hdlregression

Release 0.42.0

UVVM

Jun 30, 2023

CONTENTS:

CHAPTER

ONE

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 Testcase

A testcase 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 testcases.

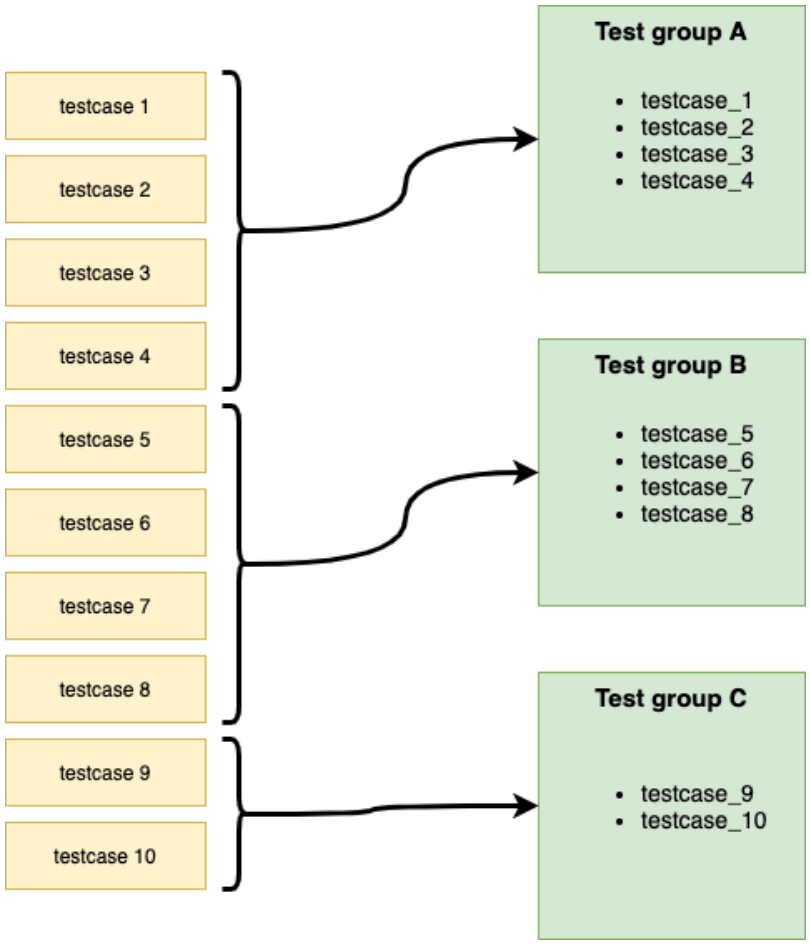
1.3.6 Test group

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

A test group can be:

- A single testbench or a collection of testbenches
- A single testcase or a collection of testcases

Typically a test group contains testbenches and/or testcases 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 testcases in the testbench (a testcase generic is required for this feature)
- Compile to the default or a specified library
- Run all testbenches
- Report simulation results to terminal

CHAPTER

THREE

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.

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

# Import the HDLRegression module to the Python script:
from hdlregression import HDLRegression

# Define a HDLRegression item to access the HDLRegression functionality:
hr = HDLRegression()

# ------------ USER CONFIG START ---------------
# => hr.add_files(<filename>)                    # Use default library my_work_lib
# => hr.add_files(<filename>, <library_name>)    # or specify a library name.

# ------------ USER CONFIG END ---------------
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.

The default simulator is Modelsim, but this can be changed 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”	“modelsim”	optional
arg_parser	argparser obj	regression_parser	None	optional

Example 1:

```
1. hr = HDLRegression()

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

An argparser 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 argparser object is passed on to HDLRegression it will add all its arguments to the argparser 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_test_harness/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 `True`.

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` HDLRregression 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. testcases and filenames.

Pattern	Meaning
*	match all
?	match a single charecter
[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 testcase(s) has been run without errors	0
No compilation error and no testcase run	1
No compilation error and testcase run with one or more errors	1
Compilation error (no testcases 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>)
```

