Lab Guide for course on “Do-it Right Universal Verification Methodology (DR-UVM)”

Pre-requisite:

SystemVerilog &OOP knowledge and simulation know how is expected.

Tool Used:

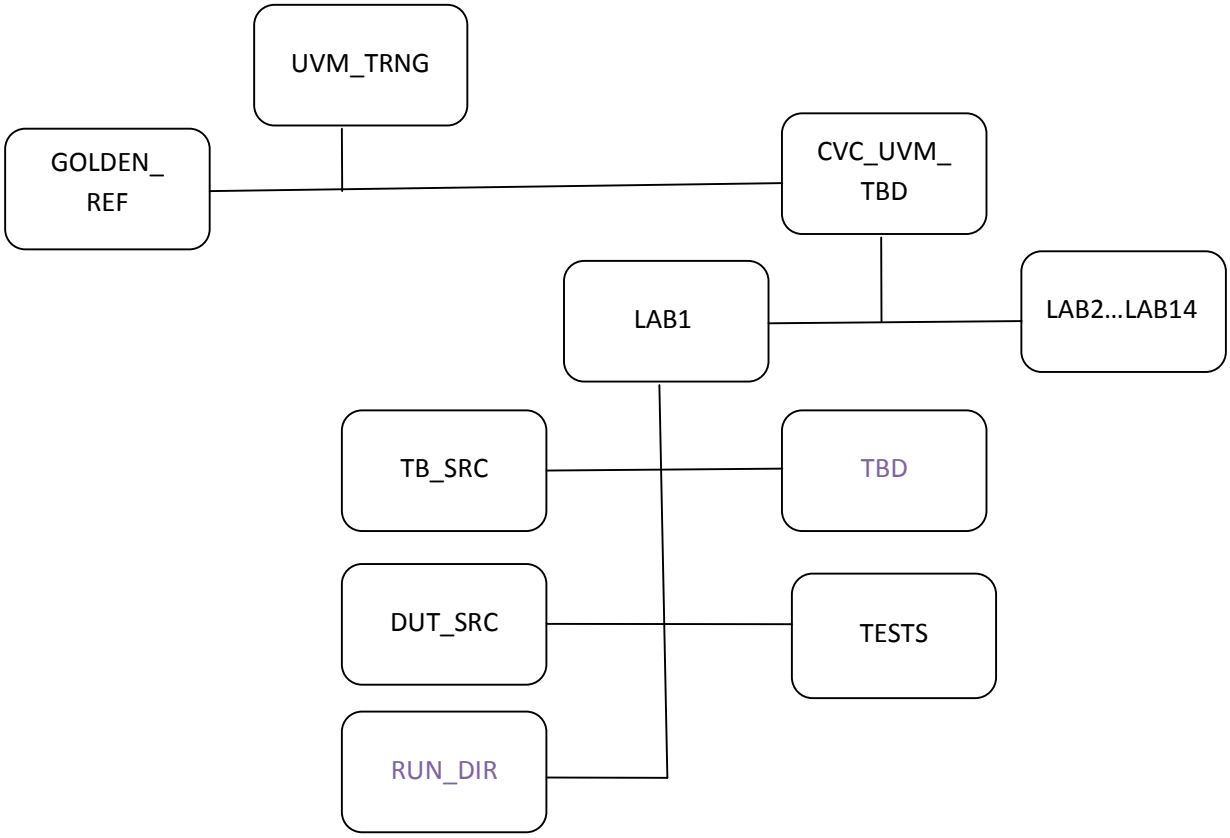
Questa, VCS.

How to run Questa/VCS?

Simplest use model is:

- **qverilog –f flist_uvm (for QUESTA)**
- **vcsi –f flist_uvm (for VCS)**
- Use Makefiles provided in labs

Directory Structure



Overall LAB objectives

LAB1 : UVM reporting features - basic usage.

LAB2 : Declare SV Interface. Capture transaction model using **uvm_*sequence_item*** base class. Build a Driver BFM using **uvm_*driver***. Simulate.

LAB3 : Add a generator/sequencer – basic usage, more details later in another lab.

LAB4 : Add ***s2p_agent*** and ***s2p_env*** to complete a scalable environment.

LAB5 : Create a simple_test with ***uvm_test*** base class. Fairly directed tests, no fanciness yet.

