# Comprehensive IP Verification using Hybrid Way (simulation and formal) : User Experience

Nikhil Tambekar, Verification Specialist
Nokia Solutions and Networks Oy, Finland

## Agenda

- Introduction
- IP verification requirements
- IP verification challenges and risks
- Proposed partitioning
- VC Formal APPs
- Regression management
- Traceability using VPlanner
- Results
- Suggestions for enhancements
- Conclusion

## Introduction, IP Verification Challenges

## Introduction

- Increased Complexity of SOCs
- Verification Challenges
- Configurable IPs
- First pass silicon and Time to market
- Simulation based verification methodology is matured
- Time to explore Formal for exhaustive verification

Combination of Formal Verification and Simulation improves quality

# IP Verification Requirements

Testbench reusability and maintainability

All functional features verified

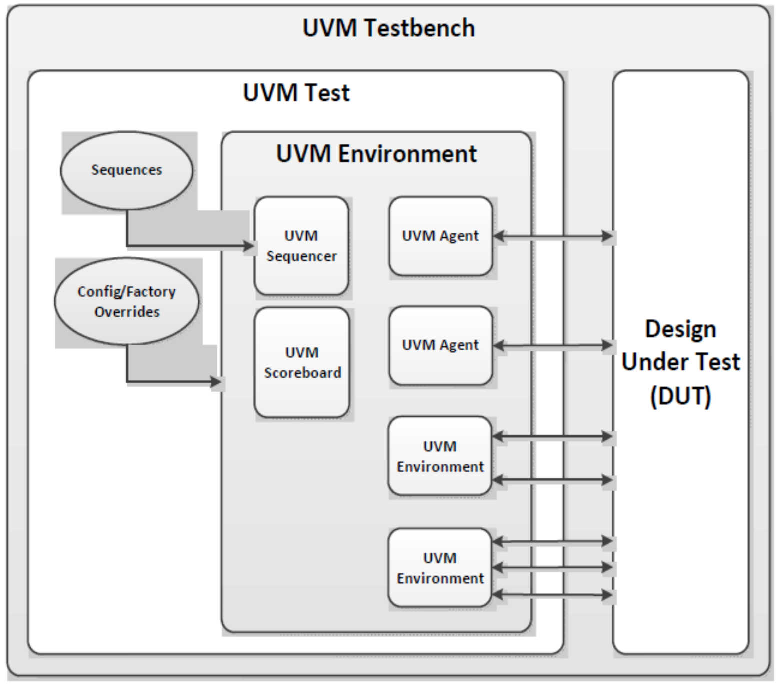
Use case or System Scenario verification (Domain Specific)

