FYS4220/9220

Lab Exercise 2

VHDL – Use of components and packages, LMPs, State machines (Vending machine)

Introduction

The purpose of the first part of this second lab exercise is to learn to include packages, components and Altera Megafunctions (IPs)/LPMs (Library of Parameterized modules) into the VHDL design. In the second part of the lab you are going to implement a state machine in VHDL given a state diagram.

How to use LPMs is described in the pdf file **"Using Library Modules in VHDL Designs**".

You should also read the text "**Debugging of VHDL Hardware Designs on Altera's DE2 Boards**".

NOTE:

For this second lab you should make a main folder called **Lab2**. In addition, it is highly recommended to make sub folders called part1_1, part1_2, part2 etc for the different parts of the lab, in order to only have one Quartus II project in each (sub) folder. It is possible to add/remove files to a project and to change the top level entity file. **However, just changing the top level entity file in an already defined project can cause problems (more setup is required)**. The safest procedure is to make a new folder, copy files to this folder, and start the New Project Wizard to create a new project.

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 lab text (e.g. your VHDL files). In addition to including the *.vhd files you should also paste your VHDL code into your lab report (in order to get comments on your VHDL code from the teaching assistant).

1. Use of components, packages and LMPs

1.1 Perform the following steps:

1. Create a folder (preferably on your M:\ disk) called *Lab2* and a sub folder called *part1_1*, and use it for your design files in this part of the lab.
2. Write a 3-bit counter in VHDL(entity and architecture) based on the symbol in Figure 1. The VHDL file should be named **counter_lab2.vhd** (remember that the entity must have the same name as the file). The reset should be implemented to be a synchronous reset (to the Clock signal), and the counter should increment on the rising Clock edge. The Enable input must be high ('1') in order for the counter to count (increment the C_out value).

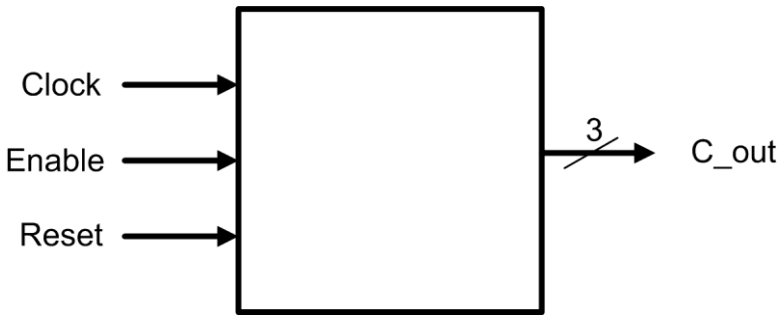


Figure 1 3-bit counter

3. Create a new Quartus II project called **Lab2_part1_1** for your circuit. Select Cyclone II EP2C70F896C6 as the target chip, which is the FPGA chip on the Altera DE2 board. Add your file **counter_lab2.vhd** to the project

4. Write a testbench for the counter; name the testbench **tb_counter_lab2.vhd**.
    a. Use a 1 MHz clock in the simulation of the counter
    b. Remember to reset the counter in the start of the simulation, in order to set the *C_out* value to "000" (important for functional simulations). Before the reset the output will be undefined ('X') in the functional simulation.
5. Perform a ***functional (RTL level) simulation*** of the counter using Modelsim-Altera, to verify correct behaviour, and make a Print Screen of the simulation.

1.2 Perform the following steps:

1. To avoid problems create a new sub folder/directory under the *Lab2* folder and name it part1_2, see the note in the introduction to the lab.
2. Copy the counter you made in section 1.1 (**counter_lab2.vhd)** and the 7-segment decoder that you made in lab1 (**seg7decoder.vhd**) to this new project folder/directory.
3. Make a new VHDL file with the name **main_lab2.vhd**. This will be the top file of the design. The functionality of the design is described in Figure 2, and the details of the implementation are as follows:
    a. The entity I/O and architecture are described by Figure 2.

