

Digital Design LU

Design Flow Tutorial

Jakob Lechner, Thomas Polzer
{lechner, tpolzer}@ecs.tuwien.ac.at
Department of Computer Engineering
University of Technology Vienna

Vienna, October 12, 2010

Contents

1	Design Flow	2
1.1	Directory Structure	3
2	Behavioral Simulation	4
2.1	Creating a new Project	4
2.2	Compiling the VHDL Files	5
2.3	Performing the Simulation	6
2.3.1	Automating Simulation Runs	8
3	Synthesis and Place&Route	9
3.1	Creating a new Project	9
3.2	Timinig Analyzer	11
3.3	Pin Mappings	12
3.4	Adding a PLL (optional)	13
3.5	Full Compilation	16
3.6	Export a Project Tcl Script - Optional	16
4	Postlayout Simulation	18
4.1	Automating Simulation Runs	19
5	Download	20
6	Logic Analyzer	21

1 Design Flow

In Figure 1.1 the FPGA¹ design flow, which we will use in this lab course, is outlined. Table 1.1 summarizes the tools required for each of these steps. A tool can either be a software tool like a simulator or a hardware analyzer like a logic analyzer or an oscilloscope.

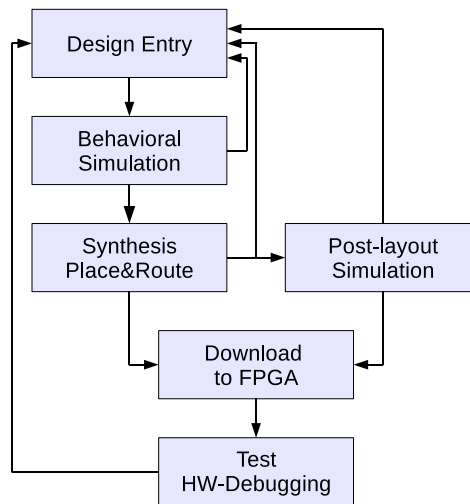


Figure 1.1: The design flow.

Table 1.1: Used tools.

Design Flow Step	Tool
Design Entry	Text editor/ModelSim/Quartus
Behavioral Simulation	ModelSim
Synthesis, Place&Route	Quartus
Post-layout Simulation	ModelSim
Download	Quartus (programmer)
Test & HW Debugging	Logic Analyzer

Design entry During the design entry you will describe your FPGA design with a high-level hardware description language. In this lab course we will use VHDL² for this purpose.

Behavioral Simulation The behavioral simulation is the most important step in verifying the correct functionality of your circuit description. First the simulator compiles your VHDL files and indicates syntax errors. If the compilation runs

¹FPGA: Field Programmable Gate Array

²VHDL: Very high speed integrated circuit Hardware Description Language

without errors, the behavior of the circuit design can be investigated in a waveform showing signal traces of external and internal signals.

Synthesis, Place & Route During the synthesis step your VHDL circuit description is analyzed by the synthesis tool and transformed into a gate-level netlist. The netlist elements can then be placed on the FPGA's configurable cells and connected by suitable interconnect lines. Finally a bitstream file is generated containing the configuration information for the FPGA.

Post-layout Simulation The post-layout simulation is performed on the netlist generated in the synthesis step. Additionally timing information, which are provided by the Place&Route-tool can be used for simulating the final circuit with correct signal timings. A post-layout simulation is useful for resolving timing problems in your circuit. For simple FPGA designs this type of simulation is usually not necessary.

Download The download transfers the bitstream data to the target board for configuring the FPGA.

Test & HW Debugging Finally, the configured FPGA can be tested. In some cases it might be necessary to attach a logic analyzer to the FPGA for verifying the correct functionality of the HW circuit or for debugging errors, which can not be reproduced during simulation.

1.1 Directory Structure

The source code provided for every task assignment is organized with the directory structure described in Table 1.2. We highly recommend you to maintain this structure and add new files you create in the designated subdirectories.

Table 1.2: Directory structure.

Directory Path	Contents
./src	VHDL design files
./src/sim	Source files for testbenches, ModelSim scripts
./src/quartus_hpe_mini	Tcl script for generating Quartus project
./quartus_hpe_mini	Quartus project for target board
./modelsim_beh	ModelSim project for behavioral simulation
./modelsim_post	ModelSim project for post-layout simulation

2 Behavioral Simulation

To start ModelSim Altera enter the following command in the terminal:

```
vsim&
```

2.1 Creating a new Project

A new ModelSim project can be created by clicking the menu item “File” \Rightarrow “New” \Rightarrow “Project...”. Subsequently the dialog which can be seen in Figure 2.1(a) opens. Enter the project name and select the path of the behavioral simulation directory as *project location*.

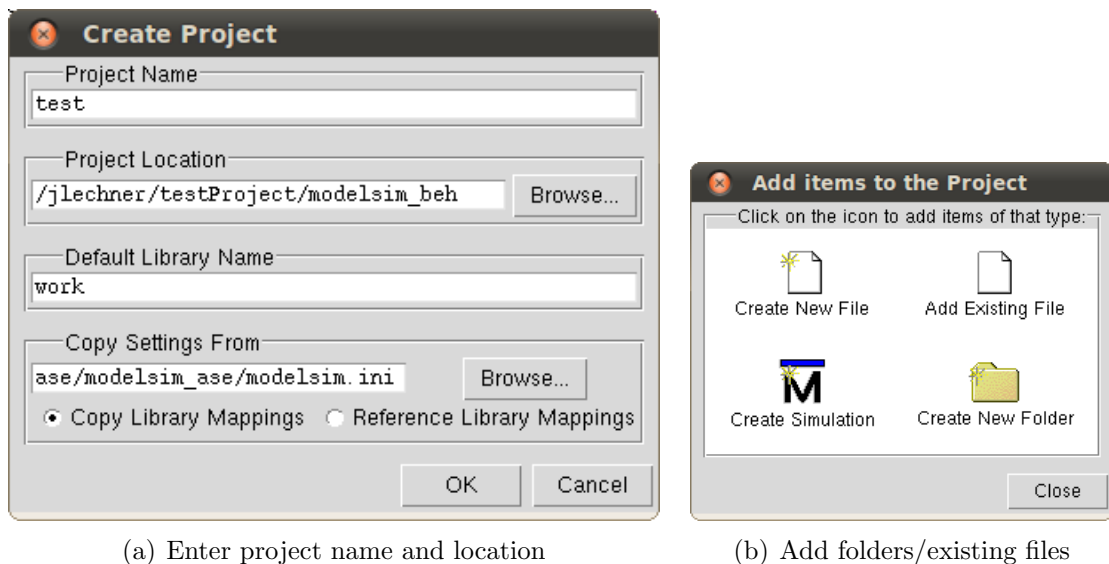


Figure 2.1: Creating a new project.

In the next dialog, ModelSim allows you to add items like source files or folders to your project (Figure 2.1(b)). For a good structure of your project we recommend you to add the following folders:

- Source folders: Create one folder for every module/IP core.
- Simulation folder for adding all your testbench files.
- Script folder for Tcl script files.

These folders will not be created on the filesystem. They are just used in the ModelSim GUI for structuring the project files. Once you have created the necessary folders, you can add VHDL source files by clicking “Add existing File”. Figure 2.2 shows the dialog for adding files. Enter the file name, select the appropriate folder

and make sure the option “*Reference from current location*” is ticked. In Figure 2.3 you can see the project view after all folders and files have been added. Of course, you can also easily add or remove folders/files by right-clicking in the project tree. As you can see in Figure 2.3 the script folder is still empty. We will explain the use of ModelSim scripts later.

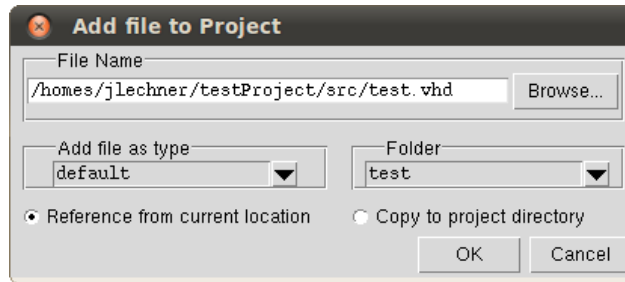


Figure 2.2: Adding a VHDL file.