Argument	Options & Type	Default	Required
gui_mode	True/False (boolean)	False	optional
regression_mode	True/False (boolean)	True	optional
stop_on_failure	True/False (boolean)	False	optional
threading	True/False (boolean)	False	optional
sim_options	string/list of string	None	optional
netlist_timing	string	None	optional
keep_code_coverage	True/False (boolean)	False	optional
no_default_com_options	True/False (boolean)	False	optional
ignore_simulator_exit_codes	list of int	[]	optional

Example:

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

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

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 simplyfies 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 testcases in one of these modes:

1. testcases added by the *add_testcase()* method or using the *command line interfaces*.
2. regression mode.

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

4.3 Advanced methods

4.3.1 add_generics()

Selects the generics to be used when running a testcase. A test run is created when generics are added to a testcase, 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])

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.

4.3.2 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**

4.3.3 add_testcase()

Adding testcase(s) will configure HDLRegression to only run these testcases. All testcases are run if no testcases are added using this command. Selecting testcase can also be done by using *command line interfaces*, and testcases selected from CLI will override any scripted testcase selecition.

Important: A testcase name is a string that consists of

1. testbench entity name
2. testbench architecture name
3. test name (optional)

And where the three testcase name elements are separated by a dot (.): `<testbench_name>.<architecture_name>.<test_name>`.

Tip: Use wildcards to more effectively filter searches, i.e. testcases and filenames.

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

```
add_testcase(<testcase>)
```

Argument	Type	Required
testcase	string / list of strings	**mandatory**

Example:

```
add_testcase("interface_tb.test_arch.read_test") # this test only

add_testcase("interface_tb.*.read_*") # all architectures and all sequencer testcases␣
↪starting with 'read'

add_testcase("interface_tb.test_arch.????_test") # all sequencer testcases mathing any 4␣
↪character start, followed by '_test'

add_testcase(interface_tests_list) # a list of selected tests
```

Note: The *start()* method will return error code 1 if no testcases matched the `testcase` keyword.

4.3.4 add_to_testgroup()

Will add tests to a collection of tests, i.e. *test group*, that can be run in groups. The test group is given a name that is used for addressing the test collection.
There are no limit for how many tests that can be added to a test group, and no limit for the number of test groups.
Running a test group can be done using a *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
testcase	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 `testcase` argument is for selecting sequencer built-in testcases.
- The *start()* method will return error code 1 if no test group or testcase were found.

Tip: Use wildcards to more effectively filter searches, i.e. testcases and filenames.

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

4.3.5 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.

4.3.6 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.

4.3.7 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	*library name* (string)	"can_ip_library"	**mandatory**
never_recompile	True/False (boolean)	True	optional
set_lib_dep	*library name* (string)	"ip_library"	optional

Example:

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

4.3.8 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	*filename* (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.csv", compile_order=True, library=True)
```

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

4.3.9 get_args()

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

Example:

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

4.3.10 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()
```

4.3.11 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.

4.3.12 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.

4.3.13 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 `exlude_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")
```

4.3.14 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).

4.3.15 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 testcase result string

```
p_seq : process
    variable v_check_ok : boolean := false;
begin
    -- testcase checks, e.g.
    -- v_check_ok := check_value(v_act_data, v_exp_data, error, "checking receive data", C_SCOPE);

    if v_check_ok = true then
        report "testcase passed";
    end if;

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

4.3.16 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	Default	Required
simulator	*simulator name* (string)	"MODELSIM", "GHDL", "NVC"	"MODELSIM"	**mandatory**
simulator_path	*simulator_executable_path* (string)	"c:/ghdl/bin"		optional
com_options	*compile optionss* (string/list of string)	"-suppress 1346,1236,1090 -2008"		optional

Example:

```
hr.set_simulator(simulator="GHDL")

hr.set_simulator(simulator="MODELSIM", 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.

4.3.17 set_testcase_identifier_name()

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

Note:

- The sequencer built-in testcase if-structure will need to match this generic.
- HDLRegression extracts all the sequencer built-in testcases 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 testcase ID generic

