Physical Security – Practical Work

Session 1 - Embedded system design and architecture: basic concepts

1. Working environment – OCAE room and tool access

1.1. Connection on PCs and workstations

Local connection on Unix PCs:
Your login will be "xm2cyse5XX" where "XX" must be replaced by the number of your group (e.g. 01, 02, etc. – Note that the group number is specific to this course). The initial password will be provided by your teachers – **this password should be changed just after the first connection and the new password must be reminded by all the students in the group (with the** `yppasswd` **command)!**

Remote connection on workstations:
In the terminal, connect to the CIME workstations *cimplv28, cimelv37,* or *cimepe38* using the ssh command.

Command: ssh -Y cimepe38 | cimplv28 | cimelv37

Session termination:
At the end of the session, it is mandatory **to close the ssh and the PC session**.

1.2. Printing

All printings must be directed to the printer in the OCAE room, called "ocaeimp". This printer can be accessed from both PCs and workstations. It is also available in the CAD environments used during the lab sessions (and in the text editors).

1.3 Disk space

Due to network policies, all the accounts have limited quotas. This limit is quite large, however keep in mind that you may **not** have **enough** space to keep all the projects stored. If you think that you may run out of space, when starting a project, **archive** the previous one on a **personal** media, such as a USB key.

2. Global presentation of the session 1 work

2.1. Contents

The goal of this first part of the practical work is to understand the basics of a development flow for SoPC-based embedded systems. Many files are therefore delivered at the beginning of the work and the first task is to understand what is in these files, and the use of the tools, as explained in the next sections of this document.
At the end of this part of the practical work, you should be able to simulate a circuit description at several abstraction levels and to implement the circuit (or IP – Intellectual Property) as a coprocessor of the embedded processor in a Virtex V device.

2.2. Goals and expected report contents

Most of the circuit descriptions are delivered, but some of them are not complete. You should complete these descriptions, so that you can achieve a simulation of the whole system, including the IP and the embedded processor. Once the validation of the whole system is achieved, the implementation will be done on a XUP board.
The report should summarize the work carried out, including the explanation of the parts added to the initial files. Problems encountered and solved during the work should be emphasized.

3. CAD environment on the workstations

3.1. Directories and project structure

The tutorial files are in the directory "Security_lab/AES". The tool configurations can be set up by running the following script:

cd Security_lab
source script_InitTP.sh

A file called "modelsim.ini" also defines the access paths to the design libraries. The location of this file must be defined using an environment variable in the .cshrc or .bashrc file (*setenv MODELSIM $HOME/Security_lab/modelsim.ini*). It will be used by the VHDL compiler and simulator. The environment variable should be properly set already.

The files related to the project of the first session are in a subdirectory named *AES*. The files related to the circuit and testbench descriptions are hence in separated directories, each of them grouping a specific set of files:

- vhd: source files of the circuit and script for behavioral simulation of the circuit blocks
- bench: source files of the testbench and script for validation
- ise_support: script files for the simulation of the architecture within ISE
- edk_support: template files for developing the EDK project (peripheral interface, software application template, simulation scripts). These files may need to be completed.

Tool commands (e.g. library creation or removal, compilation …) may be written once in a script file, that can be executed using the command "source".

Warning: all names must be coherent! It is also suggested not to include several entities in the same file, and to give the same name to the file and to the entity.

Example: Circuit source file : vhd/alu.vhd
Circuit entity : alu

Circuit architecture	: alu_behav
Circuit library	: vhd/lib_alu
Circuit script	: vhd/scriptsAlu
Bench file	: bench/alu_bench.vhd
Bench entity	: alu_bench
Bench architecture	: alu_bench_behav
Bench configurations	: cfg_alu_bench_behav, cfg_alu_bench_synth
Bench library	: bench/lib_alu_bench
Bench script	: bench/scriptsAlu_bench
etc.	

Note:
The "modelsim.ini" file should be updated each time a new library has to be created for the design. It initially defines only a few default working libraries, in addition to the technology libraries. For each new library with a logical name "lib_COMP", the full physical path must be specified, for example:

lib_COMP = <path>/lib_COMP

3.2. Tools and global simulation environment

All parts of the project will be developed in VHDL language.
The main tools used during the lab sessions (in addition to text editors) are:
- Modelsim: VHDL simulation
- ISE: Xilinx synthesis and place and route tool for programmable chips (FPGAs)
- XPS/EDK: Xilinx embedded system environment for development boards with FPGAs

4. Simulation with ModelSim (versions 6.0 or above) as a stand-alone tool

4.1. Standard packages and documentation

The source files of available packages can be found in the directory /softs/modeltech/vhdl_src/. Available packages include the following ones:

- "STANDARD": defines standard types "boolean, bit, character, severity_level, integer, real, time, string, bit_vector", and sub-types "natural, positive". Sources are in subdirectory std/
- "TEXTIO": defines ASCII VHDL Input/Output primitives. Sources are in subdirectory std/
- "STD_LOGIC_1164": defines the extended bit type (with 9 states) called "STD_ULOGIC" and a resolved subtype called "STD_LOGIC", plus the related composite types "STD_ULOGIC_VECTOR" and "STD_LOGIC_VECTOR". Usual functions on these types are also defined. Sources are in subdirectory ieee/
- "NUMERIC_STD": defines the arithmetic functions on STD_LOGIC_VECTOR. Sources are in subdirectory ieee/

Packages are available by using the following clauses in the VHDL source code:

```
library IEEE ;
use IEEE.STD_LOGIC_1164.all ;
or
use IEEE.NUMERIC_STD.all ;
```

4.2. Loading and simulating a design unit

Before simulating (using Modelsim as a stand-alone tool), a library must first be created using the **vlib** command. Please note that some commands will create many file and folders, so it is advised to run them in a suitable folder to keep your home directory cleaner and more readable.
Example: vlib lib_Project
The files to be simulated must then be compiled into the library using the **vcom** command, specifying the target library with the option **-work**. The files must be compiled with respect to the circuit hierarchy, starting from the leaves of the hierarchical tree.
Example: vcom –work lib_Project <*vhdl file*>

The graphical simulation environment is started using the command **vsim –gui &**. All files in the current directory will be available from the simulation environment, so that simulation command files can be invoked in the simulator (do MyCommands.do). **Several scripts already exist in your project directory to help you running the first commands**. For example, the compilation scripts described above are already in the *AES* directory, stored in the files *scripts*. Try to run the scripts from a terminal window, in the *AES* directory

```
source comp_aes.do
source comp_bench.do
```

do not change the directory while compiling; otherwise, the libraries will not be able to see each other. The same scripts can be run within ModelSim through the *do* command.

