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

贡献列表

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

中科院计算所

INSTITUTE OF COMPUTING TECHNOLOGY ,CAS

北京开源芯片研究院

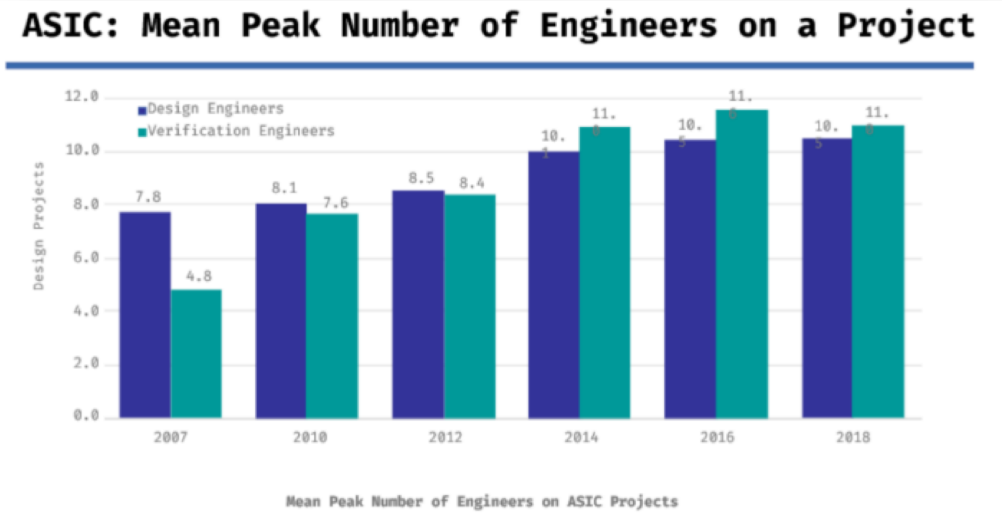
BEIJING INSTITUTE OF OPEN SOURCE CHIP

目录

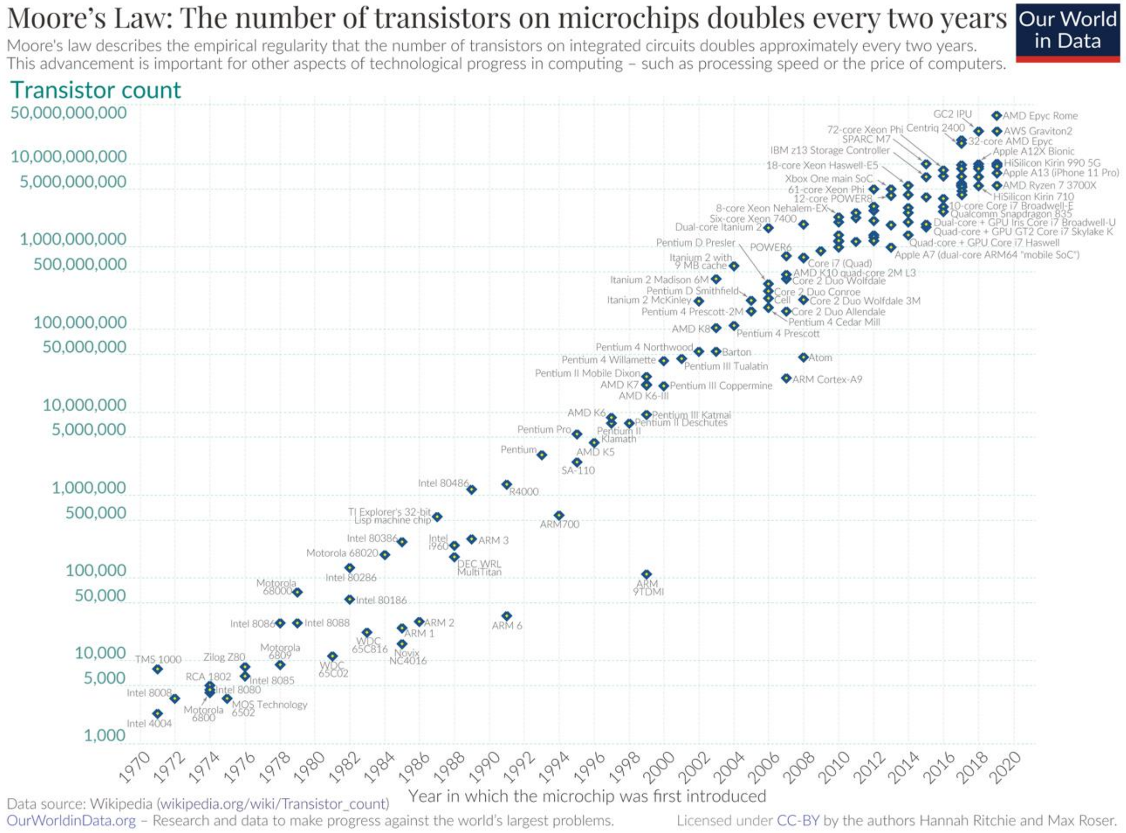
当前芯片验证面临的挑战

传统芯片验证：

- 验证工作量大，占比超过**70%**
- 验证工程师 > 设计工程师
- 源代码**闭源**，属于**商业机密**，只能在公司内完成验证



The 2018 Wilson Research Group Functional Verification Study

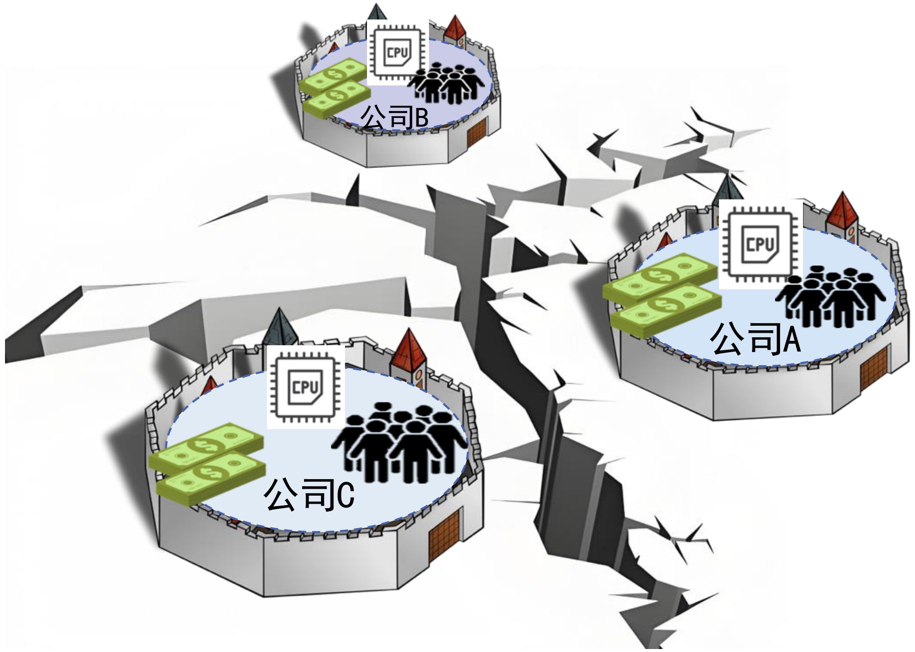


高端芯片越来越复杂，验证投入也越来越大

2024年英伟达GB200 Grace Blackwell有208,000,000,000个晶体管

