A UVM Based Methodology for Processor Verification

Abhineet Bhojak, Stephan Herrmann
Tejbal Prasad

Freescale Semiconductor

accellera SYSTEMS INITIATIVE™

2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Processor Verification Challenges

- Different kind of instructions and excessive number of GPRs leading to massive functional space and we need to target the pertinent
- Presence of asymmetric and out-of-order pipelines
  - Various hazards (e.g. RAW ,WAR,WAW, Branches)
- Dedicated Hardware Accelerators in parallel with pipeline
- Debug hooks for the ease of debug

Stimuli Generator

- *Most important*
- *Need of multiple tests*

Program generation

- *Hazard scenario*
- *Accelerator*
- *Jump & Loop cmd*

Debug

- *Debug Hooks for localization of failure*

© Accellera Systems Initiative

Existing Technologies

- Random test pattern generators (RTPG) and Test plan automation tools
  - Define a specific scenario description language and take as declarative input architecture and micro-architecture
  - Uses sophisticated CSP solver with bias to generate test programs
  - there is a significant learning curve involved to leverage these RTPG's in a project schedule along with a considerable cost factor

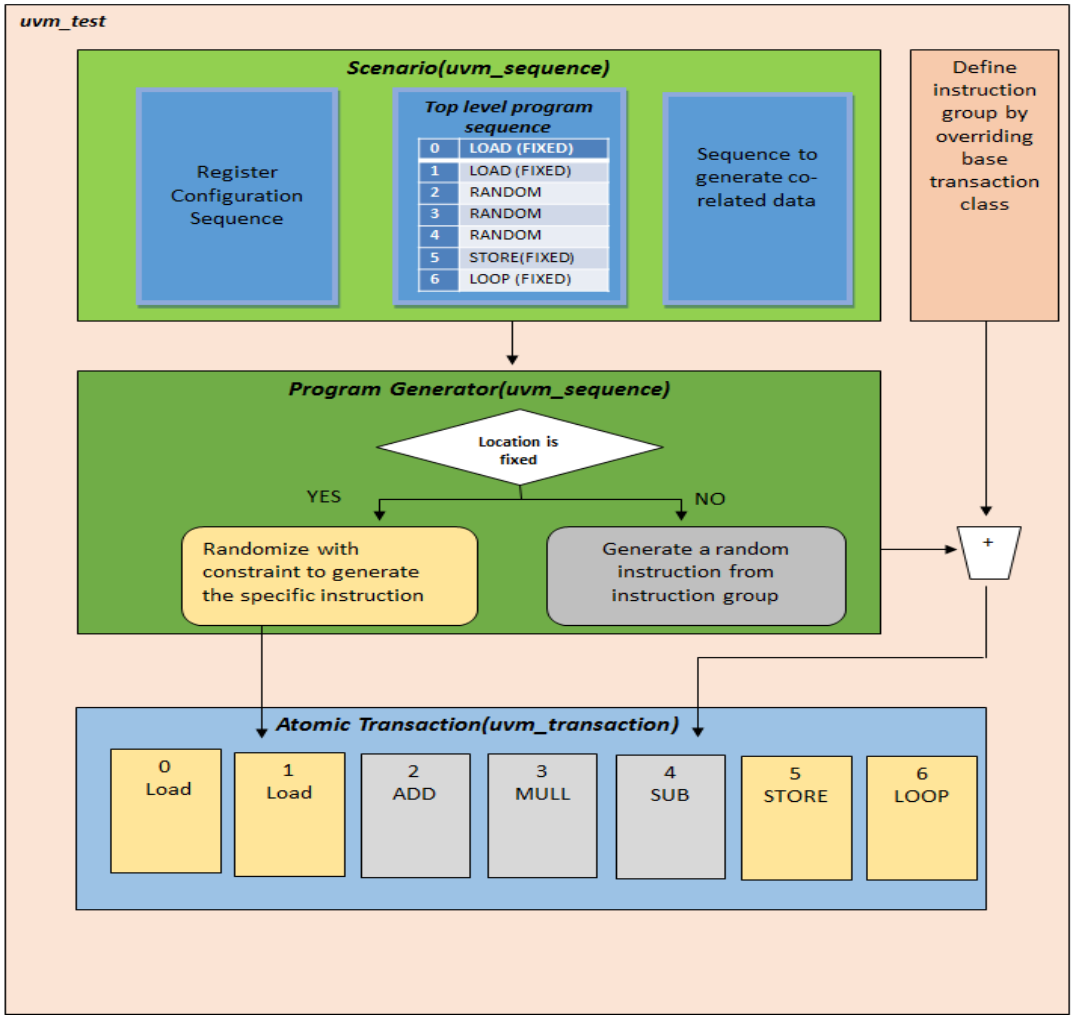
- Formal verification
  - Useful and efficient in some cases
  - it requires significant mathematics skill and computational resources to relate to the scenarios and analyze them

