Generating UVM Testbenches for Fun and Productivity

Rich Edelman and Sanna Zhou

Siemens EDA, Fremont, CA

Siemens, Shanghai, China

SIEMENS

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION CHINA

The Chosen Template Testbench

top.sv – just one interface

```
import uvm_pkg::*;
`include "uvm_macros.svh"

import __VIP__ip_pkg::*;

module top();
  reg clk;
__VIP__interface __VIP__interface_inst(clk);

  initial begin
    run_test();
  end
endmodule
```

ip_pkg.sv

```
package __VIP__ip_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"

  `include "types.svh"

  `include "transaction.svh"
  `include "driver.svh"
  `include "sequencer.svh"
  `include "sequence.svh"
  `include "agent.svh"
  `include "env.svh"
  `include "test.svh"
endpackage
```

transaction.svh

```
typedef enum logic {READ, WRITE} RW_T;
int gid;
int gserial_number;

class __VIP__transaction extends
     uvm_sequence_item;
  `uvm_object_utils(__VIP__transaction)

  bit  [2:0] id; // 0 to 7
  bit [31:0] serial_number;

  rand int delay;

rand RW_T rw;
rand bit [31:0] addr;
rand bit [31:0] data;

constraint values {
  addr > 0; addr < 100;
  data >= 0; data < 8;
  delay > 3; delay < 10;
}

function new(string name =
    "__VIP__transaction");
  super.new(name);
  id = gid++;
  serial_number = gserial_number++;
endfunction

function string convert2string();
  return $sformatf("id:%d %s(%d,%d)#%d",
   id,rw.name(),addr,data,serial_number);
endfunction

function void do_record(uvm_recorder
    recorder);
  super.do_record(recorder);
  `uvm_record_field("id", id);
  `uvm_record_field("serial_number",
     serial_number);
  `uvm_record_field("rw", rw.name());
  `uvm_record_field("addr", addr);
  `uvm_record_field("data", data);
  `uvm_record_field("delay", delay);
  endfunction
endclass
```

Not a generic transaction – but part of this template

sequence.svh

```
class __VIP__seq extends
    uvm_sequence#(__VIP__transaction);
  `uvm_object_utils(__VIP__seq)

  function new(string name = "seq");
    super.new(name);
  endfunction

  __VIP__transaction t;

  task body();
    for (int i = 0; i < 1000; i++) begin
t = __VIP__transaction::type_id::create(…);
      start_item(t);
      if (!t.randomize())
        `uvm_fatal(get_type_name(), " FATAL")
      finish_item(t);
    end
  endtask
endclass
```

driver.svh

```
class __VIP__driver extends
     uvm_driver#(__VIP__transaction);
  `uvm_component_utils(__VIP__driver)

  function new(string name= "__VIP__driver",
     uvm_component parent = null);
    super.new(name, parent);
  endfunction

 function void build_phase(uvm_phase phase);
   super.build_phase(phase);
 endfunction

  __VIP__transaction t;
  int delay;

  task run_phase(uvm_phase phase);
    forever begin
      seq_item_port.get_next_item(t);
      `uvm_info(get_type_name(),
        {"Executing: ", t.convert2string()},
                                  UVM_MEDIUM)
      delay = t.delay;
      #delay;
      seq_item_port.item_done();
    end
  endtask
endclass
```

sequencer.svh

```
class __VIP__sequencer extends
     uvm_sequencer#(__VIP__transaction);
  `uvm_component_utils(__VIP__sequencer)

  function new(string name = "__VIP__sqr",
     uvm_component parent = null);
    super.new(name, parent);
  endfunction

 function void build_phase(uvm_phase phase);
    super.build_phase(phase);
  endfunction
endclass
```

agent.svh

```
class __VIP__agent extends uvm_component;
  `uvm_component_utils(__VIP__agent)

  __VIP__driver d;
  __VIP__sequencer sqr;

  function new(string name = "__VIP__agent",
      uvm_component parent = null);
    super.new(name, parent);
  endfunction

 function void build_phase(uvm_phase phase);
    d = __VIP__driver::type_id::create(
                                  "d", this);
  sqr = __VIP__sequencer::type_id::create(
                                  "sqr", this);
  endfunction

  function void connect_phase(
      uvm_phase phase);
    d.seq_item_port.connect(
                      sqr.seq_item_export);
  endfunction
endclass
```

