2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION

UNITED STATES

SAN JOSE, CA, USA
MARCH 4-7, 2024

# Leveraging Model Based Verification for Automotive SoC Development


Aswini Kumar Tata, Sanjay Chatterjee, Kamel Belhous – Allegro MicroSystems.

Surekha Kollepara - Cyient

Bhanu Singh and Eric Cigan - MathWorks

# Agenda

- Introduction
- Problem statement
- Overview - Model-Based Design and Verification
  - Traditional verification workflow
  - Model-Based Verification
- Simulink testbench model example
- UVM bench integration
- Bugs caught using MBV flow
- Enhancement requests in MBV flow
- Conclusions

# Introduction

- Allegro Microsystems develops advanced mixed-signal sensors – primarily for automotive industry – that interface with mechanical systems that:
  - Sense
  - Regulate
  - Drive
- Customer requirements are becoming more complex with reduced time to market.
- We perform architecture and algorithm development for DSP blocks at a higher abstraction layer for better definition and implementation - Model-Based Design (MBD) with Simulink® from MathWorks.

# Problem Statement (for Verification)

- **Waiting** to find RTL bugs in UVM environment from Model-Based Design - too late and costly.
- **Waste** of critical verification time and digital simulator licenses by debugging unrealistic scenarios with MBD-generated HDL.
- **Too late** to verify changes in customer requirements AND different requirements from multiple customers on a chip during project cycle .
- **Verification inefficiency** if not reusing the Model-Based Verification (MBV) effort in UVM environment for RTL verification through complex constrained randomization and functional coverage.

# Overview of Model-Based Design and Verification

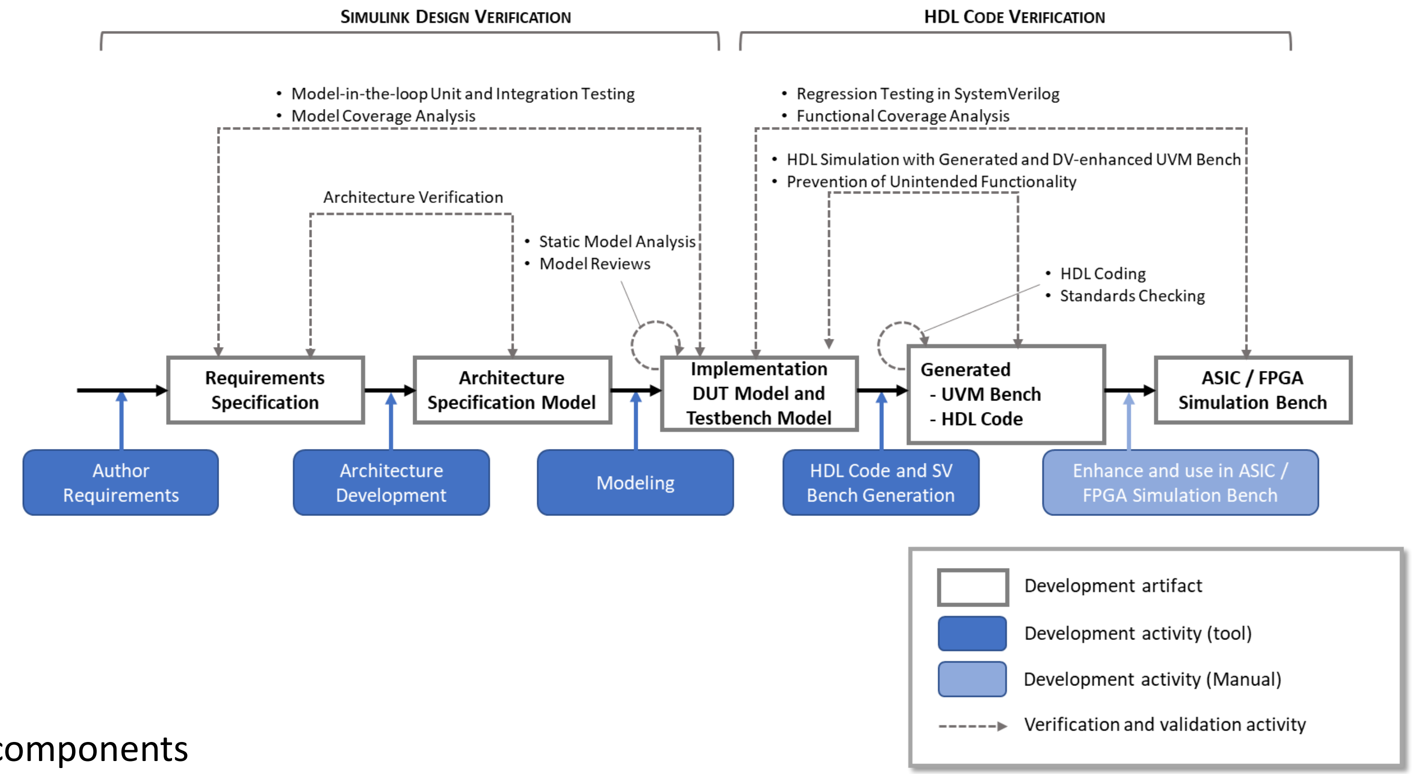
## Requirement Specification
- Define requirements
- Annotate models with requirements (when available)