- Pure directed testing
  - Gives confidence on different functionalities
  - Achieving desired coverage may take large amount of time

© Accellera Systems Initiative

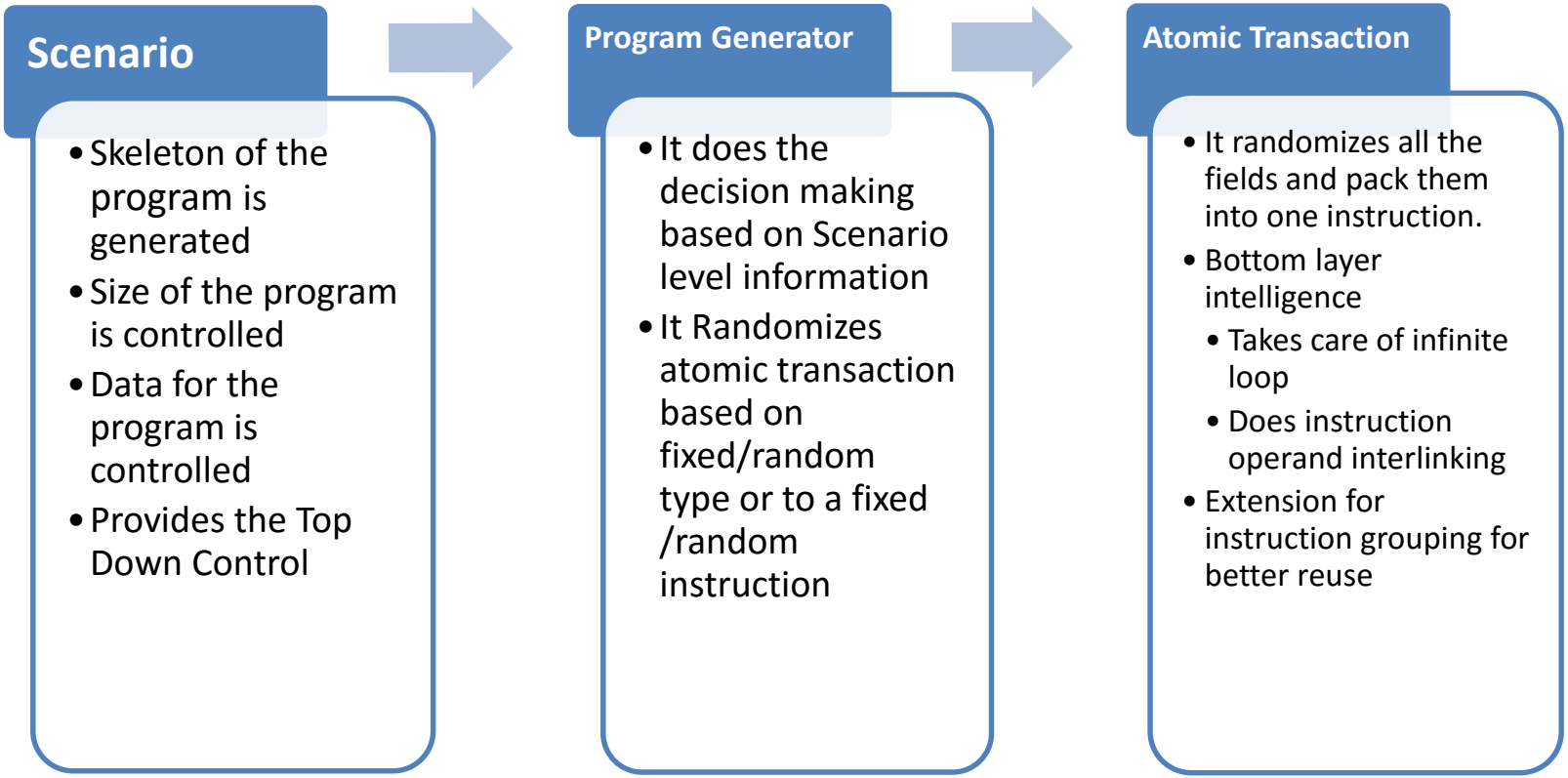
PROPOSED METHODOLOGY

- *Efficient constrained random stimuli generation mechanism for creating meaningful and highly reusable scenarios*
