



Solarec

常见问题 FAQ

文档版本 04

发布日期 2022-12-05

版权所有 © 海思技术有限公司2024。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HISILICON、海思和其他海思商标均为海思技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

海思技术有限公司

地址：上海市青浦区虹桥港路2号101室 邮编：201721

网址：<https://www.hisilicon.com/cn/>

客户服务邮箱：support@hisilicon.com



前言

概述

本文档主要介绍SolarA²解决方案中常见的问题和解决办法的FAQ。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
SolarA ²	1.0.1




读者对象

本文档（本指南）主要适用于以下工程师：



- 技术支持工程师
- 软件开发工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
	表示如不可避免则将会导致死亡或严重伤害的具有高等级风险的危害。
	表示如不可避免则可能导致死亡或严重伤害的具有中等级风险的危害。
	表示如不可避免则可能导致轻微或中度伤害的具有低等级风险的危害。



符号	说明
 须知	用于传递设备或环境安全警示信息。如不可避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
 说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修改记录

修订日期	版本	修订说明
2022-08-15	01	第1次正式版本发布。
2022-09-02	02	第2次正式版本发布。 刷新3.3.2章节 解决办法说明。
2022-09-09	03	第3次正式版本发布。 新增3.4章节 串口首次烧录打印乱码问题。 新增1.2章节 GPIO初始化同组不同PIN注册中断问题注意事项。
2022-12-05	04	第4次正式版本发布。 新增3.1章节 8/16bit读写需要注意的寄存器列表。



目录

前言.....	i
1 模拟模块使用类.....	1
1.1 ADC.....	1
1.1.1 ADC 采样转换完成的标志寄存器，查询后需要进行置位恢复为 0.....	1
1.1.1.1 问题描述.....	1
1.1.1.2 解决办法.....	1
1.2 GPIO.....	1
1.2.1 GPIO 初始化同组不同 PIN 注册中断问题注意事项.....	2
1.2.1.1 问题描述.....	2
1.2.1.2 解决办法.....	2
2 sample 使用类.....	3
2.1 调试打印.....	3
2.1.1 调用打印接口，无法正常打印.....	3
2.1.1.1 问题描述.....	3
2.1.1.2 解决办法.....	3
3 其他注意事项.....	4
3.1 寄存器读写不符合预期、读写后出现未定义行为.....	4
3.1.1 IP 寄存器读写前必须 CRG 使能.....	4
3.1.1.1 问题描述.....	4
3.1.1.2 解决办法.....	4
3.1.2 错误的寄存器访问操作.....	4
3.1.2.1 问题描述.....	4
3.1.2.2 解决办法.....	5
3.2 对寄存器读取后进行移位操作，不符合预期.....	6
3.2.1 问题描述.....	7
3.2.2 解决办法.....	7
3.3 全局变量定义使用，赋值不生效.....	7
3.3.1 问题描述.....	7
3.3.2 解决办法.....	7
3.4 JTAG 管脚相关复选功能配置不生效.....	8
3.4.1 问题描述.....	8
3.4.2 解决办法.....	8



3.5 串口首次烧录打印乱码问题.....	8
3.5.1 问题描述.....	8
3.5.2 解决办法.....	8
3.6 硬件配置不当引起的 I2C 通信失败问题.....	8
3.6.1 问题描述.....	8
3.6.2 解决办法.....	9



表格目录

表 3-1 8/16bit 读写需要注意的寄存器列表.....	5
---------------------------------	---



1 模拟模块使用类

1.1 ADC

1.1.1 ADC 采样转换完成的标志寄存器，查询后需要进行置位恢复为0

1.1.1.1 问题描述

直接读取ADC某个SOC的采样完成标志寄存器，直接读寄存器后，读取后没有恢复为默认值。

1.1.1.2 解决办法

解决办法1：使用HAL接口

使用HAL接口HAL_ADC_CheckSocFinish()时，查询后会返回采样状态，内部已经实现了置位操作。

HAL_ADC_CheckSocFinish返回值：

- BASE_STATUS_ERROR：表示SOC没有采样完成。
- BASE_STATUS_OK：表示SOC采样完成。

解决办法2：使用DCL接口

使用DCL接口时，需要两个接口配合完成，先使用DCL_ADC_GetConvState()接口获取状态，再使用DCL_ADC_ResetConvState()进行置位。

