2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

Scalable agile processor verification using SystemC UVM and friends

Eyck Jentzsch, MINRES Technologies GmbH

MINRES® TECHNOLOGIES

accellera SYSTEMS INITIATIVE

Agenda

- Quick Introduction into Universal Verification Methodology
  Basic concepts of UVM
- UVM in SystemC
  Implementation of the UVM concepts in SystemC/C++
- Example of a UVM-SystemC testbench for processor verification
  Use of UVM-SystemC (and friends) in a RISC-V core verification environment

2022
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
EUROPE
MUNICH, GERMANY
DECEMBER 6 - 7, 2022

UVM Introduction

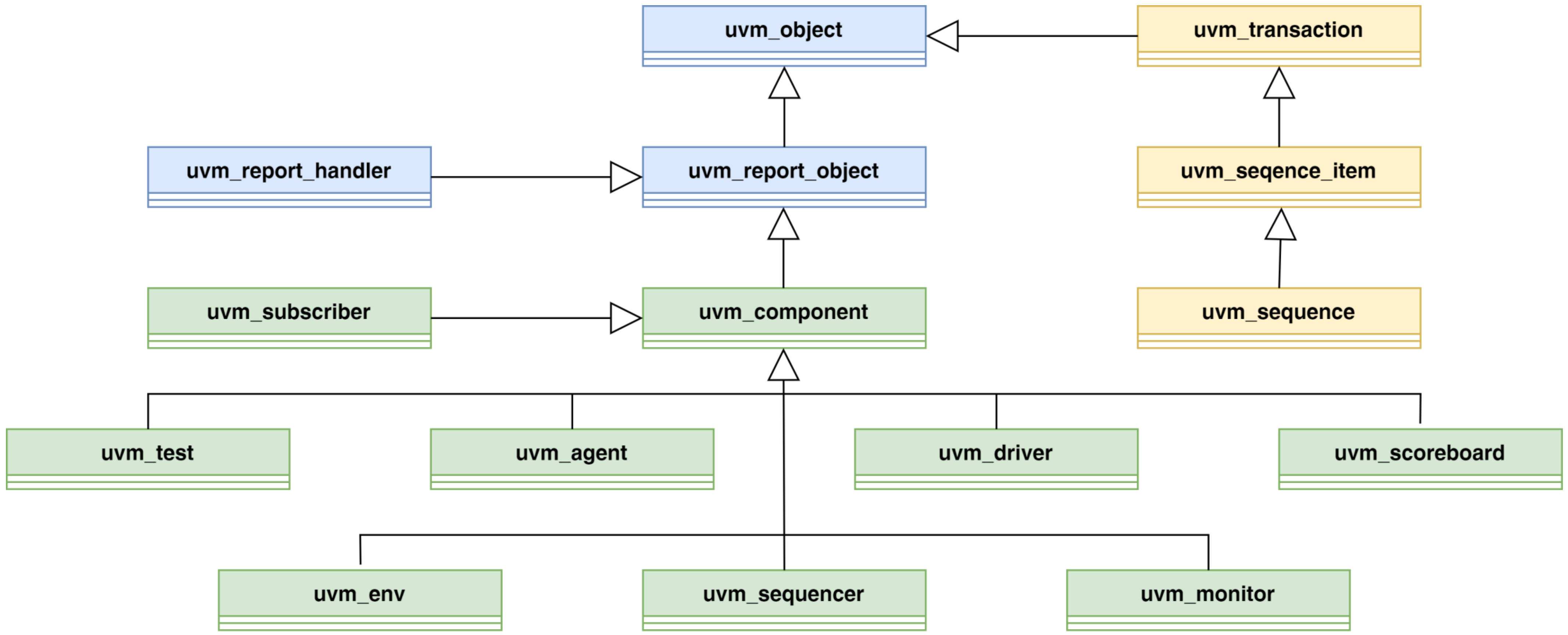
Just a quick one

accellera
SYSTEMS INITIATIVE

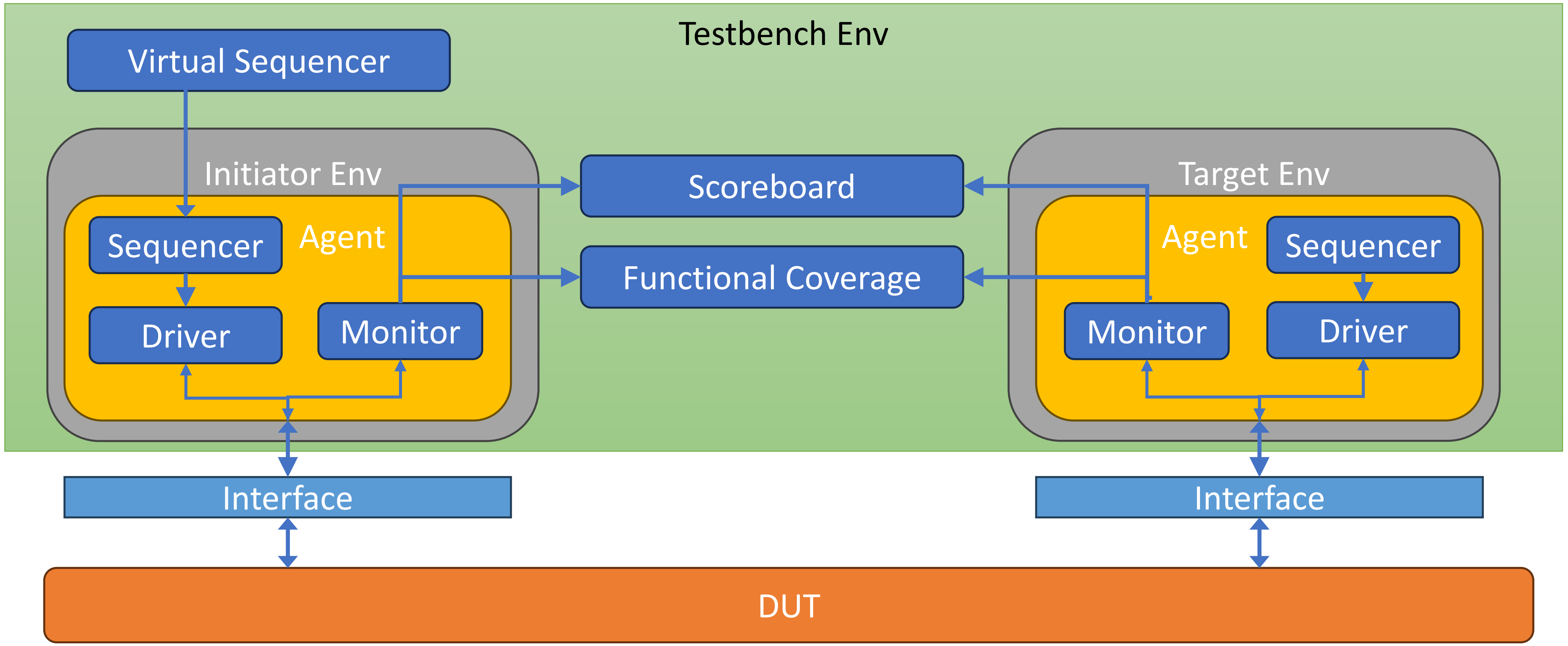
UVM Overview

- A library of base classes for building testbench components (Agent, Sequencer, Driver, Monitor, Scoreboards, Environment class etc)
- A factory for constructing objects and substituting objects
- Verification phases for synchronizing concurrent processes
- A reporting mechanism for a consistent way of printing and logging results
- Transaction Level Modeling (TLM) for communication between verification components
- Macros to semi-automate generation of required UVM code.

UVM Class Hierarchy



UVM Testbench Structure



UVM Simulation Phases

- build_phase
  instantiate all testbench objects using the factory
- connect_phase
  connect ports and signals of the testbench
- run_phase
  simulate the DUT and the stimulus
