Welcome to Exam in Digital Electronics

Welcome to this exam in Digital Electronics.

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.

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

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

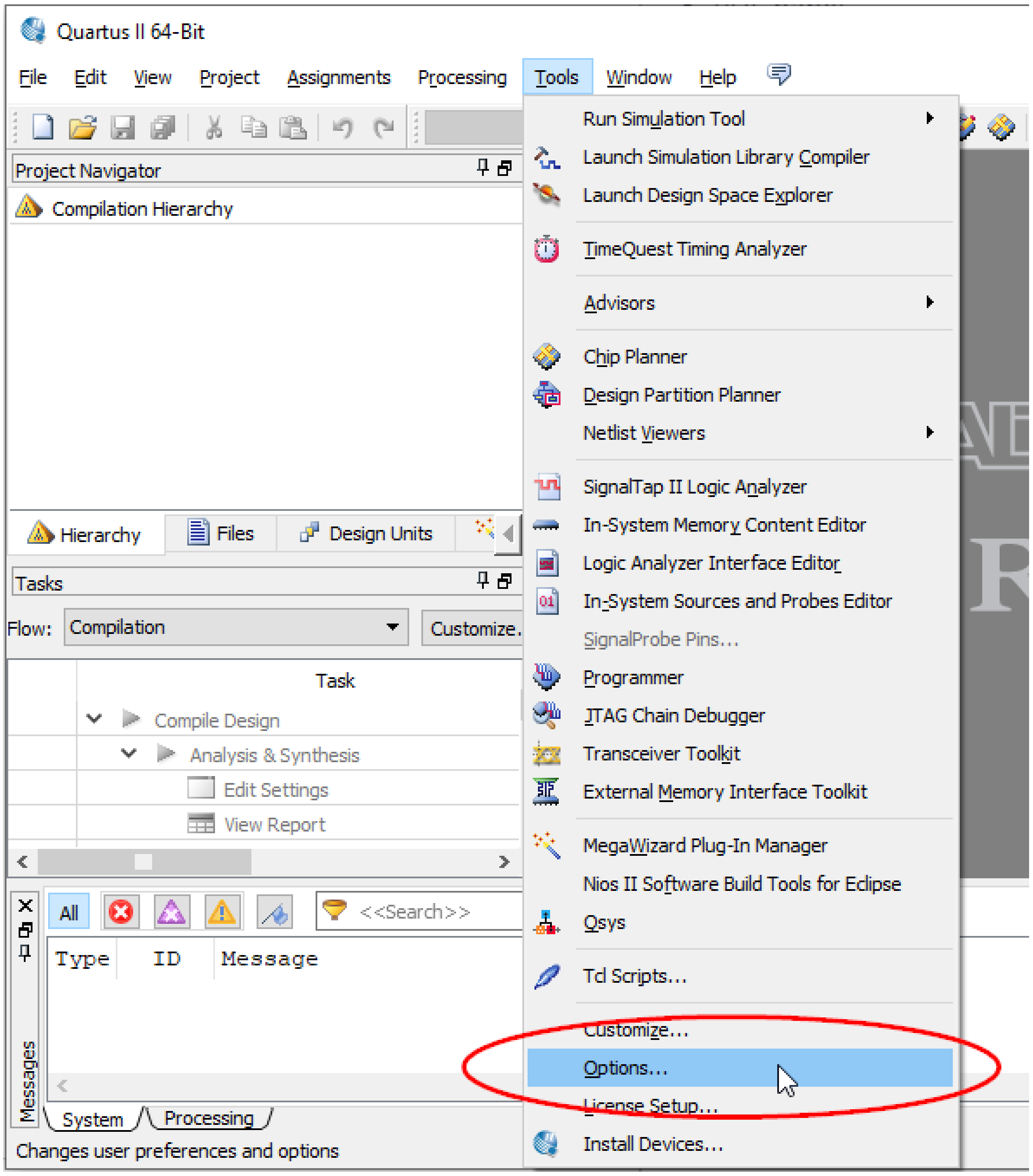
Good luck!

