FYS4220 Lab. exercise 1

August 30, 2013

Introduction and requirements

This lab exercise contains two parts. In the first part, *1a)* and *1b)*, you will implement a simple design to connect the onboard toggle switches to the onboard LEDs, and then a 7-segment decoder to show the value of the first 4 toggle swithces on the 7-segment display. To verify the correct behaviour of the 7-segment decoder you will use a test bench design which already has been prepared for you.
In the second part *2a)* you will implement a 4-bit counter that will be used as input to the 7-segment decoder. The counter will be synchronous to the onboard 50MHz crystal, and a push butten will be used to activate the counter (synchronous active low enable). To verify the correct behaviour of the counter you will again use an already prepared test bench design. The second part also contains an optional assignment *2b)* where an edge detector will be implemented to detect the falling edge of the push button signal and generate an internal enable for the counter that has a duration equivalent to one clock cycle of the 50MHz clock.

Requirements (due date: 20/9-2013):

- The complete and final version of the lab1_gx.vhd design file containing all the relevant sub-assignemnts.
- A well structured report (with coverpage and in pdf-format) describing what you have done and any problems you have encountered during the work. The report should specifically answer and discuss the questions 10,11, and 12 in part 1b) and 11 in part 2a).
- The report should also include a screenshot of the various wave diagrams produced when simulating the the 7-segment decoder and 4-bit counter.
- The lab1_gx.vhd file and report (pdf-format) should be attached and submitted by email to ketil.roed [at] fys.uio.no. The filename of the report should contain your name(s) and group number (group numbers will be defined soon). Avoid the use of spaces in the filename (an underscore can instead be used as a separator if needed).
- The lab. exercise will be approved only after the report and vhd-file have been submitted and a demonstration of the work has been carried out.

Part 1)

1 a) Switches and LEDs

The purpose of this first exercise is to learn how to connect simple input and output devices to an FPGA chip and implement a circuit that uses these devices. We will use the switches SW17-0 on the DE2 board as inputs to the circuit and we will connect the outputs to the light emitting diodes (LEDs). The DE2-70 board provides 18 toggle switches, called SW17-0, which can be used as inputs to a circuit, and 18 red LEDs, called LEDR17-0, that can be used to display output values. Figure 1 shows a simple VHDL description that uses these switches and shows their states on the red LEDs. Since there are 18 switches and lights it is convenient to represent them as arrays in the VHDL code, as shown.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity lab1_gx is
  port (
    LEDR : out std_logic_vector(17 downto 0);   -- Red LEDs
    SW   : in  std_logic_vector(17 downto 0));  -- Switches
end entity lab1_gx;


architecture top_level of lab1_gx is
begin

--Lab 1a: Assign all switches to LEDs
  LEDR <= SW;