## Architecture Model
- Behavioral model simulated to fine-tune algorithms

## Implementation and Testbench Model
- DUT model that supports HDL code generation
- Simulink testbench model for DUT

## Simulink Design Verification
- Verifying model against requirements
- Regression run and Simulink model coverage

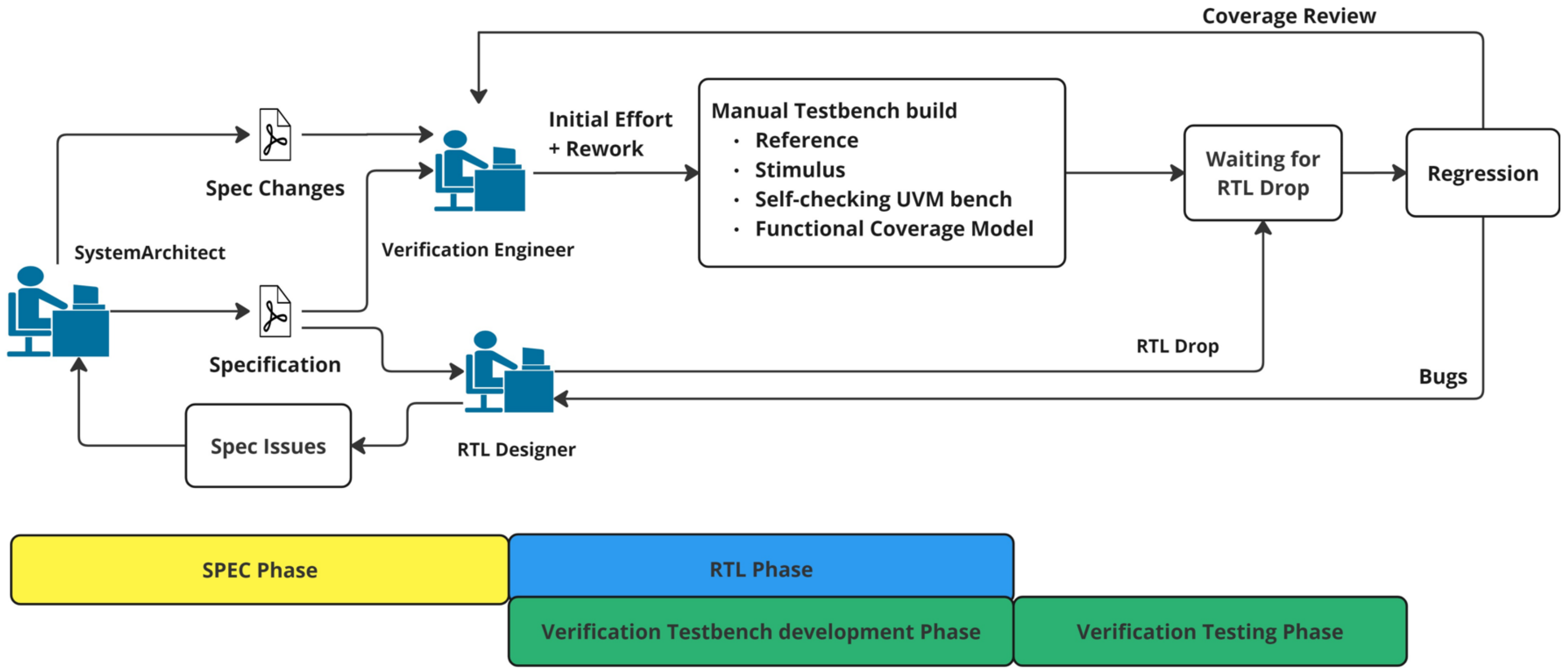
## Code Generation
- HDL code generation
- RTL verification : Reuse Simulink/MATLAB testbench components through:
  - SystemVerilog (SV) DPI-C model generation
  - UVM bench generation



