Introduction to the Integrated Circuit Verification Process and RD53C Results

Jelena Lalic *<Jelena.lalic@cern.ch>*

OUTLINE

- **Introduction to the UVM**
- **RD53C Chip and its Verification Framework**
- **SEE Verification Flow and Environment**

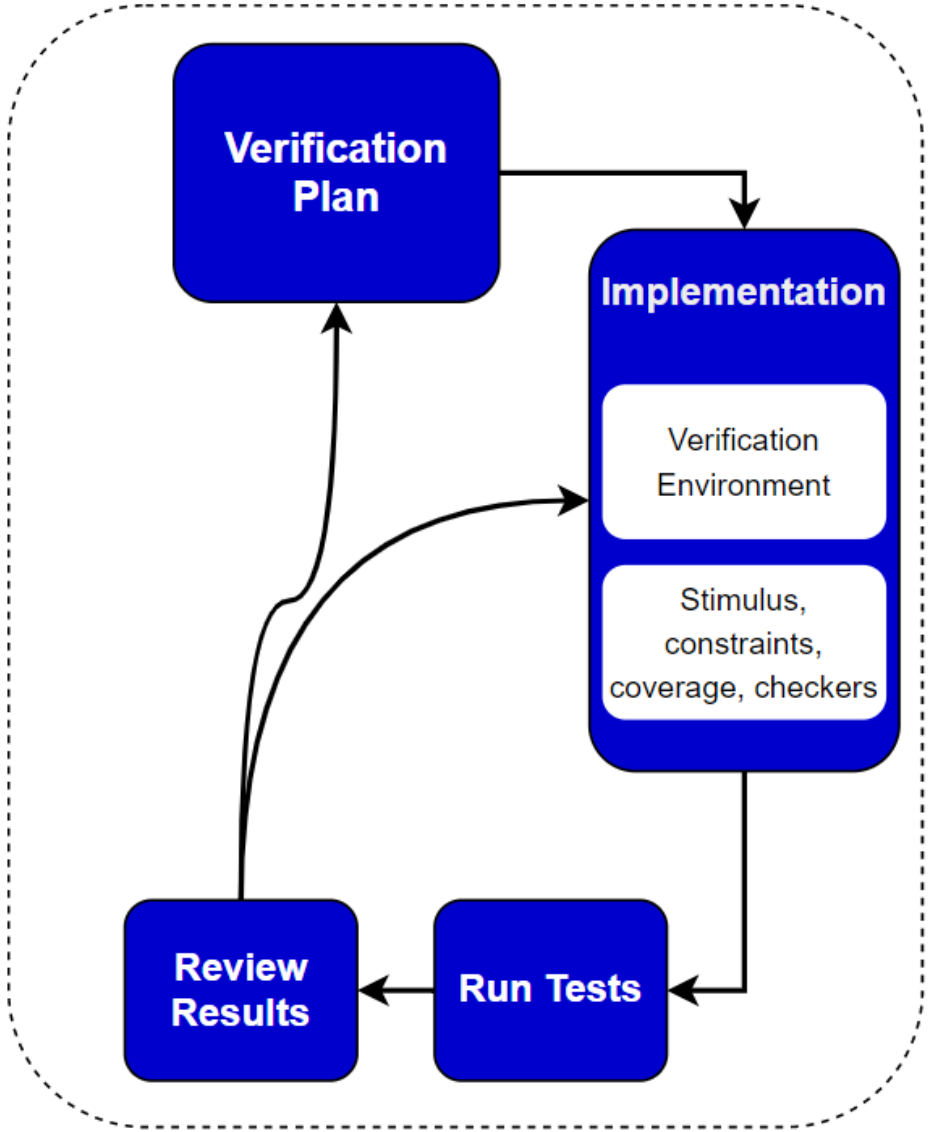
Universal Verification Methodology (UVM)

- Methodology for writing testbenches
- Open-source standard written in SystemVerilog
- Compatible and portable
- Development of well constructed and reusable verification environment

UVM codifies best practices for efficient verification.

- Constrained-random verification central to UVM
- Metric-driven verification

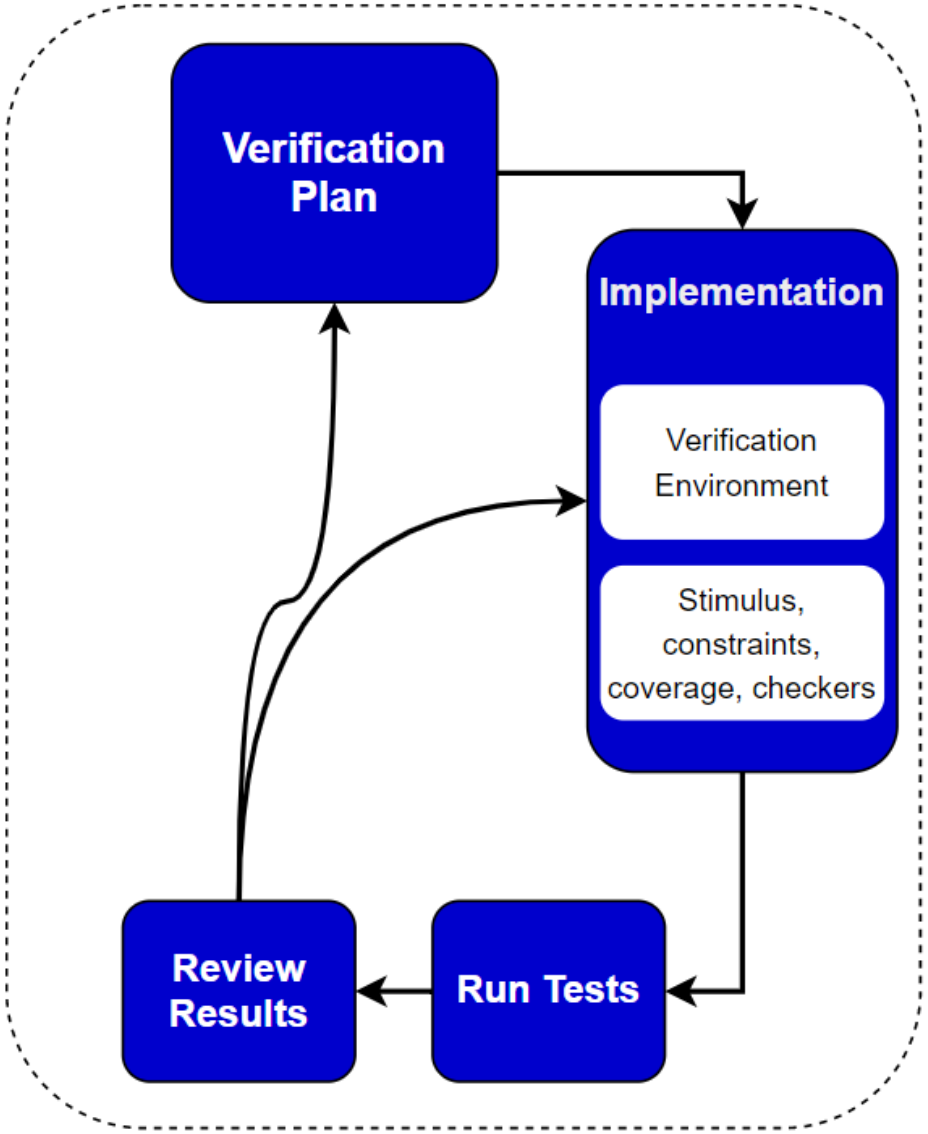
Verification Planning



Verification Plan:

- Features of the DUT are identified and prioritized for verification
- Define high-level verification goals: Coverage model

Metric-Driven Verification



- Implementation of Verification Environment
  - Generate and apply constrained-random stimuli
  - Coverage model and monitors
  - Checkers
- Simulation
- Collect and review coverage
- Annotate coverage results back onto verification plan or/and verification environment
- Run further tests to close coverage holes

