PSS in Action: Scalable Test Reuse from Design Verification to Silicon

Akshay Bhosale (AMD)
Klaus-Dieter Hilliges (Advantest)

accellera SYSTEMS INITIATIVE™
AMD
ADVANTEST®
2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Speakers' brief biography

Akshay Bhosale | Senior System Software Designer | Advanced Micro Devices.

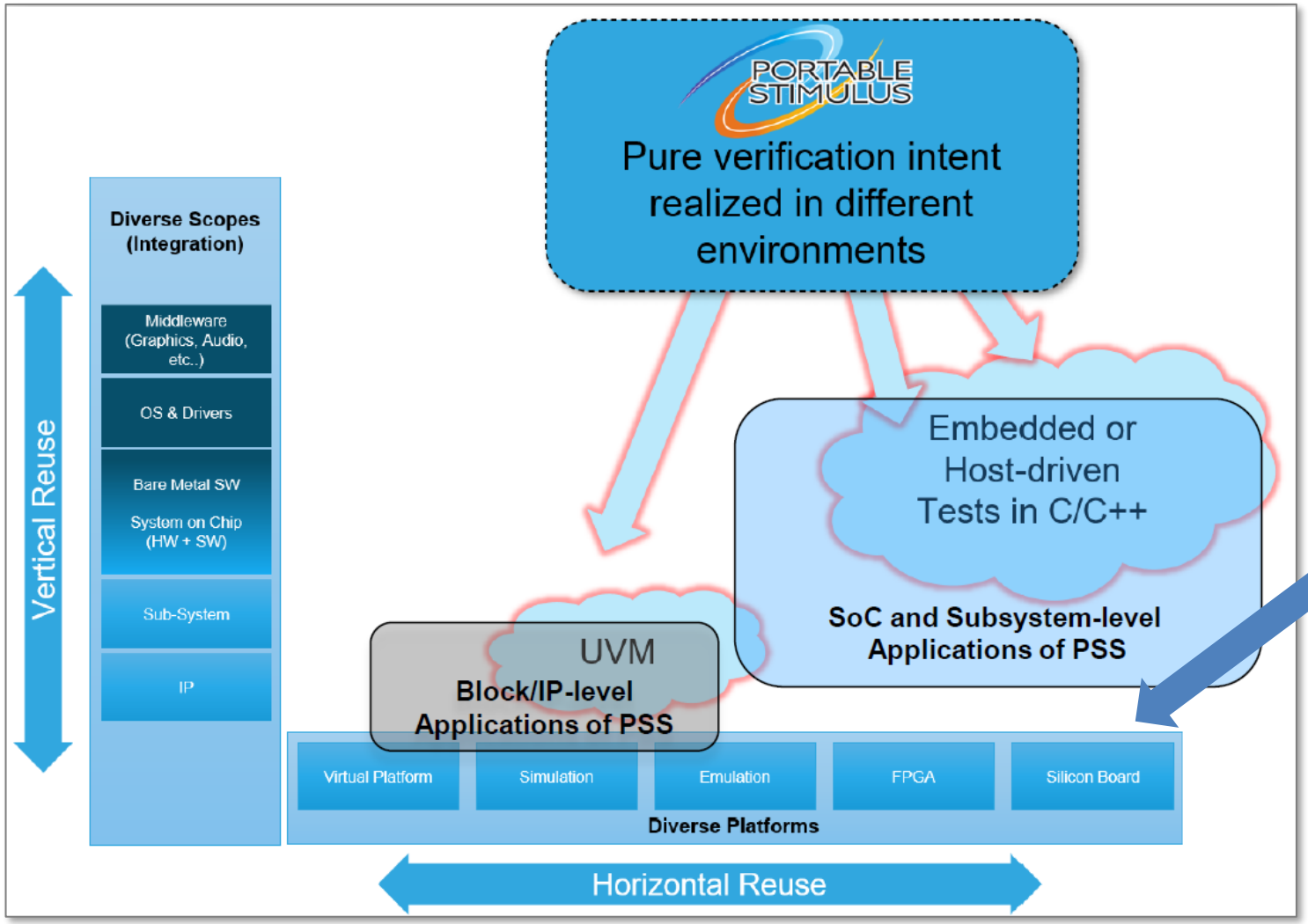
He has worked extensively on building Portable Stimulus (PSS) models for complex IPs, focusing on improving verification workflows. His current interests lie in developing scalable and reusable PSS models that can streamline adoption and enhance verification efficiency across diverse environments.

Klaus-Dieter Hilliges | Senior Director in Engineering Solutions | Advantest Germany.

Graduated from University of Stuttgart and Cornell University, Klaus entered Hewlett-Packard's IC Test Business where he pursued responsibilities in R&D, Product Marketing and Application Engineering before he was transferred to Agilent Lab's in Palo Alto to lead the semiconductor test research. For 10 years, Klaus developed the Shanghai R&D Center of Verigy and successfully integrated the R&D operations of Verigy and Advantest. Klaus is now residing in Germany after driving the extension of the V93000 SOC Test Platforms for test over functional PCIe and USB. He is now leading Advantest's initiative to address the relentless complexity growth and time-to-market challenges in silicon validation.

Portable Test and Stimulus Standard: Re-Usable Content

https://www.accellera.org/activities/working-groups/portable-stimulus



Goal:

Re-use Pre-Silicon Test Content to Post-Silicon on level of SoC and Subsystem

Sergey Khaikin / Cadence
DVCON US 2024
PSS Tutorial

accellera SYSTEMS INITIATIVE™

2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Outline

- **Post-Silicon Testing with PSS (Akshay / AMD)**
  - Goals of Post-Silicon Testing
  - PSS Benefits for Post-Silicon
  - Example
  - Conclusions

- **Automated Flow and Unified Environment for Post-Silicon Testing with PSS (Klaus / Advantest)**
  - Motivation
  - Automated Flow
  - PSS Test Setup for Automated Execution on Silicon
  - *SiConic*™ – Solution Overview & Benefits
  - Unified Ecosystem

Post-silicon testing goals

Early silicon bring-up

- Screen for defective parts
- Ensure major features and data paths are working

Systematic feature coverage

- Run tests to verify each SoC feature in isolation to build a baseline

Stress testing

- Gradually build-up test complexity from feature coverage to multiple features together
- Create and run tests that mimic real-world scenarios

All feature enablement with OS and application

- Run real production use cases, measure power and performance

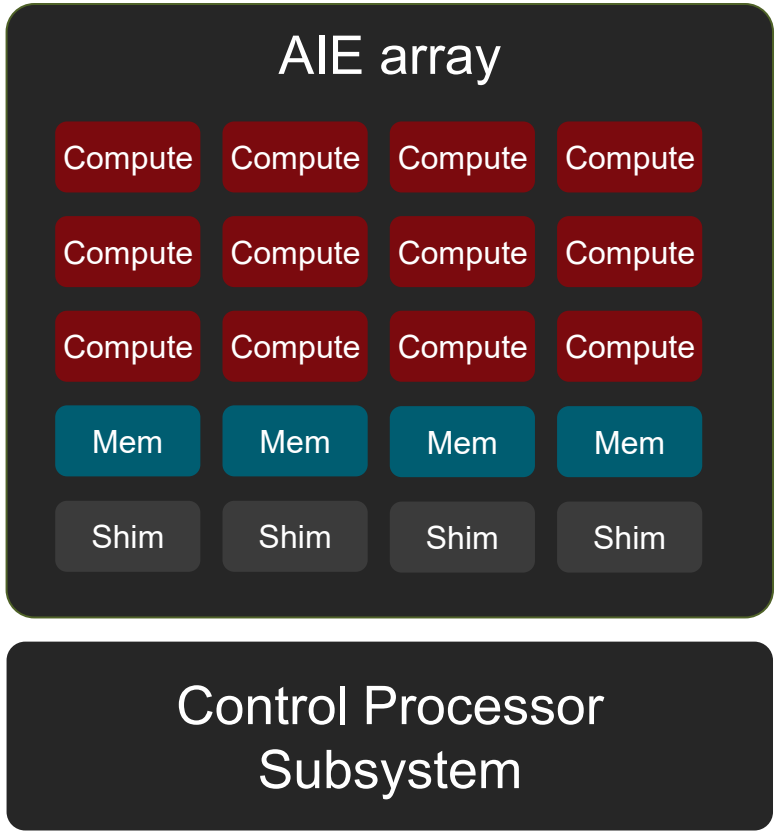
Streamline post-silicon testing with PSS

The benefits of PSS stimulus

- **Reduced test development cost**
  - Save time and money by reusing pre-silicon IP/SoC test content
  - Empower your team with formal action-based knowledge transfer of test space to enable anyone to create complex SoC tests
- **Feature coverage reports**
  - Ensure comprehensive testing with PSS coverage features to check coverage of tests before and after run
- **Failure debug in simulation or emulation**
  - Quickly root-cause failure cases for stress test fail or failures in the field with PSS Lego-block based tests
  - Connect directly with IP experts using PSS language as a common language for efficient debugging

