

Lab #1: Getting Started with VHDL Coding

1 Introduction

In this lab you will learn the basics of the Altera Quartus II FPGA design software through following a step-by-step tutorial, and use it to implement combinational logic circuits described in VHDL. You will also learn the basics of digital simulation using the ModelSim simulation program.

2 Learning Outcomes

After completing this lab you should know how to:

- Run the Intel Quartus software
- Create the framework for a new project
- Design and perform functional simulation of a Binary-to-7-segment LED decoder circuit
- Design a 5-bit adder using VHDL
- Test the adder on the Altera board

3 Run Intel Quartus

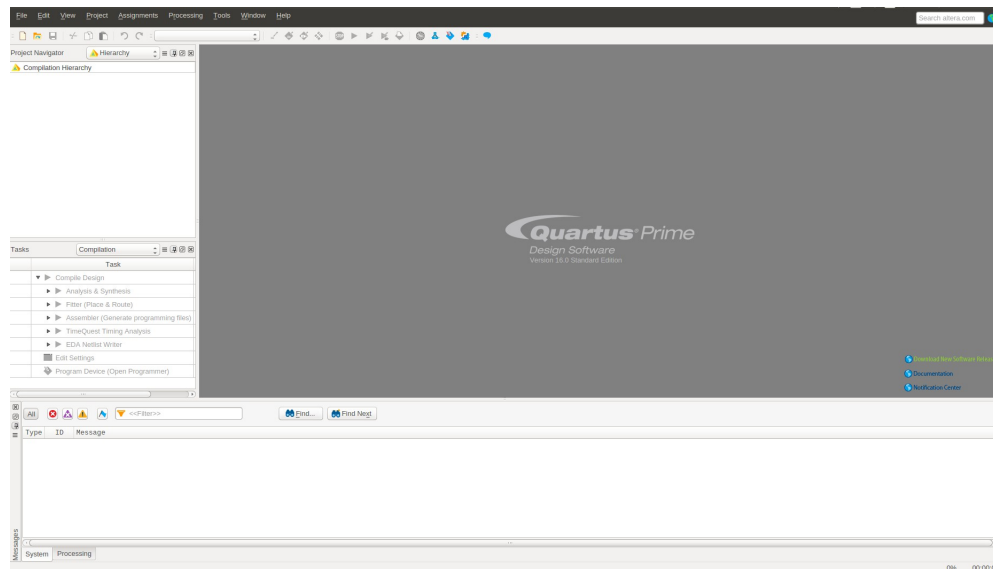
In this course you will be using commercial FPGA design software: the Intel Quartus Prime program and the Mentor Graphics ModelSim simulation program. Quartus Prime and ModelSim are installed on the computers in the lab. You can also obtain a slightly restricted version, the Quartus Lite edition, from the Intel web site¹. The program restrictions will not affect any designs you will be doing in this course. You can (and you should) install the applications on your personal computer to work on your project outside of the lab. *You should use version 18.0 of the program*, as this is the latest version that supports the prototyping board (the Altera DE1-SoC board) that you will be using.

To begin, start Quartus Prime by selecting it in the Windows Start menu:



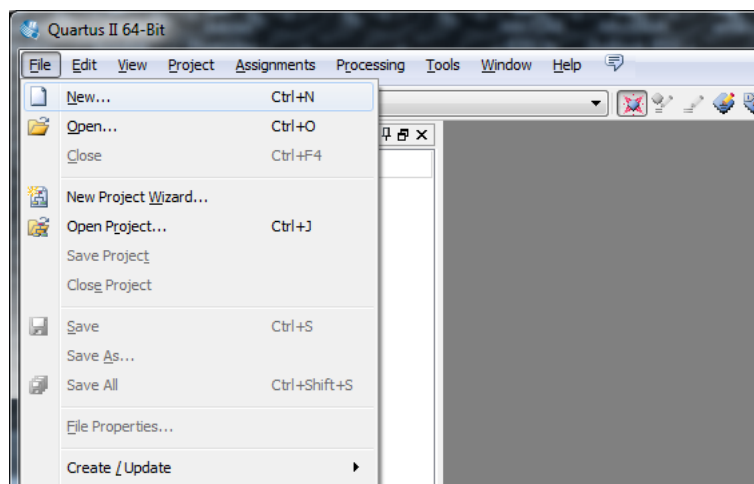
The following window will appear on startup (this shows version 18.0 downloaded from Intel's web site; the versions on the lab computers may look slightly different).

¹<https://www.intel.com/content/www/us/en/programmable/downloads/download-center.html>



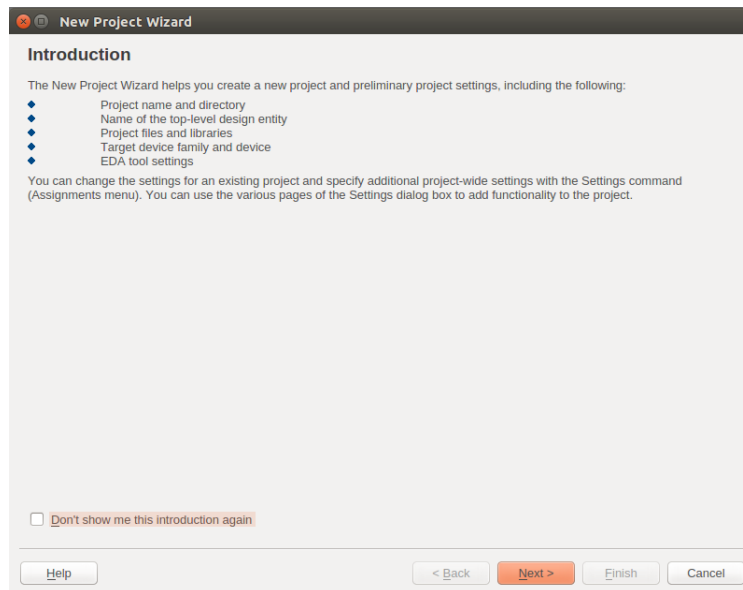
Intel Quartus Prime employs a project-based approach. The goal of a Quartus project is to develop a hardware implementation of a specific function, targeted to an FPGA (Field Programmable Gate Array) device. Typically, the project will involve a (large) number of different circuits, each designed individually, or taken from circuit libraries. Project management is therefore important. The Quartus Prime program aids in the project management by providing a project framework that keeps track of the various components of the project, including design files (such as schematic block diagrams or VHDL descriptions), simulation files, compilation reports, FPGA configuration or programming files, project specific program settings and assignments, and many others.

