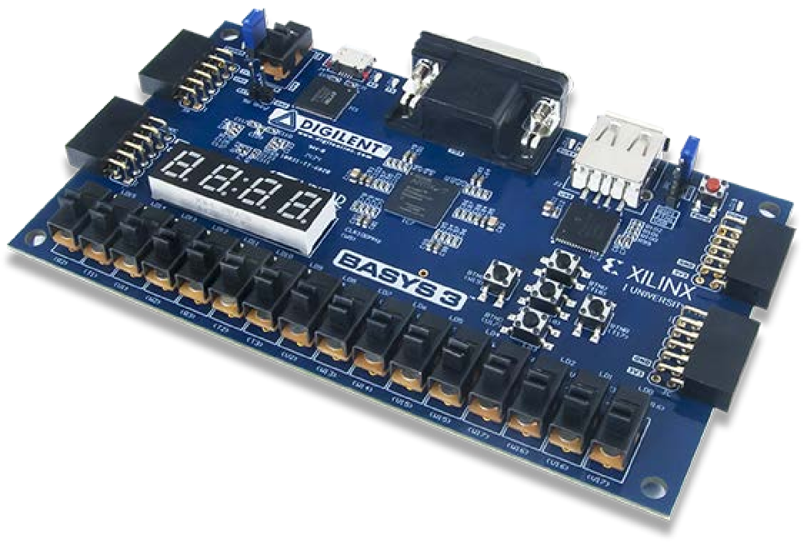
A digital circuit design flow using VHDL, ModelSim and XILINX Vivado targeting a DIGILENT BASYS 3 FPGA board

Edward Todirica, Jens Sparsø, Flemming Stassen and Rasmus Bo Sørensen

Abstract

This note is written to help the beginner implement digital circuits described in VHDL in FPGA technology using the ModelSim simulator and the XILINX Vivado synthesis tool. These tools are used in industry, they are extremely complicated, with lots of features that we will not use, and they can be used in many different ways. This note presents a specific design flow that we recommend students to follow. The note first describes the different steps of the design-flow in order to emphasize an understanding of *what* the tool actually does. Following this note provides you with a detailed description of *how* to use the tools in order to carry out these steps. This part is supported by screenshots and walks you through the entire design process step-by-step. Finally, for those who want to install the ModelSim and XILINX Vivado tools on their own PCs, the note provides a description of how to download and install the tools.



Contents

1. The design flow

The design of a digital circuit begins with a description of the circuit in a hardware description language (HDL), in this case VHDL. Next, the functionality of the description is explored and proper function verified in an iterative process using a circuit simulator, ModelSim. Once the functionality is validated, the circuit is synthesized into an implementation using XILINX tools, and then transferred to the FPGA board. More specific, we will use:

- The VHDL hardware description language.
- ModelSim for compiling and simulating digital circuits described in VHDL.
- The XILINX Vivado for synthesizing a circuit described in VHDL into an FPGA implementation.
- A Digilent BASYS 3 FPGA-board for prototyping.

1.1 Design flow perspective

VHDL is a powerful language for describing and documenting a circuit design offering a diversity of expression capabilities together with the capabilities of a programming language (such as Java or C++). In our design flow, we will appreciate neither of the above properties; rather, we will use VHDL to describe your intended circuit, simulate its functionality and finally, synthesize and implement the circuit on the XILINX FPGA board exactly as you intended. Our aim is that you understand the design flow including the transformations and optimizations performed by the tools used.

In course 02138 and subsequent courses we will be using the above-mentioned VHDL language and the ModelSim and Vivado tools. These tools are widely used in the electronics industry, and they are very complex tools to use. In fact, it is more correct to think of them as toolboxes, from which the designer will pick and use certain tools in some sequence (design flow) in order to arrive at a circuit implementation, which is functionally correct, and which satisfies other constraints like speed, power and area.

To a freshman taking his/her first course on digital electronics, the VHDL language and the ModelSim and Vivado-tools are overwhelmingly complicated – the manuals and user guides are thousands of pages.

In this note we will illustrate the fundamentals of a typical design-flow and limit ourselves to a simple design flow from specification to implementation. Later, several courses and design projects down the line, when you get more experience, you may develop personal preferences on what particular tools you use, as well as the order in which you use them. Along with this you will learn more of the advanced features in VHDL. But all this is outside the scope of course 02138.

Despite our aim to keep it simple, you will be exposed to many details and many screenshots. We do not expect you to remember all of these details. The important thing is that you understand the overall picture, and that you are able to explain the main steps of the design flow in your own words. When you later design your next circuit, you will probably need to go back and check the specific details.

1.2 Overview of the design flow

The design flow from you start with a VHDL description of your circuit until you have an implementation running on an FPGA board in the laboratory involves the following steps. In essence the tools automate the sequence of steps that you would go through if you were to perform the process by hand. Keep this in mind.

1. **Write and compile the VHDL** code describing your design (ModelSim)

   You may do this using your favorite text editor or use the editor that comes with ModelSim. You may have to correct possible syntax errors, and compile again.

   The result is a syntactically correct VHDL description of your design.

2. **Simulate the VHDL code (ModelSim)**

   To do this you need to specify a sequence of input signals (and possibly also the associated sequence of expected outputs). You may discover that your design does not exhibit the intended function. For a large design this is typically the case. Fix the errors, compile and simulate again. Repeat until simulation shows the intended circuit behavior.

   The result is a syntactically and functionally correct VHDL-file describing your design.

3. **Synthesize and implement the VHDL code (Xilinx Vivado)**

   This is a complex task involving many transformations and optimizations.

   In the first part of the process called *synthesis*, a circuit implementation using generic technology-independent components is derived from the VHDL code. This involves:

   a) Identifying flip-flops and combinatorial circuitry.

   b) Synthesizing the combinatorial circuitry by recognizing adders, multipliers, and other typical combinatorial circuit building blocks, and by minimizing the remaining combinatorial circuitry.

   The result is a schematic using flip-flops and generic technology independent combinatorial circuit building blocks.

   In the second part of the process, called *implementation*, the circuit is implemented using the specific hardware resources available on the FPGA. These resources are: flip-flops and LUTs to implement the components, and wire segments and switch-boxes to implement the signal wires in the circuit. The implementation process involves:

   c) Implementing the generic combinatorial circuit building blocks using LUTs. *The result is a schematic using only flip-flops and LUTs.*

   d) Binding the individual flip-flops and LUTs in the implemented circuit (schematic) to specific flip-flop and LUT instances in the FPGA chip.

   e) Binding the input and output signals of the circuit to specific IO-pins on the FPGA chip.

   f) Based on the above binding-decisions, it is finally possible to implement the signal wires in the circuit using the wire segments and the switchboxes available in the FPGA-chip.

   As the end result of all this, the tool generates a programming file that describes how the FPGA-chip should be configured in order to implement the design.

4. **Configure the FPGA-board (Vivado)**

   This is done by uploading the generated programming file to the FPGA-board (in our case using a USB cable).

5. **Operate and test the circuit (You)**

Some remarks:

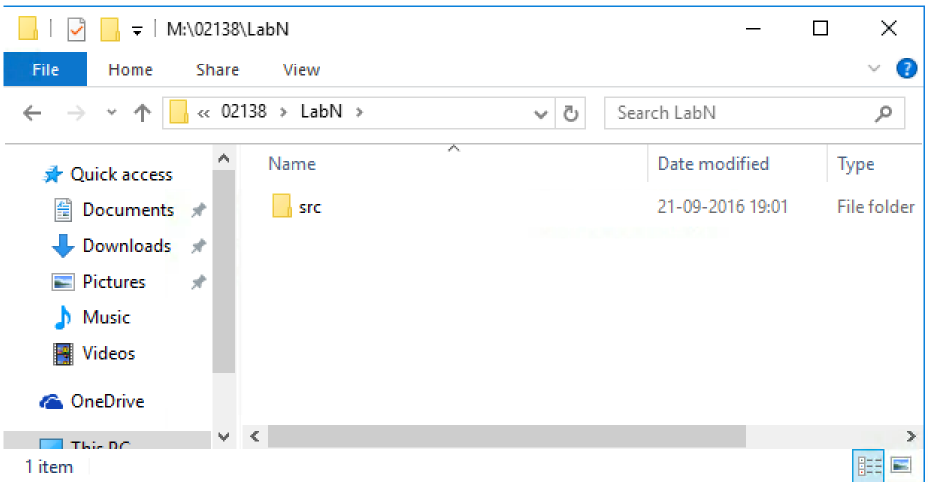
- Steps 1) and 2) are typically the most time consuming and do not require any decisions about what technology to use. Steps 3a) and 3b) also do not require any decisions about what technology to use.
- To perform steps 3c) to 3e) the Vivado tool must know the specific FPGA-chip that the designer wants to use. You indicate this when you start the Vivado tool and create a project. In order to perform step 3e) you furthermore need to specify what IO-pins on this FPGA-chip you want the IO-signals of your design to use. This again depends on the design of the FPGA-board, where specific pins on the chip are connected to specific switches and LED's on the board. A table specifying the pinout is provided in a separate file, called a **X**ilinx **D**esign **C**onstraints file (XDC), which is added to the project alongside your VHDL-file. The details will be explained later.
- As mentioned, the ModelSim and Vivado are complex tools and they need some setting up when you start a design project. Also, they use and manipulate a very large number of files, and you need to work with some discipline and adhere to certain policies. This is also explained in the following.