- report_phase
  collect the results of the simulation and report them

References

- UVM User Guide
  https://accellera.org/images/downloads/standards/uvm/uvm_users_guide_1.2.pdf
- UVM Reference Manual
  https://accellera.org/images/downloads/standards/uvm/UVM_Class_Reference_Manual_1.2.pdf
- UVM Library
  https://accellera.org/images/downloads/standards/uvm/UVM-18002-2020-20tar.gz

2022
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
EUROPE
MUNICH, GERMANY
DECEMBER 6 - 7, 2022

UVM in SystemC

accellera
SYSTEMS INITIATIVE

Libraries in a UVM-SystemC environment

- UVM-SystemC
- FC4SC
- CRAVE
- SCV

UVM-SystemC

- Open source C++ and SystemC based class library developed to improve the structure and reusability of the verification environments
- Compatible with IEEE Std 1666-2011
- Provides common APIs, which are supported by the major simulators
- Targets Coverage Driven Verification (CDV) with automated stimulus generation, independent result checking and coverage collection
- Allows reuse of tests and test benches across verification and validation platforms

UVM-SC Current State (I)

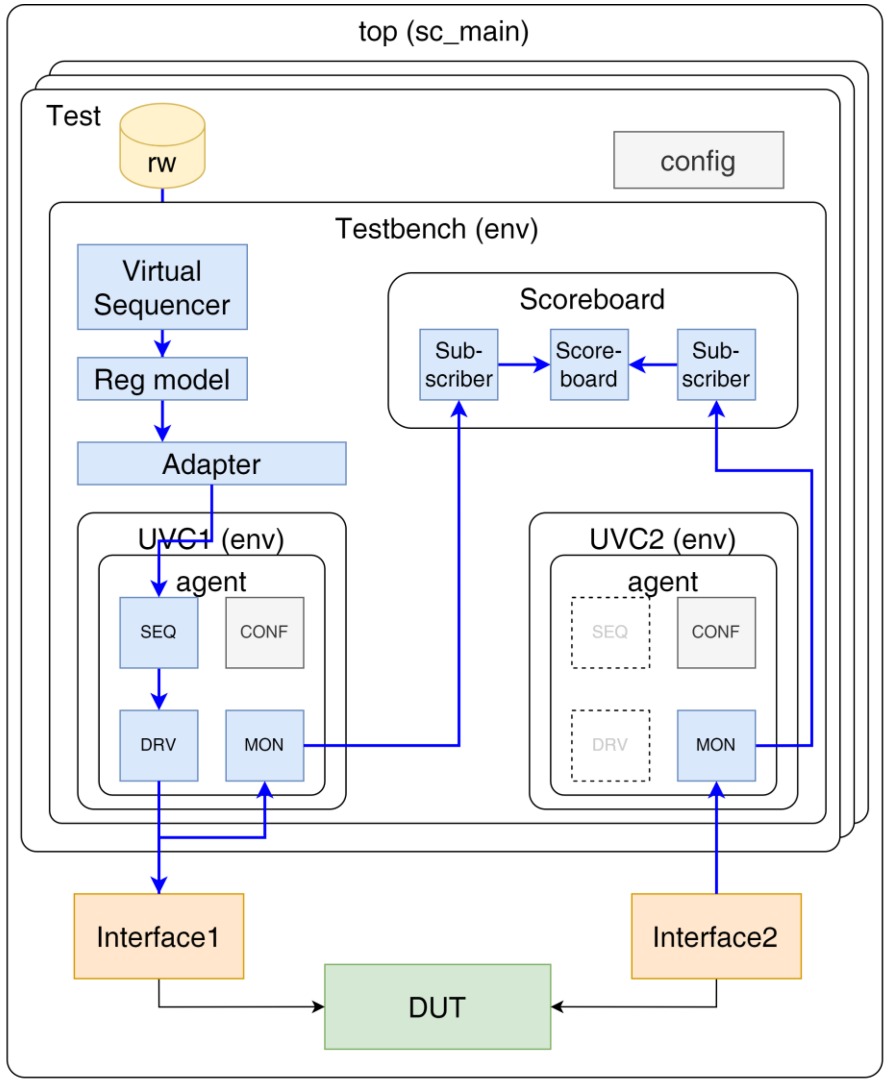
- Component classes to build the agents, sequences, drivers, etc. that comprise UVCs.
- UVCs are connected into testbenches with test and virtual sequences.
- Configuration and factory mechanisms
- Simulation control through phasing and objection handling.
- Print, compare, messaging for data management and debugging

UVM-SC Current State (II)

