Rockchip Developer Guide PCIe Performance

文件标识：RK-KF-YF-460

发布版本：V2.0.0

日期：2023-08-03

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

本文介绍 PCIe 各项主要性能测试方案及 RK 测试结果。

产品版本

芯片名称	内核版本
所有支持 PCIe 的芯片	Linux 4.4 及以上

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	林鼎强	2022-05-23	初始版本
V2.0.0	林鼎强	2023-08-03	增加 RK3528/RK3562 数据、增加中断响应速率、修改 DMA 速率测试方案为标准 EP 测试、增加通用性能优化方向章节、优化 Bar Interface 吞吐率测试

目录

1. 背景

1.1 简介

本文介绍 RK PCIe 应用相关的性能指标及其测试方法，测试目标为通过简单、准确、可控的应用接口来获取对应的性能指标。

其中 “背景” 章节主要介绍测试过程中涉及到的部分知识背景，“传输速率简介” 章节为 PCIe 理论计算数据，其余章节为具体性能指标及其测试方法。

1.2 PCIe sysfs

Linux PCIe 支持 sysfs 接口，主要提供以下资源：

```
/sys/devices/pci0000:17
|-- 0000:17:00.0
|   |-- class
|   |-- config
|   |-- device
|   |-- enable
|   |-- irq
|   |-- local_cpus
|   |-- remove
|   |-- resource
|   |-- resource0
|   |-- resource1
|   |-- resource2
|   |-- resource2_wc        # 64bits-pref mem resource 并支持写缓冲，对于 PCIe mem
to dev 操作有优化效果。
|   |-- revision
|   |-- rom
|   |-- subsystem_device
|   |-- subsystem_vendor
|   `-- vendor
`-- ...
```

详细参考：

2. 传输速率简介

2.1 PCIe 带宽

PCIe Generation	编码方案	传输速率	x1 Lane	x2 Lane	x4 Lane	x8 Lane
1.0	8b/10b	2.5GT/s	250MB/s	500MB/s	1GB/s	2GB/s
2.0	8b/10b	5GT/s	500MB/s	1GB/s	2GB/s	4GB/s
3.0	128b/130b	8GT/s	984.6MB/s	1.969GB/s	3.938GB/s	7.877GB/s
4.0	128b/130b	16GT/s	1.969GB/s	3.938GB/s	7.877GB/s	15.754GB/s

传输速率

传输速率为每秒传输量GT/s，而不是每秒位数Gbps，因为传输量包括不提供额外吞吐量的开销位； 比如PCIe 1.x和PCIe 2.x使用8b / 10b编码方案，导致占用了20% （2/10）的原始信道带宽。

