User Manual

AMetal-AM118-Core 用户手册

AMetal

V2.0.2 Date:2019/10/29

产品用户手册

类别	内容
关键词	AM118-Core、功能介绍
摘　要	本文档简述了 AM118-Core 硬件资源，详细介绍了 ametal_am118_core 软件包的结构、配置方法等。

修订历史

版本	日期	原因
发布 2.0.2	2019/10/29	创建文档

目 录

1. 开发平台简介

AM118-Core 开发平台主要用于 ZLG118 系列微控制器的学习和开发，配套 AMetal 软件包，提供了各个外设的驱动程序、丰富的例程和详尽的资料，是工程师进行项目开发的首选。该平台也可用于教学、毕业设计及电子竞赛等，开发平台如 图 1.1 所示。

注意： 当前提供的 SDK 支持的开发平台版本为 **ZLG118-Core Rev.A 150917 P/N: 1.14.07.1242**，版本号可以在开发平台的背面找到。用户在使用 SDK 前请确认 SDK 支持自己手中的开发平台。

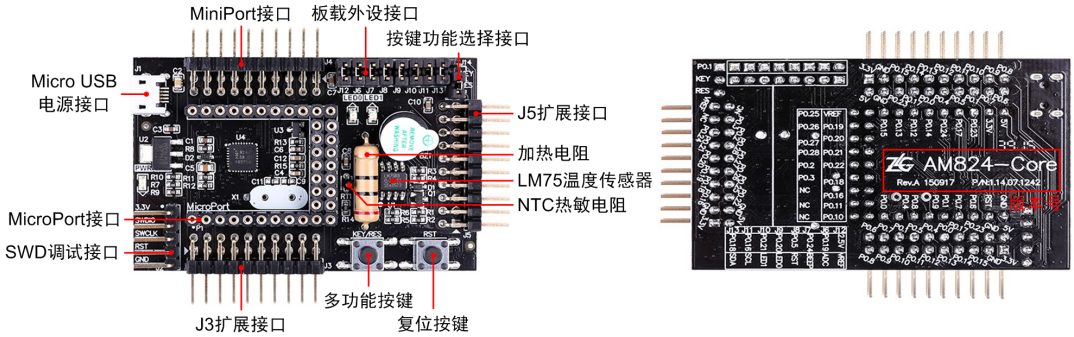

图 1.1 AM118-Core 开发平台

AM118-Core 开发板基于 ZLG 的 ZLG118 微控制器，其外形小巧、结构简单、片上资源设计合理。不到名片大小的电路板上包含了 1 路 MiniPort 接口、1 路 MicroPort 接口和 2 路 2×10 扩展接口。这些接口不仅把单片机的所有 I/O 资源引出，还可以借助 MiniPort 接口和 MicroPort 接口外扩多种模块。

1.1 ZLG118

- 工作电压 1.8V~5.5V；
- 低功耗；
- ARM Cortex-M0+ 内核；
- 2 个 AHB 总线 Master；
- 4 个 AHB 总线 Slaves；
- 256KB FLASH，32KB SRAM；
- 1 个 12 位 ADC，1M SPS 转换速度，30 路输入通道；
- 1 个 8/12 位 DAC，支持 DMA 功能；
- 4 个通用定时器，3 个高级定时器；
- 双通道 DMA 控制器。
- 4 个 UART 接口、2 个 LPUART 接口、2 个 I^2C 接口、2 个 SPI 接口；
- 80 位的芯片唯一 ID；
- LQFP100 封装；

1.2 AM118-Core

- 5VMicroUSB 供电；
- 2 个 LED 发光二极管；
- 1 个无源蜂鸣器；
- 1 个加热电阻；
- 1 个 LM75B 测温芯片；
- 1 个多功能按键（可用跳线帽选择用作加热按键或是独立按键）；
- 1 个复位按键；

基于这些资源，可以完成多种基础实验。例如：加热电阻配合 LM75B 和热敏电阻可分别实现数字和模拟测温。

2. AMetal 软件包

软件包名为 ametal_am118_core_1.0.0（不同版本，版本号会有区别，请以实际获取的 AMetal 包版本为准）。为叙述方便，下文简称软件包为 SDK，使用 {SDK} 表示软件包的路径。

2.1 AMetal 架构

如 图 2.1 所示，AMetal 共分为 3 层，硬件层、驱动层和标准接口层。

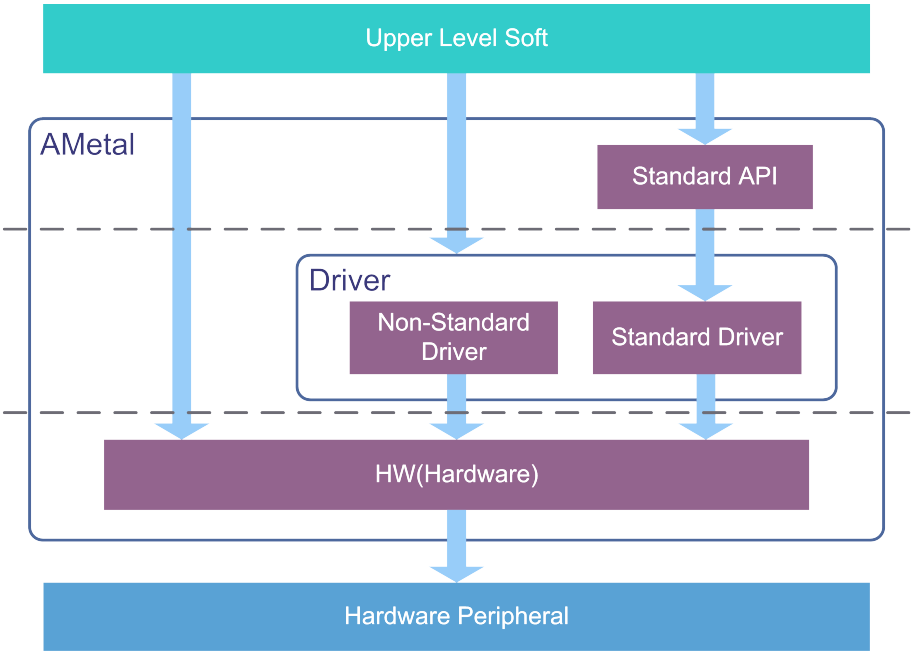

图 2.1 AMetal 框架

根据实际需求，这三层对应的接口均可被应用程序使用。对于 AWorks 平台或者其他操作系统，它们可以使用 AMetal 的标准接口层接口开发相关外设的驱动。这样，AWorks 或者其它操作系统在以后的使用过程中，针对提供相同标准服务的不同外设，不需要再额外开发相对应的驱动。

2.1.1 硬件层

硬件层对 SOC 做最原始封装，其提供的 API 基本上是直接操作寄存器的内联函数，效率最高。当需要操作外设的特殊功能，或者对效率、特殊使用等有需求时，可以调用硬件层 API。硬件层等价于传统 SOC 原厂的裸机包。硬件层接口使用 **amhw_/AMHW_ + 芯片名**作为命名空间，如 **amhw_zlg118** 、**AMHW_ZLG118** 。

参见:

更多的硬件层接口定义及示例请参考 {SDK}**documents**\《**AMetal API 参考手册.chm**》或者 {SDK}**ametal\soc\zlg\drivers\hw**\drivers\hw** 文件夹中的相关文件。

注解: 本文使用 SOC(System On Chip) 泛指将 CPU 和外设封装在一起的 MCU、DSP 等微型计算机系统。

2.1.2 驱动层

虽然硬件层对外设做了封装，但其通常与外设寄存器的联系比较紧密，用起来比较繁琐。为了方便使用，驱动层在硬件层的基础上做了进一步封装，进一步简化对外设的操作。

根据是否实现了标准层接口可以划分为标准驱动和非标准驱动，前者实现了标准层的接口，例如 GPIO、UART、SPI 等常见的外设；后者因为某些外设的特殊性，并未实现标准层接口，需要自定义接口，例如 DMA、TSI 等。驱动层接口使用 **am_/AM_ + 芯片名**作为命名空间，如 **am_zlg118** 、**AM_ZLG118** 。

参见:

更多的驱动层接口定义及示例请参考 {SDK}**documents**\《**AMetal API 参考手册.chm**》或者 {SDK}**ametal\soc\zlg\drivers** 文件夹中的相关文件。

2.1.3 标准接口层

标准接口层对常见外设的操作进行了抽象，提取出了一套标准 API 接口，可以保证在不同的硬件上，标准 API 的行为都是一样的。标准层接口使用 **am_/AM_** 作为命名空间。

参见:

更多的标准接口定义及示例请参考 {SDK}**documents**\《**AMetal API 参考手册.chm**》或者 {SDK}**ametal\interface** 文件夹中的相关文件。

2.2 目录结构

关于软件包的目录结构信息可参考软件包中的 documents 文件夹下的《AMetal API 参考手册 V1.10.chm》或者\ametal\interface 文件夹中的相关文件。

2.3 工程结构

2.3.1 Keil 工程结构

打开 Keil 版本的 template_am118_core 模板工程，其工程结构如 图 2.2 所示。

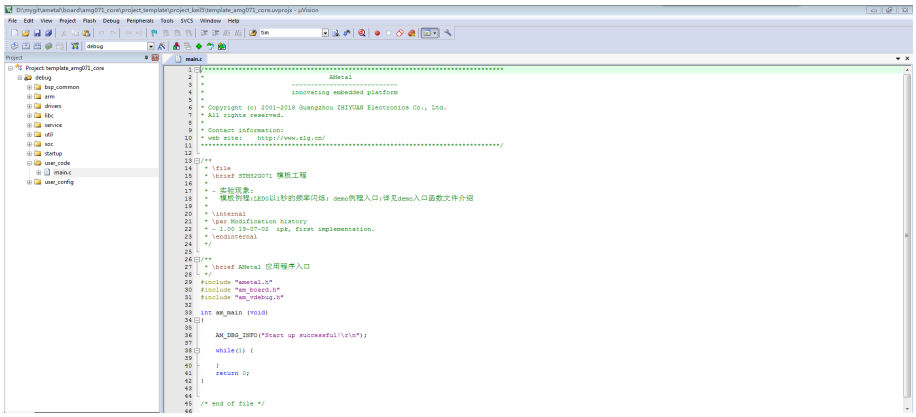

图 2.2 Keil 版本工程模板结构

在工程结构中，包含了 bsp_common、arm、drivers、libs、service、util、soc、startup、user_code、user_config 共 10 个节点。

- **bsp_common** 存放板级通用文件，如系统堆的适配文件，C 库的板级适配文件等。
- **arm** 文件夹存放了与内核相关的通用文件，如 NVIC、Systick 等。
- **drivers** 文件夹包含通用外设及模块的标准驱动文件，如按键、矩阵键盘、温度传感器模块、ZigBee 模块等。
- **libc** 文件夹下存放的是 C 库适配文件，如 armlib_adpater，主要用于适配相应的 C 库。
- **service** 文件夹主要存放通用的服务组件的驱动文件，如 ADC、蜂鸣器。
- **util** 文件夹主要 AMetal 通用的辅助工具文件，如堆管理、软件定时器以及打印输出函数等。
- **soc** 文件夹主要包含了与芯片密切相关的文件，主要是硬件层和驱动层。
- **startup** 文件夹下包含启动相关文件。
- **user_code** 下为用户程序，相关文件位于 {PROJECT}\user_code 文件夹下（为叙述方便，下文均以｛PROJECT｝表示工程所在路径），每个工程对应的应用程序均存放在该目录下，该目录默认有一个 main.c 文件，其中包含 AMetal 软件包的应用程序入口函数 am_main()。用户开发的其他源程序文件均应存放在 user_code 目录下。
- **user_config** 下为配置文件，相关文件位于 {PROJECT}\user_config 文件夹下，不同工程可以有不同的配置。

2.3.2 Eclipse 工程结构

建立 Eclipse 工作空间并导入 template_am118_core 模板工程，其工程结构如 图 2.3 所示。

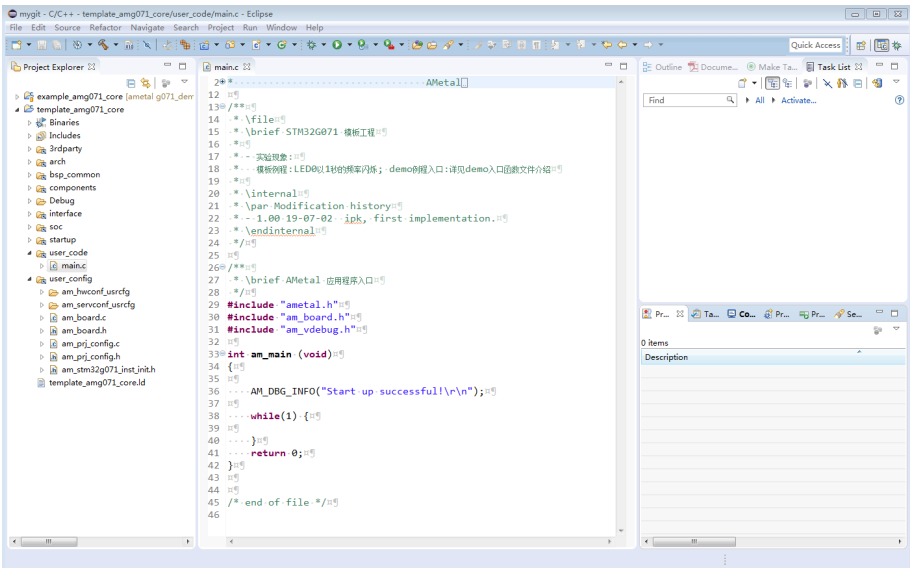

图 2.3 Eclipse 版本工程模板结构

在工程结构中，包含了 3rdparty、arch、bsp_common、components、interface、soc、startup、user_code、user_config 这几个节点。详细介绍见 2.2 章节和 Keil 工程结构小节。

3. 工程配置

由于系统正常工作时，往往需要初始化一些必要的外设，如 GPIO、中断和时钟等。同时，板上的资源也需要初始化后才能正常使用。为了操作方便，默认情况下，这些资源都在系统启动时自动完成初始化，在进入用户入口函数 **am_main()** 后，这些资源就可以直接使用，非常方便。

但是，一些特殊的应用场合，可能不希望在系统启动时自动初始化一些特定的资源。这时，就可以使用工程配置文件 {PROJECT}**\user_config\am_prj_config.h** 文件禁能一些外设或资源的自动初始化。

3.1 部分外设初始化使能/禁能

一些全局外设，如 CLK、GPIO、DMA、INT 和 NVRAM，由于需要在全局使用，因此在系统启动时已默认初始化，在应用程序需要使用时，无需再重复初始化，直接使用即可。相关的宏在工程配置文件 {PROJECT}**\user_config\am_prj_config.h** 中定义。

以 GPIO 为例，其对应的使能宏为：**AM_CFG_GPIO_ENABLE**，详细定义见 列表 3.1 。宏值默认为 1，即 GPIO 外设在系统启动时自动初始化，如果确定系统不使用 GPIO 资源或希望由应用程序自行完成初始化操作，则可以将该宏的宏值修改为 0。

列表 3.1 GPIO 自动初始化使能/禁能配置

```
1  /** \brief 为 1，启动时，完成 GPIO 初始化 */
2  #define AM_CFG_GPIO_ENABLE                1
```

其它一些外设初始化使能/禁能宏定义详见 表 3.1 。配置方式与 GPIO 相同，将宏值修改为 0 即可禁止在系统启动时自动完成初始化。

表 3.1 其它一些外设初始化使能/禁能宏

宏名	对应的外设
AM_CFG_CLK_ENABLE	系统时钟
AM_CFG_INT_ENABLE	中断
AM_CFG_DMA_ENABLE	DMA
AM_CFG_NVRAM_ENABLE	NVRAM

注解：有的资源除使能外，可能还需要其它一些参数的配置，关于外设参数的配置，可以详见 4.2 节。

3.2 板级资源初始化使能/禁能

与板级相关的资源有 LED、蜂鸣器、按键、调试串口、延时、系统滴答、软件定时器、标准库、中断延时和温度传感器 LM75 等。除 LM75 外（用户使用 LM75 时需自行完成初始化操作，详见第 5.2.6 章），其他板级资源都可以通过配置对应的使能/禁能宏来决定系统启动时是否自动完成初始化操作。相关的宏在工程配置文件 {PROJECT}**\user_config\am_prj_config.h** 中定义。

以 LED 为例，其对应的使能宏为：**AM_CFG_LED_ENABLE**，详细定义见 列表 3.2 。宏值默认为 1，即 LED 在系统启动时自动完成初始化，如果确定系统不使用 LED 资源或希望由应用程序自行完成初始化操作，则可以将该宏的宏值修改为 0。

列表 3.2 LED 自动初始化使能/禁能配置

```
1  /**
2   * \brief 如果为 1，则初始化 led 的相关功能，板上默认有两个 LED
3   *
4   * ID: 0 --- PIOC_9 （需要短接跳线帽 J9）
5   * ID: 1 --- PIOA_8 （需要短接跳线帽 J10）
6   */
7  #define AM_CFG_LED_ENABLE           1
```

其它一些板级资源初始化使能/禁能宏定义详见 表 3.2 。配置方式与 LED 相同，将宏值修改为 0 即可禁止在系统启动时自动完成初始化。

表 3.2 其它一些板级资源初始化使能/禁能宏

宏名	对应资源
AM_CFG_BUZZER_ENABLE	蜂鸣器，使能后才能正常使用蜂鸣器
AM_CFG_KEY_GPIO_ENABLE	板载按键，使能后才能正常使用板载按键
AM_CFG_DELAY_ENABLE	延时，使能初始化后才能在应用中直接使用 am_udelay() 和 am_mdelay()
AM_CFG_SYSTEM_TICK_ENABLE	系统滴答，默认使用 MRT 定时器，使能后才能正常使用系统滴答相关功能
AM_CFG_SOFTIMER_ENABLE	软件定时器，使能后才能正常使用软件定时器的相关功能
AM_CFG_DEBUG_ENABLE	串口调试，使能调试输出，使能后，则可以使用 AM_DBG_INFO() 通过串口输出调试信息
AM_CFG_KEY_ENABLE	按键系统，使能后方可管理系统输入事件
AM_CFG_ISR_DEFER_ENABLE	中断延时，使能后，ISR DEFER 板级初始化，将中断延迟任务放在 PENDSV 中处理
AM_CFG_STDLIB_ENABLE	标准库，使能标准库，则系统会自动适配标准库，用户即可使用 printf()、malloc()、free() 等标准库函数
AM_CFG_LED_ENABLE	板载 LED 灯，使能后才能正常使用板载 LED 灯
AM_CFG_CLK_ENABLE	时钟，使能初始化后可以输出时钟信号

注解： 有的资源除使能外，可能还需要其它一些参数的配置，关于这参数的配置，可以详见第 5 章。

对于延时，每个硬件平台可能具有不同的实现方法，默认实现详见 ametal\board\bsp_common\am_bsp_delay_timer.c，应用可以根据具体需求修改（例如：应用程序不需要精确延时，完全可以使用 for 循环去做一个大概的延时即可，无需再额外耗费一个定时器），因此，将延时部分归类到板级资源下。

对于调试输出，即使用一路串口来输出调试信息，打印出一些关键信息以及变量的值等等，非常方便。

对于软件定时器，需要一个硬件定时器为其提供一个的周期性的定时中断。不同的硬件平台可以有不同的提供方式，因此，同样将软件定时器的初始化部分归类到板级资源下。

4. 片上外设资源

ZLG118 包含了众多的外设资源，只要 SDK 提供了对应外设的驱动，就一定会提供一套相应的默认配置信息。所有片上外设的配置由 {PROJECT}**\user_config\am_hwconf_usrcfg** (为叙述简便，下文统一使用 {HWCONFIG} 表示该路径) 下的一组 am_hwconf_zlg118_* 开头的 **.c** 文件完成的。

注解： 为方便介绍本文将与 ARM 内核相关的文件（NVIC 和 Systick）与片上外设资源放在一起，它们的配置文件位于 {HCONFIG} 路径下，以 am_hwconf_arm_* 开头。

