



万众一芯：基于开源众包 芯片验证的探索与实践

万众一芯验证团队

报告人：姚治成

谢云龙 刘锦程 姚治成 葛琪 宋方圆 高梓源 杨尚锦 朱金彬 王俊越
陈潇 岳俊宇 饶嘉怡 杨泽辰 周泓韬 李小龙 马久跃 刘珊 安旭
朱航 徐易难 余子濠 陈璐 孟建熠 唐丹 王卅 包云岗

万众一芯验证团队



北京开源芯片研究院
BEIJING INSTITUTE OF OPEN SOURCE CHIP





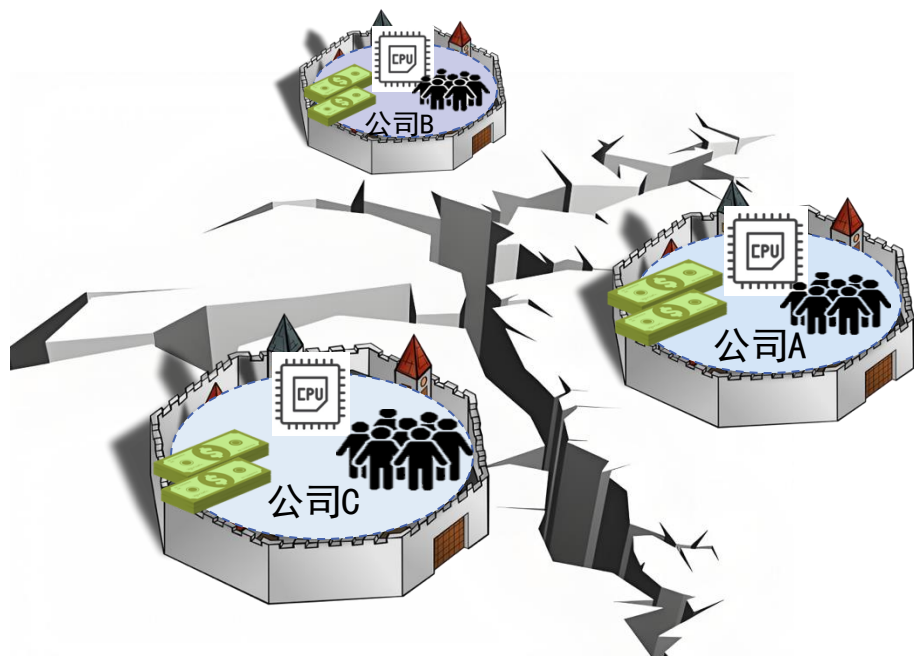
目录

当前芯片验证面临的挑战

开源众包验证的可行性

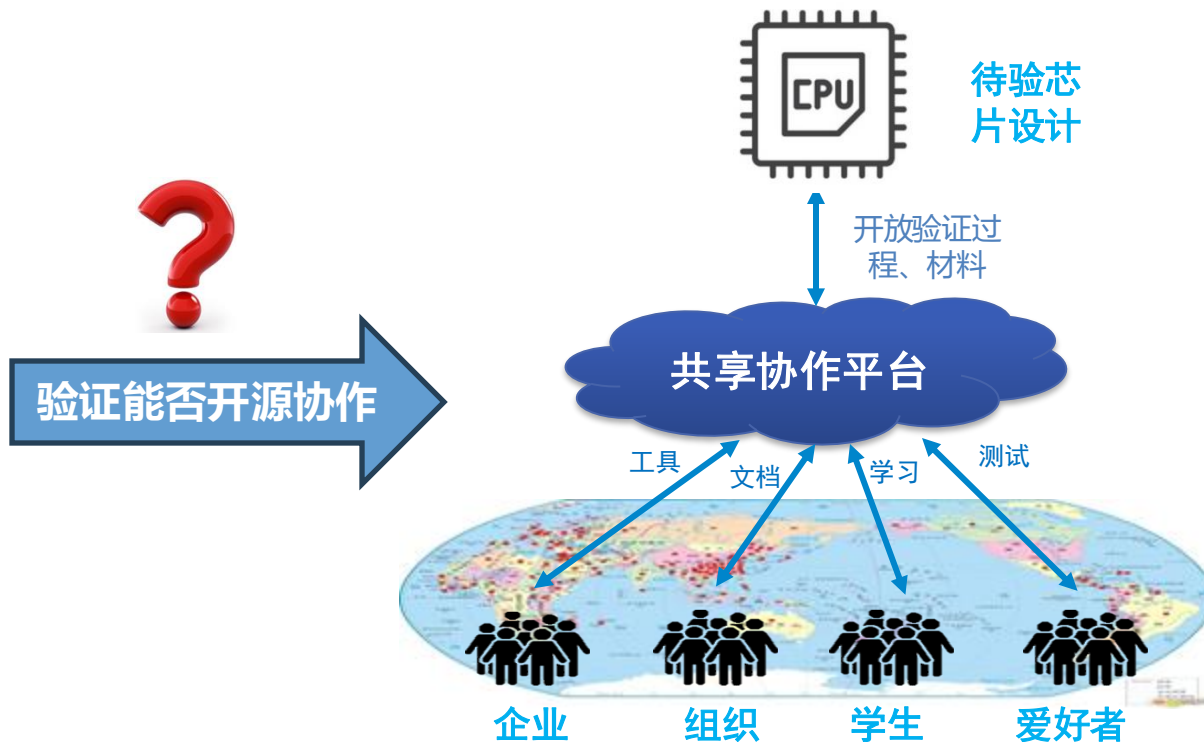
万众一芯开源验证项目

闭源硬件验证



城堡/烟囱模式：各验各的，
人力成本巨大

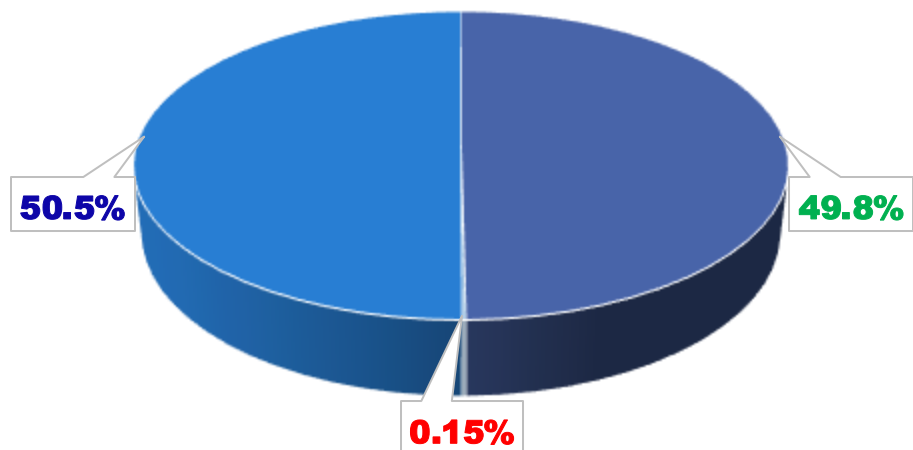
开源硬件众包验证



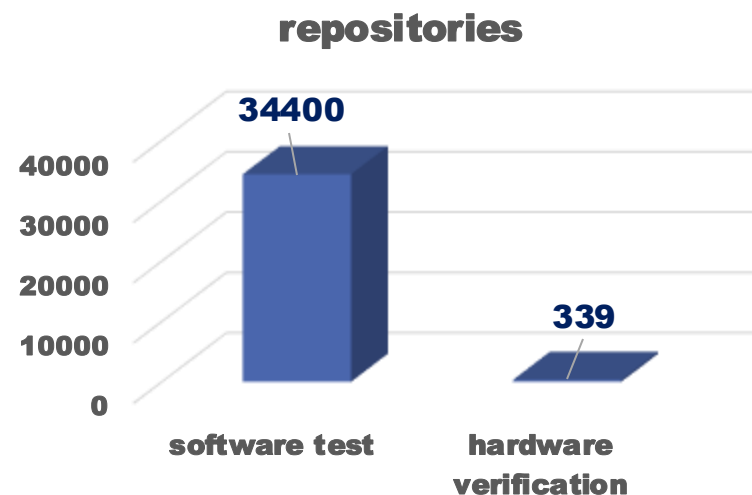
协作模式：开源共享

挑战：**开发者基数少**、学习材料少、生态薄弱等

2024第一季度各语言Github Push占比



- 49.8% 传统软件编程 (Python, C/C++, Java, Golang)
- 0.15% 传统硬件验证 (System Verilog/Verilog)
- 50.5% 其他



Github上软件测试与硬件验证仓库数对比

既然软件从业人数远多于硬件 (数量级差距)，是否可以让软件开发/测试人员参与到开源芯片验证？

相同点多：

- **目标相同**，都是发现待测试软件/硬件中的缺陷
- **指标类似**，都有代码行覆盖率，单元测试，功能覆盖率等指标
- **流程类似**，都有编写测试/验证方案，编写测试用例，bug管理与分析，测试报告等
- **方法类似**，都需要搭建测试环境，基于测试框架编写测试用例，测试理论通用
- **环境类似**，基本都是基于“软件环境”进行
- **管理类似**，人员管理，bug管理，质量要求相同

不同点少：

- **编程语言不同**，软件测试所用语言众多，硬件验证比较单一，主要为 System Verilog，**入门难，生态弱**
- **待测部分特征不同**，电路需要考虑时序（时钟）
