« 5. Exercise 3 (/comp.ce.240/spring-2024/... 6. Exercise 4 » (/comp.ce.240/spring-2024/...

COMP.CE.240 (/comp.ce.240/spring-2024/) / 5. Exercise 3 (/comp.ce.240/spring-2024/E03/) / 5.1 Exercise 03: Generic adder (RTL description)

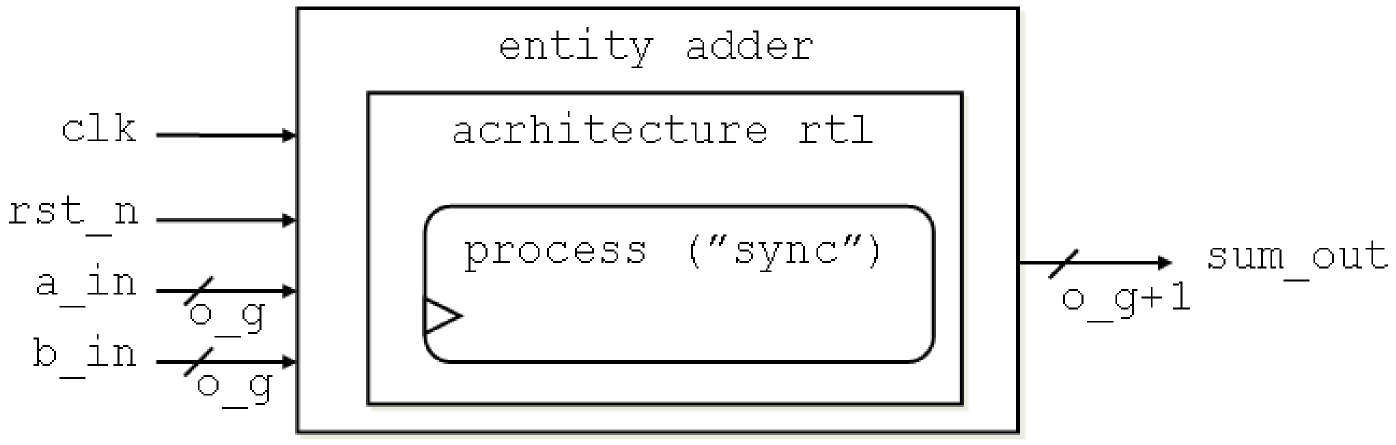
Assignment description (/comp.ce.240/spring-2024/E03/adder/) My submissions (1/1000)

Exercise 03: Generic adder (RTL description)

In this exercise we get familiarized with

- the power of expression and the brilliance of RTL-description
- defining *generic*-parameters
- synchronous processes
- types *signed* and *integer*
- type conversions

Your assignment is to describe a synchronous adder which has n-bit inputs and the output width is n+1 bits (sum = a + b)



(*)o_g = operand_width_g

- The inputs are added together and the output changes on clock edge (because this is a synchronous block)
- In an RTL-description, addition is represented with the operator "+"
    - The synthesis software chooses the most suitable adder type for the current technology (e.g. FPGA or ASIC)
    - Type `std_logic_vector` does not, however, allow arithmetic operations
    - Arithmetic is possible with types `signed`, `unsigned` and `integer`
- So you will need type conversions and width adjusting (https://plus.tuni.fi/comp.ce.240/spring-2024/material/conversion/)

Creating the adder

- Create file `adder.vhd`

- Use packages `std_logic_1164` and `numeric_std` from library `ieee`

- Create entity adder
    - Generic parameter (`generic`) of type `integer` without default values
        - `operand_width_g`
    - Five ports
        - `clk, std_logic`
        - `rst_n, std_logic`
        - `a_in, std_logic_vector`, whose width is `operand_width_g`
        - `b_in, std_logic_vector`, width `operand_width_g`
        - `sum_out, std_logic_vector`, width `operand_width_g+1`
        - remember to define indices with style `downto`
        - also remember that `x downto 0` implies a width of `x+1` bits.

- Define an architecture for the block (keyword `architecture`)

  - Give `rtl` as a name
  - Define a signal with type `signed` for the result so that it's width is `operand_width_g+1`
  - This signal becomes actually a register
  - After word `begin`, connect the registered signal to the output using a suitable type conversion
  - Define a sequential, i.e., synchronous, *process* for summation
    - Reset every register asynchronously (use active low reset-signal!)
    - Hint: long or adjustable length bitvector is easy to reset with structure `(others => '0')`
    - Sum `a_in` and `b_in` in a part sensitive to the rising edge of the clock
      - Make sure that data types and bit widths are compatible!

- Compile and verify your code using ModelSim with a ready-made test bench tb_adder.vhd (https://plus.tuni.fi/graderA/static/compce240-f2021/E03/tb_adder.vhd)
  - Open waveform window and choose radix *decimal*
  - Run simulation for 7 ms
  - At the end, the test bench prints out `# ** Failure: Simulation ended!`
  - Any other error messages should not appear

- Comment your code. Comments start with `"--"`.

- Insert to the beginning of the code file the standard form header (like for instance in the test bench)

Hints

- Miscellaneous instructions (https://plus.tuni.fi/comp.ce.240/spring-2024/material/misc/) contain amongst other things an example of directory structure
- Look for lecture slides (https://plus.tuni.fi/comp.ce.240/spring-2024/schedule/lec_all/) to get hints for VHDL syntax
- Type conversion and resizing functions (https://plus.tuni.fi/comp.ce.240/spring-2024/material/conversion/) in VHDL
- Naming conventions (https://plus.tuni.fi/comp.ce.240/spring-2024/material/naming_rules/) and coding rules (https://plus.tuni.fi/graderA/static/compce240-s2024/lgs_vhdl_coding_rules_sl_v5_1.pdf) are always compulsory!
- Make sure that your code is indented and aligned to make readability better (e.g., automagically in Emacs: VHDL -> Beautify -> Beautify buffer)

Return:

- Put your returned files under `E03` folder in your Git repository
  - Return your own `adder.vhd`

- Check that the file's 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

6 / 6

Exercise info

Assignment category
VHDL exercises

Your submissions
1 / 1000

Points required to pass
6

Deadline
Sunday, 4 February 2024, 23:59

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

Group size
1-2

Total number of submitters
64

« 5. Exercise 3 (/comp.ce.240/spring-2024/... 6. Exercise 4 » (/comp.ce.240/spring-2024/...