```
--hdlregression:tb
entity tb_top is
    generic (
        GC_TESTCASE : string := "read_test"
    );
end tb_top;

architecture test of simple_tb is
    constant C_SCOPE : string := "SIMPLE_TB";
  begin
    p_main : process
    begin
        if GC_TESTCASE = "read_test" then
            -- read tests
        elsif GC_TESTCASE = "write_test" then
            -- write tests
        else
            report "Unknown test " & GC_TESTCASE;
        end if;
    -- Finish the simulation
    std.env.stop;
    wait;  -- to stop completely
    end process;
end;
```

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()
```

CHAPTER

FIVE

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 TB_ENTITY [TB_ARCH [TC]]	–testCase TB_ENTITY [TB_ARCH [TC]]	Run selected testcase
-tg TESTGROUP	–testGroup TESTGROUP	Run selected test group
-ltc	–listTestcase	List all discovered testcases
-ltg	–listTestgroup	List all test groups
-lco	–listCompileOrder	List libraries and files in compile order
-fc	–forceCompile	Force recompile
-sof	–stopOnFailure	Stop simulations on testcase 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.

5.1 Examples

5.1.1 Full regression

Enabling the *full regression mode* ensures that all testcases 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).
```

5.1.2 Testcases

All tests that are discovered by HDLRegression can be listed using the `-ltc` or `--listTestcase` argument, and are listed as `<testbench entity>.<testbench architecture>.<sequencer built-in testcase>` or just `<testbench entity>.<testbench architecture>` if no sequencer built-in testcases are used.

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

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

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


Scanning files...
Building test suite structure...
1 - uart_vvc_demo_tb.func
2 - uart_simple_bfm_tb.func
3 - uart_vvc_tb.func.check_register_defaults
    Generics: GC_TESTCASE=check_register_defaults
4 - uart_vvc_tb.func.check_simple_transmit
    Generics: GC_TESTCASE=check_simple_transmit
5 - uart_vvc_tb.func.check_simple_receive
    Generics: GC_TESTCASE=check_simple_receive
6 - uart_vvc_tb.func.check_single_simultaneous_transmit_and_receive
    Generics: GC_TESTCASE=check_single_simultaneous_transmit_and_receive
7 - uart_vvc_tb.func.check_multiple_simultaneous_receive_and_read
    Generics: GC_TESTCASE=check_multiple_simultaneous_receive_and_read
8 - uart_vvc_tb.func.skew_sbi_read_over_uart_receive
    Generics: GC_TESTCASE=skew_sbi_read_over_uart_receive
9 - 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
```

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

---

Tip: Use wildcards to more effectively filter searches, i.e. testcases and filenames.

Pattern	Meaning
*	match all
?	match a single charecter
[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
```

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

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

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

================================================================================
  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_19.50.17.948546.
Running 1 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).

Simulation run time: 0h:0m:1s.
SIMULATION SUCCESS: 1 passing test(s).
```

Tip: Testcases are identified by:

1. <entity_name>
2. <entity_name>.<architecture_name>
3. <entity_name>.<architecture_name>.<sequencer_testcase>

When selecting testcases 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 testcases 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.

5.1.3 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 testcases that have *transmit* and *receive* in the testcase name, and is defined in testbench *uart_vvc_tb* architecture *func*. There is also a test group *selection_tests* that will run all testcases 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. testcases and filenames.

Pattern	Meaning
*	match all
?	match a single charecter
[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 testcase 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 testcase 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 testcase 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 testcase 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 testcase 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 testcase 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 testcase 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 testcase run

A failing testcase 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).