2. A walkthrough of the design flow

Let us now walk through the design flow for ModelSim and XILINX Vivado and implement the simple 1-bit adder circuit.

2.1 Step 0 – Setting up the directory structure

The design will use different tools to access the same source code. Because of this fact, it is recommended to create a folder for each tool and one for the source code. Remember that no spaces are allowed in the folder names or the path to the folders. Do not put any folders on the *Desktop* and assure that your source files will be saved in one of your personal folders. Assuming that your DTU home directory is mapped to the "M:" drive (N.B. Your DTU home directory might be mapped in another drive, for example "L:"), we recommend that you save your source code files in the directory "M:\02138\LabN\src" (as shown in Figure 1) where you should replace N with the number of the lab you are currently working on. As an alternative you can use an USB-stick.



Name	Date modified	Type
src	21-09-2016 19:01	File folder

Figure 1. Creating the directory used for the source code.

The directory "src" is the only place where you should store your source code (usually VHDL files). When you use ModelSim and XILINX Vivado you should link to the files you store in directory "src" and **not make copies of these files!**

You also need to create folders for the Vivado and ModelSim tools. We would like you to do this on the C: drive. You can use the directory structure suggested in Figure 2 where: s000000 represents the student number, 02138 represents the course number and LabN represents the lab number that you are working on (N is 1 for lab one, 2 for lab two and so on). If you do not want to create new project files each time you come to a new computer you should backup these files.

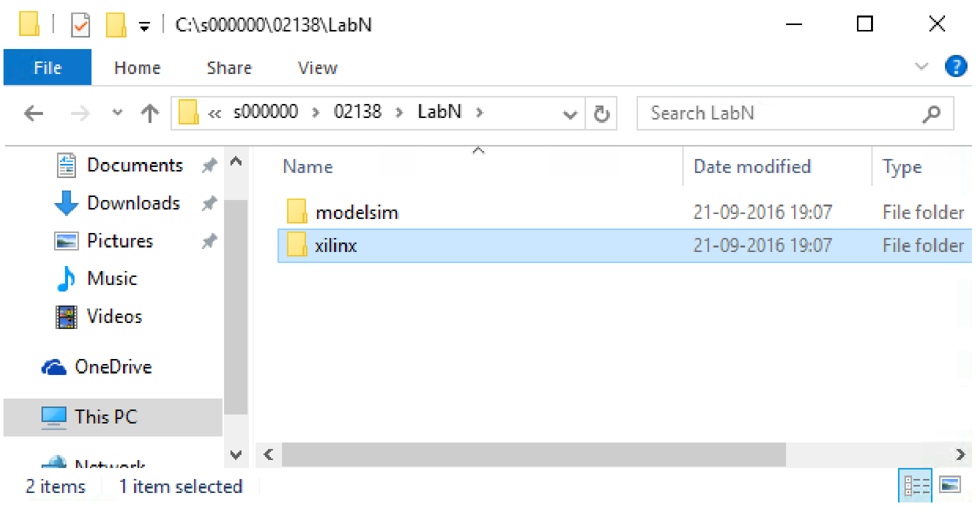


Figure 2. Creating folders for ModelSim and Xilinx Vivado.

The reasons why we split your source code from the files created by the tools are:

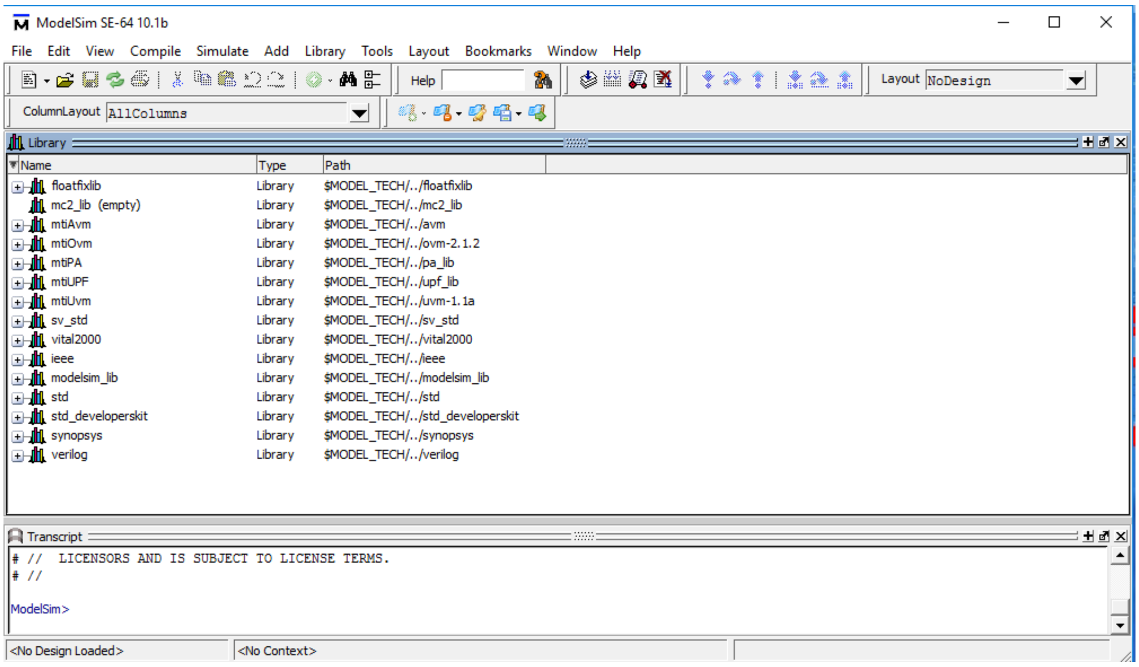
a) To not mix the files you have created with the many files that generated by the tools.

b) To save your source code in a directory that can be accessed from any lab computer, that is private and that is under your control. This is offered by the M: drive. The C: drive can be erased at any time and it is only available on the current machine.

c) To improve performance. The tools used in the lab produce many temporary files and make many hard disk accesses. For this reason you should create the folders on the C: drive (which is local on the machine you are running the tool) rather than the M: drive (which is mounted from a remote server and it is often much slower than local disks).

2.2 Step 1 – Write and compile the VHDL code

1. Start ModelSim. A window similar to the one in Figure 3 will pop up.
2. Create a new project by selecting “File -> New -> Project.” Then the window showed in Figure 4 pops up. For “Project Name” write “full_adder” and for “Project Location” select the directory that you have created in Section 2 (C:\s000000\02138\LabN\modelsim). Then click OK.

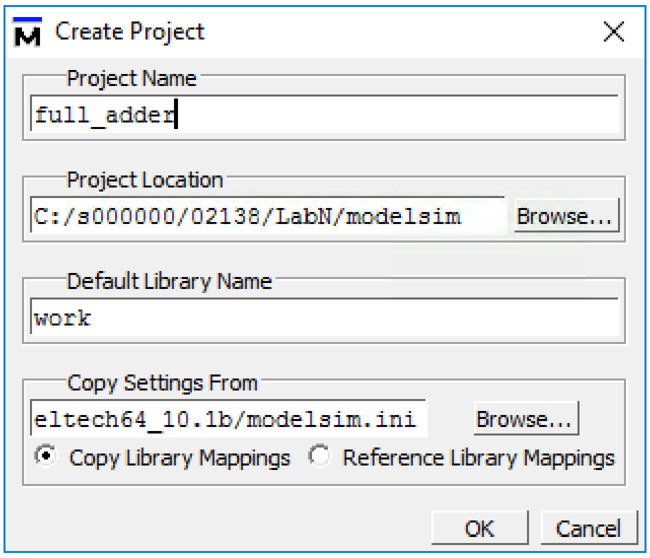


Name	Type	Path
floatfixlib	Library	$MODEL_TECH/../floatfixlib
mc2_lib (empty)	Library	$MODEL_TECH/../mc2_lib
mtiAvm	Library	$MODEL_TECH/../avm
mtiOvm	Library	$MODEL_TECH/../ovm-2.1.2
mtiPA	Library	$MODEL_TECH/../pa_lib
mtiUPF	Library	$MODEL_TECH/../upf_lib
mtiUvm	Library	$MODEL_TECH/../uvm-1.1a
sv_std	Library	$MODEL_TECH/../sv_std
vital2000	Library	$MODEL_TECH/../vital2000
ieee	Library	$MODEL_TECH/../ieee
modelsim_lib	Library	$MODEL_TECH/../modelsim_lib
std	Library	$MODEL_TECH/../std
std_developerskit	Library	$MODEL_TECH/../std_developerskit
synopsys	Library	$MODEL_TECH/../synopsys
verilog	Library	$MODEL_TECH/../verilog

```
# //  LICENSORS AND IS SUBJECT TO LICENSE TERMS.
# //

ModelSim>
```

<No Design Loaded> <No Context>

Figure 3. ModelSim start window.



