White Paper

A Hierarchical and Configurable Strategy to Verify RISC-V based SOCs

Arun Chandra, Mike Bartley, Tessolve

1.0 Abstract

RISC-V (pronounced "risk-five") is an open, free ISA enabling a new era of processor innovation through open standard collaboration. Born in academia and research, RISC-V ISA delivers a new level of free, extensible software and hardware freedom on architecture, paving the way for the next 50 years of computing design and innovation.

A RISC-V microprocessor can be configured in several architectural modes depending upon the target market and applications. Further, each microprocessor implementation can have different micro-architectural parameters depending upon performance, and power considerations. Examples of such micro-architectural parameters are cache sizes, the use of branch prediction, result forwarding, and pre-fetch to name a few.

This paper outlines a hierarchical and configurable verification strategy for RISC-V based IP and SOCs. A three-level (unit, core and SOC) hierarchy is proposed for test benches. Each level of the hierarchical test bench is configurable for both architectural and micro-architectural parameters. At the heart of the verification strategy is an ISG (Instruction Stream Generator) and a UVM test bench. The ISG can be configured according to the RISC-V architecture and then constrained to verify micro-architectural features. The generation of the specific configurable UVM test bench is automated based on a configuration file. The checkers, active test bench items like injectors, and coverage objects, are mostly portable across the various hierarchical levels, and are configurable based on the configuration file.

At the SOC level the tests are less ISG based and tend more towards C-based integration and use case tests ideally suited to the use of portable stimulus (as defined in the Accellera Portable Test and Stimulus Specification and supported by Questa inFact, Cadence Perspec and Breker Systems Trek). This allows tests to be easily ported across multiple SOCs with minimum effort, and to also be used in silicon validation.

Doc Revision:	Version 1.0
Doc Revision Date:	March 1, 2018

www.TESSOLVE.com

sales@tessolve.com

A Hero Electronix Venture

1.0 **Introduction**

A RISC-V based SOC can be configured into different implementations based on architectural or micro-architectural parameters. To address the verification challenge this poses, a hierarchical and configurable verification methodology is proposed. A three-level hierarchy is proposed. The lowest level of the hierarchy is the unit-level. Two unit-level test benches are proposed. These are 1) Execution (Pipeline) Unit, and 2) Cache (L2) Unit.

The Execution (Pipeline) Unit consists of the major pipelines components like Instruction Fetch, Instruction Decode, Instruction Execute, and Load Store. Both the level one caches (instruction, and data) are included in this unit. The Cache (L2) unit consists of the second level cache. The second level of hierarchy is the Core Level. At this level multiple Execution Units and the L2 cache are connected via a coherent bus. Both the unit-level test benches, and the core-level bench can be configured for a specific implementation. The highest level of the hierarchy is the SOC which consists of the core and peripherals like PCIe, and MIPI.

An important feature of the verification methodology is that a test bench at any level is configurable based on architectural and micro-architectural parameters. Further, based on a configuration file, the test bench is automatically generated for the desired level and configuration. Subsequently, tests both directed and automatically generated can be run on the test bench.

The stimulus for the test benches can be instruction-based for ISA heavy components like the Execution Units (Pipeline), or transaction-based for testing the L2. Checkers are reference-model based or assertion based. Checking the pipeline is done using a reference model checker, and it is a same for the L2 where a L2 behavioral model is needed. Additional checking is done via assertions. An example of an assertion check is that READ and WRITE are mutually exclusive. Additionally, loaders and injectors are part of a generated test bench, and generated via a configuration file. An example is a 32KB two-way set associative cache pre- loader.

In the rest of this document we cover Test Bench Architecture, the Stimulus, Checkers, Pre-loaders and Injectors, Coverage and Development Milestone, and Conclusion.

2.0 **Test Bench Architecture**

The hierarchical test bench architecture is show in Figure 1. The stimulus, portable checkers, and interfaces are show in this figure.

