PGS134

8bit MTP 带 12bit ADC & EEFROM 单片机

数据手册

第 0.00 版

2022 年 8 月 18 日

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

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

重要声明

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

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

应广科技不承担任何责任来自于因客户的产品设计所造成的任何损失。在应广科技所保障的规格范围内，客户应设计和验证他们的产品。为了尽量减少风险，客户设计产品时，应保留适当的产品工作范围安全保障。

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

目 录

修订历史

修订	日期	描述
0.00	2022/08/18	初版

使用警告

在使用 IC 前，请务必认真阅读 PGS134 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。

1. 功能

1.1. 特性

- 通用 MTP 系列
- 不建议使用于 AC 阻容降压供电或有高 EFT 要求的应用。应广不对使用于此类应用而不达安规要求负责
- 工作温度范围：-40°C ~ 85°C

1.2. 系统特性

- 4KW 程序存储器
- 512 Bytes EEPORM
- 256 Bytes 数据存储器（128*16）
- 一个硬件 16 位计数器
- 两个 8 位硬件 PWM 生成器
- 三个 11 位硬件 PWM 生成器(PWMG0，PWMG1 & PWMG2)
- 一个硬件比较器
- Bandgap 电路提供 1.2V 参考电压
- 最多 14 通道 12 位 ADC，其中一个来自于内部 bandgap 参考电压或者 0.25*V_{DD}
- ADC 参考高电压：外部输入，内部 VDD，Bandgap 1.2V、1.6V、2.0V、2.4V、3.0V 及 4.0V
- 一组 1T 8x8 硬件乘法器
- 最多 22 个 IO 引脚并带有上拉及下拉电阻
- 提供两种不同的 IO 驱动能力以满足不同的应用需求
  1. PB4，PB7 驱动/灌电流= 30mA/35mA (Strong) and 13mA/17mA (Normal)
  2. 其他 IO 驱动/灌电流 = 11mA/(13 or 20) mA
- 每个 IO 引脚都可设定唤醒功能
- 内置 1/2 V_{DD} LCD 偏置电压产生器，可支持最大 4×17 点的 LCD 屏
- 时钟源：IHRC，ILRC 及 EOSC(XTAL)
- 对所有带有唤醒功能的 IO，都支持两种可选择的唤醒速度：正常唤醒和快速唤醒
- 8 段 LVR 复位电压设定：4.0V，3.5V，3.0V，2.7V，2.5V，2.2V，2.0V，1.8V
- 两组 Code Option 可选的外部中断引脚

1.3. CPU 特性

- 8bit 高性能 RISC CPU
- 提供 98 个有效指令
- 大部分都是 1T（单周期）指令
- 可程序设定的堆栈指针和堆栈深度
- 数据存取支持直接和间接寻址模式，用数据存储器即可当作间接寻址模式的数据指针(index pointer)
- IO 地址以及存储地址空间互相独立

1.4. 订购/封装信息

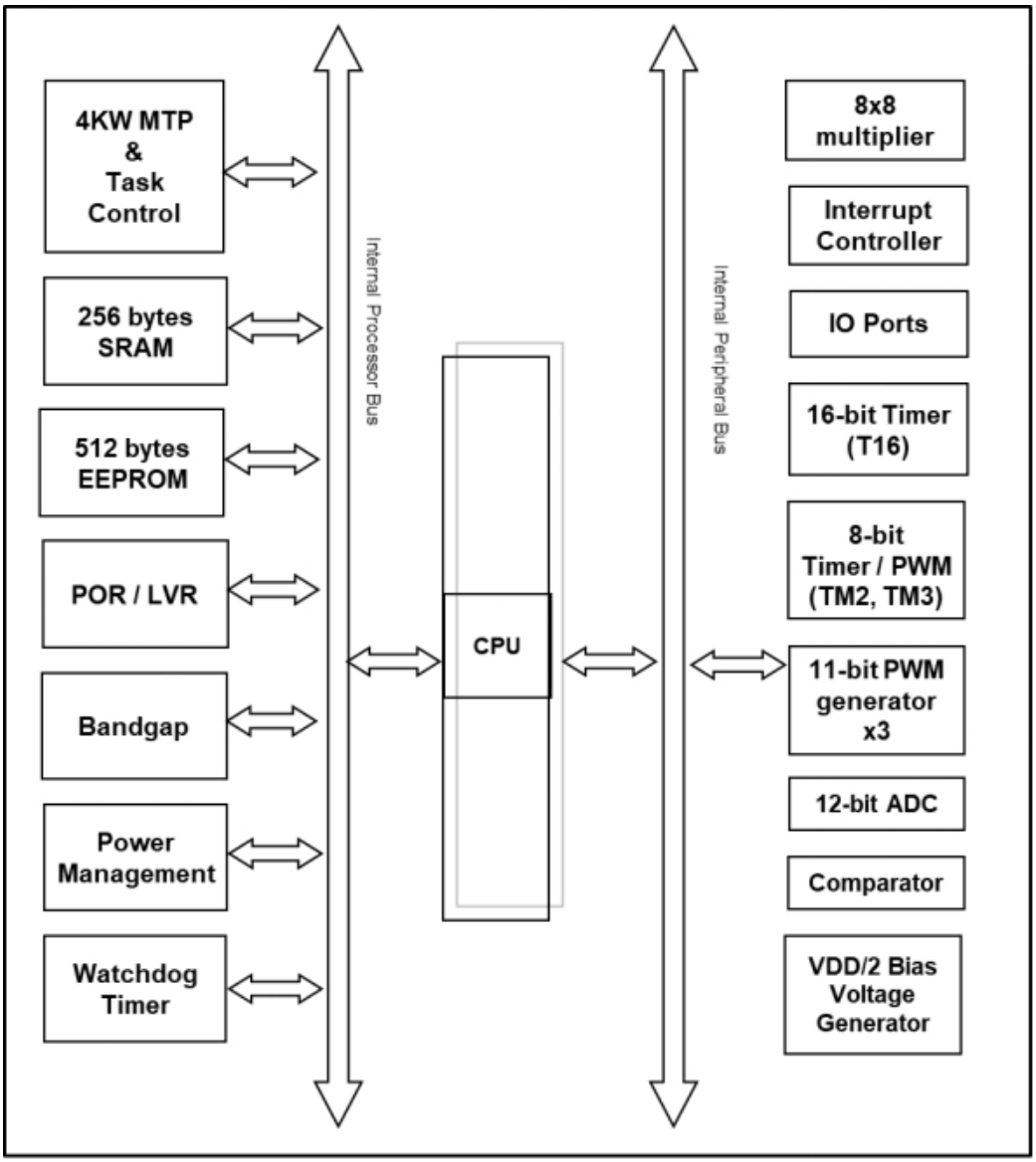
- ✧ PGS134-S08: SOP8 (150mil)
- ✧ PGS134-M10: MSOP10 (118mil)
- ✧ PGS134-S14: SOP14 (150mil)
- ✧ PGS134-S16A: SOP16A (150mil)
- ✧ PGS134-S16B: SOP16B (150mil)
- ✧ PGS134-S20: SOP20 (300mil)
- ✧ PGS134-H20: HTSOP20 (150mil)
- ✧ PGS134-S24: SOP24 (300mil)
- ✧ PGS134-Y24: SSOP24 (150mil)
- ✧ PGS134-4N10: DFN3*3-10P (0.5pitch)
- ✧ PGS134-2J16A: QFN4*4-16P (0.65pitch)
- ✧ PGS134-1J16A: QFN3*3-16P (0.5pitch)
- ✧ PGS134-2J24: QFN4*4-24P (0.5pitch)

- 封装尺寸信息请参考官网文件："封装信息"

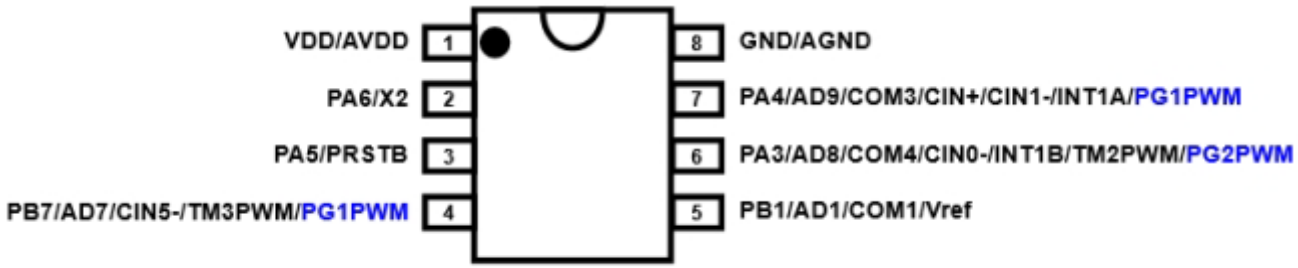
2. 系统概述和方框图

PGS134 系列是一款带 12bit ADC，以 MTP 为程序基础的 CMOS 8-bit 微处理器。它运用 RISC 的架构并且所有的指令架构的执行周期都是一个指令周期，只有少部分指令需要两个指令周期。

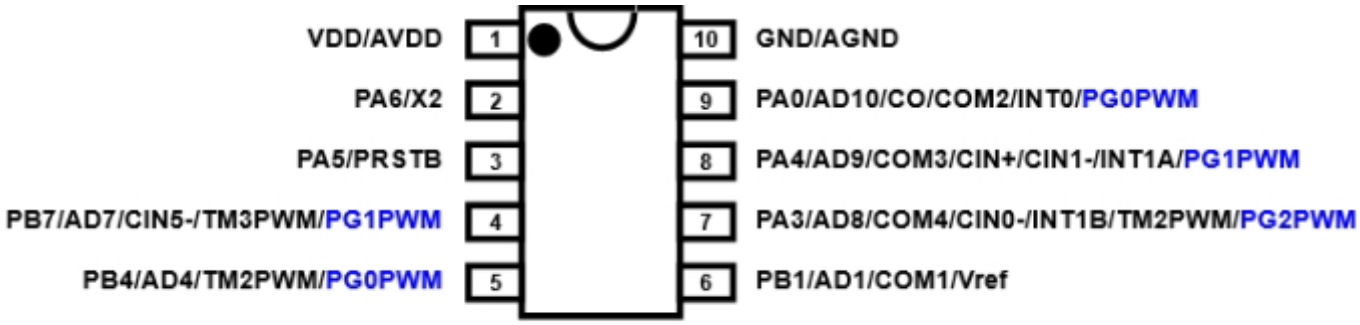
内部最多达 4KW MTP 程序存储器、512 字节 EEPROM 以及 256 字节数据存储器，还有多达 14 通道 12 位分辨率的 ADC，其中一个通道是带有多个内部参考电压可供选择。PGS134 同时提供 6 个硬件计数器：一个 16 位的硬件计数器，两个 8 位 PWM 计数器，和三个 11 位 PWM 计数器。另外，PGS134 还提供一个硬件比较器和驱动 LCD 的 1/2 V_{DD} 偏置电压。



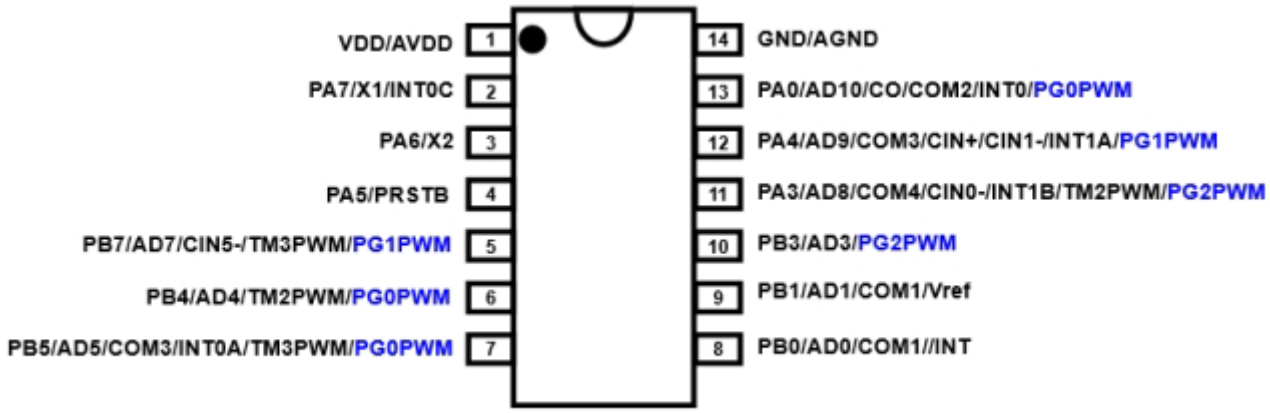
3. 引脚功能说明



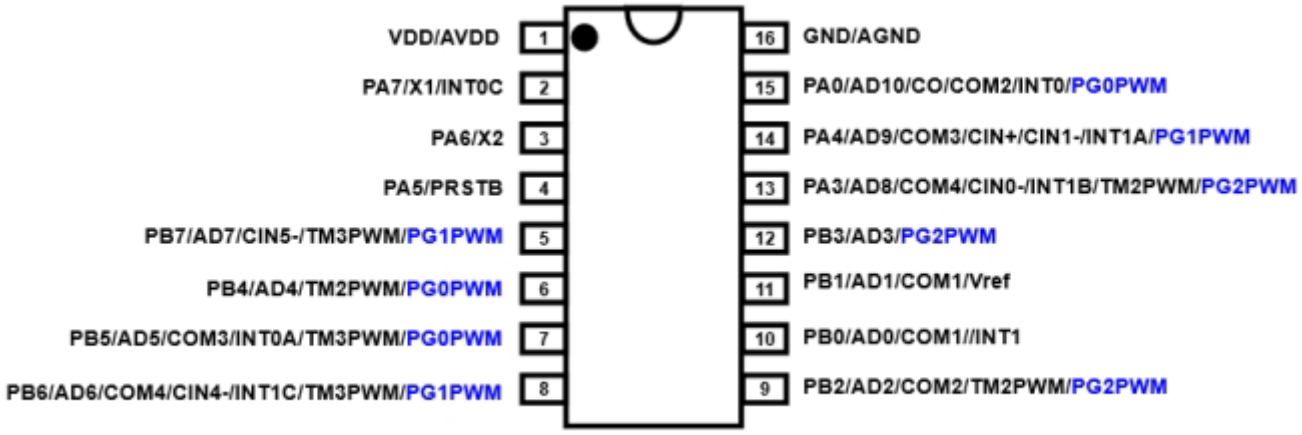
PGS134-S08 (SOP8-150mil)



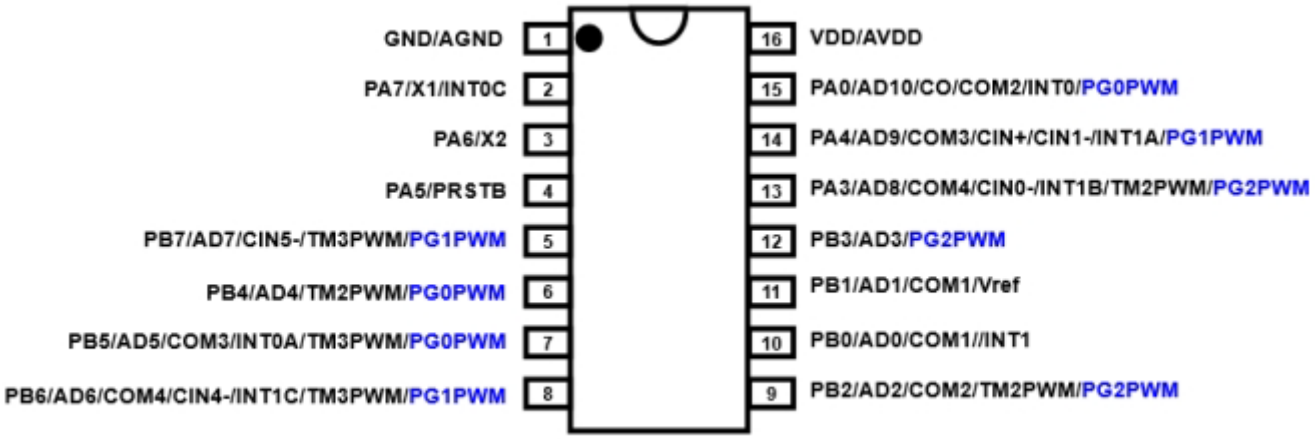
PGS134-M10 (MSOP10-118mil)



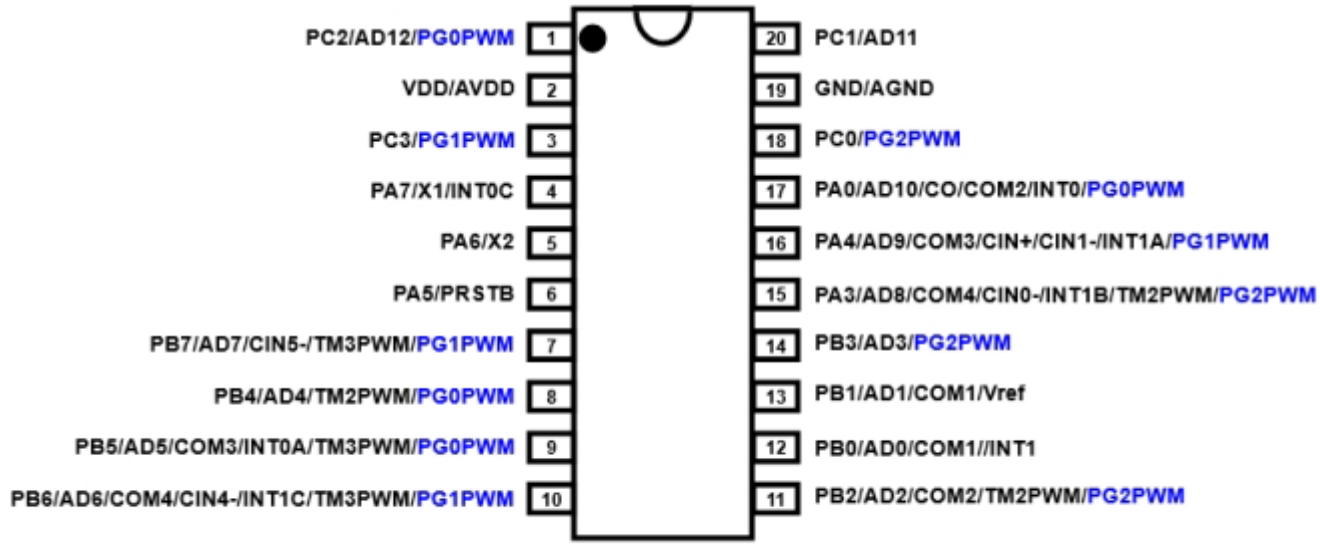
PGS134-S14 (SOP14-150mil)



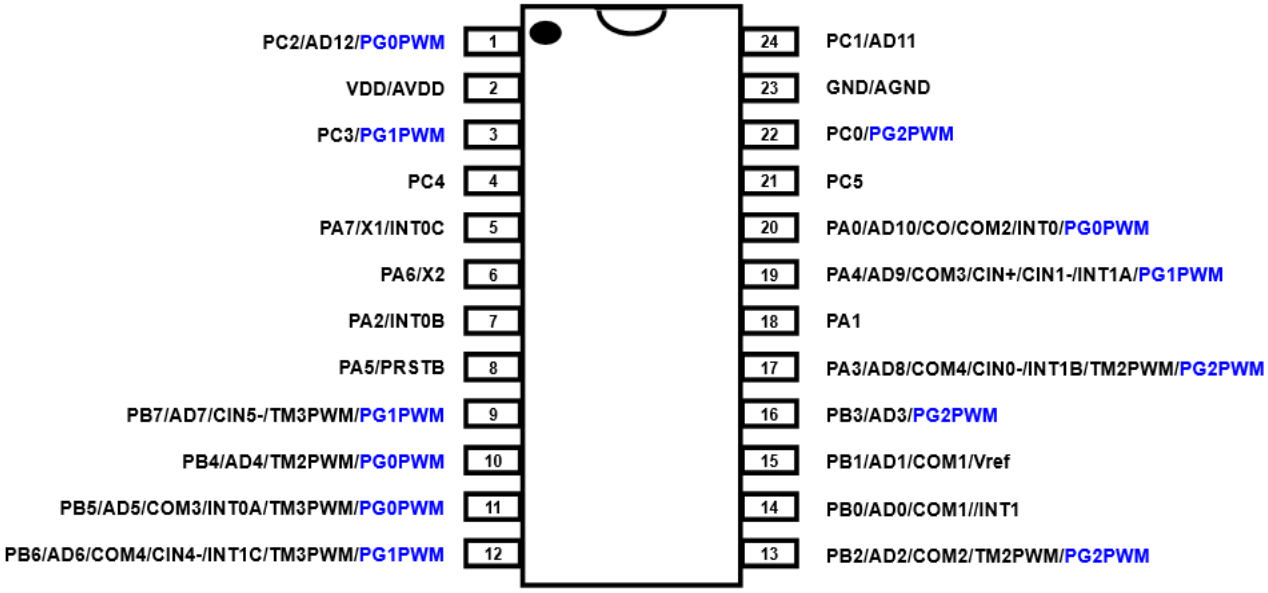
PGS134-S16A (SOP16A-150mil)



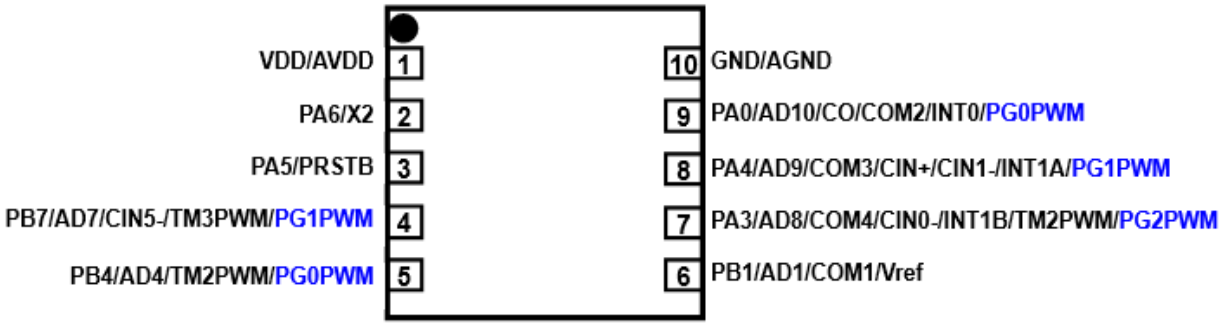
PGS134-S16B (SOP16B-150mil)



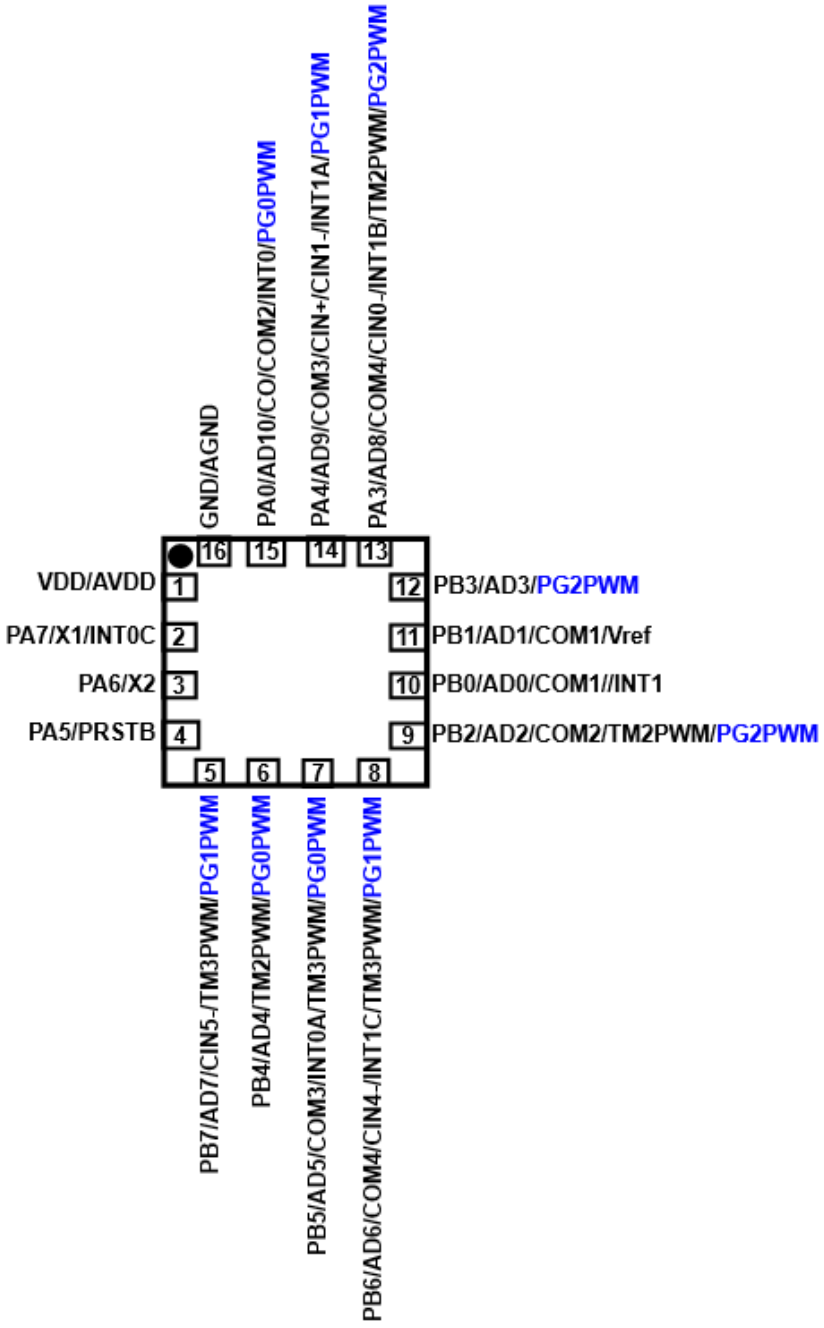
PGS134-S20 (SOP20-300mil)
PGS134-H20 (HTSOP20-150mil)



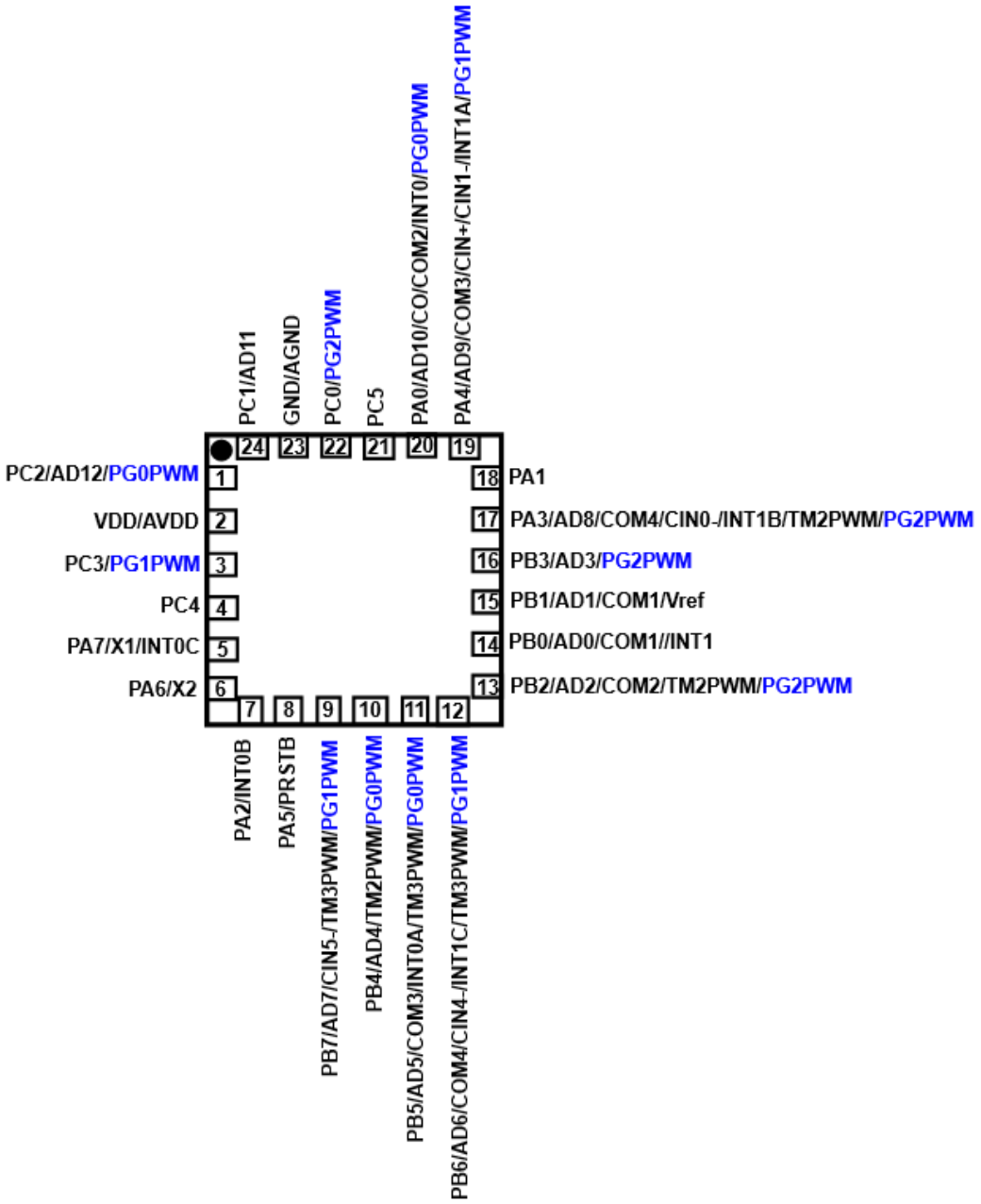
PGS134-S24 (SOP24-300mil)
PGS134-Y24 (SSOP24-150mil)