b. Include **counter_lab2.vhd** as a component in the top vhdl file (main_lab2.vhd)
c. Include **seg7decoder.vhd** as a component in the top vhdl file.
d. Make sure that the files **main_lab2.vhd, seg7decoder.vhd** and **counter_lab2.vhd** are in the same directory (your project folder).
e. Connect the two components by an internal signal called "C_int" and the components input/output to the top entity I/O ports (by port mapping)

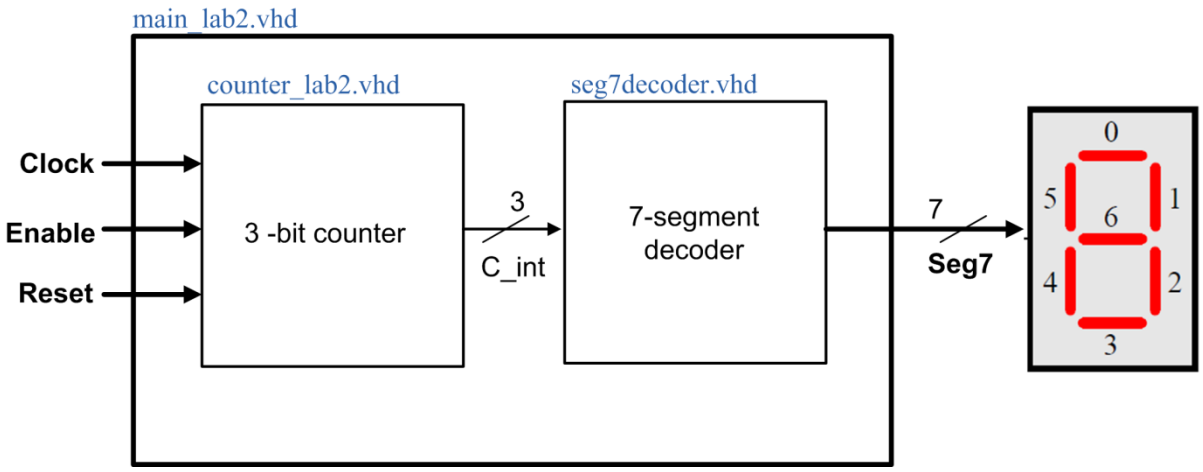


Figure 2 Block diagram

4. Create a new Quartus II project called **Lab2_part1_2**, set the **main_lab2.vhd** as the top-level entity and add the files **main_lab2.vhd**, **counter_lab2.vhd** and **seg7decoder.vhd** to your project (Note: files can also later be added/removed from a project from the menu **Project – Add/Remove files in Project**).
5. Compile the project, and correct any errors.
6. Make the pin assignment (against the DE2 board) to use the pushbutton **KEY0** as the Clock input, switches **SW1** and **SW0** as respectively Enable and Reset inputs, and the 7-segment displays HEX0 to display the counting (from Seg7 output).
7. Compile the project (with the pin assignment)
8. Download (Program) the compiled circuit into the FPGA chip.
9. Test and verify the circuit functionality by operating the implemented switches.

1.3 Perform the following steps:

1. Make a new sub folder called *part1_3* for this part of the lab.
2. Make a new top file with the name **main_lab2_v2.vhd**
3. The circuit design of the main_lab2_v2.vhd should implement the circuit in Figure 3 in the following way:
   a. Make a three-bit wide 2-to-1 multiplexer to enable the selection of two different count values; name the VHDL file **mux_lab2.vhd**
   b. Make a package called **lab2_package.vhd**, and add the components **mux_lab2.vhd, counter_lab2.vhd** and **seg7decoder.vhd** to the package (include their component declaration).
   c. Include the package **lab2_package.vhd (**in the VHDL code in the top file)

d. Use an LPM from the Library of Parameterized modules to implement a 3-bit counter in VHDL (***Tools - MegaWizard Plug-in Manager***).
   i. Select "create new ….."
   ii. Select VHDL output
   iii. Select device family
   iv. Select LPM_counter.
   v. Choose the LPM options to be consistent with the counter_lab2.vhd, i.e. with enable and synchronous reset. Name the counter generated from the LPM as **LPM_cnt.vhd**.

e. Include the different blocks (components) of the design by including the package **lab2_package.vhd** that you made in point 3.b. Note that the LPM_cnt.vhd file (your generated LPM counter) is not part of this package, such that you are to take it into your design as a separate component.

