

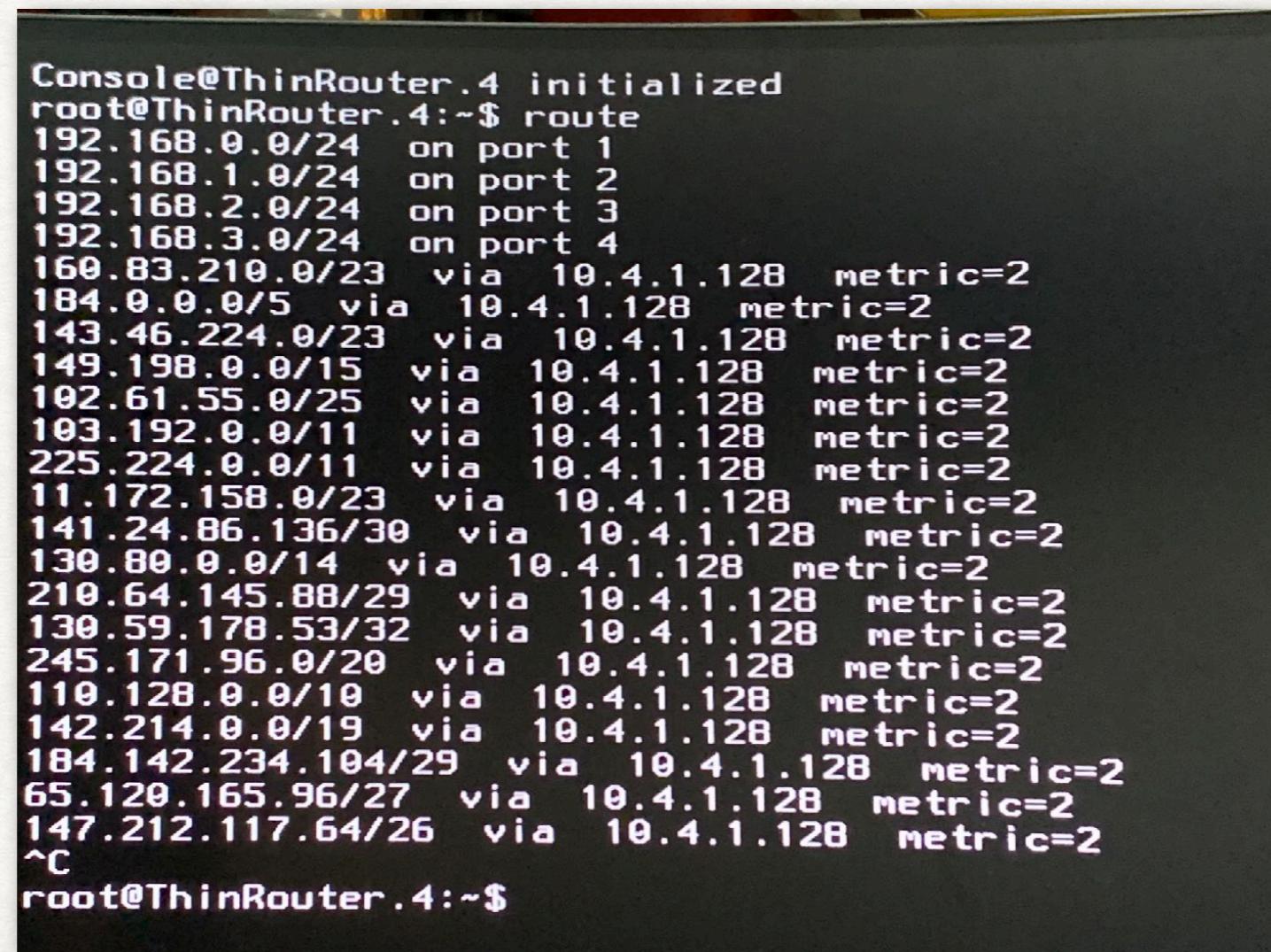
CPU@THINROUTER

计网联合实验 第 4 组
涂轶翔 王征翊 赵成钢

2019.12

我们做了什么

- 假的 ThinDOS



```
Console@ThinRouter.4 initialized
root@ThinRouter.4:~$ route
192.168.0.0/24    on port 1
192.168.1.0/24    on port 2
192.168.2.0/24    on port 3
192.168.3.0/24    on port 4
160.83.210.0/23   via 10.4.1.128 metric=2
184.0.0.0/5        via 10.4.1.128 metric=2
143.46.224.0/23   via 10.4.1.128 metric=2
149.198.0.0/15    via 10.4.1.128 metric=2
102.61.55.0/25    via 10.4.1.128 metric=2
103.192.0.0/11    via 10.4.1.128 metric=2
225.224.0.0/11    via 10.4.1.128 metric=2
11.172.158.0/23   via 10.4.1.128 metric=2
141.24.86.136/30  via 10.4.1.128 metric=2
130.80.0.0/14     via 10.4.1.128 metric=2
