Running internal testbenches and regression test

Development flow

1. Create testbench VHDL file and add necessary generics (including GC_TEST) to the generics list.
    a. Note: if several test cases are located in the same sequencer, a if-elsif structure is required for determining which test to run (see section **Multiple testcases in sequencer**).
2. Create the compile_dependencies.do with compile dependencies (UVVM Util, UVVM FW, UVVM Scoreboard + any others)
3. Create compile_tb.do for compiling test harness and testbench
4. Create run_simulation.do with methods for receiving and setting testbech generics (see file example in section **Testbench**).
5. Create sim.py simulation script (see example in section **Setup**)
    a. define tests
    b. define generics
    c. set library name
    d. set testbench name
    e. compile and run
6. The script testbench.py will scan the UVVM testbench output and report passing/failing test(s)

Testbench.py / sim.py arguments

The arguments passed to the sim.py script are forwarded to the testbench.py script where they are decoded and determine how the simulations are run:

- sim.py -v *-- verbose output*
- sim.py -vsim *-- Modelsim simulator (or -modelsim)*
- sim.py -aldec *-- Riviera Pro (or -rivierapro)*
- sim.py -gui *-- Gui simulator (or -g)*
- sim.py *-- No arguments: Modelsim simulator, no simulation output - only test result.*

Testbench requirements

The testbench has to include the following generic:

```
-- Test bench entity
entity internal_vvc_tb is
  generic (
    GC_TEST : string := "UVVM"
    );
end entity;
```

but can include several generics, e.g.

```
entity axistream_vvc_simple_tb is
  generic (
    GC_TEST               : string  := "UVVM";
    GC_DATA_WIDTH         : natural := 32;   -- number of bits in the AXI-Stream IF data vector
    GC_USER_WIDTH         : natural := 1;    -- number of bits in the AXI-Stream IF tuser vector
    GC_ID_WIDTH           : natural := 1;    -- number of bits in AXI-Stream IF tID
    GC_DEST_WIDTH         : natural := 1;    -- number of bits in AXI-Stream IF tDEST
    GC_INCLUDE_TUSER      : boolean := true; -- If tuser, tstrb, tid, tdest is included in DUT's AXI interface
    GC_USE_SETUP_AND_HOLD : boolean := false -- use setup and hold times to synchronise the BFM
  );
end entity;
```

and will have to match the run_simulation.do generic parameters, e.g.

```
set library 0
set testbench 0
set run_test 0
set data_width 0
set user_width 0
set id_width 0
set dest_width 0
set include_tuser 0
set use_setup_and_hold 0

if { [info exists 1] } {
  set library "$1"
  unset 1
}

if { [info exists 2] } {
  set testbench "$2"
  unset 2
}

if { [info exists 3] } {
  set run_test "$3"
  unset 3
}

if { [info exists 4] } {
  set data_width "$4"
  unset 4
}

if { [info exists 5] } {
  set user_width "$5"
  unset 5
}

if { [info exists 6] } {
  set id_width "$6"
  unset 6
}

if { [info exists 7] } {
  set dest_width "$7"
  unset 7
}

if { [info exists 8] } {
  if {$8 == 1} {
    set include_tuser "true"
  } else {
    set include_tuser "false"
  }
  unset 8
}

if { [info exists 9] } {
  if {$9 == 1} {
    set use_setup_and_hold "true"
  } else {
    set use_setup_and_hold "false"
  }
  unset 9
}

vsim -gGC_TEST=$run_test -gGC_DATA_WIDTH=$data_width -gGC_USER_WIDTH=$user_width -gGC_ID_WIDTH=$id_width -
gGC_DEST_WIDTH=$dest_width -gGC_INCLUDE_TUSER=$include_tuser -gGC_USE_SETUP_AND_HOLD=$use_setup_and_hold
$library.$testbench

run -all
```

Setup

The following files are required for running a testbench:

- compile_dependencies.do
- compile_tb.do
- run_simulation.do
- sim.py

Example compile_dependencies.do

```
#================================================================================================================================
# Copyright 2020 Bitvis
# Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in
compliance with the License.
# You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 and in the provided
LICENSE.TXT.
#
# Unless required by applicable law or agreed to in writing, software distributed under the License is
distributed on
# an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and limitations under the License.
#================================================================================================================================
# Note : Any functionality not explicitly described in the documentation is subject to change at any time
#--------------------------------------------------------------------------------------------------------------------------------
-----------------

#---------------------------------------------------------------------
# Call compile scripts from dependent libraries
#---------------------------------------------------------------------
set root_path "../.."
do $root_path/script/compile_src.do $root_path/uvvm_util $root_path/uvvm_util/sim
do $root_path/script/compile_src.do $root_path/uvvm_vvc_framework $root_path/uvvm_vvc_framework/sim
do $root_path/script/compile_src.do $root_path/bitvis_vip_scoreboard $root_path/bitvis_vip_scoreboard/sim
```

Example compile_tb.do

```
set lib_name "bitvis_vip_sbi"

if { [info exists ::env(SIMULATOR)] } {
  set simulator $::env(SIMULATOR)
  puts "Simulator: $simulator"

  if [string equal $simulator "MODELSIM"] {
    set compdirectives "-quiet -suppress 1346,1236,1090 -2008 -work $lib_name"
  } elseif [string equal $simulator "RIVIERAPRO"] {
    set compdirectives "-2008 -nowarn COMP96_0564 -nowarn COMP96_0048 -dbg -work $lib_name"
  } else {
    puts "No simulator! Trying with modelsim"
    quietly set compdirectives "-quiet -suppress 1346,1236 -2008 -work $lib_name"
  }
} else {
  puts "No simulator! Trying with modelsim"
  quietly set compdirectives "-quiet -suppress 1346,1236 -2008 -work $lib_name"
}

#------------------------------------------------------
# Compile tb files
#------------------------------------------------------
set root_path "../.."
set tb_path "$root_path/bitvis_vip_sbi/internal_tb"
echo "\n\n\n=== Compiling TB\n"

echo "eval vcom  $compdirectives  $tb_path/sbi_fifo.vhd"
eval vcom  $compdirectives  $tb_path/sbi_fifo.vhd

echo "eval vcom  $compdirectives  $tb_path/sbi_slave.vhd"
eval vcom  $compdirectives  $tb_path/sbi_slave.vhd

echo "eval vcom  $compdirectives  $tb_path/sbi_tb_multi_cycle_read.vhd"
eval vcom  $compdirectives  $tb_path/sbi_tb_multi_cycle_read.vhd

echo "eval vcom  $compdirectives  $tb_path/sbi_th.vhd"
eval vcom  $compdirectives  $tb_path/sbi_th.vhd

echo "eval vcom  $compdirectives  $tb_path/sbi_tb.vhd"
eval vcom  $compdirectives  $tb_path/sbi_tb.vhd
```

Example run_simulation.do

```
# Unless required by applicable law or agreed to in writing, software distributed under the License is
distributed on
# an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and limitations under the License.
#================================================================================================================
=================
# Note : Any functionality not explicitly described in the documentation is subject to change at any time
#----------------------------------------------------------------------------------------------------------------
-----------------

set library 0
set testbench 0
set run_test 0

if { [info exists 1] } {
  set library "$1"
  unset 1
}

if { [info exists 2] } {
  set testbench "$2"
  unset 2
}

if { [info exists 3] } {
  set run_test "$3"
  unset 3
}

vsim -gGC_TEST=$run_test $library.$testbench

run -all
```