end architecture top_level;
```

Figure 1:

Perform the following steps to implement a circuit corresponding to the VHDL code in Figure 1 on the DE2 board.

Create a new project:

1. Create a new directory called lab1_gx (at a suitable location, such as your network M- drive user area, which is backed up every day). Replace gx with your lab group number which is assigned to you (i.e. if you are in group 2 choose the name lab1_g2). Inside this directory create another directly called quartus.

2. Start the Quartus II program

3. Create a new Quartus II project (from File -> New Project Wizard) called lab1_gx (replace x with your group number) and choose lab1_gx/quartus as the working directory . The name of the top-level design will automatically take the same name as the project. Keep this name. Press Next (Figure 2).

4. For page 2 of 5 press Next without adding any files. (Design files will be added later).

5. For page 3 of 5, (Figure 3), select the correct FPGA model: Cyclone II EP2C70F896C6 as the target chip. Press Next.

6. For page 4 of 5 you should select Modelsim-Altera under Simulation Tool name and VHDL under Simulation format. Leave the other options as illustrated in Figure 4 and press Next.

7. Page 5 of 5 now shows a summary your settings. Press Finish to create the project.

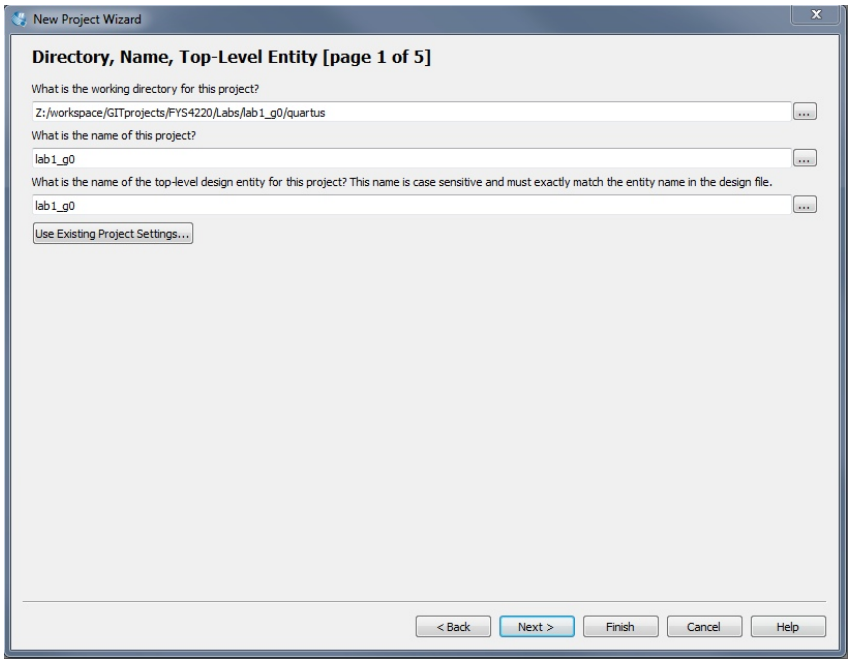

Figure 2:

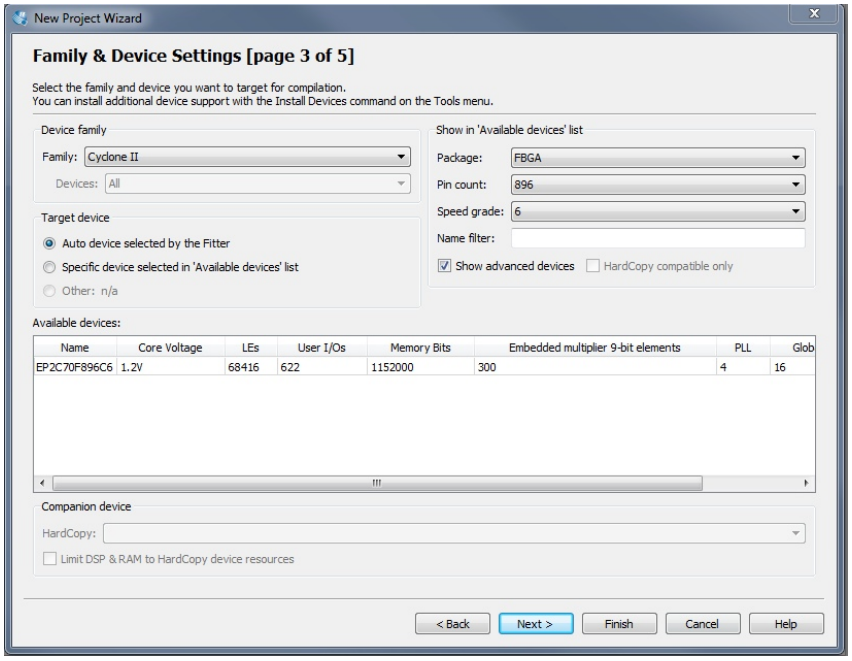

Figure 3:

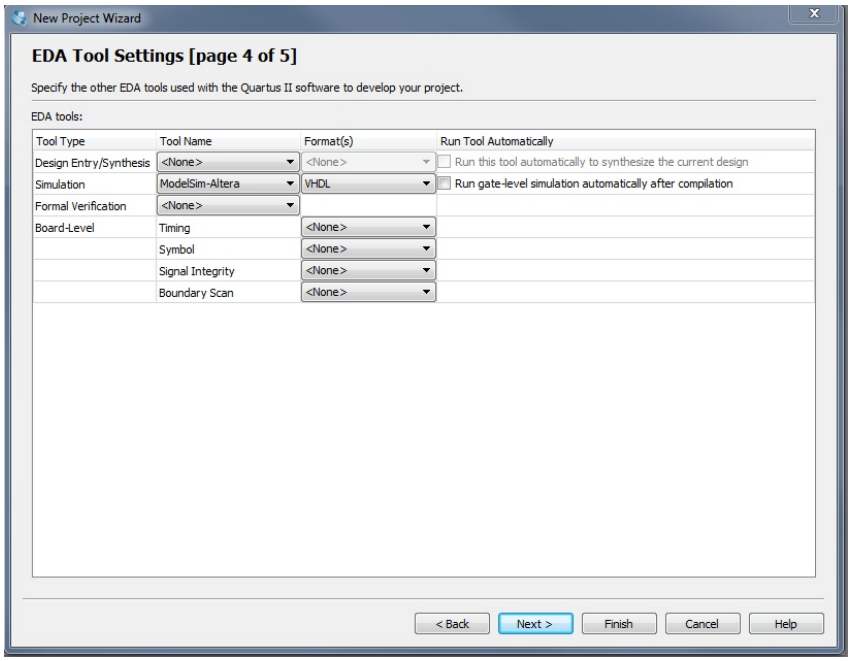

Figure 4:

Create a new VHDL file:

1. Inside the directory lab1_gx create a new directory called vhd_src.

2. Open your favourite text editor (e.g. Notepad++), enter the VHDL description as shown in Figure 1. Save the file in the directory lab1_gx/vhd_src and give it the name lab1_gx.vhd (replace x with your group number).

3. Add the new VHDL file to the project as shown in Figure 5.(Project -> Add/Remove Files in Project).

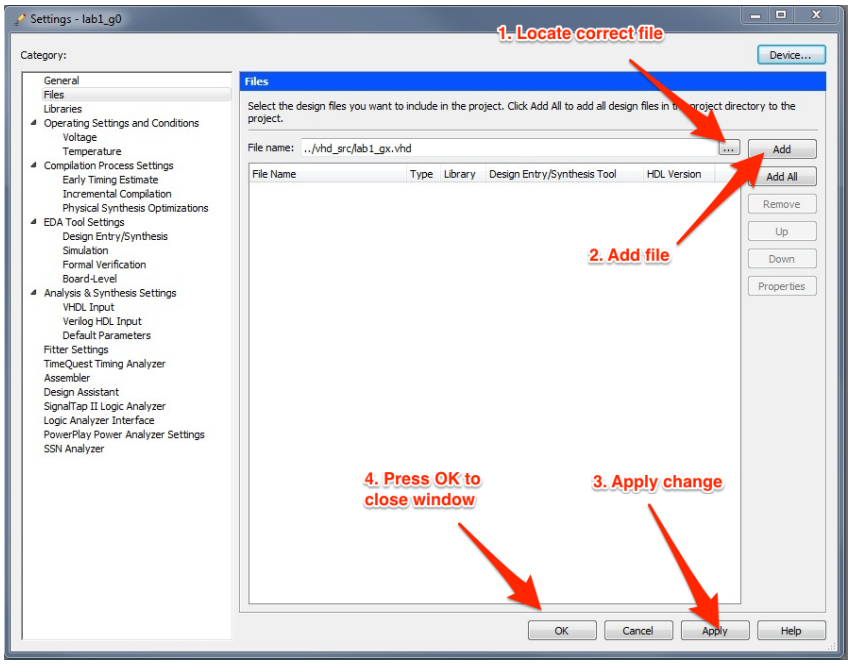

Figure 5:

Create tcl scripting file and make pin assignments:

The DE2-70 board has hardwired connections between its FPGA chip and the switches and LEDs. To use SW17-0 and LEDR17-0 it is necessary to include in your Quartus II project the correct pin assignments, which are given in the DE2-70 User Manual. For example, the manual specifies that SW[0] is connected to the FPGA PIN_AA23 and LEDR[0] is connected to PIN_AJ6. Each pin can be assigned manually through the Quartus II pin assignment manager, however, a more elegant approach would be to make the pin assignment using a tcl scripting file.

Create and import pin settings:

1. Inside the directory lab1_gx create a new directory called constraints.

2. Open your favourite text editor (e.g. Notepad++), enter the pinning constraints as shown in Figure 6. Save the file in the directory lab1_gx/constraints and give it the name pinning.tcl.

3. Add the new tcl file to the project (Project -> Add/Remove Files in Project).

4. Run the tcl script to make the pin assignments (Tools -> Tcl Scripts). The tcl script should be visible under the project folder. Mark the file and click RUN. Verify that the correct assignments have been performed by opening the pin assignment editor (Assignments -> Assignment editor). You should see a similar list as shown in Figure 7.