PGS134-4N10 (DFN3*3-10P-0.5pitch)



PGS134-2J16A (QFN4*4-16P-0.65pitch)
PGS134-1J16A (QFN3*3-16P-0.5pitch)



PGS134-2J24 (QFN4*4-24P-0.5pitch)

引脚名称	引脚类型	描述
PA7 / X1 / INT0C	IO ST / CMOS	此引脚可以用作： (1) 端口 A 位 7。并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) 当使用外部晶振时，作为 Xin(X1)引脚。 (3) INT0C。作为外部中断端口 0。通过寄存器可以设置上升沿和下降沿响应中断服务请求。 当用做晶体振荡器的功能时，为减少漏电流，请用 ***padier***.7 关闭其数字输入功能。这个引脚可以设定在睡眠中唤醒系统的功能，但是当 ***padier***.7 为”0”时，唤醒功能是被关闭的。
PA6 / X2	IO ST / CMOS	此引脚可以用作： (1) 端口 A 位 6，并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) 当使用外部晶振时，作为 Xout(X2)引脚。 当用做晶体振荡器的功能时，为减少漏电流，请用 ***padier***.6 关闭其数字输入功能. 这个引脚可以设定在睡眠中唤醒系统的功能，但是当 ***padier***.6 为”0”时，唤醒功能是被关闭的。
PA5 / PRSTB	IO ST / CMOS	此引脚可以用作： (1) 端口 A 位 5，并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) 硬件复位。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当 ***padier***.5 为”0”时，唤醒功能是被关闭的。
PA4 / AD9 / COM3 / CIN+ / CIN1- /INT1A / PG1PWM	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 A 位 4，此引脚可以设定为输入或输出，弱上拉/下拉电阻模式。 (2) ADC 模拟输入通道 9 。 (3) COM3 口，提供 1/2 V_{DD} 驱动 LCD 显示。 (4) 比较器的正输入源。 (5) 比较器的负输入源 1。 (6) INT1A。它可以用作外部中断源 1。通过寄存器可以设置上升沿和下降沿响应中断服务请求。 (7) 11 位 PWM 生成器 PWMG1 的输出端。 当用做模拟输入功能时，为减少漏电流，请用 ***padier***.4 关闭其数字输入功能。这个引脚可以设定在睡眠中唤醒系统的功能；但是，当 ***padier***.4 为”0”时，唤醒功能是被关闭的。
PA3 / AD8 / COM4 / CIN0- / INT1B / TM2PWM / PG2PWM	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 A 位 3，并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) ADC 模拟输入通道 8。 (3) COM4 口，提供 1/2 V_{DD} 驱动 LCD 显示。 (4) 比较器 0 的负输入。 (5) 外部中断源 1B。它可以用作外部中断源 1。通过寄存器可以设置上升沿和下降沿响应中断服务请求。 (6) Timer2 的 PWM 输出端。 (7) 11 位 PWM 生成器 PWMG2 的输出端。 当用做模拟输入功能时，为减少漏电流，请用 ***padier***.3 关闭其数字输入功能. 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当 ***padier***.3 为”0”时，唤醒功能是被关闭的。

引脚名称	引脚类型	描述
PA2 / INT0B	IO ST / CMOS	此引脚可以用作： (1) 端口 A 位 2，并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) INT0B。它可以用作外部中断源 0。通过寄存器可以设置上升沿和下降沿响应中断服务请求。 当用做模拟输入功能时，为减少漏电流，请用 ***padier***.2 关闭其数字输入功能。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当 ***padier***.2 为”0”时，唤醒功能是被关闭的。
PA1	IO ST / CMOS	此引脚可以用作： 端口 A 位 1，并可编程设定为输入或输出，弱上拉/下拉电阻模式。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当 ***padier***.1 为”0”时，唤醒功能是被关闭的。
PA0 / AD10 / COM2 / CO / INT0 / PG0PWM	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 A 位 0，并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) ADC 模拟输入通道 10。 (3) COM2 口，提供 1/2 V_{DD} 驱动 LCD 显示。 (4) 比较器的输出。 (5) INT0。它可以用作外部中断源 0。通过寄存器可以设置上升沿和下降沿响应中断服务请求。 (6) 11 位 PWM 生成器 PWMG0 的输出端。 当用做模拟输入功能时，为减少漏电流，请用 ***padier***.0 关闭其数字输入功能。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当 ***padier***.0 为”0”时，唤醒功能是被关闭的。
PB7 / AD7 / CIN5- / TM3PWM / PG1PWM	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 B 位 7，并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) ADC 模拟输入通道 7。 (3) 比较器的负输入源 5。 (4) Timer3 的 PWM 输出端。 (5) 11 位 PWM 生成器 PWMG1 的输出端。 当用做模拟输入功能时，为减少漏电流，请用 ***pbdier***.7 关闭其数字输入功能。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当 ***pbdier***.7 为”0”时，唤醒功能是被关闭的。

引脚名称	引脚类型	描述
PB6 / AD6 / COM4 / CIN4- / INT1C / TM3PWM / PG1PWM	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 B 位 6，并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) ADC 模拟输入通道 6。 (3) COM4 口，提供 1/2 V_{DD} 驱动 LCD 显示。 (4) 比较器的负输入源 4。 (5) INT1C。它可以用作外部中断源 1。通过寄存器可以设置上升沿和下降沿响应中断服务请求。 (6) Timer3 的 PWM 输出端。 (7) 11 位 PWM 生成器 PWMG1 的输出端。 当用做模拟输入功能时，为减少漏电流，请用 ***pbdier***.6 关闭其数字输入功能。这个引脚可以设定在睡眠中唤醒系统的功能；但是，当 ***pbdier***.6 为”0”时，唤醒功能是被关闭的。
PB5 / AD5 / COM3 / INT0A / TM3PWM / PG0PWM	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 B 位 5，并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) ADC 模拟输入通道 5。 (3) COM3 口，提供 1/2 V_{DD} 驱动 LCD 显示。 (4) INT0A。它可以用作外部中断源 0。通过寄存器可以设置上升沿和下降沿响应中断服务请求。 (4) Timer3 的 PWM 输出端。 (5) 11 位 PWM 生成器 PWMG0 的输出端。 当用做模拟输入功能时，为减少漏电流，请用 ***pbdier***.5 关闭其数字输入功能。这个引脚可以设定在睡眠中唤醒系统的功能；但是，当 ***pbdier***.5 为”0”时，唤醒功能是被关闭的。
PB4 / AD4 / TM2PWM / PG0PWM	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 B 位 4，并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) ADC 模拟输入通道 4。 (3) Timer2 的 PWM 输出。 (4) 11 位 PWM 生成器 PWMG0 的输出端。 当用做模拟输入功能时，为减少漏电流，请用 ***pbdier***.4 关闭其数字输入功能。这个引脚可以设定在睡眠中唤醒系统的功能；但是，当 ***pbdier***.4 为”0”时，唤醒功能是被关闭的。
PB3 / AD3 / PG2PWM	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 B 位 3，并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) ADC 模拟输入通道 3。 (3) 11 位 PWM 生成器 PWMG2 的输出端。 当用做模拟输入功能时，为减少漏电流，请用 ***pbdier***.3 关闭其数字输入功能。这个引脚可以设定在睡眠中唤醒系统的功能；但是，当 ***pbdier***.3 为”0”时，唤醒功能是被关闭的。

引脚名称	引脚类型	描述
PB2 / AD2 / COM2 / TM2PWM / PG2PWM	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 B 位 2，并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) ADC 模拟输入通道 2。 (3) COM2 口，提供 1/2 V_{DD} 驱动 LCD 显示。 (4) Timer2 的 PWM 输出。 (5) 11 位 PWM 生成器 PWMG2 的输出端。 当用做模拟输入功能时，为减少漏电流，请用 ***pbdier***.2 关闭其数字输入功能。这个引脚可以设定在睡眠中唤醒系统的功能；但是，当 ***pbdier***.2 为”0”时，唤醒功能是被关闭的。
PB1 / AD1 / COM1 / Vref	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 B 位 1，并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) ADC 模拟输入通道 1。 (3) COM1 口，提供 1/2 V_{DD} 驱动 LCD 显示。 (4) ADC 的外部参考高电压。 此引脚可用做端口 B 位 1，并可编程设定为输入或输出，弱上拉电阻模式。这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 ***pbdier*** 位 1 为”0”时，唤醒功能是被关闭的。
PB0 / AD0 / COM1 / INT1	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 B 位 0，并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) ADC 的模拟输入通道 0。 (3) COM1 口，提供 1/2 V_{DD} 驱动 LCD 显示。 (4) INT1。它可以用作外部中断源 1。通过寄存器可以设置上升沿和下降沿响应中断服务请求。 当用做模拟输入功能时，为减少漏电流，请用 ***pbdier***.0 关闭其数字输入功能。这个引脚可以设定在睡眠中唤醒系统的功能；但是，当 ***pbdier***.0 为”0”时，唤醒功能是被关闭的。
PC5	IO ST / CMOS	此引脚可以用作： 端口 C 位 5。并可编程设定为输入或输出，弱上拉/下拉电阻模式。 ***pcdier***.5 可以设为“0” 停用睡眠中唤醒系统的功能。
PC4	IO ST / CMOS	此引脚可以用作： 端口 C 位 4。并可编程设定为输入或输出，弱上拉/下拉电阻模式。 ***pcdier***.4 可以设为“0” 停用睡眠中唤醒系统的功能。
PC3 / PG1PWM	IO ST / CMOS	此引脚可以用作： (1) 端口 C 位 3。并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) 11 位 PWM 生成器 PWMG1 的输出端。 ***pcdier***.3 可以设为“0” 停用睡眠中唤醒系统的功能。

引脚名称	引脚类型	描述
PC2 / AD12 / PG0PWM	IO ST / CMOS / Analog	此引脚可以用作： (1) 端口 C 位 2。并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) ADC 的模拟输入通道 12。 (3) 11 位 PWM 生成器 PWMG0 的输出端。 当用做模拟输入功能时，为减少漏电流，请用 ***pcdier***.2 关闭其数字输入功能。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当 ***pcdier***.2 为”0”时，唤醒功能是被关闭的。
PC1 / AD11	IO ST / CMOS / Analog	此引脚可以用作： (4) 端口 C 位 1。并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (5) ADC 的模拟输入通道 11。 当用做模拟输入功能时，为减少漏电流，请用 ***pcdier***.1 关闭其数字输入功能。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当 ***pcdier***.1 为”0”时，唤醒功能是被关闭的。
PC0 / PG2PWM	IO ST / CMOS	此引脚可以用作： (1) 端口 C 位 0。并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) 11 位 PWM 生成器 PWMG2 的输出端。 ***pcdier***.0 可以设为“0” 停用睡眠中唤醒系统的功能。
VDD / AVDD	VDD / AVDD	VDD：数字正电源 AVDD：模拟正电源 VDD 是 IC 电源，而 AVDD 是 ADC 专用电源。在 IC 内部，AVDD 与 VDD 连在一起(double bonding)，而外部为相同引脚。
GND / AGND	GND / AGND	GND：数字负电源 AGND：模拟负电源 GND 是 IC 接地引脚，而 AGND 是 ADC 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。
注意 ：**IO**：输入/输出；**ST**：施密特触发器输入；**Analog**：模拟输入引脚；**CMOS**：CMOS 电压基准位		

4. 器件电器特性

4.1. 直流交流电气特性

下列所有数据除特别列明外，皆于 Ta = -40°C ~ 85°C, V_{DD}=5.0V, f_{SYS}=2MHz 之条件下获得。

符号	描述	最小值	典型值	最大值	单位	条件(Ta=25°C)
V_{DD}	工作电压	2.2#	5.0	5.5	V	# 受限于 LVR 公差
LVR%	低电压复位公差	-5		5	%	
f_{SYS}	系统时钟 (CLK)* = IHRC/2 IHRC/4 IHRC/8 ILRC	 0 0 0	 82K	 8M 4M 2M	Hz	 V_{DD} ≧ 3.5V V_{DD}≧ 2.5V V_{DD} ≧ 2.2V V_{DD} = 5.0V
V_{POR}	上电复位电压		2.0*			* 受限于 LVR 公差
I_{OP}	工作电流		0.6 80		mA uA	f_{SYS}=IHRC/16=1MIPS@5.0V f_{SYS}=ILRC=82KHz@3.3V
I_{PD}	掉电模式下电流 (使用 ***stopsys*** 命令)		0.9 0.5		uA uA	f_{SYS}= 0Hz, V_{DD} =5.0V f_{SYS}= 0Hz, V_{DD} =3.3V
I_{PS}	省电模式下电流 (使用 ***stopexe*** 命令)		4.0		uA	V_{DD} =5.0V; f_{SYS}= ILRC 只启用 ILRC 模块
V_{IL}	输入低电压	0		0.2 V_{DD}	V	
V_{IH}	输入高电压	0.7 V_{DD}		V_{DD}	V	
I_{OL}	IO 输出灌电流（可通过 Code Option “PB4_PB7_Drive”切换 PB4/PB7 电流）					
	PB4, PB7 (Normal) PB4, PB7 (Strong) PA0-4, PB2, PB5-6 PA5-7, PB0-1, PB3, PC0-5		24 40 25 15		mA	V_{DD}=5.0V, V_{OL}=0.5V
I_{OH}	IO 输出驱动电流					
	PB4, PB7 (Normal) PB4, PB7 (Strong) Other IOs		17 32 14		mA	V_{DD}=5.0V, V_{OH}=4.5V
V_{IN}	输入电压	-0.3		V_{DD} +0.3	V	
$I_{INJ\ (PIN)}$	引脚输入电流			1	mA	V_{DD} +0.3≧V_{IN}≧ -0.3
R_{PH}	上拉电阻		62		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}	校准后 IHRC 频率 *	15.76*	16*	16.24*	MHz	25°C, V_{DD} =2.2V~5.5V
		15.20*	16*	16.80*		V_{DD} =2.2V~5.5V, 0°C <Ta<70°C*
t_{INT}	中断脉冲宽度	30			ns	V_{DD} = 5.0V

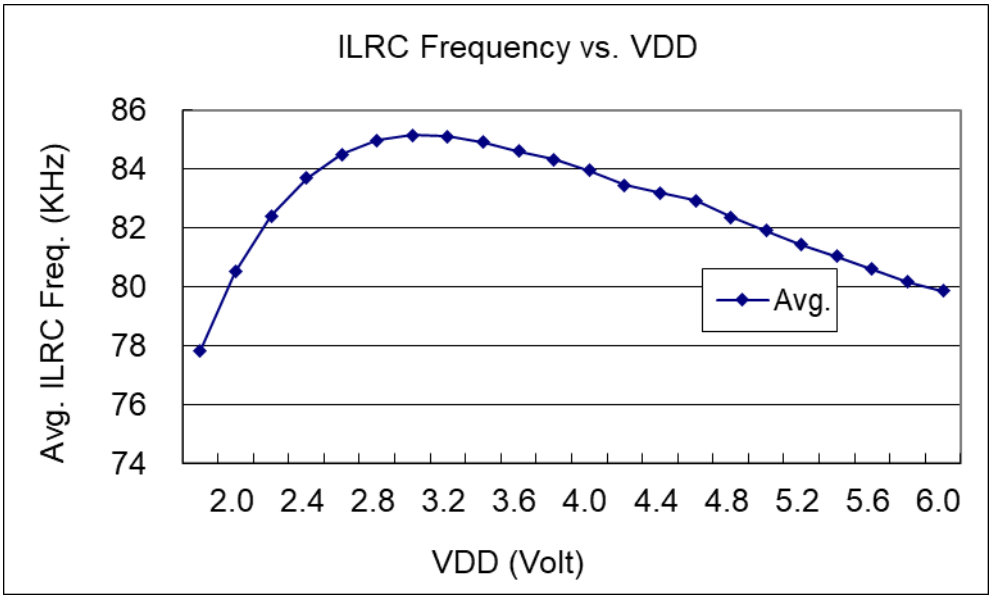
符号	描述	最小值	典型值	最大值	单位	条件(Ta=25°C)
V_{ADC}	ADC 可工作电压	2.2		V_{DD}	V	
V_{AD}	AD 输入电压	0		V_{DD}	V	
ADrs	ADC 分辨率		12		bit	
ADcs	ADC 消耗电流		1.1 1		mA	@5V @3V
ADclk	ADC 时钟周期		2		us	2.2V ~ 5.5V
t_{ADCONV}	ADC 转换时间 (T_{ADCLK}是选定AD转换时钟周期)		16		T_{ADCLK}	12-位分辨率
AD DNL	ADC 微分非线性		±2*		LSB	
AD INL	ADC 积分非线性		±4*		LSB	
ADos	ADC 失调电压		2*		mV	@ V_{DD} =3V
V_{REFH}	ADC 参考高电压 4V 3V 2V	 3.90* 2.93* 1.95*	 4* 3* 2*	 4.10* 3.07* 2.05*	V	@ V_{DD} =5V, 25 °C
V_{DR}	数据存储器数据保存电压*	1.5			V	待机模式下
t_{WDT}	看门狗超时溢出时间		8k		T_{ILRC}	misc[1:0]=00（默认）
			16k			misc[1:0]=01
			64k			misc[1:0]=10
			256k			misc[1:0]=11
t_{WUP}	快速唤醒时间		45		T_{ILRC}	T_{ILRC} 是 ILRC 时钟周期
	正常唤醒时间		3000			
t_{SBP}	系统开机时间（正常）		28		ms	V_{DD} =5V
	系统开机时间（快速）		620		us	V_{DD} =5V
t_{RST}	外部复位脉冲宽度	120			us	@ V_{DD} =5V
CPos	比较器偏置电压*	-	±10	±20	mV	
CPcm	比较器共模输入*	0		V_{DD}-1.5	V	
CPspt	比较器响应时间*		100	500	ns	上升沿和下降沿相同
CPmc	比较器模式改变的稳定时间		2.5	7.5	us	
CPcs	比较器电流消耗		20		uA	V_{DD} = 3.3V

* 这些参数是设计参考值，并不是每个芯片测试。

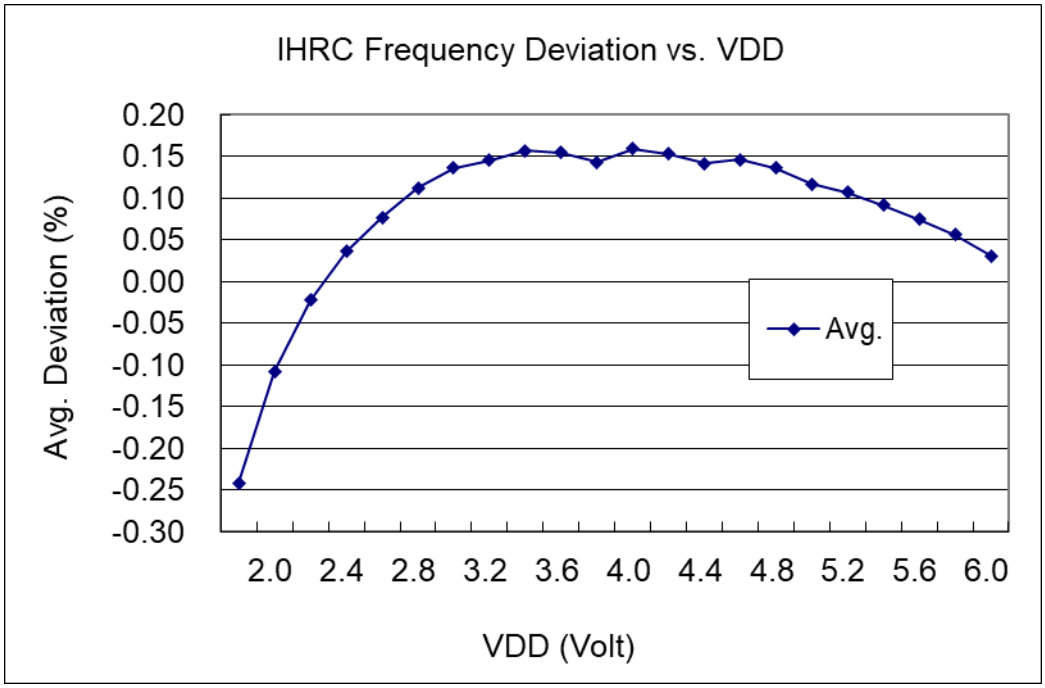
4.2. 绝对最大值范围

- 电源电压..... 2.2V ~ 5.5V
  * 最大电压不能超过 5.5v，否则可能永久性的损坏 IC。
- 输入电压..... -0.3V ~ V_{DD} + 0.3V
- 工作温度..... -40°C ~ 85°C
- 节点温度..... 150°C
- 存储温度..... -50°C ~ 125°C

