

# 可不可以优雅地复用NEMU的代码 (FOR NPC)

ysyx\_24090006 任勤博

## 问题

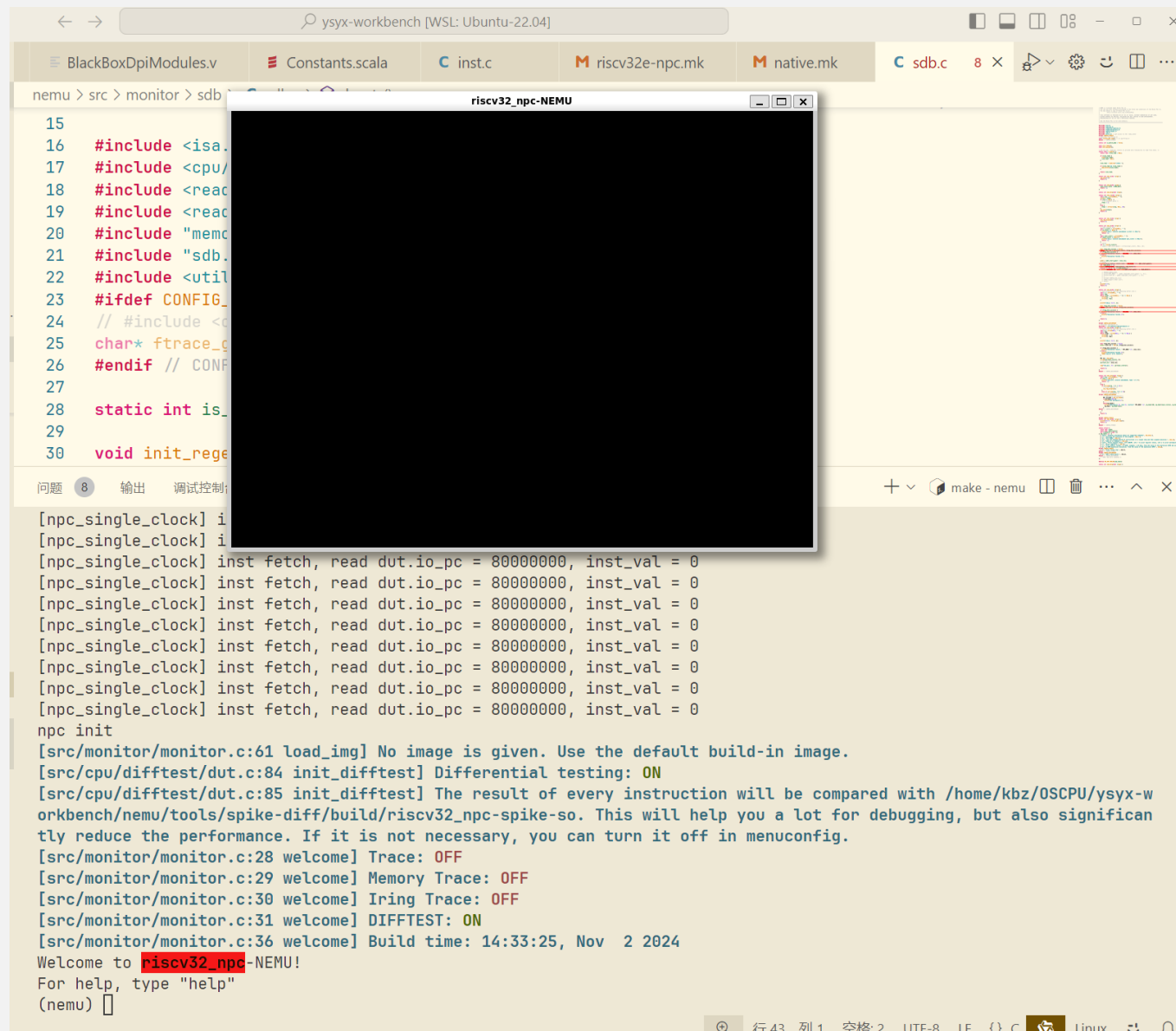
- 为NPC搭建基础设施（sdb、trace、difftest）的过程中，如何复用先前的代码？
- 另起炉灶：从头开始编写基础设置
- 循序渐进：有计划地把NEMU的代码一点一点地移过去
- 囫圇吞枣：把NEMU的大部分代码先吃进去，再考虑怎么修剪
- 移花接木：新建一个NEMU ISA，做少量的修改以连接NPC

