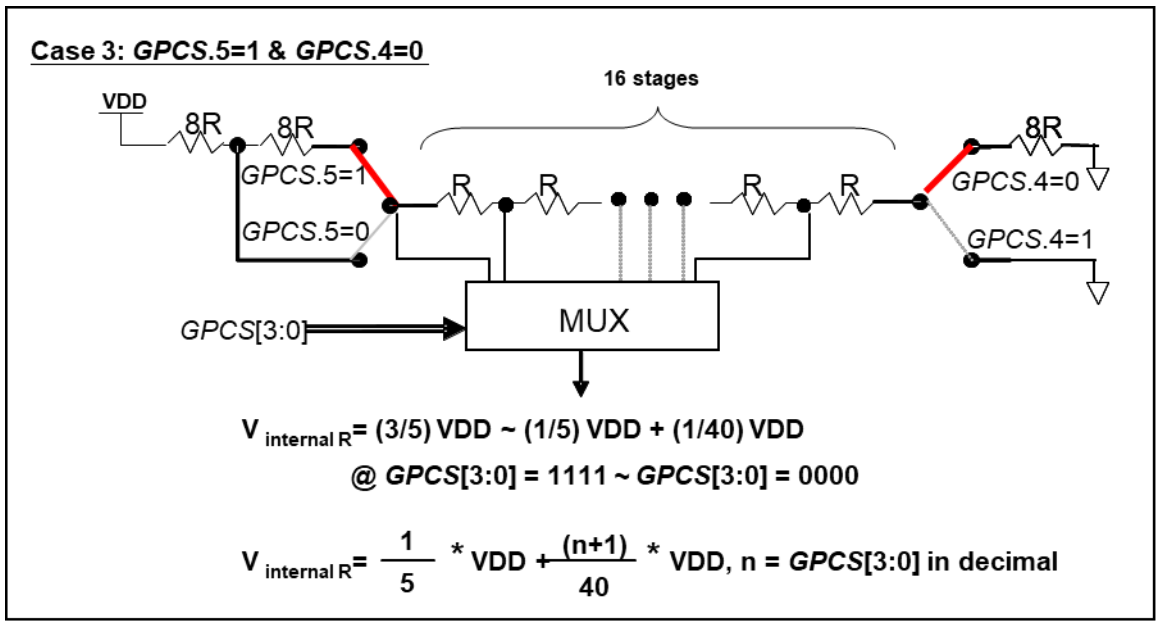


图 26 ： $V_{internal\ R}$ 硬件接法 (*GPCS*.5=1 & *GPCS*.4=0)

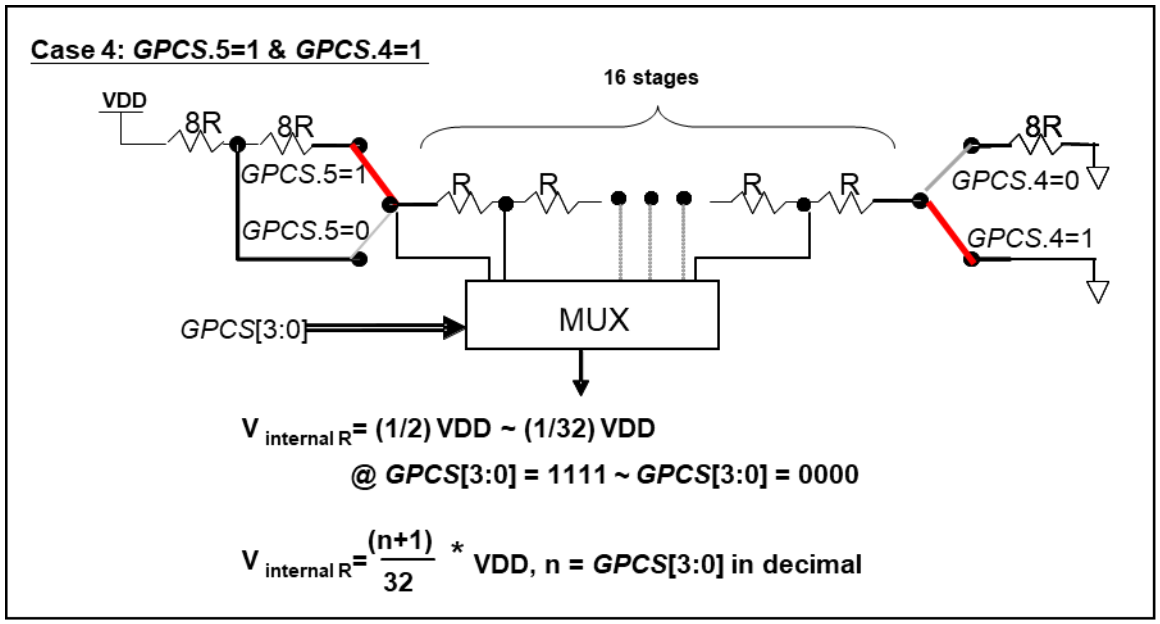


图 27 ： $V_{internal\ R}$ 硬件接法 (*GPCS*.5=1 & *GPCS*.4=1)

11.1.4. 使用比较器

例一：

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

```
GPCS    = 0b0_0_00_1001;          // Vinternal R = VDD*(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）
```

或者

```
$ 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)*VDD，选择 PA4 为正输入，比较器的结果反极性并输出到 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)*VDD + [(13+1)/40]*VDD = [(13+9)/40]*VDD = (22/40)*VDD。

```
GPCS    = 0b1_0_10_1101;          // 输出到 PA0，Vinternal R = VDD*(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）
```

或者

```
$ 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：当选择 PA0 做比较器结果输出时，*GPCS* 会影响 PA3 的仿真输出功能，但不影响实际 IC 的功能，请在仿真时需避开这个情况。

11.1.5. 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 VDD 的电压就可以透过下列公式计算：

对于 Case 1 而言：V_{DD} = [32 / (N+9)] * 1.20 volt ;
对于 Case 2 而言：V_{DD} = [24 / (N+1)] * 1.20 volt ;
对于 Case 3 而言：V_{DD} = [40 / (N+9)] * 1.20 volt ;
对于 Case 4 而言：V_{DD} = [32 / (N+1)] * 1.20 volt ;

例一：

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

11.2. VDD/2 偏置电压产生器

PFC232 的 PA0、PA3、PA4、PB0 和 PB3 这五支引脚可以产生 VDD/2 以作为驱动液晶显示器时 COM 的功能，该功能可以通过设置寄存器 *MISC*.4 为 1 来启用。

杂项寄存器(*MISC*)，地址 = 0x08			
位	初始值	读/写	描 述
7 – 5	-	-	保留。请保持为 0。
4	**0**	**只写**	**使能 LCD 显示 VDD/2 功能。 0 / 1：停用 / 启用** **（仿真器不能动态切换）**
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

COM 端口通过进入输入模式(*PAC*.x / *PBC*.x=0)就能输出 VDD/2 电压。但是注意要关闭上拉电阻 *PAPH*.x / *PBPH*.x 和数字输入 *PADIER*.x / *PBDIER*.x 防止输出电压受到干扰。图 28 显示了如何使用此功能。

COM 端口的输出功能与其他正常的 IO 一样。

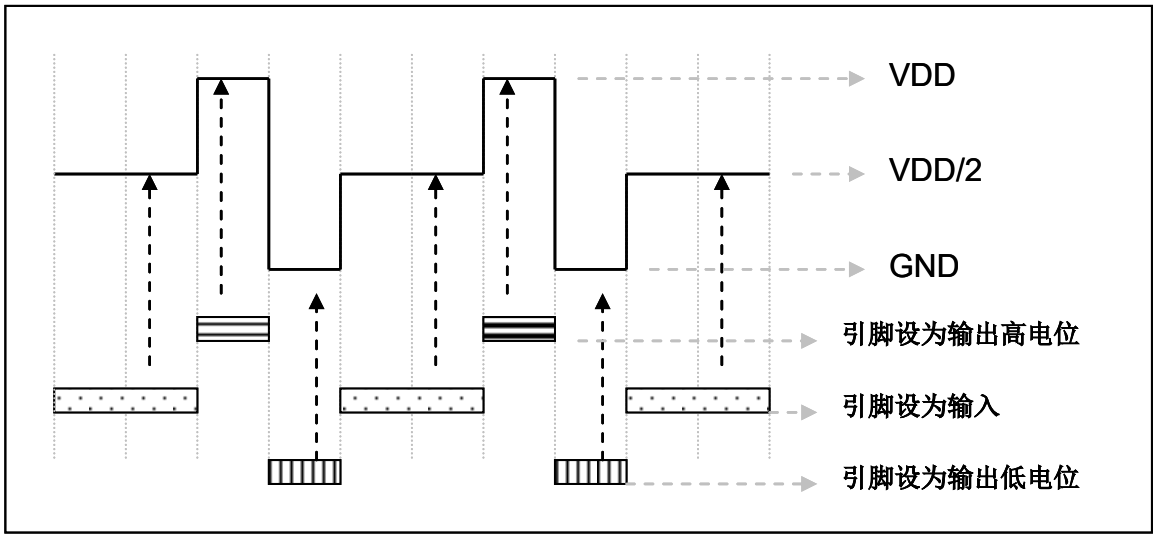
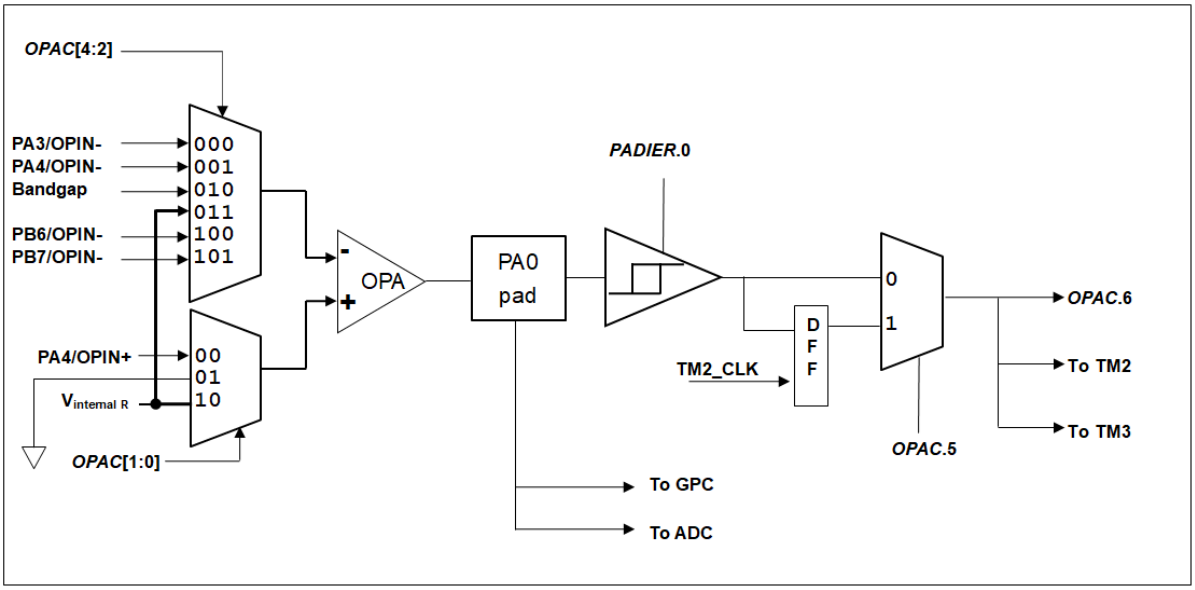


图 28：使用 VDD/2 偏置电压产生器

注意：仿真器不支持 PB3 的 VDD/2 功能。

11.3. 运算放大器(OPA)模块

PFC232 内置一个运算放大器(OPA)模块，其基本配置如下图所示。运算放大器有两种不同的结构，一种是 OPA 比较器模式，另一种是 OPA 放大器模式。



当用户通过启用 *OPAC*.7 打开 OPA 时，IO 端口 PA0 将成为 OPA 的输出。用户可以在放大器模式下通过 GPC 或 ADC 测量模拟 PA0 电压，也可以在比较器模式下直接读取 PA0 的数字 OPA 比较结果。

注意：仿真器不支持 OPA。

11.3.1. OPA 比较器模式

上图中所示的开环配置称为 OPA 比较器模式。OPA 的输入输出之间没有反馈路径(PA0)。在此模式下，用户可以启用 *PADIER*.0 从 *OPAC*.6 中读取比较结果。