Example - sim.py

Note! *The testbench.py file has to be included in every sim.py script by reference as given here.*

```
#================================================================================================================
=======
# Copyright (c) 2018 by Bitvis AS. All rights reserved.
# You should have received a copy of the license file containing the MIT License (see LICENSE.TXT), if not,
# contact Bitvis AS <support@bitvis.no>.
#
# UVVM AND ANY PART THEREOF ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT.
# IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
# WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH UVVM OR THE USE
OR
# OTHER DEALINGS IN UVVM.
#================================================================================================================
=======
from os.path import join, dirname
from itertools import product
import os, sys, subprocess

sys.path.append("../../script")
from testbench import Testbench

# Counters
num_tests_run =0
num_failing_tests =0

#============================================================================================
#
# Define tests and run - user to edit this
#
#============================================================================================

# Create testbench configuration with TB generics
def create_config(modes, data_widths, data_array_widths):
 config = []
 for mode, data_width, data_array_width inproduct(modes, data_widths, data_array_widths):
  config.append(str(mode) +' '+str(data_width) +' '+str(data_array_width))
```

```
return config


def main(argv):
 global num_failing_tests
 tests = []

 tb =Testbench()
 tb.set_library("bitvis_vip_sbi")
 tb.check_arguments(argv)

# Compile VIP, dependencies, DUTs, TBs etc
 tb.compile()

# Define tests
 tests = [ "simple_write_and_check",
           "simple_write_and_read",
           "test_of_poll_until",
           "extended_write_and_read",
           "read_of_previous_value",
           "read_of_executor_status_and_inter_bfm_delay",
           "distribution_of_vvc_commands",
           "vvc_broadcast_test",
           "vvc_setup_and_hold_time_test"]

# Setup testbench and run
 tb.set_tb_name("sbi_tb")
 tb.add_tests(tests)
 tb.run_simulation()

# Setup testbench and run
 tb.set_tb_name("sbi_tb_multi_cycle_read")
 tb.add_test("fixed_wait_read_test")
 tb.run_simulation()

# Print simulation results
 tb.print_statistics()

# Read number of failing tests for return value
 num_failing_tests = tb.get_num_failing_tests()


if __name__ =="__main__":
 main(sys.argv)
 # Return number of failing tests to caller
 sys.exit(num_failing_tests)
```

Note: sim.py script will return the number of failing tests, and this is used by the run_regression.py script for presenting the result of the regression test.

Multiple testcases in sequencer

When several testcases are located in the same testbench sequencer, the testcase name will be passed to GC_TEST generic ("UVVM" is default with only one testcase in sequencer). The sequencer will require an ***if-elsif*** structure to determine which testcase to run:

```
-- Test case entity
entity sbi_tb is
  generic (
    GC_TEST : string := "UVVM"
    );
end entity;

-- Test case architecture
architecture func of sbi_tb is

......

      ------------------------------------------------------------------------------------------
      -- Verifying
      ------------------------------------------------------------------------------------------
    if GC_TEST = "simple_write_and_check" then
      log(ID_LOG_HDR, "Test of simple write and check", C_SCOPE);
      --================================================================================

      log("Write with both interfaces");
      sbi_write(SBI_VVCT,1, C_ADDR_FIFO_PUT, x"AA", "Write PUT on FIFO 1");
      sbi_write(SBI_VVCT,2, C_ADDR_FIFO_PUT, x"FF", "Write PUT on FIFO 2");
      await_completion(SBI_VVCT,1, 16 ns, "Await execution");

      log("Read and check with both interfaces");
      sbi_check(SBI_VVCT,1, C_ADDR_FIFO_GET, x"FF", "Check GET data on FIFO 2", ERROR);
      sbi_check(SBI_VVCT,2, C_ADDR_FIFO_GET, x"AA", "Check GET data on FIFO 1", ERROR);
      await_completion(SBI_VVCT,1, 16 ns, "Await execution");

    elsif GC_TEST = "simple_write_and_read" then
      log(ID_LOG_HDR, "Test of simple write and read", C_SCOPE);
      --================================================================================
      -- Write to FIFO

.....

    else
      alert(tb_error, "Unsupported test: " & GC_TEST);
    end if;

    ------------------------------------------------------------------------------
    -- Ending the simulation
    ------------------------------------------------------------------------------
    wait for 1000 ns;             -- to allow some time for completion
    report_alert_counters(FINAL); -- Report final counters and print conclusion for simulation (Success/Fail)
    log(ID_LOG_HDR, "SIMULATION COMPLETED", C_SCOPE);

    -- Finish the simulation
    std.env.stop;
    wait;  -- to stop completely
```

See section **Example - sim.py** for setting of testcases and other generics.