- Coverage goal is 100%

Coverage Metrics

- **Code Coverage**
  - how much the RTL code is exercised
  - different metrics (statement, toggle, FSM, …)
  - automatically extracted by a simulator when enabled
- **Functional coverage**
  - how much design functionality has been exercised
  - defined by verification engineers in the form of a functional coverage model

Effective verification planning is a combination of:

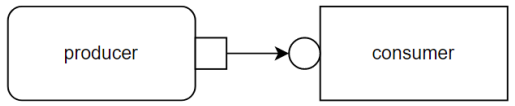
- Code coverage
- Functional coverage
- Automated checking

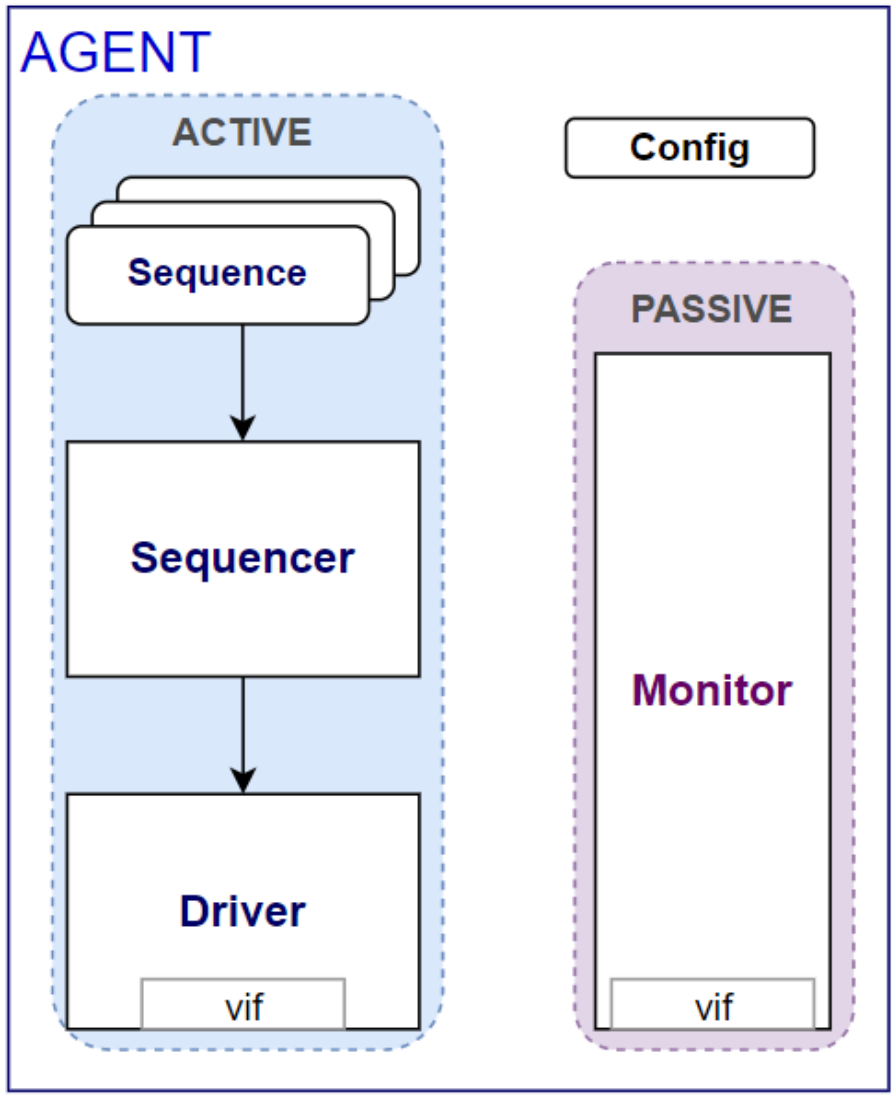
UVM Agent

Agent:

- Conventional way to structure the verification environment
- Agent provides protocol-specific stimuli creation, checking, and coverage
- Typically models either a master or slave component (in a bus-based environment)

Transaction Level Modeling (TLM)





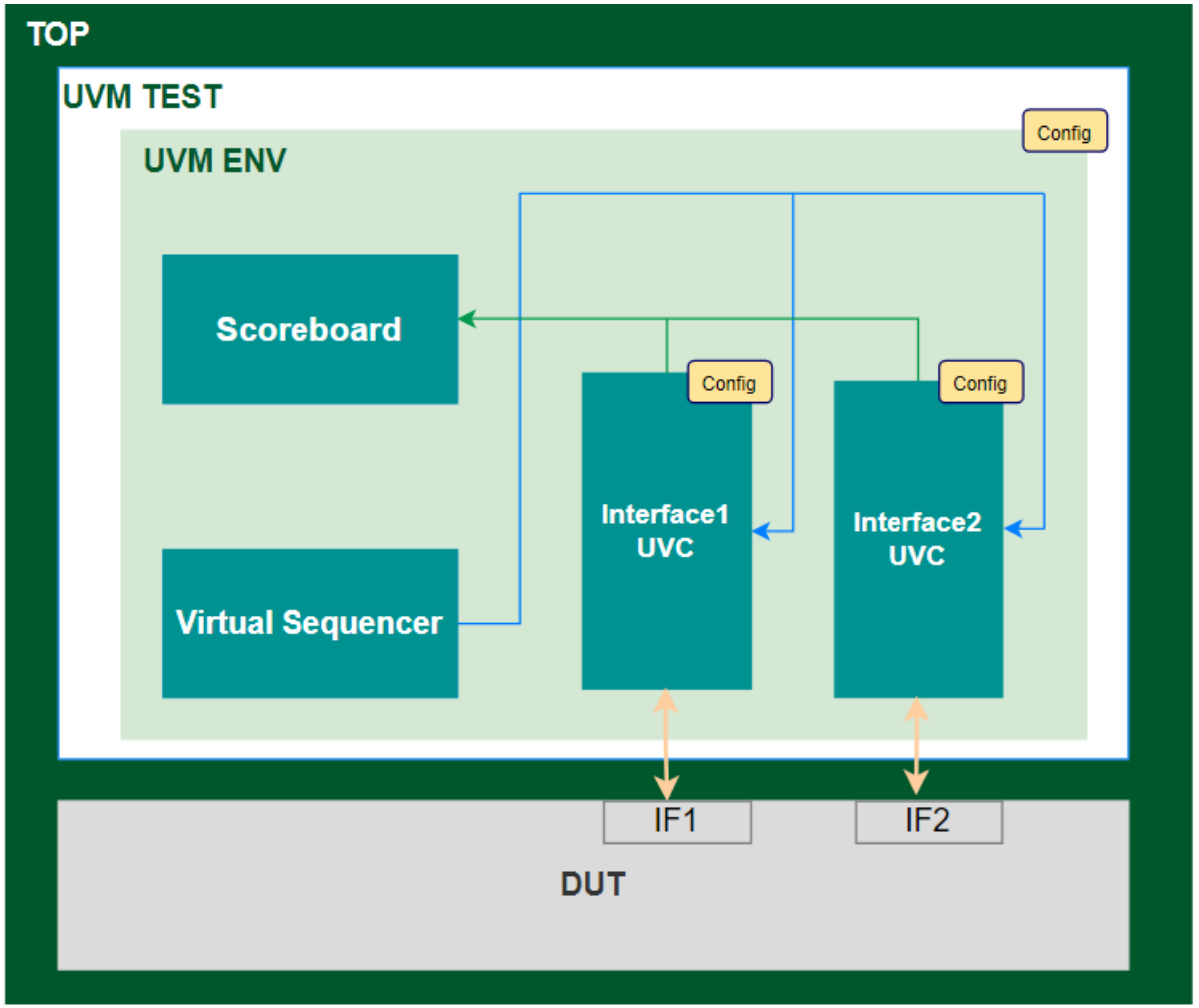
UVC

- Complete interface encapsulation level (multiple agents)
- Key building block in a verification environment
- Environment for a specific protocol
- Simplified integration
- Large-scale reuse

Virtual Interface

- Must be tied to their real counterparts
- Pointer to real interface

UVM Environment



Virtual Sequencer:

- Central control over multiple UVCs
- Not attached to a driver
- Does not process items itself
- Coordinating execution of other sequences via handles

Scoreboard:

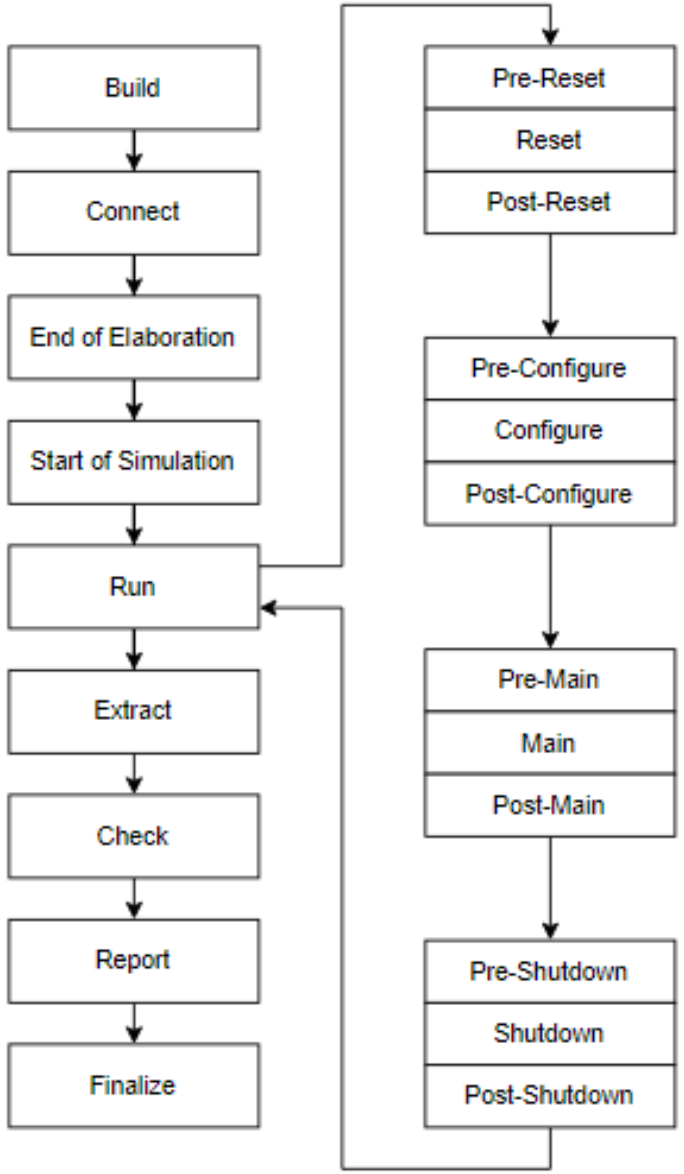
- A central element of a self-checking environment
- Verifies operation of a design at a functional level
- Track transaction level activity

Test and Environment:

- Single environment can be used by multiple tests
- Tests are not reusable (specific environment structure)
- Separating a test from the rest of a reusable environment

UVM Phases

- Simulation goes through different stages that arise from DUT behavior
- Reusable verification components might define different testbench activities for each stage



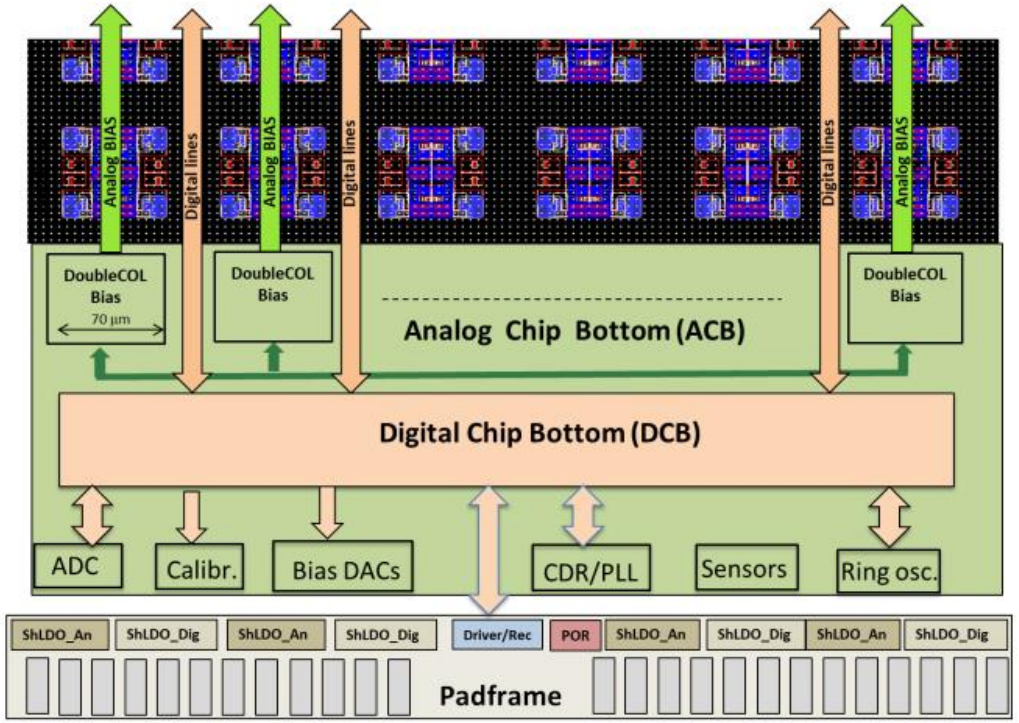
UVM Phases are a synchronizing mechanism for the environment

Reset: reset related activities
Config: device configurations
Main: sequences execution
Shutdown: graceful termination of the tests

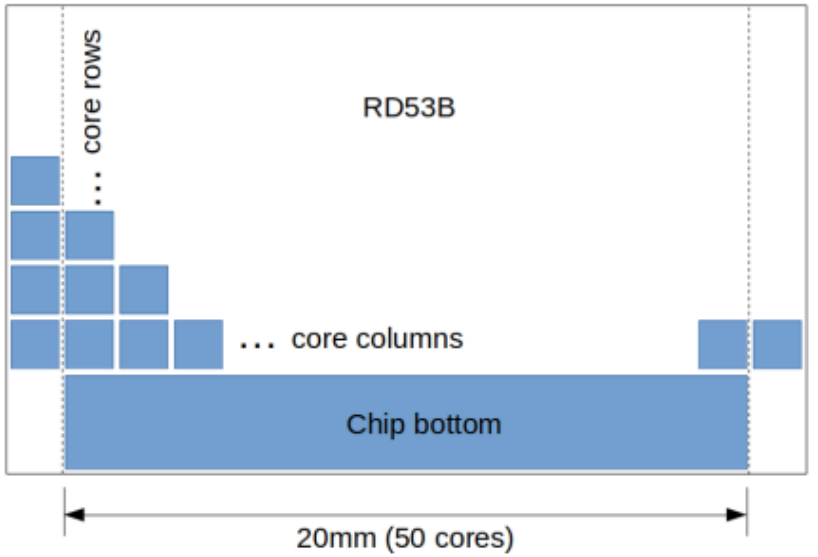
Run Tests

- Run tests after every RTL update
- Regression managers
  - Run test, automate coverage viewing, merging and analysis
- Close coverage
  - Use code coverage as a safety net to balance the functional coverage
- Sign off
  - A clean regression
  - Reach the coverage goal

Quick Introduction to the RD53C



Conceptual depiction:



Chip periphery

- Analog Chip Bottom (ACB): analog and mixed/signals building block for Calibration, Bias, Monitoring and Clock/Data recovery
- Digital Chip Bottom (DCB): synthesized digital logic
- Padframe: I/O blocks with ESD protections, ShuntLDO for serial powering

Design requirements:

- High hit rate 3 GHz/cm^2
- High trigger rate 1 MHz/750 kHz
- Hostile radiation environment: 1 Grad over 10 years, 10^{16} hadrons/cm^2
- Low power, Serial powering
- High SEE tolerance capability to assure reliable chip operation in the HL-LHC