210.64.145.88/29  via 10.4.1.128 metric=2
130.59.178.53/32  via 10.4.1.128 metric=2
245.171.96.0/20   via 10.4.1.128 metric=2
110.128.0.0/10    via 10.4.1.128 metric=2
142.214.0.0/19    via 10.4.1.128 metric=2
184.142.234.104/29 via 10.4.1.128 metric=2
65.120.165.96/27  via 10.4.1.128 metric=2
147.212.117.64/26 via 10.4.1.128 metric=2
^C
root@ThinRouter.4:~$
```

整体情况

- MIPS32 CPU
 - @clk_20M (不编译路由可以到~50M)
 - 经典五级流水线
 - 中断 + 异常
 - 实现了编译器编译出来的全部常见指令，可以启 C 程序
- 路由器
 - 三级流水线 (流水线技术的应用)

整体情况

- 实现了一个 BootROM (BlockRAM) , 无需烧指令到 SRAM, 可以直接自启动
- BootROM 里面烧录了一个用 C 写的小命令行
- 实现了 DVI 显示
 - 支持打印字符 (26个字母 + \n + \r + Backspace) 、自动换行、滚屏、清屏
- 实现了总线
 - 挂载了BootROM / BaseRAM / ExtRAM / UART / DVI / ThinRouter