LAB6 : Develop a monitor, use ***analysis_port*** and add a checker via **uvm_*subscriber*** and use the ***analysis_port_imp.***

LAB7 : Implementation of multiple subscribers using **`*uvm_analysis_imp_decl*** macros.

LAB8 : Implementation of multiple subscribers using ***uvm_tlm_analysis_fifo.***

LAB9 : Usage of ***set_config_int*** and ***set_config_obj*** functions.

LAB9 : Usage of ***uvm_config_db*** for passing interface without vif_wrapper.

LAB10: Develop UVM Sequences and concept of ***sequence_library***.

LAB11: Develop virtual Sequences and virtual Sequencers.

LAB12: Factory usage with ***set_type_override**** functions.

LAB13: UVM reporting features - Advanced usage.

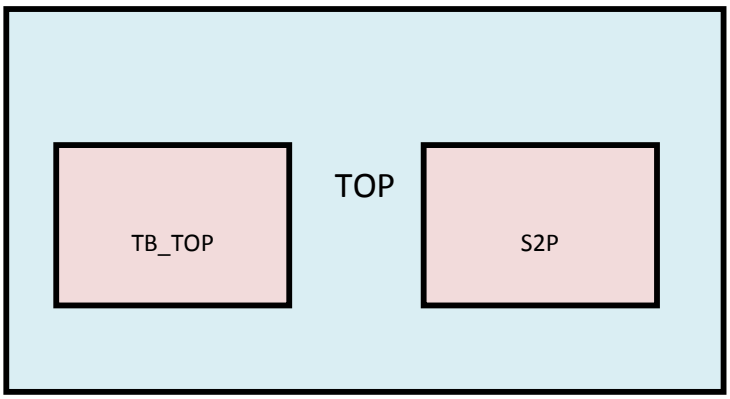
LAB14: Customizing uvm message format by extending ***uvm_report_server***.

General Instructions

- Run all labs inside ***run_dir*** Use *gmake* and *flist* to run your tb
- All lab code shall be written inside ***tbd & tests*** directory. The other directories (***tb_src, dut_src***) are provided to supplement the simulations. You are not expected to fully understand them especially in early labs. At the end of 8th lab, it is expected that you understand all those files having coded most of them yourself.
- In all files under ***tdb & tests*** dir, look for **CVC_UVM_TBD** and complete your labs

DUT Description

Serial to Parallel Converter (S2P)



- Port
  - Input – *clk, rst_n, ser_sop, ser_eop, ser_data, pkt_abort*
  - Output – par_data, par_data_valid
- Parameters
  - PAR_DWIDTH
  - PLD_WIDTH
  - RST_WIDTH
  - DBG_DWIDTH

- Operations
  - Reset
    - Different number of cycles
  - Serial Packet transmission
    - fixed length (= 8 bits)
    - Single packet
    - Multiple packets
  - ERROR packet & Good packet

Lab1

Objective

- use ***uvm_report_info*** for displaying messages

Steps

FILE: ***lab1/tb_src/top.sv***

- Inside the testcase use UVM messaging service

```
`uvm_info("CVC_UVM_TRNG",$psprintf("%s Start of Test" ,
`CVC_PREFIX),UVM_LOW);
```

- Try different options (Hint: use ***vsim +UVM_VERBOSITY=HIGH***).

```
`uvm _info("CVC",$psprintf("%s\n%s%s",`CVC_HDR, `CVC_HDR_LINE,
`CVC_HDR),UVM_MEDIUM);

`uvm _info("CVC_UVM_TRNG",$psprintf("%s End of Test",
`CVC_PREFIX),UVM_HIGH);

`uvm_warning("CVC_UVM_TRNG","Sample Warning");

`uvm_ error("CVC_UVM_TRNG","Sample Error");
```

- BONUS: Use +define+ UVM_REPORT_DISABLE_FILE_LINE to avoid file/line printing (coming from `uvm_info macros via `__FILE__, `__LINE__ macros)

Lab2

Objective

- Declare SV Interface.
- Capture transaction model using ***uvm_sequence_item*** base class.
- Build a Driver BFM using ***uvm_driver***.
- Simulate.

