Addressing SOC/IP Verification Framework Creation with UVM Centric Mechanisms

Adiel Khan, Amit Sharma

Synopsys

accellera SYSTEMS INITIATIVE™

2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

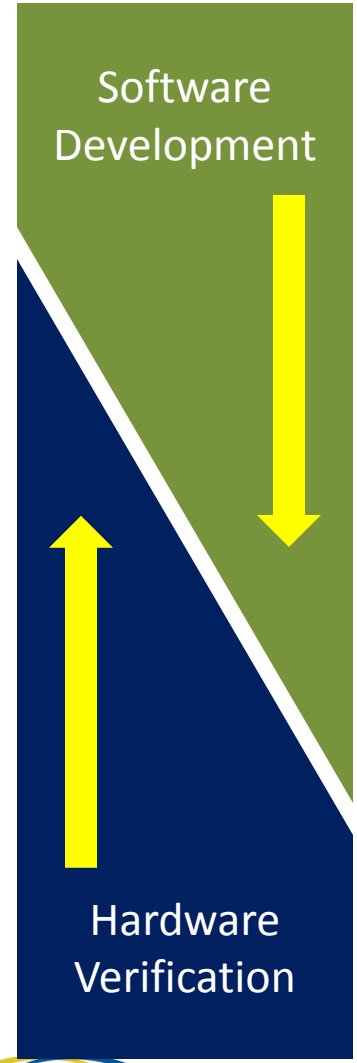
Agenda

- IP SOC Verification Tests/Tasks
- Quick UVM Introduction and Typical Usage scenarios
- Using UVM for mixed-AMS SOC verification
- Leveraging UVM TLM for System Level Verification
- Unified Register modeling between C and SV
- Enabling an UVM Testbench for simulation acceleration

Verification & Test Tasks

Verification Tasks	Block Sim	SoC Sim	Gate Sim	Emulation	Silicon
Connectivity	CRV Interface Protocol Compliance	SW Register Sweep Reset Registers	SW Limited Reg Sweep Reset Some Registers	Full Register Sweep Multiple Configs	Full Register Sweep Multiple
Functionality	CRV Block Level Functional Bugs	BFM CRV SW Simple Block to Block	SW Limited Functionality Minimal Boot Code	SW Driver Functionality Bring Up Tests	C/SWSilicon Bringup Bring Up Tests
Performance	CRV Perf SW Perf Back to back frames	SW Bare Metal OS Block to Block	None	Performance Test Verify Peak Performance	Boot OS Applications Performance Stress Tests
Software	BFM C/VPI tests Initialize + Basic Driver	BFM + VPI/C C/Bare OS Initialize + Basic Driver	SW Limited Functionality Initialize + Small Test	Boot OS Applications Real Software	Boot OS Applications Real Software

HW Based & SW Based Methodology



Software Team & Skills move down towards hardware

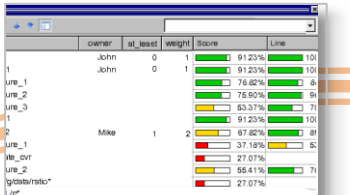
• Software Development • Software Drivers • Application Software	• Virtual Platform • Hardware Prototype • Silicon
• SoC driver testing • Bare Metal OS • Minimal Real OS	• Emulation • Test runs on embedded CPU
• SoC Register sweeps • SoC Performance • SoC Bare Metal OS	• RTL Simulation • Test runs on embeded RTL CPU
• RTL Block Verification • SoC Connectivity • SoC Block Traffic	• RTL Simulation • CPU is Bus BFM • UVM Methodology

Hardware Team & Skills move up towards software

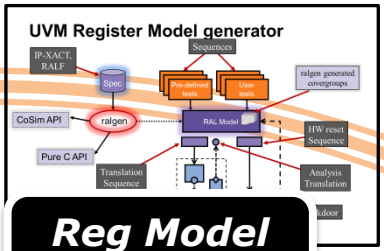
← Mix depends on team skills →

UVM INTRODUCTION

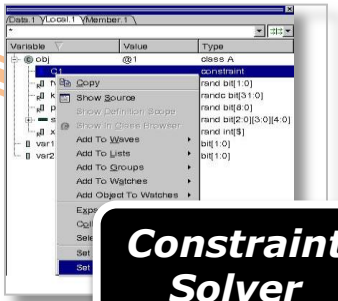
Complete UVM Ecosystem



Verification Planning

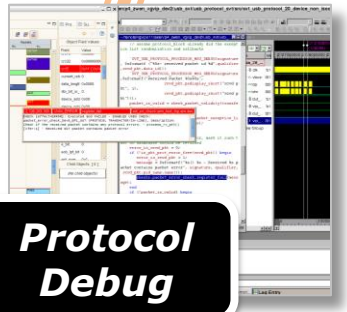


Reg Model generators



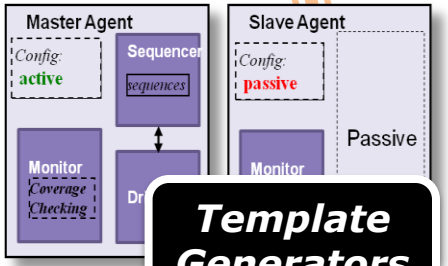
Constraint Solver

Coverage & Analysis

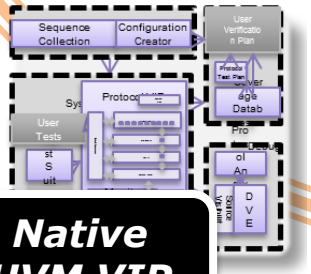


Protocol Debug

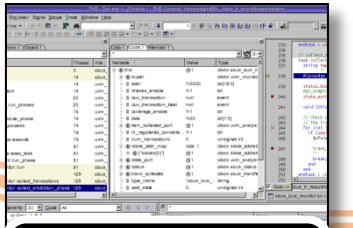
UVM



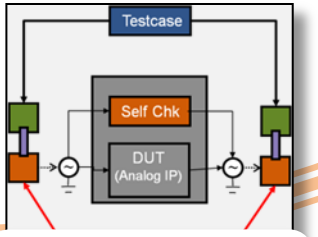
Template Generators



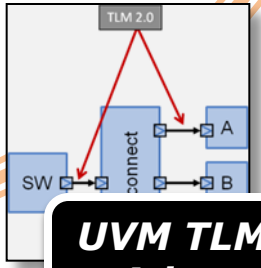
Native UVM VIP



UVM-Aware Debug



UVM AMS Testbench



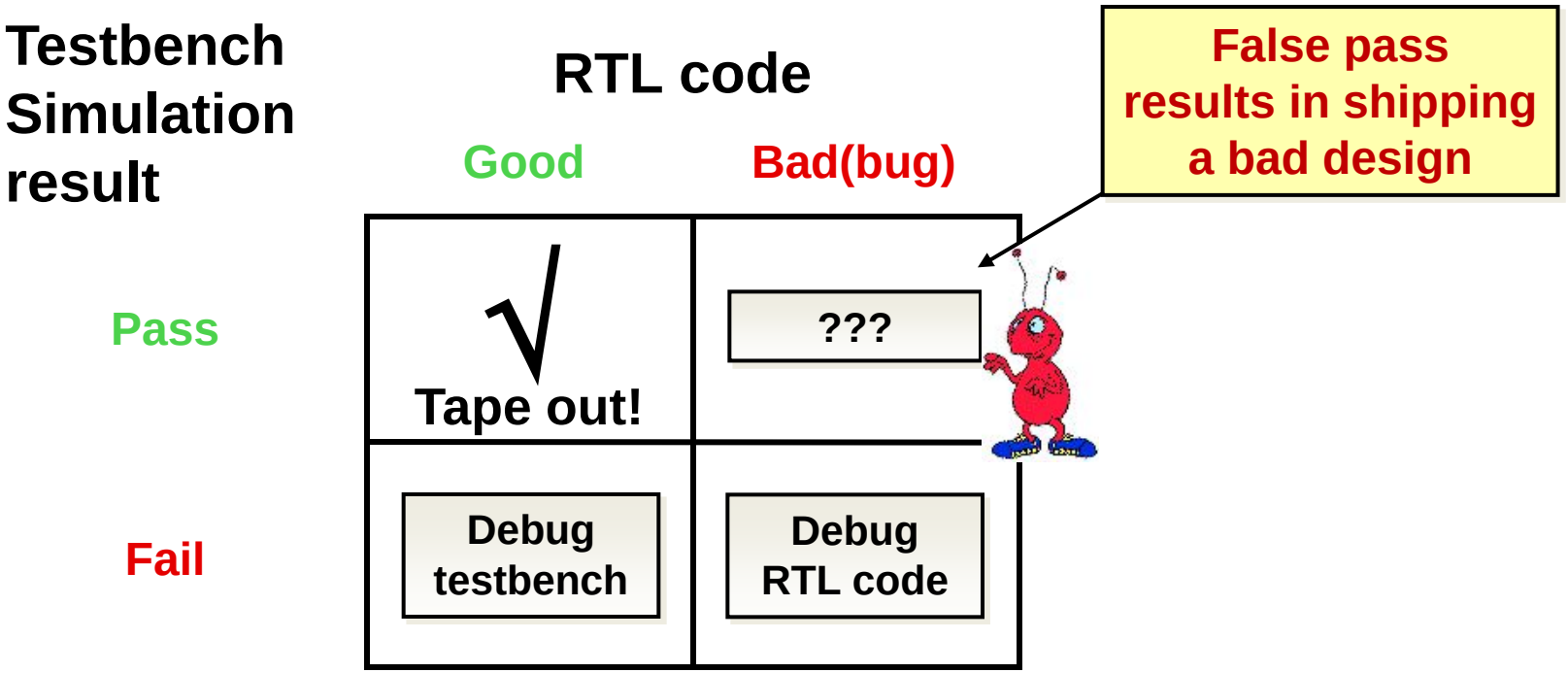
UVM TLM/SC Adaptors

accellera SYSTEMS INITIATIVE™

2015 CONFERENCE AND EXHIBITION INDIA

Verification Goal

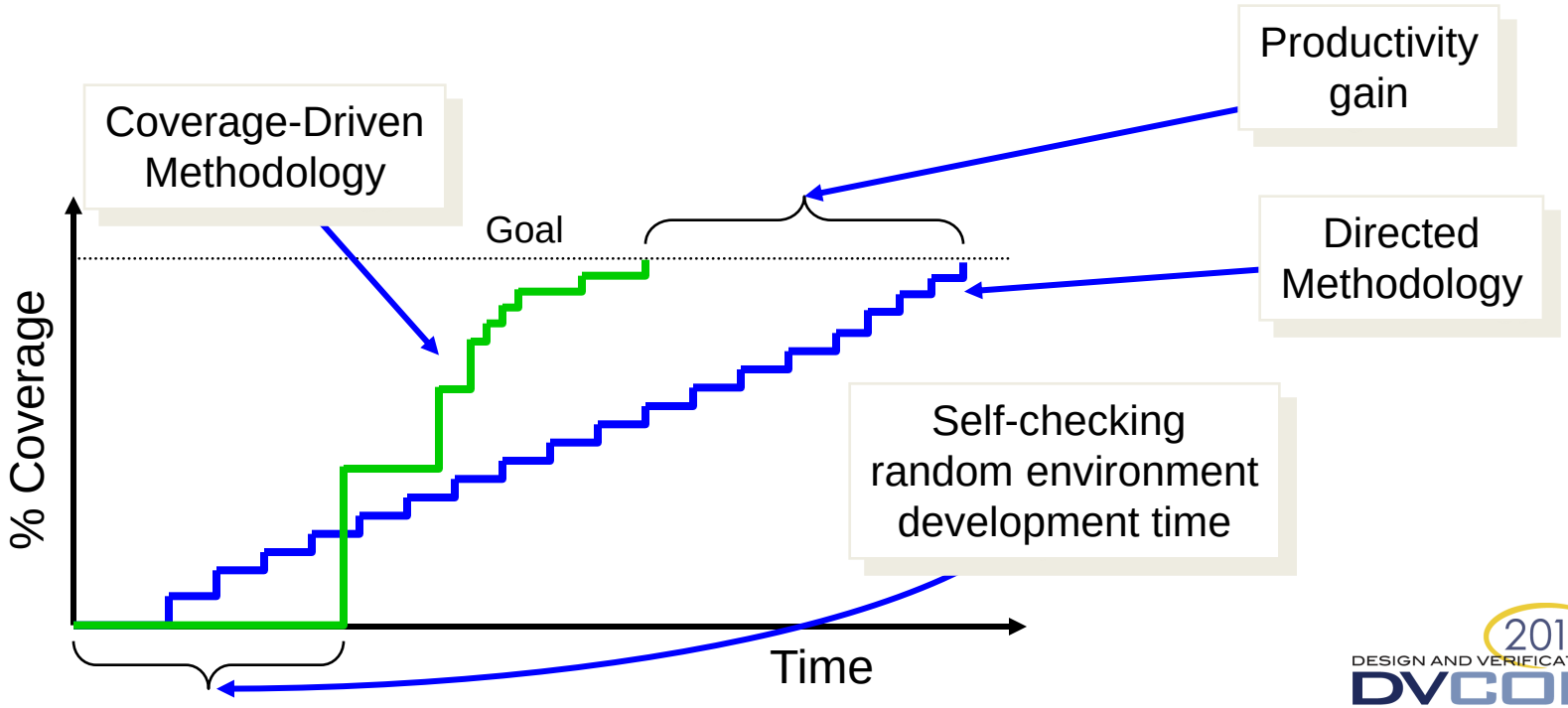
- Ensure full conformance with specification:
  - Must avoid **false** passes



How do we achieve this goal?

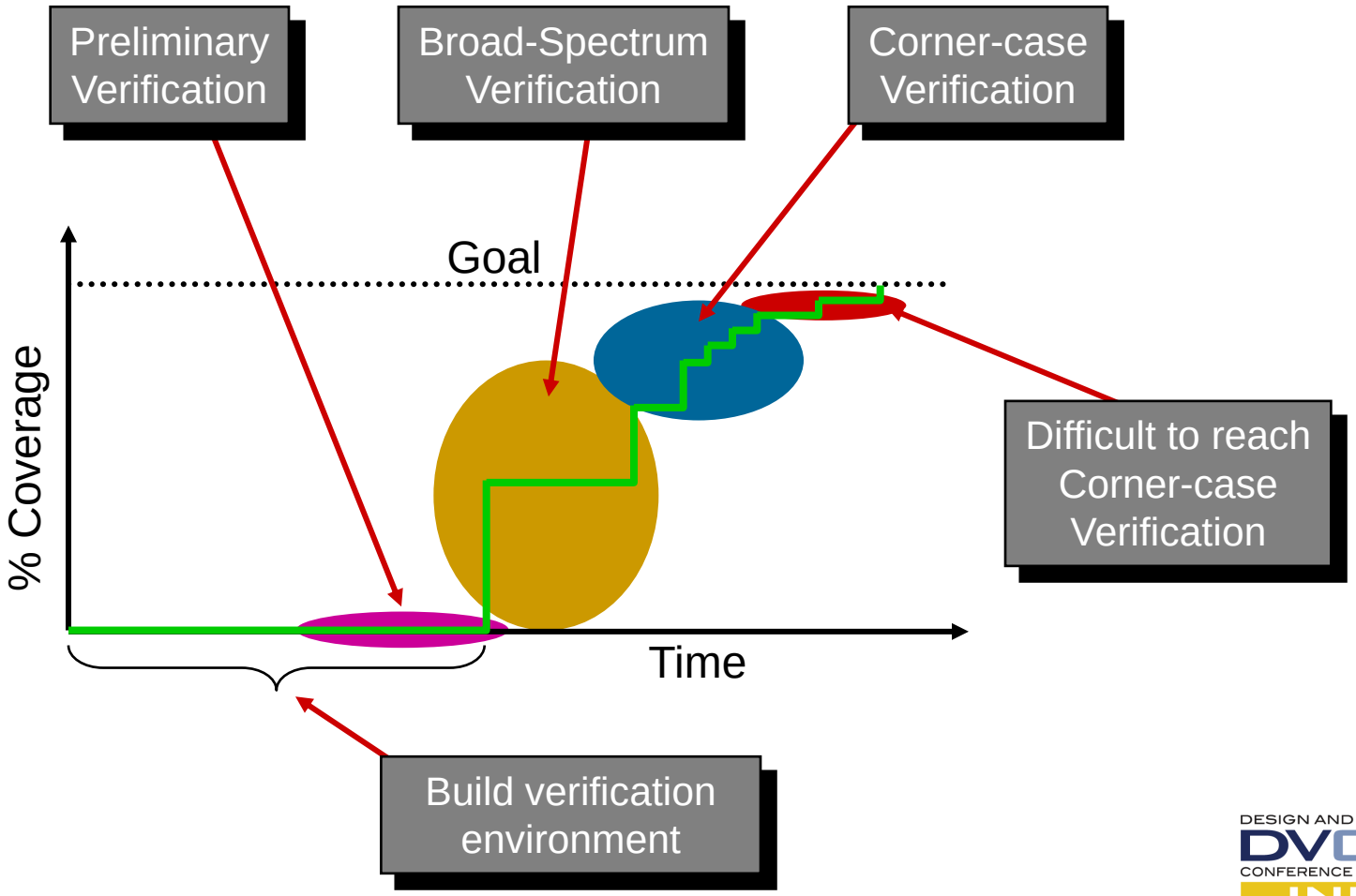
Coverage-Driven Verification

- Focus on uncovered areas
- Trade-off authoring time for run-time
- Progress measured using functional coverage metrics



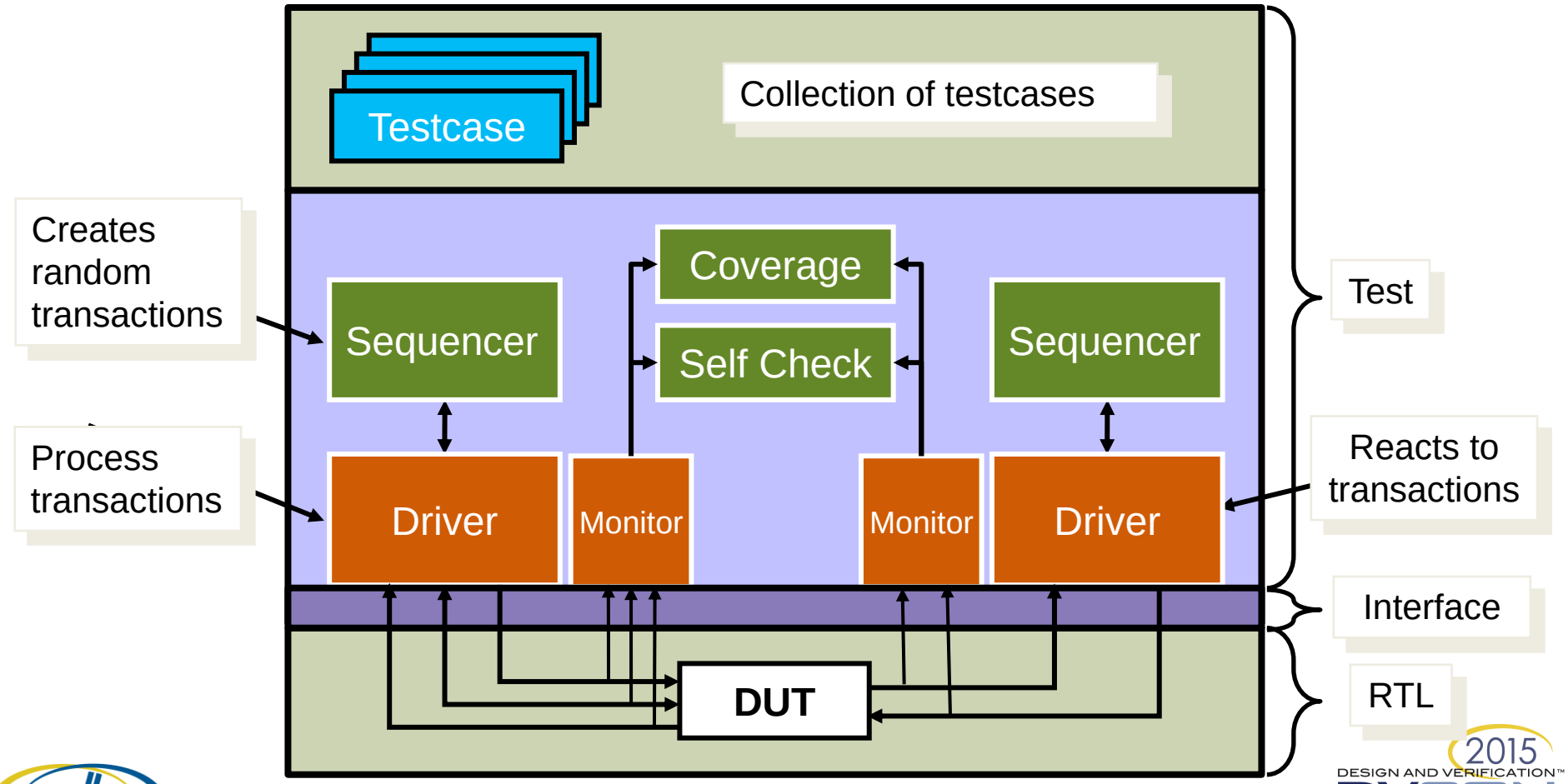
Phases of Verification

Start with fully random environment. Continue with more and more focused guided tests