当前芯片验证面临的挑战

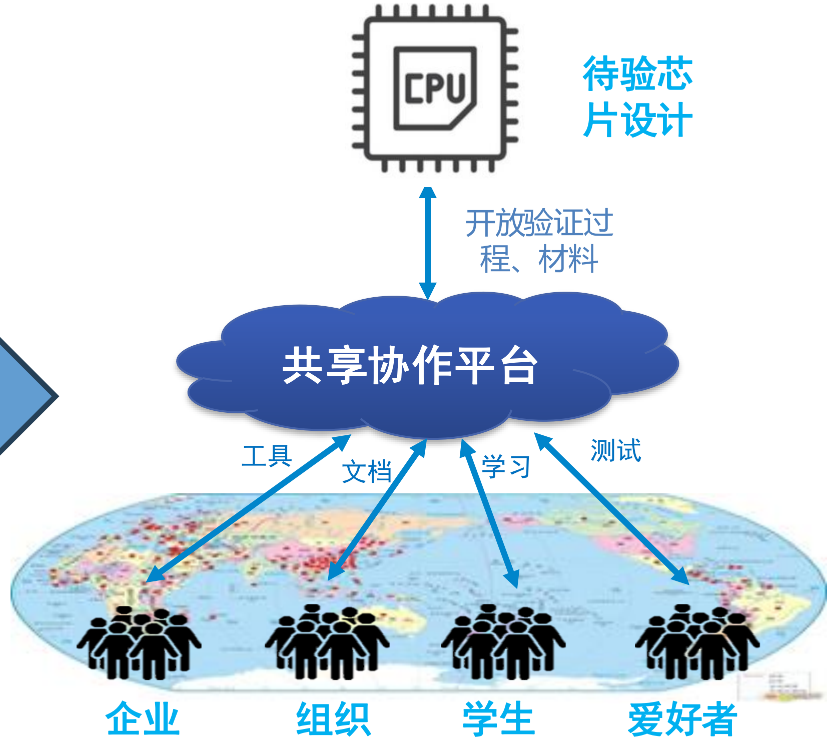
闭源硬件验证



城堡/烟囱模式：各验各的，
人力成本巨大

验证能否开源协作

开源硬件众包验证



协作模式：开源共享

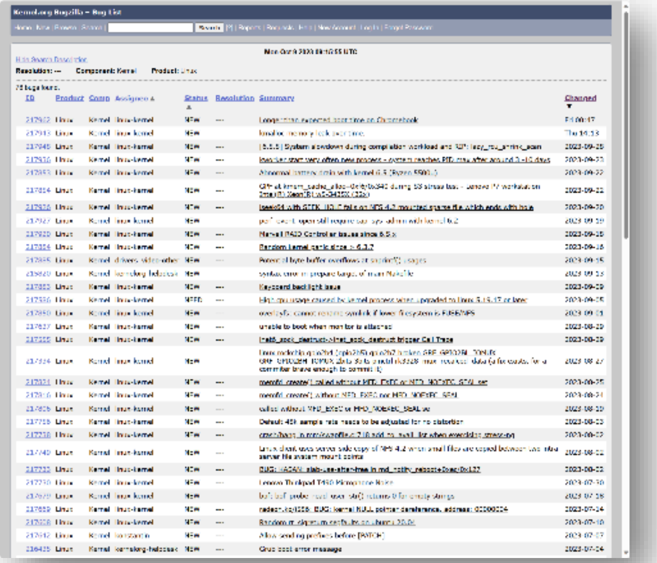
开源硬件众包验证的可行性

- **基础开源芯片验证工具、IP等**已经成熟可用
- 在软件设计、AI数据处理等领域有**成熟众包案例**

现有开源验证工具、验证框架等已基本满足验证需求

VERILATOR
Icarus Verilog
UVM
SystemC
Verible
gem5
XIANGSHAN

Linux kernel (**28M LoC**)已使用开源开放模式的bug管理（基于开源bugzilla平台）

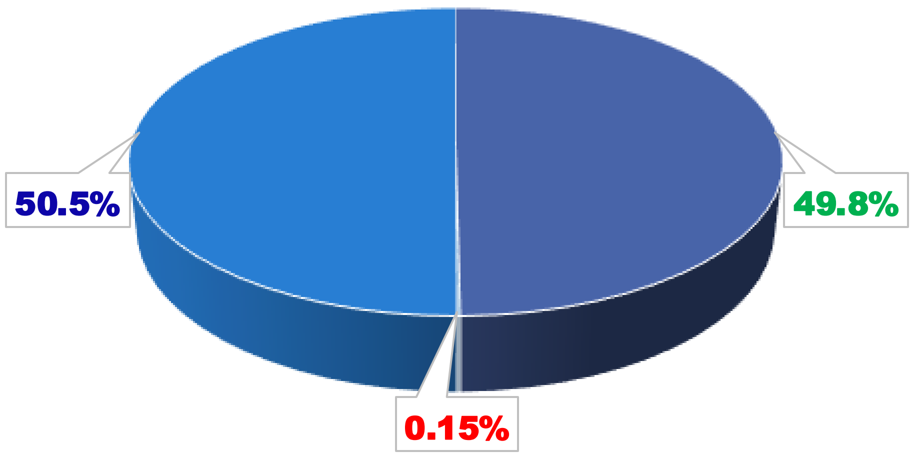


开源的全球众包模式成功有效，ImageNet项目采用该方式收集标注了**1400万**张图片

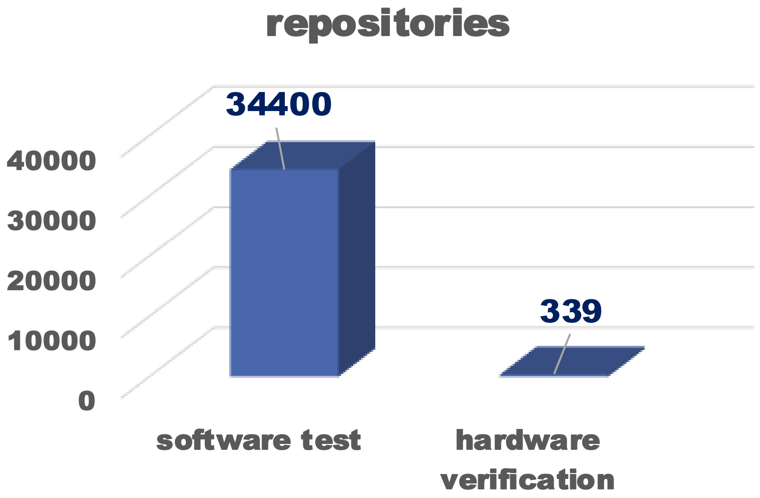
开源硬件众包验证的可行性