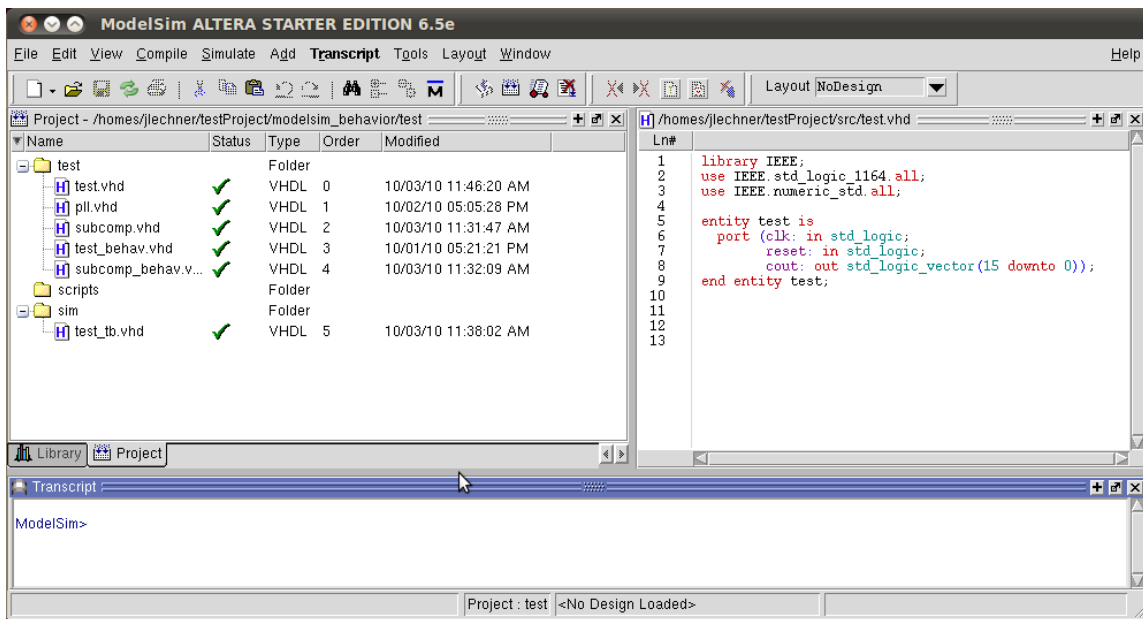


Figure 2.3: ModelSim project with files added.

2.2 Compiling the VHDL Files

Before the the VHDL files can be analyzed and compiled it is necessary to setup the right compilation order. In a VHDL design consisting of multiple files there are usually some VHDL files, which depend on other VHDL files (e.g., a file containing constant declarations will have to be compiled before other files using these constants). Fortunately, ModelSim is able to resolve these build dependencies itself.

Simply click on the menu item “*Compile*” \Rightarrow “*Compile Order...*”. In the next dialog hit the button “*Auto Generate*” (see Figure 2.4).

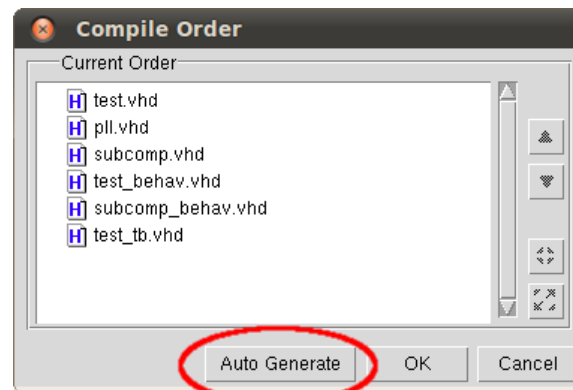


Figure 2.4: Compile order.

Once the file order is fixed, you can compile selected VHDL files with a right-click on the file name or with the “*Compile*” button in the toolbar. You can also compile all source files of the project at once by clicking on the menu item “*Compile*” \Rightarrow “*Compile all*” or the corresponding button in the toolbar.

2.3 Performing the Simulation

When all VHDL files have been compiled successfully, the behavioral simulation can be started. Click on the menu item “*Simulate*” \Rightarrow “*Start Simulation...*”. In the next dialog, which can be seen in Figure 2.5, simply select the name of the testbench and click “*Ok*”.

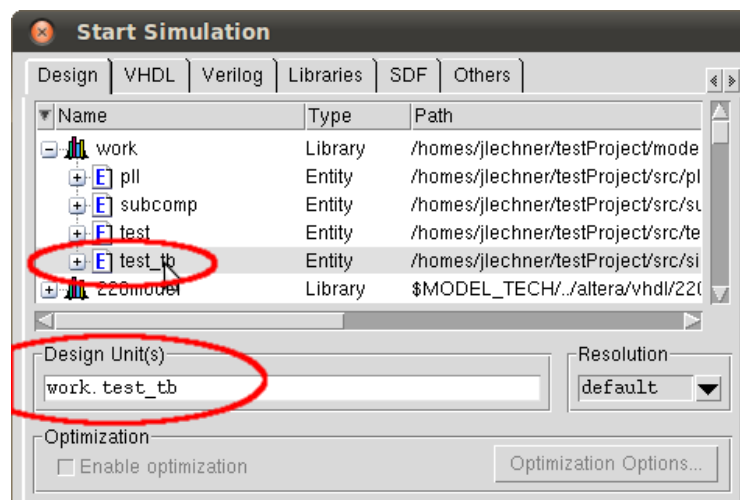


Figure 2.5: Starting the simulation.

Now the main window of ModelSim changes its appearance and shows a hierarchical view of the design to be simulated. With the treeview on the left side of the main window you can navigate through the component instances of your VHDL design. The *object inspector* on the right side shows the input/output ports and the internal signals of the currently selected component. The signals you want to be traced can now be added to the waveform viewer by selecting them in the *objects list* as illustrated in Figure 2.6.

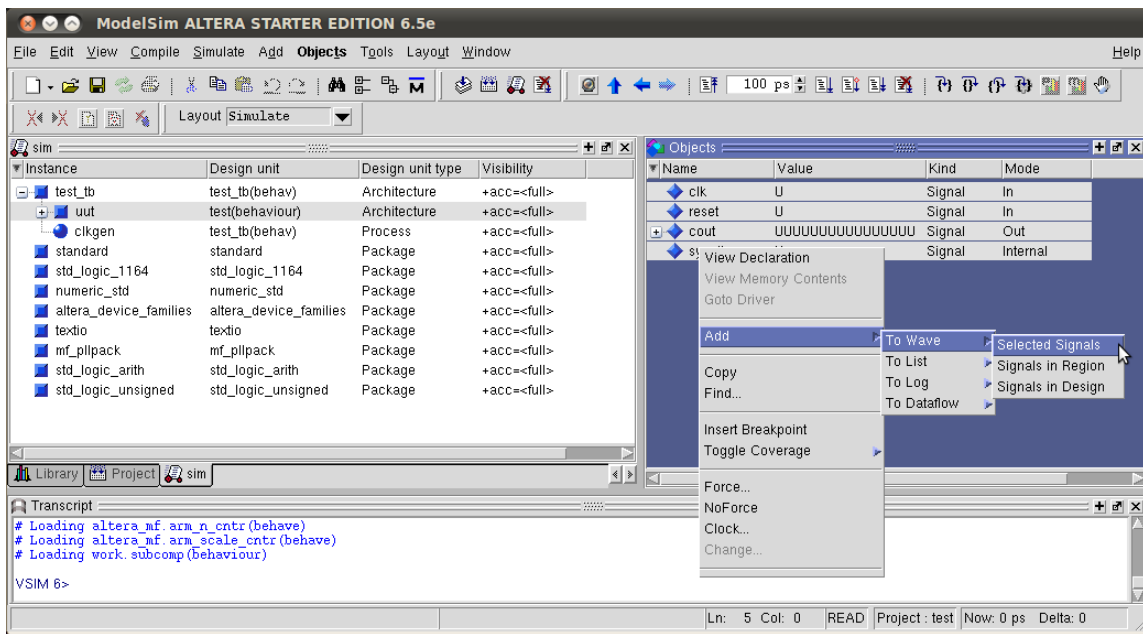


Figure 2.6: Add signals to the waveform.

