Application Note

ZHCAG50 – Dec. 2025

R5F 中断在 TDA4x 处理器中的配置与调试

Linjun Meng, Tommy Song，Biao Li *Central FAE*

摘要

德州仪器的 TDA4x 处理器在高级辅助驾驶系统产品中有着广泛的应用。该处理器采用多核异构设计，包括 ARM® Cortex®-A72、ARM® Cortex® -R5F、C7x DSP、ARM® Cortex®-M4、图形、图像、视频以及网络安全加速器等。在此架构中，中断管理对低延迟实时处理非常重要。其中断路由配置是确保外设与各处理器单元协同工作的技术支撑。

TDA4x 中断系统采用分层级联结构，能够将数千个中断源分配到各个处理器。这种设计使得系统架构师可以根据产品实时性要求和功能安全需求，合理分配中断资源，灵活处理。在高级辅助驾驶系统应用中，快速响应传感器信号和实时控制决策对中断响应延迟有严格要求，因此深入理解 TDA4x 的中断路由机制对优化系统性能至关重要。

本应用笔记将全面介绍 TDA4x 的中断架构，考虑到大多数应用的实时信号都在 R5F 中处理， 所以对 R5F 中断的配置与调试会做比较详细的说明。本文旨在帮助客户在项目开发中充分顺利部署其功能，适用于系统规划，底层软件工程师以及硬件工程师。为用户提供从概念理解到系统部署实现全链路指导。

目录

图

表

1. TDA4x 中断系统简介

1.1 TDA4x 中断数据流

总体而言，TDA4x 系统的事件或信号，经过中断路由分发到中断管理模块，中断管理模块再将信号传递到处理器，触发其响应中断的过程。如下图所示：

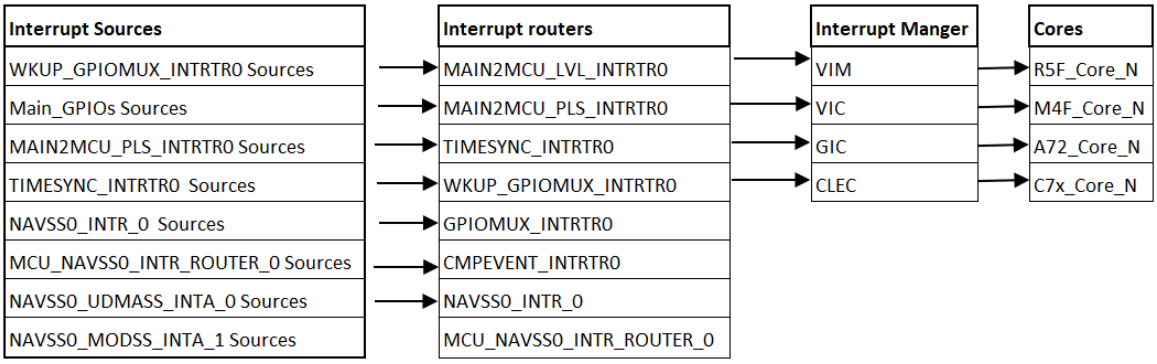

图 1 TDA4X 中断信号数据流

1.2 中断源

在 TDA4x 处理器中，中断源可以分成两类：信号和事件。 信号包括例如 GPIO 的上升沿或下降沿，高电平或者低电平。事件包括例如 DMA 在搬运过程中产生的异常事件或结束时产生的完成事件。具体中断信息，请参考 Chapter 5: SoC Family Specific Documentation — TISCI User Guide 的中断管理描述部分。

1.3 中断路由

表 1 中断路由实例及功能描述

中断路由实例	功能描述
J7X_DEV_MAIN2MCU_LVL_INTRTR0	将 Main Domain 电平信号中断源传递到 MCU Domain
J7X _DEV_MAIN2MCU_PLS_INTRTR0	将 Main Domain 脉冲信号中断源传递到 MCU Domain
J7X _DEV_TIMESYNC_INTRTR0	时间同步信号分配到 Main Domain
J7X _DEV_WKUP_GPIOMUX_INTRTR0	WKUP 域 GPIO 信号中断信号分配 MCU Domain 及 Main Domain。
J7X _DEV_GPIOMUX_INTRTR0	GPIO 信号路由到 Main Domain 及脉冲路由
J7X_ S4_DEV_CMPEVENT_INTRTR0	比较事件信号路由到 Main Domain
J7X _DEV_NAVSS0_INTR_0	NAVSS 事件信号路由到 Main Domain
J7X _DEV_MCU_NAVSS0_INTR_ROUTER_0	MCU 域 NAVSS 事件信号路由到 MCU Domain