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 Verillog/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？

开源硬件众包验证的可行性

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?

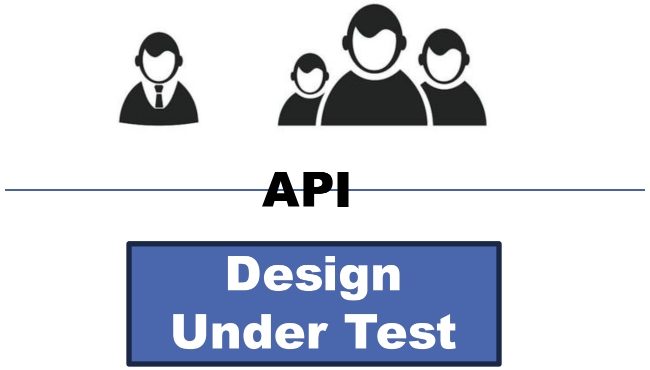
实际上，芯片的大部分验证都是基于软件仿真器进行的，先把“**RTL+**测试代码”一起转换成**C++**，然后编译成电路仿真器进行模拟，**因此芯片验证某种程度上可以看作软件测试**

开源硬件众包验证的可行性

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



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%)	

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

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）

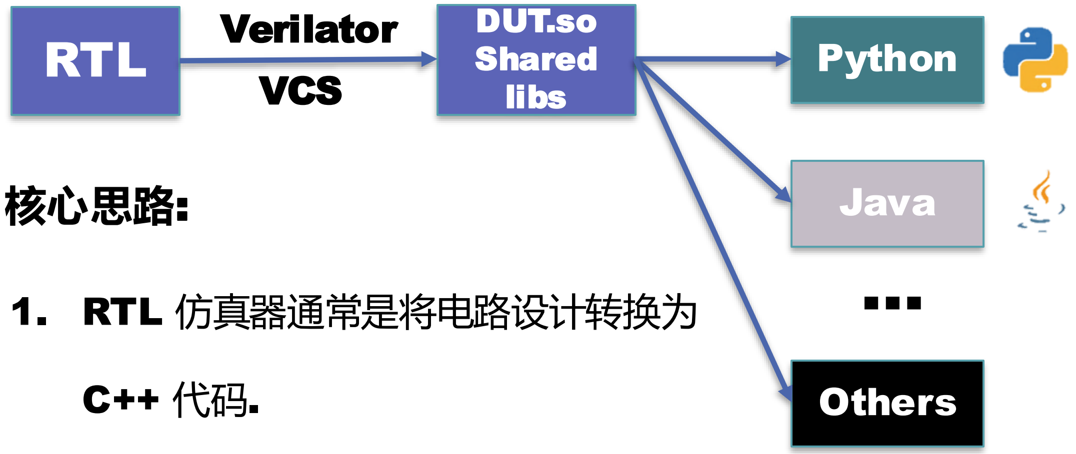
...