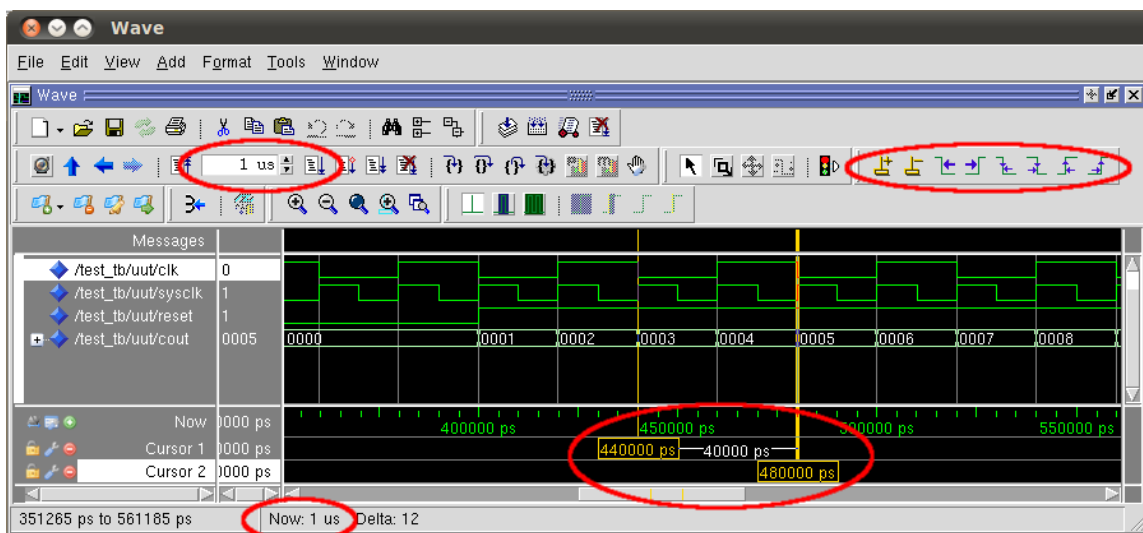


Figure 2.7: The waveform viewer.

Upon adding the signals, the waveform viewer should open automatically. Otherwise you can open it by clicking on the menu item “*View*” \Rightarrow “*Wave*”. You should see the signals you just added on the left side of the waveform viewer. The last step before running the simulation is to adjust the simulation time. This can be done with the small input field in the toolbar of the waveform viewer (see Figure 2.7). The button to the right of this input field then runs the simulation for the specified time.

Figure 2.7 shows a simulation, which has been run for 1 μ s, as can be seen in the statusbar. Two markers have been added for measuring the period of the clock signal drawn in the first line of the waveform. The toolbar buttons for adding, removing and positioning these markers are pictured in Figure 2.7. Another important feature of the waveform viewer is the possibility to change the display format of signal vectors. The values of a signal vector can be displayed in different representations such as binary, hexadecimal, decimal, ASCII, etc. Just right-click on the signal’s name and choose the suitable representation from the “*Radix*” submenu.

2.3.1 Automating Simulation Runs

Often a simulation needs to run several times until the functionality of the circuit has been verified or an error has been corrected. Therefore it is highly recommended to automate the steps necessary for setting up the simulation. Fortunately, all the actions described above (compiling, starting the simulation, adding signals to the waveform viewer, running the simulation for a specified period of time, etc.) can be scripted with *Tcl* scripts. These script files have the file extension “.do”. Listing 1 shows a short example for a *Tcl* script, which executes the tasks mentioned above.

```

1  # compile all source files
2  project compileall
3
4  # start simulation with testbench named ‘test_tb’
5  vsim work.test_tb
6
7  # add signals to waveform viewer
8  add wave -format logic /test_tb/uut/clk
9  add wave -format logic /test_tb/uut/sysclk
10 add wave -format logic /test_tb/uut/reset
11 add wave -format literal -radix hex /test_tb/uut/cout
12
13 # run simulation
14 run 1 us

```

Listing 1: *Tcl* script for running a ModelSim simulation.

We recommend to create such a *Tcl* script for every testcase in the simulation directory, which also contains the corresponding testbenches. Then you can add the do-files to your ModelSim project just like you have added your other source files. In order to execute a script, right-click on the file in the ModelSim GUI and click on the “*Execute*” item in the context menu.

3 Synthesis and Place&Route

Altera Quartus can be started in a terminal with the following command:

```
quartus&
```

3.1 Creating a new Project

In order to create a new project goto menu “File” \Rightarrow “New Project Wizard...”. The wizard lets you configure basic settings of your project. Figure 3.1 to Figure 3.4 show the settings necessary for creating a project for the FPGA board and the tool-chain of this lab course.

After you have successfully created a new project and properly added your VHDL files, you can run a first compilation by clicking the menu entry “Processing” \Rightarrow “Start” \Rightarrow “Start Analysis & Synthesis”. Alternatively, you can click the corresponding icon in the toolbar.

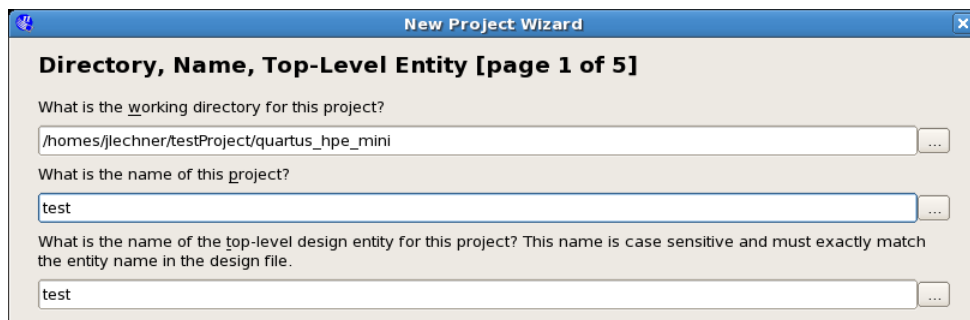


Figure 3.1: New project wizard – Step 1.

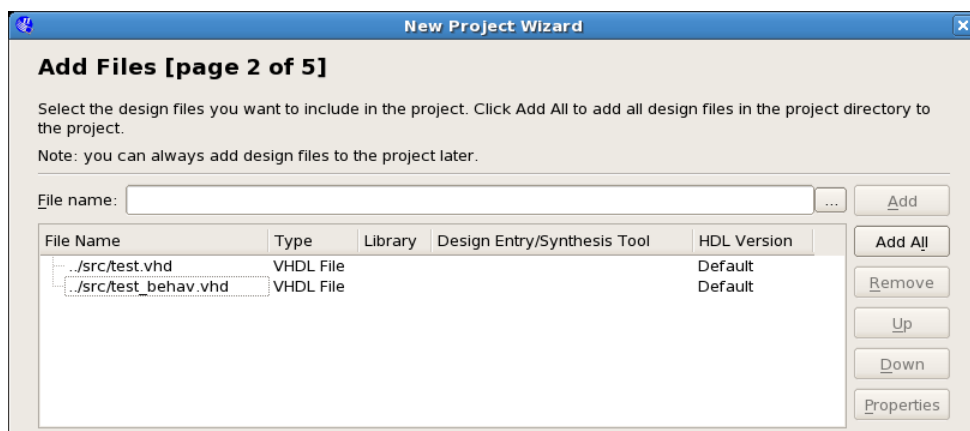


Figure 3.2: New project wizard – Step 2.

