

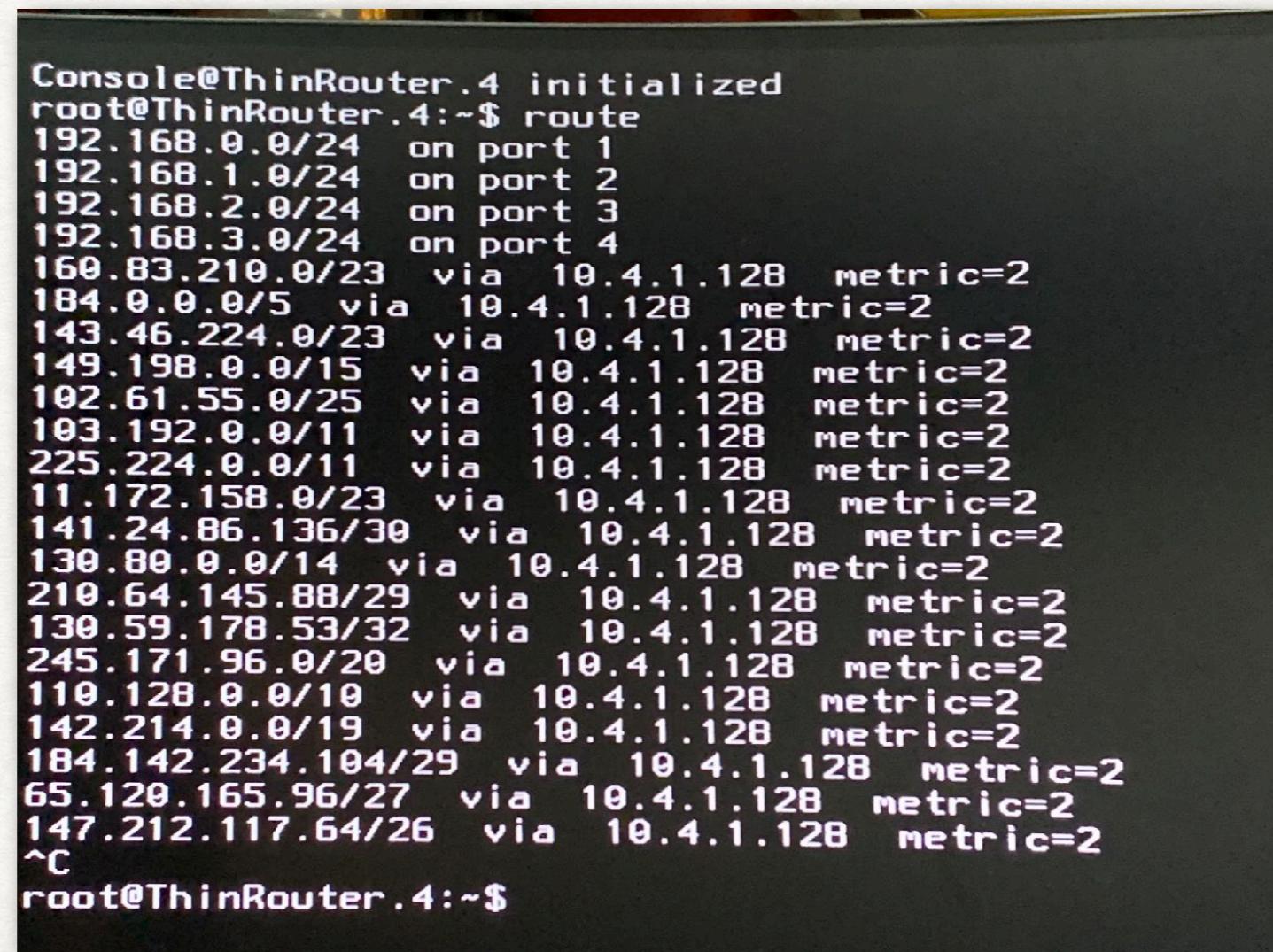
CPU@THINROUTER

计网联合实验 第 4 组

2019.12

我们做了什么

- 假的 ThinDOS

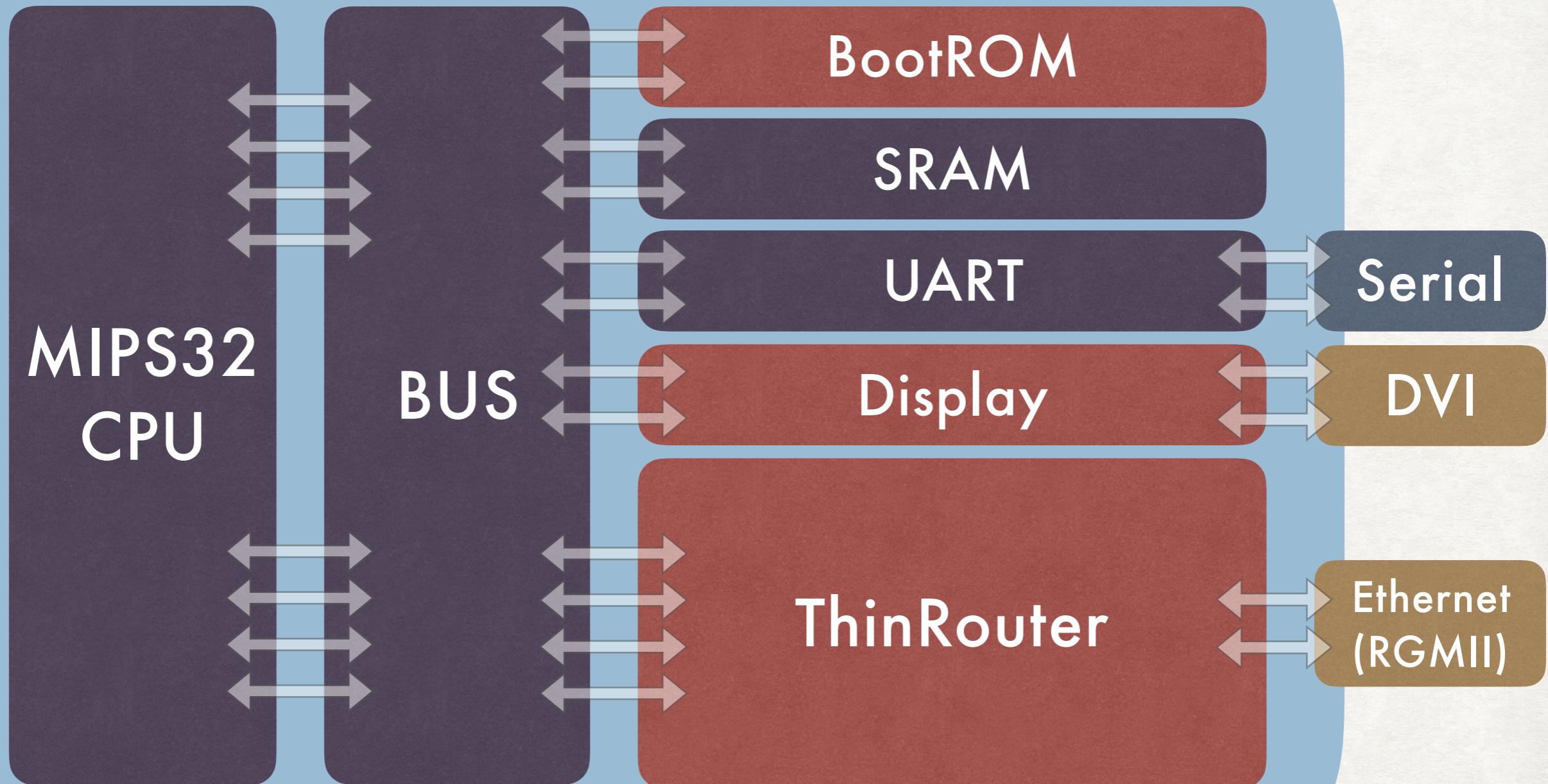


```
Console@ThinRouter.4 initialized
root@ThinRouter.4:~$ route
192.168.0.0/24    on port 1
192.168.1.0/24    on port 2
192.168.2.0/24    on port 3
192.168.3.0/24    on port 4
160.83.210.0/23   via 10.4.1.128 metric=2
184.0.0.0/5        via 10.4.1.128 metric=2
143.46.224.0/23   via 10.4.1.128 metric=2
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:~$
```

