2024
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
MARCH 4-7, 2024

Gherkin Implementation in SystemVerilog Brings Agile Behavior-Driven Development to UVM

William L. Moore

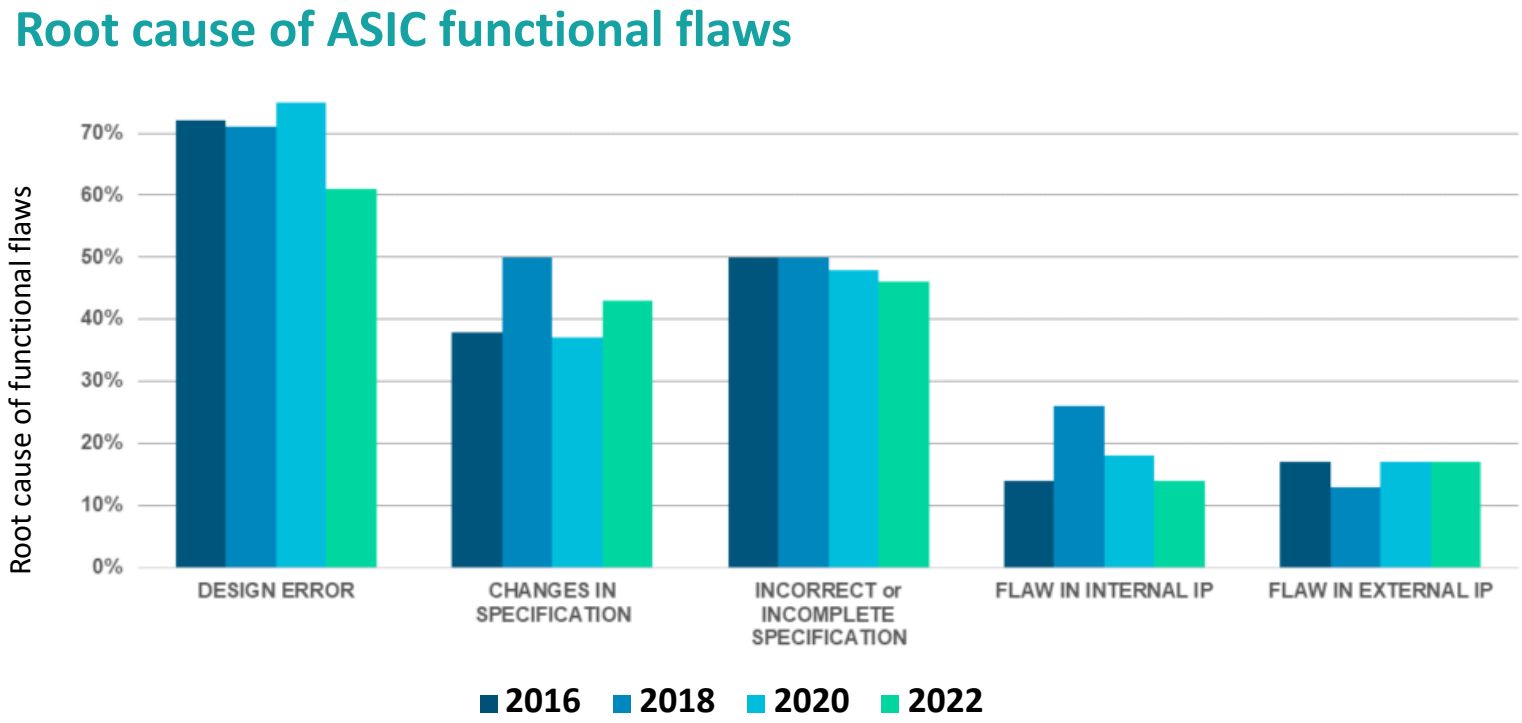
Paradigm Works

PARADIGM® WORKS

accellera SYSTEMS INITIATIVE

