MFU901

8 位 MTP 型单片机带充电

数据手册

第 0.00 版

2025 年 8 月 8 日

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

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

重要声明

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

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

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

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

目 录

修订历史

修 订	日期	描 述
0.00	2025/08/08	初版

警告

在使用 IC 前，请务必认真阅读 MFU901 相关的 APN（应用注意事项）。

https://www.padauk.com.tw/cn/product/search_list.aspx?kw=PFB

（下列图示仅供参考，依官网为主）

Feature Documents Software &Tools Application Note

Content	Description	Download (CN)	Download (EN)
APN001	Output impedance of ADC analog signal source		
APN002	Over voltage protection		
APN003	Over voltage protection		
APN004	Semi-Automatic writing handler		
APN005	Effects of over voltage input to ADC		
APN007	Setting up LVR level		
APN011	Semi-Automatic writing Handler improve writing stability		
APN013	Notification of crystal oscillator		
APN019	E-PAD PCB layout guideline		

1. 单片机特点

1.1 特殊功能

- 通用系列
- 在 AC 阻容降压供电或有高 EFT 要求的应用必要时需修改系统电路以提高抗干扰能力
- 工作温度范围: -40°C ~ 85°C

1.2 系统特性

系列	MTP 程序空间	RAM (字节)	最大 IO 数量	ADC 最大通道编号
MFU901	3KW	128	18	14

- 1 个硬件 16 位定时器
- 两个 8 位硬件 PWM 生成器
- 三个 11 位硬件 PWM 生成器 (PWMG0, PWMG1 & PWMG2)
- 提供 1.20V 基准电压的 Band-gap 电路
- 最多 13 通道 12 位分辨率 ADC，其中一个通道来自内部 Band-gap 基准电压或 0.375*V_{DD}
- ADC 基准高电压：外部输入，内部 VDD，Band-gap 1.20V、2.2V、2.4V、2.68V
- 最多 18 个 IO 引脚，可选上拉电阻和下拉电阻
- 18 个 IO 引脚 驱动能力，灌电流 = 30 / 60mA（强），驱动电流 = 20mA
- 每个 IO 引脚均可配置为启用唤醒功能
- 内置 VDD/2 LCD，电压生成最大 4×9 LCD 显示。
- 时钟源：IHRC, ILRC
- 一个低功耗时钟（NILRC）定期唤醒 stopsys
- 对于每个启用唤醒功能的 IO，都支持两种可选的唤醒速度：正常和快速
- LVR 范围: 1.8V ~4.5V
- 按代码选项设置的外部中断引脚
- VCC 输入范围: 4.3V ~ 20V
- 可编程充电电流高达 500mA
- 提供具有热调节功能的 CC/CV 操作，最大限度地提高充电速率，同时避免过热风险
- 感应器两个主要功能：麦克风电容感测（MCS）与加热电阻感测（HRS）
  1. MCS 支持互电容范围为 10pF 至 24pF，并且：
     a. 当互电容 C = 10pF 时，典型侦测噪声为 21 LSB
     b. 当互电容 C = 24pF 时，典型侦测噪声为 22 LSB
  2. MCS 支持 MEMS 电容，并且：
     a. 当 MEMS 电容 C = 1.4pF 时，典型侦测噪声为 20 LSB
  3. HRS 支持的电阻范围为 0.5Ω 至 1.5Ω，并能侦测 0.1Ω 的电阻变化

1.3 CPU 特性

- 8 位高性能 RISC CPU
- 93 条高效的指令
- 大多数指令的执行周期为 1T
- 可程序设定的堆栈指针和堆栈深度
- 用于数据访问的直接和间接寻址模式。数据存储器可用作间接寻址模式的索引指针
- IO 空间和内存空间是独立的

1.4 购/封装信息

- MFU901-2J24A:QFN4*4-24L(0.5pitch)

  - 有关封装尺寸的信息，请参阅官方网站文件：“封装信息”

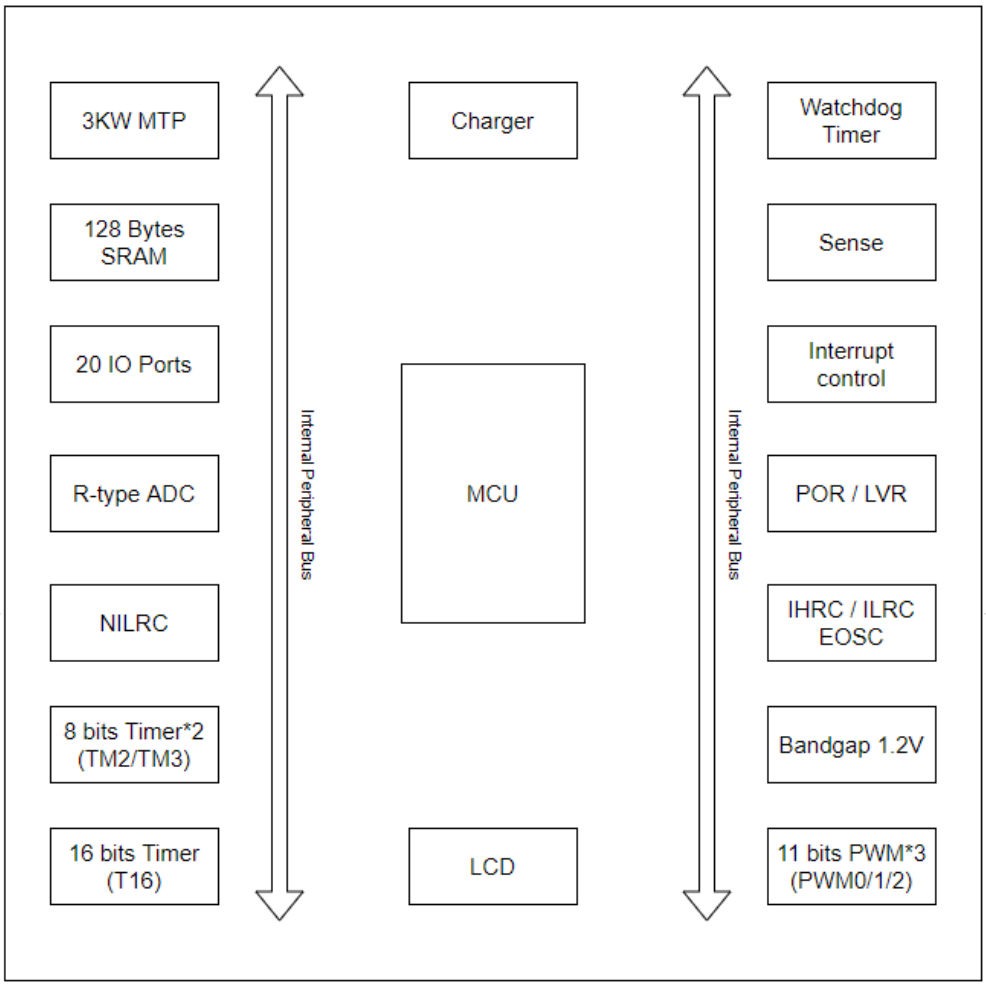
2. 系统概述和方框图

MFU901 系列是基于 MTP 的 CMOS 8 位微控制器，带有充电、感应和 12 位 ADC。它采用 RISC 架构，所有指令都在一个周期内执行，只有某些指令需要两个周期来处理间接存储器访问。

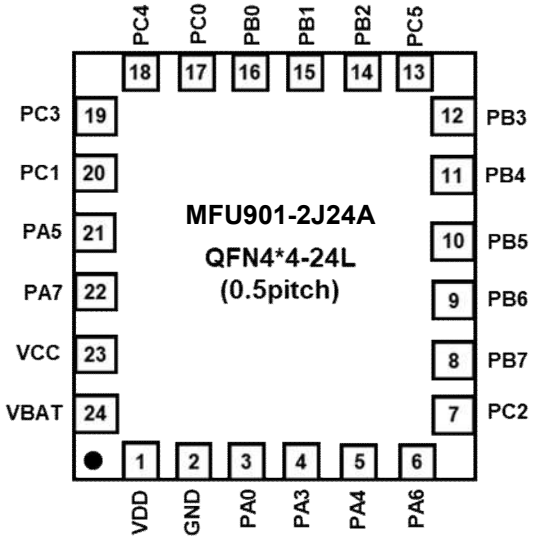
内置 3KW MTP 程序存储器和 128 字节数据 SRAM。芯片内置一个多达 13 个通道的 12 位 ADC，可选择多个参考电压源。MFU901 还提供 6 个硬件定时器：1 个 16 位定时器、2 个带 PWM 发生功能的 8 位定时器和 3 个带 PWM 发生功能的 11 位硬件定时器。MFU901 还支持用于 LCD 显示应用的 VDD/2 LCD 偏置电压发生器。

MFU901 中的充电器是用于单节锂离子电池的恒流/恒压充电器，其设计符合 USB 电源规格。当结温升高时，一个内部模块会对电流进行调节，以保护器件在高功率或高环境温度下工作。充电电压范围为 3.6V ~ 4.28V，充电电流限制可通过寄存器编程，无需外接电阻，电流最高可达 500mA。

MFU901 中的感应器模块具备两项主要功能：麦克风电容感测（MCS）与加热电阻感测（HRS）。在 MCS 模式下，感应器电路支持互电容与 MEMS 电容的检测。对于互电容检测，支持的电容范围为 10pF 至 24pF，对应的典型检测杂讯为 21 至 22 LSB。对于 MEMS 电容检测支持的电容为 1.4pF，典型检测杂讯为 20 LSB。在 HRS 模式下，Sense 电路支持的电阻范围为 0.5Ω 至 1.5Ω，能够检测最小 0.1Ω 的电阻变化。



3. 引脚分配及功能说明



引脚名称	引脚 & 缓冲器类型	描述
PA7 / 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) 外部中断 0。可用作外部中断 0 ，上升沿和下降沿均可请求中断服务，可通过寄存器设置进行配置。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 ***padier*** 位 7 为“0”时，唤醒功能是被关闭的。
PA6 / M2	CMOS	此引脚可以用作： M2 是 MIC 检测专用引脚。
PA5 / PRSTB / LCD2	IO (OD) 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。 (3) LCD2 将为第 2 组的 LCD 显示屏提供(1/2 VDD)。 当此引脚被配置为模拟输入时，请使用寄存器 ***padier*** 的第 5 位来禁用数字输入，以防止电流泄漏。可以将 ***padier*** 寄存器的第 5 位设置为“0”来禁用数字输入；通过触发此引脚实现的唤醒功能也会被禁用。
PA4 / M1	CMOS	此引脚可以用作： M1 是 MIC 检测专用引脚。

引脚名称	引脚 & 缓冲器类型	描述
PA3 / AD8 / LCD2 / INT1B / TM2PWM / PG2PWM / P2	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) LCD2 为第 2 组的 LCD 显示屏提供(1/2 V_{DD})。 (4) 外部中断 1。可用作外部中断 1，上升沿和下降沿均可请求中断服务，可通过寄存器设置进行配置。 (5) Timer2 的 PWM 输出。 (6) 11 位 PWM 生成器 PWMG2 的输出。 (7) P2 是恒流检测引脚。 当用做模拟输入功能时，为减少漏电流，请用 ***padier*** 寄存器位 3 关闭其数字输入功能。 这个引脚可以设定在睡眠中唤醒系统的功能；但是，当寄存器 ***padier*** 位 3 为”0”时，唤醒功能是被关闭的。
PA0 / AD10 / LCD2 / INT0 / PG0PWM P1	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) LCD2 为第 2 组的 LCD 显示屏提供(1/2 V_{DD})。 (4) 外部中断 0。可用作外部中断 0 ，上升沿和下降沿均可请求中断服务，可通过寄存器设置进行配置。 (5) 11 位 PWM 生成器 PWMG0 的输出。 (6) P1 是恒流检测引脚。 当用做模拟输入功能时，为减少漏电流，请用 ***padier*** 寄存器位 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，当寄存器 ***padier*** 位 0 为”0”时，唤醒功能是被关闭的。
PB7 / AD7 / TM3PWM / PG1PWM LCD0	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) Timer3 的 PWM 输出。 (4) 11 位 PWM 生成器 PWMG1 的输出。 (5) LCD0 为第 0 组的 LCD 显示屏提供(1/2 V_{DD})。 当用做模拟输入功能时，为减少漏电流，请用 ***pbdier*** 寄存器位 7 关闭数字输入功能。 ***pbdier*** 寄存器位 7 可以设置为“0”关闭其数字输入；通过切换此引脚禁用断电唤醒。

引脚名称	引脚 & 缓冲器类型	描述
PB6 / AD6 / LCD1 / 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) 外部中断 1C。可用作外部中断 1，上升沿和下降沿均可请求中断服务，可通过寄存器设置进行配置。 (4) Timer3 的 PWM 输出。 (5) 11 位 PWM 生成器 PWMG1 的输出。 当用做模拟输入功能时，为减少漏电流，请用 ***pbdier*** 寄存器位 6 关闭数字输入功能。 ***pbdier*** 寄存器位 6 可以设置为“0”关闭其数字输入；通过切换此引脚禁用断电唤醒。
PB5 / AD5 / LCD1 / 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) LCD1 为第 1 组的 LCD 显示屏提供(1/2 V_{DD}) 。 (4) 外部中断 0A。可用作外部中断 0，上升沿和下降沿均可请求中断服务，可通过寄存器设置进行配置。 (5) Timer3 的 PWM 输出。 (6) 11 位 PWM 生成器 PWMG0 的输出。 当用做模拟输入功能时，为减少漏电流，请用 ***pbdier*** 寄存器位 5 关闭数字输入功能。 ***pbdier*** 寄存器位 5 可以设置为“0”关闭其数字输入；通过切换此引脚禁用断电唤醒。
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。 当用做模拟输入功能时，为减少漏电流，请用 ***pbdier*** 寄存器位 4 关闭数字输入功能。 ***pbdier*** 寄存器位 4 可以设置为“0”关闭其数字输入；通过切换此引脚禁用断电唤醒。
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。 当用做模拟输入功能时，为减少漏电流，请用 ***pbdier*** 寄存器位 3 关闭数字输入功能。 ***pbdier*** 寄存器位 3 可以设置为“0”关闭其数字输入；通过切换此引脚禁用断电唤醒。

引脚名称	引脚 & 缓冲器类型	描述
PB2 / AD2 / LCD1 / 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) LCD1 为第 1 组的 LCD 显示屏提供 (1/2 V_{DD})。 (4) Timer2 的 PWM 输出。 (5) 11 位 PWM 生成器 PWMG2 的输出。 当用做模拟输入功能时，为减少漏电流，请用 ***pbdier*** 寄存器位 2 关闭数字输入功能。 ***pbdier*** 寄存器位 2 可以设置为“0”关闭其数字输入；通过切换此引脚禁用断电唤醒。
PB1 / AD1 / LCD1	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) LCD1 为第 1 组的 LCD 显示屏提供 (1/2 V_{DD})。 当用做模拟输入功能时，为减少漏电流，请用 ***pbdier*** 寄存器位 1 关闭数字输入功能。 ***pbdier*** 寄存器位 1 可以设置为“0”关闭其数字输入；通过切换此引脚禁用断电唤醒。
PB0 / AD0 / LCD2 / 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 模拟输入通道 0。 (3) LCD2 为第 2 组的 LCD 显示屏提供 (1/2 V_{DD})。 (4) 外部中断 1。可用作外部中断 1，上升沿和下降沿均可请求中断服务，可通过寄存器设置进行配置。 当用做模拟输入功能时，为减少漏电流，请用 ***pbdier*** 寄存器位 0 关闭数字输入功能。 ***pbdier*** 寄存器位 0 可以设置为“0”关闭其数字输入；通过切换此引脚禁用断电唤醒。
PC6 / LCD0	IO ST / CMOS	此引脚可以用作： (1) 端口 C 位 6，并可编程设定为输入或输出，弱上拉/下拉电阻模式。 (2) LCD0 为第 0 组的 LCD 显示屏提供(1/2 V_{DD})。 ***pcdier*** 寄存器的位 6 可以设置为“0”以禁用数字输入；通过切换此引脚禁用断电唤醒。
PC5 / LCD0	IO ST / CMOS	此引脚可以用作： (1) 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。 (2) LCD0 为第 0 组的 LCD 显示屏提供(1/2 V_{DD})。 ***pcdier*** 寄存器的位 5 可以设置为“0”以禁用数字输入；通过切换此引脚禁用断电唤醒。
PC4 / LCD0	IO ST / CMOS	此引脚可以用作： (1) 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。 (2) LCD0 为第 0 组的 LCD 显示屏提供 (1/2 V_{DD})。 ***pcdier*** 寄存器的位 4 可以设置为“0”以禁用数字输入；通过切换此引脚禁用断电唤醒。

引脚名称	引脚 & 缓冲器类型	描述
PC3 / PG1PWM / LCD0	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。 (3) LCD0 为第 0 组的 LCD 显示屏提供 (1/2 V_{DD})。 ***pcdier*** 寄存器的位 3 可以设置为“0”以禁用数字输入；通过切换此引脚禁用断电唤醒。
PC2 / AD12 / PG0PWM / LCD0	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 模拟输入通道 12。 (3) 11 位 PWM 生成器 PWMG0 的输出。 (4) LCD0 为第 0 组的 LCD 显示屏提供 (1/2 V_{DD})。 当该引脚配置为模拟输入时，请使用寄存器 ***pcdier*** 的第 2 位禁用数字输入，以防止电流泄漏。***pcdier*** 寄存器的位 2 可以设置为“0”以禁用数字输入；通过切换此引脚禁用断电唤醒。
PC1 / AD11 / LCD0	IO ST / CMOS / Analog	此引脚可以用作： (1) 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。 (2) ADC 模拟输入通道 11。 (3) LCD0 为第 0 组的 LCD 显示屏提供 (1/2 V_{DD})。 当该引脚配置为模拟输入时，请使用寄存器 ***pcdier*** 的第 1 位禁用数字输入，以防止电流泄漏。***pcdie***r 寄存器的位 1 可以设置为“0”以禁用数字输入；通过切换此引脚禁用断电唤醒。
PC0 / PG2PWM LCD0	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。 (3) LCD0 为第 0 组的 LCD 显示屏提供 (1/2 V_{DD})。 ***pcdier*** 寄存器的位 0 可以设置为“0”以禁用数字输入；通过切换此引脚禁用断电唤醒。
VDD/ VBAT/ VCC	VDD/ VBAT	VDD ：数字正电源。 VBAT：电池正电源。 VCC ：充电器正电源。
GND	GND	GND: 地
注意：IO：输入/输出；**ST**：施密特触发器输入；**OD**：开漏；**Analog**：模拟输入引脚； **CMOS**：CMOS 电压基准位		

4. 器件电器特性

4.1. 直流交流电气特性

下列所有数据除特别列明外，皆于 Ta = -40°C ~ 85°C，V_{BAT} / 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_{BAT} / V_{DD}	工作电压	1.8#	5.0	5.5	V	# 受限于 LVR 公差
VCC	充电器输入供电电压	4.3	5	6.5	V	VCC
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	 8M 4M 2M 90K		Hz	 V_{DD} ≧ 3.5V V_{DD} ≧ 2.2V V_{DD} ≧ 1.8V V_{DD} = 5.0V
V_{POR}	复位电压		2.0*		V	# 受限于 LVR 公差
I_{OP}	工作电流		0.7 426		mA uA	fSYS=IHRC/16=1MIPS@5.0 fSYS=ILRC=90KHz@5.0V
I_{PD}	掉电模式消耗电流 （使用 stopsys 命令）		0.7 0.3		uA uA	f_{SYS}= 0Hz, V_{DD} =5.0V f_{SYS}= 0Hz, V_{DD} =3.0V
I_{PS}	省电模式消耗电流 （使用 ***stopexe*** 命令）		2.9		uA	V_{BAT} =5.0V; f_{SYS}= ILRC 仅使用 ILRC 模式条件下
V_{IL}	输入低电压	0		0.2 V_{BAT}	V	
V_{IH}	输入高电压	0.7 V_{BAT}		V_{BAT}	V	
I_{OL}	IO 输出灌电流					
	PA0, PA3, PA5, PA7： 端口 B 与 端口 C 强 普通		 62 32		mA	V_{DD} =5.0V, V_{OL}=0.5V
I_{OH}	IO 输出驱动电流					
	全部 IO		-20		mA	V_{DD} =5.0V, V_{OH}=4.5V
V_{IN}	输入电压	-0.3		V_{BAT} +0.3	V	
$I_{INJ\ (PIN)}$	引脚输入电流			1	mA	V_{DD} +0.3≧V_{IN}≧ -0.3
R_{PH}	上拉电阻		72		KΩ	V_{DD} =5.0V
R_{PL}	下拉电阻		72		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, -40°C <Ta< 85°C*
		13.60*	16*	18.40*		V_{DD} =1.8V~5.5V, -40°C <Ta< 85°C

符号	描述	最小值	典型值	最大值	单位	条件(Ta=25°C)
f_{ILRC}	ILRC 频率 *		90		KHz	V_{DD} = 5.0V
f_{NILRC}	NILRC 频率*		15		KHz	V_{DD} = 5.0V
t_{INT}	中断脉冲宽度	30			ns	V_{DD} = 5.0V
V_{AD}	AD 输入电压	0		V_{DD}	V	
ADrs	ADC 分辨率			12 10	bit	0°C <Ta< 50°C* -40°C <Ta< 85°C*
ADcs	ADC 消耗电流		0.9 0.8		mA	@5.0V @3.0V
ADclk	ADC 时钟周期		2		us	1.8V ~ 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		±4*		LSB	12 位分辨率 LSB
AD INL	ADC 积分非线性		±8*		LSB	12 位分辨率 LSB
ADos	ADC 失调电压		5*		mV	@ V_{DD} =3.0V
N_{MCS}	MCS 检测噪声		21		LSB	C_m = 10 pF, 12 位分辨率 LSB
			22		LSB	C_m = 24 pF, 12 位分辨率 LSB
			20		LSB	$C_{MEMS-capacitor}$ = 1.4 pF, 12 位分辨率 LSB
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}	从开机启动开始的系统启动周期		32		ms	V_{DD} = 5.0V
t_{RST}	外部复位脉冲宽度	120			us	@ V_{DD} = 5.0V
I_{VCC}	充电器输入电源电流		200	500	μA	充电模式
			57			待机模式
			38			关机模式
			0			休眠模式
I_{CCM}	恒流模式充电电流	-15%	145	+15%	mA	@VCC=5V
			275			
			375			
			500			
V_{ASD}	VCC-V_{BAT} 锁定阈值电压		100		mV	VCC 上升
			30		mV	VCC 下降

符号	描述	最小值	典型值	最大值	单位	条件(Ta=25°C)
t_{RECHA}	充电比较器过滤器时间		2		mS	V_{BAT} 高到低
t_{TERM}	终端比较器滤波器时间		1		mS	I_{CCM} 少于 1/10
I_{TERM}	C/10 终止电流阈值		0.1			
$\triangle V_{RECHA}$	重新充电蓄电池阈值电压		150		mV	
T_{LIM}	恒温模式下的结点温度		90		℃	
V_{OVP}	过压保护阈值电压	6.5	6.9	7.3	V	
V_{OVPHYS}			0.8		V	

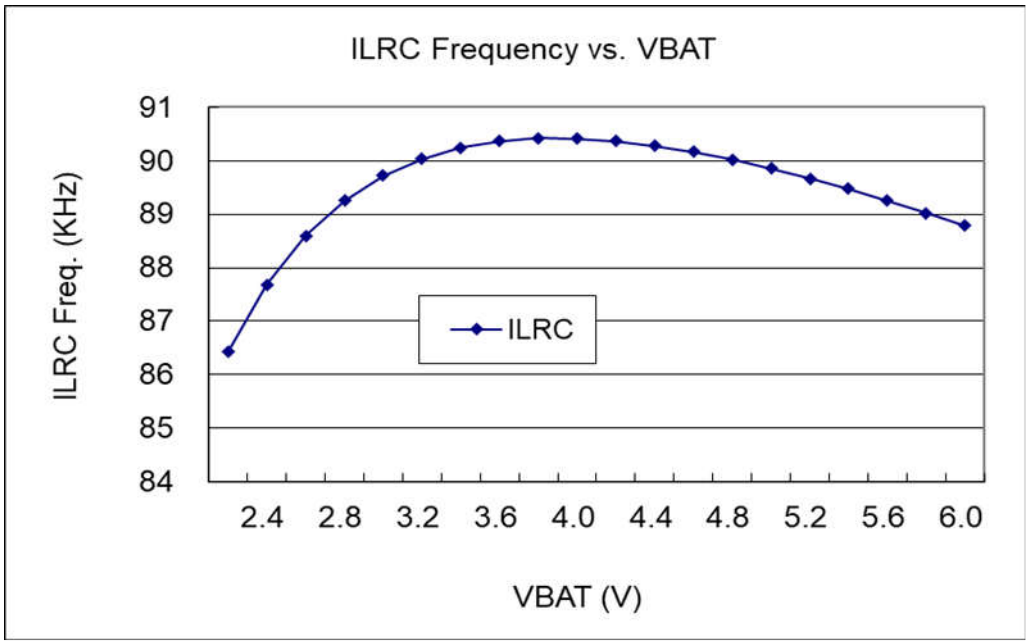
* 这些参数是设计参考值，并不是每个芯片测试。

** 特性图是实际测量值。考虑到生产飘移等因素的影响，表格中的数据是在实际测量值的安全范围内。

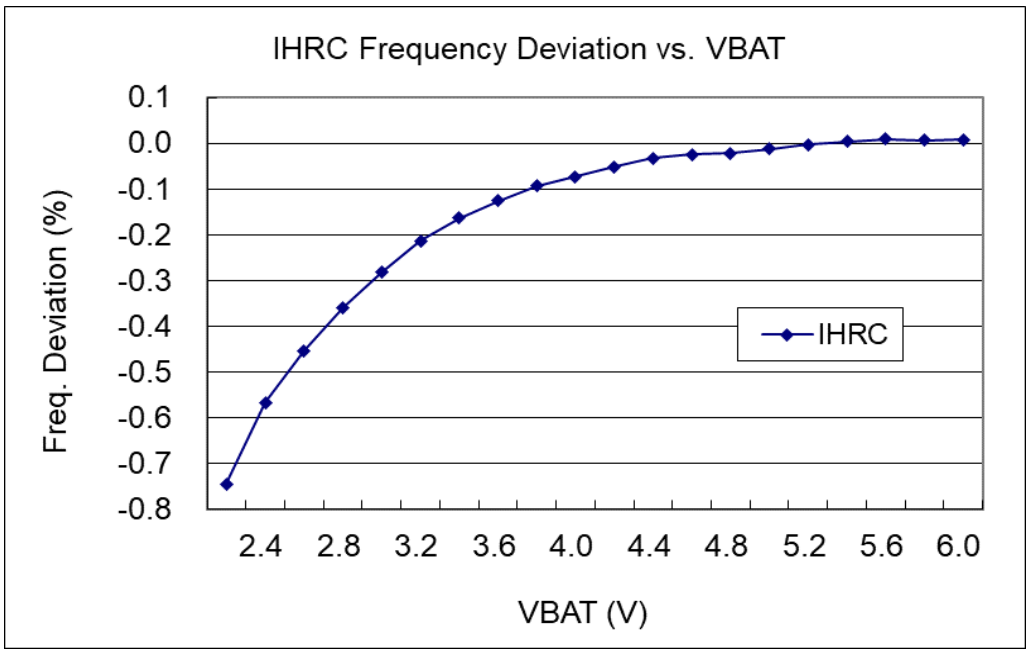
4.2. 绝对最大值范围

项目	数值范围 / 最大值	备注
电源电压	1.8V ~ 5.5V（最大值：5.5V）	如果 V_{BAT} / V_{DD} 超过最大额定值，则会损坏 IC
输入电压	-0.3V ~ V_{BAT} + 0.3V	
工作温度	-40°C ~ 85°C	
存储温度	-50°C ~ 125°C	
结点温度	150°C	

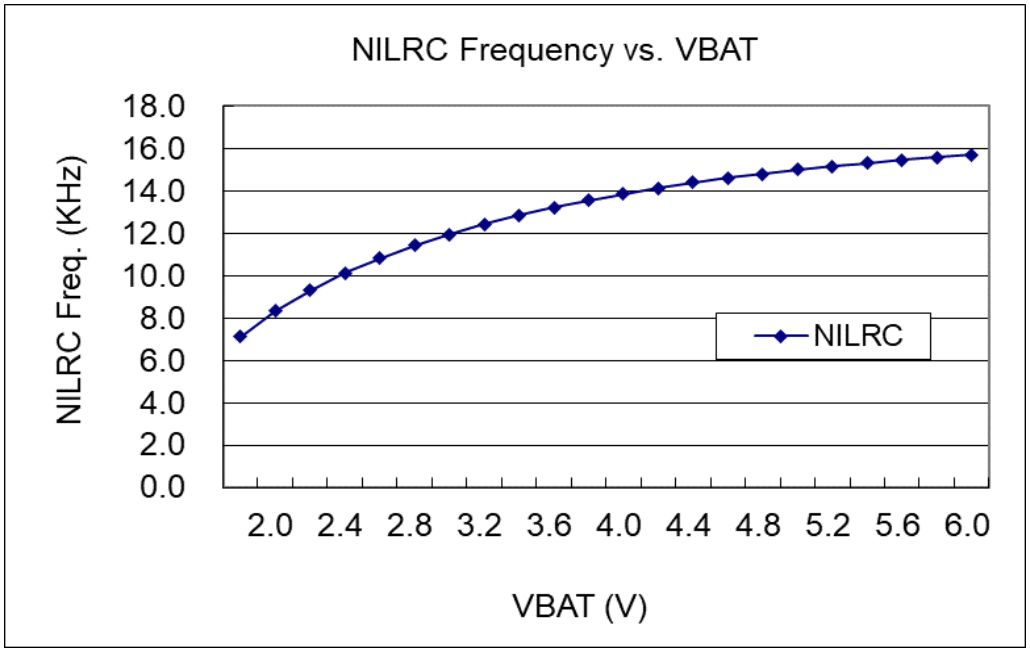
4.3. ILRC 频率与 V_{BAT} 关系曲线图 ($V_{BAT} = V_{DD}$)