Modelsim Startup Instructions

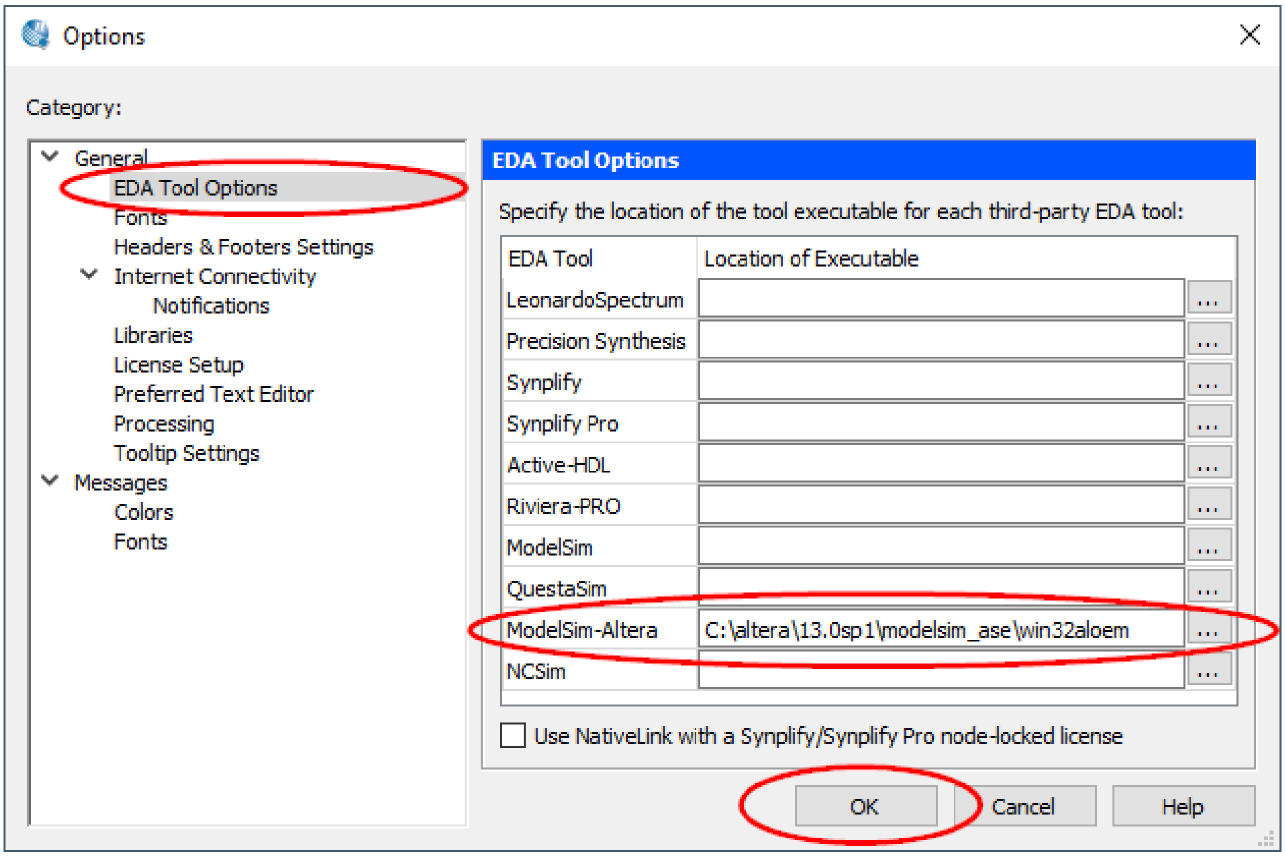
When starting Quartus / Modelsim for the First Time you might get a question on which license you want to use.
Choose that you want to run the **Web Edition License** (which is free).

