UVM and Emulation: How to Get Your Ultimate Testbench Acceleration Speed-up

Hans van der Schoot & Ahmed Yehia
{hans_vanderschoot,ahmed_yehia}@mentor.com

accellera SYSTEMS INITIATIVE™

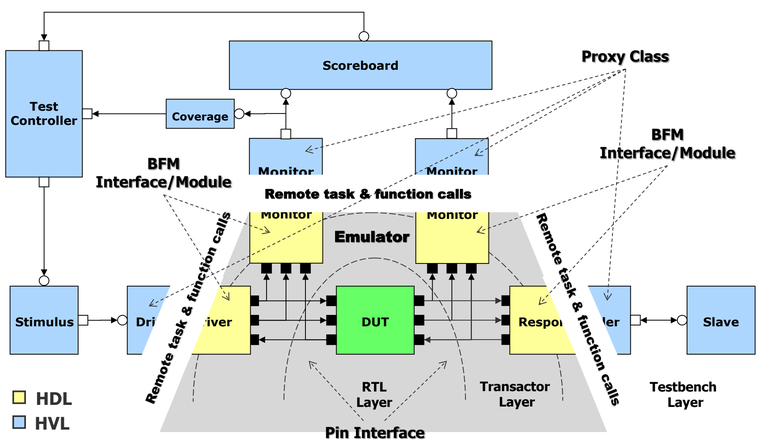
2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

DUAL-TOP FRAMEWORK FOR TESTBENCH ACCELERATION

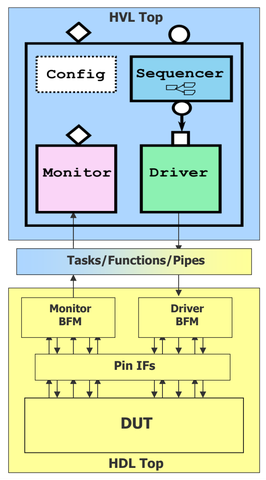
1. Employ two separated HVL and HDL sides
2. Model all timed testbench code for synthesis on the HDL side, leaving the HVL side untimed
3. Devise a transaction-level, function-based communication interface between HVL and HDL sides

Acceleratable UVM Testbench

Single Unified Testbench for Simulation and Acceleration

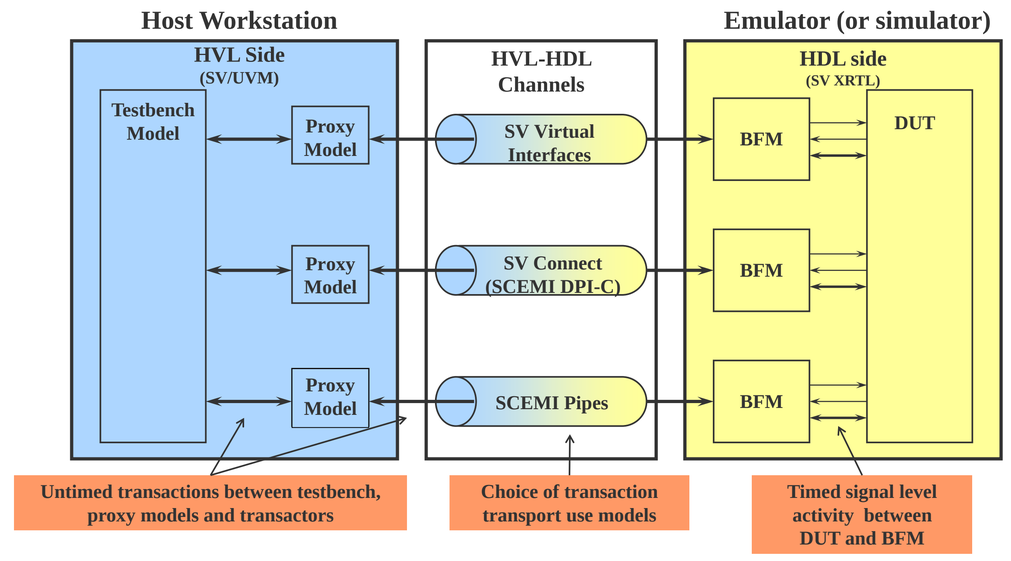


Acceleratable UVM Agent



TESTBENCH ACCELERATION PERFORMANCE DEMYSTIFIED

$$t_{[total]} = t_{[HVL]} + t_{[HVL\text{-}HDL]} + t_{[HDL]}$$



Throughput : $t_{[HDL]} / t_{[total]}$

H/W Bound : $t_{[HDL]} / t_{[total]}$ >> $(t_{[HVL]} + t_{[HVL\text{-}HDL]}) / t_{[total]}$

TACKLING TESTBENCH ACCELERATON PERFORMANCE

Questa ADVANCED VERIFICATION

S/W Time

$t_{[HVL]}$

- Workstation executing HVL-side testbench threads causing emulated design clocks to be stopped
- **Profile conventional simulation runs** to analyze the testbench portion – discount the RTL DUT to be allotted to the emulator
- **Code SV/UVM testbench for simulation performance**, not just functionality
- **Maximize constraint solver performance**
- **Move assertions and coverage models to the HDL-side** where applicable and capacity is not of concern

