

简易 Cache 测试框架

Caterpillar

2025 年 03 月 22 日

目录

1. 引入

2. 理论分析

3. 实现功能

4. 主要任务

5. 构建系统、目录结构

6. Bug

目录

1. 引入

2. 理论分析

3. 实现功能

4. 主要任务

5. 构建系统、目录结构

6. Bug

1.1 场景

你的想法

- 正在做 YSYX，完成了 ICache 和 DCache，想要进行随机测试
- 不想实现 Cache 的周期精确模拟器
- 不想用程序实现 AXI 的模拟
- 想进行随机测试
- 想统计命中率
- 想用不一样的方法完成讲义上 CacheSim

我的想法

- 2024 年龙芯杯
- 分工：我负责编写测试框架，访存和 SoC 部分
- 在流水线还没有完成时，如何单独测试 Cache？
- 流水线和访存解耦，访存和 SoC 部分可以单独测试

1.2 失败的尝试 1

一个简单的想法：带着 SoC 部分仿真，用代码模拟 CPU 侧的流水线

- 在流水线模拟器中设置一个数组
- 用这个数组作为 REF，模拟内存
- 流水线模拟器同时向 Cache 和 REF 提交 Cache 事务

Cache 事务

考虑 Cache 可以接收的请求，有

- 读写命令 `Load(addr, size)`, `Store(addr, size, data)`
- 缓存一致性维护操作 `Flush(...)` (ISA 相关)

很快，在测试 `fence.i` 时，你就发现这个方法缺陷：

- 现在 ICache 和 DCache 正在访问同一个 Cache line 对应的地址空间
- 流水线模拟器无法得知 DCache 何时将 Cache line 写入内存
- 此时 ICache 执行 IF，返回新、旧数据，都是正确的（没有执行 `fence.i` 时）

启示

不同 Cache 是不同的 AXI Master，在没有 `fence` 指令同步数据的情况下，它们看到的地址空间中的数据可能不一致

1.3 需求分析

BASIC

- 随机测试正确性
- Trace playback 统计命中率
- KISS
 - 「不」实现周期精确模拟器
 - 「不」用代码模拟 AXI 行为
 - 「不」使用  UVM
- 能支持所有 Cache 事务的测试
 - 基础读写
 - 缓存一致性维护
 - PMA Uncached 的情况

- 系统结构

- 单核无 DMA，顺序访存（限制和简化流水线模拟器的实现）
- 支持任意的缓存拓扑，可以 I/D 分离，可以多级缓存
- 支持任意 I/D Cache 间维护一致性的方式

BONUS

- 如何在「有无 SoC」，「有无仲裁器」等情况间复用代码？

目录

1. 引入

2. 理论分析

3. 实现功能

4. 主要任务

5. 构建系统、目录结构

6. Bug

2.1 相关知识

通常,实现缓存一致性 (Cache Coherence) 和内存一致性 (Memory Consistency) 需要软硬件间协作, ISA 中定义了相关指令和内存模型。YSYX (单核顺序访存无 DMA) 中不用过多考虑这些。

☞ 缓存一致性协议

- ☞ 总线嗅探: MESI, MOESI, ...
- 基于目录
- 具体协议: AMBA ACE, TileLink, ...

栅障指令 FENCE.I, FENCE

NEMU: NOP; NPC: 讲义上给出了思路;

RV A: ☞ LR/SC, 原子指令 ☞ AMO**

简单实现思路

- LR/SC: 单核无 DMA, Load/Store
- AMO**: NPC IDU 产生 μ OP, 关中断

内存模型: TSO, RVWMO (☞ Tutorial)