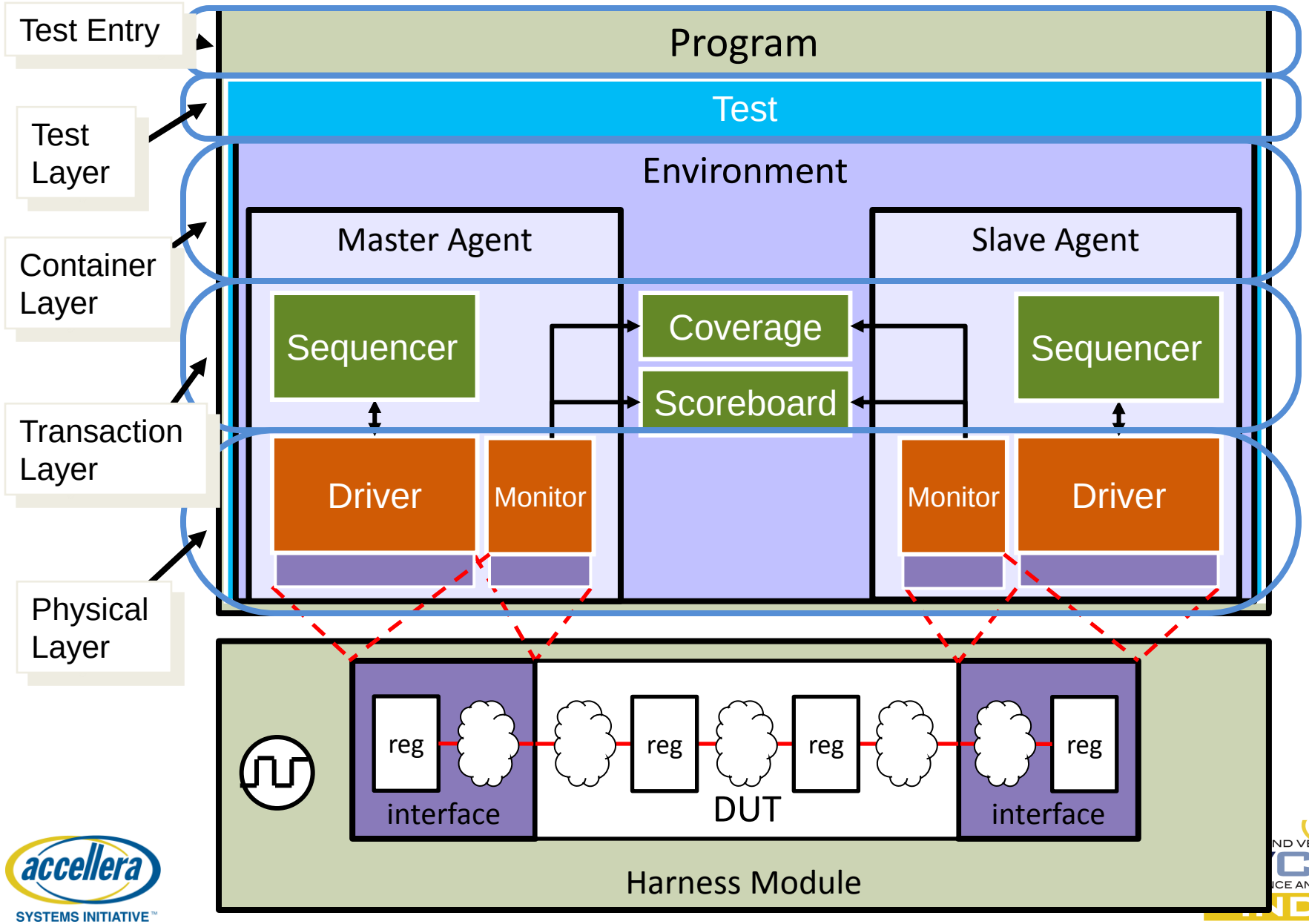


Typical Testbench Architecture

- SystemVerilog testbench structure



UVM Testbench Architecture



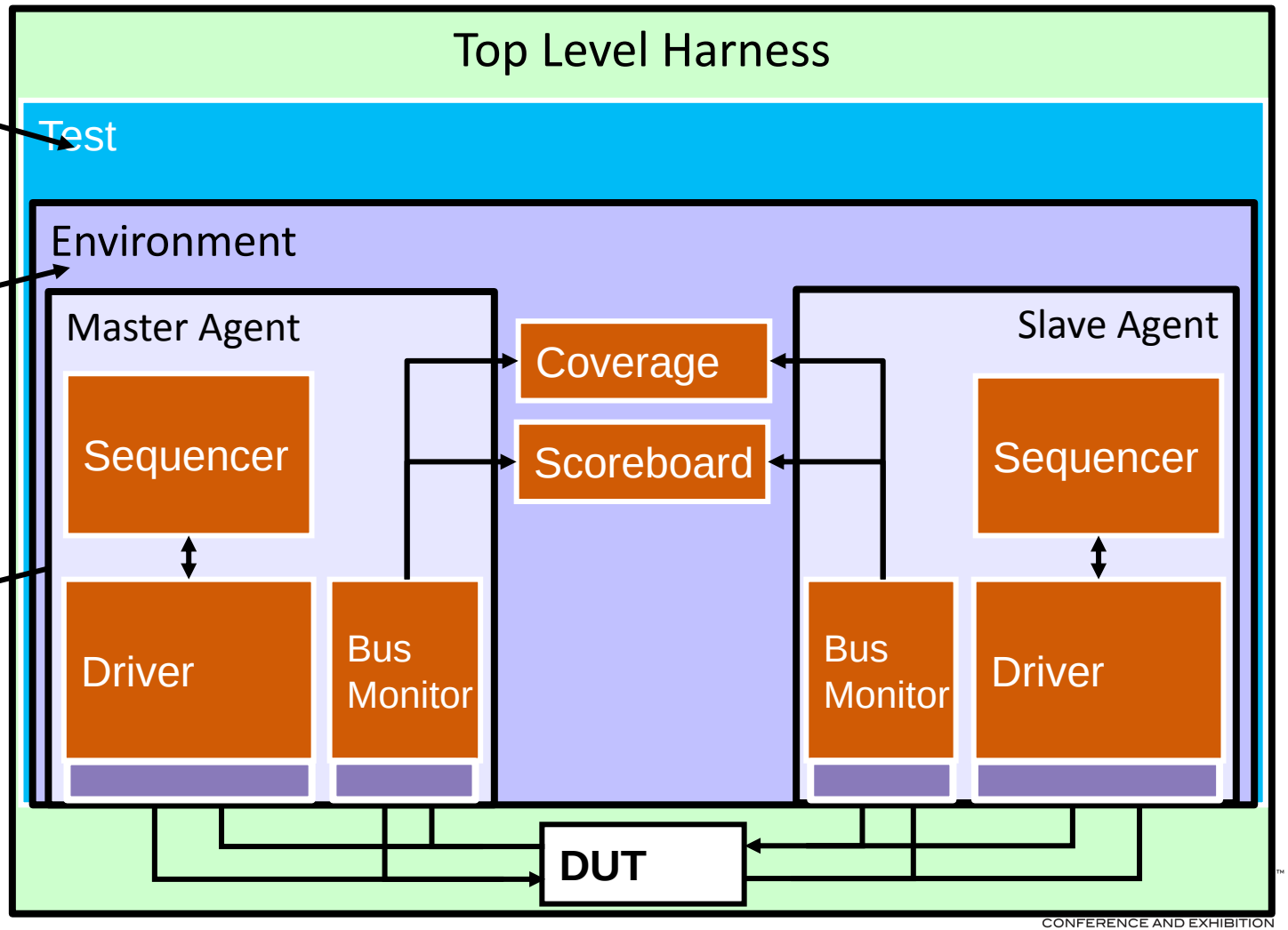
UVM Encourages Encapsulate for Reuse

- Structure should be architected for reuse

Test instantiates the environment and modifies the environment on a testcase by testcase basis

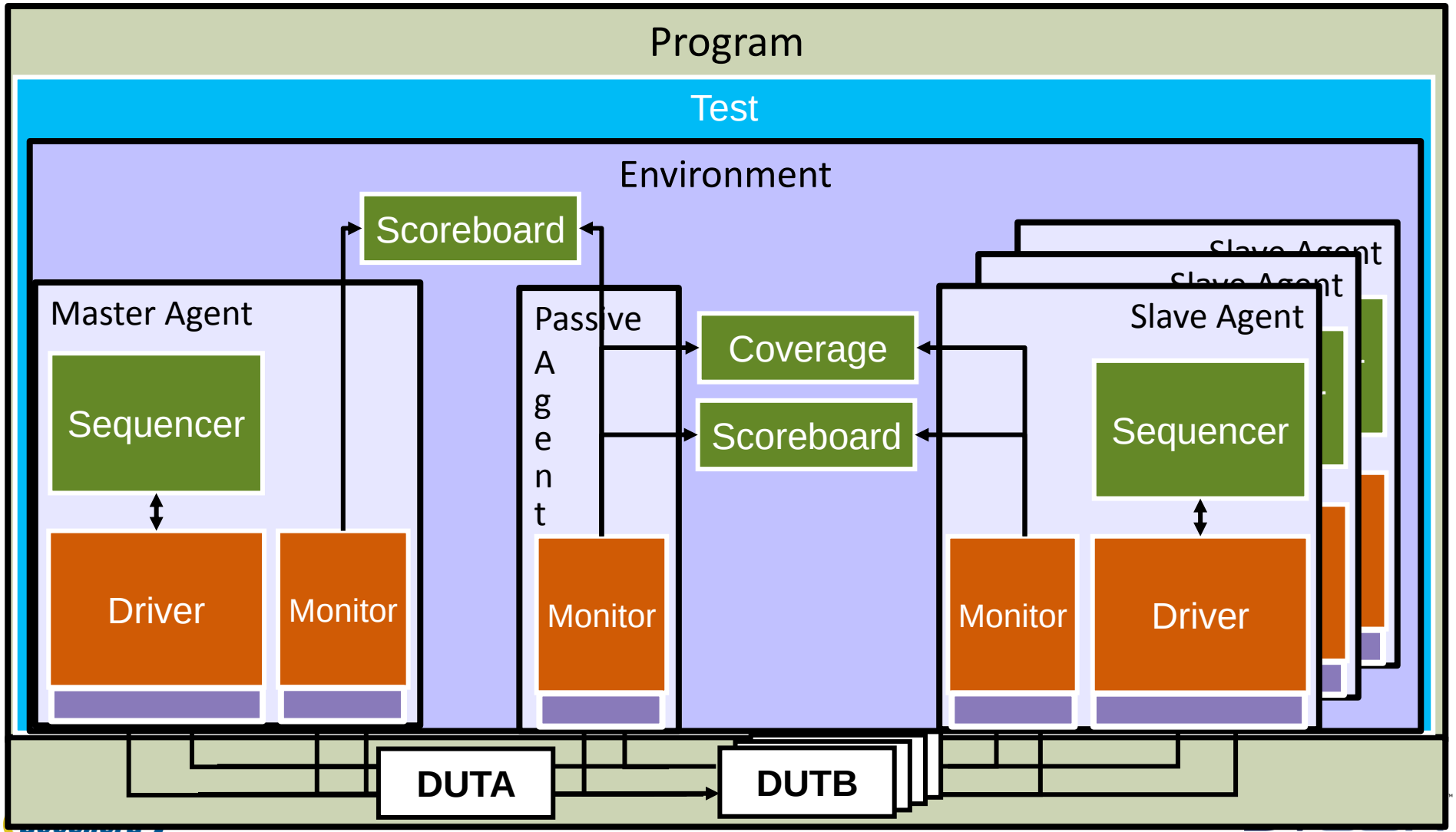
Agents, coverage and scoreboard should be encapsulated in an environment

Sequencer, driver and monitor associated with an interface should be encapsulated as an agent for that interface



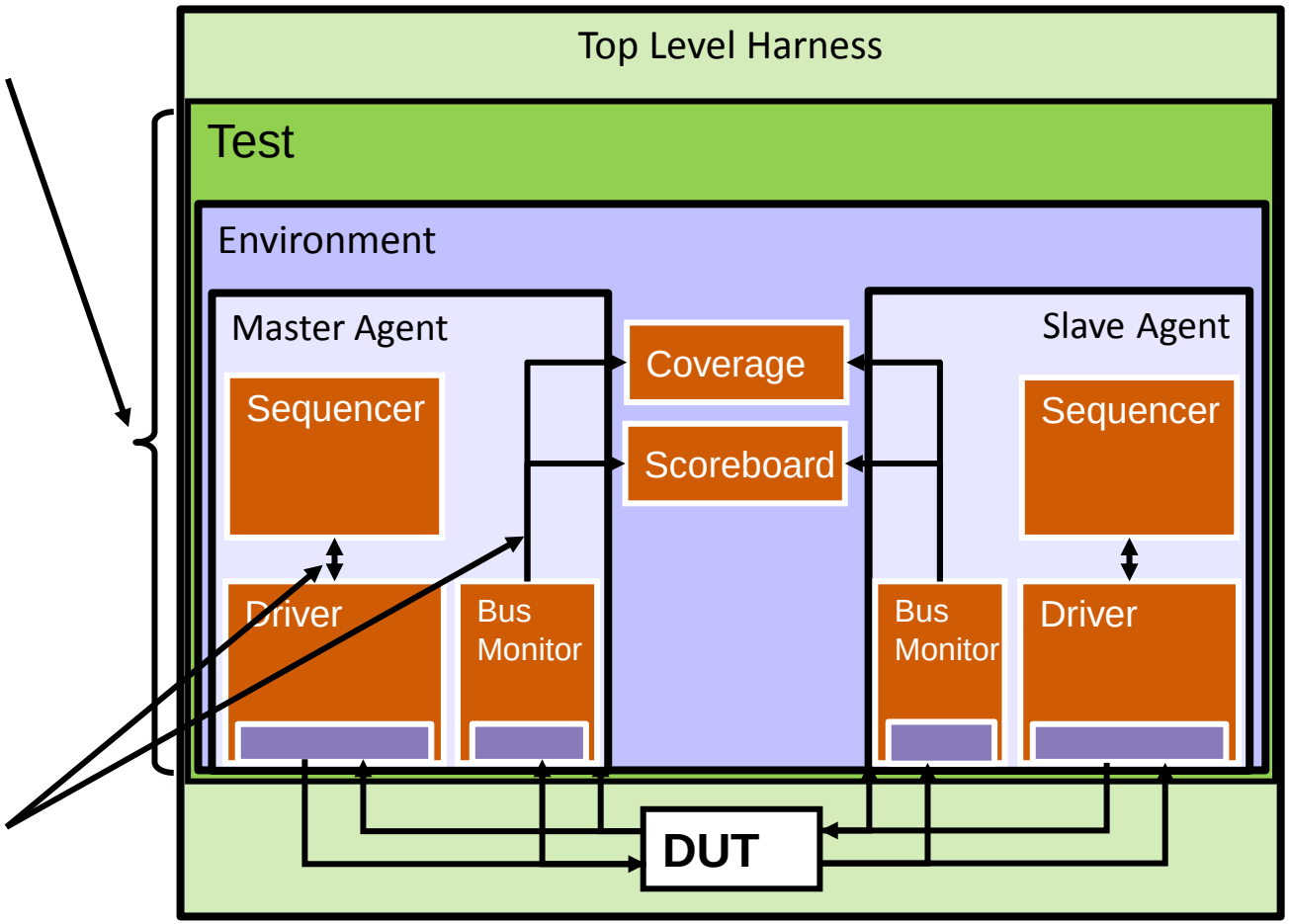
UVM Structure is Scalable

- Agents are the building blocks across test/projects



Structural Support in UVM

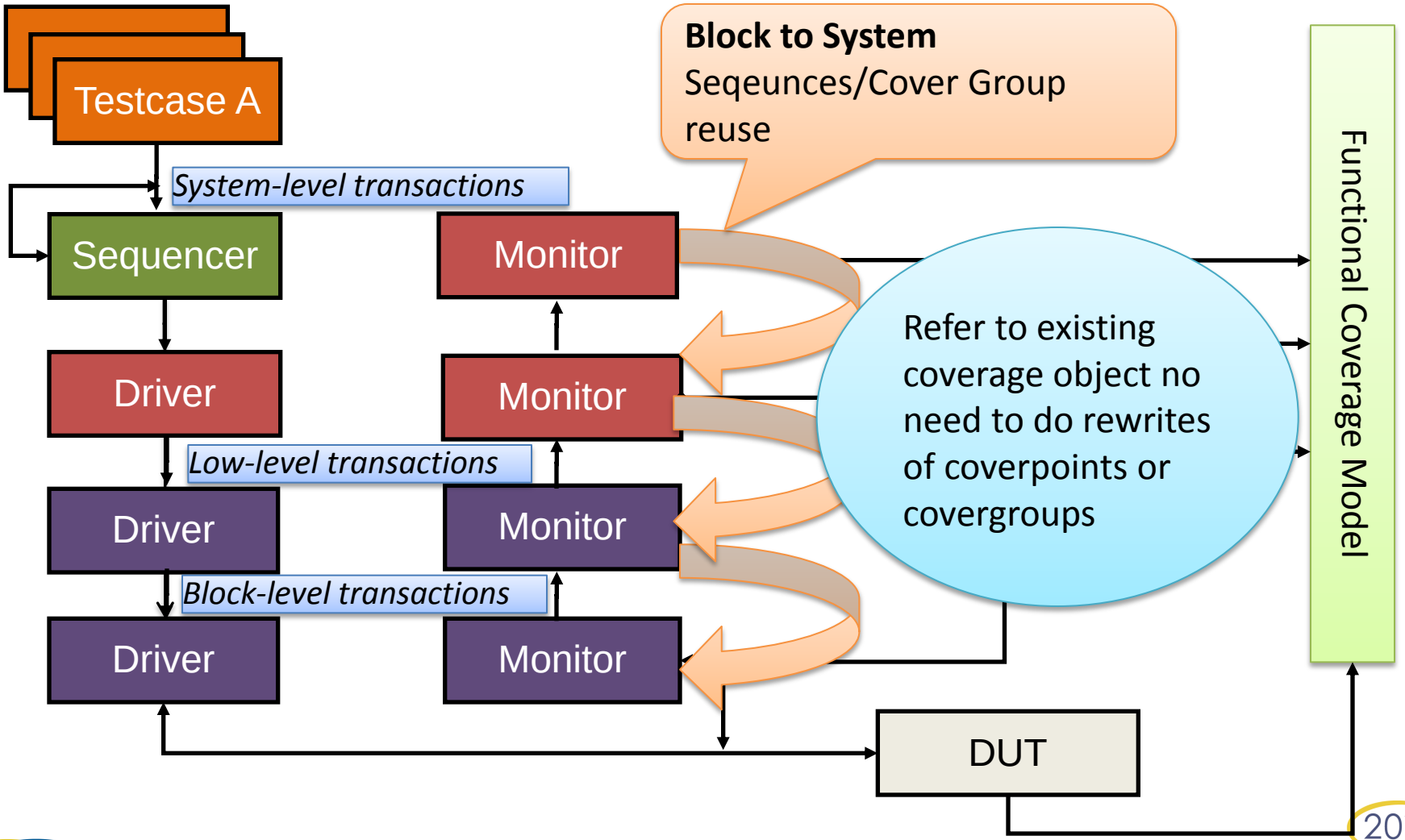
- Structural & Behavioral
  - `uvm_component`
    - `uvm_test`
    - `uvm_env`
    - `uvm_agent`
    - `uvm_sequencer`
    - `uvm_driver`
    - `uvm_monitor`
    - `uvm_scoreboard`
- Communication
  - `uvm_*_port`
  - `uvm_*_socket`
- Data
  - `uvm_sequence_item`



Typical UVM Usage scenarios

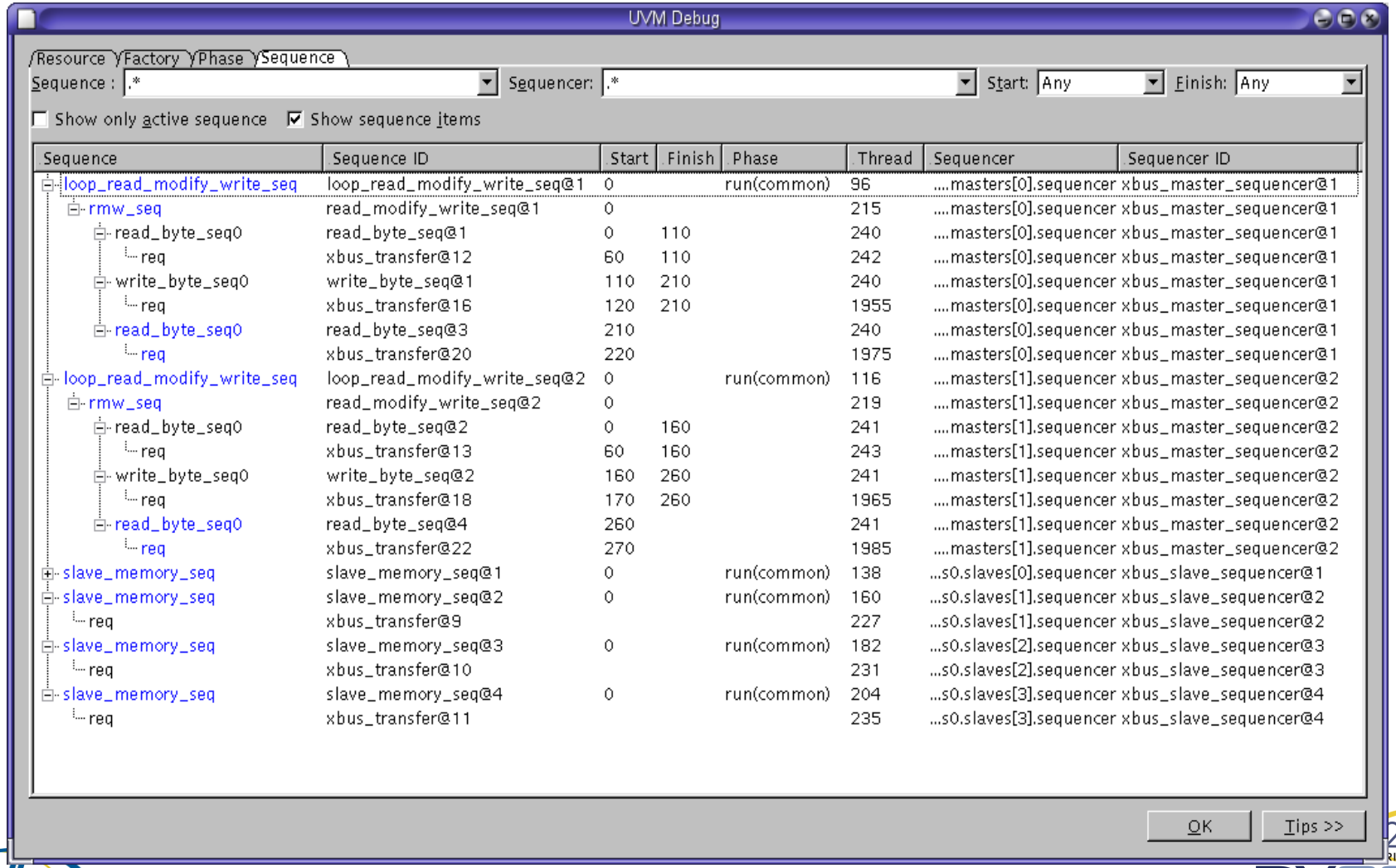
- UVM based Constrained Random Verification (CRV) for RTL Block/IP Verification using
  - ✓ UVM based VIPs with Constrained Random tests
  - ✓ Usage of standard 3rd Party UVM VIPs in IP/Subsystem Verification
- Generating SoC Block Traffic
- IP verification using Re-use of IP level testbench at SOC/System level
  - UVM VIPs as peripherals
  - UVM VIPs as passive monitors
  - Reuse of UVM Register layer from block to SOC level