与比较器(GPC)类似，OPA 的比较结果也可以作为 TM2 或 TM3 的计数源。与 GPC 一样，OPA 比较结果也可以通过程序选项 OPA_PWM 控制生成的 PWM 波形。

此外，用户可以选择 GPC 中产生的内部参考电压 $V_{internal\ R}$ 作为 OPA 正负输入之一作为比较参考电压。

11.3.2. OPA 放大器模式

OPA 的另一种结构称为放大器模式，它需要一些外部组件来制作反馈放大器。当它被配置为一个反馈放大器时，PA0 成为一个模拟输出引脚。请始终记住禁用 *PADIER*.0 以防止漏电。

PA0 可以被选择作为比较器(GPC)或 ADC 的输入，因此 OPA 的输出电压可以用 GPC 进行比较，也可以用 ADC 进行测量。

11.3.3. OPA 控制寄存器(*OPAC*)，地址 = 0x1A

位	初始值	读/写	描 述
7	0	读/写	启用 OPA. 0 / 1 : 停用 / 启用 当此位设置为启用时，还请将相应的模拟输入引脚设置为数字禁用，以防止 IO 漏电。 请注意，当 OPA 的输出被启用时，PA0 将被分配给输出。
6	-	只读	OPA 比较器模式下的比较结果 0: 正输入 < 负输入 1: 正输入 >负输入
5	0	读/写	选择 OPA 输出的比较结果是否由 TM2_CLK 采样输出 0：比较结果没有 TM2_CLK 采样输出 1：比较结果是由 TM2_CLK 采样输出
4 – 2	000	读/写	选择 OPA 负输入的来源 000 : PA3 001 : PA4 010 : 内部 1.20 V bandgap 参考电压 011 : $V_{internal\ R}$ 100 : PB6 101 : PB7 11X: 保留
1 – 0	00	读/写	选择 OPA 正输入的来源 00 : PA4 01 : GND 10 : 来自于比较器的 $V_{internal\ R}$（请参考 *GPCS* 寄存器） 11 : 保留

11.3.4. OPA 失调寄存器(*OPAOFS*)，地址 = 0x07

位	初始值	读/写	描 述
7 – 4	-	-	保留
3 – 0	0000	读/写	选择 OPA 的失调电压水平 0000 : +1mV 0001 : +2mV 0010 : +5mV 0011 : +10mV 0100 : +15mV 0101 : +20mV 0110 : +25mV 0111 : +30mV 1000 : -1mV 1001 : -2mV 1010 : -5mV 1011 : -10mV 1100 : -15mV 1101 : -20mV 1110 : -25mV 1111 : -30mV

11.4. 模拟-数字转换器(ADC) 模块

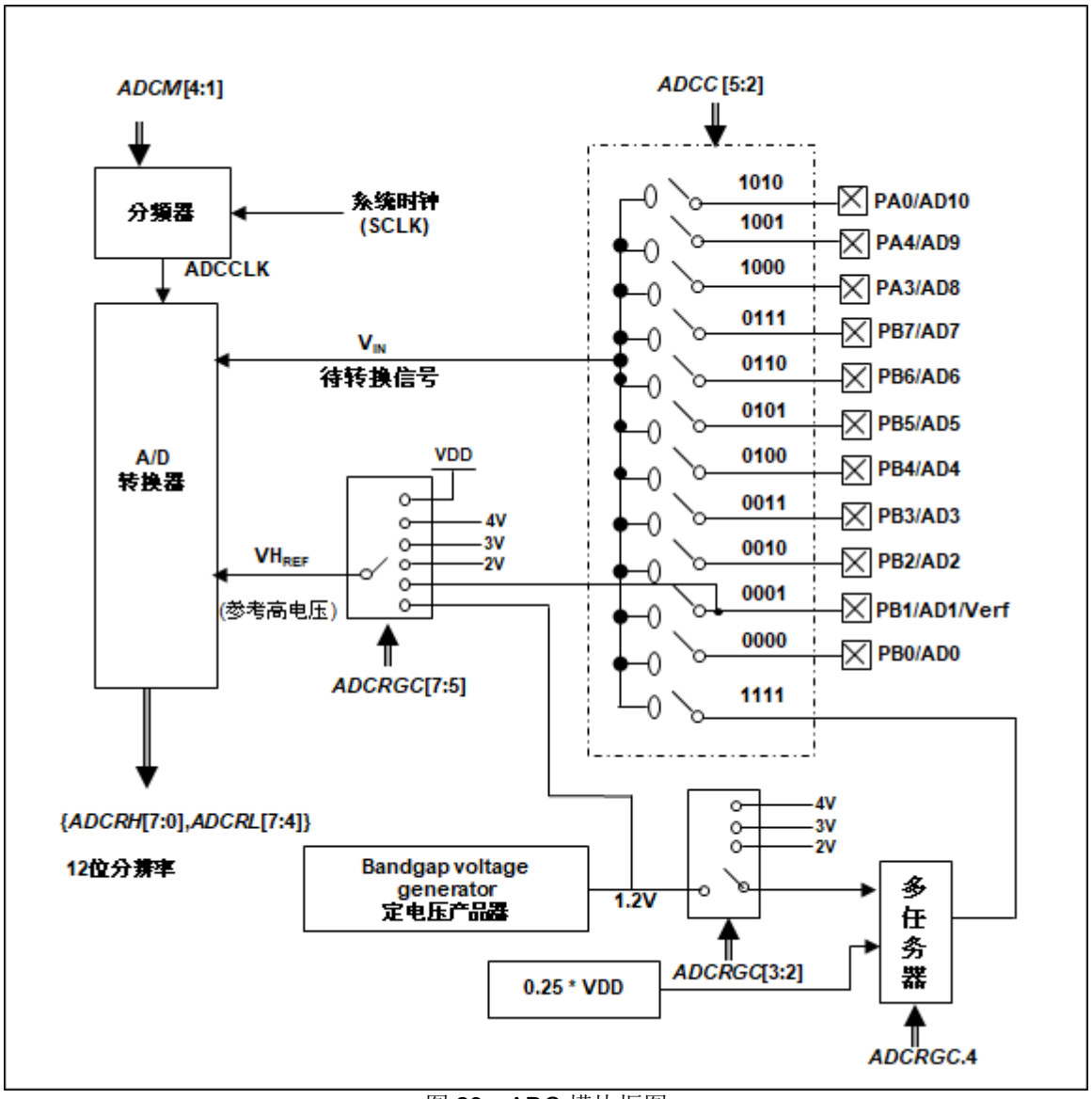


图 29：ADC 模块框图

完成 AD 转换过程需要以下步骤：

(1) 通过寄存器 *ADCRGC* 配置参考高电压
(2) 通过 *ADCM* 寄存器配置 AD 转换时钟信号
(3) 通过 *PADIER*、*PBDIER* 寄存器配置模拟输入引脚
(4) 通过 *ADCC* 寄存器选择 ADC 输入通道
(5) 通过 *ADCC* 寄存器启用 ADC 模块
(6) 启用 ADC 模块之后，延迟一段时间

条件 1：使用 bandgap 1.2V 或 2V/3V/4V 相关电路时，无论是将其用作内部参考高电压还是作为 AD 输入通道，所需的延迟时间必须超过 1ms；如果 200 个 AD 时钟已经超过 1ms，那么延迟时间只需要 200 个 AD 时钟即可。当启用内部 BG/2v/3v/4v 为参考高电压时，必须保证 IHRC 为开启状态。

条件 2：没有使用任何 bandgap 1.2V 或 2V/3V/4V 相关电路，延迟时间仅需 200 个 AD 时钟。

另注意：以上两条件所涉及的 200 个 AD 时钟，该时钟是指由 *ADCM* 寄存器配置后的 ADC 转换时钟而非系统时钟 SYSCLK。

(7) 执行 AD 转换并检查 ADC 转换数据是否已经完成 ADCC.6 设置 1 开启 AD 转换并且检测 ADCC.6 是否是‘1’。

(8) 从 ADC 寄存器读取转换结果：
先读取 *ADCRH* 寄存器的值然后再读取 *ADCRL* 寄存器的值。

应用时，如果是关掉 ADC 模块后再重新启用 ADC 的情况下，或者在切换 ADC 参考电压及输入通道时，进行 ADC 转换之前请重新执行如上步骤 6，确保 ADC 模块已经准备好。

11.4.1. AD 转换的输入要求

为了满足 AD 转换的精度要求，电容的保持电荷(C_{HOLD})必须完全充电到参考高电压的水平和放电到参考低电压的水平。模拟输入电路模型如图 20 所示，信号驱动源阻抗(Rs)和内部采样开关阻抗(Rss)会直接影响到 电容 C_{HOLD} 充电所需求的时间。内部采样开关的阻抗可能会因 ADC 充电电压而产生变化；信号驱动源阻抗会影响模拟输入信号的精度。使用者必须确保在采样前，被测信号的稳定，因此，信号驱动源阻抗的最大值与被测信号的频率高度相关。建议，在输入频率为 500khz 下，模拟信号源的最大阻抗值不要超过 10KΩ。

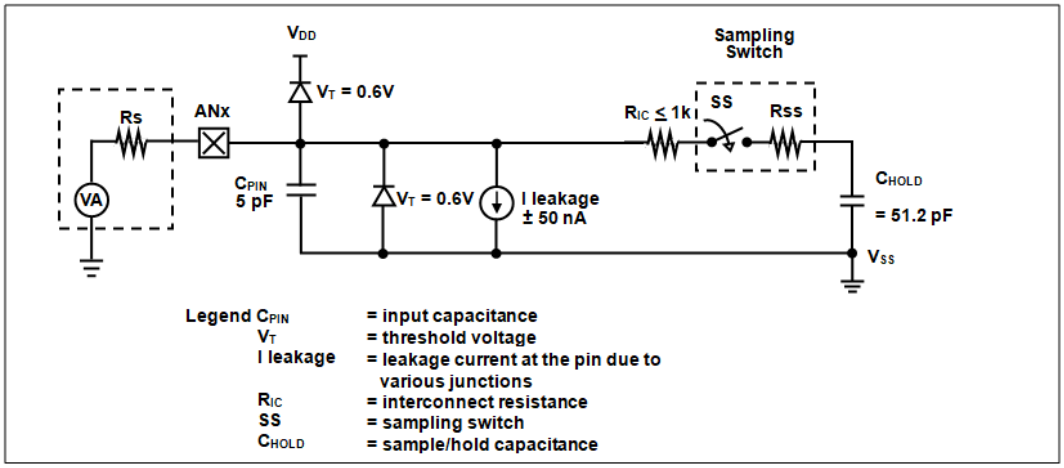


图 30：模拟输入模型

在使用 AD 转换之前，必须确认所选的模拟输入信号的采集时间应符合要求，ADCLK 的选择必须满足最短信号采集时间。

11.4.2. 选择参考高电压

ADC 参考高电压能够通过寄存器 *ADCRGC*[7:5]来选择，并且它的选择有 V_{DD}，4V，3V，2V，bandgap (1.20V)参考电压或者来自 PB1 外部引脚。

11.4.3. ADC 时钟选择

ADC 模块的时钟(ADCLK)能够通过 *ADCM* 寄存器来选择，ADCLK 从 CLK÷1 到 CLK÷128 一共有 8 个选项可被选择（CLK 是系统时钟）。由于信号采集时间 T_{ACQ} 是 ADCLK 的一个时钟周期，所以 ADCLK 必须满足这要求，建议 ADC 时钟周期是 2us。

11.4.4. 配置模拟引脚

有 12 模拟信号可以被 AD 转换选择：11 来自外部引脚的模拟输入信号和一个 bandgap 参考电压或者 0.25*V_{DD}。Bang-gap 有 4 级电压可供选择，分别是：1.2V，2V，3V 和 4V。以外部引脚而言，12 个模拟信号与 Port A[0]，Port A[3]，Port A[4]，和 Port B[7:0]共享引脚。为了避免漏电，这些引脚在使用时定义为模拟输入并应停用数字输入功能（设置 *PADIER* / *PBDIER* 寄存器的相应位为 0）。

ADC 的测量信号属于小信号，为避免测量信号在测量期间被干扰，被选定的引脚应：

(1) 设为输入模式，

(2) 关闭弱上拉/下拉电阻，

(3) 通过端口 A/B 寄存器（*PADIER* / *PBDIER*）设置模拟输入并关闭数字输入。

11.4.5. 使用 ADC

下面的示例演示使用 PB0～PB3 来当 ADC 输入引脚。

首先，定义所选择的引脚：

```
PBC =    0B_XXXX_0000;        //   PB0 ~ PB3 作为输入
PBPH     =    0B_XXXX_0000;        //   PB0 ~ PB3 没有弱上拉电阻
PBPL=    0B_XXXX_0000;        //   PB0 ~ PB3 没有弱下拉电阻
PBDIER   =    0B_XXXX_0000;   //   PB0 ~ PB3 停用数字输入
```

