

# A digital circuit design flow guide using VHDL and Xilinx Vivado targeting a Digilent BASYS 3 FPGA board

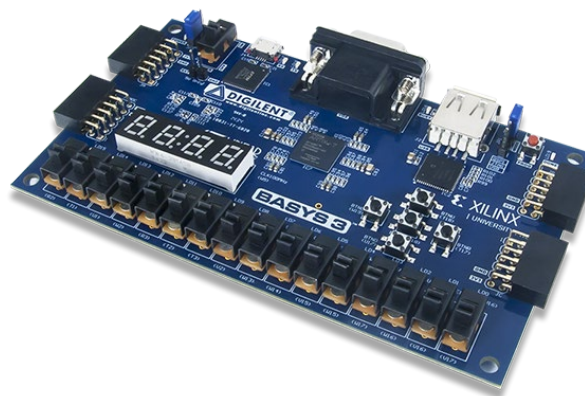
Flemming Stassen

## Preface

This guide is written to help the beginner implement digital circuits described in VHDL in FPGA technology using the Xilinx Vivado EDA tool for use in the DTU 02138 Digital Electronics class. Xilinx Vivado provides a modern full-fledged development environment that covers a vast area of tasks, from circuit simulation, to timing analysis and synthesis. The tool is used in industry and its operations can be extremely complicated as it contains advanced features that we will not dive into.

In this guide, the download instructions describe how to download Vivado ML Standard Edition 2022.1. In the data bars, older versions are installed, e.g. the Xilinx Vivado HL WebPACK 2018.2 (free). For our use, version differences are minor and most likely you won't notice the difference.

The guide first describes the different steps of the design-flow in order to provide an understanding of *what* the tool actually does. Following this, the guide provides you with a detailed description of *how* to use the tool and successfully implement your digital circuit in FPGA technology. The document is supported by screenshots and carefully walks you through the entire design process step-by-step. Finally, for those who want to install the Xilinx Vivado tool on their own PCs, the guide provides a description of how to download and install the tool.



This digital circuit design flow guide Version 13.1 is an update of previous version 12.1 by Edward Todirica, Jens Sparsø, Flemming Stassen and Eleftherios Kyriakakis. Several screen images are from version 2018.2.

## Contents

<b>Contents .....</b>	<b>2</b>
<b>1 The design flow.....</b>	<b>3</b>
1.1 Design flow perspective.....	3
1.2 Overview of the design flow .....	4
<b>2 A walkthrough of the design flow .....</b>	<b>6</b>
2.1 Step 1 - Project Management .....	6
2.1.1 Create a project in Vivado.....	6
2.1.2 Define project source files .....	8
2.1.3 Define design constraints.....	12
2.2 Step 2 - Simulation .....	15
2.2.1 Interactive simulation .....	16
2.2.2 Test bench driven simulation.....	17
2.3 [OPTIONAL] RTL analysis.....	19
2.4 Step 3 - Synthesis & Implementation .....	20
2.5 Step 4 - Program & Debug .....	22
<b>3 Installing the Xilinx Vivado ML Standard tool on your PC.....</b>	<b>24</b>
3.1 Register a personal account with Xilinx .....	24
3.2 Download the Xilinx Vivado ML Standard edition installer .....	24
3.3 Install the Xilinx Vivado ML Standard tool .....	24
3.4 Download and install the Basys3 board file .....	24
3.5 Open Vivado and create a new project.....	24

## 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 its functionality is formally verified in an iterative process using a RTL simulator. Once the functionality is validated, the circuit can be synthesized into an implementation for an FPGA using a synthesis tool such as the Xilinx Vivado. The result is a bit-file that can be downloaded to the FPGA. More specifically, we will use:

1. The VHDL hardware description language for describing the circuit logic.
2. The Xilinx Vivado for simulating and synthesizing a circuit described in VHDL into an FPGA implementation.
3. 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 this course (02138) and subsequent courses we will be using VHDL together with Vivado to carry out the lab assignments and implement our designs. Vivado is widely used in the electronics industry, and it can be a complex tool to use. In fact, it is more useful to think of it as a toolbox, from which the designer picks and uses 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 use of Vivado can be overwhelmingly complicated – the manuals and user guides are thousands of pages. In this guide we 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 starts with a VHDL description of your circuit and ends when you have a functional implementation running on an FPGA board in the laboratory. This involves the following steps:

### 1. DESIGN

Write and compile the VHDL code describing your design. You may do this using your favorite text editor or use the editor that comes with Vivado. If you choose to use the in-built editor, it provides you with an automatic syntax error check.

