Rockchip HAL Interrupt

文件标识：RK-KF-YF-189

发布版本：V1.0.0

日期：2021-06-02

文件密级：□绝密 □秘密 □内部资料 ■公开

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

本文提供 HAL 平台的中断使用说明。

产品版本

芯片名称	内核版本
RK356X	HAL

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

修订记录

版本号	作者	修改日期	修改说明
V1.0.0	谢修鑫	2021-06-02	初始版本

目录

1. 功能支持

- 支持中断 handler 注册
- 支持 GPIO 中断 handler 注册
- 支持 GIC 中断的优先级抢占、嵌套
- 支持 AMP 系统 GIC 中断分组
- 支持 GPIO 中断优先级抢占、嵌套及 CPU 分组

2. 中断handler注册

对于只有 HAL 的裸系统，需要使用下面函数注册中断 handler。

1. 通过下面函数注册中断的 handler

```
HAL_Status HAL_IRQ_HANDLER_SetIRQHandler(uint32_t irq,
                                         HAL_IRQ_HANDLER handler, void *args);
```

2. 通过下面函数注册 GPIO 中断的 handler

```
HAL_Status HAL_IRQ_HANDLER_SetGpioIRQHandler(eGPIO_bankId bank,
                                             ePINCTRL_GPIO_PINS pin,
                                             HAL_IRQ_GPIO_HANDLER handler,
                                             void *args);
```

- 参数 bank：GPIO BANK 的 ID
- 参数 pin：PIN ID in a GPIO BANK
- 参数 args：私有参数

3. GIC 中断的优先级抢占、嵌套

通过下面下面宏定义开启：

```
HAL_GIC_PREEMPT_FEATURE_ENABLED
```

通过GIC的下面操作函数配置中断的优先级

```
HAL_Status HAL_GIC_SetPriority(uint32_t irq, uint32_t priority);
```

- 参数 priority：中断优先级，数值越小，优先级越高；ARM GIC 的优先级范围是0x0~0xFF，非安全中断的优先级范围为0x80~0xFF，一些 SOC 不会实现优先级的低4 bits，所以非安全中断的优先级建议为：0x80，0x90，0xA0，0xB0，0xC0，0xD0，0xE0。

4. AMP 系统 GIC 中断分组支持

对于AMP系统，由于每个CPU运行独立的环境；所以需要指定一个 CPU 进行 GIC 初始化及通用功能配置，由于 GIC 的优先级配置对于各个 CPU 存在竞争关系，所以各个 CPU 不能独立配置中断优先级，中断优先级需要指定的 CPU 进行统一配置。

4.1 初始化配置数据

通过下面数据控制 GIC 中断分组功能的初始化：

```
static struct GIC_IRQ_AMP_CTRL irqConfig = {
    .cpuAff = CPU_GET_AFFINITY(1, 0),
    .defPrio = 0xd0,
    .defRouteAff = CPU_GET_AFFINITY(1, 0),
    .irqsCfg = &irqsConfig[0],
};
```

- 参数 cpuAff：指定进行 GIC 初始化及通用功能配置的 CPU。指定的这个 CPU 必须是 AMP 系统最先启动的。
- 参数 defPrio：指定每个 IRQ 的默认优先级。
- 参数 defRouteAff：指定每个 IRQ 默认关联到的 CPU。
- 参数 irqsCfg：如果一个 IRQ 需要指定优先级，通过这个数组指定。对于没有在这里指定的 IRQ，优先级为默认值 defPrio。

CPU_GET_AFFINITY(1, 0) 说明：

- 第一个参数为 CPU ID。
- 第二个参数为CPU 对应的 Cluster 的 ID 。

4.1.1 中断优先级统一配置

通过下面数组统一配置各个中断优先级；对于没有在这里配置的中断，优先级为默认值（struct GIC_IRQ_AMP_CTRL 的 defPrio 项）。

```
static struct GIC_AMP_IRQ_INIT_CFG irqsConfig[] = {
    GIC_AMP_IRQ_CFG(100, 0x90),
    GIC_AMP_IRQ_CFG(101, 0xA0),
};
```