The Simple Generator

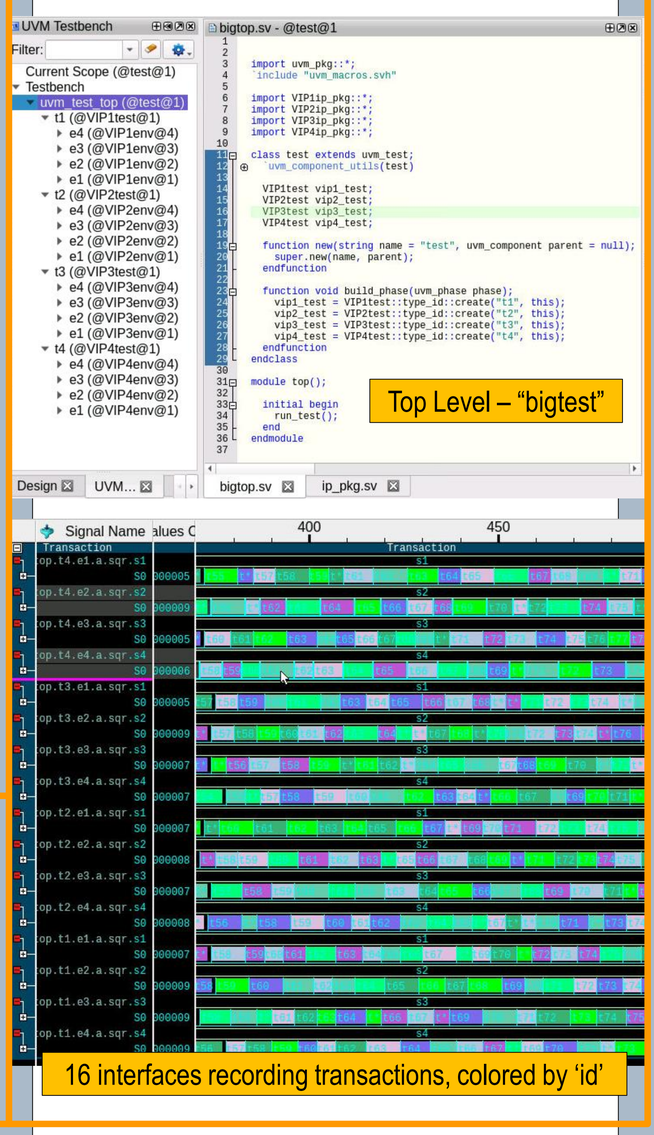
The simple script below is just that. Simple.

The generator runs against a directory full of files. Each file is processed in turn.

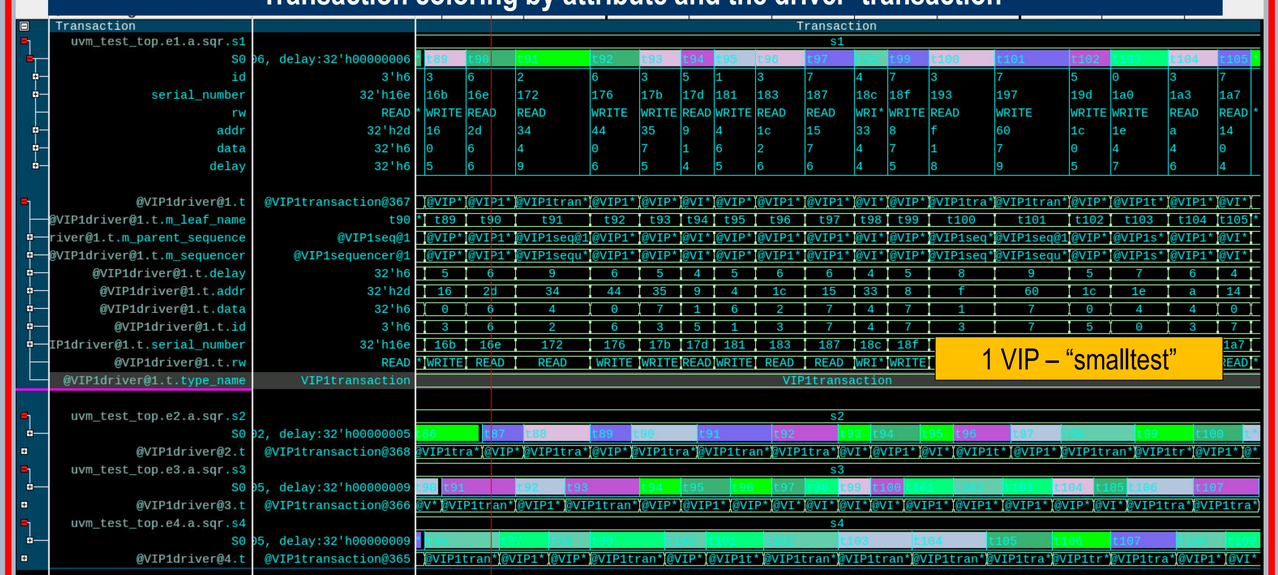
1) The string "__VIP__" in the template is replaced with the actual name of the VIP. For example, "fastbus", or "VIP1".
2) The generated files are put into a new directory. They can be compiled and run directly. But they don't have the programming for the driver or transactions for the target VIP yet.
3) The generator is responsible for generating the "boilerplate" or "plumbing" that can be tedious in the UVM.
4) The user is responsible for customizing the VIP specific generated interface.
5) The template code should be runnable (It makes development and debug are easy)

```
#!/bin/tcsh -f
# Usage: generate VIP1 [VIP2]... [VIPN]
#
foreach VIP ( $* )
  set idir = templates/template1
  set odir = tmp_generated_files/$VIP
  mkdir -p $odir
  foreach if ( $idir/* )
    set of = `basename $if`
    set of =  $odir/$of
    sed -e s/__VIP__/$VIP/g < $if > $of
  end
end
```

