Does It Pay Off To Add Portable Stimulus Layer On Top Of UVM IP Block Test Bench?

Xia Wu, Team Lead, Syosil ApS, Taastrup, Denmark (xia@syosil.com)
Jacob Sander Andersen, CTO, Syosil ApS, Taastrup, Denmark (jacob@syosil.com)
Ole Kristoffersen, Project Manager, Ericsson Lund, Sweden (ole.kristoffersen@ericsson.com)
Background

- Portable stimulus has becoming the emerging trend
  - Reusable stimulus and test
  - Aiming at faster functional coverage closure
  - Uniform way of understanding and test the requirements
- Accellera System Initiatives announced the release of the Portable Test and Stimulus Standard (PSS) on June 2018
- Support PSS domain specific language (DSL) and C++
- PSS allow the creation of a scenario from partial definition
  - Loop, branch, parallelism etc to control the activities with
  - Concepts like resource, data flow object, constraint
Motivation

• Evaluation of PSS deployment on a block level test bench
  – System-level test bench are the main target of early adoption
  – Few publication are found on block level test bench
  – Makes good sense to start from block level test bench

• Our motivation
  – Find out the effort and the challenge to migrate a UVM block level test bench into PSS-based solution
  – Evaluate the PSS support of the tool
Benefit for PSS in block level test bench

• Faster functional coverage closure
  – By aligning stimulus generation with coverage goals
  – Beneficial for projects with different parameter setups

• Reduced number of tests in the regression
  – Reduced regression time
  – Reduced use of regression license and machine power

• Reusability
  – Vertical reuse on the sub-system and full system test

• Visualisation of the test scenario
  – Improve communication across teams
PSS tool

- Perspec is a modelling tool by Cadence
  - PSS model development
  - Scenario creation
  - Scenario randomization
  - Target code generation
  - Abstract debugging
- Perspec supports both PSS-DSL and C++, as well as Cadence’s own System Level Notation (SLN).
- PSS as a layer on top of UVM
IP block details

- Our target DUT is a highly configurable filter chain system
  - A series of filter engine blocks which process data independently or in chain.
  - A combination of generics and run-time configuration
  - Benefit in closing coverage by PSS is high
  - The complexity level of UVM sequence is high. Requires correct timing and stimuli.
Modelling in PSS

- Example of a PSS based test
  - Configure six filter engines in parallel and randomize
  - Schedule data stimuli sequence and send them to DUT
- Test is build up from a sequence of atomic actions.
- The full valid sequence is also an action, called a compound action.
Problems discussed in the paper

• Compile-time parameters
  - Using Cadence SLN can simplify the coding and is a good complement to the current standard.

• Run-time configuration
  - Creating the model in layers and distributing the complexity into each layer

• Inheritance
  - A powerful methodology to create test cases, but it can be further improved for code reuse.

• Partial description
  - Successful solving is heavily dependent on good constraint sets and coding style.

• Semantics equivalence
  - Checking the potential semantics inequivalence between the PSS model and UVM tests
Problems discussed in the paper

- Compile-time parameters
  - Run-time configuration
  - Inheritance
  - Partial description
  - Semantics equivalence
How to model compile-time generics

• DUT have several different block parameter setups
  – The UVM TB includes a package with ifdef around each sets of parameter
  – The regression runs all different setups

• PSS also need to be generic
  – Generate different test cases for each parameter setup
  – Parameter sets are tedious to do in PSS-DSL
  – Cadence SLN table command provides a mechanism to capture code repetition
How to model compile-time generics

```c
#include, #nfilt, #nsec, #ncpsec, #wc,...
SETUP_DEFAULT, 6, 2, 4, 18,...
SETUP_2380, 2, 3, 8, 17,...

#ifdef SETUP {
#define SETUP "SETUP_DEFAULT";
}

table from csv_to_table("..\csv\setups.csv","Setup Info", (csv_column("name")
== SETUP)) with {
    const NUM_OF_CHANNELS: int = #nfilt;
    const NCPSEC_P: int = #ncpsec;
    const NUM_OF_SECTIONS: int = #nsec;
    const COEFFICIENT_WORD_WIDTH: int = #wc;
...
Problems discussed in the paper

• Compile-time parameters
  ➢ Run-time configuration

• Inheritance

• Partial description

• Semantics equivalence
How to model the run-time configuration

- Covering all the possible scenario of the chaining of the filter engines.
- In SystemVerilog test, we rely on the constraint random and a large number of seeds.
- In PSS based test, we define the coverage goal beforehand and generate scenarios which directly cover that goal.
How to model the run-time configuration

• Randomization should be one action per filter engine.
• Utilizing the input/output data stream in action to model a virtual chain.
• Defining variables in the action which directly links to the cover point, e.g. starting position, index in the chain, etc.

