RISC-V 架构测试初步

PLCT Lab 每周技术分享

熊家辉
浙江工商大学
2024 年 4 月 17 日

RISCOF 初步

第 1 节

简介

第 1 节
简介

第 1 节

简介

第 1.1 小节

riscv-arch-test 简介

第 1 节
简介
第 1.1 小节
riscv-arch-test 简介

项目链接

riscv-non-isa/riscv-arch-test 是 RISC-V 基础架构测试 SIG 工作的存储库。
希望自我证明其实现与 RISC-V 兼容的供应商可以在此
riscv-non-isa/riscv-arch-test-reports 存储库中存档其测试结果。

2024-04-17
RISCOF 初步
简介
riscv-arch-test 简介
项目链接

项目链接
riscv-non-isa/riscv-arch-test 是 RISC-V 基础架构测试 SIG 工作的存储库。
希望自我证明其实现与 RISC-V 兼容的供应商可以在此
riscv-non-isa/riscv-arch-test-reports 存储库中存档其测试结果。

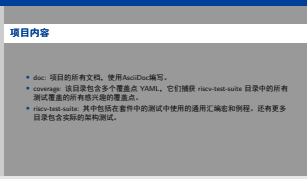
项目目的

RISC-V 架构测试是一组不断发展的测试，旨在帮助确保为给定 RISC-V 配置文件/规范编写的软件能够在符合该配置文件的所有实现上运行。
这些测试还有助于确保实施者正确理解并实施了规范。通过 RISC-V 架构测试并不意味着设计符合 RISC-V 架构。这些只是一组基本的测试，检查规范的重要方面，而不关注细节。
RISC-V 架构测试不能替代严格的设计验证。架构测试向用户提供的结果可以保证规范已被正确解释，并且被测实现 (DUT) 可以被声明为符合 RISC-V 架构测试。

项目目的
RISC-V 架构测试是一组不断发展的测试，旨在帮助确保为给定 RISC-V 配置文件/规范编写的软件能够在符合该配置文件的所有实现上运行。
这些测试还有助于确保实施者正确理解并实施了规范。通过 RISC-V 架构测试并不意味着设计符合 RISC-V 架构。这些只是一组基本的测试，检查规范的重要方面，而不关注细节。
RISC-V 架构测试不能替代严格的设计验证。架构测试向用户提供的结果可以保证规范已被正确解释，并且被测实现 (DUT) 可以被声明为符合 RISC-V 架构测试。

项目内容

- doc: 项目的所有文档，使用AsciiDoc编写。
- coverage: 该目录包含多个覆盖点 YAML，它们捕获 riscv-test-suite 目录中的所有测试覆盖的所有感兴趣的覆盖点。
- riscv-test-suite: 其中包括在套件中的测试中使用的通用汇编宏和例程。还有更多目录包含实际的架构测试。



coverage 文件夹里面包含了很多cgf文件。该文件描述了对于某一个指定的扩展，需要覆盖到的寄存器、指令

第 1 节

简介

第 1.2 小节

RISCOF 简介

第 1 节
简介
第 1.2 小节
RISCOF 简介

RISCOF 简介

RISCOF：RISC-V 兼容性框架是一个基于 Python 的框架，可使用一套 RISC-V 架构组装测试针对标准 RISC-V 黄金参考模型测试 RISC-V 目标（硬或软实现）。

RISCOF 简介
RISCOF：RISC-V 兼容性框架是一个基于 Python 的框架，可使用一套 RISC-V 架构组装测试针对标准 RISC-V 黄金参考模型测试 RISC-V 目标（硬或软实现）。

RISCOF 简介

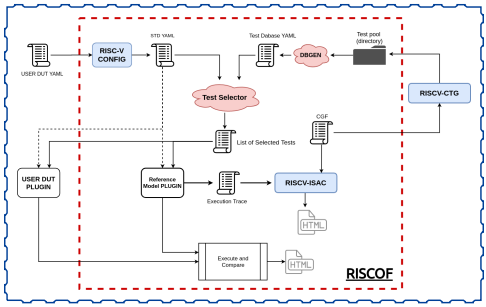
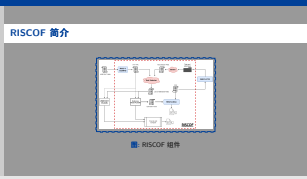


图: RISCOF 组件

2024-04-17
RISCOF 初步
└简介
└RISCOF 简介
└RISCOF 简介



1. 用户做出的 ISA 选择的基于 RISCV-CONFIG 的 YAML 规范。
2. 一个 Python 插件，框架可以使用它来编译测试、模拟测试并提取每个测试的签名。构建模型插件部分提供了定义 python 插件的步骤。
3. RISCV-CONFIG：需要该工具来验证用户提供的输入ISA YAML的合法性。
4. RISCV-ISAC：该工具用于提供覆盖率分析以及对当今 RISCOF 中可用架构套件的质量评估。
5. RISCV-CTG：RISCV-CTG 是基于 RISC-V 的兼容性测试生成器。