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

Select Tools -> Options in Quartus



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 De Morgan

Use De Morgan's Theorem and replace the AND gate in the following VHDL expression with an OR gate instead.

```
q <= (not in_a) and (not in_b);
```

This means that you are only allowed to use an OR gate and inverter(s), i.e. "or" & "not" in your answer.
The functionality shall be the same, you shall still assign the signal q and use the signals "in_a" and "in_b".

Give your answer in the text field below:

Totalpoäng: 2

2 Synth Result

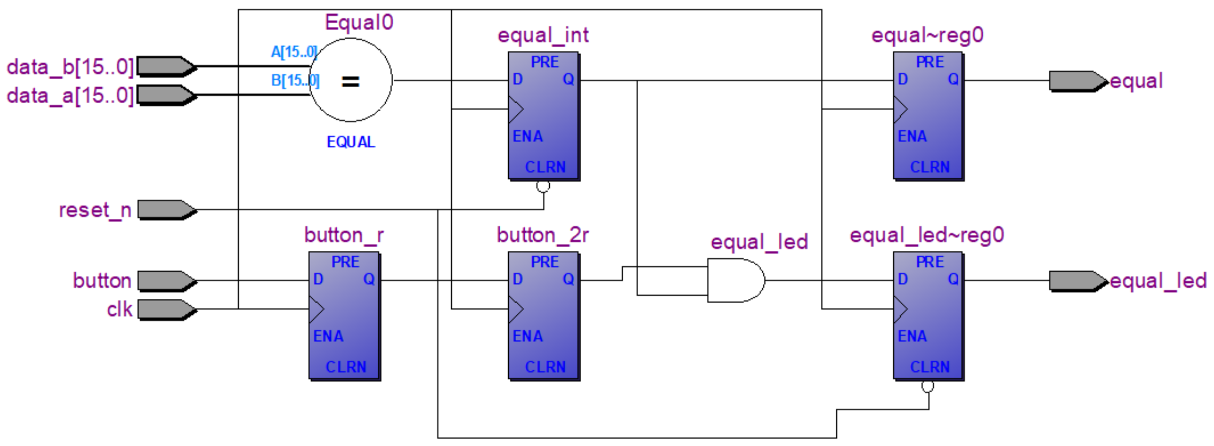
Study the synthesis result from the RTL view below.
Write the VHDL code that implements the same functionality as the RTL view shows.