RD53 Verification Framework

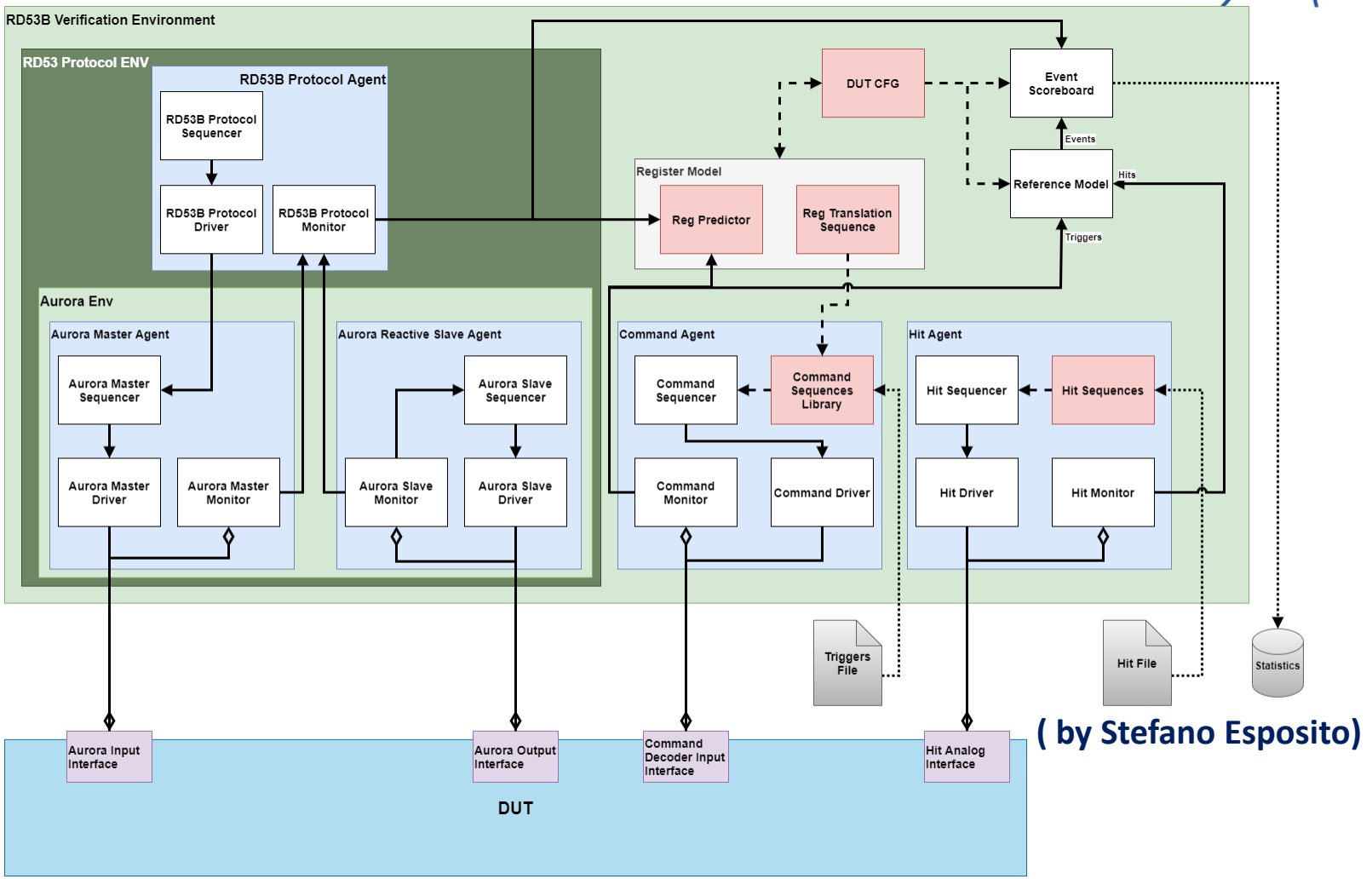
Use of UVM concepts

Command Agent

- Translates the requests into one or multiple commands on a command decoder input interface
- Respects priorities

Hit Agent

- Generate random hits (pixel address and ToT) or from a CSV file
- Several hit distributions supported



(by Stefano Esposito)

Jelena Lalic <jelena.lalic@cern.ch>

RD53 Verification Framework

Aurora UVC

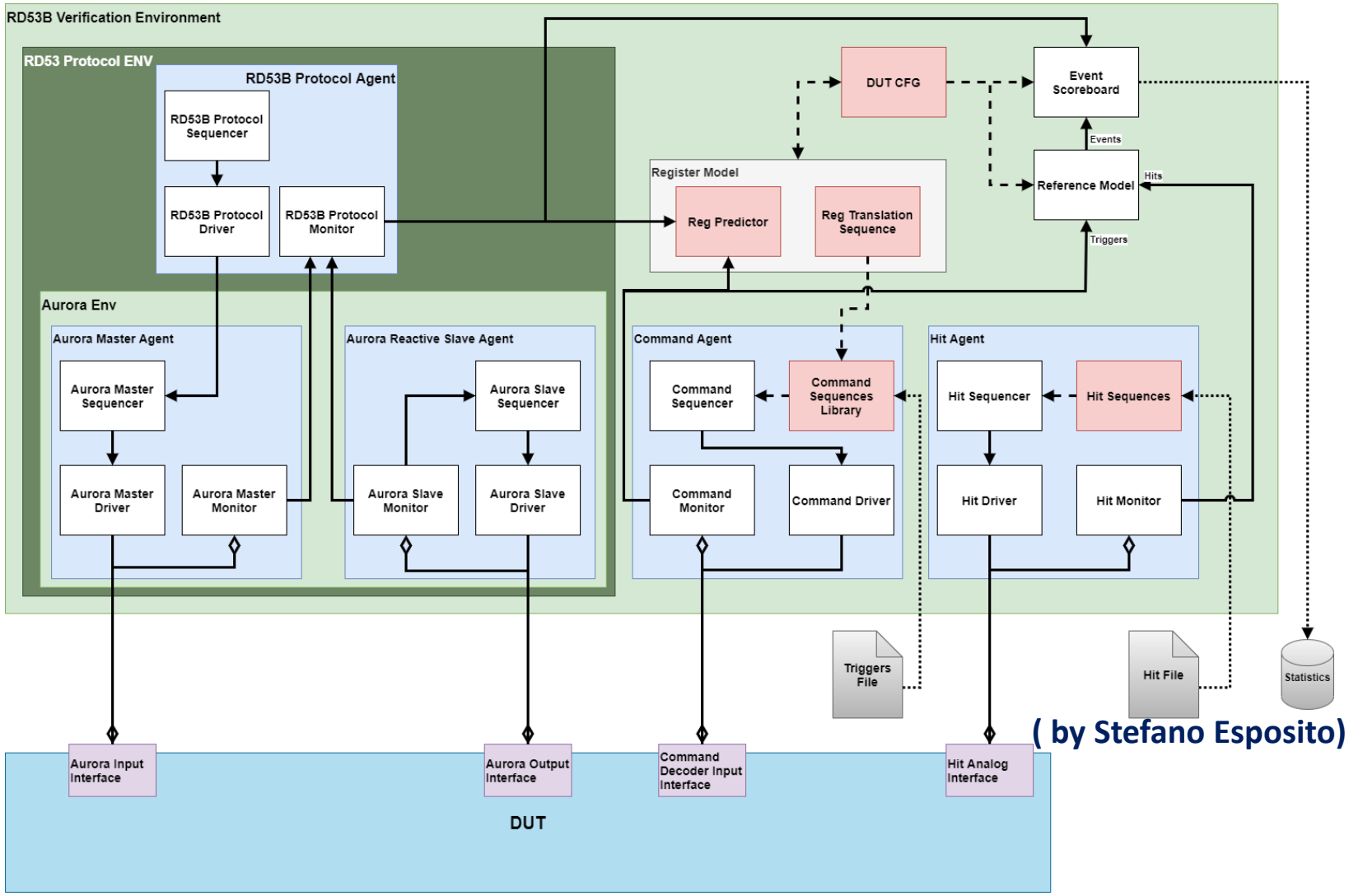
- Slave agent (receives data)
- Master agent (data merging)
- Protocol monitor
- Process service frames and data frames

