Welcome to VHDL Exam

Welcome to this exam in VHDL.

You will be challenged with 3 questions, note that you will probably need to answer two of them in order to PASS the exam.

Use the Quartus and ModelSim tools installed on the exam computer in order to solve the tasks in the exam. Please read the attached document to set up the Quartus and ModelSim tools properly.

IMPORTANT!!!
When zipping the Project Folder before uploading your answer. Make sure your design files are placed in this folder to include it in the zip file!

ModelSim and Quartus are the only allowed aids in this exam.

Kent can be reached on mobile during exam: 0703 74 84 29

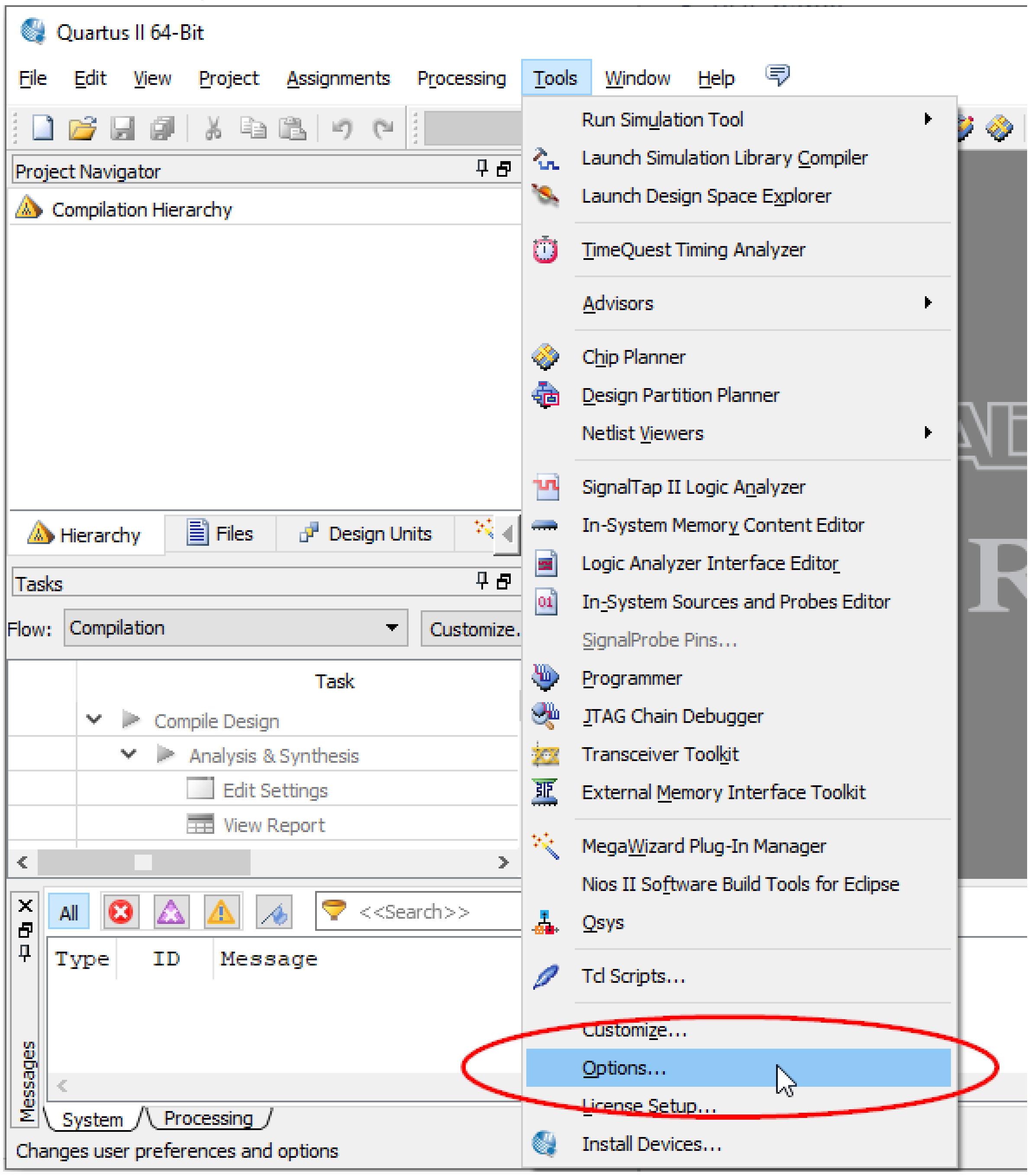
Good luck!