RISCOF 简介

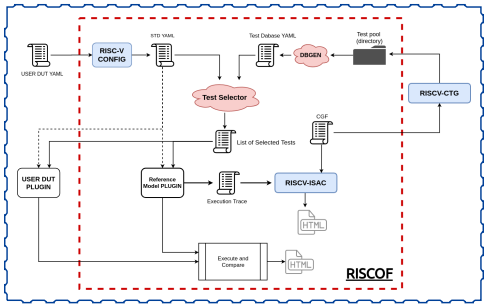
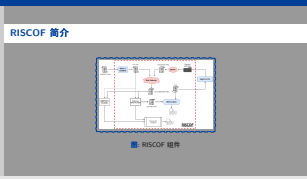


图: RISCOF 组件

2024-04-17

RISCOF 初步
└简介
└RISCOF 简介
└RISCOF 简介



1. DBGEN：RISCOF 在内部维护架构测试套件中所有可用组装测试的数据库。该数据库以 YAML 文件形式维护，用于为给定 DUT 模型选择相关测试。 DBGEN 根据 RISCOF 中可用的测试自动生成此 YAML 文件。该实用程序的目标是供测试贡献者使用其新测试更新数据库。有关其功能和用法的更多信息，请参阅数据库生成器
2. 测试选择器：随着 RISCOF 的发展，RISCOF 将包含大量的架构测试。然而，并非所有测试都适用于 RISC-V 目标的给定配置。因此，测试选择器实用程序使用来自用户的标准化 RISCV-CONFIG 规范和测试数据库来过滤和仅选择那些适用于所考虑的 RISC-V 目标的测试。此列表以 YAML 文件形式呈现，有关此格式的更多信息可在测试列表格式中找到。该实用程序目前是 RISCOF 内部的，不能作为单独的 cli（命令行界面）使用。用户或贡献者都不需要将此实用程序作为单独的模块来处理。

第 2 节

RISCOF 运行

第 2 节
RISCOF 运行

第 2 节

RISCOF 运行

第 2.1 小节

依赖准备

第 2 节
RISCOF 运行
第 2.1 小节
依赖准备

依赖安装

- Python
- RISCV-GNU Toolchain：一个工具链。
- 两个插件

依赖安装
Python
RISCV-GNU Toolchain：一个工具链。
两个插件

构造工具链

编译工具链的详细过程请参阅 在国内网络环境下编译 riscv-gnu-toolchain。选用镜像也不错。

工具链编译

```
1 git clone --recursive https://github.com/riscv/riscv-gnu-toolchain
2 git clone --recursive https://github.com/riscv/riscv-opcodes.git
3 cd riscv-gnu-toolchain
4 ./configure --prefix=/path/to/install --with-arch=rv32gc --with-abi=ilp32d
5 make
```

确认当前 PATH 内包含 `riscv32-unknown-elf-\*`。

插件简介

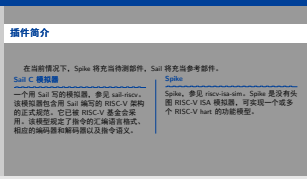
在当前情况下，Spike 将充当待测部件，Sail 将充当参考部件。

Sail C 模拟器

一个用 Sail 写的模拟器，参见 sail-riscv。该模拟器包含用 Sail 编写的 RISC-V 架构的正式规范。它已被 RISC-V 基金会采用。该模型规定了指令的汇编语言格式、相应的编码器和解码器以及指令语义。

Spike

Spike，参见 riscv-isa-sim。Spike 是没有头图 RISC-V ISA 模拟器，可实现一个或多个 RISC-V hart 的功能模型。



构造工具链

编译工具链的详细过程请参阅 在国内网络环境下编译 riscv-gnu-toolchain。选用镜像也不错。

工具链编译

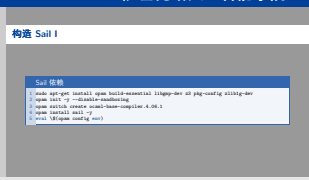
```
1 git clone --recursive https://github.com/riscv/riscv-gnu-toolchain
2 git clone --recursive https://github.com/riscv/riscv-opcodes.git
3 cd riscv-gnu-toolchain
4 ./configure --prefix=/path/to/install --with-arch=rv32gc --with-abi=ilp32d
5 make
```