整体情况

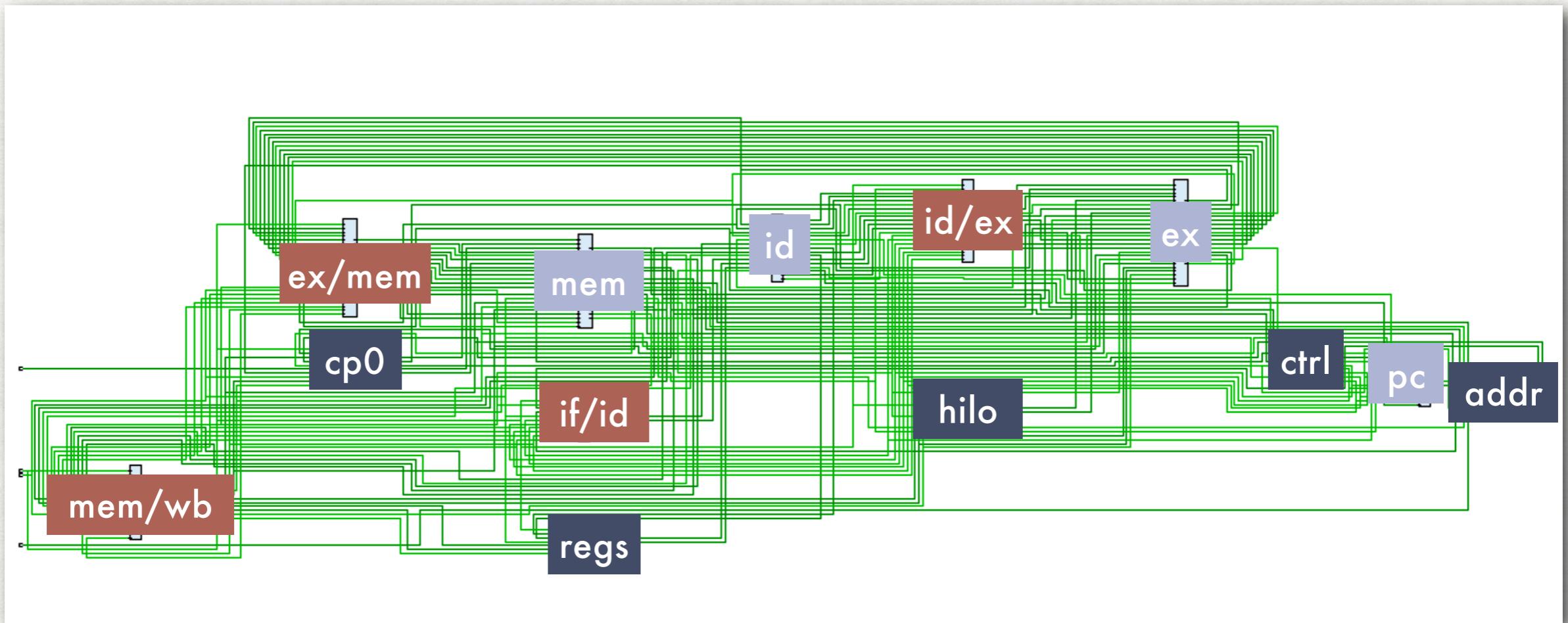
- CPU 超出正常要求的部分 / 其他亮点
 - 中断/异常 + 多实现了很多指令 + BootROM + 总线 + 交互命令行 + DVI 显示
 - 开放流程比较合规范 / 很完善的测试：3 个 CPU 的 Testbench，可以模拟和 PC 交互