i Modelsim Startup Instructions

When starting Quartus / Modelsim for the First Time you might get a question on which license you want to use.
Choose that you want to run the Web Edition License (which is free).

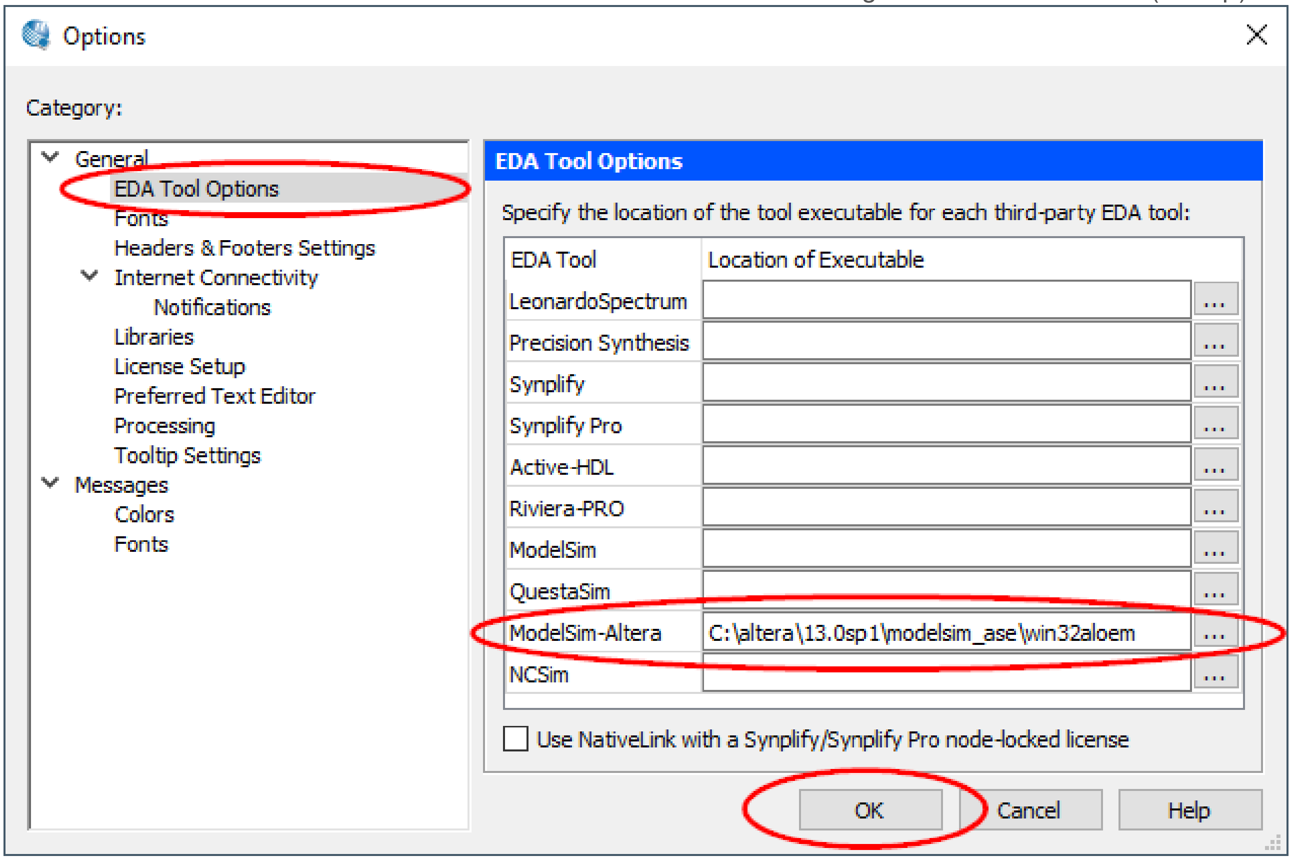
If you have problems starting ModelSim ensure that the EDA Tool Options are set up correctly.

