PADAUK 應廣科技

LeapDragon（跃龙）

PFC887

工业级 – 8 位 MTP 型单片机 (FPPA™)

数据手册

第 0.02 版

2023 年 10 月 26 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。

目录

修订历史

修 订	日期	描 述
0.01	2023/01/16	1. 新增 10.4.5.15~10.4.5.20 2. 删除 13.2 节里的 V_{FSV}, V_{PDRV}, T_{POR}, $T_{FS}V$, T_{drop}, T_{LVR}, f_{LIRC} 3. 更新 13.2 节里的 f_{SYS}, I_{OP}, I_{PD}, I_{PS}, R_{PH} 4. 更新 13.3~13.12 节 5. 其他已知细节错误修正
0.02	2023/10/26	1. 新增 PFC887-Y24 2. 更新看门狗注意说明、图 TM2/3 3. 修改复位说明

警告

在使用 IC 前，请务必认真阅读 PFC887 相关的 APN（应用注意事项）。
APN 下载地址为：http://www.padauk.com.tw/cn/technical/index.aspx

1. 单片机特点

1.1. 系统特性

- 高抗干扰系列

  特别适用于 AC 电源供电的、阻容降压电路的、需要较强抗干扰能力的，或有高 EFT 安规测试要求(±4KV)的产品。
- 工作温度范围: -40°C ~ 85°C

1.2. 系统功能

- 5KW MTP 程序空间供 8 个 FPPA 单元使用（可编程 1000 次）
- 512 Bytes data 数据空间供八个 FPPA 单元使用
- 支持 MTP 系统烧录
- 两个硬件 16 位定时器
- 两个硬件 8 位定时器
- 一个 12 位硬件 PWM 生成器，可编程设定周期和占空比
- 支持 PWM 生成器触发 AD 转换
- 高达 16 通道 11 位 ADC，其中一个通道来自于内部 Bandgap 参考电压
- 一个 8 倍电压增益 OPA 用于侦测电流
- 硬件脉冲捕获
- 一个 16X8 硬件乘法器
- 一个 16X16 硬件乘法器
- 一个 16/16 硬件除法器
- 三个通用比较器
- 支持三相无刷直流马达
- 20 个 IO 引脚带有 14mA 或 7mA 两种驱动选择和上拉电阻
- 6 个输出引脚带有 14mA 或 7mA 两种驱动选择
- 6 级 VDD 复位电压(LVR)可选：4.5V、4.0V、3.75V、3.5V、3.3V、3.15V
- 14 级 VDD 检测电压(LVD)可选：4.5V、4.0V、3.75V、3.5V、3.3V、3.15V、3.0V、2.7V、2.5V、2.4V、2.3V、2.2V、2.1V、2.0V
- 四个可选择的外部中断引脚：PA0 或 PA5、PB0 或 PB7
- 每个引脚都可配置唤醒功能
- 工作电压范围：3.15V ~ 6V

1.3. 高性能 RISC CPU 架构

- 现场可编程处理器陈列(FPPA™)技术专利
- 工作模式：八个 FPPA™处理单元运作模式
- 106 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)
- 提供可保护 MTP 资料的加密功能
- 寄存器空间、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

- PFC887-T28: TSSOP28 (173mil)
- PFC887-Y24: SSOP24
  - 有关封装尺寸的信息，请参阅官方网站文件：“封装信息”

2. 系统概述和方框图

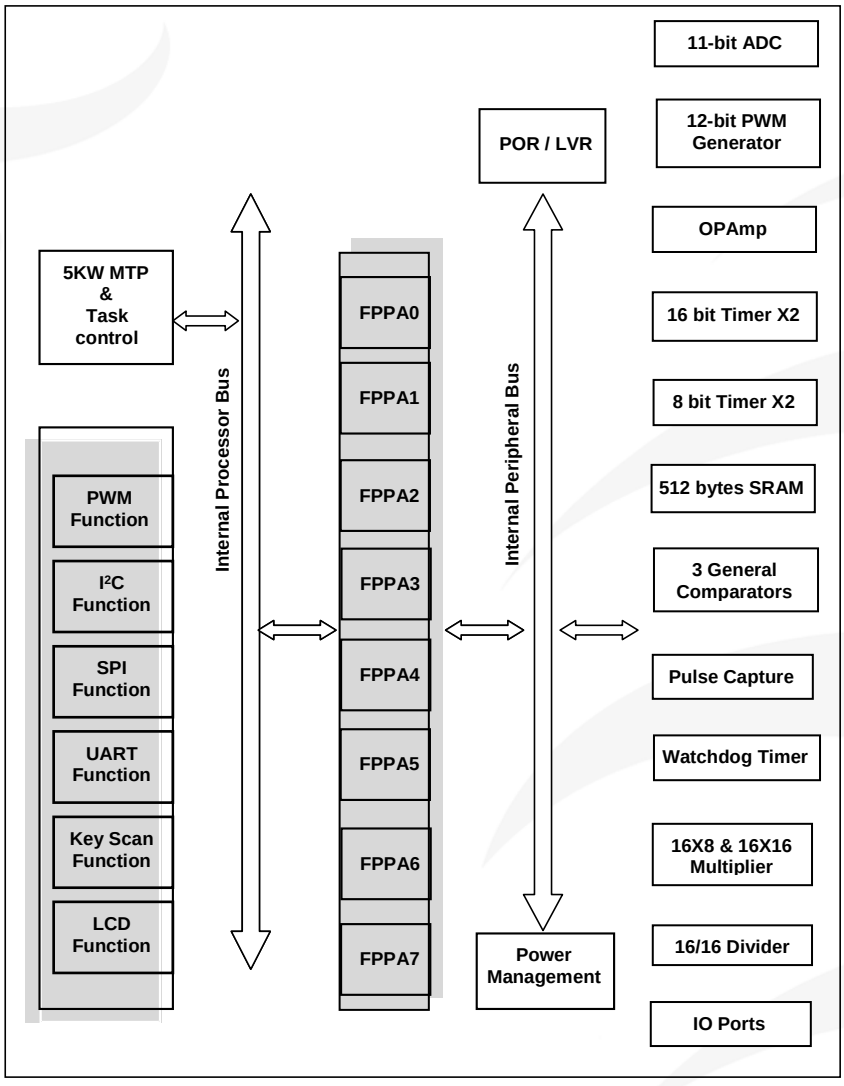
PFC887 是一个带 ADC、并行处理、完全静态，以 MTP 为程序存储基础的处理器，此处理器具有八个处理单元。它基于 RISC 架构基础，获得（Field Programmable Processor Array 现场可编程处理器阵列）技术专利。大部分指令的执行周期都是一个指令周期，只有少部分间接寻址的指令需要两个指令周期。

PFC887 内置 5KW MTP 程序存储器以及 512 字节数据存储器，供八个 FPPA 单元工作使用。

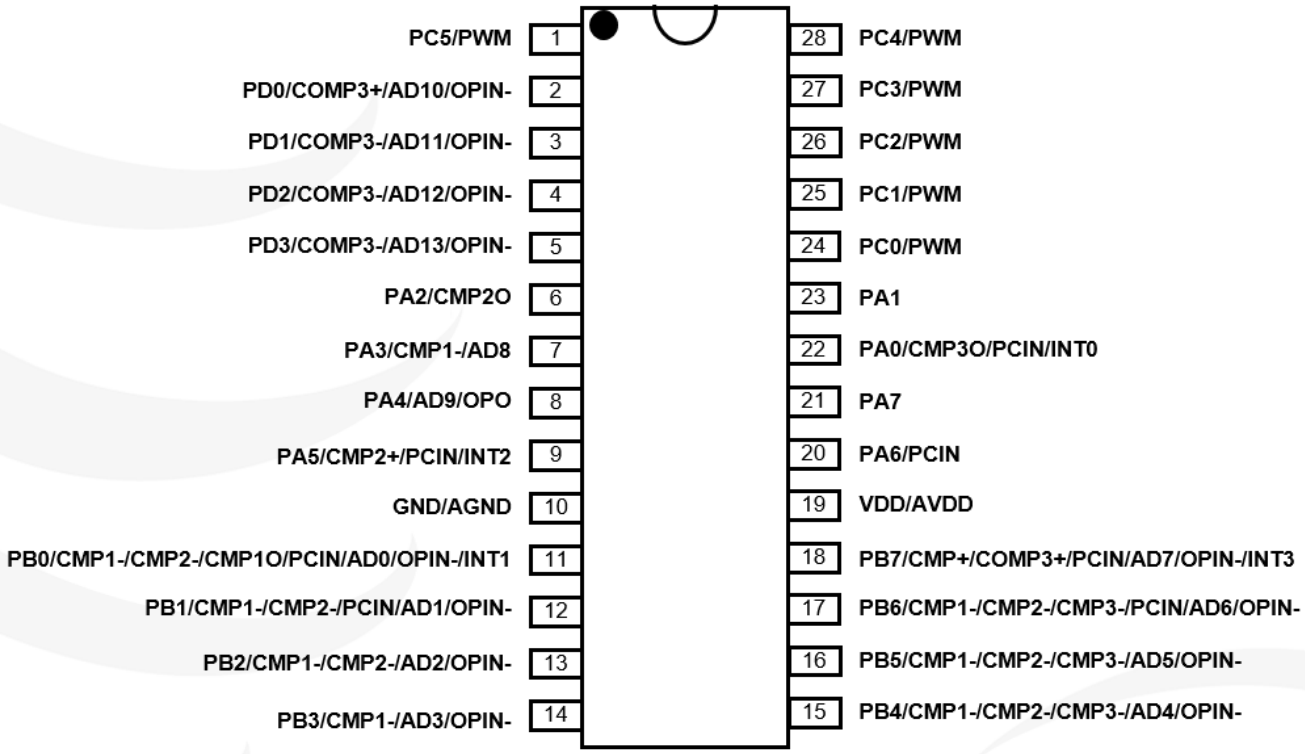
PFC887 内置 11 通道 11 位分辨率 A/D 转换器，其中一通道为内部 Bandgap 参考电压或 0.25*VDD。

PFC887 提供一个 8 倍电压增益 OPA、三个通用比较器、四个硬件计数器。计数器分别为两个 16 位计数器和两个 8 位计数器。

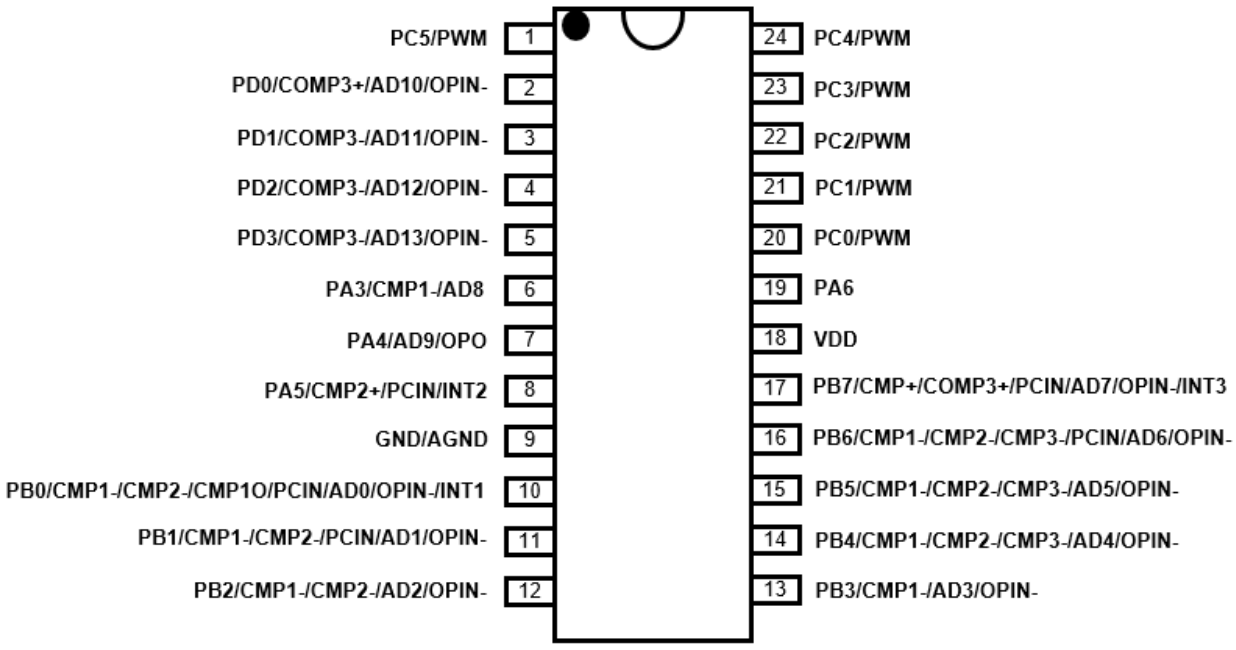
PFC887 内置一个硬件脉冲捕获、12 位硬件 PWM 生成器和 PWM 保护模块，为无刷直流控制器提供最佳的处理方案



3. 引脚分配及功能说明



PFC887-T28: TSSOP28(173mil)



PFC887-Y24

引脚说明:

引脚名称	输入/ 输出			特殊功能								
	I / O	上拉	唤醒	XIN XOUT	比较器	PWM	脉冲捕获	ADC	OPA	外部中断	外部复位	烧录
PA0	V	V	V		CMP3O		PCIN			INT0		
PA1	V	V	V									
PA2	V	V	V		CMP2O							
PA3	V	V	V		CMP1-			AD8				√
PA4	V	V	V					AD9	OPO			√
PA5	V	V	V		CMP2+		PCIN			INT2	√	√
PA6	V	V	V	V								√
PA7	V	V	V	V								
PB0	V	V	V		CMP1- CMP2- CMP1O		PCIN	AD0	OPIN-	INT1		
PB1	V	V	V		CMP1- CMP2-		PCIN	AD1	OPIN-			
PB2	V	V	V		CMP1- CMP2-			AD2	OPIN-			
PB3	V	V	V		CMP1-			AD3	OPIN-			
PB4	V	V	V		CMP1- CMP2- CMP3-			AD4	OPIN-			
PB5	V	V	V		CMP1- CMP2- CMP3-			AD5	OPIN-			
PB6	V	V	V		CMP1- CMP2- CMP3-		PCIN	AD6	OPIN-			
PB7	V	V	V		CMP2+ CMP3+		PCIN	AD7	OPIN-	INT3		
PC0	output only					V						
PC1	output only					V						
PC2	output only					V						

引脚名称	输入/ 输出			Special Functions								
	I / O	上拉	唤醒	XIN XOUT	比较器	PWM	脉冲捕获	ADC	OPA	外部中断	外部复位	烧录
PC3	output only					V						
PC4	output only					V						
PC5	output only					V						
PD0	V	V	V		COMP3+			AD10	OPIN-			
PD1	V	V	V		COMP3-			AD11	OPIN-			
PD2	V	V	V		COMP3-			AD12	OPIN-			
PD3	V	V	V		COMP3-			AD13	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. 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。
5. 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。
6. **建议在 VDD/AVDD 和 GND/AGND 引脚间并联 1uF 和 0.1uF 的电容。**

4. 中央处理器(CPU)

4.1. 功能描述

PFC887 内有八个处理单元，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，每个 FPPA 可以独立执行自己程序，达到并行处理效能。

4.1.1. 处理单元

八个 FPPA 共享 5Kx16 bits MTP 程序存储器，512 bytes 数据 SRAM 以及所有的 IO 口，这八个 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不同 FPPA 相对应的周期。图 1 所示为 FPPA 单元硬件框图。

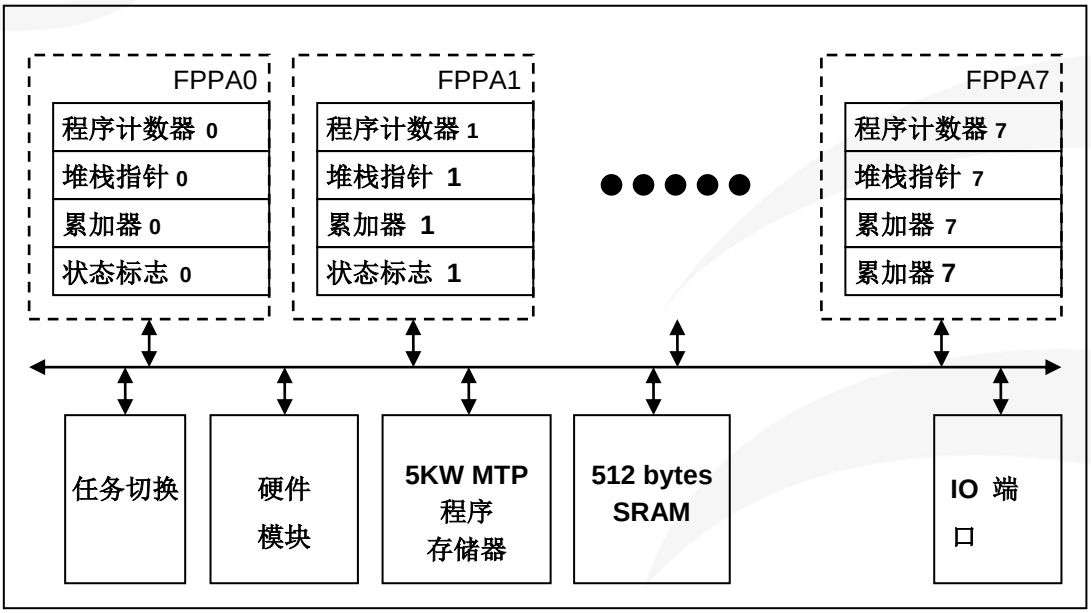


图 1：FPPA 单元结构

这八个 FPPA 单元各自独立运作在相斥的时钟周期，可以独立工作。

通过 pmode 命令可将系统性能共享给指定的 FPPA 单元；请参阅 *pmode* 说明。带宽分配与 FPPA enable 无关，即使 FPPA 被禁用，带宽也会被分配给指定的 FPPA 单元。

两个处理单元工作模式

pmode=0 时，会将带宽仅分配给两个 FPPA 单元，时序图如图 2 所示。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。

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，……第（M+4）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，……第（N+3）时钟周期执行程序。

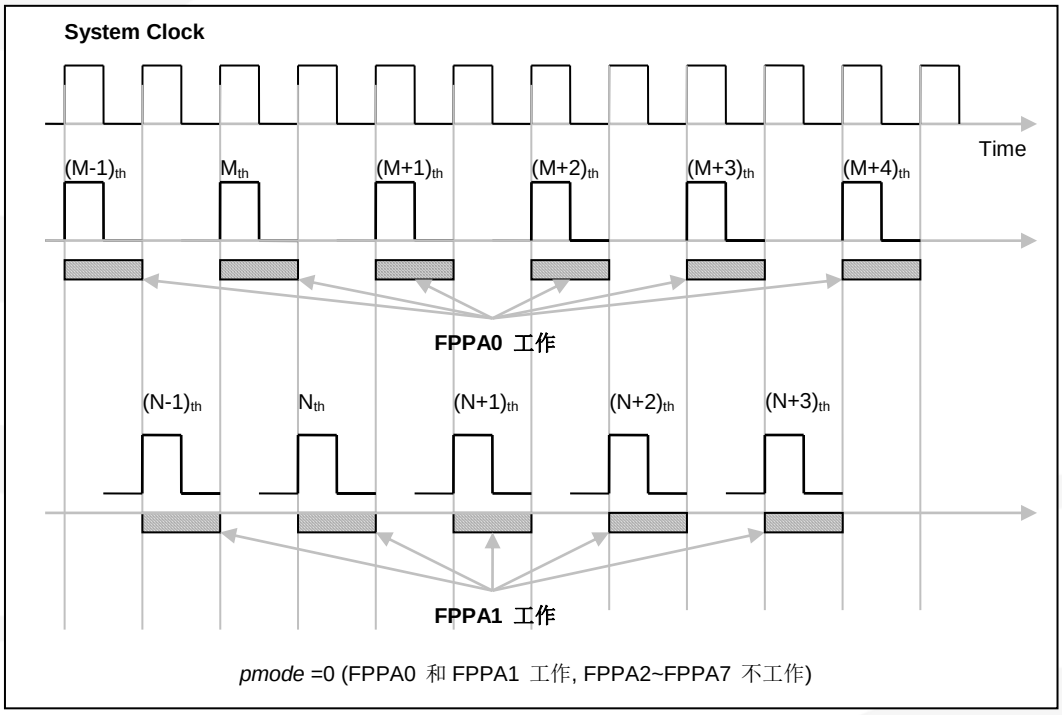


图 2：*pmode*=0 时处理单元时序图

四个处理单元工作模式

pmode=6 时，会将带宽分配给四个 FPPA 单元 (FPPA0, FPPA1, FPPA2, FPPA3)，时序图如图 3 所示。每个 FPPA 单元具有整个系统四分之一的计算能力，例如，如果系统时钟为 8MHz，FPPA0、FPPA1、FPPA2 和 FPPA3 将分别在 2MHz 时钟下工作。此时，FPPA4、FPPA5、FPPA6 和 FPPA7 不工作。

对于 FPPA0、FPPA1、FPPA2 和 FPPA3 而言，其程序将按顺序每四个系统时钟执行一次，如图：FPPA0 在第（M-1），第 M，……第（M+4）时钟周期执行程序；FPPA1 在第（N-1），第 N，……第（N+3）时钟周期执行程序；FPPA2 在第(O-1)，第 O，……第(O+3)时钟周期执行程序；FPPA3 在第(P-1)，第(P)，……第(P+3)时钟周期执行程序。

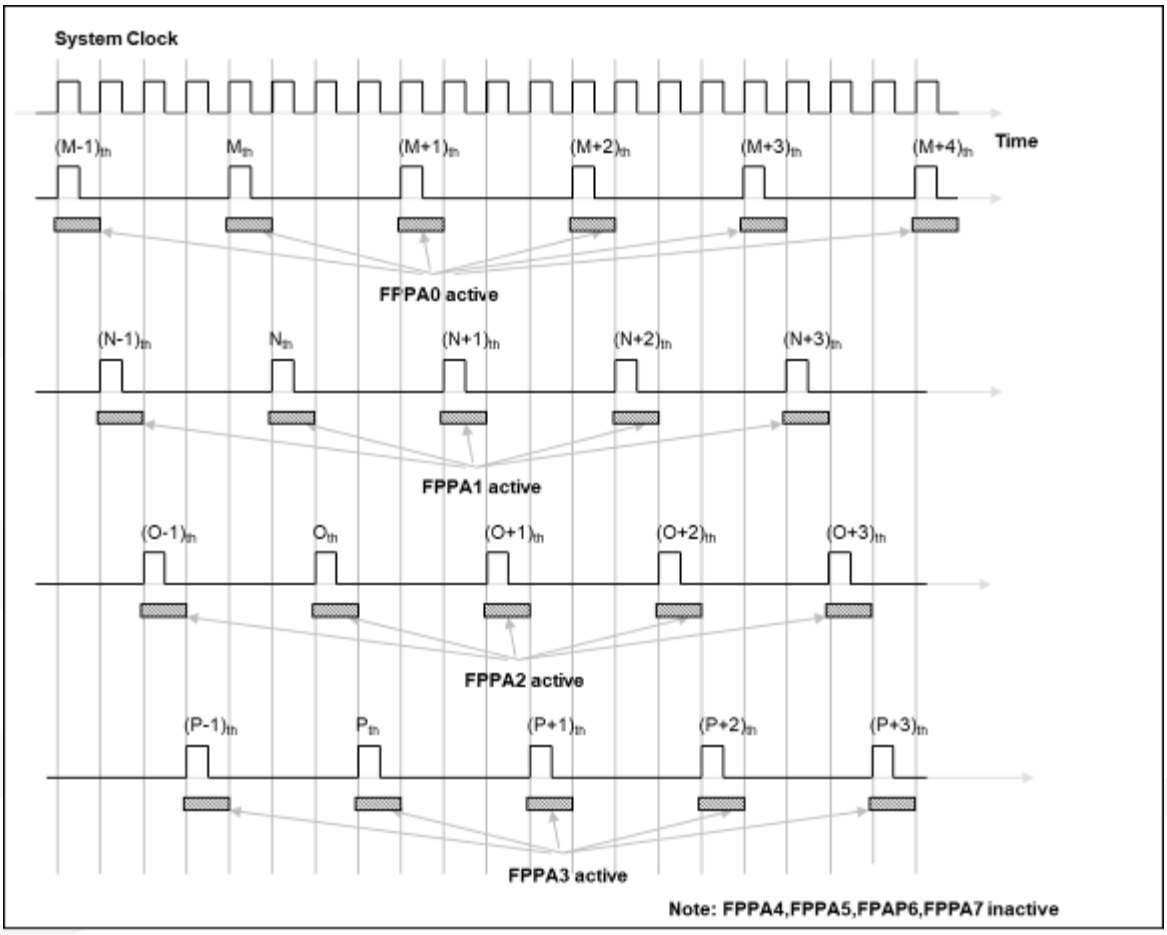


图 3：*pmode*=6 时处理单元时序图

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，其余 FPPA 单元可以由用户的程序来决定是否启用。所有 FPPA 单元均可被任意 FPPA 单元停用，包括停用本身这一 FPPA 单元。

4.1.2. 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。

PFC887 程序计数器的位长度是 13。在硬件复位后，FPPA0 的程序计数器为 0、FPPA1 为 1、FPPA2 为 2、FPPA3 为 3、FPPA4 为 4、FPPA5 为 5、FPPA6 为 6、FPPA7 为 7。当中断发生时，FPPA0 的程序计数器会跳转到 0x10 的中断服务程序处。每个 FPPA 单元都具有各自独立的程序计数器来控制其程序执行顺序。

4.1.3. 程序结构

开机后，FPPA0~FPPA7 的程序开始地址依次是 0x000~0x007。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。 PFC887 的基本软件结构如图 4 所示，使用了四个 FPPA 单元。四个 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。

```
// Page 1
.romadr   0x00
//  程序开始
goto     FPPA0Boot;
goto     FPPA1Boot;
goto     FPPA2Boot;
goto     FPPA3Boot;
//------中断服务程序----------------
.romadr    0x010
pushaf ;
t0sn       intrq.0;      //PA.0 ISR
goto       ISR_PA0;
t0sn       intrq.1;      //PB.0 ISR
//------中断服务程序结束---------
//------ FPPA0 程序开始 ----------
FPPA0Boot :
 //--- FPPA0 初始化…
…
FPPA0Loop:
…
goto FPPA0Loop:
//------ End of FPPA0 --------
```

```
// Page 2
//------ FPPA1 程序开始 ----------
FPPA1Boot :
//--- FPPA1  初始化…
FPPA1Loop:
…
goto FPPA1Loop:
//--------- FPPA1 程序结束 --------
//------ FPPA2 程序开始 ----------
FPPA2Boot :
//--- FPPA2  初始化…
FPPA2Loop:
…
goto FPPA2Loop:
//--------- FPPA2 程序结束 --------
//------ FPPA3  程序开始----------
FPPA3Boot :
//--- FPPA3  初始化…
FPPA3Loop:
…
goto FPPA3Loop:
//--------- End of FPPA3 --------
```

图 4：程序结构

4.1.4. 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。所有的 FPPA 单元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

PFC887 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。所有 FPPA 单元的用户程序代码都存储在 5KW 程序存储器中，如表 1 所示。

复位之后，FPPA0 的初始地址是 0x000，FPPA1 为 0x001，FPPA2 为 0x002，FPPA3 为 0x003，FPPA4 为 0x004，FPPA5 为 0x005，FPPA6 为 0x006，FPPA7 为 0x007。中断入口在 0x010，只有 FPPA0 能使用中断功能。

MTP 存储器从地址“0xFF8 to 0xFFF”供系统使用，从“0x008 to 0x00F”和“0x011 to 0xFF7”地址空间是用户的程序空间。地址“0x000 to 0x007”为 FPPA 单元的初始地址。

地址	功能
0x000	FPPA0 起始地址– *goto* 指令
0x001	FPPA1 起始地址– *goto* 指令
0x002	FPPA2 起始地址– *goto* 指令
0x003	FPPA3 起始地址– *goto* 指令
0x004	FPPA4 起始地址– *goto* 指令
0x005	FPPA5 起始地址– *goto* 指令
0x006	FPPA6 起始地址– *goto* 指令
0x007	FPPA7 起始地址– *goto* 指令
0x008	用户程序区
•	•
0x00F	用户程序区
0x010	中断入口地址
0x011	用户程序区
•	•
•	•
0x13DF	用户程序区
0x13E0	系统使用
•	•
0x13FF	系统使用