# 启发

- GDB 远程调试 NEMU 和 NPC - 李心杨 - 一生一芯双周分享会
  - 可以通过简单的方法为软件接入强大的功能
- Picker - 用你擅长的编程语言进行芯片验证 – 万众一心开放验证
  - A codegen tool for chip verification, which can provide C++/Python/Java/Scala/Golang interfaces for the RTL designs.
  - 可以通过动态链接的方法把NPC编译成一个DUT，供其他程序使用

效果

可以直接使用NEMU提供的Sdb、Trace、Difftest、MMIO based Devices等



The screenshot displays the NEMU development environment. At the top, a browser-like window shows the file explorer with tabs for `BlackBoxDpiModules.v`, `Constants.scala`, `inst.c`, `riscv32e-npc.mk`, `native.mk`, and `sdb.c`. The main editor window shows the `monitor/sdb.c` file with the following code:

```
15
16 #include <isa.h>
17 #include <cpu.h>
18 #include <readline.h>
19 #include <readline/readline.h>
20 #include "memc.h"
21 #include "sdb.h"
22 #include "util.h"
23 #ifdef CONFIG_SDB
24 // #include <stdio.h>
25 char* ftrace_get();
26 #endif // CONFIG_SDB
27
28 static int is_debugger = 0;
29
30 void init_reg...
```

Below the code editor, the `riscv32_npc-NEMU` window is open, displaying the output of the NEMU monitor. The output shows the initialization of the NPC (NEMU NPC) and the start of the instruction stream. The output is as follows:

```
[npc_single_clock] i
[npc_single_clock] i
[npc_single_clock] inst fetch, read dut.io_pc = 80000000, inst_val = 0
[npc_single_clock] inst fetch, read dut.io_pc = 80000000, inst_val = 0
[npc_single_clock] inst fetch, read dut.io_pc = 80000000, inst_val = 0
[npc_single_clock] inst fetch, read dut.io_pc = 80000000, inst_val = 0
[npc_single_clock] inst fetch, read dut.io_pc = 80000000, inst_val = 0
[npc_single_clock] inst fetch, read dut.io_pc = 80000000, inst_val = 0
[npc_single_clock] inst fetch, read dut.io_pc = 80000000, inst_val = 0
[npc_single_clock] inst fetch, read dut.io_pc = 80000000, inst_val = 0
[npc_single_clock] inst fetch, read dut.io_pc = 80000000, inst_val = 0
npc init
[src/monitor/monitor.c:61 load_img] No image is given. Use the default build-in image.
[src/cpu/difftest/dut.c:84 init_difftest] Differential testing: ON
[src/cpu/difftest/dut.c:85 init_difftest] The result of every instruction will be compared with /home/kbz/OSCPU/ysyx-w
orkbench/nemu/tools/spike-diff/build/riscv32_npc-spike-so. This will help you a lot for debugging, but also significan
tly reduce the performance. If it is not necessary, you can turn it off in menuconfig.
[src/monitor/monitor.c:28 welcome] Trace: OFF
[src/monitor/monitor.c:29 welcome] Memory Trace: OFF
[src/monitor/monitor.c:30 welcome] Iring Trace: OFF
[src/monitor/monitor.c:31 welcome] DIFFTEST: ON
[src/monitor/monitor.c:36 welcome] Build time: 14:33:25, Nov 2 2024
Welcome to riscv32_npc-NEMU!
For help, type "help"
(nemu) 
```