4. Check that all your design files are located in your project directory

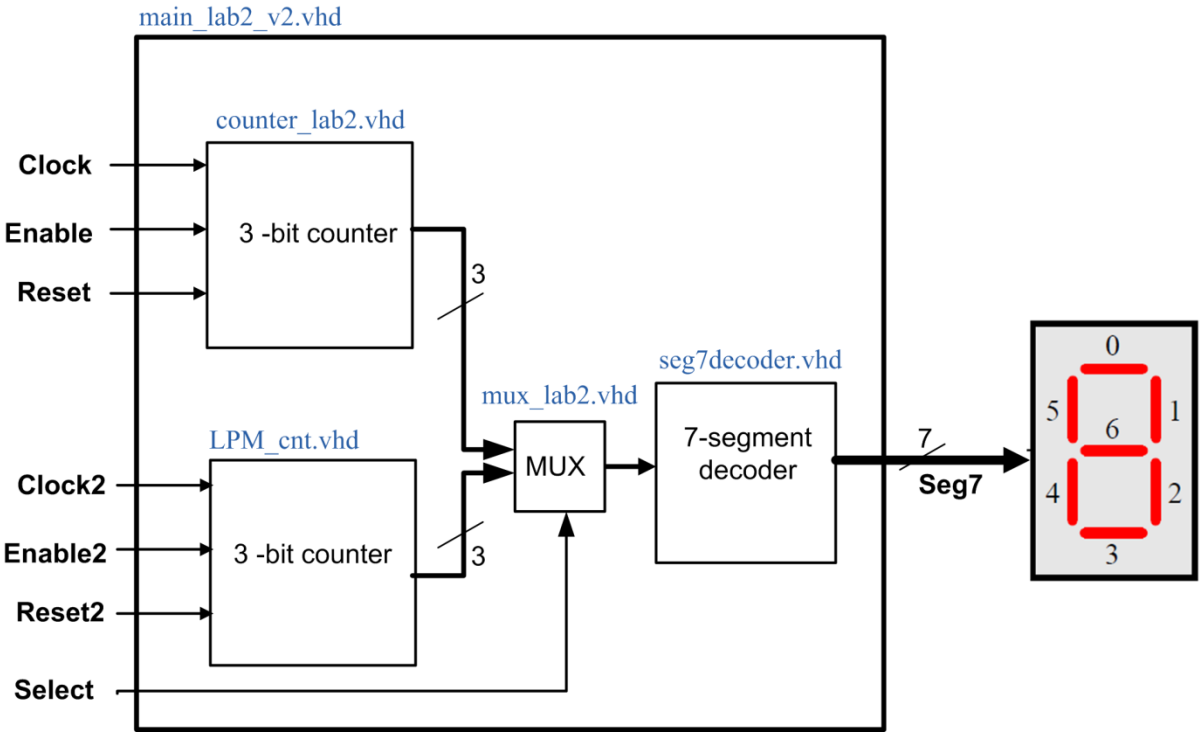
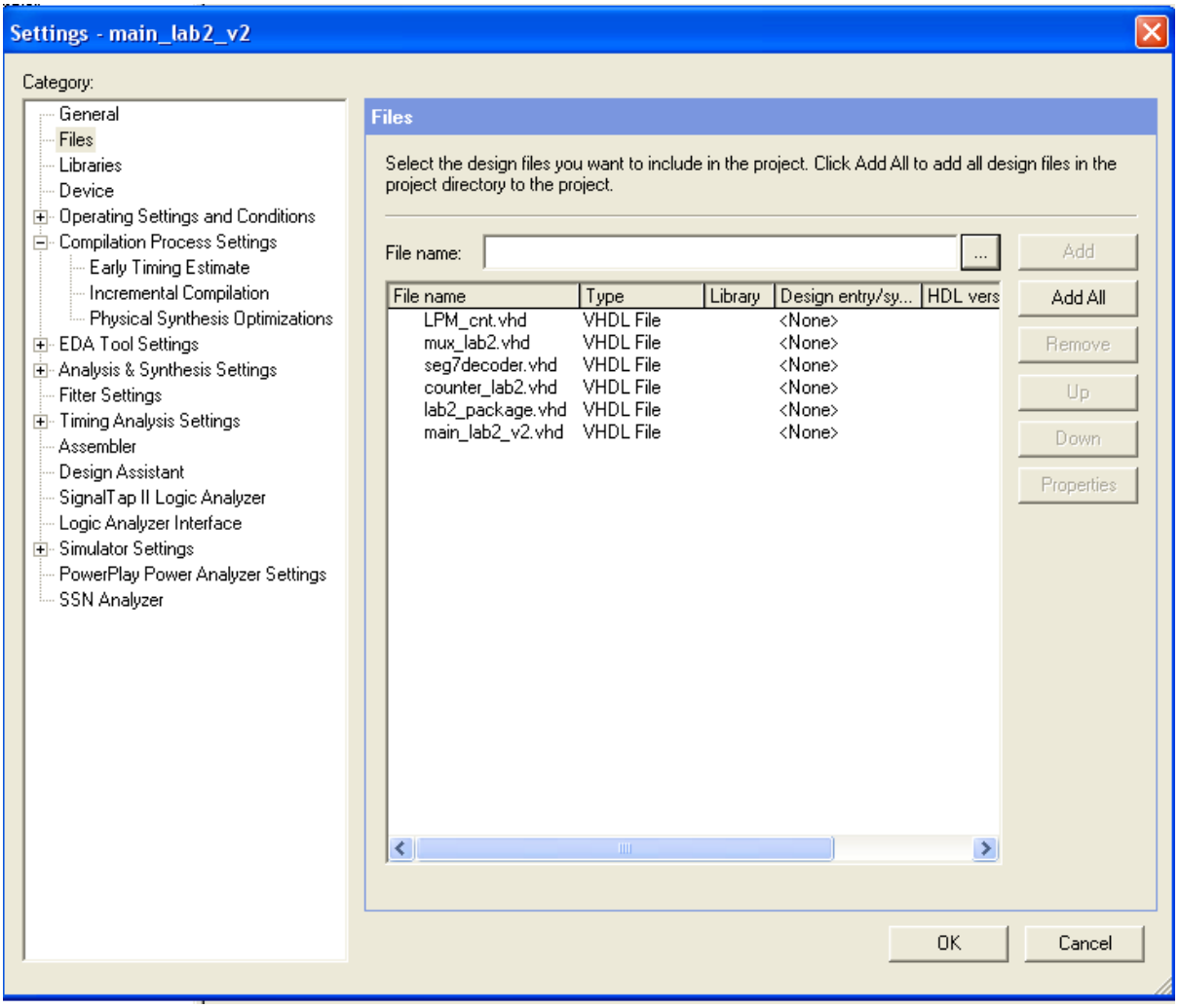


