Digital Design Lab (EDA322)

Lab 4

Examiner: Prof. Ioannis Sourdis
TAs: Ahsen Ejaz, Ghaith Abou Dan, Fredrik Jansson, Konstantinos Sotiropoulos, Magnus Östgren, Neethu Bal Mallya, Panagiotis Strikos

Last Edited: 2024-02-08
Deadline: Your respective lab session during Study Week 7

1 Introduction

In this lab, you will tie your top-level design module with a testbench that you will design. The test will involve running a simulation where a program is executed on your VHDL description of the processor. The testbench will observe values on five output ports and check these against expected values provided to you in the form of five files that contain expected sequences of values on each of the five ports. Before starting the lab, please prepare as described below.

1.1 Preparation

1. Complete Lab 3.
2. Listen to the corresponding coding tutorials.
3. Study the lecture material up to the previous study week.
4. Read through this lab manual before starting with the tasks.

1.2 Learning outcome

After completing this lab, you should be able to:

- Know the methodology one needs to follow to write a testbench.
- Be able to write a testbench for a digital design. This includes reading from reference files and comparing the design's signals' value to expected values saved in files.

2 Tasks

2.1 Task 1: Designing a Testbench

This lab **requires** you to design a testbench to verify the outputs in your implementation of the ChAcc processor. You have to observe the following output ports in the testbench:

1. `pc2seg` (value of program counter)
2. `imDataOut2seg` (value of `imDataOut`)
3. `dmDataOut2seg` (value of `dmDataOut`)
4. `acc2seg`(value of the `acc` register)
5. `ds2seg` (value of the `ds` register)

We have provided five files, namely `pc2seg.trace`, `imDataOut2seg.trace`, `dmDataOut2seg.trace`, `acc2seg.trace` and `ds2seg.trace` containing the sequences of values expected on the corresponding signals during the test (after reset release). Use the given `i_memory_lab4.mif` and `d_memory_lab4.mif` files to initialize the memory in your processor. The files will load a test program in the instruction memory and a set of initial values for the data memory. (Note: The files mentioned here are available in Canvas: *Files* > *Labs* > *Lab 4* > *lab4_files*)

Your testbench must drive the inputs to the processor (`clk`, `resetn`, `master_load_enable`, `extIn`) and monitor the outputs to ensure that they are as expected according to the trace files we have provided. To implement the testbench, follow the steps given below:

(Step 1) **Define your testbench:** Add a new VHDL module called testbench_lab4 in a file called `testbench_lab4.vhd`.

(Step 2) **Instantiate the processor in the testbench:** Create an instance of `EDA322_processor` inside the testbench. Compile your design and make sure you do not have any errors.

(Step 3) **Connect the processor:** Declare signals to connect to all the processor ports. Hardwire the `extIn` port to value "00001111". Compile again your design and correct any syntax errors.

(Step 4) **Drive the processor's inputs:** Once you have properly instantiated your processor in the testbench, you must start driving the inputs. Create processes or write continuous assignment statements that drive the following signals: `clk`, `resetn` and `master_load_enable`

- Use appropriate delays to setup `clk` with a 10ns period
- Use appropriate delays to setup `master_load_enable` toggled with a 20ns period
- Start your simulation with `resetn` active ('0') and then deactivate it after a clock period

Compile your design and correct any syntax errors.

Tips: Take a look at *testbench_lab3.vhd* and *alu_testbench.vhd* from Lab 3 and Lab 1 for ideas on how to design your own testbench.

(Step 5) **First Check:** After properly instantiating the processor in the testbench and driving the `clk`, `resetn`, and `master_load_enable` signals, run a simulation and dump all processor ports at the wave window to make sure everything is connected properly. Make sure that:

- There are no undefined red signals.
- The `resetn` signal is set correctly and that the `clk` and `master_load_enable` work as expected.

(Step 6) **Start verification - read the test vectors:**

The test vectors are the means to verify the correctness of the design. They provide the expected values for each output of the processor, for the program that is loaded in the instruction memory.

Declare a type for an array of n-bit `STD_LOGIC_VECTOR`. Each array will store the test vectors for one signal.

Now, write a function that can initialize that array from a file.