# NPC中C源文件、Verilog源文件中的stdout可以直接输出

```
[src/monitor/monitor.c:29 welcome] Memory Trace: OFF
[src/monitor/monitor.c:30 welcome] Iring Trace: OFF
[src/monitor/monitor.c:31 welcome] DIFFTEST: ON
[src/monitor/monitor.c:36 welcome] Build time: 14:38:00, Nov  2 2024
Welcome to riscv32_npc-NEMU!
For help, type "help"
(nemu) si
[npc_single_clock] inst fetch, read dut.io_pc = 80000000, inst_val = 297
[src/cpu/cpu-exec.c:162 cpu_exec] nemu_state.state: 0
(nemu) si
[npc_single_clock] inst fetch, read dut.io_pc = 80000004, inst_val = 28823
=====
[Core] PC: 80000004, branch_sel=0
[Core] Inst: 00028823
[Core] ALU: 80000010, op1=80000000, op2=00000000, aluctr= 0, less=1, zero=0
[Core] Imm: 00000010
[Core] Mem: 00000000
[Core] rs1_addr:  5, rs1: 80000000
[Core] rs2_addr:  0, rs2: 00000000
[Core] rd_addr: 16, rd: 00000001, rd_sel: 0. Warning: rd_data's value is for debugging purposes only.
[dpi_mem_write] write addr = 80000010, data = 0, len = 1
NpcMemWriteModule: addr=80000010, data=00000000, len=      1
[src/cpu/cpu-exec.c:162 cpu_exec] nemu_state.state: 0
(nemu) █
```

## 实现方法

- 在/nemu/src/isa/下Fork一份riscv32的isa实现，不妨就叫riscv32\_npc
- 写一套接口，使得NPC可以作为一个NEMU的ISA实现
- 把NPC编译为动态链接库，与NEMU连接
- 略微修改一下riscv32\_npc，使得它能驱动NPC

## 定义接口

- 初始化相关
- 单步执行
- 读pc、regs
- 读NEMU上的内存（这里使用回调实现）
- Invoke nemu\_trap（这里使用回调实现）



## 修改RISCV32\_NPC

- // 在NEMU的init\_isa()初始化NPC（设置回调函数等）
- **void** init\_isa() {
- /\* Set Callbacks of NPC\*/
- npc\_set\_callback\_mem\_read(vaddr\_read);
- npc\_set\_callback\_mem\_write(vaddr\_write);
- npc\_set\_callback\_trap(nemu\_trap\_warped);
- /\* Init NPC \*/
- npc\_init();
- /\* Load built-in image. \*/
- memcpy(guest\_to\_host(RESET\_VECTOR), img, sizeof(img));
- /\* Initialize this virtual computer system. \*/
- restart();
- }

## 修改RISCV32\_NPC

- // 修改前

- `int isa_exec_once(Decode *s) {`
- `s->isa.inst.val = inst_fetch(&s->snpc, 4);`
- `return decode_exec(s);`
- `}`

- // 修改后

- `int isa_exec_once(Decode *s) {`
- `// 触发trace和disassembler`
- `s->isa.inst.val = inst_fetch(&s->snpc, 4);`
- `// 单步执行`
- `npc_exec_once();`
- `// 拷贝pc的值`
- `s->dnpc = npc_get_pc();`
- `// 拷贝寄存器的值`
- `for(size_t i = 0; i < MUXDEF(CONFIG_RVE, 16, 32); i++) {`
- `R(i) = npc_get_reg(i);`
- `}`
- `return 0;`
- `}`

## 后果

- AbstractMachine兼容问题
- 语义混淆
- NEMU的Devices可能没那么有用
- 接口地狱
- 复杂度屏蔽失败

## 兼容问题：ABSTRACTMACHINE

- 使用am-nemu和am-npc均可正常编译
- 如果使用am-nemu作为riscv32\_npc的am：
  - 没有\_\_multi3()等函数，无法算乘除法，因此甚至不能通过全部cpu-tests
- 如果使用am-npc作为riscv32\_npc的am：
  - 能通过所有cpu-tests
  - 不一定能很好地使用NEMU上的Devices
  - 奇怪的bug：重复打印

bugging, but also significantly reduce the performance. If it is not necessary, you can turn it off in menuconfig.

