A Hybrid Verification Solution to RISC-V Vector Extension

Chenghuan Li
Mediatek.inc, Beijing, China
Chenghuan.Li@mediatek.com

Yanhua Feng
Mediatek.inc, Beijing, China
Yanhua.Feng@mediatek.com

Liam Li
Mediatek.inc, Beijing, China
Liam.Li@mediatek.com
Agenda

• Design/Testbench overview

• Instruction model auto-generation flow

• Exception verification

• Formal’s application
  • Early-stage design exploration
  • Hazard verification

• Summary
Design/Testbench Overview

Design: 12-stage pipeline, superscalar core and configurable L1/L2 cache

UVM-based simulation verification env.

Layered Sequences

- YAML
- python
- instruc
tions

RTL
agent

C Model
parser

score board
Instruction Model Auto-Generation Flow

• Issue statement:
  1. Frequent ISA changes from both RISC-V evolution and MTK’s customization.
  2. DV’s Need for RTL’s implementation information in both stimulus and checkers.

• Online vs. Offline instruction generator
Instruction Model Auto-Generation Flow (cont.)

• Auto-generate instruction classes per design spec.

YAML spec.
- Implementation related information

Template
- Instruction group

Python
- Python source code

Script
- Python script

Output
1. UVM-based instruction classes
2. Coverage

Instruction group
- GPR_ALU
- GRP_MEM
- GRP_MAC

Instruction class
- vadd
- vlw
- vmac
Instruction Class

- Base class defines the instruction’s prototype
- YAML auto-generated classes fulfill the specific implementation
Exception Verification

• RVV raises higher requirement for instruction’s contextual relevance.

### Exception category

<table>
<thead>
<tr>
<th>Instructions</th>
<th>VTYPE</th>
<th>Operand index</th>
<th>Un-align</th>
<th>Rsvd</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LMUL</td>
<td>SEW</td>
<td>VILL</td>
<td>vd</td>
<td>vs2</td>
</tr>
<tr>
<td>vsetvl(i)</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>CSR</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>ALU</td>
<td>wdn</td>
<td></td>
<td></td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>narr</td>
<td></td>
<td></td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>norm</td>
<td></td>
<td></td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Memory</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Special</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Customized</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
<td>√</td>
</tr>
</tbody>
</table>

### Snippet of RVV’s typical usage

**Loop:**

```plaintext
vsetvl a3, a0, E16, M4
vlh.v v4 (a1)
slit t1, a3, 1
add a1, a1, t1
vwmul.vx v8, v4, x10
```

```
vsetvl x0, a0, e32, m8
vsrl.vi v8, v8, 3
vsw.v v8, (a2)
slit t1, a3, 2
add a2, a2, t1
sub a0, a0, a3
```

**set vtype(CSR)**

- **load memory**
- **ALU operation**
- **store memory**
- **anymore?**
- **Exist**
Exception Verification (cont.)

**Exception verification**: traverses all possible exception cases

**Normal cases**: generate legal instruction as much as possible

1. Randomly select instruction based on current vtype (LMUL/SEW/VILL) setting
2. Decide operand index to avoid index overlap
3. Handle unaligned memory access & CSR R/W access policy

---

**Diagram**:
- **get_next_ins**
- **get_all_exception_types**
- **get_next_type**
- **blocklist?**
  - **Y**
  - **exception_scn**
  - **last_type?**
  - **Y**
  - **last_ins?**
  - **N**

---

**Flow**:
- Traverse all instructions
- Traverse all exception types of selected ins.
Early-Stage Design Exploration

• Issue statement

There are a thousand Hamlets in a thousand people’s eyes.

- Shakespeare

• Introduce formal property verification (FPV) in early stage

Traditional

DE’s design Proposal ➔ SPEC ➔ DV TB Buildup & Try run

Simulation iteration speed is low

This paper

DE’s design Proposal ➔ SPEC ➔ DV TB Buildup & Try run

Formal iteration speed is high

FPV property

GAP

SPEC

DV

Testbench

TB==RTL

RTL

Signoff

DE

RTL
Results

• ~80% early design issues detection with early-stage design exploration
Hazard Verification

• Issue statement
  1. Both inter and intra RAW/WAW/WAR data hazard needed handling in design
  2. Simulation failed to detect performance bugs w/o specific local checkers
Hazard Verification (cont.)

