2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Security Verification Using Portable Stimulus Driven Test Suite Synthesis

Adnan Hamid, Breker Verification Systems

David Kelf, Breker Verification Systems

BREKER™

accellera SYSTEMS INITIATIVE

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

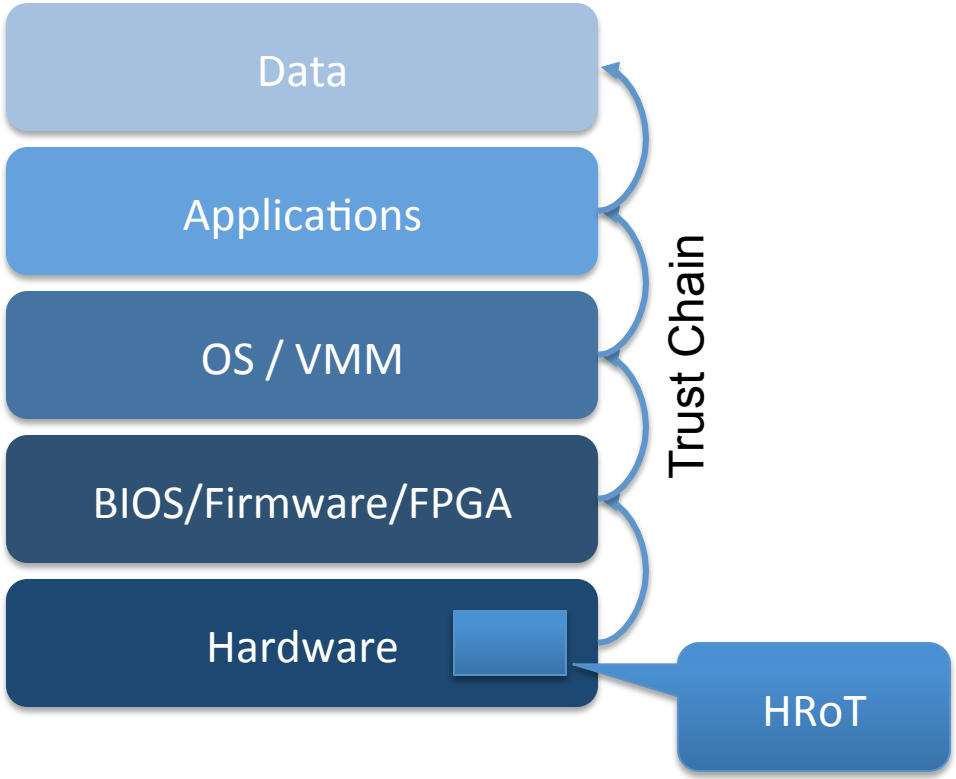
Bitcoin encrypted keys

Automotive control takeover

Personal computer data

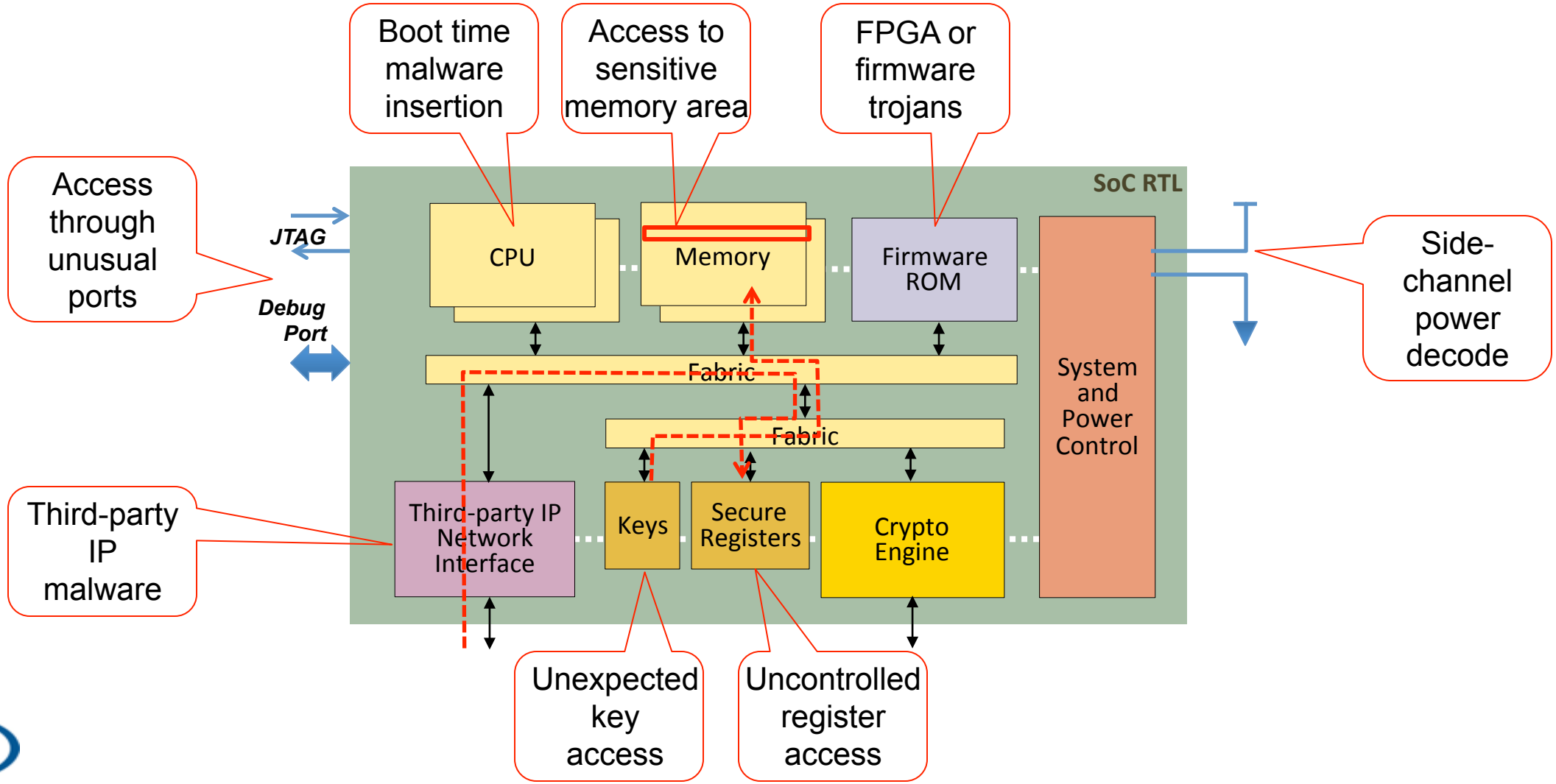
Election database tampering

Hardware Root of Trust (HRoT)



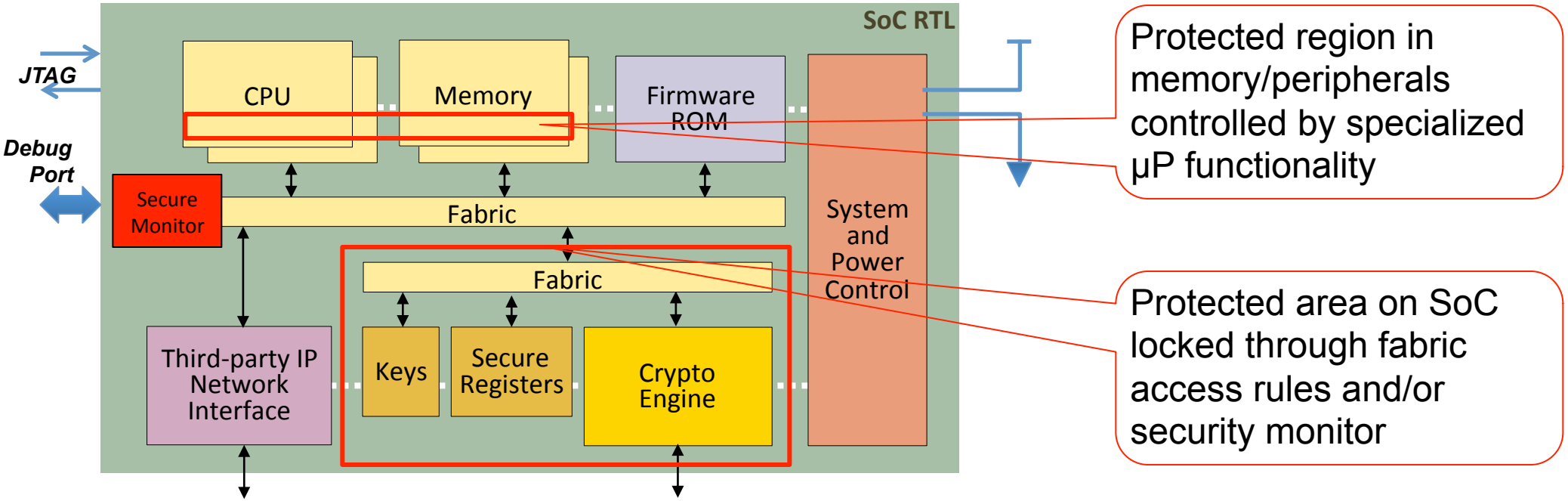
- Security vulnerabilities exist throughout the system stack
  - However, lower vulnerabilities in the stack have greater influence
- If the physical hardware is hacked it can corrupt the entire system
  - Requires a high degree of "trust"
- The Hardware Root of Trust (HRoT) provides HW security capabilities,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