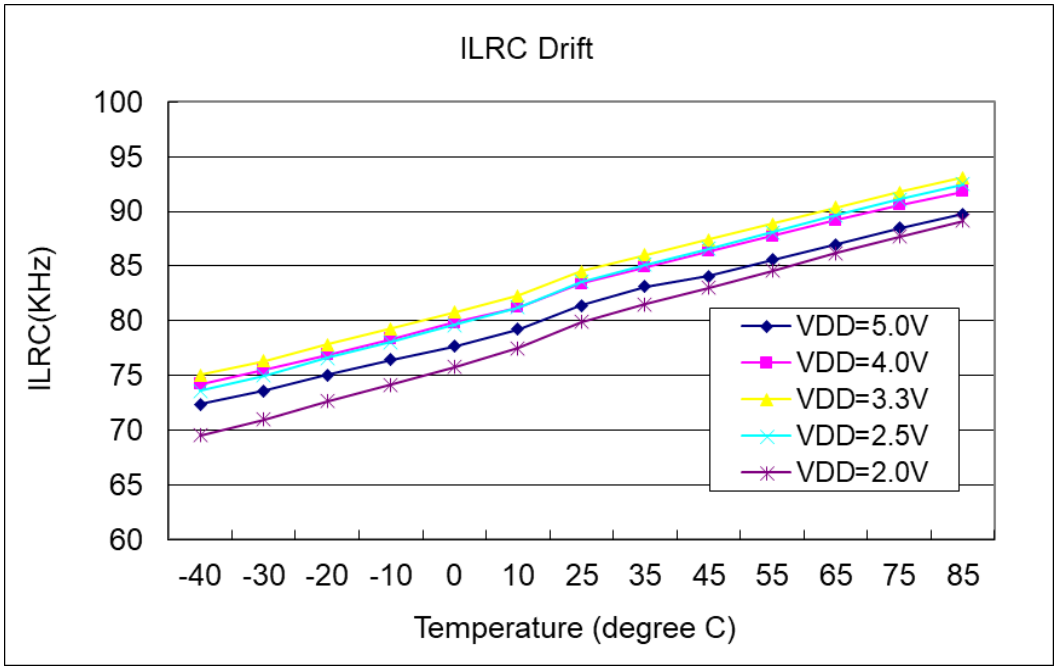
4.3. ILRC 频率与 VDD 关系曲线图



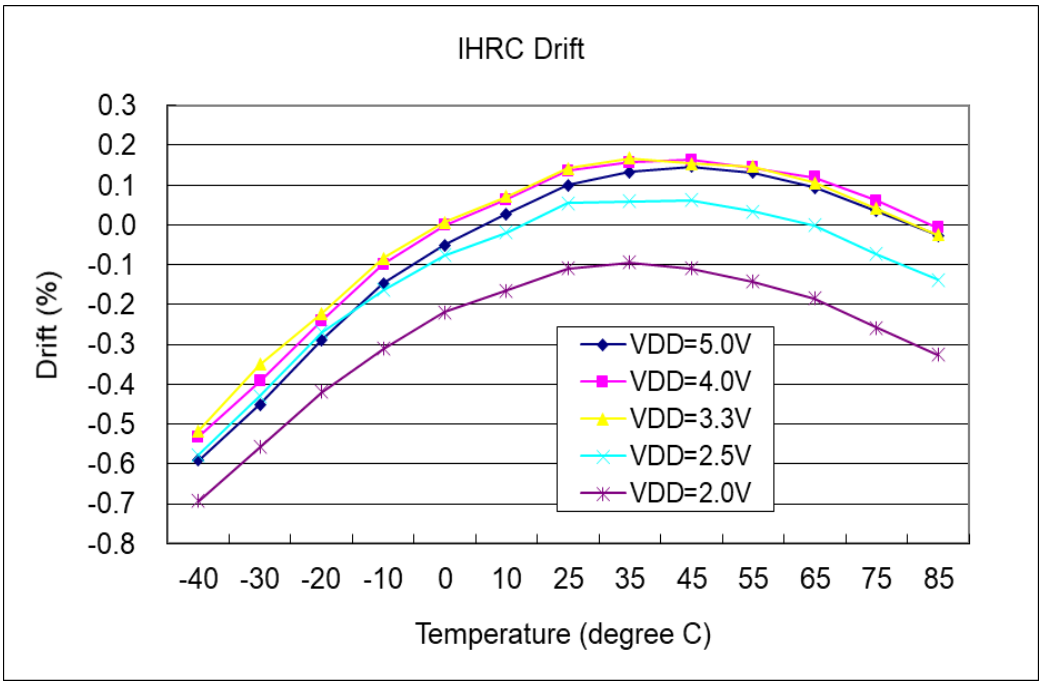
4.4. IHRC 频率与 VDD 关系曲线图（校准到 16MHz）



4.5. ILRC 频率与温度关系曲线图



4.6. IHRC 频率与温度关系曲线图（校准到 16MHz）

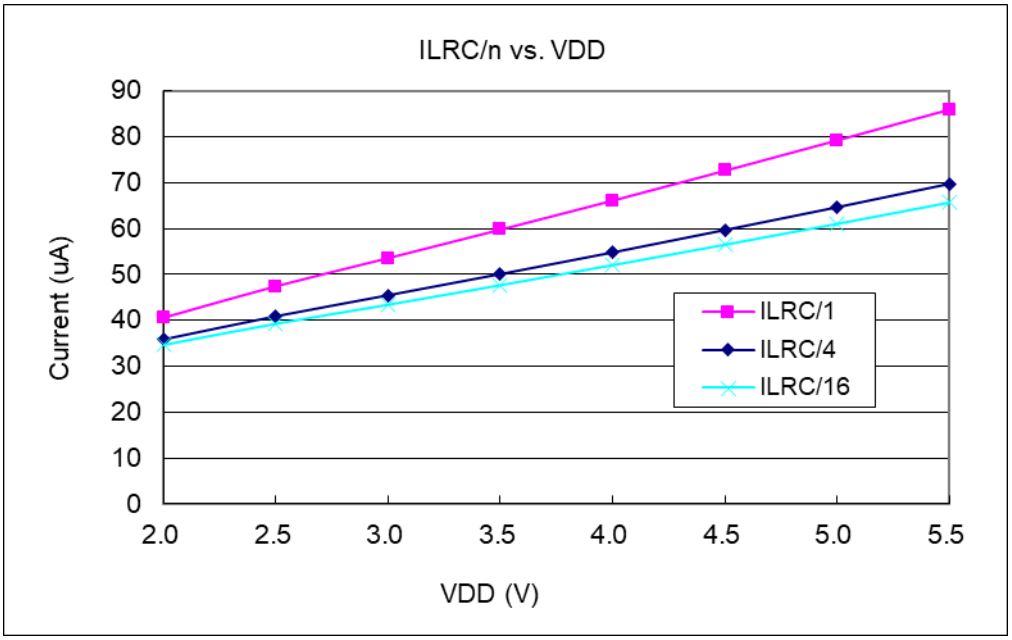


4.7. 工作电流 vs. VDD 与系统时钟 = ILRC/n 关系曲线图

测量条件：

启用：ILRC, Bandgap, LVR；**停用**：IHRC, EOSC, T16, TM2, TM3, ADC 等模块；

IO 引脚：PA0 以 0.5Hz 频率高低电压切换输出且无负载；**其他脚位**：设为输入且不浮空。

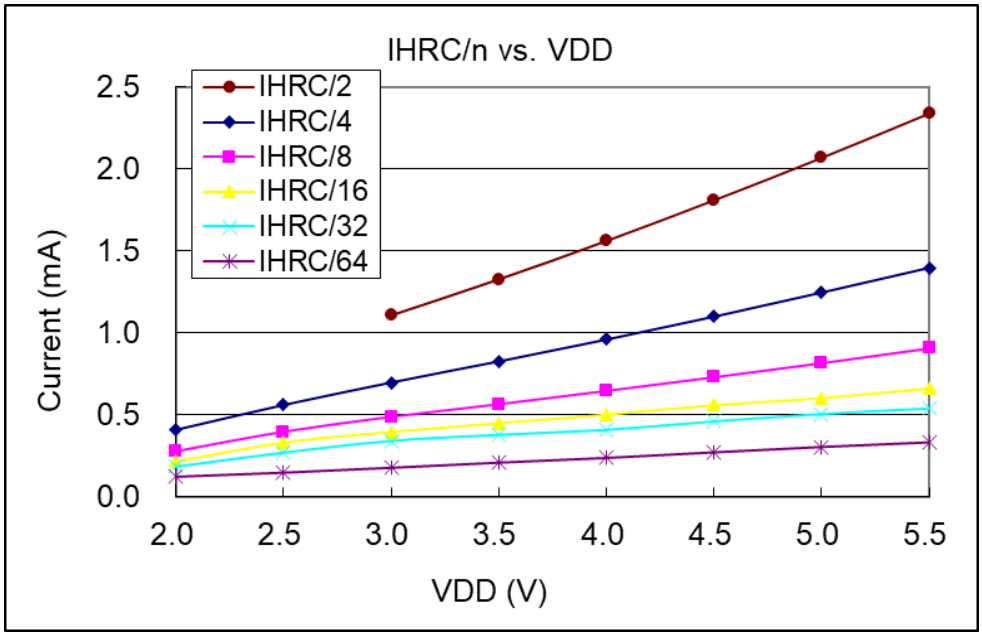


4.8. 工作电流 vs. VDD 与系统时钟 = IHRC/n 关系曲线图

测量条件：

启用：Bandgap, LVR, IHRC；**停用**：ILRC, EOSC, T16, TM2, TM3, ADC 等模块；

IO 引脚：PA0 以 0.5Hz 频率高低电压切换输出且无负载；**其他脚位**：设为输入且不浮空。

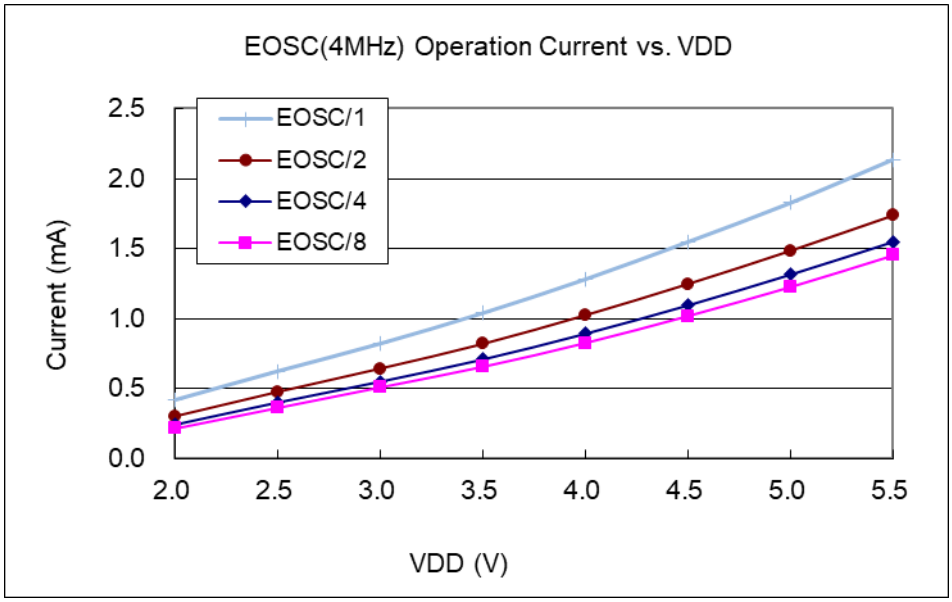


4.9. 工作电流 vs. VDD 与系统时钟 = 4MHz EOSC / n 关系曲线图

测试条件：

启用：EOSC[6,5] = [1,1], Bandgap, LVR；**停用**：IHRC, ILRC, T16, TM2, TM3, ADC 等模块；

IO 引脚：PA0 以 0.5Hz 频率高低电压切换输出且无负载，**其他脚位**：设为输入且不浮空。

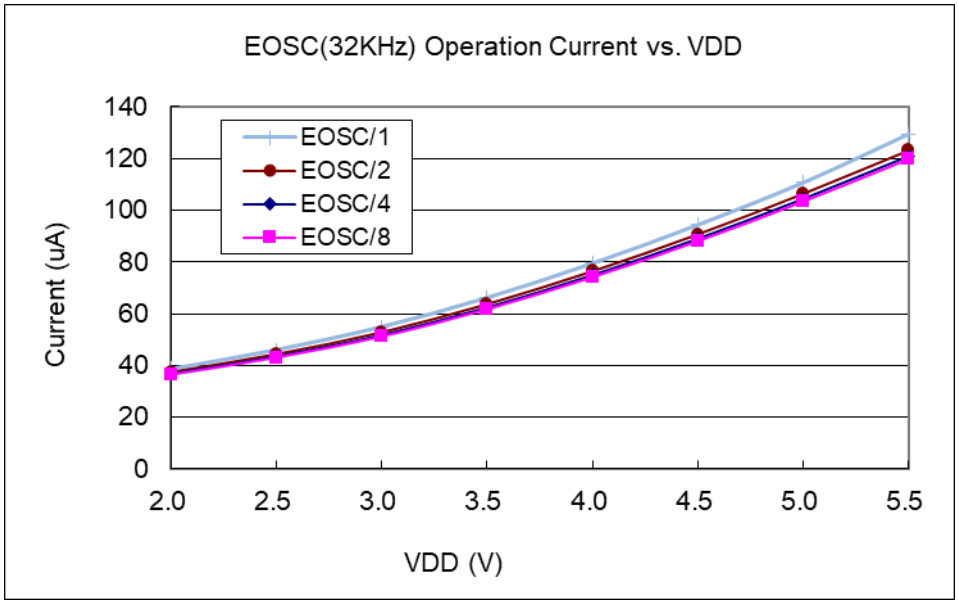


4.10. 工作电流 vs. VDD 与系统时钟 = 32KHz EOSC / n 关系曲线图

测试条件：

启用：EOSC[6,5] = [0,1], Bandgap, LVR；**停用**：IHRC, ILRC, T16, TM2, TM3, ADC 等模块；

IO 引脚：PA0 以 0.5Hz 频率高低电压切换输出且无负载；**其他脚位**：设为输入且不浮空。



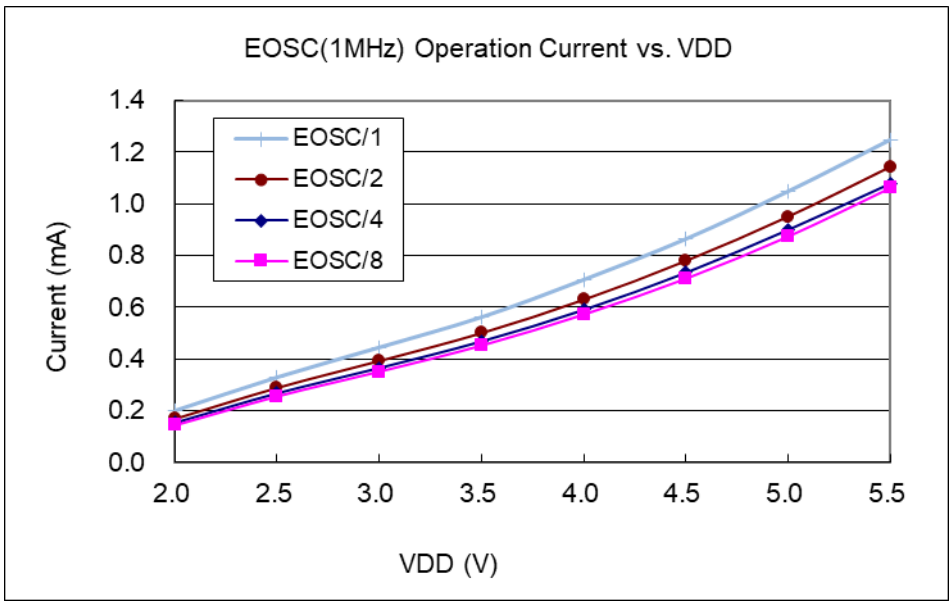
4.11. 工作电流 vs.VDD 与系统时钟 = 1MHz EOSC / n 关系曲线图

测试条件：

启用：EOSC[6,5] = [1,0], Bandgap, LVR；

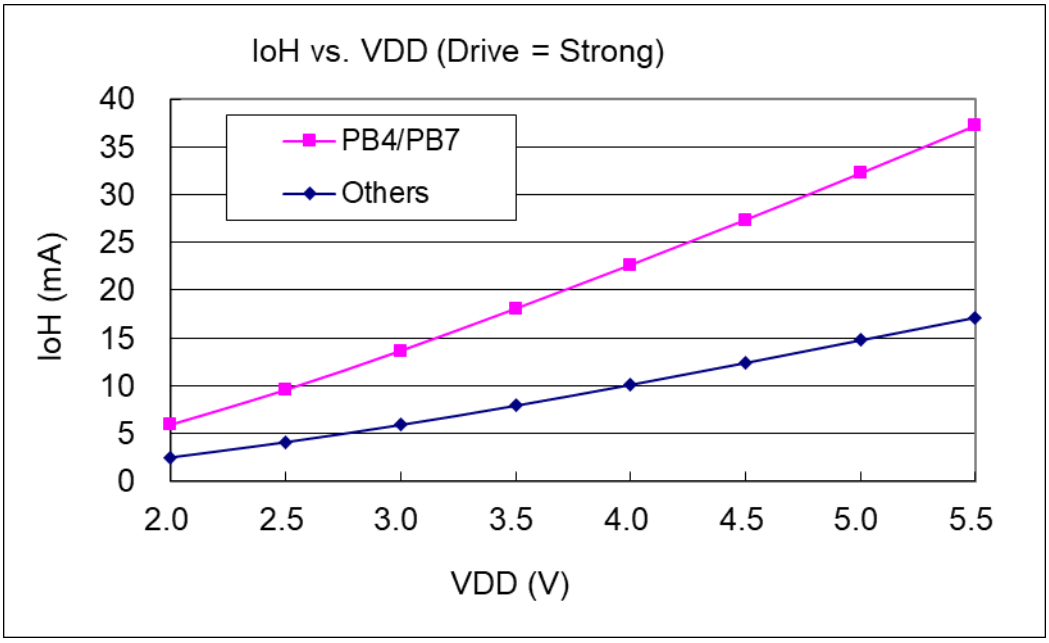
停用：IHRC, ILRC, T16, TM2, TM3, ADC 等模块；

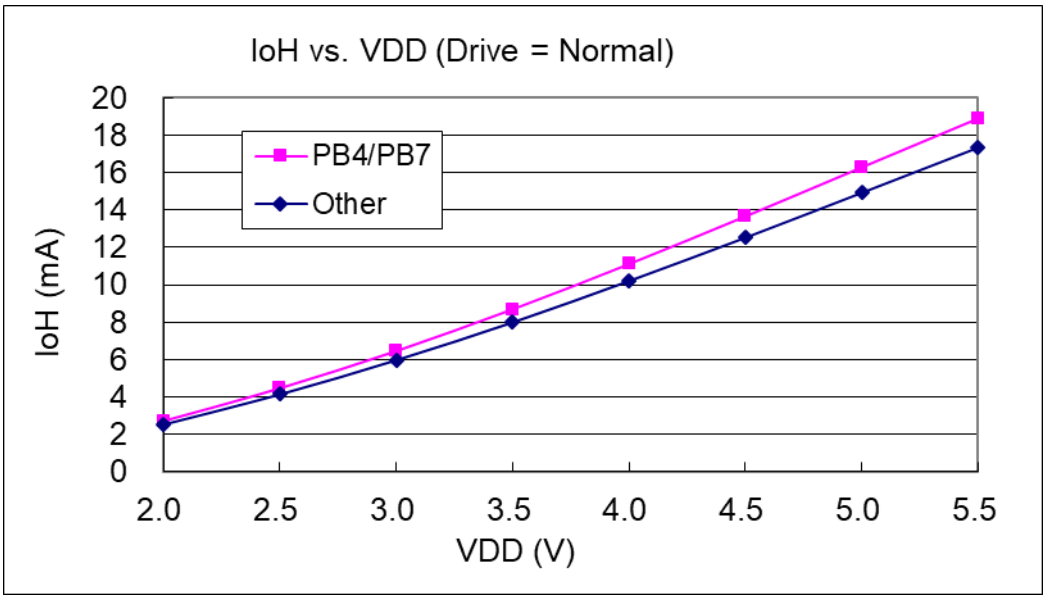
IO 引脚：PA0 以 0.5Hz 频率高低电压切换输出且无负载；**其他脚位**：设为输入且不浮空。

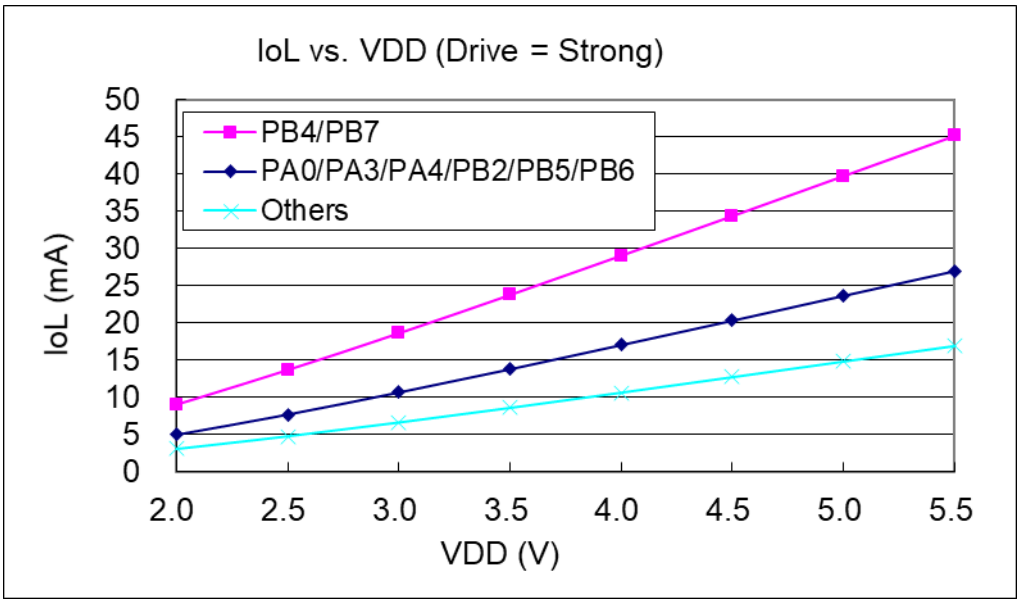


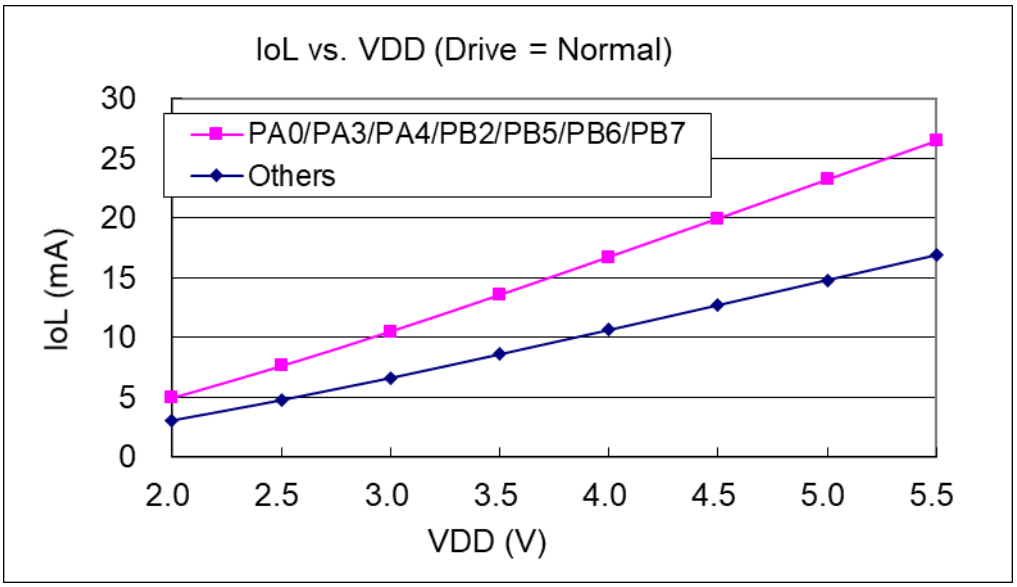
4.12. IO 引脚输出的驱动电流(I_{OH})与灌电流(I_{OL})曲线图

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

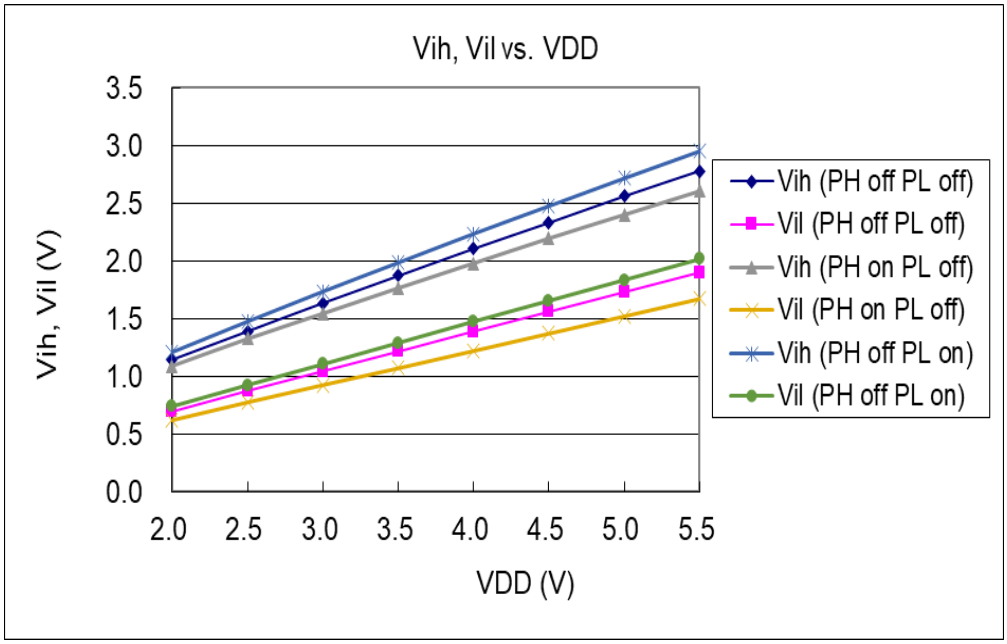




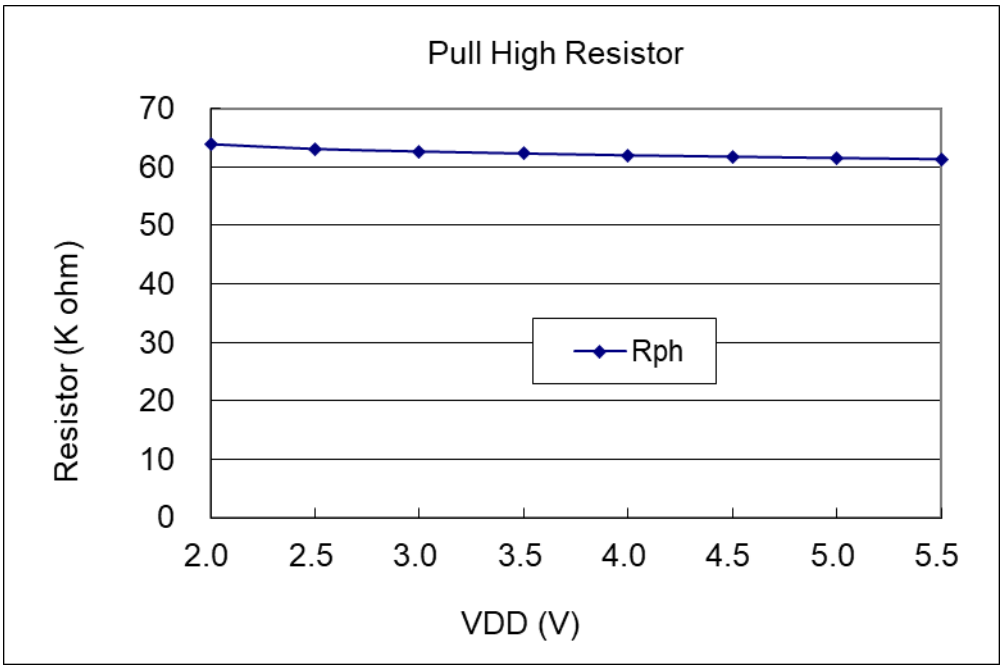

IoL vs. VDD (Drive = Strong)
IoL (mA)
50
45
40
35
30
25
20
15
10
5
0
PB4/PB7
PA0/PA3/PA4/PB2/PB5/PB6
Others
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
VDD (V)


IoL vs. VDD (Drive = Normal)
IoL (mA)
30
25
20
15
10
5
0
PA0/PA3/PA4/PB2/PB5/PB6/PB7
Others
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
VDD (V)

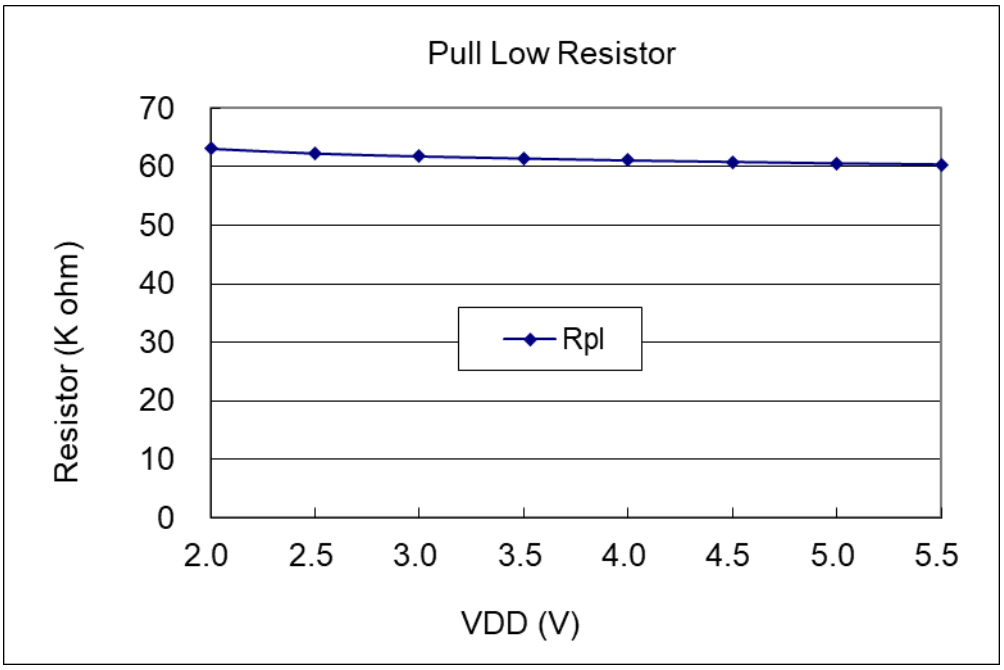
4.13. IO 引脚输入高/低阈值电压(V_{IH}/V_{IL})曲线图



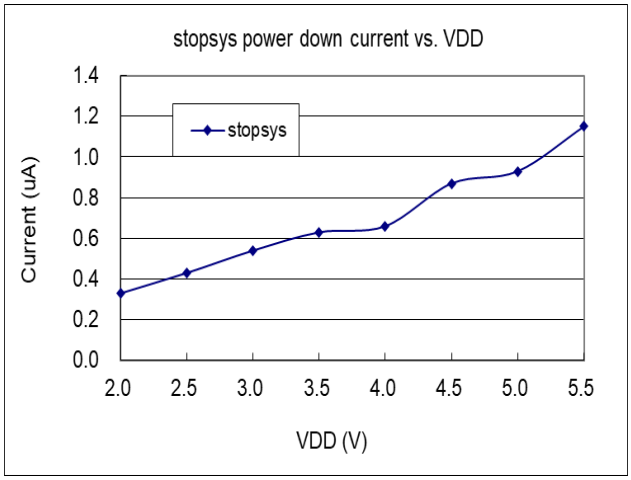
4.14. IO 引脚上拉阻抗曲线图

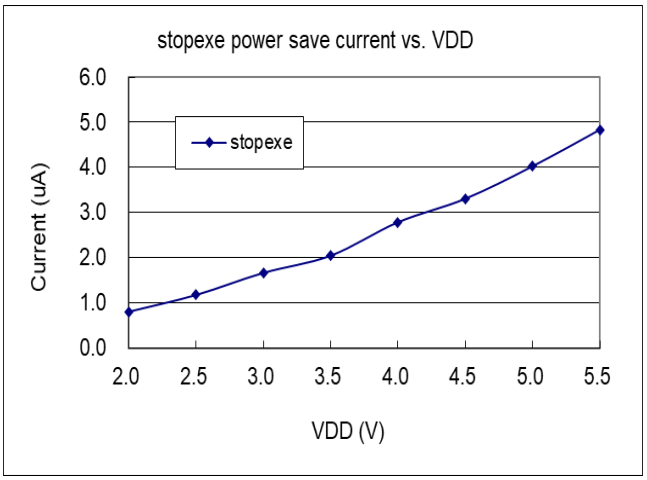


4.15. IO 引脚下拉阻抗曲线图



4.16. 掉电电流(I_{PD}) /省电电流 (I_{PS}).vs VDD 关系曲线图





5. 功能概述

5.1. MTP 程序存储器

MTP（可多次编程）程序存储器用来存放要执行的程序指令。MTP 程序存储器可以储存数据，包含：数据，表格和中断入口。复位之后，程序从 FPP0 的初始地址 0x000 开始，执行 GOTO FPPA0 语句。中断入口是 0x010。MTP 程序存储器最后 8 个地址空间是被保留给系统使用，如：校验码，序列号等。 PGS134 的 MTP 程序存储器容量为 4KW，如表 1 所示。MTP 存储器从地址 0xFF0 ~0xFFF 供系统使用，从 0x001 ~ 0x00F 和 0x011~0xFE7 地址空间是用户的程序空间。

地址	功能
0x000	GOTO FPPA0 指令
0x001	用户程序区
•	•
0x00F	用户程序区
0x010	中断入口地址
0x011	用户程序区
•	•
0xFEF	用户程序区
0XFF0	系统使用
•	•
0xFFF	系统使用

表 1：程序存储器结构

5.2. 启动程序

开机时，POR（上电复位）是用于复位 PGS134。开机时间可以通过选项设置为正常开机或者快速开机，快速开机时间为 45 个 ILRC 时钟周期，正常开机时间为 3000 个 ILRC，用户在使用时，无论选择哪种开机方式，都必须确保上电后电源电压稳定，开机时序如图 1 所示，其中 t_{SBP} 是开机时间。

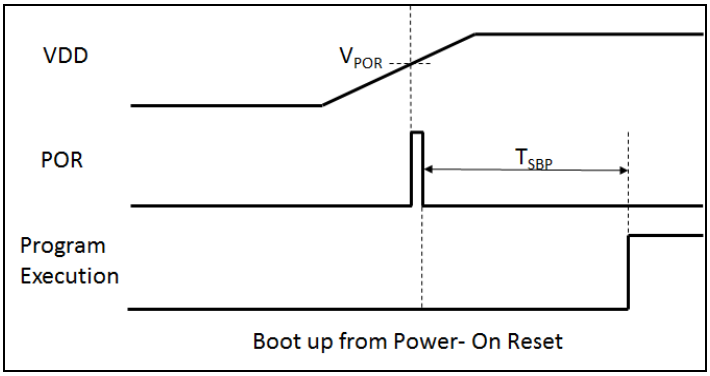
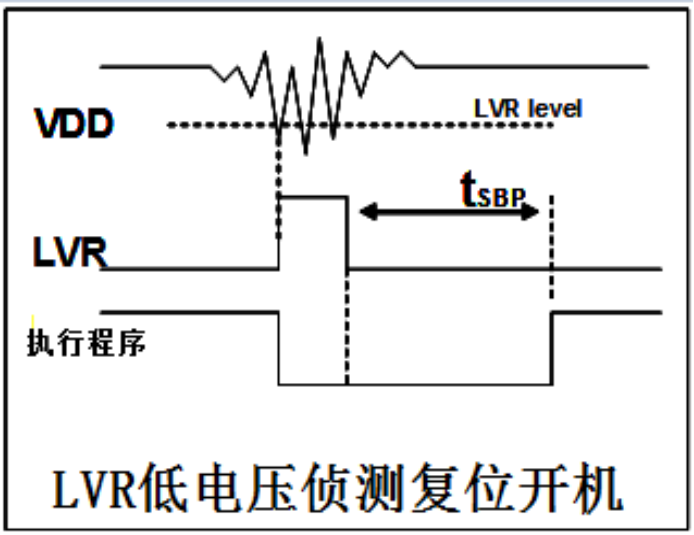
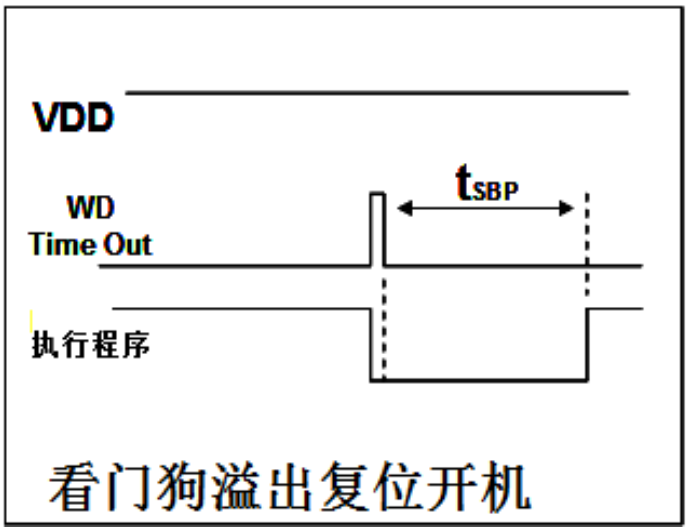
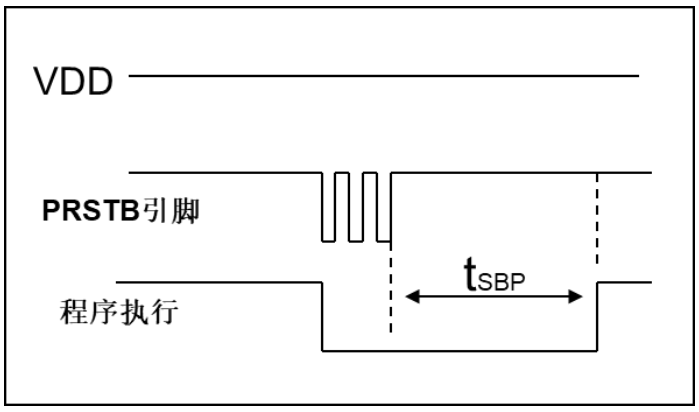


