
hdlregression

Release 1.4.0

UVVM

Dec 16, 2025

CONTENTS:

1	Introduction	1
1.1	What is HDLRegression	1
1.2	What is regression testing	1
1.3	Terminology	2
2	Usage	5
3	Installation	7
3.1	Setup script	7
3.2	Python PATH	8
4	Application Programming Interface (API)	9
4.1	HDLRegression()	9
4.2	Basic methods	10
4.3	Advanced methods	16
4.4	Statistical methods	27
5	Command Line Interface (CLI)	31
5.1	Examples	32
5.2	Threading	38
5.3	Simulation results	42
6	Graphical User Interface (GUI)	47
6.1	Modelsim	47
6.2	GHDL / NVC	48
7	Testbench	49
7.1	Prerequisites	49
7.2	Example Testbench	50
8	Template files	53
8.1	Basic usage	53
8.2	Advanced usage	54
9	Test Automation Server	59
10	Generated output	61
10.1	Test folder	61
11	Tips	67
11.1	Back annotated netlist simulations	67

INTRODUCTION

1.1 What is HDLRegression

HDLRegression is a fully customizable regression tool written in Python for running HDL testbenches, and have a very low user threshold for getting started with, but can also be used in very advanced projects. Setting up a test regression script is straight forward and requires little Python knowledge.

For an existing project to start using HDLRegression there are only two tasks that have to be carried out

- 1 - add a code comment to the top level testbench entity
- 2 - make a test script where libraries and files are specified.

HDLRegression is shipped with basic and advanced test script *Template files* files that have guidelines instructions that help the test designer in getting started in setting up a complete and ready to use HDLRegression test script. The basic template file show the only code that is required to run HDLRegression, where all that is needed is to add file(s) and potentially a compile library.

HDLRegression is distributed as open source and can be downloaded from [GitHub](#).

 Note

HDLRegression is not a verification framework but a tool for running testbenches that verifies the design behaviour.

This allows you to choose any verification framework, e.g. company internal developed, UVVM, OSVVM, VUnit and so on.

- a) Without making any changes to the verification files other than adding a single code comment in the testbench.
- b) Without the need for regression dedicated VHDL code from other frameworks - use whichever you prefer.
- c) With a verification framework independent regression suite.

1.2 What is regression testing

Regression testing is re-running functional and non-functional tests to ensure that previously developed and tested software still performs after a change. ([wikipedia](#))

1.3 Terminology

1.3.1 Testbench

A testbench is the top level entity and architecture which is used as input to the simulations. Verification of a DUT (device under test) may require one or more testbenches. The testbench will consist of:

- A test sequencer.
- A test-harness - not required, but recommended.
- Optionally other support modules, procedures or statements outside the test-harness.

Different configurations of a testbench using different DUT architectures are considered as the same testbench, running on different DUT representations. Different configurations of a testbench using different test-harness architectures are considered as different testbenches, as testbench behaviour may be different.

1.3.2 Test sequencer

The test sequencer is a single VHDL process controlling the simulations from start to end, and may sometimes be called the “central (test) sequencer”.

1.3.3 Test-harness

The test-harness consist of a VHDL entity containing the fixed parts of the verification environment - often shared between various testbenches. E.g. DUT and verification support such as verification components or processes, including concurrent procedures.

1.3.4 Test suite

A test suite is the complete set of testbenches required for a given DUT, or for a complete FPGA including modules.

1.3.5 Test case

A test case is

- A scenario or sequence of actions that are controlled by the test sequencer.
- May test one or multiple features/requirements.
- Typically testing of related functionality, or a logical sequence of events, or an efficient sequence of events.
- The minimum sequence of events possible to run in a single simulation execution. Thus, if there is an option to run of multiple test sequences (A, B or C), a set of test sequences (A and B) or all sequences (A+B+C), then all of A, B and C are defined as individual test cases.

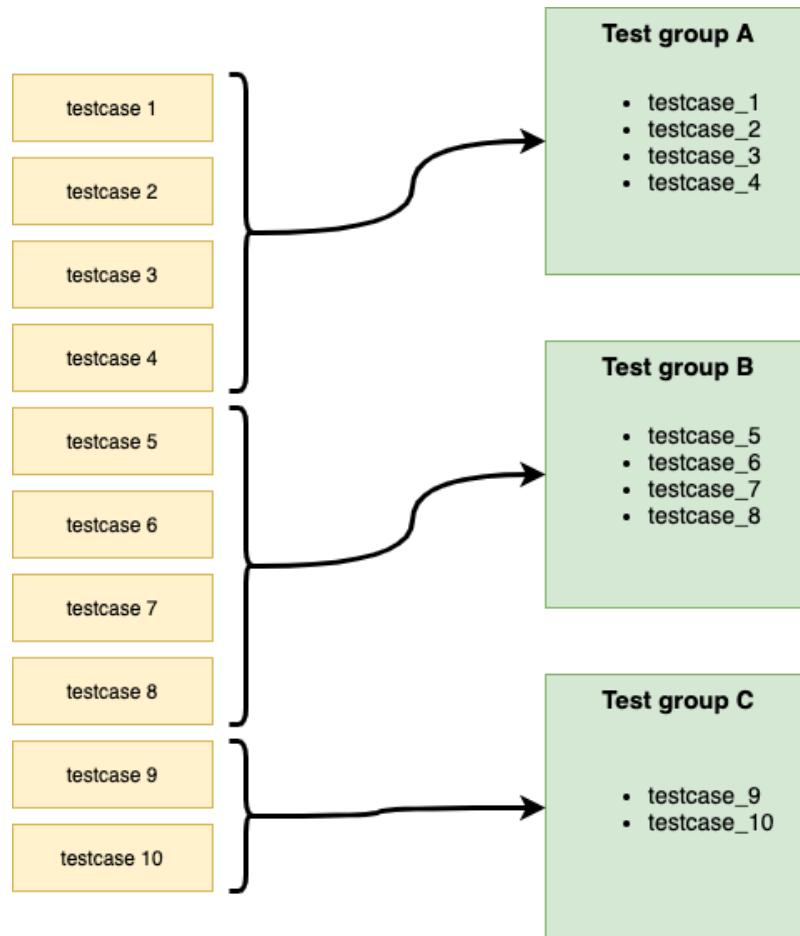
1.3.6 Test group

A test group is a collection of test cases that typically verifies the same modules or features of a DUT. There are several ways of structuring testbenches and test cases, and HDLRegression support many of these.

A test group can be:

- A single testbench or a collection of testbenches
- A single test case or a collection of test cases

Typically a test group contains testbenches and/or test cases that verifies a set of features or functionality, e.g. error injection, interface functionality or any other sub-set of DUT functionality.



CHAPTER

TWO

USAGE

HDLRegression is configured using a Python 3 script that imports the HDLRegression module which is used for creating a HDLRegression object which is used to customize the regression run using a set of *Application Programming Interface (API)* commands.

When HDLRegression is run it will perform several tasks in the background, and if any of the files added is a *testbench* file, i.e. with the `--HDLREGRESSION:TB (VHDL)` or `//HDLREGRESSION:TB (Verilog)` pragma set, HDLRegression will:

- Organize files and libraries by dependencies
- Scan for defined test cases in the testbench (a test case generic is required for this feature)
- Compile to the default or a specified library
- Run all testbenches
- Report simulation results to terminal

INSTALLATION

For the regression script to be able to use the HDLRegression package module, one has to do one of the following steps:

- Install HDLRegression using the `setup.py` script.
- Add the HDLRegression install path to *Python PATH* inside the regression script.

3.1 Setup script

There is a `setup.py` script in the HDLRegression root folder that can be used for installing HDLRegression as a Python package, and installing HDLRegression will make it importable without adding it to the Python PATH. We recommend that HDLRegression is installed as a Python package as this will make the regression script more portable and easy to write. Execute the two following commands to install HDLRegression as package:

1. Build the package

```
python3 setup.py build
```

2. Install the HDLRegression package

```
python3 setup.py develop
```

Or use the Python package-management system

```
pip install .
```

HDLRegression can be imported directly in the regression script as any standard Python module

```
from hdlregression import HDLRegression
```

3.1.1 Uninstall

Uninstalling can be done with the commands:

```
python3 setup.py develop --uninstall
```

Or if installed using pip:

```
pip uninstall hdlregression
```

3.2 Python PATH

The HDLRegression module can be used without package installation by adding the HDLRegression install path to the Python PATH variable at the beginning of the regression script:

```
import sys  
sys.path.append(<path_to_hdlregression_folder>)  
  
from hdlregression import HDLRegression
```

Note

HDLRegression will have to be added to the Python PATH in every regression script which will make the regression script less portable.

Note

Relative paths will be relative to the regression script and not where the regression script is called from.

CHAPTER
FOUR

APPLICATION PROGRAMMING INTERFACE (API)

HDLRegression is configured and run using a Python script that imports the HDLRegression module and uses a set of API methods. Because HDLRegression is written in Python the test designer can utilize the full Python API and modules to make advanced regression scripts. There are two template script files in the `/template` folder to help new users get started.

```
1 import sys
2 # ----- USER HDLRegression PATH -----
3 # If HDLRegression is not installed as a Python package (see doc)
4 # then uncomment the following line and set the path for
5 # the HDLRegression install folder :
6 #sys.path.append(<full_or_relative_path_to_hdlregression_install>
7
8 # Import the HDLRegression module to the Python script:
9 from hdlregression import HDLRegression
10
11 # Define a HDLRegression item to access the HDLRegression functionality:
12 hr = HDLRegression()
13
14 # ----- USER CONFIG START -----
15 # => hr.add_files(<filename>) # Use default library my_work_lib
16 # => hr.add_files(<filename>, <library_name>) # or specify a library name.
17
18 # ----- USER CONFIG END -----
19 hr.start()
```

4.1 HDLRegression()

This command is used for initializing the HDLRegression object which is used for defining the regression script and accessing the HDLRegression API.

HDLRegression will attempt to auto-detect available simulators and will choose either ModelSim, NVC, GHDL, or Riviera Pro based on the findings. The preferred simulator can be selected by using the `simulator` argument as shown in example 2 or using [Command Line Interface \(CLI\)](#) :

```
HDLRegression(<simulator>)
```

Argument	Type	Example	Default	Required
simulator	string	“ghdl”, “modelsim”, “nvc”	auto-detected	optional
arg_parser	argparser obj	regression_parser	None	optional
output_path	string	“hdlreg_output”	None	optional

Example 1:

```
1. hr = HDLRegression()  
  
2. hr = HDLRegression(simulator="ghdl")
```

An argparse object can be created in the regression script and passed on to the HDLRegression() object creation to allow for having local argument parsing in the regression script. When an argparse object is passed on to HDLRegression it will add all its arguments to the argparse object. The parsed arguments can be collected using the `get_args()` method as show in example 2.