Project Name: full_adder

Project Location: C:/s000000/02138/LabN/modelsim

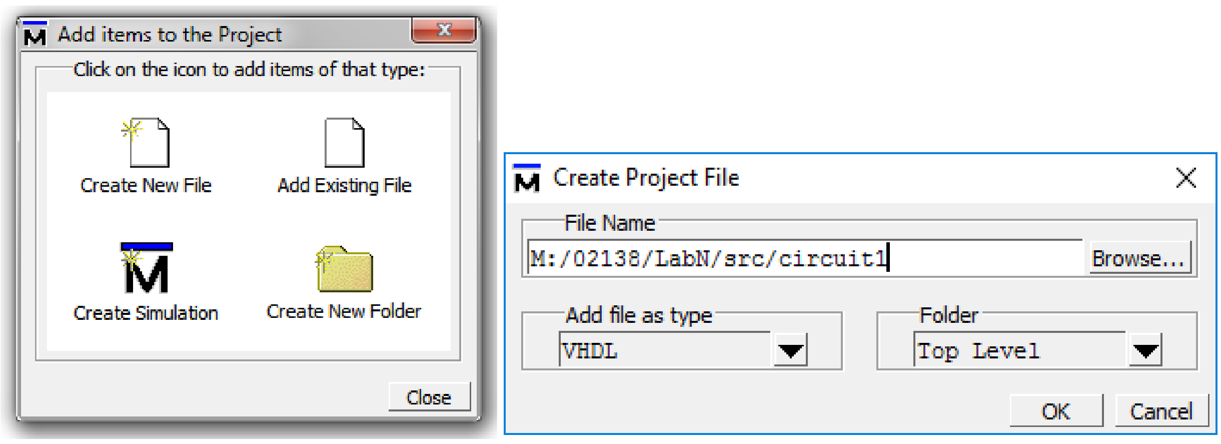
Default Library Name: work

Copy Settings From: eltech64_10.1b/modelsim.ini

[✓ Copy Library Mappings] Reference Library Mappings

Figure 4. Create project.

3. Choose “Create New File”, and name the file circuit1.vhd. See Figure 5. Note: On some versions of ModelSim it is required to type in the “.vhd” extension and other versions will give an error if you do.



File Name: M:/02138/LabN/src/circuit1

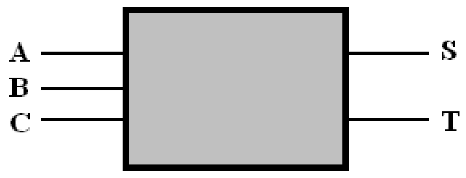
Add file as type: VHDL

Folder: Top Level

Figure 5. Add design files.

4. The file is now added to the project, close the ”Add items to the Project” window by pressing Close.

5. For the walkthrough we are going to implement a full adder. It is described by the box shown in Figure 6.a and the Boolean expressions for the outputs are given in Figure 6.b.



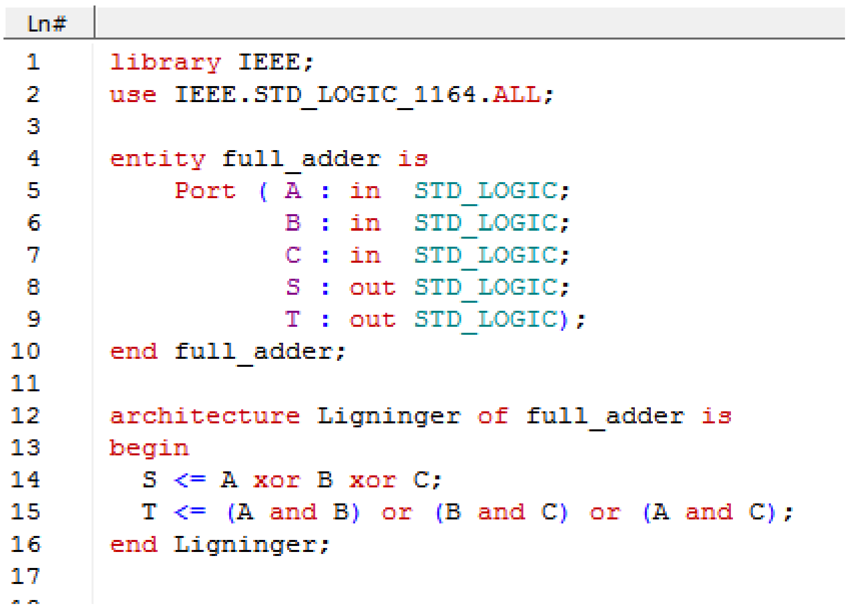
a) Black box view of the full adder

```
S <= A xor B xor C
T <= (A and B) or (A and C)
         or (B and C)
```

b) Boolean equations for the outputs of the full adder

Figure 6. Full adder description

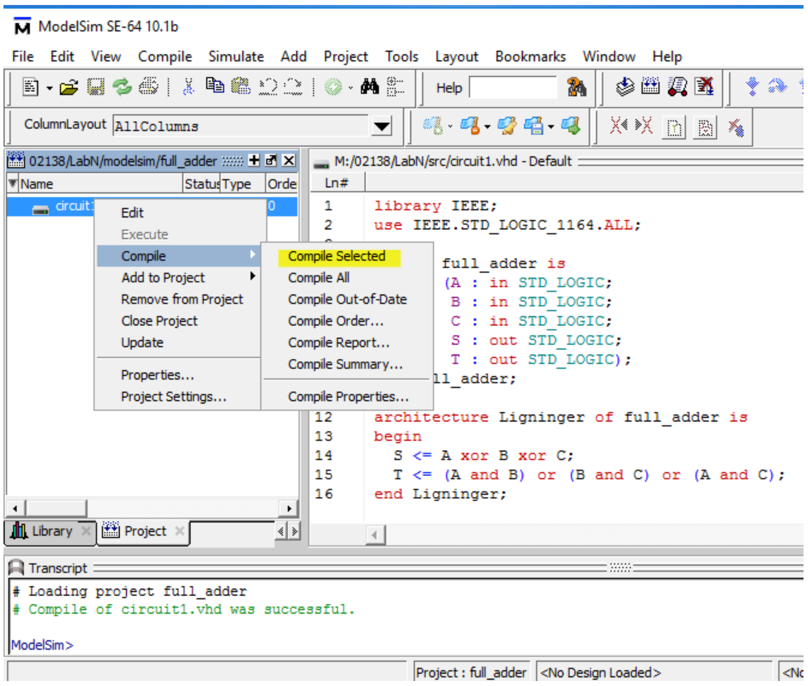
6. You can see the actual VHDL code for this component in Figure 7. Note that the signal type used is **std_logic**.



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity full_adder is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           C : in  STD_LOGIC;
           S : out STD_LOGIC;
           T : out STD_LOGIC);
end full_adder;

architecture Ligninger of full_adder is
begin
  S <= A xor B xor C;
  T <= (A and B) or (B and C) or (A and C);
end Ligninger;
```

Figure 7. VHDL code for the full adder.

7. Compile the circuit1.vhd by right-clicking on it in the project window and then select “Compile -> Compile Selected” as shown in Figure 8. In the transcript window in the bottom it should say that compilation was successful and the status icon should change to a green mark. If not, click on the error message (shown in red in the transcript window) and you will get a more detailed explanation. Fix the bug and compile again.



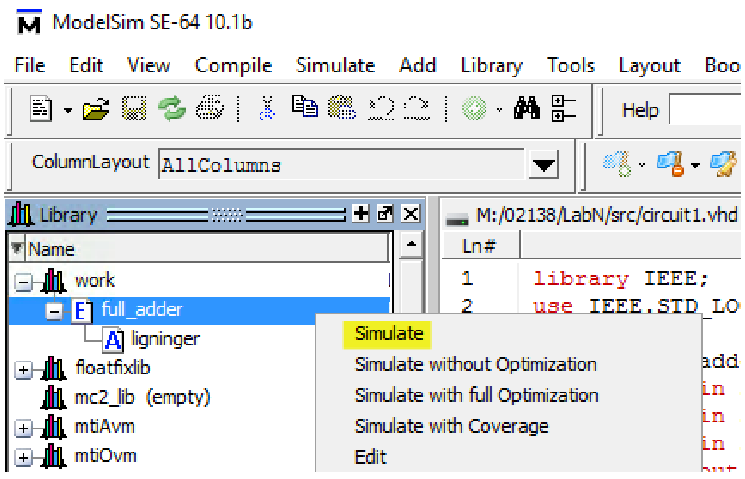
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
# Loading project full_adder
# Compile of circuit1.vhd was successful.
ModelSim>
```

Figure 8. Compile VHDL file with ModelSim.