The Problem with Documentation

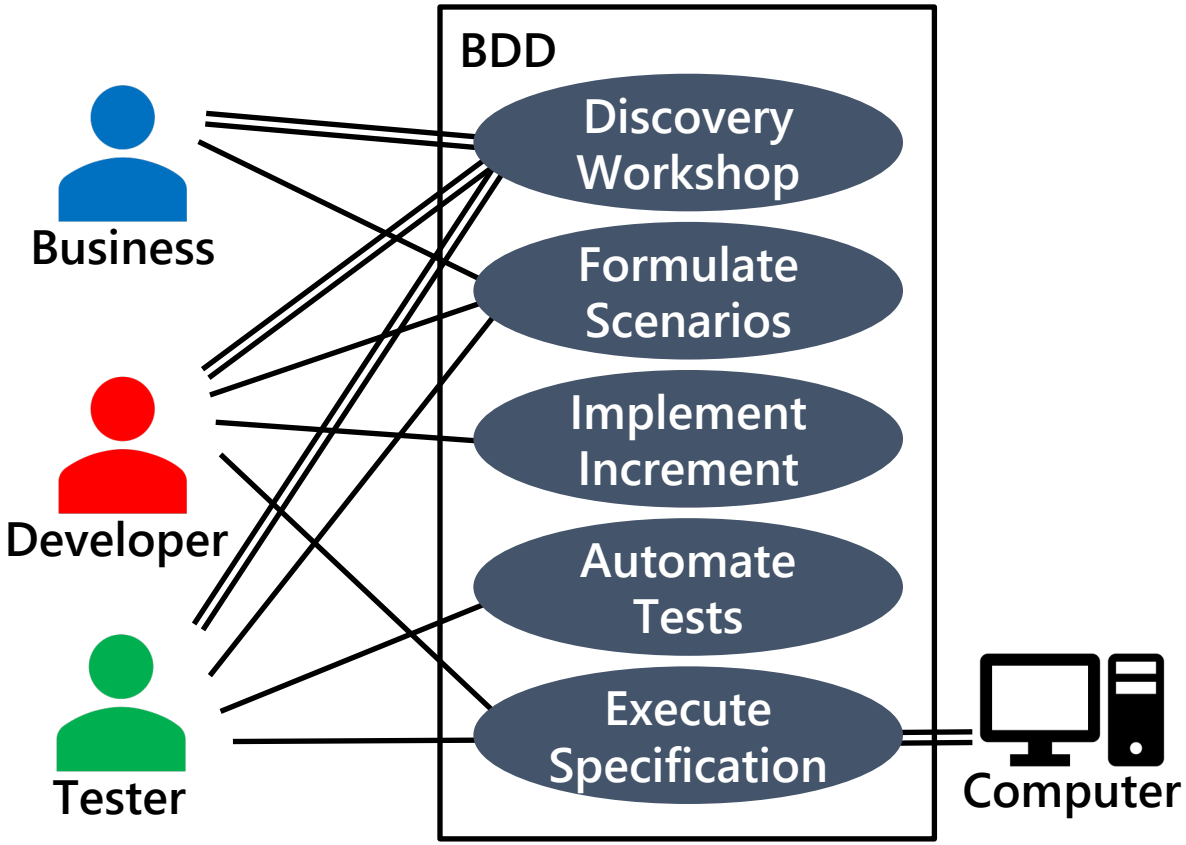
- Static
- Asynchronous
- Separate
- Richly formatted
- Specialized
- Causes bugs!



Source: Wilson Research Group and Siemens EDA, 2022 Functional Verification Study

Behavior-Driven Development (BDD)

