Run AMP System (RT-Thread + Linux) on VisionFive 2

Version: 1.1
Date: 2024/10/25
Doc ID: VisionFive 2-ANEN-020

Legal Statements

Important legal notice before reading this documentation.

PROPRIETARY NOTICE

Contact Us

Address: Room S201, Zone A, No. 2, Haoyang Road, Yunlu Community, Daliang Subdistrict, Shunde District, Foshan, Guangdong, China, 528300

Website: http://www.starfivetech.com

Email:

- Sales: sales@starfivetech.com
- Support: support@starfivetech.com

Contents

List of Tables

List of Figures

Preface

About this guide and technical support information.

About this document

This application note provides steps to run heterogeneous asymmetric multiprocessing(AMP) system (Linux + RT-Thread) on StarFive new generation SoC platform - JH-7110. The code has been released to StarFive Linux 6.6 SDK.

Version History

Table 0-1 Version History

Version	Released	Revision
1.1	2024/10/25	Updated the following sections: • AMP Related Code *(on page 9)* • RT-Thread Startup and Memory Allocation *(on page 10)* • Compilation *(on page 11)* • The Components and Drivers of RT-Thread *(on page 13)* • RT-Thread Performance *(on page 19)*
1.0	2024/01/17	The first official release.

Notes and notices

The following notes and notices might appear in this guide:

- **Tip:**
  Suggests how to apply the information in a topic or step.
- **Note:**
  Explains a special case or expands on an important point.
- **Important:**
  Points out critical information concerning a topic or step.
- **CAUTION:**
  Indicates that an action or step can cause loss of data, security problems, or performance issues.
- **Warning:**
  Indicates that an action or step can result in physical harm or cause damage to hardware.

1. Introduction

This application note provides steps to run heterogeneous asymmetric multiprocessing(AMP) system (Linux + RT-Thread) on StarFive new generation SoC platform - JH-7110. The code has been released to StarFive Linux 6.6 SDK.

JH-7110 includes 4 main CPUs. The heterogeneous AMP to be implemented in this application note is to use one CPU run RT-Thread RTOS, 3 CPUs run Linux OS, thus build a dual system AMP architecture with 3 CPUs run Linux OS and 1 CPU runs RT-Thread RTOS. Among them, real-time processes are run on the CPU of RTOS, and some real-time drivers can run for data collection. At the same time, data can be sent back to Linux through shared memory, and various non real-time applications can be run on the Linux side. This allows the system to ensure real-time performance and run powerful applications using the Linux universal OS, which becomes an important architecture in industrial systems.

This approach can solve the tricky problem of RT-Linux cannot achieve a maximum schedule delay of less than 15us. Because in JH-7110, the CPU core (running RTOS) can run at 1.5GHz and its' maximum schedule delay can run within 15us. If users need to run `Linux+other RTOS` or `Linux+Baremetal` on JH-7110, this article can help them easily port the RT-Thread repository driver to the required RTOS or Baremetal.

