GLCC-FTB 模块验证报告

李雪涛

October, 2024

版本号：*v1.0*

目录

表格

1 验证对象

FTB 是香山 BPU 的第三个子预测器，它也能一并获取到 uFTB 和 TAGE-SC 的输出。在 FTB 的输入接口中，s1 通道含有 uFTB 的基础预测结果，s2 通道和 s3 通道中仅有 br_taken_mask 一组信号被 TAGE-SC 填充，并无 FTB 项生成的基础预测结果。FTB 的工作便是为 s2 和 s3 通道提供基础预测结果。

1.1 FTB 模块结构介绍

1. 缓存更多 FTB 项，为 s2 和 s3 通道提供基础预测结果。FTB 预测器的本质是一个较大容量的存储器，其会根据当前预测的 PC 读出对应的 FTB 项，并在 s2 阶段产出预测结果。与此同时该 FTB 项还会被再保存一个周期，生成 s3 阶段预测结果。生成结果需要注意的点是要考虑到上一预测器输入的 br_taken_mask 字段，避免在生成时丢失。
2. 根据更新请求，更新存储中的 FTB 项。支持 s1 通道基础预测结果输出及 meta 信息输出

1.2 FTB 模块功能介绍

功能点主要分为四个部分，分别为:

1. 支持 FTB 项存储
2. 支持 s2, s3 通道基础预测结果输出以及 meta 信息输出
3. 支持更新请求响应，更新内部 FTB 项

2 验证方案和验证框架

2.1 验证方案

本文基于 Picker 与 MLVP 库将 Verilog 导出为 python 形式的待测设计（Design Under Test, DUT）。以 python 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

在 Picker 导出的 DUT 基础上，该项目复用 FTB-with-ftq 所提供的 Bpu-top 模块以及 ftq 模块，并且进行一定程度上的修改使其能够执行测试用例。

参考模型 参考模型用于比对 DUT 的功能行为是否正确。在本次验证中，我们为 FTB 编写了参考模型，但是该参考模型并不完善，只能在一定程度上模拟 FTB 的行为，并不能精确的作为参考模型，仅能作为代码行为参考。见/env-xs-ov-00-bpu/tests/FTBs/FTB-bank/env/路径下的 ftb.py 文件

3 FTB 功能点和测试点

FTB 在功能和结构上都与 uFTB 类似，其主要区别就是 FTB 能够容纳更多的 FTB 项，并且 FTB 的预测结果是在 s2 与 s3 通道输出。正是由于容量大，其读出的速度上会比 uFTB 慢，无法被放置在第一周期产生预测结果，但大容量也使它能够获得更加精准的预测结果。在本次验证中，我们拆分了 4 个功能点和 20 个测试点。

3.1 接收控制信号

表 3.1.1: FTB 功能点 1 接收控制信号

编号	测试点	测试用例
FTB 1.1	能够接收来自 BPU_top 的 reset 控制信号，若为真，重置预测器和缓存	控制信号测试 1
FTB 1.2	能够接收来自 BPU_top 的 io_ctrl_btb_enable 控制信号，若为真，预测输出均为未命中	控制信号测试 2
FTB 1.3	能够接收来自 BPU_top 的 io_s0(1、2)_fire_0 控制信号	控制信号测试 3 控制信号测试 4-7

3.2 输出预测结果

表 3.2.1: FTB 功能点 2 输出预测结果

编号	测试点	测试用例
FTB 2.1	缓存命中，always taken 有效，s2 与 s3 通道输出的预测结果需为真	预测结果输出测试 1
FTB 2.2	缓存命中，always taken 无效，s2 与 s3 通道输出的预测结果需维持不变	预测结果输出测试 2
FTB 2.3	缓存不命中，always taken 有效，s2 与 s3 通道输出的预测结果需维持不变	预测结果输出测试 3
FTB 2.4	缓存不命中，always taken 无效，s2 与 s3 通道输出的预测结果需维持不变	预测结果输出测试 4

3.3 提供基础条件分支结果

表 3.3.1: FTB 功能点 3 提供基础条件分支结果

编号	测试点	测试用例
FTB 3.1	缓存命中，always taken 为 0，两比特饱和计数器值为 2’b00，需预测为假	预测结果输出测试 5
FTB 3.2	传入预测请求后，经过一个 s0_fire 有效周期，产生预测结果，但尚未输出	预测结果输出测试 5
FTB 3.3	预测结果将在 s1_fire 有效时，预测结果通过 s2 通道进行输出，若命中输出命中缓存，若未命中输出 hit 为 0	预测结果输出测试 5

