WayLookup 验证报告

章耀康 Oct, 2025

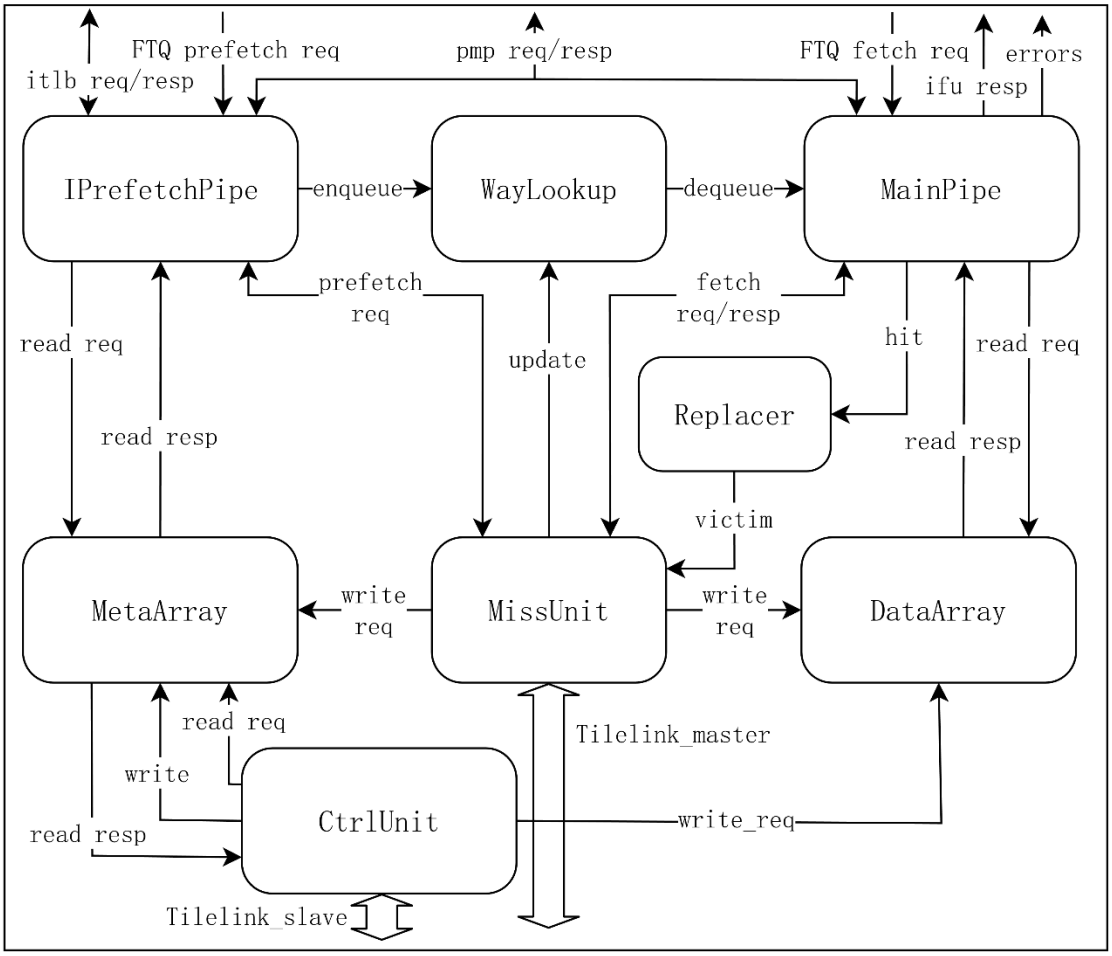
版本 V0.1

目录

1、验证对象

1.1 WayLookup 简介

在现代处理器的取指流程中，指令缓存（ICache）承担着向前端流水线提供高带宽指令数据的任务。为了提升 ICache 的访问效率，通常在其前端会设置若干预取（Prefetch）与查表（Lookup）管线，以实现指令地址的预测与缓存命中判断。



WayLookup 模块是 Icache 前端的一个 FIFO 环形队列结构，位于 IPrefetchPipe 与 MainPipe 之间，用于暂存来自 IPrefetchPipe 的查询结果（包括 MetaArray 和 ITLB 的元数据），以供 MainPipe 使用。同时，WayLookup 监听 MSHR 向 SRAM 写入 cacheline 的命中信息，对命中信息进行更新。