=============================================================================================================================================================
# UVVM:
# UVVM:
# UVVM: ID_LOG_HDR                        0.0 ns  TB seq.                     Starting simulation of TB for UART using VVCs
# UVVM: ------------------------------------------------------------------------------------------------------------------------------------------------------
# UVVM: ID_SEQUENCER                      0.0 ns  TB seq.                     Wait 10 clock period for reset to be turned off
# UVVM:
# UVVM:
# UVVM: ID_LOG_HDR                      100.0 ns  TB seq.                     Check simple transmit
# UVVM: ------------------------------------------------------------------------------------------------------------------------------------------------------
# UVVM: ID_UVVM_SEND_CMD                100.0 ns  TB seq.(uvvm)               ->sbi_write(SBI_VVC,1, x"2", x"55"): 'TX_DATA'. [9]
# UVVM: ID_UVVM_SEND_CMD                100.0 ns  TB seq.(uvvm)               ->uart_expect(UART_VVC,1,rx, x"5A"): 'Expecting data on UART RX'. [10]
# UVVM: ID_BFM                          112.5 ns  SBI_VVC,1                   sbi_write(A:x"2", x"55") completed. 'TX_DATA'  [9]
# UVVM:
# UVVM: ============================================================================================================================================
# UVVM: ***  ERROR #1  ***
# UVVM:          1810 ns   UART_VVC,1,RX
# UVVM:                    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]
# UVVM:
# UVVM: Simulator has been paused as requested after 1 ERROR
# UVVM: *** To find the root cause of this alert, step out the HDL calling stack in your simulator. ***
# UVVM: *** For example, step out until you reach the call from the test sequencer. ***
# UVVM: ============================================================================================================================================
# UVVM:
# UVVM:
# ** Note: stop
#    Time: 1810 ns  Iteration: 0  Instance: /uart_vvc_tb/i_test_harness/i1_uart_vvc/i1_uart_rx
# Break in Subprogram alert at //Mac/dev/bitvis/uvvm/uvvm_util/src/methods_pkg.vhd line 3702
# Stopped at //Mac/dev/bitvis/uvvm/uvvm_util/src/methods_pkg.vhd line 3702
# End time: 20:05:01 on Apr 28,2022, Elapsed time: 0:00:01
# 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.
```

CHAPTER

SIX

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` arugument. When enabled, the regression script will open inside Modelsim GUI with a loaded testcase 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

GHDL 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 runnning with GHDL simulator it will create `sim.vcd` files inside every testcase run folder. The VCD files can then be opened in a graphical wavefarm 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 &
```

CHAPTER

SEVEN

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

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

library uvvm_util;
context uvvm_util.uvvm_util_context;

-- Include when using VVC:
-- library uvvm_vvc_framework;
-- use uvvm_vvc_framework.ti_vvc_framework_support_pkg.all;

--hdlregression:tb
entity tb_example is
  generic (
    GC_TESTCASE : string := "UVVM"
    );
end entity tb_example;

architecture func of tb_example is

  constant C_SCOPE      : string  := C_TB_SCOPE_DEFAULT;
  constant C_CLK_PERIOD : time := 10 ns;

begin

  ------------------------------------------------------------------------------
  -- Instantiate test harness
  ------------------------------------------------------------------------------
  -- i_test_harness : entity work.test_harness;


  ------------------------------------------------
  -- PROCESS: p_main
  ------------------------------------------------
  p_main: process

  begin
    ------------------------------------------------------------------------------
    -- Wait for UVVM to finish initialization
    ------------------------------------------------------------------------------
    -- await_uvvm_initialization(VOID);

    ------------------------------------------------------------------------------
    -- Set UVVM verbosity level
    ------------------------------------------------------------------------------
    -- enable_log_msg(ALL_MESSAGES);
```

(continues on next page)

(continued from previous page)

```
    disable_log_msg(ALL_MESSAGES);
    enable_log_msg(ID_SEQUENCER);
    enable_log_msg(ID_LOG_HDR);

    ---------------------------------------------------------------------------
    -- Test sequence
    ---------------------------------------------------------------------------
    log(ID_SEQUENCER, "Running testcase: " & GC_TESTCASE, C_SCOPE);

    if GC_TESTCASE = "check_reset_defaults" then

      -- reset checks

    elsif GC_TESTCASE = "test_dut_write" then

      -- write checks

    elsif GC_TESTCASE = "test_dut_read" then

      -- read checks

    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
  end process p_main;

end func;
```

Listing 2: Verilog testbench example

```
//hdlregression:tb
module tb_verilog #(
        parameter TESTCASE = "DEFAULT"
    );

    initial
    begin
        if (TESTCASE == "reset_test")
            // reset checks

        else if (TESTCASE == "write_test")
            // write tests
```

(continues on next page)

(continued from previous page)

