

SAIL-MODEL 如何攻克 RVSC-V 可配置性

基于自定义 JSON 的可配置性模板生成与配置合规验证方法

Mingzhu Yan

2025-08-22

PLCT Lab

- 1. 背景 2
- 2. 方法 5
- 3. 展望 16

1. 背景

1.1 riscv 丰富的可配置性

1. 数十个扩展指令集
2. 不同扩展的依赖/兼容关系
3. 未定义行为
4. spec 不同版本
5. 不同的 profile (RVA22, RVA23 ...)

1.2 sail-riscv 的目标

SAIL-MODEL 项目是对 RISC-V 指令集架构的精确描述

- SAIL-MODEL 使用 SAIL 语言编写
- SAIL 是一个领域特定语言, 具有专门为编写指令集设计的语法
- SAIL 语言可以被编译为 C 语言, 再借助 gcc/clang 编译为可执行文件

SAIL-MODEL 的目标是实现**完全的可配置性**

2. 方法

2.1 sail 的 config 语法

SAIL 内置了特殊语法与 JSON 配置文件交互

```
{  
    "c1" : 64,  
    "c2" : 0,  
    "c3" : 8392413984723472389546328576138756413875644375,  
    "c4" : 999999999999999999999999999999999999999999999999999  
}
```

2.1 sail 的 config 语法

```
function main() = {  
  let x : range(0, 64) = config c1;  
  print_int("x = ", x);  
  let y : range(0, 1) = config c2;  
  print_int("y = ", y);  
  let z : int = config c3;  
  print_int("z = ", z);  
  let w : {'n, 'n >= 0. int('n)} = config c4;  
  print_int("w = ", w);  
}
```

> 为了实现任意精度整数, sail 的 cJSON 代码被修改过, 将所有的数字都视为字符串解析

2.2 配置模板生成

Generate config JSON files from a template #1151

Merged by Timmmm riscv:master ← Arielfoever:pr/jsontemp 6 hours ago

Conversation 31

Commits 3

Checks 30

Files changed 9



Arielfoever (Ariel Xiong) on Jul 19 • edited by Timmmm

Contributor

Generate a range of JSON config files from a template using `configure_file` instead of `jq`. Also remove the unused support for `.filter` files for first party tests, which also means we can remove the need for `jq` entirely.

Fixes [#1136](#)



<https://github.com/riscv/sail-riscv/pull/1151>

2.2 配置模板生成

```
{
  "base": {
    "xlen": @CONFIG__BASE__XLEN@,
    "E": false,
    "writable_misa": true,
    "writable_fiom": true,
    "writable_hpm_counters": {
      "len": 32,
      "value": "0xFFFF_FFFF"
    },
    "mtval_has_illegal_instruction_bits": false
  },
```

```
  "D": {
    "supported": true
  },
  "V": {
    "supported": true,
    "vlen_exp": @CONFIG__V__VLEN_EXP@,
    "elen_exp": @CONFIG__V__ELEN_EXP@,
    "vl_use_ceil": false
  },
  "B": {
    "supported": true
  },
```

2.2 配置模板生成

```
# Create a variety of configuration files for different
XLEN/ELEN/VLENs.
foreach (CONFIG__BASE__XLEN IN ITEMS 32 64)
  foreach(CONFIG__V__ELEN_EXP RANGE 5 6)
    foreach(CONFIG__V__VLEN_EXP RANGE 7 9)
      math(EXPR vlen "1 << ${CONFIG__V__VLEN_EXP}")
      math(EXPR elen "1 << ${CONFIG__V__ELEN_EXP}")

      set(config_filename
"rv${CONFIG__BASE__XLEN}d_v${vlen}_e${elen}.json")

      set(CONFIG_XLEN_IS_32 "false")
      set(CONFIG_XLEN_IS_64 "false")
      set(CONFIG_XLEN_IS_${CONFIG__BASE__XLEN} "true")

      configure_file(config.json.in ${config_filename})

      install(FILES ${config_filename}
              DESTINATION
${CMAKE_INSTALL_DATADIR}/${CMAKE_PROJECT_NAME}/config
              )
    endforeach()
  endforeach()
endforeach()
```

2.3 schema 自动生成

```
doc > examples > ≡ config_schema.sail
```

```
1  default Order dec
2
3  $include <prelude.sail>
4
5  val example : unit -> unit
6
7  function example() = {
8    | let n : {32, 64} = config some.integer;
9  }
```

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "some": {
      "type": "object",
      "properties": {
        "integer": {
          "description": "./doc/examples/config_schema.sail:8.21-8.40",
          "anyOf": [
            { "type": "integer", "const": 32 },
            { "type": "integer", "const": 64 }
          ]
        }
      ],
      "required": [ "integer" ]
    }
  ],
  "required": [ "some" ]
}
```

```
sail --output-schema schema.json example.sail
```

2.3 schema 自动生成







build > {} sail_riscv_config_schema.json > ...

```
1  {
2    "$schema": "https://json-schema.org/draft/2020-12/schema",
3    "type": "object",
4    "properties": {
5      "base": {
6        "type": "object",
7        "properties": {
8          "mtval_has_illegal_instruction_bits": {
9            "description": "riscv_platform.sail:52.46-52.92",
10           "type": "boolean"
11         },
12         "writable_fiom": {
13           "description": "riscv_sys_regs.sail:90.38-90.63",
14           "type": "boolean"
15         },
16         "writable_hpm_counters": {
17           "description": "riscv_sys_regs.sail:93.43-93.76",
18           "oneOf": [
19             {
20               "type": "array",
21               "items": { "type": "boolean" },
22               "minItems": 32,
23               "maxItems": 32
24             },
25             {
26               "type": "object",
27               "properties": {
28                 "len": { "type": "integer", "const": 32 },
```

验证方案选择依据

1. 许可证
2. 使用难度
3. 构建
4. 复杂度

Validator ¶

Name 	Languages	Dialects						License	Bowtie
Blaze	C++	4	6	7	2019-09	2020-12		AGPL-3.0 and Commercial	
f5-json-schema	C++	7						BSL-1.0	
JSON schema validation for JSON for Modern C++	C++	7						MIT	
jsoncons	C++	4	6	7	2019-09	2020-12		BSL-1.0	
Valijson	C++	7						BSD-2-Clause	

基于 C 库的验证方案

- <https://github.com/tristanpenman/valijson> : valijson 依赖于一个具体的工作 json 库实现, cJSON 未被支持
- <https://github.com/danielaparker/jsoncons> : header-only 但不是单文件, 支持 CBOR 等其他格式处理

基于调用外部工具的验证方案

- <https://github.com/santhosh-tekuri/boon> : 需要用户自行在本地安装 boon

3. 展望

3.1 自动生成验证代码

schema validator 无法完全满足我们的配置需求

- ELEN：单个元素的最大位数
 - 必须是 2 的幂
 - $ELEN \geq 2^3$
- VLEN：向量寄存器的位数
 - 必须是 2 的幂
 - $VLEN \geq ELEN$
 - $VLEN \leq 2^{16}$

3.1 自动生成验证代码

最终方案是修改 sail 编译器，自动生成验证代码

```
type elen_exp : Int = config extensions.V.elen_exp  
constraint 3 <= elen_exp <= 16 & elen_exp <= vlen_exp
```

sail 的 constraint 语法可以为多个类型变量施加类型约束