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

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**

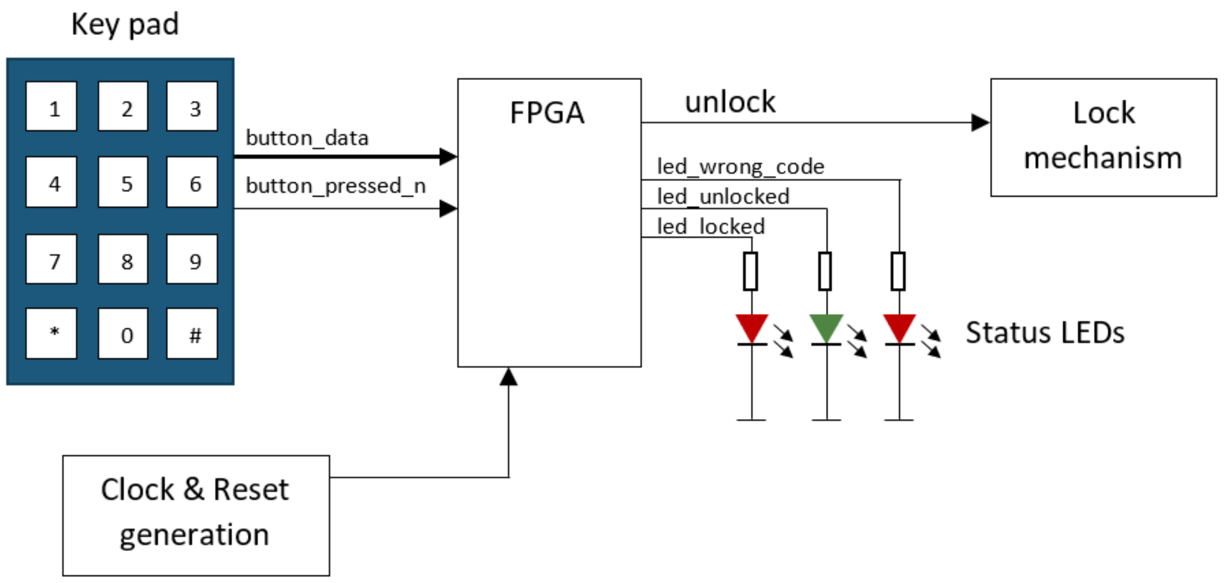
Välj fil att ladda upp

Totalpoäng: 8

3 KeyPadLock

A component named KeyPadLock shall be designed.
The component shall receive signals from a keypad connected to the FPGA. An "unlock" output signal is routed to a lock mechanism which, when set high, unlocks the "door to exam points". Additional LED outputs are used to display the state of the design.

System Sketch below:

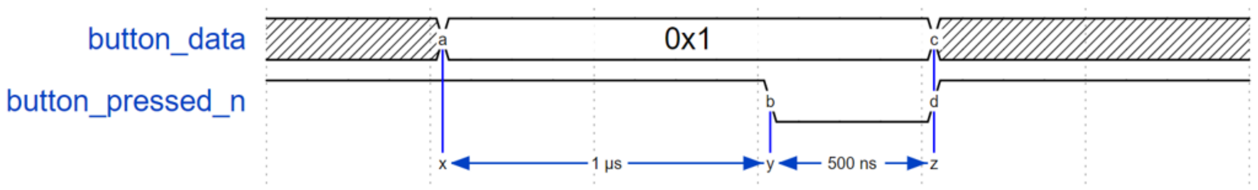


The Key pad is specified below:
The Key Pad is a hardware component that outputs a 4 bit vector together with a pulse when a button is pressed. The buttons are debounced by the key pad hardware which ensures only one pulse each time a button is pressed.
The button_pressed_n signal works as a uni directional (pure input) strobe signal and is set low together with valid data on the 4 bit bus named button_data.

The table below shows how the button_data represents the key being pressed.

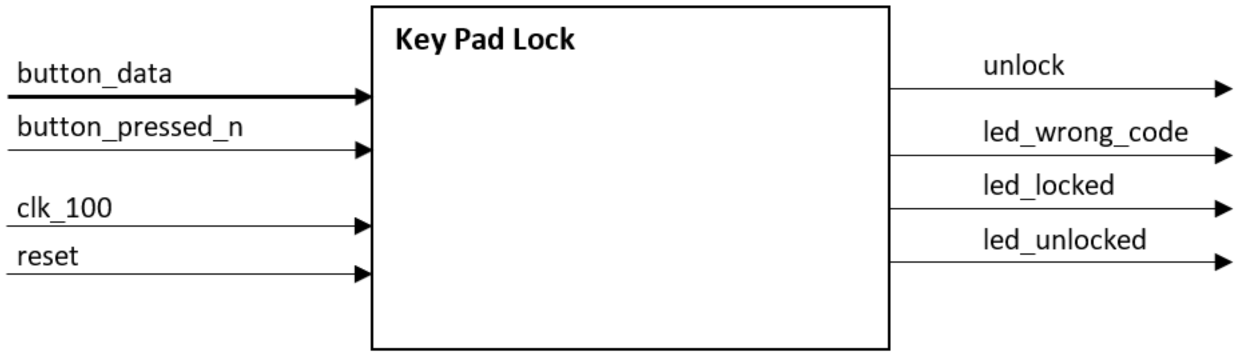
Key Pad Key (or other data)	button_data signal value (4 bit data)
Unknown key pressed (e.g. several keys)	Hex 0x0 = “0000” binary
Key 1	Hex 0x1 = “0001” binary
Key 2	Hex 0x2 = “0010” binary
Key 3	Hex 0x3 = “0011” binary
Key 4	Hex 0x4 = “0100” binary
Key 5	Hex 0x5 = “0101” binary
Key 6	Hex 0x6 = “0110” binary
Key 7	Hex 0x7 = “0111” binary
Key 8	Hex 0x8 = “1000” binary
Key 9	Hex 0x9 = “1001” binary
Key 0	Hex 0xA = “1010” binary
Key *	Hex 0xB = “1011” binary
Key #	Hex 0xC = “1100” binary
Keypad Error	Hex 0xD = “1101” binary
Reserved	Hex 0xE = “1110” binary
Reserved	Hex 0xF = “1111” binary