# Traditional Verification Workflow

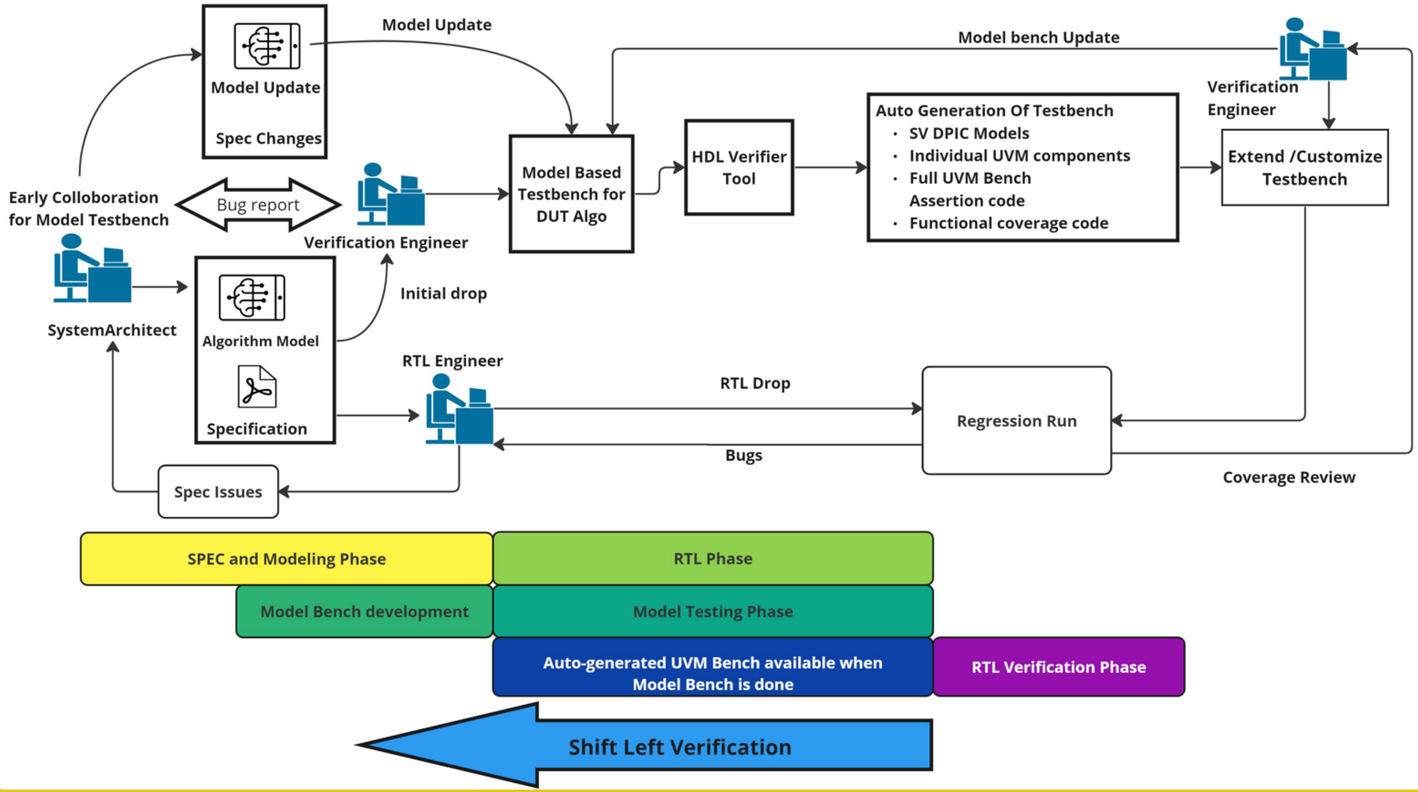
**Requirement phase is Document-based Workflow**

- Requirements as PDF/Word doc are passed from Systems team to Design team.
- Verification tests are run when RTL is available.
- Bug reports can result in spec changes due to incomplete specification. This often drives changes to RTL and testbench.



