BREKER™

THE SoC VERIFICATION COMPANY

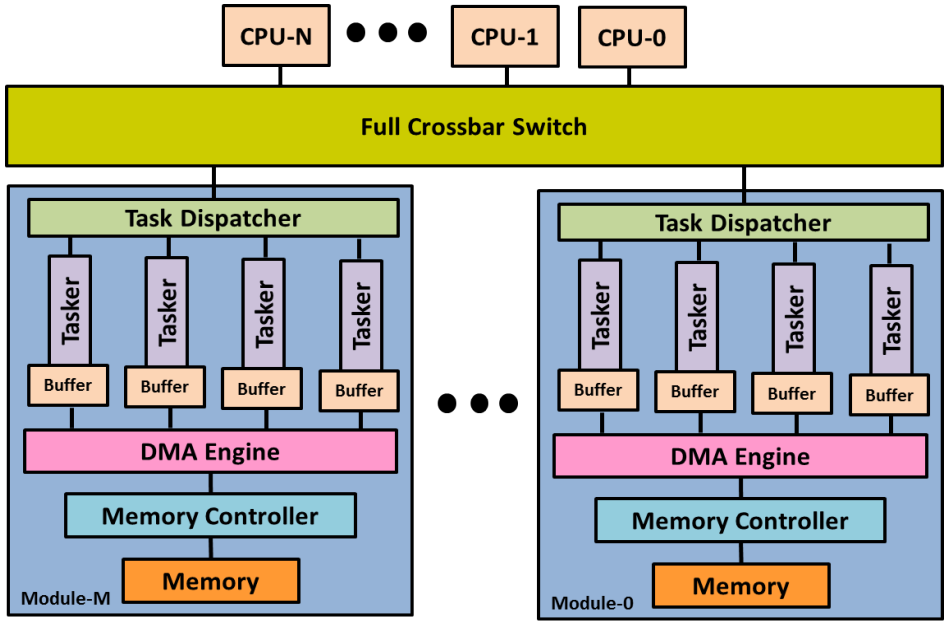
A REAL EXAMPLE OF ACCELLERA PORTABLE STIMULUS USED TO VERIFY AN LTE SWITCH

Objective

Generate software driven verification (SDV) test cases in "C" for an LTE Base Station Switch that increases coverage closure while reducing verification effort

- **Complex** design supporting many possible use models
- Particular focus on **dependency management** across tasks and resources
- Create **single verification model** that can scale from simple to complex tests
- Create **portable stimulus** model that can generate tests for simulation, emulation and post-silicon
- Create **self-checking** test cases that are easy to **debug**
- Prove **coverage closure** on system use cases

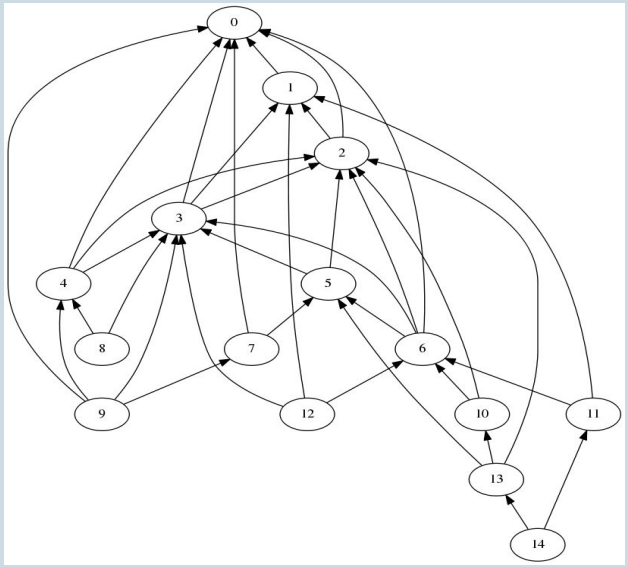
LTE Base Station Switch Design



LTE Base Station Switch

- **CPUs** configure Task Dispatcher with 1000's of tasks
- **Task Dispatcher** schedules tasks on Taskers as resources become available
- Each task has **many possible formats** with different paths through system
- Each task many have **complex dependencies** on other tasks and resources
- Task Dispatcher must **manage dependencies** across tasks and resources
- Task Dispatcher must **track progress and completion** of tasks to free resources
- One Task Dispatcher and multiple Taskers in each **cluster**
- Multiple clusters in **full chip**