A timing diagram of how one nibble (4 bits of data) with example value 0x1 is transmitted from the Key pad after a “Key pressed event” is shown below. Data is sent out 1 μs before the button_pressed_n signal falls low for 500 ns. The specified times have a tolerance of 5%.



KeyPadLock specification (your task):

Your task is to write a VHDL component which receives the signals from the Key Pad and controls the unlock and LED outputs.



A four digit code, set by generics shall, when entered correctly, toggle the unlock output signal from low to high or from high to low. Make sure you sample the incoming data properly and also make sure that incoming data is handled correctly in the clk_100 clock domain.

- The led_locked signal shall be set high when the unlock signal is set low.
- The led_unlocked signal shall be set high when the unlock signal is set high.
- The led_wrong_code shall be held high, and all new button press events shall be ignored during exactly 5 ms if the wrong code is entered (this timeout is set low to speed up simulation).

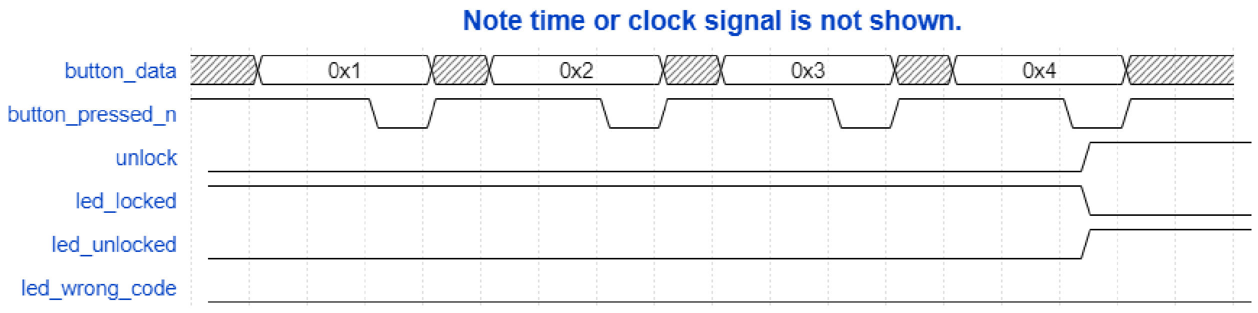
Note that the user should not see if the code is entered wrong before all 4 digits have been entered.

The clk_100 clock input shall be used, this is an 100 MHz clock. The active high reset signal shall be used to reset internal signals that needs reset.

The unlock output shall be low during and after reset.

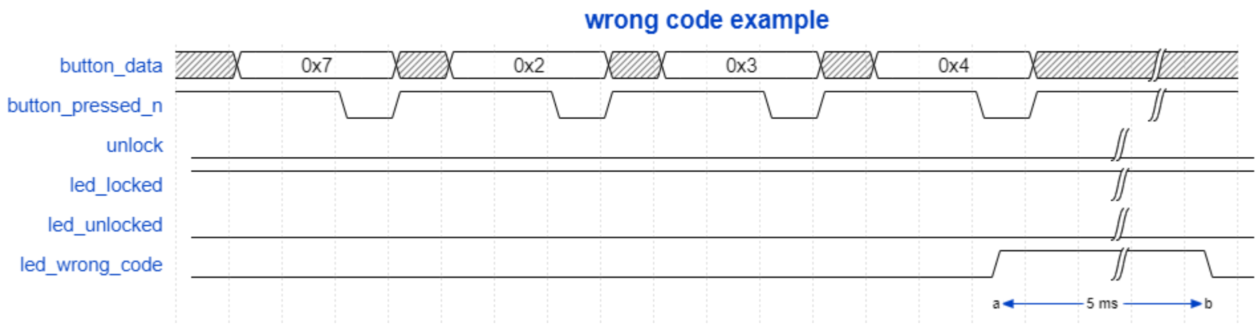
Example 1:

If correct code is set to 1, 2, 3, 4, the unlock sequence should look something similar to below sketch, note that exact time is not specified below



Example 2:

If correct code is set to 1, 2, 3, 4, and the wrong code is entered the output signals should be controlled approximately as the sketch below shows.



For full point the following tasks shall be completed:

- Create a Quartus project using the KeyPadLock as the top level component.
- A .do script shall be written to test the design in simulation.
- All processes used shall be synchronous with the clk_100 clock, you are allowed to handle reset either synchronously or asynchronously, just ensure that all important signals such as e.g. state signals, are reset properly.
- Your code shall be well structured with proper indentation.
- The "wrong_code LED" signal shall be set high exactly 5.000000 ms if wrong code is received.
- The code shall be synthesizable without any strange warnings that can be solved easily.

Input to this task is:

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

All located here: key_pad_lock_rtl.vhd

Tip:

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

```
force clk 0 0, 1 5 ns -r 10 ns
```

When finished:

Archive the whole Quartus project folder in a single zip file and upload here.

Ladda upp din fil här. Maximum en fil.

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

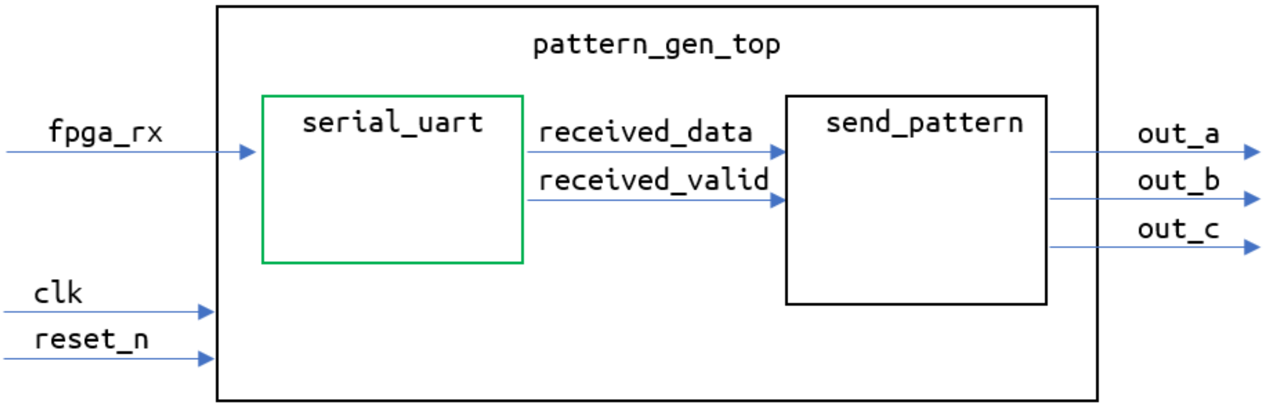
Välj fil att ladda upp

Totalpoäng: 20

4 Testbench - Pattern gen

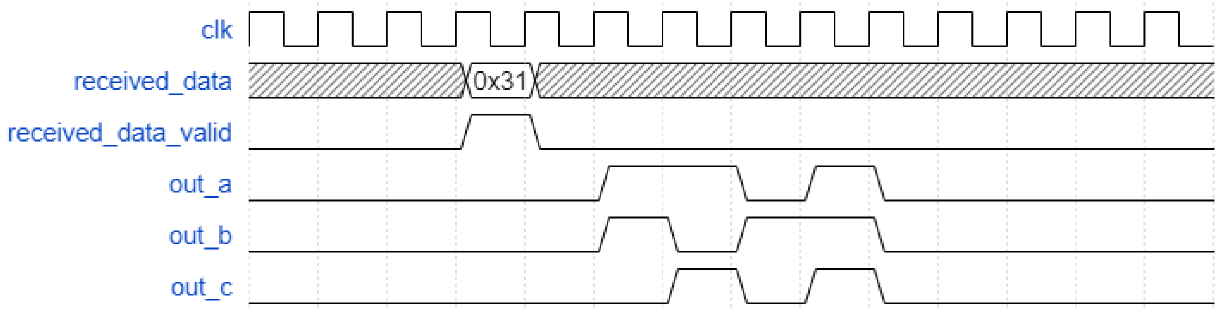
In this task you shall finish a VHDL design called "Pattern generator". The goal is to finish the design as described below and make it pass the testbench tests in the attached testbench.