确认当前 PATH 内包含 `riscv32-unknown-elf-\*`。

构造 Sail I

Sail 依赖

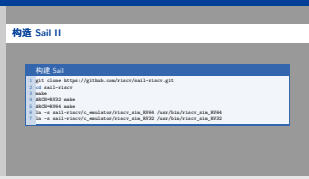
```
1 sudo apt-get install opam build-essential libgmp-dev z3 pkg-config zlib1g-dev
2 opam init -y --disable-sandboxing
3 opam switch create ocaml-base-compiler.4.06.1
4 opam install sail -y
5 eval \$(opam config env)
```

构造 Sail II

构建 Sail

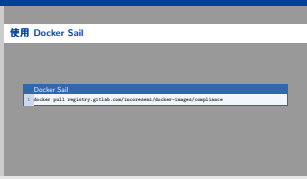
```
1 git clone https://github.com/riscv/sail-riscv.git
2 cd sail-riscv
3 make
4 ARCH=RV32 make
5 ARCH=RV64 make
6 ln -s sail-riscv/c_emulator/riscv_sim_RV64 /usr/bin/riscv_sim_RV64
7 ln -s sail-riscv/c_emulator/riscv_sim_RV32 /usr/bin/riscv_sim_RV32
```

使用 Docker Sail

Docker Sail

```
1 docker pull registry.gitlab.com/incoresemi/docker-images/compliance
```

第 2 节

RISCOF 运行

第 2.2 小节

准备配置

第 2 节
RISCOF 运行
第 2.2 小节
准备配置

插件介绍

RISCOF 提供了多个模板用作基础配置文件。

- Spike
- SAIL_cSim
- InCore Plugins：这是一组插件。
- neorv32 Plugins：第三方插件，供 neorv32 使用。

neorv32 是个微型、可定制和可扩展的 MCU 级 32 位 RISC-V 软核 CPU 和用独立于平台的 VHDL 编写的类似微控制器的 SoC。RV32 [I/E] [M] [A] [C] [B] [U] [X] [Zicsr] [Zicntr] [Zicond] [Zihpm] [Zifencei] [Zfinx] [Zmmul] [Zxcfu] [Smpmp] [Sdext] [Sdtrig]

生成配置文件 I

生成配置文件

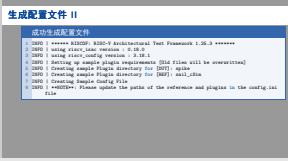
```
1 riscof setup --dutname=spike
```

该命令产生一组配置文件用于控制架构测试。

生成配置文件 II

成功生成配置文件

```
INFO | ****** RISCOF: RISC-V Architectural Test Framework 1.25.3 *******
INFO | using riscv_isac version : 0.18.0
INFO | using riscv_config version : 3.18.1
INFO | Setting up sample plugin requirements [Old files will be overwritten]
INFO | Creating sample Plugin directory for [DUT]: spike
INFO | Creating sample Plugin directory for [REF]: sail_cSim
INFO | Creating Sample Config File
INFO | **NOTE**: Please update the paths of the reference and plugins in the config.ini
    file
```

生成配置文件 III

配置文件为 `config.ini`。

Spike 插件清单

```
spike/ # 待测元件
  env
    link.ld # 待测元件描述文件
    model_test.h # 待测元件头文件
  riscof_spike.py # 插件
  spike_isa.yaml # 待测元件ISA 描述
  spike_platform.yaml # 待测元件平台描述
```


RISCOF 初步
└RISCOF 运行
└准备配置
└生成配置文件
2024-04-17

生成配置文件 IV

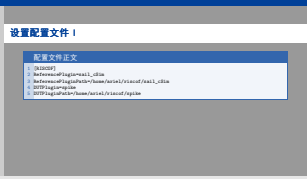
Sail 插件清单

```
sail_cSim/ # 参考模型
  env
    link.ld # 参考元件描述文件
    model_test.h # 待测元件头文件
  __init__.py
  riscof_sail_cSim.py # 插件
```


设置配置文件 I

配置文件正文

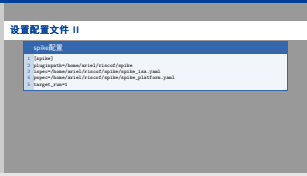
```
1 [RISCOF]
2 ReferencePlugin=sail_cSim
3 ReferencePluginPath=/home/ariel/riscof/sail_cSim
4 DUTPlugin=spike
5 DUTPluginPath=/home/ariel/riscof/spike
```

设置配置文件 II

spike配置

```
[spike]
pluginpath=/home/ariel/riscof/spike
ispec=/home/ariel/riscof/spike/spike_isa.yaml
pspec=/home/ariel/riscof/spike/spike_platform.yaml
target_run=1
```

设置配置文件 III

编译的 Sail 配置

```
[sail_cSim]
pluginpath=/home/ariel/riscof/sail_cSim
PATH=/path/to/install/riscv_sim_RV64
```