Example 2:

```
import argparse  
from hdlregression import HDLRegression  
  
arg_parser = argparse.ArgumentParser(description='Regression script parser')  
arg_parser.add_argument('--rtl', action='store_true', help='run RTL simulations')  
arg_parser.add_argument('--netlist', action='store_true', help='run netlist simulations')  
  
hr = HDLRegression(arg_parser=arg_parser)  
  
args = hr.get_args()  
if args.rtl:  
    # add rtl files  
    hr.add_files(...)  
if args.netlist:  
    # add netlist files  
    hr.add_files(...)  
  
hr.start()
```

4.2 Basic methods

4.2.1 add_files()

Specifies a single or set of files that will be associated with a library name.

The library name can be selected explicitly using the `library_name` argument or by first setting a library name using the `set_library()` method and then omitting the `library_name` argument from the `add_files()` method. See example 2 and 3 below for different approaches to setting library names.

For VHDL, the files are compiled to the `library_name` library, thus the `library_name` will need to correspond with the library name used in the design or test environment files.

Files can be referenced with the relative and absolute paths, and the `add_files()` method can be called several times in

the regressions script, addressing the same or a different library name.

```
add_files(<filename>, <library_name>, <hdl_version>, <com_options>, <netlist_inst>,
          ↵<code_coverage>)

add_files(<filename>)
```

Argument	Type	Default	Required
filename	string		mandatory
library_name	string	"my_work_lib"	optional
hdl_version	string	2008	optional
com_options	string		optional
parse_file	boolean	True	optional
netlist_inst	string		optional
code_coverage	boolean		optional

Example 1:

```
hr.add_files("../src/my_testbench.vhd", "my_testbench_lib", hdl_version='2008')

hr.add_files("../backend/my_design.sdf", "my_design_lib", hdl_version='2008', netlist_
          ↵inst='/my_testnech/i_testHarness/i_dut')

hr.add_files("../src/*.vhd", code_coverage=True)
```

Example 2: with library name

```
hr.add_files(filename="c:/tools/uvvm/uvvm_util/src/*.vhd", library_name="uvvm_util")

hr.add_files(filename="c:/project/design/src/*.vhd", library_name="design_lib")
hr.add_files(filename="c:/project/design/src/ip/*.vhd", library_name="design_lib")

hr.add_files(filename="c:/project/design/tb/*.vhd", library_name="test_lib")
```

Example 3: with set_library()

```
hr.set_library(library_name="uvvm_util")
hr.add_files(filename="c:/tools/uvvm/uvvm_util/src/*.vhd")

hr.set_library(library_name="design_lib")
hr.add_files(filename="c:/project/design/src/*.vhd")
hr.add_files(filename="c:/project/design/ip/src/*.vhd")

hr.set_library(library_name="test_lib")
hr.add_files(filename="c:/project/design/tb/*.vhd")
```

Note

Relative paths will be relative to the regression script and not where the regression script is called from.

Note

A back annotated timing file (SDF) require the `netlist_inst` arguments and a back annotated timing file (VHD) require the `parse_file` argument set to `False`.

1. The `netlist_inst` argument is a string that has to be set to design instantiation path in the design.
2. Any number of back-annotated timing files can be added.

Note

The `code_coverage` argument enables code coverage for a single file if an explicit filename is given, or a set of files when used with wildcards in the filename.

It is required that the `set_code_coverage()` method is used to set the code coverage settings.

Warning

When `parse_file` is set to `False` HDLRegression will not parse the file content, not include the file in the compilation order and not compile the file.

Tip

Use wildcards to more effectively filter searches, i.e. test cases and filenames.

Pattern	Meaning
*	match all
?	match a single character
[seq]	match any character in seq
[!seq]	match all character not in seq

4.2.2 `set_library()`

Changes the default library name used when `add_files()` is used without the `library_name` argument.

```
set_default_library(<library_name>)
```

Argument	Type	Required
library_name	string	mandatory

Example:

```
hr.set_library("testbench_lib")
```

Note

The default library name is “*my_work_lib*”.

4.2.3 start()

This method will initiate compilation, simulation, reporting etc.

After calling this method, adding files or making changes to simulation configurations in the regression script is not permitted. Ensure that all necessary files and configurations are set before invoking the method to avoid issues during the simulation process.

Return code

The return code from the start() method is either 0 or 1, based on whether the success criteria listed below are met:

Criteria	Return code
No compilation error and test case(s) has been run without errors	0
No compilation error and no test case run	1
No compilation error and test case run with one or more errors	1
Compilation error (no test cases will be run)	1

Arguments

The default operation is to run in *regression mode* without *GUI* enabled, yet this can be changed using the available arguments or by using the *command line interfaces*.

```
start(<gui_mode>, <stop_on_failure>, <regression_mode>, <threading>, <sim_options>,
      ↵<netlist_timing>, <com_options>)
```

Argument	Options & Type	Default	Description
gui_mode	True/False (boolean)	False	GUI mode control
regression_mode	True/False (boolean)	True	Regression mode control
stop_on_failure	True/False (boolean)	False	Stop on first failure
threading	True/False (boolean)	False	Enable threading
com_options	string/list of string	<i>See table 1</i>	Compilation options
sim_options	string/list of string	<i>See table 2</i>	Simulation options
runtime_options	list of string	<i>See table 3</i>	Runtime options
global_options	list of string	<i>See table 4</i>	Global options
elab_options	list of string	<i>See table 5</i>	Elaboration options
netlist_timing	string	None	Netlist timing settings
keep_code_coverage	True/False (boolean)	False	Keep code coverage data
no_default_com_options	True/False (boolean)	False	Disable default options
ignore_simulator_exit_codes	list of int	[]	Ignore specific exit codes

Example:

```
hr.start(gui_mode=True, threading=True)
```

(continues on next page)

(continued from previous page)

```
hr.start(netlist_timing=' -sdfmin')

hr.start(sim_options="-t ps -do \"quietly set NumericStdNoWarnings 1\"")
```

Table 1: Compilation Options

Simulator	Default
ModelSim (VHDL)	[“-suppress”, “1346,1236,1090”, “-2008”]
ModelSim (Verilog)	[“-vlog01compat”]
NVC	[“-relaxed”]
GHDL	[“-std=08”, “–ieee=standard”, “–frelaxed-rules”, “–warn-no-shared”, “–warn-no-hide”]
Riviera Pro	[“-2008”, “-nowarn”, “COMP96_0564”, “-nowarn”, “COMP96_0048”, “-nowarn”, “DAGGEN_0001”]
Vivado	[“-2008”]

Table 2: Simulation Options

Simulator	Default
ModelSim	[]
NVC	[]
GHDL	[]
Riviera Pro	[]
Vivado	[]

Table 3: Runtime Options

Simulator	Default
Modelsim	[]
NVC	[]
GHDL	[]
Riviera Pro	[]
Vivado	[]

Table 4: Global Options

Simulator	Default
Modelsim	[]
NVC	[“-stderr=error”, “–messages=compact”, “-M64m”, “-H64m”]
GHDL	[]
Riviera Pro	[]
Vivado	[]

Table 5: Elaboration Options

Simulator	Default
Modelsim	[]
NVC	[“-e”, “-no-save”, “-jit”]
GHDL	[]
Riviera Pro	[]
Vivado	[]

Note

- `gui_mode` selects if simulations should be run from terminal or inside *GUI* - if supported by the simulator. In GUI the simulator is started with predefined HDLRegression methods that simplifies compilation and running tests.
- `regression_mode` selects run method, i.e. only run tests that:
 1. have not previously been run
 2. have not passed
 3. are affected by file changes and need to be rerun.
- `stop_on_failure` selects if the regression run shall continue running if a test fails.
- `threading` selects if tasks are run in parallel. Depending on the workload this can decrease run time of some regression runs.
- `sim_options` adds extra commands to simulator executor call.
- `netlist_timing` is a string that has to be set to “-sdfmin”, “-sdftyp” or “-sdfmax”.
- `keep_code_coverage` will keep code coverage results from a previous test run. This can be useful in situations where a subset of tests needs to be rerun to achieve wanted code coverage.
- `no_default_com_options` disables preconfigured settings for disabling the following warnings:
 1. vcom-1236: shared variables must be of a protected type
 2. vcom-1346: default expression of interface object is not globally static
 3. vcom-1090: possible infinite loop: process contains no WAIT statement

Warning

`gui_mode` will run test cases in one of these modes:

1. test cases added by the `add testcase()` method or using the *command line interfaces*.
2. regression mode.

starting with bullet point 1, and if no test cases have been added, moving on to bullet point 2.

4.3 Advanced methods

4.3.1 add_file_to_run_folder()

Copies a single file to the test case run folder.

```
add_file_to_run_folder(<filename>, <tc_id>)
```

Argument	Type	Required
filename	string	mandatory
tc_id	string	mandatory

Example:

1. hr.add_file_to_run_folder(filename="c:/design/tb/input_data.txt", tc_id="1")
2. hr.add_file_to_run_folder(filename="../tb/input_data.txt", tc_id="7")

 **Note**

Relative paths will be relative to the regression script and not where the regression script is called from.

4.3.2 add_generics()

Selects the generics to be used when running a test case. A test run is created when generics are added to a test case, thus calling `add_generics()` two times will create two test runs.

```
add_generics(<entity>, <architecture>, <generics>)
```

Argument	Type	Required
entity	string	mandatory
architecture	string	optional
generics	list [string, int/string/bool]	mandatory

 **Important**

- All generics that are used for input or output files inside a testbench will require the PATH keyword when setting the generic in the regression script.
The generic value and the PATH keyword has to be of a Python **tuple** type. HDLRegression will make the adjustments for the generic paths to match HDLRegression test paths. See example 3.

Example:

1. hr.add_generics(entity="my_dut_tb", generics=["GC_BUS_WIDTH", 16, "GC_ADDR_WIDTH", 8])

(continues on next page)

(continued from previous page)

```
2. hr.add_generics(entity="my_dut_tb", architecture="test", generics=my_generics_list)

3. hr.add_generics(entity="my_dut_tb", generics=["GC_DATA_FILE", ("../test_data/input_
    ↵data.txt", "PATH"), "GC_MASTER_MODE", True])
```

Note

Relative paths will be relative to the regression script and not where the regression script is called from.

4.3.3 add_precompiled_library()

Specifies the name and path of a precompiled library.

Note

The library will never be compiled and only a reference is added to the modelsim.ini file. Any number of precompiled libraries can be added.

```
add_precompiled_library(<compile_path>, <library_name>)
```

Argument	Type	Required
compile_path	string	mandatory
library_name	string	mandatory

4.3.4 add testcase()

Adding test case(s) will configure HDLRegression to only run these test cases. If no test cases are added, all discovered tests are run.

Test case selection can also be done via the *command line interfaces*. Any test cases selected from the CLI override scripted test case selection.

