



香山 DiffTest 框架简介

王华强

中科院计算技术研究所

2021年7月17日

目录

- **差分测试 (difftest) 原理简介**
 - 常规指令的比对
 - 处理执行中的特殊状况
- 香山 difftest 框架介绍
 - 框架提供的接口
 - 将处理器接入 difftest 框架的方法
 - 香山 difftest 框架的实现

香山 difftest 框架

- 支持 SMP 结构的全系统仿真验证框架
 - 支持多核情景下的全系统仿真验证
- 支持多线程程序、SMP Linux 内核等 workload
- 支持检测内存一致性方面的软硬件问题

- 支持基于 Verilator 的检查点
- 支持基于 fork-wait 的检查点
 - <https://www.bilibili.com/video/BV1y54y1n74q>

Slides: <https://github.com/OpenXiangShan/XiangShan-doc/tree/main/slides>



SMP-Difftest 支持多处理器的差分测试方法

王凯帆 王华强
中科院计算技术研究所
2021年6月24日

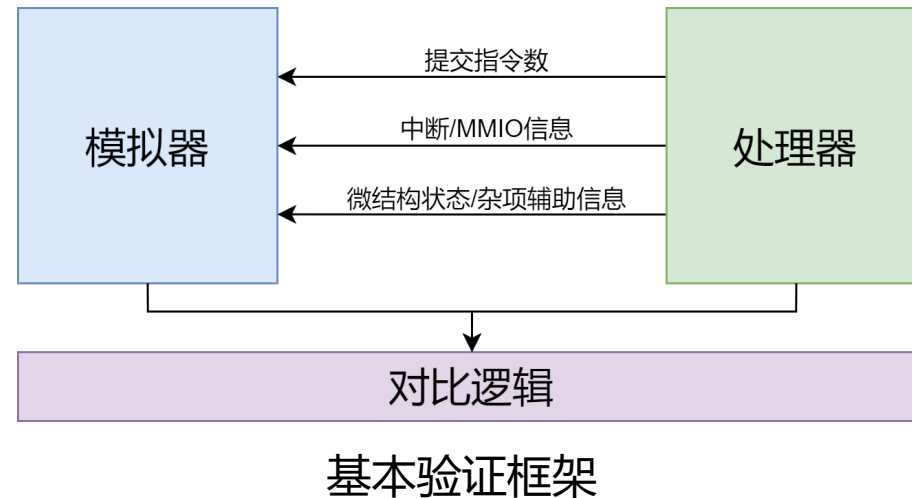
LightSSS : 基于内存的轻量级仿真快照

余子濠 金越
中科院计算所

2021.06@上海

Difftest_[1] 机制

- 香山的全系统仿真验证使用 *Difftest* 机制
- 指令级的在线仿真验证框架
- 执行流程
 - (1) 处理器仿真产生指令提交
 - (2) 模拟器执行相同的指令
 - (3) 比较两者状态



```
while (1) {  
    icnt = cpu_step();           //(1)  
    nemu_step(icnt);            //(2) [2]  
    r1s = cpu_getregs();        //(3)  
    r2s = nemu_getregs();       //(3)  
    if (r1s != r2s) { abort(); } //(3)  
}
```

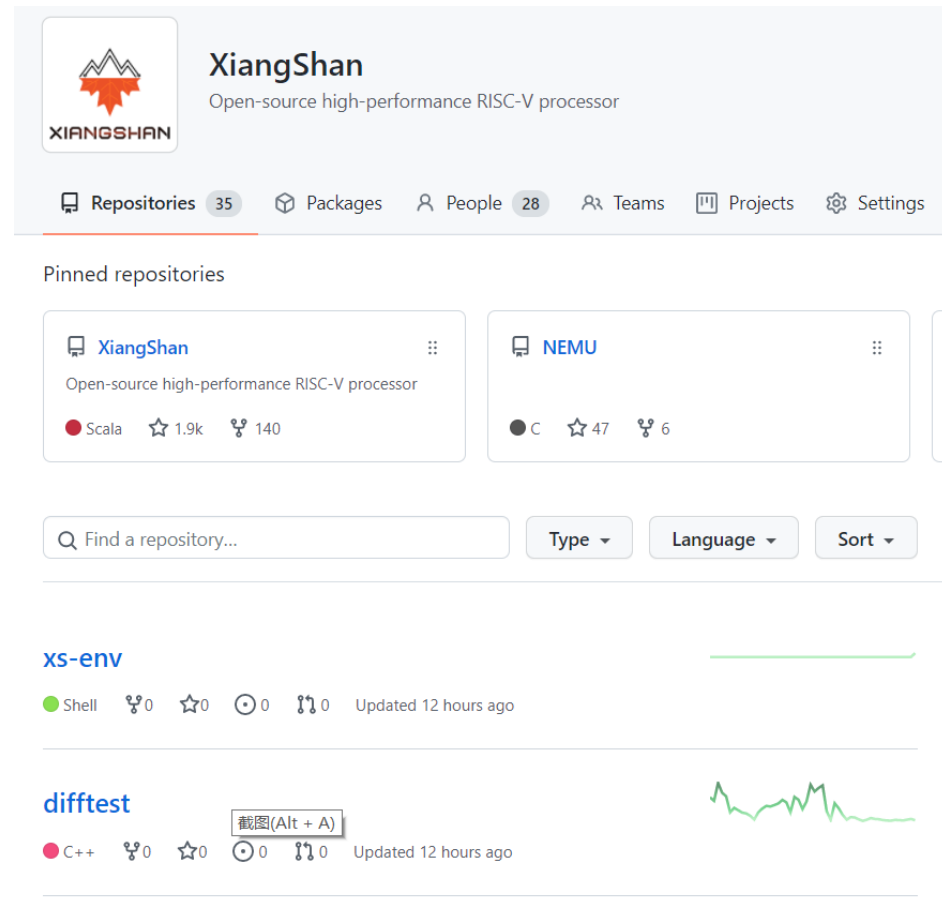
在线验证机制

[1] Yu, EasyDiff: An Effective and Efficient Framework for Processor Verification, CRVF 2019, <https://crvf2019.github.io/pdf/14.pdf>

[2] NJU Emulator (NEMU)是南京大学开发的轻量级教学用模拟器. 香山在验证中使用其作为模拟器.

