Verification Course
Exercises

Fall 2025

Contents

1 Introduction

The following document will provide to the reader an introduction to Universal Verification Methodology (UVM) verification through exercises to implement a testbench (TB) using several open-source tools.

The goal of these exercises is to:

- Understand the basic architecture of a UVM testbench using *PyUVM*;
- Understand the vertical reuse concept by integrating a Universal Verification Component (*uVC*) in a UVM testbench;
- Implement a library of tests and a library of virtual sequences, to fully verify the behavior of the Device Under Test (DUT);
- Implement UVM components, such as the Coverage and the Scoreboard, to collect testbench and design metrics.

For the exercises provided in document, it is assumed that the reader has received the source code of the UVM testbench, which is the intended template to support the exercises' development. In the document, the `<ROOT>` directory shall map to the base folder containing the source code received.

2 Exercises

2.1 Exercise 00: Computer Setup

The following exercise will consist in setting up the Linux environment and install all the required simulation tools to compile the design and the testbench code, running simulations and visualizing the waveforms of all the signals involved.

The guide `client-setup.pdf` available in `<ROOT>/client_setup`, describes the steps to achieve this.

Afterwards, create the python virutal environment.

Setup Python Environment

To setup the virtual environment and activate it:

```
[<username>@<servername> <ROOT>]$ source bin/venv.src.me
[<username>@<servername> <ROOT>]$ source .venv/bin/activate
# You are now in the Python Virtual environment
(.venv) [<username>@<servername> <ROOT>]$
```

To deactivate the virutal environment:

```
(.venv) [<username>@<servername> <ROOT>]$ deactivate
# It has now been deactivated
[<username>@<servername> <ROOT>]$
```

2.2 Exercise 01: Python Introduction

Objective: *Introduction to the Python programming language.*

Read the following pages in *Python for RTL Verification* by Ray Salemi to get familiar with Python.

- Introduction: p. 1-8
- Python Basics: p. 9-138

Or one can use the Open-Source book: "Python For Everybody", chapter 1-6, 8-9 and 14.

Additional reading and examples can be found under:

- `https://github.com/raysalemi/Python4RTLVerification`
- `https://www.youtube.com/playlist?list=PLDAnhhk0KczxDJr5ucQOZ-IcW5yeSa1e4`

2.2.1 Exercise 1 – Functions and Variables: Temperature Converter

Write a function that converts temperatures between Celsius and Fahrenheit.

- Input: value (number) and unit (‘"C"‘ or ‘"F"‘).
- Output: converted value.

Challenge: Extend the function to support Kelvin (‘"K"‘). Use a dictionary of conversion rules instead of ‘if/else‘.

2.2.2 Exercise 2 – Lists and Loops: Even Number Filter

Write a function that takes a list of integers and returns a new list with only the even numbers.

- Use a ‘for‘ loop.

Challenge: Rewrite using a **list comprehension**. Add an optional argument to filter either even or odd numbers.

2.2.3 Exercise 3 – Dictionaries: Word Counter

Write a function that counts how many times each word appears in a given string.

- Use ‘.split()‘ to separate words.
- Store results in a dictionary.

Challenge: Make it case-insensitive and ignore punctuation. Rewrite using **dictionary comprehension** or ‘collections.Counter‘.

2.2.4 Exercise 4 – Control Loops: Multiplication Table

Write a function that prints a multiplication table up to a given number (e.g., 5×5).

- Use nested ‘for‘ loops.

Challenge: Instead of printing, return the table as a **nested list** using list comprehensions.

2.2.5 Exercise 5 – OOP with Python

This exercise will work with OOP and Python by letting you create a base class for shapes and then later derive concrete shapes as circles and squares.