下一步，设定 *ADCC* 寄存器，示例如下：

```
$   ADCC Enable, PB3;          //   设置 PB3 作为 ADC 输入
$   ADCC Enable, PB2;          //   设置 PB2 作为 ADC 输入
$   ADCC Enable, PB0;          //   设置 PB0 作为 ADC 输入
                                    //   注：每次 AD 转换只能选择一个输入通道
```

下一步，设定 *ADCM* 和 *ADCRGC* 寄存器，示例如下：

```
$   ADCM 12BIT, /16;           //   建议 /16 @系统时钟=8MHz，建议 ADCLK=500KHz
$   ADCM 12BIT, /8;            //   建议 /8 @系统时钟=4MHz，建议 ADCLK=500KHz
$   ADCRGC   VDD;                  //   参考电压为 VDD，延时 200 个 ADCLK 即可
```

下一步，延迟一段时间（ADCLK=500KHz，200*ADCLK=400us），示例如下：

```
.Delay 8*400;                      //系统时钟=8MHz
.Delay 4*400;                      //系统时钟=4MHz
```

注意：若使用内部参考高电压如 bandgap 1.2V 或 2V，3V，4V 时，所需延迟时间必须超过 1ms

```
$   ADCRGC   3V;                   //   AD 参考电压为 3V
.Delay 4*1010;                     //   假设系统时钟=4MHz，延时 1ms 以上
```

注意：若使用 bandgap 1.2V 或 2V，3V，4V 作为 ADC 输入通道时，所需延迟时间同样必须超过 1ms

```
$   ADCC   ADC
$   ADCRGCVDDADC_BGBG_2V   //   参考电压为 VDD，输入通道为 BG_2V
.Delay 4*1010;                     //   假设系统时钟=4MHz，延时 1ms 以上
```

接着，开始 ADC 转换：

```
AD_START    =    1;                //   开始 ADC 转换
while (! AD_DONE)   NULL;          //   等待 ADC 转换结果
```

最后，当 AD_DONE 高电位时读取 ADC 结果：

```
WORD          Data;                //   两字节结果：放在 ADCRH 和 ADCRL
Data$1    =    ADCRH;
Data$0    =    ADCRL;
Data      =    Data >> 4;
```

ADC 也可以利用下面方法停用：

```
$   ADCC Disable;
```

或

```
ADCC          =    0;
```

11.4.6. ADC 相关寄存器

11.4.6.1. ADC 控制寄存器(*ADCC*), 地址 = 0x35

位	初始值	读/写	描述
7	0	读/写	启用 ADC 功能. 0/1：停用/启用。
6	0	读/写	ADC 转换进程控制位： 读到 “1”表明 ADC 已经准备好，或已转换完成。
5 – 2	0000	读/写	通道选择。 以下 4 位用来选择 AD 转换的输入信号: 0000：PB0/AD0, 0001：PB1/AD1, 0010：PB2/AD2, 0011：PB3/AD3, 0100：PB4/AD4, 0101：PB5/AD5, 0110：PB6/AD6, 0111：PB7/AD7, 1000：PA3/AD8, 1001：PA4/AD9, 1010：PA0/AD10, 1111：（通道 F） Bandgap 参考电压或者 0.25*V_{DD} 其他：保留。
0 – 1	-	-	保留（写 0）。

11.4.6.2. ADC 模式寄存器(*ADCM*), 地址 = 0x36

位	初始值	读/写	描述
7 – 5	000	只写	位分辨率。 100:12-bit, AD 12-bit result [11:0] = { adcrh[7:0], adcrl[7:4] }. 其它：保留。
4	-	-	保留。
3 – 1	000	只写	ADC 时钟源选择： 000：CLK（系统时钟）÷ 1, 001：CLK（系统时钟）÷ 2, 010：CLK（系统时钟）÷ 4, 011：CLK（系统时钟）÷ 8, 100：CLK（系统时钟）÷ 16, 101：CLK（系统时钟）÷ 32, 110：CLK（系统时钟）÷ 64, 111：CLK（系统时钟）÷ 128,
0	-	-	保留。

11.4.6.3. ADC 调节控制寄存器(*ADCRGC*), 地址 = 0x39

位	初始值	读/写	描述
7 – 5	000	只写	以下 3 位用来选择 ADC 输入信号的参考电压: 000：V_{DD}, 001：2V, 010：3V, 011：4V, 100：PB1, 101：Bandgap 1.20V 参考电压 其他: 保留。
4	0	只写	ADC 通道 F 选择器： 0：Bandgap 参考电压, 1：0.25*V_{DD}（电压偏移±0.01*V_{DD}）。
3 – 2	00	只写	ADC 通道 F 的 Bandgap 参考电压选择: 00：1.2V 01：2V 10：3V 11：4V
1 – 0	-	-	保留（写 0）。

11.4.6.4. ADC 数据高位寄存器(*ADCRH*), 地址 = 0x37

位	初始值	读/写	描述
7 – 0	-	只读	这 8 个只读位是 ADC 转换结果的位[11:4] ，寄存器的位 7 是 ADC 转换结果的最高位。

11.4.6.5. ADC 数据低位寄存器(*ADCRL*), 地址 = 0x38

位	初始值	读/写	描述
7 – 4	-	只读	这 4 个只读位是 ADC 转换结果的位 [3:0]。
3 – 0	-	-	保留。

11.5. 乘法器

芯片内置一 8x8 乘法器以加强硬件的运算功能。这个乘法运算方式是 8x8 的无符号运算并且在一个时钟周期内完成运算。在下达指令之前，乘数与被乘数都要放在 *ACC* 累加器和 *MULOP* (0x08)寄存器上，在下达 *mul* 指令之后，运算结果的高位字节会放在寄存器 *MULRH* (0x09)上，运算结果的低位字节会放在 *ACC* 累加器上。乘法器的硬件框图如图 31 所示。

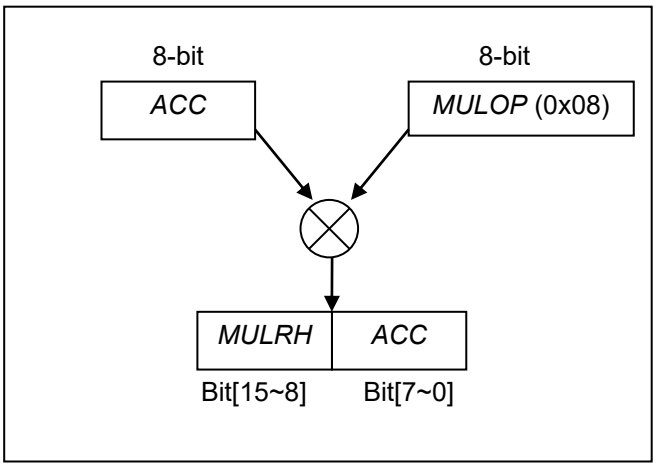


图 31：硬件乘法器框图

11.5.1. 乘法器运算对象寄存器(*MULOP*), 地址 = 0x08

位	初始值	读/写	描述
7 – 0	-	读/写	硬件乘法运算的运算对象。

11.5.2. 乘法器结果高字节寄存器(*MULRH*), 地址 = 0x09

位	初始值	读/写	描述
7 – 0	-	只读	乘法运算的高字节结果（只读）。

12. 仿真注意事项

建议使用 PDK5S-I-S01/2(B)对 PFC232 进行仿真。仿真时请注意以下事项:

(1) PDK5S-I-S01/2(B)为单核仿真器，仅支持主核（FPPA0）的仿真，不支持多核功能

(2) 用 PDK5S-I-S01/2(B)仿真时，不支持 *nadd*，*comp* 指令

(3) 用 PDK5S-I-S01/2(B)仿真时，不支持系统时钟 SYSCLK = ILRC/16

(4) 用 PDK5S-I-S01/2(B)仿真时，不支持 *TM2C.GPCRS* 和 *TM3C.GPCRS*

(5) 用 PDK5S-I-S01/2(B)仿真时，不支持 OPA

(6) 用 PDK5S-I-S01/2(B)仿真时，不支持 *PAPL/PBPL*

(7) 用 PDK5S-I-S01/2(B)仿真时，不支持 *GPCC*.P_PA0

(8) 用 PDK5S-I-S01/2(B)仿真时，不支持通过 *MISC*.4 动态设置 LCD 驱动的 VDD/2，只能是固定 1 或 0

(9) 用 PDK5S-I-S01/2(B)仿真时，不支持 *PWMG2C*.PA5

(10) 用 PDK5S-I-S01/2(B)仿真时，不支持 *ADCRGC.BG_2V/BG_3V/BG_4V*，并且只能固定是 BG_1V2

(11) 用 PDK5S-I-S01/2(B)仿真时，不支持 PB1 作为 ADC 外部参考电压

(12) 用 PDK5S-I-S01/2(B)仿真时，不支持程序选项：PB4_PB7_Drive, GPC_P_In, OPA_PWM, GPC_PWM, PWM_Source, TMx_Source and TMx_bit

(13) PDK5S-I-S01/2(B)仿真器的 ILRC 频率与实际 IC 不同，且未经校准，其频率范围大约在 34K~38KHz。

(14) PDK5S-I-S01/2(B) 不支持 PB3 的 VDD/2 功能。

(15) 当 *GPCS* 选择 Output 到 PA0 输出时，PA3 输出功能会受影响

(16) 当 *ADCRGC* 使用 PB1 时，PA1 必须浮空

(17) 用 PDK5S-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。

(18) 快速唤醒时间有差异，PDK5S-I-S01/2(B): 128 SYSCLK, PFC232: 45 ILRC

(19) IC 的看门狗溢出时间和使用 PDK5S-I-S01/2(B)仿真不同，如下：

WDT 溢出时间	PDK5S-I-S01/2(B)	PFC232
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}

13. 烧录方法

请使用 PDK-5S-P-003 进行烧录。PDK3S-P-002 或之前的烧录器皆不支持烧录 PFC232。

Jumper 连接：可依照烧录器软件上的说明，连接 jumper 即可。

请用户依据实际情况选择以下两种烧录模式。

13.1. 普通烧录模式

适用范围：

- 单独封装 IC，并在烧录器的 IC 插座或连接分选机烧录。
- 合封（MCP）IC，但与 PFC232 合封的 IC 及组件不会被以下电压破坏，也不会钳制以下电压的产生。

普通烧录模式电压条件：

(1) VDD 等于 7.5V，而最大供给电流最高可达约 20mA。

(2) PA5 等于 5.5V。

(3) 其他烧录引脚（GND 除外）等于 VDD。

重要提示：

- **如在 handler 上对 IC 进行烧录，请务必按照 APN004 及 APN011 的指示进行。**
- **为对抗烧录时的杂讯干扰，请于烧录时在分选机连接 IC 连接器一端的 VDD 和 GND 之间连接 0.01uF 电容。但切忌连接标值 0.01uF 以上的电容，以免影响普通烧录模式的运行。**

13.2. 限压烧录模式

适用范围：

- 在板烧录（On-board Writing），但其周边电路及组件不会被以下电压破坏，也不会钳制以下电压的产生。请参考在板烧录章节的详细说明。
- 合封（MCP）IC，但与 PFC232 合封的 IC 及组件不会被以下电压破坏，也不会钳制以下电压的产生。

限压烧录模式电压条件：

(1) VDD 等于 5.0V，而最大供给电流最高可达约 20mA。

(2) PA5 等于 5.0V

(3) 其他烧录引脚（GND 除外）等于 VDD。

若要启动限压烧录模式，请于烧录器界面上选择“MTP On-board VDD limitation” 或 “On-board Program”（请参考烧录器 PDK-5S-P-003 的用户手册）。

13.3. 在板烧录（On-Board Writing）

PFC232 可以支持在板烧录。所谓在板烧录，是指 IC 及其他周边电路及组件，皆已经焊接到 PCB 上，并对 IC 进行烧录的情况。在板烧录需要使用 PDK-5S-P-003 上五根引线：ICPCK、ICPDA、VDD、GND 和 ICVPP，用于与 IC 上的 PA3、PA6、VDD、GND 和 PA5 对应相连。