整体架构 - CPU

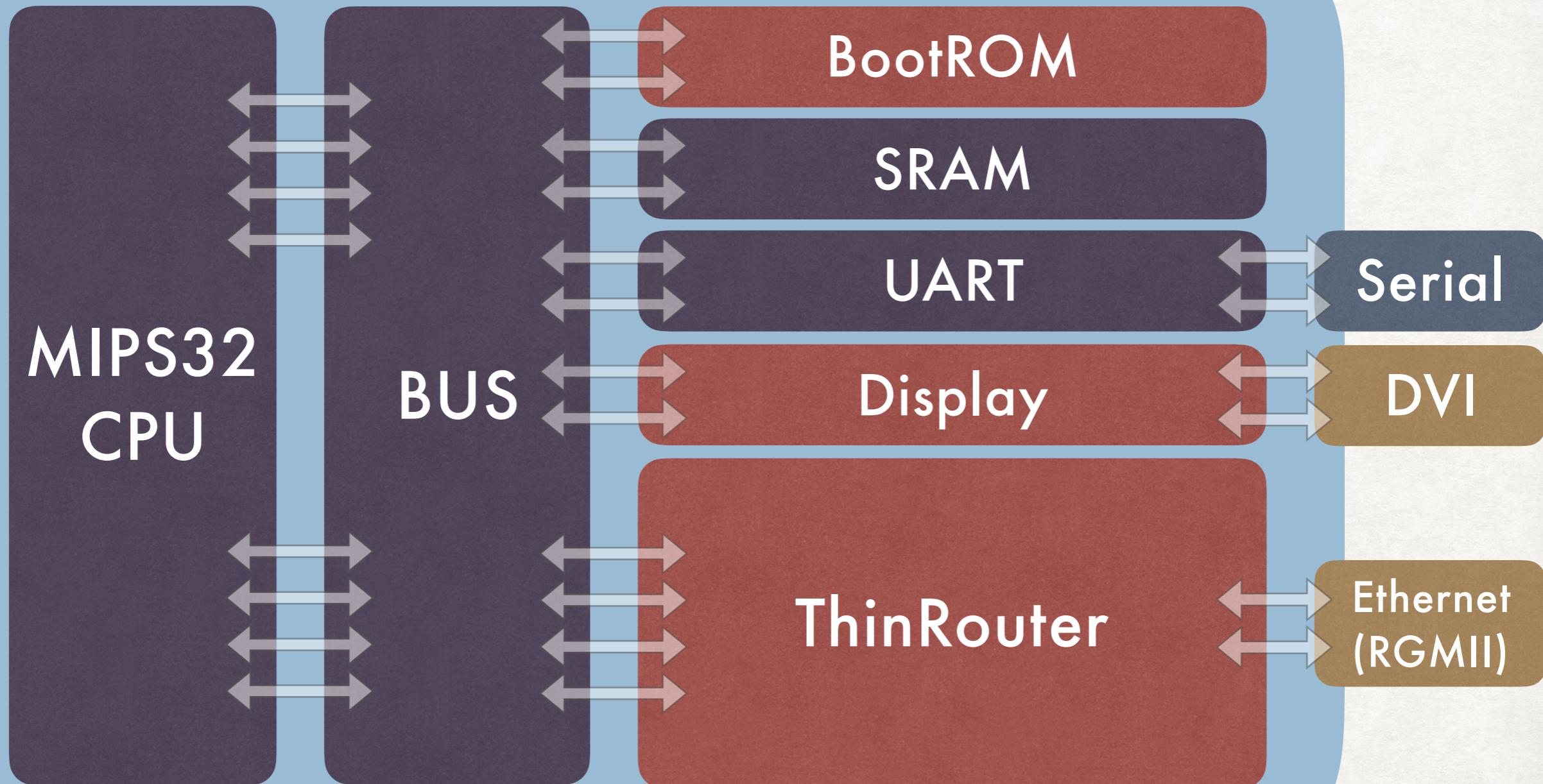
流水线：经典五级流水线

数据冲突：前传、气泡

控制冲突：跳转地址计算再译码阶段



整体架构 - THINPAD



软硬件接口

```
141     unique case (cpu_ram_addr) inside
142         // BootROM (R/O)
143     >         [32'h8000_0000 : 32'h800f_ffff]: begin...
144         end
145         // BaseRAM
146     >         [32'h8010_0000 : 32'h803f_ffff]: begin...
147         end
148         // ExtRAM
149     >         [32'h8040_0000 : 32'h807f_ffff]: begin...
150         end
151         // 串口数据
152     >         32'hbfd003f8: begin...
153         end
154         // 串口状态 (R/O)
155     >         32'hbfd003fc: begin...
156         end
157         // 路由器内存 (R/O)
158     >         [32'hc000_0000 : 32'hc00f_ffff]: begin...
159         end
160         // 路由表指针 (R/O)
161     >         32'hc011_4514: begin...
162         end
163         // 路由器读取数据 (R/O)
164         // fifo rd_en 拉高一拍：读取当前数据，fifo 出口更新下一条数据
165     >         32'hc011_4518: begin...
166         end
167         // 路由器读取数据状态 (R/O)
168         // 最低位为 1 表示有数据可读
169     >         32'hc011_451c: begin...
170         end
171         default: begin end
172     endcase
173 
```

```
// 路由器内存 (R/O)
[32'hc000_0000 : 32'hc00f_ffff]: begin
    case (cpu_ram_addr[3:2])
        0: cpu_ram_data_r <= router_mem_data[31:0];
        1: cpu_ram_data_r <= router_mem_data[63:32];
        2: cpu_ram_data_r <= {24'h0, router_mem_data[71:64]};
        3: cpu_ram_data_r <= '0;
    endcase
end
```

