FYS4220/9220

Lab Exercise 1

VHDL – Switches & LEDs, 7-segment decoder, counters, and digital debouncer

Introduction

The main purpose of this first lab exercise is to learn the design methodology for programmable logic (FPGA/CPLD) using the QUARTUS II and Modelsim-Altera design tools. You will learn the entire process of getting your VHDL program into the FPGA on the printed circuit board. In addition, you will learn how to perform both functional (RTL level) and timing (Gate level) simulations using the Altera tools.

The lab also demonstrates several techniques used in digital design (counting, decoding and synchronization).

An introduction to the QUARTUS II design tool can be found in the pdf document **"Quartus II Introduction Using VHDL Design"**. We will not use the Quartus II simulator described in section 6. This simulator is based on waveforms only (not testbenches), and because of its limitations and the fact that this simulator is no longer available in Quartus II (from version 10.0) we will use the Modelsim-Altera VHDL simulator. The setup and use of the Modelsim-Altera simulator from Quartus II is explained in the note ***"Quartus_Modelsim_setup"***.

For this first lab you should make a main folder called Lab1. In addition, it is recommended to make sub folders called part1, part2, part3, and part4 for the different parts of the lab, in order to only have one Quartus II project in each (sub) folder.

A short lab report must be created (e.g. in MS Word), including the required material (answer to questions, print screens etc.). The required submission (hand-in) for each part of the lab is specified in the text (e.g. your VHDL files).

1. Switches & 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 *SW*17−0 on the DE2 board as inputs to the circuit. We will use light emitting diodes (LEDs) as output devices.

The DE2 board provides 18 toggle switches, called *SW*17−0, which can be used as inputs to a circuit, and 18 red lights, called *LEDR*17−0, that can be used to display output values. Figure 1 shows a simple VHDL program 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. We have used a single assignment statement for all 18 *LEDR* outputs, which is equivalent to the individual assignments:

```
LEDR(17) <= SW(17);
LEDR(16) <= SW(16);
. . .
LEDR(0) <= SW(0);
```

The DE2 board has hardwired connections between its FPGA chip and the switches and lights. To use *SW*17−0 and *LEDR*17−0 it is necessary to include in your Quartus II project the correct pin assignments, which are given in the ***DE2 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*. The procedure for making pin assignments is described in the tutorial ***Quartus II Introduction using VHDL Design***.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

- - Simple module that connects the SW switches to the LEDR lights

ENTITY part1 IS
PORT (
SW     : IN STD_LOGIC_VECTOR (17 DOWNTO 0);   -- switches
LEDR : OUT STD_LOGIC_VECTOR(17 DOWNTO 0) -- red LEDs
);
END part1;

ARCHITECTURE Behavior OF part1 IS
BEGIN
        LEDR <= SW;
END Behavior;
```

Figure 1

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

1. Create a new directory called Lab1 (at a suitable location, such as your network M-drive user area, which is backed up every day).
2. Copy the provided file **part1.vhd** into this directory
3. Open and look at the provided VHDL file called **part1.vhd** in a suitable editor of your choice (using e.g. Quartus II or Notepad++). This is the same VHDL code as shown in Figure 1.
4. Start the Quartus II program, if not already started.
5. Create a new Quartus II project called Lab1_part1 for your circuit (from ***File – New Project Wizard***).

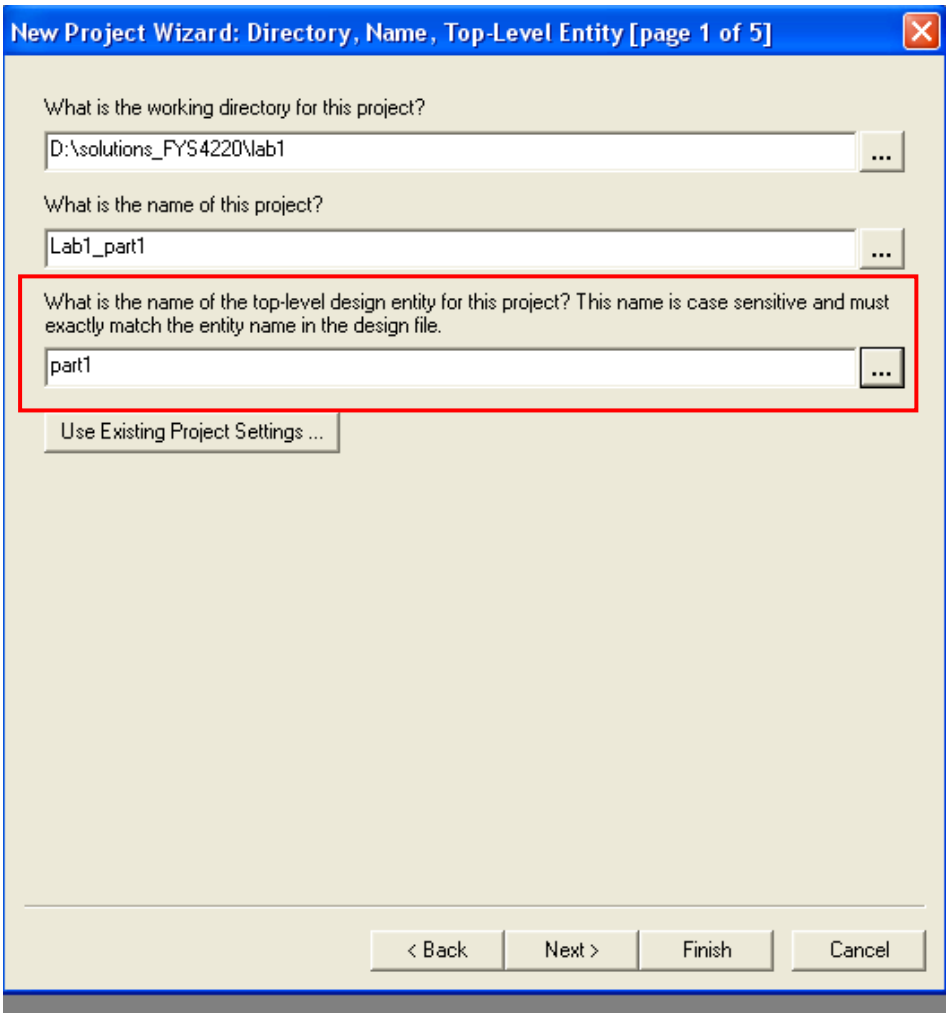

Figure 2