Select Tools -> Options in Quartus



Changes user preferences and options

After this ensure that the path to ModelSim-Altera is set up as shown below:



Specify the location of the tool executable for each third-party EDA tool:

EDA Tool	Location of Executable
LeonardoSpectrum	
Precision Synthesis	
Synplify	
Synplify Pro	
Active-HDL	
Riviera-PRO	
ModelSim	
QuestaSim	
ModelSim-Altera	C:\altera\13.0sp1\modelsim_ase\win32aloem
NCSim	

☐ Use NativeLink with a Synplify/Synplify Pro node-locked license

If the default path was chosen in installation the path should be :
C:\altera\13.0sp1\modelsim_ase\win32aloem

When the above is set up it should be no problem starting ModelSim the way you are used to in this course.

1 Synth Result

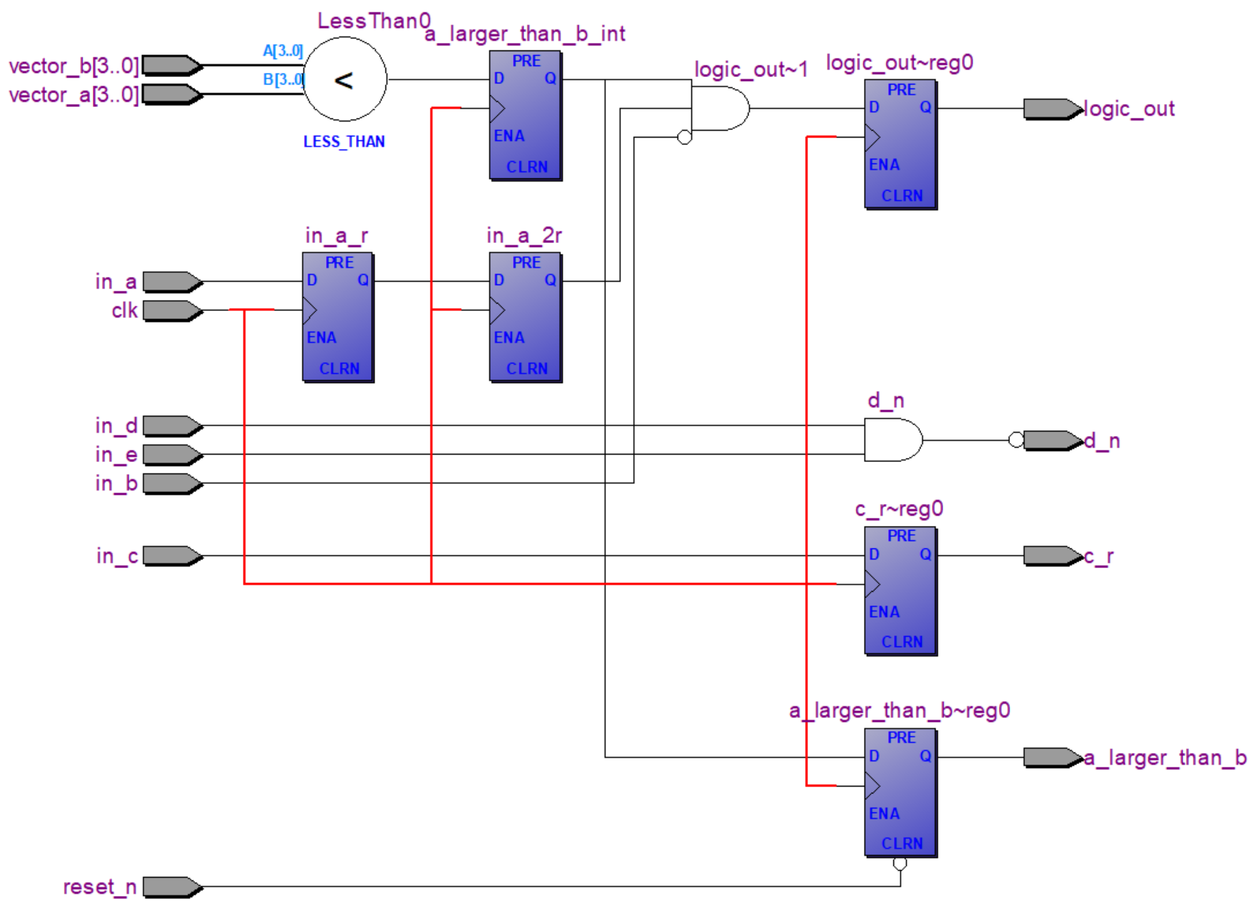
Study the synthesis result from the RTL view below. Write the VHDL code that implements the same functionality as the RTL view shows.

