HWP01

Programmable Hardware

Lab Work Handbook

Version 3.1

E.H.W. van de Logt
J.H. Peltenburg
J.Z.M. Broeders

Lab Work Handbook Programmable Hardware

Versie 3.1

E.H.W. van de Logt
J.H. Peltenburg
J.Z.M. Broeders

Contents

Introduction

In the modern world of electronics, everything must be fast; the time-to-market must be short and performance must be high. Recently, the PLD (programmable logic device) market has grown tremendously (turnover doubles about every five years[1]) because these devices have become more and more adapt to meet these criteria in many fields of electronics.

FPGAs (field-programmable gate arrays) and other reconfigurable hardware in general are becoming more popular every day. If you have a high-performance application without high power requirements, but you don't have money or time to make an ASIC (application specific integrated circuit), reconfigurable hardware is often a good choice.

HDLs (hardware description languages) capture the functionality of digital schematics in plain text. *In short; a HDL is used to "draw" digital logic with text.* The most widely-used HDLs today are VHDL and Verilog, but there are many others (System C for example). In practise, HDLs are used to design full-scale production ICs, ASICs and especially FPGA configurations.

The power of HDLs is the high level of abstraction. Developing applications with HDLs and letting tools compile and place them in your PLD is faster than drawing schematics by hand. Next to that, the HDL code can often be reused in newer, bigger PLDs and is easily scalable. For example; a well coded 16-bit microprocessor core (captured in HDL) may be changed into a 32-bit microprocessor core by adjusting a few variables only.

The goal of this course is an introduction to VHDL, FPGAs and PLDs in general. The functional designs made by students are implemented into an FPGA that resides on the Altera DE1-SoC development kit.

This document contains the introductory assignment for this course. This first assignment is a step by step introduction to designing and simulating logic with ModelSim and synthesizing and testing it with Quartus.

[1] See: `https://www.researchandmarkets.com/reports/5398214/fpga-market-by-configuration-low-end-fpga-mid`

1 Purpose and prerequisites

The purpose of the first introductory assignment is:

- to create a simple functional design with basic logic for a digital system consisting of LEDs and switches;
- to simulate a digital system and thus verify its functionality with ModelSim;
- to introduce you to working with the Quartus Prime software tool;
- to reconfigure an FPGA with a digital schematic.

The perquisites of this assignment are:

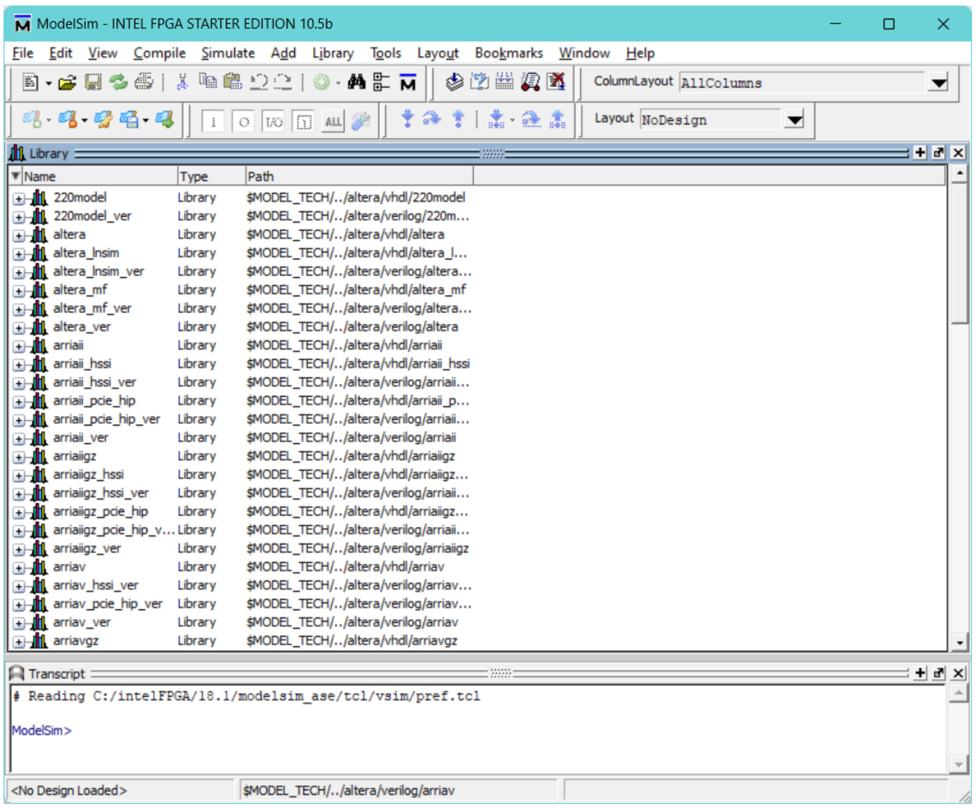
- the Terasic DE1-SoC Development Kit is available;
- Intel Quartus Prime Lite edition (version 18.1) software including the has been installed; We use this older version because we want to use the Nios softcore processor in the following up course CSC10;
- ModelSim-Intel FPGA Edition has been installed.

The first thing we will do is create a project in ModelSim to make functional simulations.

2 Creating a ModelSim Project

Step 1: Find and open the link to “ModelSim-Intel FPGA Starter Edition” on your desktop or in the start menu and start ModelSim.

After closing the “Important info” window, it should show the following window:


ModelSim - INTEL FPGA STARTER EDITION 10.5b
File Edit View Compile Simulate Add Library Tools Layout Bookmarks Window Help
ColumnLayout AllColumns
Layout NoDesign
Library
Transcript

Name	Type	Path
220model	Library	$MODEL_TECH/../altera/vhdl/220model
220model_ver	Library	$MODEL_TECH/../altera/verilog/220m...
altera	Library	$MODEL_TECH/../altera/vhdl/altera
altera_lnsim	Library	$MODEL_TECH/../altera/vhdl/altera_l...
altera_lnsim_ver	Library	$MODEL_TECH/../altera/verilog/altera...
altera_mf	Library	$MODEL_TECH/../altera/vhdl/altera_mf
altera_mf_ver	Library	$MODEL_TECH/../altera/verilog/altera...
altera_ver	Library	$MODEL_TECH/../altera/verilog/altera
arriaii	Library	$MODEL_TECH/../altera/vhdl/arriaii
arriaii_hssi	Library	$MODEL_TECH/../altera/vhdl/arriaii_hssi
arriaii_hssi_ver	Library	$MODEL_TECH/../altera/verilog/arriaii...
arriaii_pcie_hip	Library	$MODEL_TECH/../altera/vhdl/arriaii_p...
arriaii_pcie_hip_ver	Library	$MODEL_TECH/../altera/verilog/arriaii...
arriaii_ver	Library	$MODEL_TECH/../altera/verilog/arriaii
arriaiigz	Library	$MODEL_TECH/../altera/vhdl/arriaiigz
arriaiigz_hssi	Library	$MODEL_TECH/../altera/vhdl/arriaiigz...
arriaiigz_hssi_ver	Library	$MODEL_TECH/../altera/verilog/arriaii...
arriaiigz_pcie_hip	Library	$MODEL_TECH/../altera/vhdl/arriaiigz...
arriaiigz_pcie_hip_v...	Library	$MODEL_TECH/../altera/verilog/arriaii...
arriaiigz_ver	Library	$MODEL_TECH/../altera/verilog/arriaiigz
arriav	Library	$MODEL_TECH/../altera/vhdl/arriav
arriav_hssi_ver	Library	$MODEL_TECH/../altera/verilog/arriav...
arriav_pcie_hip_ver	Library	$MODEL_TECH/../altera/verilog/arriav...
arriav_ver	Library	$MODEL_TECH/../altera/verilog/arriav
arriavgz	Library	$MODEL_TECH/../altera/vhdl/arriavgz

```
# Reading C:/intelFPGA/18.1/modelsim_ase/tcl/vsim/pref.tcl

ModelSim>
```

<No Design Loaded> $MODEL_TECH/../altera/verilog/arriav

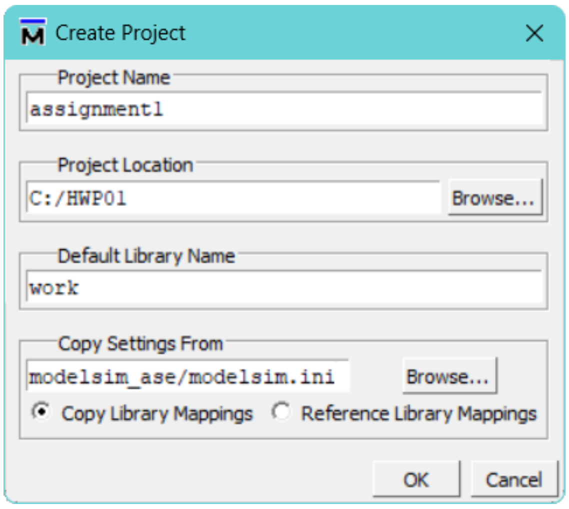
The library window shows all libraries. Libraries contain components that can be used in simulations. Most of the components in the ModelSim INTEL FPGA Starter edition shown in the list are hardware components inside the physical FPGA that can be used in your designs. There are also some other useful components that can be used for testing. For this course, we will not make use of any of them. Eventually, we will create our own library with our own components (later more on this).

We will first create a new project.

Step 2: Go to File » New » Project....

You should see the “create project” window.

Step 3: Fill it in as follows:



Create Project

Project Name: assignment1

Project Location: C:/HWP01

Default Library Name: work

Copy Settings From: modelsim_ase/modelsim.ini

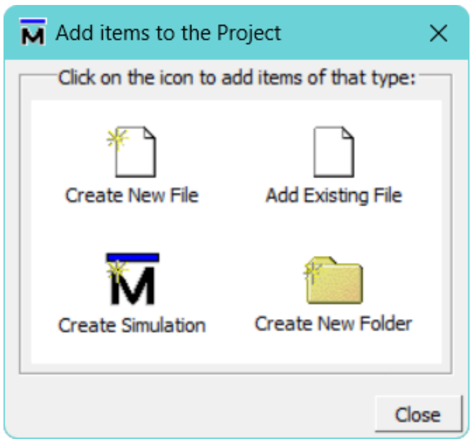
[✓ Copy Library Mappings] Reference Library Mappings

Step 4: Click OK.

It will probably show a pop-up window that asks if you want to ModelSim to create a directory for your project.

Step 5: Let ModelSim create the new directory.

A new window should pop up:

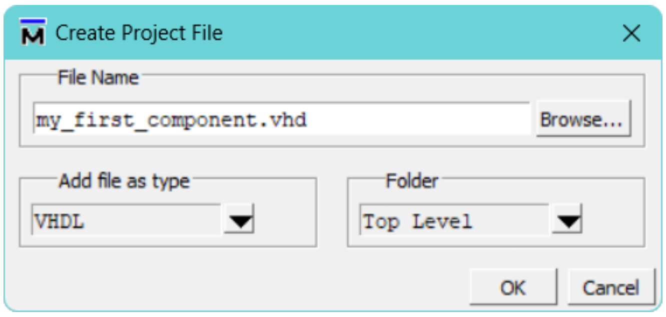


Add items to the Project

Click on the icon to add items of that type:

Step 6: Click “Create New File.”

We will now create our first VHDL file. The extension for VHDL files is `.vhd`. *The name of the file must be the same of your components name!* Since the component will be named “my_first_component” we will name the file `my_first_component.vhd`.