Important

A test case identifier is a string composed of the following elements:

1. Testbench entity name
2. Testbench architecture name (optional)
3. Sequencer / built-in test case name (optional)

The elements are separated by a dot (.):

```
<entity>[.<architecture>[.<testcase>]]
```

In addition, an optional library selector may be prepended using

```
[<library>:<entity>[.<architecture>[.<testcase>]]]
```

If the library selector is omitted, tests from all libraries are considered.

💡 Tip

Use wildcards to more effectively filter searches, i.e. test cases and filenames.

Pattern	Meaning
*	match all
?	match a single character
[seq]	match any character in seq
[!seq]	match all characters not in seq

```
add_testcase(<test case>)
```

Argument	Type	Required
<i>test case</i>	string / list of strings	mandatory

Supported selector forms

Selector	Meaning
entity	All architectures and testcases
entity.arch	All testcases in architecture
entity.arch.tc	One specific test
lib:entity.arch.tc	Same, but limited to one library
lib:	All tests in a library

Example:

```
add_testcase("interface_tb.test_arch.read_test")  
  
add_testcase("interface_tb..read_")  
  
add_testcase("lib_uart:interface_tb.func.*")  
  
add_testcase(":interface_tb.")  
  
add_testcase(testcase_list)
```

💡 Note

Library filtering is only supported via add testcase() and CLI. The *start()* method returns error code 1 if no test cases matched.

4.3.5 add_to_testgroup()

Adds one or more existing tests to a named *test group*, allowing tests to be executed as logical groups. A test group is identified by a name and can contain any number of tests. There is no limit to the number of test groups or the number of tests within a group. Test groups are executed by selecting the group name, either from the Python API or via the *command line interface*.

```
hr.add_to_testgroup(testgroup_name, entity, architecture=None, testcase=None, generic=[])
```

Argument	Type	Required
testgroup_name	string	mandatory
entity	string	mandatory
architecture	string	optional
test case	string	optional
generic	list [string, int/string/bool]	optional

Note

- `add_to_testgroup()` adds existing tests to a collection, i.e. no new tests are created.
- The `test case` argument is for selecting sequencer built-in test cases.
- The `start()` method will return error code 1 if no test group or test case were found.
- All string arguments support Unix-style wildcards.

Tip

Use wildcards to more effectively filter searches, i.e. test cases and filenames.

Pattern	Meaning
*	match all
?	match a single character
[seq]	match any character in seq
[!seq]	match all character not in seq

4.3.6 compile_uvvm()

Compiles UVVM to HDLRegression library folder, making UVVM available to all tests run by HDLRegression.

```
hr.compile_uvvm(<path_to_uvvm>)
```

Argument	Example	Required
path_to_uvvm	“./ip/UVVM”	mandatory

Example:

```
hr.compile_uvvm("c:/development/tools/UVVM")
```

 ⓘ Important

- The UVVM path has to absolute or relative to the regression script location.

4.3.7 compile_osvvm()

Compiles OSVVM to HDLRegression library folder, making OSVVM available to all tests run by HDLRegression.

```
hr.compile_osvvm(<path_to_osvvm>)
```

Argument	Example	Required
path_to_osvvm	“./ip/OSVVM”	mandatory

Example:

```
hr.compile_osvvm("c:/development/tools/OSVVM")
```

 ⓘ Important

- The OSVVM path has to absolute or relative to the regression script location.

4.3.8 configure_library()

Set special settings for a library that differs significantly from the regular settings.

```
hr.configure_library(<library>, <never_recompile>, <set_lib_dep>)
```

Argument	Options	Example	Required
library	<i>library name</i> (string)	“can_ip_library”	mandatory
never_recompile	True/False (boolean)	True	optional
set_lib_dep	<i>library name</i> (string)	“ip_library”	optional

Example:

```
hr.configure_library(library='can_ip_library', never_recompile=True)
```

4.3.9 gen_report()

Writes a test run report file to the hdlregression/test folder. The default report file is `report.txt` and can be changed using the `report_file` argument. The report file is saved in the /hdlregression/test folder, thus no path should be given to the report name.

```
gen_report(<report_file>, <compile_order>, <library>)
```

Argument	Options & Type	Default	Required
report_file	<i>filename</i> (string)	“report.txt”	optional
compile_order	True / False (boolean)	False	optional
library	True / False (boolean)	False	optional

Example:

```
hr.gen_report(report_file="sim_report.csv", compile_order=True)
hr.gen_report(report_file="sim_report.html", compile_order=True, library=True)
```

 **Important**

Supported file types are `.txt`, `.csv`, `.html`, `.xml` and `.json` and the file type is extracted from the file name.

4.3.10 get_args()

The command is used for getting the parsed arguments from HDLRegression. This method can be used when there is a argparse object that is created in the regression script. See `HDLRegression()` example 2 for usage.

Example:

```
args = hr.get_args()
```

4.3.11 get_file_list()

The command is used for reading back the files added to the libraries in HDLRegression. All files from all libraries are returned in a list.

Example:

```
file_list = hr.get_file_list()
```

4.3.12 remove_file()

Removes a file that has been added to a library, e.g. after using `add_files()` with asterix for adding several files.

```
remove_file(<filename>, <library_name>)
```

Argument	Type	Required
filename	string	mandatory
library_name	string	mandatory

Example:

```
hr.add_files("../src/*.vhd", "testbench_lib")
hr.remove_file("unused_file.vhd", "testbench_lib")
hr.start()
```

 **Note**

The filename can not include the path to the file or any wildcards.

4.3.13 run_command()

The command is executed by HLDRegression at the given stage in the regression script. I.e. pre-simulation commands will have to be called prior to `start()` and post-simulation commands need to be called after `start()`.

```
hr.run_command(<command>)
```

Argument	Type	Required
command	string	mandatory
verbose	boolean	optional

 **Note**

No output is printed to the terminal by default, but this can be changed by setting the `verbose` argument to True.

Example:

```
hr.run_command('python3 ../script/run_spec_cov.py --config ../script/config.txt')

hr.run_command('vsim -version', verbose=True)
```

 **Note**

Relative paths will be relative to the regression script and not where the regression script is called from.

4.3.14 set_code_coverage()

Sets the code coverage settings used when running the tests.

```
hr.set_code_coverage(<code_coverage_settings>, <code_coverage_file>, <exclude_file>,
                    <merge_options>)
```

Argument	Type	Example	Required
code_coverage_settings	string	“bcst”	mandatory
code_coverage_file	string	“coverage.ucdb”	mandatory
exclude_file	string	“exceptions.tcl”	optional
merge_options	string	“-testassociated”	optional

Note

- *add_files()* require the `code_coverage` argument enabled for every file that should sample code coverage.
- Each test run will generate a `code_coverage_file` inside its test folder.
- Results from the regression are accumulated in a `code_coverage_file_merge` file inside the `test/coverage/` folder.
- Exceptions are filtered from the accumulated file automatically in a `code_coverage_file_filter` file inside the `test/coverage/` folder.
- Reports are written to the `test/coverage/txt` and `test/coverage/html` folders using the filtered exception results if a `exclude_file` is set, or using the merged code coverage results if no `exclude_file` is set.
- Only the current test run is used for code coverage, meaning that a full regression run is required to sample code coverage for the complete test suite.

Example:

```
hr.set_code_coverage("bcst", "code_coverage.ucdb")
hr.set_code_coverage("bcst", "code_coverage.ucdb", "exclude.tcl")
```

4.3.15 set_dependency()

Specifies the libraries that have a dependency to the `library_name` library, and ensures that `library_name` is compiled after all of the libraries listed in `dep_library`.

```
set_dependency(<library_name>, <dep_library>)
```

Argument	Type	Required
library_name	string	mandatory
dep_library	list [string]	mandatory

Note

1. Specifying the library dependency is usually not necessary as HDLRegression is capable of detecting dependencies.
2. dep_library list has to be a list of library name(s).

4.3.16 set_pre_sim_tcl_cmd()

Sets a single Tcl command that will be executed by the simulator **before** a test starts.

This is useful for preparing the simulator session (e.g., setting variables, loading packages, or running a small start-up script) without modifying the testbench.

```
hr.set_pre_sim_tcl_cmd(<tcl_command>)
```

Argument	Type	Required
tcl_command	string	mandatory

Note

- The command must be set **before** calling start().
- The setting applies to simulators with a Tcl front-end (e.g., ModelSim/Questa, Riviera-PRO). Other simulators may ignore this setting.
- Calling this method multiple times will overwrite the previously set command.

Example:

```
hr.set_pre_sim_tcl_cmd('do ../scripts/pre_sim.do')
hr.start()
```

4.3.17 set_result_check_string()

The result of a test run is determined by scanning the simulation log file, searching after a specific string. If the string is found the test run is set as **PASS**, and **FAIL** otherwise, thus only a passing test should report the check string.

```
set_result_check_string(<check_string>)
```

Argument	Type	Required
check_string	string	mandatory

Note

The default test pass string is the UVVM `report_alert_counters(FINAL)` summary, with `SUCCESS` as criteria for a passing test.

Example:

```
hr.set_result_check_string("testcase passed")
```

Listing 1: Example TB with test case result string

```

1 p_seq : process
2   variable v_check_ok : boolean := false;
3 begin
4   -- testcase checks, e.g.
5   -- v_check_ok := check_value(v_act_data, v_exp_data, error, "checking receive data",_
6   -- C_SCOPE);
7
8   if v_check_ok = true then
9     report "testcase passed";
10  end if;
11
12  -- Finish the simulation
13  std.env.stop;
14  wait; -- to stop completely
end process;
```

4.3.18 set_simulator()

HDLRegression is configured to run using Modelsim and VHDL version 2008 as default. This method allows for changing

- Simulator
- Simulator executable path
- Simulator com_options

This can be useful when the test script should be run with a different version of Modelsim other than the one listed in the system path, where all that is needed is to change the path for the Modelsim executable.

It is also possible to select simulator when initializing the *HDLRegression* object, but without selecting compile options and setting simulator executable path, or by using *command line interfaces*.

```
set_simulator(<simulator>, <simulator_path>, <com_options>)
```

Argument	Options	Example	Required
simulator	<i>simulator name</i> (string)	“MODELSIM”/”GHDL”/”NVC”/”RIVIERA_PF	mandatory
simulator_path	<i>simulator_executable_path</i> (string)	“c:/ghdl/bin”	optional
com_options	<i>compile optionss</i> (string/list of string)	“-suppress 1346,1236,1090 -2008”	optional

Example:

```
hr.set_simulator(simulator="GHDL")  
  
hr.set_simulator(simulator="MODELSTM", path='c:/tools/intelFPGA/20.1/modelsim_ase/  
↪win32aloem')
```

Important

All path slashes has to be written as forward slash / .

Note

Relative paths will be relative to the regression script and not where the regression script is called from.

4.3.19 set testcase identifier name()

Sets the name of the test case generic used when defining several test cases inside a single testbench architecture. The default test case generic is GC_TESTCASE, but any name can be given.