6. Include the VHDL file called **part1.vhd** in your project.

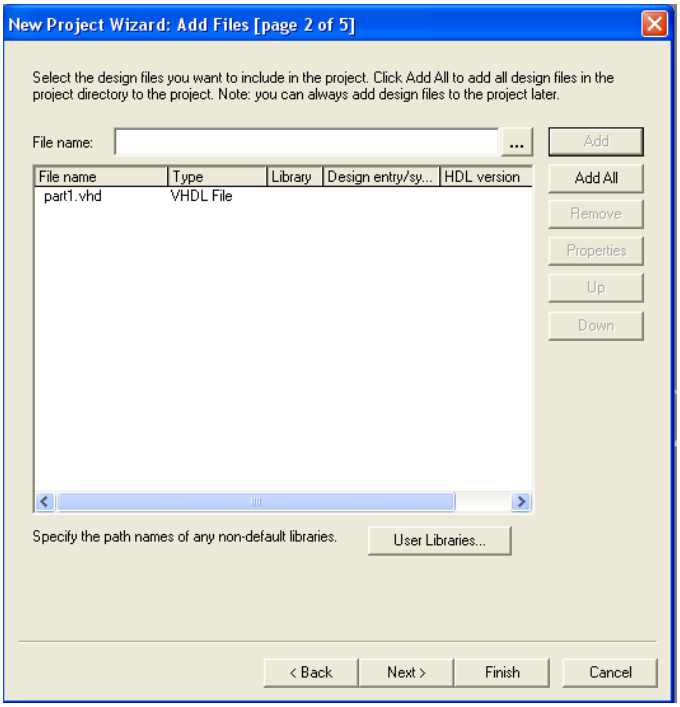

Figure 3

7. Select Cyclone II **EP2C70F896C6** as the target chip, which is the FPGA chip on the Altera DE2 board (hint: set Pin count to 896 to easier find the correct chip).

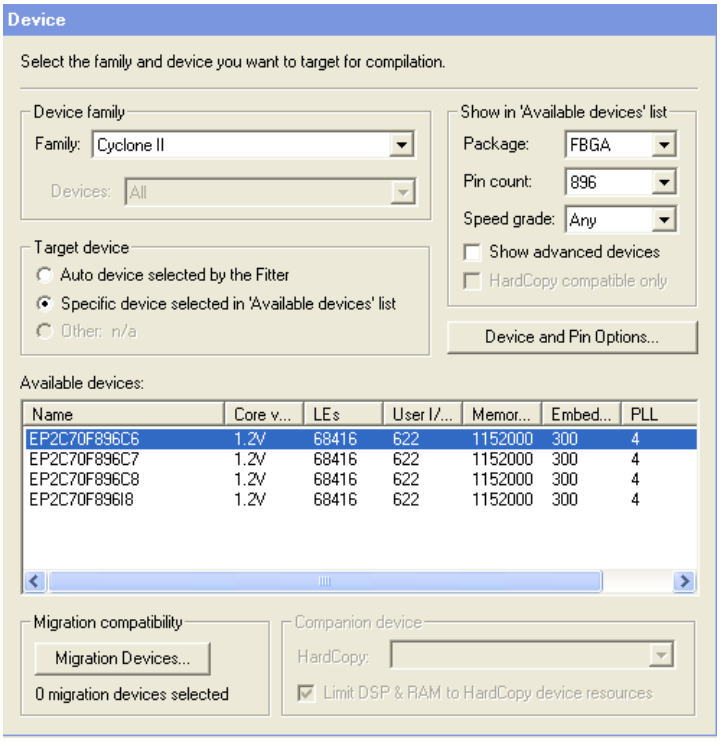

Figure 4

8. We will use the Modelsim-Altera simulator – set the EDA simulation toll to ***Modelsim-Altera***, the others tools (Synthesis and analysis) should be <None>

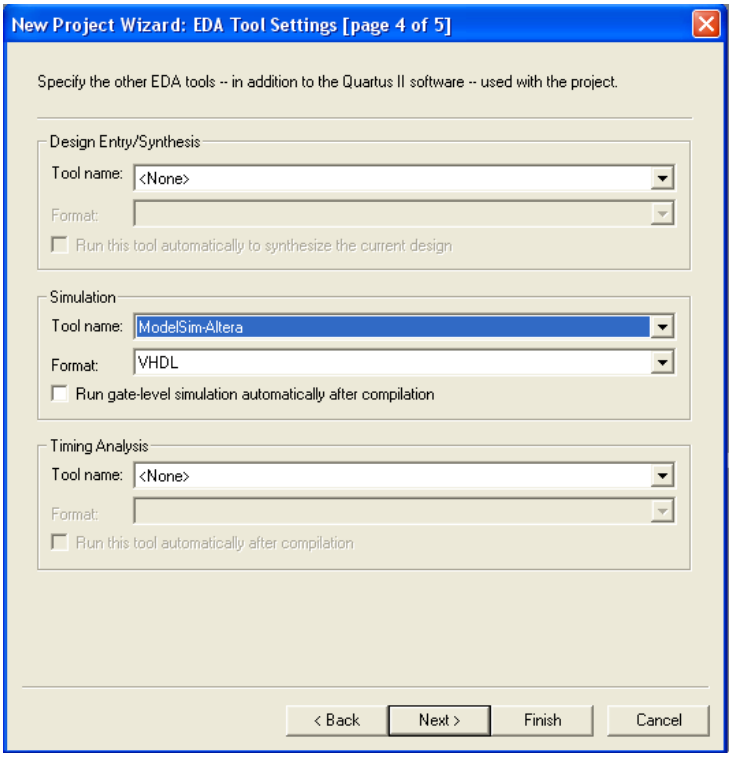

Figure 5

9. Compile the project (select ***Processing - Start Compilation***), and look at the compilation messages. The project must compile be without errors in order to continue.