Create Project File

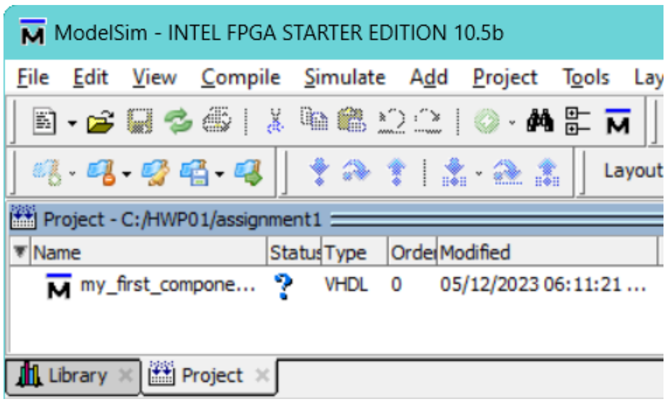
File Name: my_first_component.vhd

Add file as type: VHDL

Folder: Top Level

Step 7: Click OK and Close to close the “Add items to project” window.

Now ModelSim will show the project window:


ModelSim - INTEL FPGA STARTER EDITION 10.5b

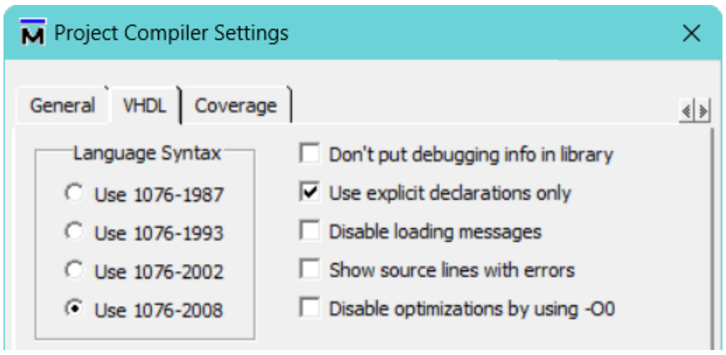
Project - C:/HWP01/assignment1

Name	Status	Type	Order	Modified
my_first_compone...	?	VHDL	0	05/12/2023 06:11:21 ...

We can see our VHDL file included in the project. The status of the file, which is a question mark, means that the file is not compiled yet. We will do this later.

Step 8: Right-click on the file and select Properties.... Select the “VHDL” tab and choose “Use 1076-2008”.

We will use the latest (2008) version of the VHDL standard.


Project Compiler Settings

Language Syntax
- ( ) Use 1076-1987
- ( ) Use 1076-1993
- ( ) Use 1076-2002
- (●) Use 1076-2008

- [ ] Don't put debugging info in library
- [✓] Use explicit declarations only
- [ ] Disable loading messages
- [ ] Show source lines with errors
- [ ] Disable optimizations by using -O0

Step 9: Double-click on the file to edit it. Add the following lines to the file:

```
library ieee;
use ieee.std_logic_1164.all;

entity my_first_component is
port (
    inputs: in std_ulogic_vector(3 downto 0);
    outputs: out std_ulogic_vector(6 downto 0)
);
end my_first_component;
```

The first two lines define the standard libraries that contain many different functions and data types in VHDL we want to use for this (and probably any other) project.

The `entity` part defines the interface of our functional block. The `port` part inside the `entity` defines which ports we want to see at the interface of our functional block. It does not define the implementation of the functionality of the block itself. This is done in the `architecture` section.

This concept, the difference between and separation of the *interface* and the *implementation* of a (sub)system, is the key to system engineering in any field.

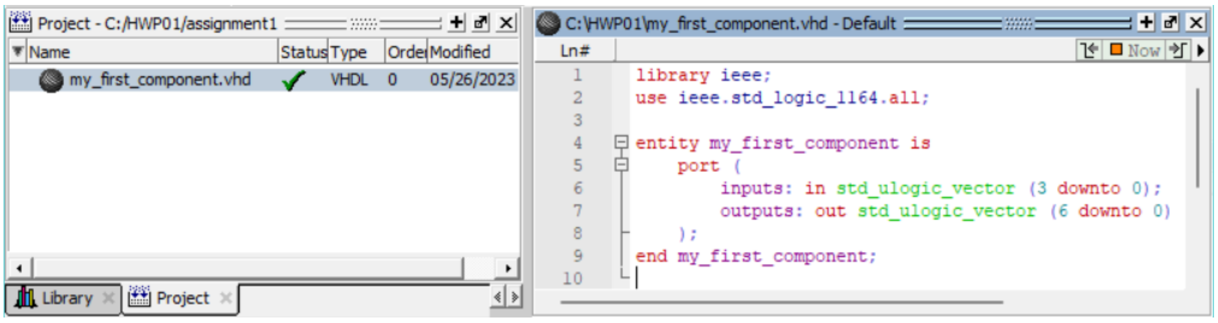
`in std_ulogic_vector(3 downto 0)` defines an input port (high impedance). `std_ulogic` is a basic digital connection. A vector makes it a group or bus of inputs. `3 downto 0` tells us that the bus is four bits wide. This is the VHDL notation for a bus. Note that we have written it in MSB-first style (Most Significant Bit first).

Step 10: Save the file.

We have now declared the interface of your component. What is missing is how it should behave. This is done in the architecture. However, only declaring the interface is enough to compile our component.

Step 11: Right click on the file and select Compile > Compile Selected.

You should now see the following:


Project - C:/HWP01/assignment1
Library
Project
C:\HWP01\my_first_component.vhd - Default
Ln#
Now

Name	Status	Type	Order	Modified
my_first_component.vhd	✓	VHDL	0	05/26/2023

```
library ieee;
use ieee.std_logic_1164.all;

entity my_first_component is
    port (
        inputs: in std_ulogic_vector (3 downto 0);
        outputs: out std_ulogic_vector (6 downto 0)
    );
end my_first_component;
```

Note that the status of the component has also changed to a green ✓. This means it compiled successfully. If it didn't compile successfully it would show a red ✗. Even though we compiled the component, we cannot simulate it, because we have not declared how it should behave. This can be done by adding the `architecture` section to the code.

Step 12: Add the following lines to the file:

```
architecture implementation of my_first_component is
begin
    outputs(0) <= inputs(0);
    outputs(1) <= inputs(1);
    outputs(2) <= inputs(0) or inputs(1);
    outputs(3) <= inputs(0) and inputs(1);
    outputs(4) <= inputs(0) and inputs(1) and inputs(2) and ↩
  ↪ inputs(3);
    outputs(5) <= inputs(0) or inputs(1) or inputs(2) or inputs(3);
    outputs(6) <= not(inputs(0) or inputs(1) or inputs(2) or ↩
  ↪ inputs(3));
end implementation;
```

The outputs are now some combinational function of the inputs.

Step 13: Fill in the truth table for this function, shown below.

inputs	outputs
0000	1000000
0001	0100101
0010	0100110
0011	0101111
0100	....
0101	....
0110	....
0111	....
1000	....
1001	....
1010	....
1011	....
1100	....
1101	....
1110	....
1111	....

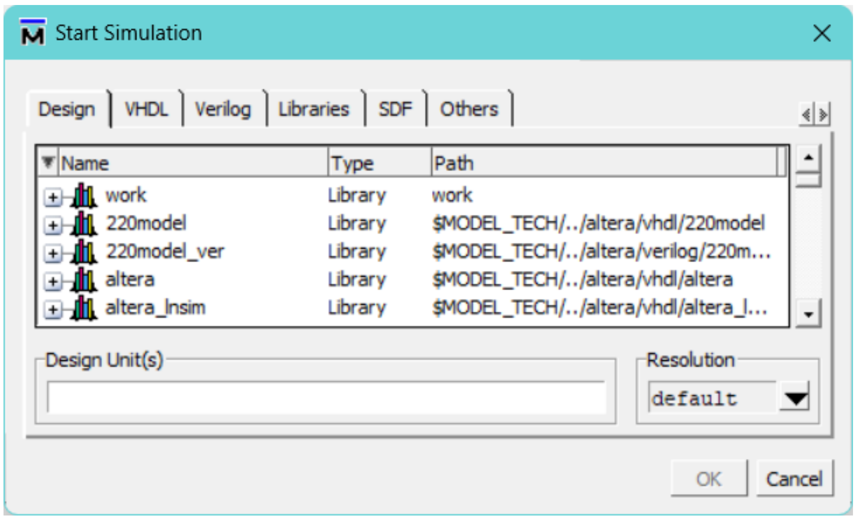
We will now verify these results by simulating our little design.

3

Simulating your design with ModelSim

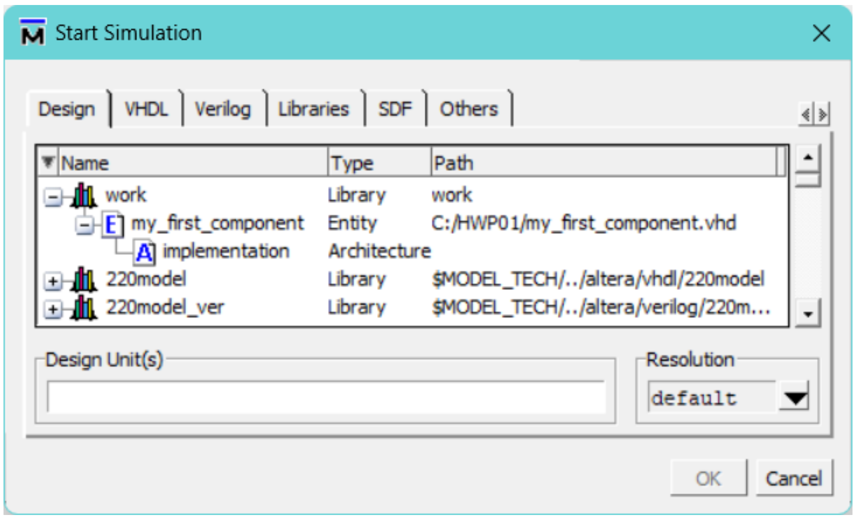
Step 14: In the ModelSim menu, select Simulate › Start Simulation....

After opening up the "work" library, you should see the following dialog:



Name	Type	Path
work	Library	work
220model	Library	$MODEL_TECH/../altera/vhdl/220model
220model_ver	Library	$MODEL_TECH/../altera/verilog/220m...
altera	Library	$MODEL_TECH/../altera/vhdl/altera
altera_lnsim	Library	$MODEL_TECH/../altera/vhdl/altera_l...

When we created the project, we called our default library "work". This library now appears with all our compiled components in the dialog. Expand the library "work", and you can see "my_first_component" appearing there. You may even expand the component in the list to see which implementations are available. In our case, we have only one implementation which is called "implementation".