AI Engine IP – An example use case

- **A 2D arrays consisting of multiple AI tiles**
  - Compute, memory and interface DMA tiles
- **Grid of engines and highly configurable network makes creating post-silicon validation tests very challenging**

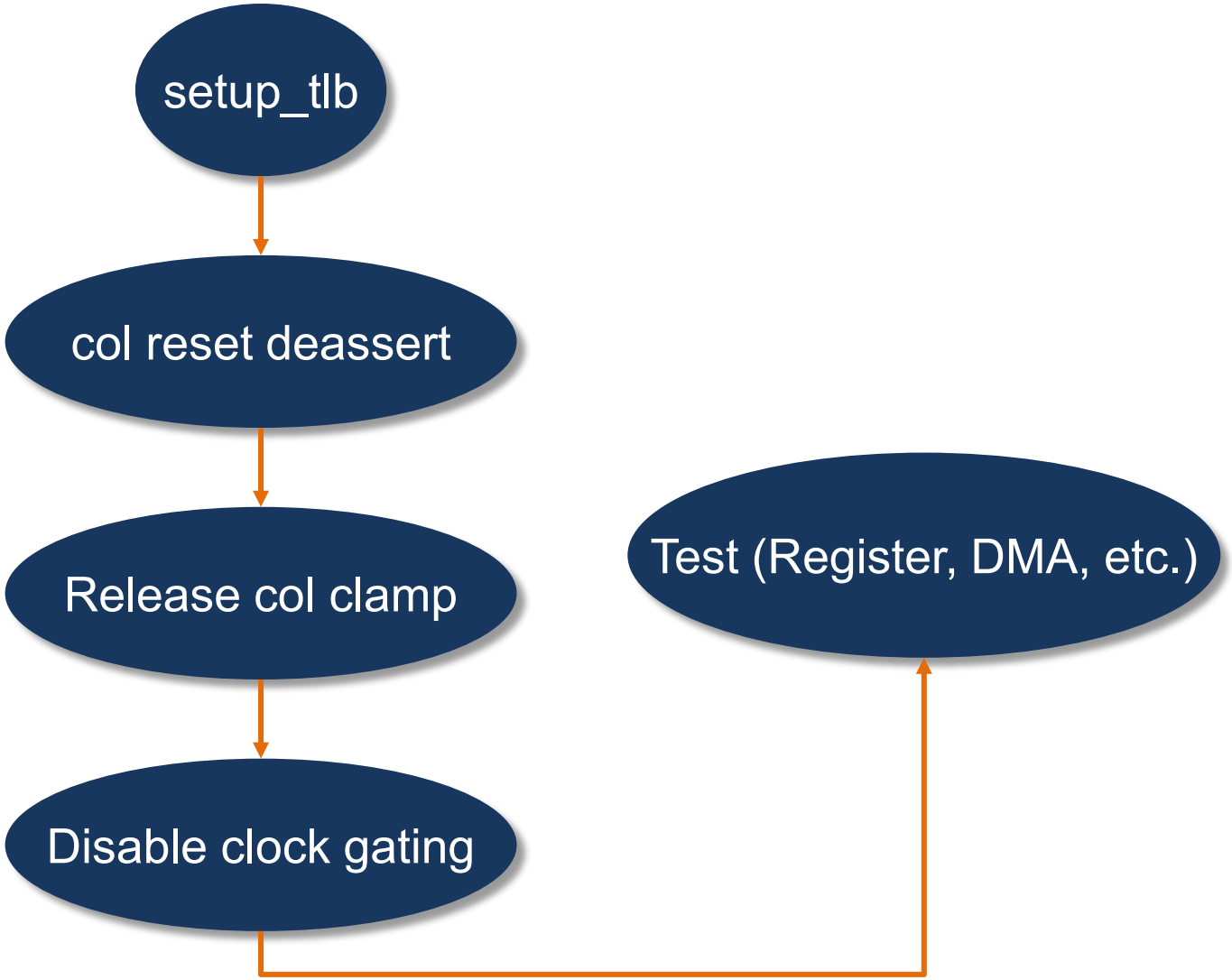


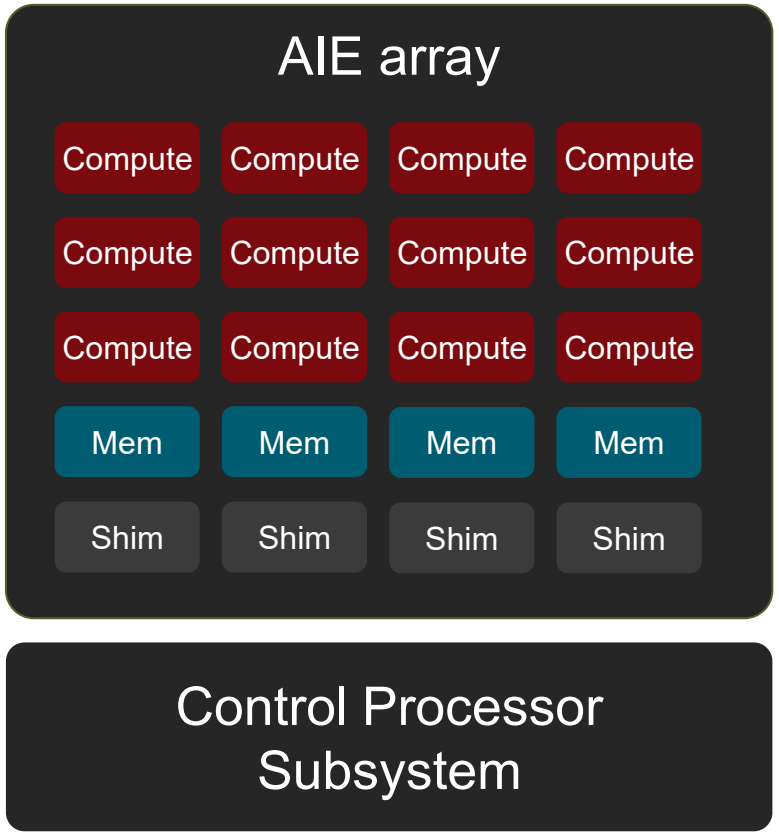
Early silicon bring-up

- **Tests for Power, reset, and clock sequencing**
  - Quickly create experimental test sequences to screen for meta-stability and manufacturing issues
  - Tested in UVM, ported to processor by PSS tool
- **Tests for major datapath and feature coverage**
  - Run isolated simple tests for datapath and major features to create a coverage baseline and to screen for manufacturing issues
- **Test coverage report**
  - Automatic coverage report with PSS coverage feature

PSS building block approach makes turnaround fast for experimental functional tests

Array boot with control processor





AI Engine PSS model

PSS model created by IP team

```
// setup address map though TLBs
action setup_tlb { };

// Power up a column
action release_clamp {
  rand int in [0..COLS-1] col;
  rand bool release;
};

// gate, un-gate the column clock
action clock_gate {
  rand int in [0..COLS-1] col;
  rand bool disable_cg;
};

// assert, de-assert the column reset
action reset_col {
  rand int in [0..COLS-1] col;
  rand bool deassert;
};
```

```
abstract action boot_base {
  output     aie_state aie_state_out;
  constraint aie_state_out.aie_state_obj.boot == true;
};

// Normal boot sequence
action boot_array : boot_base {
  activity {
    do setup_tlb;
    repeat(c: COLS) {
      do reset_col with {col == c; deassert;};
    };
    repeat(c: COLS) {
      do release_clamp with {col == c; release;};
    };
    repeat(c: COLS) {
      do clock_gate with {col == c; disable_cg;};
    };
  };
};
```