# Model-Based Verification

- Model acts as executable specification. Shift-left Verification by simulating at Model level.
- Leverage model-based testbench environment to generate SystemVerilog testbench components

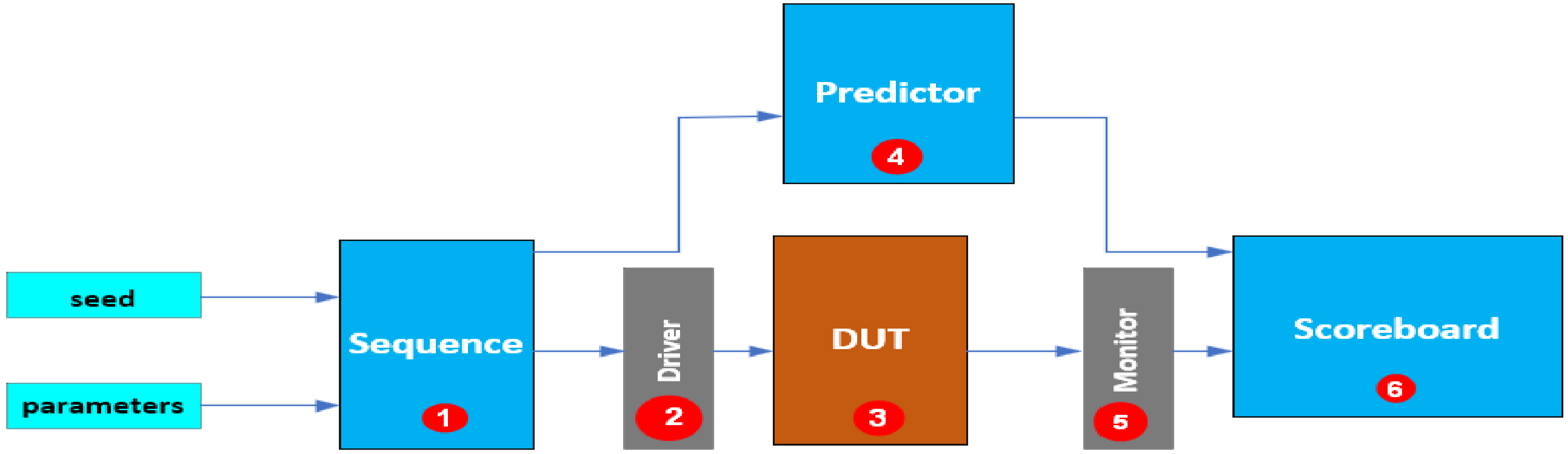


# Benefits of Model-Based Verification (MBV)

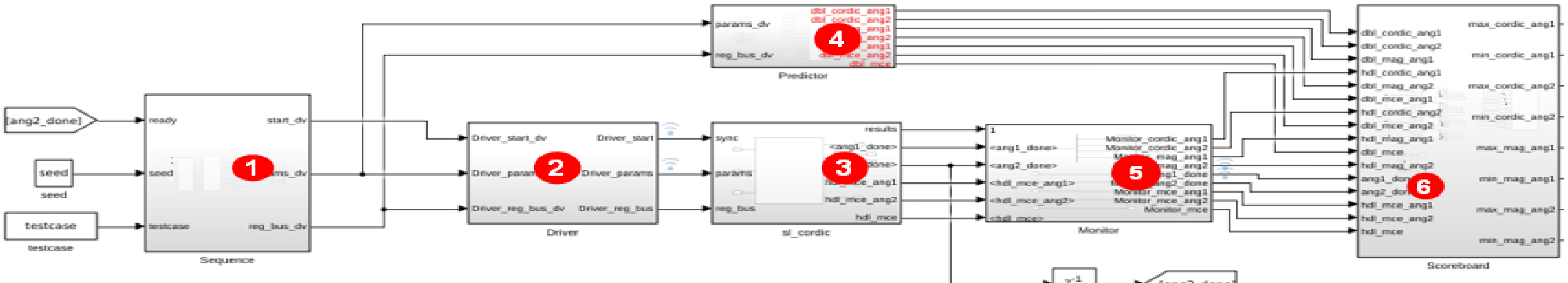
MBV is a great solution for the Shift-Left verification in terms of:

- Being easy to update when requirements change.
- Enabling earlier verification and supporting building design functionality more quickly.
- Authoring and managing regression test-suites.
- Auto generate standalone UVM components from Simulink that can be integrated into the UVM environment.

