1 Prerequisites

For this laboratory you have to checkout the snippets from the corresponding module repository - btf4220-digital. If you did previous exercises only a repository update is requires. Make sure you committed all your local changes, first.

1. Open a terminal (CLI) and change to the *home folder* by executing the command:

```
cd ˜
```

2. Create your working directory or change into it. See procedure in laboratory1 if you need further guidance.

3. Checkout the repository or update it by executing the clone or pull command, respectively:

```
git clone git://pm.ti.bfh.ch/btf4220-digital.git
git pull
```

4. Switch to the directory of this laboratory3 by executing the command:

```
cd ˜/praktika/btf4220-digital/laboratory2
```

IMPORTANT: All the following steps are executed on the server xena unless explicitly noted otherwise. Hence make sure you establish a remote session before performing the steps below.

2 Simple circuit design

In this laboratory we are going to learn the FPGA design-flow as depicted in Figure 1. We step in at the HDL level shown in Figure 1. The circuit we are going to use for this exercise is depicted in Figure 2.

Figure 1: Simplified diagram of a FPGA design-flow.

Assignment 1

In the directory ∼**/praktika/btf4220-digital/laboratory2/vhdl** you will find the example files:

- dffESR-entity.vhdl
- dffESR-behavior-generic.vhdl

The file **dffESR-entity.vhdl** is complete and describes the inputs and output of the circuit. If you modify it use Git to keep track of your modifications.

```
git commit <FILE_TO_COMMIT> -m"<YOUR_COMMIT_MESSAGE>"
```

The file **dffESR-behavior-generic.vhdl** contains the skeleton of the functionality of the circuit depicted in Figure 2. Complete this file by writing the processes (implicit and/or explicit) to realize its functionality.

Figure 2: Block diagram of a D-flipflop with synchronous reset and enable.

3 Simulation

We are going to simulate the VHDL circuit description written in Assignment 1. For this purpose log-in to the server and change to the directory:

```
cd ~/praktika/btf4220-digital/laboratory2/sandbox
```

Before continuing make sure you are in the correct directory! Execute following commands:

```
rm -rf work         # This command will remove the simulation library.
vlib work           # This command will create a new simulation library.
                    #
vcom ../vhdl/dffESR-entity.vhdl            # This command will compile the entity.
vcom ../vhdl/dffESR-behavior-generic.vhdl  # This command will compile your VHDL
                                           # description of the circuit shown in
                                           # Figure above. If you get errors, fix
                                           # them and repeat this step until this
                                           # file compiles without errors.
vsim -t ps dffESR   # This command will start the ModelSim GUI and will start a
                    # simulation ( -t ps sets the simulation mode to ps accuracy).
```

Assignment 2

We are now going to simulate all possible combinations of the inputs to verify the functionality of the block. Start up the "**wave**" window by executing the command shown below in the ModelSim command window.

▶ `add wave -hex *`

A window should pop-up with all inputs and outputs of the circuit. Next we have to provide the simulator with stimuli. For the ports *ClkxCI* and *RstxRI* you can use the commands:

▶ `force ClkxCI 0 0ns, 1 10ns -repeat 20ns`

▶ `force RstxRI 1 0ns , 0 100ns`

Formulate the test pattern for the inputs *DxDI* and *EnaxSI* such that all possibilities are covered.

NOTE: The clock is a 50 MHz clock. For how long x ns do we need to simulate? Perform this simulation by the command:

▶ `run x` ns

4 Automated simulation

For this simple example it is "easy" to type in all commands or to "click" in the GUI; however, in complex designs with a lot of error prone components we require (1) reproducibility and (2) exclusion of errors due to typing and/or clicking mistakes. For this purpose most industrial tools provide us with command-line interface (CLI) that can be used in an automated environment that guarantees the above two items. In this assignment we are going to use such an automated environment to redo the previous simulation.
The automated environment is already partially in place and uses the GNU make and command line utilities. The goal of this assignment is not to understand the automated flow, but to use it.

Assignment 3

On the server change to the directory: ∼**/praktika/btf4220-digital/laboratory2/**.
Execute in the terminal the command **make**. You will see a small explanation of the make targets. Execute in the terminal the command **make clean** to remove all temporary files from the **sandbox** directory.

IMPORTANT: Before executing this command make sure that you copy any file in the **sandbox** directory that you want to keep!

For the automated simulation there are two important files, namely (1) **config/project.files** and (2) **config/project.force**. The first file contains all the names of the vhdl files that are required for the simulation whilst the latter contains the commands required for the simulation. The file **config/project.files** is already complete; however, you need to add the commands used in Assignment 2 to the file **config/project.force**. After having added these commands to this file execute the command **make sim** in the terminal and see that you get automatically the same simulation as performed manually in Assignment 2.

5 FPGA toplevel

Figure 3: Block diagram of the FPGA toplevel given a certain design.

In a FPGA design flow it is good practice to generate a toplevel for each design as shown in Figure 3. This toplevel is used to connect the design to the external pins of the FPGA. Furthermore, the transformation of uni-directional (IN/OUT) into bi-directional (INOUT) is performed and eventual needed tri-state buffers are only inserted at this level.

Assignment 4:

Inspect the architecture of the file **lab2Top-behavior-xilinx.vhdl** on how to use the component declaration.

6 Synthesis

Figure 4: The synthesis step converts the VHDL sources into an optimized netlist.