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

### 2. SIMULATION

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. Re-iterate over steps 1-2 until the errors are fixed and the desired behavior is achieved.

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

### 3. SYNTHESIS & IMPLEMENTATION

This is a complex task involving transformations and optimizations. Luckily the tool does all this.

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

- a) Identifying flip-flops and combinational circuitry.
- b) Synthesizing the combinational circuitry by recognizing adders, multipliers, and other typical combinational circuit building blocks, and by minimizing the remaining combinational circuitry.

*The result is a schematic using flip-flops and generic technology independent combinational 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 combinational 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, the tool generates a programming file, called bitstream, that describes how the FPGA-chip should be configured in order to implement the design.*

#### 4. PROGRAM & DEBUG

This is done by uploading the generated bit stream file to the FPGA-board (in our case using a USB cable) and testing the circuit in the FPGA yourself.

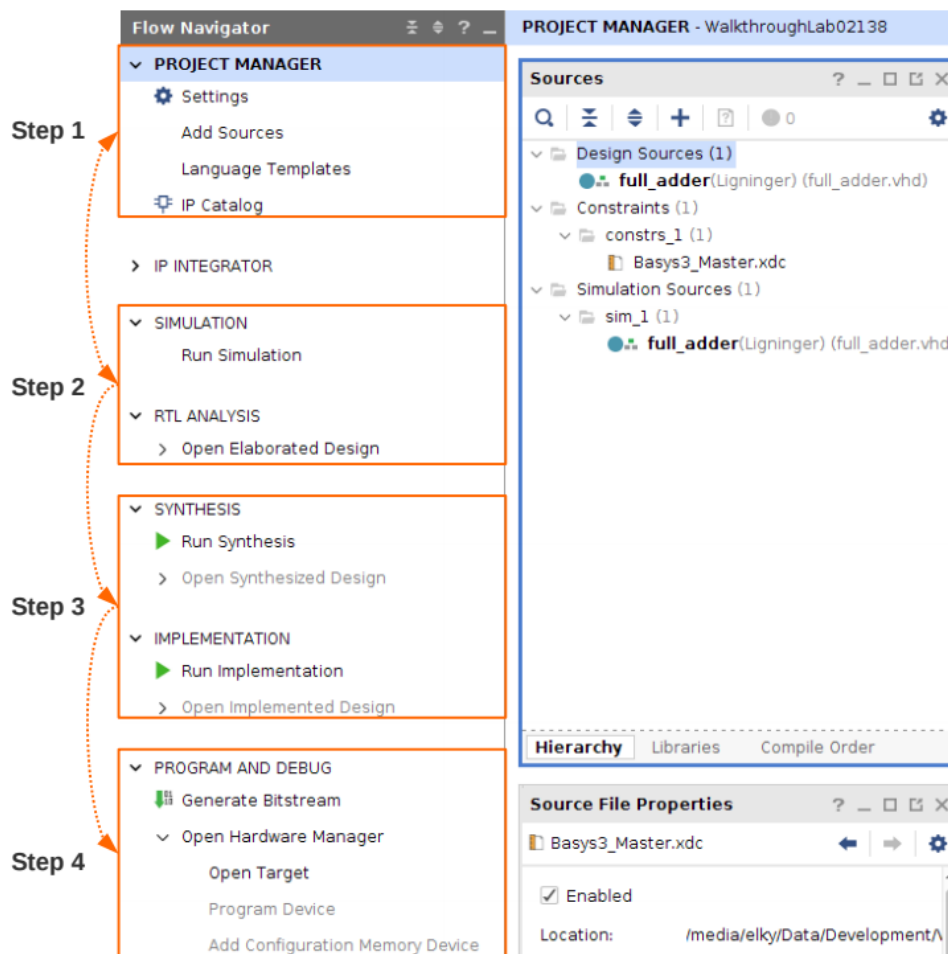


Figure 1 Design flow overview

*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 3f) Vivado 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 must also specify what IO-pins of 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 components of the board (i.e. LED). A table specifying the pinout is provided in a separate file, called a Xilinx Design Constraints file (XDC), which is added to the project alongside your VHDL-file. The details will be explained later.