规定一个 Load 操作的结果可能是什么

2.2 内存模型

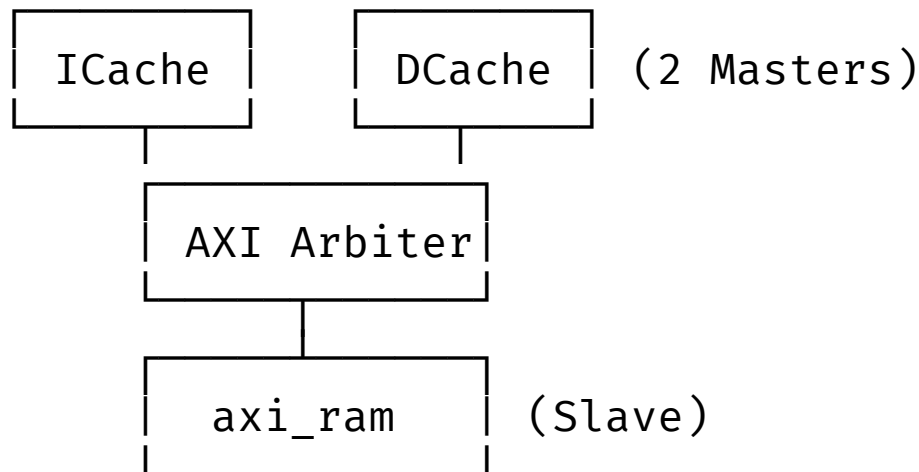
想要建模带有 ICache 和 DCache 的系统，我们可以借鉴内存一致性模型中的规则

内存一致性模型 (Tutorial)

A consistency model specifies a contract between the programmer and a system, wherein the system guarantees that if the programmer **follows the rules for operations on memory**, memory will be consistent and **the results** of reading, writing, or updating memory will be **predictable**.

- Program Order
→ 流水线模拟器提交 Cache 事务顺序
- Global Memory Order
→ Mem 观察到 ICache 和 DCache 的请求经过 AXI Arbiter 后实际提交的顺序
- Presrved Program Order
→ 在流水线模拟器侧从 Load 事务的结果中能够观测（推测）到的实际提交的顺序

2.3 建模



代码 1 拓扑

生产者: DCache Store

消费者: ICache IFetch, DCache Load

顺序访存: Cache 依次向内存请求 CPU 提交的事务 (可能不准确)

希望构建一个模型,

输入: CPU 流水线模拟器提交 Cache 事务的 Trace (事务 $\{m_i\}$, 周期 $\{t_i\}$)

输出: 对于每个 IFetch f_i 和 Load l_i 事务, 可能的结果的集合 R_{f_i}, R_{l_i} ;

- $f_i \in \{m_i\} \wedge l_i \in \{m_i\}$
- 所有 Cache 事务的下标都按照时间顺序排序, 同一周期提交的事务

2.4 简易宽松内存模型

针对 代码 1 所示的系统, 我提出一个简单易实现的“内存一致性模型”用于 Cache 测试

定义 2.4.1: (简易内存模型) 对某一内存地址的本地 Load, 结果可以是先前最后一次对同一地址本地 Load 的值, 或者在这次本地 Load 之后任意一次全局 Store 的值;

注: 请注意对于本地 Load 与 全局 Store 间关系的表述和常规内存模型间的不同: 这实际上规定了一致性结点是内存, 而且系统的结构类似  SMP。

2.4 简易宽松内存模型

单调性(常规 F/L/S)

对于一个地址 x ，一旦某次 IFetch f_i 或 Load l_i 得出 Store s_j 写入的值，则后续所有 Load $l_{i'} (i' > i)$ 得出的值一定在 $\{s_k \mid k \geq j\}$ 写入的值的集合 $R_{s_{\geq k}}$ 中；

证明：在单核 CPU 无 DMA 的系统中，地址空间的副本可能且只可能在两个位置：内存或 Cache 行。对于 x ，

- 在 f_i 后，（考虑无 `fence.i` 时的 DCache Store + ICache Ifetch）
 - 由于 DCache 向总线提交 Store 一定是顺序的，因此内存中的值一定 $\in R_{s_{\geq k}}$ ；
 - 进一步，ICache 从总线（如果需要）得到的值，一定 $\in R_{s_{\geq k}}$ ；
 - 若 ICache 从 Cache Line 中返回 f_{i+1} 的值，则是 s_j 写入的值；
 - 若 ICache 从总线返回 f_{i+1} 的值，一定 $\in R_{s_{\geq k}}$ ；
 - 在 l_i 后，（考虑 DCache Store + DCache Load）对于同一个 Master，这个顺序应该保持；
-