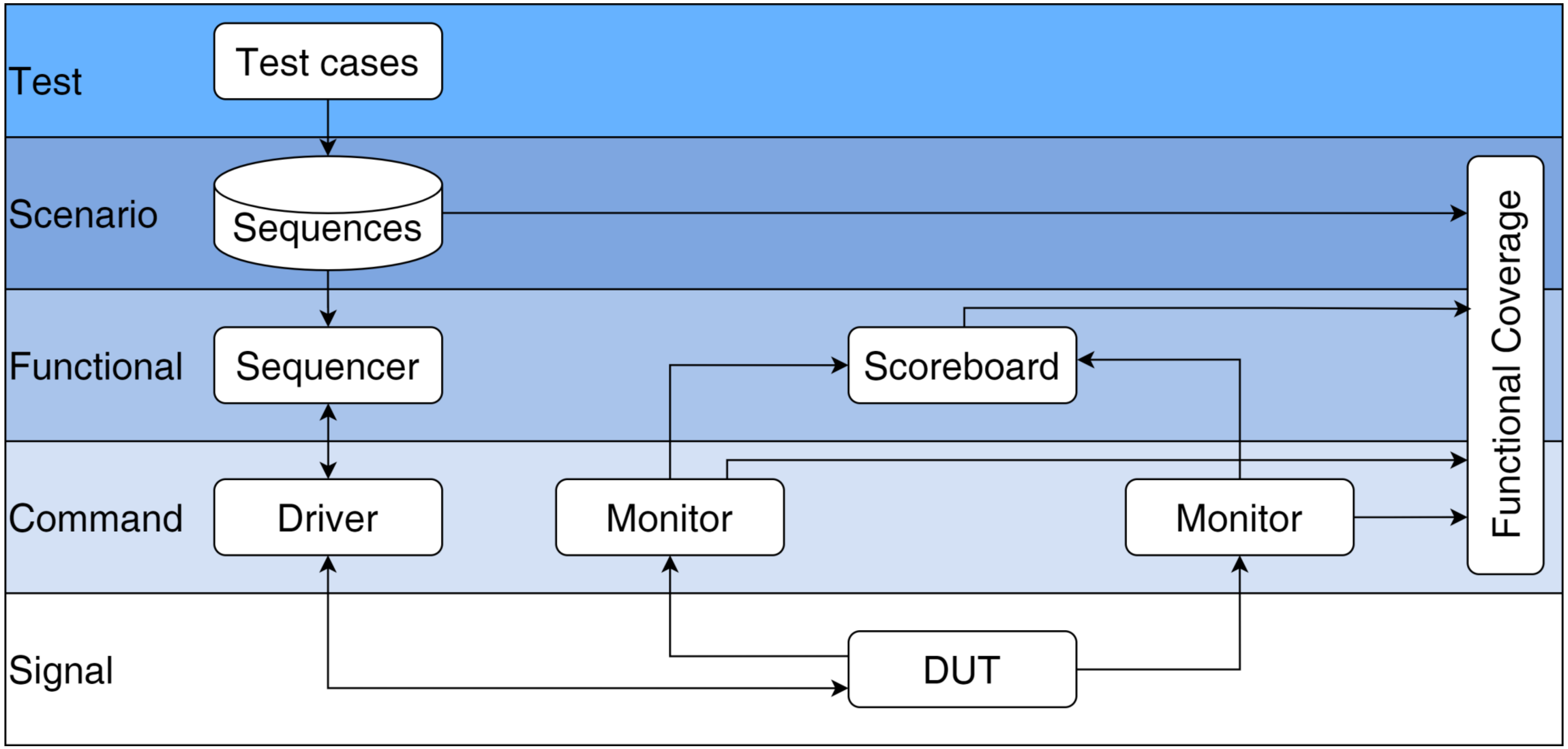
- In development (Beta 5)
  - Commandline processor, barrier and heartbeat missing
- Constrained randomization is in discussion with the Accellera SCV standard and a supplemental constraint solver (CRAVE) as possible solutions.

UVM-SC Layered Architecture

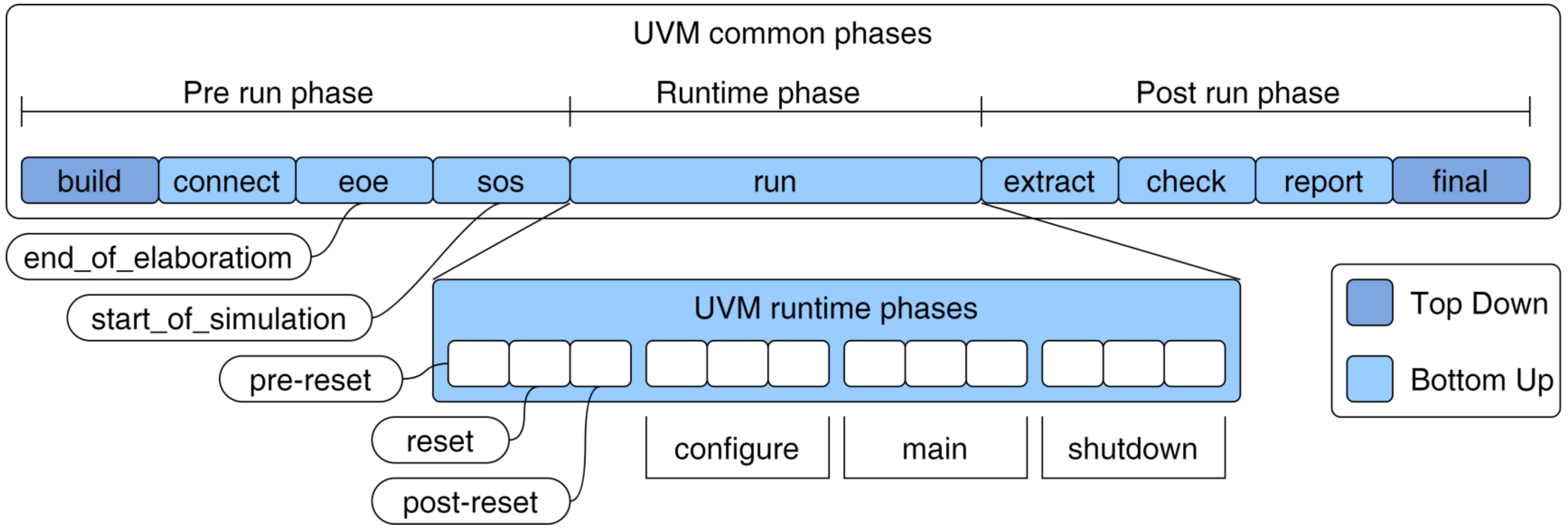
- The top-level (e.g. **sc_main**) contains the test(s), the DUT and its interfaces
- The DUT interfaces are stored in a configuration database, so it can be used by the UVCs to connect to the DUT
- The test bench contains the UVCs, register model, adapter, scoreboard and (virtual) sequencer to execute the stimuli and check the result
- The test to be executed is either defined by the test class instantiation or by the member function **run_test**