10. Create a pin assignment (from ***Assignments – Pins***) for the DE2 board, as discussed above, such that the switches control the red LEDs through the FPGA. (In the *Pin Planner*, double clicking a row in the *Location* column gives a pop-list where you can select the pin for the node name/port). You find the correct pins from table 5.1 and 5.2 in the DE2 manual. (Hint: you do not have to do all the 18x2 pin assignments, in order to save some time. If you connect e.g. four switches to the four corresponding red LEDs you will see and learn the process).
11. Compile the project again, in order to include the new pin assignments.
12. Download the compiled circuit into the FPGA chip on the DE2 board. (Programming of the DE2 board is described in the DE2 manual section in 5.1. Remember to use AS – Active Serial programming mode described under the header "*Configuring the EPCS16 in AS Mode"*. Note that you have to disable compression for Cyclone II bitstreams before you compile the design, as explained in "*License setup and device Programming"*. See also the *Quartus II intro* for how to perform the programming from Quartus II); in general the steps can be as follows:
    a. Select ***Tools – Programmer***
    b. In the mode list, select Active Serial Programming
    c. Select ***Add device,*** and select ***EPCS16***
    d. Select the line (click on it with the left mouse button)

e. Select ***Change file***
f. Select **part1.pof**, and click ***Open***
g. Select ***Program/Configure*** and ***Verify***
h. Clik the ***Start*** button to start the programming

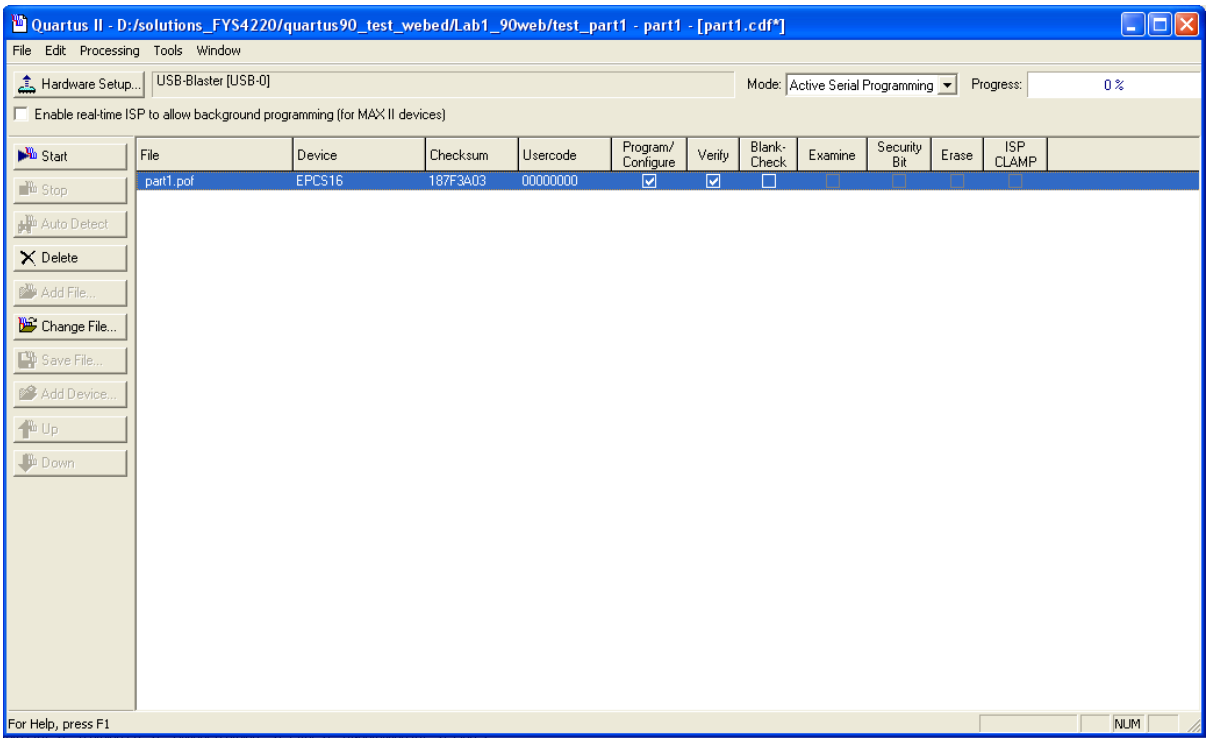

Figure 6

13. Test the functionality of the circuit by toggling the switches and observing the LEDs.

Required hand-in:

- None

2. 7-segment decoder

Figure 1 shows a *7-segment* decoder module that has the three-bit input *C*2 *C*1 *C*0. This decoder produces seven outputs that are used to display a character on a 7-segment display. Table 1 lists the characters that should be displayed for each valuation of *C*2 *C*1 *C*0.