The first step in designing a system using the Quartus Prime approach is therefore to create the project framework. The program simplifies this by providing a "Wizard" which guides you through a step-by-step setting of the most important options. To run the Project Wizard, click on the File menu and select the New Project Wizard entry.

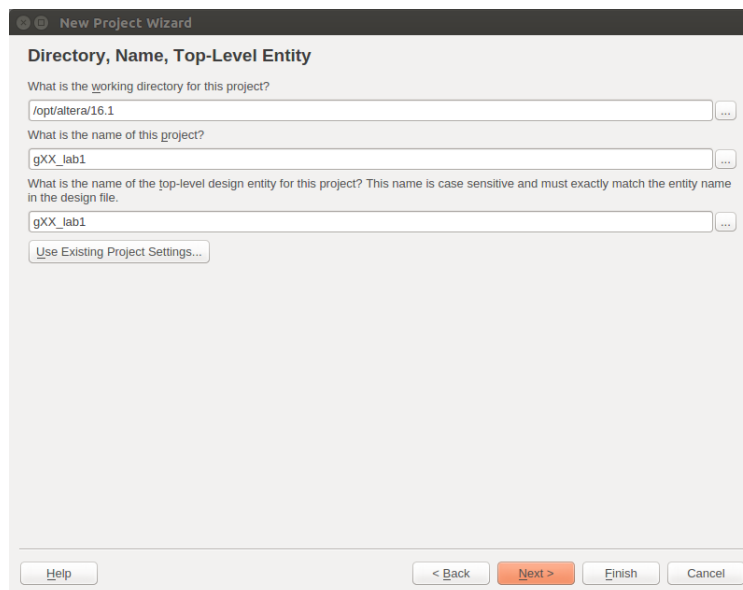


4 Creating a New Project

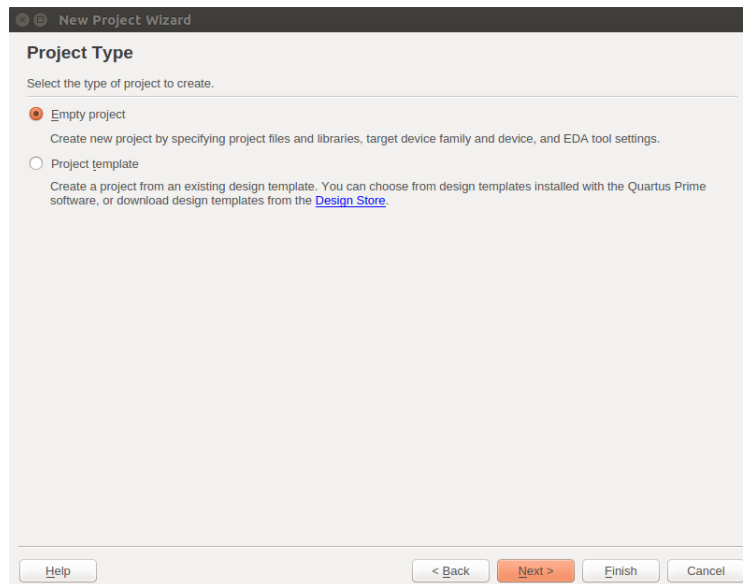
The New Project Wizard involves going through a series of windows. The first window is an introduction, listing the settings that can be applied. After reading the text on this window, click on "Next" to proceed.



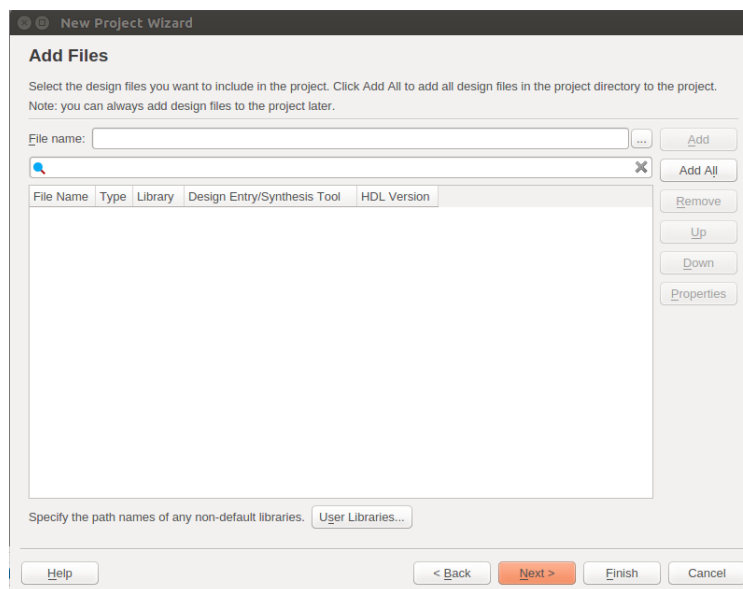
In the second window, you should give the project the following name: gNN_lab1 where NN is your 2-digit group number. The working directory for your project will be different than that shown in the screenshot below. Use your network drive for your project files.



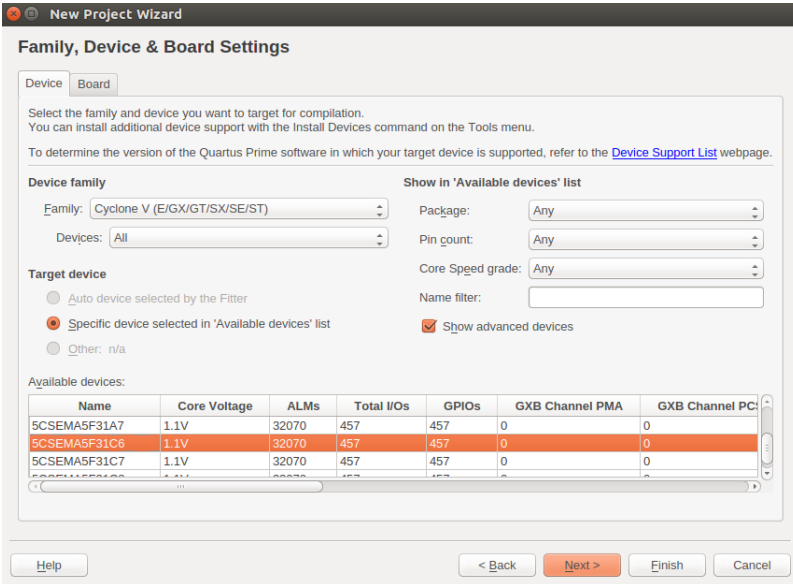
We don't have a project template at this point, so select **Empty project** and proceed.