# Simulink Testbench Model Example




*General Model Structure*

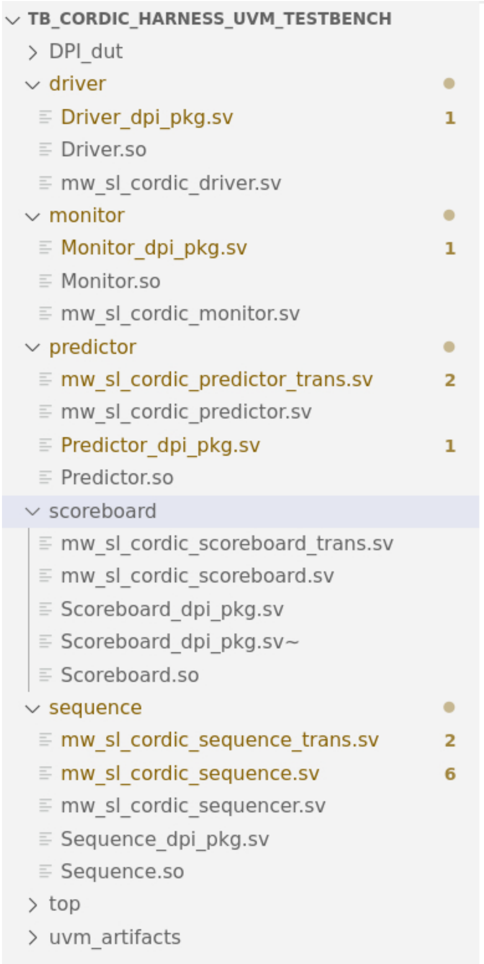



*CORDIC Testbench Model in Simulink*

# Simulink Subsystems and Generated UVM Testbench

There are six subsystems in a Simulink testbench model – each is named to reflect its functionality in a UVM bench. Each subsystem should support C code generation.

- **Sequence**
  - Test Sequence block is used to create different test scenarios consisting of functional test scenarios and randomized test scenarios.
  - There are two inputs namely **seed** which initializes the random number generator and **parameter** to choose the test scenario.
- **Driver**
  - The Driver subsystem handles the conversion of frame-based data to a scalar or floating point to a fixed-point data.
- **DUT**
  - The DUT subsystem is an implementation model of the algorithm.
  - This model has been developed using Simulink blocks and MATLAB code that supports HDL code generation.



# Simulink Subsystems and Generated UVM Testbench

## Predictor

- Predictor subsystem serves the purpose of a reference/DV model.
- MATLAB code developed here is drawn from the specifications document.

## Monitor

- The monitor subsystem converts DUT fixed-point output to floating point for comparison in scoreboard.