Note

- The sequencer built-in test case if-structure will need to match this generic.
- HDLRegression extracts all the sequencer built-in test cases based on the combined usage of this generic and if-matched strings.

```
hr.set testcase identifier name(<testcase_id>)
```

Argument	Type	Required
testcase_id	string	mandatory

Example:

```
hr.set testcase identifier name("GC_TESTCASE")
```

Listing 2: Example TB with test case ID generic

```
1  --hdlregression:tb  
2  entity tb_top is  
3      generic (  
4          GC_TESTCASE : string := "read_test"  
5      );  
6  end tb_top;  
7  
8  architecture test of simple_tb is  
9      constant C_SCOPE : string := "SIMPLE_TB";  
10     begin
```

(continues on next page)

(continued from previous page)

```

11  p_main : process
12  begin
13      if GC_TESTCASE = "read_test" then
14          -- read tests
15      elsif GC_TESTCASE = "write_test" then
16          -- write tests
17      else
18          report "Unknown test " & GC_TESTCASE;
19      end if;
20      -- Finish the simulation
21      std.env.stop;
22      wait; -- to stop completely
23  end process;
24

```

4.3.20 set_simulator_wave_file_format()

Sets the wave dump format for GHDL and NVC wave files. Options are FST and VCD.

Note

- VCD file format is default if no other is selected.
- GUI mode has to be enabled using API method or CLI option.
- Options : VCD, FST, GHW (GHDL).

```
hr.set_simulator_wave_file_format(<wave_format>)
```

Argument	Type	Required
wave_format	string	mandatory

Example:

```
hr.set_simulator_wave_file_format("VCD")
hr.start(gui_mode=True)
```

4.4 Statistical methods

4.4.1 get_results()

Returns a list of all passed, failed and not run tests.

```
hr.get_results()
```

Example:

```
result_list = hr.get_results()  
  
passed_tests = result_list[0]  
failed_tests = result_list[1]  
not_run_tests = result_list[2]
```

```
(passed_tests, failed_tests, not_run_tests) = hr.get_results()
```

4.4.2 get_num_tests_run()

Returns the number of tests run.

```
hr.get_num_tests_run()
```

Example:

```
num_tests = hr.get_num_tests_run()
```

4.4.3 get_num_pass_tests()

Returns the number of passed test runs.

```
hr.get_num_pass_tests()
```

Example:

```
num_passed_tests = hr.get_num_pass_tests()
```

4.4.4 get_num_fail_tests()

Returns the number of failed test runs.

```
hr.get_num_fail_tests()
```

Example:

```
num_failed_tests = hr.get_num_fail_tests()
```

4.4.5 get_num_pass_with_minor_alert_tests()

Returns the number of passed test runs that completed with minor alerts.

Note

This is only applicable for tests run with the UVVM verification framework.

```
hr.get_num_pass_with_minor_alert_tests()
```

Example:

```
num_passed_tests_with_minor_alerts = hr.get_num_pass_with_minor_alert_tests()
```

COMMAND LINE INTERFACE (CLI)

The configuration of a regression run can be set directly from the command line using command line interface. This can be useful when debugging the behaviour of a design, e.g. by running in *GUI mode*, or working with a *testcase* or *test group*.

The command line interface are processed at startup and will override any scripted configurations that are in conflict.

Arguments		Description
-h	-help	Help screen
-v	-verbose	Enable full verbosity
-d	-debug	Enable debug mode
-g	-gui	Run with simulator gui
-fr	-fullRegression	Run all tests
-c	-clean	Remove all before test run
-tc [LIB] TB_ENTITY [TB_ARCH [TC]]	-testCase [LIB] TB_ENTITY [TB_ARCH [TC]]	Run selected test case
-tg TESTGROUP	-testGroup TESTGROUP	Run selected test group
-ltc	-listTestcase	List all discovered test cases
-ltg	-listTestgroup	List all test groups
-lco	-listCompileOrder	List libraries and files in compile order
-fc	-forceCompile	Force recompile
-sof	-stopOnFailure	Stop simulations on test case fail
-s	-simulator	Set simulator (require path in env)
-t	-threading [N]	Run tasks in parallel
-ns	-no_sim	No simulation, compile only
	-showWarnError	Show sim error and warning messages.
	-noColor	Disable terminal output colors.
	-waveFormat	Wave file format [VCD (default) or FST]
	-wlf	Save wlf file after sim (Questa/Modelsim)
-etj	-exportTestcaseJson	Export TC to JSON file with the given path

5.1 Examples

5.1.1 Selecting simulator

HDLRegression will try to auto-detect the simulator to use, but this can be overridden using the `-s` or `--simulator` argument.

```
> python ../test/regression.py -s NVC
> python ../test/regression.py -s RIVIERA-PRO
> python ../test/regression.py -s GHDL
> python ../test/regression.py -s MODELSIM
```

5.1.2 Full regression

Enabling the *full regression mode* ensures that all test cases are run, regardless of any previous runs, i.e. re-running the complete test suite.

```
> python ../test/regression.py -fr
```

```
$ python ../test/regression.py -fr

=====
HDLRegression version 0.20.0
Please see /doc/hdlregression.pdf documentation for more information.
=====

Scanning files...
Building test suite structure...
Setting up: Y://bitvis//hdlregression//sim//hdlregression//library\modelsim.ini.
Compiling library: uvvm_util - OK -
Compiling library: uvvm_vvc_framework - OK -
Compiling library: bitvis_vip_scoreboard - OK -
Compiling library: bitvis_vip_sbi - OK -
Compiling library: bitvis_irqc - OK -
Compiling library: bitvis_vip_uart - OK -
Compiling library: bitvis_vip_clock_generator - OK -
Compiling library: bitvis_uart - OK -

Starting simulations...
Running 9 out of 9 test(s) using 1 thread(s).
Running: bitvis_uart uart_vvc_demo_tb.func
Result: PASS (0h:0m:9s).

Running: bitvis_uart uart_simple_bfm_tb.func
Result: PASS (0h:0m:1s).

Running: bitvis_uart uart_vvc_tb.func.check_register_defaults
Generics: GC_TESTCASE=check_register_defaults
Result: PASS (0h:0m:1s).

Running: bitvis_uart uart_vvc_tb.func.check_simple_transmit
Generics: GC_TESTCASE=check_simple_transmit
Result: PASS (0h:0m:2s).

Running: bitvis_uart uart_vvc_tb.func.check_simple_receive
Generics: GC_TESTCASE=check_simple_receive
Result: PASS (0h:0m:2s).

Running: bitvis_uart uart_vvc_tb.func.check_single_simultaneous_transmit_and_receive
Generics: GC_TESTCASE=check_single_simultaneous_transmit_and_receive
Result: PASS (0h:0m:1s).

Running: bitvis_uart uart_vvc_tb.func.check_multiple_simultaneous_receive_and_read
Generics: GC_TESTCASE=check_multiple_simultaneous_receive_and_read
Result: PASS (0h:0m:2s).

Running: bitvis_uart uart_vvc_tb.func.skew_sbi_read_over_uart_receive
Generics: GC_TESTCASE=skew_sbi_read_over_uart_receive
Result: PASS (0h:0m:5s).

Running: bitvis_uart uart_vvc_tb.func.skew_sbi_read_over_uart_receive_with_delay_functionality
Generics: GC_TESTCASE=skew_sbi_read_over_uart_receive_with_delay_functionality
Result: PASS (0h:0m:4s).

Simulation run time: 0h:0m:32s.
SIMULATION SUCCESS: 9 passing test(s).
```

5.1.3 Test cases

All tests that are discovered by HDLRegression can be listed using the `-ltc` or `--listTestcase` argument, and are listed as `<library>:<testbench entity>.<testbench architecture>.<sequencer built-in test case>` or just `<library>:<testbench entity>.<testbench architecture>` if no sequencer built-in test cases are used. Note that the library name is optional and only used when multiple libraries are used in the regression script.

```
> python ../test/regression.py -ltc
```

```
[✓ sim % python3 ../script/maintenance_script/test.py -ltc
=====
HDLRegression version 1.4.0
See /doc/hdlregression.pdf for documentation.
=====

Scanning files...
Building test suite structure...
TC:1 - uart_vvc_tb.func.check_register_defaults
TC:2 - uart_vvc_tb.func.check_of_uart_transmit
TC:3 - uart_vvc_tb.func.check_of_uart_receive
TC:4 - uart_vvc_tb.func.check_of_7bit_data
TC:5 - uart_vvc_tb.func.check_uart_receive_with_scoreboard
TC:6 - uart_vvc_tb.func.test_advanced_uart_expect
TC:7 - uart_vvc_tb.func.test_parity_error_and_status
TC:8 - uart_vvc_tb.func.test_reading_executor_status_and_advanced_uart_expect
TC:9 - uart_vvc_tb.func.test_terminate_function
TC:10 - uart_vvc_tb.func.check_of_sending_command_to_both_channels_and_inter_bfm_delays
TC:11 - uart_vvc_tb.func.test_unwanted_activity_detection
TC:12 - uart_monitor_tb.func
TC:13 - uvvm_demo_tb.func
```

Running a selected test is done using the `-tc <library>:<testbench.architecture.test case>` or `--testCase <library>:<testbench.architecture.test case>` argument

Tip

Use wildcards to more effectively filter searches, i.e. test cases and filenames.

Pattern	Meaning
*	match all
?	match a single character
[seq]	match any character in seq
[!seq]	match all character not in seq

```
> python ../test/regression.py -tc uart_vvc_tb.func.check_simple_receive
```

```
> python ../test/regression.py -tc lib_uart:uart_vvc_tb.func.check_simple_receive
```

A test case can also be selected using the test case number from the `-ltc` or `--listTestcase` argument

```
> python ../test/regression.py -tc 5
```

```

[V sim % python3 ./script/maintenance_script/test.py -tc uart_vvc_tb.func.test_advanced_uart_expect
=====
HDLRegression version 1.4.0
See /doc/hdlregression.pdf for documentation.
=====

Scanning files...
Building test suite structure...
Simulator: NVC
Compiling library: uvvm_util - OK -
Compiling library: uvvm_vvc_framework - OK -
Compiling library: bitvis_vip_scoreboard - OK -
Compiling library: bitvis_vip_clock_generator - OK -
Compiling library: bitvis_vip_sbi - OK -
Compiling library: bitvis_uart - OK -
Compiling library: bitvis_vip_uart - OK -

Starting simulations...
Running 1 out of 13 test(s) using 1 thread(s).
Running: bitvis_vip_uart uart_vvc_tb.func.test_advanced_uart_expect (test_id: 6)
Result: PASS (0h:0m:0s).

-----
Simulation run time: 0h:0m:0s.
SIMULATION SUCCESS: 1 passing test(s).
Number of fail tests OK.

```

Tip

Test cases are identified by:

1. <library>:<entity_name>.<architecture_name>.<sequencer_test case>
2. <entity_name>
3. <entity_name>.<architecture_name>
4. <entity_name>.<architecture_name>.<sequencer_test case>

When selecting test cases to run, you can utilize wildcards to simplify the process. However, it's important to note that the test case identifier must follow a specific naming convention.