The seven segments in the display are identified by the indices 0 to 6 shown in Figure 7. **Each segment is illuminated by driving it to the logic value 0**. You are to write a VHDL code that implements the logic function that represent circuits needed to activate each of the

seven segments in order to produce the eight characters listed in table 1 (outputs) based on the input from C2, C1 and C0. The VHDL file **seg7decoder.vhd** is created to assist you with the assignment. The entity is already finished, but the architect only contains the code for producing the 0 (zero) character on the 7-segment display when C2 C1 C0 is ”000”. Complete the architecture such that you can active the characters on the display with the C2 C1 C0 inputs according to table 1.

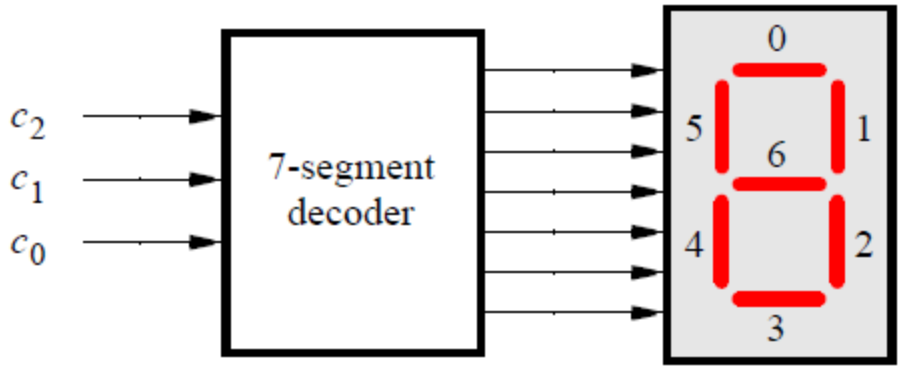

Figure 7 7-segment display decoder

C2 C1 C0 (input)	Display Number (output)
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Table 1.

Note that 4 inputs would be needed to represent numbers from 0 to 9 and hex numbers from A to F; however, this is not required for this assignment.

Perform the following steps:

1. Complete the architecture in the file **seg7decoder.vhd** such that the functionality described in table 1 is implemented.
2. Create a new Quartus II project called Lab1_part2 for your circuit, and add the **seg7decoder.vhd** file to the project.

3. Compile the project, and make sure there are no errors in the code.
4. For pin assignment, connect the *C*2 *C*1 *C*0 inputs to switches *SW*2−0, and connect the seg7 outputs of the decoder to the *HEX0* display on the DE2 board (according to table 5.4 in the DE2 manual)
5. After making the required DE2 board pin assignments, compile the project again.
6. Program the FPGA chip on the DE2 board with your code.
7. Test the functionality of the circuit by toggling the *SW*2−0 switches and observing the 7-segment display (NB: the other HEX seven segments HEX1 to HEX7 will show a static "default" value, since we have not given them any particular input).

Required hand-in:

- Your (completed) **seg7decoder.vhd** file

3 Bit counter and word counter

Telemetry systems used for space applications, such as sounding rockets, often use a telemetry format referred to as a "fixed frame format". An example of this is shown in Figure 8. In this format the different instruments onboard the payload are given dedicated words in the format for transmission of the data to the spacecraft encoder, before transmission back to the ground station. In order for the instrument to transmit the correct data in the correct word a common method applied is to have a bit counter and a word counter in the instrument, such that the instrument knows when to transmit the data. In order to make sure that the counters in the instrument and the counters in the encoder of the spacecraft are synchronized (agrees on the counting values) the counters have a reset input signal in order to reset the counters based on a synchronization signal logic that creates a reset pulse based on a synchronization signal from the encoder (this is not implemented this lab, only the counters and the possibility to reset them using a reset signal).

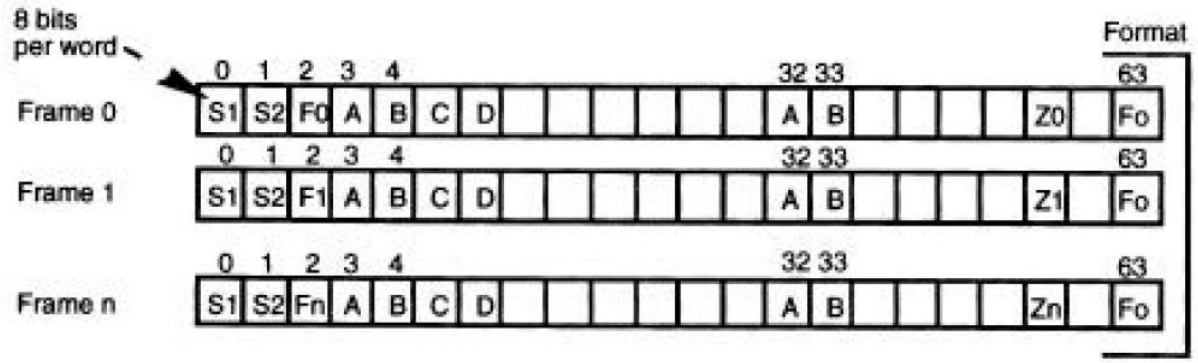

Figure 8 – Telemetry format

The VHDL program **counters_lab1.vhd** takes in a clock signal (clk) and a reset signal. A BITCOUNTER increments a bit counter on every falling clock edge, counting from 0 to 7 and then wraps back to 0 again. In order to count the words a process called WORDSTROBE will set a WordStrobe output high (‘1’) when the bit counter has the value 7, otherwise the WordStobe output is low (‘0’). The WORDCOUNTER looks at the WordStrobe signal, and increases the word counter if the WordStrobe signal is high at a falling clock edge.

