

## **i 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!

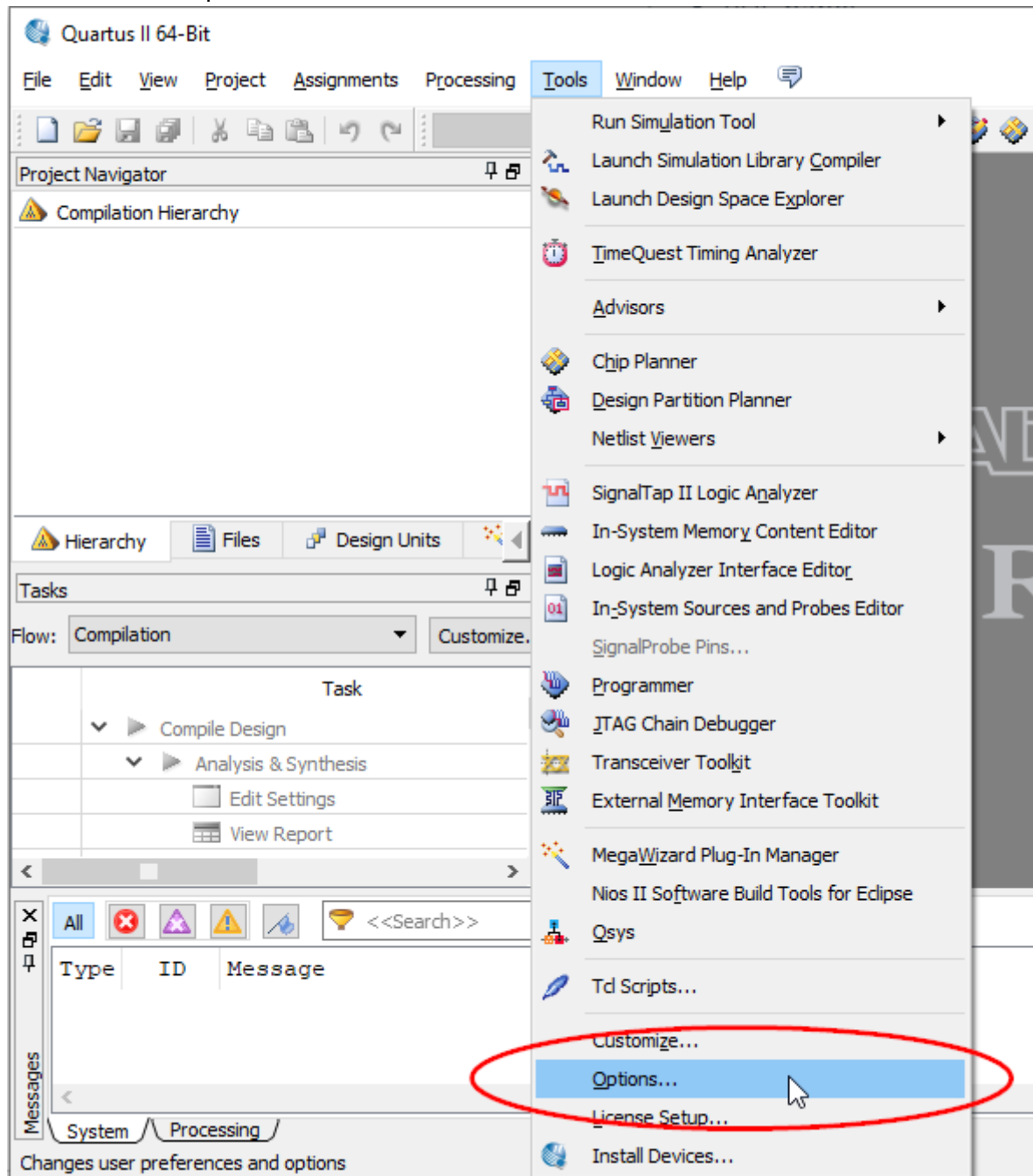
## i Modelsim Startup Instructions

When starting Quartus / Modelsim for the First Time you might get a question on which license you want to use.

