PFC460

工业级-24 触摸键

8-bit MTP 类型单片机（FPPA™）

数据手册

第 0.08 版

2026 年 1 月 23 日

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

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

重要声明

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

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

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

目　录

修订历史 **8**
使用警告 **8**
1. 单片机特点 **9**
1.1. 系统特性 9
1.2. 系统功能 9
1.3. CPU 特点 10
1.4. 订购/封装信息 10
2. 系统概述和方框图 **11**
3. 引脚分配及功能说明 **12**
4. 中央处理器 (CPU) **15**
4.1. 功能描述 15
4.1.1. 处理单元 15
4.1.2. 程序计数器 18
4.1.3. 程序结构 18
4.1.4. 算术和逻辑单元 19
4.2. 存储器 19
4.2.1. 程序存储器 (ROM) 19
4.2.2. 数据存储器 (SRAM) 22
4.2.3. 系统寄存器 23
4.2.3.1. 标志寄存器(*FLAG*)，地址 = 0x00 24
4.2.3.2. FPPA 单元允许寄存器(*FPPEN*)，地址 = 0x01 24
4.2.3.3. 杂项寄存器(*MISC*)，地址 = 0x49 24
4.3. 堆栈 25
4.3.1. 堆栈指针寄存器(*SP*)，地址 = 0x02 26
4.4. 程序选项 Code Options 26
5. 振荡器和系统时钟 **27**
5.1. 内部高频振荡器和内部低频振荡 28
5.2. 外部晶体振荡器 28
5.2.1. 外部晶体振荡器控制寄存器(*EOSCR*)，地址 = 0x0F 28
5.2.2. 外部晶体振荡器的使用及注意事项 29
5.3. 系统时钟与 IHRC 频率校准 30
5.3.1. 系统时钟 30
5.3.1.1. 时钟控制寄存器(*CLKMD*)，地址 = 0x03 30
5.3.2. 频率校准 31
5.3.2.1. 特别声明 32
5.3.3. 系统时钟切换 32
6. 复位 **33**
6.1. 上电复位(POR) 33
6.2. 低电压复位(LVR) 34
6.3. 看门狗超时溢出复位 36
6.4. 外部复位(PRSTB) 37
7. 系统工作模式 **37**

7.1. 省电模式(“*stopexe*”) 38
7.2. 掉电模式(“*stopsys*”) 39
7.3. 唤醒 40
8. 中断 41
8.1. 中断允许寄存器(*INTEN*)，地址 = 0x04 42
8.2. 中断允许寄存器 2 (*INTEN2*)，地址 = 0x06 43
8.3. 中断请求寄存器(*INTRQ*)，地址 = 0x05 43
8.4. 中断请求寄存器 2 (*INTRQ2*)，地址 = 0x07 43
8.5. 中断缘选择寄存器 (*INTEGS*), 地址 = 0x5D 44
8.6. 中断工作流程 44
8.7. 中断的一般步骤 45
8.8. 使用中断举例 46
9. I/O 端口 47
9.1. IO 相关寄存器 47
9.1.1. 端口 A 数字输入启用寄存器(*PADIER*)，地址 = 0x4C 47
9.1.2. 端口 B 数字输入启用寄存器(*PBDIER*)，地址 = 0x4D 47
9.1.3. 端口 C 数字输入启用寄存器(*PCDIER*)，地址 = 0x4E 47
9.1.4. 端口 D 数字输入启用寄存器(*PDDIER*)，地址 = 0x4F 47
9.1.5. 端口 A/B/C 数据寄存器(*PA/PB/PC*)，地址 = 0x10/0x14/0x18 47
9.1.6. 端口 A/B/C 控制寄存器(*PAC/PBC/PCC*)，地址 = 0x11/0x15//0x19 48
9.1.7. 端口 A/B/C 上拉控制寄存器(*PAPH/PBPH/PCPH*)，地址 = 0x12/0x16/0x1A 48
9.1.8. 端口 A/B/C 下拉控制寄存器(*PAPL/PBPL/PCPL*)，地址 = 0x13/0x17/0x1B 48
9.1.9. 端口 D 数据寄存器(*PD*)，地址 = 0x1C 48
9.1.10. 端口 D 控制寄存器(*PDC*)，地址 = 0x1D 48
9.1.11. 端口 D 上拉控制寄存器(*PDPH*)，地址 = 0x1E 48
9.1.12. 端口 D 下拉控制寄存器(*PDPL*)，地址 = 0x1F 48
9.2.IO 结构及功能 49
9.2.1. IO 引脚的结构 49
9.2.2. IO 引脚的一般功能 49
9.2.2.1. IO 输出驱动能力控制寄存器(*IOHD*)，地址 = 0x5F 50
9.2.3. IO 使用与设定 51
10. Timer / PWM 计数器 52
10.1. 16 位计数器 (Timer16) 52
10.1.1. Timer16 介绍 52
10.1.2. Timer16 溢出时间 53
10.2. 8 位 PWM 计数器 (Timer2,Timer3) 55
10.2.1. Timer2、Timer3 相关寄存器 56
10.2.1.1. Timer2/Timer3 上限寄存器(*TM2B/TM3B*), 地址 = 0x61/0x63 56
10.2.1.2. Timer2/Timer3 计数寄存器(*TM2CT/TM3CT*)，地址 = 0x29/0x2B 56
10.2.1.3. Timer2/Timer3 分频寄存器(*TM2S/TM3S*)，地址 = 0x60/0x62 56
10.2.1.4. Timer2/Timer3 控制寄存器(*TM2C/TM3C*)，地址 = 0x28/0x2A 57
10.2.2. 使用 Timer2 产生定期波形 58
10.2.3. 使用 Timer2 产生 8 位 PWM 波形 59
10.2.4. 使用 Timer2 产生 6 位 PWM 波形 60
10.3. 11 位 PWM 计数器 (PWMG0/1/2) 60
10.3.1. PWM 波形 61

10.3.2. 硬件和时钟框图 61
10.3.3. 11 位 PWM 生成器计算公式 62
10.3.4. 11bit PWM 计数器相关寄存器 63
10.3.4.1. PWMG0 控制寄存器(PWMG0C)，地址= 0x22 63
10.3.4.2. PWMG0 分频寄存器(PWMG0S)，地址= 0x23 63
10.3.4.3. PWMG0 占空比高位寄存器(PWMG0DTH)，地址 = 0x50 63
10.3.4.4. PWMG0 占空比低位寄存器(PWMG0DTL)，地址 = 0x51 64
10.3.4.5. PWMG0 计数上限高位寄存器(PWMG0CUBH)，地址= 0x52 64
10.3.4.6. PWMG0 计数上限低位寄存器(PWMG0CUBL)，地址= 0x53 64
10.3.4.7. PWMG1 控制寄存器(PWMG1C)，地址= 0x24 64
10.3.4.8. PWMG2 控制寄存器(PWMG2C)，地址= 0x26 65
10.3.4.9. PWMG1/PWMG2 分频寄存器(PWMG1S/PWMG2S)，地址= 0x25/0x27 65
10.3.4.10. PWMG1/PWMG2 占空比高位寄存器(PWMG1DTH/PWMG2DTH) 65
10.3.4.11. PWMG1/PWMG2 占空比低位寄存器(PWMG1DTL/PWMG2DTL) 65
10.3.4.12. PWMG1/PWMG2 计数上限高位寄存器(PWMG1CUBH/PWMG2CUBH) 66
10.3.4.13. PWMG1/PWMG2 计数上限低位寄存器(PWMG1CUBL/PWMG2CUBL) 66
10.3.5. 带互补死区的 PWM 波形范例 66
10.4. 11 位 SuLED LPWM 计数器 (LPWMG0/1/2) 68
10.4.1. LPWM 波形 68
10.4.2. 硬件框图 69
10.4.3. 11 位 LPWM 生成器计算公式 70
10.4.4. 11bit LPWM 计数器相关寄存器 71
10.4.4.1. LPWMG0 控制寄存器(LPWMG0C)，地址= 0x0C 71
10.4.4.2. LPWMG1 控制寄存器 (LPWMG1C)，地址= 0x0D 71
10.4.4.3. LPWMG2 控制寄存器(LPWMG2C)，地址 = 0x0E 72
10.4.4.4. LPWMG 时钟寄存器(LPWMGCLK)，地址 = 0x67 72
10.4.4.5. LPWMG 计数上限高位寄存器(LPWMGCUBH)，地址 = 0x68 73
10.4.4.6. LPWMG 计数上限低位寄存器(LPWMGCUBL)，地址= 0x69 73
10.4.4.7. LPWMG0/1/2 占空比高位寄存器(LPWMGxDTH, x=0/1/2)，地址 = 0x6A/0x6C/ 73
10.4.4.8. LPWMG0/1/2 占空比低位寄存器(LPWMGxDTL, x=0/1/2)，地址 = 0x6B/0x6D/ 73
10.4.5. 带互补死区的 LPWM 波形范例 73
11. 特殊功能 76
11.1. 比较器 76
11.1.1. 比较器控制寄存器(*GPCC*)，地址= 0x35 77
11.1.2. 比较器选择寄存器(*GPCS*)，地址 = 0x36 77
11.1.3. 比较结果触发 PWM 控制寄存器(*GPC2PWM*)，地址 = 0x43 78
11.1.4. 内部参考电压 ($V_{internal\ R}$) 79
11.1.5. 使用比较器 81
11.1.6. 使用比较器和 Bandgap 参考电压生成器 82
11.2. VDD/2 偏置电压产生器 83
11.3. 运算放大器(OPA)模块 84
11.3.1. OPA 比较器模式 84
11.3.2. OPA 放大器模式 85
11.3.3. OPA 控制寄存器 (*OPAC*)，地址 = 0x33 85
11.3.4. OPA 失调寄存器 (*OPAOFS*)，地址 = 0x34 86
11.4. 模拟-数字转换器(ADC) 模块 86

11.4.1. AD 转换的输入要求 87
11.4.2. 选择参考高电压 88
11.4.3. ADC 时钟选择 88
11.4.4. 配置模拟引脚 88
11.4.5. 使用 ADC 89
11.4.6. ADC 相关寄存器 90
11.4.6.1. ADC 控制寄存器(ADCC), 地址 = 0x20 90
11.4.6.2. ADC 模式寄存器(ADCM), 地址 = 0x21 90
11.4.6.3. ADC 调节控制寄存器(ADCRGC), 地址 = 0x42 91
11.4.6.4. ADC 数据高位寄存器(ADCRH), 地址 = 0x4A 91
11.4.6.5. ADC 数据低位寄存器(ADCRL), 地址 = 0x4B 91
11.5. 乘法器 92
11.5.1. 乘法器运算对象寄存器 (*MULOP*), 地址 = 0x2C 92
11.5.2. 乘法器结果高字节寄存器 (*MULRH*), 地址 = 0x2D 92
11.6. 触摸功能 93
11.6.1. 触摸选项寄存器 (TS)，地址= 0x37 95
11.6.2. 外部晶体振荡器控制寄存器(EOSCR)，地址 = 0x0F 96
11.6.3. 触摸充电控制寄存器 (*TCC*)，地址 = 0x38 96
11.6.4. 触摸按键使能 3 寄存器 (*TKE3*)，地址 = 0x39 96
11.6.5. 触摸按键使能 2 寄存器 (*TKE2*)，地址 = 0x3A 97
11.6.6. 触摸按键使能 1 寄存器 (*TKE1*)，地址 = 0x3B 97
11.6.7. 触摸按键充电计数高位寄存器 (*TKCH*)，地址= 0x7A 97
11.6.8. 触摸按键充电计数低位寄存器 (*TKCL*)，地址 = 0x7B 97
11.6.9. 触摸参数设置寄存器(*TPS*)，IO 地址 = 0x3C 97
11.6.10. 触摸参数设置寄存器 2 (*TPS2*)，IO 地址 = 0x3D 98
11.7. 可编程频率产生器 (PFG) 98
11.7.1. PFG 中心频率 99
11.7.2. PFG 输出端口 99
11.7.3. PFG 相关寄存器 100
11.7.3.1. Delta IHRC 微调高位元寄存器(*PFGRH*), 地址 = 0x30 100
11.7.3.2. Delta IHRC 微调低位元寄存器(*PFGRL*), 地址 = 0x31 100
11.7.3.3. PFG 控制寄存器(*PFGC*), 地址 = 0x32 100
12. 仿真注意事项 100
13. 烧录方法 101
13.1. 普通烧录模式 101
13.2. 在板烧录模式（On-Board Writing） 102
14. 直流交流电气特性 103
14.1. 绝对最大值 103
14.2. 器件电气特性 103
14.3. ILRC 频率与 VDD 关系曲线图 106
14.4. NILRC 频率与 VDD 关系曲线图 106
14.5. IHRC 频率与 VDD 关系曲线图（校准到 16MHz） 107
14.6. PFG 频率与 VDD 关系曲线图（校准到 36MHz） 107
14.7. ILRC 频率与温度关系曲线图 108
14.8. NILRC 频率与温度关系曲线图 108
14.9. IHRC 频率与温度关系曲线图（校准到 16MHz） 109

14.10. PFG 频率与温度关系曲线图（校准到 36MHz） 109
14.11. 工作电流与 VDD、系统时钟 CLK=ILRC/n 曲线图 110
14.12. 工作电流与 VDD、系统时钟 CLK=IHRC/n 曲线图 110
14.13. 工作电流与 VDD、系统时钟 CLK=32KHz EOSC/n 曲线图 111
14.14. 工作电流与 VDD、系统时钟 CLK=1MHz EOSC/n 曲线图 111
14.15. 工作电流与 VDD、系统时钟 CLK=4MHz EOSC/n 曲线图 112
14.16. 引脚输出驱电流(I_{OH})与灌电流(I_{OL}) 曲线图 112
14.17. 引脚输入高电压与低电压(V_{IH}/V_{IL}) 曲线图 114
14.18. 引脚上拉/下拉电阻曲线图 115
14.19. 掉电电流(IPD)与省电电流(IPS) 曲线图 116
15. 指令 117
14.20. 指令表 118
14.21. 算术运算类指令 121
14.22. 移位运算类指令 123
14.23. 逻辑运算类指令 124
14.24. 位运算类指令 126
14.25. 条件运算类指令 127
14.26. 系统控制类指令 129
14.27. 指令执行周期综述 131

修订历史

修 订	日 期	描 述
0.06	2024/09/20	1. 调整器件电气特性的数值 2. 更新 14.4 和 14.8 温飘曲线图
0.07	2025/12/10	更新第 15 章节的叙述
0.08	2026/01/23	1. 第 1.2 节：新增说明「当 VDD ≥ 3.0 V 时，所有 PWMG 源时钟频率限制为 32MHz」 2. 第 14.2 节：新增符号 fpwm 的说明

使用警告

- 芯片所有 IO 引脚处不能有过压输入（大于芯片 VDD 电压），会对触摸造成干扰导致触摸异常。
- 在使用 IC 前，请务必认真阅读 PFC460 相关的 APN（应用注意事项）。

  请至官网下载查看与之关联的最新 APN 资讯：

  http://www.padauk.com.tw/cn/product/show.aspx?num=160&kw=460

  （以下附图仅供参考。）

◆◆ PFC460 ◆◆

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

Feature Documents Software &Tools Application Note

内容	說明	中文下载	英文下载
APN001	ADC仿真信号源输出阻抗应用须知		
APN002	过压保护应用须知		
APN003	IO输出引脚连接长导线时的应用须知		
APN004	半自动烧录机台使用需知		
APN005	过电压输入对ADC的影响使用须知		
APN007	设置LVR时的使用须知		
APN011	半自动烧录机台提高烧录稳定性		
APN013	晶振使用须知		
APN015	电容式触摸按键面板PCB设计须知		
APN017	提升IC在电源插拔测试下的抗干扰能力		
APN019	E-PAD 产品的PCB布局指南		

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

- 4KW MTP 程序空间供四个 FPPA 单元使用（可编程 1000 次）
- 512 Bytes SRAM 数据空间供四个 FPPA 单元使用
- 一个硬件 16 位定时器
- 两个 8 位硬件 PWM 生成器 Timer2/Timer3，Timer2/Timer3 还配置 NILRC 振荡器，它的频率比 ILRC 更慢，适合做更省电的唤醒时钟
- Timer2/Timer3 PWM 分辨率可以是 6/7/8bit
- 三个 11 位硬件 PWM 生成器 PWMG0/PWMG1/PWMG2
  (注意：**仅当 VDD ≥ 3.0 V 时，所有 PWMG 源时钟频率才能设置为 32 MHz。)**
- 一组可设三路 11 位 SuLED（Super LED）PWM 生成器和计数器 LPWMG0/LPWMG1/LPWMG2
- 提供一个 PFG 硬件电路支持精准的调频输出
- 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
- 26 个 IO 引脚，都具有可选的上拉/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
- 最大 24 IO 引脚可选择为触摸按键
- 内置 LDO 硬件电路为触摸功能提供 2V 电压基准
- 内部 Bandgap 电路提供 1.2V 参考电压
- 最大 28 通道（包含 GND）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：内部高频振荡器（IHRC）、内部低频振荡器（ILRC）、外部晶体震荡器（EOSC）
- 提供四组 IO 驱动能力以满足不同的应用需求
  (1) PB0 驱动电流可选 0/10mA，灌电流可选 108/20mA
  (2) PB2~PB7 驱动电流可选 28/10mA，灌电流可选 75/20mA
  (3) PA0~PA4 驱动电流/灌电流= 10mA/ 20mA
  (4) PA5~PA7，PB1，PC0~PC7，PD0~PD1 驱动电流/灌电流= 10mA/ 14mA
- 内建 VDD/2 偏置电压产生器，可支持 5COM ×21SEG 点阵的 LCD 屏
- 14 段 LVR 复位设定，从 2.0V 到 4.5V
- 8 个可选外部中断输入脚

1.3. CPU 特点

- 工作模式：四个 FPPA™ 处理单元运作模式或单一处理单元运作模式
- 104 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、SRAM 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

- PFC460-S08: SOP8 (150mil);
- PFC460-S14: SOP14 (150mil);
- PFC460-S16: SOP16 (150mil);
- PFC460-S20: SOP20 (300mil);
- PFC460-H20: HTSOP20 (150mil);
- PFC460-T20: TSSOP20 (173mil);
- PFC460-S24: SOP24 (300mil);
- PFC460-Y24: SSOP24 (150mil);
- PFC460-S28: SOP28 (300mil);
- PFC460-T28: TSSOP28 (173mil);

- 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

PFC460 是一个带 Touch、带 ADC、并行处理、完全静态，以 MTP 为程序存储基础的处理器，此处理器具有四个处理单元。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。

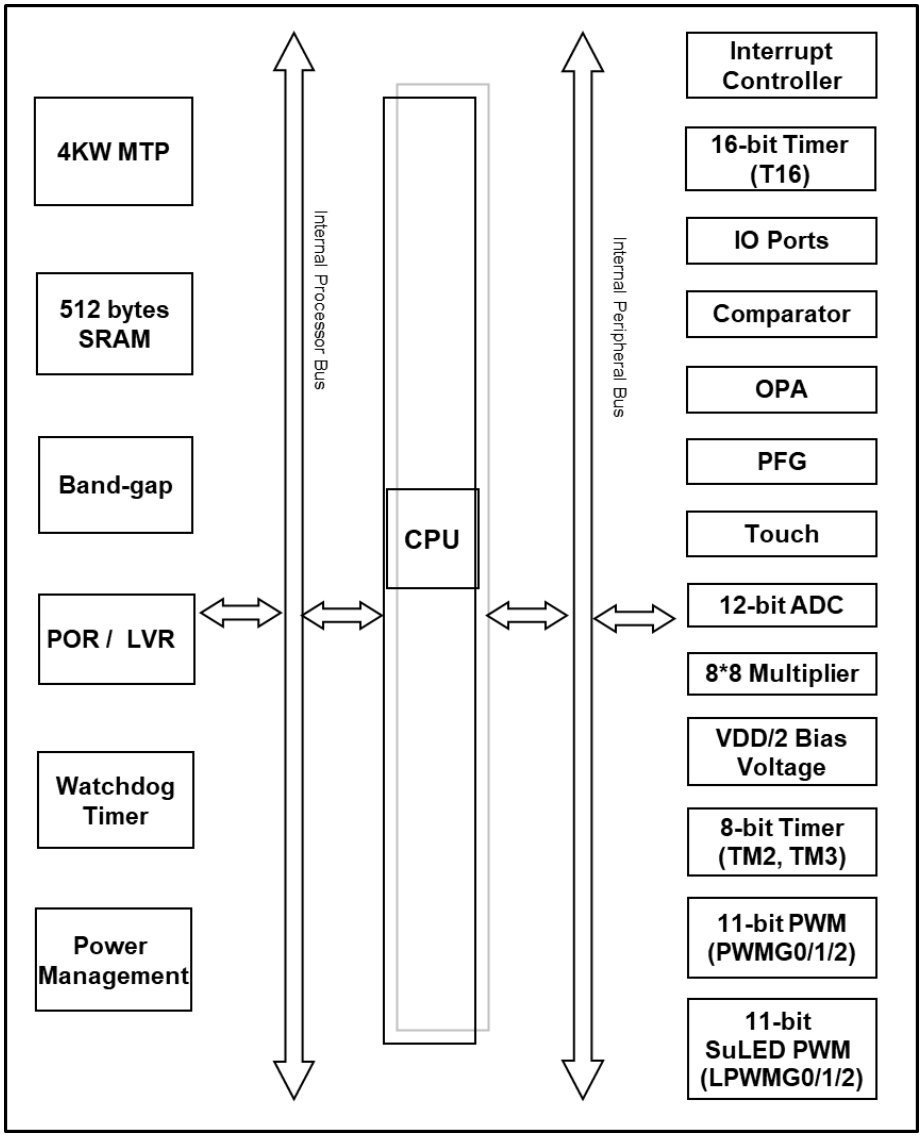
PFC460 内置 4KW MTP 程序存储器以及 512 字节 SRAM 数据存储器。

PFC460 内置 28 通道(包含 GND)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。

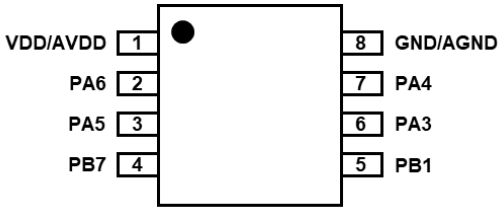
PFC460 内置 C-Touch 硬件电路，最大可提供 24 IO 作为触摸按键。

PFC460 提供一个 16 位的硬件计数器(Timer16)、两个 8 位计数器(Timer2、Timer3)、3 个 11 位计数器(PWMG0/1/2)和一组新的三路 11 位计数器带 SuLED PWM 生成器(LPWMG0/1/2)。

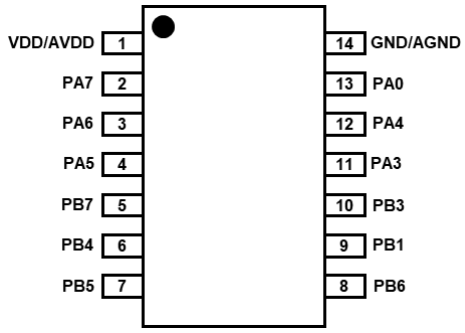
PFC460 还提供一个 PFG 调频电路、运算放大器（OPA）、硬件比较器、驱动 LCD 的 VDD/2 偏置电压生成器以及加强硬件运算功能的 8×8 乘法器。



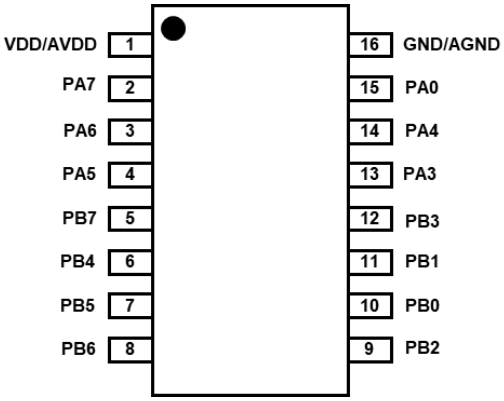
3. 引脚分配及功能说明



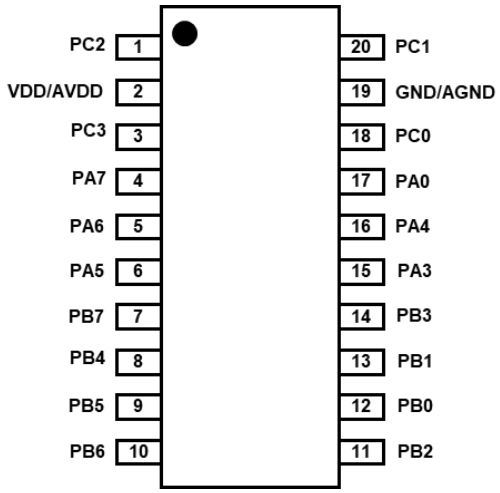
PFC460-S08: SOP8 (150mil)



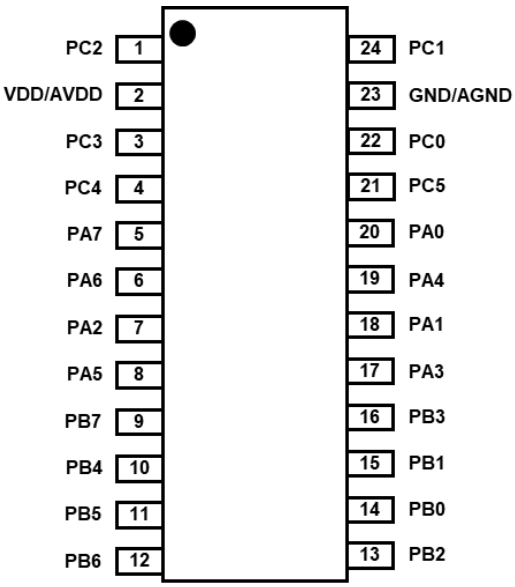
PFC460-S14: SOP14 (150mil)



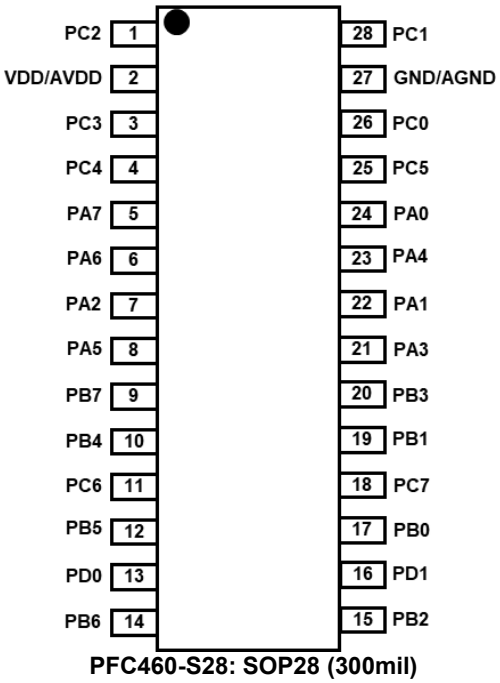
PFC460-S16: SOP16 (150mil)



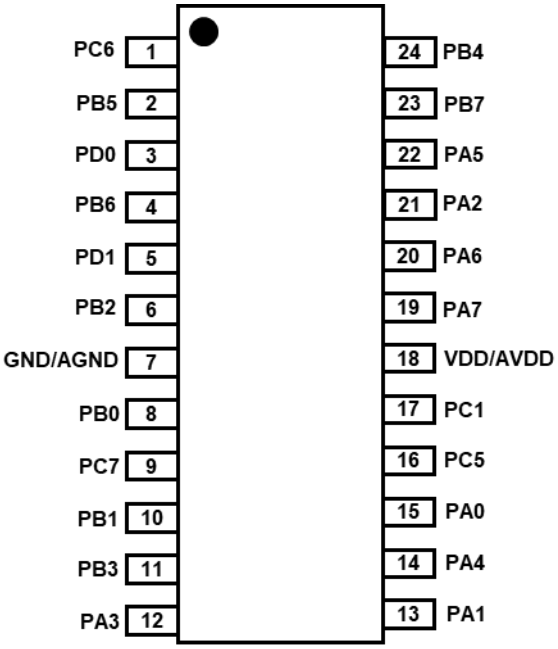
PFC460-S20: SOP20 (300mil)
PFC460-H20: HSOP20 (300 mil)
PFC460-T20: TSSOP20



PFC460-S24: SOP24 (300mil)



PFC460-S28: SOP28 (300mil)
PFC460-T28: TSSOP28



PFC460-Y24: SSOP24 (150mil)

注意: **PFC460** 的引脚排位与 **S24** 脚位互不相容

引脚名称	I/O	上/下拉	晶振	VDD/2	PWM	PFG	触摸	CS 电容	ADC	比较器	OPA	外部中断	外部复位	烧录
PA0	√	H/L		COM2	PWMG0/ LPWMG0		TK7		AD10	CO	OPO	INT0		
PA1	√	H/L					TK13		AD16					
PA2	√	H/L					TK14		AD17			INT0		
PA3	√	H/L		COM2	TM2PWM/ PWMG2/ LPWMG2		TK5		AD8	CIN-	OPIN-	INT1		√
PA4	√	H/L		COM2	PWMG1/ LPWMG1		TK6		AD9	CIN+/ CIN-	OPIN+/ OPIN-	INT1		
PA5	√	H/L			LPWMG2		TK17	CS1	AD18				√	√
PA6	√	H/L	Xout			√	TK8		AD19					√
PA7	√	H/L	Xin					CS0	AD20			INT0		
PB0	√	H/L		COM1/ COM2	TM2PWM	√	TK11		AD0			INT1		
PB1	√	H/L		COM1/ COM2			TK12		AD1/ Vref					
PB2	√	H/L		COM1	TM2PWM/ PWMG2/ LPWMG2		TK9		AD2					
PB3	√	H/L			PWMG2/ LPWMG2		TK10		AD3					
PB4	√	H/L			TM2PWM/ PWMG0/ LPWMG0	√	TK1		AD4					
PB5	√	H/L		COM1	TM3PWM/ PWMG0/ LPWMG0/ LPWMG2		TK2		AD5			INT0		