Simulations can be either behavioral or after synthesis or after placement and routing, using the same simulation environment (see respective sections for specificities).

Loading the design unit
The "Workspace" window shows the project structure. Select the library corresponding to the testbench to be simulated (i.e. lib_bench) and load one of the configurations.

Loading a new simulation unit can be done with the menu `Simulate → Start simulation`. The time unit can be specified in this menu ("ns" by default). You can also run the simulation with the command `vsim <entity>:`

```
vsim lib_bench.test_core
```

You can also start the simulation by double-clicking the corresponding entity from the list.

Observing signals and waveforms

The "Objects" window opens when loading a simulation unit. This window shows the Inputs/Outputs of the block selected in the Workspace (signals defined in the entity), and can also be obtained from the menu View -> Debug Windows.
Source VHDL descriptions can be displayed from the "Files" section in the Workspace.

Signals to observe as waveforms are defined in the "Objects" window (pop-up menu called with a click on the right mouse button, then "Add to Wave"). The added signals can be:

"Selected signals": signals selected in the "Objects" window before activating the command.
"Signals in Region": entity ports.
"Signals in Design": all internal signals.

From a command file or from the command line, the commands "add wave signal_name" or "add wave /*" can also be used to display either one signal or all the signals in the top-level entity.

In the waveform window, cursors can be placed at significant positions to measure durations. Additional cursors can be obtained from menu "Add -> Cursor".

Waveforms can be printed or saved in a Postscript file: menu "File -> Print Postscript", printing in a file.

Stimuli definition

A testbench should normally define most of the simulation conditions, as it is the case for this tutorial. However, in some cases, simulation stimuli may be more easily defined directly in the simulation environment. The basic command is the "force" command (available in the pop-up menu in the "Objects" window, or from the command line.

Examples:	force x 000	forces value 000 on signal x from the current simulation time
	force x(1) 1	forces at value 1 the first bit of signal x from the current simulation time
	force clk 1 30, 0 80 -repeat 100	creates a waveform for signal clk with period 100, clk = 1 at time 30 and clk = 0 at time 80 within the period

Simulation execution

Simulation execution is started with the menu Simulate →Run.
The "Step" command allows step-by-step simulation with source code visualization.
The "run Duration" command executes "Duration" time units (ex.: run 100). For this example, run for 4.05 us (`run 4050 ns`, for example).

Command file

A command file may be used either to define the simulation environment (signals to monitor, display options, etc.) or to reproduce some stimuli. It is recommended to call such a file with extension ".do".
The file can be executed from the command line with the command "**do file_name**".

4.3. Simulation after synthesis

The result of the synthesis step is saved as a netlist in VHDL or Verilog (VHDL is recommended here, to be easily able to use the same testbench as before synthesis).

In order to simulate this netlist, it is necessary to specify the simulation models of the cells in the library used during synthesis. This implies to define the cell library to be used. One example is:

```
Library C35_CORELIB;
Use C35_CORELIB.Vcomponents.all;
```

Warning: in case of hierarchical netlist, this must be added before EACH entity definition.

The library to be used after a synthesis on Xilinx device is UNISIM. The netlist is available after synthesis in the "netgen" directory.

The testbench (and/or the simulation command file) used for behavioral simulations can be reused, by modifying the pointed descriptions in the configuration files (e.g. libraries in the "synth" directory rather than the "vhd" directory).
Post-synthesis simulation will be described more in detail in the next chapter.

4.4. Simulation after placement and routing

After placement and routing, the parasitic data must be fed back in the simulation by means of an "sdf" file, specified in the menu "Simulate" (validate "Reduce SDF errors to warnings"). Post P&R simulation will be described more in detail in the next chapter.

5. Synthesis, placement and routing with ISE (version 12.2)

5.1. Documentation

The ISE environment can be started using the `ise` command at the terminal prompt. The easiest way is to run the `ise` command from the directory `Security_lab/AES/synth`. Run

```
ise &
```

If the environment variables are properly set up, the main application will pop up in a few seconds. An extensive online help can be invoked from the menu. In particular, an HTML version of the documentation can be retrieved with the menu command

```
Help → Help Topics,
```

whereas an electronic version of the user manual can be read by invoking

```
Help → Software Manuals.
```

In the next sections it will be shown how to create a simple project implementing an AES encryption core.