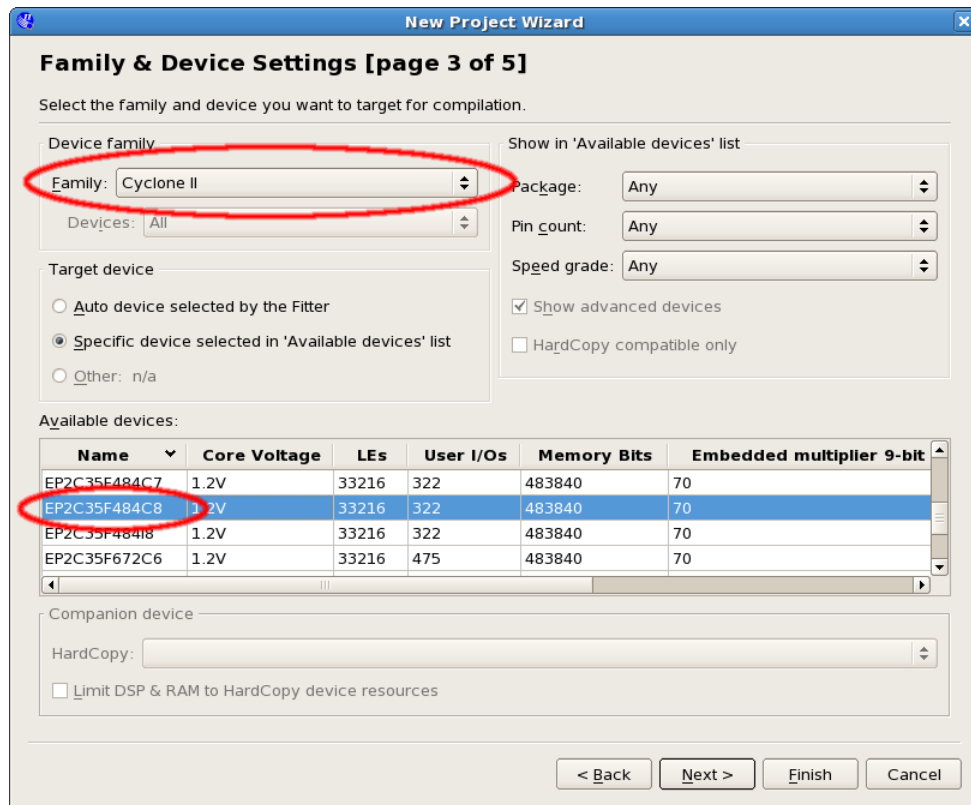


Figure 3.3: New project wizard – Step 3.

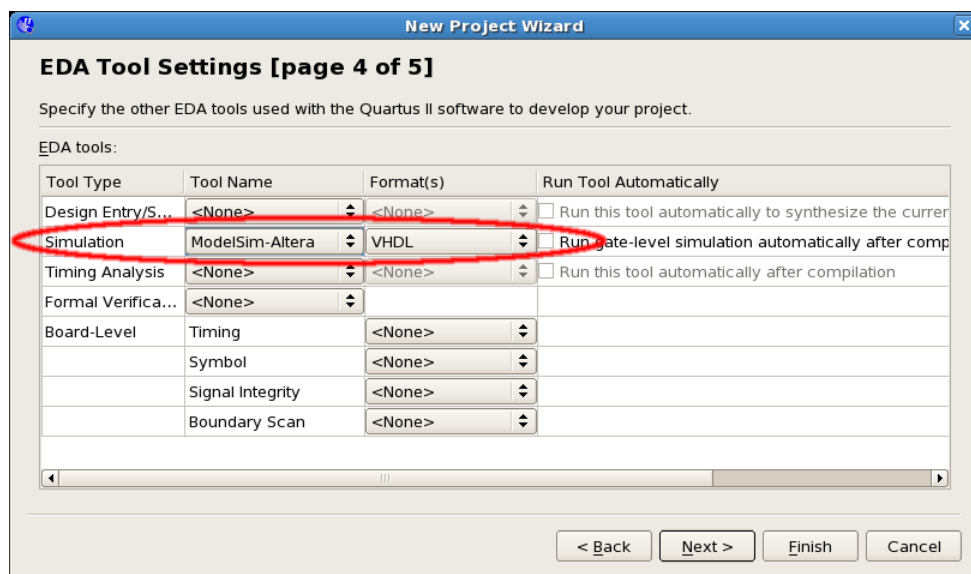


Figure 3.4: New project wizard – Step 4.

3.2 Timinig Analyzer

For this lab course we recommend to use Altera TimeQuest for timing analysis. Goto the menu “*Assignments*” \Rightarrow “*Settings...*”. In the next window select the entry “*Timing Analysis Settings*” in the treeview on the left window side. Subsequently check the option with the title “*Use TimeQuest Timing Analyzer during compilation*”, as can be seen in Figure 3.5.



Figure 3.5: Use TimeQuest Timing Analyzer.

The Timing Analyzer needs to be parametrized with a constraints file. For this purpose Quartus provides a simple wizard, which can be open via the menu “*Assignments*” \Rightarrow “*TimeQuest Timinig Analyzer Wizard...*”. The only constraint, which needs to be specified for a simple FPGA project is the clock period of the external oscillator (see Figure 3.6).

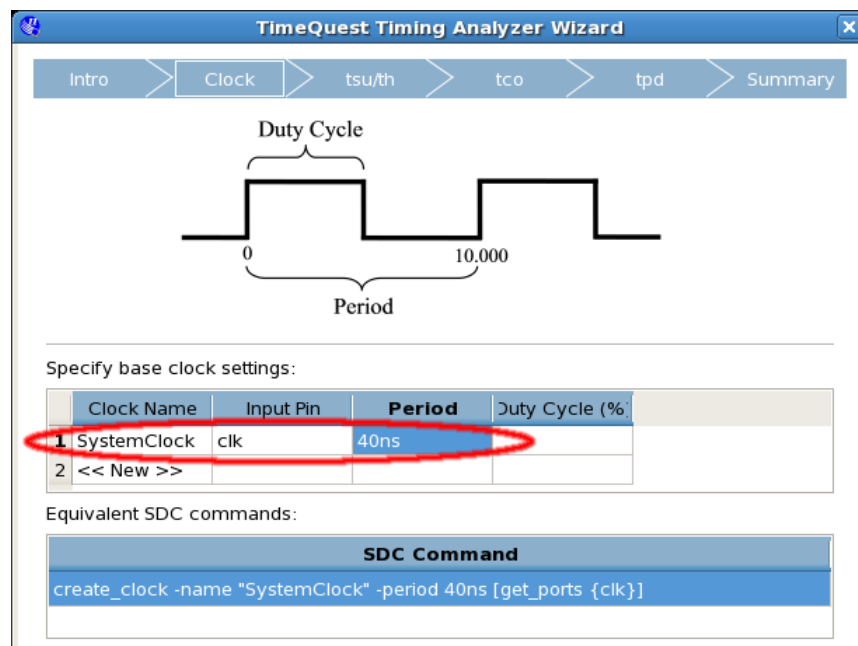


Figure 3.6: Setting the clock period.

The *Clock Name* (first column) can be chosen freely. In the second column the name of the clock input port of your top-level entity needs to be entered. The period of the external oscillator connected to this port can then be specified in the third column. The duty cycle column can be left empty.

All other settings of the wizard can be ignored. They are not necessary for this lab course. Thus, click “*Next*” until you arrive at the last page of the wizard. Then click “*Finish*” in order to add a constraints file, which contains your clock settings, to the project.

3.3 Pin Mappings

Finally, we need to configure the pin assignments of your design. In this task you assign the logical input and output ports of your design’s top-level entity to the physical pins of the FPGA. Obviously, in most cases you can not randomly choose a pin. E.g., the reset port of your design needs to be assigned to exactly the pin, which is connected to the reset button on the FPGA board. The same is true for all other peripheral interfaces like VGA, RS232, key matrix, leds etc. The specific FPGA pin numbers for all this component can be found in the manual of the FPGA board.