```
13
14          else if (TESTCASE == "read_test")
15              // read tests
16
17      end
18
19  endmodule
```

CHAPTER

EIGHT

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.

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

# Import the HDLRegression module to the Python script:
from hdlregression import HDLRegression

# Define a HDLRegression item to access the HDLRegression functionality:
hr = HDLRegression()

# ------------ USER CONFIG START ---------------
# => hr.add_files(<filename>)                    # Use default library my_work_lib
# => hr.add_files(<filename>, <library_name>)    # or specify a library name.

# ------------ USER CONFIG END ---------------
hr.start()
```

8.1.1 Basic example

```
import sys
# User specify HDLRegression install path:
sys.path.append('c:/tools/hdlregression/')

# Import the HDLRegression module to the Python script:
from hdlregression import HDLRegression

# Define a HDLRegression item to access the HDLRegression functionality:
hr = HDLRegression()

# ------------ USER CONFIG START ---------------

```

(continues on next page)

(continued from previous page)

```
# Add all .vhd files in the /src directory to library my_dut_lib:
hr.add_files("./src/*.vhd", "my_dut_lib")

# Add testbech file to library my_tb_lib:
hr.add_files("./tb/my_tb.vhd", "my_tb_lib")

# ------------ USER CONFIG END ---------------
hr.start()
```

8.2 Advanced usage

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

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

# Import the HDLRegression module to the Python script:
from hdlregression import HDLRegression

# ----------- USER IMPORT -----------------
# Import other Python package(s):

# Define a HDLRegression item to access the HDLRegression functionality:
hr = HDLRegression()

# ------------ USER CONFIG START ---------------

# Add Python functions here if needed:

# Add design files, repeat call if needed:
# => hr.add_files(<src_files>, <compile_library>)

# Add testbench and related files:
# => hr.add_files(<src_files>, <compile_library>)

# Define testbench configurations/generics if any, repeat call if needed:
# => hr.add_generics(entity=<testbench_name>, architecture=<architecture_name>, generics=
↪<generics_list>)

# Define simulation report format:
# => hr.gen_report() # default is full report (testbench, testcase, configurations, pass/
↪fail) to report.txt


# ------------ USER CONFIG END ---------------
```

(continues on next page)

(continued from previous page)

```
hr.start()
```

8.2.1 Advanced example

```
from itertools import product
from hdlregression import HDLRegression
import sys
# User specify HDLRegression install path:
sys.path.append('c:/tools/hdlregression/')

# Import the HDLRegression module to the Python script:

# Import other Python package(s):


# Define a HDLRegression item to access the HDLRegression functionality:
hr = HDLRegression()

# ------------ USER CONFIG START ---------------

# Add Python functions here if needed:

# Return a list with the product of the generics


def create_generics(bus_width, master_mode, input_file, output_file):
    generics = []
    for bus_width, master_mode, input_file, output_file in product(bus_width, master_
↪mode, input_file, output_file):
        generics.append(["GC_BUS_WIDTH",bus_width, "GC_MASTER_MODE",master_mode, "GC_
↪INPUT_FILE",input_file, "GC_OUTPUT_FILE",output_file])
    return generics


# Add all source files to library my_dut_lib:
hr.add_files("./src/*.vhd", "my_dut_lib")


# Add all testbench related files to library my_tb_lib:
hr.set_library("my_tb_lib")
hr.add_files("./tb/my_dut_tb.vhd")
hr.add_files("./tb/my_dut_th.vhd")
hr.add_files("./tb/my_dut_if_stuck_tb.vhd")
hr.add_files("./tb/my_dut_pin_pulse_tb.vhd")