A "bigtest"
4 VIPs used, each instantiated 4 times



A test with 1 VIP instantiated 4 times
Transaction coloring by attribute and the driver 'transaction'



Customizations

Interface customization

```
interface VIP1interface(input clk);
  reg valid;
  reg  [2:0] id; // 0 to 7
  reg [31:0] serial_number;
  reg        rw;
  reg  [7:0] addr;
  reg  [7:0] idata;
  reg  [7:0] odata;
endinterface
```

Top Level customization

```
package __VIP__ip_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"

import ip_pkg::*;
import test_pkg::*;

module top();
  reg clk;

  VIP1interface e1_vif(clk);
  dut dut1(
   .clk(e1_vif.clk),
   .valid(e1_vif.valid),
   .id(e1_vif.id),
   .serial_number(e1_vif.serial_number),
   .rw(e1_vif.rw),
   .addr(e1_vif.addr),
   .idata(e1_vif.idata),
   .odata(e1_vif.odata),
  );

  initial begin
    uvm_config_db#(virtual
      VIP1interface)::set(null,
      "uvm_test_top.e1.a", "vif", e1_vif);
    run_test();
  end

  always begin
    #10 clk = 0;
    #10 clk = 1;
  end
endmodule
```

Driver customization

```
typedef enum logic {READ, WRITE} RW_T;
class VIP1driver extends
     uvm_driver#(transaction);
  `uvm_component_utils(VIP1driver)

  virtual VIP1interface vif;
  uvm_analysis_port #(transaction) ap;

  function new(string name = "VIP1driver",
     uvm_component parent = null);
    super.new(name, parent);
  endfunction

  function void build_phase(uvm_phase
     phase);
    super.build_phase(phase);
    ap = new("ap", this);
  endfunction

  VIP1transaction t;
  int delay;

  task send_to_bus(VIP1transaction t);
    `uvm_info(get_type_name(),
      {"Executing: ", t.convert2string()},
                                  UVM_MEDIUM)
    vif.addr = t.addr;
    if (t.rw == READ)
      vif.idata = t.data;
    else
      vif.idata = t.data;
    vif.rw = t.rw;
    vif.id = t.id;
    vif.serial_number = t.serial_number;
    vif.valid = 1;
    @(posedge vif.clk);
    vif.valid = 0;
    @(negedge vif.clk);
  endtask

  task run_phase(uvm_phase phase);
    `uvm_info(get_type_name(),
      "...starting", UVM_MEDIUM)
    forever begin
      seq_item_port.get_next_item(t);
      ap.write(t);
      send_to_bus(t);
      seq_item_port.item_done();
    end
  endtask
endclass
```


Rich Edelman
rich.edelman@siemens.com
Sanna Zhou
xuqing.zhou@siemens.com