

#### Lab Work Handbook

Version 3.1

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



Lab Work Handbook *Programmable Hardware* from Rotterdam University of Applied Sciences is licensed by Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Netherlands license.

**Rotterdam University** 

#### Contents

In	troduction	4
1	Purpose and prerequisites	6
2	Creating a ModelSim Project	7
3	Simulating your design with ModelSim	13
4	Creating a Quartus project	19
5	Assigning signals to physical pins	25
6	Place and route the design for FPGA implementation	28
7	Summary	39
8	Tips and further reading	40
Bi	bliography	42

#### 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<sup>1</sup>) 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.

<sup>&</sup>lt;sup>1</sup> See: https://www.researchandmarkets.com/reports/5398214/fpga-market-by-configuration-lowend-fpga-mid

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

## **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 FPC	GA STARTE	EDITION 10.5b	-	×
<u>File Edit View Comp</u>	ile <u>S</u> imu	ate A <u>d</u> d L <u>i</u> brary T <u>o</u> ols Layo <u>u</u> t Boo <u>k</u> marks <u>W</u> indow <u>H</u> elp		
🖻 • 🚅 🖬 🦈 🍈	X 🖻 🕯	1 🗠 🗠   🖉 - 🗛 🖺 🗰 🛛 🕸 🖄 🔛 🗛 🕱 🛛 ColumnLayout AllColumns		
🛛 🖏 - 📭 - 🥵 🖏 - 🗳		0 150 🛐 🚛 🌽   🛊 🏞 🋊   🏦 - 🏤 🏦   Layout NoDesign 🗨		
Library				 H at X
▼ Name	Туре	Path		<b>^</b>
+-1 220model	Library	\$MODEL_TECH//altera/vhdl/220model		
+-1 220model ver	Library	\$MODEL_TECH//altera/verilog/220m		
	Library	\$MODEL_TECH//altera/vhdl/altera		
+ dltera_Insim	Library	\$MODEL_TECH//altera/vhdl/altera_l		
+ altera_Insim_ver	Library	\$MODEL_TECH//altera/verilog/altera		
+ dlera_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		_
🛨 👖 arriai	Library	\$MODEL_TECH//altera/vhdl/arriaii		
	Library	\$MODEL_TECH//altera/vhdl/arriaii_hssi		
	Library	\$MODEL_TECH//altera/verilog/arriaii		
	Library	\$MODEL_TECH//altera/vhdl/arriaii_p		
	Library	\$MODEL_TECH//altera/verilog/arriaii		
	Library	\$MODEL_TECH//altera/verilog/arriaii		
	Library	\$MODEL_TECH//altera/vhdl/arriaiigz		
⊕_ arriaiigz_hssi	Library	\$MODEL_TECH//altera/vhdl/arriaiigz		
	Library	\$MODEL_TECH//altera/verilog/arriaii		
	Library	\$MODEL_TECH//altera/vhdl/arriaiigz		
I → I arriaiigz_pcie_hip_v	. Library	\$MODEL_TECH//altera/verilog/arriaii		
	Library	\$MODEL_TECH//altera/verilog/arriaiigz		
💽 👖 arriav	Library	\$MODEL_TECH//altera/vhdl/arriav		
	Library	\$MODEL_TECH//altera/verilog/arriav		
diametria arriav_pcie_hip_ver	Library	\$MODEL_TECH//altera/verilog/arriav		
arriav_ver	Library	\$MODEL_TECH//altera/verilog/arriav		
	Library	\$MODEL_TECH//altera/vhdl/arriavgz		-
0				
H Transcript				
# Reading C:/intelF	PGA/18.1/	modelsim_ase/tcl/vsim/pref.tcl		-
Madalows				
model5Im>				
				-
J				
<no design="" loaded=""></no>		\$MODEL_TECH//altera/verilog/arriav		1

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		
assignmentl		
Project Location		
C:/HWP01		Browse
Default Library Name		
work		
Copy Settings From		
modelsim_ase/modelsim.ini	Brov	vse
	Librar	y Mappings
	OK	Cancel

#### 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:



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		Browse
Add file as type	Folder	
VHDL VHDL	Top Level	<b>_</b>
	Oł	Cancel

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

Now ModelSim will show the project window:



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.