图 1：上电时序

5.2.1.复位时序图


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


VDD
WD
Time Out
tSBP
执行程序
看门狗溢出复位开机


VDD
PRSTB引脚
tSBP
程序执行

5.3. 数据存储器 – SRAM

数据存储可以是字节或位操作。除了存储数据外，数据存储器还可以担任间接存取方式的数据指针，以及堆栈存储器。

堆栈定义在数据存储器里面，堆栈指针定义在堆栈指针寄存器，用户可在使用时自行定义堆栈深度，堆栈存储器对堆栈的排列是非常灵活的，用户可以动态调整堆栈。

对于间接存储指令而言，数据存储器可以用作数据指针来当作数据地址。所有的数据存储器都可以当作资料指针，这对于间接存储指令是相当灵活和有效的。由于数据宽度是 8 位，PMS133/134 的所有 256 字节的数据存储器都可以利用间接存取指令有存取。

5.4. 数据存储器 – EEPROM

EEPROM 同样应用与用户数据存储。其与 SRAM 不同之处在于，EEPROM 为非易失性存储器，数据使用 EEPROM 存储后，即使是在电源关闭后仍旧能够被读出。EEPROM 只能通过间接寻址指令 STEER 和 LDEER 来访问。

例 1：EEPROM 字节写

(1) 定义 EEPROM 寻址变量

```
word    E2Adr_Index =    0;
```

(2) *EERMC* 写入 0x5A 使能 Byte Write

```
EERMC =    0x5A;
```

(3) *EERL* 赋值要写入的数据

```
EERL    =    10;
```

(4) 使用 *STEER* 指令往 EEPROM（地址：E2Adr_Index (0) ）里写入数据

```
STEER       E2Adr_Index;
```

(5) 等待写入完成

```
while(EERMC.Busy)   NOP;
```

注意：寻址变量 E2Adr_Index 的值自行修改，以实现往 EEPROM 不同 Byte 地址内写入数据

例 2：EEPROM 字节读

(1) 定义 EEPROM 寻址变量

```
word    E2Adr_Index =    0;
```

(2) *EERMC* 写入 0x5A 使能 Byte Read

```
EERMC =    0x5A;
```

(3) 使用 *LDEER* 指令从 EEPROM（地址：E2Adr_Index (0) ）里读出数据

```
LDEER       E2Adr_Index;
```

(4) 等待读出完成后，可使用读出的数据

```
while(EERMC.Busy)   NOP;
```

注意：读出的数据保存在 ***eerl*** 寄存器里

例 3：EEPROM 全擦除

(1) 定义 EEPROM 寻址变量

word E2Adr_Index = 0;

(2) *EERMC* 写入 0xA5 使能 All Page Erase

EERMC = 0xA5;

(3) 使用 *LDEER* 指令擦除 EEPROM 里所有数据

LDEER E2Adr_Index;

(4) 等待擦除完成

while(EERMC.Busy) NOP;

注意：1. All Page Erase 会擦除 EEPROM 所有的数据，无法分页擦除。

2. 数据擦除后，EEPROM 里的所有数据为 0xFF。

5.5. 振荡器和时钟

PGS134 有 3 个振荡器电路：外部晶体振荡器(EOSC)，内部高频 RC 振荡器(IHRC) 和内部低频振荡器(ILRC)，这 3 个振荡器可以分别通过寄存器 eoscr.7, clkmd.4 和 clkmd.2 来启用或停用。使用者可以选择不同的振荡器作为系统时钟源，同时可以通过设置 ***clkmd*** 寄存器来满足不同的应用要求。

振荡器模块	启用/停用
EOSC	**eoscr**.7
IHRC	**clkmd**.4
ILRC	**clkmd**.2

表 2：3 个振荡器电路

5.5.1. 内部高频 RC 振荡器和内部低频 RC 振荡器

开机后，IHRC 和 ILRC 振荡器是自动启用的。IHRC 频率能通过 ***ihrcr*** 寄存器校准，通常校准到 16MHz。校准后的频率偏差通常在 1%以内，然而，IHRC 频率会因为电源电压和工作温度产生漂移，详细请参考 IHRC 与 VDD 及温度关系曲线图。

ILRC 的频率会因生产工艺，使用的电源电压和温度的差异而产生漂移，请参考直流电气特性规格数据，建议不要应用在要求精准时序的产品上。

5.5.2. 芯片校准

在芯片生产制造时，IHRC 频率和 bandgap 参考电压都有可能稍微不同，PGS134 提供 IHRC 频率校准来消除这些差异，校准功能可以被用户的程序选择并编译，同时这个命令会自动嵌入用户的程序里面，校准命令如下所示：

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

Where, **p1**=2, 4, 8, 16, 32; 用以提供不同的系统时钟。

p2=14 ~ 18; 用以校准芯片到不同的频率，16MHz 是通用的选择。

p3=2.5 ~ 5.5; 用以在不同的工作电压下校准频率。

5.5.3. IHRC 频率校准和系统时钟

在用户编译程序时，IHRC 频率校准和系统时钟的选项如表 3 所示：

SYSCLK	CLKMD	IHRCR	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	不改变	没改变	IHRC 不校准, CLK 不改变

表 3： IHRC 频率校准选项

通常，.ADJUST_IC 是开机后第一条指令，以便系统开机后能设定系统频，IHRC 频率校准仅在烧录 MTP 程序代码的时候执行一次，烧录之后不会重复执行了。如果用户选择了不同的频率校准选项，PGS134 的系统状态在开机后也会不同。以下所示为不同的选项开机后，PGS134 执行此命令后的状态：

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

开机后，CLKMD = 0x34：

- IHRC 频率在 V_{DD}=5V 时校准到 16MHz，并且 IHRC 模块是启用的
- 系统时钟= IHRC/2 = 8MHz
- 看门狗计数器停用，ILRC 启用，PA5 引脚是输入模式

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

开机后，CLKMD = 0x14：

- IHRC 频率在 V_{DD}=3.3V 时校准到 16MHz，并且 IHRC 模块是启用的
- 系统时钟= IHRC/4 = 4MHz
- 看门狗计数器停用，ILRC 启用，PA5 引脚是输入模式

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

开机后, CLKMD = 0x3C：

- IHRC 频率在 V_{DD}=2.5V 时校准到 16MHz，并且 IHRC 模块是启用的
- 系统时钟= IHRC/8 = 2MHz
- 看门狗计数器停用，ILRC 启用，PA5 引脚是输入模式

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

开机后，CLKMD = 0x1C：

- IHRC 频率在 V_{DD}=2.5V 时校准到 16MHz，并且 IHRC 模块是启用的
- 系统时钟= IHRC/16 = 1MHz
- 看门狗计数器停用，ILRC 启用，PA5 引脚是输入模式

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

开机后，CLKMD = 0x7C：

- IHRC 频率在 V_{DD}=5V 时校准到 16MHz，并且 IHRC 模块是启用的
- 系统时钟= IHRC/32 = 500kHz
- 看门狗计数器停用，ILRC 启用，PA5 引脚是输入模式

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

开机后，CLKMD = 0XE4：

- IHRC 频率在 V_{DD}=5V 时校准到 16MHz，并且 IHRC 模块是停用的
- 系统时钟 = ILRC
- 看门狗计数器停用，ILRC 启用，PA5 引脚是输入模式

(7) .ADJUST_IC　　DISABLE

开机后， CLKMD 寄存器没有改变（没任何动作）：

- IHRC 没有校准并且 IHRC 模块是停用的
- 系统频率= ILRC 或 IHRC/64
- 看门狗计数器启用，ILRC 启用，PA5 引脚是输入模式

5.5.4. 外部晶体振荡器

如果使用晶体振荡器，X1 和 X2 之间需要晶体和谐振器，图 2 所示为硬件连接应用线路，晶体振荡器的工作频率范围可以从 1MHz 到 4MHz，超过 4MHz 则不支持。

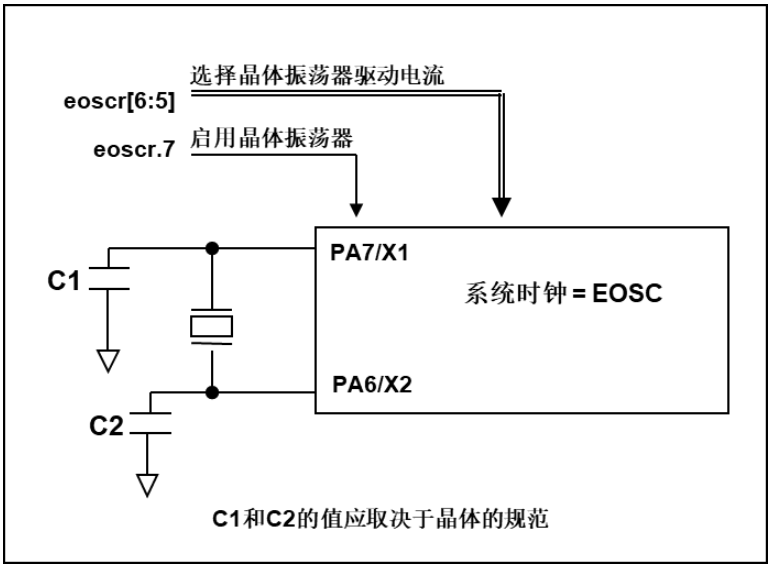


图 2：晶体振荡器使用接法

为了得到更好的正弦波形，除了选用更好的晶体，外部谐振电容 C1 和 C2 需要做电容值调整，同时，PGS134 的寄存器 *eoscr* (0x0a) 也需要做参数匹配。寄存器 *eoscr*.位 7 用来启用晶体振荡器，寄存器 *eoscr*.位 6 和寄存器 *eoscr*.位 5 用来提供不同的驱动电流来满足不同的晶体振荡器频率的要求：

- *eoscr*.[6:5]=01：低驱动能力，适用于较低频率，例如：32KHz 晶体振荡器。
- *eoscr*.[6:5]=10：中驱动能力，适用于中间频率，例如：1MHz 晶体振荡器。
- *eoscr*.[6:5]=11：高驱动能力，适用于较高频率，例如：4MHz 晶体振荡器。

表 4 所示针对不同的晶体振荡器推荐的 C1 和 C2 电容值，以及在对应条件下所测试到的起振时间。因为晶体或谐振器都有不同的特性，所需要的 C1，C2 值和起振时间会因为不同的晶体或起振器而有所差异，请参考其规格并选择恰当的 C1 和 C2 电容值。

频率	C1	C2	测量起振时间	条件
4MHz	4.7pF	4.7pF	6ms	(*eoscr*[6:5]=11, *misc*.6=0)
1MHz	10pF	10pF	11ms	(*eoscr*[6:5]=10, *misc*.6=0)
32KHz	22pF	22pF	450ms	(*eoscr*[6:5]=01, *misc*.6=0)

表 4：不同的晶体或谐振器建议的 C1，C2 电容值

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

```
void    FPPA0 (void)
{
    . ADJUST_IC    SYSCLK=IHRC/16, IHRC=16MHz, VDD=5V
$   EOSCR      Enable, 4MHz;          // EOSCR = 0b110_00000;

$   T16M  EOSC, /1, BIT13;            // T16 收到 2^14=16384 个晶体振荡时钟,
                                      // Intrq.T16 =>1，晶体振荡器已稳定
    WORD       count =      0;
    stt16  count;
    Intrq.T16    =      0;
    do
    {   nop; }while(!Intrq.T16);      // 计数从 0x0000 to 0x2000，然后设置 INTRQ.T16
    clkmd=     0xB4;                  // 将系统时钟切换到 EOSC;

    Clkmd.4 = 0;                      // 关闭 IHRC
    ...
```

需要注意的是：在进入睡眠模式之前，为了避免不可预期的唤醒发生，请将晶体振荡器完全关闭。

5.5.5. 系统时钟和 LVR 基准位

系统时钟的时钟源来自 EOSC，IHRC 和 ILRC，PGS134 的时钟系统的硬件框图，如图 3 所示。

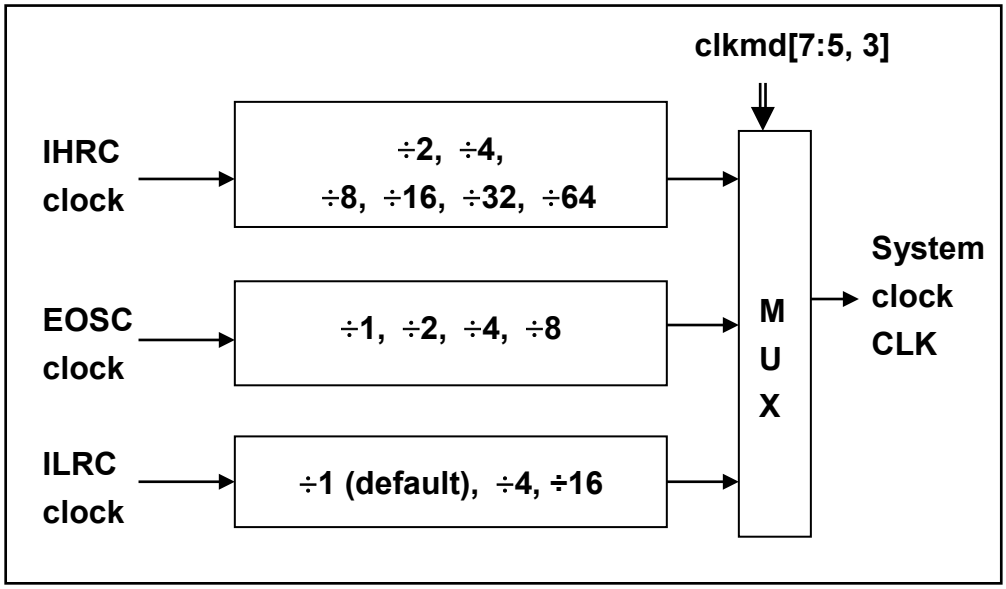


图 3：系统时钟源选项

使用者可以在不同的需求下选择不同的系统时钟，选定的系统时钟应与电源电压和 LVR 的基准位结合起来才能使系统稳定。LVR 的基准位是在编译过程中选择，不同系统时钟对应的 LVR 设定，请参考章节 4.1 中系统时钟的最低工作电压。

5.5.6. 系统时钟切换

IHRC 校准后，用户可能要求切换系统时钟到新的频率或者可能会随时切换系统时钟来优化系统性能及功耗。基本上，PGS134 的系统时钟能够随时通过设定寄存器 ***clkmd*** 在 IHRC，ILRC 和 EOSC 之间切换。在设定寄存器 ***clkmd*** 之后，系统时钟立即转换成新的频率。**请注意，在下命令给 *clkmd* 寄存器时，不能同时关闭原来的时钟模块**，下面这些例子显示更多时钟切换需知道的信息，请参阅 IDE 工具“求助” -> “使用手册” -> “IC 介绍” -> “缓存器介绍” -> CLKMD” 。

例 1: 系统时钟从 ILRC 切换到 IHRC/2

```
...                             //  系统时钟是 ILRC
CLKMD.4         =   1;          //  先打开 IHRC，可以提高抗干扰能力
CLKMD           =   0x34;       //  切换到 IHRC/2，ILRC 不能在这里停用
// CLKMD.2      =   0;          //  假如需要，ILRC 可以在这里停用
...
```

例 2: 系统时钟从 ILRC 切换到 EOSC

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

例 3: 系统时钟从 IHRC/2 切换到 ILRC

```
...                             //  系统时钟是 IHRC/2
CLKMD           =   0xF4;       //  切换到 ILRC，IHRC 不能在这里停用
CLKMD.4         =   0;          //  IHRC 可以在这里停用
...
```

例 4: 系统时钟从 IHRC/2 切换到 EOSC

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

例 5: 系统时钟从 IHRC/2 切换到 IHRC/4

```
...                             //  系统时钟是 IHRC/2, ILRC 在这里是启用的
CLKMD           =   0X14;       //  切换到 IHRC/4
...
```

例 6: 如果同时切换系统时钟关闭原来的振荡器，**系统会当机**

```
...                             //  系统时钟是 ILRC
CLKMD           =   0x30;       //  不能从 ILRC 切换到 IHRC/2 同时关闭 ILRC 振荡器
...
```

5.6. 比较器

PGS134 内置一个硬件比较器，如图.4 所示比较器硬件原理框图。它可以比较两个引脚之间的信号或者与内部参考电压 $V_{internal\ R}$ 或者与内置 bandgap(1.2v)做比较。两个信号进行比较，一个是正输入，另一个是负输入。比较器的负输入可以是 PA3，PA4，内置 bandgap(1.2v)，PB6，PB7，或者内部参考电压 $V_{internal\ R}$，并由寄存器 gpcc 的[3:1]位来选择。比较器的正输入 可以是 PA4 或者 $V_{internal\ R}$，并由 gpcc 寄存器的位 0 来选择。

比较器的输出结果可以选择直接输出到 PA0，或者通过 Timer2 计数器时钟模块(TM2_CLK)采样，另外，信号是否反极性也是可选的，比较器输出结果可以用产生中断信号或者通过 gpcc 寄存器的方式读取。

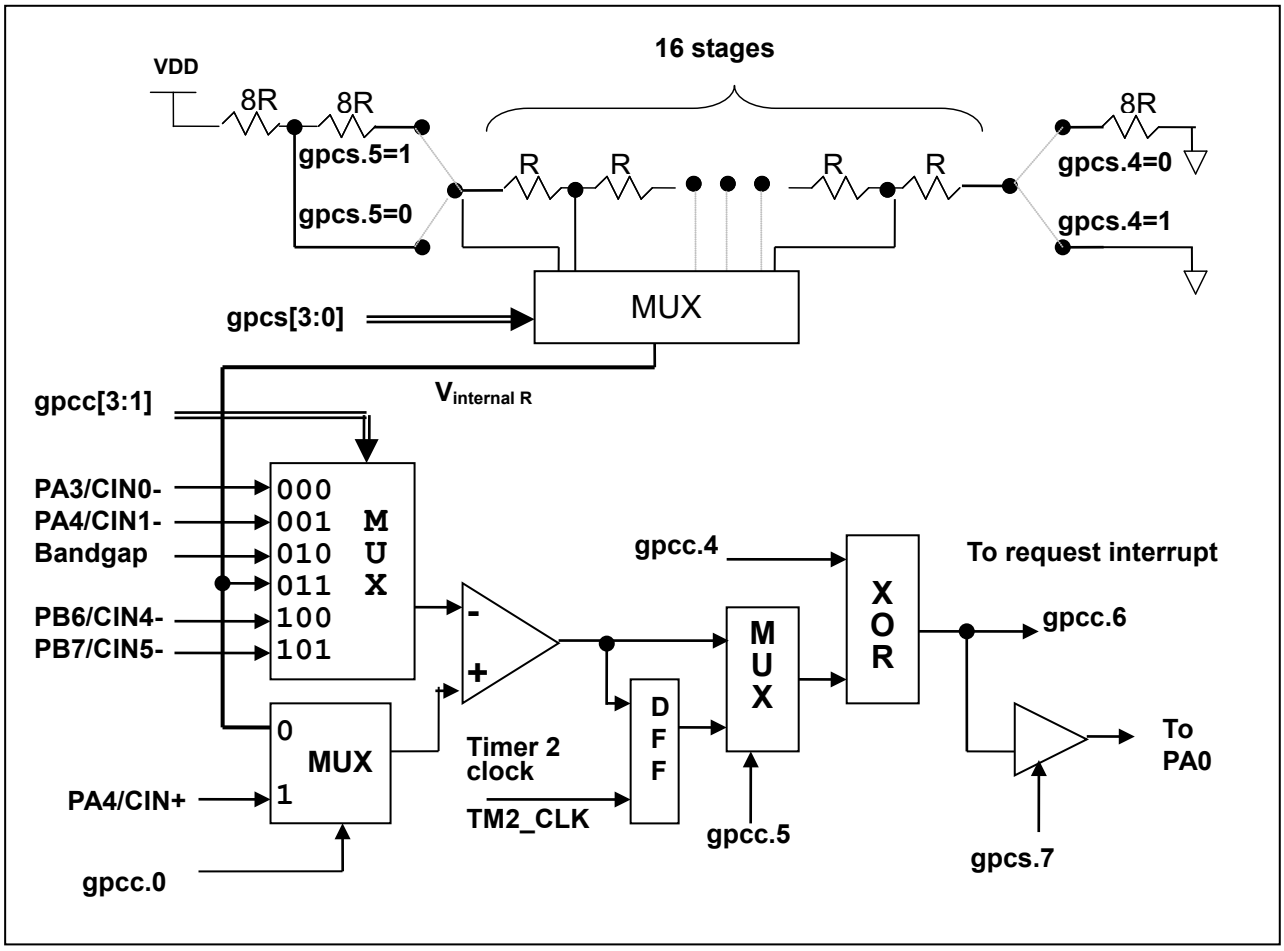


图 4：比较器硬件原理框图

5.6.1. 内部参考电压 ($V_{internal\ R}$)

内部参考电压 $V_{internal\ R}$ 由一连串电阻所组成，可以产生不同层次的参考电压，***gpcs*** 寄存器的位 4 和位 5 是用来选择 $V_{internal\ R}$ 的最高和最低值，位[3:0]用于选择所要的电压水平，这电压水平是由 $V_{internal\ R}$ 的最高和最低值均分 16 等份，由位[3:0]选择出来。图.5~图.8 显示四个条件下有不同的参考电压 $V_{internal\ R}$。内部参考电压 $V_{internal\ R}$ 可以通过 gpcs 寄存器来设置，范围从(1/32)*V_{DD} 到 (3/4)*V_{DD}。

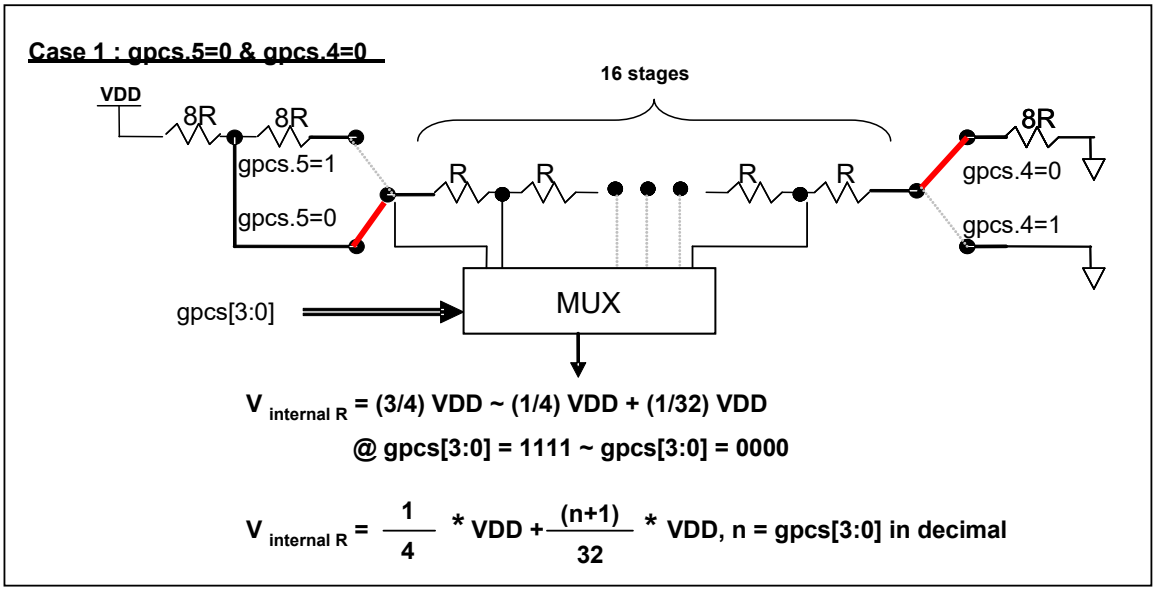


图 5：$V_{internal\ R}$ 硬件接法(gpcs.5=0 & gpcs.4=0)

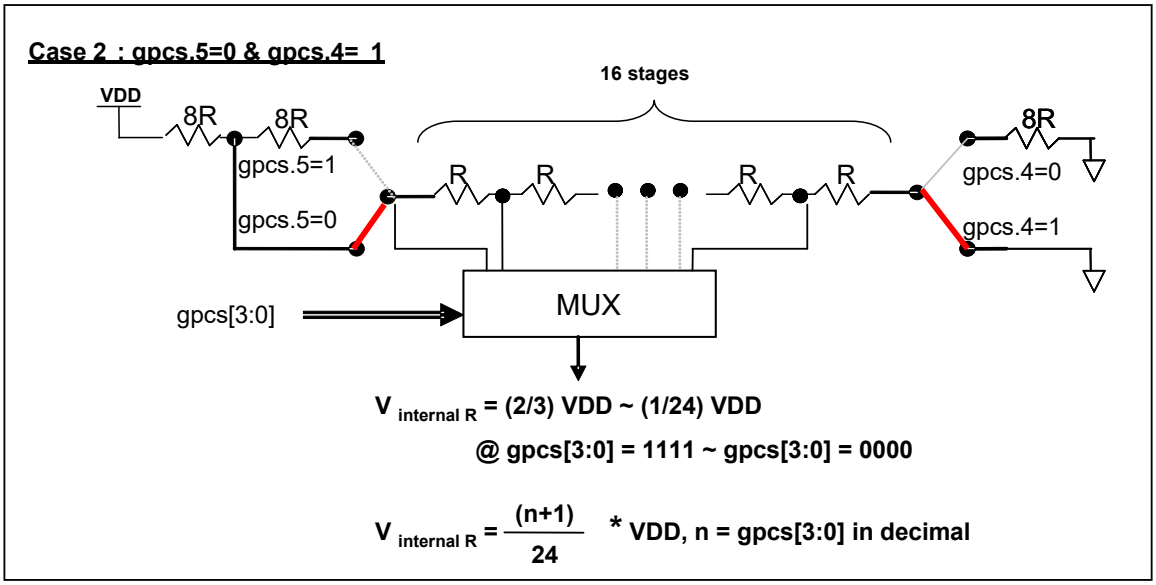


图 6：$V_{internal\ R}$ 硬件接法(gpcs.5=0 & gpcs.4=1)

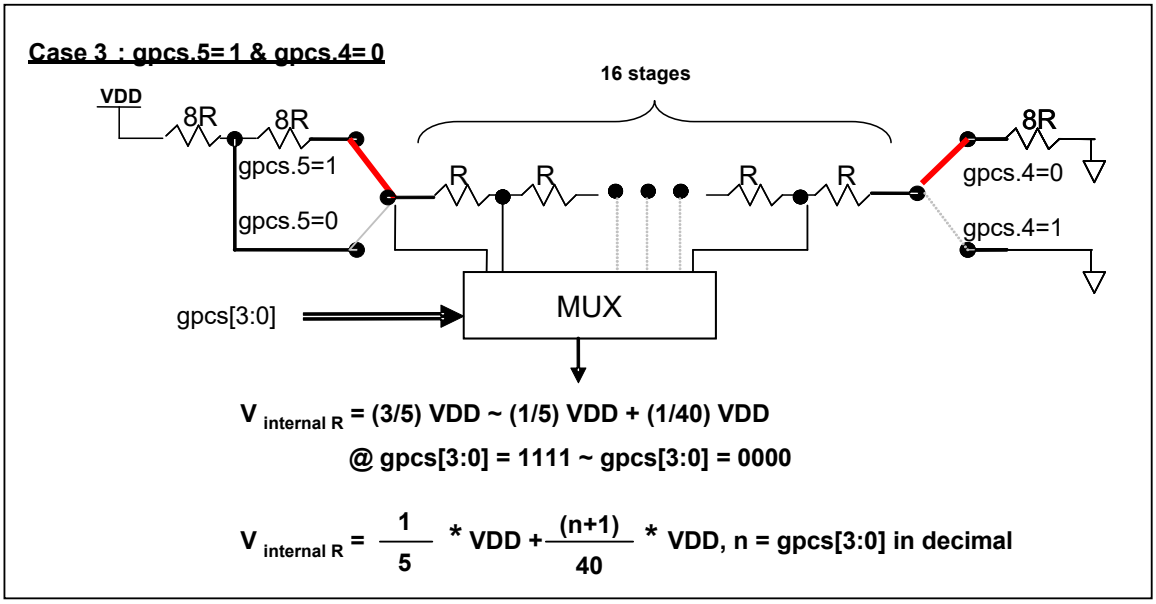


图 7：$V_{internal\ R}$ 硬件接法(gpcs.5=1 & gpcs.4=0)

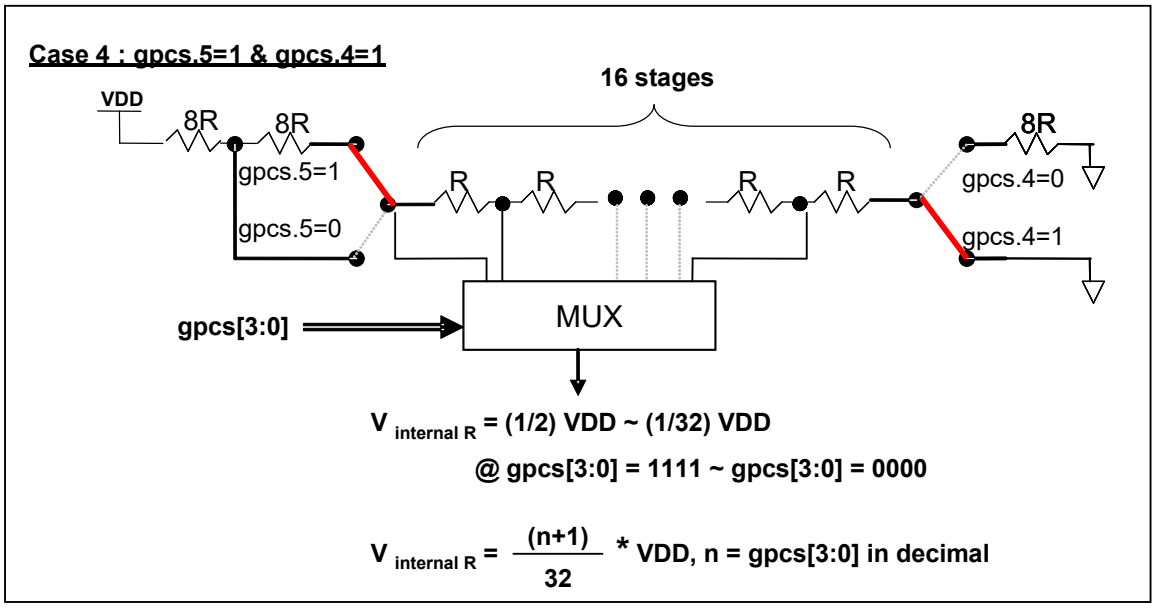


图 8：$V_{internal\ R}$ 硬件接法(gpcs.5=1 & gpcs.4=1)