片上外设及其对应的配置文件如 表 4.1 所示。

表 4.1 片上外设及对应的配置文件

序号	外设	配置文件
1	ADC	am_hwconf_zlg118_adc.c
2	AES	am_hwconf_zlg118_aes.c
3	NVIC	am_hwconf_zlg118_arm_nvic.c
4	时钟部分 (CLK)	am_hwconf_zlg118_clk.c
5	循环冗余检验 (CRC)	am_hwconf_zlg118_crc.c
6	DAC	am_hwconf_zlg118_dac.c
7	DMA	am_hwconf_zlg118_dma.c
8	GPIO	am_hwconf_zlg118_gpio.c
9	I^2C slave	am_hwconf_zlg118_i2c_slv.c
10	I^2C master	am_hwconf_zlg118_i2c.c
11	LCD	am_hwconf_zlg118_lcd.c
12	LED_GPIO	am_hwconf_zlg118_led_gpio.c
13	LPTIM	am_hwconf_zlg118_lptim.c
14	LPUART	am_hwconf_zlg118_lpuart.c
15	LVD	am_hwconf_zlg118_lvd.c
16	OPA	am_hwconf_zlg118_opa.c
17	PCNT	am_hwconf_zlg118_pcnt.c
18	RTC	am_hwconf_zlg118_rtc.c
19	SPI(中断方式)	am_hwconf_zlg118_spi_int.c
20	SPI(轮询方式)	am_hwconf_zlg118_spi_poll.c
21	SPI(DMA)	am_hwconf_zlg118_spi_dma.c
22	SYSTICK	am_hwconf_zlg118_systick.c
23	TIM(捕获方式)	am_hwconf_zlg118_tim_cap.c
24	TIM(PWM 方式)	am_hwconf_zlg118_tim_pwm.c
25	TIM(计时方式)	am_hwconf_zlg118_tim_timing.c
26	TRNG	am_hwconf_zlg118_trng.c
27	UART	am_hwconf_stm32g071_uart.c
28	VC	am_hwconf_zlg118_vc.c
29	WDT	am_hwconf_stm32g071_wdt.c

每个外设都提供了对应的配置文件，使得看起来配置文件的数量非常之多。但实际上，所有配置文件的结构和配置方法都非常类似，同时，由于所有的配置文件已经是一种常用的默认配置，因此，用户在实际配置时，需要配置的项目非常之少，往往只需要配置外设相关的几个引脚号就可以了。

4.1 配置文件结构

配置文件的核心是定义一个设备实例和设备信息结构体，并提供封装好的实例初始化函数和实例解初始化函数。下面以 GPIO 为例，详述整个配置文件的结构。

4.1.1 设备实例

设备实例为整个外设驱动提供必要的内存空间，设备实例实际上就是使用相应的设备结构体类型定义的一个结构体变量，无需用户赋值。因此，用户完全不需要关心设备结构体类型的具体成员变量，只需要使用设备结构体类型定义一个变量即可。在配置文件中，设备实例均已定义。打开 {HWCONFIG}**am_hwconf_stm32g071_gpio.c**，可以看到设备实例已经定义好。详见 列表 4.1 。

列表 4.1 定义设备实例

```
/** \brief GPIO 设备实例 */
am_zlg118_gpio_dev_t __g_gpio_dev;
```

这里使用 **am_zlg118_gpio_dev_t** 类型定义了一个 GPIO 设备实例。设备结构体类型在相对应的驱动头文件中定义。对于通用输入输出 GPIO 外设，该类型即在 {SDK}**ametal\soc\zlg\drivers\include\am_zlg118_gpio.h** 文件中定义。

4.1.2 设备信息

设备信息用于在初始化一个设备时，传递给驱动一些外设相关的信息，如常见的该外设对应的寄存器基地址、使用的中断号等等。设备信息实际上就是使用相应的设备信息结构体类型定义的一个结构体变量，与设备实例不同的是，该变量需要用户赋初值。同时，由于设备信息无需在运行过程中修改，因此往往将设备信息定义为 **const** 变量。

打开 {HWCONFIG}**am_hwconf_stm32g071_gpio.c**，可以看到定义的设备信息如 列表 4.2 所示。

列表 4.2 GPIO 设备信息定义

```
/** \brief GPIO 设备信息 */
const am_zlg118_gpio_devinfo_t __g_gpio_devinfo = {
    ZLG118_PORT0_BASE,              /**< \brief GPIO 控制器寄存器块基址 */

    {
        INUM_PORTA,
        INUM_PORTB,
        INUM_PORTC_E,
        INUM_PORTD_F,
    },

    PIN_NUM,                        /**< \brief GPIO PIN 数量 */
    PIN_INT_MAX,                    /**< \brief GPIO 使用的最大外部中断线编号 +1 */

    &__g_gpio_infomap[0],           /**< \brief GPIO 引脚外部事件信息 */
    &__g_pin_afio[0],               /**< \brief GPIO PIN 重映像信息 */
    &__g_gpio_triginfos[0],         /**< \brief GPIO PIN 触发信息 */

    __zlg118_plfm_gpio_init,        /**< \brief GPIO 平台初始化 */
    __zlg118_plfm_gpio_deinit       /**< \brief GPIO 平台去初始化 */
};
```

这里使用 **am_zlg118_gpio_devinfo_t** 类型定义了一个 GPIO 设备信息结构体。设备信息结构体类型在相应的驱动头文件中定义。对于 GPIO，该类型在 {SDK}**ametal\soc\zlg\drivers\include\gpio\am_zlg118_gpio.h** 文件中定义。详见 列表 4.3 。

列表 4.3 GPIO 设备信息结构体类型定义

```
/**
 * \brief GPIO 设备信息
 */
typedef struct am_zlg118_gpio_devinfo {

    /** \brief GPIO 寄存器块基址 */
    uint32_t      gpio_regbase;

    /** \brief 引脚中断号列表 */
    const int8_t  inum_pin[4];

    /** \brief GPIO 引脚数量 */
    const uint8_t pin_count;

    /** \brief GPIO 使用的最大外部中断线编号 +1 */
    const uint8_t exti_num_max;

    /** \brief 触发信息映射 */
    uint8_t      *p_infomap;

    /** \brief 引脚重映像外设信息 */
    amhw_zlg118_gpio_afio_t              *p_afio;

    /** \brief 指向引脚触发信息的指针 */
    struct am_zlg118_gpio_trigger_info   *p_triginfo;

    void (*pfn_plfm_init)(void);   /**< \brief 平台初始化函数 */

    void (*pfn_plfm_deinit)(void); /**< \brief 平台去初始化函数 */

} am_zlg118_gpio_devinfo_t;
```

可见，共计有 9 个成员，看似很多，这是由于 GPIO 关联的外设较多，SWM、GPIO、IOCON、PINT 等都统一归到 GPIO 下管理，仅寄存器基地址就有三个成员。无论怎样，设备信息一般仅由 5 部分构成：寄存器基地址、中断号、需要用户根据实际情况分配的内存、平台初始化函数和平台解初始化函数。下面一一解释各个部分的含义。

1. 寄存器基地址

每个片上外设都有对应的寄存器，这些寄存器有一个起始地址（基地址），只要根据这个起始地址，就能够操作到所有寄存器。因此，设备信息需要提供外设的基地址。

一般来讲，外设关联的寄存器基地址都只有一个，而 GPIO 属于较为特殊的外设，它统一管理了 GPIO SYSCFG EXIT 共计 3 个部分的外设，因此，在 GPIO 的设备信息中，需要四个基地址，对应四个成员变量，分别为：gpio_regbase、syscfg_regbase 和 exit_regbase。

寄存器基地址已经在 {SDK}**\ametal\soc\st\stm32g071\stm32g071_regbase.h** 文件中使用宏定义好了，用户直接使用即可。对于 GPIO 相关的寄存器基地址，详见 列表 4.4 。

列表 4.4 外设寄存器基地址定义

```
/** \brief GPIO 基地址 */
#define ZLG118_GPIO_BASE              (0x50000000UL)

/** \brief SYSCFG 基地址 */
#define ZLG118_SYSCFG_BASE            (0x40010000UL)

/** \brief EXTI 控制器基地址 */
```

```
8  #define ZLG118_EXTI_BASE              (0x40021800UL)
```

可见，列表 4.3 中，设备信息前 3 个成员的赋值均来自于此。

2. 中断号

中断号对应了外设的中断服务入口，需要将该中断号传递给驱动，以便驱动使用相应的中断资源。

对于绝大部分外设，中断入口只有一个，因此中断号也只有一个，一些特殊的外设，中断号可能存在多个，如 GPIO，中断的产生来源于 EXIT，EXIT 最高可提供 16 路中断，为了快速响应，与之对应地，系统提供了 16 路中断服务入口给 EXIT，因此，PINT 共计有 4 个中断号，在设备信息结构体类型中，为了方便提供所有的中断号，使用了一个大小为 4 的数组。详见 列表 4.5 。

列表 4.5 GPIO 设备信息结构体类型——中断号成员定义

```
1  /** \brief 引脚中断号列表 */
2  const int8_t  inum_pin[4];
```

所有中断号已经在 {SDK}**\ametal\soc\st\stm32g071\stm32g071_inum.h** 文件中定义好了，与 PINT 相关的中断号定义详见 列表 4.6 。

列表 4.6 PINT 各个中断号定义

```
1  #define INUM_PORTA                0    /**< \brief PORTA 中断 */
2  #define INUM_PORTB                1    /**< \brief PORTB 中断 */
3  #define INUM_PORTC_E              2    /**< \brief PORTC_E 中断 */
4  #define INUM_PORTD_F              3    /**< \brief PORTD_F 中断 */
```

实际为结构体信息的中断号成员赋值时，只需要使用定义好的宏为相应的设备信息结构体赋值即可。可见，列表 4.3 中，设备信息中断号成员的赋值均来自于此。

3. 时钟 ID 号

时钟 ID 号对应了外设的时钟来源，需要将该时钟 ID 号传递给驱动，以便驱动中可以概外设的频率及使能该外设的相关时钟。所有时钟 ID 号已经在 {SDK}**\ametal\soc\zlg\zlg118\zlg118_clk.h** 文件中定义好了，详见 列表 4.7 。

列表 4.7 时钟 ID 号

```
1  /* zlg118 所有外设时钟均挂载在 PCLK 时钟下 */
2  #define CLK_FLASH     ((0x1ul << CLK_PCLK) | 31ul)
3  #define CLK_DMA       ((0x1ul << CLK_PCLK) | 29ul)
4  #define CLK_GPIO      ((0x1ul << CLK_PCLK) | 28ul)
5  #define CLK_AES       ((0x1ul << CLK_PCLK) | 27ul)
6  #define CLK_CRC       ((0x1ul << CLK_PCLK) | 26ul)
7  #define CLK_SWD       ((0x1ul << CLK_PCLK) | 25ul)
8  #define CLK_TICK      ((0x1ul << CLK_PCLK) | 24ul)
9  #define CLK_LCD       ((0x1ul << CLK_PCLK) | 22ul)
10 #define CLK_CLOCKTRIM ((0x1ul << CLK_PCLK) | 21ul)
11 #define CLK_RTC       ((0x1ul << CLK_PCLK) | 20ul)
12 #define CLK_PCNT      ((0x1ul << CLK_PCLK) | 19ul)
13 #define CLK_RNG       ((0x1ul << CLK_PCLK) | 18ul)
14 #define CLK_VC_LVD    ((0x1ul << CLK_PCLK) | 17ul)
15 #define CLK_ADC_BGR   ((0x1ul << CLK_PCLK) | 16ul)
16 #define CLK_WDT       ((0x1ul << CLK_PCLK) | 15ul)
17 #define CLK_PCA       ((0x1ul << CLK_PCLK) | 14ul)
18 #define CLK_OPA       ((0x1ul << CLK_PCLK) | 13ul)
```

```
#define CLK_TIM3      ((0x1ul << CLK_PCLK) | 11ul)
#define CLK_TIM456    ((0x1ul << CLK_PCLK) | 10ul)
#define CLK_LPTIM0    ((0x1ul << CLK_PCLK) | 9ul)
#define CLK_TIM012    ((0x1ul << CLK_PCLK) | 8ul)
#define CLK_SPI1      ((0x1ul << CLK_PCLK) | 7ul)
#define CLK_SPI0      ((0x1ul << CLK_PCLK) | 6ul)
#define CLK_I2C1      ((0x1ul << CLK_PCLK) | 5ul)
#define CLK_I2C0      ((0x1ul << CLK_PCLK) | 4ul)
#define CLK_LPUART1   ((0x1ul << CLK_PCLK) | 3ul)
#define CLK_LPUART0   ((0x1ul << CLK_PCLK) | 2ul)
#define CLK_UART1     ((0x1ul << CLK_PCLK) | 1ul)
#define CLK_UART0     ((0x1ul << CLK_PCLK) | 0ul)
#define CLK_UART3     ((0x1ul << CLK_PCLK) | 40ul)
#define CLK_UART2     ((0x1ul << CLK_PCLK) | 39ul)
#define CLK_LPTIM1    ((0x1ul << CLK_PCLK) | 35ul)
#define CLK_DAC       ((0x1ul << CLK_PCLK) | 34ul)
```

在 GPIO 设备信息当中，没有使用时钟 ID 号，故不需要配置。在很大一部分外设，需要使用时钟 ID 号，如串口外设，详见 列表 4.8 。

列表 4.8 串口外设时钟 ID 号

```
/** \brief 串口 0 设备信息 */
static const am_zlg118_uart_devinfo_t __g_uart0_devinfo = {

    ZLG118_UART0_BASE,            /**< \brief 串口 0 */
    INUM_UART0_2,                 /**< \brief 串口 0 的中断编号 */
    AMHW_ZLG118_UART_WORK_MODE_1, /**< \brief 串口工作模式 */
    CLK_UART0,                    /**< \brief 串口 0 的时钟 */
    AMHW_ZLG118_UART_CLK_DIV_MODE13_8,/* 8 分频 */

    AMHW_ZLG118_UART_PARITY_NO |  /**< \brief 无极性 */
    AMHW_ZLG118_UART_STOP_1_0_BIT,/**< \brief 1 个停止位 */

    115200,                       /**< \brief 设置的波特率 */

    0,                            /**< \brief 无其他中断 */

    {
        AM_FALSE,                 /**< \brief 禁能多机地址自动识别 */
        0x00,                     /**< \brief 地址 0x00 */
        0x00,                     /**< \brief 地址全部不关心 */
    },

    {
        AM_FALSE,                 /**< \brief 禁能流控 */
        0,                        /**< \brief CTS 引脚编号 */
        0,                        /**< \brief RTS 引脚编号 */
    },

    NULL,                         /**< \brief 使用 RS485 */
    __zlg118_plfm_uart0_init,     /**< \brief UART0 的平台初始化 */
    __zlg118_plfm_uart0_deinit,   /**< \brief UART0 的平台去初始化 */
};
```

4. 需要用户根据实际情况分配的内存

前文已经提到，设备实例是用来为外设驱动分配内存的，为什么在设备信息中还需要分配内存呢？

这是因为系统有的资源提供得比较多，而用户实际使用数量可能远远小于系统提供的

资源数，如果按照默认都使用的操作方式，将会造成不必要的资源浪费。

以 PINT 为例，系统提供了 16 路外部中断，因此，最多可以将 16 个 GPIO 用作触发模式。每一路 GPIO 触发模式就需要内存来保存用户设定的触发回调函数。如果按照默认，假定用户可能会使用到所有的 16 路 GPIO 触发，则就需要 16 份用于保存相关信息的内存空间。而实际上，用户可能只使用 1 路，这就导致了不必要的空间浪费。

基于此，某些可根据用户实际情况增减的内存由用户通过设备信息提供。以实现资源的最优化利用。

在设备信息结构体类型中，相关的成员有 3 个，详见 列表 4.9 。

列表 4.9 GPIO 设备信息结构体类型——内存分配相关成员定义

```
/** \brief GPIO 使用的最大外部中断线编号 +1 */
const uint8_t exti_num_max;

/** \brief 触发信息映射 */
uint8_t      *p_infomap;

......

/** \brief 指向引脚触发信息的指针 */
struct am_zlg118_gpio_trigger_info   *p_triginfo;
```

- 成员 **exti_num_max** 使用的最大中断线加 1，例如仅使用 EXTI 线 0（对应引脚为 PIOA0/PIOB0/PIOC0/PIOD0），则该变量应为 1。
- 成员 **p_infomap** 用于保存触发信息的映射关系，对应内存的大小应该与 **exti_num_max** 一致。
- 成员 **p_triginfo** 用于保存触发信息，主要包括触发回调函数和回调函数对应的参数，对应内存的大小应该与 **exti_num_max** 一致。类型 **am_zlg118_gpio_trigger_info** 同样在 GPIO 驱动头文件 {SDK}**\ametal\soc\zlg\drivers\include\gpio\am_stm32g071_gpio.h** 中定义，详见 列表 4.10 。

列表 4.10 GPIO 触发信息结构体类型定义

```
/**
 * \brief 引脚的触发信息
 */
struct am_zlg118_gpio_trigger_info {

    /** \brief 触发回调函数 */
    am_pfnvoid_t  pfn_callback;

    /** \brief 回调函数的参数 */
    void *p_arg;
};
```

可见，虽然有三个成员，但实际上可配置的核心就是 **exti_num_max**，另外两个成员的实际的内存大小应该与该参数一致，为了方便用户根据实际情况配置，用户配置文件中，提供了默认的这三个成员的值定义，详见 列表 4.11 。

列表 4.11 需要用户根据实际情况分配的内存定义

```
/** \brief 引脚触发信息内存 */
static struct am_zlg118_gpio_trigger_info __g_gpio_triginfos[PIN_INT_MAX];

```

```
/** \brief 引脚触发信息映射 */
static uint8_t __g_gpio_infomap[PIN_INT_MAX];
```

设备信息结构体的赋值详见 __g_gpio_devinfo ，其中，使用 **PIN_INT_MAX** 宏作为 **pint_count** 的值；使用 **__g_gpio_infomap** 作为 **p_infomap** 的值；使用 **__g_gpio_triginfos** 作为 **p_triginfo** 的参数。

默认情况下，**PIN_INT_MAX** 的值即为硬件支持的最大 PINT 通道数目，如果用户实际使用到的 GPIO 触发数目小于该值，可以修改为实际使用到的数量，如 5。

注解： 实际中，有的外设可能不需要根据实际分配内存。那么，设备信息结构体中将不包含该部分内容。

---

5. 平台初始化函数

平台初始化函数主要用于初始化与该外设相关的平台资源，如使能该外设的时钟，初始化与该外设相关的引脚等。一些通信接口，都需要配置引脚，如 USART、SPI、I^2C 等，这些引脚的初始化都需要在平台初始化函数中完成。

在设备信息结构体类型中，均有一个用于存放平台初始化函数的指针，以指向平台初始化函数，详见 列表 4.12 。当驱动程序初始化相应外设前，将首先调用设备信息中提供的平台初始化函数。

列表 4.12 GPIO 设备信息结构体类型——平台初始化函数指针定义

```
/** \brief 平台初始化函数 */
void     (*pfn_plfm_init)(void);
```

平台初始化函数均在设备配置文件中定义，GPIO 的平台初始化函数在 {HWCONFIG}**am_hwconf_zlg118_gpio.c** 文件中定义，详见 gpio_code13 。

列表 4.13 GPIO 平台初始化函数

```
/** \brief GPIO 平台初始化 */
void __zlg118_plfm_gpio_init (void)
{
    am_clk_enable(CLK_GPIO);

    am_zlg118_clk_reset(CLK_GPIO);
}
```

平台初始化函数中，使能了与 GPIO 端口的门控时钟。

am_zlg118_clk_reset() 函数用于复位一个外设，在\ametal\soc\zlg\drivers\source\clk\am_zlg118_clk.c 文件中定义，详见 gpio_code14 。

列表 4.14 am_zlg237_clk_reset() 函数原型

```
/**
 * \brief CLK 外设复位
 *
 * \param[in] clk_id 时钟 ID (由平台定义)
 *
 * \retval AM_OK : 操作成功
 */
int am_zlg237_clk_reset (am_clk_id_t clk_id);
```