UVM-SC Layered Architecture



Phases of Elaboration and Simulation

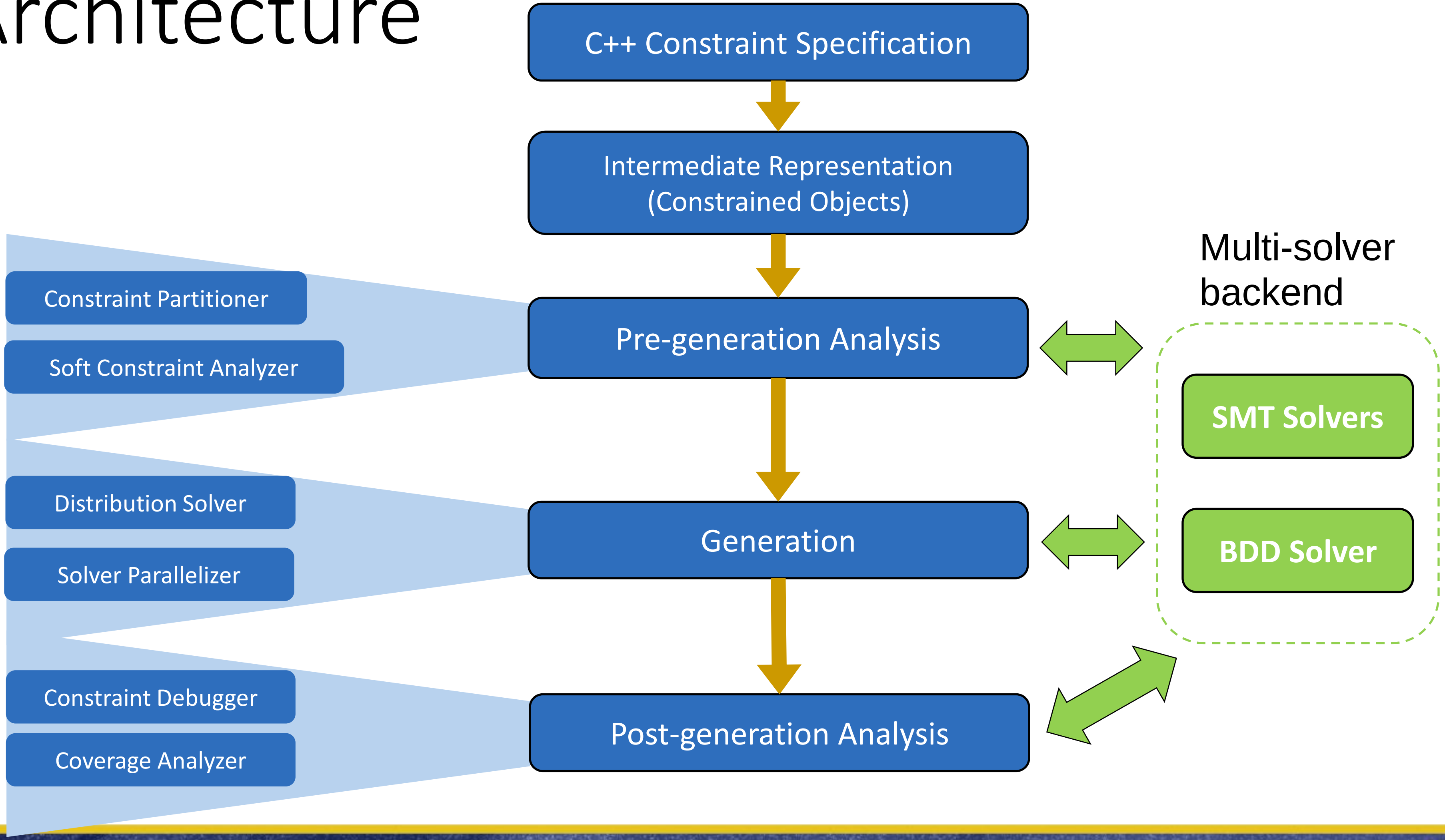


- UVM-SystemC phases made consistent with SystemC phases
- UVM-SystemC supports the 9 common phases and the (optional) refined runtime phases
- Objection mechanism supported to manage phase transitions
- Multiple domains can be created to facilitate execution of different concurrent runtime phase schedules

Randomization: CRAVE

- **C**onstrained **RA**ndom **V**erification **E**nvironment
- Syntax and semantics follow closely SystemVerilog IEEE 1800 std
- Random objects
- Random variables
- Hard/soft constraints
- Efficient constraint solvers
- MIT license

CRAVE Architecture



Randomization using CRAVE – Example

```
struct sysc_cont : public crv_sequence_item {
  crv_variable<sc_int <5>>    x{ "x" };
  crv_variable<sc_uint<6>>    y{ "y" };
  crv_variable<sc_bv  <7>>    z{ "z" };

  sc_uint<5> t = 13;

  crv_constraint constr{ "constr" };

  sysc_cont(crv_object_name) {
    constr = { dist(x(), make_distribution(range<int>(5, 8))), y() > 0, y() % reference(t) == 0, y() != y(prev),
               (z() & 0xF) == 0xE };
  }
};
```

Random variables

SystemC Datatypes

Constraint expression

special operator

distribution

operators

Special operators

- inside
- dist
- if_then
- if_then_else
- Foreach
- unique
- bitslice
- …

Coverage: FC4SC

- C++11 header only library:
  - built from scratch, with no 3rd party library dependencies
  - Based on IEEE 1800 - 2012 SystemVerilog Standard
- Features:
  - Coverage model construction
  - Coverage sampling control & options
  - Runtime coverage queries
  - Coverage database saving

FC4SC Elements

- Covergroup: encapsulates a set of coverpoints and crosses
- Coverpoint: defines
  - an expression to be sampled
  - a collection of bins containing values to be sampled
  - optionally, a boolean expression which conditions sampling
- Cross: is the cartesian product of its member coverpoints' bins.