Figure 3 Block diagram of the new circuit

5. Create a new Quartus II project, select **main_lab2_v2.vhd** as the top entity file, and add the necessary files to the project.
6. Remember to include the package **lab2_package.vhd** to your project.



7. Make the pin assignments. Clock, Enable, Reset and Seg7 should have the same pin assignments that you made under part 1.2 of the lab. The new assignments are as follows:
   a. Clock2 : KEY1
   b. Enable2 : SW3
   c. Reset2 : SW2
   d. Select : SW4 (selects which counter to show on the seven segment display)
8. Compile the project (for the FPGA on the DE2 board).
9. Download (Program) the compiled circuit into the FPGA chip.
10. Verify the operation of the circuit programmed into the FPGA on the DE2 board

Required hand-in for part 1(1.1 – 1.3):

- counter_lab2.vhd
- tb_counter_lab2.vhd
- A Print Screen of the counter_lab2.vhd simulation
- main_lab2.vhd
- mux_lab2.vhd
- lab2_package.vhd

- LPM_cnt.vhd
- main_lab2_v2.vhd
- Answer to the following questions:
    - Q1: Did you in any way verify the different components you made (like the mux and the LPM counter) before you used the components in your top (main) design?
    - Q2: Did you discover any errors in your different components when you tested the main designs programmed into the FPGA on the DE2 board?
    - Q3: What would be the recommended design strategy for this design and all other designs, regarding what and when to simulate?