2.3 Step 2 – Simulate the VHDL code (Note: Behavioral simulation)

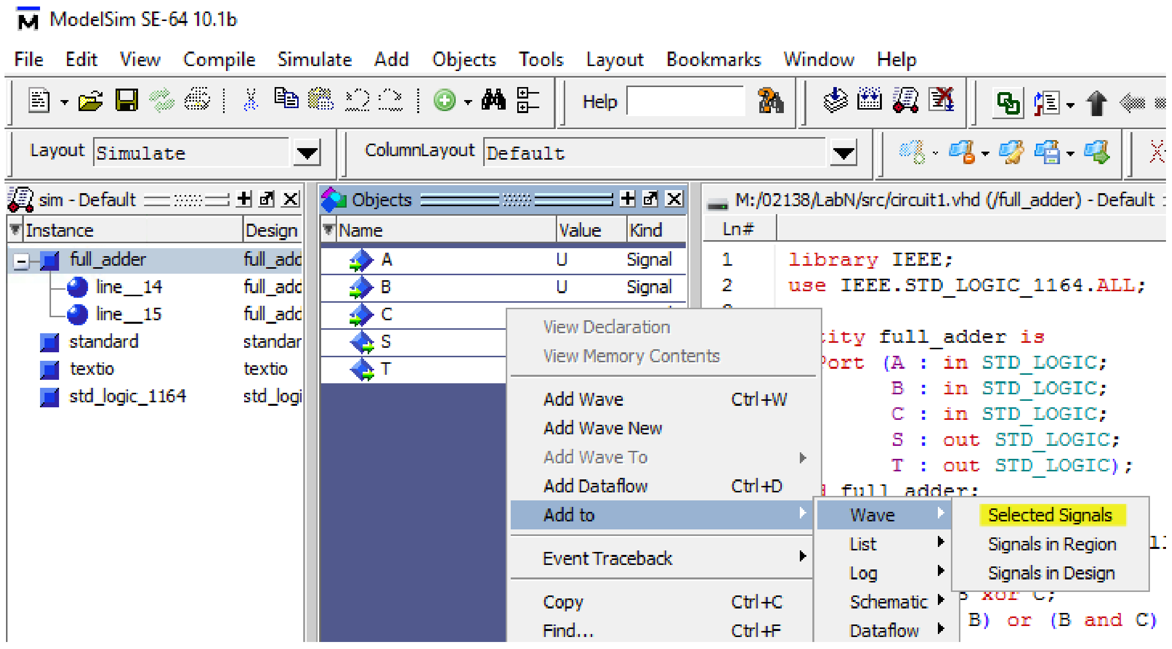
8. After your entity has been compiled, it is added to the "work" library. You can check this by clicking on the "Library" tab in the "workspace" window as shown in Figure 9. You can start the simulation of only one entity at a time by double clicking on it (or by right clicking on it and selecting Simulate from the popup menu). In our case we shall simulate the entity "full_adder".



```
library IEEE;
use IEEE.STD_LO
```

Figure 9. The full adder component is placed in the work library after compilation.

9. When the simulation starts, new windows appear on the screen. In the "sim" window we can see the hierarchy of the component we are trying to simulate. In the object window we can see the signals visible at the current level of hierarchy. Select the signals in the Objects window (by clicking on them while holding the CTRL key pressed, alternatively you can select the first signal, then hold the SHIFT key pressed while clicking on the last signal). Next, right click somewhere inside the "objects" window and select "Add-> To wave -> Selected signals" from the popup menu (as shown in Figure 10). At this point you will see the signals being added to the wave window.



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

ity full_adder is
ort (A : in STD_LOGIC;
     B : in STD_LOGIC;
     C : in STD_LOGIC;
     S : out STD_LOGIC;
     T : out STD_LOGIC);
 full_adder;
                                        1:
                                  3 xor C;
                                  B) or (B and C)
```

Figure 10. Add the relevant signals to the wave window.

10. In order to see some output values produced by our circuit we need to set the values of the inputs first. In our case we have 3 input values (A, B and C) and by writing the following commands in the transcript window we can generate all the input combinations from “000” to “111”.

