Calling All Checkers: Collaboratively Utilizing SVA In UVM Based Simulation

Hui C. K. Zhang
Broadcom/Avago Tech

Abstract

Scoreboard and SVA are two commonly used self-checking mechanisms in verification. However in common practice, they are employed independently without any elaboration except in the interface. This paper presents a new method and practical solution to utilize SVA collaboratively in UVM based simulations. By combining SVA and scoreboard in conjunction with uvm_config_db, control event and informational parameters can be shared and communicated bidirectional between SVA and scoreboards to quicken and ease the debug.

Keyword

Scoreboard; SVA; UVM; uvm_config_db;

Introduction

- Common practice:

  - SVA in formal analysis
  - Scoreboard in simulation
  - SVA in interface & module

  - ***Separate***

- Our method:

  - Combine Scoreboard & SVA In UVM based simulation
  - Check forward & Backward

  - ***Collaborate***

Scoreboard

- *Pro:*
  - Major checker in classed based simulation
  - Predicator : Reference

  Model to expect RTL behavior
  - Comparator: report Pass/Fail information

- *Con:*
  - Only report high level outline of the failure at transaction level
  - Maybe many clock cycles later after the error source
  - Usually get info from monitor (interface signal)
  - Lack inside information of

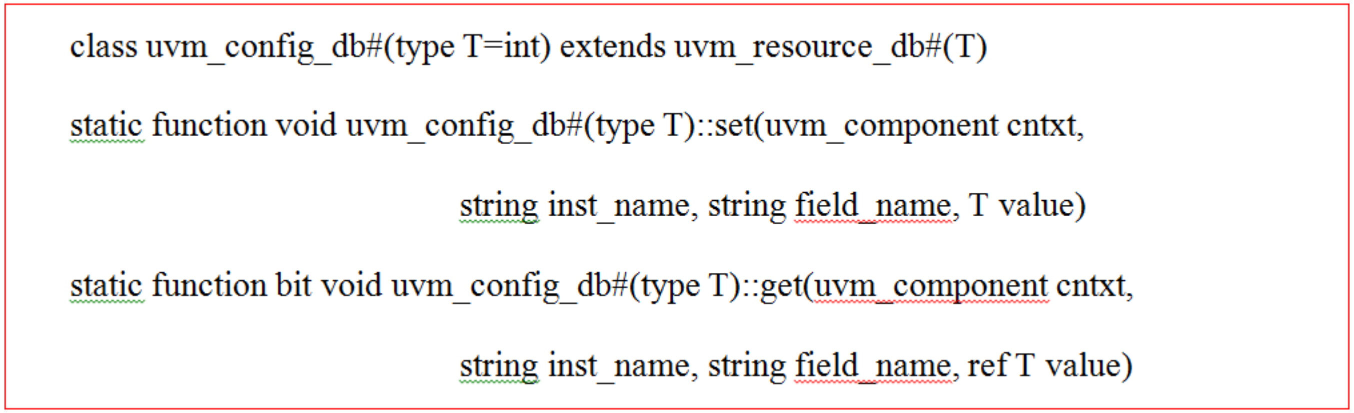
  RTL (internal protocol & FSM)

SVA

- *Pro:*
  - ➢ Used in formal analysis
  - ➢ Used in interface in Class based simulation or binding in TB module
- *Con:*
  - ➢ Concurrent SVA not allowed in class
  - ➢ Simulation performance penalty due to the checking in every clock ticks

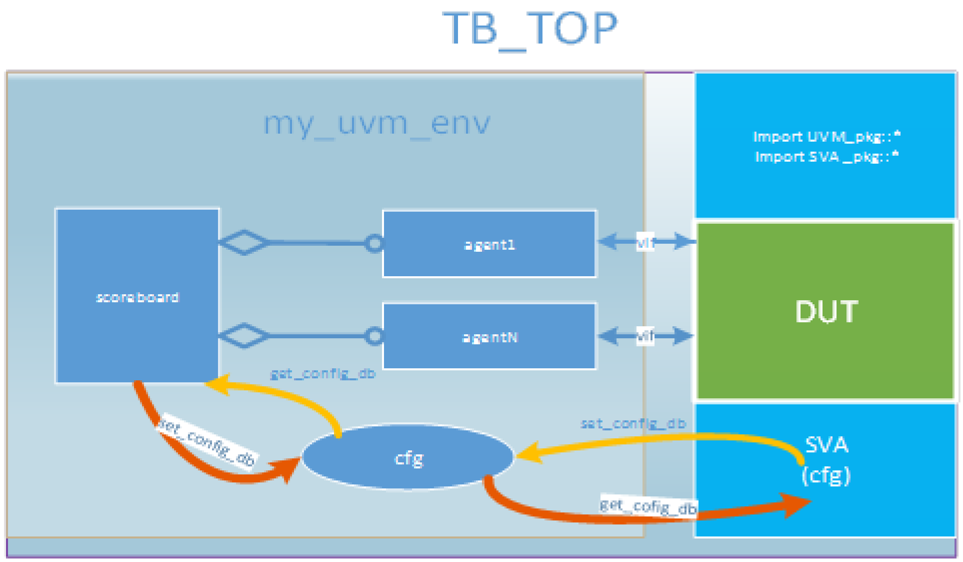
uvm_config_db:
the Connecting Bridge

- Allows objects and variables been stored and retrieved across various verification components within different hierarchical setup in testbench.
- Use lookup strings and a pair of set/get functions
- uvm_config_db function syntax:



```
class uvm_config_db#(type T=int) extends uvm_resource_db#(T)

static function void uvm_config_db#(type T)::set(uvm_component cntxt,

                                     string inst_name, string field_name, T value)

static function bit void uvm_config_db#(type T)::get(uvm_component cntxt,

                                     string inst_name, string field_name, ref T value)
```