Reference model

- Predict events
- Receive hits and triggers

Scoreboard

- Receive predictions from the Reference Model
- Receive data readouts from the DUT
- Tracks and reports readout events and hits

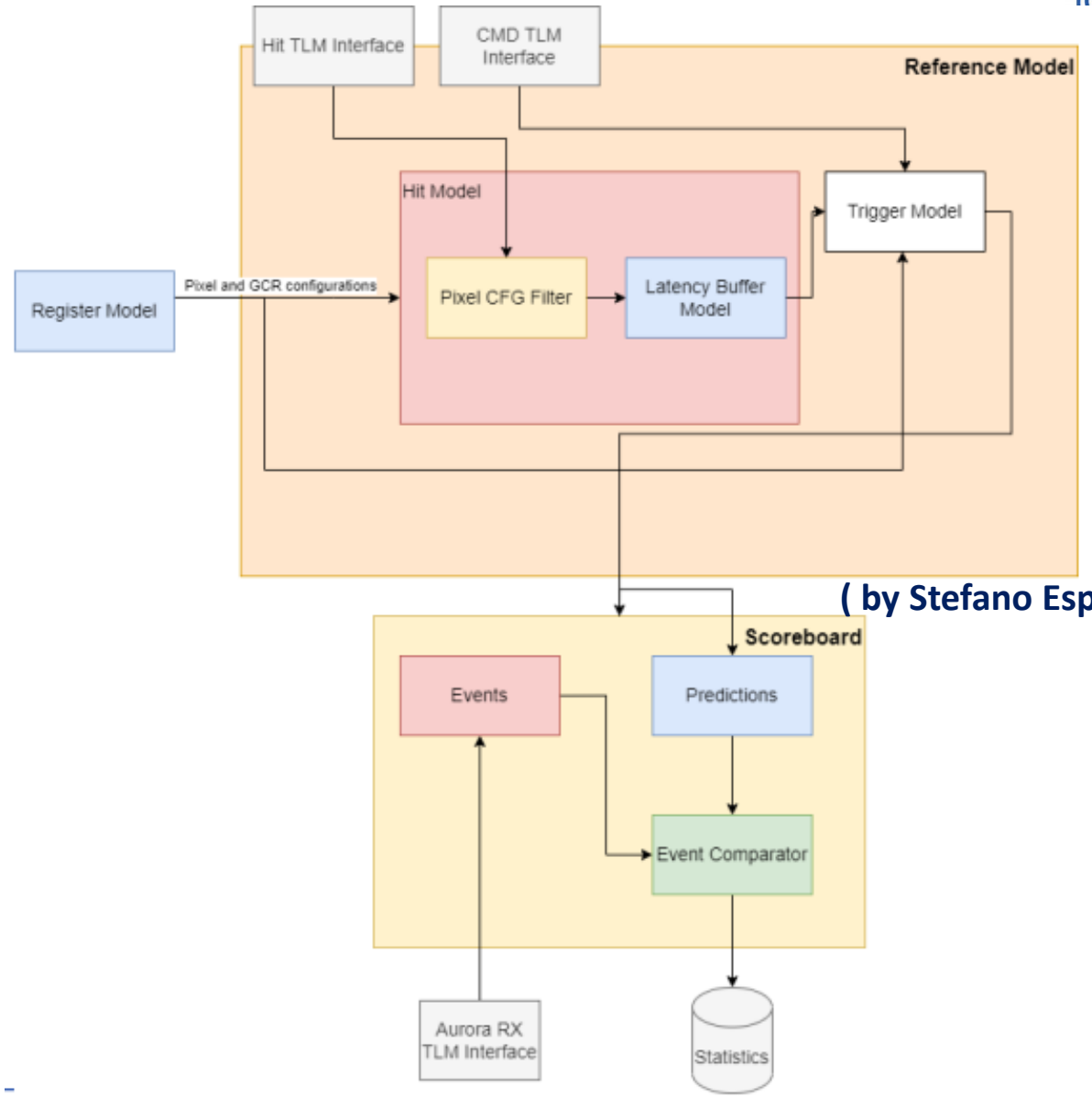


(by Stefano Esposito)

Jelena Lalic <jelena.lalic@cern.ch>

RD53 Reference Model

- **TL model**
  - Receives the same hits as the DUT
  - Receives the same commands as the DUT
    - Trigger, Clear, ReadTrigger, GlobalPulse
  - Filters hits according to pixel configuration
  - Reacts to triggers accordint to GCR values
  - Uses the register model
- **Scoreboard**
  - Receives predictions from ref model
  - Receives events from DUT (through Aurora UVC)



(by Stefano Esposito)

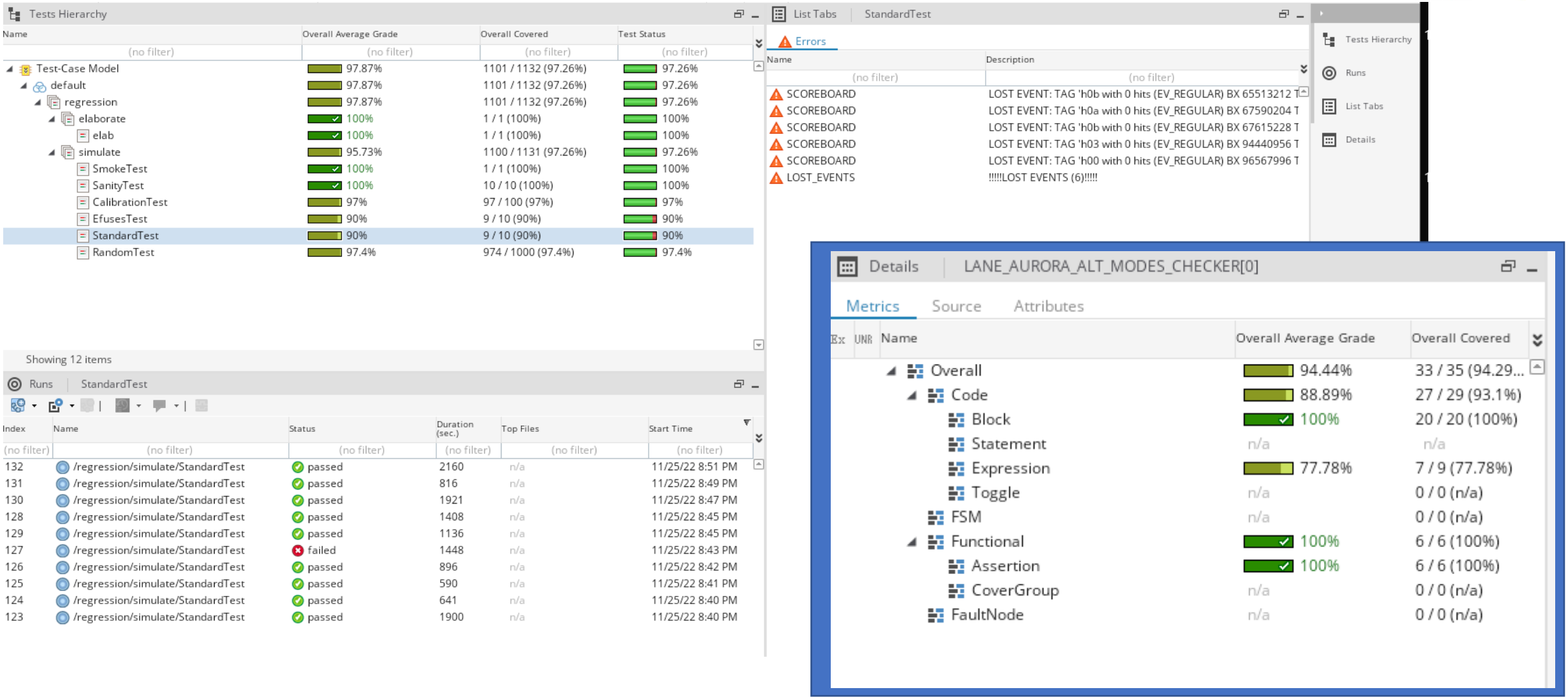
RD53 Tests

- **Smoke and Sanity Test**
  - Used to validate the TB itself rather than the DUT
- **Random Test**
  - Constrained randomization of all sequences
- **Calibration Test**
  - Random Calibration commands
- **StandardCase Test**
  - Default Pixel and Global conf; 3.5GHz/cm2 hit rate and 1kHz trigger rate
  - Random hits and triggers
- **RandomSEETest**
  - Random Test with SEE UVC
- **Directed Tests** (e.g. BlackEventTest)