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



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:



Note that the status of the component has also changed to a green  $\checkmark$ . This means it compiled successfully. If it didn't compile successfully it would show a red  $\checkmark$ . 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;</pre>
```

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 Model-Sim

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

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

Start Simulation		
Design VHDL Verilog	Libraries SDF	Others
* Name	Туре	Path
	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_Insim	Library	\$MODEL_TECH//altera/vhdl/altera_l
Design Unit(s)		Resolution default
		OK Cano

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

**Rotterdam University** 

see which implementations are available. In our case, we have only one implementation which is called "implementation".

M Start Simulation		;	×
Design VHDL Verilog Libra	aries ) SDF	Others ]	*
▼ Name	Туре	Path	
work	Library	work	4
E my_first_component	Entity	C:/HWP01/my_first_component.vhd	
A implementation	Architectur	re 🛛	
+ 220model	Library	\$MODEL_TECH//altera/vhdl/220model	
+ 220model_ver	Library	\$MODEL_TECH//altera/verilog/220m	ıI.
Design Unit(s)		Resolution default	
		OK Cance	1

**Step 15:** Select "my\_first\_component" and click OK to start the simulation.

You should now see the following window:

ModelSim - INTEL FPGA STARTER EE	ITION 10.5b		– 🗆 X
<u>File Edit View Compile Simulate</u>	A <u>d</u> d T <u>r</u> anscript T <u>o</u> ols Layo <u>u</u> t Boo <u>k</u> ma	rks <u>W</u> indow <u>H</u> elp	
🛛 🖬 • 🚅 🔛 🛸 🎒   🗼 🛍 🛍 🖄	2 🗈   💿 - 🗛 🏝 🗖   🤣 🛱 🛱 🧛	X	
🕒 👚 🦛 🖦   📑 🛛 100 ps 🝨	El El El 🕺 😩   🏦 🖺 🖉 🛛 Colum	nLayout AllColumns	
X X 🕅 🌾 🛛 I O 10 🗈	🔝 🖗 🛛 🦓 - 🖓 - 🤣 📑 - 🥵 🎽 🏌	🕀 🛊 🗄 🕇 🗙 🏦 📗 Layou	t Simulate 💌
🌆 sim - Default 🛨 🛃 🗙	💫 Objects 🖬 📩	wj_first_component.vhd (/my_first	t_component) - Default 🕬 🛨 🖻 🗙
▼ Instance Design unit	▼Name Value 💽 Now 🛧 🕨	Ln#	1 🗠 Now 🔊 🕨
my_first_comp my_first_com	🛨 🍫 inputs UUUU Signal In	<pre>1 library ieee;</pre>	
— line13 my_first_com	🛨 🚣 outputs UUUUUUU Signal Out	2 use ieee.std_lo	ogic_1164.all;
— line14 my_first_com		3	
line15 my_first_com		5 D port (	_component 18
line_16 my_first_com		6 inputs	in std plogic vector
line 18 my first com		7 outputs	: out std ulogic vecto
line 19 mv first com		8 - );	
standard standard		9 end my_first_c	component;
📕 textio 🛛 🛛 🗖	🐡 Processes (Active) 🛨 🛃 🗙	10 L	
<pre>std_logic_1164 std_logic_1164</pre>	▼ Name Type (filtered) State ▲	11 architecture i	mplementation of my_1
	line_19 VHDL Process Activ	12 🛛 begin	
	line_18 VHDL Process Read	13 outputs (0)	<= inputs(0);
	line_17 VHDL Process Read	14 outputs (1)	<= inputs(1);
	line16 VHDL Process Read	16 outputs (2)	<= inputs(0) of in
	Ine_15 VHDL Process Read	17 outputs (4)	<= inputs(0) and int
• •	line 12 VHDL Process Read	18 outputs (5)	<= inputs(0) or int
Ibrary × I Project × D sim *	The 15 who Process Read		
The second secon	· · ·		
🔒 Transcript ======			
ModelSim > vsim -gui work.my_first	_component		
# vsim -gui work.my_first_comp	onent		
# Start time: 12:06:32 on May	13,2023		
# Loading std.standard			
# Loading Std. text10 (Dody)	(who		
<pre># Loading work.my_first_compone</pre>	ent(implementation)		
VSIM 2>]			•
	Project : assignment1 N	ow: 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:



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