## 2 A walkthrough of the design flow

Let us now walk through the design flow for Xilinx Vivado with an example circuit that implements a simple 1-bit full adder.

### 2.1 Step 1 - Project Management

#### 2.1.1 Create a project in Vivado

1. Start the Xilinx Vivado tool by clicking on the Vivado icon  from the desktop. Create a new project, by selecting “File -> New -> Project...” and name it (i.e., “LabN”). It is advised to place the project in a parent folder named as the course (i.e. “/home/02138”). Make sure there are no spaces in the path shown in “Project location”. Make also sure that the checkbox “Create project subdirectory” is ticked as shown in Figure 2. Press “Next >”.

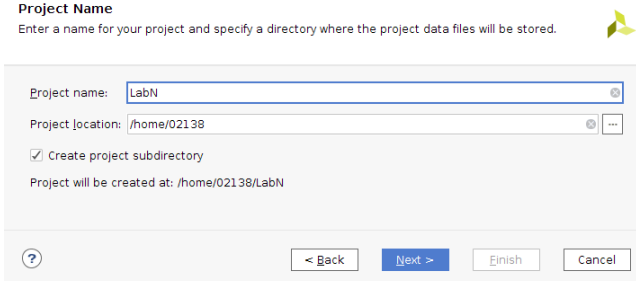


Figure 2 Vivado project wizard.

2. In the window shown in Figure 3, we specify the type of project we want to create. In our case it's an RTL project. RTL stand for “Register Transfer Level” and the hardware is described using VHDL files. Since this is the first project we tick the checkbox “Do not specify sources at this time”.

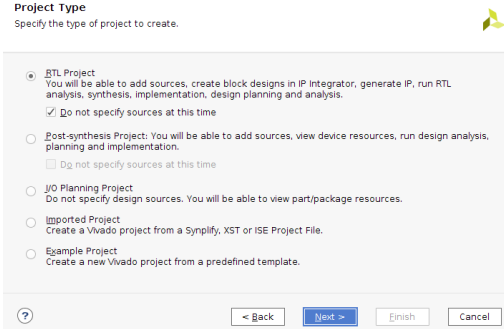


Figure 3 Choosing a project type

3. In the window shown in Figure 4 you have to specify the FPGA device that we use for the lab. First select the boards tab and then select Basys 3. Press “Next >” and you should now see a summary of your project as shown in Figure 5. Press “Finish”.