```
#Toggle switches
set_location_assignment PIN_AA23 -to SW[0]
set_location_assignment PIN_AB26 -to SW[1]
set_location_assignment PIN_AB25 -to SW[2]
set_location_assignment PIN_AC27 -to SW[3]
set_location_assignment PIN_AC26 -to SW[4]
set_location_assignment PIN_AC24 -to SW[5]
set_location_assignment PIN_AC23 -to SW[6]
set_location_assignment PIN_AD25 -to SW[7]
set_location_assignment PIN_AD24 -to SW[8]
set_location_assignment PIN_AE27 -to SW[9]
set_location_assignment PIN_W5 -to SW[10]
set_location_assignment PIN_V10 -to SW[11]
set_location_assignment PIN_U9 -to SW[12]
set_location_assignment PIN_T9 -to SW[13]
set_location_assignment PIN_L5 -to SW[14]
set_location_assignment PIN_L4 -to SW[15]
set_location_assignment PIN_L7 -to SW[16]
set_location_assignment PIN_L8 -to SW[17]
#LED outputs
#Red LEDs
set_location_assignment PIN_AJ6 -to LEDR[0]
set_location_assignment PIN_AK5 -to LEDR[1]
set_location_assignment PIN_AJ5 -to LEDR[2]
set_location_assignment PIN_AJ4 -to LEDR[3]
set_location_assignment PIN_AK3 -to LEDR[4]
set_location_assignment PIN_AH4 -to LEDR[5]
set_location_assignment PIN_AJ3 -to LEDR[6]
set_location_assignment PIN_AJ2 -to LEDR[7]
set_location_assignment PIN_AH3 -to LEDR[8]
set_location_assignment PIN_AD14 -to LEDR[9]
set_location_assignment PIN_AC13 -to LEDR[10]
set_location_assignment PIN_AB13 -to LEDR[11]
set_location_assignment PIN_AC12 -to LEDR[12]
set_location_assignment PIN_AB12 -to LEDR[13]
set_location_assignment PIN_AC11 -to LEDR[14]
set_location_assignment PIN_AD9 -to LEDR[15]
set_location_assignment PIN_AD8 -to LEDR[16]
set_location_assignment PIN_AJ7 -to LEDR[17]