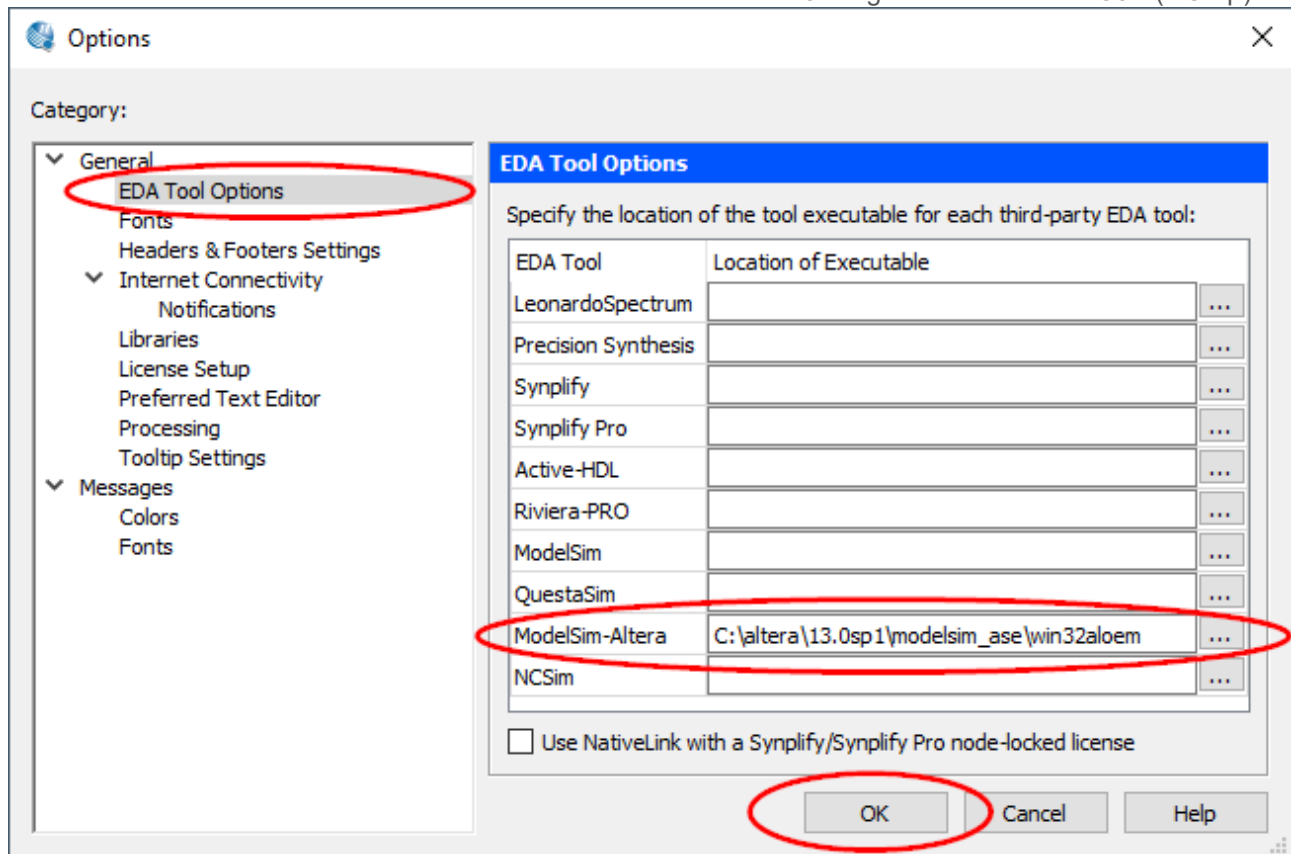
Choose that you want to run the **Web Edition License** (which is free).

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

Select Tools -> Options in Quartus



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



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](#)

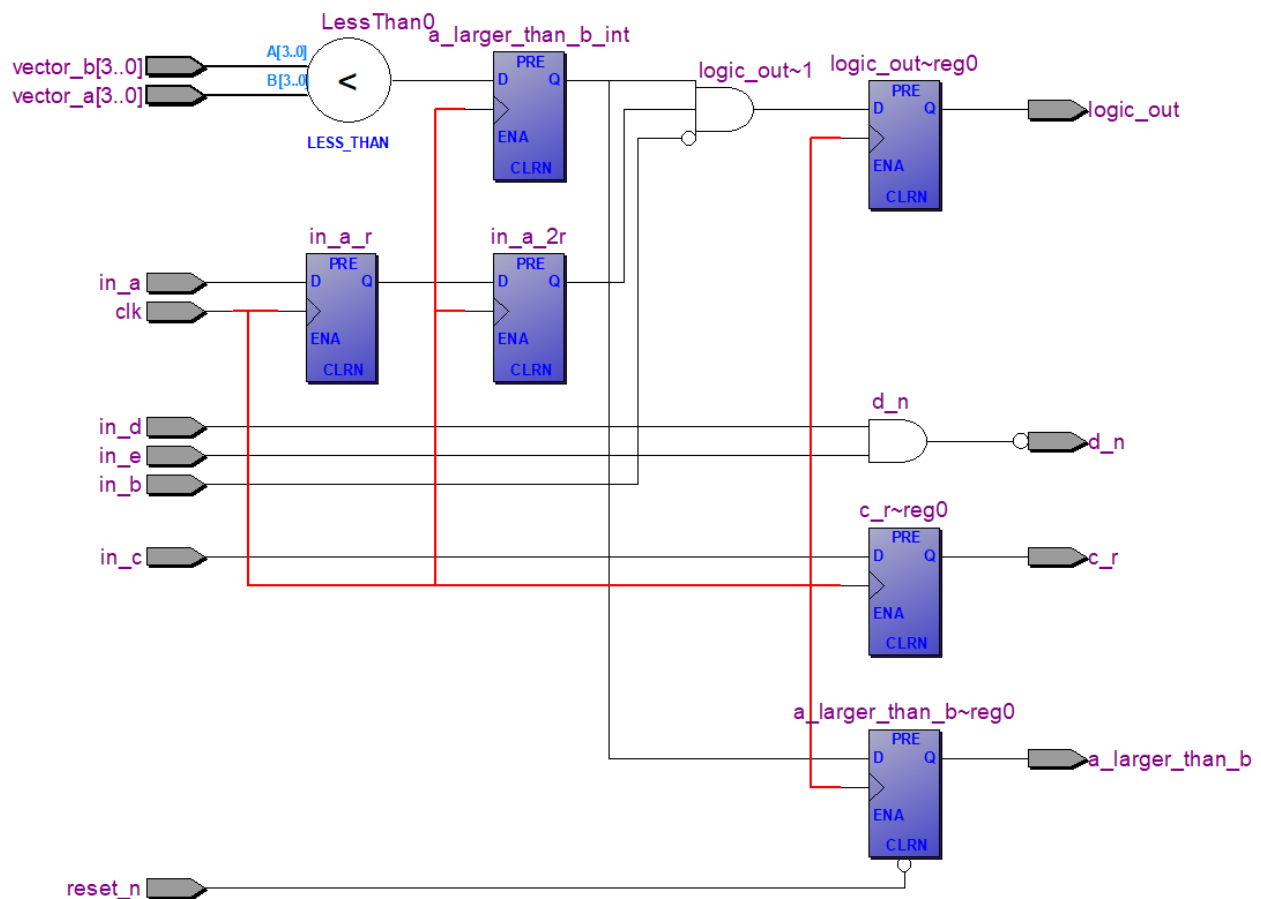
## Requirement:

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

---

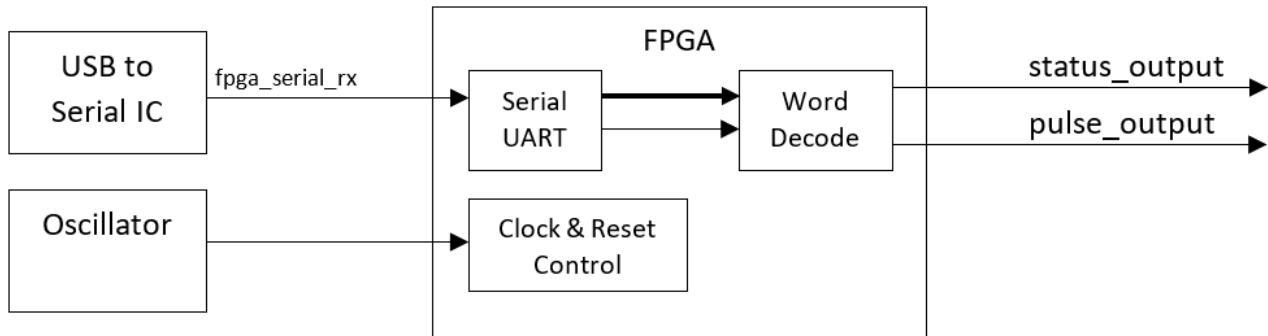
Totalpoäng: 10

## 2 Word Decode

A component named "Word Decode" shall be designed.

The component shall receive ASCII characters from a serial interface and control two output signals based on the given command on the input.

**System sketch below (all signals are not shown):**



Three commands shall be supported, and these are:

- **"on"** - switch status output on (1), no change if status output is already on
- **"off"** - switch status output off (0), no change if status output is already off
- **"pulse"** - a 840 ns long (active high) pulse on pulse\_output shall be generated

All other commands (series of ASCII characters) shall be ignored i.e. not cause any change on the outputs.

**Stripped down ASCII table below:**

Character	Hex Value	Dec Value
e	0x65	101
f	0x66	102
l	0x6C	108
n	0x6E	110
o	0x6F	111
p	0x70	112
s	0x73	115
u	0x75	117

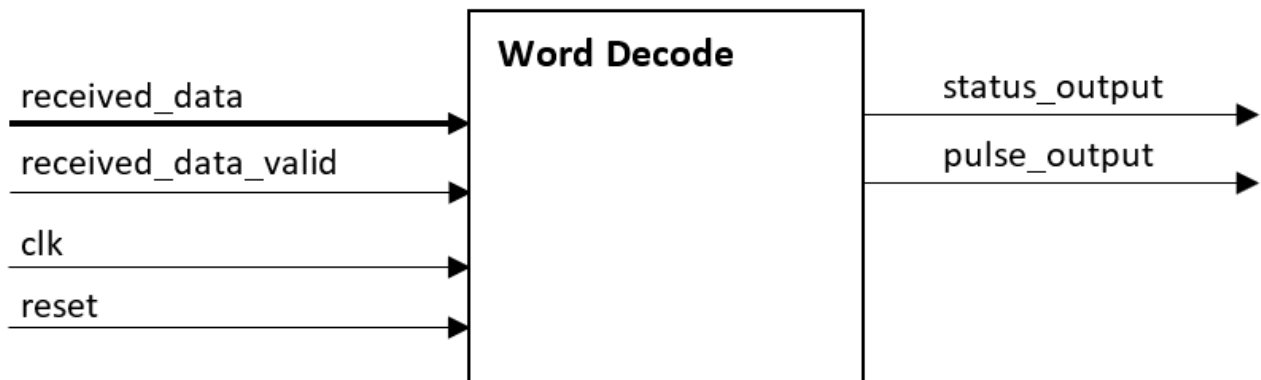
A Serial UART component is instantiated outside the "Word Decode" component. Clock and an initial reset is also taken care of in the FPGA Top Level outside the "Word Decode" component (as shown in the system sketch above).