DCL_ADC_GetConvState返回值：

- 返回值等于0：表示SOC没有采样完成。
- 返回值不等于0：表示SOC采样完成。

1.2 GPIO



1.2.1 GPIO 初始化同组不同 PIN 注册中断问题注意事项

1.2.1.1 问题描述

同一组GPIO的不同pin分别注册中断回调函数的时候，通过对每个pin命名一个handle进行操作时，注册中断回调的时候对每个pin重命名的handle进行注册，出现中断只响应同组中的一个pin问题，同组的其余pin的回调函数不响应。

例如：

```
g_led1.baseAddress = GPIO0;  
g_led1.pins = GPIO_PIN_0;  
g_led1.dir = GPIO_INPUT_MODE;  
g_led1.value = GPIO_LOW_LEVEL;  
g_led1.interruptMode = GPIO_INT_TYPE_RISE_EDGE;  
HAL_GPIO_Init(&g_led1);  
HAL_GPIO_RegisterCallback(&g_led1, GPIO_PIN_0, GPIO_CallbackFunc);  
HAL_GPIO_IRQService(&g_led1);  
IRQ_SetPriority(g_led1.irqNum, 1); /* set gpio1 interrupt priority to 1, 1~7 */  
IRQ_EnableN(g_led1.irqNum); /* gpio interrupt enable */  
  
g_led2.baseAddress = GPIO0;  
g_led2.pins = GPIO_PIN_1;  
g_led2.dir = GPIO_INPUT_MODE;  
g_led2.value = GPIO_LOW_LEVEL;  
g_led2.interruptMode = GPIO_INT_TYPE_RISE_EDGE;  
HAL_GPIO_Init(&g_led2);  
HAL_GPIO_RegisterCallback(&g_led2, GPIO_PIN_1, GPIO_CallbackFunc);  
HAL_GPIO_IRQService(&g_led2);  
IRQ_SetPriority(g_led2.irqNum, 1); /* set gpio1 interrupt priority to 1, 1~7 */  
IRQ_EnableN(g_led2.irqNum); /* gpio interrupt enable */
```

1.2.1.2 解决办法

解决办法：合并同组GPIO下不同pin的handle名称为一个名称

由于中断注册的时候每个中断号只会注册第一个回调函数，如果采用上述方式进行同组GPIO不同pin的初始化，则在注册回调函数的时候导致第二个pin的回调函数失效，正确的解法如下，合并同组不同pin的handle名称。

```
g_led.baseAddress = GPIO0;  
g_led.pins = GPIO_PIN_0 | GPIO_PIN_1;  
g_led.dir = GPIO_INPUT_MODE;  
g_led.value = GPIO_LOW_LEVEL;  
g_led.interruptMode = GPIO_INT_TYPE_RISE_EDGE;  
HAL_GPIO_Init(&g_led);  
HAL_GPIO_RegisterCallback(&g_led, GPIO_PIN_0, GPIO_CallbackFunc);  
HAL_GPIO_RegisterCallback(&g_led, GPIO_PIN_1, GPIO_CallbackFunc);  
HAL_GPIO_IRQService(&g_led);  
IRQ_SetPriority(g_led.irqNum, 1); /* set gpio1 interrupt priority to 1, 1~7 */  
IRQ_EnableN(g_led.irqNum); /* gpio interrupt enable */
```



2 sample 使用类

2.1 调试打印

2.1.1 调用打印接口，无法正常打印

2.1.1.1 问题描述

DBG_PRINTF_USE宏定义定义为DBG_USE_UART_PRINTF后，DBG_PRINTF()默认使用UART来进行打印，此时直接调用打印接口DBG_PRINTF()，没有打印输出。

2.1.1.2 解决办法

默认打印是通过UART来实现，所以在使用打印DBG_PRINTF前请确保已经用HAL_UART_Init来初始化串口。

此外，其他需要注意内容如下：

- 如果需要使能打印，请包含头文件“debug.h”。
- 需要将feature.h中的DBG_PRINTF_USE宏定义定义为DBG_USE_UART_PRINTF，否则没有串口打印输出。
- 为了让UART可以正常输出，请确保UART在IOConifg()调用后，再进行初始化。否则UART会因为未初始化而不能正常执行。
- 支持UART打印，SDK默认为UART0输出打印。需要修改的话，在feature.h修改DBG_PRINTF_UART_PORT宏定义，来选择其他串口。