1.1.1 内部结构与工作机制

WayLookup 内部由一个环形 FIFO 队列构成，通过读指针（readPtr） 和写指针（writePtr）管理读写位置。当发生 read.fire 且队列不为空时，进行读操作，读指针自增；当发生 write.file 且队列不为满时，且 gpf_valid 不为真，则进行写操作，

写指针自增。读写指针越过队列上限时，回绕至队列头部。

WayLookup 支持 bypass（旁路）机制：当队列为空但有写请求时，写入数据可直接旁路到读端口，而不进入队列。这在流水结构上避免了额外延迟，有利于维持访问路径的关键时序。

WayLookup 还包含一条储存 GPF（Guest Page Fault）信息的辅助通路，通过 gpf_entry 寄存器与 gpfPtr 指针维护，以在不增加主队列资源占用的情况下，记录并输出异常信息。

在有 MSHR 写入 SRAM 的情况下，WayLookup 会暂停出队，以防止与 DataArray 访问路径竞争。由于此时 MainPipe 的 S0 流水级也被阻塞，因此不会影响整体性能。

1.1.2 刷新机制

当接收到 flush 信号时，WayLookup 会执行刷新机制。读写指针（readPtr.value, readPtr.flag, writePtr.value 与 writePtr.value）会被清零，并清空 gpf 信息，gpf_entry.valid 与 gpf_entry.bits 置零。

1.1.3 读写操作

（1）读操作

读操作根据读指针 readPtr 从环形队列中取出数据：

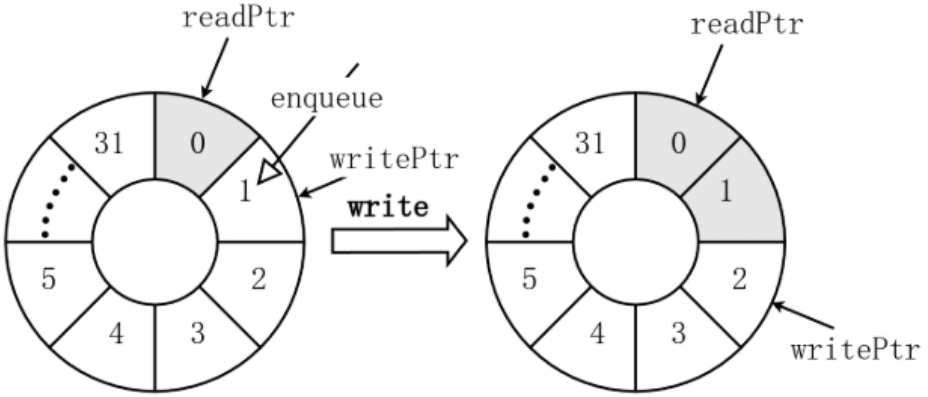
- 正常读：当队列非空且 io.read.valid 为高时，从 entries(readPtr.value)中读出信息；
- Bypass 读：当队列为空且写有效（empty && io.write.valid）时，直接旁路写入数据到读端口
- GPF 命中读：若 readPtr == gpfPtr 且 gpf_entry.valid 为高，则一并输出 GPF 信息。读出后（io.read.fire 为高）清除 gpf_entry.valid 标志位
- GPF 未命中读：正常输出访存信息，GPF 信息清零