You will add files later, so for now, just click on "Next".



In this lab, you will be downloading a design to an FPGA device on the DE1-SoC board. These devices belong to the **Cyclone V** family of FPGAs, with the following part number: **5CSEMA5F31C6**. To ensure proper configuration of the FPGAs, select this device as shown below.



Family, Device & Board Settings

Device: Board

Select the family and device you want to target for compilation. You can install additional device support with the Install Devices command on the Tools menu.

To determine the version of the Quartus Prime software in which your target device is supported, refer to the [Device Support List](#) webpage.

Device family

Family: Cyclone V (E/GX/GT/SX/SE/ST) Package: Any

Devices: All Pin count: Any

Target device

☐ Auto device selected by the Filter
☒ Specific device selected in 'Available devices' list
☐ Other: n/a

Core Speed grade: Any

Name filter:

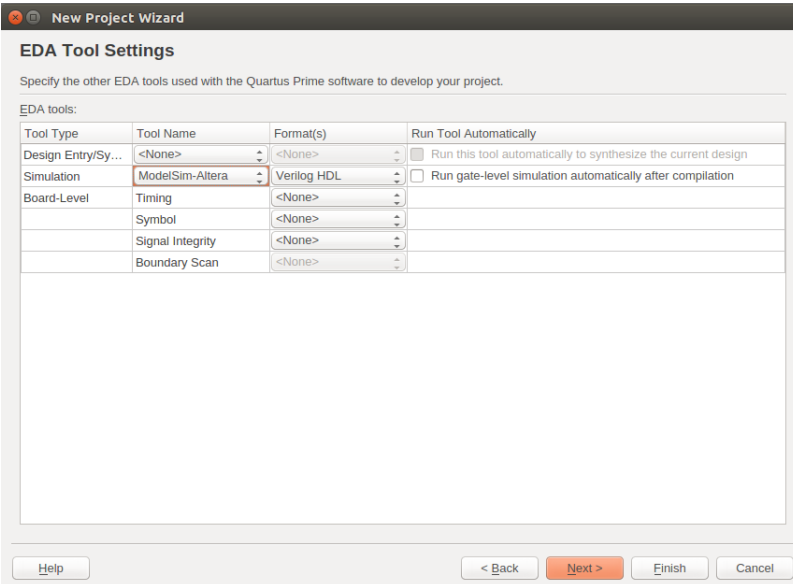
☒ Show advanced devices

Available devices:

Name	Core Voltage	ALMs	Total I/Os	GPIOs	GXB Channel PMA	GXB Channel PC
5CSEMA5F31A7	1.1V	32070	457	457	0	0
5CSEMA5F31C6	1.1V	32070	457	457	0	0
5CSEMA5F31C7	1.1V	32070	457	457	0	0

Help < Back Next > Finish Cancel

The dialog box in the next window permits the designer to specify 3rd-party tools to use for various parts of the design process. We will be using a 3rd-party Simulation tool called **ModelSim-Altera**, so select this item from the Simulation drop-down menu.



EDA Tool Settings

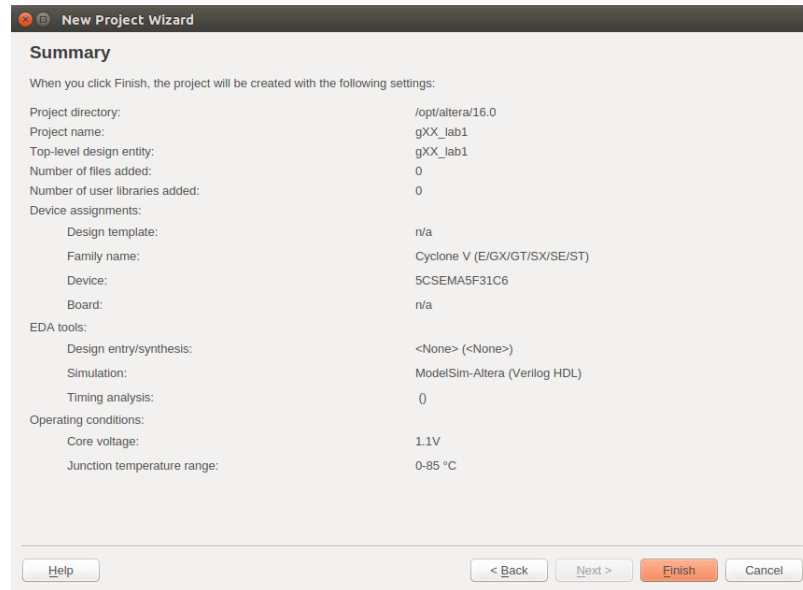
Specify the other EDA tools used with the Quartus Prime software to develop your project.

