SLSV
The Shakti LockStep Verification Framework

Paul George | SHAKTI Group | CSE Dept | RISE Lab | IIT Madras
Outline

- SLSV - An Overview
- Verifying a RISC-V System
- Constraints to Random Simulation Based Verification
- The Beautiful RISC-V Software Ecosystem <3
- SLSV Architecture
- Challenges
- Contributing ! :)

The Shakti LockStep Verification (SLSV) Framework is an Open Source Dynamic Simulation based Verification and post silicon Validation Framework for RISC-V System on Chip solutions.

- Allows designers to perform functional verification with directed and random test vectors against target device(s) under verification (DUV).
- Tracks relevant architectural and micro-architectural states and evaluates them against a specified golden model. The golden model can be a functional simulator/emulator or a executable formal model.
- Can be setup against RISC-V designs that can be wrapped with an SLSV compatible interface.
SLSV Objectives

1. Functionally Verify RISC-V SoCs.

2. Detect design inaccuracies and present relevant context to effectively debug the target.

Verifying a RISC-V System
Verifying a RISC-V System

- ISA Spec.
- Accelerator Spec.
- Peripheral Spec.
- Interconnect Spec.

- Platform Spec. → $\mu$-Arch in HLS → RTL → Synthesised Netlist → GDS → Silicon

- Design Error
- HLS Tool Guarantees
- Static Formal Equivalence Tools Tool Guarantees
- Process Variance
Verifying a RISC-V System

Executable Formal/Golden Model

ISA Spec.
Accelerator Spec.
Peripheral Spec.
Interconnect Spec.

Platform Golden Model

Design Error

Platform Spec.

μ-Arch in HLS

HLS Simulator

HLS Tool Guarantees

Emulation w/o Trace

Arch. + μ-Arch State
Cycle Accurate
State Variable

Emulation with Data Trace

Arch. + μ-Arch State
Cycle Accurate
State Variable Visibility

Through Data trace

more LUT's / Emulation rate.

FPGA Emulation

RTL

Synthesised Netlist

GDS

Process Variance

Silicon

:: Architectural State Variables
:: Not Necessarily Cycle Accurate
:: All State Variables visible with no costly overheads

:: Arch. + μ-Arch State
:: Cycle Accurate
:: All State Variables visible with no costly overheads

:: Simulation
:: Arch. + μ-Arch State
:: Cycle Accurate
:: All State Variables visible with no costly overheads

:: Simulation Rate
:: Slow

:: Arch. + μ-Arch State
:: Cycle Accurate
:: All State Variables visible with no costly overheads

:: Simulation Rate
:: Slow

:: Arch. + μ-Arch State
:: Cycle Accurate
:: All State Variables visible with no costly overheads

:: Simulation Rate
:: Very Slow
Simulation based Verification

Pros.

● Simple approach!
● Do not really have any other way other than simulation based verification to check equivalence between Architectural Specs & complex u-Architectures.
● Trades off between the cost of writing directed tests and the additional simulation time necessary to have possibly “covered” corner cases.

Cons.

● *VERY* Large state space to “cover”.
● “Coverage” largely dependant on effective test vector generation & number of simulation cycles run.
Constraints To Simulation based Verification Strategies

1. Maximize Simulation Cycles :: More Cycles => More States explored => better “Coverage” (in terms of states traversed)

2. Maximise State Visibility :: More Visibility => Better Context for design engineer to debug the discovered error.

3. Minimise Cost :: Engineer Hours / Simulation Emulation Infrastructure / Test Tapeout Iterations / Delay to market(opportunity).
Verifying a RISC-V System