GT/s： Giga transation per second （千兆传输/秒），即每一秒内传输的次数

Gbps： Giga Bits Per Second （千兆位/秒）。GT/s 与Gbps 之间不存在成比例的换算关系

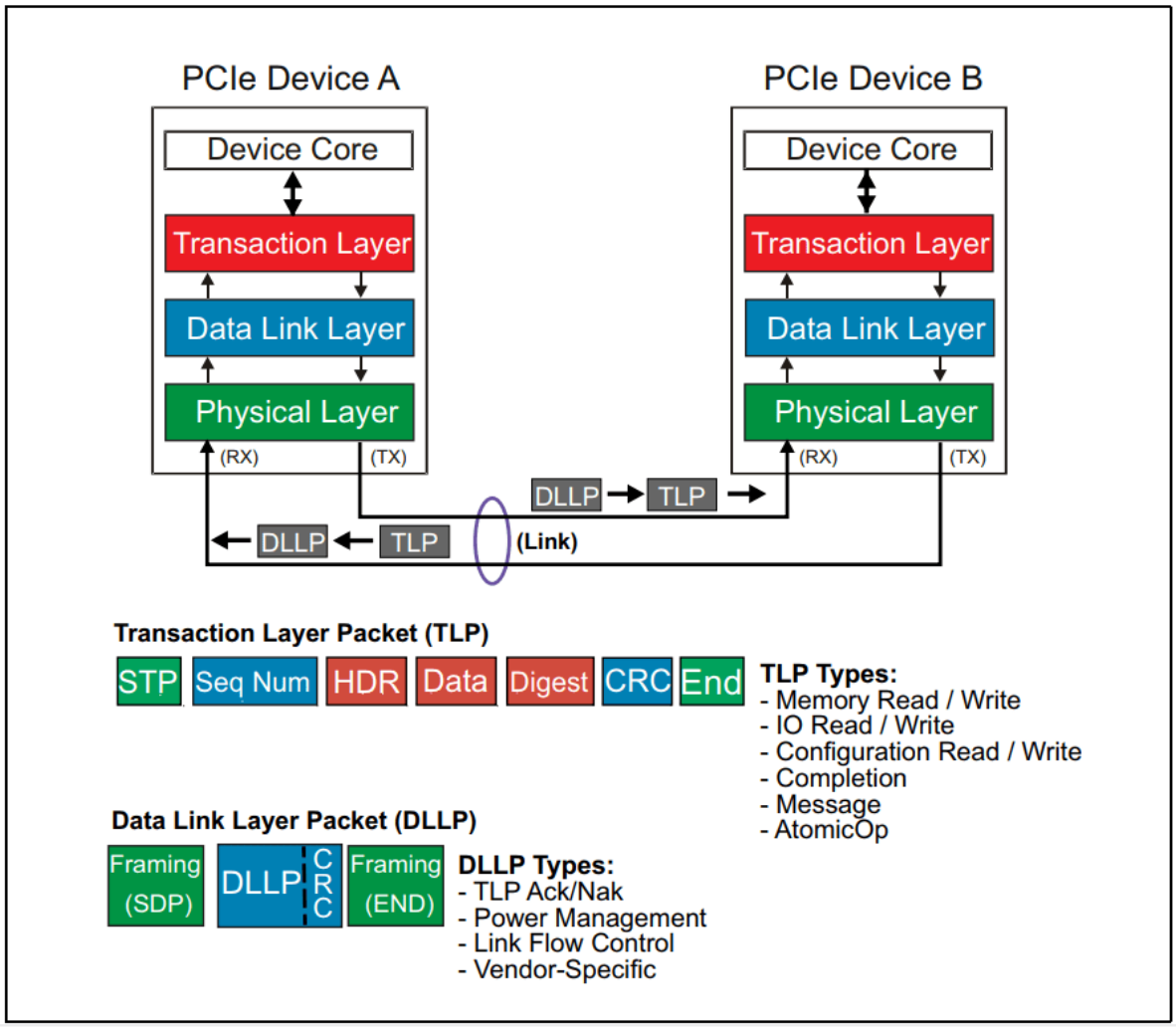
PCIe 可用带宽

吞吐量 = 传输速率 * 编码方案

例如：PCIe 2.0 协议的每一条 Lane 支持 5*8 / 10 = 4 Gbps = 500 MB/s 的速率，PCIe 2.0 x8的通道为例，x8 的可用带宽为 4*8 = 32 Gbps = 4 GB/s。

2.2 PCIe TLP 损耗

由于数据经过 Transaction Layer 和 Data Link Layer 后会添加打包信息，例如数据包的包头、包尾（ LCRC 和 ECRC 等）、数据链路层添加的包编号等等，所以 Data Payload 的真实速率相较于带宽会更低。



28 Byte TLP overhead 是冗余最大的传输包，包含 the header (16 bytes), the optional ECRC (4 bytes), the LCRC (4 bytes), the sequence number (2 bytes) and the framing Symbols STP and END (2 bytes).

所以以 Gen3 PCIe 带宽为例：

PCIe Generation	编码方案	传输速率	x1 Lane	x2 Lane	x4 Lane	x8 Lane
3.0	128b/130b	8GT/s	984.6MB/s	1.969GB/s	3.938GB/s	7.877GB/s

计入 TLP overhead 后：

PCIe Generation Gen3	x1 Lane	x2 Lane	x4 Lane	x8 Lane
PCIe 带宽（未计入 TLP overhead）	984.6MB/s	1.969GB/s	3.938GB/s	7.877GB/s
256B payload 扣除 TLP 损耗（256/284）	887.5MB/s	1.775GB/s	3.550GB/s	7.1002GB/s
128B payload 扣除 TLP 损耗（128/156）	807.8MB/s	1.616GB/s	3.232GB/s	6.464GB/s