```c
stream data_t {
    rand int in [0..NUM_OF_CHANNELS-1] source;
    rand int in [1..NUM_OF_CHANNELS-1] step;
    rand int in [0..NUM_OF_CHANNELS-1] chain_starting_location;
    rand int in [1..NUM_OF_CHANNELS] chain_size;
    rand int in [1..NCPSEC_F] npsec_in_use;
}

action channel {
    rand role_in_ch_e ch_role;
    rand int in [1..NCPSEC_F] npsec_in_use;
    rand int in [0..NUM_OF_CHANNELS-1] chain_starting_location;
    rand int in [1..NUM_OF_CHANNELS] chain_size;
    rand int in [0..NUM_OF_CHANNELS-1] index_in_chain;
    lock channel_t channel_r;
};
```
How to model the run-time configuration

```plaintext
action channel_middle : channel {
  constraint ch_role == MIDDLE;
  input data_t in_data;
  output data_t out_data;
  // Use this to find the previous channel
  constraint in_data.source == channel_r.instance_id - 1;
  // Assign the value from the input data
  constraint chain_starting_location == in_data.chain_starting_location;
  constraint index_in_chain == in_data.step;
  constraint chain_size == in_data.chain_size;
  constraint npsec_in_use == in_data.npsec_in_use;
  // Pass the value to the output data
  constraint out_data.source == channel_r.instance_id;
  constraint out_data.chain_starting_location == chain_starting_location;
  constraint out_data.step == index_in_chain + 1;
  constraint out_data.chain_size == chain_size;
  constraint out_data.npsec_in_use == npsec_in_use;
};
```
Problems discussed in the paper

- Compile-time parameters
- Run-time configuration
  - Inheritance
- Partial description
- Semantics equivalence
Inheritance

- Inheritance is supported in PSS LRM
  - Action can be extended
  - Important to plan a good structure before implementation
- One action per UVM sequence, and move the scheduling into PSS model
- Limitation in DPI function
Problems discussed in the paper

- Compile-time parameters
- Run-time configuration
- Inheritance
  - Partial description
- Semantics equivalence
PARTIAL DESCRIPTION

• No need to specify all the steps
• Rely on the data flow and the solver to get a complete test scenario.
• Difficult for multiple parallel process, with different configuration from action to action
• The success of abstract partial configuration is highly dependent on the coding style
PARTIAL CONFIGURATION

```
action run_on_one_chan {
  input cfg_t in_cfg;
  lock cfg_rsnc_t cfg_rsnc;
  // force different id by lock
  constraint in_cfg.chan_no == cfg_rsnc.slot_num;
  rand int in [0..NUM_OF_CHANNELS-1] channel_num;
  rand int in [1..NCPSEC_P] npsec_in_use;
  constraint npsec_in_use == in_cfg.npsec_in_use;
  activity {
    sequence {
      do run_1 with {
        npsec_in_use == this.npsec_in_use;
        channel_num == this.channel_num;
      };
      do power_control_on with {
        channel_num == this.channel_num;
      };
      do run_2 with {
        npsec_in_use == this.npsec_in_use;
        channel_num == this.channel_num;
      };
      do power_control_off with {
        channel_num == this.channel_num;
      };
      do run_3 with {
        npsec_in_use == this.npsec_in_use;
        channel_num == this.channel_num;
      };
    };
  };
};
```
Problems discussed in the paper

• Compile-time parameters
• Run-time configuration
• Inheritance
• Partial description
  ➢ Semantics equivalence
Semantics equivalence

- **Test steps:**
  - Configure the DUT and send data stimuli.
  - And then randomize and configure the DUT again and send data stimuli again.

- **Problem with the second configuration**
Semantics equivalence

• Problems with the reconfiguration test
  – Synchronization between C and SystemVerilog should be extended
  – PSS randomization is done before the test is created. Therefore more buffer is needed to store the randomization results.

• Equivalence check
  – Test result
  – Regression result
  – Unexpected coverage hole.
Conclusion

- Realistic to add portable stimulus layer to an UVM test bench with reasonable effort.
- The effort is paid off by improved verification efficiency, faster functional coverage closure and reduced tests in the regression.
- Promotes reusability and potentially reduce redundant test development time in other target platforms.
- Useful add-on to the existing dynamic verification techniques.
Questions