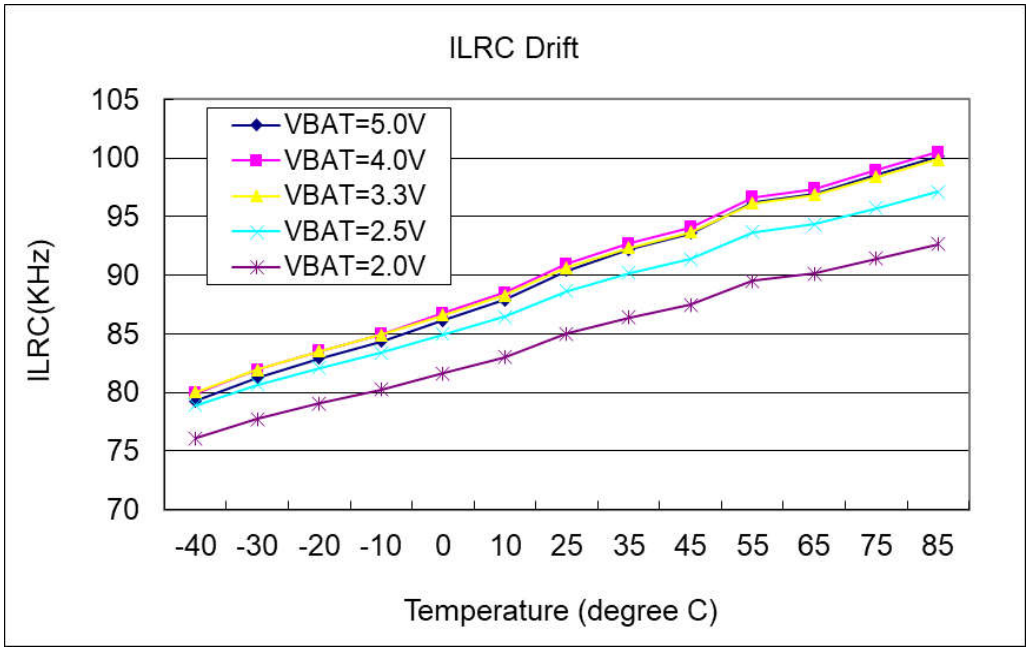
4.4. IHRC 频率与 V_{BAT} 关系曲线图（校准到 16MHz, $V_{BAT} = V_{DD}$）



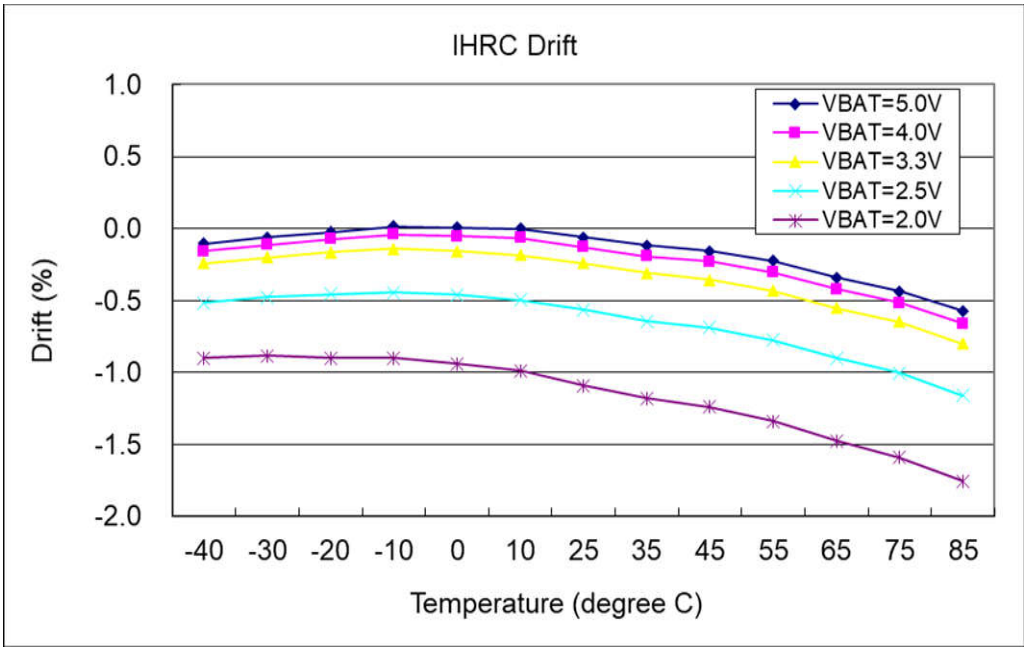
4.5. NILRC 频率与 V_{BAT} 关系曲线图 ($V_{BAT} = V_{DD}$)



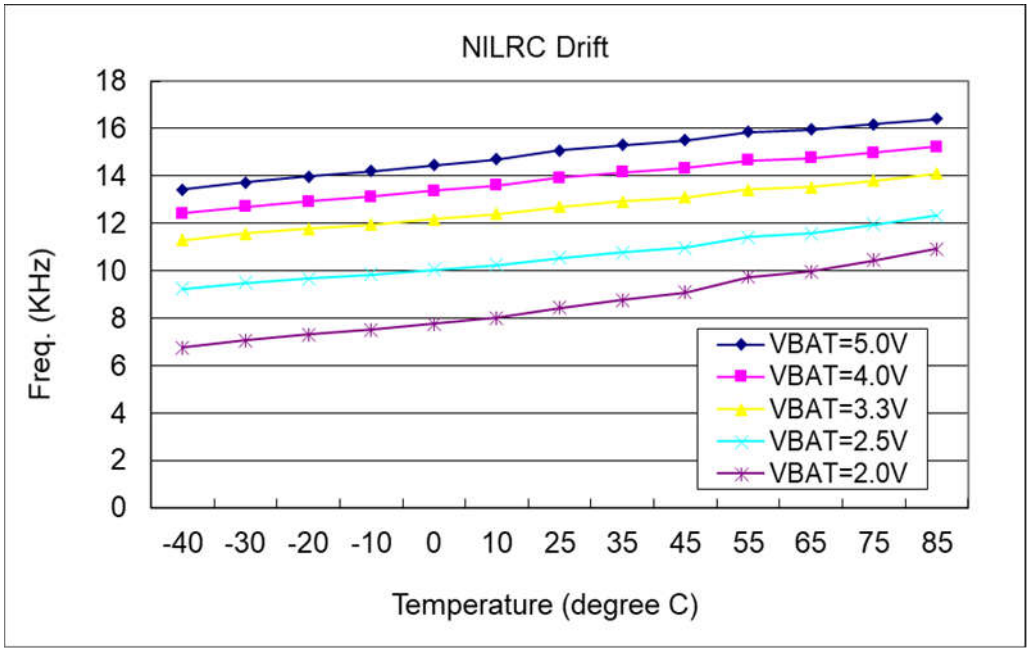
4.6. ILRC 频率与温度关系曲线图 (V_{BAT} = V_{DD})



4.7. IHRC 频率与温度关系曲线图（校准到 16MHz, V_{BAT} = V_{DD}）

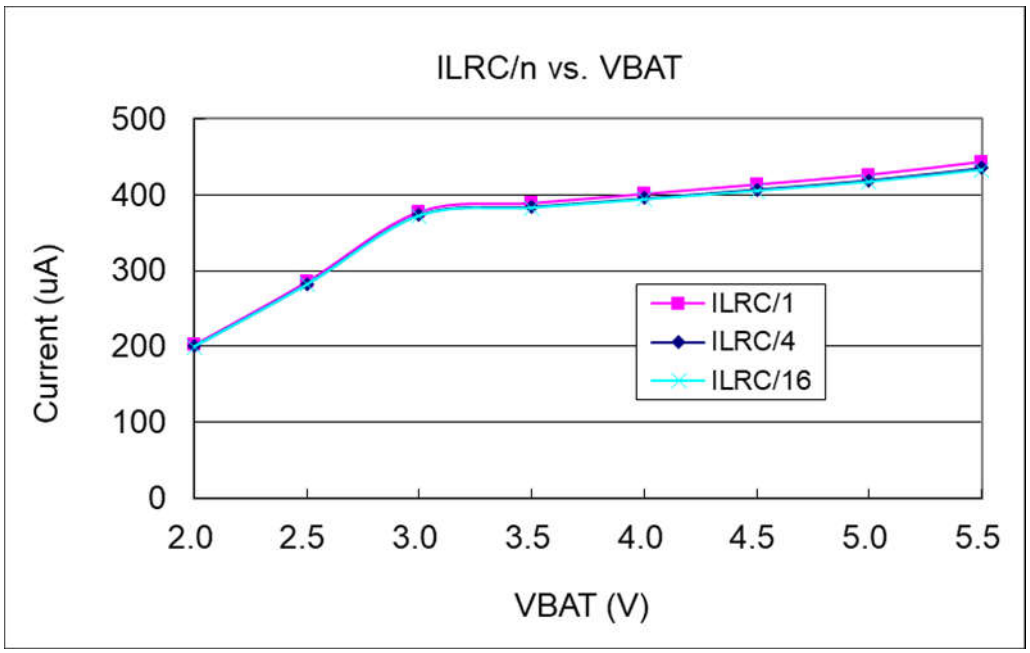


4.8. NILRC 频率与温度关系曲线图 (V_{BAT} = V_{DD})



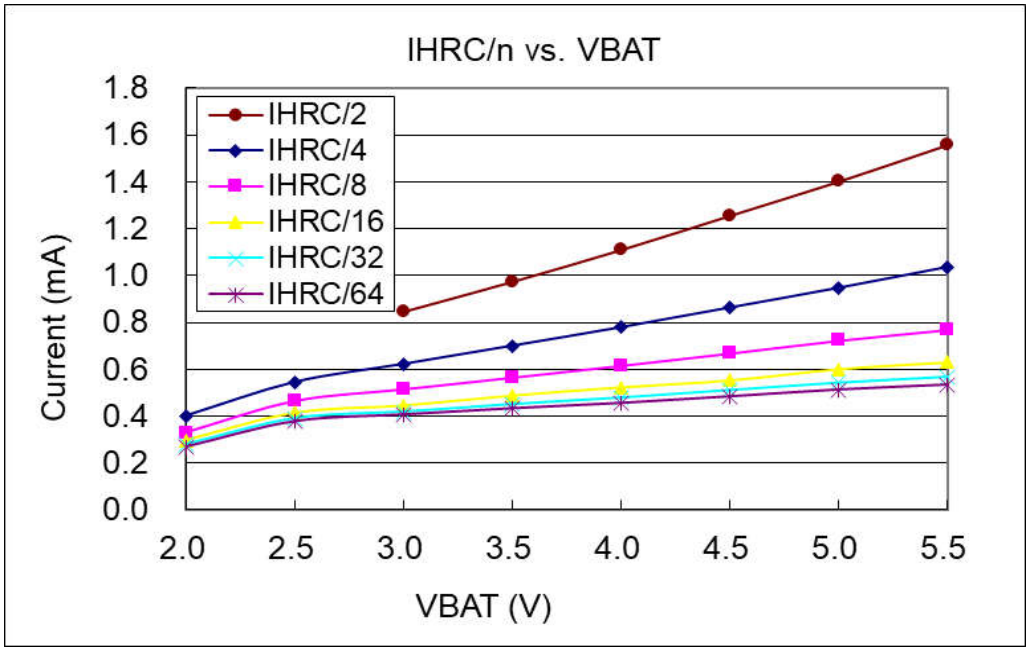
4.9. 工作电流 vs. V_{BAT} 与系统时钟 = ILRC/n 关系曲线图 (V_{BAT} = V_{DD})

- 条件： **ON**: Bandgap, LVR, ILRC; **OFF**: IHRC, EOSC, T16, TM2, LPWM, GPC;
  IO: PA0:0.5Hz 输出翻转不悬空， **其它**: 输入且 IO 引脚不悬空



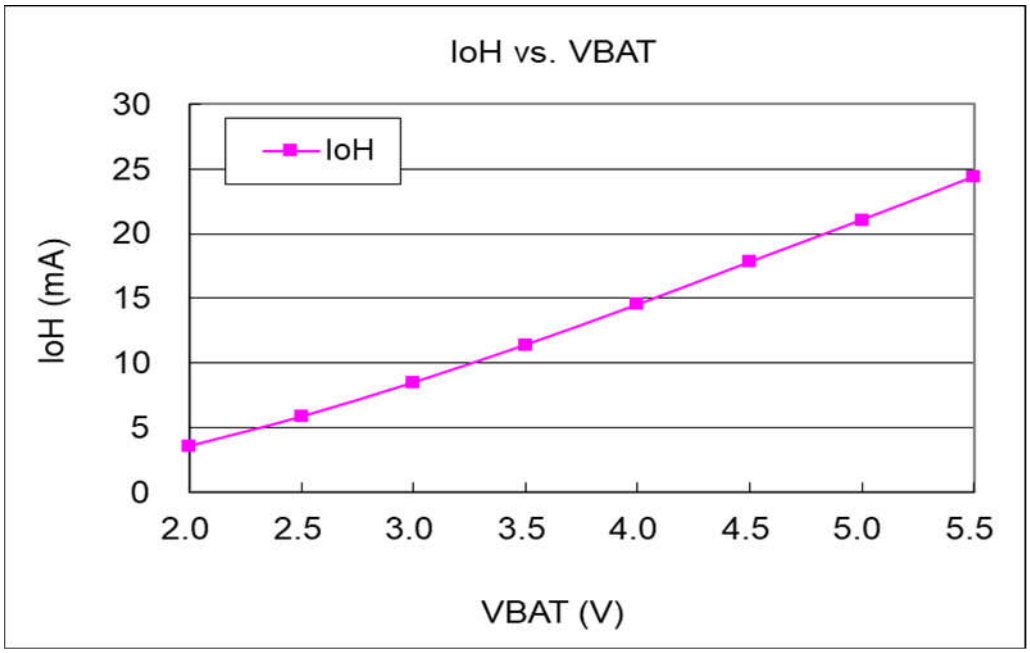
4.10. 工作电流 vs. V_{BAT} 与系统时钟 = IHRC/n 关系曲线图 ($V_{BAT} = V_{DD}$)

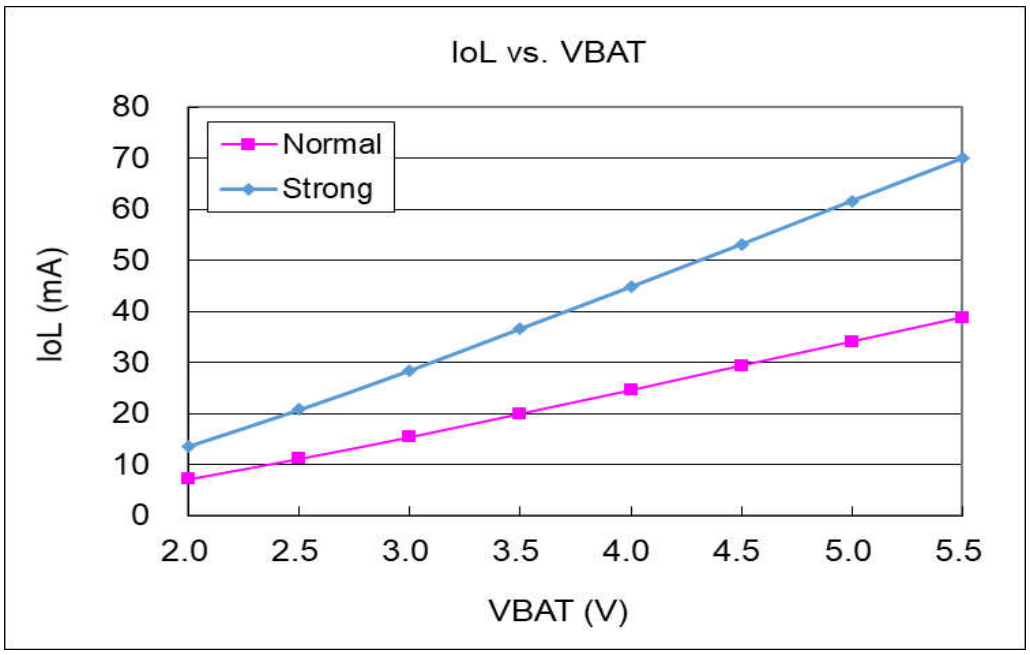
- 条件：**ON**: Bandgap, LVR, IHRC; **OFF:** ILRC, EOSC, T16, TM2, LPWM, GPC;
  IO: PA0:0.5Hz 输出翻转不悬空， **其它**: 输入且 IO 引脚不悬空。.



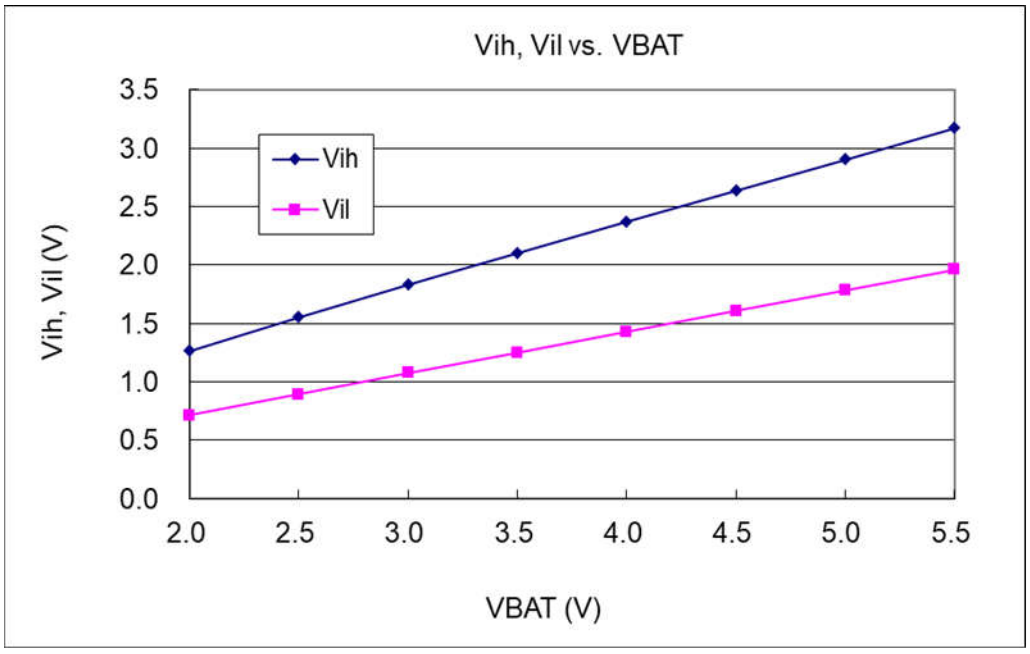
4.11. IO 引脚输出的驱动电流(I_{OH})与灌电流(I_{OL})曲线图 ($V_{BAT} = V_{DD}$)

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

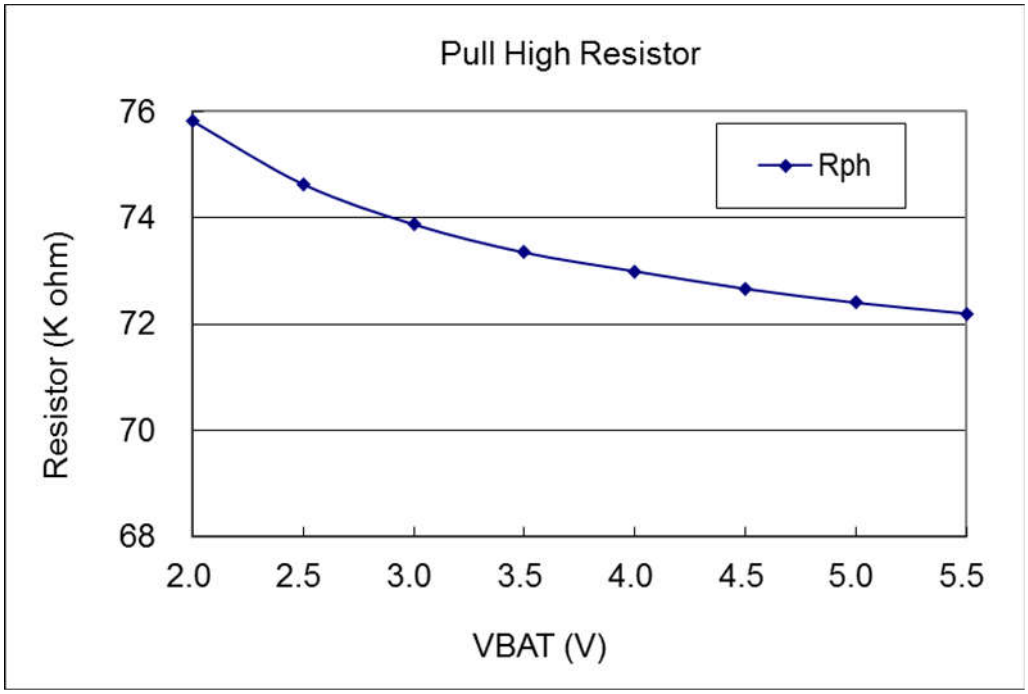


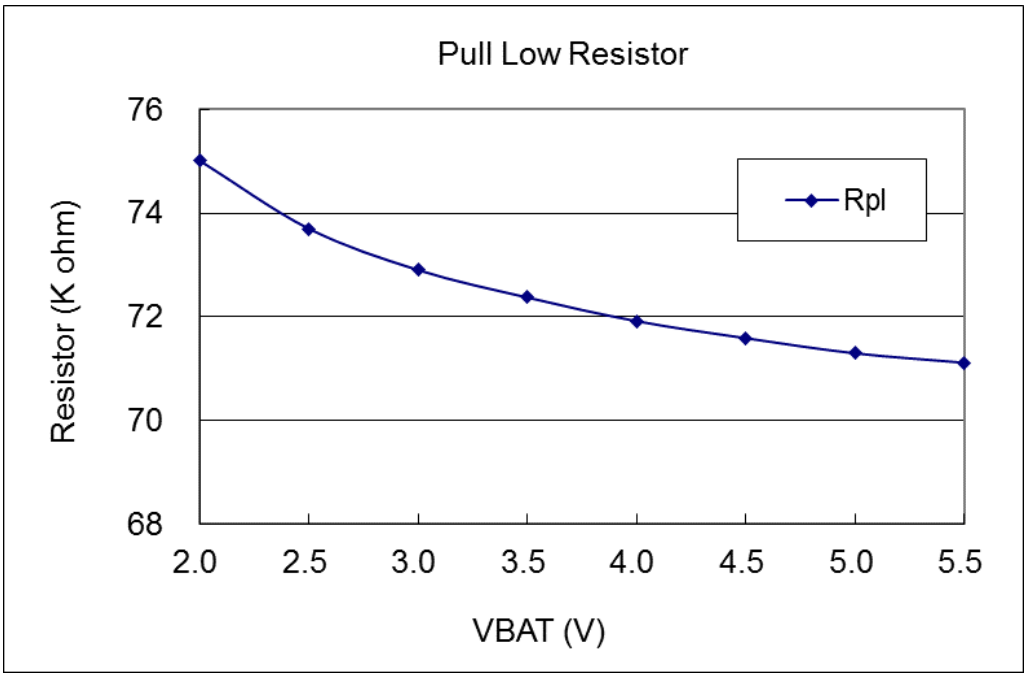


4.12. IO 引脚输入高/低阈值电压曲线图 (V_{IH}/V_{IL}) ($V_{BAT} = V_{DD}$)

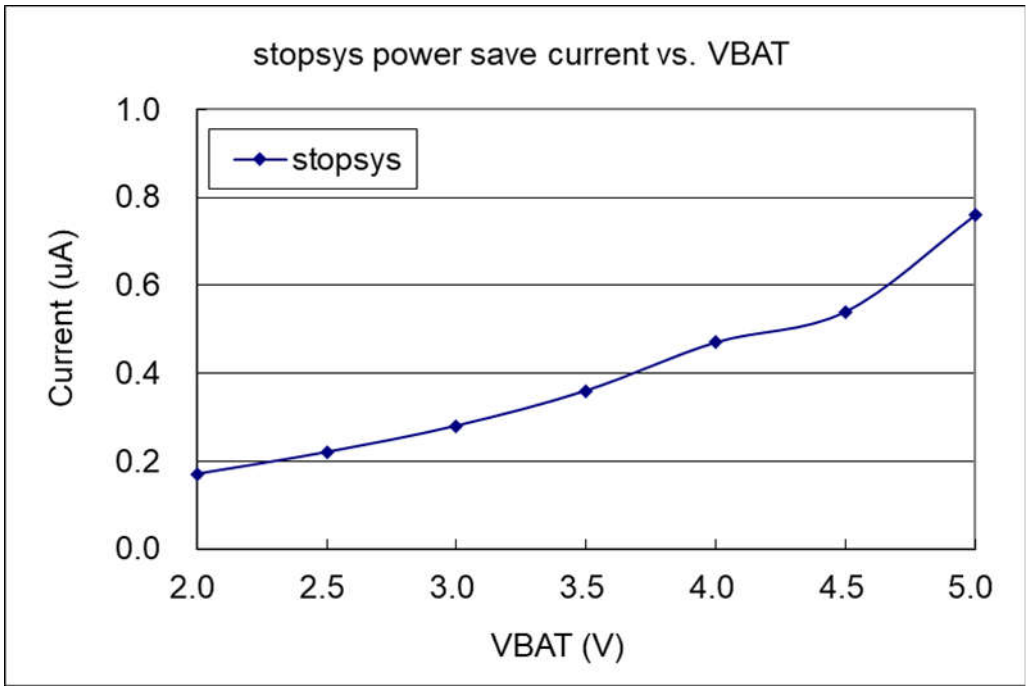


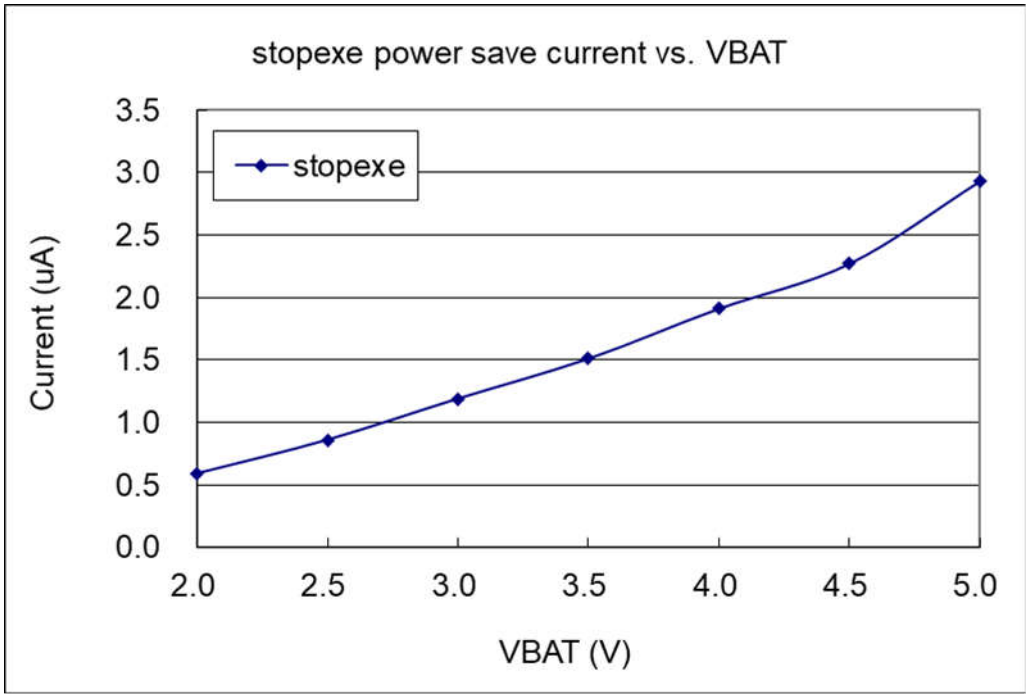
4.13. IO 引脚上拉/下拉阻抗曲线图 (V_{BAT} = V_{DD})





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





5. 功能概述

5.1 程序内存 - MTP

MTP 多次可编程程序存储器用于存储要执行的程序指令。MTP 程序存储器可包含数据、表格和中断入口。复位后，程序将从初始地址 0x000 开始，通常是 GOTO FPPA0 指令。如果使用，中断入口为 0x10，最后 32 个地址保留给系统使用，如校验和、序列号等。MFU901 的 MTP 程序存储器为 3KW，分区如表 1 所示。地址 0xBE0 至 0xBFF 的 MTP 存储器供系统使用，地址空间 0x001 至 0x00F 和 0x011 至 0XBDF 为用户程序空间。