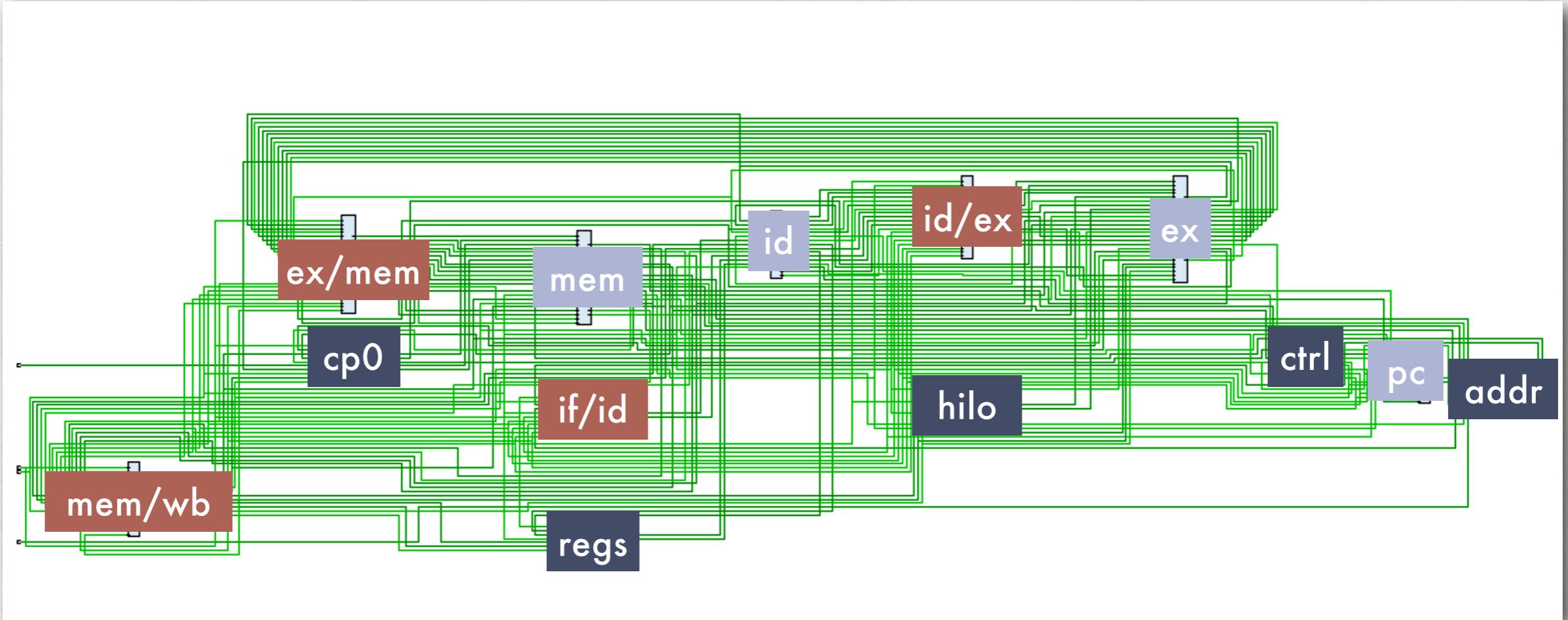
整体情况

- MIPS32 CPU Release 1
 - @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
- 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