2.4 简易宽松内存模型

单调性 (常规 F/L/S)

对于一个地址 x ，一旦某次 IFetch f_i 或 Load l_i 得出 Store s_j 写入的值，则后续所有 Load $l_{i'} (i' > i)$ 得出的值一定在 $\{s_k \mid k \geq j\}$ 写入的值的集合 $R_{s_{\geq k}}$ 中；

可以使用队列保存上述最近 Store 的值的集合 $R_{s_{\geq k}}$ 。

2.5 简易宽松内存模型模拟器

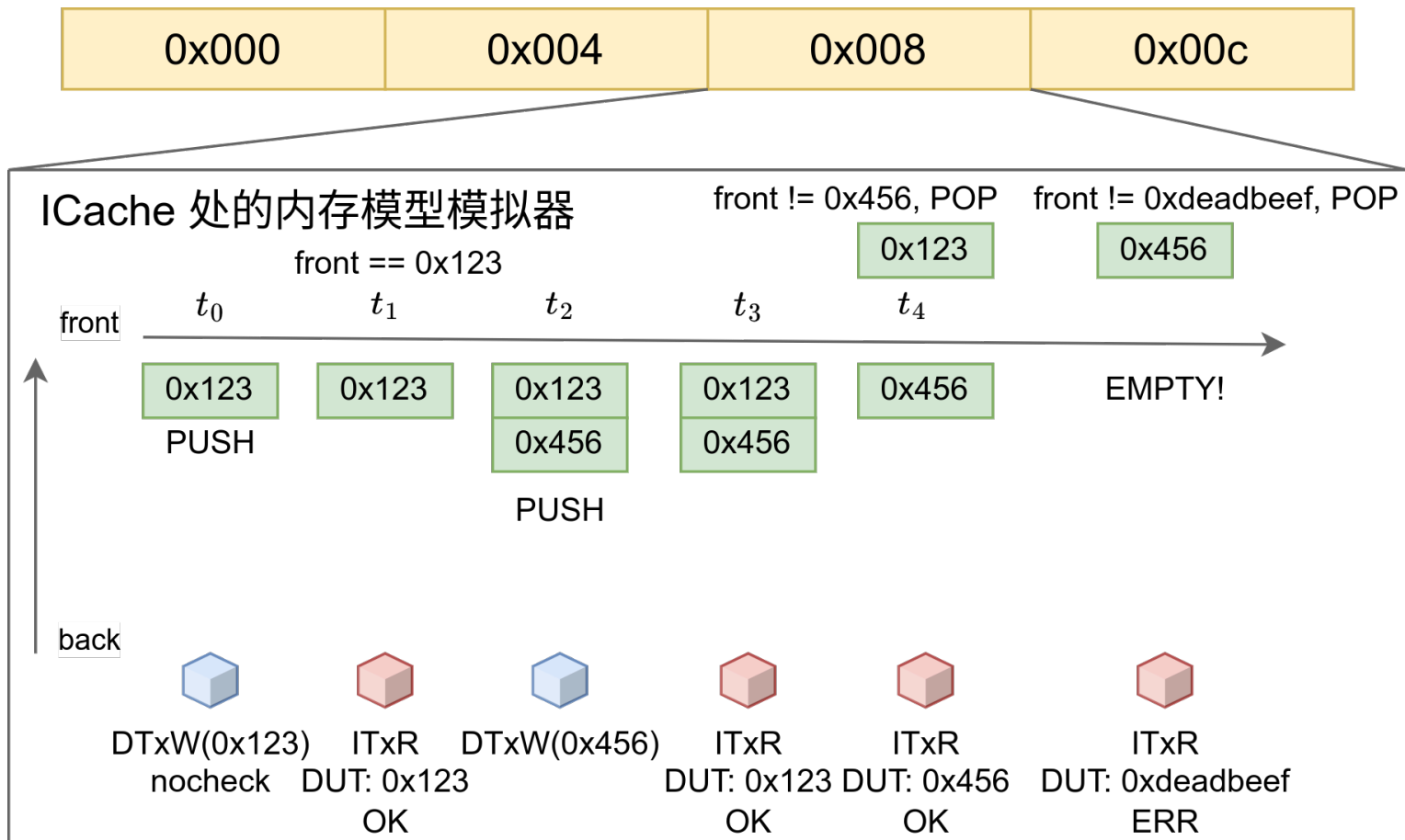


图 1 简易宽松内存模型模拟器

2.6 Under-approximation

使用简易内存一致性模型验证 Cache，自有其代价：不能实现 Sound 的正确性判定，只能通过增加随机测试的样本数弥补这一缺陷；

目录

1. 引入

2. 理论分析

3. 实现功能

4. 主要任务

5. 构建系统、目录结构

6. Bug

3. 实现功能

在 [OpenJiuXiang](#) 测试框架 中，实现了以下功能：

- **框架和测试分离**: 可使用多种框架测试多个 DUT
 - 框架的 Makefile 支持测试其他模块
- **声明式用户测试**: 生成一个 `std::vector<Tx *>`
