StarFive
赛昉科技

在昉·星光 2上运行AMP双系统（RT-Thread + Linux）

版本：1.1

日期：2024/10/25

Doc ID：VisionFive 2-ANCH-020

法律声明

阅读本文件前的重要法律告知。

版权注释

本文档中的说明均基于“视为正确”提供，可能包含部分错误。内容可能因产品开发而定期更新或修订。广东赛昉科技有限公司 （以下简称“赛昉科技”）保留对本协议中的任何内容进行更改的权利，恕不另行通知。

赛昉科技明确否认任何形式的担保、解释和条件，无论是明示的还是默示的，包括但不限于适销性、特定用途适用性和非侵权的担保或条件。

赛昉科技无需承担因应用或使用任何产品或电路而产生的任何责任，并明确表示无需承担任何及所有连带责任，包括但不限于间接、偶然、特殊、惩戒性或由此造成的损害。

联系我们：

地址： 广东省佛山市顺德区大良街道云路社区昊阳路2号A区S201室

网站： http://www.starfivetech.com

邮箱：

- sales@starfivetech.com（销售）
- support@starfivetech.com（支持）

前言

关于本指南和技术支持信息

关于本手册

本手册主要为开发者阐述了在赛昉科技新一代SoC平台——昉·惊鸿-7110上运行异构AMP双系统（Linux + RT-Thread）的演示示例。该代码已发布到赛昉科技Linux 6.6 SDK。

修订历史

Table 0-1 修订历史

版本	发布说明	修订
1.1	2024/10/25	更新了以下章节： • AMP相关代码 *(on page 9)* • RT-Thread启动和Memory分配 *(on page 10)* • 编译步骤 *(on page 11)* • RT-Thread组件和驱动 *(on page 13)* • RT-Thread性能 *(on page 19)*
1.0	2024/01/17	首次正式发布。

注释和注意事项

本指南中可能会出现以下注释和注意事项：

- **Tip:**
  建议如何在某个主题或步骤中应用信息。
- **Note:**
  解释某个特例或阐释一个重要的点。
- **Important:**
  指出与某个主题或步骤有关的重要信息。
- **CAUTION:**
  表明某个操作或步骤可能会导致数据丢失、安全问题或性能问题。
- **Warning:**
  表明某个操作或步骤可能导致物理伤害或硬件损坏。

Contents

List of Tables

List of Figures

1. 概述

本手册主要为开发者阐述了在赛昉科技新一代SoC平台——昉·惊鸿-7110上运行异构AMP双系统（Linux + RT-Thread）的演示示例。该代码已发布到赛昉科技Linux 6.6 SDK。

昉·惊鸿-7110包含4个主CPU，本文中要实现的异构AMP即让其中1个CPU跑RT-Thread RTOS，以此形成3个CPU跑Linux操作系统，1个CPU跑RT-Thread的双系统AMP架构。其中在RTOS的CPU运行实时的进程，并把部分实时驱动运行在RTOS中进行数据采集，同时把数据通过共享内存方式发回到Linux上，Linux端可以运行各种非实时的应用程序。这种方式可以使系统既保证实时性，又能使用Linux通用操作系统运行功能强大的应用。这已成为工业系统中一种重要架构。

这种方式可以解决RT-Linux无法达到最大调度延时15us以内的棘手问题， 在昉·惊鸿-7110中，运行RTOS的CPU Core主频可以跑1.5GHz， 最大调度延时可以跑到15us以内。 企业用户如果有需要在昉·惊鸿-7110运行Linux + other RTOS或者Linux + Baremetal，通过本文可以方便的把RT-Thread仓库的驱动移植到需要的RTOS或者Baremetal上。

