Oblig 5
VHDL Subprograms and packages - Functions and Procedures, Packages and Libraries

INF3160/4160

Version 2.2/16.02.2021

In this exercise, we will explore VHDL subprograms (`function` and `procedure`) and learn to create packages. Optionally, a2) will highlight some architecture features in FPGA design.

a) Simulate the attached code in `pargen.vhd` and `tb_pargen.vhd`, where two 16-bit vectors are read in and a parity signal is generated. Create a PNG-image of the simulation result (File->Export->Image).

b) Modify the code in `pargen.vhd` to encapsulate the two different methods in separate functions for creating parity calculation.

c) Move the functions in b) to a `subprog_pck` package. Modify `pargen.vhd` from part b of the exercise to use the functions from the `subprog_pck` package.

d) Create a procedure for tb_pargen that tests values from x"0000" to x"00FF" (one each clock cycle). Move the procedure to `subprog_pck` package. Modify `tb_pargen.vhd` to use the procedure from the `subprog_pck` package such that a new test vector for indata2 is created every clock cycle.

Hint #1: Type conversions can be a bit tricky in VHDL. Going from integer to std_logic_vector requires deciding on whether you will use signed data or not. Here is an example on how to convert from integer to an arbitrary length std_logic_vector using numeric_std and std_logic_1164 libraries:

```
my_var := std_logic_vector( to_unsigned(i, my_var'length) );
```

Hint #2: For procedures used in testbenches you may want to specify the use of signals. Default parameters are constant inputs. Anything deviating from this will need further specification:

```
procedure my_procedure(signal clk: in std_logic; signal my_data: out unsigned)
```

Approval:

- Simulation result (png)
- VHDL source file for the individual questions.
- Optional: Answers to optional questions (below).

a2) (optional addition after a))

Create a project using these two files (pargen and tb_pargen before modifications) in Vivado. (Remember to choose simulation only for the testbench file, and to select VHDL 2008 for both sources).
Open the *RTL-analysis->Elaborated Design*, and look at the schematic generated. Zoom in and compare the paths of the parity_toggle and the XOR_Parity (RTL_REDUCTION_XOR).

- How many gates is required for each path?
  (Not counting inverters, only (N)AND/ (N)OR/ X(N)OR).
- If we would implement this schematic in a full custom ASIC, which version would be favorable? (consider how many gate delays they will induce)

Synthesize the design, and open the synthesized design schematic. Note that the differences you will see compared to the RTL schematic are mostly due to the FPGA architecture consisting of logical blocks having mostly look-up tables (LUTs) and flip-flops.

Does it seem to make any difference whether our code indicates the use of multiplexers or reduction XOR if we implement it on a Xilinx Zynq-series FPGA?