BOOTROM 里面的内容

```
console > ASM linker.ld.S
You, 6 days ago | 1 author (You)
1 ENTRY(_start);

2
3 MEMORY
4 {
5     BOOTROM  (rx)    : ORIGIN = 0x80000000, LENGTH = 4K
6     RAM      (rwx)   : ORIGIN = 0x80400000, LENGTH = 4M
7 }
8
9 # define TEXT_AREA BOOTROM
10 # define DATA_AREA RAM
11
12 SECTIONS {
13     . = ORIGIN(TEXT_AREA);
14
15     _mem_start = ORIGIN(DATA_AREA);
16     _mem_end = ORIGIN(DATA_AREA) + LENGTH(DATA_AREA);
17
18     _mem_avail_start = ORIGIN(DATA_AREA);
19     _mem_avail_end = ORIGIN(DATA_AREA) + LENGTH(DATA_AREA);
20
21     .text :
22 {
23         _text = .;
24         *(.text.startup)
25         *(.text*)
26         *(.rodata*)
27         *(.reginfo)
28         *(.init)
29         *(.stub)
30         *(.gnu.warning)
31         *(.MIPS.abiflags)
32         _text_end = .;
33 } > TEXT_AREA
34 }
```

```
console > ASM startup.S
You, 8 days ago | 1 author (You)
1 .globl _start
2 .section .text.startup
3 .set noreorder
4
5 .org 0x0
6 _start:
7     # setup stack pointer
8     la $sp, _stack
9     la $gp, _gp
10
11    # jump to console entry
12    jal _main
13    nop
```

BOOTROM 里面的内容

- C to MIPS32 交叉编译
 - 替换了原来的监控程序
- 采用轮询的方式等待串口输入
 - 实现了getchar
- 采用轮询的方式写入串口
 - 实现了putchar, puts

```
30 // TODO: 有时间改成中断
31 static void putc(u32 data) {
32     volatile u32 stat;
33     while (true) {
34         stat = CATCH(u32, ADDR_UART_STATUS);
35         if (stat & 1) { // 可写
36             WRITE(int, ADDR_UART_DATA, data);
37             break;
38         }
39     }
40 }
```

交互体验

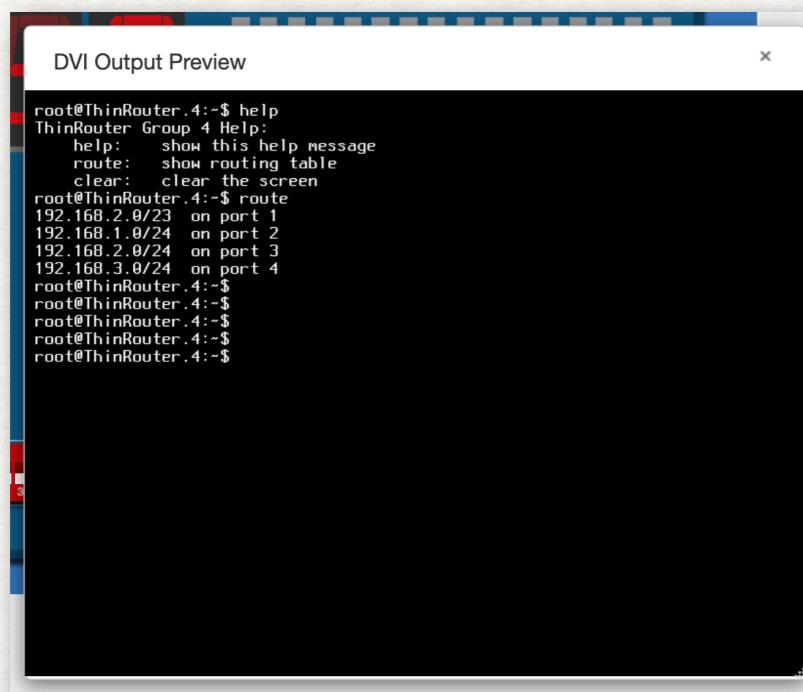
- (假的) 命令行
 - (支持显示帮助信息)
 - 支持清屏
 - 支持显示路由信息
 - 替换掉了监控程序
- 显示路由
 - 在总线上给路由器数据划分一段地址
 - C 语言直接访问地址取数
 - 用 C 简直就是想干嘛干嘛
 - 之间把路由表的 Trie 树节点取出来
 - 然后用 C 解析
- 没有实现 USB 键盘，如何交互？