UVM Coverage Driven Verification



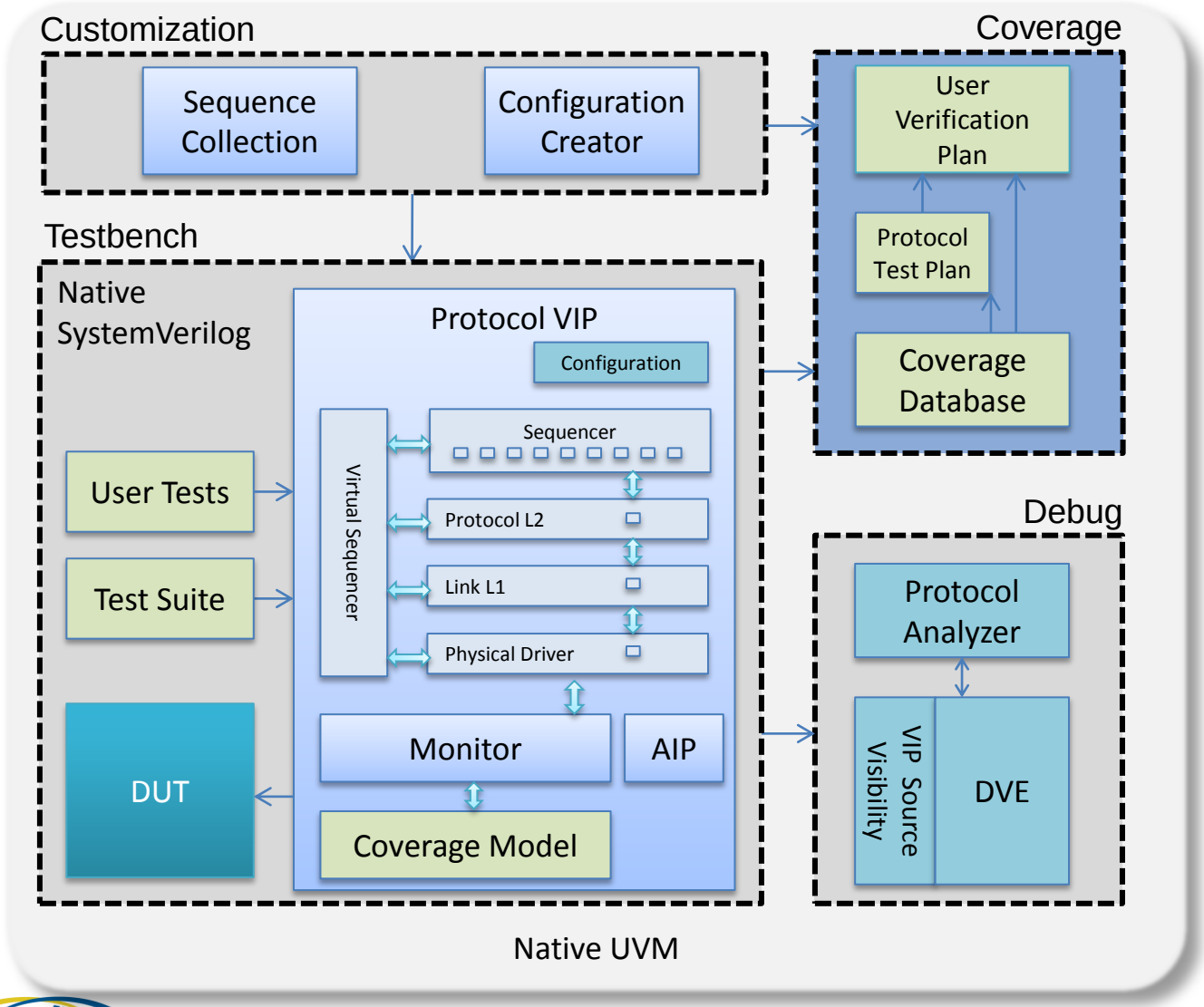
UVM Constrained Random Verification

Quick tracing of sequences through phases and sequencers



Sequence	Sequence ID	Start	Finish	Phase	Thread	Sequencer	Sequencer ID
loop_read_modify_write_seq	loop_read_modify_write_seq@1	0		run(common)	96	....masters[0].sequencer	xbus_master_sequencer@1
rmw_seq	read_modify_write_seq@1	0			215	....masters[0].sequencer	xbus_master_sequencer@1
read_byte_seq0	read_byte_seq@1	0	110		240	....masters[0].sequencer	xbus_master_sequencer@1
req	xbus_transfer@12	60	110		242	....masters[0].sequencer	xbus_master_sequencer@1
write_byte_seq0	write_byte_seq@1	110	210		240	....masters[0].sequencer	xbus_master_sequencer@1
req	xbus_transfer@16	120	210		1955	....masters[0].sequencer	xbus_master_sequencer@1
read_byte_seq0	read_byte_seq@3	210			240	....masters[0].sequencer	xbus_master_sequencer@1
req	xbus_transfer@20	220			1975	....masters[0].sequencer	xbus_master_sequencer@1
loop_read_modify_write_seq	loop_read_modify_write_seq@2	0		run(common)	116	....masters[1].sequencer	xbus_master_sequencer@2
rmw_seq	read_modify_write_seq@2	0			219	....masters[1].sequencer	xbus_master_sequencer@2
read_byte_seq0	read_byte_seq@2	0	160		241	....masters[1].sequencer	xbus_master_sequencer@2
req	xbus_transfer@13	60	160		243	....masters[1].sequencer	xbus_master_sequencer@2
write_byte_seq0	write_byte_seq@2	160	260		241	....masters[1].sequencer	xbus_master_sequencer@2
req	xbus_transfer@18	170	260		1965	....masters[1].sequencer	xbus_master_sequencer@2
read_byte_seq0	read_byte_seq@4	260			241	....masters[1].sequencer	xbus_master_sequencer@2
req	xbus_transfer@22	270			1985	....masters[1].sequencer	xbus_master_sequencer@2
slave_memory_seq	slave_memory_seq@1	0		run(common)	138	...s0.slaves[0].sequencer	xbus_slave_sequencer@1
slave_memory_seq	slave_memory_seq@2	0		run(common)	160	...s0.slaves[1].sequencer	xbus_slave_sequencer@2
req	xbus_transfer@9				227	...s0.slaves[1].sequencer	xbus_slave_sequencer@2
slave_memory_seq	slave_memory_seq@3	0		run(common)	182	...s0.slaves[2].sequencer	xbus_slave_sequencer@3
req	xbus_transfer@10				231	...s0.slaves[2].sequencer	xbus_slave_sequencer@3
slave_memory_seq	slave_memory_seq@4	0		run(common)	204	...s0.slaves[3].sequencer	xbus_slave_sequencer@4
req	xbus_transfer@11				235	...s0.slaves[3].sequencer	xbus_slave_sequencer@4

Enables Next-Generation VIP Architecture



Time to 1st Test
- Configuration Creation GUI
- Built-in test plan
- Sequence Collection

Time to Verify
- Native SystemVerilog
- No wrappers
- Up to 4x faster

Time to Debug
- Protocol-aware debug
- Source code visibility
- Error Diagnostics

Time to Coverage
- Built-in coverage
- Verification Plan
- Test Suite
- Sequence Library

Performance Validation for AXI Protocol Using UVM testbench

Measure bus bandwidth based on traffic profiles

Performance Requirements

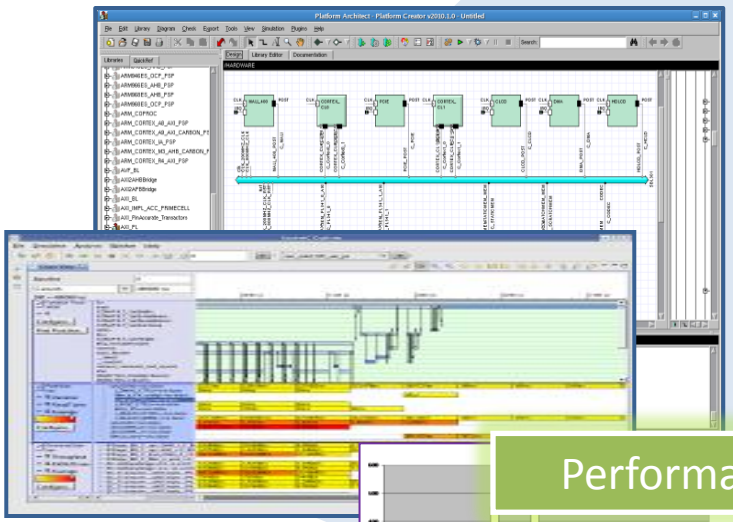
Specification

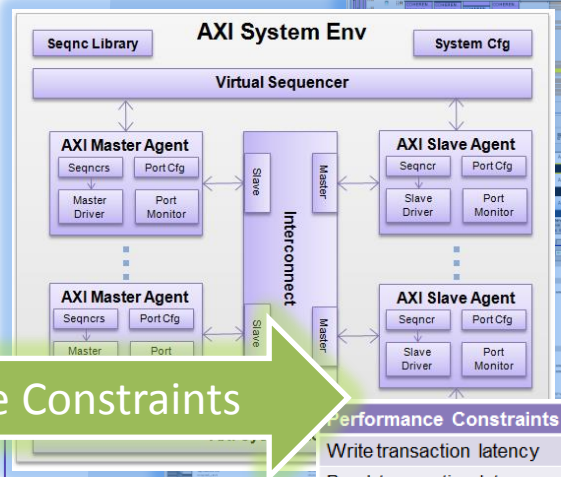
✓ ✓

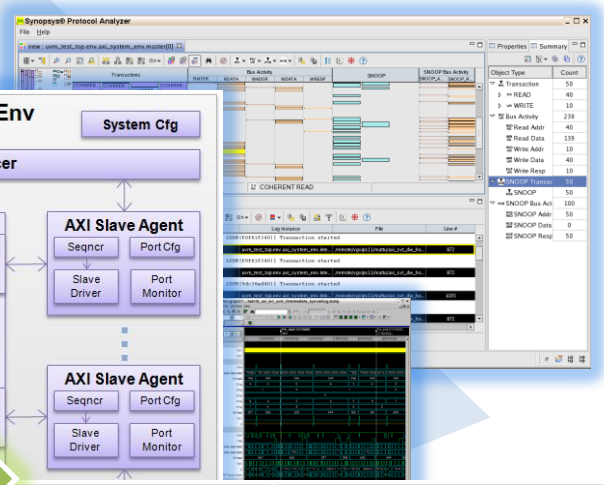
Final Silicon and Applications

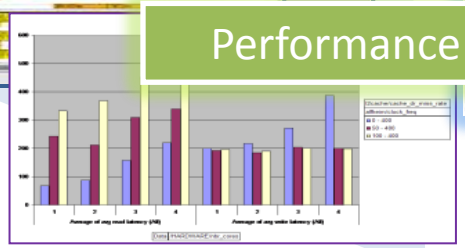
System Architects

Verification Engineers



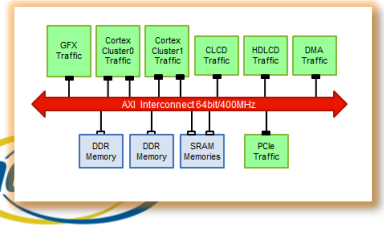






Performance Constraints

Performance Constraints	Settings
Write transaction latency	Min, Max, Avg latency
Read transaction latency	Min, Max, Avg latency
Write transaction throughput	Max, Min Number of Bytes/Time Unit
Read transaction throughput	Max, Min Number of Bytes/Time Unit



SYSTEMS INITIATIVE™

2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Performance Analysis

- User sets
  - Performance constraints
  - Recording interval
- Registration using UVM callbacks
- UVM_ERROR for violation
- Dynamically Configurable
- Prints Performance Summary

****

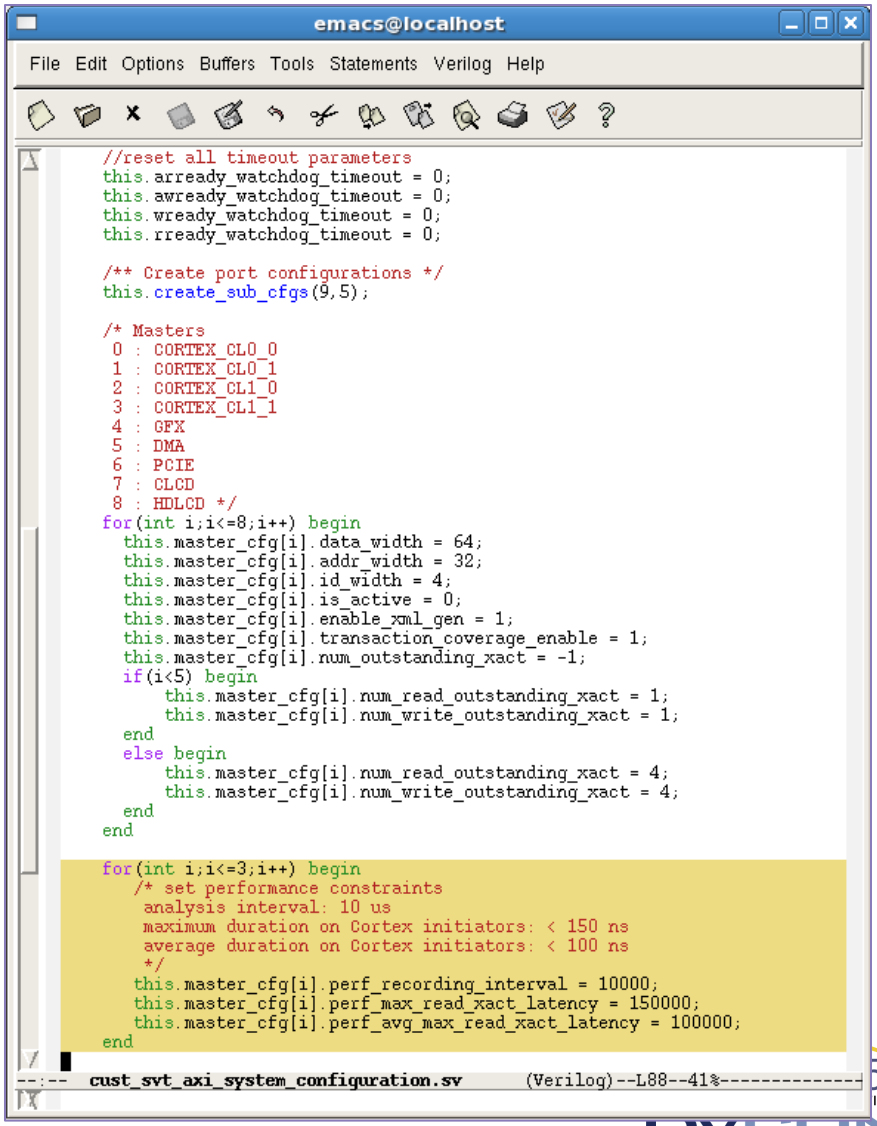
PERFORMANCE REPORT FOR MASTER 0:

****

==========

Interval Start Time:0.000000; Interval End Time: 5425.000000
Configured avg_max read xact latency:15000.000000;
Observed avg_max read xact latency: 360.000000

==========



Metric Compliance Checking

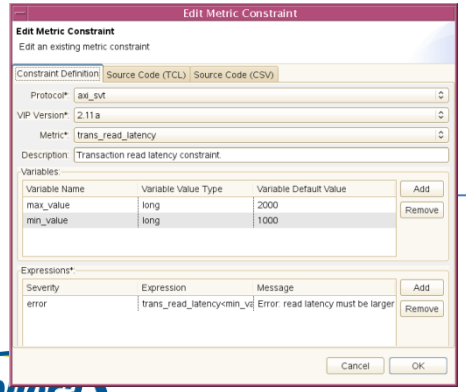
- Visualize Performance Data in Graphs/charts/spreadsheets
- Performance constraints can be fed into spreadsheets/EDA tools for checking performance data

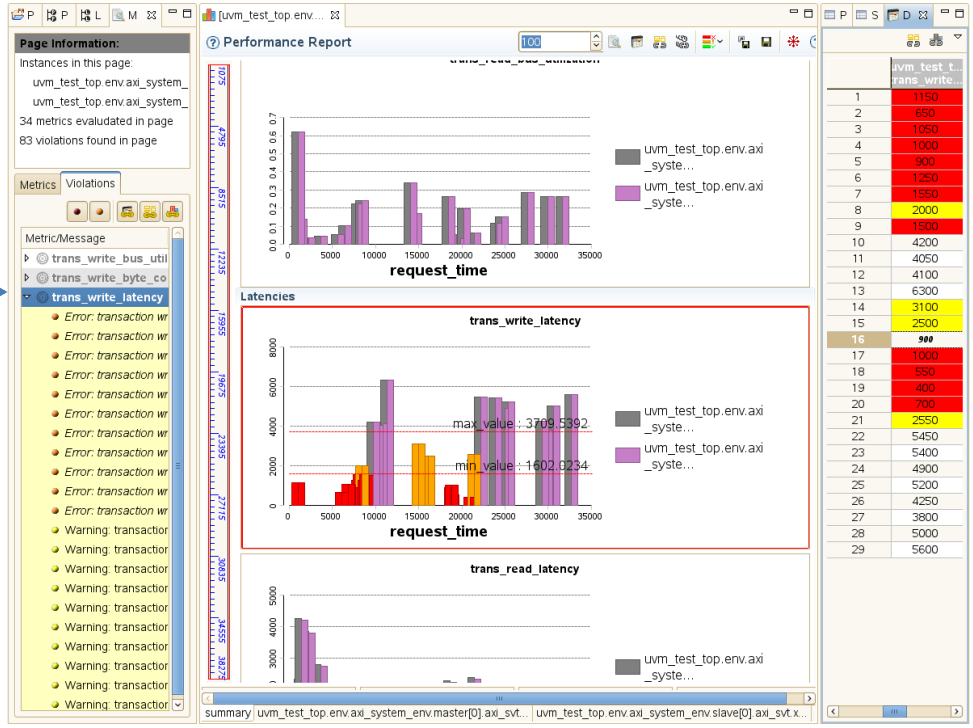
TCL File:

CSV File:

Apply

Constraint Editor:



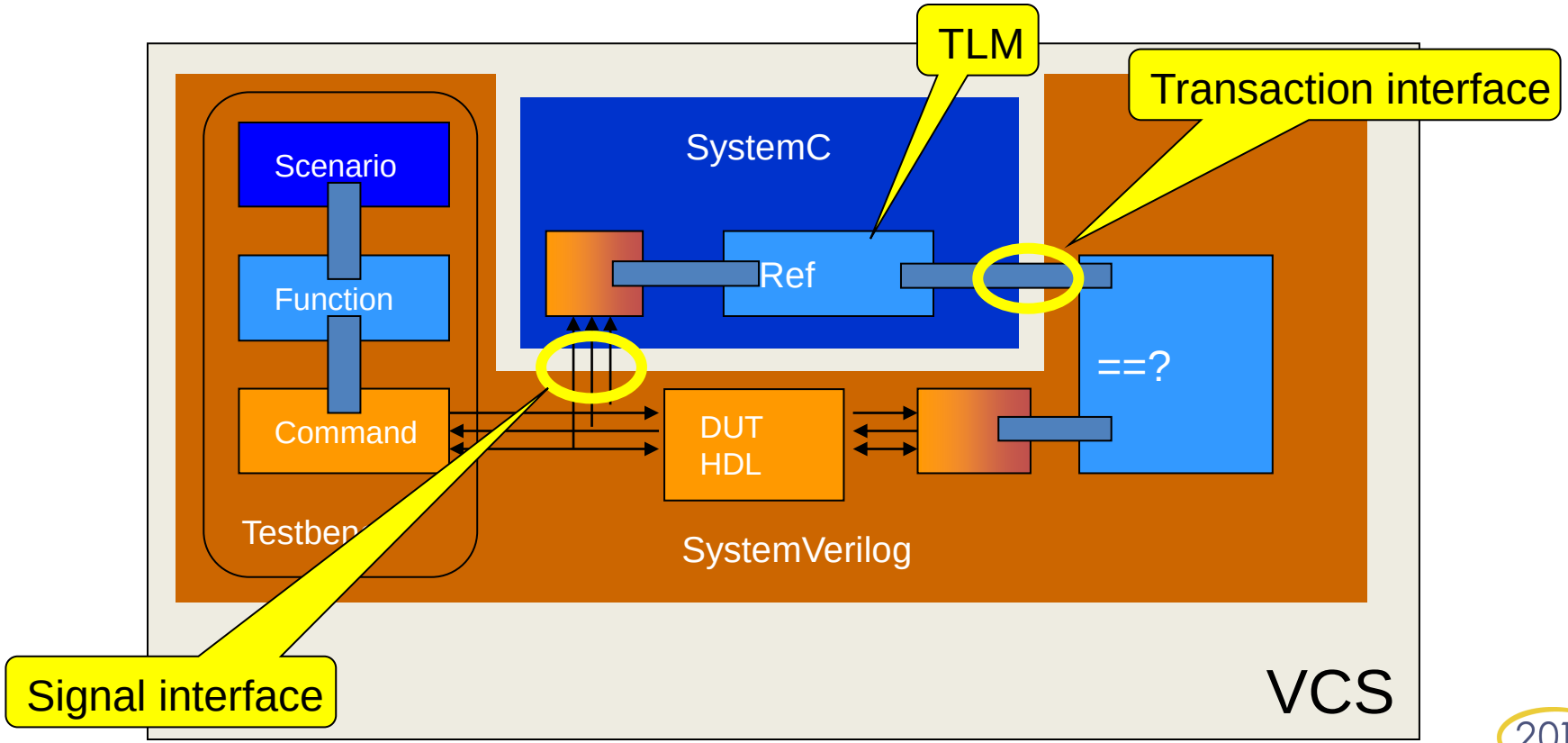


Architecture Exploration, TLM Verification, leveraging System C reference models

LEVERAGING UVM TLM FOR SYSTEM LEVEL VERIFICATION

Mixed language/abstraction level Challenge

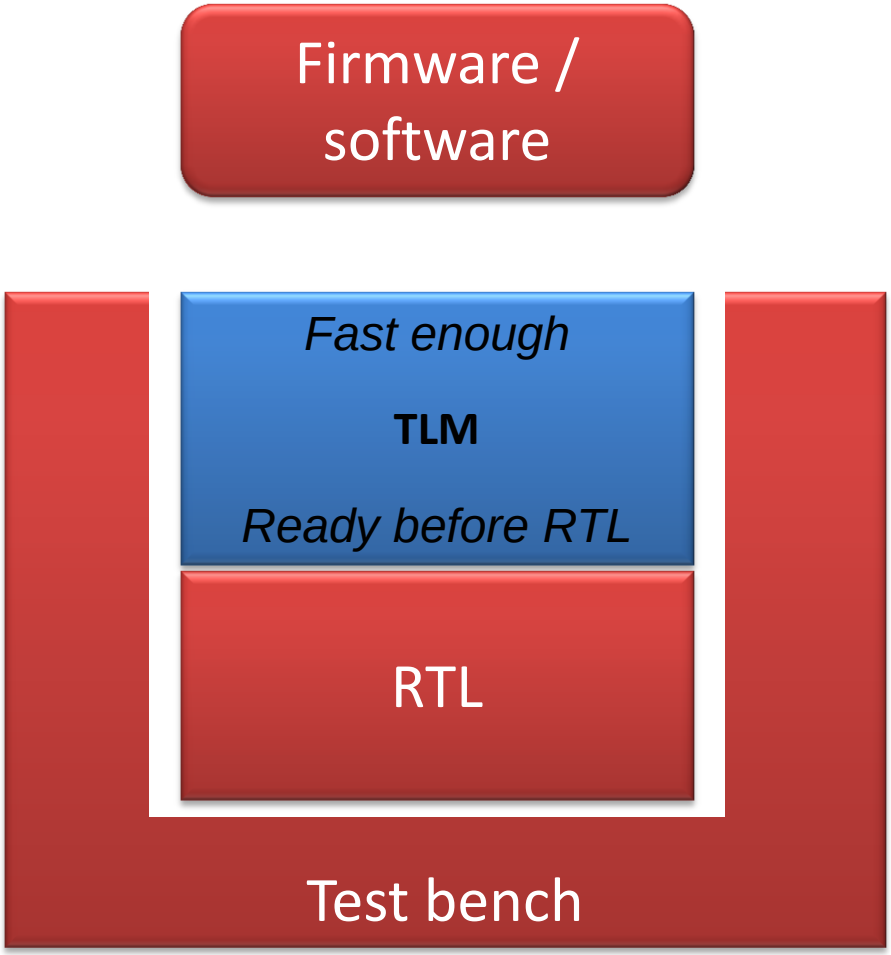
- Single, golden testbench for TL and RTL
- Transaction-level models in SystemC or SystemVerilog
- Verification in a mixed language with different abstraction levels



What is OSCI TLM?

- TLM 2.0
  - Standard for interoperability between memory mapped Bus model
  - Loosely timed and Approximately timed model
  - Used for Virtual platform, Architectural analysis, Golden model for Hardware verification
  - Provided Sockets(Transport, DMI and Debug interfaces)
  - Generic payload and extension mechanism
- TLM 1.0 is included inside TLM 2.0 standard (put, get, nb_put, nb_get, transport).

Reasons for Using TLM



Software development

Accelerates product release schedule
Available before RTL

Architectural modeling

Hardware verification

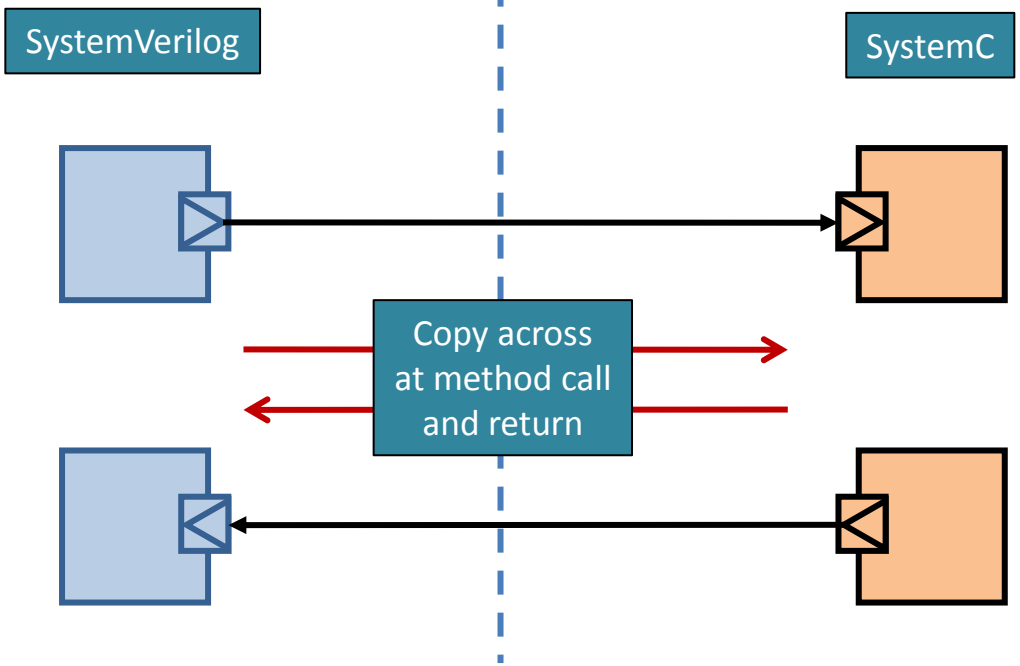
TLM = golden model

Represents key architectural components of hardware platform
Simulates much faster than RTL

Fast!

TLM-2.0 across SV and SC

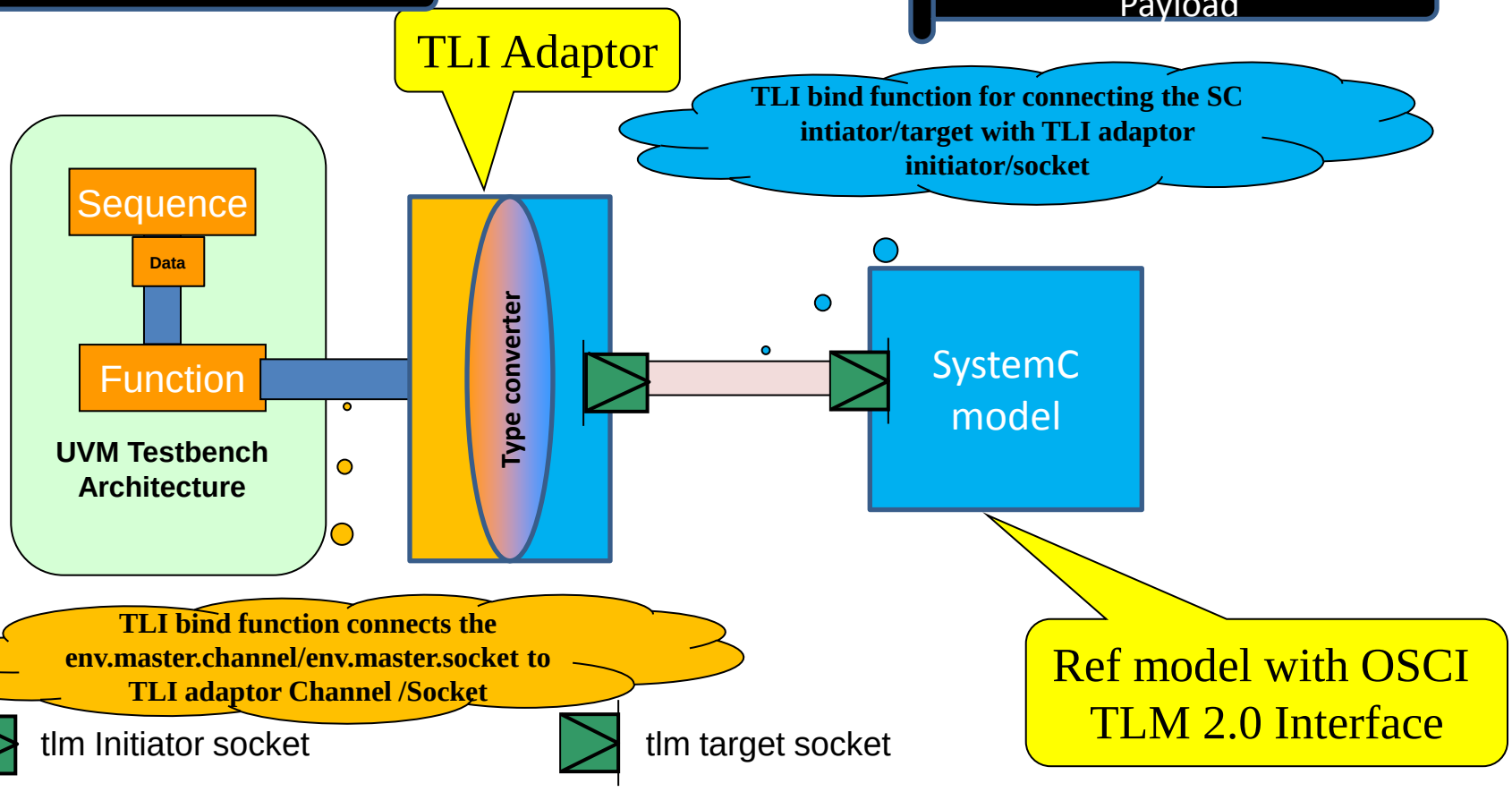
- Tool-specific mechanism
  - Not part of UVM
  - VCS: TLI-4
  - Open Source: UVM-Connect



Methodology Overview : How does it work

Also Work from SC to UVM direction

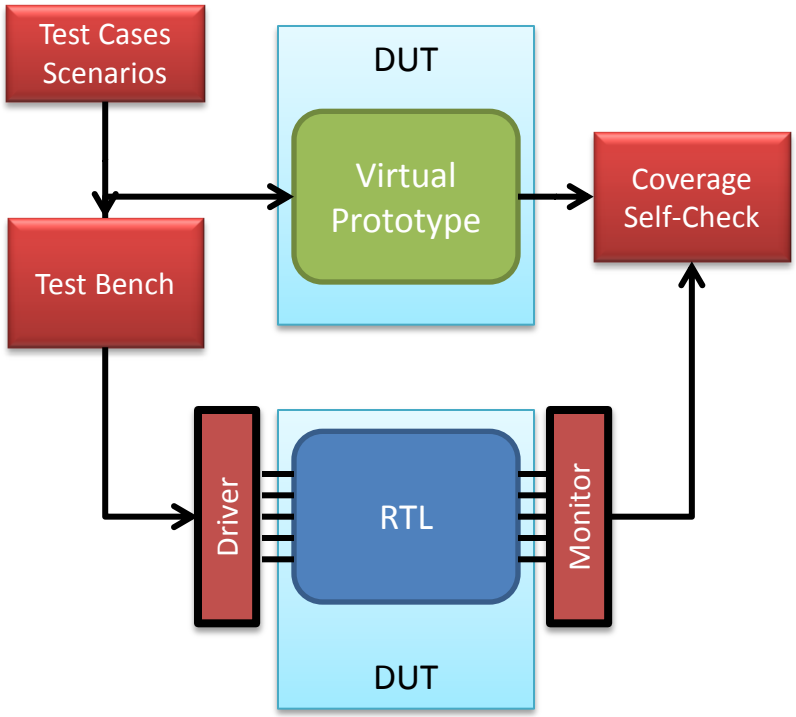
Data Pkt is user-defined/TLM2.0 Generic Payload



SC | SC-SV TLI | PIN IF | SV-SC TLI | SV | Channel

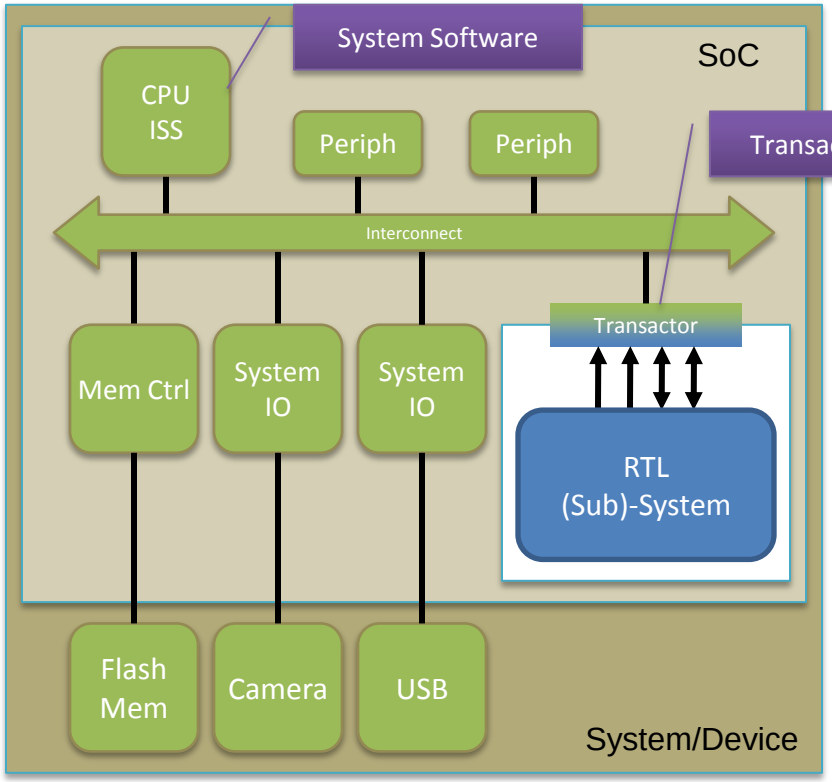
Leveraging TLM2.0 for HW/SW Integration

Early Test Bench Development



1. Develop test bench infrastructure
2. Develop early test cases and scenarios

HW/SW Co-Verification



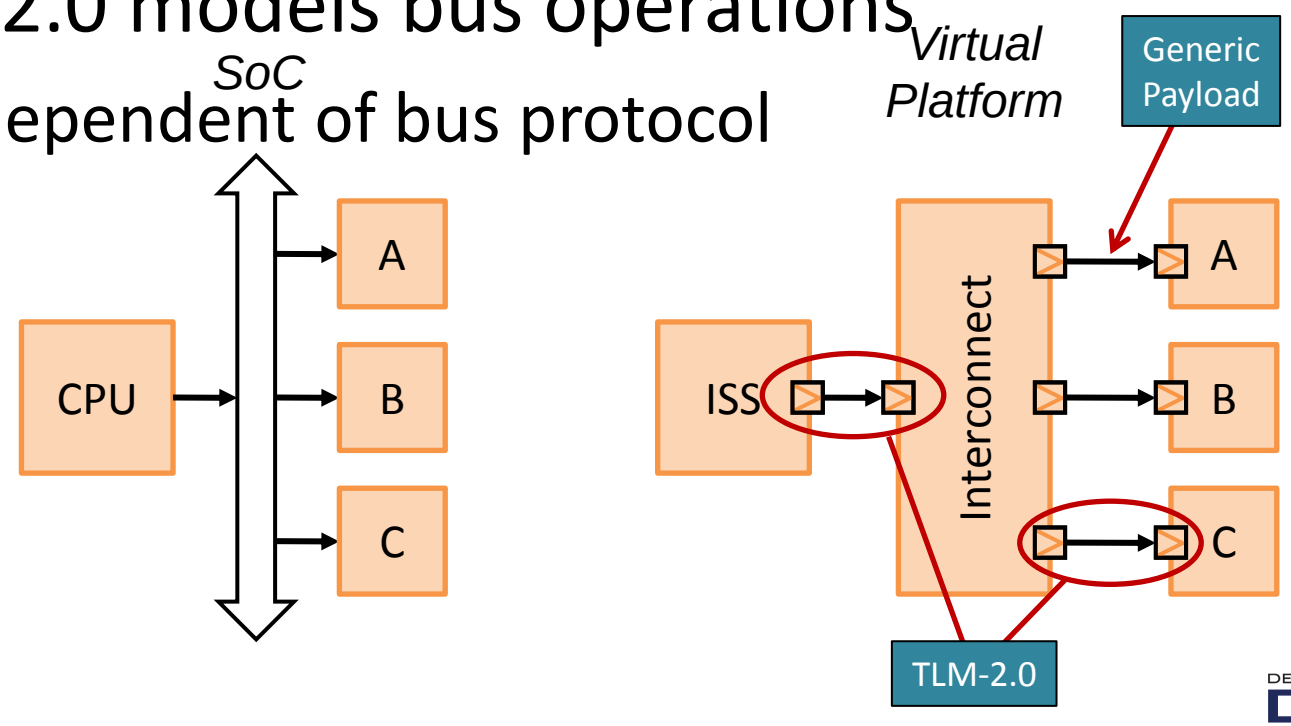
1. SW Driven Verification
2. SoC HW/SW integration

accellera SYSTEMS INITIATIVE™

2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Virtual Platforms and TLM-2.0

- Virtual Platform written in SystemC
- Bus-based SoC
- TLM-2.0 models bus operations
  - Independent of bus protocol



RTL & Fast Models

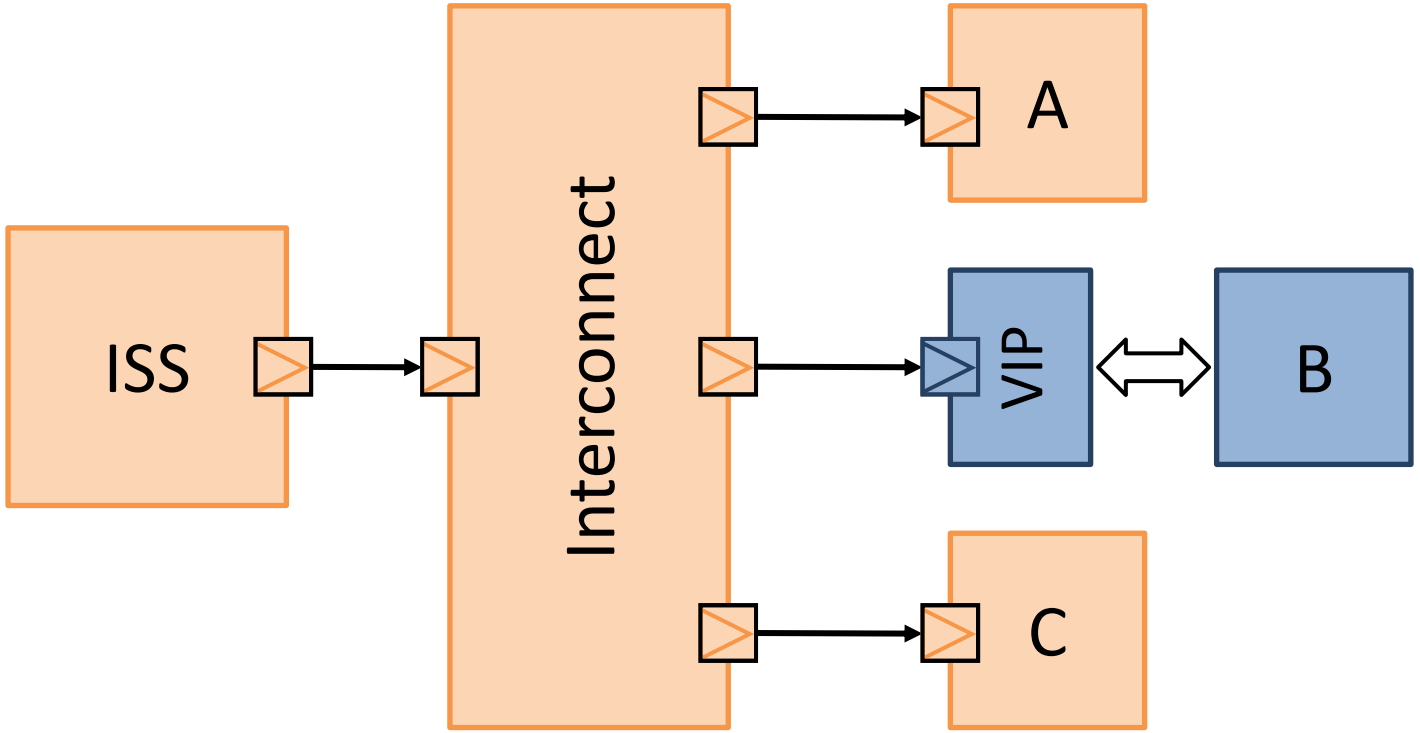
- Speed and accuracy tradeoff
  - RTL – full cycle accuracy
  - TLM – transaction accuracy
  - Some intermediate points also available (consider ROI)

Model	TLM Virtualizer	TLM Co-Sim	RTL Co-Sim	RTL Backdoor	RTL
Speed	Fast	Fast	Slow	Slow/Med	Slow
Accuracy	TLM	TLM	TLM	TLM	Cycle