Verification Requirements



Generated task dependency graph

Complexity Requirements

- Individual tasks have **many possible formats** and many possible **resource dependencies**
- Need to test **complex dependency scenarios** across tasks, task dispatchers and taskers
- Need to generate complex **data structures** in memory
- Need to **manage memory** across multiple regions
- Task Dispatchers have **complex configurations** with many variations

Scalability Requirements

- Scaling of generated tests from **simple to complex**
- Scaling of system under test from **sub-system to full-chip**

Portability Requirements

- Single source of tests must run on **simulation, emulation, post-silicon**
- Support d**ifferent test case mechanics** for each platform

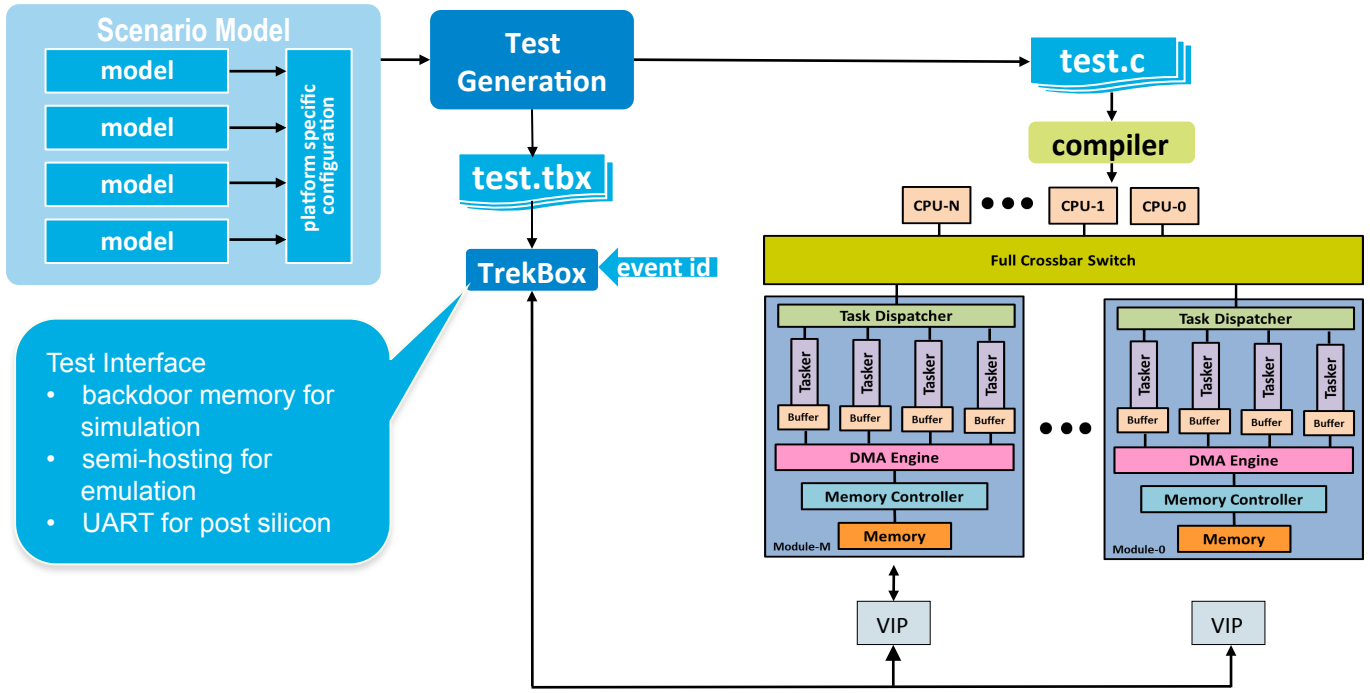
Checking & Debug Requirements

- **Automatic Generation** of self-checking tests
- Track progress of each task through **life-cycle**
- **Predict & check** results in memories and registers
- **Interactive debug** to identify and root-cause errors

