ELE00011H Digital Engineering
ELE00121M Digital Engineering for MSc

Laboratories: Session 2

Design for performance – Part 1

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: 7.5% (15 marks)

Task 0: Setup

Create a new project with an appropriate name. Next, download from the module website the "algorithm.vhd" file and import it into the project using the procedure described in lab 1 (remember to add a *copy* of the source file into your design). Double-click on it to open it in the Xilinx tools.

The VHDL implements, very straightforwardly, a sequence of operations:

O <= (A*3 + B*C)/D + C +5

A few notes regarding the coding:

- The use of the IEEE.NUMERIC_STD.ALL library is crucial for arithmetic / mathematical operations, but it implies that signals need to be explicitly defined as types SIGNED or UNSIGNED (in this case, we will work with unsigned numbers). Nevertheless, Xilinx still needs to use STD_LOGIC_VECTOR as a default for all top-level entity inputs and outputs.
- Note the use of a generic value for data size. This parameterizable implementation will work for any data size. Note also that **there is no provision for overflow**. Some of your input vectors might well cause overflow and the result will be incorrect. You should ignore this.
- The size of internal signals is not random! Arithmetic operations defined in the NUMERIC_STD library have precise relationships between the size of the inputs and the size of the outputs. If in any doubt, check the library itself:
  http://www.csee.umbc.edu/portal/help/VHDL/numeric_std.vhdl
- Proper circuit design should always include **registered inputs and outputs** at the top level of your design (i.e. for the signals that will be connected to pins on the FPGA) – unless of course this is impossible because of other constraints. All registers (unless there is a good reason to do otherwise) should be implemented with a synchronous reset. Note also that D is the divisor in one of the operations, and therefore **cannot have value 0** and must be reset to a non-zero value (note the "odd" syntax of the test for reset in the I/O register – this is imposed by the simulation setup coupled with the non-zero requirement). **Keep this in mind when initialising signals in your testbench!**
- Note also the lack of comments in the file… not a good example for your code! Remember to always comment your code for all the hand-ins of this module. **Part of your task within the assignment is to comment the existing code**.

Task 1: Timing simulation [9 marks]

Create a new simulation source (testbench). This should be a self-checking testbench and use a record together with assert/report statements (see details below). Set the clock period to **120ns** and the initial wait period to **500ns**. Remember to re-synchronize to the falling edge of the clock!

Immediately after the re-synchronization, assign initial values to all the inputs (note that the default for input D will have be set to a non-zero value, so use x"0001" for D and x"0000" for the others), then reset the circuit. Next, inject at least **5 different sets of input values.** Choose your test vectors so that every vector inside the circuit is "large" (i.e. use some of the most significant bits in the vectors) at least once in the test sequence.

The inputs should change at every clock cycle. This is of fundamental importance for this lab (and the next) as we need to evaluate the maximum performance of the circuit. Failing to do so will invalidate your testbenches and result in a heavy mark penalty.

Note also that the I/O registers in the design impose a 2-cycle delay before the result for a given set of inputs arrives at the output – **this delay should be defined as a constant and the testbench should work for any value of this delay.** Keep in mind that the simplest approach in this case is to use **two separate processes**, one for assigning the inputs and one (delayed by the constant value) to check if the output is correct. For each set of inputs, an error message should be output to the console if the output is incorrect and a confirmation message should be output if they are correct.

1.1.1: Print out the self-checking testbench.

Run the behavioural simulation. By default, Xilinx sets a simulation duration of 1us, which might not be sufficient to observe the operation of the circuit. Set the duration to 5us and re-run it. Satisfy yourself that the circuit performs the correct operation (set the display radix of the output to unsigned decimal and note the output values). Familiarize yourself with the interface by observing the values of INT1-INT5.

1.1.2: Print out a screenshot of the behavioural simulation window, zoomed in to display, in readable format, all inputs, the output, and internal signals INT1-INT5 and INTO in unsigned decimal format. Include the console output (as a separate screenshot).

Now synthesize your circuit and open the Synthesis Report. Observe the "HDL Synthesis" section of the report. Can you readily identify the components you are expecting? You can also look at the RTL schematic for the circuit by selecting "Open Elaborated Design" in the "RTL Analysis" section of the flow navigator. You should readily recognise the circuit schematic.

Unfortunately, this design has too many inputs to be implemented on the board, but it is still possible to run implementation to let the tools create the circuit for you. The one thing that is absolutely necessary is to specify the clock and its behaviour (after all, we want to analyse the maximum frequency to evaluate the performance!)

Follow the instructions in lab 1 to create a XDC file. Instead of assigning pins, write the line:

```
create_clock -period 120.000 -name clk -waveform {0.000 60.000} [get_ports clk]
```

This tells the tools that the clk input is a clock with a 120ns period, with a rising edge at 0ns and a falling edge at 60ns (50% *duty cycle* - ratio between the time when the clock is 1 and when it is at 0). **In this and the next lab, you will modify the clock period several times: make sure to change the fall time in order to maintain (roughly) a 50% duty cycle at all times!**

Next, to prevent the tools from "cheating" (we will see this in the next lab), click on "Settings" under the Project Manager entry of the Flow Navigator, select "Synthesis", then scroll down the options until you see an entry called "-max_dsp" (which should have value -1). **Set the value to 0**, then click OK (answer "no" if prompted to preserve the previous run).