Example Hardware Vulnerabilities

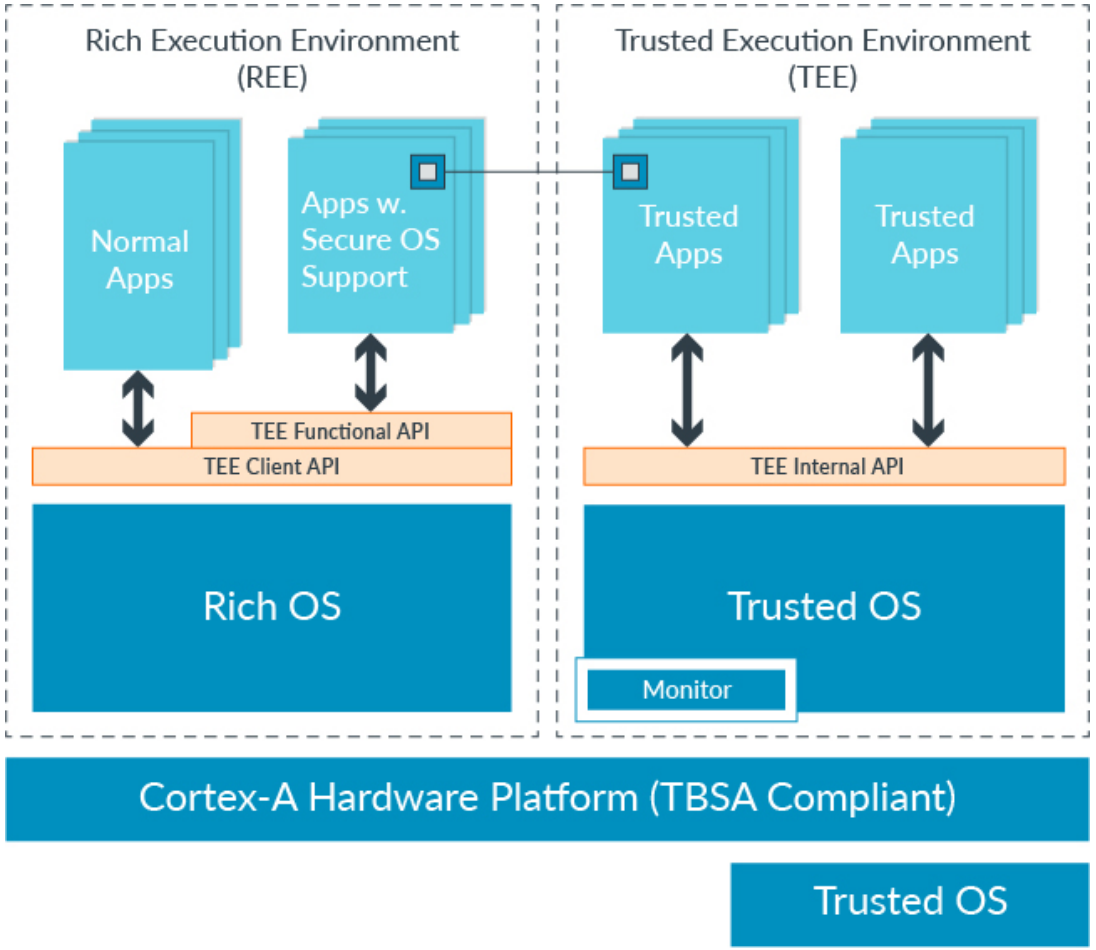


Protected Regions & Registers

Focus of this workshop. Eliminates many vulnerabilities

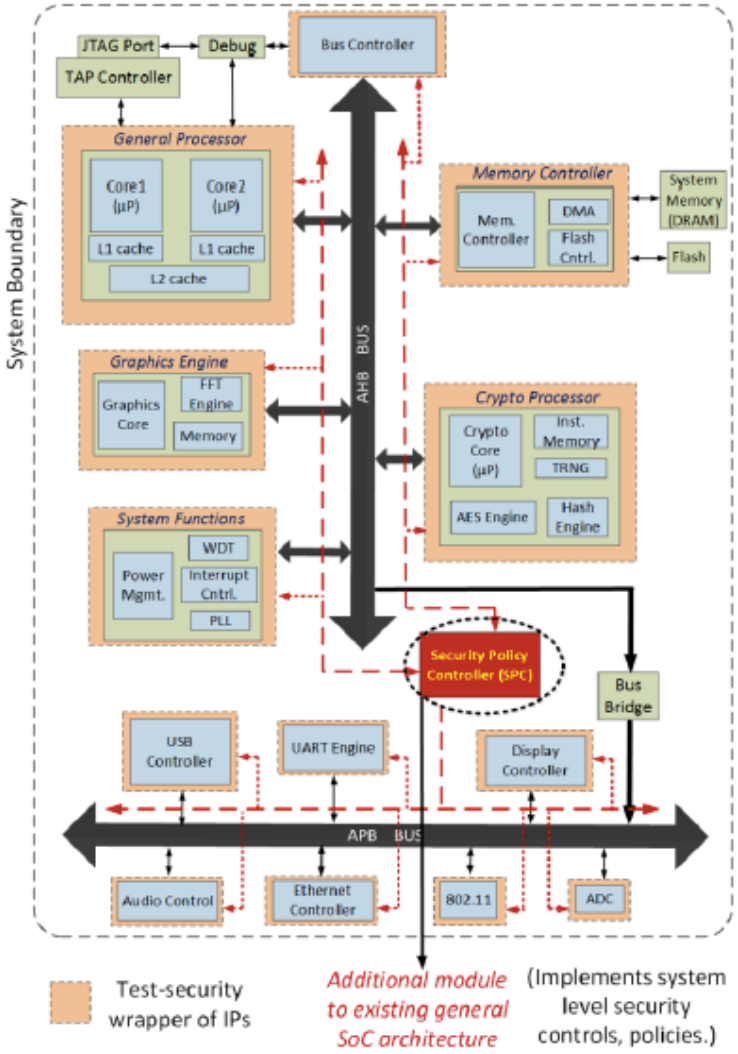


Example: ARM TrustZone



- 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

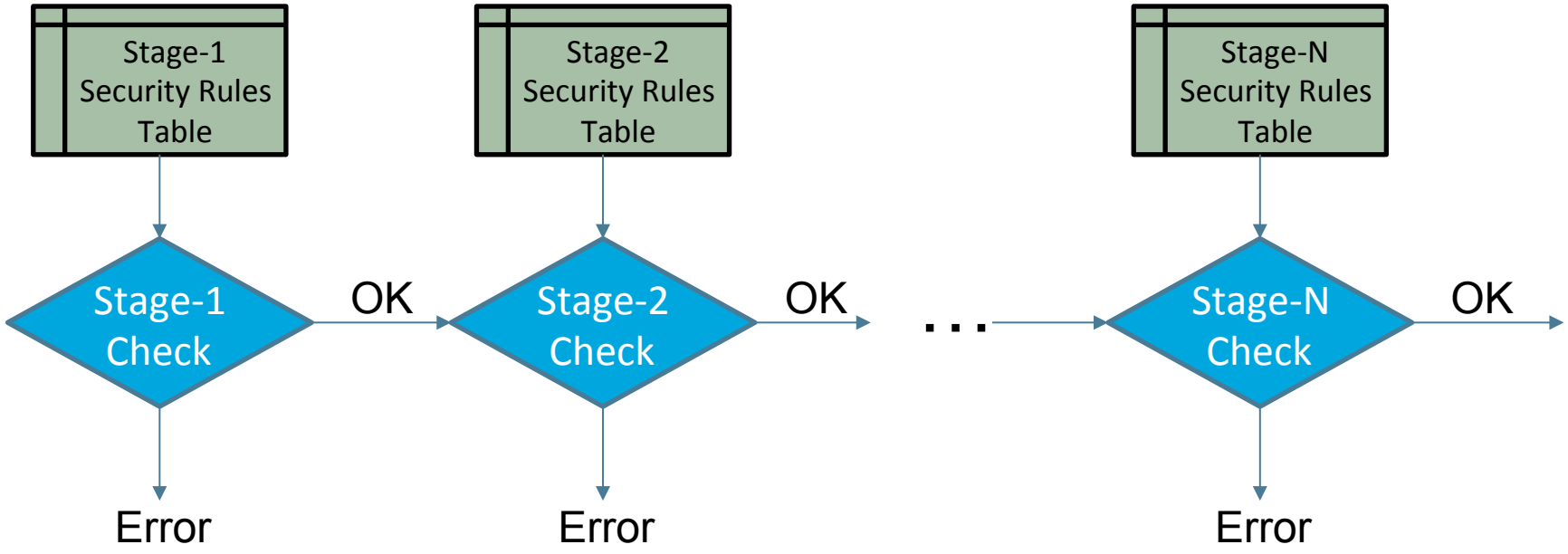


- Many HRoT solutions come down to HW access rules on the SoC fabric
- A Security Policy Control Device (SPCD) is included as part of the fabric and provides access control
- The SPCD interfaces with IP block security wrappers that handle local events, based on firmware
- This provides a highly flexible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

Source: System-on-Chip Platform Security Assurance: Architecture and Validation, Ray et al, IEEE