香山 difftest 框架

- 独立的 difftest 框架代码地址：
<https://github.com/OpenXiangShan/difftest>
 - 这一 difftest 框架依赖特定版本的 NEMU (模拟器)
- 可以在这一仓库中找到对应版本的其他项目：
<https://github.com/OpenXiangShan/xs-env>
 - 其中的 NutShell 接入了这个 difftest 框架
 - 其中的 NEMU 与 difftest 框架匹配
 - 可以当作实现的参考



The screenshot shows the GitHub profile for XiangShan, an open-source high-performance RISC-V processor. The profile includes a repository count of 35, 28 people, and various project and settings options. Under the 'Pinned repositories' section, two repositories are highlighted: 'XiangShan' (Open-source high-performance RISC-V processor, Scala, 1.9k stars, 140 forks) and 'NEMU' (C, 47 stars, 6 forks). Below this, a search bar and filters for 'Type', 'Language', and 'Sort' are visible. Two repository cards are shown: 'xs-env' (Shell, 0 stars, 0 forks, updated 12 hours ago) and 'difftest' (C++, 0 stars, 0 forks, updated 12 hours ago). A '截图(Alt + A)' button is overlaid on the difftest repository card.

DiffTest 机制的性能表现

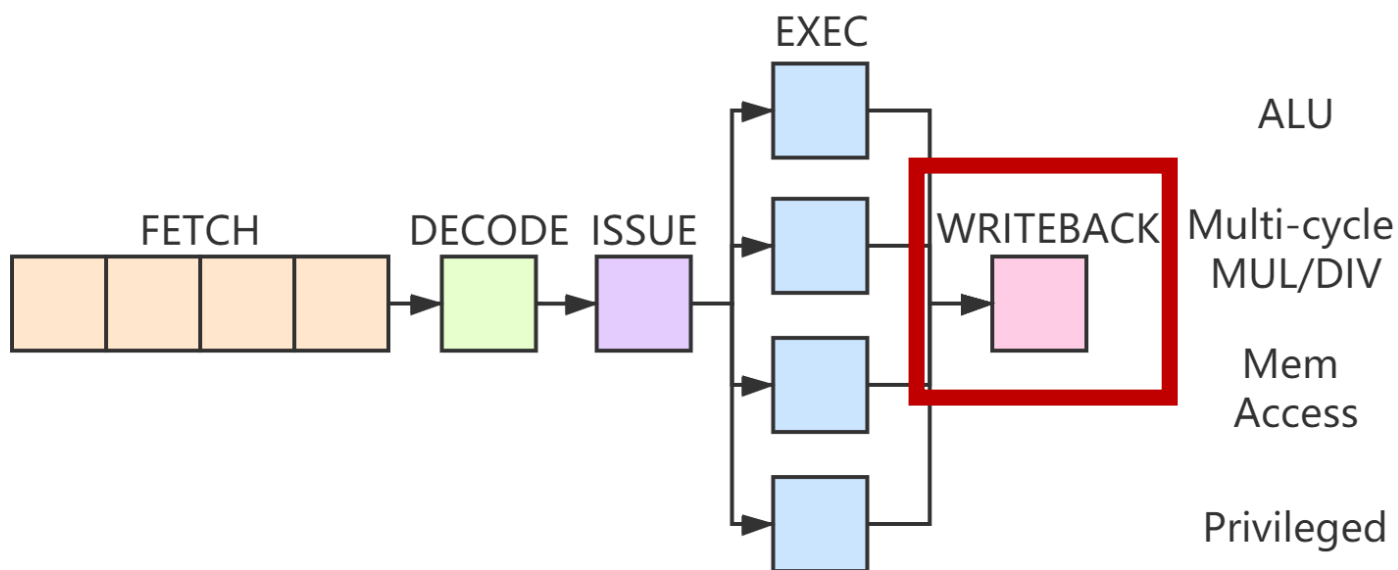
- 模拟器的速度比仿真速度快几个数量级

程序输入/每秒指令数	模拟器	处理器仿真	接入DiffTest后 处理器仿真
CoreMark	1079948	7543	7503

- 模拟器通过**动态链接**接入 DiffTest 机制, 通信开销小
- DiffTest 对仿真的速度的影响**微乎其微**

背景知识: 指令提交

- 乱序处理器的提交
- 经典顺序五级流水处理器的“提交”: 写回级
- 指令执行结果的检查在指令提交时触发



一个顺序处理器的流水级示意图, 指令提交在写回级

🌟 验证一条普通指令的执行

- 处理器将提交指令数、寄存器堆状态、PC提交给 difftest
- 模拟器执行相同数量的指令
- 比较处理器和模拟器的寄存器堆状态、PC

```
INTERFACE_INT_REG_STATE {  
    RETURN_NO_NULL  
    auto packet = difftest[coreid]->get_arch_reg_state();  
    packet->gpr[ 0] = gpr_0;  
    packet->gpr[ 1] = gpr_1;  
    // .....  
}
```