2.3 PCIe 应用场景传输速率

如果再考虑用于 Ack/Nak 和 Flow Control 等的 DLLP 和用于链路训练和 Skip 的 Order Sets 等不定因素对速率的影响，实际真实数据传输速率会更低。

具体应用场景下以 eDMA 传输为最优解。

3. 通用性能优化方向

3.1 CPU 和 DDR 定高频

请参考 DVFS 及 DRAM 相关源码做调整。

3.2 PCIe RC MPS 策略

请参考 Rockchip_Developer_Guide_PCIe_CN.pdf 文档 “如何配置max payload size ” 章节，可尝试配置为 pci=pcie_bus_perf 配置，开启后确认所设定的 MPS 为所有接入的 PCIe 设备都支持的值。

3.3 PCIe 控制器总线 QoS 调节

PCIe 传输尤其是 DMA 传输，当吞吐率提升后会占用 PCIe 控制器到 DRAM 的总线带宽资源，存在和其他 IP 的总线竞争仲裁，如果出现该极端情况，可通过修改 PCIe 控制器总线 QoS 来提高其优先级。调整方法参考 Rockchip_Developer_Guide_PCIe_CN.pdf 文档 “PCIe设备性能抖动” 章节说明。

3.4 PCIe PM 调节

PCIe PM 管理对传输性能有一定的影响，所以测试时建议关闭 PM 管理，主要包括：

- ASPM

实际产品可以在 EP function 驱动中构建 QoS 场景，动态开关 PM 支持。

4. 控制器 - DMA Interface 吞吐率

4.1 理论速率

DMA 传输速率理论上接近 "PCIe TLP 损耗" 计入后的速率，理论值参考 “PCIe TLP 损耗” 章节说明。

4.2 测试方法

4.2.1 标准 EP 测试

测试基于 “标准EP功能件开发” + “内核 DMATEST” 测试，详细内容参考《Rockchip_Developer_Guide_PCIe_CN.pdf》相关章节，先实现标准 EP 开发，后开启 EP 端的 PCIe 内核 DMATEST 测试。

4.2.2 其他参数配置

1. CPU 和 DDR 定高频
2. PCIe RC MPS 策略设置为 pci=pcie_bus_perf
3. 为了提高实时性，准确测试 DMA 带宽，使用 PIO 轮询 DMA 完成状态

```
diff --git a/drivers/pci/controller/dwc/pcie-dw-dmatest.c
b/drivers/pci/controller/dwc/pcie-dw-dmatest.c
index 4b0d4d3e01c8..df4595af962e 100644
--- a/drivers/pci/controller/dwc/pcie-dw-dmatest.c
+++ b/drivers/pci/controller/dwc/pcie-dw-dmatest.c
@@ -231,7 +231,7 @@ struct dma_trx_obj *pcie_dw_dmatest_register(struct device
*dev, bool irq_en)
        }

        /* Enable IRQ transfer as default */
-       dmatest_dev->irq_en = irq_en;
+       dmatest_dev->irq_en = false;
        cur_dmatest_dev++;

        return obj;
```

4.3 测试结果

4.3.1 RK3568 - 标准 EP 测试

测试设备

RC：RK3568-EVB1-DDR4-V10 PCIe 3.0 2 lanes ARM：1.99G DDR：1.5G

EP：RK3568-IOTEST-DDR3-V10 PCIe 3.0 2 lanes ARM: 1.99G DDR：1.0G

测试结果

DMA 大小	64B	256B	4K	64KB	1MB	4MB
写	12.2MB/s	51.7MB/s	529.1MB/s	941.6MB/s	1418.2MB/s	1454.3MB/s
读	6.0MB/s	44.3MB/s	442.3MB/s	680.5MB/s	938.9MB/s	1080.2MB/s