Protecting Regions

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

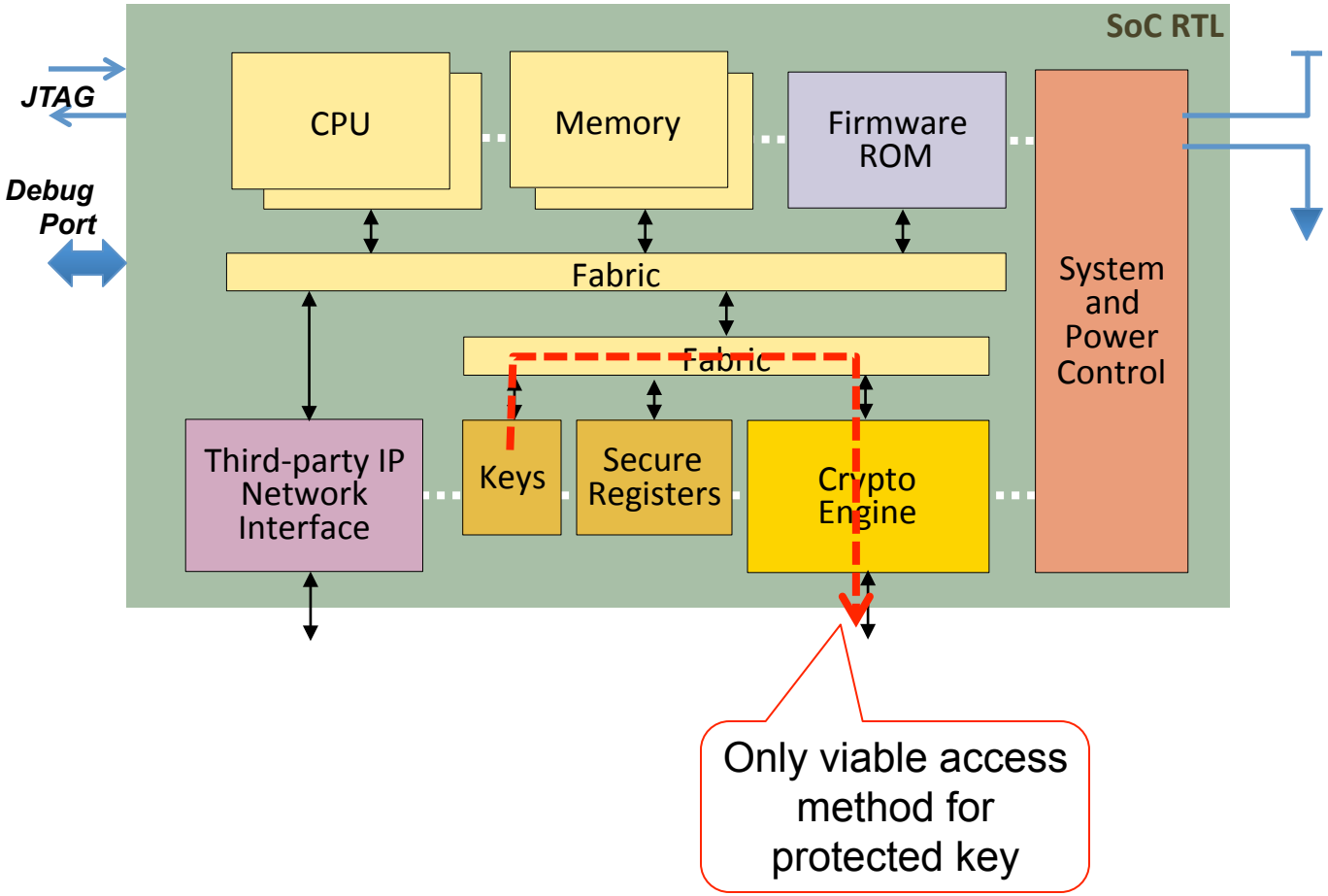


We will use this approach in our demonstration

Security Verification

Security: A "Negative" Verification Problem

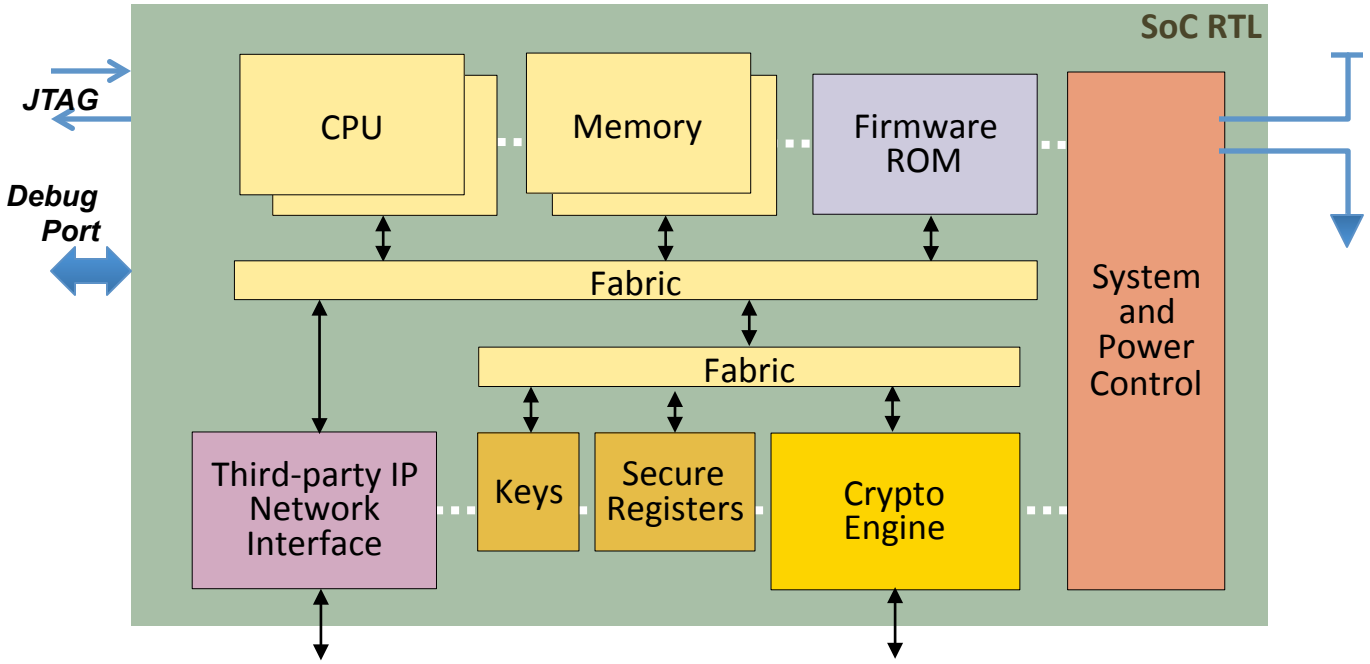
- Using example:
  Assume key should only be accessible through Crypto engine
- Positive test would be to ensure key can be read in this manner
- Negative test would say
  "Is there any other way a key can be accessed in this device"
- Tests for the negative case requires exploration of a broad state-space



Manual Testing of System Security

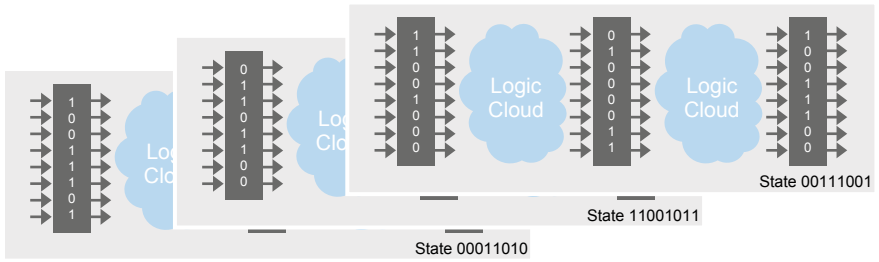
A manually composed testbench (UVM, C-code, etc.) requires a prediction of all the possible corner-cases where a vulnerability could be exposed at the SoC implementation level.

This is extremely error-prone and requires hacker expertise.

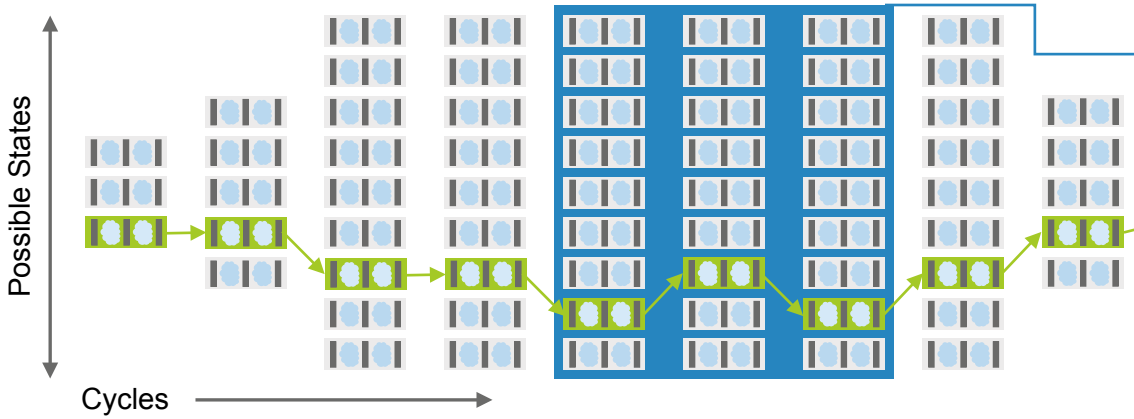


Formal Verification for Block Level Test

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

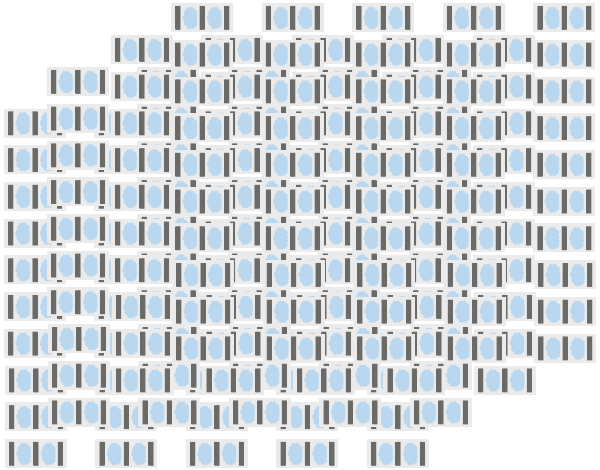
Simulation checks one sequence of states driven by stimulus to replicate operation