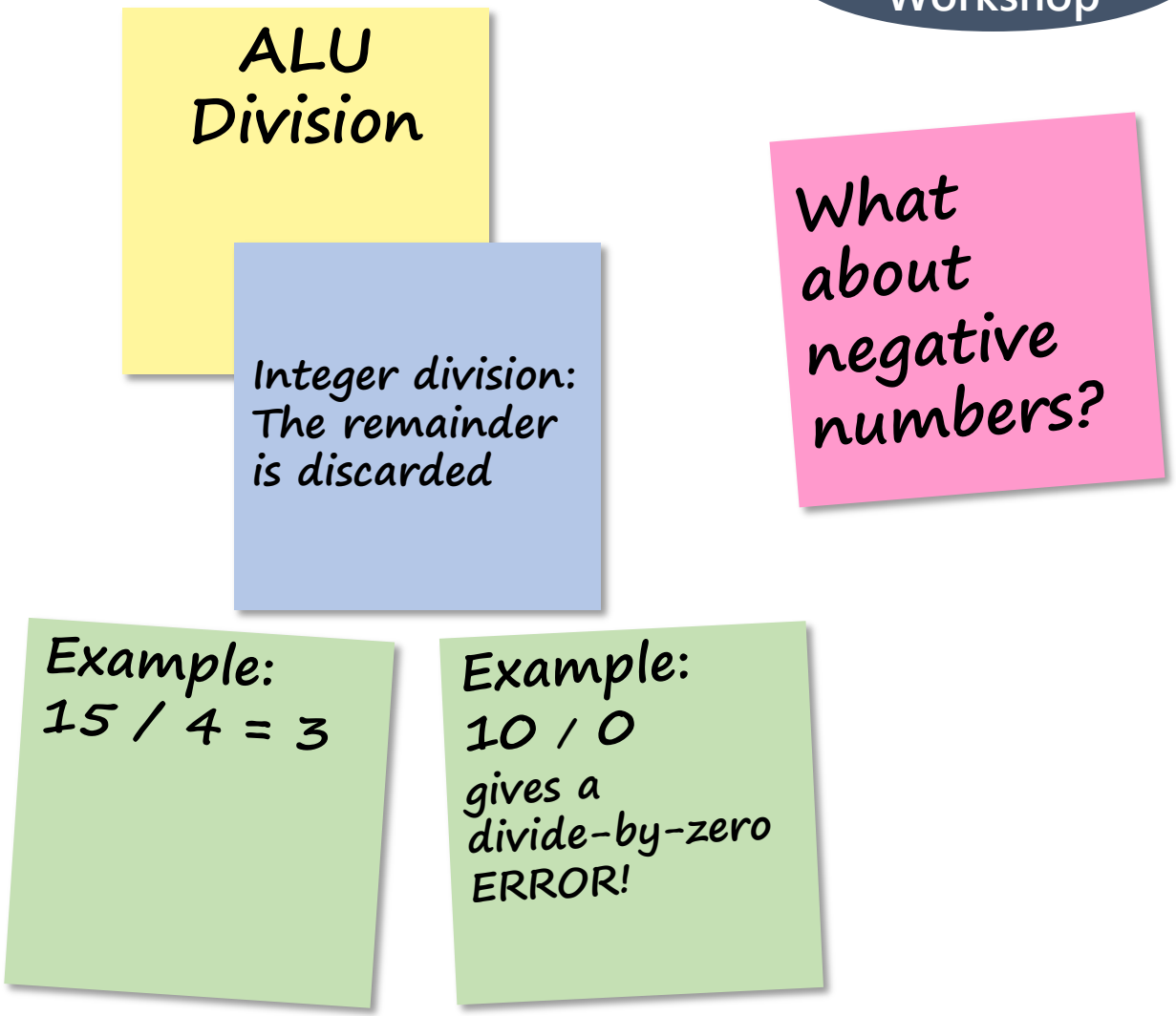
- Agile software methodology
- Team collaboration and communication
- Concrete examples
- Natural language text files
- Automated executable specifications
- Living documentation



Discovery Workshop

Illustrative Example

- Arithmetic logic unit (ALU)
- Integer division feature
- Behavior
- Corner cases and exceptions
- Examples
- Questions



Gherkin

- Executable specification AKA feature file
- Open-source domain-specific language
- Plain text file
- Natural language
- Keywords provide structure
- *Given-When-Then* steps

```
# This Gherkin feature file's name is alu_division.feature

Feature: Arithmetic Logic Unit division operations

  The arithmetic logic unit performs integer division.

  Scenario: In integer division, the remainder is discarded
    Given operand A is 15 and operand B is 4
    When the ALU performs the division operation
    Then the result should be 3
    And the DIV_BY_ZERO flag should be clear

  Scenario: Attempting to divide by zero results in an error
    Given operand A is 10 and operand B is 0
    When the ALU performs the division operation
    Then the DIV_BY_ZERO flag should be raised
```