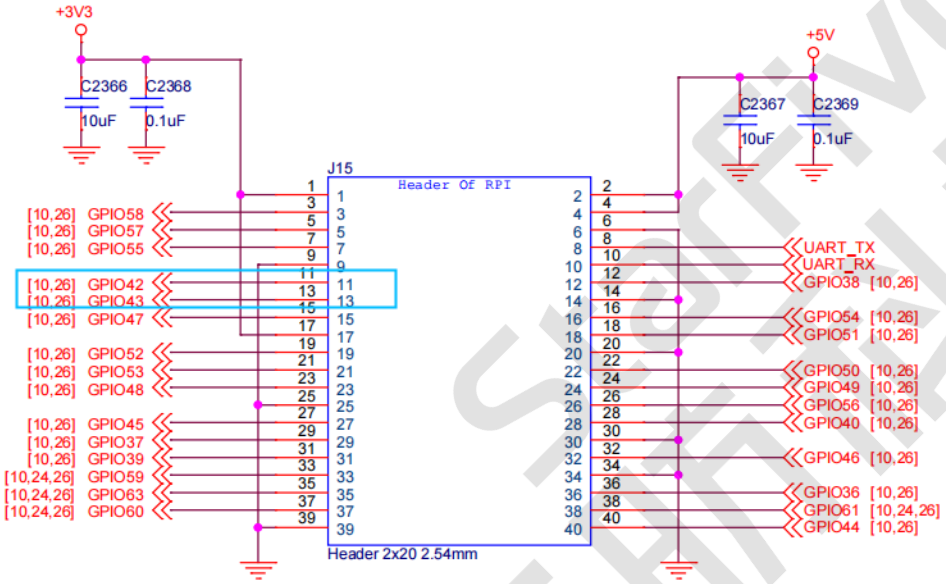
2. RT-Thread调试串口

Linux沿用UART0作为系统串口，而RT-Thread使用UART2作为系统串口。本文中使用昉·星光 2 40-pin接口上的pin11和pin13作为RX/TX pin，下图为昉·星光 2 40-pin接口电路图。

Pin9、Pin11和Pin13组成一个完整的串口：

- **Pin9 (GND)**
- **Pin11 (GPIO42)：**UART2 RX
- **Pin13 (GPIO43)：**UART2 TX

Figure 2-1 40-Pin Out



3. AMP相关代码

本章主要介绍了以下两个内容：

3.1. Linux代码

Linux代码中包含以下三个内容：

1. **RPMsg代码：**

   两核通信使用标准的virtio-base的RPMsg协议。RPMsg，全称为Remote Processor Messaging，它定义了异构多核处理系统（AMP，Asymmetric Multiprocessing）中核与核之间进行通信时所使用的标准二进制接口。

   在Linux内核代码中，RPMsg的代码是：

   ```
   driver/rpmsg/virtio_rpmsg_bus.c
   drivers/rpmsg/starfive_rpmsg.c
   ```

2. **Mailbox代码：**

   ```
   drivers/mailbox/starfive_ipi_mailbox.c
   ```

3. **AMP DTS文件：**

   `arch/riscv/boot/dts/starfive/jh7110-starfive-visionfive-2-amp.dts`

3.2. RT-Thread代码

RT-Thread 开发基于5.0.2的版本，在此基础上增加昉·惊鸿-7110的代码：

1. **昉·惊鸿-7110代码：**包含昉·惊鸿-7110 Clock、Pinctrl 和GPIO驱动代码，简单的应用等。

   ```
   rtthread/bsp/starfive/jh7110/
   ```

2. **RPMsg代码：**

   RT-Thread使用开源的`rpmsg-lite`代码，也是开源的`virtio-base`的RPMsg代码，能够按照协议和Linux收发数据。核间的IPI中断和共享内存配合能实现异构核间的数据传输。

   ```
   rtthread/bsp/starfive/libraries/component/rpmsg-lite
   ```

3. **驱动代码：** 移植到RT-Thread的驱动，包括UART、GMAC、PCIe和CAN驱动。

   ```
   rtthread/bsp/starfive/libraries/driver
   ```

4. **测试应用程序**位于以下路径：

   `rtthread/bsp/starfive/libraries/applications`

4. RT-Thread启动和Memory分配

AMP启动中，Linux和RT-Thread各自独立启动，其配置入口设置在U-Boot的DTS中，其中分割了Linux domain和RTOS domain。在OpenSBI中每个核会根据不同配置跳转到不同的地址，其中RT-Thread没有跳转到U-Boot的第二阶段，直接从OpenSBI跳转到RT-Thread。