For example, if you want to run all sequencer test cases that contain the word "write," you would need to specify the identifier as <entity_name>.<architecture_name>.write.

Note that you can also use wildcards for <entity_name> and/or <architecture_name> to further refine your filter.

5.1.4 Test groups

Listing of test groups that have been defined in the regression script. In the code snippet below there are defined two test groups, *transmit_tests* and *receive_tests*, that will run all test cases that have *transmit* and *receive* in the test case name, and is defined in testbench *uart_vvc_tb* architecture *func*. There is also a test group *selection_tests* that will run all test cases that are part of the *uart_vvc_demo_tb* and *uart_simple_bfm_tb* entities.

Defining test groups

```
hr.add_to_testgroup('transmit_tests', 'uart_vvc_tb', 'func', '*transmit*') # run all
˓→transmit related tests
hr.add_to_testgroup('receive_tests', 'uart_vvc_tb', 'func', '*receive*')    # run all
˓→receive related tests
hr.add_to_testgroup('selection_tests', entity='uart_vvc_demo_tb')          # run this
˓→testbench
hr.add_to_testgroup('selection_tests', entity='uart_simple_bfm_tb')         # run this
˓→testbench
```

Tip

Use wildcards to more effectively filter searches, i.e. test cases and filenames.

Pattern	Meaning
*	match all
?	match a single character
[seq]	match any character in seq
[!seq]	match all character not in seq

Listing test groups

```
> python ../test/regression.py -ltg
```

```
$ python ../test/regression.py -ltg

=====
HDLRegression version 0.20.0
Please see /doc/hdlregression.pdf documentation for more information.
=====

Scanning files...
Building test suite structure...
|---- uart_vvc_tb_all
|  |-- uart_vvc_tb.func
|---- transmit_tests
|  |-- uart_vvc_tb.func.*transmit*
|---- receive_tests
|  |-- uart_vvc_tb.func.*receive*
|---- selection_tests
|  |-- uart_vvc_demo_tb
|  |-- uart_simple_bfm_tb
```

Running test groups

Running one of the test groups, e.g. *receive_tests*, will run all tests with names that matches:

- testbench entity with `uart_vvc_tb`
- testbench architecture with `func`
- sequencer built-in test case with `receive`

```
> python ../test/regression.py --testGroup receive_tests
```

```
$ python ../test/regression.py --testGroup receive_tests
=====
HDLRegression version 0.20.0
Please see /doc/hdlregression.pdf documentation for more information.
=====

Scanning files...
Building test suite structure...

Starting simulations...
Moving previous test run to: ./hdlregression\test_2022-04-28_19.55.22.345699.
Running 5 out of 9 test(s) using 1 thread(s).
Running: bitvis_uart uart_vvc_tb.func.check_simple_receive
Generics: GC_TESTCASE=check_simple_receive
Result: PASS (0h:0m:1s).

Running: bitvis_uart uart_vvc_tb.func.check_single_simultaneous_transmit_and_receive
Generics: GC_TESTCASE=check_single_simultaneous_transmit_and_receive
Result: PASS (0h:0m:1s).

Running: bitvis_uart uart_vvc_tb.func.check_multiple_simultaneous_receive_and_read
Generics: GC_TESTCASE=check_multiple_simultaneous_receive_and_read
Result: PASS (0h:0m:1s).

Running: bitvis_uart uart_vvc_tb.func.skew_sbi_read_over_uart_receive
Generics: GC_TESTCASE=skew_sbi_read_over_uart_receive
Result: PASS (0h:0m:5s).

Running: bitvis_uart uart_vvc_tb.func.skew_sbi_read_over_uart_receive_with_delay_functionality
Generics: GC_TESTCASE=skew_sbi_read_over_uart_receive_with_delay_functionality
Result: PASS (0h:0m:4s).

Simulation run time: 0h:0m:15s.
SIMULATION SUCCESS: 5 passing test(s).
```

5.2 Threading

HDLRegression will run all tasks (pre-processing and test case simulations) in a sequential order, but this can be changed using the `-t` / `--threading` option, and optioinally with a number of threads.

Note

- Running simulations in parallel using N threads may require N simulator licenses.
- **Pre-processing threads are scaled to:**
 - > the number of libraries
 - > the number of files in each library
 - > the number of parsers

5.2.1 Sequential

All pre-processing steps and test case running are performed sequentially.

```
> python ../test/regression.py -tg receive_tests
```

```
$ python ../test/regression.py --testGroup receive_tests

=====
HDLRegression version 0.20.0
Please see /doc/hdlregression.pdf documentation for more information.
=====

Scanning files...
Building test suite structure...

Starting simulations...
Moving previous test run to: ./hdlregression\test_2022-04-28_19.55.22.345699.
Running 5 out of 9 test(s) using 1 thread(s).
Running: bitvis_uart uart_vvc_tb func check_simple_receive
Generics: GC_TESTCASE=check_simple_receive
Result: PASS (0h:0m:1s).

Running: bitvis_uart uart_vvc_tb func check_single_simultaneous_transmit_and_receive
Generics: GC_TESTCASE=check_single_simultaneous_transmit_and_receive
Result: PASS (0h:0m:1s).

Running: bitvis_uart uart_vvc_tb func check_multiple_simultaneous_receive_and_read
Generics: GC_TESTCASE=check_multiple_simultaneous_receive_and_read
Result: PASS (0h:0m:1s).

Running: bitvis_uart uart_vvc_tb func skew_sbi_read_over_uart_receive
Generics: GC_TESTCASE=skew_sbi_read_over_uart_receive
Result: PASS (0h:0m:5s).

Running: bitvis_uart uart_vvc_tb func skew_sbi_read_over_uart_receive_with_delay_functionality
Generics: GC_TESTCASE=skew_sbi_read_over_uart_receive_with_delay_functionality
Result: PASS (0h:0m:4s).

Simulation run time: 0h:0m:15s.
SIMULATION SUCCESS: 5 passing test(s).
```

5.2.2 Pre-processing in parallel, simulations sequentially

All pre-processing steps are performed in parallel and test case running is performed sequentially.

```
> python ../test/regression.py -tg receive_tests -t
```

```
$ python ../test/regression.py --testGroup receive_tests -t

=====
HDLRegression version 0.20.0
Please see /doc/hdlregression.pdf documentation for more information.
=====

Scanning files...
Building test suite structure...

Starting simulations...
Moving previous test run to: ./hdlregression\test_2022-04-28_19.55.48.517420.
Running 5 out of 9 test(s) using 1 thread(s).
Running: bitvis_uart uart_vvc_tb func check_simple_receive
Generics: GC_TESTCASE=check_simple_receive
Result: PASS (0h:0m:1s).

Running: bitvis_uart uart_vvc_tb func check_single_simultaneous_transmit_and_receive
Generics: GC_TESTCASE=check_single_simultaneous_transmit_and_receive
Result: PASS (0h:0m:2s).

Running: bitvis_uart uart_vvc_tb func check_multiple_simultaneous_receive_and_read
Generics: GC_TESTCASE=check_multiple_simultaneous_receive_and_read
Result: PASS (0h:0m:2s).

Running: bitvis_uart uart_vvc_tb func skew_sbi_read_over_uart_receive
Generics: GC_TESTCASE=skew_sbi_read_over_uart_receive
Result: PASS (0h:0m:5s).

Running: bitvis_uart uart_vvc_tb func skew_sbi_read_over_uart_receive_with_delay_functionality
Generics: GC_TESTCASE=skew_sbi_read_over_uart_receive_with_delay_functionality
Result: PASS (0h:0m:4s).

Simulation run time: 0h:0m:15s.
SIMULATION SUCCESS: 5 passing test(s).
```

5.2.3 Pre-processing and simulations in parallel using 10 threads

All pre-processing steps and test case running is performed in parallel.

```
> python ../test/regression.py -tg receive_tests -t 10
```

```
$ python ../test/regression.py --testGroup receive_tests -t 10
=====
HDLRegression version 0.20.0
Please see /doc/hdlregression.pdf documentation for more information.
=====

Scanning files...
Building test suite structure...

Starting simulations...
Moving previous test run to: ./hdlregression\test_2022-04-28_19.57.27.209667.
Running 5 out of 9 test(s) using 5 thread(s).
Running: bitvis_uart uart_vvc_tb func check_multiple_simultaneous_receive_and_read
Generics: GC_TESTCASE=check_multiple_simultaneous_receive_and_read
Result: PASS (0h:0m:3s).

Running: bitvis_uart uart_vvc_tb func check_simple_receive
Generics: GC_TESTCASE=check_simple_receive
Result: PASS (0h:0m:4s).

Running: bitvis_uart uart_vvc_tb func check_single_simultaneous_transmit_and_receive
Generics: GC_TESTCASE=check_single_simultaneous_transmit_and_receive
Result: PASS (0h:0m:4s).

Running: bitvis_uart uart_vvc_tb func skew_sbi_read_over_uart_receive_with_delay_functionality
Generics: GC_TESTCASE=skew_sbi_read_over_uart_receive_with_delay_functionality
Result: PASS (0h:0m:8s).

Running: bitvis_uart uart_vvc_tb func skew_sbi_read_over_uart_receive
Generics: GC_TESTCASE=skew_sbi_read_over_uart_receive
Result: PASS (0h:0m:9s).

Simulation run time: 0h:0m:9s.
SIMULATION SUCCESS: 5 passing test(s).
```

5.3 Simulation results

Running simulations in terminal will output the necessary information, such as the test case name, generics used, simulation run time and result.