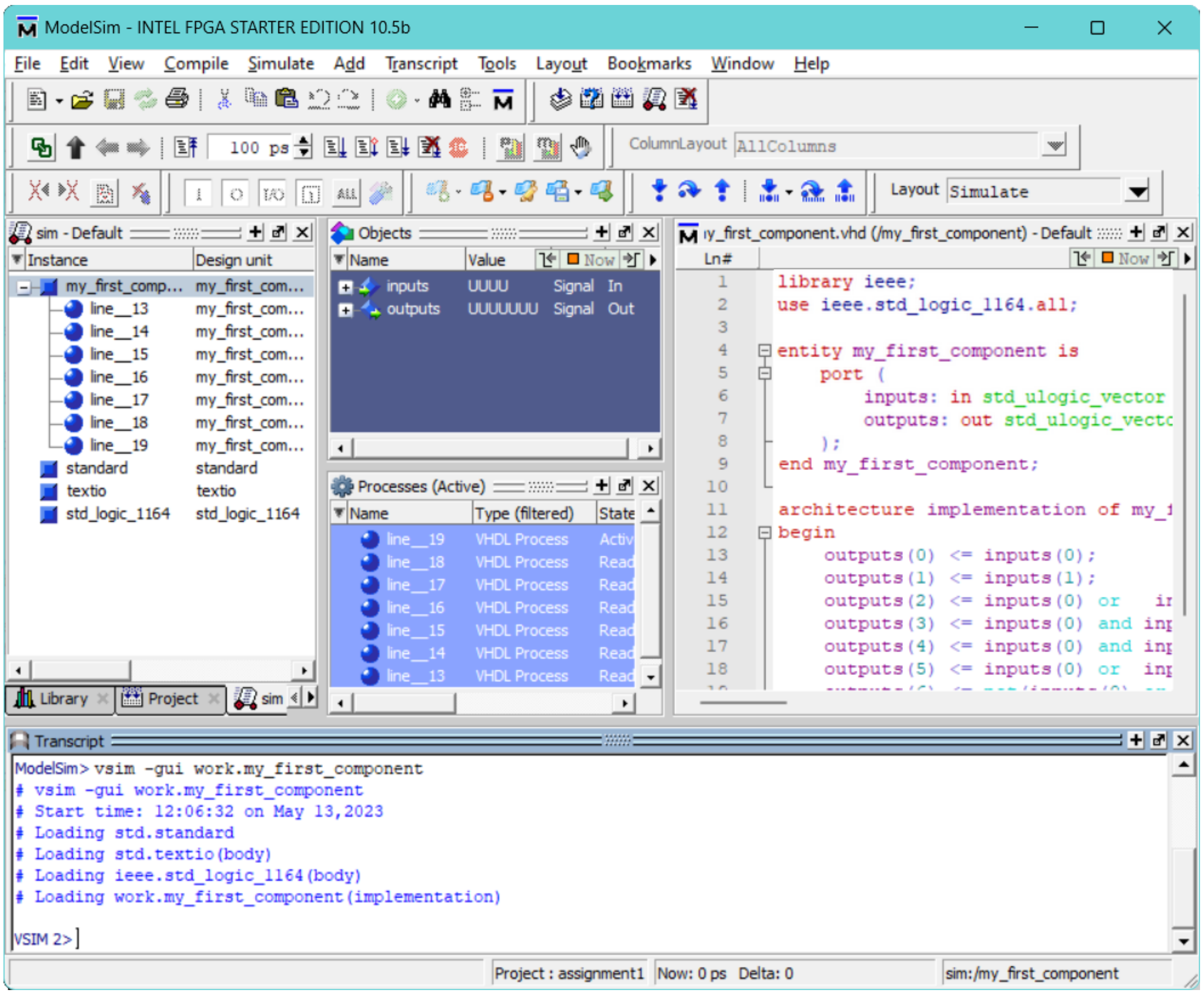
Name	Type	Path
work	Library	work
my_first_component	Entity	C:/HWP01/my_first_component.vhd
implementation	Architecture	
220model	Library	$MODEL_TECH/../altera/vhdl/220model
220model_ver	Library	$MODEL_TECH/../altera/verilog/220m...

Design Unit(s):

Resolution: default

Step 15: Select “my_first_component” and click OK to start the simulation.

You should now see the following window:



Instance	Design unit
my_first_comp...	my_first_com...
line__13	my_first_com...
line__14	my_first_com...
line__15	my_first_com...
line__16	my_first_com...
line__17	my_first_com...
line__18	my_first_com...
line__19	my_first_com...
standard	standard
textio	textio
std_logic_1164	std_logic_1164

Name	Value		
inputs	UUUU	Signal	In
outputs	UUUUUU	Signal	Out

Name	Type (filtered)	State
line__19	VHDL Process	Activ
line__18	VHDL Process	Read
line__17	VHDL Process	Read
line__16	VHDL Process	Read
line__15	VHDL Process	Read
line__14	VHDL Process	Read
line__13	VHDL Process	Read

ıy_first_component.vhd (/my_first_component) - Default

```
library ieee;
use ieee.std_logic_1164.all;

entity my_first_component is
    port (
        inputs: in std_ulogic_vector
        outputs: out std_ulogic_vecto
    );
end my_first_component;

architecture implementation of my_f
begin
    outputs(0)  <= inputs(0);
    outputs(1)  <= inputs(1);
    outputs(2)  <= inputs(0) or  ir
    outputs(3)  <= inputs(0) and inp
    outputs(4)  <= inputs(0) and inp
    outputs(5)  <= inputs(0) or  inp
```

```
ModelSim> vsim -gui work.my_first_component
# vsim -gui work.my_first_component
# Start time: 12:06:32 on May 13,2023
# Loading std.standard
# Loading std.textio(body)
# Loading ieee.std_logic_1164(body)
# Loading work.my_first_component(implementation)

VSIM 2>
```

Project : assignment1 | Now: 0 ps Delta: 0 | sim:/my_first_component

This is the simulation setup of ModelSim. In the “Instance” window we can see all components and their underlying architectures and processes from which we can select objects to include in the simulation. All objects that can be included are shown in the “Objects” window.

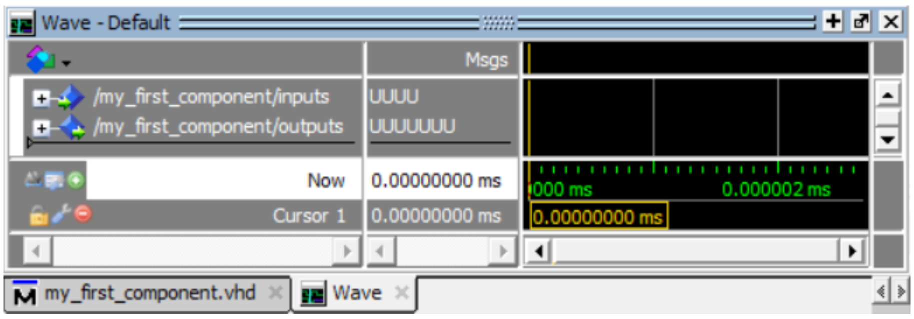
We can clearly see our signals `inputs` and `outputs` here.

Now, if we want to simulate our design with all the signals that are in it do the following:

Step 16: Right click in the “Objects” window.

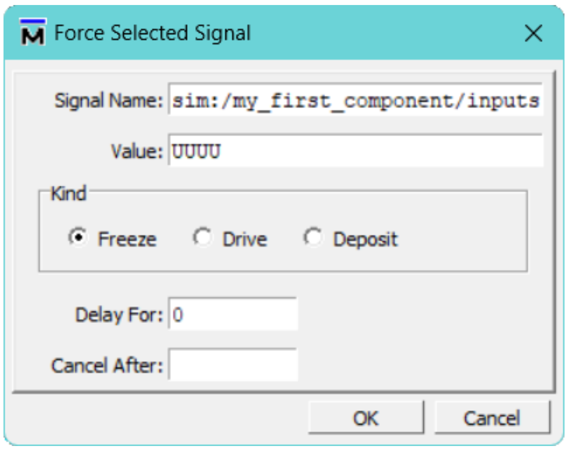
Step 17: Click Add to » Wave » Signals in Design.

All signals are now added to the “Wave” window. We can now see our signals in the wave window:



Their values are all U’s. This means the values are unknown. To set the value of a signal, right-click this signal’s name or value in the wave window and select Force... . Of course, we should only force signals that are inputs.

Step 18: Open the force window on the signal `inputs`.



The value here is shown as `UUUU`. Remember that the inputs signal is a vector of four values. The inputs are currently all unknown (`U`).

Step 19: Change the value to `0000`.

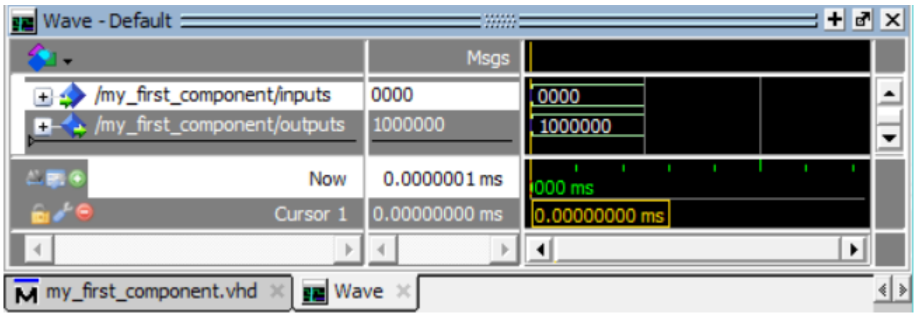
This means that `input(0)` will get the value `0`. `input(1)` will get the value `0` as well, and so forth.

Step 20: Click OK to apply the changes.

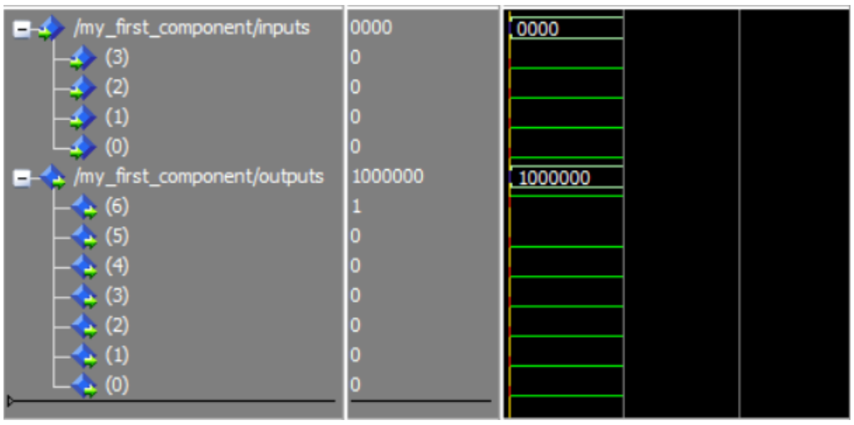
You will not see any changes right away in the wave window, but that is because the simulation hasn’t started yet.

Step 21: To run the simulation for 100 time steps, go to menu. Click Simulate » Run » Run 100.

The Wave window should now show (to enlarge hold the Ctrl button down while scrolling with the mouse):



You can also expand the signals that are vectors to show their individual elements. This should look like this:


/my_first_component/inputs 0000
(3) 0
(2) 0
(1) 0
(0) 0
/my_first_component/outputs 1000000
(6) 1
(5) 0
(4) 0
(3) 0
(2) 0
(1) 0
(0) 0
0000
1000000

Verify that the outputs have the correct values (corresponding to your code).

Step 22: Now force the inputs to `0001` and repeat until you have tested all possible inputs (assume the values of each individual input can only be `0` or `1`.

Take a screenshot of the wave window which shows all possible input values and the associated output values. Compare the output values with the expected values given in the truth table you filled in at step 13.

Verifying your design by forcing all possible input combinations one by one and comparing the simulation results from the wave window with the intended behavior in the truth table is a time-consuming and error-prone job. During this course you will learn a better way to verify your design by using a so called “testbench”.

If you are not sure your design works properly, call your instructor to verify your results.

