2022

Low-Level HW Digital Systems II

LAB MANUAL & COURSEWORK

VASILIS F. PAVLIDIS – ASSOC. PROFESSOR, ECE DEPT.

Contents

1 Introduction 2
2 Simulation Tools 2
2.1 Questa – Intel FPGA Edition 2
3 Coursework 4
3.1 Setting up the Simulation Environment 4
3.1.1 Design under test 5
3.1.2 Simulation tool 7
3.2 Coursework Exercises 8
3.2.1 Up and Down Counter 8
3.2.2 Simple synchronous FIFO 9
4 Deliverables for Submission 10
5 References 11
6 Appendix 11

1 Introduction

This document describes briefly the prerequisites required to set up the tools for compiling, simulating, and verifying the circuits you will design as part of the coursework for this course. The document is divided into two sections. The following section includes some guidelines regarding the simulators you can use and some concise instructions on how to install and use a simulator offered for free for Intel's FPGAs. Section 3 describes the simulator set up, the coursework you need to complete and the deliverables.

2 Simulation Tools

A well-established tool for RTL simulation has been the Modelsim simulator [1] of Mentor-Graphics (now part of Siemens). Unfortunately, the student edition of this great tool is not available any longer. Thus, you need to seek another simulator that effectively have integrated Modelsim in their environment. There are few available options, which you can use for free for student projects. These options include, for example, the Quartus tool from Intel [2] and Libero® SoC Design Suite from Microchip [3].

We recommend the use of Quartus provided on [2], which supports both a Windows and a Linux version. Unfortunately, there are no simulation tools that we know of and presently support Mac OS. Thus, Mac users need to find a work around. In the following, we succinctly describe how to install and use the Intel simulator. Note that we cannot offer you any technical support on this topic (I managed to get the tool up and running, hence, I am confident you will have no problems either).

2.1 Questa – Intel FPGA Edition

Follow the steps below to install the Quartus simulator for RTL simulation. Alternatively, you can use any other tools you like (even the one you used in the last semester but make sure that all work fine).

- Go to [2], click on Individual files in the Download Section and select to download the Questa*-Intel® FPGA Edition. Please check you have sufficient disk space to proceed with the installation. Although the tool requires about 1 GB during the installation process you may need much more (possibly up to 30 GB [4]). Also, check the memory requirements of the simulator (as with any other tool).
- During the installation process, you SHOULD select Intel FPGA **Starter** Edition as only this edition offers a free license for 1 year.
- Once the tool installation is complete, a process that takes quite long (tens of minutes depending on your computer), you need to acquire a license file. Please visit the Intel® FPGA Self Service Licensing Center (SSLC) and sing up using your **institutional email** (please do not use your personal email, e.g., gmail).
- After registration, you will receive a confirmation email from Intel. Using this email, log on SSLC and click on the "Sign up for Evaluation or Free Licences" tab and select Questa*-Intel® FPGA Starter Edition, as depicted in Fig. 1, where you edit the number of seats to one. Click on "Get license".