参数为 am_clk_id_t 类型，用于指定需要复位的外设时钟门控，在\ametal\soc\zlg\zlg118\zlg118_clk.h 文件中定义。可见列表 4.7 。在平台初始化函数中，参数 CLK_IOPA 表示复位 GPIO PORTA 外设。am_clk_enable() 函数用于使能一个外设的时钟，在\ametal\soc\zlg\zlg118\zlg118_clk.h 文件中定义。详见：numref:*gpio_code15* 。

列表 4.15 am_clk_enable() 函数原型

```
/**
 * \brief 使能时钟
 *
 * \param[in] clk_id 时钟 ID (由平台定义)
 *
 * \retval AM_OK 成功
 * \retval -AM_ENXIO 时钟频率 ID 不存在
 * \retval -AM_EIO 使能失败
 */
int am_clk_enable (am_clk_id_t clk_id);
```

参数为 am_clk_id_t 类型，用于指定需要使能时钟的外设，在\ametal\soc\zlg\zlg118\zlg118_clk.h 文件中定义。可见列表 4.7 。平台初始化函数中，参数 CLK_IOPA 和 CLK_AFIO 分别使能了 GPIO PORTA 和 AFIO 的时钟。

在设备信息结构体赋值时，详见 列表 4.3 ，直接以该函数的函数名作为平台初始化函数指针成员的值。

某些特殊的外设，很可能不需要平台初始化函数，这时，只需要将平台初始化函数指针成员赋值为 NULL 即可。

6. 平台解初始化函数

平台解初始化函数与平台初始化函数对应，平台初始化函数打开了的时钟等，就可以通过平台解初始化函数关闭。

在设备信息结构体类型中，均有一个用于存放平台解初始化函数的指针，以指向平台解初始化函数，详见 列表 4.16 。当不再需要使用某个外设时，驱动在解初始化相应外设后，将调用设备信息中提供的平台解初始化函数，以释放掉平台提供的相关资源。

列表 4.16 GPIO 设备信息结构体类型——平台解初始化函数指针定义

```
/** \brief 平台解初始化函数 */
void (*pfn_plfm_deinit)(void);
```

平台解初始化函数均在设备配置文件中定义，GPIO 的平台解初始化函数在 {HWCONFIG}**\am_hwconf_zlg118_gpio.c** 文件中定义，详见 列表 4.17 。

列表 4.17 GPIO 平台解初始化函数

```
/** \brief GPIO 平台解初始化 */
void __zlg118_plfm_gpio_deinit (void)
{
    am_zlg118_clk_reset(CLK_GPIO);

    am_clk_disable(CLK_GPIO);
}
```

平台解初始化函数中，复位了 GPIO，并关闭了各个相关外设的时钟。

amhw_clk_disable() 函数，用于关闭相关外设的时钟，该函数在 {SDK}**\ametal\soc\st\drivers\hw\include\amhw_stm32g071_clk.h** 文件中定义。详见：列

表 4.18。

列表 4.18 am_clk_disable() 函数原型

```
/**
 * \brief 禁能时钟
 *
 * \param[in] clk_id 时钟 ID (由平台定义), 参见\ref grp_clk_id
 *
 * \retval AM_OK 成功
 * \retval -AM_ENXIO 时钟频率 ID 不存在
 * \retval -AM_EIO 禁能失败
 */
int am_clk_disable (am_clk_id_t clk_id);
```

在设备信息结构体赋值时，详见 列表 4.3 ，直接以该函数的函数名作为平台解初始化函数指针成员的值。

某些特殊的外设，很可能不需要平台解初始化函数，这时，只需要将平台解初始化函数指针成员赋值为 NULL 即可。

综上，以 GPIO 为例，讲述了设备信息结构体中的 5 个部分，这些均只需要了解即可，在查看其它外设的设备信息结构体时，只要按照这个结构，就可以很清晰的理解各个部分的用途。

实际上，设备信息结构体中所有的成员，均已提供一种默认的配置。需要用户手动配置的地方少之又少。从 GPIO 的设备配置信息可以看出，虽然设备信息包含了 9 个成员，而真正需要用户根据实际情况配置的内容，仅仅只有一个宏 **pint_count** （详见 列表 4.11 ）。

除了常见的 GPIO 设备信息中的这 5 个部分外，还可能包含一些需要简单配置的值，如 ADC 中的参考电压等，这些配置内容，从意义上很好理解，就不再赘述。

4.1.3 实例初始化函数

任何外设使用前，都需要初始化。通过前文的讲述，设备配置文件中已经定义好了设备实例和设备信息结构体。至此，只需要再调用相应的驱动提供的外设初始化函数，传入对应的设备实例地址和设备信息的地址，即可完成该外设的初始化。

以 GPIO 为例，GPIO 的驱动初始化函数在 GPIO 驱动头文件 {SDK}**\ametal\soc\zlg\drivers\include\gpio\am_zlg118_gpio.h** 中声明。详见 列表 4.19 。

列表 4.19 GPIO 初始化函数

```
/**
 * \brief GPIO 初始化
 *
 * \param[in] p_dev    : 指向 GPIO 设备的指针
 * \param[in] p_devinfo : 指向 GPIO 设备信息的指针
 *
 * \retval AM_OK : 操作成功
 */
int am_zlg118_gpio_init (am_zlg118_gpio_dev_t           *p_dev,
                         const am_zlg118_gpio_devinfo_t *p_devinfo);
```

因此，要完成 GPIO 的初始化，只需要调用一下该函数即可，详见 列表 4.20 。

列表 4.20 完成 GPIO 初始化

```
1  am_zlg118_gpio_init(&__g_zlg118_gpio_dev, &__g_zlg118_gpio_devinfo);
```

__g_zlg118_gpio_dev 和 **__g_zlg118_gpio_devinfo** 分别为前面在设备配置文件中定义的设备实例和设备信息。

可见，该初始化动作行为很单一，仅仅是调用一下外设初始化函数，并传递已经定义好的设备实例地址和设备信息地址。

为了进一步减少用户的工作，设备配置文件中，将该初始化动作封装为一个函数，该函数即为实例初始化函数，用于初始化一个外设。

以 GPIO 为例，实例初始化函数定义在 {HWCONFIG}**am_hwconf_zlg118_gpio.c** 文件中，详见 列表 4.21 。

列表 4.21 GPIO 实例初始化函数

```
1  /**
2   * \brief GPIO 初始化
3   *
4   * \param[in] p_dev     : 指向 GPIO 设备的指针
5   * \param[in] p_devinfo : 指向 GPIO 设备信息的指针
6   *
7   * \retval AM_OK : 操作成功
8   */
9  int am_zlg118_gpio_init (am_zlg118_gpio_dev_t           *p_dev,
10                           const am_szlg1181_gpio_devinfo_t *p_devinfo);
```

这样，要初始化一个外设，用户只需要调用对应的实例初始化函数即可。实例初始化函数无任何参数，使用起来非常方便。

关于实例初始化函数的返回值，往往与对应的驱动初始化函数返回值一致。根据驱动初始化函数的不同，可能有三种不同的返回值。

(1) 返回值为 int 类型

一些资源全局统一管理的设备，返回值就是一个 int 值。**AM_OK** 即表示初始化成功；其它值表明初始化失败。

(2) 返回值为标准服务句柄

绝大部分外设驱动初始化函数均是返回一个标准的服务句柄（handle），以提供标准服务。值为 NULL 表明初始化失败；其它值表明初始化成功。若初始化成功，则可以使用获取到的 handle 作为标准接口层相关函数的参数，操作对应的外设。

(3) 返回值为驱动自定义服务句柄

一些较为特殊的外设，功能还没有被标准接口层标准化。此时，为了方便用户使用一些特殊功能，相应驱动初始化函数就直接返回一个驱动自定义的服务句柄（handle），值为 NULL 表明初始化失败；其它值表明初始化成功。若初始化成功，则可以使用该 handle 作为该外设驱动提供的相关服务函数的参数，用来使用一些标准接口未抽象的功能或该外设的一些较为特殊的功能。特别地，如果一个外设在提供特殊功能的同时，还可以提供标准服务，那么该外设对应的驱动还会提供一个标准服务 handle 获取函数，通过自定义服务句柄获取到标准服务句柄。

4.1.4 实例解初始化函数

每个外设驱动都提供了对应的驱动解初始化函数，以便当应用不再使用某个外设时，释放掉相关资源。以 GPIO 为例，GPIO 的驱动解初始化函数在 GPIO 驱动头文件 {SDK}**ametal\soc\zlg\drivers\include\gpio\am_zlg118_gpio.h** 中声明。详见 列表 4.22 。

列表 4.22 GPIO 解初始化函数

```
/**
 * \brief GPIO 解初始化
 *
 * \param[in] p_dev : 指向 GPIO 设备的指针
 *
 * \return 无
 */
void am_zlg118_gpio_deinit (void);
```

当应用不再使用该外设时，只需要调用一下该函数即可，详见 列表 4.23 。

列表 4.23 完成 GPIO 解初始化

```
/** \brief GPIO 实例解初始化 */
void am_zlg237_gpio_inst_deinit (void)
{
    am_zlg237_gpio_deinit();
}
```

所有实例解初始化函数均无返回值。解初始化后，该外设即不再可用。如需再次使用，需要重新调用实例初始化函数。

根据设备的不同，实例解初始化函数的参数会有不同。若实例初始化函数返回值为 int 类型，则解初始化时，无需传入任何参数；若实例初始化函数返回了一个 handle，则解初始化时，应该传入通过实例初始化函数获取到的 handle 作为参数。

4.2 典型配置

在上一节中，以 GPIO 为例，详细讲解了设备配置文件的结构以及各个部分的含义。虽然设备配置文件内容较多，但是对于用户来讲，需要自行配置的项目却非常少，往往只需要配置极少的内容，然后使用设备配置文件提供的实例初始化函数即可完成一个设备的初始化。

由于所有配置文件的结构非常相似，下文就不再一一完整地列出所有外设的设备配置信息内容。仅仅将各个外设在使用过程中，实际需要用户配置的内容列出，告知用户该如何配置。

4.2.1 ADC

1. ADC 参考电压

ZLG118 有 1 个 12 位 ADC，它支持可配置的参考电压和精度、可选的硬件转换触发等功能。下面为 ADC 标准转换功能为示例，讲解 ADC 设备信息结构体 am_zlg_adc_v0_devinfo_t ，一些用户根据具体的情况可能需要配置。一般来说，仅仅需配置 ADC 的参考电压、转换精度。

ADC 参考电压由引脚 VREFP 和 VREFN 之间的电压差值决定，对于 AM118-Core，

VREFN 接地，VREFP 默认接基准源输出（3.3V）。因此，ADC 参考电压默认为 3.3V，即 3300mV。设备信息中，默认的参考电压即为 3300（单位:mV）。详见 列表 4.24 。

列表 4.24 ADC 参考电压默认配置

```
/** \brief 设备信息 */
static const am_zlg118_adc_devinfo_t __g_adc_devinfo = {
    ZLG118_ADC_BASE,                    /**< \brief ADC */
    INUM_ADC_DAC,                       /**< \brief ADC 的中断编号 */
    CLK_ADC_BGR,                        /**< \brief ADC 时钟号 */
    AMHW_ZLG118_ADC_REFVOL_AVCC,        /**< \brief 参考电压选择 */
    3300,                               /**< \brief 参考电压 (mv)*/
    0,                                  /**< \brief ADC 通道 28 内部温度传感器开启使能
                                         *          1: 开启， 0: 关闭
                                         */
    12,                                 /**< \brief 转换精度，zlg118 精度只能为 12 位 */
    &__g_adc_ioinfo_list[0],            /**< \brief 引脚信息列表 */
    __zlg118_plfm_adc_init,             /**< \brief ADC1 的平台初始化 */
    __zlg118_plfm_adc_deinit,           /**< \brief ADC1 的平台去初始化 */

};
```

如需修改，只需要将该值修改为实际的值即可。.. note

```
如果用户需要用到 ADC 内部温度传感器通道，则需要将配置文件中的 ADC 通道 28 内部温度传感器开启使能位置 1，如不使用置零。

**2. ADC 通道引脚配置 **
```

zlg118 中，ADC 最高可支持 30 个外设通道，0-26 通道对应的引脚详见 表 4.2 ，27~29 通道无对应 GPIO 引脚。

表 4.2 ADC 通道引脚号

ADC 通道	引脚号	功能号
0	PIOA_0	PIOA_0_AIN
1	PIOA_1	PIOA_1_AIN
2	PIOA_2	PIOA_2_AIN
3	PIOA_3	PIOA_3_AIN
4	PIOA_4	PIOA_4_AIN
5	PIOA_5	PIOA_5_AIN
6	PIOA_6	PIOA_6_AIN
7	PIOA_7	PIOA_7_AIN
8	PIOB_0	PIOB_0_AIN
9	PIOB_1	PIOB_1_AIN
10	PIOC_0	PIOC_0_AIN
11	PIOC_1	PIOC_1_AIN
12	PIOC_2	PIOC_2_AIN
13	PIOC_3	PIOC_3_AIN
14	PIOC_4	PIOC_4_AIN
15	PIOC_5	PIOC_5_AIN
16	PIOB_2	PIOB_2_AIN
17	PIOB_10	PIOB_10_AIN
18	PIOB_11	PIOB_11_AIN
19	PIOB_12	PIOB_12_AIN
20	PIOB_13	PIOB_13_AIN
21	PIOB_14	PIOB_14_AIN
22	PIOB_15	PIOB_15_AIN
23	PIOE_15	PIOE_15_AIN
24	PIOE_14	PIOE_14_AIN
25	PIOE_13	PIOE_13_AIN
26	PIOE_15	PIOE_15_AIN

如需使用一个通道，需要先配置一个与该通道对应的引脚。引脚的配置在平台初始化函数中完成。

为了方便用户配置，默认情况下，列出了所有外设通道 0～26 的引脚配置，该引脚配置在 ADC 用户配置文件中的 __g_adc_ioinfo_list[] 结构体中全部列举出来。当用户需要使用一个通道时，只需要简单的调用该结构体中的成员信息即可。

注解： 通常仅将使用到的通道对应的引脚配置为模拟引脚功能，如果不使用，应将该引脚初始化为默认配置以免影响其他使用到该引脚的外设正常工作。

4.2.2 AES

AES 没有自定义参数需要配置，同时，也没有外部相关的引脚。因此，该外设完全不需要用户参与配置，实际需要使用时，调用其对应的实例初始化函数即可。

4.2.3 CLK

时钟配置，主要是配置时钟源以及时钟源的倍频，分频系数。详见 列表 4.25 。

列表 4.25 时钟默认设备信息

```
/** \brief CLK 设备信息 */
static const am_zlg118_clk_devinfo_t __g_clk_devinfo =
{
    /**
     * \brief XTH 外部高速时钟晶振频率（Hz）
     */
    32000000,

    /**
     * \brief XTL 外部低速时钟晶振频率（Hz）
     */
    32768,

    /**
     * \brief RCH 内部高速时钟晶振频率（Hz）
     *
     * \note 频率选择    24M、 22.12M、16M、 8M 或 4M
     *       设置其他频率，将默认使用 4M
     */
    AMHW_ZLG118_RCH_FRE_4MHz,

    /**
     * \brief RCL 内部低速时钟晶振频率（Hz）
     *
     * \note 频率选择    38.4k  或      32.768K
     *       设置其他频率，将默认使用 38.4k
     */
    AMHW_ZLG118_RCl_FRE_38400Hz,

    /** \brief
     *    PLL 时钟源选择
     *    -# AMHW_ZLG118_PLL_INPUT_FRE_SRC_XTH_XTAL : XTH 晶振生成的时钟
     *    -# AMHW_ZLG118_PLL_INPUT_FRE_SRC_XTH_PF00 : XTH 从管脚 PF00 输入的时钟
     *    -# AMHW_ZLG118_PLL_INPUT_FRE_SRC_RCH      : RCH 时钟
     */
    AMHW_ZLG118_PLL_INPUT_FRE_SRC_RCH,

    /**
     * \brief PLL 倍频系数，允许范围 2 ~ 12
     *        PLLOUT = PLLIN * pll_mul
     */
    12,

    /** \brief
     *    系统 时钟源选择
     *    -# AMHW_ZLG118_SYSCLK_RCH : 内部高速时钟作为系统时钟
     *    -# AMHW_ZLG118_SYSCLK_XTH : 外部高速时钟作为系统时钟
     *    -# AMHW_ZLG118_SYSCLK_RCL : 内部低速时钟作为系统时钟
     *    -# AMHW_ZLG118_SYSCLK_XTL :   外部低速时钟作为系统时钟
     *    -# AMHW_ZLG118_SYSCLK_PLL :  内部 PLL 作为系统时钟
     */
```

```
    AMHW_ZLG118_SYSCLK_PLL,

    /**
     * \brief HCLK 分频系数，HCLK = SYSCLK / (2 ^ hclk_div)
     */
    0,

    /**
     * \brief PCLK 分频系数，PCLK = HCLK / (2 ^ pclk_div)
     */
    1,

    ...
};
```

可见，由于默认配置中，主时钟源选择 PLL，PLL 内部 PLL 作为系统时钟。PLL 倍频系数为 4，因此系统时钟频率为 48MHz。

4.2.4 CRC

CRC 没有自定义参数需要配置，同时，也没有外部相关的引脚。因此，该外设完全不需要用户参与配置，实际需要使用时，调用其对应的实例初始化函数即可。

4.2.5 DAC

DAC 支持 8/12 位它支持可配置的参考电压和精度、参考电压源等功能。其用户配置信息，详见 列表 4.26 。

列表 4.26 DAC 设备信息

```
/** \brief DAC0 设备信息 */
static const am_zlg118_dac_devinfo_t __g_dac0_devinfo =
{
    ZLG118_DAC_BASE,                    /**< \brief 指向 DAC 寄存器块的指针 */

    12,                                 /**< \brief DAC 转换精度 */

    /**< \brief DAC 参考电压源
     *
     *  \note: 参考 amhw_zlg118_dac.h 文件中宏定义：DAC 0 通道参考电压选择掩码
     *
     */
    AMHW_ZLG118_DAC_CHAN_MASK_EXTER_REF,

    3300,                               /**< \brief DAC 参考电压，单位：mv*/

    AM_ZLG118_DAC_ALIGN_WAY_12_RIGHT,   /**< \brief DAC 数据对齐方式 */

    __zlg118_plfm_dac0_init,            /**< \brief DAC 平台初始化函数 */

    __zlg118_plfm_dac0_deinit,          /**< \brief DAC 平台解初始化函数  */
};
```

用户可以根据实际情况修改转换精度、参考电压源值即可。其他值可以使用默认值。

注解： 这里默认接外部参考电压源 3.3V，用户需要将外部参考电压接入 DAC 参考电压输入端。如用户修改为其他参考源需要修改正确的参考电压值。当用户选择为内部 1.5V 或者

2.5V 时需要使能 BGR。

4.2.6 DMA

DMA 没有自定义参数需要配置，同时，也没有外部相关的引脚。因此，该外设完全不需要用户参与配置，实际需要使用时，调用其对应的实例初始化函数即可。

4.2.7 GPIO

没有自定义参数需要配置，因此，该外设完全不需要用户参与配置，实际需要使用时，调用其对应的实例初始化函数即可。

4.2.8 I²C

平台共计有 2 个 I²C 总线接口，定义为 I²C1、I²C2。以 I²C0 为例，讲述其配置内容。一般地，只需要配置 I²C 总线速率、超时时间和对应的 I²C 引脚即可。

1. I²C 总线速率

I²C 总线速率由设备配置文件 {HWCONFIG}**\am_hwconf_zlg118_i2c.c** 中的 I2C 初始化设备结构体 **__g_i2c1_devinfo** 第四个参数进行配置，详见 列表 4.27 。默认为标准 I²C 速率，即 50KHz，如需修改为其它频率，直接修改其宏定义值。

列表 4.27 I²C0 速率配置

```
/**
 * \brief I2C1 设备信息
 */
static const am_stm32g071_i2c_devinfo_t __g_i2c1_devinfo = {

    ...
    __BUS_SPEED_I2C1,                          /**< \brief I2C 速率 */
    ...
};
```