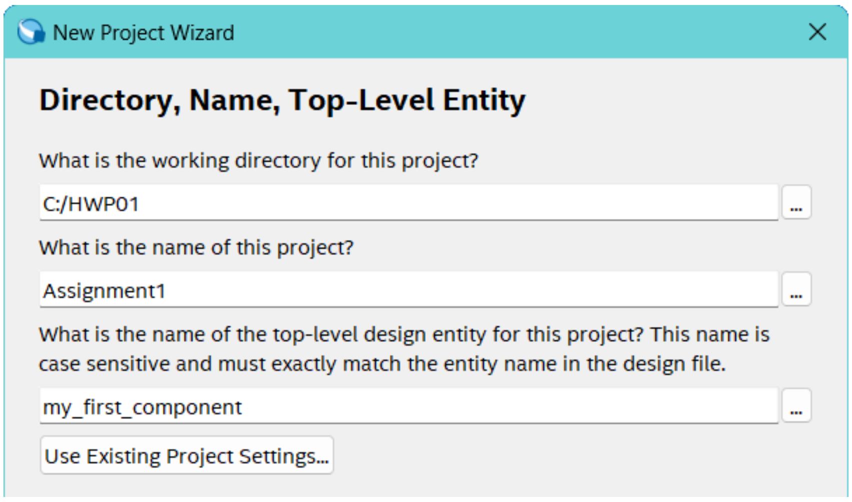
4

Creating a Quartus project

Now the design is ready to be exported to Quartus, so we may continue to implement it with the FPGA.

- Start “Quartus (Quartus Prime 18.1)” from the desktop or from the start menu.
- Start the “New Project Wizard”.
- Skip the introduction of the Wizard.
- In the “Directory, Name, etc.” page of the wizard, select the same directory as your ModelSim project as the working directory.
- Name the project “assignment1” as well.
- The top-level entity is our component called “my_first_component”. If you decide to add a different top level later, it is easy to change after you’ve completed the wizard.

Eventually, the current page should look as follows:



Directory, Name, Top-Level Entity

What is the working directory for this project?

C:/HWP01

What is the name of this project?

Assignment1

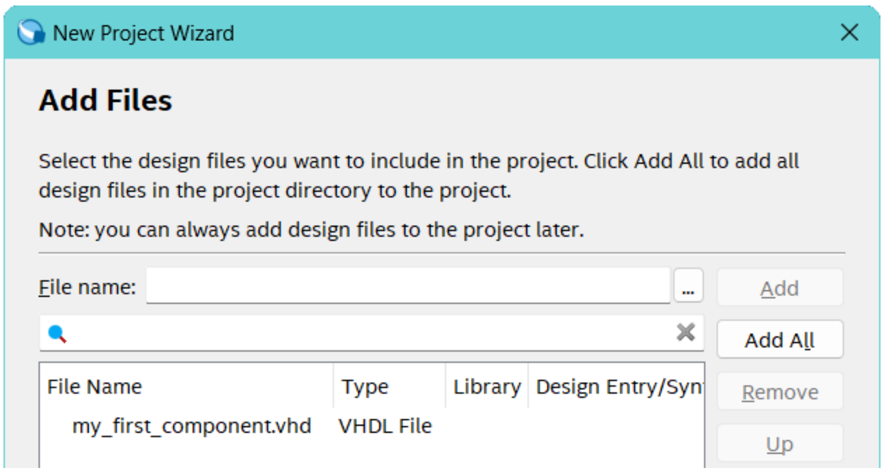
What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

my_first_component

Step 23: Click Next.

Step 24: Set Project Type to “Empty Project” and click Next.

On the next page (Add Files), since we’ve already created a VHDL file, we can easily include it in the project by clicking Add All. The page should now look like this:


New Project Wizard

Add Files

Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.

Note: you can always add design files to the project later.

File name:

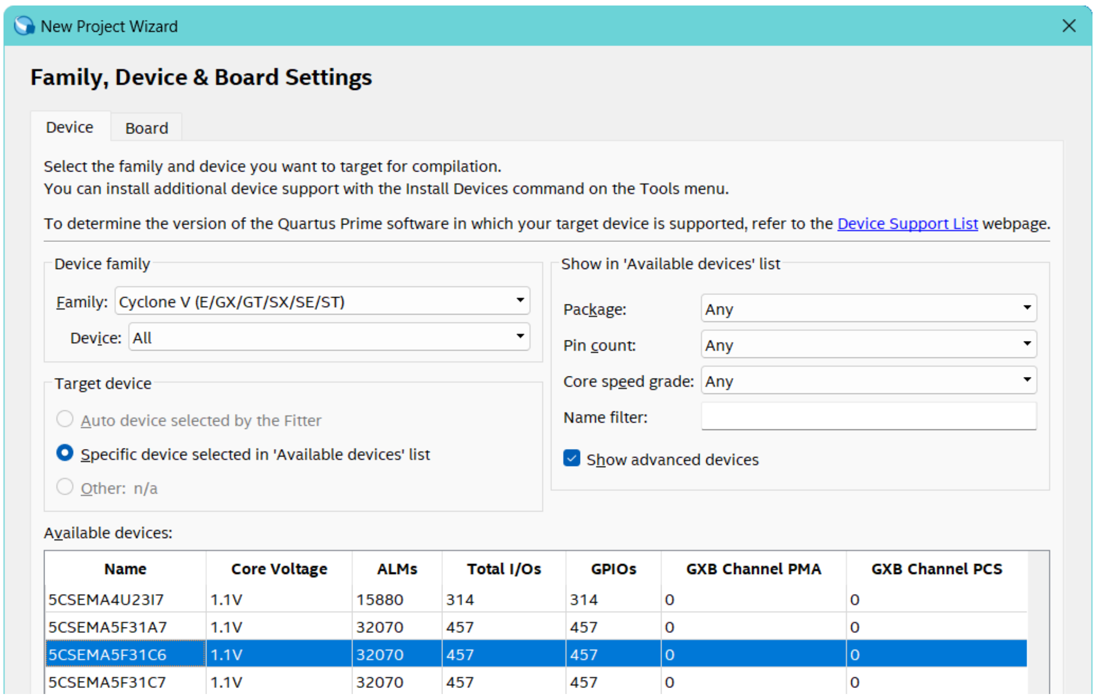
File Name	Type	Library	Design Entry/Syn
my_first_component.vhd	VHDL File		

Step 25: Click Next.

In the Family and Device Settings page, we have to select which FPGA we're using. Our programming tools are well capable of selecting the proper device automatically (this is done through the JTAG chain), but let us select the right FPGA from the start.

The right device is the *Cyclone V 5CSEMA5F31C6*.

Step 26: Select this device:


New Project Wizard

Family, Device & Board Settings

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)

Device: All

Target device

Auto device selected by the Fitter

[✓ Specific device selected in 'Available devices' list]

Other: n/a

Show in 'Available devices' list

Package: Any

Pin count: Any

Core speed grade: Any

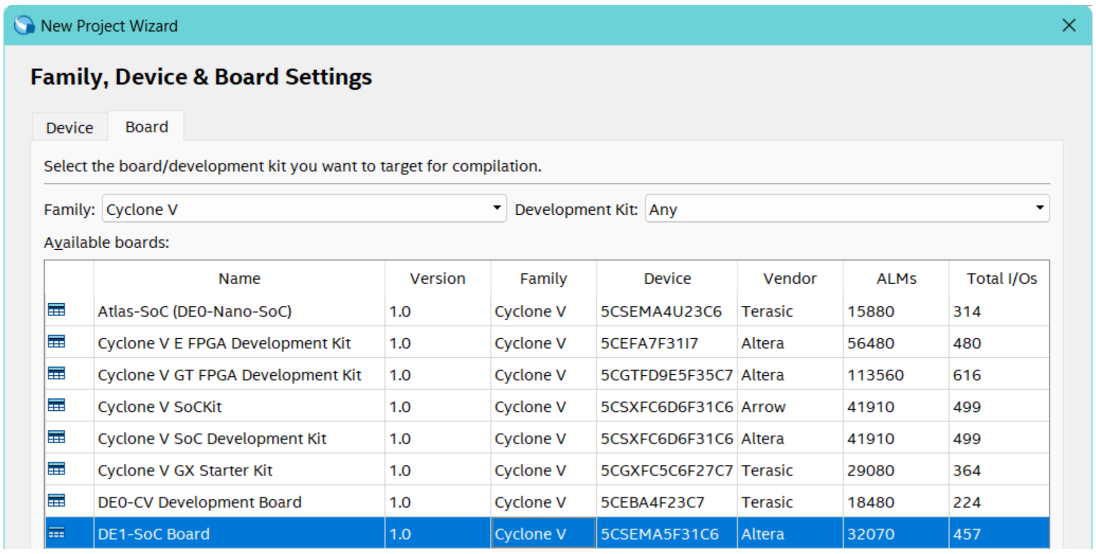
Name filter:

[✓ Show advanced devices]

Available devices:

Name	Core Voltage	ALMs	Total I/Os	GPIOs	GXB Channel PMA	GXB Channel PCS
5CSEMA4U23I7	1.1V	15880	314	314	0	0
5CSEMA5F31A7	1.1V	32070	457	457	0	0
5CSEMA5F31C6	1.1V	32070	457	457	0	0
5CSEMA5F31C7	1.1V	32070	457	457	0	0

Alternatively, you can select the Board tab and select the DE1-SoC board:



Family, Device & Board Settings

Select the board/development kit you want to target for compilation.

Family: Cyclone V

Development Kit: Any

Available boards:

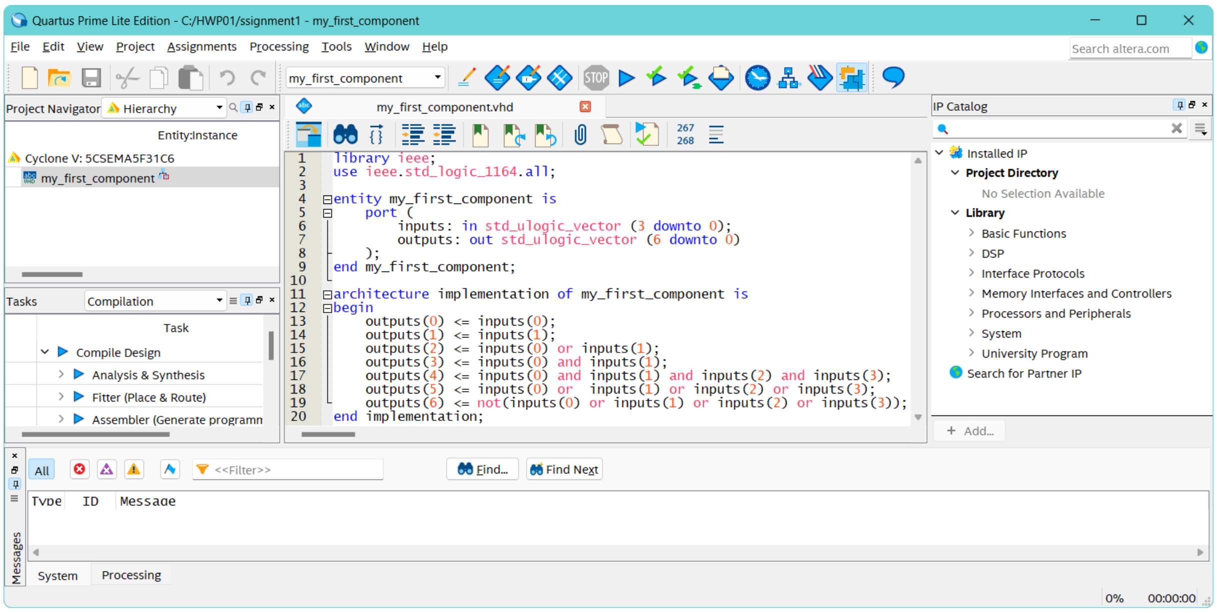
	Name	Version	Family	Device	Vendor	ALMs	Total I/Os
	Atlas-SoC (DE0-Nano-SoC)	1.0	Cyclone V	5CSEMA4U23C6	Terasic	15880	314
	Cyclone V E FPGA Development Kit	1.0	Cyclone V	5CEFA7F31I7	Altera	56480	480
	Cyclone V GT FPGA Development Kit	1.0	Cyclone V	5CGTFD9E5F35C7	Altera	113560	616
	Cyclone V SoCKit	1.0	Cyclone V	5CSXFC6D6F31C6	Arrow	41910	499
	Cyclone V SoC Development Kit	1.0	Cyclone V	5CSXFC6D6F31C6	Altera	41910	499
	Cyclone V GX Starter Kit	1.0	Cyclone V	5CGXFC5C6F27C7	Terasic	29080	364
	DE0-CV Development Board	1.0	Cyclone V	5CEBA4F23C7	Terasic	18480	224
	DE1-SoC Board	1.0	Cyclone V	5CSEMA5F31C6	Altera	32070	457

Step 27: Now click Next until you can Finish the wizard.