The beginning of the file is provided below, feel free to use it if you like:
synth_question_rtl.vhd

Requirement:
Only synchronous *process(es)* shall be used, however the active low reset_n signal shall be handled asynchronously (as you are used to in this course).

It is the *functionality* of your code that shall be identical to the functionality from the netlist view below, i.e. synthesis tools might draw synthesis result different even though functionality is identical.

The clk signal is highlighted in red to ease interpretation.
Take extra notice on how the reset_n signal is connected.



When finished, archive the whole Quartus project folder into a single zip-file and upload it here as your answer. (or just upload the .vhd-file if no Quartus project has been created)

Ladda upp din fil här. Maximum en fil.

Alla filtyper är tillåtna. Maximal filstorlek är**1 GB**

Välj fil att ladda upp

Totalpoäng: 10

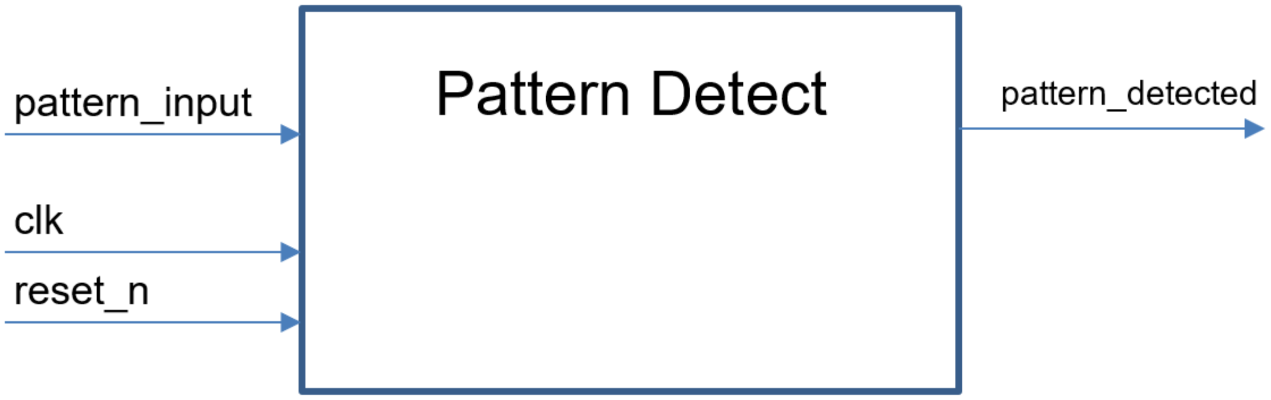
2 pattern_detect

A component called “pattern_detect” shall be designed. The component shall be used to detect a specific pattern on a single bit input called pattern_input. An output signal called “pattern_detected” shall be pulsed high one clock cycle when the pattern is detected successfully.