The clock signal called clk should be a 1 MHz clock. The reset signal should be low during the entire simulations except for 5 clock periods starting from 5 ms into the simulation. In addition, the counters are to be reset shortly after power up. (This is coded into the testbench file). Simulate the circuit for 10 ms.

Perform the following steps:

1. Create a new Quartus II project called Lab1_part3
2. Add the file **counters_lab1.vhd** file to your project.
3. Open the VHDL file called **counters_lab1.vhd**. Make sure that you understand the VHDL code.
4. Add the testbench file **tb_counters_lab1.vhd** to the project (***Project – Add/Remove Files in project***)
5. Open the testbench file and take a look. The file has been generated by the Quartus II testbench templar writer, and edited to add the wanted stimuli (in the process with the label *stimuli*, in addition to a clock (clk) generation and initial value on the clk signal).
6. Perform a functional (RTL level) simulation using Modelsim-Altera simulator (as explained in the ***“Quartus_Modelsim_setup.pdf”*** note. Simulate the circuit for 10 ms.

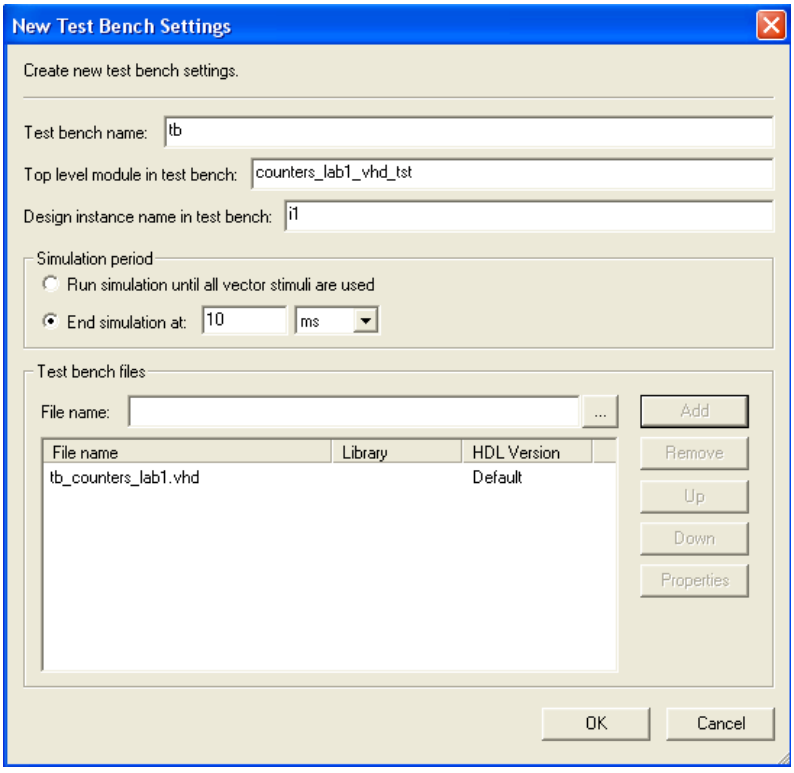

Figure 9

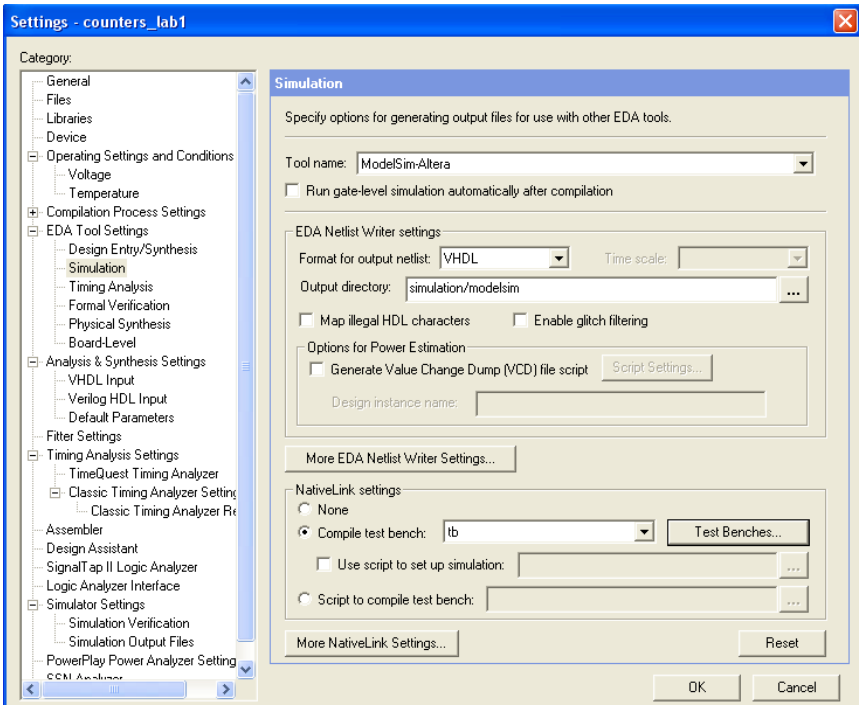

Figure 10

7. Look at the simulation results, and verify correct behaviour of the circuit.
    a. If you click symbol next to the Wave label in the upper left corner of the wave window you can zoom (and unzoom) the wave window, such that it is easier to look at the waveforms.
    b. To change the value of the bit and word counter in the simulations from binary to decimal you can select the signal names, right click and select ***Radix – Unsigned***, se Figure 11.