1.4 中断管理单元

TDA4x 中 每个处理器核心，有独立的中断管理单元。整理归纳如下。R5F 的中断由 VIM 管理。 TDA4x 处理器有数个 R5F 核。 每个核都只能访问到自己的 VIM，其主域和 MCU 域的 VIM 内存映射地址不同。调试时需要根据 TRM memory map 查看 VIM 所在的基地址。 A72 的中断由 GIC 管理，C7X 的中断由 CLEC 管理，M4F 的中断由 NVIC 管理。

2. 中断路由配置

2.1. 中断路由资源分配依据

中断路由的资源分配依据主要是根据物理连接来确定路由分配，这里以 TDA4VH 的 GPIOMUX_INTR0 为例，说明 TDA4VH 中断路由连接情况，详情请查看 TISCI。其他的型号比如 TDA4VM，名称是根据内部核心数命名，所以稍有差异，本质相同，在此不再一一赘叙。

表 2 GPIOMUX 中断端点连接表

IR Name	IR Device ID	IR Output Index	Destination Name	Destination Interface	Destination Index
J784S4_DEV_GPIOMUX_INTRTR0	10	0	J784S4_DEV_ESM0	esm_pls_event0	664
			J784S4_DEV_ESM0	esm_pls_event1	672
J784S4_DEV_GPIOMUX_INTRTR0	10	0	J784S4_DEV_ESM0	esm_pls_event2	680
			J784S4_DEV_MAIN2MCU_PLS_INTRTR0	in	64
J784S4_DEV_GPIOMUX_INTRTR0	10	0	J784S4_DEV_R5FSS0_CORE0	intr	396
			J784S4_DEV_R5FSS0_CORE1	intr	396
J784S4_DEV_GPIOMUX_INTRTR0	10	0	J784S4_DEV_R5FSS1_CORE0	intr	396
			J784S4_DEV_R5FSS1_CORE1	intr	396
J784S4_DEV_GPIOMUX_INTRTR0	10	0	J784S4_DEV_R5FSS2_CORE0	intr	396
			J784S4_DEV_R5FSS2_CORE1	intr	396

上图所示是 TDA4VH 的 GPIOMUX_INTR0 索引 0 的连接情况。这里有三条信息需要说明： 根据中断的目的名称判断中断路由可以到达的端点；没有列出的终端名称即此中断路由不可以到达的端点；
MAIN2MCU_PLS_INTRTR0 是主域到 MCU 域的脉冲路由。中断路由资源的分配在 RTOS 的 SDK sciclient_defaultBoardcfg_rm.c 文件中。

以 TISCI_DEV_GPIOMUX_INTRTR0 为例：

```
{
   .num_resource = 4,   /* 分配给此核心的资源数目 */
   .type = TISCI_RESASG_UTYPE(TISCI_DEV_GPIOMUX_INTRTR0,
                              TISCI_RESASG_SUBTYPE_IR_OUTPUT),
   .start_resource = 0,  /* 中断分配起始索引 */
   .host_id = TISCI_HOST_ID_MAIN_0_R5_0, /* 资源分配的目的处理器标识 */
},
```

用户也可以根据项目需要，利用 SYSCONFIG 进行分配。

2.2. 应用层对资源的申请

以 GPIOMUX_INTRTR0_CFG 为例，说明其配置的底层寄存器原理。其地址 offset 为 4h+ (j * 4h); j 从 0 to 63，基地址是 0x00A0 0000，也就是说 GPIOMUX_INTRTR0 有 64 个通道。寄存器结构如下：
GPIOMUX_INTRTR0_CFG_muxcntl[bit16]: 中断使能位，置 1 使能 GPIOMUX_INTRTR0_CFG_muxcntl[bit7-0]: MUX_CNTL， 也就是选择的中断源的编号。