引脚名称	I/O	上/下拉	晶振	VDD/2	PWM	PFG	触摸	CS 电容	ADC	比较器	OPA	外部中断	外部复位	烧录
PB6	√	H/L		COM1	TM3PWM/ PWMG1/ LPWMG1/ LPWMG0		TK3		AD6	CIN-	OPIN-	INT1		
PB7	√	H/L			TM3PWM/ PWMG1/ LPWMG1	√	TK4		AD7	CIN-	OPIN-			
PC0	√	H/L			PWMG2/ LPWMG2		TK15		AD11					
PC1	√	H/L				√	TK18		AD12					
PC2	√	H/L			PWMG0/ LPWMG0		TK16		AD21					
PC3	√	H/L			PWMG1/ LPWMG1		TK19		AD22					
PC4	√	H/L					TK20		AD23					
PC5	√	H/L					TK21		AD24					
PC6	√	H/L					TK22		AD25					
PC7	√	H/L					TK23		AD26					
PD0	√	H/L					TK0		AD27					
PD1	√	H/L							AD28					
VDD/ AVDD														√
GND/ AGND														√

引脚简要说明：（其他未尽事宜，请参考 I/O 端口相关章节）

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. PA5 不再是开漏输出。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。
3. 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。
4. 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。
5. 当某引脚作为 PWM 输出端口时，其 I/O 功能自动停用。

4. 中央处理器 (CPU)

4.1. 功能描述

PFC460 内有四个处理单元：FPPA0 ~ FPPA3，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

FPPA0 ~ FPPA3 共享 4KWx16bits MTP 程序存储器，512 bytes 数据 SRAM 以及所有的 IO 口，这四个 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 ~ FPPA3 相对应的周期。图 1 所示为 FPPA 单元硬件框图。

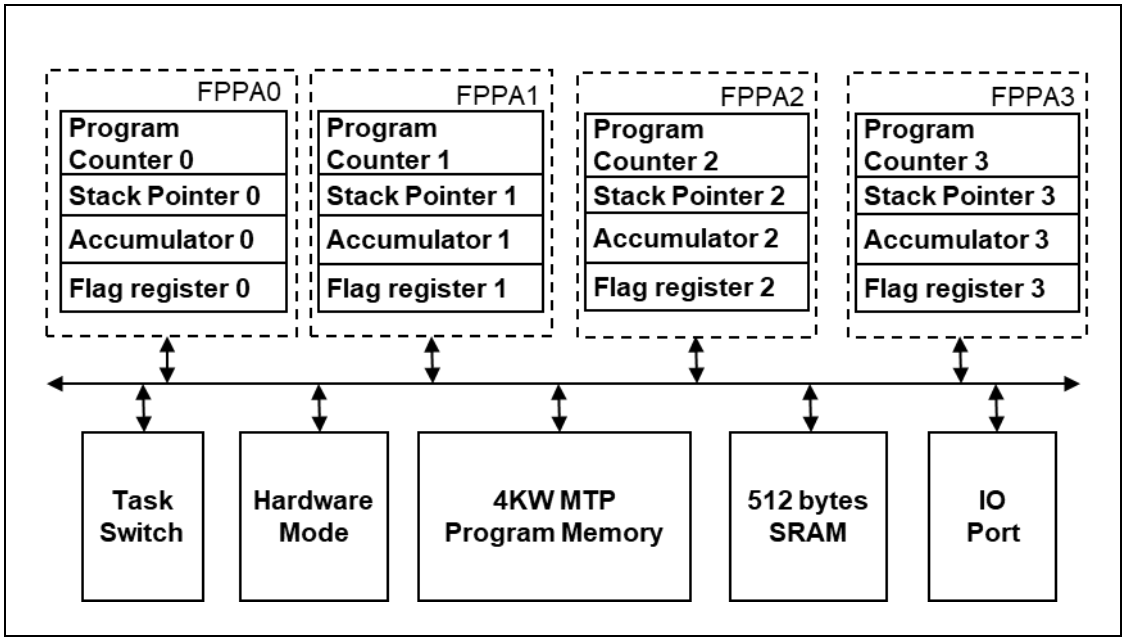


图 1：FPPA 单元架构

四个 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，可参阅指令表中关于 *pmode* 的说明。带宽分配与 FPPA enable 与否无关，即使 FPPA 被禁用，带宽也会被分配给指定的 FPPA 单元。

各 FPPA 单元可以通过 FPPA 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 ~ FPPA3 可以由用户的程序来决定是否启用。FPPA0 ~ FPPA3 可以被任一个 FPPA 停用，包括停用本身这一 FPPA 单元。

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 2 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。

对于 FPP0、FPP1、FPP2 和 FPP3 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。

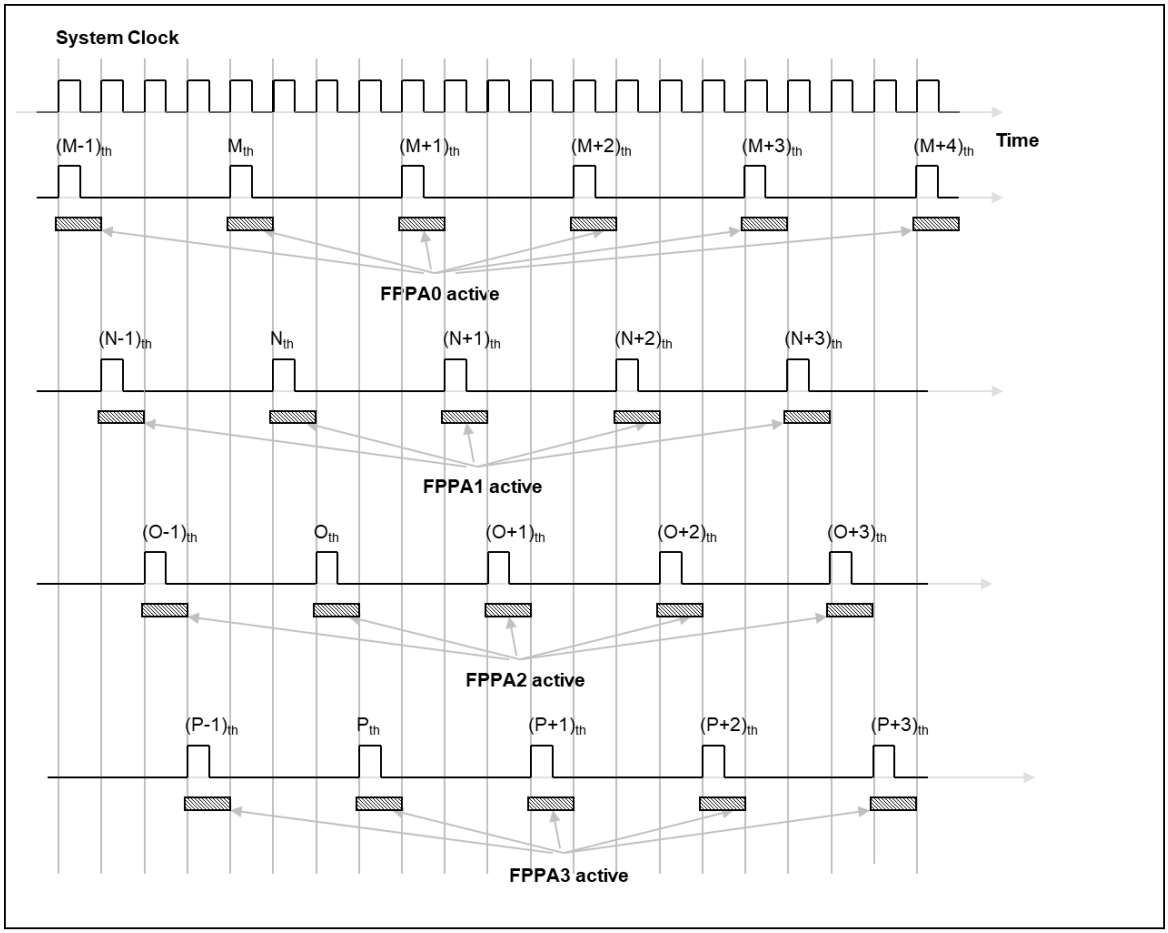


图 2：4-FPPA 单元架构以及基本时序

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 3 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：FPP1 在第（N-1），第 N，……第（N+3）时钟周期执行程序。

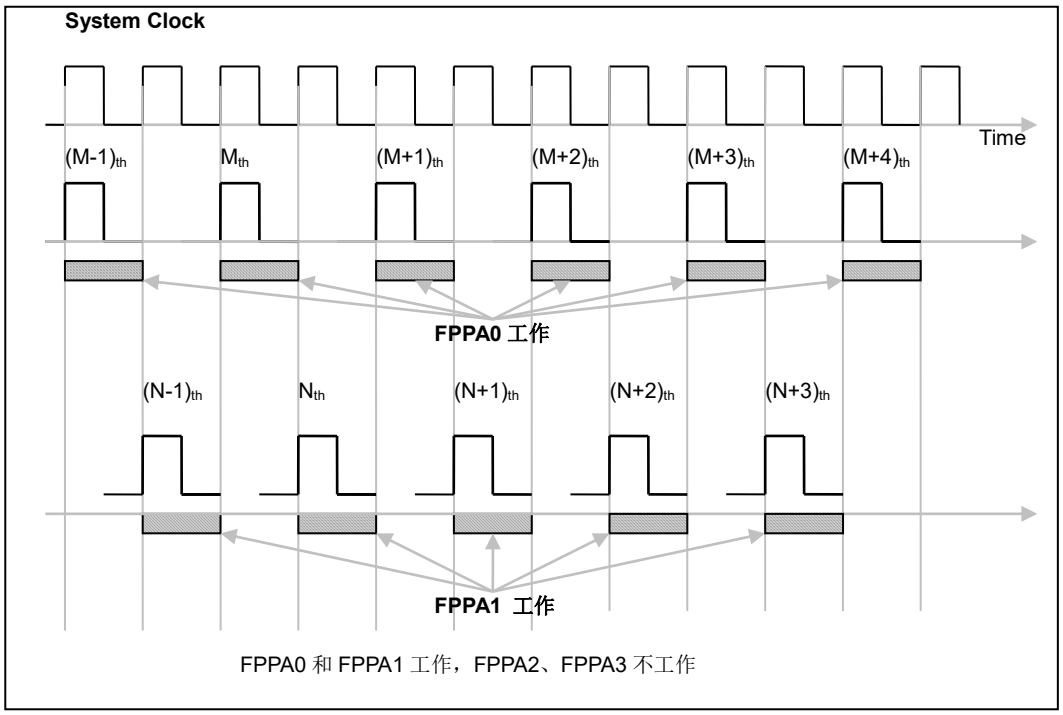


图 3：2-FPPA 单元架构以及基本时序

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，PFC460 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 ~ FPPA3 始终停用，只有 FPPA0 是使能的。图 4 显示了每个 FPPA 单元的时序图，FPPA1/2/3 总是停用，只 FPPA0 活跃。

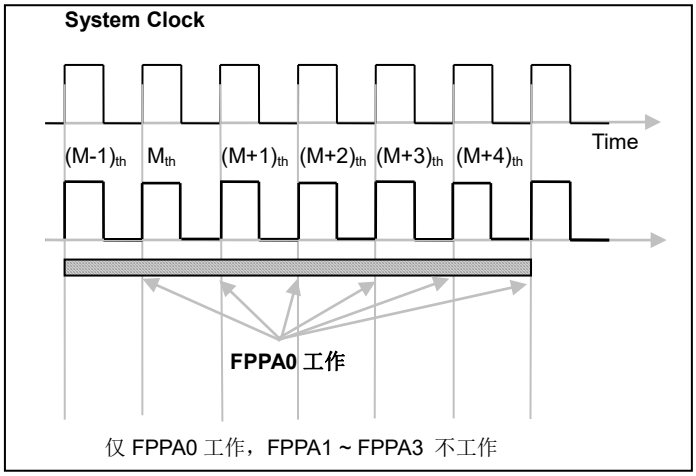


图 4：1-FPPA 单元架构以及基本时序

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。PFC460 程序计数器的位宽是 16。在硬件复位后，FPPA0 的程序计数器为 0、FPPA1 为 1、FPPA2 为 2、FPPA3 为 3。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 ~ FPPA3 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

四个处理单元工作模式下程序结构

开机后，FPPA0 ~ FPPA3 的程序开始地址分别是 0x000、0x001、0x002、0x003。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。 PFC460 的基本软件结构如图 5 所示。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 5：四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。

4.1.4. 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。FPPA0 ~ FPPA3 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)

PFC460 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。PFC460 的程序存储器容量为 4K words，所有 FPPA 单元的用户程序代码都存储在 4KW MTP ROM 中，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，FPPA2 为 0x002，FPPA3 为 0x003。中断入口在 0x010，只有 FPPA0 能使用中断功能。

MTP 存储器从地址“0xFE0 to 0xFFF”供系统使用，从“0x004 ~ 0x00F”和“0x011~0xFDF”地址空间是用户的程序空间。

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	FPPA2 起始地址 – *goto* 指令
0x003	FPPA3 起始地址 – *goto* 指令
0x004	使用者程序区
•	•
0x00F	使用者程序区
0x010	中断入口地址
0x011	使用者程序区
•	•
0xFDF	使用者程序区
0xFE0	系统使用
•	•
0xFFF	系统使用

表 1：PFC460 程序存储器结构

四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。

表 2 显示了一个例子，使用四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 – *goto* 指令(*goto* 0x400)
0x002	FPPA2 程序开始 – *goto* 指令(*goto* 0x5A8)
0x003	FPPA3 程序开始
•	•
0x00F	*goto* 0xC00 继续 FPPA3 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
•	•
0x3FF	FPPA0 程序结束
0x400	FPPA1 程序开始
•	•
0x5A7	FPPA1 程序结束
0x5A8	FPPA2 程序开始
•	•
0xBFF	FPPA2 程序结束
0xC00	继续 FPPA3 程序
•	•
0xFDF	使用者程序区结束
0xFE0	系统使用
•	•
0xFFF	系统使用

表 2：四个处理单元工作模式之程序存储器分配案例

两个处理单元工作模式下程序存储器分配例子

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
0x003	保留
0x004	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
•	•
0xFDF	不使用
0xFE0	系统使用
•	•
0xFFF	系统使用

表 3：两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 4 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
•	•
0x00F	*goto* 指令(*goto* 0x020)
0x010	中断入口地址
•	• 中断程序
0x01F	中断程序结束(取决于用户程序大小)
0x020	使用者程序区
•	•
0xFDF	使用者程序区
0xFE0	系统使用
•	•
0xFFF	系统使用

表 4：一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 6 显示了 PFC460 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 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存取方法的数据指针和所有 FPPA 单元的堆栈记忆体。

FPPA0 ~ FPPA3 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，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，PFC460 的 512 个字节数据存储器都可以利用间接存取指令来存取。

位寻址可定义在 RAM 区的 0x00 到 0x1FF 全空间，然而寄存器地址空间内的位寻址操作只可定义在 0x00 到 0x3F 空间。

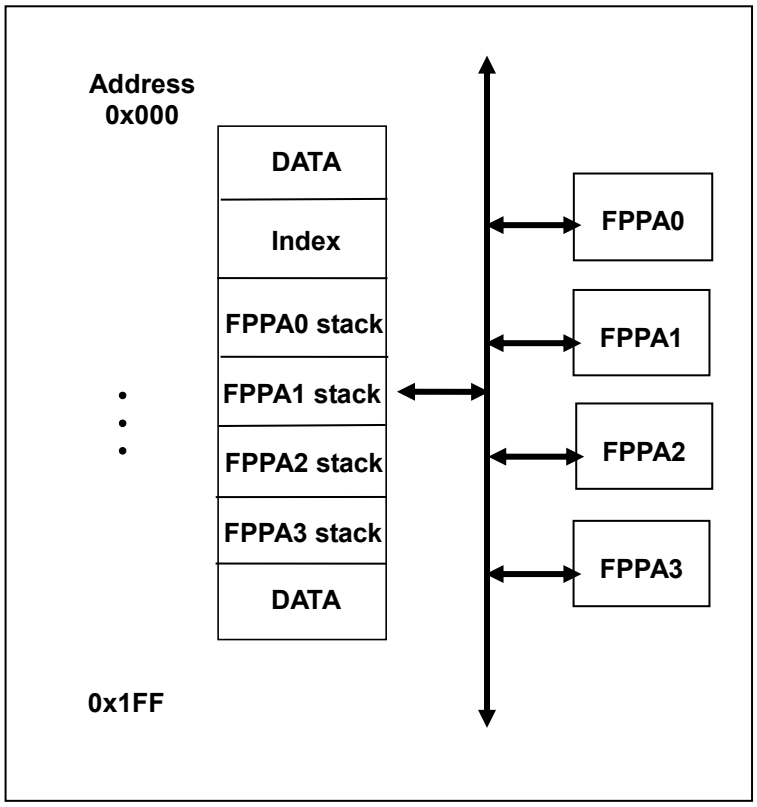


图 6：数据存储器结构和使用

4.2.3. 系统寄存器

PFC460 的寄存器地址空间与 SRAM 数据存储空间、MTP 程序空间三者互相独立。

以下是 PFC460 的各寄存器存放地址，至于用法及说明请参考其各自相关章节。

	+0	+1	+2	+3	+4	+5	+6	+7
0x00	*FLAG*	*FPPEN*	*SP*	*CLKMD*	*INTEN*	*INTRQ*	*INTEN2*	*INTRQ2*
0x08	*T16M*	-	-	-	*LPWMG0C*	*LPWMG1C*	*LPWMG2C*	*EOSCR*
0x10	*PA*	*PAC*	*PAPH*	*PAPL*	*PB*	*PBC*	*PBPH*	*PBPL*
0x18	*PC*	*PCC*	*PCPH*	*PCPL*	*PD*	*PDC*	*PDPH*	*PDPL*
0x20	*ADCC*	*ADCM*	*PWMG0C*	*PWMG0S*	*PWMG1C*	*PWMG1S*	*PWMG2C*	*PWMG2S*
0x28	*TM2C*	*TM2CT*	*TM3C*	*TM3CT*	*MULOP*	*MULRH*	-	-
0x30	*PFGRH*	*PFGRL*	*PFGC*	*OPAC*	*OPAOFS*	*GPCC*	*GPCS*	*TS*
0x38	*TCC*	*TKE3*	*TKE2*	*TKE1*	*TPS*	*TPS2*	-	-
0x40	-	-	*ADCRGC*	*GPC2PWMG*	-	-	-	-
0x48	-	*MISC*	*ADCRH*	*ADCRL*	*PADIER*	*PBDIER*	*PCDIER*	*PDDIER*
0x50	*PWMG0DTH*	*PWMG0DTL*	*PWMG0CUBH*	*PWMG0CUBL*	*PWMG1DTH*	*PWMG1DTL*	*PWMG1CUBH*	*PWMG1CUBL*
0x58	*PWMG2DTH*	*PWMG2DTL*	*PWMG2CUBH*	*PWMG2CUBL*	-	*INTEGS*	-	*IOHD*
0x60	*TM2S*	*TM2B*	*TM3S*	*TM3B*	-	-	-	*LPWMGCLK*
0x68	*LPWMCUBH*	*LPWMCUBL*	*LPWMG0DTH*	*LPWMG0DTL*	*LPWMG1DTH*	*LPWMG1DTL*	*LPWMG2DTH*	*LPWMG2DTL*
0x70	-	-	-	-	-	-	-	-
0x78	-	-	*TKCH*	*TKCL*	-	-	-	-

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 – 4	-	-	保留。
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. 杂项寄存器(*MISC*)，地址 = 0x49