- **生态不一样**，**软件测试生态丰富，框架与案例众多，硬件验证相对封闭，工具与案例匮乏**



如果不考虑编程语言的差异，且把电路特征进行隐藏（或简单学习），是否软件开发/测试人员就能以其熟悉的方式进行芯片验证呢？

实际上，芯片的大部分验证都是基于软件仿真器进行的，先把“RTL+测试代码”一起转换成C++，然后编译成电路仿真器进行模拟，**因此芯片验证某种程度上可以看作软件测试**



Who

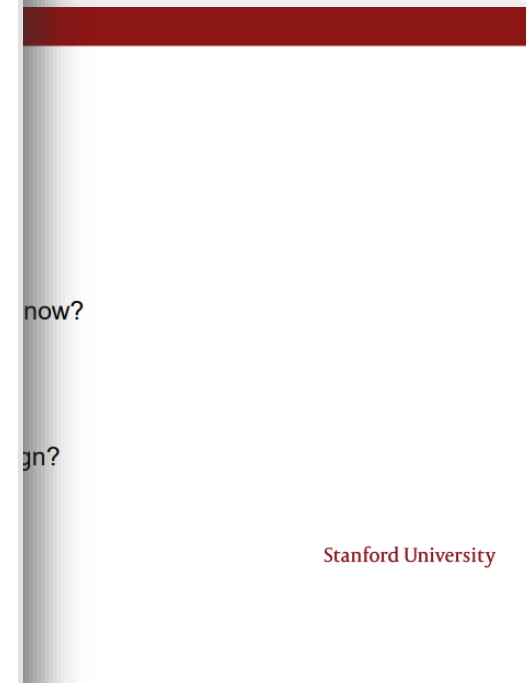
- Need hardware / software co-design

Application Designers

- Yes, software people

38

Stanford University



**Life Post Moore's Law: The New CAD Frontier
(Prof. Mark Horowitz, Stanford University
Keynote, MICRO 2023)**

万众一芯开源验证 (UnityChip Verification)



目标：以开源众包的方式让所有感兴趣的人参与芯片验证



API

Design Under Test



硬件生态



软件生态

挑战一：用哪种语言进行验证

挑战二：软件层面如何体现电路特征

挑战三：如何使用现有生态，如何建设生态

挑战一：用哪种语言进行验证

# Ranking	Programming Language	Percentage (YoY Change)	YoY Trend
1	Python	16.925% (-0.284%)	
2	Java	11.708% (+0.393%)	
3	Go	10.262% (-0.162%)	
4	JavaScript	9.859% (+0.306%)	^
5	C++	9.459% (-0.624%)	v
6	TypeScript	7.345% (-0.554%)	
7	PHP	5.665% (+0.357%)	
8	Ruby	4.706% (-0.307%)	
9	C	4.616% (+0.208%)	
10	C#	3.442% (+0.300%)	

动态语言，面向数据处理、AI等

静态语言，面向服务（云）端

动态语言，浏览器端执行（前端）

...