使用 DPI-C 将处理器的执行结果传递给 difftest 框架

<https://github.com/OpenXiangShan/difftest/blob/master/src/test/csrc/difftest/interface.cpp>

验证一条普通指令的执行

- 处理器将提交指令数、寄存器堆状态、PC提交给 difftest
- **模拟器执行相同数量的指令**
- 比较处理器和模拟器的寄存器堆状态、PC

```
while (num_commit < DIFFTEST_COMMIT_WIDTH && dut.commit[num_commit].valid) {  
    do_instr_commit(num_commit);  
    dut.commit[num_commit].valid = 0;  
    num_commit++;  
}
```

每个时钟周期, difftest 框架会检查提交的指令的数量, 让模拟器执行与处理器提交数量相同的指令

<https://github.com/OpenXiangShan/difftest/blob/master/src/test/csrc/difftest/difftest.cpp>

验证一条普通指令的执行

- 处理器将提交指令数、寄存器堆状态、PC提交给 difftest
- 模拟器执行相同数量的指令
- **比较处理器和模拟器的寄存器堆状态、PC**

```
proxy->get_regs(ref_regs_ptr);  
if (memcmp(dut_regs_ptr, ref_regs_ptr, DIFFTEST_NR_REG * sizeof(uint64_t))) {  
    display();  
    return 1;  
}
```

获取模拟器执行结果, 对比模拟器与处理器执行结果

<https://github.com/OpenXiangShan/difftest/blob/master/src/test/csrc/difftest/difftest.cpp>

特殊情况的处理

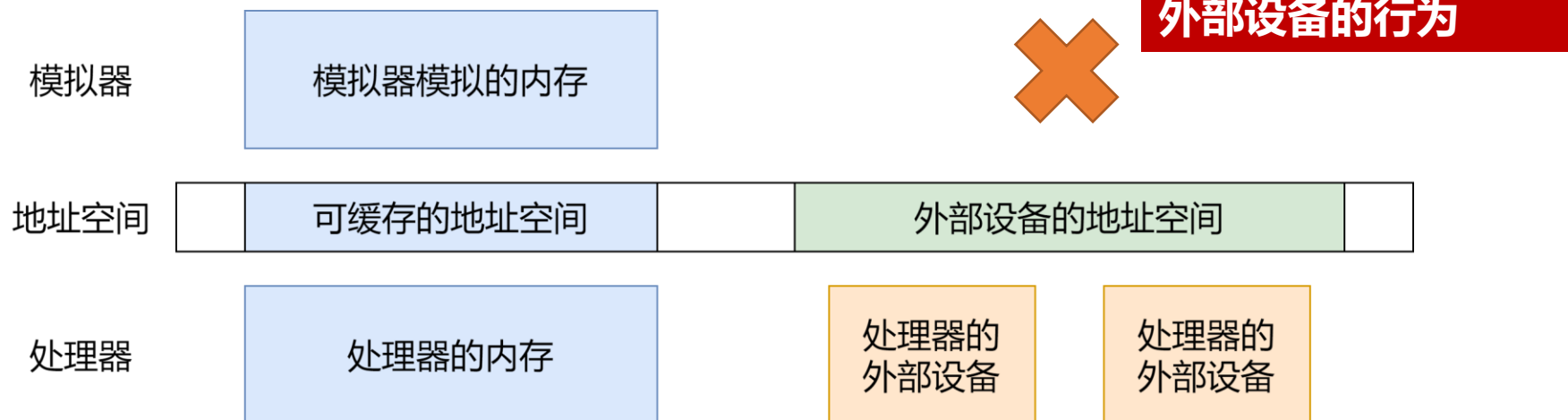
- 模拟器无法仅靠自己的一些行为上与正确的处理器对齐
- 无法依靠模拟器直接验证处理器的行为

与外部输入相关的行为	与微结构相关的行为	与一致性相关的行为
外部中断	时钟中断	LR/SC
MMIO	Store Page Fault	多核

无法依靠直接比对验证的行为, 红色的部分是一生一芯项目中需要关注的点

🏔️ 特殊情况的处理

- Memory Mapped IO (MMIO)
 - 映射到内存地址的IO指令
 - 表现为访问特定地址区域的 load / store 指令
 - 例如, 读取(内存地址映射的)时钟计数器
 - 模拟器**不能模拟**所有的外部设备
 - 模拟器无法得知这些 load 指令的正确结果



解决方案

- 上述问题的原因
 - 模拟器缺乏微结构信息
- 解决方案
 - 传递微结构状态，以同步模拟器与处理器
- 思考三个具体实现上的问题
 - **传递什么**微结构状态？
 - 模拟器**如何检查**传入的微结构状态是否合法？
 - 如何根据传入的状态**更新**模拟器状态？

特殊情况的处理

- Memory mapped IO:
 - 处理器标识出这样的访存指令
 - 在这样的指令提交时从处理器将访存结果复制到模拟器中

```
if (dut.commit[i].skip) {
    proxy->get_regs(ref_regs_ptr);
    ref.csr.this_pc += dut.commit[i].isRVC ? 2 : 4;
    if (dut.commit[i].wen && dut.commit[i].wdest != 0) {
        ref_regs_ptr[dut.commit[i].wdest] = dut.commit[i].wdata;
    }
    proxy->set_regs(ref_regs_ptr);
    return;
}
```

传递什么微结构状态?

模拟器**如何检查**传入的微结构状态是否合法?

如何根据传入的状态**更新**模拟器状态?