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
•	•
0xBDF	用户程序区
0XBE0	系统使用
•	•
0xBFF	系统使用

表 1: 程序内存结构

5.2 启动程序

POR（上电复位）用于在上电时复位 MFU901。开机时间可选择快速或正常。上电顺序如图 1 所示，t_{SBP} 为启动时间。

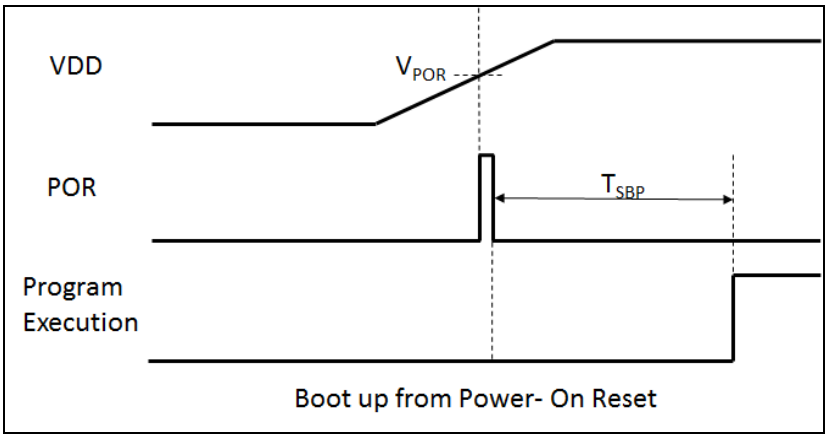
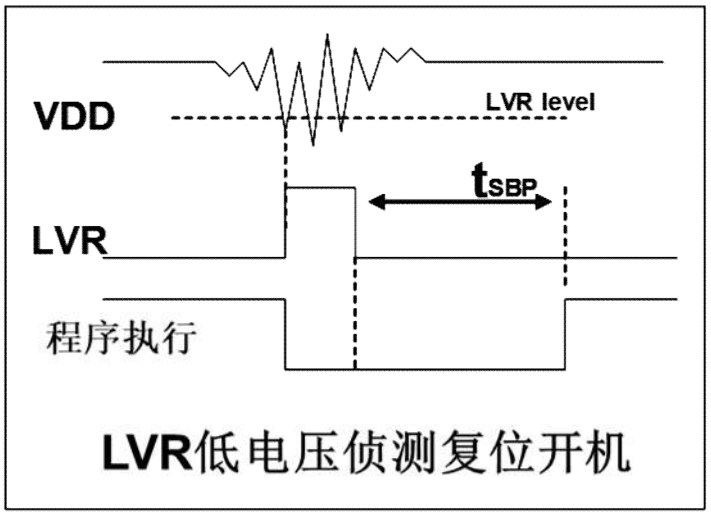
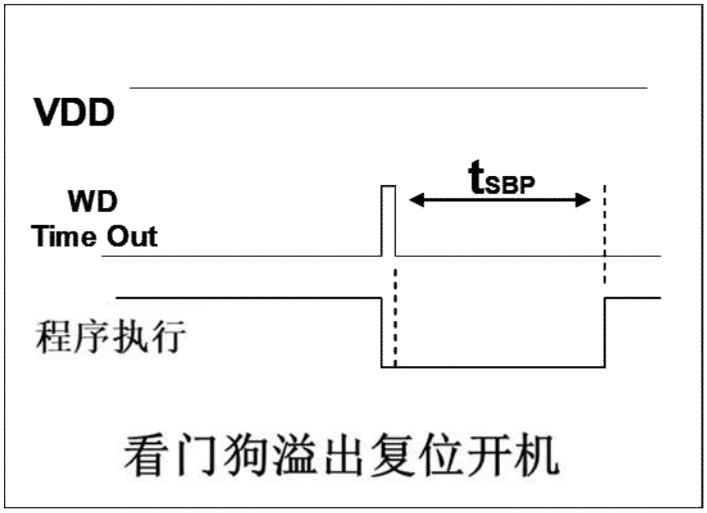
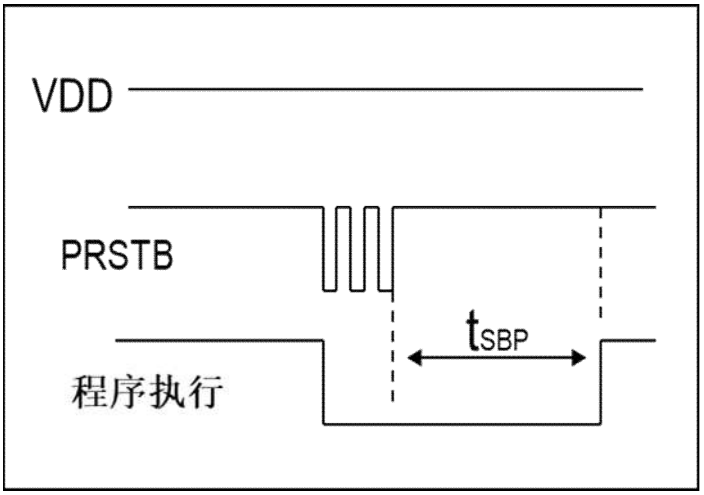


图.1: 上电时序

5.2.1 复位时序图


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


VDD
WD
Time Out
tSBP
程序执行
看门狗溢出复位开机


VDD
PRSTB
tSBP
程序执行

5.3 数据存储器 - SRAM

对数据存储器的访问可以是字节操作，也可以是位操作。SRAM 数据存储器除了存储数据外，还用作间接存取方式的数据指针和堆栈存储器。

堆栈存储器在数据存储器中定义。堆栈指针定义在堆栈指针寄存器中；每个处理单元的堆栈存储器深度由用户定义。堆栈存储器的排列完全灵活，用户可以动态调整。

在间接存储器访问机制中，数据存储器被用作数据指针来寻址数据字节。所有数据存储器都可以作为数据指针；这对于间接存储器访问非常灵活和有用。由于数据宽度为 8 位，因此 MFU901 的 128 字节数据存储器均可通过间接访问机制进行访问。

5.4 振荡器和时钟

MFU901 提供三个振荡电路：内部高 RC 振荡器（IHRC）和内部低 RC 振荡器（ILRC），这两个振荡器分别由寄存器 clkmd.4 和 clkmd.2 启用或禁用。用户可以从这两个振荡器中选择一个作为系统时钟源，并使用 clkmd 寄存器将所需频率作为系统时钟，以满足不同的应用需求。

振荡器模块	启用/停用
IHRC	clkmd.4
ILRC	clkmd.2
NILRC	保持启用

表 2：振荡器模块

5.4.1 内部高频 RC 振荡器和内部低频 RC 振荡器

开机后，IHRC 和 ILRC 振荡器是自动启用的。IHRC 频率能通过 ***ihrcr*** 寄存器校准，通常校准到 16 MHz。校准后的频率偏差通常在 1%以内；且校准后 IHRC 的频率仍然会因电源电压和工作温度而略有漂移。请参阅 IHRC 频率和 VDD、温度的测量图表。

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

5.4.2 芯片校准

IHRC 频率和带隙基准电压可能因芯片制造差异而不同，MFU901 提供 IHRC 频率校准以消除这种差异，在编译用户程序时可选择此功能，命令将自动插入用户程序。校准命令如下所示：

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

式中，**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。

5.4.3 IHRC 频率校准和系统时钟

在编译用户程序时，IHRC 校准和系统时钟选项如表 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 将是启动后的第一条命令，以便在启动系统时设置目标工作频率。IHRC 频率校准的程序代码仅在将代码写入 MTP 存储器时执行一次，之后将不再执行。如果选择了不同的 IHRC 校准选项，系统启动后的状态也会不同。下面显示了不同选项下 MFU901 的状态：

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

开机后，CLKMD = 0x34：

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

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

开机后 CLKMD = 0x14：

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

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

开机后 CLKMD = 0x3C：

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

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

开机后 CLKMD = 0x1C：

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

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

开机后 CLKMD = 0x7C：

- IIHRC 频率在 VDD=5V 时校准到 16MHz，并且 IHRC 模块是启用的
- 系统时钟 = IHRC/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：

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

(7) .ADJUST_IC　　DISABLE

开机后 CLKMD 寄存器没有改变（没任何动作）：

- IHRC 没有校准并且 IHRC 模块是停用的
- 系统时钟 = ILRC or IHRC/64 (由开机时间决定)
- 看门狗计数器停用，ILRC 启用，PA5 引脚是输入模式

5.4.4 统时钟和 LVR 基准位

系统时钟来自 IHRC 或者 ILRC，MFU901 的时钟系统的硬件框图，如图 2 所示：

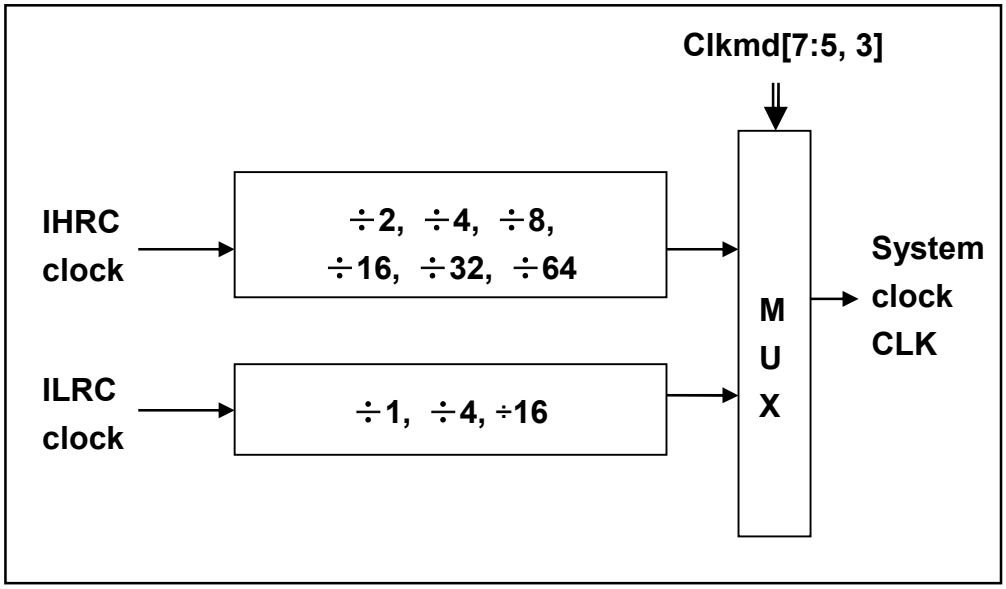


图 2：系统时钟选项

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

5.4.5 系统时钟切换

IHRC 校准后，用户可能要求切换系统时钟到新的频率或者可能会随时切换系统时钟来优化系统性能及功耗。基本上，MFU901 的系统时钟能够随时通过设定寄存器 ***clkmd*** 在 IHRC 和 ILRC 之间切换。在设定寄存器 ***clkmd*** 之后，系统时钟立即转换成新的频率。**请注意，在下命令给 *clkmd* 寄存器时，不能同时关闭原来的时钟模块，**下面这些例子显示更多时钟切换需知道的信息，请参阅 IDE 工具“求助” → “使用手册” → “IC 介绍” → “缓存器介绍” → CLKMD” 。

例 1： 系统时钟从 ILRC 切换到 IHRC/2

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

例 2： 系统时钟从 IHRC/2 切换到 ILRC

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

例 3： 系统时钟从 IHRC/2 切换到 IHRC/4

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

例 4： 如果同时切换系统时钟关闭原来的振荡器，系统会当机

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

5.5 VDD/2 LCD 偏置电压发生器

该功能可通过 misc.4 和代码选项 LCD2 启用。有三组引脚可以选择作为 LCD COM 端口。通过选择 LCD2 的 PB0_PA035，PB0、PA0、PA3 和 PA5 在输入模式下被定义为输出 VDD/2 电压，并用作 LCD 应用的 COM 功能。选择 LCD2 的 PB1256 时，PB1、PB2、PB5 和 PB6 被定义为 COM 端口。通过选择 LCD2 的 PB7_PC0~6，PB7 和 PC0 至 PC6 被定义为 COM 端口。

如果用户希望输出 VDD、VDD/2、GND 三级电压，启用 VDD/2 偏置电压（通过设置 misc.4=1），然后将 VDD 设置为输出高电平，VDD/2 设置为输入模式，GND 设置为输出低电平，图 3 显示了如何使用该功能。

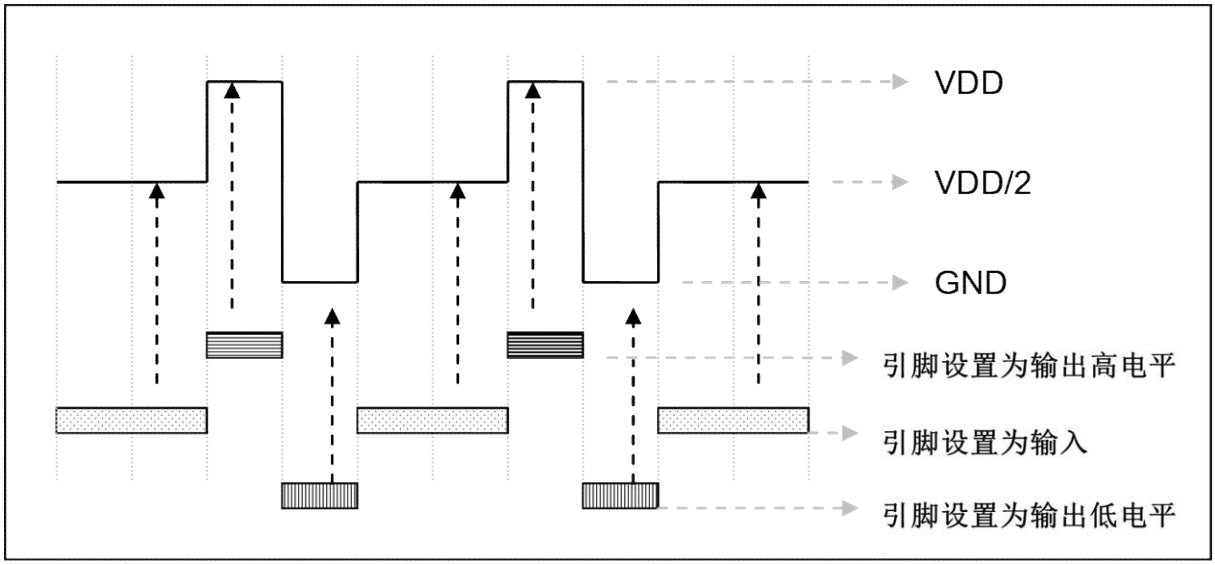


图. 3: 使用 VDD/2 产生 LCD 偏置电压

5.6 16 位计数器 (Timer16)

MFU901 中集成了一个 16 位的硬件定时器（Timer16），Timer16 的时钟源可以来自系统时钟（CLK）、内部高速 RC 振荡器（IHRC）、内部低速 RC 振荡器（ILRC）、PA4 引脚和 PA0 引脚，通过一个多路复用器来为时钟源选择时钟输出。在将时钟信号发送到 16 位计数器之前，会使用一个预分频逻辑，其分频系数为 1、4、16 和 64，以实现大范围的计数。

这个 16 位计数器仅执行向上计数操作，计数器的初始值可以通过 stt16 指令从存储器中存储，而计数值则可以通过 ldt16 指令加载到存储器中。使用一个选择器来选择 Timer16 的中断条件，每当发生溢出时，就可以触发 Timer16 中断。Timer16 的硬件原理图如图 4 所示。Timer16 的中断源来自 16 位计数器的第 8 位到第 15 位中的某一位，并且中断类型可以是上升沿触发或下降沿触发，这在 integs 寄存器（输入输出地址为 0x0C）的第 4 位中进行指定。

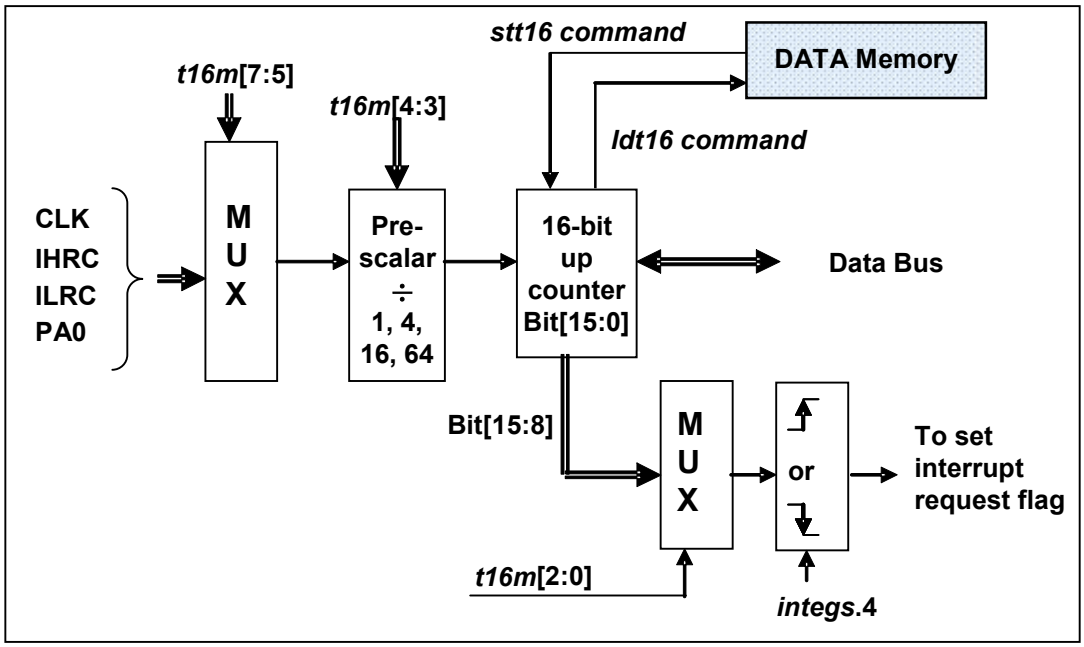


图.4: Timer16 模块框图

当使用 16 位硬件定时器（Timer16）时，Timer16 的语法已在.INC 文件中进行了定义。定义 Timer16 需要三个参数；第一个参数用于定义 Timer16 的时钟源，第二个参数用于定义预分频系数，最后一个参数用于定义中断源。详细说明如下：

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

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

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

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

$ T16M STOP;
    // 停止 Timer16 计数
```

如果 Timer16 在自由运行状态下运行，中断频率如下所示：

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

式中，F 为所选时钟源是 Timer16 的频率；

P 是 t16m[4:3]的选择；(1, 4, 16, 64)

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

5.7 8 位定时器 (Timer2/Timer3) PWM 生成器

MFU901 中集成了两个带有脉宽调制（PWM）生成功能的 8 位硬件定时器（定时器 2 和定时器 3）。以下的描述仅针对定时器 2。这是因为定时器 3 与定时器 2 的结构相同。请参考图 5 所示的定时器 2 硬件原理图，定时器 2 的时钟源可以来自系统时钟、内部高速 RC 振荡器（IHRC）、内部低速 RC 振荡器（ILRC）、PA0 引脚、PB0 引脚以及比较器。寄存器 tm2c 的第[7:4]位用于选择定时器 2 的时钟源。如果选择内部高速 RC 振荡器（IHRC）作为定时器 2 的时钟源，那么在使用在线仿真器（ICE）处于暂停状态时，发送给定时器 2 的时钟将继续运行。

定时器 2 的输出可以发送到 PB2 引脚、PA3 引脚或 PB4 引脚，这取决于 tm2c 寄存器的第[3:2]位。有一个时钟预分频模块，提供 1、4、16 和 64 的分频选项，由 tm2s 寄存器的第[6:5]位控制；还提供了一个分频范围为 1 到 32 的分频模块，由 tm2s 寄存器的第[4:0]位控制。结合预分频功能和分频功能，定时器 2 时钟（TM2_CLK）的频率可以实现大范围且灵活的调整。 定时器 2 的计数器仅执行 8 位向上计数操作；计数器的值可以通过 tm2ct 寄存器进行设置或读回。当 8 位计数器的值达到上限寄存器的值时，计数器将自动清零，上限寄存器用于定义定时器的周期或脉宽调制（PWM）的占空比。定时器 2 有两种工作模式：周期模式和脉宽调制（PWM）模式；周期模式用于生成周期性的输出波形或中断事件；脉宽调制（PWM）模式用于生成具有可选的 6 位到 8 位 PWM 分辨率的 PWM 输出波形，图 6 展示了定时器 2 在周期模式和 PWM 模式下的时序图。

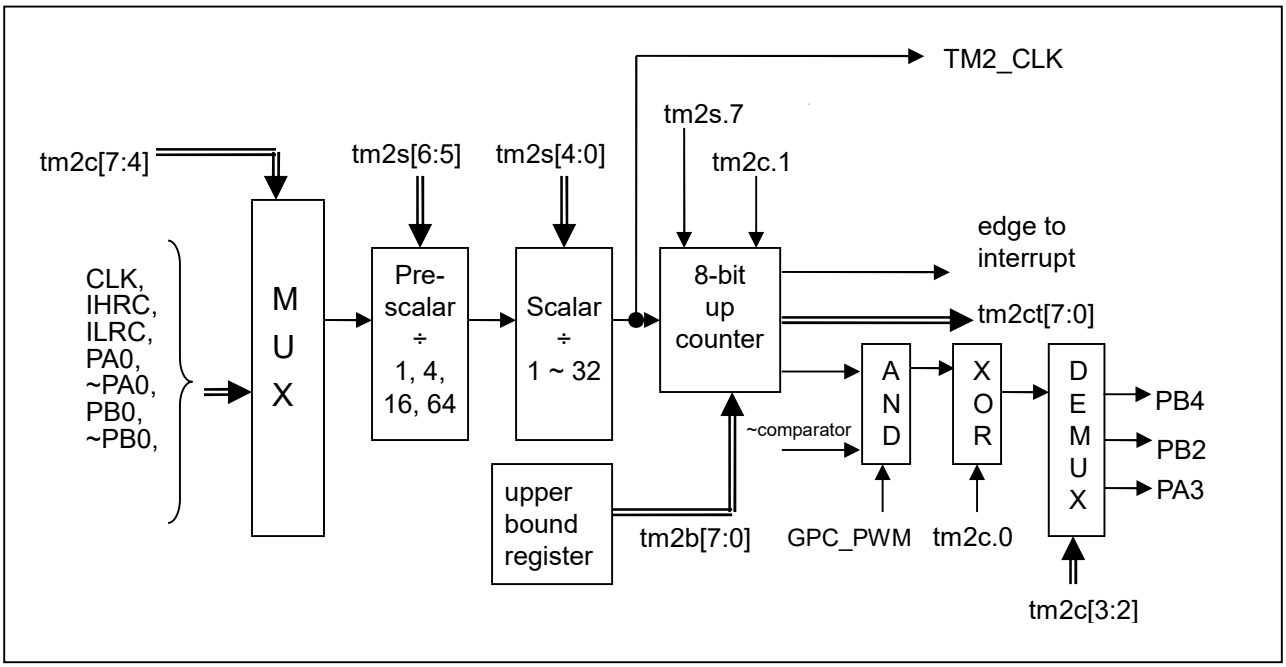


图 5: Timer2 硬件框图

Timer3 的输出可发送至 PB5、PB6 或 PB7 引脚。

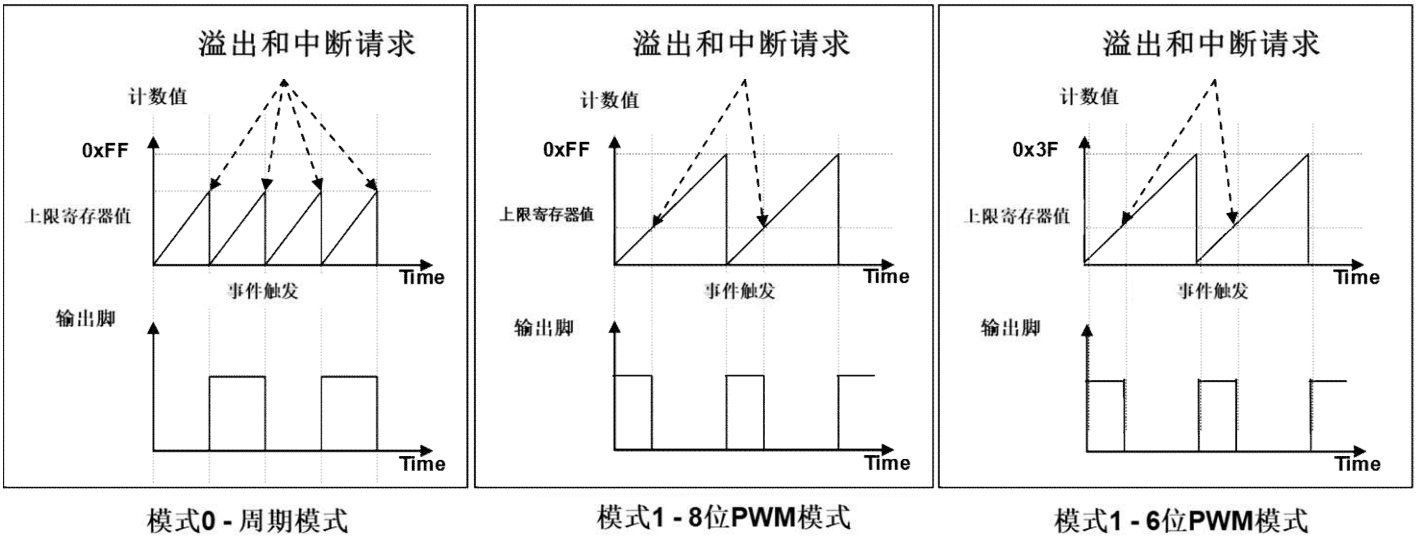


图 6：Timer2 在周期模式和 PWM 模式下的时序图（tm2c.1=1）

代码选项 GPC_PWM 适用于需要由比较器结果控制 PWM 波形的应用。如图 7 所示，如果选择了代码选项 GPC_PWM，则当比较器输出为 1 时，PWM 输出停止，然后当比较器输出回到 0 时，PWM 输出开启。

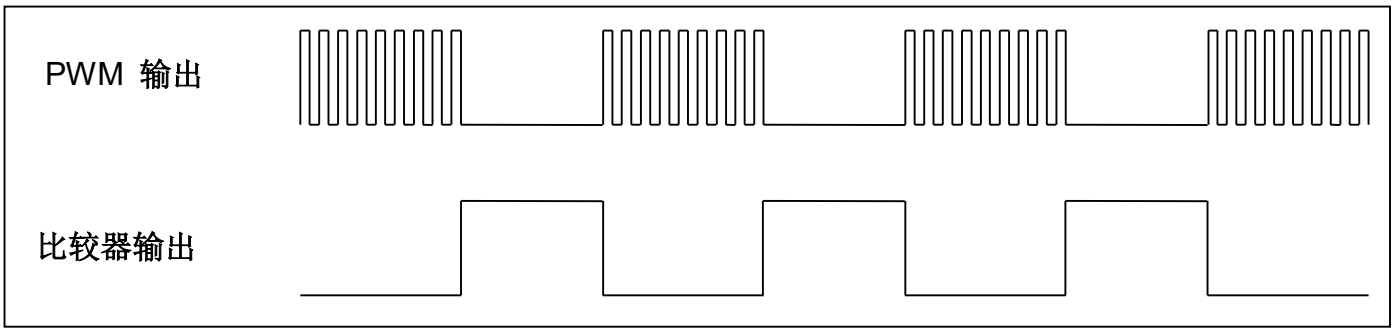


图.7：比较器控制 PWM 波形输出

5.7.1 使用 Timer2 生成周期性波形

如果选择周期模式，输出占空比始终为 50%；其频率可概括如下：

输出频率= Y ÷ [2 × (K+1) × S1 × (S2+1)]

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

例 1:

tm2c = 0b0001_1000, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s = 0b0000_00000, S1=1, S2=0
➔ 输出频率 = 8MHz ÷ [2 × (127+1) × 1 × (0+1)] = 31.25KHz

例 2:

tm2c = 0b0001_1000, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s[7:0] = 0b0111_11111, S1=64 , S2 = 31
➔ 输出频率 = 8MHz ÷ (2 × (127+1) × 64 × (31+1)) =15.25Hz

例 3:

tm2c = 0b0001_1000, Y=8MHz
tm2b = 0b0000_1111, K=15
tm2s = 0b0000_00000, S1=1, S2=0
➔ 输出频率 = 8MHz ÷ (2 × (15+1) × 1 × (0+1)) = 250KHz

例 4:

tm2c = 0b0001_1000, Y=8MHz
tm2b = 0b0000_0001, K=1
tm2s = 0b0000_00000, S1=1, S2=0
➔ 输出频率 = 8MHz ÷ (2 × (1+1) × 1 × (0+1)) =2MHz

使用 Timer2 定时器从 PA3 引脚产生周期波形的示例程序如下所示：