5.6.2. 使用比较器

例 1：

选择 PA3 为负输入和 $V_{internal\ R}$ 的电压为(18/32)*V_{DD} 作为正输入。 $V_{internal\ R}$ 选择上图 gpcs[5:4] = 2b'00 的配置方式，gpcs [3:0] = 4b'1001 (n=9)以得到 $V_{internal\ R}$ = (1/4)*V_{DD} + [(9+1)/32]*V_{DD} = [(9+9)/32]*V_{DD} = (18/32)*V_{DD} 的参考电压。

```
gpcs    = 0b0_0_00_1001;        // Vinternal R = 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 表示用户自定)
```

或

```
$ GPCS      VDD*18/32;
$ GPCC      Enable, N_PA3, P_R;     //  N_xx 是负输入，P_R 代表正输入是内部参考电压
PADIER = 0bxxxx_0_xxx;
```

例 2：

选择 $V_{internal\ R}$ 为负输入，$V_{internal\ R}$ 的电压为(22/40)*V_{DD}，选择 PA4 为正输入，比较器的结果将反极性并输出到 PA0。$V_{internal\ R}$ 选择上图 gpcs[5:4] = 2b'10 的配置方式，gpcs [3:0] = 4b'1101 (n=13)以得到 $V_{internal\ R}$ = (1/5)*V_{DD} + [(13+1)/40]*V_{DD} = [(13+9)/40]*V_{DD} = (22/40)*V_{DD}。

```
gpcs    = 0b1_0_10_1101;        // 输出到 PA0，Vinternal R = VDD*(22/40)
gpcc    = 0b1_0_0_1_011_1;      // 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
```

或

```
$ GPCS      Output, VDD*22/40;
$ GPCC      Enable, Inverse, N_R, P_PA4;   //  N_R 代表负输入是内部参考电压，P_xx 是正输入
PADIER=0bxxx_0_xxxx;
```

注意：当选择 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。

5.6.3. 使用比较器和 bandgap 1.20V

内部 Bandgap 参考电压生成器可以提供 1.20V，它可以测量外部电源电压水平。该 Bandgap 参考电压可以选做负输入去和正输入 $V_{internal\ R}$ 比较。$V_{internal\ R}$ 的电源是 V_{DD}，利用调整 $V_{internal\ R}$ 电压水平和 Bandgap 参考电压比较，就可以知道 V_{DD} 的电压。如果 N（***gpcs***[3:0]十进制）是让 $V_{internal\ R}$ 最接近 1.20V，那么 V_{DD} 的电压就可以透过下列公式计算：

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

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

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

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

例一：

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

5.7. VDD/2 LCD 偏置电压产生器

LCD 功能可以透过寄存器 misc.4 和代码选项 LCD2 启动与设置。有二组引脚可做为 LCD COM 的端口，每一组各有 4 支引脚。通过代码选项 LCD2 选择 PB0_A034，PB0、PA0、PA4 和 PA3 这四支引脚可以输出 VDD/2，以做为驱动液晶显示器时 COM 的功能。而通过代码选项 LCD2 选择 PB1256，PB1、PB2、PB5 和 PB6 将做为 COM 的端口。

如果使用者想要输出 VDD、VDD/2 和 GND 三个电位，只要设置 *misc*.4=1 启动功能，然后输出高电位(VDD)、设置为输入(VDD/2)和输出低电位(GND)即可产生三种相对应的电位，图 9 显示了如何使用此功能。

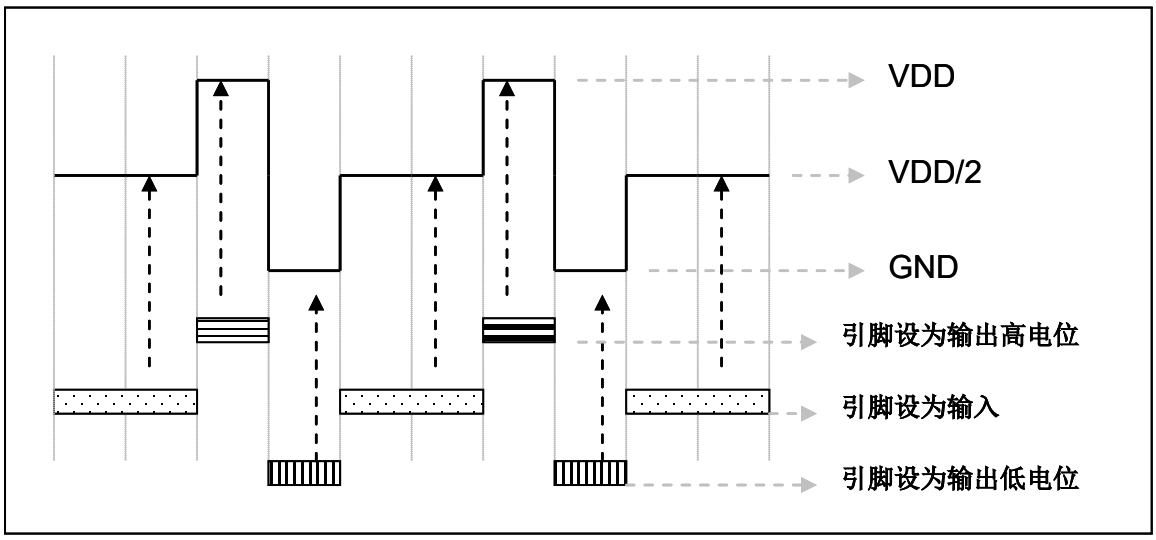


图 9：使用 VDD/2 LCD 偏置电压产生器

5.8. 16 位计数器(Timer16)

PGS134 内置一个 16 位硬件计数器(Timer16)，计数器时钟可来自于系统时钟(CLK)，外部晶体振荡器时钟(EOSC)，内部高频振荡时钟(IHRC)，内部低频振荡时钟(ILRC)，PA4 和 PA0，一个多任务器用来选择时钟输出的时钟来源。在送到 16 位计数器之前，1 个可软件编程的预分频器提供÷1、÷4、÷16、÷64 选择，让计数范围更大。

16 位计数器只能向上计数，计数器初始值可以使用 stt16 指令来设定，而计数器的数值也可以利用 ldt16 指令存储到 SRAM 数据存储器。可软件编程的选择器用于选择 Timer16 的中断条件，当计数器溢出时，Timer16 可以触发中断。Timer16 模块框图如图.10 所示。中断源是来自 16 位计数器的位 8 到位 15，中断类型可以上升沿触发或下降沿触发，定义在寄存器 ***integs*.5**（IO 地址是 0x0C）。

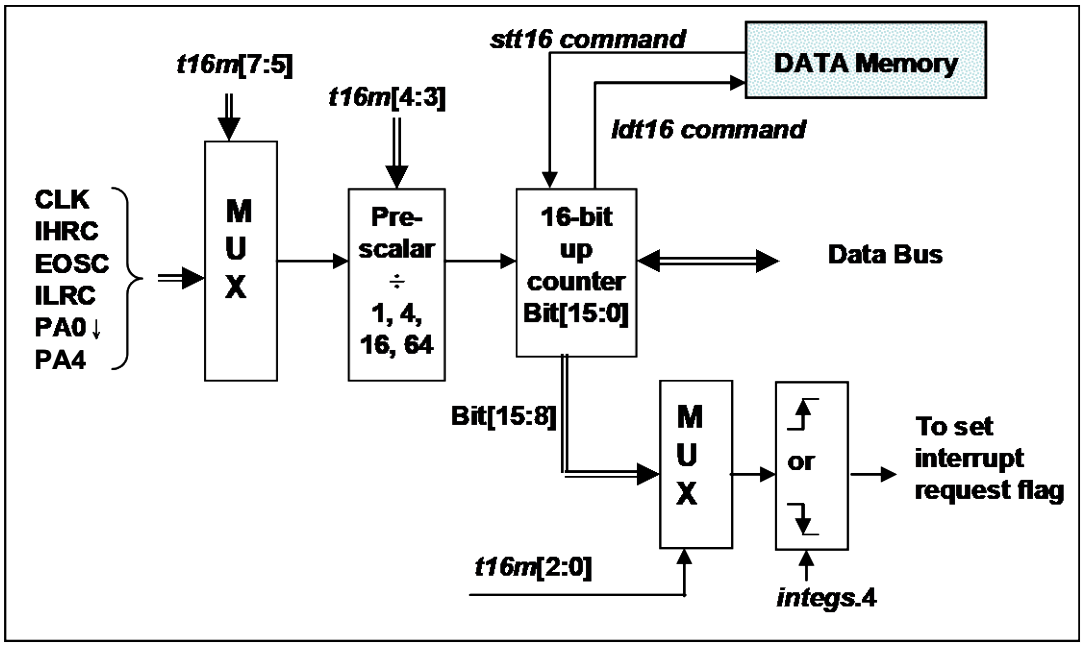


图 10：Timer16 模块框图

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

```
T16M        IO_RW           0x06
        $ 7~5: STOP, SYSCLK, X, PA4_F, IHRC, 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，更多例子请参考 IDE 软件“帮助→ 使用手册→ IC 介绍 → 缓存器介绍 → T16M”。

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

$ T16M EOSC, /1, BIT13;
    // 选择(EOSC/1)当 Timer16 时钟源，每 2^14 个时钟周期产生一次 INTRQ.2=1
    // EOSC=32768 Hz，32768 Hz/(2^14) = 2Hz，每 0.5S 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。

5.9. 8 位 PWM 计数器(Timer2/Timer3)

PGS134 内置 2 个 8 位硬件 PWM 计数器(Timer2/Timer3)。以下描述只以 Timer2 为例，因为 Timer3 和 Timer2 结构是一样的。图 11 为 Timer2 硬件框图，计数器的时钟源可以来自系统时钟(CLK) ，内部高频 RC 振荡器时钟(IHRC)，内部低频 RC 振荡器时钟(ILRC)，外部晶体振荡器(EOSC)，PA0，PB0，PA4 和比较器。寄存器 tm2c 的位[7:4]用来选择 Timer2 的时钟。如果 IHRC 作为 Timer2 的时钟源，当仿真器停住时，IHRC 时钟仍然会送到 Timer2，所以 Timer2 仍然会计数。 根据 tm2c 寄存器位[3:2]的设定，Timer2 的输出可以是 PB2，PA3 或 PB4 引脚。利用软件编程寄存器 ***tm2s*** 位[6:5]，时钟预分频模块提供÷1，÷4，÷16 和÷64 的选择，另外，利用软件编程寄存器 ***tm2s*** 位[4:0]，时钟分频器的模块提供了÷1~÷31 的功能。在结合预分频器以及分频器，Timer2 时钟(TM2_CLK)频率可以广泛和灵活，以提供不同产品应用。

8 位 PWM 定时器只能执行 8 位上升计数操作，经由寄存器 ***tm2ct***，定时器的值可以设置或读取。当 8 位定时器计数值达到上限寄存器设定的范围时，定时器将自动清除为零，上限寄存器用来定义定时器产生波形的周期或 PWM 占空比。8 位 PWM 定时器有两个工作模式：周期模式和 PWM 模式；周期模式用于输出固定周期波形或中断事件；PWM 模式是用来产生 PWM 输出波形，PWM 分辨率可以为 6 位到 8 位。图 12 显示出 Timer2 周期模式和 PWM 模式的时序图。

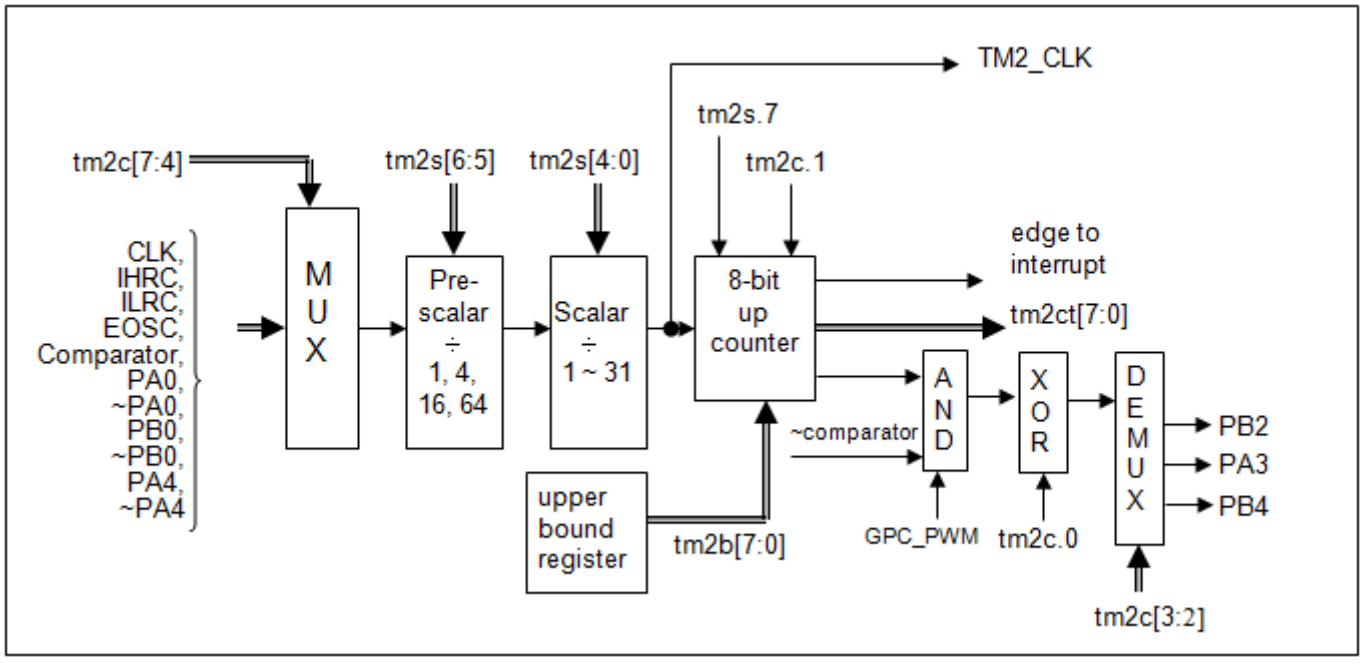


图 11：Timer2 硬件框图

Timer3 的输出可以是 PB5，PB6 或 PB7。

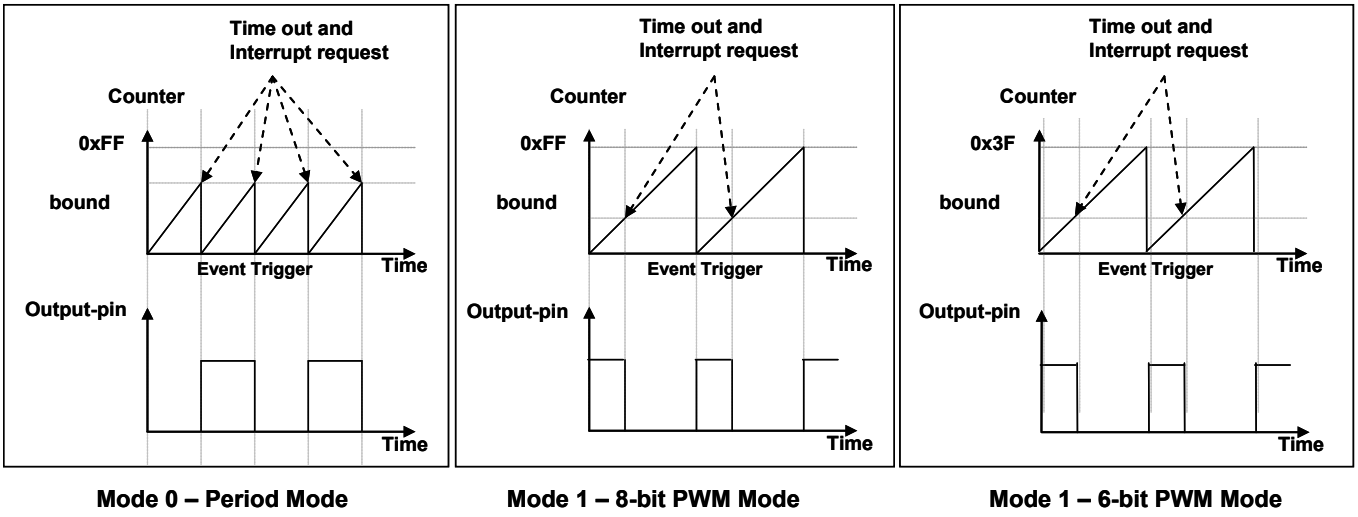


图 12：Timer2 周期模式和 PWM 模式的时序图(tm2c.1=1)

程序选项" GPC_PWM "是指根据需求由比较器结果控制生成 PWM 波形的功能。如果程序选项"GPC_PWM" 被选中后，此时当比较器输出是 1 时，PWM 停止输出；而比较器输出是 0 时，PWM 恢复输出，如图 13 所示。

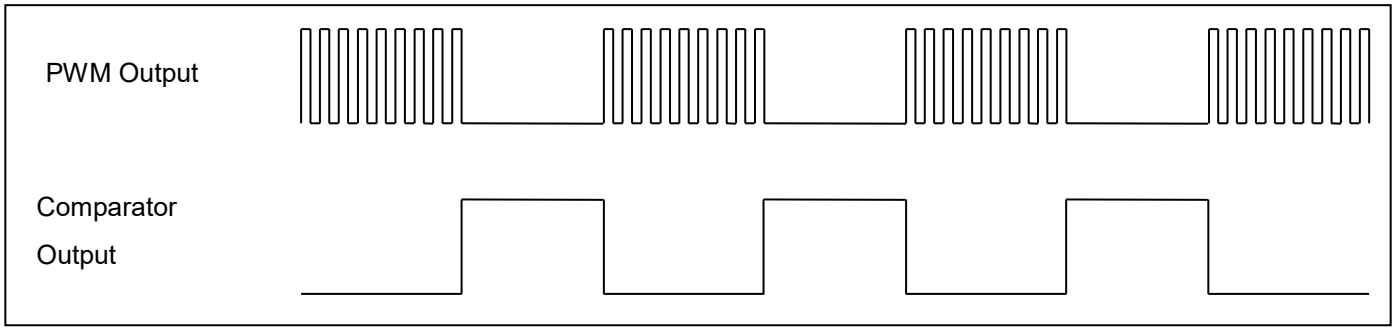


图 13：比较器控制 PWM 输出

5.9.1. 使用 Timer2 产生周期波形

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)]

Y = tm2c[7:4] : Timer2 所选择的时钟源频率

K = tm2b[7:0] : 上限寄存器设定的值（十进制）

S1 = tm2s[6:5] : 预分频器设定值 (S1= 1，4，16，64)

S2 = tm2s[4:0] : 分频器值（十进制，S2= 0 ~ 31）

例 1：

tm2c = 0b0001_1000, Y=8MHz

tm2b = 0b0111_1111, K=127

tm2s = 0b0000_00000, S1=1, S2=0

➔ 输出频率= 8MHz ÷ [2 × (127＋1) × 1 × (0＋1)] = 31.25kHz

例 2：

tm2c = 0b0001_1000, Y=8MHz

tm2b = 0b0111_1111, K=127

tm2s[7:0] = 0b0111_11111, S1=64 , S2 = 31

➔ 输出频率= 8MHz ÷ (2 × (127＋1) × 64 × (31＋1)) =15.25Hz

例 3：

tm2c = 0b0001_1000, Y=8MHz

tm2b = 0b0000_1111, K=15

tm2s = 0b0000_00000, S1=1, S2=0

➔ 输出频率= 8MHz ÷ (2 × (15＋1) × 1 × (0＋1)) = 250kHz

例 4：

tm2c = 0b0001_1000, Y=8MHz

tm2b = 0b0000_0001, K=1

tm2s = 0b0000_00000, S1=1, S2=0

➔ 输出频率 = 8MHz ÷ (2 × (1＋1) × 1 × (0＋1)) =2MHz

使用 Timer2 定时器从 PA3 引脚产生周期波形的示例程序如下所示：

```
Void FPPA0 (void)
{
    . ADJUST_IC    SYSCLK=IHRC/2, IHRC=16MHz, VDD=5V
    …
    tm2ct = 0x0;
    tm2b = 0x7f;
    tm2s = 0b0_00_00001;              //    8-bit PWM, pre-scalar = 1, scalar = 2
    tm2c = 0b0001_10_0_0;             //    系统时钟，输出=PA3，周期模式
    while(1)
    {
        nop;
    }
}
```

5.9.2. 使用 Timer2 产生 8 位 PWM 波形

如果选择 8 位 PWM 的模式，应设立 tm2c [1] = 1，tm2s [7] = 0，输出波形的频率和占空比可以概括如下：

输出频率 = Y ÷ [256 × S1 × (S2+1)]
输出占空比 = [(K＋1) ÷ 256]×100%

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

例 1：

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s = 0b0000_00000, S1=1, S2=0
➔ 输出频率 = 8MHz ÷ (256 × 1 × (0+1)) = 31.25kHz
➔ 输出占空比 = [(127+1) ÷ 256] × 100% = 50%

例 2：

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s = 0b0111_11111, S1=64, S2=31
➔ 输出频率 = 8MHz ÷ (256 × 64 × (31+1)) = 15.25Hz
➔ 输出占空比 = [(127+1) ÷ 256] × 100% = 50%

例 3：

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b1111_1111, K=255
tm2s = 0b0000_00000, S1=1, S2=0
➔ PWM 输出是高电平
➔ 输出占空比 = [(255+1) ÷ 256] × 100% = 100%

例 4：

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0000_1001, K = 9
tm2s = 0b0000_00000, S1=1, S2=0
➔ 输出频率 = 8MHz ÷ (256 × 1 × (0+1)) = 31.25kHz
➔ 输出占空比 = [(9+1) ÷ 256] × 100% = 3.9%

使用 Timer2 定时器从 PA3 产生 PWM 波形的示例程序如下所示：

