Router with a RISC-V CPU

Rocket Chip 在 AX7021 上的适配和路由器实验

陈嘉杰　霍江浩　赵博文　张宇翔

计算机科学与技术系

Table of Contents

- 我们目标是什么
- Rocket Chip 适配
- 路由器实现
- OS 与路由器交互

Beamer 模板由 @scateu (Kang Wang) 友情提供

自我介绍

- 陈嘉杰，计 72
- Email: noc@jiegec.ac.cn
- 大家都认识我了
- 没有梗了

希望大家

- 随时提问，这次讲的内容由和上次只有很少的关联性
- 随时质疑，因为我写的真的很粗暴，很多东西都是我第一次写
- 提供帮助，特别感谢张宇翔，教会了我这次讲的内容的 80%，剩下 20% 他也会

背景故事

- 2019 年 3 月 29 日 Tunight
- 讲 rCore 讲了很多 OS 相关的内容 + 部分外设（网卡、块设备）
- 契机 - 数字逻辑设计开始进入实验，暑假参加龙芯杯
- 动力 - 课改，实现硬件的路由器

双线开发

- 一条线是把 rCore port 到 Rocket Chip 平台上（OS 课）
  - FPGA 是 xc7z020clg484-2 ，AX7021 开发板
  - Rocket Chip 有 Zedboard 的参考设计（ucb-bar/fpga-zynq）
- 另一条线是实现纯硬件的路由器（数字逻辑设计）
  - AX7021 有四个网口，四个 Microrel 的 PHY，四路 RGMII
  - 实现 ARP 表、IP 转发和 RIP 状态机
  - 后续：thinrouter by 张宇翔

适配 Rocket Chip

- 编写语言：Chisel3 in Scala
- 直接抄 ucb-bar/fpga-zynq 的代码中 Zedboard 的设计
- 一开始不敢动配置，觉得学习成本过高
- 捣腾外部设备

锅 #1

- 综合出来的 bitstream 烧不进去
  - FPGA 不同，虽然是同一个型号，但是 speed type 不同
- 信号输入输出不干活
  - IO constraints 没有进行相应配置，从 AX7021 参考工程中复制
- FSBL U-Boot Linux ??
  - 一开始完全没搞懂这些是什么关系
  - 可以用的：Alinx 的 FSBL 和 U-Boot 、fpga-zynq 仓库的 PetaLinux
  - 坑：U-Boot 版本不同、镜像版本不同

PS 上的系统

1. 从 UCB BAR 提供的原版 U-Boot 镜像中提取 Kernel 和 ramdisk
2. 用 image.its 指定新格式打包内核 +DeviceTree+ramdisk 到一个文件中。

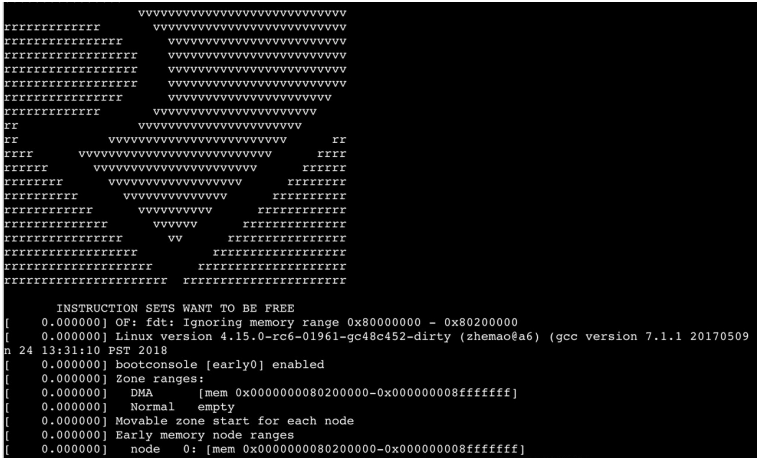
这里的 image.its 是新版 U-Boot 一种 image 的描述语言（和 dts 相同）