AI Engine PSS model

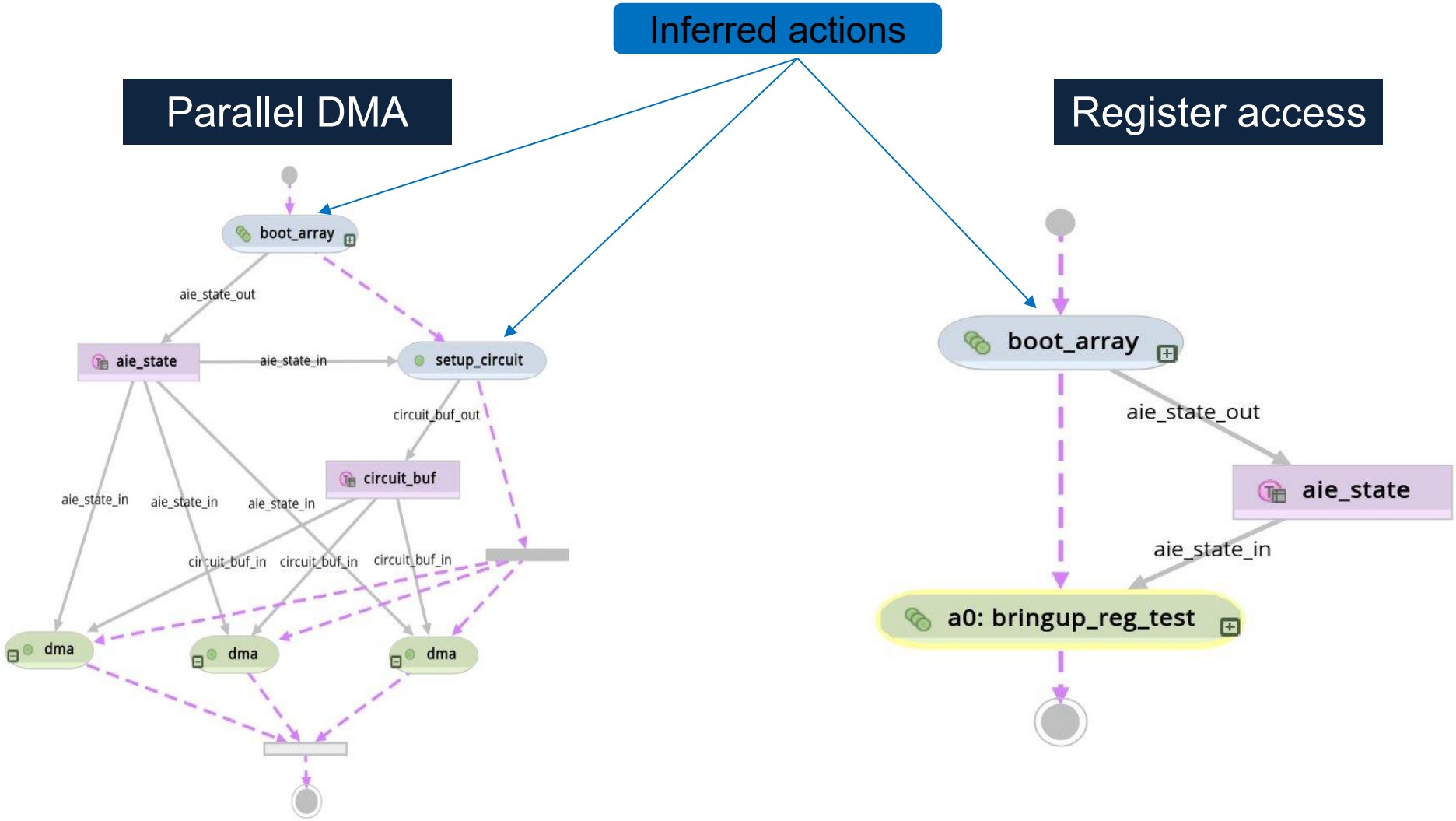
PSS model created by IP team

```
// Base action for all building block actions
abstract action aie_base {
  input aie_state aie_state_in;
  constraint aie_state_in.aie_state_obj.boot == true;
};

// A few random register accesses
action register_access : aie_base {
  rand int in [0..NOCS-1] noc;
};

// Program one circuit through the array
action setup_circuit : aie_base {
  output circuit_buf circuit_buf_out;
};

//
action dma : aie_base {
  input circuit_buf circuit_buf_in;
  rand conn_s src;
  rand conn_s dst;
};
```

```
// Simplest early bringup test
action bringup_reg_test {
  activity {
    do register_access;
  };
};
```

```
// Bringup feature test
action bringup_test_dma_all_cols {
  activity {
    parallel {
      replicate(c: COLS) {
        do dma with {
          src.tile.col == c;
          circuit_buf_in.col_circuit == true;
        };
      }
    }
  };
};
```

Solved bringup tests



Nothing is working in the lab!

- **Access to registers inside the AI Array failing randomly**
- **Is the boot sequence wrong?**
- **Is there manufacturing fault**
- **Are we seeing metastability?**
- **Need to create lots of experimental bootcode and tests**

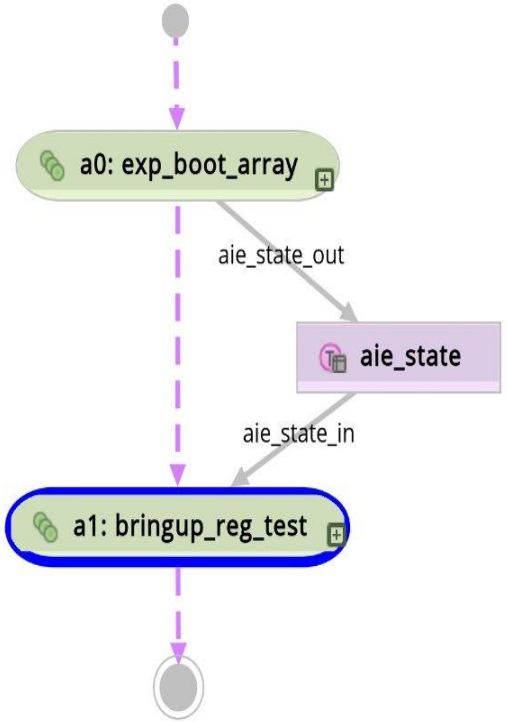
```
// Experimental boot sequence

action exp_boot_array : boot_base {
  activity {
    do setup_tlb;
    repeat(c: COLS) {
      do release_clamp with {col == c; release;};
    };
    repeat(c: COLS) {
      do clock_gate with {col == c; disable_cg;};
    };
    repeat(c: COLS) {
      do reset_col with {col == c; deassert;};
    };
  };
};
```

All building block actions may have complex rules about sequencing that can not be violated. Action may have random attributes with constraints. A person in lab doesn't need to be an expert to be able to create experimental tests. PSS tool can create a test with minimal user input

New experimental tests quickly created in Lab

```
action new_reg_test {
  activity {
    do exp_boot_array;
    do bringup_reg_test;
  }
};
```

```
action bringup_test_dma_col_0 {
 activity {
  do exp_boot_array;
  parallel {
   do dma with {
     src.tile.col == 0;
     circuit_buf_in.col_circuit == true;
   };
  };
 };
};
```



Conclusion

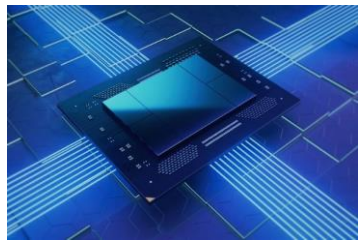
- **Embrace the Future with PSS**
  - PSS, the state-of-the-art unified pre- and post-silicon testing methodology, leverages randomization and high-level constructs to revolutionize testing
- **Streamlined Test Documentation**
  - Formal test space documentation, coupled with action building blocks, empowers anyone to create tests quickly and efficiently
- **First-Class Automated Coverage Reporting**
  - With PSS, coverage reporting is elevated to a first-class citizen, ensuring comprehensive and accurate testing results
- **Comprehensive Test Suite Generation**
  - Generate a large suite of tests designed for rigorous stress testing and yield optimization, ensuring the robustness and reliability of your systems
- **Simplified Debugging Process**
  - PSS building blocks streamline the debugging process, making it easier to identify and rectify issues, enhancing overall efficiency and productivity

Outline

- **Post-Silicon Testing with PSS (Akshay / AMD)**
  - Goals of Post-Silicon Testing
  - PSS Benefits for Post-Silicon
  - Example
  - Conclusions

- **Automated Flow and Unified Environment for Post-Silicon Testing with PSS (Klaus / Advantest)**
  - Motivation
  - Automated Flow
  - PSS Test Setup for Automated Execution on Silicon
  - *SiConic*™ – Solution Overview & Benefits
  - Unified Ecosystem

Motivation: Address Escalating Engineering Challenges



Increased complexity

2.5D and 3D integration

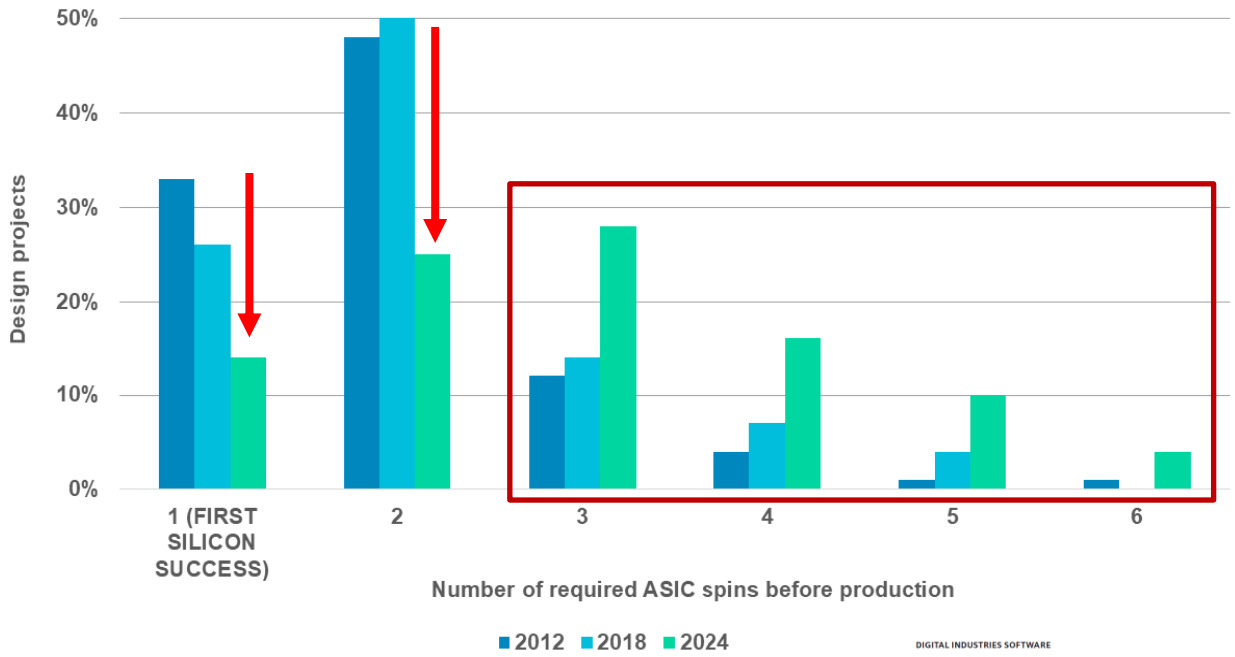
TTM/TTQ pressure

Accelerating development cycles for high-performance designs

Engineering efficiency

More design variations to win in diversified semiconductor market

Success Rate of 1st and 2nd Silicon Dropping (>50% need 3 tape-outs to reach target functionality, performance, yield & more)