Force Selected Signal		×		
Signal Name: sim:/my_first_component/input				
Freeze O Drive O Deposit				
Delay For: 0				
	ОК	Cancel		

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):

📰 Wave - Default 🚃			ar X
<b>€1</b> •	Msgs		
	0000	0000	<b>_</b>
▶ /my_first_component/outputs	1000000	1000000	⊸
A 📰 💿 Now	0.0000001 ms	000 ms	
🔓 🌽 🤤 Cursor 1	0.0000000 ms	0.0000000 ms	
×	. →	۱ ۱	
M my_first_component.vhd 🗶 👥 Wa	ve ×		< >

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

/my_first_component/inputs	0000	0000
(3)	0	
-4 (2)	0	
-4 (1)	0	
L_\$ (0)	0	
	1000000	1000000
	1	
(5)	0	
	0	
(3)	0	
(2)	0	
	0	

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:



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:





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						
amily, Device	& Board Settir	igs				
Device Board						
Select the family an You can install addi	d device you want to 1 tional device support	arget for con with the Insta	npilation. all Devices comn	nand on the Too	ls menu.	
To determine the ve	ersion of the Quartus I	Prime softwa	re in which your	target device is	supported, refer to the	Device Support List webpage
Device family				Show in 'Availa	ble devices' list	
Eamily: Cyclone	/ (E/GX/GT/SX/SE/ST)		•	Pac <u>k</u> age:	Any	•
Device: All			•	Pin <u>c</u> ount:	Any	•
Target device				Core speed grade: Any		
O <u>A</u> uto device se	lected by the Fitter			Name filter:		
Specific device	e selected in 'Available	devices' list		🗹 S <u>h</u> ow advar	nced devices	
◯ <u>O</u> ther: n/a						
A <u>v</u> ailable devices:						
Name	Core Voltage	ALMs	Total I/Os	GPIOs	GXB Channel PMA	GXB Channel PCS
5CSEMA4U23I7	1.1V	15880	314	314 (	)	0
5CSEMA5F31A7	1.1V	32070	457	457 (	)	0
5CSEMA5F31C6	1.1V	32070	457	457 (	)	0
5CSEMA5F31C7	1.1V	32070	457	457 0	)	0

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

New F	Project Wizard										
Family, Device & Board Settings											
Devic	e Board										
Selec	t the board/development kit you want to	target for com	pilation.								
Avail	able boards:		Developin	Any							
_	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				
<b>=</b>	Cyclone V SoC Development Kit	1.0	Cyclone V	5CSXFC6D6F31C6	Altera	41910	499				
<b>=</b>	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:



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:

-	Settings - my_first_component	- o x
c	ategory:	Device/Board
	General	VHDL Input
	Files	Options for directly compiling or simulating VHDL input files. (Click on the EDA Tool Settings
	Libraries	category to enter options for VHDL files generated by other EDA tools.)
1	✓ IP Settings	
	IP Catalog Search Locatior	VHDL version
	Design Templates	O VHDL 19 <u>8</u> 7
1	<ul> <li>Operating Settings and Cond</li> </ul>	
	Voltage	
	Temperature	• VHDL <u>2</u> 008
	<ul> <li>Compilation Process Settings</li> </ul>	
	Incremental Compilation	Library Mapping File
	✓ EDA Tool Settings	File name:
	Design Entry/Synthesis	
	Simulation	□ Snow information messages describing LMF mapping during compilation
	Board-Level	
	Compiler Settings	
	VHDL input	

# 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:

Quartus Prime Lite Edition - C:/HWP01/ssignme	ent1 - my_first_component			-				
Elle Edit View Project Assignments Processing Tools Window Help Search altera.com								
□ Image: Im								
Project Navigator 📥 Hierarchy 🔹 🤉 🖉 🛪	🔶 my_first_component.vhd 🖾	Compilation Report - my_first	t_component 🛛	IP Catalog	📮 🗗 ×			