BDD for Silicon Projects

- RTL and verification environments are code
- Same roles in silicon as software
- Different increments
- Feature could be a verification component
- No native SystemVerilog support for Gherkin
- Until…

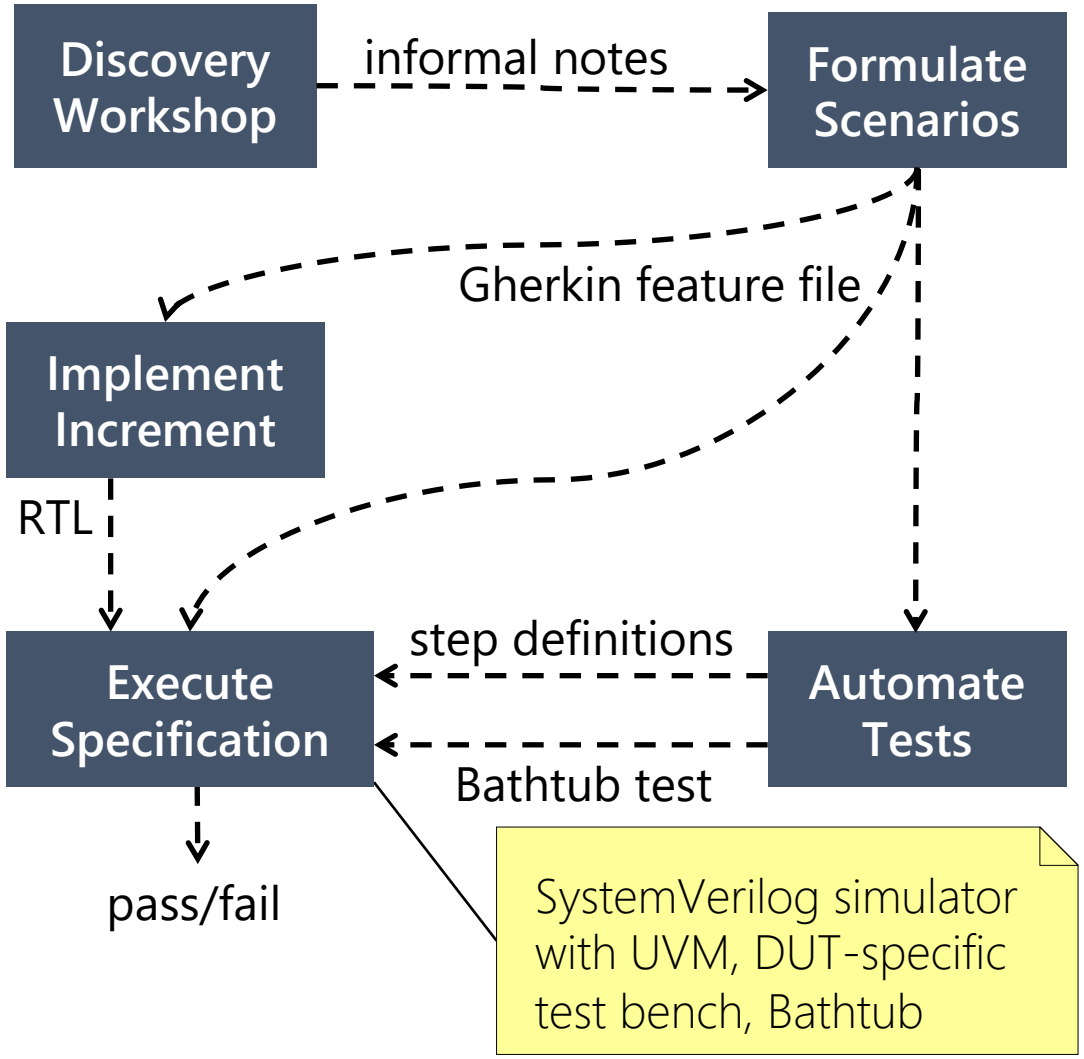
Bathtub

- Pure native SystemVerilog library
- Built on UVM
- Runs in simulator
- Parses Gherkin files
- Executes specification against DUT
- Runs virtual sequences on environment's sequencer