```
Void   FPPA0 (void)
{
    . ADJUST_IC    SYSCLK=IHRC/2, IHRC=16MHz, VBAT =5V
    …
    tm2ct = 0x00;
    tm2b = 0x7f;
    tm2s = 0b0_00_00001;          //  8 位 PWM，预分频器 = 1，分频器 = 2
    tm2c = 0b0001_10_0_0;         //  系统时钟，输出 = PA3，周期模式
    while(1)
    {
        nop;
    }
}
```

5.7.2 使用 Timer2 产生 8 位 PWM 波形

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

输出频率 = Y ÷ [256 × S1 × (S2+1)]

占空比 = [(K+1) ÷ 256]×100%

式中，　Y = tm2c[7:4] ： Timer2 所选择的时钟源频率

K = tm2b[7:0] ： 上限寄存器设定的值（十进制）

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

S2 = tm2s[4:0] ： 分频器值（十进制，S2=0 ~ 31）

例 1:

tm2c = 0b0001_1010, Y=8MHz

tm2b = 0b0111_1111, K=127

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

➔ 输出频率 = 8MHz ÷ (256 × 1 × (0+1)) = 31.25KHz

➔ 占空比 = [(127+1) ÷ 256] × 100% = 50%

例 2:

tm2c = 0b0001_1010, Y=8MHz

tm2b = 0b0111_1111, K=127

tm2s = 0b0111_11111, S1=64, S2=31

➔ 输出频率 = 8MHz ÷ (256 × 64 × (31+1)) = 15.25Hz

➔ 占空比 = [(127+1) ÷ 256] × 100% = 50%

例 3:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b1111_1111, K=255
tm2s = 0b0000_00000, S1=1, S2=0
➔ PWM 输出持续为高
➔ 占空比 = [(255+1) ÷ 256] × 100% = 100%

例 4:

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

使用 Timer2 从 PA3 生成 PWM 波形的示例程序如下所示：

```
void    FPPA0 (void)
{
    .ADJUST_IC      SYSCLK=IHRC/2, IHRC=16MHz, VBAT =5V
    wdreset;
    tm2ct = 0x00;
    tm2b = 0x7f;
    tm2s = 0b0_00_00001;            //  8 位 PWM，预分频器 = 1，分频器 = 2
    tm2c = 0b0001_10_1_0;           //  系统时钟，输出 = PA3，周期模式
    while(1)
    {
        nop;
    }
}
```

5.7.3 使用 Timer2 产生 6 位 PWM 波形

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

输出频率= Y ÷ [64 × S1 × (S2+1)]

占空比 = [(K＋1) ÷ 64] × 100%

式中， 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）

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

例 1:

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

例 2:

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

例 3:

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

例 4:

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

5.8 11-bit PWM 生成器

MFU901 内置一组三路 11 位 SuLED（Super LED）硬件 PWM 发生器。它由三个 PWM 发生器（LPWMG0、LPWMG1 和 LPWMG2）组成。各路输出端口如下：

- LPWMG0 – PA0, PB4, PB5
- LPWMG1 – PB6, PB7
- LPWMG2 – PA3, PB2, PB3, PA5

5.8.1 PWM 波形

PWM 输出波形（图 8）有一个时基（T_{Period} =周期时间）和一个周期里输出高电平的时间（占空比）。PWM 输出的频率取决于时基($f_{PWM} = 1/T_{Period}$)。

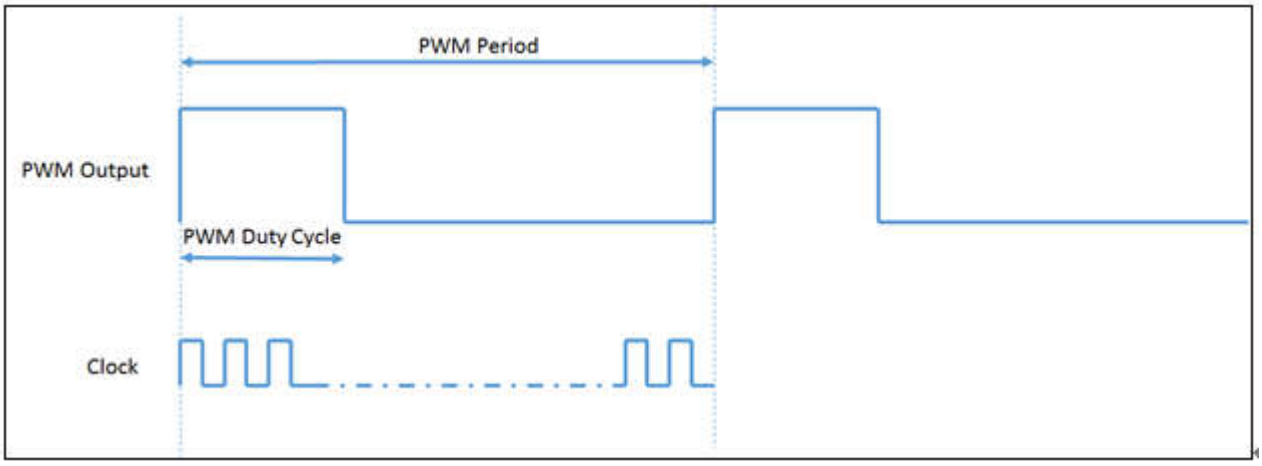


图.8: PWM 输出波形

5.8.2 硬件框图

图 9 显示了整组 SuLED 11 位硬件 PWM 发生器的硬件方框图。这三组 PWM 生成器使用共同的 Up-Counter 和时钟源选择开关来产生时基，所以 PWM 周期的起点（上升沿）是同步的，时钟源可以是 IHRC 或系统时钟。PWM 信号输出引脚通过寄存器 ***lpwmgxc*** 选择的。PWM 波形的周期由公共 PWM 上限高和低寄存器决定，各路 PWM 波形的占空比由各路 PWM 占空比高和低压寄存器决定。

LPWMG0 通道的附加 OR 和 XOR 逻辑门是用于产生互补非重叠并有死区的开关控制波形的。选择代码选项 GPC_LPWM 也可以根据比较器结果控制生成的 PWM 波形。

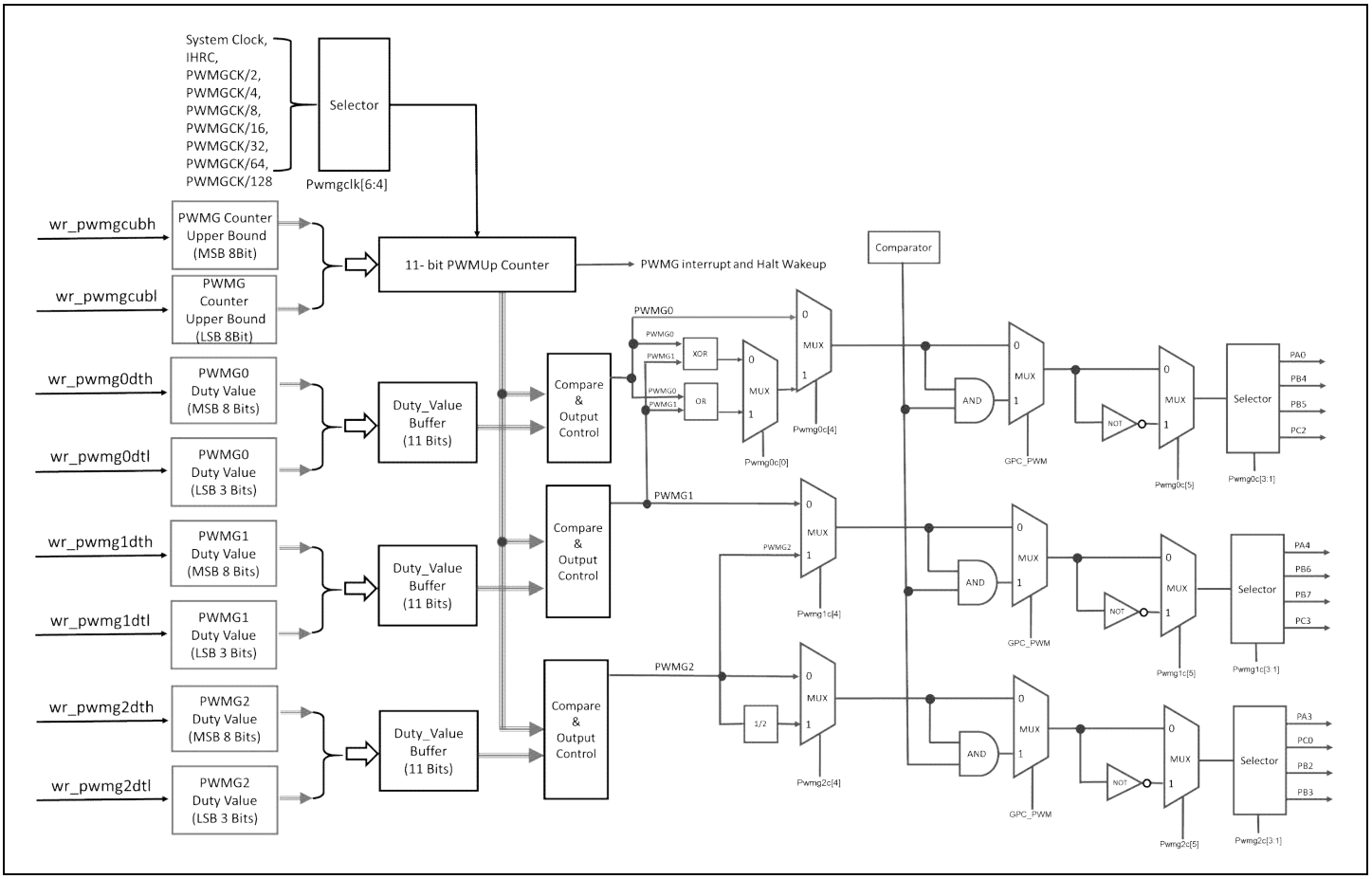


图.9: 一组 3 连套 11 位 SuLED PWM 生成器硬件框图

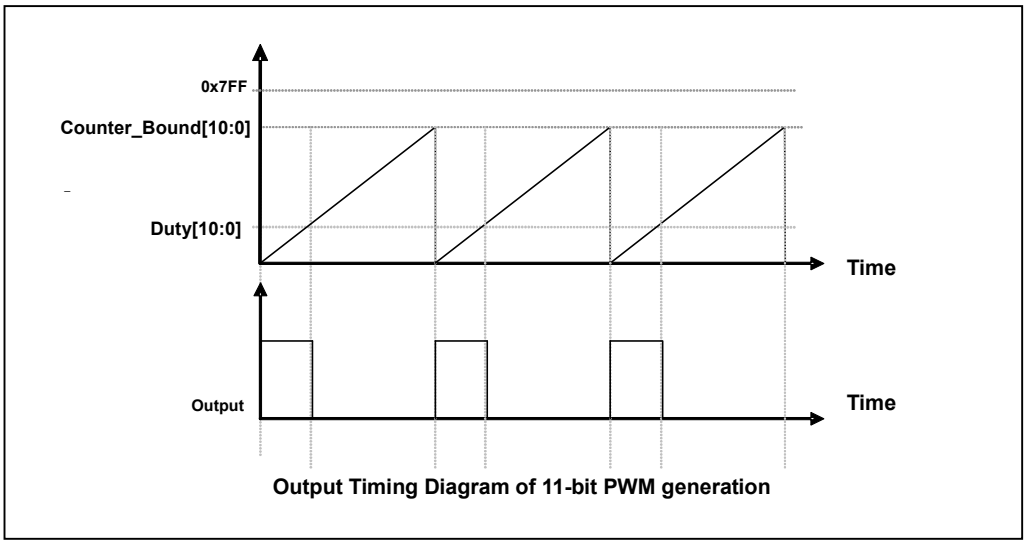


图.10: 11 位 PWM 生成器输出时序图

5.8.3 11 位 PWM 生成器计算公式

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

PWM 占空比(时间) = (1 / F_{PWM}) × (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%

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

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

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

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

5.8.4 带互补死区的 PWM 波形范例

基于 MFU901 特有的 11 位 PWM 架构，我们在 PWM0 或 PWM1 之后采用 PWM2 输出和 PWM0 反输出，以产生两个具有互补死区的 PWM 波形。

示例如下:

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

#define   PWM_Pulse_1     35      //   该互补死区 PWM 占空比为 35%
#define   PWM_Pulse_2     60      //   该互补死区 PWM 占空比为 60%
#define   switch_time     400*2   //   切换占空比时，用于调整切换时间
// 注：为防止杂波产生，switch_time 应为 PWM 周期的倍数。此例 PWM 周期 = 400us
// 故切换时间为 400*2 us

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

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

LPWMG2DTL   =   0x00;
LPWMG2DTH   =   PWM_Pulse + dead_zone*2;

LPWMGCUBL   =   0x00;
LPWMGCUBH   =   100;
```

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

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

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

下图显示了不同条件下的波形。

1. 固定占空比中的 PWM 波形：

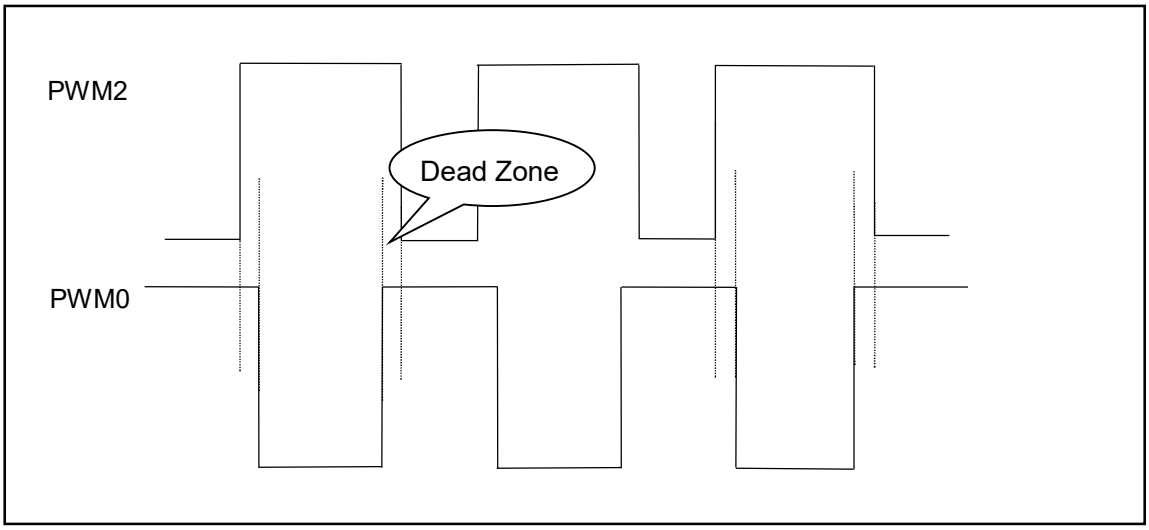


图.11: 互补死区 PWM 波形

2. 切换两个占空比时的 PWM 波形：

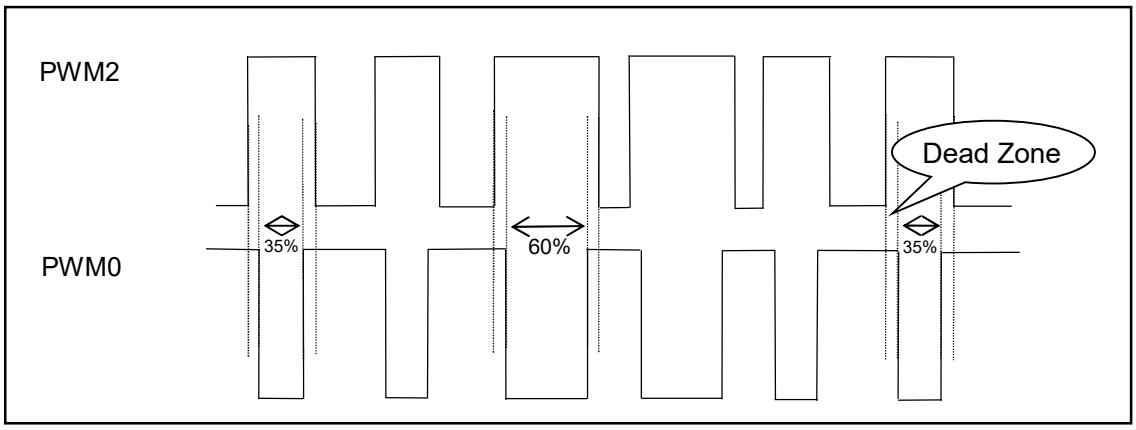


图 12: 互补死区的 PWM 波形

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

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

5.9 看门狗计数器

看门狗定时器（WDT）是一个计数器，时钟来自 ILRC。看门狗定时器可通过上电复位或 wdreset 命令随时清零。通过设置 misc 寄存器，可以选择四种不同的看门狗定时器超时周期，它们是：

- 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，MFU901 将复位并重新运行程序。看门狗时序图如图 13 所示。

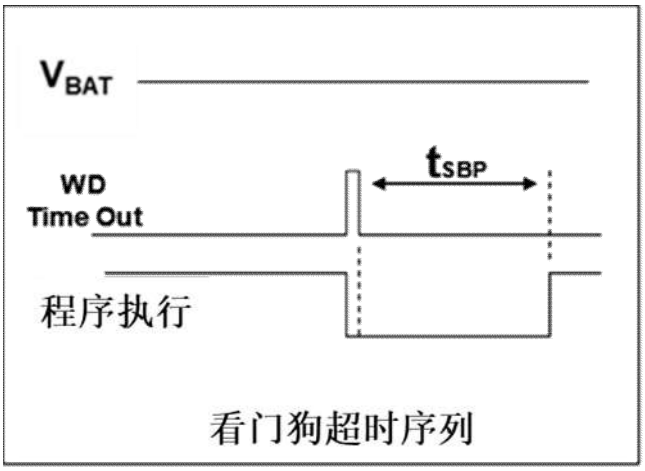


图.13: 看门狗超时溢出时序图

5.10 中断

MFU901 有 7 个中断源：

- 外部中断源 PA0/PA7/PB5
- 外部中断源 PB0/PB6/PA3
- ADC 中断
- Timer16 中断
- LPWMG 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 14 所示。所有的中断请求标志位是由硬件置位并且并通过软件写寄存器 intrq 清零。中断请求标志设置点可以是上升沿或下降沿或两者兼而有之，这取决于对寄存器 integs 的设置。所有的中断请求源最后都需由 engint 指令控制（启用全局中断）使中断运行，以及使用 disgint 指令（停用全局中断）停用它。

中断堆栈与数据存储器共享，其地址由堆栈寄存器 sp 指定。由于程序计数器是 16 位宽度，堆栈寄存器 sp 位 0 应保持 0。此外，用户可以使用 pushaf 指令存储 ACC 和标志寄存器的值到堆栈，以及使用 popaf 指令将值从堆栈恢复到 ACC 和标志寄存器中。由于堆栈与数据存储器共享，在 Mini-C 模式，堆栈位置与深度由编译程序安排。在汇编模式或自行定义堆栈深度时，用户应仔细安排位置，以防地址冲突。

注：外部中断源可通过代码选项中的中断 Src0 或中断 Src1 进行切换。

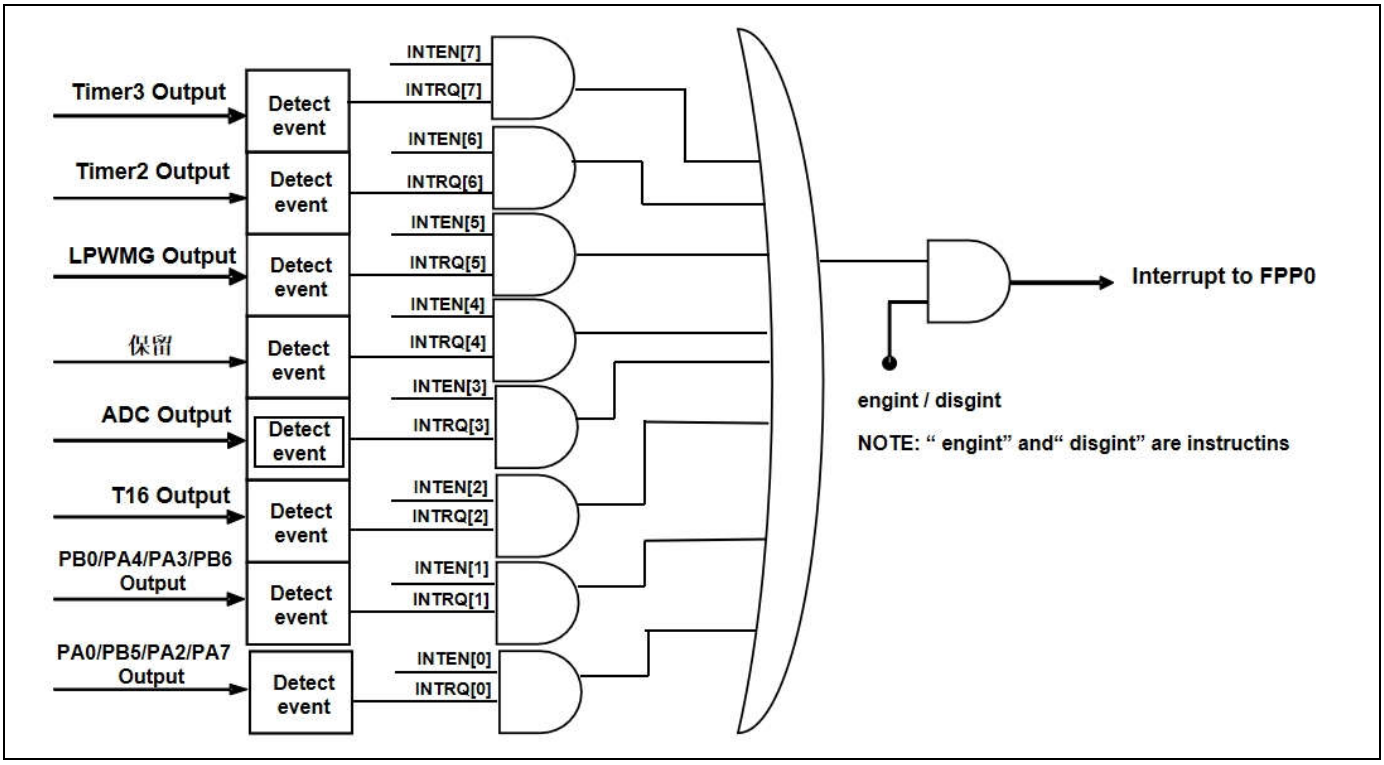


图.14: 中断控制器硬件框图

一旦发生中断，其具体工作流程将是：

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

在中断服务程序中，可以通过读寄存器 intrq 知道中断发生源。

注意：即使 INTEN 为 0，INTRQ 还是会被中断发生源触发。

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

- 从 ***sp*** 寄存器指定的堆栈内存自动恢复程序计数器。
- 新的 ***sp*** 将被更新为 ***sp-2***。
- 全局中断将自动启用。

中断返回后，下一条指令将是中断前的原始指令。

用户必须预留足够的堆栈内存以存中断向量，一级中断需要两个字节，两级中断需要 4 个字节。下面的示例程序演示了如何处理中断，请注意，处理一级中断和 ***pushaf*** 总共需要四个字节堆栈内存。

```
void        FPPA0       (void)
{
    ...
    $  INTEN   PA0;          //  INTEN =1; 当 PA0 准位改变，产生中断请求
    INTRQ  =  0;             // 清除 INTRQ
    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
    }
```

5.11 省电和掉电

MFU901 有三个由硬件定义的操作模式，分别为：正常工作模式，电源省电模式和掉电模式。正常工作模式是所有功能都正常运行的状态，省电模式(***stopexe***)是在降低工作电流而且 CPU 保持在随时可以继续工作的状态，掉电模式(***stopsys***)是用来深度的节省电力。因此，省电模式适合在偶尔需要唤醒的系统工作，掉电模式是在非常低消耗功率且很少需要唤醒的系统中使用。表 5 显示省电模式(***stopexe***)和掉电模式(***stopsys***)之间在振荡器模块的差异（没改变就是维持原状态）。

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

表 5:省电模式和掉电模式在振荡器模块的差异

5.11.1 省电模式 (“*stopexe*”)

使用“***stopexe***”指令进入省电模式时，仅系统时钟被禁用，其余所有振荡器模块仍处于激活状态。对于中央处理器（CPU）而言，它会停止执行；然而，对于 16 位硬件定时器（Timer16）来说，如果其时钟源不是系统时钟，计数器会继续计数。当 Timer16 的时钟源是内部高速 RC 振荡器（IHRC）或内部低速 RC 振荡器（ILRC）模块时，“stopexe”的唤醒源可以是输入输出引脚的触发信号，或者是 Timer16 计数到设定值。当 tm2c/tm3c 的 NILRC（可能是某种特定的低功耗振荡器相关设置）唤醒时，从输入引脚唤醒可被视为正常执行的延续。有关省电模式的详细信息如下：

- 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 内存关闭。
- Timer 计数器：若 Timer 计数器的时钟源是系统时钟或其相应的时钟振荡器模块被停用，则 Timer 停止计数；否则，仍然保持计数。（其中，Timer 包含 Timer16，TM2，LPWMG0，LPWMG1，LPWMG2）。
- 唤醒源：
  a. IO Toggle 唤醒：IO 在数字输入模式下的电平变换（*PAC* 位是 0，*PADIER* 位是 1）。
  b. Timer 唤醒：如果计数器 (Timer)的时钟源不是系统时钟，则当计数到设定值时，系统会被唤醒。
  c. TM2C 唤醒（使用 NILRC 作时钟源）

举例使用 Timer16 唤醒省电模式 “***stopexe***”：

```
$ T16M      ILRC, /1, BIT8          // Timer16 setting
...
WORD        count =     0;
STT16       count;
stopexe;
...
```

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

5.11.2 掉电模式（“stopsys”）

掉电模式是关闭所有振荡模块的深度省电状态。通过使用“stopsys ”指令，该芯片将直接进入 掉电模式。下面显示了发出“stopsys ”指令时 MFU901 的详细内部状态：

- IHRC 和 ILRC 模块关闭
- NILRC 关闭
- MTP 内存被关闭
- SRAM 和寄存器内容保持不变
- 唤醒源：设定为数字模式（PADIER 对应位为 1）的 IO 切换。

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

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

5.11.3 唤醒

进入省电或掉电模式后，MFU901 可通过 Timer2 与 Timer3 拨动 IO 引脚或 NILRC 唤醒恢复正常工作，定时器中断仅适用于 Power-Save 模式。表 6 显示 STOPSYS 和 STOPEXE 唤醒源的不同之处。

掉电模式(stopsys)和省电模式(stopexe)在唤醒源的差异			
	IO 引脚切换	Timer 唤醒	使用 Timer2 与 Timer3 唤醒 NILRC
STOPSYS	是	否	是
STOPEXE	是	是	是

表 5: 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 MFU901，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，MFU901 提供快速唤醒功能，透过 misc 寄存器选择快速唤醒大约 45 个 ILRC 时钟周期。

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

表 6: 快速/正常唤醒之间的唤醒时间差异

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

5.12 IO 引脚

通过配置数据寄存器（***pa, pb, pc***）、控制寄存器（***pac, pbc, pcc***）和拉高/拉低电阻（***paph/papl, pbph/pbpl, pcph/pcpl***），所有引脚可以独立设置为两种状态输出或输入。所有这些引脚都具有具有 CMOS 电平的施密特触发器输入缓冲器和输出驱动器。当它设置为低输出时，拉高电阻会自动关闭。如果用户想读取 pin 状态，请注意在读取数据端口之前应将其设置为输入模式；如果用户在设置为输出模式时读取数据端口，则读取数据来自数据寄存器，而不是来自 IO。例如，表 7 显示了端口 A 的位 0 的配置表。IO 缓冲器的硬件图也如图 15 所示。

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

表 7: PA0 设定配置表

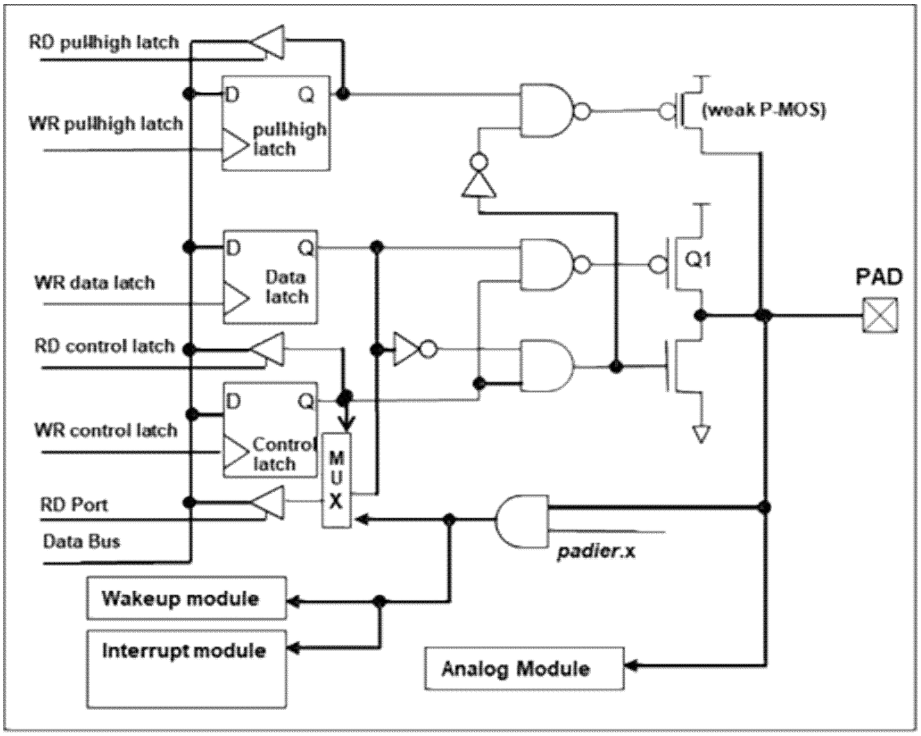


图.15: IO 引脚缓冲区硬件图

所有 IO 引脚结构相同。对于选择模拟功能的引脚，寄存器 ***padier***、***pbdier*** 和 ***pcdier*** 中的相应位应设置为低，以防止漏电流。当 MFU901 进入掉电或省电模式时，每个引脚都可以通过切换状态来唤醒系统。因此，需要唤醒系统的引脚必须设置为输入模式，并将寄存器 ***padier***、***pbdier*** 和 ***pcdier*** 的相应位设置为高电平。同样，当 PA0 用作外部中断引脚时，***padier***.0 也应设置为高电平。