Entity:Instance	Table of Contents 🔍 🗗	Flow Summary		۹.	× =			
Cyclone V: 5CSEMA5E31C6	🖽 Flow Summary	< <ri>Filter&gt;&gt;</ri>		🗸 🍓 Installed IP				
my first component	Flow Settings	Flow Status	Successful - Tue May 23 22:17:43 2023	<ul> <li>Project Directory</li> </ul>				
	🏛 Flow Non-Default Global Settir	Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edit	No Selection Available				
	Flow Elapsed Time	Revision Name	my_first_component	✓ Library				
	Flow OS Summary	Top-level Entity Name	my_first_component	> Basic Functions				
	Flow Log	Family	Cyclone V	> DSP				
	Analysis & Synthesis	Device	5CSEMA5F31C6	> Interface Protocols				
Tasks Compilation 💌 🗏 🖉 *	Flow Messages	Timing Models	Final	> Memory Interfaces and Co	ntrollers			
Task _	Flow Suppressed Messages	Logic utilization (in ALMs)	N/A	> Processors and Peripheral	5			
Compile Design		Total registers	0	> System				
Analysis & Synthesis		Total pins	11	> University Program				
Fitter (Place & Deute)		Total virtual pins	0	Search for Partner IP				
		Total block memory bits	0	t Add				
				- Add				
x     all     O     A     A     Y     <								
Type ID Message	necources often comthesis	the final necounce of	count might be different					
> 0 Quartus Prime Analys	is & Synthesis was successf	Ful. 0 errors, 1 warning	) )		· · · ·			
ssa								
System Processing (11)								
				100%	00:00:24			

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:

HWP01

∛ P	in Pla	anner - C	:/HWP01/ssig	nment1	- my_first_o	component						-		×
ile	<u>E</u> dit	<u>V</u> iew	P <u>r</u> ocessing	<u>T</u> ools	Window	<u>H</u> elp						Search alt	era.con	n5
	Repo	ort		<u></u> Ф	×		Top	/iew - Wire B	ond		Pin Lege	nd		
	R	eport no	ot available				Cyclone	V - 5CSEMA	5F31C6		Symbol	Pin Type		
							E-DF_M. KADK_A. exignality (secigred) & exignality	KAVA 70 GANK/X Exclusion English	KADH/B KADH/3. Socipatel Socipatel			User I/O		- 1
									20 21 22 23 24 25 26 2	7 28 29 30		User assign.		- 1
2												Fitter assign		- 1
W/								V O O O O O O	ŎŎŎŎŎŎŎŎ			Unbonded		- 1
							70000000			7000		Deserved ni		- 1
Д,									ŎĂŎŎŎŎĂ		E			- 1
_														- 1
5							)X <del>XXX</del> QXC	XQXQXQX				DIFF_N		- 1
C							X <del>QXXQ</del> XC	XAX®X®X			12	ынг_р		
							YXAXXAXA					DIFF_n outp		
:	Gr	oups	Report			. XXXX	YXXXXXXXXX	, XAXXXXA	<u>AVAX88</u>			DIFF_p outp		
	Tack					w XXXX	SOXXXXAAX	VAnoovo	000A000	Võõ .	Q	DQ		
- <u>v</u> -	Task			<u> </u>			<u> </u>		ÅõQVõ§ Å⊟Z		S	DQS		
E		Ear	'ly Pin Plannii 5 d. oʻ. ol	ng	11					J©V©^ª D©©⊡~	S	DQSB		
-			Early Pin Plar	nning							P	Hard proce.		
# I			Run I/O Assi	gnment /	•						1	CLK_n		
3		_	Export Pin As	signmer	1		ŎŎŎŎŎŎŎĊ				F	CLK_p		
_		Pin	Finder			AK 0001	700000000	©õ∆õõ	AD000V00	ÓĎ ×		Other PLL		
Ρ	Ň	🖊 📂 Hig	hlight Pins			1234	S 6 7 8 9 10 11 12 CRUM, M. CRUM, M. CR		20 21 22 23 24 25 26 2. Rose, a apert Kinige 45	7 28 29 30	0	MSEL0		
	a	Named:	* <u> </u>	» Edit: 2	× 💉							Filter: Pins: a	u	•
	Ţ	Noc	le Name	Direc	tion L	ocation	I/O Bank	VREF Group	I/O Standard	Reserved	irrent Stre	ngi Slew R	ate i	fferer
-		🖫 input	:s[3]	Input					2.5 Vfault)		12mAau	lt)		
00	1	- input	s[2]	Input					2.5 Vfault)		12mAau	lt)		
6		input	:s[1]	Input					2.5 Vfault)		12mAau	lt)		- 1
£.			S[U]	Output					2.5 Vfault)		12mAau	lt) 1 (defaul	4	
> 10	SU		uts[5]	Output					2.5 V fault)		12mA au	lt) 1 (defaul	() ()	
_	II Pi	out	[4]	Output					DEM fault)		12	+) 4 (dofaul	4	
¥	A	_										0%	00:	00:00