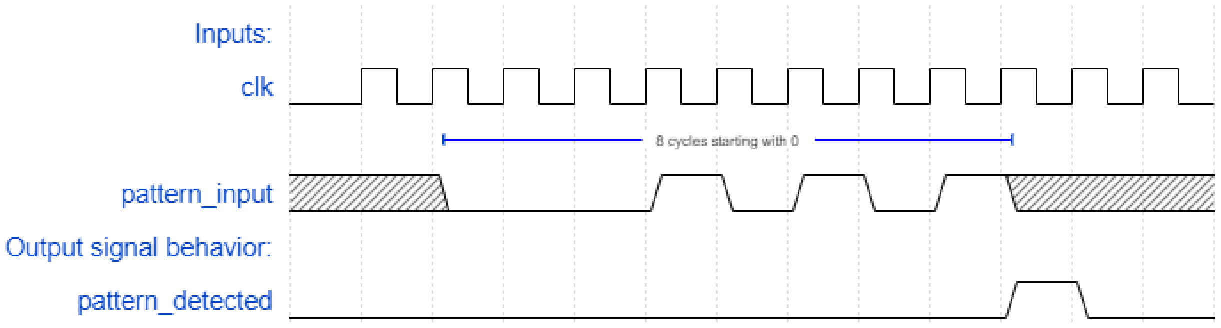
The pattern_input is synchronous with the clk signal, i.e. input is driven by the clk clock domain.

reset_n is an active low asynchronous reset signal which can be used to reset internal states or other signals if needed.

A sketch of the component is shown below (internal signal connections are not shown).



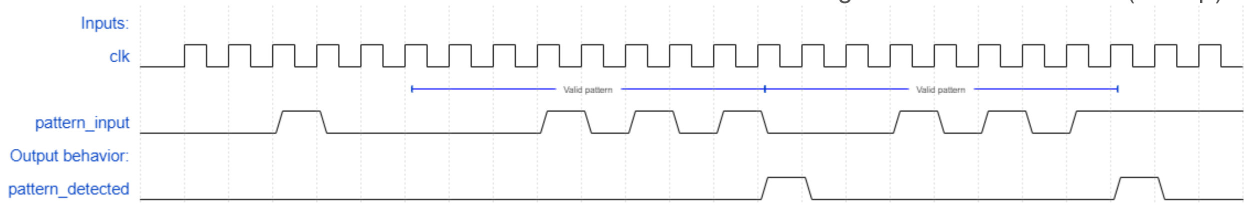
The pattern that shall be detected is 8 clock cycles long and specified according to the following timing diagram:



As seen the pattern starts with three periods of logical low followed by five periods with values 1, 0, 1, 0, 1.

For full point the pattern_detected output signal shall be set high directly after the pattern is detected on the input, exactly as shown above, i.e. timing is critical.

As seen above the input signal level before and after the input pattern shall be don’t care. I.e. there should be no dependencies how the input is set before or after the pattern is detected. Also two patterns sent to the component directly after each other shall be able to be detected as shown in the example below:



Requirements for full point:

- Your task is to design the component to fulfill the above specification and ensure that your component works as shown in the timing diagrams above.
- You are free to implement the functionality using any design strategy but for full point all processes shall be synchronous with the clock signal clk.
- A Quartus project shall be set up with the pattern_detect component as design top level.
- A .do script or testbench shall test a few different input cases to test the pattern_detect component functionality.
- The pattern_detect component shall pass synthesis without any strange warnings that can be solved easily.

Input to this task is:

- A base for the pattern_detect component with a few syntax examples.

This file is found here: task2.zip
Note that there are more syntax examples in task3 files.