5.13 复位和 LVR

5.13.1 复位

复位 MFU901 的原因很多，一旦复位，MFU901 中的大部分寄存器将被设置为默认值，一旦出现异常情况，应重新启动系统，或将程序计数器跳转到地址 0x00。

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

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

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

5.13.2 LVR 复位

根据代码选项，有多种不同的 LVR 复位级别。通常，用户选择的 LVR 复位级别与工作频率和电源电压有关。

5.14 充电器

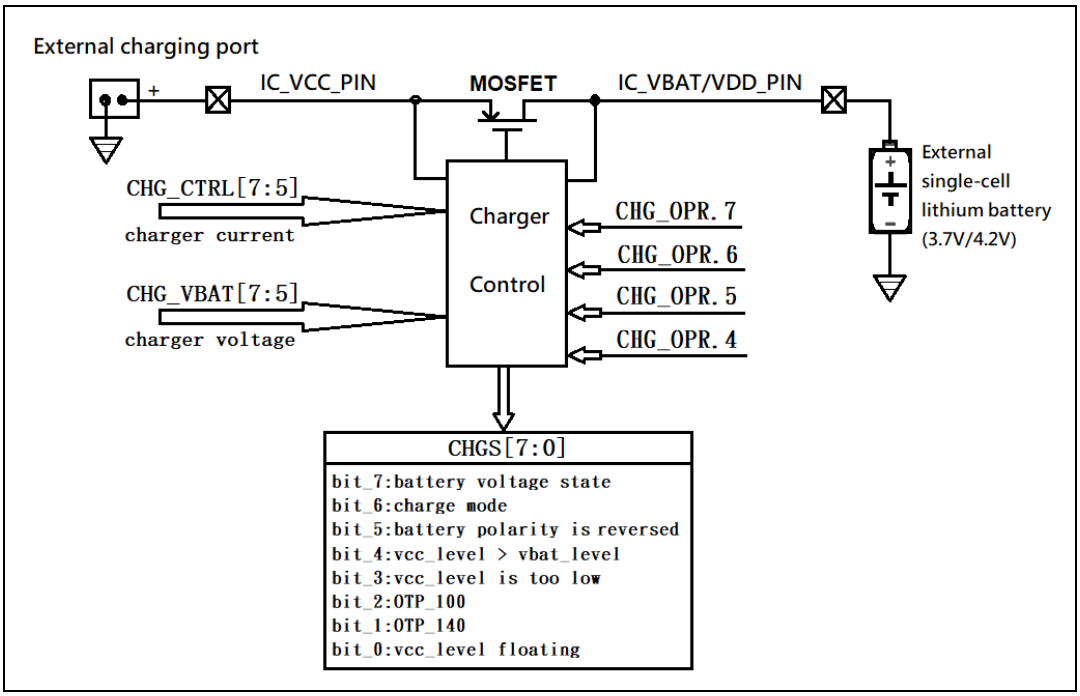
MFU901 内置硬件充电器。该充电器为完全恒流/恒压线性充电，用于单节锂离子电池充电管理。由于采用内部 MOSFET 结构，因此无需外部感应电阻或阻断二极管，最大充电电流可达 500 mA。充电器在供电时自动工作，无需 MCU 程序进行启动设置，这意味着它可以在出厂默认状态下管理电池充电。

用户可以通过 CHG_CTRL[7:5]的三个位来设置或调整充电电流。可设置的充电电流有 4 组，最大 500mA，最小 125mA。

MCU 程序可读取寄存器 CHG_TEMP[6] 来判断充电器工作状态。读取寄存器 CHG_TEMP[4] 可用于判断充电 Vcc 电压是否大于 Vbat。读取寄存器 CHG_TEMP[3] 可判别充电 Vcc 电压是否过低至使充电器自动关闭。当 CHG_TEMP[4:3] = 0b00 时可判读为充电接口已接入，且充电电压正常。

这款充电器还集成了一个充电过热保护电路，并且充电过热保护有两种可选温度：100 摄氏度和 140 摄氏度。可以通过写入 CHGS[2:1]来控制充电过热保护功能。充电过热保护还可通过写入 CHG_OPR[6:5] 进行控制，并且通过读取 CHGS[2:1]能够判断是否触发了充电过热保护。 MFU901 充电器的充电电压和电流已在工厂进行校准，校准值被写入系统参数保护区域。当微控制单元（MCU）成功上电并执行.Adjust_IC 宏指令时，充电器的电压/电流工厂校准值将被填充到 CHG_TRIM / CHG_CUR 寄存器中，然后充电器的充电电压和电流将按照工厂校准值进行测量。当 MCU 的微控制可编程存储器（MTP）为空、没有程序或者上电失败时，充电器的充电电压和电流将处于未校准状态。不建议用户重写 CHG_Trim / CHG_CUR 寄存器。关于禁用 Adjust_IC 指令或编写汇编代码的问题，可以与现场应用工程师（FAE）进行讨论。如果想要微调充电器的充电电压和电流，可以使用 ReLoad_Vbat BGTRIM 和 ReLoad_ChargerCURTRIM 宏指令。

MFU901 使用模数转换器（ADC）来检测电池电压 Vbat。将 BG2V4 或 BG2V68 设置为 ADC 的参考电压。ADC 通道选择通道 F，并将 3/8 VDD 作为待测试电压，这样就可以获取 Vbat 的电压值。



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

卸下输入电源（墙壁插座或 USB 电源）后，MFU901 充电器 IP 自动进入低电流状态，使电池漏电流降至低于 2uA。

充电器在设计上也俱备其他功能如 欠压锁定、自动充电和涓流充电模式。

5.14.1 热限值

充电器具备热反馈调节充电电流以限制在高功率操作或高环境温度。当 MFU901 IC 内部温度上升超过约 90°C 的预设值，内部热反馈回路将减少编程的充电电流。此功能可保护 MFU901 不受过高温度的影响，并允许用户在不损坏 MFU901 的情况下推动给定电路板的功率处理能力极限。充电电流可根据典型（非最坏情况）环境温度进行设置，确保充电器在最坏情况下自动降低电流。

5.14.2 功耗

导致 MFU901 通过热反馈降低充电电流的条件，可以通过考虑集成电路（IC）中的功耗来近似得出。几乎所有这些功耗都是由内部金属氧化物半导体场效应晶体管（MOSFET）产生的其计算值约为：

$$P_D = (V_{VCC} – V_{VBAT}) • I_{VBAT}$$

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

$$T_A = 90°C – P_D\theta_{JA}$$
$$T_A = 90°C – (V_{VCC} – V_{VBAT}) • I_{VBAT} • \theta_{JA}$$

示例：MFU901 由 5V USB 供电，通过编程向放电锂离子电池提供 400mA 满标度电流，电压为 3.75V。假设 θ_{JA} 为 150°C/W（参见 PCB 布局注意事项），MFU901 开始降低充电电流的环境温度约为：

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

T_A = 15°C MFU901 可在环境温度高于 15°C 时使用，但充电电流将从 400mA 降低。给定环境温度下的近似电流约为：

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

使用环境温度为 60°C 的前一个示例，充电电流将减少至约为：

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

$$I_{VBAT} = 160mA$$

注意，应用 MFU901 时不要在最热条件下进行设计，因为当结温达到约 90°C 时，IC 将自动降低功耗。

5.14.3 热温条件

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

5.14.4 EPAD

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

5.15 模拟-数字转换器(ADC) 模块

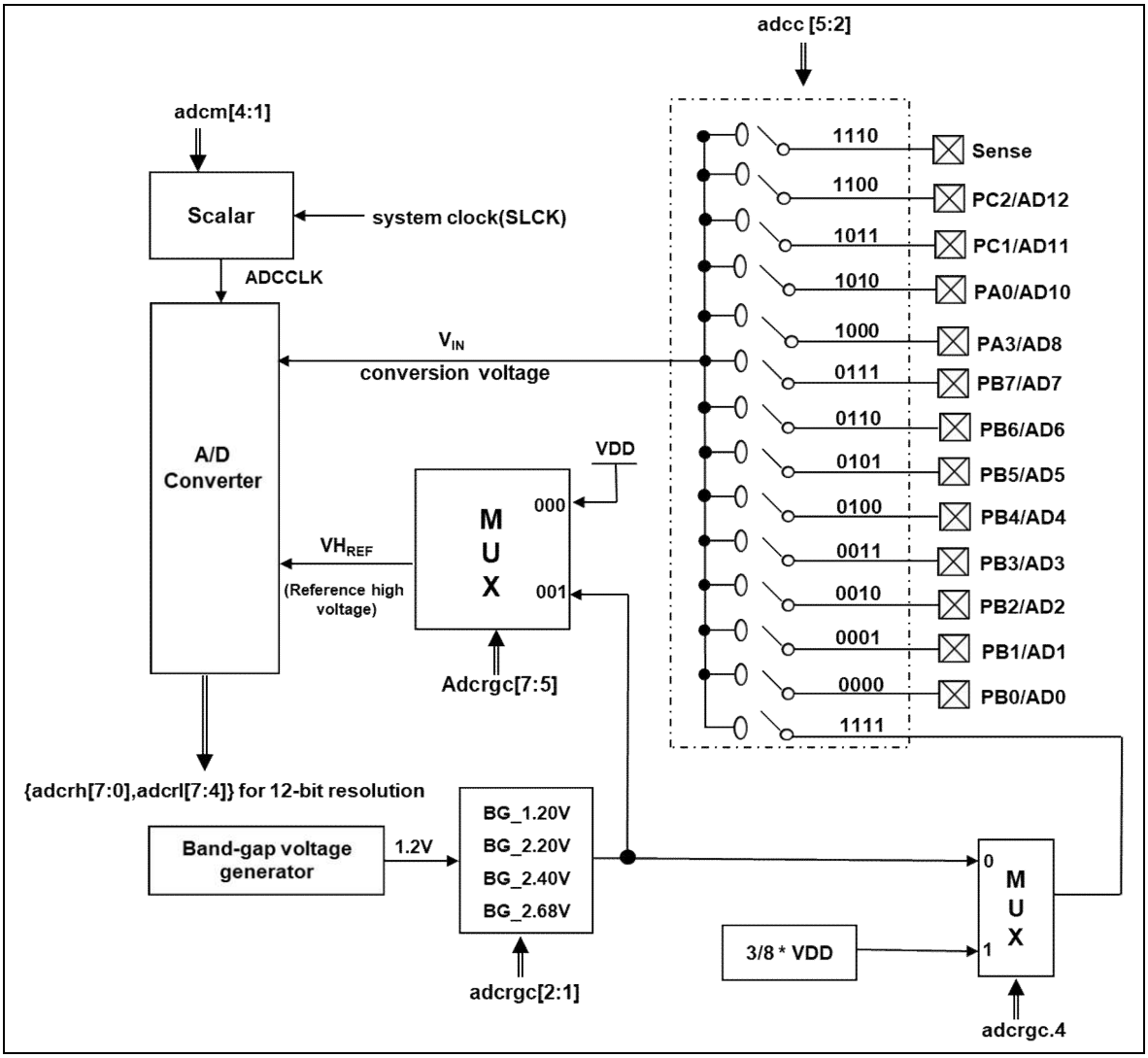


图.16: ADC 模块框图

当使用 ADC 模块时有 8 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 结果寄存器（***adcrh, adcrl***）
- 端口 A/B/C 数字输入启用寄存器（***padier***, ***pbdier, pcdier***）

如下是在使用 ADC 转换的设置步骤：

(1) 通过寄存器 ***adcrgc*** 配置参考高电压

(2) 通过 ***adcm*** 寄存器配置 AD 转换时钟信号

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

(4) 通过 ***adcc*** 寄存器选择 ADC 输入通道

(5) 通过 ***adcc*** 寄存器启用 ADC 模块

(6) 启用 ADC 模块之后，延迟一段时间

条件 1：使用 bandgap 1.2V 或 2.2V/2.4V/2.68V 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 200 个 AD 时钟已经超过 1ms，那么延迟时间只需要 200 个 AD 时钟即可。当启用内部 BG1.2V 或 2.2V/2.4V/2.68V 为参考高电压时，必须保证 IHRC 为开启状态。

条件 2：没有使用任何 bandgap1.2V 或 2.2V/2.4V/2.68V 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，而不是系统时钟（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。

5.15.1 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 17 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。

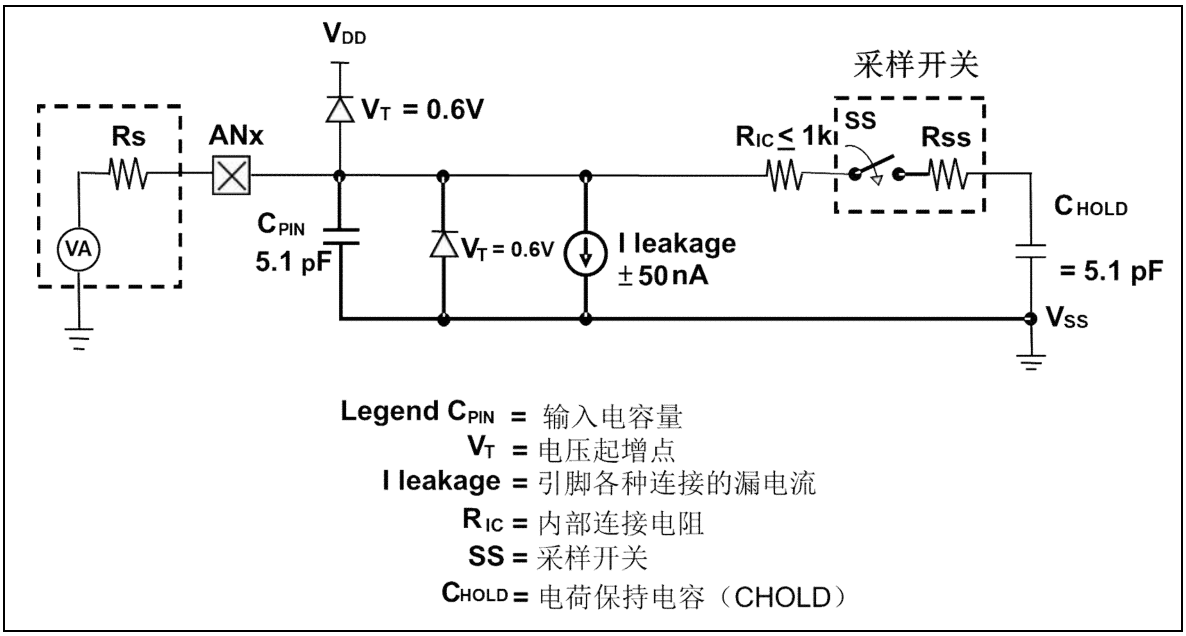


图.17: 模拟输入模型

在开始 AD 转换之前，应满足所选模拟输入信号的最小信号采集时间，ADCLK 的选择必须满足最小信号采集时间。

5.15.2 选择参考高电压

ADC 参考电压通道可通过寄存器 ***adcrgc*** 的位[7:5]选择，其选项可以是 VDD 或 bandgap 参考电压。bandgap 基准电压由 ADCRG 位 [2:1] 设置决定。

5.15.3 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 TACQ 是 ADCLK 的一个时钟周期，所以 ADCLK 必须满足这一要求，建议 ADC 时钟周期是 2us。

5.15.4 配置模拟引脚

有 15 个模拟信号可供选择用于 AD 转换，其中 13 个模拟输入信号来自外部引脚，1 个来自 Sense，1 个来自 bandgap 基准电压或 3/8*VDD。内部带隙基准有 4 个电压电平可选，分别为 1.2V、2.2V、2.4V 和 2.68V。外部引脚为避免数字电路产生漏电流，用于模拟输入的引脚应禁用数字输入功能（将寄存器 ***padier / pbdier / pcder*** 的相应位设置为 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) 将所选引脚设置为输入模式
(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。

5.15.5 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_X0000;      //  PB0 ~ PB3 作为输入
PBPH    =   0B_XXXX_X0000;      //  PB0 ~ PB3 关闭弱上拉电阻
PBDIER  =   0B_XXXX_X0000;      //  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;                 //  参考电压是 VDD,
```

下一步，延迟 400 us（ADCLK = 500KHz，200 × ADCLK = 400 us），如下例所示：

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

注意：如果使用内部参考高电压（如 1.2V 带隙电压），延迟时间必须大于 1ms。

```
$   ASDCRGC   BG, BG_2V;        //  AD 参考电压为 2.0V
.Delay 4*1010;                  //  如果系统时钟 = 4MHz
                                //  延迟时间必须大于 1ms
```

请注意：如果使用 1.2V 带隙电压作为 ADC 输入通道，延迟时间必须大于 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;                      //    如果系统时钟 =4MHz
                                    //    延迟时间必须大于 1ms
```

然后，启动 ADC 转换:

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

最后，当 AD_DONE 为高电平时，可以读取 ADC 结果:

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

可以使用以下方法关闭 ADC

```
$   ADCC    Disable;
or
            ADCC      =     0;
```

5.15.6 如何计算 Vbat 电压

对于 MFU901，可以选择 VDD 和 Bandgap 电压作为 ADC 的 VREF。当 VDD 电压由电池提供时，如果要测量电池电压，可以将 ADC 基准通道设置为 BG2V4 或 BG2V68，选择通道 F 作为 ADC 输入通道，并将其设置为 3/8VDD。

5.16 感应功能

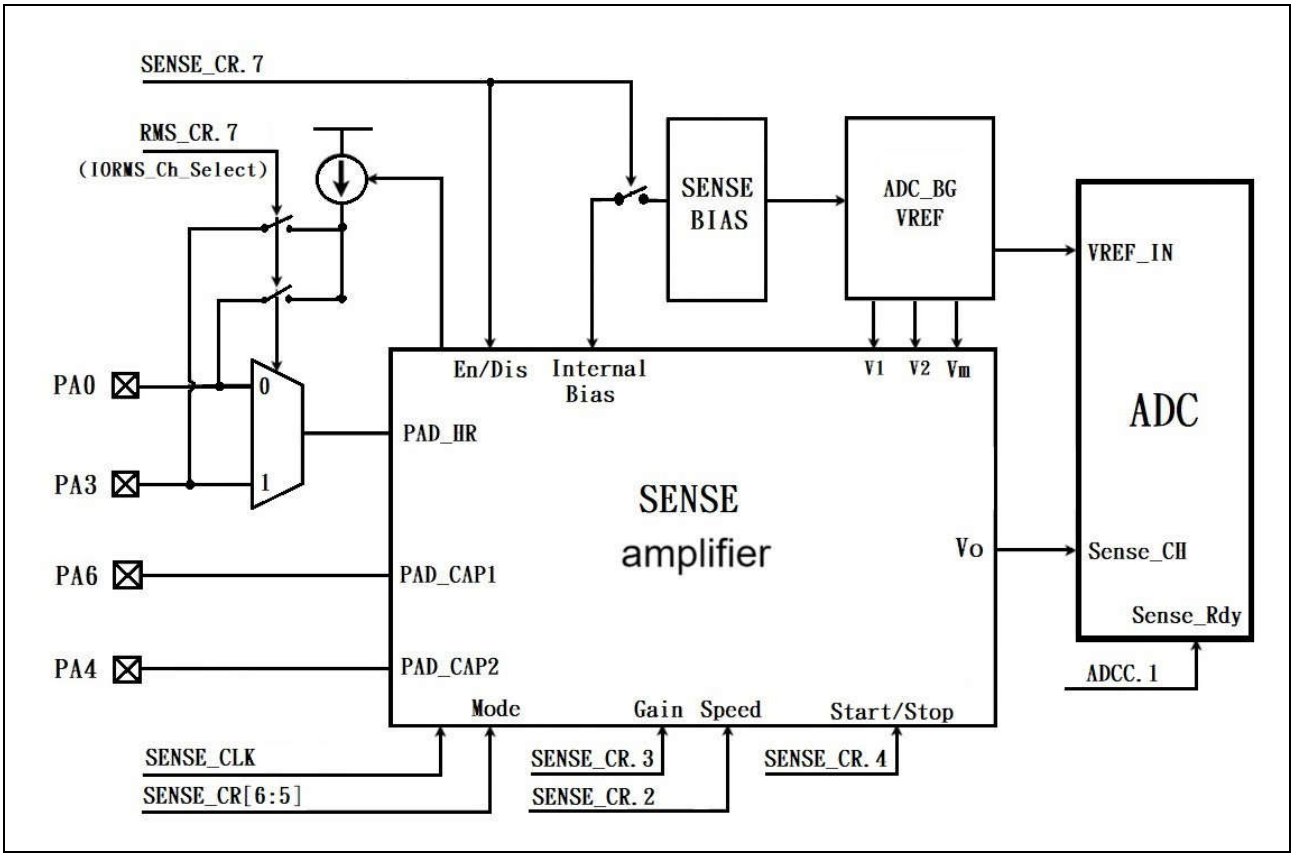


图 18: Sense 模块

MFU901 中的 Sense 模块具有两个主要功能：麦克风电容检测（MCS）和加热电阻检测（HRS）。
MCS 的功能是获取外部的无源电容，可用于测量电容式麦克风；HRS 的功能是获取外部的电阻，可用于加热线的电阻检测。Sense 模块会将外部的电容或电阻转换为电压信号，然后通过内部的 ADC 模块将其转换为数字信号。

使用 Sense 模块检测电容和电阻时，有几个寄存器需要配置。它们是:

- ADC 控制寄存器（***adcc***）
- ADC 调整控制寄存器（***adcrgc***）
- ADC 模式寄存器（***adcm***）
- ADC 结果寄存器（***adcrh***, ***adcrl***）
- 端口 A 数字输入使能寄存器（***padier***）
- SENSE 控制寄存器（***sense_cr***）
- SENSE 偏置电压寄存器（***sense_bias***）
- HRS 控制寄存器（***rms_cr***）

以下是 SENSE 与 ADC 转换过程的步骤：

(1) 通过寄存器 *adcrgc* 配置基准高电压。
(2) 通过寄存器 *adcm* 配置 AD 转换时钟信号。
(3) 通过寄存器 *padier* 配置模拟输入引脚。
(4) 通过寄存器 *sense_cr* 设置感应应使能与模式。
(5) 通过 *adcc* 寄存器启用 ADC 模块，选择感应信道和感应模式。
(6) 延迟至少 5 微秒（5μs）。
(7) 将 ADCC.1 设置为 1（ADC 切换至感应模式）。
(8) 执行 AD 转换并检查 ADC 转换数据是否已完成。
(9) 执行 AD 转换，并检查 ADC 转换数据是否完成。
将 addc.6 设为 1 以启动 AD 转换，并检查 addc.6 是否为 “1”。
(10) 从 ADC 寄存器中读取转换结果。
(11) 通过 *sense_cr* 寄存器关闭感应功能。

5.16.1 电容感应：（MCS 模式）

MFU901 的电容检测（麦克风电容检测，MCS）支持两种类型的无源电容检测：互电容（Mutual-Capacitor）和 MEMS 电容（MEMS-Capacitor）。

在互电容检测模式下，适用于 10pF ~ 24pF 的负载电容，负载电容的两个引脚分别连接至 PA4 和 PA6 引脚。

在 MEMS 电容检测模式下，适用于约 1.4pF 的负载电容，负载电容的两个引脚同样分别连接至 PA4 和 PA6 引脚。

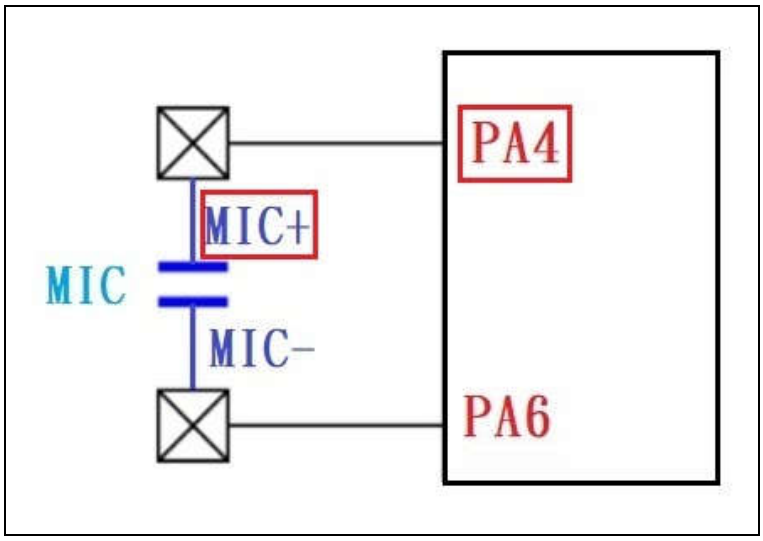


图 19: MIC 在 MCS 模式下的连接图

5.16.2 电容感应偏置校准：

在该应用中，MIC 元件的电容值在以下三种状态下应有所不同：

1. **未使用时的电容值（Sense_ADC 约为 ADC 分辨率的一半刻度）**
2. **吸气时的电容值（Sense_ADC 将随电容值的变化而变化）**
3. **吹气时的电容值（Sense_ADC 将随电容值的变化而变化）**

电容感应需要在 MIC 空转且无外部气流压力的情况下进行校准。感测偏置校准的目的是将 MIC 电容值转换为合适的 ADC 值，该值通常设置为 ADC 分辨率的一半。调整 sense_bias 寄存器的校准值，使传感模块输出的电压接近 ADC_Vref 的半量程电压。

感应偏压校准可在以下两种条件下进行（校准应在麦克风空闲且无外部气流压力的情况下执行）：

1. 芯片上电并完成程序初始化后，执行感应偏压校准流程；
2. 程序在完成特定操作设置后，进入重新进行感应偏压校准流程。

感应偏压校准流程的步骤如下：

步骤 1：将 SENSE_BIAS 寄存器设置为 0xFF，读取 MIC_SENSE 数值，并执行步骤 2。

步骤 2：判断 MIC_SENSE 数值是否接近 ADC 参考电压的一半（对于 11 位 ADC，为 1024）。
若判断结果为“是”，执行步骤 4；若判断结果为“否”，执行步骤 3。

步骤 3：减少 SENSE_BIAS 寄存器的设定值，读取 MIC_SENSE 数值，并重复执行步骤 2。

步骤 4：感应偏压校准完成。

感应偏压校准程序代码如下：

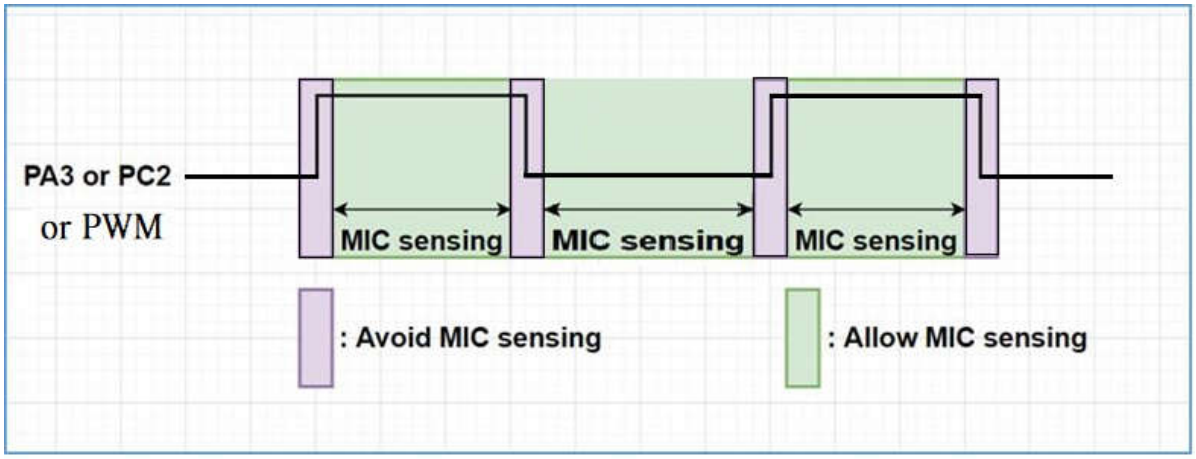
```
// 麦克风偏压电压校准
// 初始化 SENSE_BIAS 并将 Vo 校准至 0.5 * ADC_Vref
void SENSE_BIAS_Trim(void)
{
    byte all_tune = 0xFF;
    byte SENSE_VREF_Buf = all_tune;              // SENSE_BIAS 寄存器缓冲区
    $ ADCRGC BG, ADC_BG, BG_2V68;                // ADC 参考电压 = BG 2.68V
    $ ADCM    /(2000000/500000);                 // ADC 时钟 = 500K，系统时钟 = 2M
@@.Begin:
    SENSE_BIAS = all_tune;
    $ SENSE_CR Enable, SENSE_MIC_MODE;           // 设置 Sense 为麦克风模式
    $ ADCC   Enable, Sense, Sense_Rdy;           // 选择 ADC 信道为 Sense 信道
    .delay 10;                                   // 必须延迟 5 微秒，系统时钟=2M
    AD_START = 1;                                // 启动 ADC
    SENSE_CR.Start = 1;                          // 启动 Sense ADC
    while(!AD_DONE) {}                           // 等待 ADC 测量完成
    Sense_Value$1 = ADCRH;
    Sense_Value$0 = ADCRL;
```

```
    Sense_Value >>= 5;                          // ADC 值采用 11 位精度计算
    SENSE_CR.Stop = 1;                          // 停止 Sense ADC

    // 将 Sense ADC 数据与 0.5*ADC 参考电压（11 位 ADC 数据，即 1024）进行比较，
    // 并找到最接近 0.5*ADC 参考电压的组合
    static word min_Detal = 0xFFFF;
    static word Diff;

    Diff = 1024 - Sense_Value;
    T1SN FLAG.1;      // CF: 运算进位借位符号
    goto @F;
    Diff = -Diff;
@@:
    if(Diff <= min_Detal)                       // 获取最接近 1024 的配置
    {
        min_Detal = Diff;                       // 更新最小差值及寄存器配置
        SENSE_VREF_Buf = all_tune;
    }
    if (all_tune != 0)
    {
        all_tune--;                             // 重新配置 SENSE_VREF 寄存器
        goto @B.Begin;
    }
@@.End:
    // 配置最佳的 SENSE_BIAS，使 Sense_ADC 最接近 1024
    SENSE_BIAS = SENSE_VREF_Buf;
}
```