表 3.3.1 – 接上页表格

编号	测试点	测试用例
FTB 3.4	预测结果将在 s2_fire 有效时，预测结果通过 s3 通道进行输出，若命中输出命中缓存，若未命中输出 hit 为 0	预测结果输出测试 5
FTB 3.5	预测结果将在一个 s1 流水有效周期和一个 s2 流水有效周期后，预测结果输出为 last_stage_meta	预测结果输出测试 5

3.4 执行更新

表 3.4.1: FTB 功能点 4 执行更新

编号	测试点	测试用例
FTB 4.1	s1_ready 无效时会执行更新步骤，但不会写入 FTB 项	提供基础条件分支结果测试 1
FTB 4.2	update_valid 有效时正常执行更新，无效时不进行更新	提供基础条件分支结果测试 2
FTB 4.3	io_update_bits_old_entry 无效时正常执行更新，有效时不进行更新	提供基础条件分支结果测试 3
FTB 4.4	若更新请求中的 meta hit 有效，更新仅需一个周期	提供基础条件分支结果测试 4
FTB 4.5	若更新请求中的 meta hit 无效，更新需要三个周期	提供基础条件分支结果测试 4
FTB 4.6	能根据更新请求的 pc 值计算目标组	提供基础条件分支结果测试 5
FTB 4.7	能够接收更新请求，更新请求若已被缓存则更新命中的缓存，内容直接覆盖	提供基础条件分支结果测试 6
FTB 4.8	更新请求若未命中，且目标组内 FTB 缓存未满，则有空让进	提供基础条件分支结果测试 7
FTB 4.9	更新请求若未命中，且目标组内 FTB 缓存已满，则根据 LRU 算法替换	提供基础条件分支结果测试 7

4 测试环境

4.1 硬件环境

1. AMD Ryzen 7 5800H
2. 16GB 内存

4.2 软件环境

操作系统:

1. 物理机：Windows 11 家庭中文版 23H2
2. 虚拟机：Linux-5.15.153.1-microsoft-standard-WSL2-x86_64-with-glibc2.35-WSL2.2.4.0

测试工具集:

名称	版本号
*python	3.10.12
*pytest	8.2.2
*pytest-xdist	3.6.1
*pytest-reporter-html1	0.9.0
*mlvp	0.0.1-master-6624111
*picker	0.1.0-master-36d6882
*cmake	3.15.7
*gcc/g++	11.4.0
*Verilator	4.218 2022-01-17
*verible-verilog-format	v0.0-3716-g914652db
*SWIG	4.2.1

其中，标注 * 的测试工具为必要的环境。测试代码详见FTBs。

5 FTB 测试用例

在本次验证中，总共有 20 个测试用例。FTB 的测试用例分为控制信号测试、预测结果输出测试、提供基础条件分支结果测试三个部分。

5.1 控制信号测试

5.1.1 reset 信号

表 5.1.1: FTB 测试用例组 1 控制信号测试 1 reset 信号

测试步骤	预期结果
步骤一：FTB 预测器初始化，将 reset 置 1，等待 s1_ready 步骤二：送入更新请求以缓存 4 条 FTB 项 步骤三：s0_fire、s1_fire 置为有效，送入上面 4 条 FTB 项的 pc，观察输出结果 步骤四：FTB 预测器初始化，将 reset 置 1 步骤五：s0_fire、s1_fire 置为有效，送入上面 4 条 FTB 项的 pc，观察输出结果	步骤三：预测结果均为命中且跳转 步骤五：预测结果均为不命中
覆盖测试点 接收控制信号	

5.1.2 io_ctrl_btb_enable 信号

表 5.1.2: FTB 测试用例组 1 控制信号测试 2 io_ctrl_btb_enable 信号

测试步骤	预期结果
步骤一：FTB 预测器初始化，将 reset 置 1，等待 s1_ready 步骤二：送入更新请求以缓存 4 条 FTB 项 步骤三：s0_fire、s1_fire 置为有效，io_ctrl_btb_enable 置为有效，送入上面 4 条 FTB 项的 pc，观察输出结果 步骤四：s0_fire、s1_fire 置为有效，io_ctrl_btb_enable 置为无效，送入上面 4 条 FTB 项的 pc，观察输出结果	步骤三：预测结果均为命中且跳转 步骤四：预测结果均为不命中
覆盖测试点 接收控制信号	