2. Vending machine

A possible simplified implementation of a Coke (Pepsi) vending machine is shown in the state diagram in Figure 4. The cost of a Coke is assumed to be 50 cents, and the machine accepts quarter (25 cents), dime (10 cents) and nickel (5 cents). When the machine has received 50 cents or more it goes to the state called "Vend", and it returns a Coke (Pepsi). If the machine has received exactly 50 cents it then goes from the "Vend" state to the "Exit" state. If the machine has received more than 50 cents it goes to the "Change" state before the "Exit" state, in order to give the buyer back the change above 50 cents. The user can at any time before he has given the machine 50 cents push the "Change" button (which set the *ChangeRequest* signal high) in order to cancel the order and request his money back.

The state machine has the following inputs and outputs:

Inputs:

- Clock
- Nickel
- Dime
- Quarter
- ChangeRequest

Outputs:

- ReturnDime
- ReturnNickel
- DropCoke

It is assumed that this machine only has Coke (Pepsi), and when the user insert the coins into the machine it recognises the coins and set then signals Nickel, Dime or Quarter high, respectively. These inputs are used by the control state machine. (When 50 cents is received the machine drops the Coke without any other user interaction). In a similar way it is

assumed that in order to drop the Coke the output signal "DropCoke" is set to a high value, and to return the money the correct "Return signals" are set to logic high values. If e.g. the user adds one quarter and 3 dimes he has given the machine 55 cents, and the machine should set the "ReturnNickel" output high. Note that it would be possible to give the machine 70 cent; 25 + 10 + 10 + 25 = 70. Then 20 cent should be returned. However, we are assuming that this is not going to happen, such that it is sufficient to give back only one of each coin. We are also assuming that the user is not adding more than 15 cents before he change his mind and presse the Change button to get his money back from the machine. If 15 cent should be returned both the *ReturnDime* and the *ReturnNickel* outputs should be set high ('1'). The outputs should be set according to Table 1 when the state machine is in the "**Change"** state.

Table 1: Outputs in the **Change** state

Cent	ReturnNickel	ReturnDime
5	1	0
10	0	1
15	1	1
55	1	0
60	0	1
65	1	1

It is assumed that it is sufficient to set the signals high for one clock period for all the output signals. **This corresponds to e.g. setting the *ReturnDime* output signal high in the *Change* state, and then go to the *Exit* state and set all output signals to low**. **In the *Initialize* state all outputs should be set to a logic low ('0') value.**

Optional:

In order to make a more correct machine you could loop around in the Change state and give back one and one coin until the correct amount of money is returned. This would also take care of the 70 cent case. If the *ChangeRequest* signal is high all the money should be returned, and if the amount of money is more than 50 cent the machine should give back the change above 50 cents .