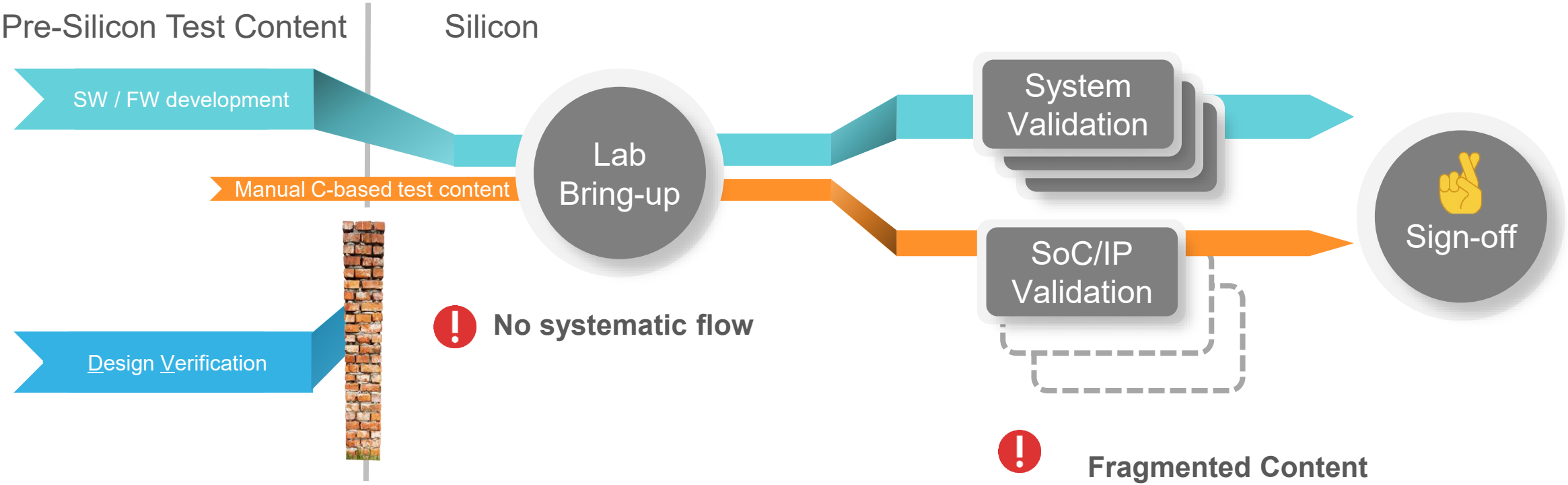
DIGITAL INDUSTRIES SOFTWARE

2024 Wilson Research Group IC/ASIC functional verification trend report

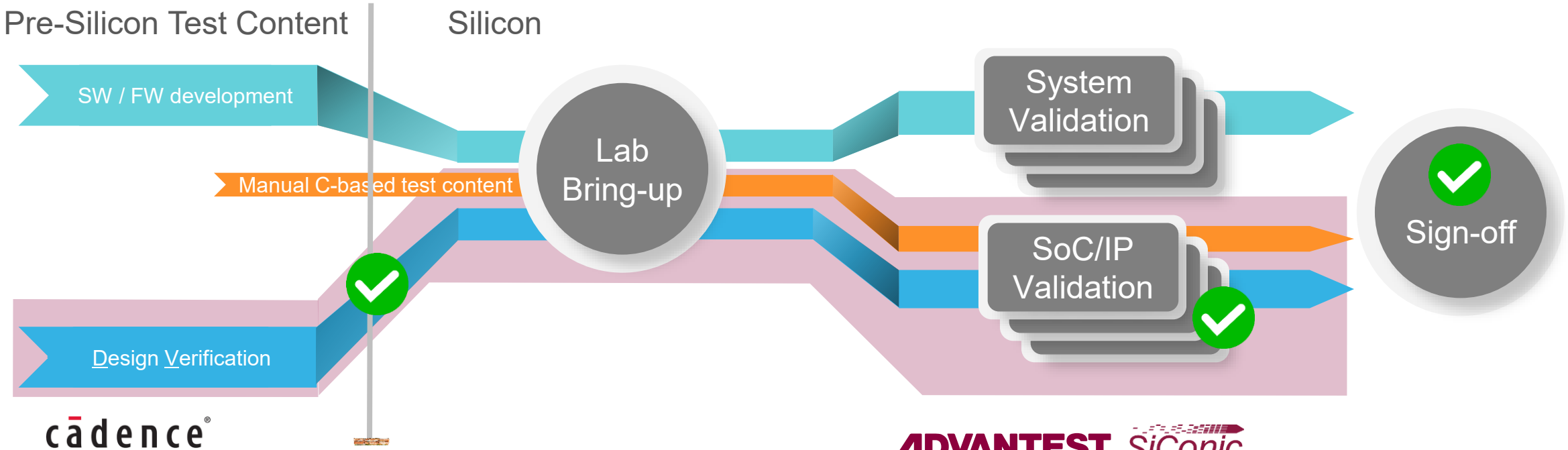
accellera SYSTEMS INITIATIVE™

2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Silicon Validation Challenges



Unified Software and Hardware environment for Automated Silicon Validation



cadence
SIEMENS
SYNOPSYS
LAUTERBACH DEVELOPMENT TOOLS

ADVANTEST SiConic

Unified, Automated and Scalable Ecosystem

DV Tests on Silicon:

- Faster Execution: 10x to 10,000x ➔ More Coverage / Time
- Real power/performance measurements ➔ Reliable, Data-Driven Decision

accellera SYSTEMS INITIATIVE

2025 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION INDIA

PSS Test Re-use for Silicon Validation and HVM

Pre-Silicon Design Verification

- Slow execution
- Model-based PPA estimates
- Internal controllability and observability (depending on pre-silicon platform)
- Simulation, emulation, prototyping with well-defined scope, abstractions

Silicon Validation & High-Volume Manufacturing (HVM)

- Execution at the speed of silicon enables increased coverage (validation, HVM)
- Real device physics: i.e. control/observe power, temperature, performance (clock frequency, runtime)
- Black-box execution (pass/fail, signature)
- Real device challenges: power-up, “boot”, re-target executable (e.g. simulate for traditional ATE) & more

⇨ **SiConic™ PSS Lib:** SoC/IP config, runtime para’s, tracing, result/state access

⇨ **SiConic™ Unified Env.:** Directly load, debug and execute code for device setup and test on silicon

Overview: Automated Flow for SoC Validation

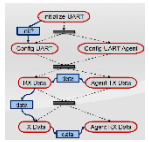
Test Generation, EDA

Functional Test Environment For DV/SV Engineering

PSS tools

Test case generation with PSS re-use

Advantest SiConic PSS Lib.