Use **liveness** property to assert typical instruction's retirement

Use **cover** property to explore longest stall cycles

Change liveness property to **safety** ones, re-prove it

---

```verilog
property ast_vmv_nfr_eventually_retire;
  logic [4:0] colored_idx;
  @(posedge clk) disable iff (!rstn)
  (insn_vld_e1 & insn_is_vmv_nfr_e1, colored_idx=insn_idx_e1) ->
      s_eventually (insn_wr_vrf & vrf_idx==colorder_idx);
endproperty

rvv_ast_vmv_nfr_eventually_retire: assert property (ast_vmv_nfr_eventually_retire);
```

```verilog
property cov_stall_raised_20t (stall);
  @(posedge clk) disable iff (!rstn)
  $rose (stall) -> stall [*20];
endproperty

rvv_cov_stall_raised_20t: cover property (cov_stall_raised_20t (x_stall));
```

```verilog
property ast_vmv_nfr_eventually_retire;
  logic [4:0] colored_idx;
  @(posedge clk) disable iff (!rstn)
  (insn_vld_e1 & insn_is_vmv_nfr_e1, colored_idx=insn_idx_e1) ->
      # [8:30] (insn_wr_vrf & vrf_idx==colorder_idx);
endproperty

rvv_ast_vmv_nfr_eventually_retire: assert property (ast_vmv_nfr_eventually_retire);
```
Function Bug Example

• Assert property: instruction ‘vnclip.qv’ cannot be retired forever.
• Root cause: design didn’t consider LMUL when handling RAW stall
  • Consider double narrowing, ‘vnclip.qv’ needed v0, v1, v2 and v3, total 4 registers.
  • Consider LMUL=MF4, ‘vnclip.qv’ only need v0.

Instruction sequence

1st vlw.v v1, (x2)
2nd vnclip.qv, v8, v0 (MF4)
3rd vlw.v v2, (x2)
4th vlw.v v1, (x2)
5th vlw.v v2, (x2)
6th vlw.v v1, (x2)
...

Cycle T

E1 | E2 | E3 | E4 | E5 | E6 | E7 | E8 | E9
---|----|----|----|----|----|----|----|----
6th | 5th | 4th | 3rd | 1st |     |    |    |    

Cycle T+1

E1 | E2 | E3 | E4 | E5 | E6 | E7 | E8 | E9
---|----|----|----|----|----|----|----|----
6th | 5th | 4th | 3rd | 1st |     |    |    |    

\[\text{unexpected RAW stall}\]
Performance Bug Example

- Cover property detected unexpected long stall
- Register retired order
  - Expected: v8 (vlw) -> v4 (vadd) -> v8 (vmac), totally need 10 cycles
  - Actual: v4(vadd) -> v8 (vlw) -> v8(vmac), totally need 14 cycles.

### Instruction sequence

1\textsuperscript{st} vadd v4, v2, v0  
2\textsuperscript{nd} vlw.v v8, (x2)  
3\textsuperscript{rd} vmac v8, v16, v24  
4\textsuperscript{th} vlh.v v10, (x3)  
5\textsuperscript{th} vlw.v v22, (x6)  
6\textsuperscript{th} vlw.v v27, (x7)

### Diagram

- E1, E2, E3, E4, E5, E6, E7, E8, E9
- 6\textsuperscript{th} 5\textsuperscript{th} 4\textsuperscript{th} 2\textsuperscript{nd} 1\textsuperscript{st}
- Intra waw stall
- Unexpected WAW stall
Results

- Early-stage bug hunting
- Signoff schedule shift left
- Better design quality
Summary

• YAML-based instruction model auto-generation flow responds well to RISC-V evolution and customization changes.

• Exceptions thoroughly verified with UVM-based solution.

• Hazard handling schedule shift left with simulation & formal hybrid solution.
Questions

• Any questions?