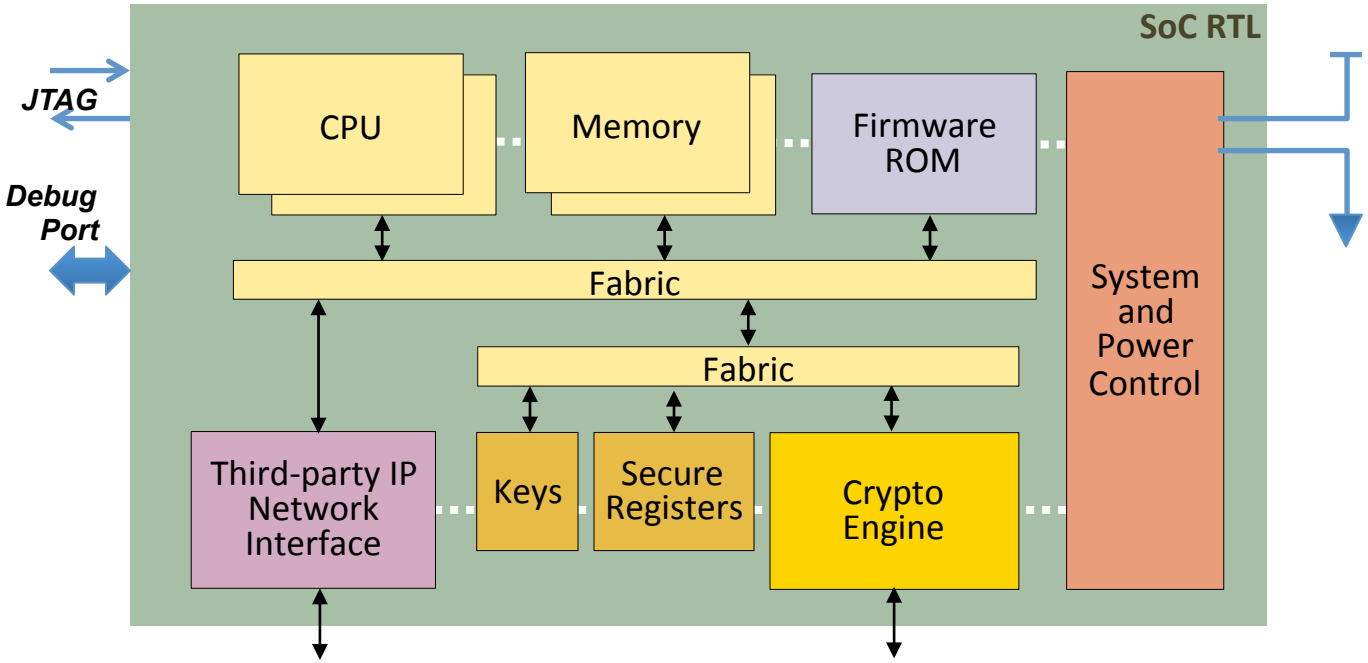
Diagrams courtesy OneSpin Solutions

Prove the property "there is no other way to access key register X except through port A"

Formal Capacity Limits for System-level Tests

For an SoC the state space to be search explodes, quickly hitting formal verification capacity limits

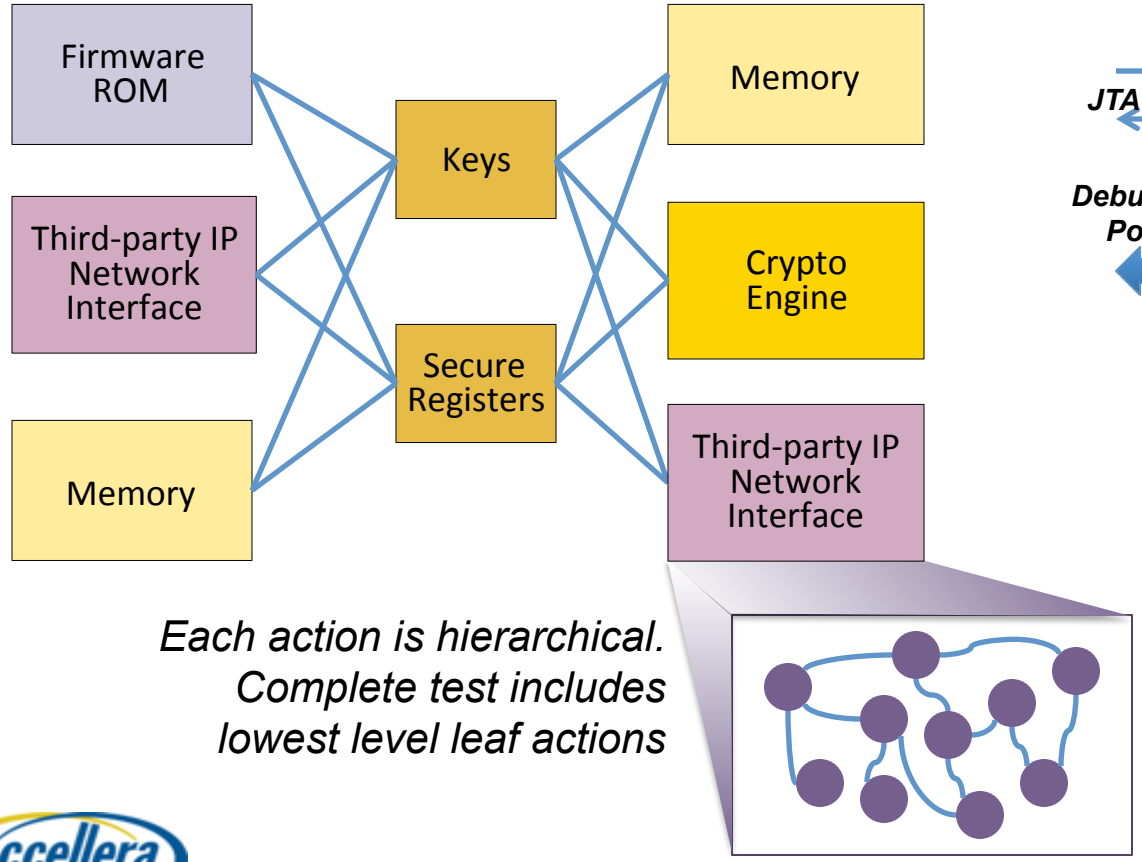




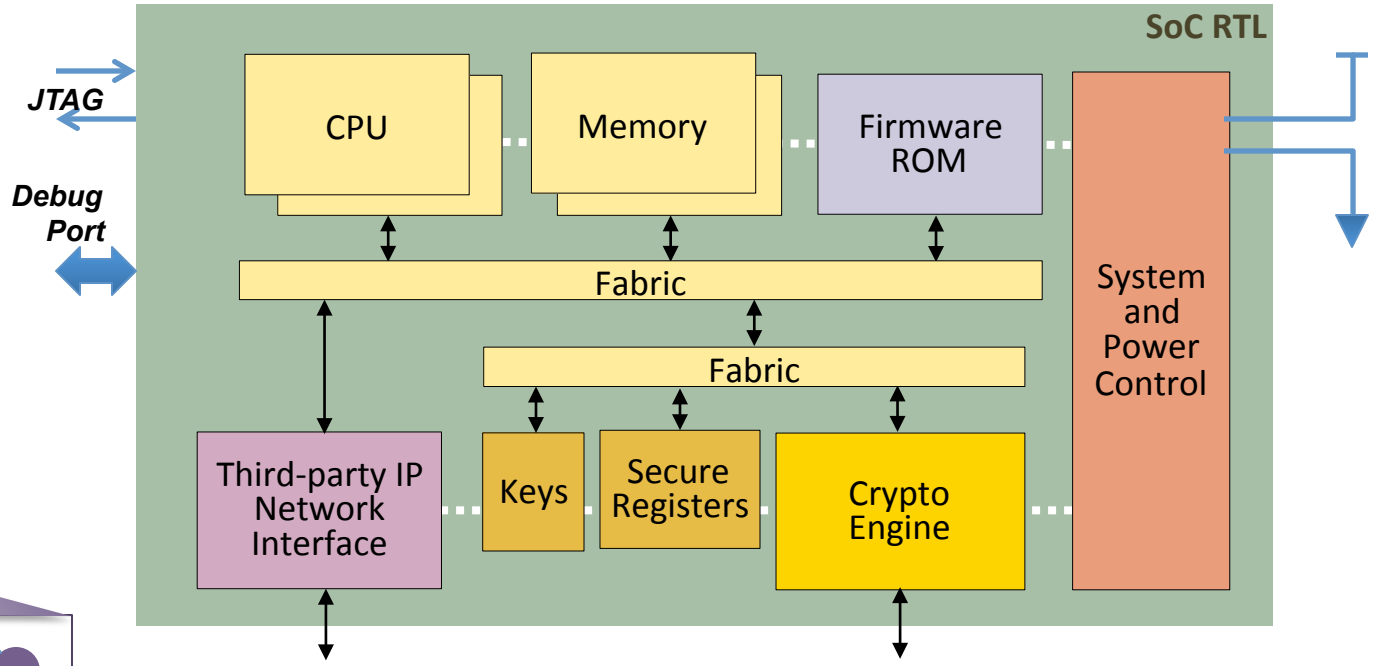
We could constrain the formal tool to only check certain code segments over a specific cycle number, but then we run the risk of missing something, particularly easy in security

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?

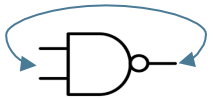
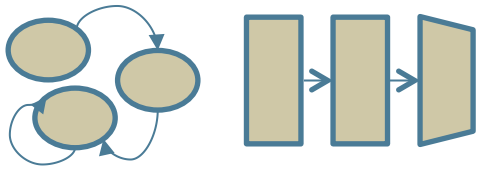
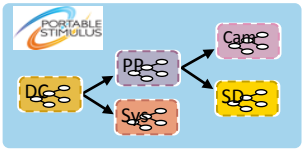
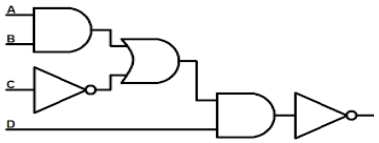
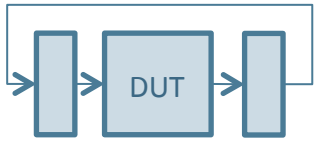
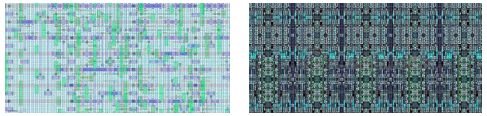