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<sup>2</sup> 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:

Node Name	Directior	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)	
Ӵ 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.

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.

# 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:



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:

Task	s Compilation	▼ = ₽ ₽ ×
	Task	Time
	✓ ► Compile Design	
<ul> <li>Image: A second s</li></ul>	✓ ► Analysis & Synthesis	00:00:24
	Edit Settings	
	🖽 View Report	
<	Analysis & Elaboration	
	> Partition Merge	
	💙 芦 Netlist Viewers	
	🗨 RTL Viewer	
	🍳 State Machine Viewer	

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

It should show the following:

RTL Viewer - C:/HWP01/ssignment	1 - my_first_component	—		×
<u>F</u> ile <u>E</u> dit <u>V</u> iew <u>T</u> ools <u>W</u> indow	<u>H</u> elp	Search	altera.com	6
	📐 🔍 💢 🕊 🔚 🔂 🌣 .	<b>»</b>	Page: 1 o	f1 🔻
Netlist Navigator 🛛 🗜 🖉 🔍	my_first_component:1 +			
> 🔚 my_first_component	inputs[30]		outputs[6	0]
		-		
		100	0:00	00:03

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:

Task	S	Compilation	- = ₽ ₽	×
	Task		Time	i.
	✓ ► Compile Design		L	
<ul> <li>Image: A second s</li></ul>	> 🕨 Analysis & Synthes	00:00:24	L	
<ul> <li>Image: A second s</li></ul>	🛩 🕨 Fitter (Place & Rout	00:00:55	L	
	Edit Settings			5
	📅 View Report			
	🔇 Chip Planner			
	🂐 Technology Ma	o Viewer (Post-Fitting)		
	🔰 🕨 Design Assistan	t (Post-Fitting)		

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:



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<sup>3</sup>. Under the Hardware Setup button in the Programmer window, make sure the currently selected hardware is set to the DE-SoC.

G Settings									
Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.									
e: DE-SoC [	USB-1]	•							
Server	Port	Add Hardware							
Local	USB-1	Remove Hardware							
		Close							
	ware setup to ly to the curre re: DE-SoC [ Server Local	ware setup to use when programmir y to the current programmer windov e: DE-SoC [USB-1] Server Port Local USB-1							

<sup>&</sup>lt;sup>3</sup> 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.

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

V Select Device X
Found devices with shared JTAG ID for device 2. Please select your device.
O 5CSEBA5
SCSEMA5
○ 5CSTFD5D5
⊖ 5CSXFC5C6
⊖ 5CSXFC5D6
ОК

Step 35: Select 5CSEMA5 and click OK.

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



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:



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

♦ Programmer - C:/HWP01/ssignment1 - my_first_component - [my_first_component.cdf]*									
<u>F</u> ile <u>E</u> dit <u>V</u> iew	ile <u>E</u> dit <u>V</u> iew P <u>r</u> ocessing <u>T</u> ools <u>W</u> indow <u>H</u> elp <u>Search altera.com</u>								
Hardware Setup DE-SoC [USB-1] Mode: JTAG      Progress:     Enable real-time ISP to allow background programming when available									
▶ Start	File	Device	Checksum Usercode	Program/	Verify	Blank- Check			
📲 Stop	<none></none>	SOCVHPS	00000000 <none></none>						
<table-of-contents> Auto Detec</table-of-contents>	output_files/my_first_component.sof	5CSEMA5F31	00AF3415 00AF3415						
× Delete									
📥 Add File									
😤 Change File	intel								
🕒 Save File									
Add Device	SOCVHPS 5CSEMA5F3								
t <sup>™</sup> Up	TDO								
J <sup>™</sup> Down									

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. 41).