每种编程语言都有自己的
独有特点、群体与优势

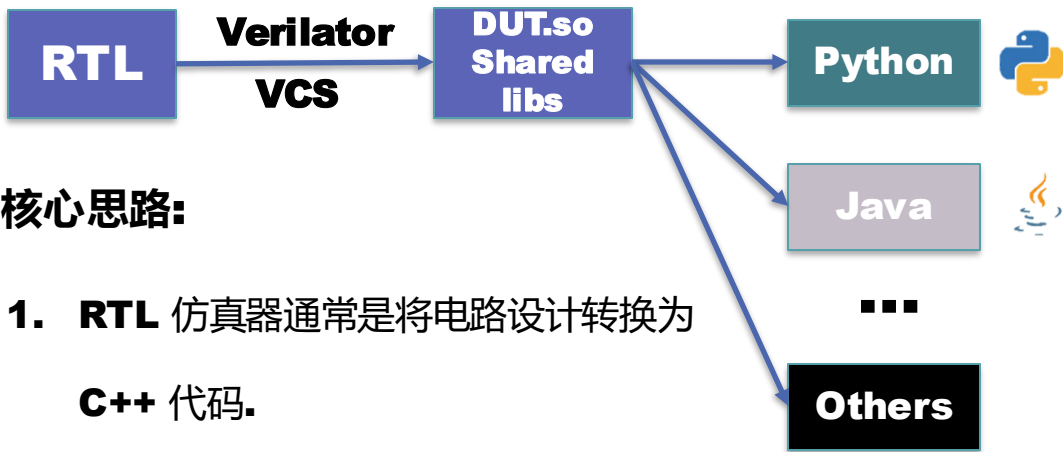
https://madnight.github.io/githut/#/pull_requests/2024/1

让参与者用自己擅长的编程语言进行芯片验证

多语言芯片验证转换工具picker



- 可以将 RTL 转换为常见高级编程语言 (目前已支持: Python, Java, Scala, C++ 和 Go).
- 转换后的RTL以库或者模块的方式存在, 与对应编程语言的测试环境兼容



核心思路:

1. RTL 仿真器通常是将电路设计转换为 C++ 代码.
2. C++ 可以被编译为so动态库
3. 几乎所有的编程语言都可以调用动态库
4. 上述过程可以被自动化

```
5 import UT_RAS as ras # import RTL as python module (UT_RAS is generated by picker)
6 import ras_pins as p # DUT wrapper functions (rest, pop, push, commit, redirect) based on pins
7
8 # pytest automatically recognizes test functions start with "test"
9 def test_ras_push_compress():
10     # The RAS (Return Address Stack) module in the BPU supports identical address compression.
11     # When pushing the same address, the stack pointer TOSR_value does not increase.
12     # Test start:
13     dut = ras.DUTRAS(waveform_filename="RAS.fst", # 1. new DUT (RAS module)
14                     coverage_filename="RAS.dat") # can custom wave/coverage output name
15     dut.init_clock("clock") # 2. init clock
16     ras_func = p.RASpins(dut) # 3. bind dut with wrapper class
17     ras_func.reset() # 4. reset dut
18     meta = None
19     for i in range(5): # 5. push same data 5 times
20         stack_top, meta = ras_func.push(0x100001)
21     assert meta["TOSR_value"] == 0, "TOSR_value should be zero" # 6. check result
22     dut.finalize() # 7. delete dut
```

Pytest 测试代码

基于Python和Pytest框架对香山BPU中的模块进行验证。
该过程几乎与传统python软件测试一致



多语言芯片验证转换工具picker



```
// A verilog 64-bit full adder with carry in and carry out

module Adder #(
    parameter WIDTH = 64
) (
    input [WIDTH-1:0] a,
    input [WIDTH-1:0] b,
    input cin,
    output [WIDTH-1:0] sum,
    output cout
);

assign {cout, sum} = a + b + cin;

endmodule
```

简单加法器RTL

picker export Adder.v --lang python

```
.
|-- Adder.fst # 测试的波形文件
|-- UT_Adder
|   |-- Adder.fst.hier
|   |-- _UT_Adder.so # Swig生成的wrapper动态库
|   |-- __init__.py # Python Module的初始化文件, 也是库的定义文件
|   |-- libDPIAdder.a # 仿真器生成的库文件
|   |-- libUTAdder.so # 基于dut_base生成的libDPI动态库封装
|   |-- libUT_Adder.py # Swig生成的Python Module
|   |-- xspcomm # xspcomm基础库, 固定文件夹, 不需要关注
|-- example.py # 示例代码
```