启动 → Alinx 提供的 FSBL → Alinx 提供的 U-Boot → UCB BAR 提供的 Linux

再烧自己的 bitstream: cat xxx.bit > /dev/xdevcfg

锅 #2

此时，按照 UCB BAR 给的样例文件，已经可以在 Rocket Chip 里跑起 bbl + Linux 了：

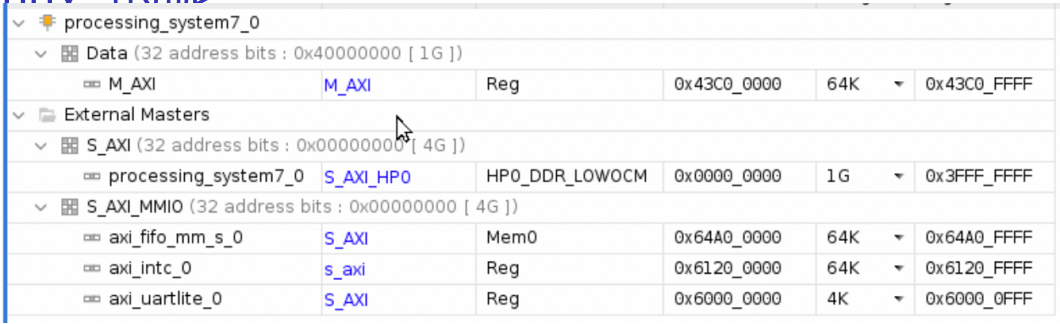


但是问题来了：怎么把 rCore 跑起来?

bbl 老矣，OpenSBI 更加科学，开始适配 OpenSBI ，照着 bbl 的代码和其他 platform 写。

OpenSBI 十分科学，平台无关的和平台有关的代码分离的很清楚。

PS 是如何控制 PL 中的 Rocket Chip 让它跑起 Linux 的呢



processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
M_AXI	M_AXI	Reg	0x43C0_0000	64K	0x43C0_FFFF
External Masters					
S_AXI (32 address bits : 0x00000000 [4G])					
processing_system7_0	S_AXI_HP0	HP0_DDR_LOWOCM	0x0000_0000	1G	0x3FFF_FFFF
S_AXI_MMIO (32 address bits : 0x00000000 [4G])					
axi_fifo_mm_s_0	S_AXI	Mem0	0x64A0_0000	64K	0x64A0_FFFF
axi_intc_0	s_axi	Reg	0x6120_0000	64K	0x6120_FFFF
axi_uartlite_0	S_AXI	Reg	0x6000_0000	4K	0x6000_0FFF

注意这里的 M_AXI 地址：0x43C0_0000。用户态程序通过打开 /dev/mem 访问物理地址 0x43C0_0000，在 AXI 总线上，这个地址会对应到 Rocket Chip 端的 0x0000_0000 地址。

这个地址有若干的寄存器：

- 调试的串口
- 自定义的 Block Device
- 自定义的 Network

这里只用第一条，通过这个串口，fesvr 通过调试的方法就写入了程序并执行代码。

但是，那个串口只是用来调试的，输入输出怎么办？HTIF.

What is HTIF?

就是在 ELF 里定义两个 symbol：fromhost 和 tohost，然后轮询里面存储的消息。

它不同的位存了不同的信息：设备（8bit)、命令（8bit）和返回值（48bit)。比如写串口就是：

```
__set_tohost(1, 1, ch);
```

特殊地，在一些模式下，可以填写 syscall 号，可以 proxy kernel。

你可能会问：这里的内存地址是 Rocket Chip 的地址空间，那外面怎么读？

答：从调试接口读内存地址。它可以往内核里插入任意代码执行。

这样输入输出都可以通过这个接口完成了。但是没有中断，只能轮询。

rCore 移植

输入输出部分已经在 OpenSBI 部分移植好了，接下来就是把 rCore 也丢进去了。

一开始遇到了一个很玄学的问题：只要启用页表，就会 fault 。后来发现把 PMP 权限放开就好了，但实际上页表的地址并不在原来受保护的 PMP 内存范围里。怀疑是 Rocket Chip 的实现问题，这个问题在 HiFive Unleashed 上也可以复现。