RT-Thread端

RT-Thread的`rtthread.bin`和`u-boot.bin`文件一起生成`visionfive2_fw_payload_amp.img`，SPL镜像`u-boot-amp-spl.bin.normal.out`会把该镜像读到DDR的起始物理地址`0x40000000`。该镜像的组成部分如下：

Table 4-1 内存地址范围

RT-Thread端	范围	内存大小	是否linux kernel回收
OpenSBI	0×40000000 - 0×401fffff	2M	一直保留
U-Boot (S Mode)	0×40200000 - 0×4032ffff	1.19M	启动后Linux kernel会被回收
RT-Thread	0×6e800000 - 0×6effffff	8M	U-Boot-SPL第一次把RT-Thread加载到内存中，起始地址是为0×40330000，然后迁移到0×6e800000地址，0×40330000的地址会被回收。

Linux端

Linux端为AMP保留了28M内存，其中共享内存设置为4M。内存分布如下：

Table 4-2 内存地址范围

Linux端	范围	内存大小
共享内存	0×6e400000 - 0×6e7fffff	4M
RT-Thread代码，栈空间	0×6e800000 - 0×6effffff	8M
RT-Thread堆空间	0×6f000000 - 0×6fffffff	16M

5. 编译步骤

AMP所有代码已经合并到赛昉科技Linux 6.6 SDK，AMP镜像的编译目录可以和常规的SDK目录共存，即您可在同一SDK目录下编译出AMP和常规镜像，如果没下载Linux 6.6 SDK， 请先按照该链接的步骤下载。

下载完成后，请按照以下步骤进行编译：

1. 执行以下命令， 下载代码和编译AMP镜像。

```
$ ./build-rtthread-amp-sdk.sh
```

Note:

- 其中会从github下载RT-Thread的工具链，该工具链可支持Ubunbu 18/20/22版本， 请根据Ubuntu的版本选择正确的工具链。
- 第一次下载和编译的时间较长，请耐心等待。

结果：

最终在`work`目录下编译生成`sdcard_amp.img`，可以直接刷写到SD卡并启动。如需net boot，其他镜像为：

```
work/u-boot-amp-spl.bin.normal.out   #uboot spl image
work/visionfive2_fw_payload_amp.img   #sbi payload image, including rt-thread.bin
work/amp/image.fit   #AMP kernel image
```

2. 如仅修改了RT-Thread，可以单独编译`visionfive2_fw_payload_amp.img`：

```
$ make amp-clean
$ make ampuboot_fit -j8
```