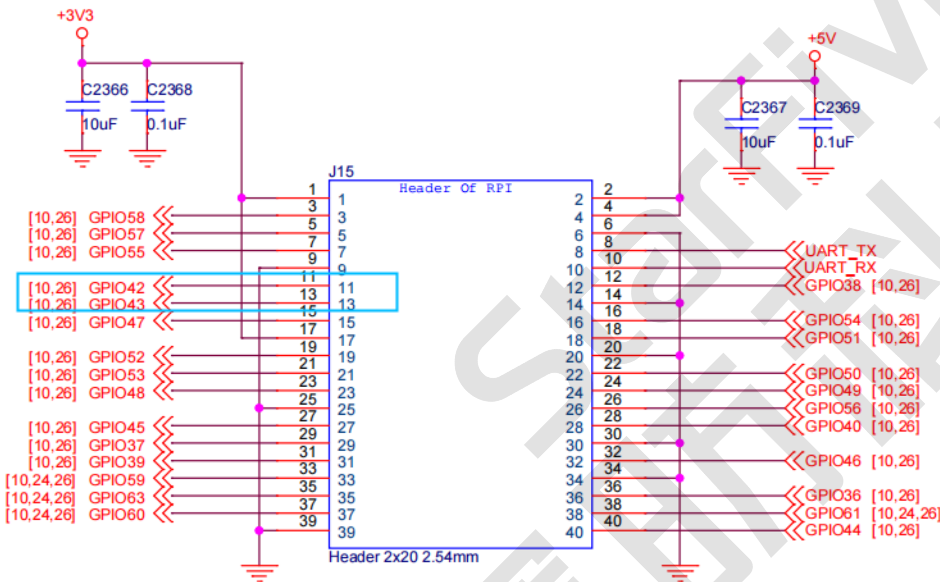
2. RT-Thread Debug Port

Linux uses UART0 as the system serial port, while RT-Thread uses UART2 as the system serial port. In this application note, pin11 and pin13 on 40-pin GPIO of VisionFive 2 are used as RX/TX pins. The following shows the circuit diagram of the VisionFive 2 40-pin GPIO.

Pin9, pin11 and pin13 form a complete serial port:

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

Figure 2-1 40-Pin Out



3. AMP Related Code

3.1. Linux Code

The Linux code contains the following three elements:

1. **RPMsg code:**

   Inter-processor uses the standard `virtio-base` RPMsg protocol to communicate. RPMsg, also known as Remote Processor Messaging, defines the standard binary interface used for communication between cores in heterogeneous AMP system.

   In Linux kernel, the codes of RPMsg are:

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

2. **Mailbox code:**

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

3. **AMP DTS file:**

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

3.2. RT-Thread code

RT-Thread is developed by version 5.0.2, based on which add the code of JH-7110:

1. **JH-7110 code:** Contains driver code of JH-7110 Clock, Pinctrl and GPIO, and other simple applications.

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

2. **RPMsg code:**

   RT-Thread: uses open-source `rpmsg-lite` code, which is also the open-source `virtio-base` code of RPMsg. It can send and receive data with Linux according to the protocol. The combination of IPI interrupts between cores and shared memory can achieve data transmission between heterogeneous cores. The filepath of RT-Thread code is as follows:

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

3. **Driver code:** Port to the RT-Thread drivers, including UART, GMAC, PCIe, and CAN drivers.

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

4. The test application is located in the following path:

   `rtthread/bsp/starfive/libraries/applications`

4. RT-Thread Startup and Memory Allocation

In AMP startup process, Linux and RT-Thread RTOS start up independently, with their configuration entry set in the DTS of U-Boot, which separates Linux domain and RTOS domain. In OpenSBI, each core will jump to a different address based on different configurations. RT-Thread does not jump to the second stage of U-Boot, but directly jumps from OpenSBI to RT-Thread.

RT-Thread Side

The `rtthread.bin` and `u-boot.bin` files of the RT-Thread are used together to generate `visionfive2_fw_payload_amp.img`, and SPL image `u-boot-amp-spl.bin.normal.out` will read this image to the starting physical address of DDR, which is `0×40000000`. The components of this image are as follows:

Table 4-1 Memory Address Range

RT-Thread Side	Range	Memory	Reclaim Linux kernel or not?
OpenSBI	0×40000000 - 0×401fffff	2M	Always keep it
U-Boot (S Mode)	0×40200000- 0×4032ffff	1.19M	Linux kernel will be reclaimed after startup
RT-Thread	0×6e800000 - 0×6effffff	8M	First U-Boot SPL loads RT-Thread into memory, the starting address is `0x40330000`, and then migrates to `0x6e800000`. The `0x40330000` address will be reclaimed.

Linux Side

The Linux side has reserved 28M for AMP, with shared memory set to 4M. The memory distribution is as follows:

Table 4-2 Memory Address Range

Linux Side	Range	Memory
Shared Memory	0×6e400000 - 0×6e7fffff	4M
RT-Thread code, stack space	0×6e800000 - 0×6effffff	8M
RT-Thread code, stack space	0×6f000000 - 0×6fffffff	16M

5. Compilation

All AMP code has been merged into StarFive Linux 6.6 SDK. The compilation directory of the AMP image can coexist with the regular SDK directory, which means you can compile both AMP and regular images in the same SDK directory. If you haven't downloaded the Linux 6.6 SDK, please follow the steps in this link to download it first.