5.16.3 电容感应说明：

MIC 检测是一种高精度转换，很容易受到外部和其他数字信号波形的干扰。因此，MIC 检测应避免在 PWM 波形的正边和负边采样。建议 MIC 传感与 PWM 同步，并固定在 PWM 波的高电平或低电平处进行采样。

(1) 避免在 PA3/PC2 切换边沿期间检测 MIC。
(2) 避免在 PWM 边沿期间进行 MIC 检测。
(3) 如下图所示。



5.16.4 电阻检测：（加热电阻检测 HRS）

MFU901 电阻检测（加热电阻检测，HRS）支持对一般加热丝电阻值的检测（0.5Ω 到 1.5Ω）。电阻检测器（HRS）的工作原理是，Sense 模块会从 PA0 或 PA3 引脚输出恒定电流 50mA 至外部负载电阻（通常为加热丝）。在外部加热丝电阻上会产生一个小电压，该电压经过内部的感测放大器放大后，再与 Vm（1.2V）进行差分，得到感测输出电压 Vo。Vo 将被输出到 ADC 模块的 Sense 通道，并由 ADC 模块转换为数字信号。RMS 测量通道可通过 CodeOption 设置，或由用户程序通过 RMS_CR 寄存器的第 7 位进行配置。

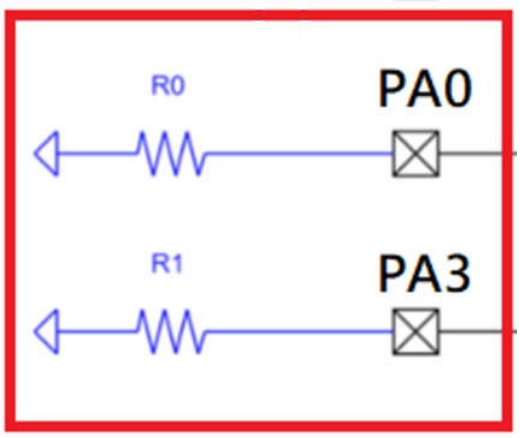


图 20： 外部加热线连接图

HRS 计算公式	
$V_o = V_m - Gain * I_{rms} * R$	
$\frac{V_o}{ADC\_Vref} = \frac{ADC\_Value}{2^n}$	
公式参数介绍	
Vo	是感测块输出电压
Vm	固定为 1.2V。 ADC_Vref 由寄存器 ADCRGC[2:1] 位控制 ADCRGC[2:1] = (BG_1V2,BG_2V2,BG_2V4,BG_2V68)
Gain	固定为 x4
Irms	芯片内部可调电流源，由寄存器 RMS_CR[5:0]控制 不建议客户手动调节。
R	是加热丝的电阻值
2n	是 ADC 的分辨率，ADC 的最大分辨率为 12BIT
ADC_Value	是 12 位 ADC 采集的数值，由 ADCRH[7:0] 和 ADCRL[7:4] 组成
ADC_Result[11:0] = ADRCRH[7:0] | ADCRL[7:5]	

示例:

- 在这种情况下，外部电阻负载 R 为 1 Ω
- 内部恒定电流，出厂时校准为 50 mA
- sense_gain 设为 4
- ADC VREF at 2.40V
- $V_o = 1.2V - 4 \cdot 50mA \cdot 1\Omega = 1.00V$
- $SO \ldots 12\text{bits}\ ADC_{cout} = \frac{1.00V}{1LSB} = \frac{1.00V}{\frac{2.40V}{4096}} \approx 1706$

MIC 和 HRMS 的 Sense_ADC 转换代码如下：

```
// 获取 Sense 的值
void Get_Sense_Data(void)
{
// 在采集 MIC 数据之前，需要关闭 PWM 输出引脚，防止干扰 MIC 采样数据。
// ---- PWM 设置可由用户根据实际情况修改 ----
// byte LPWMG0C_BUF = LPWMG0C;
// byte LPWMG1C_BUF = LPWMG1C;
// $ LPWMG0C;
// $ LPWMG1C;

Byte tmp = SENSE_CR & 0b_0_11_0_0000;
if (tmp == 0b_0_10_0_0000)
    $ ADCRGC BG, BG_2V4;          // HRMS 模式
else
    $ ADCRGC BG, BG_2V68;         // MIC 模式

    $ ADCC   Enable, Sense, Sense_Rdy;  // 选择 ADC 通道 Sense 通道，Sense 模式并使能
    .delay 10;                          // 延迟 5 微秒，系统时钟=2M
    AD_START = 1;                       // 启动 ADC
    SENSE_CR.Start = 1;                 // 启动 Sense ADC
    while(!AD_DONE) {}                  // 等待 ADC 测量完成
    // ---- PWM 设置可由用户根据实际情况修改 ----
    // LPWMG0C = LPWMG0C_BUF;
    // LPWMG1C = LPWMG1C_BUF;

    Sense_Value$1 = ADCRH;
    Sense_Value$0 = ADCRL;
    Sense_Value >>= 5;                  // ADC 值采用 11 位精度
    SENSE_CR.Stop = 1;                  // 停止 Sense ADC
}
```

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。请勿使用。
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 01x：保留 100：保留 101：保留 110：ILRC/4 111：ILRC（默认）	000：IHRC/16 001：IHRC/8 010：ILRC/16（仿真器不支持） 011：IHRC/32 100：IHRC/64 110：保留 1x1：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	使能 Timer3 中断。0/1：停用/启用。
6	0	读/写	使能 Timer2 中断。0/1：停用/启用。
5	0	读/写	使能 PWMG0 中断。0/1：停用/启用。
4	0	读/写	保留。
3	0	读/写	使能 ADC 中断。0/1：停用/启用。
2	0	读/写	使能 Timer16 溢出中断。0/1：停用/启用。
1	0	读/写	使能 PB0/PA3/PB6 中断。0/1：停用/启用。
0	0	读/写	使能 PA0/PB5/PA7 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：不请求 / 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：不请求 / 请求
5	-	读/写	PWMG0 的中断请求，此位是由硬件置位并由软件清零。 0/1：不请求 / 请求
4	-	读/写	保留
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：不请求 / 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：不请求 / 请求
1	-	读/写	引脚 PB0/PA3/PB6 的中断请求，此位是由硬件置位并由软件清零。 0/1：不请求 / 请求
0	-	读/写	引脚 PA0/PB5/PA7 的中断请求，此位是由硬件置位并由软件清零。 0/1：不请求 / 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：保留 100：IHRC 101：保留 110：ILRC 111：PA0 下降沿（从外部引脚）
4 - 3	00	读/写	Timer16 时钟分频： 00：÷1 01：÷4 10：÷16 11：÷64
2 - 0	000	读/写	中断源选择。当所选择的状态位变化时，中断事件发生。 0：Timer16 位 8 1：Timer16 位 9 2：Timer16 位 10 3：Timer16 位 11 4：Timer16 位 12 5：Timer16 位 13 6：Timer16 位 14 7：Timer16 位 15

6.7 端口 A 下拉控制寄存器 (*papl*), IO 地址= 0x08

位	初始值	读/写	描述
7 - 0	0x00	读/写	端口 A 上拉控制寄存器。这些寄存器是用来控制上拉高端口 A 每个相应的引脚。 0/1：停用/启用 PAPL.4 及 PAPL.6：保留

6.8 端口 B 下拉控制寄存器 (*pbpl*), IO 地址= 0x09

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 A 每个相应的引脚。 0/1：停用/启用。

6.9 端口 C 下拉控制寄存器 (*pcpl*), IO 地址= 0x0A

位	初始值	读/写	描述
7	-	-	保留
6 - 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 A 每个相应的引脚。 0/1：停用/启用

6.10 中断缘选择寄存器 (*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/PA3/PB6 中断缘选择：. 00：上升缘和下降缘都请求中断 01：上升缘请求中断 10：下降缘请求中断 11：保留
1 - 0	00	只写	PA0/PB5/PA7 中断缘选择： 00：上升缘和下降缘都请求中断 01：上升缘请求中断 10：下降缘请求中断 11：保留

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	0x00	只写	启用端口 A 数字输入，以防止该引脚分配为 AD 输入时发生泄漏。选择禁用时，该引脚的唤醒功能也会被禁用。 0 / 1：停用 / 启用 PADIER.4 和 PADIER.6 未被使用。

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	0x00	只写	启用端口 B 数字输入，以防止该引脚分配为 AD 输入时发生泄漏。选择禁用时，该引脚的唤醒功能也会被禁用。 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	-	-	保留
6 - 0	0x00	只写	启用端口 C 数字输入，以防止该引脚分配为 AD 输入时发生泄漏。选择禁用时，该引脚的唤醒功能也会被禁用。 0 / 1：停用 / 启用

注： 详细设置请参阅第 9.2 节。

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。这些寄存器用于定义 Port A 各对应引脚的输入模式或输出模式。 0/1：输入 / 输出 PAC.4 及 PAC.6 未被使用

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。该上拉功能仅在输入模式下有效。 0/1：停用/启用 PAPH.4 及 PAPH.6：保留

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。该上拉功能仅在输入模式下有效。 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	-	-	保留
6 - 0	0x00	读/写	数据寄存器的端口 C。

6.21 端口 C 控制寄存器 (*pcc*), IO 地址= 0x17

位	初始值	读/写	描述
7	-	-	保留
6 - 0	0x00	读/写	端口 C 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

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

位	初始值	读/写	描述
7	-	-	保留
6 - 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。 0/1：停用 / 启用

6.23 ADC 控制寄存器 (adcc), IO 地址= 0x20

位	初始值	读/写	描述
7	0	读/写	启用 ADC 功能。0/1：停用/启用
6	0	读/写	ADC 转换进程控制位： 写入“1”以启动转换。 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,

位	初始值	读/写	描述
			0111：PB7/AD7, 1000：PA3/AD8, 1001：保留, 1010：PA0/AD10, 1011：PC1/AD11, 1100：PC2/AD12, 1110：Sense 1111：Bandgap 电压或 3/8*V_{DD} 其他：保留
1	0	读/写	0：来自 ADC 的 ADC 控制（正常 ADC 模式） 1：Sense_Rdy（Sense_ADC 模式）
0	-	-	保留

6.24 ADC 模寄存器式 (*adcm*), IO 地址= 0x21

位	初始值	读/写	描述
7 - 4	-	-	保留（保留 0）
3 - 1	000	读/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

6.25 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	这三个位用于选择 ADC 基准高压通道的输入信号 000：V_{DD}, 001：ADC Bandgap 其它：保留
4	0	只写	ADC 通道 F 选择器： 0：Bandgap 参电压考 1：3/8*V_{DD}
3	-	-	保留
2 - 1	00	只写	ADC Vref 来着 Bandgap 00：1.2V 01：2.2V 10：2.4V 11：2.68V
0	-	-	保留。请写 0。

6.26 ADC 高位结果寄存器(adcrh), IO 地址= 0x22

位	初始值	读/写	描述
7 - 0	-	只读	这 8 个只读位将 AD 转换结果的[11:4] 位。寄存器的第 7 位是任何分辨率 ADC 转换结果的 MSB。

6.27 ADC 低位结果寄存器 (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

6.28 杂项寄存器 (*misc*), IO 地址= 0x26

位	初始值	读/写	描述
7 - 6	-	-	保留。（保留为 0，以确保未来兼容性）。
5	0	只写	快唤醒功能。 0：正常唤醒。唤醒时间为 3000 个 ILRC 时钟周期（不适用于快速启动） 1：快速唤醒。唤醒时间为 45 个 ILRC 时钟周期。
4	0	只写	使能 VDD/2 LCD bias 电压生成器。 0 / 1：停用/ 开启（ICE 不能实时转换） 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 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：开启/ 停用

位	初始值	读/写	描述
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

6.29 Timer2 控制寄存器 (*tm2c*), IO 地址= 0x28

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 or IHRC *2（由 code option TMx_ source 决定） 0011：保留 0100：ILRC 0101：NILRC 1000：PA0（上升沿） 1001：~PA0（下降沿） 1010：PB0（上升沿） 1011：~PB0（下降沿） 110X：保留 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：周期模式 1：PWM 模式
0	0	读/写	启用 Timer2 反极性输出。 0：停用 1：启用

6.30 Timer2 计数寄存器 (*tm2ct*), IO 地址= 0x29

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

6.31 Timer2 分频寄存器(*tm2s*), IO 地址= 0x2A

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

6.32 Timer2 上限寄存器 (*tm2b*), IO 地址= 0x2B

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

6.33 Timer3 控制寄存器(*tm3c*), IO 地址= 0x2C

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 or IHRC *2 （由 code option TMx_ source 决定） 0011：保留 0100：ILRC 0101：NILRC 1000：PA0（上升沿） 1001：~PA0（下降沿） 1010：PB0（上升沿） 1011：~PB0（下降沿） 110X：保留 其他：保留
3 - 2	00	读/写	Timer3 输出选择。 00：停用 01：PB5 10：PB6 11：PB7

位	初始值	读/写	描述
1	0	读/写	Timer3 模式选择。 0：周期模式 1：PWM 模式
0	0	读/写	启用 Timer3 反极性输出。 0/1：停用/启用

6.34 Timer3 计数寄存器 (*tm3ct*), IO 地址= 0x2D

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

6.35 Timer3 分频寄存器 (*tm3s*), IO 地址= 0x2E

位	初始值	读/写	描述
7	0	只写	PWM 分辨率选择。 0：8 位 1：6 位或者 7 位（由 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。

6.36 Timer3 上限寄存器(*tm3b*), IO 地址= 0x2F

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

6.37 充电电流控制寄存器 (*chg_ctrl*), IO 地址= 0x32

位	初始值	读/写	描述
7	-	-	保留
6 - 5	00	只写	充电电流选择 00：145mA 00：250mA 00：375mA 00：500mA
4 - 0	-	-	保留

6.38 充电电压控制寄存器(*chg_vbat*), IO 地址= 0x33

位	初始值	读/写	描述
7 - 5	0x000	只写	充电电压选择 000：4.2V 001：4.1V 010：4.35V 011：3.6V 100：3.7V 101：3.8V 110：4.24V 111：4.28V
4 - 0	-	-	保留

6.39 充电输出信号寄存器 (*chgs*), IO 地址= 0x34

位	初始值	读/写	描述
7	-	只读	电池电压状态 0 / 1：电池未充满 / 充满。
6	-	只读	充电模式。 0 / 1：正常模式/充电模式。
5	-	-	保留
4	-	只读	Vcc 电压源的状态指示位。可用于判断 Vcc 充电状态。 1：Vcc < V_{BAT} 0：Vcc > V_{BAT}
3	-	只读	充电操作指示位。 1：正在充电。Vcc 过低，充电器关闭。 0：Vcc 正常。
2	-	只读	过温充电保护位（温度低于 90° C）。 读取“0 ”触发过温保护，读取“1 ”则正常。
1	-	只读	过温充电保护位（温度低于 140° C）。 读取“0 ”触发过温保护，读取 “1 ”则正常。
0	-	只读	0/1：VCC 处于悬空状态/处于连接状态。

6.40 充电电流控制寄存器 (*chg_opr*), IO 地址= 0x35

位	初始值	读/写	描述
7	1	只写	启用充电功能模块。0/1：禁用/启用。
6	1	-	保留
5	1	-	保留
4	1	只写	OVP 功能控制 0 / 1：关 / 开
3	0	只写	保留，只能写 0。
2 - 0	-	-	保留

6.41 感应控制寄存器(*sense_cr*), IO 地址= 0x37

位	初始值	读/写	描述
7	0	读/写	0 / 1: 感应功能 禁用/启用
6 - 5	01	读/写	感应模式选择 00：麦克风电容感测（MCS） 01：保留 10：发热电阻感测（HRS） 11：保留
4	0	读/写	0 / 1：感测功能停止/启动。
3	0	读/写	保留
2	0	读/写	感应操作时间 正常/双倍（X2） 0：x1 1：x2
1 - 0	-	-	保留

6.42 检测偏置寄存器 (*sense_bias*), IO 地址= 0x38

位	初始值	读/写	描述
7-6	01	读/写	V2, MCS 模式的感应偏置电压。 (粗调) 00：最小 11：最大
5 - 0	0x20	读/写	V1, MCS 模式的感应偏置电压。. (微调) 00_0000：最小 11_1111：最大

6.43 RMS 控制寄存器(*rms_cr*), IO 地址= 0x39

位	初始值	读/写	描述
7	-	读/写	0 / 1：RMS 通道选择 PA0 / PA3
6 - 0	-	-	保留

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

位	初始值	读/写	描述
7	-	-	保留
6	-	只读	PWMG0 生成器输出状态。
5	0	读/写	PWMG0 输出。 0 / 1：缓冲 / 反相
4	0	读/写	PWMG0 输出选择 0：PWMG0 输出 1：(PWMG0 ^ PWMG1) | (PWMG0 | PWMG1) (by pwmg0c.0)
3 - 1	000	读/写	PWMG0 输出端口选择 000：PWMG0 输出关闭 001：PWMG0 输出到 PB5 010：PWMG0 输出到 PC2 011：PWMG0 输出到 PA0 100：PWMG0 输出到 PB4 其它：保留
0	0	读/写	PWMG0 输出预选择。 0：PWMG0 ^ PWMG1 1：PWMG0 | PWMG1

6.45 PWMG 时钟寄存器 (*pwmgclk*), IO 地址= 0x41

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

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 bit[10:3]。

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

位	初始值	读/写	描述
7 - 5	-	只写	PWMG0 占空比值 bit [2:0]。
4 - 0	-	-	保留

注意：必须先写入 PWMG0 Duty_Value 低位寄存器，然后再写入 PWMG0 Duty_Value 高位寄存器。

6.48 PWMG 计数上限高位寄存器 (*pwmgcubh*), IO 地址= 0x44

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

6.49 PWMG 计数上限低位寄存器 (*pwmgcubl*), IO 地址= 0x45

位	初始值	读/写	描述
7 - 6	-	只写	PWMG0 上限寄存器 Bit[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	-	-	保留
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：缓冲 / 反相
4	0	读/写	PWMG1 输出选择。 0：PWMG1。 1：PWMG2。
3 - 1	000	读/写	PWMG1 端口选择： 000：PWMG1 不输出 001：PWMG1 输出到 PB6 010：PWMG1 输出到 PC3 011：保留 100：PWMG1 输出到 PB7 其它：保留
0	-	-	保留

6.51 PWMG1 占空比高位寄存器(*pwmg1dth*), IO 地址= 0x47

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

6.52 PWMG1 占空比低位寄存器 (*pwmg1dtl*), IO 地址= 0x48

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：必须先写入 PWMG1 Duty_Value 低位寄存器，然后再写入 PWMG1 Duty_Value 高位寄存器。

6.53 PWMG2 控制寄存器 (*pwmg2c*), IO 地址= 0x49

位	初始值	读/写	描述
7	-	-	保留
6	-	只读	PWMG2 生成器输出状态。
5	0	读/写	选择 PWMG2 的输出的结果是否反极性： 0 / 1：缓冲 / 反相
4	0	读/写	PWMG2 输出选择。 0：PWMG2 1：PWMG2/2
3 - 1	000	读/写	PWMG2 输出端口选择： 000：PWMG2 不输出。 001：PWMG2 输出到 PB3 010：PWMG2 输出到 PC0 011：PWMG2 输出到 PA3 100：PWMG2 输出到 PB2 其它：保留
0	-	-	保留

6.54 PWMG2 控制寄存器 (*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]。

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

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) （在无符号数运算中，结果为加法产生进位或减法需要借位。）
AC	辅助进位标志(Auxiliary Carry) （如果在 ALU 运算结果中低位半字节产生了进位，则该位被置为 1。）
M.n	只允许寻址在地址 0~0x3F (0~63) 的位置

7.1 数据传输类指令

mov a, I	移动即时数据到累加器。 例如： *mov* a, 0x0f; 结果： a ← 0fh; 受影响标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
mov M, a	移动数据由累加器到存储器。 例如：*mov* MEM, a; 结果：MEM ← a 受影响标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
mov a, M	移动数据由存储器到累加器。 例如：*mov* a, MEM ; 结果：a ← MEM; 当 MEM 为零时，标志位 Z 会被置位。 受影响标志位：Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
mov a, IO	移动数据由 IO 到累加器。 例如：*mov* a, pa ; 结果：a ← pa; 当 pa 为零时，标志位 Z 会被置位。 受影响标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
mov IO, a	移动数据由累加器到 IO。 例如：*mov* 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	通过使用 index 作为 OTP 地址把数据的高位存进程序存储器给累加器。该操作需要 2T 指令周期。
	例如：*ldtabh index;*
	结果：a ← {bit 15~8 of OTP [index]};
	受影响标志位：『N』Z 『N』C 『N』AC 『N』OV
	应用范例：
	--- 	
	word ROMptr ; // 在 RAM 里面声明指针
	...

```
    mov         a, la@TableA ;      // 指针指向 ROM 里面的 TableA (LSB)
    mov         lb@ROMptr, a ;      // 保存指针给 RAM(LSB)
    mov         a, ha@TableA ;      // 指针指向 ROM 里面的 TableA (MSB)
    mov         hb@ROMptr, a ;      // 保存指针给 RAM(MSB)
    ...
    ldtabh      ROMptr ;            // 存储 TableA MSB 给 ACC (ACC=0X02)
    ...
    TableA :    dc      0x0234, 0x0042, 0x0024, 0x0018 ;
```

---

ldtabl index

通过使用 index 作为 OTP 地址把数据的低位存进程序存储器给累加器。该操作需要 2T 指令周期。
例如： *ldtabl index;*
结果： a ← {bit7~0 of OTP [index]};
受影响标志位：『N』Z 『N』C 『N』AC 『N』OV
应用范例：

```
    word        ROMptr ;            // 在 RAM 里面声明指针
    ...
    mov         a, la@TableA ;      // 指针指向 ROM 里面的 TableA (LSB)
    mov         lb@ROMptr, a ;      // 保存指针给 RAM (LSB)
    mov         a, ha@TableA ;      // 指针指向 ROM 里面的 TableA (MSB)
    mov         hb@ROMptr, a ;      // 保存指针给 RAM (MSB)
    ...
    ldtabl      ROMptr ;            // 存储 TableA LSB 给 ACC (ACC=0x34)
    ...
    TableA :    dc      0x0234, 0x0042, 0x0024, 0x0018 ;
```

---

ldt16 word

将 Timer16 的 16 位计算值复制到 RAM。
例如： *ldt16* word;
结果： word ← 16-bit timer
受影响标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
应用范例：

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

---

stt16 word	将放在 word 的 16 位 RAM 复制到 Timer16。
	例如： *stt16* word;
	结果： 16-bit timer ← word
	受影响标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』
	应用范例：
	`word T16val ; // 定义一个 RAM word`
	`…`
	`mov a, 0x34 ;`
	`mov lb@T16val, a ; // 将 0x34 搬到 T16val (LSB)`
	`mov a, 0x12 ;`
	`mov hb@T16val, a ; // 将 0x12 搬到 T16val (MSB)`
	`stt16 T16val ; // Timer16 初始化 0x1234`
	`…`

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』
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』

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 PFS123 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`

ldxm index, a	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)，必须为 0
	mov hb@RAMIndex, a ; // 将指针存到 RAM (MSB)
	...
	mov a, 0xA5 ;
	idxm RAMIndex, a ; // 0xA5 存入 RAM 地址为 0x5B

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 ;

7.2 算数运算类指令

add a, I	将立即数据与累加器相加，然后把结果放入累加器。 例如： *add* a, 0x0f ; 结果： a ← a + 0fh 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
add a, M	将 RAM 与累加器相加，然后把结果放入累加器。 例如： *add* a, MEM ; 结果： a ← a + MEM 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
add M, a	将 RAM 与累加器相加，然后把结果放入 RAM。 例如： *add* MEM, a; 结果： MEM ← a + MEM 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
addc a, M	将 RAM、累加器以及进位相加，然后把结果放入累加器。 例如： *addc* a, MEM ; 结果： a ← a + MEM + C 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
addc M, a	将 RAM、累加器以及进位相加，然后把结果放入 RAM。 例如： *addc* MEM, a ; 结果： MEM ← a + MEM + C 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
addc a	将累加器与进位相加，然后把结果放入累加器。 例如： *addc* a ;

	结果： a ← a + C 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
addc M	将 RAM 与进位相加，然后把结果放入 RAM。 例如： *addc* MEM ; 结果： MEM ← MEM + C 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
nadd a, M	将累加器的负逻辑(2补码)与RAM相加，然后把结果放入累加器。 例如: *nadd* a, MEM ; 结果: a ← ∓a + MEM 受影响的标志位: Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
nadd M, a	将RAM的负逻辑(2补码)与累加器相加，然后把结果放入RAM。 例如: *nadd* MEM, a ; 结果: MEM ← ∓MEM + a 受影响的标志位: Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
sub a, I	累加器减立即数据，然后把结果放入累加器。 例如： *sub* a, 0x0f; 结果： a ← a - 0fh (a + [2’s complement of 0fh]) 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
sub a, M	累加器减 RAM，然后把结果放入累加器。 例如： *sub* a, MEM ; 结果： a ← a - MEM (a + [2’s complement of M]) 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
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』
subc a, M	累加器减 RAM，再减进位，然后把结果放入累加器。 例如： *subc* a, MEM; 结果： a ← a – MEM - C 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
subc M, a	RAM 减累加器，再减进位，然后把结果放入 RAM。 例如： *subc* MEM, a ; 结果： MEM ← MEM – a - C 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
subc a	累加器减进位，然后把结果放入累加器。 例如： *subc* a; 结果： a ← a - C 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
subc M	RAM 减进位，然后把结果放入 RAM。 例如： *subc* MEM; 结果： MEM ← MEM - C 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
inc M	RAM 加 1。 例如： *inc* MEM ; 结果： MEM ← MEM + 1

	受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
dec M	RAM 减 1。 例如： dec MEM; 结果： MEM ← MEM - 1 受影响的标志位： Z：『受影响』， C：『受影响』， AC：『受影响』， OV：『受影响』
clear M	清除 RAM 为 0。 例如： *clear* MEM ; 结果： MEM ← 0 受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

mul	乘法运算，将执行 8x8 无符号乘法运算。
	例如： *mul* ;
	结果： {MulRH,ACC} ← ACC * MulOp
	受影响的标志位：N』Z 『N』C 『N』AC 『N』OV
	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』

7.4 逻辑运算类指令

and a, I	累加器和立即数据执行逻辑 AND，然后把结果保存到累加器。 例如：*and* a, 0x0f ; 结果：a ← a & 0fh 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
and a, M	累加器和 RAM 执行逻辑 AND，然后把结果保存到累加器。 例如：*and* a, RAM10 ; 结果：a ← a & RAM10 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
and M, a	累加器和 RAM 执行逻辑 AND，然后把结果保存到 RAM。 例如： *and* MEM, a ; 结果： MEM ← a & MEM 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
or a, I	累加器和立即数据执行逻辑 OR，然后把结果保存到累加器。 例如： *or* a, 0x0f ; 结果： a ← a | 0fh 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
or a, M	累加器和 RAM 执行逻辑 OR，然后把结果保存到累加器。 例如：*or* a, MEM ; 结果：a ← a | MEM 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
or M, a	累加器和立即数据执行逻辑 AND，然后把结果保存到累加器。