Create a base class: shape Name the base class shape

- Add construcor
- Add empty member function called area

Extend shape to circle

- Extend shape to a derived class called circle which has a radius member variable as an int
- Add a: function void setRadius(int radius); which set the radius
- Add a: function int area(); which computes the area of the circle (Use 3 as a value for Pi)

Extend shape to square

- Extend shape to a derived class called square which has a length member variable as an int
- Add a: function void setLength(int length); which set the length
- Add a: function int area(); which computes the area of the square

Instantiate circle and square

Write a small propgram which instantiates a circle (named c) and a square handle (named s) and set the radius and length and print the computed area

Polymorphism Create a program which creates a list of shapes and instantiates random number of circles and squares. Then make a loop priting each area and finally the total area of all the shapes.

2.2.6 Exercise 6 – Combining Concepts: Student Grades

Create a program that:

- Stores student names and lists of grades in a dictionary.
- Calculates each student's average grade using a loop.
- Finds the student with the highest average.

Challenge: Use a **dictionary comprehension** to compute averages, and ‘max()‘ with ‘key=‘ to find the best student in one line.

2.3 Exercise 02: Intro CocoTB Exercises

These exercises will introduce you to:

- The (Make) flow for running CocoTB simulations
- CocoTB constructs

2.3.1 Exercise A: Simple *cocotb* Test for Adder Design

Objective: *Introduction to the cocotb tests.*

Task: *Run cocotb tests for simple RTL and view waveforms.*

Look at the test example for the adder in

`<ROOT>/exercises/E02_intro_cocotb_exercises/A_example_adder`

The example can be run by going to the `test`-folder, make sure that the virutal enviroment is activated. Run the tests using `make` and the flag `WAVES=1` to generate a waveform-file. The waveforms can be seen by opening the file in e.g. gtkwave

```
# Run the tests
(.venv) [<username>@<servername> test]$ make WAVES=1

# visualize the waveforms
(.venv) [<username>@<servername> test]$ gtkwave sim\_build/adder.fst
```

2.3.2 Exercise B: Simple *cocotb* Test for Multiplexer Design

Objective: *Introduction to the cocotb tests*

Task: *Development of a cocotb test for simple RTL.*

Create two *cocotb* tests similar to the adder example for the multiplexer design (basic and random tests), using the test-setup found `<ROOT>/exercises/E02_intro_cocotb_exercises/B_mux`.

NOTE: Create a python module called test_mux.py in

`<ROOT>/exercises/E02_intro_cocotb_exercises/B_mux/test`

HINT: Look at the RTL for MUX. It has different ports than the adder!

Create the two *cocotb* tests in the test_mux.py file:

- `async def mux_basic_test(dut)`: Drive a single transaction through the DUT
- `async def mux_randomized_test(dut)`: Drive 10 random transactions thorugh the DUT

Afterwards, modify the Makefile (if needed) and run the tests.

Cocotb Triggers Try using the different methods for increasing the time in simulation that can be imported from `cocotb.triggers`.

Example:

```
# generating clock signal, driving the clk of the DUT
cocotb.start_soon(Clock(signal=dut.clk,period=4,units='ns').start())

# allowing time to pass
await Timer(2, 'ns')
await ClockCycles(dut.clk, 2)
await RisingEdge(dut.clk)
```

Add the differnt triggers to the end of the mux_basic_test.

2.3.3 Exercise C: Parallel *cocotb* Test

Objective: *Introduction to the cocotb tests.*

Task: *Development of a cocotb test for simple RTL.*

This exercise will introduce usage of `Combine` and `First`. Do the following exercise using the files in

`<ROOT>/exercises/E02_intro_cocotb_exercises/C_parallel`

Create cocotb test for parallel design

Create a coroutine for each signal (A, B, C) that drives them at different intervals.

See example below.

```
await RisingEdge(dut.clk)

for _ in range(20):
    A = random.randint(0, 7)
    dut.A.value = A
    await ClockCycles(dut.clk, 3)

dut.A.value = LogicArray('x'*4)
```

Create a test that starts all the coroutines using `cocotb.start_soon()`.

Use `Combine` to await all coroutines.

Similarly, create a test that starts all the coroutines and uses `First`.

- What are the differences between the two triggers?
- In what use cases could they be useful?