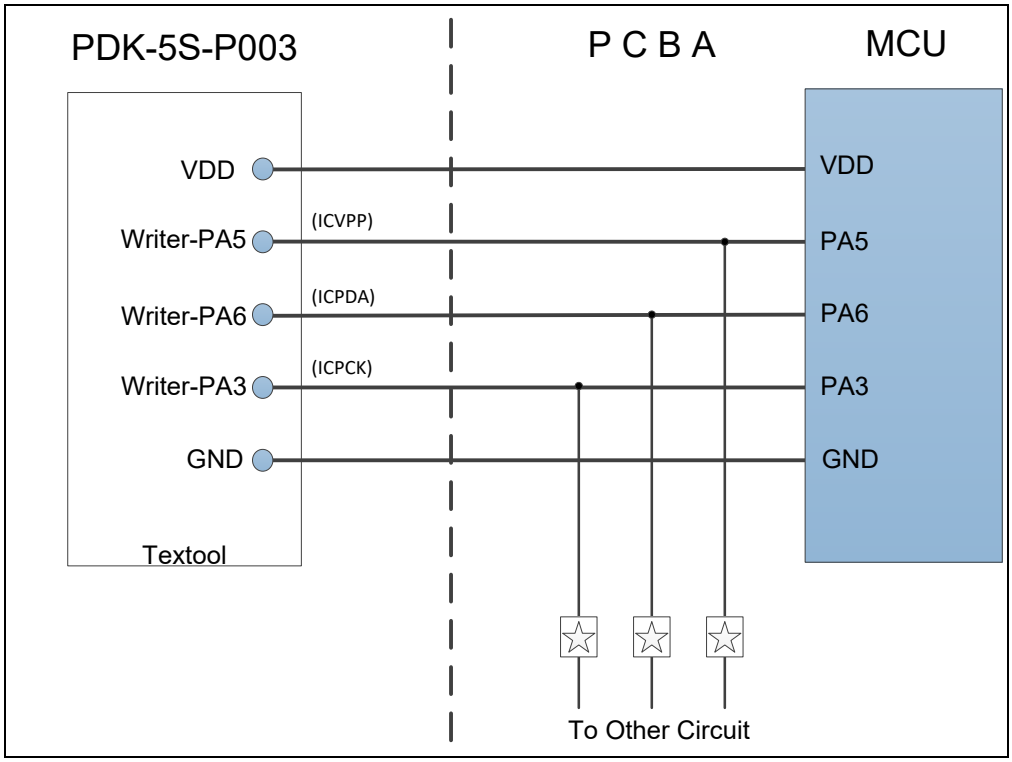


图 32：在板烧录接线示意图

图 32 中的 ☆ 为电阻或电容，用于隔离烧录引线和其他电路。电阻应≧10KΩ，电容应≦220pF。

注意：

- 一般来说，在板烧录应使用限压烧录模式。请参考在限压烧录模式的详细说明。
- PCB 上的 VDD 与 GND 之间**不可**连接有 5.0V 或以下的稳压二极管或其他钳制 5.0V 产生的电路或组件。
- PCB 上的 VDD 与 GND 之间**不可**连接有标值 500uF 或以上的电容器。
- 一般来说，用于烧录讯号的 PA3，PA5 及 PA6 引脚，**不能**作为强输。

14. 直流交流电气特性

14.1. 绝对最大值

名称	最小值	典型值	最大值	单位	备 注
电源电压 (VDD)	2.2		5.5	V	**电源电压最大不能超过 5.5V，否则可能损坏 IC**
输入电压	-0.3		V_{DD} + 0.3	V	
工作温度	-40		85	°C	
储藏温度	-50		125	°C	
节点温度		150		°C	

14.2. 器件电气特性

下列所有数据除特别列明外，皆于 V_{DD}=5.0V，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	2.2#		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	 60K	 8M 4M 2M	Hz	 V_{DD}≧3.75V V_{DD}≧2.5V V_{DD}≧2.2V V_{DD} = 5V
P_{cycle}	烧录次数	1000			cycles	
I_{OP}	工作电流		1 65		mA uA	f_{SYS}=IHRC/16=1MIPS@5V f_{SYS}=ILRC=40KHz@5V
I_{PD}	掉电模式消耗电流 （用 *stopsys* 命令）		1		uA	f_{SYS}= 0Hz,V_{DD}=5V
I_{PS}	省电模式消耗电流 （用 *stopexe* 命令）		3		uA	V_{DD} =5V; f_{SYS}= ILRC 仅启用 ILRC 模块
V_{IL}	输入低电压	0		0.1V_{DD}	V	
V_{IH}	输入高电压	0.7 V_{DD}		V_{DD}	V	
I_{OL}	IO 输出灌电流 PA0, PA3, PA4, PB2 PA5, PA6, PA7, PB0, PB1, PB3 , PB5, PB6 PB4, PB7 (Strong) PB4, PB7 (Normal)		 23 17 23 37 23		mA	V_{DD}=5V, V_{OL}=0.5V
I_{OH}	IO 输出驱动电流 PB4, PB7 (Strong) PB4, PB7 (Normal) Others IO		 -26 -10 -10		mA	V_{DD}=5V, V_{OH}=4.5V

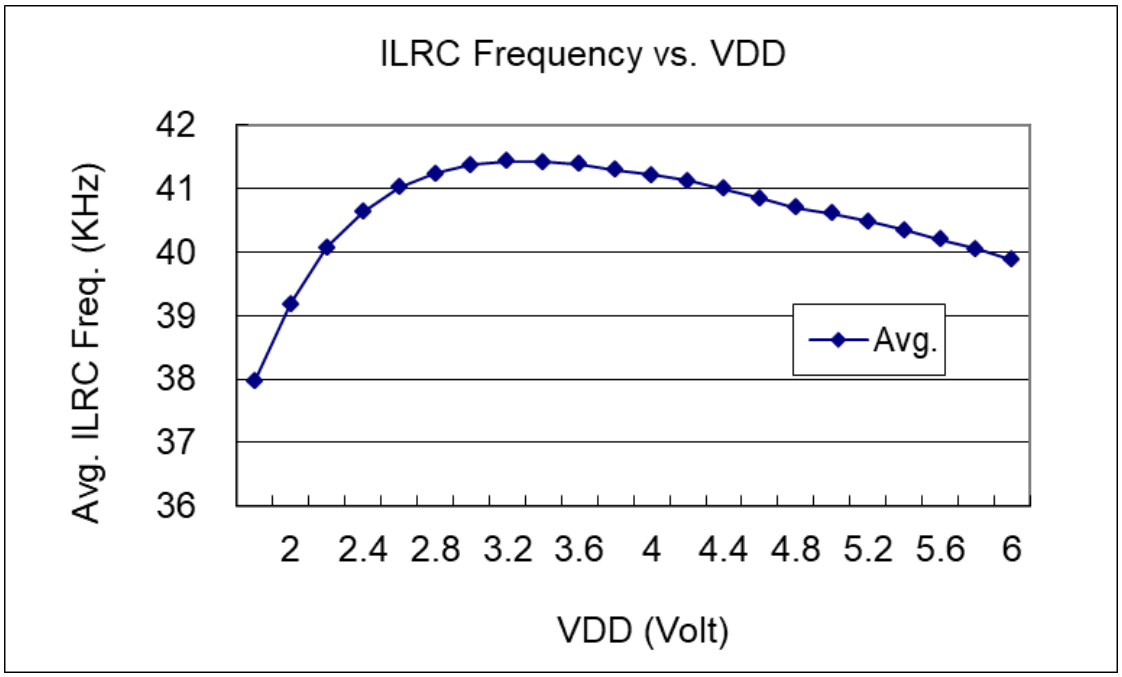
符 号	特 性	最小值	典型值	最大值	单位	条 件
V_{IN}	输入电压	-0.3		V_{DD}+0.3	V	
$I_{INJ\ (PIN)}$	脚位的引入电流		1		uA	V_{DD}+0.3≧V_{IN}≧ -0.3
R_{PH}	上拉电阻		82		KΩ	V_{DD}=5.0V
R_{PL}	下拉电阻		82		KΩ	V_{DD}=5.0V
V_{BG}	Bandgap 参考电压	1.145*	1.20*	1.255*	V	V_{DD} =2.2V ~ 5.5V -40°C < Ta < 85°C*
f_{IHRC}	IHRC 输出频率（校准后） *	15.84*	16*	16.16*	MHz	V_{DD} =5V, Ta=25°C
		15.20*	16*	16.80*		V_{DD} =2.2V~5.5V, -40°C <Ta<85°C*
t_{INT}	中断脉冲宽度	30			ns	V_{DD}=5V
V_{AD}	AD 输入电压	0		V_{DD}	V	
ADrs	ADC 分辨率		12		bit	
ADcs	ADC 消耗电流		0.8 0.75		mA	@5V @3V
ADclk	ADC 时钟周期		2		us	2.2V ~ 5.5V
t_{ADCONV}	ADC 转换时间 （T_{ADCLK} 是选定 AD 转换时钟周期）		16		T_{ADCLK}	12-bit resolution
AD DNL	ADC 微分非线性		±2*		LSB	
AD INL	ADC 积分非线性		±4*		LSB	
ADos	ADC 失调电压*		2		mV	V_{DD} =3V
V_{REFH}	ADC 参考高电压 4V 3V 2V	 3.90 2.93 1.95	 4 3 2	 4.10 3.07 2.05		V_{DD} =5V, 25 °C
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		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}	快速唤醒时钟周期		45		T_{ILRC}	T_{ILRC} 是 ILRC 时钟周期
	普通唤醒时钟周期		3000			
t_{SBP}	系统上电开机时间		72		ms	V_{DD}=5V
t_{RST}	外部复位脉冲宽度	120			us	V_{DD}=5V

符 号	特 性	最小值	典型值	最大值	单位	条 件
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
OPAcm	OPA 共模输入电压*	0		VDD -1.3	V	
OPAos	OPA 偏压*		±10		mV	V_{DD} =5V
I_{OPA}	OPA 输出电流*	200			uA	
OPAgain	OPA 直流增益*		80		dB	