	例如：*and* a, 0x0f ; 结果：a ← a & 0fh 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
xor a, I	累加器和立即数据执行逻辑 XOR，然后把结果保存到累加器。 例如： *xor* a, 0x0f ; 结果： a ← a ^ 0fh 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』

xor IO, a	累加器和 IO 寄存器执行逻辑 XOR，然把结果存到 IO 寄存器。
	例如： *xor pa, a ;*
	结果： pa ← a ^ pa ; // pa 是 port A 资料寄存器
	受影响的标志位： Z：『不变』， C：『不变』， AC：『不变』， OV：『不变』

xor a, M	累加器和 RAM 执行逻辑 XOR，然后把结果保存到累加器。 例如： *xor* a, MEM ; 结果： a ← a ^ RAM10 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』
xor M, a	累加器和 RAM 执行逻辑 XOR，然后把结果保存到 RAM。 例如： *xor* MEM, a ; 结果： MEM ← a ^ MEM 受影响的标志位： Z：『受影响』， C：『不变』， AC：『不变』， OV：『不变』

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』

swapc IO.n	IO 口的位 N 与 C 位互换。
	例如： swapc IO.0;
	结果： C ← IO.0 , IO.0 ← C
	当 IO.0 是输出端口，进位 C 数值给 IO.0;
	当 IO.0 是输入端口，IO.0 数值给进位 C;
	受影响的标志位： 『不变』Z 『受影响』C 『不变』AC 『不变』OV
	应用范例 1（连续输出）：
	--- 	
	...
	set1 pac.0 ; // 设置 PA.0 作为输出
	...
	set0 flag.1 ; // C=0
	swapc pa.0 ; // 送 C 给 PA.0 （位操作），PA.0=0
	set1 flag.1 ; // C=1
	swapc pa.0 ; // 送 C 给 PA.0 （位操作），PA.0=1

	...
	--- 	
	应用范例 2 （连续输入） ：
	--- 	
	...
	set0 pac.0 ; // 设置 PA.0 作为输入
	...
	swapc pa.0 ; // 读 PA.0 的值给 C（位操作）
	src a ; // 把 C 移位给 ACC 的位 7
	swapc pa.0 ; // 读 PA.0 的值给 C（位操作）
	src a ; // 把新进 C 移位给 ACC 的位 7，上一个 PA.0 的值给 ACC 的位 6
	...
	--- 	

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

7.6 条件运算类指令

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

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

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
	…
	--- 	

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	条件成立	*ceqsn,* cneqsn,*t0sn, t1sn, dzsn, izsn*
1 个周期	条件不成立	
1 个周期		其他

7.9 指令影响标志综述

指令	Z	C	AC	OV	指令	Z	C	AC	OV	指令	Z	C	AC	OV
mov a, I	-	-	-	-	*mov* M, a	-	-	-	-	*mov* a, M	Y	-	-	-
mov a, IO	Y	-	-	-	*mov* IO, a	-	-	-	-	*ldt16* word	-	-	-	-
stt16 word	-	-	-	-	*idxm* a, index	-	-	-	-	*idxm* index, a	-	-	-	-
xch M	-	-	-	-	*pushaf*	-	-	-	-	*popaf*	Y	Y	Y	Y
add a, I	Y	Y	Y	Y	*add* a, M	Y	Y	Y	Y	*add* M, a	Y	Y	Y	Y
addc a, M	Y	Y	Y	Y	*addc* M, a	Y	Y	Y	Y	*addc* a	Y	Y	Y	Y
addc M	Y	Y	Y	Y	*nadd* *a, M*	Y	Y	Y	Y	*nadd* *M, a*	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	-	-	-	-	*mul*	-	-	-	-	*sr* a	-	Y	-	-
src a	-	Y	-	-	*sr* M	-	Y	-	-	*src* M	-	Y	-	-
sl a	-	Y	-	-	*slc* a	-	Y	-	-	*sl* M	-	Y	-	-
slc M	-	Y	-	-	*swap* a	-	-	-	-	*and* a, I	Y	-	-	-
and a, M	Y	-	-	-	*and* M, a	Y	-	-	-	*or* a, I	Y	-	-	-
or a, M	Y	-	-	-	*or* M, a	Y	-	-	-	*xor* a, I	Y	-	-	-
xor IO, a	-	-	-	-	*xor* a, M	Y	-	-	-	*xor* M, a	Y	-	-	-
not a	Y	-	-	-	*not* M	Y	-	-	-	*neg* a	Y	-	-	-
neg M	Y	-	-	-	*comp* *a, M*	Y	Y	Y	Y	*comp* *M, a*	Y	Y	Y	Y
set0 IO.n	-	-	-	-	*set1* 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	*cneqsn* *a,M*	Y	Y	Y	Y	*cneqsn* *a, I*	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*	-	-	-	-
ldtabl index	-	-	-	-	*ldtabh* index	-	-	-	-	*xor* a, IO	Y	-	-	-
swap M	-	-	-	-	*nmov* M, a	-	-	-	-	*nmov* a, M	Y	-	-	-

7.10 位定义

位寻址只能定义在 RAM 区地址的 0x00 to 0x3F。

8. 程序选项

选项	选择	描述
Security	**Enable**	**内容加密，程序不允许被读取**
	Disable	内容不加密，程序可以被读取
LVR	4.0V	选择 LVR = 4.0V
	3.5V	选择 LVR = 3.5V
	3.0V	选择 LVR = 3.0V
	2.7V	选择 LVR = 2.7V
	2.5V	选择 LVR = 2.5V
	2.2V	选择 LVR = 2.2V
	2.0V	选择 LVR = 2.0V
	1.8V	选择 LVR = 1.8V
LCD2 (请参阅 MISC.4)	**Disable**	**禁用 VDD/2 LCD 偏置电压发生器。所有 IO 引脚均正常**
	PB0_A035	启用 VDD/2 LCD 偏置电压发生器，输入模式下 PB0 和 PA[0,3,5] 为 VDD/2。
	PB7_C0~6	启用 VDD/2 LCD 偏置电压发生器，输入模式时 PB7 和 PC[0~6] 为 VDD/2。
	PB1256	启用 VDD/2 LCD 偏置电压发生器，输入模式时 PB[1,2,5,6] 为 VDD/2
Interrupt Src0	**PA.0**	**INTEN/ INTRQ.Bit0 用于 PA.0**
	PB.5	INTEN/ INTRQ.Bit0 用于 PB.5
	PA.7	INTEN/ INTRQ.Bit0 用于 PA.7
Interrupt Src1	**PB.0**	**INTEN/ INTRQ.Bit1 用于 PB.0**
	PA.3	INTEN/ INTRQ.Bit1 用于 PA.3
	PB.6	INTEN/ INTRQ.Bit1 用于 PB.6

选项	选择	描述
IO_Drive	Normal	第 1 组：PA0/PA3/PB5/PB6/PB7/PC2 第 2 组：PA5/PA7/PC0/PC1/PC3/PC4 第 3 组：PB0/PB1/PB2/PB3/PB4/PC5/PC6
	Strong	第 1 组：PA0/PA3/PB5/PB6/PB7/PC2 第 2 组：PA5/PA7/PC0/PC1/PC3/PC4 第 3 组：PB0/PB1/PB2/PB3/PB4/PC5/PC6
PWM_Source	**16MHZ**	当 pwmgclk.0= 1, LPWMG 时钟源 = IHRC = 16MHZ
	32MHZ	当 pwmgclk.0= 1, LPWMG 时钟源 = IHRC*2 = 32MHZ
TMx_Source	**16MHZ**	**当 tm2c[7:4]= 0010, TM2 时钟源 = IHRC = 16MHZ** **当 tm3c[7:4]= 0010, TM3 时钟源 = IHRC = 16MHZ**
	32MHZ	当 tm2c[7:4]= 0010, TM2 时钟源 = IHRC*2 = 32MHZ 当 tm3c[7:4]= 0010, TM3 时钟源 = IHRC*2 = 32MHZ (ICE does NOT Support.)
TMx_Bit	**6 Bit**	**当 tm2s.7=1, TM2 PWM 分辨率为 6 位** **当 tm3s.7=1, TM3 PWM 分辨率为 6 位**
	7 Bit	当 tm2s.7=1, TM2 PWM 分辨率为 7 位 当 tm3s.7=1, TM3 PWM 分辨率为 7 位 (ICE 不支持）。
RMS_Channel	**PA.0**	**RMS 通道为 PA.0**
	PA.3	RMS 通道为 PA.3

注：**粗体**选项为默认选项。

9. 特别注意事项

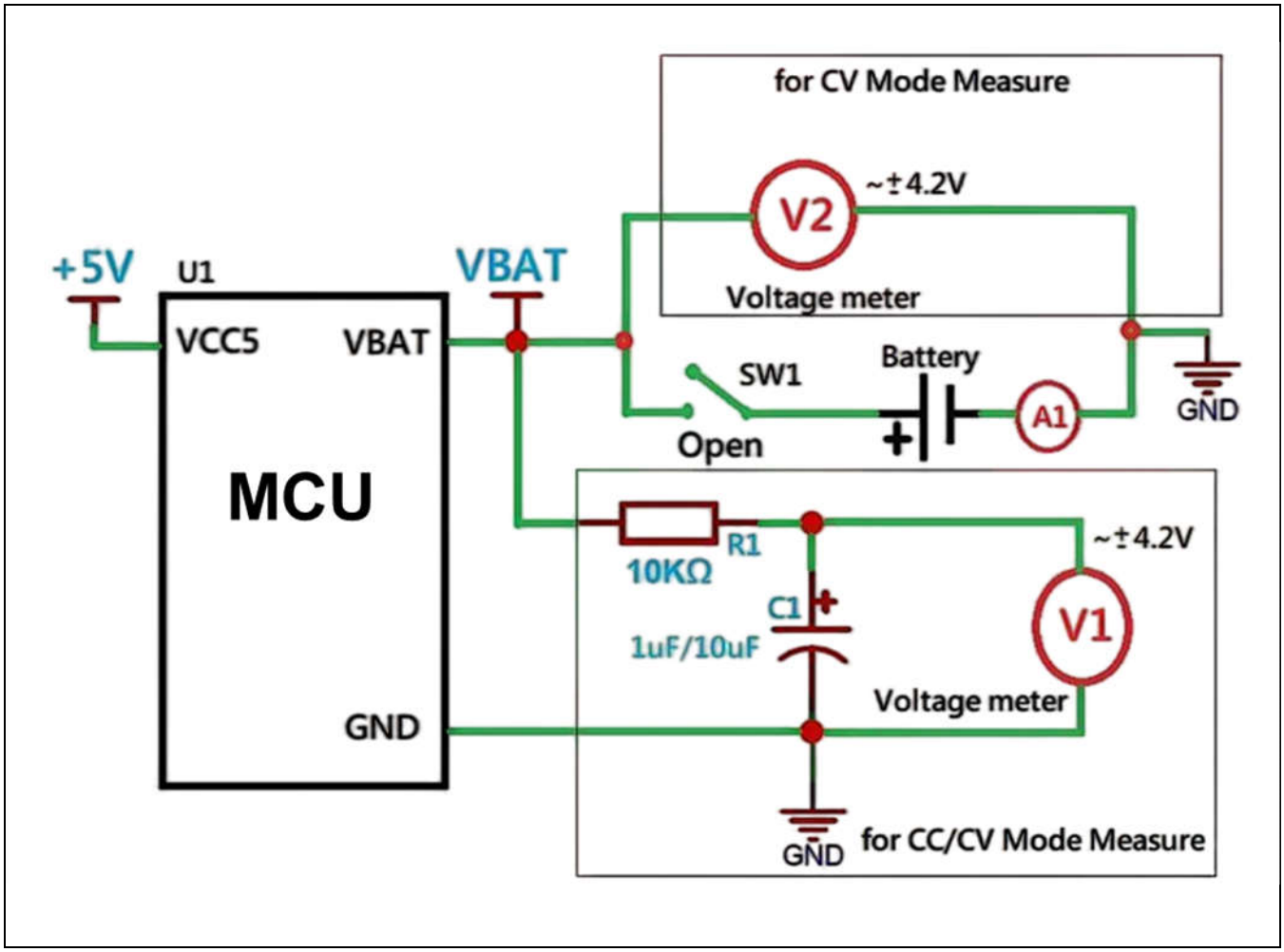
此章节提醒用户在使用 MFU901 系列 IC 时避免常犯的一些错误。

9.1. 使用IC

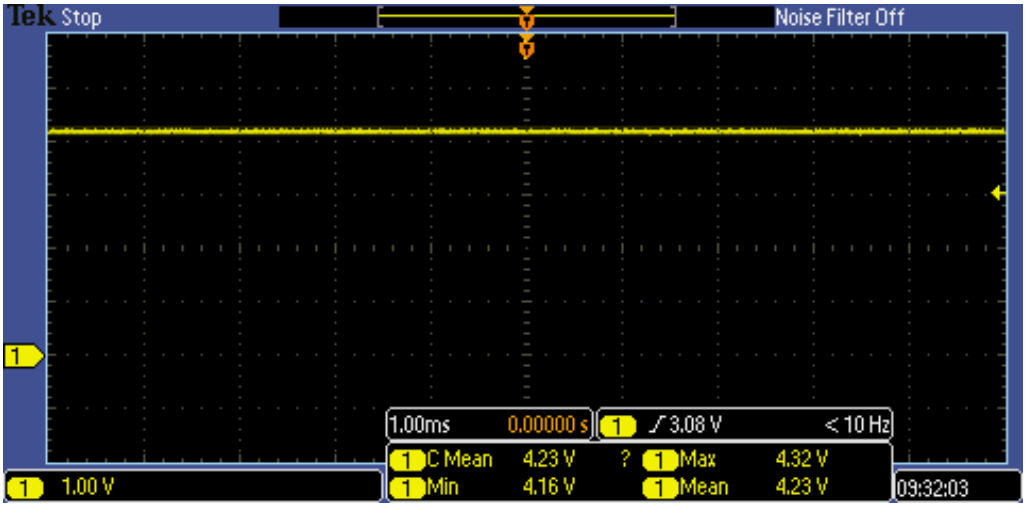
9.1.1.充电器使用与设定

(1) 充电电压及电流量测

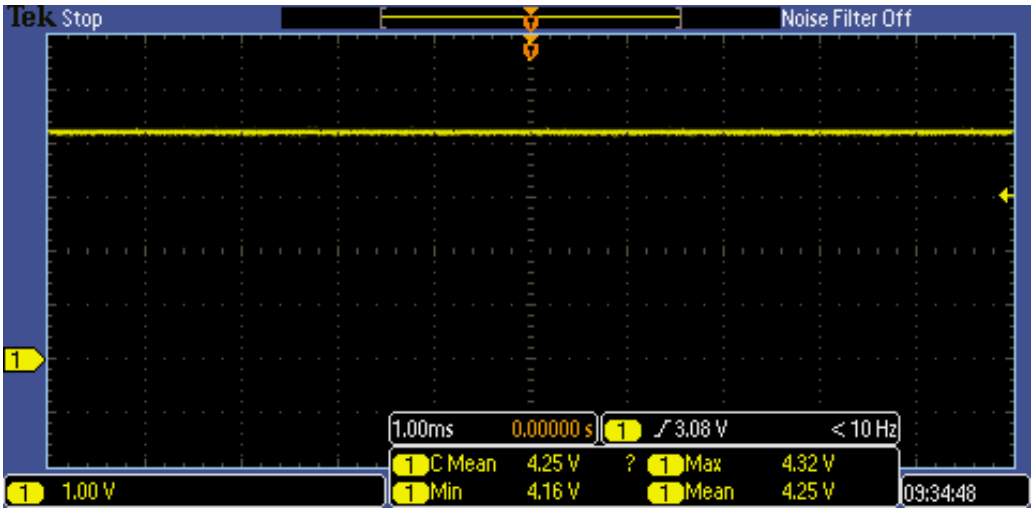
- MFU901 充电器与 MCU 执行程序是个别独立工作, MCU 复位并不影响到充电动作。
- 对空片的 MFU901 充电器做充电电压及电流做量测时所得数据均属充电器未载入校正参数时的电压 / 电流值。
- MFU901 需在程序正常启动工作后将会为充电器写入校正参数，此时可对 MFU901 的充电器做电性量测。
- 量测 MFU901 充电器的充满电压接线图如下示意：在 CC Mode 下需在 VBAT 引脚串联 RC 电路(仿真电池内阻等效电路)，电压表 V1 并接于电容器 C1 上，在 CV Mode 下可将电压表 V2 并联接于 VBAT 引脚。



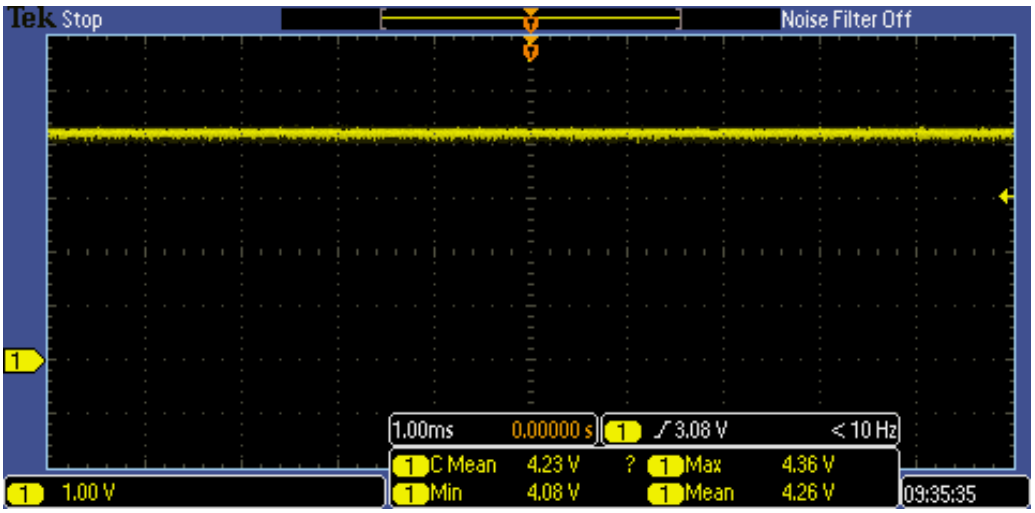
V1 电压波形：(CC Mode, R1 = 10Kohm, C1 = 1uF)



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

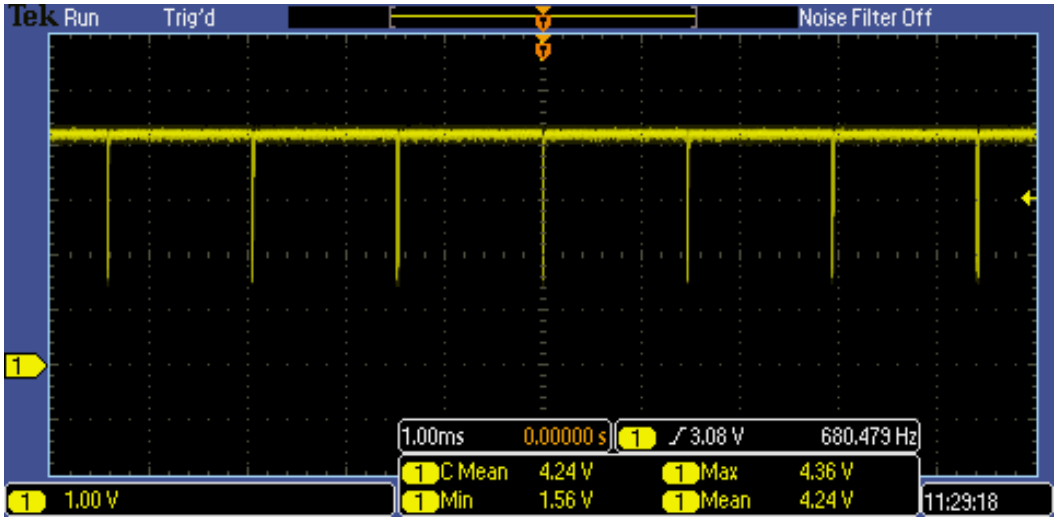


V2 电压波形：(CV Mode, No R1 and C1)

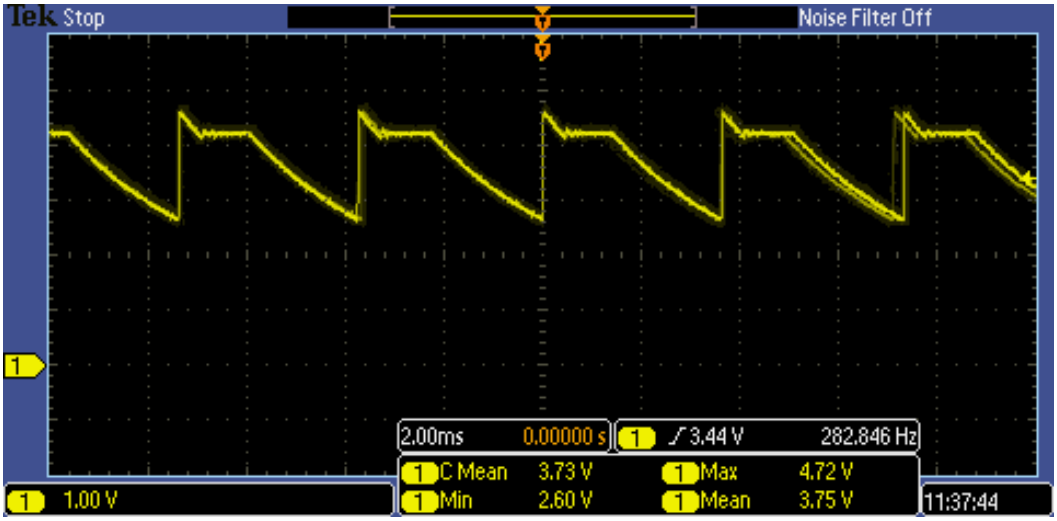


在 CC Mode 下 VBAT 引脚空接或是只接上滤波电容/电解电容时所量到的电压可能会因为充电器对外部电容的充放电效应，使得平均电压偏低。充电重新循环时间 Cycle 约在 1~3ms 在充电至 4.2V 满电时，若 VBAT 引脚未接电池或是只接电容，将会因为放电时间导致电压表量到的平均值变低。例如使用电压表直接量测 VBAT 电压时，在 VBAT 引脚只加 1uF 电容反而会造成量测错误(远低于目标值)，因为放电时间会变长，导致电压表量到的平均值变低。如下图波形：

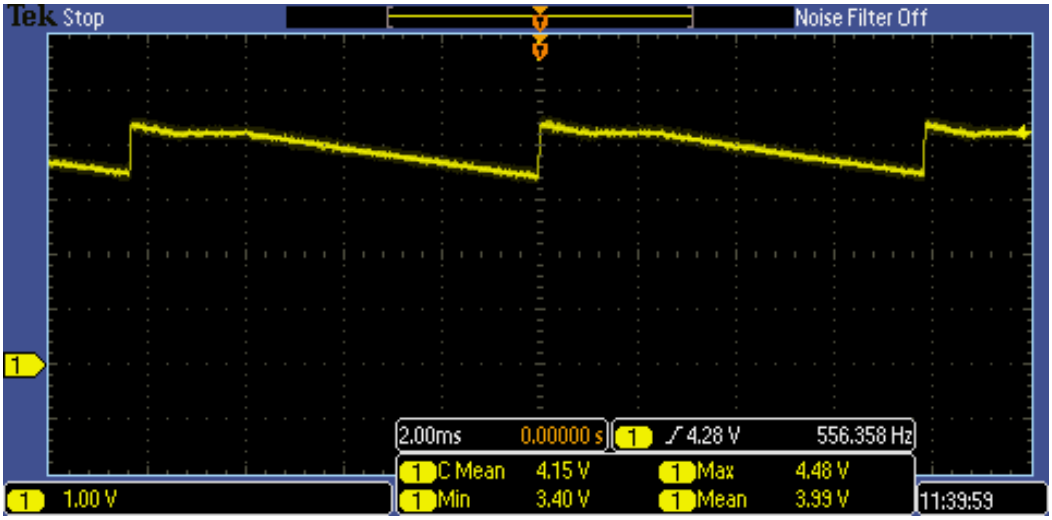
V2 电压波形：(CC Mode, No R1 and C1)



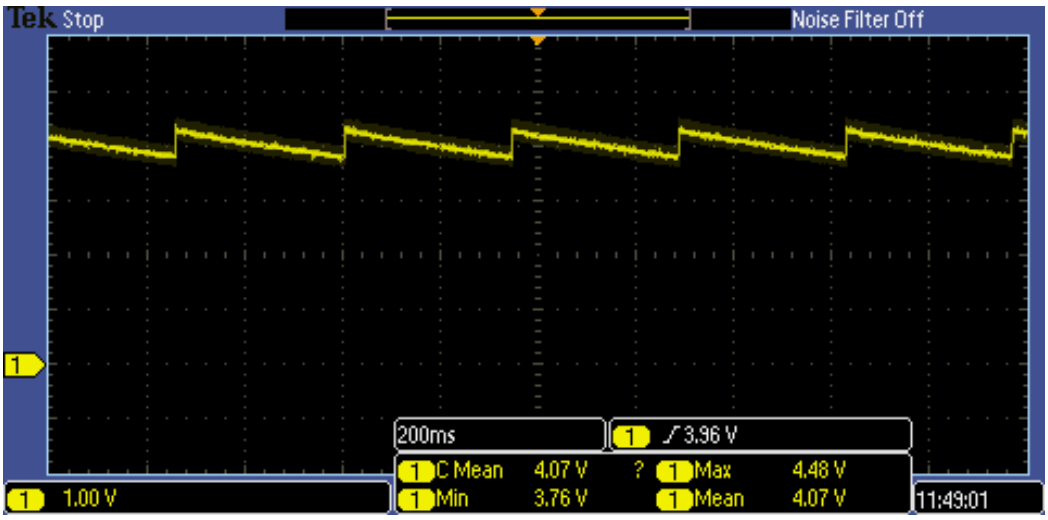
V2 电压波形：(CC Mode, R1 = 0R , C1 = 1uF)



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



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



9.1.2.引脚的使用和设定

(1) IO 作为数字输入时

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

(2) IO 作为数字输入和打开唤醒功能

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

(3) 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

(4) PA5 作为输入并通过长导线连接至按键或者开关

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

9.1.3. 中断

(1) 使用中断功能的一般步骤如下：

步骤 1：设定 INTEN 寄存器，开启需要的中断的控制位

步骤 2：清除 INTRQ 寄存器

步骤 3：主程序中，使用 ENGINT 指令允许 CPU 的中断功能

步骤 4：等待中断。中断发生后，跳入中断子程序

步骤 5：当中断子程序执行完毕，返回主程序

*在主程序中，可使用 DISGINT 指令关闭所有中断

* 跳入中断子程序处理时，可使用 PUSHAF 指令来保存 ALU 和 FLAG 寄存器资料，并在 RETI 之前，使用 POPAF 指令复原，步骤如下：

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

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

9.1.4. 系统时钟选择

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

◆ 例一：系统时钟从 ILRC 切换到 IHRC/2

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

◆ **错误的写法**：ILRC 切换到 IHRC，同时关闭 ILRC

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

9.1.5. 看门狗

看门狗默认为开，但程序执行 ADJUST_IC 时，会将看门狗关闭，若要使用看门狗，需重新配置打开。当 ILRC 关闭时，看门狗也会失效。

9.1.6. TIMER 溢出

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

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

9.1.7. IIHRC

(1) IHRC 频率会在使用烧录器烧录程序时校准。

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

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

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

9.1.8. LVR

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

下面是工作频率、电源电压和 LVR 水平设定的建议：

系统时钟	V_{DD}	LVR
2MHz	≧ 1.8V	≧ 2.0V / 1.8V
4MHz	≧ 2.2V	≧ 2.5V / 2.2V
8MHz	≧ 3.5V	≧ 4.0V/ 3.5V

表 8：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 为 1 将 LVR 关闭，但此时应确保 V_{DD} 在最低工作电压以上，否则 IC 可能工作不正常。

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

9.1.9. 烧录方法

MFU901 的烧录脚为: PA5, PA7, V_{DD} 和 GND 这 4 个引脚.