Steps

File: tbd/s2p_if.sv

- Fill in SV Interface file

```
  clocking ser_cb @ (posedge clk);
    output  rst_n,
            dbg_tb_data_in;
    inout   pkt_abort,
            ser_data,
            ser_eop,
            ser_sop;
  endclocking : ser_cb

  clocking par_cb @ (posedge clk);
    input   par_data,
            par_pkt_invalid,
            par_data_valid;
  endclocking : par_cb

  modport ser_drv_mp (clocking ser_cb);
  modport par_if_mp (clocking par_cb);
```

File: tbd/s2p_xactn.sv

- Define s2p_xactn model, extend from ***uvm_sequence_item*** Add `uvm_field* macros

```
class s2p_xactn  extends uvm_sequence_item;


rand logic err_pkt;
// .. more of them, already done for you
`uvm_object_utils_begin(s2p_xactn)
  `uvm_field_int(err_pkt , UVM_ALL_ON)
  `uvm_field_int(no_of_rst, UVM_ALL_ON)
  `uvm_field_int(pkt_len , UVM_ALL_ON)
  `uvm_field_int(pkt_pld, UVM_ALL_ON)
  `uvm_field_int(ipg, UVM_ALL_ON)
  `uvm_field_enum ( s2p_pkt, kind, UVM_ALL_ON )
`uvm_object_utils_end
```

BONUS:

May want to try

```
`uvm_field_int(pkt_len, UVM_ALL_ON | UVM_DEC)
`uvm_field_int(ipg, UVM_ALL_ON | UVM_NOPRINT)
```

File: tbd/s2p_driver.sv

- Define s2p_driver model, extend from ***uvm_driver***
- Add `uvm_component_utils* macros

```
class s2p_driver extends uvm_driver #(s2p_xactn);
  virtual s2p_if.ser_drv_mp vif;
  `uvm_component_utils_begin(s2p_driver)
  `uvm_field_object(item, UVM_ALL_ON)
  `uvm_component_utils_end
  ---------------------------------
Endclass:s2p_driver
task s2p_driver::main_phase(uvm_phase phase);
  forever
    begin
      seq_item_port.get_next_item(item);
      phase.raise_objection(this);
      get_and_drive();
      seq_item_port.item_done();
      phase.drop_objection(this);
    end
endtask : main_phase
task get_and_drive();
    begin
      send_to_dut(item);
    end
endtask : get_and_drive
```

- Review ***reset_phase, send_ser_pkt, main_phase()*** etc. as they are already implemented in tbd/s2p_drvier.sv
- Run the lab with ***run_dir/Makefile***
- BONUS: Make the tasks as “externs”

Lab3

Objective

- LAB3 – Add a generator/sequencer – basic usage, more details later in another lab

Steps

File: lab3/tbd/s2p_sequencer.sv

- Derive it from uvm_sequencer
- Use the uvm_component_utils macro

```
class s2p_sequencer extends uvm_sequencer #(s2p_xactn);
  `uvm_component_utils(s2p_sequencer)
function new(string name,uvm_component parent);
  super.new(name,parent);
  endfunction : new
endclass : s2p_sequencer
```

- Run the code using **run_dir/Makefile**
- There is lot of code that goes around this sequencer to make the test running. If interested, look at tb_src/s2p_agent.sv, s2p_env.sv etc. And tests/rand_test.sv file. You will learn each one of them in next labs, so hold on if you don't follow the whole thing just yet!

Lab4

Objective

- LAB4 – Add ***s2p_agent*** and ***s2p_env*** to complete a scalable environment

Steps

File: lab4/tbd/s2p_agent.sv

- Use base class s2p_agent
- Instantiate driver, sequencer
- Declare an enum to control ACTIVE/PASSIVE
- Use macros.

```
class s2p_agent extends uvm_agent;


uvm_active_passive_enum is_active;

s2p_sequencer sequencer;

s2p_driver driver;


`uvm_component_utils_begin(s2p_agent)

  `uvm_field_enum(uvm_active_passive_enum, is_active,
UVM_ALL_ON)

`uvm_component_utils_end
```