表 1：PFC887 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

为了保证用户编写程序时最大的弹性，所有 FPPA 单元共享用户程序存储器，由编译器自动分配程序空间，如果没有特别需求，用户不需要指定地址。使用两个处理单元工作模式下，程序存储器分配情形如表 2 所示：

地址	功能
0x000	FPPA0 起始地址– goto 指令(*goto* 0x020)
0x001	FPPA1 起始地址– goto 指令(*goto* 0x7A1)
0x002	保留
•	•
0x007	保留
0x008	不使用
•	•
0x00F	不使用
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
•	•
•	•
0x7A0	FPPA0 程序结束
0x7A1	FPPA1 程序开始
•	•
•	•
0xF37	FPPA1 程序结束
0xF38	不使用
•	•
•	•
0x13DF	不使用
0x13E0	系统使用
•	•
0x13FF	系统使用

表 2：程序存储器分配案例

4.2.2. 数据存储器 – SRAM

图 5 显示了 PFC887 数据存储器的结构以及使用，所有的 SRAM 数据存储器均可以透过 FPPA 单元在 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所有 FPPA 单元的堆栈记忆体。

每个 FPPA 单元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。各 FPPA 单元的堆栈指针通过堆栈指针寄存器各自定义，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。PFC887 的 512 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。

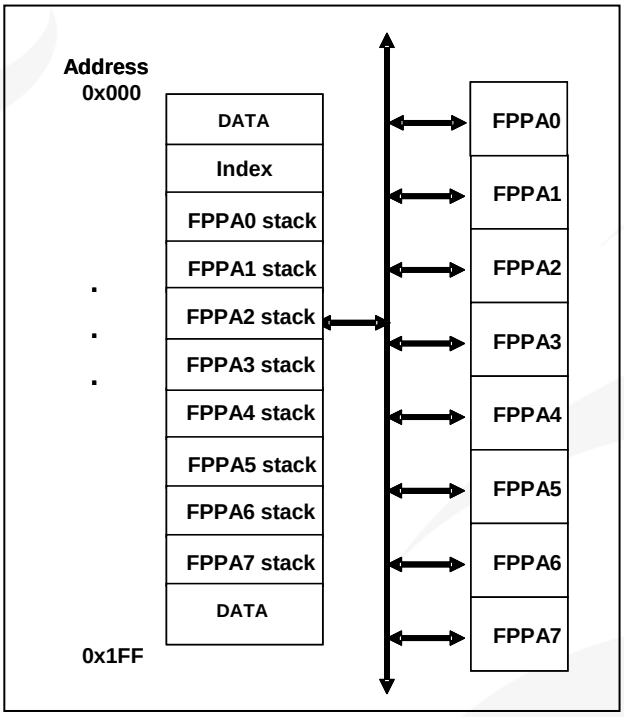


图 5：数据存储器结构和使用

4.2.3. 系统寄存器

PFC887 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 PFC887 的各寄存器存放地址及简要描述：

	+0	+1	+2	+3	+4	+5	+6	+7
0x00	*FLAG*	*FPPAEN*	*SP*	*CLKMD*	*INTEN*	*INTRQ*	*T16M*	*GDIO*
0x08	*INTEN2*	*INTRQ2*	*EOSC*	-	*INTEGS*	*PADIER*	*PBDIER*	*PDDIER*
0x10	*PA*	*PAC*	*PAPH*	-	*PB*	*PBC*	*PBPH*	*PC*
0x18	*AMPC*	*T16NM*	-	-	-	-	*GPC2C*	*GPC2S*
0x20	*ADCC*	*ADCM*	*GPC3C*	*HOP*	*TM2C*	*TM2CT*	*TM2S*	*TM2B*
0x28	*TM3C*	*TM3CT*	*TM3S*	*TM3B*	-	-	-	-
0x30	*PWMGC*	*PWMGM*	*PLSCC*	*PLSCS*	*GPC1C*	*GPC1S*	-	*PWMGC1*
0x38	*PWMGC2*	-	*OPR2*	*MISC*	*EARITH*	*PD*	*PDC*	*PDPH-*
0x40	*PWM2PC0*	*PWM2PC1*	*PWM2PC2*	*PWM2PC3*	*M8OP1H*	*M8OP1L*	*M8OP2*	*M8RS2*
0x48	*M8RS1*	*M8RS0*	*ADCRH*	*ADCRL*	*PLSPWH*	*PLSPWL*	*PLSPHH*	*PLSPHL*
0x50	*PWMCUBH*	*PWMCUBL*	*PWM0DTH*	*PWM0DTL*	*PWM1DTH*	*PWM1DTL*	*PWM2DTH*	*PWM2DTL*
0x58	*PWMDZ*	*PWMGS1*	*PWM2PC4*	*PWM2PC5*	-	-	*PWMADCH*	*PWMADCL*
0x60	*M16OP1H*	*M16OP1L*	*M16OP2H*	*M16OP2L*	*M16RS3*	*M16RS2*	*M16RS1*	*M16RS0*
0x68	*D16DEH*	*D16DEL*	*D16DSH*	*D16DSL*	*D16QUH*	*D16QUL*	*D16REH*	*D16REL*
0x70	-	-	-	*T16NBH*	*T16NBL*	-	-	-
0x78	*GPC3S*	*INTEGS2*	-	-	-	-	-	-

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 (*FPPAEN*), 地址= 0x01

位	初始值	读/写	描 述
7	0	读/写	FPPA7 启用。此位是用来启用 FPPA7。 0/1：停用/启用
6	0	读/写	FPPA6 启用。此位是用来启用 FPPA6。 0/1：停用/启用
5	0	读/写	FPPA5 启用。此位是用来启用 FPPA5。 0/1：停用/启用
4	0	读/写	FPPA4 启用。此位是用来启用 FPPA4。 0/1：停用/启用
3	0	读/写	FPPA3 启用。此位是用来启用 FPPA3。 0/1：停用/启用
2	0	读/写	FPPA2 启用。此位是用来启用 FPPA2。 0/1：停用/启用
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.跳跃设置寄存器 (*HOP*), 地址= 0x23

位	初始值	读/写	描 述
7	0	读/写	PWM 中断模式 0：当计数值达到上限寄存器设定值时产生中断请求 1：当计数为 0 时产生中断请求
6	0	读/写	PWM 计数器触发 ADC 模式 0：向上计数达到寄存器 PWMADC 设定值时触发 ADC 模式 1：向下计数达到寄存器 PWMADC 设定值时触发 ADC 模式
5	-	-	保留
4	0	读/写	PWM 触发 ADC 功能（由 ADC 转换完成后硬件清零） 0：停用 1：启用
3 - 0	-	-	保留

4.2.3.4.选项寄存器 2 (*OPR2*)，地址 = 0x3A

位	初始值	读/写	描 述
7 - 4	-	-	保留，请保持为 0
3	0	只写	Timer16N 时钟预分频器选择选项。请查看 *T16NM* 寄存器
2	0	只写	Timer16 时钟预分频器选择选项。请查看 *T16M* 寄存器
1	0	只写	外部中断 1 引脚选择选项
0	0	只写	外部中断 0 引脚选择选项

4.2.3.5.杂项寄存器 (*MISC*)，地址= 0x3B

位	初始值	读/写	描 述
7	-	-	保留
6	0	WO	外部晶体低功率设置 0 / 1：启用 / 停用
5 - 3	-	-	保留
2	0	WO	停用 LVR 功能： 0 / 1：启用 / 停用
1 - 0	00	WO	看门狗时钟超时时间设定： 00：保留 01：4096 个 ILRC 时钟周期 10：16384 个 ILRC 时钟周期 11：保留

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 512 字节以内，不可以超过 512 字节。PFC887 每个 FPPA 单元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，意味着每个 FPPA 单元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：

```
        .ROMADR0
        GOTO        FPPA0
        GOTO        FPPA1
        ...
        .RAMADR0                    // 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 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 6 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。

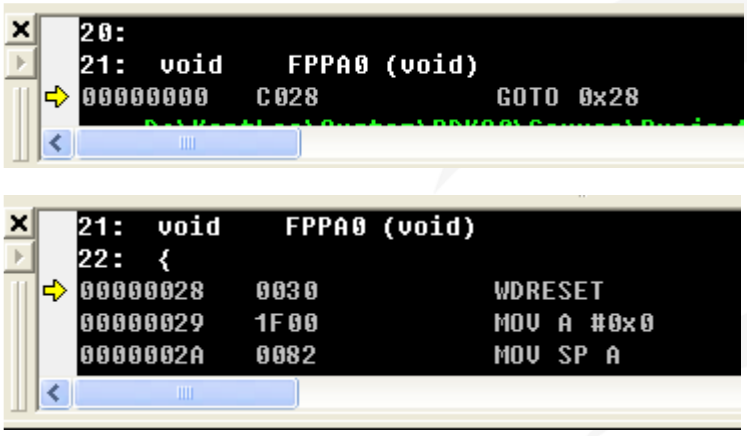


图 6：在 Mini-C 语言下的堆栈设定

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。

4.4. 程序选项 Code Option

选项	选择	说明
Security	Enable	MTP 内容加密，程序不允许被读取
	Disable（默认）	MTP 内容不加密，程序可以被读取
LVR	4.5V	选择 LVR = 4.5V
	4.0V	选择 LVR = 4.0V
	3.75V	选择 LVR = 3.75V
	3.5V	选择 LVR = 3.5V
	3.3V	选择 LVR = 3.3V
	3.15V	选择 LVR = 3.15V
Drive	Low	IO 驱动 / 灌电流为 low
	Normal（默认）	IO 驱动 / 灌电流为 Normal

5. 振荡器和系统时钟

PFC887 提供 3 个振荡器电路：内部高频 RC 振荡器(IHRC)和内部低频 RC 振荡器(ILRC)和外部晶体振荡器(EOSC)。

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	开机后默认状态
EOSC	*EOSCR*.7	停用
IHRC	*CLKMD*.4	启用
ILRC	*CLKMD*.2	启用

表 3：PFC887 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器

开机后，IHRC 和 ILRC 振荡器是自动启用的。IHRC 频率能通过 *IHRCR* 寄存器校准,通常校准到 16MHz。校准后的频率偏差通常在 2%以内。然而，IHRC 的频率会因为电源电压和工作温度产生漂移。ILRC 的频率也会因工厂生产、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。

PFC887 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

晶体振荡器的工作频率可以从 32 KHz 到 16MHz，取决于所放置的晶体。图 8 显示了该应用程序下的硬件连接。

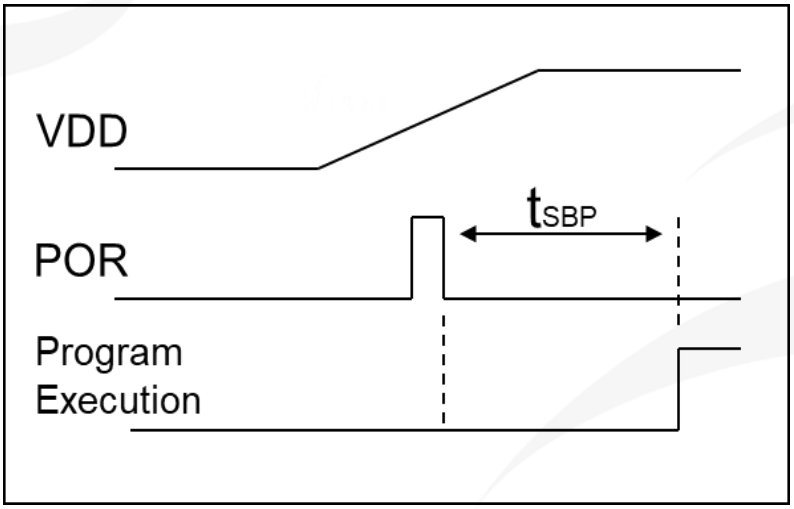


图 8：晶体振荡的连接

5.2.1. 外部振荡器设置寄存器 (*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/ 启用
6 - 5	00	只写	外部晶体振荡器的选择 00：保留 01：低驱动能力，适用于较低频率晶体，例如：32KHZ 晶体振荡器 10：中驱动能力，适用于中等频率晶体，例如：4MHZ 晶体振荡器 11：高驱动能力，适用于较高频率晶体，例如：8MHZ 晶体振荡器
4	-	-	保留，请设为 1
3 - 2	-	只写	为了兼容，请设为 11
1 - 0	-	-	保留

5.2.2. 外部振荡器的使用及注意事项

除了晶体外，外部电容和 PFC887 的选项应该在 EOSCR 寄存器中微调，以获得良好的正弦波形。EOSCR.7 用于使能晶体模块。采用 EOSCR.6 和 EOSCR.5 设定不同的驱动电流，以满足不同晶体振荡器频率的要求。

表 6 给出了不同晶体振荡器的 C1 和 C2 的推荐值；并给出了相应条件下的实测启动时间。由于晶体或谐振器有其自身的特性，不同类型的晶体或谐振器的电容和启动时间可能略有不同，C1 和 C2 的合适值请参考其规范。

频率	C1	C2	测量开机时间	条件
8MHz	10pF	10pF	10ms	(*EOSCR*[6:5]=11)
4MHz	10pF	10pF	20ms	(*EOSCR*[6:5]=10)
32KHz	10pF	10pF	450ms	(*EOSCR*[6:5]=01)

表 6：晶体振荡器和谐振振荡器的推荐值 C1 和 C2

使用晶振时 PA7 和 PA6 的配置：
(1) 设置 PA7 和 PA6 为输入；
(2) PA7 和 PA6 内上拉电阻为关闭;
(3) 使用 PADIER 寄存器将 PA7 和 PA6 设置为模拟输入，防止漏电。

注意: 请仔细阅读 PMC-APN013。根据 PMC-APN013 的要求，晶体振荡器应合理运用。如出现以下情况导致 IC 启动缓慢或无法启动，用户晶振质量不好，使用条件不合理、PCB 清洗剂漏电流、PCB 布局不合理，均不承担责任。

使用晶振时，必须注意晶振开启的稳定时间。振荡器的稳定时间取决于频率、晶体类型、外部电容和电源电压。在切换到晶振之前，用户必须确保晶振是稳定的。参考程序如下：

```
void    FPPA0 (void)
{
    .ADJUST_IC   SYSCLK=IHRC/16, IHRC=16MHz, VDD=5V
    ...
    $   EOSCR   Enable, 4Mhz;       // EOSCR = 0b110_10000;
    $   T16M    EOSC, /1, BIT13;    // T16M 由 Bit13 0 => 1, Intrq.T16 => 1
                                    // 假设晶振振荡器稳定
    WORD    count   =   0;
    stt16   count;
    Intrq.T16 =   0;
    while (! Intrq.T16)   NULL;     // 从 0x0000 到 0x2000 计数, 然后触发
                                    INTRQ.T16
    clkmd=0xB4;                     // 切换系统时钟到 EOSC;
    clkmd.4 = 0;                    // 停用 IHRC
    ...
}
```

请注意，进入掉电模式前应将晶振完全关闭，以免发生意外唤醒事件。

5.3. 系统时钟和 IHRC 校准

5.3.1. 系统时钟

系统时钟的时钟源来自 IHRC、ILRC、或 EOSC,PFC887 系统时钟的硬件框图如图 7 所示。

上电后，运行的系统时钟为 ILRC/1，用户可以通过设置 CLKMD 寄存器随时更改系统时钟，写入 CLKMD 寄存器后立即将新的系统时钟更改为新的系统时钟。

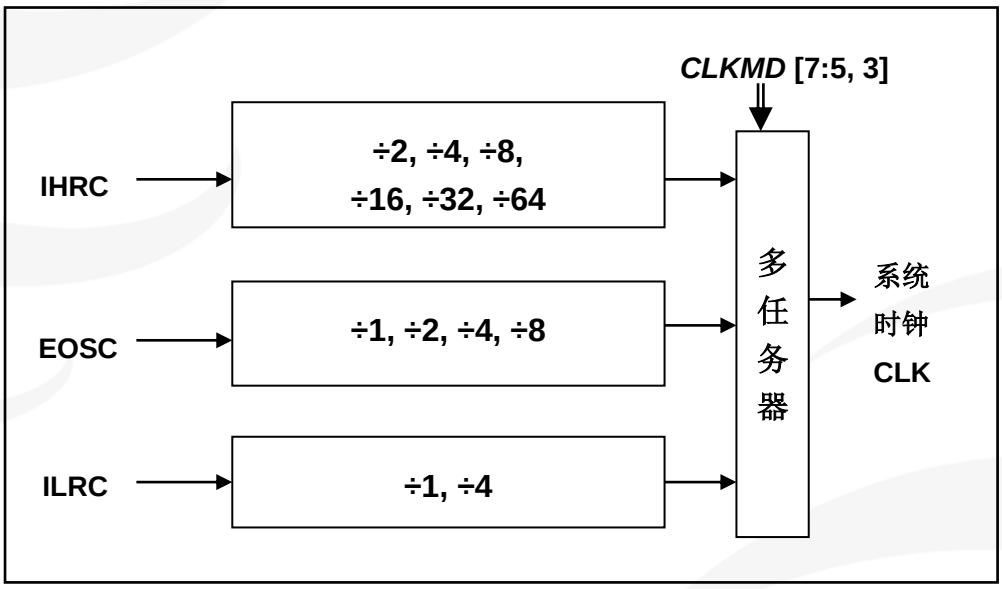


图 7：系统时钟选择

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：保留 011：IHRC/32 100：IHRC/64 101：EOSC/8 11x：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

在芯片生产制造时， IHRC 频率和 Bandgap 参考电压都有可能稍微不同， PFC887 提供 IHRC 频率 Bandgap 校准来消除这些差异，校准功能可以被用户的程序选择并编译，同时这个命令会自动嵌入用户的程序里面。

校准命令如下所示：

.ADJUST_IC SYSCLK=IHRC/(**p1**), IHRC=(**p2**)MHz, V_{DD}=(**p3**)V, Bandgap=(**p4**);

p1=2, 4, 8, 16, 32, 64; 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=3.15 ~ 5.5; 根据不同的电源电压校准芯片。

p4= On 或者 Off; 打开或者关闭 Bandgap 校准。

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 4 所示：

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

表 4：IHRC 频率校准选项

下面显示在不同的选项下，PFC887 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@VDD=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=ILRC, IHRC=16MHz, V_{DD}=5V**

开机后，CLKMD = 0xE4：

a. IHRC 的校准频率为 16MHz@VDD=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

(3) ***.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
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，PFC887 系统时钟可以随意在 IHRC 和 ILRC、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：“帮忙”→“使用手册”→“IC 介绍” →“缓存器介绍”→“CLKMD”。

例 1: 系统时钟从 ILRC 切换到 IHRC/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 能在这里停用
```

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 PFC887 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，PFC887 的所有寄存器将被设置为默认值。

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

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

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

6.1. 上电复位 - POR

开机时，POR（上电复位）是用于复位 PFC887；但是，上电后电源电压可能不太稳定，为确保单片机是工作在电压稳定的状态，在执行第一条指令之前，PFC887 会延迟 1024 个 ILRC 时钟周期，这时间就是 T_{SBP}，如图 8 所示。开机后，默认的系统时钟是 ILRC。如果用户希望将系统时钟源从 ILRC 切换到 IHRC，则必须启用相应的振荡器模块，并确保时钟已经稳定。

发生上电复位时，PFC887 数据存储器的值处于不确定的状态。

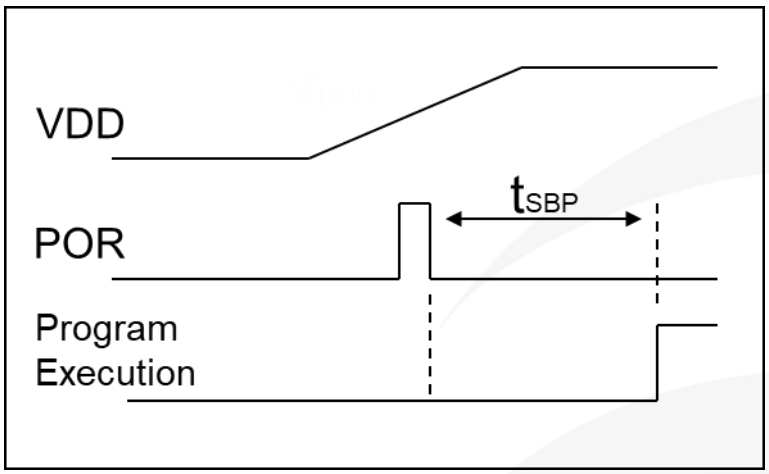


图 8：上电复位时序图

图 9 显示的是典型开机流程。请注意，上电复位后 FPPA1~FPPA7 是停用，建议不要在 FPPA0 以及系统初始化完成前，启用 FPPA1~FPPA7。

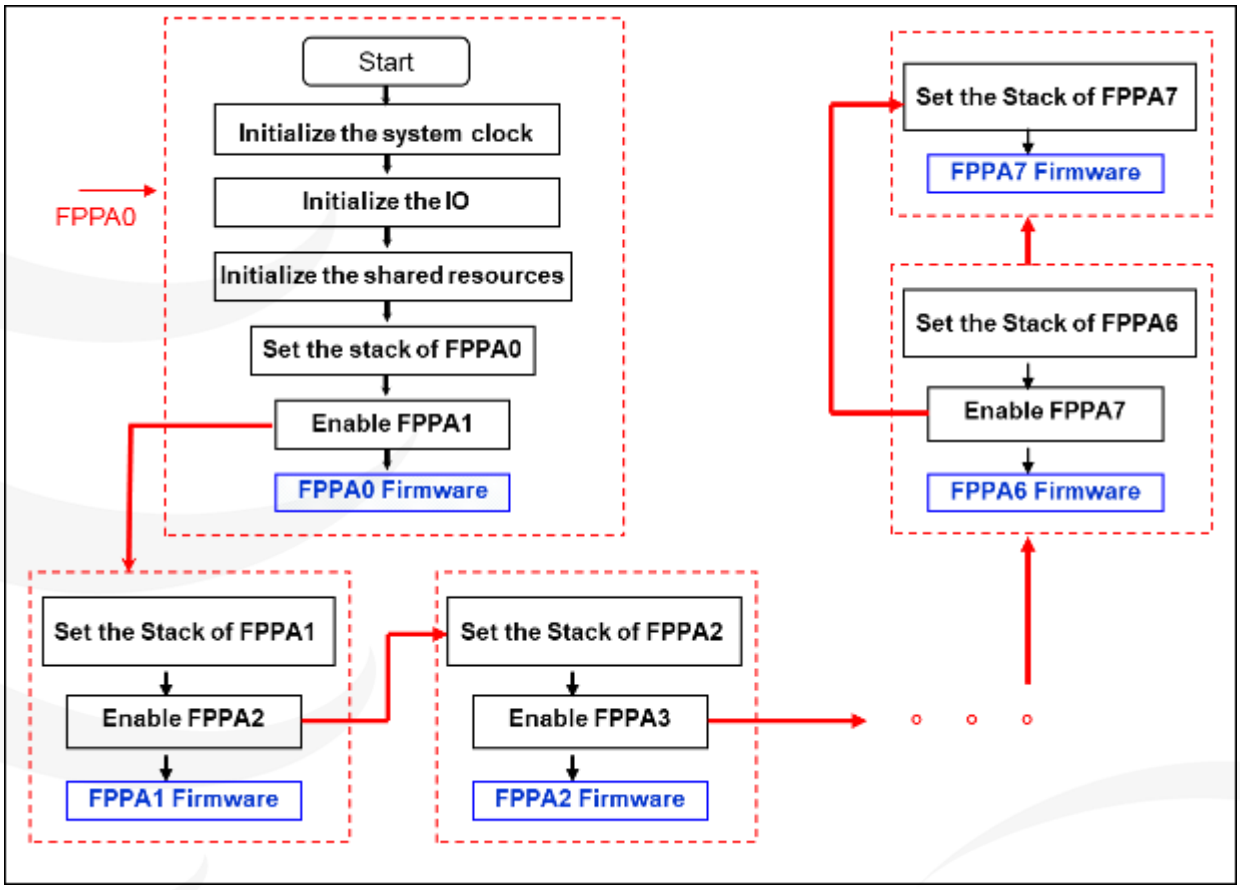


图 9：开机流程

6.2. 低压复位- LVR

用户可以根据需要选择不同的操作系统时钟；选择的操作系统时钟应与供电电压和 LVR 水平相结合，以保证系统的稳定。如果 VDD 低于 LVR 电平，则系统将出现 LVR 复位，其时序图如图 10 所示。

LVR 复位后，当 VDD 大于 VDR(SRAM 数据保留电压)时，将保留 SRAM 数据。但如果重新上电后清除 SRAM，数据将无法保存，VDD 小于 VDR 时数据内存处于不确定状态。

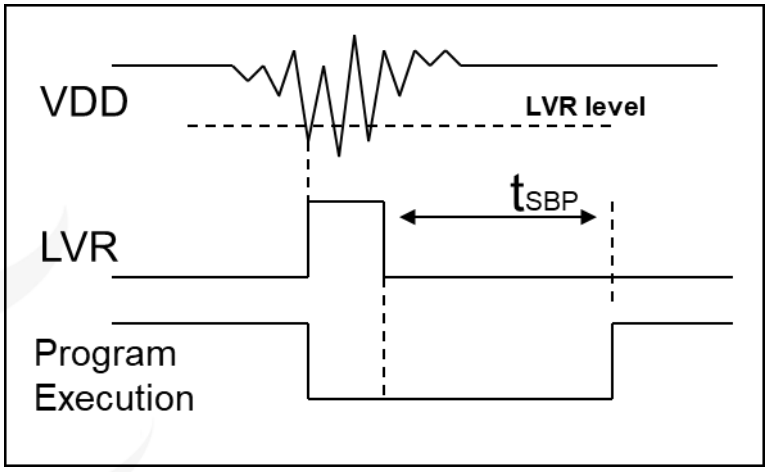


图 10：低压复位时序图

使用者可以在不同的需求下选择不同的系统时钟，选定的系统时钟应与电源电压和 LVR 的基准位结合起来才能使系统稳定。LVR 的基准位是在编译过程中选择，不同系统时钟对应的 LVR 设定，请参考章节 13.1 中系统时钟的最低工作电压。

下面是工作频率、电源电压和 LVR 水平设定的建议：

SYSCLK	VDD	LVR
8MHz	≧3.3V	3.3V
4MHz	≧ 3.15V	3.15V

表 5：LVR 设置参考

使用者可以设定寄存器 *MISC*.2 为 1 停用 LVR 功能，但此时应确保 VDD 在芯片最低工作电压以上，否则 IC 可能工作不正常。

杂项寄存器 (*MISC*)， 地址= 0x3b			
位	初始值	读/写	描 述
7	-	-	
6	0	WO	外部晶体低功率设置 0 / 1： 停用/ 启用
5 - 3	-	-	
2	0	WO	停用 LVR 功能 0 / 1：启用 / 停用
1 - 0	00	WO	看门狗时钟超时时间设定： 00：保留 01：4096 个 ILRC 时钟周期 10：16384 个 ILRC 时钟周期 11：保留

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，看门狗默认为开，但程序执行 ADJUST_IC 时，会将看门狗关闭，若要使用看门狗，需重新配置打开。当 ILRC 关闭时，看门狗也会失效。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，PFC887 将复位并重新运行程序，其复位时序图如图 14 所示。发生 WDT 复 PFC887 数据存储器的值将被保留，*GDIO* 寄存器（地址=0x07）的值也将保持不变。