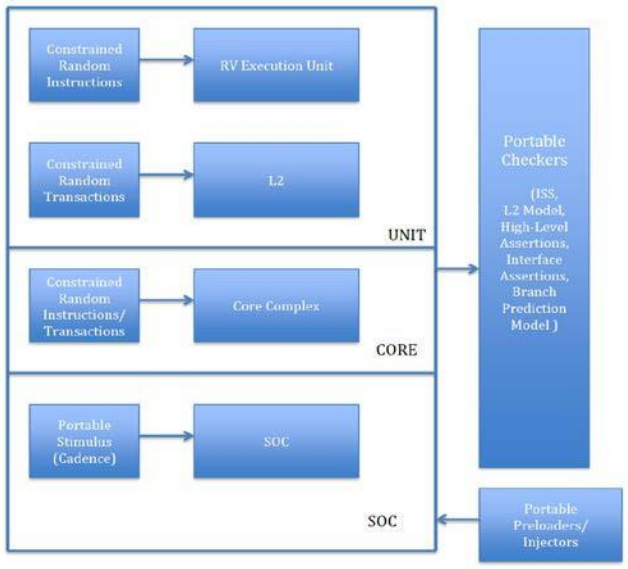
Constrained Random Instructions
RV Execution Unit
Constrained Random Transactions
L2
UNIT
Constrained Random Instructions/ Transactions
Core Complex
CORE
Portable Stimulus (Cadence)
SOC
SOC
Portable Checkers (ISS, L2 Model, High-Level Assertions, Interface Assertions, Branch Prediction Model)
Portable Preloaders/ Injectors

Figure 1 Hierarchical Test Bench Architecture

2.1 Unit-Level Test Benches

There are two test benches at the unit level, these are the execution unit (pipeline), and the second-level cache (L2)

2.1.1 RV Execution Unit (Pipeline) Test Bench

The RV Execution (Pipeline) Unit Test Bench verifies the single-issue, in-order pipeline. The five-stage pipeline consists of instruction fetch, instruction decode, execute, memory access, and write back. The instruction cache and data cache are also part of the execution unit. A unit to handle interrupts from the pipeline perspective is also part of the execution unit.

As the short pipeline does not have to deal with micro-architectural verification bottlenecks of a longer superscalar pipeline, it is recommended that all the components in the pipeline be treated and tested as one unit.

Examples of such bottlenecks are register renaming, floating point converts for non-committed instructions, stalls due to load store dependencies, integer and vector register un-naming due to branch misprediction, and reservation station stalls. However, one micro-architectural area that needs to be handled is branch prediction. The branch predictor comprises a branch target buffer (BTB), a branch history table (BHT), and a return-address stack (RAS).

The stimulus to this unit will be RISC-V instructions whose binary values will be loaded into a L2 behavioral model. These instructions could be directed hand-written tests, or tests output from a random Instruction Stream Generator (ISG). Additional inputs apart from instructions will be interrupts, and injected errors. The test bench also contains pre-loaders to preload the caches, and branch predictor array structures.

The reference-model checker for this unit will be an instruction-based instruction set simulator (ISS). A trace tool will monitor the RTL for PC, and Register Value Updates, and will compare against the output of the ISS. Additionally, micro-architectural checkers will be added especially with respect to branch prediction, and exception and interrupt handling.

2.1.2 L2 Test Bench

The L2 test bench will verify the second-level cache. Both its interface to the level-one (I and D) caches, and main memory will be verified. The stimulus to this unit will be transactions including, Read, Write, Invalidate, and Refill. The stimulus to this unit will come from a constrained-random transaction generator like a UVM sequencer. Additional inputs to this unit will be injected errors to test the ECC mechanisms. The test bench also contains pre-loaders to preload the caches,

The reference-model checker for this unit will be a reference model (L2 Behavioural Model). This model will model the L2 at the transaction level. The L2 state, in addition to transactions output will be compared against the RTL. Additionally, micro-architectural checkers will be added, especially if there are low-power features included in the implementations.

2.2 Core Test Bench

The Core-Level Test bench tests the component in the core complex. These include the RISC V Execution units, the Coherent Bus, and the L2 Cache. It also contains the interrupt units, and the debug unit. The stimulus into the core will be both instruction-based, and transaction-based. As the RISC-V execution units, and the L2 have been verified at the unit level, the focus of core-level verification will be stressing the interconnect fabric, and the interrupt, and debug units.

The stimulus for this bench will come from the various ports. RISC-V assembly stimulus generated by hand, or using a random instruction generator or ISG will come from the main memory behavioral model connected to the memory port. The instruction-based tests should generate traffic to the system port for un-cached access to high bandwidth peripherals. The instruction-based tests should also be able to generate traffic to the peripheral port for accessing peripheral devices. The system port and peripheral port can be mapped into two different address ranges.

