Prof. Dr.-Ing. Dr. h.c. J. Becker

Digital Hardware Design Laboratory (DHL)

Exercise 1- Fundamentals of the Hardware Synthesis with VHDL

Institut für Technik der Informationsverarbeitung, Karlsruher Institut für Technologie (KIT)

1 Introduction

For many participants of this laboratory it will be the first time that they actively use the hardware specification language VHDL. It is important that you recognize from the beginning that VHDL is not a classical programming language. Instead it is a language that allows you to describe a hardware structure, i.e. a circuit consisting of gates and wires. This circuit can then be implemented on a target technology, like standard-cells or on a programmable logic component like a FPGA. As we will see the description can take place on different abstraction levels. For synthesis purpose, it has to be considered that the VHDL syntax and the resulting modeling space is more powerful than what the current synthesis tools as for example XST (Xilinx) or Design compiler (Synopsys) do support at the moment. That means in turn that for synthesis purposes (i.e. for the production of a hardware structure) certain rules must be followed, in order to allow the synthesis tool to produce the desired netlist.

Therefore, a goal of this lab is that the participants learn some basic modeling rules and develop a feeling which resulting netlist will be produced. Thereby, also frequent pitfalls and difficulties are highlighted, which don’t lead to the desired synthesis results.

The exercise is divided into small manageable tasks, so it is possible to concentrate on individual aspects of the hardware description. The sample designs are intentionally very simple, so anyone is able to reconstruct the synthesis result in form of a netlist.

2 Preparation

Prior to the laboratory afternoon for this exercise you should get familiar especially with the following topics:

- Basic concept of hardware description languages
- Fundamental VHDL Elements (Entity, Architecture, Signal, Process, etc.) and VHDL Syntax
    - See also the additional material about VHDL

3 Task Description

During the course of this lab you will develop, analyze and test different hardware units. The Xilinx ZEDBOARD, equipped with a Zynq (XC7Z020) FPGA will be used as the final test platform. More information about the ZEDBOARD is provided separately in User Guides to be found in the supplementary material.

The tool Xilinx Vivado is used as the development environment for the tasks of this lab. Vivado offers a Xilinx specific integrated development environment, in which all steps of the hardware development process are covered under a graphic user interface.

For the individual exercises different Vivado project templates are available. These can be loaded with (File, open Project, ..., xy.xpr). After the project is loaded, in the middle within the „Sources" tab, hierarchically all source files that are assigned to the project are represented. On the left side in the panel "Flow Navigator", all tasks for the highlighted source file are shown. However, the complete tool chain up to the generation of the Configuraiton Bitfile is available only for the top-level module.

Within the "Sources > Libraries" tab all known libraries within the project are shown. By default all design units are assigned to the library "xil_defaultlib". By clicking on the plus sign in front of the "Design Sources", "VHDL" or "library" Tab, the content of the library is shown. The library view can also be used to shift a design unit into a different library (Right-click on the design unit, Set library).

3.1 Exercise 1 – Synthesis of a Multiplexer

In this task the synthesis process of a simple x to 1 multiplexer, i.e. the generation of a gates/macro netlist from a VHDL description, should be analyzed and the VHDL library concept should be understood. First load the project template "VHDL_Ex1.xpr" and select top_ex1 as the top source file (right click on the file, "Set as Top"). Take some time to analyze the project structure. Pay attention into which libraries the design units are mapped after synthesis and how the parameterization of the multiplexer is realized. *What other possibility to parameterize a design do you know?*

For the entity "mux" 4 different architectures are available. They demonstrate different possibilities of how a multiplexer can be modeled in a VHDL behavioral description. Which architecture is used when the component is instantiated is determined by a so called configuration. Within the entity "top" the instantiation of the component "mux" is already prepared for you. First add an inline configuration to the architecture of the top module in order to select the architecture "arch1" for the multiplexer instance and check if the description of the entity top is complete. Then start the synthesis process by clicking on "Run

Synthesis" within the Flow Navigator. If there are errors correct them until the synthesis is performed successfully. If you rerun the Synthesis, make sure to close or reload the synthesized design if it is currently opened. Otherwise the results of the previous synthesis run are displayed.

In the next step open the generated netlist ("View RTL Schematic") and analyze the result. In the netlist editor you can click on an instance to move one step down the hierarchy. Use the pop-button to go one step up the hierarchy.

Does the netlist look like you expected? Look at the synthesis report, in order to get a hint what happened during synthesis.

If not try to figure out which modeling mistake was made. Correct the mistake, restart the synthesis and analyze the result.

Now change the configuration in order to select the architecture "arch2" and start the synthesis. Which modeling mistake was made here and why does the synthesis tool nevertheless produce the correct netlist. Use the synthesis report to get a hint.