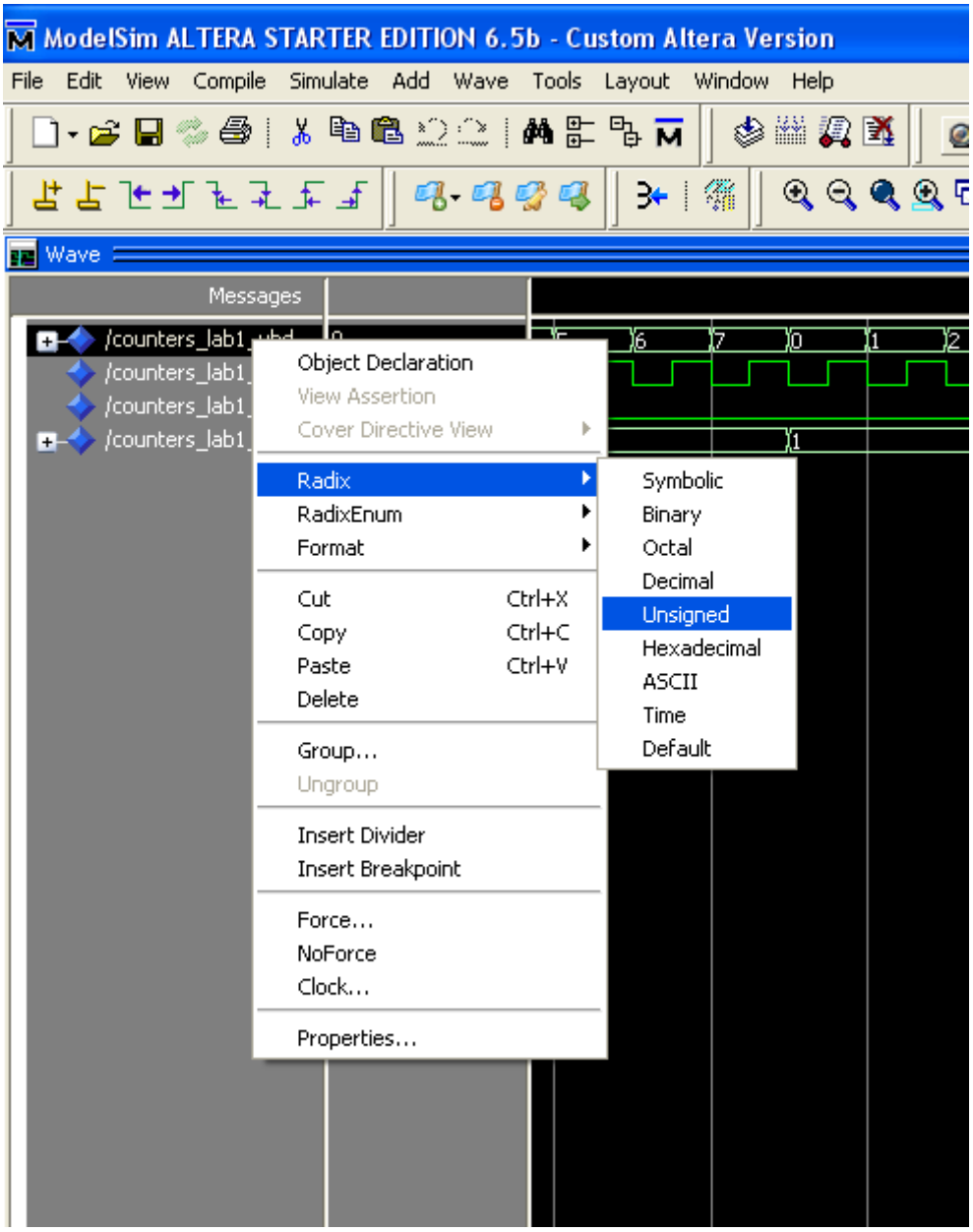

Figure 11

8. Perform a timing (gate level) simulation using the Modelsim-Altera simulator, and compare the result with the functional simulation. (Hint: you should now see delays and also some non-valid counting values just after the clock transition. This should give you a good idea of why asynchronous sequential designs in general are not recommended!).

Required hand-in:

- A print screen of both the functional simulation and the timing simulation (you can paste the result after ***PrntScrn*** into the Paint program, and cut out only the wave part of the simulator window). Adjust the zoom level in the simulator such that it is possible to see that both the bit counter and the word counter are counting .
- Answer the following questions:

- Q1: In the timing (gate level simulation), what is the delay (in ns) between the falling edge of the clock (clk) and the change of the bit counter (bit_cnt) output value to the next valid value? (Hint: Press the ***new cursor*** button (below the ***New File*** icon in the top left corner), such that you get two cursors in the wave window. Then drag one of the cursors to a falling edge of the clock, and place the other at the closest change in bit counter value.

4 Digital debouncer

If you want to input a manual switch signal into a digital circuit you'll need to debounce the signal so a single press doesn't appear like multiple presses. The left-hand image in Figure 12 shows a simple push switch with a pull-up resistor. The right hand image shows the trace at the output terminal, Vout, when the switch is pressed.