Coverage Requirements

- Prove **coverage closure** on all use cases

Method



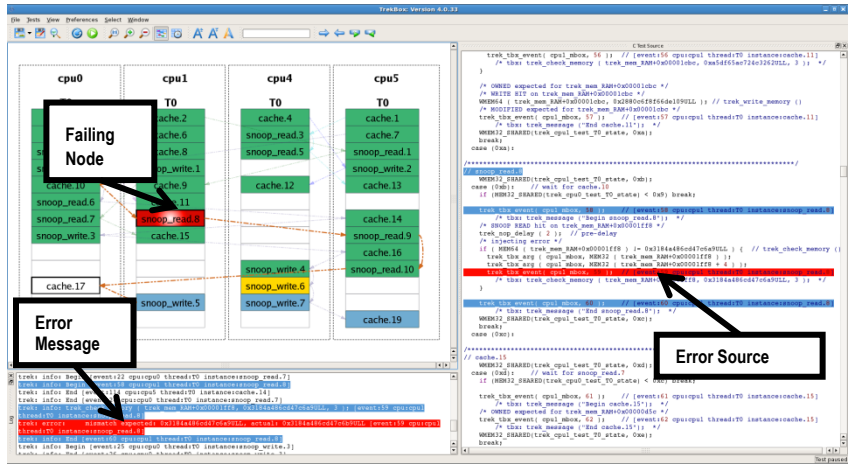
SW Driven Verification Flow

Test Generation Tool

- Use **Breker TrekSoC** to provide scalable portable stimulus-based solution
- **Algorithmic graph-based scenario models** to capture verification space