2. 超时时间

由于 I²C 总线的特殊性，I²C 总线可能由于某种异常情况进入“死机”状态，为了避免该现象，I²C 驱动可以由用户提供一个超时时间，若 I²C 总线无任何响应的持续时间超过了超时时间，则 I²C 自动复位内部逻辑，以恢复正常状态。

超时时间的设置在 {HWCONFIG}**\am_hwconf_zlg118_i2c.c** 设备信息中设置，详见 列表 4.28 。

列表 4.28 I²C1 设备信息——超时时间配置

```
/**
 * \brief I2C1 设备信息
 */
static const am_stm32g071_i2c_devinfo_t __g_i2c1_devinfo = {

    ...
    20,                          /* 超时时间 */
    ...
};
```

默认值为 20，即超时时间为 20ms。若有需要，可以将该值修改为其它值。例如：将其

修改为 5，表示将超时时间设置为 5ms。

3. I²C 引脚

每个 I²C 都需要配置相应的引脚，包括时钟线 SCL 和数据线 SDA。I²C 引脚在平台初始化函数中完成。以 I²C1 为例，详见 列表 4.29 。

列表 4.29 I²C1 平台初始化函数——引脚配置

```
/**
 * \brief I2C1 平台初始化函数
 */
am_local void __zlg118_i2c1_plfm_init (void)
{
    am_gpio_pin_cfg(PIOA_11, PIOA_11_I2C1_SCL | PIOA_11_OUT_OD);
    am_gpio_pin_cfg(PIOA_12, PIOA_12_I2C1_SDA | PIOA_12_OUT_OD);

    am_clk_enable(AMHW_ZLG118_RCC_I2C1);
    am_zlg118_clk_reset(AMHW_ZLG118_RCC_RESET_I2C1);
}
```

可见，程序中，将 PIOA_11 配置为 I²C1 PIOA_12 配置为 I²C1 的数据线。各 I²C 可使用的引脚详见 表 4.3 。注意不能交叉使用。

表 4.3 I²C 引脚选择

I²C 总线接口	SCL 引脚	SDA 引脚
I²C1	PIOA_9	PIOA_10
I²C1	PIOB_6	PIOB_7
I²C1	PIOB_8	PIOB_9
I²C1	PIOF_0	PIOF_1
I²C2	PIOA_11	PIOA_12
I²C2	PIOB_10	PIOB_11
I²C2	PIOB_13	PIOB_14
I²C2	PIOF_6	PIOB_7

可以直接使用 am_gpio_pin_cfg() 函数将一个引脚配置为相应的 I²C 功能。

4.2.9 LCD

液晶控制器，由高度灵活的帧速率控制。支持设置多种占空比、偏置等参数。因此 LCD 用户配置文件需要对这些参数进行配置。

1. LCD 控制参数

由于用户选择使用的 LCD 可能存在差异，所以在配置 LCD 控制器的控制参数时需要用户根据具体的硬件进行选择。详见 列表 4.30 。

列表 4.30 LCD 默认设备信息

```
/** \brief LCD 设备信息 */
static const am_zlg118_lcd_devinfo_t __g_lcd_devinfo =
{
    /**< \brief 指向 LCD 寄存器块的指针 */
    ZLG118_LCD_BASE,

    /**< \brief LCD 中断编号 */
```

```
    INUM_LCD,

    /**
     * \brief LCD 偏置源
     *
     * \note : 其值参考 amhw_zlg118_lcd.h 中宏定义:Bias 电压来源选择掩码
     */
    AMHW_ZLG118_LCD_BIAS_CAP_DIV,

    /**
     * \brief LCD 占空比
     *
     * \note : 其值参考 amhw_zlg118_lcd.h 中枚举定义: LCD duty 配置
     *
     */
    AMHW_ZLG118_DUTY_1_4,

    /**
     * \brief LCD 偏置电压
     *
     * \note : 其值参考 amhw_zlg118_lcd.h 中宏定义: 偏置电压配置掩码
     */
    AMHW_ZLG118_BIAS_1_3,

    /**
     * \brief LCD 电压泵时钟频率
     *
     * \note : 其值参考 amhw_zlg118_lcd.h 中宏定义: 电压泵时钟频率选择掩码
     */
    AMHW_ZLG118_CPCLK_2K_HZ,

    /**
     * \brief LCD 扫描频率
     *
     * \note : 其值参考 amhw_zlg118_lcd.h 中宏定义: LCD 扫描频率选择掩码
     */
    AMHW_ZLG118_LCDCLK_128_HZ,

    /**
     * \brief 时钟源     (XTL RCL)
     *
     */
    AMHW_ZLG118_LCD_CLK_SRC_RCL,

    /**< \brief LCD 平台初始化函数 */
    __zlg118_plfm_lcd_init,

    /**< \brief LCD 平台解初始化函数  */
    __zlg118_plfm_lcd_deinit,
};
```

结构体前两个成员分别为控制器基地址和 LCD 中断号这个不需要用户去进行设置，其他成员例如：占空比、时钟频率、偏置电压等需要用户根据使用的 LCD 屏去进行配置。

时钟源，LCD 控制器可由两种时钟源提供时钟频率默认是不需要用户进行配置可以直接使用默认配置。如需要修改直接修改结构体参数即可。

2. 初始化配置

LCD 需要通过配置与控制器连接的相应引脚才能使用，其配置过程在平台初始化函数中完成，详见 列表 4.31 。

列表 4.31 LCD 引脚初始化

```
/**
 * \brief LCD 平台初始化
 */
void __zlg118_plfm_lcd_init (void)
{
    /* 初始化 LCD 外部 GPIO 引脚 */
    am_gpio_pin_cfg(PIOA_9,  PIOA_9_AIN);  /**< \brief COM0 */
    am_gpio_pin_cfg(PIOA_10, PIOA_10_AIN); /**< \brief COM1 */
    am_gpio_pin_cfg(PIOA_11, PIOA_11_AIN); /**< \brief COM2 */
    am_gpio_pin_cfg(PIOA_12, PIOA_12_AIN); /**< \brief COM3 */

    am_gpio_pin_cfg(PIOA_8,  PIOA_8_AIN);  /**< \brief SEG0 */
    am_gpio_pin_cfg(PIOC_9,  PIOC_9_AIN);  /**< \brief SEG1 */
    am_gpio_pin_cfg(PIOC_8,  PIOC_8_AIN);  /**< \brief SEG2 */
    am_gpio_pin_cfg(PIOC_7,  PIOC_7_AIN);  /**< \brief SEG3 */
    am_gpio_pin_cfg(PIOC_6,  PIOC_6_AIN);  /**< \brief SEG4 */
    am_gpio_pin_cfg(PIOB_15, PIOB_15_AIN); /**< \brief SEG5 */
    am_gpio_pin_cfg(PIOB_14, PIOB_14_AIN); /**< \brief SEG6 */
    am_gpio_pin_cfg(PIOB_13, PIOB_13_AIN); /**< \brief SEG7 */
    am_gpio_pin_cfg(PIOB_3,  PIOB_3_AIN);  /**< \brief VLCDH */
    am_gpio_pin_cfg(PIOB_4,  PIOB_4_AIN);  /**< \brief VLCD3 */
    am_gpio_pin_cfg(PIOB_5,  PIOB_5_AIN);  /**< \brief VLCD2 */
    am_gpio_pin_cfg(PIOB_6,  PIOB_6_AIN);  /**< \brief VLCD1 */

}
```

因为 LCD 控制器需要对 LCD 的 COM、SEG1 端口进行控制所以在初始化时需要对引脚配置为模拟端口。

这里没有初始化时钟，与其他配置文件不同。因为需要配置时钟源所以在程序设计时，默认的时钟初始化放在初始化函数中，所以不需要用户操心。

4.2.10 LVD

1. LVD 控制参数

ZLG118 提供了 LVD 低压检测模块可以用于监测芯片管脚电压，其特性有：多路监测源、16 阶阈值电压、8 种触发条件、2 种触发结果等等。所以用户需要对这些参数进行配置。其用户配置文件详见 列表 4.32 。

列表 4.32 LVD 配置

```
/** \brief LVD 设备信息 */
static const am_zlg118_lvd_devinfo_t __g_lvd_devinfo =
{
    /**< \brief 指向 LVD 寄存器块的指针 */
    ZLG118_LVD_BASE,

    /**< \brief LVD 中断编号 */
    INUM_LVD,

    /**
     * \brief LVD 触发条件
     *
     * \note 八种触发条件，如需要选择多个触发条件，
     *       例：AMHW_ZLG118_LVD_FTEN | AMHW_ZLG118_LVD_RTEN。
     */
    AMHW_ZLG118_LVD_FTEN,

```

```
    /**
     * \brief LVD 数字滤波时间，用户配置值参考枚举定义：LVD 数字滤波时间
     */
    AMHW_ZLG118_LVD_DEB_TIME_7P2_MS,

    /**
     * \brief LVD 阈值电压，用户配置值参考枚举定义：LVD 阈值电压
     */
    AMHW_ZLG118_LVD_VIDS_2P5_V,

    /**
     * \brief LVD 监测来源，用户配置值参考宏定义：LVD 监测来源选择掩码
     */
    ANHW_ZLG118_LVD_SRC_PB07,

    /**
     * \brief LVD 触发动作，用户配置值参考宏定义：LVD 触发动作选择掩码
     *
     * \note 此配置位只能选择为中断或者复位触发动作
     */
    ANHW_ZLG118_LVD_TRI_ACT_NVIC_INT,

    /**< \brief LVD 平台初始化函数 */
    __zlg118_plfm_lvd_init,

    /**< \brief LVD 平台解初始化函数  */
    __zlg118_plfm_lvd_deinit,
};
```

LVD 用于监测电压输入源低压与阈值电压的大小，所以首先用户需要配置 LVD 监测来源然后选择阈值电压。LVD 数字滤波时间可以采用默认配置即可，最后需要设置 LVD 的动作方式。LVD 提供了两种：复位和中断，可以根据程序设计具体情况进行设置，这里需要用户参考配置触发条件。

2. 初始化配置

平台初始化函数需要对模块时钟和输入源引脚、输出引脚进行配置。详见 列表 4.33 。

列表 4.33 LVD 平台初始化

```
/**
 * \brief LVD 平台初始化
 */
void __zlg118_plfm_lvd_init (void)
{
    /* 开启 LVD 时钟 */
    am_clk_enable (CLK_VC_LVD);

    /* PB07 LVD 监测输入端 */
    am_gpio_pin_cfg(PIOB_7, PIOB_7_AIN);

    /* PA04 LVD 输出端 */
    am_gpio_pin_cfg(PIOA_4, PIOA_4_LVD_OUT | PIOA_4_OUT_PP);
}
```

LVD 模块时钟与 VC 共用，后面就不再赘述。当用户选择监测输入端为引脚输入时需要在平台初始化函数中将相应引脚配置为模拟端口。输出端默认配置为以上代码形式不需要用户修改。

当用户选择其他外设的输入、输出端或者其他输入源作为 LVD 输入源时，需要在平台

初始化使能其时钟和其他相关配置或者直接实例初始化该模块。

4.2.11 OPA

一般采用默认配置即可，如需修改其用户参数参考\soc\drivers\include\opa\hw\amhw_zlg118_opa.h文件中的参数定义。

4.2.12 PCNT

PCNT 没有自定义参数需要配置，只是在平台初始化时需要根据用户自己选择输入引脚然后进行配置，配置信息参考默认引脚配置。

4.2.13 RTC

1. RTC 控制参数

RTC 可选择输出一般精度，较高和 3 种 1Hz 时钟。当误差补偿功能时钟。当误差补偿功能有效时输出较高精度的 1Hz 时钟；当使用不同频率的时钟；当使用不同频率的 PCLK 时输出高精度的 1Hz 时钟。所以需要用户配置有关补偿时钟的参数。详见 列表 4.34 。

列表 4.34 RTC 配置

```
/** \brief RTC 设备信息 */
const struct am_zlg118_rtc_devinfo __g_rtc_devinfo = {

    /** \brief RTC 设备基地址 */
    ZLG118_RTC_BASE,

    /** \brief RTC 设备时钟源 */
    AMHW_ZLG118_RTC_CLK_SRC_XTL_32768Hz,

    /**< \brief RTC 中断号 */
    INUM_RTC,

    {
        AM_TRUE,                        /* 是否使能 1Hz 输出功能
                                         * AM_TRUE :  使能 （请阅读下面两项配置的注释）
                                         * AM_FALSE:  禁能（下面两项配置无效）
                                         */
        -92.554,                        /* 误差补偿值（-274.6 ~ 212.6）ppm，范围外代表关闭 */
        AM_TRUE,                        /* 在 ppm 处于范围内时（即开启补偿的情况下），
                                         * 选择是否开启高速时钟作为补偿时钟
                                         *
                                         * AM_TRUE :  开启高速时钟作为补偿时钟
                                         * AM_FALSE:  关闭高速时钟作为补偿时钟
                                         */
        AMHW_ZLG118_RTCCLK_ADJUST_24M,/* 高速时钟补偿时钟选择 ，和 PCLK 频率保持一致相同  */
    },

    /** \brief 平台初始化函数 */
    __zlg118_plfm_rtc_init,

    /** \brief 平台去初始化函数 */
    __zlg118_plfm_rtc_deinit
};
```

当 1HZ 输出功能有效时，其后面三个参数有效，误差补偿值也必须在范围内。高速时钟补偿时钟选择，和 PCLK 频率保持一致相同。

2. 初始化配置

RTC 没有独立的时钟，所以在平台初始化时需要初始化 RTC 时钟。如选择了 1HZ 输出功能有效，还要初始化相应引脚。详见 列表 4.35 。

列表 4.35 RTC 平台初始化函数

```
/** \brief RTC 平台初始化 */
void __zlg118_plfm_rtc_init()
{
    am_clk_enable(CLK_RTC);

    am_gpio_pin_cfg(PIOB_14, PIOB_14_RTC_1HZ | PIOB_14_OUT_PP);
}
```

RTC 1HZ 输出可以有多个选择，各个输出引脚及配置详见 `rtc_table1` 。

表 4.4 RTC 1HZ 输出引脚选择

RTC 1HZ 输出引脚	SCL 引脚配置
PIOB_14	PIOB_14_GPIO | PIOB_14_INPUT_PU
PIOA_13	PIOA_13_GPIO | PIOA_13_INPUT_PU
PIOC_13	PIOC_13_GPIO | PIOC_13_INPUT_PU

4.2.14 TIM

ZLG237 有 3 个 16 位通用定时器、3 个高级定时器。可以实现定时、捕获、和 PWM 输出功能。用户可以直接使用 AMetal 抽象好的 PWM、CAP、Timing 标准接口服务，也可以直接调用驱动层提供的相关接口操作 Timer 的一些功能。由于 Timer 抽象出来的标准服务功能各不相同，在初始化时，就需要确定将 Timer 用于何种功能，不同功能对应的设备信息存在不同，这就使得 Timer 部分对应了几套设备配置文件，使用 Timer 的何种功能，就使用对应的配置文件。下面以 TIM1 为例，讲解其配置内容。

1. 使用标准捕获（CAP）服务

在该模式下，对应的配置文件为 {HWCONFIG}**\am_hwconf_stm32g071_tim_cap.c** ，TIM 最多可以支持 2 路捕获通道，实际使用到的通道数目可以在配置文件中的设备信息中修改。详见 列表 4.36 。

列表 4.36 TIM1 CAP 设备信息

```
/** \brief TIM1 用于捕获功能的设备信息 */
const am_zlg118_tim_cap_devinfo_t  __g_tim1_cap_devinfo = {
    ZLG118_TIM1_BASE,                   /**< \brief TIM1 寄存器块的基地址 */
    INUM_TIM1,                          /**< \brief TIM1 中断编号 */
    AMHW_ZLG118_TIM_CLK_DIV1,           /**< \brief 时钟分频系数 */
    2,                                  /**< \brief 2 个捕获通道 */
    AMHW_ZLG118_TIM_TYPE_TIM1,          /**< \brief 定时器选择 */

    &__g_tim1_cap_ioinfo_list[0],
    __zlg118_plfm_tim1_cap_init,        /**< \brief 平台初始化函数 */
    __zlg118_plfm_tim1_cap_deinit       /**< \brief 平台解初始化函数 */
};
```

使用捕获功能时，每个捕获通道都需要设置一个对应的引脚，相关引脚信息由设备信息

文件中的 **__g_tim1_cap_ioinfo_list** 数组定义，详见 列表 4.37 。

列表 4.37 TIM1 CAP 各捕获通道相关引脚设置

```
/** \brief TIM1 用于捕获功能的引脚配置信息列表 */
am_zlg118_tim_cap_ioinfo_t __g_tim1_cap_ioinfo_list[] = {
    /**< \brief 通道 0 */
    {
        AM_ZLG118_TIM_CAP_CH0A,
        PIOA_0,
        PIOA_0_TIM1_CHA  | PIOA_0_INPUT_FLOAT,
        PIOA_0_GPIO  | PIOA_0_INPUT_PU
    },

    /**< \brief 通道 1 */
    {
        AM_ZLG118_TIM_CAP_CH0B,
        PIOA_1,
        PIOA_1_TIM1_CHB  | PIOA_1_INPUT_FLOAT,
        PIOA_1_GPIO  | PIOA_1_INPUT_PU
    },
};
```

每个数组元素对应了一个捕获通道，0 号元素对应通道 0，1 号元素对应通道 1，以此类推。数组元素的类型为 **am_zlg118_tim_cap_ioinfo_t** ，该类型在对应驱动头文件 {SDK}**ametal\user_config\am_hwconf_usrcfg\am_hwconf_zlg118_tim_cap.c** 中定义，详见 列表 4.38 。

列表 4.38 TIM_CAP 通道引脚信息结构体类型

```
/**
 * \brief TIM 捕获功能相关的 GPIO 信息
 */
typedef struct am_zlg118_tim_cap_ioinfo {
    int8_t   channel;        /**< \brief CAP 所使用的通道标识符 */
    uint32_t gpio;           /**< \brief 对应的 GPIO 管脚 */
    uint32_t func;           /**< \brief 为捕获功能时的 GPIO 功能设置 */
    uint32_t dfunc;          /**< \brief 禁能管脚捕获功能时的默认 GPIO 功能设置 */
} am_zlg118_tim_cap_ioinfo_t;
```

2. 使用标准 PWM 服务

在该模式下，设置与上述基本相同，此处不再赘述，此处要注意的是 PWM 的模式与输出电平，用户可以根据自己的需求设置。对应的配置文件为 {HWCONFIG} am_hwconf_zlg118_tim_pwm.c ，TIM1 用于 PWM 的设备信息详见 列表 4.39 。

列表 4.39 TIM1 PWM 设备信息

```
/** \brief TIM1 用于 PWM 设备信息 */
const am_zlg118_tim_pwm_devinfo_t  __g_tim1_pwm_devinfo = {
    ZLG118_TIM1_BASE,                           /**< \brief TIM1 寄存器块的基地址
↪*/
    AM_NELEMENTS(__g_tim1_pwm_chaninfo_list),   /**< \brief 配置输出通道个数 */
    0,                                          /**< \brief 互补 PWM 选择
                                                 *          1: 互补 PWM
                                                 *          0: 独立 PWM
                                                 */
    AM_ZLG118_TIM_PWM_OCPOLARITY_HIGH,          /**< \brief 脉宽极性 */
    &__g_tim1_pwm_chaninfo_list[0],             /**< \brief 通道配置信息列表
↪   */
    AMHW_ZLG118_TIM_TYPE_TIM1,                  /**< \brief 定时器类型 */
    __zlg118_plfm_tim1_pwm_init,                /**< \brief 平台初始化函数 */
```

```
    __zlg118_plfm_tim1_pwm_deinit                /**< \brief 平台解初始化函数 */
};
```

使用 PWM 功能时，每个 PWM 输出通道都需要设置一个对应的引脚，相关引脚信息由设备信息文件中的 **__g_tim1_pwm_chaninfo_list** 数组定义，详见 列表 4.40 。