位	初始值	读/写	描 述
7	-	-	保留。请保持为 0。
6	-	只写	保留，请手动设置为 1。
5	0	只写	快唤醒功能。快速唤醒功能 EOSC 模式下不支持。 0：正常唤醒。唤醒时间是 2048 个 ILRC 时钟（不适用快速开机）。 1：快速唤醒。唤醒时间为 32 个 ILRC 时钟。
4	0	只写	使能 VDD/2 偏置电压产生器： 0 / 1：停用 / 启用
3	0	只写	VDD/2 偏置电压输出脚位选择： 0: LCD_B01256，包含 PB0/PB1/PB2/PB5/PB6 1: LCD_A034_B01,包含 PA0/PA3/PA4/PB0/PB1
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 256 字节以内的偶数位置，不可以超过 256 字节。PFC460 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 FPPA0 ~ FPPA3 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)
{
        ...
}
```

使用者可以在程序分解的窗口里检查堆栈的设定，图 7 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。

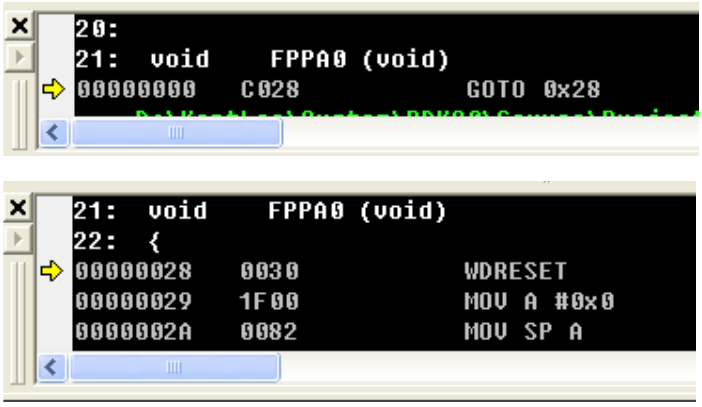


图 7：使用 Mini-C 的堆栈设定

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	14 级	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
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
	PA.2	*INTEN/ INTRQ*.位 0 来自于 PA.2
	PA.7	*INTEN/ INTRQ*.位 0 来自于 PA.7
Interrupt Src1	**PA.4（默认）**	*INTEN/ INTRQ*.位 1 来自于 PA.4
	PA.3	*INTEN/ INTRQ*.位 1 来自于 PA.3
	PB.0	*INTEN/ INTRQ*.位 1 来自于 PB.0
	PB.6	*INTEN/ INTRQ*.位 1 来自于 PB.6
TMx_Source	**16MHz（默认）**	当 *TMxC*[7:4]=0010, TMx 时钟源 = 16 MHz (x=2/3)
	32MHz	当 *TMxC*[7:4]=0010, TMx 时钟源 = 32 MHz (x=2/3)
TMx_Bit	**6 Bit（默认）**	当 *TMxS*.7 = 1,TMx PWM 分辨率为 6 位 (x=2/3)
	7 Bit	当 *TMxS*.7 = 1,TMx PWM 分辨率为 7 位 (x=2/3)
TM2C[3:2]=1	**PB2（默认）**	*TM2C[3:2]*=01 时，输出脚选择为 PB2
	PB0	*TM2C[3:2]*=01 时，输出脚选择为 PB0
OPA_PWM	**Disable（默认）**	OPA / PWM 互相独立
	Enable	OPA 输出结果控制 PWM 波形

选项	选择	说明
PWM_Source	**16MHz（默认）**	当 *PWMGxC*.0 = 1,PWMGx 时钟源 = 16 MHz (x=0/1/2)
	32MHz	当 *PWMGxC*.0 = 1,PWMGx 时钟源 = 32 MHz (x=0/1/2)
LPWM_Source	**16MHz（默认）**	当 *LPWMGCLK*.0 = 1,LPWMG 时钟源 = 16 MHz
	32MHz	当 *LPWMGCLK*.0 = 1,LPWMG 时钟源 = 32 MHz
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
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
CS_Sel	**PA7（默认）**	将 PA7 设置为触摸 CS 引脚
	PA5	将 PA5 设置为触摸 CS 引脚
	Disable	关闭触摸 CS 引脚
EMI	**Disable（默认）**	停用 EMI 优化选项
	Enable	系统时钟会轻微调整以获得更好的 EMI 性能
FPPA	**1-FPPA（默认）**	单一 FPPA 处理单元模式
	4-FPPA	四核心 FPPA 处理单元模式
Burning_Enhancement	**Newer（默认）**	增强烧录效果（推荐使用）
	Older	兼容旧版本（0.97C5 之前）

5. 振荡器和系统时钟

PFC460 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)，可以分别用寄存器 *EOSCR*.7，*CLKMD*.4 与 *CLKMD*.2 来控制各振荡模块启用或停用。

使用者可以选择 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 5：PFC460 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。

PFC460 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，PFC460 不支持频率在 4MHz 以上的振荡器。图 8 显示了使用外部晶体振荡器的的硬件连接。

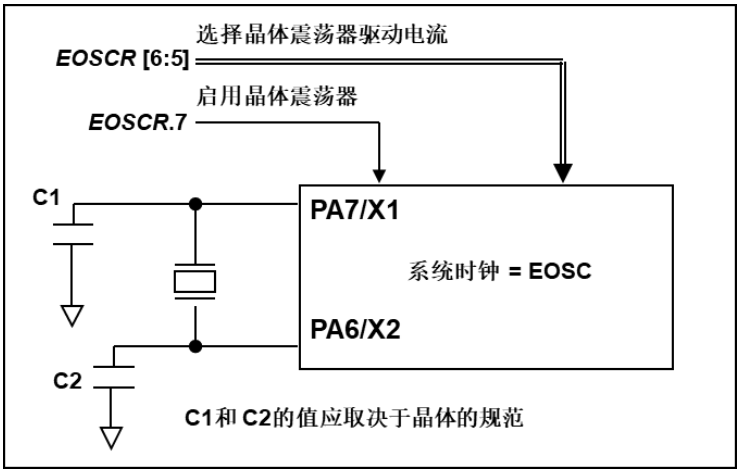


图 8：外部晶体振荡器的的硬件连接

5.2.1. 外部晶体振荡器控制寄存器(*EOSCR*)，地址 = 0x0F

位	初始值	读/写	描 述
7	0	只写	使能外部晶体振荡器。0 / 1：停用/启用
6 – 5	00	只写	晶体振荡器的选择。 00：保留 01：低驱动电流。适用于较低频率晶体，例如：32KHz 10：中驱动电流。适用于中等频率晶体，例如：1MHz 11：高驱动电流。适用于较高频率晶体，例如：4MHz
4	-	-	保留。
3	-	只写	LDO 输出电压选择。0/1：2.4V/2V
2	-	只写	触摸模块电源选择。0/1：VDD/LDO
1 – 0	-	-	保留。

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 6 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 6：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，PFC460 的时钟系统的硬件框图如图 9 所示。

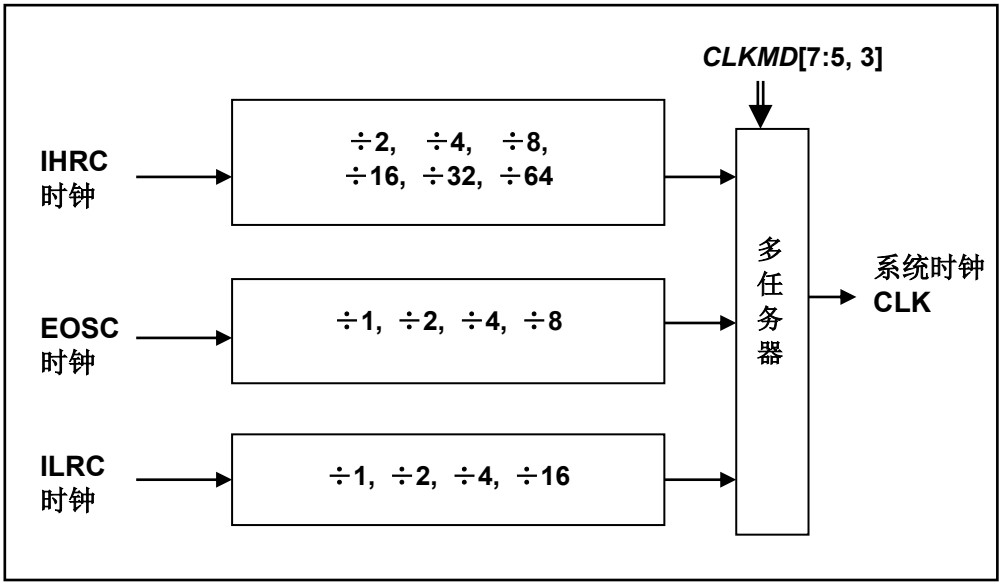


图 9：系统时钟源选择

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/16 011：IHRC/32 100：IHRC/64 101：EOSC/8 110：ILRC/2 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;

式中，**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 =3.9～5.5；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 7 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 7：IHRC 频率校准选项

下面显示在不同的选项下，PFC460 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
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）的特牲会对 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，PFC460 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” → “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 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 PFC460 复位的原因很多，一旦复位发生 PFC460 的所有寄存器将被设置为默认值，系统会重新启动，程序计数器会跳跃地址 0x00。

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

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

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

6.1. 上电复位(POR)

开机时，POR(Power On Reset)是用于复位 PFC460，开机时间大约 2048 ILRC，其时序图如图 10 所示。用户必须确保上电后电源电压稳定。

发生上电复位时，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。

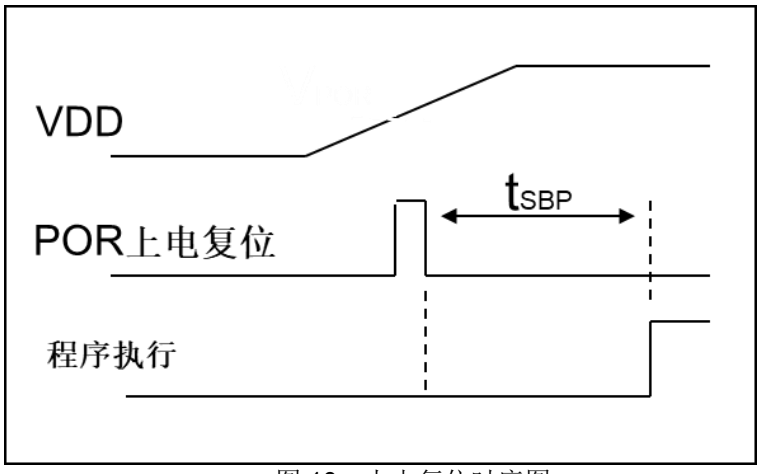


图 10：上电复位时序图

图 11 显示的是典型开机流程。请注意，上电复位后 FPPA1-FPPA3 是停用，建议不要在 FPPA0 以及系统初始化完成前，启用其他 FPPA 单元。

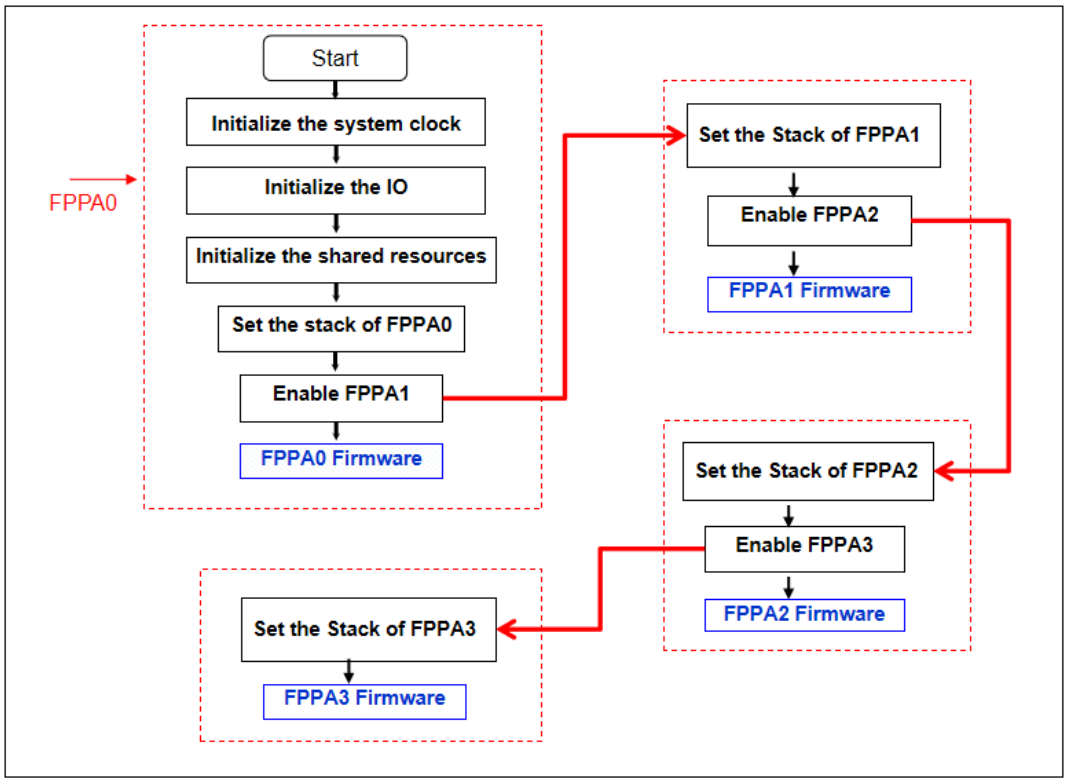


图 11：开机流程

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 12。

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。

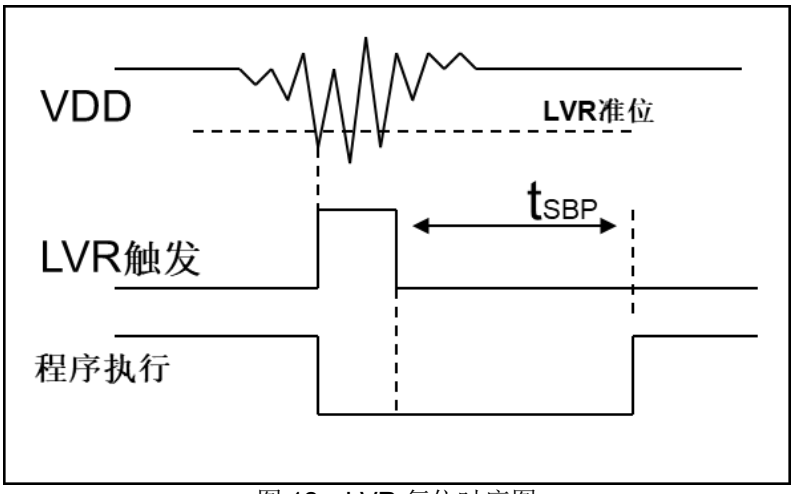


图 12：LVR 复位时序图

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	≧ 4.5V	4.5V
4MHz	≧ 2.7V	2.7V
2MHz	≧ 2.2V	2.2V

表 8：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（2.0V ~ 4.5V）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 = 0x49			
位	**初始值**	**读/写**	**描 述**
7	-	-	保留。请保持为 0。
6	0	只写	晶体振荡器驱动电流选择
5	0	只写	快唤醒功能
4	0	只写	使能 VDD/2 偏置电压产生器
3	0	只写	VDD/2 偏置电压输出脚位选择
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，PFC460 将复位并重新运行程序，其复位时序图如图 13 所示。发生 WDT 复位时，PFC460 数据存储器的值将被保留。

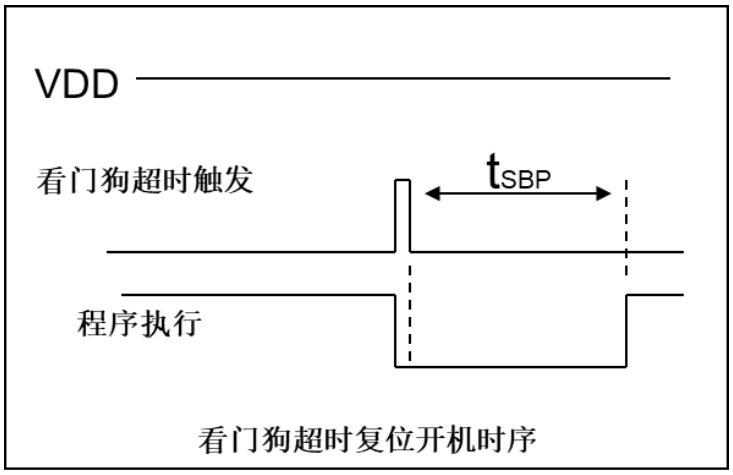


图 13：看门狗超时溢出的相关时序

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)

PFC460 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 14 所示。

当发生 PRSTB 复位时，PFC460 数据存储器的值将被保留。

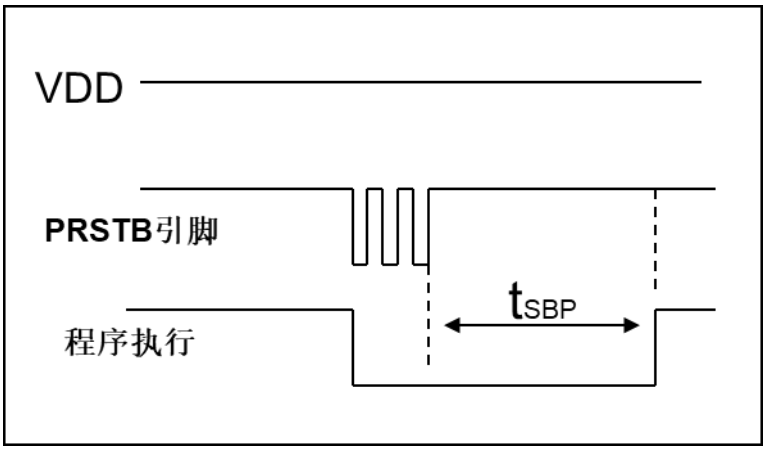


图 14：外部引脚复位的相关时序

7. 系统工作模式

PFC460 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 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，TM2/3，PWMG0/1/2，LPWMG）

(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，PFC460 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：

a. IO 在数字输入模式下电平变换（*PxDIER* 位是 1）。

b. *TM3C*.NILRC 唤醒：Timer3 的时钟源需选择 NILRC。

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，PFC460 可以通过切换 IO 引脚或 TM3C.NILRC 唤醒恢复正常工作。而其他 Timer 的唤醒只适用于省电模式。表 9 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	TM3C.NILRC 唤醒	其他 Timer 计时器唤醒
stopsys	是	是	否
stopexe	是	是	是

表 9：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 PFC460，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 2048ILRC 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 32 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	32* 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	2048* T_{ILRC}, 这里的 T_{ILRC} 是指 ILRC 时钟周期

表 10：快速唤醒与正常唤醒的时间差异

8. 中断

PFC460 有 11 个中断源，包含 2 个外部 IO 中断和 10 个内部中断，其中两外部中断源总计共有 8 个 IO 可供用户选择使用，具体参考如下：

- 外部中断源 PA0/PB5/PA2/PA7（由 code option Interrupt Src0 决定）
- 外部中断源 PB0/PA4/PA3/PB6（由 code option Interrupt Src1 决定）
- ADC 中断
- 比较器 COMP 中断
- Timer16 中断
- Timer2 中断
- Timer3 中断
- LPWM（SuLED PWM）中断
- PWMG0 中断
- TK_OV 中断
- TK_END 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 15。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/INTRQ2* 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/INTRQ2* 知道中断发生源。

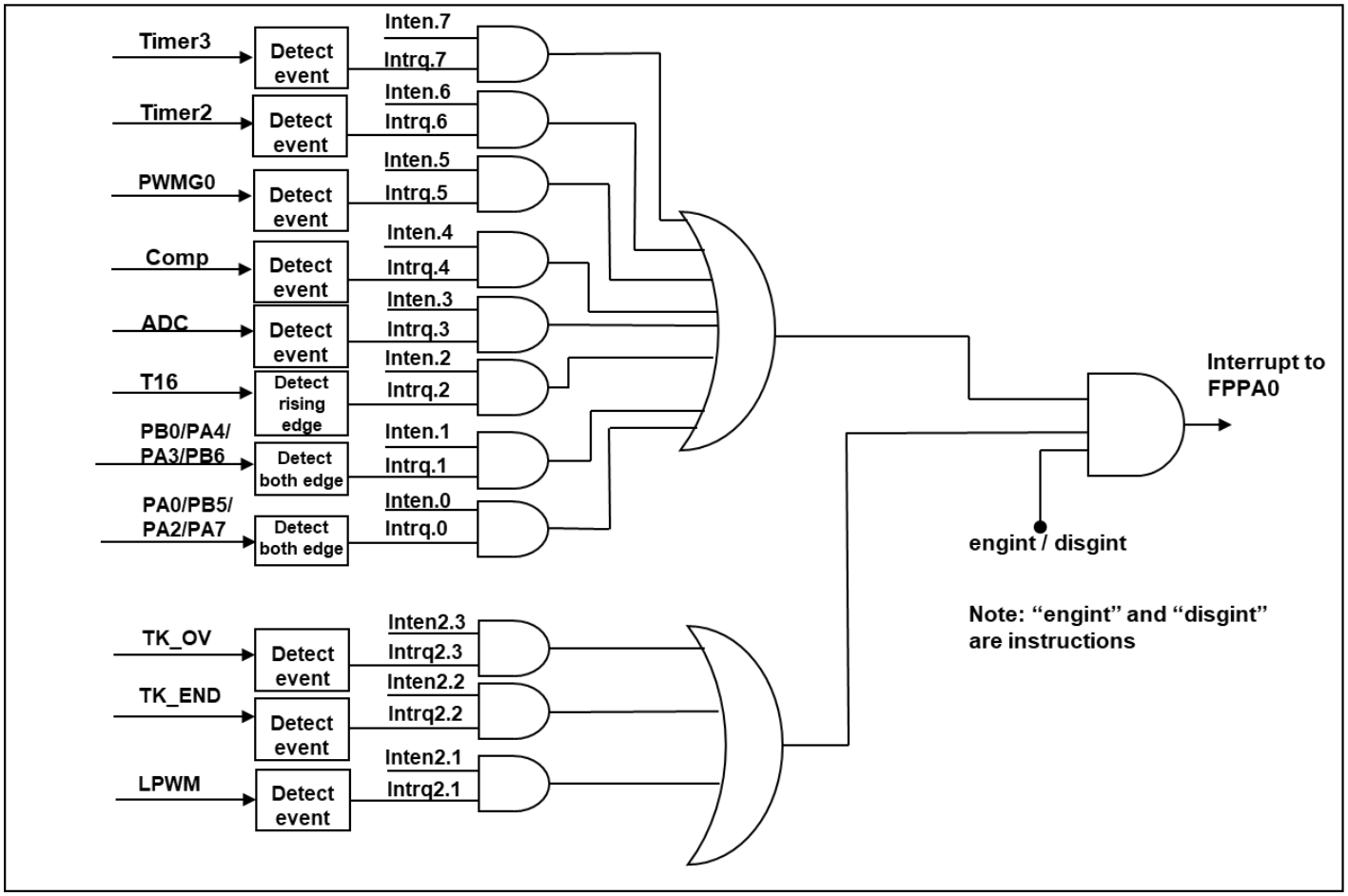


图 15：中断硬件框图

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(COMP)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/PA3/PB6 的中断。0/1：停用/启用 （由 code option Interrupt Src1 决定）
0	-	读/写	启用从 PA0/PB5/PA2/PA7 的中断。 0/1：停用/启用 （由 code option Interrupt Src0 决定）

8.2. 中断允许寄存器 2 (*INTEN2*)，地址 = 0x06

位	初始值	读/写	描 述
7 – 4	-	-	保留
3	-	读/写	启用从 TK_OV 的中断。0/1：停用/启用
2	-	读/写	启用从 TK_END 的中断。0/1：停用/启用
1	-	读/写	启用从 LPWM 的中断。0/1：停用/启用
0	-	-	保留

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	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/PA3/PB6 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/PA2/PA7 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

8.4. 中断请求寄存器 2 (*INTRQ2*)，地址 = 0x07

位	初始值	读/写	描 述
7 – 4	-	-	保留
3	-	读/写	TK_OV 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
2	-	读/写	TK_END 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求
1	-	读/写	LPWM 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	-	-	保留

8.5. 中断缘选择寄存器 (*INTEGS*), 地址 = 0x5D

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/PA3/PB6 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/PA2/PA7 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。

8.6. 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。

8.7. 中断的一般步骤

步骤 1：设定 *INTEN/INTEN2* 寄存器，开启需要的中断的控制位。

步骤 2：清除 *INTRQ/INTRQ2* 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。

8.8. 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 = 0x4C

位	初始值	读/写	描 述
7	1	只写	启用 PA7 数字输入、系统唤醒和中断请求。 1 / 0：启用 / 停用 当使用外部晶体振荡器时，该位应设置为低，以防止漏电。
6	1	只写	启用 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，该位应设置为低，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 – 2	111	只写	启用 PA4/PA3/PA2 数字输入、系统唤醒和中断请求。 1 / 0：启用 / 停用
1	1	只写	启用 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

9.1.2. 端口 B 数字输入启用寄存器(*PBDIER*)，地址 = 0x4D

位	初始值	读/写	描 述
7	1	只写	启用 PB7 数字输入和系统唤醒。 1 / 0：启用 / 停用
6 – 5	11	只写	启用 PB6~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 C 数字输入启用寄存器(*PCDIER*)，地址 = 0x4E

位	初始值	读/写	描 述
7 – 0	0xFF	只写	启用 PC7~PC0 数字输入和系统唤醒。 1 / 0：启用 / 停用

9.1.4. 端口 D 数字输入启用寄存器(*PDDIER*)，地址 = 0x4F

位	初始值	读/写	描 述
7 – 2	-	-	保留
1 – 0	11	只写	启用 PD1~PD0 数字输入和系统唤醒。 1 / 0：启用 / 停用

9.1.5. 端口 A/B/C 数据寄存器(*PA/PB/PC*)，地址 = 0x10/0x14/0x18

位	初始值	读/写	描 述
7 – 0	0x00	读/写	端口 A/B/C 的数据寄存器

9.1.6. 端口 A/B/C 控制寄存器(*PAC/PBC/PCC*)，地址 = 0x11/0x15//0x19

位	初始值	读/写	描 述
7 – 0	0x00	读/写	端口 A/B/C 控制寄存器。用来定义端口各引脚的输入或输出模式。 0 / 1：输入/输出

9.1.7. 端口 A/B/C 上拉控制寄存器(*PAPH/PBPH/PCPH*)，地址 = 0x12/0x16/0x1A

位	初始值	读/写	描 述
7 – 0	0x00	读/写	端口 A/B/C 上拉控制寄存器。用来控制端口各引脚上拉电阻的使能。 0/1：停用/启用

9.1.8. 端口 A/B/C 下拉控制寄存器(*PAPL/PBPL/PCPL*)，地址 = 0x13/0x17/0x1B

位	初始值	读/写	描 述
7 – 0	0x00	读/写	端口 A/B/C 下拉控制寄存器。用来控制端口各引脚下拉电阻的使能。 0/1：停用/启用

9.1.9. 端口 D 数据寄存器(*PD*)，地址 = 0x1C

位	初始值	读/写	描 述
7 – 2	-	-	保留
1 – 0	00	读/写	端口 D 的数据寄存器位[1:0]。

9.1.10. 端口 D 控制寄存器(*PDC*)，地址 = 0x1D

位	初始值	读/写	描 述
7 – 2	-	-	保留
1 – 0	00	读/写	端口 D 控制寄存器。用来定义端口 D 引脚 PD1~PD0 的输入或输出模式。 0/1：输入/输出

9.1.11. 端口 D 上拉控制寄存器(*PDPH*)，地址 = 0x1E

位	初始值	读/写	描 述
7 – 2	-	-	保留
1 – 0	00	读/写	端口 D 上拉控制寄存器。用来控制端口 D 引脚 PD1~PD0 上拉电阻的使能。 0/1：停用/启用