- *Focus on running top level use cases with minimum efforts to achieve high confidence*
- *Reducing the debug time for better time-to-market*
- *A methodology for processor verification using the open sources UVM , SV & C/C++.*



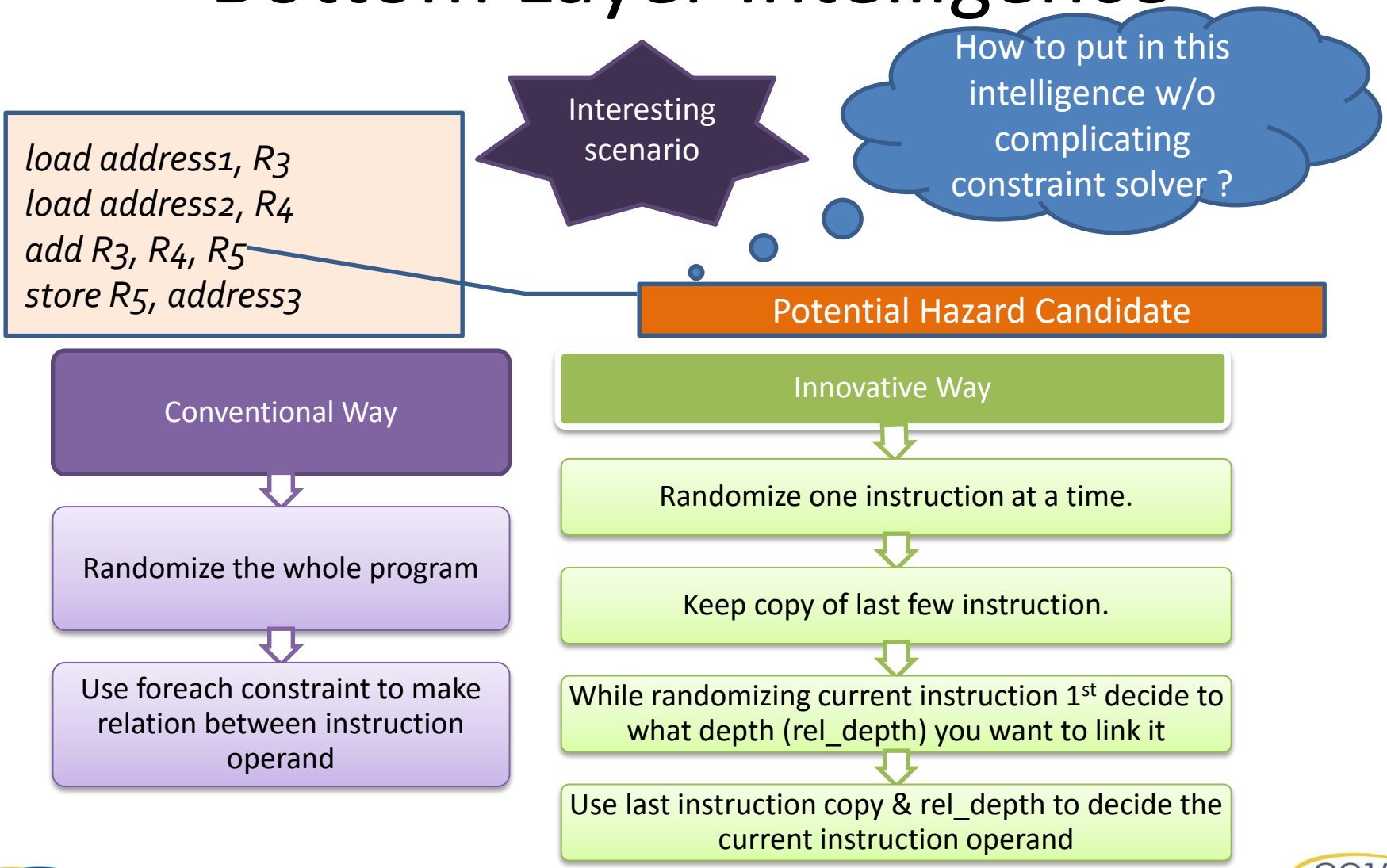
Proposed Stimulus Generation Flow

- A fine blend of Top Down control and Bottom layer intelligence
- Better control over random stimuli and high reuse