```
force A 0 0, 1 10 -repeat 20
force B 0 0, 1 20 -repeat 40
force C 0 0, 1 40 -repeat 80
```

The first line means ”force A to assume the value ‘0’ at moment 0. Then force A to be ‘1’ at moment 10 and repeat the operations with a period of 20”. The times are by default given in ns (10^{-9} seconds). It is possible to skip the option “-repeat”, which means that a will get the value 1 at time instance 10 and keep it forever. When the assignment is periodic, the times are measured relative to the start of each period. The values for A, B and C as a function of time are shown in Figure 11.

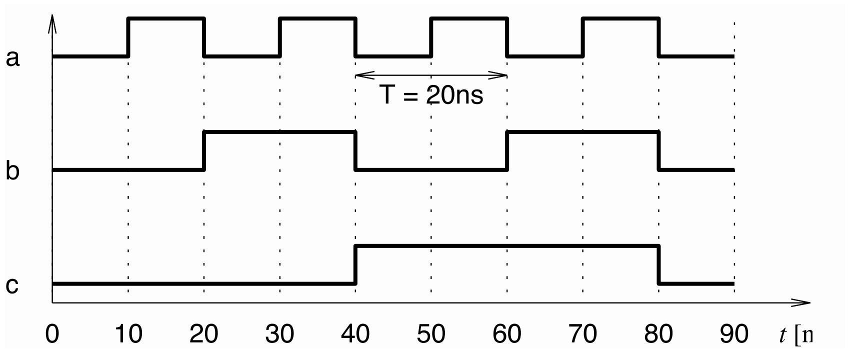


Figure 11. The waveform produced by forcing signals A, B and C.

11. Run the simulation. This can be done in several ways:
    a. Click on the icon from the toolbar. The simulation will run for the amount specified in the box to the left of this icon. In case the box looks like this 100 ns then the simulation will run for 100 ns.
    b. Write on the transcript window “run 100”. The value 100 specifies that the simulation should run for 100 ns.
    c. From the menu on the top of the ModelSim window select: “Simulate -> Run ->Run 100”. Also in this case the simulation will run for 100 ns.

12. Check that the result is correct by inspecting the wave diagram produced by the simulator. To see the full simulation in the same Window, right-click on the wave and select “Zoom Full”. For easy zooming you can also use the following icons from the toolbar that stand for: zoom in, zoom out and zoom full.

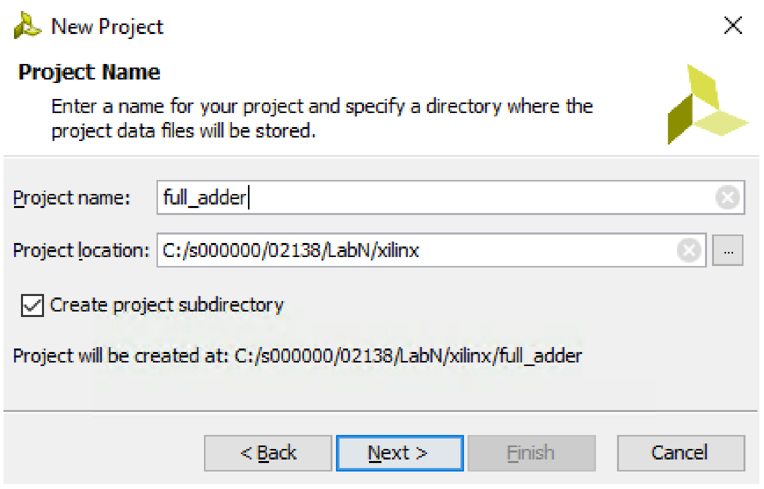
2.4 Step 3 – Synthesize and implement the VHDL code

Now that you have a syntactically and functionally correct VHDL-description of your design the next step is to synthesize it and eventually implement it on the FPGA board.

As was the case with ModelSim you first need to set up the Vivado tool by creating a project and specifying which FPGA-chip you will be targeting, and adding your VHDL source file to the project.

Step 3a – Create a project in Vivado

13. Start the XILINX Vivado tool. This is done by clicking on the Vivado icon from the desktop.
14. Create a new project, by selecting "File -> New Project." Give it a name (i.e., "full_adder") and place the project in the XILINX directory that you have created in Section 2 (C:\s000000\02138\LabN\xilinx). Make sure there are no spaces in the path shown in "Project loca-tion". Make also sure that the box "Create project subdirectory" is ticked. Press Next.



New Project

Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: full_adder

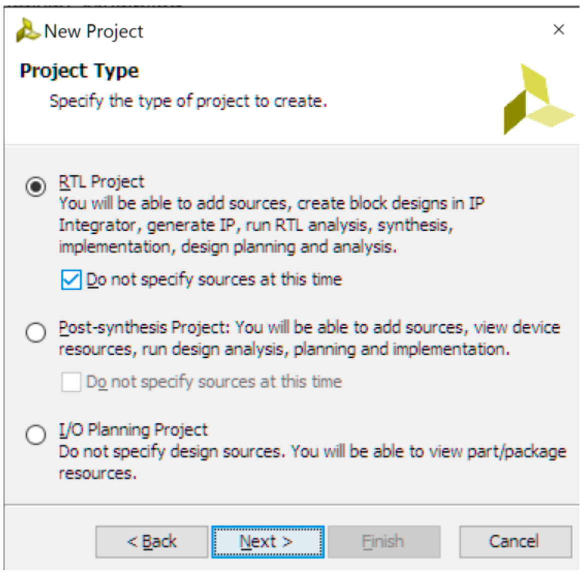
Project location: C:/s000000/02138/LabN/xilinx

[✓] Create project subdirectory

Project will be created at: C:/s000000/02138/LabN/xilinx/full_adder

Figure 12. Vivado project wizard.

15. In the window shown in Figure 13 we specify the type of project we want to create. In our case it's an RTL project. RTL stand for "**R**egister **T**ransfer **L**evel" and the hardware is described using VHDL files. If you don't have any existing source code then tick also the box called "Do not specify sources at this time".


New Project

Project Type

Specify the type of project to create.

[✓ RTL Project]
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.

[✓ Do not specify sources at this time]

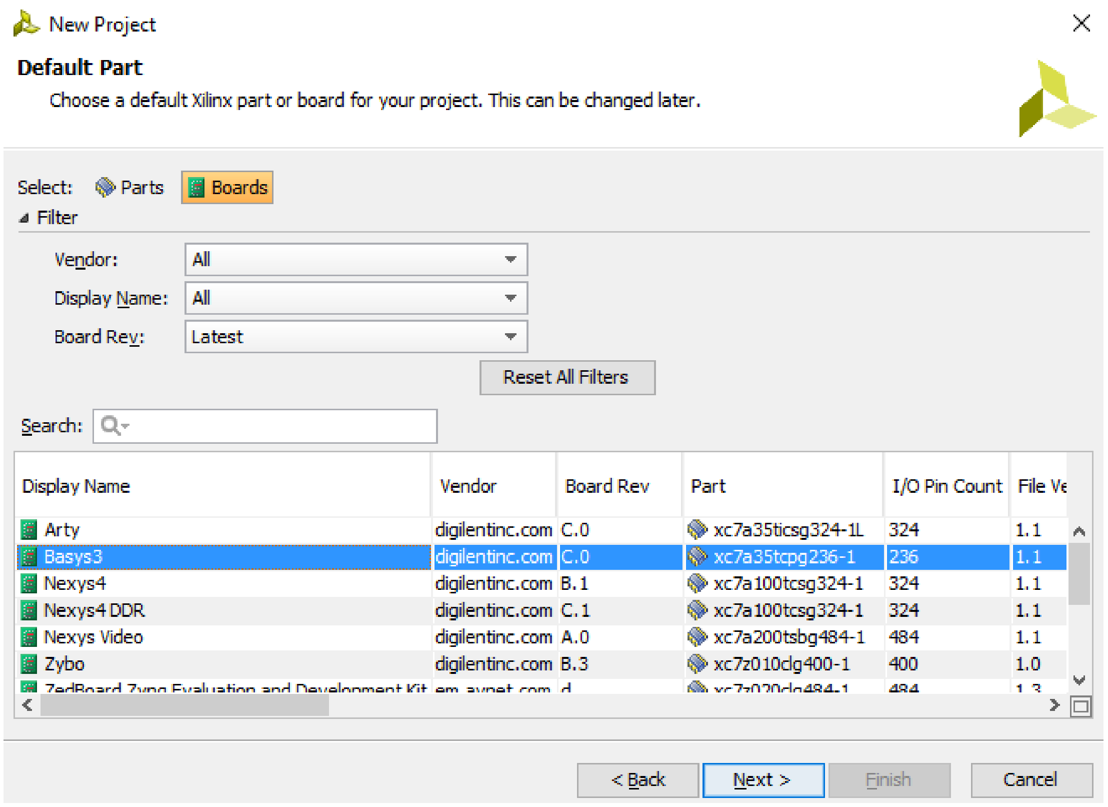
Post-synthesis Project: You will be able to add sources, view device resources, run design analysis, planning and implementation.

Do not specify sources at this time

I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

Figure 13 Project Type

16. In the window shown in Figure 14 you have to specify the parameters of the FPGA device available on the BASYS 3 board that we use for the lab. First select the boards tab and then select Basys3 as shown in Figure 14


New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Select: Parts [✓ Boards]

Filter

Vendor: All

Display Name: All

Board Rev: Latest

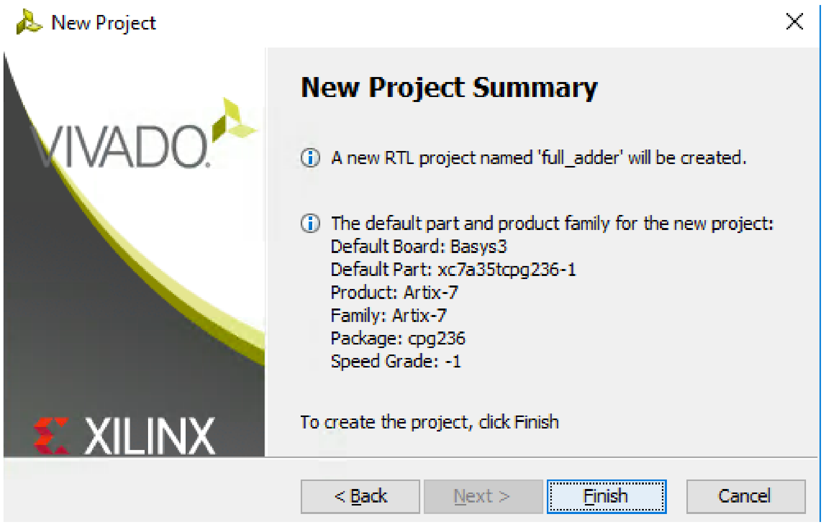
Search:

Display Name	Vendor	Board Rev	Part	I/O Pin Count	File Ve
Arty	digilentinc.com	C.0	xc7a35ticsg324-1L	324	1.1
Basys3	digilentinc.com	C.0	xc7a35tcpg236-1	236	1.1
Nexys4	digilentinc.com	B.1	xc7a100tcsg324-1	324	1.1
Nexys4 DDR	digilentinc.com	C.1	xc7a100tcsg324-1	324	1.1
Nexys Video	digilentinc.com	A.0	xc7a200tsbg484-1	484	1.1
Zybo	digilentinc.com	B.3	xc7z010clg400-1	400	1.0
ZedBoard Zynq Evaluation and Development Kit	em.avnet.com	d	xc7z020clg484-1	484	1.3

Figure 14. FPGA device parameters.

***NOTE**: If you try to run Vivado on your own computer and you don't find the Basys3 board listed in the boards section then you need to install the board files from Digilent by following the instructions from this link:* https://reference.digilentinc.com/reference/software/vivado/board-files

17. You should now see a summary of you design, like on Figure 15. Press "Finish".