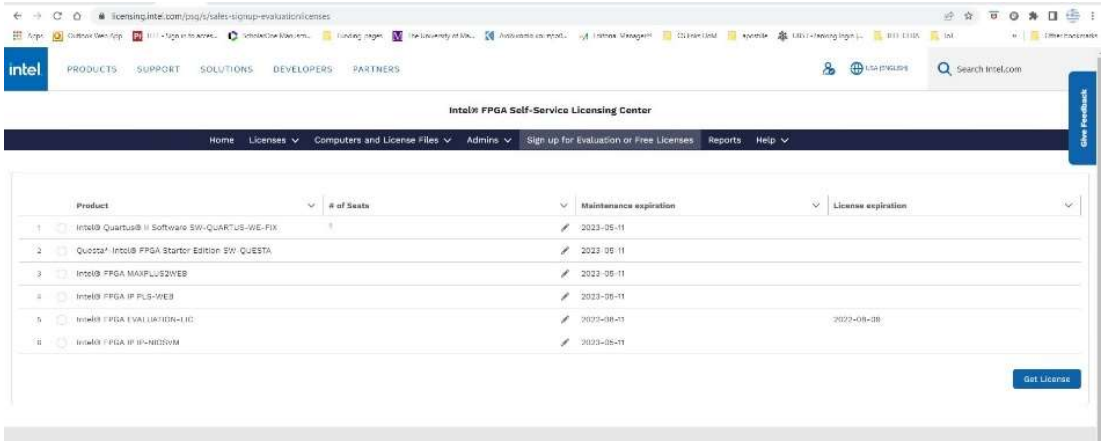

Fig. 1. SSCL Screen

- You will need to provide the NIC of your computer (physical or MAC address of your Ethernet or wireless card, which consists of 12 hex digits) that is used identify your computer (execute `ipconfig -all` on the Windows command prompt to obtain this hex number). Please note that you can support up to 3 devices with the same license file. In this case, you need to install the software on all three machines. In addition, note that you cannot run the software remotely but only locally with this type of free license.
- Typically, it takes some time (nominally up to 12 hrs) to register your account on SSCL, although you have received the activation email. So, allow some time for this to happen before you attempt to check out a license.
- By selecting the proper type of license, you will receive an email where the license file with the extension .dat will be attached. Save this file to a desired location, typically, some folder within the top level directory `intelFPGA`. This file is used by the license daemon of the tools, located in the path C:\intelFPGA\21.1\questa_fse\win64.
- An environment variable should also be added to the system. For Windows, right click on Start, select System, Advanced System Settings, Environment Variables, and create a New variable called MGLS_LM_LICENSE_FILE and provide the path where you stored the license file you received from Intel (*.dat). Click OK. An example of this setup is shown in
- You can now start using the software. Also use the Intel® FPGA Software Installation and Licensing manual available on the elearning course webpage. Note that you are interested in the Intel FPGA **Starter** Edition.

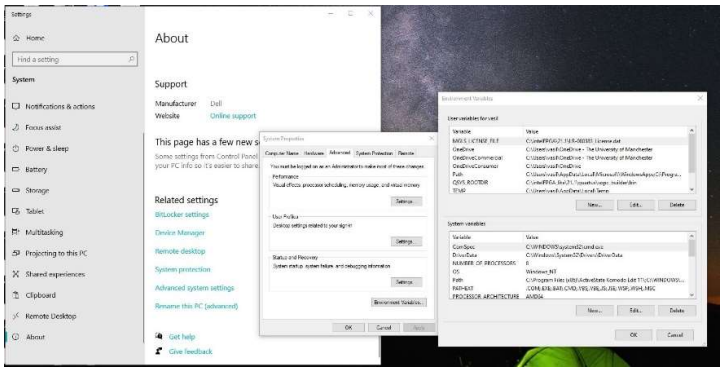

Fig. 2. Setting up the environment library for Questa – Intel FPGA simulator.

3 Coursework

This section describes the tasks that you need to complete as part of the laboratory exercises. An example of a circuit described in SytemVerilog (SV) and tested with typical testbenches and SV Assertions (SVA) is discussed in subsection 3.1. The circuits that you need to design and verify with SV and SVA are presented in subsection 3.2. Each student should individually submit a report (**strictly in pdf format**) with all the deliverables as required in Section 4.

3.1 Setting up the Simulation Environment

The basics of setting up a circuit for functional simulation is first described. Specific steps that you need to follow in the Questa-Intel simulator to examine SVA are also provided. To simulate a design, you need a testbench (similar to the methodology you used in Low-Level HW Digital Systems I in the 7th semester). In addition to this file, another file that contains all the SVA, you have created for the design (DUT) or circuit (CUT) under test, must be included for simulation. This is done through the `bind` command described in the course. This binding process is schematically shown in Fig. 3.

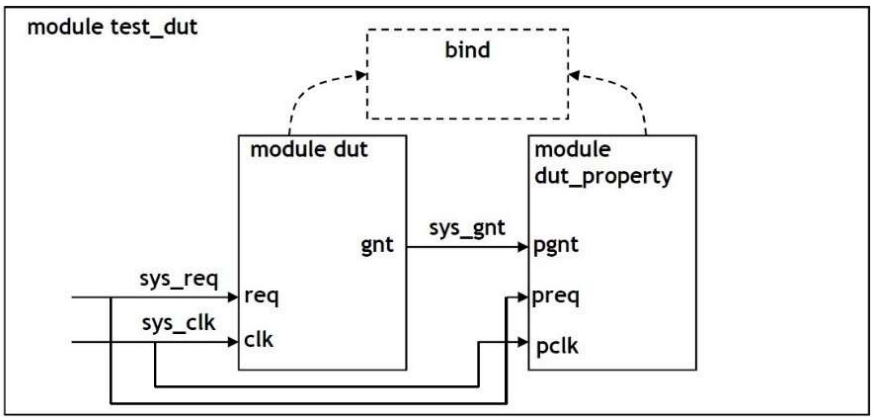

Fig. 3. In the module test_dut, the dut and dut_property modules are bound. The test_dut is the testbench that contains all required information to test the module dut. SVA are contained in the dut_property module.

In the following, the code for each of these three modules is provided such that you understand the methodology of binding modules, using SVAs, and producing an appropriate testbench. Note that you need to read and understand how these files work altogether, using also the examples and material you have been provided during lectures.