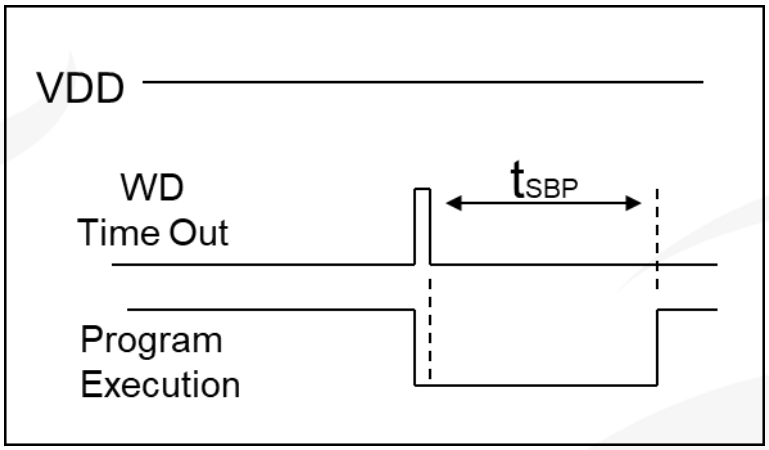


图 11：看门狗超时溢出的相关时序

利用寄存器 *MISC*[1:0]可选择两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

PFC887 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，PFC887 数据存储器的值将被保留。

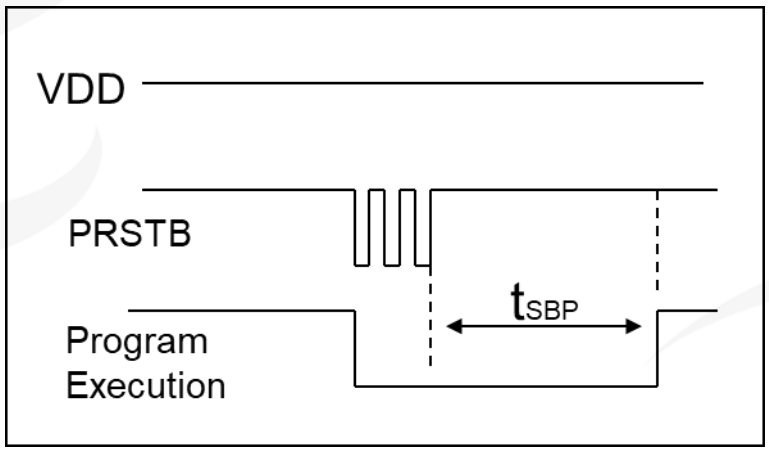


图 12：外部引脚复位的相关时序

7. 系统工作模式

PFC887 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。建议在 *stopexe* 指令后添加 *nop* 指令。

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 仅包含 Timer16。请注意 Timer 16N，TM2，TM3，PWMG 均无唤醒功能）

(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(*GPC1C/GPC2C*)与 *GPCS*.6 为 1(*GPC1C/GPC2C*)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。

使用 stopexe 指令之前，必须关闭看门狗定时器。示例如下：

```
CLKMD.En_WatchDog    =    0;          // 关闭看门狗计数器
stopexe;
nop;
....                                  // 省电
Wdreset;
CLKMD.En_WatchDog    =    1;          // 启用看门狗计数器
```

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。进入掉电模式前，必须启用内部低频 RC 振荡器，也就是说在执行 *stopsys* 命令之前，寄存器 *CLKMD* 的第 2 位必须被设置为 1。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，PFC887 内部详细的状态：

(1) 所有的振荡器模块被关闭。

(2) 启用内部低频 RC 振荡器（设置 *CLKMD* 寄存器的第 2 位）

(3) MTP 存储器被关闭。

(4) SRAM 和寄存器内容保持不变。

(5) 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，PFC887 可以通过切换 IO 引脚恢复正常工作。而 Timer 的唤醒只适用于省电模式。表 6 显示 *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
stopsys	是	否
stopexe	是	是

表 6：掉电模式和省电模式在唤醒源的差异

当使用 IO 引脚来唤醒 PFC887，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 1024 个 ILRC 时钟周期。

模式	**唤醒模式**	**切换 IO 引脚的唤醒时间(t_{WUP})**
STOPEXE 省电模式	任一模式	1024 * T_{ILRC}，这里的 T_{ILRC} 是指 ILRC 时钟周期
STOPSYS 掉电模式	任一模式	1024 * T_{ILRC}，这里的 T_{ILRC} 是指 ILRC 时钟周期

表 7：唤醒时间

8. 中断

PFC887 有 12 个中断源：

(1) 四个外部中断源（PA0 或 PA5，PB0 或 PB7，由 INTEGS 寄存器指定触发中断的边缘类型）
(2) LVD 中断
(3) ADC 中断
(4) GPC 中断
(5) GPC2 中断
(6) GPC3 中断
(7) Timer16 中断
(8) Timer16N 中断
(9) Timer2 中断
(10) Timer3 中断
(11) PWM 产生器中断
(12) 三相霍尔中断

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 16。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。

只有 FPPA0 可以接受中断请求，其他 FPPA 单元不被中断影响。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，用户应该谨慎使用存储器。可由软件编程调整栈点在存储器里的位置，每个 FPPA 单元堆栈指针的深度可以完全由用户指定，以实现最大的系统弹性。

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。

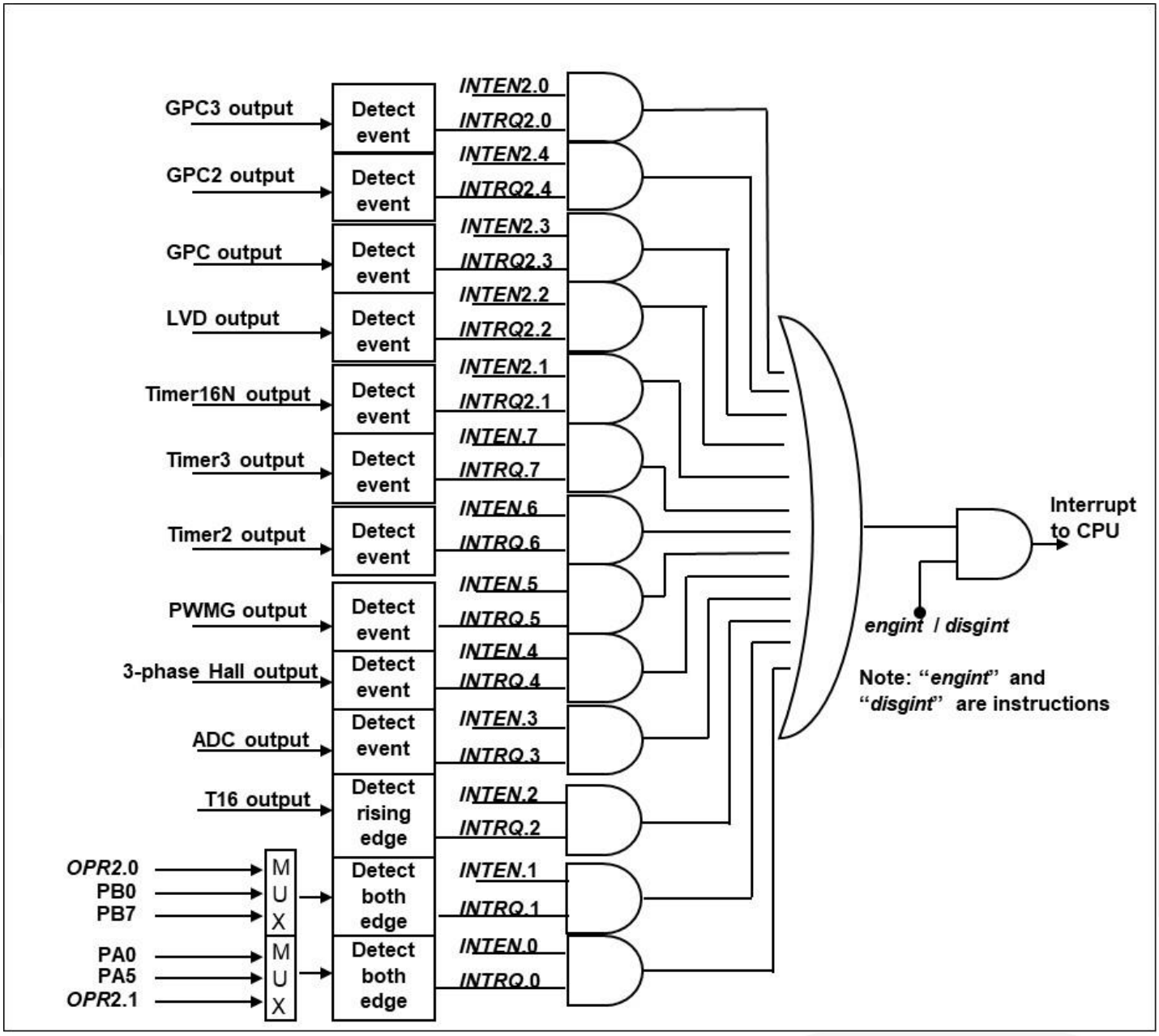


图 13：中断控制硬件图

选项寄存器 2 (*OPR2*)，地址=0x3A			
位	初始值	读/写	描 述
7 – 4	-	-	保留。请保持为 0.
3	0	只写	Timer16N 时钟预分频器选择选项。请查看 T16NM 寄存器。
2	0	只写	Timer16 时钟预分频器选择选项。请查看 T16M 寄存器。
1	**0**	**只写**	**外部中断 1 引脚选择选项**
0	**0**	**只写**	**外部中断 0 引脚选择选项**

8.1. 中断允许寄存器 (*INTEN*), 地址= 0x04

位	初始值	读/写	描 述	
7	0	读/写	启用从 Timer3 的溢出中断。0/1：停用/启用。	
6	0	读/写	启用从 Timer2 的溢出中断。0/1：停用/启用。	
5	0	读/写	启用从 PWM 产生器的溢出中断。0/1：停用/启用。	
4	0	读/写	启用来自三相 Hall 的中断。0/1：停用/启用。	
3	0	读/写	启用从 ADC 的中断。0/1：停用/启用。	
2	0	读/写	启用从 Timer16 的溢出中断。0/1：停用/启用。	
1	0	读/写	*OPR2*.0=0	启用从 PB0 的溢出中断。0/1：停用/启用。
			OPR2.0=1	启用从 PB7 的溢出中断。0/1：停用/启用。
0	0	读/写	*OPR2*.1=0	启用从 PA0 的溢出中断。0/1：停用/启用。
			OPR2.1=1	启用从 PA5 的溢出中断。0/1：停用/启用。
其中，OPR2 是选择寄存器，地址=0x3A。				

8.2. 中断允许 2 寄存器 (*INTEN2*), 地址= 0x08

位	初始值	读/写	描 述
7 - 5	-	-	保留。
4	0	读/写	启用从 GPC2 的中断。0/1：停用/启用。
3	0	读/写	启用从 GPC1 的中断。0/1：停用/启用。
2	0	读/写	启用从 LVD 侦测器的中断。0/1：停用/启用。
1	0	读/写	启用从 Timer16N 的溢出中断。0/1：停用/启用。
0	0	读/写	启用从 GPC3 的中断。0/1：停用/启用。

8.3. 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	PWM 产生器的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求	
4	-	读/写	三相 Hall 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	*OPR2*.0=0	PB0 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
		读/写	*OPR2*.0=1	PB7 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	*OPR2*.1=0	PA0 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
			OPR2.1=1	PA5 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
其中，OPR2 是可选寄存器，地址=0x3A。				

8.4. 中断请求 2 寄存器(*INTRQ2*), 地址= 0x09

位	初始值	读/写	描 述
7 - 5	-	-	保留
4	-	读/写	GPC2 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	GPC1 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	LVD 侦测器的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
1	-	读/写	Timer16N 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	-	R/W	GPC3 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

8.5. 中断缘选择寄存器 (*INTEGS*), 地址= 0x0C

位	初始值	读/写	描 述
7 - 5	-	-	保留。
4	0	WO	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	WO	PB0/PB7 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：*OPR2*.0=0，选择 PB0 中断；*OPR2*.0=1，选择 PB7 中断。
1 - 0	00	WO	PA0/PA5 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：*OPR2*.1=0，选择 PA0 中断；*OPR2*.1=1，选择 PA5 中断。

8.6. 中断缘选择寄存器 2 (*INTEGS2*), 地址= 0x79

Bit	Reset	R/W	Description
7 – 6	-	-	保留。
5 - 4	00	WO	GPC3 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。
3 - 2	00	WO	GPC2 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	WO	GPC 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：
INTEN/INTEN2，*INTRQ/INTRQ2* 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/INTEN2* 为 0，*INTRQ/INTRQ2* 还是会被中断发生源触发。

8.7. 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。

8.8. 中断的一般步骤

当使用中断函数时，程序应该是:

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

8.9. 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
    INTRQ2  =  0;          // 清除 INTRQ2

    ENGINT                 // 启用全局中断
    ...
    DISGINT                // 停用全局中断
    ...
}

void Interrupt  (void)     // 中断程序
{
    PUSHAF                 // 存储 ALU 和 FLAG 寄存器

    //  如果 INTEN.PA0 在主程序会动态开和关，则表达式中可以判断 INTEN.PA0 是否为 1。
    //  例如： If (INTEN.PA0 && INTRQ.PA0)  {…}

    //  如果 INTEN.PA0 一直在使能状态，就可以省略判断 INTEN.PA0，以加速中断执行

    If (INTRQ.PA0)
    {                          // PA0 的中断程序
        INTRQ.PA0 = 0;         // 只须清除相对应的位 (PA0)
        ...
    }
    ...
    // (X:) INTRQ = 0;         不建议在中断程序最后，才使用 INTRQ = 0 一次全部清除
                               因为它可能会把刚发生而尚未处理的中断，意外清除掉.
    POPAF                      // 恢复 ALU 和 FLAG 寄存器
}
```

9. I/O 端口

9.1. IO 相关寄存器

9.1.1. IO 通用数据寄存器 (*GDIO*), 地址= 0x07

位	初始值	读/写	描述
7 - 0	00	读/写	IO 的通用数据。该端口是 IO 空间的通用数据缓冲区，在 POR 或 LVR 时被清除，在看门狗超时复位时保留原数据。 它可以执行 IO 操作 *wait0* gdio.x, *wait1* gdio.x 和 *tog* gdio.x 来替换寄存器空间支持的指令操作 *wait1* mem; *wait0* mem; *tog* mem。

9.1.2. 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
5	1	只写	启用 PA5 数字输入和系统唤醒。 1 / 0：启用 / 停用
4 - 1	1111	只写	启用 PA4 ~ PA1 数字输入和系统唤醒。 1 / 0：启用 / 停用
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

9.1.3. 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	0xFF	只写	启用 PB7 数字输入、系统唤醒和中断请求。 1 / 0：启用 / 停用
6 - 1		只写	启用 PB6~PB1 数字输入和系统唤醒。 1 / 0：启用 / 停用
0		只写	启用 PB0 数字输入、系统唤醒和中断请求。 1 / 0：启用 / 停用

9.1.4. 端口 D 数字输入启用寄存器 (*PDDIER*), 地址= 0x0F

位	初始值	读/写	描述
3 - 0	0XF	WO	启用 PD3~PD0 数字输入和系统唤醒。 1 / 0：启用 / 停用

9.1.5. 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

9.1.6. 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

9.1.7. 端口 A 上拉控制寄存器(*PAPH*)，地址= 0x12

位	初始值	读/写	描 述
7 - 0	0x00	读/写	端口 A 上拉控制寄存器。这些寄存器是用来定义端口 A 每个相应的引脚的上拉电阻的使能，且上拉电阻功能仅在输入模式有效。 0 / 1：输入/输出

9.1.8. 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

9.1.9. 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

9.1.10. 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

9.1.11. 端口 C 数据寄存器 (*PC*)，地址= 0x17

位	初始值	读/写	描 述
5 - 0	0x00	读/写	数据寄存器的端口 C

9.1.12. 端口 D 数据寄存器 (*PD*), 地址= 0x3d

位	初始值	读/写	描 述
3 - 0	0x0	读/写	数据寄存器的端口 D

9.1.13. 端口 D 控制寄存器(*PDC*), 地址= 0x3e

位	初始值	读/写	描 述
3 - 0	0x0	读/写	端口 D 控制寄存器。这些寄存器是用来定义端口 D 每个相应的引脚的输入模式或输出模式。 0/1：输入/输出

9.1.14. 端口 D 上拉控制寄存器 (*PDPH*), 地址= 0x3f

位	初始值	读/写	描 述
3 - 0	0x0	读/写	端口 D 上拉控制寄存器。这些寄存器是用来控制端口 D 每个相应引脚上拉电阻的使能。 0/1：停用/启用

9.2. IO 结构及功能

9.2.1. IO 引脚的结构

PFC887 的所有 IO 引脚都具有相同的结构，如下图 14.

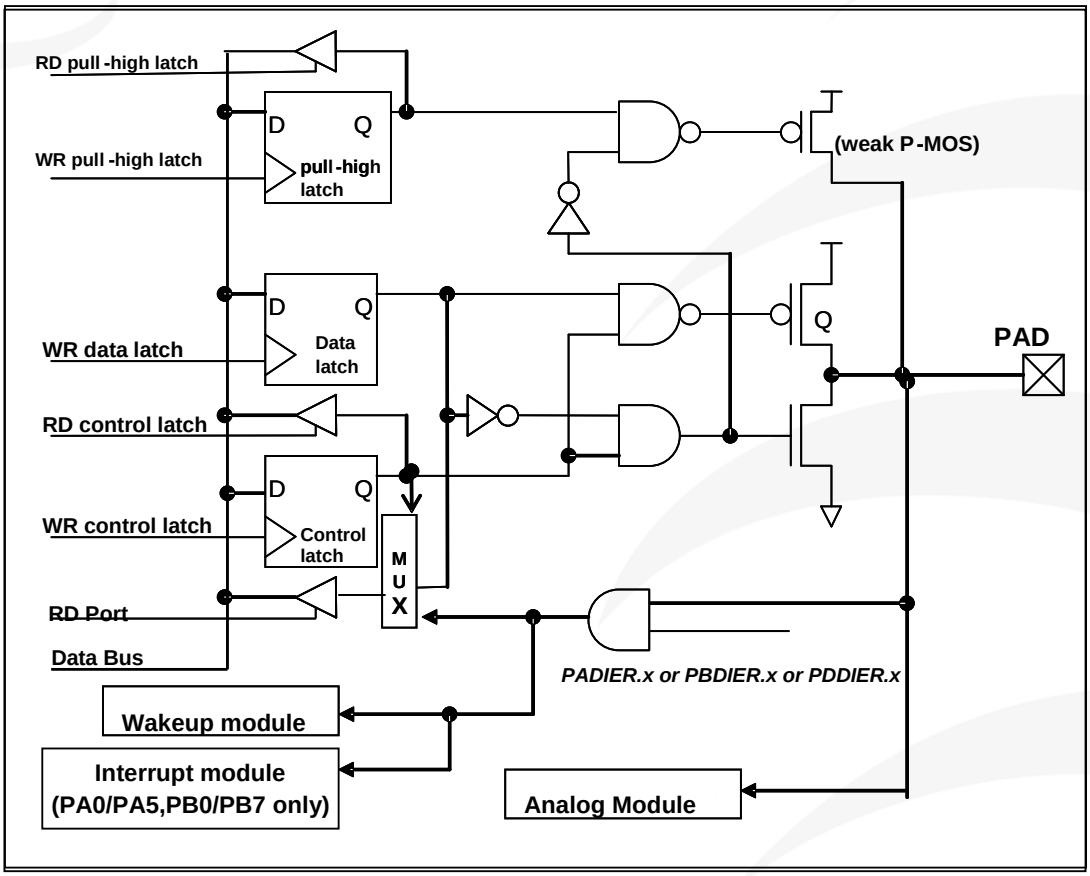


图 14：引脚缓冲区硬件图

9.2.2. IO 引脚的一般功能

(1) 输入、输出功能：

PFC887 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*)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 8 为端口 PA0 的设定配置表。

例如，表 8 顯示了連接埠 A 的位元 0 的設定表。

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

表 8：PA0 设定配置表

数字输出端口 C 可通过配置数据寄存器(*PC*)独立配置不同状态。

(2) 睡眠唤醒功能：

当 PFC887 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。

9.2.3. IO 的使用与设定

(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 is set to be PRSTB input pin.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

PFC887 内置一个 16 位硬件计数器 Timer16，其模块框图如图 18。

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]和 *OPR2*[2]可对时钟进行预分频处理，有÷1、÷2、÷4、÷8、÷16、÷32、÷64、÷128 等八种选项，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。当 *T16NM*[0]=0 时，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 en *T16NM*[0] 。

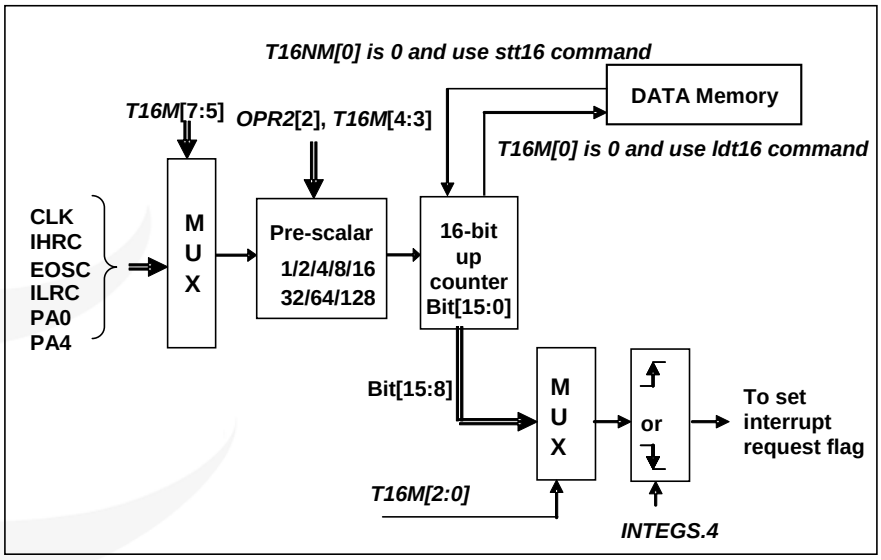


图 15：Timer16 模块框图

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      // 1st par.
$ 4~3:      /1, /4, /16, /64(OPR2[2]=0), /2, /8, /32, /128(OPR2[2]=1)      // 2nd par.
$ 2~0:      BIT8, BIT9, BIT10, BIT11, BIT12, BIT13, BIT14, BIT15      // 3rd par.
```

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;
     // stop 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 *OPR2*[2]和 *T16M* [4:3]的选项 (1, 4, 16, 32, 64, 128)
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 100：IHRC 101：保留 110：ILRC 111：PA0	
4 - 3	00	读/写	*OPR2*.2=0	Timer16 的时钟分频器。00：/1 01：/4 10：/16 11：/64
			OPR2.2=1	Timer16 的时钟分频器。00：/2 01：/8 10：/32 11：/128
2 - 0	000	读/写	中断源选择。当选择位由低变高时，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	
其中，OPR2 是选择寄存器，地址=0x3A。				

选项寄存器 2 (*OPR2*)，地址=0x3A			
位	初始值	读/写	描 述
7 – 4	-	-	保留位。请设置为 0.
3	0	只写	Timer16N 时钟预分频器选择选项。请查看 T16NM 寄存器。
2	**0**	**只写**	**Timer16 时钟预分频器选择选项。请查看 T16M 寄存器。**
1	0	只写	外部中断 1 引脚选择选项
0	0	只写	外部中断 0 引脚选择选项

10.2. 16-bit Timer (Timer16N)

10.2.1. Timer16N Introduction

PFC887 提供另一个 16 位硬件计数器 Timer16N，其模块框图如图 16。

计数器时钟源由寄存器 *T16NM*[7:5]来选择，在时钟送到 16 位计数器(counter16)之前，*T16NM*[4:3]和 *OPR2*[3]可对时钟进行预分频处理，有÷1、÷2、÷4、÷8、÷16、÷32、÷64、÷128 等 8 种选项，让计数范围更大。

当 16 位定时器计数值达到上限寄存器 *T16NBH* 和 *T16NBL* 设定的范围时，定时器将自动清除为零并产生中断请求。上限寄存器用于定义 Timer16N 的周期，使用时需要先写 *T16NBH*。

16 位计数器只能向上计数，当 *T16NM*[0]为 1 时，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。

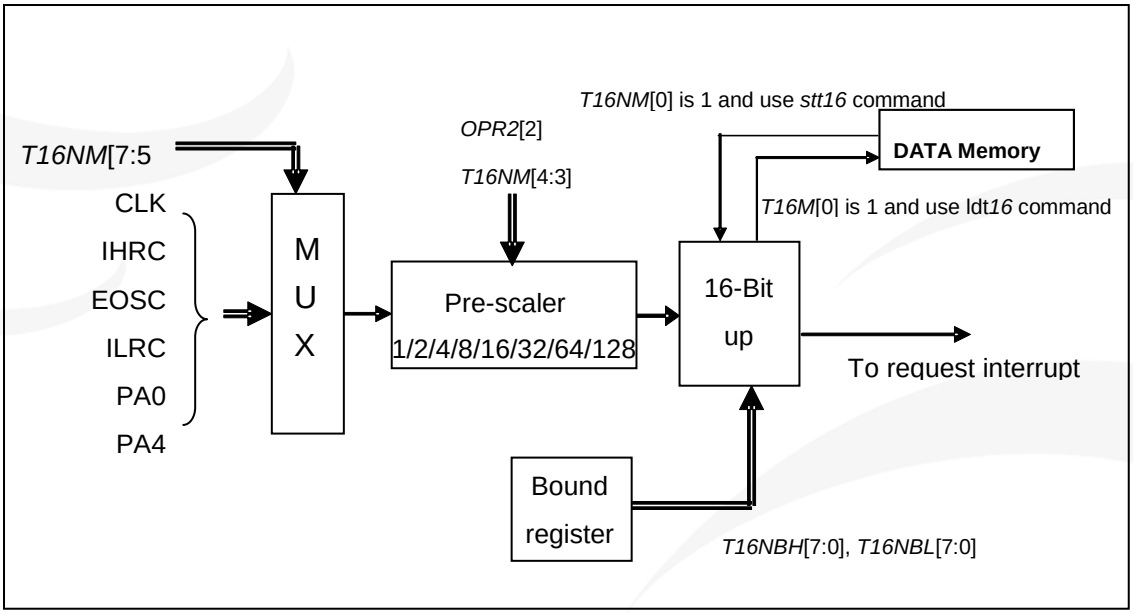


图 16：Timer16N 模块框图

选项寄存器 2，地址 = 0x3A			
位	**初始值**	**读/写**	**描述**
7 – 4	-	-	保留。请保持为 0.
3	**0**	**只写**	**Timer16N 时钟预分频器选择选项。请查看 T16NM 寄存器。**
2	0	只写	Timer16 时钟预分频器选项选择。请查看 T16M 寄存器。
1	0	只写	外部中断 1 引脚选择选项
0	0	只写	外部中断 0 引脚选择选项

10.2.2. T16N 控制寄存器 (*T16NM*)，地址=0x19

位	初始值	读/写	描 述	
7 - 5	000	读/写	Timer 时钟选择： 000：停用 Timer16N 001：CLK 系统时钟 010：保留 011：PA4 100：IHRC 101：保留 110：ILRC 111：PA0	
4 - 3	00	读/写	*OPR2*.3=0	Timer16N 时钟预分频器。 00：/1 01：/4 10：/16 11：/64
			OPR2.3=1	Timer16N 时钟预分频器。 00：/2 01：/8 10：/32 11：/128
2 - 1	00	只读	Read as 0	
0	0	读/写	ldt16/stt16 指令支持： 1：T16N 0：T16	
其中，OPR2 是选择寄存器，地址=0x3A				

10.2.3. Timer16N 上限高字节寄存器(*T16NBH*)，地址=0x73

位	初始值	读/写	描 述
7 - 0	0x00	只写	Timer16N 上限高字节寄存器。

10.2.4. Timer16N 上限低字节寄存器 (*T16NBL*)，地址 = 0x74

位	初始值	读/写	描 述
7 - 0	0x00	只写	Timer16N 上限低字节寄存器。

10.3. 8 位计数器(Timer2, Timer3)

PFC887 内置 2 个 8 位硬件计数器(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 20。

寄存器 *TM2C*[7:4]用来选择计数器时钟。利用软件编程寄存器 *TM2S* 位[6:5]，时钟预分频模块提供÷1，÷4，÷16 和÷64 的选择，另外，利用软件编程寄存器 *TM2S* 位 4:0]，时钟分频器的模块提供了÷1~÷32 的功能。在结合预分频器以及分频器，Timer2 时钟(TM2_CLK)频率可以广泛和灵活，以提供不同产品应用。寄存器 *TM2CT* 用于设置或读取计数器的计数值。

8 位 PWM 计数器只能执行 8 位上升计数操作。当 8 位计数器计数值达到上限寄存器设定的范围时，定时器将自动清除为零。上限寄存器用来定义计数器产生波形的周期。