New Project

New Project Summary

A new RTL project named 'full_adder' will be created.

The default part and product family for the new project:
Default Board: Basys3
Default Part: xc7a35tcpg236-1
Product: Artix-7
Family: Artix-7
Package: cpg236
Speed Grade: -1

To create the project, click Finish

Figure 15. Project summary.

18. You should now add the VHDL file that you have just created using ModelSim and is located on "src" directory from the M drive (for example: M:\02138\LabN\src). This is done by right-clicking anywhere in the "Sources window" and then selecting "Add Sources…" as shown in Figure 16.

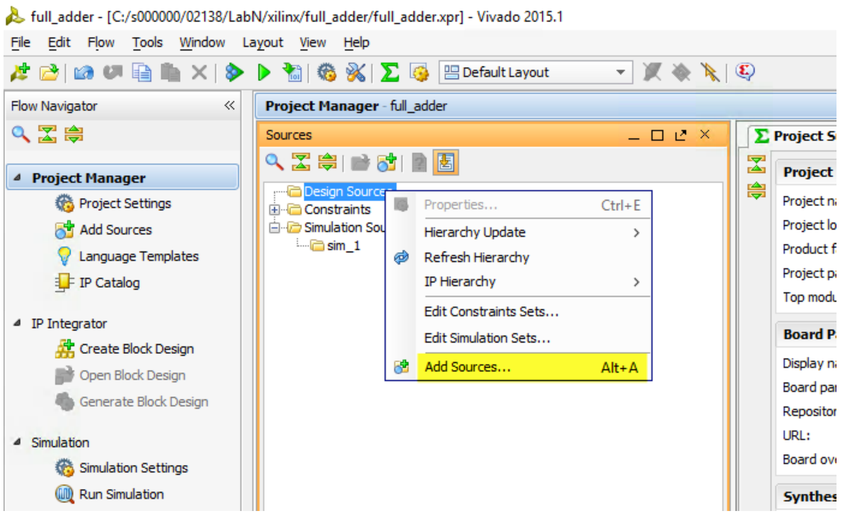
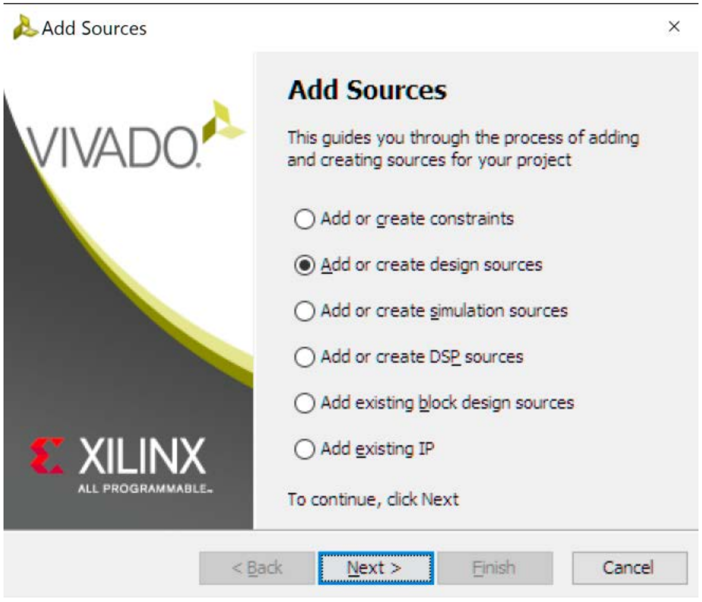


Figure 16. Source files.

19. Next, the wizard from Figure 17 pops up. Select “Add or create design sources” and press Next.


Add Sources

Add Sources

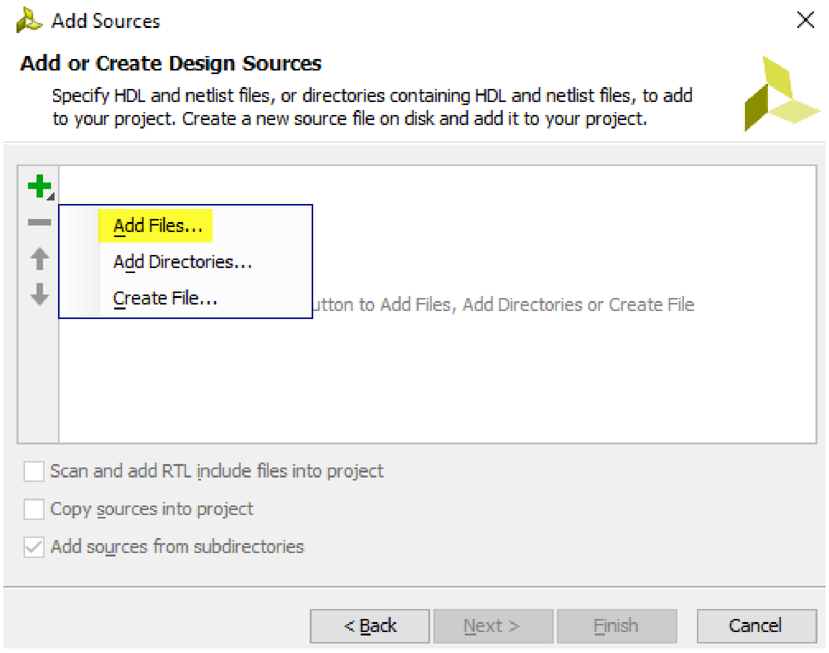
This guides you through the process of adding and creating sources for your project

( ) Add or create constraints
[✓ Add or create design sources]
( ) Add or create simulation sources
( ) Add or create DSP sources
( ) Add existing block design sources
( ) Add existing IP

To continue, click Next

Figure 17 Add Sources Window

20. We want to reference the original source file that we’ve saved in directory “M:\02138\LabN\src” so, make sure that the box “Copy sources into project “ is **NOT** selected. Next, click on the green plus symbol and select “Add file…” as shown in Figure 18. Then go to the directory “M:\02138\LabN\src” and select the “circuit1.vhd” file. Next click OK. In the “Add or Create Design Sources file” click Finish. You have now completed the setup of a Vivado project.


Add Sources

Add or Create Design Sources

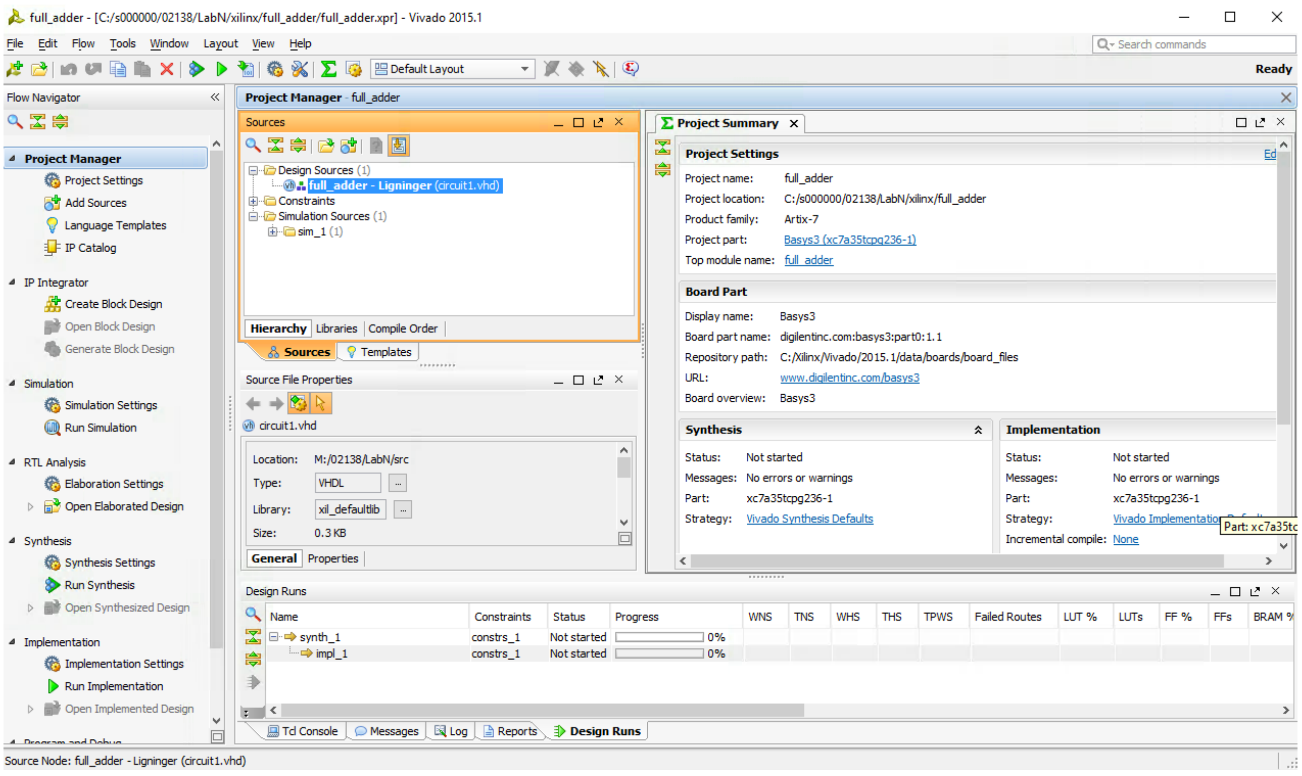
Specify HDL and netlist files, or directories containing HDL and netlist files, to add to your project. Create a new source file on disk and add it to your project.

[ ] Scan and add RTL include files into project
[ ] Copy sources into project
[✓] Add sources from subdirectories

Figure 18. Adding a source file

Step 3b – Perform the actual synthesis and implementation

21. The Vivado window should now look like Figure 19.



Location: M:/02138/LabN/src
Type: VHDL
Library: xil_defaultlib
Size: 0.3 KB

Project Settings

Project name: full_adder
Project location: C:/s000000/02138/LabN/xilinx/full_adder
Product family: Artix-7
Project part: Basys3 (xc7a35tcpg236-1)
Top module name: full_adder

Board Part

Display name: Basys3
Board part name: digilentinc.com:basys3:part0:1.1
Repository path: C:/Xilinx/Vivado/2015.1/data/boards/board_files
URL: www.digilentinc.com/basys3
Board overview: Basys3

Synthesis

Status: Not started
Messages: No errors or warnings
Part: xc7a35tcpg236-1
Strategy: Vivado Synthesis Defaults

Implementation

Status: Not started
Messages: No errors or warnings
Part: xc7a35tcpg236-1
Strategy: Vivado Implementation Defaults
Incremental compile: None

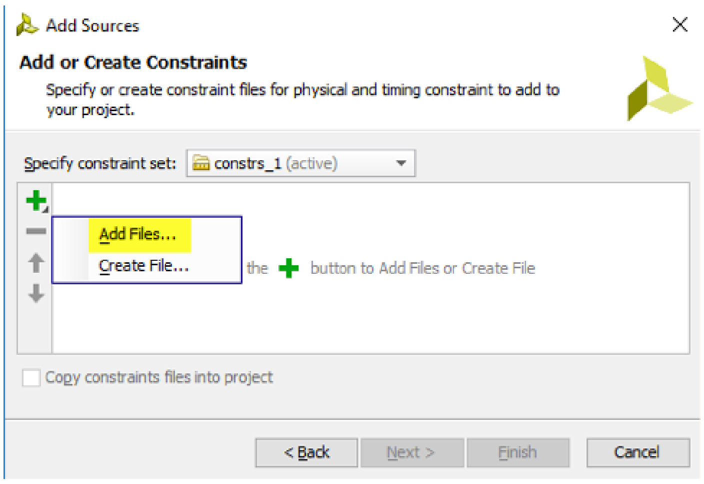
Design Runs

Name	Constraints	Status	Progress	WNS	TNS	WHS	THS	TPWS	Failed Routes	LUT %	LUTs	FF %	FFs	BRAM %
synth_1	constrs_1	Not started	0%											
impl_1	constrs_1	Not started	0%											

Source Node: full_adder - Ligninger (circuit1.vhd)

Figure 19. Vivado window.

22. It is now possible to synthesize the design and implement it on the board, but to be able to control the user inputs of the FPGA board it is necessary to map the input and output signals of the "full_adder" to the correct FPGA pins. This is done with the help of a **X**ilinx **D**esign **C**onstraints file (with the extension xdc). Download the XDC-file called Basys3.xdc from CampusNet to the src directory that you've created on the M: drive and rename it to Circuit1.xdc. Next, you'll need to add this XDC file to your project. On the top menu bar click on "File -> Add Sources…" and select "Add or create constraints". Then click Next.
23. In the "Add Sources" window click on the green plus icon and select "Add Files" as shown in Figure 20. Then select the Basys3.xdc file from the place where you've saved it (M:\02138\LabN\src). Next make sure that the tick is **NOT** set for the box "Copy constraints files into project". Next click on Finish.


Add Sources

Add or Create Constraints

Specify or create constraint files for physical and timing constraint to add to your project.

Specify constraint set: constrs_1 (active)

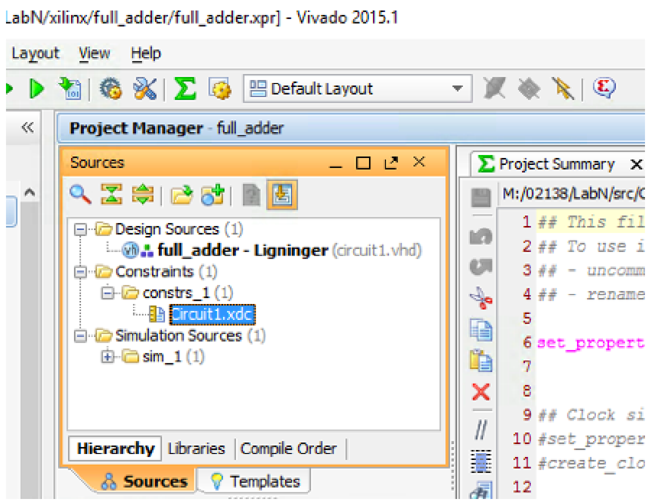
Add Files...
Create File...

the button to Add Files or Create File

Copy constraints files into project

Figure 20. Adding the constraints file

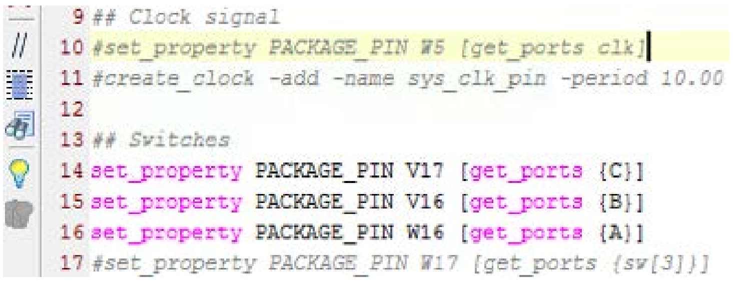
24. Go to “Project Manager -> Constraints -> constr_1” as shown in Figure 21 and double click on Circuit2.xdc to edit it.


LabN/xilinx/full_adder/full_adder.xpr] - Vivado 2015.1