#To avoid that the FPGA is driving an unintended value on pins that are not in use:
set_global_assignment -name RESERVE_ALL_UNUSED_PINS "AS INPUT TRI-STATED"
#Pin AD25 is default assigned for two purposes and must be set to regular IO after configuration.
set_global_assignment -name CYCLONEII_RESERVE_NCEO_AFTER_CONFIGURATION "USE AS REGULAR IO"
```

Figure 6:

Compile the project and program the FPGA:

1. Compile the project: (Processing -> Start Compilation)

2. During compilation of the project some warnings about missing .sdc file, "No clocks defined in design" and "output pins without output pin load capacitance assign-

	tatu	From	To	Assignment Name	Value	Enabled
1	✔		SW[0]	Location	PIN_AA23	Yes
2	✔		SW[1]	Location	PIN_AB26	Yes
3	✔		SW[2]	Location	PIN_AB25	Yes
4	✔		SW[3]	Location	PIN_AC27	Yes
5	✔		SW[4]	Location	PIN_AC26	Yes
6	✔		SW[5]	Location	PIN_AC24	Yes
7	✔		SW[6]	Location	PIN_AC23	Yes
8	✔		SW[7]	Location	PIN_AD25	Yes
9	✔		SW[8]	Location	PIN_AD24	Yes
10	✔		SW[9]	Location	PIN_AE27	Yes
11	✔		SW[10]	Location	PIN_W5	Yes
12	✔		SW[11]	Location	PIN_V10	Yes
13	✔		SW[12]	Location	PIN_U9	Yes
14	✔		SW[13]	Location	PIN_T9	Yes
15	✔		SW[14]	Location	PIN_L5	Yes
16	✔		SW[15]	Location	PIN_L4	Yes
17	✔		SW[16]	Location	PIN_L7	Yes
18	✔		SW[17]	Location	PIN_L8	Yes
19	✔		LEDR[0]	Location	PIN_AJ6	Yes
20	✔		LEDR[1]	Location	PIN_AK5	Yes
21	✔		LEDR[2]	Location	PIN_AJ5	Yes
22	✔		LEDR[3]	Location	PIN_AJ4	Yes
23	✔		LEDR[4]	Location	PIN_AK3	Yes
24	✔		LEDR[5]	Location	PIN_AH4	Yes
25	✔		LEDR[6]	Location	PIN_AJ3	Yes
26	✔		LEDR[7]	Location	PIN_AJ2	Yes
27	✔		LEDR[8]	Location	PIN_AH3	Yes
28	✔		LEDR[9]	Location	PIN_AD14	Yes
29	✔		LEDR[10]	Location	PIN_AC13	Yes
30	✔		LEDR[11]	Location	PIN_AB13	Yes
31	✔		LEDR[12]	Location	PIN_AC12	Yes
32	✔		LEDR[13]	Location	PIN_AB12	Yes
33	✔		LEDR[14]	Location	PIN_AC11	Yes
34	✔		LEDR[15]	Location	PIN_AD9	Yes
35	✔		LEDR[16]	Location	PIN_AD8	Yes
36	✔		LEDR[17]	Location	PIN_AJ7	Yes
37		<<new>>	<<new>>	<<new>>		

Figure 7:

ment" will be display in the message window. These warnings can for the moment be ignored.

3. If the compilation of the project was successful, Quartus has generated and SRAM object file (.sof) either in the project directory itself or in the directory output-files under the project directory. This is the programming file that will be downloaded to the FPGA. Make sure to connect the USB cable to the USB connector on the DE2-70 board marked with BLASTER, and the switch to the left of the LCD panel is set in the position marked RUN. Connect the power cable and turn on the power by pressing the red button.

4. Open the Quartus programmer (Tools -> Programmer).

5. If the field next to the button "Hardware Setup" shows "No Hardware", make sure the USB cable is connect to both the PC and the DE2-70 board, and that the power is turned on. Press the "Hardware Setup and choose "USB-blaser" under the "Currently selected hardware". Press Close.

6. If the device does not show in the main window as illustrated in Figure 8, press Auto-Detect.

7. If the programming file does not show in the Programmer window, double click the area indicated in Figure 8. and select the correct programming file as shown in Figure 9.

8. To program the FPGA press Start.

9. Verify that your design works by changing the position of the toggle switches on the DE2-70 board.

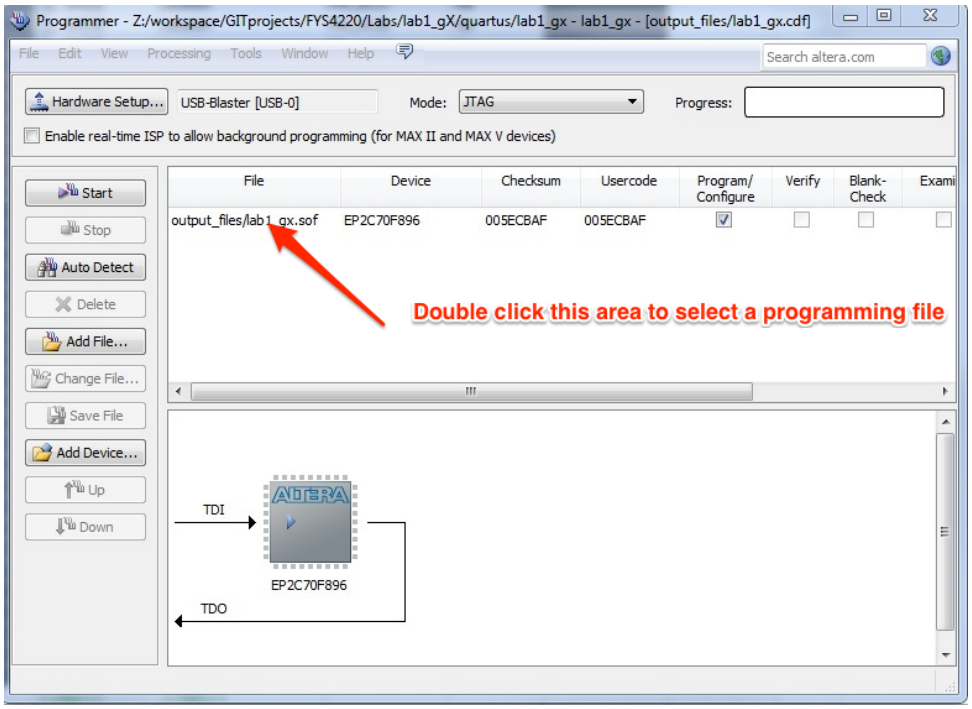

Figure 8:

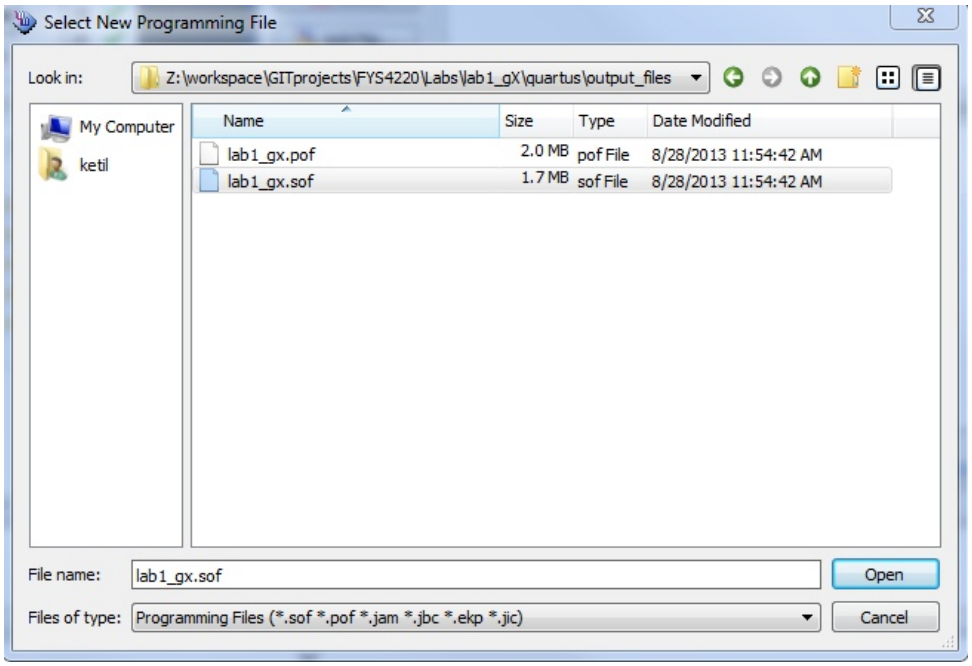

Figure 9:

1 b) 7-segment decoder with functional and timing simulation

In this part you will design a 7-segment decoder using the VHDL selected signal assignment (with-select). The decoder is activated by a transaction on the input ports SW(3 downto 0) and drives seven the outputs HEX0(6 downto 0) that are used to display a character on a 7-segment display. The seven segments in the display are identified by the indices 0 to 6. Each segment is illuminated by driving it to the logic value 0.

Perform the following steps:

1. The 7-segment decoder shall be added as a statement in the lab1_gx.vhd design file and implemented using selected signal assignment (with-select).

2. You also need to modify the entity description by adding an additional output port vector (7 bits wide) using the identifier name HEX0.

3. The four inputs of the decoder will be controlled by the toggle switches SW(3 downto 0).

4. In order to make sure that the new output ports will be connected to the correct pins, you need to add the pin assignment listed in Figure 10 to the pinning.tcl file. After saving the files run the tcl script (Tools -> Tcl Script). Again, verify the the correct pin assignments have been performed in the pinning assignment editor.

5. Before downloading the new design to the FPGA we want to verify that the design is working as expected. For this purpose we will be using the test bench design shown in Figure 11 . Create a new file called tb_lab1_gX.vhd in the directory vhd_src, add the test bench design, and save the file (as before, replace x with your group number, both in the filename and in relevant places of the design file).

6. To include this test-bench file into the project open the Settings window (Assignments -> Settings), navigate to EDA Tools Settings and Simulation. Select Compile test bench and press the button Test Benches (see Figure 12). This will open a new window (see Figure 13). Press the button New and fill in the fields according to Figure 14. When filling in the test-bench name the next field will automatically be filled out with the same value. Select the test bench file tb_lab1_gx.vhd, press ADD, and then OK.

7. Before running the simulation analyse the design to verify correct syntax etc (Processing -> Start -> Start Analysis & Elaboration). If successfully completed run the functional simulation (Tools -> Run Simulation Tool -> RTL Simulation). This will open Modelsim and automatically open the wave window and run the simulation. If the simulation tool does not start and you get an error message, verify that the quartus software has the correct path to the simulation tool as shown in Figure 15 (Tools -> Options). You should see a similar result as shown in Figure 16.

8. In order to run a timing simulation you first need to run a full compilation of the project (Processing -> Start Compilation).

9. Run the timing simulation (Tools -> Run Simulation Tool -> Gate level simulation). Select the "Slow model". If a Modelsim window is still open, close Modelsim before running the timing simulation.

10. What is the result of the timing simulation? Is there a difference compared to the functional simulation? Can you explain this result?

11. Increase the time between the transitions on SW(3 downto 0) by changing all the "wait for 5 ns" statements to "wait for 20 ns" in the tb_lab1_gx.vhd file. Close the Modelsim window and restart the timing simulation. Does this change the outcome of the timing simulation?

12. In the Modelsim wave window, use cursors to measure the time it takes before a transaction on the SW(3 downto 0) results in a stable signal on the HEX0. E.g. for the transaction "0000" -> "0001" on SW(3 downto 0) as shown in Figure 17. Can you explain this delay?