Crosses of functionality with configurations

Micro-architecture and interface protocol checking

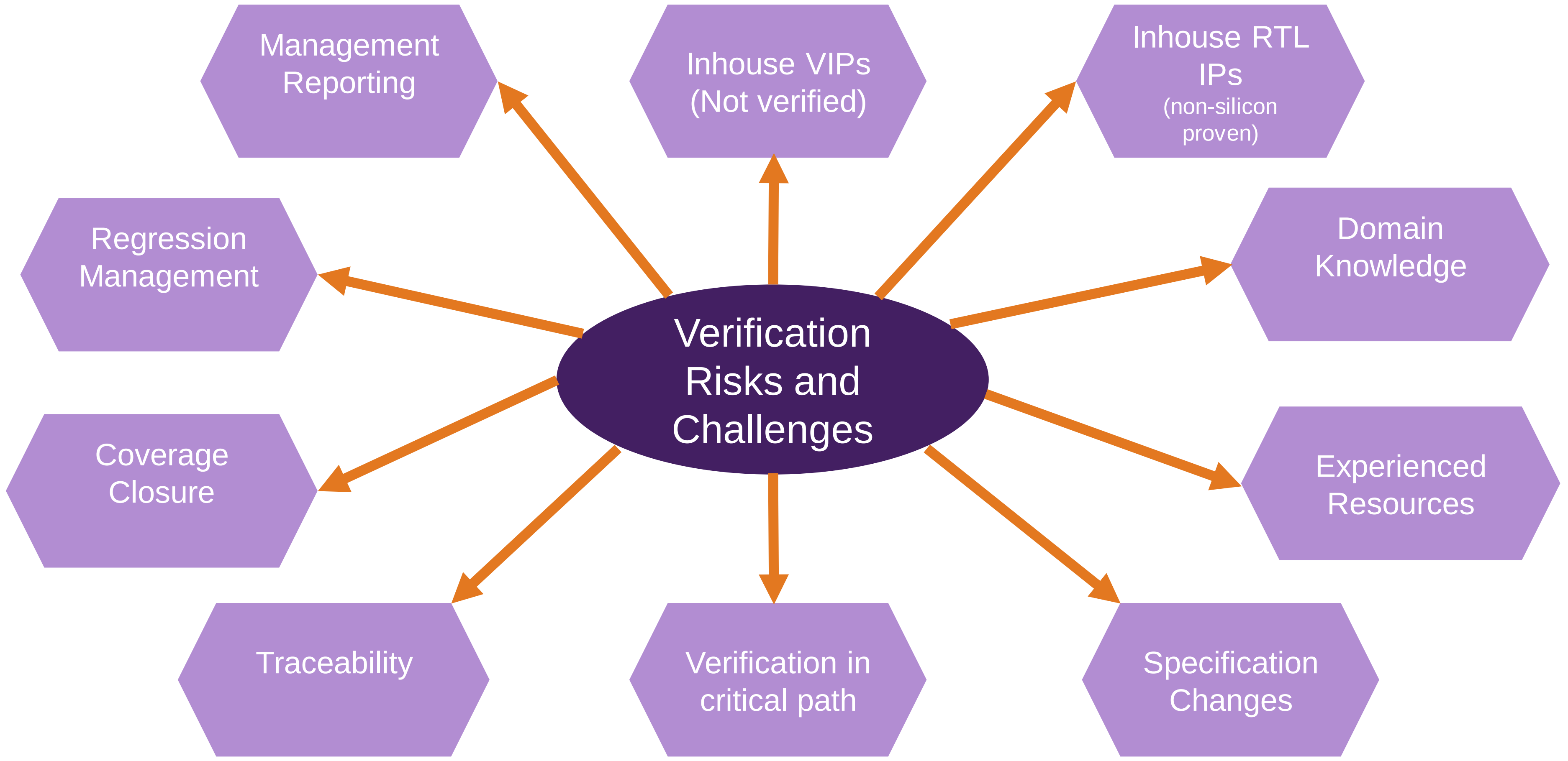
Code and Functional Coverage should be 100% achieved

Traceability between feature requirements to coverage



Reference:Accelera UVM User guide 1.2

# Verification Challenges and Risks



## Proposed Solution

## Proposed Solution

**Divide-and-Conquer**

- Formal Techniques for micro-architecture verification
- Simulations for use-case and system verification

**Regression Management**

- Execution Manager based regression
- YAML config + templates = Regression configurations
- Jenkins CI for automation

**Reporting**

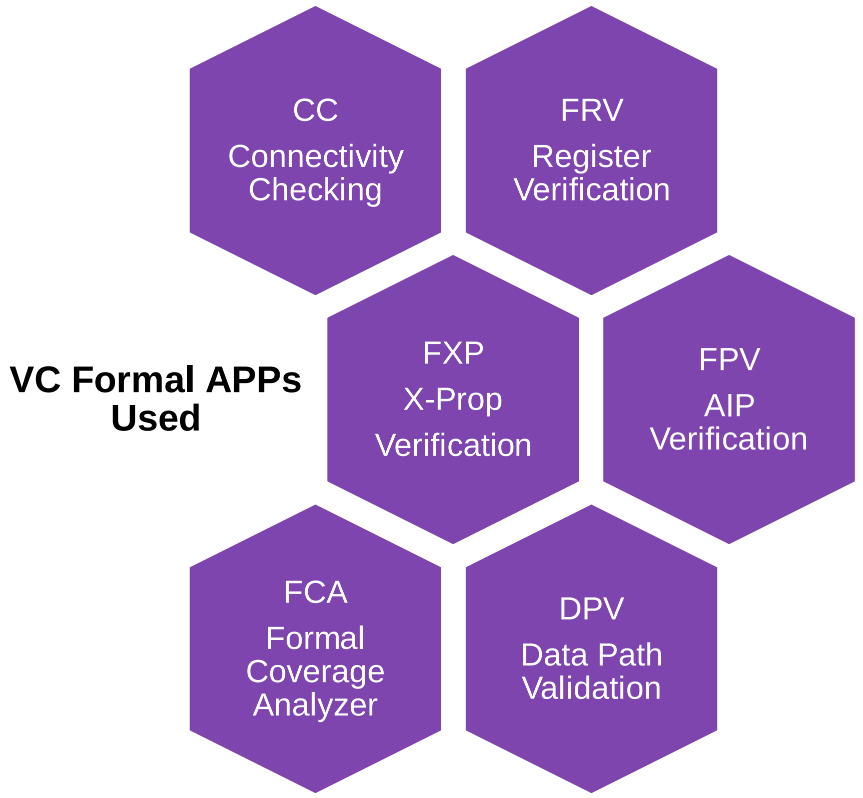
- VPlanner for test plan development
- Requirement to coverage mapping
- Single document for status reporting and tracking

