ELE00011H Digital Engineering
ELE00121M Digital Engineering for MSc

Laboratories: Session 1

Debouncers – Implementation and simulation

UG Students: **- ALL LABS SHOULD BE DONE IN GROUPS OF TWO STUDENTS**
- A mark penalty will apply to single-student submissions unless agreed in advance
- Any issues related to groups (conflicts) should be communicated as soon as possible

MSc Students: **- ALL LABS SHOULD BE INDIVIDUAL SUBMISSIONS**

Report formatting:

There is no formal report structure – you will be marked on the items listed within each lab script. Most reports will include code printout and simulation screenshots – see Lab 1 appendices for guidelines.

The reports for labs 1 - 4 must be handed in, **together in a single zip archive**, via the VLE by the deadline indicated on the front page of the script. A single submission should be handed in for each group (by any member of the group).

Each lab report, containing all the material required in the order specified, should be submitted as a **separate PDF file**. The exam numbers of all members of the group should be printed on the front page of each PDF.

The PDF files must be named **Yxxxxxxx-Yxxxxxxx_DE_Lab#.pdf**, where the Yxxxxxxx are the exam numbers of the group members (normally two for UG students, one for MSc students) and # is the lab number.

In all cases, read carefully the instructions on the VLE submission page. Failure to follow the instructions could lead to your assignment not being marked and in any case to a mark penalty.

Submission weight on module mark: 5% (10 marks)

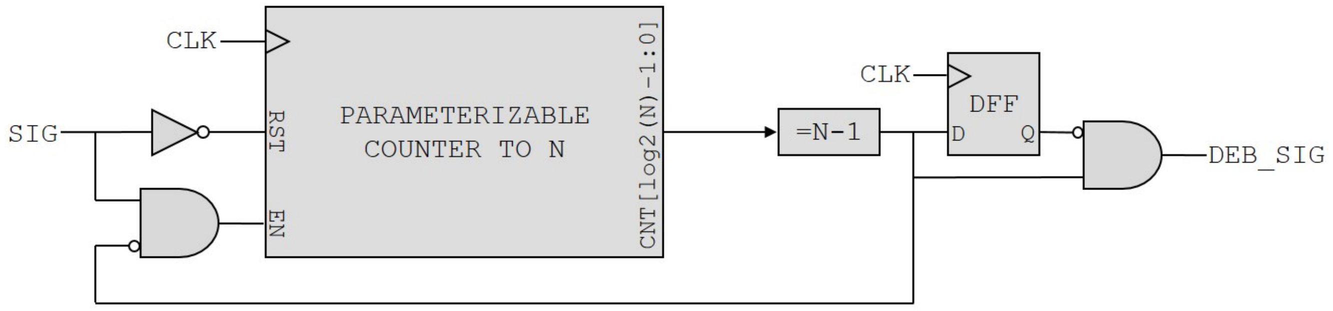
Task A: Debouncer implementation and simulation by parameterization [7 marks]

Note: this exercise will use the Fibonacci memory circuit you designed in Lab 3a of the VHDL last year (last term for MSc students). **You are welcome to re-use the same VHDL files.** If you made mistakes in your code, integrate any feedback you have received, as the code will be again marked for quality: if you have not kept a copy of your submission and feedback, let me know!

NOTE: if you do not remember the procedure to create new projects, launch simulations, etc., please refer to lab 1 of the VHDL module (the script is still online on the module website).

To retrieve the code, create a new project, then **import** your VHDL files from the previous project. To do so, click on "Add Sources", then select "Add or create design sources". In the next window, click on "Add Files" and navigate to the VHDL files you want to import (do not include your testbench at this stage). Double-click on the files to add them the list, then **tick the box next to "Copy sources into project"**. Click "Finish". The files should now be in your Design Sources list. Repeat the procedure for your testbench, this time selecting "Add or create design sources" in the pop-up window. The TB should appear in your Simulation Sources. Note that you will also need to include the DigEng.vhd library from last term in your project (import it using the same procedure as above).

As you probably have noticed, the Xilinx debouncer you used in Digital Design is quite poor. In this task, you will design a more efficient debouncer. The schematic layout for the new debouncer is the following:



Note the reset-less DFF at the output of the counter. Also, the box labelled "=N-1" is a simple combinational comparator that outputs '1' when the input is equal to N-1 (you have done several of these in the VHDL module, usually to generate the "done" signal).

Create a new entity (name it something sensible, but *not* "debouncer", as that name is already in use by the Xilinx version) and implement the circuit. Can you understand how it works? **Describe its operation as comments to the entity**.