Python模块

```
from UT_Adder import *
import random

# 生成无符号随机数
def random_int():
    return random.randint(-(2**63), 2**63 - 1) & ((1 << 63) - 1)

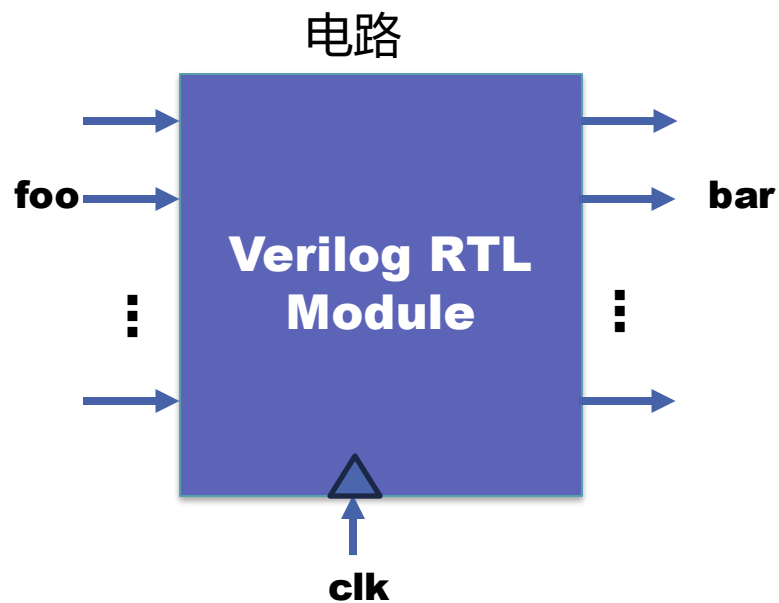
# 通过python实现的加法器参考模型
def referce_adder(a, b, cin):
    sum = (a + b) & ((1 << 64) - 1)
    carry = sum < a
    sum += cin
    carry = carry or sum < cin
    return sum, 1 if carry else 0

def random_test():
    # 创建DUT
    dut = DUTAdder()
    # 默认情况下, 引脚赋值不会立马写入, 而是在下一次时钟上升沿写入, 这对于时序电路适用, 但是Adder为组合电路, 所以需要
    # 因此需要调用AsImmWrite()方法更改引脚赋值行为
    dut.a.AsImmWrite()
    dut.b.AsImmWrite()
    dut.cin.AsImmWrite()
    # 循环测试
    for i in range(114514):
        a, b, cin = random_int(), random_int(), random_int() & 1
        # DUT: 对Adder电路引脚赋值, 然后驱动组合电路 (对于时序电路, 或者需要查看波形, 可通过dut.Step()进行驱动)
        dut.a.value, dut.b.value, dut.cin.value = a, b, cin
        dut.RefreshComb()
        # 参考模型: 计算结果
        ref_sum, ref_cout = referce_adder(a, b, cin)
        # 检查结果
        assert dut.sum.value == ref_sum, "sum mismatch: 0x{dut.sum.value:x} != 0x{ref_sum:x}"
        assert dut.cout.value == ref_cout, "cout mismatch: 0x{dut.cout.value:x} != 0x{ref_cout:x}"
        print(f"[test {i}] a=0x{a:x}, b=0x{b:x}, cin=0x{cin:x} => sum: 0x{ref_sum}, cout: 0x{ref_cout}")
    # 完成测试
    dut.Finish()
    print("Test Passed")

if __name__ == "__main__":
    random_test()
```

测试代码 (验证代码)

挑战二：软件层面如何体现电路特征



天然的并行性

顺序执行

时序与流水线

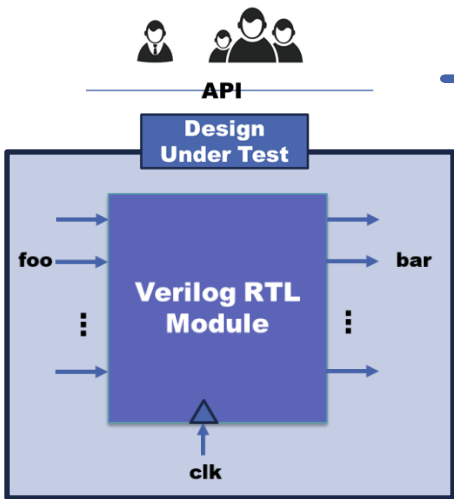
无对应特征

程序

```
function_exists("hex2rgb") {
    $hex_str = $hex_str;
    $return_string = "";
    $rgb_array = "";
    $strim($hex_str);
    $color_val = $hex_str;
    $rgb_array["r"] = $hex_val($color_val, 0, 1);
    $rgb_array["g"] = $hex_val($color_val, 2, 1);
    $rgb_array["b"] = $hex_val($color_val, 3, 1);
    $return_string = $rgb_array["r"] . " " . $rgb_array["g"] . " " . $rgb_array["b"];
}
return $return_string;
}
// $hex_str = "FF0000";
// $return_string = "255 0 0";
```

在软件侧，用“状态机”描述电路

对状态机进行分级抽象 (结合编程模式)



提过**Step**切换电路状态:

Step内包含了时序, 波形

通过回调函数处理数据:

通过异步/协程处理数据:

```
def test_rg(self, callback3) -> None:
    # clk 引脚接入时钟源
    self.dut.init_clock("clk")
    self.dut.seed_value = self.CEEN

async def send_req(self, is_push):
    self.port_dict["in_valid"].value = 1
    self.port_dict["in_cmd"].value = self.BusCMD.PUSH.value if is_push else self.BusCMD.POP.value
    self.port_dict["in_data"].value = random.randint(0, 2**8-1)
    await self.dut.AStep(1)

    await self.dut.Acondition(lambda: self.port_dict["in_ready"].value != 1)
    self.port_dict["in_valid"].value = 0

    if is_push:
        self.model.commit_push(self.port_dict["in_data"].value)

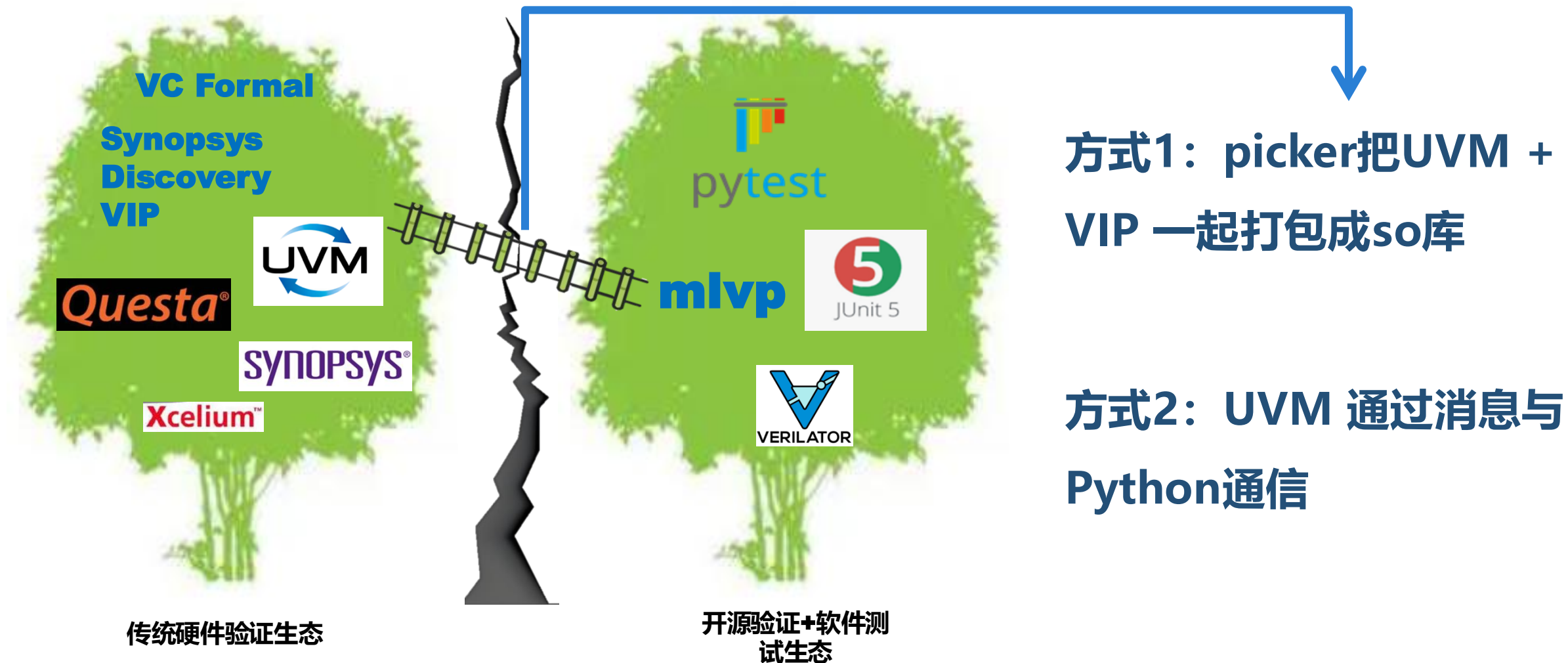
async def receive_resp(self):
    self.port_dict["out_ready"].value = 1
    await self.dut.AStep(1)

    await self.dut.Acondition(lambda: self.port_dict["out_valid"].value != 1)
    self.port_dict["out_ready"].value = 0

    if self.port_dict["out_cmd"].value == self.BusCMD.POP_OKAY.value:
        self.model.commit_pop(self.port_dict["out_data"].value)

    self.greater += 1
else:
    self.less_equal += 1
```

挑战三：如何使用现有生态，如何建设生态



UVM + TLM2 + Python (其他语言)



传统硬件验证生态



资深UVM验证工程师写Driver + 覆盖率 + 基本Assert

时序可见

TLM
↔



初级人员写激励、收集覆盖率

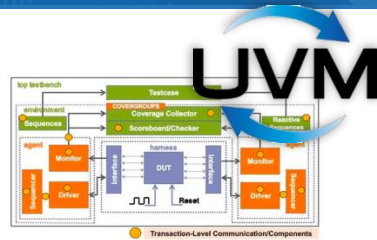
时序不可见



开源验证+软件测试生态

指定Message格式, 自动化生成通信代码与使用例子

基于软件编程语言的验证框架——mlvp



轻量化
软件化



mlvp

使用领域	面向人群	硬件验证工程师	软件&硬件领域开发者
	开源现状	UVM 开源, 但无开源模拟器支持	框架开源, 支持开源/非开源模拟器
易用性	学习门槛	学习门槛较高	轻量化 UVM 概念, 通过简单文档上手
	编程语言	SystemVerilog	软件领域编程语言(Python, Java)
功能特性		<ul style="list-style-type: none"> 定义了验证主流的方法学, 规范高效 UVM 框架实现了众多实用验证功能 	<ul style="list-style-type: none"> 吸收UVM方法学, 拥有相似平台架构 以软件概念函数为核心组织平台, 而非事务 参考模型自动同步、比较 集成软件测试框架, 强大的用例管理, 并支持生成报告

生态建设：提供案例与学习材料



香山微架构开放验证第一期：昆明湖BPU模块UT验证实战

站内搜索...

分支预测器与昆明湖微架构实现

香山分支预测单元 (BPU) 基础设计

何为分支预测

分支预测单元，顾名思义需要完成一件基本的任务——分支预测。在深入探讨分支预测单元之前，必须要了解分支预测是什么。

Tags: 香山 BPU 分支预测

Categories: 香山 BPU 基础设计

为何需要分支预测?

分支预测主要有两方面的原因：一是程序的执行流中含有分支指令，二是高性能处理器使用流水线设计。

程序的执行流中含有分支指令

```
int x = 10;
int y = 20;
int result = 0;

if (x >= y) {
    result = x + y;
} else {
    result = x - y;
}
```

上述是一段C语言代码，这段代码首先定义了三个变量 x, y 和 result，然后根据 x 和 y 值的大小情况对result进行赋值。可以发现，程序在前三行对变量进行赋值时是顺序往下执行的，而在第 5 行时，由于 if 指令的出现，程序产生了分支，从第 5 行直接跳转到了第 8 行继续运行，这就造成了程序执行的分支。

翻译成 RISC-V 汇编之后的代码如下：

```
li a0, 10      # x = 10
li a1, 20      # y = 20
li a2, 0       # result = 0

blt a0, a1, else_branch # 如果 x < y, 则跳转到 else_branch
add a2, a0, a1 # 否则执行 result = x + y
j end         # 跳转到 end
else_branch:
sub a2, a0, a1 # result = x - y
end:
```