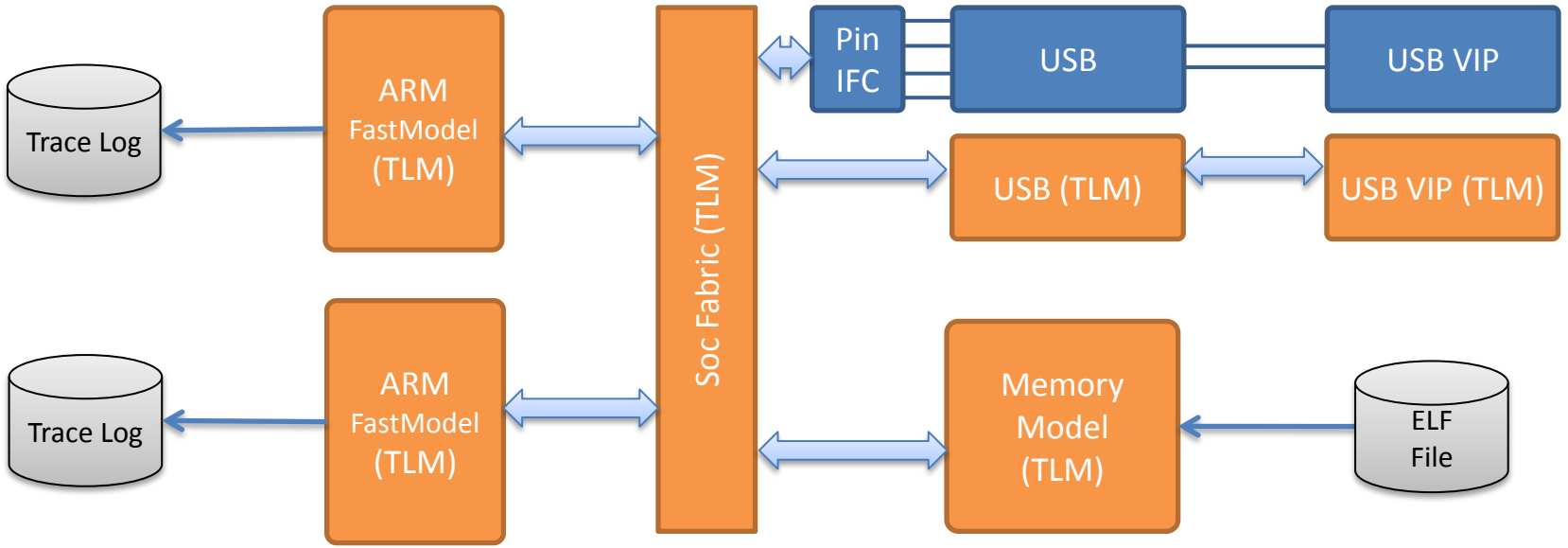
Accuracy (More RTL)

Need to consider ROI for each of these environments

RTL in Virtual Platforms



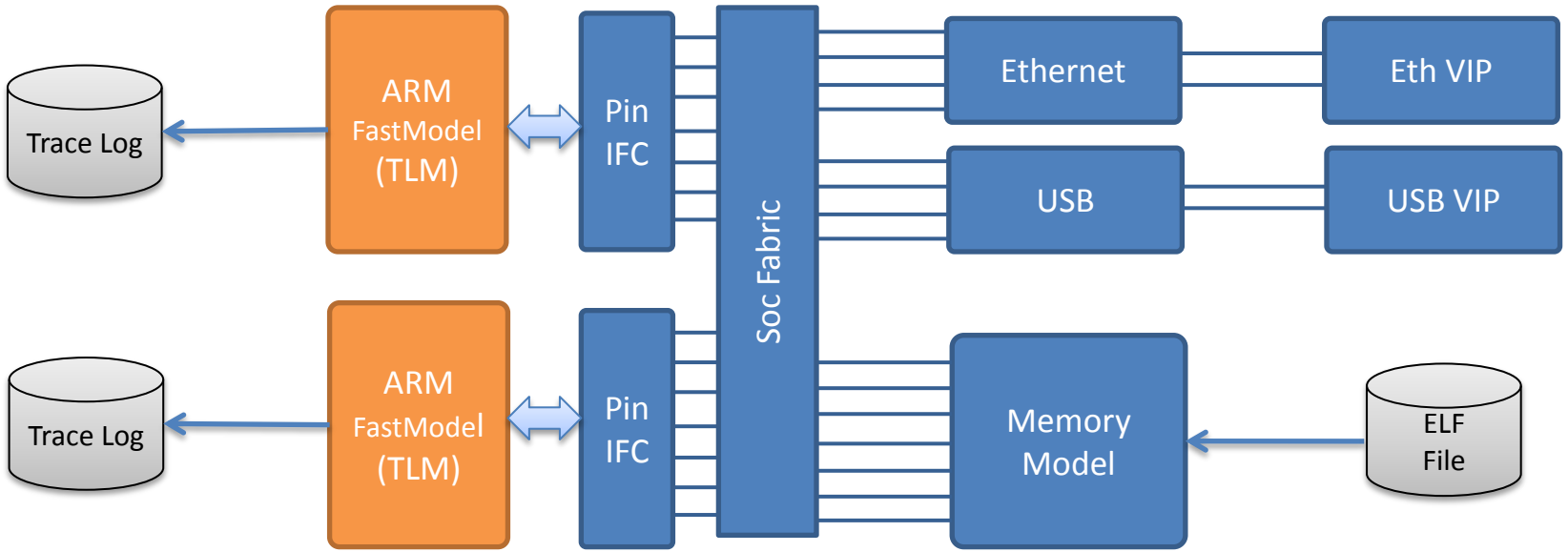
Co-Simulation: Virtualizer TLM & RTL



TLM/RTL Co-Simulation

- Transaction level accuracy
- Fast until RTL is touched
- Minimal RTL to maintain speed
- Minimal design changes, only when RTL is inserted

Co-Simulation: TLM & RTL

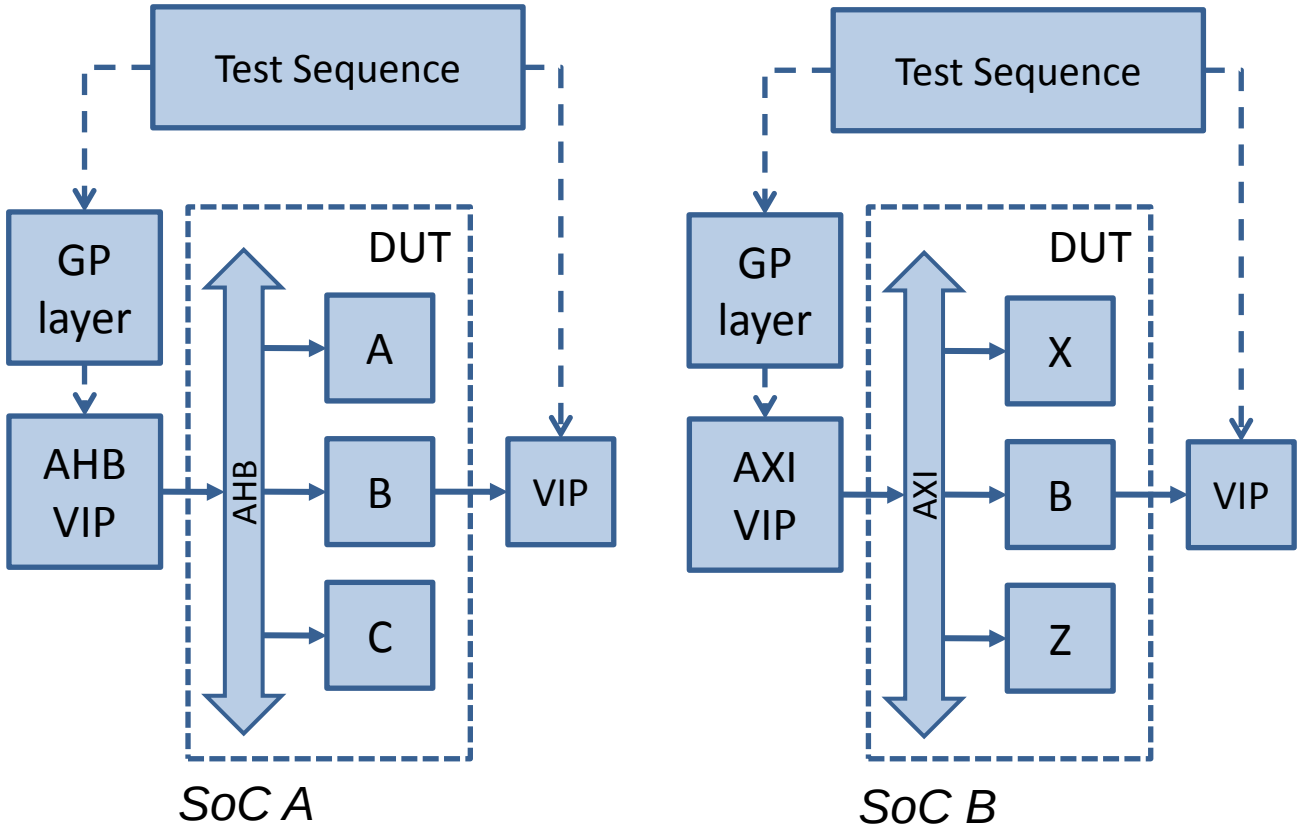


TLM/RTL Co-Simulation

- Transaction level accuracy
- Mostly RTL
- Simulation performance similar magnitude range as RTL simulation speed,
- Design change required to add Pin Interfaces/Adapters