ELF

Manual or other EDA tools

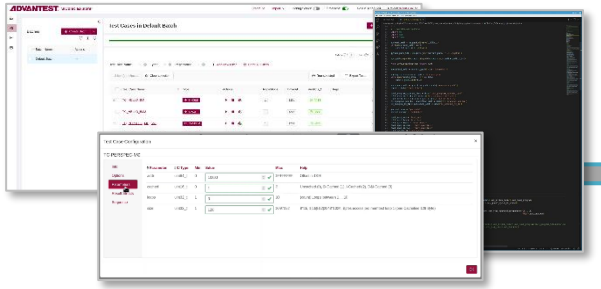
Generate test files using custom tools

.SVF

ELF

FDAT Interface

Load & Parameter Variation



Bench Setup

or Link Scale

SOC Device

JTAG HSIO

Load binary (incl. On-Chip-Monitor)

Execution

Activity Trace

Results

Debug IF

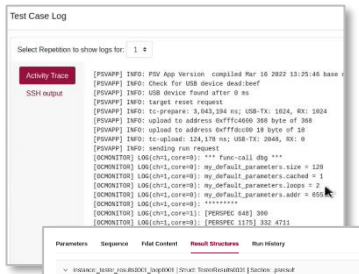
JTAG

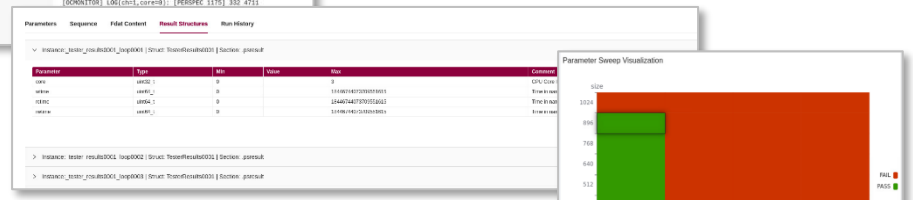
Debug

LAUTERBACH DEVELOPMENT TOOLS

Activity Trace

Results





N-Dimensional parameter variation and visualization

EDA debug tools

Analyze on-chip activity

accellera SYSTEMS INITIATIVE

2025 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION INDIA

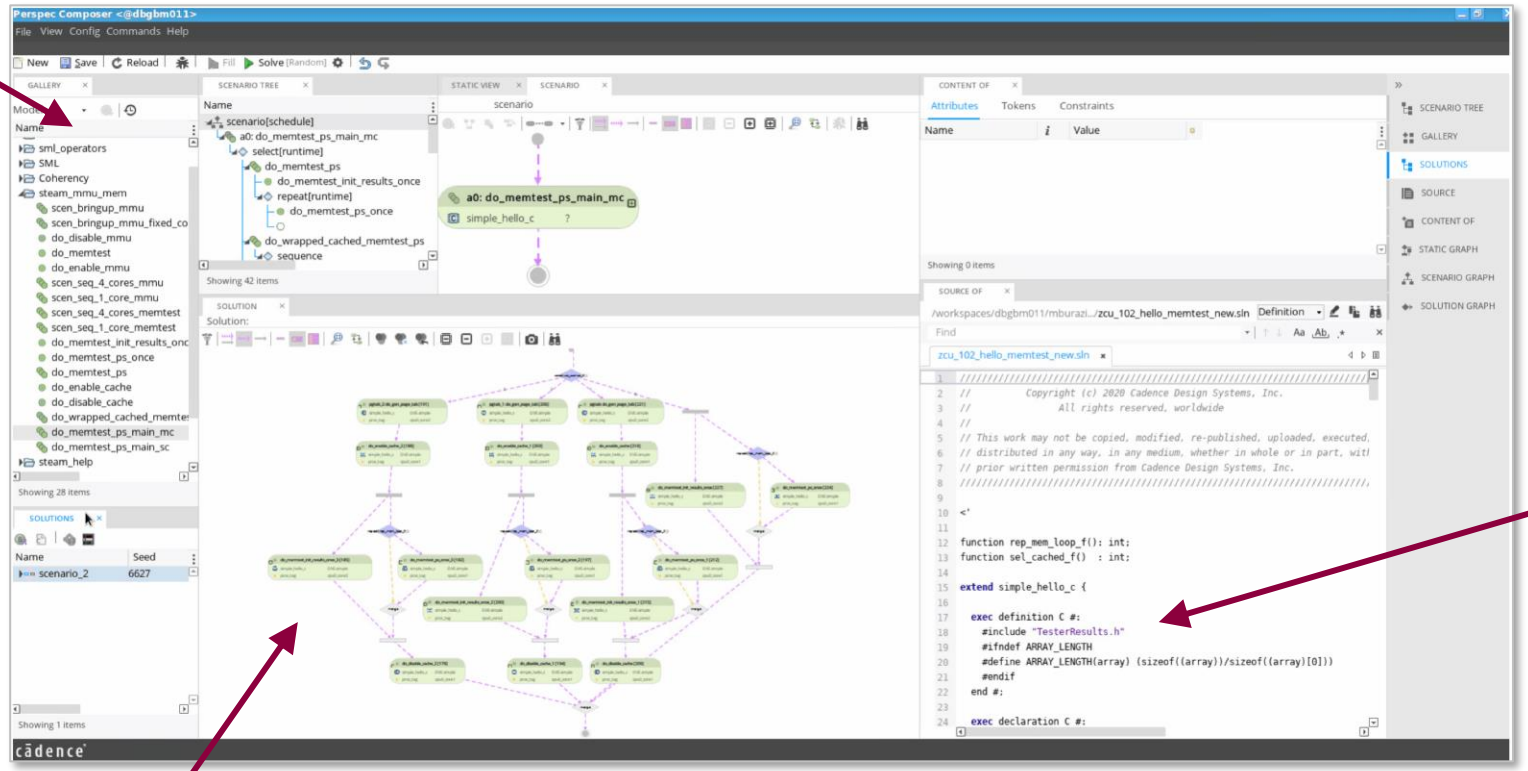
PSS Tool + SiConic PSS Lib: Extend PSS to Silicon

Example: Cadence' Perspec System Verifier supports SiConic as Silicon Platform

Library of DUT Blocks with their Actions

Library of Helper Blocks incl. SiConic PSS Lib .

Library of Tests: i.e. Top-Level Blocks



Created Source Code

Graphical Visualization of Sequence of Actions and Conditional Branches

accellera SYSTEMS INITIATIVE™

2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Runtime parameterize components with SiConic PSS Lib

pll_ip_pkg.pss

```
package pll_ip_pkg {
  // import SiConic packages
  import siconic_pkg::*;
  import siconic_target_types_pkg::*;

  // declare runtime parameter set
  struct pll_params_s : siconic_base_param_set_s {
    //@tooltip("Source: board = 0, test = 1")
    rand uint32_t source_p;
    constraint source_p_min_max { source_p in [0..1]; }

    //@tooltip("Divider")
    rand uint32_t divider_p;
    constraint divider_p_min_max { seed_p in [0..8]; }
  }




  // declare parameterized component
  component pll_c {
    // declare parameter-set component
    // for parameter set user_param_set_test_s
    siconic__param_set_c<pll_params_s> params_c;
  }
}
```

Declare a runtime param set in a native PSS struct for each component user_component_param_set_test_s.

Publish documentation to the bench user.

Declare the range for a parameter with native PSS constraint.

To parameterize your component with a runtime paramter set, declare a native PSS component that holds your parameter set.

Use runtime parameters in your test

pll_ip_pkg.pss

```
package pll_ip_pkg {
  import siconic_pkg::*;

  extend component pll_c {
    exec header C = """
      #include “pll.h";
    """;

    // declare target functions with runtime code
    target function void pll_configure(chandle param_set_pointer, int id);
    import target C function pss_configure;

    int id;

    action configure_a {
      exec body {
        message(DEBUG, “Configure PLL.”);

        // hand over parameter set pointer to runtime code
        comp.pll_configure(comp.params_c.get_pointer(), id);
      }
    }
  }
}
```

pll.h

```
void pll_configure(void * voidPointerToParamSet, int id);
```

Use native PSS runtime messaging.

Use native PSS getter function
chandle siconic__param_set_c<..>.get_pointer()
to retrieve a pointer to a parameter set.

Instantiate runtime parameterized components with native PSS in a single line.

test_pkg.pss

```
package test_pkg {
  import pll_ip_pkg::*;

  extend component pss_top {
    pll_c my_plls[2];

    action my_test_a {
      activity {
        // configure PLLs
        pll_c::pll_configure with { comp == this.comp.units[0]; };
        pll_c::pll_configure with { comp == this.comp.units[1]; };
      }
    }
  }
}
```

Resulting test code

my_generated_test.c

```
#include "pll.h"
#include "vip.h"
#include "TesterParams.h"

void my_main_core0(void) {
  write_message_to_trace(3);

  pll_configure(siconic__parameter_pointer_0, 0);

  write_message_to_trace(4);

  pll_configure(siconic__parameter_pointer_1, 0);
}
```

Runtime messages stream via HSIO or JTAG buffer to activity trace.

Pointers refer to instances of parameter sets in C struct.

TesterParams.c

```
#include “TesterParams.h"

pll_pkg_pll_params_s * siconic__parameter_pointer_0 = (pll_pkg_pll_params_s *) &(my_default_parameters_0);
pll_pkg_pll_params_s * siconic__parameter_pointer_1 = (pll_pkg_pll_params_s *) &(my_default_parameters_1);
```

C code for parameters reflects hierarchy of PSS model and contains min, max, and documentation.