5.1.3 io_s0_fire_0 信号

方便起见，在控制信号测试 3-7 中，输入的三元组为 [io_s0_fire_0, io_s1_fire_0, io_s2_fire_0]。**X 的所有可能组合均需要覆盖**

表 5.1.3: FTB 测试用例组 1 控制信号测试 3 io_s0_fire_0 信号

测试步骤	预期结果
步骤一：FTB 预测器初始化，将 reset 置 1，等待 s1_ready 步骤二：送入更新请求以缓存 1 条 FTB 项，等待 3 个周期更新缓存	
步骤三：送入 [0, x, x]，及该 FTB 项的 pc，观测输出结果	步骤三：在 s2 和 s3 通道输出预期结果
步骤四：送入 [x, 1, 1]，及该 FTB 项的 pc，观测输出结果	步骤四：在 s2 和 s3 通道输出预期结果
步骤五：送入 [1, x, x]，及该 FTB 项的 pc，观测输出结果	步骤五：在 s2 和 s3 通道输出预期结果
步骤六：送入 [x, 1, 1]，及该 FTB 项的 pc，观测输出结果	步骤六：在 s2 和 s3 通道输出预期结果
覆盖测试点 接收控制信号	

5.1.4 io_s(1、2)_fire_0 控制信号（0，0）

表 5.1.4: FTB 测试用例组 1 控制信号测试 4 io_s(1、2)_fire_0 控制信号（0，0）

测试步骤	预期结果
步骤一：FTB 预测器初始化，将 reset 置 1，等待 s1_ready 步骤二：送入更新请求以缓存 1 条 FTB 项，等待 3 个周期更新缓存	
步骤三：送入 [1, x, x]，及该 FTB 项的 pc，观测输出结果	步骤三：输出不发生变化
步骤四：送入 [x, 0, 0]，及该 FTB 项的 pc，观测输出结果	步骤四：输出不发生变化
覆盖测试点 接收控制信号	

5.1.5 io_s(1、2)_fire_0 控制信号（0，1）

表 5.1.5: FTB 测试用例组 1 控制信号测试 5 io_s(1、2)_fire_0 控制信号（0，1）

测试步骤	预期结果
步骤一：FTB 预测器初始化，将 reset 置 1，等待 s1_ready 步骤二：送入更新请求以缓存 1 条 FTB 项，等待 3 个周期更新缓存	
步骤三：送入 [1, x, x]，及该 FTB 项的 pc，观测输出结果	步骤三：输出不发生变化
步骤四：送入 [x, 0, 1]，及该 FTB 项的 pc，观测输出结果	步骤四：命中，在 s3 通道输出预测结果

覆盖测试点 接收控制信号

5.1.6 io_s(1、2)_fire_0 控制信号（1，0）

表 5.1.6: FTB 测试用例组 1 控制信号测试 6 io_s(1、2)_fire_0 控制信号（1，0）

测试步骤	预期结果
步骤一：FTB 预测器初始化，将 reset 置 1，等待 s1_ready 步骤二：送入更新请求以缓存 1 条 FTB 项，等待 3 个周期更新缓存 步骤三：送入 [1, x, x]，及该 FTB 项的 pc，观测输出结果 步骤四：送入 [x, 1, 0]，及该 FTB 项的 pc，观测输出结果	 步骤三：输出不发生变化 命中，在 s2 通道输出预测结果
覆盖测试点 接收控制信号	

5.1.7 io_s(1、2)_fire_0 控制信号（1，1）

表 5.1.7: FTB 测试用例组 1 控制信号测试 7 io_s(1、2)_fire_0 控制信号（1，1）

测试步骤	预期结果
步骤一：FTB 预测器初始化，将 reset 置 1，等待 s1_ready 步骤二：送入更新请求以缓存 1 条 FTB 项，等待 3 个周期更新缓存 步骤三：送入 [1, x, x]，及该 FTB 项的 pc，观测输出结果 步骤四：送入 [x, 1, 1]，及该 FTB 项的 pc，观测输出结果	 步骤三：输出不发生变化 命中，在 s2 和 s3 通道输出预测结果
覆盖测试点 接收控制信号	