3.1.1 Design under test

A simple behavioral Verilog model that acts as the DUT driving a simple req/gnt protocol follows.

```
module dut(clk, req, gnt);
input logic clk,req;
output logic gnt;

initial
begin
  gnt=1'b0;
end
initial
begin
@(posedge req);
  @(negedge clk); gnt=1'b0;
  @(negedge clk); gnt=1'b1;
@(posedge req);
  @(negedge clk); gnt=1'b0;
 @(negedge clk); gnt=1'b0;
end
endmodule
```

The code in the text box below includes different assertions that may be desired to check the functionality of the DUT. Read this code carefully combined with the taught material in the lectures. Note the presence of macros (defined with the `ifdef elsif endif` structure), which can help you choose the property that is checked each time. You may opt to completely ignore such macros. Also, note that any command starting with (`) is a compilation directive rather than part of the RTL code and, hence, of the DUT.

```
module dut_property(pclk,preq,pgnt);
input pclk,preq,pgnt;

`ifdef no_implication
property pr1;
  @(posedge pclk) preq ##2 pgnt;
endproperty
preqGnt: assert property (pr1) $display($stime,,,"\t\t %m PASS");
          else $display($stime,,,"\t\t %m FAIL");

`elsif implication
property pr1;
  @(posedge pclk) preq |-> ##2 pgnt;
endproperty

preqGnt: assert property (pr1) $display($stime,,,"\t\t %m PASS");
          else $display($stime,,,"\t\t %m FAIL");

`elsif implication_novac
property pr1;
  @(posedge pclk) preq |-> ##2 pgnt;
endproperty
preqGnt: assert property (pr1) else $display($stime,,,"\t\t %m FAIL");

property pr2;
  @(posedge pclk) preq ##2 pgnt;
endproperty
cpreqGnt: cover property (pr2) $display($stime,,,"\t\t %m PASS");
`endif

endmodule
```

The binding process and the stimuli used in this example are presented in the following textbox. Invest as much time as required to understand what this code does, how inputs are stimulated and how real-time report messages are produced during the simulation. You can remove the $finish command, as this command pops up a window that asks to quit the tool, not just the simulation.

```
module test_dut;
bit sys_clk,sys_req;
wire sys_gnt;

/* Instantiate 'dut' */
dut dut1 (.clk(sys_clk),.req(sys_req),.gnt(sys_gnt));

bind dut dut_property dutbound (clk,req,gnt);
//
// You need to know the names of the ports in the design and the property module
// to be able to bind them. So, here they are:
// Design module (dut.v)
// ----------------------
// module dut(clk, req, gnt);
//          input logic clk,req; //          output logic gnt;
// Property module (dut_property.sv)
// --------------------------------
//module dut_property(pclk,preq,pgnt); //input pclk,preq,pgnt;

always @(posedge sys_clk)
  $display($stime,,,"clk=%b req=%b gnt=%b",sys_clk,sys_req,sys_gnt);
always #10 sys_clk = !sys_clk;

initial
begin
      sys_req = 1'b0;
  @(posedge sys_clk) sys_req = 1'b1; //30
  @(posedge sys_clk) sys_req = 1'b0; //50
  @(posedge sys_clk) sys_req = 1'b0; //70
  @(posedge sys_clk) sys_req = 1'b1; //90
  @(posedge sys_clk) sys_req = 1'b0; //110
  @(posedge sys_clk) sys_req = 1'b0; //130

@(posedge sys_clk);
@(posedge sys_clk); $finish(2);
end
endmodule
```

3.1.2 Simulation tool

You can always compile and simulate these files using the Questa Intel HDL simulation tool as you did in the coursework of in Low-Level HW Digital Systems I and, therefore, no detailed description will be provided. In general, you need to

- Create a new project (give a name of your preference) and fill in the fields on the displayed window.
- Add existing files or create a new file through the tool editor. When you add existing files, think whether you want to copy the files to the working directory (easier for command line reference to the files) or keep them in the reference directory (Fig. 6 and Fig. 7 in Appendix).
- Once you have successfully compiled all related files (Fig. 8 and Fig. 9 in Appendix), you are ready to start the simulation. Click on Simulate -> Start Simulation. On the window that

pops up, expand "work" library and pick the files you want to simulate as shown on the related figure below (Fig. 10 in Appendix).

- Click on the optimization options and select full (or custom) visibility (Fig. 11 in Appendix) that allows to retrieve all signals in the design hierarchy to view these signals on the waveform window. Select the component dut and add its signals to waveform (Fig. 12 in Appendix).
- Click on the Run icon and you can simulate your circuit for 100 ns or, whichever, other option you prefer (Fig. 13 in Appendix).
- To activate the properties since these are encapsulated within macros, you need to explicitly define these arguments (one or more) during the compilation of the dut_property.sv file. The compile command (vlog) is of the form:

```
vlog -sv dut_property.sv +define+no_implication
```

- The argument "-sv" means compile with SystemVerilog options and "+define" defines a macro, in this case the "no_implication" property. More macros can be simultaneously defined by adding more macros to the command. The tool returns a warning in this case that multiple macros are defined but you can safely ignore this.
- Restart the simulation (not the tool!) and you will see that some assertions now appear. Information on assertions can be found by clicking on the Assertions tab right at the bottom of the waveform subwindow (Fig. 14 in Appendix).
- **Tips**: type help <command> in the command line of the simulator to get help about any command. The simulator also provides auto-complete on any command or path. Use the Up arrow in the Transcript window to see the history of executed commands as you proceed with different tasks through GUI or typed commands.
- Click Help -> Questa sim – Intel FPGA Edition – Bookcase to find a rich set of manuals. Use these with caution and at your own discretion as they contain hundreds of pages!