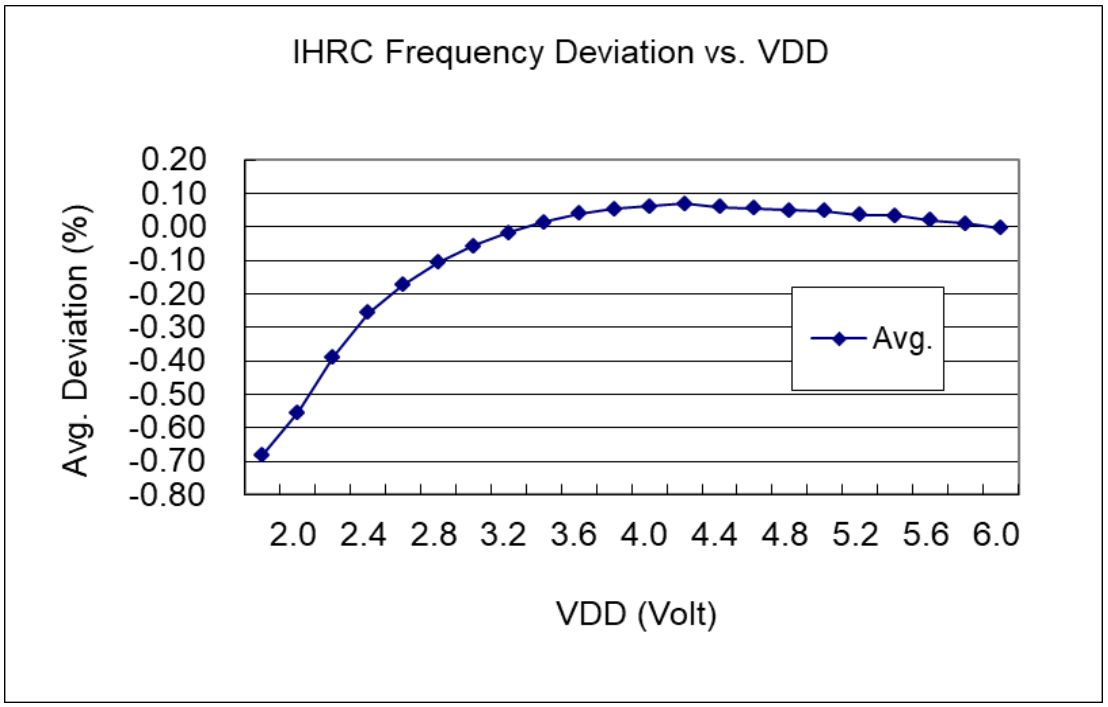
*这些参数是设计参考值，并不是每个芯片测试。

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。

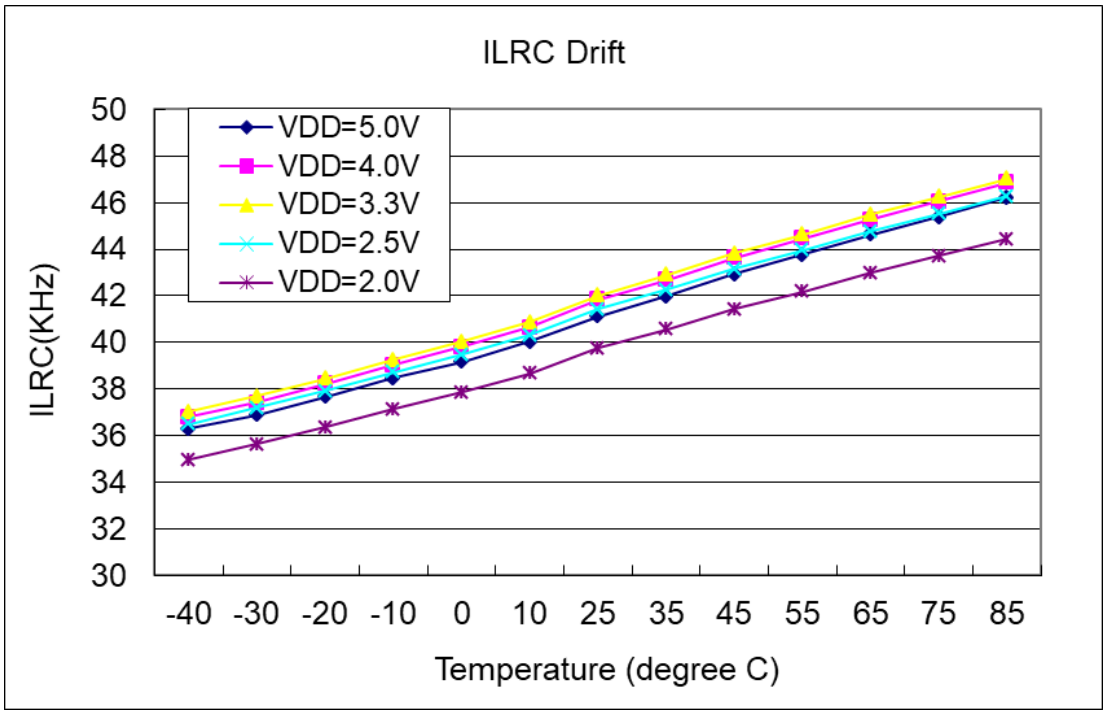
14.3. ILRC 频率与 VDD 关系曲线图



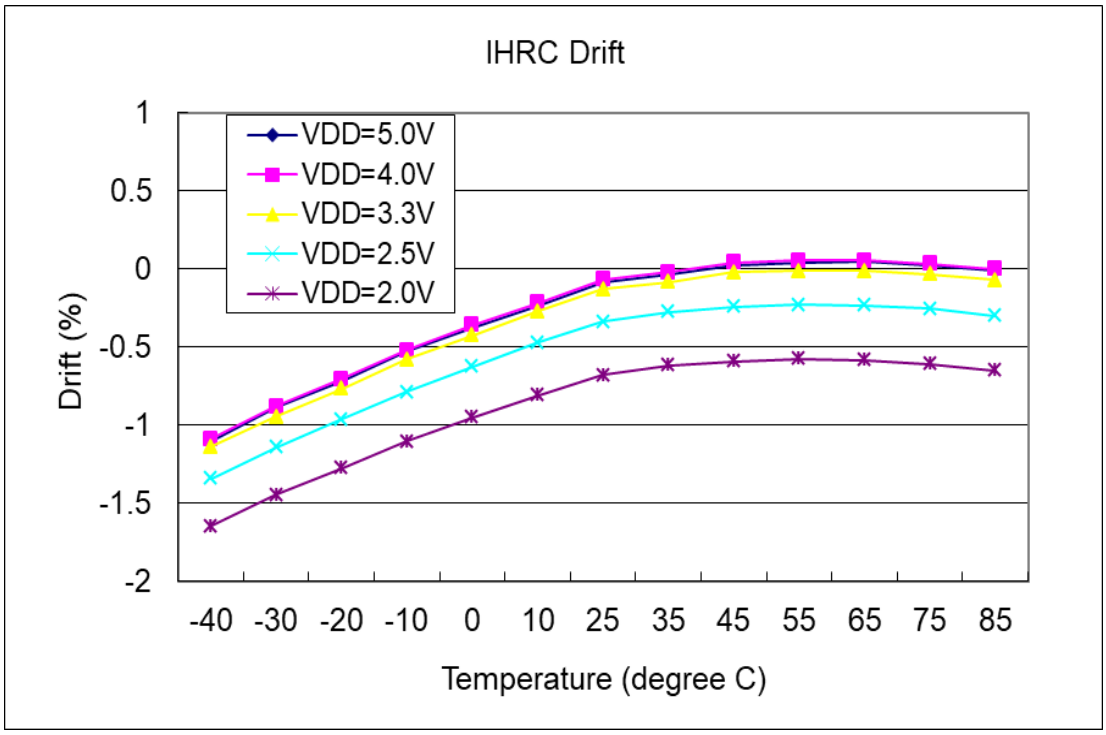
14.4. IHRC 频率与 VDD 关系曲线图（校准到 16MHz）



14.5. ILRC 频率与温度关系曲线图



14.6. IHRC 频率与温度关系曲线图（校准到 16MHz）

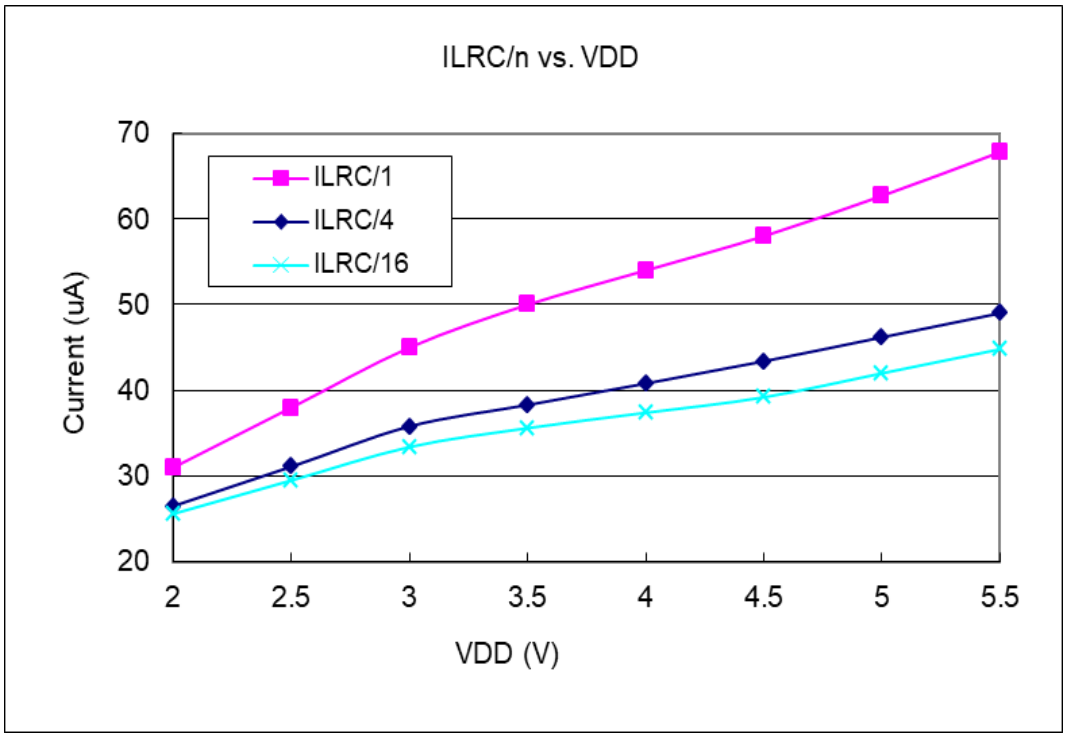


14.7. 工作电流与 VDD、系统时钟 CLK=ILRC/n 曲线图

条件：2-FPPA (FPPA0: tog PA0, FPPA1: idle)

开启的硬件模块：ILRC，Bandgap，LVR；**关闭**的硬件模块：IHRC, EOSC, T16, TM2, TM3, ADC 模块；

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

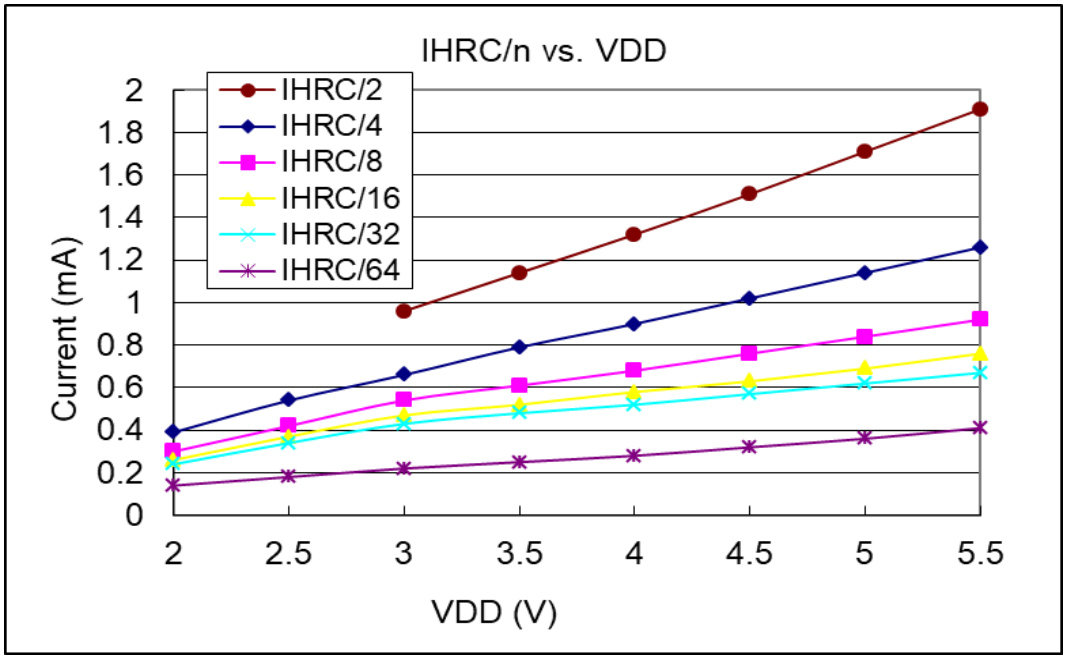


14.8. 工作电流与 VDD、系统时钟 CLK=IHRC/n 曲线图

条件：2-FPPA (FPPA0: tog PA0, FPPA1: idle)

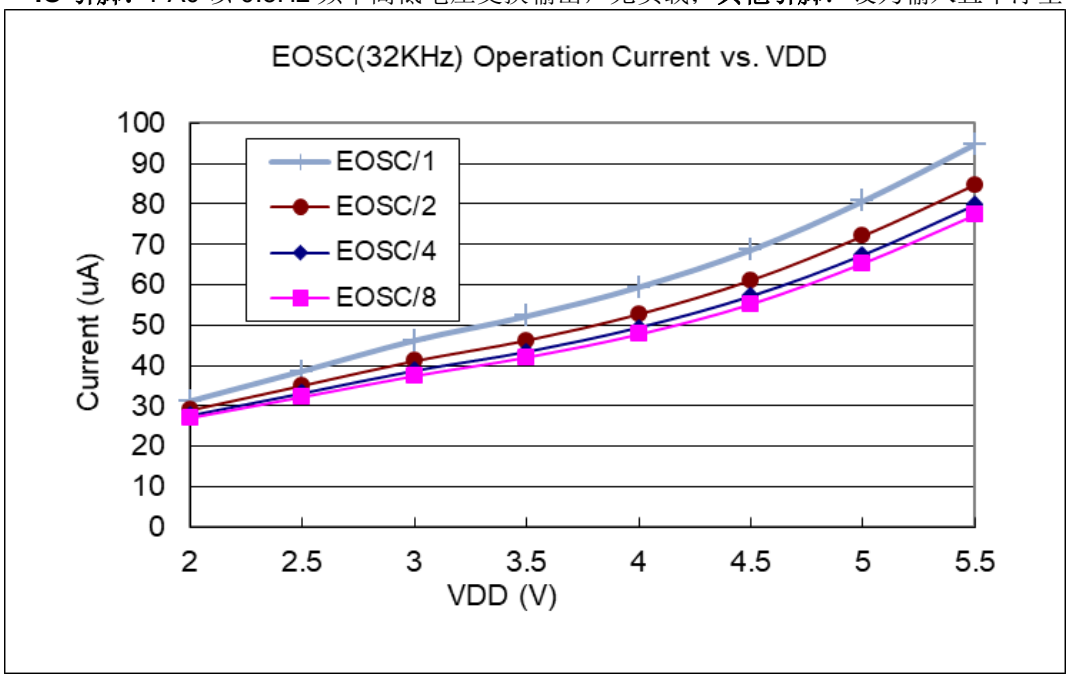
开启的硬件模块：IHRC，Bandgap，LVR；**关闭**的硬件模块：ILRC, EOSC, LVR, T16, TM2, TM3, ADC 模块；