```
## This fil
## To use i
## - uncomm
## - rename

set_propert


## Clock si
#set_proper
#create_clo

```

Figure 21. Opening the editor for the XDC file

The constraint file binds pins (physical pins from the FPGA chip) with ports (signals on the top level component from the FPGA design). The general format of a line in the XDC file used for binding ports to pins is: set_property PACKAGE_PIN V17 [get_ports {sw[0]}] #sw[0]
The symbol # is used in XDC files for commenting. In order to enable pins on the FPGA we need to uncomment some relevant lines from the XDC file (by removing some # symbols). In our case we want to connect the inputs “A”, “B” and “C” from the entity full adder to switches 2, 1 and 0 from the Basys3 board. First we need to identify in the XDC file where are the constraints related to the Switches. Then we need to uncomment the 3 lines where sw[0] to sw[2] are defined and then replace the generic names (for example sw[2]) with the actual signal names (for example “A”) that is used in our design. See Figure 22.



```
## Clock signal
#set_property PACKAGE_PIN W5 [get_ports clk]
#create_clock -add -name sys_clk_pin -period 10.00

## Switches
set_property PACKAGE_PIN V17 [get_ports {C}]
set_property PACKAGE_PIN V16 [get_ports {B}]
set_property PACKAGE_PIN W16 [get_ports {A}]
#set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
```

Figure 22. Connecting signals "A" and “B” and “C” to sw[2],sw[1] and sw[0]

25. Do the same for the output signals “S” and “T”. You should end up with the following lines uncommented in your XDC file (all the other lines in the file are commented out):