3.2 Coursework Exercises

You are required to design with SV and verify with a testbench and SVA two different circuits as presented in the following subsections. For each of these two circuits you must,

- Produce the SV code
- Test the code correctness through a testbench
- Verify correct functionality of the circuit with SVA as requested for each circuit.

3.2.1 Up and Down Counter

This is simple counter circuit (16-bit) that increments or decrements at each clock edge. The block diagram of the circuit is shown in Fig. 4. All the inputs and outputs are shown. These should be defined as the ports of the top level module of your design. Any discrepancy in the port names from those provided to you will result in mark loss.

The specifications of the counter are:

- The counter has 16-bit input and output data.
- When ld_cnt is asserted (active Low), the input data is loaded and outputs this data.
- When count_enb (active High) is asserted

    - if updn_cnt is **high**, count up
    - if updn_cnt is **low**, count down
- When count_enb is low, the counter maintains its data stable.
- A load operation has higher priority over count_enb
- The counter operates on a positive edge clock and has an asynchronous active low reset.

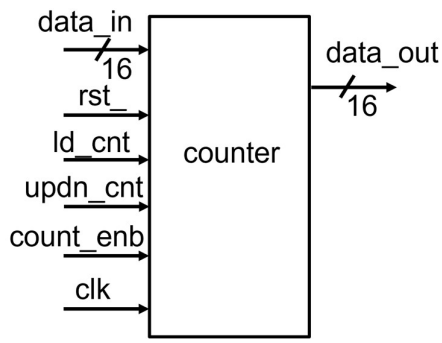

Fig. 4 Top level diagram of the 16-bit up/down counter.

You also need to check about the following scenarios through SVA:

1. When reset is asserted, then the output of the counter is zero
2. If ld_cnt is deasserted and count_enb is not enabled the counter output does not change
    a. Reasonably, this property should be disabled when rst is active (low)
3. If ld_cnt is deasserted and count_enb is enabled then if updn_cnt is high, the counter increments and if updn_cnt is low, the counter decrements
    a. Reasonably, this property should be disabled when rst is active (low).

3.2.2 Simple synchronous FIFO

The second circuit that you have to design is a simple synchronous FIFO, the width and depth of which should be parameterized. In your implementation, the width and depth should be 16. The block diagram of the circuit is shown in Fig. 5. All the inputs and outputs are shown. These should be defined as the ports of the top level module of your design. Any discrepancy in the port names from those provided to you will result in mark loss.

The specifications of the FIFO memory are:

- The behavior of the memory is, effectively, controlled by a write and read pointer, denoted as wr_ptr and rd_ptr, respectively.
- The write pointer increments by 1, each time there is write request and the FIFO is not full
- The read pointer increments by 1, each time there is read request and the FIFO is not empty
- The memory contains one counter (cnt) that increments on a write event and decrements on a read event. This internal counter is used to flag the following conditions:
    - When FIFO is full, fifo_full should be asserted
    - When FIFO is empty, fifo_empty should be asserted
    - Both of these condition signals should be active high
- The FIFO operates on a positive edge clock and has an asynchronous active low reset.

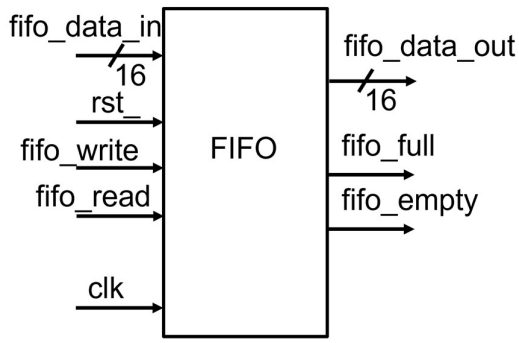

Fig. 5 Top level diagram of the synchronous FIFO.

You also need to check about the following scenarios through SVA:

1. On reset, the read and write pointers are zero, empty and full indicators are '1' and '0', respectively (since the memory is indeed empty), and the counter is zero as well.
2. The fifo empty signal is asserted whenever cnt = 0. Disable this property if the reset is active.
3. The fifo full signal is asserted whenever cnt >= 16. Disable this property if the reset is active.
4. If the FIFO is full and there is write request (but not read), then the write pointer should not change.
5. If the FIFO is empty and there is read request (but not write), then the read pointer should not change.

4 Deliverables for Submission

For each of the circuits discussed in Section 3.2, you SHOULD:

- Produce the SystemVerilog code (.sv file)
- Produce a testbench that demonstrates the correct operation of the circuit (.sv file)
- Using SVA, provide the necessary properties that produce proper messages on the simulator Transcript window (std out) to report on the PASS/FAIL of the additional checks required. These checks (scenarios) are listed in the corresponding subsection of each circuit (.sv file).

You should prepare a report (in pdf format) that will include all of the above files, screenshots that show the correct functionality of the circuit and as much discussion as required to describe what you have designed and how you have verified these two DUTs. In addition to the report, prepare one folder for each circuit that contains the three related files (RTL code, testbench, SVA) and submit these folders along with the report in a zip file named as *<lastname_AEM>.zip* on elearning.