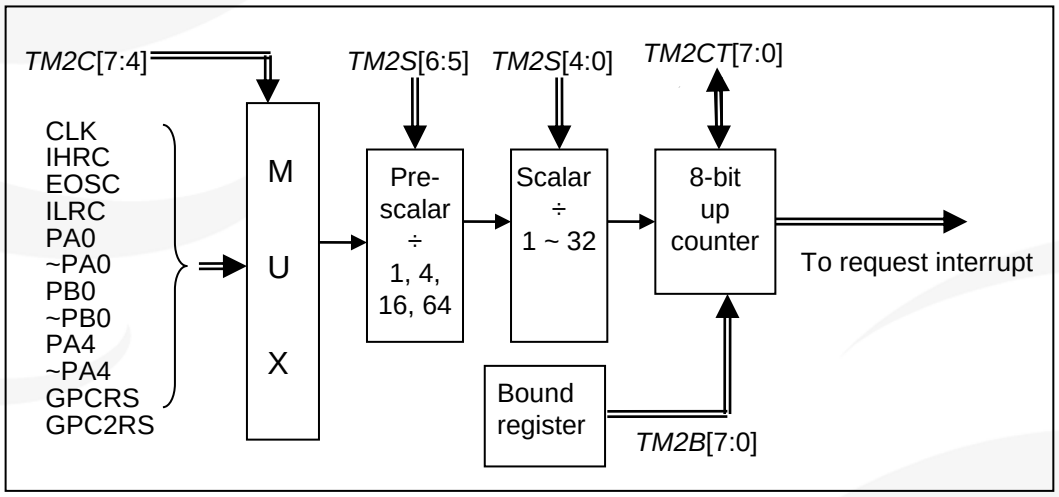


图 17：Timer2 模块框图

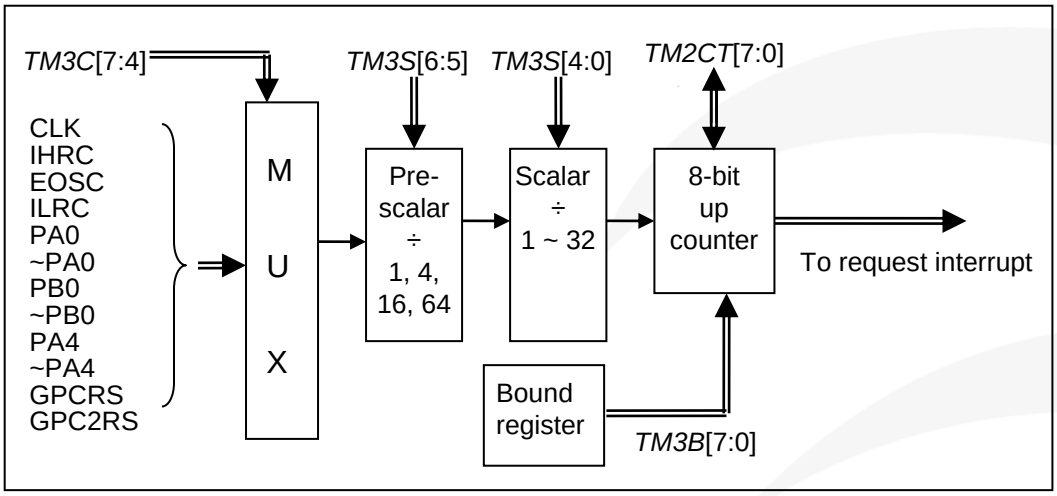


图 18：Timer3 模块框图

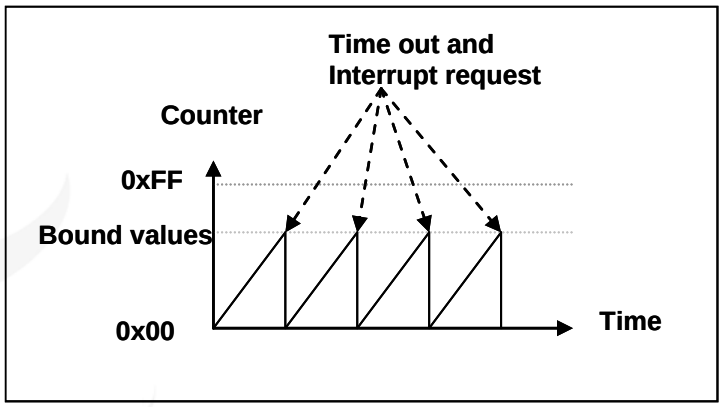


图 19：Timer2/Timer3 时序图

10.3.1. Timer2 控制寄存器(*TM2C*)，地址 = 0x24

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 0011：EOSC 0100：ILRC 0101：GPCRS 0110：GPCR2S 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）
3 - 0	-	-	保留。请保持为 0。

10.3.2. Timer2 计数寄存器(*TM2CT*)，地址 = 0x25

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

10.3.3. Timer2 分频寄存器(*TM2S*)，地址 = 0x26

位	初始值	读/写	描 述
7	-	-	保留。
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。

10.3.4. Timer2 上限寄存器(*TM2B*)，地址 = 0x27

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

10.3.5. Timer3 控制寄存器(*TM3C*)，地址 = 0x28

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 0011：EOSC 0100：ILRC 0101：GPCRS 0110：GPCR2S 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）
3 - 0	-	-	保留。请保持为 0。

10.3.6. Timer3 计数寄存器(*TM3CT*)，地址 = 0x29

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

10.3.7. Timer3 分频寄存器(*TM3S*)，地址 = 0x2A

位	初始值	读/写	描 述
7	-	-	保留。
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。

10.3.8. Timer3 上限寄存器(*TM3B*)，地址 = 0x2B

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

10.4. 12 位 PWM 产生器

PFC887 内置一个 12 位硬件 PWM 产生器，可编程设定周期和占空比。PWM 可以配置为正常模式或互补模式。时钟源可以是 IHRC 或通过 IHRCX2，通过分频可以有÷1、÷4、÷16、÷64 等四种选项。

10.4.1. PWM 波形

PWM 波形（图 20）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, $2^N \times T_{clock} = T_{Period}$）。

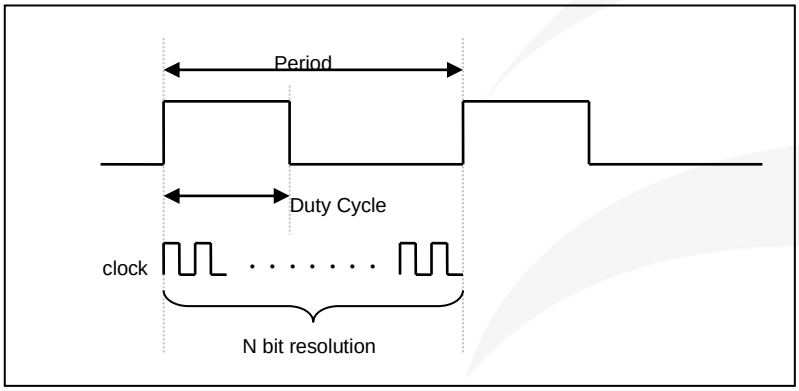


图 20：PWM 输出波形

10.4.2. 不带死区 PWM 硬件和时序框图

图 21 是 12 位 PWM 生成器的硬件框图（不带死区）。在正常模式，寄存器 *PWMGC* 用来选择其 PWM 输出端口，可以为 PA2，PA3，PA4 或 PA7。PWM 的周期由寄存器 *PWMCUBH* 和 *PWMCUBL* 决定，PWM 的占空比由寄存器 PWM 占空比高位寄存器和低位寄存器决定。

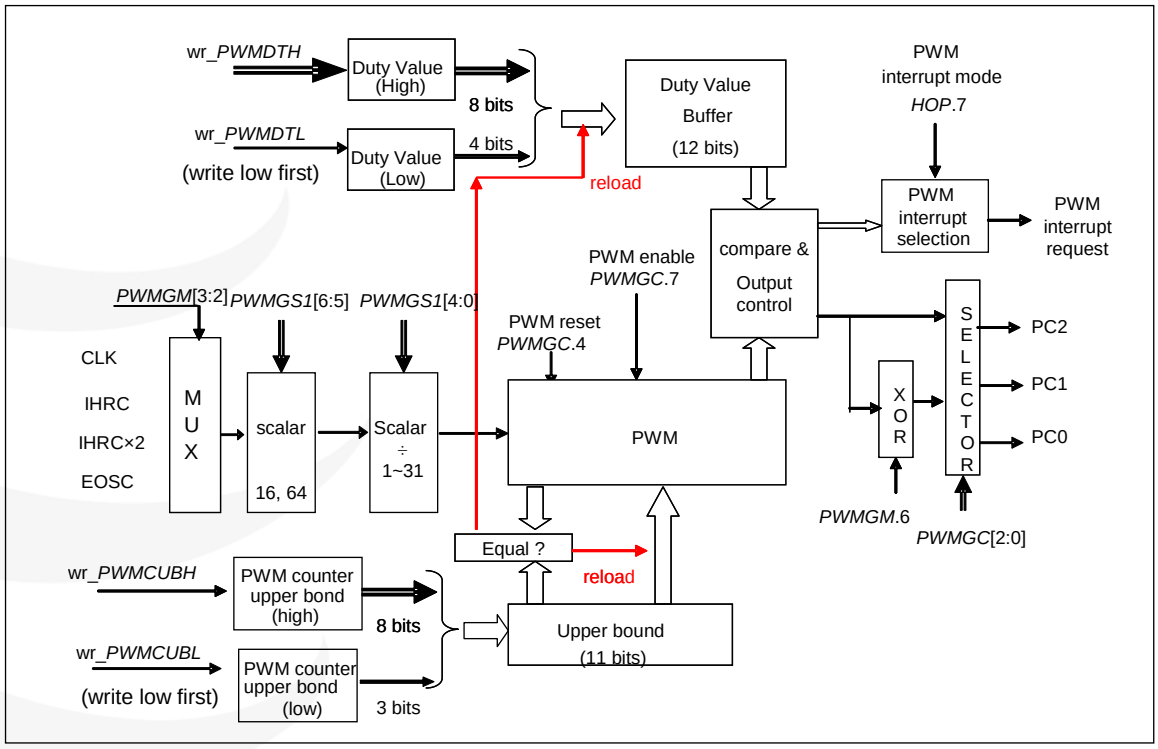


图 21：12 位 PWM 生成器硬件框图

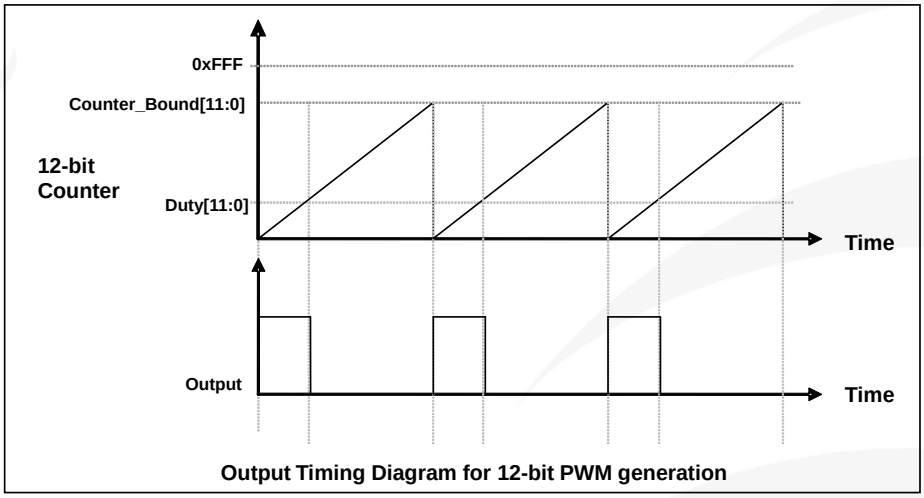


图 22：12 位 PWM 生成器的输出时序图

10.4.3. 12 位 PWM 生成器计算公式

如果 F_{IHRC} 代表 IHRC 振荡器的频率且 IHRC 被选为 12 位 PWM 的时钟源，则它的频率和占空比可由以下公式得出：

PWM 输出频率 $F_{PWM} = F_{IHRC} \div [P \times CB]$

PWM 占空比（时间）$= (1/F_{IHRC}) * [DB \div CB]$

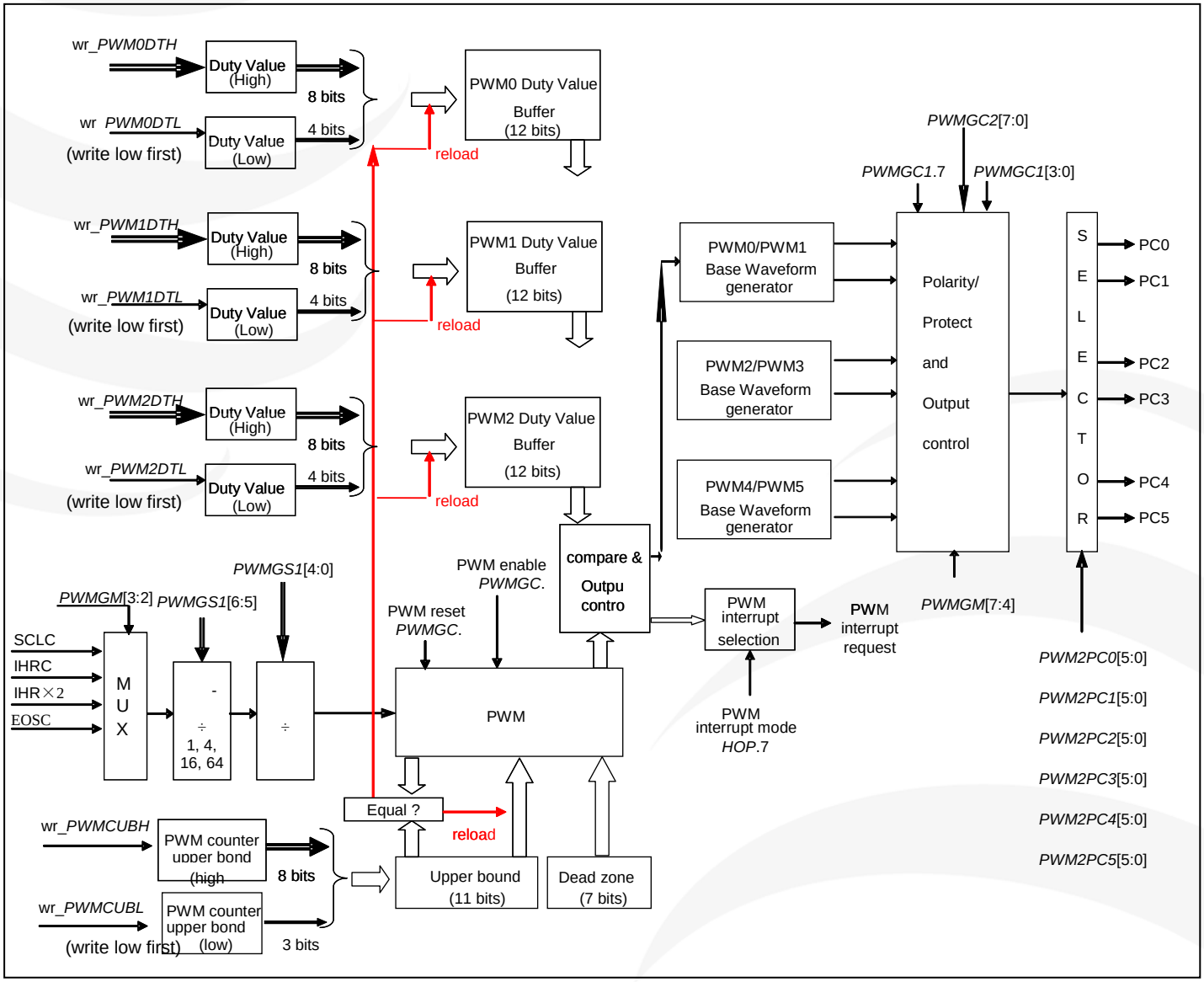
这里， *PWMGM*[1:0] = P：预分频

12-bit Duty_Bound[11:0] = {pwmdth[7:0],pwmdtl[7:4]} = **DB**;占空比

11-bit Counter_Bound[11:1] X2 = {pwmcubh[7:0], pwmcubl[7:5]} X2 = **CB**;计数器

10.4.4. 死区 PWM 硬件框图

在互补模式下，PFC887 内置一个带死区控制的 PWM 生成器，下图是它的硬件框图。互补组的输出端口可以为 PWM0/PWM1，PWM2/PWM3，PWM4/PWM5。PWM 的周期由寄存器 *PWMCUBH* 和 *PWMCUBL* 决定，PWM 的占空比由寄存器 PWM0/PWM1/PWM2 占空比高位寄存器和低位寄存器决定。



10.4.5. 12-bit PWM 生成器相关寄存器

10.4.5.1. *PWMG* 控制寄存器(*PWMGC*)，地址= 0x30

位	初始值	读/写	描 述
7	0	读/写	启用 PWMG。0 / 1：停用 / 启用
6	-	只读	PWM0 生成器输出状态
5	0	读/写	PWM 排列模式 0：边排列模式 1：中心排列模式
4	0	读/写	PWM 计数器清零。 写“1”清零 PWM 计数，清零 PWM 计数后，这个位会自动归 0。
3	-	-	
2	0	读/写	PWM0 输出到 PC2 1：启用 0：停用
1	0	读/写	PWM0 输出到 PC1 1：启用 0：停用
0	0	读/写	PWM0 输出到 PC0 1：启用 0：停用

10.4.5.2. PWMG 分频寄存器(*PWMGM*)，地址 = 0x31

位	初始值	读/写	描 述
7	0	只写	选择上臂 PWM 输出的结果是否反极性。0/1：停用/启用
6	0	只写	选择下臂 PWM 输出的结果是否反极性。0/1：停用/启用
5	0	只写	臂保护选择。0/1：停用/启用
4	0	只写	臂保护模式。 0：上臂高电位时执行臂保护 1：上臂低电位时执行臂保护
3 - 2	00	只写	PWM 生成器时钟源 00：系统时钟 01：IHRC 10：IHRC*2 11：EOSC
1 - 0	-	-	

10.4.5.3. PWM 发生器标量寄存器 (*PWMGS1*), 地址= 0x59

位	初始值	读/写	描述
7	-	-	
6 - 5	00	WO	PWM 发生器时钟预标量 00 : /1 01 : /4 10 : /16 11 : /64
4 - 0	00	WO	PWMG0 clock = (标量+ 1) * PWMG0 分割 clock

10.4.5.4. PWMG 控制寄存器 1(*PWMGC1*)，地址 = 0x37

位	初始值	读/写	描 述
7	0	读/写	启用互补 PWM0~PWM5 生成器输出。0/1：停用/启用
6 - 5	-	只写	01：保留 00/10/11：PWM 生成器支持三相 BLDC
4	0	只读	读值为 0
3 - 2	00	读/写	PWM5 输出类型 00：强制输出低电位 01：PWM+ 10：PWM- 11：强制输出高电位
1 - 0	00	读/写	PWM4 输出类型 00：强制输出低电位 01：PWM+ 10：PWM- 11：强制输出高电位

10.4.5.5. PWMG 控制寄存器 2*(PWMGC2)*，地址 = 0x38

位	初始值	读/写	描 述
7 - 6	00	读/写	PWM3 输出类型。 00：强制输出低电位 01：PWM+ 10：PWM- 11：强制输出高电位
5 - 4	00	读/写	PWM2 输出类型。 00：强制输出低电位 01：PWM+ 10：PWM- 11：强制输出高电位
3 - 2	00	读/写	PWM1 输出类型。 00：强制输出低电位 01：PWM+ 10：PWM- 11：强制输出高电位
1 - 0	00	读/写	PWM0 输出类型。 00：强制输出低电位 01：PWM+ 10：PWM- 11：强制输出高电位

10.4.5.6. PWM 计数上限高位寄存器(*PWMCUBH*)，地址 = 0x50

位	初始值	读/写	描 述
7 - 0	8'h00	只写	PWM 上限寄存器位[11:4] 。

10.4.5.7. PWM 计数上限低位寄存器(*PWMCUBL*)，地址= 0x51

位	初始值	读/写	描 述
7 - 5	3'h0	只写	PWM 上限寄存器位[3:1]
4 - 0	-	-	保留

10.4.5.8. PWM0 占空比高位寄存器 (*PWM0DTH*)，地址 = 0x52

位	初始值	读/写	描 述
7 - 0	8'h00	只写	PWM0 生成器占空比值位[11:4]

10.4.5.9. PWM0 占空比低位寄存器 (*PWM0DTL*)，地址 = 0x53

位	初始值	读/写	描 述
7 - 4	4'h0	只写	PWM0 生成器占空比值位[3:0]
3 - 0	-	-	保留

10.4.5.10. PWM1 占空比高位寄存器(*PWM1DTH*)，地址 = 0x54

位	初始值	读/写	描 述
7 - 0	8'h00	只写	PWM1 生成器占空比值位[11:4]

10.4.5.11. *PWM1* 占空比低位寄存器(*PWM1DTL*)，地址 = 0x55

位	初始值	读/写	描 述
7 - 4	4'h0	只写	PWM1 生成器占空比值位[3:0]
3 - 0	-	-	保留

10.4.5.12. *PWM2* 占空比高位寄存器 (*PWM2DTH*)，地址 = 0x56

位	初始值	读/写	描 述
7 - 0	8'h00	只写	PWM2 生成器占空比值位[11:4]

10.4.5.13. *PWM2* 占空比低位寄存器 (*PWM2DTL*)，地址 = 0x57

位	初始值	读/写	描 述
7 - 4	4'h0	只写	PWM1 生成器占空比值位[3:0]
3 - 0	-	-	保留

10.4.5.14. PWM 死区寄存器 (*PWMDZ*)，地址 = 0x58

位	初始值	读/写	描 述
7 - 1	-	只写	PWM 生成器死区值位[7:1]
0	-	-	保留

10.4.5.15. PWM 发生器输出引脚 PC0 寄存器 (*PWM2PC0*), 地址= 0x40

位	初始值	读/写	描 述
5	0	读/写	PWM5 输出到引脚 PC0 0：停用 1：启用
4	0	读/写	PWM4 输出到引脚 PC0 0：停用 1：启用
3	0	读/写	PWM3 输出到引脚 PC0 0：停用 1：启用
2	0	读/写	PWM2 输出到引脚 PC0 0：停用 1：启用
1	1	读/写	PWM1 输出到引脚 PC0 0：停用 1：启用
0	0	读/写	PWM0 输出到引脚 PC0 0：停用 1：启用

10.4.5.16. PWM 发生器输出引脚 PC1 寄存器 (*PWM2PC1*), 地址= 0x41

位	初始值	读/写	描 述
5	0	读/写	PWM5 输出到引脚 PC1 0：停用 1：启用
4	0	读/写	PWM4 输出到引脚 PC1 0：停用 1：启用
3	0	读/写	PWM3 输出到引脚 PC1 0：停用 1：启用
2	0	读/写	PWM2 输出到引脚 PC1 0：停用 1：启用
1	1	读/写	PWM1 输出到引脚 PC1 0：停用 1：启用
0	0	读/写	PWM0 输出到引脚 PC1 0：停用 1：启用

10.4.5.17. PWM 发生器输出引脚 PC2 寄存器 (*PWM2PC2*), 地址= 0x42

位	初始值	读/写	描 述
5	0	读/写	PWM5 输出到引脚 PC2 0：停用 1：启用
4	0	读/写	PWM4 输出到引脚 PC2 0：停用 1：启用
3	0	读/写	PWM3 输出到引脚 PC2 0：停用 1：启用
2	0	读/写	PWM2 输出到引脚 PC2 0：停用 1：启用
1	1	读/写	PWM1 输出到引脚 PC2 0：停用 1：启用
0	0	读/写	PWM0 输出到引脚 PC2 0：停用 1：启用

10.4.5.18. PWM 发生器输出引脚 PC3 寄存器 (*PWM2PC3*), 地址= 0x43

位	初始值	读/写	描 述
5	0	读/写	PWM5 输出到引脚 PC3 0：停用 1：启用
4	0	读/写	PWM4 输出到引脚 PC3 0：停用 1：启用
3	0	读/写	PWM3 输出到引脚 PC3 0：停用 1：启用
2	0	读/写	PWM2 输出到引脚 PC3 0：停用 1：启用
1	1	读/写	PWM1 输出到引脚 PC3 0：停用 1：启用
0	0	读/写	PWM0 输出到引脚 PC3 0：停用 1：启用

10.4.5.19. PWM 发生器输出引脚 PC4 寄存器 (*PWM2PC4*), 地址= 0x5a

位	初始值	读/写	描 述
5	0	读/写	PWM5 输出到引脚 PC4 0：停用 1：启用
4	0	读/写	PWM4 输出到引脚 PC4 0：停用 1：启用
3	0	读/写	PWM3 输出到引脚 PC4 0：停用 1：启用
2	0	读/写	PWM2 输出到引脚 PC4 0：停用 1：启用
1	1	读/写	PWM1 输出到引脚 PC4 0：停用 1：启用
0	0	读/写	PWM0 输出到引脚 PC4 0：停用 1：启用

10.4.5.20. PWM 发生器输出引脚 PC5 寄存器 (*PWM2PC5*), 地址= 0x5b

位	初始值	读/写	描 述
5	0	读/写	PWM5 输出到引脚 PC5 0：停用 1：启用
4	0	读/写	PWM4 输出到引脚 PC5 0：停用 1：启用
3	0	读/写	PWM3 输出到引脚 PC5 0：停用 1：启用
2	0	读/写	PWM2 输出到引脚 PC5 0：停用 1：启用
1	1	读/写	PWM1 输出到引脚 PC5 0：停用 1：启用
0	0	读/写	PWM0 输出到引脚 PC5 0：停用 1：启用

11. 特殊功能

11.1. 通用比较器

11.1.1. 通用比较器硬件框图

PFC887 内置两个硬件比较器，硬件框图如图 23。它可以比较两个输入端之间的信号大小或者与内部参考电压 $V_{internal\ R}$ 或者与内置 band-gap(1.2V)作比较。进行比较的两个信号，一个是正输入，另一个是负输入。对于通用比较器 1，正输入由寄存器 *GPC1C*.0 选择，负输入由 *GPC1C*[3:1] 选择；对于通用比较器 2，正输入由寄存器 *GPC2C*.0 选择，负输入由 *GPC2C*[3:1] 选择。对于通用比较器 3，正输入由寄存器 *GPC3C*.0 选择，负输入由 *GPC3C*[3:1] 选择。

比较器输出的结果可以：

(1) 由 *GPC1C*.6、*GPC2C*.6 或 *GPC3C*.6 读取出来；

(2) 由 *GPC1C*.4、*GPC2C*.4 或 *GPC3C*.4 选择输出信号是否反极性；

(3) 选择是否由 Time2(TM2_CLK)或 Timer3(TM3_CLK)的上升沿采样输出；

(4) 由 *GPC1S*.7 / *GPC2S*.7 / *GPC3S*.7 选择是否输出到 PB0 / PA2 / PA0；

(5) 由 INTEGS2[1:0] / INTEGS2[3:2] / INTEGS2[5:4]产生中断信号。

上电复位后比较器处于停用状态，通过设置 *GPC1C*.7=1，*GPC2C*.7=1 或 *GPC31C*.7=1 可启用比较器。只有当执行 *stopsys* 指令时，比较器模块才会进入掉电模式，此时 PFC887 也处于掉电状态。