每种编程语言都有自己的独有特点、群体与优势

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

```python
import UT_RAS as ras    # import RTL as python module (UT_RAS is generated by picker)
import ras_pins as p    # DUT wrapper functions (rest, pop, push, commit, redirect) based on pins

# pytest automatically recognizes test functions start with "test"
def test_ras_push_compress():
    # The RAS (Return Address Stack) module in the BPU supports identical address compression.
    #   When pushing the same address, the stack pointer TOSR_value does not increase.
    # Test start:
    dut = ras.DUTRAS(waveform_filename="RAS.fst",  # 1. new DUT (RAS module)
                     coverage_filename="RAS.dat")  #    can coustom wave/coverage output name
    dut.init_clock("clock")                         # 2. init clock
    ras_func = p.RASPins(dut)                       # 3. bind dut with wrapper class
    ras_func.reset()                                # 4. reset dut
    meta = None
    for i in range(5):                              # 5. push same data 5 times
        stack_top, meta = ras_func.push(0x100001)
    assert meta["TOSR_value"] == 0, "TOSR_value should be zero" # 6. check result
    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

工具地址： https://github.com/XS-MLVP/picker

多语言芯片验证转换工具picker

```verilog
// 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
```

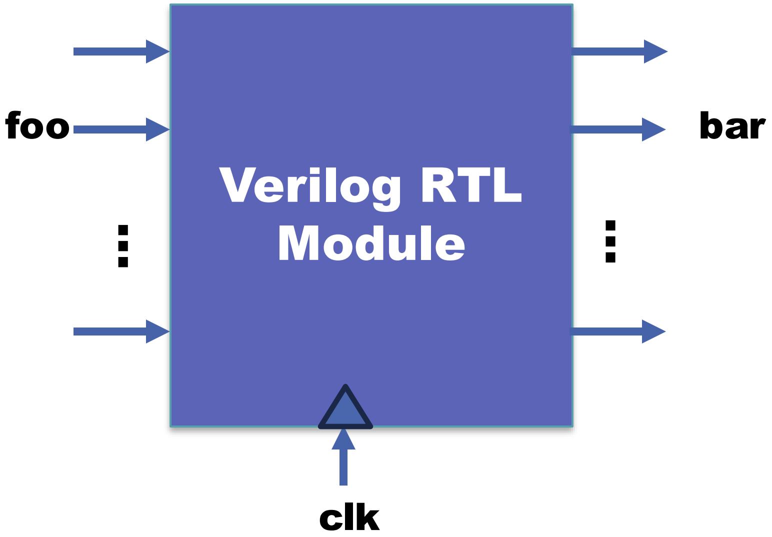
Python模块

```python
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 refecce_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
    #   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 = refecce_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