4.3.2 RK3588 - 标准 EP 测试

测试设备

RC：RK3588-EVB1-LP4X-V10 PCIe 3.0 4 lanes 大核 2.4G 小核 1.8G

EP：RK3588-EVB4-LP4X-V10 PCIe 3.0 4 lanes 大核 2.4G 小核 1.8G

测试结果

DMA 大小	64B	256B	4K	64KB	1MB	4MB
写	14.7MB/S	60.9MB/S	660.6MB/S	2313.4MB/S	3026.1MB/S	3066.1MB/s
读	13.9MB/S	52.6MB/S	568.7MB/S	1490.5MB/S	1619.8MB/S	1660.6MB/s

5. 控制器 - Bar Interface 吞吐率

5.1 简介

通常用户最常用的 Bar 访问行为为 Load/Store 32bits 指令，所以该指标以"用户态 Load/Store 32bits 指令"为测试方案，通过 Linux 提供的 PCIe sysfs 的 pci_mmap_resource 接口做读写测试。

5.2 测试 APP

详细参考附录 “resource_mmap_test.c” 源码，要求确认源码内以下参数：

- 选择 device memory 属性的 mmap Resource，例如：resource2

APP 编译方式参考，以 RK3588 Android 为例：

```
/home1/ldq/rk-linux/prebuilts/gcc/linux-x86/aarch64/gcc-arm-10.2-2020.11-x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu-gcc -mcpu=cortex-a76 -O0 -static -DROPT -o resource_mmap_test resource_mmap_test.c
```

说明：

- 要求使用 -O0 编译优化等级以避免读写命令被优化

5.3 测试结果

RC	EP	Store 32bits	Load 32bits
RK3568 PCIe 3.0 2 lanes	RK3588 PCIe 3.0 4 lanes 大核 2.4G 小核 1.8G	52.5MB/s	3.7MB/s
RK3568 PCIe 2.0 1 lane	RK3588 PCIe 3.0 4 lanes 大核 2.4G 小核 1.8G	36.0MB/s	2.4MB/s
RK3588 PCIe 3.0 4 lanes	RK3588 PCIe 3.0 4 lanes 大核 2.4G 小核 1.8G	53.3MB/s	4.2MB/s
RK3588 PCIe 2.0 1 lane	RK3588 PCIe 3.0 4 lanes 大核 2.4G 小核 1.8G	26.7MB/s	2.6MB/s
RK3528 PCIe 2.0 1 lane	RK3588 PCIe 3.0 4 lanes 大核 2.4G 小核 1.8G	67.7MB/s	2.3MB/s
RK3562 PCIe 2.0 1 lane	RK3588 PCIe 3.0 4 lanes 大核 2.4G 小核 1.8G	66.3MB/s	2.3MB/s

说明:

- 64bits-pref (resource0..N) 与 32bits-np (resource0..N) 测试结果一致

5.4 性能优化方向

5.4.1 提高数据位宽

简介

PCIe Bar 读写速率的瓶颈主要在于 PCIe TLP 数量，而提高 TLP 有效数据负载能降低 TLP 数量，也就是提高总线的 Load/Store 请求的数据位宽能减少总的 TLP 数量，常见有以下几种用户访问 PCIe Bar 的行为:

- 用户态 Load/Store 数据位宽 32bits，实际以汇编结果为准，例如:

```
str    w1, [x0]
```

- 用户态 Load/Store SIMD 数据位宽 128bits
  - 通常为编译器优化后的行为，例如支持 SIMD 的 SOC 选用 -O3 编译优化选项就可能实现，实际以汇编结果为准，例如:

```
str    q1, [x1], #16
```

- 内核态 Load/Store 数据位宽 32bits
- 内核态 memset_io/memcpy_fromio 接口

测试结果

以 RK3588 为例:

访问行为	Store（MB/s）	Load（MB/s）
用户态 Load/Store 32bits	53	4
用户态 Load/Store SIMD 128bits	203	16
内核态 Load/Store 32bits	53	4
内核态 memset_io/memcpy_fromio	106	7

