EDA322Digital Design Lab

LAB5

Designed by: Anurag Negi, Angelos Arelakis, Ioannis Sourdis

In this lab you will tie your top level design module with a testbench that you will design. This is the only **task** of this lab assignment. The test will involve running a ModelSim 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 do the preparation as described below.

Preparation

Preparing for the fifth lab requires to:

1. Complete lab4.
2. Study the lecture material of up to the previous study week.
3. Answer the following questions in a document:
    a. Assuming that signal "expected_output" holds the correct value of signal "output" at cycle X. Show the VHDL code that will detect the case when "output" has different value than "expected_output".
    b. In the beginning of Task 1 (below), it is described which output ports are of interest to check in this testbench. It is also explained which files keep the expected values. Please write down the signals that will be needed for the expected outputs. How many are they and what is their width?
4. Read through the Lab PM before starting to do each task.

Designing a Testbench – Task1

Make sure that your top level design entity description looks exactly like the one shown below:

```
entity EDA322_processor is
    Port (    externalIn : in  STD_LOGIC_VECTOR (7 downto 0); -- "extIn" in Figure 1
              CLK : in STD_LOGIC;
              master_load_enable: in STD_LOGIC;
              ARESETN : in STD_LOGIC;
       pc2seg : out  STD_LOGIC_VECTOR (7 downto 0); -- PC
       instr2seg : out  STD_LOGIC_VECTOR (11 downto 0); -- Instruction register
       Addr2seg : out  STD_LOGIC_VECTOR (7 downto 0); -- Address register
       dMemOut2seg : out  STD_LOGIC_VECTOR (7 downto 0); -- Data memory output
       aluOut2seg : out  STD_LOGIC_VECTOR (7 downto 0); -- ALU output
       acc2seg : out  STD_LOGIC_VECTOR (7 downto 0); -- Accumulator
       flag2seg : out  STD_LOGIC_VECTOR (3 downto 0); -- Flags
       busOut2seg : out  STD_LOGIC_VECTOR (7 downto 0); -- Value on the bus
              disp2seg: out STD_LOGIC_VECTOR(7 downto 0); --Display register
              errSig2seg : out STD_LOGIC; -- Bus Error signal
              ovf : out STD_LOGIC; -- Overflow
              zero : out STD_LOGIC); -- Zero
end EDA322_processor;
```

There are five output ports that must be observed in the testbench: *disp2seg* (value of display register), *acc2seg*(value of the acc register), *pc2seg* (the program counter value),*dMemOut2seg* (value of the data memory output register) and *flag2seg* (flag values).

Five files have been provided to you -- namely *disptrace.txt, acctrace.txt, pctrace.txt, dMemOuttrace.txt, flagtrace.txt* -- containing the sequences of values that must be observed on the corresponding signals during the test after *reset* release.

A test program has been provided to you as well in the form of an instruction memory initialization file (*inst_mem.mif*) and a set of initial values for the data memory (*data_mem.mif*). These files should be used to initialize the instruction and data memories in your processor module.

What your testbench must do quite simply is to drive the inputs to the processor (clk, aresetn, master_load_enable, external_in) and monitor the outputs to make sure that they are as expected according to the trace files we have provided.

To structure your work in this lab, it is recommended that you follow the steps shown below:

Step 1:
Add a new VHDL module called *EDA322Testbench.vhdl* to the project of the previous lab that contains your code for the processor.

