Advanced RISC-V Verification Technique Learnings for SoC Validation

Using Breker SystemVIP for RISC-V System Ready

Adnan Hamid, CTO, Breker Verification Systems

Verification Futures Austin 2023
Agenda

- Test Suite Synthesis and SystemVIP
- RISC-V Core Verification SystemVIP
- RISC-V SoC Verification SystemVIP
Agenda

- Test Suite Synthesis and SystemVIP
- RISC-V Core Verification SystemVIP
- RISC-V SoC Verification SystemVIP
The High Cost of Developing Test Content

Largest Functional Verification Challenge

- Creating Sufficient Tests to Verify the Design: 40%
- Knowing my Verification Coverage: 30%
- Managing the Verification Process: 20%
- Time to Isolate and Resolve a Bug: 15%
- Time to Discover the Next Bug: 10%
- Defining Appropriate Coverage Metrics: 5%
- Other: 5%

Source: Wilson Research 2020

Project Resource Deployment

- Verification: Debug 25%
- Verification: Content Development 30%
- Verification: Other 13%
- Design: 32%

Test development drives debug

Complex tests hard to get right

Can we abstract verification intent?

Can we re-use the same knowledge?

SOC Integration

Ad hoc verification methods

System Validation

Low coverage real workloads

Block UVM

Lack of abstraction & reuse

Why? Resource Intensive Test Content
Test Suite Synthesis... Analogous to Logic Synthesis

Design Synthesis

1. Specify goals
2. Generate implementation
3. Map to platform
4. Optimize

Test Suite Synthesis

1. Describe intent
2. Constrain
3. Synthesize
4. Optimize

Breker Core Technology

1. AI Planning Algorithms
2. 3D Coverage Closure
3. Synthesizable VerificationOS

Specifications

- Model
- Specification
- Coverage
- Debug
- SD
- Sys
- DC
- PP
- SiC
- Cam

Verification Tools

- UVM
- SoC
- Silicon

Timing/Area Constraints
SoC SystemVIP Library

- The **RISC-V Core TrekApp** provides fast, pre-packaged tests for RISC-V Core and SoC integrity issues
- The **Coherency TrekApp** verifies cache and system-level coherency in a multiprocessor SoC
- The **End-to-end IP TrekApp** IP test sets ported from UVM to SoC
- The **Power Management TrekApp** automates power domain switching verification
- The **Security TrekApp** automates testing of hardware access rules for HRoT fabrics
- The **Networking & Interface TrekApp** automates packet generation, CXL, UCIe interface tests
Constrained Random vs AI Planning Algorithm Synthesis

Constrained Random Generation
UVM SV & other PSS tools

Design black box, shotgun tests to search for key state
Low probability of finding complex bug

AI Planning Algorithm
Breker Test Suite Synthesis

Starts with key state and intelligently works backward through space
Deep sequential, optimized test discovers complex corner-cases

White Paper Discussing AI Planning Algorithm Test Generation on Breker Website
A Look At RISC-V

- Open Instruction Set Architecture (ISA) creating a discontinuity in the market
- Appears to be gaining significant traction in multiple applications
- Significant verification challenges
  - Arm spends $150M per year on $10^{15}$ verification cycles per core
  - Hard for RISC-V development group to achieve this same quality
  - Lots of applications expands verification requirements
  - Requires automation, reuse and other new thinking
RISC-V Verification & Validation Tasks

Core Integrity
- Interrupts/Paging/Memory Order
- Micro-architecture functionality
- ISA compliance
- First Instruction Completion

IP Integrity
- End-to-End Use Cases
- Concurrency Testing
- IP Configuration

SoC Integrity
- Power/Performance Profiling
- Security
- Power Management
- Interrupts/Paging/Memory Order
- Cache Coherency

Firmware Integrity
- End-to-End Use Cases
- Concurrency Testing
- HW/Firmware Compatibility

Complexity

© Breker Verification Systems, Inc. All rights reserved.
Breker Systems Confidential
Breker RISC-V SystemVIP Portfolio

SVIPs for Core Integrity
- Register Hazards
- Load/Store
- Core Cache Coherency
- Core Interrupts
- ...

SVIPs for IP Integrity
- Mem2Mem (dma)
- IO Offload (PCIE/Eth)
- WQ Servicing
- ...