FC4SC Example

```
class data_cvg : public covergroup {
public:
   int value = 0;
   int flags = 0;

   CG_CONS(data_cvg) {}

   COVERPOINT(int, values_cvp, value) {
            // intervals are inclusive
            bin<int>("low1", interval(1,6), 7),
            bin<int>("med", interval(10,16), 17),
            bin<int>("high", interval(20,26), 27)
   };
```

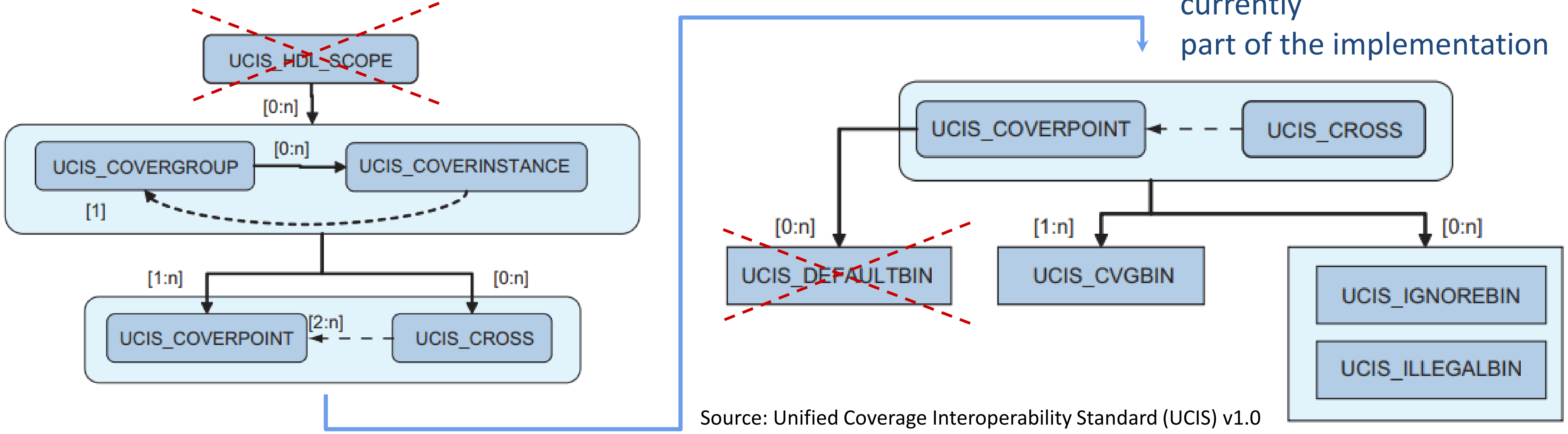
```
   COVERPOINT(int, flags_cvp, flags) {
            bin<int>("zero", 0),
            bin<int>("one", 1),
            bin<int>("ten", 10),
            illegal_bin<int>("illegal_config", 3),
            ignore_bin<int>("uninteresting", 8)
   };


   // Cross (cartesian product) of the two
   // coverpoints
   auto valid_data_cross = cross<int,int>(
          this, &flags_cvp, &values_cvp);
};
```

Coverage Definition API: Overview

- Follows UCIS DB coverage data model
  - Elements: bin, coverpoint, cross, covergroup

Crossed out elements are not currently
part of the implementation



Source: Unified Coverage Interoperability Standard (UCIS) v1.0

MIN

References

- UVM-SystemC Library
  https://accellera.org/images/downloads/drafts-review/uvm-systemc-1.0-beta5.tar.gz
- Crave Library
  https://github.com/accellera-official/crave
- FC4SC Library
  https://github.com/accellera-official/fc4sc

2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

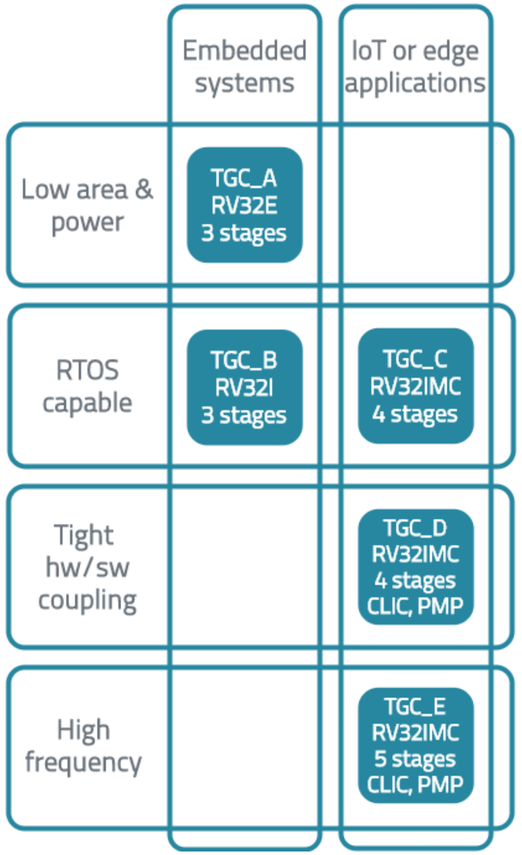
MUNICH, GERMANY
DECEMBER 6 - 7, 2022

UVM-SystemC Example

A testbench for processor family verification

accellera SYSTEMS INITIATIVE

TGC RISC-V Family



	Embedded systems	IoT or edge applications
Low area & power	TGC_A RV32E 3 stages	
RTOS capable	TGC_B RV32I 3 stages	TGC_C RV32IMC 4 stages
Tight hw/sw coupling		TGC_D RV32IMC 4 stages CLIC, PMP
High frequency		TGC_E RV32IMC 5 stages CLIC, PMP

- Part of ‘The Good Folk Series’ (TGFS)
- Highly flexible, scalable and extendable
- Single issue in-order pipeline
- Standard configurations as starting points
- Easy combinations of features and options
  - ***Different bus interfaces***
  - Interrupt controllers
  - Processor caches
  - ***Custom instructions***
  - Safety features (lockstep, GPRs parity bits, ECC)
  - Security (physical memory protection)

