Improve the Quality of the Testbenches using specialized PySlint solutions

- Open-source based, customizable TestBench analysis

Balram Meghavath (Balram.Meghavath@Broadcom.com)
Srinivasan Venkataramanan (svenkat@asfigo.com)
Lint and Code Analysis

- Find bugs early
- RTL Lint well proven
- Verification
  - SystemVerilog TB
  - UVM Code base
  - Assertions
  - Acceleration
  - Emulation
  - Formal Verification

Source: Siemens EDA
RTL Lint is widely used in the field of digital design. It helps to identify and fix issues in RTL code.

 Ensures adherence to naming guidelines and coding style checks.

 Improves overall code quality and reliability.
Open-source RTL Lint - options

- Widely used flow – RTL Lint
  - Naming guidelines
  - Coding style checks
  - Synthesis checks
  - DFT checks
- Many teams even integrate this to CI/CD flows
- Good opensource alternates becoming available
TB Lint - Challenges and Alternatives

- New Paradigm: Code analysis approaches changing
- Lack of good parsers/tools/API for newer languages
- Slang Verible, Svlint, and PySlint are good opensource alternatives
- Provide flexibility and customization options
Testbench Linting

• Need solid parser front-end
• Verible – C++ based rules [https://github.com/chipsalliance/verible](https://github.com/chipsalliance/verible)
• Svlint – Rust based, custom plugin [https://github.com/dalance/svlint](https://github.com/dalance/svlint)
• UVMLint – Python based, custom plugin for UVM TB
  – [https://github.com/AsFigo/UVMLint](https://github.com/AsFigo/UVMLint)
• PySlint – Python based, works on top of slang/pyslang
  – [https://github.com/AsFigo/pyslint](https://github.com/AsFigo/pyslint)
### Sample rules in PySlint – Class, SVA, UVMLint

<table>
<thead>
<tr>
<th>PySlint Rule ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME_CL_PREFIX</td>
<td>Naming/style checks</td>
</tr>
<tr>
<td>NAME.CG_PREFIX</td>
<td></td>
</tr>
<tr>
<td>NAME_ASM_PREFIX</td>
<td></td>
</tr>
<tr>
<td>NAME_COV_PREFIX</td>
<td></td>
</tr>
<tr>
<td>SVA_MISSING_LABEL</td>
<td>Assertion specific. Performance, Debug categories</td>
</tr>
<tr>
<td>SVA_MISSING_ENDLABEL</td>
<td></td>
</tr>
<tr>
<td>SVA_NO_PASS_AB</td>
<td></td>
</tr>
<tr>
<td>SVA_MISSING_FAIL_AB</td>
<td></td>
</tr>
<tr>
<td>CL_METHOD_NOT_EXTERN</td>
<td>Class specific – style, functionality</td>
</tr>
<tr>
<td>CL_MISSING_ENDLABEL</td>
<td></td>
</tr>
<tr>
<td>FUNC_CNST_MISSING_CAST</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UVMLint Rule ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UVM_DRV_MISSING_PARAMS</td>
<td>Driver should have REQ parameter overridden</td>
</tr>
<tr>
<td>UVM_AGENT_IS_ACT_REUSE</td>
<td>Agent should check is_active before constructing drivers, sequencer for reuse</td>
</tr>
<tr>
<td>UVM_AGENT_IS_ACT_HIDE</td>
<td>Do not re-declare is_active field in uvm_agent</td>
</tr>
<tr>
<td>UVM_MON_MISSING_VIF</td>
<td>Monitor should have a handle to virtual interface</td>
</tr>
</tbody>
</table>
Case study: SystemVerilog constraint dist issue

class data;
rand bit [15:0] field1;
constraint c_f1 { field1
dist {[0:31] := 1, [32:65535] := 1}
Case study: UVM agent - is_active redeclared

- The value of is_active as declared above would be set to be UVM_PASSIVE - this is against the UVM standard implementation that makes it UVM_ACTIVE as the default.
- Protected would prevent an env class unable to access this field - a common code style other UVCs (that do not override base class’s is_active unlike above).
- What’s the point of using uvm_agent here as base class and not uvm_component? - make no mistake; we are not suggesting you do that, rather questioning why the re-declaration in user agent code.
Case study – improper use of array reduction methods in constraints

New rule - SV Constraint using array reduction method, check for casting #10

svenka3 opened this issue 12 hours ago · 1 comment

Solutions: cast into a variable that won’t overflow num_list.sum() with (6*(item)) == 1

Describe the solution you’d like

Add a lint rule to check for missing cast in such cases.

Pyslint: Violation: [FUNC_CNST_MISSING_CAST]: Potentially incorrect constraint expression! An expression involving array-reduction method sum() was found, but is missing an explicit cast. This can lead to strange results as array reduction methods return an expression of the size of its elements, check if you need a with (int’( cast around the following expression:

num_list.sum()
Testbench Linting - summary

• Exciting new frontiers

• Opensource can now lead the way for commercial tools to follow
  – Than just playing a catch-up game
  – Faster turn-around time
  – Potentially vast set of developers!

• Invite-to-innovate:
  – What analysis would you build on top?
Integrate Lint to CI/CD pipeline flow

- Subset of Lint rules
- Customize as per project stage/phase
- Stricter as it gets mature
- Applies to RTL & TB
- Ideally use open-source tools
- Free of license restrictions
- Easy to customize
- Commercial license models available (MIT)
Pyslint approach

- Python based SystemVerilog Linter built based on slang/pyslang (SV parser)
- Help customers build on top with:
  - Own rules (Customers own the rules, proprietary if need-be)
  - Customize existing rules
  - Newer analysis tools/apps
- Not intended as a traditional EDA product!
  - Rather open room for innovation

- Qualitative & Quantitative metrics
  - Unique in Lint/static analysis
- Quick Setup & Less dependencies
  - Pyslang (open-source)
  - make
  - Python 3.xx
- Many style check rules now available
  - SystemVerilog TestBench rules
- Introducing UVMLint (UVM Linter)
  - Linter for UVM code
  - UVM rules check now available via GitHub!
Conclusion

Integration with CI/CD flows helps in continuous improvement

Adopting new approaches and alternatives is essential for evolving languages and paradigms

RTL & TB linting and code analysis are essential for quality digital design

Using popular tools and alternatives improve code reliability