File: lab4/tbd/s2p_agent.sv

- Fill build_phase() with control of is_active field
- Remember in UVM all components are built/constructed via ***type_id::create()*** and not via ***new()***. More on that during factory section

```
virtual function void build_phase (uvm_phase phase);
super.build_phase(phase);
if (!get_config_int("is_active",is_active_int_val))
begin : def_val_for_is_active
this.is_active =   uvm_active_passive_enum'(
                   is_active_int_val);
`uvm_warning(get_name,$sformatf("No override for
   is_active: Using default is_active as:%s" ,
   this.is_active.name));
end : def_val_for_is_active
this.is_active =
   uvm_active_passive_enum'(is_active_int_val);
`uvm_info(get_name(),$sformatf("is_active is set to
   %s",this.is_active.name),UVM_MEDIUM);
monitor=s2p_monitor::type_id::create("monitor",this);
if (is_active == UVM_ACTIVE)
begin
driver=s2p_driver::type_id::create("driver",this);
sequencer=s2p_sequencer::type_id::create("sequencer",this);
end
endfunction : build_phase
```

File: lab4/tbd/s2p_agent.sv

- Now that we have sequencer and driver, let's connect them.
- Sequencer == producer of transactions, sends it via ***seq_item_export***

- Driver == consumer of transactions. Receives via ***seq_item_port*** (Pre-declared in ***uvm_driver***).
- Driver's virtual interface needs to be "assigned"

```
virtual function void connect_phase (uvm_phase phase);
  if (is_active == UVM_ACTIVE)
  begin
    driver.seq_item_port.connect(sequencer.seq_item_export);
    monitor.vif = this.s2p_if_0;
    // CVC_UVM_TBD Review port connection
    this.driver.vif = this.s2p_if_0;
  end
endfunction : connect_phase
```

File: lab4/tbd/s2p_env.sv

```
class s2p_env extends uvm_env;
s2p_agent agent0;
`uvm_component_utils(s2p_env)
function new(string name, uvm_component parent);
   super.new(name,parent);
endfunction : new
virtual function void build_phase();
  super.build_phase();
  //  CVC_UVM_TBD Construct the agent object using factory
pattern agent0=s2p_agent::type_id::create("agent0",this);
  //  CVC_UVM_TBD useset_ config_int to configure agent0
  set_config_int("*","is_active",UVM_ACTIVE);
endfunction : build_phase
endclass : s2p_env
```

File: lab4/tbd/top.sv

- Need to connect agent's virtual interface to PHYSICAL interface. It is done a wrapper object as there is no "set_config_interface" function
- Review tb_src/vif_wrapper.sv file for the wrapper class
- In file tbd/top.sv the vif_wrapper is already instantiated and constructed
- Use ***set_config_object*** to configure the agent's vif_wrapper object

```
module top();

initial

begin

  vif_c = new(s2p_if_0);

  //CVC_UVM_TBD use set_config_object to set
"uvm_test_top.env_0.agent0"

  //s2p_if_wrapper_0 object - this hooks up vir-if to PHY if

   set_config_object("uvm_test_top.env_0.agent0",
"s2p_if_wrapper_0", vif_c, 0);

end

endmodule : top
```

Lab5

Objective

- LAB5 – Create a simple_test with uvm_test base class. Fairly directed tests, no fanciness yet.

Steps

File: lab5/tbd/simple_test.sv

- UVM provides a base class ***uvm_test***
- Derive a simple_test from this class & Add UVM macro

```
class simple_test extends s2p_base_test;

`uvm_component_utils(simple_test)
```

- UVM test also has the *main_phase()* task built-in, use that to add a procedural test case
- Use built-in task ***uvm_top.print_topology*** to see structure of the env built so far
- Do a simple reset, few pkts sending.

```
task simple_test::main_phase();
    uvm_top.print_topology();
    // Do reset
    env_0.agent0.driver.reset_dut(10);
    x0 = new();
    void'(x0.randomize());
    env_0.agent0.driver.send_to_dut(x0);
    #100;
    void'(x0.randomize());
    env_0.agent0.driver.send_to_dut(x0);
    #100;
   `uvm_info(get_name,"End of simulation", UVM_MEDIUM)
  endtask :  main_phase
```

Lab 6

Objective

- Develop a Monitor and use ***analysis_port***
- Add a checker via ***uvm_subscriber*** and use the ***analysis_port_imp***
- Review the *check_it* function provided for this design.