其中 `class Tx` 是所有 Cache 事务的基类，在此基础上可以实现
 - **随机测试**: 写一个 `class Tx` 随机生成器
 - **Trace playback**: 写一个加载 NEMU mtrace 的 `class Tx` 生成器
- **命中率统计**: 可以使用 DPI-C，或者将 `hit` 线暴露于 `Cache` 接口
- **简单条件判断**: 使用 `std::function` 回调函数，框架在运行过程中调用

目录

1. 引入

2. 理论分析

3. 实现功能

4. 主要任务

5. 构建系统、目录结构

6. Bug

4. 主要任务

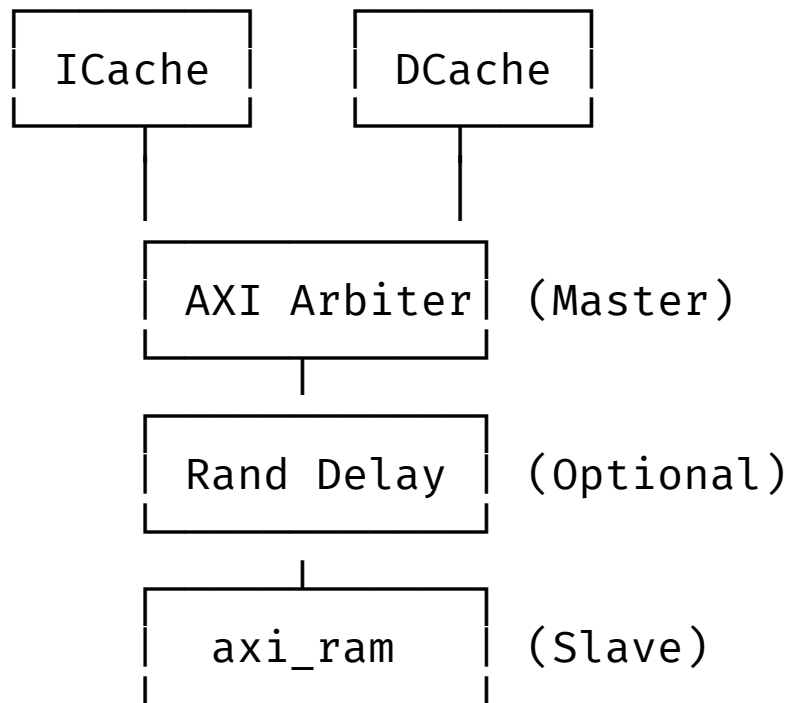
CPU 侧

- 模拟 Ifu 和 Lsu
 - 对 Cache 施加事务 `class Tx`
 - 统计命中率
 - 执行条件判断
- LA32R 中, Cache 事务有读写和 CACOP
 - 相应地, 有 `[ID]CacheTx[RW]H?` 等 `class Tx` 的子类 (H 表示应该命中)
 - `[ID]CacheTxCacopC(INV|IDX|LOOKUP)` 等 LA32R ISA 特定的 Cache 事务