列表 4.40 TIM1 PWM 各输出通道相关引脚设置

```
/** \brief TIM1 用于 PWM 功能的引脚配置信息列表 */
am_zlg118_tim_pwm_chaninfo_t __g_tim1_pwm_chaninfo_list[] = {

    /** \brief 通道 0 引脚配置 */
    {
        AM_ZLG118_TIM_PWM_CH0A,
        PIOA_0,
        PIOA_0_TIM1_CHA | PIOA_0_OUT_PP,
        PIOA_0_GPIO  | PIOA_0_INPUT_PU
    },

    /** \brief 通道 1 引脚配置 */
    {
        AM_ZLG118_TIM_PWM_CH0B,
        PIOA_1,
        PIOA_1_TIM1_CHB | PIOA_1_OUT_PP,
        PIOA_1_GPIO  | PIOA_1_INPUT_PU
    },
};
```

每个数组元素对应了一个 PWM 输出通道，0 号元素对应通道 0，1 号元素对应通道 1，以此类推。数组元素的类型为 **__g_tim1_pwm_chaninfo_list** ，该类型在对应驱动头文件 {SDK}**\ametal\user_config\am_hwconf_usrcfg\am_hwconf_zlg118_tim_pwm.c** 文件中定义。

3. 使用标准定时器服务

在该模式下，对应的配置文件为 {HWCONFIG}**\am_hwconf_zlg118_tim_timing.c** ，详见 列表 4.41 。

列表 4.41 TIM1 Timing 设备配置信息

```
/** \brief TIM1 设备信息 */
const am_zlg118_tim_timing_devinfo_t  __g_tim1_timing_devinfo = {
    ZLG118_TIM1_BASE,                  /**< \brief TIM1 寄存器块的基地址 */
    INUM_TIM1,                         /**< \brief TIM1 中断编号 */
    AMHW_ZLG118_TIM_TYPE_TIM1,         /**< \brief 定时器类型 */
    AMHW_ZLG118_TIM_MODE0_COUNTER_16,  /**< \brief 16 位重载计数 */
    AMHW_ZLG118_TIM_GATE_DISABLE,      /**< \brief 门控状态（默认关闭） */
    AM_TRUE,                           /**< \brief 门控信号为真（1）有效 */
    AMHW_ZLG118_TIM_CLK_SRC_TCLK,      /**< \brief 计数时钟选择（默认内部） */
    __zlg118_plfm_tim1_timing_init,    /**< \brief 平台初始化函数 */
    __zlg118_plfm_tim1_timing_deinit   /**< \brief 平台解析初始化函数 */
};
```

定时器类型由 **__g_tim1_timing_devinfo** 中第四个参数确定，定时器有四种类型，每种类型定时器具体功能有所不同。定时器类型在对应的驱动头文件 {SDK}**\ametal\soc\zlg\drivers\include\tim\hw\amhw_zlg118_tim.h** 文件中定义。详见 列表 4.42 。

列表 4.42 TIM Timing 类型

```
/**
 * \brief 定时器类型
 */
typedef enum amhw_zlg118_tim_type {
    AMHW_ZLG118_TIM_TYPE_TIM0 = 0,  /**< \brief TIM0 */
    AMHW_ZLG118_TIM_TYPE_TIM1,      /**< \brief TIM1 */
    AMHW_ZLG118_TIM_TYPE_TIM2,      /**< \brief TIM2 */
    AMHW_ZLG118_TIM_TYPE_TIM3,      /**< \brief TIM3 */
} amhw_zlg118_tim_type_t;
```

4.2.15 LPTIM

ZLG118 系统提供两个低功耗定时器。在 AMetal 软件包提供的驱动中实现了 LPTIM 功能。一般来讲配置文件用户只需要选择其工作方式，详见 列表 4.43 。

列表 4.43 LPTIM 设备信息

```
/** \brief LPTIM0 设备信息 */
const am_zlg118_lptim_timing_devinfo_t  __g_lptim0_timing_devinfo = {
    ZLG118_LPTIM0_BASE,                 /**< \brief LPTIM0 寄存器块的基地址 */
    INUM_LPTIM0_1,                      /**< \brief LPTIM0 中断编号 */
    AMHW_ZLG118_LPTIM_CLK_SRC_PCLK,     /**< \brief 时钟源选择 */
    AMHW_ZLG118_LPTIM_FUNCTION_TIMER,   /**< \brief 定时器/计数器功能选择
                                         *
                                         *（计数器功能需要初始化相关 ETR 引脚）
                                         *
                                         */
    AMHW_ZLG118_LPTIM_MODE_RELOAD,      /**< \brief 自动重载模式 */
    AM_FALSE,                           /**< \brief 门控状态（默认关闭） */
    AM_TRUE,                            /**< \brief 门控信号为真（1）有效 */
    AM_TRUE,                            /**< \brief TOG、TOGN 输出使能
                                         *
                                         *    AM_FALSE :  TOG,TOGN 同时输出 0
                                         *    AM_TRUE  :  TOG,TOGN 输出相位相反的
信号
                                         *
                                         */

    __zlg118_plfm_lptim0_timing_init,   /**< \brief 平台初始化函数 */
    __zlg118_plfm_lptim0_timing_deinit  /**< \brief 平台解析初始化函数 */
};
```

用户通过选择第五个参数，来选择 LPTIM 工作方式。

在使用 LPTIM 之前需要配置 GATE 门控引脚、ETR 引脚等，具体配置在 LPTIM 初始化函数中实现。详见 列表 4.44 。

列表 4.44 LPTIM 引脚配置

```
/** \brief LPTIM0 平台初始化 */
void __zlg118_plfm_lptim0_timing_init (void)
{
    /* 配置 GATE 引脚，以实现门控功能 */
    am_gpio_pin_cfg(PIOB_3, PIOB_3_INPUT_FLOAT | PIOB_3_LPTIM0_GATE);

    /* 配置 ETR 引脚，以外部时钟作为计数时钟功能 */
    am_gpio_pin_cfg(PIOB_4, PIOB_4_INPUT_FLOAT | PIOB_4_LPTIM0_ETR);

    /* 配置 TOG、TOGN 引脚，输出相关电平 */
    am_gpio_pin_cfg(PIOC_1, PIOC_1_OUT_PP | PIOC_1_LPTIM0_TOG);
```

```
    am_gpio_pin_cfg(PIOC_2, PIOC_2_OUT_PP | PIOC_2_LPTIM0_TOGN);

    am_clk_enable(CLK_LPTIM0);
    am_zlg118_clk_reset(CLK_LPTIM0);
}
```

4.2.16 SPI

平台有 2 个 SPI 总线接口，定义为 SPI1、SPI2。SPI 可以选择中断方式、DMA 传输方式和轮询方式，这三种种方式的配置略有不同，同时 SPI 可以配置成主机或者从机模式。因此，平台分别提供了不同方式的配置文件。

1. 主机中断方式

以 SPI1 为例，如果使用 SPI 的中断方式，需要在 {HWCONFIG}\am_hwconf_zlg118_spi_int.c 文件中进行 SPI1 相关引脚的设置，需要设置的引脚仅有 SCK、MOSI 和 MISO，片选引脚无需设置，因为在使用 SPI 标准接口层函数（参见：ametalinterfaceam_spi.h）时，片选引脚可以作为参数任意设置。SPI 相关引脚的设置详见 列表 4.45 。

列表 4.45 SPI1 平台初始化函数

```
/** \brief SPI1 平台初始化 */
static void __zlg118_plfm_spi1_int_init (void)
{
    am_gpio_pin_cfg(PIOB_10, PIOB_10_SPI1_SCK  | PIOB_10_OUT_PP);
    am_gpio_pin_cfg(PIOB_15, PIOB_15_SPI1_MOSI | PIOB_15_OUT_PP);
    am_gpio_pin_cfg(PIOC_2,  PIOC_2_SPI1_MISO  | PIOC_2_INPUT_PU);

    am_clk_enable(CLK_SPI1);
}
```

如果需要设置为别的引脚，请参考以上引脚初始化。

设备信息结构体主要配置 MOSI 引脚，详见 列表 4.46 。

列表 4.46 SPI1 主机中断传输方式设备信息结构体定义

```
/** \brief SPI1 设备信息 */
const  struct am_zlg118_spi_int_devinfo  __g_spi1_int_devinfo = {
    ZLG118_SPI1_BASE,                        /**< \brief SPI1 寄存器指针 */
    CLK_SPI1,                                /**< \brief 时钟 ID 号 */
    INUM_SPI1,                               /**< \brief SPI1 中断号 */
    PIOB_15_SPI1_MOSI | PIOB_15_OUT_PP,      /**< \brief SPI1 配置标识 */
    PIOB_15,                                 /**< \brief MOSI 引脚号 */
    __zlg118_plfm_spi1_int_init,             /**< \brief SPI1 平台初始化函数 */
    __zlg118_plfm_spi1_int_deinit            /**< \brief SPI1 平台解初始化函数 */
};
```

2. 主机 DMA 传输方式

如果需要使用 SPI 的 DMA 方式，需要在 {HWCONFIG}am_hwconf_zlg118_spi_dma.c 文件中进行 SPI 引脚的配置，引脚配置方法与中断方式配置引脚的方法完全一样，在此不再赘述。

而设备信息有不同，需要增加配置 DMA 的发送和接收通道，DMA 的通道信息可参考 ametalsoczlgzlg237zlg118_dma_chan.h ，设备配置详见 列表 4.47 。

列表 4.47 SPI1 主机 DMA 传输方式设备信息结构体定义

```
/** \brief SPI1 设备信息 */
static const  struct am_zlg118_spi_dma_devinfo  __g_spi1_dma_devinfo = {
    ZLG118_SPI1_BASE,                          /**< \brief SPI1 寄存器指针 */
    CLK_SPI1,                                  /**< \brief 时钟 ID 号 */
    INUM_SPI1,                                 /**< \brief SPI1 中断号 */
    DMA_CHAN_1,
    DMA_CHAN_2,
    __zlg118_plfm_spi1_dma_init,               /**< \brief SPI1 平台初始化函数 */
    __zlg118_plfm_spi1_dma_deinit              /**< \brief SPI1 平台解初始化函数 */
};
```

3. SPI 主机轮询传输方式

如果需要使用 SPI 的 poll 方式，需要在 {HWCONFIG}am_hwconf_zlg118_spi_poll.c 文件中进行 SPI 引脚的配置，引脚配置方法与中断方式配置引脚的方法完全一样，设备信息配置也和中断方式配置完全一样，在此不再赘述。设备配置详见 列表 4.48 。

列表 4.48 SPI1 设备信息配置

```
/**
 * \brief SPI1 设备信息
 */
const struct am_stm32g071_spi_int_devinfo  __g_spi1_int_devinfo = {
    STM32G071_SPI1_I2S1_BASE,                  /**< \brief SPI1 寄存器指针 */
    CLK_SPI1,                                  /**< \brief 时钟 ID 号 */
    AM_STM32G071_SPI_CRC_DISABLE |             /**< \brief CRC 关闭 */
    AM_STM32G071_SPI_CLKPL,                    /**< \brief CLK 低电平主机空闲 */
    PIOB_5,                                    /**< \brief MOSI 引脚 */
    INUM_SPI1,                                 /**< \brief SPI 中断号 */
    __stm32g071_plfm_spi1_int_init,            /**< \brief SPI1 平台初始化函数 */
    __stm32g071_plfm_spi1_int_deinit           /**< \brief SPI1 平台解初始化函数 */
};
```

注意：默认是 CRC 校验关闭的。

4.2.17 UART

平台有 4 个 USART 接口，定义为 USART0、USART1、USART2、USART3。在 SDK 提供的驱动中，仅只实现了 UART 功能。对于用户来讲，一般只需要配置串口的相关引脚即可。特别地，可能需要配置串口的输入时钟频率。下面以 UART0 为例，讲解其配置内容。

注解： 串口波特率、数据位、停止位、校验位等的设置应直接使用 UART 标准接口层相关函数配置，详见 UART 标准接口文件 {SDK}**\ametal\common\interface\am_uart.h** 。

---

1. 引脚配置

USART1 相关的引脚在配置文件 {HWCONFIG}**\am_hwconf_zlg118_uart.c** 文件中配置，详见 `usart_code1` 。

列表 4.49 UART1 平台初始化函数

```
/** \brief 串口 0 平台初始化 */
static void __zlg118_plfm_uart0_init (void)
{
    am_gpio_pin_cfg(PIOA_9,  PIOA_9_UART0_TXD  | PIOA_9_OUT_PP );
```

```
    am_gpio_pin_cfg(PIOA_10, PIOA_10_UART0_RXD | PIOA_10_INPUT_FLOAT);
}
```

程序中，将 PIOA_9 配置为了串口 1 的发送引脚，PIOA_10 配置为了串口 1 的接收引脚。各个串口的发送和接收引脚对应的 GPIO 功能宏详见 表 4.5 。

表 4.5 UART 引脚功能宏定义

UART 接口	RXD 功能引脚	TXD 功能引脚
UART0	PIOA_10	PIOA_9
UART1	PIOA_3	PIOA_2
UART2	PIOC_2	PIOC_3
UART3	PIOC_6	PIOC_7

2. 串口的参数配置

设备平台初始化已经进行了相关串口参数配置，如果需要修改参数。直接调用标准接口函数设置即可。

在 USART1 设备配置信息中可以设置串口位数、奇偶校验、停止位、波特率等信息，串口默认设置为 8 位数据，无奇偶校验，1 个停止位，波特率为 115200，详见 列表 4.50 。

列表 4.50 USART 设备信息结构体类型

```
/** \brief 串口 0 设备信息 */
static const am_zlg118_uart_devinfo_t __g_uart0_devinfo = {

    ZLG118_UART0_BASE,            /**< \brief 串口 0 */
    INUM_UART0_2,                 /**< \brief 串口 0 的中断编号 */
    AMHW_ZLG118_UART_WORK_MODE_1, /**< \brief 串口工作模式 */

    AMHW_ZLG118_UART_PARITY_NO |  /**< \brief 无极性 */
    AMHW_ZLG118_UART_STOP_1_0_BIT,/**< \brief 1 个停止位 */

    115200,                       /**< \brief 设置的波特率 */

    0,                            /**< \brief 无其他中断 */

    {
        AM_FALSE,                 /**< \brief 禁能多机地址自动识别 */
        0x00,                     /**< \brief 地址 0x00 */
        0x00,                     /**< \brief 地址全部不关心 */
    },

    {
        AM_FALSE,                 /**< \brief 禁能流控 */
        0,                        /**< \brief CTS 引脚编号 */
        0,                        /**< \brief RTS 引脚编号 */
    },

    NULL,                         /**< \brief 使用 RS485 */
    __zlg118_plfm_uart0_init,     /**< \brief UART0 的平台初始化 */
    __zlg118_plfm_uart0_deinit,   /**< \brief UART0 的平台去初始化 */
};
```

4.2.18 LPUART

1. 串口的参数配置

平台有两个低功耗串口，其用户配置选项跟普通定时器无区别，在这里就不在赘述。其用户配置信息，详见 列表 4.51 。

列表 4.51 LPUART 设备信息结构体类型

```
/** \brief LPUART0 设备信息 */
static const am_zlg118_lpuart_devinfo_t __g_lpuart0_devinfo = {

    ZLG118_LPUART0_BASE,              /**< \brief LPUART0 */
    INUM_LPUART0,                     /**< \brief LPUART0 的中断编号 */
    AMHW_ZLG118_LPUART_WORK_MODE_1,   /**< \brief LPUART 工作模式 */
    AMHW_ZLG118_LPUART_SCLK_SRC_PCLK, /**< \brief 通信传输时钟  */
    AMHW_ZLG118_LPUART_PARITY_NO |    /**< \brief 无极性 */
    AMHW_ZLG118_LPUART_STOP_1_0_BIT,  /**< \brief 1 个停止位 */

    115200,                           /**< \brief 设置的波特率 */

    0,                                /**< \brief 无其他中断 */

    {
        AM_FALSE,                     /**< \brief 禁能多机地址自动识别 */
        0x00,                         /**< \brief 地址 0x00 */
        0x00,                         /**< \brief 地址全部不关心 */
    },

    {
        NULL,                         /**< \brief 硬件流控引脚初始化、解初始化函数
↪*/
        0,                            /**< \brief CTS 引脚编号 */
        0,                            /**< \brief RTS 引脚编号 */
    },

    NULL,                             /**< \brief 使用 RS485 */
    __zlg118_plfm_lpuart0_init,       /**< \brief LPUART0 的平台初始化 */
    __zlg118_plfm_lpuart0_deinit,     /**< \brief LPUART0 的平台去初始化 */
};
```

2. 引脚配置

LPUART 相关的引脚在配置文件 {HWCONFIG}**am_hwconf_zlg118_lpuart.c** 文件中配置，详见 `lpuart_code2` 。

列表 4.52 UART1 平台初始化函数

```
/** \brief LPUART0 平台初始化 */
static void __zlg118_plfm_lpuart0_init (void)
{
    am_clk_enable(CLK_LPUART0);

    am_gpio_pin_cfg(PIOB_11, PIOB_11_LPUART0_RXD | PIOB_11_INPUT_FLOAT);
    am_gpio_pin_cfg(PIOB_12, PIOB_12_LPUART0_TXD | PIOB_12_OUT_PP );
}
```

程序中，将 PIOB_12 配置为了低功耗串口 1 的发送引脚，PIOB_11 配置为了接收引脚。各个串口的发送和接收引脚对应的 GPIO 功能宏详见 表 4.6 。

表 4.6 LPUART 引脚功能宏定义

LPUART 接口	RXD 功能引脚	TXD 功能引脚
LPUART0	PIOB_12	PIOB_11
LPUART0	PIOC_4	PIOC_5
LPUART0	PIOC_11	PIOC_10
LPUART1	PIOA_1	PIOA_0
LPUART1	PIOC_10	PIOC_11
LPUART1	PIOB_2	PIOB_15

4.2.19 WDT

WDT 没有自定义参数需要配置，同时，也没有外部相关的引脚。因此，该外设完全不需要用户参与配置，实际需要使用时，调用其对应的实例初始化函数即可。

4.2.20 NVIC

在 ZLG118 中，可以产生中断的外设都有对应的中断号，每个中断号都可以对应一个中断服务的入口。外设中断号详见 列表 4.53

列表 4.53 中断号相关定义

```
/**
 * 中断号使用说明
 *
 * 某一外设的中断号独立、单一（如 INUM_xx 形式），在使用时无特别注意事项
 *
 * 某一外设的中断号不独立、多个外设使用同一中断（如 INUM_xx_xx 形式），在使用时需要注意，因为是多个外设，
 * 意味着相关控制寄存器的基地址不同，在中断连接、断开时，尽管多次调用，实际上仅一次有效。回调函数、中断服务
 * 函数和传入参数，可能生效的只有一组。
 *
 * 假如，该中断号的所有外设都使用中断，最直接的现象：
 *     1. 在多个外设调用初始化函数，只有最后一个初始化的外设中断能正常运行。
 *     2. 在多个外设调用解初始化函数，只要发生一个解初始化过程，即中断断开连接，其余所有外设全部解开中断。
 *
 *  考虑到用户使用情况的不确定性，仅在 INUM_LPTIM0_1 的中间层函数，使用多个设备结构体指针缓存 + 判断的方式解决了
 *  这一现象，可以参考 am_zlg118_lptim_timing.c 该解决方法，以解决其他中断号使用时的上述现象。
 *
 */
#define INUM_PORTA                0    /**< \brief PORTA 中断 */
#define INUM_PORTB                1    /**< \brief PORTB 中断 */
#define INUM_PORTC_E              2    /**< \brief PORTC_E 中断 */
#define INUM_PORTD_F              3    /**< \brief PORTD_F 中断 */
#define INUM_DMA                  4    /**< \brief DMA 中断 */
#define INUM_TIM3                 5    /**< \brief TIM3 中断 */
#define INUM_UART0_2              6    /**< \brief UART0_2 中断 */
#define INUM_UART1_3              7    /**< \brief UART1_3 中断 */
#define INUM_LPUART0              8    /**< \brief LPUART0 中断 */
#define INUM_LPUART1              9    /**< \brief LPUART1 中断 */
#define INUM_SPI0                10    /**< \brief SPI0 中断 */
#define INUM_SPI1                11    /**< \brief SPI1 中断 */
#define INUM_I2C0                12    /**< \brief I2C0 中断 */
#define INUM_I2C1                13    /**< \brief I2C1 中断 */
```

```
#define INUM_TIM0                14    /**< \brief TIM0 中断 */
#define INUM_TIM1                15    /**< \brief TIM1 中断 */
#define INUM_TIM2                16    /**< \brief TIM2 中断 */
#define INUM_LPTIM0_1            17    /**< \brief LPTIM0_1 中断 */
#define INUM_TIM4                18    /**< \brief TIM4 中断 */
#define INUM_TIM5                19    /**< \brief TIM5 中断 */
#define INUM_TIM6                20    /**< \brief TIM6 中断 */
#define INUM_PCA                 21    /**< \brief PCA 中断 */
#define INUM_WDT                 22    /**< \brief WDT 中断 */
#define INUM_RTC                 23    /**< \brief RTC 中断 */
#define INUM_ADC_DAC             24    /**< \brief ADC_DAC 中断 */
#define INUM_PCNT                25    /**< \brief PCNT 中断 */
#define INUM_VC0                 26    /**< \brief VC0 中断 */
#define INUM_VC1_VC2             27    /**< \brief VC12 中断 */
#define INUM_LVD                 28    /**< \brief LVD 中断 */
#define INUM_LCD                 29    /**< \brief LCD 中断 */
#define INUM_FLASH_RAM           30    /**< \brief FLASH_RAM 中断 */
#define INUM_CLKTRIM             31    /**< \brief CLKTRIM 中断 */

/** @} */

/**
 * \brief 总中断数为: (INUM_CLKTRIM - INUM_PORTA + 1),
 *
 */
#define INUM_INTERNAL_COUNT     (INUM_CLKTRIM - INUM_PORTA + 1)

/**
 * \brief 最大中断号为:  INUM_CLKTRIM
 */
#define INUM_INTERNAL_MAX        INUM_CLKTRIM

/** \brief 最小中断号: INUM_PORTA */
#define INUM_INTERNAL_MIN        INUM_PORTA
```