5.1.8 last_stage_meta 信号输出

表 5.1.8: FTB 测试用例组 1 控制信号测试 8 last_stage_meta 信号输出

测试步骤	预期结果
步骤一：FTB 预测器初始化，将 reset 置 1，等待 s1_ready 步骤二：送入更新请求以缓存 4 条 FTB 项，等待 3 个周期更新缓存 步骤三：送入 [1, x, x]，及每个 FTB 项的 pc，观测输出结果	 步骤三：io_out_last_stage_meta 为不发生变化

步骤四：送入 [0, 1, 0]，及每个 FTB 项的 pc，观测输出的 last_stage_meta 步骤五：送入 [0, 0, 1]，及每个 FTB 项的 pc，观测输出的 last_stage_meta 步骤六：送入 [0, 1, 1]，及每个 FTB 项的 pc，观测输出的 last_stage_meta	步骤四：io_out_last_stage_meta 为不发生变化 步骤五：输出的 io_out_last_stage_meta 含有上一周期预测结果，其源信息符合 io_out_last_stage_meta = 213'h0, resp_meta_pred_way_r_1, resp_meta_hit_r_1 的规则 步骤六：输出的 io_out_last_stage_meta 含有上一周期预测结果，其源信息符合 io_out_last_stage_meta = 213'h0, resp_meta_pred_way_r_1, resp_meta_hit_r_1 的规则
覆盖测试点 执行更新	

5.2 预测结果输出测试

方便起见，以下内容中，输入的四元组为:
[io_in_bits_resp_in_0_s2_full_pred_0_br_taken_mask_0,
io_in_bits_resp_in_0_s2_full_pred_0_br_taken_mask_1,
io_in_bits_resp_in_0_s3_full_pred_0_br_taken_mask_0,
io_in_bits_resp_in_0_s3_full_pred_0_br_taken_mask_1]
输出的四元组为:
[io_out_s2_full_pred_0_br_taken_mask_0,
io_out_s2_full_pred_0_br_taken_mask_1,
io_out_s3_full_pred_0_br_taken_mask_0,
io_out_s3_full_pred_0_br_taken_mask_1]

5.2.1 缓存命中，always taken 有效

表 5.2.1: FTB 测试用例组 2 预测结果输出测试 1

测试步骤	预期结果
步骤一：FTB 预测器初始化，将 reset 置 1，等待 s1_ready 步骤二：送入更新请求以缓存 1 条 FTB 项，tail_slot 有效且为 sharing 状态，io_update_valid 有效，且 io_update_bits_ftb_entry_always_taken_0 有效，空转两个时钟周期等待 FTB 项更新完毕	

步骤三：输入该 FTB 项的 pc，io_s0(1、2)_fire_0 均有效，输入 [0, 0, 0, 0]，在下一周期预测结果 步骤四：送入更新请求以缓存 1 条 FTB 项，tail_slot 有效且为 sharing 状态，io_update_valid 有效，且 io_update_bits_ftb_entry_always_taken_1 有效，空转两个时钟周期等待 FTB 项更新完毕 ‘ 步骤五：输入该 FTB 项的 pc，io_s0(1、2)_fire_0 均有效，输入 [0, 0, 0, 0]，在下一周期预测结果 步骤六：送入更新请求以缓存 1 条 FTB 项，tail_slot 有效且为 sharing 状态，io_update_valid 有效，且 io_update_bits_ftb_entry_always_taken_0(1) 均有效，空转两个时钟周期等待 FTB 项更新完毕 步骤七：输入该 FTB 项的 pc，io_s0(1、2)_fire_0 均有效，输入 [0, 0, 0, 0]，在下一周期预测结果	步骤三：预测结果为 [1, 0, 1, 0]，s3 通道的输出结果会延迟一个周期 步骤五：预测结果为 [0, 1, 0, 1]，s3 通道的输出结果会延迟一个周期 步骤七：预测结果为 [1, 1, 1, 1]，s3 通道的输出结果会延迟一个周期
覆盖测试点 输出预测结果	