5.2. Creating an ISE project

1. Open the ISE application
2. **Create a New Project**: choose the menu item

   `File → New Project…`

   This will start the *New Project Wizard*.
3. **New Project Wizard.** A new window pops up:

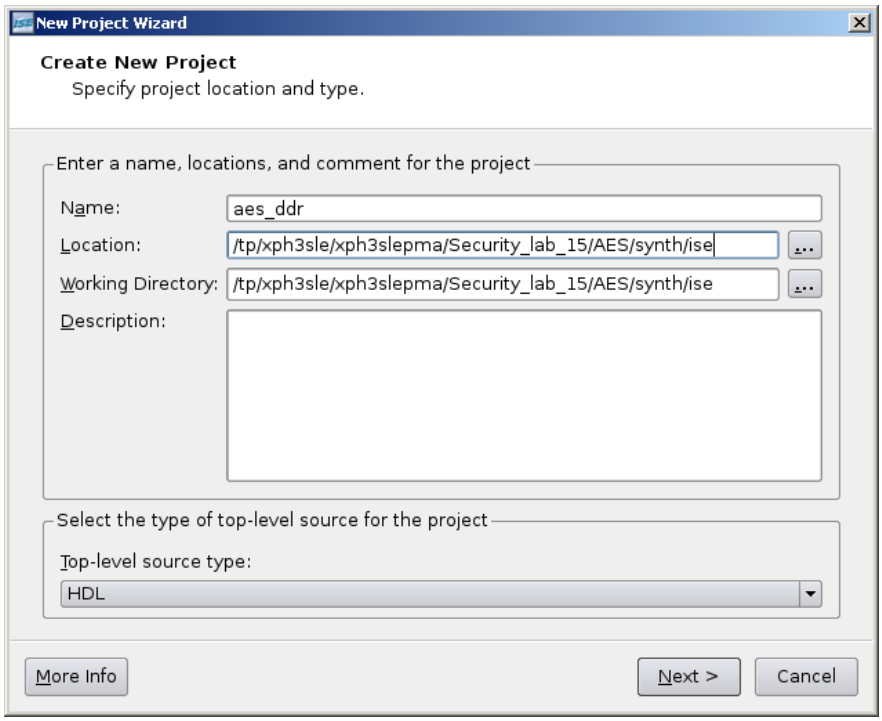


Figure 1: New ISE project wizard

Write a name for the project in *Project Name* field and optionally change the location of the project folder (a subfolder is created using the project name by default). The name you choose is not mandatorily the one shown in Figure 1, but consider that changing this may force you to modify several parameters and files that are already provided you to start the project.
Leave the other fields at their default value. Click the *Next* button.
Note: it is advisable to avoid spaces in the name; rather, use points (“.”), underscores (“_”), or dashes (“-”).

4. The window is now entitled “*Device properties*”. Here you can choose the target implementation device: either choose the specifications of your target board or leave the default values. This can be changed anytime later: for the instant, **define** the board parameters as shown in Figure 2; **set** also the *Synthesis* and *Simulation* tool as in the figure. Finally, click *Next.* The project wizard is now complete. Before returning to the main application, a summary of the project is presented, then click *Finish*.

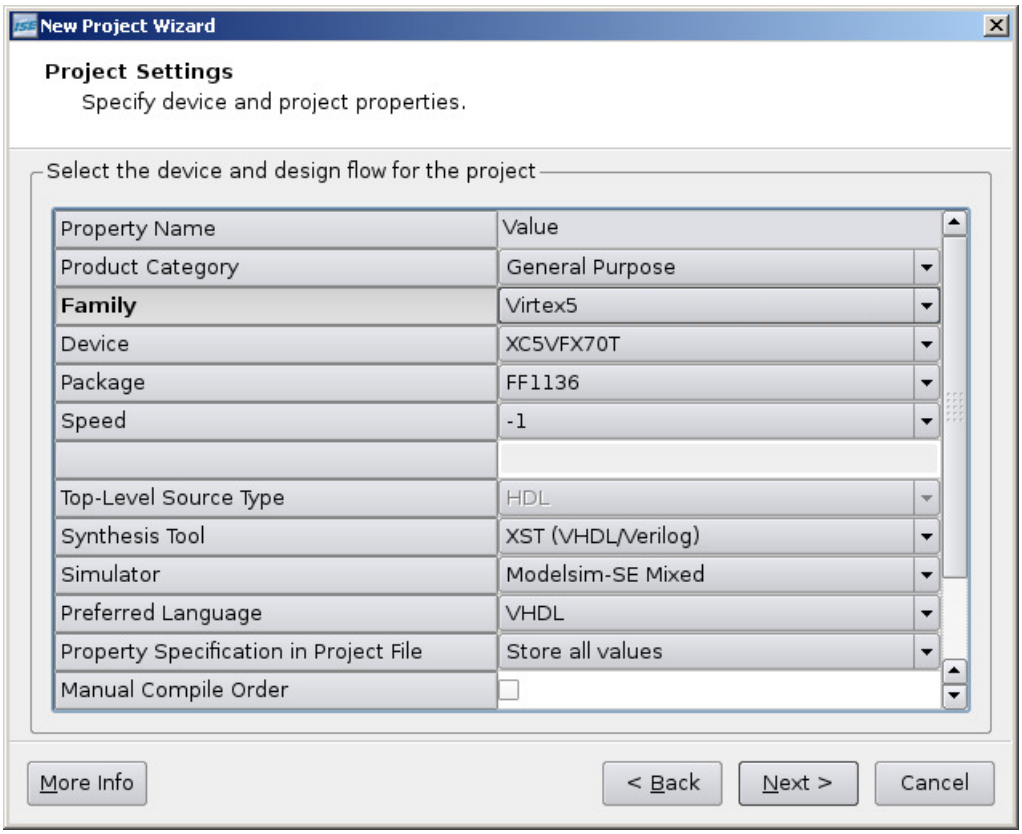


Figure 2: Device properties

5. Select the menu *Project→Add Sources*. You can add any already existing file to the project. The selected files do not need to be complete or error free: debugging and completion can be performed later, once the project is defined and open.
Add all the provided VHDL source files: the AES description (in the `vhd` directory, for both implementation and simulation) and the test bench (in the `bench` directory, only for simulation). Since you will use custom scripts for simulations, you may either leave the default value in the library field, or set a new one yourself as in the figure below.