Each action is hierarchical. Complete test includes lowest level leaf actions



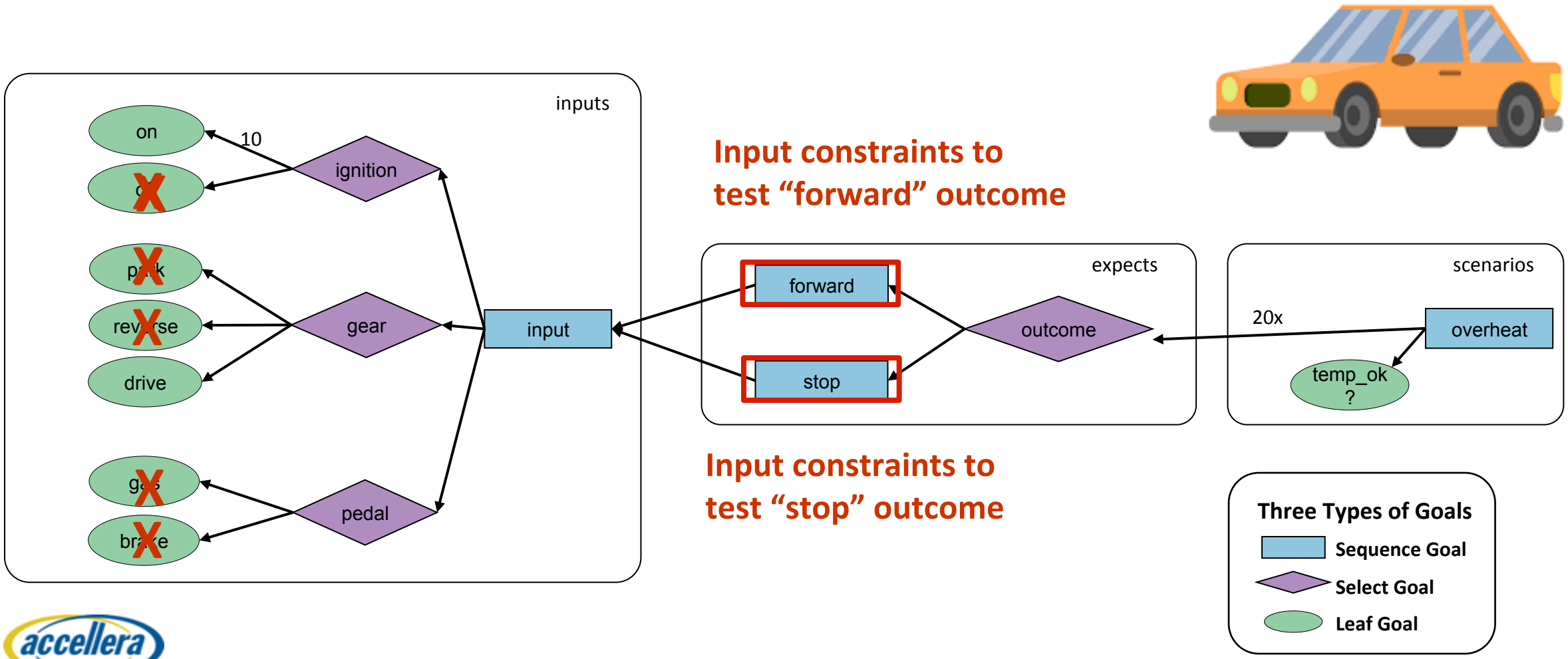
PSS Test Suite Synthesis
(and why is it good for security verification)

Changing the Verification Content Perspective

Design Synthesis		Test Suite Synthesis
 Design Constraints	Specify goals	 Coverage Constraints
Constraints		Constraints
	Describe intent	
Synthesis		Synthesis
	Generate implementation	
Optimization		Optimization
	For existing environments	UVM Block Testbench Software Driven Testbench Post-Silicon & Prototype

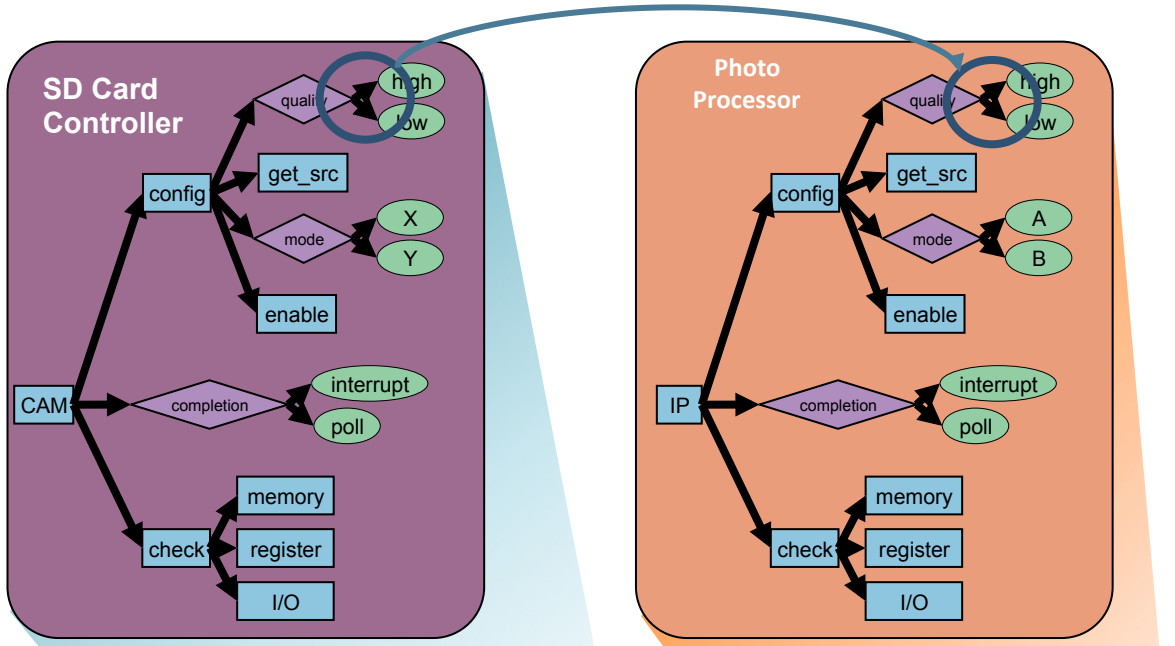
AI Planning Algorithms for Test Generation

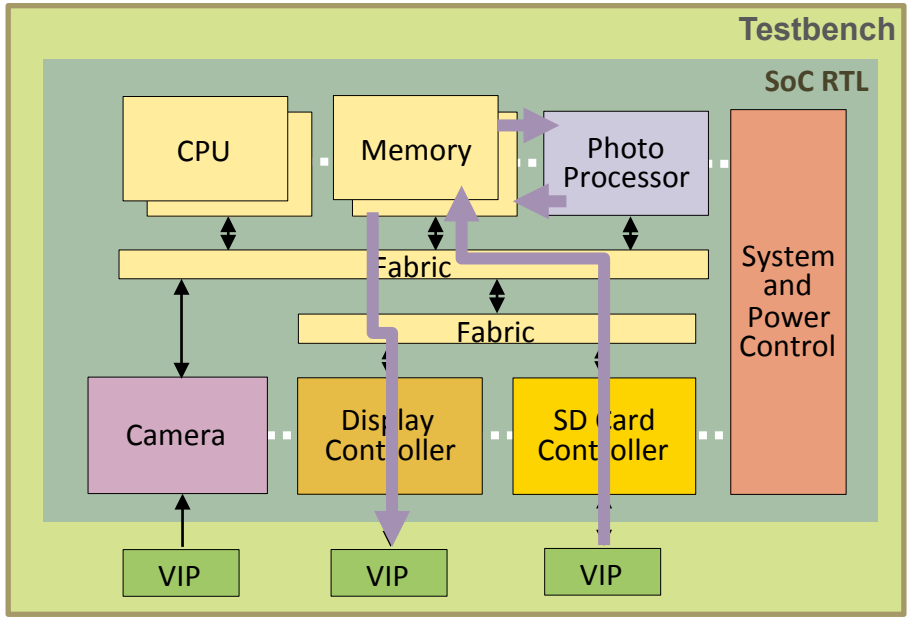
Example: Simple Car Operational Scenario

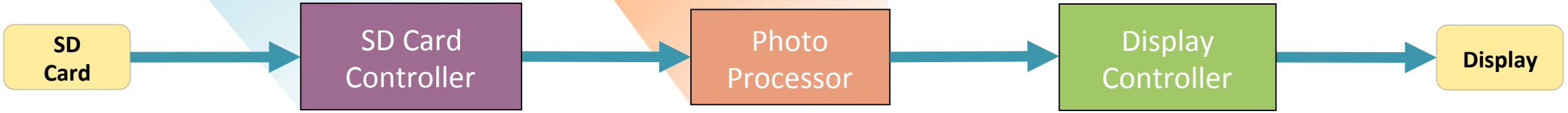


Digital Camera Application: Single Test Example

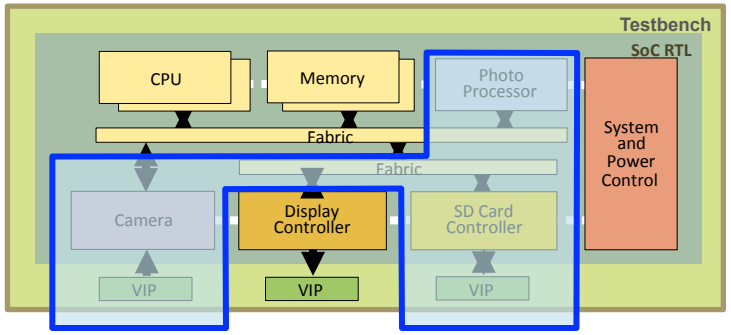
Breker path constraint: **PP.quality == SD.quality**