说明：

- 测试环境：
  - RC：RK3588-EVB1-LP4X-V10 PCIe 3.0 4 lanes 大核 2.4G 小核 1.8G
  - EP：RK3588-EVB4-LP4X-V10 PCIe 3.0 4 lanes 大核 2.4G 小核 1.8G
- 用户态 Load/Store 32bits 访问行为，汇编 V8-a 指令参考：

```
LDR <Wt>, [<Xn|SP>, (<Wm>|<Xm>){, <extend> {<amount>}}]
STR <Wt>, [<Xn|SP>, (<Wm>|<Xm>){, <extend> {<amount>}}]
```

- 用户态 Load/Store SIMD 128bit 访问行为，汇编 V8-a 指令参考：

```
LDR <Qt>, [<Xn|SP>, (<Wm>|<Xm>){, <extend> {<amount>}}]
STR <Qt>, [<Xn|SP>, (<Wm>|<Xm>){, <extend> {<amount>}}]
```

- 内核态 Load/Store 32bits

5.4.2 多线程-无效

多个 resource_mmap_test.c 同时运行，读写不同 Bar，通过逻辑分析仪确认，总线利用率没有提高。

对于实际带宽没有显著提升（不超过 10%）,带宽利用率也没有提升，但是确实看到 TLP 不同 Bar 空间交替的情况。

6. 控制器 - 中断响应速率-ELBI

6.1 简介

DWC PCIe EP 模式下的 ELBI（Endpoint Local Interrupt Bypass）通过EP（Endpoint）的配置空间映射来实现。RC 通过写 EP 的配置空间中的 ELBI 映射寄存器来触发 EP 的 ELBI 中断，从而提高 EP 系统的响应性。

ELBI 中断响应速率主要取决于 EP 系统总线带宽、设备驱动程序的优化程度以及处理器的处理能力等因素。

6.2 测试 APP

PCIe ELBI RC 端发包代码

```
#include <linux/kthread.h>

// 创建一个新的内核线程来触发ELBI中断
static int pcie_rkep_elbi_test_real(void *p)
{
    struct pcie_rkep *pcie_rkep = (struct pcie_rkep *)p;

    // 在一个无限循环中，调用rockchip_pcie_raise_elbi_irq函数触发ELBI中断（使用IRQ号0）
    while(1)
        rockchip_pcie_raise_elbi_irq(pcie_rkep, 0);
}

// 使用pcie_rkep_elbi_test_real函数创建一个内核线程来触发ELBI中断
static int pcie_rkep_elbi_test(struct pcie_rkep *pcie_rkep)
{
    struct task_struct *tsk;

    // 使用pcie_rkep_test函数执行一些测试操作
    pcie_rkep_test(pcie_rkep);

    // 创建一个名为"rkep-elbi"的内核线程，并将pcie_rkep作为参数传递给
pcie_rkep_elbi_test_real函数
    tsk = kthread_run(pcie_rkep_elbi_test_real, pcie_rkep, "rkep-elbi");
    if (IS_ERR(tsk)) {
        pr_err("rkep elbi start rk-pcie thread failed\n");
        return PTR_ERR(tsk);
    }

    return 0;
}
```

PCIe ELBI RC 端发包代码集成

```
 static int pcie_rkep_probe(struct pci_dev *pdev, const struct pci_device_id *id)
 {
        int ret, i;
@@ -774,6 +812,7 @@ static int pcie_rkep_probe(struct pci_dev *pdev, const struct
pci_device_id *id)
                pcie_rkep->obj_info->version);

        device_create_file(&pdev->dev, &dev_attr_rkep);
+       pcie_rkep_elbi_test(pcie_rkep);

        return 0;
 err_register_obj:
```