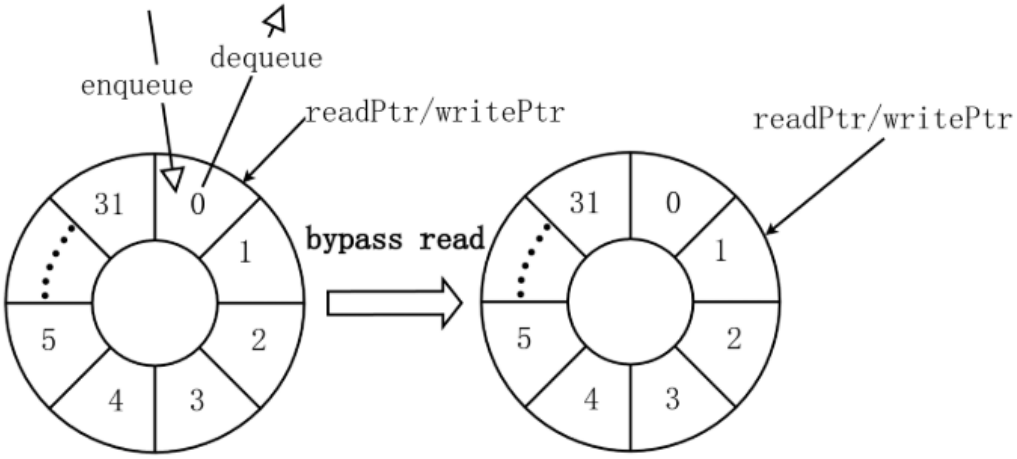
（2）写操作

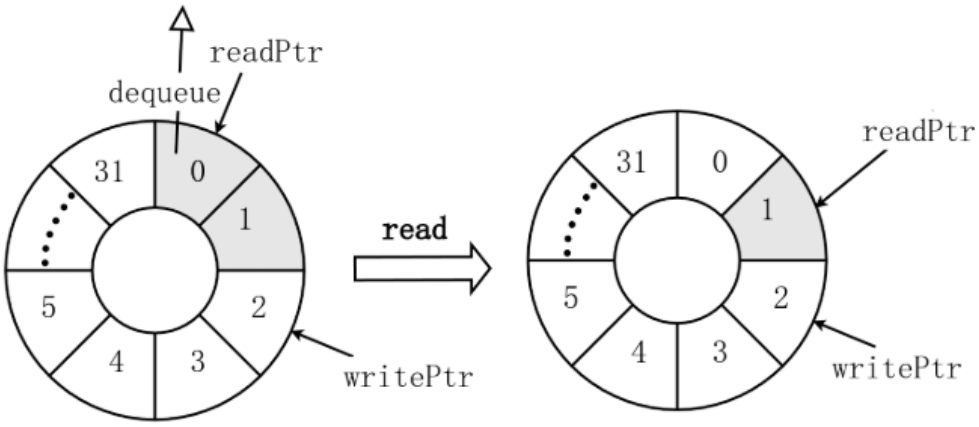
写操作根据写指针 writePtr 写入 IPrefetchPipe 提供的数据：

- GPF 停止：若已有有效 GPF 信息待读出且未被命中（gpf_entry.valid && !(io.read.fire && gpf_hit)），则暂停写入
- 写就绪无效：当队列为满或发生 GPF 停止时，io.write.ready 为低
- 正常写：当 io.write.valid 为高且队列未满且无 GPF 阻塞时，将写入数据存入环形队列

