EEL 4712C: Digital Design
Spring 2024

LAB 06: Verification
(30 Points)

Pre-Lab Objectives

Constrained Random Verification, commonly known as CRV, is a method of testing a design using pseudo-randomized inputs. CRV uses pre-defined goals to achieve a certain amount of "coverage" that indicates when a design has been tested enough. Designers can decide certain constraints they would like to implement for testing, making this very useful if there are certain ranges of inputs that will expose bugs.

To read more about CRV and the library we will use in this lab, check out these links:

https://github.com/OSVVM/OSVVM/tree/main

https://vhdlwhiz.com/constrained-random-verification/

Lab Requirements

Part 1: Library Downloads and Software Setup

OSVVM is a free library that contains many resources for constrained random verification. Please download the zip file in Canvas, which contains the OSVVM library as well as some TCL scripts that you will need to run in ModelSim or QuestaSim.

For this lab, I recommend running ModelSim as a third party application. You can do so by downloading ModelSim https://www.intel.com/content/www/us/en/software-kit/750368/modelsim-intel-fpgas-standard-edition-software-version-18-1.html and using the .exe file to set up the program.

Make sure you remember where you put the unzipped zip file. Open ModelSim, and in the transcript, type:

```
do PATH_TO_EXTRACTED_FILES/run.do
```

This path will look different for everyone. It should map from the ModelSim application or whatever ModelSim project you have open to the files you just extracted.

After doing so, the transcript should show an output like this:

```
#
# 3 compiles, 0 failed with no errors.
# ********************************************************************
# *                                                                  *
# * Thank you for downloading this example from VHDLwhiz.com         *
# *                                                                  *
# * Type "runtb" in the ModelSim console to run the testbench        *
# *                                                                  *
# * Like this:                                                       *
# *                                                                  *
# * ModelSim> runtb                                                  *
# *                                                                  *
# ********************************************************************
```

This means that the appropriate libraries have been installed. You can now type “runtb” in the ModelSim transcript and look at the testbench waveform output.

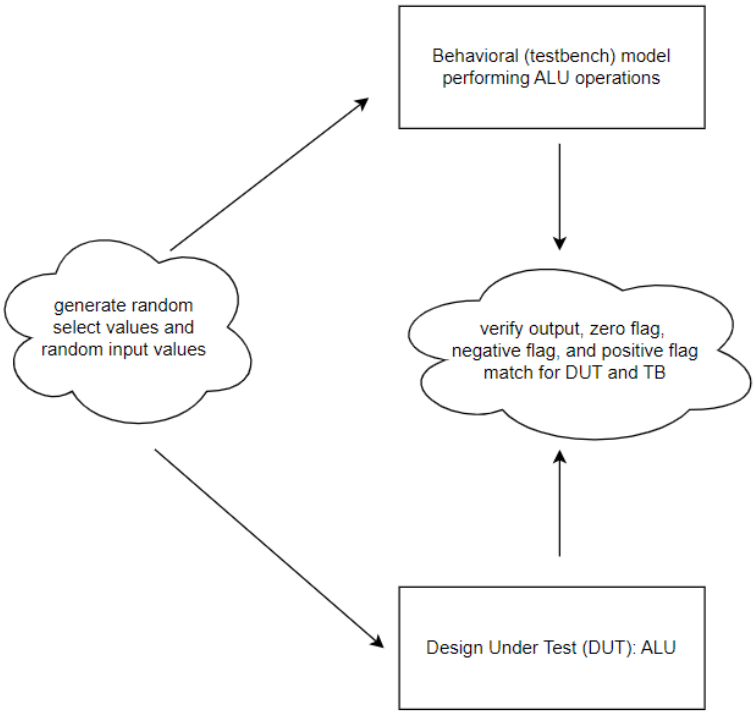
Part 2: Create a CRV Testbench

Now that your libraries have been downloaded, you will create your own CRV based testbench. You may use the provided ring_buffer_tb.vhd file to look at how these testbenches are created, but you will create one for an ALU, not a memory buffer.

Many functions used in the ring_buffer_tb.vhd file are ones you are expected to use in this lab. You will need to read about these functions—parsing through the documentation is a part of this lab. The PIs will not explain what a function does if you cannot show that you have tried to figure it out on your own.

The image below shows the general flow of a CRV testbench. While most testbenches are written using SystemVerilog, we will use VHDL for this one.

You can download the provided alu_tb.vhd file on Canvas, which contains a header with all of the libraries that need to be included.



The ALU you will use can be found in Canvas, under Files. It is an extremely simple 4-function ALU that performs an addition, subtraction, and operation, and an or operation. It has three inputs: two used to perform these operations, and one that selects which operation to perform. It also has four outputs: one for the result of the operation and three for status flags (positive, negative, or zero) of the result.

Your testbench should do the following:

- Instantiate the Design Under Test (DUT)
- Generate random input values
- Collect coverage data and monitor the coverage goals
- Use a behavioral model of the DUT
- Verify the DUT's outputs with the behavioral model's

At a minimum, you should have seven areas of coverage. You may have more, but seven is the minimum number required to earn full points in this lab. Your testbench should show that the ALU output and flags match the testbench's output and flags.

Demo Procedure

Run the testbench in front of a TA. This is required for full credit.

Deliverables & Submission Guidelines

- Summarize your work in a report. Template for lab report is provide on Canvas
- Zip your lab solutions and report in a file name <Your_UFID>.zip with a folder structure as shown below.
    - Make sure to use the file names shown below in your submission
    - Make sure to include the alu.vhd file, an annotated simulation, and the alu_tb.vhd file.

```
Your_UFID/
├──  lab_report.pdf
├── <lab files>.vhd
```

Extra Credit

The OSVVM library contains many packages not required by this lab. However, if you sift through the documentation of the library, you can figure out how to use these packages to verify outputs using other properties available. If you do something beyond what is required, you may earn extra credit on this lab. Using eight coverage bins instead of seven is NOT an example of this. Some acceptable options for extra credit include:

- Provide constraints on the random inputs you are using
- Print coverage reports in the transcript periodically
- Filtering messages
- Creating a CRV-based testbench in SystemVerilog

You may also do something else. You can earn a maximum of 10 points extra credit on this assignment. In addition, no extra credit will be awarded unless the main portion of this lab has been completed.