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



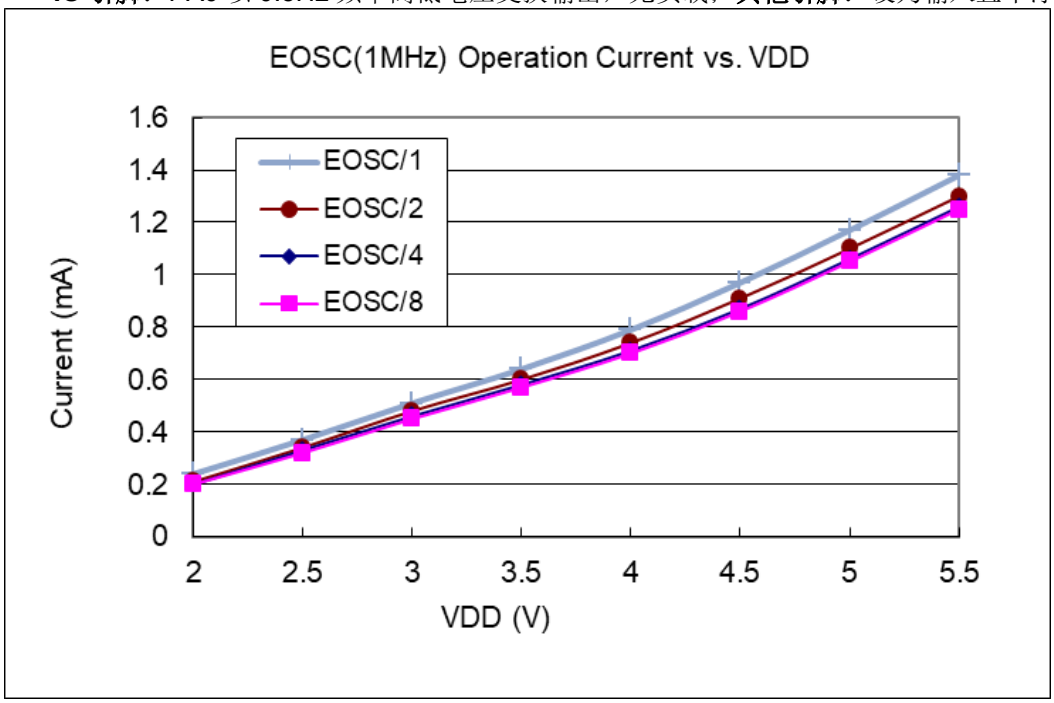
14.9. 工作电流与 VDD、系统时钟 CLK=32KHz EOSC/n 曲线图

条件：**开启**的硬件模块：EOSC， MISC.6 = 1，Bandgap，LVR；
关闭的硬件模块：IHRC, ILRC, T16, TM2, TM3, ADC 模块；
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



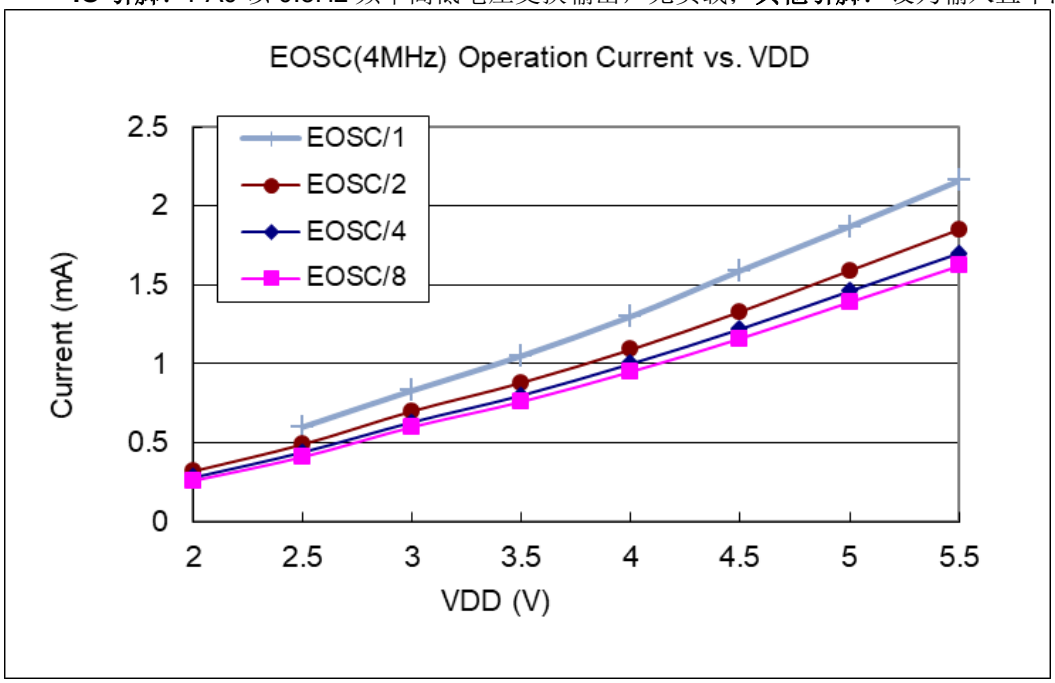
14.10. 工作电流与 VDD、系统时钟 CLK=1MHz EOSC/n 曲线图

条件：**开启**的硬件模块：EOSC，MISC.6 = 1，Bandgap，LVR；
关闭的硬件模块：IHRC, ILRC, T16, TM2, TM3, ADC 模块；
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



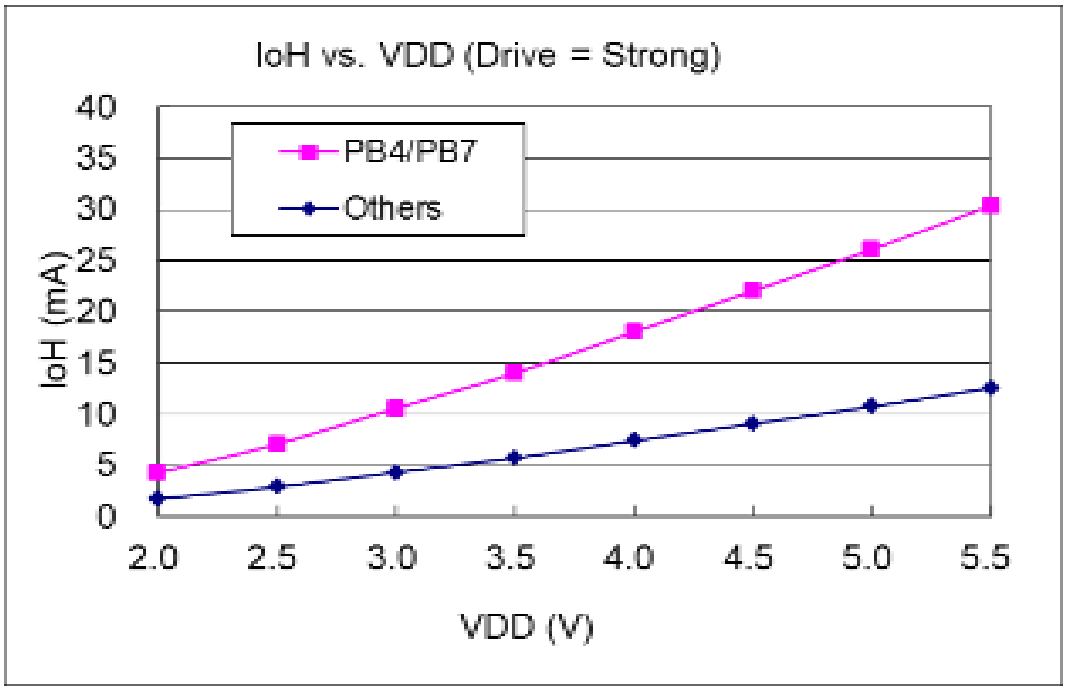
14.11. 工作电流与 VDD、系统时钟 CLK=4MHz EOSC/n 曲线图

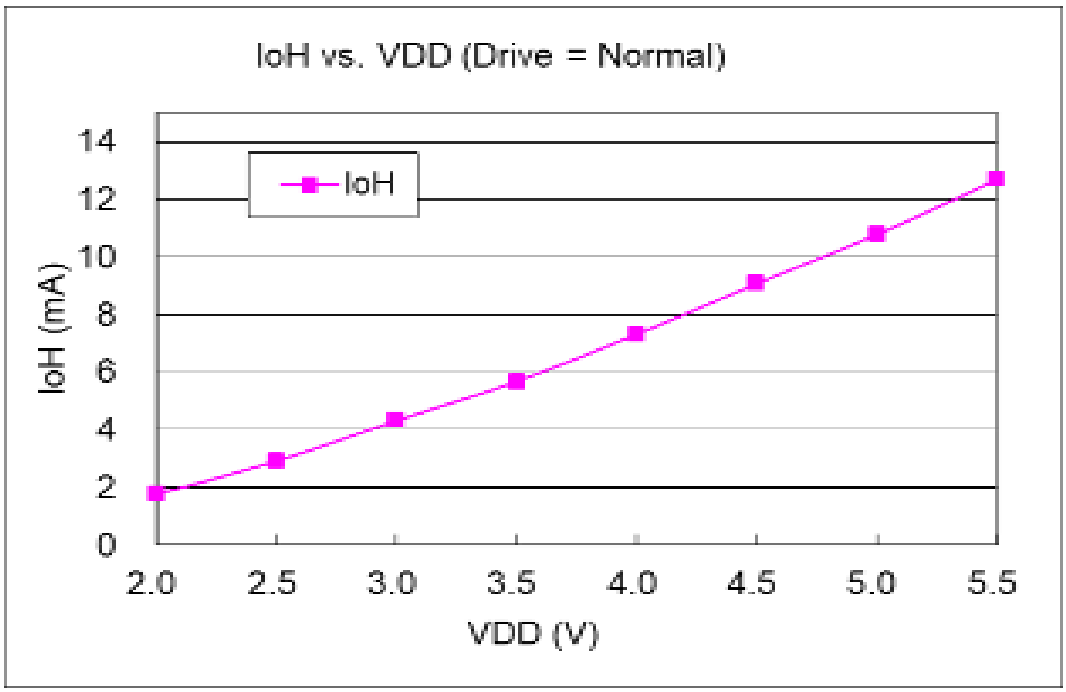
条件：**开启**的硬件模块：EOSC，MISC.6 = 1，Bandgap，LVR；
关闭的硬件模块：IHRC, ILRC, T16, TM2, TM3, ADC 模块；
IO 引脚：PA0 以 0.5Hz 频率高低电压交换输出，无负载；**其他引脚**：设为输入且不浮空

