Boost Verification Results by Bridging the Hw/Sw Testbench Gap

by
Matthew Ballance
Verification Technologist
Mentor Graphics
Agenda

• Block-level verification techniques
• SoC-level verification and the testbench gap
• UVM Software-Driven Verification package
Block-level Verification

- Verification from the outside in
  - Plan, create scenarios to verify
  - Apply at the design interfaces
  - Confirm correctness of results

- Over a decade of tool development and standardization
  - Automation
    - Constrained random generation
    - Intelligent testbench automation
  - Standardization – Verilog, VHDL, SystemVerilog
  - Methodology and reuse
    - AVM, OVM, VMM, UVM

3 of (total number of slides)
SoC Verification

- Software central to design operation
- Verification from the outside in
  - VIP connected to design interfaces
- Verification from the inside out
  - Software running on the processor

Challenges
- Difficult to coordinate Hw, Sw scenarios
- Little or no automation for Sw test creation
- Difficult to extract data from Sw test
<table>
<thead>
<tr>
<th>Hardware Domain Info</th>
<th>Software Domain Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Test configuration</td>
<td>• Info/error messages</td>
</tr>
<tr>
<td>• Automated test stimulus</td>
<td>• Software state</td>
</tr>
<tr>
<td>• Analysis-data collection</td>
<td>• Sw scenario control</td>
</tr>
<tr>
<td>• Hw scenario control</td>
<td></td>
</tr>
</tbody>
</table>
The Need Isn’t New

- Hw/Sw Testbench connections aren’t new
- Often addressed environment-specific requirements
  - Signal software test pass/fail
  - Obtain randomized data

- Often proprietary
  - Project / company specific
  - Difficult to reuse

- Not UVM-centric
  - Connecting existing UVM infrastructure requires work
What is the UVM SDV Package?

- UVM-based package to enable Software-Driven Verification
  - Connects Hw and Sw domains
  - Simplifies coordination of Hw/Sw verification scenarios
- Scalable across environments
  - Block-level verification in simulation
  - SoC-level in simulation
  - SoC-level in emulation
- Light-weight Sw components
  - Small memory footprint
  - Low processing overhead
UVM SDV Architecture

- ‘C’ API
- SystemVerilog UVM-based classes
- Modular communication mechanism between Hw/Sw
  - Shared memory
  - DPI
- App-level services
  - Hw/Sw synchronization
  - Sw stimulus from UVM
  - Sw state to UVM
  - Sw-initiated Hw stim
Base Hw/Sw Synchronization

- Sw API to pack/unpack data structures
- Obtain Configuration information from UVM
  - Test-customization parameters
- Send Sw messages to UVM
  - Enables common management of Hw/Sw status
  - Reduced overhead vs printf via UART
- Sw control of objections
  - Enables Sw to participate in test termination
SW Stimulus from UVM

- Enables SW to drive stimulus generated in UVM
- Sequence item packed/unpacked by UVM API
- Item unpacked by SDV ‘C’ API

- Reuse existing UVM
  - Sequences
  - Sequence items
  - Sequencers
**SW Stimulus from UVM**

**SW-side Implementation**

- Software implements the UVM driver component
- ‘C’ API mirrors UVM sequencer-port API

```c
req_txn req, rsp;
uvm_sdv_sequencer_driver_t txn_drv;

uvm_sdv_sequencer_driver_init(&txn_drv, "*.m_sdv_driver", // inst path of drv
&req_txn_unpack, // unpack function
&rsp_txn_pack); // pack function

while (true) {
    uvm_sdv_sequencer_driver_get_next_item(
        &txn_drv,
        &req);

    uvm_sdv_sequencer_driver_item_done(
        &txn_drv,
        &rsp);
}
```
SW State to UVM

- SW API exposes SW state via UVM analysis port
- Enables analysis of SW state in UVM
  - Scoreboard
  - Coverage
- More efficient than printf
  - Reduce analysis overhead
- Example:
  - printf via UART: 1600uS
  - Analysis port: 50uS
SW State to UVM
SW-side Implementation

- SW initializes an analysis port
  - Specifies path of publisher in UVM environment

```c
uvm_sdv_analysis_port ap;
sw_txn state;
uvm_sdv_analysis_port_init(&ap, "*.m_state_pub",
                   &sw_txn_pack);
while (true) {
    process_data();
    state.A = read_A();
    state.B = read_B();
    uvm_sdv_analysis_port_write(&ap, &state);
}
```
Initiate UVM Stimulus from SW

- SW can launch sequences in UVM environment
  - Launch any sequence on any agent
- SW can monitor completion
Initiate UVM Stimulus from SW

SW-side Implementation

- Sequence-launch call specifies
  - Sequence name
  - Sequencer path
- Start is non-blocking

- API returns run status

```c
uint32_t id;

id = uvm_sdv_sequence_start("traffic_gen_seq", "*.traffic_seqr");

// Perform other operations while // the sequence is running
while (uvm_sdv_sequence_is_running(id)) {
    // Run software activity
    process_data();
}
```
Extensibility
Graph-Based Stimulus

- Extension API provided for new applications
- Coordinated Hw/Sw tests benefit from stimulus that is
  - Sequential
  - Dynamically reactive to Hw/Sw state
- Graph-Based Stimulus
  - Procedural interaction with Sw
  - Stimulus/feedback mixed
- Enables description of complex scenarios
UVM Software-Driven Verification

- Reusable infrastructure
  - Easily customized for specific environment
- Connects Hw and Sw domains
  - Extends UVM services to Sw domain
  - Extensible to domain-specific applications
- Enables automation and visibility
  - Automated stimulus generation
    - Random, Intelligent
  - Sw-state visibility
- Enables coordinated Hw/Sw tests
Q&A