We can now see the new project in the "Project Navigator" pane on the left of the Quartus II window. We want to implement "my_first_component" on the Cyclone V FPGA that is on the DE1-SoC development and education kit.

You may double-click on "my_first_component" on the left side at the tab "Files" to show its source. This can be a schematic, a VHDL file, a Verilog file or anything else that Quartus can handle.

In our case, we know it is our VHDL file we created in ModelSim, so double-clicking it should result in the following:



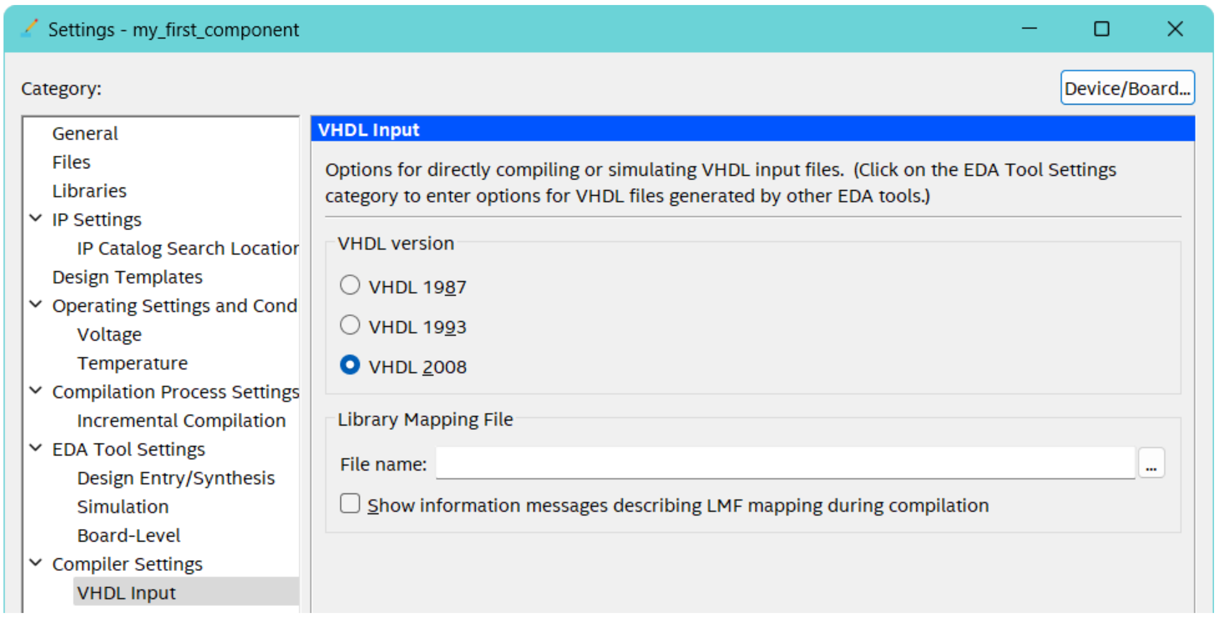
```
library ieee;
use ieee.std_logic_1164.all;

entity my_first_component is
    port (
        inputs: in std_ulogic_vector (3 downto 0);
        outputs: out std_ulogic_vector (6 downto 0)
    );
end my_first_component;

architecture implementation of my_first_component is
begin
    outputs(0) <= inputs(0);
    outputs(1) <= inputs(1);
    outputs(2) <= inputs(0) or inputs(1);
    outputs(3) <= inputs(0) and inputs(1);
    outputs(4) <= inputs(0) and inputs(1) and inputs(2) and inputs(3);
    outputs(5) <= inputs(0) or  inputs(1) or inputs(2) or inputs(3);
    outputs(6) <= not(inputs(0) or inputs(1) or inputs(2) or inputs(3));
end implementation;
```

```
Type  ID  Message
```

Because we want ti use VHDL 2008 features in our designs, we have to tell Quartus to use VHDL 2008 as well.

Step 28: Right-click on “my_first_component” in the “Project Navigator” pane and select “Settings…”.

Step 29: Select Compiler Settings › VHDL Input and select VHDL 2008 and click OK:



VHDL Input

Options for directly compiling or simulating VHDL input files. (Click on the EDA Tool Settings category to enter options for VHDL files generated by other EDA tools.)

VHDL version

( ) VHDL 1987
( ) VHDL 1993
[✓ VHDL 2008]

Library Mapping File

File name:

[ ] Show information messages describing LMF mapping during compilation

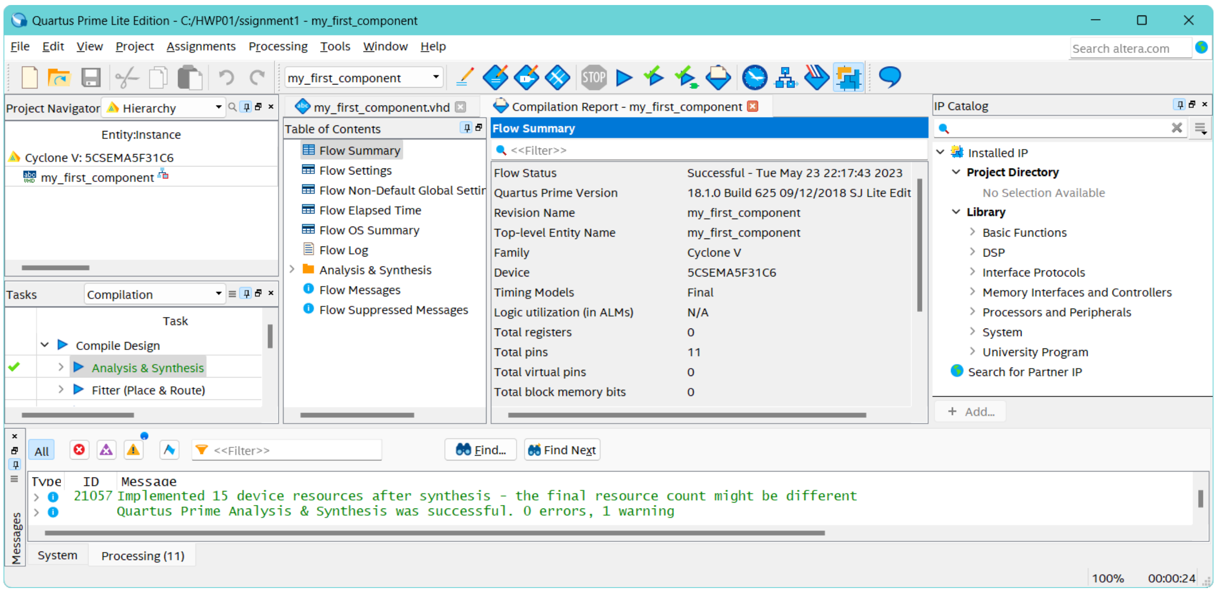
5 Assigning signals to physical pins

The first thing to do now, is to compile the design in Quartus, to let Quartus know which signals we are using and which signals have to go to the outside world.

Step 30: In the “Tasks” pane, double-click “Analysis & Synthesis”.

If Quartus is done, it will show a green check mark next to the task you performed (if everything went OK). If there are any errors or warnings investigate them and determine if they need to be resolved.

Quartus should show this:



Flow Summary

Flow Status	Successful - Tue May 23 22:17:43 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edit
Revision Name	my_first_component
Top-level Entity Name	my_first_component
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	0
Total pins	11
Total virtual pins	0
Total block memory bits	0

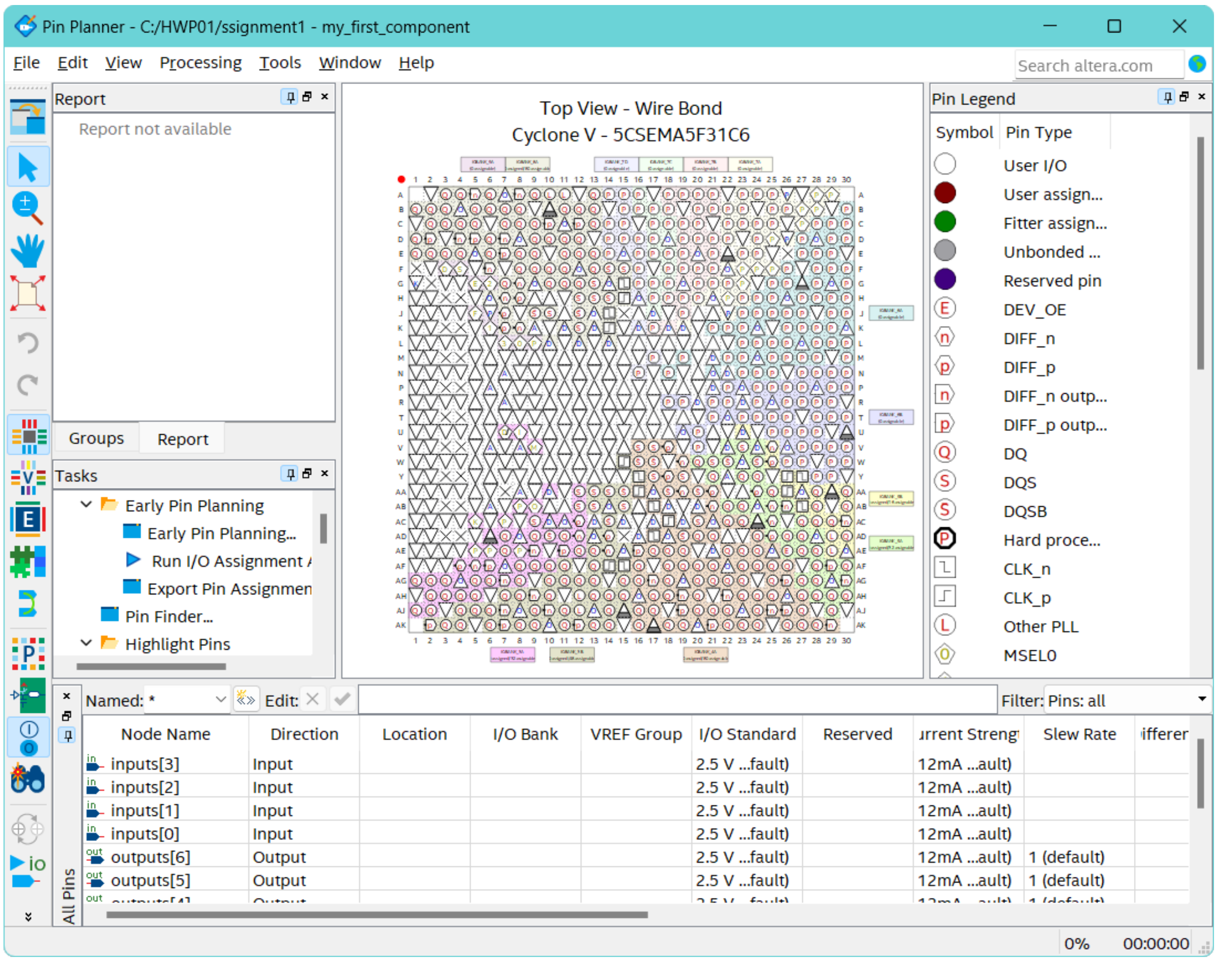
```
Type  ID    Message
      21057 Implemented 15 device resources after synthesis - the final resource count might be different
            Quartus Prime Analysis & Synthesis was successful. 0 errors, 1 warning
```

In the flow summary, we can see that our design uses 11 pins. The number of pins is correct, since we have 4 inputs and 7 outputs. This means Quartus properly detected which signals we want to connect to the outside world.