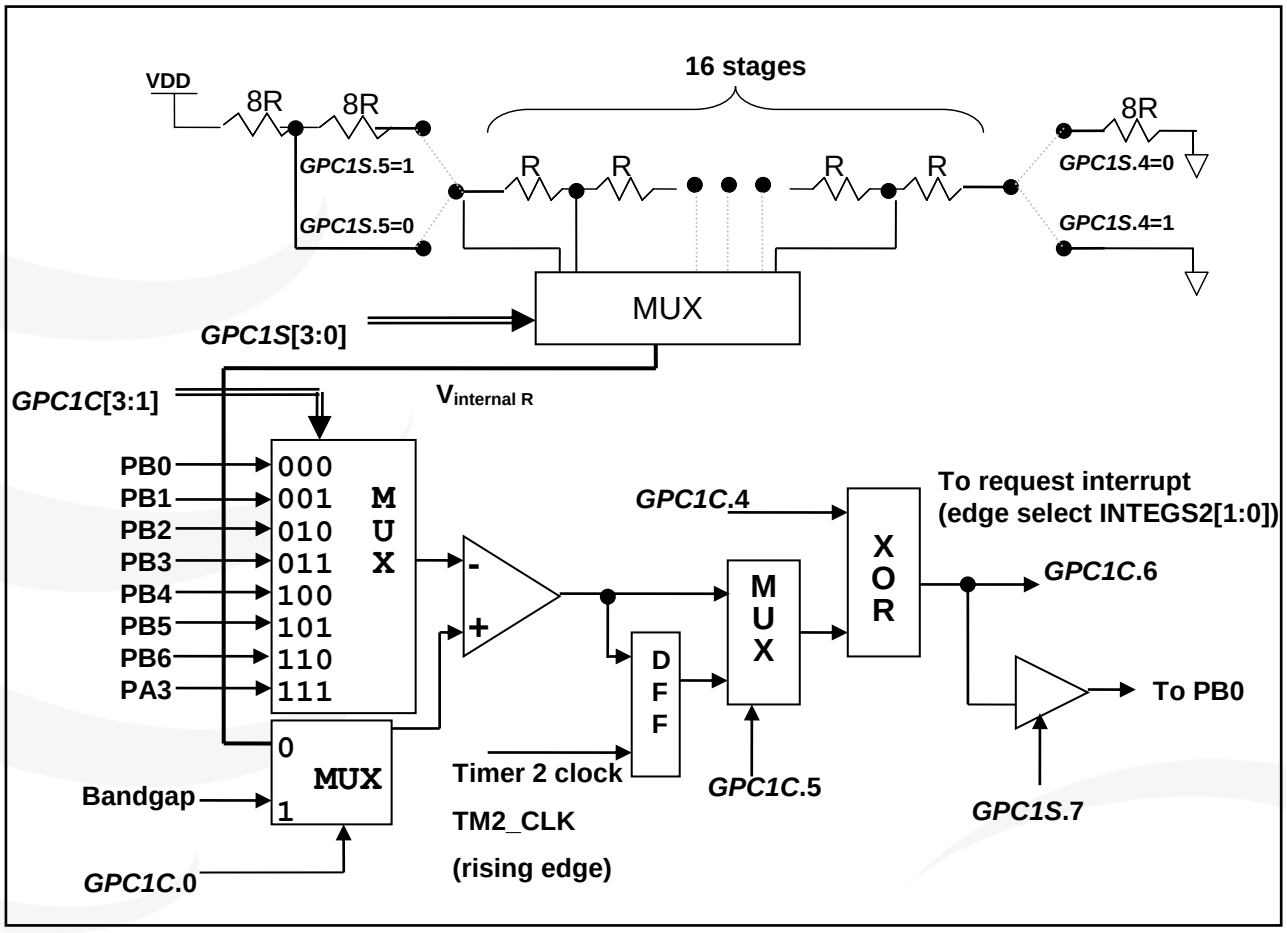


图 23：比较器 1 硬件框图

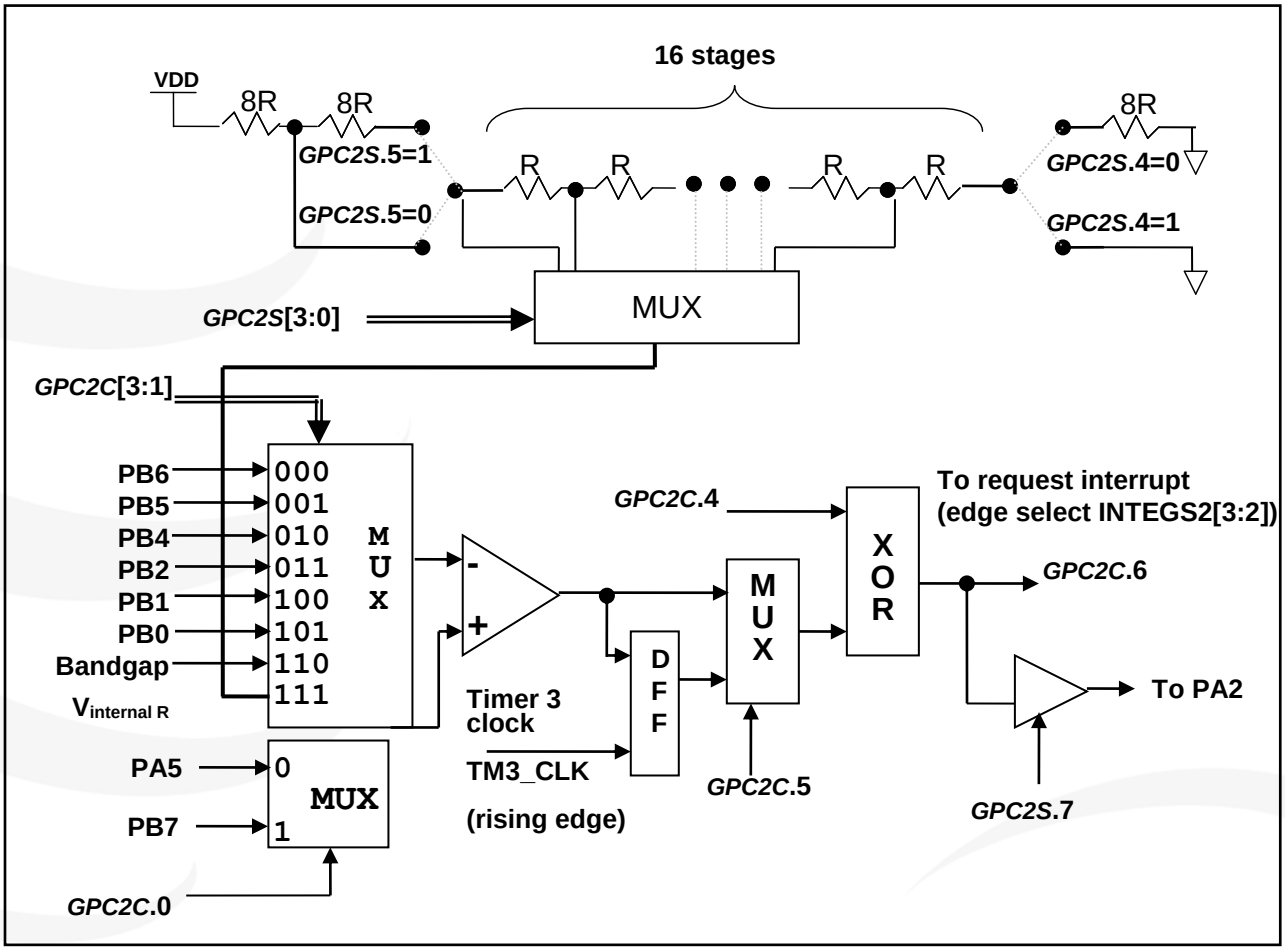


图 24：比较器 2 硬件框图

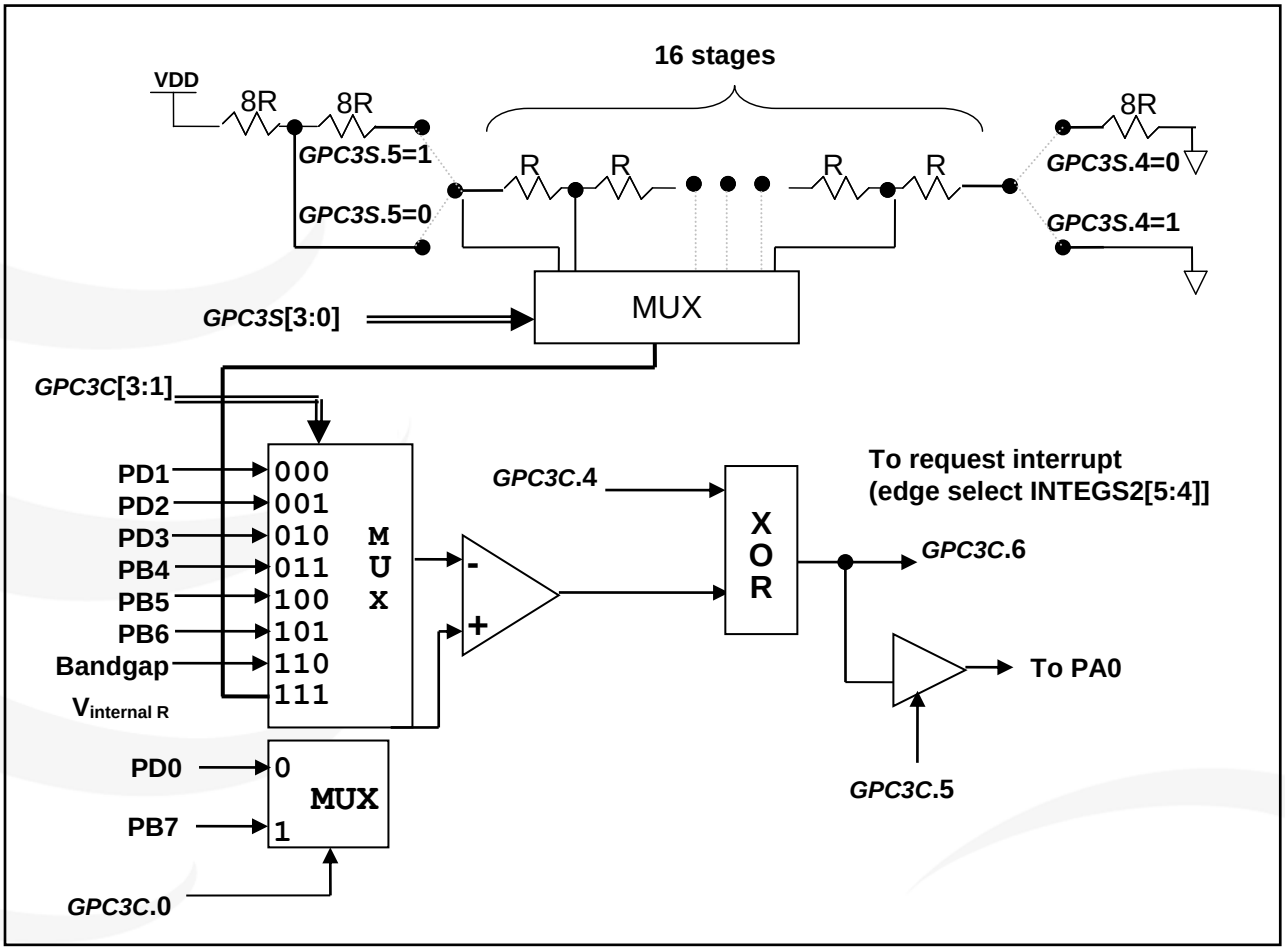


图 25：比较器 3 硬件框图

11.1.2. 通用比较器 1 控制寄存器(*GPC1C*), 地址= 0x34

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	选择通用比较器负输入的来源。 000：PB0 001：PB1 010：PB2 011：PB3 100：PB4 101：PB5 110：PB6 111：PA3
0	0	读/写	选择比较器正输入的来源。 0：$V_{internal\ R}$ 1：内部 1.20 V Bandgap 参考电压

11.1.3. 通用比较器 1 选择寄存器(*GPC1S*), 地址= 0x35

位	初始值	读/写	描 述
7	0	只写	比较器 1 输出启用（到 PB0） 0 / 1：停用/启用
6	0	只写	启用通用比较器 1 唤醒系统功能功能 0 / 1：停用/启用 当比较器结果变为 1 时，将从掉电模式唤醒系统。
5	0	只写	选择比较器参考电压 Vinternal 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）

11.1.4. 通用比较器 2 控制寄存器 (*GPC2C*), 地址= 0x1E

位	初始值	读/写	描 述
7	0	读/写	启用比较器 2。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 2 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 2 的结果是否由 TM3_CLK 采样输出 0：比较器的结果没有 TM3_CLK 采样输出 1：比较器的结果是由 TM3_CLK 采样输出
4	0	读/写	选择比较器 2 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 2 负输入的来源 000：PB6 001：PB5 010：PB4 011：PB2 100：PB1 101：PB0 110：内部 1.20 V Bandgap 参考电压 111：$V_{internal\ R}$
0	0	读/写	选择比较器 2 正输入的来源 0：PA5 1：PB7

11.1.5. 通用比较器 2 选择寄存器(*GPC2S*)，地址 = 0x1F

位	初始值	读/写	描 述
7	0	只写	比较器 2 输出启用（到 PA2） 0 / 1：停用/启用
6	0	只写	启用通用比较器 2 唤醒系统功能 0 / 1：停用/启用 比较器结果变为 1 时，将从掉电模式唤醒系统
5	0	只写	选择比较器 2 参考电压 $V_{internal\ R}$ 最高的范围
4	0	只写	选择比较器 2 参考电压 $V_{internal\ R}$ 最低的范围
3 - 0	0000	只写	选择比较器 2 参考电压 $V_{internal\ R}$ 0000（最低）~ 1111（最高）

11.1.6. 通用比较器 3 控制寄存器 (*GPC3C*), 地址= 0x22

位	初始值	读/写	描 述
7	0	读/写	启用比较器 3 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 3 结果 0：正输入 < 负输入 1：正输入 > 负输入
5	0	读/写	比较器 3 输出启用（到 PA0） 0 / 1 : disable / enable
4	0	读/写	选择比较器 3 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 3 负输入的来源。 000：PD1 001：PD2 010：PD3 011：PB4 100：PB5 101：PB6 110：内部 1.20 V Bandgap 参考电压 111：$V_{internal\ R}$
0	0	读/写	选择比较器 3 正输入的来源。 0：PD0 1：PB7

11.1.7. 通用比较器 3 选择寄存器(*GPC3S*)，地址 = 0x78

位	初始值	读/写	描 述
7 - 6	-	-	
5	0	只写	选择比较器 3 参考电压 $V_{internal\ R}$ 最高的范围。
4	0	只写	选择比较器 3 参考电压 $V_{internal\ R}$ 最低的范围。
3 - 0	0000	只写	选择比较器 3 参考电压 $V_{internal\ R}$。 0000（最低）~ 1111（最高）

11.1.8. 模拟讯号输入

该模拟输入的简化电路如图 26 所示。比较器所有的模拟输入引脚与数字讯号输入共享，而数字讯号输入有 ESD 二极管反向偏置到 VDD 和 GND，因此比较器的模拟输入信号必须在 VDD 和 GND 之间。

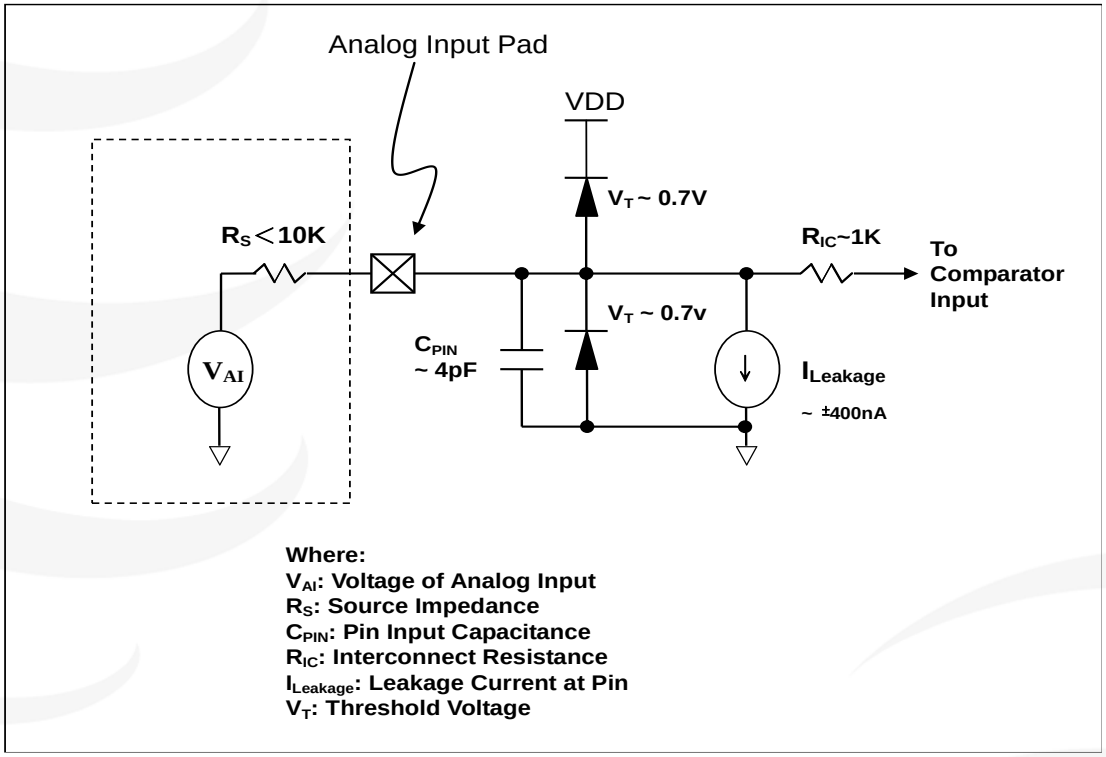


图 26：通用比较器模拟输入引脚模型

11.1.9. 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 *GPC1S/GPC2S/GPC3S*（以下简称 *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；*GPCS* 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。

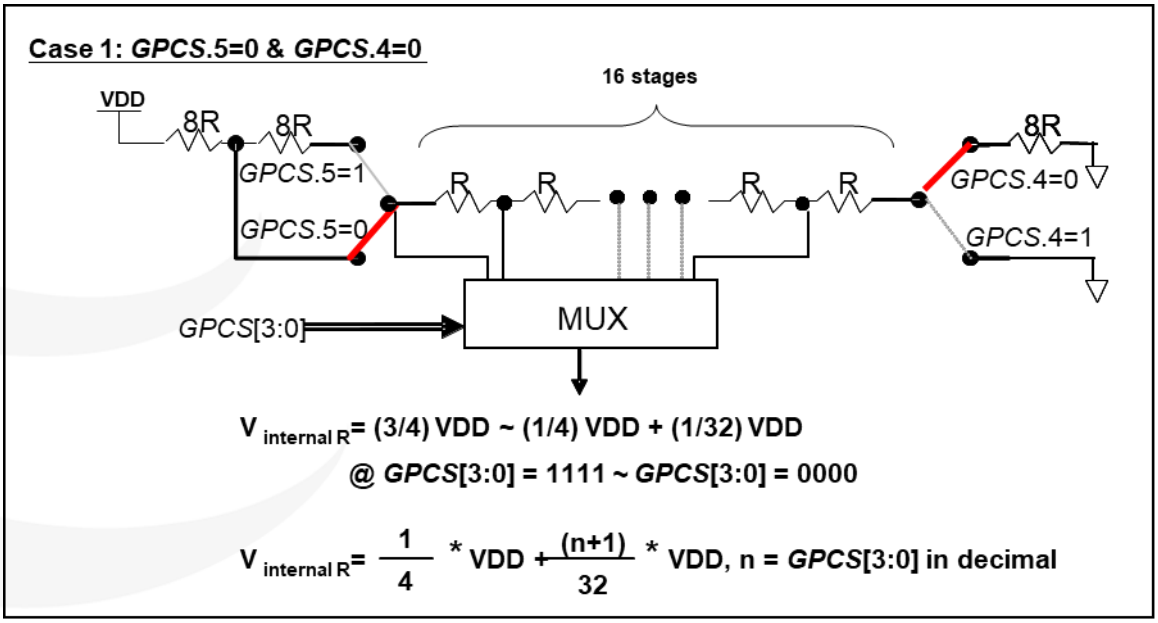


图 27：$V_{internal\ R}$ 硬件接法 (*GPCS*.5=0 & *GPCS*.4=0)

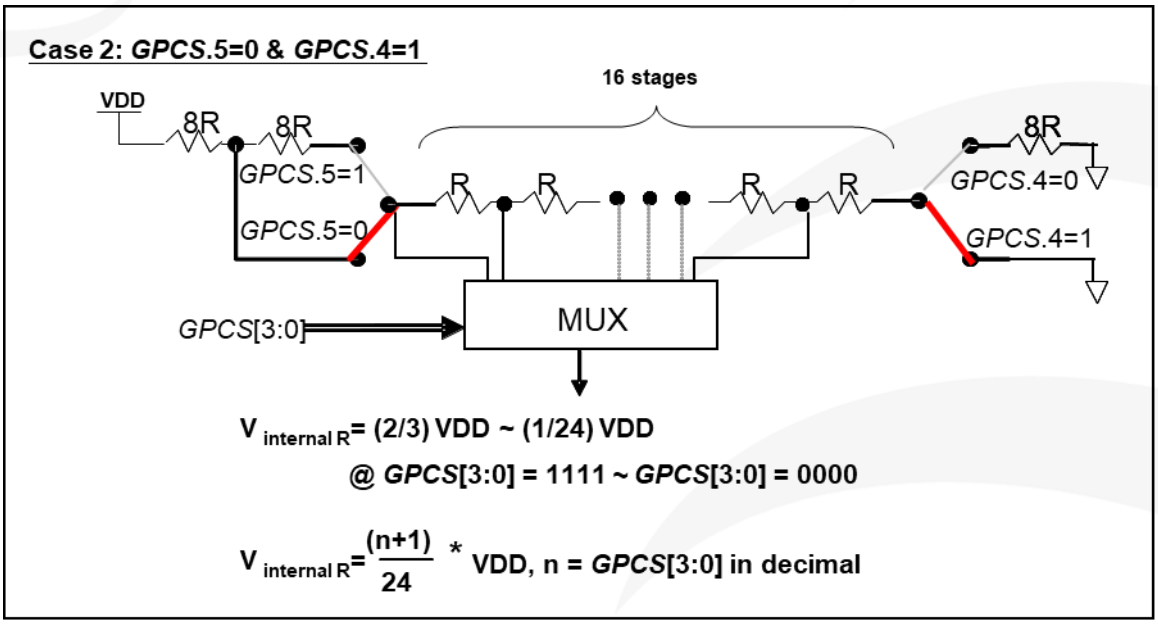


图 28：$V_{internal\ R}$ 硬件接法 (*GPCS*.5=0 & *GPCS*.4=1)

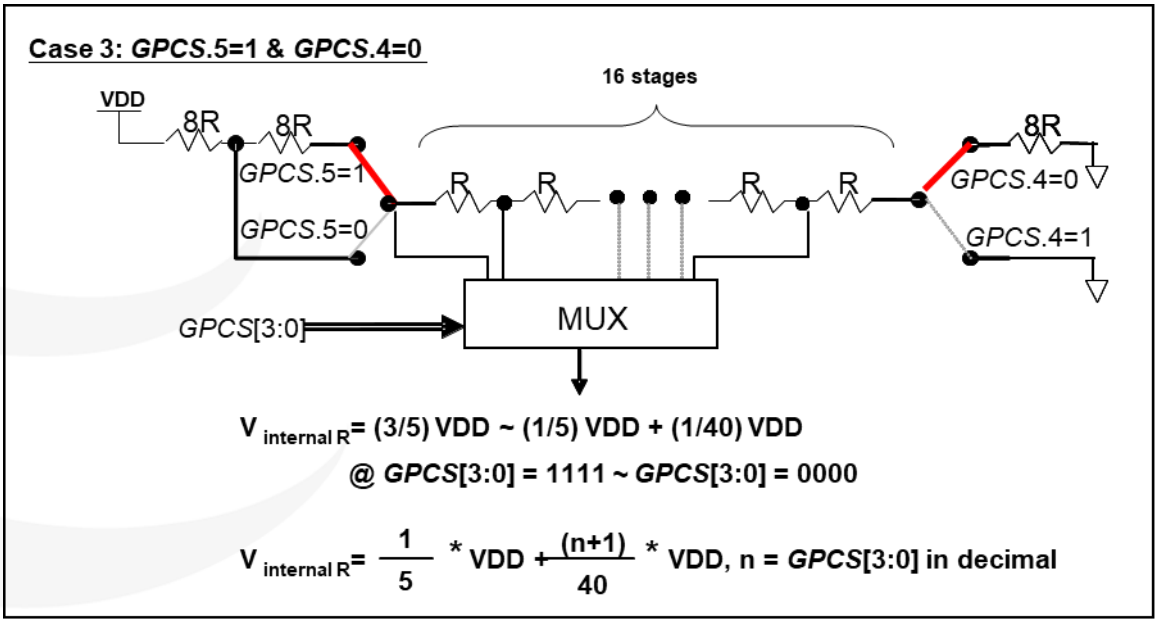


图 29：$V_{internal\ R}$ 硬件接法 (*GPCS*.5=1 & *GPCS*.4=0)

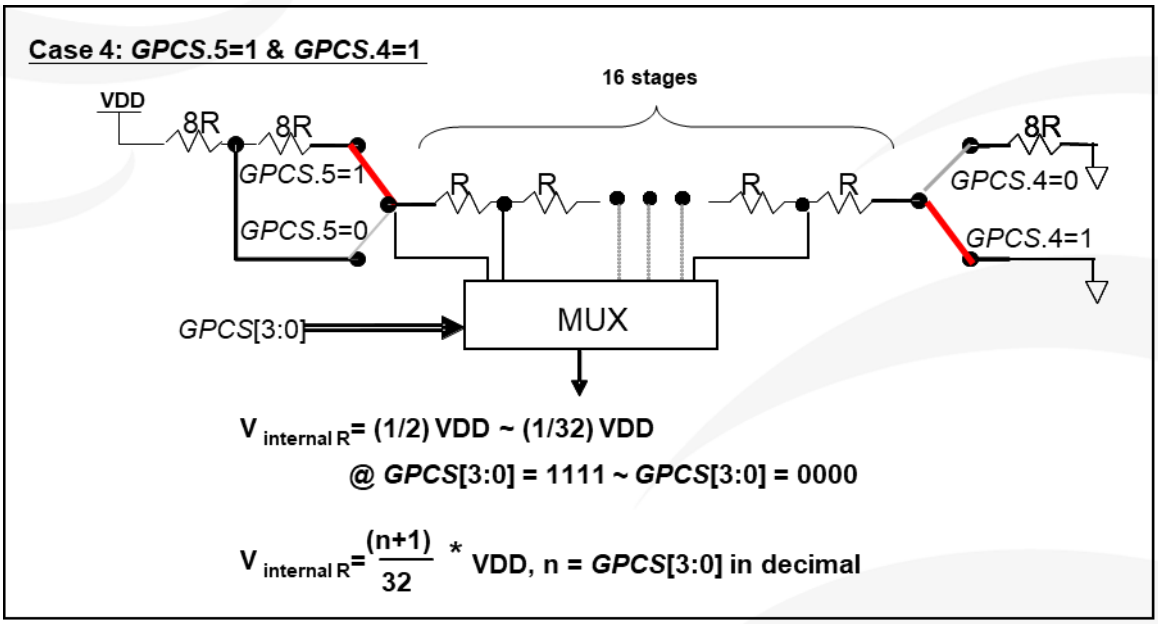


图 30：$V_{internal\ R}$ 硬件接法 (*GPCS*.5=1 & *GPCS*.4=1)

11.1.10. 比较器 1 输出同步到 Timer2

通过设置 *GPC1C*.5=1，可以将比较器的输出同步到 Timer2。当启用该功能时，比较器的输出是由 Timer2 时钟源（TM2_CLK）的上升沿采样。请参考 Timer2 的硬件框图和比较器 1 的硬件框图，如果 TM2 的时钟使用预分频器处理，则 TM2_CLK 是经过分频和预分频的时钟源，将用于 TM2 的定时器计数和比较器采样。

11.1.11. 比较器 2 输出同步到 Timer3

通过设置 *GPC2C*.5=1，可以将比较器的输出同步到 Timer3。当启用该功能时，比较器的输出是由定时器 3 的时钟源（TM3_CLK）的上升沿采样。请参考 Timer3 的硬件框图和比较器 2 的硬件框图，如果 TM3 的时钟使用预分频器处理，则 TM3_CLK 是经过分频和预分频的时钟源，将用于 TM3 的定时器计数和比较器采样。

11.1.12. 使用比较器 1

例 1：

选择 PB6 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 *GPC1S*[5:4] = 2b'00 的配置方式，*GPC1S*[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。

```
GPC1S   = 0b1_0_00_1001;        // 输出到 PB0，Vinternal R = VDD*(18/32)
GPC1C   = 0b1_0_0_0_110_0;      // 启用比较器，负输入：PB6，正输入：Vinternal R
PBDIER  = 0bx_0_xx_xxxx;        // 停用 PB6 数字输入防止漏电（x：由客户自定）
```

例 2：

选择 $V_{internal\ R}$ 为正输入，$V_{internal\ R}$ 的电压为(22/40)*VDD，选择 PB0 为负输入，比较器的结果反极性并不输出到 PB0。$V_{internal\ R}$ 选择上图的配置方式 "*GPC1S*[5:4] = 2b'10" 和 *GPC1S*[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。

```
GPC1S   = 0b0_0_10_1101;        // Vinternal R = VDD*(22/40)
GPC1C   = 0b1_0_0_1_000_0;      // 反极性输出，正输入=Vinternal R，负输入=PB0
PBDIER  = 0bxxxx_xxx_0;         // 停用 PB0 数字输入防止漏电（x：由客户自定）
```

