StarFive
赛昉科技

在昉·星光 2上运行AMP双系统（RT-Thread + Linux）

版本：1.0

日期：2024/01/17

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。上海赛昉科技有限公司 （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： 浦东新区盛夏路61弄张润大厦2号楼502，上海市，201203，中国

网站： http://www.starfivetech.com

邮箱：

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

目录

表格清单

插图清单

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。

修订历史

表 0-1 修订历史

版本	发布说明	修订
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：

- **提示：**
  建议如何在某个主题或步骤中应用信息。
- **注：**
  解释某个特例或阐释一个重要的点。
- **重要：**
  指出与某个主题或步骤有关的重要信息。
- **警告：**
  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。
- **警告：**
  表明某个操作或步骤可能导致物理伤害或硬件损坏。

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。

昉·惊鸿-7110包含4个U74的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U74跑Linux操作系统，1个U74跑RT-Thread RTOS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。

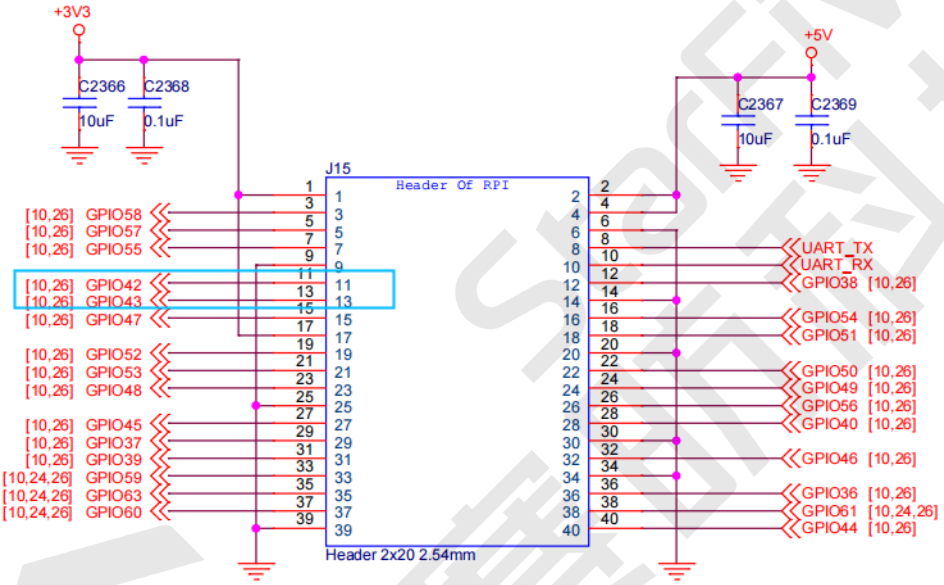
2. RT-Thread调试串口

Linux沿用UART0作为系统串口，而RT-Thread使用UART1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)：**UART1 RX
- **Pin13 (GPIO43)：**UART1 TX

图 2-1 40-Pin Out



3. 核间通信方式

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。

- Linux：在Linux内核代码中，RPMsg的代码主要位于`drivers/rpmsg/`下，相关的代码是：

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

- 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。RT-Thread代码路径为：

```
bsp/starfive/jh7110/driver/rpmsg_lite
```

3.1. 代码分支

AMP修改了U-Boot、OpenSBI、Kernel仓库，下图为五个仓库的地址和分支：

表 3-1 代码分支

仓库	地址	分支
Visionfive	https://github.com/starfive-tech/VisionFive2.git	rtthread_AMP
U-Boot	https://github.com/starfive-tech/uboot.git	
OpenSBI	https://github.com/starfive-tech/opensbi.git	
Kernel	https://github.com/starfive-tech/linux.git	
RT-Thread	https://github.com/starfive-tech/rt-thread.git	

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.img`，SPL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：

表 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 - 0×403fffff	2M	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0×40400000，然后迁移到0×6e800000地址，0×40400000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：

表 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

请按照以下步骤进行编译：

1. RT-Thread是从scons编译， 编译之前先执行以下命令，安装scons：

```
sudo apt-get install scons
```

2. 执行以下命令， 下载昉·星光 2的github 分支：

```
$ git clone https://github.com/starfive-tech/VisionFive2.git
$ cd VisionFive2
$ git checkout --track origin/rtthread_AMP
$ git submodule update --init --recursive
```