```
set_property IOSTANDARD LVCMOS33 [get_ports *]
set_property PACKAGE_PIN V17 [get_ports {C}]
set_property PACKAGE_PIN V16 [get_ports {B}]
set_property PACKAGE_PIN W16 [get_ports {A}]
set_property PACKAGE_PIN U16 [get_ports {S}]
set_property PACKAGE_PIN E19 [get_ports {T}]
```

Note: *The first line (which is already uncommented) specifies the IO standard that is used by all the IO ports. In the case of the Basys3 board this is LVCMOS33.*
Make sure you save the XDC file before continuing to the next step.

26. Next we need to generate the bit file that we can upload to the FPGA. You can do this by going to the Flow Navigator (on the left side of the screen) and selecting “Program and Debug -> Generate bitstream” as shown in Figure 23. This will perform many steps. First the design will be synthesized, then it will be implemented and in the end a bit stream will be generated.

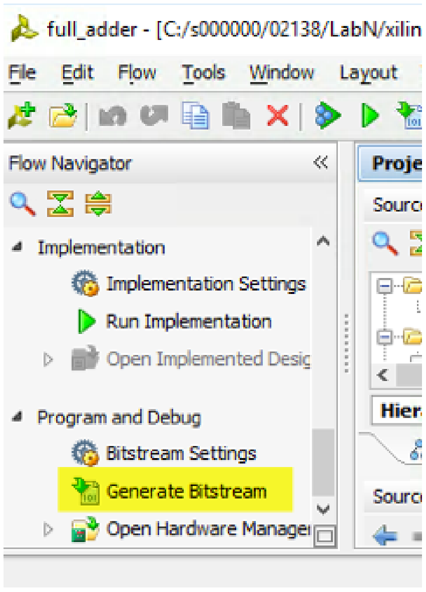
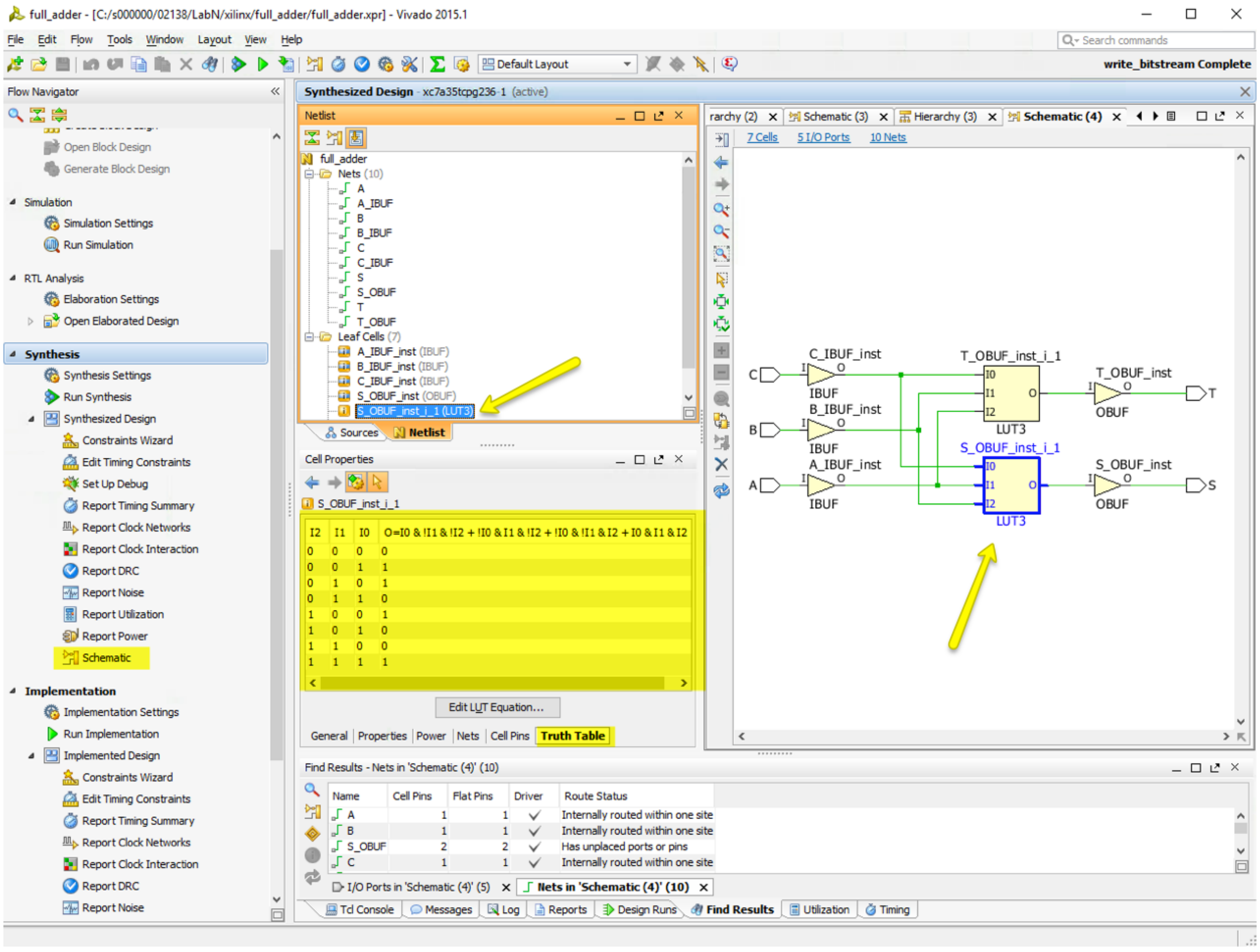


Figure 23. Generating the bit stream

27. If you want to get more details about the synthesized design you can go to the “Flow Navigator” window (on the left side of the Vivado tool) and select “Synthesis -> Synthesized design -> Schematic”. See Figure 24.



I2	I1	I0	O=I0 & !I1 & !I2 + !I0 & I1 & !I2 + !I0 & !I1 & I2 + I0 & I1 & I2
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Name	Cell Pins	Flat Pins	Driver	Route Status
A	1	1	✓	Internally routed within one site
B	1	1	✓	Internally routed within one site
S_OBUF	2	2	✓	Has unplaced ports or pins
C	1	1	✓	Internally routed within one site

Figure 24. Schematic of the synthesized design

In Figure 24 you can see that it’s possible to get information like how the schematic of your design is, what is the truth table stored in the lookup tables used to implement your design, how the components are wired together, etc.

2.5 Step 4 – Configure the FPGA board

28. If you haven't done this yet then plug in the Basys3 board into a USB-port (if it's not already plugged in) and then turn it on.

29. Next we need to upload to the FPGA the bit file that we've generated. From the Flow Navigator click on "Program and Debug -> Hardware Manager", as shown in Figure 25.

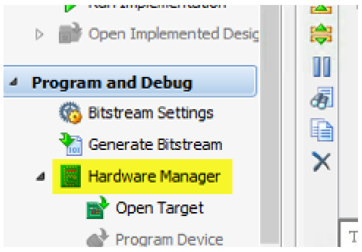


Figure 25. Starting the Hardware Manager

30. A new window called hardware manager is opened. Click on the link called "Open target" from the top of this window as shown in Figure 26 and select "Auto Connect".

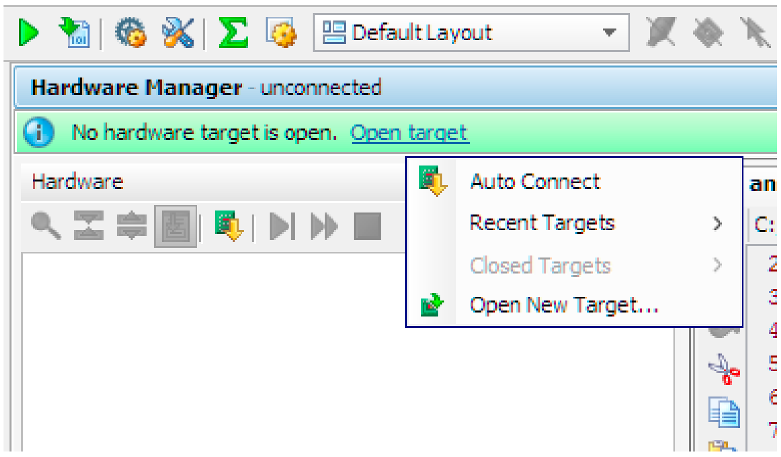
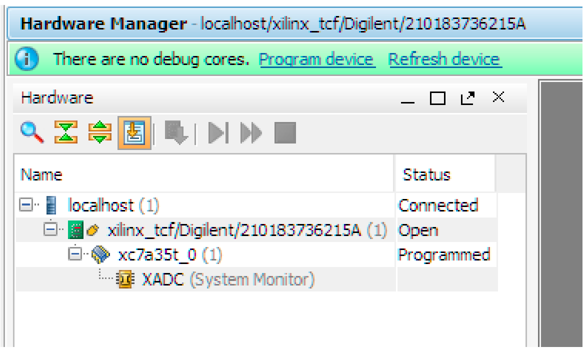


Figure 26 Connecting to the FPGA

31. If the FPGA was correctly connected then the FPGA should be listed as shown in Figure 27.

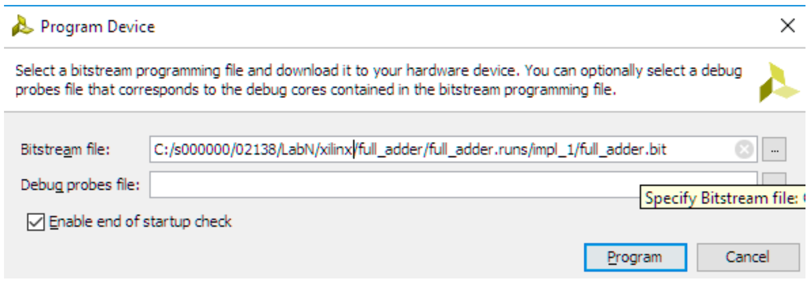

Hardware Manager - localhost/xilinx_tcf/Digilent/210183736215A
There are no debug cores. Program device Refresh device
Hardware

Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/210183736215A (1)	Open
xc7a35t_0 (1)	Programmed
XADC (System Monitor)	

Figure 27. FPGA is connected

32. Next click on the "Program Device" link from the top of the "Hardware Manager" window (see Figure 27) and select the FPGA chip (in our case it should be xc7a35t_1). Next you'll get a new win-

dow called “Program device” as shown in Figure 28. Click on Program. If the FPGA was successfully program you will see the DONE led from the FPGA lit up. This is a green LED in the rightmost upper corner of the Basys3 board.



Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.

Bitstream file: C:/s000000/02138/LabN/xilinx/full_adder/full_adder.runs/impl_1/full_adder.bit

Debug probes file:

Specify Bitstream file:

☑ Enable end of startup check

Figure 28. Program device window

Now the FPGA is configured with your design. Well DONE! ☺

3. Where to get the tools

The tools can be downloaded to and the most of the exercise can be run from your PC. This section briefly discusses where and how this may be done.

The tools used are free versions of commercial software and will require license files to function. You will have to sign up for downloading the software on the product webpages by making your personal account as well as request license files. We suggest using your DTU student email account for this purpose.

3.1 XILINX Vivado WebPACK

The Vivado WebPACK installation tar archive is approx. 6GB large and will take some time to download. The installation will unpack this into approx. 16GB.

The "XILINX Vivado WebPACK (free)" can be downloaded from the XILINX webpage:

https://www.xilinx.com/products/design-tools/vivado/vivado-webpack.html

You download the latest version of Vivado (2017.x). The PCs in the lab may have version 2015.x installed. The differences are minor and most probably you won't notice. It's best if you install the windows version. At the exercises, we do not support Linux installations. The installation process will invoke you to make an account on the XILINX webpage to allow the download. Furthermore, the installation process may invoke downloading other software packages (WinPcap and Jungo) depending on your PC's current configuration. When the file is downloaded unpack the tar file and start the installation. Follow the instructions on the screen. You should choose to install the Webpack version (which comes with a free license). At the end of the installation the Xilinx License Configuration Manager (XCLM) will start automatically and guide you through the licensing process.

If you need more information about installing Vivado you can also see this video:
http://www.xilinx.com/video/hardware/vivado-design-suite-installation-overview.html

Finally, in order to use your Basys 3 board with Vivado you need to install a so-called board-file provided by Digilent. Go to https://reference.digilentinc.com/learn/software/tutorials/vivado-board-files/start and follow the instructions (download the file and save it in the proper directory)

3.2 Mentor Graphics ModelSim

The "ModelSim PE Student Edition" can be downloaded by following the instructions in:

http://www.mentor.com/company/higher_ed/modelsim-student-edition

The installation is quite small compared to the Vivado tool, approx. 550 MB. To download the file you will have to make an account on the Mentor Graphics webpage. Select "Download" from the list of instructions to initiate the installation and follow the instructions.

At the end of the installation your internet browser will be directed to a license request form. Complete the form with all fields and select Request Student Edition to submit. The reply email will contain the license file as well as instruction on where to save the license file.