The coursework takes 2.5 marks out of the total mark (i.e., 25%) and is compulsory. If the coursework is not submitted on time, no marks are given but you can take the exam. In this case, the maximum mark that can be achieved is 7.5. The mark for the coursework is maintained for the next two exam periods ONLY (Sept. 2022, Feb. 2023).

The exact submission deadline will be discussed in the lecture theater and announced on elearning.

Good Luck!!

5 References

[1]. ModelSim [Online]. Available: https://eda.sw.siemens.com/en-US/ic/modelsim/ (accessed on 12/5/22).

[2]. Questa*-Intel® FPGA Edition [Online]. Available: https://www.intel.com/content/www/us/en/software-kit/684216/intel-quartus-prime-lite-edition-design-software-version-21-1-for-windows.html (accessed on 12/5/2022).

[3]. Libero [Online]. Available: https://www.microchip.com/en-us/products/fpgas-and-plds/fpga-and-soc-design-tools/fpga/libero-software-later-versions (accessed on 12/5/2022).

[4]. https://www.intel.com/content/www/us/en/docs/programmable/683472/22-1/minimum-hardware-requirements.html (accessed on 13/5/2022).

[5]. Intel® FPGA Self Service Licensing Center (SSLC) [Online]. Available: https://tinyurl.com/yc8f8es4 (accessed 13/5/2022).

6 Appendix

The following figures help you understand the simulation tool and should be read with subsection 3.1.2.

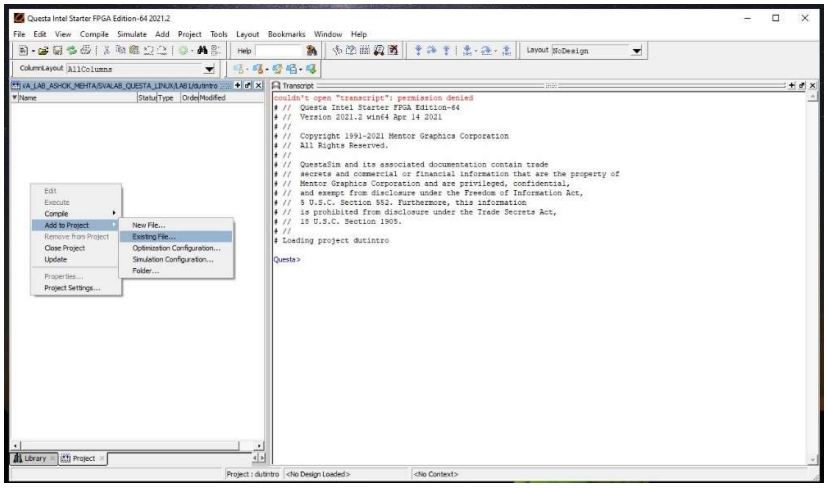

Fig. 6. Started a new project and ready to add files (right click in the project area for these menus to appear).

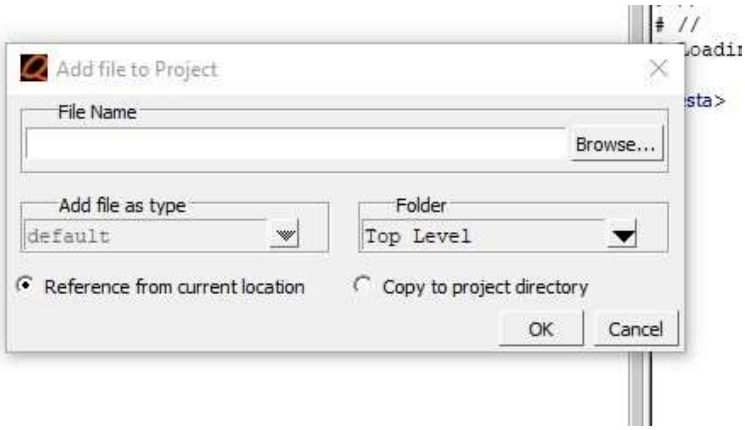

Fig. 7. Browse and pick the files you need (copy or not the files to the project directory).

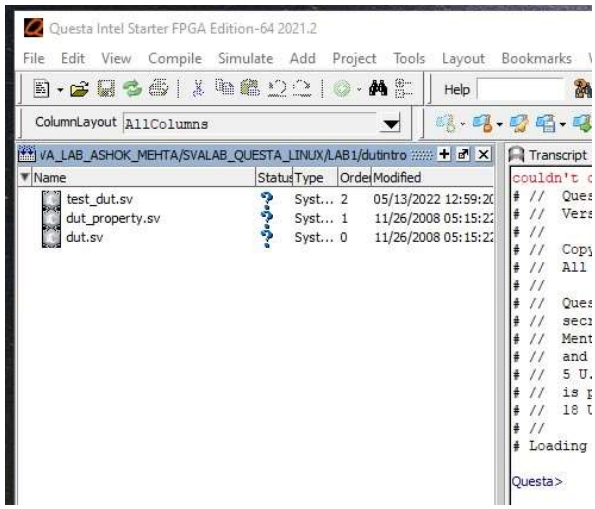

Fig. 8. Files were added to the project and are now ready to be compiled.