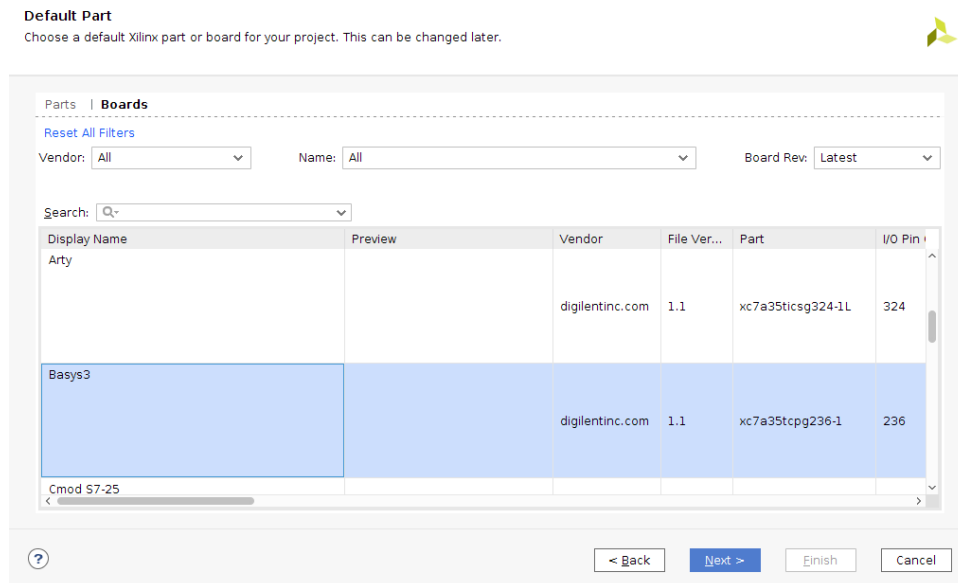


Figure 4 FPGA device parameters.

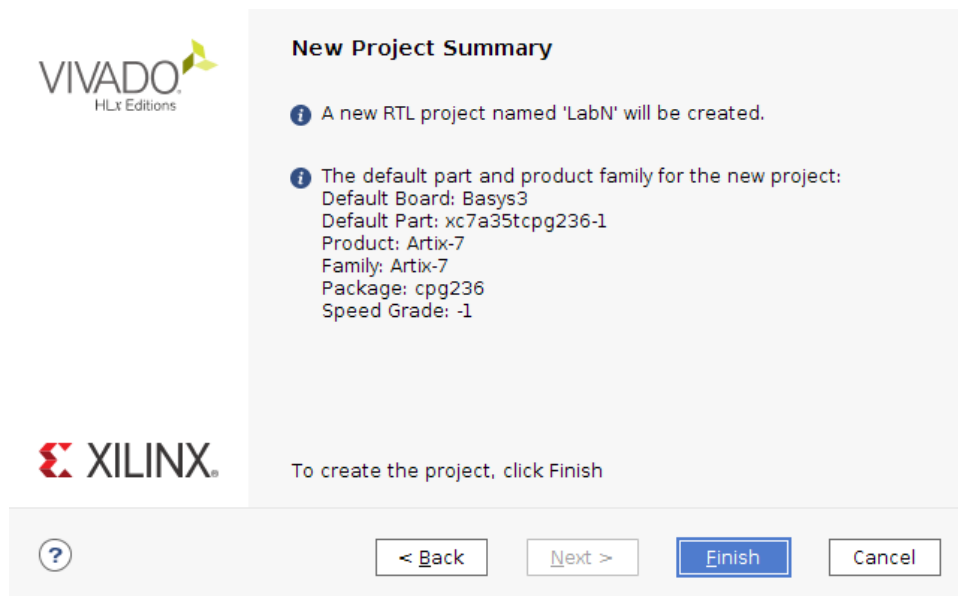


Figure 5 Project summary

*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>.*

## 2.1.2 Define project source files

You are now ready to specify and add your sources to the project.

1. First, we start by adding our design sources (HDL files). This is done by right-clicking anywhere in the “Sources window” and then selecting “Add Sources...” as shown in Figure 6. Or by selecting it from the Project Manager section under the “Flow Navigator” window.