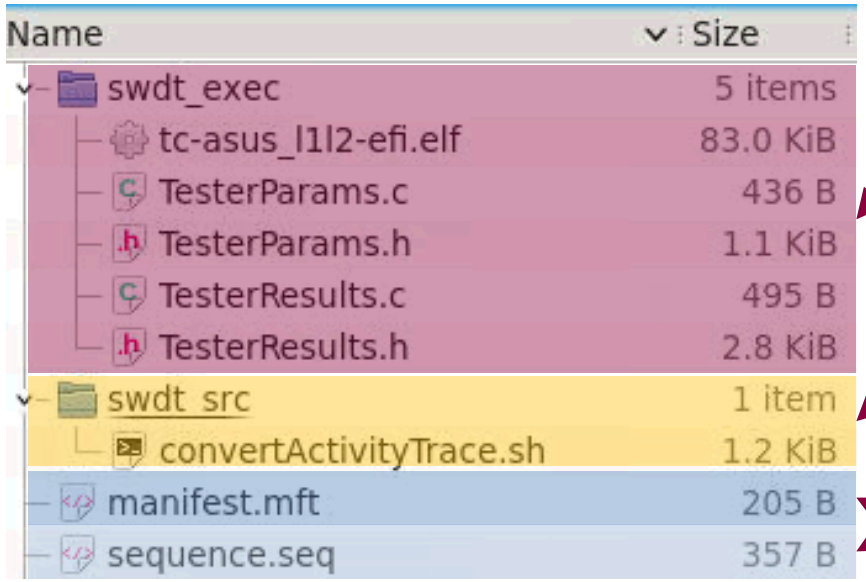
TesterParams.h

```
#include “stdint.h"

__attribute__((section(".psparam"))) struct pll_pkg_pll_params_s = {
  uint32_t pss_top__my_ppls0__pll_params_c__source_p; /* [min: 0, max: 1, comment: Source: board = 0, test = 1] */
  uint32_t pss_top__my_ppls0__pll_params_c__divider_p; /* [min: 0, max: 8, comment: Divider] */
};

extern struct pll_pkg_pll_params_s my_default_parameters_0;
extern struct pll_pkg_pll_params_s my_default_parameters_1;
```

FDAT Interface – Integrating All Content Needed for Silicon

FDAT: Test content container implemented as archive (.zip format)

Example FDAT file:



Test case executable (.elf)
Test case parameter def. (.c/.h)
Test case results def. (.c/.h)

Optionally: **Test case source files**
Additionally: **Scripts** (e.g. to analyze trace)

Test exec. steps:
e.g. **1. Device power-on** (e.g. external power)
2. Device setup (e.g. on-chip sensors)
2. Test execution
3. Post-proc. (e.g. sensor output)

FDAT Manifest: i.e. File version, owner & more (custom fields)

Executable scripts: SH, TCL, SVF, CMM (Software Debugger Script), & more

PSS on SiConic enabling Advanced Features on Silicon

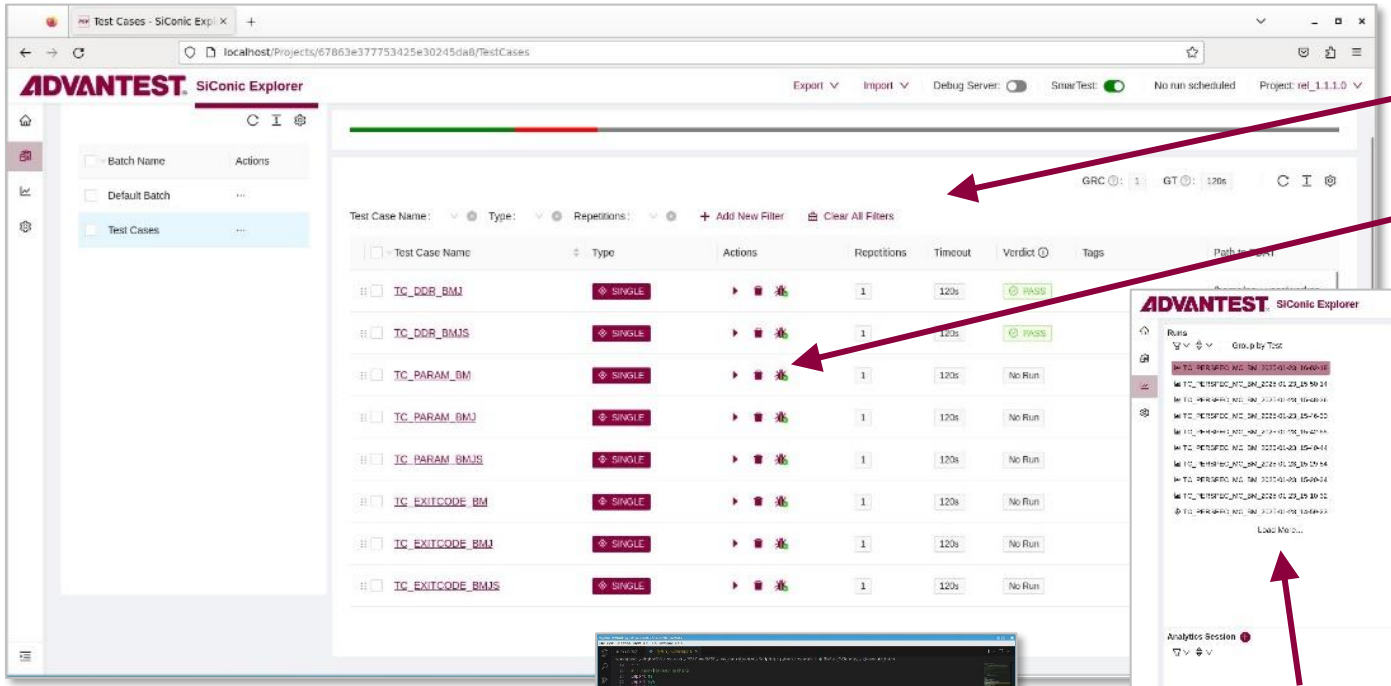
Overcoming Limitations of Typical Legay Tests in Lab. Environment (C-code)

Required Preparations:

- PSS TC with **SiConic PSS Lib.**
- HSIO use requires **OCM** (On-Chip Monitor)

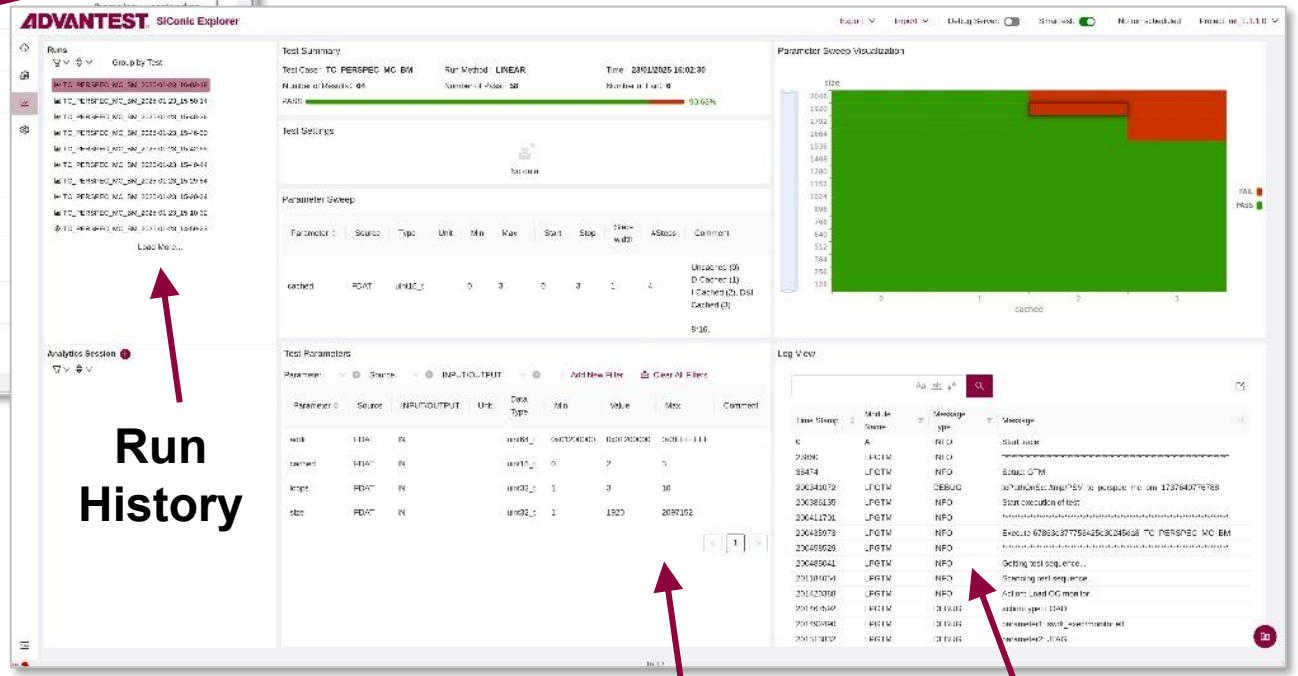
Features \ Test content	Typical Legacy Content	PSS
JTAG	✓	✓ OCM Not Needed
HSIO (USB / PCIe)	✗	✓ OCM Needed
PASS / FAIL	✓	✓
Runtime parameters	✗	✓
Result parameters	✗	✓
Activity Tracing	✗	✓ Buffered, Streamed
Environment (parameter) control	✓	✓
Sweep (Shmoo)	(✓) Only Env. Param's	✓ Runtime & Environ.