Tip #1:

To define the clock signal in a .do-script for Modelsim follow the example below, (which creates a clock with 20 ns period):
force clk 0 0, 1 10 ns -r 20 ns

Create a Quartus project and archive the complete project folder in a zip file before uploading it as your answer.
Make sure to include any .do-scripts or testbench files to show your simulation efforts.

Ladda upp din fil här. Maximum en fil.

Alla filtyper är tillåtna. Maximal filstorlek är**1 GB**

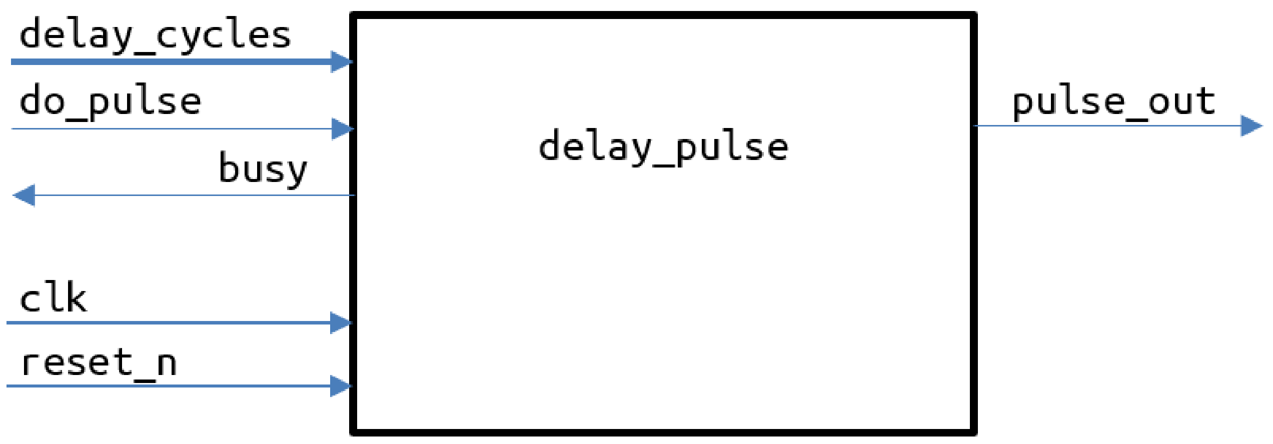
Välj fil att ladda upp

Totalpoäng: 20

3 Delay Pulse - Testbench Check

Testbench check!

In this assignment you shall design a testbench that tests the component named delay_pulse.



The delay_pulse component outputs a one clock cycle pulse on the *pulse_out* signal, *delay_cycles* cycles after *do_pulse* input is set high.

The *delay_cycles* signal is a 10 bit unsigned vector that decides the delay of the pulse out.
The *pulse_out* signal is directly connected to *do_pulse* input if *delay_cycles* is set to zero.
The *busy* signal is set high when *do_pulse* is detected high and *delay_cycles* is not zero.
The *busy* signal is held high until the delayed pulse has been sent out from the delay_pulse component.

For full point the following tasks shall be completed:
- Create a Quartus project using the delay_pulse as the top level component.
- You shall create a testbench that tests all possible delay values and verifies that the delay is as intended.
- busy signal shall be checked to behave as expected.
- Your testbench code shall be well structured with proper indentation.

Input to this task is:
- The delay_pulse component as vhd-file.
- A starting point for the testbench in testbench_top.vhd

Note that the testbench_top.vhd file is given for you to start with but is far from finished and contains a few mistakes. You are of course free to use any design strategy as you wish to test the design, i.e. you can use the testbench_top.vhd file as you wish.
Both files can be found in the zip file below (the downloaded file might need to be renamed after downloaded from Inspera):

delay_pulse_tb_check.zip

When finished:
Archive the whole Quartus project folder in a single zip file and upload here.
Your design efforts will be reviewed for exam points.

Ladda upp din fil här. Maximum en fil.

Alla filtyper är tillåtna. Maximal filstorlek är**1 GB**

Välj fil att ladda upp

Totalpoäng: 20