Disclaimer

‚While this guide offers a set of instructions to perform one or more specific verification tasks, it should be supplemented by education, experience, and professional judgment. Not all aspects of this guide may be applicable in all circumstances.‘

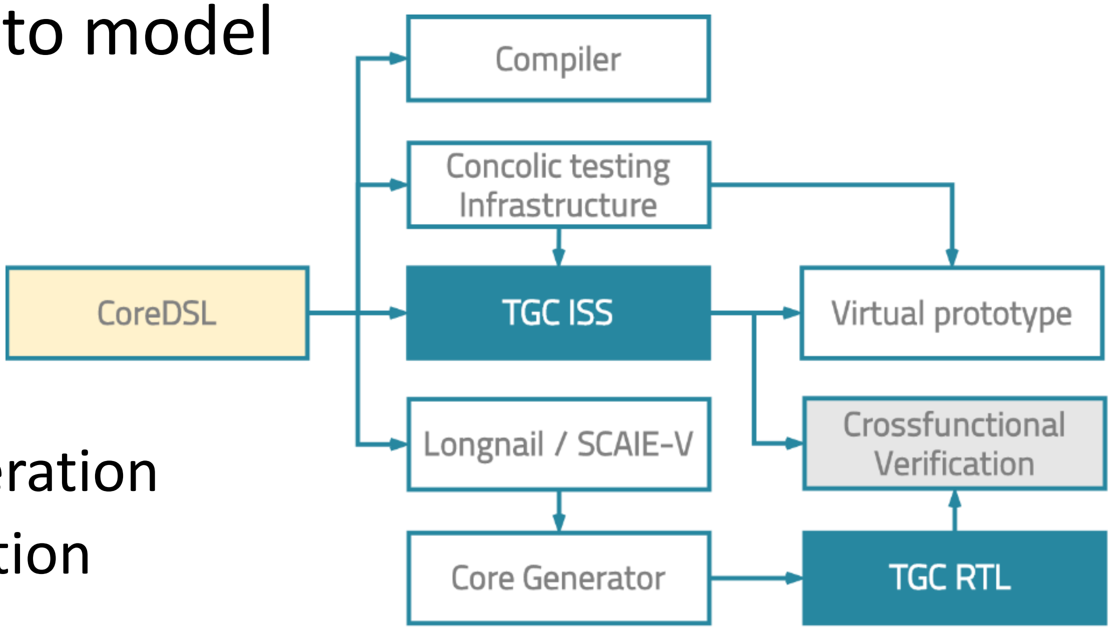
Universal Verification Methodology

(UVM) 1.2 User’s Guide

October 8, 2015

Single Source of Truth

- CoreDSL: Domain-specific language to model processor cores at the level of their *instruction set architecture* (ISA)
- Automatically generated:
  - Accurate ISS reference model
  - Configuration for random stimuli generation
  - List of instructions for coverage collection
  - Properties for formal verification
  - Artifacts for toolchain compatibility
  - Hardware for custom instructions