这个时候就可以正常跑起来 rCore 了。由于我们之前适配过 HiFive Unleashed ，基本没有什么问题。

Interleave 一下路由器

讲了这么多 Rocket Chip 的事情，现在按照时间顺序，讲讲路由器。这两件事基本是同一时间并发地在搞的。

那路由器大概要怎么搞呢？首先四个网口，那我先做两个网口做测试。首先需要做的事情：

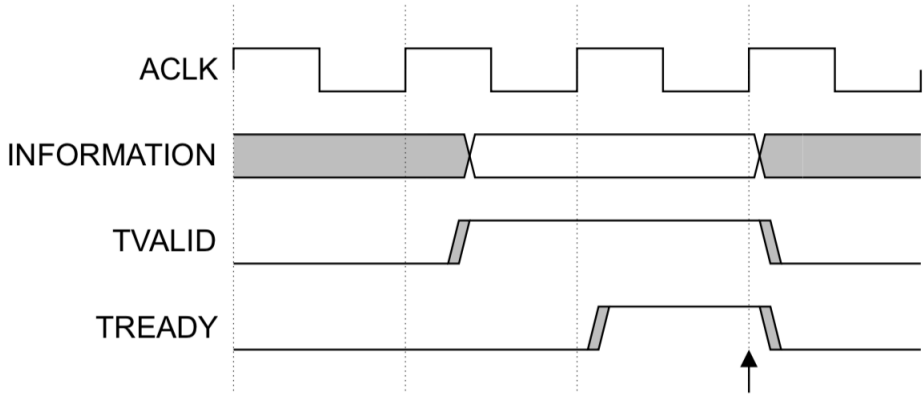
1. 调通 RGMII 接口，能够收和发
2. 把收到的数据存下来，然后转出去
3. 收的同时记录一下 ARP 表
4. 如果收到了 ARP request，回复一个 ARP reply

RGMII

RGMII (Reduced Gigabit Media Independence Interface) 是一种 MAC 和 PHY 之间的接口规范。它支持十兆、百兆和千兆，分别用不同速率来传。其中千兆是 DDR 信号，其余不是。

一开始尝试自己写，在模拟器上是可以正常工作，但是到了真板子上就是不行，怀疑是各种延迟没有调好，毕竟 DDR 信号，频率又高。我们尝试过用示波器来看，但示波器的带宽也不大够，看不出太多有用的东西。

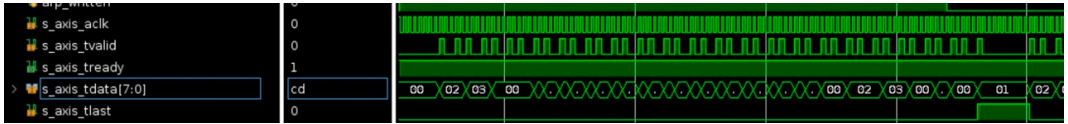
最后退而求其次采用了 Xilinx 的 Tri Mode Ethernet Mac 。它提供的是 AXI4-Stream 的接口，这就灰常友善了



这就是 AXI4-Stream 接口，没有显示出来的还有 tlast 等。

看起来很简单是吗?

不，也有很迷的：



毕竟标准也没有要求 tvalid 一定是连续的，tlast 只能一个周期为高嘛。

对于这个友善的接口来写 verilog 就比较简单了。很快就自己写了一个 loopback ，仿真过了，但是实际跑的时候不 work。原因是时钟域不同，宇翔教我，这里要用 Async FIFO。

于是给每个网口加了一个 TX 一个 RX 的 Async FIFO，内部都是同一个时钟。于是第一个 loopback 的“路由器”就做出来了。

ARP Table

还没写，瞎讲