<https://github.com/OpenXiangShan/difftest/blob/master/src/test/csrc/difftest/difftest.cpp>

特殊情况的处理

- 外部中断 & 时钟中断:
 - 处理器传出中断信息, 模拟器进入相同的处理流程

```
if (dut.event.interrupt) {
    dut.csr.this_pc = dut.event.exceptionPC;
    do_interrupt();
} else if(dut.event.exception) {
    dut.csr.this_pc = dut.event.exceptionPC;
    do_exception();
} else {
    // 正常的处理流程
}
```

传递什么微结构状态?

模拟器**如何检查**传入的微结构状态是否合法?

如何根据传入的状态**更新**模拟器状态?

<https://github.com/OpenXiangShan/difftest/blob/master/src/test/csrc/difftest/difftest.cpp>

特殊情况的处理

- LR/SC:
 - 将SC的结果同步到模拟器, 仅允许成功 -> 失败的单向变动

```
// sync lr/sc reg status
if (dut.commit[i].scFailed) {
    struct SyncState sync;
    sync.lrscValid = 0;
    sync.lrscAddr = 0;
    proxy->set_mastatus((uint64_t*)&sync); // sync lr/sc microarchitectural regs
}
```

传递什么微结构状态?

模拟器**如何检查**传入的微结构状态是否合法?

如何根据传入的状态**更新**模拟器状态?

<https://github.com/OpenXiangShan/difftest/blob/master/src/test/csrc/difftest/difftest.cpp>

其他需要处理的杂项

- 处理第一条指令的提交
- 判断仿真是否终止
 - 处理器卡死
 - 程序执行完成
 - 处理器运行了指定的周期数
- 记录必要的信息来辅助调试

<https://github.com/OpenXiangShan/difftest/blob/master/src/test/csrc/difftest/difftest.cpp>

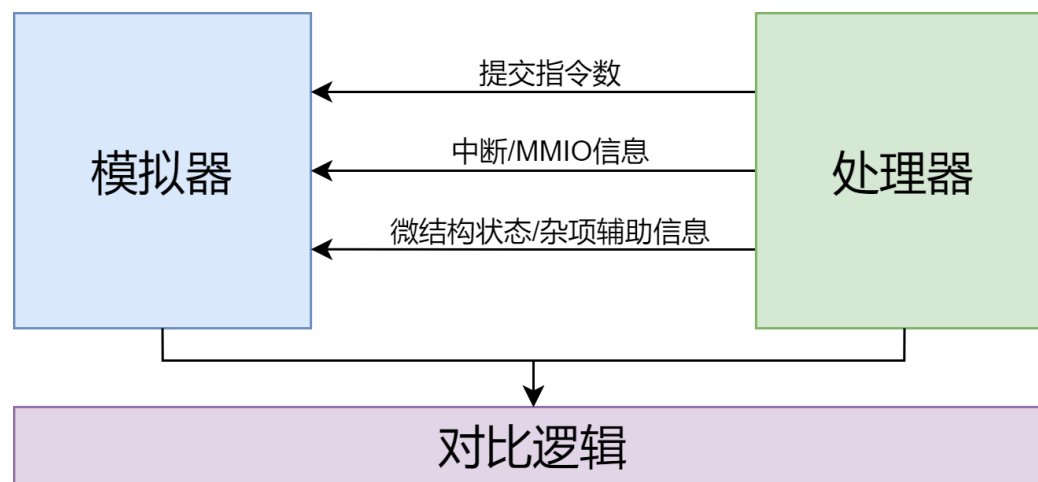
验证一次提交结果的整体流程

- 将上述的所有整合在一起, 我们得到了一步 difftest 的执行流程:
 - 仿真运行一个时钟周期
 - (判断第一条指令是否提交)
 - 判断是否达到仿真时间限制
 - 检查是否有中断或例外, 有的话进行处理
 - 检查正常指令的执行结果
- 对应代码中的 `int Difftest::step()` 函数

<https://github.com/OpenXiangShan/difftest/blob/master/src/test/csrc/difftest/difftest.cpp>

🌟 小结：通过微结构状态同步实现协同仿真

- 处理器向模拟器传递微结构状态
- 模拟器检查处理器传来的状态，调整自身状态
- 将处理器与模拟器的执行结果比对



- 实现**单核**下处理器与模拟器的协同仿真

目录

- 差分测试 (difftest) 原理简介
 - 常规指令的比对
 - 处理执行中的特殊状况
- **香山 difftest 框架介绍**
 - 框架提供的接口
 - 将处理器接入 difftest 框架的方法
 - 香山 difftest 框架的实现

香山 difftest 仿真框架

- 香山 difftest 框架包含一个写好的 verilator 仿真的顶层
- 用户只需要提供一个按要求修改好的 Verilog .v 文件
- 下面将以使用 Chisel 的设计为例, 介绍如何将一个新的设计接入这个框架

```
make[4]: Leaving directory '/home52/whq/xs-env/nexus-am/am'
make[3]: Leaving directory '/home52/whq/xs-env/nexus-am'
make[3]: Entering directory '/home52/whq/xs-env/nexus-am/libs/klib'
# Building lib-klib [riscv64-nutshell]
make[3]: Leaving directory '/home52/whq/xs-env/nexus-am/libs/klib'
# Creating binary image [riscv64-nutshell]
+ LD -> build/dummy-riscv64-nutshell.elf
+ OBJCOPY -> build/dummy-riscv64-nutshell.bin
make[3]: Entering directory '/home52/whq/xs-env/NutShell'
make[4]: Entering directory '/home52/whq/xs-env/NutShell/difftest'
Emu compiled at Jul 14 2021, 14:59:53
Using seed = 3826
The image is /home52/whq/xs-env/nexus-am/tests/cputest/build/dummy-riscv64-nutshell.bin
Using simulated 8192MB RAM
Using /home52/whq/xs-env/NEMU/build/riscv64-nemu-interpreter-so for difftest
[src/device/io/mmio.c,13,add_mmio_map] Add mmio map 'clint' at [0xa2000000, 0xa200ffff]
[src/device/io/mmio.c,13,add_mmio_map] Add mmio map 'sdhci' at [0xa3000000, 0xa300007f]
[src/device/sdcard.c,118,init_sdcard] Can not find sdcard image: /home/yzh/projectn/debian.img
The first instruction of core 0 has committed. Difftest enabled.
Core 0: HIT GOOD TRAP at pc = 0x8000002c
total guest instructions = 20
instrCnt = 20, cycleCnt = 672, IPC = 0.029762
```