S/W-H/W Communication Time

$t_{[HVL\text{-}HDL]}$

- Workstation-emulator context-switching and data transfer
- **Profile acceleration runs**
- **Choose an efficient HVL-HDL communication scheme** appropriate for the application at hand
  - For **reactive applications** (with instantaneous transfer of control and data), use **VIF-based inbound and outbound functions and tasks**, or equivalently, **DPI-C export and import functions**
  - For **streaming interfaces** (with producer and consumer decoupled), use **SCEMI transaction pipes** (e.g. audio, video, Ethernet, etc.)
- **Maximize H/W–S/W concurrency**

H/W time

$t_{[HDL]}$

- Emulator executing HDL-side RTL DUT along with BFMs and clock/reset generators
- **Optimize for best possible emulator clock frequency**, which is a function of the combinational logic critical path
- Typical **optimization considerations** to achieve higher emulator frequency and level out capacity are **critical path analysis, clock utilization (inactive edges, edge alignment), h/w parallelism**, etc.
- Use **automated performance and capacity advisor technology**

```
class ahb_seq_item
  extends uvm_sequence_item;
```

```
class ahb_driver
   extends uvm_driver #(ahb_seq_item);

virtual ahb_driver_bfm bfm;

...

virtual task run_phase(uvm_phase phase);
  bfm.wait_for_reset();
  forever begin
    seq_item_port.get_next_item(req);
    bfm.drive(req.we,
              req.addr, req.data, ...);
    seq_item_port.item_done();
   end
endtask

...

endclass
```

Virtual interface pointer from testbench side to HDL-side BFM

Flexible modeling options:
- Separate read/write calls
- Separate address/data transfers (possibly forked in parallel)
- Reactive vs. streaming communication

```
interface ahb_driver_bfm(ahb_if pins);

...

task wait_for_reset();
  ...
endtask

task drive(bit we,
           bit [31:0] addr, data, ...);
  @(posedge pins.clk);
  // Drive request on pin i/f
  ...
endtask

endinterface
```

Time consuming task called directly from the testbench side to the HDL side

The UVM driver wiggles the DUT pins indirectly, no longer directly

```
import ahb_types_pkg::*;

class ahb_monitor extends uvm_monitor;

virtual ahb_monitor_bfm bfm;

...

function void connect_phase(uvm_phase phase);
  ...
  bfm.proxy = this;
endtask

task run_phase(uvm_phase phase);
  bfm.run();
endtask

function void write(ahb_seq_item_s req_s);
  req.from_struct(req_s);
  ap.write(req);
endfunction
...

endclass
```

Assigning the back-pointer in the build or connect phase

One-way outbound function call via back pointer from BFM back to monitor proxy instance in testbench

```
interface ahb_monitor_bfm (ahb_if pins);
import ahb_types_pkg::*;

...

import ahb_pkg::ahb_monitor;
ahb_monitor proxy;

function void run();
  -> start;
endfunction

initial begin
  @(start);
  @(negedge pins.clk);
  monitor_daemon();
end

task monitor_daeom();
  forever begin
    // Sample next request on pin i/f
    ...
    proxy.write(req_s);
  end
endtask

endinterface
```

Package import of back-pointer class type

Time consuming FSM initiated from the testbench side via 0-time function call

```
class pmu_monitor extends uvm_monitor;
...
uvm_event pmu_st_80_e;
function void trigger_pmu_state_80();
  if (pmu_st_80_e == null)
    pmu_st_80_e = m_config.ev_pool.get("pmu_st_80");
  pmu_st_80_e.trigger();
endfunction

endclass

class pmu_sanity_test extends pmu_test_base;
...
uvm_event pmu_st_80_e;
task run();
  // Reset, init and register settings
  ...
  // Power sequences
  ...
  // Ping on PMU state
  if (pmu_st_80_e == null)
    pmu_st_80_e = ev_pool.get("pmu_st_80");
  pmu_st_80_e.wait_ptrigger();
  ...
endtask

endclass
```

PMU monitor receives notification from HDL BFM and triggers corresponding uvm_event

Test waits on uvm_event trigger

HDL BFM waits on PMU state and notifies HVL domain via proxy backpointer trigger

```
interface bfm;
...

pmu_kg::pmu_monitor proxy;

always begin
  wait(hdl_top.cluster_0.core_1.pmu.current_state == 8'h80);
  @(posedge clk);
  proxy.trigger_pmu_state_80();
end

endinterface
```

Mentor Graphics®

SV/UVM Testbench Acceleration Case Studies

	Design Size (gates)	Simulation Time (hrs)	Acceleration Time (secs)	Speed-up
Application Processor	+200M	151	3060	177X
Network Switch	34M	16 ½	240	245X
Graphics Sub-System	8M	86 ½	635	491X
Mobile Display Processor	1.2M	5	46	399X
Memory Controller	1.1M	5	308	60X
Face Recognition Engine	1M	½	6.58	128X
Wireless Multi-Media Sub-System	1M	53	658	288X
Raid Engine Controller I	25M	13	174	268X
Raid Engine Controller II	25M	15.5	327	171X