2.3. 中断配置示例

例一 ：将 WKUP_GPIO0_61 配置到 WKUP_GPIOMUX_INTRTR0 输出端口 1 上。函数实现如下：

```
/**
* @brief 配置 WKUP_GPIO0_61 中断路由
 * @return 配置状态：CSL_PASS 为成功，其他为失败
 */
static int32_t WKUP_GPIO0_61_IntR_cfg(void)
{
   int32_t status = CSL_PASS;
   struct tisci_msg_rm_irq_set_req rmIrqReq;
   struct tisci_msg_rm_irq_set_resp rmIrqResp;

   /* 初始化请求和响应结构体 */
   memset(&rmIrqReq, 0x0, sizeof(rmIrqReq));
   memset(&rmIrqResp, 0x0, sizeof(rmIrqResp));

   /* 配置中断路由参数 */
   rmIrqReq.secondary_host = TISCI_MSG_VALUE_RM_UNUSED_SECONDARY_HOST;
   rmIrqReq.src_id = TISCI_DEV_WKUP_GPIOMUX_INTRTR0;  /* 源 ID: 125 */
   rmIrqReq.src_index = 61;                    /* GPIO0_61 */
   rmIrqReq.dst_id = TISCI_DEV_WKUP_GPIOMUX_INTRTR0;
   rmIrqReq.dst_host_irq = 1;                   /* 输出端口 1 */

   /* 设置有效参数标志 */
   rmIrqReq.valid_params |= TISCI_MSG_VALUE_RM_DST_ID_VALID;
   rmIrqReq.valid_params |= TISCI_MSG_VALUE_RM_DST_HOST_IRQ_VALID;

   /* 执行中断路由配置 */
   status = Sciclient_rmIrqSetRaw((const struct tisci_msg_rm_irq_set_req *)&rmIrqReq,
                        &rmIrqResp,
                        SCICLIENT_SERVICE_WAIT_FOREVER);

   if(status == CSL_PASS)
   {
      UART_printf("WKUP_GPIO0_61 中断路由配置成功\n");
   }
   else
   {
      UART_printf("WKUP_GPIO0_61 中断路由配置失败，错误码: %d\n", status);
   }

   return status;
}
```

