Portable Stimuli over UVM
using portable stimuli in HW verification flow

Efrat Shneydor, Cadence Design Systems, Israel (efrat@cadence.com)
Slava Salnikov, Ben Gurion University & Texas Instruments, Israel (slava.s@ti.com)
Liran Kosovizer, Texas Instruments, Israel (lirank@ti.com)
Dr’ Shlomo Greenberg, Ben Gurion University, Israel (shlomog@ee.bgu.ac.il)
Agenda

• UVM challenges
• PSS solvability
• PSS to UVM flow
• Summary
Texas Instruments Wi-Fi router

- Multiple CPU cores, power domains & HW hierarchies
- Advanced verification environments, using Specman and UVM-e
  - eight levels of reuse
Verification requirements, stimuli generation

- Capture rules of system behavior
- Achieve a robust, re-usable solution for system level test composition
- Changes of DUT should not require more than minimal modifications of the TB
Each scenario is well defined
Sequences libraries

WIFI packet

```plaintext
extend WIFI_TRANS trans_seq {
  wait @env.ready;
  do check_state;
  case cur_state {
    disconnect : {
      do scan_for_router;
    };
    connected : {
      gen next_state keeping {
        //...
      };
    };
  };
  emit env.wifi_done;
}
```

Power management

```plaintext
extend POWER trans_seq {
  if cur_state == idle {
    gen next_trans keeping{
      //...
    };
    driver.execute_item(next_trans);
    do next_trans keeping {
      //...
    };
    emit env.power_done;
  }
}
```

I2C external interface

```plaintext
extend i2c trans_seq {
  case cur_state {
    idle : {
      driver.i2c_start();
    };
    r_w : {
      gen next_trans keeping{
        //...
      };
      driver.execute_item(next_trans);
    };
    //...
    emit env.i2c_done;
  }
}
```
Interdependence of sub-modules

• Sequence should contain synchronization aids
• Multi-channels sequences

Device shall not enter low power mode before notifying router with a dedicated packet sequence. Device shall not send a sleep mode request from any power state but ‘idle’.
How far can we stretch UVM?

- Delay SW reset requests until end of I2C transaction
- Device shall not initiate I2C transaction from low power mode
Agenda

- UVM challenges
- PSS solvability
- PSS to UVM flow
- Summary
The PSS reuse solution

- Fully reusable
- One-time modelling effort
- Invariant to internal env changes

PSS MODEL

Component 1
- Action A
- Action B
- Action C

Component 2
- Action A
- Action B
- Action C

Non platform specific!

Platform specific API

generate

test code

generate

test C code

Non platform specific!

Platform specific API
Generating Scenarios Using PSS

Drag action/s

Click ‘solve’ to create concrete scenario.

Click ‘generate test’ to create code
Portable Stimuli actions

**action** ce_tx_assoc_req {
*input* prev : from state_var;
*output* next : to state_var;

constraint prev.state == auth_unassoc;
constraint next.state == wait_assoc_rsp;

**action** ce_rx_assoc_rsp {
*input* prev : from state_var;
*output* next : to state_var;

constraint prev.state == wait_assoc_rsp;
constraint next.state == connected_entry;
Generating Scenarios Using PSS

Drag action/s
Click ‘solve’ to create concrete scenario.
Click ‘generate test’ to create code
PSS over UVM

Scenario generated by PSS

Tests run on top of UVM TB

UVM TB takes run time decisions
Agenda

• UVM challenges
• PSS solvability
• PSS to UVM flow
• Summary
PSS/UVM Partitioning – the hybrid model

- Perspec scenario provides high level test case backbone
- UVM sequencers handle signal – level transactions
Driving the scenario, from PSS to $e$

- Config struct passed information from PSS to UVM TB:
  - PSS action defines the high level of what has to be sent
  - Action body passes config struct to the $e$ testbench
  - $e$ testbench executes the required transaction/s
From action to test

```cpp
action change_state {
    input prev : from state_var;
    output next : to state_var;

    cfg : cfg_s ;

    exec body {
        // Imported function
        send_next(cfg);
    };
}
```
The action conditions – when can be executed, and what the results are:

```
action tx_auth like change_state {
  constraint prev.state == sup_ap_found;
  constraint next.state == wait_auth;
  constraint cfg.direction == TX;
  constraint cfg.transfer_kinds.size() == 1;
  constraint cfg.transfer_kinds[0] == AUTHENTICATION;
}
```

The required scenario:
```
exec body {
  send_next(cfg);
}
```
From action to test

action tx_auth like change_state {
  constraint prev.state == sup_ap_found;
  constraint next.state == wait_auth;
  constraint cfg.direction == TX;
  constraint cfg.transfer_kinds.size() == 1;
  constraint cfg.transfer_kinds[0] == AUTHENTICATION;
};

Create scenario, according to config struct

All fields not constrained here will be randomized, according to UVC constraints
The test flow

```c
extend sys {
    run() is also {
        start perspec_main();
    };

    perspec_main()@sys.any is {
        raise_objection(TEST_DONE);
        t.pss_main();
        drop_objection(TEST_DONE);
    };
}
```

The simulator and Specman start running
Specman calls the C `main` in `run` phase
From now – C test controls the scenario

This code is created automatically by the tool

In each test `pss_main()` is different, based on generated actions
Alterning the e-C synchronization

extend sys {
    run() is also {
        start perspec_main();
    };

    perspec_main()@sys.any is {
        raise_objection(TEST_DONE);
        t.pss_main();
        drop_objection(TEST_DONE);
    };
}

Created automatically

Start the test only after reach MAIN_TEST phase

extend sys {
    perspec_main()@sys.any is first {
        var tf_mgr := tf_get_domain_mgr_of(CORE);
        while TRUE {
            wait @tf_mgr.new_phase_starting;
            if tf_mgr.get_current_phase() == MAIN_TEST {
                break;
            }
        };
        ///...
}

Created manually
Agenda

• UVM challenges
• PSS solvability
• PSS to UVM flow
• Summary
Yet to be added

- Seamless regression invocation
  - Vmanager integration, Perspec regression abilities
- Enhance Debug abilities
- Sync UVM test phases with Perspec scenarios
- Perspec/Specman migration to validation platforms (embedded C code)
- Full coverage closure using Perspec WIFI simulator
• **PSS model**
  – Inputs, Outputs
  – Rules of coexistence
• **e API to test platform**
  – Platform specific implementation
Summary

- Few weeks ramp up period, hundreds of tests created
  - What usually takes several months
  - Model is easily updated to new needs

- Concept shift makes integration not intuitive
- Perspec – C – Specman API impairs seamless integration
- Need to adjust debugging techniques

Bottom line: TI decided to expand the usage of Perspec over UVM
Questions?