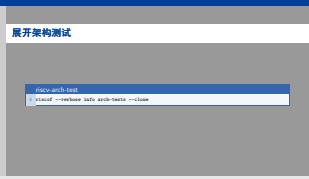
Docker Sail 配置

```
[sail_cSim]
pluginpath=/path/to/riscof/sail_cSim
docker=True
image=registry.gitlab.com/incoresemi/docker-images/compliance
```


展开架构测试

riscv-arch-test

```
1 riscof --verbose info arch-tests --clone
```

第 2 节

RISCOF 运行

第 2.3 小节

运行

第 2 节
RISCOF 运行
第 2.3 小节
运行

运行架构测试 I

第一步是检查输入的 yaml 文件是否配置正确。

检查输入

```
1 riscof validateyaml --config=config.ini
```

YAML 文件验证

```
1 [INFO]    : Reading configuration from: /scratch/git-repo/incoresemi/riscof/config.ini
2 [INFO]    : Preparing Models
3 [INFO]    : Input-ISA file
4 [INFO]    : Loading input file: /scratch/git-repo/incoresemi/riscof/spike/sample_isa.yaml
5 [INFO]    : Load Schema /home/neel/.pyenv/versions/3.7.0/envs/venv/lib/python3.7/site-
      packages/riscv_config/schemas/schema_isa.yaml
6 [INFO]    : Initiating Validation
```


选中测试 I

第二步是生成测试用例表。

生成测试用例表

```
riscof testlist --config=config.ini --suite=riscv-arch-test/riscv-test-suite/ --env=riscv-arch-test/riscv-test-suite/env
```

选中用例

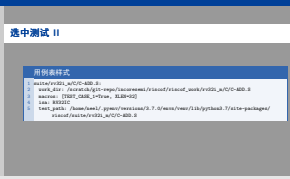
```
[INFO]   : Selecting Tests.
```

选中的测试会列在 `riscof_work/test_list.yaml`。

选中测试 II

用例表样式

```
suite/rv32i_m/C/C-ADD.S:
  work_dir: /scratch/git-repo/incoresemi/riscof/riscof_work/rv32i_m/C/C-ADD.S
  macros: [TEST_CASE_1=True, XLEN=32]
  isa: RV32IC
  test_path: /home/neel/.pyenv/versions/3.7.0/envs/venv/lib/python3.7/site-packages/
      riscof/suite/rv32i_m/C/C-ADD.S
```

运行测试 I

第三步是运行测试。

工具链编译

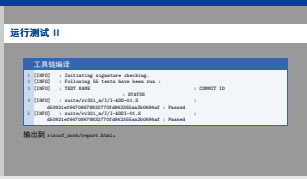
```
1 riscof run --config=config.ini --suite=riscv-arch-test/riscv-test-suite/ --env=riscv-arch
      -test/riscv-test-suite/env
```


运行测试 II

工具链编译

```
1 [INFO]   : Initiating signature checking.
2 [INFO]   : Following 55 tests have been run :
3 [INFO]   : TEST NAME                                    : COMMIT ID
                                  : STATUS
4 [INFO]   : suite/rv32i_m/I/I-ADD-01.S                   :
     d50921ef64708678832770fd842355aa2b0684af : Passed
5 [INFO]   : suite/rv32i_m/I/I-ADDI-01.S                  :
     d50921ef64708678832770fd842355aa2b0684af : Passed
```

输出到 `riscof_work/report.html`。



覆盖率检查 I

这是可选步骤。该步骤生成了覆盖率报告。

生成覆盖率报告

```
1 riscof --verbose debug coverage --suite /path/to/suite --env /path/to/suite
```

工具链编译

```
1 INFO | [--root--]: Test report generated at /home/neel/temp/riscof_work/coverage.html.
```

输出到 `riscof_work/coverage.html`。

2024-04-17
RISCOF 初步
└RISCOF 运行
└运行
└覆盖率检查

第 3 节

RISCV-CTG

第 3 节
RISCV-CTG

第 3 节

RISCV-CTG

第 3.1 小节

RISCV-CTG 简介

第 3 节
RISCV-CTG
第 3.1 小节
RISCV-CTG 简介

项目链接

riscv-software-src/riscv-ctg 是基于 RISC-V 的兼容性测试生成器。该工具用于生成官方 RISC-V 架构测试套件和 RISC-V 架构测试框架 RISCOF 中使用的测试。 CTG 生成的所有测试均符合官方测试格式规范。
CTG 类似于约束测试生成器，能够生成针对特定约束集的测试。这些约束使用覆盖组格式 (CGF) 文件提供给 CTG。 CGF 文件包含不同指令的各种覆盖点。 CTG 将每个覆盖点视为约束，并使用求解器来识别潜在的解决方案。 CTG 使用 python-constraint 包提供的约束满足问题 (CSP) 求解器。