[src/cpu/difftest/dut.c:85 init\_difftest] The result of every instruction will be compared with /home/kbz/OSCPU/ysyx-workbench/nemu/tools/spike-diff/build/riscv32\_npc-spike-so. This will help you a lot for debugging, but also significantly reduce the performance. If it is not necessary, you can turn it off in menuconfig.

[src/monitor/monitor.c:164 init\_monitor] FTrace: No elf file path provided. ftrace disabled.

[src/monitor/monitor.c:164 init\_monitor] FTrace: No elf file path provided. ftrace disabled.

[src/monitor/monitor.c:28 welcome] Trace: ON

[src/monitor/monitor.c:28 welcome] Trace: ON

[src/monitor/monitor.c:29 welcome] Memory Trace: ON

[src/monitor/monitor.c:29 welcome] Memory Trace: ON

[src/monitor/monitor.c:30 welcome] Iring Trace: ON

[src/monitor/monitor.c:30 welcome] Iring Trace: ON

[src/monitor/monitor.c:31 welcome] DIFFTEST: ON

[src/monitor/monitor.c:31 welcome] DIFFTEST: ON

[src/monitor/monitor.c:33 welcome] If trace is enabled, a log file will be generated to record the trace. This may lead to a large log file. If it is not necessary, you can disable it in menuconfig

[src/monitor/monitor.c:33 welcome] If trace is enabled, a log file will be generated to record the trace. This may lead to a large log file. If it is not necessary, you can disable it in menuconfig

[src/monitor/monitor.c:36 welcome] Build time: 15:05:48, Nov 2 2024

[src/monitor/monitor.c:36 welcome] Build time: 15:05:48, Nov 2 2024

Welcome to riscv32\_npc-NEMU!

For help, type "help"

(nemu) a

bash nemu

Verilog ysyx-w...

make cpu-tests

## 语义混淆

```
• int isa_exec_once(Decode *s) {  
•   // 触发tracer和disassembler  
•   s->isa.inst.val = inst_fetch(&s->snpc, 4);  
  
   // 单步执行  
•   npc_exec_once();  
   // 拷贝pc的值  
•   s->dnpc = npc_get_pc();  
•   // 拷贝寄存器的值  
•   for(size_t i = 0; i < MUXDEF(CONFIG_RVE, 16,  
32); i++) {  
•       R(i) = npc_get_reg(i);  
•   }  
•   return 0;  
• }
```

大家看riscv32\_npc上面标红的那一行：我写这一行代码，仅仅是为了能够触发tracer和disassembler，而不关心它取指会取到什么值。

可以这么说：一个函数的效果体现在 a)它的返回值 b)它的副作用，而我在这写这行代码仅仅只是为了达到效果b。

这种仅仅使用函数一半效果的行为，可能会给人带来极大的困惑。

后面在实现异常处理的时候，如果复用nemu/src/isa/riscv32的代码，还会导致这样的语义混淆问题。

## NEMU的DEVICES可能没那么有用

- 要注意在ysyx后面章节，接入SOC时，要接入的外设是nvboard，而不是nemu devices，这就显得费尽周章把npc接入NEMU的收益没那么高

## 接口地狱

- sdb不能与npc直接连接，意味着做什么都要加新接口
- 比如说打印npc的引脚、打印CSRs，都要定义新接口

## 复杂度屏蔽失败

- 本意是想借助NEMU ISA提供的抽象，使得npc可以方便地复用NEMU提供的功能
- 但一旦npc需要的功能超出了NEMU ISA提供的抽象，就不得不修改NEMU框架代码以支持新的功能，复杂度因此泄露出/nemu/src/isa/riscv32\_npc这个文件夹，向整个系统传递



## 如何评价

- 另起炉灶：从头开始编写基础设置（推荐）
- 循序渐进：有计划地把NEMU的代码一点一点地移过去（推荐）
- 囫圇吞枣：把NEMU的大部分代码先吃进去，再考虑怎么修剪（一般不推荐）
- 移花接木：新建一个NEMU ISA，做少量的修改以连接NPC（不是很推荐）

总结：复杂度不会凭空消失