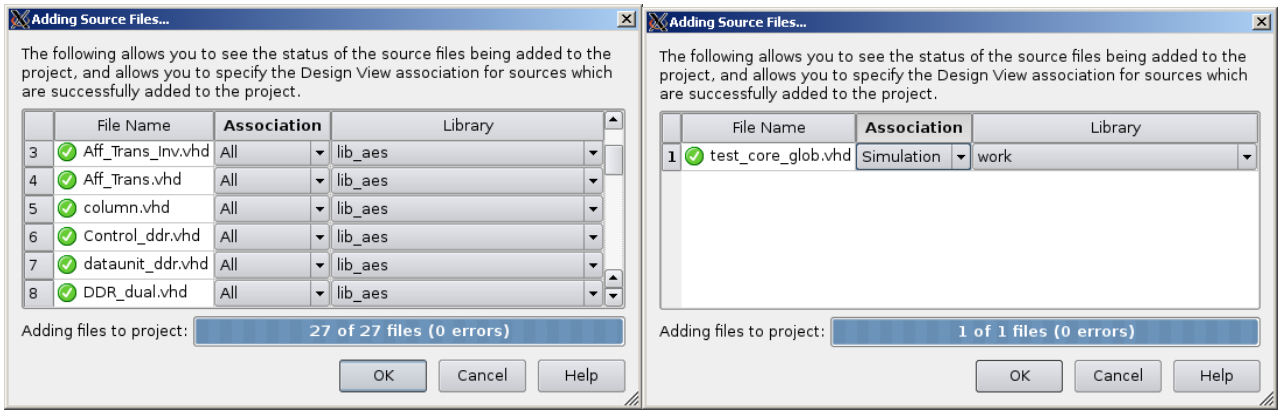


Figure 3: Add existing HDL sources

6. **Application interface.** The application window is divided into several areas which have different goals:

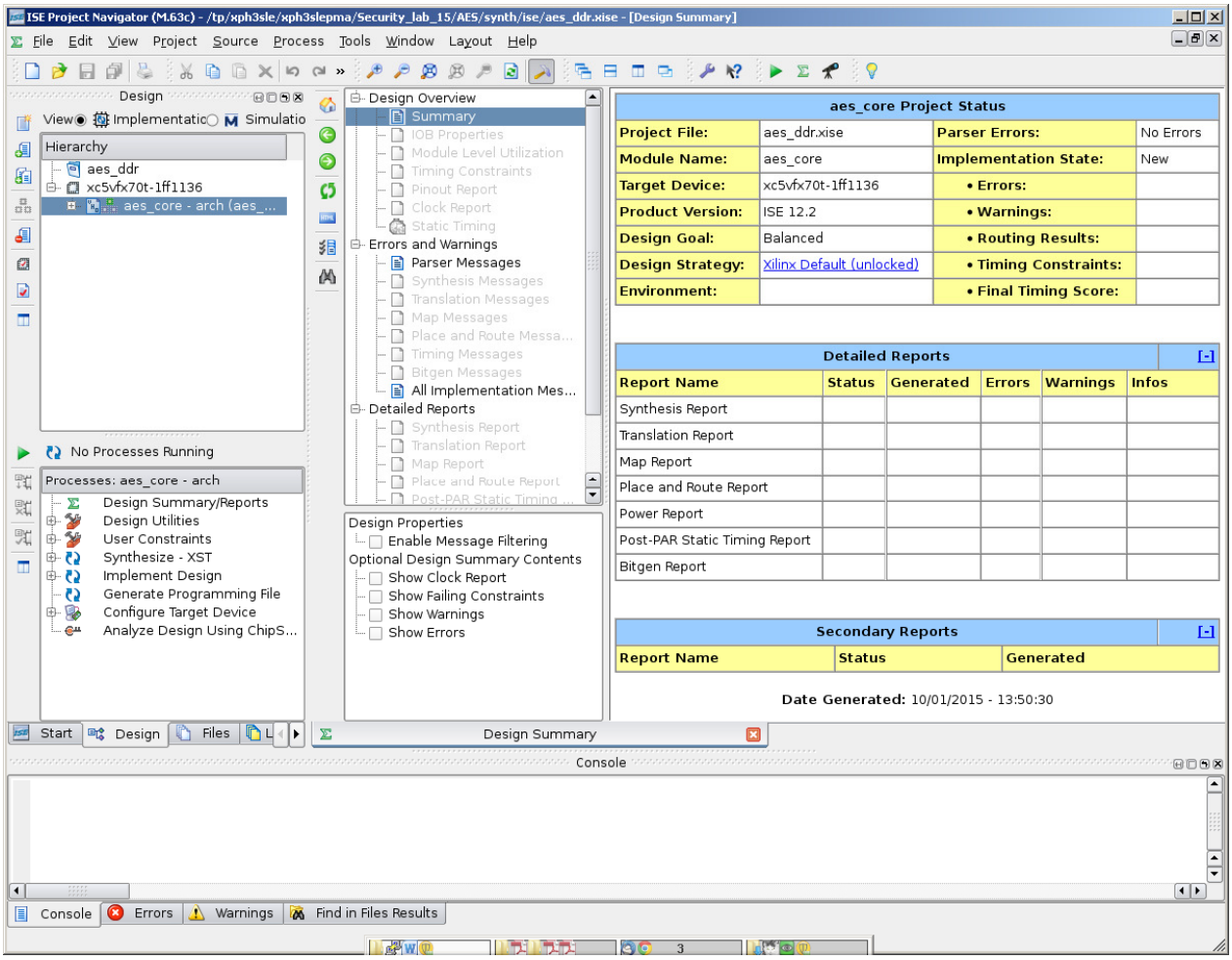


Figure 4: Main application window

In the top left part, there is the main project browser, which can be used to browse within the files and the library of the project. Note that a scroll down menu at the top edge allows selecting the current activity: synthesis or simulation at different levels. The root of the file

hierarchy is the target device: double clicking allows modifying the target device and other configuration settings.
The middle left window lists the processes that can be started on the element selected in the file browser: for instance, for a synthesizable component it allows starting the synthesis, the mapping, the placing and routing and so on. The application tracks all the changes to files and settings: hence, if a process has already run and no changes to the source files and settings occurred, running it again must be explicitly ordered. Likely, after changes to sources or settings, the validity of the results expires and the process must be run again. If a specific process requires that others are run in advance, the application resolves all the dependencies by itself and executes all those required.
In the right part of the window there is the *Design Summary*. Here the results of the executed processes can be accessed quickly and examined. For instance, this window reports the device utilization rate and the number of errors/warning occurred during each process, which can be accessed easily through these links.
The bottom window displays the output of the current running process. During the process execution, it shows the current activity, warnings and errors.