项目链接
riscv-software-src/riscv-ctg 是基于 RISC-V 的兼容性测试生成器。该工具用于生成官方 RISC-V 架构测试套件和 RISC-V 架构测试框架 RISCOF 中使用的测试。 CTG 生成的所有测试均符合官方测试格式规范。
CTG 类似于约束测试生成器，能够生成针对特定约束集的测试。这些约束使用覆盖组格式 (CGF) 文件提供给 CTG。 CGF 文件包含不同指令的各种覆盖点。 CTG 将每个覆盖点视为约束，并使用求解器来识别潜在的解决方案。 CTG 使用 python-constraint 包提供的约束满足问题 (CSP) 求解器。

项目目的

CTG 的目标用户/受众是验证和设计工程师，他们希望创建一套专注于覆盖特定极端情况的测试。然后可以使用这些测试来展示指令本身的功能。
请注意，CTG 的功能完全受到 CGF 中覆盖点的限制。可以向 CTG 提供涵盖指令所有极端情况的更精细的 CGF，以便为该指令创建近似验证测试。

2024-04-17
RISCOF 初步
RISCV-CTG
RISCV-CTG 简介
项目目的

项目目的
CTG 的目标用户/受众是验证和设计工程师，他们希望创建一套专注于覆盖特定极端情况的测试。然后可以使用这些测试来展示指令本身的功能。
请注意，CTG 的功能完全受到 CGF 中覆盖点的限制。可以向 CTG 提供涵盖指令所有极端情况的更精细的 CGF，以便为该指令创建近似验证测试。

RISCV-CTG 组件

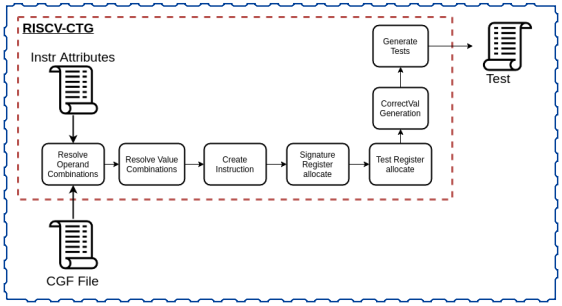
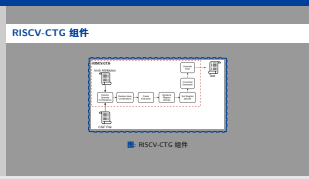


图: RISCV-CTG 组件



1. Instr Attributes: 为了生成给定指令的测试，需要事先知道指令的属性。此信息以 YAML 格式存储在 CTG 中，称为属性文件。
2. Op and Val Combinations:CTG 的前两个阶段接收输入 CGF 文件并识别操作数和值组合的解决方案。这些解决方案是相互独立执行的。在这些阶段中，求解器用于寻找满足相应覆盖点的解决方案。此处还分配了立即值。如果启用随机化，则使用随机求解器。
3. Instruction Creation:本阶段将前几个阶段导出的操作数和值组合解在本阶段相互组合，完成待测指令的所有字段。

RISCV-CTG 组件

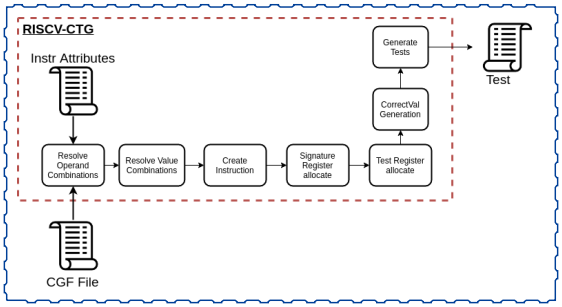
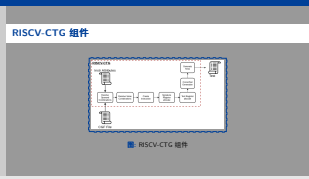


图: RISCV-CTG 组件



1. Signature and Test Register Allocation:还应该为指令的每个实例提供一个签名寄存器来保存操作结果，以及一个附加的测试寄存器来执行备用的特定于目标的检查或调试。寄存器以贪婪方式分配，使得最大数量的指令使用相同的签名和测试寄存器。因此，这导致寄存器之间的指针传输最少。
2. CorrectVal Generation:对于一些算术指令，如 add、sll、sub 等，可以轻松定义属性 YAML 中相应的操作字段来捕获这些指令的行为。在此，CTG 阶段使用这些字段来定义这些操作的预期值/结果。测试中的这些 Correctval 字段可用于执行结果的内联检查和调试失败。(还在开发)
3. Generate Tests:最后，定义所有字段后，使用符合测试格式规范的特定测试模板来生成每条指令的汇编文件。