可以看到，总中断数目的最终值为 31。

用户还可以在设备信息中配置 STM32G071 这款芯片的内核类型为 **AM_ARM_NVIC_CORE_M0PLUS**，这款芯片使用的优先级位数有 2 位（一般来说，M0 与 M0+ 内核的芯片，NVIC 最多可有 8 位用于设置优先级，在这 8 位里面可以分成多少位设置组中断，多少位分成设置子中断。而 M0 与 M0+，NVIC 仅有 2 位用于设置优先级）。NVIC 的设备信息详见 列表 4.54 。

列表 4.54 NVIC 设备信息

```
/** \brief 设备信息 */
static const am_arm_nvic_devinfo_t __g_nvic_devinfo =
{
    {
        INUM_INTERNAL_MIN,    /* 起始中断号 */
        INUM_INTERNAL_MAX     /* 末尾中断号 */
    },

    AM_ARM_NVIC_CORE_M0PLUS, /* Cortex-M0+ 内核 */

    2,                        /* 仅有子优先级，且子优先级有 2 位 */
    0,                        /* 组中断 */

    INUM_INTERNAL_COUNT,      /* 总中断数量 */
    __nvic_isr_map,           /* 中断信息映射 */
    __ISRINFO_COUNT,          /* 中断信息数量 */
```

```
    __nvic_isr_infor,          /* 中断信息映射内存 */

    NULL,                      /* 无需平台初始化函数 */
    NULL                       /* 无需平台解初始化函数 */
};
```

4.2.21 VC

模拟比较器用于比较两个输入端电压大小，支持三种终端触发方式。一般来说对 VC 的配置，主要是对 VC 输入端的配置。详见 列表 4.55 。

列表 4.55 VC 设备信息

```
/** \brief vc 设备信息 */
static const am_zlg118_vc_devinfo_t __g_vc_devinfo =
{
    /**< \brief 指向 VC 寄存器块的指针 */
    ZLG118_VC_BASE,

    /**< \brief vc 中断编号 */
    INUM_VC0,

    /**
     * \brief VC 通道 0，用户配置值查看宏定义：模拟比较器通道选择掩码
     */
    AMHW_ZLG118_VC0,

    /**
     * \brief vc 参考 选择，用户配置值查看宏定义：VC_DIV 参考电压 Vref 选择掩码
     *
     */
    AMHW_ZLG118_VC_REF2P5_VCC,

    /**
     * \brief VC 通道 0 迟滞电压 10mv , 为 NULL 则表示没有迟滞
     *        用户配置值查看宏定义：VC0-2 迟滞电压选择掩码
     */
    AMHW_ZLG118_VC0_HYS_10_MV,

    /**
     * \brief VC 功耗选择，用户配置值查看宏定义：VC0-2 功耗选择掩码
     *
     * \note : 功耗越大，响应时间越快。当功耗选择>=10uA 需要开启 BGR，BGR 启动时间大约
20us
     */
    AMHW_ZLG118_VC0_BIAS_1_2_UA,

    /**
     * \brief VC 滤波时间，用户配置值查看枚举 amhw_zlg118_vc_deb_time:
     *
     * \note : 滤波时间的配置只有在 FLTEN=1 时才有效，如不需要滤波值为：AMHW_ZLG118_
↪DEB_TIME_NO
     */
    AMHW_ZLG118_DEB_TIME_28_US,

    /**
     * \brief vc p 端输入 选择，用户配置值查看枚举：VC0-2 “+”端输入选择
     *
     * \note : 每个通道输入端口枚举不同，需要查看相应使用通道的枚举定义
     */
    AMHW_ZLG118_VC0_P_INPUT_PC0,
```

```
47
48      /**
49       * \brief vc n 端输入 选择，用户配置值查看枚举：VC0-2 “-”端输入选择
50       *
51       * \note : 每个通道输入端口枚举不同，需要查看相应使用通道的枚举定义
52       */
53      AMHW_ZLG118_VC0_N_INPUT_PA7,
54
55      /**
56       * \brief vc 输出配置，用户配置值参考：VC0-2 输出寄存器功能使能宏
57       *
58       * \note : 不用时输出禁能，不能写为 NULL，需写：AMHW_ZLG118_VC_OUT_CFG_DISABLE
59       */
60      AMHW_ZLG118_VC_OUT_CFG_DISABLE,
61
62      /**
63       *  \brief vc 输出触发中断类型选择，用户配置参考宏定义：VC0-2 输出信号触发中断选择掩码
64       *
65       */
66      AMHW_ZLG118_VC_OUT_TRI_INT_RISING,
67
68      /**< \brief VC 平台初始化函数 */
69      __zlg118_plfm_vc_init,
70
71      /**< \brief VC 平台解初始化函数  */
72      __zlg118_plfm_vc_deinit,
73  };
```

用户需要自己选择模拟比较器的 P 端和 N 端输入，然后在平台初始化函数中对引脚进行配置，详见 列表 4.56 。

列表 4.56 VC 平台初始化函数

```
1   /**
2    * \brief VC 平台初始化
3    */
4   void __zlg118_plfm_vc_init (void)
5   {
6
7       /* 配置 PIOA_6 为 VC_OUT 功能 */
8       am_gpio_pin_cfg(PIOA_6, PIOA_6_VC0_OUT | PIOA_6_OUT_PP);
9
10      /* 配置 PIOC_0 为 VC0 P 端输入 */
11      am_gpio_pin_cfg(PIOC_0, PIOC_0_AIN);
12
13      /* 配置 PIOA_7 为 VC0 P 端输入 */
14      am_gpio_pin_cfg(PIOA_7, PIOA_7_AIN);
15
16      /* 开启 VC 时钟 */
17      am_clk_enable(CLK_VC_LVD);
18
19      /* 开启 BGR 时钟 */
20      am_clk_enable(CLK_ADC_BGR);
21  }
```

使用 VC 必须使能 BGR！

4.2.22 TRNG

TRNG 没有自定义参数需要配置，同时，也没有外部相关的引脚。因此，该外设完全不需要用户参与配置，实际需要使用时，调用其对应的实例初始化函数即可。

4.3 使用方法

使用外设资源的方法有两种，一种是使用软件包提供的驱动，一种是不使用驱动，自行使用硬件层提供的函数完成相关操作。

4.3.1 使用 AMetal 软件包提供的驱动

一般来讲，除非必要，一般都会优先选择使用经过测试验证的驱动完成相关的操作。使用外设的操作顺序一般是初始化、使用相应的接口函数操作该外设、解初始化

4.3.1.1 初始化

无论何种外设，在使用前均需初始化。所有外设的初始化操作均只需调用用户配置文件中提供的设备实例初始化函数即可。

所有外设的实例初始化函数均在 {PROJECT}**\user_config\am_zlg118_inst_init.h** 文件中声明。使用实例初始化函数前，应确保已包含 **am_zlg118_inst_init.h** 头文件。片上外设对应的设备实例初始化函数的原型详见 表 4.7 。

表 4.7 片上外设及对应的实例初始化函数

序号	外设	实例初始化函数原型
1	ADC	am_adc_handle_t am_zlg118_adc_inst_init (void);
2	AES	am_aes_handle_t am_zlg118_aes_inst_init (void);
3	NVIC	int am_zlg118_nvic_inst_init (void);
4	CLK	int am_zlg118_clk_inst_init (void);
5	CRC	am_crc_handle_t am_zlg118_crc_inst_init (void);
6	DAC	am_dac_handle_t am_zlg118_dac0_inst_init (void);
7	DMA	int am_zlg118_dma_inst_init (void);
8	GPIO	int am_zlg118_gpio_inst_init (void);
9	I^2c0	am_i2c_handle_t am_zlg118_i2c0_inst_init (void);
10	I^2c1	am_i2c_handle_t am_zlg118_i2c1_inst_init (void);
11	LCD	am_lcd_handle_t am_zlg118_lcd_inst_init (void);
12	LPTIM0	am_timer_handle_t am_zlg118_lptim0_timing_inst_init (void);
13	LPTIM1	am_timer_handle_t am_zlg118_lptim1_timing_inst_init (void);
14	LPUART0	am_uart_handle_t am_zlg118_lpuart0_inst_init (void);
15	LPUART1	am_uart_handle_t am_zlg118_lpuart1_inst_init (void);
16	LVD	am_lvd_handle_t am_zlg118_lvd_inst_init (void);
17	OPA	am_opa_handle_t am_zlg118_opa_inst_init (void);
18	PCNT	am_zlg118_pcnt_handle_t am_zlg118_pcnt1_inst_init (void);

下页继续

表 4.7 – 续上页

序号	外设	实例初始化函数原型
19	RTC	am_rtc_handle_t am_zlg118_rtc_inst_init (void);
20	SPI_DMA	am_spi_handle_t am_zlg118_spi0_dma_inst_init (void);
21	SPI_INT	am_spi_handle_t am_zlg118_spi1_int_inst_init (void);
22	SPI_POLL	am_spi_handle_t am_zlg118_spi1_poll_inst_init (void);
23	SYSTICK	am_timer_handle_t am_zlg118_systick_inst_init (void);
24	TIM0_CAP	am_cap_handle_t am_zlg118_tim0_cap_inst_init (void);
25	TIM1_CAP	am_cap_handle_t am_zlg118_tim1_cap_inst_init (void);
26	TIM2_CAP	am_cap_handle_t am_zlg118_tim2_cap_inst_init (void);
27	TIM3_CAP	am_cap_handle_t am_zlg118_tim3_cap_inst_init (void);
28	ADTIM4_CAP	am_cap_handle_t am_zlg118_tim4_cap_inst_init (void);
29	ADTIM5_CAP	am_cap_handle_t am_zlg118_tim5_cap_inst_init (void);
30	ADTIM6_CAP	am_cap_handle_t am_zlg118_tim6_cap_inst_init (void);
31	TIM0_PWM	am_pwm_handle_t am_zlg118_tim0_pwm_inst_init (void);
32	TIM1_PWM	am_pwm_handle_t am_zlg118_tim1_pwm_inst_init (void);
33	TIM2_PWM	am_pwm_handle_t am_zlg118_tim2_pwm_inst_init (void);
34	TIM3_PWM	am_pwm_handle_t am_zlg118_tim3_pwm_inst_init (void);
35	ADTIM4_PWM	am_pwm_handle_t am_zlg118_tim4_pwm_inst_init (void);
36	ADTIM5_PWM	am_pwm_handle_t am_zlg118_tim5_pwm_inst_init (void);
37	ADTIM6_PWM	am_pwm_handle_t am_zlg118_tim6_pwm_inst_init (void);
38	TIM0_TIMING	am_timer_handle_t am_zlg118_tim0_timing_inst_init (void);
39	TIM1_TIMING	am_timer_handle_t am_zlg118_tim1_timing_inst_init (void);
40	TIM2_TIMING	am_timer_handle_t am_zlg118_tim2_timing_inst_init (void);
41	TIM3_TIMING	am_timer_handle_t am_zlg118_tim3_timing_inst_init (void);
42	ADTIM4_TIMING	am_timer_handle_t am_zlg118_tim4_timing_inst_init (void);
43	ADTIM5_TIMING	am_timer_handle_t am_zlg118_tim5_timing_inst_init (void);
44	ADTIM6_TIMING	am_timer_handle_t am_zlg118_tim6_timing_inst_init (void);
45	TRNG	am_trng_handle_t am_zlg118_trng_inst_init (void);
46	UART0	am_uart_handle_t am_zlg118_uart0_inst_init (void);
47	UART1	am_uart_handle_t am_zlg118_uart1_inst_init (void);
48	UART2	am_uart_handle_t am_zlg118_uart2_inst_init (void);
49	UART3	am_uart_handle_t am_zlg118_uart3_inst_init (void);
50	VC	am_vc_handle_t am_zlg118_vc_inst_init (void);
51	WDT	am_wdt_handle_t am_zlg118_wdt_inst_init (void);

4.3.1.2 操作外设

根据实例初始化函数的返回值类型，可以判断后续该如何继续操作该外设。实例初始化函数的返回值可能有以下三类：

- int 型；
- 标准服务 handle 类型（**am_*_handle_t**，类型由标准接口层定义），如 **am_adc_handle_t**；
- 驱动自定义 handle 类型（**am_zlg118_*_handle_t**，类型由驱动头文件自定义），如 **am_zlg118_crc_handle_t**。

下面分别介绍这三种不同返回值的含义以及实例初始化后，该如何继续使用该外设。

1. 返回值为 int 型

常见的全局资源外设对应的实例初始化函数的返回值均为 int 类型。相关外设详见 表 4.8 。

表 4.8 返回值为 int 类型的实例初始化函数

序号	外设	实例初始化函数原型
1	CLK	int am_zlg118_clk_inst_init (void);
2	DMA	int am_zlg118_dma_inst_init (void);
3	GPIO	int am_zlg118_gpio_inst_init (void);
4	NVIC 中断	int am_zlg118_nvic_inst_init (void);

若返回值为 **AM_OK**，表明实例初始化成功；否则，表明实例初始化失败，需要检查设备相关的配置信息。

后续操作该类外设直接使用相关的接口操作即可，根据接口是否标准化，可以将操作该外设的接口分为两类。

接口已标准化，如 GPIO 提供了标准接口，在 {SDK}**\ametal\common\interface\am_gpio.h** 文件中声明。则可以查看相关接口说明和示例，以使用 GPIO。简单示例如 列表 4.57 。

列表 4.57 GPIO 标准接口使用范例

```
1  am_gpio_pin_cfg(PIOA_0, AM_GPIO_OUTPUT_INIT_HIGH);   /* 将 GPIO 配置为输出模式并且为高电平 */
```

参见：

接口原型及详细的使用方法请参考 {SDK}**\documents\《AMetal AM118-Core API 参考手册.chm》** 或者 {SDK}**\ametal\common\interface\am_gpio.h** 文件。

接口未标准化，则相关接口由驱动头文件自行提供，如 DMA，相关接口在 {SDK}**\ametal\soc\st\drivers\include\dma\am_zlg118_dma.h** 文件中声明。

注意： 无论是标准接口还是非标准接口，使用前，均需要包含对应的接口头文件。需要特别注意的是，这些全局资源相关的外设设备，一般在系统启动时已默认完成初始化，无需用户再自行初始化。详见 表 3.1 。

2. 返回值为标准服务句柄

有些外设实例初始化函数后返回的是标准服务句柄，相关外设详见 表 4.9 。可以看到，绝大部分外设实例初始化函数，均是返回标准的服务句柄。若返回值不为 **NULL**，表明初始化成功；否则，初始化失败，需要检查设备相关的配置信息。

表 4.9 返回值为标准服务句柄的实例初始化函数

序号	外设	实例初始化函数原型
1	ADC	am_adc_handle_t am_zlg118_adc_inst_init (void);
2	AES	am_aes_handle_t am_zlg118_aes_inst_init (void);
3	CRC	am_crc_handle_t am_zlg118_crc_inst_init (void);
4	DAC	am_dac_handle_t am_zlg118_dac0_inst_init (void);
5	I^2c0	am_i2c_handle_t am_zlg118_i2c0_inst_init (void);
6	I^2c1	am_i2c_handle_t am_zlg118_i2c1_inst_init (void);
7	LCD	am_lcd_handle_t am_zlg118_lcd_inst_init (void);
8	LPTIM0	am_timer_handle_t am_zlg118_lptim0_timing_inst_init (void);
9	LPTIM1	am_timer_handle_t am_zlg118_lptim1_timing_inst_init (void);
10	LPUART0	am_uart_handle_t am_zlg118_lpuart0_inst_init (void);
11	LPUART1	am_uart_handle_t am_zlg118_lpuart1_inst_init (void);
12	LVD	am_lvd_handle_t am_zlg118_lvd_inst_init (void);
13	OPA	am_opa_handle_t am_zlg118_opa_inst_init (void);
14	PCNT	am_zlg118_pcnt_handle_t am_zlg118_pcnt1_inst_init (void);
15	RTC	am_rtc_handle_t am_zlg118_rtc_inst_init (void);
16	SPI_DMA	am_spi_handle_t am_zlg118_spi0_dma_inst_init (void);
17	SPI_INT	am_spi_handle_t am_zlg118_spi1_int_inst_init (void);
18	SPI_POLL	am_spi_handle_t am_zlg118_spi1_poll_inst_init (void);
19	SYSTICK	am_timer_handle_t am_zlg118_systick_inst_init (void);
20	TIM0_CAP	am_cap_handle_t am_zlg118_tim0_cap_inst_init (void);
21	TIM1_CAP	am_cap_handle_t am_zlg118_tim1_cap_inst_init (void);
22	TIM2_CAP	am_cap_handle_t am_zlg118_tim2_cap_inst_init (void);
23	TIM3_CAP	am_cap_handle_t am_zlg118_tim3_cap_inst_init (void);
24	ADTIM4_CAP	am_cap_handle_t am_zlg118_tim4_cap_inst_init (void);
25	ADTIM5_CAP	am_cap_handle_t am_zlg118_tim5_cap_inst_init (void);
26	ADTIM6_CAP	am_cap_handle_t am_zlg118_tim6_cap_inst_init (void);
27	TIM0_PWM	am_pwm_handle_t am_zlg118_tim0_pwm_inst_init (void);
28	TIM1_PWM	am_pwm_handle_t am_zlg118_tim1_pwm_inst_init (void);
29	TIM2_PWM	am_pwm_handle_t am_zlg118_tim2_pwm_inst_init (void);
30	TIM3_PWM	am_pwm_handle_t am_zlg118_tim3_pwm_inst_init (void);
31	ADTIM4_PWM	am_pwm_handle_t am_zlg118_tim4_pwm_inst_init (void);
32	ADTIM5_PWM	am_pwm_handle_t am_zlg118_tim5_pwm_inst_init (void);
33	ADTIM6_PWM	am_pwm_handle_t am_zlg118_tim6_pwm_inst_init (void);
34	TIM0_TIMING	am_timer_handle_t am_zlg118_tim0_timing_inst_init (void);
35	TIM1_TIMING	am_timer_handle_t am_zlg118_tim1_timing_inst_init (void);
36	TIM2_TIMING	am_timer_handle_t am_zlg118_tim2_timing_inst_init (void);

