⚠ You have previously submitted this assignment with David Rama Jimeno. Group can only change between different assignments.

This course has already ended.

« 7. Exercise 5 (/comp.ce.240/spring-2024/... 8. Coding Style Questionnaire » (/comp.ce....

COMP.CE.240 (/comp.ce.240/spring-2024/) / 7. Exercise 5 (/comp.ce.240/spring-2024/E05/) / 7.1 Exercise 05: VHDL Test bench design

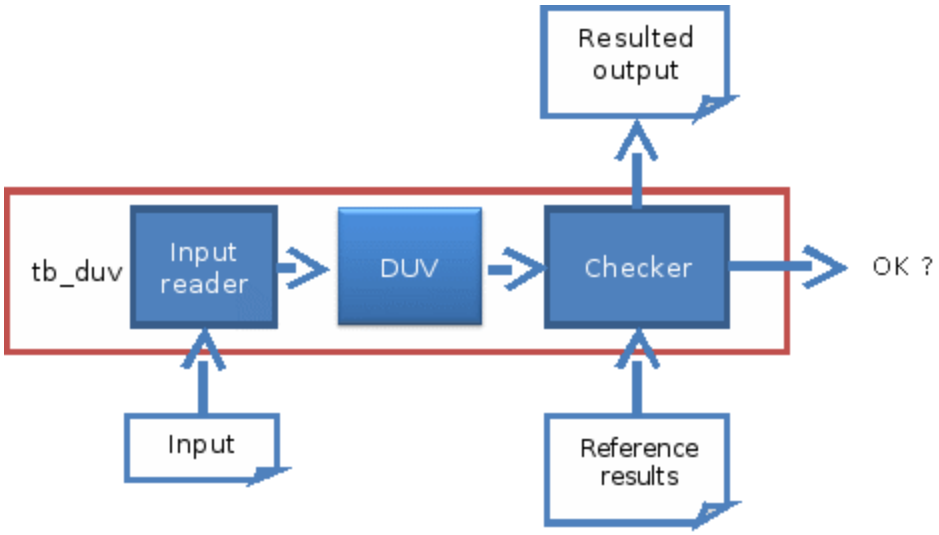
Assignment description (/comp.ce.240/spring-2024/E05/tb_multi_port_adder/)

My submissions (34/1000)

Exercise 05: VHDL Test bench design

The purpose of the exercise is to learn:

- Implementation of a test bench with the VHDL-language
- How to manage files using VHDL '93 standard
  - Attention! File management differs from the older standard (VHDL '87), which can be seen from the example on the lecture slides
  - More information on the differences of the standards can be found for instance from HARDI VHDL handbook (http://www.csee.umbc.edu/help/VHDL/VHDL-Handbook.pdf) on page 71
  - Options `vcom -87` and `vcom -93` force the compilation to follow a specified standard
- Implementation of more complex processes



Your task is to implement a test bench for the multiport adder (`multi_port_adder`) implemented in the previous exercise (https://plus.tuni.fi/comp.ce.240/spring-2024/E04/multi_port_adder/)

- The tested block is called Design Under Verification (DUV), aka DUT (Design Under Test)
- The test bench reads the test input from a file which contains four integers per line, separated with spaces
- The input files might contain comments starting with hash and the test bench should ignore them
- The multiport adder is tested now using 3-bit inputs
- Every input vector has a *reference result* which is compared to the output of the DUV
- Furthermore, the DUV's outputs are saved to a separate file (*Resulted output* in the figure)
- Take into account that the DUV might contain delay. In this case, the delay of `multi_port_adder` is 2 clock cycles since there are two consecutive synchronous adder blocks in the design.

Creating the test bench

Create a VHDL-entity with the name `tb_multi_port_adder`

- Use one generic-value `operand_width_g` with default value 3
- The entity does not contain ports as it is the top level of the test bench where the needed signals are generated and checked
- Needed packages: `ieee.std_logic_1164`, `ieee.numeric_std` and `std.textio`
  - `std.textio` does not need the attribute `library`

Define an architecture for the entity

- Name: `testbench`

- Define constants
  - Constant for the clock cycle length: 10 ns
  - Constant which defines the number of operands: 4
  - Constant for DUV's delay: 2

- Create signals
  - Clock signal and active-low reset with initial values '0'
    - Attention! Initial values cannot be used in synthesized code but in test benches they work.
  - Signal `operands_r` is connected as input to the tested block and signal `sum` is the output of the DUV
  - Shift register `output_valid_r` for delay compensation. Use width `duv_delay_c+1` as the *input reader*'s output is registered.
    - The type is `std_logic_vector(duv_delay_c+1-1 downto 0);`

- Define three files whose types are `text`
  - The first two are opened in read mode (`READ_MODE`) and last one is opened in write mode (`WRITE_MODE`)
    - `input_f` which reads in `input.txt`-file
    - `ref_results_f` which reads in `ref_results.txt`-file
    - `output_f` which writes to `output.txt`-file
  - Syntax: `file <file_handle_name> : <file_type> open <mode> is <filename>`

- Assignment of `not clk` to `clk`-signal after `period_c/2` to create a clock signal
  - Clock signal is made to oscillate with the desired frequency (*f = 1 / period_c*)
  - Attention! The clock signal needs an initial value for the `not`-operation to work.