GIC_AMP_IRQ_CFG(100, 0x90)说明：

- 第一个参数100为中断的 ID 号码。

- 第二个参数0x90为中断的优先级。

4.2 初始化

通过下面函数进行 GIC 的初始化及通用功能配置：

```
HAL_GIC_Init(&irqConfig);
```

5. GPIO 中断分组功能

GPIO 中断分组概念：从一个 GPIO bank 对应的32个 PINs 中选择一部分 PINs 作为一组，将这一组 PINs 的中断以一定优先级映射到指定 CPU 上。将这样一组 PINs 称为一个 PIN group。

目前 SOC 的设计，一个 GPIO bank 对应的32个 pin 共用一个硬件中断（后面用 HWIRQ 表述)；通过 HWIRQ 进行中断分发。所以要支持将32个 PINs 的中断以不同 PIN group 对应到不同 CPU 上，需要软件再分发一次。分发的原理就是为一个 PIN group 分配一个 SOC 软件触发的分发中断-DIRQ(dispatching IRQ)。当一个 GPIO 收到 HWIRQ 后，检测到一个PIN group 上有中断需要响应，触发对应的 DIRQ。

HWIRQ 的使用：由于 HWIRQ 负责各个 PIN group 的中断分发，所以 HWIRQ 要分配最高的 GPIO 中断优先级，HWIRQ进行各个PIN group 中断分发后；需要处理对应到 HWIRQ 自己上的 PIN group（该PIN group也要对应最高优先级中断）中断分发。所以对于最高优先级的 GPIO 中断，一个 PIN group 的中断对应到 HWIRQ 进行分发要比 DIRQ 慢一些，因为 HWIRQ 要先进行各个 PIN group 的 DIRQ 分发。

配置上支持两种GPIO中断实现，不同GPIO BANK之间可以使用不同的实现；两种实现说明如下：

- GPIO_IRQ_GROUP_EN_BANK_TYPE：一个 GPIO bank 对应的32个 PIN 通过硬件中断（HWIRQ）进行分发，即不支持中断分组。本文称为普通（不分组）中断模式。
- GPIO_IRQ_GROUP_EN_GROUP_TYPE：中断分组模式。

5.1 基本配置

通过下面数据结构配置所有 GPIO BANKS 的中断支持；每个 GPIO BANK 对应一个 const struct GPIO_IRQ_GROUP_CFG 数据，用于管理这个 GPIO BANK 的中断实现。

```
static const struct GPIO_IRQ_GROUP_CFG gpioIrqCfg[GPIO_BANK_NUM] = {
    [GPIO_BANK0] = {
        GPIO_IRQ_GROUP_GPIO_CFG(GPIO_BANK0, GPIO0_IRQn, GPIO0),
        .groupIrqEn = GPIO_IRQ_GROUP_EN_BANK_TYPE,
        .bankTypeCfg = {
            .hwIrqCpuAff = CPU_GET_AFFINITY(1, 0),
            .prio = 0xd0,
        },
    },
    [GPIO_BANK1] = {
        GPIO_IRQ_GROUP_GPIO_CFG(GPIO_BANK1, GPIO1_IRQn, GPIO1),
```

```
        .groupIrqEn = GPIO_IRQ_GROUP_EN_BANK_TYPE,
        .bankTypeCfg = {
            .hwIrqCpuAff = CPU_GET_AFFINITY(1, 0),
            .prio = 0xd0,
        },
    },
    [GPIO_BANK2] = {
        GPIO_IRQ_GROUP_GPIO_CFG(GPIO_BANK2, GPIO2_IRQn, GPIO2),
        .groupIrqEn = GPIO_IRQ_GROUP_EN_BANK_TYPE,
        .bankTypeCfg = {
            .hwIrqCpuAff = CPU_GET_AFFINITY(1, 0),
            .prio = 0xd0,
        },
        .prioGroup[0] = {
            .cpuGroup = { LEVEL0_CPU0_GPIO_BITS, LEVEL0_CPU1_GPIO_BITS,
                          LEVEL0_CPU2_GPIO_BITS, LEVEL0_CPU3_GPIO_BITS },
            .GIRQId = { RSVD_IRQn(39), GPIO2_IRQn,
                        RSVD_IRQn(40), RSVD_IRQn(41) },
            .prio = 0x80,
        },

        .prioGroup[1] = {
            .cpuGroup = { LEVEL1_CPU0_GPIO_BITS, LEVEL1_CPU1_GPIO_BITS,
                          LEVEL1_CPU2_GPIO_BITS, LEVEL1_CPU3_GPIO_BITS },
            .GIRQId = { RSVD_IRQn(42), RSVD_IRQn(43),
                        RSVD_IRQn(44), RSVD_IRQn(45) },
            .prio = 0x90,
        },

        .prioGroup[2] = {
            .cpuGroup = { LEVEL2_CPU0_GPIO_BITS, LEVEL2_CPU1_GPIO_BITS,
                          LEVEL2_CPU2_GPIO_BITS, 0 },
            .GIRQId = { RSVD_IRQn(46), RSVD_IRQn(47),
                        RSVD_IRQn(48), RSVD_IRQn(49) },
            .prio = 0xa0,
        },
    },
    [GPIO_BANK3] = {
        GPIO_IRQ_GROUP_GPIO_CFG(GPIO_BANK3, GPIO3_IRQn, GPIO3),
        .groupIrqEn = GPIO_IRQ_GROUP_EN_BANK_TYPE,
        .bankTypeCfg = {
            .hwIrqCpuAff = CPU_GET_AFFINITY(1, 0),
            .prio = 0xd0,
        },
    },
};
```