EDA tools:

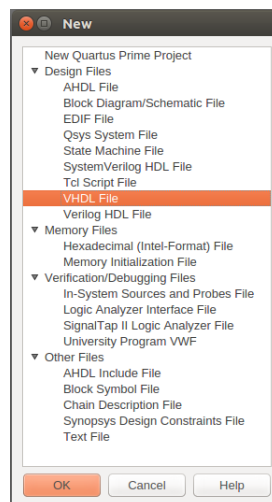
Tool Type	Tool Name	Format(s)	Run Tool Automatically
Design Entry/Synthesis	<None>	<None>	<input type="checkbox"/> Run this tool automatically to synthesize the current design
Simulation	ModelSim-Altera	Verilog HDL	<input type="checkbox"/> Run gate-level simulation automatically after compilation
Board-Level	Timing	<None>	
	Symbol	<None>	
	Signal Integrity	<None>	
	Boundary Scan	<None>	

Help < Back Next > Finish Cancel

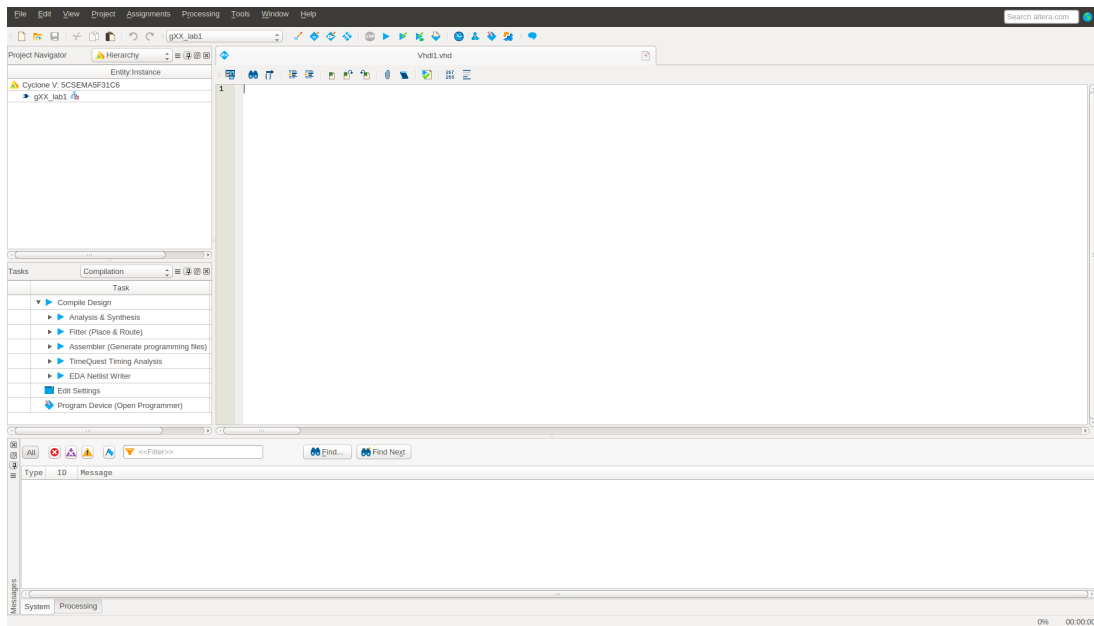
The final page in the New Project Wizard is a summary. Check it over to make sure everything is OK (e.g., the project name, directory, and device assignment), then click **Finish**.



Your project framework is now ready. In **File**, click on **New**, and then select **VHDL file** from the list as shown below.

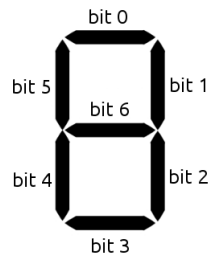


You should have a VHDL editor opened in your framework. You will write and edit your code from this editor.

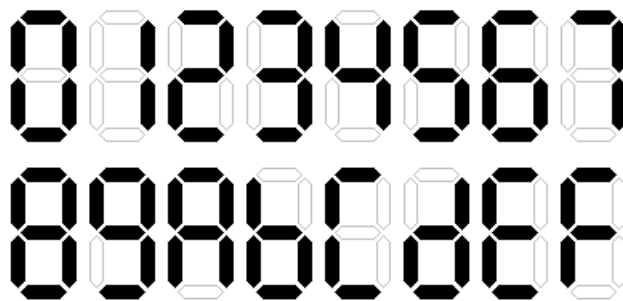


5 Design a Binary to 7-Segment LED Decoder

A 7-segment LED display has 7 individual LED segments, as shown below. By turning on different segments at any one time we can obtain different characters or numbers. There are six of these on the DE1-SoC board, which you will use later in your full implementation of the adder to display the result.



In this part of the lab, you will design a circuit that will be used to drive the 7-segment LEDs on the DE1 board. It takes a 4-bit binary code representing the 16 hexadecimal digits between 0 and F (see figure below) as input, and generates the appropriate 7-segment display associated with the input code. Note that the outputs should be made active-low. This is convenient, as many LED displays, including the ones on the DE1 board, turn on when their segment inputs are driven low. Note that active low means “1” is off and “0” is on.



To implement the 7-segment LED decoder, write a VHDL description using a single selected signal assignment statement. Use the following entity declaration, replacing the NN in `gNN_7_segment_decoder` with your group's number

(e.g., g08).