Then follow the steps below to compile:

1. Execute the following commands to download the code and compile the AMP images:

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

   Note:

   - The RT-Thread tool chain will be downloaded from the github, which can support Ubuntu 18/20/22 versions. Please select the correct tool chain according to the Ubuntu version.
   - This process may take long time. Please be patient.

   Result:

   Finally, the `sdcard_amp.img` file will be generated under the `work` directory, and it can be directly written to the SD card to boot. For net boot, the other images are:

   ```
   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. If you only modify the RT-Thread, you can compile the `visionfive2_fw_payload_amp.img` independently:

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

3. If you want to recompile the AMP image, execute the following command:

   ```
   $ make amp_img
   ```

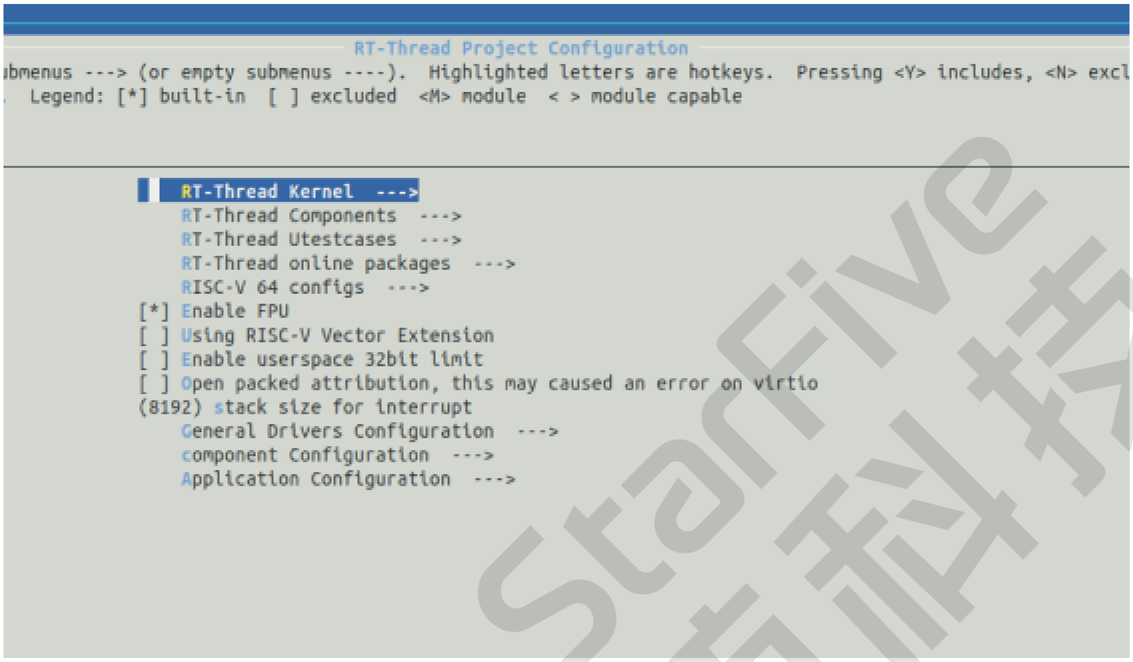
4. To configure the crop RT-Thread, enter the following command under JH-7110 directory:

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

Tip:

Menuconfig is a graphical configuration tool that is a feature in RT-Thread 3.0 and above versions. It allows for free cropping of kernels, components, and software packages, allowing the system to be built in a building block manner.

Figure 5-1 RT-Thread Configuration



6. The Components and Drivers of RT-Thread

This section introduced the components and drivers of RT-Thread:

- RPMsg *(on page 13)*
- GPIO Driver *(on page 15)*
- GMAC Driver *(on page 15)*
- PCIe Driver *(on page 16)*

6.1. RPMsg

Perform the following steps to run AMP image and test:

1. Connect Linux and the debug serial port *(on page 8)* of RTOS, set the baud rate to115,200.
2. Flash the compiled file `sdcard_amp.img` into SD card.
3. Power on: RT-Thread starts quickly after power on and runs the program. When running to the `main` program, RT-Thread needs waiting for the Linux side to send an IPI interrupt, while the Linux side is the master of Rpmsg and needs to configure the control memory and shared memory of the virtual queue.

   Figure 6-1 Power on

```
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. Boot Linux: During the process of booting Linux, the `virtio_rpmsg_bus` driver and the `starfive_rpsmg` driver will be registered. After registration is completed, an IPI interrupt will be sent to RT-Thread.