- Set the reset-signal to value '1' after four clock cycles from the beginning of the simulation using the `after`-command

- Instantiate the multiport adder
  - Map the corresponding signals to correct ports

- Create a synchronous process for reading input files (`input_reader`)
  - The process does two things:
    1. Read four integer values from one line in the input text file every rising clock edge
    2. Assign the values to the multiport adders' inputs via signal `operands_r`
  - Step by step:
    - Create variables for the process for one line (type `line`) and four values (likely array of integers)
    - At reset: reset `operands_r` and `output_valid_r`to zero (hint: `others => '0'`)
    - On rising clock edge
      - Set the least significant bit of `output_valid_r` to '1' and shift the register to the left direction in order to delay the check process of the output conveniently

        - When the MSB is 1, the number of clock cycles set by `duv_delay_c` have been spent and the checker process of the test bench can begin. The checker process reads `output_valid_r` signal to check this.
    - Use the function `readline` to read the next line of the input file (in this case `input_f`)
        - `readline(file_handle_f, line_v)`
    - Use the function `read` to read one value from a line
        - `read(line_v, integer_variable_v)`
        - You can repeat the `read`-operation multiple times for one line
        - Use loops :)
    - When the end of the file (EOF) is reached and it is still tried to be read, the execution of the simulation fails. This causes problems as the last output arrives two clock cycles after reading the last input. In order to resume simulation despite of that, the handling of the file has to be put inside an if-clause which checks that EOF has not been reached.
        - Use function `endfile(file_handle_f)`

- Create a synchronous process for the checker (`checker`)
    - When the MSB of `output_valid_r` is one, you can begin to read reference values and to check the correctness of output
    - The process reads one line from the reference file `ref_results_f` and one value from the line to a variable
        - Use a similar structure as in the previous process and add two variables: one for a line and one for a reference value
        - Add similar EOF-checks. Use else-clause to inform of successful simulation "Simulation done" (assert false...) when the EOF is reached.
    - Add an assert that the read value is equivalent to the value of `sum`
        - Naturally we need a suitable type conversion
    - In addition to reading, we need to write the output of the tested block (DUV) to the output file `output_f`
        - Add a new variable for the line of ouput
        - Use function `write` to write one value to a line
            - `write(line_v, value)`
            - The value can be an integer conversion from a vector signal
        - Use function `writeline` to write a line to a file
            - `writeline(file_handle_f, line_v)`

- Remember to reset every registered signal at the corresponding processes

Misc

- Remember to compile *every* entity to the same work library (also `adder.vhd`)

- Verify the functionality of the multiport adder using input.txt (https://plus.tuni.fi/graderA/static/compce240-f2021/E05/input.txt) as an input file and ref_results.txt (https://plus.tuni.fi/graderA/static/compce240-f2021/E05/ref_results.txt) as a reference. There should not appear any errors in the simulation if the tested block works correctly.
- The default three bit width in the adder causes an overflow. Using four bits, this should not happen so you can use optionally file ref_results_4b.txt (https://plus.tuni.fi/graderA/static/compce240-f2021/E05/ref_results_4b.txt) as a reference. This way you can also verfy that all the generics work correctly.
    - In the final return, change the width back to 3 bits! Also check that your `.txt` file names match to same as told in the beginning.
- Hints for debugging
    - Because Modelsim does not give a good view for variables, it is useful to create corresponding signals and connect the variables to the signals
    - Modelsim might optimize your signals away. You can preserve them by using the `-voptargs=+acc` switch for `vsim`
    - Remember that clocked processes create registered signals which have one clock cycle delay
    - If the overflow causes troubles, check how the bits are discarded in the multiport adder's output
- Comment your code. (Comments start with "--")
- Make sure that your code is indented and aligned for better readability

---

Return:

- Put your returned files under `E05` folder in your Git repository
    - Return both `multi_port_adder.vhd` and `tb_multi_port_adder.vhd`
- Check that the files' header comments are valid, made according to instructions, and you have followed the coding rules
- Push the changes to your repository and submit (with your partner if you have a group!)
    - Use the **ssh variant** of the repository url in the submission. Otherwise the tests will fail 100% even with working design.
    - The url looks like *git@course-gitlab.tuni.fi:compce240-spring2024/<your_group_number>.git*.

Enter your Git repository address for grading

Did you remember git add - git commit - git push?

Submit with David Rama Jimeno Submit

Earned points

10 / 10

Exercise info

Assignment category
VHDL exercises

Your submissions
34 / 1000

Points required to pass
10

Deadline
Sunday, 18 February 2024, 23:59

Late submission deadline
Friday, 31 May 2024, 23:59

Group size
1-2

Total number of submitters
62

« 7. Exercise 5 (/comp.ce.240/spring-2024/... 8. Coding Style Questionnaire » (/comp.ce....