RISCV-CTG 组件

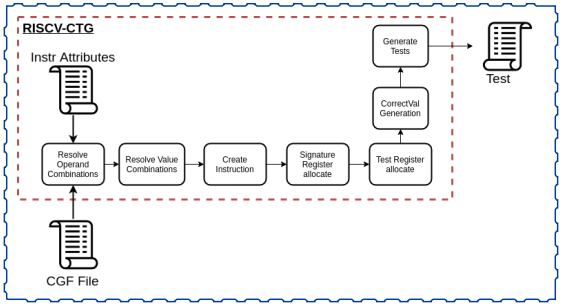
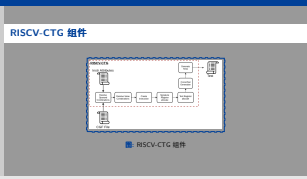


图: RISCV-CTG 组件



由于 CTG 采用 CSP 求解器，涉及大型数据集的某些约束的运行时间可能很快就会增加。CTG 通常还用于生成跨指令的一套测试，而不是单个测试。因此，为了减少套件生成的运行时间，CTG 在内部将 CGF 中的覆盖组分配给单个主机进程，从而并行运行。随机求解器对于提高速度至关重要。由于 CGF 中定义松散的约束（例如 rs1_val 0 和 rs2_val 0），当数据集很大时，求解器可能会花费相当长的时间来找到所有解决方案。另一方面，随机求解器提供第一个满足问题的解决方案，然后退出，从而节省时间。

对松散定义的约束使用随机值的第二个好处是，它增加了在同一指令中覆盖多个覆盖点（稍后可能会介绍）的机会。因此，这减少了覆盖 CGF 覆盖组中提到的所有覆盖点所需的指令数量。

必须注意的是，测试的覆盖范围绝不会因为使用随机求解器而受到影响。

使用随机求解器的一个明显缺点是无法再次重现相同的测试。然而，CTG 用户的典型意图是生成满足 CGF 中定义的覆盖点的测试。由于这是由 CTG 保证的，所以可重复性问题可以放在一边。

Instr Attributes

CTG 属性文件

```
add:
  xlen: [32,64]
  isa: I
  operation: 'hex((rs1_val + rs2_val) & (2**(xlen)-1))'
  formattype: 'rformat'
  rs1_op_data: *all_regs
  rs2_op_data: *all_regs
  rd_op_data: *all_regs
  rs1_val_data: 'gen_sign_dataset(xlen) + gen_sp_dataset(xlen,True)'
  rs2_val_data: 'gen_sign_dataset(xlen) + gen_sp_dataset(xlen,True)'
  template: TEST_RR_OP(\$inst, \$rd, \$rs1, \$rs2, \$correctval, \$rs1_val, \$rs2_val, \$
      swreg, \$offset, \$testreg)
```


// opcode: $inst ; op1:$rs1; op2:$rs2; dest:$rd; op1val:$rs1_val; op2val:$rs2_val

1. name:指令名称
2. xlen: 该指令适用的 XLEN 值列表
3. isa: 该指令属性所属的RISC-V ISA扩展
4. operation：一个Python计算的字符串，它定义了指令的功能
5. formattype: 指示指令格式类型的字符串
6. rs1_op_data: 可用作操作数 1 的合法寄存器列表
7. rs2_op_data: 可用作操作数 2 的合法寄存器列表
8. rd_op_data: 可用作目标的合法寄存器列表
9. rs1_val_data: 可用作操作数 1 的值的整数列表
10. rs2_val_data: 可用作操作数 2 的值的整数列表
11. template: 一个字符串，指示用于创建测试的汇编宏。

