RTR532 2018
Problem set #02

Total value - 100 points

Submission deadline – 2018-03-07 23:59

Aim of this problem set – to practice:

1. Including an existing (yours or somebody else's) VHDL entity in your design
2. Training of VHDL conditional and sequential assignments to implement the ALU

Variant table

Student	Task 2 VHDL construction
Gotlaufs Roberts	When-else
Ivanovs Ruslans	With-select
Kuzminovs Glebs	If-else
Palamarcuks Dmitrijs	Case
Smirnovs Glebs	When-else
Smeiksts Linards	With-select
Zuters Karlis	If-else
"I was not present at first lecture"	Case

Starting notes

Copy the repo \rtr532-2018\problem_sets\ps02\ contents to your repository (\repo-student\problem_sets\ps02\)

Remember to commit changes (of .vhd files) to your local repository, before pushing it to the remote repository.

Problem sets will be graded from your repositories, as before.

Quartus project shall compile with no errors (Analysis and Synthesis step).

Task 1 – Combining ALU (alu.vhd module) with the full design (ps02.vhd) (20 / 100 points)

In the \rtr532-2018\problem_sets\ps02\ps02q there are three VHDL files present:

- Ps02.vhd – top level entity for the project
- Ps02_siggen.vhd – vhdl module that implements test signal generation for alu.vhd module
- Alu.vhd – vhdl module with entity declaration only – you will have to implement the architecture of the module in Task 2

Your first task is to include the ALU.vhd module (entity name **alu**) in the PS02.vhd (entity name – ps02). To do this, please look at ps02_siggen.vhd (entity name ps02_siggen) entity declaration (the part with input and output ports) and how it has been included in ps02.vhd. Refer to the example and do it similarly for the ALU.vhd module.

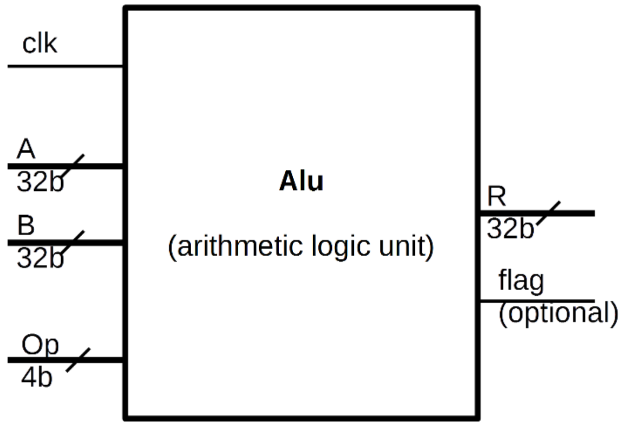


Figure 1 - drawing of Alu entity.

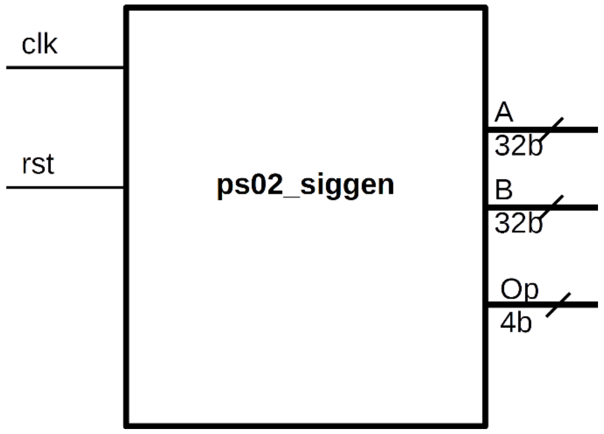


Figure 2 - drawing of ps02_siggen entity.

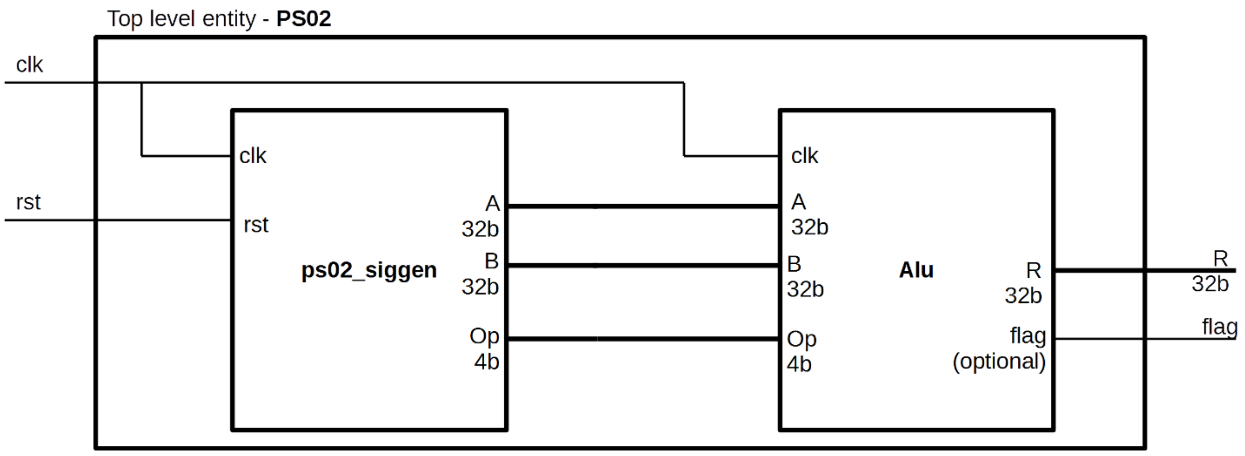


Figure 3 - drawing of ps02 entity with two other entities inside it.