First, you will need to create a parameterizable counter (it might be simpler to create a separate entity and use it as a component). Use the code provided in the slides from the parameterizable VHDL lecture last year/term. If the counter is a component, make sure to include a generic map and a generic input to your debouncer as well. In any case, debouncer entity should receive a generic value that sets the maximum value of the counter (N in the figure).

Next, replace the Xilinx debouncers with the new version in the top level of your circuit. You will need to add a generic map to the instantiation and a new generic input. **Set the default value of the generic to 50000000 (50 million)**: this will provide a default debounce of 0.5 seconds (any button press shorter than 0.5 seconds will be ignored). This is the value that will be used by the tools when you synthesize your circuit.

Of course, simulating this debouncer for 50 million cycles is impossible. However, parameterization will help.

Open you existing testbench and add the generic map to the UUT. Set the generic value in the map to 5. Now the debouncer will trigger for every pulse longer than 4 clock cycles (i.e. any pulse that includes at least 4 rising edges of the clock). Because this is longer than the Xilinx debouncer, you might have to make your pulses longer in the stimulus process of your TB. Also, in order to test the new debouncer, try some shorter and longer pulses to verify that the debouncer operates correctly (ignores short pulses and transform long pulses into a single pulse of 1 clock period). Observe the internal signals of the debouncers to verify that it operates as expected. Make sure to describe your testing strategy in the comments.

Synthesize the circuit and verify that the implementation uses the “real” debouncer (check the size of the counter that is generated by synthesis: does it match what you expect?)

Finally, implement the circuit on the board (the same XDC file you designed for the previous lab should still work). You should now be able to verify easily that short presses are being ignored!

Report:

The report for this lab should include, <u>in this order</u>:

- The commented VHDL code for the complete circuit, including all sub-components.
- The commented VHDL testbench for the simulation.
- The output of the simulation, including **all significant internal signals**. Note that the objective of this task is to verify the operation of the debouncers, so while it is important to show that the circuit as a whole is working correctly, you will also need to show that, in particular, the debouncers are operating as expected.
- The “RTL Component Statistics” and “RTL Hierarchical Component Statistics” part of the synthesis report. What is the size of the counters? Is it what you expected?
- The XDC file.

Task B: Simulating alternative versions [3 marks]

The procedure you followed in Task A works extremely well when you can use parameterization to simulate smaller components and implement larger versions of the same components. While not devoid of potential problems (something that works with one timescale does not necessarily always work with different timings – e.g. when FSMs are involved) it nevertheless allows you to simulate the *same* hardware that will be implemented, even if "scaled down".

However, in some cases this approach might not be possible, either because parameterization (and hence "scaling down") is impossible or because you want to have to flexibility of simulating alternative implementations. In these cases, a more "general-purpose" solution is available through the use of the *if-generate* VHDL construct. **Note however that this solution does not allow you to test the hardware that will actually be implemented, which will therefore have to be verified separately.** It should therefore be used with a lot of caution!

Create a new project and import all the VHDL files from the previous task (including the Xilinx debouncer). Open the top level file and **add a new generic "Simulation" of type Boolean. Set the default value to FALSE**. This implies that in synthesis the value will be false, whereas in simulation you can override the value with a generic map.

Next, scroll down to the COUNT debouncer and replace the current instantiation to something like the following (the precise names depend on your choice in task A):

```
-- Debounce count input – simulation version
SIM_CNT_DEBOUNCE: if SIMULATION generate
  Count_Debouncer: entity work.Debouncer
    PORT MAP(
       CLK => clk,
       Sig => count,
       Deb_Sig => deb_count
    );
end generate;

-- Debounce count input – implementation version
IMP_CNT_DEBOUNCE: if not SIMULATION generate
  Count_Debouncer: entity work.Debouncer_new
    GENERIC MAP (LIMIT => DEB_LIMIT)
    PORT MAP(
       CLK => clk,
       Sig => count,
       Deb_Sig => deb_count
    );
end generate;
```

Do the same for the RESET input.

Now open your testbench and add the new generic to the generic map of the UUT, setting its value to TRUE. The simulation will now use the Xilinx debouncer, while the implementation will use your new debouncer instead. You should not need to modify the input stimuli, except that you might want to re-visit your delays (as the Xilinx debouncer will accept all inputs longer than 2 clock cycles).

Implement the design in the boards and verify that it still operates as in task A.

Report:

The report for this lab should include, in this order:

- The commented VHDL code for only the components that were modified from task A (probably only the top level).
- The commented VHDL testbench for the simulation.
- The output of the simulation, including **all significant internal signals**. Note that the objective of this task is to verify the *if-generate* construct, so make sure to show that the debouncers are operating as expected.
- The "RTL Component Statistics" and "RTL Hierarchical Component Statistics" part of the synthesis report. What is the size of the counters? Is it what you expected?