The top level block diagram is shown below

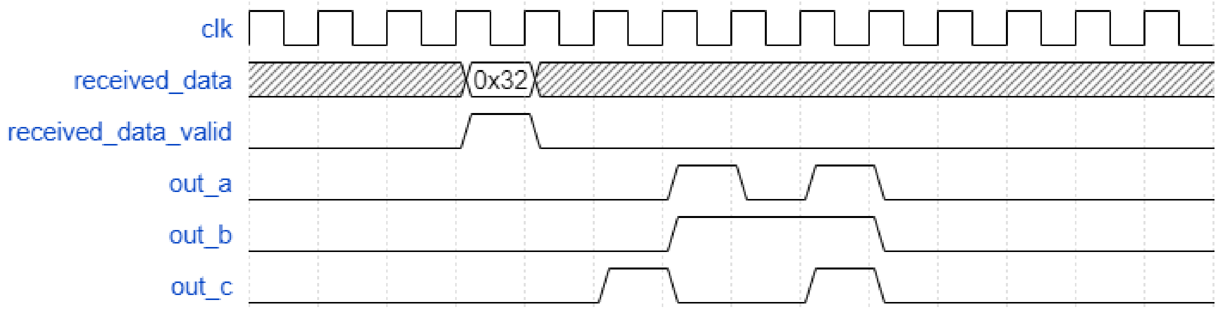


The serial_uart component is given to you as an input file but you need to instantiate and set up the generics properly in the pattern_gen_top component.

The send_pattern sub-component shall receive one byte of data from the serial_uart component and output two different patterns, each triggered by receiving a specific byte of data.
"Pattern 1" specified in the timing diagram below shall be sent out when receiving the ASCII character for a "1" which is the hex value **0x31**.



"Pattern 2" specified in the timing diagram below shall be sent out when receiving the ASCII character for a "2" which is the hex value **0x32**.