下页继续

表 4.9 – 续上页

序号	外设	实例初始化函数原型
37	TIM3_TIMING	am_timer_handle_t am_zlg118_tim3_timing_inst_init (void);
38	ADTIM4_TIMING	am_timer_handle_t am_zlg118_tim4_timing_inst_init (void);
39	ADTIM5_TIMING	am_timer_handle_t am_zlg118_tim5_timing_inst_init (void);
40	ADTIM6_TIMING	am_timer_handle_t am_zlg118_tim6_timing_inst_init (void);
41	TRNG	am_trng_handle_t am_zlg118_trng_inst_init (void);
42	UART0	am_uart_handle_t am_zlg118_uart0_inst_init (void);
43	UART1	am_uart_handle_t am_zlg118_uart1_inst_init (void);
44	UART2	am_uart_handle_t am_zlg118_uart2_inst_init (void);
45	UART3	am_uart_handle_t am_zlg118_uart3_inst_init (void);
46	VC	am_vc_handle_t am_zlg118_vc_inst_init (void);
47	WDT	am_wdt_handle_t am_zlg118_wdt_inst_init (void);

对于这些外设，后续可以利用返回的 handle 来使用相应的标准接口层函数。使用标准接口层函数的相关代码是可跨平台复用的！

例如，ADC 设备的实例初始化函数的返回值类型为 **am_adc_handle_t** ，为了方便后续使用，可以定义一个变量保存下该返回值，后续就可以使用该 handle 完成电压的采集了。详见 列表 4.58 。

列表 4.58 ADC 简单操作示例

```
#include "ametal.h"
#include "am_vdebug.h"
#include "am_board.h"
#include "am_zlg118.h"
#include "am_zlg118_adc.h"
#include "am_zlg118_inst_init.h"

/**
 * \brief ADC 硬件触发转换，DMA 传输转换结果，通过标准接口实现
 *
 * \return 无
 */
int am_main (void)
{
    int             i;
    int             chan = 0;     /* 通道 0 */
    uint32_t        adc_mv[5];    /* 采样电压 */
    am_adc_handle_t adc_handle;  /* ADC 标准服务操作句柄 */

    am_zlg118_dma_inst_init ();      /* DMA 实例初始化 */

    adc_handle = am_zlg118_adc_inst_init (); /* ADC 实例初始化并获取句柄值 */

    am_kprintf("The ADC STD HT Demo\r\n");

    while (1) {

        /* 获取 ADC 采集电压，采样完成才返回 */
        am_adc_read_mv(adc0_handle, chan, adc_mv, 5);
        for (i = 1; i < 5; i++) {
            adc_mv[0] += adc_mv[i];
        }
```

```
33          adc_mv[0] /= 5;
34          am_kprintf("Vol: %d mv\r\n", adc_mv[0]);
35      }
36  }
```

4.3.1.3 解初始化

外设使用完毕后，应该调用相应设备配置文件提供的设备实例解初始化函数，以释放相关资源。所有外设的实例解初始化函数均在 {PROJECT}**\user_config\am_zlg118_inst_init.h** 文件中声明。使用实例解初始化函数前，应确保已包含 **am_zlg118_inst_init.h** 头文件。各个外设对应的设备实例解初始化函数的原型详见 表 4.10 。

表 4.10 片上外设及对应的实例解初始化函数

序号	外设	实例解初始化函数原型
1	ADC	void am_zlg118_adc_inst_init (am_adc_handle_t);
2	AES	void am_zlg118_aes_inst_init (am_aes_handle_t);
3	NVIC	void am_zlg118_nvic_inst_init (void);
4	CRC	void am_zlg118_crc_inst_init (am_crc_handle_t);
5	DAC	viod am_zlg118_dac0_inst_init (am_dac_handle_t);
6	DMA	void am_zlg118_dma_inst_init (void);
7	GPIO	void am_zlg118_gpio_inst_init (void);
8	I^2c0	void am_zlg118_i2c0_inst_init (am_i2c_handle_t);
9	I^2c1	void am_zlg118_i2c1_inst_init (am_i2c_handle_t);
10	LCD	void am_zlg118_lcd_inst_init (am_lcd_handle_t);
11	LPTIM0	void am_zlg118_lptim0_timing_inst_init (am_timer_handle_t);
12	LPTIM1	void am_zlg118_lptim1_timing_inst_init (am_timer_handle_t);
13	LPUART0	void am_zlg118_lpuart0_inst_init (am_uart_handle_t);
14	LPUART1	void am_zlg118_lpuart1_inst_init (am_uart_handle_t);
15	LVD	void am_zlg118_lvd_inst_init (am_lvd_handle_t);
16	OPA	void am_zlg118_opa_inst_init (am_opa_handle_t);
17	PCNT	void am_zlg118_pcnt1_inst_init (am_zlg118_pcnt_handle_t);
18	RTC	void am_zlg118_rtc_inst_init (am_rtc_handle_t);
19	SPI_DMA	void am_zlg118_spi0_dma_inst_init (am_spi_handle_t);
20	SPI_INT	void am_zlg118_spi1_int_inst_init (am_spi_handle_t);
21	SPI_POLL	void am_zlg118_spi1_poll_inst_init (am_spi_handle_t);
22	SYSTICK	void am_zlg118_systick_inst_init (am_timer_handle_t);
23	TIM0_CAP	void am_zlg118_tim0_cap_inst_init (am_cap_handle_t);
24	TIM1_CAP	void am_zlg118_tim1_cap_inst_init (am_cap_handle_t);
25	TIM2_CAP	void am_zlg118_tim2_cap_inst_init (am_cap_handle_t);
26	TIM3_CAP	void am_zlg118_tim3_cap_inst_init (am_cap_handle_t);
27	ADTIM4_CAP	void am_zlg118_tim4_cap_inst_init (am_cap_handle_t);

下页继续

表 4.10 – 续上页

序号	外设	实例解初始化函数原型
28	ADTIM5_CAP	void am_zlg118_tim5_cap_inst_init (am_cap_handle_t);
29	ADTIM6_CAP	void am_zlg118_tim6_cap_inst_init (am_cap_handle_t);
30	TIM0_PWM	void am_zlg118_tim0_pwm_inst_init (am_pwm_handle_t);
31	TIM1_PWM	void am_zlg118_tim1_pwm_inst_init (am_pwm_handle_t);
32	TIM2_PWM	void am_zlg118_tim2_pwm_inst_init (am_pwm_handle_t);
33	TIM3_PWM	void am_zlg118_tim3_pwm_inst_init (am_pwm_handle_t);
34	ADTIM4_PWM	void am_zlg118_tim4_pwm_inst_init (am_pwm_handle_t);
35	ADTIM5_PWM	void am_zlg118_tim5_pwm_inst_init (am_pwm_handle_t);
36	ADTIM6_PWM	void am_zlg118_tim6_pwm_inst_init (am_pwm_handle_t);
37	TIM0_TIMING	void am_zlg118_tim0_timing_inst_init (am_timer_handle_t);
38	TIM1_TIMING	void am_zlg118_tim1_timing_inst_init (am_timer_handle_t);
39	TIM2_TIMING	void am_zlg118_tim2_timing_inst_init (am_timer_handle_t);
40	TIM3_TIMING	void am_zlg118_tim3_timing_inst_init (am_timer_handle_t);
41	ADTIM4_TIMING	void am_zlg118_tim4_timing_inst_init (am_timer_handle_t);
42	ADTIM5_TIMING	void am_zlg118_tim5_timing_inst_init (am_timer_handle_t);
43	ADTIM6_TIMING	void am_zlg118_tim6_timing_inst_init (am_timer_handle_t);
44	TRNG	void am_zlg118_trng_inst_init (am_trng_handle_t);
45	UART0	void am_zlg118_uart0_inst_init (am_uart_handle_t);
46	UART1	void am_zlg118_uart1_inst_init (am_uart_handle_t);
47	UART2	void am_zlg118_uart2_inst_init (am_uart_handle_t);
48	UART3	void am_zlg118_uart3_inst_init (am_uart_handle_t);
49	VC	void am_zlg118_vc_inst_init (am_vc_handle_t);
50	WDT	void am_zlg118_wdt_inst_init (am_wdt_handle_t);

注意： 时钟部分不能被解初始化。

外设实例解初始化函数相对简单，所有实例解初始化函数均无返回值。

关于解初始化函数的参数，若实例初始化时返回值为 **int** 类型，则实例解初始化时无需传入任何参数；若实例初始化函数返回了一个服务句柄，则实例解初始化时应该传入实例初始化函数获得的服务句柄。

4.3.2 直接使用硬件层函数

一般情况下，使用设备实例初始化函数返回的 handle，再利用标准接口层或驱动层提供的函数。已经能满足绝大部分应用场合。若在一些效率要求很高或功能要求很特殊的场合，可能需要直接操作硬件。此时，则可以直接使用 HW 层提供的相关接口。

通常，HW 层的接口函数都是以外设寄存器结构体指针为参数（特殊地，系统控制部分功能混杂，默认所有函数直接操作 SYSCON 各个功能，无需再传入相应的外设寄存器结构

体指针）。

以 SPI 为例，所有硬件层函数均在 {SDK}**\ametal\soc\zlg\drivers\include\spi\hw\amhw_zlg118_spi.h** 文件中声明（一些简单的内联函数直接在该文件中定义）。简单列举几个函数，详见 列表 4.59 。

列表 4.59 SPI 硬件层操作函数

```
/**
 * \brief SPI 模块使能设置
 * \param[in] p_hw_spi : 指向 SPI 寄存器结构体的指针
 * \param[in] flag     : AM_TRUE or AM_FALSE
 * \return none
 */
am_static_inline
void amhw_zlg118_spi_enable (amhw_zlg118_spi_t *p_hw_spi, am_bool_t flag)
{
    p_hw_spi->cr = (p_hw_spi->cr & (~(1u << 6))) | (flag << 6);
}
```

其它一些函数读者可自行打开 {SDK}** \ametal\soc\zlg\drivers\include\spi\hw\amhw_zlg118_spi.h** 文件查看。这些函数均是以 amhw_zlg118_spi_t * 类型作为第一个参数。**amhw_zlg118_spi_t** 类型在 {SDK}**\ametal\soc\st\drivers\spi\hw\include\amhw_zlg118_spi.h** 文件中定义，用于定义出 SPI 外设的各个寄存器。详见 列表 4.60 。

列表 4.60 SPI 寄存器结构体定义

```
/**
 * \brief SPI structure of register
 */
typedef struct amhw_zlg118_spi {
    __IO uint32_t  cr;       /**< \brief SPI 配置寄存器 */
    __IO uint32_t  ssn;      /**< \brief SPI 片选配置寄存器 */
    __I  uint32_t  stat;     /**< \brief SPI 状态寄存器 */
    __IO uint32_t  data;     /**< \brief SPI 数据寄存器 */
    __IO uint32_t  cr2;      /**< \brief SPI 配置寄存器 2 */
    __O  uint32_t  iclr;     /**< \brief SPI 中断清除寄存器 */
} amhw_zlg118_spi_t;
```

该类型的指针已经在 {SDK}**\ametal\soc\zlg\zlg118\zlg118_periph_map.h** 文件中定义，应用程序可以直接使用。详见 列表 4.61 。

列表 4.61 SPI 寄存器结构体指针定义

```
/** \brief SPI0 寄存器块指针 */
#define ZLG118_SPI0           ((amhw_zlg118_spi_t *)ZLG118_SPI0_BASE)
```

注解： 其中的 **zlg118_SPI0** 是 **SPI0** 外设寄存器的基地址，在 **{SDK}**\ametal\soczlg\zlg118\zlg118_regbase.h** 文件中定义，其他所有外设的基地址均在该文件中定义。

有了这个 SPI 寄存器结构体指针宏后，就可以直接使用 SPI 硬件层的相关函数了。使用硬件层函数时，若传入参数为 **zlg118_SPI0** ，则表示操作的是 SPI0。如 列表 4.62 所示，用于使能 SPI0。

列表 4.62 使能 SPI0

```
amhw_zlg118_spi_enable(zlg118_SPI0);
```

特别地，可能想要操作的功能，硬件层也没有提供出相关接口，此时，可以基于各个外设指向寄存器结构体的指针，直接操作寄存器实现，例如，要使能 SPI0，也可以直接设置寄存器的值，详见 列表 4.63。

列表 4.63 直接设置寄存器的值使能 SPI0

```
/*
 * 设置 CR1 寄存器的 bit6 为 1，使能 SPI0
 */
amhw_zlg118_spi_t->CR |= (1 << 6);
```

注解： 一般情况下，均无需这样操作。若特殊情况下需要以这种方式操作寄存器，应详细了解该寄存器各个位的含义，谨防出错。

所有外设均在 {SDK}**\ametal\soc\zlgzlg118\zlg118_periph_map.h** 文件中定义了指向外设寄存器的结构体指针，与各外设对应的指向该外设寄存器的结构体指针宏详见 表 4.11 。

表 4.11 指向各片上外设寄存器结构体的指针宏

序号	外设	指向该外设寄存器结构体的指针宏
1	UART0	ZLG118_UART0
2	UART1	ZLG118_UART1
3	UART2	ZLG118_UART2
4	UART3	ZLG118_UART3
5	LPUART0	ZLG118_LPUART0
6	LPUART1	ZLG118_LPUART1
7	I^2C0	ZLG118_I2C0
8	I^2C1	ZLG118_I2C1
9	SPI0	ZLG118_SPI0
10	SPI1	ZLG118_SPI1
11	TIM0	ZLG118_TIM0
12	TIM1	ZLG118_TIM1
13	TIM2	ZLG118_TIM2
14	TIM3	ZLG118_TIM3
15	TIM4	ZLG118_TIM4
16	TIM5	ZLG118_TIM5
17	TIM6	ZLG118_TIM6
18	LPTIM0	ZLG118_LPTIM0
19	LPTIM1	ZLG118_LPTIM1
20	AES	ZLG118_AES

下页继续

表 4.11 – 续上页

序号	外设	指向该外设寄存器结构体的指针宏
21	ADC	ZLG118_ADC
22	CLKTRIM	ZLG118_CLKTRIM
23	CRC	ZLG118_CRC
24	DAC	ZLG118_DAC
25	DMA	ZLG118_DMA
26	FLASH	ZLG118_FLASH
27	GPIO0	ZLG118_GPIO0
28	LCD	ZLG118_LCD
29	LVD	ZLG118_LVD
30	RCC	ZLG118_RCC
31	OPA	ZLG118_OPA
32	PCNT	ZLG118_PCNT
33	RAM	ZLG118_RAM
34	PCA	ZLG118_PCA
35	RTC	ZLG118_RTC
36	TRNG	ZLG118_TRNG
37	VC	ZLG118_VC
38	WDT	ZLG118_WDT

5. 板级资源

与板级相关的资源默认情况下使能即可使用。特殊情况下，LED、蜂鸣器、按键、调试串口、温度传感器 LM75、系统滴答和软件定时器可能需要进行一些配置。所有资源的配置由 {HWCONFIG} 下的一组 am_hwconf_* 开头的 **.c** 文件完成的。

各资源及其对应的配置文件如 表 5.1 所示。

表 5.1 板级资源的配置文件

序号	外设	配置文件
1	按键	am_hwconf_key_gpio.c
2	LED	am_hwconf_led_gpio.c
3	蜂鸣器	am_hwconf_buzzer.c
4	温度传感器 (LM75)	am_hwconf_lm75.c
5	调试串口	am_hwconf_debug_uart.c
6	系统滴答和软件定时器	am_hwconf_system_tick_softimer.c

5.1 配置文件结构

板级资源的配置文件与片上外设配置文件结构基本类似。一般来说，板级资源配置只需要设备信息和实例初始化函数即可。而且实例初始化函数通常情况下不需要用户手动调用，也不需要用户自己修改。只需要在工程配置文件

{PROJECT}**user_config\am_prj_config.h** 中打开或禁用相应的宏，相关资源会在系统启动时在 {SDK}**ametal\board\am118_core\am_board.c** 中自动完成初始化。以 LED 为例，初始化代码详见 列表 5.1 。

列表 5.1 LED 实例初始化函数调用

```
/**
 * \brief 板级初始化
 */
void am_board_init (void)
{
    ......
#if (AM_CFG_LED_ENABLE == 1)
    am_led_gpio_inst_init();
#endif /* (AM_CFG_LED_ENABLE == 1) */
    ......
}
```

5.2 典型配置

5.2.1 LED 配置

AM824-Core 板上有两个 LED 灯，默认引脚分别为 PIO0_20 和 PIO0_21，使用时，需要使用跳线帽短接 AM118-Core 板上的 J9 和 J10。LED 相关信息定义在 {SDK}**user_config\am_hwconf_usrcfg\am_hwconf_led_gpio.c** 文件中，详见 列表 5.2 。

列表 5.2 LED 相关配置信息

```
/** \brief LED 引脚号 */
am_local am_const int __g_led_pins[] = {
    PIOC_9,     /* LED0 引脚 */
    PIOC_10     /* LED1 引脚 */
};

/** \brief LED 设备信息 */
am_local am_const am_led_gpio_info_t __g_led_gpio_devinfo = {
    {
        0,                                  /* LED 起始编号 */
        AM_NELEMENTS(__g_led_pins) - 1    /* LED 结束编号 */
    },
    __g_led_pins,
    AM_TRUE
};
```

其中，am_led_gpio_info_t类型在 {SDK}**ametal\common\components\service\include\am_led_gpio.h** 文件中定义，详见 列表 5.3 。

列表 5.3 LED 引脚配置信息类型定义

```
typedef struct am_led_gpio_info {

   /** \brief LED 基础服务信息 ，包含起始编号和结束编号      */
   am_led_servinfo_t  serv_info;

   /** \brief 使用的 GPIO 引脚，引脚数目应该为 （结束编号 - 起始编号 + 1）    */
   const int         *p_pins;

   /** \brief LED 是否是低电平点亮  */
   am_bool_t          active_low;

```

```
} am_led_gpio_info_t;
```

其中，serv_info 为 LED 的基础服务信息，包含 LED 的起始编号和介绍编号，p_pins 指向存放 LED 引脚的数组首地址，在本平台可选择的管脚在 zlg118_pin.h 文件中定义，active_low 参数用于确定其点亮电平，若是低电平点亮，则该值为 AM_TRUE，否则，该值为 AM_FALSE。

可见，在 LED 配置信息中，LED0 和 LED1 分别对应 PIOC_9 和 PIOC_10，均为低电平点亮。如需添加更多的 LED，只需在该配置信息数组中继续添加即可。

可使用 LED 标准接口操作这些 LED，详见 {SDK}**\ametal\common\interface\am_led.h** 。led_id 参数与该数组对应的索引号一致。

注解： 由于 LED 使用了 PIOC_9 和 PIOC_10，若应用程序需要使用这两个引脚，建议通过使能/禁能宏禁止 LED 资源的使用。

5.2.2 蜂鸣器配置

板载蜂鸣器为无源蜂鸣器，需要使用 PWM 驱动才能实现发声。默认使用 SCT 的输出通道 1（SCT_OUT1）输出 PWM。可以通过 {PROJECT}**\user_config\am_hwconf_usrcfg\am_hwconf_buzzer.c** 文件中的两个相关宏来配置 PWM 的频率和占空比，相应宏名及含义详见 表 5.2 。

表 5.2 蜂鸣器配置相关宏

宏名	含义
__BUZZER_PWM_FREQ	PWM 的频率，默认为 2.5KHz
__BUZZER_PWM_DUTY	PWM 的占空比，默认为 50（即 50%）

注解： 由于蜂鸣器使用了 SCT 的 PWM 功能，若应用程序需要使用 SCT，建议通过使能/禁能宏禁止蜂鸣器的使用，以免冲突。

5.2.3 按键

AM118-Core 有两个板载按键 KEY/ RES 和 RST，默认引脚分别为 PIOC_7 和 RST，使用时，需要使用跳线帽短接 AM118-Core 板上的 J14 和 J8。其中 RST 默认为复位按键，可供使用的按键只有 KEY/ RES。KEY 相关信息定义在 {SDK}**\user_config\am_hwconf_usrcfg\am_hwconf_key_gpio.c** 文件中，详见 列表 5.4 。