5.3.1 Regression initial run

```
$ python .../test/regression.py

=====
HDLRegression version 0.20.0
Please see /doc/hdlregression.pdf documentation for more information.
=====

Scanning files...
Building test suite structure...
Setting up: Y://bitvis//hdlregression//sim//hdlregression//library\modelsim.ini.
Compiling library: uvvm_util - OK -
Compiling library: uvvm_vvc_framework - OK -
Compiling library: bitvis_vip_scoreboard - OK -
Compiling library: bitvis_vip_sbi - OK -
Compiling library: bitvis_irqc - OK -
Compiling library: bitvis_vip_uart - OK -
Compiling library: bitvis_vip_clock_generator - OK -
Compiling library: bitvis_uart - OK -

Starting simulations...
Running 9 out of 9 test(s) using 1 thread(s).
Running: bitvis_uart uart_vvc_demo_tb.func
Result: PASS (0h:0m:8s).

Running: bitvis_uart uart_simple_bfm_tb.func
Result: PASS (0h:0m:1s).

Running: bitvis_uart uart_vvc_tb.func.check_register_defaults
Generics: GC_TESTCASE=check_register_defaults
Result: PASS (0h:0m:1s).

Running: bitvis_uart uart_vvc_tb.func.check_simple_transmit
Generics: GC_TESTCASE=check_simple_transmit
Result: PASS (0h:0m:1s).

Running: bitvis_uart uart_vvc_tb.func.check_simple_receive
Generics: GC_TESTCASE=check_simple_receive
Result: PASS (0h:0m:2s).

Running: bitvis_uart uart_vvc_tb.func.check_single_simultaneous_transmit_and_receive
Generics: GC_TESTCASE=check_single_simultaneous_transmit_and_receive
Result: PASS (0h:0m:2s).

Running: bitvis_uart uart_vvc_tb.func.check_multiple_simultaneous_receive_and_read
Generics: GC_TESTCASE=check_multiple_simultaneous_receive_and_read
Result: PASS (0h:0m:1s).

Running: bitvis_uart uart_vvc_tb.func.skew_sbi_read_over_uart_receive
Generics: GC_TESTCASE=skew_sbi_read_over_uart_receive
Result: PASS (0h:0m:6s).

Running: bitvis_uart uart_vvc_tb.func.skew_sbi_read_over_uart_receive_with_delay_functionality
Generics: GC_TESTCASE=skew_sbi_read_over_uart_receive_with_delay_functionality
Result: PASS (0h:0m:5s).

Simulation run time: 0h:0m:30s.
SIMULATION SUCCESS: 9 passing test(s).
```

5.3.2 Regression run without changes

No tests are run when no changes are detected in the DUT or test case files, unless full regression is enabled using the *Command Line Interface (CLI)* `-fr` or using the *Application Programming Interface (API)* in the regression script `hr.start(full_regression=True)`.

```
$ python ../test/regression.py

=====
HDLRegression version 0.20.0
Please see /doc/hdlregression.pdf documentation for more information.
=====

Scanning files...
Building test suite structure...

Starting simulations...
Test run not required. Use "-fr"/"--fullRegression" to re-run all tests.
-----
Test run completed with 0 failing tests:
-----
```

5.3.3 Failing test case run

A failing test case will be reported as **FAIL** with a short summary from the test log:

```
$ python ../test/regression.py
=====
 HDLRegression version 0.20.0
 Please see /doc/hdlregression.pdf documentation for more information.
=====

Scanning files...
Building test suite structure...
Compiling library: bitvis_uart - OK -

Starting simulations...
Moving previous test run to: ./hdlregression/test_2022-04-28_20.02.38.299482.
Running 9 out of 9 test(s) using 1 thread(s).
Running: bitvis_uart uart_vvc_demo_tb.func
Result: PASS (0h:0m:8s).

Running: bitvis_uart uart_simple_bfm_tb.func
Result: PASS (0h:0m:1s).

Running: bitvis_uart uart_vvc_tb.func.check_register_defaults
Generics: GC_TESTCASE=check_register_defaults
Result: PASS (0h:0m:1s).

Running: bitvis_uart uart_vvc_tb.func.check_simple_transmit
Generics: GC_TESTCASE=check_simple_transmit
Result: FAIL (0h:0m:1s).

=====
# UVM:
# UVM:
# UVM: ID_LOG_HDR          0.0 ns  TB seq.           Starting simulation of TB for UART using VVCs
# UVM: -----
# UVM: IDSEQUENCER          0.0 ns  TB seq.           Wait 10 clock period for reset to be turned off
# UVM:
# UVM:
# UVM: ID_LOG_HDR          100.0 ns TB seq.          Check simple transmit
# UVM: -----
# UVM: ID_UVM_SEND_CMD     100.0 ns TB seq.(uvm)      ->sbi_write(SBI_VVC,1, "x"2", "x"55"): 'TX_DATA'. [9]
# UVM: ID_UVM_SEND_CMD     100.0 ns TB seq.(uvm)      ->uart_expect(UART_VVC,1,rx, "x"5A"): 'Expecting data on UART RX'. [10]
# UVM: ID_BFM               112.5 ns SBI_VVC,1        sbi_write(A:x"2", X"55") completed. 'TX_DATA' [9]
# UVM:
# UVM: =====
# UVM: *** ERROR #1 ***
# UVM:   1810 ns  UART_VVC,1,RX
# UVM:         uart_expect(x"5A")=> Failed. Expected value x"5A" did not appear within 1 occurrences, received value x"55". 'Expecting data on UART RX' [10]
# UVM:
# UVM: Simulator has been paused as requested after 1 ERROR
# UVM: *** To find the root cause of this alert, step out the HDL calling stack in your simulator. ***
# UVM: *** For example, step out until you reach the call from the test sequencer. ***
# UVM: =====
# UVM:
# UVM: ** Note: stop
# UVM: Time: 1810 ns Iteration: 0 Instance: /uart_vvc_tb/i_testHarness/i1_uart_vvc/i1_uart_rx
# UVM: Break in Subprogram alert at //Mac/dev/bitvis/uvm/uvm_util/src/methods_pkg.vhd line 3702
# UVM: Stopped at //Mac/dev/bitvis/uvm/uvm_util/src/methods_pkg.vhd line 3702
# UVM: End time: 20:05:01 on Apr 28, 2022, Elapsed time: 0:00:01
# UVM: Errors: 0, Warnings: 1

=====
Running: bitvis_uart uart_vvc_tb.func.check_simple_receive
Generics: GC_TESTCASE=check_simple_receive
Result: PASS (0h:0m:2s).

Running: bitvis_uart uart_vvc_tb.func.check_single_simultaneous_transmit_and_receive
Generics: GC_TESTCASE=check_single_simultaneous_transmit_and_receive
Result: PASS (0h:0m:2s).

Running: bitvis_uart uart_vvc_tb.func.check_multiple_simultaneous_receive_and_read
Generics: GC_TESTCASE=check_multiple_simultaneous_receive_and_read
Result: PASS (0h:0m:2s).

Running: bitvis_uart uart_vvc_tb.func.skew_sbi_read_over_uart_receive
Generics: GC_TESTCASE=skew_sbi_read_over_uart_receive
Result: PASS (0h:0m:5s).

Running: bitvis_uart uart_vvc_tb.func.skew_sbi_read_over_uart_receive_with_delay_functionality
Generics: GC_TESTCASE=skew_sbi_read_over_uart_receive_with_delay_functionality
Result: PASS (0h:0m:4s).

Simulation run time: 0h:0m:30s.
SIMULATION FAIL: 9 tests run, 1 test(s) failed.
```

GRAPHICAL USER INTERFACE (GUI)

6.1 Modelsim

Sometimes debugging a design or test require the use of GUI (graphical user interface) and HDLRegression can run tests in GUI when called with the -g or --gui argument. When enabled, the regression script will open inside ModelSim/QuestaSim/Rivier-PRO GUI with a loaded test case ready to run.

```
> python ../test/regression.py -tc uart_vvc_tb.func.check_simple_transmit -g
```

HDLRegression in GUI mode provides a set of functions for compiling and running a test:

```
# -----
# - HDLRegression test runner -
# -----
#
# Script commands are:
#
#   s = Start simulation
#   r = Recompile changed and dependent files
#   ra = Recompile All and restart
#   ro = Recompile Only
#   rs = ReStart
#   rr = Restart and Run
#   h = Help (this menu)
#   q = Quit (this test run)
#   qc = Quit Completely (regression)
#
# Current test:
#   uart_vvc_tb.func.check_register_defaults
#
```

VSIM 2>

6.2 GHDL / NVC

GHDL and NVC does not have a GUI, but can create simulation waveform files that can be opened to have a graphical representation of the signals in a VCD format (Value Change Dump).

When HDLRegression is called with GUI arguments and running with GHDL/NVC simulator it will create `sim.vcd` files inside every test case run folder. The VCD files can then be opened in a graphical waveform viewer such as GTKWave.

```
> python ../test/regression.py -tc uart_vvc_tb.func.check_simple_transmit -g -s ghdl
> gtkwave ./hdlregression/test/uart_vvc_tb/54005228/sim.vcd &
```

TESTBENCH

For HDLRegression to extract the correct information from the testbench files, there are some code requirements that have to be fulfilled. This information is used by HDLRegression to detect

7.1 Prerequisites

This are the requirements of a HDLRegression supporting testbench:

- The testbench file has to have the HDLRegression pragma above the entity declaration:
 - Only the top testbench file can have the HDLRegression testbench pragma.
 - Each top level testbench file has to have the HDLRegression testbench pragma.
- A testbench simulation result report will have to match the `set_result_check_string()` to **PASS**, and the test run will **FAIL** if this string is not found in the simulation transcript.

 Note

UVVM `report_alert_counters(FINAL)` is the default method for verifying a passing or failing test, and will have to be added to the testbench if no other `check_string` is selected. See example testbench for suggested implementation.

VHDL testbench

```
--HDLRegression:TB
entity my_dut_tb is
```

Verilog testbench

```
//HDLRegression:TB
module my_dut_tb;
```

 Note

A testbench with multiple testcases requires the GC_TESTCASE generic (VHDL) or TESTCASE parameter (Verilog), and these should only be used in the top level testbench entity.

The testcase names will be included in simulation reports.

```
GC_TESTCASE : string := "" -- VHDL
parameter TESTCASE = "" // Verilog
```

Note that the GC_TESTCASE generic or TESTCASE parameter name can be changed using the `set testcase identifier name()` method.

7.2 Example Testbench

For HDLRegression to discover a VHDL file to be used as a testbench the only requirement is that the `--hdlregression:tb` (VHDL) or `//hdlregression:tb` (Verilog) pragma is present:

Listing 1: VHDL testbench example

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3 use IEEE.numeric_std.all;
4
5 library uvvm_util;
6 context uvvm_util.uvvm_util_context;
7
8 -- Include when using VVC:
9 -- library uvvm_vvc_framework;
10 -- use uvvm_vvc_framework.ti_vvc_framework_support_pkg.all;
11
12 --hdlregression:tb
13 entity tb_example is
14     generic (
15         GC_TESTCASE : string := "UVVM"
16     );
17 end entity tb_example;
18
19 architecture func of tb_example is
20
21     constant C_SCOPE      : string  := C_TB_SCOPE_DEFAULT;
22     constant C_CLK_PERIOD : time    := 10 ns;
23
24 begin
25
26     -----
27     -- Instantiate test harness
28     -----
29     -- i_test_harness : entity work.test_harness;
30
31     -----
32     -- PROCESS: p_main
33     -----
34
35     p_main: process
36
37         begin
38             -----
```

(continues on next page)

(continued from previous page)

```

39   -- Wait for UVVM to finish initialization
40   -----
41   -- await_uvvm_initialization(VOID);
42
43   -----
44   -- Set UVVM verbosity level
45   -----
46   -- enable_log_msg(ALL_MESSAGES);
47   disable_log_msg(ALL_MESSAGES);
48   enable_log_msg(ID_SEQUENCER);
49   enable_log_msg(ID_LOG_HDR);

50
51   -----
52   -- Test sequence
53   -----
54   log(ID_SEQUENCER, "Running testcase: " & GC_TESTCASE, C_SCOPE);
55
56   if GC_TESTCASE = "check_reset_defaults" then
57
58     -- reset checks
59
60   elsif GC_TESTCASE = "test_dut_write" then
61
62     -- write checks
63
64   elsif GC_TESTCASE = "test_dut_read" then
65
66     -- read checks
67
68   end if;
69
70   -----
71   -- Ending the simulation
72   -----
73   wait for 1000 ns;           -- to allow some time for completion
74   report_alert_counters(FINAL); -- Report final counters and print conclusion for_
75   simulation (Success/Fail)
76   log(ID_LOG_HDR, "SIMULATION COMPLETED", C_SCOPE);
77
78   -- Finish the simulation
79   std.env.stop;
80   wait; -- to stop completely
81   end process p_main;
82
83 end func;

```

Listing 2: Verilog testbench example

```

1 //hdlregression:tb
2 module tb_verilog #(
3   parameter TESTCASE = "DEFAULT"
4 );

```

(continues on next page)

(continued from previous page)

```
5      initial
6      begin
7          if (TESTCASE == "reset_test")
8              // reset checks
9
10         else if (TESTCASE == "write_test")
11             // write tests
12
13         else if (TESTCASE == "read_test")
14             // read tests
15
16     end
17
18 endmodule
```

TEMPLATE FILES

8.1 Basic usage

The HDLRegression package comes with a basic template file to ease the process of getting started for new users.