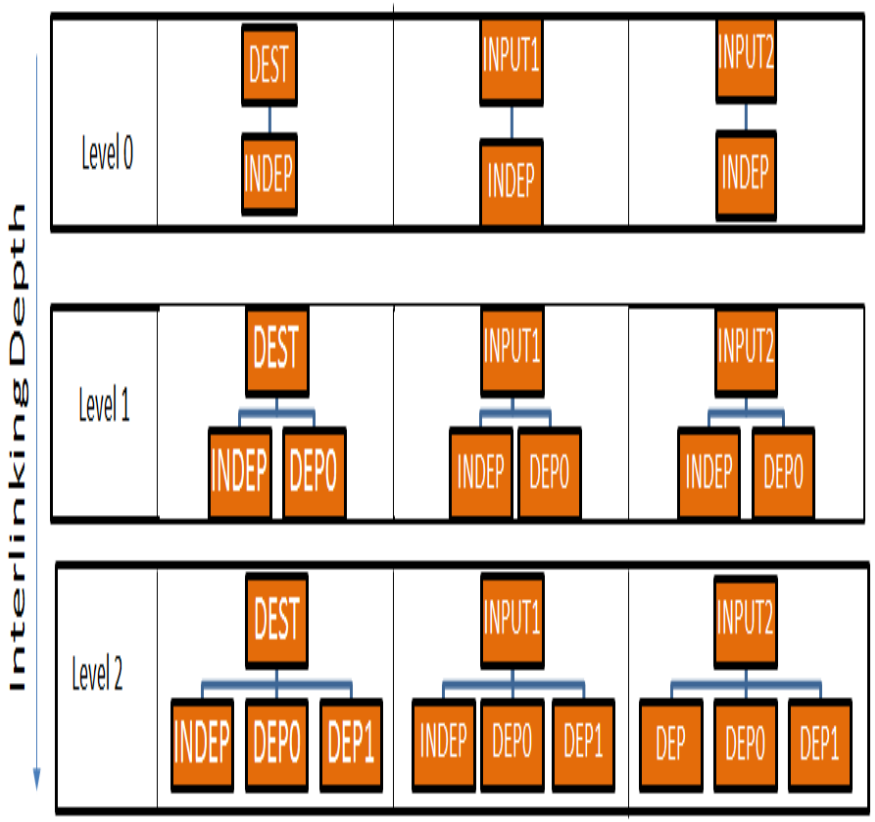
© Accellera Systems Initiative

Bottom Layer Intelligence



© Accellera Systems Initiative

Bottom Layer Intelligence

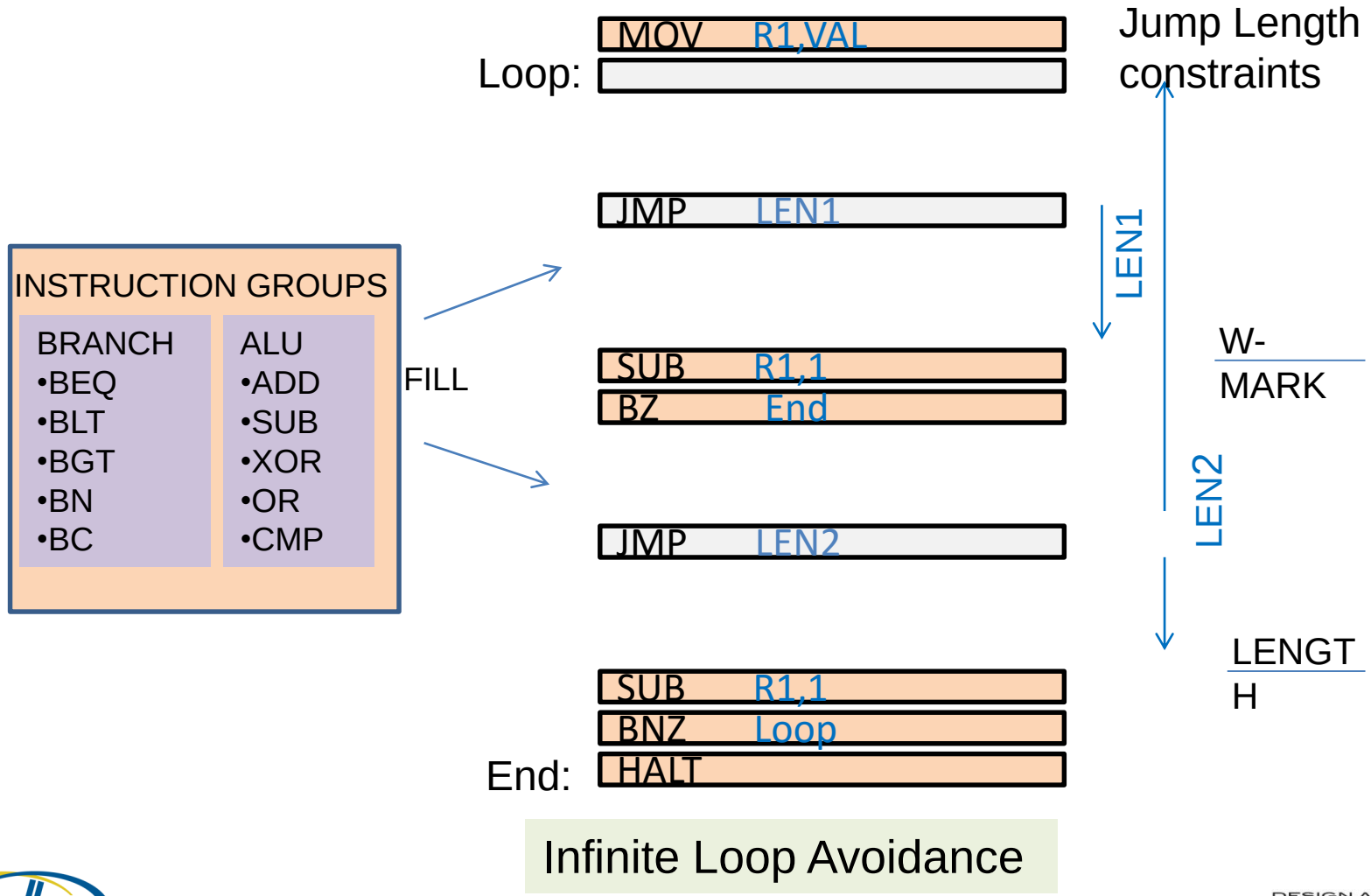



```
class processor_transaction_atomic extends uvm_sequence_item;

// represent instruction groups
rand    processorMasterCommandT    master_cmd;
//  represents individual instructions
rand    processorCommandT          processor_command_type;

// Variables for construction the final instruction
bit      [31:0]   processor_instruction;
rand bit [7:0]    processor_input_operand_1;
rand bit [7:0]    processor_input_operand_2;
rand bit [7:0]    processor_destination_address;

constraint processor_master_command_decoding {
  (master_cmd) inside
      { /*define the valid instruction groups here */};
  (master_cmd == MASTER_ADD)  ->
      processor_command_type == PROCESSOR_CMD_S_ADD;
  (master_cmd == SCALAR_COMMANDS) -> {processor_command_type inside
    {PROCESSOR_CMD_S_MOV ,PROCESSOR_CMD_S_ADD };}}

constraint processor_operand interlinking {
 (dest_op_relation) inside { INDEP, DEP0, DEP1 }
 (dest_op_relation == INDEP) -> {};
 (dest_op_relation == DEP0)  ->
   {processor_destination_address == last_operand_type};
 (dest_op_relation == DEP1 ) ->
   {processor_destination_address == 2nd_last_operand_type}};

function generate_instruction_data();
   if(processor_command_type ==PROCESSOR_CMD_S_ADD)
   begin
    processor_instruction[23:16]   = processor_destination_address;
    processor_instruction[15:8]    = processor_input_operand_1;
    processor_instruction[7:0]     = processor_input_operand_2;
   end
endfunction : generate_instruction_data

endclass : processor_transaction_atomic
```

Operand Interlinking

© Accellera Systems Initiative

Proposed flow in action



Infinite Loop Avoidance

Debugging Hooks

- Zero time Reference model Vs pipelined processor
- Checking only at interfaces is not enough for complex scenarios
- Register trace queue based Checker

Debug cannot be an afterthought.