电路



天然的并行性

时序与流水线

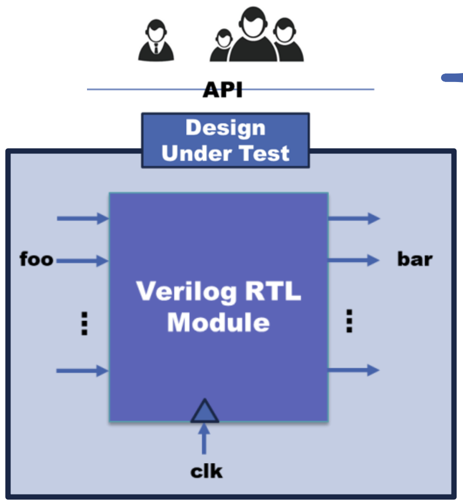
顺序执行

无对应特征

程序

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：

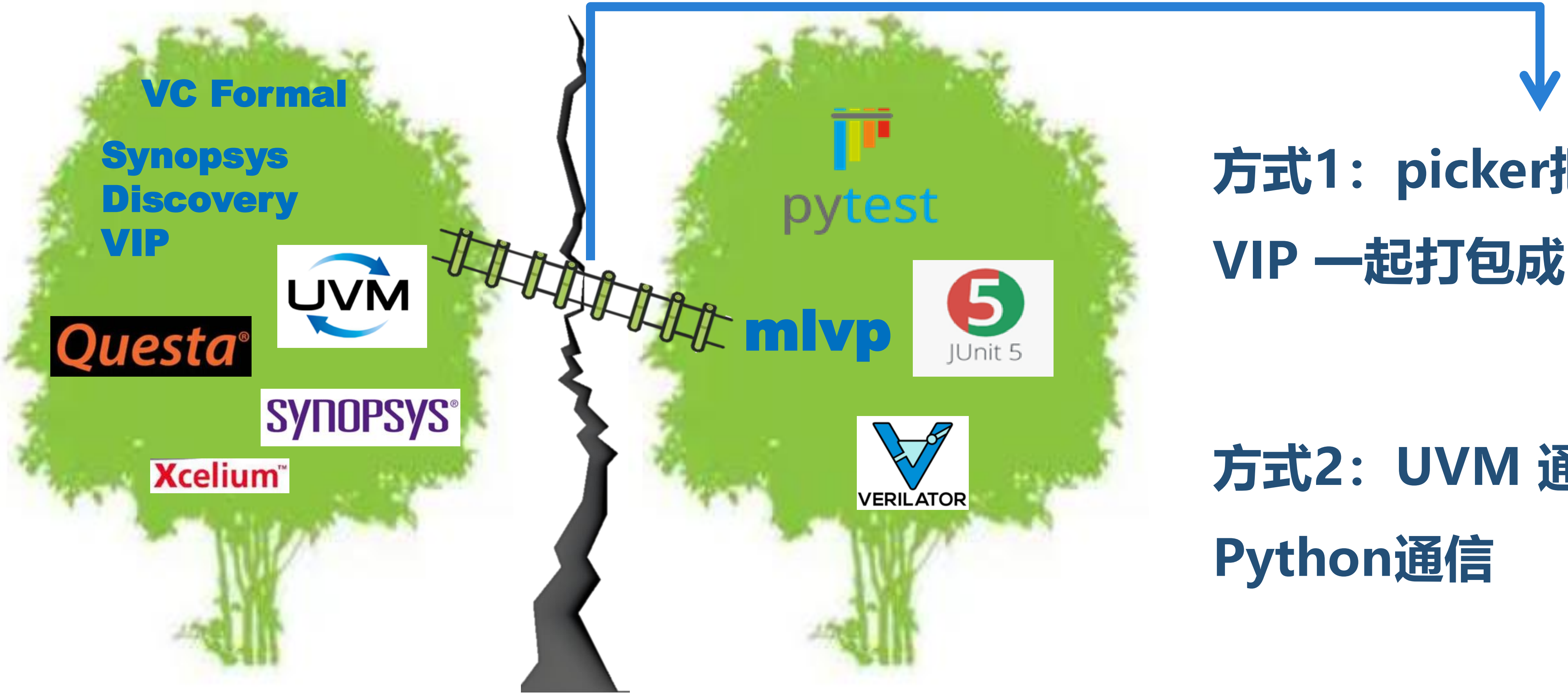
通过异步/协程处理数据：

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

```python
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)
```

```python
            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



传统硬件验证生态

开源验证+软件测试生态

方式1：picker把UVM + VIP 一起打包成so库

方式2：UVM 通过消息与Python通信

UVM + TLM2 + Python （其他语言）

传统硬件验证生态

RTL + UVM + Driver

TLM

System C

Msg

