Royal Institute of Technology (KTH)
School of Electrical Engineering and
Computer Science (EECS)
Department of Electronics (ELE)

Version 1.2
Last revision 2019-08-26

Laboration 1

IL1331 VHDL Design

IL2203 Digital Design and Validation using HDLs

Combinational Components

Student Name:……..

Personal Number:……

Date of Approval:……

Assistant:……

Laboration 1

Combinational Components

In the laboration, we first exercise structural modelling by building a ripple-carry adder and then build a simple test benches test it by comparing to the RTL-model using assertions. In the second part, we build the first component of the microprocessor, the ALU, using RTL-modelling. We also exercise how to build a more complicated testbench using functions and procedures. Finally, we download the ALU on the prototype FPGA board.

The FPGA boards will be handed out to the students during the first lab occasion.

Preparations

1) Download and the install the Quartus II Student Edition (version 13.1 supports the lab kits). It includes the Modelsim starter edition.
2) Write down the Boolean equations for the full adder. A full adder has three one-bit wide inputs *{ A,B, Cin}*, and two one-bit outputs *{ Sum, Cout }*.
3) Think through carefully how the Carry-In (cin) and the Carry-Out (cout) can be connected using recursive enumeration of signal index (cin(i)<=cout(??)).
4) Think through carefully which test patterns you need to toggle all wires in the ripple carry adder used in task 4-5, and write them down in a table. This table will serve as your test protocol.
5) Think through carefully how to write down the Boolean equations for the *(Z, N, and O)* flags and how to model it in VHDL for task 2.

Tasks

Task 1. Structural modelling of a ripple-carry adder.

1) Open Modelsim
2) Create a new project. *(File->New->Project…)*
   Remember to organize your files in a good way so they are easy to find, for instance as <course number>/<lab number>/<files>. The project directory is preferably the <course number>/<lab number> directory.
   The files should preferably be named according to the functionality they contain *<entity>[_<architecture style>].vhd*
3) Create a new VHDL file *(Project->Add to Project…).*
   Name it full_adder.vhd. Build the full adder component in VHDL. Use the IEEE std_logic type and write down the Boolean equations using the corresponding VHDL operators.
   Compile it by pressing the Compile button in Modelsim.
4) Create a new VHDL file.
   Name it ripple_carry_adder_structural.vhd. Write the VHDL code for the ripple carry adder (RCA). It should have a generic parameter *{N}* that should be used in the port statements to create a generic ripple carry adder with ports *(A,B, Sum)* of generic length *N*. The RCA architecture should be named *structure* and should use the *full_adder* component you just created in task 3 as a sub-component.
5) Create a test_bench *(test_ripple_carry_adder.vhd)* for the RCA architecture. To properly test, the following procedure should be followed:

- To check that it has been properly connected, the test vectors should toggle all internal and external wires in the structural model at least once.
- To check that the output is functionally correct, we compare the *Sum* output with the result of an RTL addition operation. Comparing two signals should be done using an Assertion. You perform the RTL-addition on std_logic_vector type signals (e.g., *Sum_RTL<=A+B;)* in the test-bench by including a proper IEEE package (num_std, std_logic_arith, std_logic_unsigned, or std_logic_signed). Most people recommend to use num_std.

Task 2. RTL Modelling of the ALU

1) Build an RTL model of the ALU. Name it *ALU.vhd*. The functionality/specification of the ALU can be found in the Appendix.
2) Build an automatic test bench for the ALU that goes through all possible logic combinations and all possible functions.
    - A proper test methodology says that the design and the test bench should be written by separate persons. Ideally, that would mean that one of the students should build the ALU model and the second student should build the test bench. However, at this point of the course, it is also important that students in the group understand both aspects, so it is ok to do them together.

Task 3. Prototyping the ALU on an FPGA board

1) Open Quartus II
2) Create a new project (File->New->Quartus II Project) and fill in the information requested by the New Project Wizard.
    - Create a directory called <course number>/FPGA_designs. Create a subdirectory under it and use it as a working directory for the project. It is good practice to give it the same name it as your top entity (alu.vhd)
    - Use the Device Family and the device that corresponds to the FPGA-board your lab group is using.
    - Include the VHDL files from your Modelsim directory (not the testbenches, only the structural files).
3) Start Analysis & Synthesis.
4) Assign the pins according to the board-manual.
5) Compile the design
6) Program the design on the board. Check that the downloaded FPGA design matches the specification and the results from the test bench simulation done in Task 2.2.

Passing Requirements

To pass the lab, the student should be able to show the assistants:

- Well-documented, well-commented and proper indented VHDL-models.
- Simulation results and waveforms in Modelsim.
- A functioning prototype on the FPGA board.
- Answer any questions that the assistants may have during the lab examination.
- When the lab assistant has approved the lab, he/she will sign the front page of it. Keep that document until the course has ended and your grade has been registered in Ladok. It is the only proof you have that you have made the lab.

Appendix: ALU

The Arithmetic-Logic Units have as its task to perform all arithmetic and logic operations in the computer. Its external pins are shown in the block diagram below.

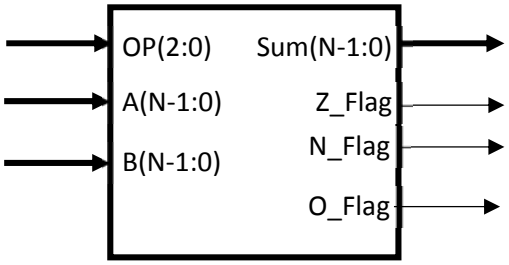

Figure 1. Block diagram ALU

The table below describes the ALUs functionality. The ALU is using 2's complement to represent negative numbers.

Operation				Outputs			
	Op_2	Op_1	Op_0	Y	Z_Flag	N_Flag	O_Flag
ADD	0	0	0	Y=A+B	Is active (=1) if the all bits in Y are 0.	Is active (=1) if the result on Y is negative.	Is active (=1) if the result of the operation gives the wrong sign bit.
SUB	0	0	1	Y=A-B			
AND	0	1	0	Y=A AND B			
OR	0	1	1	Y=A OR B			
XOR	1	0	0	Y=A XOR B			
NOT	1	0	1	Y=NOT A			
MOV	1	1	0	Y=A			
Zero	1	1	1	Y=0			