```
1 import sys
2 # ----- USER HDLRegression PATH -----
3 # If HDLRegression is not installed as a Python package (see doc)
4 # then uncomment the following line and set the path for
5 # the HDLRegression install folder :
6 #sys.path.append(<full_or_relative_path_to_hdlregression_install>)
7
8 # Import the HDLRegression module to the Python script:
9 from hdlregression import HDLRegression
10
11 # Define a HDLRegression item to access the HDLRegression functionality:
12 hr = HDLRegression()
13
14 # ----- USER CONFIG START -----
15 # => hr.add_files(<filename>) # Use default library my_work_lib
16 # => hr.add_files(<filename>, <library_name>) # or specify a library name.
17
18 # ----- USER CONFIG END -----
19 hr.start()
```

8.1.1 Basic example

```
1 import sys
2 # User specify HDLRegression install path:
3 sys.path.append('c:/tools/hdlregression/')
4
5 # Import the HDLRegression module to the Python script:
6 from hdlregression import HDLRegression
7
8 # Define a HDLRegression item to access the HDLRegression functionality:
9 hr = HDLRegression()
10
11 # ----- USER CONFIG START -----
```

(continues on next page)

(continued from previous page)

```

13 # Add all .vhd files in the /src directory to library my_dut_lib:
14 hr.add_files("./src/*.vhd", "my_dut_lib")
15
16 # Add testbech file to library my_tb_lib:
17 hr.add_files("./tb/my_tb.vhd", "my_tb_lib")
18
19 # ----- USER CONFIG END -----
20 hr.start()

```

8.2 Advanced usage

The HDLRegression package comes with an advanced template file for advanced users to extend with even more functionality.

```

1 import sys
2 # ----- USER HDLRegression PATH -----
3 # If HDLRegression is not installed as a Python package (see doc)
4 # then uncomment the following line and set the path for
5 # the HDLRegression install folder :
6 #sys.path.append(<full_or_relative_path_to_hdlregression_install>)
7
8 # Import the HDLRegression module to the Python script:
9 from hdlregression import HDLRegression
10
11 # ----- USER IMPORT -----
12 # Import other Python package(s):
13
14 # Define a HDLRegression item to access the HDLRegression functionality:
15 hr = HDLRegression()
16
17 # ----- USER CONFIG START -----
18
19 # Add Python functions here if needed:
20
21 # Add design files, repeat call if needed:
22 # => hr.add_files(<src_files>, <compile_library>)
23
24 # Add testbench and related files:
25 # => hr.add_files(<src_files>, <compile_library>)
26
27 # Define testbench configurations/generics if any, repeat call if needed:
28 # => hr.add_generics(entity=<testbench_name>, architecture=<architecture_name>, generics=
29   #<generics_list>
30
31 # Define simulation report format:
32 # => hr.gen_report() # default is full report (testbench, testcase, configurations, pass/
33   #<fail>) to report.txt
34
# ----- USER CONFIG END -----

```

(continues on next page)

(continued from previous page)

35 hr.start()

8.2.1 Advanced example

```

1  from itertools import product
2  from hdlregression import HDLRegression
3  import sys
4  # User specify HDLRegression install path:
5  sys.path.append('c:/tools/hdlregression/')
6
7  # Import the HDLRegression module to the Python script:
8
9  # Import other Python package(s):
10
11
12 # Define a HDLRegression item to access the HDLRegression functionality:
13 hr = HDLRegression()
14
15 # ----- USER CONFIG START -----
16
17 # Add Python functions here if needed:
18
19 # Return a list with the product of the generics
20
21
22 def create_generics(bus_width, master_mode, input_file, output_file):
23     generics = []
24     for bus_width, master_mode, input_file, output_file in product(bus_width, master_
25     mode, input_file, output_file):
26         generics.append(["GC_BUS_WIDTH",bus_width, "GC_MASTER_MODE",master_mode, "GC_"
27         "INPUT_FILE",input_file, "GC_OUTPUT_FILE",output_file])
28     return generics
29
30
31
32 # Add all source files to library my_dut_lib:
33 hr.add_files("./src/*.vhd", "my_dut_lib")
34
35
36 # Add all testbench related files to library my_tb_lib:
37 hr.set_library("my_tb_lib")
38 hr.add_files("./tb/my_dut_tb.vhd")
39 hr.add_files("./tb/my_dut_th.vhd")
40 hr.add_files("./tb/my_dut_if_stuck_tb.vhd")
41 hr.add_files("./tb/my_dut_pin_pulse_tb.vhd")
42
43
44 # Get a list with the product of selected generics:
45 generics = create_generics(bus_width=[2, 4, 8, 16], master_mode=[
46                         True, False], input_file="in_data.txt", output_file="out_data.
47                         txt")

```

(continues on next page)

(continued from previous page)

```

44
45 # Add generics for testbench run:
46 hr.add_generics(entity="my_dut_tb", generics=generics)
47
48 # Specify output report as CSV:
49 hr.gen_report(report_file="project_report.csv")
50
51 # ----- USER CONFIG END -----
52 hr.start(regression_mode=True)

```

8.2.2 RTL and Netlist script example

```

1 import sys
2 # ----- USER HDLRegression PATH -----
3 # If HDLRegression is not installed as a Python package (see doc)
4 # then uncomment the following line and set the path for
5 # the HDLRegression install folder :
6 #sys.path.append(<full_or_relative_path_to_HDLRegression_install>)
7
8 # Import the HDLRegression module to the Python script:
9 from hdlregression import HDLRegression
10
11
12 def run_rtl():
13     """
14         Setup test environment for RTL simulations.
15     """
16     # Define a HDLRegression item to access the HDLRegression functionality:
17     hr = HDLRegression()
18
19     # ----- USER CONFIG START -----
20     # => hr.add_files(<filename>)                      # Use default library my_work_lib
21     # => hr.add_files(<filename>, <library_name>)      # or specify a library name.
22
23     # ----- USER CONFIG END -----
24     hr.start()
25
26
27
28 def run_netlist():
29     """
30         Setup test environment for Netlist simulations.
31     """
32     # Define a HDLRegression item to access the HDLRegression functionality:
33     hr = HDLRegression()
34
35     # ----- USER CONFIG START -----
36     # => hr.add_files(<filename>)                      # Use default library my_work_lib
37     # => hr.add_files(<filename>, <library_name>)      # or specify a library name.
38

```

(continues on next page)

(continued from previous page)

```
39      # ----- USER CONFIG END -----
40      hr.start()
41
42
43  def main():
44      """
45          Main method, selecting RTL or Netlist simulations.
46      """
47
48      args = sys.argv[1:]
49
50      if len(args) > 0:
51          selection = args[0].lower()
52          sys.argv.remove(selection)
53
54          if 'rtl' == selection:
55              run_rtl()
56          elif 'netlist' == selection:
57              run_netlist()
58          else:
59              print('Please select "rtl" or "netlist" run.')
60      else:
61          print('Please select "rtl" or "netlist" run.')
62
63
64  if __name__ == "__main__":
65      main()
```


TEST AUTOMATION SERVER

HDLRegression support development automation server tools such as Jenkins and GitLab. When using HDLRegression with an automation server the test runner script will need to utilize the statistical method `get_num_fail_tests()` in HDLRegression and exit with an exit code to trigger a PASS/FAIL test in the automation server.

Example code - returning the number of failing tests to the automation server:

```
# run tests
ret_code = hr.start()
# exit with the return code from start()
sys.exit(ret_code)
```

 **Note**

The automation server will indicate a passing test when the test runner script returns ‘0’ exit code, and `start()` will return ‘0’ if all tests have passed and there are no compilation errors.

Example of building HDLRegression package and running test script in Jenkins:



The screenshot shows a Jenkins build configuration step titled "Execute shell". The "Command" field contains the following shell script:

```
python3 setup.py build
python3 setup.py develop --user
cd sim
python3 ../test/test.py -c -s ghdl
```

Below the command field, there is a link "See the list of available environment variables". At the bottom right of the step, there is a button labeled "Avansert...".

At the very bottom left of the configuration page, there is a button labeled "Add build step ▾".

GENERATED OUTPUT

When a HDLRegression regression script is run a folder *hdlregression* will be created in the same folder as the script was called from, e.g. *sim*. The folder will hold important project information in *.dat* files, a list of all run commands inside a *commands.do* file, library compilations inside a *library* folder, and test run outputs and report information inside a *test* folder. Note that each time the regression script is run it will back-up the *test* folder with a date-and-time suffix to ensure that no important test run results are overwritten.

- */library*
- */test*
- *commands.do*
- *library.dat*
- *settings.dat*
- *testgroup.dat*
- *testgroup_collection.dat*

 **Note**

The library folder will include one or more folders for the compiled libraries. The test folder will include one or more test case folders and - if selected - a coverage folder.

10.1 Test folder

Inside the */test* folder there can be several sub-folders and files. Each testbench entity will have a folder of its own which again has sub-folders for used architecture and generics. These *test run* folders have unique names that are hash generated, thus identifying a specific test run can be done by inspecting the test mapping file, *test_mapping.csv*.

Listing 1: HDLRegression output folder example

```
1 hdlregression
2   └── commands.do
3   └── generic.dat
4   └── library
5     └── bitvis_uart
6       ├── _info
7       ├── _lib.qdb
8       └── _lib1_0.qdb
```

(continues on next page)

(continued from previous page)