可以发现程序依然保持着先前的分支行为，在代码的前三行，指令顺序执行，之后，在程序的第 5 行，出现了一条特殊指令blt，我们称之为分支指令，它会根据 x 和 y 的大小关系决定指令流顺序往下执行还是跳转到其他地方，该指令的出现导致程序的执行出现了分支。

查看页面源码
编辑此页
添加页面
提交文档问题
提交项目问题
整页打印

为何需要分支预测?
程序的执行流中含有分支指令
高性能处理器使用流水线设计
分支预测的可行性
分支预测的基本类型
分支指令的方向预测
分支指令的目标地址预测
预测指令类型
分支预测的一般步骤

Tag Cloud

- BPU 顶层 1
- Docs 1
- Examples 1
- Feature List 6
- FTB 2
- FTB 项 1
- ITTAGE 2
- RAS 2
- TAGE-SC 2
- uFTB 2
- 全篇分支历史 1
- 分支预测历史 1
- 分支预测 2
- 分支预测块 1
- 取值目标队列 (FTQ) 1
- 子预测器 8
- 更新请求 2
- 流水线控制 2
- 重定向请求 2
- 香山 BPU 时序 1
- 香山 BPU 结构 1

Categories

- Feature List 6
- 分支预测单元 (BPU) 2
- 教程 1
- 示例 1
- 香山 BPU 基础设计 5
- 香山 BPU 子模块 5
- 香山 BPU 接口 5

env-xs-ov-00-bpu Public

Edit Pins Unwatch 1 Fork 2 Starred 6

main 1 Branch 0 Tags

Go to file Add file Code

Makiras update picker parameters 2d02a12 · last month 50 Commits

- .github resize 3 months ago
- doc add english translation 2 months ago
- mk update picker parameters last month
- rtl update rtl source code with H-extension 3 months ago
- tests Change the FTQ bundle to new bundle 3 months ago
- utils add random br trace 3 months ago
- .gitignore update readme for uFTB-raw 3 months ago
- LICENSE Update LICENSE 3 months ago
- Makefile add usage & tests 3 months ago
- Readme.md add english translation 2 months ago
- Readme_cn.md add demo link 3 months ago

README MulanPSL-2.0 license

香山微架构开放验证第一期：昆明湖BPU模块UT验证实战

English Readme

本项目的目标是对高性能开源RISC-V香山处理器的微架构进行开放式分包验证的探索。它提供了基于Python的新工具、新方法，让所有对芯片设计与验证感兴趣的同学们能够快速了解香山微架构，学习香山微架构。本期活动对香山昆明湖架构的分支预测模块的原理以及实现进行了详细介绍，并提供了对应的开源验证环境。参本次活动的同学，通过提交Bug、编写验证报告等获取积分与奖励。

开源开放验证官网: open-verify.cc

简介

本项目是基于开源工具对开源芯片进行的开源开放验证。本期验证的对象是香山昆明湖微架构中的BPU模块。

昆明湖微架构

昆明湖架构是香山开源处理器的第三代高性能微架构，架构图请参考：[昆明湖架构图](#)。

About

香山微架构开放验证第一期：昆明湖BPU模块UT测试模块及环境

examples test-environment

Readme

MulanPSL-2.0 license

Activity

Custom properties

6 stars

1 watching

2 forks

Report repository

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Contributors 5

Languages

- SystemVerilog 98.7%
- Other 1.3%

Suggested workflows

Based on your tech stack

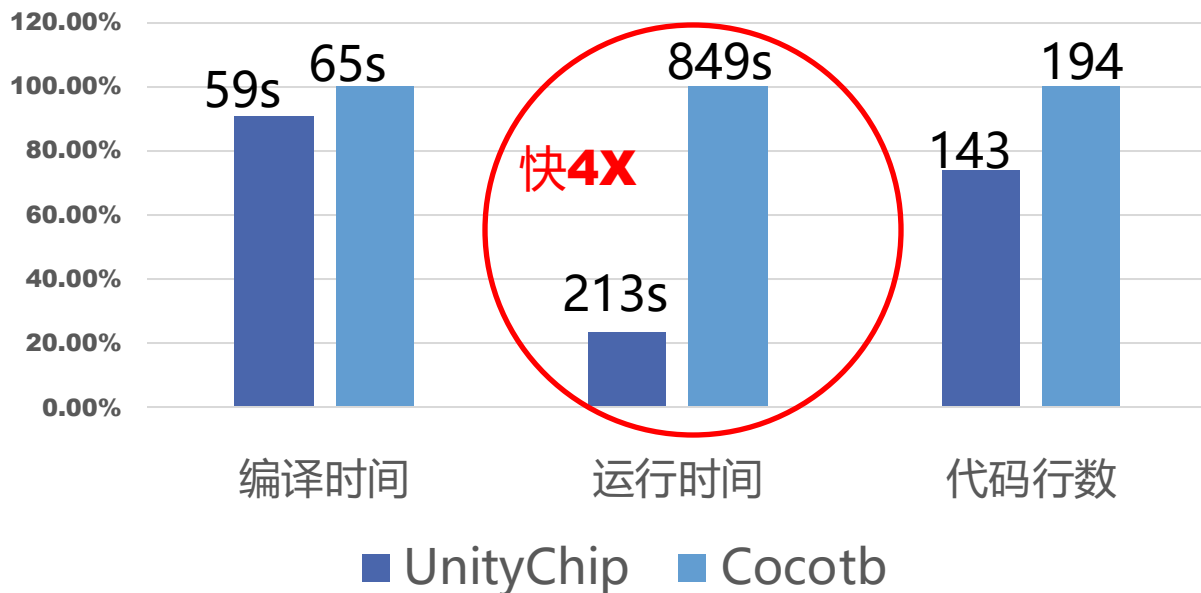
- PyLint Configure
- Python Package using Anaconda Configure