- 写入带 ITLB 异常：若写入信息包含 ITLB 异常，则更新 gpf_entry 与 gpfPtr



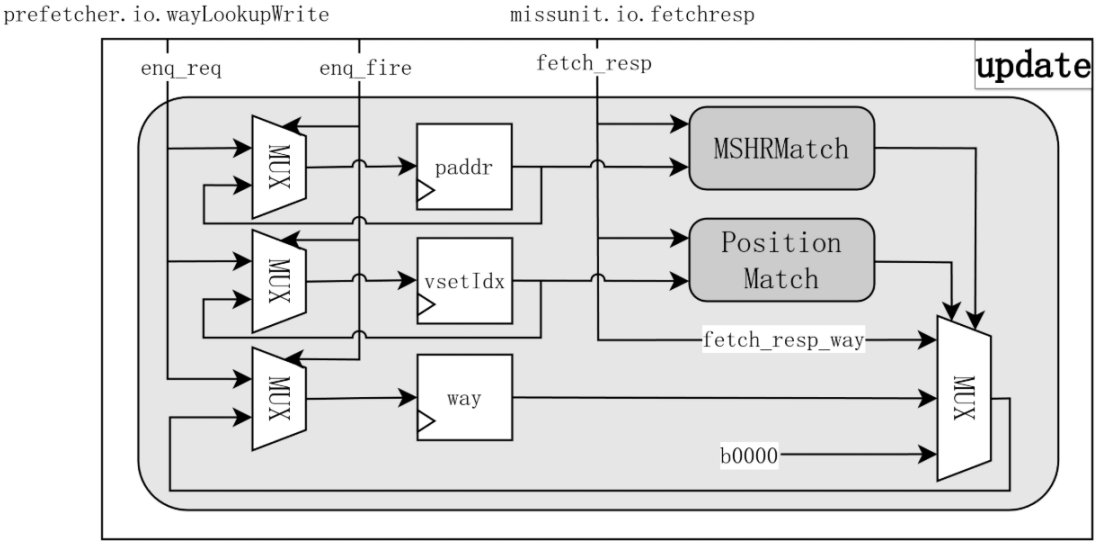




1.1.4 更新机制

WayLookup 会同时监听来自 MissUnit 的更新请求。MissUnit 处理完 Cache miss 后，会向 WayLookup 写入命中信息：

- 命中更新：若 vSetIdx 与 ptag 均匹配（vset_same && ptag_same），则更新对应项的 waymask 与 meta_codes
- 未命中更新：若仅 vset_same && way_same 为真，则清空 waymask
- 不更新：其他情况下不进行更新



WayLookup 更新结构

2、验证方案与验证框架

2.1 验证方案

WayLookup 模块的验证基于 Picker 和 toffee 工具，使用 python 语言进行。

Picker 工具支持将模块的 RTL 代码转换成 python 等语言的 DUT 对象。通过 Picker 工具的桥接，可以在保留硬件特征的前提下，使用 python 中的 pytest、toffee 等软件工具便捷快速地进行验证工作。

Toffee 是使用 Python 语言编写的一套硬件验证框架。其吸收了部分 UVM 验证方法学，以保证验证环境的规范性和可复用性，并重新设计了整套验证环境的搭建方式，使开发者可以轻易地上手硬件验证工作。

综合以上，我们得到以下的验证方案：

1) 将 UnityChipForXiangShan 编译为 RTL 代码
2) 使用 picker 工具 make dut，将 WayLookup 部分的 RTL 代码转换成 python 形式
3) 以软件的形式驱动其顶层接口，观察特定接口行为和结果反馈。

2.2 验证框架

结合 UVM 思想，验证环境由几个核心组件组成：Bundle、Agent、Env、Model。这些模块共同构成一个高层次、可复用、可扩展的验证体系。

2.2.1 Bundle

Toffee 中的 Bundle 是对 DUT 端口信号的封装。其将 dut 中的相关信号组织起来，并将 dut 的 io 接口与内部信号映射到对应的 python 对象上。bundle 使上层对 DUT 接口的访问变得更加清晰、方便。并构建出了 Agent 与 DUT 之间交互的中间层，保证 Agent 与 DUT 之间的解耦。

2.2.2 Agent

Agent 是 UVM 中主要的验证组件，一般包含了 Driver、Monitor、Sequencer 多个组件。Agent 是对信号的高层封装，使得上层驱动代码可以在不关心具体信号赋值的情况下，完成对 Bundle 中信号的驱动及监测。在 Toffee 中，一个 Agent 主要由驱动方法(driver_method)和监测方(monitor_method)组成，其中驱动方法用于主动驱动 Bundle 中的信号，而监测方法用于被动监测 Bundle 中的信号。

2.2.3 Env

Env 是用来组织和管理多个 Agent、Model 等组件的顶层容器。Env 在 toffee 验证环境中用于打包整个验证环境。Env 中直接实例化了验证环境中需要用的所有 agent，并负责将这些 Agent 需要的 bundle 传递给它们。此外，可以直接将参考模型附加到 Env 上，由 Env 来完成参考模型的自动同步。

2.2.4 Model

Model（参考模型）用于模拟 DUT 的理想行为，以便在验证过程中对设计进行验证。验证环境通过比较 DUT 的输出与 Model 的预测结果，来判断 DUT 是否正确。在 toffee 验证环境中，参考模型需遵循 Env 的接口规范，以便能够附加到 Env 上，由 Env 来完成参考模型的自动同步。

2.3 覆盖率统计

在验证过程中，覆盖率（Coverage）是衡量验证完整性的重要指标。

Toffee_test 提供了 toffee_request 类，用于自动收集验证的行覆盖率（Line Coverage）和功能覆盖率（Functional Coverage）。