3. 如需要重新编译AMP镜像，请执行以下命令：

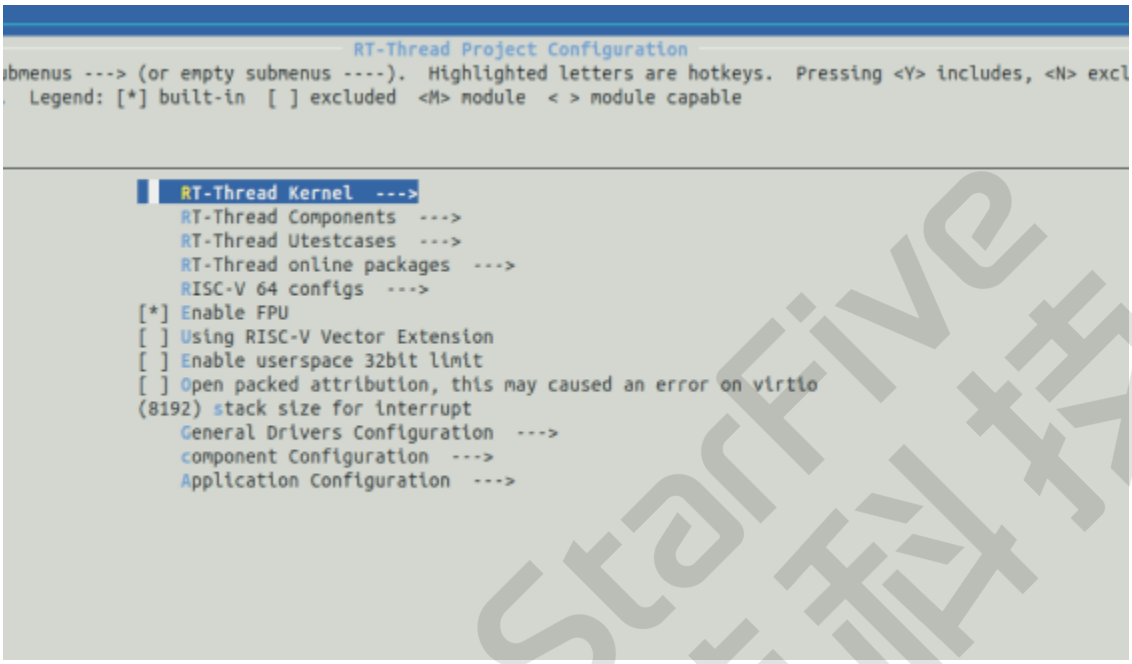
```
$ make amp_img
```

4. 如需配置裁剪RT-Thread，在`jh7110`目录下，输入以下命令：

```
$ cd rtthread/bsp/starfive/jh7110
$ cp configs/vf2_defconfig .config
$ cp configs/vf2_rtconfig.h rtconfig.h
$ scons --menuconfig
```

Tip:
menuconfig 是一种图形化配置工具，是RT-Thread 3.0以上版本的特性，可对内核、组件和软件包进行自由裁剪，使系统以搭积木的方式进行构建。

Figure 5-1 配置RT-Thread



6. RT-Thread组件和驱动

本章介绍了RT-Thread组件和驱动：

6.1. RPMsg

按照以下步骤，运行AMP镜像，并使用RPMsg组件测试：

1. 连接Linux和RTOS的调试串口 *(on page 8)*，串口的波特率均设置为115,200。
2. 将编译出来的`sdcard_amp.img`刷写到SD卡上。
3. 上电启动：RT-Thread启动很快，运行到main程序的时候，需要等Linux启动完成后发送IPI中断，Linux端是RPMsg的master，需要配置virtio queue的控制内存和共享内存。

   Figure 6-1 上电启动

```
SBI: OpenSBI v1.2
SBI Specification Version: 1.0 1
heap: [0x6f000000 - 0x70000000]
initialize rti_board_start:0 done
initialize riscv_cputime_init:0 done
initialize rt_ktime_hrtimer_lock_init:0 done

 \ | /
- RT -     Thread Operating System
 / | \     5.0.2 build Jul  4 2024 14:20:38
 2006 - 2022 Copyright by RT-Thread team
do components initialization.
initialize rti_board_end:0 done
initialize dfs_init:0 done
initialize rt_work_sys_workqueue_init:0 done
initialize lwip_system_initlwIP-2.0.3 initialized!
:0 done
initialize null_device_init:0 done
initialize random_device_init:0 done
initialize urandom_device_init:0 done
initialize zero_device_init:0 done
initialize sal_init[I/sal.skt] Socket Abstraction Layer initialize success.
:0 done
initialize rt_posix_stdio_init:0 done
initialize finsh_system_init:0 done
Hello RISC-V
```

4. 启动Linux：启动linux过程中，`virtio_rpmsg_bus`驱动会注册，`starfive_rpsmg`驱动也会被注册，注册完成后会发IPI中断给RT-Thread。

Figure 6-2 启动Linux

```
Populating /dev using udev: [    9.649011] udevd[170]: starting version 3.2.10
[    9.669120] mmc0: Failed to initialize a non-removable card
[    9.715116] udevd[171]: starting eudev-3.2.10
[    9.833767] virtio_rpmsg_bus virtio0: rpmsg host is online
[    9.847098] virtio_rpmsg_bus virtio1: rpmsg host is online
[    9.853371] jpu: loading out-of-tree module taints kernel.
[    9.861054] virtio_rpmsg_bus virtio2: rpmsg host is online
[    9.866552] cnm_jpu 13090000.jpu: init device.
[    9.875400] SUCCESS alloc_chrdev_region
[    9.879618] virtio_rpmsg_bus virtio3: rpmsg host is online
[    9.885520] vdec 130a0000.vpu_dec: device init.
[    9.889710] virtio_rpmsg_bus virtio4: rpmsg host is online
[    9.890097] SUCCESS alloc_chrdev_region
[    9.900524] virtio_rpmsg_bus virtio5: rpmsg host is online
[    9.920233] virtio_rpmsg_bus virtio6: rpmsg host is online
[    9.930510] virtio_rpmsg_bus virtio7: rpmsg host is online
[    9.936080] starfive-rpmsg 6e404000.rpmsg: registered 6e404000.rpmsg
```