Finally, the multiplexer should be implemented on the FPGA and the function should be tested. Therefore, the interface signals of the top module must be connected to the correct pins of the FPGA package. These connections are defined by so called LOC-constraints within the user constraint file "top.xdc". To enter the missing constraints open the constraint file "top.xdc" (Sources → Constraints).

Alternatively, this assignments can also be configured graphically, when you open the "Elaborated Design" in the Flow Navigator and select "Window > I/O Ports" in the menu bar. This will open a new "I/O Ports" view in the bottom, where each top module port can be assigned to a FPGA package pin (in the "Site" column). Changes in this tab need to be saved to the user constraint file using the save button in the toolbar.

The multiplexer should be connected as described in the following table:

output	LED_1
input(0..4)	Push-Button(0..4)
Sel(0..2)	Switch(0..2)

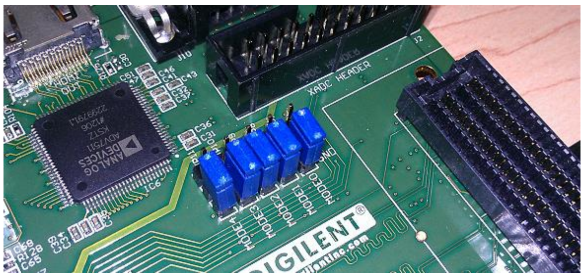
Information of how the LEDs, switches and buttons are connected to the FPGA-pins on the ZEDBOARD is available in the ZEDBOARD Hardware User Guide.

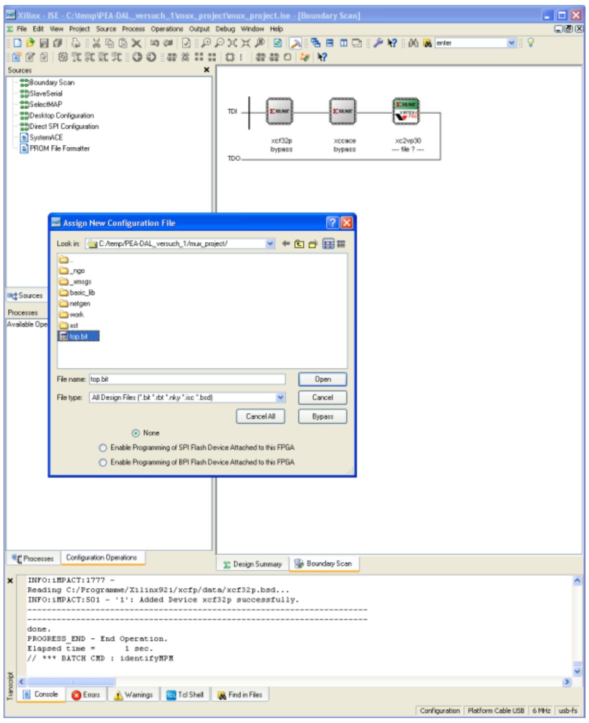
After that start the tool chain to implement the multiplexer design by clicking "Run implementation" and after that "Generate Bitstream". Before trying to connect to the FPGA or initialize the boundary-scan chain, make sure the Mode switches are all set to GND as depicted below.

After that open the Hardware device and program the FPGA.

„Open Hardware Manager" → „Open Target" → „Program Device"

Before you continue check if all steps of the tool chain have been performed successfully. *In which step of the tool chain are the LOC constraints merged into the design?*





Look in: C:/temp/PEA-DAL_versuch_1/mux_project/

- ..
- _ngo
- _xmsgs
- basic_lib
- netgen
- work
- xst
- top.bit

File name: top.bit

File type: All Design Files (*.bit *.rbt *.nky *.isc *.bsd)

[✓ None]

Enable Programming of SPI Flash Device Attached to this FPGA

Enable Programming of BPI Flash Device Attached to this FPGA

```
INFO:iMPACT:1777 -
Reading C:/Programme/Xilinx92i/xcfp/data/xcf32p.bsd...
INFO:iMPACT:501 - '1': Added Device xcf32p successfully.
----------------------------------------------------------------------
----------------------------------------------------------------------
done.
PROGRESS_END - End Operation.
Elapsed time =      1 sec.
// *** BATCH CMD : identifyMPM
```

Figure 1: Mode Switch configuration, Impact dialog

Alternatively you can use the Xilinx iMPACT Tool to program the FPGA as described below.

The FPGA will now be configured with the generated bitstream with “Impact” by using the JTAG-interface. In the opening window you see on the right side the JTAG-scan-chain with two detected devices. We want to leave the first device blank and just want to configure the FPGA. Therefore, choose “bypass” in the “Assign New Configuration File” dialog for the first device. For the FPGA choose the generated bit-file “.bit” file which can be found in the “XYZ.runs/impl_1” directory. Then click on the FPGA and choose “Programm” in the popup dialog to download your bit-file to the FPGA. Finally test the multiplexer design by using the switches and buttons.