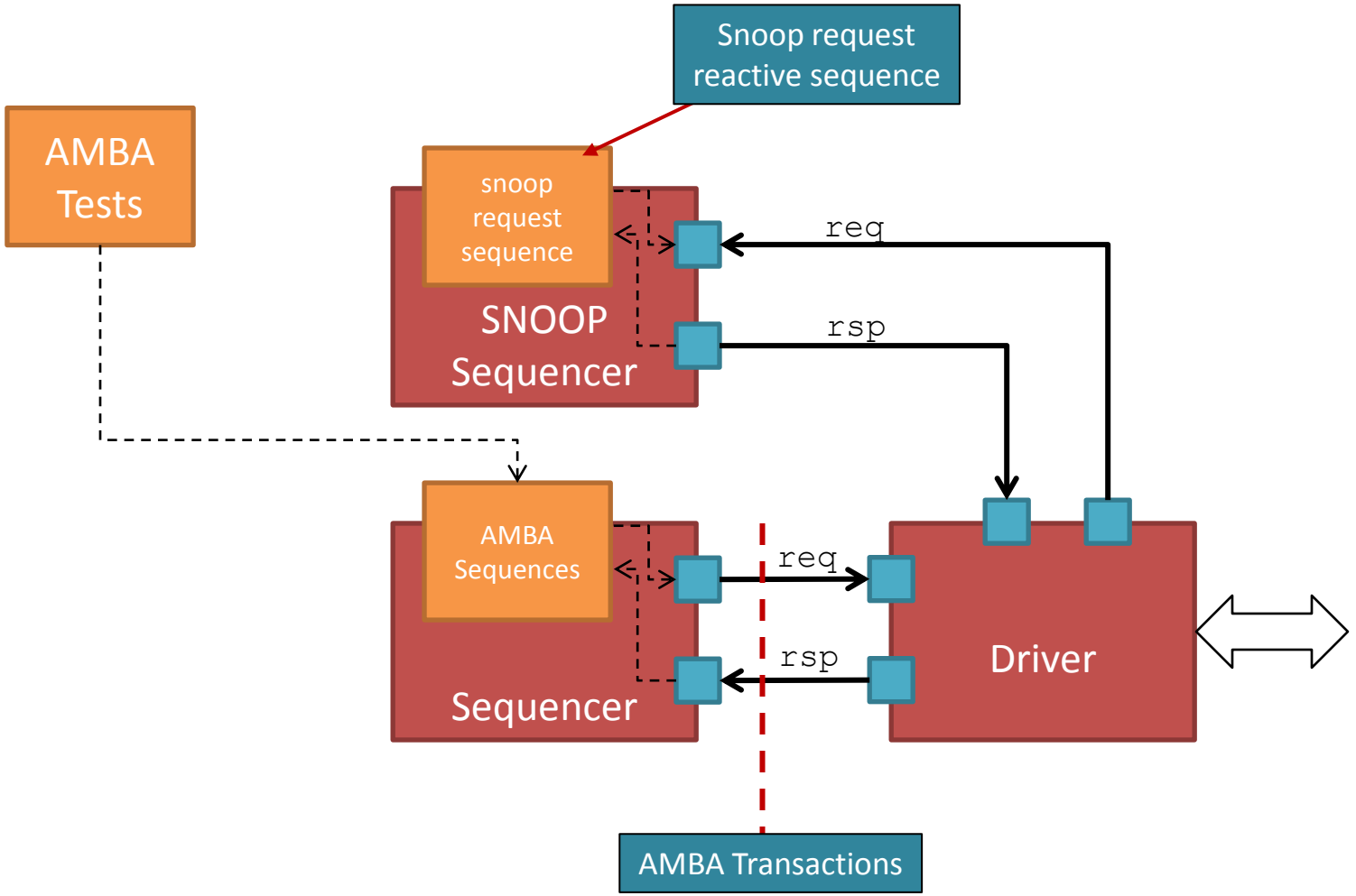
Protocol-Agnostic Tests

- Test written using Generic Payload
- Can work on any bus

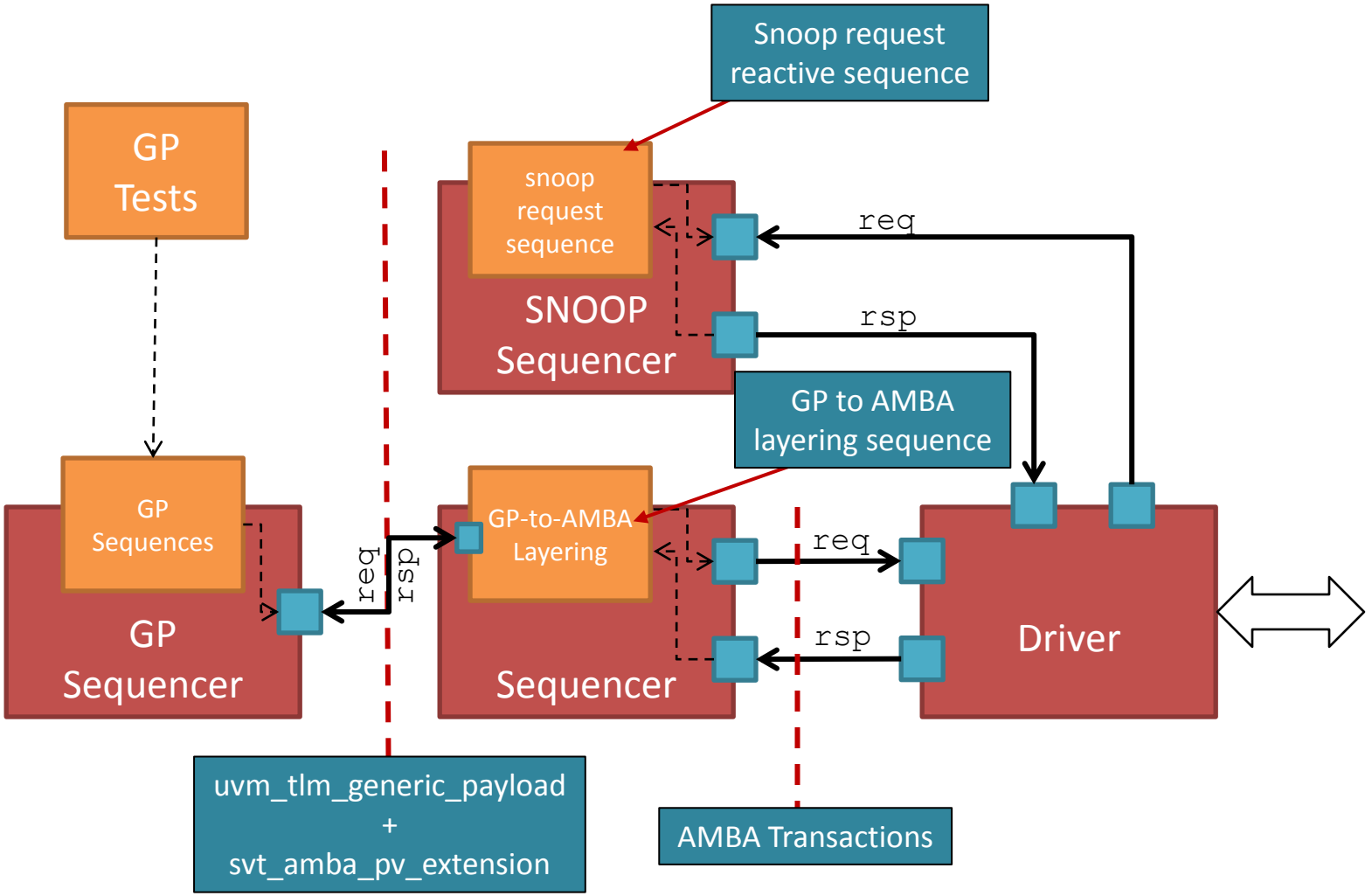


VIP ARCHITECTURE FOR TLM GP

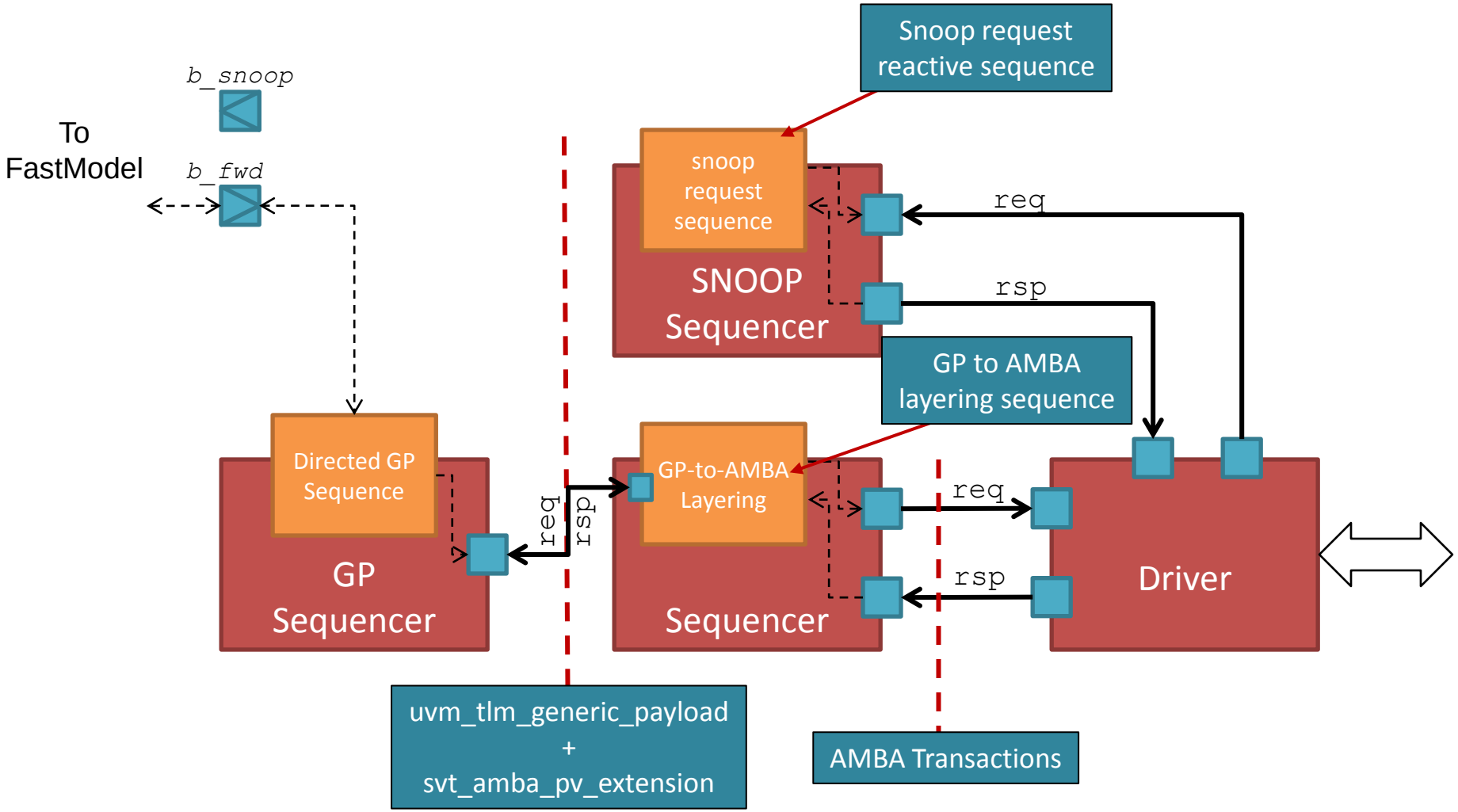
AMBA Sequences in SV



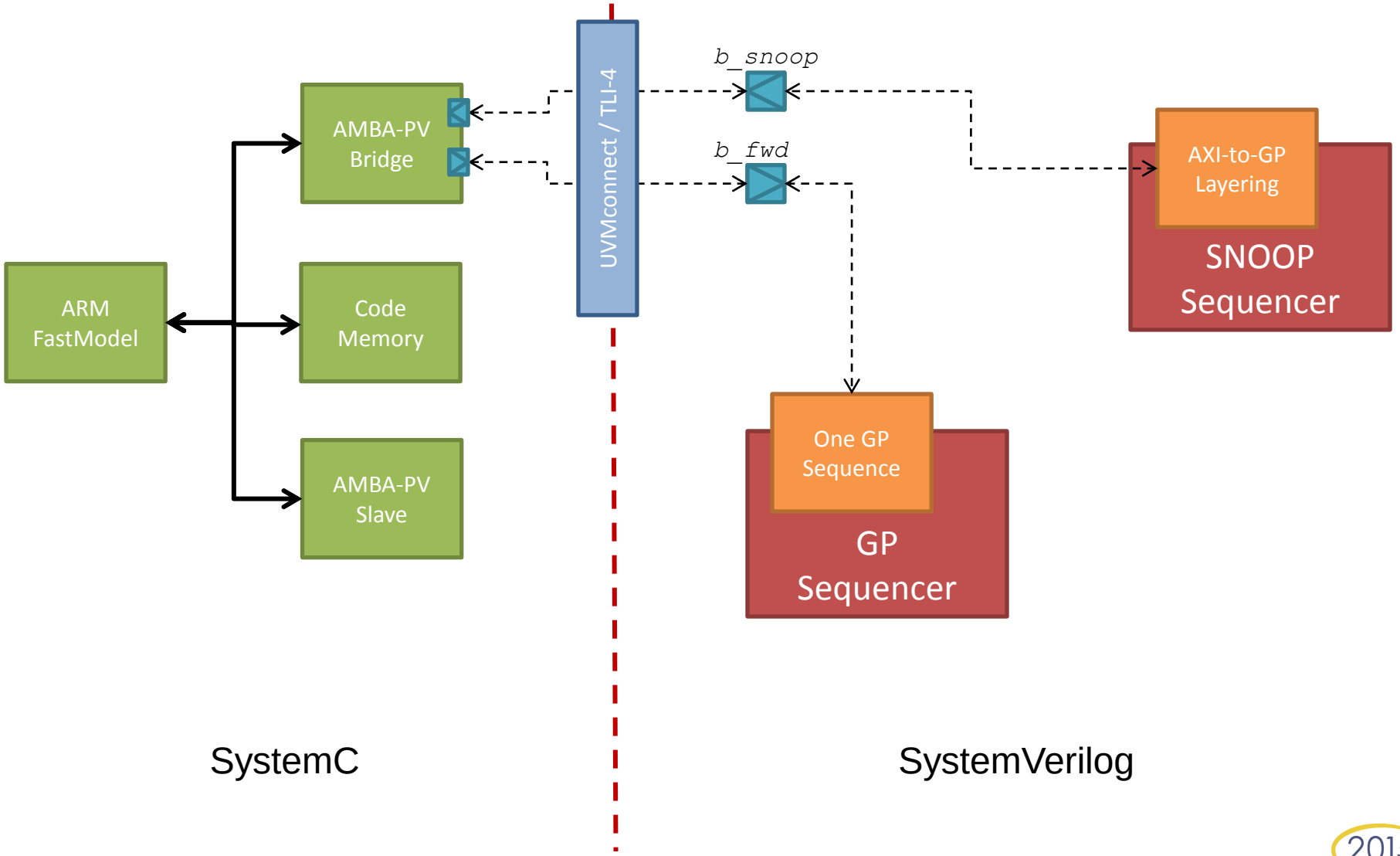
GP-Based Sequences in SV



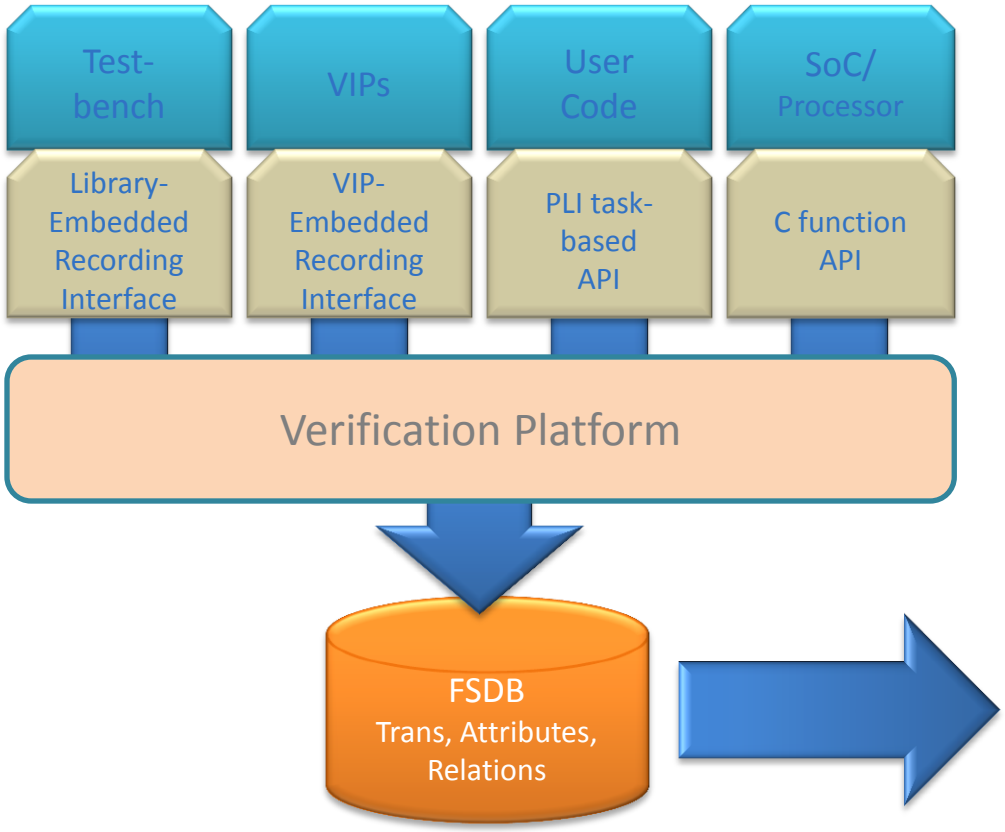
SystemC-Based Tests



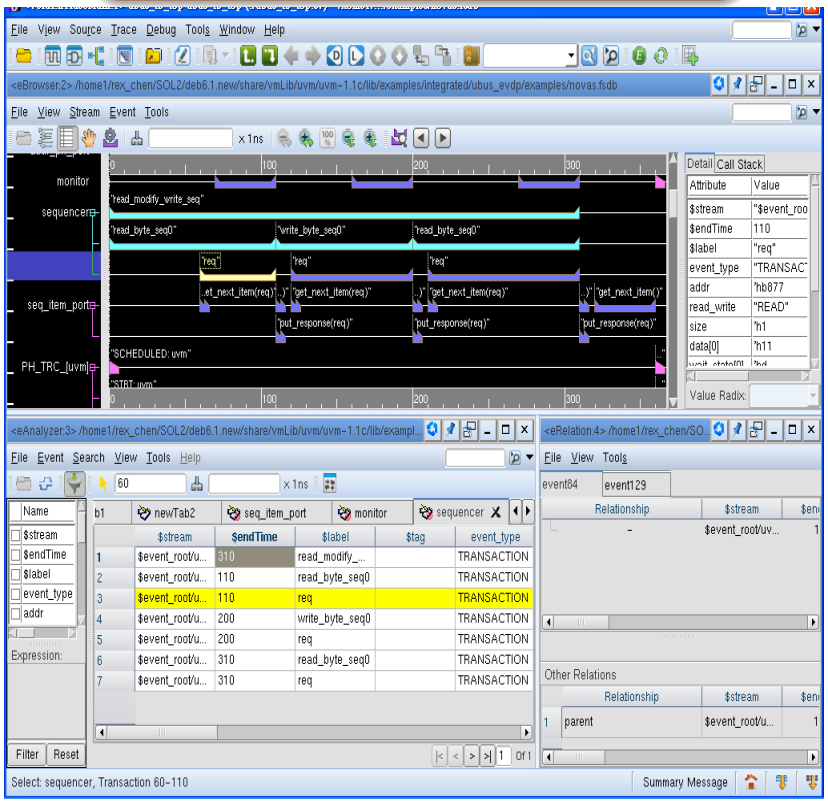
FastModel Connection



UVM Transaction Debug

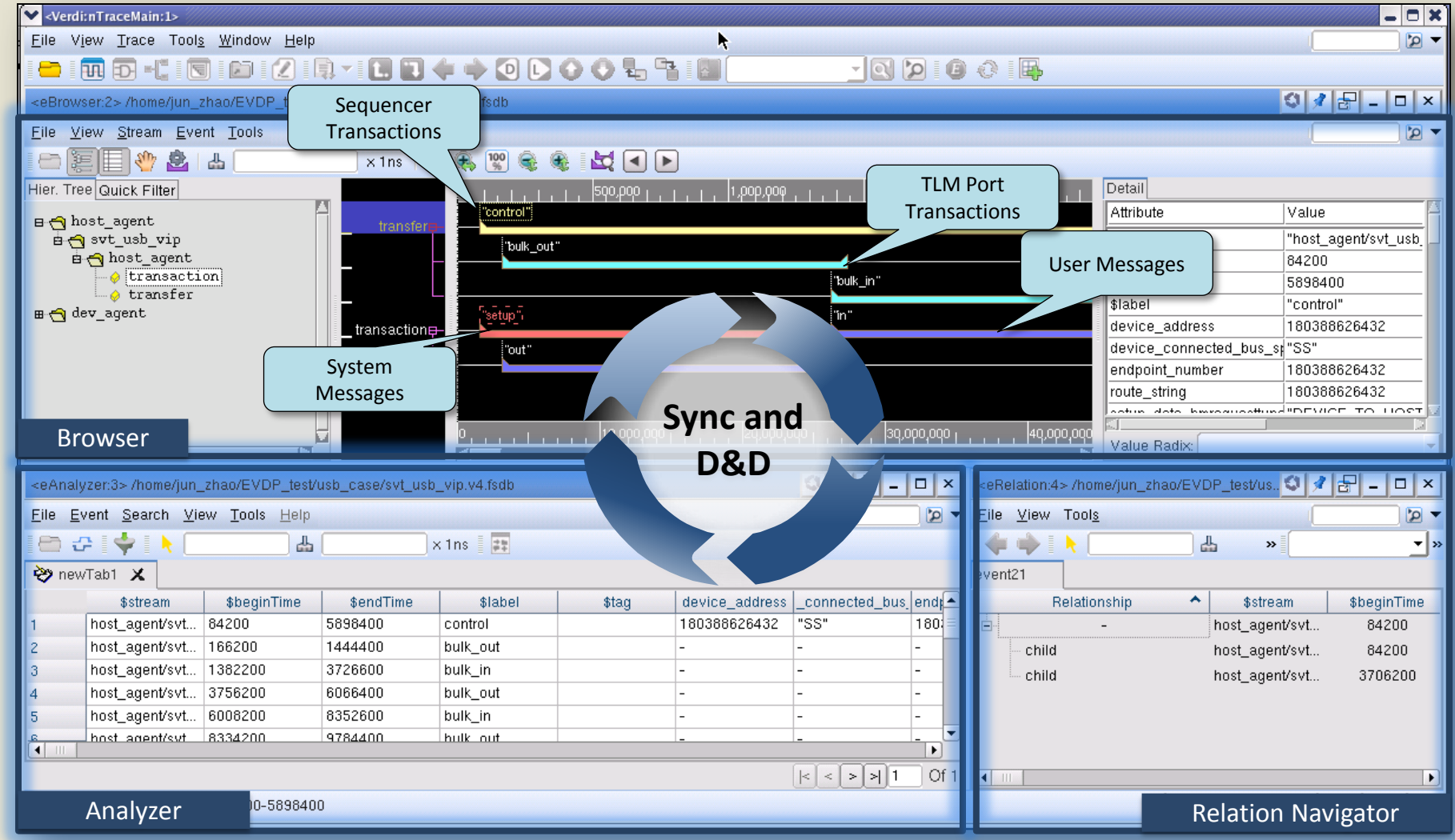


New Transaction Based Debug Apps



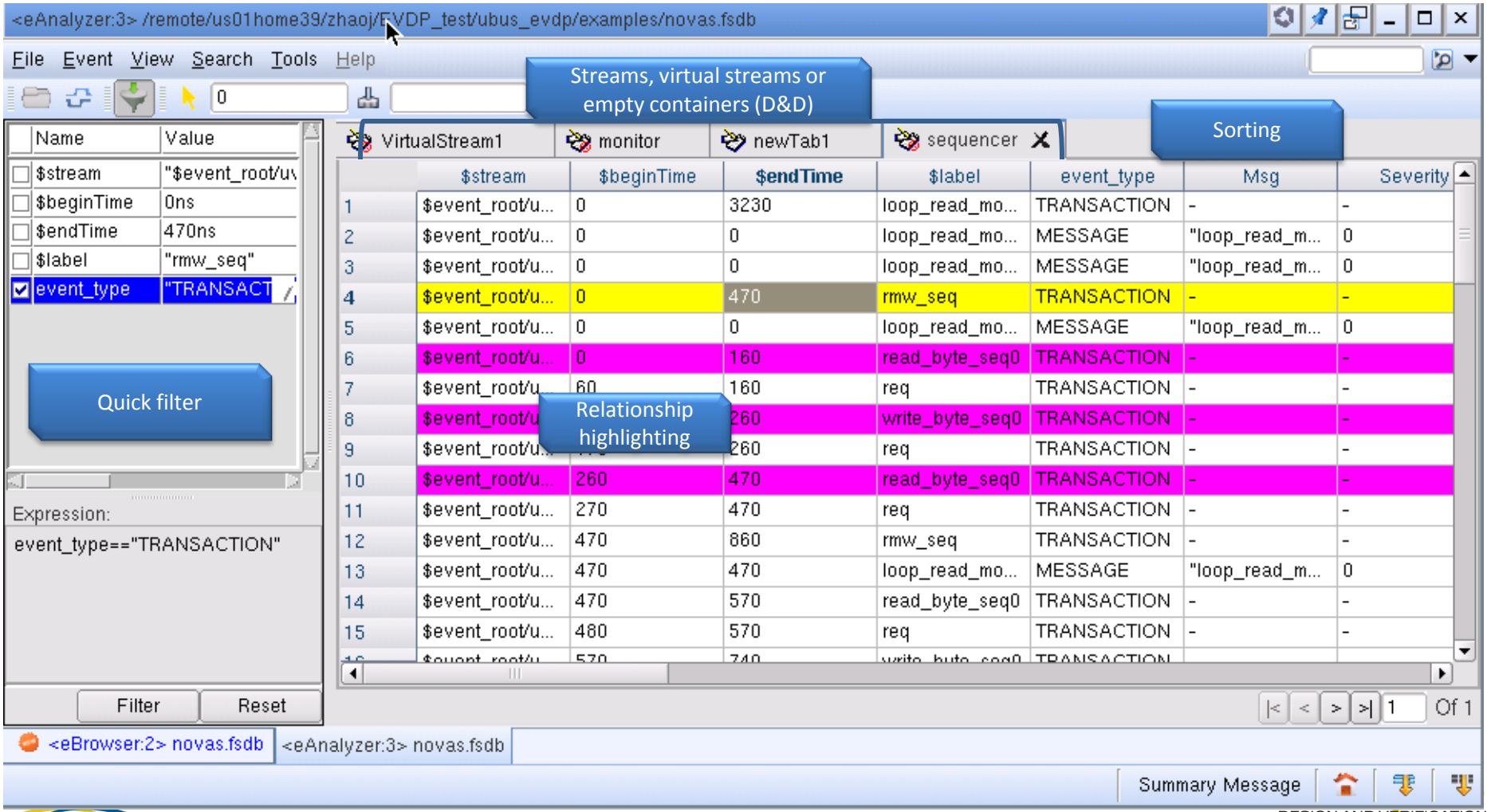
Transaction Debug Apps

Enhancing Post Process Testbench Debug



Transaction Analyzer

Analysis and data mining tool for abstract data

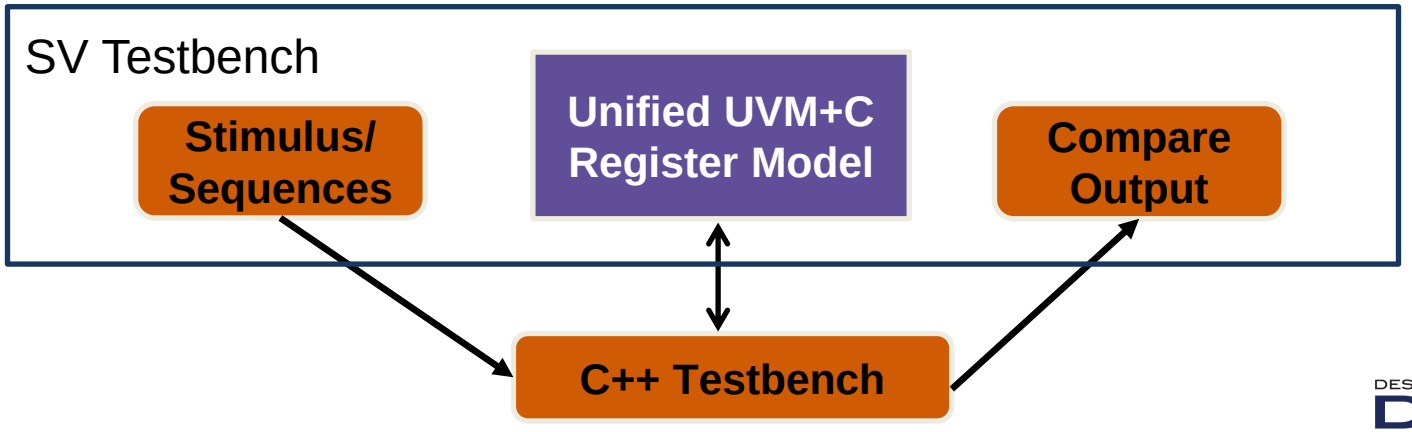


Unified register Modeling between C an SV

Unified Register Model for C and SV

Challenges and Motivation

- To enhance register model interaction between C and UVM , a new mechanism is needed such that an identical and consistent register model can be accessed from either side
- This framework can alleviate the need to maintain and verify two separate and disjointed models
- Updates to this common register model from one side need to be immediately visible to the other and vice versa



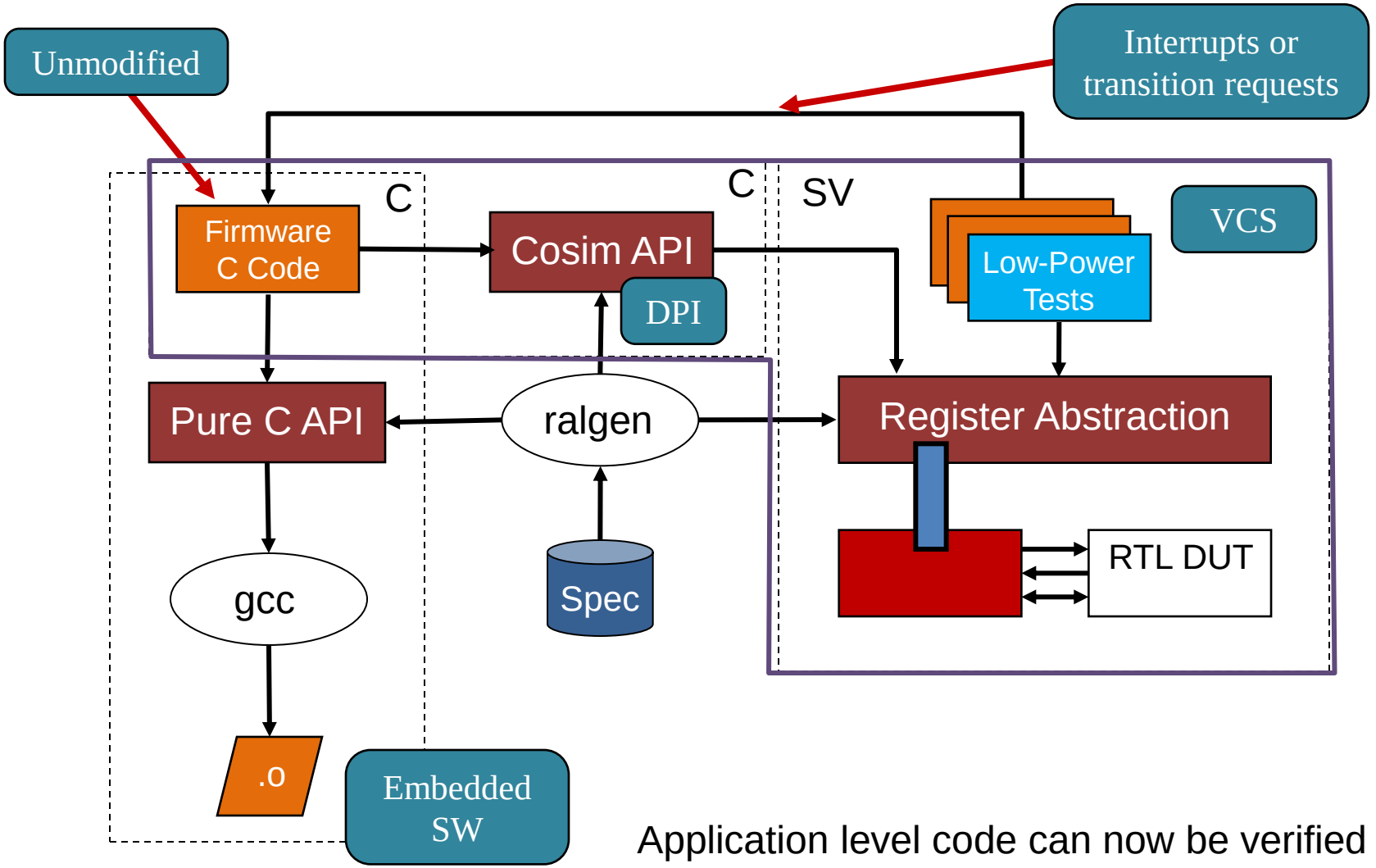
Implementing a Unified Model

- The standard UVM RAL(Register Abstraction Layer) models the registers and memories of the Design Under Test(DUT) in System Verilog(SV)
- A C-model of the design exists and has a C++ based environment. The APIs provided allow access of the register fields and memories in the SV-RAL model from the application level C code
- The RAL-SV model can be updated from the C side or the SV testbench. Any updates in the RAL-SV model from the SV side are reflected in the C side

Introduction to RAL-C++ interface

- Allows firmware and application-level code to be developed and debugged in VCS
- Preserve abstraction offered by the RAL
  - Hide physical address, field positions
- Provides C++ API to access RAL components
  - Fields, Registers
- Two versions of the RAL C++ API can be generated
  - Interface to RAL model using DPI
  - Stand-alone C++ code targeted to S/W

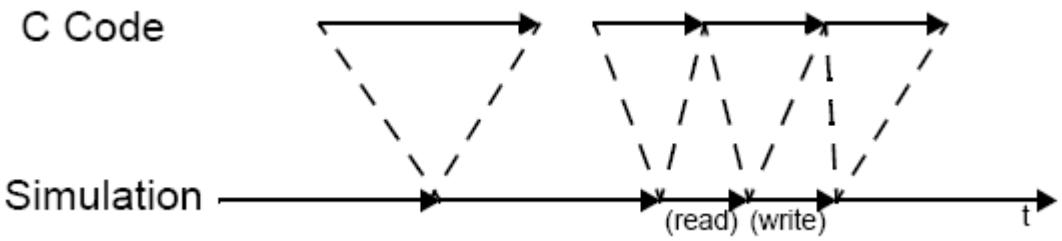
RAL C API



Application level code can now be verified against a simulation and then used, unmodified, in the final application

Execution timeline

- Execution non concurrent unlike code
- Rest of Simulation frozen when ‘C’ code is running
- Entire execution in ‘C’ is in ‘0’ time in the simulation timeline
- ‘Polling’ strategy can impact overall performance
- Interrupt driven service strategy more ideal



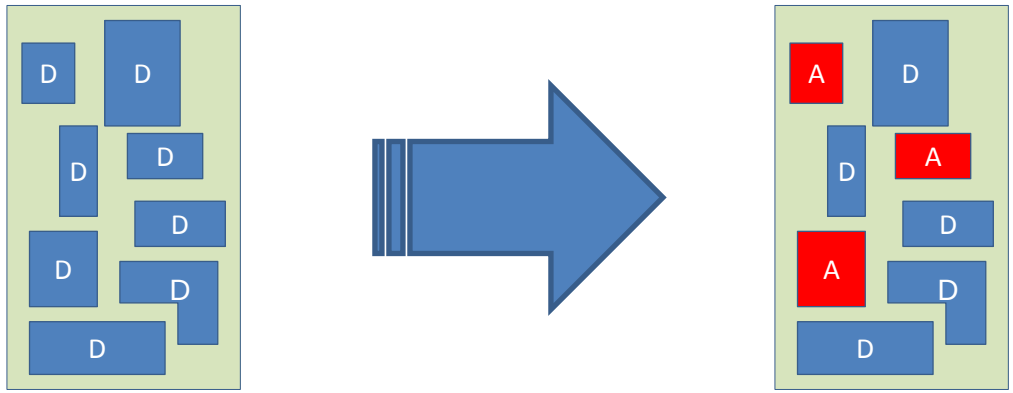
Using UVM for mixed-AMS SOC verification

accellera SYSTEMS INITIATIVE™

2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Need for a UVM-AMS Testbench?