使用 difftest 框架验证简单程序在 NutShell 上的运行

香山 difftest 仿真框架

- 使用 Chisel 生成 Verilog, 或完成 Verilog 文件的编写后,
- 为 difftest 提供以下的目录结构:

```
.
├─ build
│   └─ SimTop.v // 处理器 verilog 源代码
├─ difftest // difftest 仓库, 可以作为 submodule 引入
└─ .....
```

- 如果是使用 Chisel 的设计, 需要将 difftest import 进来
- Difftest 会使用系统环境变量中指定的 NEMU 作为用来对比的模拟器
 - 需要正确配置环境变量 NEMU_HOME

仿真框架的顶层

- 仿真框架的顶层默认连接了一些 IO 端口, 它们的定义在 `diffptest/src/main/scala/Diffptest.scala` 中

- `LogCtrlIO`: 控制 debug 信息的输出
- `PerfInfoIO`: 控制性能计数器
- `UARTIO`: 用于 uart 输入输出的处理

```
import diffptest._
// .....
class SimTop extends Module {
  val io = IO(new Bundle(){
    val logCtrl = new LogCtrlIO
    val perfInfo = new PerfInfoIO
    val uart = new UARTIO
    // .....
  })
// .....
}
```

- 这部分的内部的逻辑需要 RTL 代码作者自行实现, 顶层只给出控制信号
- 请注意: 这些端口必须出现在仿真顶层中, 但是 RTL 代码可以选择忽略这些信号

框架提供的接口

- 这一版本的 difftest 采用 DPI-C 来将仿真中的信号传递到 difftest 框架中
- 在仿真程序执行的过程中会**调用 DPI-C 函数**
 - 将 difftest 感兴趣的信号写入到对应的结构体中
- 需要传递的信息的最小子集包括:
 - DifftestInstrCommit
 - DifftestArchIntRegState
 - DifftestCSRState
 - DifftestArchEvent
 - DifftestTrapEvent

```
if (!env.FPGAPlatform) {  
  val difftest = Module(new DifftestArchEvent)  
  difftest.io.clock := clock  
  difftest.io.coreid := hardId.U  
  difftest.io.intrNO := RegNext(difftestIntrNO)  
  // .....  
}
```

在 Chisel 源码中向 difftest 传递信息

框架提供的接口

- 传递信号给 difftest 的核心原则:

在指令提交的时刻其产生的影响恰好生效

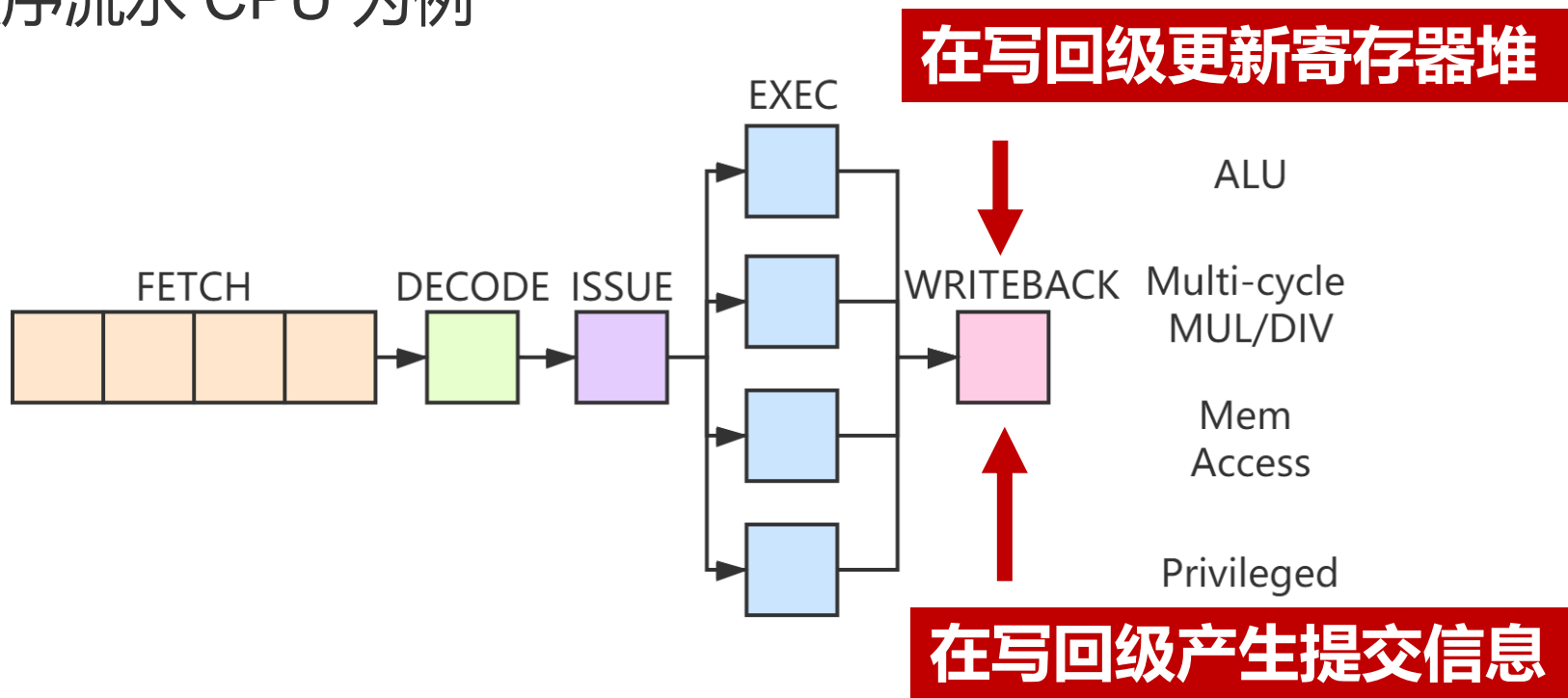
- 为了满足这一原则, 一些传递给 difftest 的信号需要被延迟一拍
- 具体如何延迟**因不同设计而异**

```
if (!env.FPGAPlatform) {  
    val difftest = Module(new DifftestArchEvent)  
    difftest.io.clock := clock  
    difftest.io.coreid := hardId.U  
    difftest.io.intrNO := RegNext(difftestIntrNO)  
    // .....  
}
```