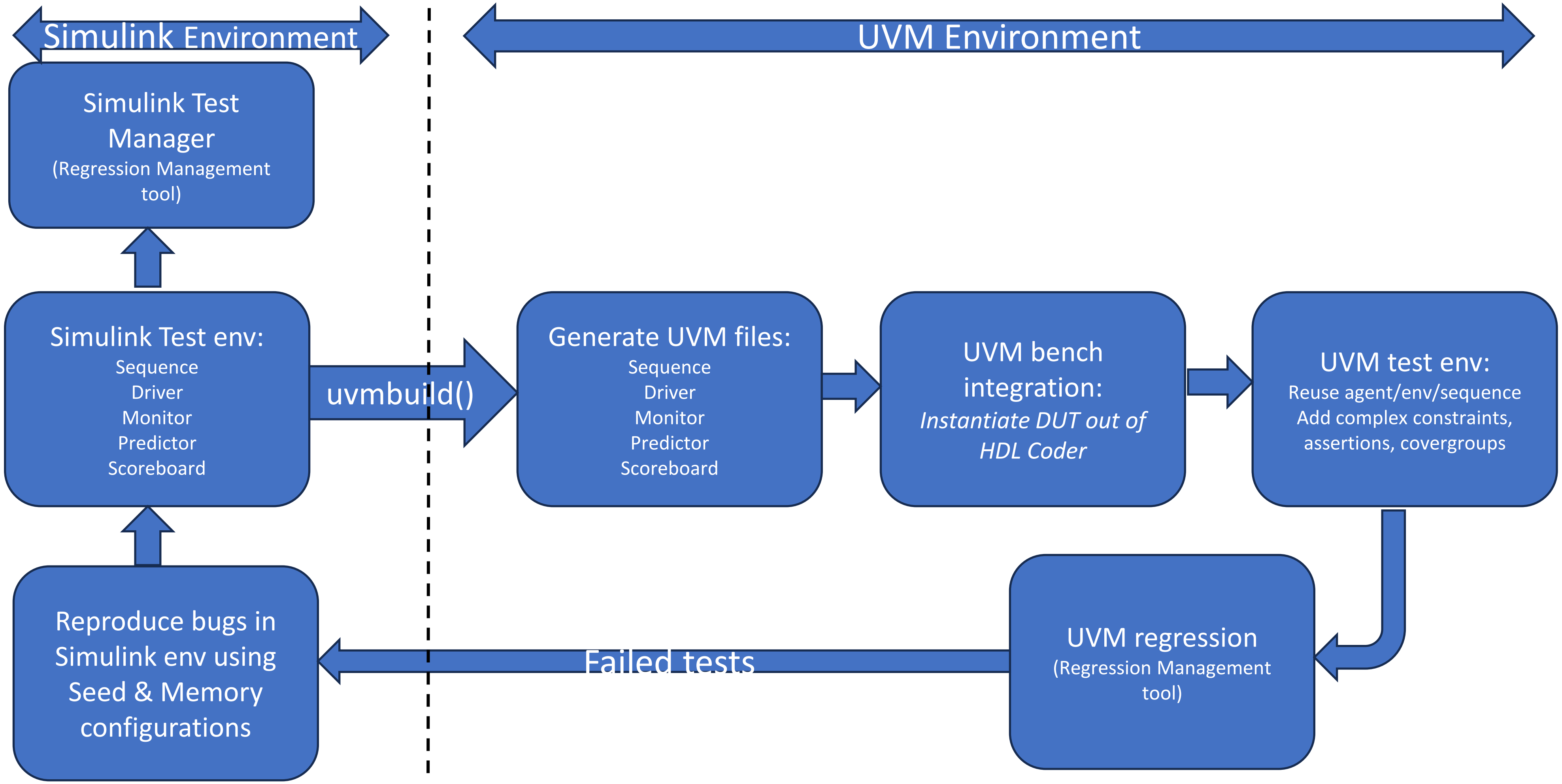
## Scoreboard

- Assertions are modeled in scoreboard using the ‘Assertion for DPI-C’ block.
- Cover-groups are modeled using the verify statement in a *Test Sequence* or *Assessment* block.

```
//File: ./uvm_build/tb_Cordic_harness_uvm_testbench/scoreboard/mw_sl_cordic_scoreboard.sv
//Created: 2023-07-06 15:11:10
//Generated by MATLAB 9.12 and HDL Verifier 6.5

import Scoreboard_dpi_pkg::*;

class mw_sl_cordic_scoreboard extends uvm_scoreboard;
  `uvm_component_utils (mw_sl_cordic_scoreboard)

  chandle               objhandle;


  Scoreboard_dpi_pkg::VerifyInterfaceT vcomp;

  uvm_analysis_export   #(mw_sl_cordic_scoreboard_trans) mw_sl_cordic_agent_imp;
  uvm_analysis_export   #(mw_sl_cordic_scoreboard_trans) mw_sl_cordic_agent_imp_input_pred;
  uvm_tlm_analysis_fifo #(mw_sl_cordic_scoreboard_trans) mw_sl_cordic_agent_fifo;
  uvm_tlm_analysis_fifo #(mw_sl_cordic_scoreboard_trans) mw_sl_cordic_agent_fifo_input_pred;

  mw_sl_cordic_scoreboard_trans mw_sl_cordic_agent_trans;
  mw_sl_cordic_scoreboard_trans mw_sl_cordic_agent_trans_input_pred;

  function new (string name = "mw_sl_cordic_scoreboard", uvm_component parent);
    super.new (name, parent);

  endfunction // new

  virtual function void build_phase (uvm_phase phase);
    super.build_phase (phase);
    mw_sl_cordic_agent_trans = new ("mw_sl_cordic_agent_trans");
    mw_sl_cordic_agent_trans_input_pred = new ("mw_sl_cordic_agent_trans_input_pred");
    mw_sl_cordic_agent_imp     = new ("mw_sl_cordic_agent_imp", this);
    mw_sl_cordic_agent_imp_input_pred    = new ("mw_sl_cordic_agent_imp_input_pred", this);
    mw_sl_cordic_agent_fifo  = new ("mw_sl_cordic_agent_fifo", this);
    mw_sl_cordic_agent_fifo_input_pred  = new ("mw_sl_cordic_agent_fifo_input_pred", this);

  endfunction:build_phase

  virtual function void connect_phase (uvm_phase phase);
    super.connect_phase (phase);
    mw_sl_cordic_agent_imp.connect (mw_sl_cordic_agent_fifo.analysis_export);
    mw_sl_cordic_agent_imp_input_pred.connect (mw_sl_cordic_agent_fifo_input_pred.analysis_export);

  endfunction // connect_phase
```