- No clear methodology for mixed-AMS SoC verification
  - Need interaction between all IPs, including AMS
  - Need signal access between analog IPs and others (xmrs/oomrs)
- Holes in mixed-AMS Block-level verification
  - Synchronous verification is not enough, Ref models, Flow automation needed for characterization, regression
- The solution for self-checking verification environment



UVM

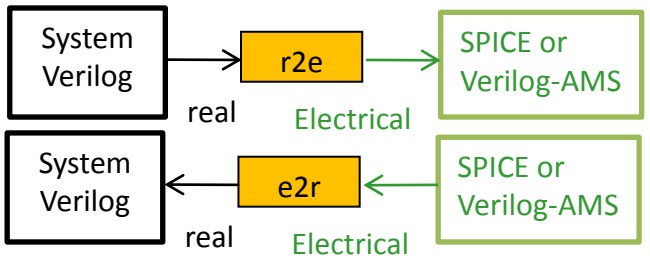
UVM AMS TB

UVM-AMS Testbench Overview

Technology for mixed-signal SoC functional verification

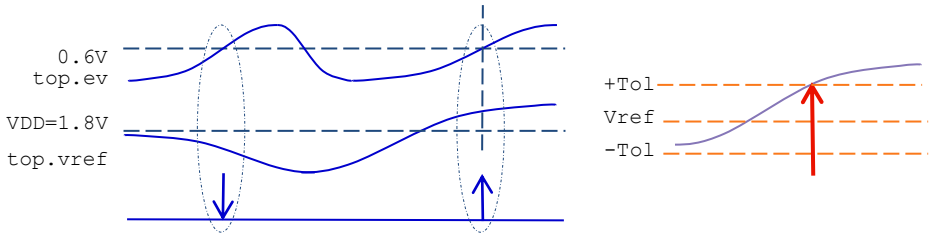
Basic Usage

- Electrical ⇔ Real conversion
- Asynchronous analog events
- AMS toggle coverage



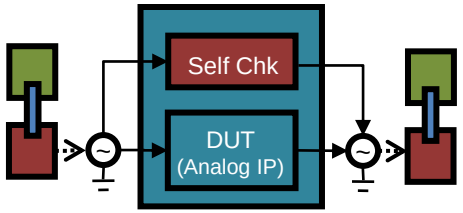
Intermediate usage

- AMS SystemVerilog assertions
- AMS SystemVerilog testbench
- AMS Checker Library
- SystemVerilog Real Number Modeling

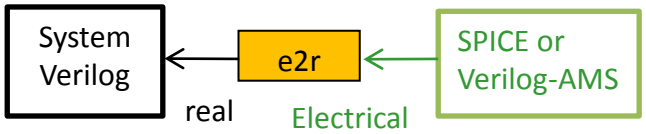


Advanced usage

- UVM AMS testbench
- AMS Source generators



Real ⇔Analog Conversion



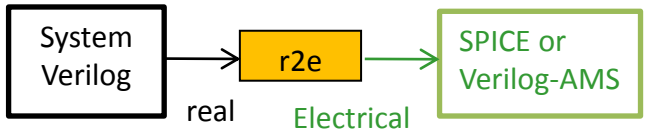
Easy XMR read access to internal analog signal voltage and current

```
$snps_get_volt(anode)
$snps_get_port_current(anode)
```

anode: full hierarchical analog node name

Example:

```
real r;
always @(posedge clk)
  r <=$snps_get_volt(top.i1.ctl);
```



Easy XMR write access to internal analog signal voltage.

```
$snps_force_volt(anode,val|real)
$snps_release_volt(anode)
```

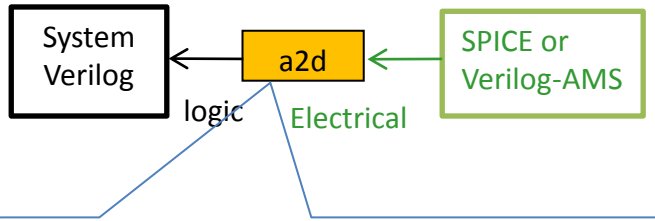
anode: full hierarchical analog node name

val|real: absolute value or real variable

Example:

```
real r;
initial
 $snps_force_volt(top.i1.ctl,0.0);

always @(posedge clk) begin
  r <= r+0.1;
  $snps_force_volt(top.i1.ctl,r);
```

Logic⇔Analog Conversion

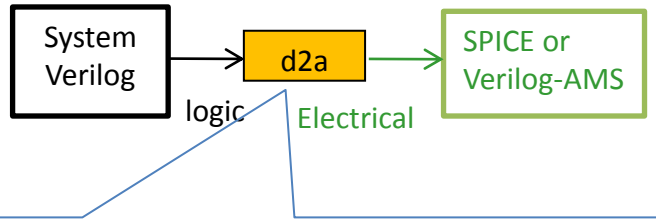


Automatic insertion of a2d connect models between SystemVerilog and SPICE.

User can re-define the threshold

Example:

```
assign verilog_wire =
   top.i1.i2.x1.clk;
initial begin
  verilog_reg =
              top.i1.i2.x1.strb;
...
```



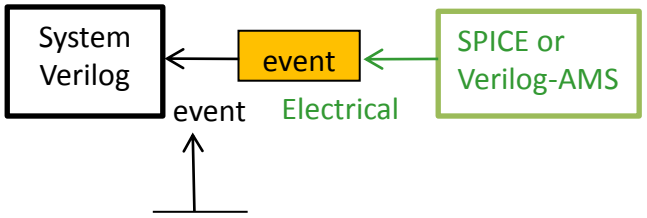
Automatic insertion of d2a connect models between SystemVerilog and SPICE.

User can re-define the threshold

Example:

```
reg rst_reg;
assign top.i1.i2.x1.rst = rst_reg;
initial begin
...
rst_reg = 1'b0;
#5 rst_reg = 1'b1;
...
end
```

Asynchronous Analog Events

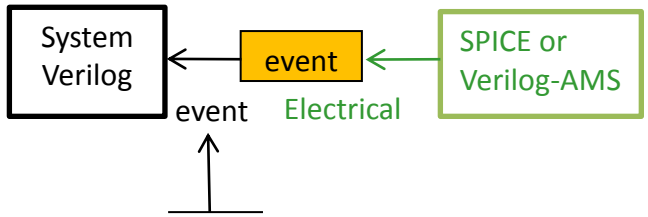



```
$snps_cross(aexpr[,dir[,time
   _tol [,expr_tol]]]);
```

aexpr: analog expression based on system function

Example:

```
always
   @(snps_cross($snps_get_volt(
   top.i1.ctl)-0.6,1))
begin
  $display("Signal ctl is raising
   above 0.6V");
```




```
$snps_above(aexpr[,time_tol
   [,expr_tol]]]);
```

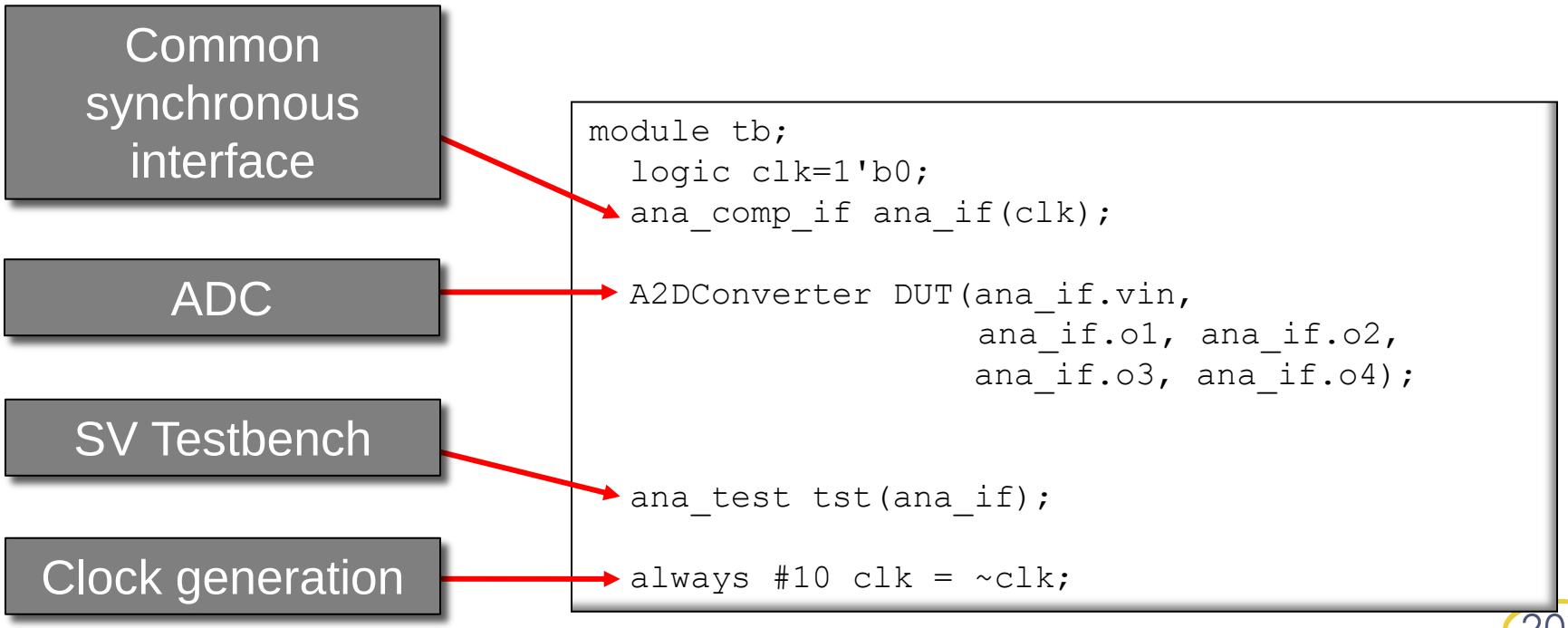
aexpr: analog expression based on system function

Example:

```
always
   @(snps_above($snps_get_volt(
   top.i1.ctl)-0.6))
begin
  $display("Signal ctl is above
   0.6V");
```

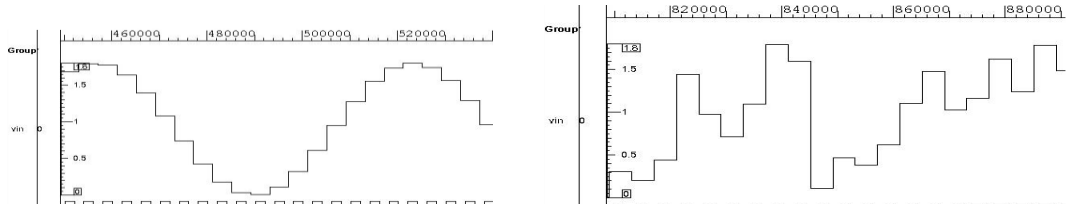
Instantiation of Analog DUT & Testbench

- DUT and testbench are instantiated and connected using SV interface
- VCS automatically inserts necessary e2r and r2e models



AMS Testbench Generators

UVM



Sine Voltage Gen
- Vmax=1.0V,
- Vmin=-1.0V
- F=1.0MHz

Construct sin Wave generator.
Default is auto-run throughout run_phase()

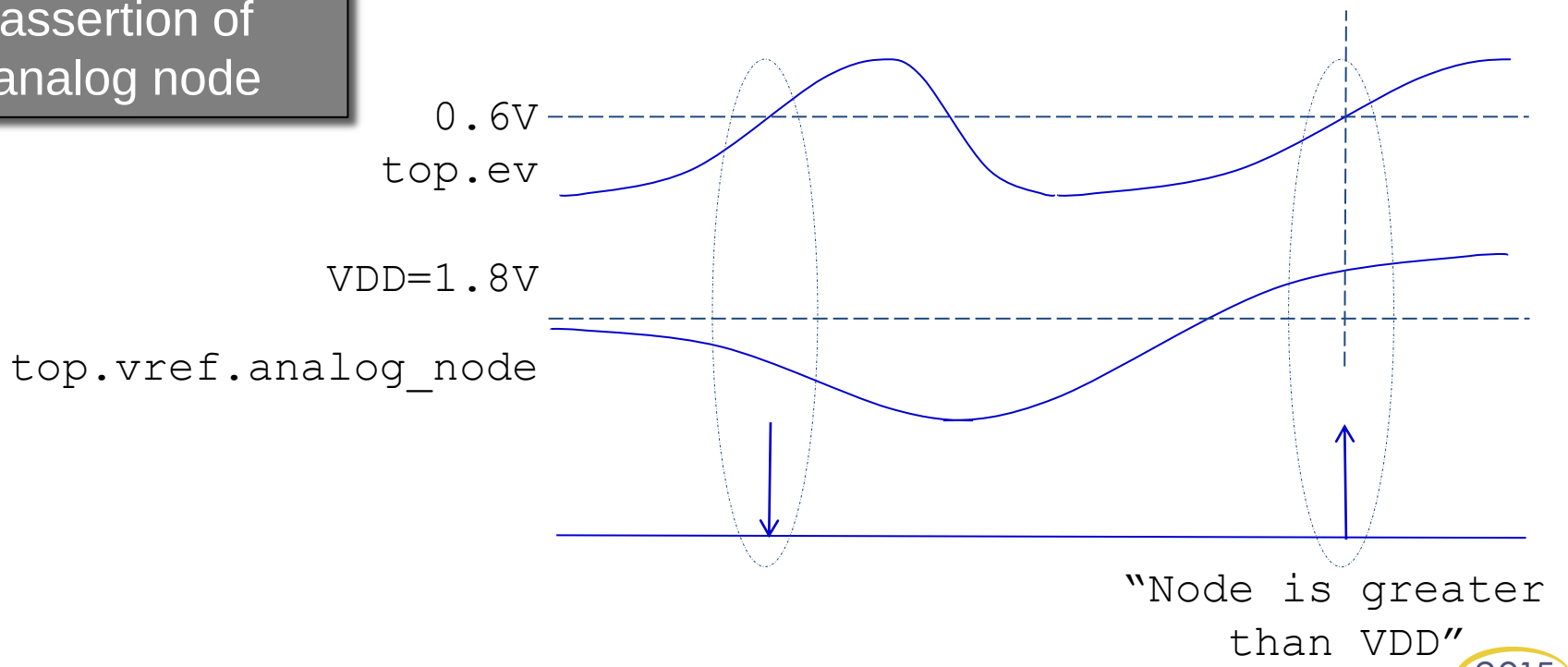
```
...

class my_env extends uvm_component;
   ...
  sv_ams_sine_voltage_gen#(-1.0, +1.0, 1.0E6) sGen_IN;

 function void build_phase(uvm_phase phase);
   super.build_phase(phase);
   uvm_resource_db#(virtual ams_src_if)::
                set("*", "uvm_ams_src_if", aif, this);
   sGen_IN  = sv_ams_sine_voltage_gen#
   (-1.0, +1.0, 1.0E6)::type_id::create("sine", this);

 endfunction
```

Immediate Assertions

Asynchronous

Asynchronous immediate assertion of analog node

```
always @(snps_cross($snps_get_volt(top.ev)-0.6,1))
   assert(top.vref.analog_node <= 1.8)
   else $error("Node is greater than VDD");
```



AMS Testbench Checkers

SV interface containing the Clock generator

Clock generator instance

Checks:
- Vmax=2.25V,
- Vmin=-2.25V
- F=2.5MHz
- Tolerance: +-1.0%

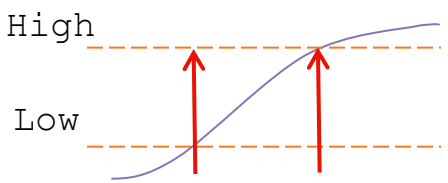
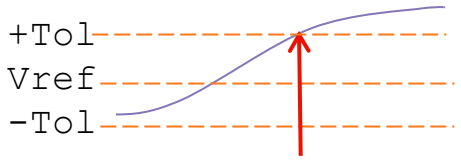
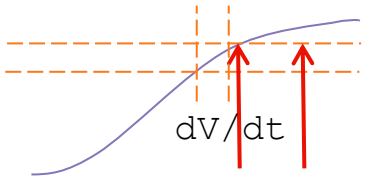
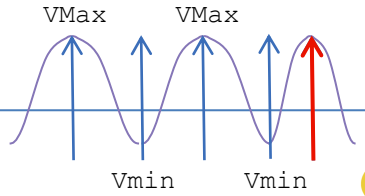
Start/stop frequency checking

Change vmin, vmax

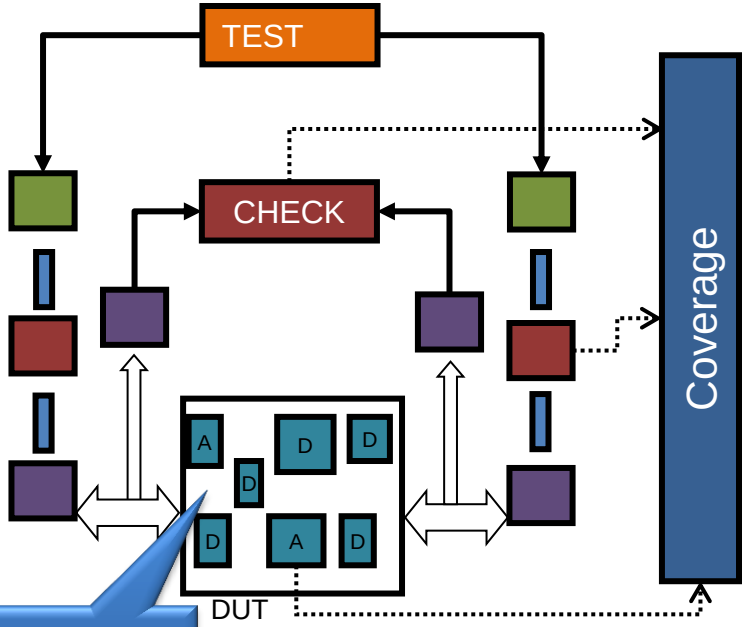
Restart frequency checking

```
module test;
  ana_vref_if ana_if();
  spice_clk_ref ckref(ana_if.clk_out);

  sv_ams_frequency_checker#(-2.25,+2.25, 2.5E6)
       freq_meas = new("Freq Meas", "0", 0,
                       ana_if.clk_out);
initial begin
   @(posedge rst); // Wait end of reset
   saw_freq_meas.start_xactor();
   #2000 saw_freq_meas.stop_xactor();
   saw_freq_meas.set_params(-1.8, +1.8, 2.5E6);
  saw_freq_meas.start_xactor();
endmodule
```