**Coverage Signoff**

- VPlanner and regression result back annotation
- Merging simulation and formal coverages
- Define organization specific coverage metric

# Formal Verification Made Easy

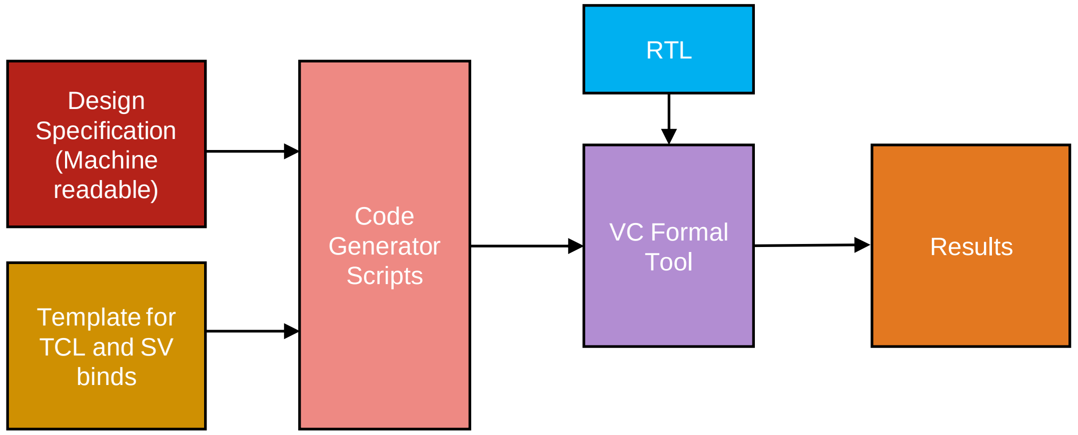
## VC Formal APP Based Verification



- CC
  - Top Level connectivity check
- FRV
  - Register field attributes check
- FXP
  - Catches X Prop issues at RTL level
- FPV-AIP
  - AMBA Assertion IPs (AXI4, AXI stream, AXI4_lite)
- Data Path Validation
  - Mathematical blocks, C/C++ reference models
- Formal Coverage Analyzer
  - Improves Coverage closure with auto exclusions

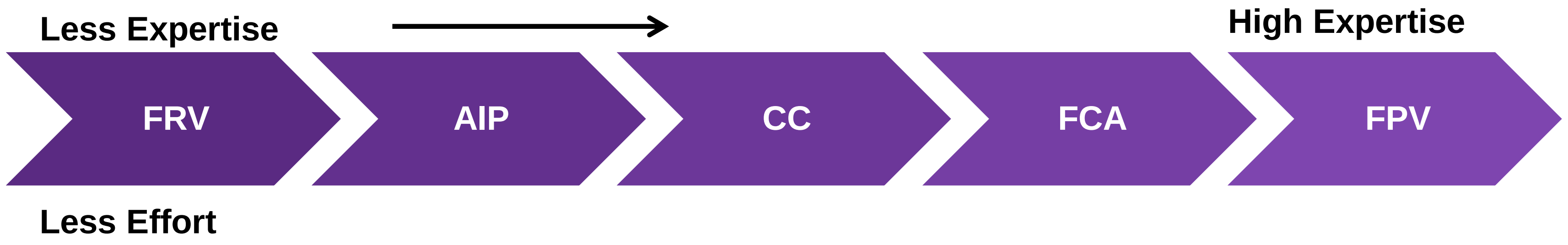
## Benefits of Formal Verification APPs

- Exhaustive verification using formal techniques
- Easy and fast VC Formal APP setup
- Can be used without expert level formal verification knowledge
- No dependency on testbench
- Speed up the setup using automation
- Easy failure analysis using counter example
- Used for small RTL blocks