SVIPs for SoC Integrity
- SoC Cache Coherency
- Memory Ordering
- Power Management
- System Interrupts
- ...

SVIPs for Firmware Integrity
- Mem2Mem (dma)
- IO Offload (PCIE/Eth)
- WQ Servicing
- ...

© Breker Verification Systems, Inc. All rights reserved. Breker Systems Confidential
Single Source of Truth for all stages of Verification & Validation

SVIPs for IP Integrity
- Mem2Mem (dma)
- IO Offload (PCIE/Eth)
- WQ Servicing
- ...

SVIPs for Core Integrity
- Register Hazards
- Load/Store
- Core Cache Coherency
- Core Interrupts
- ...

SVIPs for SoC Integrity
- SoC Cache Coherency
- Memory Ordering
- Power Management
- System Interrupts
- ...

SVIPs for FW Integrity
- Mem2Mem (dma)
- IO Offload (PCIE/Eth)
- WQ Servicing
- ...

Test Suite Synthesis

Virtual Platform Environment

UVM Block Environment

Simulation Acceleration

Hybrid Emulation Environment

Silicon / Prototyping Environment

© Breker Verification Systems, Inc. All rights reserved.
Different Challenges for Core vs SoC Verification

**RISC-V Core Verification Challenges**

- **Random Instructions**: Do instructions yield correct results
- **Register/Register Hazards**: Pipeline perturbations due to register conflicts
- **Load/Store Integrity**: Memory conflict patterns
- **Conditionals and Branches**: Pipeline perturbations from synchronous PC change
- **Exceptions**: Jumping to and returning from ISR
- **Asynchronous Interrupts**: Pipeline perturbations from asynchronous PC change
- **Privilege Level Switching**: Context switching
- **Core Security**: Register and Memory protection by privilege level
- **Core Paging/MMU**: Memory virtualization and TLB operation
- **Sleep/Wakeup**: State retention across WFI
- **Voltage/Freq Scaling**: Operation at different clock ratios
- **Core Coherency**: Caches, evictions and snoops

**RISC-V SoC Verification Challenges**

- **System Coherency**: Cover all cache transitions, evictions, snoops
- **System Paging/IOMMU**: System memory virtualization
- **System Security**: Register and Memory protection across system
- **Power Management**: System wide sleep/wakeup and voltage/freq scaling
- **Packet Generation**: Generating networking packets for I/O testing
- **Interface Testing**: Analyzing coherent interfaces including CXL & UCIe
- **Random Memory Tests**: Test Cores/Fabrics/Memory controllers across DDR, OCRAM, FLASH etc
- **Random Register Tests**: Read/write test to all uncore registers
- **System Interrupts**: Randomized interrupts through CLINT
- **Multi-core Execution**: Concurrent operations on fabric and memory
- **Memory Ordering**: For weakly order memory protocols
- **Atomic Operation**: Across all memory types
Agenda

- Test Suite Synthesis and SystemVIP
- RISC-V Core Verification SystemVIP
- RISC-V SoC Verification SystemVIP
Random register instructions
Instruction Coverage Analysis

27/103 reachable opcode have been exercised

Atoms, loads and stores not reachable in register only test
Locality of write addrs
Example Address Allocation Patterns

- Random Clusters with locality of reference
  ```
  // memAllocAddrSlice allocated setId:0x1 of 0x4 blocks
  // memAllocAddrRand size:0x8 addr: trek_mem_ddr+0x08b810c8
  // memAllocAddrRand size:0x8 addr: trek_mem_ddr+0x2380e378
  // memAllocAddrRand size:0x8 addr: trek_mem_ddr+0x2380e380
  // memAllocAddrRand size:0x8 addr: trek_mem_ddr+0x2380e370
  ```

- Stride Patterns across fixed address distances
  ```
  // memAllocAddrSlice allocated setId:0x1 of 0x4 blocks
  // memAllocAddrStride stride_len:0x2000 size:0x8 addr: trek_mem_ddr+0x08b830c8
  // memAllocAddrStride stride_len:0x2000 size:0x8 addr: trek_mem_ddr+0x08b850c8
  // memAllocAddrStride stride_len:0x2000 size:0x8 addr: trek_mem_ddr+0x08b870c8
  // memAllocAddrStride stride_len:0x2000 size:0x8 addr: trek_mem_ddr+0x08b890c8
  ```