3 其他注意事项

3.1 寄存器读写不符合预期、读写后出现未定义行为

本章主要描述寄存器读写的限制，寄存器读写访问不同于普通的RAM，必须使用32bits的形式来进行读写，否则有可能因编译器不同或者编译选项不同出现异常行为，包括写不成功或者总线访问出错的现象。

3.1.1 IP 寄存器读写前必须 CRG 使能

3.1.1.1 问题描述

直接对某个模块的寄存器进行写操作，然后再读取，出现写操作的数值和读取出来的数值不一样。

3.1.1.2 解决办法

访问IP寄存器前，必须通过HAL_CRG_IpEnableSet将该IP时钟使能，否则会出现读写不成功的情况，以UART为例，以下为使能UART0时钟的操作。

```
static void UART0_Init(void)
{
    HAL_CRG_IpEnableSet(UART0_BASE, IP_CLK_ENABLE);
    HAL_CRG_IpClkSelectSet(UART0_BASE, CRG_PLL_NO_PREDV);
    ....
}
```

以上两行代码为UART0模块的时钟使能。

3.1.2 错误的寄存器访问操作

3.1.2.1 问题描述

按8/16位比特读写寄存器，可能导致程序行为不可知。

出错示例：

```
/* I2C寄存器定义 */
typedef union {
```



```
unsigned int reg;
struct {
    unsigned int i2c_enable      : 1;
    unsigned int reserved0      : 7;
    unsigned int sda_hold_duration : 16;
    unsigned int reserved1      : 8;
} BIT;
} I2C_GLB_REG;

typedef struct {
    I2C_GLB_REG      I2C_GLB;          /**< Offset Address: 0x0000. */
    ...
} volatile I2C_RegStruct;

/* 错误示例代码 */
I2C_RegStruct *i2cReg = address; /* address是I2C寄存器基地址 */
unsigned int duration = i2cReg->I2C_GLB.BIT.sda_hold_duration; /* 直接对地址按16bits进行读操作 */
```

如上的寄存器操作中，sda_hold_duration在寄存器中占16bits，如果直接对读操作，会造成程序异常。

3.1.2.2 解决办法

先将32bit的寄存器读取到变量中，再读变量按位读取，避免对地址直接按8/16bit读写寄存器。除了示例中的寄存器外，用户在读写8/16bit时，需要注意的寄存器列表如表3-1所示。

正确代码写法：

```
I2C_RegStruct *i2cReg = address; /* address是I2C寄存器基地址 */
I2C_GLB_REG glb;
unsigned int duration;
glb.reg = i2cReg->I2C_GLB.reg; /* 将寄存器读到32位的结构体变量中 */
duration = glb.BIT.sda_hold_duration; /* 对32位的结构体变量再进行读操作 */
```

表 3-1 8/16bit 读写需要注意的寄存器列表

模块名	寄存器名
I2C	I2C_GLB
	I2C_DEV_ADDR
	I2C_DATA_BUF
	I2C_PATTERN_DATA1
	I2C_PATTERN_DATA2
DAC	DAC_CTRL
CAN	IF1_DATA_A1
	IF1_DATA_A2
	IF1_DATA_B1
	IF1_DATA_B2



模块名	寄存器名
	IF2_DATAA1
	IF2_DATAA2
	IF2_DATAB1
	IF2_DATAB2
FLASH	EFLASH_CAPACITY_1
	BUF_CLEAR
PMC	LOWPOWER_STATUS
SPI	SPICR0
APT	EM_WD_CNT
	EM_WD_STS
	EM_VCAP_STS1
	EM_TCAP_VAL
ADC	ADC_INT1_CTRL
	ADC_INT2_CTRL
	ADC_INT3_CTRL
	ADC_INT4_CTRL
	ADC_PPB0_PPB1_DLY
	ADC_PPB2_PPB3_DLY
CMM	CMVER
CFD	CFDVER
QDM	QDMVER
SYSCTRL	SC_RST_CNT0
	SC_RST_CNT1
	APT_POE_FILTER
	APT_EVTIO_FILTER
	APT_EVTMP_FILTER

3.2 对寄存器读取后进行移位操作，不符合预期

寄存器读写访问不同于普通的RAM，必须使用32bits的形式来进行操作，否则有可能因编译器不同或者编译选项不同出现异常行为。



3.2.1 问题描述

对寄存器地址读出数据后赋值给局部变量，再对局部变量进行移位操作，结果不符合预期。产生结果的原因是，局部变量如果没有在其他地方使用，编译器有可能把局部变量优化掉，变成对寄存器直接进行移位操作，导致异常。