Python

Python module

开源验证+软件测试生态

资深UVM验证工程师写Driver + 覆盖率 + 基本Assert

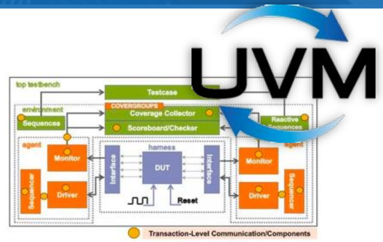
初级人员写激励、收集覆盖率

时序可见

时序不可见

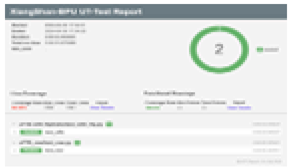
指定**Message**格式，自动化生成通信代码与使用例子

基于软件编程语言的验证框架——mlvp



轻量化

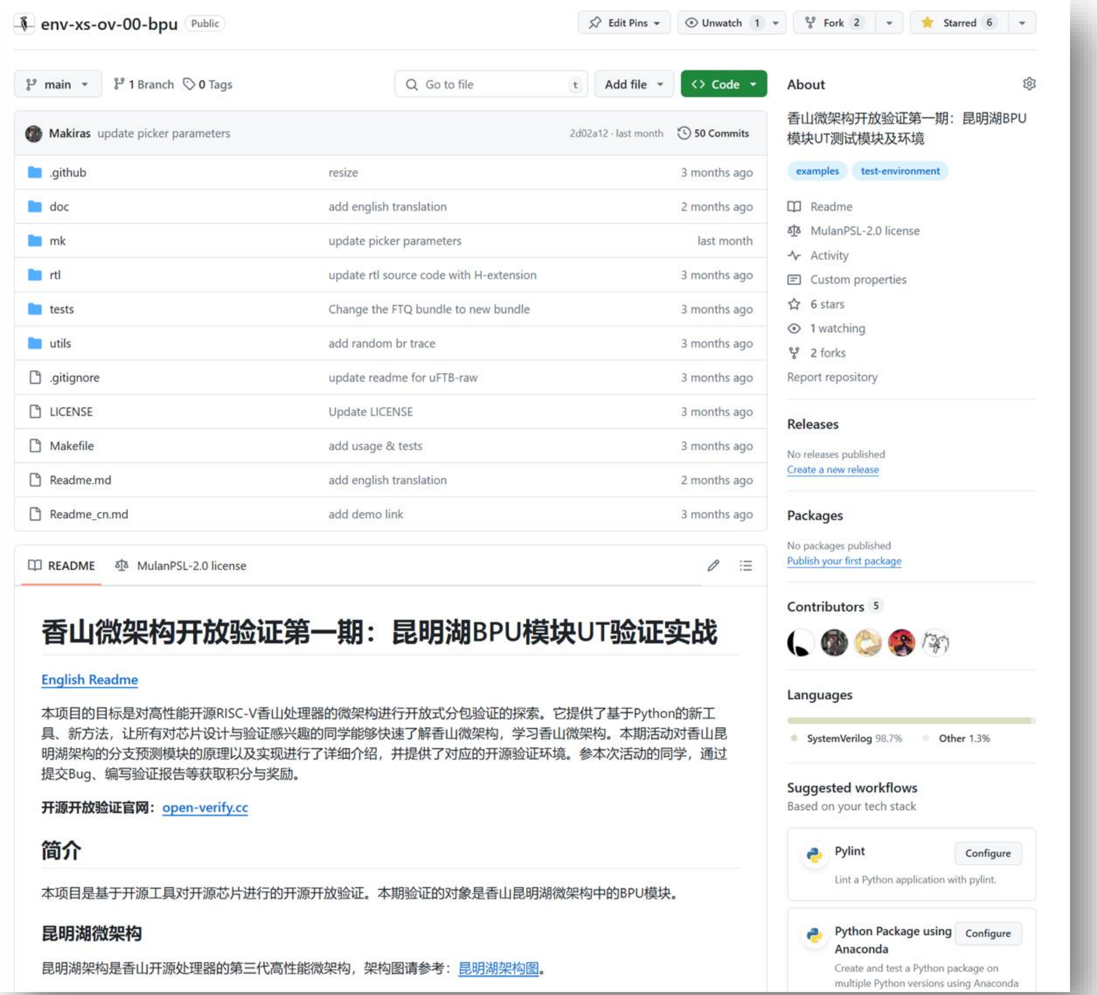
软件化



mlvp

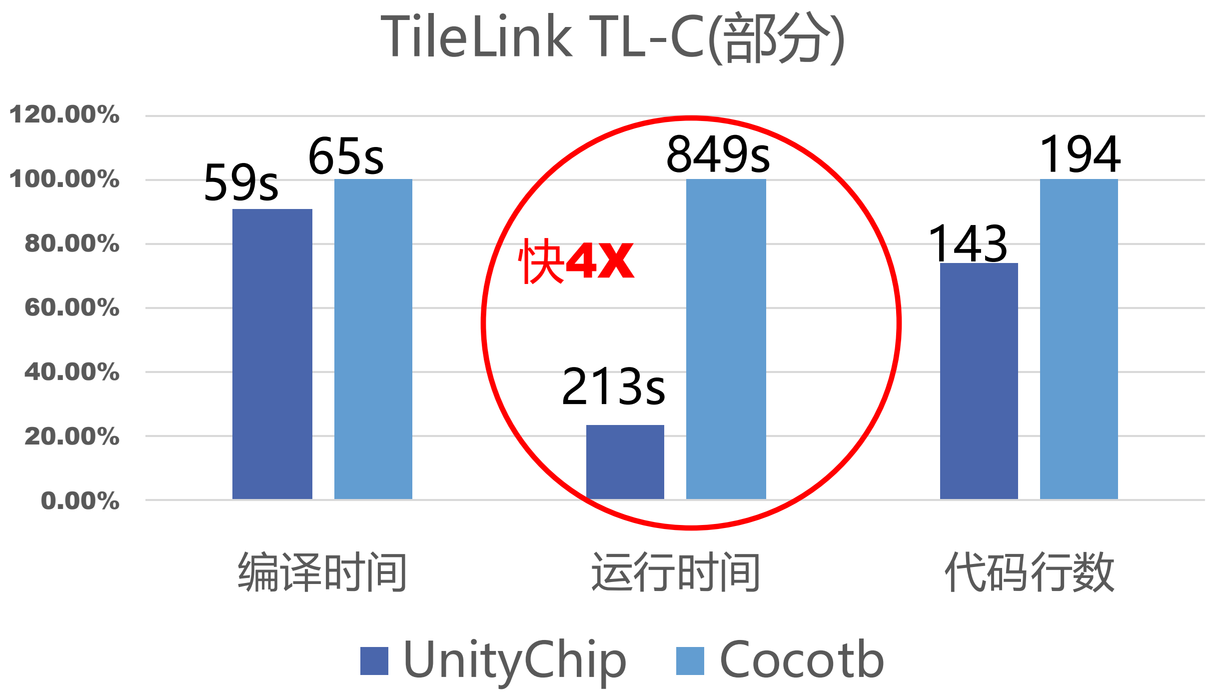
使用领域	面向人群	硬件验证工程师	软件&硬件领域开发者
	开源现状	**UVM**开源，但无开源模拟器支持	框架开源，支持开源/非开源模拟器
易用性	学习门槛	学习门槛较高	轻量化**UVM**概念，通过简单文档上手
	编程语言	**SystemVerilog**	软件领域编程语言(**Python, Java**)