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity gNN_7_segment_decoder is
    Port (
        code      : in  std_logic_vector (3 downto 0);
        segments   : out std_logic_vector (6 downto 0));
end gNN_7_segment_decoder;
```

6 Simulation of the circuit using ModelSim

Once you have your circuit described in VHDL you should simulate it. The purpose of simulation is generally to determine:

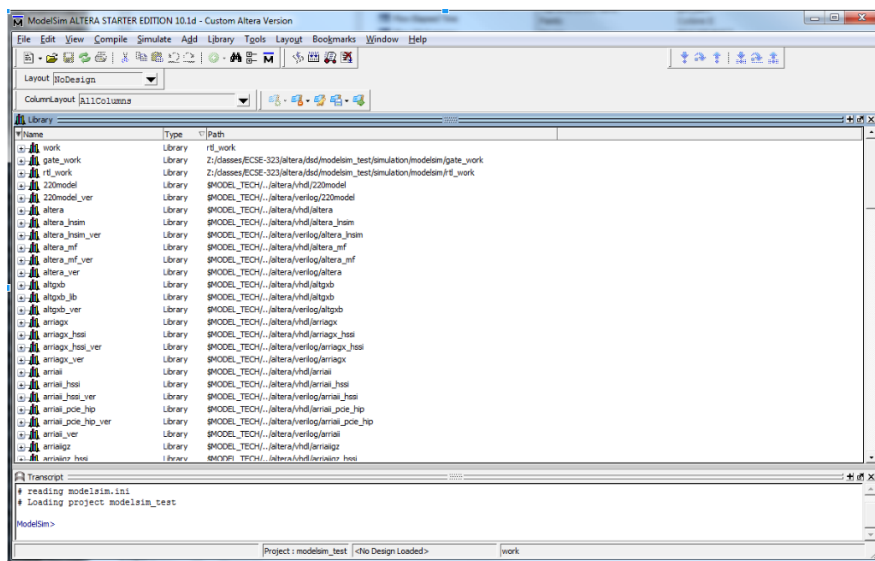
1. if the circuit performs the desired function, and
2. if timing constraints are met.

In the first case, we are only interested in the functionality of our implementation. We do not care about propagation delays and other timing issues. Because of this, we do not have to map our design to a target hardware. This type of simulation is called functional simulation. This is the type of simulation we will learn about in this lab.

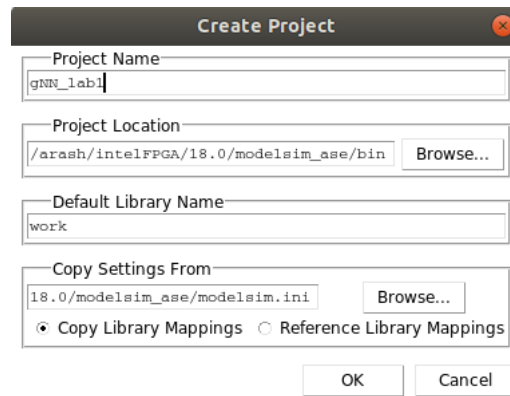
The other form of simulation is called timing simulation. It requires that the design be mapped onto a target device, such as an FPGA. Based on the model of the device, the simulator can predict propagation delays, and provide a simulation that takes these into account. Thus, the timing simulation may produce results that are quite different from the purely functional simulation.

In this course, you will be using the ModelSim simulation software, created by the company Mentor Graphics (actually you will use a version of it specific to Quartus, called Modelsim-Altera). The Modelsim software operates on an Hardware Description Language (HDL) description of the circuit to be simulated, written either in VHDL, Verilog, or System-Verilog. You will use VHDL.

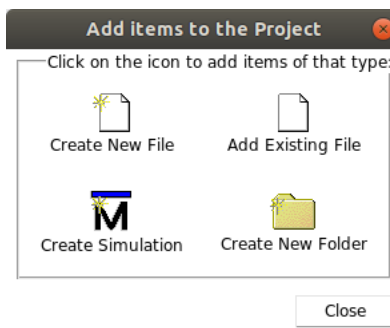
Double-click on the ModelSim desktop icon to run the ModelSim program. A window similar to the one shown below will appear.



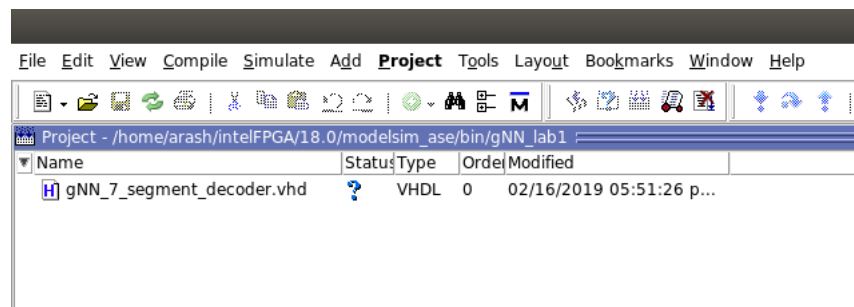
Select **FILE>New>Project** and, in the window that appears, give the project the name gNN_lab1.



Once you click OK, another dialog box will appear allowing you to add files to the project. Click on “Add Existing File” and select the VHDL file that was generated earlier (gNN_7_segment_decoder). You can also add files later.

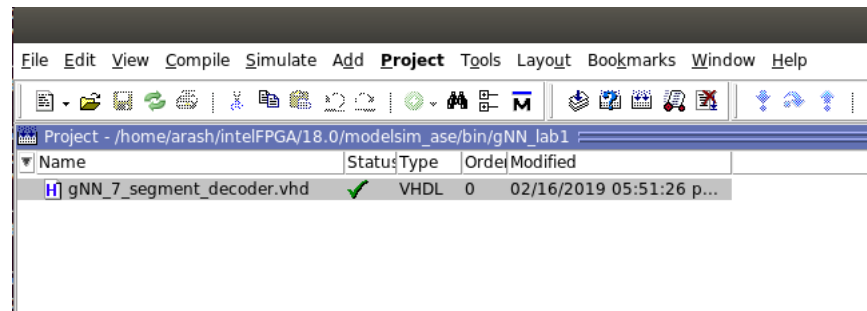


The ModelSim window will now show your VHDL file in the Project pane.

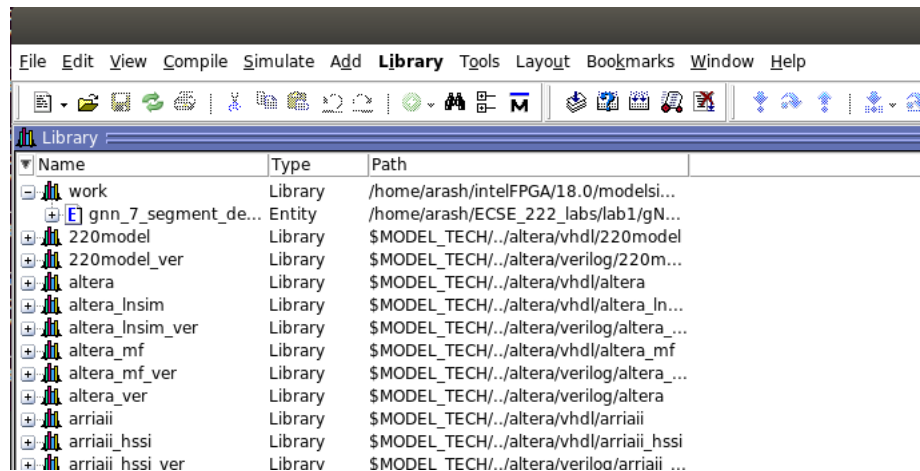


To simulate the design, ModelSim must analyze the VHDL files, a process known as compilation. The compiled files are stored in a library. By default, this is named “work”. You can see this library in the “library” pane of the ModelSim window.

The question marks in the **Status** column in the Project tab indicate that either the files have not been compiled into the project or the source file has changed since the last compilation. To compile the files, select **Compile > Compile All** or right click in the Project window and select **Compile > Compile All**. If the compilation is successful, the question marks in the Status column will turn to check marks, and a success message will appear in the **Transcript** pane (see figure below).



The compiled VHDL files will now appear in the library “work”.



Since all of the inputs are undefined, if you ran the simulation now, the outputs would be undefined. So you need to have a means of setting the inputs to certain patterns, and of observing the outputs' responses to these inputs. In ModelSim, this is done by using a special VHDL entity called a Testbench. A testbench is special VHDL code that generates different inputs that will be applied to your circuit so that you can automate the simulation of your circuit and see how its outputs respond to different inputs.

Note that the testbench *is only used in Modelsim for the purposes of simulating your circuit*. You will eventually synthesize your circuits into a real hardware chip called an FPGA. However, you will NOT synthesize the testbench into real hardware. Because of its special purpose (and that it will not be synthesized), the testbench entity is unique in that it has NO inputs or outputs, and uses some special statements that are only used in test benches. These special statements are not used when describing circuits that you will later synthesize to a FPGA.

The testbench contains a single component instantiation statement that inserts the module to be tested (in this case the `gNN_7_segment_decoder` module), as well as some statements that describe how the test inputs are generated.

After you gain more experience you will be able to write VHDL testbenches from scratch. However, Quartus has a convenient built-in process, called the **Test Bench Writer**, which produces a VHDL template from your design that will get you started. To get the template, go back to the Quartus program, making sure that you have the `gNN_7_segment_decoder` project loaded. Then, in the Processing toolbar item, select “Start/Start Test Bench Template Writer”. This will generate a VHDL file named `gNN_7_segment_decoder.vht` and place it in the simulation/modelsim directory.