Our Solution



Execution Flow

- 1. Run simulation. Record the interest point and parameters reported by scoreboard.
- 2. Create property and Pass the control parameters to SVA through uvm_config_db.
- 3. Run simulation with the same seed with SVA.
- 4. Debug the failure reported by SVA.
- 5. Adjust control parameters adaptively and repeat the step 3-5 if required. (optional for backward case).
- 6. Feedback from SVA can be used to control simulation (Optional).

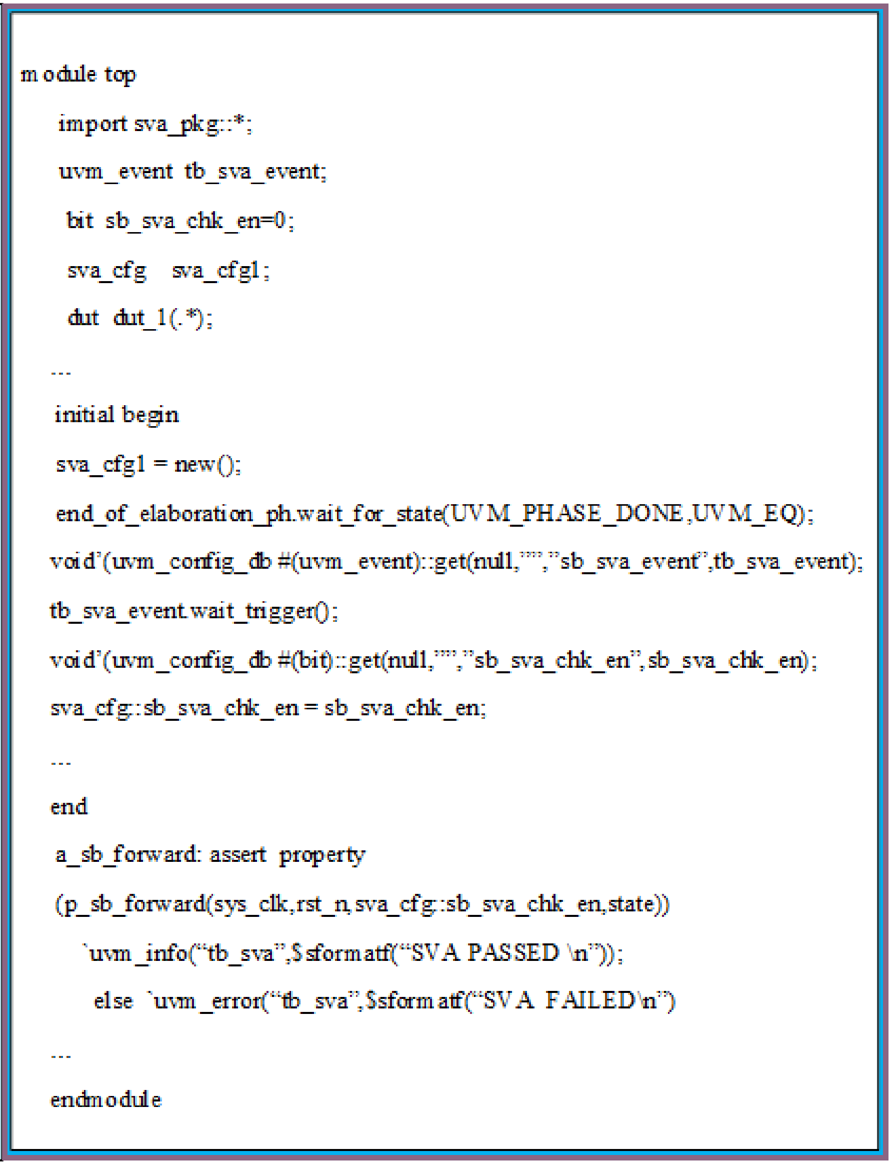
Application

- Forward Method:
  - ➢Look forward in time
  - ➢Suit for error propagation and recovery
  - ➢Check for low power state transition

- Backward method:
  - ➢Backward in time
  - ➢ Back trace the sources of errors
  - ➢ Identify error sources

