Security Verification Using Portable Stimulus Driven Test Suite Synthesis

Adnan Hamid, Breker Verification Systems
David Kelf, Breker Verification Systems
Agenda

• HW Security Fundamentals
• Security Verification
• Portable Stimulus Test Suite Synthesis
  – Why is it good for Security Verification
• A Breker End-user Security Methodology
• Security Verification Demonstration
• Q&A

Note Code Examples to Follow
Hardware Security Fundamentals
Security Is Important

- General cloud server data
- Bitcoin encrypted keys
- Personal computer data
- Automotive control takeover
- Election database tampering
Hardware Root of Trust (HRoT)

- Security vulnerabilities exist throughout the system stack
  - However, lower vulnerabilities in the stack have greater influence
- If the physical hardware is hacked it can corrupt the entire system
  - Requires a high degree of “trust”
- The Hardware Root of Trust (HRoT) provides HW security capabilities, e.g.
  - A Trusted Execution Environment (TEE)
  - Cryptography, where necessary
  - Protection against specific threats
  - A trusted control mechanism
Example Hardware Vulnerabilities

- Access through unusual ports
- Third-party IP malware
- Boot time malware insertion
- Access to sensitive memory area
- FPGA or firmware trojans
- Third-party IP malware
- Unexpected key access
- Uncontrolled register access
- Side-channel power decode
Protected Regions & Registers

Focus of this workshop. Eliminates many vulnerabilities

Protected region in memory/peripherals controlled by specialized µP functionality

Protected area on SoC locked through fabric access rules and/or security monitor
Example: ARM TrustZone

- ARM technology built into most of the newer architectures
- Creates a secure HW zone (Trusted Execution Environment) that runs a secure kernel for a software protected VM
- The non-secure zone runs concurrently, with very limited access into the secure zone
- Secure zone includes protected memory regions which can hold secure assets e.g. protected keys
Protecting Regions

- Many HRoT solutions come down to HW access rules on the SoC fabric
- A Security Policy Control Device (SPCD) is included as part of the fabric and provides access control
- The SPCD interfaces with IP block security wrappers that handle local events, based on firmware
- This provides a highly flexible SoC architecture, while maintaining security on the most complex of devices

Protecting Regions

A Breker customer uses this methodology, with a multi-stage rule based solution built into their fabric.

We will use this approach in our demonstration.
Security Verification
Security: A “Negative” Verification Problem

• Using example:
  Assume key should only be accessible through Crypto engine
• Positive test would be to ensure key can be read in this manner
• Negative test would say “Is there any other way a key can be accessed in this device”
• Tests for the negative case requires exploration of a broad state-space
A manually composed testbench (UVM, C-code, etc.) requires a prediction of all the possible corner-cases where a vulnerability could be exposed at the SoC implementation level.

This is extremely error-prone and requires hacker expertise.
Formal Verification for Block Level Test

Formal Verification is good at exploring all the states in a block to perform negative testing.

A design transitions between many states during operation.

Formal checks all possible states over set time to exhaustively test properties.

Simulation checks one sequence of states driven by stimulus to replicate operation.

Prove the property “there is no other way to access key register X except through port A”
For an SoC the state space to be search explodes, quickly hitting formal verification capacity limits.

We could constrain the formal tool to only check certain code segments over a specific cycle number, but then we run the risk of missing something, particularly easy in security.
A “Semi-Formal” Look At System Security

What if we can define an abstract, hierarchical intent state-space and then walk through it to extract all required test-cases?

Each action is hierarchical. Complete test includes lowest level leaf actions.
PSS Test Suite Synthesis
(and why is it good for security verification)
Changing the Verification Content Perspective

Design Synthesis

- Specify goals
- Describe intent
- Generate implementation
- For existing environments
- Constraints
- Synthesis
- Optimization

Test Suite Synthesis

- Constraints
- Synthesis
- Optimization

UVM
Software Driven
Post-Silicon & Prototype
Block Testbench
Testbench
Prototype

Constraints
Synthesis
Coverage
DUT
Optimization
Design
Constraints
Specify goals
Describe intent
Generate implementation
For existing environments

UVM Block Testbench
Software Driven Testbench
Post-Silicon & Prototype
AI Planning Algorithms for Test Generation

Example: Simple Car Operational Scenario

Input constraints to test “forward” outcome

Input constraints to test “stop” outcome

Three Types of Goals
- Sequence Goal
- Select Goal
- Leaf Goal
Digital Camera Application: Single Test Example

Breker path constraint: \texttt{PP.quality == SD.quality}
Digital Camera Application: Multiple tests

Set coverage goals as appropriate to match verification requirement.

Perform pre-synthesis analysis before test generation and understand coverage.