11.2. 8X 运算放大器(OPA)模块

PFC887 内置一个运算放大器(OPA)模块，其基本配置如下图所示。

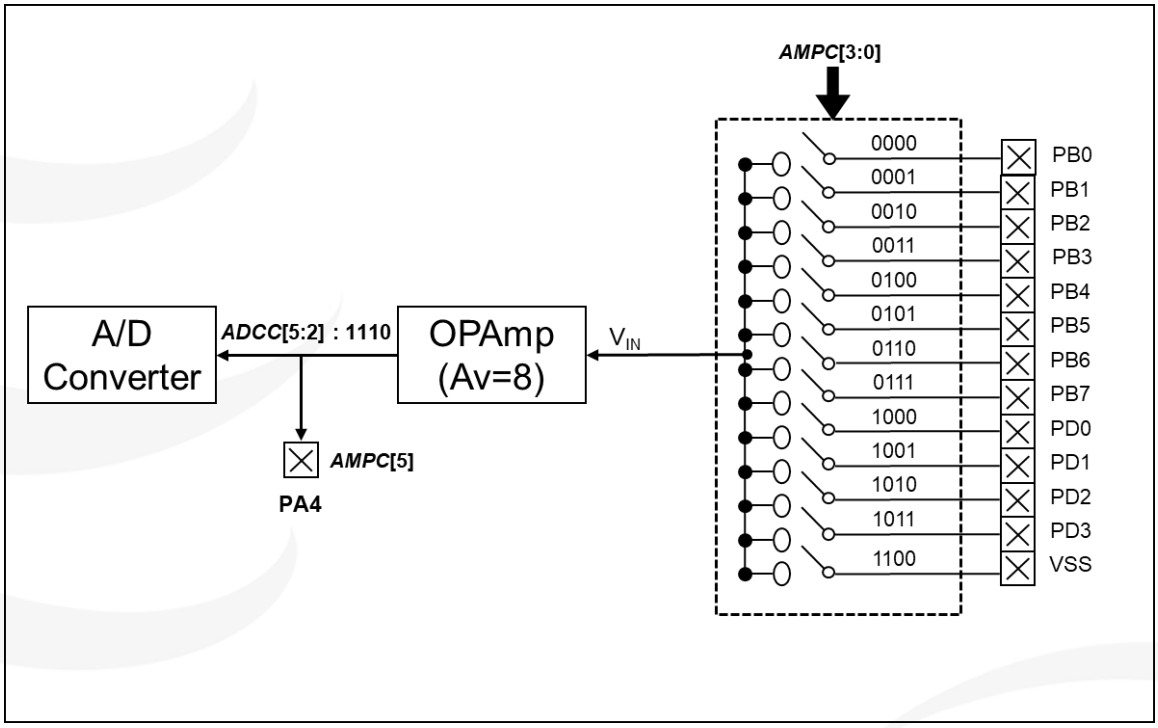


图 31：OPA 硬件框图

PFC887 提供一个单端输入、单端输出 OPA 模块，该模块具有 12 个外部通道和一个用于内部 VSS 参考电压的内部通道。该 OPA 模块处理传感器单端输入的小信号，放大 8 倍至 ADC 模块或 PA6。在使用 OPA 之前，需要使用 VSS 通道记录偏移电压，然后切换到另一个通道产生放大信号。

建议在 VDD 和 GND 之间并联一个大于 1uF 电容和一个 0.1uF 的电容，以获得更好的 OPA 特性。

11.2.1. 配置模拟输入引脚

OPA 的 12 个外部模拟输入信号与端口 PB0、PB1、PB2、PB3、PB4、PB5、PB6、PB7、PD0、PD1、PD2、PD3 共享引脚。为了避免在设置为数字电路时发生漏电流，这些引脚在当模拟输入时一定要利用寄存器 *PBDIER* / *PDDIER* 设置为模拟输入（设置 *PBDIER* / *PDDIER* 寄存器的相应位为 0）。OPA 的测量信号属于小信号，为避免测试信号在测量期间被干扰，被选定的引 脚应该：

(1) 设置为输入模式

(2) 关闭弱上拉电阻高寄存器

(3) 利用 *PBDIER* / *PDDIER* 寄存器配置所选定的引脚为模拟输入。

建议在模拟输入通道上连接一个 RC 滤波电路（100 ohm 的电阻和一个 1nF 的电容），以获得更好的 OPA 特性。

11.2.2. OPA 控制寄存器 (*AMPC*)，地址 = 0x18

位	初始值	读/写	描 述
7	0	只写	支持 Bandgap 和 LVR 硬件模块掉电模式。 0/1：正常/掉电模式
6	1	读/写	ADC 内部参考高电压选择。 1：VDD 0：1.2V
5	0	读/写	OPA 输出到 PA6。 1：启用 0：停用
4	0	读/写	读 0
3 - 0	00	读/写	OPA 通道选择。这两位用于选择 OPA 的输入信号引脚。 0000: PB0, 0001: PB1, 0010: PB2, 0011: PB3, 0100: PB4, 0101: PB5, 0110: PB6, 0111: PB7, 1000: PD0, 1001: PD1, 1010: PD2, 1011: PD3, 1100: VSS

11.3. 模拟-数字转换器 (ADC) 模块

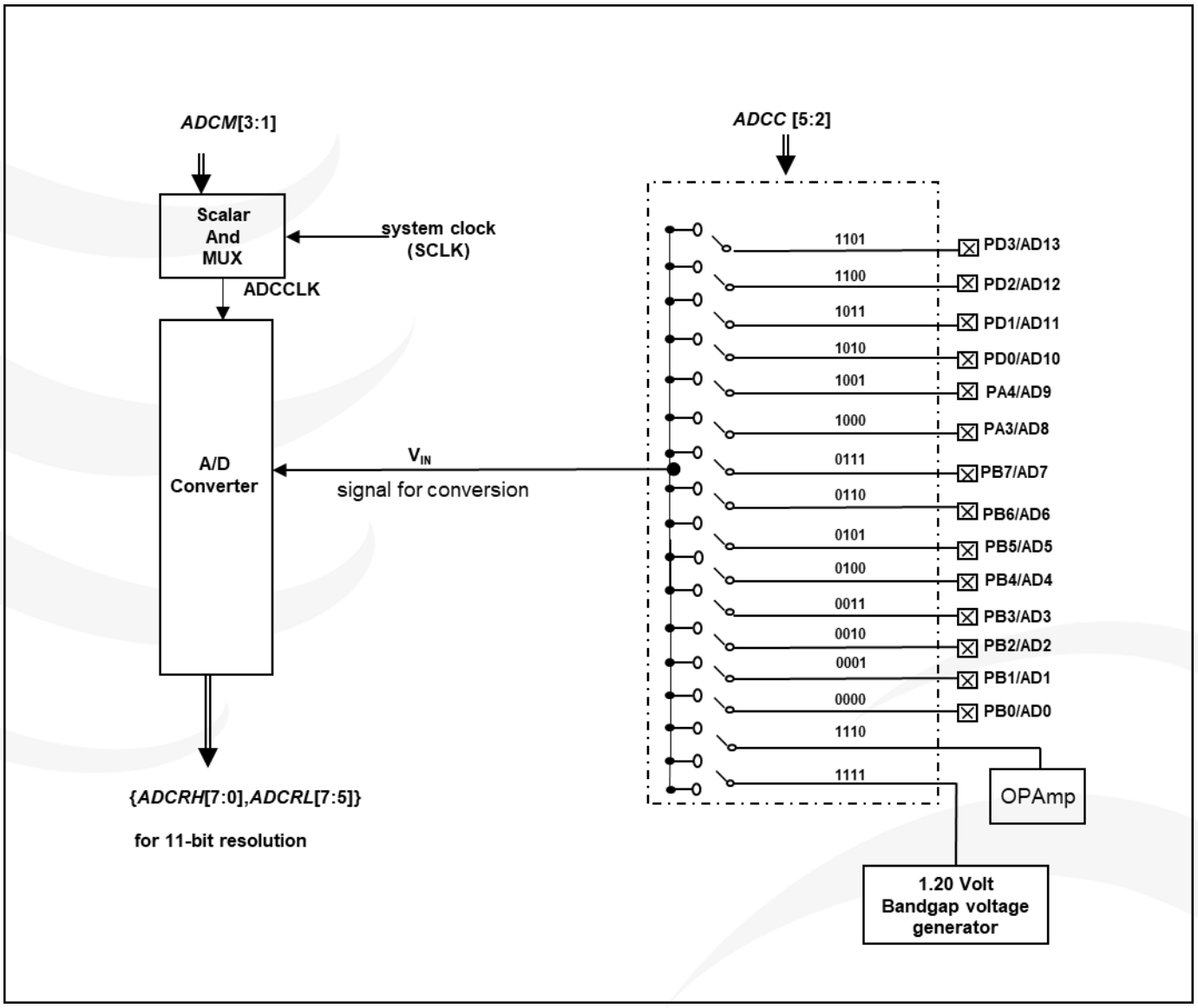


图 32：ADC 模块框图

PFC887 提供一个 11 位分辨率的模拟-数字转换模块，该模块具有 14 个外部通道和 2 个内部通道。内部通道分别来源于 1.20V Bandgap 和 OPAmp 的放大信号。

在转换过程中，首先对模拟信号进行采样和保持，然后通过逐次逼近的算法将模拟信号送入转换器。ADC 的模拟参考高电压为正电源电压(VDD)，参考低电压为 GND。

建议在 VDD 和 GND 之间连接大于 1uF 的电容，以获得更稳定的 AD 转换结果。

11.3.1. 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 33 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。

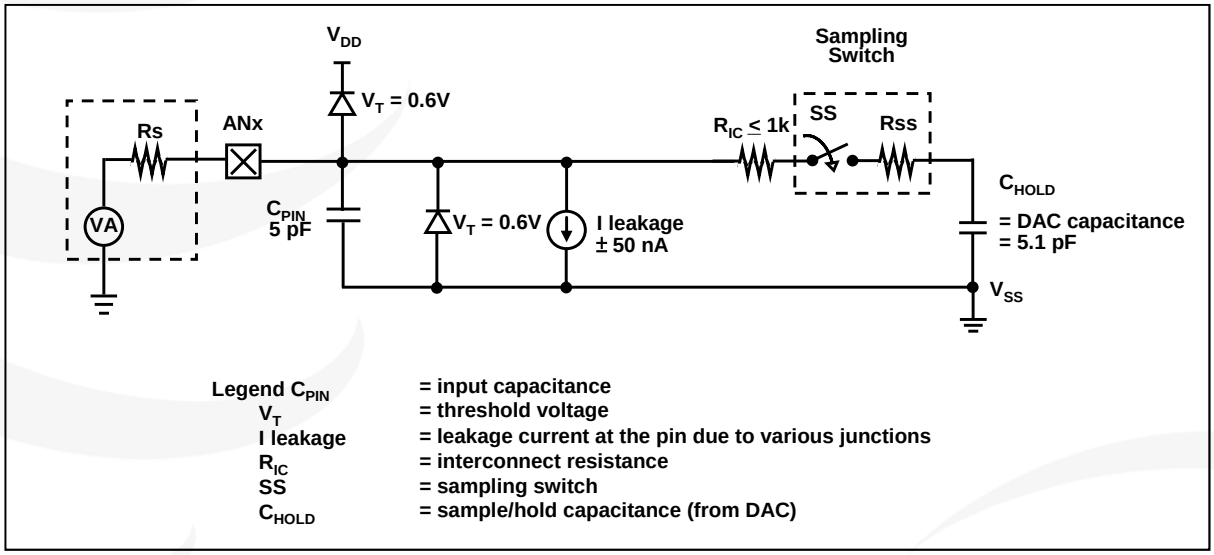


图 33：模拟输入模型

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，PFC887 系列 ADC 的信号采集时间(T_{ACQ})固定为 ADCLK 的一个时钟周期，ADCLK 的选择必须满足最短信号采集时间。

11.3.2. 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。

11.3.3. AD 转换

AD 转换的过程，从设置 START/DONE(ADCC.6) 为高开始，START/DONE 的标志位内部将会自动清零，然后转换模拟信号将会一位一位的转换，当 AD 转换完成时，START/DONE 将自动置高表示完成转换。当 ADCLK 被选定后，ADCLK 的周期是 TADCLK，而 AD 转换的时间将是如下：

- 11 位分辨率：AD 转换时间 = 14 T_{ADCLK}

11.3.4. 配置模拟引脚

有 16 模拟信号可以被 AD 转换选择：14 来自外部引脚的模拟输入信号，一个来自 Bandgap1.20V 参考电压，一个来自 OPA。以外部引脚而言，模拟信号与 PA5，PA6 和 PB[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

建议使用以下步骤来执行 AD 转换过程：

(1) 配置 ADC 模块
- 通过寄存器 *ADCC* 配置参考高电压
- 通过 *ADCC* 寄存器选择 ADC 输入通道
- 通过 *ADCM* 寄存器配置 ADC 转换时钟
- 通过 *PADIER/PBDIER* 寄存器配置所选定的引脚作为模拟输入
- 通过 *ADCC* 寄存器启用 ADC 模块

(2) 配置 ADC 模块的中断：（如果需要）
- 清零 *INTRQ* 寄存器位 3 的 ADC 中断请求标志
- 启用 *INTEN* 寄存器位 3 的 ADC 中断请求
- 通过 *ENGINT* 指令启用全局中断

(3) 启动 ADC 转换：
- 通过 *ADCC* 寄存器置位 ADC 转换过程控制位启动转换（set1 *ADCC.6*）

(4) 等待完成 AD 转换标志位置位，方法可以用如下的任一种：
- 通过使用指令“*wait1 ADCC.6*”来等待完成标志；或
- 等待 ADC 的中断

(5) 读取 ADC 的数据寄存器：
- 读取 *ADCRH* 和 *ADCRL* 数据寄存器

(6) 下一个转换，依要求转到步骤 1 或步骤 2。

11.3.5. 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
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
```

下一步，设定 *ADCM* 和 *ADCRGC* 寄存器，示例如下：

```
$   ADCM   /16;                //  建议 /16 @系统时钟=8MHz
$   ADCM   /8;                 //  建议 /8 @系统时钟=4MHz
```

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;          //  或 Data = (ADCRH << 8) | ADCRL;
```

ADC 也可以利用下面方法停用：

```
$   ADCC Disable;
```

或

```
ADCC        =   0;
```

11.3.6. ADC 相关寄存器

11.3.6.1. ADC 控制寄存器 (*ADCC*)，地址 = 0x20

位	初始值	读/写	描述
7	0	读/写	启用 ADC 功能。0/1：停用/启用。
6	-	读/写	ADC 转换过程控制位 写“1”开始 AD 转换，同时自动清零完成标志。 读到“1”表示完成 AD 的转换。读到“0”表示进行中。
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：PD0/AD10, 1011：PD1/AD11, 1100：PD2/AD12, 1101：PD3/AD13. 1110：OPAmp 1111：Bandgap 1.2 V 参考电压 其他：保留
1 - 0	-	-	保留（写 0）。

11.3.6.2. ADC 模式寄存器 (*ADCM*)，地址 = 0x21

位	初始值	读/写	描述
7 - 4	-	-	保留（写 0）
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。

11.3.6.3. ADC 数据高位寄存器(*ADCRH*)，地址 = 0x4A

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。

11.3.6.4. ADC 数据低位寄存器(*ADCRL*), 地址 = 0x4B

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。

11.4. PWM 生成器触发 AD 转换

PWM 生成器还可用于触发 AD 转换。启用寄存器 Hop 的触发使能位(*HOP*.4)和寄存器 ADCC 的 ADC 使能位(*ADCC*.7)后，PWM 计数值达到寄存器 *PWMADCH* 和 *PWMADCL* 的设定值时会触发 ADC 模块，待 ADC 转换完成后，*HOP*.4 会自动清零，如图 34 所示

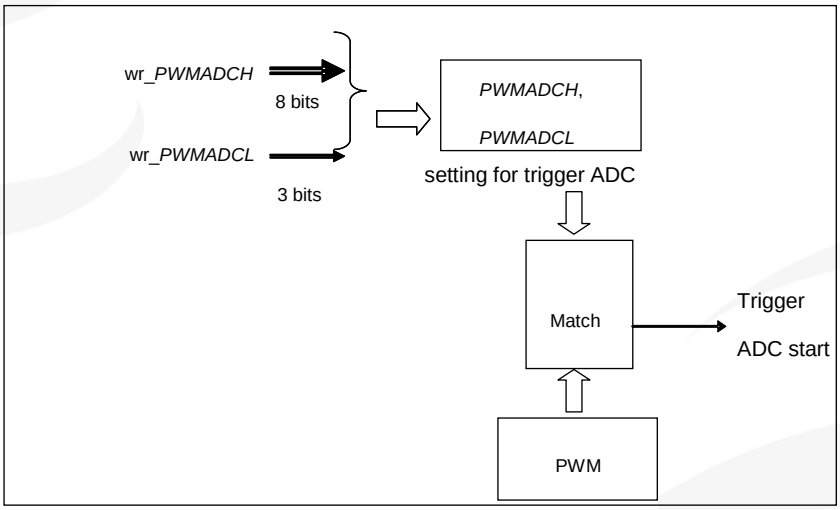


图 34：PWM 生成器触发 AD 转换框图

11.4.1. 跳跃设置寄存器(*HOP*)，地址 = 0x23

位	初始值	读/写	描 述
7	0	读/写	PWM 中断模式 0：当计数值达到上限寄存器设定值时产生中断请求 1：当计数为 0 时产生中断请求
6	0	读/写	PWM 计数器触发 ADC 模式。 0：向上计数达到寄存器 PWMADC 设定值时触发 ADC 模式 1：向下计数达到寄存器 PWMADC 设定值时触发 ADC 模式
5	-	-	保留。
4	0	读/写	PWM 触发 ADC 功能（由 ADC 转换完成后硬件清零） 0：停用 1：启用
3 - 0	-	-	保留

11.4.2. PWM 触发 ADC - PWM 计数高位寄存器(*PWMADCH*)，地址 = 0x5E

位	初始值	读/写	描 述
7 - 0	8'h00	只写	触发 ADC - PWM 计数值位[11:4]

11.4.3. PWM 触发 ADC - PWM 计数低位寄存器 (*PWMADCL*)，地址 = 0x5F

位	初始值	读/写	描 述
7 - 5	3'h0	只写	触发 ADC - PWM 计数值位[3:1]。
4 - 0	-	-	保留

11.5. 三相无刷直流马达

11.5.1. 三相 120 度 PWM 保护

对于三相电机的应用来说，避免工作时上臂或下臂同时开启十分重要。PFC887 提供硬件 PWM 保护电路，如图 35 所示。硬件 PWM 保护模块由寄存器 *PWMGC* 和 *PWMGM* 控制，适用于三相 120 度无刷直流保护的应用。在启用保护时，如果 *PWMGM.4* 的设置值与上臂高低电位状态不匹配，则 PWM 生成器将被禁用，使输出无效。

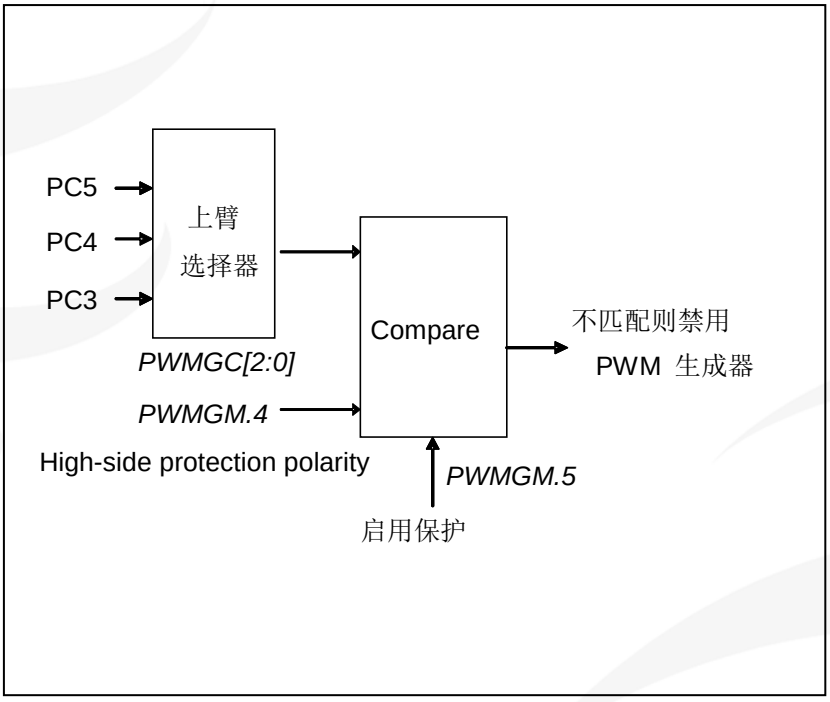


图 35：单相 PWM 保护框图

11.5.1.1. PWM 发生控制寄存器 (*PWMGC*), 地址= 0x30

位	初始值	读/写	描 述
7	0	读/写	启用 PWMG。0 / 1：停用 / 启用。
6	-	只读	PWM0 生成器输出状态
5	0	读/写	PWM 排列模式 0：边排列模式 1：中心排列模式
4	0	读/写	PWM 计数器清零 写“1”清零 PWM 计数，清零 PWM 计数后，这个位会自动归 0。
3	-	-	
2	0	读/写	PWM0 输出到 PC2 1：启用 0：停用
1	0	读/写	PWM0 输出到 PC1 1：启用 0：停用
0	0	读/写	PWM0 输出到 PC0 1：启用 0：停用

11.5.1.2. PWMG 分频寄存器 (*PWMGM*)，地址 = 0x31

PWMG 分频寄存器 (*PWMGM*)，地址 = 0x31			
位	初始值	读/写	描 述
7	0	只写	选择上臂 PWM 的输出结果是否反极性。 0/1：停用/启用
6	0	只写	Enable to inverse the polarity of low-site PWM output. 0 / 1 : disable / enable. 选择下臂 PWM 的输出结果是否反极性。 0/1：停用/启用
5	**0**	**读/写**	**臂保护启用。** **0/1：停用/启用。**
4	**0**	**读/写**	**臂保护模式。** **0：上臂高电位时执行臂保护。** **1：上臂低电位时执行臂保护。**
3 – 2	00	读/写	PWM generator 时钟源
1 – 0	-	-	

11.6. 输入脉冲捕获

输入脉冲捕获特性适用于需要测量频率和脉冲的应用。图 36 为 PFC887 中输入脉冲捕获的硬件框图，脉冲捕获模块的时基可以是 EOSC、IHRC 和 IHRCX2，用于测量的输入信号可以来自比较器 23 的输出、PA6、PA5、PB0、PB1、PB6 或 PB7。

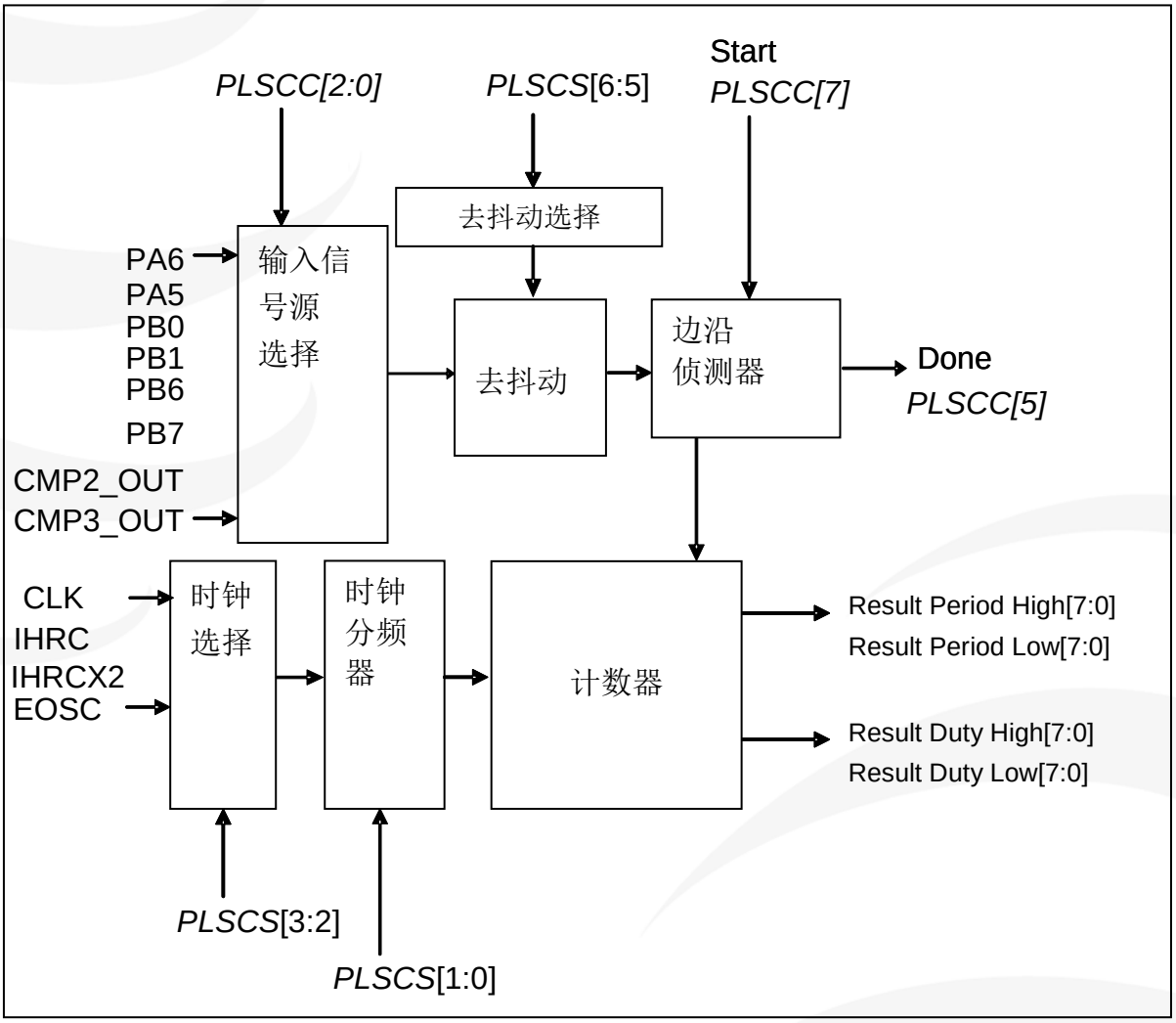


图 36：输入脉冲捕获的硬件框图

11.6.1. 脉冲捕获控制寄存器 (*PLSCC*)，地址 = 0x32

位	初始值	读/写	描 述
7	0	读/写	写：开始执行脉冲捕获并清除脉冲捕获溢出或完成标志。 读：脉冲捕获溢出或完成标志。
6	0	只读	脉冲捕获溢出标志。脉冲捕获完成后，该位内部自动清零。
5	0	读/写	写：清除脉冲捕获完成标志。 读：脉冲捕获完成标志。
2 - 0	000	读/写	脉冲捕获信号源。 000：PA6 001：PA5 010：PB0 011：PB1 100：PB6 101：PB7 110：比较器 2 的输出 111：比较器 3 的输出

仅支持使用 MOV 指令来进行写寄存器操作

11.6.2. 脉冲捕获分频寄存器(*PLSCS*)，地址 = 0x33

位	初始值	读/写	描 述
7	-	-	保留。请保持为 0。
6 - 5	00	读/写	脉冲捕获输入源去抖动时间选择。 00：无 01：1 个脉冲捕获时钟 10：2 个脉冲捕获时钟 11：3 个脉冲捕获时钟
4	-	-	保留。请保持为 0。
3:2	00	读/写	脉冲捕获时钟源。 00: system CLK 01: IHRC 10: IHRC*2 11: EOSC
1 - 0	00	读/写	脉冲捕获的时钟分频。 00 : /1 01 : /4 10 : /16 11 : /64

11.6.3. 脉冲捕获脉宽高位寄存器 (*PLSPWH*)，地址 = 0x4C

位	初始值	读/写	描 述
7 - 0	-	只读	脉冲捕获脉宽的高字节。

11.6.4. 脉冲捕获脉宽低位寄存器 (*PLSPWL*)，地址 = 0x4D

位	初始值	读/写	描 述
7 - 0	-	只读	脉冲捕获脉宽的低字节。

11.6.5. 脉冲捕获高准位脉宽高位寄存器 (*PLSPHH*)，地址 = 0x4E

位	初始值	读/写	描 述
7 - 0	-	只读	脉冲捕获高准位脉宽的高字节。

11.6.6. 脉冲捕获高准位脉宽低位寄存器 (*PLSPHL*)，地址 = 0x4F

位	初始值	读/写	描 述
7 - 0	-	只读	脉冲捕获高准位脉宽的低字节。

11.7. 乘法器和除法器

11.7.1. 16×8 乘法器

芯片内置一 16x8 乘法器以加强硬件的运算功能。这个乘法运算方式是 16x8 的无符号运算并且在 9 个时钟周期内完成运算。在设置开始位 (*EARITH*.6) 之前，乘数与被乘数都要放在寄存器 *M8OP1H/M8OP1L* 和 *M8OP2* 上。完成 16x8 后，内部设置完成位 (*EARITH*.1)，运算结果会放在寄存器 *M8RS2/M8RS1/M8RS0* 上。乘法器的硬件框图如图 37 所示。