9.1.12. 端口 D 下拉控制寄存器(*PDPL*)，地址 = 0x1F

位	初始值	读/写	描 述
7 – 2	-	-	保留
1 – 0	00	读/写	端口 D 下拉控制寄存器。用来控制端口 D 引脚 PD1~PD0 下拉电阻的使能。 0/1：停用/启用

9.2.IO 结构及功能

9.2.1. IO 引脚的结构

PFC460 的几乎所有 IO 引脚都具有相同的结构，如下图 16。

PFC460 的 PA5 与其他 IO 一致，可作通用输入/输出引脚，非开漏输出。

关于驱动能力的描述，可参考下文 *IOHD* 寄存器相关说明。

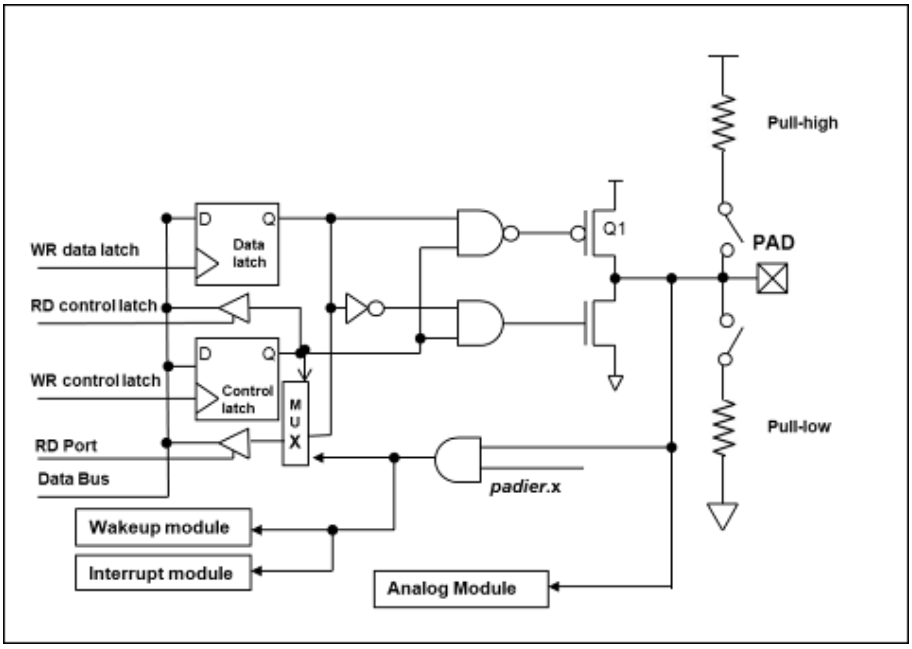


图 16：引脚缓冲区硬件图

9.2.2. IO 引脚的一般功能

(1) 输入、输出功能：

PFC460 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/PC/PD*)，控制寄存器(*PAC/PBC/PCC/PDC*)，上拉控制寄存器(*PAPH/PBPH / PCPH/PDPH*)和下拉控制寄存器(*PAPL/PBPL/PCPL/PDPL*)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 11 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 11：PA0 设定配置表

(2) 睡眠唤醒功能：

当 PFC460 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，必须设置为数字输入模式，寄存器 *PxDIER* 相应位应为高。

(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：

PFC460 的 PB0、PB2~PB7 作输出时有 Strong/Normal 两种驱动能力可选，用户可通过 *IOHD* 寄存器依实际需求灵活调整。

9.2.2.1. IO 输出驱动能力控制寄存器(*IOHD*)，地址 = 0x5F

位	初始值	读/写	描 述
7	0	只写	PB7 IOH/IOL 驱动能力选择。 0/1：Normal/Strong
6	0	只写	PB6 IOH/IOL 驱动能力选择。 0/1：Normal/Strong
5	0	只写	PB5 IOH/IOL 驱动能力选择。 0/1：Normal/Strong
4	0	只写	PB4 IOH/IOL 驱动能力选择。 0/1：Normal/Strong
3	0	只写	PB3 IOH/IOL 驱动能力选择。 0/1：Normal/Strong
2	0	只写	PB2 IOH/IOL 驱动能力选择。 0/1：Normal/Strong
1	-	-	保留
0	0	只写	PB0 IOH/IOL 驱动能力选择。 0/1：Normal/Strong

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 PD 中没有用到的 IO 口漏电，*PDDIER*[7:2]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。

(5) 为了让 IC 有良好的电气特性，请于尽可能靠近 IC VDD/GND 处加上 0.1uF 电容，同时建议再并联一个至少 10uF 电解电容。

10. Timer / PWM 计数器

10.1. 16 位计数器 (Timer16)

10.1.1. Timer16 介绍

PFC460 内置一个 16 位硬件计数器 Timer16，其模块框图如图 17。

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。

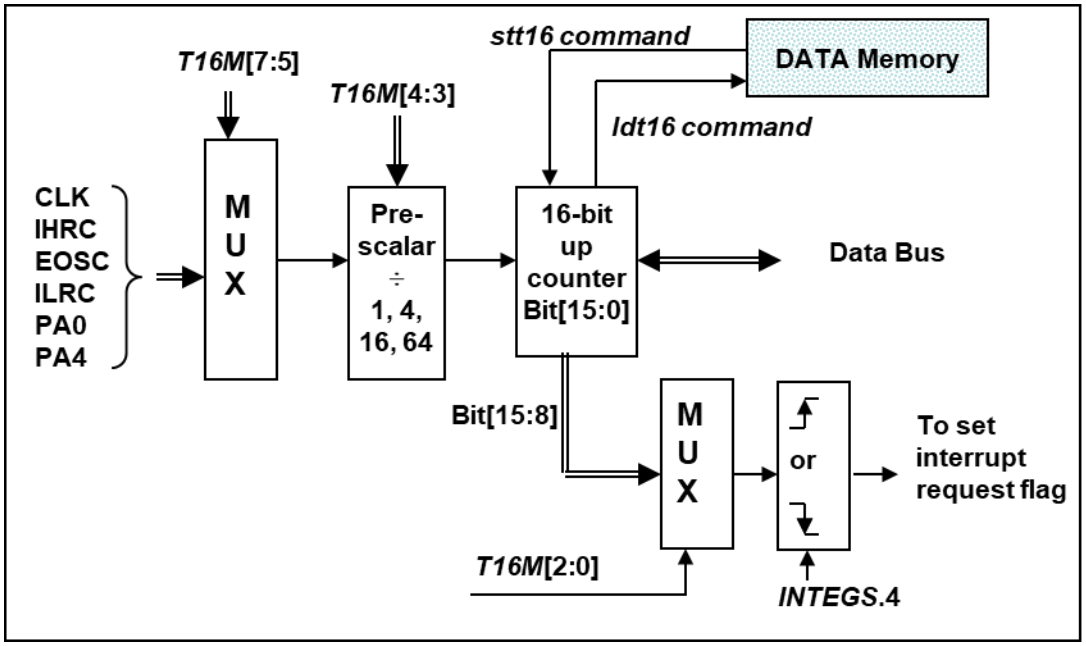


图 17：Timer16 模块框图

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

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

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 = 0x08

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)

PFC460 内置 2 个 8 位 PWM 硬件定时器(Timer2/TM2,Timer3/TM3)，两个计数器的原理类似，以下以 Timer2 来说明，TM2 硬件框图请参考图 18。

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。

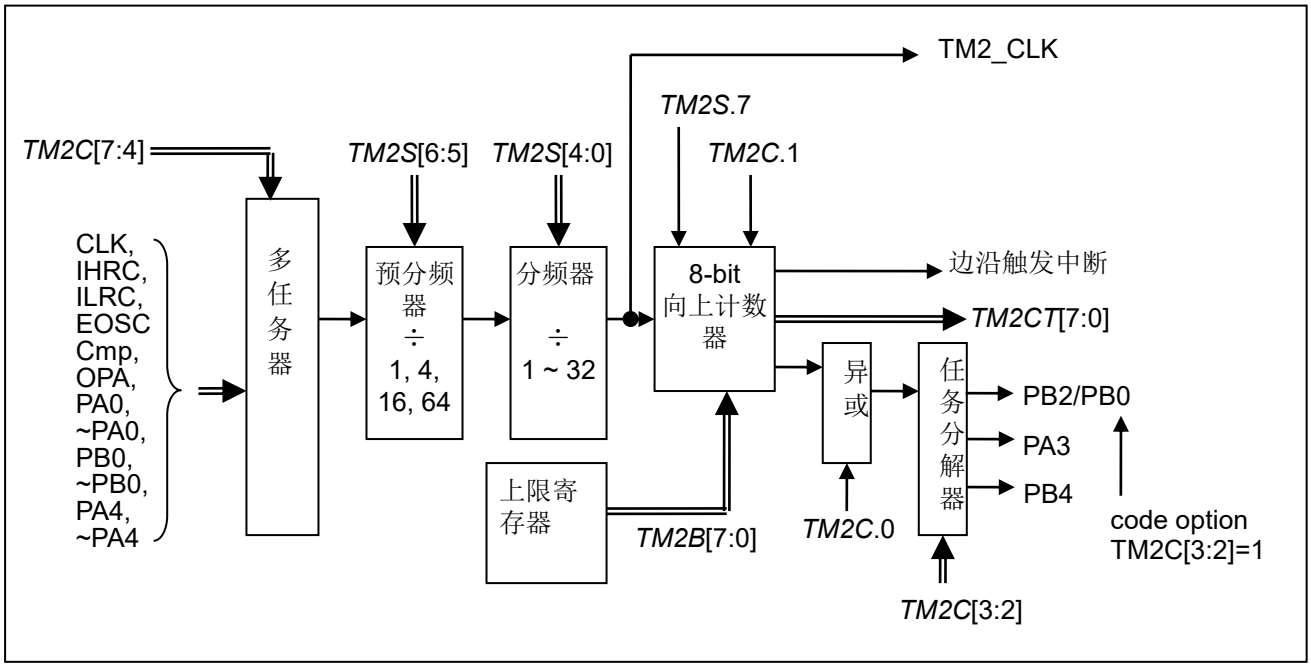


图 18：Timer2 模块框图

Timer3 的模块框图与 Timer2 类似，但其计数时钟源比 Timer2 多出一个 NILRC 振荡器，并且其 PWM 输出端口是在 PB5，PB6 或 PB7。

寄存器 *TM3C* 的位[7:4]可将时钟源选择为 NILRC，以支持更低功耗定时唤醒“stopexe”和“stopsys”。NILRC 振荡器是比 ILRC 更慢的时钟，用来做更省电的唤醒时钟，并且只为 Timer3 所用。NILRC 和 ILRC 都可通过 IHRC 估算频率，但 NILRC 的误差更大，所以需要先估算频率才可使用。若需相关 demo，请恰 FAE。

用户可以通过使用程序选项 OPA_PWM，使 OPA 比较结果也能控制 PWM 波形的输出，具体请参考 OPA 章节。

图 19 显示出 Timer2 周期模式和 PWM 模式的时序图：

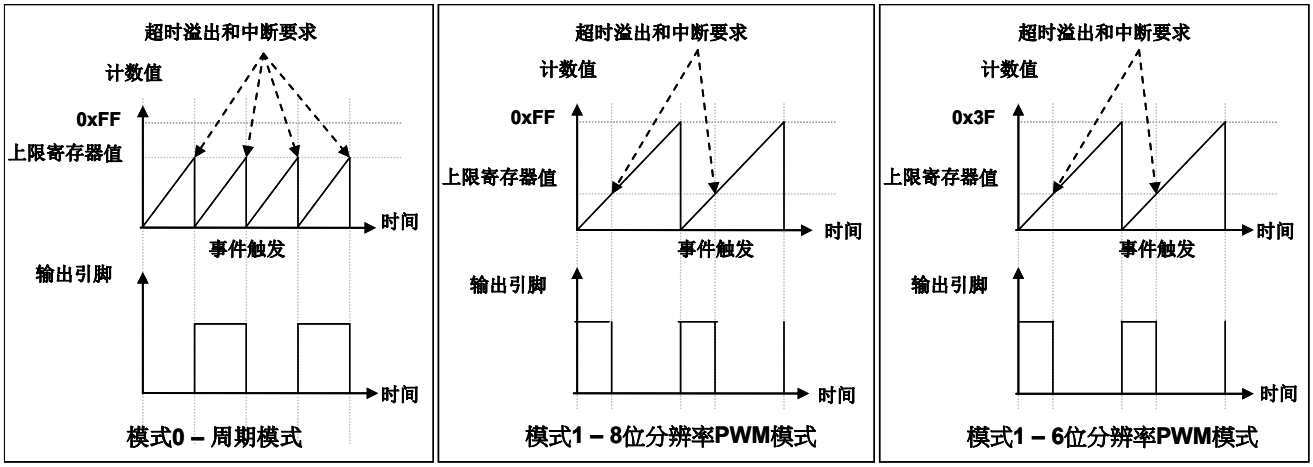


图 19：Timer2 周期模式和 PWM 模式的时序图

10.2.1. Timer2、Timer3 相关寄存器

10.2.1.1. Timer2/Timer3 上限寄存器(*TM2B/TM3B*)，地址 = 0x61/0x63

位	初始值	读/写	描 述
7 – 0	0x00	只写	Timer2/Timer3 上限寄存器。

10.2.1.2. Timer2/Timer3 计数寄存器(*TM2CT/TM3CT*)，地址 = 0x29/0x2B

位	初始值	读/写	描 述
7 – 0	0x00	读/写	Timer2/Timer3 定时器位[7:0]。

10.2.1.3. Timer2/Timer3 分频寄存器(*TM2S/TM3S*)，地址 = 0x60/0x62

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/Timer3 时钟预分频器。 00：÷ 1 01：÷ 4 10：÷ 16 11：÷ 64
4 – 0	00000	只写	Timer2/Timer3 时钟分频器。

10.2.1.4. Timer2/Timer3 控制寄存器(*TM2C/TM3C*)，地址 = 0x28/0x2A

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	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 0111：NILRC 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
3 – 2	00	读/写	Timer2 输出选择。 00：停用 01：PB2 或 PB0 （由 code option TM2C[3:2]=1 选定） 10：PA3 11：PB4	Timer3 输出选择。 00：停用 01：PB5 10：PB6 11：PB7
1	0	读/写	Timer2/Timer3 模式选择。 0 / 1：周期模式 / PWM 模式。	
0	0	读/写	启用 Timer2/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 (PWMG0/1/2)

PFC460 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, PC2
(2) PWMG1 – PA4, PB6, PB7, PC3
(3) PWMG2 – PA3, PB2, PB3, PC0

10.3.1. 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, $2^N \times T_{clock} = T_{Period}$）。此外，用户可以通过使用程序选项 OPA_PWM，使 OPA 比较结果也能控制 PWM 波形的输出，具体请参考 OPA 章节。

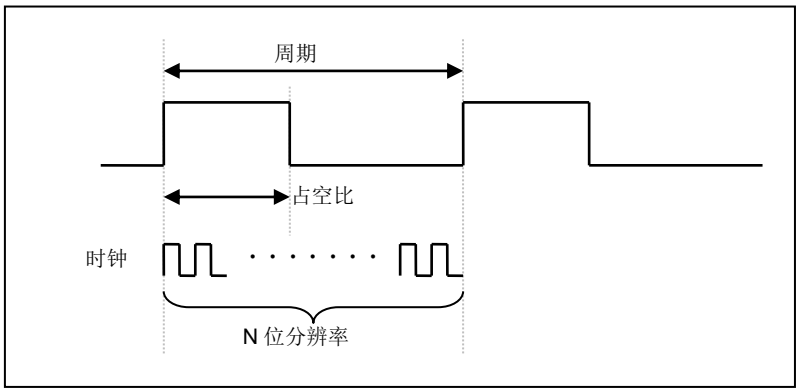


图 20：PWM 输出波形

10.3.2. 硬件和时钟框图

图 21 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。

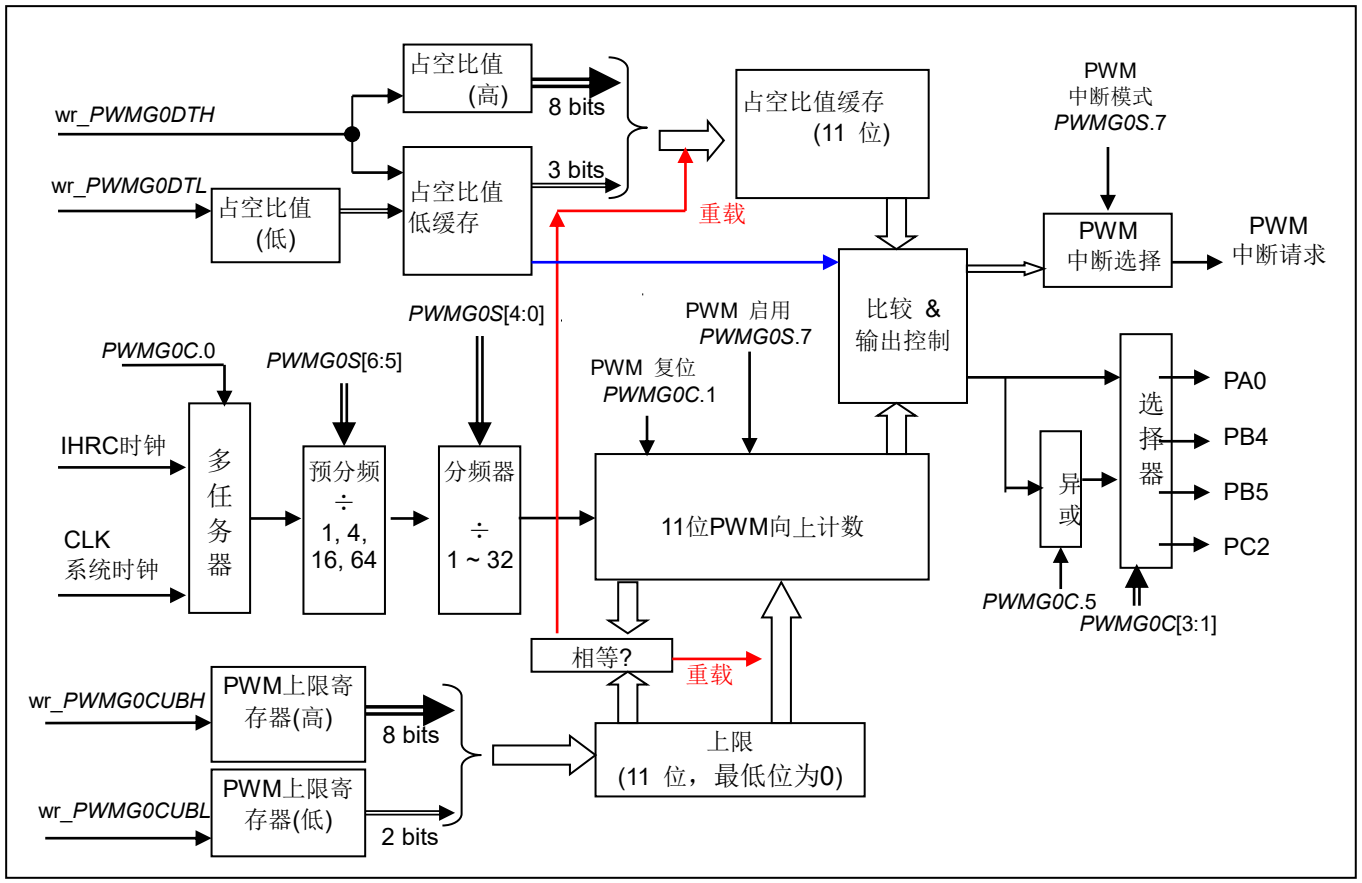


图 21：11 位 PWM 生成器（PWMG0）硬件框图

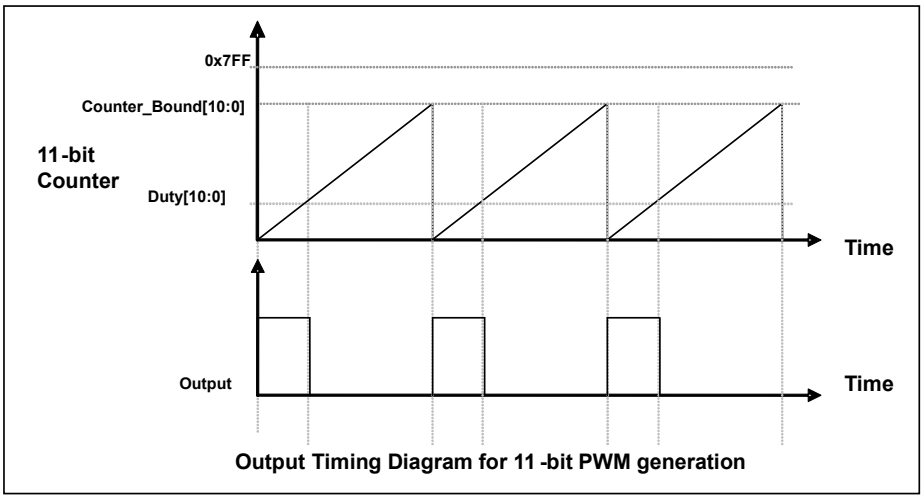


图 22：11 位 PWM 生成器(PWMG0)的输出时序图

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]},（x=0/1/2）占空比

DB0 = Duty_Bound[0] = *PWMGxDTL*[5] （x=0/1/2）

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= 0x22

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	000	读/写	选择 PWMG0 输出： 000：不输出 001：PB5 010：PC2 011：PA0 100：PB4 其他：保留
0	0	读/写	PWMG0 时钟源选择。 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= 0x23

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	00	只写	PWMG0 预分频。 00：÷1 01：÷4 10：÷16 11：÷64
4 – 0	00000	只写	PWMG0 分频。

10.3.4.3. PWMG0 占空比高位寄存器(*PWMG0DTH*)，地址 = 0x50

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 = 0x51

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= 0x52

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= 0x53

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= 0x24

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	000	读/写	选择 PWMG1 输出： 000：不输出 001：PB6 010：PC3 011：PA4 100：PB7 其他：保留
0	0	读/写	PWMG1 时钟源选择。 0：CLK 1：IHRC 或者 IHRC*2（由程序选项 PWM_source 控制）

10.3.4.8. PWMG2 控制寄存器(*PWMG2C*)，地址= 0x26

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 010：PC0 011：PA3 100：PB2 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）

10.3.4.9. PWMG1/*PWMG2* 分频寄存器(*PWMG1S/PWMG2S*)，地址= 0x25/0x27

位	初始值	读/写	描 述
7	0	只写	PWMG1/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	00	只写	PWMG1/PWMG2 预分频： 00：÷1 01：÷4 10：÷16 11：÷64
4 – 0	00000	只写	PWMG1/PWMG2 分频。

10.3.4.10. PWMG1/*PWMG2* 占空比高位寄存器(*PWMG1DTH/PWMG2DTH*)，地址 = 0x54/0x58

位	初始值	读/写	描 述
7 – 0	0x00	只写	PWMG1/PWMG2 占空比值位[10:3] 。

10.3.4.11. PWMG1/*PWMG2* 占空比低位寄存器(*PWMG1DTL/PWMG2DTL*)，地址 = 0x55/0x59

位	初始值	读/写	描 述
7 – 5	000	只写	PWMG1/PWMG2 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*。对于 PWMG2 也应同理设置。

10.3.4.12. PWMG1/*PWMG2* 计数上限高位寄存器(*PWMG1CUBH/PWMG2CUBH*)，地址= 0x56/0x5A

位	初始值	读/写	描 述
7 - 0	0x00	只写	PWMG1/PWMG2 上限寄存器位[10:3] 。

10.3.4.13. PWMG1/*PWMG2* 计数上限低位寄存器(*PWMG1CUBL/PWMG2CUBL*)，地址= 0x57/0x5B

位	初始值	读/写	描 述
7 – 6	00	只写	PWMG1/PWMG2 上限寄存器位[2:1] 。
5	0	只写	PWMG1/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 23 所示。

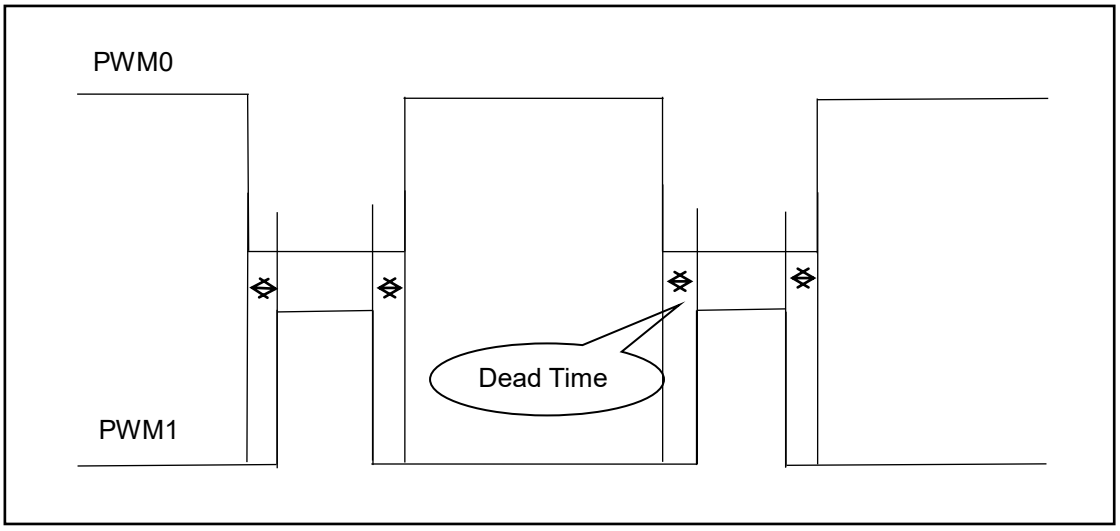


图 23：两路互补 PWM 波形

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 12 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 12：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
...	...

表 13：死区参数规则

10.4. 11 位 SuLED LPWM 计数器 (LPWMG0/1/2)

PFC460 内置一组三路 11 位 SuLED (Super LED)硬件 LPWM 生成器(LPWMG0、LPWMG1 和 LPWMG2)。各路输出端口如下：

- LPWMG0 – PA0, PB4, PB5, PB6, PC2
- LPWMG1 – PA4, PB6, PB7, PC3
- LPWMG2 – PA3, PA5, PB2, PB3, PB5, PC0

其中，LPWMG0 与 LPWMG2 共享输出端口 PB5，但不可二者同时输出 LPWM 到 PB5；
同理，LPWMG0 与 LPWMG1 共享输出端口 PB6，但不可二者同时输出 LPWM 到 PB6。

10.4.1. LPWM 波形

LPWM 输出波形（图 24）有一个时基（T_{Period} = 时间周期）和一个周期里输出高电平的时间（占空比）。LPWM 输出的频率取决于时基($f_{LPWM} = 1/T_{Period}$)。

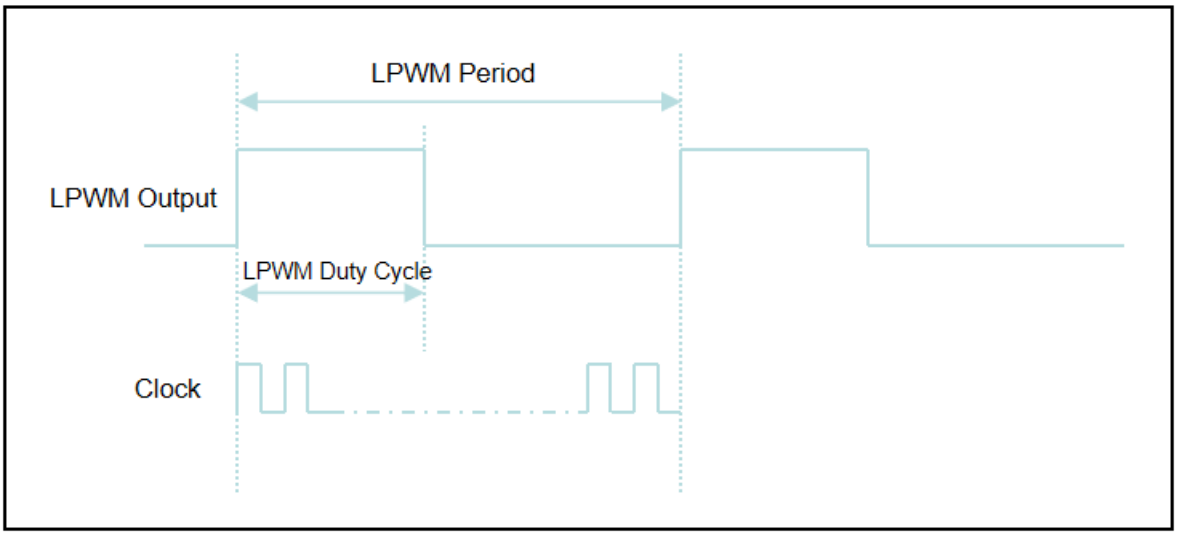


图 24：LPWM 输出波形

10.4.2. 硬件框图

图 25 所示是整组 SuLED 11 位 LPWM 生成器的硬件方框图。这三组 LPWM 生成器使用共同的 Up-Counter 和时钟源选择开关来产生时基，所以 LPWM 周期的起点（上升沿）是同步的，时钟源可以是 IHRC 或者系统时钟。LPWM 信号输出引脚通过 *LPWMGxC* 寄存器来选择。LPWM 波形的周期由 LPWM 上限高和低寄存器决定，各路 LPWM 波形的占空比由各路 LPWM 占空比高和低寄存器决定。

在 LPWMG0 通道的那两个附带的 OR 和 XOR 逻辑门是用于产生互补非重叠并有死区的开关控制波形的。

此外，用户也可以通过使用程序选项 OPA_PWM，使 OPA 比较结果也能控制 LPWM 波形的输出，具体请参考 OPA 章节。

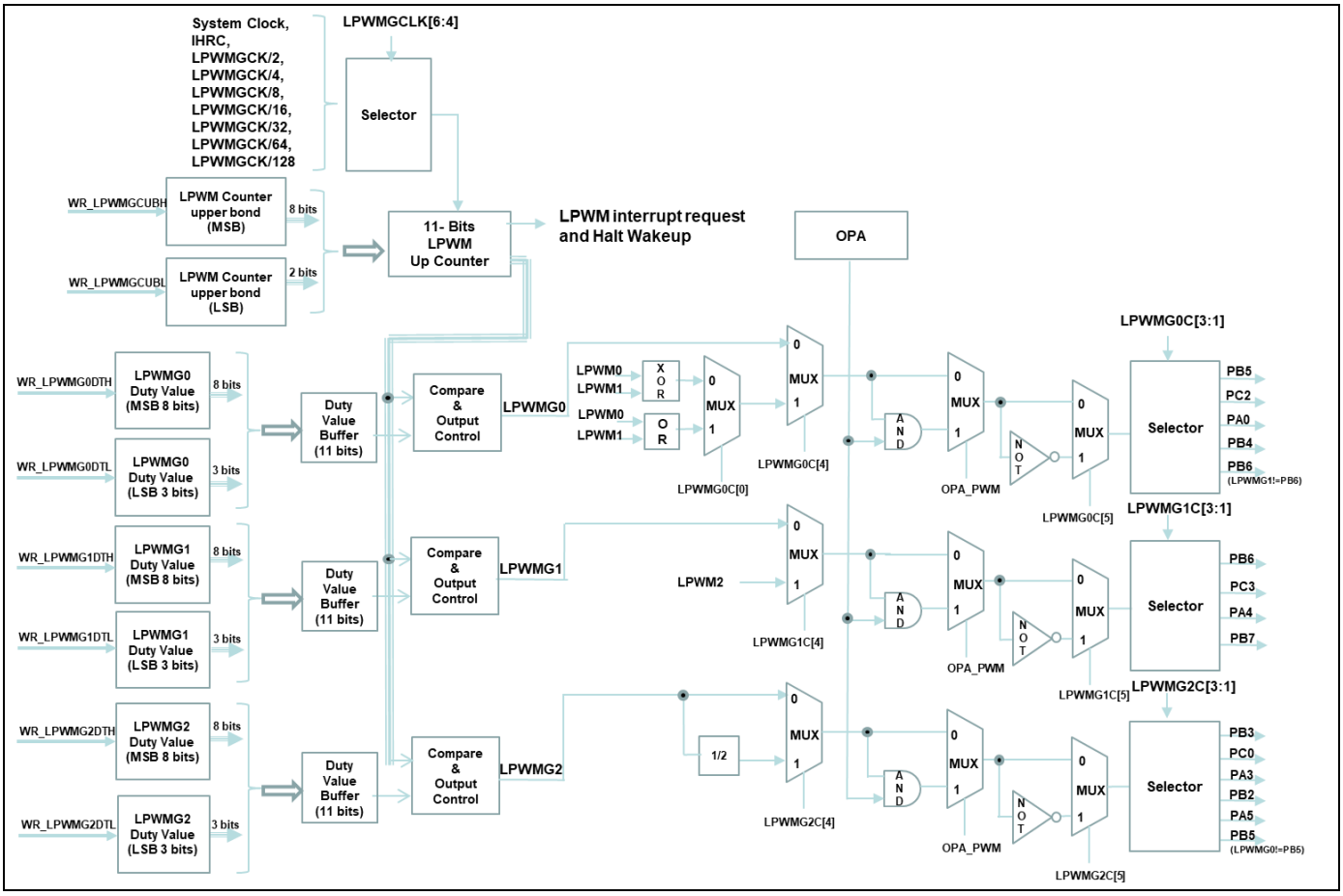


图 25：整组 SuLED 三路 11 位 LPWM 生成器硬件框图

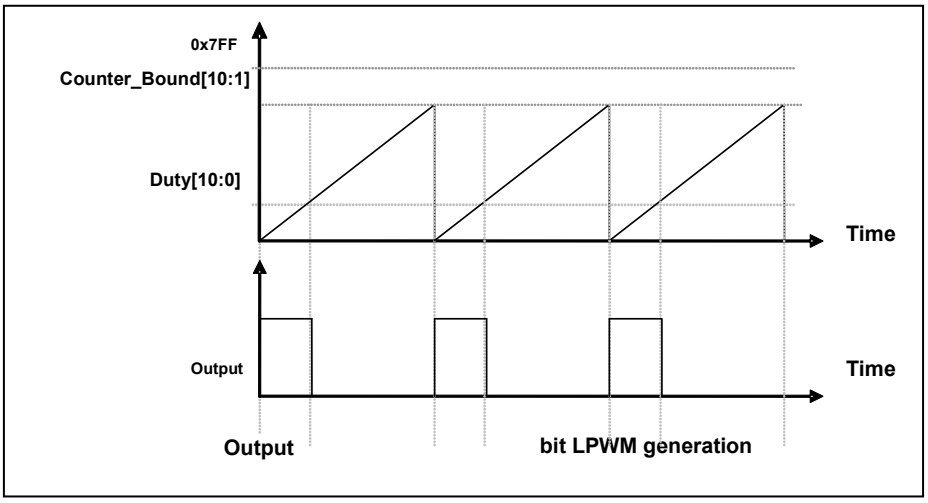


图 26：11 位 LPWM 生成器输出时序图

10.4.3. 11 位 LPWM 生成器计算公式

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

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

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

这里,

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

DB10_1 = Duty_Bound[10:1] = {*LPWMGxDTH* [7:0],*LPWMGxDTL*[7:6]},（x=0/1/2） 占空比

DB0 = Duty_Bound[0] = *LPWMGxDTL*[5] （x=0/1/2）

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

10.4.4. 11bit LPWM 计数器相关寄存器

10.4.4.1. LPWMG0 控制寄存器(*LPWMG0C*)，地址= 0x0C

位	初始值	读/写	描述
7	-	-	保留。
6	-	只读	LPWMG0 生成器输出状态。
5	0	只写	选择 LPWMG0 的输出的结果是否反极性： 0/1：停用/启用
4	0	只写	LPWMG0 输出选择。 0：LPWMG0 输出 1：LPWMG0 XOR LPWMG1 或者 LPWMG0 OR LPWMG1（通过 LPWMG0C.0 位来选择）
3 – 1	000	读/写	LPWMG0 输出端口选择。 000：输出停用 001：PB5 010：PC2 011：PA0 100：PB4 101：PB6（仅适用于 PB6 未用作 LPWMG1 输出时） 其他：保留
0	0	读/写	LPWMG0 输出预选择。 0：LPWMG0 XOR LPWMG1 1：LPWMG0 OR LPWMG1

10.4.4.2. LPWMG1 控制寄存器 (*LPWMG1C*)，地址= 0x0D

位	初始值	读/写	描述
7	-	-	保留。
6	-	只读	LPWMG1 生成器输出状态。
5	0	读/写	选择 LPWMG1 的输出的结果是否反极性： 0/1：停用/启用。
4	0	读/写	LPWMG1 输出选择： 0：LPWMG1 1：LPWMG2
3 – 1	000	读/写	LPWMG1 输出端口选择： 000：输出停用 001：PB6 010：PC3 011：PA4 100：PB7 1xx：保留
0	-	读/写	保留。

10.4.4.3. LPWMG2 控制寄存器(*LPWMG2C*)，地址 = 0x0E

位	初始值	读/写	描述
7	-	-	保留。
6	-	只读	LPWMG2 生成器输出状态。
5	0	读/写	选择 LPWMG2 的输出的结果是否反极性： 0/1：停用/启用
4	0	读/写	LPWMG2 输出选择: 0：LPWMG2 1：LPWMG2 ÷2
3 – 1	000	读/写	LPWMG2 输出端口选择： 000：输出停用 001：PB3 010：PC0 011：PA3 100：PB2 101：PA5 110：PB5（仅适用于 PB5 未用作 LPWMG0 输出时） 111：保留
0	-	读/写	保留。

10.4.4.4. LPWMG 时钟寄存器(*LPWMGCLK*)，地址 = 0x67

位	初始值	读/写	描述
7	0	只写	LPWMG 停用/ 启用。 0：LPWMG 停用 1：LPWMG 启用
6 – 4	000	只写	LPWMG 时钟预分频。 000：÷1 001：÷2 010：÷4 011：÷8 100：÷16 101：÷32 110：÷64 111：÷128
3 – 1	-	-	保留。
0	0	只写	LPWMG 时钟源选择。 0：系统时钟 1：IHRC 或者 IHRC*2（由 code option LPWM_Source 决定）

10.4.4.5. LPWMG 计数上限高位寄存器(*LPWMGCUBH*)，地址 = 0x68

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

10.4.4.6. LPWMG 计数上限低位寄存器(LPWMGCUBL)，地址= 0x69

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

10.4.4.7. LPWMG0/1/2 占空比高位寄存器(*LPWMGxDTH, x=0/1/2*)，地址 = 0x6A/0x6C/0x6E

位	初始值	读/写	描述
7 – 0	-	只写	LPWMG0/LPWMG1/LPWMG2 占空比值。位[10:3]。

10.4.4.8. LPWMG0/1/2 占空比低位寄存器(*LPWMGxDTL, x=0/1/2*)，地址 = 0x6B/0x6D/0x6F

位	初始值	读/写	描述
7 – 5	-	只写	LPWMG0/LPWMG1/LPWMG2 占空比值。位[2:0]。
4 – 0	-	-	保留。

注意：必须先写入 LPWMGx 占空比低位寄存器，再写入 LPWMGx 占空比高位寄存器。（x=0/1/2）

10.4.5. 带互补死区的 LPWM 波形范例

基于 PFC460 独特的 11 bit SuLED LPWM 结构，在此采用 LPWM2 输出、LPWM0 与 LPWM1 异或后通过 LPWM0 反极性输出，来获得两路互补带死区 LPWM 波形。示例如下：