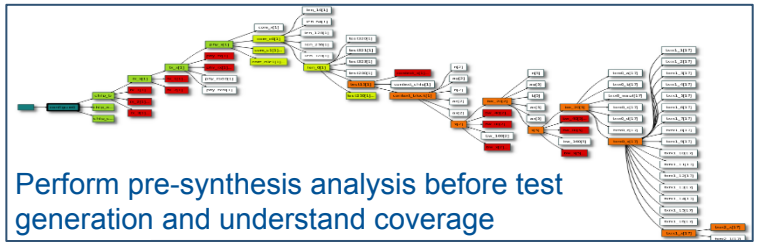


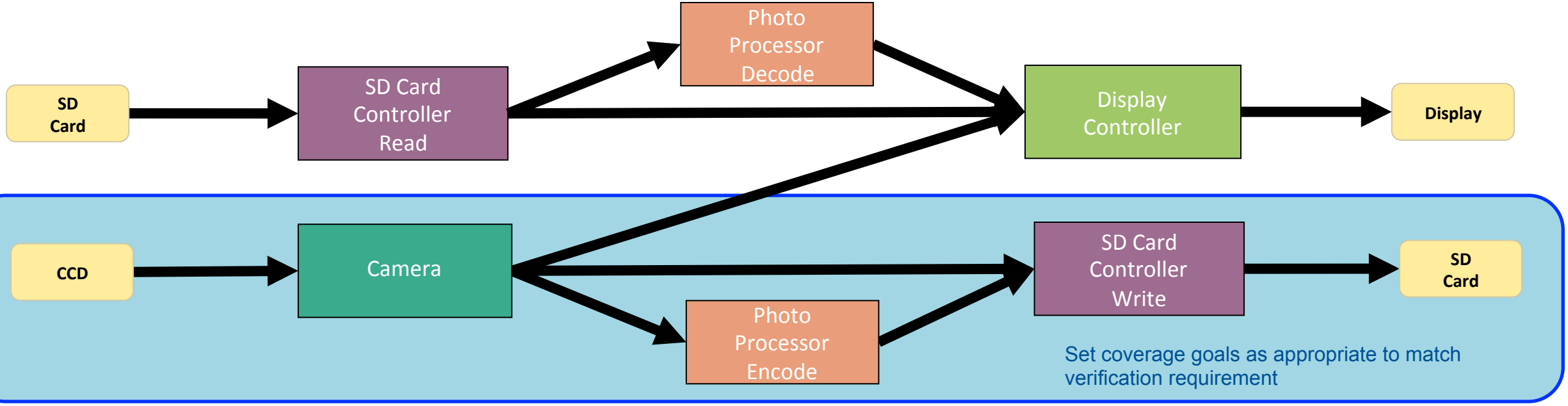




Digital Camera Application: Multiple tests

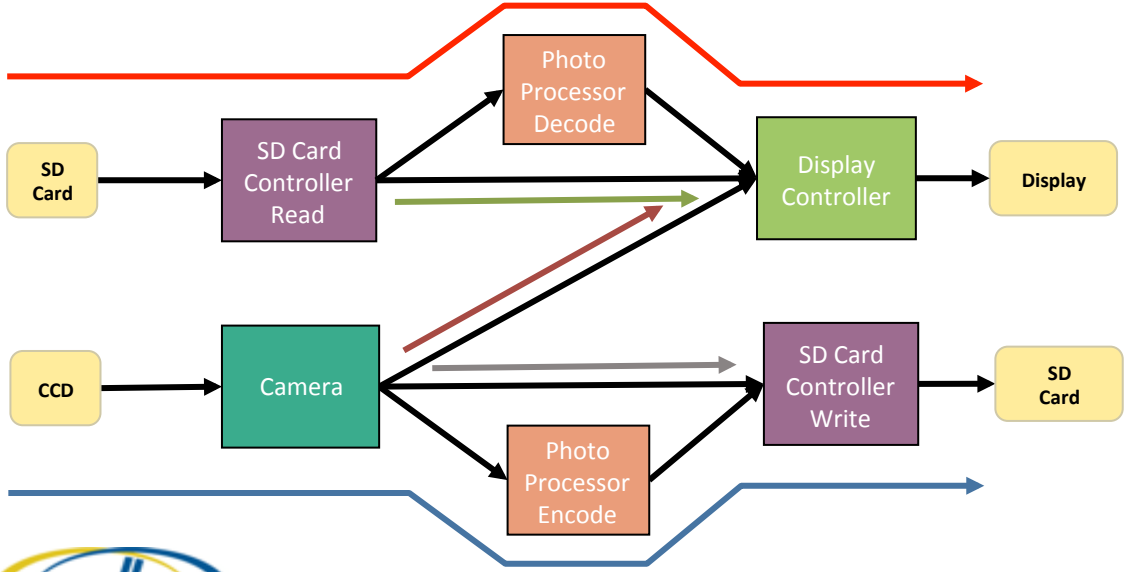


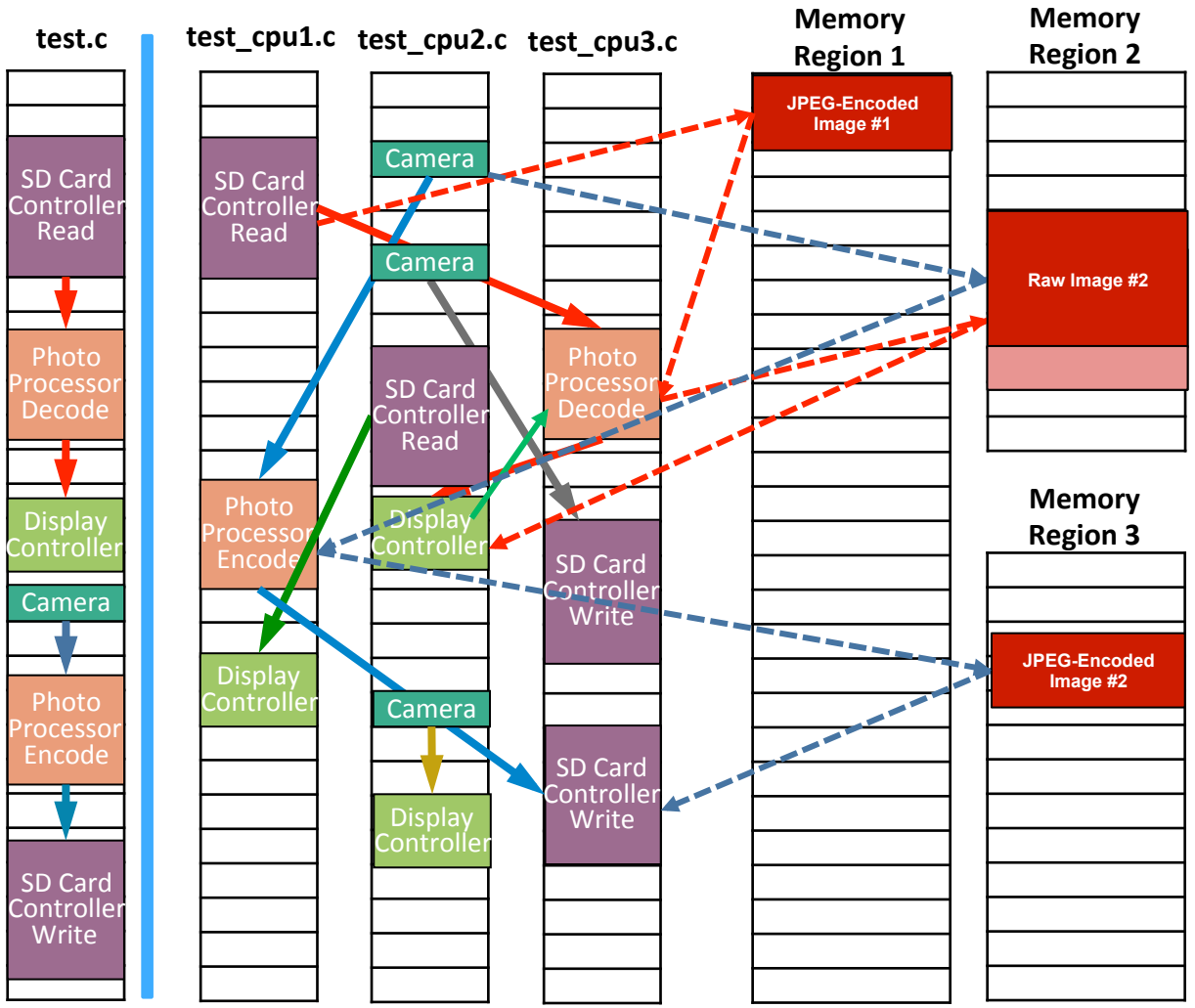




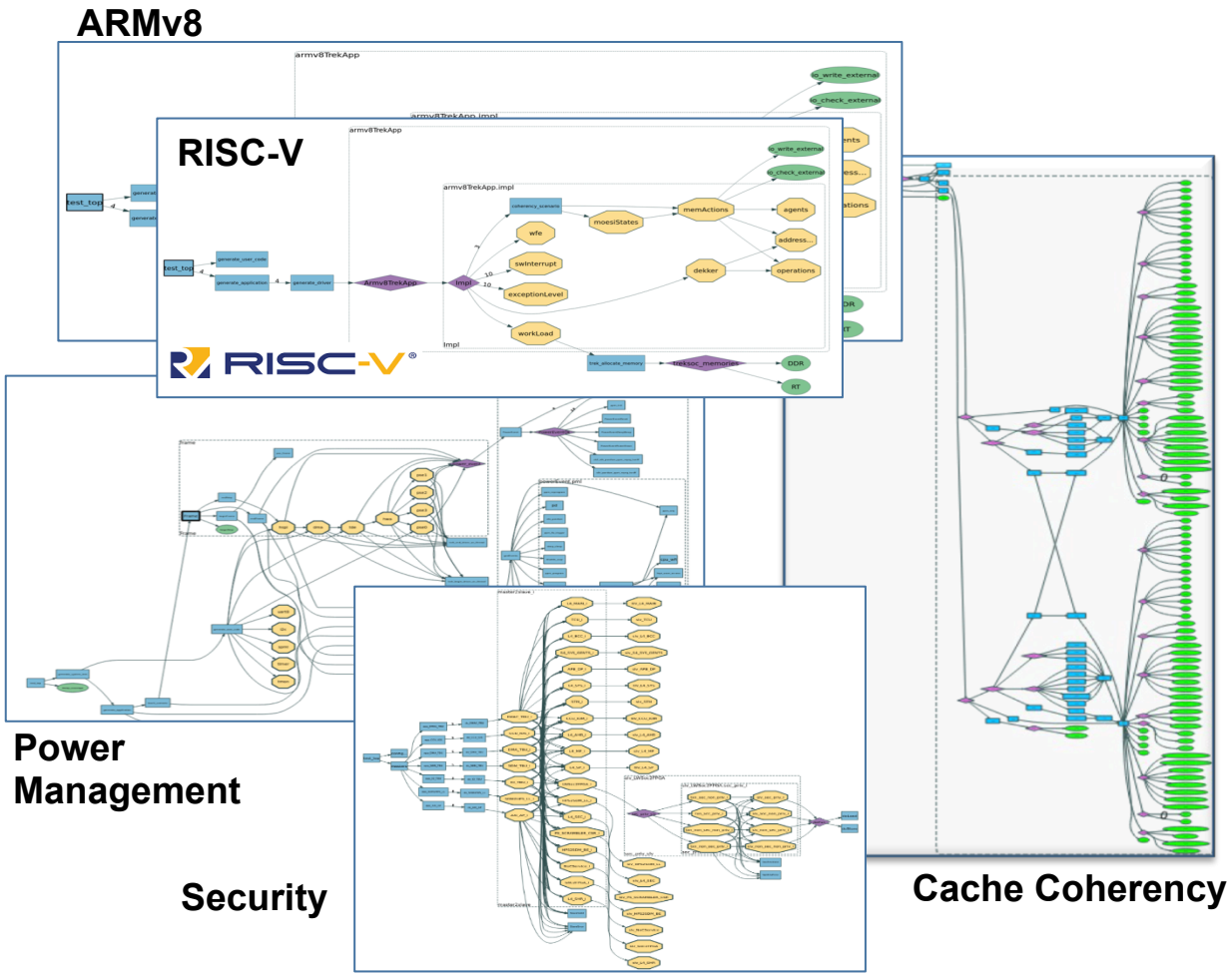
Synthesizing Multi-threaded Test Suite with Resources

