SIMU Documentation

SIMU is an HDL simulation framework for FPGA development.

Hardware Description Language (HDL) simulation is a specific task requiring many tools to work together and usually everyone developing an FPGA is developing a custom simulation environment. This is not very efficient and over time it becomes quite tedious. For this reason the simu framework was developed, with the added flexibility to support different simulation tool vendors and different FPGA vendors. In a custom environment it is not often the case to have the ability to support different simulators and FPGA vendors.

Why is SIMU so good:

- short setup time and learning curve, because we don't like wasting time
- sticking to simple language constructs for fast ramp-up
- promote all good practices in the field of verification - regression, randomization, scalability, coverage, etc.
- flexible scripting structure to switch quickly between different OS, HDL languages and simulation tool vendors

Table of contents

simu-documenation exported as a single .md file

simu-documenation exported as a .pdf

Releases

Release 5.0.a - alpha - 2024-06-12

Alpha release:

- many features are not well tested
- somewhat significant architecture change
- added universal compile/optimize/simulation/coverage calls to enable simulator seamless switching and easy addition of new simulators
- added non vendor TCL shell simulation calls
- added easy integration of FPGA vendor exported simulation scripts
- simplified compile scripts
- unified simulation calls
- added manual regression

Release 4.6 - stable - 2018-31-10

Stable release:

- almost all features are tested and functional# Installation

Copy 'symphony/dev/sim' folder to your project simulation folder.

That's it.

The 'sim' folder contains all scripts and libraries you will need to run HDL simulations.# Features

The previously supported features will be maintained in the new SIMU reversion, unless explicitly mentioned that the support is obsolete.

Test-case Simulation Features Supported

TC Sim Feature	SIMU rev.	Feature ID[1]
Universal test-case(TC) .tcl file (no need to edit each one for tc folder and name)	5.1(planned)	f100
Randomization, seed force	4.6	f101
HDL Verilog, SystemVerilog and VHDL Compilation	4.6	f102
Elaboration phase	4.6(xsim)	f103
Optimization phase	4.6	f104
Simulation phase	4.6	f105
Coverage collection and report	4.6(vsim)	f106
Open weveforms after closing simulation (using cheap viewer license)	4.6	f107
Open weveforms after simulation is finished but not closed (using expensive simulator license)	4.6	f108
Execution in sim vendor TCL shell	4.6	f109
Execution in Linux tclsh shell	5.0	f109
Integration of FPGA vendor tool exported FPGA shell script	4.6(translation might be required), 5.0	f110
Coloured print of sim result	4.6	f111
Debug prints, enable/disable	5.0	f112
Disable DUT RTL compilation for speed and debugging	5.0	f113

Regression Simulation Features Supported

Regression Feature	SIMU rev.	Feature ID [1]
Randomization, seed force	4.6	f200
Regression Pass/Fail statistics, per module, per TC, total	4.6	f201
Automatic regression TC list generation, white module list, black TC list	4.6	f202
Regression TC list (manually or automatically generated) editing by hand	4.6	f203
Regression history accumulation	4.6	f204
Regression running on many threads in parallel, using one sim license per thread	4.6	f205
Send email when regression is completed	4.6	f206
Manual regression (not identical to manually generated TC list), single thread only	5.0	f207
Coverage aggregation and report	4.6(vsim)	f208
Submit jobs on remote machines	(planned)	f209
Regression run call with a number indicating the number of threads to execute	(planned)	f210
Regression progress for auto generated/manual/local/remote/threads regression	4.6(auto list), (planned)	f211
run folder with date/time hash	5.1(planned)	f212