5.2.2 缓存命中，always taken 无效

表 5.2.2: FTB 测试用例组 2 预测结果输出测试 2

测试步骤	预期结果
步骤一：FTB 预测器初始化，将 reset 置 1，等待 s1_ready 步骤二：送入更新请求以缓存 1 条 FTB 项，tail_slot 有效且为 sharing 状态，io_update_valid 有效，空转两个时钟周期等待 FTB 项更新完毕 步骤三：输入该 FTB 项的 pc，io_s0(1、2)_fire_0 均有效，输入 [0, 0, 0, 0]，在下一周期预测结果 步骤四：送入更新请求以缓存 1 条 FTB 项，tail_slot 有效且为 sharing 状态，io_update_valid 有效，空转两个时钟周期等待 FTB 项更新完毕 步骤五：输入该 FTB 项的 pc，io_s0(1、2)_fire_0 均有效，输入 [1, 1, 1, 1]，在下一周期预测结果	步骤三：预测结果为 [0, 0, 0, 0] 步骤五：预测结果为 [1, 1, 1, 1]
覆盖测试点 输出预测结果	

5.2.3 缓存不命中，always taken 有效

表 5.2.3: FTB 测试用例组 2 预测结果输出测试 3

测试步骤	预期结果
步骤一：FTB 预测器初始化，将 reset 置 1，等待 s1_ready 步骤二：送入更新请求以缓存 1 条 FTB 项，tail_slot 有效且为 sharing 状态，io_update_valid 有效，且 io_update_bits_ftb_entry_always_taken_0(1) 均有效，空转两个时钟周期等待 FTB 项更新完毕 步骤三：输入不是该 FTB 项的 pc，io_s0(1、2)_fire_0 均有效，输入 [0, 0, 0, 0]，在下一周期预测结果 步骤四：送入更新请求以缓存 1 条 FTB 项，tail_slot 有效且为 sharing 状态，io_update_valid 有效，且 io_update_bits_ftb_entry_always_taken_0(1) 均有效，空转两个时钟周期等待 FTB 项更新完毕 步骤五：输入不是该 FTB 项的 pc，io_s0(1、2)_fire_0 均有效，输入 [1, 1, 1, 1]，在下一周期预测结果	步骤三：预测结果为 [0, 0, 0, 0] 步骤五：预测结果为 [1, 1, 1, 1]
覆盖测试点 输出预测结果	

5.2.4 缓存不命中，always taken 无效

表 5.2.4: FTB 测试用例组 2 预测结果输出测试 4

测试步骤	预期结果
步骤一：FTB 预测器初始化，将 reset 置 1，等待 s1_ready 步骤二：送入更新请求以缓存 1 条 FTB 项，tail_slot 有效且为 sharing 状态，io_update_valid 有效，空转两个时钟周期等待 FTB 项更新完毕 步骤三：输入不是该 FTB 项的 pc，io_s0(1、2)_fire_0 均有效，输入 [0, 0, 0, 0]，在下一周期预测结果 步骤四：送入更新请求以缓存 1 条 FTB 项，tail_slot 有效且为 sharing 状态，io_update_valid 有效，空转两个时钟周期等待 FTB 项更新完毕 步骤五：输入不是该 FTB 项的 pc，io_s0(1、2)_fire_0 均有效，输入 [1, 1, 1, 1]，在下一周期预测结果	步骤三：预测结果为 [0, 0, 0, 0] 步骤五：预测结果为 [1, 1, 1, 1]
覆盖测试点 输出预测结果	

5.2.5 tailSlot_sharing

表 5.2.5: FTB 测试用例组 2 预测结果输出测试 5

测试步骤	预期结果
步骤一：uFTB 预测器初始化，将 reset 置 1 步骤二：送入更新请求以缓存 1 条 FTB 项，tail_slot 有效且不处于 sharing 状态，io_update_valid 有效，输入 [0, 0]，且 io_update_bits_ftb_entry_always_taken_1 有效	
步骤三：输出该 FTB 项的预测结果	步骤三：预测结果为 [0, 1]
步骤四：io_update_valid 有效，输入 [0, 0]	
步骤五：输出该 FTB 项的预测结果	步骤五：预测结果为 [0, 1]
步骤六：io_update_valid 有效，输入 [0, 0]	
步骤七：输出该 FTB 项的预测结果	步骤七：预测结果为 [0, 1]
覆盖测试点 提供基础条件分支结果	

5.3 提供基础条件分支结果测试

5.3.1 s1_ready

表 5.3.1: FTB 测试用例组 3 提供基础条件分支结果测试 1

测试步骤	预期结果
步骤一：FTB 预测器初始化，将 reset 置 1，不等待 s1_ready 步骤二：送入更新请求以缓存 1 条 FTB 项，等待两个时钟周期缓存完毕	
步骤三：输出该 FTB 项的预测结果	
步骤四：等待 s1_ready	步骤四：输出为未命中
步骤五：送入更新请求以重新缓存 1 条 FTB 项，等待两个时钟周期缓存完毕	
步骤六：输出该 FTB 项的预测结果	步骤六: 输出的 FTB 项与送入的 FTB 项内容一致
覆盖测试点 执行更新	