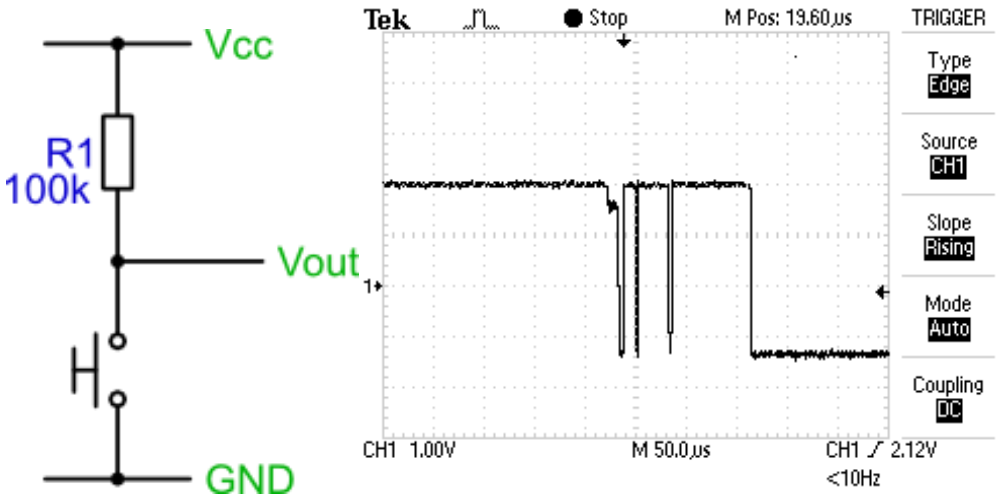

Figure 12: Left: Simple push switch circuit. Right: Output form the switch

As can be seen, pressing the switch does not provide a clean edge. If this signal was used as an input to a digital counter, for example, you'd get multiple counts rather than the expected single count. Note that the same can also occur on the release of a switch. The problem is that the contacts within the switch don't make contact cleanly, but actually slightly 'bounce'

There are many different approaches to cleaning up switch bounce. Below in Figure 13 is a debouncing circuit. The basic idea is to use a capacitor to filter out any quick changes in the switch signal.

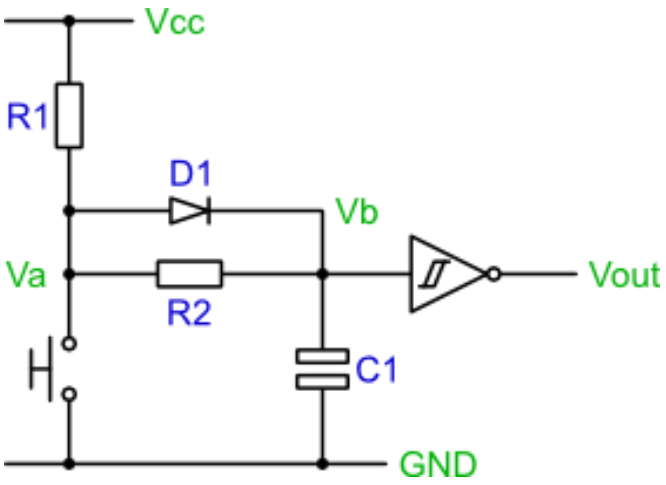

Figure 13 External debouncer citcuit

If no debouncing circuit is available a digital debouncer can be programmed into the FPGA (or CPLD) using VHDL. The digital switch debouncing circuit shown in Figure 14 is a basic switch debouncer.

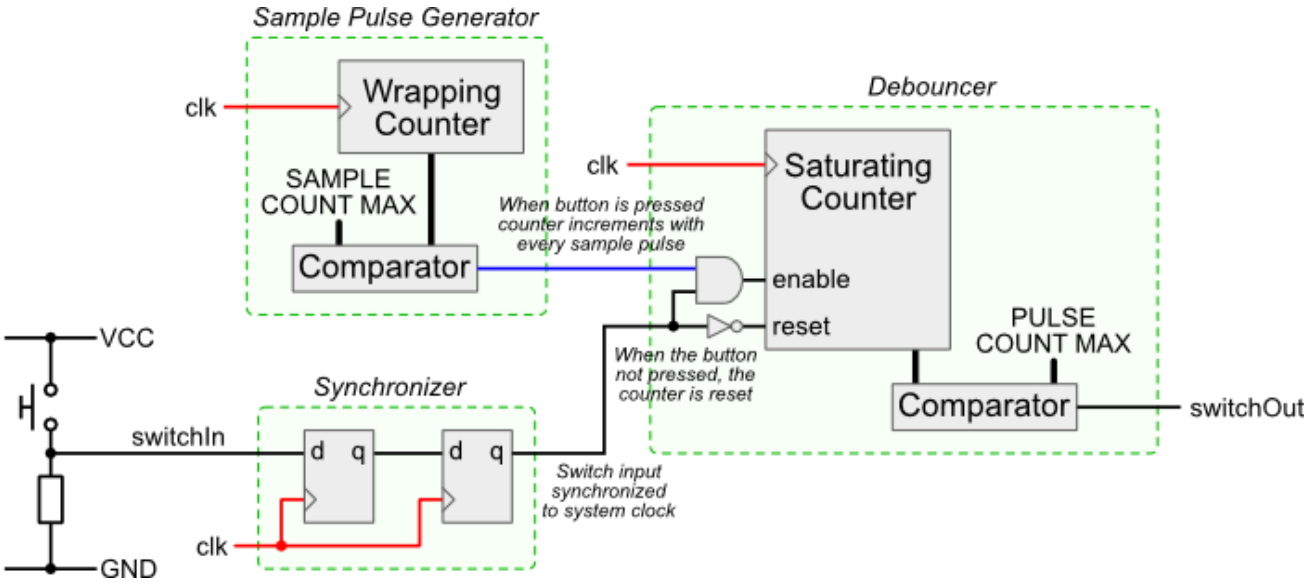

Figure 14 Digital debouncer block diagram

The three main parts are in the block diagram of Figure 14 is:

Sample Pulse Generator - A counter that increments every clock edge. When the maximum is reached a sample pulse is generated and the count value resets to zero. The sample pulse is used to sample the switch input.

Synchronizer - The *Synchronizer* makes sure the switch signal is synchronized with the system clock, the double flip-flop minimizes the chance of metastability.