Open the template in Quartus. Note that the template already includes the instantiation of the under test circuit (i.e., `gNN_7_segment_decoder` component). It also includes the skeletons of two “process” blocks, one labeled “init” and the other labeled “always”. It is not important to understand process blocks at this point. We will learn about them later on. The init process block can be deleted. You should edit the “always” process block to suit your needs, so in this case it will be used to generate the code signal waveform. You will notice that inside the process block, signal code are assigned multiple times! This should not make sense right now. If a signal was assigned multiple times using concurrent signal statements, this would be an error! However, the rules for statements inside a process block are different. We will discuss process blocks later in the course. The “wait for x ns” statement is a special VHDL statement that is only used in

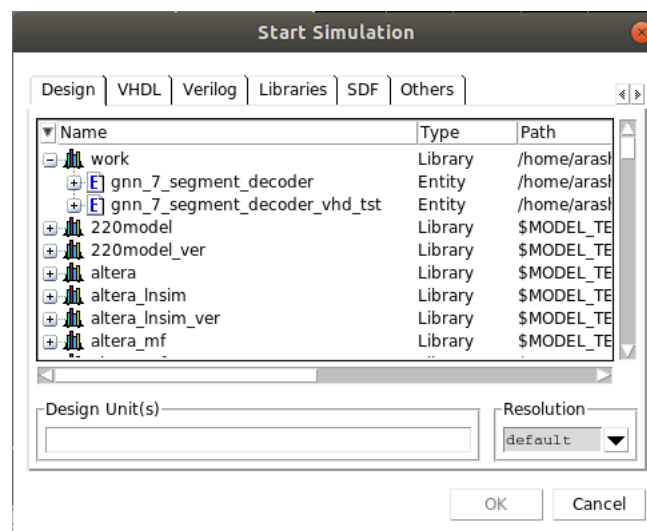
VHDL testbenches, and not in VHDL descriptions of synthesizable circuits that are intended to be implemented in real hardware. We never indicate time this way in synthesizable VHDL.

There are 2^4 or 16 possible patterns in the `gNN_7_segment_decoder` circuit, so complete testing of the circuit will require you to simulate all of these patterns. In order to run through all possible 16 cases, we use a FOR LOOP that increments the value of code signal in the loop. This is done in the testbench shown below.

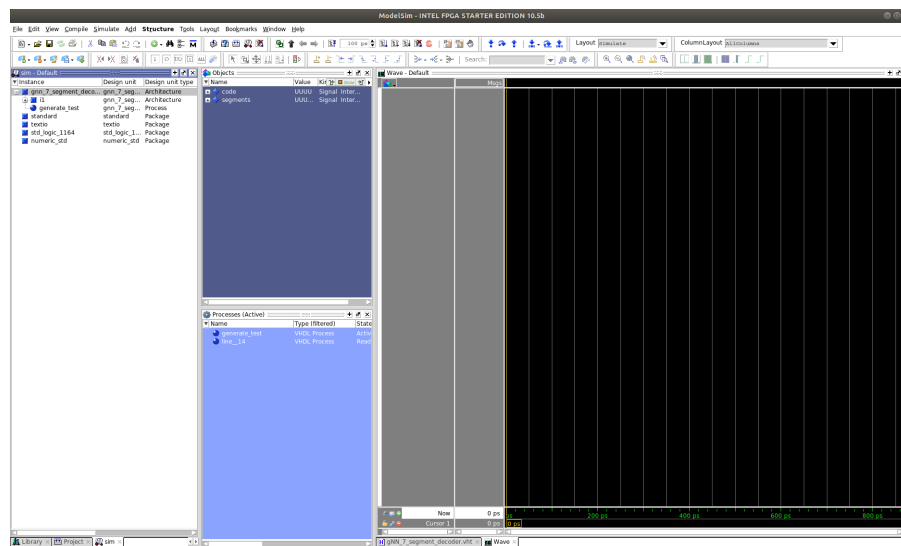
```
generate_test : PROCESS
BEGIN
  FOR i IN 0 to 15 LOOP -- loop over all code values
    code <= std_logic_vector(to_unsigned(i,4)); -- convert the loop variable i to
    std_logic_vector
    WAIT FOR 10 ns; -- suspend process for 10 nanoseconds at the start of each loop
  END LOOP; -- end the i loop
  WAIT; -- we have gone through all possible input patterns, so suspend simulator forever
END PROCESS generate_test;
```

The process block generates all of the possible input patterns using a FOR loop. The RANGE attribute is equivalent to specifying the loop range as over the minimum value of the signal to its maximum value. Replace the “always” process block to perform the complete test. Once you have finished editing the testbench file, you need to add it to the project in ModelSim by selecting **Project > Add to Project > Existing File...**

Once the testbench file has been added to the project, you should select the testbench file in the **Project** pane, and click on **Compile Selected** from the **Compile** toolbar item. This will compile the testbench file. Now everything is ready for you to actually run a simulation! Select “Start Simulation” from the **Simulate** toolbar item in the ModelSim program. The window shown below will appear.



Select the `gNN_7_segment_decoder_tst` entity and click on OK. The ModelSim window should now look like the figure below.



At first, the “Wave” window will not have any signals in it. You can drag signals from the “Objects” window by clicking on a signal, holding down the mouse button, and dragging the signal over to the Wave window. Do this for all the signals. The Wave window will now look like the one shown in Figure 1.

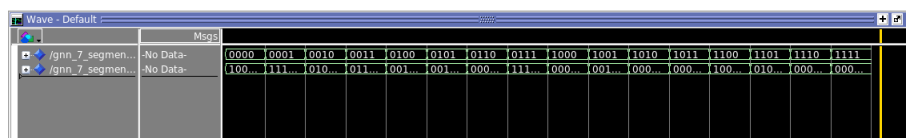


Figure 1: Signal waveform in ModelSim.

Now, to actually run the simulation, click on the “Run all” icon in the toolbar. Check the output of your implementation for every single case. If you get an incorrect output waveform, you will have to go back and look at your design. If you make a correction to your VHDL code, you will have to re-run the compilation of the changed files in ModelSim. Finally, to rerun the simulation, first click on the “Restart” button, then click on the “Run all” button.

7 Design a 5-bit Adder

In this part of the lab, you will design a circuit performing addition on two 5-bit inputs A and B. It also displays inputs and the result of the addition in hexadecimal format on the 7-segment LEDs on the DE1-SoC board. Note that you should use the binary-to-7-segment LED decoder to obtain appropriate 7-segment display code. Moreover, each signal requires two hexadecimal digits for representation. Therefore, you will need to use all the six 7-segment LEDs on the board in this lab.

Use the following entity declaration to write a VHDL description of the adder circuit. Note that you have to instantiate from the `gNN_7_segment_decoder` circuit in your VHDL description.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity gNN_adder is
    Port (
        A, B           : in  std_logic_vector(4 downto 0);
        decoded_A       : out std_logic_vector(13 downto 0);
        decoded_B       : out std_logic_vector(13 downto 0);
        decoded_AplusB   : out std_logic_vector(13 downto 0));
end gNN_adder;
```