```
9      └── _lib1_0.qpg
10     └── _lib1_0.qtl
11     └── _vmake
12   └── bitvis_vip_clock_generator
13     ├── _info
14     ├── _lib.qdb
15     ├── _lib1_0.qdb
16     ├── _lib1_0.qpg
17     ├── _lib1_0.qtl
18     └── _vmake
19   └── bitvis_vip_sbi
20     ├── _info
21     ├── _lib.qdb
22     ├── _lib1_0.qdb
23     ├── _lib1_0.qpg
24     ├── _lib1_0.qtl
25     └── _vmake
26   └── bitvis_vip_scoreboard
27     ├── _info
28     ├── _lib.qdb
29     ├── _lib1_0.qdb
30     ├── _lib1_0.qpg
31     ├── _lib1_0.qtl
32     └── _vmake
33   └── bitvis_vip_uart
34     ├── _info
35     ├── _lib.qdb
36     ├── _lib1_0.qdb
37     ├── _lib1_0.qpg
38     ├── _lib1_0.qtl
39     └── _vmake
40   └── modelsim.ini
41   └── uvvm_util
42     ├── _info
43     ├── _lib.qdb
44     ├── _lib1_0.qdb
45     ├── _lib1_0.qpg
46     ├── _lib1_0.qtl
47     └── _vmake
48   └── uvvm_vvc_framework
49     ├── _info
50     ├── _lib.qdb
51     ├── _lib1_0.qdb
52     ├── _lib1_0.qpg
53     ├── _lib1_0.qtl
54     └── _vmake
55   └── library.dat
56   └── settings.dat
57   └── test
58     └── irqc_demo_tb
59       └── func_1
60         └── _Alert.txt
```

(continues on next page)

(continued from previous page)

```

61      └── _Log.txt
62
63      └── run.do
64      └── transcript
65
66  └── irqc_tb
67      └── func_2
68          ├── UVVM_Alert.txt
69          ├── UVVM_Log.txt
70          └── run.do
71          └── transcript
72
73  └── sim_report.json
74
75  └── test_mapping.csv
76
77  └── uart_simple_bfm_tb
78      └── func_4
79          ├── UVVM_Alert.txt
80          ├── UVVM_Log.txt
81          └── run.do
82          └── transcript
83
84  └── uart_vvc_demo_tb
85      └── func_3
86          ├── _Alert.txt
87          ├── _Log.txt
88          └── run.do
89          └── transcript
90
91  └── uart_vvc_tb
92      └── func_10
93          ├── run.do
94          ├── skew_sbi_read_over_uart_receive_Alert.txt
95          ├── skew_sbi_read_over_uart_receive_Log.txt
96          └── transcript
97
98  └── func_11
99      ├── run.do
100     ├── skew_sbi_read_over_uart_receive_with_delay_functionality_Alert.txt
101     ├── skew_sbi_read_over_uart_receive_with_delay_functionality_Log.txt
102     └── transcript
103
104  └── func_5
105      ├── check_register_defaults_Alert.txt
106      ├── check_register_defaults_Log.txt
107      └── run.do
108      └── transcript
109
110  └── func_6
111      ├── check_simple_transmit_Alert.txt
112      ├── check_simple_transmit_Log.txt
113      └── run.do
114      └── transcript
115
116  └── func_7
117      ├── check_simple_receive_Alert.txt
118      ├── check_simple_receive_Log.txt
119      └── run.do
120      └── transcript
121
122  └── func_8
123      ├── check_single_simultaneous_transmit_and_receive_Alert.txt
124      ├── check_single_simultaneous_transmit_and_receive_Log.txt

```

(continues on next page)

(continued from previous page)

```
113      └── run.do
114      └── transcript
115  func_9
116      ├── check_multiple_simultaneous_receive_and_read_Alert.txt
117      ├── check_multiple_simultaneous_receive_and_read_Log.txt
118      └── run.do
119      └── transcript
120
121  test_2024-01-10_14.51.25.645865
122      └── irqc_demo_tb
123          └── func_1
124              ├── _Alert.txt
125              ├── _Log.txt
126              └── run.do
127              └── transcript
128
129      └── irqc_tb
130          └── func_2
131              ├── UVVM_Alert.txt
132              ├── UVVM_Log.txt
133              └── run.do
134              └── transcript
135
136      └── sim_report.json
137
138      └── test_mapping.csv
139
140      └── uart_simple_bfm_tb
141          └── func_4
142              ├── UVVM_Alert.txt
143              ├── UVVM_Log.txt
144              └── run.do
145              └── transcript
146
147      └── uart_vvc_demo_tb
148          └── func_3
149              ├── _Alert.txt
150              ├── _Log.txt
151              └── run.do
152              └── transcript
153
154      └── uart_vvc_tb
155          └── func_10
156              ├── run.do
157              ├── skew_sbi_read_over_uart_receive_Alert.txt
158              ├── skew_sbi_read_over_uart_receive_Log.txt
159              └── transcript
160
161          └── func_11
162              ├── run.do
163              ├── skew_sbi_read_over_uart_receive_with_delay_functionality_Alert.txt
164              ├── skew_sbi_read_over_uart_receive_with_delay_functionality_Log.txt
165              └── transcript
166
167          └── func_5
168              ├── check_register_defaults_Alert.txt
169              ├── check_register_defaults_Log.txt
170              └── run.do
171              └── transcript
172
173          └── func_6
174              ├── check_simple_transmit_Alert.txt
```

(continues on next page)

(continued from previous page)

```

165      └── check_simple_transmit_Log.txt
166      └── run.do
167      └── transcript
168  func_7
169      ├── check_simple_receive_Alert.txt
170      ├── check_simple_receive_Log.txt
171      └── run.do
172      └── transcript
173  func_8
174      ├── check_single_simultaneous_transmit_and_receive_Alert.txt
175      ├── check_single_simultaneous_transmit_and_receive_Log.txt
176      └── run.do
177      └── transcript
178  func_9
179      ├── check_multiple_simultaneous_receive_and_read_Alert.txt
180      ├── check_multiple_simultaneous_receive_and_read_Log.txt
181      └── run.do
182      └── transcript
183  testcase.dat
184  testgroup.dat
185  testgroup_collection.dat

```

10.1.1 Test mapping

A test mapping file *test_mapping.csv* is located in every *test* folder to help identify test runs with test output folders. An example of the layout of a test mapping file is shown below:

Listing 2: *test_mapping.csv* example

```

1, ./hdlregression/test/irqc_demo_tb/func_1, bitvis_irqc.irqc_demo_tb(func)
2, ./hdlregression/test/irqc_tb/func_2, bitvis_irqc.irqc_tb(func)
3, ./hdlregression/test/uart_vvc_demo_tb/func_3, bitvis_uart.uart_vvc_demo_tb(func)
4, ./hdlregression/test/uart_simple_bfm_tb/func_4, bitvis_uart.uart_simple_bfm_tb(func)
5, ./hdlregression/test/uart_vvc_tb/func_5, bitvis_uart.uart_vvc_tb(func):GC_
  ↳ TESTCASE=check_register_defaults
6, ./hdlregression/test/uart_vvc_tb/func_6, bitvis_uart.uart_vvc_tb(func):GC_
  ↳ TESTCASE=check_simple_transmit
7, ./hdlregression/test/uart_vvc_tb/func_7, bitvis_uart.uart_vvc_tb(func):GC_
  ↳ TESTCASE=check_simple_receive
8, ./hdlregression/test/uart_vvc_tb/func_8, bitvis_uart.uart_vvc_tb(func):GC_
  ↳ TESTCASE=check_single_simultaneous_transmit_and_receive
9, ./hdlregression/test/uart_vvc_tb/func_9, bitvis_uart.uart_vvc_tb(func):GC_
  ↳ TESTCASE=check_multiple_simultaneous_receive_and_read
10, ./hdlregression/test/uart_vvc_tb/func_10, bitvis_uart.uart_vvc_tb(func):GC_
  ↳ TESTCASE=skew_sbi_read_over_uart_receive
11, ./hdlregression/test/uart_vvc_tb/func_11, bitvis_uart.uart_vvc_tb(func):GC_
  ↳ TESTCASE=skew_sbi_read_over_uart_receive_with_delay_functionality

```


11.1 Back annotated netlist simulations

Running RTL and Netlist simulations require two individual test runs, i.e. different HDLRegression instances, and solving this can be done using one or two regression scripts:

- Use two run scripts, e.g. `run_rtl.py` and `run_netlist.py`, and setup both scripts as individual runs, one running RTL simulations and the other running Netlist simulations.
- Combine both run scripts in a single file, e.g. `run_regression.py`, and use a selection mechanism inside the run script to select which run to execute.

 Note

The single runner script example will support HDLRegression CLI arguments when implemented with argument modifications as shown in the example below.

11.1.1 Regression script

Example of running RTL and Netlist from two runner scripts

```
python3 ../script/run_rtl.py  
python3 ../script/run_netlist.py
```

Example of running RTL and Netlist from a single runner script

```
python3 ../script/run_regression.py rtl  
python3 ../script/run_regression.py netlist
```

Example setup for running RTL and Netlist from a single runner script

```
1 import sys  
2 # ----- USER HDLRegression PATH -----  
3 # If HDLRegression is not installed as a Python package (see doc)  
4 # then uncomment the following line and set the path for  
5 # the HDLRegression install folder :  
6 #sys.path.append(<full_or_relative_path_to_HDLRegression_install>)  
7
```

(continues on next page)

(continued from previous page)

```
8 # Import the HDLRegression module to the Python script:
9 from hdlregression import HDLRegression
10
11
12 def run_rtl():
13     """
14     Setup test environment for RTL simulations.
15     """
16     # Define a HDLRegression item to access the HDLRegression functionality:
17     hr = HDLRegression()
18
19     # ----- USER CONFIG START -----
20     # => hr.add_files(<filename>)                      # Use default library my_work_lib
21     # => hr.add_files(<filename>, <library_name>)      # or specify a library name.
22
23     # ----- USER CONFIG END -----
24     hr.start()
25
26
27
28 def run_netlist():
29     """
30     Setup test environment for Netlist simulations.
31     """
32     # Define a HDLRegression item to access the HDLRegression functionality:
33     hr = HDLRegression()
34
35     # ----- USER CONFIG START -----
36     # => hr.add_files(<filename>)                      # Use default library my_work_lib
37     # => hr.add_files(<filename>, <library_name>)      # or specify a library name.
38
39     # ----- USER CONFIG END -----
40     hr.start()
41
42
43 def main():
44     """
45     Main method, selecting RTL or Netlist simulations.
46     """
47
48     args = sys.argv[1:]
49
50     if len(args) > 0:
51         selection = args[0].lower()
52         sys.argv.remove(selection)
53
54         if 'rtl' == selection:
55             run_rtl()
56         elif 'netlist' == selection:
57             run_netlist()
58         else:
59             print('Please select "rtl" or "netlist" run.')
```

(continues on next page)

(continued from previous page)

```
60     else:  
61         print('Please select "rtl" or "netlist" run.')  
62  
63  
64 if __name__ == "__main__":  
65     main()
```