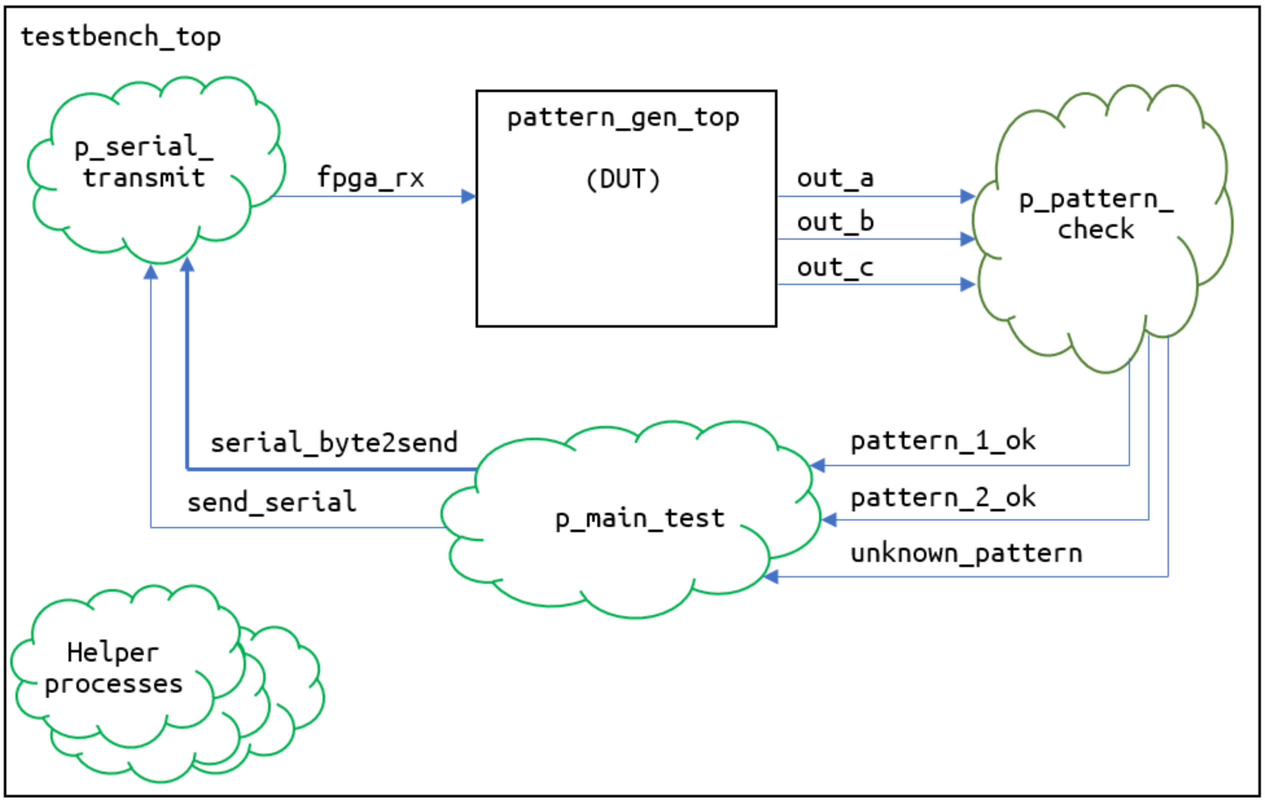
- The clock signal shall be handled as a 100 MHz clock signal
- The reset_n, reset signal shall be used as an active low reset, it's optional if you wish to use it as an asynchronous or synchronous reset.
- All outputs shall be held low when in idle, i.e. when no pattern is sent out.

- Both patterns are defined to be 4 clock cycles long, starting with at least one output set high, i.e. the patterns are shown in full above.
- All received data that is different from the "pattern triggers" (0x31 / 0x32) shall be ignored, and all outputs shall still be held low in this case.
- The time between receiving the byte of data from the serial UART component and starting the pattern output is not required to follow the timing diagrams above exactly, but if a pattern shall be sent it should be finished within 20 clock cycles after receiving the byte of data (received_data_valid set high).
- Serial UART shall be set up through generic map to use serial speed of **115200 bps**, and **no parity**.
- All unused outputs on the Serial UART shall be left open and inputs shall be set to static inactive values.

Testbench

The pattern generator top level shall be tested in the testbench which is supplied in this task. **The testbench contains a few simple syntax errors or minor mistakes** that needs to be fixed in order to get it started, note that this is only simple mistakes and no big redesign is needed.

A testbench block diagram sketch is shown below, all signals are not shown.



For full point the following tasks shall be completed:

- Requirements specified above shall be fulfilled

- Create a Quartus project using the pattern_gen_top as the top level component.

- All processes you design shall be synchronous with the 100 MHz clock, you are allowed to handle reset either synchronously or asynchronously, just ensure that all important signals such as e.g. state signals, are reset properly.

- Your code shall be well structured with proper indentation.

- The pattern_gen_top level and its sub-components shall be synthesizable without any strange warnings that can be solved easily.

Input to this task is:

- Serial UART
- A base for the pattern_gen_top.
- The testbench top file (needs minor debugging).

All located in this zip file:

Task3.zip

When finished:
Archive the whole Quartus project folder in a single zip file and upload here.

Ladda upp din fil här. Maximum en fil.

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

Välj fil att ladda upp

Totalpoäng: 20