PCIe ELBI EP 端测速代码

```
#include <linux/kthread.h>

// 创建一个新的内核线程来测量ELBI中断的触发频率
```

```c
static int pcie_rkep_elbi_test_real(void *p)
{
    struct rockchip_pcie *rockchip = (struct rockchip_pcie *)p;
    int cur_count, last_count = rockchip->elbi_count;
    u64 us = 0;

    // 在一个无限循环中，每隔1秒执行以下操作
    while(1) {
        msleep(1000); // 等待1秒

        // 获取当前的ELBI中断计数
        cur_count = rockchip->elbi_count;

        // 如果当前中断计数不为0，则计算上一次和当前的中断计数之间的时间间隔（单位：微秒）
        if (cur_count) {
            us = 1000000 / (cur_count - last_count);
        }

        // 打印当前的ELBI中断计数和触发频率
        pr_err("elbi count=%d %lluus\n", cur_count, us);

        // 更新上一次的中断计数
        last_count = rockchip->elbi_count;
    }
}

// 使用pcie_rkep_elbi_test_real函数创建一个内核线程来测量ELBI中断的触发频率
static __maybe_unused int pcie_rkep_elbi_test(struct rockchip_pcie *rockchip)
{
    struct task_struct *tsk;

    // 创建一个名为"rkep-elbi"的内核线程，并将rockchip作为参数传递给
pcie_rkep_elbi_test_real函数
    tsk = kthread_run(pcie_rkep_elbi_test_real, rockchip, "rkep-elbi");
    if (IS_ERR(tsk)) {
        pr_err("rkep elbi start rk-pcie thread failed\n");
        return PTR_ERR(tsk);
    }

    return 0;
}
```

PCIe ELBI EP 端测速代码集成

```
 static int rockchip_pcie_ep_probe(struct platform_device *pdev)
 {
        struct device *dev = &pdev->dev;
@@ -1276,6 +1316,7 @@ static int rockchip_pcie_ep_probe(struct platform_device
*pdev)
                goto deinit_phy;

        rockchip_pcie_add_misc(rockchip);
+       pcie_rkep_elbi_test(rockchip);

        return 0;
```

6.3 测试结果

RC	EP	默认延迟 (us)
RK3568-EVB1-DDR4-V10 PCIe 3.0 2 lanes ARM：1.99G DDR：1.5G	RK3568-IOTEST-DDR3-V10 PCIe 3.0 2 lanes ARM: 1.99G DDR：1.0G	6
RK3588-EVB1-LP4X-V10 PCIe 3.0 4 lanes 大核 2.4G 小核 1.8G	RK3588-EVB4-LP4X-V10 PCIe 3.0 4 lanes 大核 2.4G 小核 1.8G	6

说明：

- 请注意，具体的实现和应用场景可能会因系统配置、硬件平台和驱动要求而有所不同。

6.4 性能优化方向

ELBI 中断响应速率主要取决于 EP 系统总线带宽、设备驱动程序的优化程度以及处理器的处理能力等因素。

以 RK3588 为例：

型号	默认延迟 (us)	寄存器读写速率 (us/s)	关闭CPU_SLEEP模式延迟 (us)	CPU定频最高频率延迟 (us)
RK3588	6	2	6	4

说明：

- 寄存器读写速率：调用 ELBI write config 接口的极限速率，即 RC 发起 ELBI 请求的最高速率，但实际由于 EP 的中断调度响应时间得到的响应速率会低于此结果

7. 控制器 - 中断响应速率-MSI

7.1 简介

MSI (Message Signaled Interrupt) 是 PCIe 中断传输机制的一种类型。MSI 通过在特定的消息地址上发送中断消息来触发中断。每个设备可以拥有多个消息地址，并且每个地址可以对应一个中断。这样就实现了并行处理中断请求，大大提高了响应速率和系统性能。

MSI 中断响应速率取决于 RC 系统总线带宽、设备驱动程序的优化程度以及处理器的处理能力等因素。一般来说，采用 MSI 中断机制的系统可以显著降低中断延迟，并且在高负载情况下仍能保持较好的响应速率。

7.2 测试 APP

PCIe MSI EP 端发包代码

```
#include <linux/kthread.h>

// PCIe MSI RC端和EP端示例代码

static int pcie_rkep_elbi_test_real(void *p)
{
    struct rockchip_pcie *rockchip = (struct rockchip_pcie *)p;

    // EP端循环发送MSI中断到RC端
    while(1) {
        rockchip_pcie_raise_msi_irq(rockchip, 0);
    }
}

static int pcie_rkep_elbi_test(struct rockchip_pcie *rockchip)
{
    struct task_struct *tsk;

    // 创建内核线程来执行pcie_rkep_elbi_test_real函数
    tsk = kthread_run(pcie_rkep_elbi_test_real, rockchip, "rkep-elbi");

    // 检查线程创建是否成功
    if (IS_ERR(tsk)) {
        pr_err("Failed to start rkep elbi thread\n");
        return PTR_ERR(tsk);
    }

    return 0;
}
```