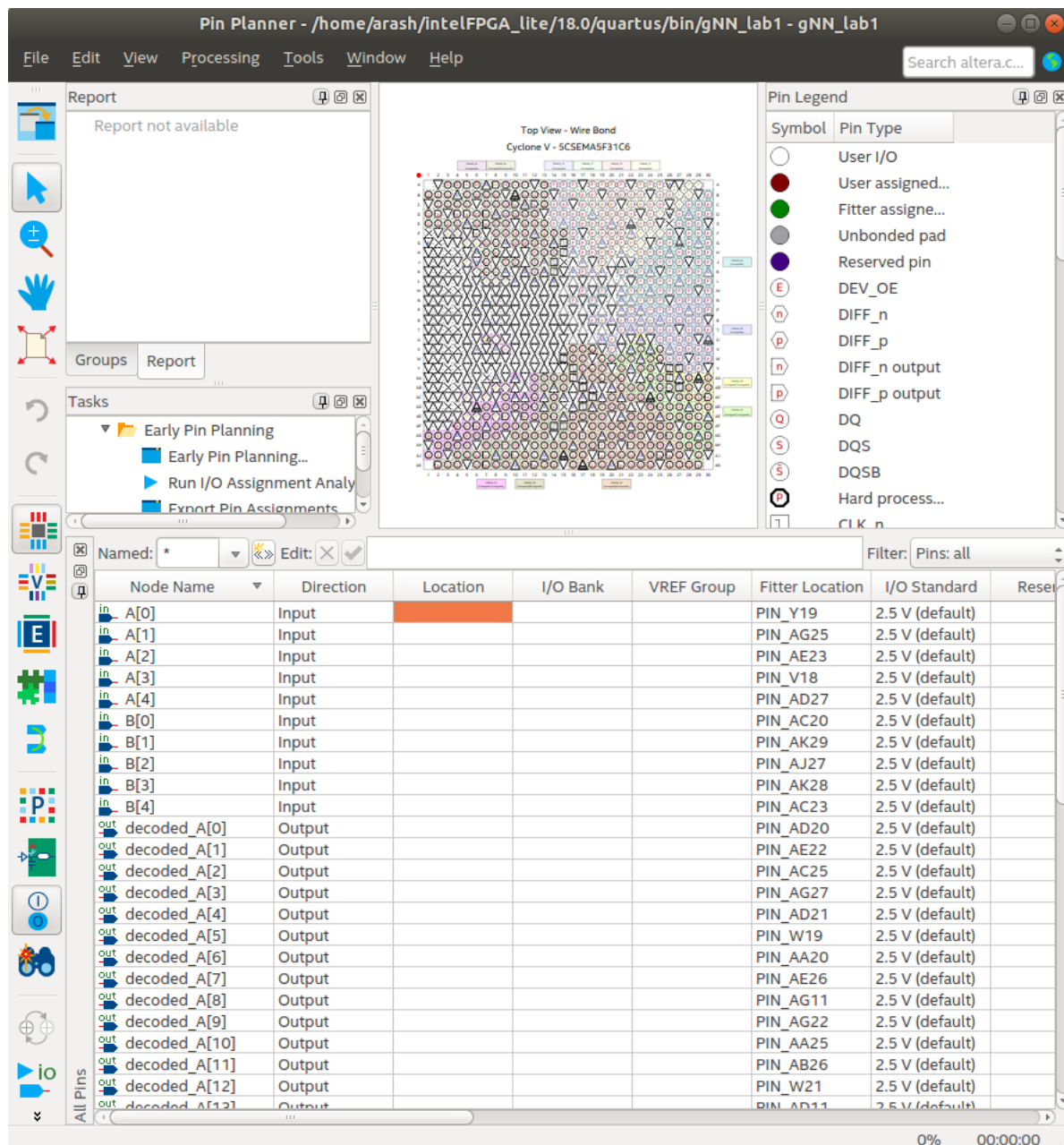
8 Testing the Adder on the Altera Board

You will now test the adder circuit you designed in Section 7. Compile the decoder in the Quartus software. Once you have compiled the adder circuit, it is time to map it onto the target hardware, in this case the Cyclone V chip on the Altera DE1-SoC board. Please begin by reading over the DE1-SoC user's manual, which can be found on the myCourses lab experiments page.