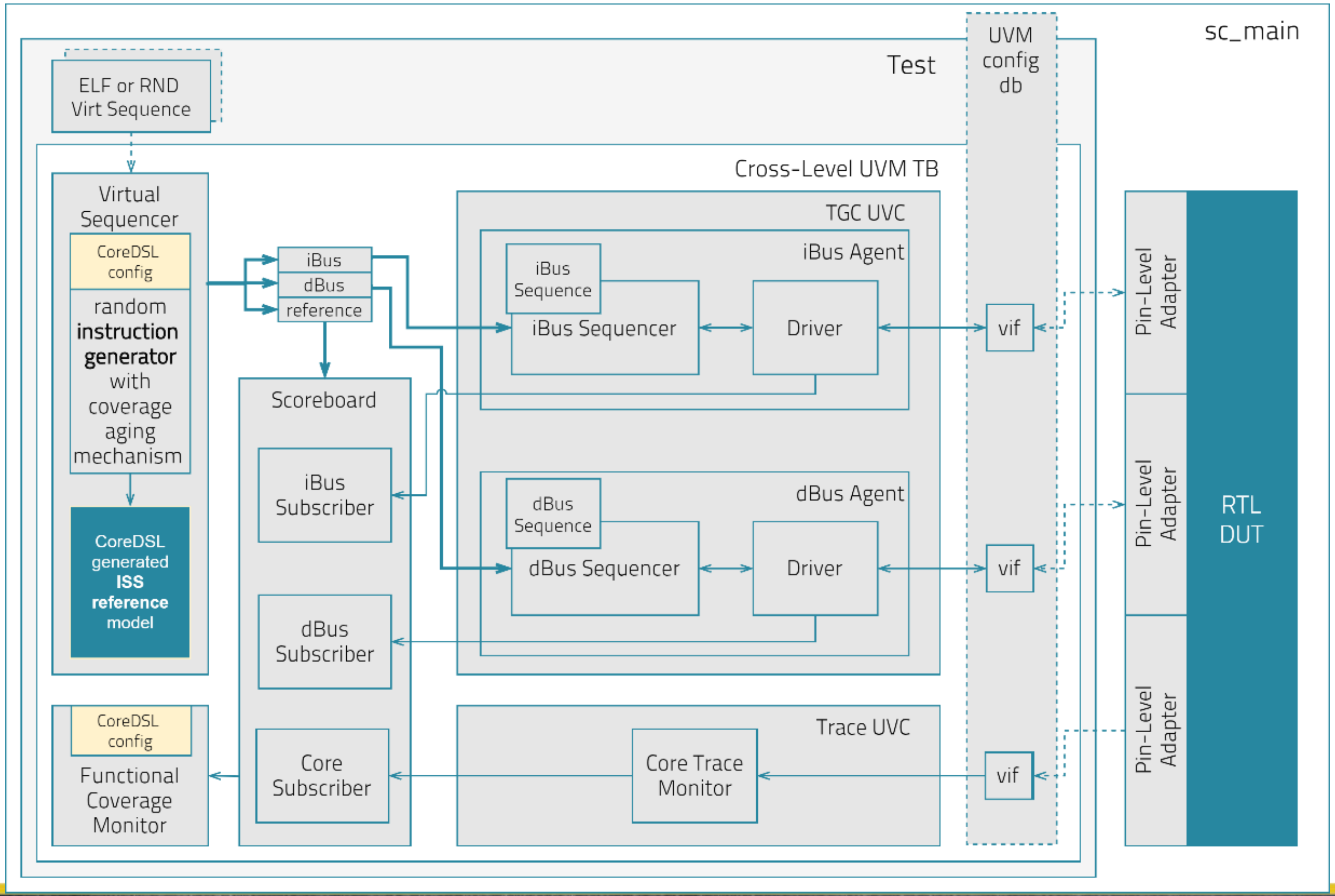
CoreDSL

- Open Source Specification & Frontend
- Contents:
    - Architectural state
        - Implementation parameter definition
        - General purpose register file
        - Single register with attribute
    - Instructions
        - Instruction name
        - Specification of instruction encoding
        - Functional behavior
- Simple definition of custom instructions using C-like syntax

```
Core My32bitRISCVCore {
  architectural_state {
     unsigned int    REG_LEN = 32;
     unsigned int    XLEN = 32;
     register unsigned<XLEN>  X[REG_LEN];
     register unsigned<XLEN>  PC [[is_pc]];
  }
  instructions {
    LUI {
      encoding: imm[31:12] :: rd[4:0] :: 7'b0110111;
      behavior: if (rd != 0) X[rd] = imm;
    }
  }
}
```

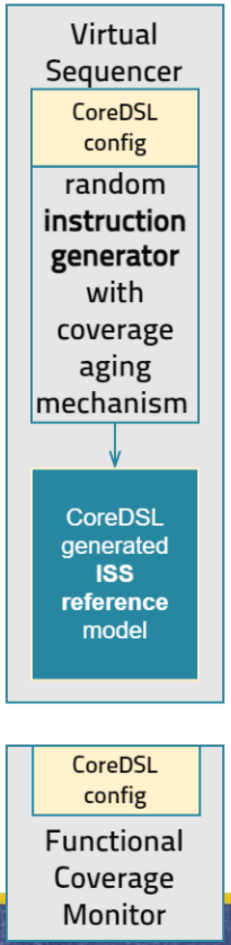
Cross-Level TB Overview

- UVM-SystemC Testbench
  - Seamless integration of generated components
  - Instruction Generator sends random instructions to ISS and RTL
  - ISS behavior and state compared with RTL results in the Scoreboard
- TB operation modes:
  - Pseudo-random instruction generation with aging based feedback
  - Load and execute ELF file



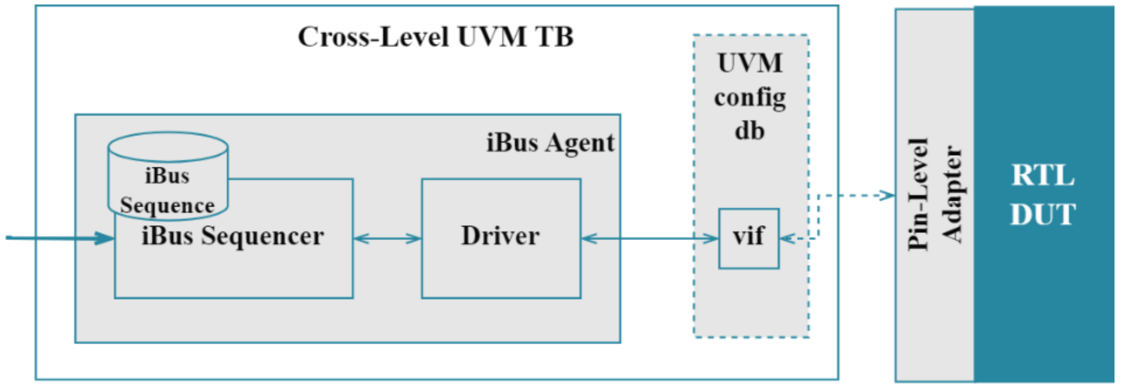
CoreDSL generated Components

- Virtual sequencer:
  - Instruction generator + ISS reference model
  - Instruction generator dynamically adjusts instruction frequency for optimized coverage
  - Coverage aging mechanism speeds up coverage achievement
- Instruction accurate ISS model
  - DBT-RISE infrastructure is the basis for the reference model
- Functional coverage monitor:
  - Utilizes information from CoreDSL description to accurately track and report coverage metrics


Virtual Sequencer
CoreDSL config
random instruction generator with coverage aging mechanism
CoreDSL generated ISS reference model
CoreDSL config
Functional Coverage Monitor

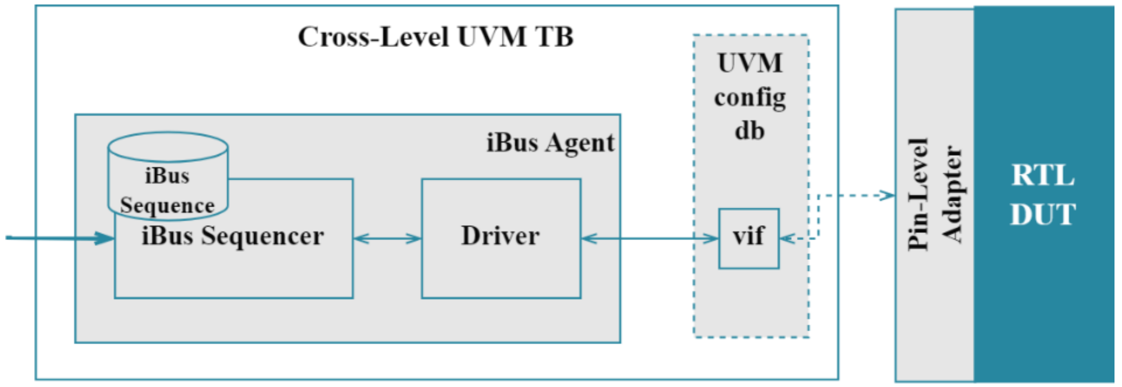
UVM-SystemC TB Agents

- iBus and dBus agents: sequencer + driver
    - Connected to the DUT through a virtual interface (vif)
    - The DUT initiates instruction fetches as well as data bus accesses over vif