# Get a list with the product of selected generics:
generics = create_generics(bus_width=[2, 4, 8, 16], master_mode=[
                           True, False], input_file="in_data.txt", output_file="out_data.
↪txt")
```

(continues on next page)

(continued from previous page)

```

# Add generics for testbench run:
hr.add_generics(entity="my_dut_tb", generics=generics)

# Specify output report as CSV:
hr.gen_report(report_file="project_report.csv")

# ------------ USER CONFIG END ---------------
hr.start(regression_mode=True)
```

8.2.2 RTL and Netlist script example

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

# Import the HDLRegression module to the Python script:
from hdlregression import HDLRegression


def run_rtl():
    '''
    Setup test environment for RTL simulations.
    '''
    # Define a HDLRegression item to access the HDLRegression functionality:
    hr = HDLRegression()

    # ------------ USER CONFIG START ---------------
    # => hr.add_files(<filename>)                   # Use default library my_work_lib
    # => hr.add_files(<filename>, <library_name>)   # or specify a library name.

    # ------------ USER CONFIG END ---------------
    hr.start()



def run_netlist():
    '''
    Setup test environment for Netlist simulations.
    '''
    # Define a HDLRegression item to access the HDLRegression functionality:
    hr = HDLRegression()

    # ------------ USER CONFIG START ---------------
    # => hr.add_files(<filename>)                   # Use default library my_work_lib
    # => hr.add_files(<filename>, <library_name>)   # or specify a library name.

```

(continues on next page)

(continued from previous page)

```
    # ------------ USER CONFIG END ---------------
    hr.start()


def main():
    '''
    Main method, selecting RTL or Netlist simulations.
    '''

    args = sys.argv[1:]

    if len(args) > 0:
        selection = args[0].lower()
        sys.argv.remove(selection)

        if 'rtl' == selection:
            run_rtl()
        elif 'netlist' == selection:
            run_netlist()
        else:
            print('Please select "rtl" or "netlist" run.')
    else:
        print('Please select "rtl" or "netlist" run.')


if __name__ == "__main__":
    main()
```

CHAPTER

NINE

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 runnner 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:

CHAPTER

TEN

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 testcase folders and - if selected - a coverage folder.

10.1 Test folder

Inside thet */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

```
hdlregression
├── commands.do
├── generic.dat
├── library
│   ├── bitvis_uart
│   │   ├── _info
│   │   ├── _lib.qdb
│   │   ├── _lib1_0.qdb
│   │   ├── _lib1_0.qpg
│   │   ├── _lib1_0.qtl
```

(continues on next page)

(continued from previous page)

```
│   │   └── _vmake
│   ├── bitvis_vip_clock_generator
│   │   ├── _info
│   │   ├── _lib.qdb
│   │   ├── _lib1_0.qdb
│   │   ├── _lib1_0.qpg
│   │   ├── _lib1_0.qtl
│   │   └── _vmake
│   ├── bitvis_vip_sbi
│   │   ├── _info
│   │   ├── _lib.qdb
│   │   ├── _lib1_0.qdb
│   │   ├── _lib1_0.qpg
│   │   ├── _lib1_0.qtl
│   │   └── _vmake
│   ├── bitvis_vip_scoreboard
│   │   ├── _info
│   │   ├── _lib.qdb
│   │   ├── _lib1_0.qdb
│   │   ├── _lib1_0.qpg
│   │   ├── _lib1_0.qtl
│   │   └── _vmake
│   ├── bitvis_vip_uart
│   │   ├── _info
│   │   ├── _lib.qdb
│   │   ├── _lib1_0.qdb
│   │   ├── _lib1_0.qpg
│   │   ├── _lib1_0.qtl
│   │   └── _vmake
│   ├── modelsim.ini
│   ├── uvvm_util
│   │   ├── _info
│   │   ├── _lib.qdb
│   │   ├── _lib1_0.qdb
│   │   ├── _lib1_0.qpg
│   │   ├── _lib1_0.qtl
│   │   └── _vmake
│   └── uvvm_vvc_framework
│       ├── _info
│       ├── _lib.qdb
│       ├── _lib1_0.qdb
│       ├── _lib1_0.qpg
│       ├── _lib1_0.qtl
│       └── _vmake
├── library.dat
├── settings.dat
├── test
│   ├── sim_report.json
│   ├── test_mapping.csv
│   ├── uart_simple_bfm_tb
│   │   └── 70910381
│   │       ├── UVVM_Alert.txt
```