*Generated UVM Scoreboard Code fragment*

# UVM Bench Integration



# Bugs caught using MBV flow

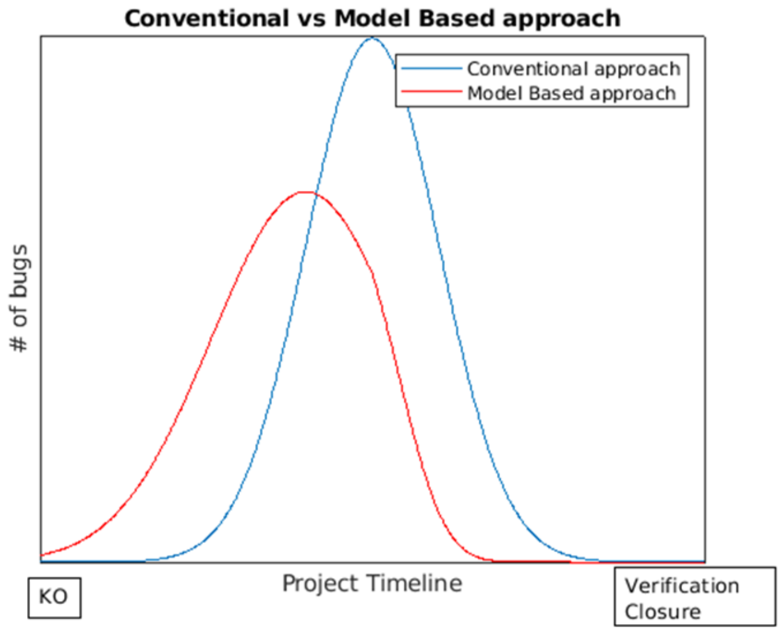
- Incomplete definition of equations and data type inconsistencies were identified in the preliminary specification document.
- Some of the configurations were added in the later stages of the implementation which does not exercise the safety flag checks.
- The conditional statements of a block turned out to be contradicting each other when the block did not result in a valid output for all the input stimulus scenarios.

# Enhancement requests for MBV flow

- Constraints on the input stimulus are limited to the minimum and maximum ranges.
- Input stimulus is streamed based on the feedback/acknowledge signal received from the DUT.
- **uvmbuild()** currently does not support feedback between DUT and Sequence.
- Scoreboard subsystem needs an acknowledge signal from DUT for synchronization.
- Modeling of complex concurrent assertions is currently a challenge.
- MBV flow only supports basic cover groups modeling.

# Conclusions

- Early model verification **is more exhaustive** because verification engineers are best equipped to find out how to break a design.
- Generation of **better-quality RTL** from the models with expected saving of 2 months of verification effort.
- **Reuse of models** with their associated Simulink test environments by **Verification team** for upcoming projects is expected to save 2 months.
- **Reuse of models** by **Systems Engineering** to confirm that implemented designs do what requirements specify.
- **Allegro's customers** could **reuse models** within their own environments to confirm their requirements are met.



*Thank you!*