Solution Overview – *SiConic Explorer* for DV/SV Engineers



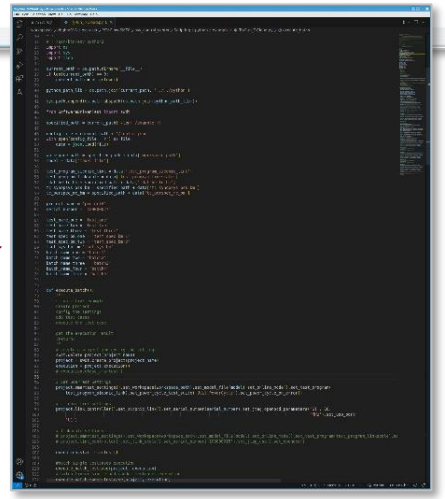
Test List (“Batch”)

Test Actions: e.g. Start Software Debugger



Setup & Execute

Run History



Python Scripting

Automate

Visualize & Explore

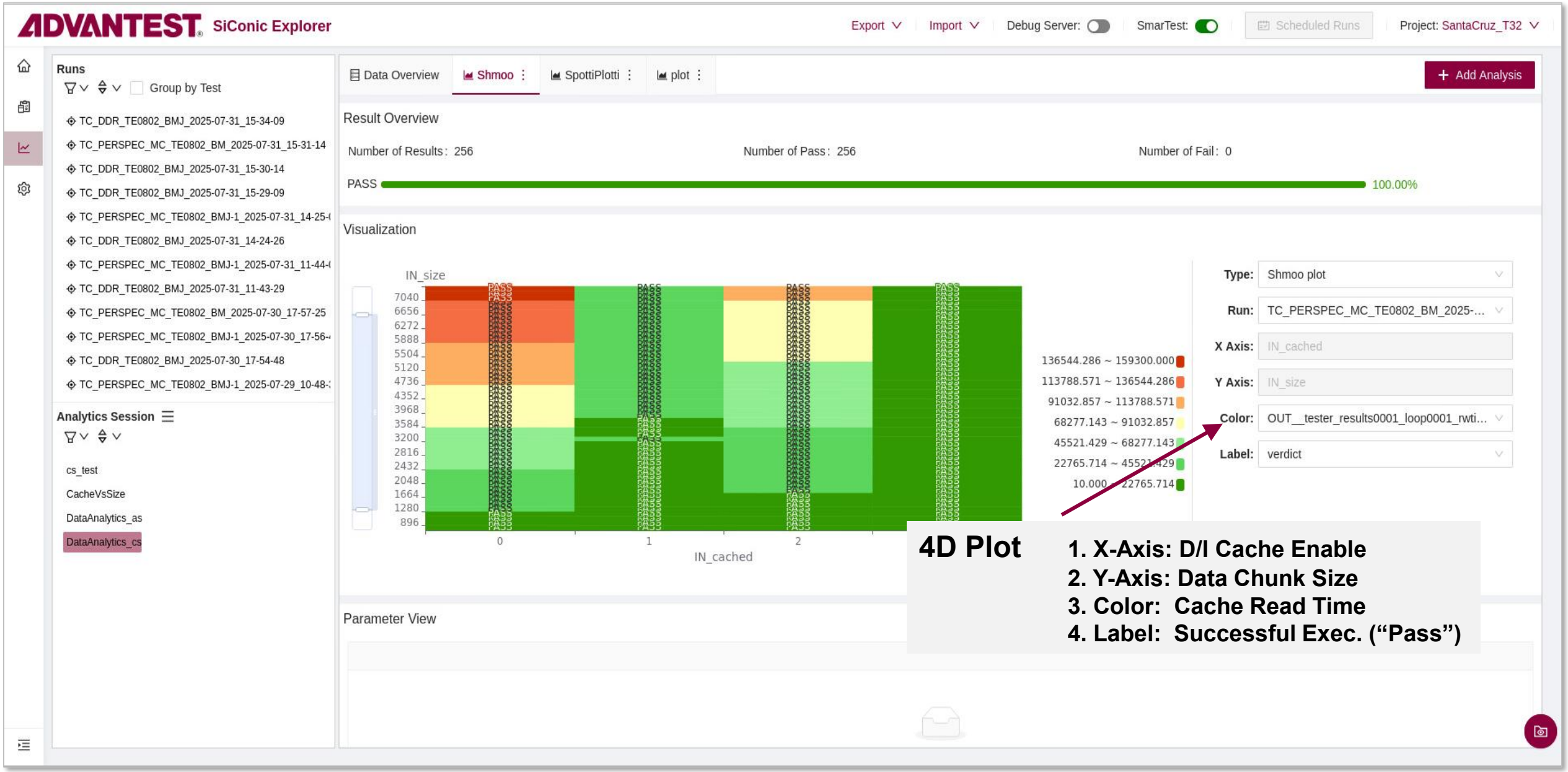
Parameters, Results

Activity Trace

accellera SYSTEMS INITIATIVE™

2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

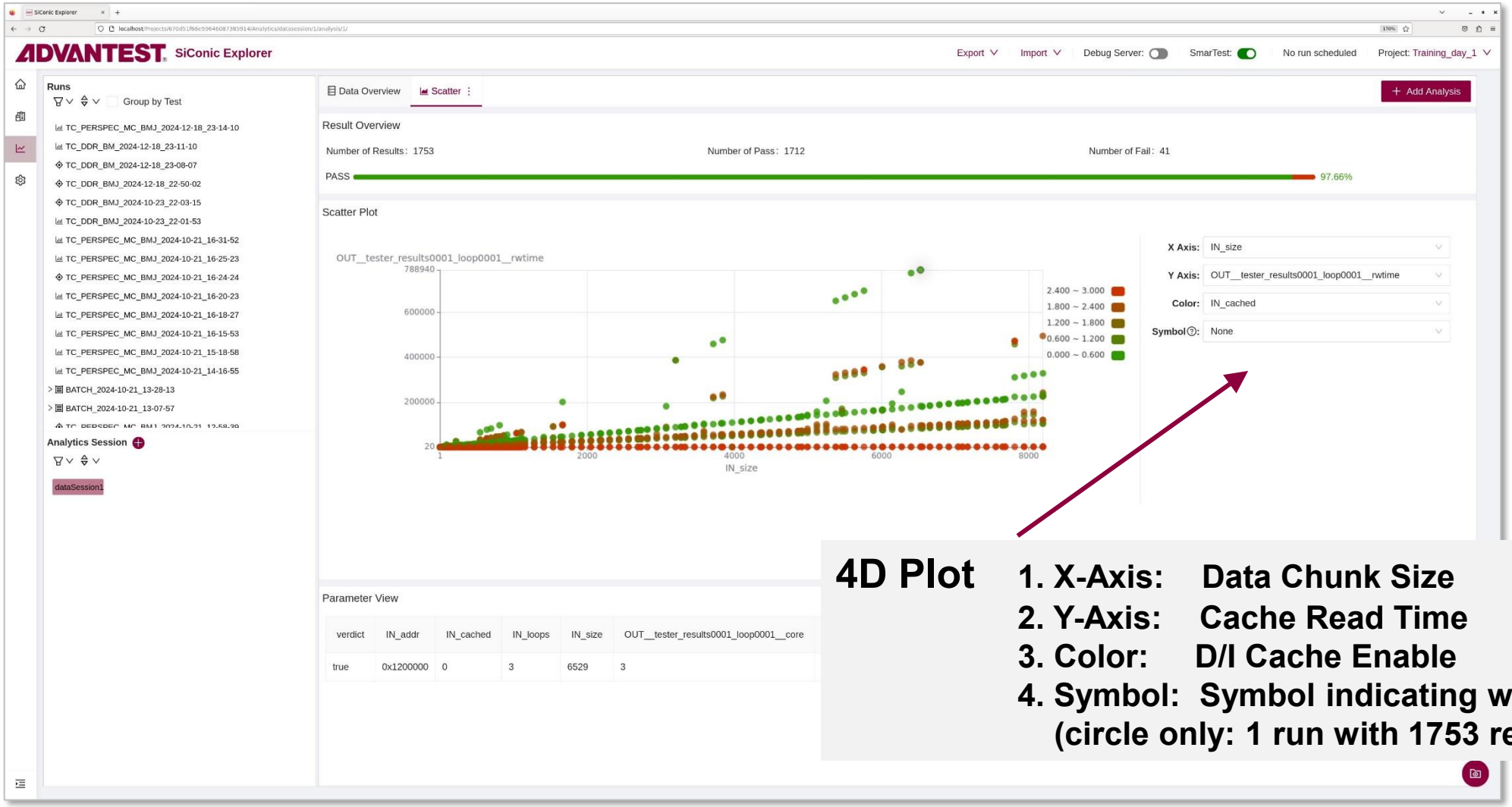
SiConic Explorer – Sweep (aka Shmoo) Analysis



4D Plot

1. **X-Axis: D/I Cache Enable**
2. **Y-Axis: Data Chunk Size**
3. **Color: Cache Read Time**
4. **Label: Successful Exec. (“Pass”)**