```
void    FPPA0 (void)
{
    .ADJUST_IC      SYSCLK=IHRC/2, IHRC=16MHz, VDD=5V
    tm2ct = 0x0;
    tm2b = 0x7f;
    tm2s = 0b0_00_00001;            //      8-bit PWM，预分频 = 1，分频 = 2
    tm2c = 0b0001_10_1_0;           //      系统时钟，输出=PA3, PWM 模式
    while(1)
    {
        nop;
    }
}
```

5.9.3. 使用 Timer2 产生 6 位 PWM 波形

如果选择 6 位 PWM 的模式，应设立 tm2c [1] = 1，tm2s [7] = 1，输出波形的频率和占空比可以概括如下：

输出频率 = Y ÷ [64 × S1 × (S2+1)]
输出占空比 = [(K＋1) ÷ 64] × 100%

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

用户可以通过用 TMx_Bit 这个 code option，选择 7 位 PWM 模式替代原来的 6 位 PWM 模式。这时在上述方程式中的计算因子将从原来的 64 变成 128。

例 1：

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0001_1111, K=31
tm2s = 0b1000_00000, S1=1, S2=0
➔ 输出频率 = 8MHz ÷ (64 × 1 × (0+1)) = 125kHz
➔ 输出占空比 = [(31+1) ÷ 64] × 100% = 50%

例 2：

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0001_1111, K=31
tm2s = 0b1111_11111, S1=64, S2=31
➔ 输出频率= 8MHz ÷ (64 × 64 × (31+1)) = 61.03 Hz
➔ 输出占空比 = [(31+1) ÷ 64] × 100% = 50%

例 3：

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0011_1111, K=63
tm2s = 0b1000_00000, S1=1, S2=0
➔ PWM 输出是高电平
➔ 输出占空比 = [(63+1) ÷ 64] × 100% = 100%

例 4：

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0000_0000, K=0
tm2s = 0b1000_00000, S1=1, S2=0
➔ 输出频率= 8MHz ÷ (64 × 1 × (0+1)) = 125kHz
➔ 输出占空比 = [(0+1) ÷ 64] × 100% =1.5%

5.10.11 位 PWM 计数器

PGS134 内置一路 11 位计数器同时提供三个硬件 PWM 生成器(PWMG0, PWMG1 & PWMG2) 。以下只描述 PWMG0 的应用，因为 PWMG1 & PWMG2 的用法和 PWMG0 相同。

各路输出端口如下：

- PWMG0 – PA0, PB4, PB5, PC2
- PWMG1 – PA4, PB6, PB7, PC3
- PWMG2 – PA3, PB2, PB3, PC0

5.10.1. PWM 波形

PWM 波形（图 13）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($f_{PWM} = 1/T_{Period}$)。

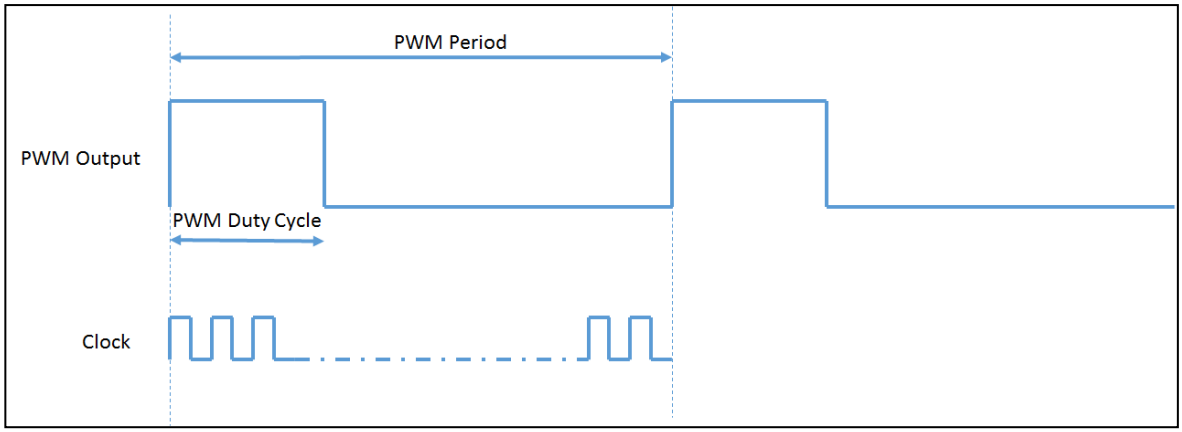


图 14：PWM 输出波形

5.10.2. 硬件时序框图

PGS134 内置三个 11 位硬件 PWM 生成器，图.14 所示是硬件框图。时钟源可以是 IHRC 或者系统时钟，输出引脚可以通过 ***pwmc*** 寄存器来选择。PWM 的周期由 PWM 上限高和低寄存器决定，PWM 的占空比由 PWM 占空比高和低寄存器决定。用户也可以通过用 GPC_PWM option code，令比较器可控制 PWM 波形的输出。

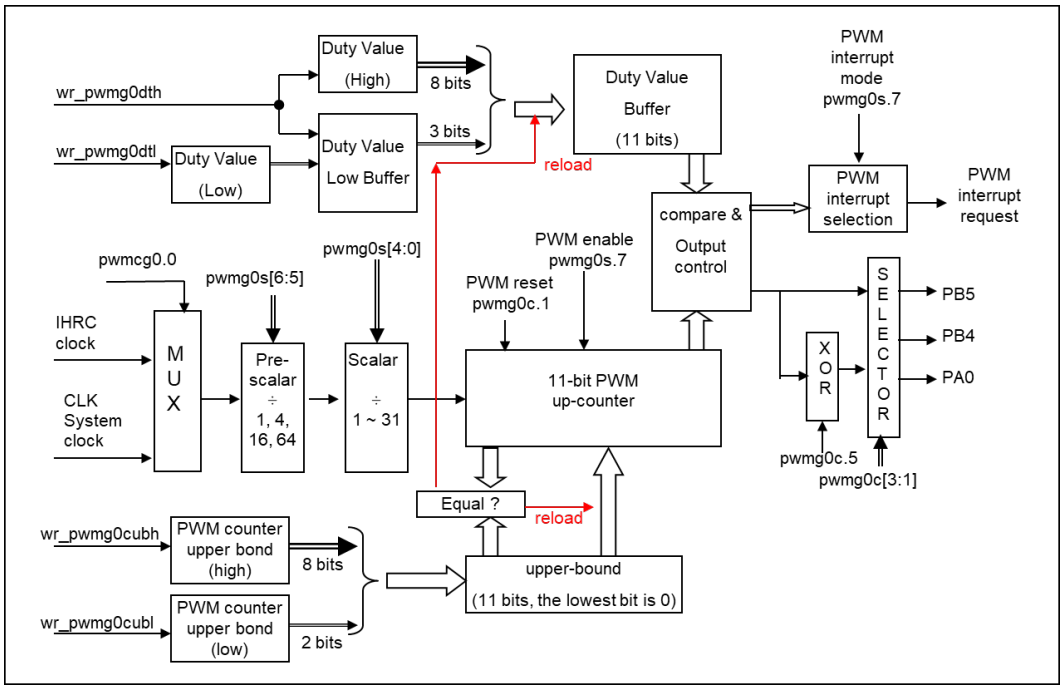


图 15：11 位 PWM 生成器硬件框图

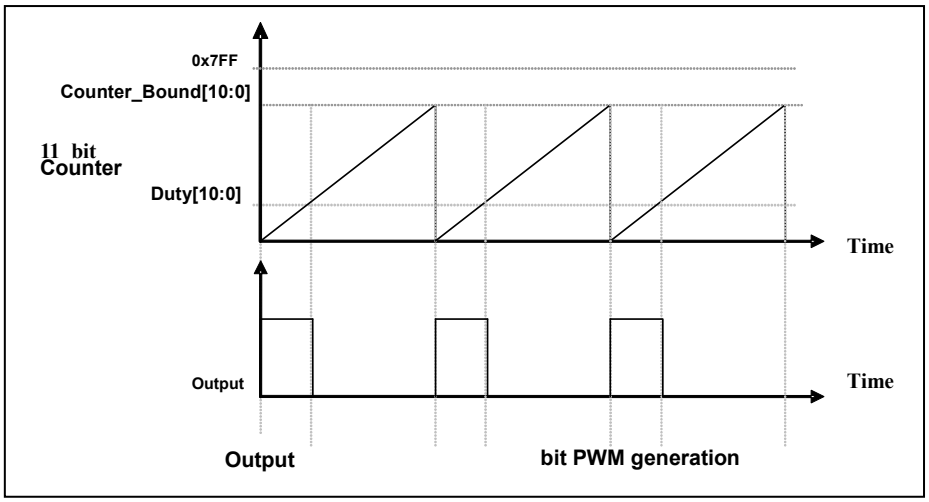


图 16：11 位 PWM 生成器输出时序图

5.10.3. 11 位 PWM 生成器计算公式

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%

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

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

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

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

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

5.11. 看门狗计数器

看门狗是一个计数器(WDT)，其时钟源来自内部低频振荡器(ILRC)，可以通过上电复位和 ***wdreset*** 指令随时清零看门狗计数，利用 ***misc*** 寄存器的选择，可以设定四种不同的看门狗超时时间，它是：

- 当 misc[1:0]=00（默认）时：8k ILRC 时钟周期
- 当 misc[1:0]=01 时：16k ILRC 时钟周期
- 当 misc[1:0]=10 时：64k ILRC 时钟周期
- 当 misc[1:0]=11 时：256k ILRC 时钟周期

ILRC 的频率有可能因为工厂制造的变化，电源电压和工作温度而漂移很多，使用者必须预留安全操作范围。由于在系统重启或者唤醒之后，看门狗计数周期会比预计要短，为防止看门狗计数溢出导致复位，建议在系统重启或唤醒之后使用立即 wdreset 指令清零看门狗计数。

当看门狗超时溢出时，PGS134 将复位并重新运行程序。看门狗时序图如图.16 所示。

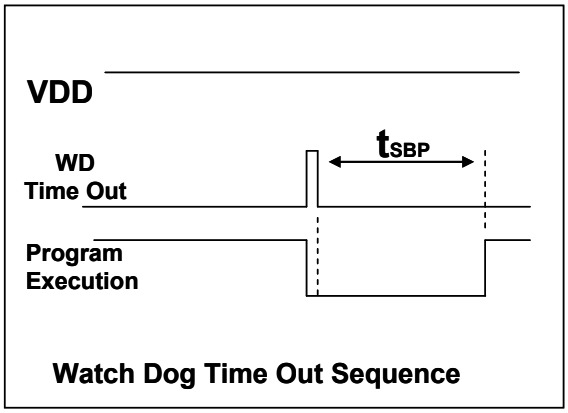


图 17：看门狗超时溢出时序图

5.12. 中断

PGS134 有 8 个中断源：

- 外部中断源 PA0/PB5/PA2/PA7
- 外部中断源 PB0/PA4/PB6/PA3
- ADC 中断
- Timer16 中断
- GPC 中断
- PWMG0 或 EEW_Done 中断
- Timer2 中断
- Timer3 中断

每个中断请求源都有自己的中断控制位来启用或停用。中断功能的硬件框图如图.16 所示。所有的中断请求标志位是由硬件置位并且并通过软件写寄存器 ***intrq*** 清零。中断请求标志设置点可以是上升沿或下降沿或两者兼而有之，这取决于对寄存器 ***integs*** 的设置。所有的中断请求源最后都需由 ***engint*** 指令控制（启用全局中断）使中断运行，以及使用 ***disgint*** 指令（停用全局中断）停用它。

中断堆栈是共享数据存储器，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。

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

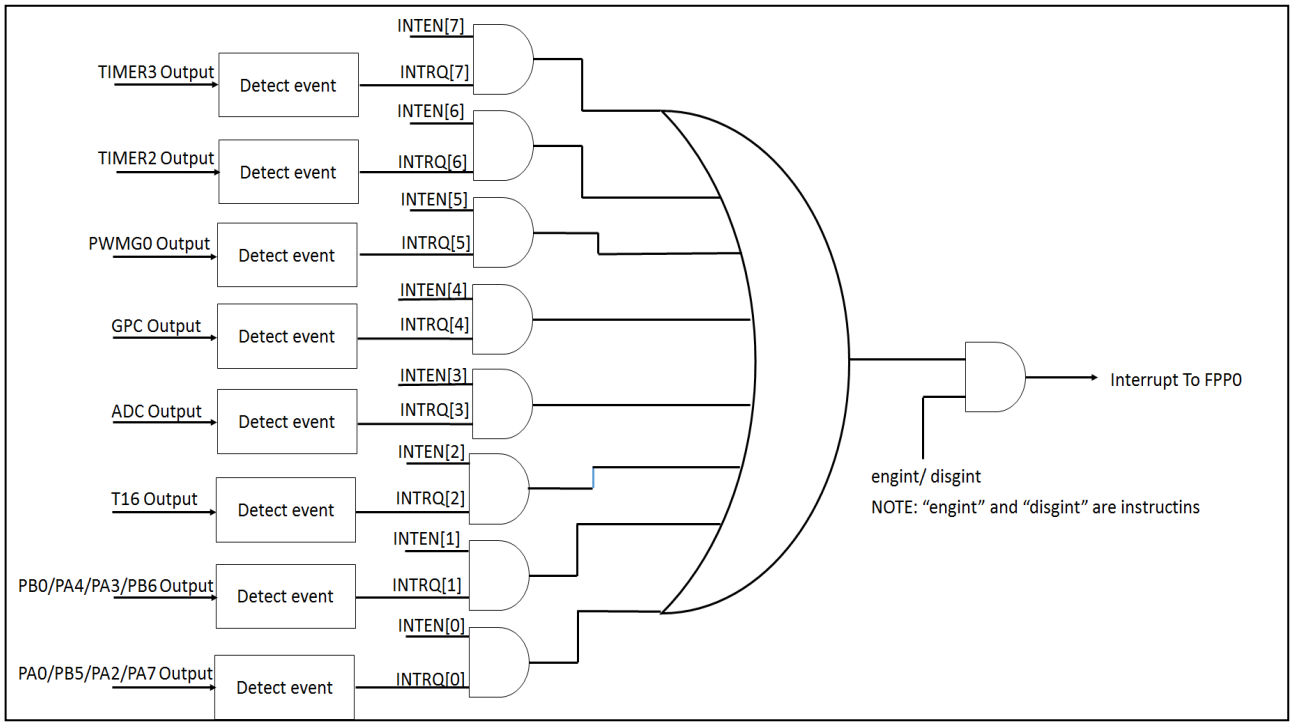


图 18：中断控制器硬件框图

一旦发生中断，其具体工作流程将是：

- 程序计数器将自动存储到 ***sp*** 寄存器指定的堆栈存储器。
- 新的 ***sp*** 将被更新为 ***sp+2***。
- 全局中断将被自动停用。
- 将从地址 0x010 获取下一条指令。

在中断服务程序中，可以通过读寄存器 ***intrq*** 知道中断发生源。

注意：即使 INTEN 为 0，INTRQ 还是会被中断发生源触发。

中断服务程序完成后，发出 **reti** 指令返回既有的程序，其具体工作流程将是：

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

使用者必须预留足够的堆栈存储器以存中断向量，一级中断需要两个字节，两级中断需要 4 个字节。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
}
```

5.13. 省电与掉电

PGS134 有三个由硬件定义的操作模式，分别为：正常工作模式，电源省电模式和掉电模式。正常工作模式是所有功能都正常运行的状态，省电模式(***stopexe***)是在降低工作电流而且 CPU 保持在随时可以继续工作的状态，掉电模式(***stopsys***)是用来深度的节省电力。因此，省电模式适合在偶尔需要唤醒的系统工作，掉电模式是在非常低消耗功率且很少需要唤醒的系统中使用。表 5 显示省电模式(***stopexe***)和掉电模式(***stopsys***)之间在振荡器模块的差异，没改变就是维持原状态。

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

表 5：省电模式和掉电模式在振荡器模块的差异

5.13.1. 省电模式(“*stopexe*”)

使用 ***stopexe*** 指令进入省电模式，只有系统时钟被停用，其余所有的振荡器模块都仍继续工作。所以只有 CPU 是停止执行指令，然而，对 Timer16 计数器而言，如果它的时钟源不是系统时钟，那 Timer16 仍然会保持计数。***stopexe*** 的省电模式下，唤醒源可以是 IO 的切换，或者 Timer16 计数到设定值时（假如 Timer16 的时钟源是 IHRC 或者 ILRC），或比较器唤醒（需同时设定 GPCC.7 为 1 与 GPCS.6 为 1 来启用比较器唤醒功能）。假如系统唤醒是因输入引脚切换，那可以视为系统继续正常运行。省电模式的详细信息如下所示：

- IHRC 和 EOSC 振荡器模块：没改变，如果被启用，则仍然保持运行状态。
- ILRC 振荡器模块：必须保持启用，唤醒时需要靠 ILRC 启动。
- 系统时钟：停用，因此 CPU 停止运行。
- MTP 存储器关闭。
- Timer 计数器：若 Timer 计数器的时钟源是系统时钟或其相应的时钟振荡器模块被停用，则 Timer 停止计数；否则，仍然保持计数。（其中，Timer 包含 Timer16，TM2，TM3，PWMG0，PWMG1，PWMG2）
- 唤醒源：
  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
$ INTEGS    BIT_R, xxx;             //默认值即为 BITx 0 ->1 触发

...
WORD        count =     0;
STT16       count;
stopexe;
...
```

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

5.13.2. 掉电模式(“*stopsys*”)

掉电模式是深度省电的状态，所有的振荡器模块都会被关闭。通过使用“***stopsys***” 指令，芯片会直接进入掉电模式。在下达 stopsys 指令之前建议将 GPCC.7 设为 0 来关闭比较器。下面显示发出 ***stopsys*** 命令后，PGS134 内部详细的状态：

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

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

```
    CLKMD    =    0xF4;        //    系统时钟从 IHRC 变为 ILRC，关闭看门狗时钟
    CLKMD.4  =    0;           //    IHRC 停用
    …
    while (1)
{
                  STOPSYS;     //    进入断电模式
                  if (…) break;//    假如发生唤醒而且检查 OK，就返回正常工作
                               //    否则，停留在断电模式。
}
CLKMD  =    0x34;              //    系统时钟从 ILRC 变为 IHRC/2
```

5.13.3. 唤醒

进入掉电或省电模式后，PGS134 可以通过切换 IO 引脚恢复正常工作；而 Timer 中断的唤醒只适用于省电模式。表 6 显示 ***stopsys*** 掉电模式和 ***stopexe*** 省电模式在唤醒源的差异。

掉电模式（stopsys）和省电模式（stopexe）在唤醒源的差异			
	IO 引脚切换	计时器中断	比较器唤醒
STOPSYS	是	否	否
STOPEXE	是	是	是

表 6：掉电模式和省电模式在唤醒源的差异

当使用 IO 引脚来唤醒 PGS134, ***padier*** 寄存器应对每一个相应的引脚正确设置“使能唤醒功能”。 从 唤醒事件发生后开始计数，正常的唤醒时间大约是 3000 个 ILRC 时钟周期，另外，PGS134 提供快速唤醒功能，透过 ***misc*** 寄存器选择快速唤醒大约 45 个 ILRC 时钟周期。

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

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

5.14. IO 引脚

除了 PA5，PGS134 所有 IO 引脚都可以设定成输入或输出，透过数据寄存器 (***pa, pb, pc***)，控制寄存器 (***pac, pbc, pcc***)和弱上拉电阻 (***paph, pbph, pcph***)或弱下拉电阻 (***papl, pbpl, pcpl***)设定，每一 IO 引脚都可以独立配置成不同的功能；所有这些引脚设置有施密特触发输入缓冲器和 CMOS 输出驱动电位水平。当这些引脚为输出低电位时，弱上拉电阻会自动关闭。如果要读取端口上的电位状态，一定要先设置成输入模式；在输出模式下，读取到的数据是数据寄存器的值。表 7 为端口 PA0 位的设定配置表。图.18 显示了 IO 缓冲区硬件图。

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

表 7：PA0 设定配置表

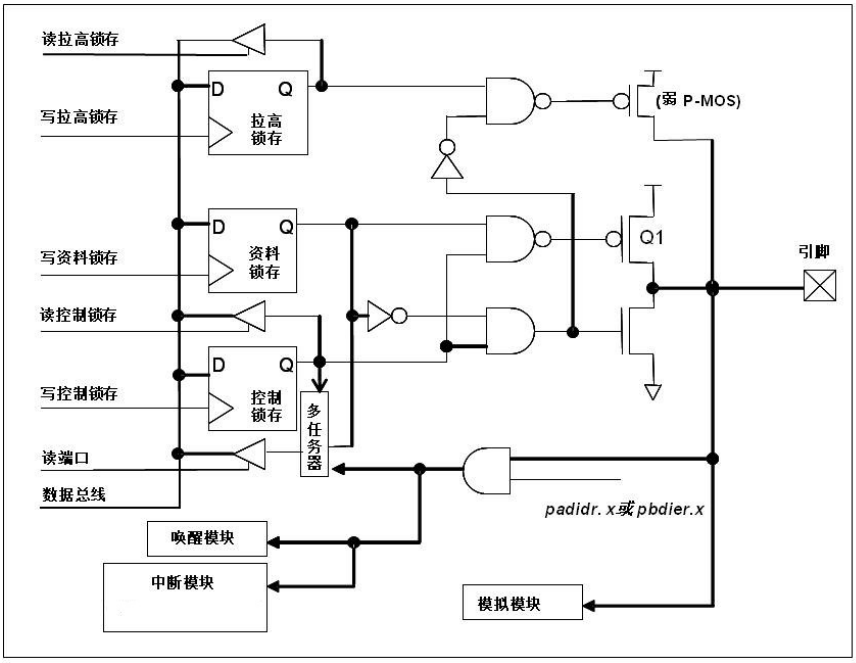


图 19：IO 引脚缓冲区硬件图

用户可使用 code option PB4_PB7_Drive 来控制 PB4 及 PB7 的驱动及灌电流能力。

除了 PA5 外，所有的 IO 引脚具有相同的结构；PA5 的输出只能是漏极开路模式（没有 Q1）。 对于被选择为模拟功能的引脚，必须在寄存器 ***padier*** / ***pbdier*** / ***pcdier*** 相应位设置为低，以防止漏电流。当 PGS134 在掉电或省电模式，每一个引脚都可以切换其状态来唤醒系统。对于需用来唤醒系统的引脚，必须设置为输入模式以及寄存器 ***pxdier*** 相应的 bit 为高。 同样的原因，当 PA0 用作外部中断引脚时，***padier***.0 应设置为高，诸如其他外部中断引脚 PB0，PA4，PB5，PB6，PA2，PA3 及 PA7 都是同样的用法。

5.15. 复位和 LVR

5.15.1. 复位

引起 PGS134 复位的原因很多，一旦复位发生，PGS134 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 V_{DR}（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。

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

5.15.2. LVR 复位

程序编译时，用户可以选择不同级别的 LVR。通常情况下，使用者在选择 LVR 复位水平时，必须结合单片机工作频率和电源电压，以便让单片机稳定工作。

5.16.模拟-数字转换器(ADC) 模块

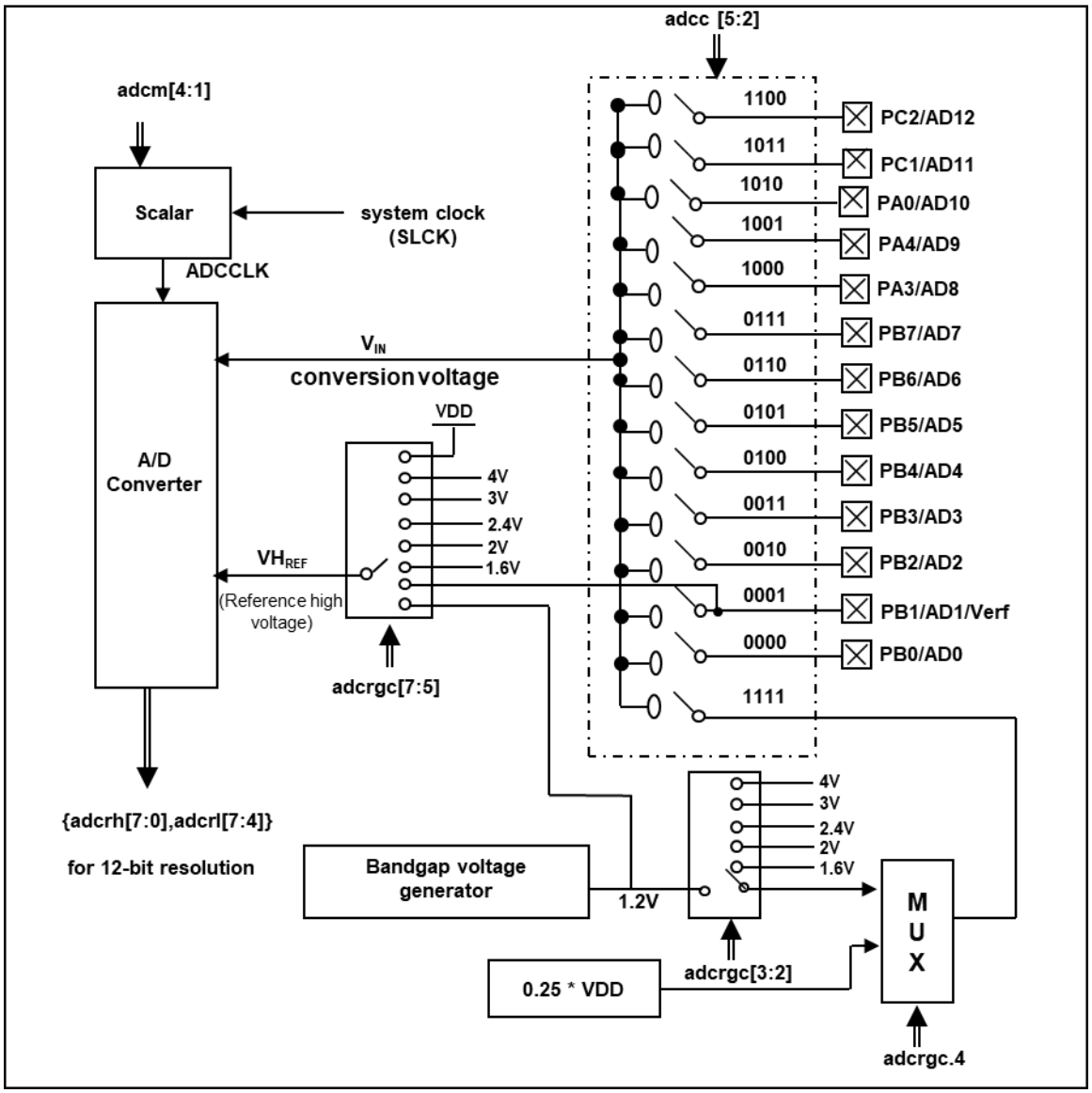


图 20：ADC 模块框图

当使用 ADC 模块时有 7 个寄存器需要配置，它们是：

- ADC 控制寄存器(***adcc***)
- ADC 调节控制寄存器(***adcrgc***)
- ADC 模式寄存器(***adcm***)
- ADC 数据高位/低位寄存器(***adcrh***，***adcrl***)
- 端口 A/B/C 数字输入启用寄存器(***padier***，***pbdier***，***pcdier***)

如下是使用 ADC 的步骤：

(1) 通过寄存器 ***adcrgc*** 配置参考高电压

(2) 通过 ***adcm*** 寄存器配置 AD 转换时钟信号

(3) 通过 ***padier***，***pbdier*** 寄存器配置模拟输入引脚

(4) 通过 ***adcc*** 寄存器选择 ADC 输入通道

(5) 通过 ***adcc*** 寄存器启用 ADC 模块

(6) 启用 ADC 模块之后，延迟一段时间

条件 1：使用 bandgap 1.2V/1.6V/2.4V 或 2V/3V/4V 相关电路时，无论是将其用作内部参考高电压还是作为 AD 输入通道，所需的延迟时间必须超过 1ms；如果 200 个 AD 时钟已经超过 1ms，那么延迟时间只需要 200 个 AD 时钟即可。

条件 2：没有使用任何 bandgap 1.2V/1.6V/2.4V 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

addc.6 设置 1 开启 AD 转换并且检测 ***addc***.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。

5.16.1. AD 转换的输入要求

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

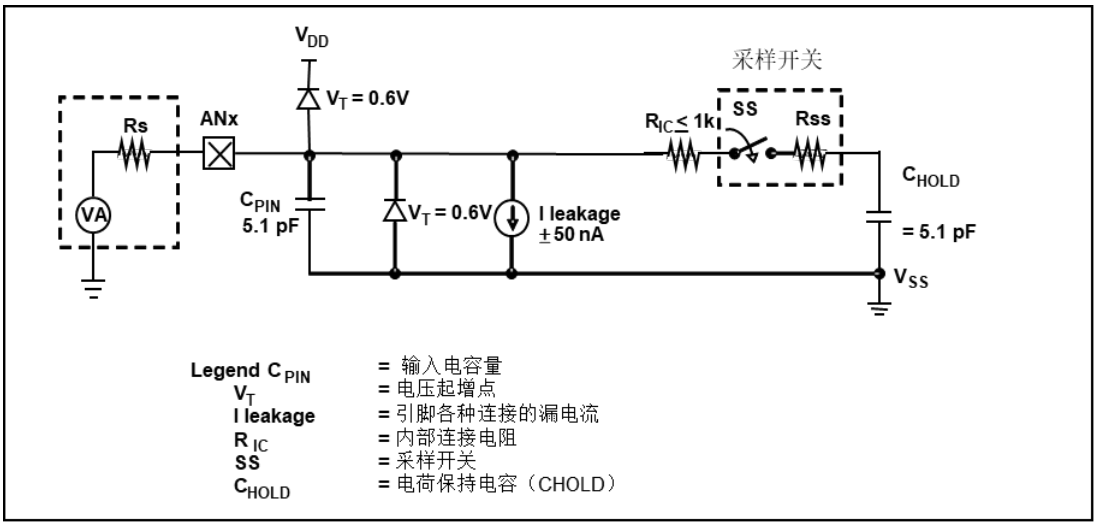


图 21：模拟输入模型

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。

5.16.2. 选择参考高电压

ADC 参考高电压能够通过寄存器 ***adcrgc*** 的位[7:5]来选择，并且它的选择有 V_{DD}，4V，3V，2V，1.20V/1.6V/2.4V bandgap 参考电压或者来自 PB1 外部引脚。

5.16.3. 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 必须满足这已要求，建议 ADC 时钟周期是 2us。

5.16.4. 配置模拟引脚

有 14 模拟信号可以被 AD 转换选择：13 来自外部引脚的模拟输入信号和一个 bandgap 参考电压或者 0.25*V_{DD}。Banggap 有 6 级电压可供选择，分别是：1.2V，1.6V，2V，2.4V，3V 和 4V。以外部引脚而言，12 个模拟信号与 Port A[0]，Port A[3]，Port A[4]和 Port B[7:0]共享引脚。为了避免漏电，这些引脚在使用时定义为模拟输入并应停用数字输入功能（设置 ***padier / pbdier / pcder*** 寄存器的相应位为 0）。

ADC 的测量信号属于小信号，为避免测量信号在测量期间被干扰，被选定的引脚应：(1) 设为输入模式，(2) 关闭弱上拉电阻，(3) 通过端口 A/B/C 寄存器(***padier / pbdier / pcdier***)设置模拟输入并关闭数字输入。

注：PC1 或 PC2 用做 AD 输入通道时，赋值指令不同，仿真与实际情况就会不同，具体差异请参考表 8。其他通道均一致。

实际 IC 所用 AD 通道	指令	仿真时 AD 通道
PC1	$ ADCC enable, PC1;	PC1
PC2	$ ADCC enable, PC2;	PA1

表 8：PC1 和 PC2 仿真时与实际 IC 的差异

5.16.5. 使用 ADC

下面的示例演示使用 PB0∽PB3 来当 ADC 输入引脚。

首先，定义所选择的引脚：

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

下一步，设定 **ADCC** 寄存器，示例如下：

```
$  ADCC Enable, PB3;          //   设置 PB3 作为 ADC 输入
$  ADCC Enable, PB2;          //   设置 PB2 作为 ADC 输入
$  ADCC Enable, PB0;          //   设置 PB0 作为 ADC 输入
                              //   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
$  ADCM 12BIT, /8;            //   建议 /8 @系统时钟=4MHz
$  ADCRGC VDD;
```

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;
```

5.17.乘法器

芯片内置一 8x8 乘法器以加强硬件的运算功能。这个乘法运算方式是 8x8 的无符号运算并且在一个时钟周期内完成运算。在下达指令之前，乘数与被乘数都要放在 ***ACC*** 累加器和 ***mulop***(0x08)寄存器上，在下达 ***mul*** 指令之后，运算结果的高位字节会放在寄存器 ***mulrh***(0x09)上，运算结果的低位字节会放在 ***ACC*** 累加器上。乘法器的硬件框图如图 21 所示。

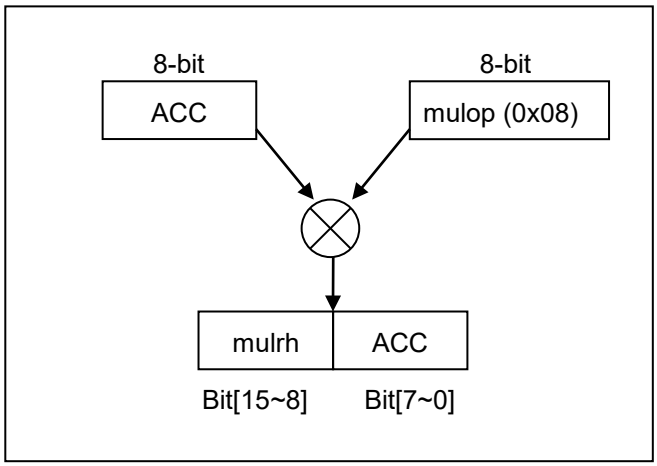


图 22：硬件乘法器框图

6. IO 寄存器

6.1. ACC 状态标志寄存器(*flag*), IO 地址 = 0x00

位	初始值	读/写	描述
7 - 4	-	-	保留。
3	0	读/写	OV（溢出标志）。溢出时置 1。
2	0	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	0	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	0	读/写	Z（零）。此位将被设置为 1，当算术或逻辑运算的结果是 0；否则将被清零。

6.2. 堆栈指针寄存器(*sp*), IO 地址 = 0x02

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

6.3. 时钟模式寄存器(*clkmd*), IO 地址 = 0x03

位	初始值	读/写	描述	
7 - 5	111	读/写	系统时钟(CLK)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（默认值）	000：IHRC÷16 001：IHRC÷8 010：ILRC÷16（仿真器不支持） 011：IHRC÷32 100：IHRC÷64 101：EOSC÷8 11x：保留
4	1	读/写	内部高频 RC 振荡器功能。 0/1：停用/启用	
3	0	读/写	时钟类型选择。这个位是用来选择位 7~位 5 的时钟类型。 0 / 1：类型 0 /类型 1	
2	1	读/写	内部低频 RC 振荡器功能。0/1：停用/启用 当内部低频 RC 振荡器功能停用时，看门狗功能同时被关闭。	
1	1	读/写	看门狗功能。 0/1：停用/启用	
0	0	读/写	引脚 PA5/PRSTB 功能。 0 / 1：PA5 / PRSTB	

6.4. 中断允许寄存器(*inten*), IO 地址 = 0x04

位	初始值	读/写	描述
7	0	读/写	启用从 Timer3 的溢出中断。0/1：停用/启用
6	0	读/写	启用从 Timer2 的溢出中断。0/1：停用/启用
5	0	读/写	启用从 PWMG0 或 EEW_Done 的溢出中断。0/1：停用/启用
4	0	读/写	启用从比较器的溢出中断。0/1：停用/启用
3	0	读/写	启用从 ADC 的溢出中断。0/1：停用/启用
2	0	读/写	启用从 Timer16 的溢出中断。0/1：停用/启用
1	0	读/写	启用从 PB0/PA4/PA3/PA6 的溢出中断。0/1：停用/启用
0	0	读/写	启用从 PA0/PB5/PA2/PA7 的溢出中断。0/1：停用/启用

6.5. 中断请求寄存器(*intrq*), IO 地址 = 0x05

位	初始值	读/写	描述
7	-	读/写	Timer3 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求。
6	-	读/写	Timer2 的中断请求，此位是由硬件置位并由软件清零。 0/1：不要求/请求。
5	-	读/写	PWMG0 或 EEW_Done 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。