DVI Output Preview

```
root@ThinRouter:~$ help
ThinRouter Group 4 Help:
    help:    show this help message
    route:   show routing table
    clear:   clear the screen
root@ThinRouter:~$ route
192.168.2.0/23  on port 1
192.168.1.0/24  on port 2
192.168.2.0/24  on port 3
192.168.3.0/24  on port 4
root@ThinRouter:~$
```

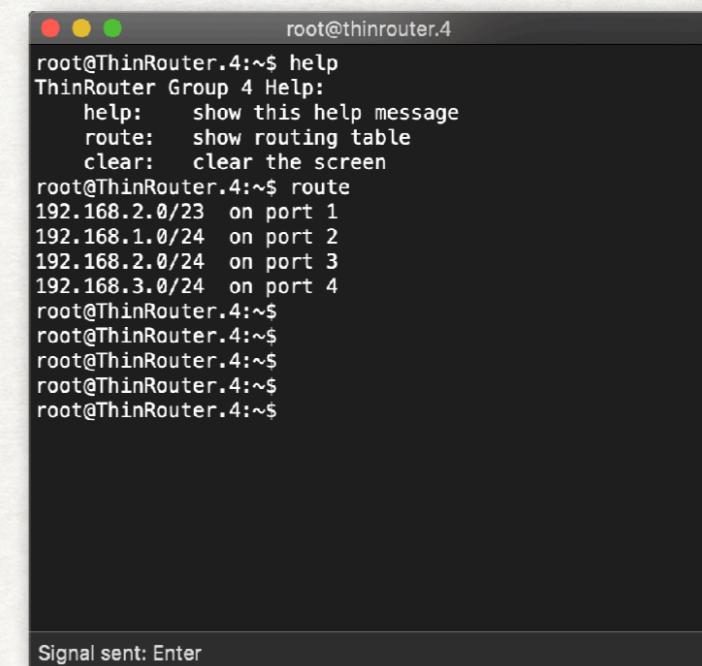
交互体验

- 命令行：PC 的键盘输入通过 UART 给 CPU
 - TCP 传给远程板子
 - UART 传给自己的板子
- 重写了 Term
 - (复习了大一小学期的 Python 和 Qt 知识)
 - 实现了一个 PyQt 的数据同传 Term



TCP
远程板子

UART
自己板子



ThinPad
(把输出信息打印屏幕 + 传回 UART)

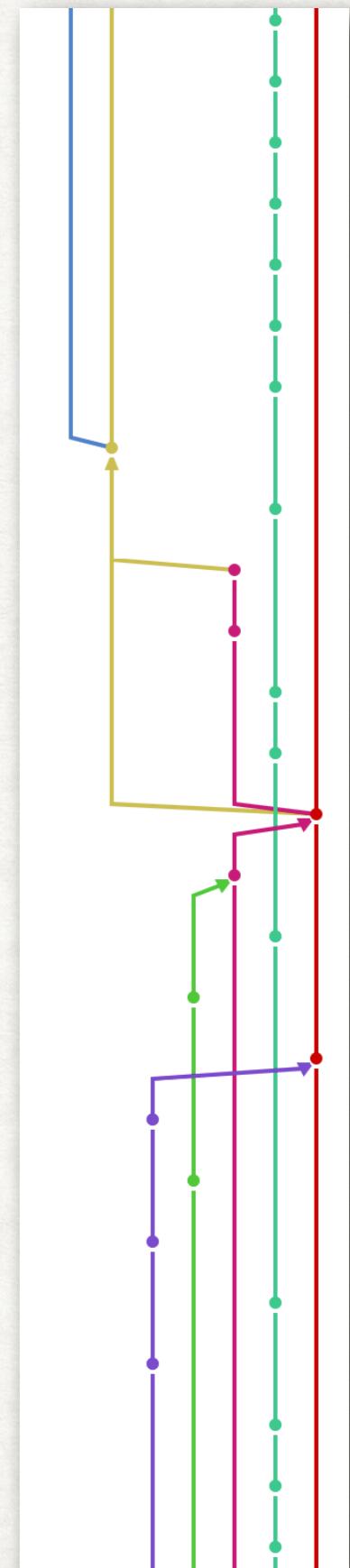
PyQt 电脑控制
(把键盘数据传给CPU + 同步显示)