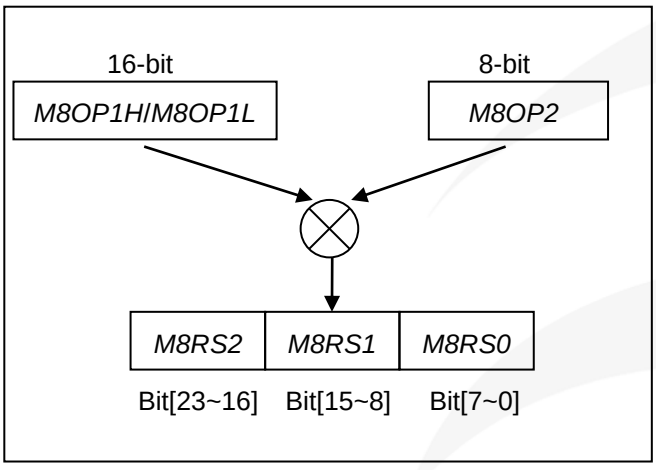


图 37：硬件乘法器框图

11.7.2. 16×16 乘法器

芯片内置一 16x16 乘法器以加强硬件的运算功能。这个乘法运算方式是 16x16 的无符号运算并且在 9 个时钟周期内完成运算。在设置开始位(*EARITH*.7)之前，乘数与被乘数都要放在寄存器 *M16OP1H/M16OP1L* 和 *M16OP2H/M16OP2L* 上。完成 16x16 后，内部设置完成位 (*EARITH*.2)，运算结果会放在寄存器 *M16RS3/M16RS2/M16RS1/M16RS0* 上。乘法器的硬件框图如图 38 所示。

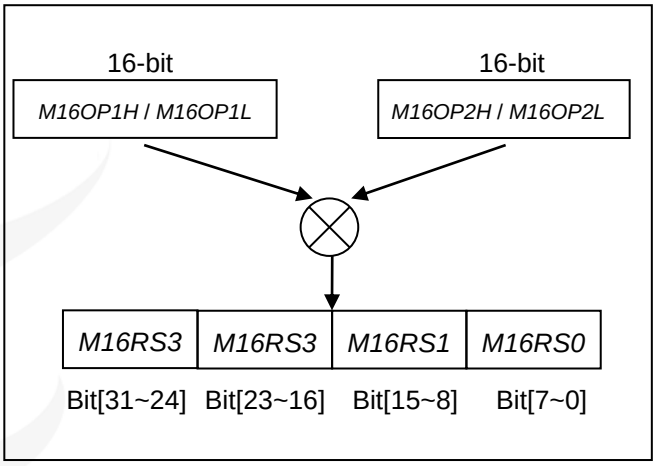


图 38：硬件乘法器框图

11.7.3. 16/16 除法器

芯片内置一 16/16 除法器以加强硬件的运算功能。这个除法运算方式是 16x16 的无符号运算并且在 14 个时钟周期内完成运算。在设置开始位(*EARITH*.5)之前，除数与被除数都要放在寄存器 *D16DEH/D16DEL* 和 *D16DSH/D16DSL* 上。完成 16/16 后，内部设置完成位 (*EARITH*.0)，商会放在寄存器 *D16QUH/D16QUL* 上，余数会放在寄存器 *D16REH/D16REL* 上。除法器的硬件框图如图 39 所示。

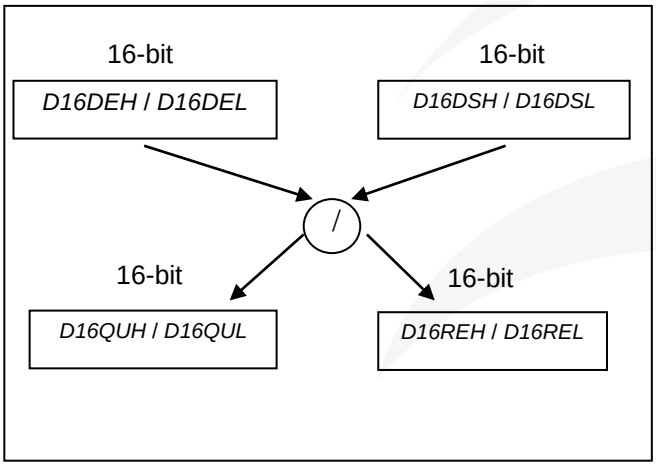


图 39：硬件除法器框图

11.7.4. 算术运算寄存器 (*EARITH*)，地址 = 0x3C

位	初始值	读/写	描述
7	-	只写	16X16 乘法器开始位，此位由硬件清零。
6	-	只写	16X8 乘法器开始位，此位由硬件清零。
5	-	只写	16/16 除法器开始位，此位由硬件清零。
4 - 3	-	只写	保留。请保持为 0。
2	-	只读	16X16 乘法器完成标志，该位由开始位清零。
1	-	只读	16X8 乘法器完成标志，该位由开始位清零。
0	-	只读	16/16 除法器完成标志，该位由开始位清零。

11.7.5. 16X8 乘法器运算对象 1 高字节寄存器 (*M8OP1H*)，地址 = 0x44

位	初始值	读/写	描述
7 - 0	-	只写	乘法运算运算对象 1 的高字节。

11.7.6. 16X8 乘法器运算对象 1 低字节寄存器 (*M8OP1L*)，地址 = 0x45

位	初始值	读/写	描述
7 - 0	-	只写	乘法运算运算对象 1 的低字节。

11.7.7. 16X8 乘法器运算对象 2 寄存器(*M8OP2*)，地址 = 0x46

位	初始值	读/写	描述
7 - 0	-	只写	乘法运算的运算对象 2。

11.7.8. 16X8 乘法器结果 2 寄存器(*M8RS2*)，地址 = 0x47

位	初始值	读/写	描述
7 - 0	-	只读	乘法运算的结果位 23~16。

11.7.9. 16X8 乘法器结果 1 寄存器(*M8RS1*)，地址 = 0x48

位	初始值	读/写	描述
7 - 0	-	只读	乘法运算的结果位 15~8。

11.7.10. 16X8 乘法器结果 0 寄存器(*M8RS0*)，地址 = 0x49

位	初始值	读/写	描述
7 - 0	-	只读	乘法运算的结果位 7~0。

11.7.11. 16X16 乘法器运算对象 1 高字节寄存器(*M16OP1H*)，地址 = 0x60

位	初始值	读/写	描述
7 - 0	-	只写	乘法运算运算对象 1 的高字节。

11.7.12. 16X16 乘法器运算对象 1 低字节寄存器(*M16OP1L*)，地址 = 0x61

位	初始值	读/写	描述
7 - 0	-	只写	乘法运算运算对象 1 的低字节。

11.7.13. 16X16 乘法器运算对象 2 高字节寄存器(*M16OP2H*)，地址 = 0x62

位	初始值	读/写	描述
7 - 0	-	只写	乘法运算运算对象 2 的高字节。

11.7.14. 16X16 乘法器运算对象 2 低字节寄存器(*M16OP2L*),，地址 = 0x63

位	初始值	读/写	描述
7 - 0	-	只写	乘法运算运算对象 2 的低字节。

11.7.15. 16X16 乘法器结果 3 寄存器(*M16RS3*)，地址 = 0x64

位	初始值	读/写	描述
7 - 0	-	只读	乘法运算的结果位 31~24（只读）。

11.7.16. 16X16 乘法器结果 2 寄存器(*M16RS2*)，地址 = 0x65

位	初始值	读/写	描述
7 - 0	-	只读	乘法运算的结果位 23~16（只读）。

11.7.17. 16X16 乘法器结果 1 寄存器(*M16RS1*)，地址 = 0x66

位	初始值	读/写	描述
7 - 0	-	只读	乘法运算的结果位 15~8（只读）。

11.7.18. 16X16 乘法器结果 0 寄存器(*M16RS0*)，地址 = 0x67

位	初始值	读/写	描述
7 - 0	-	只读	乘法运算的结果位 7~0（只读）。

11.7.19. 16/16 除法器被除数高字节寄存器(*D16DEH*)，地址 = 0x68

位	初始值	读/写	描述
7 - 0	-	只读	除法运算被除数的高字节。

11.7.20. 16/16 除法器被除数低字节寄存器 (*D16DEL*)，地址 = 0x69

位	初始值	读/写	描述
7 - 0	-	只读	除法运算被除数的低字节。

11.7.21. 16/16 除法器除数高字节寄存器(*D16DSH*)，地址 = 0x6A

位	初始值	读/写	描述
7 - 0	-	只读	除法运算除数的高字节。

11.7.22. 16/16 除法器除数低字节寄存器(*D16DSL*)，地址 = 0x6B

位	初始值	读/写	描述
7 - 0	-	只写	除法运算除数的低字节。

11.7.23. 16/16 除法器商高字节寄存器(*D16QUH*)，地址 = 0x6C

位	初始值	读/写	描述
7 – 0	-	只读	除法运算商的高字节。

11.7.24. 16/16 除法器商低字节寄存器(*D16QUL*)，地址 = 0x6D

位	初始值	读/写	描述
7 - 0	-	只读	除法运算上的低字节。

11.7.25. 16/16 除法器余数高字节寄存器(*D16REH*)，地址 = 0x6E

位	初始值	读/写	描述
7 - 0	-	只读	除法运算余数的高字节。

11.7.26. 16/16 除法器余数低字节寄存器 (*D16REL*)，地址 = 0x6F

位	初始值	读/写	描述
7 - 0	-	只读	除法运算余数的低字节。

12. 烧录方法

请使用 5S-P-003 进行烧录。3S-P-002 或之前的烧录器皆不支持烧录 PFC887。

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。

12.1. 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 PFC887 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 0.22uF 以上的电容，以免影响普通烧录模式的运行。**

12.2. 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 13.3 的详细说明。
- 合封（MCP）IC，但与 PFC887 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 5S-P-003 的用户手册）。

12.3. 在板烧录 (On-Board Writing)

PFC887 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 PDK5S-P-003 上六根引线：ICPCK、ICPDA、ICPDA2、VDD、GND 和 ICVPP，用于与 IC 上的 PA3、PA6、PA4、VDD、GND 和 PA5 对应相连。

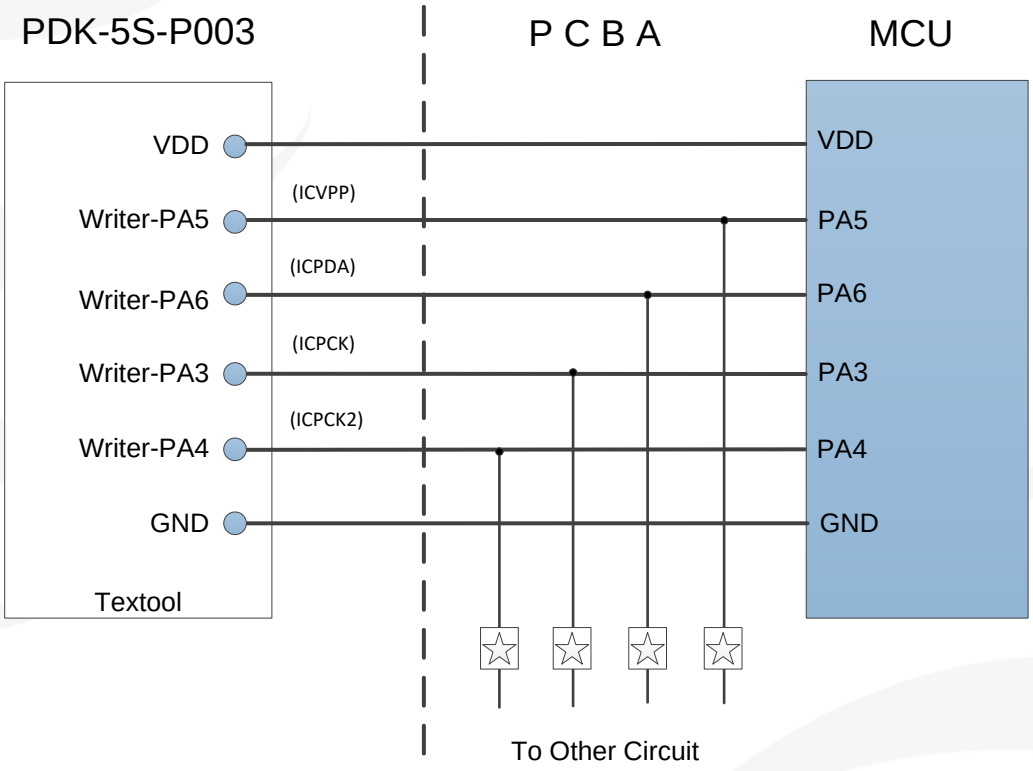


图 40：在板烧录接线示意图

图 40 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 12.2 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，用于烧录讯号引脚，**不能**作为强输出脚。

13. 器件电气特性

13.1. 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)	3.15		6	V	**电源电压不能超过最大值，否则可能损坏 IC**
输入电压	-0.3		V_{DD} + 0.2	V	
工作温度	-40		85	°C	
储藏温度	-50		125	°C	
节点温度		150		°C	

13.2. 直流/交流特性

符号	特性	最小值	典型值	最大值	单位	条件(Ta=25°C)
V_{DD}	工作电压	3.15	5.0	6	V	* 受限于 V_{BRD} 公差
f_{SYS}	系统时钟(CLK)* = IHRC/2 IHRC/4 ILRC	 0 0	 60K	 8M 4M	Hz	 V_{DD} = 3.3V V_{DD} = 3.15V V_{DD} = 5.0V
P_{cycle}	烧录次数	1000			cycles	
I_{OP}	工作电流		1.0 2.2 140 100		mA mA uA uA	f_{SYS}=1MIPS@5.0V f_{SYS}=8MIPS@5.0V f_{SYS}=ILRC ~ 60KHz@5.0V f_{SYS}=ILRC ~ 60KHz@3.3V
I_{PD}	掉电模式消耗电流 （使用 *stopsys* 命令）		1.7 1		uA uA	V_{DD}=5.0V V_{DD}=3.3V
I_{PS}	省电模式消耗电流 （使用 *stopexe* 命令）		19		uA	V_{DD}=5.0V; Bandgap, LVR, IHRC, ILRC, Timer16 modules are ON.
V_{IL}	输入低电压	0		0.2V_{DD}	V	
V_{IH}	输入高电压	0.8 V_{DD}		V_{DD}	V	
I_{OL}	IO 输出灌电流	11	14 7	17	mA	V_{DD}=5.0V, V_{OL}=0.5V, Normal V_{DD}=5.0V, V_{OL}=0.5V, Low
I_{OH}	IO 输出驱动电流	-8	-10 -6	-12	mA	V_{DD}=5.0V, V_{OH}=4.5V, Normal V_{DD}=5.0V, V_{OH}=4.5V, Low
R_{PH}	上拉电阻		31.5 32		KΩ	V_{DD}=5.0V V_{DD}=3.3V
V_{BRD}	低电压侦测电压*（欠压电压）	4.2 3.7 3.35 3.25 3.05 2.9	4.5 4 3.75 3.5 3.3 3.15	4.8 4.3 4.05 3.75 3.55 3.4	V	

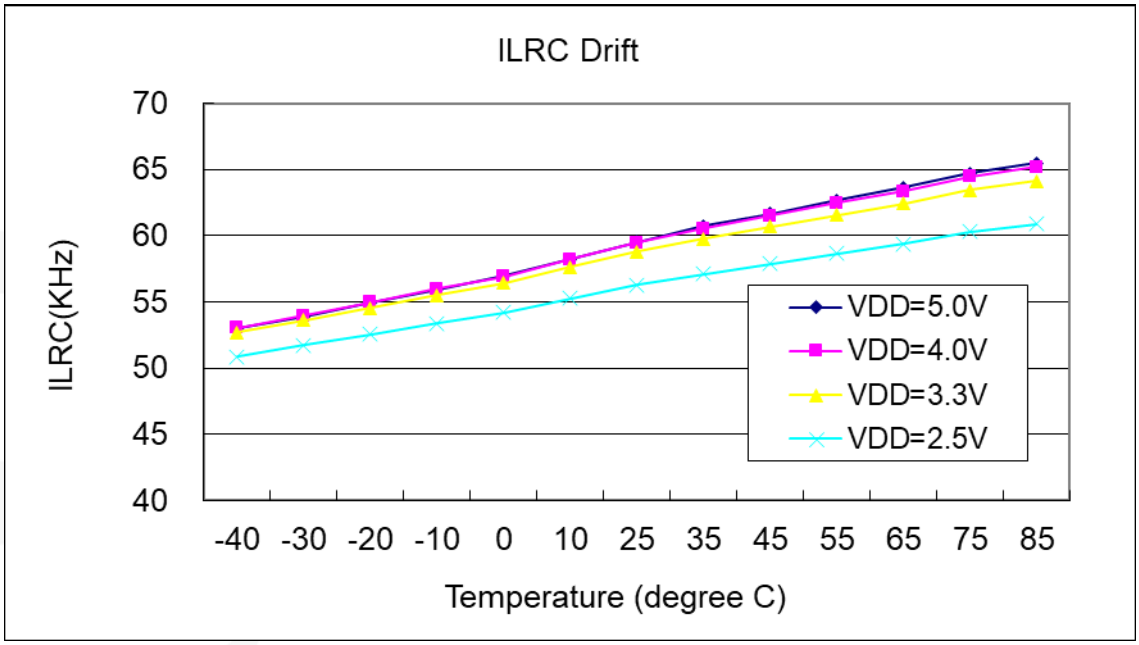
符号	特性	最小值	典型值	最大值	单位	条件(Ta=25°C)
V_{BG}	Bandgap 参考电压*（校准前）	1.12	1.20	1.28	V	V_{DD}=5V, 25°C
	Bandgap 参考电压 *(校准后)	1.17*	1.20*	1.23*		V_{DD}=3.15V ~ 5.5V, -40°C <Ta<85°C*
f_{IHRC}	IHRC 输出频率（校准后）*	15.52*	16*	16.48*	MHz	25°C, V_{DD}=3.15V~5.5V
		14*	16*	17.28*		V_{DD}=3.15V~5.5V, -40°C <Ta<85°C*
V_{ADC}	ADC 可工作电压	3.15		5.0	V	
V_{AD}	AD 输入电压	0		V_{DD}	V	
ADrs	ADC 分辨率			11	bit	
ADclk	ADC 时钟周期		2		us	3.15V ~ 5.5V
t_{ADCONV}	ADC 转换时间（T_{ADCLK} 是选定 AD 转换时钟周期）		14		T_{ADCLK}	
AD DNL	ADC 微分非线性		±3*		LSB	
AD INL	ADC 积分非线性		±3*		LSB	
ADos	ADC 失调电压*		3		LSB	-40°C <Ta<85°C*
t_{INT}	中断脉冲宽度	30			ns	V_{DD} = 5.0V
V_{DR}	数据存储器数据保存电压*	1.5			V	In power-down mode.
t_{WDT}	看门狗超时溢出时间（T_{ILRC} 是 ILRC 的时钟周期）		4096			misc[1:0]=01
			16384			misc[1:0]=10
t_{SBP}	系统上电开机时间		2500		T_{ILRC}	Where T_{ILRC} is the clock period of ILRC
t_{WUP}	系统唤醒时间					
	STOPEXE 省电模式和 STOPSYS 掉电模式下，切换 IO 引脚的普通唤醒		2500		T_{ILRC}	T_{ILRC} 是 ILRC 时钟周期
HCPos	*比较器偏压*	-	±10	±20	mV	
HCPcm	比较器共模输入电压*	0		V_{DD}-1.5	V	
HCPspt	比较器响应时间**		100	500	ns	上升沿和下降沿一样
HCPmc	比较器模式改变稳定时间		2.5	7.5	us	

*这些参数是设计参考值，并不是每个芯片测试。

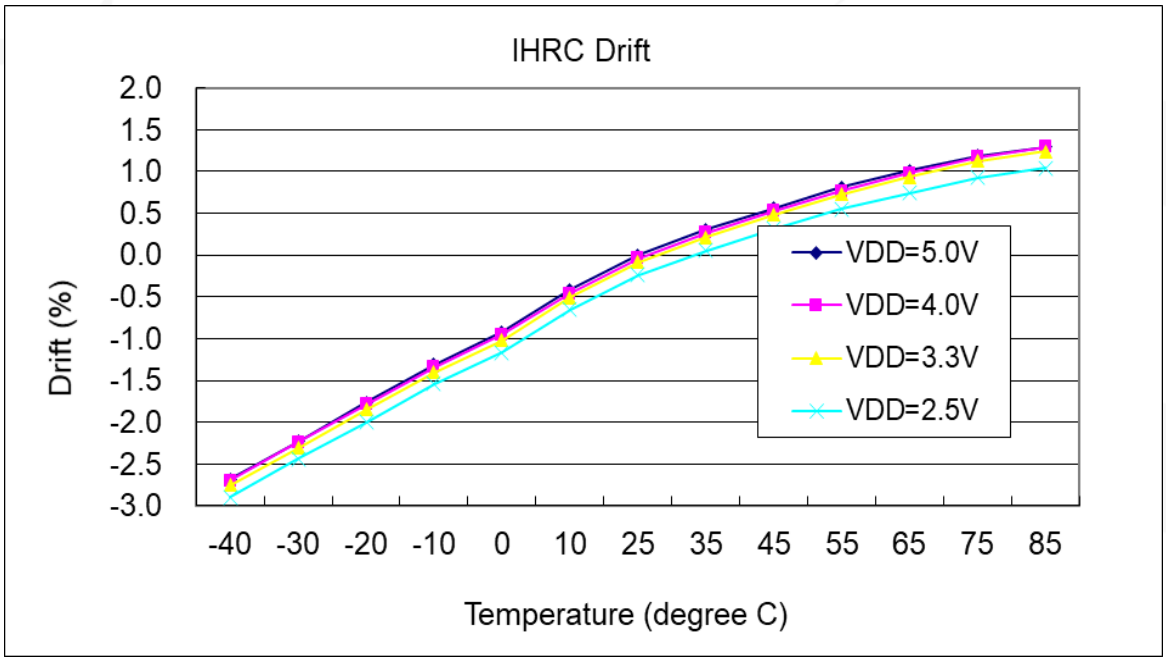
**比较器响应时间测量条件：输入电压为 (V_{DD}-1.5)/2 -100mV 和 (V_{DD}-1.5)/2+100mV

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。

13.3. ILRC 频率与 VDD 和温度关系曲线图



13.4. IHRC 频率偏差与 VDD、温度关系的量测图



13.5. 工作电流与 VDD、系统时钟 CLK=IHRC/n 曲线图

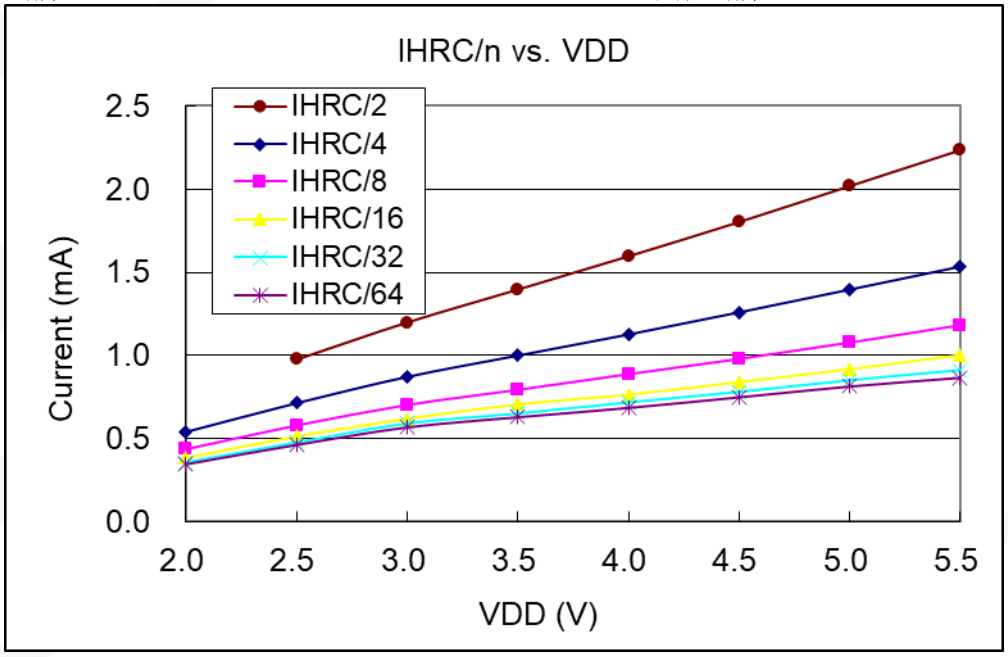
条件：

1-FPPA (code option)

启用：Bandgap, LVR, IHRC;

停用：ILRC, T16, TM2, ADC, PWM;

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



13.6. 工作电流与 VDD、系统时钟 CLK=ILRC/n 曲线图

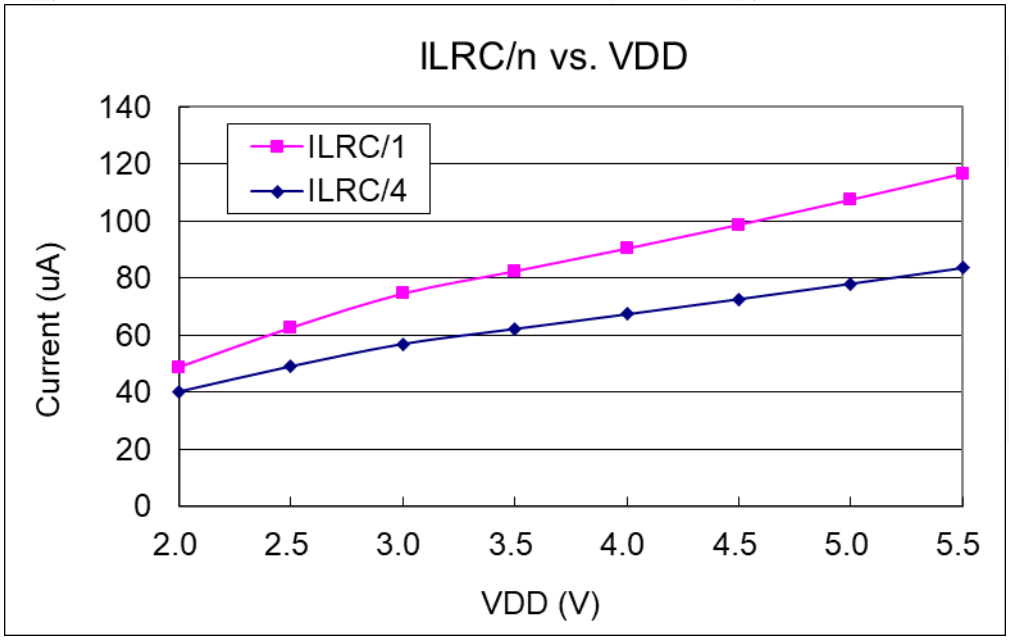
条件：

1-FPPA (code option)

启用：ILRC;

停用：Bandgap, LVR, IHRC, T16, TM2, ADC, PWM;

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



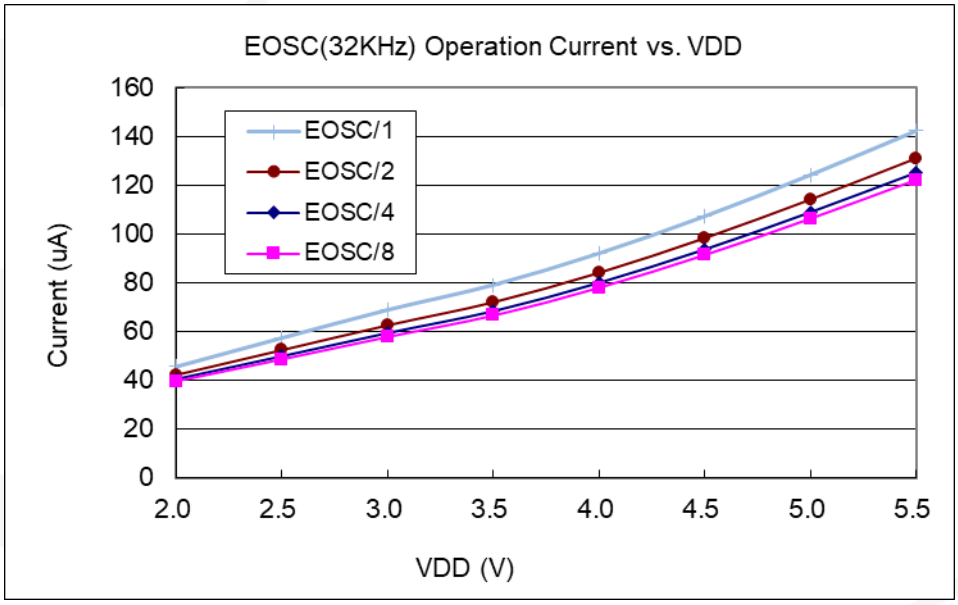
13.7. 工作电流与 VDD、系统时钟= 32KHz EOSC / n 曲线图