(continues on next page)

(continued from previous page)

```
│   │   │   ├── UVVM_Log.txt
│   │   │   ├── run.do
│   │   │   └── transcript
│   │   ├── uart_vvc_demo_tb
│   │   │   └── 70910381
│   │   │       ├── _Alert.txt
│   │   │       ├── _Log.txt
│   │   │       ├── run.do
│   │   │       └── transcript
│   │   └── uart_vvc_tb
│   │       ├── 274144510
│   │       │   ├── run.do
│   │       │   ├── skew_sbi_read_over_uart_receive_with_delay_functionality_Alert.txt
│   │       │   ├── skew_sbi_read_over_uart_receive_with_delay_functionality_Log.txt
│   │       │   └── transcript
│   │       ├── 305729147
│   │       │   ├── check_simple_transmit_Alert.txt
│   │       │   ├── check_simple_transmit_Log.txt
│   │       │   ├── run.do
│   │       │   └── transcript
│   │       ├── 3122926221
│   │       │   ├── run.do
│   │       │   ├── skew_sbi_read_over_uart_receive_Alert.txt
│   │       │   ├── skew_sbi_read_over_uart_receive_Log.txt
│   │       │   └── transcript
│   │       ├── 3532724195
│   │       │   ├── check_multiple_simultaneous_receive_and_read_Alert.txt
│   │       │   ├── check_multiple_simultaneous_receive_and_read_Log.txt
│   │       │   ├── run.do
│   │       │   └── transcript
│   │       ├── 50272463
│   │       │   ├── check_single_simultaneous_transmit_and_receive_Alert.txt
│   │       │   ├── check_single_simultaneous_transmit_and_receive_Log.txt
│   │       │   ├── run.do
│   │       │   └── transcript
│   │       ├── 54005228
│   │       │   ├── check_simple_receive_Alert.txt
│   │       │   ├── check_simple_receive_Log.txt
│   │       │   ├── run.do
│   │       │   └── transcript
│   │       └── 796266300
│   │           ├── check_register_defaults_Alert.txt
│   │           ├── check_register_defaults_Log.txt
│   │           ├── run.do
│   │           └── transcript
├── test_2022-01-27_15.32.55.452648
│   ├── sim_report.json
│   ├── test_mapping.csv
│   ├── uart_simple_bfm_tb
│   │   └── 70910381
│   │       ├── UVVM_Alert.txt
│   │       ├── UVVM_Log.txt
```

(continues on next page)

(continued from previous page)

```
│   │   │   ├── run.do
│   │   │   └── transcript
│   │   ├── uart_vvc_demo_tb
│   │   │   └── 70910381
│   │   │       ├── _Alert.txt
│   │   │       ├── _Log.txt
│   │   │       ├── run.do
│   │   │       └── transcript
│   │   └── uart_vvc_tb
│   │       ├── 274144510
│   │       │   ├── run.do
│   │       │   ├── skew_sbi_read_over_uart_receive_with_delay_functionality_Alert.txt
│   │       │   ├── skew_sbi_read_over_uart_receive_with_delay_functionality_Log.txt
│   │       │   └── transcript
│   │       ├── 305729147
│   │       │   ├── check_simple_transmit_Alert.txt
│   │       │   ├── check_simple_transmit_Log.txt
│   │       │   ├── run.do
│   │       │   └── transcript
│   │       ├── 3122926221
│   │       │   ├── run.do
│   │       │   ├── skew_sbi_read_over_uart_receive_Alert.txt
│   │       │   ├── skew_sbi_read_over_uart_receive_Log.txt
│   │       │   └── transcript
│   │       ├── 3532724195
│   │       │   ├── check_multiple_simultaneous_receive_and_read_Alert.txt
│   │       │   ├── check_multiple_simultaneous_receive_and_read_Log.txt
│   │       │   ├── run.do
│   │       │   └── transcript
│   │       ├── 50272463
│   │       │   ├── check_single_simultaneous_transmit_and_receive_Alert.txt
│   │       │   ├── check_single_simultaneous_transmit_and_receive_Log.txt
│   │       │   ├── run.do
│   │       │   └── transcript
│   │       ├── 54005228
│   │       │   ├── check_simple_receive_Alert.txt
│   │       │   ├── check_simple_receive_Log.txt
│   │       │   ├── run.do
│   │       │   └── transcript
│   │       └── 796266300
│   │           ├── check_register_defaults_Alert.txt
│   │           ├── check_register_defaults_Log.txt
│   │           ├── run.do
│   │           └── transcript
├── testgroup.dat
└── 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