- 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





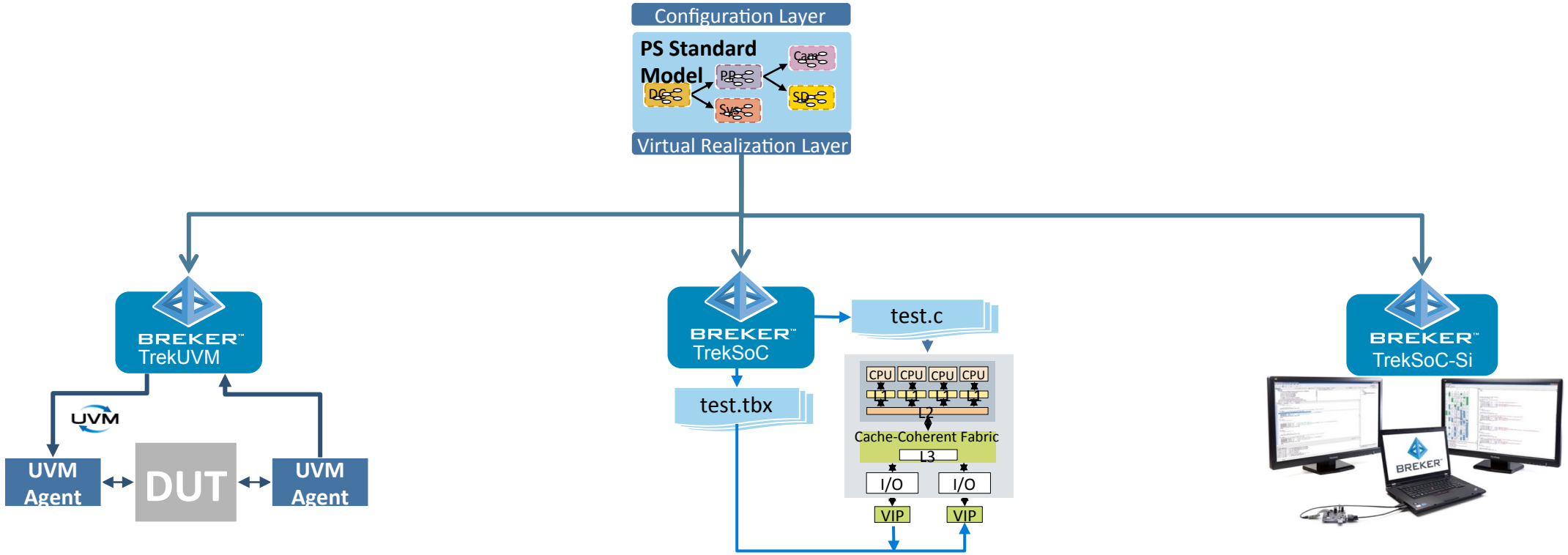
Packaging Automated Test Content



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

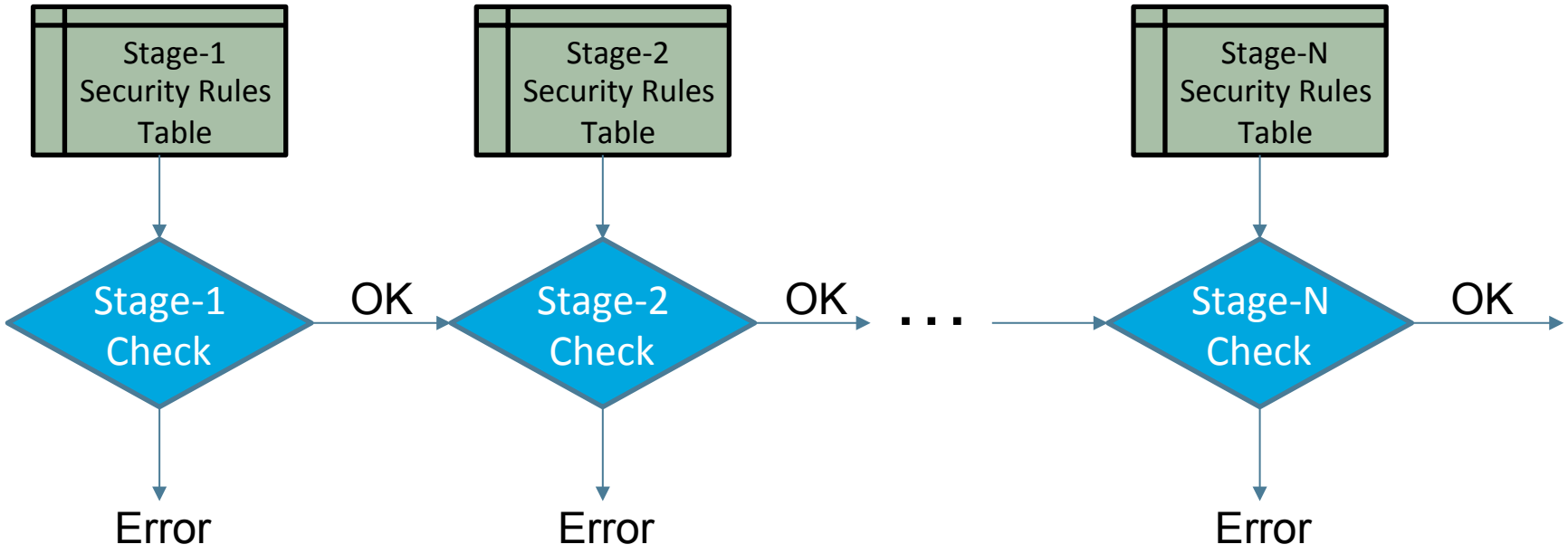
Software-driven SoC test
Rigorous, high-coverage tests from a single, simple specification

Prototyping & silicon diagnostics
Verification test reuse with observability/controllability

Breker End-user Proven Security Methodology

Note Code Examples to Follow

Objective: Multi-level Security Checking

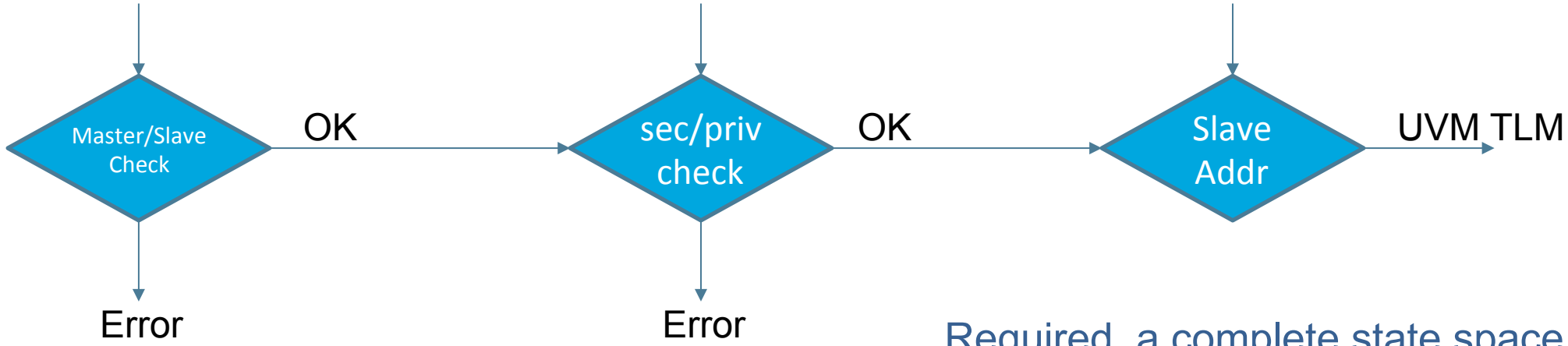


Tables Used To Define Security Policy

Master	CCU_IOM	TCU	L4_AHB	L4_MAIN	L4_MP	L4_SP	L4_SYS	S4_SYS_G	L4_EC
AXI_AP	1	1	1	1	1	1	1	1	
CCU_IOS	0	1	1	1	1	1	1	1	
DMA_TBU	1	0	0	1	0	1	1	1	
EMAC_TB	1	0	0	0	0	0	0	0	
IO_TBU	1	0	0	0	0	0	0	0	
SDM2HPS	1	1	1	1	1	1	1	1	
SDM_TBU	1	0	0	0	0	0	0	0	