HDL Simulators Supported

HDL Simulation Tool	Vendor	Tool Version Tested(note)	SIMU Template	SIMU rev.	Feature ID [1]
Questa®	Mentor®		complete example [2]	4.6	f300
ModelSim®	Mentor®		complete example [2] (identical to Questa)	4.6	f301
Xsim®	Xilinx®		complete example [2]	4.6	f302
ActiveHDL® GUI and CLI	Aldec®		complete example [2]	4.6	f303
Rviera®	Aldec®		no examples	capable [3]	f304
Xcelium®	Cadence®		partial configuration(missing coverage), complete TC, partial regression(missing coverage aggreagation)	5.0	f305
VCS®	Synopsys®		no examples	capable [3]	f306

FPGA Vendors Supported

FPGA Compile Tool	Vendor	Tool Version Tested(note)	SIMU Template	SIMU rev.	Feature ID [1]
Vivado®	Xilinx®		IPI example	4.6	f400
ISE®	Xilinx®		no examples	4.6	f401
Quartus®	Altera®		Platform Designer example	4.6	f402
Diamond®	Lattice®		no examples	(capable)	f403
Radiant®	Lattice®		no examples	(capable)	f404
	Microchip®		no examples	(capable)	f405
Vitis HLS®	Xilinx®			5.1(planned)	f406

OS and TCL Interpreters

OS	Vendor	Version	SIMU Mode	SIMU rev.	Feature ID [1]
Windows®	Microsoft®	10	simulator vendor TCL shell	4.6	f500
RedHat®	RedHat®	7,8	simulator vendor TCL shell, Linux tclsh shell	4.6	f501
Linux	Linux		simulator vendor TCL shell, Linux tclsh shell	4.6	f502

HDL Languge

HDL Language	Template	SIMU rev.	Feature ID
Verilog	complete example[^3]	4.6	f600
SystemVerilog	complete example[^3]	4.6	f601
VHDL	complete example[^3]	4.6	f602
Xilinx® HLS	Complete example[^3]	5.1(planned)	f603
Mentor® HLS		(planned)	f604

Frameworks and Libraries

Frameworks and Libraries	Template	SIMU rev.(note)	Feature ID [1]
UVM		(planned)	f700
OSVVM		(planned)	f701

Notes

Quick Reference

User Guide

In the examples below we will use Mentor's Questa® simulator calls. Other simulators will have very similar, almost identical calls and behaviour.

SIMU Configuration

The cloned repo doesn't need any configuration to run the existing test-cases, except the avalability of the simulation tools and licenses.

If new test-cases are created then some configurations have to chanchanged, see details further below.

Run Single Test-Case simulation

There are several ways to run simulation of a single test-case. They acheive the same result, running a test-case simulation, but have some subtle differences.

Test-case simulation - in simulator vendor TCL interpreter

```
$ cd symphony/dev/sim/run
$ vsim -c -do
../testcases_envFidus_sv_simMquestaXvivadoCxcelium/tc_fidus_common/tc_fidus_clock_reset.tcl -
do exit
```

The simulator vendor TCL shell can be accessed in GUI or in CLI. The example above useds the call to Quetta/Modelsim in CLI mode invoked from Linux bash sell.

Test-case simulation - in Linux TCL interpreter and SIMU shell with SIMU test-case simulation call

```
$ cd symphony/dev/sim/run
$ tclsh runme_simu_shell.tcl
$ run_testcase
../testcases_envFidus_sv_simMquestaXvivadoCxcelium/tc_fidus_common/tc_fidus_clock_reset.tcl
```

Test-case simulation - in Linux TCL interpreter and SIMU shell

```
$ cd symphony/dev/sim/run
$ tclsh runme_simu_shell.tcl
> source ../testcases/tc_reset/tc_reset.tcl
```

Run Regression Simulation

There are several ways to run a regression simulation of a all test-cases. They acheive the practically the same result, running regression simulation, but have some subtle differences.

Regression using atomatic test-case list - in simulator vendor TCL interpreter

```
$ cd symphony/dev/sim/run
$ vsim -c -do ../scripts_configure/run_regression.tcl -do exit
```

The simulator vendor TCL shell can be accessed in GUI or in CLI. The example above useds the call to Quetta/Modelsim in CLI mode invoked from Linux bash sell.