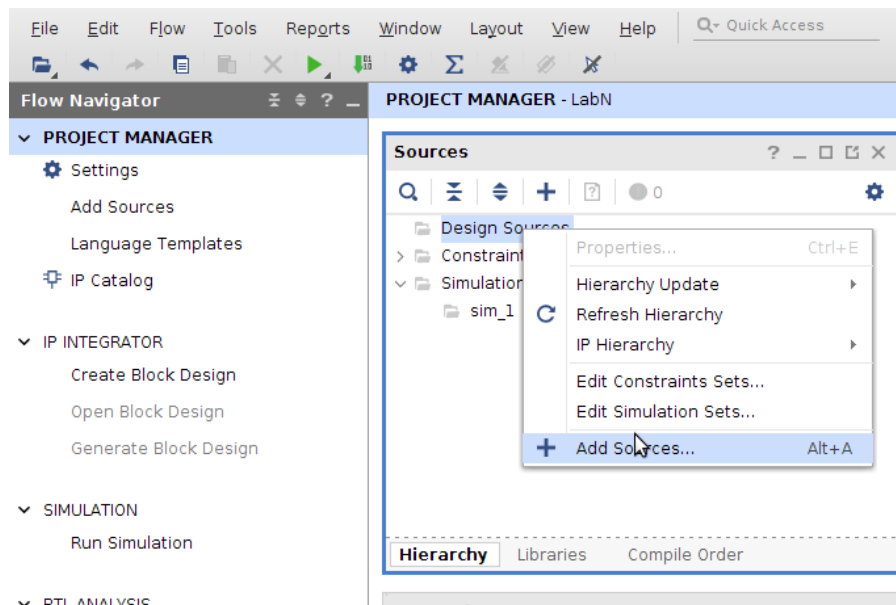


Figure 6 Add source files

2. Next, the wizard from Figure 7 pops up. Select “Add or create design sources” and press Next.

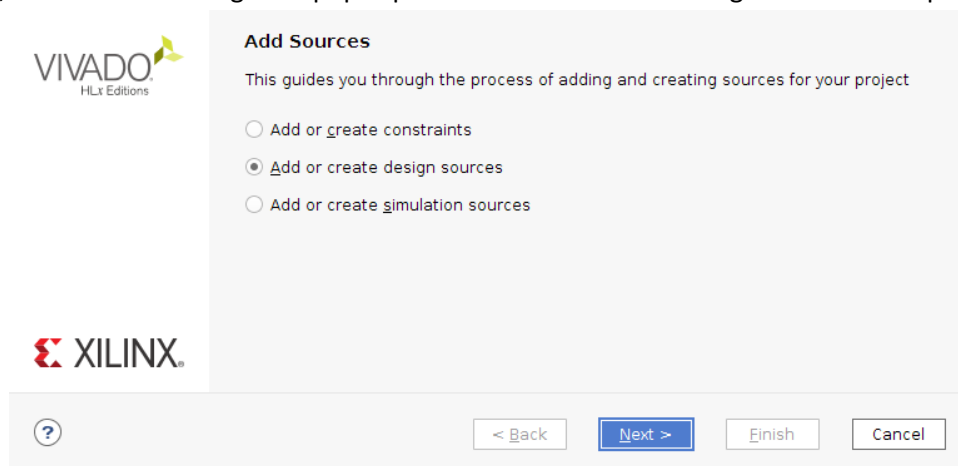


Figure 7 Add sources window



3. We select “Create File...” to add our first VHDL file as shown in Figure 8. Select “VHDL” in the field “File type:” and name the file “full\_adder”. Press “OK” and then “Finish”. By default the file will be created local to your project in a directory named “<project\_name>.srcs/new”. Optionally, you can specify a different location by changing the field “File Location”.