In the Quartus user interface the pin assignment task is done with the so-called *Pin Planner* (menu “*Assignments*” \Rightarrow “*Pin Planner*”). In the bottom half of the *Pin Planner* you can see a table listing the input and output ports of your design. The associated pin names need to be entered in the location column of this table. Simply select one entry and start typing the name of the pin (e.g., M1 for the clock input).

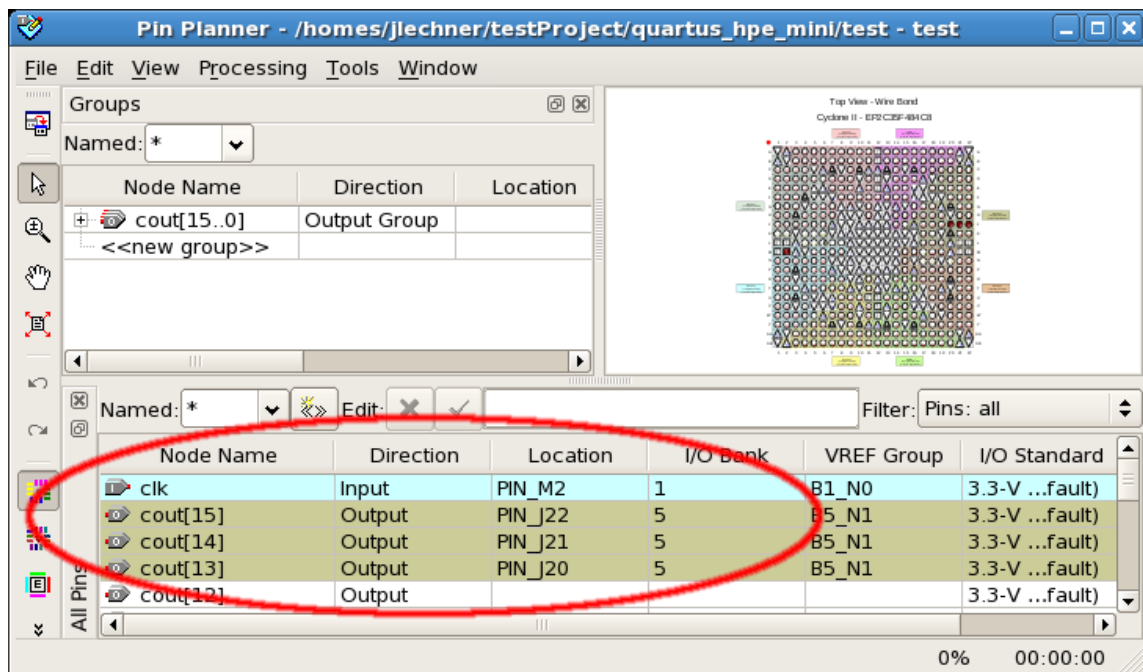


Figure 3.7: Assigning input and output pins.

Make sure *all* ports of your design are assigned to one FPGA pin. All other *unused* FPGA pins should be configured as tri-state input pins with weak pull-up resistors.

This can be done in the device settings dialog (menu “*Assignments*” \Rightarrow “*Device...*”) where you need to click the button “*Device and Pin Options...*”. In the following dialog, select the category “*Unused Pins*” and change the combo-box to “*As input tri-stated with weak pull-up*”.

Caution: The default setting “*As output driving ground*” should only be used if you really know what you are doing! Otherwise it might be harmful for some FPGA boards as it can cause short-circuits in certain circumstances.

3.4 Adding a PLL (optional)

Sometimes it is necessary to run the implemented circuit design with another clock frequency as provided by the external oscillator. Therefore Altera FPGAs include PLLs (phase locked loop). PLLs generate a stable clock signal which can be a rational multiple of the external reference clock.

Quartus provides a wizard, which allows you to configure a PLL component. This wizard can be started over the menu “*Tools*” \Rightarrow “*MegaWizard Plug-In Manager*”. Open the I/O node in the treeview on the left side of the window pictured in Figure 3.8 and select the entry *ALTPLL*. Furthermore you need to specify the file path and file name of the configuration file that will be generated. Then click “*Next*” to proceed.

Figure 3.9 to Figure 3.12 show how to generate a simple PLL configuration necessary for this lab course. The central settings are the frequency of the input clock, i.e., the external oscillator and the frequency of the PLL’s output clock. As can be seen in the symbol in Figure 3.11, the customized PLL component only has two ports (input and output clock). The final screenshot shows the files which are generated by the wizard: The most important file is the VHDL file containing the description of the PLL component based on your settings. Furthermore a cmp-file is generated, which provides the component declaration of the PLL. This declaration can be copied into your top-level design unit to be able to instantiate the PLL.

Finally a dialog box appears asking if you want to add the IP file to the Quartus project. Answer the dialog with “Yes”. The PLL’s VHDL file will then be automatically added to your project.

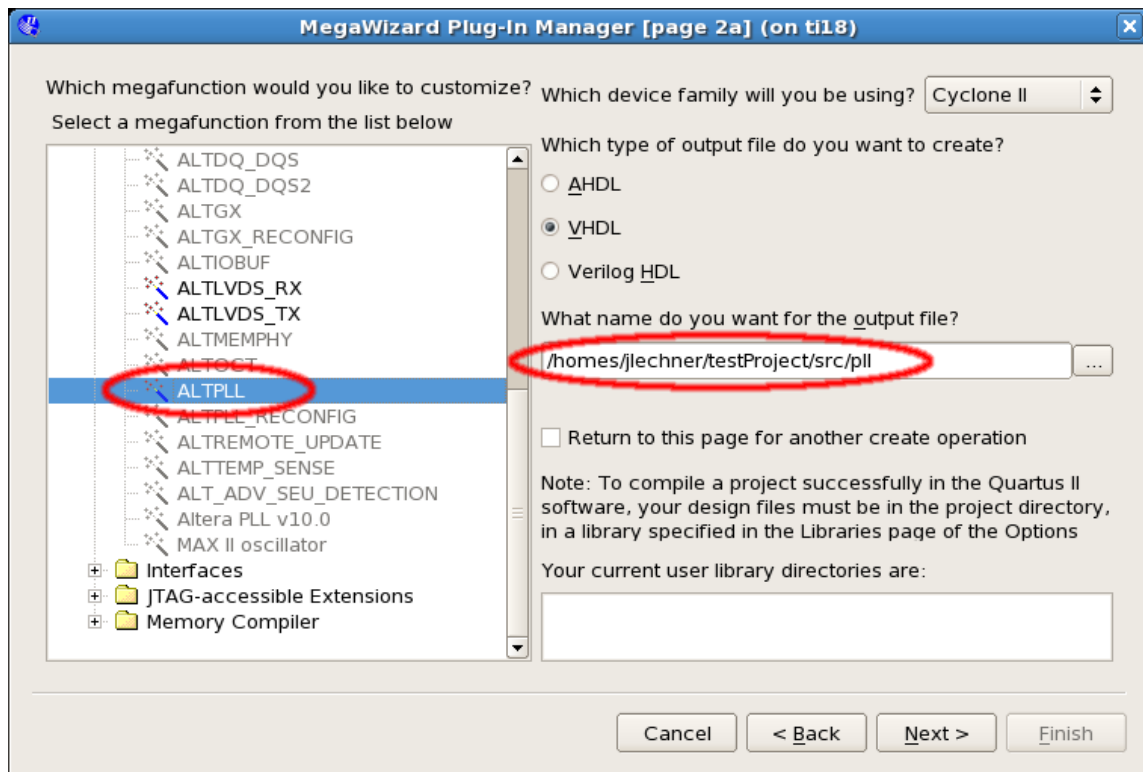


Figure 3.8: Creating a PLL - Step 1.