行覆盖率是代码覆盖率（Code Coverage）的一种，表示在仿真运行过程中，设计 RTL 代码的每一行是否被执行到。其能反映激励信号是否触发了设计的所有逻辑路径，并指导测试用例的补充与改进。Toffee_test 会自动在验证报告中生成验证的行覆盖率

情况。

功能覆盖率用于评估验证环境是否覆盖了设计规格要求的所有功能点，通过 covergroup、coverpoint 等机制定义和采样，体现测试激励的完整性。在 toffee 中，可以注册并统计功能覆盖率情况，并在时钟上升沿自动采样，或借助协程在特定情况下进行采样。Toffee_test 生成的验证报告中，会体现验证过程中的功能覆盖率情况。

3、测试环境

3.1 硬件环境

13th Gen Intel® Core™ i7-137000

3.2 软件环境

操作系统：Ubuntu 22.04 LTS

测试工具集：

名称	版本
python	3.11.13
pytest	8.4.2
picker	version: 0.9.0-master-b3b9601-2025-09-25
toffee	1.0.1
toffee_test	0.3.2.dev5+gbb7412569

4、功能点和测试点

WayLookup 的功能包含环形队列的读写、GPF 信息处理，以及 update 更新。根据每个功能点的行为，继续将功能点拆分为测试点，如下表所示：

功能点	测试点	简述
Flush	1	Flush 操作
读写指针	2.1.1 2.1.2 2.2.1	读指针自增 读指针回绕 写指针自增

	2.2.2	写指针回绕
读操作	3.1.1	正常读
	3.2.1	队列为空时无法读
	3.2.2	队列为空时 Bypass 读
	3.3.1	读到 gpf 信息
写操作	4.1.1	正常写
	4.2.1	队列已满时无法写
	4.2.2	有 gpf_valid 时无法写
	4.3.1	正常写 gpf 数据
	4.3.2	bypass 读写 gpf 数据
更新操作	5.1.1	命中更新
	5.1.2	未命中更新
	5.1.3	不更新

4.1 功能点 1 Flush

WayLookup 具有基本的 Flush 功能，当 flush 引脚拉高时，应将读写指针归零，并清空存储的 gpf 数据。

测试点：1 Flush 操作
测试用例：所有测试用例、随机测试

4.2 功能点 2 读写指针

读写信号握手完毕之后，对应指针加一。 因为是在环形队列上，所以超过队列大小后，指针会回绕到队列头部。 队列已满时无法写

有 gpf_valid 时无法写

正常写 gpf 数据

bypass 读写 gpf 数据

测试点：2.1.1 读指针自增 2.1.2 读指针回绕 2.2.1 写指针自增 2.2.2 写指针回绕 测试用例：test_ptr_roll、其它读写测试、随机测试

4.3 功能点 3 读操作

通过读操作，可以从 Entries 队列中读取读指针指向的数据

测试点：3.1.1 正常读 测试用例：test_read_write_normal、随机测试

当队列为空时，无法直接读取

测试点：3.2.1 队列为空时无法读 测试用例：test_read_empty、随机测试

但当队列为空且写有效时，可以将写数据旁读到读通道，发生 Bypass 读取

测试点：3.2.2 队列为空时 Bypass 读 测试用例：test_read_bypass、随机测试

如果发生了 gpf_hit，需要同时输出 gpf 数据

测试点：3.3.1 读到 gpf 信息 测试用例：test_read_write_gpf_not_bypass、随机测试

4.4 功能点 4 写操作

通过写操作，可以将数据写入 Entries 队列中写指针指向的位置

测试点：4.1.1 正常写 测试用例：test_read_write_normal、随机测试

当队列已满时，无法再继续写入数据

测试点：4.2.1 队列已满时无法写 测试用例：test_write_full、随机测试

如果有仍为被读取的 gpf 数据，应暂停继续写入

测试点：4.2.2 有 gpf_valid 时无法写 测试用例：test_write_gpf_valid

此外，如果写入的数据包含 itlb 异常，相关 gpf 数据应被储存在 gpf_entry 中

测试点：4.3.1 正常写 gpf 数据 测试用例：test_read_write_gpf_not_bypass

特殊的，如果写入 gpf 数据时发生 bypass，相关数据不保存在 gpf_entry，而是立即返回给读通道

测试点：4.3.2bypass 读写 gpf 数据 测试用例：test_write_gpf_bypass

4.5 功能点 5 更新操作

MissUnit 处理完 Cache miss 后，向 WayLookup 写入命中信息。MissUnit 返回的更新信息和 WayLookup 的信息相同时，vset_same 和 ptag_same 为真，更新 waymask 和 meta_codes。