- **Executable**
  - Formal/Golden Model
- **Platform**
  - Golden Model
- **HLS Simulator**
  - Simulation
  - Arch. + µ-Arch State
  - Cycle Accurate
  - All State Variables visible with no costly overheads
  - Slower than HLS
- **RTL**
  - Simulation
  - Arch. + µ-Arch State
  - Cycle Accurate
  - All State Variables visible with no costly overheads
  - Simulation Rate - Slower than HLS
- **Synthesised Netlist**
  - Simulation
  - Arch. + µ-Arch State
  - Cycle Accurate
  - All State Variables visible with no costly overheads
  - Simulation Rate - Significantly Slower than RTL
- **GDS**
  - Simulation
  - Arch. + µ-Arch State
  - Cycle Accurate
  - All State Variables visible with no costly overheads
  - Simulation Rate - "Slow"
- **FPGA Emulation**
  - Emulation with Data Trace
  - Arch. + µ-Arch State
  - Cycle Accurate
  - State Variable Visibility Through Data trace
- **Silicon**
  - Target with Data Trace
  - Arch. + (µ-Arch State ?)
  - Cycle Accurate
  - State Variable Visibility Through Data trace
  - Emulation w/o Trace
  - Arch. + µ-Arch State
  - Cycle Accurate
  - State Variable Visibility at the cost of more LUT's / Emulation rate.
  - Target w/o Trace
  - Arch. + (µ-Arch State ?)
  - Cycle Accurate
  - State Variable Visibility at the cost of significant debugger overhead.
Verifying a RISC-V System

2 Types of Spec Reference Models

6+ Types of Verification Targets

Target Specific Interface Mechanisms
Verification Framework Building Blocks

- Test Sources
- Test Dispatch mechanism
- Reference model(s)
- Trace mechanism
Open Source Verification Aids - RISC-V

RISC-V Tests :: Test Suite for the Base ISA

RISC-V Compliance :: Group Chartered with the Development with Directed Tests to establish compliance. Good Source for Bring up tests.

Shakti AAPG :: Constrained Random Test Program generator.

RISC-V Torture :: Constrained Random ISA Verification Framework evaluating its results against Executable ISA simulators
Constrained Random Test Sources

Formal Spec TG (Formal Model Efforts)
Many formal specs written across different specification approaches.
Varying degrees of completion. Most Complete can (almost ?) Boot Linux as a platform.

Memory Model TG (Memory Consistency Model Efforts)
Models Describing Memory Consistency Spec. (RV-WMO) critical to Multi HART Designs.
LockStep Verification W/ Formal Models

Designed along the lines of ARM`s ISA Formal Approach.

- SLSV acts as a monitor operating on the Arch & uArch state trace.

Fig. 2. A 5-stage processor pipeline with state abstraction and follower

Fig. 3. Defect Detection by Phase

Fig. 4. Defect Detection by Time


Accepted for publication at Computer Aided Verification, 2016.
LockStep Verification W/ Formal Models

- ISA Spec.
- Accelerator Spec.
- Interconnect Spec.
- Peripheral Spec.

Executable Formal/Golden Model(s)

- Functional
- Structural

Coverage Estimators

Verification Event Handler

Test Dispatch

- RISC-V Tests
- RISC-V Compliance
- Benchmarks
- RISC-V Torture
- Shakti AAPG
LockStep Verification Against Existing Verified References
The SLSV Interface

1. Initialise();
2. LoadBinary();
3. Synchronise(); // Execute Bootrom and reach the test vector
4. Single_Step(); // Get next temporally available state vector
5. GetVariable();
6. SetVariable();
7. GetMemory();
8. SetMemory();
9. DumpMemory();
10. Checkpoint();
11. Restore();
12. Any Additional Methods required to Provide the above Interface To your Target
The SLSV Interface Minimal

1. Initialise();
2. LoadBinary();
3. Synchronise(); // Execute Bootrom and reach the test vector
4. Single_Step(); // Get next temporally available state vector
5. GetVariable();
6. SetVariable();
7. GetMemory();
8. SetMemory();
9. DumpMemory();
10. Checkpoint();
11. Restore();
12. Any Additional Methods required to Provide the above Interface To your Target
LockStep Verification W/ Formal Models

System state updates are cumulatively read at the minimum granularity at which model checking can be done.