Tests are grouped to create a regression

Collected metrics are refined

Vmanager Regression Framework Example



RD53 Gate Level Verification

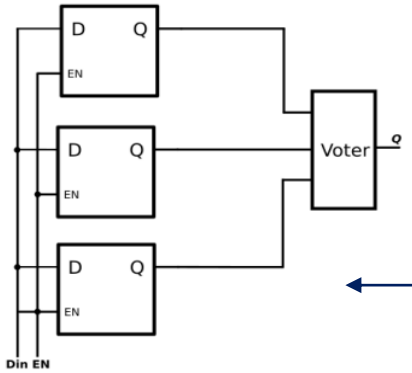
Complementary to RTL verification

- Find timing violations escaped to STA
- Verify asynchronous circuits with real delays

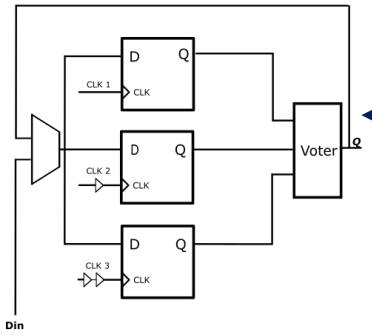
Run with minimal set of seeds extracted from RTL regressions

- Requires RTL regressions to be complete
- Defining a minimal set of random seeds that will fully exercise the DUT
- Requires lots of computational resources and time

SEE mitigation in the RD53

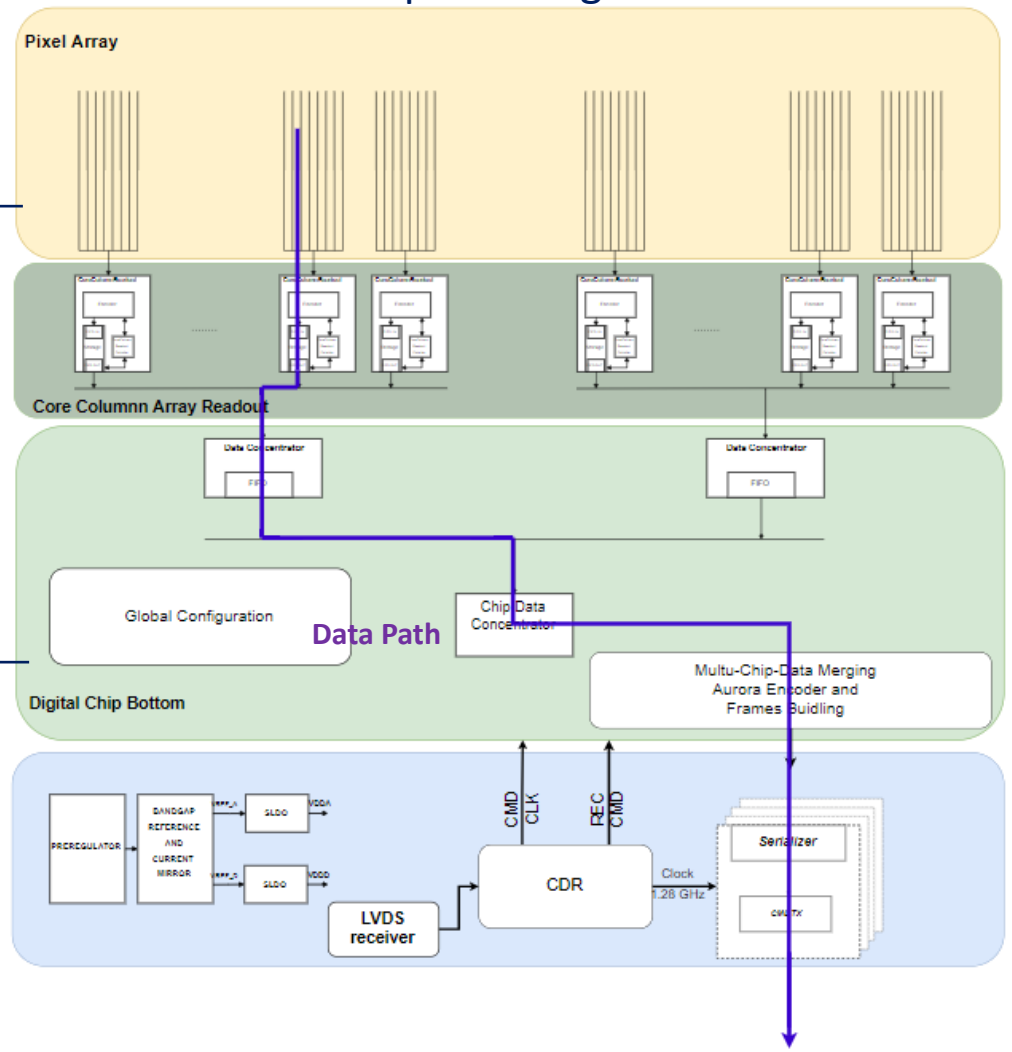


- Each pixel has a configuration register
- Critical configuration bits are protected
- SEU here has a limited effect
- Continuous reconfiguration by the DAQ is needed



- Global configuration – critical for the optimal chip functionality
- SEU and SET protection (triplicated clock, time skew)

How do we protect against SEEs?



Very complex chip:

~500 millions transistors:
ATLAS: 153 600 pixels
SEU rate in the inner layer: ~100 Hz/chip
Can we protect everything?

- No

Many pixels, huge storage, complex data path –Protecting everything would cause huge power and area overhead.

What is done:

- - Only critical information in the data path is protected (state machines, buffer pointes, critical event info.)
- Pixel configuration and global configuration registers have TMR protection
- Critical blocks in the ACB are optimized for SETs (PLL, LVDS receiver, CML driver, biasing blocks)

What is the target behaviour:

- **Occasional hit/event is allowed to be lost**
- **Need to recover without power cycling**

RD53 SEE Verification Strategy

FAULT CAMPAIGN 1:

Goal:

Verify that SEUs in un-triplicated logic will not require a clear or PLL reset or power cycling. Hit losses and event losses are expected and tolerated

Fault injection scope:

SEU in non-triplicated flops

SEU @ RTL

FAULT CAMPAIGN 2:

Goal:

Verify that all logic which is intended to be triplicated is correctly triplicated

Fault injection scope:

SEU in triplicated flops

SEU @ Gate Level

FAULT CAMPAIGN 3:

Goal:

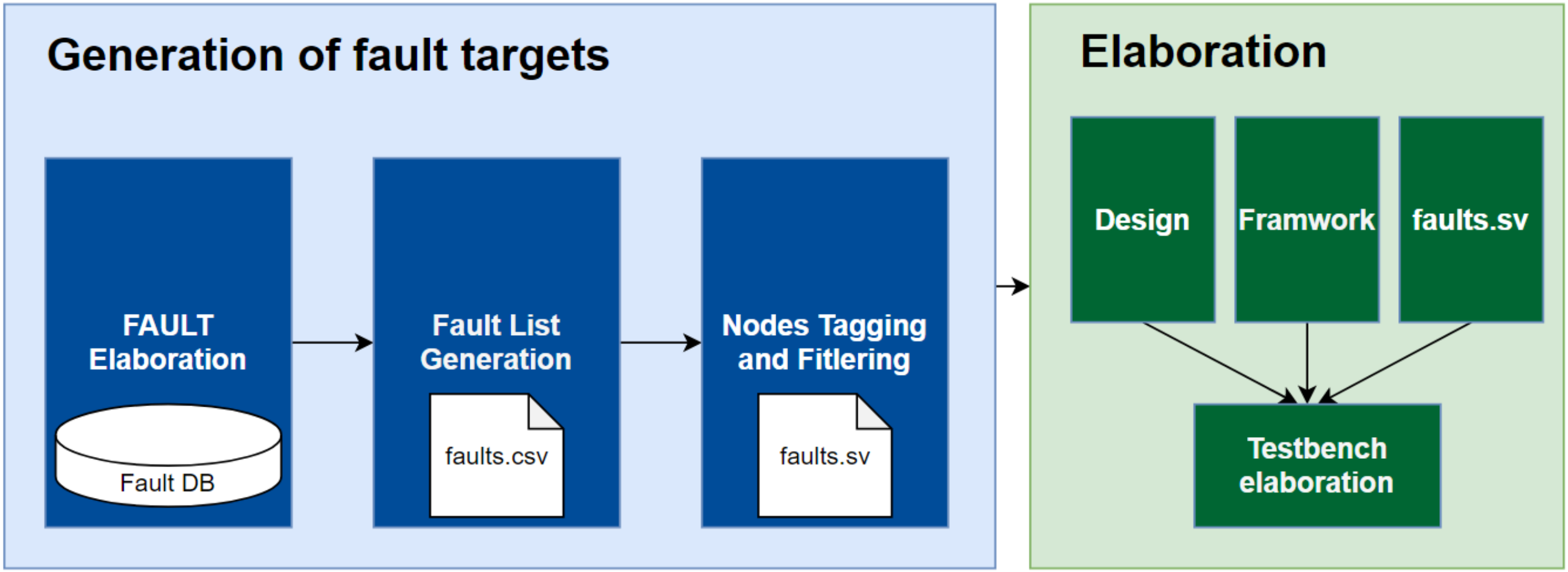
Verify that the chip self-recovers from the long SETs and that no critical behavior is observed

Fault injection scope:

SET on the clock network

SET @ Gate Level

Faultable Target Generation



SEE UVC

SEE sequence

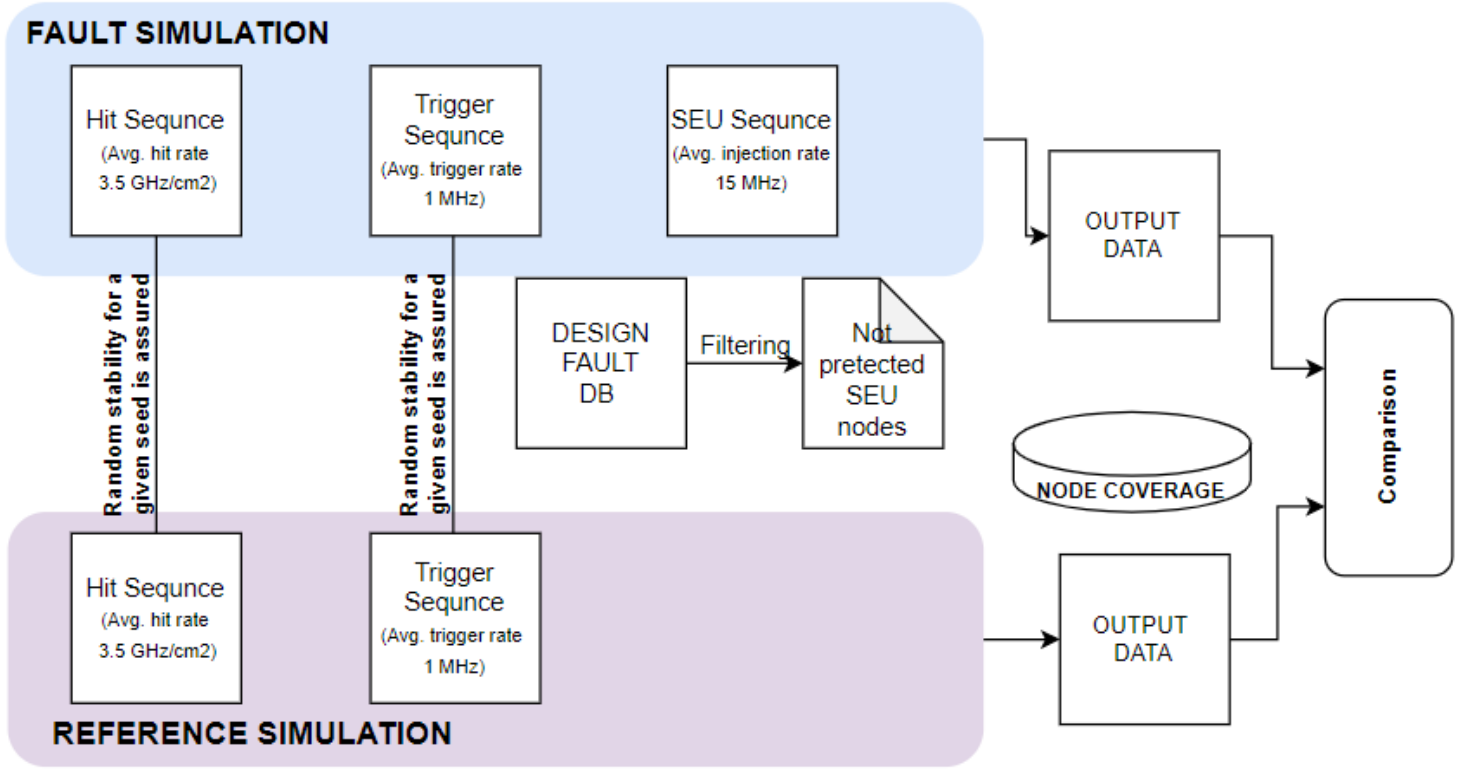
- Controls fault injections
- Randomly selects a node and the time for the SEE injection
- Modelling a SEU
  - value of the sequential logic flipped using the *uvm_hdl_deposit*
- Modelling a SET
  - Flips the value of the net using the *uvm_hdl_force*
  - Waits for a given (randomized) amount of time
  - Releases the net by using the *uvm_hdl_release*
- Can do multiple simultaneous injections if specified by the test

Timeout Monitors:

- Monitors service and data frames
- Gives an error if service or data frames are not received after a defined timeout

SEU verification @ RTL

SUE injection rate ~ 3 10^6 times higher than the expected SEU rate in the inner layer.

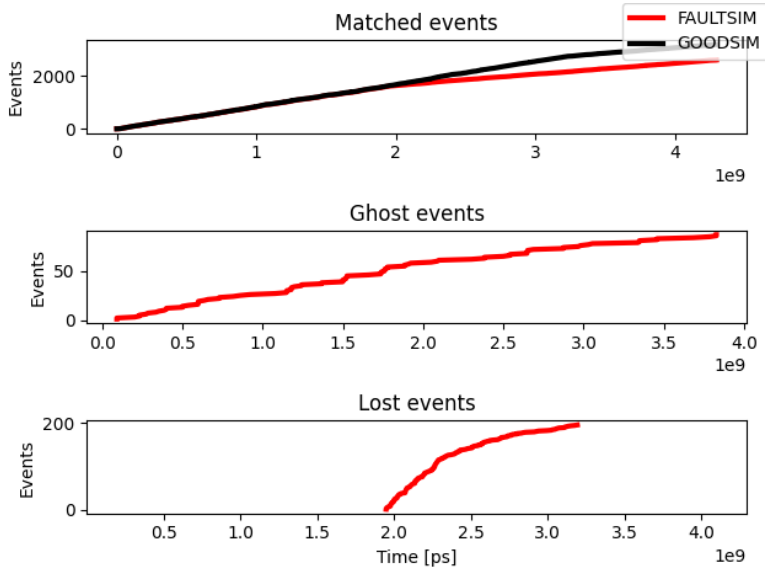


FAULTSIM:

- ❑ Full chip simulation with a high hit and trigger rate (specified by the design requirements) and with an extreme SEU injection rate (to increase the SEU coverage)
- ❑ SEU faults are injected in not triplicated faultable nodes

GOODSIM:

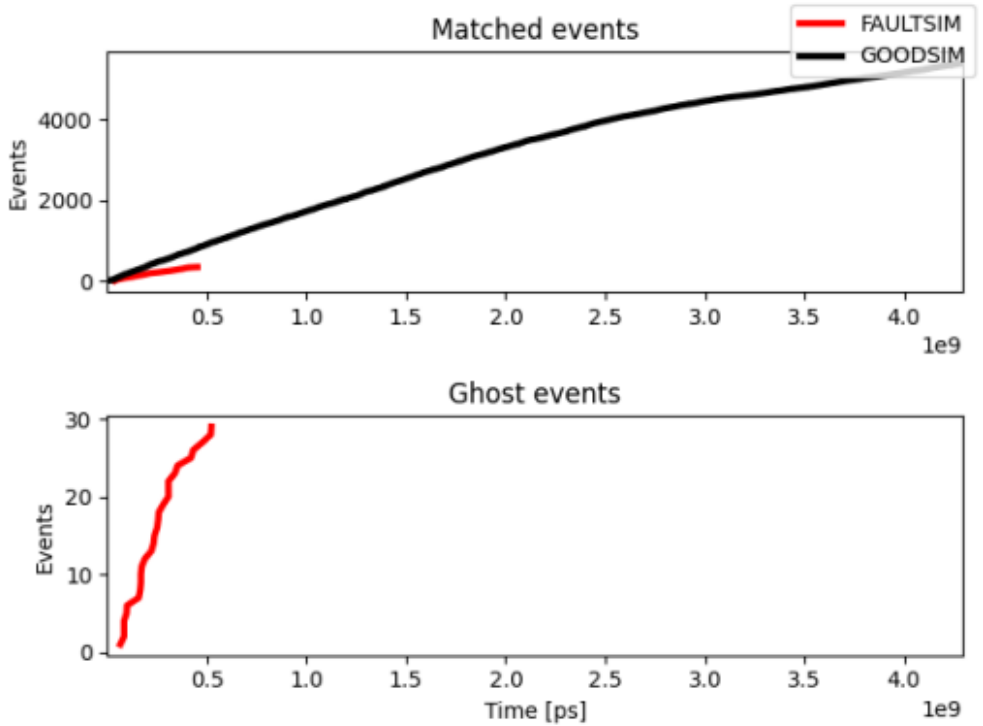
Faults are not injected. Same hit and trigger sequence as in the FAULTSIM for a given SEED

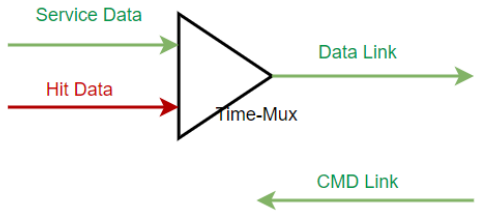


Stuck hit readout in the simulations

Matched events: Readout events with the correct tag (trigger ID)

Ghost events: Readout events with the unexpected tag





2 independent output channels: hit and service data (time-multiplexed onto the serial output)

Service data: Configuration register readback

Automated checking by the SEE UVC Monitor for Data and Service frames

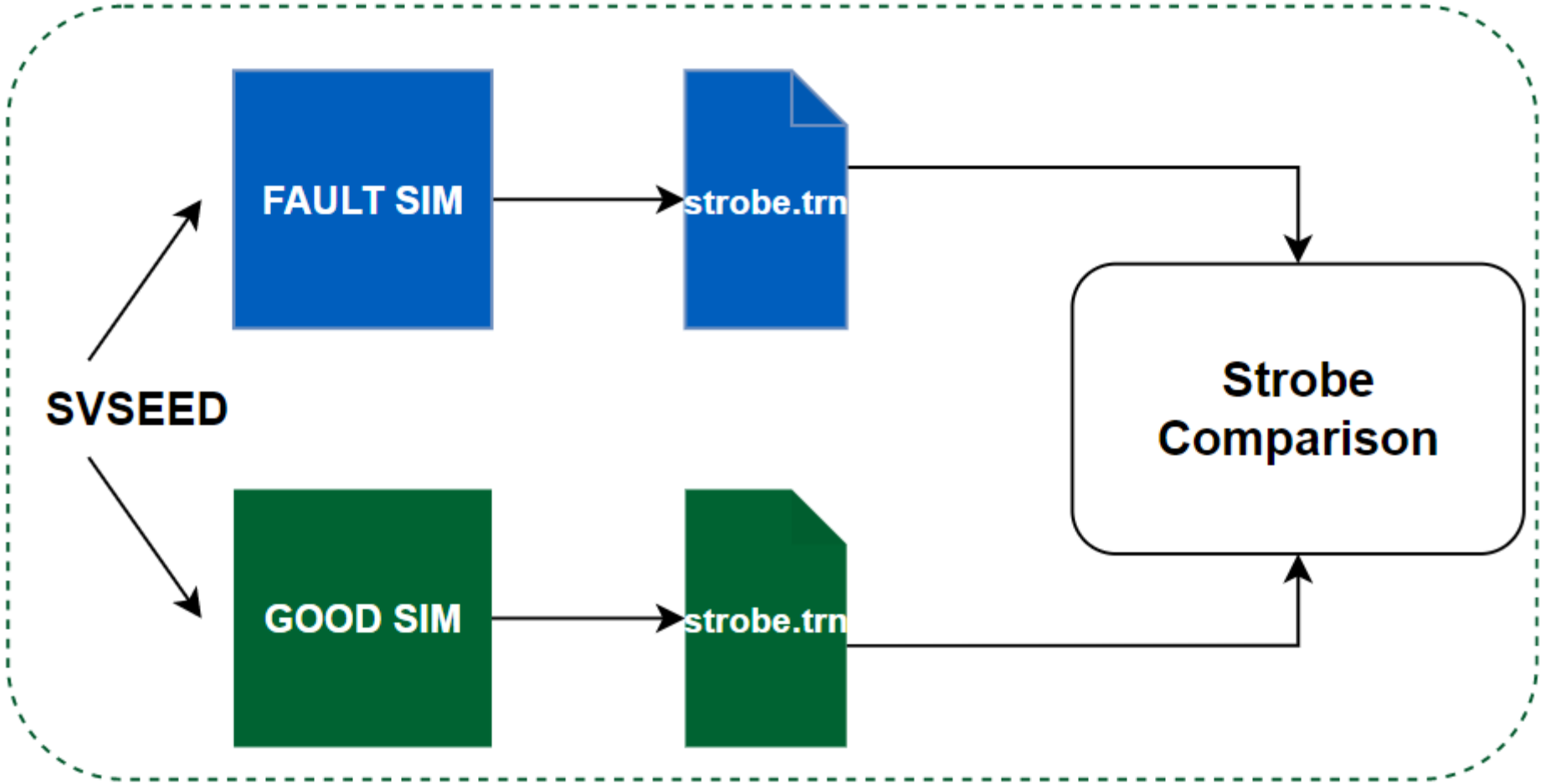
No problem with the service frames in the RD53

- chip always responds to command

A few issues that can cause hit data redout are identified:

- Interaction between Pixel array and readout state machines
  - Only one logical AND could cause a stuck hit readout with a cross-section of 1 Hz in the HL-LHC innermost layer: *Illustrates the importance of the SEU regressions over each design change since only a small code change can later cause disasters in the radiation environment*
- Identified handshaking signal that was unprotected in the RD53B chip: *Illustrates the importance of the good SEU coverage (> 100 injections per node)*

SEE @ GateLevel



For SEU and SET simulations on netlist

Summary

- UVM covers the fundamental methodology for building reusable verification environments.
- UVM enables verification IPs to be shared and reused between projects.
- We covered basics guidelines for developing a UVM environment.
- The UVM concepts are used in development of the Verification framework for the RD53C.
- The SEE verification component and the SEE verification strategies used in the verification of the RD53C are shown.