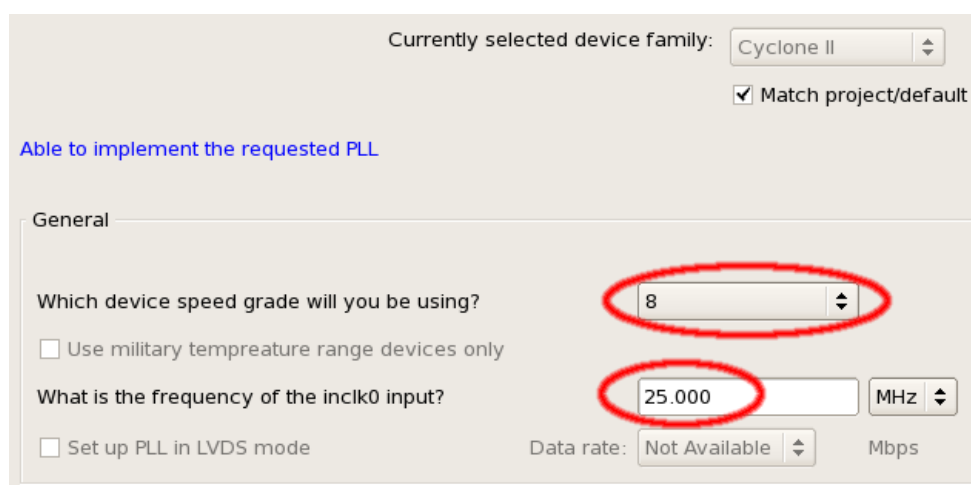


Figure 3.9: Creating a PLL - Step 2.

Optional Inputs

☐ Create an 'pllena' input to selectively enable the PLL

☐ Create an 'areset' input to asynchronously reset the PLL

☐ Create an 'pfdena' input to selectively enable the phase/frequency detector

Lock Output

☐ Create 'locked' output

☐ Enable self-reset on loss lock

Figure 3.10: Creating a PLL - Step 3.

pll

inclk0 frequency: 25.000 MHz
Operation Mode: Normal

Clk	Ratio	Ph (deg)	DC (%)
c0	2/1	0.00	50.00

Cyclone II

c0 - Core/External Output Clock

Able to implement the requested PLL

☒ Use this clock

Clock Tap Settings

Enter output clock frequency:

50.00000000

MHz

Enter output clock parameters:

Clock multiplication factor

1

2

Clock division factor

1

1

Clock phase shift

0.00

deg

0.00

Clock duty cycle (%)

50.00

50.00

Figure 3.11: Creating a PLL - Step 4.

The MegaWizard Plug-In Manager creates the selected files in the following directory:
/homes/jlechner/testProject/src/

File	Description
<input checked="" type="checkbox"/> pll.vhd	Variation file
<input checked="" type="checkbox"/> pll.ppf	PinPlanner ports PPF file
<input type="checkbox"/> pll.inc	AHDL Include file
<input checked="" type="checkbox"/> pll.cmp	VHDL component declaration file
<input type="checkbox"/> pll.bsf	Quartus II symbol file
<input type="checkbox"/> pll_inst.vhd	Instantiation template file

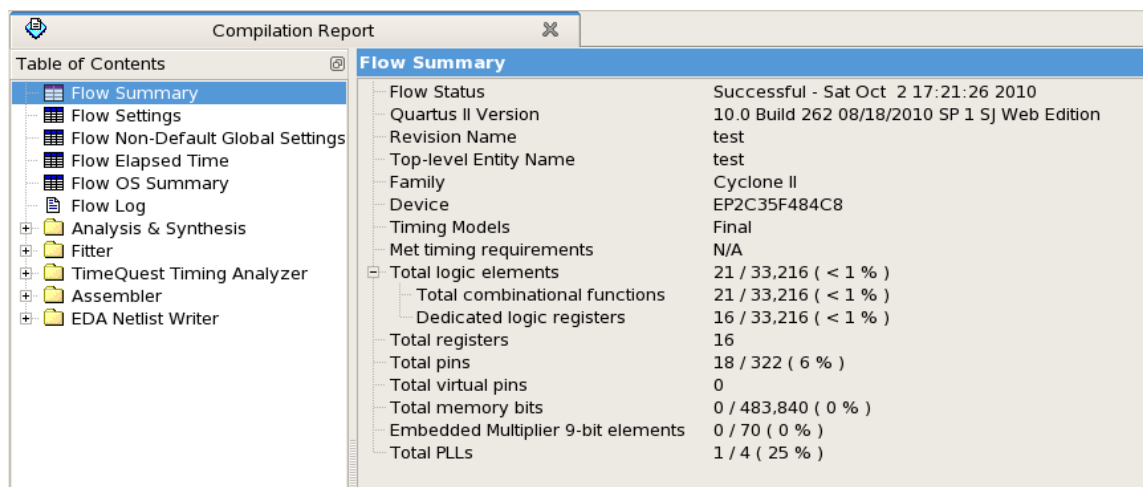
Figure 3.12: Creating a PLL - Step 5.

3.5 Full Compilation

Once the project has been set up correctly, a full compilation can be performed by clicking the menu entry “*Processing*” \Rightarrow “*Start Compilation*”. Alternatively, you can click on the corresponding button in the toolbar. If the compilation runs successfully, Quartus generates the following files:

- Bitstream file for programming the FPGA (.sof)
- Netlist file for post-layout simulation (.vho)
- Timing file for post-layout simulation (.sdo)

The bitstream file can be found directly in the project directory, whereas the simulation files are stored in the subdirectory *simulation/modelsim*. Detailed results of the compilation and the synthesized circuit are shown in the compilation report window (Figure 3.13). Carefully inspect all warnings produced during the compilation. A tidy VHDL design should have no critical warnings and also non-critical warnings should be resolved whenever possible.



Flow Summary	
Flow Status	Successful - Sat Oct 2 17:21:26 2010
Quartus II Version	10.0 Build 262 08/18/2010 SP 1 SJ Web Edition
Revision Name	test
Top-level Entity Name	test
Family	Cyclone II
Device	EP2C35F484C8
Timing Models	Final
Met timing requirements	N/A
Total logic elements	21 / 33,216 (< 1 %)
Total combinational functions	21 / 33,216 (< 1 %)
Dedicated logic registers	16 / 33,216 (< 1 %)
Total registers	16
Total pins	18 / 322 (6 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	1 / 4 (25 %)

Figure 3.13: Compilation Report.

3.6 Export a Project Tcl Script - Optional

Alternatively to using the Quartus GUI, it is also possible to manage a Quartus project with a Tcl script. Quartus can generate such a script for an existing project and include all settings you have made so far. Simply click on the menu item “*Project*” \Rightarrow “*Generate Tcl File for Project...*” and store the script file in the source directory. The Tcl script can then be executed with the following shell command:

```
quartus_sh -t path/to/script.tcl
```

If you execute the above command in an empty directory, a new Quartus project is created in this directory with all settings as specified by the script. If you execute the command in a directory where the corresponding project already exists, the script will simply update the project settings. E.g., if you have modified the pin assignments in the Tcl script, you can update these settings in the existing project by executing the script. This can even be done while the project is opened in Quartus – the new settings will be applied immediately. Listing 2 shows a short code snippet with some Tcl statements.

```
1  # use timequest
2  set_global_assignment -name USE_TIMEQUEST_TIMING_ANALYZER ON
3
4  # add project files
5  set_global_assignment -name VHDL_FILE ../src/subcomp_behav.vhd
6  set_global_assignment -name VHDL_FILE ../src/test_behav.vhd
7  set_global_assignment -name VHDL_FILE ../src/subcomp.vhd
8  set_global_assignment -name VHDL_FILE ../src/test.vhd
9
10 # pin assignments
11 set_location_assignment PIN_M1 -to clk
```

Listing 2: Tcl script for managing a Quartus project.

4 Postlayout Simulation

Create a new ModelSim project in the post-layout simulation directory as described in Section 2. But in contrast to the behavioral simulation the only design file you need to add for a post-layout simulation is the netlist file (.vho). This file is produced by Quartus during the full compilation and is located in the “*simulation/modelsim*” subdirectory of your Quartus project. Furthermore you need to add your testbench files and the simulation scripts to the ModelSim project. In Figure 4.1 you can see a simple post-layout simulation project in ModelSim.