Steps

File: lab6/tbd/s2p_monitor.sv

- UVM provides a base class ***uvm_monitor***
- Derive a s2p_monitor from this class
- Add virtual interface
- Add cur_xactn handle to collect transactions
- Add analysis port named m_aport

```
// CVC_UVM_TBD declare a class s2p_monitor derived from
uvm_monitor

class s2p_monitor extends uvm_monitor;

  string m_name;

  // CVC_UVM_TBD declare virtual -modport

   virtual s2p_if.par_if_mp vif;

  // CVC_UVM_TBD instantiate s2p_xactn as cur_xactn

  s2p_xactn cur_xactn;

  //TLM port for scoreboard communication

  // (implement scoreboard write method if needed)

 // CVC_UVM_TBD instantiate uvm_analysis_port as m_aport

     uvm_analysis_port #(s2p_xactn) m_aport;
```

- // CVC_UVM_TBD Review collect_data method

File: lab6/tbd/s2p_checker.sv

- UVM provides a base class ***uvm_subscriber***
- Declare a class s2p_checker derived from uvm_subscriber
- Implement scoreboard write method
- Instantiate uvm_analysis_imp as m_aport with parameter is s2p_xactn and s2p_checker

```
// CVC_UVM_TBD declare a class s2p_checker derived from
uvm_subscriber

class s2p_checker extends uvm_subscriber #(s2p_xactn);

  string m_name;

  s2p_xactn m_xactn;

  // implement scoreboard write method

  // CVC_UVM_TBD instantiate uvm_analysis_imp as m_aport
with parameter is s2p_xactn and s2p_checker

  uvm_analysis_imp #(s2p_xactn, s2p_checker) m_aport;
```

File: lab6/tbd/s2p_checker.sv

- Implement write method to perform checking
- Use this.check_it routine

```
// CVC_UVM_TBD implement write method

function void s2p_checker:: write(s2p_xactn t);

    this.m_xactn = t;

    `uvm_info ("CHKR", $sformatf("Found 1 xactn %s",

                    m_xactn.sprint), UVM_MEDIUM);

    this.check_it();

  endfunction : write
```

Lab 7

Objective

- Implementation of multiple subscribers using **`*uvm_analysis_imp_decl*** macros.

Steps

- Use lab7_checker_ex_with_2_analysis_imp.

  In s2p_checker.sv

```
  `uvm_analysis_imp_decl(_1)
  `uvm_analysis_imp_decl(_2)
class s2p_checker extends uvm_scoreboard;
  s2p_xactn m_ser_xactn;
  s2p_xactn m_par_xactn;
 `uvm_component_utils_begin(s2p_checker)
    `uvm_field_object(m_ser_xactn, UVM_ALL_ON)
    `uvm_field_object(m_par_xactn, UVM_ALL_ON)
  `uvm_component_utils_end
  uvm_analysis_imp_1 #(s2p_xactn, s2p_checker) m_aport_1;
  uvm_analysis_imp_2 #(s2p_xactn, s2p_checker) m_aport_2;
  bit received_ser_mon_data, received_par_mon_data;
  static int m_x_count = 1;
function new(string name, uvm_component parent);
  super.new(name,parent);
  m_ser_xactn = new();
  m_par_xactn = new();
  m_aport_1 = new("m_aport_1", this);
  m_aport_2 = new("m_aport_2", this);
endfunction
```

In s2p_checker.sv

```
function void s2p_checker::write_1(s2p_xactn t);
  this.m_ser_xactn = t;
  m_ser_xactn.print();
  received_ser_mon_data = 1;
endfunction : write_1
function void s2p_checker::write_2(s2p_xactn t);
  this.m_par_xactn = t;
  m_par_xactn.print();
  received_par_mon_data = 1;
  if (received_par_mon_data && received_ser_mon_data)
      this.check_it();
  endfunction : write_2
```

In s2p_agent.sv

```
function void s2p_agent::connect_phase(uvm_phase phase);
  if (is_active == UVM_ACTIVE)
  begin
    this.driver.seq_item_port.connect(sequencer.
seq_item_export);
    this.driver.vif = this.s2p_if_0;
  end
this.ser_monitor.m_ser_aport.connect(this.m_checker.m_apor
t_1);
this.par_monitor.m_par_aport.connect(this.m_checker.m_apor
t_2);
endfunction : connect_phase
```