第 4 节

RISCV-Config

第 4 节
RISCV-Config

第 4 节

RISCV-Config

第 4.1 小节

RISCV-Config 简介

项目链接

riscv/riscv-config RISCV-Config 是用于生成标准化描述文件的工具。

项目链接
riscv/riscv-config RISCV-Config 是用于生成标准化描述文件的工具。

项目目的

该工具用于确定唯一的描述文件，用于描述目标 ISA 和用户实现的平台特定功能。

2024-04-17
RISCOF 初步
RISCV-Config
RISCV-Config 简介
项目目的

项目目的
该工具用于确定唯一的描述文件，用于描述目标 ISA 和用户实现的平台特定功能。

RISCV-Config 组件

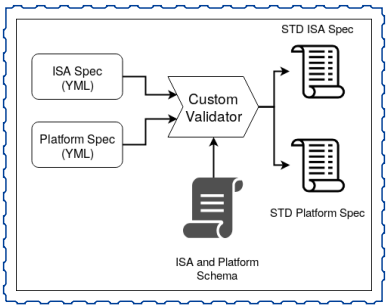


图: RISCV-Config 组件

验证器首先检查 ISA 和平台规范是否存在不一致之处。像“F”("F"即单精度浮点（single-precision floating-point)这样的检查是否存在“D”(双精度浮点指令（依赖于基本的单精度浮点指令子集F）)是由验证者执行的。D扩展依赖F扩展。如果为规范提供了任何非法配置，验证器将退出并显示错误。一旦验证器检查通过，就会生成两个单独的标准 yaml 文件，每个文件对应一种输入类型。这些标准 yaml 文件包含每个节点的所有详细字段和附加信息。虽然用户不需要指定输入 yaml 文件中的所有字段，但验证器将为这些字段分配默认值，并为 ISA 和平台规范生成标准的详尽 yaml。

1. ISA 规范：此 YAML 文件旨在捕获用户实现的 ISA 相关功能。该输入文件的详细信息可以在这里找到：ISA YAML Spec。
2. 平台规范：此 YAML 文件旨在捕获用户实现的平台特定功能。此输入文件的详细信息可以在此处找到：Platform YAML Spec。

第 5 节

RISC-V ISA Coverage

第 5 节
RISC-V ISA Coverage

第 5 节

RISC-V ISA Coverage

第 5.1 小节

RISC-V ISAC 简介

第 5 节
RISC-V ISA Coverage
第 5.1 小节
RISC-V ISAC 简介

项目链接

riscv/riscv-isac 用于提供覆盖率分析以及对当今 RISCOF 中可用架构套件的质量评估。

项目链接
riscv/riscv-isac 用于提供覆盖率分析以及对当今 RISCOF 中可用架构套件的质量评估。

项目目的

RISC-V ISAC 是一种 ISA 覆盖率提取工具。给定一组覆盖点以及在模型上运行的测试/应用程序的执行跟踪，ISAC 可以提供一份报告，详细指示测试/应用程序覆盖了哪些覆盖点。 ISAC 还能够对测试/应用程序中发生的数据传播提供详细的质量分析。

2024-04-17
RISCOF 初步
RISC-V ISA Coverage
RISC-V ISAC 简介
项目目的

项目目的
RISC-V ISAC 是一种 ISA 覆盖率提取工具。给定一组覆盖点以及在模型上运行的测试/应用程序的执行跟踪，ISAC 可以提供一份报告，详细指示测试/应用程序覆盖了哪些覆盖点。 ISAC 还能够对测试/应用程序中发生的数据传播提供详细的质量分析。

RISC-V ISAC 主组件

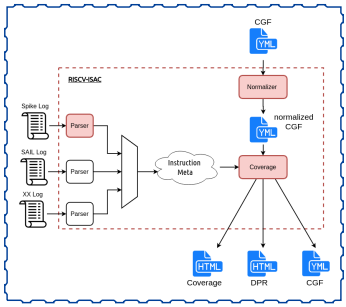
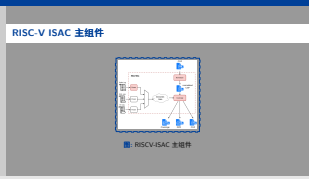


图: RISCV-ISAC 主组件



1. Cover Group Format：感兴趣的覆盖点以直观的 YAML 格式捕获。CGF 文件通常由单个数据集节点和多个覆盖组组成。每个覆盖组可以为不同的指令集定义多个覆盖点。目前仅支持操作数寄存器和操作数值的叉积。
2. Execution Trace Format:RISCV-ISAC 需要在 RISC-V 目标（ISS 或 RTL）上运行的测试/应用程序的执行跟踪作为输入。 RISCV-ISAC 使用此跟踪来分析所覆盖的覆盖点。RISCV-ISAC 中要使用或支持的执行跟踪需要满足以下条件：模型提交/执行的每条指令（可以是 ISS 或 RTL）都应按照提交的顺序捕获为日志中的条目。每个指令条目必须包含指向该指令开头的程序计数器; 每个指令条目必须包含已提交/执行的指令的十六进制编码。每个指令条目还必须包括由于该指令的执行而发生的任何架构状态更新。例如，更新的目标寄存器、修改的 csr、写入的内存区域等。每个指令条目可以跨越多行,每条指令的信息必须可以通过正则表达式检索。指令的助记符是可能的，也应该提供。

RISC-V ISAC 主组件

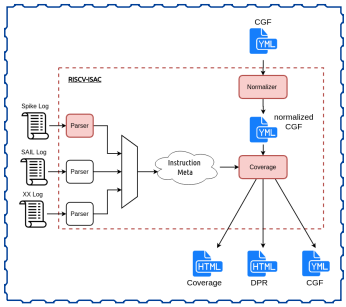
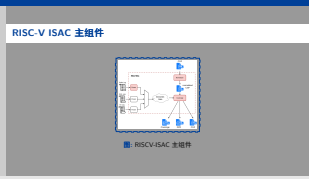