BDD
Automated
Tests
Helping
Teams
Understand
Behavior

Bathtub Flow

- Map Gherkin steps to SystemVerilog tasks
- Encapsulate tasks, assertions in UVM sequences
  - Parameterized step definitions
  - Class needed for every step
- Write a Bathtub UVM test
  - Instantiate bathtub object
  - Configure with feature file names and virtual sequencer
- Compile and simulate until tests pass



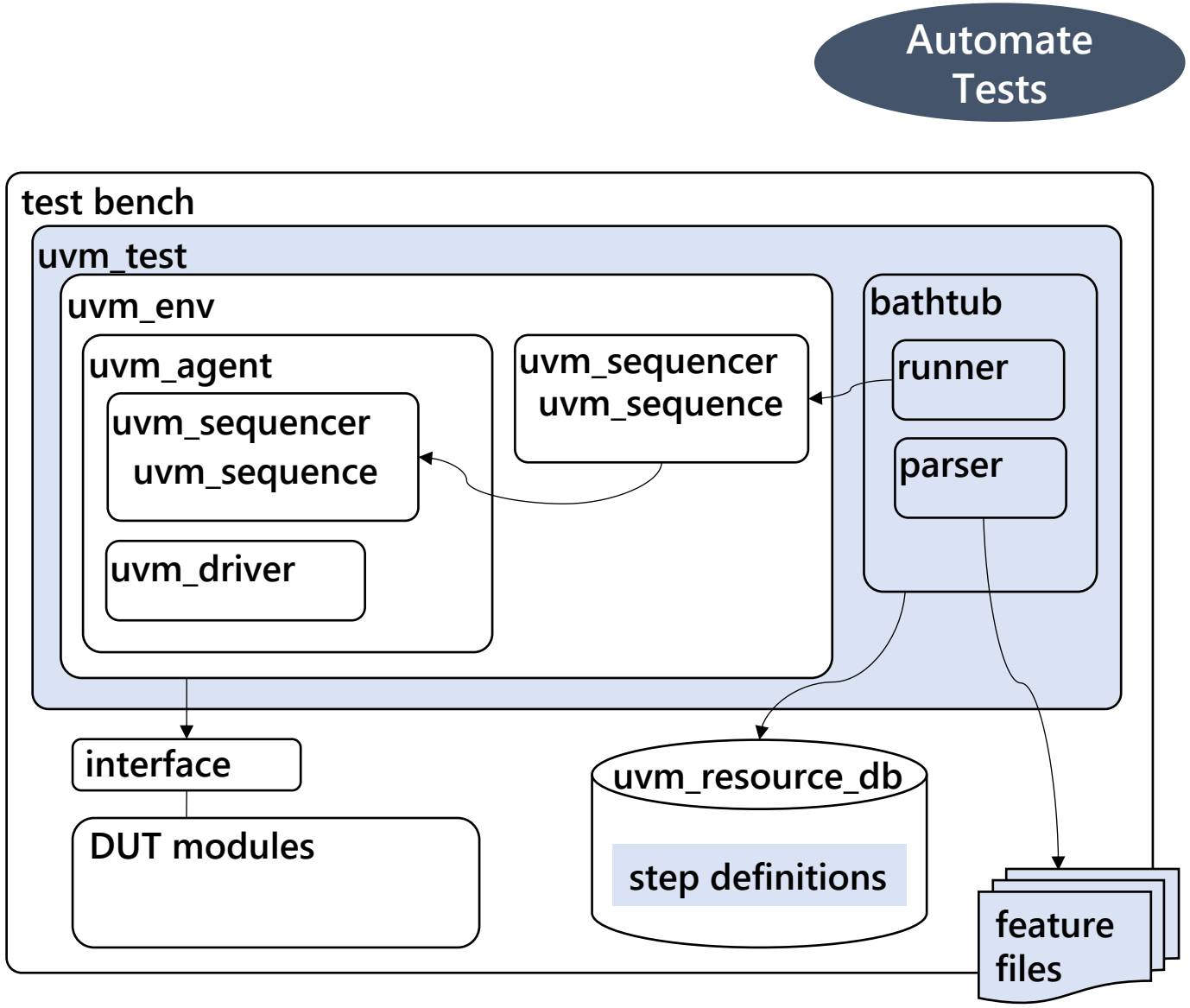
Automate Tests

Sample Step Definition

```
// Given operand A is 15 and operand B is 4
class set_operand_A_and_B_vseq
  extends alu_base_vsequence
  implements bathtub_pkg::step_definition_interface;

    `Given("operand A is %d and operand B is %d")

    int operand_A, operand_B;

   `uvm_object_utils(set_operand_A_and_B_vseq)
    function new (string name);
        super.new(name);
    endfunction : new

    virtual task body();
        // Extract the parameters
        `step_parameter_get_args_begin()
        operand_A = `step_parameter_get_next_arg_as(int);
        operand_B = `step_parameter_get_next_arg_as(int);
        `step_parameter_get_args_end

        // Do the work using the API in the base sequence
        super.set_operand_A(operand_A);
        super.set_operand_B(operand_B);
    endtask : body
endclass : set_operand_A_and_B_vseq
```

Bathtub and UVM

- Shaded regions are Bathtub-specific
- Unshaded regions are reused unchanged
- Resource database stores step definitions
- Bathtub parses Gherkin, collects steps
- Runs step definitions on virtual sequencer



Execute Specification

Bathtub Operation at Time Zero

```
`Given("operand A is %d and operand B is %d")
```

Macro with parameterized template string from step definition.

```
uvm_resource_db#(uvm_object_wrapper)::set(
  "operand A is ([0-9]+) and operand B is ([0-9]+)",
  "bathtub_pkg::step_definition_interface",
  set_operand_A_and_B_vseq::get_type());