The first step in the transformation towards a hardware implementation is the synthesis step. For both the FPGA as well as the ASIC flow this step is performed. The synthesis transforms the VHDL code into logical functions, optimizes it to achieve a minimal implementation, and writes out an intermediate netlist that is *target* dependent (e.g., Xilinx FPGA, Altera FPGA, ASIC, etc.). This intermediate netlist is in *Electronic Design Interchange Format (EDIF)*.

One could expect that the synthesis tool only requires the functionality contained in the VHDL files and optimize for the smallest logic function; however, most designs require a predefined timing and the smallest logic function does not guarantee the least area taken. Therefore, the synthesis tool requires besides the functionality contained in the VHDL-files also information about the target platform, e.g., FPGA, ASIC, etc.

As each target requires some specific optimizations there exists several synthesis tools that are optimized for a specific target. Synopsis *Design Compiler* is a synthesis tool optimized for the ASIC target, whilst Synopsys *Synplify Premier DP* is optimized for the various FPGA targets. In the rest of this course we will use the latter tool.

7 Automated synthesis

The synthesizer requires as stated before two important portions of information, namely:

The functionality : This information is provided in the VHDL file(s). For the automation to be able to provide the synthesizer with the names of the VHDL files to use, they are listed in file called **project.files** in the **config** directory. In case of a hierarchical design, the synthesizer also requires to know the toplevel of the design. The name of this toplevel is provided by the file **project.toplevel** in the config directory. NOTE: VHDL does not make a difference between capitals and non capitalized characters; therefore, the toplevel can be specified in either capitals or in a non capitalized version.

The target platform : This information is dependent on the FPGA or CPLD used. The specification of the target platform is dependent on the synthesis tool and is provided in the file **project.device** in the **config** directory.

For this laboratory both files are given. For the rest of the laboratories the file **project.device** will always be used as is, as we use for all the laboratories the same hardware; however, for the other laboratories the files **project.files** and **project.toplevel** need to be adapted accordingly.

Assignment 5:

To start an automated synthesis, execute the command **make synth** in the directory **/praktika/btf4220-digital/laboratory2/**.

TROUBLESHOOTING: It can happen that the synthesis tool does not complete its tasks. To identify the errors/warnings inspect the file <**TOPLEVEL_NAME**>**.srr** in the directory **sandbox/synth/**. This file contains the log file of the synthesis tool. <**TOPLEVEL_NAME**> is the name specified in the file **projects.toplevel** in the **config** directory.

8 Place & Route

Figure 5: The place & route step converts the optimized netlist into a hardware implementation.

The second step in the transformation towards a hardware implementation is the place & route step. This step is highly target dependent and almost every FPGA/ASIC company provides its own tools. For this laboratory we will use the *Xilinx ISE* suite.

To perform the place & route step, the place & route (PAR) tool requires two portions of information, namely:

1. **What.** The tool requires to know the functionality and the target device. This information is provided by the *EDIF* file generated by the synthesis tool.

2. **Where.** The tool also requires to know which ports of the design are connected to which pin of the device (see Figure 3). This information is, amongst others, specified in the *User Constraint File (UCF)*.

9 Place & Route automation

To use the PAR tool in an automated fashion, the UCF-file needs to be provided. The user constraints are listed in the file **project.ucf** in the **config** directory. As providing wrong information in this file may lead to short-circuits, wrong connections, etc. You always have to let your supervisor check this file before continuing. For this laboratory the file is already provided. Furthermore, the PAR tool has many parameters that can be specified, the most important ones are provided in the file **project.xilinx** in the **config** directory. This file will be used as is for all the other laboratories.

NOTE: As the PAR tool is dependent on the EDIF file there exists a natural dependence of the PAR tool on the synthesis tool; therefore, the command **make synth** is superseeded by the command **make xilinx_bit** making the former redundant.

An example of the most important UCF definitions is shown below:

Clock definitions:

```
NET CLOCK         LOC = AA12;
NET CLOCK         TNM_NET = "CLOCK_50";
TIMESPEC          TS_CLOCK_50 = PERIOD CLOCK_50 50 MHz HIGH 50 %;
```

Uncritical timing definition:

```
NET reset         LOC = D8;
NET reset         TIG;
```

Pin Assignment:

```
NET LED          LOC = W2;
NET SWITCH_n     LOC = D1;

NET VGA_RED<2>   LOC = X4;
NET VGA_RED<1>   LOC = W5;
NET VGA_RED<0>   LOC = T9;
```

Output driver:

```
NET my_netname   FAST;
NET my_netname   DRIVE = 12;
NET "*"          IOSTANDARD = LVTTL;
```

Pullup/-down:

```
NET netname      PULLUP;
NET netname      PULLDOWN;
```

Combination of multiple instructions:

```
NET netname      LOC = A1 | DRIVE = 2 | FAST;
```

Assignment 6:

To start an automated PAR, execute the command **make xilinx_bit** in the directory **/praktika/btf4220-digital/laboratory2/**.

TROUBLESHOOTING: It can happen that the PAR tool does not complete its tasks. There are a few steps in the PAR, and each can run into problems. Ask your supervisor for help in case PAR does not complete successfully.

10 Download and testing

If all went well, the PAR tool generated a device specific file called **<TOPLEVEL_NAME>.bit** in the directory **sandbox**. We can now test our design on the hardware by downloading the bit file to the FPGA.

IMPORTANT: The FPGA board is connected to your local machine and not to the server. The steps below must, therefore, be executed on your local machine.

We can upload the bitfile to the FPGA by the Xilinx tool **impact**.

DEMO 1: Your supervisor will show how one can upload a bitfile to the FPGA board.

Assignment 7:

Upload your own design to the board and test it.