7. **Pre-synthesis verification.** Before starting the implementation of the architecture, it is advisable to do a preliminary test of the design. This can be done by simulating the design at the *behavioral* level. A behavioral simulation takes into consideration only the functionality of a component and it does not require details about its actual implementation or internal structure. The behavioral simulation can be either performed by defining a new standalone project in the simulation tool (e.g., ModelSim by Mentor), or it can run directly from the ISE application. The simulator can be launched directly from the ISE interface by selecting *Behavioral Simulation* in the drop down source filter and the test bench file, which should be the first element just below the target board name. The simulator can be started by executing the *Simulate* command:

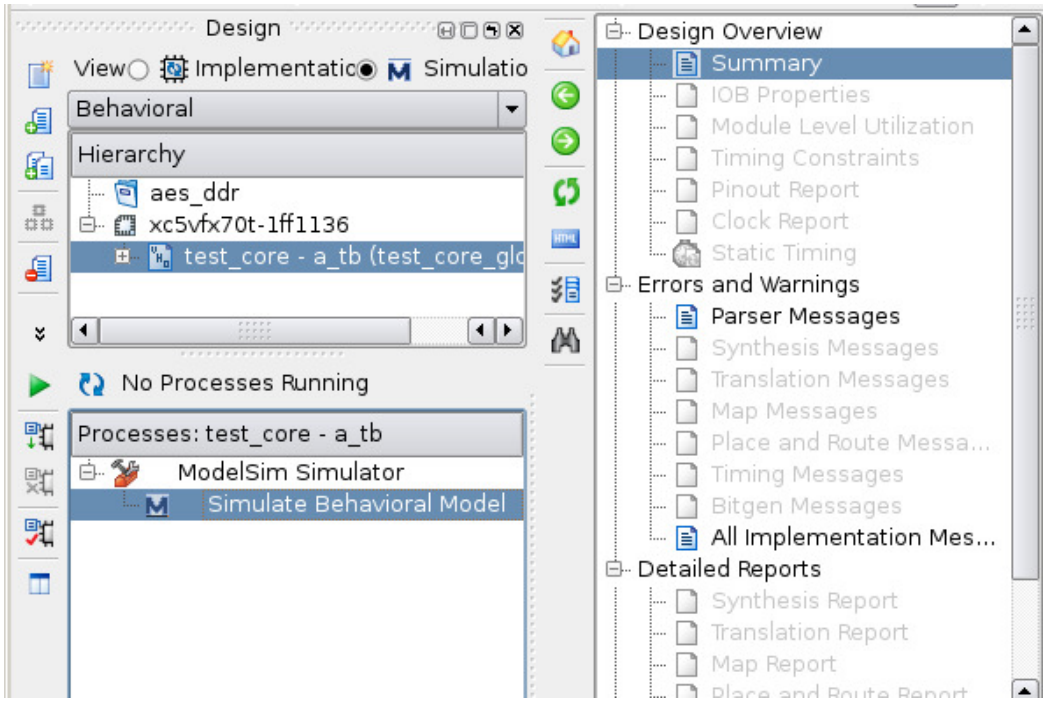


Figure 5: Behavioral simulation (pre-synthesis)

This command starts the simulator; when set appropriately, it also loads and compiles the source modules, and starts the simulation for a predetermined amount of time (usually 1 µs). This is accomplished since the default behavior is to use the script file that is generated automatically. For our example, however, **open** the *Properties* window of the *Simulate Behavioral Model* command and **select the user script file** as shown in Figure 6.