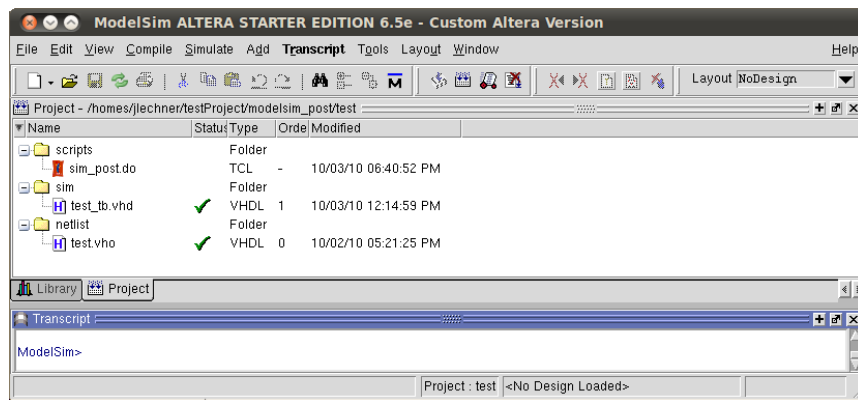
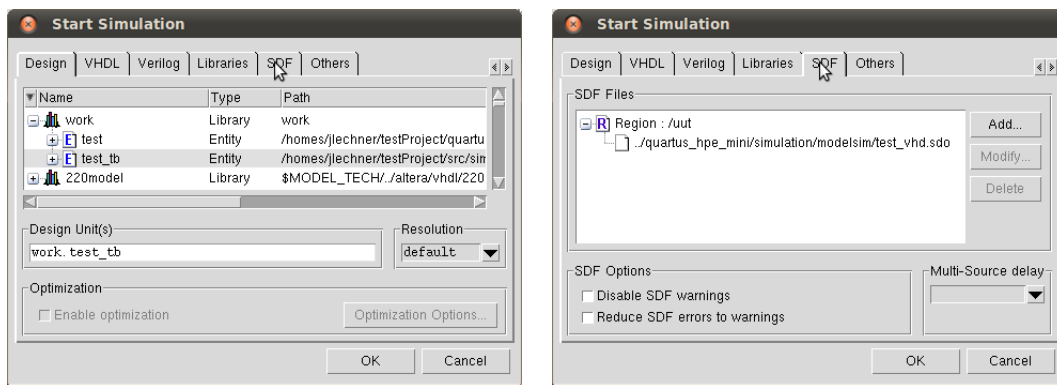


Figure 4.1: ModelSim project for a post-layout simulation.

Hit the “*Compile all*” button to compile the netlist and the testbenches. Then the simulation can be started with the menu “*Simulate*” \Rightarrow “*Start Simulation...*”.



(a) Select testbench name

(b) Add timing file

Figure 4.2: Starting the simulation.

As in a behavioral simulation you have to specify the name of the testbench to be simulated (see Figure 4.2(a)). In case of a post-layout simulation additionally a timing file (.sdo) should be added. This file is also located in the simulation subdirectory of your Quartus project. Click the tab “*SDF*” as illustrated in Figure 4.2(b). and

add the timing file to the list of SDF files. It is important to specify the correct *region* where the sdo file should be applied to. In most cases the testbench directly instantiates the the top-level entity of the design. Thus the region path would simply be the instance name of the design's component in the testbench file. E.g., consider the code snippet of a testbench shown in listing 3. The instance name is “*uut*”, therefore the region path for the timing file needs to be “/*uut*” (compare Figure 4.2(b)).

```
1 uut : test
2     port map (
3         clk    => clk ,
4         reset  => reset ,
5         cout   => cout );
```

Listing 3: Instantiation of the design unter test.

Adding signals to the waveform viewer and running the simulation for a specified time works exactly like when doing a behavioral simulation. The only difference is, that internal signals may have been renamed during the synthesis with Quartus or even removed due to optimization steps. Therefore it might be necessary to add debug output ports to the top-level entity and connect the internal signals to these debug ports. Then the internal signals can be traced using these debug ports.

4.1 Automating Simulation Runs

A post-layout simulation can also be automated very easily with the help of Tcl scripts. Listing 4 shows a basic sample script.

```
1 # compile all source files
2 project compileall
3
4 # start simulation with testbench named "test_tb"
5 vsim -sdftyp /uut=../quartus_hpe/simulation/modelsim/test.vhd.sdo work.test_tb
6
7 # add signals to waveform viewer
8 add wave -format logic /test_tb/uut/clk
9 add wave -format logic /test_tb/uut/reset
10 add wave -format literal -radix hex /test_tb/uut/cout
11
12 # run simulation
13 run 1 us
```

Listing 4: Tcl script for running a post-layout simulation.

5 Download

Programming the FPGA can be easily done with the Quartus Programmer, which is pictured in Figure 5.1. Open the programmer over the menu entry “*Tools*” \Rightarrow “*Programmer*” or the corresponding button in the toolbar. Typically all settings are correct (compare with Figure 5.1) when the window opens and you can simply start the programming routine by clicking the “*Start*” button. The bitstream file (.sof) generated during the compilation is downloaded over a parallel interface and stored in the configuration SRAM of the FPGA. Since an SRAM is not a permanent memory, the configuration data is lost, if the board is disconnected from power. The FPGA then needs to be reprogrammed.

If the programmer window is not configured correctly or the programming procedure fails, click the button “*Auto Detect*” to check if the FPGA is properly connected to your PC. If the FPGA device appears, you can right click on the device symbol and assign the sof file using the menu item “*Change file*”. Make sure “*Program/-Configure*” is checked and hit “*Start*” to begin programming.

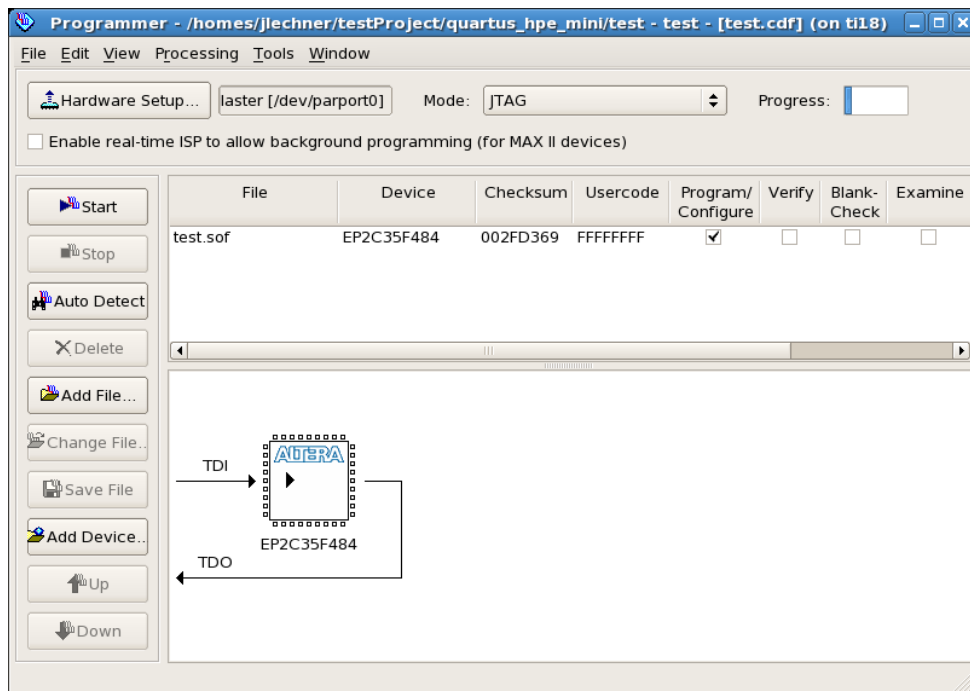


Figure 5.1: Programming the FPGA.