The Serial UART component will output the received ASCII data on the 8 bit received\_data vector together with a one clock cycle high pulse on received\_data\_valid. The serial transmission speed is not expected to be higher than 115200 bps, which means that it will be at least 86  $\mu$ s between every received data byte (every pulse on received\_data\_valid). The Serial UART component is connected to the same clock signal as the "Word Decode" component.

The clock signal is a **50 MHz clock**, the reset will be **active high** for at least 10 clock cycles after FPGA configuration, i.e. when FPGA has started. Reset will never be set high after this.

Your task is to design the "Word Decode" component which receives the ASCII character data, byte by byte, and control the outputs.

Word Decode component sketch below:



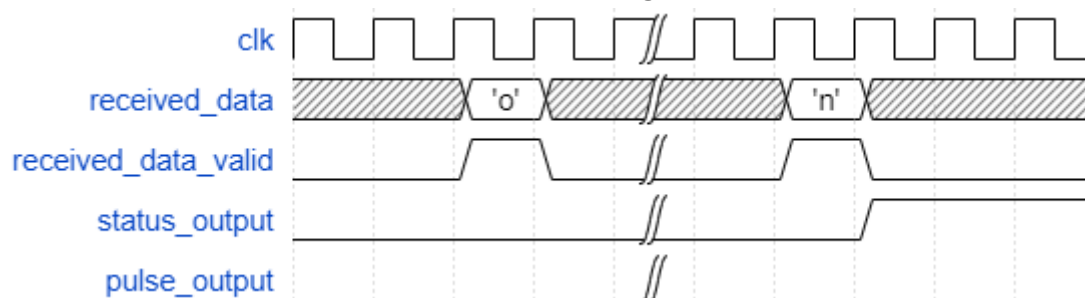
Both status and pulse outputs shall be set low during initial reset.

The timing diagrams below show examples on how incoming data towards the Word Decode component could look and how output signals are expected to be controlled. The data on the 8 bit input vector received\_data\_valid is shown as ASCII characters in the timing diagrams.

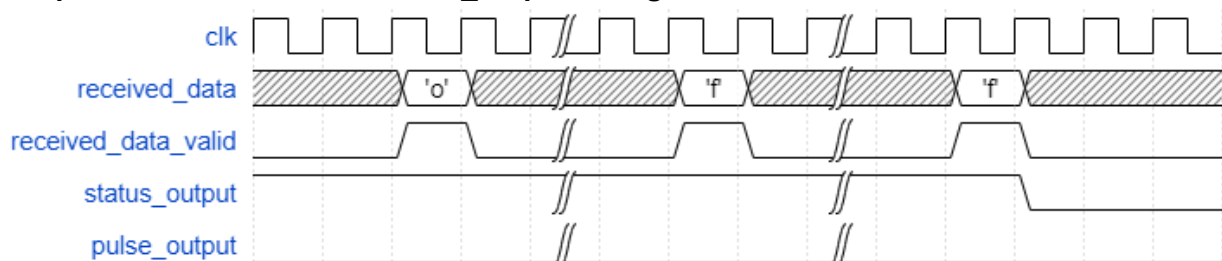
There are no requirements on exact timing relation between incoming data and output signals.

The vertical "cuts" in the timing diagrams means that there is an unknown number of clock cycles cut out in the timing diagram to keep it more compact.

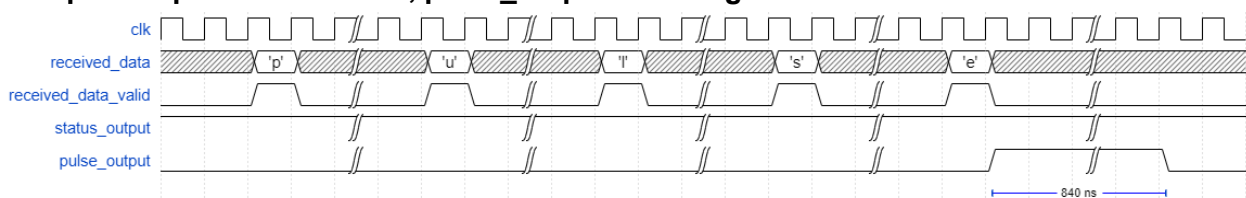
**Example 1 - "on" command, status\_output changes from 0 to 1**



**Example 2 - "off" command, status\_output changes from 1 to 0**



**Example 3 - "pulse" command, pulse\_output is set high for 840 ns**



**Error handling:**

All unknown ASCII values shall abort any internal decoding sequence and be ready for a new command. E.g. if the command "pulx" is received the "x" character shall make the decoding of commands to be reset. This ensures that sending an unknown character, e.g. 'x' will reset any internal decoding sequence and make sure that a new command is handled as specified. This is the only error handling that is required, i.e. you do **not** need to handle the character sequence "ofpulse" as a valid pulse command. But the character sequence "ofxpulse" shall be able to be decoded as a valid pulse command.