🏔️ 框架提供的接口

原则: 在指令提交的时刻其产生的影响恰好生效

- 为什么需要这样的延迟操作?
- 以一个顺序流水 CPU 为例



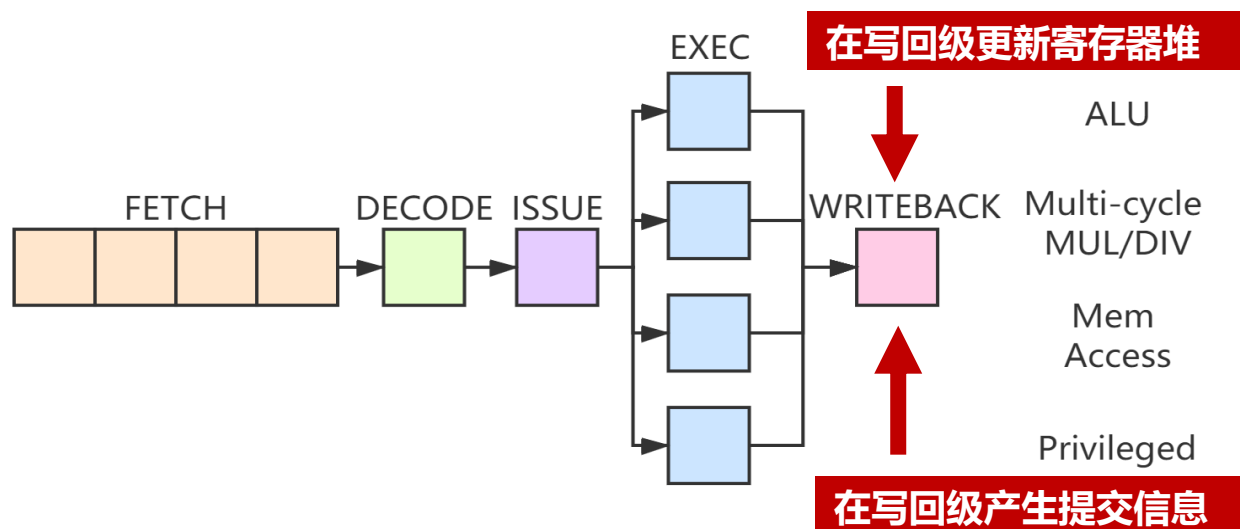
- 寄存器堆被更新后的下一个时钟周期才能从中读出新的值
- Difttest 进行比对需要知道这条指令提交产生的影响

🏔️ 框架提供的接口

原则: 在指令提交的时刻其产生的影响恰好生效

- 于是我们这样将信息传递给 difftest
 - DifftestInstrCommit <--> RegNext(指令提交)
 - DifftestArchIntRegState <--> 体系结构整数寄存器堆读结果