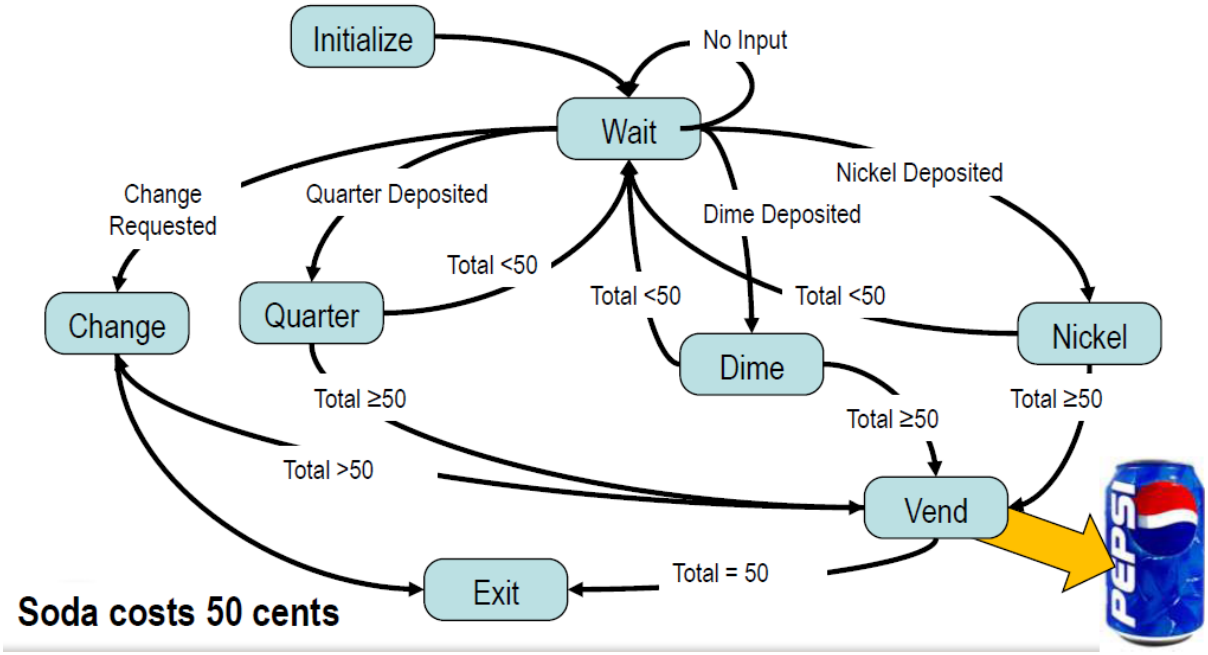


Figure 4: Vending machine state diagram

Perform the following steps:

1. Create a file called **vending_machine.vhd**. The entity of the VHDL program should use the 5 listed input names and the 3 listed output names.
2. Create the ASM chart from the state diagram in Figure 4 (to make the VHDL implementation easier). The ASM chart should have the same number of states (8) and the same state names as given in Figure 4.
3. Write the VHDL code for the state machine (in the architecture part of the VHDL file). Your VHDL code should have the same number of states and the same state names as used in the ASM chart. Remember that the ASM chart is the documentation of your VHDL code for the state machine! Therefore, a change in the VHDL code requires a change in the ASM chart!
4. Create a new Quartus II project called **Lab2_part2**.
5. Add your design file called **vending_machine.vhd** to the project, and check for errors (and correct any errors).
6. Create a testbench to test your state machine, and name the file **tb_vending_machine.vhd**
7. Add the testbench file *tb_vending_machine.vhd* to your project.
8. Compile or Analyze the files in the project (**vending_machine.vhd** and **tb_vending_machine.vhd)**, and correct any errors.
9. Simulate your vending machine (Functional simulation)
10. Verify that the functionality of the vending machine is according to the description in the text and in the state diagram

11. Program the vending machine into the FPGA on the DE2 board (and do changes/additions to the code if needed), and verify the implementation.
    a. Use LEDs to indicate the status of the output signals.
    b. Use a KEY pushbutton switch as clock input (since these are debounced). Note the functionality of the switches from the DE2 manual.
    c. Use the toggle switches (sliders) to set the level of the other inputs (Nickel, Dime, Quarter, ChangeRequest).
    d. (To monitor the present state and the state transitions you could add test outputs to LEDs on the DE2 board, in order to show the state number as a binary number).

Required hand-in:

- Your **vending_machine.vhd** design file
- Your **tb_vending_machine.vhd** testbench file
- A suitable screen shot of the simulated vending machine
- Answer to the following questions:
    - Q1: When this vending machine has returned a Coke and given back any change it goes to an "Exit" state. How is this not usable for a real vending machine, and how to improve the machine by adding new state transitions? (Hint: do you need the "Exit" state?)
    - Q2: Did you do any change to your vending machine (state machine) code to test the code in the DE2 board?