The stimulus for the Front Port is transaction-based and comes in the form of requests to the ITIM, and DTIM. In addition, the core-level test bench has transaction-based stimulus for interrupts, and debug requests.

All the checkers from the unit-level are ported to the core level. Additionally, at least four categories of checkers are added at the core-level. These area: 1) An interface checker to check for bus transactions on the coherent bus, 2) A checker to check interrupt request, and subsequent servicing, 3) A checker for debug requests, and servicing, and 4) A checker for Port requests and servicing. These checkers can be implemented as UVM style scoreboards.

Checkers for arbitration and micro-architectural checkers will be added as needed. An example of an arbitration checker is to guarantee that the I/O ports get fair access and are not timed out. An interrupt priority checkers checks that if two interrupts are pending, the higher priority one gets serviced first.

2.3 SOC Test Bench

The SOC test bench is the top-level bench and exercises the interfaces between the core complex and the peripherals like PCIe and MIPI. The input into the SOC bench is based on the Portable Stimulus standard proposed by Accellera and supported by Mentor, Cadence and Breker Systems.

Portable stimulus provides a specification of test intent and coverage at a higher-level of abstraction. Also, it provides graph-based randomization. The Portable Stimulus will be generated for a specific implementation using the configuration file.

All the checkers from the unit-level and core level are ported to the SOC level. Additional VIP checkers from the PCIe, and MIPI will be integrated. Finally, interface checkers will be built at the SOC level.

2.4 Configurable Test Bench Generation

The generation of an implementation specific test bench is based on a configuration file shown in Figure 2. The fields in the configuration file, which are both architectural and micro-architectural determine the implementation specific test bench. The major fields in the configuration are described below.

```
SOC-COMPONENTS = RVCore, PCIe, L2, MPHY;
RVEXE-COMPONENT-ARCH = I,M,F;
RVEXE-COMPONENT-URACH = 32I(2), 32D(2), BTB, BHT(1), RAS;
L2-COMPONENT-UARCH = 2MB(2), ECC;
L2-COMPONENT-POWER = 1;
CORE-COMPONENTS = RVExe[0..3], L2, TileLink, CLINT, PLIC, Debug,
I/O[M,F,S,P];
LOW-POWER = Domain(1), Clock-Gating(ON);
```

Figure 2 Configuration File for Test Bench Generation

The SOC-COMPONENTS field lists all the components of the implementation. The example shown below shows a RISC- V cores complex, PCIe, L2, and MIPI amongst other components.

The RVEXE-COMPONENT-ARCH field lists the ISA variant for the RISC-V core. This describes the number of general purpose registers, and the various extensions (M, C, A, F, D, and Q). The example below shows a RISC-V execution unit supporting 32 registers, multiply and divide, and single precision floating point.

The RVEXE-COMPONENT-UARCH field lists the micro-architectural features for the specific implementation. These include the cache sizes, and associativity, and branch prediction structures. The example below shows a two-way set associative 32KB instruction and data cache, a BTB, a single-level BHT, and a RAS.

The L2-COMPONENT-UARCH lists the second-level cache micro-architectural features. This includes the cache size and associativity, and error correction if enabled. The example below shows a 4-way set associative 2 MB cache.

The L2-COMPONENT-POWER field lists the number of power domains for the second level cache. This is a low-power feature.

The RVCORE-COMPONENTS field lists all the core components for the specific implementation. The example below shows, four RV Execution units, the L2 cache, the inter-connect, the interrupt, and debug unit, and the I/O ports.

The LOW-POWER field describes the low power techniques used in the SOC like clock-gating, and multiple power domains.

LOW-POWER = Domain(1), Clock-Gating(ON)

The desired test benches are generated based on a command line that specifies the benches needs, and the configuration file. As an example, all the test benches (Unit, Core. SOC) are generated using the configuration file, and the unit-level (RVEXE) bench shown in Figure 3 is generated using the second command line.

Cmd1: rvtbgen –all rv101.config

Cmd2: rvtbgen –rvexe rv102.config

Subsequently, after the appropriate test bench is generated, random tests using the test pattern generators are generated. Additionally, directed tests can also be written to run on the appropriate test bench. The overall methodology for configurable test bench generation is shown in Figure 4.