Tips: To create an unconstrained array, you can do something like,

```
type vector_array is array (natural range <>) of std_logic_vector;
```

Unconstrained arrays are a feature of vhdl-2008, so you must switch the version of VHDL that Modelsim uses for this `.vhd` file. To do the same, right-click the `.vhd` file in the **Project** window, go to **Properties**, and change the **Language Syntax** to ***Use 1076-2008*** in the **VHDL** tab.

If you have to create a signal of such an unconstrained array, follow the syntax given below. Note that the fields inside the <> must be determined by you (and should omit the symbols '<' and '>').

```
signal <name> : vector_array(<length>)(<width>) := <initialization>;
```

Tips: Reading text files using functions: There are several ways to read a text file in VHDL. You can write a function very similar to how we initialized the memory array in Lab 2 (`init_memory_wfile`). If you decide to use the function from Lab 2 (which assumes that the files are as big as the array), you will be forced to declare multiple copies of the function (since different traces have different elements). To avoid writing multiple instances of the function, one way is to set the maximum needed size for the array and then use:

```
while not endfile(in_file) loop
...
end loop;
```

i.e., change the "`for`" loop in `init_mem_wfile` function with "`while not endfile (mif_file)`" to read every file to completion.

Once you have the function and it compiles without errors, declare five signals and initialize them with the function for every trace file `pc2seg.trace`, `imDataOut2seg.trace`, `dmDataOut2seg.trace`, `acc2seg.trace` and `ds2seg.trace`). Recompile the design and simulate it to ensure that the arrays are initialized correctly with the file contents. Remember to compile your design and correct any syntax errors before they pile up.

(Step 7) **Run the verification using the test vectors:** Write processes to check for transitions on each of the five signals of interest. You can code five processes. Each process is triggered when the signal of interest changes.

Remember that the trace files only contain values for transitions seen after `resetn` has been released. For correct comparison, **you only need to check for transitions seen after `resetn` is set to '1'**. Make sure that in your simulation `resetn` is assigned an initial value of 0 (active) and is deactivated only once. You have to design your processes to ignore any transitions on the five signals until `resetn` is released.

Inside each process, you must compare the expected value stored in the corresponding array and the value observed on the signal. Terminate the test using an assertion check if a mismatch occurs. Print an appropriate error message on the console.

Terminate the test after 1700 *ns* (you can use a separate assertion for that). If all previous checks until this point have passed, you can declare a successful test. The testbench should report success with an appropriate message on the console.

Tips: You can navigate to different signals of your design through the `sim` tab in QuestaSim, by selecting and expanding the desired component.

Tips: The instructions used for evaluating the functionality of the processor at the end of Lab 3 do not thoroughly check the `ds2seg` register. Hence, there is a chance that although the design has passed Lab 3, the testbench in Lab 4 will detect a mismatch. In this case, start debugging by double-checking the connections for the `ds2seg`.

(Step 8) **Debug, debug, and debug once more**.

If your testbench indicates an error, it's time to use QuestaSim to debug your design.

Start with the signal that has an incorrect value, and use the wave window and the processor's datapath to walk your way back to the component that started that behavior. If a signal does not behave as expected, examine its source.

Use the opcode, the FSM states, and the description of the executed program (`lab4code.txt`) to understand the expected behavior of the system.

You can navigate to different signals of your design through the `sim` tab in QuestaSim, by selecting and expanding the desired component.

Tips: Make sure you are using the correct *.mif* files given in *lab4_files*

3 Demonstration and Evaluation

The lab will be evaluated according to the checked aspects in the table below. To demonstrate your successful completion of Lab 4, keep all the essential files and simulation results ready to be presented to a TA.

Task#	Files	Coding Style	Simulation
1	`testbench_lab4.vhd`	✓	✓

– **The demo must be completed during your registered lab session**. Should you require an exception outside your registered lab session, discuss it with the TAs.
– CODE UPLOAD: After a successful demonstration, you will upload all of the code used to design the processor and the testbench. To complete this step, with the presence of a teaching assistant compress only the VHDL files that you have written as a ZIP file Your submission should be named with your Group ID. For example, if your GroupID is *101*, your submission should be named *EDA322_101.zip*.