```
./hdlregression/test/uart_simple_bfm_tb/70910381, bitvis_uart.uart_simple_bfm_tb(func)
./hdlregression/test/uart_vvc_tb/1419120764, bitvis_uart.uart_vvc_tb(func): gc_
↪testcase=check_register_defaults
./hdlregression/test/uart_vvc_tb/886640571, bitvis_uart.uart_vvc_tb(func): gc_
↪testcase=check_simple_transmit
./hdlregression/test/uart_vvc_tb/613945132, bitvis_uart.uart_vvc_tb(func): gc_
↪testcase=check_simple_receive
./hdlregression/test/uart_vvc_tb/1155471887, bitvis_uart.uart_vvc_tb(func): gc_
↪testcase=check_single_simultaneous_transmit_and_receive
./hdlregression/test/uart_vvc_tb/301996323, bitvis_uart.uart_vvc_tb(func): gc_
↪testcase=check_multiple_simultaneous_receive_and_read
./hdlregression/test/uart_vvc_tb/3913552845, bitvis_uart.uart_vvc_tb(func): gc_
↪testcase=skew_sbi_read_over_uart_receive
./hdlregression/test/uart_vvc_tb/1589059134, bitvis_uart.uart_vvc_tb(func): gc_
↪testcase=skew_sbi_read_over_uart_receive_with_delay_functionality
./hdlregression/test/uart_vvc_demo_tb/70910381, bitvis_uart.uart_vvc_demo_tb(func)
```

CHAPTER

ELEVEN

TIPS

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

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

# Import the HDLRegression module to the Python script:
```

(continues on next page)

(continued from previous page)

```
from hdlregression import HDLRegression


def run_rtl():
    '''
    Setup test environment for RTL simulations.
    '''
    # Define a HDLRegression item to access the HDLRegression functionality:
    hr = HDLRegression()

    # ------------ USER CONFIG START ---------------
    # => hr.add_files(<filename>)                   # Use default library my_work_lib
    # => hr.add_files(<filename>, <library_name>)   # or specify a library name.

    # ------------ USER CONFIG END ---------------
    hr.start()



def run_netlist():
    '''
    Setup test environment for Netlist simulations.
    '''
    # Define a HDLRegression item to access the HDLRegression functionality:
    hr = HDLRegression()

    # ------------ USER CONFIG START ---------------
    # => hr.add_files(<filename>)                   # Use default library my_work_lib
    # => hr.add_files(<filename>, <library_name>)   # or specify a library name.

    # ------------ USER CONFIG END ---------------
    hr.start()


def main():
    '''
    Main method, selecting RTL or Netlist simulations.
    '''

    args = sys.argv[1:]

    if len(args) > 0:
        selection = args[0].lower()
        sys.argv.remove(selection)

        if 'rtl' == selection:
            run_rtl()
        elif 'netlist' == selection:
            run_netlist()
        else:
            print('Please select "rtl" or "netlist" run.')
    else:
```

(continues on next page)

(continued from previous page)

```
        print('Please select "rtl" or "netlist" run.')


if __name__ == "__main__":
    main()
```