3. 执行以下命令，把相关的仓库切换到`rtthread_AMP`分支：

```
$ cd buildroot && git checkout --track origin/JH7110_VisionFive2_devel
 && cd ..
$ cd u-boot && git checkout --track origin/rtthread_AMP && cd ..
$ cd linux && git checkout --track origin/rtthread_AMP && cd ..
$ cd opensbi && git checkout rtthread_AMP && cd ..
$ cd soft_3rdpart && git checkout JH7110_VisionFive2_devel && cd ..
$ cd rtthread && git checkout rtthread_AMP && cd ..
```

4. 编译RT-Thread用到了嵌入式的`riscv64-unknown-elf`工具链，已经上传到RT-Thread仓库的toolchain文件夹（https://github.com/starfive-tech/rt-thread/tree/rtthread_AMP/toolchain）下（`toolchain/tool-root1.tar.gz`），请执行以下命令，把它复制到`/opt`文件夹下，并解压缩：

```
$ sudo tar xf rtthread/toolchain/tool-root1.tar.gz -C /opt/
```

5. 编译步骤不变，在`visionfive`文件夹下运行`make`即可，最后编译出的`visionfive2_fw_payload.img`超过了4M，刷写到SPI Nor时要注意镜像的大小。

6. 如仅修改了RT-Thread，可以单独编译RT-Thread。到`jh7110`文件夹下，运行`scons`生成`rtthread.bin`文件：

```
$cd rtthread/bsp/starfive/jh7110
$scons
```

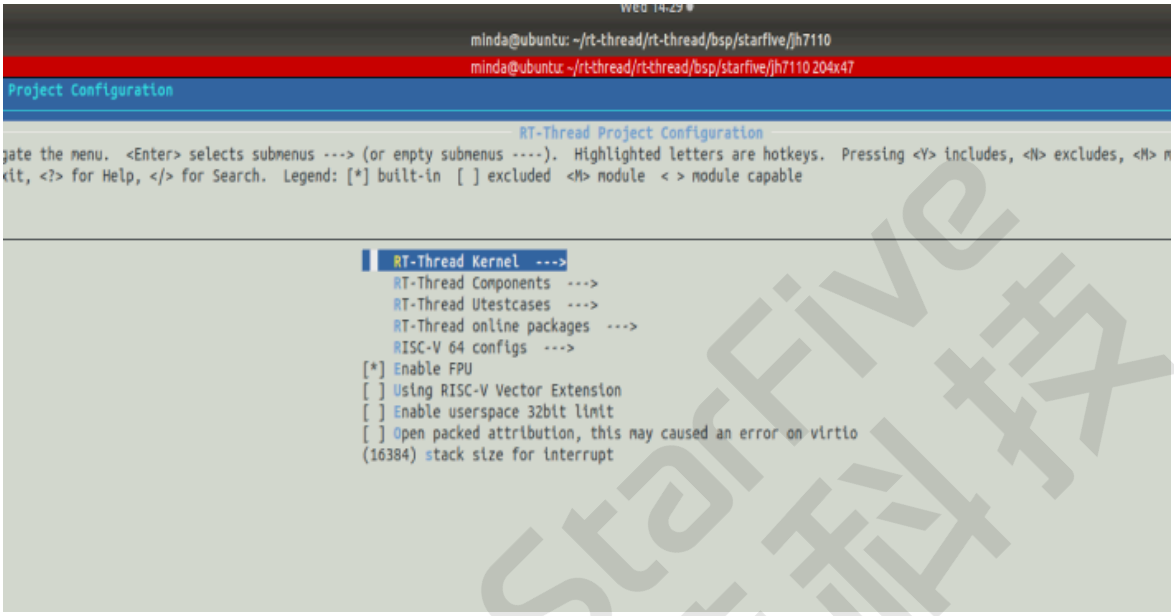
7. 如需配置裁剪RT-Thread，在`jh7110`目录下，输入以下命令：

```
$ scons --menuconfig
```