- Sequential Addresses matching a specific Hash
  ```
  // memAllocAddrSlice allocated setId:0x1 of 0x4 blocks
  // memAllocAddrHash hash:0x44 size:0x100 addr: trek_mem_ddr+0x08bc1100
  // memAllocAddrHash hash:0x44 size:0x100 addr: trek_mem_ddr+0x08c01100
  // memAllocAddrHash hash:0x44 size:0x100 addr: trek_mem_ddr+0x08c41100
  // memAllocAddrHash hash:0x44 size:0x100 addr: trek_mem_ddr+0x08c81100
  ```
Application to Unit Bench and Sub-System Bench
RV64 Core Exception Testing

Generates for example, `asm("UNIMP");`

Check exception counts
Page Based Virtual Memory Tests

Figure 3.2: RISC-V address translation details.
RV64 Core Page Based MMU Tests

Swap MMU PTE’s and Check memory access

```c
// swapOne.2

trek_c2t_event(0, 0x38);  // {event:0x38 agent:hart0 thread:T0 instance:s
/* tbx: trek_message("Begin swapOne.2."); */
/* swapping Pages: trek_mem_ddr+0x0d0a0000 and trek_mem_ddr+0x0d090000 */
const trek_uint64_t addrA = trek_mem_ddr+0x0d0a0000ULL;
const trek_uint64_t addrB = trek_mem_ddr+0x0d090000ULL;
// Find table entries for each address.
trek_uint64_t* const pte1 = trek_find_pte(addrA);
trek_uint64_t* const pte2 = trek_find_pte(addrB);
const trek_uint64_t entry1 = *pte1;
const trek_uint64_t entry2 = *pte2;
// Insert the new table entries with addrA and addrB swapped.
*pte1 = entry2;
*pte2 = entry1;
trek_c2t_event(0, 0x39);  // {event:0x39 agent:hart0 thread:T0 instance:s
/* tbx: trek_message("End swapOne.2."); */
trek_write32_shared(0x17, trek_hart0_T0_state);
brk_break;
}
```
Agenda

- Test Suite Synthesis and SystemVIP
- RISC-V Core Verification SystemVIP
- RISC-V SoC Verification SystemVIP
RISC-V SoC Testbench Integration

Scenario Model

Virtualized OS Services

test.c

Compiler

test.tbxml

TrekBox

Cache-Coherent Switching Fabric

L3 Cache / Snoop Filter

Memory Controller

Memory Controller

PCIE

Ethernet

Offload

PCIE VIP

Ethernet VIP
Multi-Agent Scheduling Plans: Overview

- True Sharing within scenario
- False Sharing across scenarios

N Transition Sequences

Concurrent Scenario Test Case

Schedule Memory
Interleave & Pack
Resolve Dependencies
RV64 MultiCore MoesiStates

Planned Cache State Transitions
Efficacy of System-Integrity Testing using the RISC-V TrekApp

Typical directed coherency test ...

... vs. RISC-V TrekApp automated Sys-Integrity tests
Check result is aggregate of synchronized atomic operations
RISC-V SoC Memory Ordering: Dekker Algorithm

- Assume initial state $A=0$, $B=0$

- The Dekker Algorithm States
  
  core 0: ST A, 1; MEM_BARRIER; LD B
  core 1: ST B, 1; MEM_BARRIER; LD A
  
  error iff $(A == 0 \&\& B == 0)$

- This is a test for a weakly ordered memory system
  
  - Such a system must preserve the property that a LD may not reorder ahead of a previous ST from the same agent
Check ordering across synchronized Dekker scenarios
MultiCore MMU Tests

All cores Swap MMU PTE’s and check memory access
False-Share Memory Stress Tests

Allocate set of memory blocks

Each core operates on a “slice” of memory

Random cores with synchronized start

Each core has free running loop

```c
void * addr[] = {
    (void *)(trek_mem_ddr+0x08b830ca),
    (void *)(trek_mem_ddr+0x08b850ca),
    (void *)(trek_mem_ddr+0x08b870ca),
    (void *)(trek_mem_ddr+0x08b890ca),
};

for (int ii = 0; ii < 1000; ++ii) {
    errorCount += trek_microloop_write_check8(addr[4], 184);
}
```
Thanks for Listening!
Any Questions?