```

Bathtub converts "%d" to regular expression "([0-9]+)" and stores the step definition object wrapper in the resource database. The name is a constant.

Scope	Name	Value
operand A is ([0-9]+) and operand B is ([0-9]+)	**bathtub_pkg::step_definition_interface**	**set_operand_A_and_B_vseq**
the ALU performs the division operation	bathtub_pkg::step_definition_interface	do_division_operation_vseq
The result should be ([0-9]+)	bathtub_pkg::step_definition_interface	check_result_vseq
The DIV_BY_ZERO flag should be (\S*)	bathtub_pkg::step_definition_interface	check_DIV_BY_ZERO_flag_vseq

Execute Specification

Bathtub Runtime Operation

```
Given operand A is 15 and operand B is 4
```

Step text from feature file.

```
uvm_resource_db#(uvm_object_wrapper)::get_by_name(
  "operand A is 15 and operand B is 4",
  "bathtub_pkg::step_definition_interface", 1);
```

Bathtub looks for regular expression in resource database that matches step text.

Scope	Name	Value
operand A is ([0-9]+) and operand B is ([0-9]+)	**bathtub_pkg::step_definition_interface**	**set_operand_A_and_B_vseq**
the ALU performs the division operation	bathtub_pkg::step_definition_interface	do_division_operation_vseq

Finds match, instantiates sequence from object wrapper, and configures instance with feature file step text.

```
`uvm_do_on(sequence, sequencer);
```

Runs sequence. Step parameter macros in body() task use $sscanf() and format string to extract values 15 and 4 from step text.

Bathtub Results

- Ran on two SoC project blocks
- Success from discovery workshop to regression
- BDD is not verification, but dovetails
- Feature file & step definition advantages, challenges
- Communication, planning, reuse, debug benefits
- Power of SystemVerilog language
- Available open-source at https://github.com/williaml33moore/bathtub

Questions

Thank you!