阶段性实践效果

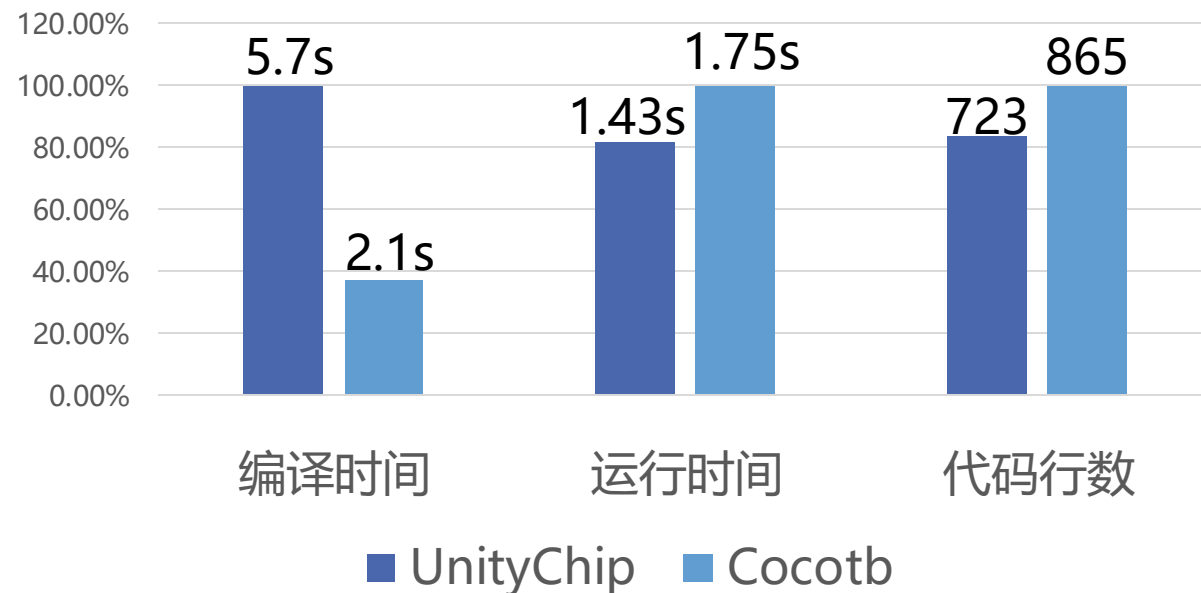
开源众包验证实践效果



TileLink TL-C(部分)



RAS



开源众包验证实践案例 (1)

新生基本情况:

学校: 西北工业大学

专业: 网络信息安全

年级: 大一

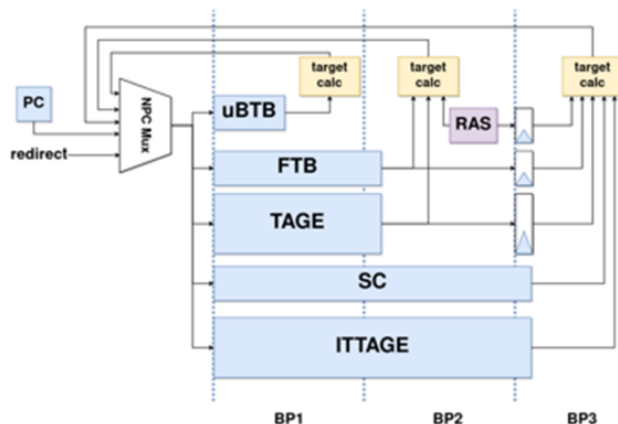
技能: 了解Python

目标: 具有编写RAS验证

日期	进行的项目
7.8	linux环境的配置
7.8-7.9	picker依赖的安装 (学习linux)
7.9-7.10	安装picker并配置picker环境 (进行python)
7.11-7.12	对案例一和案例二进行python代码解读
7.15-7.16	阅读分支预测与RAS文档, 对RAS结构有了
7.17	阅读RAS代码: ras_pins.py, 对基础操作代
7.18-7.19	阅读test_spec_func.py代码, 尝试进行复现
7.22	对比spec代码, 编写了一个简单的测试点
7.22-7.23	阅读test_commit_func.py代码, 尝试进行复
7.23-7.24	对比commit代码, 编写了一个简单的测试
7.24-7.25	阅读redirect代码, 并在一个原有的测试点
7.25-7.26	阅读学习update的case代码, 4种类型的ca
7.26	整理日志, 编写汇报ppt

开源众包验证实践案例 (2)

- 中国科学院大学大三同学3人、安徽大学大四同学1人、齐鲁工业大学大四同学1人
- 目标: 基于开源众包验证工具集完成香山BPU模块验证 (已完成)



- ① 拆分出162个功能点
- ② 编写了69个测试用例 (1718LoC, 平均25)
- ③ 编写了4182行环境代码 (含RM、Trace处理)
- ④ 发现了10个bug
- ⑤ 代码行覆盖率大于99% (目标代码100%覆盖)
- ⑥ 完成64页验证报告撰写

验证对象: 香山昆明湖BPU
(chisel 3076 LoC)

2个月验证成果

开源众包验证实践效果



3 功能介绍与功能点拆分

BPU中，
将每个模块分
下：

- uFTB:
- FTB:
- TAGE-S
- ITTAGE
- RAS:

具体内容详

3.1 uFTB

uFTB维
根据pc从缓
新和饱和计
测结果和更新

3.1.1 缓存

缓存中有
命中。

编号
uFTB 1.1
uFTB 1.2
uFTB 1.3
uFTB 1.4

3.1.2 两比特

每个缓存

编号
uFTB 2.1
uFTB 2.2

3.3.2 TAGE 训练

T0和Tn的表项都包含了饱和计数器。当饱和计数器达到最大值时，增加值计数器会保持最大值，这种情况在本文中称为“上饱和更新”；当饱和计数器达到最小值时，增加值计数器会保持最小值，这种情况在本文中称为“下饱和更新”。