```
#define   dead_zone          10          //   死区时间 = 10% * (1/LPWM_Frequency) us
#define   LPWM_Pulse    50          //   该互补死区 LPWM 占空比为 50%

#define   LPWM_Pulse_1   35          //   该互补死区 LPWM 占空比为 35%
#define   LPWM_Pulse_2   60          //   该互补死区 LPWM 占空比为 60%
#define   switch_time    400*2     //   切换占空比时，用于调整切换时间
//注：为防止杂波产生，switch_time 应为 LPWM 周期的倍数。此例 LPWM 周期：1/2.5KHz = 400 us，故切
//换时间为 400*2 us

void    FPPA0 (void)
{
.ADJUST_IC SYSCLK=IHRC/16, IHRC=16MHz, VDD=5V;
//******* 产生固定占空比 ************************
//------- 设置计数上限及占空比 --------------------
LPWMG0DTL    =    0x00;
LPWMG0DTH    =    LPWM_Pulse + dead_zone;
LPWMG1DTL    =    0x00;
LPWMG1DTH    =    dead_zone;          // LPWMG0 与 LPWMG1 异或后，LPWM 占空比
                                      // 为 LPWM_Pulse%
LPWMG2DTL     =   0x00;
```

```
LPWMG2DTH     =     LPWM_Pulse + dead_zone*2;

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

while(1)
{
   //******** 切换占空比 *********************************
      // 切换占空比时，为避免可能出现的瞬间死区消失，应遵循如下顺序。
      // 占空比由大变小：50%/60%  →  35%
   LPWMG0DTL   =     0x00;
   LPWMG0DTH   =     LPWM_Pulse_1 + dead_zone;
   LPWMG2DTL   =     0x00;
   LPWMG2DTH   =     LPWM_Pulse_1 + dead_zone*2;
   .delay      switch_time