6.6. Timer16 控制寄存器(*t16m*), IO 地址 = 0x06

位	初始值	读/写	描述
7 - 5	000	读/写	Timer16 时钟选择： 000：停用 001：CLK（系统时钟） 010：保留 011：PA4 下降沿（从外部引脚） 100：IHRC 101：EOSC 110：ILRC 111：PA0 下降沿（从外部引脚)）
4 - 3	00	读/写	Timer16 时钟分频： 00：÷1 01：÷4 10：÷16 11：÷64
2 - 0	000	读/写	中断源选择。当所选择的状态位变化时，中断事件发生。 0：bit 8 of Timer16 1：bit 9 of Timer16 2：bit 10 of Timer16 3：bit 11 of Timer16 4：bit 12 of Timer16 5：bit 13 of Timer16 6：bit 14 of Timer16 7：bit 15 of Timer16

6.7. 乘法器运算对象寄存器(*mulop*), IO 地址 = 0x08

位	初始值	读/写	描述
7 - 0	-	读/写	硬件乘法运算的运算对象

6.8. 乘法器结果高字节寄存器(*mulrh*), IO 地址 = 0x09

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

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

位	初始值	读/写	描述
7	0	只写	使能外部晶体振荡器。0 / 1：停用/使能。
6 - 5	00	只写	晶体振荡器的选择。 00：保留 01：低驱动电流。适用于较低频率晶体，例如：32KHz 10：中驱动电流。适用于中等频率晶体，例如：1MHz 11：高驱动电流。适用于较高频率晶体，例如：4MHz
4 - 1	-	-	保留。请设为 0。
0	0	只写	将 Bandgap 和 LVR 硬件模块断电。 0 / 1：正常/ 断电。 请注意：Bandgap 关闭后将仅 ILRC/T16/TM2/TM3 及 I/O 功能可用。

6.10. 中断边缘选择寄存器(*integs*), IO 地址 = 0x0c

位	初始值	读/写	描述
7 - 5	-	-	保留。
4	0	只写	Timer16 中断边缘选择： 0：上升缘请求中断。 1：下降缘请求中断。
3 - 2	00	只写	PB0/PA4/PA3/PB6 中断边缘选择： 00：上升缘和下降缘都请求中断。 01：上升缘请求中断。 10：下降缘请求中断。 11：保留。
1 - 0	00	只写	PA0/PB5/PA2/PA7 中断边缘选择： 00：上升缘和下降缘都请求中断。 01：上升缘请求中断。 10：下降缘请求中断。 11：保留。

6.11. 端口 A 数字输入使能寄存器(*padier*), IO 地址 = 0x0d

位	初始值	读/写	描述
7 - 0	0xFF	只写	使能端口 A 数字输入，该对应位作为 AD 输入时，设为 0 可以防止耗电。同时，如果该对应位设为 0，则不能用来唤醒系统。 0 / 1：停用/ 启用

6.12. 端口 B 数字输入使能寄存器(*pbdier*), IO 地址 = 0x0e

位	初始值	读/写	描述
7 - 0	0xFF	只写	使能端口 B 数字输入，该对应位作为 AD 输入时，设为 0 可以防止耗电。同时，如果该对应位设为 0，则不能用来唤醒系统。 0 / 1：停用/ 启用

6.13. 端口 C 数字输入使能寄存器(*pcdier*), IO 地址 = 0x0f

位	初始值	读/写	描述
7 - 0	0xFF	只写	使能端口 C 数字输入，该对应位作为 AD 输入时，设为 0 可以防止耗电。同时，如果该对应位设为 0，则不能用来唤醒系统。 0 / 1：停用/ 启用

注意：详细 *PCDIER* 设定请参考 9.2。

6.14. 端口 A 数据寄存器(*pa*), IO 地址 = 0x10

位	初始值	读/写	描述
7 - 0	0x00	读/写	数据寄存器的端口 A。

6.15. 端口 A 控制寄存器(*pac*), IO 地址 = 0x11

位	初始值	读/写	描述
7 – 0	0x00	读/写	端口 A 控制寄存器。这些寄存器是用来定义端口 A 每个相应的引脚的输入模式或输出模式。 0/1：输入/输出。 请注意：PA5 可设为输入或输出低，当 PA5 设为输出高时，为 OC/OD 输出。

6.16. 端口 A 上拉控制寄存器(*paph*), IO 地址 = 0x12

位	初始值	读/写	描述
7 – 0	0x00	读/写	端口 A 上拉控制寄存器。这些寄存器是用来控制上拉高端口 A 每个相应的引脚，并且只在引脚为输入模式时有效 0/1：停用/启用。

6.17. 端口 B 数据寄存器(*pb*), IO 地址 = 0x13

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

6.18. 端口 B 控制寄存器(*pbc*), IO 地址 = 0x14

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

6.19. 端口 B 上拉控制寄存器(*pbph*), IO 地址 = 0x15

位	初始值	读/写	描述
7 – 0	0x00	读/写	端口 B 上拉控制寄存器。这些寄存器是用来控制上拉高端口 B 每个相应的引脚，并且只在引脚为输入模式时有效 0/1：停用/启用。

6.20. 端口 C 数据寄存器(*pc*), IO 地址 = 0x16

位	初始值	读/写	描述
7 – 0	0x00	读/写	数据寄存器的端口 C。

6.21. 端口 C 控制寄存器(*pbc*), IO 地址 = 0x17

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

6.22. 端口 C 上拉控制寄存器(*pcph*), IO 地址 = 0x18

位	初始值	读/写	描述
7 - 0	0x00	读/写	端口 C 上拉控制寄存器。这些寄存器是用来控制上拉高端口 C 每个相应的引脚，并且只在引脚为输入模式时有效 0/1：停用/启用。

6.23. 端口 A 下拉控制寄存器(*papl*), IO 地址 = 0x19

位	初始值	读/写	描述
7 – 0	0x00	读/写	端口 A 下拉控制寄存器。这些寄存器是用来控制下拉端口 A 每个相应的引脚，并且只在引脚为输入模式时有效 0/1：停用/启用。

6.24. 端口 B 下拉控制寄存器(*pbpl*), IO 地址 = 0x1A

位	初始值	读/写	描述
7 - 0	0x00	读/写	端口 B 下拉控制寄存器。这些寄存器是用来控制下拉端口 B 每个相应的引脚，并且只在引脚为输入模式时有效 0/1：停用/启用。

6.25. 端口 C 下拉控制寄存器(*pcpl*), IO 地址 = 0x1B

位	初始值	读/写	描述
7 - 0	0x00	读/写	端口 C 下拉控制寄存器。这些寄存器是用来控制下拉端口 C 每个相应的引脚，并且只在引脚为输入模式时有效 0/1：停用/启用。

6.26. ADC 控制寄存器(*adcc*), IO 地址 = 0x20

位	初始值	读/写	描述
7	0	读/写	ADC 功能使能。0/1：停用/启用。
6	0	读/写	ADC 转换进程控制位： 写入 ”1” 可开始 AD 转换。 读到 “1” 表明 ADC 已经准备好，或已转换完成。
5 – 2	0000	读/写	通道选择。以下 4 位用来选择 AD 转换的输入信号： 0000：PB0/AD0, 0001：PB1/AD1, 0010：PB2/AD2, 0011：PB3/AD3, 0100：PB4/AD4, 0101：PB5/AD5, 0110：PB6/AD6, 0111：PB7/AD7, 1000：PA3/AD8, 1001：PA4/AD9, 1010：PA0/AD10, 1011：PC1/AD11（仿真器在此位置为 PC1 或 PA1，参考第 5.15.4 节） 1100：PC2/AD12（仿真器在此位置为 PC1 或 PA1，参考第 5.15.4 节） 1111：（通道 F）Bandgap 参考电压或者 0.25*V_{DD} 其他：保留。
0 – 1	-	-	保留（写 0）。

6.27. ADC 模式寄存器(*adcm*), IO 地址 = 0x21

位	初始值	读/写	描述
7 - 4	-	-	保留（写 0）。
3 - 1	000	只写	ADC 时钟源选择： 000：CLK（系统时钟）÷ 1， 001：CLK（系统时钟）÷ 2， 010：CLK（系统时钟）÷ 4， 011：CLK（系统时钟）÷ 8， 100：CLK（系统时钟）÷ 16， 101：CLK（系统时钟）÷ 32， 110：CLK（系统时钟）÷ 64， 111：CLK（系统时钟）÷ 128。
0	-	-	保留。

6.28. ADC 调节控制寄存器(*adcrgc*), IO 地址 = 0x24

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， 110：Bandgap 1.60v 参考电压， 111：Bandgap 2.40v 参考电压。
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 - 1	00	只写	ADC 通道 F 的 Bandgap 参考电压选择：（仿真器固定只有 1.2V） 000：1.2V 010：2V 100：3V 110：4V 001：1.6V 011：2.4V
0	-	-	保留（写 0）。

6.29. ADC 数据高位寄存器(*adcrh*), IO 地址 = 0x22

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。

6.30. ADC 数据低位寄存器(*adcrl*), IO 地址 = 0x23

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。

6.31. 杂项寄存器(*misc*), IO 地址 = 0x26

位	初始值	读/写	描述
7 - 6	-	-	保留。请设为 0。
5	0	W	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 3000 个 ILRC 时钟（不适用快速开机）。 1：快速唤醒。 唤醒时间为 45 个 ILRC 时钟。
4	0	W	使能 VDD/2 bias 电压生成器 0 / 1：停用/ 启用（ICE 仿真时无法动态切换） 如果 Code Option 有选择 LCD 输出，但 MISC.4 没有设为 1，则在 IC 上还是无法输出 VDD/2 bias，但仿真器则恒可以，此处两边现象不同。
3			保留。
2	0	只写	停用 LVR 功能： 0 / 1：启用 / 停用
1 - 0	00	只写	看门狗时钟超时时间设定： 00：8k ILRC 时钟周期 01：16k ILRC 时钟周期 10：64k ILRC 时钟周期 11：256k ILRC 时钟周期

6.32. 比较器控制寄存器(*gpcc*), IO 地址 = 0x2b

位	初始值	读/写	描述
7	0	读/写	启用比较器。 0 / 1：停用/启用。 当此位被设置为启用，请同时设置相应的模拟输入引脚是数字停用，以防止漏电。
6	-	只读	比较器结果。 0：正输入 < 负输入 1：正输入 > 负输入
5	0	读/写	选择比较器的结果是否由 TM2_CLK 采样输出。 0：比较器的结果没有 TM2_CLK 采样输出 1：比较器的结果是由 TM2_CLK 采样输出
4	0	读/写	选择比较器输出的结果是否反极性。 0：比较器输出的结果没有反极性 1：比较器输出的结果是反极性
3 - 1	000	读/写	选择比较器负输入的来源。 000：PA3 001：PA4 010：内部 1.20 V bandgap 参考电压（不适用于比较器唤醒功能） 011：$V_{internal\ R}$ 100：PB6 101：PB7 11X：保留
0	0	读/写	选择比较器正输入的来源。 0：$V_{internal\ R}$ 1：PA4

6.33. 比较器选择寄存器(*gpcs*), IO 地址 = 0x2c

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。
6	-	只写	比较器唤醒启用。（gpcc.6 发生电平变化时才可唤醒） 0/1：停用/启用
5	0	只写	选择比较器参考电压 $V_{internal\ R}$ 最高的范围。
4	0	只写	选择比较器参考电压 $V_{internal\ R}$ 最低的范围。
3 - 0	0000	只写	选择比较器参考电压 $V_{internal\ R}$。 0000（最低）~ 1111（最高）

6.34. Timer2 控制寄存器(*tm2c*), IO 地址 = 0x30

位	初始值	读/写	描述
7 - 4	0000	读/写	Timer2 时钟源选择： 0000：停用 0001：CLK（系统时钟） 0010：IHRC 或 IHRC*2（由 code option TMx_ source 决定） 0011：EOSC 0100：ILRC 0101：比较器输出（仿真器不支持） 1000：PA0（上升沿） 1001：~PA0（下降沿） 1010：PB0（上升沿） 1011：~PB0（下降沿） 1100：PA4（上升沿） 1101：~PA4（下降沿） 其他：保留 **注意**：在 ICE 模式且 IHRC 被选为 Timer2 定时器时钟，当 ICE 停下时，发送到定时器的时钟是不停止，定时器仍然继续计数。
3 - 2	00	读/写	Timer2 输出选择： 00：停用 01：PB2 10：PA3 11：PB4
1	0	读/写	Timer2 模式选择： 0 / 1：定周期模式 / PWM 模式。
0	0	读/写	启用 Timer2 反极性输出： 0 / 1：停用/启用。

6.35. Timer2 计数寄存器(*tm2ct*), IO 地址 = 0x31

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

6.36. Timer2 分频寄存器(*tm2s*), IO 地址 = 0x32

位	初始值	读/写	描述
7	0	只写	PWM 分辨率选择： 0：8 位 1：6 位或 7 位（由 code option TMx_Bit 决定）
6 - 5	00	只写	Timer2 时钟预分频器： 00：÷ 1 01：÷ 4 10：÷ 16 11：÷ 64
4 - 0	00000	只写	Timer2 时钟分频器。

6.37. Timer2 上限寄存器(*tm2b*), IO 地址 = 0x33

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

6.38. Timer3 控制寄存器(*tm3c*), IO 地址 = 0x34

位	初始值	读/写	描述
7 - 4	0000	读/写	Timer3 时钟选择。 0000：停用 0001：CLK（系统时钟） 0010：IHRC 或 IHRC*2 （由 code option TMx_ source 决定） 0011：EOSC 0100：ILRC 0101：比较器输出 1000：PA0（上升沿） 1001：~PA0（下降沿） 1010：PB0（上升沿） 1011：~PB0（下降沿） 1100：PA4（上升沿） 1101：~PA4（下降沿） 其他：保留 **注意**：在 ICE 模式且 IHRC 被选为 Timer3 定时器时钟，当 ICE 停下时，发送到定时器的时钟是不停止，定时器仍然继续计数。
3 - 2	00	读/写	Timer3 输出选择。 00：停用 01：PB5 10：PB6 11：PB7
1	0	读/写	Timer3 模式选择。 0：周期模式 1：PWM 模式

6.39. Timer3 计数寄存器(*tm3ct*), IO 地址 = 0x35

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

6.40. Timer3 分频寄存器(*tm3s*), IO 地址 = 0x36

位	初始值	读/写	描述
7	0	只写	PWM 分辨率选择。 0：8 位 1：6 位或 7 位（由 code option TMx_Bit 决定）
6 - 5	00	只写	Timer3 时钟预分频器。 00：÷ 1 01：÷ 4 10：÷ 16 11：÷ 64
4 - 0	00000	只写	Timer3 时钟分频器。

6.41. Timer3 上限寄存器(*tm3b*), IO 地址 = 0x37

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

6.42. EEPROM 数据寄存器(*eerl*)，IO 地址 = 0x3C

位	初始值	读/写	描述
7 - 0	-	读/写	EEPROM 低字节数据寄存器。

6.43. EEPROM 控制寄存器(*eermc*)，IO 地址 = 0x3D

位	初始值	读/写	描述
7	0x00	读	保留
6			EEPROM 的 busy 信号。 0/1：完成/Busy
5 - 0			保留
7 - 0	0x00	写	在每次 STEER 指令前，通过写入 0x5A 使能 EEPROM Byte 读/写程序
			在每次 STEER 指令前，通过写入 0xA5 使能 EEPROM Erase All

6.44. PWMG0 控制寄存器(*pwmg0c*), IO 地址 = 0x40

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	只写	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 其它：保留
0	0	读/写	PWMG0 时钟源。 0：CLK 1：IHRC 或 IHRC*2（由 Code option PWM_source 决定）

6.45. PWMG0 分频寄存器 (*pwmg0s*), IO 地址 = 0x41

位	初始值	读/写	描述
7	0	读/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	读/写	PWMG0 预分频。 00：÷1 01：÷4 10：÷16 11：÷64
4 - 0	00000	读/写	PWMG0 时钟分频。

6.46. PWMG0 占空比高位寄存器(*pwmg0dth*), IO 地址 = 0x42

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

6.47. PWMG0 占空比低位寄存器(*pwmg0dtl*), IO address = 0x43

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 占空比低位寄存器的值必须写在 PWMG0 占空比高位寄存器之前。

6.48. PWMG0 计数上限高位寄存器(*pwmg0cubh*), IO 地址 = 0x44

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

6.49. PWMG0 计数上限低位寄存器(*pwmg0cubl*), IO 地址 = 0x45

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

6.50. PWMG1 控制寄存器(*pwmg1c*), IO 地址 = 0x46

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。 0：CLK 1：IHRC 或 IHRC*2（由 Code option PWM_source 决定）

6.51. PWMG1 分频寄存器(*pwmg1s*), IO 地址 = 0x47

位	初始值	读/写	描述
7	0	只写	PWMG1 中断模式。 0：当计数为设定的占空比时产生中断 1：当计数为 0 产生中断
6 - 5	00	只写	PWMG1 预分频。 00： ÷1 01： ÷4 10： ÷16 11： ÷64
4 - 0	00000	只写	PWMG1 时钟分频。

6.52. PWMG1 占空比高位寄存器(*pwmg1dth*), IO 地址 = 0x48

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

6.53. PWMG1 占空比低位寄存器(*pwmg1dtl*), IO 地址 = 0x49

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

注意：PWMG1 占空比低位寄存器的值必须写在 PWMG1 占空比高位寄存器之前。

6.54. PWMG1 计数上限高位寄存器(*pwmg1cubh*), IO 地址 = 0x4a

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

6.55. PWMG1 计数上限低位寄存器(*pwmg1cubl*), IO 地址 = 0x04b

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

6.56. PWMG2 控制寄存器(*pwmg2c*), IO 地址 = 0x4C

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	只写	选择 PWMG2 的输出的结果是否反极性： 0/1：停用/启用。
4	0	读/写	PWMG2 计数器清零。写“1”清零 PWMG2 计数，清零 PWMG2 计数后，这个位会自动归 0。
3 - 1	000	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 IHRC*2（由 Code option PWM_source 决定）

6.57. PWMG2 分频寄存器 (*pwmg2s*), IO 地址 = 0x4D

位	初始值	读/写	描述
7	0	只写	PWMG2 中断模式。 0：当计数为设定的占空比时产生中断 1：当计数为 0 产生中断
6 - 5	00	只写	PWMG2 预分频。 00：÷1 01：÷4 10：÷16 11：÷64
4 - 0	00000	只写	PWMG2 时钟分频。

6.58. PWMG2 占空比高位寄存器(*pwmg2dth*), IO 地址 = 0x4E

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

6.59. PWMG2 占空比低位寄存器(*pwmg2dtl*), IO 地址 = 0x4F

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

注意：PWMG2 占空比低位寄存器的值必须写在 PWMG2 占空比高位寄存器之前。

6.60. PWMG2 计数上限高位寄存器(*pwmg2cubh*), IO 地址 = 0x50

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

6.61. PWMG2 计数上限低位寄存器(*pwmg2cubl*), IO 地址 = 0x51

位	初始值	读/写	描述
7 - 6	000	只写	PWMG2 上限寄存器位[2:1]。
5 - 0	-	-	保留。

7. 指令

符号	描述
ACC	累加器（Accumulator 的缩写）
a	累加器（Accumulator 在程序里的代表符号）
sp	堆栈指针
flag	ACC 标志寄存器
I	立即数据
&	逻辑与
|	逻辑或
←	移动
^	异或
+	加
—	减
～	按位取反（逻辑补数，1 补数）
∓	负数（2 补码）
OV	溢出（2 补数系统的运算结果超出范围）
Z	零（如果零运算单元操作的结果是 0，这位设置为 1）
C	进位(Carry)
AC	辅助进位标志(Auxiliary Carry)
M.n	只允许寻址在地址 0x00~0x7F (0~127) 的位置

7.1. 数据传输类指令

mov a, I	移动即时数据到累加器 例如：*mov* a, 0x0f; 结果：a ← 0fh; 受影响标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
mov M, a	移动数据由累加器到存储器 例如：*mov* MEM, a; 结果：MEM ← a 受影响标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
mov a, M	移动数据由存储器到累加器 例如：*mov* a, MEM; 结果：a ← MEM; 当 MEM 为零时，标志位 Z 会被置位。 受影响标志位：Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
mov a, IO	移动数据由 IO 到累加器 例如：*mov* a, pa ; 结果：a ← pa; 当 pa 为零时，标志位 Z 会被置位。 受影响标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
mov IO, a	移动数据由累加器到 IO 例如：*mov* pb, a; 结果：pb ← a 受影响标志位：Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
nmov M, a	将累加器的负逻辑(2 补码)放入 RAM 例如： *mov* MEM, a; 结果： MEM ← 〒a 受影响标志位：Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』 应用范例： --- *mov* a, 0xf5 ; // ACC is 0xf5 *nmov* ram9, a; // ram9 is 0x0b, ACC is 0xf5 ---
nmov a, M	将 RAM 的负逻辑(2 补码)放入累加器 例如： *mov* a, MEM ; 结果： a ← 〒MEM; Flag Z is set when 〒MEM is zero. 受影响标志位：Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』 应用范例： --- *mov* a, 0xf5 ; *mov* ram9, a ; // ram9 is 0xf5 *nmov* a, ram9 ; // ram9 is 0xf5, ACC is 0x0b ---

ldtabh index

使用索引作为 MTP 的地址并将 MTP 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 ← {bit15~8 of MTP [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 MTP 的指针
…
mov     a, la@TableA ;  // 指定 MTP TableA 指针（LSB）
mov     lb@ROMptr, a ;  // 将指针存到 RAM （LSB）
mov     a, ha@TableA ;  // 指定 MTP 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 MTP 的地址并将 MTP 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 MTP [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 MTP 的指针
…
mov     a, la@TableA ;  // 指定 MTP TableA 指针（LSB）
mov     lb@ROMptr, a ;  // 将指针存到 RAM （LSB）
mov     a, ha@TableA ;  // 指定 MTP 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 ;
```

steer index

使用索引作为EEPROM的地址并将IO寄存器eerl(和eerh)的字节(Word)数据存储入EEPROM。启动 EEPROM 编译进程。

例如：*steer* index;

结果：EEPROM[index] ← {(eerh,)eerl};

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

应用范例：

```
word        point    =  0;
eerl    =   0x12;
eerh    =   0x34;
steer       point;
```

ldeer index

使用索引作为 EEPROM 的地址并将 EEPROM 中的低字节数据读取并载入到 IO 寄存器 eerl(和 eerh)。启动 EEPROM 数据读取进程。

例如：*ldeer* inde ;

结果：{(eerh,)eerl} ← EEPROM[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        point    =  0;
ldeer       point;
```

ldt16 word

将 Timer16 的 16 位计算值复制到 RAM。

例如：*ldt16* word;

结果：word ← 16-bit timer

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

应用范例：

```
 word       T16val ;          // 定义一个 RAM word
…
clear       lb@T16val ;       // 清零 T16val (LSB)
clear       hb@T16val ;       // 清零 T16val (MSB)
stt16       T16val ;          // 设定 Timer16 的起始值为 0
…
set1        t16m.5 ;          // 启用 Timer16
…
set0        t16m.5 ;          // 停用 Timer16
ldt16       T16val ;          // 将 Timer16 的 16 位计算值复制到 RAM T16val
….
```

stt16 word	将放在 word 的 16 位 RAM 复制到 Timer16。 例如：*stt16* word; 结果：word ← 16-bit timer 受影响标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
	应用范例： --- word T16val ; // 定义一个 RAM word … *mov* a, 0x34 ; *mov* lb@T16val, a ; // 将 0x34 搬到 T16val (LSB) *mov* a, 0x12 ; *mov* hb@T16val, a ; // 将 0x12 搬到 T16val (MSB) *stt16* T16val ; // Timer16 初始化 0x1234 … ---
idxm a, index	使用索引作为 RAM 的地址并将 RAM 的数据读取并载入到累加器。它需要 2T 时间执行这一指令。 例如：*idxm* a, index; 结果：a ← [index], index 是用 word 定义。 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』 应用范例： --- word RAMIndex ; // 定义一个 RAM 指针 … *mov* a, 0x5B ; // 指定指针地址 (LSB) *mov* lb@RAMIndex, a ; // 将指针存到 RAM (LSB) *mov* a, 0x00 ; // 指定指针地址为 0x00 (MSB)，在 PMS133/134 要为 0 *mov* hb@RAMIndex, a ; // 将指针存到 RAM (MSB) … *idxm* a, RAMIndex ; // 将 RAM 地址为 0x5B 的数据读取并载入累加器 ---
Idxm index, a	使用索引作为 RAM 的地址并将累加器的数据读取并载入到 RAM。它需要 2T 时间执行这一指令。 例如：*idxm* index, a; 结果：[index] ← a; index 是以 word 定义。 受影响的标志位：Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』 应用范例： --- word RAMIndex ; // 定义一个 RAM 指针 … *mov* a, 0x5B ; // 指定指针地址 (LSB) *mov* lb@RAMIndex, a ; // 将指针存到 RAM (LSB) *mov* a, 0x00 ; // 指定指针地址为 0x00 (MSB)，在 PMS133/134 要为 0 *mov* hb@RAMIndex, a ; // 将指针存到 RAM (MSB) … *mov* a, 0Xa5 ; *idxm* RAMIndex, a ; // 将累加器数据读取并载入地址为 0x5B 的 RAM ---

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

7.2. 算数运算类指令

add a, I	将立即数据与累加器相加，然后把结果放入累加器 例如：*add* a, 0x0f; 结果：a ← a + 0fh 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
add a, M	将 RAM 与累加器相加，然后把结果放入累加器 例如：*add* a, MEM; 结果：a ← a + MEM 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
add M, a	将 RAM 与累加器相加，然后把结果放入 RAM 例如：*add* MEM, a; 结果：MEM ← a + MEM 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
addc a, M	将 RAM、累加器以及进位相加，然后把结果放入累加器 例如：*addc* a, MEM; 结果：a ← a + MEM + C 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
addc M, a	将 RAM、累加器以及进位相加，然后把结果放入 RAM 例如：*addc* MEM, a; 结果：MEM ← a + MEM + C 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
addc a	将累加器与进位相加，然后把结果放入累加器 例如：*addc* a; 结果：a ← a + C 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
addc M	将 RAM 与进位相加，然后把结果放入 RAM 例如：*addc* MEM; 结果：MEM ← MEM + C 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
nadd a, M	将累加器的负逻辑(2补码)与RAM相加，然后把结果放入累加器 例如：*nadd* a, MEM; 结果：a ← ∓a + MEM 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
nadd M, a	将RAM的负逻辑（2补码）与累加器相加，然后把结果放入RAM 例如：*nadd* MEM, a; 结果：MEM ← ∓MEM + a 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
sub a, I	累加器减 RAM，然后把结果放入累加器 例如：*sub* a, 0x0f; 结果：a ← a - 0x0f (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, MEN; 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，再减进位，然后把结果放入 RAM 例如：*subc* 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* a; 结果：MEN ← MEN - 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：MEN ← 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』
mul	乘法运算， 8x8 无符号乘法执行指令 例如：*mul* ; 结果：{MulRH,ACC} ← ACC * MulOp 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, 0x5a ; mov mulop, a ; mov a, 0xa5 ; mul // 0x5A * 0xA5 = 3A02 (mulrh + ACC) mov ram0, a ; // LSB, ram0=0x02 mov a, mulrh ; // MSB, ACC=0X3A … ---

7.3. 移位运算类指令

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

7.4. 逻辑运算类指令

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

xor M, a	累加器和 RAM 执行逻辑 XOR，然后把结果保存到 RAM 例如： *xor* MEM, a ; 结果： MEM ← a ^ MEM 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
not a	累加器执行 1 补码运算，结果放在累加器 例如：*not* a ; 结果：a ← ～a 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』 应用范例： --- *mov* a, 0x38 ; // ACC=0X38 *not* a ; // ACC=0XC7 ---
not M	RAM 执行 1 补码运算，结果放在 RAM 例如：*not* MEM ; 结果：MEM ← ～MEM 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』 应用范例： --- *mov* a, 0x38 ; *mov* mem, a ; // mem = 0x38 *not* mem ; // mem = 0xC7 ---
neg a	累加器执行 2 补码运算，结果放在累加器 例如：*neg* a; 结果：a ←a 的 2 补码 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』 应用范例： --- *mov* a, 0x38 ; // ACC=0X38 *neg* a ; // ACC=0XC8 ---
neg M	RAM 执行 2 补码运算，结果放在 RAM 例如：*neg* MEM; 结果：MEM ← MEM 的 2 补码 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』 应用范例： --- *mov* a, 0x38 ; *mov* mem, a ; // mem = 0x38 *not* mem ; // mem = 0xC8 ---
comp a, I	比较累加器与立即数 例如：*comp* *a, 0x55;* 结果：等效于(a – 0x55)，并改变标志位 Flag。

	受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』 应用范例： --- *mov a, 0x38 ;* *comp a, 0x38 ; // Z 标志被设为 1* *comp* a, 0x42 ; // C *标志被设为 1* *comp* a, 0x24 ; // C, Z *标志被清除* *comp* a, 0x6a ; // C, AC *标志被设为 1* ---
comp a, M	比较累加器和 RAM 的内容 例如：*comp a, MEM;* 结果：等效于(a - MEM)，并改变标志位 Flag。 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』 应用范例： --- *mov a, 0x38 ;* *mov mem, a ;* *comp a, mem ; // Z 标志被设为 1* *mov a, 0x42 ;* *mov mem, a ;* *mov a, 0x38 ;* *comp a, mem ; // C 标志被设为 1* ---
comp M, a	比较累加器和 RAM 的内容 例如：*comp MEM, a;* 结果：等效于(MEM - a)，并改变标志位 Flag。 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』