列表 5.4 KEY 相关配置信息

```
/** \brief 按键引脚号 */
am_local am_const int __g_key_pins[] = {
    PIO0_1    /* KEY/RES 按键引脚 */
};

/** \brief 按键编码 */
am_local am_const int __g_key_codes[] = {
```

```
    KEY_KP0    /* KEY/RES 按键编码 */
};

/** \brief 按键设备信息 */
am_local am_const am_key_gpio_info_t __g_key_gpio_devinfo = {
    __g_key_pins,               /* 按键引脚号 */
    __g_key_codes,              /* 各个按键对应的编码（上报） */
    AM_NELEMENTS(__g_key_pins), /* 按键数目 */
    AM_TRUE,                    /* 是否低电平激活（按下为低电平） */
    10                          /* 按键扫描时间间隔，一般为 10ms */
};
```

其中，KEY_KP0 为默认按键编号；AM_NELEMENTS 是计算按键个数的宏函数；am_key_gpio_info_t类型在 {SDK}**\ametal\common\components\service\include\am_key_gpio.h** 文件中定义，详见 列表 5.5 。

列表 5.5 KEY 引脚配置信息类型定义

```
/**
 * \brief 按键信息
 */
typedef struct am_key_gpio_info {
    const int  *p_pins;             /**< \brief 使用的引脚号                 */
    const int  *p_codes;            /**< \brief 各个按键对应的编码（上报） */
    int         pin_num;            /**< \brief 按键数目 */
    am_bool_t   active_low;         /**< \brief 是否低电平激活（按下为低电平）
↪*/
    int         scan_interval_ms;   /**< \brief 按键扫描时间间隔，一般 10ms */
} am_key_gpio_info_t;
```

p_pins 指向存放 KEY 引脚的数组首地址，在本平台可选择的管脚在 lpc82x_pin.h 文件中定义；p_codes 指向存放按键对应编码的数组首地址；pin_num 为按键数目；active_low 参数用于确定其点亮电平，若是低电平点亮，则该值为 AM_TRUE，否则，该值为 AM_FALSE；scan_interval_ms 按键扫描时间，一般为 10ms 。

可见，在 KEY 配置信息中，KEY/RES 对应 PIOC_7 ，低电平有效。如需添加更多的 KEY，只需在 **__g_key_pins** 和 **__g_key_codes** 数组中继续添加按键对应的管脚和编码即可。

注解： 由于 KEY/RES 使用了 PIOC_7，若应用程序需要使用这个引脚，建议通过使能/禁能宏禁止 KEY 资源的使用。为保证芯片能正常复位，复位按键 RST 对应的 PIO0_5 管脚不建议配置为普通的 GPIO 口。

5.2.4 调试串口配置

AM118-Core 具有 4 个串口，可以选择使用其中一个串口来输出调试信息。使用 {PROJECT}**\user_config\am_hwconf_usrcfg\am_hwconf_debug_uart.c** 文件中的两个相关宏用来配置使用的串口号和波特率，相应宏名及含义详见 表 5.3 。

表 5.3 调试串口相关配置

宏名	含义
__DEBUG_UART	串口号，0-UART0，1-UART1，2-UART2，2-UART2
__DEBUG_BAUDRATE	使用的波特率，默认 115200

注解： 每个串口还可能需要引脚的配置，这些配置属于具体外设资源的配置，详见第 4 章中的相关内容。若应用程序需要使用串口，应确保调试串口与应用程序使用的串口不同，以免冲突。调试串口的其它配置固定为：8-N-1（8 位数据位，无奇偶校验，1 位停止位）。

5.2.5 系统滴答和软件定时器配置

系统滴答需要 TIM4 定时器为其提供一个周期性的定时中断，默认使用 TIM4 定时器的通道 0。其配置还需要使用 {PROJECT}**user_config\am_hwconf_usrcfg\am_hwconf_system_tick_softimer.c** 文件中的 **__SYSTEM_TICK_RATE** 宏来设置系统滴答的频率，默认为 1KHz。详细定义见 列表 5.6 。

列表 5.6 系统滴答频率配置

```
/** \brief 设置系统滴答的频率，默认 1KHz */
#define __SYSTEM_TICK_RATE      1000
```

软件定时器基于系统滴答实现。它的配置也需要使用 {PROJECT}**user_config\am_hwconf_usrcfg\am_hwconf_system_tick_softimer.c** 文件中的 **__SYSTEM_TICK_RATE** 宏来设置运行频率，默认 1KHz。详细定义见 列表 5.6 。

注解： 使用软件定时器时必须开启系统滴答。

5.2.6 温度传感器 LM75

AM118-Core 自带一个 LM75B 测温芯片，使用 LM75 传感器需要配置 {PROJECT}**user_config\am_hwconf_usrcfg\am_hwconf_lm75.c** 文件中 LM75 的实例信息 **__g_temp_lm75_info**，**__g_temp_lm75_info** 存放的是 I^2C 从机地址，详细定义见 列表 5.7 。

列表 5.7 LM75 地址配置

```
/** \brief LM75 设备信息 */
am_local am_const am_temp_lm75_info_t __g_temp_lm75_devinfo = {
    0x48    /* LM75 I2C 7 位地址  */
};
```

LM75 没有相应的使能/禁能宏，配置完成后，用户需要自行调用实例初始化函数获得温度标准服务操作句柄，通过标准句柄获取温度值。

5.3 使用方法

板级资源对应的设备实例初始化函数的原型详见 表 5.4 ，使用方法可以参考 4.3.1.2 。

表 5.4 板级资源及对应的实例初始化函数

序号	板级资源	实例初始化函数原型
1	按键	int am_key_gpio_inst_init (void);
2	LED	int am_led_gpio_inst_init (void);
3	蜂鸣器	am_pwm_handle_t am_buzzer_inst_init (void);
4	温度传感器 (LM75)	am_temp_handle_t am_temp_lm75_inst_init (void);
5	调试串口	am_uart_handle_t am_debug_uart_inst_init (void);
6	系统滴答	am_timer_handle_t am_system_tick_inst_init (void);
7	系统滴答和软件定时器	am_timer_handle_t am_system_tick_softimer_inst_init (void);

6. MicroPort 系列扩展板

为了便于扩展开发板功能，ZLG 制定了 MicroPort 接口标准，MicroPort 是一种专门用于扩展功能模块的硬件接口，其有效地解决了器件与 MCU 之间的连接和扩展。其主要功能特点如下：

- 具有标准的接口定义；
- 接口包括丰富的外设资源，支持 UART、I^2C、SPI、PWM、ADC 和 DAC 功能；
- 配套功能模块将会越来越丰富；
- 支持上下堆叠扩展。

AM118-Core 板载 1 路带扩展的 MicroPort 接口，如 图 6.1 所示。用户可以依据需求，选择或开发功能多样的 MicroPort 模块，快速灵活地搭建原型机。由于 ZLG118 片上资源有限，还有极少部分 MicroPort 接口定义的引脚功能不支持，其相应的引脚可以当做普通 I/O 使用。

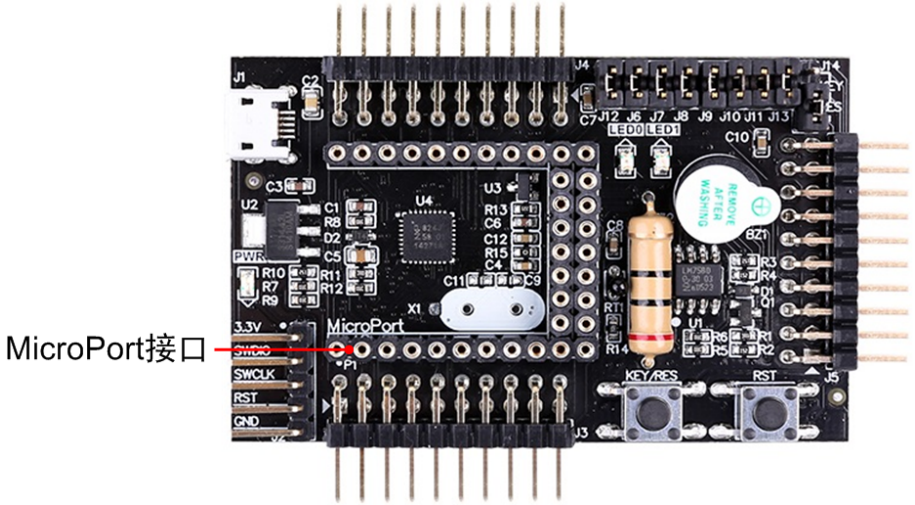

图 6.1 AM118-Core MicroPort

6.1 配置文件结构

当前可用的 MicroPort 扩展板有：MicroPort-DS1302、MicroPort-EEPROM、MicroPort-FLASH、MicroPort-RS485、MicroPort-RTC 和 MicroPort-RX8025T，与 MicroPort 相关的配置由 {PROJECT}**\user_config\am_hwconf_usrcfg** 下的一组 am_hwconf_microport_* 开头的 **.c** 文件完成的，通常情况下不需要用户自己修改，详见 表 6.1 。MicroPort 扩展板的配置文件

与片上外设配置文件结构基本类似。但是，MicroPort 扩展板的配置文件中不提供实例解初始化函数。

表 6.1 MicroPort 对应的配置文件

序号	外设	配置文件
1	MicroPort-DS1302	am_hwconf_microport_ds1302.c
2	MicroPort-EEPROM	am_hwconf_microport_eeprom.c
3	MicroPort-FLASH	am_hwconf_microport_flash.c
4	MicroPort-RS485	am_hwconf_microport_rs485.c
5	MicroPort-RTC	am_hwconf_microport_rtc.c
6	MicroPort-RX8025T	am_hwconf_microport_rx8025t.c

6.2 使用方法

MicroPort 扩展板对应的实例初始化函数的原型详见 表 6.2 。使用方法可以参考 4.3.1.2 ，也可以参考 {SDK}**ametal\examples\am118_core\microport_board** 目录下的例程。

表 6.2 MicroPort 扩展板实例初始化函数

序号	外设	实例初始化函数原型
1	MicroPort-DS1302(使用芯片特殊功能)	am_ds1302_handle_t am_microport_ds1302_inst_init (void);
2	MicroPort-DS1302(使用通用的 RTC 功能)	am_rtc_handle_t am_microport_ds1302_rtc_inst_init (void);
3	MicroPort-DS1302(用作系统时间)	int am_microport_ds1302_time_inst_init (void);
4	MicroPort-EEPROM(使用 EP24CXX 标准接口)	am_ep24cxx_handle_t am_microport_eeprom_inst_init (void);
5	MicroPort-EEPROM(用作标准的 NVRAM 器件)	int am_microport_eeprom_nvram_inst_init (void);
6	MicroPort-FLASH(使用 MX25XX 标准接口)	am_mx25xx_handle_t am_microport_flash_inst_init (void);
7	MicroPort-FLASH(使用 MTD 标准接口)	am_mtd_handle_t am_microport_flash_mtd_inst_init (void);
8	MicroPort-FLASH(使用 FTL 标准接口)	am_ftl_handle_t am_microport_flash_ftl_inst_init (void);
9	MicroPort-RS485	am_uart_handle_t am_microport_rs485_inst_init (void);
10	MicroPort-RTC(使用芯片特殊功能)	am_pcf85063_handle_t am_microport_rtc_inst_init (void);
11	MicroPort-RTC(使用通用的 RTC 功能)	am_rtc_handle_t am_microport_rtc_rtc_inst_init (void);
12	MicroPort-RTC(使用通用的闹钟功能)	am_alarm_clk_handle_t am_microport_rtc_alarm_clk_inst_init (void);
13	MicroPort-RTC(用作系统时间)	int am_microport_rtc_time_inst_init (void);
14	MicroPort-RX8025T(使用芯片特殊功能)	am_rx8025t_handle_t am_microport_rx8025t_inst_init (void);
15	MicroPort-RX8025T(使用通用的 RTC 功能)	am_rtc_handle_t am_microport_rx8025t_rtc_inst_init (void);
16	MicroPort-RX8025T(使用通用的闹钟功能)	am_alarm_clk_handle_t am_microport_rx8025t_alarm_clk_inst_init (void);
17	MicroPort-RX8025T(用作系统时间)	int am_microport_rx8025t_time_inst_init (void);

7. MiniPort 系列扩展板

MiniPort 接口是一个通用板载标准硬件接口，通过该接口可以与配套的标准模块相连，便于进一步简化硬件设计和扩展。其特点如下：

- 采用标准的接口定义，采用 2x10 间距 2.54mm 的 90° 弯针；
- 可同时连接多个扩展接口模块；
- 具有 16 个通用 I/O 端口；
- 支持 1 路 SPI 接口；
- 支持 1 路 I^2C 接口；
- 支持 1 路 UART 接口；
- 支持 1 路 3.3V 和 1 路 5V 电源接口。

AM118-Core 开发板搭载了 1 路 MiniPort，接口标号 J4，如 图 7.1 所示。

注意：由于 LPC118 芯片 I/O 口有限，AM118-Core 平台中 J3 和 J4 接口定义完全相同，但是 J3 和 J5 一样都是扩展接口。在 I/O 口资源比较多的平台中，J3 扩展接口和 J4 MiniPort 接口的定义是不相同的。

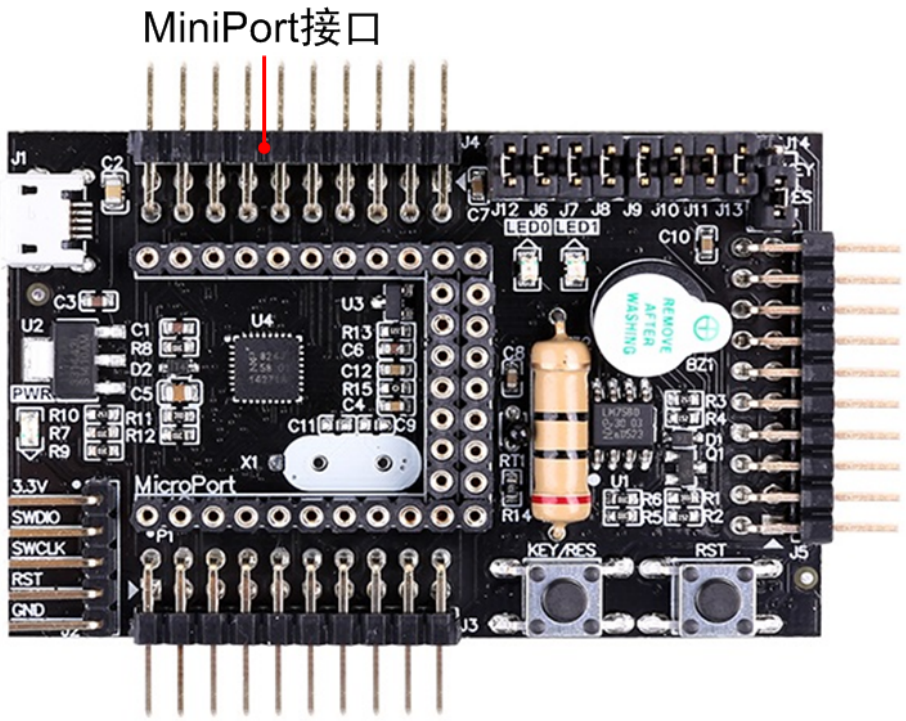

图 7.1 AM118-Core MiniPort

7.1 配置文件结构

当前可用的 MiniPort 扩展板有：MiniPort-595、MiniPort-KEY、MiniPort-LED、MiniPort-VIEW 和 MiniPort-ZLG72128，与 MiniPort 相关的配置由 {PROJECT}**user_config\am_hwconf_usrcfg** 下的一组 am_hwconf_miniport_* 开头的 **.c** 文件完成的，通常情况下不需要用户自己修改，详见 表 7.1 。MiniPort 扩展板的配置文件与片上外设配置文件结构基本类似。但是，MiniPort 扩展板的配置文件中不提供实例解初始化函数。

表 7.1 MiniPort 对应的配置文件

序号	外设	配置文件
1	MiniPort-595	am_hwconf_miniport_595.c
2	MiniPort-KEY	am_hwconf_miniport_key.c
3	MiniPort-LED	am_hwconf_miniport_led.c
4	MiniPort-VIEW	am_hwconf_miniport_view.c
5	MiniPort-VIEW KEY	am_hwconf_miniport_view_key.c
6	MiniPort-ZLG72128	am_hwconf_miniport_zlg72128.c

7.2 使用方法

MiniPort 扩展板对应的实例初始化函数的原型详见 表 7.2 。使用方法可以参考 4.3.1.2 ，也可以参考 {SDK}**ametal\examples\am118_core\miniport_board** 目录下的例程。

表 7.2 MiniPort 配板实例初始化函数

序号	外设	实例初始化函数原型
1	MiniPort-595	am_hc595_handle_t am_miniport_595_inst_init (void);
2	MiniPort-KEY	int am_miniport_key_inst_init (void);
3	MiniPort-LED	int am_miniport_led_inst_init (void);
4	MiniPort-LED(LED 595 联合初始化)	int am_miniport_led_595_inst_init (void);
5	MiniPort-View	int am_miniport_view_inst_init (void);
6	MiniPort-View(View 595 联合初始化)	int am_miniport_view_595_inst_init (void);
7	MiniPort-View KEY(View KEY 联合初始化)	int am_miniport_view_key_inst_init (void);
8	MiniPort-View KEY(View 595 KEY 联合初始化)	int am_miniport_view_key_595_inst_init (void);
9	MiniPort-ZLG72128	int am_miniport_zlg72128_inst_init (void);

MiniPort 扩展板通过排母与 AM118-Core 开发板相连，同时采用排针将所有引脚引出，实现模块的横向堆叠。例如实例初始化函数 int am_miniport_view_595_inst_init (void); 可以初始化 MiniPort-View 和 MiniPort-595，初始化成功之后能够通过 MiniPort-595 驱动 MiniPort-View。

8. 免责声明

应用信息：本应用信息适用于嵌入式产品的开发设计。客户在开发产品前，必须根据其产品特性给予修改并验证。

销售与服务网络

广州立功科技股份有限公司

地址：广州市天河区龙怡路 117 号银汇大厦 16 楼
邮编：510630
网址：www.zlgmcu.com

全国服务热线电话：400-888-2705

华南地区

广州总部

广州市天河区龙怡路 117 号银汇大厦 16 楼

华南汽车

深圳市坪山区比亚迪路大万文化广场 A 座 1705

厦门办事处

厦门市思明区厦禾路 855 号英才商厦 618 室

深圳分公司

深圳市福田区深南中路 2072 号电子大厦 1203 室

华东地区

上海分公司

上海市黄浦区北京东路 668 号科技京城东座 12E 室

苏州办事处

江苏省苏州市广济南路 258 号（百脑汇科技中心 1301 室）

南京分公司

南京市秦淮区汉中路 27 号友谊广场 17 层 F、G 区

合肥办事处

安徽省合肥市蜀山区黄山路 665 号汇峰大厦 1607

杭州分公司

杭州市西湖区紫荆花路 2 号杭州联合大厦 A 座 4 单元 508

宁波办事处

浙江省宁波市高新区星海南路 16 号轿辰大厦 1003

华北、东北地区

北京分公司

北京市海淀区紫金数码园 3 号楼（东华合创大厦）8 层 0802 室

天津办事处

天津市河东区十一经路与津塘公路交口鼎泰大厦 1004 室

山东办事处

山东省青岛市李沧区青山路 689 号宝龙公寓 3 号楼 701

沈阳办事处

沈阳市浑南新区营盘西街 17 号万达广场 A4 座 2722 室

华中地区

武汉分公司

武汉市武昌区武珞路 282 号思特大厦 807 室

西安办事处

西安市高新区科技二路 41 号高新水晶城 C 座 616 室

郑州办事处

河南郑州市中原区百花路与建设路东南角锦绣华庭 A 座 1502 室

长沙办事处

湖南省长沙市岳麓区奥克斯广场国际公寓 A 栋 2309 房

西南地区

重庆办事处

重庆市渝北区龙溪街道新溉大道 18 号山顶国宾城 11 幢 4-14

成都办事处

成都市一环路南二段 1 号数码科技大厦 403 室

请您用以上方式联系我们，我们会为您安排样机现场演示，感谢您对我公司产品的关注！