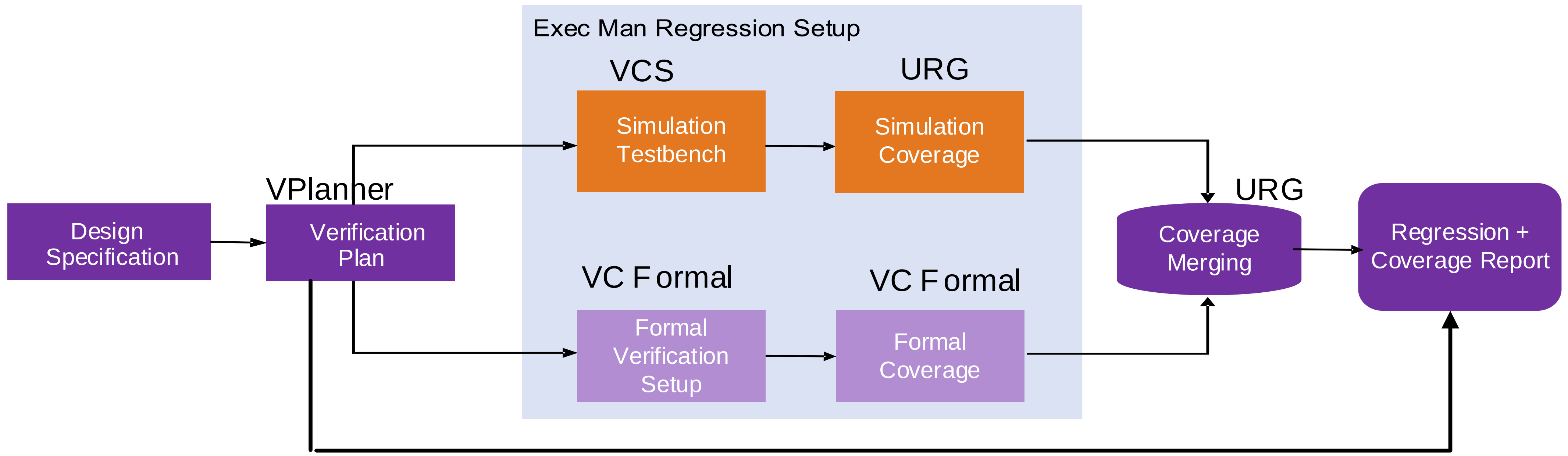
RTL
Design Specification (Machine readable)
Template for TCL and SV binds
Code Generator Scripts
VC Formal Tool
Results

Automation setup for VC Formal Run



## Verification Setup

- Identify features for simulation and formal in the plan
- Reuse compilation: VCS and VC Formal
- Merge simulation and formal Coverage
- V Planner for back-annotation of results
- Unified Coverage report for analysis



## Regression management and Traceability

# Regression Setup Automation

- Automation of regression setup using Jenkins
- YAML based regression modes
- Exec man does Coverage merging, HVP annotation
- Enabled Simulation and formal builds
- Sequential VC Formal run using Jenkins Pipeline

Run Modes:

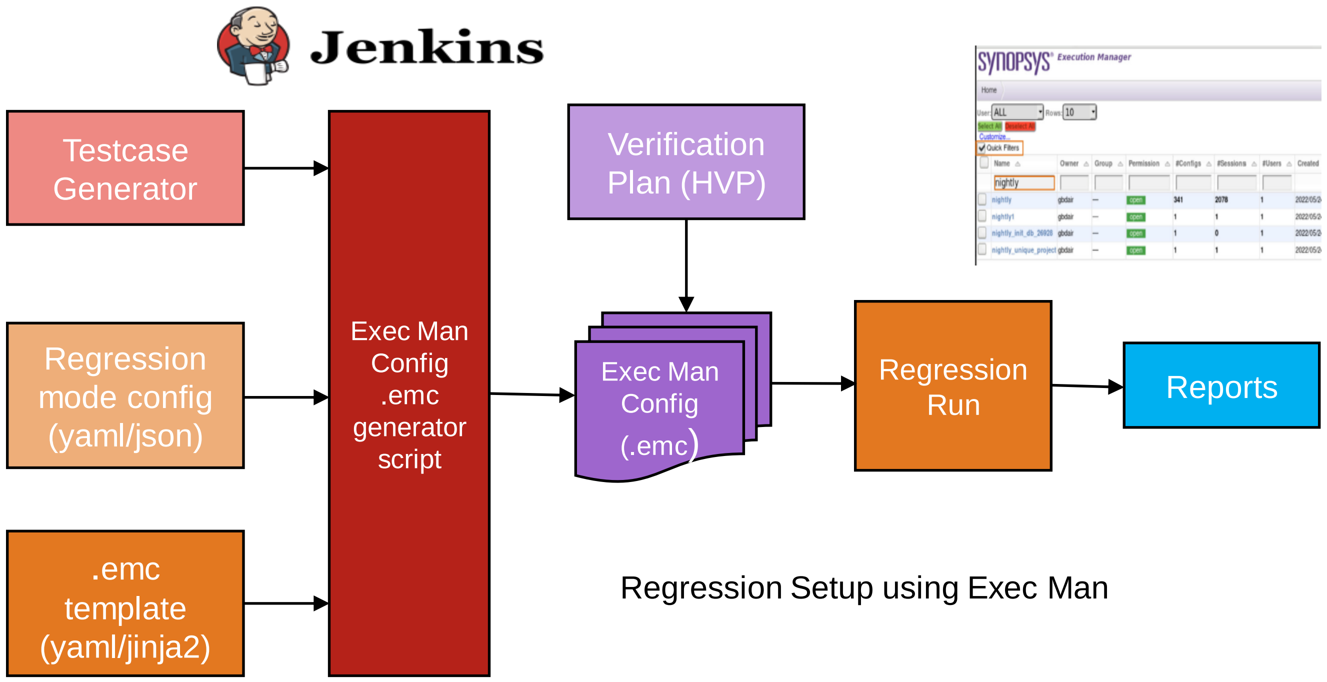
- Full (3000+)
- Nightly(200+)
- Sanity(15)
- Formal Only
- Technology sim
- Power FSDB sim

**YAML Config file for regression modes**

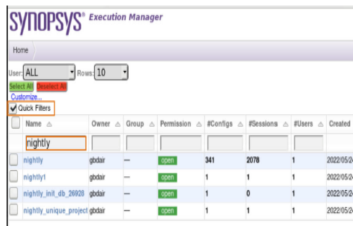
```
regr_modeA:
- name: [test1, test2, test3, test4]
- build_name: build_full_rtl
- run_opts: “ENA_MODEA=1”
- group: stress_modeA_regr

regr_modeB:
- name: [test1, test10, test11]
- build_name: build_full_rtl
- run_opts: “ENA_MODEB=1”
- group: stress_modeB_regr
```

**Test list Template**

```
Test_list:
{% for test in tests %}
-test_name: {{test[‘name’]}}
    -build_name: {{test[‘build’]}}
    -run_opts: test.opts
{{test[‘run_opts’]}}
    -priority: 2
    -test_group: {{test[‘group’]}}
{% endfor %}
```



Jenkins
Testcase Generator
Regression mode config (yaml/json)
.emc template (yaml/jinja2)
Exec Man Config .emc generator script
Verification Plan (HVP)
Exec Man Config (.emc)
Regression Run
Reports



User: ALL Rows: 10

[✓] Quick Filters

| Name | Owner | Group | Permission | #Configs | #Sessions | #Users | Created |
|---|---|---|---|---|---|---|---|
| nightly | | | | | | | |
| nightly | gbdair | — | open | 341 | 2078 | 1 | 2022/05/2 |
| nightly1 | gbdair | — | open | 1 | 1 | 1 | 2022/05/2 |
| nightly_init_db_26928 | gbdair | — | open | 1 | 0 | 1 | 2022/05/2 |
| nightly_unique_project | gbdair | — | open | 1 | 1 | 1 | 2022/05/2 |