- Out-of-order execution of pipeline Vs In-order execution of model
- Need checker based on register content change – Data trace checker

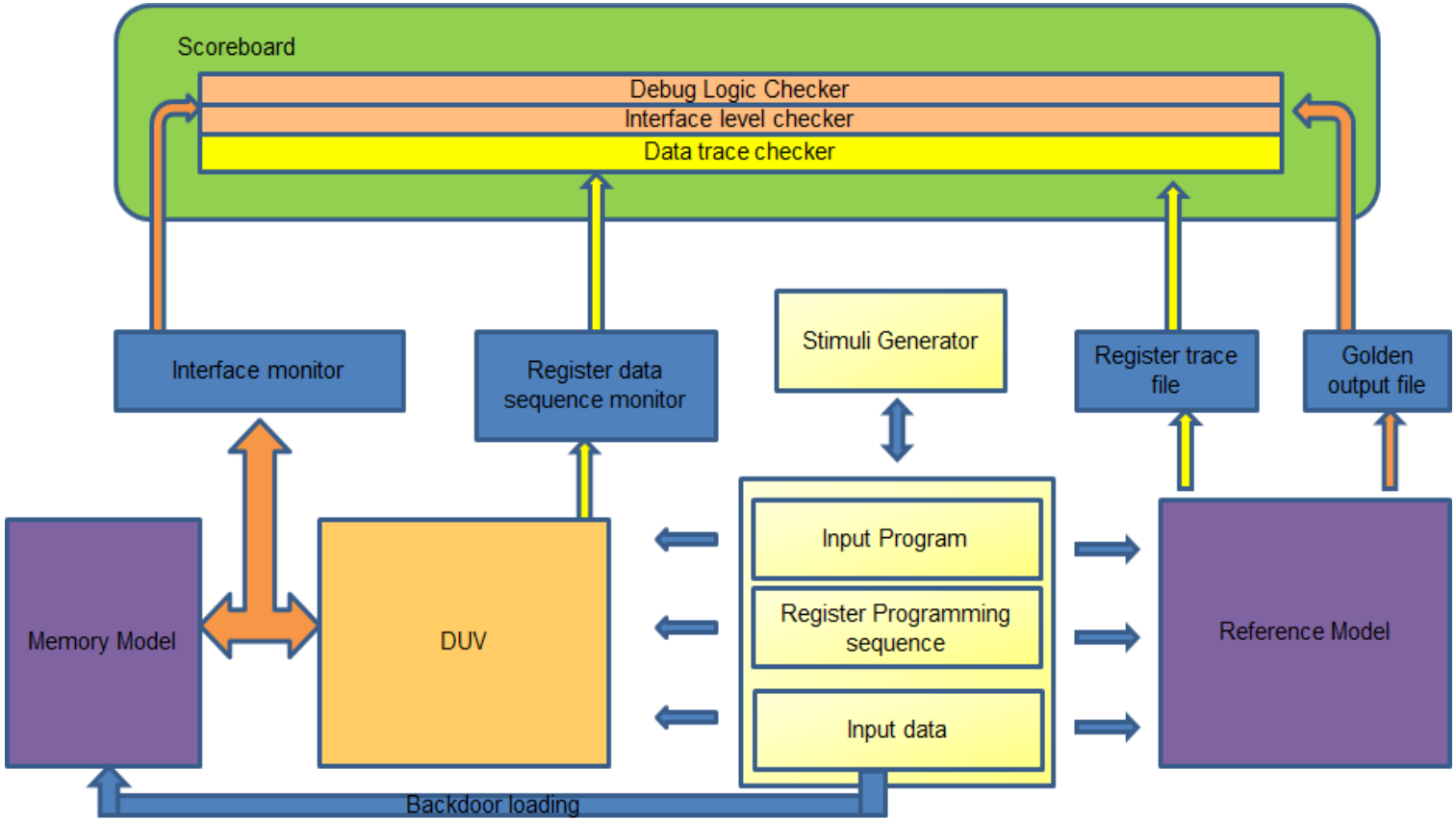
Localization of Failure

- To get the desired confidence running directed use cases is a must
- Switch based flow for directed stimulus which uses program , data/images and configuration as file based input