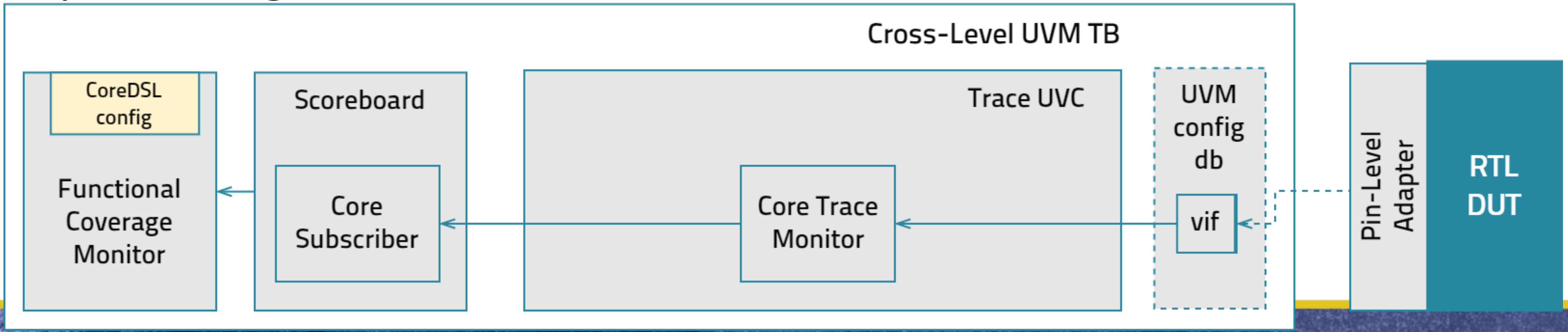
UVM-SystemC TB Agents

- vif enables communication without being tied to a specific implementation
  - DUT can be exchanged without changing the interface itself
  - Simulation engine can be exchanged
    - Verilator via SystemC wrapper and Pin-Level adapter
    - SystemVerilog Simulator via UVM Connect
    - Hybrid simulation with FPGA using RAVEN



UVM-SystemC TB Agents

- Trace interface maps the internal state of the core
  - Register values
  - Program counter
  - Traps
- The scoreboard analyzes and compares the iBus, dBus, and trace monitor sequences against the reference.

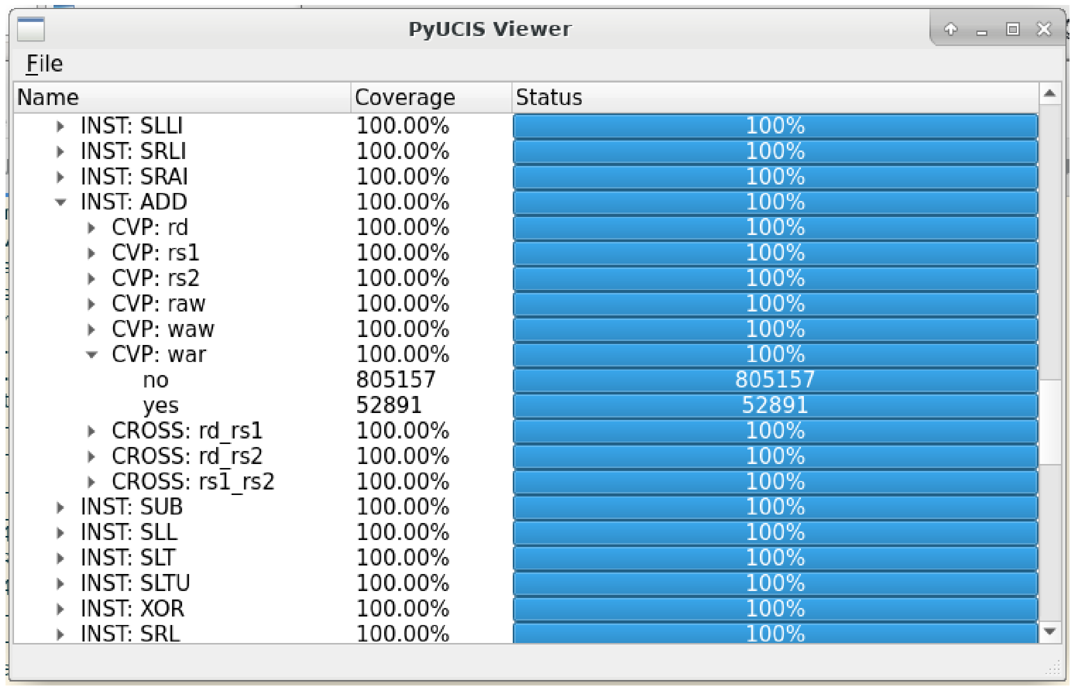


Live Code Demo

Functional Coverage

- Coverage Monitor:
  - Defines coverage for different instruction types
  - Coverpoints: coverage criteria for instructions
    - Parameters
    - Dependencies
    - Hazards
  - Covergroups: Summarize coverage information

- Functional coverage provides:
  - Parameter toggling frequency
  - Cross-coverage analysis
  - Identification of data hazards



PyUCIS Viewer

File

Name	Coverage	Status
INST: SLLI	100.00%	100%
INST: SRLI	100.00%	100%
INST: SRAI	100.00%	100%
INST: ADD	100.00%	100%
CVP: rd	100.00%	100%
CVP: rs1	100.00%	100%
CVP: rs2	100.00%	100%
CVP: raw	100.00%	100%
CVP: waw	100.00%	100%
CVP: war	100.00%	100%
no	805157	805157
yes	52891	52891
CROSS: rd_rs1	100.00%	100%
CROSS: rd_rs2	100.00%	100%
CROSS: rs1_rs2	100.00%	100%
INST: SUB	100.00%	100%
INST: SLL	100.00%	100%
INST: SLT	100.00%	100%
INST: SLTU	100.00%	100%
INST: XOR	100.00%	100%
INST: SRL	100.00%	100%

Live Code Demo

References

- CoreDSL
  https://github.com/Minres/CoreDSL
- RISC-V ISA as CoreDSL:
  https://github.com/Minres/RISCV_ISA_CoreDSL
- SystemC Components Library:
  https://github.com/Minres/SystemC-Components
- PyUCIS:
  https://github.com/fvutils/pyucis
- PyUCIS Viewer:
  https://github.com/fvutils/pyucis-viewer

Questions