Lab 8

Objective

- Implementation of multiple subscribers using ***uvm_tlm_analysis_fifo***

Steps

- Use lab8_checker_ex_with_analysis_fifo

  In s2p_checker.sv

```
class s2p_checker extends uvm_scoreboard;
      s2p_xactn m_ser_xactn;
      s2p_xactn m_par_xactn;
      `uvm_component_utils_begin(s2p_checker)
      `uvm_field_object(m_ser_xactn, UVM_ALL_ON)
      `uvm_field_object(m_par_xactn, UVM_ALL_ON)
    `uvm_component_utils_end

    uvm_tlm_analysis_fifo #(s2p_xactn) m_aport_1;
    uvm_tlm_analysis_fifo #(s2p_xactn) m_aport_2;
function new(string name, uvm_component parent);
  super.new(name,parent);
  m_ser_xactn = new();
  m_par_xactn = new();
  m_aport_1 = new("m_aport_1", this);
  m_aport_2 = new("m_aport_2", this);
endfunction : new
extern virtual task main_phase(uvm_phase phase);
endclass : s2p_checker
```

In s2p_checker.sv

```
task s2p_checker::main_phase(uvm_phase phase);

    `uvm_info(get_name(),$sformatf(":main_phasening
..\n"),UVM_MEDIUM)

    forever

    begin

      this.m_aport_1.get(m_ser_xactn);

      this.m_aport_2.get(m_par_xactn);

      `uvm_info ("CHKR", $sformatf("Found 1 xactn from
SER_MON \n"), UVM_MEDIUM);

      m_ser_xactn.print();

      `uvm_info ("CHKR", $sformatf("Found 1 xactn from
PAR_MON \n"), UVM_MEDIUM);

      m_par_xactn.print();

    end

endtask : main_phase
```

In s2p_agent.sv

```
function void s2p_agent::connect_phase(uvm_phase phase);

-------------------------------------

this.ser_monitor.m_ser_aport.connect(this.m_checker.m_apor
t_1.analysis_export);

this.par_monitor.m_par_aport.connect(this.m_checker.m_apor
t_2.analysis_export);

endfunction : connect_phase
```

Lab 9

Objective

- Understand configuration mechanisms in UVM
- Use set_config_int, get_config_int
- Use set_config_object for interface passing

Steps

File: lab9_set_cfg_int/src/set_cfg_int.sv

- Look for CVC_UVM_TBD and add at relevant place a call to ***get_config_int***
- In this example, a field to control functional coverage is shown. In ***s2p_agent*** class, we use ***get_config_int***

```
function void s2p_agent::connect_phase(uvm_phase phase);
  bit tmp;
  // CVC_UVM_TBD use get_config_int to retrieve value for
field coverage
    tmp = get_config_int(.field_name("coverage"),
```

- At test level we can use ***set_config_int*** to enable/disable coverage

```
function void simple_test::build_phase(uvm_phase phase);
  env_0 = s2p_env::type_id::create("env_0",this);
 //CVC_UVM_TBD Use set_config_int to set coveargе field to 1
    set_config_int(.inst_name("*"),
                   .field_name("coverage"),
                   .value(1));
endfunction : build_phase
```

In Lab9 we also have an example for using ***set_config_object*** to demonstrate usage of ***vif_wrapper*** to pass virtual interface across testbench components.

Steps

File: lab9_set_cfg_obj/src/vif_wrapper.sv

- Review this file to understand the simple idea behind wrapping a virtual interface inside a class (derived from ***uvm_object***)

File: lab9set_cfg_obj/tbd/set_config_obj.sv

- At Driver level a virtual interface is used
- At the env level, the driver is built, connected – and at that point it requires a virtual interface. Use ***get_config_object*** to retrieve a virtual_interface from config-table.

```
function void env::build_phase(uvm_phase phase);

  uvm_object dummy;

  vif_wrapper vif_wrapper_0;

  bit get_cfg_success;

   this.drvr_inst = driver::type_id::create("driver_inst",
this);

   // CVC_UVM_TBD

   // Use get_config_object for retrieving VIF wrapper

   get_cfg_success = get_config_object(

                     .field_name("vif_wrapper_0"),

                         .value(dummy), .clone(0));

endfunction : build_phase
```

- At top level a physical interface is instantiated inside a ***module***

At the env level, the driver is built, connected – and at that point it requires a virtual interface. Use ***get_config_object***

```
 module top;
simple_if p_if(.*); // physical interface
 vif_wrapper vif_wrap_0;     // interface class
env e;
 initial begin
   vif_wrap_0 = new;
   vif_wrap_0.set_vif(p_if);
   e = new("env");
   // CVC_UVM_TBD
   // Use set_config_object for VIF wrapper
   set_config_object(.inst_name("*"),
                     .field_name("vif_wrapper_0"),
                     .value(vif_wrap_0),
                     .clone(0));
   run_test();
 end
endmodule : top
```

Lab 9

Objective

- Usage of ***uvm_config_db*** for passing interface without vif_wrapper.