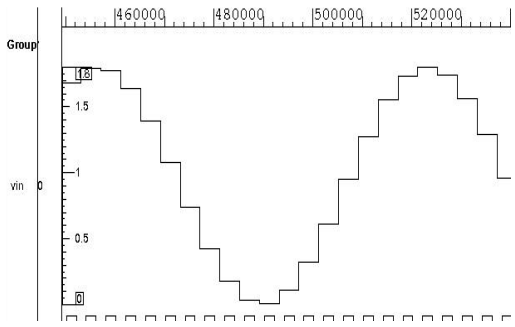
AMS Testbench Checkers

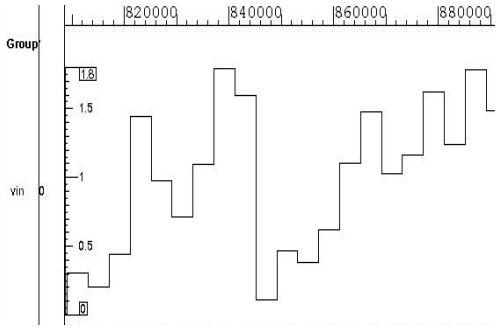
Checkers		
sv_ams_threshold_checker	Checks that analog signal remains within a given high and low threshold. Can perform this check synchronously or asynchronously	
sv_ams_stability_checker	Checks that analog signal remains below or above a given threshold. Can perform this check synchronously or asynchronously	
sv_ams_slew_checker	Checks that analog signal rises/falls with a given slew rate(+/- tolerance). Can perform this check synchronously or asynchronously	
sv_ams_frequency_checker	Checks that analog signal frequency is within a given tolerance	

UVM AMS Verification of A+D SoC



AMS TB contain analog blocks



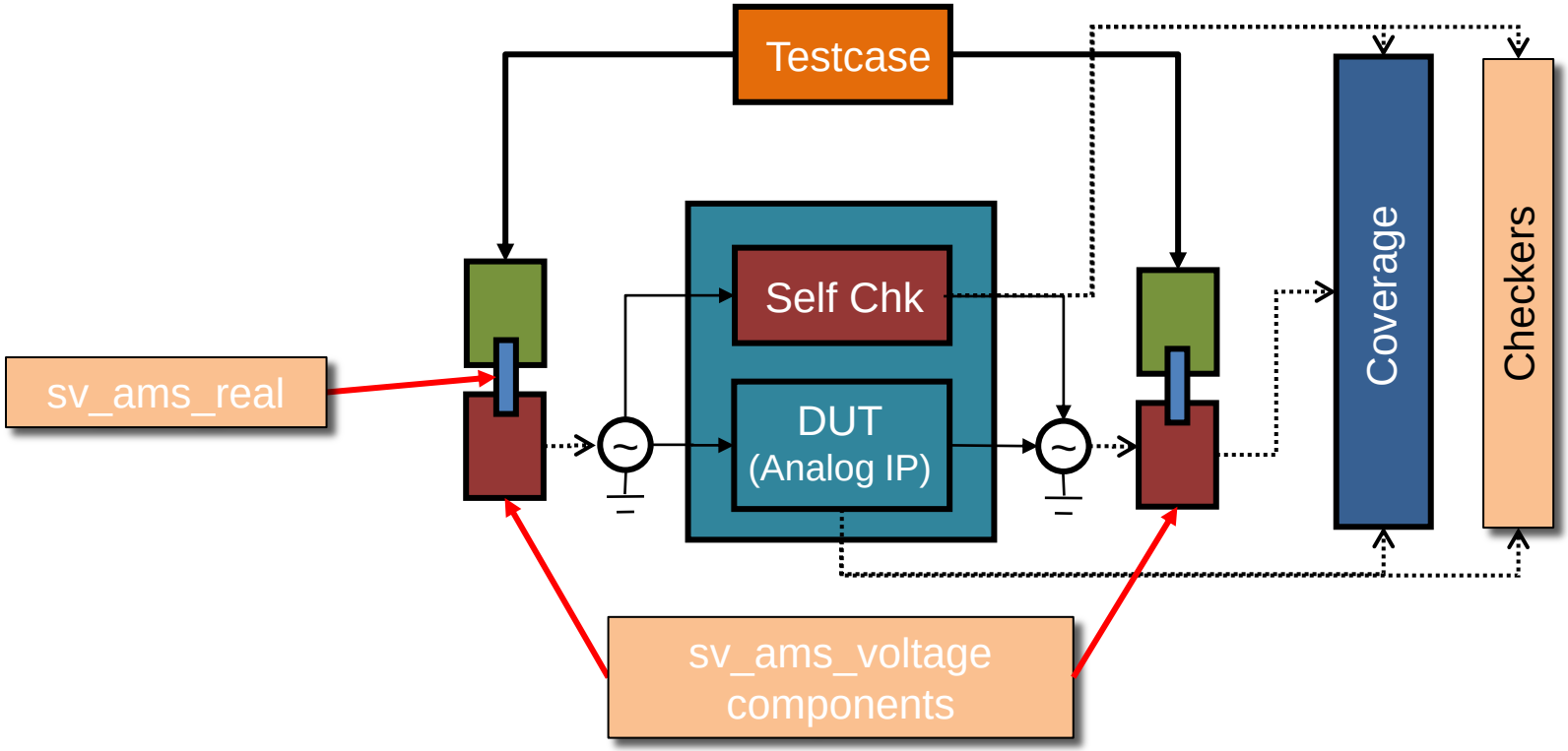



```
...

class my_env extends uvm_component;
   ...
  sv_ams_sine_voltage_gen#(-1.0, +1.0, 1.0E6) sGen_IN;

 function void build_phase(uvm_phase phase);
   super.build_phase(phase);
   uvm_resource_db#(virtual ams_src_if)::
                set("*", "uvm_ams_src_if", aif, this);
   sGen_IN  = sv_ams_sine_voltage_gen#
   (-1.0, +1.0, 1.0E6)::type_id::create("sine", this);

 endfunction
```

Sine Voltage Gen
- Vmax=1.0V,
- Vmin=-1.0V
- F=1.0MHz

Construct sin Wave generator. Default is auto-run throughout run_phase()

Other use:
Verifying Analog IP before SoC integration



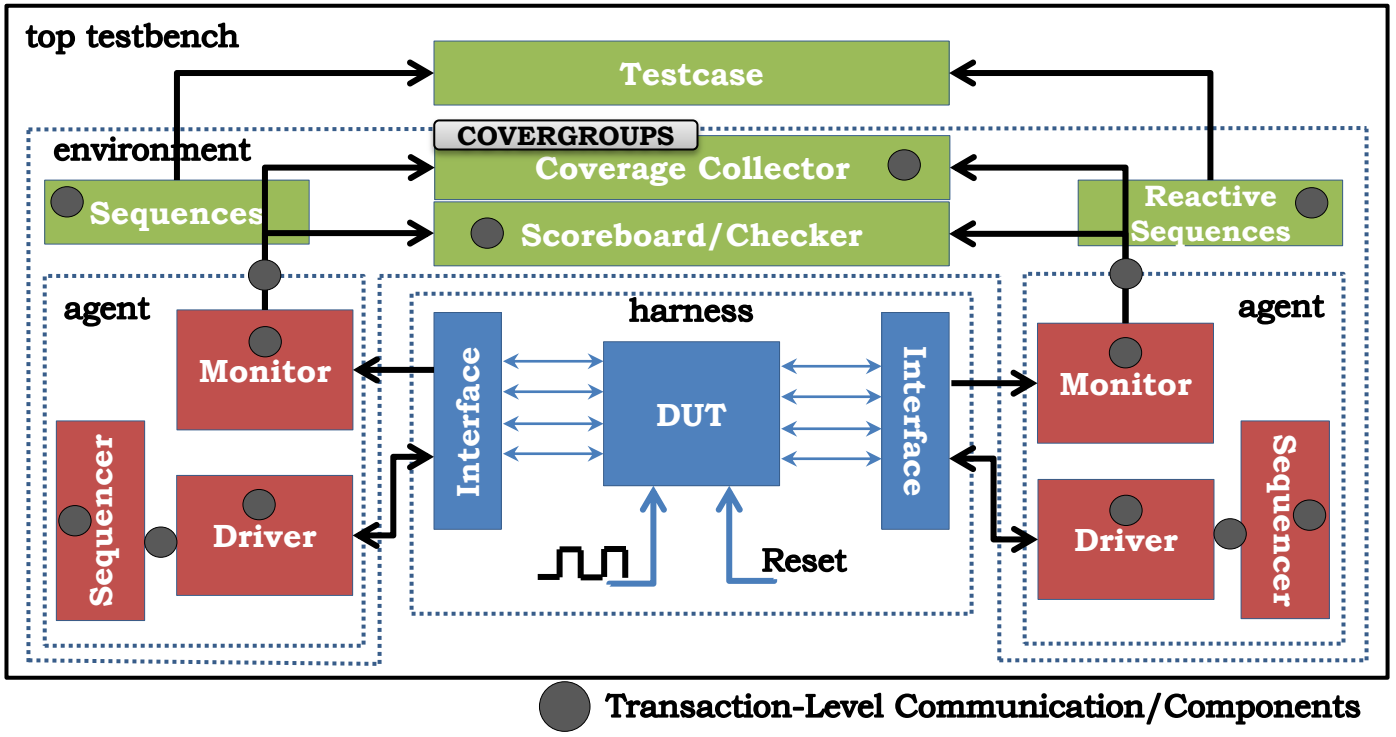
Enabling an UVM Testbench for simulation acceleration

accellera SYSTEMS INITIATIVE™

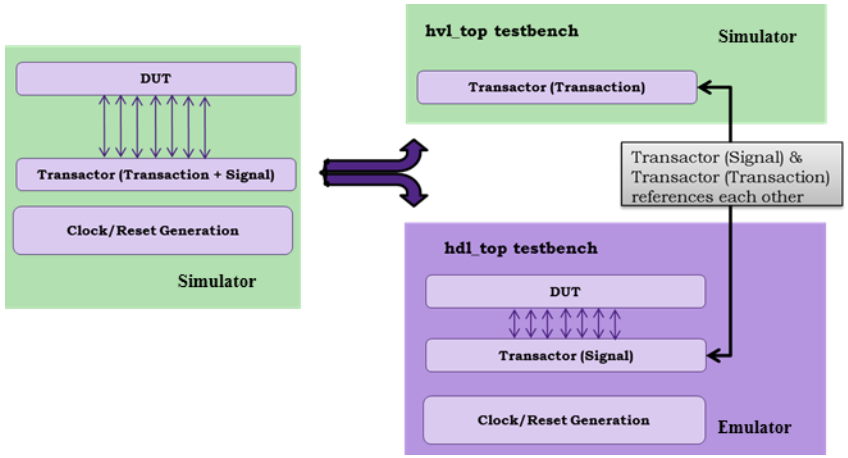
2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Typical UVM Testbench Environment

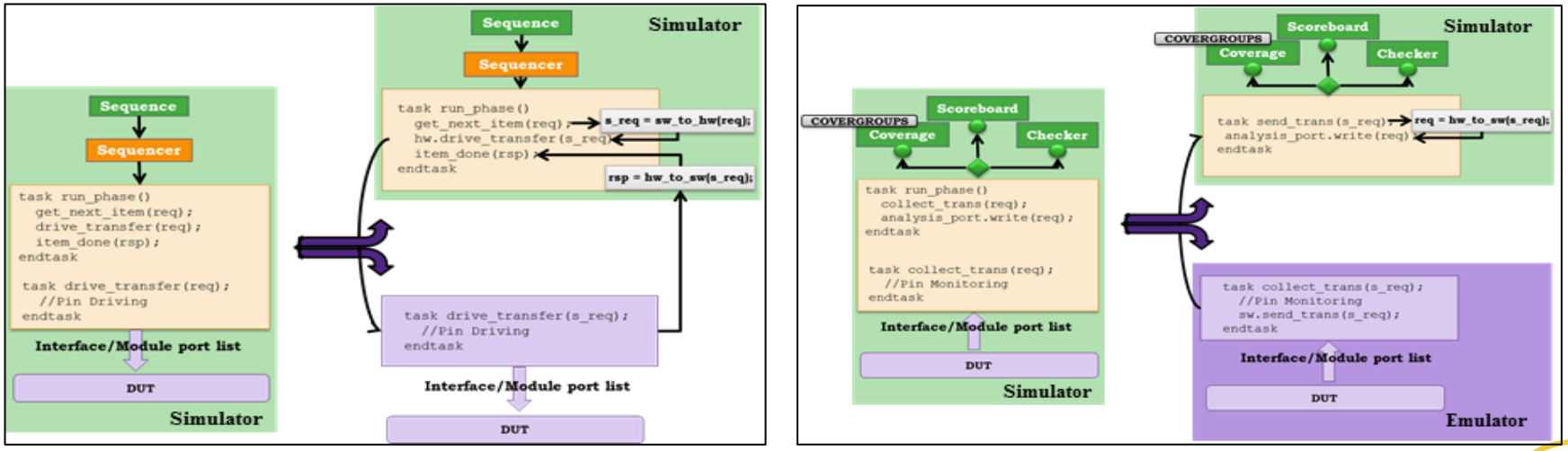
- Stimulus Generation: Constrained Random, Object-Oriented
- Testcase: Simulation Control
- Bus Functional Models (BFM's)
- Responder, Slave (Memory)
- Result Analyzers
- Coverage Driven Verification
- Scoreboards



Environment Transformations for TBA

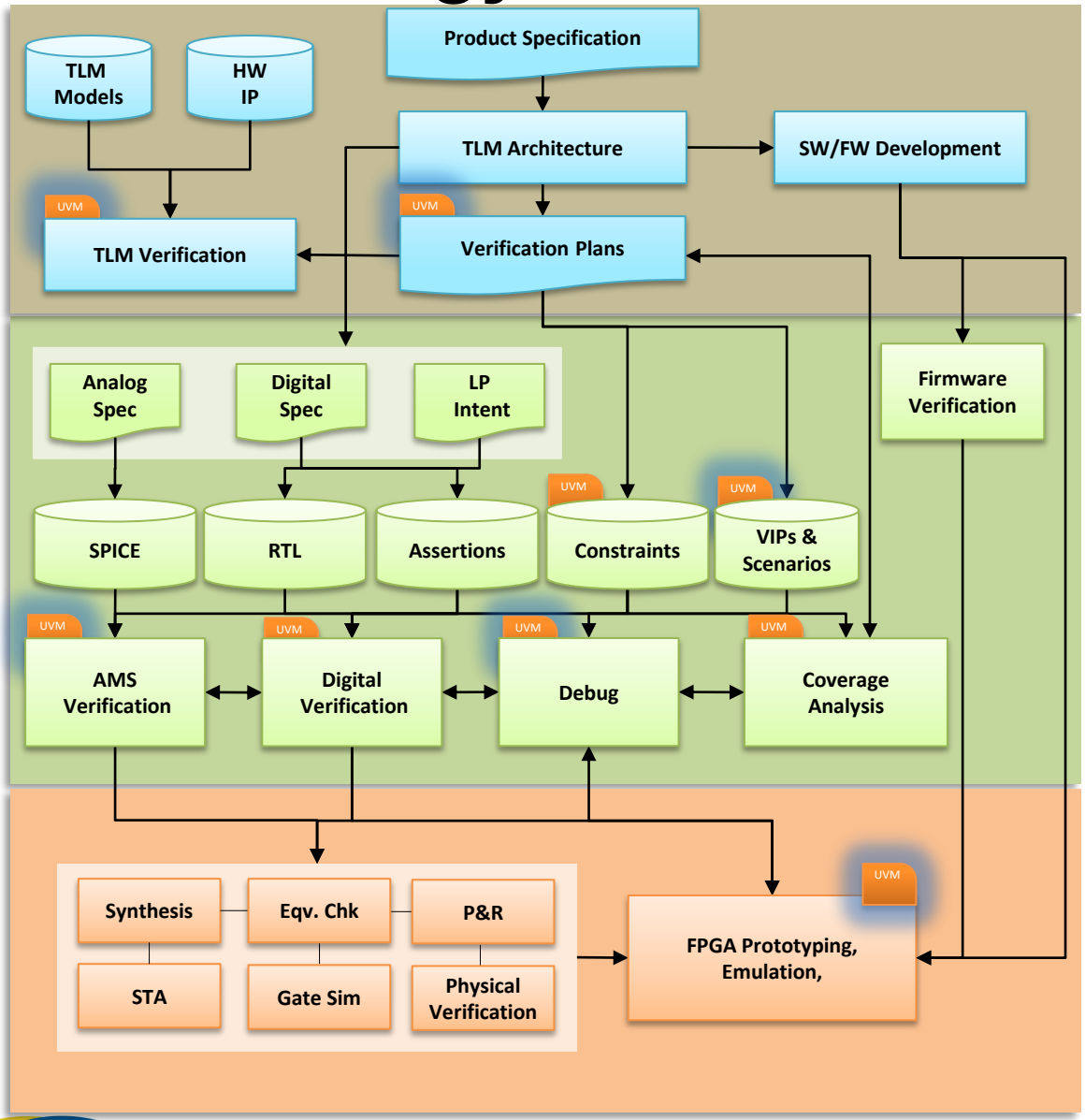


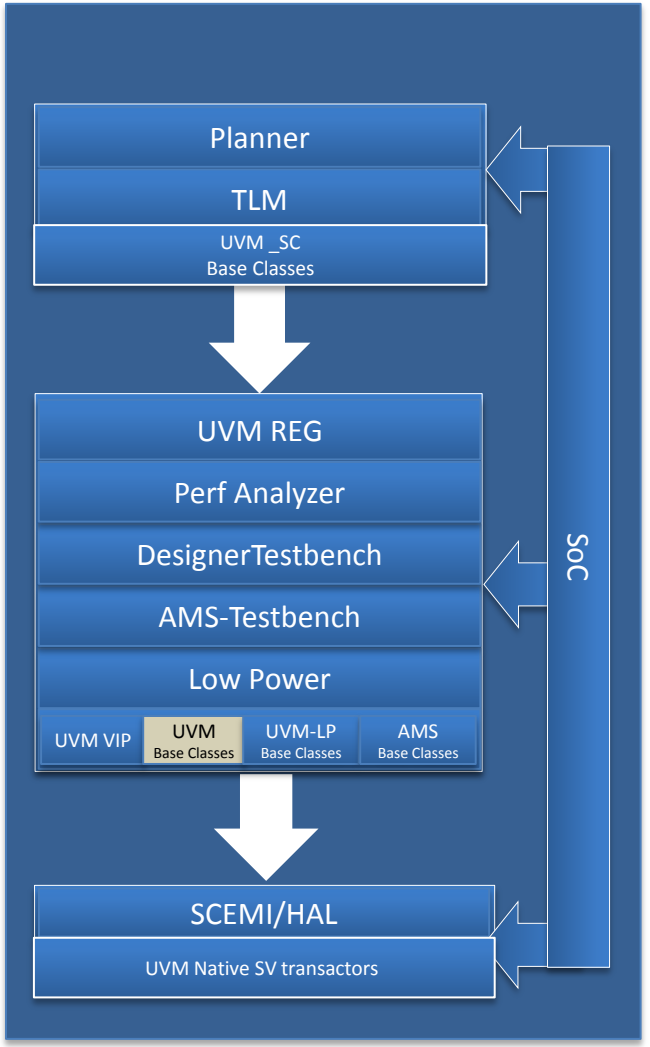
Two-Top Entity Approach



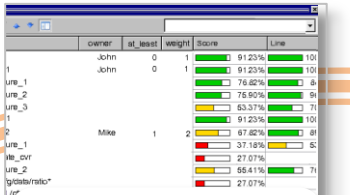
Partitioning the Active & Passive BFM's

Methodology Investment - Direction

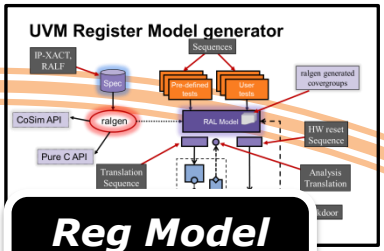




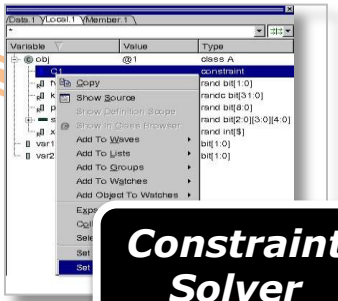
Complete UVM Ecosystem



Verification Planning

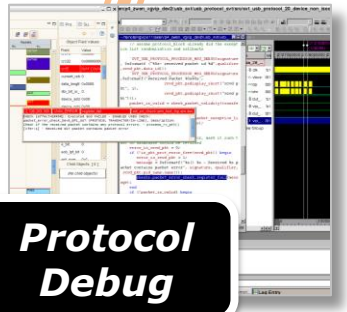


Reg Model generators



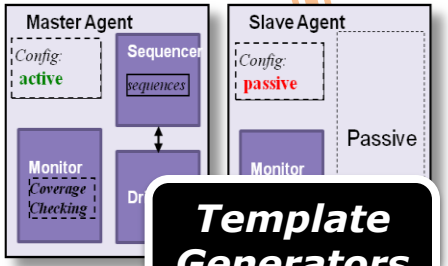
Constraint Solver

Coverage & Analysis

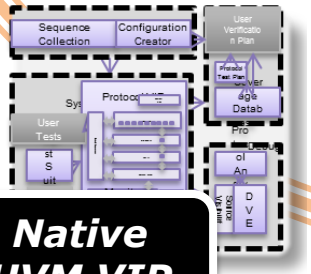


Protocol Debug

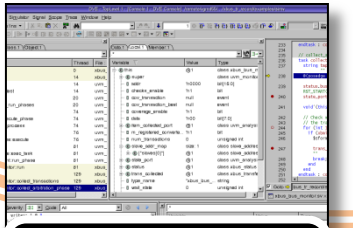
UVM



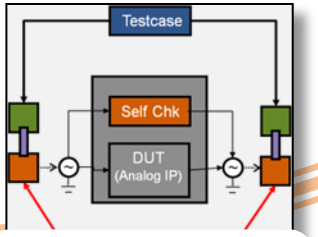
Template Generators



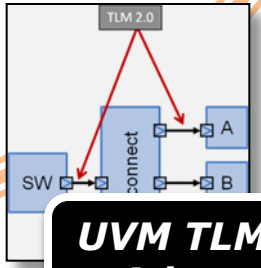
Native UVM VIP



UVM-Aware Debug



UVM AMS Testbench



UVM TLM/SC Adaptors

accellera SYSTEMS INITIATIVE™

2015 CONFERENCE AND EXHIBITION INDIA

Questions