# Exercise 3: ALU: Arithmetic Logic Unit

This exercise tackles the Arithmetical Logical Unit (ALU). During this exercise, you will first setup the arithmetical units, then extend it with logic units, and finally see it in action on the FPGA. If you are quick enough, you get to implement an ALU for the full basic RISC-V integer instruction set.

**Task 1.** Setup a project for the ALU and add the following files.

- `op_add.vhd` (Design Source)
- `op_sub.vhd` (Design Source)
- `alu.vhd` (Design Source)
- `alu_tb.vhd` (Simulation Source)
- `isa_riscv.vhd` (Design Source)

The given file `isa_riscv.vhd` is a custom package which provides opcodes for RISC-V instructions. Implement the addition, subtraction and ALU entities. For the implementation of the adder, you can reuse your adder design from the last exercise. Verify the functionality of the design using the provided ALU testbench.

Now, the arithmetical part of the ALU is complete, as is the basic ALU design. Before extending the ALU to the also incorporate logical functions, you will delve into an analysis on resource sharing.

**Task 2.** Set the ALU to a small bit-width (e.g. 4 bit or 8 bit). Adjust the synthesis settings as follows:

- `-mode out_of_context`
- flatten_hierarchy: `none`

Now, execute the synthesis and compare the schematic following RTL elaboration with the schematic post synthesis. Change the flatten_hierarchy-setting and redo the above step. What happens to the logic of the addition and subtraction units? Explore the truth tables of the implemented LUTs and explain, how the logic is mapped to resources during synthesis.

**Task 3.** This third task is concerned with the ALU extension for logical operations. In a similar fashion, create VHDL descriptions for an `and`, `or`, and `xor` operation. All entities should consist of a generic for bit width and the following ports (these are the same as for `op_add` and `op_sub` given in task 1):

- 2 inputs: operands to the logical operation. Bit-width of these vectors should be generic.
- 1 output: result of the logical operation. Bit-width of this vector should be generic.

Now, extend the ALU by incorporating the newly added operations. Verify your design using the testbench given in the first task. Now, operations for add, sub, and, or, xor should be implemented correctly.

**Task 4.** Execute your ALU design on the FPGA. For this purpose, add the following files to the project.

- `alu_wrapper.vhd` (Design Source)
- `alu_constraints.xcd` (Constraint)

Make sure, that `out_of_context` mode is disabled. Execute Synthesis and Implementation and generate a bitstream. While you wait for these steps, look at `alu_wrapper.vhd` to see, how to interact with the ALU using buttons, switches and LEDs.

**Bonus Task 1.** When you take a close look at `isa_riscv.vhd`, you will see that there are more op codes defined than present in your implementation. These operations are

- `slt`: *Set less than*, result should be `1` if `a` $<$ `b`
- `sll`: *Shift left logically*, result should be `a` shifted logically to the left by `b` positions
- `srl`: *Shift right logically*, result should be `a` shifted logically to the right by `b` positions
- `sra`: *Shift right arithmetically*, result should be `a` shifted arithmetically to the right by `b` positions

These operations require special care with data type conversion. Implement these missing operations to finialize your ALU.