- **Per Cycle ::** uArch and peripheral and interconnects
- **Per Instruction retirement ::** Single Issue - ISA only
- **Per Issue Width Instructions retired retirement ::** Multi Issue uArch.

Using the SLSV Framework

Interface SLSV with your Verification Target. (Run Control + Encoded Trace stream)

Script Test Driver.

Evaluate State Trace against Models and any other Trace Subscriber.

SLSV can be used as a Python Library (right now). SLSV has been written with a multi-threaded (To be MPI enabled for Distributed testing) C++ backend.

The Scripting frontend interface has been done with SWIG, and can in future support Perl or TCL too.
Representation Example

State Add HART, Memory, Non Hart State variables (uArch etc.).

Interface, Type, Configuration.

Configure Verification Events

Test example :: Run Risc-V Tests.

Load Tests on target.

Run test.

   Get State Updates.

   Check Trace updates for validity against the reference,

   Repeat

If Not Valid, Raise Event

If End of Test, Raise Event

If End of test Event, Terminate Test, Begin Next Test.
Challenges

SLSV is not a Substitute of Module-level IP Verification. Verifies overall system integration and The RISC-V Compliant Core ISA.

Architectural state can be lost or difficult to define with certain uArch`s.

Writing Functional Coverage Estimators is an *art*. Beauty is in the eye of the beholder.

Reiterating :: Only as good as the effectiveness of Random Test Generation tool and The Number of Machine cycles checked.

[1] :: End To End Verification of ARM Processors with ISA-FORMAL https://alastairreid.github.io/papers/CAV_16/

Conclusions

SLSV - The Shakti LockStep Verification framework

- Open Source Interface Agnostic Verification Framework
- Scalable State variable Addressing Scheme for interfacing with Trace Subscribers Like Formal Models, Coverage estimators, Other Analytics.
- Script Test Drivers in Python.
Contributing To SLSV!

Please Evaluate SLSV as an Architecture.

SLSV is still early alpha - IDEAS

- Help Refine SLSV’s Architecture,

Contribute Code

- Contribute Formal Models For your IP.
- Formal Models for Inter-connects.
- Test Drivers to use SLSV with your Open Source SOCs
- Interface/Wrapper scripts to EDA Tools.
- Data Trace IP
- Test Vector Sources, Directed Tests for RISC-V, IP.

SLSV vs UVM

SLSV Strengths

- Simple Approach - using python / or any other scripting language
- Can be used with high levels design flows where intermediate verilog not necessarily interactable.
- Accelerated Verification with emulation. (Trade off State Variables tracked vs - emulation rate)
- Integration of formal models into Verification Flow!

UVM Strengths

- Existing functional coverage evaluation approaches for some IP.
- Standard Interfaces, Existing Experience pool.
<table>
<thead>
<tr>
<th>SLSV</th>
<th>vs</th>
<th>RISC-V Torture</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Randomized Simulation based verification</td>
<td></td>
<td>- Randomized Simulation based verification</td>
</tr>
<tr>
<td>- Constrained random test sources :: RISC-V Torture + Shakti AAPG</td>
<td></td>
<td>- RISC-V Torture Test Generator as test vector source.</td>
</tr>
<tr>
<td>- Directed tests from RISC-V Tests, RISC-V Compliance.</td>
<td></td>
<td>- Compares “Signature” - Relevant architectural state against that output by an ISA simulator. Algorithmically searches for failure point in the test case.</td>
</tr>
<tr>
<td>- Works with Platform ISS in a fashion similar to RISC-V Torture, But evaluates state for correctness at every update. Trade off Simulation time vs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Differentiation

- Tracks State Updates over a variety of Temporal Granularities.
- User defined event handlers to Verification events.
- Designed to be interfaced with a variety of models => Can be used for full system - verification.
- Sequentially Evaluates the data trace and reports bugs.