功能特性		• 定义了验证主流的方法学，规范高效 • **UVM** 框架实现了众多实用验证功能	• 吸收**UVM**方法学，拥有相似平台架构 • 以软件概念函数为核心组织平台，而非事务 • 参考模型自动同步、比较 • 集成软件测试框架，强大的用例管理，并支持生成报告

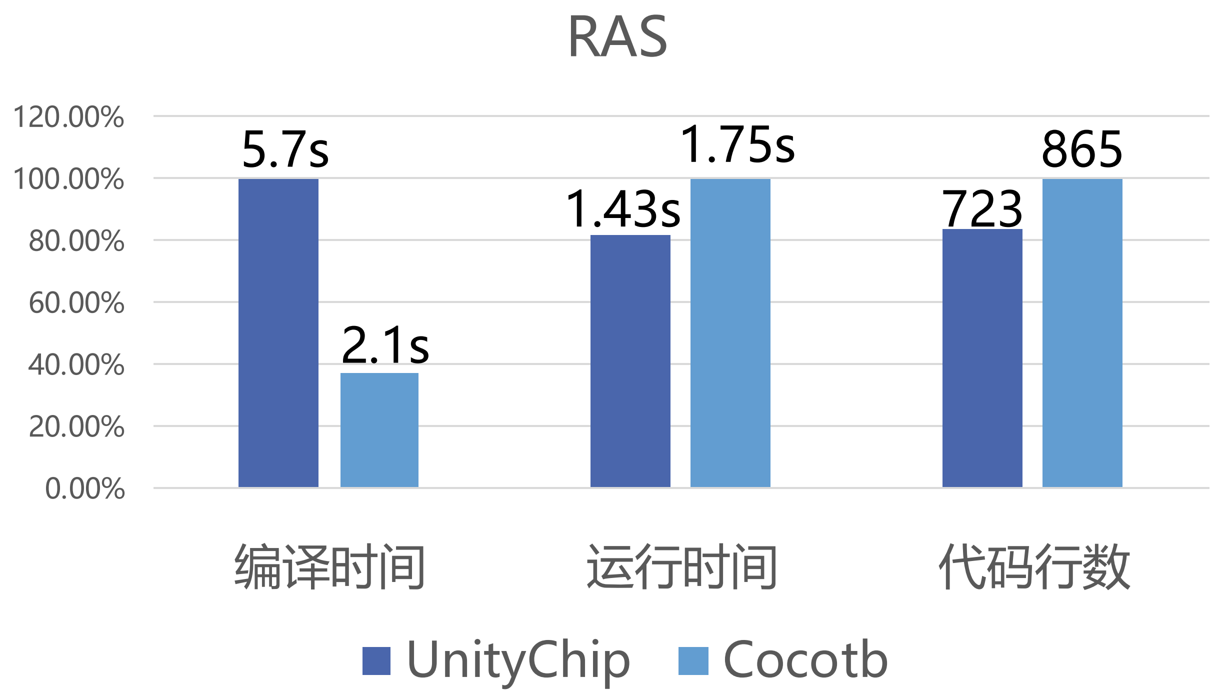
生态建设：提供案例与学习材料



阶段性实践效果

开源众包验证实践效果


TileLink TL-C(部分)
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
59s
65s
快4X
213s
849s
143
194
编译时间
运行时间
代码行数
UnityChip
Cocotb


RAS
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
5.7s
2.1s
1.43s
1.75s
723
865
编译时间
运行时间
代码行数
UnityChip
Cocotb

开源众包验证实践效果

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

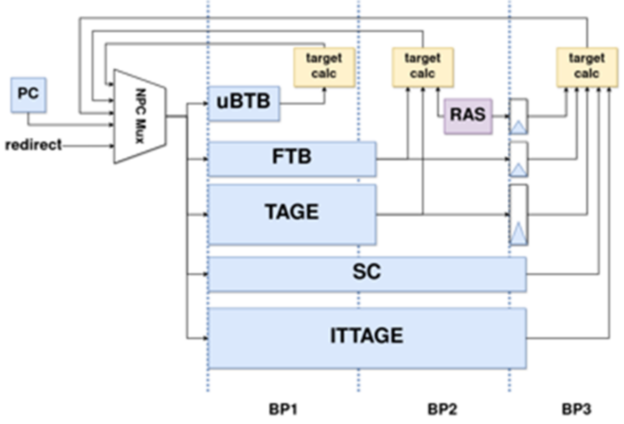
技 能：了解Python