**For full point the following tasks shall be completed:**

- Create a Quartus project using the Word Decode (word\_decode) as the top level component.
- A .do script shall be written to test the design in simulation. All valid commands and at least one case with an unknown character shall be tested.
- All processes used shall be synchronous with the "clk" clock signal, you are allowed to handle the **active high 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 code shall be synthesizable without any strange warnings that can be solved easily.

**Input to this task is:**

- A base for the word\_decode component with a few syntax examples.  
located here (word\_decode\_rtl.vhd):

**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
```

**Tip #2:**

To force a signal vector named input\_vector to a decimal value in Modelsim follow the example below:

```
force input_vector 10#111
```

To force a signal vector named input\_vector a hexadecimal value in Modelsim follow the example below:

```
force input_vector 16#6F
```

**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**

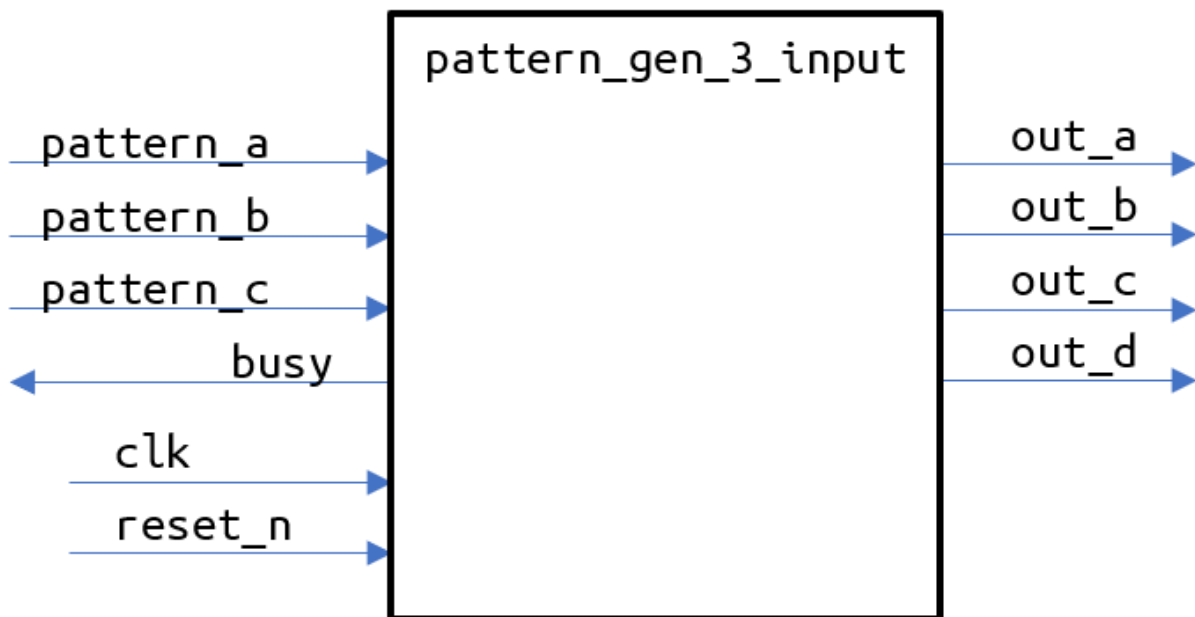
 Välj fil att ladda upp

---

Totalpoäng: 20