Figure 6-2 Open 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:
8 RPMsg device nodes are registered in the figure and can support multiple applications for data inter-communication.

After receiving an IPI interrupt, `rpmsg_linux_test` will continue to execute, and at this point, the finish shell of RT-Thread can also be used normally.

Figure 6-3 RT-Thread Process

```
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. Running the following command on the Linux side can see the IPI interrupt sent by RT-Thread to Linux:

```
cat /proc/interrupts
```

Figure 6-4 IPI Interrupt

```
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. Run the test program below:

```
rpmsg_echo
```

Tip:
RVspace has provided the [compiled applications and source code](). This application sends a string to the remote side of RPMsg, After receiving it, RT-Thread will send the received string back to Linux, and the test result is shown below:

Figure 6-5 Test Result

```
# ./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 interrupt:

Figure 6-6 IPI interrupt

```
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 Driver

The RTOS supports GPIO driver. Under RT-Thread finsh, enable pin command can manipulate the output and input of GPIO, taking GPIO47 as an example:

```
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
```

Result: You can check the high and low voltage levels by measuring GPIO47.

RTOS supports GPIO interrupt drivers. Since there is only one GPIO interrupt resource, and it is assumed to be occupied by Linux, RTOS can only use GPIO interrupt when Linux is not using it. In VisionFive 2 SDK, due to the use of GPIO interrupts on the Linux side, RT-Thread does not activate GPIO interrupts by default. If GPIO interrupts need to be activated, `BSP_USING_GPIO` needs to be enabled in the RT-Thread configuration.

6.3. GMAC Driver

RT-Thread supports the lwIP TCP/IP protocol stack and allows network programming on it. In AMP SDK, GMAC1 is moved to the RTOS side by default while GMAC drivers are ported to RT-Thread. Therefore, the programming of network applications can be done on RT-Thread of JH-7110, and it can also be used as a real-time network card to run a small EtherCAT master station.

As shown in the figure below, when RT-Thread starts, it initializes GMAC and registers the GMAC driver with the lwIP TCP/IP protocol stack. DHCP function is enabled by default, and the IP address is successfully obtained from the DHCP network. GMAC can work normally.

Figure 6-7 GMAC Driver

```
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
```

At the same time, it can access to the external network and check the network port status, as shown in the following figure:

Figure 6-8 Ping Website

```
msh />
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 Port Status