PCIe MSI EP 端发包代码集成

```
 static int pcie_rkep_probe(struct pci_dev *pdev, const struct pci_device_id *id)
 {
        int ret, i;
@@ -774,6 +812,7 @@ static int pcie_rkep_probe(struct pci_dev *pdev, const struct
pci_device_id *id)
                pcie_rkep->obj_info->version);

        device_create_file(&pdev->dev, &dev_attr_rkep);
+       pcie_rkep_elbi_test(pcie_rkep);

        return 0;
 err_register_obj:
```

PCIe MSI RC 端测速代码

```
#include <linux/kthread.h>

static int pcie_rkep_elbi_test_real(void *p)
{
```

```
    struct pcie_rkep *pcie_rkep = (struct pcie_rkep *)p;
    int cur_count, last_count = pcie_rkep->msi_count;
    u64 us = 0;

    // 在一个无限循环中执行，每秒打印一次当前的msi_count值和两次打印之间的时间间隔
    while(1) {
        msleep(1000);
        cur_count = pcie_rkep->msi_count;
        if (cur_count) {
            us = 1000000 / (cur_count - last_count); // 计算时间间隔（微秒）
        }
        pr_err("msi count=%d %lluus\n", cur_count, us); // 打印msi_count和时间间隔
        last_count = pcie_rkep->msi_count;
    }

    return 0;
}

static __maybe_unused int pcie_rkep_elbi_test(struct pcie_rkep *pcie_rkep)
{
    struct task_struct *tsk;

    tsk = kthread_run(pcie_rkep_elbi_test_real, pcie_rkep, "rkep-elbi"); // 创建内
核线程来执行pcie_rkep_elbi_test_real函数
    if (IS_ERR(tsk)) {
        pr_err("rkep elbi start rk-pcie thread failed\n");
        return PTR_ERR(tsk);
    }

    return 0;
}
```

PCIe MSI RC 端测速代码调用处参考

```
 static int pcie_rkep_probe(struct pci_dev *pdev, const struct pci_device_id *id)
 {
        int ret, i;
@@ -774,6 +812,7 @@ static int pcie_rkep_probe(struct pci_dev *pdev, const struct
pci_device_id *id)
                pcie_rkep->obj_info->version);

        device_create_file(&pdev->dev, &dev_attr_rkep);
+       pcie_rkep_elbi_test(pcie_rkep);

        return 0;
 err_register_obj:
```

7.3 测试结果

RC	EP	默认延迟 (us)
RK3568-EVB1-DDR4-V10 PCIe 3.0 2 lanes ARM：1.99G DDR：1.5G	RK3568-IOTEST-DDR3-V10 PCIe 3.0 2 lanes ARM: 1.99G DDR：1.0G	2
RK3588-EVB1-LP4X-V10 PCIe 3.0 4 lanes 大核 2.4G 小核 1.8G	RK3588-EVB4-LP4X-V10 PCIe 3.0 4 lanes 大核 2.4G 小核 1.8G	2

说明:

- 请注意，具体的实现和应用场景可能会因系统配置、硬件平台和驱动要求而有所不同。

7.4 性能优化方向

MSI 速率的优化方向同 ELBI 中断优化方向一致，详细参考 “控制器 - 中断响应速率-ELBI” 章节。

8. 外设 - NVME 吞吐率

8.1 测试方法

测试方法详细参考《Rockchip_Developer_Guide_NVME.pdf》中 “性能评估” 章节，该文档目前为内部文档，如有需要请联系内部工程师获取。

8.2 测试结果

8.2.1 RK3568 Gen3x2

主机简介

- RK3568 EVB1 pcie3x2主控
- Gen3x2 PCIe 传输速率瓶颈主要在 PCIe 控制器，所以测试设备 CPU/DDR 等参数使用默认配置