### 3 Testbench - Pattern gen 3 inputs

In this task you shall create a VHDL component called "Pattern Gen 3 inputs". The goal is to design the component as specified below and make it pass the testbench tests in the attached testbench. The design block diagram is shown below



The pattern generator shall generate 3 different signal patterns, 5 cycles long on the output signals out\_a/b/c/d when requested by an incoming pulse on one of the pattern\_a/b/c inputs.

The busy output signal shall be held high when the component is busy generating the required output patterns.

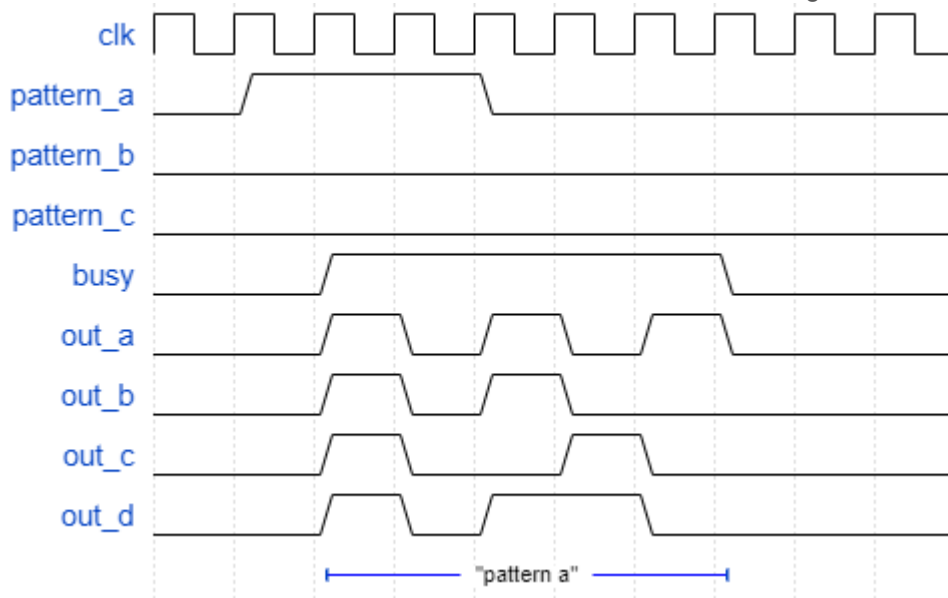
The signal patterns are defined in the timing diagrams below, all signal patterns are 5 clock cycles long. All output signals shall be held low when no signal pattern is generated.

The busy signal shall be set high immediately (one cycle after) one of the pattern\_a/b/c inputs are set high. The busy signal shall be held high until the component is finished generating the output signal patterns, this may take a few cycles longer than the actual pattern but the pattern generator shall be finished, and the busy signal shall be set low within 10 cycles from when it was set high. There shall be at least one cycle pause (where all outputs are low) between all generated patterns.

If more than one of the pattern\_a/b/c inputs are set high simultaneously this shall be considered to be an invalid request and the input shall be ignored.