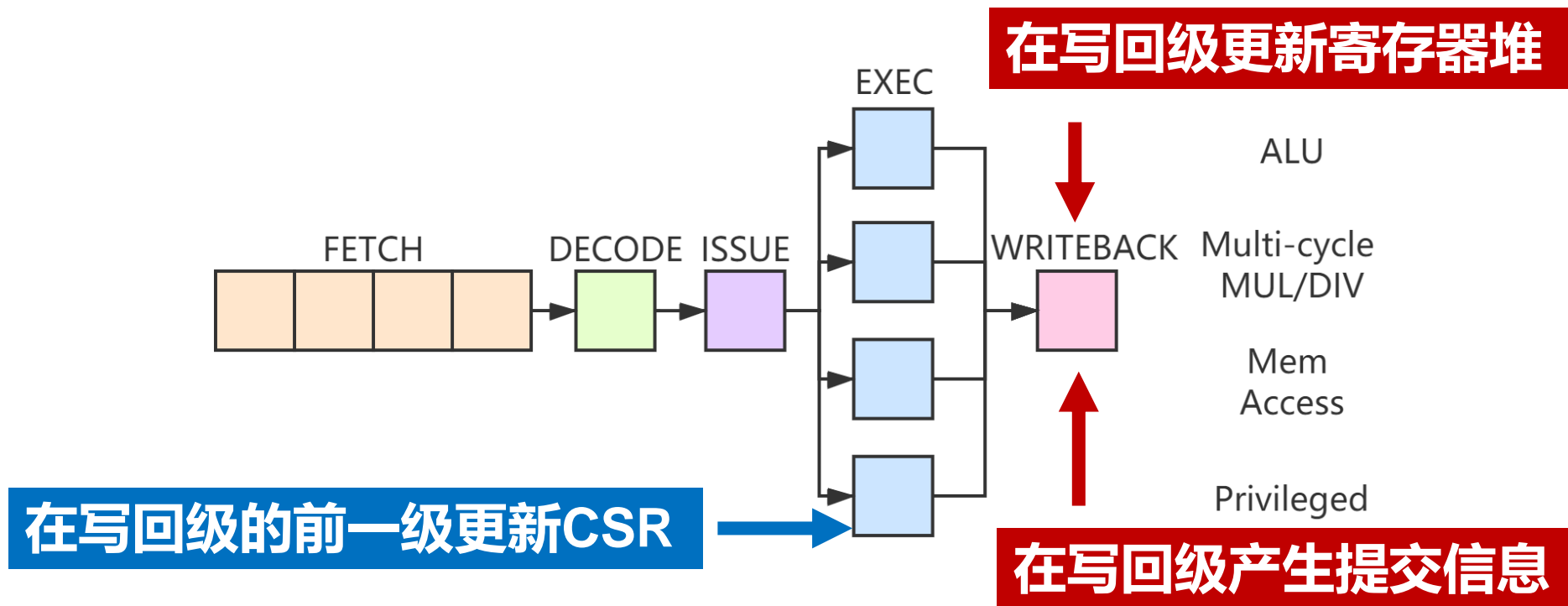
cycle	0	1
	顺序处理器指令写回	上一周周期写回信息传递到 difftest 框架
	寄存器堆更新	读取更新后的寄存器堆的值传递到 difftest 框架



🏔️ 框架提供的接口

原则: 在指令提交的时刻其产生的影响恰好生效

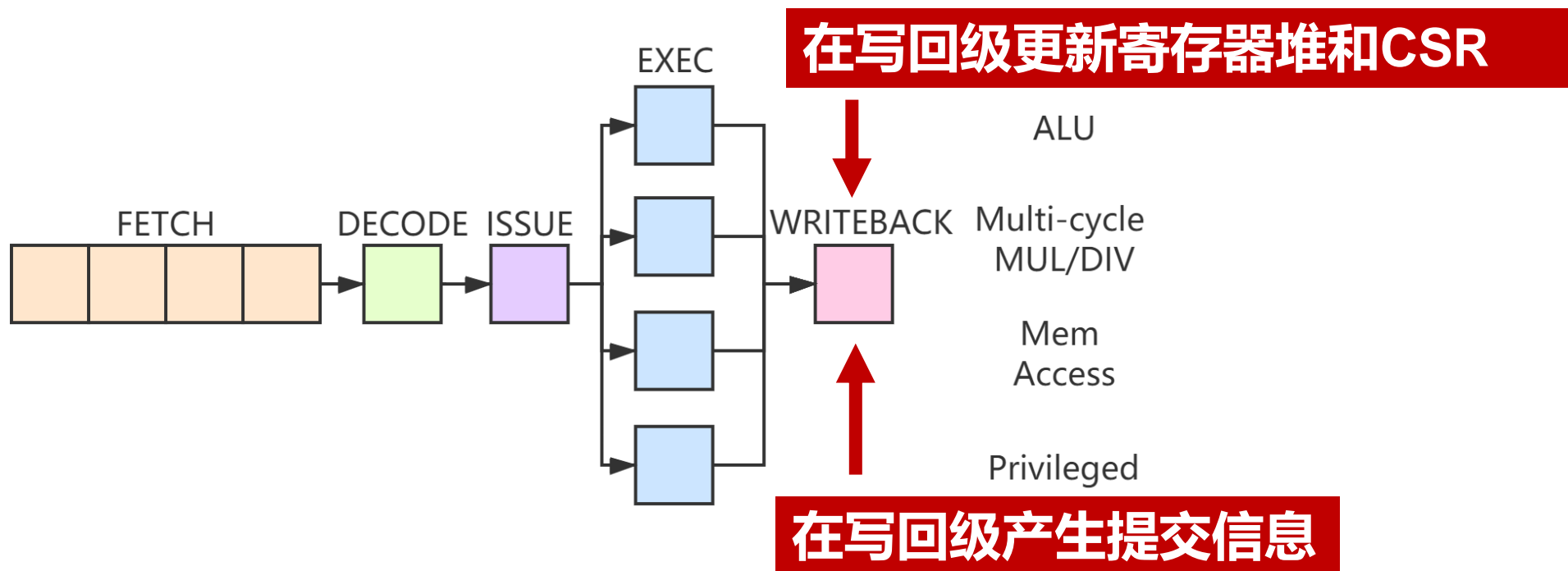
- 练习1:
 - CSR 的更新在写回级的前一级发生
 - 现在需要将 CSR 的比对也加入 difftest 中
 - 该如何设计 difftest 的接口?



🏔️ 框架提供的接口

原则: 在指令提交的时刻其产生的影响恰好生效

- 练习2:
 - CSR 的更新在写回级发生
 - 现在需要将 CSR 的比对也加入 difftest 中
 - 该如何设计 difftest 的接口?



使用 diffptest 框架开始 debug

- 在完成上述所有工作之后,我们就完成了协同仿真框架的接入

- 在 diffptest 目录下,执行 make emu 来使用 verilator 生成 C++ 仿真程序

- 仿真程序生成之后,可以执行 ./build/emu --help 来查看仿真程序的运行参数.