Steps

- Use lab9_uvm_config_db.

In uvm_config_db.sv

```
module top;

----------------------------------

initial begin

   e = new("env");

   // CVC_UVM_TBD

   // Use uvm_config_db set() for VIF

   uvm_config_db #(virtual simple_if)::set(.cntxt(null),
.inst_name("*"),   .field_name ("vif"),

                                          .value(p_if));

   run_test();

 end

endmodule : top

function void env::build_phase(uvm_phase phase);

  uvm_object dummy;

  bit get_cfg_success;

   this.drvr_inst = driver::type_id::create("driver_inst",
this);

   get_cfg_success = uvm_config_db #(virtual
simple_if)::get(.cntxt(this),.inst_name(""),
.field_name("vif"),.value(vif));

-------------------------------------

endfunction:build_phase
```

Lab 10

Objective

- Develop UVM Sequences and concept of sequence_library

Steps

- Add 2 sequences in tests directory

  1. File: tests/s2p_seq_with_rst.sv

```
    // CVC_UVM_TBD

  // Implement body() with few `uvm_do calls


 task s2p_seq1::body();

 `uvm_info(get_name(),$sformatf(":Sequence Running
....\n"),UVM_MEDIUM)

  `uvm_do_with(req, {this.kind == SEND_PKT;} ) //this line
sends the transaction

  `uvm_do_with(req, {this.kind == SEND_PKT;} ) //this line
sends the transaction

  `uvm_do_with(req, {this.kind == SEND_PKT;} ) //this line
sends the transaction

  `uvm_info(get_name(),$sformatf(":Sequence Is Complete
....\n"),UVM_LOW)

 endtask : body
```

- Add another sequences in tests directory

  1. s2p_more_sequences.sv

- Create a Sequence Library and include all these sequences

  tests/s2p_seq_lib.sv

- Add tests to call these sequences in your order of choice/requirement (as per test intent)

  Files:

  tests/rand_test.sv

  tests/more_seq_test.sv

```
class rand_test extends s2p_base_test;

  `uvm_component_utils(rand_test)

    // CVC_UVM_TBD instantiate seq of choice (as developed in
prev files)

  s2p_seq1 s2p_seq01;

//..

task rand_test::main_phase(uvm_phase phase);

          phase.raise_objection(this);

    // CVC_UVM_TBD

// Create sequence object via factory

     s2p_seq01 = s2p_seq1::type_id::create("s2p_seq");

    // CVC_UVM_TBD

  // Start the sequence in sequencer

     s2p_seq01.start(env_0.agent0.sequencer);
```

- Add these tests to top.sv for compilation (can also use a tests_inc.svh if you wish)
- Finally use +UVM_TESTNAME=test_name to select via Makefile

Lab 11

Objective

- Virtual sequencer, sequences
- Will use AHB & OCP interfaces
- Demo code, not full VIP
- Each have their own UVC, agent, seqr, sequence etc.
- Will focus on creating a vir-seq and vir-sqr across these 2 interfaces

Steps

- Under lab11_vseq/ahb_vip, review the code quickly
- Under lab11_vseq/ocp_vip, review the code quickly
- File: ***ahb_ocp_vir_sequencer.sv*** add ahb & ocp sub-sequencers

```
class ahb_ocp_vir_sequencer extends uvm_sequencer
#(ahb_ocp_vir_seq);

  `uvm_component_utils(ahb_ocp_vir_sequencer)

  // CVC_UVM_TBD

  // Add a handle for  ahb_sequencer ahb_sqr_0;

  ahb_sequencer ahb_sqr_0;

  // CVC_UVM_TBD

  // Add a handle for  ocp_sequencer ocp_sqr_0;

  ocp_sequencer ocp_sqr_0;
```