表 3.3.2: TAGE

编号	描述
TAGE-SC 2.1	T0 下饱和更新
TAGE-SC 2.2	T0 上饱和更新
TAGE-SC 2.3	Tn(1 ≤ n ≤ 4) 下饱和更新

编号	描述
TAGE-SC 2.4	Tn(1 ≤ n ≤ 4) 上饱和更新
TAGE-SC 2.5	历史表申请新表项成功
TAGE-SC 2.6	历史表申请新表项失败
TAGE-SC 2.7	主预测错误，在更长历史
TAGE-SC 2.8	主预测错误，在更长历史
TAGE-SC 2.9	useAltOnNaCtrs 寄存器组

3.3.3 SC 预测

SC进行预测的前提条件TAGE启用且TA
否改变TAGE的预测结果，以及totalSum的

表 3.3.3:

编号	描述
TAGE-SC 3.1	预测时 TotalSum 计算正确

LCOV - code coverage report

Coverage	Total	Hit
Lines: 99.7 %	3621	3611
Functions: -	0	0

LCOV - code coverage report

Coverage	Total	Hit
Lines: 100.0 %	401	401
Functions: -	0	0

LCOV - code coverage report

Coverage	Total	Hit
Lines: 100.0 %	703	703
Functions: -	0	0

LCOV - code coverage report

Coverage	Total	Hit
Lines: 100.0 %	1058	1058
Functions: -	0	0

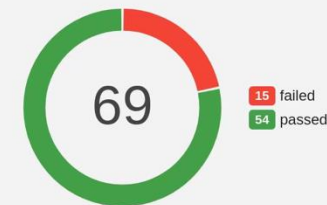
LCOV - code coverage report

Coverage	Total	Hit
Lines: 99.5 %	578	575
Functions: -	0	0

```
Line data Source code
1 // Generated by CIRCT firtool-1.62.0
2 // Standard header to adapt well known macros for register randomization.
3 #ifndef RANDOMIZE
4 #define RANDOMIZE
5 #define RANDOMIZE
6 #endif // RANDOMIZE_MEM_INIT
7 #ifndef RANDOMIZE
8 #define RANDOMIZE
9 #define RANDOMIZE_REG_INIT
10 #define RANDOMIZE
11 #endif // RANDOMIZE_REG_INIT
12 #define RANDOMIZE
13 #endif // not def RANDOMIZE
14 // RANDOM may be set to an expression that produces a 32-bit random unsigned value.
15 #ifndef RANDOM
16 #define RANDOM random
17 #define RANDOM
18 #endif // not def RANDOM
19 // Users can define INIT_RANDOM as general code that gets injected into the
```

XiangShan-BPU UT-Test Report

Started 2024-08-14 15:07:49
Ended 2024-08-14 15:10:13
Duration 0:02:23.785423
Total run time 0:09:54.940490
test_case .



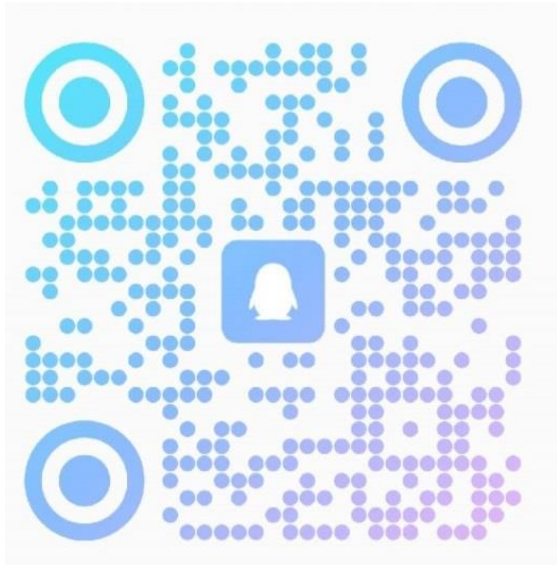
Line Coverage

Coverage Rate	Hint Lines	Total Lines	Detail
94.77%	19656	20741	View Details

Functional Coverage

Coverage Rate	Hint Points	Total Points	Detail
91.36%	148	162	View Details

- > FTB/tests/test_ftb_predict.py 1 0:00:04.617883
- > FTB/tests/test_ftb_update.py 7 0:00:25.316415
- > ITTAGE/tests/test_alloc_train.py 1 0:00:02.602200
- > ITTAGE/tests/test_alt_pred.py 1 0:00:01.316322
- > ITTAGE/tests/test_main_pred.py 1 0:00:01.159620
- > ITTAGE/tests/test_random.py 1 0:00:29.878638
- > ITTAGE/tests/test_reset.py 1 0:00:00.804142
- > ITTAGE/tests/test_saturation_train.py 1 0:00:01.514940
- > ITTAGE/tests/test_src_pred.py 1 0:00:01.230282
- > RAS/tests/test_ras_base_func.py 1 0:00:00.502889
- > RAS/tests/test_ras_commit_func.py 2 3 0:00:11.820139
- > RAS/tests/test_ras_redirect_func.py 3 2 0:00:03.313642
- > RAS/tests/test_ras_spec_func.py 3 2 0:00:03.386082
- > RAS/tests/test_ras_update_pop_1.py 6 0:00:04.024587
- > RAS/tests/test_ras_update_pop_2.py 6 0:00:03.908092
- > RAS/tests/test_ras_update_push_1.py 2 4 0:00:03.959704
- > RAS/tests/test_ras_update_push_2.py 1 5 0:00:03.914478
- > TAGE-SC/tests/test_with_case.py 1 8 0:02:54.068289
- > uFTB/tests/test_with_ftq.py 4 0:05:11.592426
- > uFTB/tests/test_with_raw.py 1 0:00:06.009721



QQ群，欢迎您的加入
可联系管理员申请线下实习



<https://open-verify.cc/>

谢谢!