   //占空比由小变大：35% → 60%
   LPWMG2DTL   =     0x00;
   LPWMG2DTH   =     LPWM_Pulse_2 + dead_zone*2;
   LPWMG0DTL   =     0x00;
   LPWMG0DTH   =     LPWM_Pulse_2 + dead_zone;
   .delay      switch_time
    }
   }
```

上述程序中，固定占空比时对应的 LPWM0/LPWM2 波形如图 27 所示。

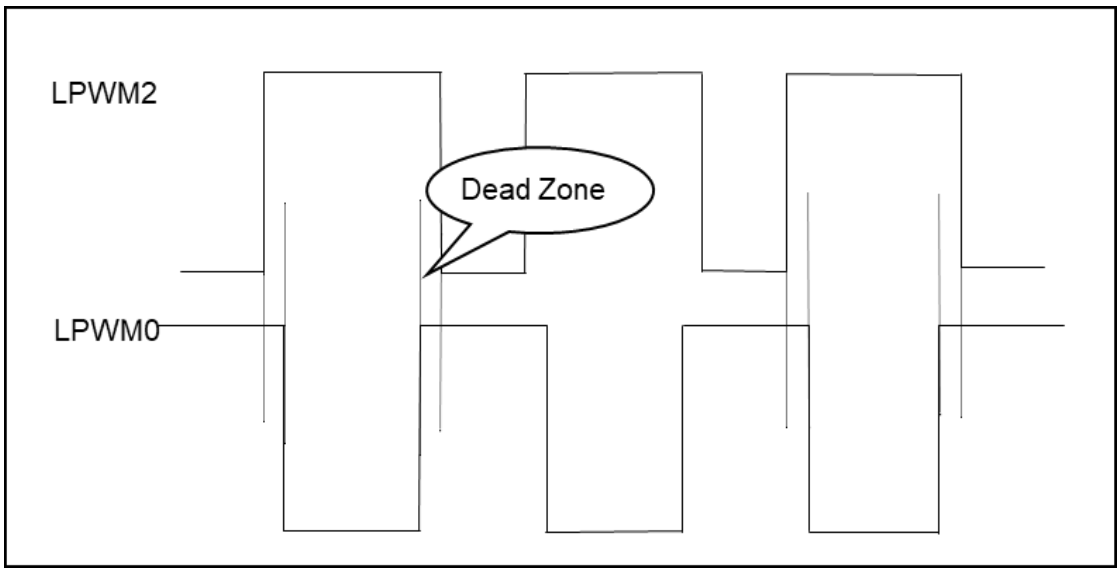


图 27：两路互补 LPWM 波形

切换占空比时对应的 LPWM0/LPWM2 波形如图 28。

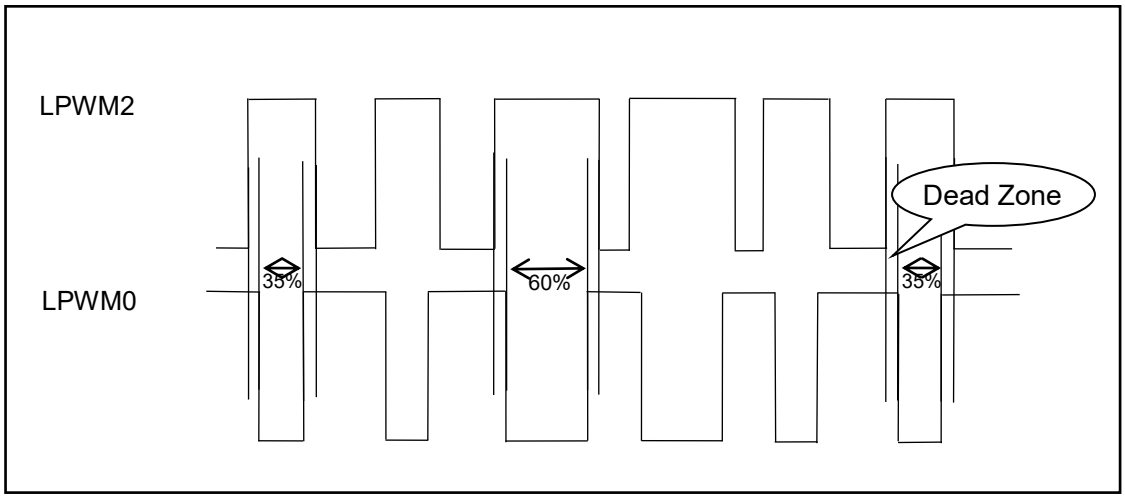


图 28：两路互补 LPWM 波形

可以发现，上述例程所得波形，其死区是两组 LPWM 同时为高。若用户需要 LPWM 同时为低的死区，仅需改变各自输出控制的 Inverse 即可。如：

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

11. 特殊功能

11.1. 比较器

PFC460 内置一个硬件比较器，硬件框图如图 29。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；

(6) 控制各类 PWM 输出，详细用法请参考 *GPC2PWM* 寄存器。

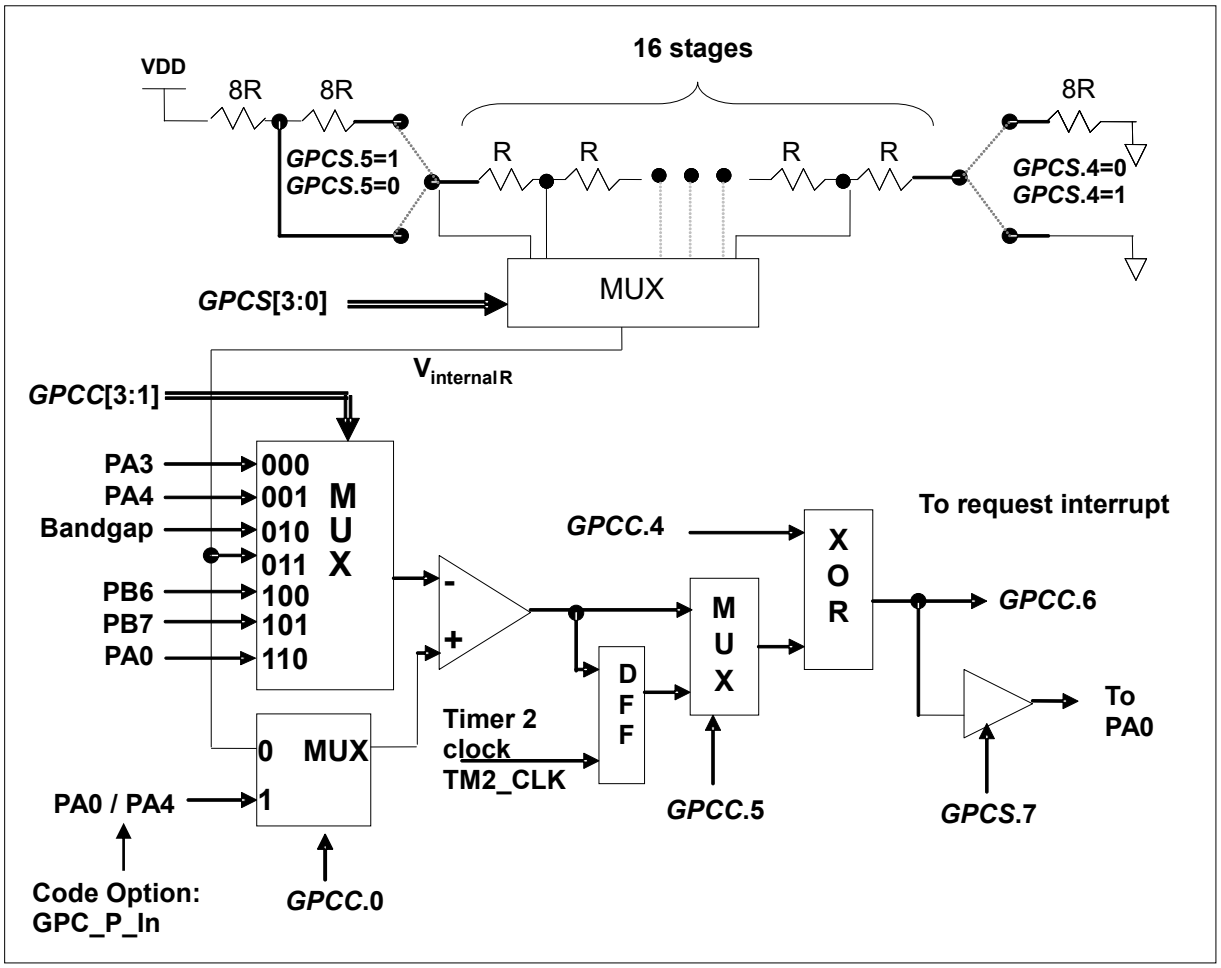


图 29：比较器硬件图框

11.1.1. 比较器控制寄存器(*GPCC*)，地址= 0x35

位	初始值	读/写	描 述
7	0	读/写	启用比较器。 0 / 1 ：停用/启用 当此位被设置为启用，请同时设置相应的模拟输入引脚是数字停用，以防止漏电。
6	-	只读	比较器结果。 0：正输入 < 负输入 1：正输入 > 负输入
5	0	读/写	选择比较器的结果是否由 TM2_CLK 采样输出。 0：比较器的结果没有 TM2_CLK 采样输出 1：比较器的结果是由 TM2_CLK 采样输出
4	0	读/写	选择比较器输出的结果是否反极性。 0：比较器输出的结果没有反极性 1：比较器输出的结果是反极性
3 – 1	000	读/写	选择比较器负输入的来源。 000：PA3 001：PA4 010：内部 1.20 V Bandgap 参考电压 011：$V_{internal\ R}$ 100：PB6 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 code option: GPC_P_In 决定）

11.1.2. 比较器选择寄存器(*GPCS*)，地址 = 0x36

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. 比较结果触发 PWM 控制寄存器(*GPC2PWM*)，地址 = 0x43

位	初始值	读/写	描 述
7	0	只写	比较器结果控制 TM2 PWM 输出。0/1：停用/启用
6	0	只写	比较器结果控制 TM3 PWM 输出。0/1：停用/启用
5	0	只写	比较器结果控制 PWMG2 PWM 输出。0/1：停用/启用
4	0	只写	比较器结果控制 PWMG1 PWM 输出。0/1：停用/启用
3	0	只写	比较器结果控制 PWMG0 PWM 输出。0/1：停用/启用
2	0	只写	比较器结果控制 LPWMG2 PWM 输出。0/1：停用/启用
1	0	只写	比较器结果控制 LPWMG1 PWM 输出。0/1：停用/启用
0	0	只写	比较器结果控制 LPWMG0 PWM 输出。0/1：停用/启用

比较器控制 PWM 输出的原理可参考下图，当比较器输出结果为 1 时，PWM 停止输出；比较结果为 0 时，PWM 恢复输出。如图 30 所示。

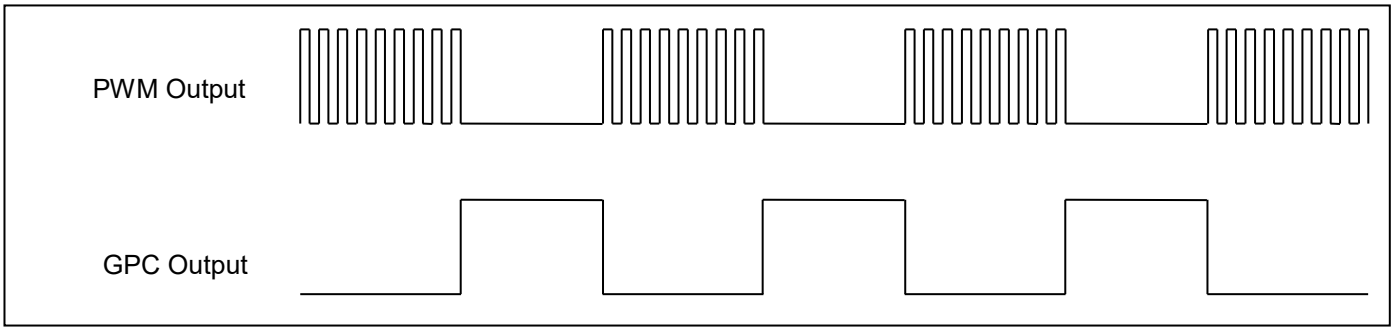


图 30：比较器结果控制 PWM 输出

11.1.4. 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。

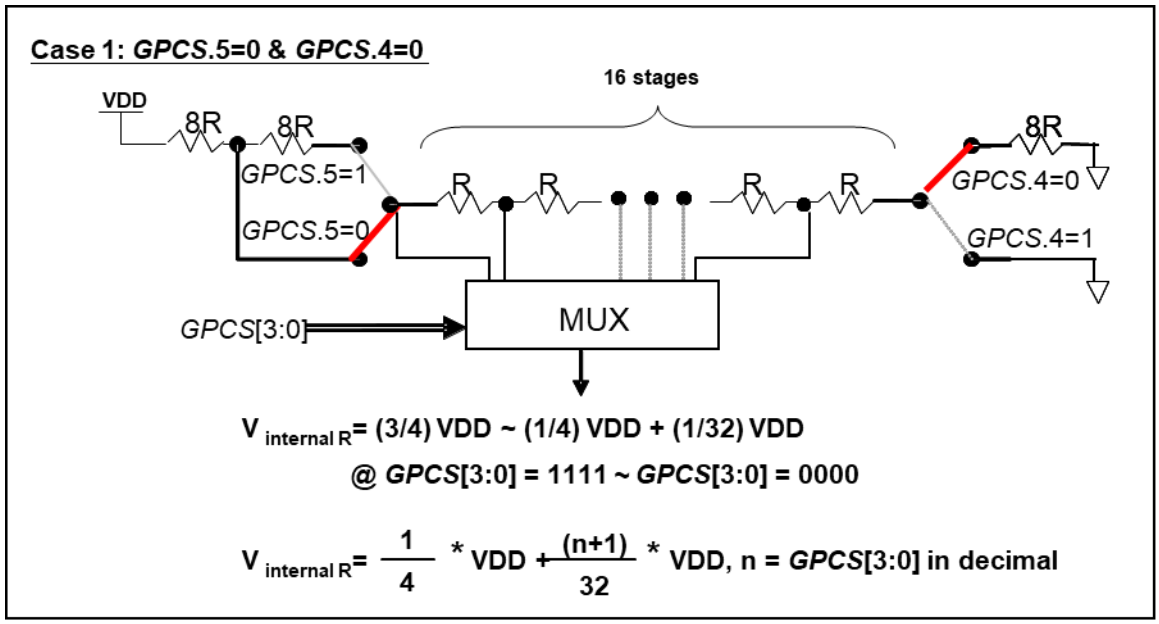


图 31：$V_{internal\ R}$ 硬件接法 (*GPCS*.5=0 & *GPCS*.4=0)

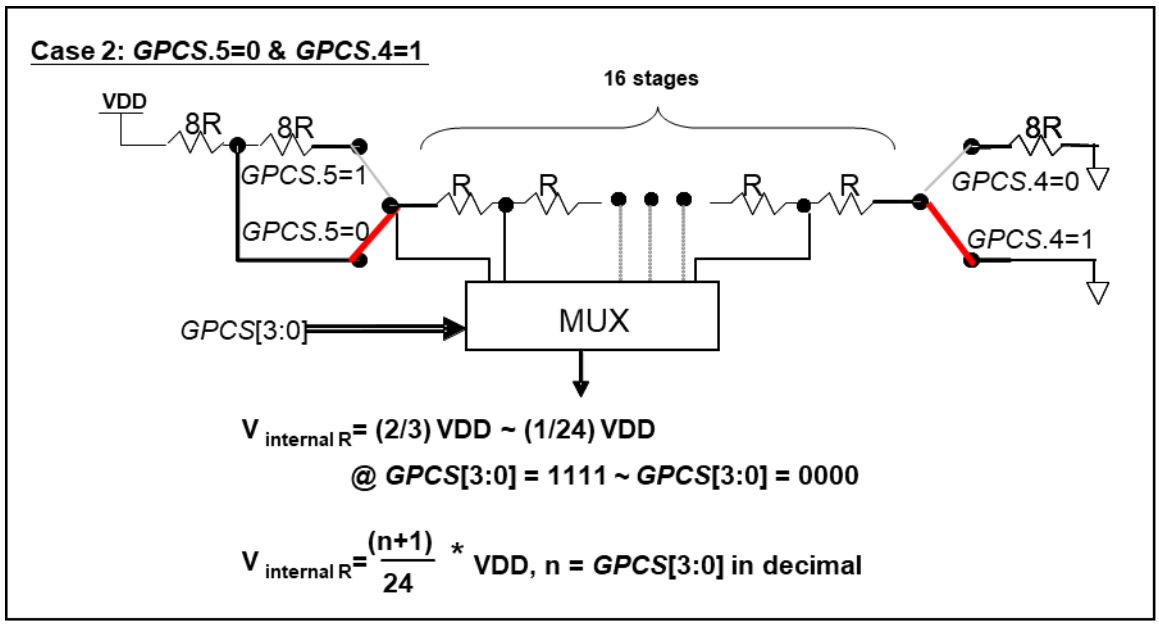


图 32： $V_{internal\ R}$ 硬件接法 (*GPCS*.5=0 & *GPCS*.4=1)

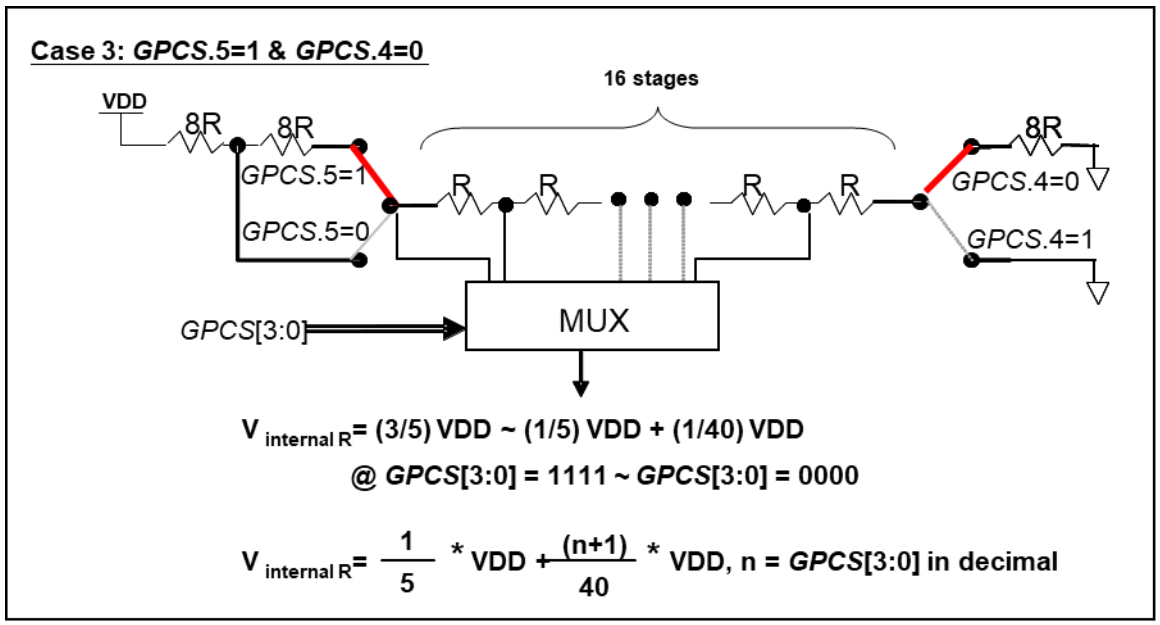


图 33：$V_{internal\ R}$ 硬件接法 (*GPCS*.5=1 & *GPCS*.4=0)

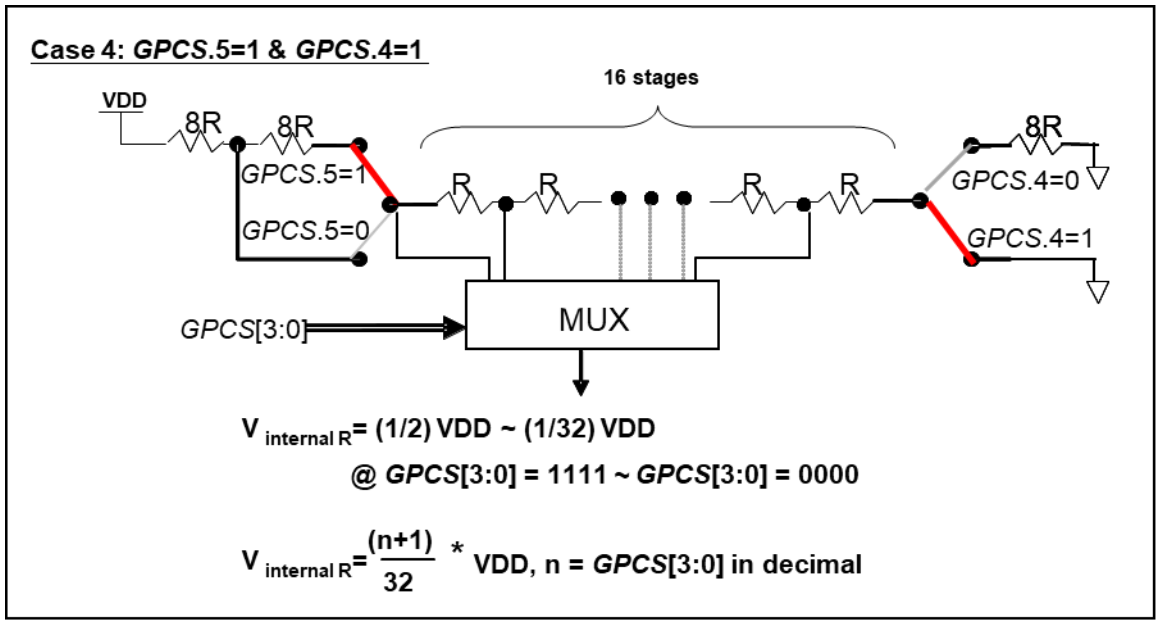


图 34：$V_{internal\ R}$ 硬件接法 (*GPCS*.5=1 & *GPCS*.4=1)

11.1.5. 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。

11.1.6. 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

PFC460 提供一个 VDD/2 偏置电压产生器，可作为驱动液晶显示器的 COM 功能，并且具有两组各五支可选的 VDD/2 输出引脚：LCD_B01256 表示 VDD/2 偏置电压分别经 PB0/PB1/PB2/PB5/PB6 这五支引脚输出，同理，LCD_A034_B01 则表示为 PA0/PA3/PA4/PB0/PB1 这五支引脚。该功能可以通过寄存器 *MISC*.4 来选择启用或关闭，通过 *MISC*.3 选择输出脚位。

杂项寄存器(*MISC*)，地址 = 0x49			
位	初始值	读/写	描 述
7 – 5	-	-	保留，请保持为 0。
6	-	只写	保留，请手动设置为 1。
5	0	只写	快唤醒功能。
4	**0**	**只写**	**使能 LCD 显示 VDD/2 功能。 0 / 1：停用 / 启用**
3	**0**	**只写**	**选择 VDD/2 输出脚位。 0: LCD_B01256，包含 PB0/PB1/PB2/PB5/PB6 1: LCD_A034_B01,包含 PA0/PA3/PA4/PB0/PB1**
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/下拉电阻 *PxPH*.x / *PxPL*.x 和数字输入 *PADIER*.x / *PBDIER*.x 防止输出电压受到干扰。图 35 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。

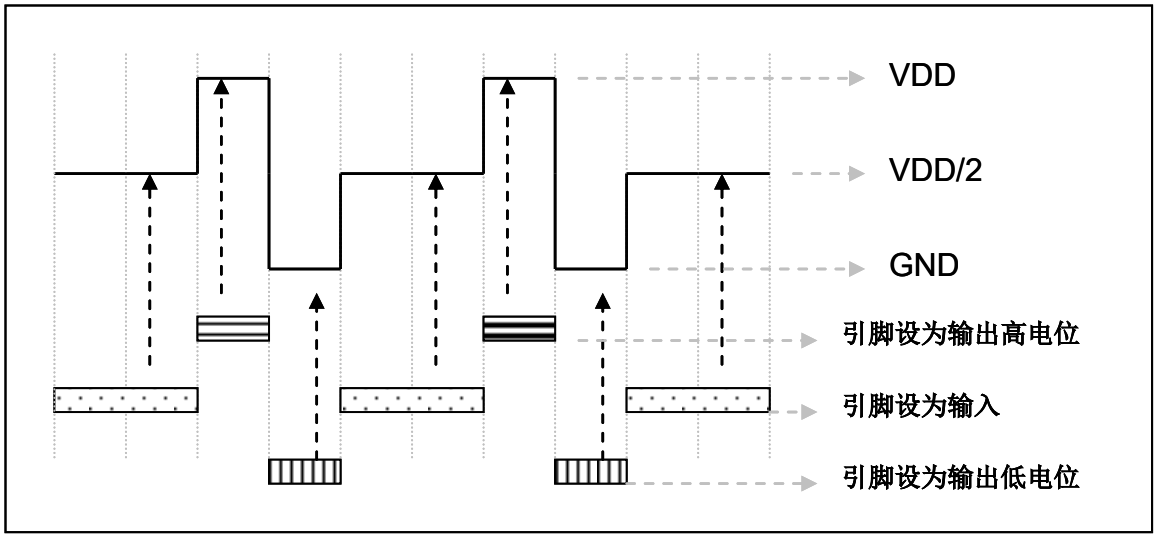


图 35：使用 VDD/2 偏置电压产生器

11.3. 运算放大器(OPA)模块

PFC460 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。

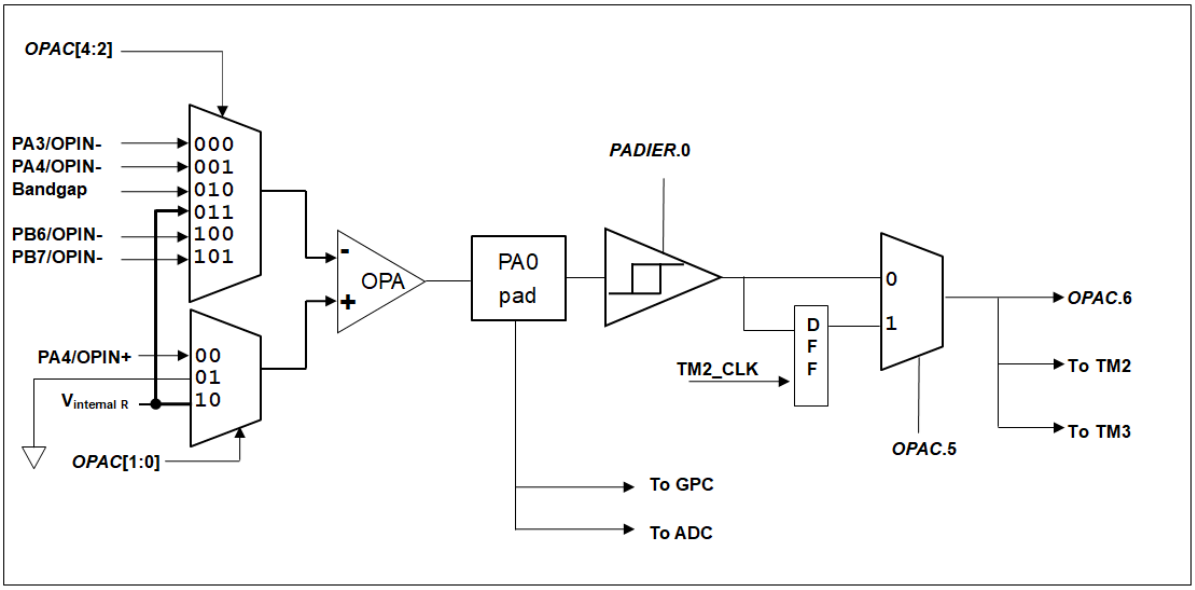


图 36：OPA 硬件图框

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。

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。

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。

程序选项“OPA_PWM”是指由 OPA 比较器模式的输出结果控制 PWM 波形。选用此功能后，当 OPA 输出结果为 1 时，PWM 停止输出；输出为 0 时，PWM 恢复输出。如图 37 所示。此功能对 8bit/11bit PWM 均有效。

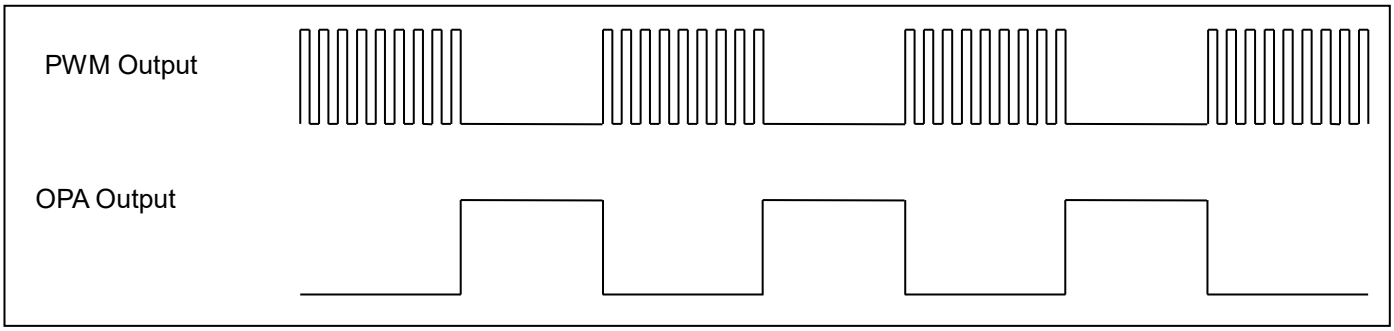


图 37：OPA 比较器模式控制 PWM 输出

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 = 0x33

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 = 0x34

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

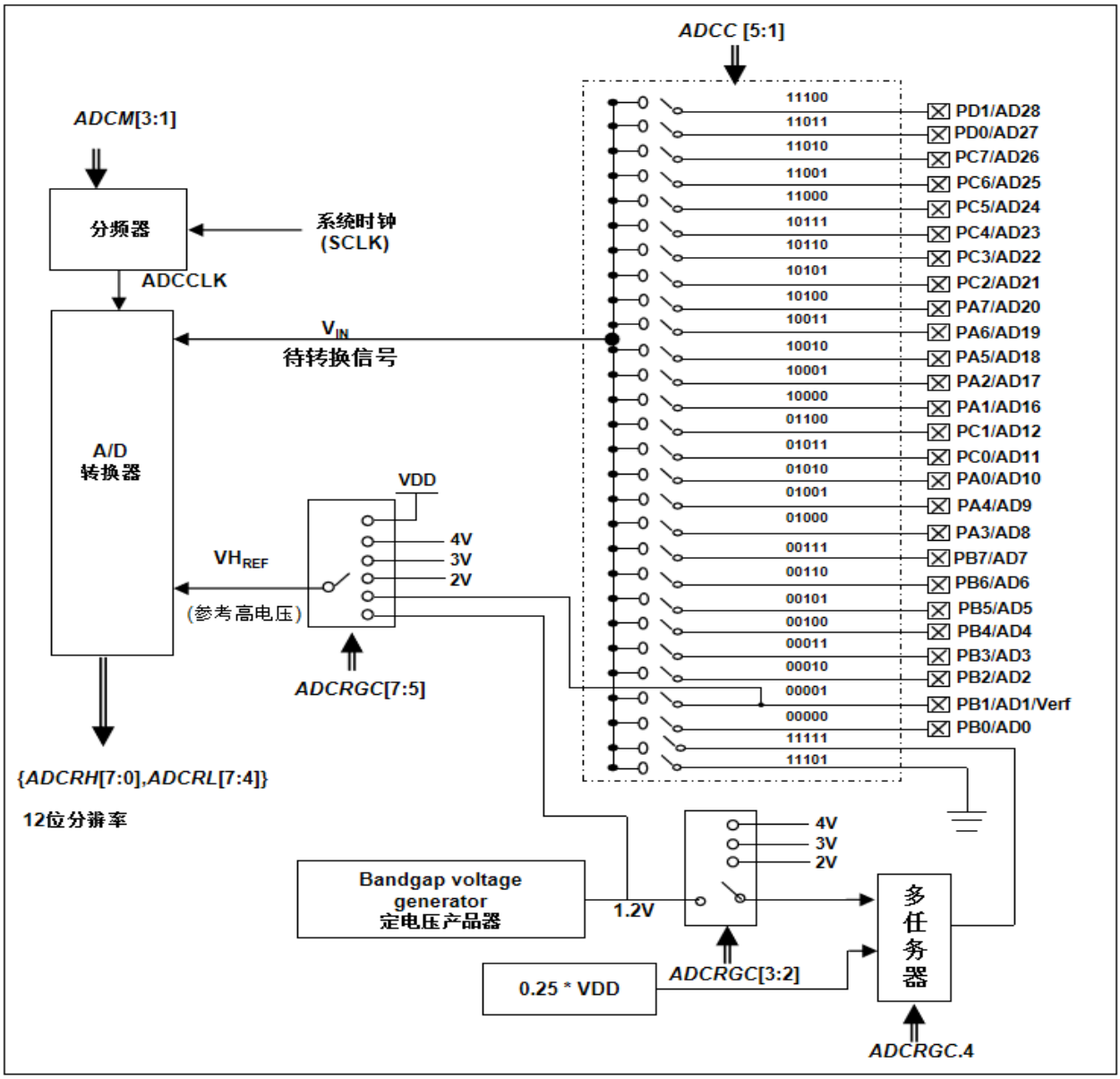


图 38：ADC 模块框图

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 *PxDIER* 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，那么只需要延迟 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 39 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。

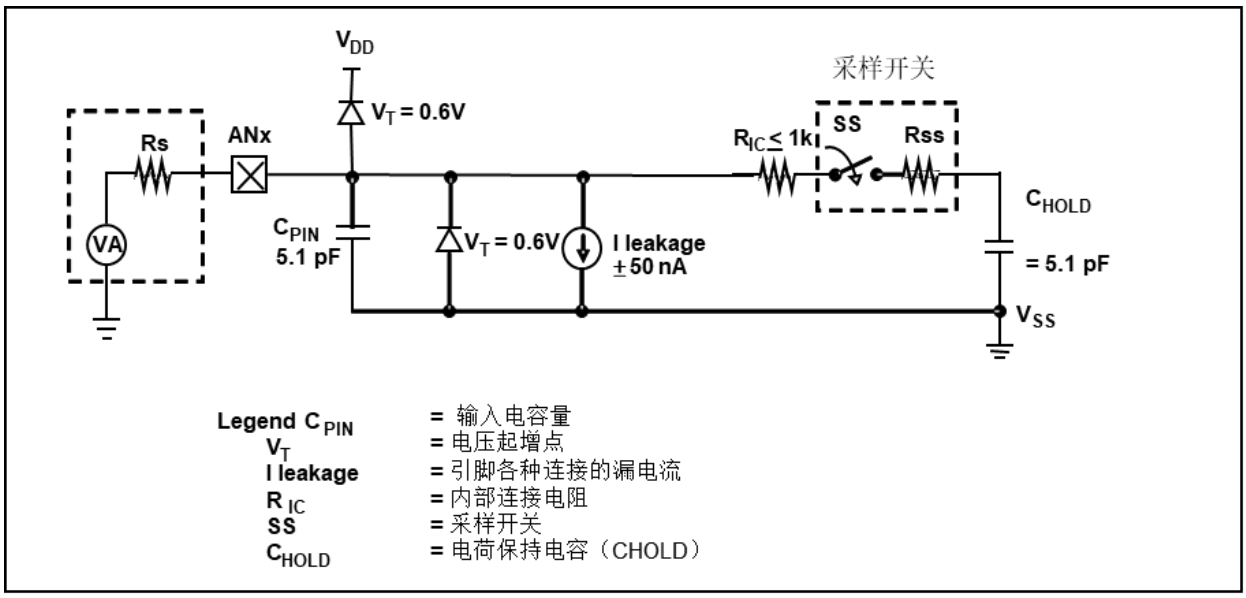


图 39：模拟输入模型

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 28 模拟信号可以被 AD 转换选择：26 个来自外部引脚的模拟输入信号，一个 bandgap 参考电压或者 0.25*V_{DD}，还有一个是以 GND 作为 ADC 输入通道。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，为了避免漏电，这些引脚在使用时应停用数字输入功能（设置 *PxDIER* 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/C/D 寄存器（*PxDIER*）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
$   ADCRGC  VDD ADC_BG  BG_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 = 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	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 – 1	00000	读/写	通道选择。 以下 5 位用来选择 AD 转换的输入信号：	
			00000：PB0/AD0, 00001：PB1/AD1, 00010：PB2/AD2, 00011：PB3/AD3, 00100：PB4/AD4, 00101：PB5/AD5, 00110：PB6/AD6, 00111：PB7/AD7, 01000：PA3/AD8, 01001：PA4/AD9, 01010：PA0/AD10, 01011：PC0/AD11 01100：PC1/AD12 01101：保留 01110：保留 01111：保留	10000：PA1/AD16 10001：PA2/AD17 10010：PA5/AD18 10011：PA6/AD19 10100：PA7/AD20 10101：PC2/AD21 10110：PC3/AD22 10111：PC4/AD23 11000：PC5/AD24 11001：PC6/AD25 11010：PC7/AD26 11011：PD0/AD27 11100：PD1/AD28 11101：GND 11110：保留 11111：（通道 F） Bandgap 参考电压或者 0.25*V_{DD}
0	-	-	保留（写 0）。	

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

位	初始值	读/写	描述
7 – 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 = 0x42

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 = 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。

11.4.6.5. 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

11.5. 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* 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* 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 40 所示。

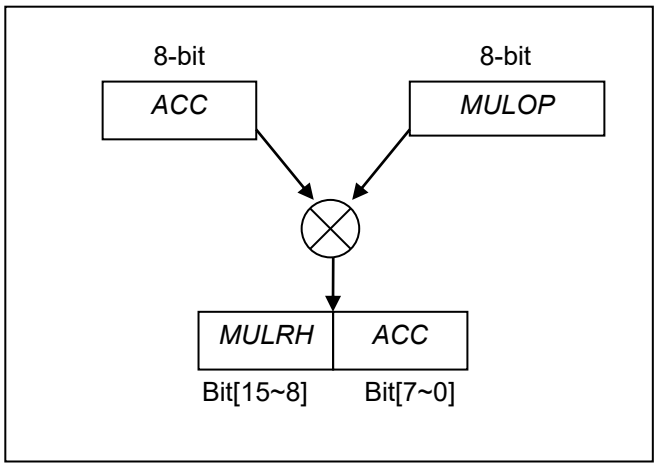


图 40：硬件乘法器框图

11.5.1. 乘法器运算对象寄存器 (*MULOP*), 地址 = 0x2C

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 = 0x2D

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

11.6. 触摸功能

PFC460 内含一个触摸检测电路，图 41 为其功能方框图：

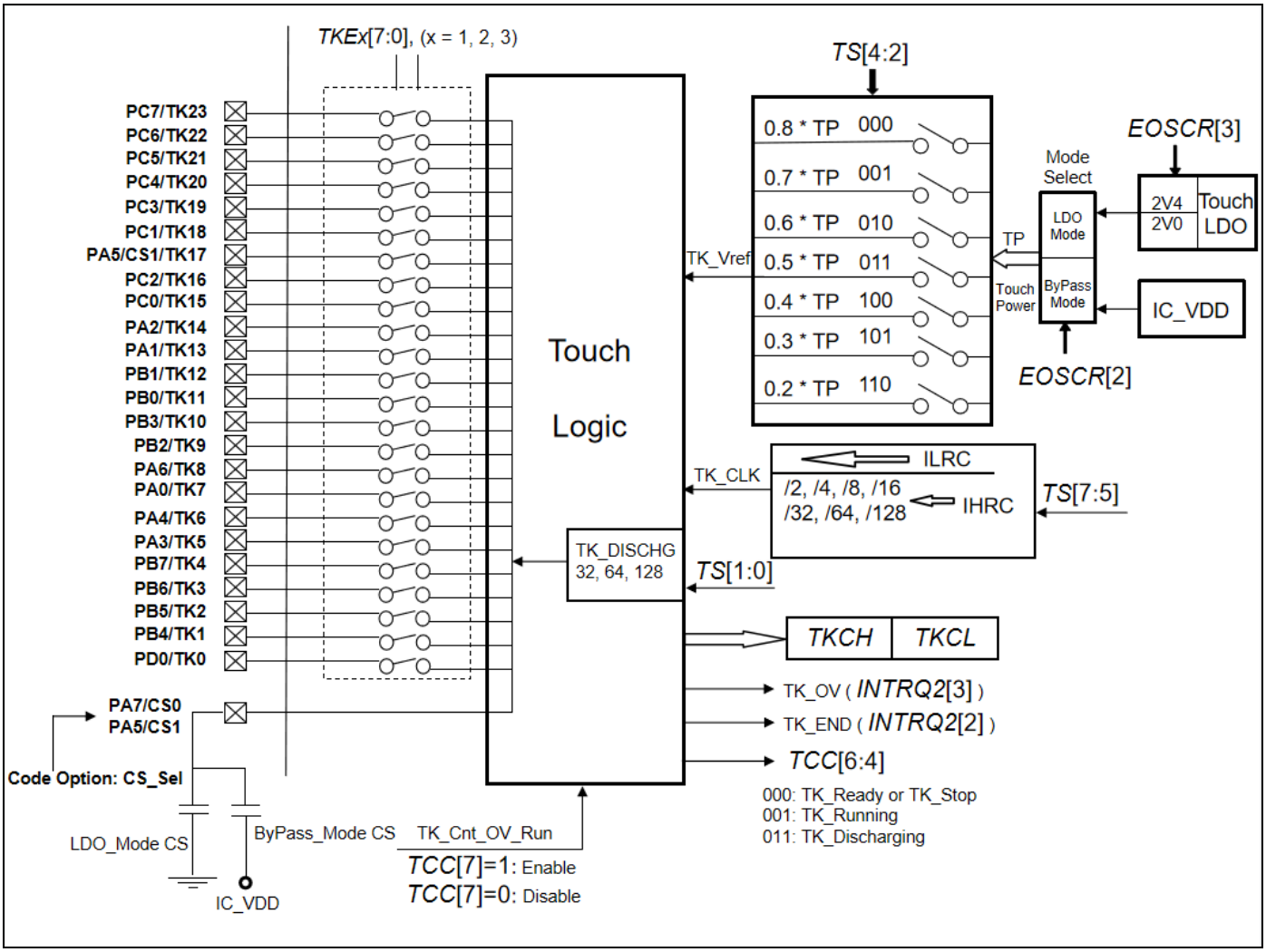


图 41：触摸检测电路的功能方框图

PFC460 中的触摸检测电路应用电容式感应的方法，检测手指的虚拟地面效应电容，或感应极片之间的电容。

使用触摸功能时，用户可通过寄存器 *ESOCR*[3:2]配置触摸模块电源。

1. 设置 *ESOCR*[2] 选择 ByPass/LDO 模式。
2. 若选择 ByPass 模式，触摸模块电源是芯片 VDD，需要在 CS（电容触摸灵敏度）引脚和 VDD 之间连接一颗精确而低漏电率的外部电容器 CS。
3. 若选择 LDO 模式时，通过 ESOCR[3]选择 2.4V/2V 的 LDO 提供触摸模块电源，需要在 CS（电容触摸灵敏度）引脚和 GND 之间连接一颗精确而低漏电率的外部电容器 CS。
4. 同时，用户可通过程序选项 CS_Sel 选择将 PA5 或者 PA7 配置为 CS 引脚。但请注意，PA5 端口的 CS1/TK17 功能不可同时使用。

要开动触摸检测，使用者应跟从以下步骤：

1. 用户通过设置 *TKE1/TKE2/TKE3* 寄存器来选择要测量的感应极片（引脚）。每次应只选择一个感应极片。
2. 用户可通过将“0x10”写入 TCC 寄存器以发出 Touch START 命令。电容 CS 首先被完全放电到 VSS，放电时间可以透过 TS[1:0]从 32，64 和 128 个触摸电路时钟周期中选择。
3. 电容值越大，将电容器完全放电到 VSS 所需的放电时间就越长。然而有些情况下，128 个触控时钟仍不足以把 CS 电容完全放电，这时用户应通过将“0x30” 而不是“0x10”写入 *TCC* 寄存器来启动此手动放电过程。 在由用户控制的一定放电时间之后，用户可以发出 Touch START (0x10)命令来继续此触摸转换进程， 或者使用者也可以通过将“0x00”写入 *TCC* 寄存器中止转换进程。
4. 在放电之后，CS 会在每个触摸时钟周期（TK_CLK）朝着 VCC 充电。充电速度是由所选感应极片的电容值决定。
5. 当其电压达到内部产生的阈电压 VREF 时，充电进程将自动停止。程序可以透过读取 *INTRQ*[3]来判断充电过程是否停止。VREF 电压可透过 *TS[4:2]*，在 0.2*TP ~ 0.8*TP 之间选择。
6. 经过读取触摸计数器 *TKCH* 和 *TKCL* 的值，用户可监测感应极片上的电容量变化。 读取到的值与 CS 和 CP（电容触摸传感器引脚）的比例有关，而 CP 表示电容可以通过因用户手指的触摸而变化的 PCB，导线和触摸板的组合的总电容。一旦 CP 值被改变，将 CS 充电到 VREF 所需的时间缩短。用户可通过读取触摸计数器的差值来判断触摸盘是否有被按下或放开。
7. 用户可透过调整 CS 电容容值大小来改变触摸的灵敏度。使用一个过大容值的 CS 电容，触摸计数器值有可能会溢位，此时 INTRQ2.TK_OV 旗号将会被硬件自动设起，且触摸计数值将会继续从 0 再开始计数。

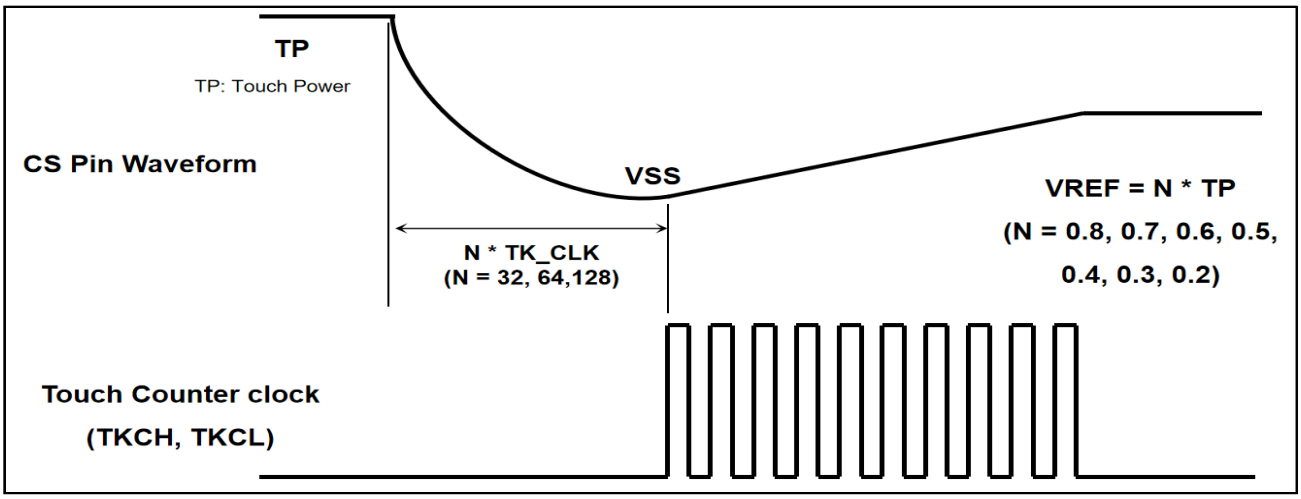


图 42：触摸转换的时序图

注意：

1. 当 VREF 电压首次设置或者中途切换参考电压选项时，请舍弃在此之后读取到的第一笔 TKCH 和 TKCL 的数据。
2. 触摸按键通道与 ADC 通道不可同时选用相同 IO 引脚，同时启用将使得触摸按键计数值降低。ADC 转换通道默认为 PB0/TK11，当使用 PB0/TK11 当触摸引脚时须将 ADC 默认通道设至其他引脚。
3. 相同条件下，不同引脚的触摸按键计数值将因个别 IO 引脚的电容效应(驱动电流、封装…等等因素)而有些差异。触摸按键信道 TK7/PA0 及 TK17/PA5/CS1 的触摸按键计数值会略小于其他引脚。
4. 在 ByPass 模式，执行 Touch START (将“0x10”写入 TCC 寄存器)命令必须以系统频率为 250KHz（IHRC/64）。而 LDO 模式则无此限制。

11.6.1. 触摸选项寄存器 (TS)，地址= 0x37

位	初始值	读/写	描述
7 – 5	000	读/写	触摸时钟选择 (TK_CLK) 000：ILRC 001：IHRC/2 010：IHRC/4 011：IHRC/8 100：IHRC/16 101：IHRC/32 110：IHRC/64 111：IHRC/128
4 – 2	011	读/写	触摸 VREF 选择（TP：Touch Power，默认为 LDO 2V） 000：0.8 * TP 001：0.7 * TP 010：0.6 * TP 011：0.5 * TP 100：0.4 * TP 101：0.3 * TP 110：0.2 * TP 111：保留
1 – 0	00	读/写	在开始触摸功能前选择放电时间 (TK_DISCHG) 00：保留 01：32 * CLK 10：64 * CLK 11：128 * CLK

注意：LDO 模式 TP 默认为 LDO 2V，ByPass 模式 TP 为 IC_VDD。

11.6.2. 外部晶体振荡器控制寄存器(EOSCR)，地址 = 0x0F

位	初始值	读/写	描 述
7	0	只写	使能外部晶体振荡器。0 / 1：停用/启用
6 – 5	00	只写	晶体振荡器的选择。 00：保留 01：低驱动电流。适用于较低频率晶体，例如：32KHz 10：中驱动电流。适用于中等频率晶体，例如：1MHz 11：高驱动电流。适用于较高频率晶体，例如：4MHz
4	-	-	保留。
3	**-**	**只写**	**LDO 输出电压选择。0/1：2.4V/2V**
2	**-**	**只写**	**触摸模块电源选择。0/1：VDD/LDO**
1 - 0	-	-	保留。

注意：使用 EOSCR[3:2]设置触摸模块电源(TP)。

11.6.3. 触摸充电控制寄存器 (*TCC*)，地址 = 0x38

位	初始值	读/写	描述		
7	0	-	溢出使能。 0/1：停用/启用 启用后，计数器计满溢出时会自动从零开始计数。		
6 – 4	-	只写	**触摸控制和状态**		
			数据	**命令(W)**	**状态(R)**
			000	TK_STOP（触摸模块掉电）	准备中/ 结束
			001	TK_RUN (Touch START)	运行中
			011	放电（CS 电容放电）	放电中
			其他	保留	保留
3 – 0	-	-	保留		

注意：ByPass 模式：执行 Touch START（将“0x10”写入 TCC 寄存器）命令必须以系统频率为 250KHz（IHRC/64）。
LDO 模式则无此限制。

11.6.4. 触摸按键使能 3 寄存器 (*TKE3*)，地址 = 0x39

位	初始值	读/写	描述
7	0	读/写	使能 PC7/TK23。0/1：停用/启用
6	0	读/写	使能 PC6/TK22。0/1：停用/启用
5	0	读/写	使能 PC5/TK21。0/1：停用/启用
4	0	读/写	使能 PC4/TK20。0/1：停用/启用
3	0	读/写	使能 PC3/TK19。0/1：停用/启用
2	0	读/写	使能 PC1/TK18。0/1：停用/启用
1	0	读/写	使能 PA5/TK17。0/1：停用/启用
0	0	读/写	使能 PC2/TK16。0/1：停用/启用

注意：触摸按键通道与 ADC 通道不可同时选用相同 IO 引脚。

11.6.5. 触摸按键使能 2 寄存器 (*TKE2*)，地址 = 0x3A

位	初始值	读/写	描述
7	0	读/写	使能 PC0/TK15。0/1：停用/启用
6	0	读/写	使能 PA2/TK14。0/1：停用/启用
5	0	读/写	使能 PA1/TK13。0/1：停用/启用
4	0	读/写	使能 PB1/TK12。0/1：停用/启用
3	0	读/写	使能 PB0/TK11。0/1：停用/启用
2	0	读/写	使能 PB3/TK10。0/1：停用/启用
1	0	读/写	使能 PB2/TK9。0/1：停用/启用
0	0	读/写	使能 PA6/TK8。0/1：停用/启用

注意：1. 触摸按键通道与 ADC 通道不可同时选用相同 IO 引脚。

2. 使用 PB0/TK11 作为触摸按键通道时，需注意将 ADC 默认通道改设成其他通道。

11.6.6. 触摸按键使能 1 寄存器 (*TKE1*)，地址 = 0x3B

位	初始值	读/写	描述
7	0	读/写	使能 PA0/TK7。0/1：停用/启用
6	0	读/写	使能 PA4/TK6。0/1：停用/启用
5	0	读/写	使能 PA3/TK5。0/1：停用/启用
4	0	读/写	使能 PB7/TK4。0/1：停用/启用
3	0	读/写	使能 PB6/TK3。0/1：停用/启用
2	0	读/写	使能 PB5/TK2。0/1：停用/启用
1	0	读/写	使能 PB4/TK1。0/1：停用/启用
0	0	读/写	使能 PD0/TK0。0/1：停用/启用

注意：触摸按键通道与 ADC 通道不可同时选用相同 IO 引脚。

11.6.7. 触摸按键充电计数高位寄存器 (*TKCH*)，地址= 0x7A

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

11.6.8. 触摸按键充电计数低位寄存器 (*TKCL*)，地址 = 0x7B

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

11.6.9. 触摸参数设置寄存器(*TPS*)，IO 地址 = 0x3C

位	初始值	读/写	描述
7 - 0	0x00	读/写	系统保留 保持默认值或填 0x00

注意：TPS = 0x00

11.6.10. 触摸参数设置寄存器 2 (*TPS2*)，IO 地址 = 0x3D

位	初始值	读/写	描述
7 - 6	-	读/写	触摸工作模式：(不同电源的具体设置请参照章节 11.6) 00：模式_A (ByPass 模式，CS 电容接 VDD) 01：模式_B (LDO 模式，CS 电容接 GND)
5 - 2	0000	读/写	系统保留，请填 0
1 - 0	00	读/写	01：VREF 续电保持。初始化后建议填入 0x01

示例：