Debouncer - A counter that is incremented every sample pulse while the switch is pressed. When the switch is not pressed the counter is reset to zero. If the counter is at its maximum, the debouncer output is high, otherwise it is low.

The timing of this circuit needs some consideration. The maximum values of the two counters need to be set depending upon a) the system clock frequency, b) the shortest expected press time and c) the longest bounce time.

The two circuit parameters are set as follows:

SAMPLE COUNT MAX - Set this value to give a sensible switch sampling rate, something around a pulse every 500 us is fine. Higher values mean the pulse counter needs to be bigger, too low and you'll miss short switch presses. Given a system clock frequency of 50 MHz:

SAMPLE COUNT MAX = (Clock Frequency / 2000) – 1 = (50000000 / 2000) - 1 = 24999

PULSE COUNT MAX - Set this high enough that bouncing regions are ignored, but low enough that the shortest button press will still cause the counter to saturate at this value. Given a pulse timing of 500 us a good value is 20, this equates to **10 ms**. Bounces shorter than 10 ms are ignored and switch presses longer than this are detected

In this lab we are assuming that the available system clock that we will use is 1 MHz

Perform the following steps:

1. Create a new Quartus II project called Lab1_part4 for your circuit.
2. Add the debouncer.vhd file to your project.
3. Open the VHDL file called **debouncer.vhd**.
4. Make sure that you understand the VHDL code for the debouncer (compare the VHDL code with the block diagram in Figure 14).
5. Create a VHDL testbench for the debouncer
    a. Create a testbench for the deboncer.vhd file by using the *Testbench template writer* in Quartus II (See ***Quartus_Modelsim_setup.pdf***)
    b. Create suitable stimuli in the test bench to test the debouncer circuit.
    c. The clock signal called clk should be a 1 MHz clock
    d. The reset signal must be high in the beginning (for a few clock periods) in order to reset the counters in the circuit. After this the reset signal can stay low for the rest of the simulation time.
    e. To test the circuit the input signal (SwitchIn) should have some high periods that are shorter than the required high period (10 ms), and then one period that is longer such that the debouncer output (SwitchOut) goes high, see Figure 15 for an example.

Figure 15

6. Perform a functional simulation of the debouncer circuit. Simulate the circuit for 100 ms.
7. Look at the simulation results, and verify correct behaviour of the circuit.
8. In order to look at other internal signals in the circuit do the following:

a. In the Modelsim-altera **sim**-window, mark the unit under test which is called **i1** in this case
b. Select other signals to watch, e.g *sample_cnt* and *debounce_cnt*, and add them to the wave; see Figure 16.
c. Type **restart** in the command (Transcript window), to restart the simulation
d. Click **OK** on the box that opens.
e. To simulate again with the added signals type **run 100 ms (**in the Transcript window), and press enter.
f. Look at the counting values.

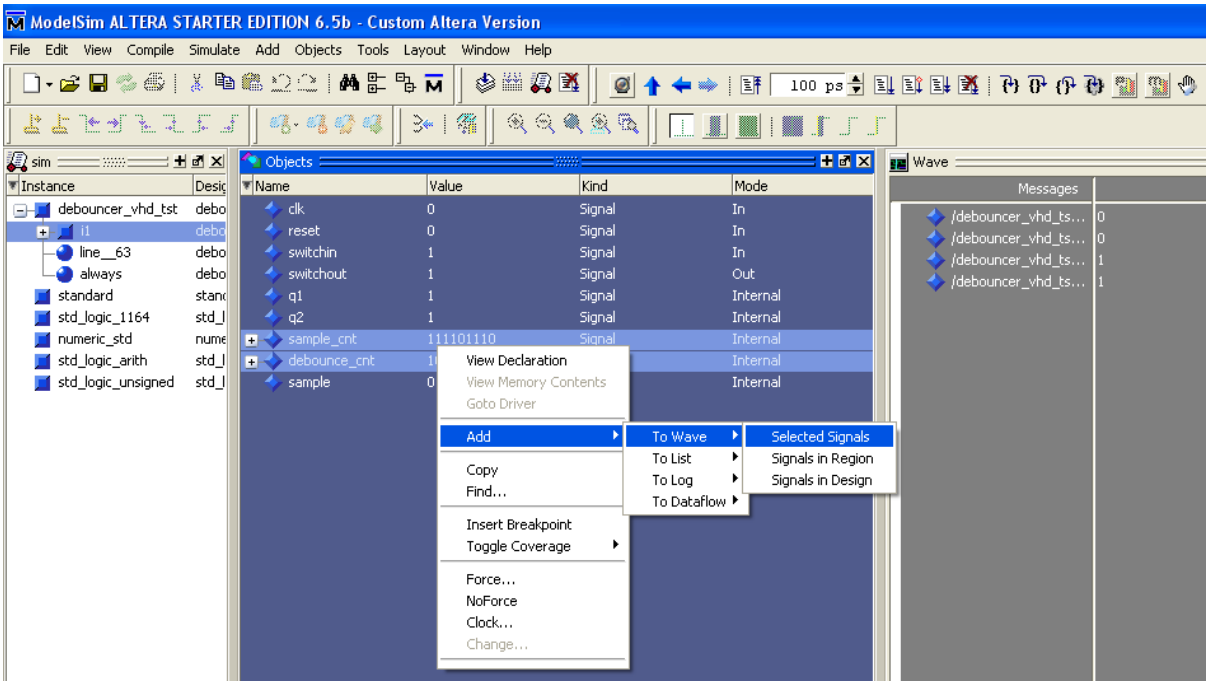

Figure 16

Required hand-in:

- Print screen of the functional simulation result. Adjust the zoom level such that it is possible to see that the debouncer is working properly.
- Your testbench file **tb_debouncer.vhd**