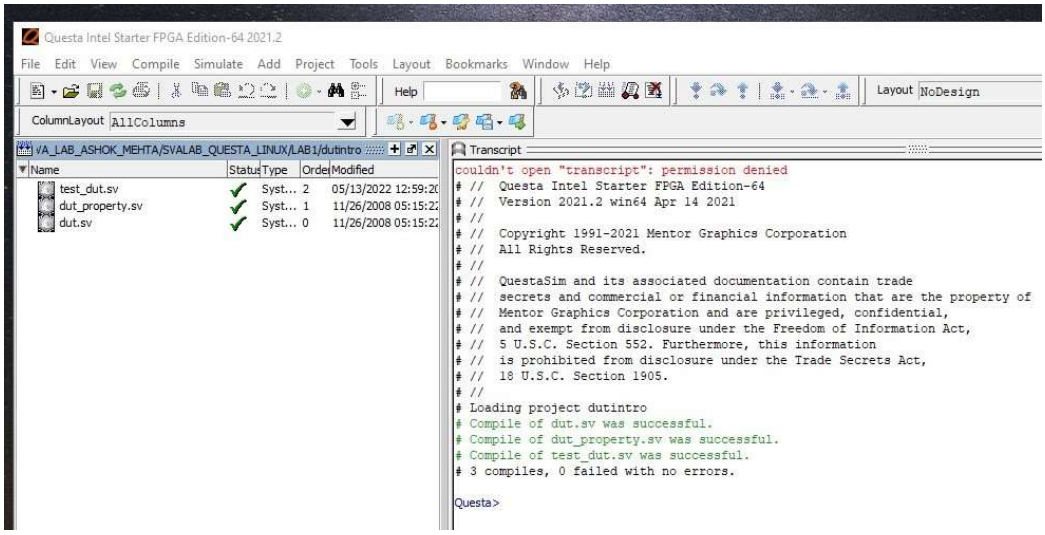

Fig. 9. All files have now been complied (question marks turns to ticks).

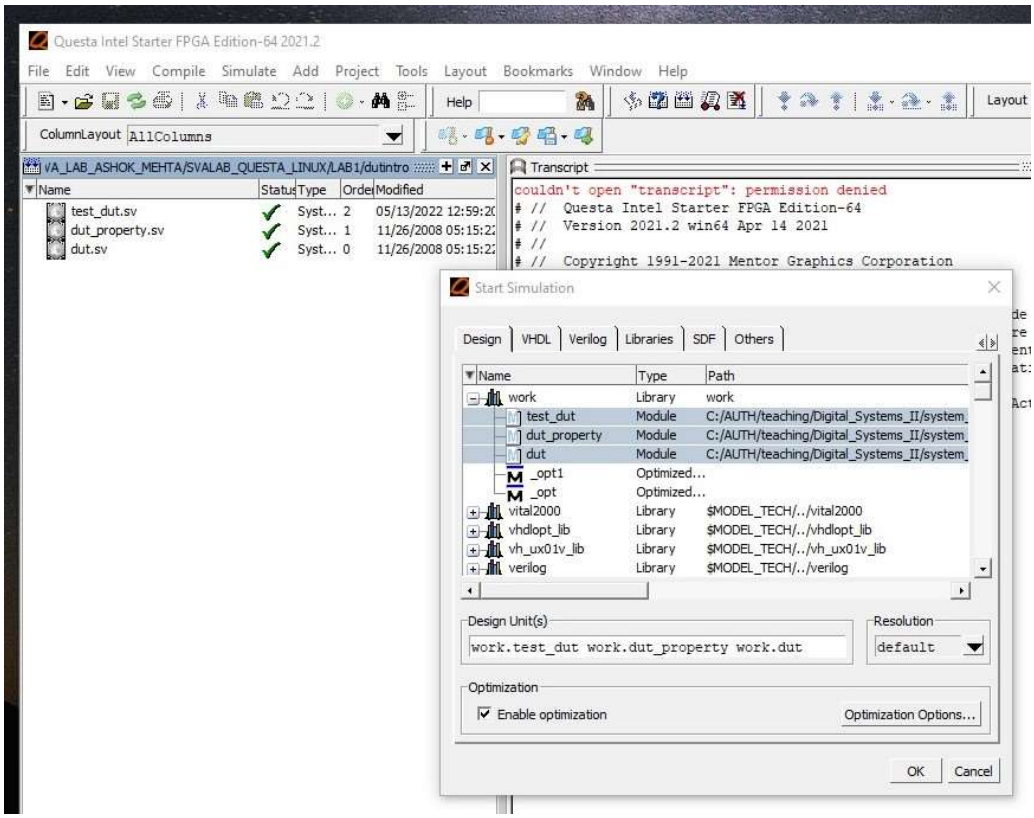

Fig. 10. Click on Simulate -> Start Simulation and select the compiled files to be simulated. Click on optimization options (see next Figure).