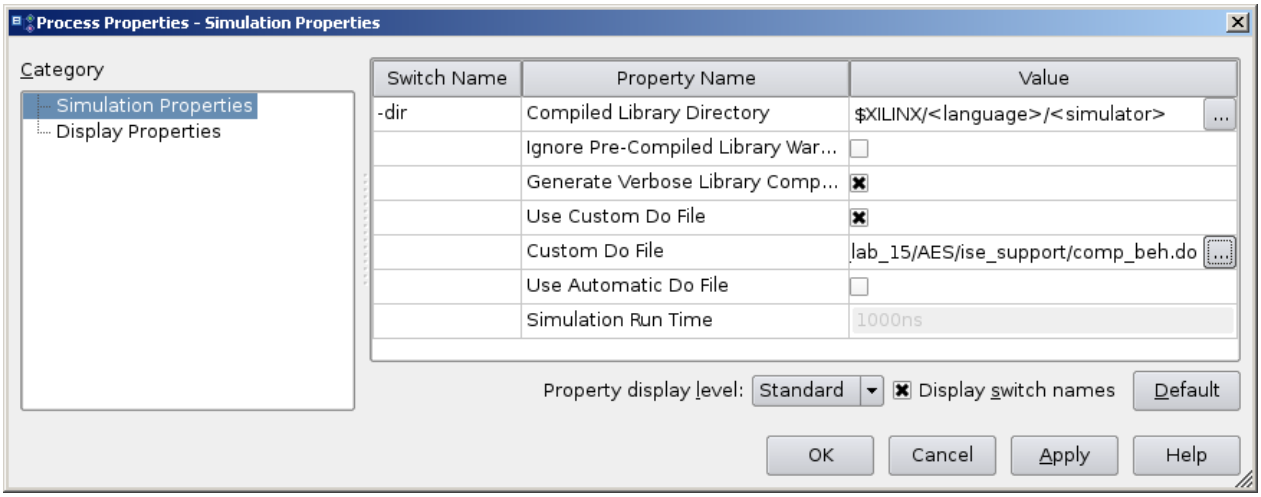


Figure 6: Parameters for behavioral simulation

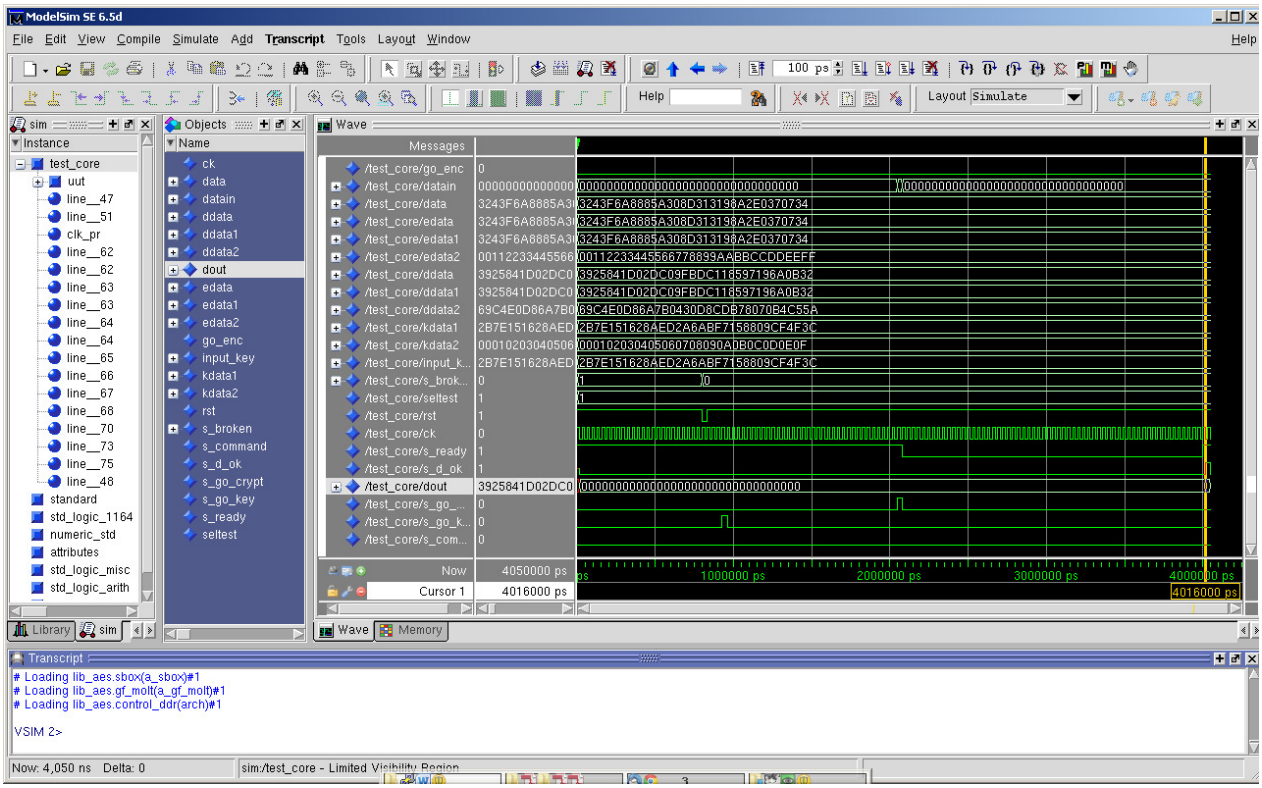


Figure 7: Behavioral simulation in ModelSim

It can be noted that the simulation environment shows all the signals defined in the test bench. Internal signals can be added and the simulation restarted to further explore the

behavior of the system. It can be seen that many internal signals are still encoded in symbolic form (namely, the state signals), which is typical of behavioral simulation.