6 Logic Analyzer

The configuration of a logic analyzers is basically done in three steps:

- Defining signals, assignment of signals to the corresponding probes of a pod
- Configuration of the sampling mode – timing or state
- Providing a trigger condition, which defines when data is captured

Figure 6.1 to Figure 6.3 illustrate how these steps can be accomplished with an Agilent 16803 logic analyzer. Figure 6.4 shows the main window, which displays captured waveforms. Simple trigger conditions can be directly entered on the left side of this window.

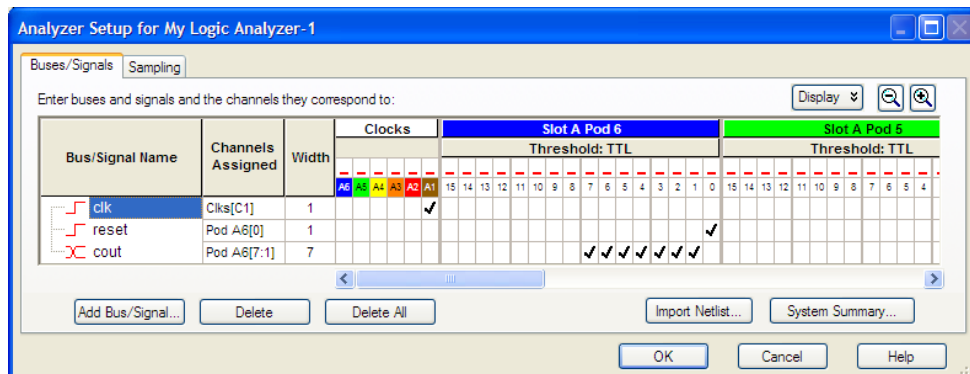


Figure 6.1: Signal setup.

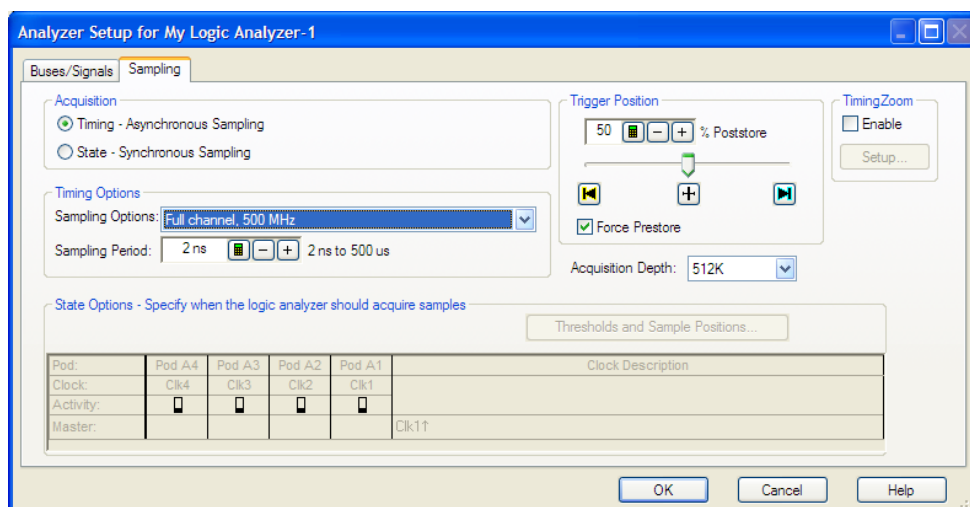


Figure 6.2: Sampling setup.

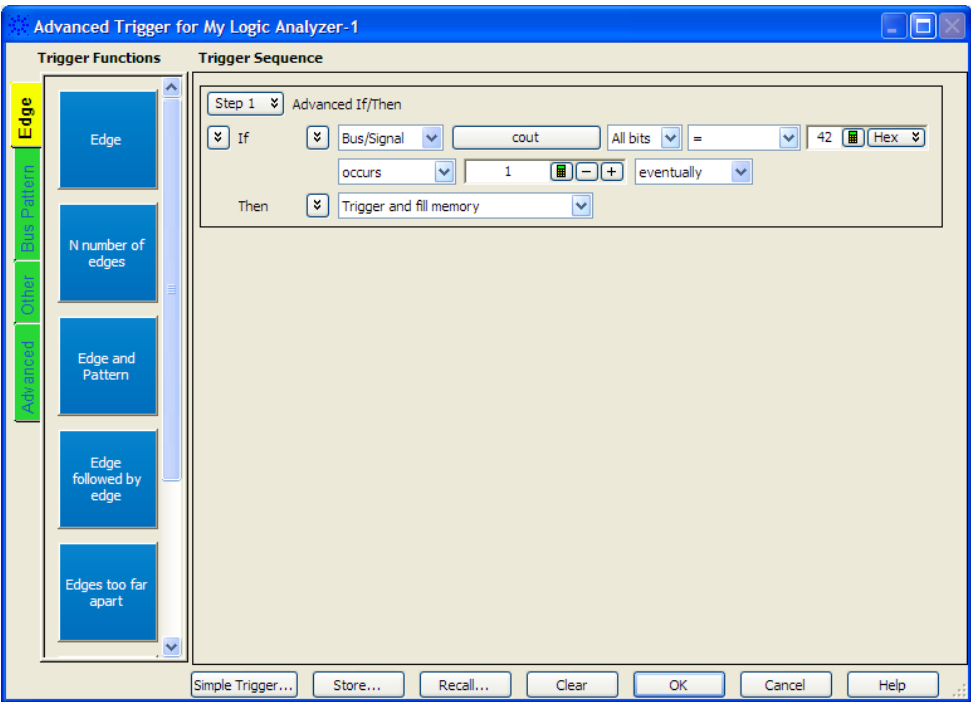


Figure 6.3: Configuring the trigger condition.

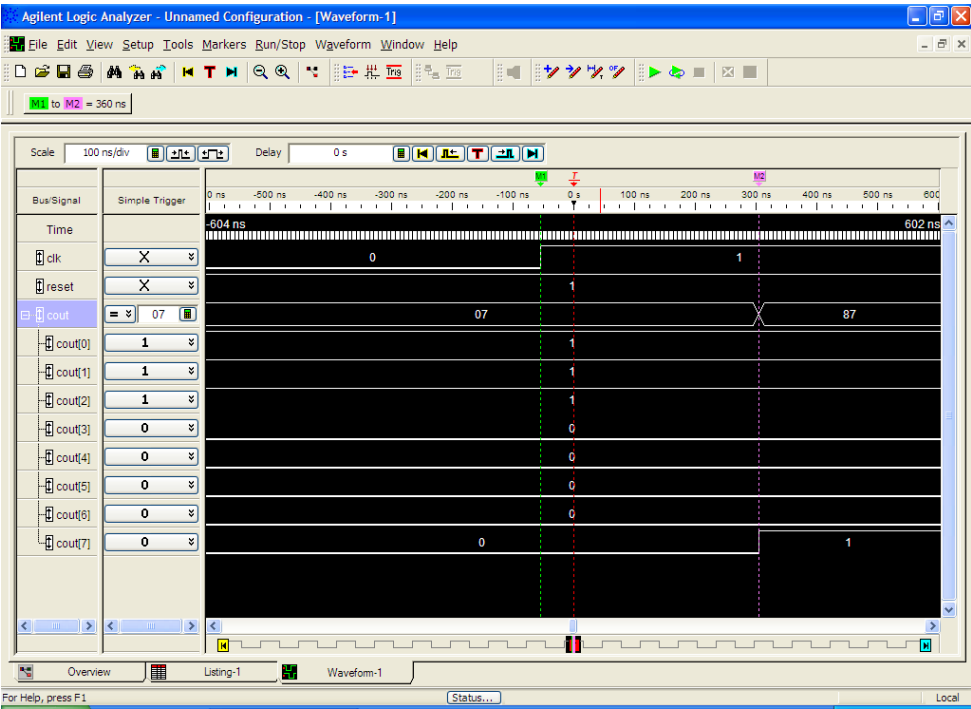


Figure 6.4: Waveform viewer.