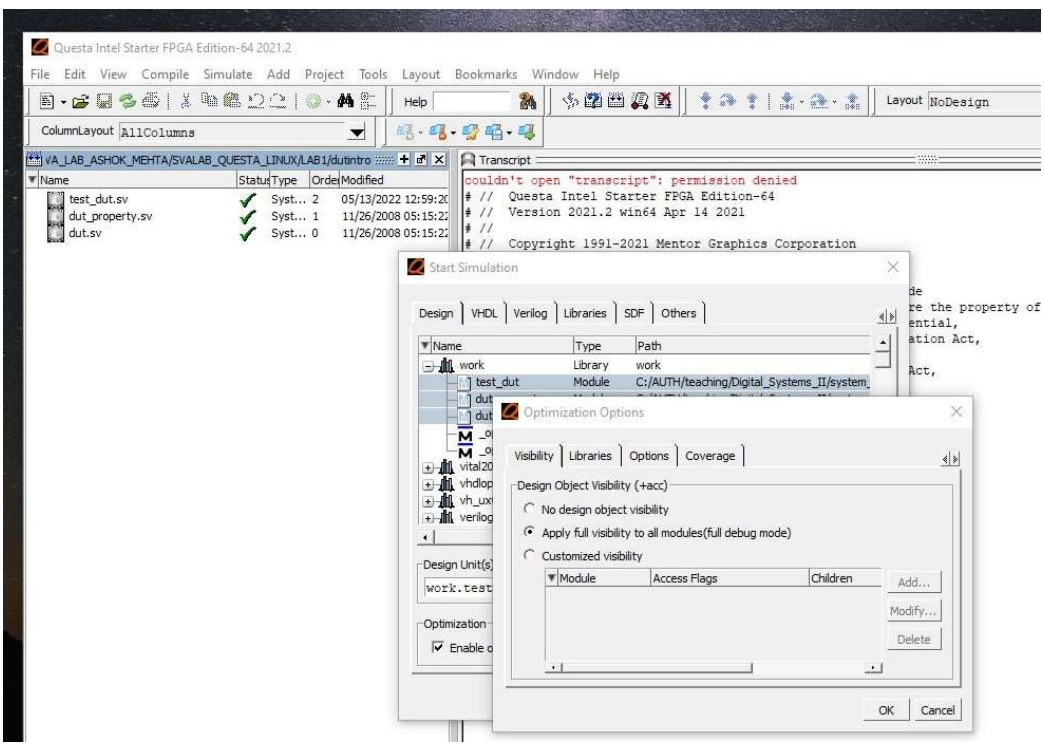

Fig. 11. Full visibility allows you to display all internals signals in the waveform window.

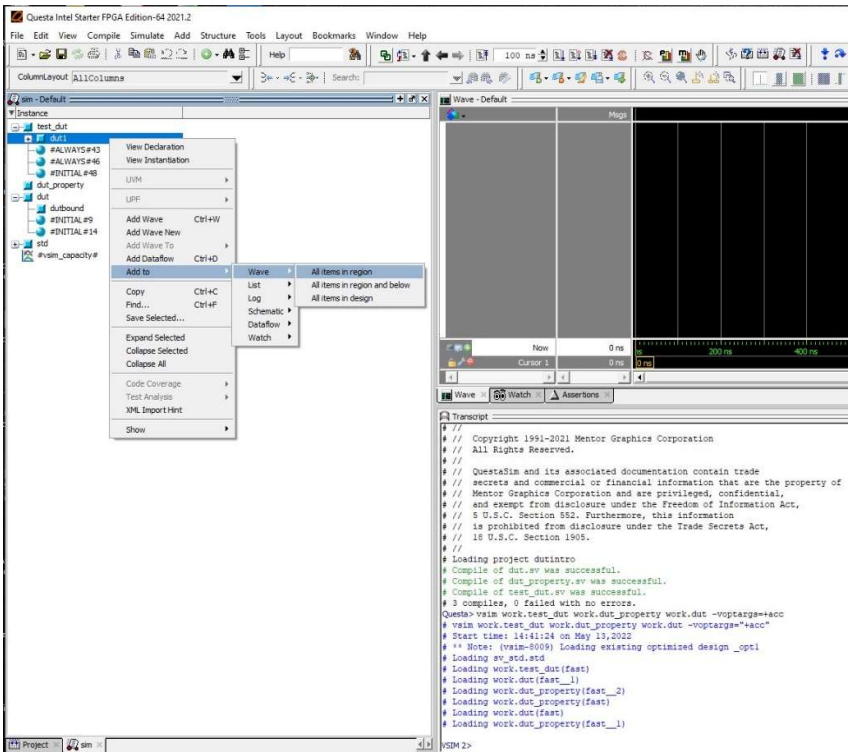

Fig. 12. Once the files are loaded for simulation, you can select which signals to be displayed (right click).