请使用 5S-P-003Bx / 5S-P-003C 或 5S-P-C01 以后的版本烧录 MFU901 实际芯片

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

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

(2) 其他烧录引脚（GND 除外）的电位与 V_{BAT} 相同。

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

重要注意事项：

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

9.1.9.1. 使用 5S-P-003Bx/ 5S-P-003C 烧录 MFU901

使用 5S-P-003Bx / 5S-P-003C 烧录 MFU901，使用 Jumper7 转接程序信号，信号的连接取决于 IC 封装。 请参阅 Writer 用户手册的第 5 章，为目标 IC 封装制作 Jumper7 转接板。用户可以从以下网页链接获取用户手册。

http://www.padauk.com.tw/en/technical/index.aspx?kind=37

单芯片普通烧录：

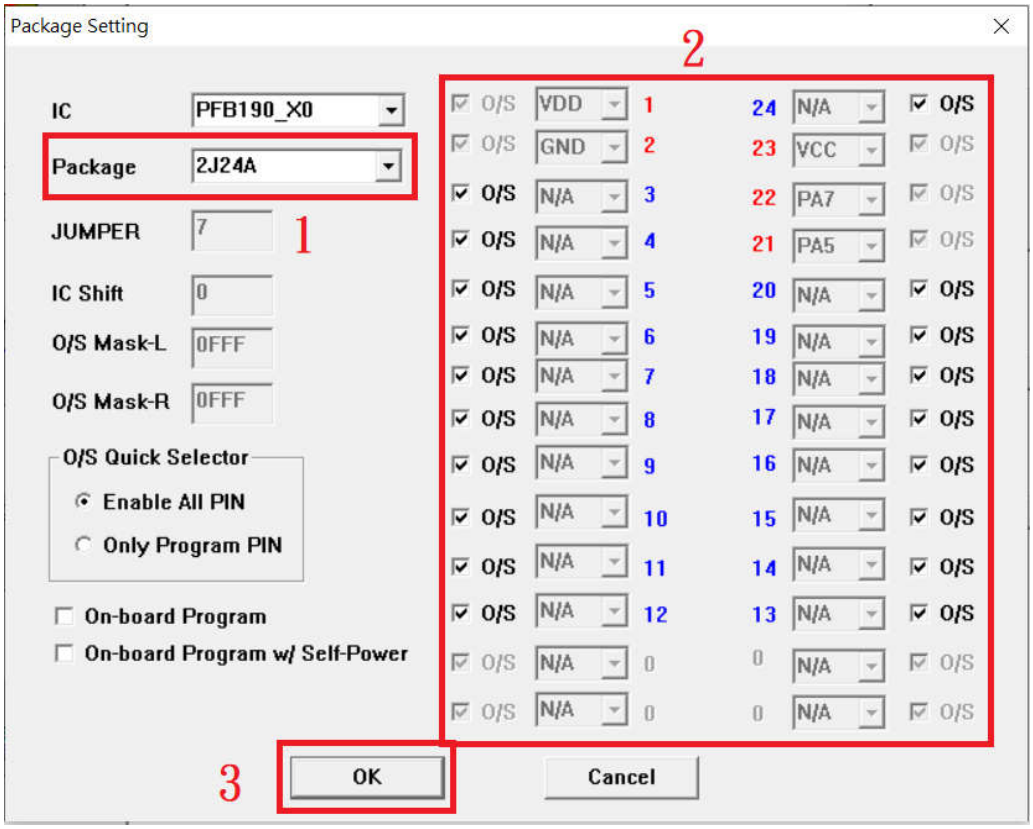
下载 PDK 文件后，单击 <Convert> ➔ <To Package> ➔"选择要下载的 PDK" ➔"选择封装"➔ 保存文件后 ➔ 点击 Auto Program

5 线在板烧录:

(1) P-003B2/P-003C 烧录器不支持在板自供电 4 线烧录

(2) 系统板没有供电（没有锂电池供电）

下载 PDK 文件后，单击 <Convert> ➔ <To Package> ➔ "选择要下载的 PDK" ➔ "选择封装"➔ **"选择 On-Board Program"** ➔ 保存文件后 ➔ 点击 Auto Program



从图形用户界面加载 PDK，插入 JP7，然后在插座上输入 IC，无需移位。LCDM 显示 IC 就绪后，即可写入。

注意：烧录器上不支持 VCC 脚位的 O/S Test

另 PDK 转文件信息亦可直接定义于程序中，如下：

```
//              1          2    3   4   5  6   7   8   9   10    11     12     13     14        15
// S20      pin_cnt       vbat pa0 pa3 pa4 pa5 pa6 pa7 gnd vcc5  agnd   mask1  mask2  shift  option
.writer package 20,        2,   0,  0,  12,  0,  9,  0,  20,  1,  0,     0x102, 0x101,  0,     0x10

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

1. 如图 21 所示，这是 Jumper7 连接方法。(以 QFN24 为例）

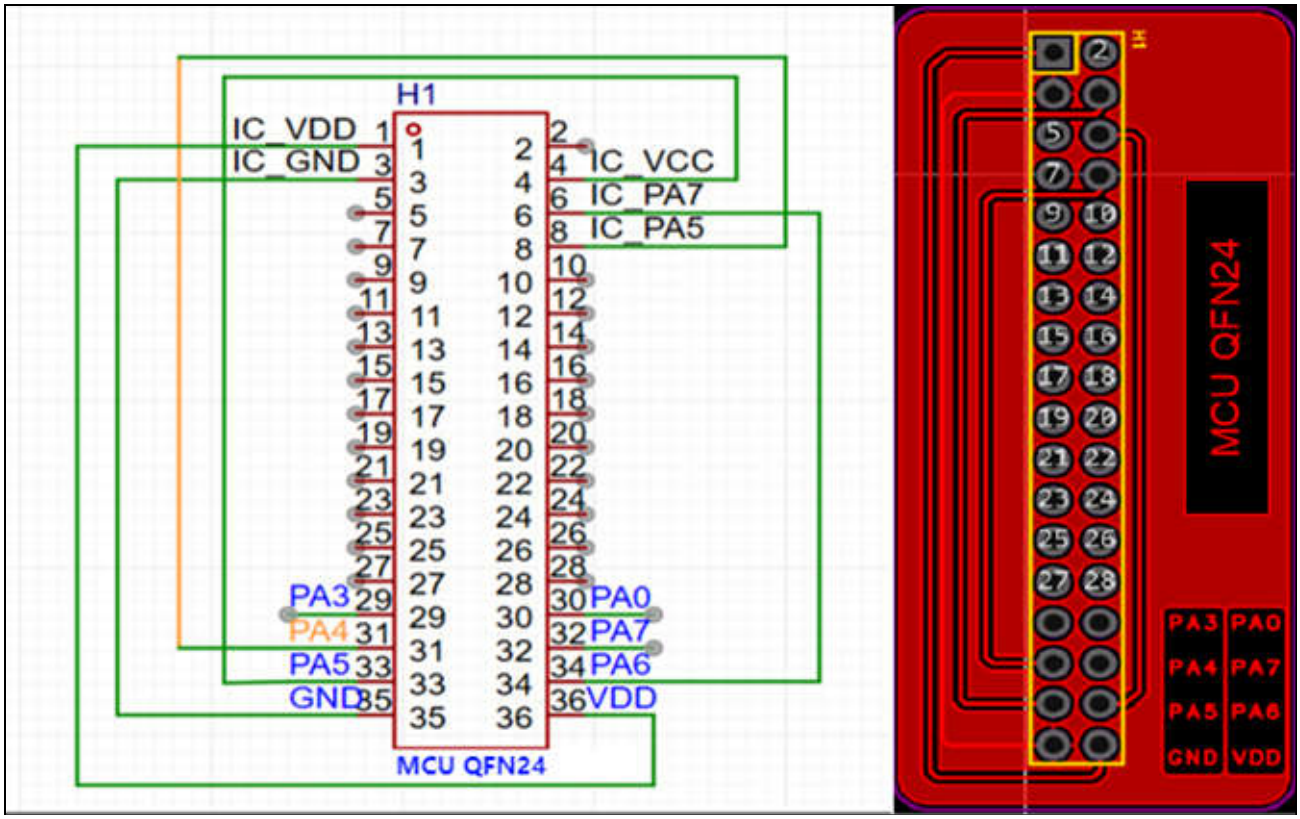
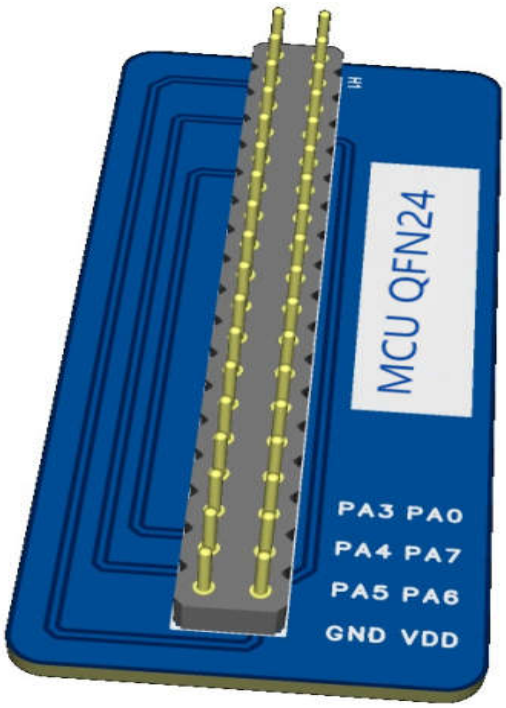


图.21: P-003B2 / P-003C Jumper7 的示意图



JP7 跳线:

WRT_VDD ←→ IC_VDD

WRT_GND ←→ IC_GND

WRT_PA5 ←→ IC_VCC

WRT_PA6 ←→ IC_PA7

WRT_PA4 ←→ IC_PA5

2. 将 JP7 插入插座并输入 IC，无需移位。LCDM 显示 IC 就绪后，即可写入。

9.1.9.2. 使用 5S-P-C01 烧录 MFU901

(1) 仅支持板载自供电 4 线烧录，PCBA 必须由锂电池供电。

(2) 5S-P-C01 烧录器仅支持选中 OBP 设置的 PDK 文件下载。

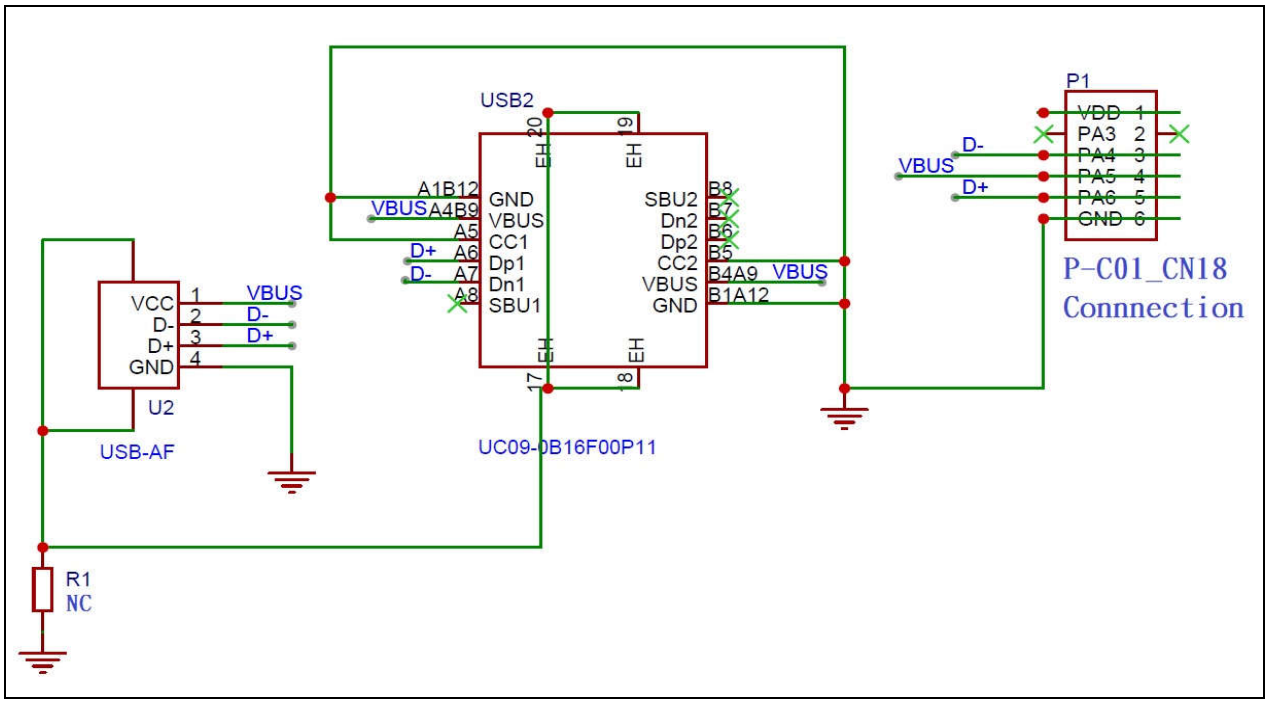
(3) 5S-P-C01 烧录器支持 MFU901 独立离线在板烧录。

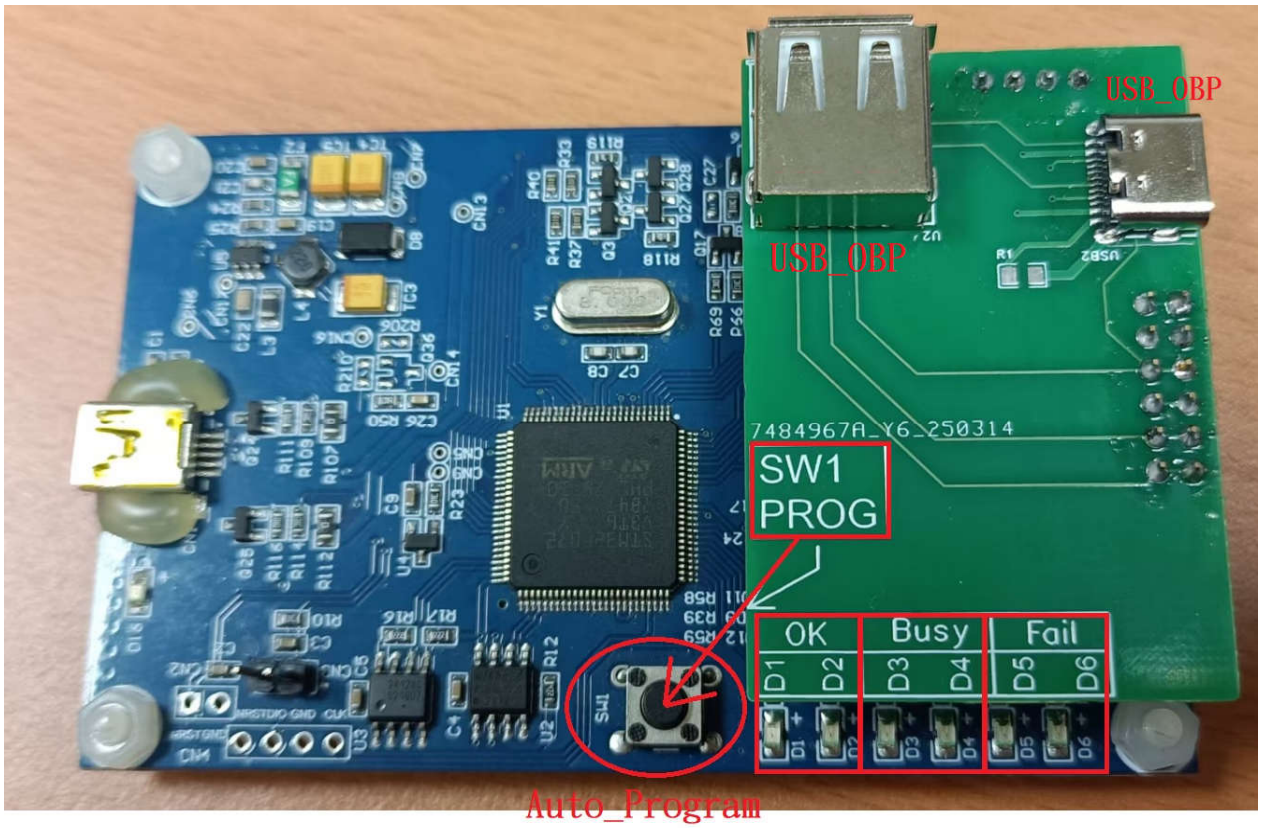
(4) 当 5S-P-C01 用于独立离线烧录时，SW1 是自动烧录按钮

	D1 / D2	D3 / D4	D5 / D6
IC_Remove	**On**	**On**	**On**
IC_Ready	**Off**	**Off**	**Off**
Program OK	**On**	**Off**	**Off**
Program Fail	**Off**	**Off**	**On**
Programming Busy	**Off**	**On ⇔ Off Flash**	**Off**

表 9: 5S-P-C01 LED 状态表

下载 PDK 文件后，单击 <Convert> ➔ <To Package> ➔ "选择要下载的 PDK" ➔"选择封装"➔ **"选择 'On-Board Program w/ Self-Power"** ➔ "保存文件后 ➔ 点击 Auto Program "


USB2
A1B12
VBUS A4B9
A5
D+ A6
D- A7
A8
GND
VBUS
CC1
Dp1
Dn1
SBU1
SBU2
Dn2
Dp2
CC2
VBUS
GND
B8
B7
B6
B5
B4A9 VBUS
B1A12
EH 20
EH 19
EH 17
EH 18
UC09-0B16F00P11
VCC 1 VBUS
D- 2 D-
D+ 3 D+
GND 4
U2
USB-AF
R1
NC
P1
VDD 1
PA3 2
PA4 3
PA5 4
PA6 5
GND 6
D-
VBUS
D+
P-C01_CN18
Connnection


USB_OBP
USB_OBP
7484967A_Y6_250314
SW1
PROG
OK
Busy
Fail
D1
D2
D3
D4
D5
D6
Auto_Program

烧录线 ：

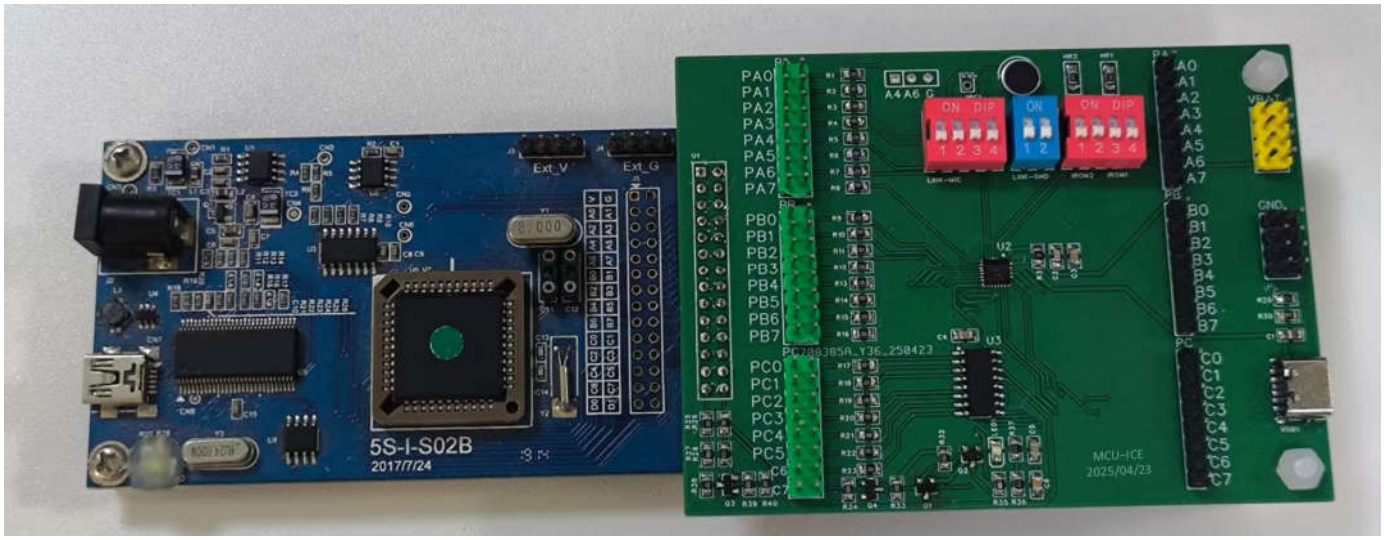
P-C01		MFU901		电池
		IC_VDD	←→	锂电池 (+)
WRT_GND	←→	IC_GND	←→	锂电池 (-)
WRT_PA5	←→	IC_VCC		
WRT_PA6	←→	IC_PA7		
WRT_PA4	←→	IC_PA5		

9.1.10. 应用手册

关于充电器更多的使用注意事项，请参阅 APN-021 文件说明。

9.2. 使用 ICE

5S-I-CEB001 为应广科技所推出的仿真工具，需要搭配应广科技的 IDE 软件做联机仿真，5S-I-CEB001 使用时需搭配 5S-I-S01/2(B)使用，参考如下图片：



5S-I-CEB001 仿真注意事项：

1. 5S-I-CEB001 必须搭配 5S-I-S01/2(B)仿真器才能仿真 PXPL/ LPWM/ LVDC/ TIMER2/ GPC/ CHARGE 功能。
2. 5S-I-CEB001 电源由 5S-I-S01/2(B) 提供，为了稳定电源请将 5S-I-S01/S02(B) 接上 DC9V 的电源调适器。
3. IDE 从 1.04D3 版本开始支持 5S-I-CEB001 的仿真。
4. 仿真 PXPL/ LPWM/ LVDC/ TIMER2/ GPC/ CHARGE 相关功能时，5S-I-S01/2(B)会和仿真板 5S-I-CEB001 进行通信，所以仿真时序慢与实际 IC，主要影响的寄存器有 intrq/ PxPL/ LVDC/ CHGC/ CHGS/ LPWMGCLK/ LPWMGCUBH/ LPWMGCUBL/ LPWMGxC/ LPWMGxDTH/ LPWMGxDTL/ GPCC/ GPCS/ TM2CT/ TM2B/ TM2S/ TM2C。

5. 仿真时，5S-I-S01/2(B)和 5S-I-CEB001 通信会使中断延后触发，延后时间约 0~335us。
6. 使用 T16 中断定时，中断定时有误差，5S-I-S01/2(B) 和 5S-I-CEB001 通信时会切换系统时钟，且会使中断延后触发，此现象常出现在定时器时钟源选择 SYSCLK 时或定时时间较短时。故建议仿真时单个定时中断周期 ≥ 10ms。
7. 仿真不支持比较器唤醒。
8. 仿真不支持 GPC 控制 PWM 信号。
9. 仿真 GPC/ Timer2/ LPWM 时，中断不支持硬件触发，只能以轮询的方式获取中断触发信号。
10. 由于 5S-I-CEB001 上 IC 程序是 8M，仿真 Timer2 或 LPWM 时，选择系统时钟为 SYSCLK，模块是以 8M 运行的，即系统时钟被强制到了 8M，实际 IC 是正常的系统时钟。
11. 仿真 Timer2 或 LPWM 时不支持 32M 时钟，实际 IC 是可以的。
12. Timer2 仿真不支持 7BIT 分辨率。
13. 仿真 MFU901 时，对 PC IO 口的操作要注意，PC7 在接电池要保证为输出高，不接电池时要保证输出低。
14. 当不接电池时，5S-I-S01/2(B) 比 5S-I-CEB001 的电压要高 0.3V-0.4V (5S-I-CEB001 上的二极管导致的) 。

WDT 周期	5S-I-S01/2(B)	MFU901
misc[1:0]=00	2048 * T_{ILRC}	8192 * T_{ILRC}
misc[1:0]=01	4096 * T_{ILRC}	16384 * T_{ILRC}
misc[1:0]=10	16384 * T_{ILRC}	65536 * T_{ILRC}
misc[1:0]=11	256 * T_{ILRC}	262144 * T_{ILRC}

9.3. 典型应用

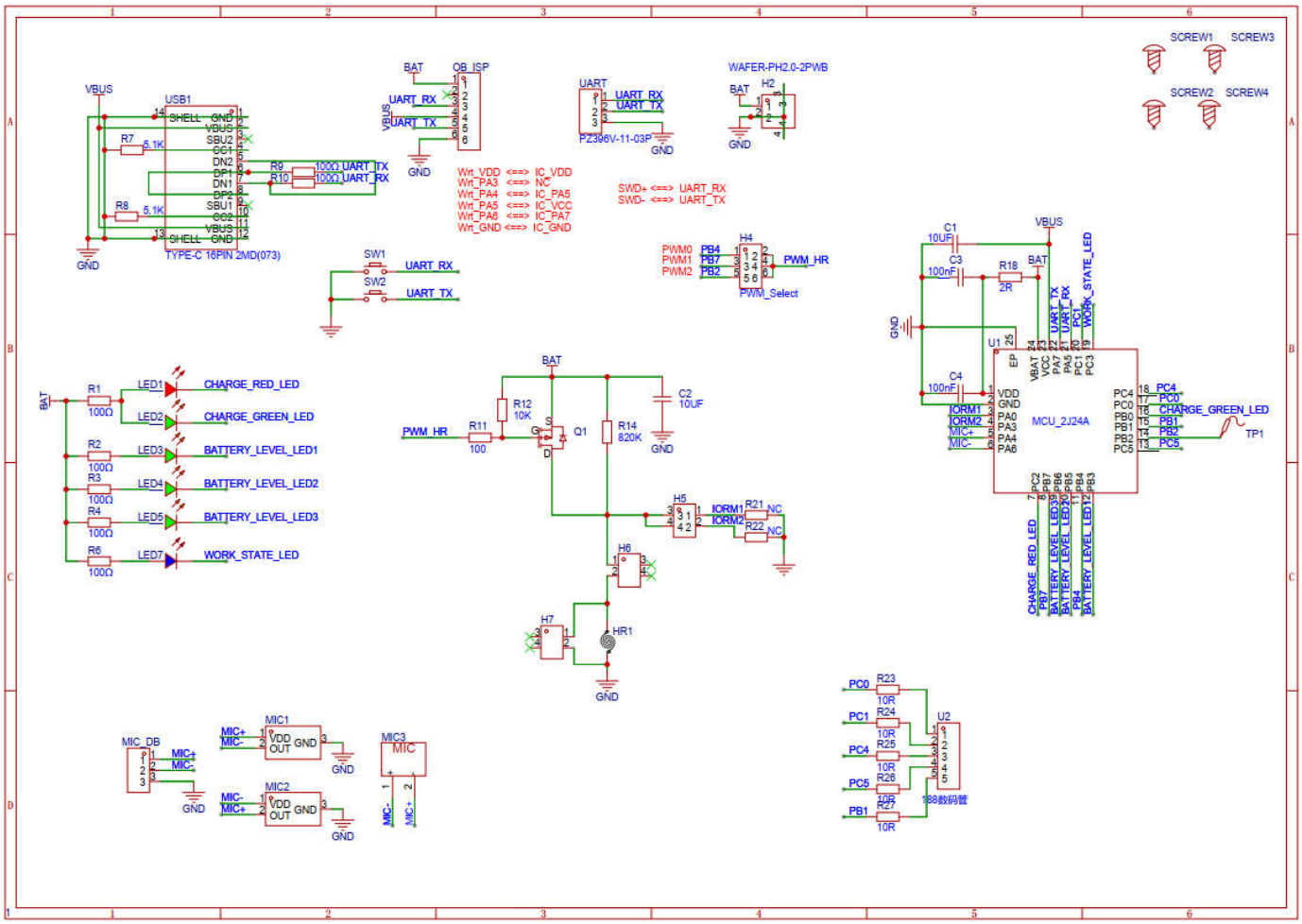


图.22: DemoBoard 原理图

MIC（+）脚连接至 **PA4**，MIC（–）脚连接至 **PA6**

9.4. 针对锂电池电源上电及抖动干扰的程序对策

MFU901 应用板在生厂过程中常会有锂电池电极片与应用板需要透过点焊加工做连接。在此状况会造成应用板上的电源跳动不稳定，使得 MFU901 上电波形出现抖动干扰。此一过程有可能会持续数十至数百毫秒，这对于芯片上电开机是一严苛的挑战，有可能造成芯片出现死机、程序跑飞、功能异常、系统参数错乱…等状况。针对此一状况可在程序依照下列几点设置可提升 MFU901 IC 在电源上电及电源抖动时的稳定性。

(1) CodeOption 设置：

- CodeOption 开机速度选择 Slow。(若芯片有支持此功能)

(2) 开机程序在.Adjust_IC 前优先做 IO 输出低及延时设置：

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

(3) 系统频率：

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

```
//------------------------------------------------------------
// Macro Name: PowerOn_Delay
// Parameter: none
//
//------------------------------------------------------------
byte pndt;
PowerOn_Delay  macro          // delay 2048 ILRC period = 35.31ms
    PA = 0x00;
    PAC = 0xFF;               // change to Output Mode
    pndt = 0xFF;
    do
     {      nop; nop; nop; nop; nop;   }
    while(pndt--);
    PAC = 0x00;               // change to Output Mode
  endm
 void    FPPA0 (void)
{
```

```
#if     _SYS(AT_EV)
   $ MISC WDT_64K; Fast_Wake_Up;            // 64K*ILRC = 40ms
#endif
   PowerOn_Delay   //delay 35ms
   .ADJUST_IC      SYSCLK=ILRC (IHRC/16), IHRC=16MHz, VDD=4.2V, O_WDRST, X_FIRST;
   .wdreset;
   .delay 4000                          // delay about 69ms for VDD = 5V
//---- ReLoad_All_Param
   ReLoad_IHRC                          //Reload IHRCR Parameter
   ReLoad_ChargerCURTRIM     //Reload Charger Current Trim Bits
   ReLoad_VbatBGTRIM         //Reload Charger Vbat Trim Bits
  $ MISC WDT_64K;                           // 64K*ILRC = 1103ms
   .wdreset;
   $ CLKMD IHRC/16, En_IHRC, En_ILRC, En_WatchDog;

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

(4) 看门狗：

- 保持看门狗为 Enable 的状态，不关闭看门狗。.Adjust_IC 的参数加 **O_WDTRST。**
- 进入休眠 Stopexe Mode 时不必关闭看门狗，芯片硬件会自动关闭看门狗，并且在唤醒后自动重新启用看门狗，看门狗计数器也将一并会被清除。
- 看门狗清零指令执行周期建议不要加的太密集。建议在主程序中执行一次即可
- 若有使用中断，则不建议在中断程序中加清看门狗指令。
- 切换系统频率时需注意要保持看门狗开启状态，避免误关看门狗。
- 建议看门狗时间设成 64K 个 ILRC，约 1103ms。考虑 ILRC 的频漂误差，清看门狗周期建议抓 550ms ~ 800ms 的时间误差。

(5) Stopexe/Stopsys 唤醒：

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