5.1.1 GPIO 基本信息配置

每个struct GPIO_IRQ_GROUP_CFG 通过 GPIO_IRQ_GROUP_GPIO_CFG() 配置一个 GPIO 的基本信息，每个平台固定，不能修改。

GPIO BANK0 的 GPIO_IRQ_GROUP_GPIO_CFG(GPIO_BANK0, GPIO0_IRQn, GPIO0) 参数说明如下：

- 参数 GPIO_BANK0：GPIO BANK 的 ID。
- 参数 GPIO0_IRQn： 这个 GPIO BANK 的硬件中断。
- 参数 GPIO0： 这个 GPIO BANK 的基地址。

5.1.2 中断类型选择与使能

通过struct GPIO_IRQ_GROUP_CFG 的 groupIrqEn 项选择、使能一个 GPIO BANK 的中断类型，配置如下：

- GPIO_IRQ_GROUP_DISABLE：关闭这个 GPIO BANK 中断支持。
- GPIO_IRQ_GROUP_EN_BANK_TYPE：启动普通（不分组）中断模式。
- GPIO_IRQ_GROUP_EN_GROUP_TYPE： 启动中断分组模式。

5.1.3 普通（不分组）中断模式配置

通过struct GPIO_IRQ_GROUP_CFG 的 bankTypeCfg 项配置该模式下 GPIO 硬件中断（HWIRQ）属性。

```
.bankTypeCfg = {
.hwIrqCpuAff = CPU_GET_AFFINITY(1, 0),
.prio = 0xd0,
},
```

- 参数 hwIrqCpuAff：GPIO 硬件中断关联的 CPU。
- 参数 prio： GPIO 硬件中断对应的优先级。

5.1.4 分组配置

通过 struct GPIO_IRQ_GROUP_CFG 的 prioGroup[GROUP_PRIO_LEVEL_MAX] 项管理一个 GPIO BANK 中断分组。其中每一个 prioGroup[] 项管理相同优先级的一组 PINs；prioGroup[0] 管理的一组 PINs 对应的中断优先级最高。通过 prioGroup[0].cpuGroup[] 管理这个优先级下每一个 CPU 上对应的一组 PINs。如：通过 prioGroup[0].cpuGroup[1] 管理 CPU 1 上面的 PINS。

```
.prioGroup[0] = {
    .cpuGroup = { LEVEL0_CPU0_GPIO_BITS, LEVEL0_CPU1_GPIO_BITS,
                  LEVEL0_CPU2_GPIO_BITS, LEVEL0_CPU3_GPIO_BITS },
    .GIRQId = { RSVD_IRQn(39), GPIO2_IRQn,
                RSVD_IRQn(40), RSVD_IRQn(41) },
    .prio = 0x80,
},

.prioGroup[1] = {
    .cpuGroup = { LEVEL1_CPU0_GPIO_BITS, LEVEL1_CPU1_GPIO_BITS,
                  LEVEL1_CPU2_GPIO_BITS, LEVEL1_CPU3_GPIO_BITS },
    .GIRQId = { RSVD_IRQn(42), RSVD_IRQn(43),
                RSVD_IRQn(44), RSVD_IRQn(45) },
    .prio = 0x90,
```

```
},

.prioGroup[2] = {
    .cpuGroup = { LEVEL2_CPU0_GPIO_BITS, LEVEL2_CPU1_GPIO_BITS,
                  LEVEL2_CPU2_GPIO_BITS, 0 },
    .GIRQId = { RSVD_IRQn(46), RSVD_IRQn(47),
                RSVD_IRQn(48), RSVD_IRQn(49) },
    .prio = 0xa0,
},
```