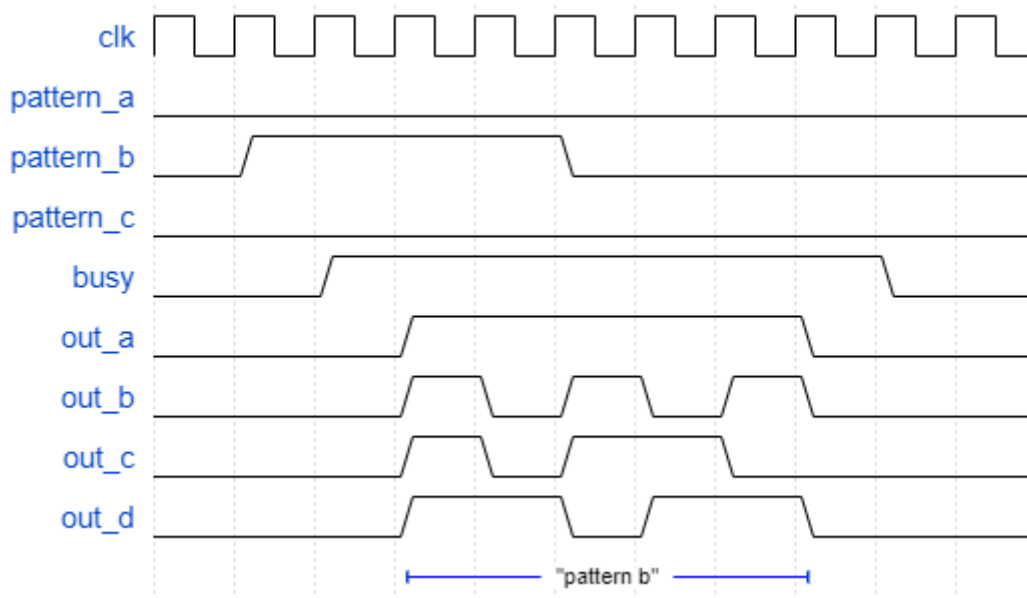
#### Example - "Pattern a"

"Pattern a" is specified in the timing diagram below, note that busy signal is set high immediately after pattern\_a is detected high.



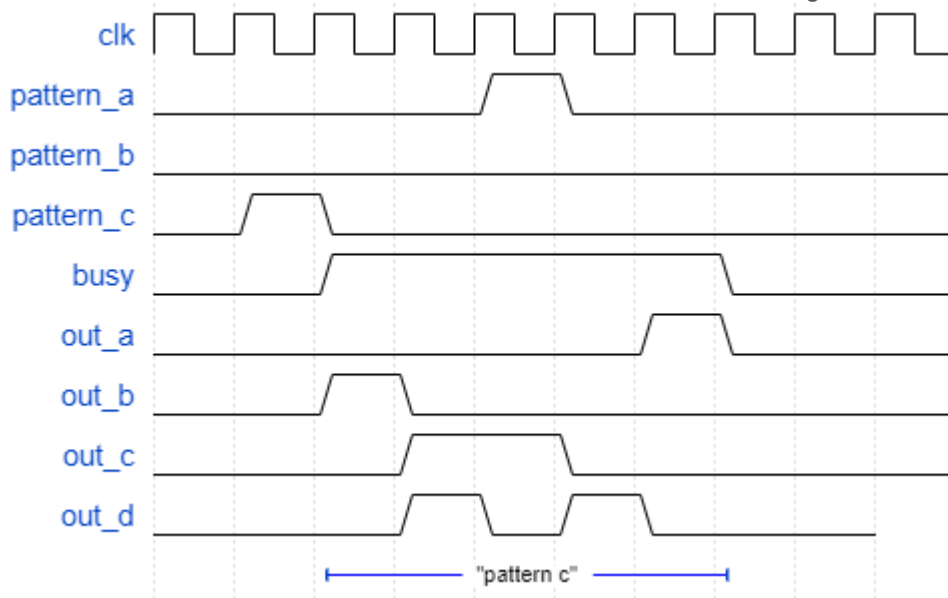
### Example - "Pattern b"

"Pattern b" is specified in the timing diagram below. In this example the busy signal is held high a little longer than the pattern output and pattern output is also delayed one cycle compared to "pattern a" example above. This behavior is also OK!



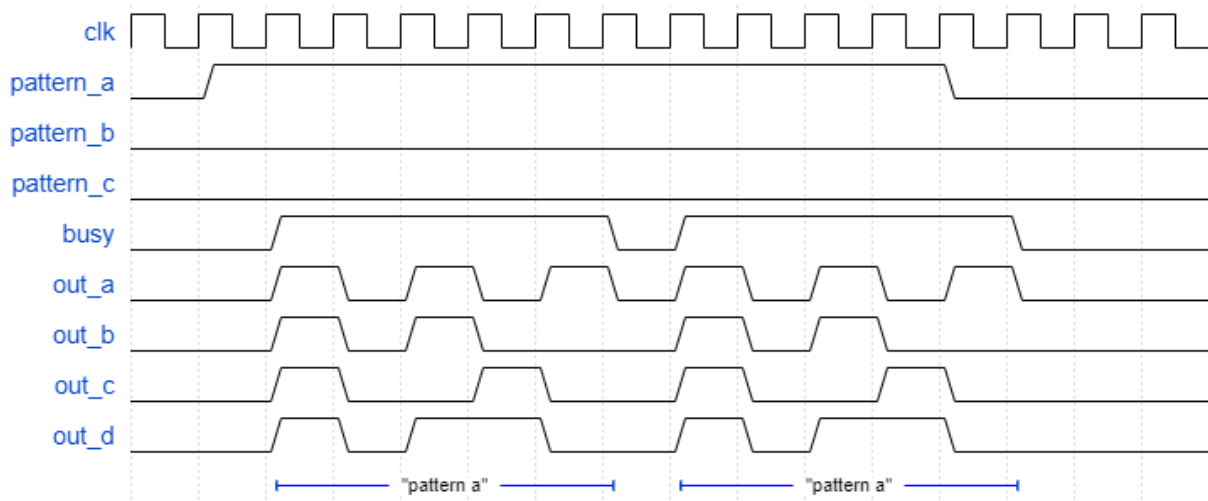
### Example - "Pattern c"

"Pattern c" is specified in the timing diagram below. In this example the pattern\_a input is pulsed when the component has set the busy signal high, the "pattern a request" shall be ignored in this case.



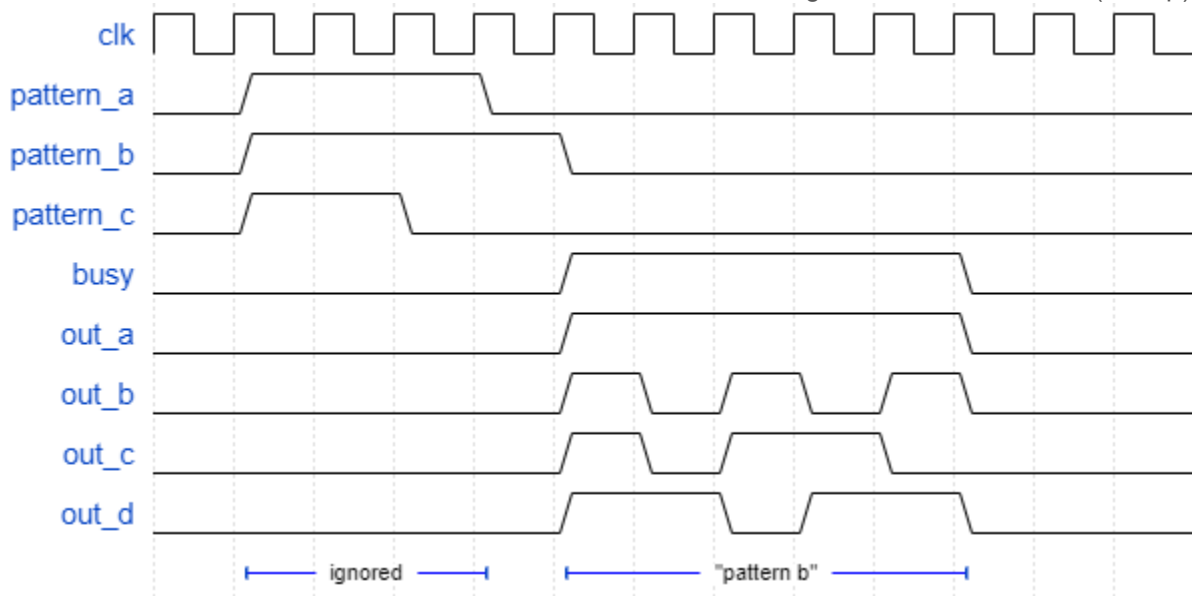
### Example - Input held high

If one of the pattern\_a/b/c signals are held high the requested pattern shall be generated over and over again. The busy signal shall be pulsed low in this case for every time the output pattern has been generated and pattern generation restarts. This is shown in example below.



