VHDL Tutorial

EEL 4720/5721 – Reconfigurable Computing

Introduction:

In this lab, you will be implementing and testing some basic digital circuits in VHDL to familiarize yourself with VHDL simulations and synthesis. Before starting, download all provided code off the website, which includes skeleton VHDL files and testbenches for simulations. If you are confused about the how each entity works, use google or look at the testbench code to see the correct output. Make sure to read my provided VHDL tutorial. Instructions for how to simulate and synthesis will be given in class, although I encourage you to figure it out from the Vivado documentation.

<u>You will submit this lab on e-learning, but it will not be graded</u>. Instead, in the case of borderline final grades, I will check to see how much effort was made on this lab. When submitting the lab, include all vhd files in addition to screenshots of working simulation waveforms.

Part 1 – Install Xilinx Vivado

See the link on the lab website.

Part 2 – Vivado Tutorials

Look over the tutorials linked off the lab website. Note that the tutorials are very long, so you do not need to know all the details. For now, as long as you are reasonably comfortable with the simulation features and basic synthesis features of Vivado, you will be fine. I will be demonstrating much of the functionality during class.

Part 3 – VHDL Tutorial

For all of the following parts, complete the specified entity and the simulate using the testbench with the same name and a _tb suffix. Ensure there are no failed assertions. Then, synthesize the entity in Vivado (for any FPGA) and ensure that there are no warnings.

3.1 – 2-to-4 Decoder

Modify the dec2to4.vhd file to implement a 2-to-4 decoder using 4 different architectures. Implement each architecture using the construct suggested by the architecture name (e.g., with select, when else, if, case).

3. 2 – 4-to-2 Priority Encoder

Create a 4-to-2 priority encoder (enc4to2.vhd). For this part, you only need two architectures based on the if and the case statement. Assume that higher inputs have priority over lower inputs. The valid output should be asserted when any of the inputs are 1 and should be 0 when all the inputs are 0. This valid bit is needed to understand the output of "00" for an input of "0001" and for an input of "0000".

3.3 – Adder + Register

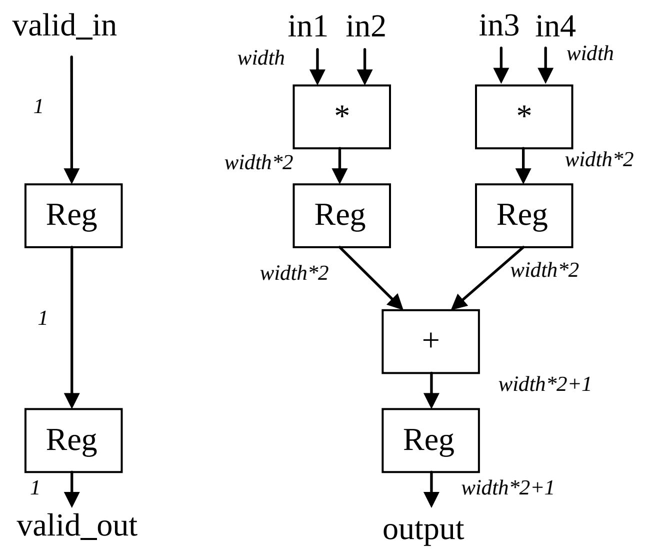
Create an adder followed by a register (add_pipe.vhd). As specified in the provided file, you should use a behavioral implementation. Note that the output of the adder is one bit wider than the inputs. In other words, the output should include the carry. Although it shouldn't matter, you can assume the inputs are unsigned.

3.4 – Multiplier + Register

Repeat part 3.3, but replace the adder with a multiplier (mult_pipe.vhd). The output of the multiplier should be twice the width of the inputs.

3.5 – Datapath

Implement the following datapath structurally. Use the entities from part 3.3 and part 4.3 where applicable, along with the provided register entity (reg.vhd) for the valid output logic on the left.


valid_in
1
Reg
1
Reg
1
valid_out
in1
in2
in3
in4
width
width
*
*
width*2
width*2
Reg
Reg
width*2
width*2
+
width*2+1
Reg
width*2+1
output