5.3.2 FTB 项更新

表 5.3.2: FTB 测试用例组 3 提供基础条件分支结果测试 2

测试步骤	预期结果
步骤一：FTB 预测器初始化，将 reset 置 1 步骤二：送入更新请求以缓存 1 条 FTB 项 步骤三：送入更新请求以更新该 FTB 项，除了 pc 外的所有值均置随机数	

步骤四：输出该 FTB 项的预测结果	步骤四: 输出的 FTB 项与送入的 FTB 项更新后的内容一致
覆盖测试点 执行更新	

5.3.3 io_update_valid

表 5.3.3: FTB 测试用例组 3 提供基础条件分支结果测试 3

测试步骤	预期结果
步骤一：FTB 预测器初始化，将 reset 置 1，等待 s1_ready 步骤二：送入更新请求以缓存 1 条 FTB 项，等待两个时钟周期缓存完毕 步骤三：送入更新请求以更新该 FTB 项，io_update_valid 置为无效，空转两个时钟周期等待更新完成 步骤四：输出该 FTB 项的预测结果 步骤五：送入更新请求以更新该 FTB 项，io_update_valid 置为有效，空转两个时钟周期等待更新完成 步骤六：输出该 FTB 项的预测结果	步骤四: 输出的 FTB 项未被更新 步骤六: 输出的 FTB 项与送入的 FTB 项更新后的内容一致
覆盖测试点 执行更新	

5.3.4 io_update_bits_old_entry

表 5.3.4: FTB 测试用例组 3 提供基础条件分支结果测试 4

测试步骤	预期结果
步骤一：FTB 预测器初始化，将 reset 置 1，等待 s1_ready 步骤二：送入更新请求以缓存 1 条 FTB 项，等待两个时钟周期缓存完毕 步骤三: 送入更新请求以更新该 FTB 项, io_update_bits_old_entry 置为有效，空转两个时钟周期等待更新完成 步骤四：输出该 FTB 项的预测结果 步骤五: 送入更新请求以更新该 FTB 项, io_update_bits_old_entry 置为无效，空转两个时钟周期等待更新完成 步骤六：输出该 FTB 项的预测结果	步骤四: 输出的 FTB 项未被更新 步骤六: 输出的 FTB 项与送入的 FTB 项更新后的内容一致
覆盖测试点 执行更新	

5.3.5 FTB 项更新，meta hit

表 5.3.5: FTB 测试用例组 3 提供基础条件分支结果测试 5

测试步骤	预期结果
步骤一：FTB 预测器初始化，将 reset 置 1，等待 s1_ready 步骤二：送入更新请求以缓存 1 条 FTB 项，等待两个时钟周期缓存完毕步骤三：送入更新请求以更新该 FTB 项，meta hit 无效 步骤四：在等待两个周期该 FTB 项才成功更新 步骤五：送入更新请求以更新该 FTB 项，meta hit 无效 步骤六：在等待一个周期该 FTB 项即可成功更新	 步骤四：在等待两个周期该 FTB 项才成功更新 步骤六：在等待一个周期该 FTB 项即可成功更新
覆盖测试点 执行更新	

5.3.6 能根据更新请求的 pc 值计算目标组

表 5.3.6: FTB 测试用例组 3 提供基础条件分支结果测试 6

测试步骤	预期结果
步骤一：FTB 预测器初始化，将 reset 置 1，等待 s1_ready 步骤二：送入更新请求以缓存 1 条 FTB 项，等待两个时钟周期使其更新完毕 步骤三：送入该 FTB 项 pc ，s0(1, 2)_fire 均置为有效，等待一个时钟周期获得 last_stage_meta，比较 meta 中的 hit_way 信息是否符合预期的位置 步骤四: 重复步骤二与步骤三, 直至缓存的 FTB 项涵盖所有 FTB-Bank 缓存位置	 步骤三：hit_way 信息符合预期
覆盖测试点 执行更新	