交互体验



软工般的体验

- 即使计网联合，开发过程也（比较）软工
(不，我们不写文档)
- 全部代码 System Verilog
 - 学习了很多奇怪的语法和特性
 - (@Harry 说代码写得跟 C 一样)
- 开发过程比较合规范
 - 8000+ LoC (~3k Router, ~2.5k CPU, ~2.5k Tests)
 - 420 Commits, 65 Issues, 3 Milestones, 50 Merge Requests



软工般的体验

- 非常完备的测试
- 7 个 Testbench
 - CPU 自启动
 - CPU 和 Term 交互测试（可以模拟 PC 发送命令）
 - CPU 和监控程序交互测试（可以模拟 PC 发送命令）
 - ARP 表测试
 - 路由表测试
 - 整个 RIP 测试（先模拟其他路由发 RIP，然后 ping）
 - RIP 打包测试
- 5 个 Python 测例生成器可生成任意规模测例
 - 生成 ARP 插入/删除请求
 - 生成路由表插入/删除/查询请求
 - 生成以太网帧给 RIP 测试
 - 生成监控程序的命令模拟 PC 发送给 CPU
 - 生成 Term 的命令模拟 PC 发送给 CPU

软工般的体验

- 并非为了写而写，而是很大程度上提高了 Debug 效率
- 包括用 enum 代替 hardcoded 等技巧可以直接在仿真看到每个周期在运行的指令的名字
- Testbench 可以模拟 PC 和监控程序 / 自制命令行交互
- 路由器的 Testbench 直接会 Assert 结果对不对

```
RECV: b5 (e)
RECV: 64 (d)
RECV: 2e (.)
SEND: Command G
SEND: Addr (G): 0x8000200c
RECV: (G) Start running
RECV: Running (G): 4f (0)
RECV: Running (G): 4b (K)
RECV: (G) End running
```

Tcl Console x Messages Log

295570 ns get 209.109.53.124
correct
1275.
295590 ns query 24.239.110.236
295730 ns get 0.0.0.0
correct
1276.
295750 ns insert 194.88.0.0/13 -> 4.114.148.154
296250 ns insert done
1277.
296250 ns insert 136.74.224.0/21 -> 38.199.197.42
296850 ns insert done
1278.



分工

- 涂轶翔：路由器实现，路由器 CPU 连接，DVI 字符矩阵显示，部分单元测试
- 王征翊：CPU 基础功能，RIP Response 封装模块
- 赵成钢：CPU 基础功能，中断/异常，用户交互 term，部分单元测试

整体情况

- MIPS32 CPU
 - @clk_20M (不编译路由可以到~45M)
 - 经典五级流水线
 - 中断 + 异常
 - 实现了编译器编译出来的全部常见指令，可以启 C 程序
- 路由器
 - 三级流水线 (流水线技术的应用)
- 实现了一个 BootROM (BlockRAM) , 无需烧指令到 SRAM, 可以直接自启动
- BootROM 里面烧录了一个用 C 写的小命令行
- 实现了 DVI 显示
 - 支持打印字符 (26个字母 + \n + \r + Backspace) 、自动换行、滚屏、清屏
- 实现了总线
 - 挂载了BootROM / BaseRAM / ExtRAM / UART / DVI / ThinRouter
- CPU 超出正常要求的部分 / 其他亮点
 - 中断/异常 + 多实现了很多指令 + BootROM + 总线 + 交互命令行 + DVI 显示
 - 开放流程比较合规范 / 很完善的测试：3 个 CPU 的 Testbench, 可以模拟和 PC 交互

谢谢

欢迎提问 @第四组