提示：

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。

图 5-1 配置RT-Thread



6. 运行RT-Thread

按照以下步骤，启动RT-Thread：

1. 连接Linux和RTOS的调试串口 (第 8页)，串口的波特率均设置为115,200。
2. 将编译出来的`u-boot-spl.bin.normal.out`和`visionfive2_fw_payload.img`文件刷写到SPI NOR FLASH上。
3. 上电启动：RT-Thread启动很快，并且运行rpmsg linux test的测试程序，RT-Thread在等待Linux端发送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。

图 6-1 上电启动

```
msh />: OpenSBI v1.2
SBI Specification Version: 1.0
heap: [0x6f000000 - 0x70000000]

 \ | /
- RT -     Thread Operating System
 / | \     5.1.0 build Dec  6 2023 15:05:01
 2006 - 2022 Copyright by RT-Thread team
lwIP-2.0.3 initialized!
Hello RISC-V
Hello Starfive RT-Thread! CPU_ID(4)
rpmsg linux test: receive data from linux then send back
rpmsg remote: remote core cpu_id-4
rpmsg remote: shmem_base-0x6e410000 shmem_end-0x6e7fffff
```

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

图 6-2 启动Linux

```
[    6.079393] virtio_rpmsg_bus virtio0: rpmsg host is online
[    6.082402] virtio_rpmsg_bus virtio0: creating channel rpmsg_chrdev addr 0x4004
[    6.084864] starfive-rpmsg soc:rpmsg@0: registered virtio0
```

RT-Thread接受到IPI中断后，`rpmsg_linux_test`会继续执行，这时RT-Thread的finsh shell也能正常使用。

图 6-3 RT-Thread进程

```
Hello starfive RT-Thread! CPU_ID(4)
rpmsg linux test: receive data from linux then send back
rpmsg remote: remote core cpu_id-4
rpmsg remote: shmem_base-0x6e410000 shmem_end-0x6e7fffff
rpmsg remote: link up! link_id-0x0
msh />
msh />
msh />ps
thread           pri  status      sp     stack size max used left tick   error  tcb addr
---------------- ---  ------- ---------- ---------- ------  ---------- ------- ----------------
rpmsg_linux_test   9  suspend 0x00000270 0x00001000    15%   0x00000014 EINTRPT 0x000000006f004d18
tshell            20  running 0x00000290 0x00001000    51%   0x00000009 OK      0x000000006f003458
tcpip             10  suspend 0x000001e0 0x00000800    28%   0x00000014 EINTRPT 0x000000006f002388
etx               12  suspend 0x00000178 0x00000800    19%   0x00000010 EINTRPT 0x000000006e84ab70
erx               12  suspend 0x00000188 0x00000800    19%   0x00000010 EINTRPT 0x000000006e84b530
tidle0            31  ready   0x000002c0 0x00004000    06%   0x00000002 OK      0x000000006e84c6f0
timer              4  suspend 0x00000108 0x00004000    05%   0x00000009 OK      0x000000006e850b30
msh />
```

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

```
cat /proc/interrupts
```

图 6-4 IPI中断

```
 59:          0          0          0  13040000.gpio  15 Edge
IPI0:        89         46         91  Rescheduling interrupts
IPI1:       834       1898       1311  Function call interrupts
IPI2:         0          0          0  CPU stop interrupts
IPI3:      2570       1800        153  IRQ work interrupts
IPI4:         0          0          0  Timer broadcast interrupts
IPI5:         1          0          0  AMP rpmsg interrupts
```

6. 运行以下测试程序:

```
rpmsg_echo
```

提示：

该程序由创龙科技提供。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:

图 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:

```
cat /proc/interrupt
IPI5: 12 0 0 AMP rpmsg interrupts
```

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。

7.2. 中断延时

中断延时分为IPI中断延时和外设延时。

IPI 中断延时

由于IPI中断需要经过M mode核间中断来发送，因此需要切换到M mode来发送，有一定延时，比RT-Thread的外设延时要大。

在`rpmsg_echo.c`进行性能测量，统计Linux用户态一个rpmsg echo、十多个字符串round trip time时间：

- 测试时长：数个小时
- 主频：1.5GHz
- 测试次数：20000多次
- 一个IPI来回：25us左右
- 最大延时：70us左右

```
Sending message #21998: hello there 21998!
Receiving message #21998: test this time 24000 ns, avg time 24785 ns,
 maxtime 69500 ns
```

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。