3.2 Exercise 2 – Synthesis of Flip-Flops

In the previous multiplexer exercise we regarded the modeling of a pure combinatorial circuit. This means the value at the output of the multiplexer directly depends on the input values. Hence, there is no storing element included. In clock synchronous hardware circuits storing elements are realized through Flip-Flops or Latches. In contrast to Latches, Flip-Flops are clock-edge triggered. This means they transfer the value of a one bit signal at the input to the output at the rising or falling edge of a clock signal. If a vector of values is stored the circuit is referred to as a register. In this exercise we will analyze the synthesis of Flip-Flops. First set “ff_ex2” as top entity (right-click > “Set as Top”).

Look at the architecture of the entity “ff”. *How many Flip-Flops will be synthesized by this description?*

Start the synthesis and confirm your assumption by looking at the synthesis report and the generated netlist.

Next change the architecture so that a Flip-Flop with a synchronous reset will be generated instead. Start the synthesis and check the generated netlist.

3.3 Exercise 3 – Comparison Behavioral Modeling - Structural Modeling

In this exercise we will examine a 1-bit wide ring-shift-register. The shift-register can be modelled using multiplexers and flip-flops. The interface of the ring-shift-register is shown in Figure 2.

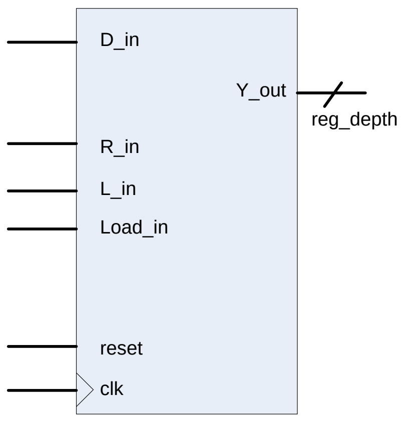


Figure 2: Ring_shift_register

First, select “top_ex3” as top entity in Vivado. For the entity „ring_shift_reg_s“, two architectures are available. The first architecture contains a behavioral model of the ring-shift-register. Analyze the given behavioral description and describe with your own words the properties and the functionality.

Next start the synthesis and examine the generated netlist. *Which components are used to construct the ring-shift-register? Also give their instance names.*

The second architecture is a structural model similar to the generated netlist of the first architecture. The multiplexer of the first exercise and a d-Flip-Flop are used as basic building blocks. The multiplexer is configured as a 3 to 1 multiplexer. By using the VHDL generate construct the structural model is also configurable. Figure 3 shows the structure of the circuit for a register depth of reg_depth=3.

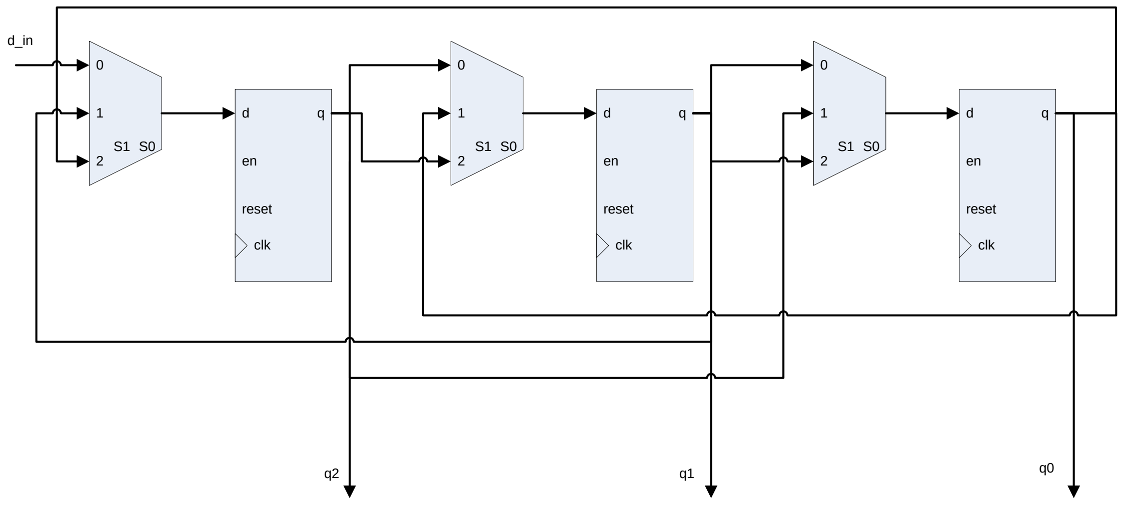


Figure 3: Structure of ring-shift-register, reg_depth = 3

Your task is to add the missing control logic for the multiplexer and the register inputs/outputs. The shift register should allow loading as well as right and left shifting. First draw truth-tables for the missing control signals of the multiplexers and registers dependent on the ports of the interface in Figure 2. In a second step minimize the switching function by using the KV-diagrams provided below in Figure 4.