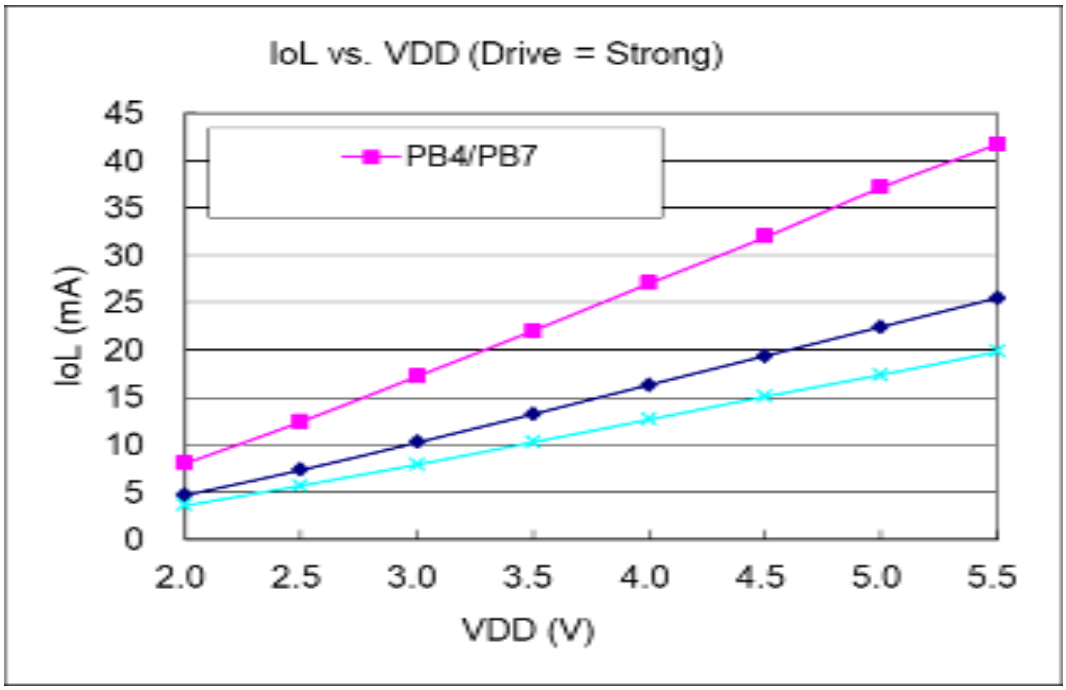


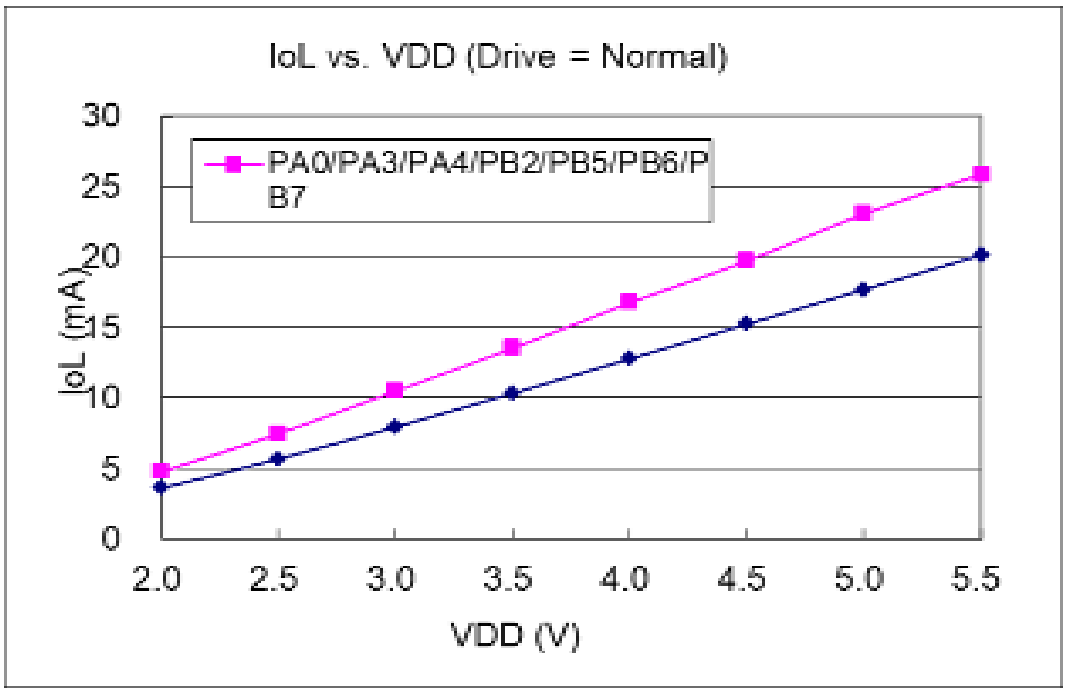
14.12. 引脚输出驱电流(I_{OH})与灌电流(I_{OL}) 曲线图

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

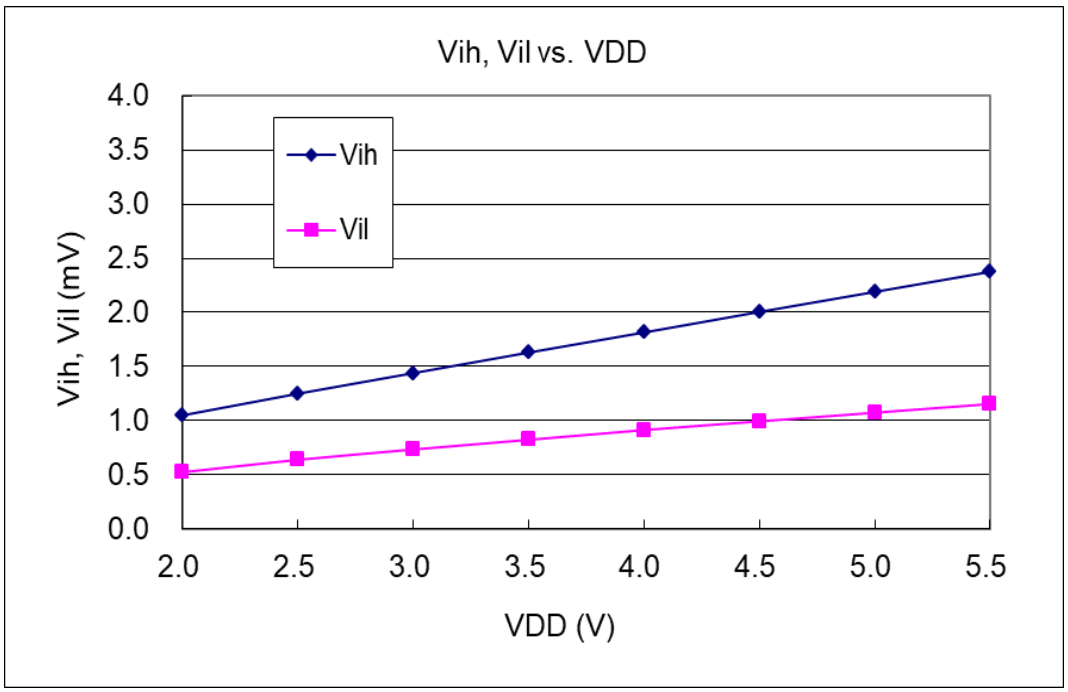



IoH vs. VDD (Drive = Normal)
IoH
IoH (mA)
14
12
10
8
6
4
2
0
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
VDD (V)

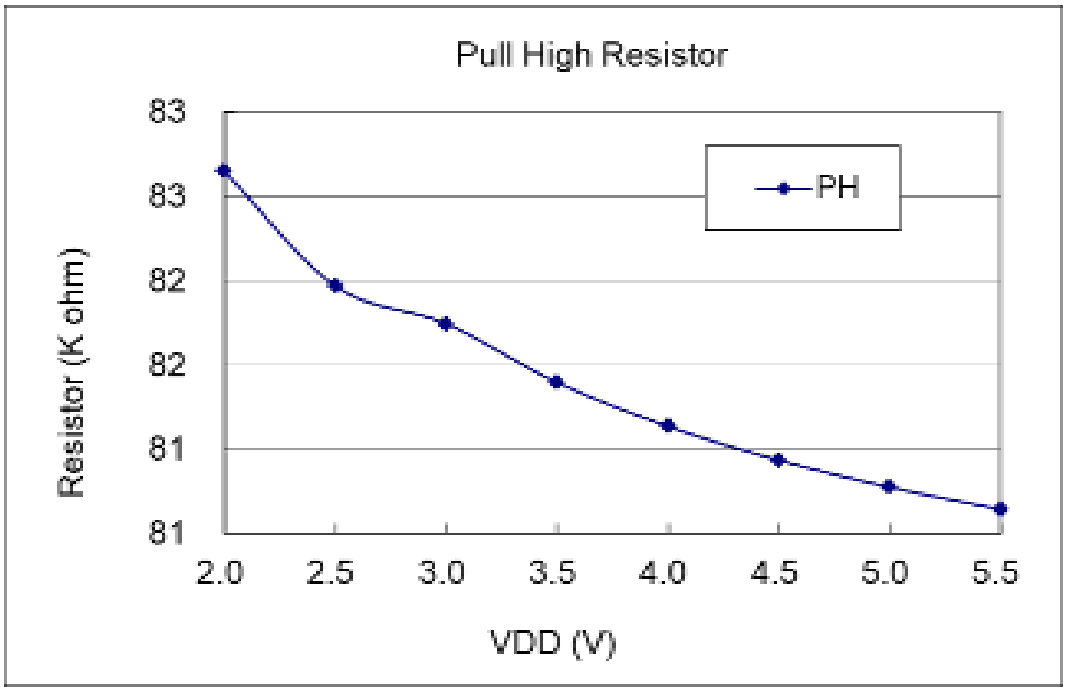

IoL vs. VDD (Drive = Strong)
PB4/PB7
IoL (mA)
45
40
35
30
25
20
15
10
5
0
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
VDD (V)