3.2.2 解决办法

在对寄存器地址操作时，为了避免被工具链优化，在对寄存器地址添加关键字“volatile”，阻止编译器对寄存器的读、写、移位等操作进行不符合预期的优化。

3.3 全局变量定义使用，赋值不生效

3.3.1 问题描述

全局变量使用，在中断中赋值给该全局变量值，在中断外再读取此全局变量，上一步赋值操作不生效。

错误使用示例：

```
static unsigned int g_flag; /* 定义全局变量 */
.....
void InterruptCallBack(void)
{
    g_flag = 1;           /* 中断中修改标志位 */
}

int main (void)
{
    .....
    while ( g_flag == 1 ) { /* 中断外读取全局变量 */
        .....
    }
}
```

3.3.2 解决办法

如果全局变量在中断中进行更新，而且其他位置进行了读操作，请确保用volatile来修饰该全局变量。否则读该全局变量的处理可能被优化，未按预期循环读取。

正确示例：

```
static volatile unsigned int g_flag; /* 定义全局变量，添加了volatile进行修饰 */
.....
void InterruptCallBack(void)
{
    g_flag = 1;           /* 中断中修改标志位 */
}

int main (void)
{
    .....
    while ( g_flag == 1 ) { /* 中断外读取全局变量 */
        .....
    }
}
```



3.4 JTAG 管脚相关复选功能配置不生效

3.4.1 问题描述

例如：GPIO1_3/GPT1_PWM/CAPM2_SRC0/UART1_RXD/I2C0_SCL等与JTAG复用pin脚的模块，复选为非JTAG功能时无法正常工作，如：UART1接收不到数据，GPIO1_3翻转不生效，GPT1无法产生PWM。

复用关系如下：

GPIO1_3/GPT1_PWM/CAPM2_SRC/UART1_RXD/I2C0_SCL 复用IOCMG14。

3.4.2 解决办法

由于系统启动时，boot引脚电平决定当前系统处于正常启动还是升级模式，而试制阶段考虑调试升级功能，boot管脚外围电路采用上拉设计使系统处于升级模式，则单板上电复位后，JTAG相关管脚强制为JTAG功能，且无法通过软件切换为其他复用功能。从而导致JTAG端口复用功能配置失效，需要更改boot管脚外围电路下拉设计使系统状态寄存器处于正常启动模式或通过写系统升级清除标识寄存器，然后再配置端口复用。写系统升级清除标识寄存器的代码如下。与JTAG管脚相关的复选功能均存在以上问题，考虑到量产安全和JTAG调试功能，因此建议此解决办法只适用于试制和实验室调试阶段。

```
SYSCtrl0->SC_SYS_STAT.BIT.update_mode_clear = 1;
```

3.5 串口首次烧录打印乱码问题

3.5.1 问题描述

通过IDE串口烧录文件后，如果程序运行之初就有大量打印，可能出现乱码。通过按复位键硬复位才恢复正常。

问题的原因在于：串口在PC串口打开前已经发送了大量数据，导致串口fifo满，导致pc串口软件打开后，接收数据时因为数据丢失，导致bit错位。

3.5.2 解决办法

按复位键重新执行程序。

3.6 硬件配置不当引起的 I2C 通信失败问题

3.6.1 问题描述

在软件正确配置I2C模块初始化后，通过I2C通信管脚与对接设备进行通信，可能出现无数据信号发出或通信过程中时钟紊乱，导致通信失败。



3.6.2 解决办法

1. 通信过程中无数据信号发出：
 - a. I/O复用功能需配置为I2C通信管脚。在使用I2C管脚进行通信时，需将I/O复用关系配置为I2C_SCL和I2C_SDA功能，并且通信需使用此两个管脚进行通信；
 - b. I2C_SCL管脚和I2C_SDA管脚需连接上拉电阻。由于I2C通信管脚是开漏输出，无电平拉高能力，在I2C通信时I2C_SCL管脚和I2C_SDA管脚需要连接上拉电阻；
2. 通信过程中时钟紊乱现象：

I2C通信时不仅需要连接通信两端的SCL和SDA线，还需连接GND管脚，保证通信两端设备共地。