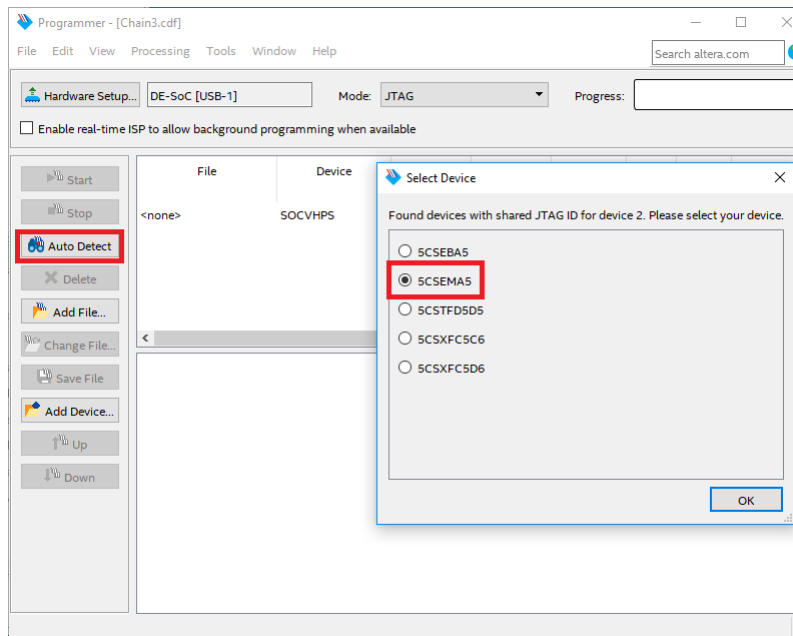
Since you will now be working with an actual device, you have to be concerned with which device package pins the various inputs and outputs of the project are connected. In particular, you will want to connect the LED segment outputs from the instances of the `gNN_7_segment_decoder` circuit (i.e., the outputs of the adder circuit) to the corresponding segments of one of the six 7-segment LED displays on the board. The mapping of the board's 7-segment LEDs's segments to the pins on the Cyclone FPGA device is listed in Table 3-9 on page 24 of the DE1-SoC Development and Education Board Users Manual.

You will also want to connect, for testing purposes, 5 of the slide switches on the DE1-SoC board to the input A and the rest to the input B of the `gNN_adder` circuit. The mapping of the slide switches to the FPGA pins is given in Table 3-6 on pages 23 of the DE1 user's manual.

You can tell the compiler of your choices for pin assignments for your inputs and outputs by opening the **Pin Planner**, which can be done by choosing the Pins item in the **Assignments** menu, as shown below.



Once you have assigned all of the inputs and outputs of your circuit to appropriate device pins, re-compile your design. Your design is now ready to be downloaded to the target hardware. Read section 4.1 of the DE1-SoC user's manual for information on configuring (programming) the Cyclone V FPGA on the board. You will be using the JTAG mode to configure the device. Take the board out of the kit box, and connect the USB cable to the computer's USB port and to the USB connector on the board. Next, select the Programmer item from the Tools menu. Click Auto Detect and then select the correct device (5CSEMA5), as shown below. Both FPGA device and HPS should be detected.



Next, double-click the FPGA device (5CSEMA5), and from the window that opens add the .sof file created by Quartus. Finally, check the “Program/configure” box beside the 5CSEMA5 device, and then click “Start”. Now, you should be able to use slide switches to insert values for inputs A and B. The 7-segment LEDs should also display inputs and outputs in hexadecimal format.

9 Deliverables and Grading

9.1 Demo

Once completed, you will demo your project to the TA. You will be expected to:

- fully explain how the HDL code works,
- perform functional simulation using ModelSim, and
- demonstrate that the adder circuit is functioning properly using the slide switches and 7-segment LEDs on the DE1-SoC board.

9.2 Written report

You are also required to submit a written report and your code on myCourses. Your report must include:

- A description of the 7-segment decoder circuit. Explain why you used the selected signal assignment instead of the conditional signal assignment.
- A discussion of how the 7-segment decoder circuit was tested, showing representative simulation plots. How do you know that the circuit works correctly?
- A description of the adder circuit. How many 7-segment decoder instances did you use in your design and why?
- A discussion of how the adder circuit was tested.
- A summary of the FPGA resource utilization (from the Compilation Report's Flow Summary) and the RTL schematic diagram for both the 7-segment decoder and the adder circuits. Clearly specify which part of your code maps to which part of the schematic diagram.

Finally, when you prepare your report have in mind the following:

- The title page must include the lab number, name and student ID of the students, as well as the group number.
- All figures and tables must be clearly visible.
- The report should be submitted in PDF format.
- It should document every design choice clearly.
- The grader should not have to struggle to understand your design. That is,
 - Everything should be organized for the grader to easily reproduce your results by running your code through the tools.
 - The code should be well-documented and easy to read.

Grading Sheet

Group Number:
Name 1:
Name 2:

Task	Grade	/Total	TA Signature
Creating Project		/10	
VHDL code for the 7-segment decoder circuit		/20	
Creating testbench code for the 7-segment decoder circuit		/10	
Functional simulation of the 7-segment decoder circuit		/10	
VHDL code for the adder circuit		/20	
Testing the adder circuit on the DE1 - SoC board		/30	
Total		/100	