Tip:
图中注册了8个RPMsg设备节点， 可以支持多个应用程序进行数据交互通信。

RT-Thread接受到IPI中断后，会初始化驱动和创建`rpmsg_linux_test`应用进程，这时RT-Thread的finsh shell也能正常使用。

Figure 6-3 RT-Thread进程

```
msh />ps
rt_thread_t        thread           pri  status      sp     stack size max used left tick  error
------------------ ---------------- ---  ------- ---------- ---------- ------  ---------- ---
0x000000006f1d0238 rpmsg_linux_test   9  suspend 0x00000408 0x00001000    25%   0x00000014 EINTRPT
0x000000006f1c9c88 link_detect       13  suspend 0x00000328 0x00001000    19%   0x00000014 EINTRPT
0x000000006f006690 tshell            20  running 0x00000708 0x00001000    43%   0x00000002 OK
0x000000006f003d28 tcpip             10  suspend 0x00000398 0x00002000    20%   0x0000000a EINTRPT
0x000000006e879c10 etx               12  suspend 0x00000318 0x00002000    09%   0x00000010 EINTRPT
0x000000006e87bda8 erx               12  suspend 0x00000328 0x00002000    20%   0x00000002 EINTRPT
0x000000006f001a98 sys workq         23  suspend 0x00000288 0x00002000    15%   0x0000000a OK
0x000000006e87e860 tidle0            31  ready   0x00000268 0x00001000    16%   0x0000000e OK
0x000000006e87fc78 timer              4  suspend 0x00000278 0x00001000    18%   0x00000001 OK
```

5. Linux端运行以下命令能看到 RT-Thread发给Linux的IPI中断：

```
cat /proc/interrupts
```

Figure 6-4 IPI中断

```
IPI0:       304        341        451  Rescheduling interrupts
IPI1:      2317       1487       2436  Function call interrupts
IPI2:         0          0          0  CPU stop interrupts
IPI3:         0          0          0  CPU stop (for crash dump) interrupts
IPI4:       269        203        252  IRQ work interrupts
IPI5:         0          0          0  Timer broadcast interrupts
IAMP:         1          0          0  AMP rpmsg interrupts
#
```

6. 运行以下测试程序：

```
rpmsg_echo
```

Tip:
RVspace上提供了已编译的应用程序和源代码。该应用程序向RPMsg的remote端发送一个字符串， RT-Thread接收到后会把收到字符串发回给Linux，测试结果如下面所示：

Figure 6-5 测试结果

```
# ./rpmsg_echo
Sending message #0: hello there 0!
Receiving message #0: hello there 0!
Sending message #1: hello there 1!
Receiving message #1: hello there 1!
Sending message #2: hello there 2!
Receiving message #2: hello there 2!
Sending message #3: hello there 3!
Receiving message #3: hello there 3!
Sending message #4: hello there 4!
Receiving message #4: hello there 4!
Sending message #5: hello there 5!
Receiving message #5: hello there 5!
Sending message #6: hello there 6!
Receiving message #6: hello there 6!
Sending message #7: hello there 7!
Receiving message #7: hello there 7!
Sending message #8: hello there 8!
Receiving message #8: hello there 8!
Sending message #9: hello there 9!
Receiving message #9: hello there 9!
```

IPI中断情况：

Figure 6-6 IPI中断

```
IPI0:       907       1329       1166  Rescheduling interrupts
IPI1:      3814       2129       3691  Function call interrupts
IPI2:         0          0          0  CPU stop interrupts
IPI3:         0          0          0  CPU stop (for crash dump) interrupts
IPI4:       702        428        684  IRQ work interrupts
IPI5:         0          0          0  Timer broadcast interrupts
IAMP:        18          0          0  AMP rpmsg interrupts
# ■
```