总线侧

- 模拟 AXI: 有开源 AXI RAM 实现, 可以使用 [alexforencich/verilog-axi](https://github.com/alexforencich/verilog-axi)
 - 维护内存模型: 保存某地址可能读出的值
- Store: DCache
 - Load: ICache, DCache
 - 处理任意延迟: 牺牲 Soundness, 每个地址一个 `std::queue`, 保存历史上所有可能的值 (见下文);

4.1 拓扑



4.2 流程

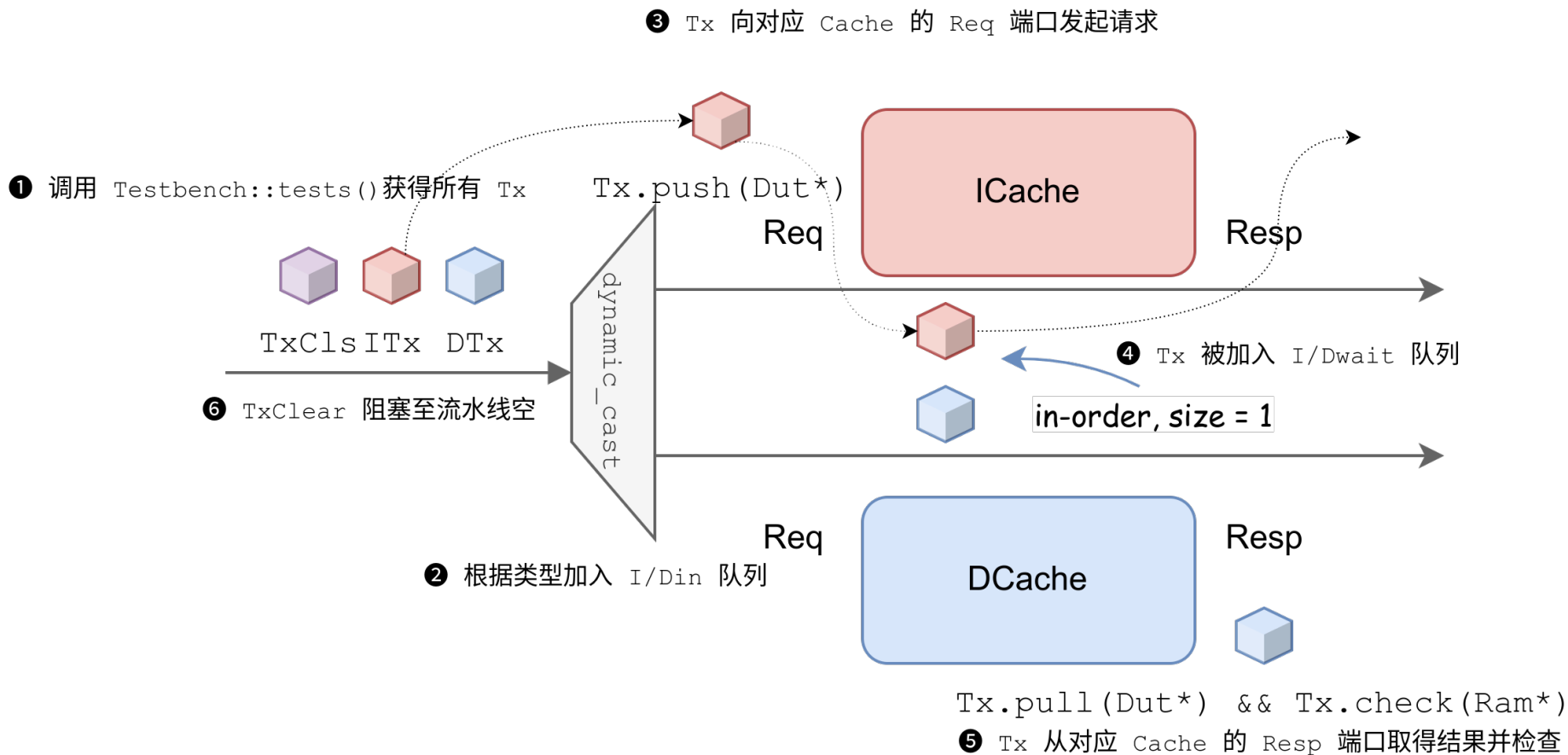


图 2 Cache 事务 Tx 的数据通路

目录

1. 引入

2. 理论分析

3. 实现功能

4. 主要任务

5. 构建系统、目录结构

6. Bug

5. 构建系统、目录结构

逻辑

由于需要对各个部件分别写单元测试；不同类型的部件的单元测试形态差别大。所以需要写出多个种类的测试框架，让每个单元测试单独选择顶层模块和测试框架。


- 在 `tests` 中，每一个目录对应一个单元测试；
- 在 `framework` 中，每一个目录对应一种测试框架，单元测试可以选择不同的测试框架；
- 在 `model` 中，`Makefile` 能够根据单元测试选择的顶层模块生成 Verilator 模型（静态链接库）；

三个部分通过静态库的形式链接起来生成可执行文件；

目录结构

```
$ tree
├── framework          # Frameworks
│   ├── nvboard        # testing using NVBoard
│   ├── cache-axi-v3    # Uncached support
│   ├── cache-axi-v4    # CACOP support
│   └── simple-timing   # judge based on time
├── model              # Design of CPU Core
│   ├── Makefile        # generate verilated lib
│   └── vsrc
├── tests              # Unit tests
│   ├── regfile-spec
│   │   ├── regfile-spec.gtkw
│   │   ├── tb.cpp
│   │   └── Makefile    # specify `TOP`,
│   │                   `FW`, ... here