We will now assign real world pins of the FPGA to our signals.

Step 31: In the menu, select Assignments › Pin Planner.

You will now get the following window:



Symbol	Pin Type
○	User I/O
●	User assign...
●	Fitter assign...
●	Unbonded ...
●	Reserved pin
E	DEV_OE
n	DIFF_n
p	DIFF_p
n	DIFF_n outp...
p	DIFF_p outp...
Q	DQ
S	DQS
S	DQSB
P	Hard proce...
L	CLK_n
L	CLK_p
L	Other PLL
◇	MSEL0

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	urrent Streng	Slew Rate	ifferer
inputs[3]	Input				2.5 V ...fault)		12mA ...ault)		
inputs[2]	Input				2.5 V ...fault)		12mA ...ault)		
inputs[1]	Input				2.5 V ...fault)		12mA ...ault)		
inputs[0]	Input				2.5 V ...fault)		12mA ...ault)		
outputs[6]	Output				2.5 V ...fault)		12mA ...ault)	1 (default)	
outputs[5]	Output				2.5 V ...fault)		12mA ...ault)	1 (default)	

We see the BGA side of the FPGA chip that we've selected. Our goal is to connect our signals to one of the pins of the chip. We cannot just arbitrarily select some pins because our FPGA resides on a given circuit board. Luckily we have the schematics and a user manual for the board.

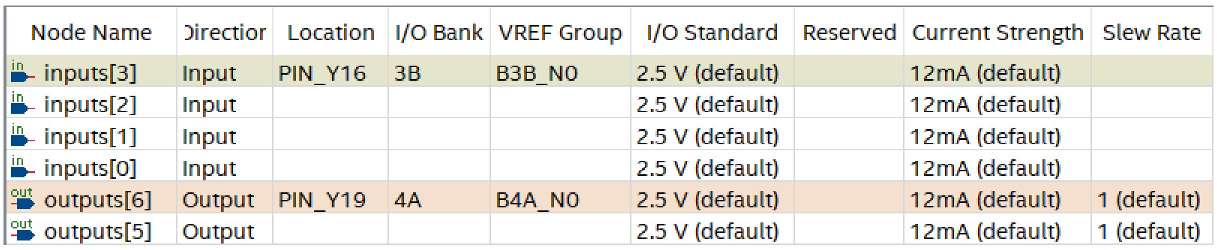
We see a list of so-called "nodes" which are actually our signals that have to get a physical connection to the outside world.

Our goal is to connect inputs 0 to 3 to keys 0 to 3 on the DE1-SoC board. Open the user manual[2] and find out at which location those keys are connected.

Do the same for the outputs 0 to 6. We want to connect these to the red LEDs 0 to 6. Once you know all the pin locations for the keys and the LEDs, fill in those locations in the column locations next to their respective nodes. You may just type the pin location instead of searching for it in the dropdown menu.

An example is shown for the first input and output in the list:

[2] For boards numbered 1 to 15 the DE1-SoC User Manual revisie C can be find here: `https://bitbucket.org/HR_ELEKTRO/hwp01/wiki/docs/DE1-SoC_User_manual_revd.pdf`. For boards numbered 16 to 50 the DE1-SoC User Manual revisie G can be find here: `https://bitbucket.org/HR_ELEKTRO/hwp01/wiki/docs/DE1-SoC_User_manual_revf.pdf`.



Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Current Strength	Slew Rate
inputs[3]	Input	PIN_Y16	3B	B3B_N0	2.5 V (default)		12mA (default)	
inputs[2]	Input				2.5 V (default)		12mA (default)	
inputs[1]	Input				2.5 V (default)		12mA (default)	
inputs[0]	Input				2.5 V (default)		12mA (default)	
outputs[6]	Output	PIN_Y19	4A	B4A_N0	2.5 V (default)		12mA (default)	1 (default)
outputs[5]	Output				2.5 V (default)		12mA (default)	1 (default)

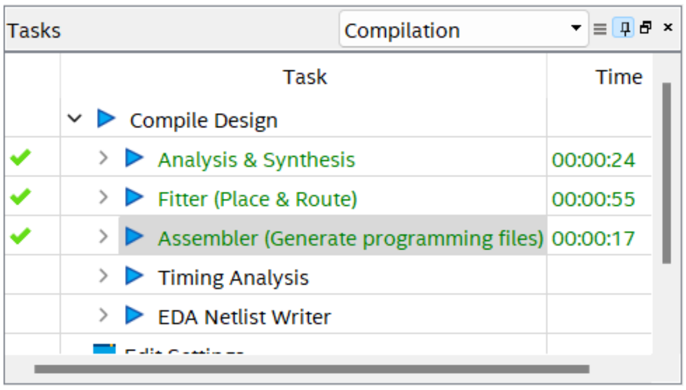
Once you are done connecting all nodes, you may close the pin planner.

6

Place and route the design for FPGA implementation

Now that all nodes are connected to some physical pin, we may let Quartus place and route our new design in the FPGA. This will generate a programming file with which the FPGA will be configured to work as we want.

Step 32: To do this, double-click on the Assembler task:



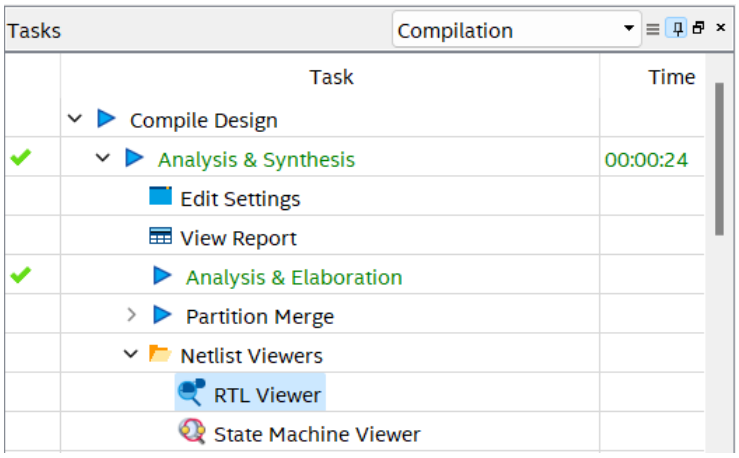
	Task	Time
	Compile Design	
✓	Analysis & Synthesis	00:00:24
✓	Fitter (Place & Route)	00:00:55
✓	Assembler (Generate programming files)	00:00:17
	Timing Analysis	
	EDA Netlist Writer	

This step usually takes a relatively long time, compared to compiling a small piece of software, for example. This is another reason why we want to simulate our design before implementing it into the FPGA.

If everything went OK, it should eventually display a pop-up that says the compilation is successful. Ignore any warnings for now.

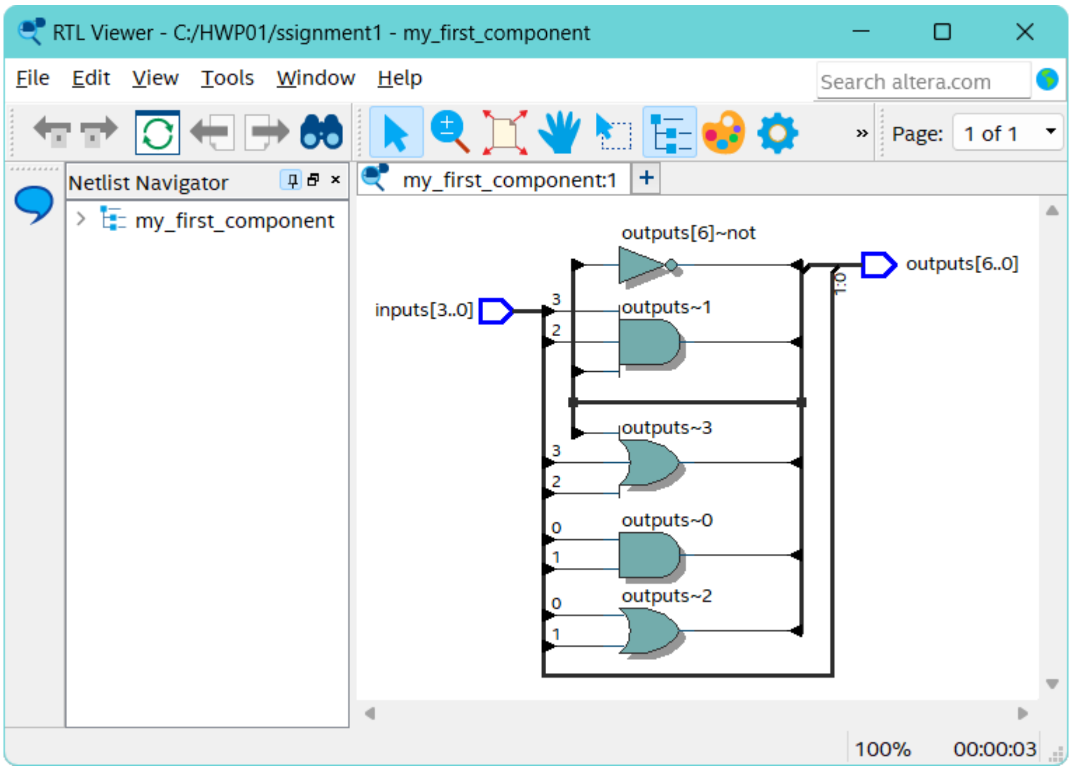
Some interesting things to check is the RTL schematic that Quartus has synthesized from your code.

You may find it here:



The RTL viewer will show the register transfer level schematic of your design.

It should show the following:

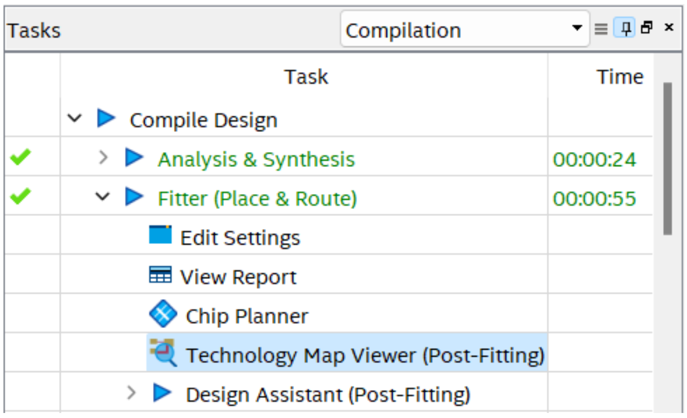


Here we may see the logic function that our architecture implements with the given inputs and outputs.