- File: ***ahb_ocp_vir_sqr_env_top.sv*** add ahb & ocp sub-sequencers

```
class ahb_ocp_vir_sequencer extends uvm_sequencer
#(ahb_ocp_vir_seq);

  `uvm_component_utils(ahb_ocp_vir_sequencer)

  // CVC_UVM_TBD

  // Add a handle for  ahb_sequencer ahb_sqr_0;

  ahb_sequencer ahb_sqr_0;

  // CVC_UVM_TBD

  // Add a handle for  ocp_sequencer ocp_sqr_0;

  ocp_sequencer ocp_sqr_0;
```

Lab 12

Objective

- Factory usage with set_type_override* functions
- Add a test to constrain the payload pattern

  File: tests/factory_test.sv
- In the build(), use:

  // Factory override code

  uvm_report_info("FACTORY", "Overriding s2p_xactn with pattern testcase");

  factory.set_type_override_by_type (s2p_xactn::get_type (),

  s2p_xactn_with_pld_pattern::get_type ());

  set_config_int("env_0.agent0.sequencer","count",4);
- Create tests/all_tests.svh, include all tests (This step is DONE for you)
- Add the all_tests.svh to top.sv for compilation
- Finally use +UVM_TESTNAME=test_name to select via Makefile
- Review Makefile targets: ftest

In tests/factory_test.sv

```
class s2p_xactn_with_pld_pattern extends s2p_xactn;

 constraint cst_pld_aa { this.pkt_pld inside {'haa, 'hbb,
'hcc, 'hdd, 'hee, 'hff};}

 //Use the macro in a class to implement factory
registration along with other

 //utilities (create, get_type_name).

 `uvm_object_utils(s2p_xactn_with_pld_pattern)

endclass : s2p_xactn_with_pld_pattern
```

In tests/factory_test.sv

```
class factory_test extends s2p_base_test;

`uvm_component_utils(factory_test)

  s2p_xactn x0;

function new(string name, uvm_component parent);

   super.new(name,parent);

endfunction : new

virtual function void build_phase(uvm_phase phase);

  super.build_phase(phase);

       `uvm_info("FACTORY","Overriding s2p_xactn with
pattern testcase",UVM_LOW)

  factory.set_type_override_by_type(s2p_xactn::get_type(),
s2p_xactn_with_pld_pattern::get_type());

//set_config_string("env_0.agent0.sequencer","default_sequ
ence","s2p_seq1");

uvm_config_db#(uvm_object_wrapper)::set(this,"env_0.agent0
.sequencer.main_phase","default_sequence",s2p_seq1::type_i
d::get() );

endfunction : build_phase

task main_phase(uvm_phase phase);

    phase.raise_objection(this);

    #1500;

    `uvm_info("FACTORY","User activated end of
simulation",UVM_LOW)

    phase.drop_objection(this);

endtask : main_phase

endclass : factory_test
```

Lab 13

Objective

- UVM reporting features - Advanced usage

Steps

- Use lab13_advanced_reporting.
- Send all messages from s2p_driver to LOG file

  In s2p_driver.sv

```
  function void s2p_driver::end_of_elaboration_phase
                  (uvm_phase phase);
    UVM_FILE f = $fopen({get_name(),".log"},"w");
    set_report_default_file(f);
    // CVC_UVM_TBD set_report_severity_action for INFOS
and WARNINGS
    set_report_severity_action(UVM_INFO,UVM_LOG);
    set_report_severity_action(UVM_WARNING,UVM_LOG);
    set_report_severity_action(UVM_ERROR,UVM_LOG);
    set_report_severity_action(UVM_FATAL,UVM_EXIT |
UVM_DISPLAY |UVM_LOG);
  endfunction:end_of_elaboration_phase
```

Lab 14

Objective

- Customizing uvm message format by extending ***uvm_report_server***

Steps

- Use lab14_msg_formater

In cvc_msg_formatter.sv

```
class cvc_msg_formatter extends uvm_report_server;

 virtual function string compose_message( uvm_severity
severity, string name,  string id,  string message, string
filename, int line );

  uvm_severity_type severity_type =
uvm_severity_type'(severity);

  return $psprintf("%0t | %0s | [%0s] | %0s",$time,
severity_type.name(), id, message );

  endfunction: compose_message

  endclass : cvc_msg_formatter
```

In s2p_env.sv

```
function void s2p_env::start_of_simulation_phase(uvm_phase
phase);

  cvc_msg_formatter my_server = new();

  uvm_report_server::set_server(my_server);

endfunction: start_of_simulation_phase
```