│   ├── cache-spec
│   │   ├── cache-dummy-v3
│   │   ├── dcache-v4
│   │   ├── dcache-v4-lru
│   │   ├── dcache-v5-cacop
│   │   ├── hitrate
│   │   └── icache-v5-cacop
│   └── tests.mk        # rules for unit tests
```

5.1 一键回归测试



A terminal window with a dark background. The title bar shows the user 'pc' at 'pc' in the directory '~/Digital/NSCSCC/Testing/nscsc-team-la32r/Core'. The prompt is '~/.Dig/NS/T/nscsc-team-la32r/Core'. The command 'cache x3 !4 ?2' is entered, followed by 'sudo apt install peek'. A large white number '1' is displayed in the center of the terminal.

Talk is cheap. Show me the code.

— Linus Torvalds

目录

1. 引入

2. 理论分析

3. 实现功能

4. 主要任务

5. 构建系统、目录结构

6. Bug

6. Bug

理想是美好的，现实是残酷的，实现时，更新 REF 和 DUT 不同步（对 REF 的更新在执行 `Tx::check(Ram ★)` 时执行，而每周期 ICache 和 DCache 是独立更新的），导致有极小概率触发 False Positive, 考虑如下情况：

第 t_1 个周期时，地址 x 的值集合为 \emptyset , DCache 开始执行 s_1 (Uncached)

第 t_2 个周期时，DCache 的请求已经被仲裁器接收并写入了内存，DCache 开始等待 B 通道上的回应；与此同时，ICache 对 x 的读请求在 DCache 后一个周期被内存处理，读出了 s_1 的值；

第 t_3 个周期，ICache 比 DCache 先返回结果，然而此时因为 DCache 还没有得出结果，内存的 REF 还没有更新，模拟器发现 x 的值集为 \emptyset 报错！

暂时没有 Workround，需要人工排除。

Q & A