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）**

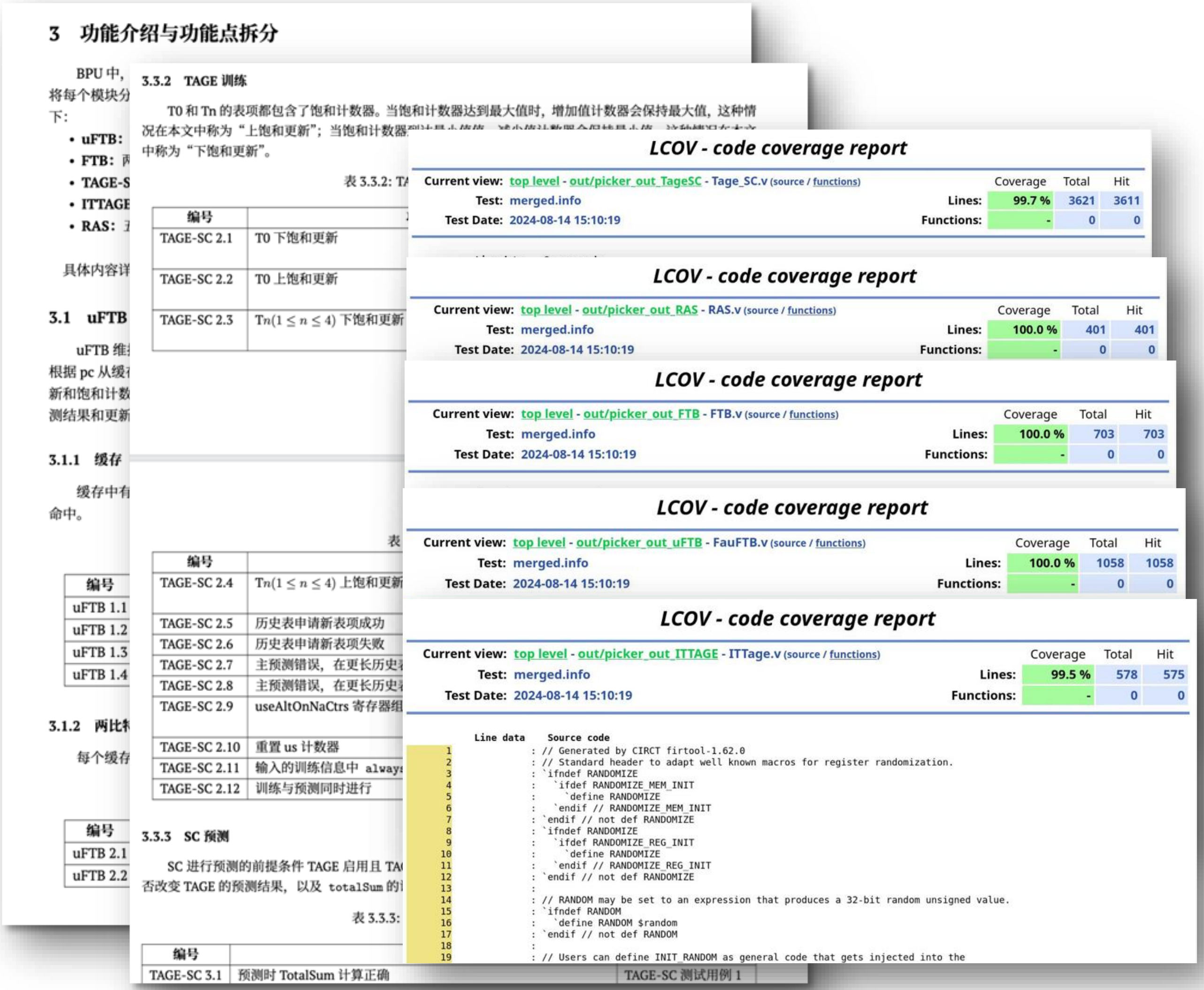


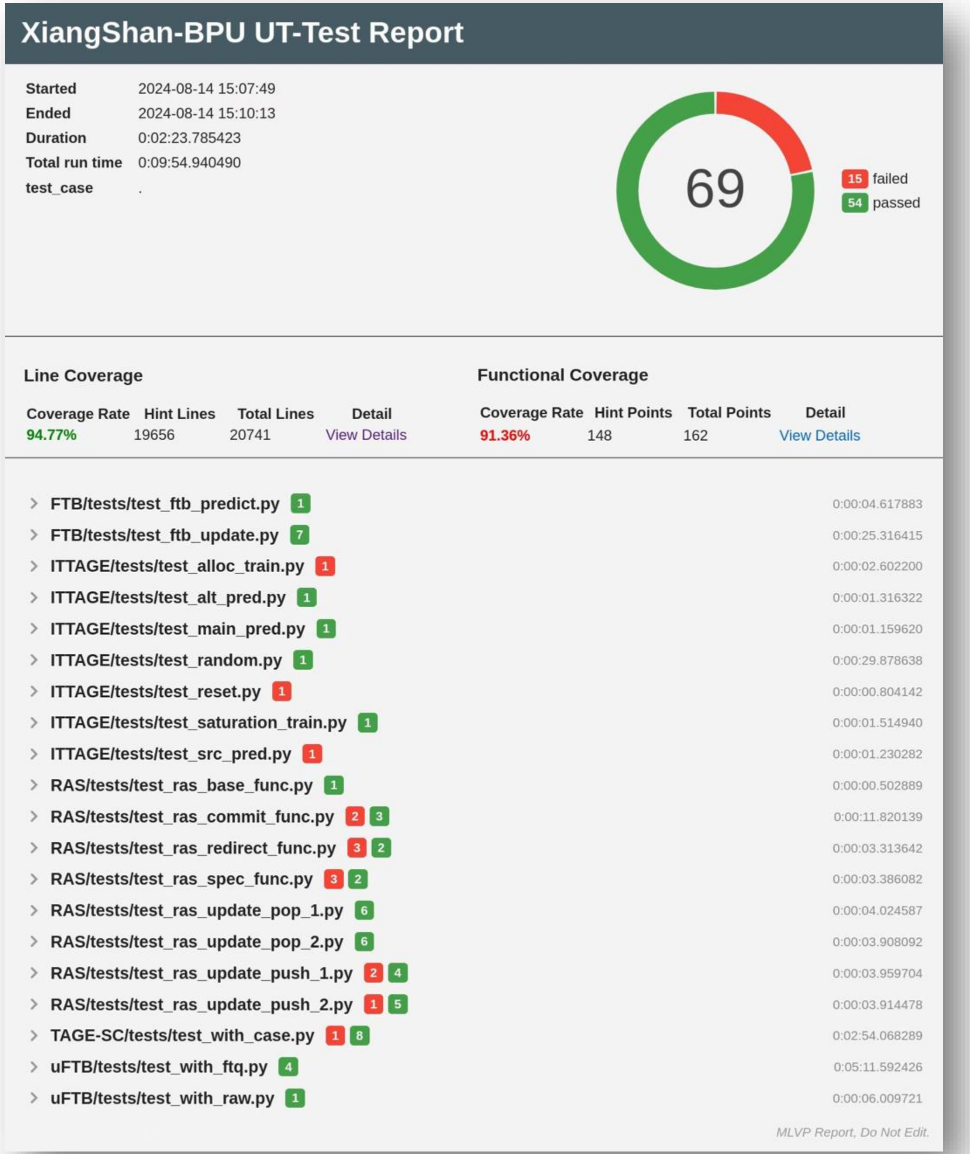
验证对象：香山昆明湖BPU
（chisel 3076 LoC）

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

2个月验证成果

开源众包验证实践效果





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	.

69 — 15 failed, 54 passed

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

Test	Failed	Passed	Time
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

MLVP Report, Do Not Edit.

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！