整体架构 - THINPAD



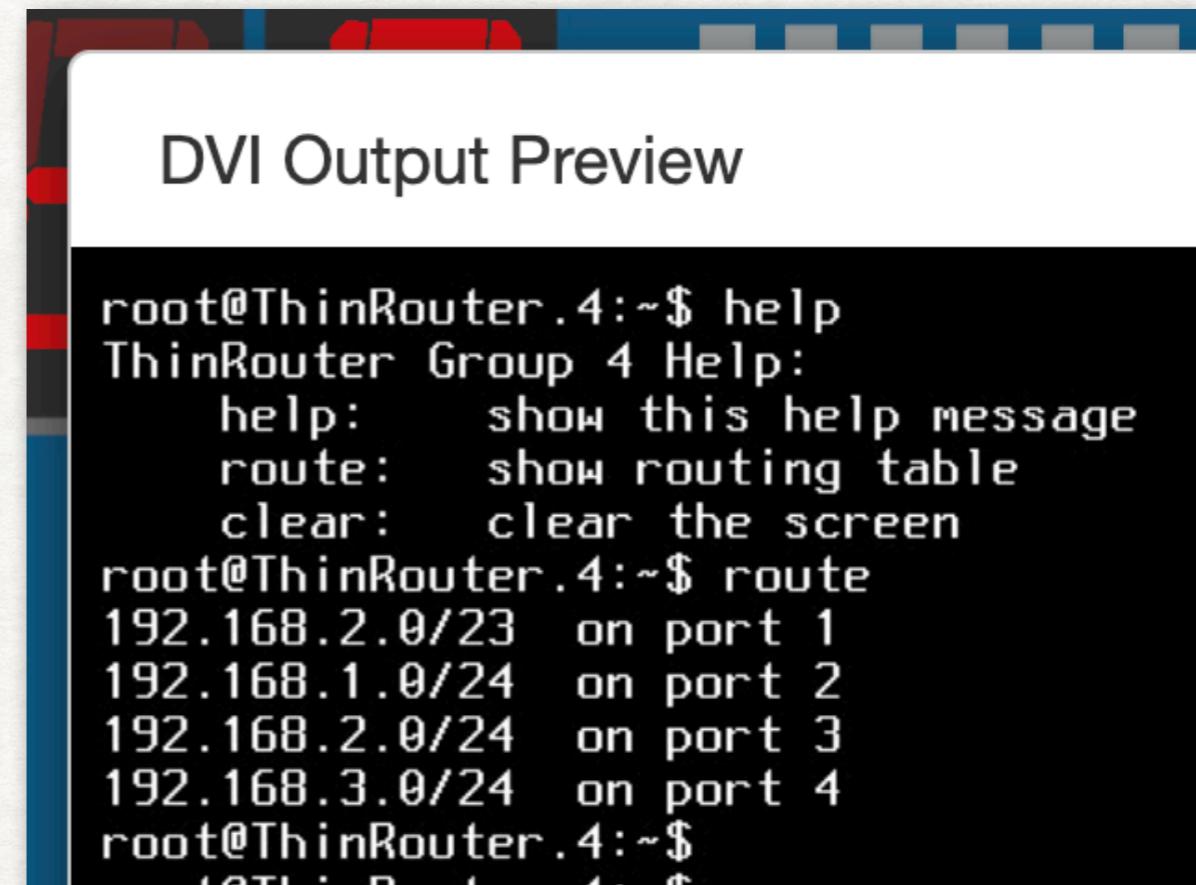
整体架构 - CPU



- 以 Load 为例

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
- 有了上面的实现
 - 可以非常容易实现👉这个假命令行



DVI Output Preview

```
root@ThinRouter.4:~$ help
ThinRouter Group 4 Help:
    help:    show this help message
    route:   show routing table
    clear:   clear the screen
root@ThinRouter.4:~$ 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.4:~$
```

交互体验

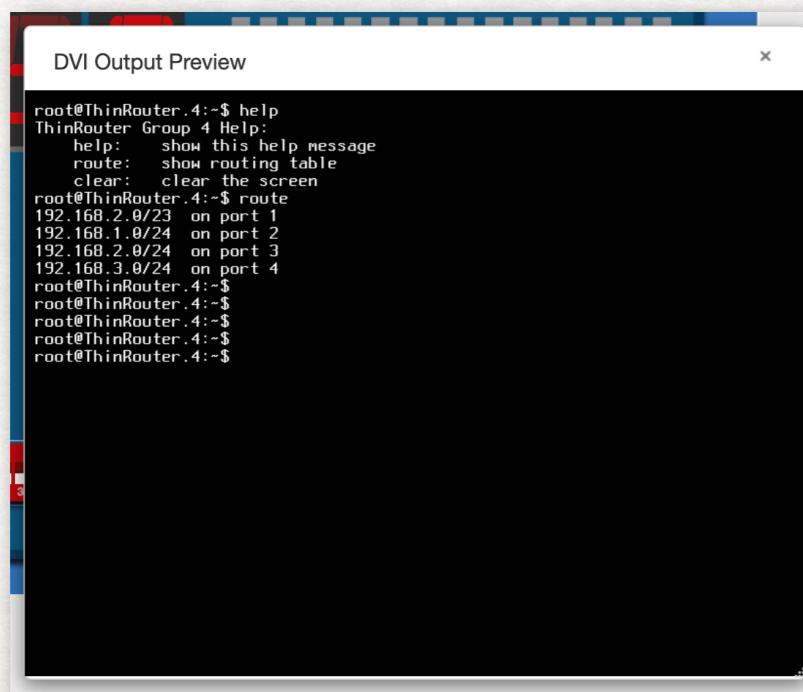
- (假的) 命令行
 - (支持显示帮助信息)
 - 支持清屏
 - 支持显示路由信息
 - 替换掉了监控程序
- 显示路由
 - 在总线上给路由器数据划分一段地址
 - C 语言直接访问地址取数
 - 用 C 简直就是想干嘛干嘛
 - 之间把路由表的 Trie 树节点取出来
 - 然后用 C 解析
- 没有实现 USB 键盘，如何交互？

```
DVI Output Preview

root@ThinRouter.4:~$ help
ThinRouter Group 4 Help:
    help:    show this help message
    route:   show routing table
    clear:   clear the screen
root@ThinRouter.4:~$ 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.4:~$
```