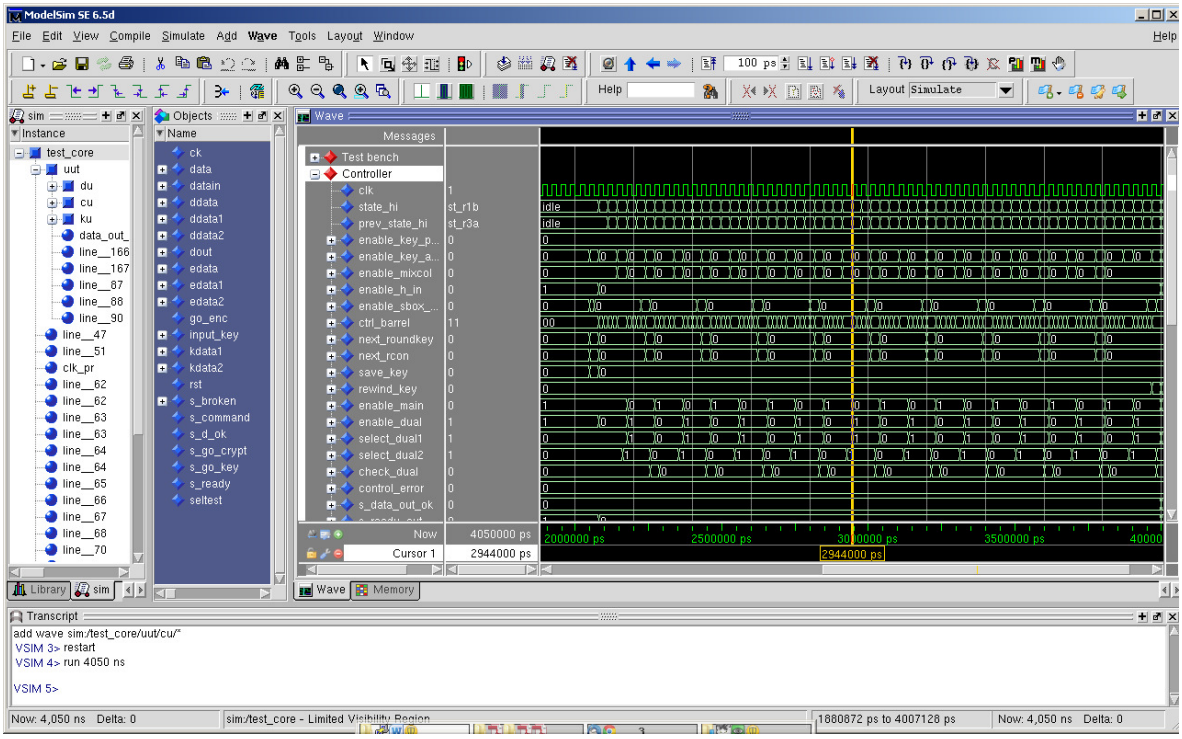
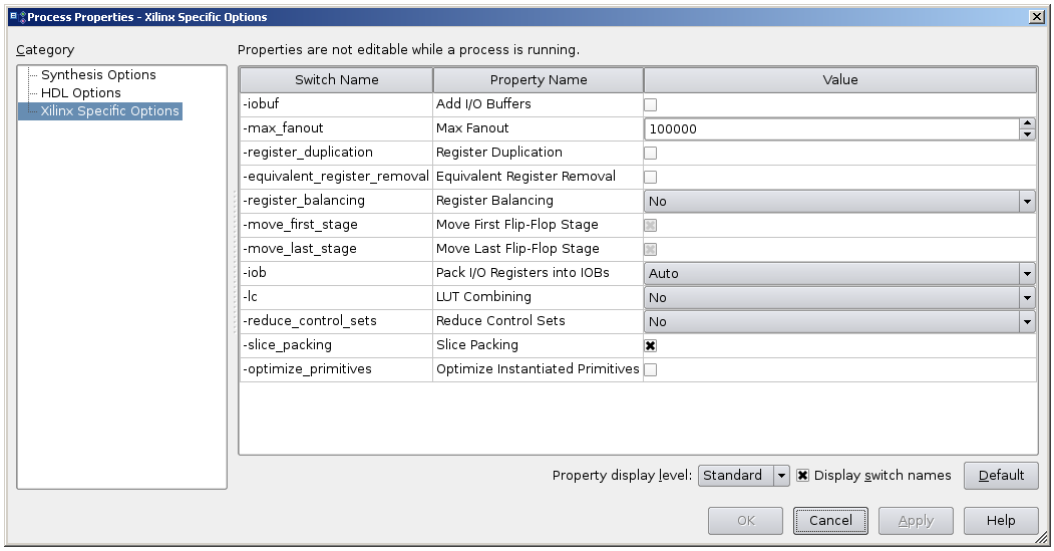


Figure 8: Detailed behavioral simulation in ModelSim

Please refer to the ModelSim section for further details.

8. **Synthesis.** The synthesis process can be started. Be sure to select *Synthesis/Implementation* in the drop down menu and the AES core in the file browser. Hence, start the synthesis process. This can be accomplished in several way: double clicking the *Synthesize – XST* command, right clicking it and selecting *Run*, or clicking on the appropriate button in the toolbar. Remember to uncheck the embedding of I/O buffers, Register duplication, and Equivalent Register Removal in the Properties of the Synthesis Process as shown below:



The synthesis process starts. If the compilation does not succeed, the summary window lists the number of errors; the user must examine the extended synthesis report (or the output window), look for any error in the source code and fix it before launching the synthesis again. If there are no errors, then a green tick appears near the *Synthesize* command and the summary windows reports the results: device utilization rate, number of warnings, possible errors and auxiliary information. Now it is possible to proceed to verify the synthesized design by simulation.

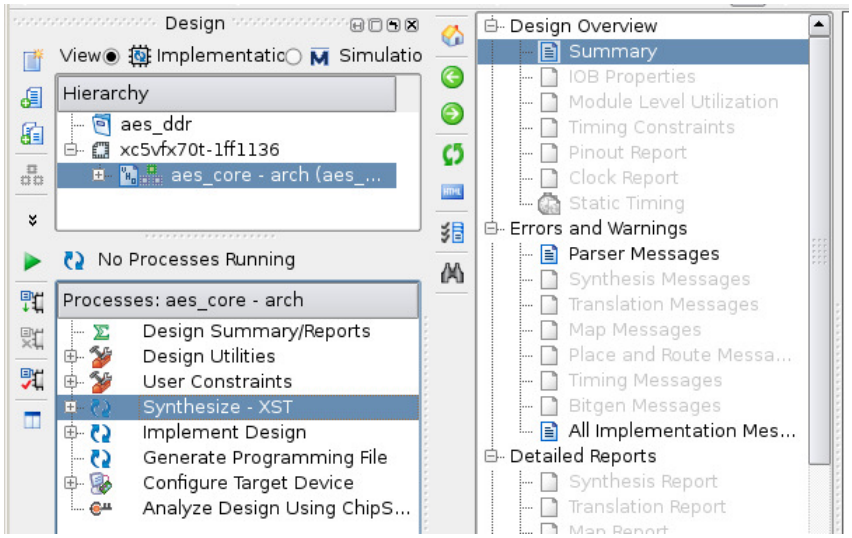


Figure 9: Starting the synthesis process

9. **Post-synthesis verification.** If the synthesis completed successfully, then it is advisable to simulate the result, in order to verify that it did not alter the expected timing of the signals. This requires that the design is synthesized and translated, and the simulation model is generated after the translation step (Figure 10). The simulation of the post-translate model can be started by selecting *Post-Translate Simulation* in the drop-down menu, and then running the command *Simulate*…. In the project of the first session, open the *Process Properties* window of the *Simulate* command, and select the custom script file as in Figure 11. Verify that your design behaves as expected even after synthesis. Note that the representation, or the existence itself, of some signals has changed.