```
// Bypass 模式:
$ EOSCR TK_VDD;
$ TPS2 Type A, Always On        // ByPass, Type A, CS 电容接 VDD

// LDO 模式
$ EOSCR TK 2V,TK_LDO
$ TPS2 Type B, Always_On        // LDO, Type B, CS 电容接 GND
```

11.7. 可编程频率产生器 (PFG)

PFC460 提供一个可编程频率产生器 PFG（Programmable Frequency Generator）用于精准频率输出。一般可用于需要较高频率的输出的应用，如雾化加湿器…等。其硬件框图如下图 43 所示。

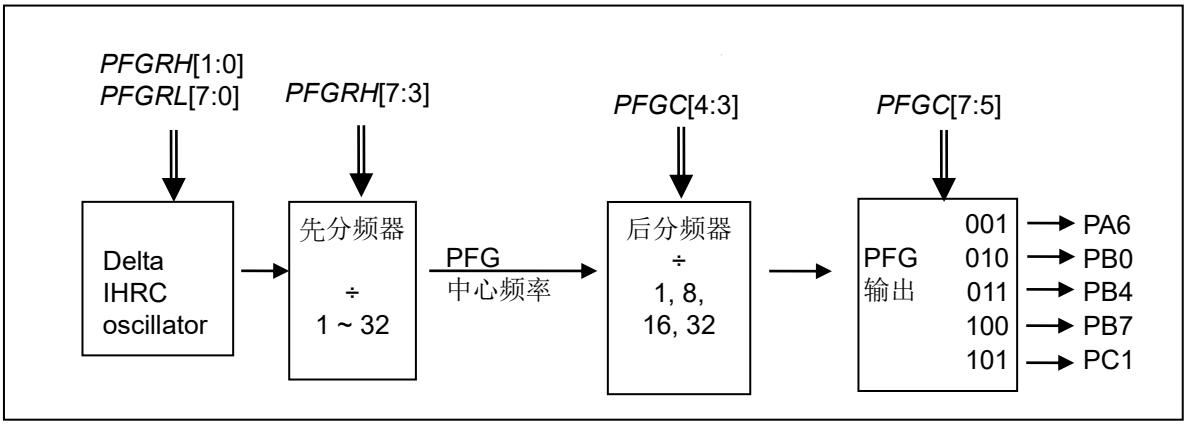


图 43：PFG 硬件电路的模块框图

PFG 振荡电路由 Delta IHRC oscillator 单独提供，与 IHRC 相互独立。Delta IHRC oscillator 可经由 10-bits 校准调整寄存器做振荡频率调整。经由设置寄存器 *PFGRH*[1:0]和 *PFGRL*[7:0]，可将 Delta IHRC 频率调整至 36MHz 使用，共有 1024 级微调，分±512 级，每级精度大约 0.12%。系统默认的 Delta IHRC 频率大约为 36MHz（*PFGRH*[1:0]=0x2，*PFGRL=0x00*）。

Delta IHRC oscillator 经寄存器 *PFGRH*[7:3]做前分频以产生 PFG 中心频率，之后再将 PFG 中心频率做后分频处理，用以支持更多系统需求，最后由 PFGC[7:5]设置决定输出到 IO 引脚。

Delta IHRC oscillator 可能为非精准 36MHz 振荡器。其会受制造工艺、工作电压、工作温度…等因素影响而使其振荡频率在 36MHz 附近飘移。建议在使用 PFG 前必须先对 Delta IHRC oscillator 先做频率的校正，频率校正可以使用内部 IHRC 或是系统运行时间当标准的时钟校准源。校正完后重新填写入合适的 10-Bits 校正数值 (*PFGRH*[1:0]+*PFGRL*[7:0])。

11.7.1. PFG 中心频率

设置寄存器 *PFGRH*[7:3], 可设定所需的 PFG 中心频率，常用 PFG 中心频率如下表所示, 分别为 3.3MHz / 3MHz / 2.4MHz / 1.7MHz。对于其他频率，客户仍可依需求自行配置。

Delta_IHRC (MHz)	*PFGRH*[7:3] (十进制)	Result (MHz)
36	11	3.27
36	12	3.00
36	15	2.40
36	21	1.71

表 14：PFG 中心频率配置参数

PFG 中心频率計算公式如下：

PFG 中心频率 = Y ÷ S1

这里，

Y = PFGRH[1:0]+PFGRL[7:0] : Delta IHRC oscillator 频率

S1 = PFGRH[7:3] : 預分頻寄存器设定的值（十进制）

11.7.2. PFG 输出端口

设置后分频器寄存器 *PFGC*[4:3]可再将中心频率经由 1 分频/8 分频/16 分频/32 分频，而设置寄存器 *PFGC*[7:5]后,系统会自动将 PFG 信号由对应端口输出。

设定 PFG 输出通道后即启用 Delta IHRC oscillator。当 *PFGC*[7:5]=000 时，则停用 PFG 模块。

PFG 输出频率 = Y ÷ S1 ÷ S2

这里，

Y = PFGRH[1:0]+PFGRL[7:0] : Delta IHRC oscillator 频率

S1 = PFGRH[7:3] : 預分頻寄存器设定的值（十进制）

S2 = PFGC[4:3] : 后分頻寄存器设定，共有 4 種后分頻，分别為/1 分频，/8 分频，/16 分频，/32 分频

11.7.3. PFG 相关寄存器

11.7.3.1. Delta IHRC 微调高位元寄存器(*PFGRH*), 地址 = 0x30

位	初始值	读/写	描 述
7 – 3	0x0b	读/写	分频寄存器，中心频率 = Delta IHRC / *PFGRH*[7:3]
2	-	-	保留
1 – 0	0x2	读/写	Delta IHRC 微调高位元寄存器

11.7.3.2. Delta IHRC 微调低位元寄存器(*PFGRL*), 地址 = 0x31

位	初始值	读/写	描 述
7 – 0	0x00	读/写	Delta IHRC 微调低位元寄存器

注意：*PFGRL* 的配置要写在 *PFGRH* 之后。

11.7.3.3. PFG 控制寄存器(*PFGC*), 地址 = 0x32

位	初始值	读/写	描 述
7 – 5	000	读/写	PFG 输出端口选择 000：PFG Disable（Delta IHRC oscillator Disable） 001：PA6 010：PB0 011：PB4 100：PB7 101：PC1 其他：保留
4 – 3	00	只写	PFG 后分频器。 00：÷ 1 01：÷ 8 10：÷ 16 11：÷ 32

12. 仿真注意事项

可使用 6S-M-001 对 PFC460 进行仿真，但仍请注意以下事项：

(1) 可支持单核仿真除错模式和单/多核全速运行模式。

(2) 仿真时将会占用部分内存空间：ROM 从 0xD00 后被占用，RAM 从 0x1F8 ~ 0x1FF 被占用。

(3) 仿真时 PD0 / PD1 端口将被占用。

(4) 其他仿真注意事项请至官网查阅参考 6S-M-001 的使用手册。

13. 烧录方法

PFC460 的烧录脚为 VDD, PA3, PA5, PA6, GND。

请使用 5S-P-003x 进行烧录。3S-P-002 或之前的烧录器皆不支持烧录 PFC460。

Jumper 连接：可依照烧录器软件上的说明，连接 jumper 即可。

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 PFC460 合封的 IC 及元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 5.8V。

(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. 在板烧录模式（On-Board Writing）

PFC460 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 5S-P-003x 上五根引线：ICPCK(PA3)、ICPDA(PA6)、VDD、GND 和 ICVPP(PA5)，用于与 IC 上的 PA3、PA6、VDD、GND 和 PA5 对应相连。

若要启动在板烧录，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-003x 的用户手册）。

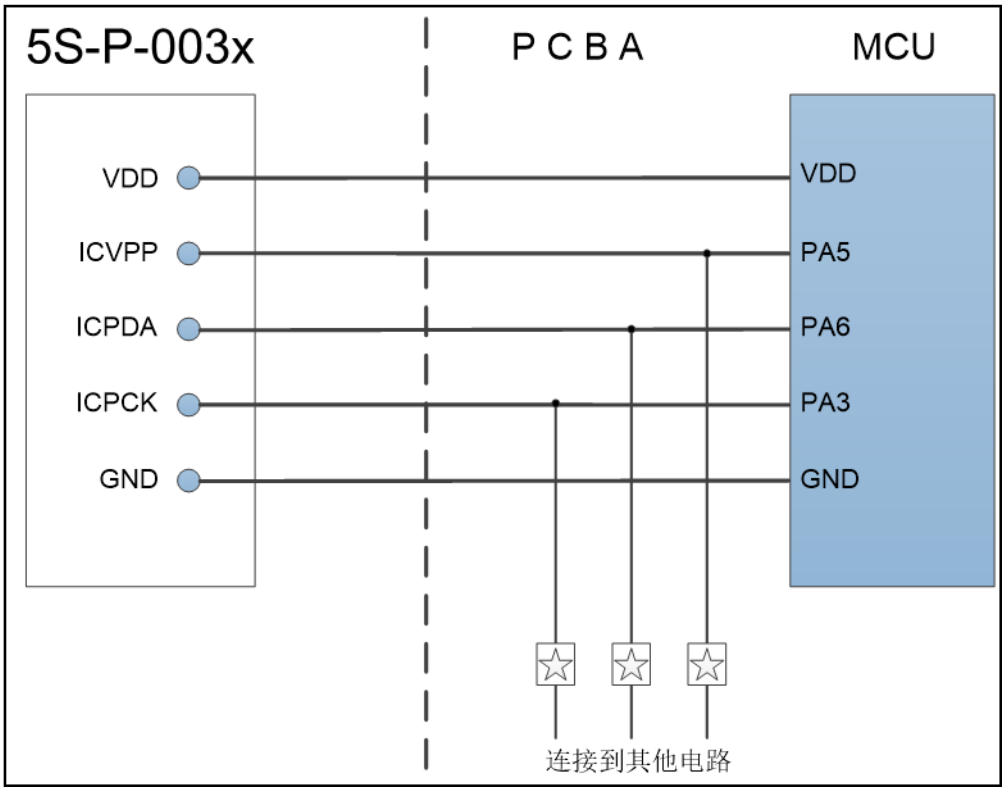


图 44：在板烧录接线示意图

图 44 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：

- 在板烧录时的各电特性条件与普通烧录模式相同，VDD 最高可达 7.5V，请参考普通烧录模式的详细说明。
- PCB 上的 VDD 与 GND 之间**不可**连接有 7.5V 或以下的稳压二极管或其他钳制 7.5V 产生的电路或元件。
- 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**作为强输出脚。

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# / 2.1		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		VDD + 0.3	V	
工作温度	-40		85	°C	
储藏温度	-50		125	°C	
节点温度		150		°C	

首批晶元特性参数

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# / 2.1		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 0	 59K	 8M 4M 2M	Hz	 VDD≧4.5V#；VDD≧4.0V VDD≧2.7V#；VDD≧2.4V VDD≧2.2V#；VDD≧2.1V VDD = 5V
P_{cycle}	烧录次数	1000			cycles	
f_{pwm}	PWMG0/1/2 的时钟源频率			32	MHz	仅当 VDD ≥ 3.0 V 时，32 MHz 可用
I_{OP}	工作电流		0.89 96		mA uA	f_{SYS}=IHRC/16=1MIPS@5V f_{SYS}=ILRC=59KHz@5V
I_{PD}	掉电模式消耗电流 （用 stopsys 命令）		0.8 1.2		uA uA	fSYS= 0Hz,VDD=5V fSYS= 0Hz,VDD=5V,NILRC 使能
I_{PS}	省电模式消耗电流 （用 stopexe 命令）		3.5 4.0		uA uA	VDD =5V; fSYS= ILRC VDD =5V; fSYS= ILRC, NILRC 使能 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_{OH}	IO 输出驱动电流 PB2~PB7 (Strong) PB2~PB7 (Normal) Others IO		 -28 -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