3.0 **Stimulus**

3.1 Areas Under Test

The configuration file is key to generating the constrained-random stimulus. As an example, no floating instructions will be generated if the F mode is not supported. Another example is that back to back branch instruction generation-weight will be lower if branch prediction is not supported. At the unit level, stimulus will be provided to the RVEXE unit or L2 unit. Examples are integer instructions for the RVEXE unit, and L2 DCache interaction for the L2 unit.

The core complex unit stimulus will include Cache Coherency, and Virtual Memory and Protection handling. Finally at the SOC-level interrupt handling, reset, and Low Power features will be tested.

3.2 Instruction-Based Stimulus

Instruction-based stimulus comes from a RISC-V instruction stream generator, constrained by both architectural and micro-architectural constraints. Additionally, instruction-based stimulus can also come from directed tests. Examples are integer instruction, branch instructions, floating point instructions, or memory instructions.

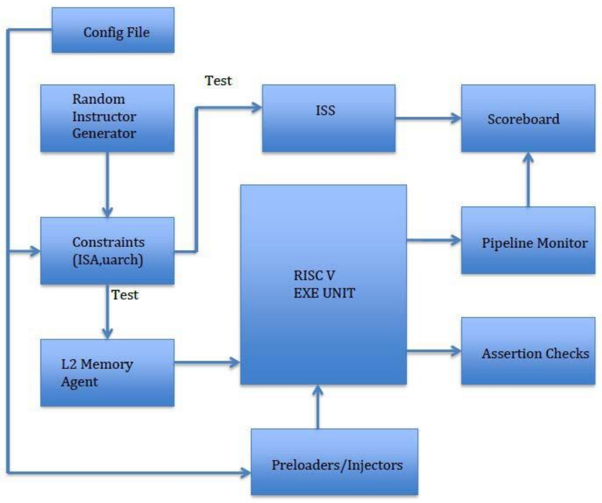

Figure 3 Unit-Level (EXE) Test Bench

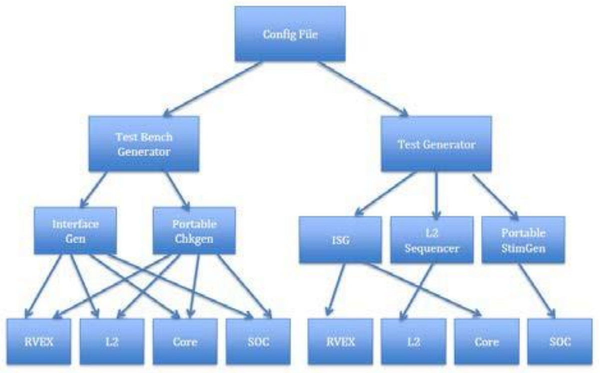

Figure 4 Hierarchical Test-Bench Generation

3.3 Transaction-Based Stimulus

Transaction-based stimulus used in the L2 and Core is constrained by both architectural and micro-architectural constraints. It is generated by UVM-Style sequencers, which subsequently call sequences. Additionally, transaction-based stimulus can also come from directed tests. Examples are L2-I/D Cache interactions, L2-Memory interaction, and L2-Error Handling.

4.0 **Checkers**

4.1 Reference Model-Based Checkers

The following reference-model checkers are need. They will be generated from the configuration file. As an example, the L2 cache size and associativity will determine the L2 behavioral model, and subsequent checker. Also, the branch prediction checker can be customized based on the RAS availability.

1. Pipeline Checker
2. Branch Prediction Checker
3. L2 Checker

At the unit-level, for the RVEXE unit the following reference-model based checkers will be needed:

Pipeline Checker – The reference model for this checker will be the ISS. A pipeline monitor from the RTL will extract PC update, and Register updates at instruction commit. These values will be checked against the output of the ISS

Branch Prediction Checker – To accurately verify branch prediction, a reference model will be built to model the branch prediction structures (BTB, BHT, RAS).

For the L2 unit, the following reference model checkers will be needed.

L2 Checker – To accurately verify the L2, a L2 behavioral model will be built. The output of this checker will be checked against the RTL at a transaction granularity.

These reference model checkers will be portable to the core complex and SOC level.