7.5. 位运算类指令

set0 IO.n	IO 口的位 N 拉低电位 例如： *set0* pa.5 ; 结果： PA5=0 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
set1 IO.n	IO 口的位 N 拉高电位 例如：*set1* 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 口的位 N 电平变换 例如：*tog* pa.5 ; 结果： 切换 PA.5 的电平状态 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
swapc IO.n	IO 口的位 N 与进位 C 交换 受影响的标志位: 『不变』Z 『受影响』C 『不变』AC 『不变』OV 应用范例 1（连续输出）： ... set1 pac.0 ; // 设置 PA.0 作为输出 ... set0 flag.1 ; // C=0 swapc pa.0 ; // 送 C 给 PA.0 (位操作), PA.0=0 set1 flag.1 ; // C=1 swapc pa.0 ; // 送 C 给 PA.0 (位操作), PA.0=1 ... 应用范例 2（连续输入）： ... set0 pac.0 ; // 设置 PA.0 作为输入 ... swapc pa.0 ; // 读 PA.0 的值给 C（位操作） src a ; // 把 C 移位给 ACC 的位 7 swapc pa.0 ; // 读 PA.0 的值给 C（位操作） src a ; // 把新进 C 移位给 ACC 的位 7，上一个 PA.0 的值给 ACC 的位 6 ... ----------
set0 M.n	RAM 的位 N 设为 0 例如：*set0* MEM.5 ; 结果： MEM 位 5 为 0 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
set1 M.n	RAM 的位 N 设为 1 例如： *set1* MEM.5 ; 结果： MEM 位 5 为 1 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

7.6. 条件运算类指令

ceqsn a, I	比较累加器与立即数据，如果是相同的，即跳过下一指令。标志位的改变与 (a ← a - I)相同 例如： *ceqsn* a, 0x55 ; *inc* MEM ; *goto* error ; 结果： 假如 a=0x55, then “goto error”；否则, “inc MEM”. 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
ceqsn a, M	比较累加器与 RAM，如果是相同的，即跳过下一指令。标志位改变与 (a ← a - M)相同 例如：*ceqsn* a, MEM; 结果：假如 a=MEM，跳过下一个指令 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
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 (a ← a - M)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	比较累加器和 RAM 的值，如果不相等就跳过下一条指令。标志改变与(a ← a - M)相同 例如：*cneqsn a, MEM;* 结果：如果 a≠MEM, 跳到下一条指令 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
cneqsn a, I	比较累加器和立即数的值，如果不相等就跳过下一条指令。标志改变与(a ← a - I)相同 例如： *cneqsn a,0x55 ;* *inc MEM ;* *goto error ;* 结果：如果 a≠0x55, 然后 “goto error”；否则, “inc MEM”. 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
t0sn IO.n	如果 IO 的指定位是 0，跳过下一个指令。 例如：*t0sn* pa.5; 结果：如果 PA5 是 0，跳过下一个指令。 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
t1sn IO.n	如果 IO 的指定位是 1，跳过下一个指令。 例如：*t1sn* pa.5 ; 结果：如果 PA5 是 1，跳过下一个指令。 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
t0sn M.n	如果 RAM 的指定位是 0，跳过下一个指令。 例如：*t0sn* MEM.5 ; 结果：如果 MEM 的位 5 是 0，跳过下一个指令。 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
t1sn M.n	如果 RAM 的指定位是 1，跳过下一个指令。 例如：*t1sn* MEM.5 ; 结果：如果 MEM 的位 5 是 1，跳过下一个指令。 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

izsn a	累加器加 1，若累加器新值是 0，跳过下一个指令。 例如：*izsn* a; 结果：a ← a + 1，若 a=0，跳过下一个指令。 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
dzsn a	累加器减 1，若累加器新值是 0，跳过下一个指令。 例如：*dzsn* a; 结果：a ← a - 1，若 a=0，跳过下一个指令。 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
izsn M	RAM 加 1，若 RAM 新值是 0，跳过下一个指令。 例如：*izsn* MEM; 结果：MEM ← MEM + 1，若 MEM=0，跳过下一个指令。 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
dzsn M	RAM 减 1，若 RAM 新值是 0，跳过下一个指令。 例如：*dzsn* MEM; 结果：MEM ← MEM - 1，若 MEM=0，跳过下一个指令。 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』

7.7. 系统控制类指令

call label	函数调用，地址可以是全部空间的任一地址 例如： *call* function1; 结果： [sp] ← pc + 1 pc ← function1 sp ← sp + 2 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
goto label	转到指定的地址，地址可以是全部空间的任一地址 例如： *goto* error; 结果： 跳到 error 并继续执行程序 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
ret I	将立即数据复制到累加器，然后返回 例如： *ret* 0x55; 结果： A ← 55h ret ; 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
ret	从函数调用中返回原程序 例如： *ret;* 结果： sp ← sp - 2 pc ← [sp] 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
reti	从中断服务程序返回到原程序。在这指令执行之后，全部中断将自动启用 例如： *reti*; 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
nop	没任何动作 例如： *nop*; 结果： 没任何改变 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

pcadd a	目前的程序计数器加累加器是下一个程序计数器 例如： *pcadd a*; 结果： pc ← pc + a 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』 应用范例： --- … mov a, 0x02 ; pcadd a ; // PC <- PC+2 goto err1 ; goto correct ; // 跳到这里 goto err2 ; goto err3 ; … correct: // 跳到这里 … ---
engint	允许全部中断 例如：*engint*; 结果：中断要求可送到 FPP0，以便进行中断服务 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
disgint	停止全部中断 例如：*disgint* ; 结果：送到 FPP0 的中断要求全部被挡住，无法进行中断服务 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
stopsys	系统停止 例如：*stopsys*; 结果：停止系统时钟和关闭系统 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
stopexe	CPU 停止。所有震荡器模块仍然继续工作并输出：但是系统时钟是被停用以节省功耗 例如：*stopexe*; 结果：停住系统时钟，但是仍保持震荡器模块工作 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
reset	复位整个单片机，其运行将与硬件复位相同 例如：*reset*; 结果：复位整个单片机 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
wdreset	复位看门狗 例如：*wdreset* ; 结果：复位看门狗 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

7.8. 指令执行周期综述

2 个周期		*goto, call, idxm, pcadd, ret, reti, ldtabl, ldtabh*
2 个周期	条件满足	*ceqsn,* cneqsn,*t0sn, t1sn, dzsn, izsn*
1 个周期	条件不满足	
1 个周期		其他

7.9. 指令影响标志综述

指令	Z	C	AC	OV	指令	Z	C	AC	OV	指令	Z	C	AC	OV
mov a, I	-	-	-	-	*mov* M, a	-	-	-	-	*mov* a, M	Y	-	-	-
mov a, IO	Y	-	-	-	*mov* IO, a	-	-	-	-	*nmov* M, a	-	-	-	-
nmov a, M	Y	-	-	-	*ldtabh* index	-	-	-	-	*ldtabl* index	-	-	-	-
ldt16 word	-	-	-	-	*stt16* word	-	-	-	-	*idxm* a, index	-	-	-	-
idxm index, a	-	-	-	-	*xch* M	-	-	-	-	*pushaf*	-	-	-	-
popaf	Y	Y	Y	Y	*add* a, I	Y	Y	Y	Y	*add* a, M	Y	Y	Y	Y
add M, a	Y	Y	Y	Y	*addc* a, M	Y	Y	Y	Y	*addc* M, a	Y	Y	Y	Y
addc a	Y	Y	Y	Y	*addc* M	Y	Y	Y	Y	*sub* a, I	Y	Y	Y	Y
sub a, M	Y	Y	Y	Y	*sub* M, a	Y	Y	Y	Y	*subc* a, M	Y	Y	Y	Y
subc M, a	Y	Y	Y	Y	*subc* a	Y	Y	Y	Y	*subc* M	Y	Y	Y	Y
inc M	Y	Y	Y	Y	*dec* M	Y	Y	Y	Y	*clear* M	-	-	-	-
sr a	-	Y	-	-	*src* a	-	Y	-	-	*sr* M	-	Y	-	-
src M	-	Y	-	-	*sl* a	-	Y	-	-	*slc* a	-	Y	-	-
sl M	-	Y	-	-	*slc* M	-	Y	-	-	*swap* a	-	-	-	-
swap M	-	-	-	-	*and* a, I	Y	-	-	-	*and* a, M	Y	-	-	-
and M, a	Y	-	-	-	*or* a, I	Y	-	-	-	*or* a, M	Y	-	-	-
or M, a	Y	-	-	-	*xor* a, I	Y	-	-	-	*xor* *a, IO*	Y	-	-	-
xor IO, a	-	-	-	-	*xor* a, M	Y	-	-	-	*xor* M, a	Y	-	-	-
not a	Y	-	-	-	*not* M	Y	-	-	-	*neg* a	Y	-	-	-
neg M	Y	-	-	-	*comp* a, I	Y	Y	Y	Y	*comp* a, M	Y	Y	Y	Y
comp M, a	Y	Y	Y	Y	*set0* IO.n	-	-	-	-	*set1* IO.n	-	-	-	-
tog IO.n	-	-	-	-	*set0* M.n	-	-	-	-	*set1* M.n	-	-	-	-
swapc IO.n	-	Y	-	-	*ceqsn* a, I	Y	Y	Y	Y	*ceqsn* a, M	Y	Y	Y	Y
ceqsn M, a	Y	Y	Y	Y	*cneqsn* *a, I*	Y	Y	Y	Y	*cneqsn a, M*	Y	Y	Y	Y
t0sn IO.n	-	-	-	-	*t1sn* IO.n	-	-	-	-	*t0sn* M.n	-	-	-	-
t1sn M.n	-	-	-	-	*izsn* a	Y	Y	Y	Y	*dzsn* a	Y	Y	Y	Y
izsn M	Y	Y	Y	Y	*dzsn* M	Y	Y	Y	Y	*call* label	-	-	-	-
goto label	-	-	-	-	*ret* I	-	-	-	-	*ret*	-	-	-	-
reti	-	-	-	-	*nop*	-	-	-	-	*pcadd* a	-	-	-	-
engint	-	-	-	-	*disgint*	-	-	-	-	*stopsys*	-	-	-	-
stopexe	-	-	-	-	*reset*	-	-	-	-	*wdreset*	-	-	-	-

指令	Z	C	AC	OV	指令	Z	C	AC	OV	指令	Z	C	AC	OV
nadd a, M	Y	Y	Y	Y	*nadd* M, a	Y	Y	Y	Y	*steer* index	-	-	-	-
ldeer index	-	-	-	-	*mul*	-	-	-	-					

7.10. BIT 定义

位寻址只能定义在 RAM 区地址的 0x00 to 0x7F。

8. 代码选项(Code Options)

配置	选项	描述
Security	**Enable**	**启用内容加密**
	Disable	不启用内容加密
LVR	4.0V	选择 LVR = 4.0V
	3.5V	选择 LVR = 3.5V
	3.0V	选择 LVR = 3.0V
	2.7V	选择 LVR = 2.7V
	2.5V	**选择 LVR = 2.5V**
	2.2V	选择 LVR = 2.2V
	2.0V	选择 LVR = 2.0V
	1.8V	选择 LVR = 1.8V
Boot-up_Time	Slow	请参考第 4.1 节 t_{WUP} 和 t_{SBP}
	Fast	**请参考第 4.1 节 t_{WUP} 和 t_{SBP}**
LCD2 （请参考 MISC.4 设定）	PB0_A034	VDD/2 LCD 偏置电压产生器启用，如果为输入模式, PB0, PA[0,3,4]为 VDD/2
	PB1256	VDD/2 LCD 偏置电压产生器启用，如果为输入模式, PB[1,2,5,6]为 VDD/2
Interrupt Src0	**PA.0**	**INTEN/INTRQ.0 源自 PA.0 中断**
	PB.5	INTEN/INTRQ.0 源自 PB.5 中断
	PA.2	INTEN/INTRQ.0 源自 PA.2 中断
	PA.7	INTEN/INTRQ.0 源自 PA.7 中断
Interrupt Src1	**PB.0**	**INTEN/INTRQ.1 源自 PB.0 中断**
	PA.4	INTEN/INTRQ.1 源自 PA.4 中断
	PA.3	INTEN/INTRQ.1 源自 PA.3 中断
	PB.6	INTEN/INTRQ.1 源自 PB.6 中断

配置	选项	描述
PB4_PB7_Drive	Normal	PB4 与 PB7 的驱动/灌电流为 Normal
	Strong	**PB4 与 PB7 的驱动/灌电流为 Strong**
Comparator Edge	**All_Edge**	**比较器在上升/下降缘都会触发中断**
	Rising_Edge	比较器在上升缘触发中断
	Falling_Edge	比较器在下降缘触发中断
GPC_PWM	**Disable**	**GPC 不会控制全部的 PWM 输出**
	Enable	GPC 会控制全部的 PWM 输出（仿真器不支持）
PWM_Source	**16MHZ**	**当 Pwmg0c.0= 1 时，PWMG0 时钟源 = IHRC = 16MHZ** **当 Pwmg1c.0= 1 时，PWMG1 时钟源 = IHRC = 16MHZ** **当 Pwmg2c.0= 1 时，PWMG2 时钟源 = IHRC = 16MHZ**
	32MHZ	当 Pwmg0c.0= 1 时，PWMG0 时钟源 = IHRC*2 = 32MHZ 当 Pwmg1c.0= 1 时，PWMG1 时钟源 = IHRC*2 = 32MHZ 当 Pwmg2c.0= 1 时，PWMG2 时钟源 = IHRC*2 = 32MHZ （仿真器不支持）
TMx_Source	**16MHZ**	**当 tm2c[7:4]= 0010 时，TM2 的时钟源 = IHRC = 16MHZ** **当 tm3c[7:4]= 0010 时，TM3 的时钟源 = IHRC = 16MHZ**
	32MHZ	当 tm2c[7:4]= 0010 时，TM2 的时钟源 = IHRC*2 = 32MHZ 当 tm3c[7:4]= 0010 时，TM3 的时钟源 = IHRC*2 = 32MHZ （仿真器不支持）
TMx_Bit	**6 Bit**	**当 tm2s.7=1 时，TM2 的 PWM 分辨率为 6 位** **当 tm3s.7=1 时，TM3 的 PWM 分辨率为 6 位**
	7 Bit	当 tm2s.7=1 时，TM2 的 PWM 分辨率为 7 位 当 tm3s.7=1 时，TM3 的 PWM 分辨率为 7 位 （仿真器不支持）

注：使用**黑粗**字体的为预置选项 (default options)

9. 特别注意事项

此章节提醒用户在使用 PGS134 系列 IC 时避免常犯的一些错误。

9.1. 使用 IC

9.1.1. IO 引脚的使用和设定

(1) IO 作为数字输入

- IO 作为数字输入时，Vih 与 Vil 的准位，会随着电压与温度变化，请遵守 Vih 的最小值，Vil 的最大值规范。
- 内部上拉电阻值也将随着电压、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

- 设置 IO 为输入
- 用 PXDIER 寄存器将对应位设为“1”。

(3) PA5 设置为输出引脚

- PA5 只能做 Open Drain 输出，输出高需要加上拉电阻。

(4) PA5 设置为 PRSTB 输入引脚

- 设定 PA5 作输入。
- 设定 CLKMD.0=1 来启用 PA5 作为 PRSTB 输入引脚。

(5) PA5 作为输入并通过长导线连接至按键或者开关

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

(6) PA7 和 PA6 作为外部晶体振荡器。

- PA7 和 PA6 设定为输入。
- PA7 和 PA6 内部上拉电阻设为关闭。
- 用 PADIER 寄存器将 PA6 和 PA7 设为模拟输入。
- EOSCR 寄存器位[6:5]选择对应的晶体振荡器频率：
  - 01 ： 低频，例如：32KHz
  - 10 ： 中频，例如：455KHz、1MHz
  - 11 ： 高频，例如：4MHz
- 设置 EOSCR.7 =1 启用晶体振荡器。
- 从 IHRC 或 ILRC 切换到 EOSC，要先确认 EOSC 已经稳定振荡。

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

9.1.2. 中断

(1) 当使用中断功能的一般步骤如下：

步骤 1：设定 INTEN 寄存器，开启需要的中断的控制位

步骤 2：清除 INTRQ 寄存器

步骤 3：主程序中，使用 ENGINT 指令允许 CPU 的中断功能
步骤 4：等待中断。中断发生后，跳入中断子程序
步骤 5：当中断子程序执行完毕，返回主程序

* 在主程序中，可使用 DISGINT 指令关闭所有中断
* 跳入中断子程序处理时，可使用 PUSHAF 指令来保存 ALU 和 FLAG 寄存器数据，并在 RETI 之前，使用 PUSHAF 指令复原，步骤如下：

```
void Interrupt (void)     //中断发生后，跳入中断子程序
{                         //自动进入 DISGINT 的状态，CPU 不会再接受中断
    PUSHAF;
    …
    POPAF;
}    //系统自动填入 RETI，直到执行 RETI 完毕才自动恢复到 ENGINT 的状态
```

(2) INTEN，INTRQ 没有初始值，所以要使用中断前，一定要根据需要设定数值。

(3) 外部 IO 脚中断源有两组，每组由 code option Interrupt Src0 和 Interrupt Src1 来决定实际对应的唯一中断接脚；选中的 IO 接脚，需遵守 ***inten/ intrq/ integs*** 寄存器的规范。

9.1.3. 系统时间选择

利用 CLKMD 寄存器可切换系统时钟源。**请注意，不可在切换系统时钟源的同时把原时钟源关闭**。例如：从 A 时钟源切换到 B 时钟源时，应该先用 CLKMD 寄存器切换系统时钟源，然后再通过 CLKMD 寄存器关闭 A 时钟振荡源。

◆ 例子：系统时钟从 ILRC 切换到 IHRC/2

```
CLKMD   =   0x36;          // 切到 IHRC，但 ILRC 不要停用
CLKMD.2 =   0;             // 此时才可关闭 ILRC
```

◆ **错误**：ILRC 切换到 IHRC，同时关闭 ILRC

```
CLKMD   =   0x50;          // MCU 会死机
```

9.1.4. 看门狗

当 ILRC 关闭时，看门狗也会失效。

9.1.5. TIMER 溢出

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

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

9.1.6. IHRC

(1) 当 IC 在烧录器烧录时，会校准 IHRC 频率。
(2) 由于 EMC 的特性或者在 IC 封装或 COB 时，会不同程度影响 IHRC 频率。如果频率校准在 IC 封塑之前已经完成，那么实际的 IHRC 频率会在 IC 封塑之后有可能出现偏差或者超出规格指标。通常情况下该频率会稍稍变慢。
(3) 通常在 COB 封胶或 QTP 时会发生如上描述的情况，应广科技不负任何责任。
(4) 用户可以根据使用经验来做频率补偿，例如，用户可以在使用时调高 IHRC 频率约 0.5%~1%，以便得到比 IC 封塑之后更好的 IHRC 频率。

9.1.7. LVR

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

下面是工作频率、电源电压和 LVR 水平设定的建议：

系统时钟	VDD	LVR
2MHz	≧ 2.2V	≧ 2.2V
4MHz	≧ 2.5V	≧ 2.5V
8MHz	≧ 3.5V	≧ 3.5V

表 9：LVR 设置参考

(1) 只有当 IC 正常起动后，设定 LVR（1.8V ~ 4.0V）才会有效。
(2) 可以设定寄存器 MISC.2 为 1 将 LVR 关闭，但此时应确保 V_{DD} 在 chip 最低工作电压以上，否则 IC 可能工作不正常。
(3) 在省电模式 stopexe 和掉电模式 stopsys 下，LVR 功能无效。

9.1.8. 烧录方法

- 请使用 PDK5S-P-003(B)进行烧录。PDK3S-P-002 以及之前的烧录器皆不支持 PGS134 的烧录。
- PGS134 的烧录脚为 PA4，PA5，PA6，VDD，GND 这 5 只引脚。
- 当合封（MCP）或在板烧录时，关于电压与电流的特别注意事项：
  (1) PA5 (VPP) 可能高于 5.5V。
  (2) VDD 可能高于 5.5V，且其最大电流可能到达 20mA。
  (3) 其他烧录引脚（GND 除外）与 VDD 相同。
  (4) 当用户使用 MCP 或在板烧录时，用户需要确认，外围电路或元件不会被上述电压所破坏或限制上述电压的产生。

- 重要提示：

(1) 如在 handler 上对 IC 进行烧录，请务必按照 APN004 及 APN011 的指示进行。

(2) 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。

请用户依据实际情况选择下述烧录模式。

普通烧录模式

Jumper 连接：可依照烧录器软件上的说明，连接 jumper 即可。请参考烧录器“PDK5S-P-003”的用户手册，了解当使用 PDK5S-P-003(B) 烧录时使用的 JP7 跳线方式。

注意：除了 ICVPP(PA5)需要通过 JP7 跳线与 IC 的 PA5 短接外, ICVPP2(PA3)也必须与 IC 的 PA5 短接。

在板烧录模式

所谓在板烧录，是指 IC 及其他周边电路及组件，皆已经焊接到 PCB 上，并对 IC 进行烧录的情况。在板烧录需要使用 PDK5S-P-003 上五根引线：ICPCK(PA4)、ICPDA(PA6)、VDD、GND、ICVPP(PA5)、ICVPP2(PA3)。用于与 IC 上的 PA3、PA6、VDD、GND 和 PA5（两线）对应相连。为实现在板烧录，PA5 脚的写入信号必须设置为输入模式，且 Code Option 中的 ISP 设置必须是开启的。(无此 code option)

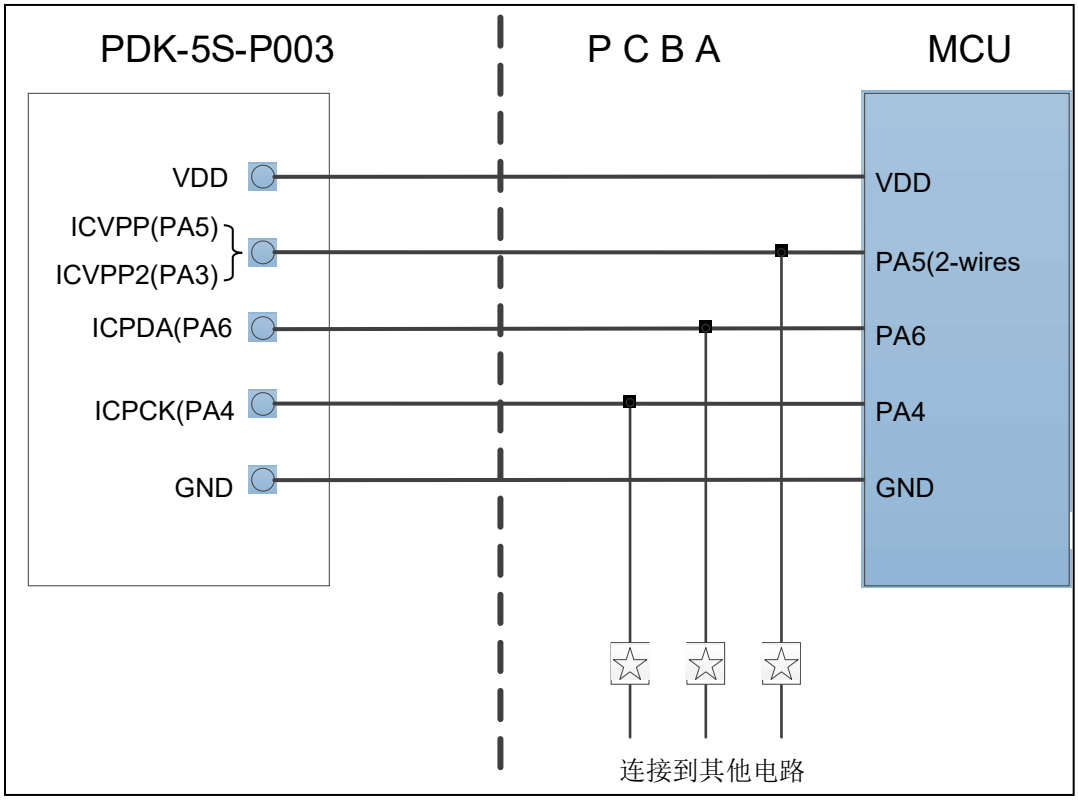


图 23：在板烧录原理图

图 23 中的‘☆’可代表电容或电阻。用于隔离烧录引线和其他电路。电阻应≧10KΩ，电容应≦220pF。

注意：

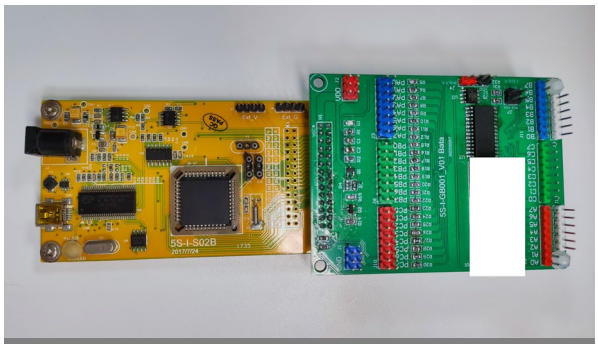
- 一般来说，在板烧录应使用限压烧录模式。请参考在限压烧录模式的详细说明。
- PCB 上的 VDD 与 GND 之间**不可**连接有 5.0V 或以下的稳压二极管或其他钳制 5.0V 产生的电路或元件。
- PCB 上的 VDD 与 GND 之间**不可**连接有标值 500uF 或以上的电容器。
- 一般来说，用于烧录讯号的 PA3，PA5 及 PA6 引脚，**不能**作为强输出脚。

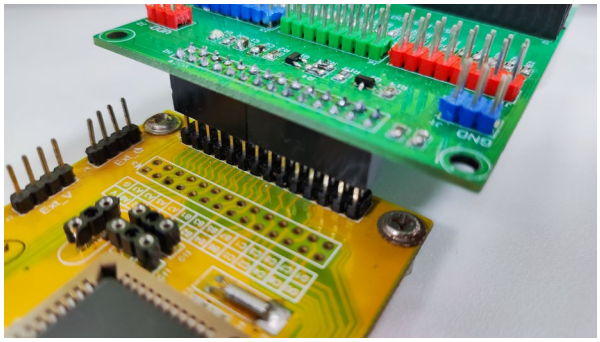
自供电模式下的在板烧录：

PGS134 同样支持由客户电路板供电的方式进行在板烧录。烧录器将在烧录前侦测客户电路板上的 VDD 电压。所以请保证电路板上的 VDD 电压高于 2.5V。其余与在板烧录模式相同。

9.2. 使用 ICE

仿真 PGS134 请使用 PDK5S-I-S01/2(B)外挂仿真板 5S-I-GB001，参考如下图片:





仿真时请注意以下事项：

(1) 关于仿真时序

- 仿真时，尤其是仿真 EEPROM 和 PxPL 相关功能时，PDK5S-I-S01/2(B)会和仿真板 5S-I-GB001 进行通信，所以仿真时许会略慢于实际 IC，主要影响以下寄存器配置 EERL/EERMC/STEER/LDEER/MISC2/INTRQ/INTRQ2/PAPL/PBPL/PCPL/LPWMGxC/LPWMGxDTH /LPWMGxDTL /LPWMGCLK/LPWMBCUBH/LPWMCUBL 等

(2) 关于仿真电压

- 仿真时，受仿真板工作电压影响，建议仿真电压≥2.5V

(3) 其他注意事项

- PDK5S-I-S01/2(B)仿真时，不支持指令 NMOV/SWAP/NADD/COMP 的 RAM 运算。
- PDK5S-I-S01/2(B)仿真时，不支持 SYSCLK=ILRC/16。
- PDK5S-I-S01/2(B)仿真时，不支持 ***misc.4*** 动态设定（只能固定为 0 或 1）。
- PDK5S-I-S01/2(B)仿真时，不支持 ***Tm2.gpcrs/Tm3.gpcrs*** 功能。
- PDK5S-I-S01/2(B)无法仿真 ADCRGC[3:2]的 ADC 通道 F 的 Bandgap 参考电压，并固定只有 1.2V。

- PDK5S-I-S01/2(B)仿真时，ADCC 寄存器的 PC2 和 PC1 设定位置与 IC 不同。
- PDK5S-I-S01/2(B)仿真时，不支持 GPC_PWM，TMx_source，PWM_Source，TMx_bit 等 code option。
- PDK5S-I-S01/2(B)仿真时，RAM 只有 240 字节的数据存储器。
- PDK5S-I-S01/2(B)仿真时，ROM 只有 0xF00 个程序存储器。
- PDK5S-I-S01/2(B)仿真时，PCDIER 的设置与实际芯片不同。PDK5S-I-S01/2(B)的 PCDIER[0]用于设置 PC0~PC3 为数字输入，PCDIER[1] 用于设置 PC4~PC7 的数字输入。建议在使用 PDK5S-I-S01/2(B)做 PGS134 的仿真时，不要去设置 PCDIER。
- 当使用 ADCRGC 里的 PB1 时，PA1 必须浮空。
- 当使用 GPCC 输出时，PA3 会受影响。
- 仿真 PWM 波形时，建议用户在程序运行期间查看波形，当仿真器暂停或单步运行时波形可能会与实际不符。
- PDK5S-I-S01/2(B)仿真器的 ILRC 频率与实际 IC 不同，且未经校准，其频率范围大约在 34K~38KHz。
- 用 PDK5S-I-S01/2(B)仿真时，在 timer2/timer3 定周期模式下，改变 tm2ct/tm3ct 的值会影响占空比，对于实际 IC 则不会。
- 快速唤醒时间在仿真时有差异：PDK5S-I-S01/2(B): 128 系统时钟周期，PGS134: 45 ILRC。
- PDK5S-I-S01/2(B)仿真时，看门狗溢出时间也有差异：

WDT 周期设置	**PDK5S-I-S01/2**(B)	**PGS134**
misc[1:0]=00	2048 * T_{ILRC}	8192 * T_{ILRC}
misc[1:0]=01	4096 * T_{ILRC}	16384 * T_{ILRC}
misc[1:0]=10	16384 * T_{ILRC}	65536 * T_{ILRC}
misc[1:0]=11	256 * T_{ILRC}	262144 * T_{ILRC}