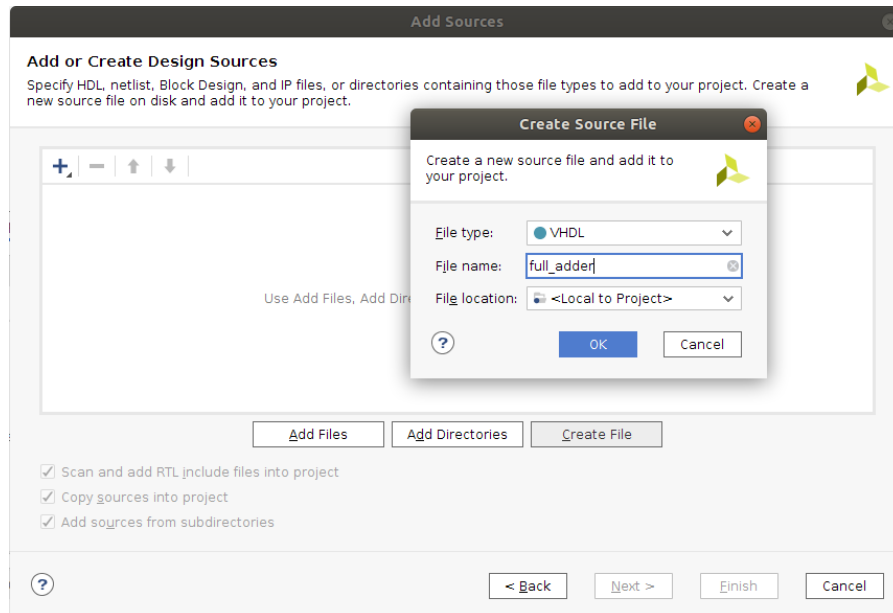


Figure 8 Creating a VHDL source file

4. A wizard popup as shown in Figure 9 is displayed that allows you to optionally define the ports of our newly created file. You can either skip this step by pressing “OK” or complete the form by defining the ports of your design and press “OK”.

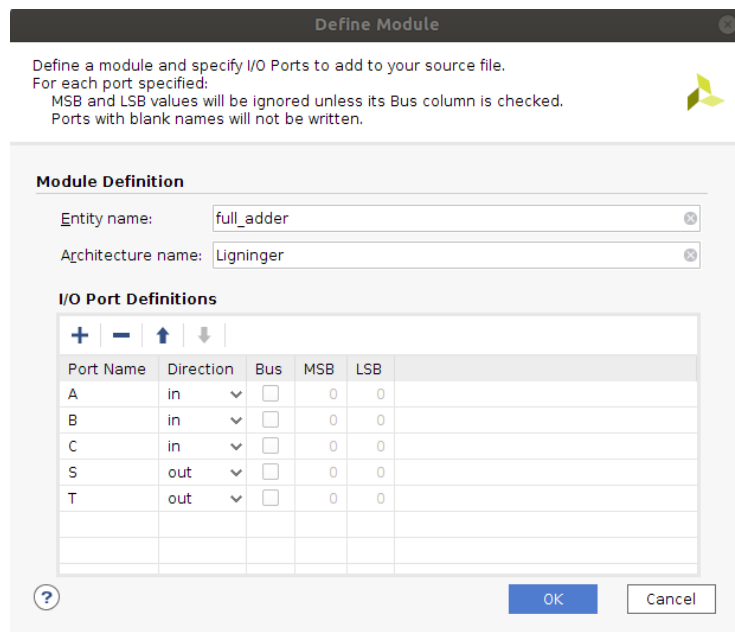


Figure 9 Defining the full adder module

You should now be presented with the main window view of Vivado like Figure 10. The window “Sources” under “Project Manager” lists all the files related to the current project.