Set coverage goals as appropriate to match verification requirement.
Synthesizing Multi-threaded Test Suite with Resources

- Breker scheduling synthesis interleaves tests across resources
- Synchronized transactions (including UVM) and C-tests during execution
- Multi-memory scheduling and allocation uncovers complex SoC bugs
Packaging Automated Test Content

The Breker TrekApps

- The **ARMv8 TrekApp** automated integration testing of ARM based systems.
- The **RISC-V TrekApp** automated integration testing of RISC-V based systems.
- The **Cache Coherency TrekApp** automated integration testing of generic CPU based systems.
- The **Power Management TrekApp** automates power domain switching verification
- The **Security TrekApp** automates testing of access control rules
- The **Networking TrekApp** automates packet generation, dissection and prediction.
Test Case Optimization Across the Flow

**UVM test content synthesis**
Complex sequence, coverage scoreboard synthesis

**Software-driven SoC test**
Rigorous, high-coverage tests from a single, simple specification

**Prototyping & silicon diagnostics**
Verification test reuse with observability/controllability

**Configuration Layer**
PS Standard Model
Virtual Realization Layer

**Virtual Realization Layer**

**Test Case**
Optimization Across the Flow

**Scenario**
Model
SD
Sys
DC
PP
Cam
PS
Standard
Model

**DUT**
UVM
Agent

**UVM**

test.c
test.tbx

**Cache-Coherent Fabric**
CPU
L1
L1
L1
L2
L3
I/O

**Virtual Realization Layer**

**Configuration Layer**

**VIP**

**Software-driven SoC test**
Rigorous, high-coverage tests from a single, simple specification

**Prototyping & silicon diagnostics**
Verification test reuse with observability/controllability
Breker End-user Proven Security Methodology

*Note Code Examples to Follow*
Objective: Multi-level Security Checking

Stage-1 Check
Stage-2 Check
Stage-N Check

Stage-1 Security Rules Table
Stage-2 Security Rules Table
Stage-N Security Rules Table

Error
Error
Error
Tables Used To Define Security Policy

<table>
<thead>
<tr>
<th>Master/Slave Check</th>
<th>OK</th>
<th>sec/priv check</th>
<th>OK</th>
<th>Slave Addr</th>
<th>UVM TLM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td></td>
<td>Error</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Required, a complete state space exploration of the table combination

trek_cfg_add_memory_region "mr_CCU_IOM"
trek_cfg_add_memory_region "mr_TCU"
trek_cfg_add_memory_region "mr_L4_AHB"

trek_cfg_set_memory_region_property "mr_CCU_IOM" "base_address" 0x01000000
trek_cfg_set_memory_region_property "mr_TCU" "base_address" 0x02000000
trek_cfg_set_memory_region_property "mr_L4_AHB" "base_address" 0x03000000

trek_cfg_set_memory_region_property "mr_CCU_IOM" "bytes_available" 0x10000
 trek_cfg_set_memory_region_property "mr_TCU" "bytes_available" 0x10000
 trek_cfg_set_memory_region_property "mr_L4_AHB" "bytes_available" 0x10000

<table>
<thead>
<tr>
<th>Master</th>
<th>CCU_IOM</th>
<th>TCU</th>
<th>L4_AHB</th>
<th>L4_MAIN</th>
<th>L4_MNP</th>
<th>L4_SP</th>
<th>L4_SYS</th>
<th>S4_SYS_Q1</th>
<th>L4_EC</th>
</tr>
</thead>
<tbody>
<tr>
<td>AXI_AP</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CCU_IOM</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DMA_TBU</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EMAC_TBU</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IO_TBU</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SDM_TBU</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table</th>
<th>TenKilowedge</th>
<th>Slaterounds</th>
<th>SlaterPrivledge</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>sec</td>
<td>priv</td>
<td>sec</td>
<td>non_priv</td>
<td>1</td>
</tr>
<tr>
<td>sec</td>
<td>priv</td>
<td>sec</td>
<td>non_priv</td>
<td>0</td>
</tr>
<tr>
<td>sec</td>
<td>priv</td>
<td>non_sec</td>
<td>non_priv</td>
<td>0</td>
</tr>
<tr>
<td>sec</td>
<td>non_sec</td>
<td>non_sec</td>
<td>non_priv</td>
<td>0</td>
</tr>
<tr>
<td>sec</td>
<td>non_sec</td>
<td>non_sec</td>
<td>non_priv</td>
<td>0</td>
</tr>
<tr>
<td>sec</td>
<td>non_sec</td>
<td>sec</td>
<td>non_priv</td>
<td>0</td>
</tr>
<tr>
<td>sec</td>
<td>non_sec</td>
<td>sec</td>
<td>non_sec</td>
<td>0</td>
</tr>
<tr>
<td>sec</td>
<td>non_sec</td>
<td>sec</td>
<td>non_sec</td>
<td>0</td>
</tr>
<tr>
<td>sec</td>
<td>non_sec</td>
<td>sec</td>
<td>non_sec</td>
<td>0</td>
</tr>
<tr>
<td>sec</td>
<td>non_sec</td>
<td>sec</td>
<td>non_sec</td>
<td>0</td>
</tr>
</tbody>
</table>

27
Need to Generate Intent State-Space Graph

- Leveraging CSV to SQL translation
- `table2graph` utility reads SQL tables and generates self-checking intent graph
- Test Suite Synthesis Graph explodes graph for complete state-space analysis
- State space analysis used to set coverage goals pre-test generation to generate fabric test content

Graphs Auto-Generated from Table
Synthesized Fabric Test Suite
Applied to UVM Fabric Testbench
Security Verification Demonstration

Note Code Examples to Follow
Thank You For Listening

For more information
Please come by the Breker booth, #701
Or go to BrekerSystems.com