Step 2:
Create an instance of *EDA322_processor* inside the testbench. Compile your design and make sure you do not have any errors. Hardwire the "*externalIn*" port to value "00000000". And declare signals to connect to all the processor ports. **Compile your design and correct any syntax errors.**

Step 3:
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*, *master_load_enable* and *ARESETN*.
Please use appropriate delays to setup a clock with a 10ns period. "*master_load_enable*" can be toggled at the same rate as the clock. Start your simulation with *ARESETN* active ('0') and then deactivate it after 2 clock pulses. (HINT:Take a look at Testbenchlab4.vhdl). **Compile your design and correct any syntax errors.**

Step 4:
Now that you have properly instantiated the processor in the testbench and driven the *CLK*, *master_load_enable* and *ARESETN* signals, run a simulation and dump all processor ports at the wave window to make sure everything is connected properly (No undefined -red- signals). Make sure the ARESETN signal is set correctly and that the clock and master_load_enable work as expected. Also look at the *pc2seg* signal. If everything is correct it should change values 0,1,2,3,4,5,6,0,1,2,3,4,5,6,0,…

Step 5:
Declare a type for array of 8 bit std_logic_vector and write a function that can be used to initialize that array from a file. This is very similar to the way we initialized the memory arrays and you can use the same function. (Be careful with the size of each array as the function from lab2 assumes that the files are as big as the array). (HINT: change the for loop in the init_mem_wfile function with a "while not endfile(mif_file)" statement to read every file to completion). Once you have the function and in compiles without errors declare five signals and initialize them with the function for every trace file (*disptrace.txt, acctrace.txt, pctrace.txt, dMemOuttrace.txt, flagtrace.txt*). Compile the design again and simulate it to make sure the arrays are properly initialized from the file contents. **Remember to Compile your design and correct any syntax errors before they pile up.**

Step 6:
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 *ARESETN* has been released**. For correct comparison you only need to check for transitions seen after *ARESETN* is released (set to**

"1"). ***Make sure that in your simulation ARESETNis assigned an initial value of 0 (active) and is deactivated only once.*** You have to design your processes such that they ignore any transitions on the five signals until *ARESETN* 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 if a value of 144(decimal) is reached on the signal disp2seg. If all previous checks until this point have passed you can declare that that a test was successful. The testbench should report success with an appropriate message on the console.

Step7:
Does your processor pass the testbench checks? If not then simulate and examine the waveforms to see what goes wrong and fix it. Check the file code_to_test.txt in ping pong which explains what each instruction in instr_mem.mif does. Run the simulation slowly and examine the waveform step by step to make sure each instruction is executed correctly.

Demonstration

Tasks to be done for successfully completing this lab:

1. Design and implement the testbench. Show the code to the instructor. Demonstrate a successful run to your instructor.

Evaluation:

The instructor will check for the following:

Task#	Coding style	Simulation
1	X	X

Make sure that when you are done with the lab, you have demonstrated all checked aspects of each task. This is necessary for successful completion of the lab.

Lab report:

Describe what you did in lab5. More specifically, describe how you made the testbench to verify that your processor design was functionally correct. For example, you can focus on the following points:

- Mention briefly the general purpose of a testbench and the core principle used to achieve it.

- Show how you generated inputs to the processor during the testing, how the expected outputs were determined and how you compare the expected outputs with the actual outputs.
- Mention if your processor design was working correctly from the beginning and if not, describe how you backtracked the bugs.

Learning outcome:

After completing this lab, you should:

- 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 value of design's signals to expected values that are saved in files.

Hints and Tips

Take a look at alu_testbench.vhd and Testbenchlab4.vhd from labs 2 and 4 for ideas on how to design your own testbench.

Make sure you are using the correct files

As mentioned in the main part of this document, for lab 5 you are supplied with different .mif files for initializing the two memories of the processor. Please make sure you are using the correct files, since the testbench will obviously fail if you run the test program of lab 4. Pay attention to this point especially if you need to switch focus between labs 4 and 5.

Reading text files using functions

As mentioned before, there are several ways to read a text file in VHDL. If you decided to use the function given in lab 3 (*init_memory_wfile* function), since different traces have different number of elements, you will be forced to declare multiple copies of the function. In order 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;
```

Disp2reg

The instructions used for evaluating the functionality of the processor at the end of lab4 do not fully check the *disp2reg* register. Hence, there is a chance that although the design has passed lab 4, the testbench in lab 5 will detect a mismatch. In this case, start your debugging by double checking the connections for the *disp2reg*.