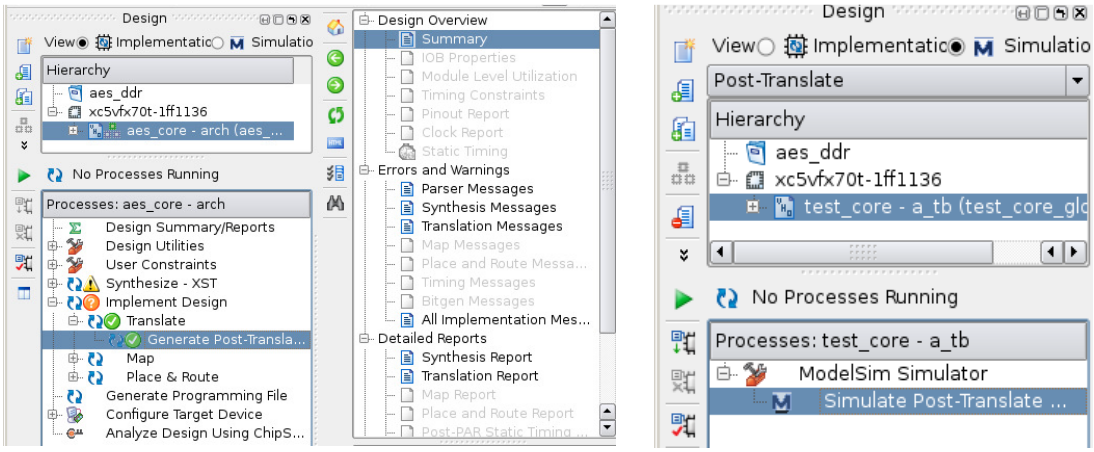


Figure 10: Starting the post-synthesis simulation

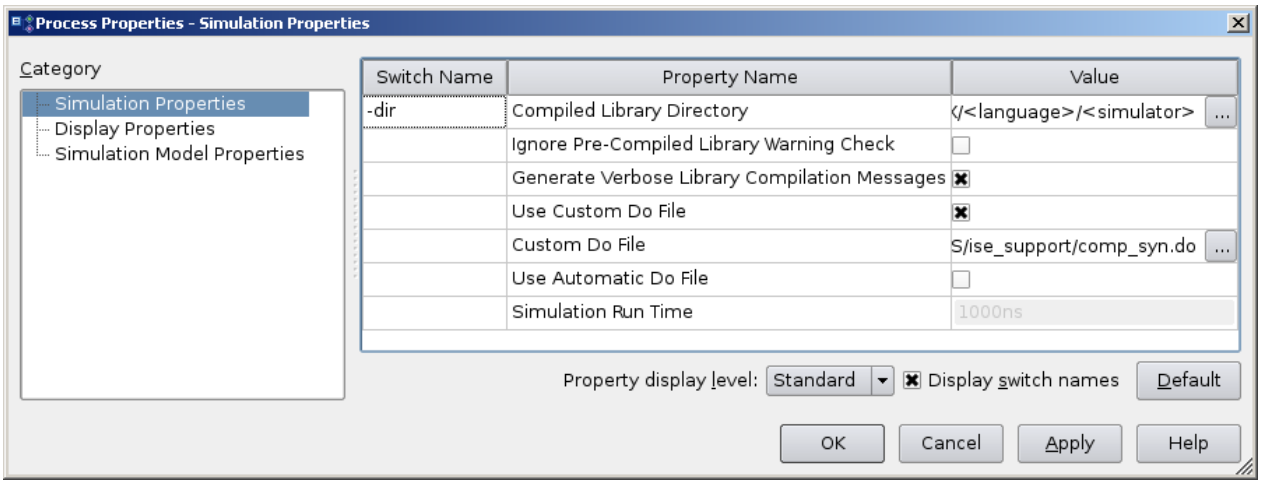


Figure 11: Parameters for the Post-Translate Simulation

10. **Mapping, Placing and Routing.** After synthesis, the FPGA workflow can be completed by running the *mapping* process and then running the *Place & Route* phase. This produces the actual bit stream that will be sent to the device board during the programming phase. Both phases can be verified by generating the appropriate simulation model and starting the simulator. The methodology is the same described for the post-synthesis simulation, provided that the correct commands are selected. However, in order to embed the device core in the EDK design flow, this is **not** required.

    Note. The synthesis converts HDL code into a gate-level netlist (represented in the terms of the *UNISIM* component library, a *Xilinx* library containing basic primitives). The implementation stage is intended to translate netlist into the placed and routed FPGA design. During the translate phase the NGC/EDIF netlist is converted to an NGD netlist. The difference between them is in that NGC netlist is based on the UNISIM component library, designed for behavioral simulation, and NGD netlist is based on the SIMPRIM library. The netlist produced by the NGDBUILD program contains some approximate information about switching delays. During the map phase the SIMPRIM primitives from an NGD netlist are mapped on specific device resources: LUTs, flip-flops, BRAMs and other. The output file contains precise information about switching delays, but no information about propagation delays, since the layout hasn't been processed yet. Finally, Place and route defines how device resources are located and interconnected inside an FPGA. It is the most important and time consuming step of the implementation.

    NOTE: The mapping, placing and routing phase are not required when embedding the synthesized core in an EDK project. More precisely, these steps can be completed only when IO ports of the top module are buffered, which is in contrasts with the integration of a netlist within EDK; **on the other hand, this option must NOT be selected in the *Synthesis Properties* window when synthesizing the core for EDK**.