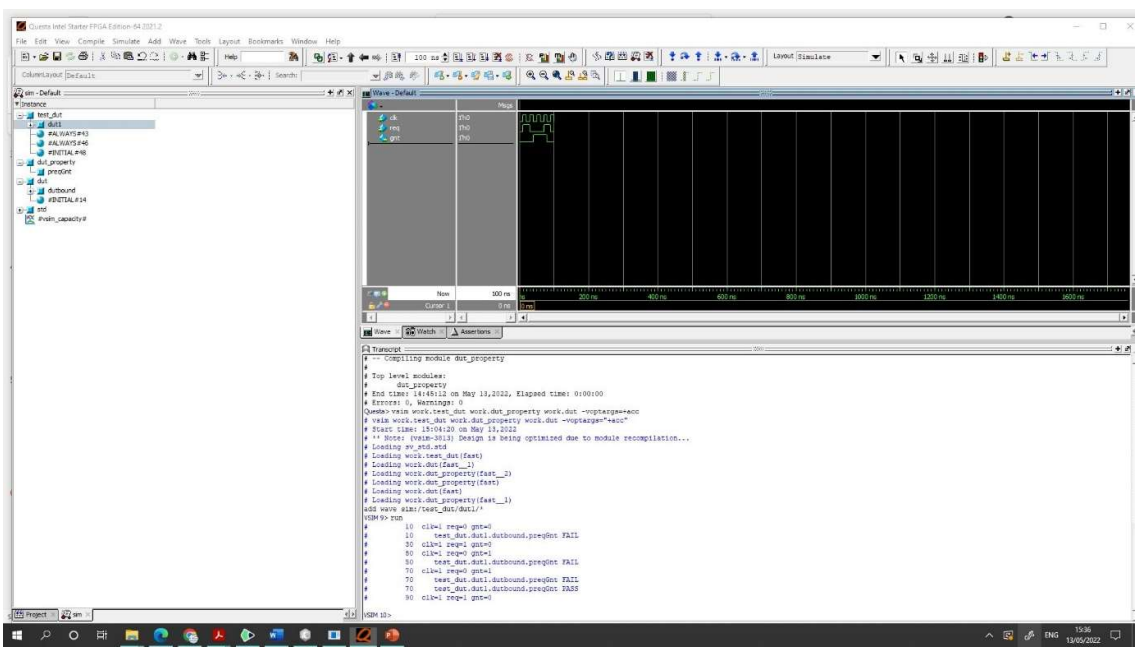

Fig. 13. Simulated DUT. See also the simulator transcript window at the bottom and the related reports.

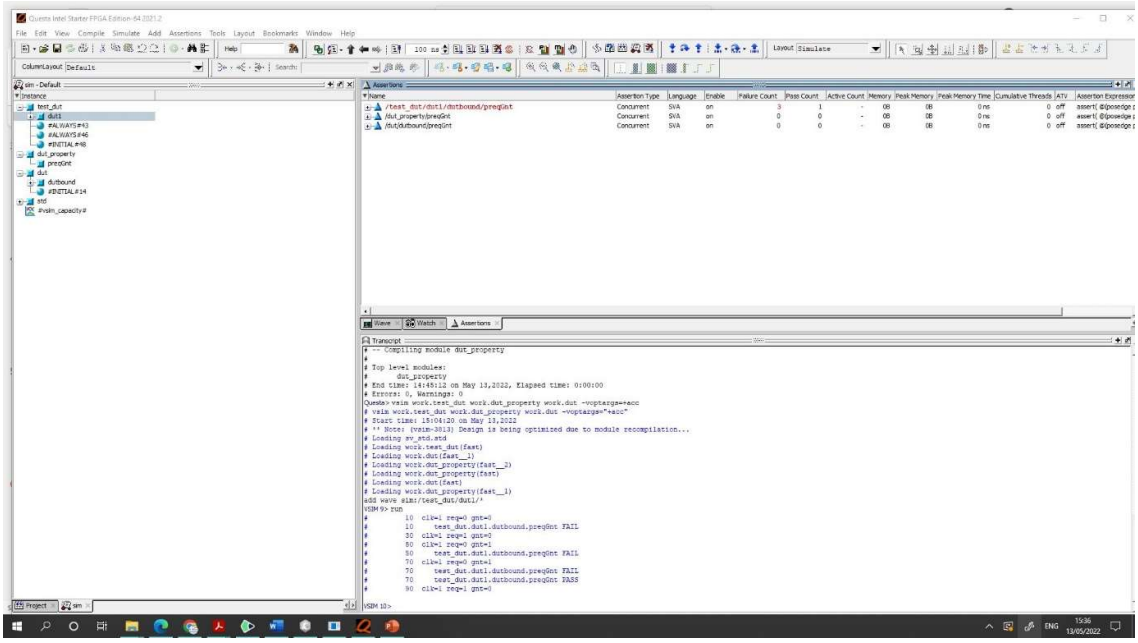

Fig. 14. Simulated DUT. The Assertions tab displays all related info about the properties checked in file dut_property.sv