6.2. GPIO驱动

RTOS端支持GPIO 驱动。在RT-Thread的finsh下使能pin 命令可操作GPIO的输出和输入， 下面以GPIO47为例：

```
msh />pin mode 47 output
msh />pin write 47 low
msh />pin read 47
pin[47] = low
msh />pin write 47 high
msh />pin read 47
pin[47] = high
```

结果：可以通过测量GPIO47来查看高低电平情况。

RTOS支持GPIO中断驱动，但由于GPIO中断资源只有一份， 默认为Linux占用，因此仅Linux没有使用GPIO中断时，RTOS才能使用GPIO中断。昉·星光 2 SDK中由于Linux端有使用GPIO中断，RT-Thread默认没有激活GPIO中断， 如果需要激活GPIO中断，需要在RT-Thread配置中使能`BSP_USING_GPIO`。

6.3. GMAC驱动

RT-Thread支持lwIP TCP/IP协议栈，可以在RT-Thread上进行网络编程，在AMP SDK中，默认把GMAC1移到RTOS端，GMAC的驱动移植到RT-Thread。因此，可以在昉·惊鸿-7110的RT-Thread进行网络应用程序编程，也可以作为实时网卡运行小型的Ethercat主站。

如下图所示，RT-Thread启动后会初始化GMAC，并把GMAC驱动注册到lwIP TCP/IP协议栈，默认开启了DHCP功能， 成功从DHCP 网络获取IP地址，GMAC能正常工作。

Figure 6-7 GMAC驱动

```
netif: IP address of interface g1 set to 0.0.0.0
netif: netmask of interface g1 set to 0.0.0.0
netif: GW address of interface g1 set to 0.0.0.0
Waiting for PHY auto negotiation to complete......
speed 1000 duplex 1
gmac g1 link up
netif: setting default interface g1
netif: added interface g1 IP addr 0.0.0.0 netmask 0.0.0.0 gw 0.0.0.0
Hello Starfive RT-Thread! CPU_ID(4)
rpmsg linux test: receive data from linux then send back
rpmsg remote: link up! link_id-0x0
msh />netif: netmask of interface g1 set to 255.255.255.0
netif: GW address of interface g1 set to 192.168.125.1
netif_set_ipaddr: netif address being changed
netif: IP address of interface g1 set to 192.168.125.132
```

同时可以Ping到外网，并查看网口情况，如下图所示:

Figure 6-8 网卡正常工作

```
msh />ping www.baidu.com
ping: not found specified netif, using default netdev g1.
60 bytes from 180.101.50.242 icmp_seq=0 ttl=52 time=32 ms
60 bytes from 180.101.50.242 icmp_seq=1 ttl=52 time=30 ms
60 bytes from 180.101.50.242 icmp_seq=2 ttl=52 time=30 ms
60 bytes from 180.101.50.242 icmp_seq=3 ttl=52 time=30 ms
```

Figure 6-9 查看网口情况

```
msh />ifconfig
network interface device: g1 (Default)
MTU: 1500
MAC: 6c cf 39 00 27 48
FLAGS: UP LINK_UP INTERNET_UP DHCP_ENABLE ETHARP BROADCAST IGMP
ip address: 192.168.125.132
gw address: 192.168.125.1
net mask  : 255.255.255.0
dns server #0: 192.168.110.101
dns server #1: 202.207.240.225
```