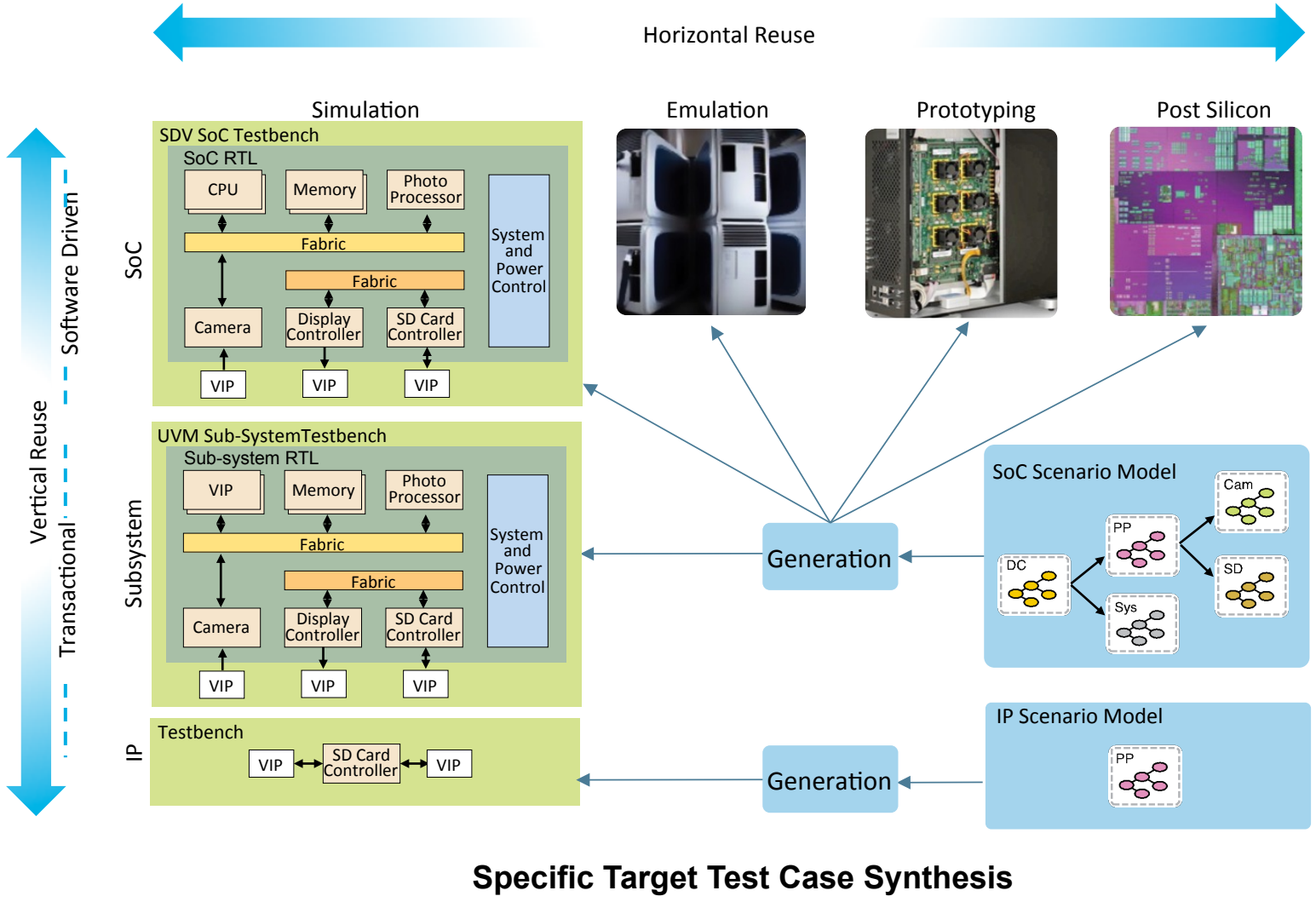
Checking & Debug

- Automatically generated **self-checking test cases**
- Interactive Test Map view to **identify** and **root-cause errors**



Test Map Debug View

Portable Stimulus



Specific Target Test Case Synthesis

- Common **scenario model** used at all stages of verification
- Tool **configured** for each environment
  - Complexity of tests to generate
  - Number of CPUs, Task Dispatchers and Taskers
  - Number and size of memory regions
  - Communication mechanism between test and TrekBox

Results

Scenario Model Construction

- Algorithmic graph-based portable stimulus scenario model **"blind-built"** from spec before simulation available

- First generated test running **within hours** of initial bring-up
  - graph-based models allow simple creation of complex test scenarios
  - graph-based models allow incremental build out of scenario model
  - test cases correct by construction, even with 1000's of interacting tasks and complex Task Dispatcher programming cases

Test Complexity Scaling

- Seamless scaling of test complexity from **single task to 1000's of inter-dependent tasks**
- Seamless scaling from **single** CPU/Task Dispatcher/Tasker to **multiple** CPUs/Task Dispatchers/Taskers across multiple clusters

Portable Stimulus

- **Target specific** self-checking tests generated for simulation and emulation with different test case mechanics
  - Easy to replay failing emulator cases on simulation

- **Waiting** for silicon to generate post-silicon tests
  - post-silicon flow proven on earlier generation design

Checking and Debug

- **Task dependency graph** generated from tool used to help follow test case intent

- **Design errors** rapidly identification and root-caused

- **Common debug view** across all simulation, emulation and post-silicon aided debug

Coverage Closure

- **Path coverage** of graph-based models provide metric for use case coverage
  - different from RTL functional coverage or line coverage

- Tool based automatic **coverage closure** on graph-based model