Application: Forward Flow

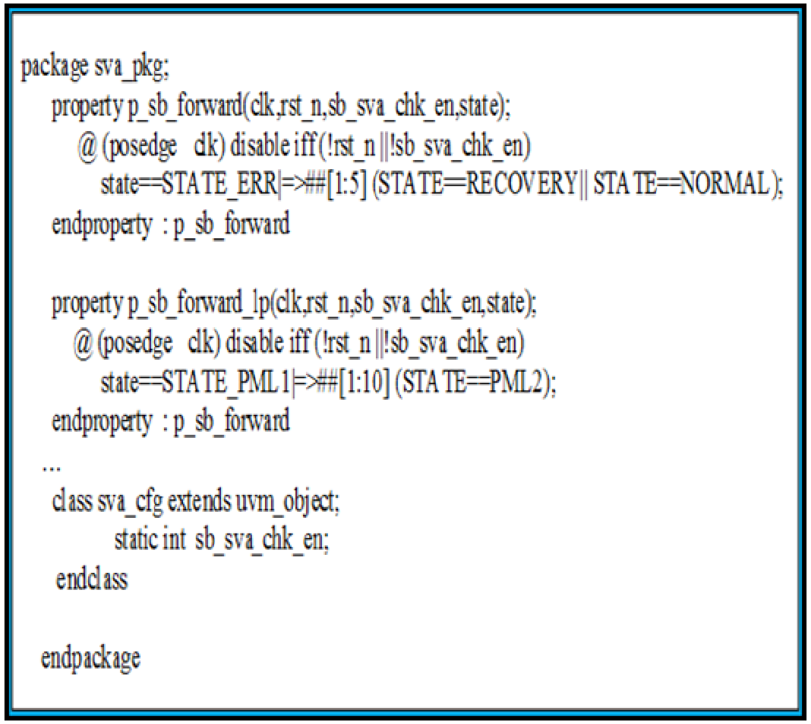
```
  class sb_comparator extends uvm_component;

       uvm_event   sb_sva_event;   bit  sb_sva_chk_en;

        function void build_phase (uvm_phase phase);

           sb_sva_event = new ("sb_sva_event");

           uvm_config_db#(uvm_event)::set(  null,
"“*”,“sb_sva_event”,sb_sva_event)

       endf;unction

  task run_phase(uvm_phase phase);

    forever  begin

     fork

         cmp_exp_fifo.get(exp_tran);

         cmp_act_fifo.get(act_tran);

     join

    compare(exp_tran,act_tran);

  if (comp_error)   begin

   sb_sva_event.trigger();

     uvm_config_db#(bit)::set(  null, “*”,“sb_sva_chk_en”,1);

     …     end

  endtask   … endclass
```



```
module top
   import sva_pkg::*;
   uvm_event  tb_sva_event;
    bit  sb_sva_chk_en=0;
    sva_cfg    sva_cfg1;
    dut  dut_1(.*);
  ---
  initial begin
   sva_cfg1 = new();
   end_of_elaboration_ph.wait_for_state(UVM_PHASE_DONE,UVM_EQ);
  void'(uvm_config_db #(uvm_event)::get(null,"","sb_sva_event",tb_sva_event);
  tb_sva_event.wait_trigger();
  void'(uvm_config_db #(bit)::get(null,"","sb_sva_chk_en",sb_sva_chk_en);
  sva_cfg::sb_sva_chk_en = sb_sva_chk_en;
  ---
  end
   a_sb_forward: assert  property
   (p_sb_forward(sys_clk,rst_n,sva_cfg::sb_sva_chk_en,state))
       `uvm_info(“tb_sva”,$sformatf(“SVA PASSED \n”));
        else  `uvm_error(“tb_sva”,$sformatf(“SVA  FAILED\n”)
  ---
  endmodule
```

Property Package

- Forward Method:



```
package sva_pkg;
   property p_sb_forward(clk,rst_n,sb_sva_chk_en,state);
      @(posedge  clk) disable iff(!rst_n||!sb_sva_chk_en)
         state==STATE_ERR|=>##[1:5] (STATE==RECOVERY|| STATE==NORMAL);
   endproperty : p_sb_forward

   property p_sb_forward_lp(clk,rst_n,sb_sva_chk_en,state);
     @(posedge  clk) disable iff (!rst_n||!sb_sva_chk_en)
         state==STATE_PML1|=>##[1:10] (STATE==PML2);
   endproperty : p_sb_forward
  ...
   class sva_cfg extends uvm_object;
          static int  sb_sva_chk_en;
   endclass

  endpackage
```

- Backward Method:

```
   package sva_pkg;
     property p_sb_backward(clk,rst_n,sb_time0,state);
       @ (posedge   clk) disable iff (!rst_n)
                      ($time> (sb_time0-
therhold_backforward)*peroid)&&(state==STATE_ERR)|=>

$past(state,2)==STATE_PRE2||$past(state,1)==STATE_PRE1;
   endproperty  : p_sb_backward
   …
   class sva_cfg extends uvm_object;
              static int  sb_chk_en;
                       static int sb_time0;
   endclass
   endpackage
```

Conclusion

- A new method and practical solution to utilize SVA collaboratively in UVM based simulations, as by combining SVA with scoreboard in conjunction with uvm_config_db which share & pass control event and informational parameters .
- It can facilitate the debugging process and localize the root cause as well as analyze the preceding and following protocol/sequences efficiently.
- It can minimize the SVA simulation performance penalty .