Verification using PyUVM

This is a testbench using PyUVM and cocotb.

In the following It is assumed that the repository root is named `<REPO_ROOT>` and the root of the example test bench is named `<SSTBEX_ROOT>` which is `<REPO_ROOT>/verification/student_ss`.

The in the provided example the RTL code being tested is the `<REPO_ROOT>/src/rtl/student_ss_1.sv`.

Setup

To be able to run tests, the following software is required:

- Python > 3.6
- Icarus Verilog 11.0

Furthermore, a Python virtual enviroment is needed, for instructions on how see How to set up a virtual environment.

Within the virtual environment several Python libraries should be installed, which can be seen in `<SSTBEX_ROOT>/requirements.txt`. The requirements can be installed once the virtual environment is active using `pip install -r <SSTBEX_ROOT>/requirements.txt`.

How to use it

Running example An example test is located in the `<SSTBEX_ROOT>/example` folder and can be run using the Makefile.

To run the tests use: `make` This will run all files added to the `MODULE` variable and automatically detect all test-classes which has the `@pyuvm.test()` decorator.

To run a specific test `MODULE` can be specified with the make-argument directly e.g. `make MODULE=new_folder.cl_student_new_test` for a file `cl_student_new_test.py` located in `new_folder`-folder

Waveforms To generate waveforms run `make WAVES=1`, this generates a `toplevel.fst`-file located in the `<SSTBEX_ROOT>/sim_build` folder. The waveforms can be viewed using e.g. GTKWave

Coverage Coverage can be viewed in the txt-file `<testname>_cov.txt` located `<SSTBEX_ROOT>/sim_build` folder or visually using PyUCIS viewer by running `pyucis-viewer <testname>_cov.xml`

Adding testcases

New tests can be added in a new folder and should extend from the `cl_student_base_test`-class. See the example test `cl_student_example_test.py` for reference.

The `MODULE` variable in the Makefile should be changed to the file name of the new test. In case of multiple tests use a comma separated list.

Any code defining behavior of the test e.g. writing to and reading from signals should be added in the run-phase.

Changing the RTL

The RTL code being tested can be changed by modifying the Makefile and `toplevel.sv` file. In the Makefile the file should be added to the `VERILOG_SOURCES` variable.

In the `toplevel.sv` file the wanted module should be instantiated replacing the current module of `student_ss_1`. Keeping the signal names of the toplevel module allows the test to access the correct signals independently of the subsystem.