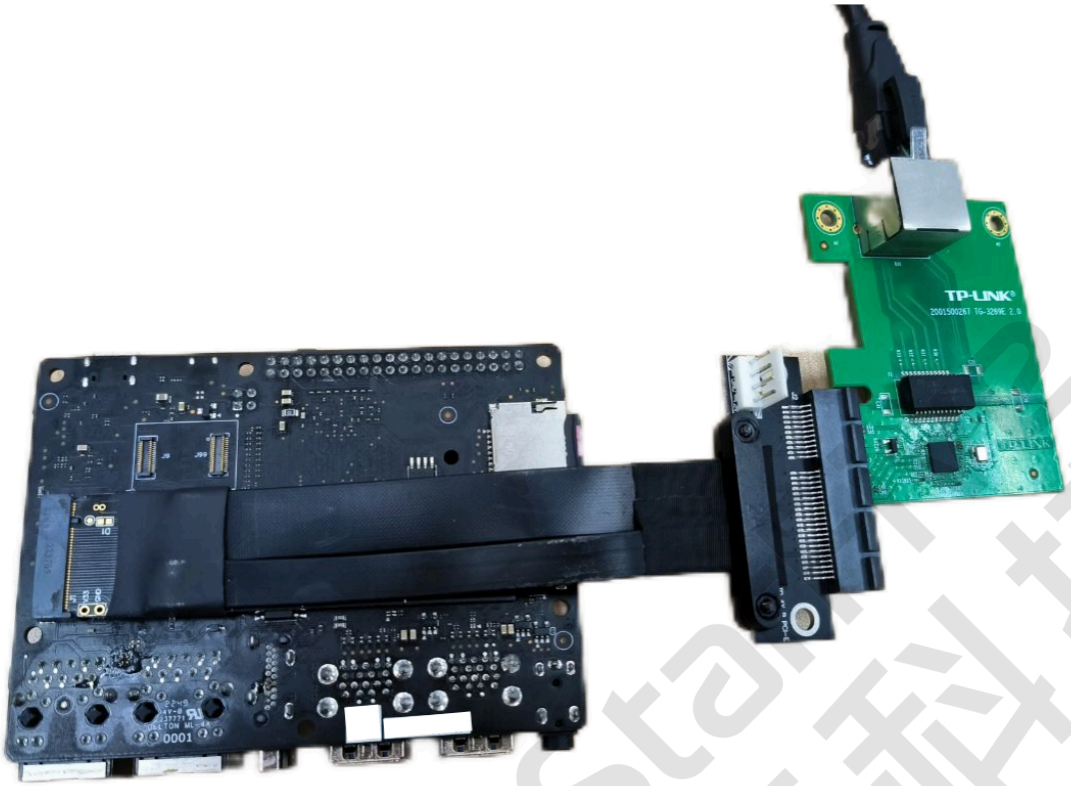
6.4. PCIe驱动

在工业场景中，可能需要用到PCIe扩展网卡，又或者通过PCIe接FPGA的PCIe EP设备等，形成SoC + FPGA设计。FPGA用于数据采集，和SoC通信可能有实时要求，昉·惊鸿-7110包含2个PCIe 2.0 host，单个PCIe2.0理论上最高支持速度可达500MB/s，在工业上应用场景广阔。因此，赛昉科技移植了PCIe驱动到RTOS端，为验证PCIe驱动， 接入了PCIe 网卡来验证RTOS上的PCIe驱动。

Tip:
目前驱动只支持PCIe bus单设备的情况，暂不支持PCIe接switch下多设备驱动。

昉·星光 2上PCIe1使用M.2 Key的接口，可以通过转接线扩展为PCIe接口， 通过接RTL816X系列的PCIe网卡来验证PCIe能否正常工作，RTOS上也支持RTL816x驱动程序。在下面演示中， 可以看到RT-Thread下支持了昉·惊鸿-7110的GMAC1和RTL8161的双千兆网卡。

Figure 6-10 连接PCIe驱动



下图是PCIe驱动初始化打印和RTL8161网卡初始化流程，PCIe 初始化后会扫描PCI bus上的设备，能扫描到网卡设备。RTL8161网卡驱动程序匹配到ID后会执行初始化，并且申请MSI中断，图中可见初始化并注册了GE网卡， 成功申请到了IP地址，PCIe网卡能正常工作。