例二：将主域 GPIO0_61 路由到 MCU1_0 核上。

```
/**  @brief 配置主域 GPIO0_61 到 MCU 域的跨域中断路由
 * @return 配置状态：CSL_PASS 为成功，其他为失败*/
static int32_t GPIO0_61_CrossDomain_IntR_cfg(void)
{
    int32_t status = CSL_PASS;
    struct tisci_msg_rm_irq_set_req rmIrqReq;
    struct tisci_msg_rm_irq_set_resp rmIrqResp;
      /* 第一级路由：GPIOMUX_INTRTR0 */
    memset(&rmIrqReq, 0x0, sizeof(rmIrqReq));
    memset(&rmIrqResp, 0x0, sizeof(rmIrqResp));
      rmIrqReq.secondary_host = TISCI_MSG_VALUE_RM_UNUSED_SECONDARY_HOST;
    rmIrqReq.src_id = TISCI_DEV_GPIOMUX_INTRTR0;
    rmIrqReq.src_index = 61;          /* GPIO0_61 */
    rmIrqReq.dst_id = TISCI_DEV_GPIOMUX_INTRTR0;
    rmIrqReq.dst_host_irq = 16;       /* 输出到索引 16 */
      rmIrqReq.valid_params |= TISCI_MSG_VALUE_RM_DST_ID_VALID;
    rmIrqReq.valid_params |= TISCI_MSG_VALUE_RM_DST_HOST_IRQ_VALID;

    status = Sciclient_rmIrqSetRaw((const struct tisci_msg_rm_irq_set_req *)&rmIrqReq,
                        &rmIrqResp,
                        SCICLIENT_SERVICE_WAIT_FOREVER);
      if(status == CSL_PASS)
    {
      /* 第二级路由：MAIN2MCU_PLS_INTRTR0 */
      memset(&rmIrqReq, 0x0, sizeof(rmIrqReq));
      memset(&rmIrqResp, 0x0, sizeof(rmIrqResp));
            rmIrqReq.secondary_host = TISCI_MSG_VALUE_RM_UNUSED_SECONDARY_HOST;
      rmIrqReq.src_id = TISCI_DEV_MAIN2MCU_PLS_INTRTR0;
      rmIrqReq.src_index = 16;        /* 来自第一级路由的输出 */
      rmIrqReq.dst_id = TISCI_DEV_MAIN2MCU_PLS_INTRTR0;
      rmIrqReq.dst_host_irq = 0;      /* MCU 域输出索引 0 */
            rmIrqReq.valid_params |= TISCI_MSG_VALUE_RM_DST_ID_VALID;
      rmIrqReq.valid_params |= TISCI_MSG_VALUE_RM_DST_HOST_IRQ_VALID;
            status = Sciclient_rmIrqSetRaw((const struct tisci_msg_rm_irq_set_req *)&rmIrqReq,
                          &rmIrqResp,
                          SCICLIENT_SERVICE_WAIT_FOREVER);
            if(status == CSL_PASS)
      {
        UART_printf("跨域 GPIO 中断路由配置成功\n");
        UART_printf("MCU 域中断号: 224\n");
      }
      else
      {
        UART_printf("第二级路由配置失败，错误码: %d\n", status);
      }
    }
    else
    {
      UART_printf("第一级路由配置失败，错误码: %d\n", status);
    }

    return status;
}
```

此例中主域的 GPIO0_61 需要经过两级路由：一级路由是 GPIOMUX_INTR0，将 GPI00_61 配置到 GIIPMUX_INTR0_16 的一个输出。二级路由是 MAIN2MCU_PLS_INTRTR0 将 GPIOMUX_INTR0_16 配置到 MAIN2MCU_PLS_INTRTR0_0。根据 J784S4 Interrupt Management Device Descriptions — TISCI User Guide 可以看到，MAIN2MCU_PLS_INTRTR0_0 连接到了 J784S4_DEV_MCU_R5FSS0_CORE0 和 J784S4_DEV_MCU_R5FSS0_CORE1，对应核的中断号是 224。使能 224 中断，从主域到 MCU 域的 GPIO 路由就配置完成。

2.4. R5F 中断函数及其说明

表 3 OSAL 中断函数接口说明

中断接口函数	描述
Osal_RegisterInterrupt_initParams	初始化中断处理的数据结构
Osal_RegisterInterrupt	注册中断服务函数
Osal_RegisterInterruptDirect	中断初始化及服务函数一起进行
Osal_EnableInterrupt	中断使能
Osal_DisableInterrupt	中断关闭
Osal_ClearInterrupt	中断状态清除

使用示例：

```
/* 中断服务函数 */
void gpio_isr(uintptr_t arg)
{
   /* 处理 GPIO 中断 */
   GPIO_clearInt(GPIO_LED_BASE_ADDR, GPIO_LED_PIN_NUM);

   /* 用户中断处理逻辑 */
   // ...
}
/* 中断注册和配置 */
HwiP_Params hwiParams;
HwiP_Params_init(&hwiParams);
hwiParams.arg = (uintptr_t)NULL;
hwiParams.priority = 15;  /* 设置中断优先级 */
HwiP_Handle hwiHandle = HwiP_create(224, gpio_isr, &hwiParams);
if(hwiHandle == NULL)
{
   /* 中断注册失败处理 */
   UART_printf("中断注册失败\n");
}
```

具体应用实例请参考 PDK 的例子 osal_extended_testapp_freertos。

3. 中断故障排查要点

3.1. 中断路由配置检查

检查项目：

- 验证中断源输入配置是否正确

- 确认中断路由通道是否已分配给目标 CPU
- 检查多级路由的每一级配置是否正确
- 验证 TISCI 配置参数是否有效

调试方法示例：

```
/* 检查中断路由寄存器状态 */
uint32_t regVal = CSL_REG32_RD(GPIOMUX_INTRTR0_BASE +
                  GPIOMUX_INTRTR0_CFG_OFFSET +
                  (channel * 4));
UART_printf("路由通道%d 配置: 0x%08x\n", channel, regVal);
```