图: RISCV-ISAC 主组件



1. Parser-Module: 解析器模块旨在解析执行跟踪以提取基于每条指令提交的信息。给定遵循上述格式的执行跟踪，解析器模块能够推导信息并将其提取到公共指令类对象中。然后将该对象传递到覆盖模块以进行覆盖点分析。随着解析器模块与覆盖模块解耦，对解析不同执行跟踪格式的支持可以轻松集成到 RISCV-ISAC 中。目前支持以下 RISC-V 模型的执行跟踪SAIL;SPIKE.
2. Normalizer-Module: 输入 CGF 文件中定义的覆盖点可能包含抽象函数.规范化模块负责将这些抽象函数展开到各个覆盖点，并生成覆盖模块使用的规范化 CGF 文件。

RISC-V ISAC 主组件

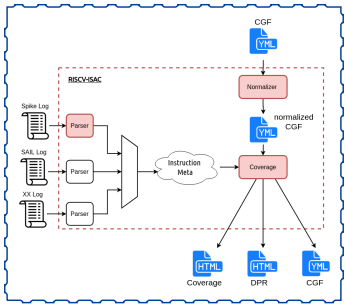
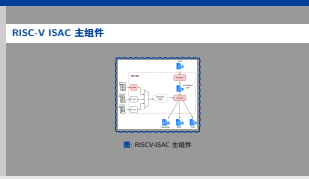


图: RISCV-ISAC 主组件



1. Coverage Module: 覆盖率模块负责执行覆盖率分析并生成 YAML 和 HTML 报告。覆盖模块维护 RISC-V 的简单架构状态，如 PC、整数寄存器文件等。该状态根据执行跟踪中遇到的指令进行更新。每次解析器呈现指令类对象时，覆盖模块都会检查规范化的 CGF 文件是否有任何覆盖点被该特定指令命中。一条指令可以命中多个覆盖点。覆盖模块还允许将覆盖范围限制到测试的特定区域。这些区域由用户根据测试中可用的标签指定。覆盖模块使用这些标签和 elf 来推断确切的地址范围。覆盖模块可以通过 –cov-label (-l) 参数进一步限制为仅收集 CGF 文件中指定的某些覆盖组的覆盖范围。覆盖模块还可以提供数据传播报告，该报告捕获覆盖点是如何将有助于它们的指令存储在存储区域中的。这在创建基于签名的测试时特别有用。然而，此功能需要指定内存区域的标签，类似于使用 –sig-label 参数指定测试区域的方式。覆盖模块在执行结束时生成 4 个工件。第一个是更新的 CGF 文件，其中添加了每个覆盖点的频率。当合并不同运行中的多个覆盖点时，此文件非常有用。基于 YAML 的报告，捕获详细的高级覆盖点比率。基于 HTML 的报告捕获与 YAML 相同的信息;Markdown 格式的数据传播报告。仅当签名/数据区域被指定为参数时才可用。

RISC-V ISAC 主组件

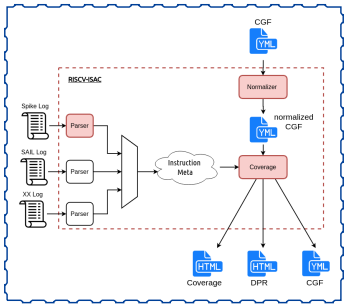
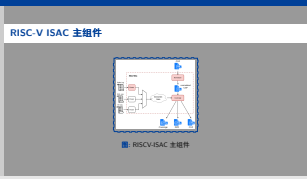


图: RISCV-ISAC 主组件

2024-04-17
RISCOF 初步
└RISC-V ISA Coverage
└RISC-V ISAC 简介
└RISC-V ISAC 主组件



RISC-V ISAC 合并组件

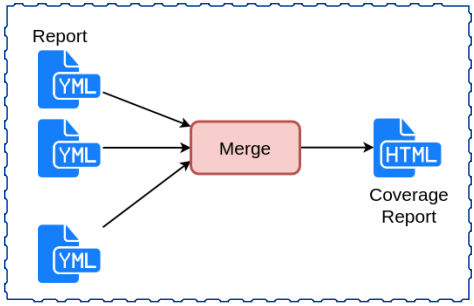
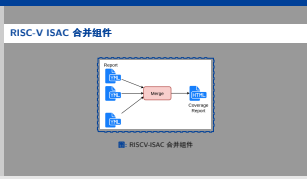


图: RISCV-ISAC 合并组件



RISCV-ISAC 还提供了一个简单的合并模块，可以合并多个 CGF 报告以创建单个报告。这对于为整套测试创建覆盖率报告非常有用。

谢 谢

谢 谢