测试点 5.1.1 命中更新 测试用例：test_update_hit、随机测试

如果未命中，vset_same 和 way_same 为真，waymask 清零。

测试点 5.1.2 未命中更新 测试用例：test_update_miss、随机测试

对于其它情况，不做任何操作

测试点 5.1.3 不更新 测试用例：test_update_no、随机测试

5、测试用例

为覆盖 WayLookup 模块的所有功能点及其对应测试点，设计如下测试用例。由于 Flush 操作及随机行为在多种测试中都会被触发，因此所有测试用例均覆盖测试点 1。下图展示了功能点、测试点与测试用例的对应关系。

5.1用例 test_ptr_roll

这个测试用例主要用来检查读写指针的自增和回绕情况

目标功能点：2.1.1, 2.1.2, 2.2.1, 2.2.2 步骤：①初始化及 flush ②写操作 ③读操作 ④重复步骤②③35 次 预期结果：读写指针在相应操作时自增，value 达到 31 时，下一次指针更新 value 回绕至 0 且 flag 翻转。

5.2测试用例 test_read_write_normal

这个测试用例主要用于测试正常读写 Entries 队列的回环逻辑

目标功能点：3.1.1，4.1.1

步骤：①初始化及 flush ②写入数据 ③读取数据

预期结果：

3.1.1 预设数据成功写入队列

4.1.1 成功从队列读出数据，数据内容与预设数据一致

5.3测试用例 test_read_empty

这个测试用例用于测试在队列为空时尝试读取。此时读信号无效，读取不会成功。

目标功能点：3.2.1

步骤：①初始化及 flush ②尝试读取

预期结果：3.2.1 无法读取数据

5.4测试用例 test_read_bypass

这个测试用例用于测试队列为空时的 bypass 机制。

目标功能点：3.2.2

步骤：①初始化及 flush ②同时写入和读取

预期结果：3.2.2 读取通道立即返回写入通道的数据

5.5测试用例 test_write_full

这个测试用例用于测试队列已满的情况下出现写有效。此时无 write_ready，写操作无法完成

目标功能点：4.2.1

步骤：①初始化及 flush ②连续写入 32 次，将队列填满 ③再次尝试写入

预期结果：4.2.1 无法写入数据

5.6测试用例 test_write_gpf_valid

此用例用于测试在队列中存在未被读取的 gpf 信息时进行的写操作。此时 gpf_valid 为真，write_ready 不为真，写操作无法完成

目标功能点：4.2.2

步骤：①初始化及 flush ②写入一次带 itlb 异常的数据 ③尝试再次写入

预期结果：4.2.2 无法写入数据

5.7测试用例 test_read_write_gpf_not_bypass

这个测试用例主要用于 GPF 数据的正常写入和读取。如果写入数据包含 itlb 异常，则设置 gpfPtr 并储存 gpf 数据。直到 readPtr==gpfPtr，发生 gpf_hit 的情况，读操作返回储存的 gpf 数据