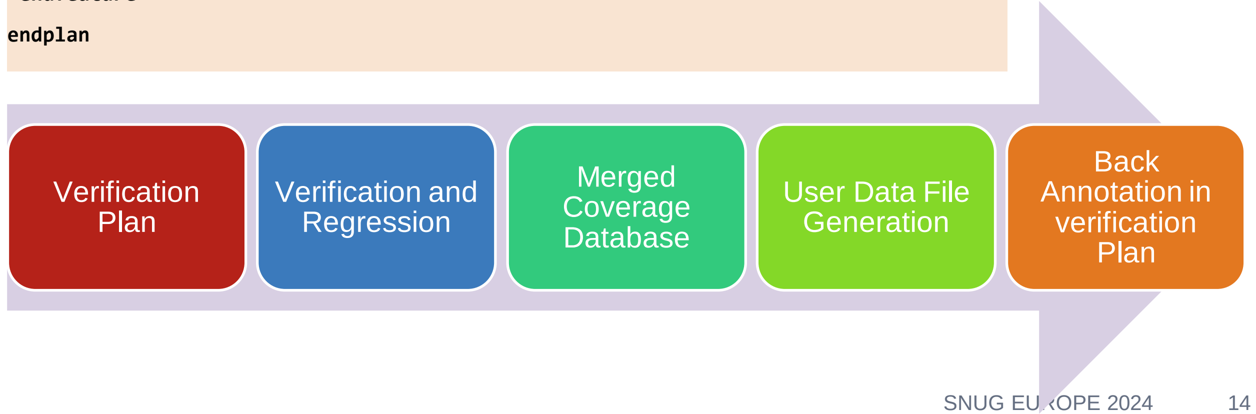
Regression Setup using Exec Man

# Verification Planner for traceability

## Requirement to coverage completion

- Verification Plan with test scenarios
- Requirement mapping -> Features
- Test development status tracking
- Regression result back-annotation
- Code and functional coverage mapping
- Formal property coverage mapping
- Python based coverage mapping
- Project Milestone data for reporting
- Auto HVP generation for AIPs

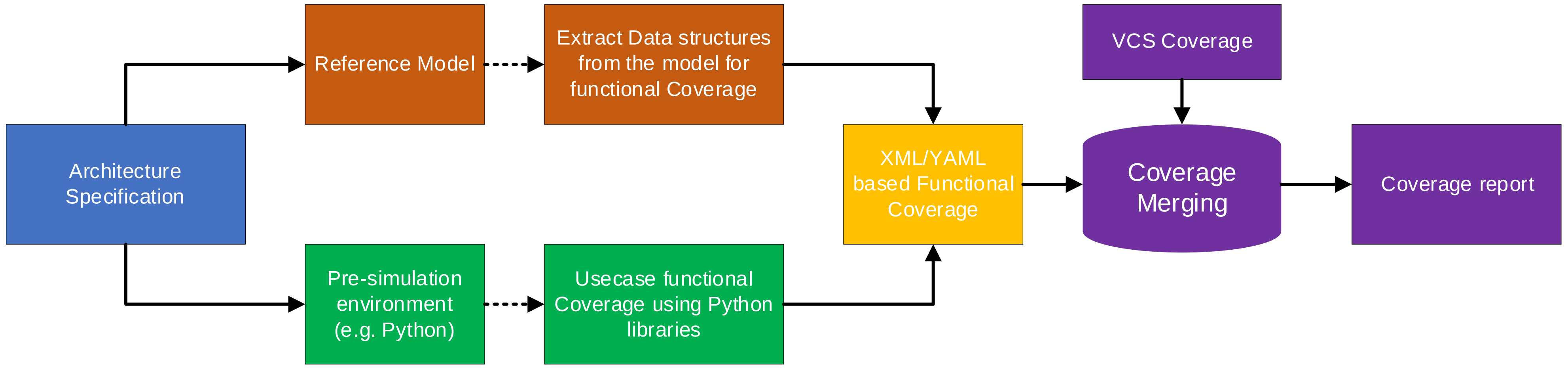
```
plan ip_plan;

   metric enum {proven, falsified, vacuous,inconclusive} formal;
   metric aggregate {Line, Cond, FSM, Toggle, Branch, Assert, Group} all_coverage
   metric aggregate {Assert, Group, test.percent.pass} Functional_cov;
    annotation string Requirement_ID = "";
    annotation string Feature_ID = "";
    annotation string Release_ID = "";
    annotation string Owner = "";
    annotation string TR_tag = "";
    annotation enum{NotStarted,InProgress,Done,Redundant} Test_Status;
    annotation enum {}project_milestone =;
    annotation integer Priority = 0;


 feature Config_register_access;
    description = "Verify reset value of all the Config registers";
    Requirement_tag = "IP_REQUIREMENT_20";
    Feature_ID = "ip_config_2";
    Release = "1.0.0";
    Owner = "Nikhil Tambekar";
    Test_Status = "InProgress";
    TR_tag = "TR1.2";
    measure test, test.percent.pass  uvm;
        source = "ip_tr1_2_hw_reset_test";
    endmeasure
 endfeature

endplan
```