- 例如:

 - ./build/emu -b 0 -e 0 -i ./ready-to-run/coremark-2-iteration.bin

```
===== REF Regs =====
^[[1;34m[src/device/sdcard.c,118,init_sdcard] Can not find sdcard image: /home/yzh/projectn/debian.img^[[0m
[WARNING] diffptest store queue overflow, diffptest store commit disabled
$0: 0x0000000000000000 ra: 0x000000008000352c sp: 0x000000008000ff30 gp: 0x51ad80e12e2762e8
tp: 0x51ad80e12e2762e8 t0: 0x51ad80e12e2762e8 t1: 0x000000008000ff28 t2: 0x51ad80e12e2762e8
s0: 0x0000000000000029 s1: 0x0000000000000000 a0: 0x000000000294823 a1: 0x000000008000ff00
a2: 0x0000000000000000 a3: 0x0000000000000064 a4: 0x0000000080010000 a5: 0x000000000290000
a6: 0x0000000000000000 a7: 0x51ad80e12e2762e8 s2: 0x0000000080007448 s3: 0x0000000080007440
s4: 0x0000000000000001 s5: 0x0000000000000000 s6: 0x0000000080007280 s7: 0x51ad80e12e2762e8
s8: 0x51ad80e12e2762e8 s9: 0x0000000000000000 s10: 0x0000000080006d30 s11: 0x0000000080006520
t3: 0x51ad80e12e2762e8 t4: 0x51ad80e12e2762e8 t5: 0x51ad80e12e2762e8 t6: 0x51ad80e12e2762e8
ft0: 0xffffb210ffffb220 ft1: 0xffffb1f0ffffb200 ft2: 0xffffb0f4fffffb0f4 ft3: 0xffffb0f4fffffb0f4
ft4: 0xffffb0f4fffffb0f4 ft5: 0xffffb0f4fffffb0f4 ft6: 0xffffb0f4fffffb0f4 ft7: 0xffffb0f4fffffb0f4
fs0: 0xffffb1e0ffffb0f4 fs1: 0xffffb1d0ffffb0f4 fa0: 0xffffb0f4fffffb0f4 fa1: 0xffffb0f4fffffb0f4
fa2: 0xffffb0f4fffffb0f4 fa3: 0xffffb0f4fffffb0f4 fa4: 0xffffb0f4fffffb0f4 fa5: 0xffffb0f4fffffb0f4
fa6: 0xffffb0f4fffffb0f4 fa7: 0xffffb0f4fffffb0f4 fs2: 0xffffb0f4fffffb0f4 fs3: 0xffffb0f4fffffb0f4
fs4: 0xffffb0f4fffffb0f4 fs5: 0xffffb0f4fffffb0f4 fs6: 0xffffb0f4fffffb0f4 fs7: 0xffffb0f4fffffb0f4
fs8: 0xffffb0f4ffffb1a4 fs9: 0xffffb194ffffb08c fs10: 0xffffb284ffffb2a8 fs11: 0xffffb24cffffb268
ft8: 0xffffb1f0ffffb228 ft9: 0xffffb164ffffb1d4 ft10: 0x3e3e3e3e2b3e2b3e ft11: 0x5d2c3e2b3e5b2c
pc: 0x0000000080003540 mstatus: 0x00000000001800 mcause: 0x0000000000000000 mepc: 0x03165c7fe8fffa9a
sstatus: 0x0000000000000000 scause: 0x0000000000000000 sepc: 0x0000000000000000
satp: 0x0000000000000000
mip: 0x0000000000000000 mie: 0x0000000000000000 mscratch: 0x0000000000000000 sscratch: 0x0000000000000000
mideleg: 0x0000000000000000 medeleg: 0x0000000000000000
mtval: 0x0000000000000000 stval: 0x0000000000000000mtvec: 0x0000000000000000 stvec: 0x0000000000000000
priviledgeMode: 3
a0 different at pc = 0x008000353c, right= 0x000000000294823, wrong = 0x0000000000000000
Core 0: ^[[31mABORT at pc = 0x8000353c
^[[0m^[[35mtotal guest instructions = 2,911
^[[0m^[[35minstrCnt = 2,911, cycleCnt = 8,487, IPC = 0.342995
^[[0m^[[34mSeed=7120 Guest cycle spent: 8,488 (this will be different from cycleCnt if emu loads a snapshot)
^[[0m^[[34mHost time spent: 22ms
^[[0mmake[3]: *** [verilator.mk:151: emu-run] Error 1
make[3]: Leaving directory '/home52/whq/xs-env/NutShell/diffptest'
```

Diffptest 报错示意

香山 difftest 仿真框架: 文档导读

- Difftest 自带的文档介绍了接入的过程
 - <https://github.com/OpenXiangShan/difftest/blob/master/doc/usage.md>
- Step by step 参考
 - <https://github.com/OSCPU/NutShell/commits/3991293b3af2026231d9d071635fb7a1b8252cd7>
 - 可以从 git commit 中看到如何将 NutShell 接入这个 difftest 框架
- 香山项目, NutShell项目都有 difftest 支持, 可以作为参考
 - <https://github.com/OpenXiangShan/XiangShan>
 - <https://github.com/OSCPU/NutShell>
- **RTFSC (Read The Friendly Source Code)** is important

Take home message

- 了解 difftest 框架的原理, 以及**如何自行实现**一个 difftest 框架
- 学会如何将设计接入到现有的 difftest 框架中
- 学会使用 difftest 框架排查问题

谢谢!
请批评指正