目标功能点：3.3.1, 4.3.1

步骤：①初始化及 flush ②带有 itlb 异常的写入 ③读取含有 gpf 信息的数据

预期结果：

3.3.1 成功写入数据，gpf_entry 储存写入的信息，gpf_valid 为真

4.3.1 此时 gpf_valid 为真，读取到 gpf 信息后 gpf_valid 为假。且读取到的数据与预设数据相符

5.8测试用例 test_read_write_gpf_bypass

这个用例用于测试在 bypass 的情况下进行有 ITLB 异常的写。此时不需要储存 entry

数据和 gpf 数据，将输入直接旁路到输出

目标功能点：4.3.2

步骤：①初始化及 flush ②同时进行读和带 ITLB 异常的写

预期结果：4.3.2 读通道立即读到含有 gpf 信息的预设数据，且 gpf_valid 未置 1，队列和 gpf_entry 不包含写入的信息

5.9测试用例 test_update_hit

这个用例用于测试命中更新的情况

目标功能点：5.1.1

步骤：①初始化及 flush ②写入预设数据 ③发送更新请求，且此次更新会命中 ④读取更新后的数据

预期结果：5.1.1 成功读到数据，且 waymask 和 metacodes 更新为 update 请求数据，其余仍与预设数据一致。

5.10 测试用例 test_update_miss

这个用例用于测试命中更新的情况

目标功能点：5.1.2

步骤：①初始化及 flush ②写入预设数据 ③发送更新请求，且此次更新不会命中 ④读取更新后的数据