Verification Plan
Verification and Regression
Merged Coverage Database
User Data File Generation
Back Annotation in verification Plan

## Functional Coverage outside SV

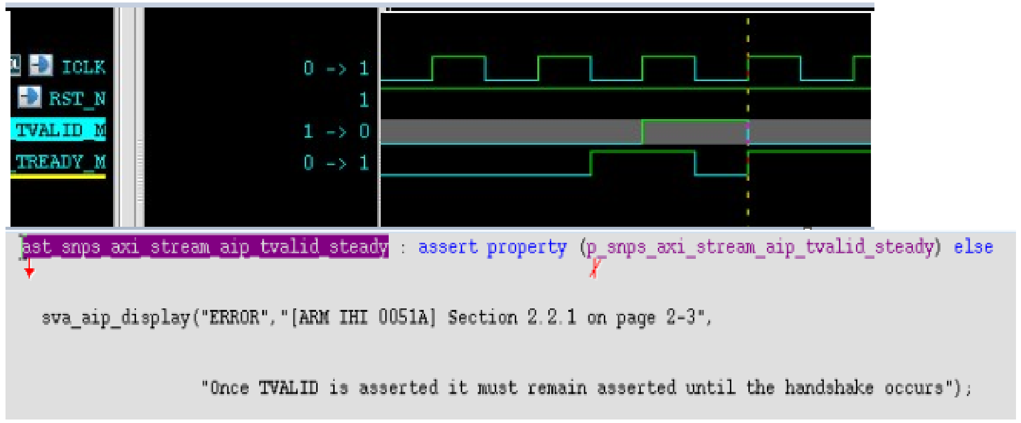
- Use case scenario coverage is important for completeness
- Some features are difficult to extract in IP level testbench
  - E.g. Computation graphs, end to end usecase scenarios etc
- Coverage extraction outside the System Verilog testbench
- Use reference model for coverage data structure generation
- Python based cover groups from test generation environment
- Map the usecase coverage in VPlanner


Reference Model
Extract Data structures from the model for functional Coverage
VCS Coverage
Architecture Specification
XML/YAML based Functional Coverage
Coverage Merging
Coverage report
Pre-simulation environment (e.g. Python)
Usecase functional Coverage using Python libraries

## Results and Conclusion

# Results

- Critical bugs found using formal verification
  - [AIP]  AXI stream protocol bugs (tvalid-tready handshake, access errors)
  - [FPV] Architecture Spec vs RTL Endianness issues
  - [FPV] Architecture Spec vs RTL mismatch in decoding logic
- Formal techniques improve IP quality and early bug finding
- Single document (HVP Plan) for status reporting
- Saved bandwidth used for domain specific Usecase verification and debug
- Faster coverage closure and frame-work scalable for other IPs
- Traceability achieved for verification completion signoff
- Significant Efficiency improvement with automation



```
ast_snps_axi_stream_aip_tvalid_steady : assert property (p_snps_axi_stream_aip_tvalid_steady) else

  sva_aip_display("ERROR","[ARM IHI 0051A] Section 2.2.1 on page 2-3",

                       "Once TVALID is asserted it must remain asserted until the handshake occurs");
```

Handshake bug using AIP AXI stream APP

Verification team achieved all the project milestones on time with high quality deliverables

## Future Improvements

- Encourage third party  coverage database merging
  - More support is needed for UCIS or user-friendly APIs for coverage database manipulation
  - Open-source libraries for verification
- Improvements in Execution Manager tool
  - Support for running only failing tests
  - Improving coverage merging efficiency
- More exhaustive testing for the VCS and VC Formal coverage merging and interoperability
- VC Formal elaboration logs are not self-explanatory, needs AE’s assistance
- YAML/JSON based configuration option for handling tool command line arguments

## Conclusion

- Simulation is necessary for Use cases and System scenarios
- Formal verification for micro-architectural features
- Automation in Regression management
- Single document VPlanner for reporting
- Combination of Simulation and Formal techniques gives best results

Thanks to Jukka Heikkila, Patrick Blestel and Toni Rastio
for Technical Support

# *THANK YOU*

***YOUR***
INNOVATION
***YOUR***
COMMUNITY