5.3.7 缓存更新未命中，LRU 更新测试

表 5.3.7: FTB 测试用例组 3 提供基础条件分支结果测试 7

测试步骤	预期结果
步骤一：FTB 预测器初始化，将 reset 置 1，等待 s1_ready 步骤二：送入更新请求以缓存 4 条 FTB 项 步骤三：每个周期 s0_fire 置为有效，依次更新步骤二前 2 条 FTB 项，观察输出结果 步骤四：送入更新请求以缓存新的 2 条 FTB 项 步骤五：每个周期 s0_fire 置为有效，依次送入步骤二的后 2 条 FTB 项，观察输出结果	 步骤三: 输出的 FTB 项与送入的 FTB 项更新后的内容一致 步骤五：输出未命中

步骤六：每个周期 s0_fire 置为有效，依次送入步骤四的 2 条 FTB 项，观察输出结果	步骤六: 输出的 FTB 项与步骤四中送入的 FTB 项内容一致
覆盖测试点 执行更新	

5.3.8 缓存更新未命中，LRU 不更新测试

表 5.3.8: FTB 测试用例组 3 提供基础条件分支结果测试 8

测试步骤	预期结果
步骤一：FTB 预测器初始化，将 reset 置 1，等待 s1_ready	
步骤二：送入更新请求以缓存 4 条 FTB 项	
步骤三：每个周期 s0_fire 置为有效，依次送入步骤二前 2 条 FTB 项，观察输出结果	步骤三: 输出的 FTB 项与送入的 FTB 项更新后的内容一致
步骤四：送入更新请求以缓存新的 2 条 FTB 项	
步骤五：每个周期 s0_fire 置为有效，依次送入步骤二的后 2 条 FTB 项，观察输出结果	步骤五：输出未命中
步骤六：每个周期 s0_fire 置为有效，依次送入步骤四的 2 条 FTB 项，观察输出结果	步骤六: 输出的 FTB 项与步骤四中送入的 FTB 项内容一致
覆盖测试点 执行更新	

6 FTB 结果分析

这里是对整个测试结果的简单分析和介绍。

6.1 测试用例分析

控制信号测试 4 不通过。问题描述见缺陷分析7。

6.2 行覆盖率分析

表 6.2.1: FTB 行覆盖率

名称	命中行数	总行数	行覆盖率
DelayNWithValid.sv	19	20	95.0%
DelayNWithValid_1.sv	127	128	99.2%
DelayN_2.sv	15	15	100%
DelayN_4.sv	9	9	100%
ext_sram.v	16	16	100%
FTBBank.sv	4419	4420	99.9%
SRAMTemplate_13.sv	182	183	99.5%
array_3.sv	5	5	100%
FTB.v	700	703	99.6%

DelayNWithValid.sv 文件、DelayNWithValid_1.sv 文件、FTBBank.sv 文件、SRAMTemplate_13.sv 文件均只有一行未被覆盖，为 initial 块中的判断语句，由于 initial 块中使 reset 生效需要设置 dut.reset.AsImmWrite 将引脚置成立即写入，故在一次测试中不能覆盖两种情况，该行语句无法被覆盖。实际上，为了测试该行语句的功能，我通过进行两次测试，引脚写入两种值，验证结果均符合预期，可以视为该行语句已经被覆盖。

FTB.v 文件的行覆盖率为 **99.6%**。

未覆盖到的代码仅有三行，其中一行未覆盖的原因同上，另两行未覆盖的原因为信号不可达，所以无法覆盖。

6.3 功能覆盖率分析

表 6.3.1: FTB 功能覆盖率

命中功能点	总功能点	功能覆盖率
20	20	100%

所有功能点均被覆盖，功能覆盖率达到 100%。但是 FTB 存在功能上的错误，详见缺陷分析7。

7 缺陷分析

Bug 1： reset 信号不会清空 FTB 内部的寄存器

reset 信号不会清空 FTB 内部的寄存器，会导致 reset 之后虽然不进行任何写操作但是读操作会仍然会命中的情况

触发过程：

1. 向 FTB 写入 FTB 项
2. 在写入成功后将 s0_fire, s1_fire, s2_fire 分别置为 1, 1, 1，读出该项
3. 得到正确的读出结果后，将 s0_fire, s1_fire, s2_fire 分别置为 0, 0, 0
4. 向 FTB 发送 reset 信号，等待 reset 完成 s1_ready 变为有效
5. 将 s0_fire, s1_fire, s2_fire 分别置为 1, 1, 1
6. 在读接口传入步骤 1 的 FTB 项，在更新接口传入任意有效请求但是需要使 meta 信号为 0 以确保 pred_rdata_REG 信号一直无效
7. 可以发现，虽然 reset 之后 FTB 缓存已经被清空，但是步骤 6 读请求的结果依然是命中的。
8. 除此之外，如果在第三步的时候 s1_fire 信号置 1，可以在 reset 的过程中观察到读请求的结果一直是命中的