预期结果：5.1.2 成功读到数据，且 waymask 清零，其余部分与预设数据一致。

5.11 测试用例 test_update_no

这个用例用于测试命中更新的情况

目标功能点：5.1.3

步骤：①初始化及 flush ②写入预设数据 ③发送更新请求，且此次不更新 ④读取更新后的数据

预期结果：5.1.3 成功读到数据，且读到的数据与预设数据一致，未发生改变。

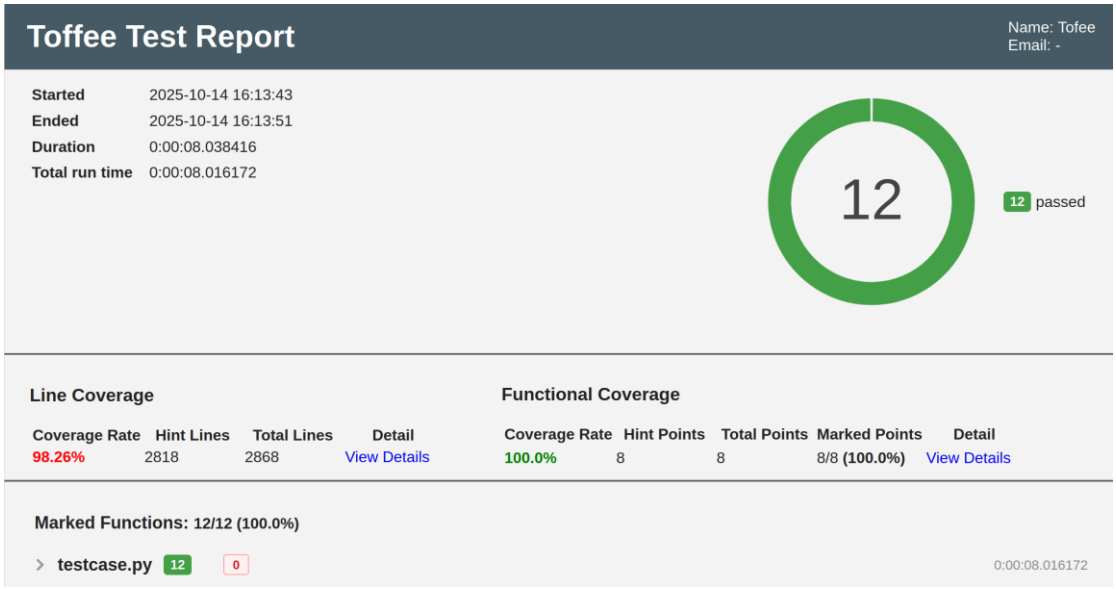
5.12 随机测试 test_random

此外，测试用例中包含随机测试 test_random，包含随机的数据读写及 update

6、结果分析

6.1 测试用例分析

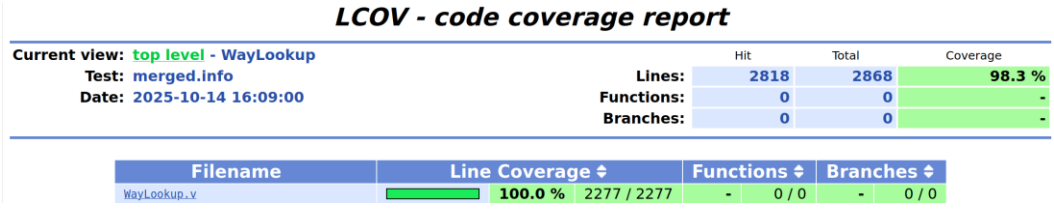
在所有测试用例的运行过程中，均覆盖了预期测试的功能点，完成了既定的目标。



6.2 覆盖率分析

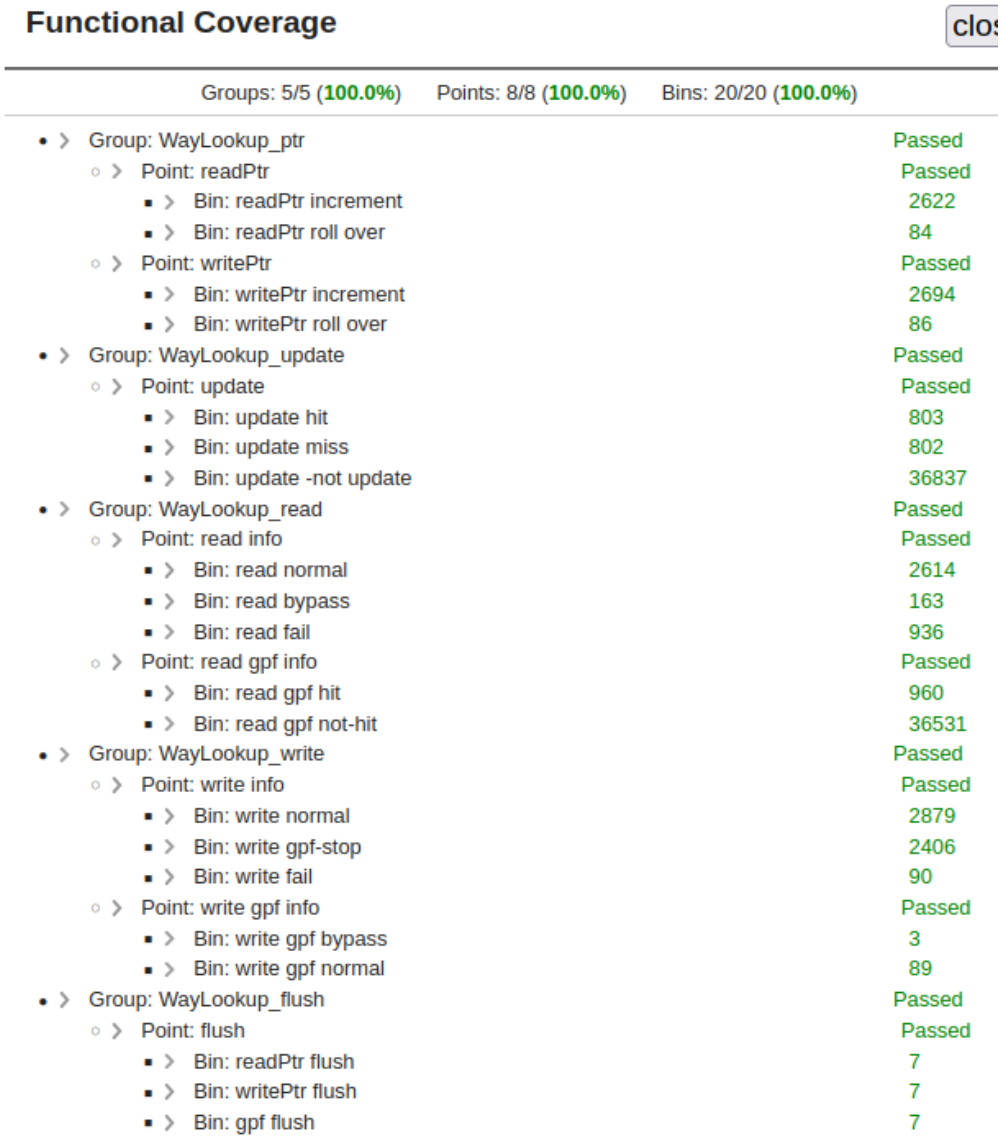
6.2.1 行覆盖率

完整运行所有测试，最终 WayLookup.v 的代码行覆盖率为 100%



6.2.2 功能覆盖率

根据最终产生的测试报告，所有功能均被覆盖且正确。



7、缺陷分析

验证结果显示 WayLookup 模块未存在缺陷。

8、测试结论

针对 WayLookup 的读写、更新等情况的测试情况显示，WayLookup 模块功能符合预期，行为与理想模型相符。综上，认为 Icache 中的 WayLookup 模块功能正常。