```
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 Driver

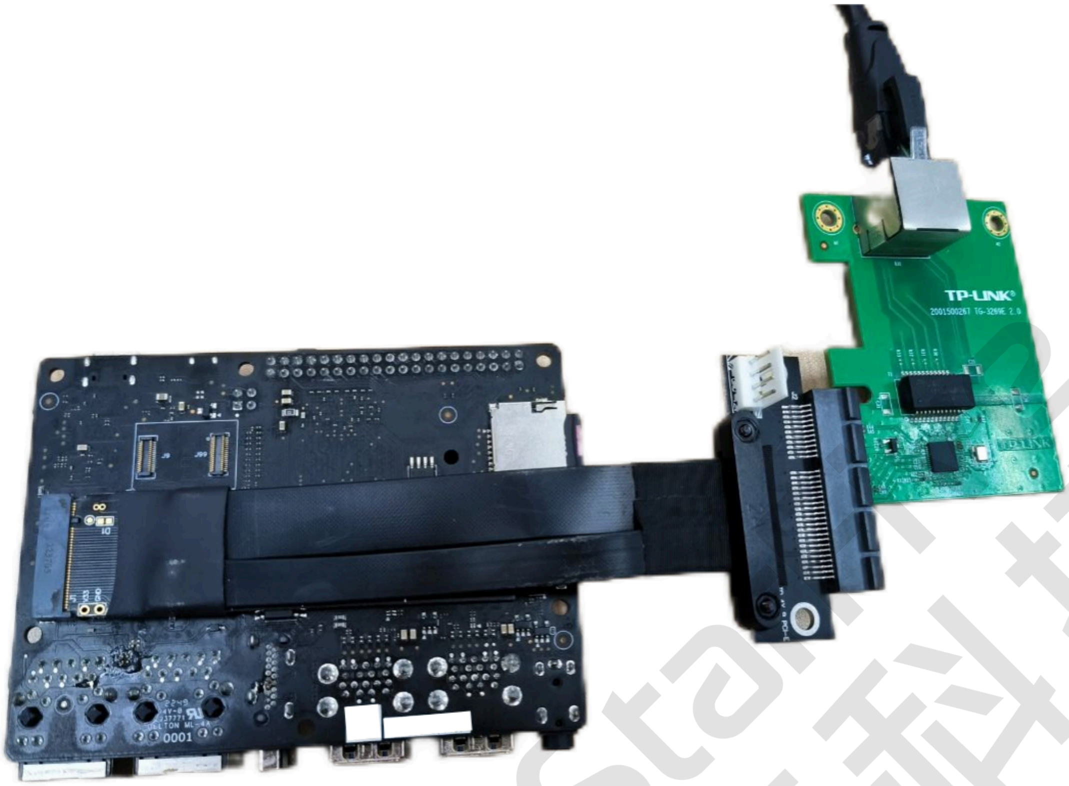
In industrial scenarios, to form a SoC+FPGA design, PCIe expansion network cards or PCIe EP devices (connected to FPGA via PCIe) are required. FPGA is used for data collection and may have real-time requirements for communication with SoC. JH-7110 includes two PCIe 2.0 hosts, and a single PCIe 2.0 theoretically supports a maximum speed of 500MB/s, with a wide range of industrial applications. Therefore, StarFive ported PCIe drivers to the RTOS side and connected PCIe network cards to verify the PCIe drivers on the RTOS.

Tip:

At present, the driver only supports PCIe bus single device, but does not support the multi-device driver under PCIe connected to switch.

The PCIe1 on VisionFive 2 uses an M.2 Key interface, which can be extended to a PCIe interface through a patch cord. PCIe can be verified by connecting to an RTL816*X* series PCIe network card, and the RTL816*X* driver is also supported on the RTOS. In the following demo, it can be seen that the RT-Thread supports dual gigabit network cards of GMAC1 and RTL8161 for JH-7110.

Figure 6-10 PCIe Driver



The following figure shows the PCIe driver initialization printing and RTL8161 network card initialization process. After PCIe initialization, it will scan devices on the PCI bus and be able to scan network card devices. After matching the ID, the RTL8161 network card driver will perform initialization and apply for MSI interrupt. As shown in the figure below, the GE network card has been initialized and registered, and the IP address has been successfully applied and PCIe network card can work normally.

Figure 6-11 Initialization Process

```
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
```

At the same time, both network cards can get IP address and access the network.

- Get IP address:

Figure 6-12 Get IP Address

```
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
```

- Access the network:

Figure 6-13 Access the Network

```
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 Performance

This section introduces the performance of RT-Thread from the following two aspects:

- Schedule Delay *(on page 19)*
- Interrupt Delay *(on page 19)*

7.1. Schedule Delay

Perform a schedule delay test similar to cyclictest under RT-Thread, and the following are the test conditions:

- **U74 main frequency:** 1.5GHz
- **Running time in idle state:** 12 hours

Result: The average delay is less than 1us, and the maximum delay is 2us.

When working with an Ethernet card and in mixed mode, the maximum latency is 10us.

7.2. Interrupt Delay

Interrupt delay can be divided into IPI interrupt delay and peripheral delay.

UART Interrupt Delay

Test the RX delay of UART at 1.5GHz, from **receive > interrupt > finish shell process receives characters**, the whole process takes about 6us.