Figure 6-11 初始化流程

```
pcie port link up
ATR entry: 0x09c0000000 -> 0x0000000000 [0x0010000000] (param: 0x000001)
ATR entry: 0x0980000000 -> 0x0980000000 [0x0040000000] (param: 0x000000)
ATR entry: 0x0038000000 -> 0x0038000000 [0x0008000000] (param: 0x000000)
PCI Autoconfig: Bus Memory region: [38000000-3fffffff],
PCI Autoconfig: Bus Prefetchable Mem region: [980000000-9bfffffff],
PCI Autoconfig: Found P2P bridge, device 0
PCI Autoconfig: BAR 0, Prf64, size=0x0, No room in resource, avail start=980000000 / size=40000000, need=0
PCI: Failed autoconfig bar 10

supports D1 D2
PME# supported from D0 D1 D2 D3hot D3cold
pm current state 0
PCI Autoconfig: BAR 0, I/O, size=0x100, No resource
PCI: Failed autoconfig bar 10

PCI Autoconfig: BAR 1, Mem64, size=0x1000, address=0x38000000 bus_lower=0x38001000

PCI Autoconfig: BAR 2, Mem64, size=0x4000, address=0x38004000 bus_lower=0x38008000

supports D1 D2
PME# supported from D0 D1 D2 D3hot D3cold
pm current state 0
netif: IP address of interface g1 set to 0.0.0.0
netif: netmask of interface g1 set to 0.0.0.0
netif: GW address of interface g1 set to 0.0.0.0
netif: setting default interface g1
netif: added interface g1 IP addr 0.0.0.0 netmask 0.0.0.0 gw 0.0.0.0
iobase 38000000
MAC Address:f4:6d:2f:de:3c:f4
netif: IP address of interface ge set to 0.0.0.0
netif: netmask of interface ge set to 0.0.0.0
netif: GW address of interface ge set to 0.0.0.0
rtl8169: REALTEK RTL8169 @0x38000000 0x38000000
rtl8169_eth_open: FuncEvent/Misc (0xF0) = 0x0000003F
msi#0 address_hi 0x0 address_lo 0x190
gmac ge link up
netif: added interface ge IP addr 0.0.0.0 netmask 0.0.0.0 gw 0.0.0.0
Hello Starfive RT-Thread! CPU_ID(4)
rpmsg linux test: receive data from linux then send back
rpmsg remote: link up! link_id-0x0
msh />netif: netmask of interface ge set to 255.255.255.0
netif: GW address of interface ge set to 192.168.125.1
netif_set_ipaddr: netif address being changed
netif: IP address of interface ge set to 192.168.125.91
```

同时，双网卡均能获得IP地址和正常访问网络。

- 获得IP地址:

Figure 6-12 获得IP地址

```
msh />ifconfig
network interface device: g1
MTU: 1500
MAC: 6c cf 39 00 27 48
FLAGS: UP LINK_UP INTERNET_UP DHCP_ENABLE ETHARP BROADCAST IGMP
ip address: 192.168.125.132
gw address: 192.168.125.1
net mask   : 255.255.255.0
dns server #0: 192.168.110.101
dns server #1: 202.207.240.225

network interface device: ge (Default)
MTU: 1500
MAC: f4 6d 2f de 3c f4
FLAGS: UP LINK_UP INTERNET_UP DHCP_ENABLE ETHARP BROADCAST IGMP
ip address: 192.168.125.91
gw address: 192.168.125.1
net mask   : 255.255.255.0
dns server #0: 192.168.110.101
dns server #1: 202.207.240.225
```

- 正常工作:

Figure 6-13 正常工作

```
msh />ping www.baidu.com ge
60 bytes from 180.101.50.242 icmp_seq=0 ttl=52 time=33 ms
60 bytes from 180.101.50.242 icmp_seq=1 ttl=52 time=34 ms
60 bytes from 180.101.50.242 icmp_seq=2 ttl=52 time=33 ms
60 bytes from 180.101.50.242 icmp_seq=3 ttl=52 time=33 ms
msh />ping www.baidu.com g1
60 bytes from 180.101.50.242 icmp_seq=0 ttl=52 time=35 ms
60 bytes from 180.101.50.242 icmp_seq=1 ttl=52 time=34 ms
60 bytes from 180.101.50.242 icmp_seq=2 ttl=52 time=34 ms
60 bytes from 180.101.50.242 icmp_seq=3 ttl=52 time=33 ms
msh />
```

7. RT-Thread性能

本节从以下两个方面介绍了RT-Thread的性能：

7.1. 调度延时

在RT-Thread下进行类似于cyclictest调度延时测试，U74在1.5GHz的情况下，idle状态下跑了12个小时，平均延时为1us， 最大延时为2us。

在有以太网卡，并处于混杂模式的工作情况下，最大延时为10us。

7.2. 中断延时

UART中断延时

在1.5GHz下测试UART的RX延时，从**接收 > 中断 > finsh shell进程接收到字符**，时间大约是6us。