References –

Lecture 2 (slides 30-33) \rtr532-2018\lectures\l02\L02_43 2018-03-01 eng.pdf

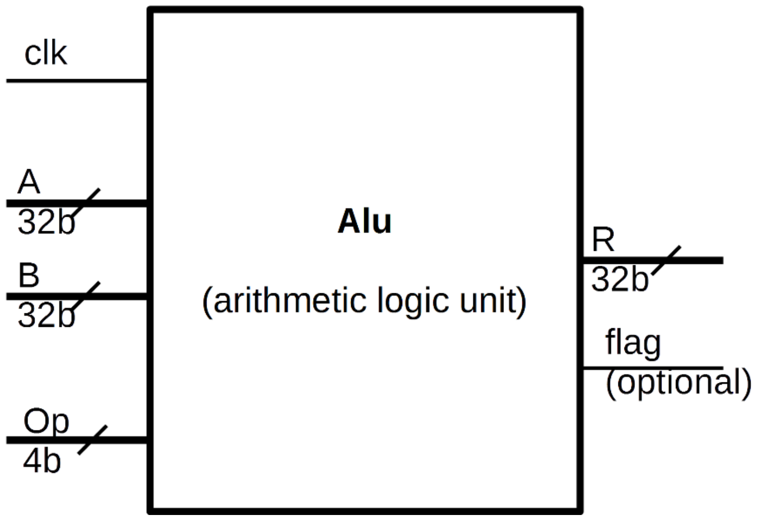
http://vhdlguru.blogspot.com/2010/03/entity-instantiation-easy-way-of-port.html

Task 2 – Architecture implementation of the arithmetic logic unit. (80 /100 points) (+ optional 20 pts)

Consider the instruction set called ***KrissRTR532*** shown in the table below:

Operation	Description	Operation code (bin)	Operation code (hex)
Add (signed)	R = A + B	0000	0
Subtract (signed)	R = A – B	0001	1
Bitwise AND	R(i) = A(i) AND B(i)	0010	2
Bitwise NAND	R(i) = A(i) NAND B(i)	0011	3
Bitwise OR	R(i) = A(i) OR B(i)	0100	4
Bitwise NOR	R(i) = A(i) NOR B(i)	0101	5
Bitwise XOR	R(i) = A(i) XOR B(i)	0110	6
Bitwise NOT A	R(i) = NOT A(i)	0111	7
Bitwise NOT B	R(i) = not B(i)	1000	8
Increment A by one	R = A + 1	1001	9
Increment B by one	R = B + 1	1010	A
Subtract A by one	R = A – 1	1011	B
Subtract B by one	R = B – 1	1100	C
Shift Logic Left (A)	R(i) = A(i + 1)	1101	D
Shift Logic Right A)	R(i) = A(i - 1)	1110	E
No operation	R(i) = 0	1111	F

Your task is to implement the defined instruction set in the arithmetic logic unit (ALU) module.



VHDL File and Entity declaration is already prepared for you (\rtr532-2018\problem_sets\ps02\ps02q\alu.vhd). Implement the ***KrissRTR532*** instruction set in the entity Alu architecture by using the corresponding VHDL construction from the variant table.

Output vector R and flag must be synchronous to clk (Those who must use **if-else** or **case** can combine it with rising_edge(clk) in the process. Others have to use an async-to-sync technique as shown in Lecture 02 (slides 25-27) \rtr532-2018\lectures\ l02\L02_43 2018-03-01 eng.pdf

Usually ALU also has an output flag, indicating overflow or underflow of output R (in addition or subtraction). If someone wishes to also implement the output flag, that will result in 20 more points. No explicit information on how to do it is given, that's why this is optional, but with +20 pts.

Tips, tricks and references

Explanation of ALU - https://www.youtube.com/watch?v=UsK5KV1FPmA

For a simple VHDL example of ALU, you can refer to http://vhdlguru.blogspot.com/2011/06/vhdl-code-for-simple-alu.html

Remember, that arithmetic operations are defined in ieee.numeric_std.all for signed() and unsigned() data types. Use https://www.doulos.com/knowhow/vhdl_designers_guide/numeric_std/ for reference on operations and type cast functions. Possibly you will have to convert types from/to std_logic_vector.

You can define anything you want inside the architecture (even things we have not covered in lectures) but you should NOT change anything in the entity declaration.