SVA Encapsulation in UVM
enabling phase and configuration aware assertions

by
Mark Litterick
Verification Consultant
Verilab GmbH, Munich, Germany
Introduction

• To be provided
UVM Verification Environment

- **STIMULUS**
- **CHECKS**
- **COVERAGE**
- **MESSAGES**

**CONFIG**

- **CONFIGURATION**
- **SEQUENCER**
- **DRIVER**
- **MONITOR**
- **ENVIRONMENT**
- **DEVICE UNDER TEST**

**DUT**

**LIB**

**SB**

**DUT**

**TESTBENCH ENV**

**SLAVE UVC**

**RESP AGENT**

3 of (total number of slides)
Distributed UVC Checks

- Types of UVC checks:
  - **signal protocol** and timing
  - **transaction content** and functional correctness
  - **transaction comparison** and relationships
- Each is handled by different component

Concurrent assertions are **not allowed** in SystemVerilog classes

All checks **belong** to UVC

4 of (total number of slides)
Check Configuration & Control

- All checks can be affected by:
  - control knobs (e.g. `checks_enable`)
  - config fields (e.g. `cfg.mode`)
- Configuration object fields can be:
  - constant after build-phase
  - dynamic during run-phase

```cpp
class my_monitor ...
if (checks_enable)
  trans.check_crc();
```

```cpp
class my_monitor ...
  trans.check_parity(cfg.mode);
```

```cpp
class my_test ...
  uvm_config_db#(my_config)::set
      (this,"*","cfg",cfg);
  uvm_config_db#(bit)::set
      (this,"*","checks_enable",0);
```

```cpp
class my_monitor ...
  if (condition)
      cfg.speed_mode = FAST;
```
SVA Configuration & Control

```verbatim
sequence s_fast_transfer;
    REQ #1 !REQ[*1:4] ##0 ACK;
endsequence

sequence s_slow_transfer;
    REQ #1 !REQ[*3:10] ##0 ACK;
endsequence

property p_transfer;
    @(posedge CLK)
        disable iff (!checks_enable)
            REQ |->
                if (cfg_speed_mode == FAST)
                    s_fast_transfer;
                else
                    s_slow_transfer;
endproperty

assert property (p_transfer)
    else $error(“illegal transfer”);
```

- **range operators must be constants**
- **no class variables in concurrent assertions**
- **cfg.speed_mode is not allowed**
- **local variables used for SVA**
- **cfg copied to local variables**
SVA Encapsulation

```vhs
interface my_interface;

// local signal definitions
// modport declarations
// clocking blocks
// bus-functional methods

// support code for SVA
// property definitions
// assertion statements

endinterface
```

SVA checks creates clutter

SVA code is verbose, complex & not related to signal communication

=> isolate and encapsulate SVA
SVA Interface

config db

config_db::get

config_db::set

ENV
UVC
AGENT
M VIF
D VIF

OTHER AGENTS

OTHER UVCS

TESTBENCH MODULE
I/F
SVA

DUT

8 of (total number of slides)
interface my_interface;
    // local signals
    logic    CLK;
    logic    REQ;
    logic    ACK;
    logic [7:0] DATA;
    logic    OTHER;

    // modports, etc.

    // protocol checker
    my_sva_checker
        sva_checker(.*);
endinterface

interface my_sva_checker(
    // signal ports
    input logic    CLK,
    input logic    REQ,
    input logic    ACK,
    input logic [7:0] DATA
);
    // support code
    // properties
    // assertions
endinterface

panel

ports have same name as interface signals (but can be a subset)

required signals well encapsulated

9 of (total number of slides)
SVA Configuration

```verilog
interface my_sva_checker(...);
   // control knobs
   bit checks_enable = 1;
   // config object
   my_config cfg;
   // local variables for SVA
   my_speed_enum cfg_speed_mode;
   int unsigned cfg_max_value;
   bit cfg_data_en;

   // properties and assertions...
   // update local vars from cfg...
endinterface
```

- Interface is not a UVM phased component
- No class variables in concurrent assertions
- allowed in support code
Method API

**push CFG from class environment to SVA interface**

**manual** and only partially **encapsulated** by interfaces
Method API

interface my_sva_checker(...);
...
function void set_config
  (my_config cfg);
  cfg_speed_mode = cfg.speed_mode;
  cfg_max_value = cfg.max_value;
  cfg_data_en = cfg.data_en;
endfunction

function void set_checks_enable
  (bit en);
  checks_enable = en;
endfunction
...
endinterface

user API via VIF

interface my_interface;
...
function void set_config(my_config cfg);
  sva_checker.set_config(cfg);
endfunction

function void set_checks_enable(bit en);
  sva_checker.set_checks_enable(en);
endfunction
...
endinterface

required fields well encapsulated

12 of (total number of slides)
API - Build Configuration

could be done in agent or monitor

class my_monitor extends uvm_monitor;
    ...
    function void end_of_elaboration_phase(...);
    ...
   // set interface config after build
    vif.set_config(cfg);
    vif.set_checks_enable(checks_enable);
endfunction
    ...
endclass

must be after VIF assigned
API – Dynamic Configuration

must be **PASSIVE** component

```verbatim
class my_monitor extends uvm_monitor;
  ...
  task run_phase(...);
  forever begin
    ...
    if (cfg updated) vif.set_config(cfg);
  end
  endtask
endclass
```

**additional work** for monitor
**hard to debug** if done wrong
Phase-Aware Interface

reference CFG from class environment inside SVA interface

automatic and fully encapsulated in interface
Class Inside SVA Interface

locally declared class can see all local variables in interface

class is UVM phased component

interface my_sva_checker(...);
    // declare local variables (cfg, checks_enabled, etc.)
    ...
    class checker_phaser extends uvm_component;
    // use local variables (cfg, checks_enabled, etc.)
    // use UVM phases (build, connect, run, etc.)
    endclass

    // construct unique instance of phaser class
    // (at the top-level under uvm_top)
    checker_phaser m_phase = new($psprintf("%m.m_phase"));
endinterface

unique name required for multiple instances

unique name provides better debug messages
Parallel Phases

SVA interface **class** in **parallel** with main **env** under **uvm_top**

UVM phases run in **parallel**

Potential **race** scenario

cannot rely on ENV top-down build in SVA class

each phase completes, but order of components is unknown
Phase – Build Configuration

```verilog
class checker_phaser extends uvm_component;

... function void end_of_elaboration_phase(...);
...
if (!uvm_config_db#(my_config)::get(this,"","cfg",cfg))
    `uvm_fatal("CFGERR","no my_config cfg in db")

void'(uvm_config_db#(bit)::get
    (this,"","checks_enable",checks_enable));

    cfg_speed_mode = cfg.speed_mode;
    cfg_max_value = cfg.max_value;
    cfg_data_en = cfg.data_en;
endfunction
endclass
```

use `end_of_elaboration_phase` to ensure `env build` and `connect` are complete

cfg must be provided by db

checks _enable may be overloaded by db

copy required cfg fields to local variables for use in SVA
Phase – Dynamic Configuration

always@ and always_comb in interface do not work (cfg is null at start of simulation => use phases)

use run_phase to check for dynamic cfg changes

only required dynamic cfg fields

changes need to be manually copied to local variables (class sees update, SVA not)

forever @* and @(*) do not work (cfg object does not change only fields inside the object)
Conclusion

- To be provided