Example output

Example of running a sim.py simulation script without any arguments and with 11 defined testcases. Simulations are run from bitvis_vip_sbi/sim folder:

```
$ py ../script/maintenance_script/sim.py
Running with simulator : MODELSIM
Compiling dependencies...
Compiling src...
Compiling testbench...
(1/10) [sbi_tb] test=simple_write_and_check, result=PASS
(2/10) [sbi_tb] test=simple_write_and_read, result=PASS
(3/10) [sbi_tb] test=scoreboard_test, result=PASS
(4/10) [sbi_tb] test=test_of_poll_until, result=PASS
(5/10) [sbi_tb] test=extended_write_and_read, result=PASS
(6/10) [sbi_tb] test=read_of_previous_value, result=PASS
(7/10) [sbi_tb] test=read_of_executor_status_and_inter_bfm_delay, result=PASS
(8/10) [sbi_tb] test=distribution_of_vvc_commands, result=PASS
(9/10) [sbi_tb] test=vvc_broadcast_test, result=PASS
(10/10) [sbi_tb] test=vvc_setup_and_hold_time_test, result=PASS
(1/1) [sbi_tb_multi_cycle_read] test=fixed_wait_read_test, result=PASS
Simulations done [97.60 sec]. Pass=11, Fail=0, Total=11
```

Testbench object - available methods

- File name: testbench.py
- Folder: uvvm_internal/release/regression_test
- Type: testbench object

Available methods in testbench object:

- set_library(library_name : string)
- get_library()
- set_tb_name(tb_name : string)
- get_tb_name()
- set_cleanup(mode : boolean) : remove output files from tests without errors
- reset_counters() : set number of tests run and number of failing tests counters to zero
- get_num_tests_run()
- get_num_failing_tests()
- add_test(test : string) : used as input to GC_TEST generic in testbench
- add_tests(tests : []) : used as input to GC_TEST generic in testbench
- get_tests()
- remove_tests() : internal tests array is flushed
- remove_configs() : internal configs array is flushed
- add_config(config : string) : add config setting for GC_X generic(s) in testbench
- add_configs(configs : []) : add config settings for GC_X generic(s) in testbench
- get_configs()
- check_arguments(args : []) : check script arguments
- compile() : call ../internal_script/compile_all.do
- run_simulation() : run testbench with given tests and configs
- print_statistics() : present number of tests run and number of failing tests

Internal / private methods:

- cleanup() : class internal
- check_result() : class internal
- increment_num_tests() : class internal
- increment_num_failing_tests() : class internal

Regression test

The regression test script is located in the release/regression_test folder of uvvm_internal, and consist of the following files:

- testbench.py - each sim.py script creates a testbench object of this type
- module_list.txt - a list of all UVVM modules that have a sim.py script in the internal_script folder and are part of the regression test
  Note: putting a "#" in front of a module name will omit the module from regression test
- run_regression.py - script which will run the sim.py script in all modules listed in module_list.txt, catch the number of failing tests, and present the final results.
  Note: run_regression.py will pass the command line arguments to the sim.py scripts, thus the run_regression.py script support the same command line arguments as sim.py

module_list.txt as of august 2020:

```
uvvm_util
uvvm_vvc_framework
bitvis_irqc
bitvis_uart
bitvis_vip_avalon_mm
bitvis_vip_avalon_st
bitvis_vip_axilite
bitvis_vip_axistream
bitvis_vip_clock_generator
bitvis_vip_error_injection
bitvis_vip_gmii
bitvis_vip_gpio
bitvis_vip_i2c
bitvis_vip_rgmii
bitvis_vip_sbi
bitvis_vip_scoreboard
bitvis_vip_spi
bitvis_vip_uart
bitvis_vip_spec_cov
#bitvis_vip_hvvc_to_vvc_bridge - no internal test
bitvis_vip_ethernet
#bitvis_vip_wishbone
```