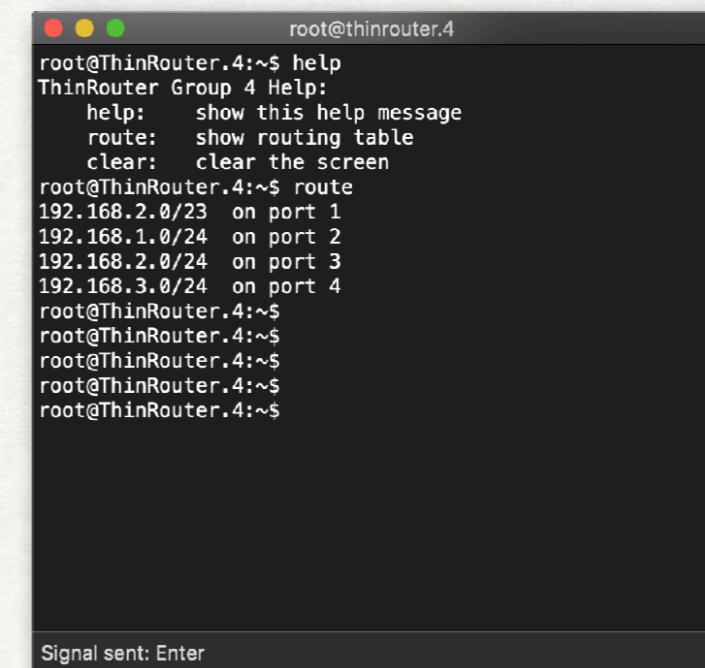
交互体验

- 用自己电脑的键盘，然后电脑通过
 - 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

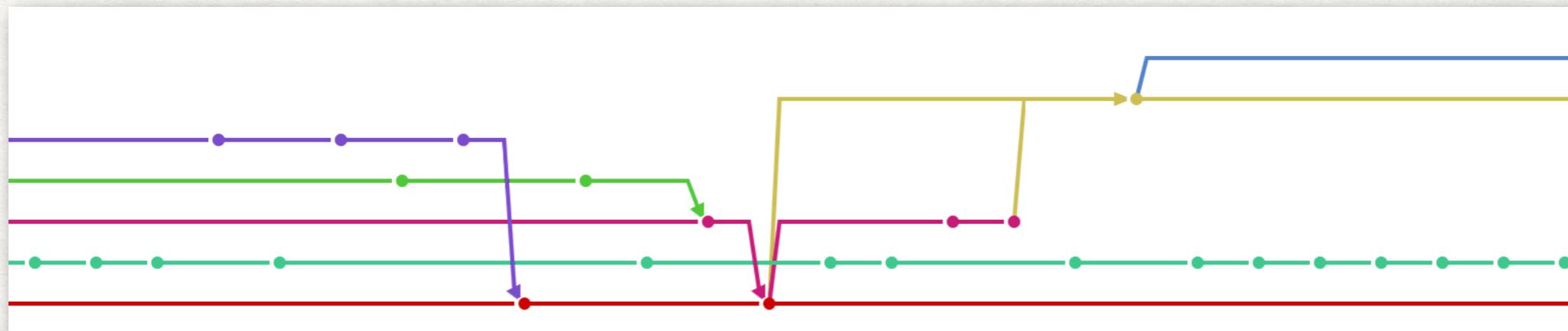
- 命令行：PC 的键盘输入通过 UART 给 CPU
- 可以通过指令查询路由表的状态
- 电脑通过一个 PyQt 的 Term 控制端同步显示数据



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

```
15      typedef enum logic [47:0] {
16          RouterMAC1 = 48'ha8_88_08_18_88_88,
17          RouterMAC2 = 48'ha8_88_08_28_88_88,
18          RouterMAC3 = 48'ha8_88_08_38_88_88,
19          RouterMAC4 = 48'ha8_88_08_48_88_88,
20          McastMAC = 48'h01_00_5e_00_00_09,
21          BcastMAC = 48'hff_ff_ff_ff_ff_ff
22      } _mac_constants;
23
24
25      function logic [2:0] port;
26          input logic [31:0] ip;
27      begin
28          case (ip[31:8])
29              RouterIP1[31:8]: port = 1;
30              RouterIP2[31:8]: port = 2;
```



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：路由器部分，DVI 字符矩阵显示，部分单元测试
- 王征翊：CPU 部分，RIP Response 封装模块
- 赵成钢：CPU 部分，计网联合部分，部分单元测试

整体情况

- MIPS32 CPU Release 1
 - @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
- 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