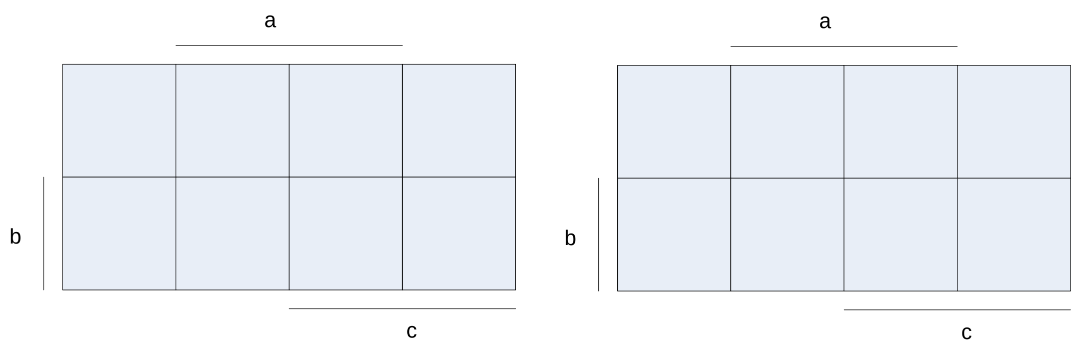


Figure 4: KV-diagram template

3.4 Exercise 4 – Design and Synthesis of FSMs

The structural ring-shift-register model developed from the previous exercise will now be implemented and tested on the FPGA. Therefore, we continue with the previous project.

The outputs of the single shift-register stages should be displayed on 4 LEDs of the ZEDBOARD. The button “BTNC” should be used to load a new value into the register. The value is determined by the button “BTNU”. If the button “BTNU” is pressed while pressing the “BTNC” button a logical 1 will be loaded otherwise a logical 0. The button “BTNR” should be used to right-shift and the button “BTNL” for left-shift respectively. The 100 MHz board-clk should be used as the system-clk. Add the missing constraints in the constraint file.

However, a direct control of the ring-shift-register through the buttons like in exercise 1 is not possible in this case. The following two reasons exist:

1. The buttons on the board are not debounced. This means if a button is pressed or released the resulting signal does not directly settle to its final value. Instead it randomly oscillates for a while like shown in Figure 5. Instead of a single button stroke the sample logic will detect several successive button strokes.

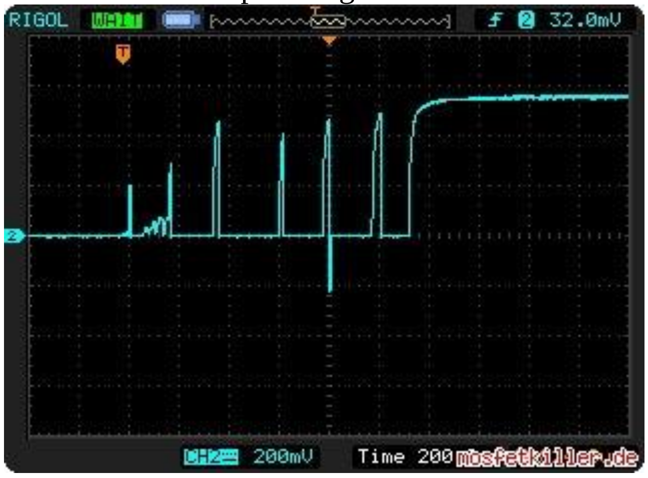


Figure 5: Example of a bouncing button

2. With a button a human operator is not able to produce a pulse that is just one clk-period wide. This is necessary to selectively control the ring-shift-register (load, left-shift, right-shift).

The debounce unit is already provided in the project template. It avoids the coupling of short pulses into the circuit. It was realized by using a shift-register and a control FSM (debounce.vhd).

Now it is your task to realize the missing unit “input_ctrl”. It implements the second requirement that is necessary to selectively control the ring-shift-register.

First add a new VHDL-source file to the project where you can enter the description of the entity and architecture. To model the functionality it is best to describe it as a FSM. Take a look at the given debounce logic (debounce.vhd) if you are not sure how to model a FSM.

Start the synthesis and look at the resulting netlist. *How was the modeling of the FSM transferred into a netlis? Does the number of registers match the expected numbers? Can you reconstruct the logic for the state transitions and the output logic?*

To test the complete design all unit are already instantiated and connected on the top level design file. You can now implement the design and test it on the board. *Is there any difference between the behavioral model of the ring-shift-register and the structural model?*

- Design the Pin-Map of the Buttons/LEDs of the ZEDBOARD und and test the Design using the Board.