外设

三星980 M.2 NVMe SSD 1TBPCIe Gen3x4

fio 模拟 crystalmark 8.0.1 测试结果

APP 测试名	传输速率MB/s	IOPS（K）
SEQ1M Q32T1 READ	1510	\
SEQ1M Q32T1 WRITE	1488	\
RND4K Q32T1 READ	152	40
RND4K Q32T1 WRITE	145	37

8.2.2 RK3588 Gen3x4

主机简介

- RK3588 EVB1 pcie3x4 主控 Port0/1 x4 RC
- 定频大核 2.4G 小核 1.8G、mq-deadline、f2fs nobarrier、LPDDR4X, 2112MHz

外设

三星980 M.2 NVMe SSD 1TBPCIe Gen3x4

PC CrystalDiskMark 测试结果：

PC crystalmark 8.0.1	**传输速率MB/s**	**IOPS（K）**
SEQ1M Q32T1 READ	3222	\
SEQ1M Q32T1 WRITE	2600	\
RND4K Q32T1 READ	708	\
RND4K Q32T1 WRITE	486	\

fio 模拟 crystalmark 8.0.1测试结果

APP 测试名	**传输速率MB/s**	**IOPS（K）**
SEQ1M Q32T1 READ	2514.2	\
SEQ1M Q32T1 WRITE	1342.1	\
RND4K Q32T1 READ	556	143
RND4K Q32T1 WRITE	436	112

8.2.3 RK3588 Gen3x2

主机简介

- RK3588 EVB1 pcie3x4 主控 Port0 x2 RC
- 定频大核 2.4G 小核 1.8G、mq-deadline、f2fs nobarrier、LPDDR4X, 2112MHz

外设

PHISON 128G BGA SSD

PC CrystalDiskMark 测试结果：

crystalmark 8.0.1	**传输速率MB/s**	**IOPS（K）**
SEQ1M Q32T1 READ	1615	\
RND4K Q32T1 READ	434	\
SEQ1M Q32T1 WRITE	626	\
RND4K Q32T1 WRITE	230	\

fio 模拟 crystalmark 8.0.1 测试结果

APP 测试名	传输速率MB/s	IOPS（K）
SEQ1M Q32T1 READ	1572.4	\
SEQ1M Q32T1 WRITE	600	\
RND4K Q32T1 READ	422	101
RND4K Q32T1 WRITE	355	90

8.2.4 RK3588s Gen2x1

主机简介

- RK3588s EVB1 pcie2x1l1主控 Port0 x1 RC
- 定频大核 2.4G 小核 1.8G、mq-deadline、f2fs nobarrier、LPDDR4X, 2112MHz

外设

PHISON 128G BGA SSD。

fio 模拟 crystalmark 8.0.1 测试结果

APP 测试名	传输速率MB/s	IOPS（K）
SEQ1M Q32T1 READ	407	\
SEQ1M Q32T1 WRITE	350	\
RND4K Q32T1 READ	400	102
RND4K Q32T1 WRITE	134	32

8.2.5 RK3528 Gen2x1

主机简介

- RK3528 EVB2 pcie2x1主控
- Gen2x1 PCIe 传输速率瓶颈完全在 PCIe 控制器，所以测试设备 CPU/DDR 等参数使用默认配置

外设

三星980 M.2 NVMe SSD 1TBPCIe Gen3x4

fio 模拟 crystalmark 8.0.1 测试结果

APP 测试名	传输速率MB/s	IOPS（K）
SEQ1M Q32T1 READ	410	\
SEQ1M Q32T1 WRITE	392	\
RND4K Q32T1 READ	327	83
RND4K Q32T1 WRITE	300	76

8.2.6 RK3562 Gen2x1

主机简介

- RK3562 EVB1 pcie2x1主控
- Gen2x1 PCIe 传输速率瓶颈完全在 PCIe 控制器，所以测试设备 CPU/DDR 等参数使用默认配置

外设

三星980 M.2 NVMe SSD 1TBPCIe Gen3x4

fio 模拟 crystalmark 8.0.1 测试结果

APP 测试名	传输速率MB/s	IOPS（K）
SEQ1M Q32T1 READ	409	\
SEQ1M Q32T1 WRITE	391	\
RND4K Q32T1 READ	343	85
RND4K Q32T1 WRITE	253	63