- 参数 cpuGroup[PLATFORM_CORE_COUNT]：即该优先级下分配到每个CPU上的PINs对应的MASK BIT。如：

  ```
  prioGroup[0].cpuGroup[1] =  GPIO_PIN_A1 | GPIO_PIN_A2;
  ```

  表示 GPIO_PIN_A1 和 GPIO_PIN_A2 这两个 PINs 的中断被映射到CPU 1上面。对应的优先级为 prioGroup[0].prio。

- 参数 GIRQId[PLATFORM_CORE_COUNT]：每个 cpuGroup[] 项指定的一组 PINs 对应的分发中断（DIRQ）的中断 ID。如：prioGroup[0].cpuGroup[1] 的一组 PINs，对应的分发中断为 prioGroup[0].GIRQId[1]；对应的优先级为 prioGroup[0].prio。由于 GPIO 的物理中断(HWIRQ)也需要负责一组 PINs 的中断分发，并且对应的中断优先级最高；所以只能分配到 prioGroup[0] 对应的 GIRQId[]；这个例子中被分配到了prioGroup[0].GIRQId[1]。
- 参数 prio：这个 prioGroup 下每一组 PINs 对应的中断（GIRQId[]）优先级。非安全中断支持的通用优先级为：0x80，0x90，0xA0，0xB0，0xC0，0xD0，0xE0。

5.1.5 中断抢占嵌套

GIC 的 优先级抢占嵌套开启后；对应的 GPIO 中断行为如下：

- 普通（不分组）中断模式：根据每个 GPIO BANK 的物理中断(HWIRQ)优先级进行优先级抢占。
- GPIO 分组模式：根据每一组 PINs (PIN group) 对应的 DIRQ 的优先级进行优先级抢占。

5.2 功能能使

1. 启动中断分组功能需要定义下面宏：

```
#define HAL_GPIO_IRQ_GROUP_MODULE_ENABLED
# ifdef HAL_GPIO_IRQ_GROUP_MODULE_ENABLED
# define HAL_GPIO_IRQ_GROUP_PRIO_LEVEL_MAX (3)
# endif
```

- HAL_GPIO_IRQ_GROUP_PRIO_LEVEL_MAX：定义支持的 GPIO 中断优先级的最高级数。

2. 分发中断 DIRQ 分配

```
# define GPIO_IRQ_GROUP_DIRQ_BASE    RSVD_IRQn(37) /* gic irq: 320 */
# define GPIO_IRQ_GROUP_DIRQ_NUM     (NUM_INTERRUPTS - GPIO_IRQ_GROUP_DIRQ_BASE)
```

- 通过 GPIO_IRQ_GROUP_DIRQ_BASE 指定可以用于中断分发的分发中断（DIRQ）起始号码。

- 通过 GPIO_IRQ_GROUP_DIRQ_NUM 指定分配的分发中断数量。

5.3 初始化

通过下面函数进行初始化：

```
HAL_GPIO_IRQ_GROUP_Init(CPU_GET_AFFINITY(1, 0), gpioIrqCfg,
HAL_IRQ_HANDLER_GetGpioIrqGroupOps());
```

- 第一个参数：对于AMP系统，需要指定一个固定CPU进行基本功能初始化，指定的这个CPU必须是最先运行的。
- 第二个参数：GPIO 中断的配置信息。
- 第三个参数：不同的系统可能提供不同的的系统中断操作，比如只有 HAL 的系统，HAL 可以通过 HAL_GIC 及 HAL_IRQ_HANDLER 两个模块提供通用的中断控制及 IRQ handler 注册函数。如果使用 RT-Thread 系统，就会使用 RT-Thread 系统的中断控制操作，所以通过这个参数兼容不同的系统实现。如果是基于 HAL 中的HAL_GIC 及 HAL_IRQ_HANDLER 实现的系统中断控制，可以通HAL_IRQ_HANDLER_GetGpioIrqGroupOps()这个函数获取默认的ops。