Scenario Replication

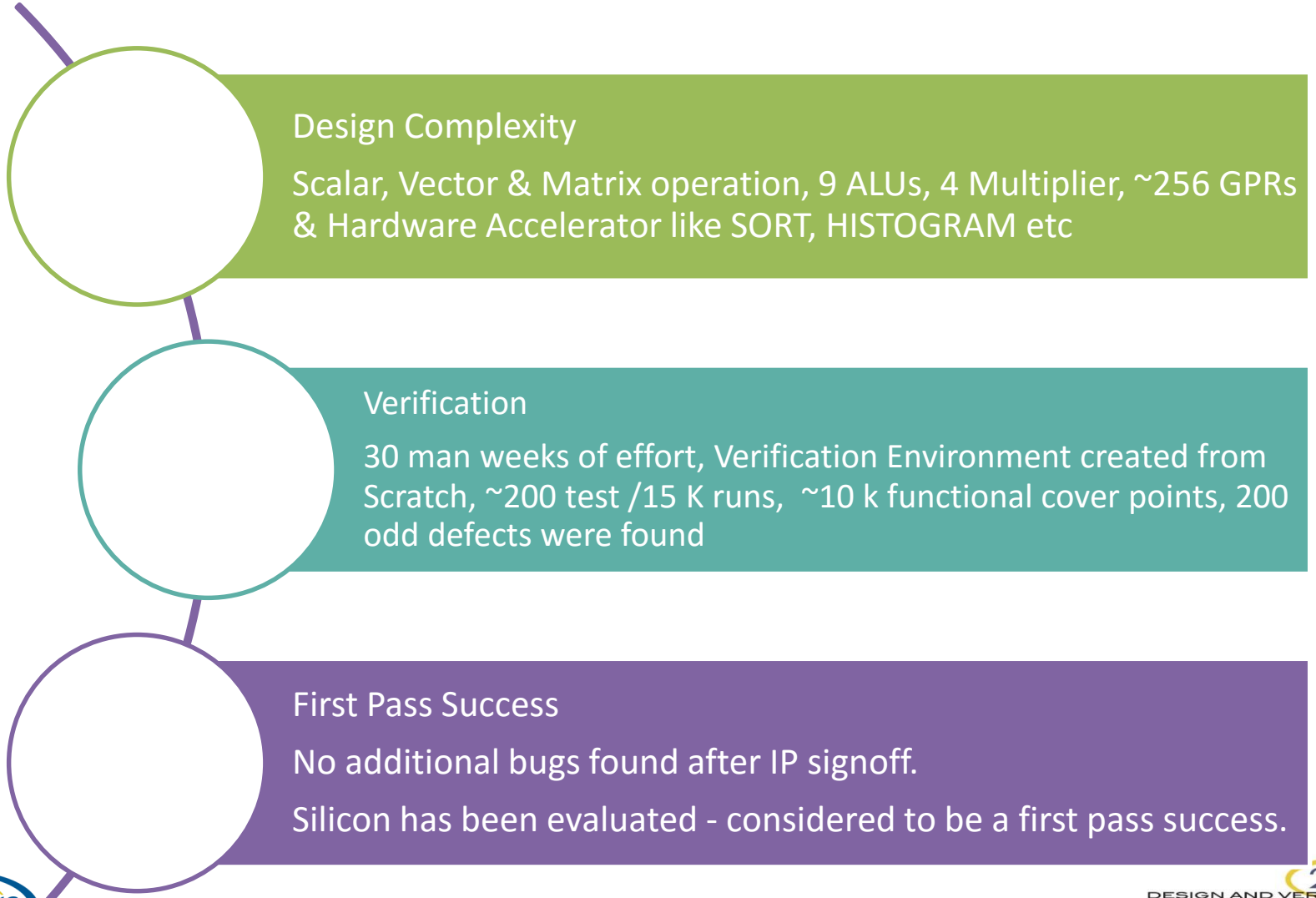
© Accellera Systems Initiative

Debugging Hooks



© Accellera Systems Initiative

Evaluation of the Proposed Flow



© Accellera Systems Initiative

Thank You
Q & A