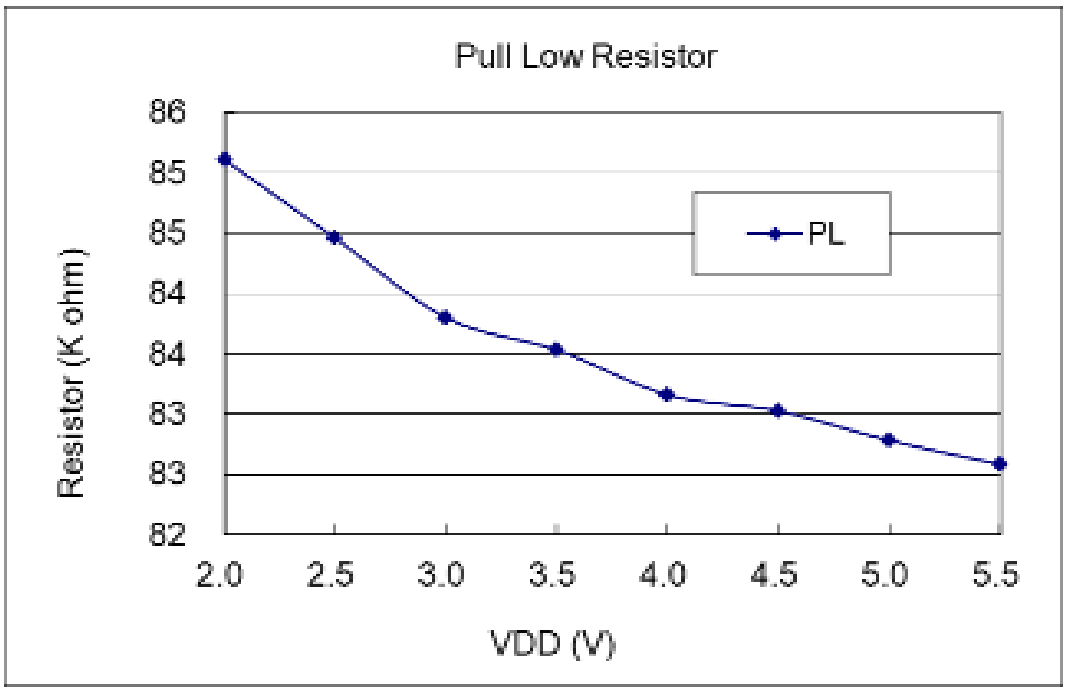


14.13. 引脚输入高电压与低电压(V_{IH}/V_{IL}) 曲线图

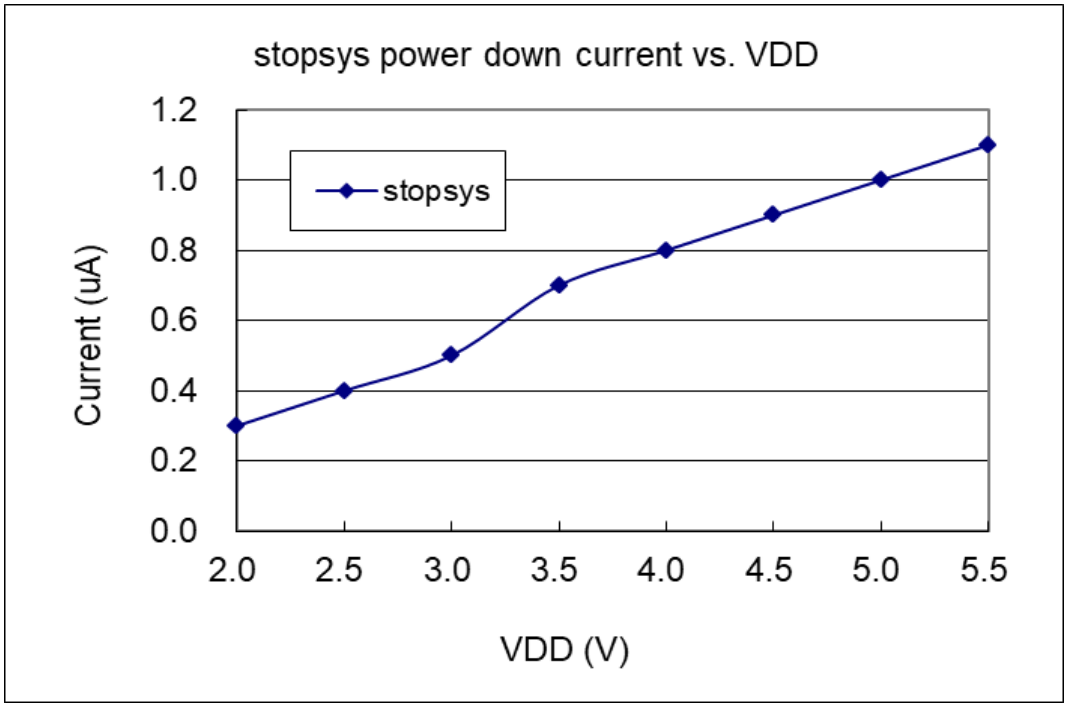


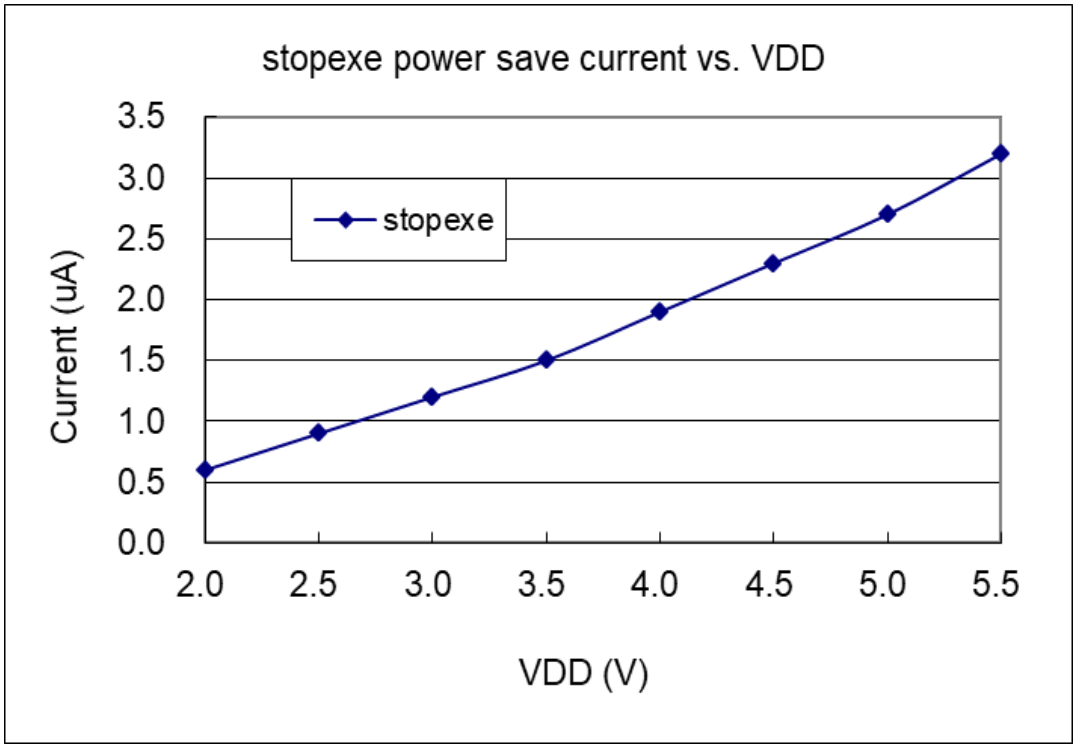
14.14. 引脚上拉/下拉电阻曲线图





14.15. 掉电电流(I_{PD})与省电电流(I_{PS}) 曲线图





15. 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

15.1. 指令表

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』

nmov ***M, a***	取累加器的负逻辑（2 补数）并复制到存储器。
	例如： nmov MEM, a;
	结果： MEM ← a 的 2 补码
	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, 0xf5 ; // ACC=0xf5`
	`nmov ram9, a; // ram9=0x0b，ACC=0xf5`

nmov ***a, M***	取存储器的负逻辑（2 补数）并复制到累加器。
	例如： nmov a, MEM ;
	结果： a ←MEM 的 2 补码; 当 MEM 的 2 补码为零时，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』
	应用范例：
	`mov a, 0xf5 ;`
	`mov ram9, a ; // ram9=0xf5`
	`nmov a, ram9 ; // ram9=0xf5, ACC=0x0b`

ldtabh index	使用索引作为 OTP 的地址将 OTP 程序存储器的高字节数据读取并载入到累加器。需要 2T 时间执行这一指令。
	例如： ldtabh index;
	结果： a ← {bit 15~8 of OTP [index]};
	受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
	应用范例：
	`word ROMptr ; // 在 RAM 定义 OTP 的指针`
	`…`

```
    mov         a, la@TableA ;      // 指定 OTP TableA 指针 （LSB）
    mov         lb@ROMptr, a ;      // 将指针存到 RAM（LSB）
    mov         a, ha@TableA ;      // 指定 OTP TableA 指针(MSB)
    mov         hb@ROMptr, a ;      // 将指针存到 RAM (MSB)
    …
    ldtabh      ROMptr ;            // 读取数据并载入到累加器 (ACC=0X02)
    ….
TableA :    dc      0x0234, 0x0042, 0x0024, 0x0018 ;
```

---

ldtabl index

使用索引作为 OTP 的地址并将 OTP 程序存储器的低字节数据读取并载入到累加器。需要 2T 时间执行这一指令。

例如： ldtabl index;

结果： a ← {bit7~0 of OTP [index]};

受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

应用范例：

```
    word        ROMptr ;            // 在 RAM 定义 OTP 的指针
    …
    mov         a, la@TableA ;      // 指定 OTP TableA 指针（LSB）
    mov         lb@ROMptr, a ;      // 将指针存到 RAM （LSB）
    mov         a, ha@TableA ;      // 指定 OTP TableA 指针 （MSB）
    mov         hb@ROMptr, a ;      // 将指针存到 RAM （MSB）
    …
    ldtabl      ROMptr ;            // 读取数据并载入到累加器 （ACC=0x34）
    ….
TableA :    dc      0x0234, 0x0042, 0x0024, 0x0018 ;
```

---

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
	…
	--- 	

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』

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。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），
	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。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）
	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
	--- 	

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』

15.2. 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 的 2 补码 + 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 的 2 补码 + 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』

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

15.4. 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 **a, IO**	累加器和 IO 寄存器执行逻辑 XOR，然后把结果保存到累加器
	例如： ***xor a, pa ;***
	结果： a ← a ^ pa ; // pa 是 A 端口的数据寄存器
	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，然后把结果保存到 IO 寄存器
	例如： ***xor pa, a ;***
	结果： pa ← a ^ pa ; // pa is the data register of port A
	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
	neg mem ; // mem = 0xC8
	--- 	

comp **a, I**	累加器和立即数据比较运算，影响的是标志，标志的改变与（a - I）运算相同 例如： *comp* a, 0x55; 结果： 标志的改变与 （a - 0x55）运算相同 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 ;
	comp a, 0x38 ; // Z 标志位被设置为 1
	comp a, 0x42 ; // C 标志位被设置为 1
	comp a, 0x24 ; // C, Z 标志位被清除为 0
	comp a, 0x6a ; // C, AC 标志位被设置为 1
	--- 	

comp **a, M**	累加器和 RAM 比较运算，影响的是标志位，标志位的改变与（a - MEM）运算相同
	例如： *comp* a, MEM;
	结果：标志位的改变与（a - MEM）运算相同
	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 标志位被设置为 1
	mov a, 0x42 ;
	mov mem, a ;
	mov a, 0x38 ;
	comp a, mem ; // C 标志位被设置为 1
	--- 	

comp **M, a**	累加器和 RAM 比较运算，影响的是标志位，标志位的改变与（MEM - a）运算相同 例如： *comp* MEM, a; 结果：标志位的改变与（MEM - a）运算相同 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』

15.5. 位运算类指令

set0 **IO.n**	IO 的位 N 设为 0 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 的位 N 设为 1 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』
tog **IO.n**	IO 的位 N 改变为相反状态 例如： *tog* pa.5 ; 结果： PA5=>1 假如 PA5=0 ; PA5=>0 假如 PA5=1 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
set0 **M.n**	RAM 的位 N 设为 0 例如：*set0* MEM.5 ; 结果： MEM 位 5 为 0 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

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

swapc **IO.n**	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
	…
	--- 	

15.6. 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』

ceqsn **M, a**	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 (M ← M - a)相同 例如： *ceqsn* MEM, a; 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。 Example: *t1sn* pa.5 ; 结果：如果 PA5 是 1，跳过下一个指令。 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
t0sn **M.n**	如果 RAM 的指定位是 0，跳过下一个指令。 例如： *t0sn* MEM.5 ; 结果： 如果 MEM 的位 5 是 0，跳过下一个指令。 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
t1sn **M.n**	如果 RAM 的指定位是 1，跳过下一个指令。 例如： *t1sn* MEM.5 ; 结果： 如果 MEM 的位 5 是 1，跳过下一个指令。 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
izsn **a**	累加器加 1，若累加器新值是 0，跳过下一个指令。 例如： *izsn* a; 结果： a ← a + 1，若 a=0，跳过下一个指令。 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
dzsn **a**	累加器减 1，若累加器新值是 0，跳过下一个指令。 例如： *dzsn* a; 结果： a ← a – 1，若 a=0，跳过下一个指令。 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
izsn **M**	RAM 加 1，若 RAM 新值是 0，跳过下一个指令。 例如： *izsn* MEM; 结果： MEM ← MEM + 1，若 MEM=0，跳过下一个指令。 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』

dzsn **M**	RAM 减 1，若 RAM 新值是 0，跳过下一个指令。 例如： *dzsn* MEM; 结果： MEM ← MEM - 1，若 MEM=0，跳过下一个指令。 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
wait0 **IO.n**	直到 IO 的 N 位为 0，才转到下一个指令；否则，在这里等候。

	例如： *wait0* pa.5; 结果： 等候 PA5=0 才转到下一个指令 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
wait1 **IO.n**	直到 IO 的 N 位为 1，才转到下一个指令；否则，在这里等候。 例如： *wait1* pa.5; 结果： 等候 PA5=0 才转到下一个指令 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

15.7. 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』
delay **a**	延迟（N+1）周期，N 是由累加器所指定，时间周期是根据执行此指令的 FPP 单元的 1 个指令周期。指令执行后，累加器将为零。 例如： *delay* a; 结果：假如 ACC=0fh，在此延迟 16 个周期 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』 注意：由于 ACC 是指令计数时的暂时缓冲区，请确保执行此指令时不会被中断。否则，延迟时间可能不是所预期的。
delay **I**	延迟（N+1）周期，N 是立即指定的数据，时间周期是根据执行此指令的 FPP 单元的 1 个指令周期。指令执行后，累加器将为零。 例如： *delay* 0x05; 结果： 在此延迟 6 个周期 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』 注意：由于 ACC 是指令计数时的暂时缓冲区，请确保执行此指令时不会被中断。否则，延迟时间可能不是所预期的。
delay **M**	延迟（N+1）周期，N 是由 RAM 所指定，时间周期是根据执行此指令的 FPP 单元的 1 个指令周期。指令执行后，累加器将为零。 例如： *delay* M; 结果：假如 M=ffh，在此延迟 256 个周期 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』 注意：由于 ACC 是指令计数时的暂时缓冲区，请确保执行此指令时不会被中断。否则，延迟时间可能不是所预期的。
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成为下一个程序计数器。
	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	复位看门狗定时器 例如： *wdreset* ; 结果： 复位看门狗定时器 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

15.8. 指令执行周期综述

指令	条件	单核心	双核心
goto, call		**2T**	1T
ceqsn, cneqsn, t0sn, t1sn, dzsn, izsn	判断条件成立	**2T**	1T
	判断条件不成立	1T	1T
ldtabh, ldtabl, idxm, pcadd, ret, reti		2T	2T
Others		1T	1T