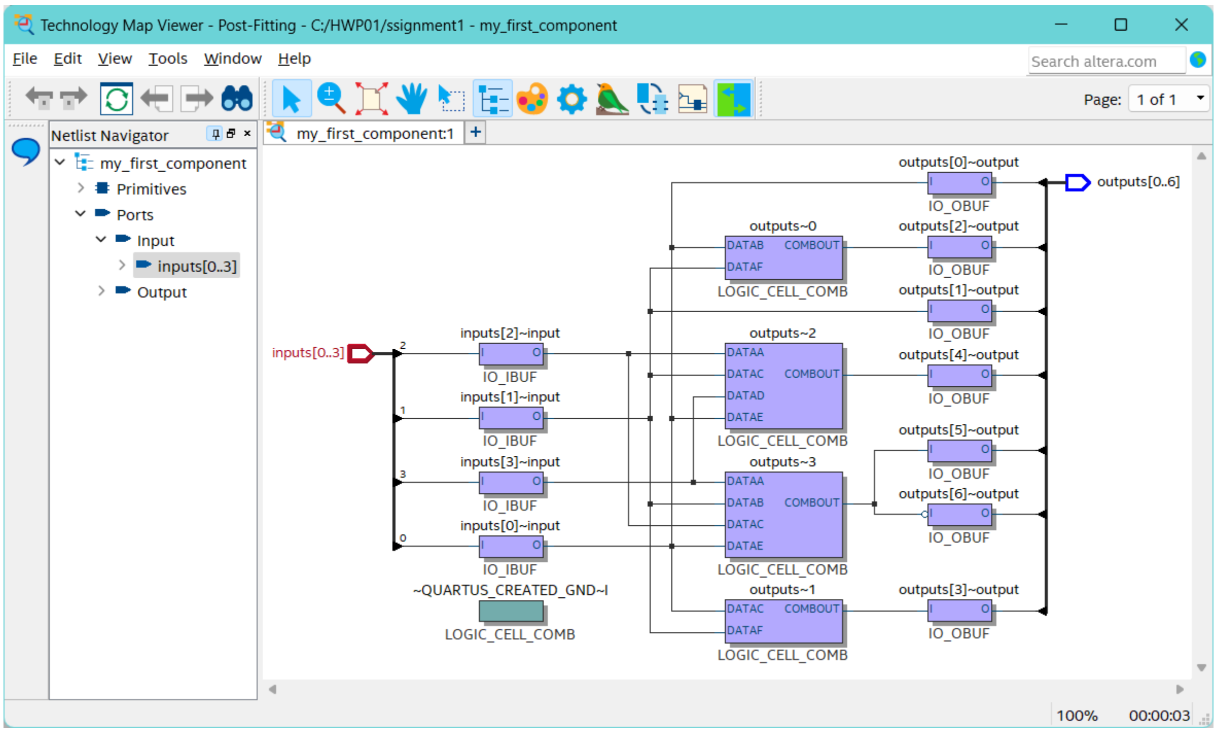
Eventually, writing VHDL code is about thinking how this schematic will look when you write a certain piece of code. (Usually you will think about your schematic in a more macroscopic way, and not as detailed as this.)

Another interesting view is the "Post-Fitting Technology Map" viewer. Here you can see how the logic elements and other hardware components in the FPGA are used to create your schematic.

It can be found here:



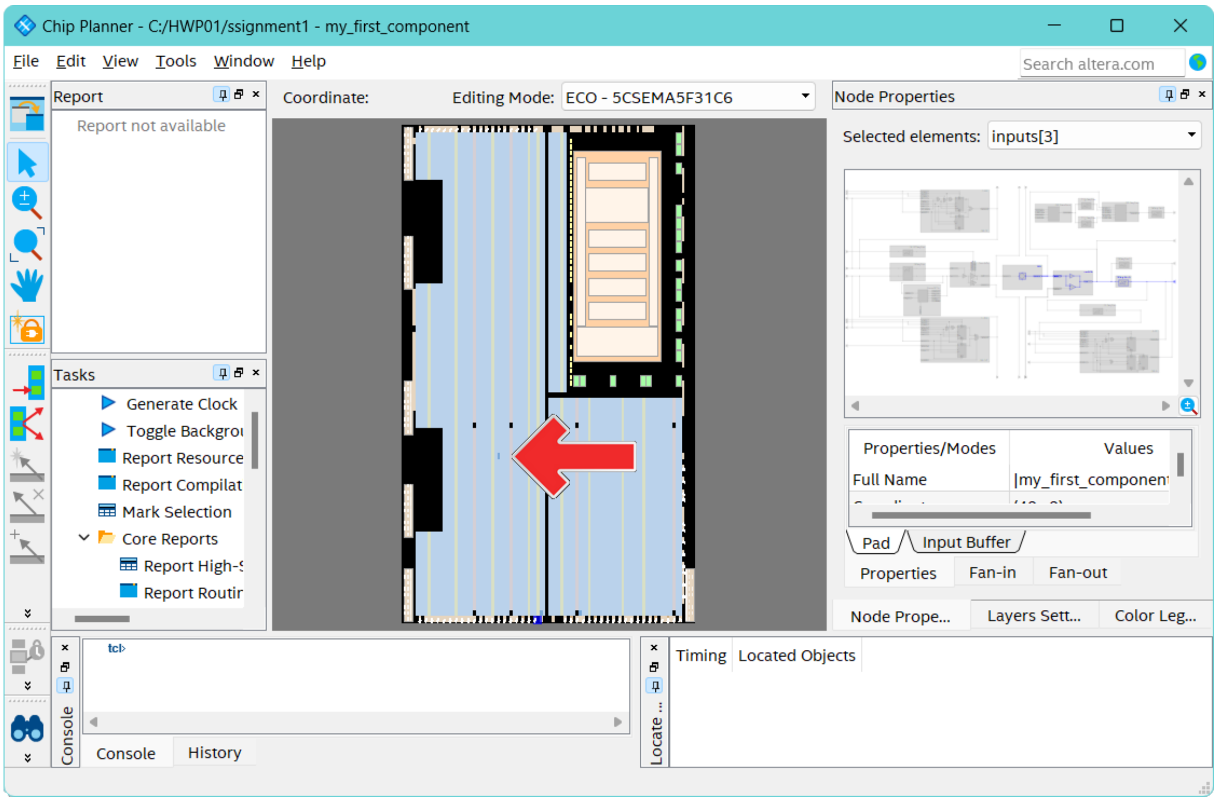
We can see the following:



Here we see that it uses just a few logic elements. This is only a tiny fraction of the number of logic elements available (over thirty thousand).

Another interesting thing to look at is the “Chip Planner” which can be found above the “Technology Map Viewer” task.

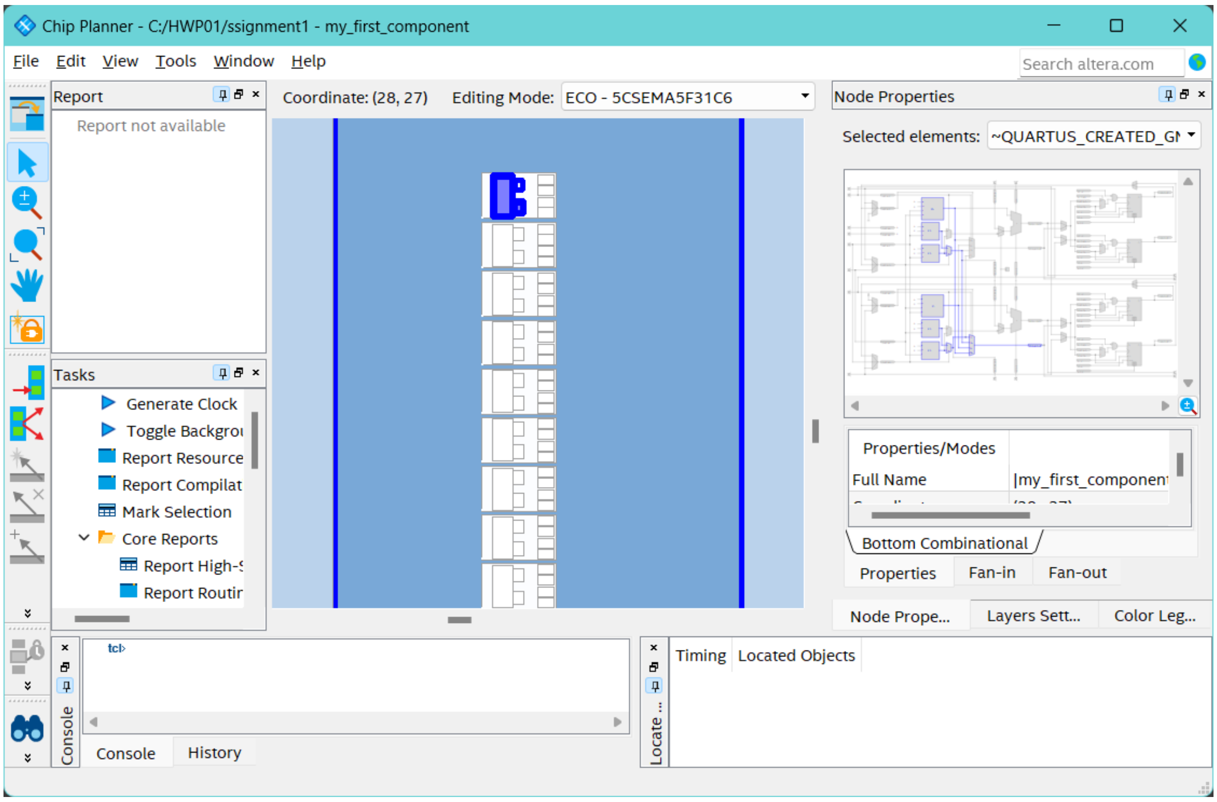
Here we can see the following:



This is a schematic view of the FPGA chip itself, and it also shows which elements of the chips are used. Only one region is used, and most of the time the unused regions are shut down to save power. The bigger your design is, the more power it will therefore use.

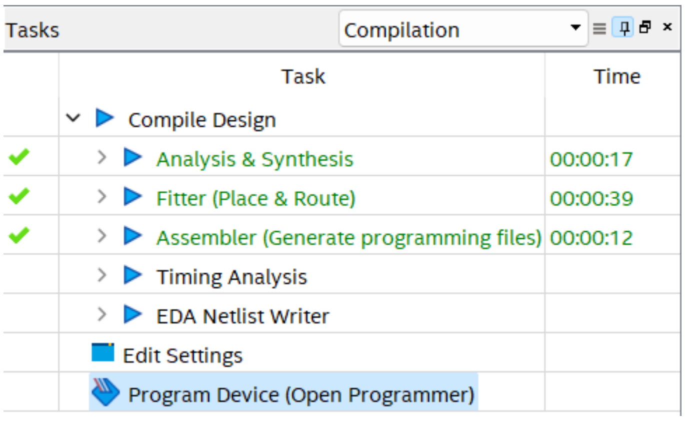
By, the way, the region that is used is not the black region. It is in the middle left, the bit more blueish rectangle, indicated by the red arrow.

If you zoom in on this, you can see the logic elements that are used:



For specialized and high-speed designs, the tools which you have just seen are extremely useful, because you can manually implement designs in the FPGA if you wish, so to optimize your design. In this course, we will not use them much (except for the RTL viewer).