期待表现：

步骤 6 读请求的结果不应该命中

问题表项：

在整个 reset 的过程中观察 pred_rdata_hold_data_0_tag 和 pred_rdata_hold_data_0_entry_valid 这两个寄存器，理论上 reset 之后这两个寄存器应该被清空，但从波形图来看这两个寄存器仍然维持 reset 之前的值。

从 Verilog 代码中来看，如图1，FTB 和 FTBBank 的所有寄存器值都没有在 reset 的时候被重置，导致寄存器仍维持 reset 之前的值，reset 功能异常。

```
always @(posedge clock or posedge reset) begin  // @[src/main/scala/xiangshan/frontend/FTB.
scala:284:7]
  if (reset) begin  // @[src/main/scala/xiangshan/frontend/FTB.scala:284:7]
    s2_hit_dup_0 <= 1'h0; // @[src/main/scala/xiangshan/frontend/BPU.scala:176:26, src/main/
    scala/xiangshan/frontend/FTB.scala:432:49]
    s2_hit_dup_1 <= 1'h0; // @[src/main/scala/xiangshan/frontend/BPU.scala:176:26, src/main/
    scala/xiangshan/frontend/FTB.scala:432:49]
    s2_hit_dup_2 <= 1'h0; // @[src/main/scala/xiangshan/frontend/BPU.scala:176:26, src/main/
    scala/xiangshan/frontend/FTB.scala:432:49]
    s2_hit_dup_3 <= 1'h0; // @[src/main/scala/xiangshan/frontend/BPU.scala:176:26, src/main/
    scala/xiangshan/frontend/FTB.scala:432:49]
    s3_hit_dup_0 <= 1'h0; // @[src/main/scala/xiangshan/frontend/BPU.scala:176:26, src/main/
    scala/xiangshan/frontend/FTB.scala:433:76]
    s3_hit_dup_1 <= 1'h0; // @[src/main/scala/xiangshan/frontend/BPU.scala:176:26, src/main/
    scala/xiangshan/frontend/FTB.scala:433:76]
    s3_hit_dup_2 <= 1'h0; // @[src/main/scala/xiangshan/frontend/BPU.scala:176:26, src/main/
    scala/xiangshan/frontend/FTB.scala:433:76]
    s3_hit_dup_3 <= 1'h0; // @[src/main/scala/xiangshan/frontend/BPU.scala:176:26, src/main/
    scala/xiangshan/frontend/FTB.scala:433:76]
  end
```

图 1: Bug

代码分析：

FTB 应该在 reset 信号有效时重置所有的寄存器。图2为 reset 信号有效后 1000 周期的波形图，也就是触发过程中第五步及之后的波形图

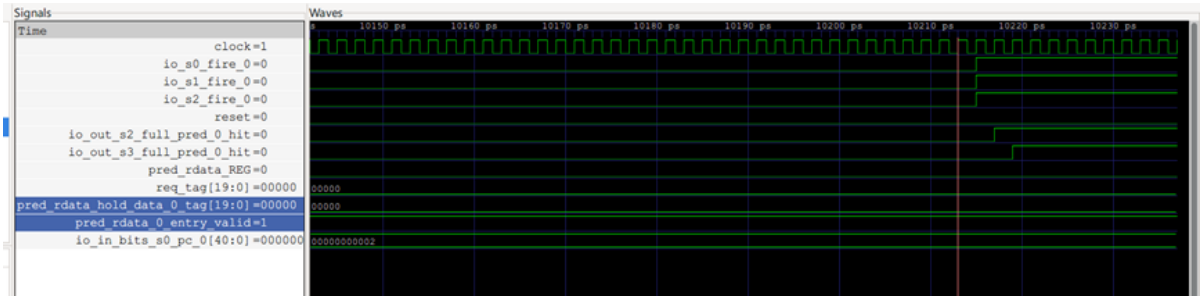

图 2: 波形图

8 测试结论

针对 FTB 项缓存、更新、预测的用例测试显示，FTB 大部分功能符合预期。发现的 Bug 经回归测试问题已被修复。综上所述，在验证内容范围内，FTB 功能基本正确。