I_{OL}	IO 输出灌电流 PA0~PA4 PB0 (Strong) PB0 (Normal) PB2~PB7 (Strong) PB2~PB7 (Normal) Others IO		 20 102 20 78 20 14		mA	V_{DD}=5V, V_{OL}=0.5V
V_{IN}	输入电压	-0.3		V_{DD}+0.3	V	
$I_{INJ\ (PIN)}$	脚位的引入电流			1	mA	V_{DD}+0.3≧V_{IN}≧ -0.3
R_{PH}	上拉电阻		80		KΩ	V_{DD}=5.0V
R_{PL}	下拉电阻		81		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*
f_{DIHRC}	Delta IHRC 振荡频率（校准后） *		36		MHz	V_{DD} = 5.0V
f_{ILRC}	ILRC 频率 *		59		KHz	V_{DD} = 5.0V
f_{NILRC}	NILRC 频率 *		15		KHz	
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		0.93 0.83		mA	@5V @3V
ADclk	ADC 时钟周期		2		us	3.9V ~ 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

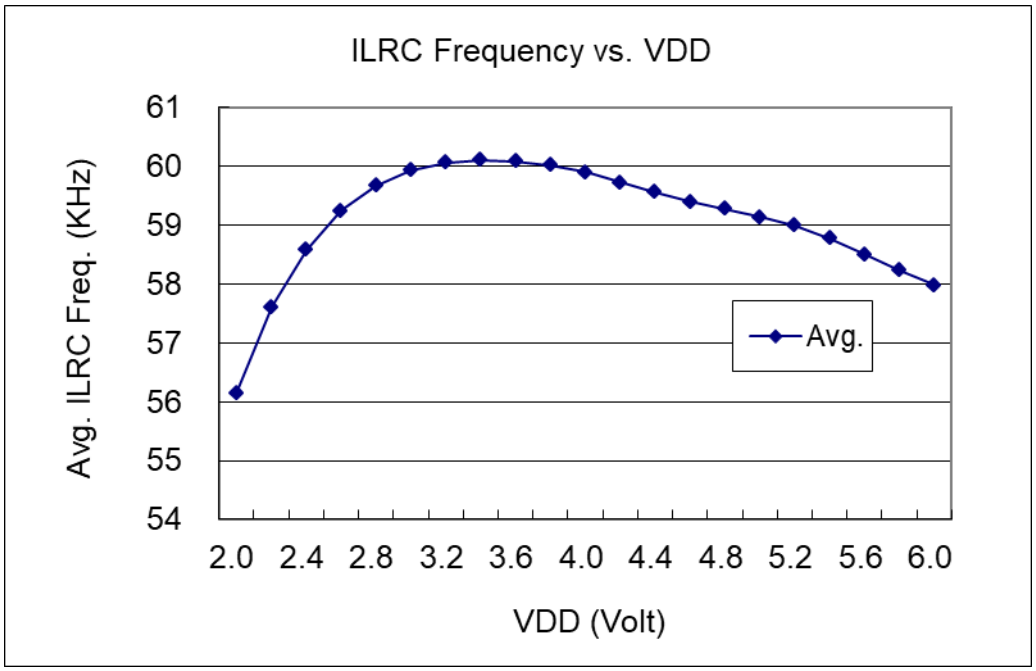
符 号	特 性	最小值	典型值	最大值	单位	条 件
CPos	比较器偏压*	-	±10	±20	mV	
CPcm	比较器共模输入电压*	0		V_{DD} -1.5	V	
CPspt	比较器响应时间**		100	500	ns	上升沿和下降沿一样
CPmc	比较器模式改变稳定时间		2.5	7.5	us	
CPcs	比较器电流消耗		25		uA	V_{DD} = 5V
OPAcm	OPA 共模输入电压*	0		VDD -1.3	V	
OPAos	OPA 偏压*		±10		mV	V_{DD} =5V
I_{OPA}	OPA 输出电流*	200			uA	
OPAgain	OPA 直流增益*		80		dB	

首批晶元特性参数

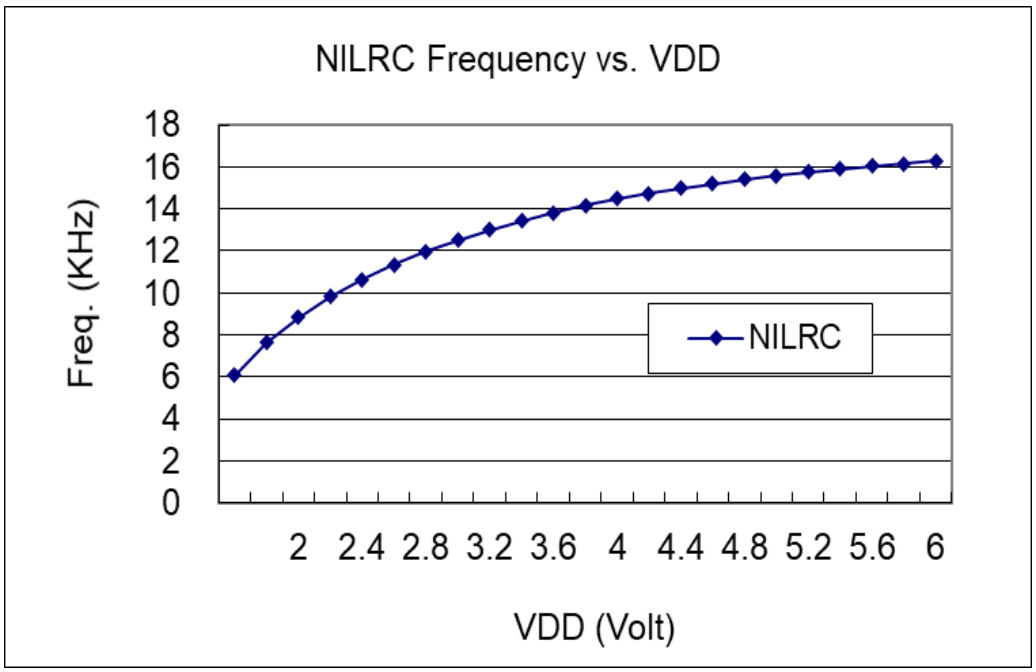
*这些参数是设计参考值，并不是每个芯片测试。

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。

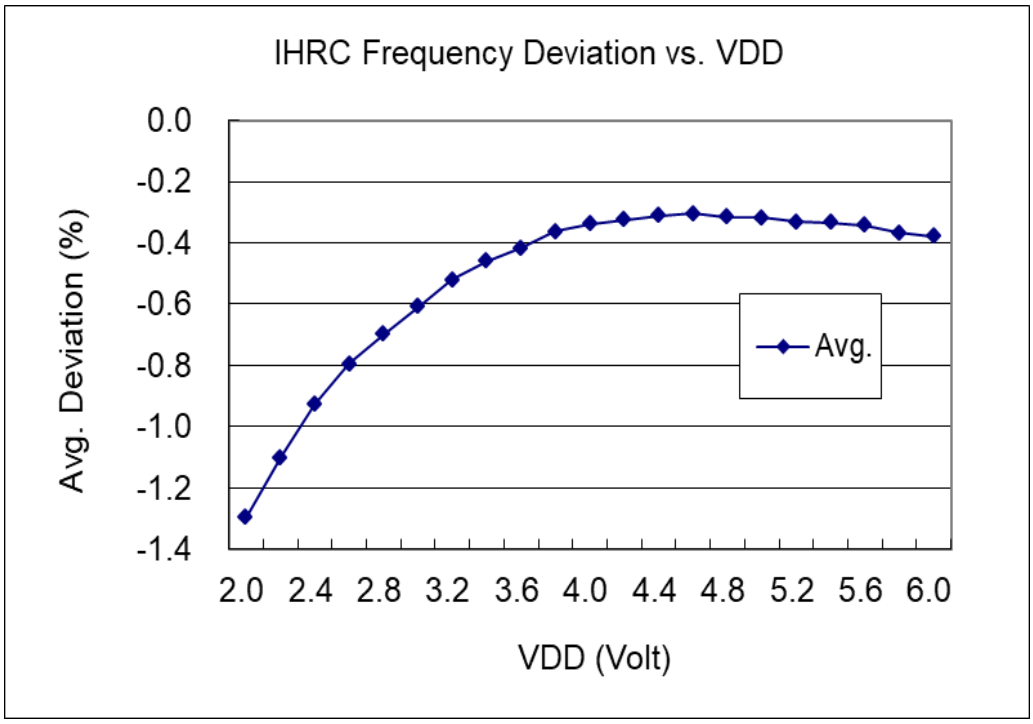
14.3. ILRC 频率与 VDD 关系曲线图



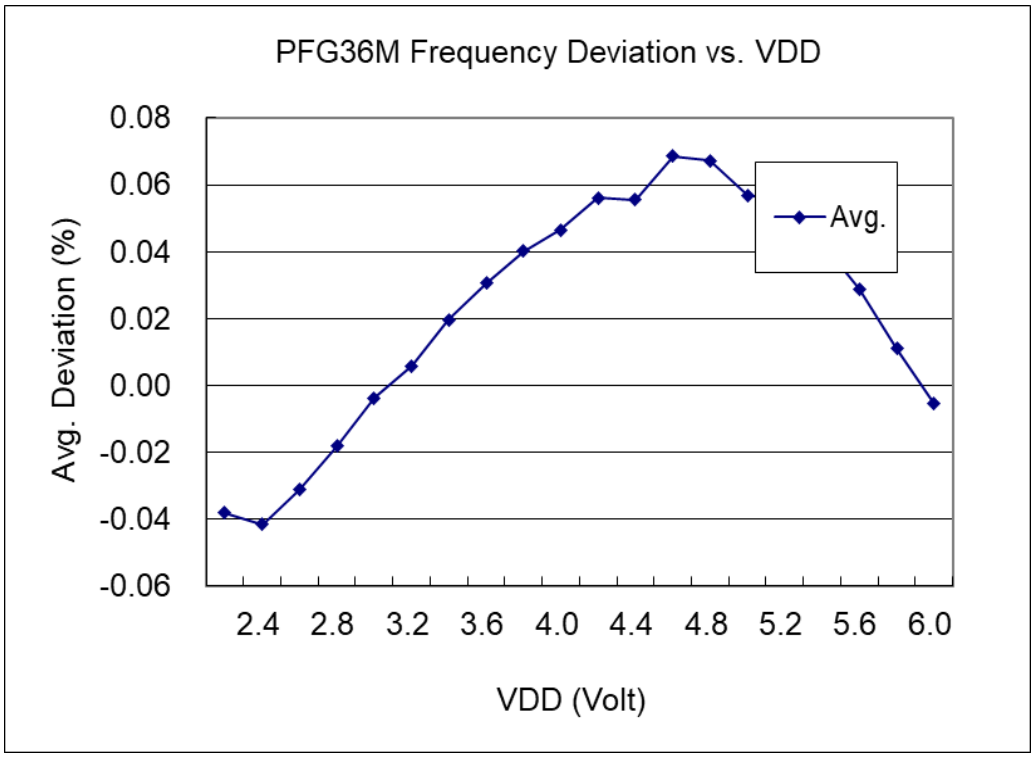
14.4. NILRC 频率与 VDD 关系曲线图



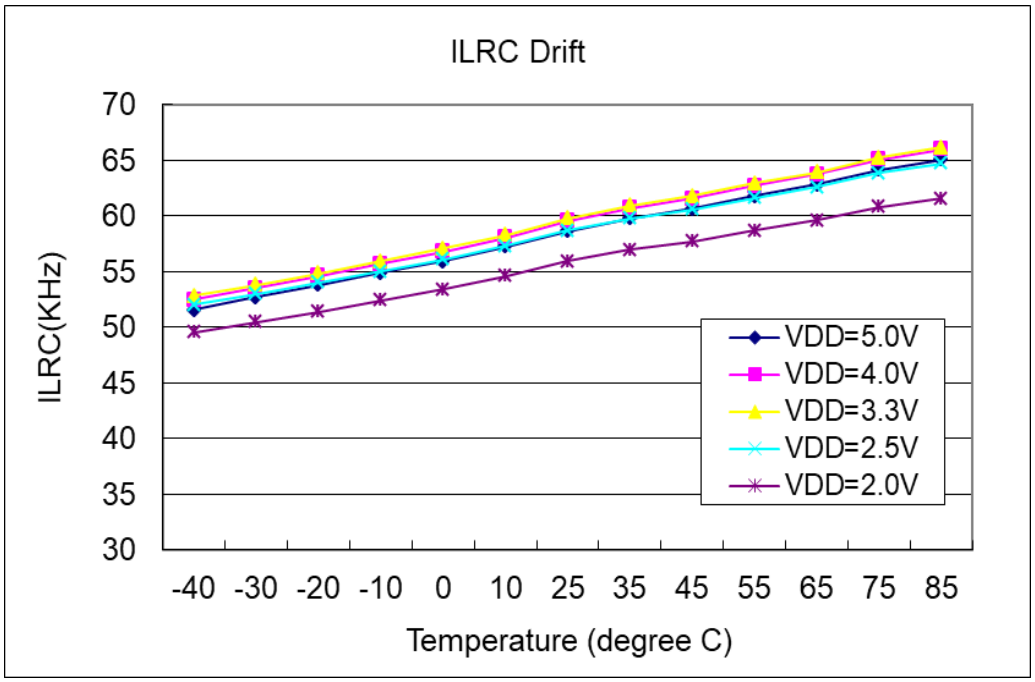
14.5. IHRC 频率与 VDD 关系曲线图（校准到 16MHz）



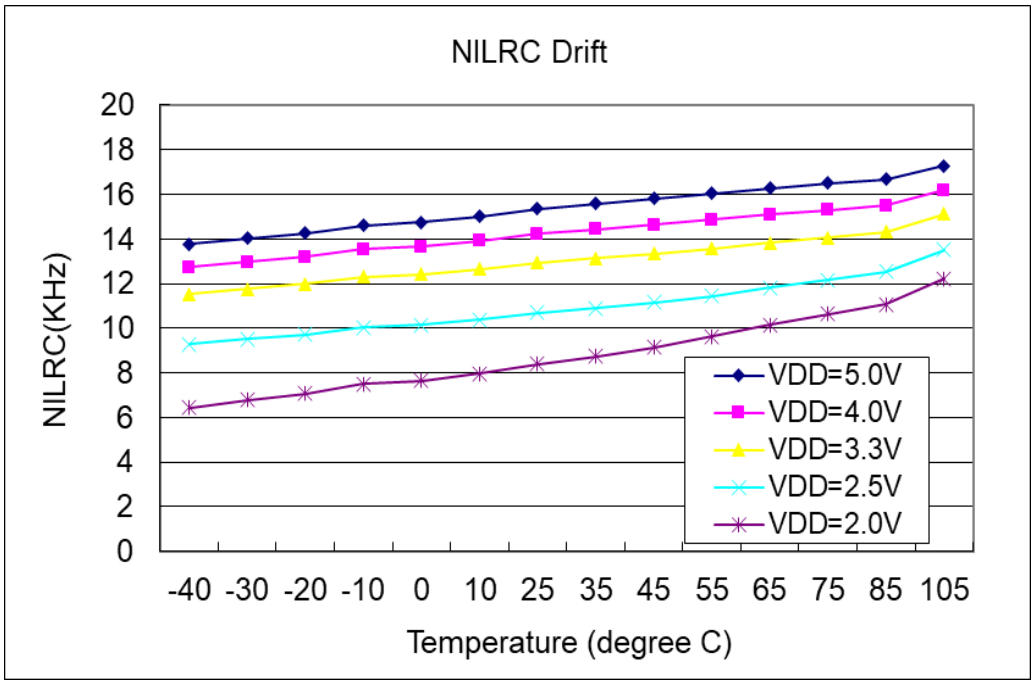
14.6. PFG 频率与 VDD 关系曲线图（校准到 36MHz）



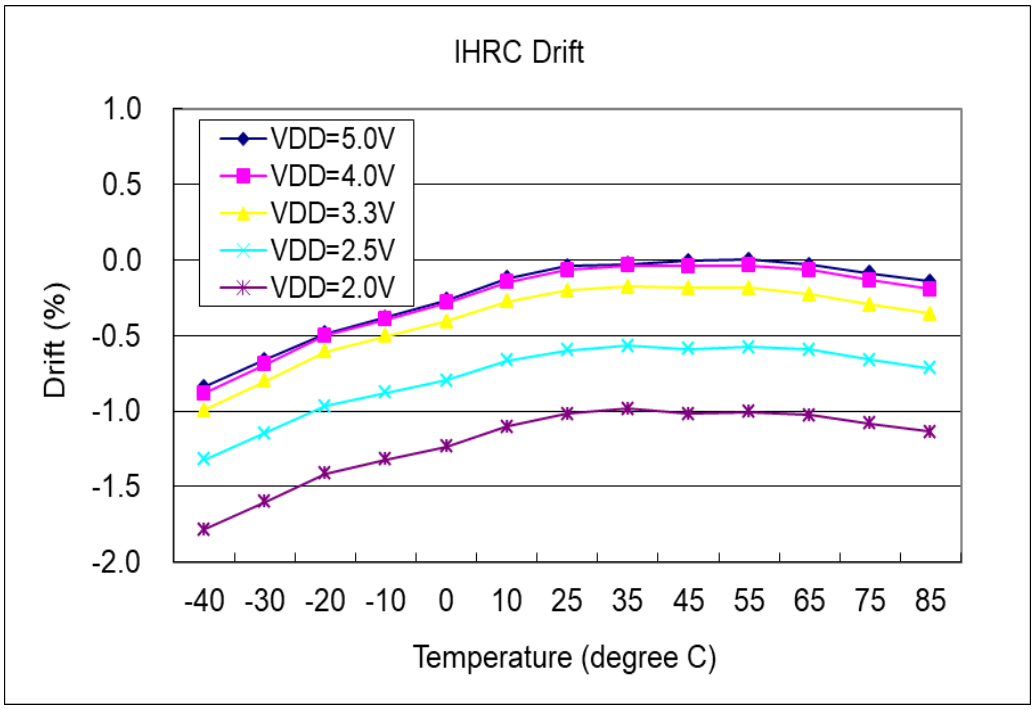
14.7. ILRC 频率与温度关系曲线图



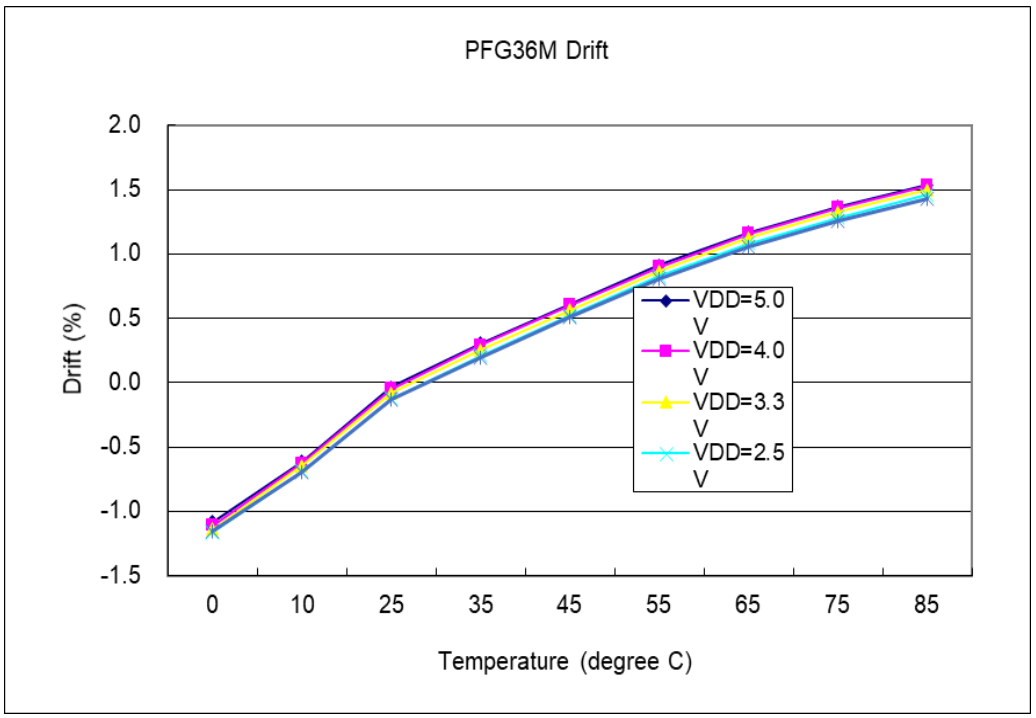
14.8. NILRC 频率与温度关系曲线图



14.9. IHRC 频率与温度关系曲线图（校准到 16MHz）



14.10. PFG 频率与温度关系曲线图（校准到 36MHz）

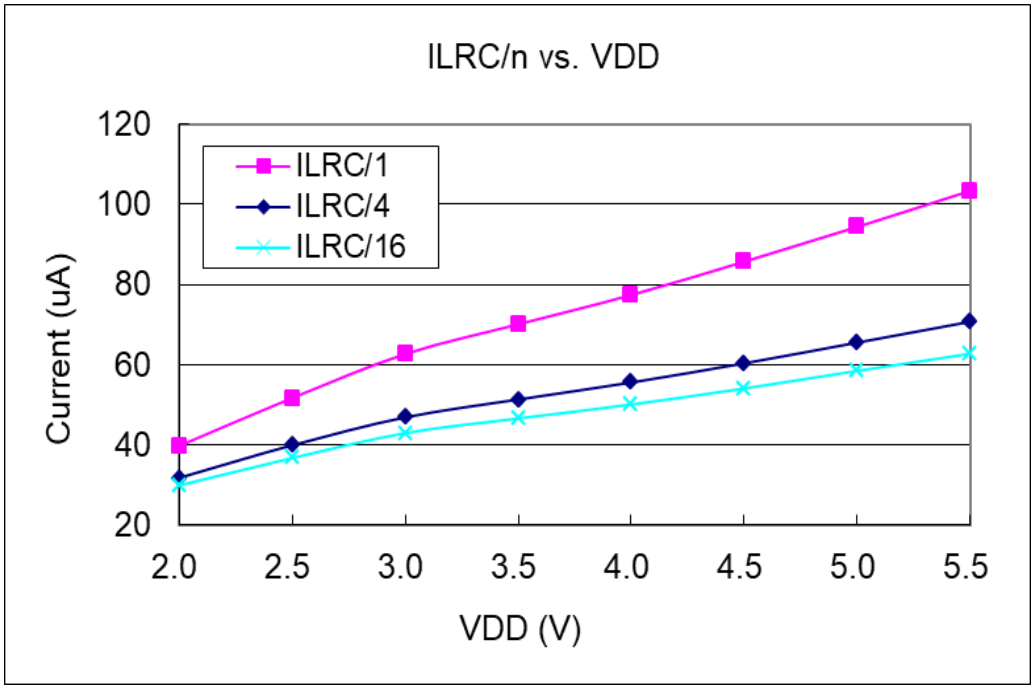


14.11. 工作电流与 VDD、系统时钟 CLK=ILRC/n 曲线图

条件：1-FPPA (FPPA0: tog PA0)

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

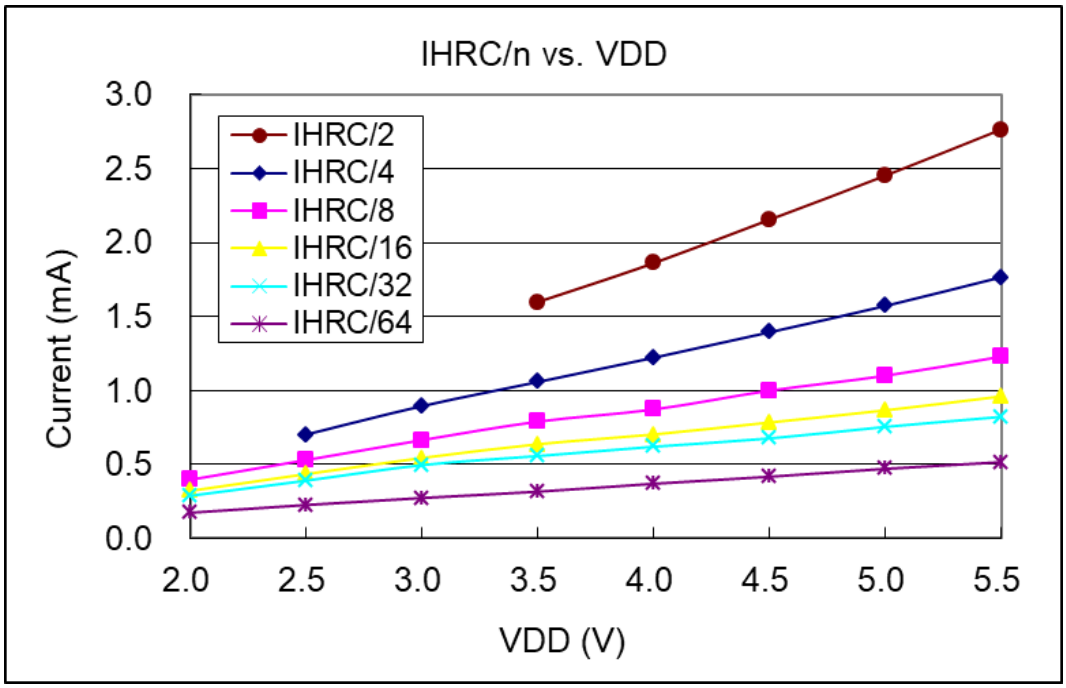


14.12. 工作电流与 VDD、系统时钟 CLK=IHRC/n 曲线图

条件：1-FPPA (FPPA0: tog PA0)

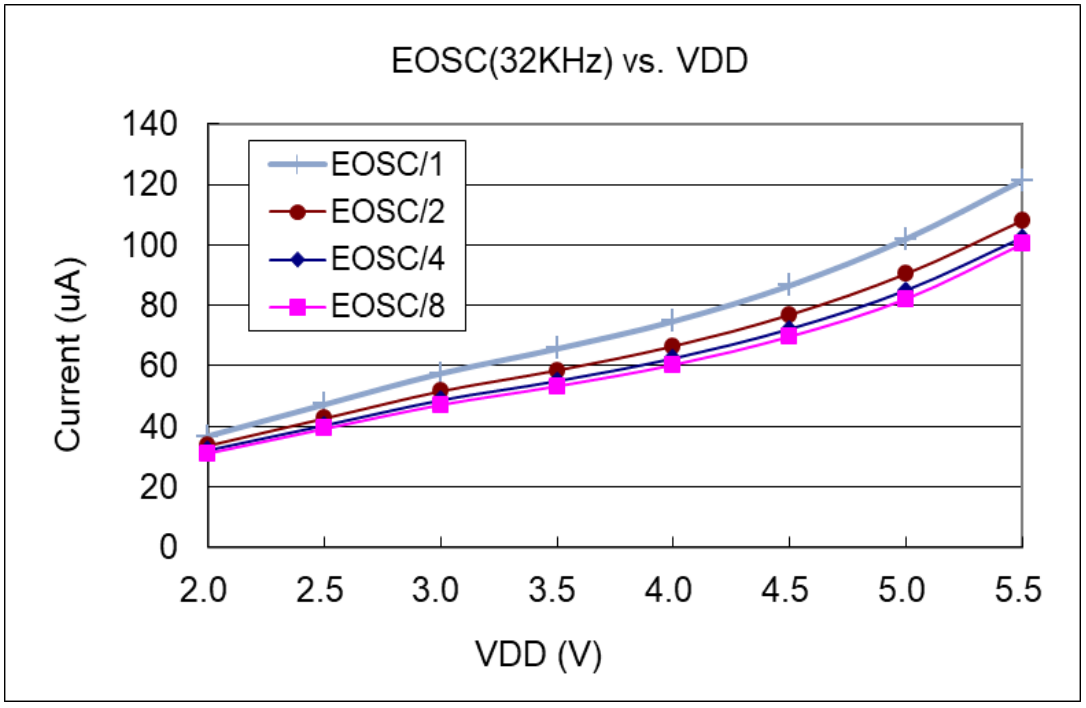
开启的硬件模块：IHRC，Bandgap，LVR；**关闭**的硬件模块：ILRC, EOSC, LVR, T16, TM2, TM3, ADC 模块；

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



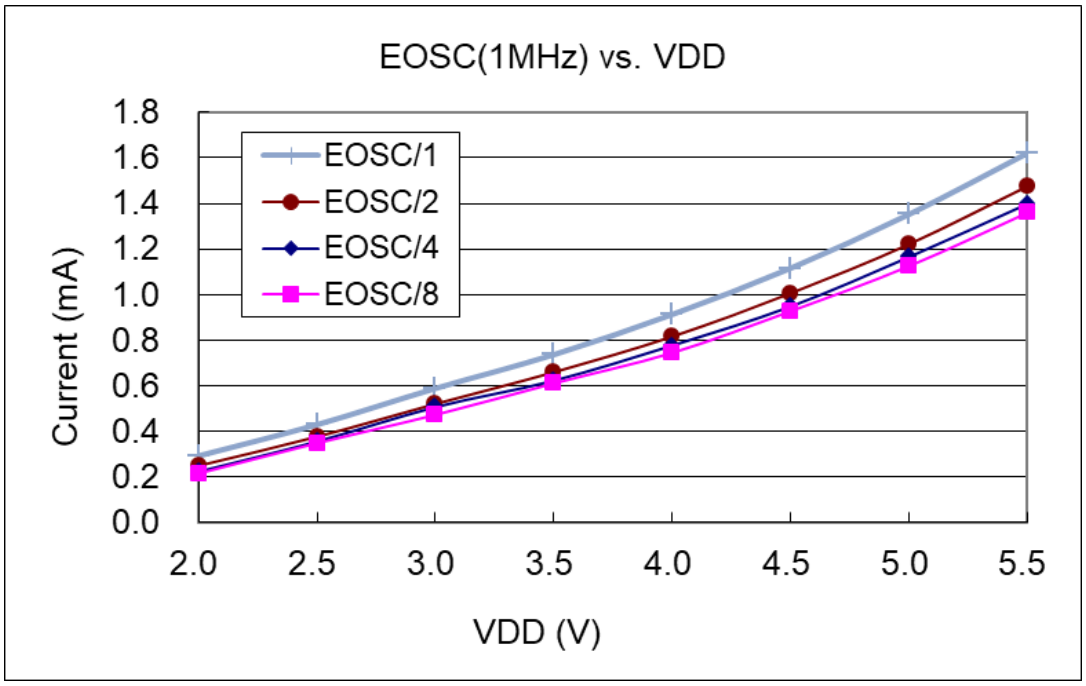
14.13. 工作电流与 VDD、系统时钟 CLK=32KHz EOSC/n 曲线图

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



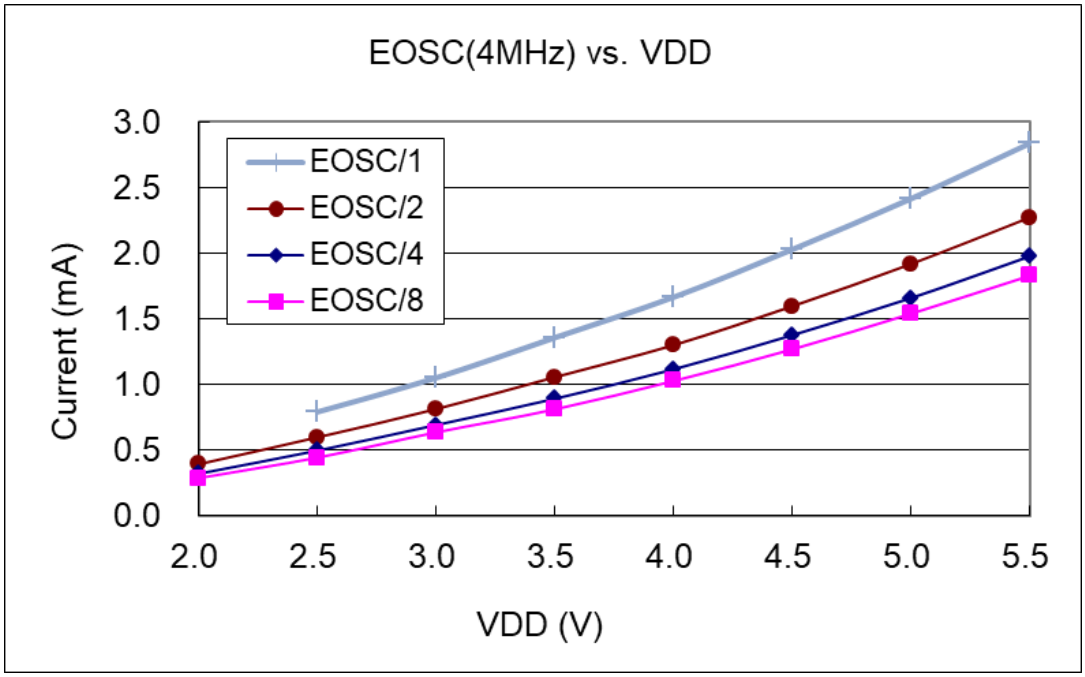
14.14. 工作电流与 VDD、系统时钟 CLK=1MHz EOSC/n 曲线图

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



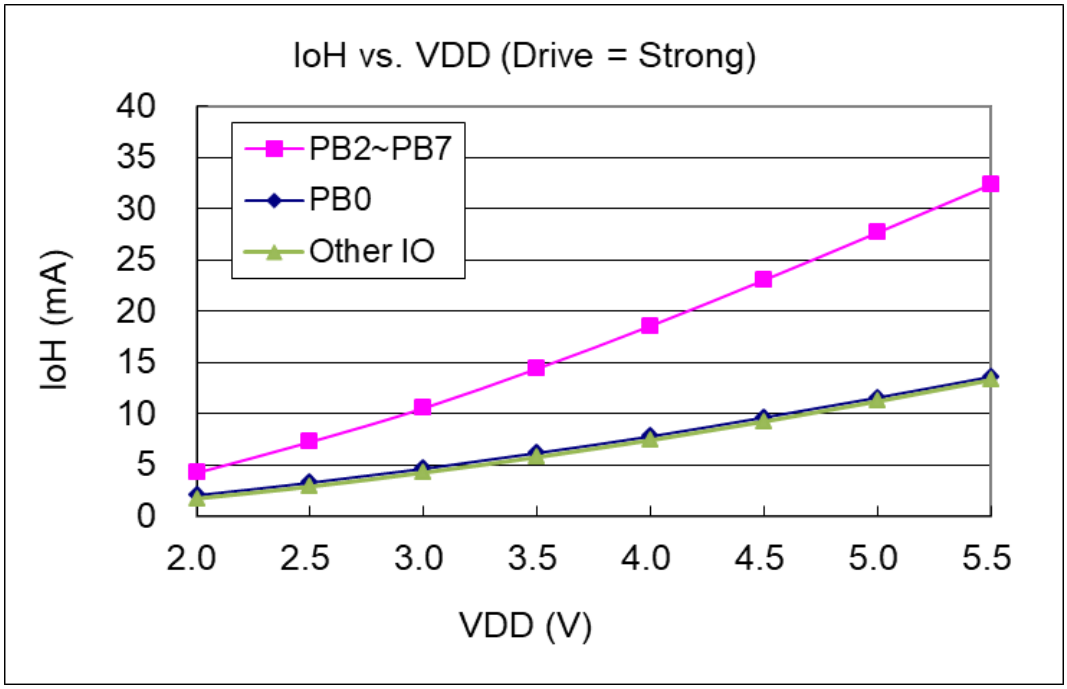
14.15. 工作电流与 VDD、系统时钟 CLK=4MHz EOSC/n 曲线图

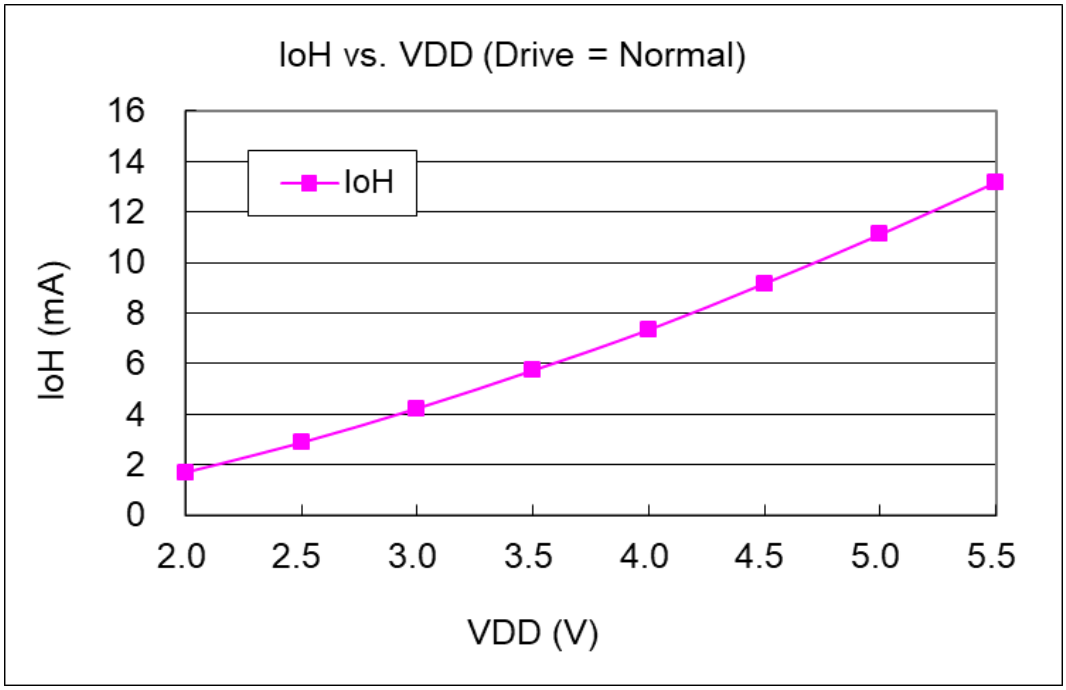
条件：**开启**的硬件模块：EOSC，MISC.6 = 1，Bandgap，LVR；
关闭的硬件模块：IHRC, ILRC, T16, TM2, TM3, ADC 模块；
IO 引脚：PA0 以 0.5Hz 频率高低电压交换输出，无负载；**其他引脚**：设为输入且不浮空

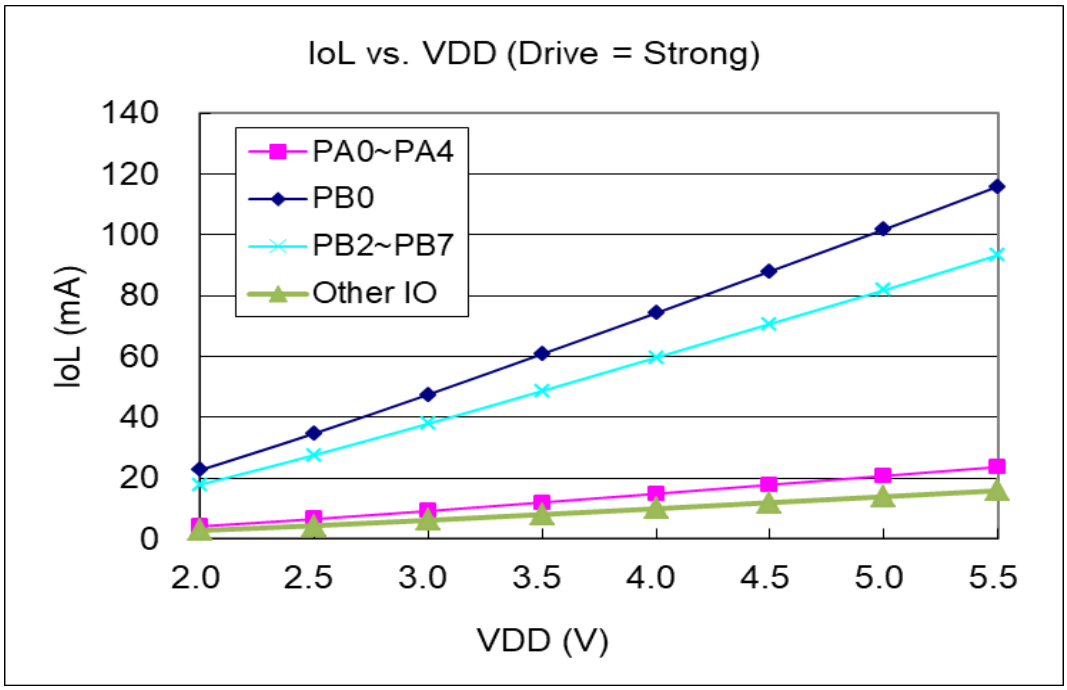


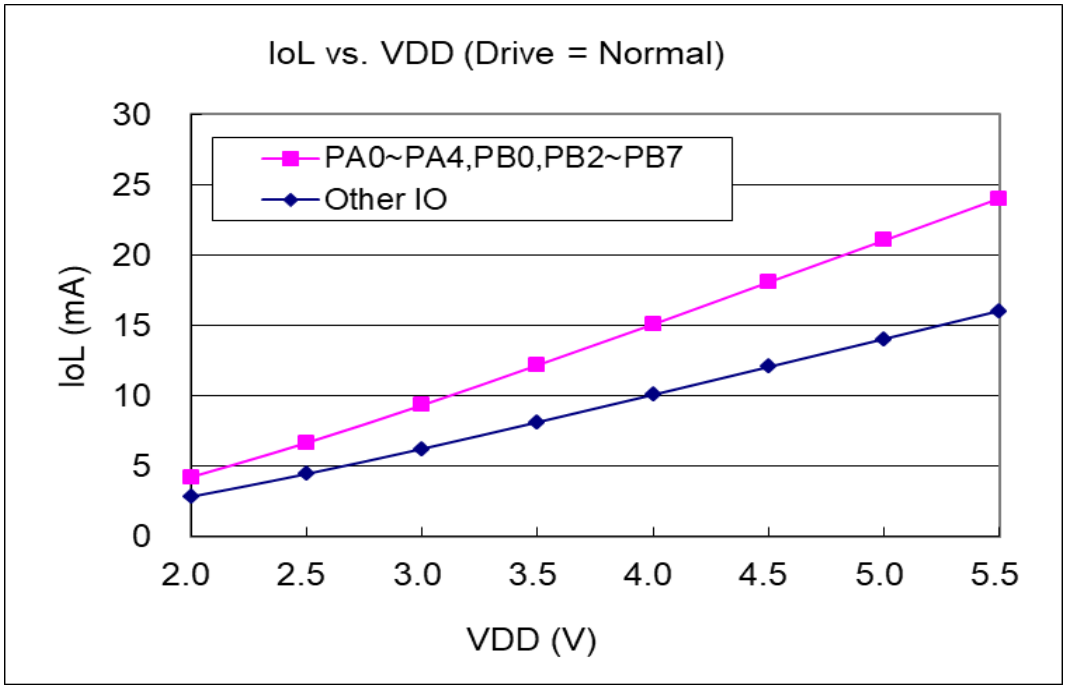
14.16. 引脚输出驱电流(I_{OH})与灌电流(I_{OL}) 曲线图

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

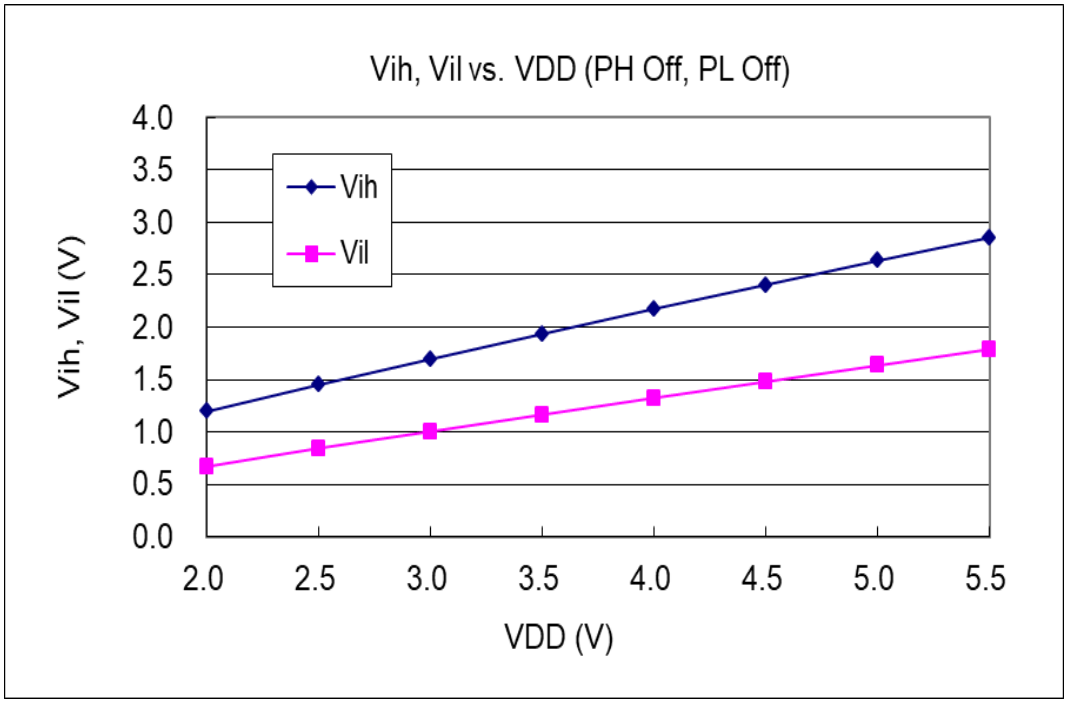



IoH vs. VDD (Drive = Normal)
IoH (mA)
16
14
12
10
8
6
4
2
0
IoH
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
VDD (V)

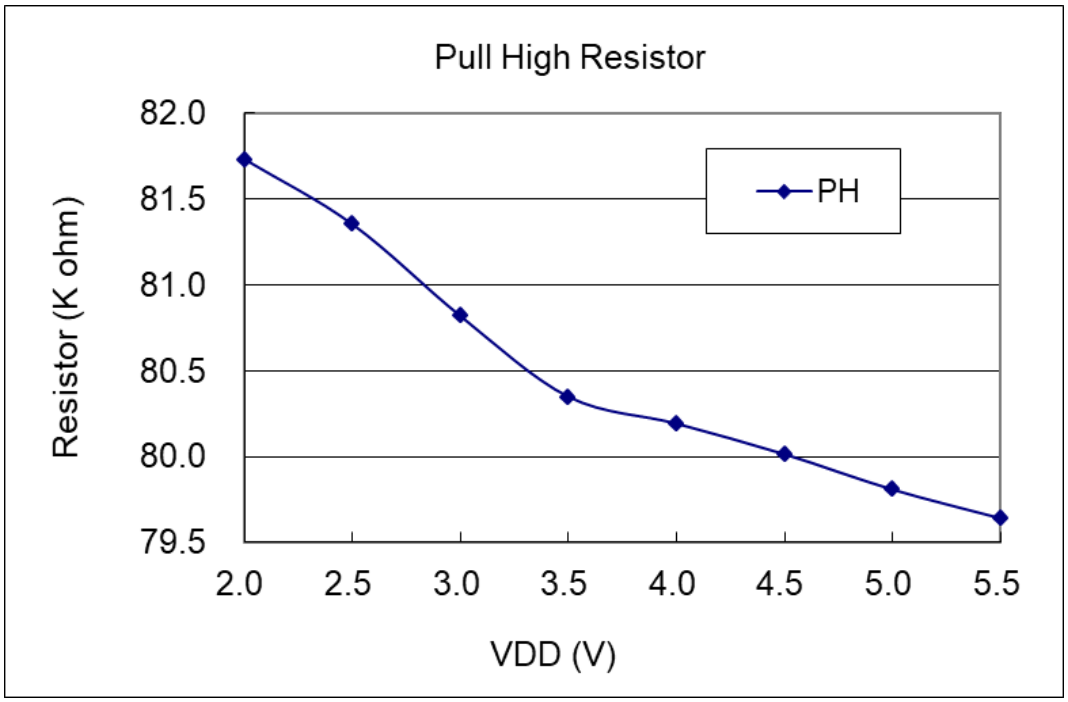

IoL vs. VDD (Drive = Strong)
IoL (mA)
140
120
100
80
60
40
20
0
PA0~PA4
PB0
PB2~PB7
Other IO
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
VDD (V)

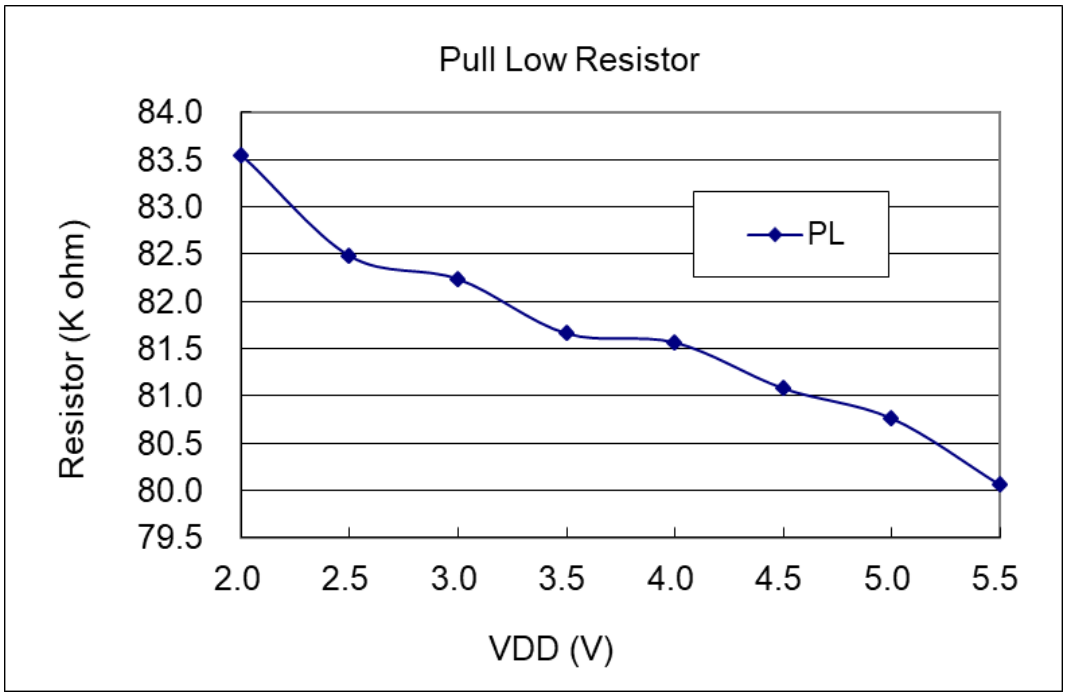


14.17. 引脚输入高电压与低电压(V_{IH}/V_{IL}) 曲线图

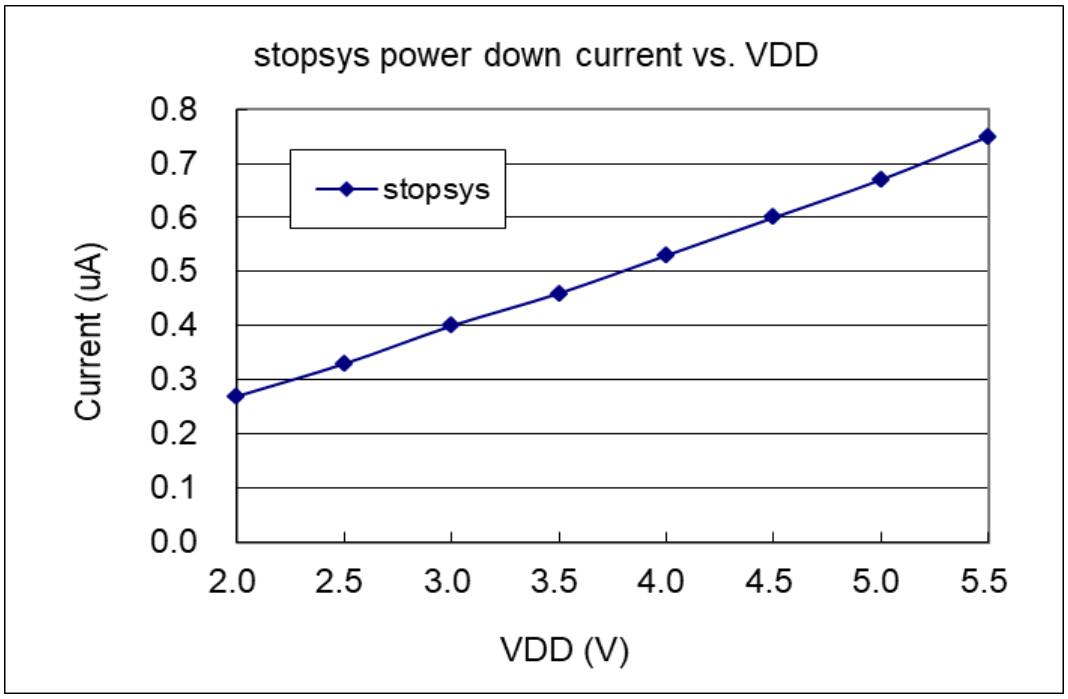


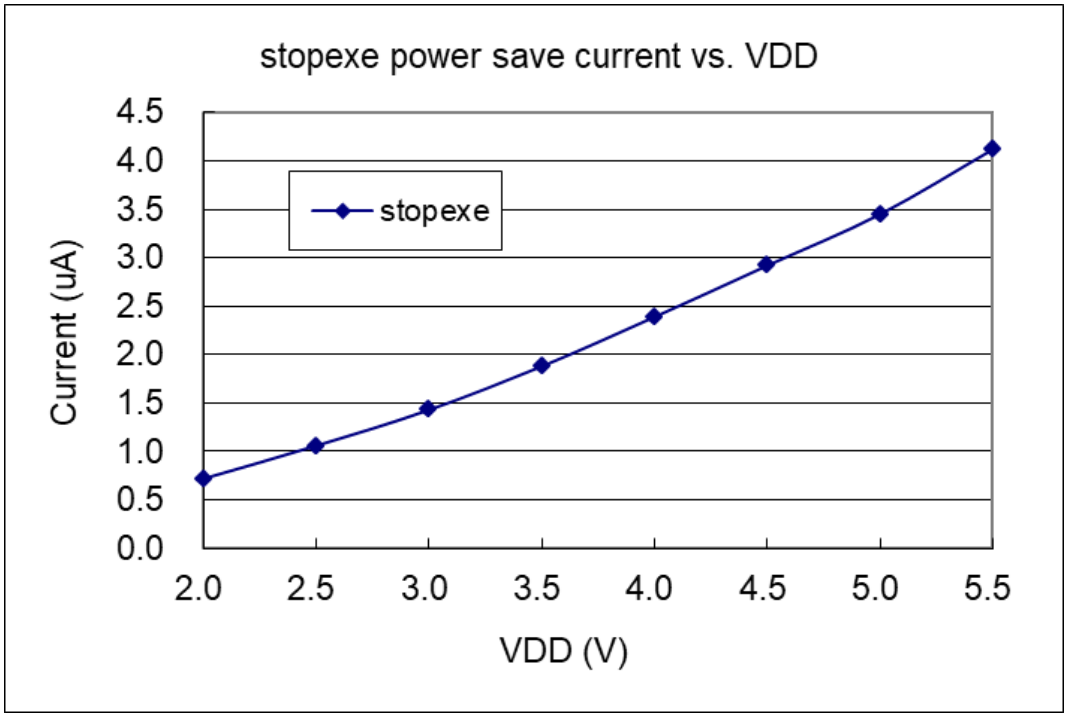
14.18. 引脚上拉/下拉电阻曲线图





14.19. 掉电电流(IPD)与省电电流(IPS) 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，只允许寻址在 address 0~0x3F (0~63) 的位置
M.n	可在 SRAM 全空间内寻址
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

14.20. 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』

14.21. 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』

14.22. 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』

14.23. 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』

14.24. 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
	…
	--- 	

14.25. 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』

14.26. 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』

14.27. 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