	TxnSecure	TxnPriviledge	SlaveSecure	SlavePriviledge	Valid
	sec	priv	sec	priv	1
	sec	priv	sec	non_priv	0
	sec	priv	non_sec	priv	0
	sec	priv	non_sec	non_priv	0
	sec	non_priv	sec	priv	0
	sec	non_priv	sec	non_priv	1
	sec	non_priv	non_sec	priv	0
	sec	non_priv	non_sec	non_priv	0
0	non_sec	priv	sec	priv	0
1	non_sec	priv	sec	non_priv	0
2	non_sec	priv	non_sec	priv	1
3	non_sec	priv	non_sec	non_priv	0
4	non_sec	non_priv	sec	priv	0
5	non_sec	non_priv	sec	non_priv	0
6	non_sec	non_priv	non_sec	priv	0
7	non_sec	non_priv	non_sec	non_priv	1

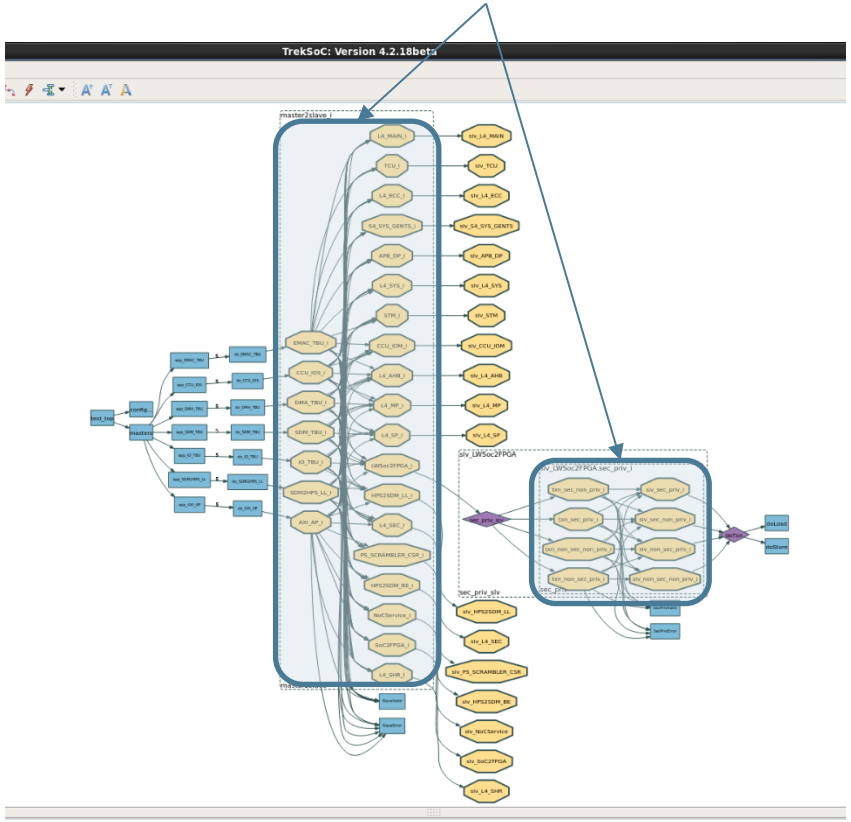
```
trek_cfg_add_memory_region "mr_CCU_IOM"
trek_cfg_add_memory_region "mr_TCU"
trek_cfg_add_memory_region "mr_L4_AHB"

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
```



Required, a complete state space exploration of the table combination

Need to Generate Intent State-Space Graph

Graphs Auto-Generated from Table



- 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

Synthesized Fabric Test Suite

TrekBox: Version 4.2.18beta

File Tests View Preferences Select Window

Find: in C Test Source Match Case

AXI_AP T1: cfg_done.3, AXI_AP.1, AXI_AP.2, AXI_AP.3, AXI_AP.4, AXI_AP.5

CCU_IOS T1: config.1, cfg_done.1, CCU_IOS.1, CCU_IOS.2, CCU_IOS.3, CCU_IOS.4, CCU_IOS.5

DMA_TBU T1: cfg_done.7, DMA_TBU.1, DMA_TBU.2, DMA_TBU.3, DMA_TBU.4, DMA_TBU.5

EMAC_TBU T1: cfg_done.5, EMAC_TBU.1, EMAC_TBU.2, EMAC_TBU.3, EMAC_TBU.4, EMAC_TBU.5

IO_TBU T1: cfg_done.4, IO_TBU.1, IO_TBU.2, IO_TBU.3, IO_TBU.4, IO_TBU.5

SDM2HPS_LL T1: cfg_done.2, SDM2HPS_LL.1, SDM2HPS_LL.2, SDM2HPS_LL.3, SDM2HPS_LL.4, SDM2HPS_LL.5

SDM_TBU T1: cfg_done.6, SDM_TBU.1, SDM_TBU.2, SDM_TBU.3, SDM_TBU.4, SDM_TBU.5

Memory Map

Memory Values

EMAC_TBU.2 Transactions

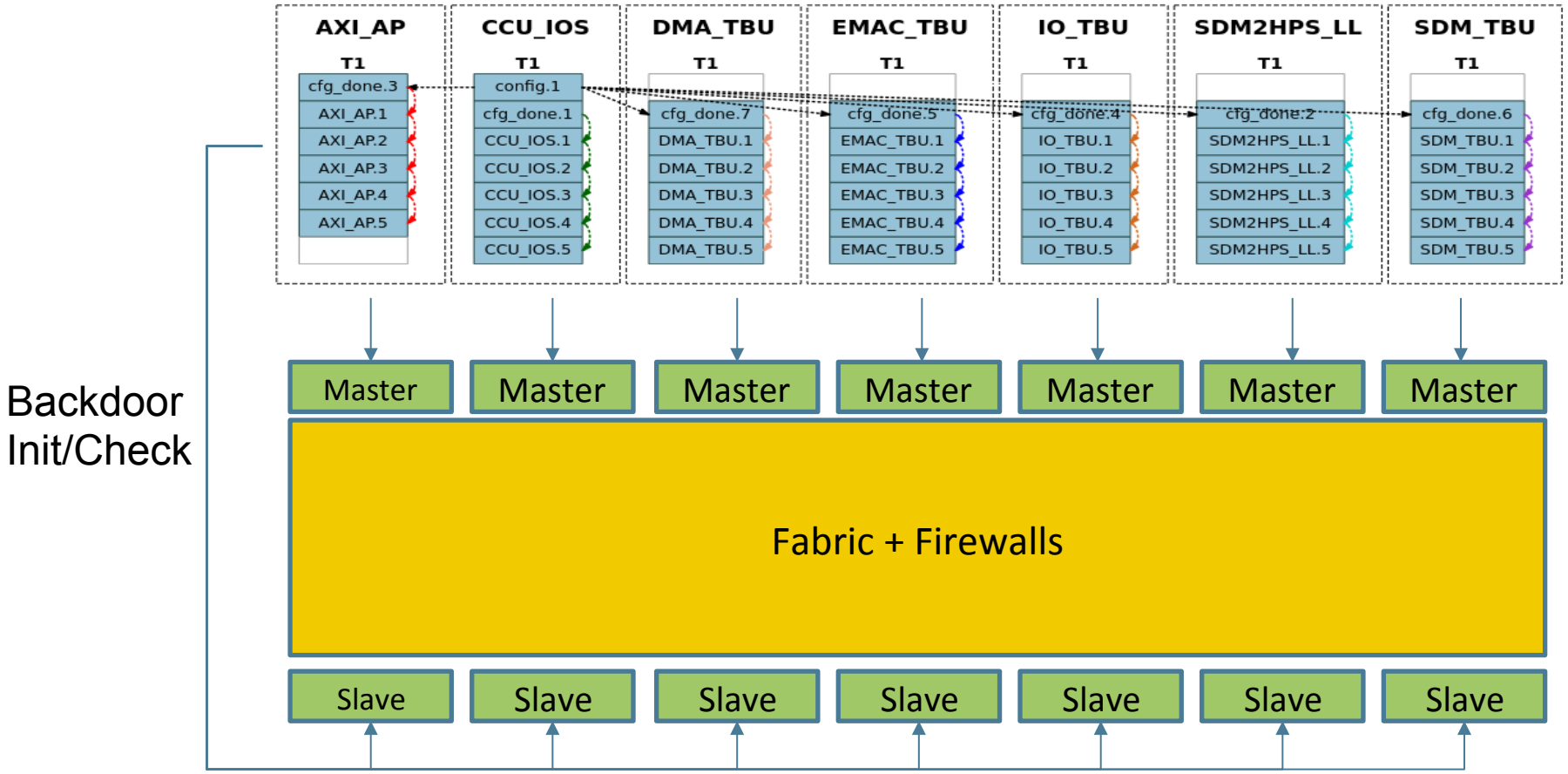
```
trek_message ("Begin");  // [event:58 cpu:EMAC_TBU thread:T1 instance:EMAC_TBU
verbatim ("    /* EMAC_TBU -> PS_SCRAMBLER_CSR SlaveError */");
verbatim ("    /* SecPrivError: non_sec priv */");
verbatim ("    /* Store */");
verbatim ("    /* addr:            trek_mem_mr_PS_SCRAMBLER_CSR+0x00000004 */
verbatim ("    /* data:            _03 */");
verbatim ("    /* reqMasterName:   EMAC_TBU */");
verbatim ("    /* reqSlaveName:    PS_SCRAMBLER_CSR */");
verbatim ("    /* reqSlaveStatus:  error */");
verbatim ("    /* txnSec:          non_sec */");
verbatim ("    /* txnPriv:         priv */");
verbatim ("    /* txnSecPrivStatus: error */");
trek_message ("End");  // [event:59 cpu:EMAC_TBU thread:T1 instance:EMAC_TBU.2
```

```
TrekSoc Copyright (C) 2004-2014 Breker Verification Systems
http://www.brekersystems.com/
Created on Fri Jun  8 10:09:42 2018

trek: info: Initializing TrekBox
trek: info: trek_write_memory_block_backdoor ( trek_mem_mr_APB_DP+0x7c0, <0x10
```

Memory locations of IO_TBU.1

Running test /home/adnan/svn/sup/customers/altera/secApp2/run/trek_test.tbx.log

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