```
#Seven segment display
set_location_assignment PIN_AE8 -to HEX0[0]
set_location_assignment PIN_AF9 -to HEX0[1]
set_location_assignment PIN_AH9 -to HEX0[2]
set_location_assignment PIN_AD10 -to HEX0[3]
set_location_assignment PIN_AF10 -to HEX0[4]
set_location_assignment PIN_AD11 -to HEX0[5]
set_location_assignment PIN_AD12 -to HEX0[6]
set_location_assignment PIN_AF12 -to HEX0_DP
set_location_assignment PIN_AG13 -to HEX1[0]
set_location_assignment PIN_AE16 -to HEX1[1]
set_location_assignment PIN_AF16 -to HEX1[2]
set_location_assignment PIN_AG16 -to HEX1[3]
set_location_assignment PIN_AE17 -to HEX1[4]
set_location_assignment PIN_AF17 -to HEX1[5]
set_location_assignment PIN_AD17 -to HEX1[6]
set_location_assignment PIN_AC17 -to HEX1_DP
```

Figure 10:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity tb_lab1_gX is
end;

architecture testbench of tb_lab1_gX is

  signal LEDR : std_logic_vector(17 downto 0);
  signal SW   : std_logic_vector(17 downto 0);
  signal HEX0 : std_logic_vector(6 downto 0);

  component lab1_gX is
    port (
      LEDR : out std_logic_vector(17 downto 0);
      SW   : in  std_logic_vector(17 downto 0);
      HEX0 : out std_logic_vector(6 downto 0));
  end component lab1_gX;

begin

  UUT : lab1_gX
    port map(
      LEDR => LEDR,
      SW   => SW,
      HEX0 => HEX0
      );

  stimuli_process : process
  begin
    --set '1' as the default values for input SW.
    SW(17 downto 0) <= (others => '0');
    wait for 20 ns;
    --Run through all combinations of input values
    --for SW(3 downto 0) and validate the expected value
    --of HEX0. Chang value every 5 ns.
    SW(3 downto 0)  <= X"0";
    wait for 5 ns;
    SW(3 downto 0)  <= X"1";
    wait for 5 ns;
    SW(3 downto 0)  <= X"2";
    wait for 5 ns;
    SW(3 downto 0)  <= X"3";
    wait for 5 ns;
    SW(3 downto 0)  <= X"4";
    wait for 5 ns;
    SW(3 downto 0)  <= X"5";
    wait for 5 ns;
    SW(3 downto 0)  <= X"6";
    wait for 5 ns;
    SW(3 downto 0)  <= X"7";
    wait for 5 ns;
    SW(3 downto 0)  <= X"8";
    wait for 5 ns;
    SW(3 downto 0)  <= X"9";
    wait for 5 ns;
    SW(3 downto 0)  <= X"A";
    wait for 5 ns;
    SW(3 downto 0)  <= X"B";
    wait for 5 ns;
    SW(3 downto 0)  <= X"C";
    wait for 5 ns;
    SW(3 downto 0)  <= X"D";
    wait for 5 ns;
    SW(3 downto 0)  <= X"E";
    wait for 5 ns;
    SW(3 downto 0)  <= X"F";
     wait;
  end process stimuli_process;
end architecture testbench;
```

Figure 11:

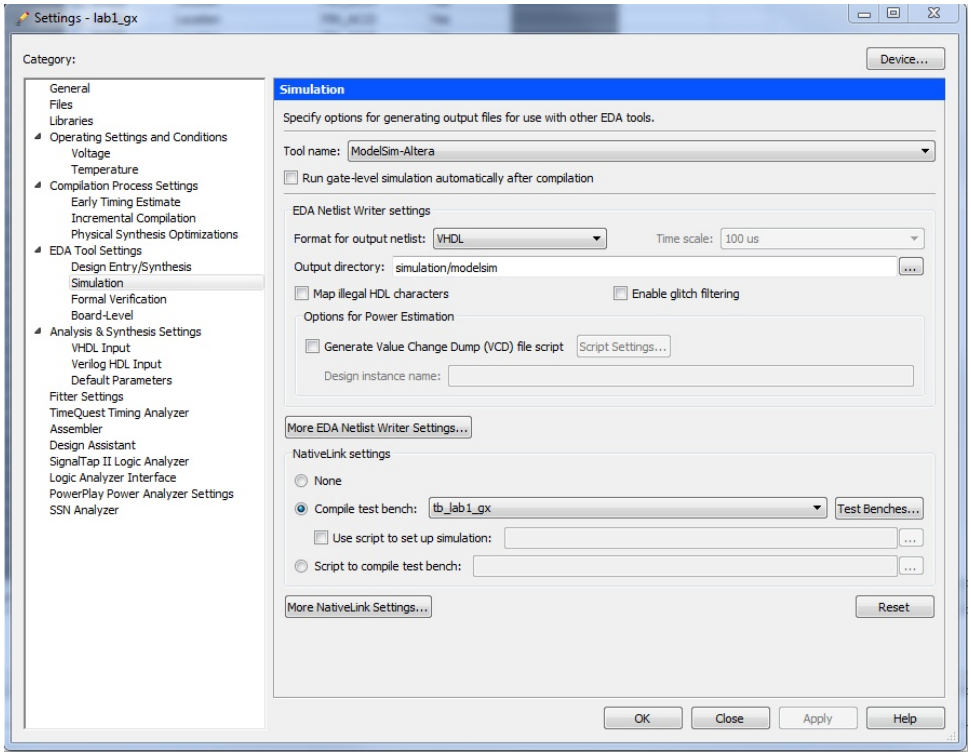

Figure 12:

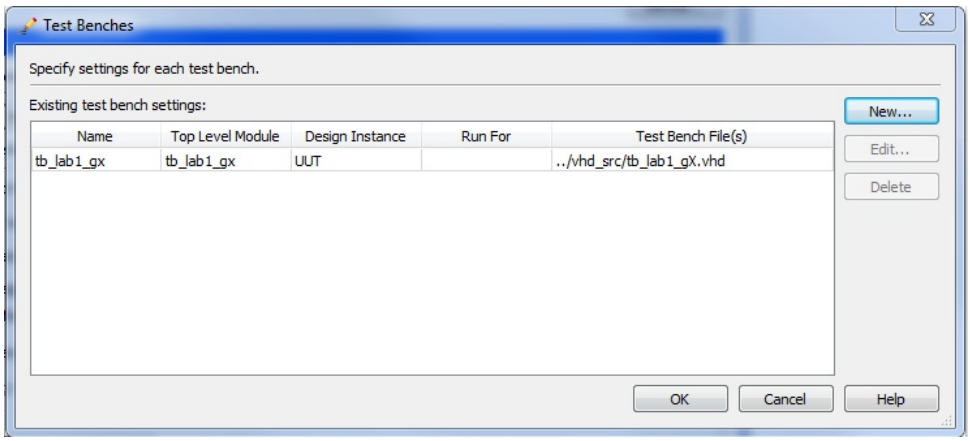

Figure 13:

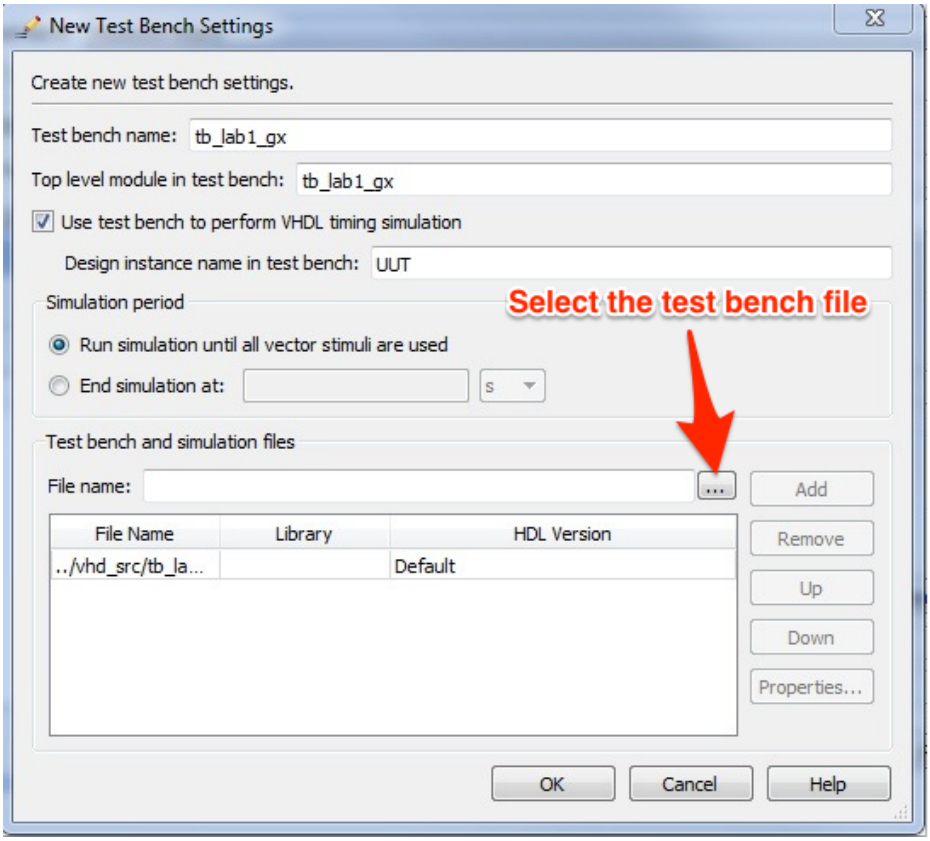

Figure 14:

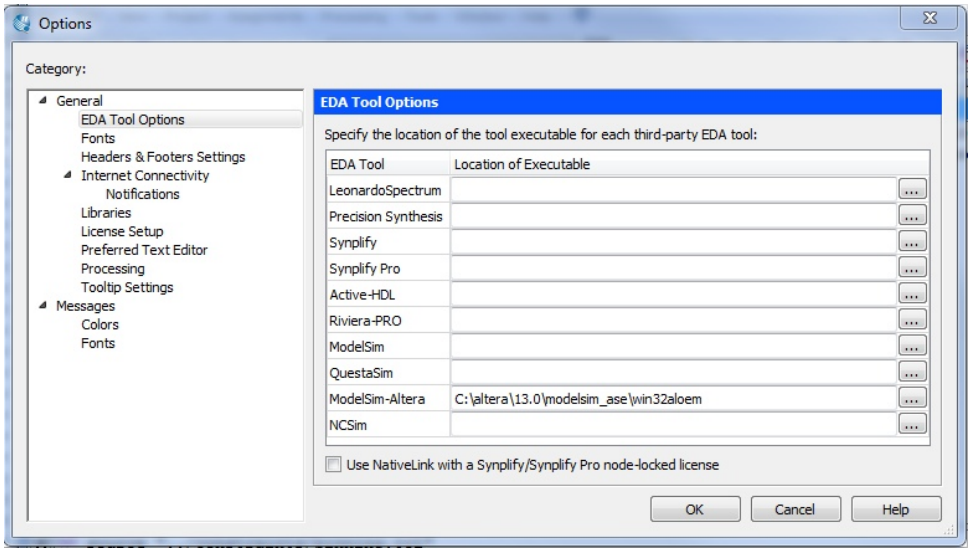

Figure 15:

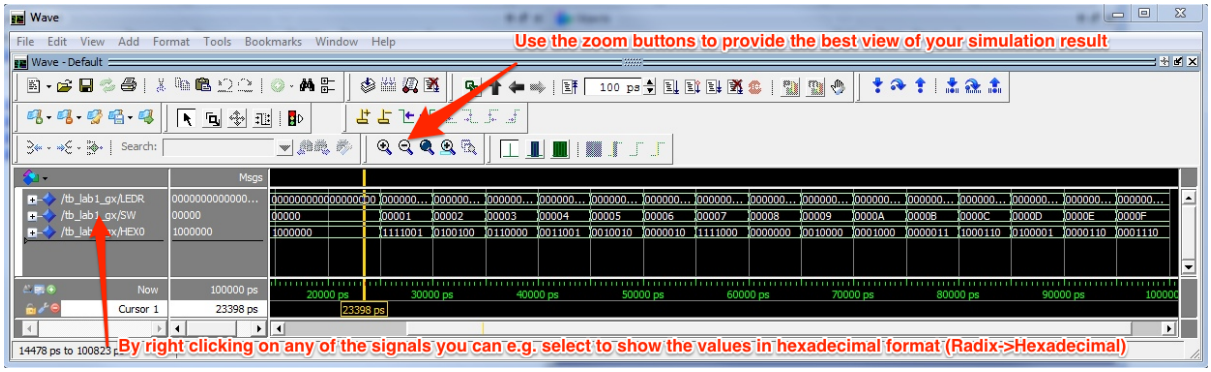

Figure 16:

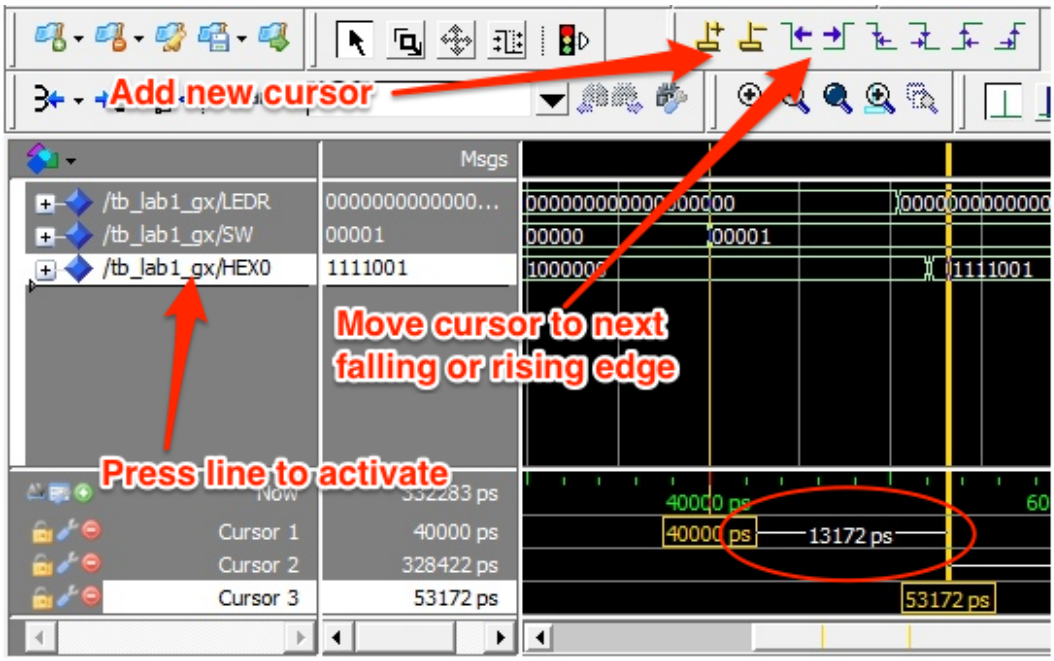

Figure 17:

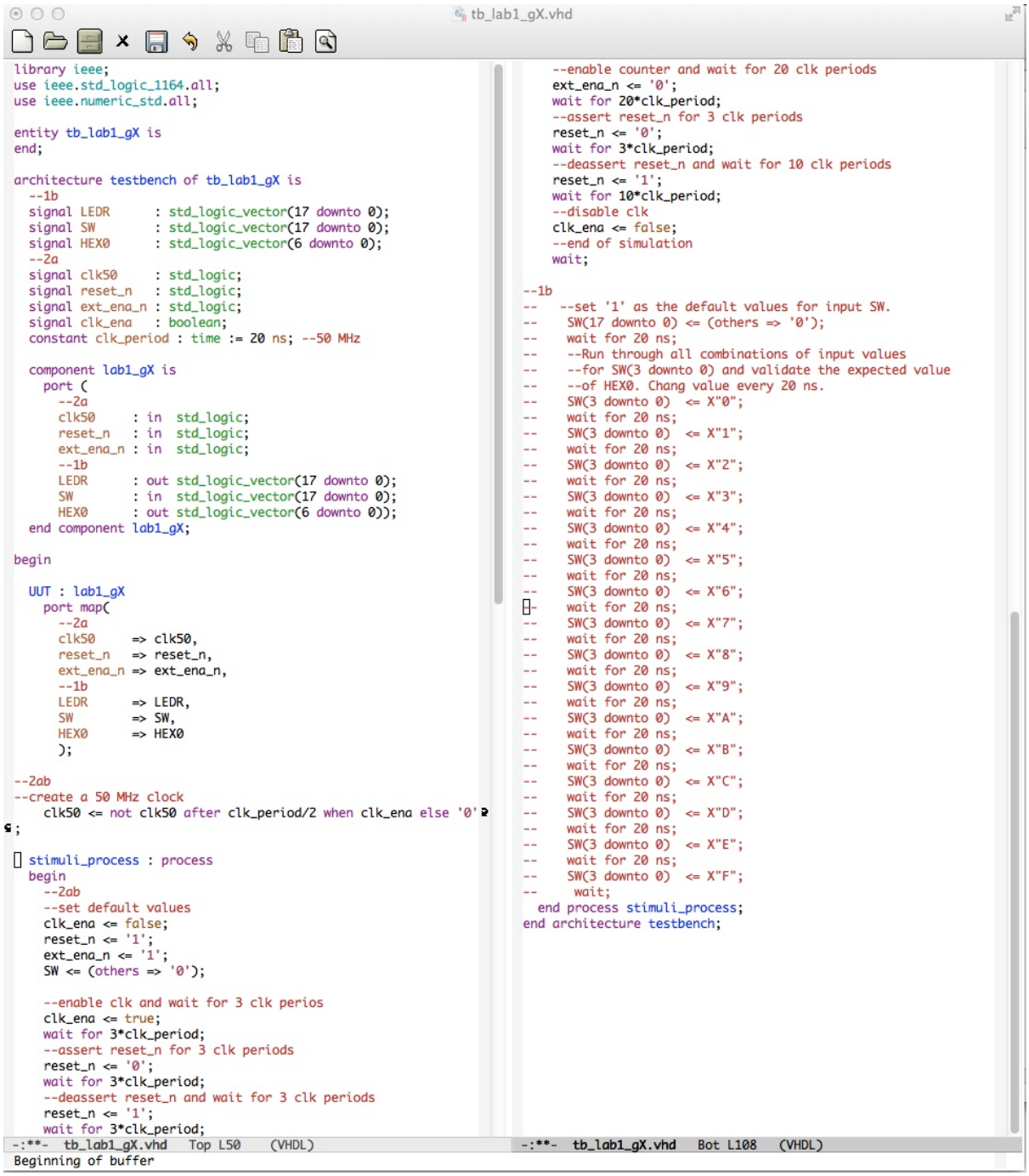

Figure 18:

```
#50MHz clock
set_location_assignment PIN_AD15 -to clk50
#External asynchronous inputs
#Push buttons for external reset and enable
set_location_assignment PIN_T29 -to reset_n
set_location_assignment PIN_T28 -to ext_ena_n
```

Figure 19:

Part 2

2 a) 4-bit counter and push button

In the second part of this lab. exercise you will implement a counter and use one of the push button KEY1 on the DE2-70 card to increase the counter value in steps of one. The counter value will be shown on the 7-segment display. A second push button KEY0 will be used to reset the counter. The reset will be synchronous to the system clock.

Create the 4 bit counter with active low synchronous reset:

1. Modify the entity of lab1_gx and add the following ports:
   - clk50 : in std_logic_vector
   - reset_n : in std_logic_vector
   - ext_ena_n : std_logic
2. Declare the internal signal:
   - signal counter : unsigned(3 downto 0) := "0000";
3. Write a VHDL process for the 4 bit counter. The counter value should be increased in steps of 1 (counter <= counter + 1) on the rising edge of the clk and only when the ext_ena_n = '0'.
4. Include the library numeric_std.all in order to allow for arithmetic operations and the use of the type unsigned.
5. Preform an analysis of the design to check for error (Processing -> Start -> Start Analysis & Elaboration).
6. Modify the test bench file tb_lab1_gx.vhd according to Figure 18. You need to update the component and port map of lab1_gx, add some additional signals, generate a 50 MHz clock, and produce the relevant stimuli to test the counter.
7. Run a functional simulation (Tools -> Run Simulation Tool -> RTL Simulation) and verify that the output HEX0 counts up as expected.
8. Before trying the new design on the DE2-70 card, yo need to first add the correct pinning assignments for the new ports. Add the assignments shown in Figure 19. to the pinning.tcl file and re-run the tcl script. Verify that the new pinging assignments have been performed.

9. Compile the full project.
10. Program the FPGA
11. Press KEY0 to start the counter. What happens? Are you able increment the value of the 7-segment display in steps of 1? If not, can you explain why?

2 b) Edge detector (OPTIONAL)

Implement the edge detector as presented and described in the lecture notes in order to control the single stepping of the 7-segment display.