Regression using atomatic test-case list - in Linux TCL interpreter and SIMU shell with SIMU regression call

```
$ cd symphony/dev/sim/run
$ tclsh runme_simu_shell.tcl
> source ../scripts_configure/run_regression.tcl
```

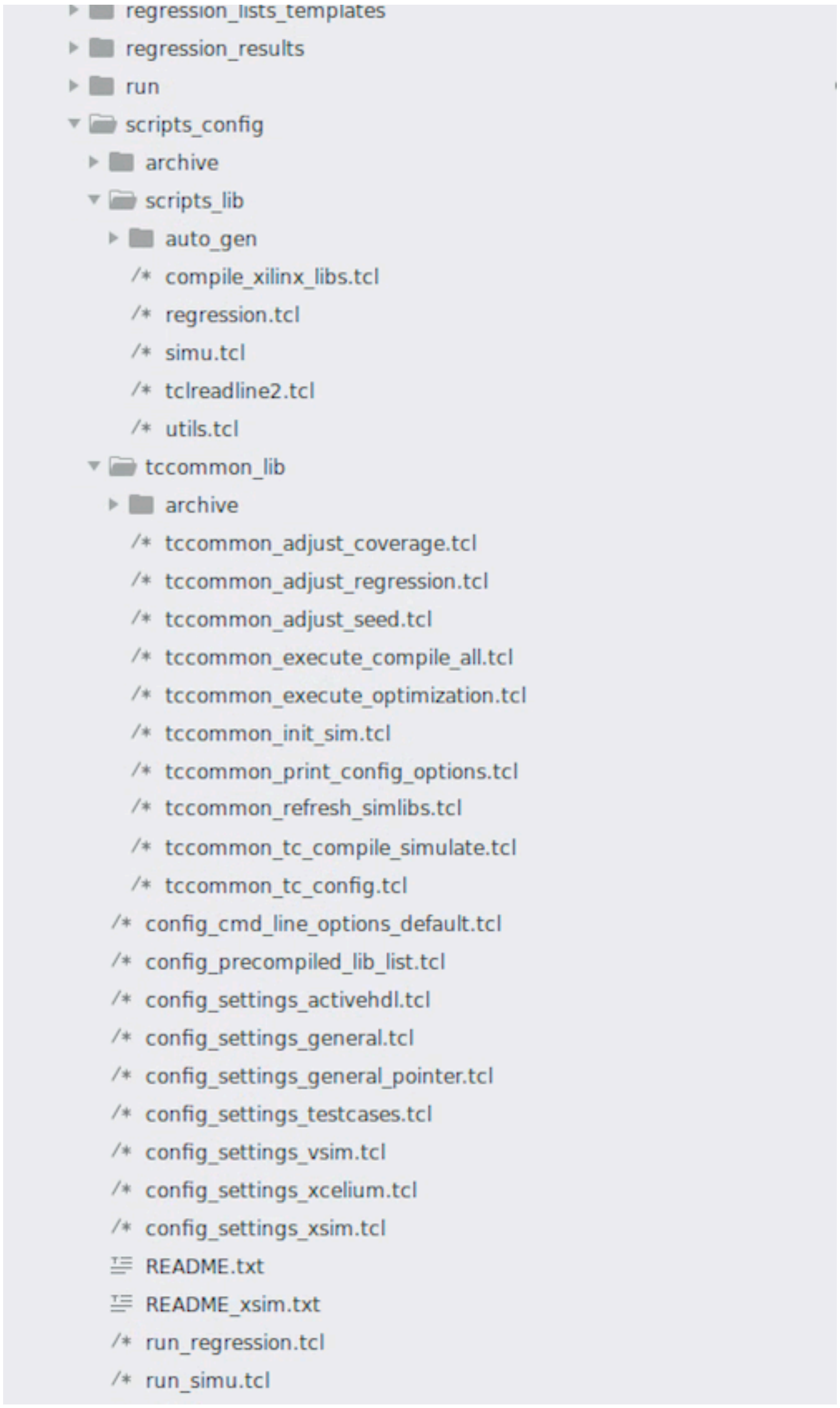
Regression using manual test-case list - in Linux TCL interpreter and SIMU shell

```
$ cd symphony/dev/sim/run
$ tclsh runme_simu_shell.tcl
> source
../home/work/des.v/trunk/simu_fixes/simu/dev/sim/regression_lists_templates/REGRESSION_TC_LIS
T_MANUAL.tcl
```