### Example - Inputs ignored

As described above, if more than one of the "pattern\_a/b/c" inputs are set high the inputs shall be ignored and no pattern shall be generated, this is illustrated in the timing diagram below.

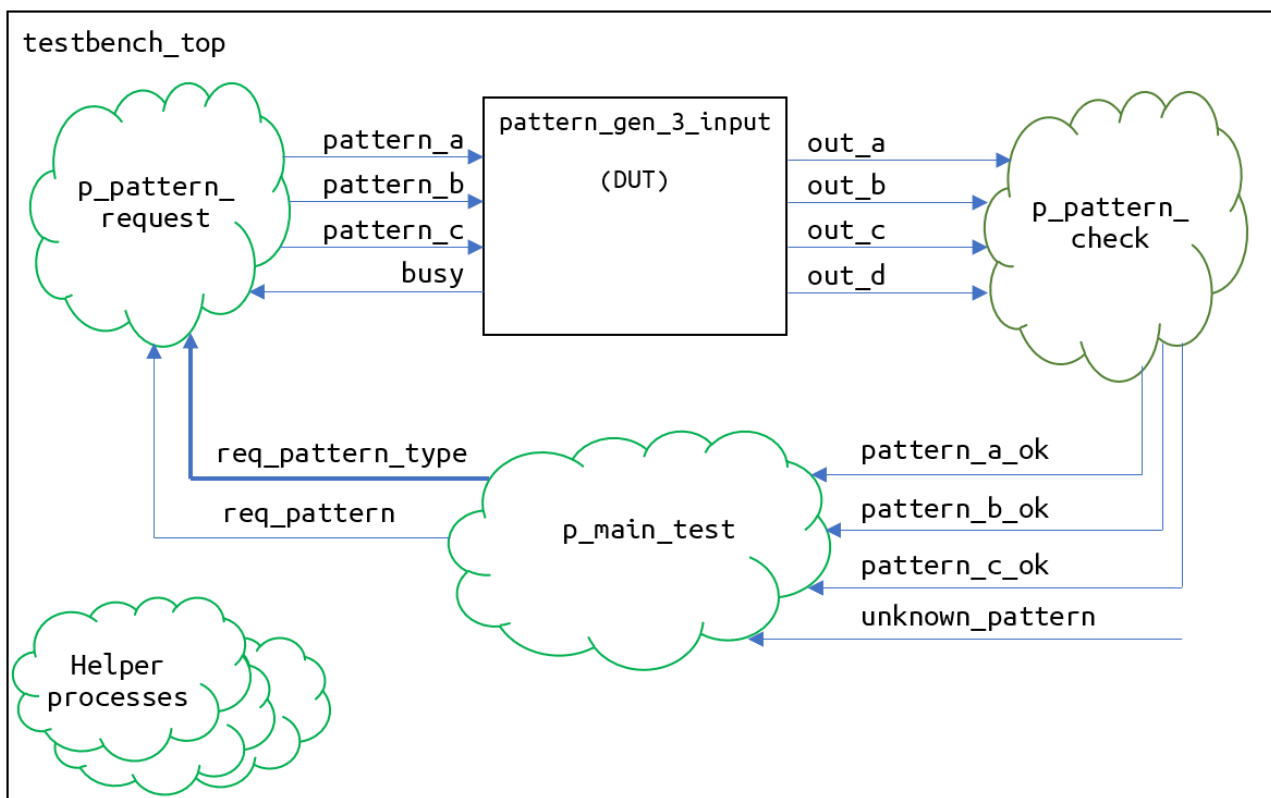


- The reset\_n, reset signal shall be used as an active low reset, it's optional if you wish to use it as an asynchronous or synchronous reset.
- All outputs shall be held low when in idle, i.e. when no pattern is sent out.
- There shall be at least one cycle pause (where all outputs are low) between all generated patterns.

## Testbench

The pattern generator 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:**

- Functional requirements specified above shall be fulfilled
- Create a Quartus project using the pattern\_gen\_3\_input design as the top level component.
- All processes you design shall be synchronous with the clock signal, you are allowed to handle reset either synchronously or asynchronously, just ensure that all important signals such as state signals, are reset properly.
- Your code shall be well structured with proper indentation.
- The pattern\_gen\_3\_input design file shall be synthesizable without any strange warnings that can be solved easily.

**Input to this task is:**

- A base for the pattern\_gen\_3\_input design file.
- 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