条件:

启用: EOSC[6,5] = [0,1], 基准电压, LVR;

停用: IHRC, ILRC, T16, TM2, TM3, ADC modules;

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



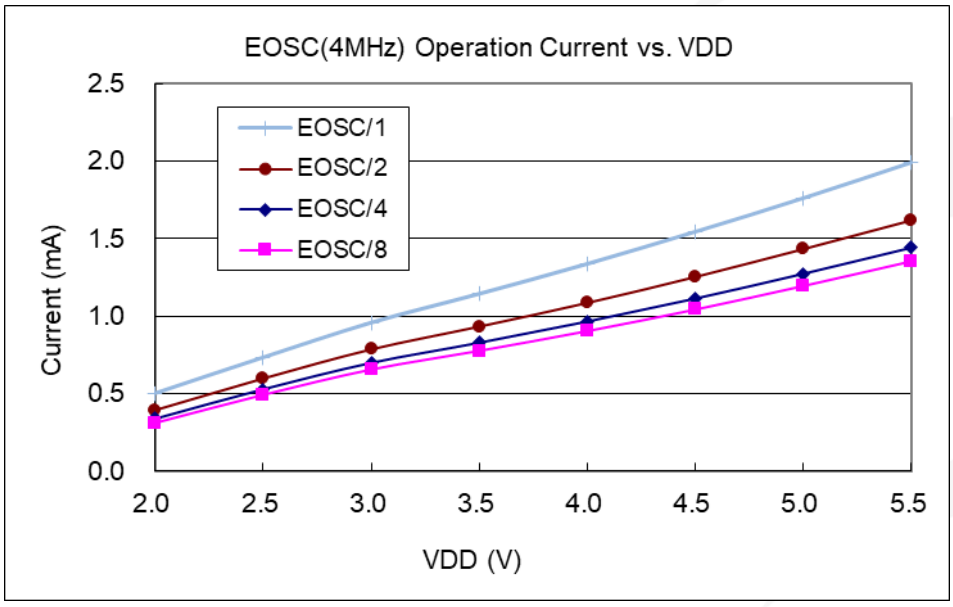
13.8. 工作电流与 VDD、系统时钟= 4MHz EOSC / n 曲线图

条件:

启用: EOSC[6,5] = [0,1], 基准电压, LVR;

停用: IHRC, ILRC, T16, TM2, TM3, ADC modules;

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



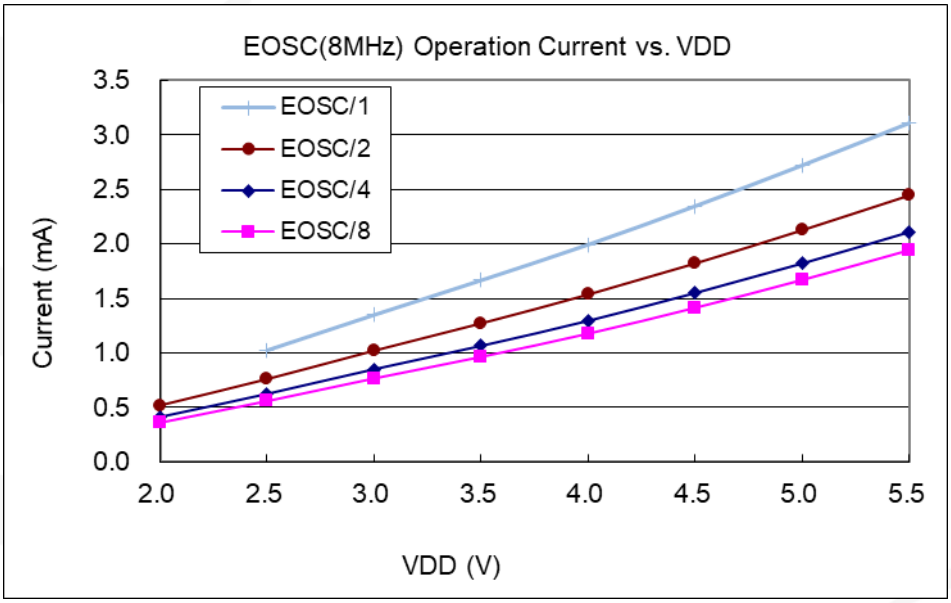
13.9. 工作电流与 VDD、系统时钟= 4MHz EOSC / n 曲线图

条件:

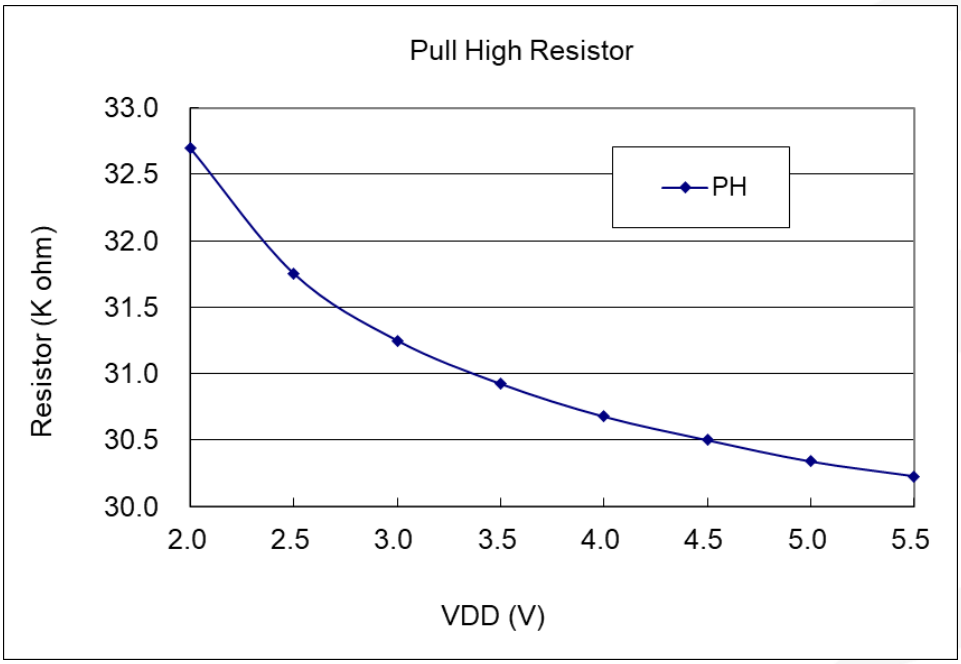
启用: EOSC[6,5] = [0,1], 基准电压, LVR;

停用: IHRC, ILRC, T16, TM2, TM3, ADC modules;

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

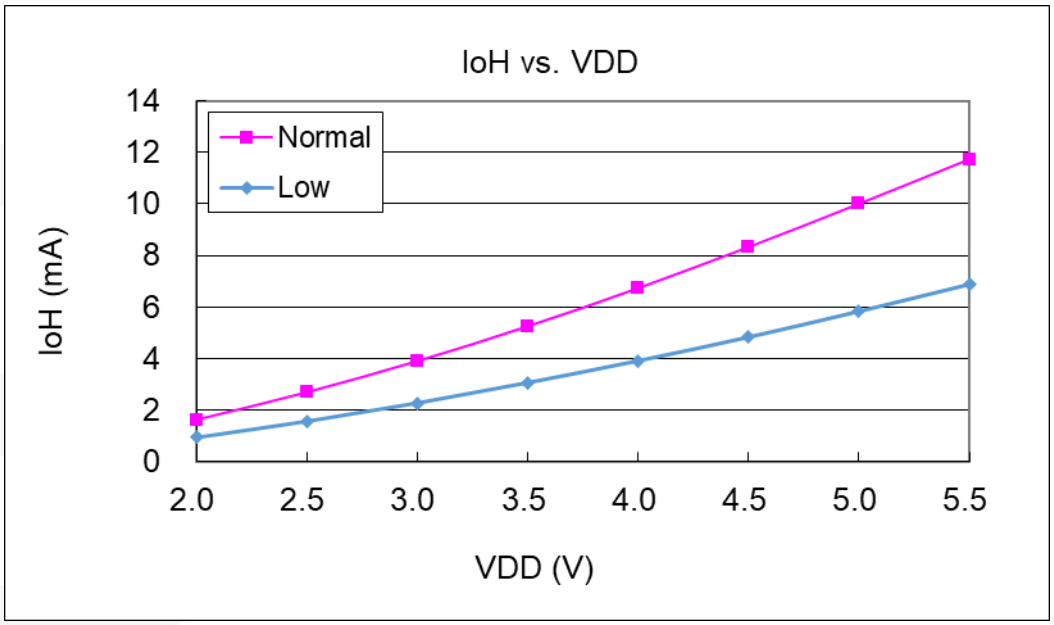


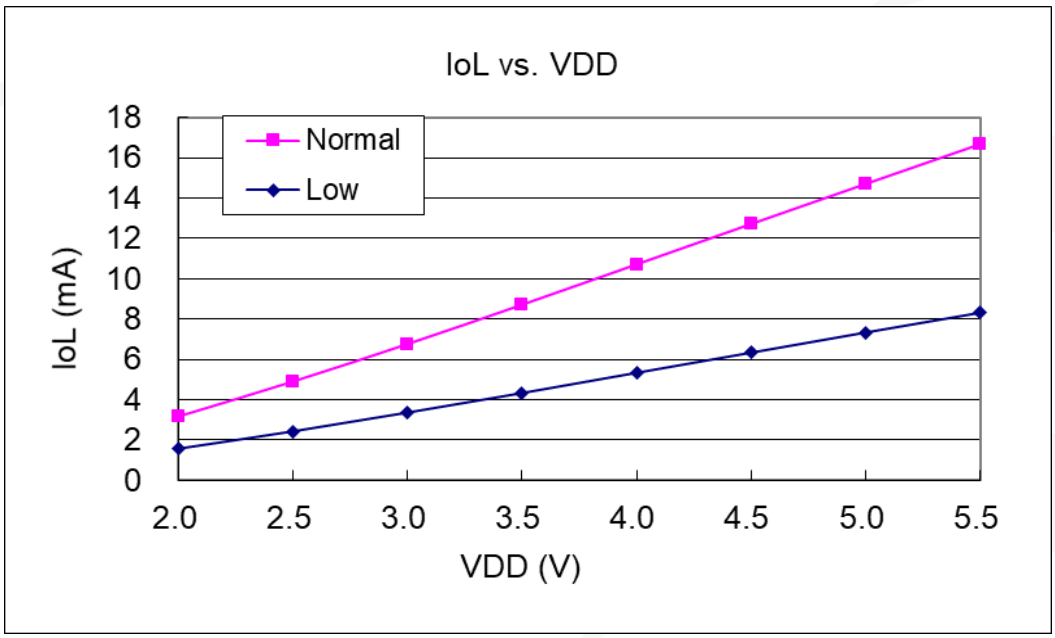
13.10. 引脚上拉电阻曲线图



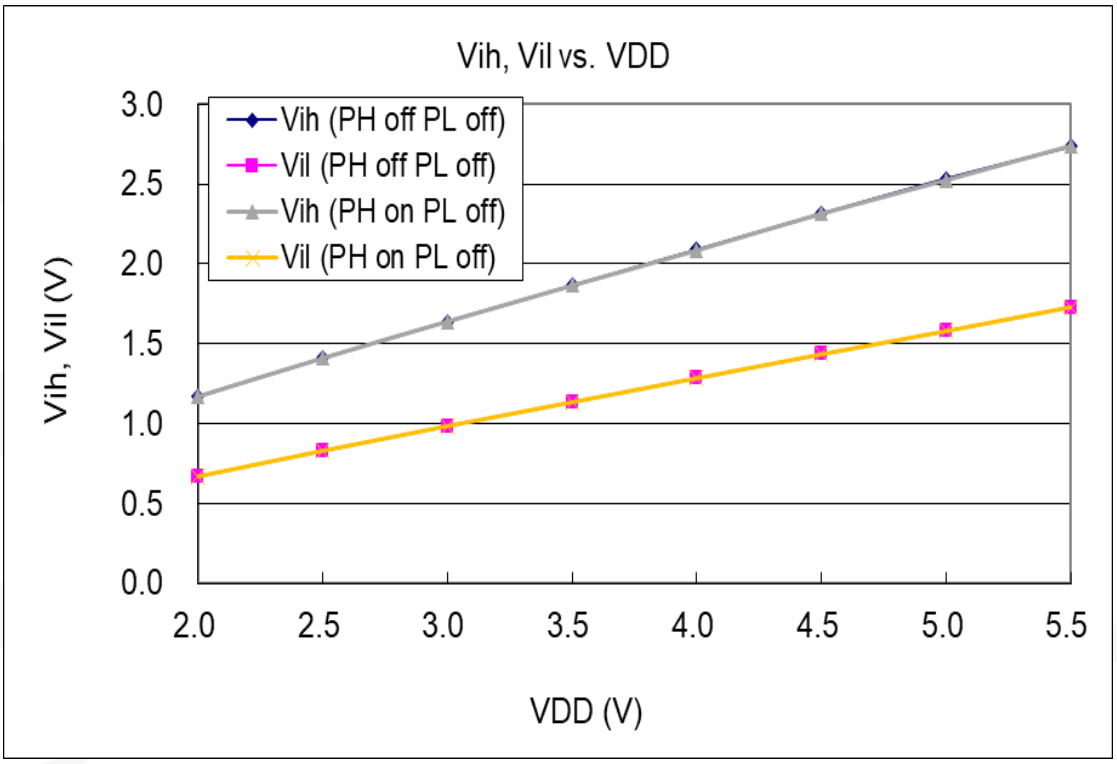
13.11. 引脚输出驱电流(I_{OH})与灌电流(I_{OL}) 曲线图

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





13.12. 引脚输入高电压与低电压（V_{IH}/V_{IL}）曲线图



14. 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)。
pc0	FPPA0 的程序计数器
pc1	FPPA1 的程序计数器
pc2	FPPA2 的程序计数器
pc3	FPPA3 的程序计数器
pc4	FPPA4 的程序计数器
pc5	FPPA5 的程序计数器
pc6	FPPA6 的程序计数器
pc7	FPPA7 的程序计数器

14.1. 指令表

指令	功能	周期	Z	C	AC	OV
数据传输类指令						
mov a, I	*mov* a, 0x0f; a ← 0fh;	1	-	-	-	-
mov M, a	*mov* MEM, a; MEM ← a	1	-	-	-	-
mov a, M	*mov* a, MEM ; a ← MEM; 当 MEM 为零时，标志位 Z 会被置位.	1	Y	-	-	-
mov a, IO	*mov* a, pa ; a ← pa; 当 pa 为零时，标志位 Z 会被置位	1	Y	-	-	-
mov IO, a	*mov* pb, a; pb ← a ;	1	-	-	-	-
nmov M, a	*nmov* MEM, a; MEM ← ∓a	1	-	-	-	-
nmov a, M	*mov* a, MEM ; a ← ∓MEM; Flag Z is set when ∓MEM is zero.	1	-	-	-	-
ldtabh index	*ldtabh index;* a ← {bit 15~8 of MTP [index]};	**2**	-	-	-	-
ldtabl index	*ldtabl index;* a ← {bit7~0 of MTP [index]};	**2**	-	-	-	-
ldt16 word	*ldt16* word; word ← 16-bit timer	1	-	-	-	-
stt16 word	*stt16* word; 16-bit timer ← word	1	-	-	-	-
idxm a, index	*idxm* a, index; a ← [index], where index is declared by word.	**2**	-	-	-	-
idxm index, a	*idxm* index, a; [index] ← a; where index is declared by word.	**2**	-	-	-	-
xch M	*xch* MEM ; MEM ← a , a ← MEM	1	-	-	-	-
pushaf	*pushaf*; [sp] ← {flag, ACC}; sp ← sp + 2 ;	1	-	-	-	-
popaf	*popaf;* sp ← sp - 2 ; {Flag, ACC} ← [sp] ;	1	Y	Y	Y	Y
pushw word	*pushw word*; [sp] ← *word* ; sp ← sp + 2	**2**	-	-	-	-
popw word	*popw word*; sp ← sp - 2 ; word ← [sp] ;	**2**	-	-	-	-

指令	功能	周期	Z	C	AC	OV
算术运算类指令						
add a, I	*add* a, 0x0f ; a ← a + 0fh	1	Y	Y	Y	Y
add a, M	*add* a, MEM ; a ← a + MEM	1	Y	Y	Y	Y
add M, a	*add* MEM, a; MEM ← a + MEM	1	Y	Y	Y	Y
addc a, M	*addc* a, MEM ; a ← a + MEM + C	1	Y	Y	Y	Y
addc M, a	*addc* MEM, a ; MEM ← a + MEM + C	1	Y	Y	Y	Y
addc a	*addc* a ; a ← a + C	1	Y	Y	Y	Y
addc M	*addc* MEM ; MEM ← MEM + C	1	Y	Y	Y	Y
nadd a, M	*nadd* a, MEM ; a ← ∓a + MEM	1	Y	Y	Y	Y
nadd M, a	*nadd* MEM, a; MEM ← ∓MEM + a	1	Y	Y	Y	Y
sub a, I	*sub* a, 0x0f; a ← a - 0fh (a + [2's complement of 0fh])	1	Y	Y	Y	Y
sub a, M	*sub* a, MEM ; a ← a - MEM (a + [2's complement of M])	1	Y	Y	Y	Y
sub M, a	*sub* MEM, a; MEM ← MEM - a (MEM + [2's complement of a])	1	Y	Y	Y	Y
subc a, M	*subc* MEM, a ; a ← a – MEM - C	1	Y	Y	Y	Y
subc M, a	*subc* MEM, a ; MEM ← MEM – a - C	1	Y	Y	Y	Y
subc a	*subc* a; a ← a - C	1	Y	Y	Y	Y
subc M	*subc* MEM; MEM ← MEM - C	1	Y	Y	Y	Y
inc M	*inc* MEM ; MEM ← MEM + 1	1	Y	Y	Y	Y
dec M	*dec* MEM; MEM ← MEM - 1	1	Y	Y	Y	Y
clear M	*clear* MEM ; MEM ← 0	1	-	-	-	-

指令	功能	周期	Z	C	AC	OV
移位运算类指令						
sr a	*sr* a ; a (0,b7,b6,b5,b4,b3,b2,b1) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a(b0)	1	-	Y	-	-
src a	*src* a ; a (c,b7,b6,b5,b4,b3,b2,b1) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a(b0)	1	-	Y	-	-
sr M	*sr* MEM ; MEM(0,b7,b6,b5,b4,b3,b2,b1) ← MEM(b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM(b0)	1	-	Y	-	-
src M	*src* MEM ; MEM(c,b7,b6,b5,b4,b3,b2,b1) ← MEM (b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM(b0)	1	-	Y	-	-
sl a	*sl* a ; a (b6,b5,b4,b3,b2,b1,b0,0) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a (b7)	1	-	Y	-	-
slc a	*slc* a ; a (b6,b5,b4,b3,b2,b1,b0,c) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a(b7)	1	-	Y	-	-
sl M	*sl* MEM ; MEM (b6,b5,b4,b3,b2,b1,b0,0) ← MEM (b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM(b7)	1	-	Y	-	-
slc M	*slc* MEM ; MEM (b6,b5,b4,b3,b2,b1,b0,C) ← MEM (b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM (b7)	1	-	Y	-	-
swap a	*swap* a ; a (b3,b2,b1,b0,b7,b6,b5,b4) ← a (b7,b6,b5,b4,b3,b2,b1,b0)	1	-	-	-	-
swap M	*swap* MEM ; MEM (b3,b2,b1,b0,b7,b6,b5,b4) ← MEM (b7,b6,b5,b4,b3,b2,b1,b0)	1	-	-	-	-
逻辑运算类指令						
and a, I	*and* a, 0x0f ; a ← a & 0fh	1	Y	-	-	-
and a, M	*and* a, RAM10 ; a ← a & RAM10	1	Y	-	-	-
and M, a	*and* MEM, a ; MEM ← a & MEM	1	Y	-	-	-
or a, I	*or* a, 0x0f ; a ← a | 0fh	1	Y	-	-	-
or a, M	*or* a, MEM ; a ← a | MEM	1	Y	-	-	-
or M, a	*or* MEM, a ; MEM ← a | MEM	1	Y	-	-	-
xor a, I	*xor* a, 0x0f ; a ← a ^ 0fh	1	Y	-	-	-
xor IO, a	*xor* *pa, a ;* pa ← a ^ pa ;	1	-	-	-	-
xor a, M	*xor* a, MEM ; a ← a ^ RAM10	1	Y	-	-	-
xor M, a	*xor* MEM, a ; MEM ← a ^ MEM	1	Y	-	-	-
not a	*not* a ; a ← ~a	1	Y	-	-	-
not M	*not* MEM ; MEM ← ~MEM	1	Y	-	-	-
neg a	*neg* a; a ← ∓a	1	Y	-	-	-
neg M	*neg* MEM; MEM ← ∓MEM	1	Y	-	-	-
comp a, I	*comp* a, 0x55; 等效于(a - 0x55)，并改变标志位	1	Y	Y	Y	Y
comp a, M	*comp* a,MEM;等效于(a - MEM)，并改变标志位 Flag	1	Y	Y	Y	Y
comp M, a	*comp* MEM,a; 等效于(MEM - a)，并改变标志位 Flag	1	Y	Y	Y	Y

指令	功能	周期	Z	C	AC	OV
位运算类指令						
set0 IO.n	*set0* pa.5 ; PA5=0	1	-	-	-	-
set1 IO.n	*set1* pb.5 ; PB5=1	1	-	-	-	-
set0 M.n	*set0* MEM.5 ; set bit 5 of MEM to low	1	-	-	-	-
set1 M.n	*set1* MEM.5 ; set bit 5 of MEM to high	1	-	-	-	-
swapc IO.n	*swapc* IO.0; 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	1	-	Y	-	-
tog IO.n	*tog* pa.5 ; PA5 改变为相反状态	1	-	-	-	-
条件运算类指令						
ceqsn a, I	*ceqsn* a, 0x55 ; *inc* MEM ; *goto* error ; 假如 a=0x55, then “goto error”; 否则, “inc MEM”.	**1 / 2**	Y	Y	Y	Y
ceqsn a, M	*ceqsn* a, MEM; 假如 a=MEM，跳过下一个指令	**1 / 2**	Y	Y	Y	Y
ceqsn M, a	*ceqsn* MEM, a; 假如 a=MEM，跳过下一个指令	**1 / 2**	Y	Y	Y	Y
cneqsn a, M	*cneqsn* a, MEM; 假如 a≠MEM，跳过下一个指令	**1 / 2**	Y	Y	Y	Y
cneqsn M, a	*cneqsn* MEM,a; 假如 a≠MEM，跳过下一个指令	**1 / 2**	Y	Y	Y	Y
cneqsn a, I	*cneqsn* a, 0x55 ; *inc* MEM ; *goto* error ; 假如 a≠0x55, then “goto error”; 否则, “inc MEM”.	**1 / 2**	Y	Y	Y	Y
t0sn IO.n	*t0sn* pa.5;如果 PA5 是 0，跳过下一个指令	**1 / 2**	-	-	-	-
t1sn IO.n	*t1sn* pa.5; 如果 PA5 是 1，跳过下一个指令	**1 / 2**	-	-	-	-
t0sn M.n	*t0sn* MEM.5 ; 如果 MEM 的位 5 是 0，跳过下一个指令	**1 / 2**	-	-	-	-
t1sn M.n	*t1sn* MEM.5 ; 如果 MEM 的位 5 是 1，跳过下一个指令	**1 / 2**	-	-	-	-
izsn a	*izsn* a; a ← a + 1,若 a=0，跳过下一个指令	**1 / 2**	Y	Y	Y	Y
dzsn a	*dzsn* a; a ← a – 1，若 a=0，跳过下一个指令	**1 / 2**	Y	Y	Y	Y
izsn M	*izsn* MEM; MEM ← MEM + 1，若 MEM=0，跳过下一个指令	**1 / 2**	Y	Y	Y	Y
dzsn M	*dzsn* MEM; MEM ← MEM - 1，若 MEM=0，跳过下一个指令	**1 / 2**	Y	Y	Y	Y
wait0 IO.n	*wait0* pa.5; 直到 PA5 为 0，才转到下一个指令	1	-	-	-	-
wait1 IO.n	*wait1* pa.5; 直到 PA5 为 1，才转到下一个指令	1	-	-	-	-

指令	功能	周期	Z	C	AC	OV
系统控制类指令						
call label	*call* function1; [sp] ← pc + 1, pc ← function1, sp ← sp + 2	**2**	-	-	-	-
goto label	*goto* routine1; 跳到 routine1 并继续执行程序	**2**	-	-	-	-
delay I	*delay* 0x05; 在此延迟 6 个周期	1	-	-	-	-
delay a	*delay* a; 假如 ACC=0fh，在此延迟 16 个周期	1	-	-	-	-
delay M	*delay* M; 假如 M=ffh，在此延迟 256 个周期	1	-	-	-	-
***delay* 指令的注意事项：** (1) 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，延时时间可能不是预期的。 (2) 单一 FPPA 模式下不支持此指令。						
ret I	*ret* 0x55; A ← 55h ret ;	**2**	-	-	-	-
ret	*ret;* sp ← sp - 2 pc ← [sp]	**2**	-	-	-	-
reti	*reti*; 从中断服务程序返回到原程序 在这指令执行之后，全局中断将自动启用	**2**	-	-	-	-
nop	*nop*; 没任何改变	1	-	-	-	-
pcadd a	*pcadd* *a*; pc ← pc + a	**2**	-	-	-	-
engint	*engint*; 中断请求可送到 FPPA0	1	-	-	-	-
disgint	*disgint* ; 送到 FPPA0 的中断请求全部被挡住	1	-	-	-	-
stopsys	*stopsys*; 停止系统时钟和关闭系统	1	-	-	-	-
stopexe	*stopexe*; 停止系统时钟，但时钟源还是保持原来的状态	1	-	-	-	-
reset	*reset*; 复位整个单片机	1	-	-	-	-
wdreset	*wdreset* ; 复位看门狗定时器	1	-	-	-	-
pmode n	各 FPPA 单元的操作模式选择 *pmode* 0; 设置 FPPA 单元的带宽共享模式为模式 0	1	-	-	-	-

指令	功能	周期	Z	C	AC	OV
系统控制类指令						
pmode n	各 FPPA 单元的操作模式选择 *pmode* 0; 设置 FPPA 单元的带宽共享模式为模式 0 模式 FPPA0 ~ FPPA7 带宽共享 0: /2, /2 1: /2, /4, /4 2: /4, /2, /4 3: /2, /4, /8, /8 4: /4, /2, /8, /8 5: /8, /2, /4, /8 6: /4, /4, /4, /4 7: /8, /4, /4, /4, /8 8: /2, /8, /8, /8, /8 9: /4, /4, /4, /8, /8 10: /8, /2, /8, /8, /8 11: /2, /8, /8, /8, /16, /16 12: /16, /2, /8, /8, /8, /16 13: /4, /4, /8, /8, /8, /8 14: /8, /4, /4, /8, /8, /8 15: /4, /4, /4, /8, /16, /16 16: /8, /4, /4, /4, /16, /16 17: /16, /4, /4, /4, /8, /16 18: /2, /8, /8, /16, /16, /16, /16 19: /8, /2, /8, /16, /16, /16, /16 20: /16, /2, /8, /8, /16, /16, /16 21: /4 , /4, /4, /16, /16, /16, /16 22: /16, /4, /4, /4, /16, /16, /16 23: /4, /8, /8, /8, /8, /8, /8 24: /8, /2, /16, /16, /16, /16, /16, /16 25: /4, /8, /4, /8, /16, /16, /16, /16 26: /8, /4, /4, /8, /16, /16, /16, /16 27: /2, /8, /16, /16, /16, /16, /16, /16 28: /4, /4, /8, /8, /16, /16, /16, /16 29: /16, /2, /8, /16, /16, /16, /16, /16 30: /8, /4, /4, /8, /16, /16, /16, /16 31: /8, /8, /8, /8, /8, /8, /8, /8	1	-	-	-	-