4.2 Assertion Checkers

Two kinds of assertion checks will be needed, these are low-level assertion checks, and high-level assertion checks. All these assertion checks will be portable for all three hierarchical levels, unit, core, and SOC.

Low-level assertions are written at the unit-level for the RVEXE, and L2 and are internal to the module. It is highly recommended that the RTL writer creates these assertions in conjunction with the RTL. Examples of low-level assertions are:

Request-Grant: A request is granted within a certain number of cycles.

One-Hot: The output of a signal is always one-hot.

Mutually-Exclusive: Read and Write are mutually exclusive.

High-level assertion checks can be written using low-level assertion checks, and it is recommended that the verification engineer write these checks. Areas where high-level assertion checks are recommended are:

Interface Checks – Checking the interface between the various components of a SOC. As an example, the interface between the L2 and the RVEXE.

Cache Coherence Protocol Checks – The cache coherence protocol can be verified by providing a high-level SVA-based checker to check the finite-state machine. In some cases cache coherence can also be checked by developing a reference model.

Bus Transactions – Checking that the bus or the interconnect, handled all requests, and handled them in order with the right priority.

All assertion-based checkers should be written in System Verilog, to be fully compatible with UVM. All assertions can be input into a formal verification tool for static formal verification.

5.0 **Pre-loaders/Injectors**

5.1 Cache/Array Loaders

The cache pre-loaders will be generated based on a specific implementation configuration (cache size, associativity) provided in the configuration file. The cache loaders will be needed to preload the level one and level two caches during reset. This is to prevent running through the entire boot sequence. Additionally, cache preloading is required to get the cache initialized to a certain state to verify interesting scenarios (cache coherence) in an accelerated fashion.

The array pre-loaders will be generated based on a specific implementation configuration (BTB size) provided in the configuration file. The array loaders will have the ability to preload array structures in the design, like the BHT. The primary use for array pre-loaders will be to be verify hard to test features in an accelerated fashion.

5.2 Injectors

Injectors can be used in all three levels of hierarchical test benches. These injectors will be generated based on the configuration file. For example, for ECC supported memory single bit errors, and double bit errors can be injected. Three categories of injectors will be needed

Interrupt Injection – To test both local and global interrupts, the interrupt injector will inject an interrupt as a request.

Error Injection – To test the reliability features in the design like ECC the error injector will be needed to inject single or double bit errors.

Event Injector – Debug requests, and other interesting events will be injected by the event injection mechanism.

All injectors will be portable in all hierarchical levels, and configurable via the configuration file.

6.0 **Coverage**

6.1 Coverage-Based Methodology

At each level of the hierarchy unit, core, or SOC a coverage-based methodology will be used. Coverage categories are list below:

Machine-Generated Code Coverage: Line, Code, Toggle, Expression.

Functional Internal: Internal coverage objects.

Functional Interface: Interface coverage objects.

Functional coverage objects are generated from a test plan. Also, functional coverage objects will leverage assertions both low-level, and high-leve

7.0 Conclusion

This document shows hierarchical and configurable verification strategy to for RISC-V based SOCs. A three-level hierarchy is proposed for test benches. The three levels are:

1. Unit,
2. Core
3. SOC

Each level of the hierarchical test bench is configurable for both architectural and micro-architectural parameters. The generation of the specific configurable test bench is automated based on a configuration file.

This document also lists the areas under test, and stimulus and checkers needed.

References

1. SiFive, "U54-MC Core Complex Manual," Oct 4, 2017.
2. A. Waterman and K. Asanovic, Eds.,The RISC-V Instruction Set Manual, Volume I:User-Level ISA, Version 2.2, May 2017.
3. The RISC-V Instruction Set Manual Volume II: Privileged Architecture Version 1.10, May 2017.
4. Accellera, Portable Stimulus Early Adopter Specification," June, 2017.
5. S. Gupta, "Efficient Verification of Mobile SOCs with Perspec and Portable Stimulus, CDN Live Conference, April 2017.
6. University of Berkeley Architecture Research, "TileLink Protocol v0.3.3," 2017.

About Tessolve

TessolveDTS, Inc, (Tessolve) provides services and products to organisations developing complex products in the microelectronics and embedded software and systems industries.

Tessolve operates globally with offices in the UK, China, Germany, India, Singapore, Malaysia, Ja the USA plus a network of international partners.

www.tessolve.com