Click "Run Implementation" in the Flow Navigator (it will probably ask you to re-run synthesis). In the console, select the "Design Runs" tab. The second line is the *timing report* and the first value will be "WNS(ns)" (approximately 30ns, but there might be fairly significant variations – a few ns either way - due to the random nature of the routing algorithms). This is the *worst time slack*: essentially, it is the margin you have between the period you asked for (120ns) and what the circuit can run at (which should be somewhere around 120 - 30 =90ns). The value will be positive, which tells you that your request for a 120ns period (8.333MHz frequency) has been satisfied.

1.1.3: Print out a screenshot of the "Design Runs" tab, showing all columns up to "DSP". In your own words, and in relation to the lecture material, explain how the WNS is calculated. What is the maximum frequency at which the circuit can run, according to the tools?

Now click again on "Run Simulation". This time, you will see that more options are available, including post-synthesis and post-implementation simulations. This is because now the tools know the timing of the circuit (*back-annotation*). Run the "Post-Implementation Timing Simulation". Note that the same VHDL Test Bench as the behavioural simulation will be applied by default.

As for the behavioural simulation, run for 5us and observe the results. You are likely to see some fluctuations/errors at the start of the simulation (after all, 'U' is not really a valid signal in real hardware), but after the reset things should settle down. Can you observe the differences? Verify that the circuit outputs the correct results. Try to observe the internal signals again: can you find INT1-INT5? How about INTO?

1.1.4: Print out a screenshot of the timing simulation window, zoomed in to display, in readable format, all inputs and the outputs, as well as INTO, in unsigned decimal format. Include the console output.

1.1.5: Try to explain why some internal signals are not available for observation and why some are. Comment on and explain the output of the simulation, particularly the behaviour of INTO and of the output, relating it to the material covered in the lectures.

Close the window and change the clock period in your testbench to 50ns. Re-launch the simulation. Can you observe the differences? Is the output still correct (if it is, you really want to change your input vectors)? Why?

1.1.6: Print out a screenshot of the timing simulation window, zoomed in to display, in readable format, all inputs and the outputs, as well as INTO, in unsigned decimal format. Include the console output.

1.1.7: Comment on and explain the output of the simulation, particularly the behaviour of INTO and of the output, relating it to the material covered in the lectures.

Task 2: Tool optimizations [3 marks]

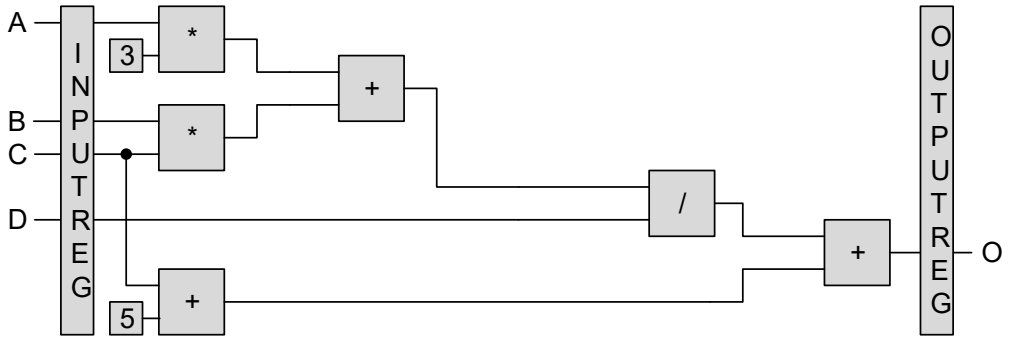
The tools should have reported a WNS of approximately 30ns from the 120ns period. This means that the circuit has a minimum period of approximately 90ns, right?

Open the XDC file and change the period to 85ns. Run the implementation again and check the timing report. Surprised? If all has gone well, the timing requirement has been met, even if you requested a frequency higher than what the first run had told you was possible. Verify this by running the simulation again, with a 85ns period defined in the testbench. Then try the implementation and simulation again by changing the XDC and the testbench periods to 80ns, then 75ns.

1.2.1: Print out a screenshot of the "Design Runs" tab, showing all columns up to "DSP", and calculate the maximum frequency for each implementation of the circuit. Using your own words, explain why the WNS changes and what happens when the timing requirements are not met. Relate this to what you see in the timing simulations.

Task 3: Logic optimizations [3 marks]

Open the VHDL file for your circuit and the RTL schematic. Observe the implementation of the algorithm. Is there anything that can be done to reduce the critical path? Note the sequence of operations and in particular the last two sums. Can you think of any ways to rearrange the order of the operations to reduce the critical path? Consider this re-arrangement:



Modify the circuit (making sure to minimise the vector sizes), then run the implementation again with the 80.0ns clock constraint in the XDC file, and access the port-route timing report. Check the WNS for the clock. Has it changed with respect to your previous result? What can you conclude?

1.3.1: Print out the modified VHDL code. Add comments within the VHDL to illustrate its operation and your modifications (in particular, their effect on the critical path). Print out a screenshot of the "Design Runs" tab, showing all columns up to "DSP", and comment on the comparison between the WNS value here and in the pre-modification case.

Save the current version of the VHDL file as it will be the basis for the following laboratory.

The report required for this lab will consist of the answers / printouts required in the numbered items above.