Step 33: On the bottom of the Tasks list, double-click “Program Device”. This will open the “Programmer” window:

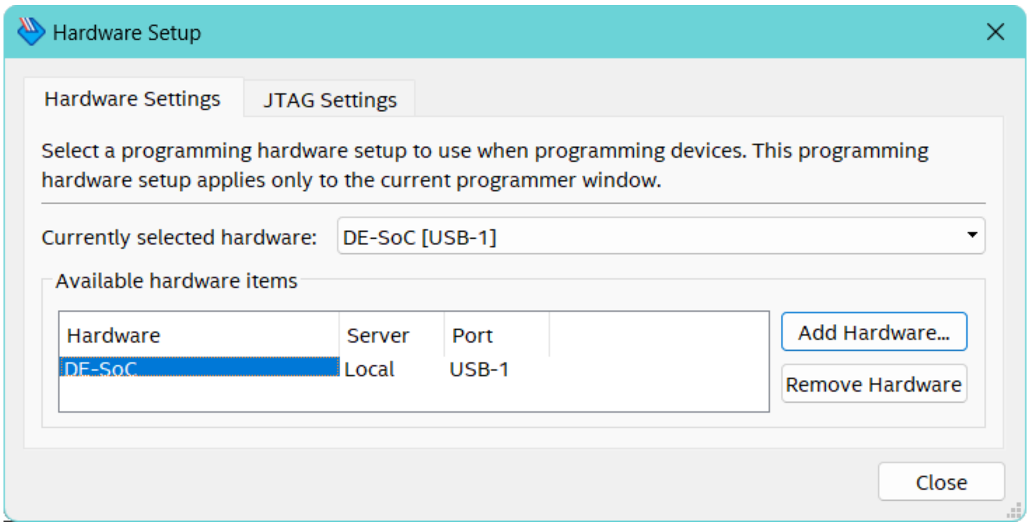


	Task	Time
	Compile Design	
✓	Analysis & Synthesis	00:00:17
✓	Fitter (Place & Route)	00:00:39
✓	Assembler (Generate programming files)	00:00:12
	Timing Analysis	
	EDA Netlist Writer	
	Edit Settings	
	Program Device (Open Programmer)	

Connect the DE1-SoC kit to the computer using the USB Blaster connection on the board, and also connect it to a power supply, and turn on the board[3]. Under the `Hardware Setup`

[3] Typically, the USB Blaster driver software is installed when installing Quartus, but if this is not the case a wizard will pop up to install the driver for the USB Blaster. Choose “Browse my computer for drivers”, and select the location: `C:\intelFPGA_lite\18.1\quartus\drivers`. A more detailed explanation can be found here: https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Tutorials/Getting_Started_with_DE-series_boards.pdf#section.4.

button in the Programmer window, make sure the currently selected hardware is set to the DE-SoC.


Hardware Setup

Hardware Settings | JTAG Settings

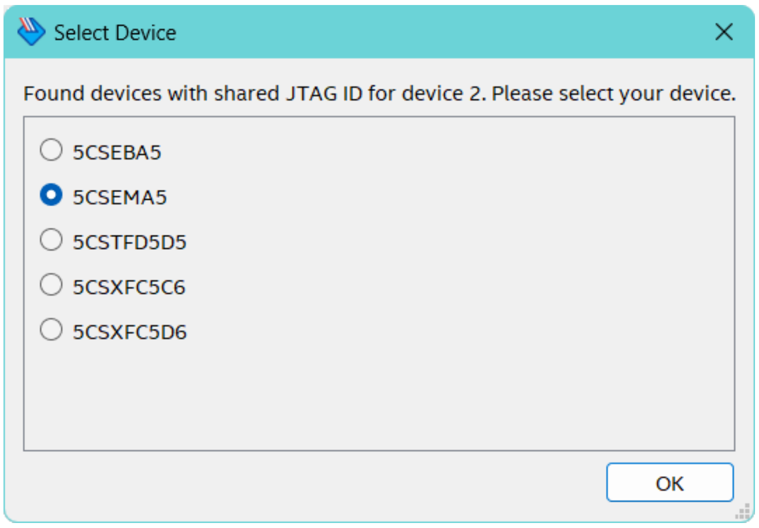
Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: DE-SoC [USB-1]

Available hardware items

Hardware	Server	Port
DE-SoC	Local	USB-1

Step 34: Now close the Hardware Setup window and click Auto-Detect in the Programmer window. The “Select Device” window appears:

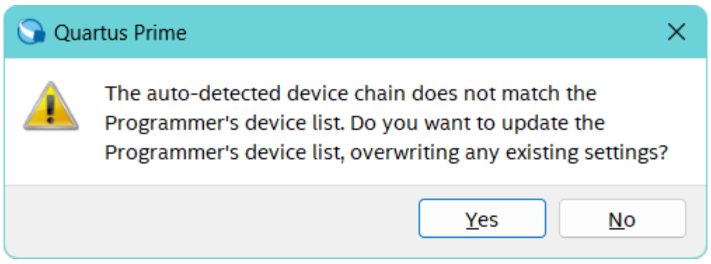

Select Device

Found devices with shared JTAG ID for device 2. Please select your device.

- 5CSEBA5
- [✓ 5CSEMA5]
- 5CSTFD5D5
- 5CSXFC5C6
- 5CSXFC5D6

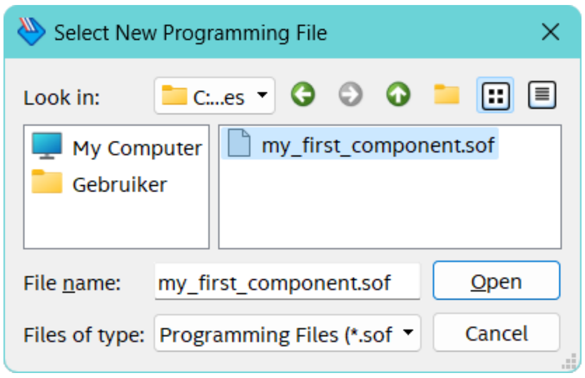
Step 35: Select 5CSEMA5 and click OK.

Step 36: If the following window pops up, click Yes:


Quartus Prime

The auto-detected device chain does not match the Programmer's device list. Do you want to update the Programmer's device list, overwriting any existing settings?

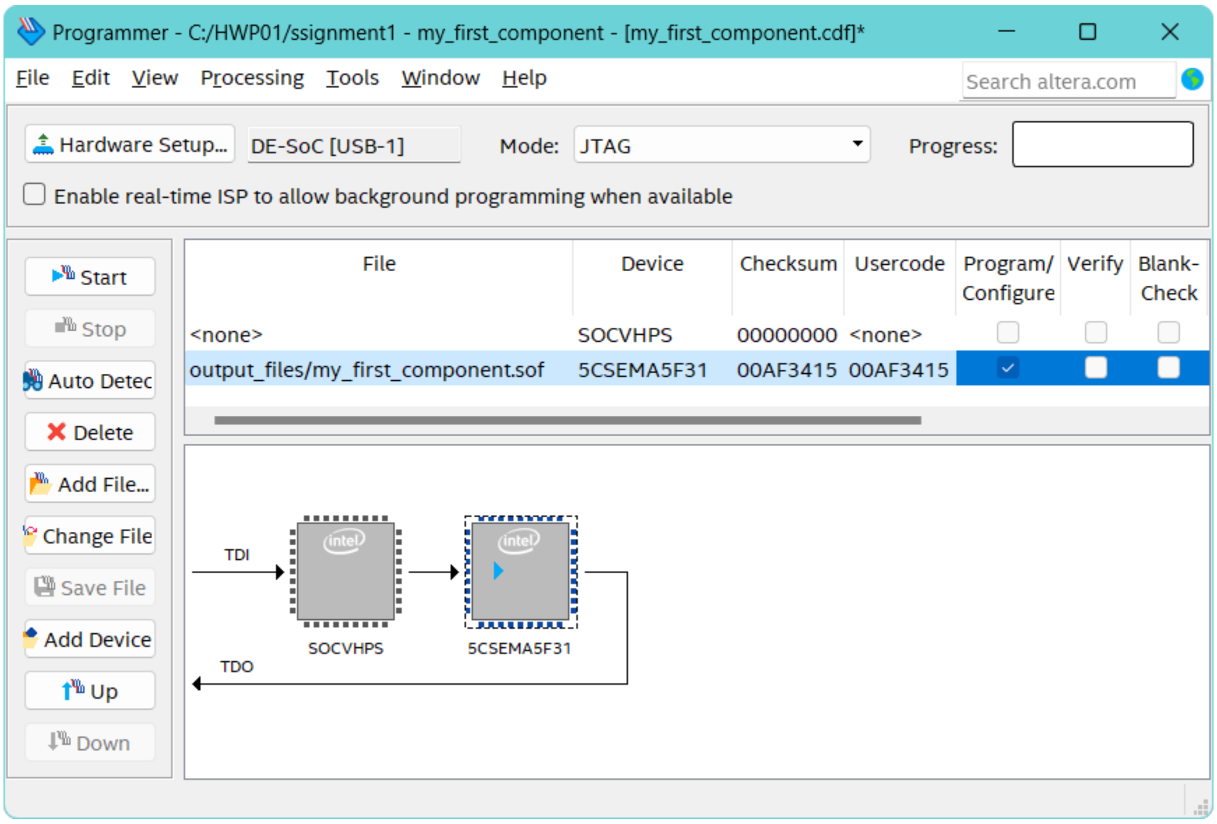
Step 37: Double-click under the "File" column next to device 5CSEMA5 and select `my_first_component.sof` in the `output_files/` folder of your project:



File name: my_first_component.sof

Files of type: Programming Files (*.sof

Step 38: Also check the "Program/Configure" checkbox next to the device.



DE-SoC [USB-1]

Mode: JTAG

☐ Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check
<none>	SOCVHPS	00000000	<none>	☐	☐	☐
output_files/my_first_component.sof	5CSEMA5F31	00AF3415	00AF3415	✓	☐	☐

By doing this, you are letting the programming interface know that the bitstream to be programmed in the FPGA is in `my_first_component.sof`. The extension `.sof` stands for SRAM Object File.

We can see the boundary scan chain that will be used to program our devices. In this case we will only program the FPGA chip, so only this one is seen in the chain.

Step 39: Look at the DE1-SoC kit while pressing start.

Step 40: While loading the new FPGA configuration, the JTAG TX LED should be on that shows a data transfer is in progress in the JTAG boundary scan chain.

Your new design will be programmed into the FPGA.

Notice that LEDR0 to LEDR5 are on. Try all the combinations with KEY0, KEY1, KEY2, and KEY3. Compare the LED values with the expected values given in the truth table you filled in at step 13.

As you can see, the inputs are somehow inverted compared to the truth table. Find out why this has happened (check the DE1-SoC manual where the switches are described). Change the VHDL code, so that a pressed key is interpreted as a logical one.

Congratulations, you have designed, simulated and programmed (probably) your first FPGA design!

Consider what questions arose while doing this assignment. Think about what feedback you want to receive from or give to your instructor. Now call your instructor to ask your questions and receive and give feedback. Your instructor will then sign off this assignment.

7 Summary

We have now finished our first assignment and our first design. We have done the following:

- created a new project in ModelSim and Quartus;
- made a VHDL design;
- simulated our design with ModelSim;
- connected the pins of our top-level design to the real FPGA pins;
- generated programming files and programmed them into the FPGA.

You are now ready to move on to the next assignments. You can use this manual as a reference for making new projects and simulations.

Good luck with the rest of the course!

8

Tips and further reading

Engineering tools are often very versatile and offer a lot of functionality. All those buttons and settings might confuse you a bit in the beginning. It is always handy to keep user manuals and tutorials at hand. Most tools have a user community with forums. On these you can often find questions that arise already asked and answered. If not, of course you can always post a new question yourself.

Quartus Help

In Quartus you can click Help > Search and type in anything you want to know about in Quartus.

ModelSim

In ModelSim you can click Help > PDF Documentation > User Manual to find out about ModelSim in general.

TCL Scripts (DO Files) for ModelSim

You can open the Reference manual to see what commands are available for the DO files (TCL scripting). Also you can see what syntax the commands we've used above use, so you can experiment with those. There are also examples given.

Hierarchical Designs

If you want to create hierarchical designs (this is recommended for most exercises) read chapter 10.3 of the book [1] for more information on the `component` keyword.

Bibliography

[1] Volnei A. Pedroni. *Circuit Design and Simulation with VHDL, Third Edition*. 2nd. The MIT Press, 2020. ISBN: 978-0-262-04264-2 (cit. on p. 33).