3.2. 中断系统配置验证

检查项目：

- 确认中断号是否正确注册
- 检查中断是否已使能
- 验证中断状态是否正确触发
- 确认 VIM 基地址映射是否正确

调试方法示例：

```
  /* 检查 VIM 中断状态 */
uint32_t intStatus = CSL_vimGetIntrType(vimBaseAddr, intrNum);
uint32_t pendStatus = CSL_vimGetIntrPending(vimBaseAddr, intrNum);
UART_printf("中断%d 状态: 类型=%d, 挂起=%d\n",
        intrNum, intStatus, pendStatus);
```

3.3. GPIO 相关配置检查

检查项目：

- 验证管脚复用功能是否正确配置为 GPIO
- 确认 GPIO 方向是否设置为输入模式
- 检查 GPIO 中断触发条件配置
- 验证 GPIO 时钟是否正确使能

调试方法示例：

```
/* 检查 GPIO 配置状态 */
uint32_t gpioDir = GPIO_getDir(gpioBaseAddr, gpioPin);
uint32_t gpioIntStatus = GPIO_getIntrStatus(gpioBaseAddr, gpioPin);
UART_printf("GPIO%d: 方向=%s, 中断状态=%d\n",
        gpioPin,
        (gpioDir == GPIO_DIRECTION_INPUT) ? "输入" : "输出",
        gpioIntStatus);
```

3.4. 常用调试工具和技巧

CCS 调试：

- 使用寄存器视图查看 VIM 状态寄存器
- 通过内存浏览器检查中断路由器配置
- 设置断点验证中断服务函数是否被调用

日志调试示例：

```
/* 中断流程跟踪宏 */
#define INT_DEBUG_PRINT(fmt, ...) \
   UART_printf("[INT_DEBUG] " fmt, ##__VA_ARGS__)
/* 在关键点添加调试信息 */
INT_DEBUG_PRINT("配置中断路由开始\n");
INT_DEBUG_PRINT("中断服务函数被调用\n");
```

4. 性能优化与实践

4.1. 中断优先级分配策略

优先级分配原则：实时性要求高的中断设置高优先级；频率高的中断适当降低优先级避免系统阻塞；预留部分优先级用于紧急事件处理。

以下是推荐配置示例：

```
/* 高优先级：安全关键中断 */
#define SAFETY_CRITICAL_INT_PRIORITY   1
/* 中等优先级：实时控制中断 */
#define REALTIME_CONTROL_INT_PRIORITY  8
/* 低优先级：通用 I/O 中断 */
#define GENERAL_IO_INT_PRIORITY       15
```

4.2. 中断延迟优化

减少中断延迟的方法：使用 VIM 的硬件优先级排序功能；优化中断服务函数，避免长时间处理；将耗时操作移至任务层处理。

多核负载均衡的中断分配策略：根据各 CPU 的负载情况分配中断；相关中断分配到同一核心减少核间通信；使用中断亲和性绑定提高 cache 命中率。

5. 总结

通过本应用笔记的详细阐述，开发者能够：深入理解 TDA4x 的分层中断架构和数据流；掌握中断路由的配置方法和寄存器操作；熟练运用 R5F 中断系统的接口函数；有效排查 中断相关的常见问题；优化中断系统性能和实时响应能力。

正确配置和优化 TDA4 的中断系统，能够充分发挥其多核异构架构的优势，为高级辅助驾驶系统提供可靠的实时处理能力。

6. 参考文献

1. J784S4 J742S2 Technical Reference Manual (Rev. E)
2. J784S4 Interrupt Management Device Descriptions — TISCI User Guide
3. Processor SDK RTOS J784S4 — Processor SDK RTOS J784S4
4. 5.4. OSAL — Platform Development Kit (PDK) - J784S4 User Guide

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月