SiConic Explorer – Analysis in Scatter Plot



4D Plot

1. **X-Axis: Data Chunk Size**
2. **Y-Axis: Cache Read Time**
3. **Color: D/I Cache Enable**
4. **Symbol: Symbol indicating which run (circle only: 1 run with 1753 results)**

Solution Overview – *SiConic Link* interfacing with DUT/EVB

SiConic Link

Dimensions

plus
HP Workstation

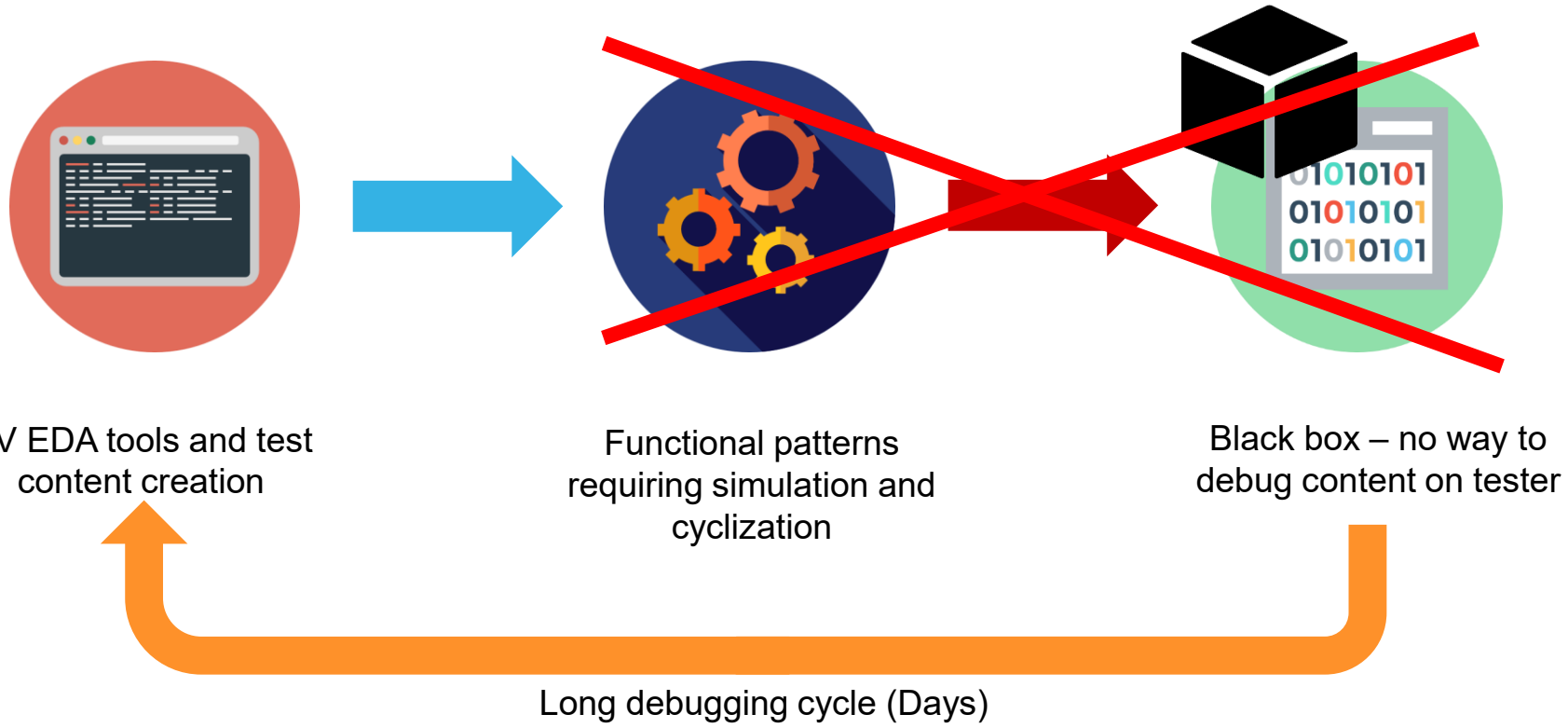
plus
DUT on Evaluation Board

Versatile – High Bandwidth – Extendable

High-Speed interfaces	PCIe Interfaces (Root Complex or Endpoint) USB Interfaces
Control Interface, GPIOs	GPIO ports: JTAG/SWD/GPIO/UART/SPI/... SW controlled relays (“push buttons” on eval. board)
Extension slots	For additional interfaces or T&M equipment
COM Computer-on-Module	High Perf. x86 COM, 128GByte DRAM
Lab. Infrastructure	Basic lab infrastructure req.: air cooled, 110V/220V Small formfactor and weight for bench and stacking

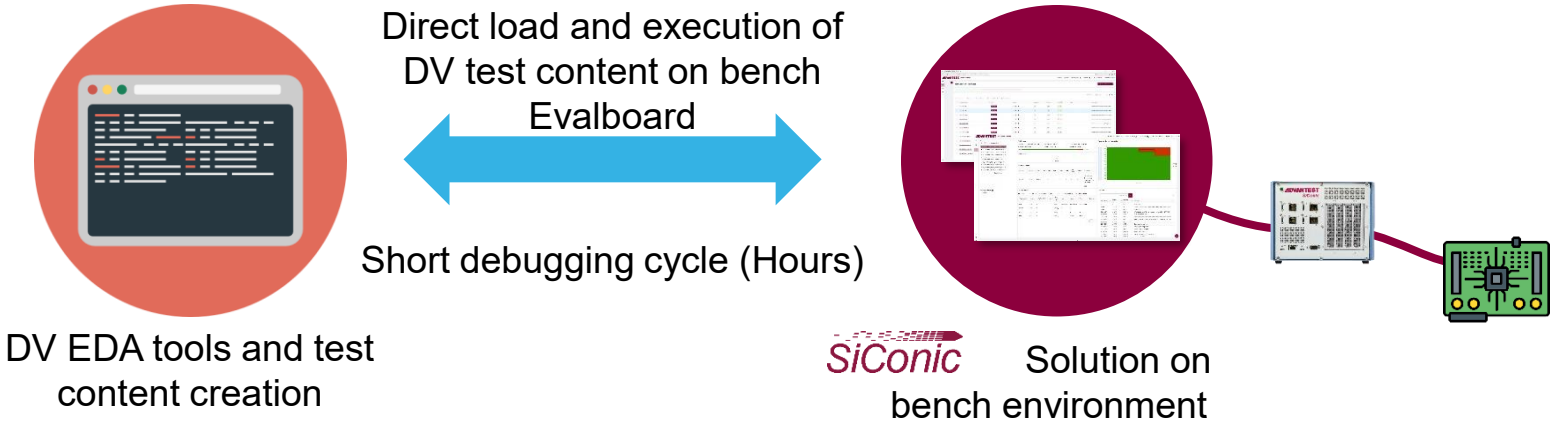
Customer success case

Challenge



Customer success case

Solution



Benefits

- Faster and More Efficient Test Execution and Debugging
- Improved Collaboration and Seamless Test Handover Between DV and Test: e.g. differentiate design / non-design issues
- Through a Standard Solution, SiConic brings Scalability and Reusability Across Projects

⇨ 75% reduced time to bring-up software-driven functional test
⇨ Avoid lengthy root-cause analysis (hunting) – delaying tape out of new silicon

SiConic's Unified Ecosystem

Design Verification

Design-for-Test

Silicon Validation

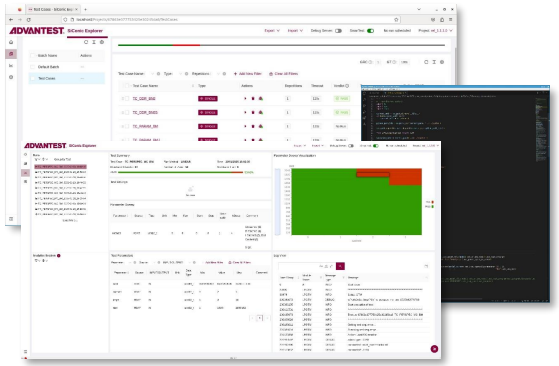
Test Engineering

cadence

SIEMENS

SYNOPSYS®

LAUTERBACH DEVELOPMENT TOOLS



SiConic Explorer

ADVANTEST SiConic

Unified, Automated and Scalable Ecosystem

ActivATE™

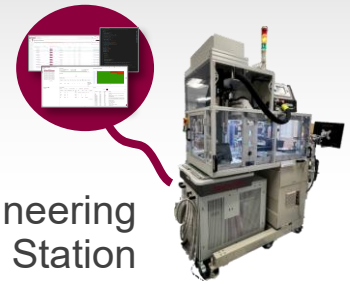
SmarTEST 8

Tester Operating System and Development Env.

Emulation / Prototyping

SiConic Link

Lab Engineering



Engineering Station

Production Preparation

accellera SYSTEMS INITIATIVE™

2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA


ADVANTEST®

Thank You!

Any Questions?

Meet Advantest team at Booth#30