Architecture

Scripts folder structure

The TCL scripts are located in the scrpts_config folder as described below:



```
▶ regression_lists_templates
▶ regression_results
▶ run
▼ scripts_config
  ▶ archive
  ▼ scripts_lib
    ▶ auto_gen
      /* compile_xilinx_libs.tcl
      /* regression.tcl
      /* simu.tcl
      /* tclreadline2.tcl
      /* utils.tcl
  ▼ tccommon_lib
    ▶ archive
      /* tccommon_adjust_coverage.tcl
      /* tccommon_adjust_regression.tcl
      /* tccommon_adjust_seed.tcl
      /* tccommon_execute_compile_all.tcl
      /* tccommon_execute_optimization.tcl
      /* tccommon_init_sim.tcl
      /* tccommon_print_config_options.tcl
      /* tccommon_refresh_simlibs.tcl
      /* tccommon_tc_compile_simulate.tcl
      /* tccommon_tc_config.tcl
    /* config_cmd_line_options_default.tcl
    /* config_precompiled_lib_list.tcl
    /* config_settings_activehdl.tcl
    /* config_settings_general.tcl
    /* config_settings_general_pointer.tcl
    /* config_settings_testcases.tcl
    /* config_settings_vsim.tcl
    /* config_settings_xcelium.tcl
    /* config_settings_xsim.tcl
    README.txt
    README_xsim.txt
    /* run_regression.tcl
    /* run_simu.tcl
```

scripts_config/tccommon_lib

None of the scripts in this folder should be called directly.

This folder contains all scripts executed during the run of a test-case (TC). All these scripts are called only from the test-case script.

scripts_config/scripts_lib

None of the scripts in this folder should be called directly, except the library compilation script in special cases.

There are three main scripts in this folder, which are at the heart of the simu scripts - the simu.tcl library script, the regression.tcl script and the utils.tcl library. The simu.tcl script is calling the other two scripts, and the TC[4] execution TCL script is calling the simu.tcl script.

The tclreadline2.tcl script provides the user friendly experience keeping the history of the executed commands. It is very helpful when using the Linux tclsh TCL interpreter that lacks basic user friendliness. It is automatically called by the simu.tcl script.

The compile_xilinx_libs.tcl script is used to compile Xilinx libraries, but the same functionality can be done manually from the Vivado tool GUI. It can be used for scripting automation purposes if we don't want to use the tool GUI.

scripts_config/scripts_lib/auto_gen

In this folder is located cmd_line_options.tcl which is automatically generated and should not be changed manually because it will be overwritten. The script is generated from the config_cmd_line_option_defailt.tcl and in some cases takes into account with higher precedence the CLI passed arguments when the simu library is used to run test-case or regression simulations.

Developer's Guide

Appendix

Abbreviations

External References

1. 'Feature ID' - unique ID used to keep track of the features developemnt, for internal developes use ony ↩ ↩ ↩ ↩ ↩ ↩

2. 'complete example' - one or several examples that include - simulator configuration, Test-Bench(TB), Test-Case(TC), Regression, Sim Vendor TCL shell, Linux tclsh shell, optimization, coverage ↩ ↩ ↩ ↩

3. 'capable' - it is potentially possible to use it, although a new simulator configuration and TB/TC example has to be created based on exiting templates ↩ ↩

4. TC - Test-Case ↩