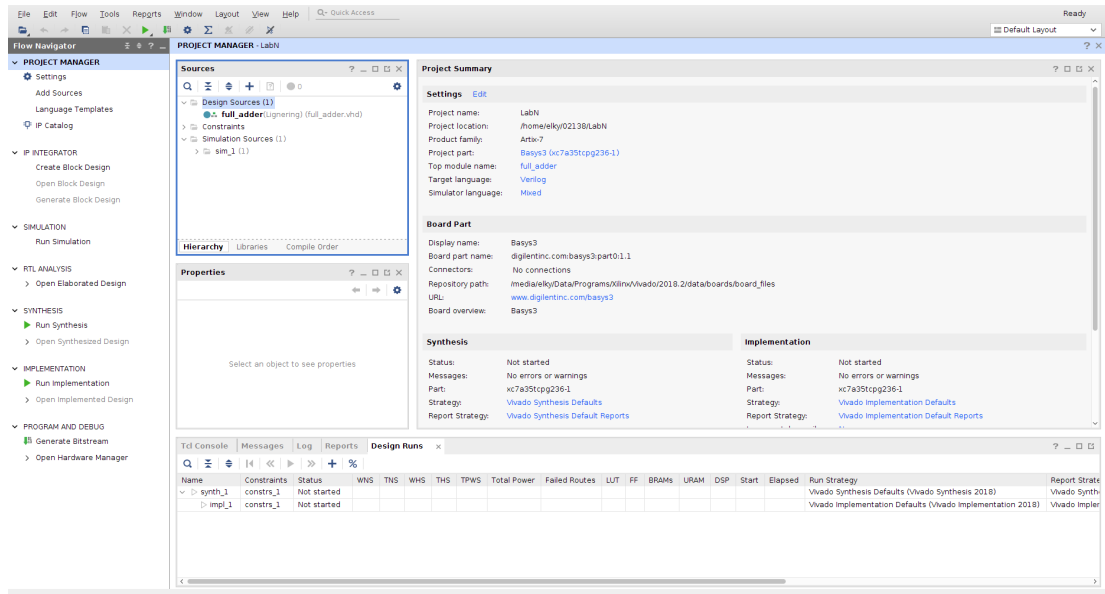


Figure 10 Vivado main window view

5. You are now ready to add the VHDL code that describes the functionality of your digital circuit. For this example, we create a Full Adder. Proceed to double-click on the file “full\_adder.vhd” in the “Sources” window and add the code as shown in Figure 11.

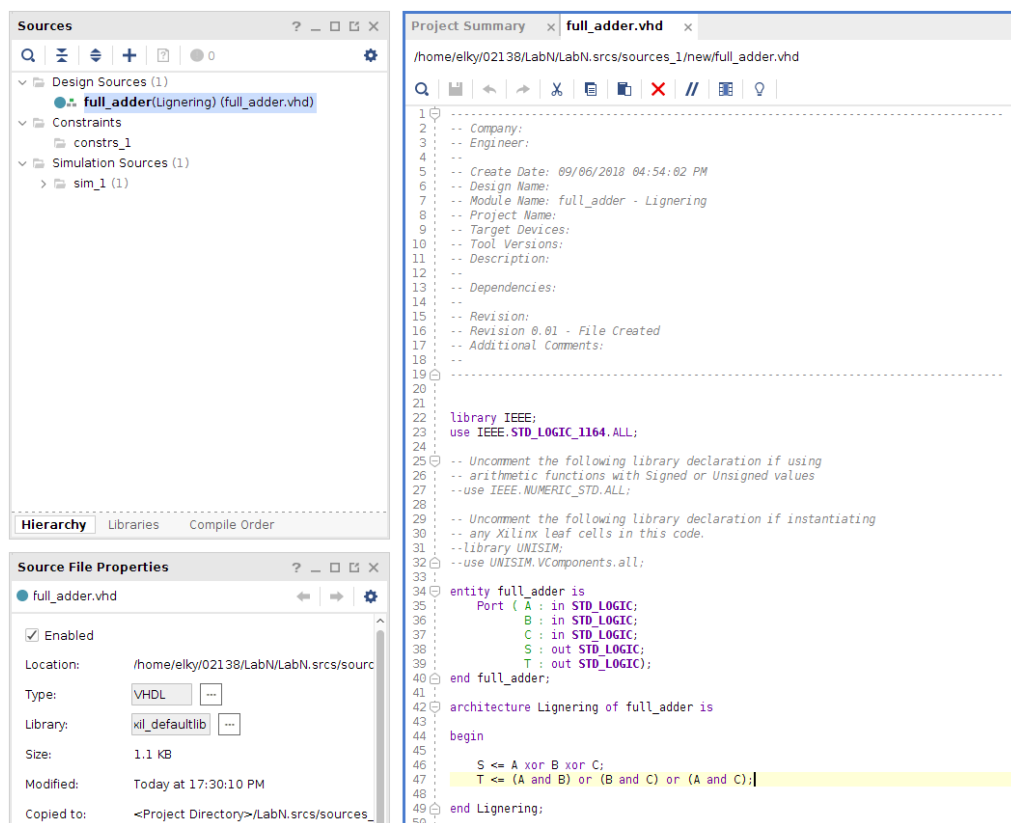


Figure 11 Full adder source code

It is worth mentioning that any text editor can be used for editing the VHDL files. The motivation of using the in-built editor is the automatic syntax check that is provided. The upper part of Figure 12 shows a screenshot with a fully correct code, as indicated by a small green box in the top right corner of the editor. In the lower part of the figure an error has been deliberately introduced to illustrate the automatic check. Note that the small box is now red. Also Vivado provides a line-by-line red indication that can be clicked and place the cursor on the closest line with an error.

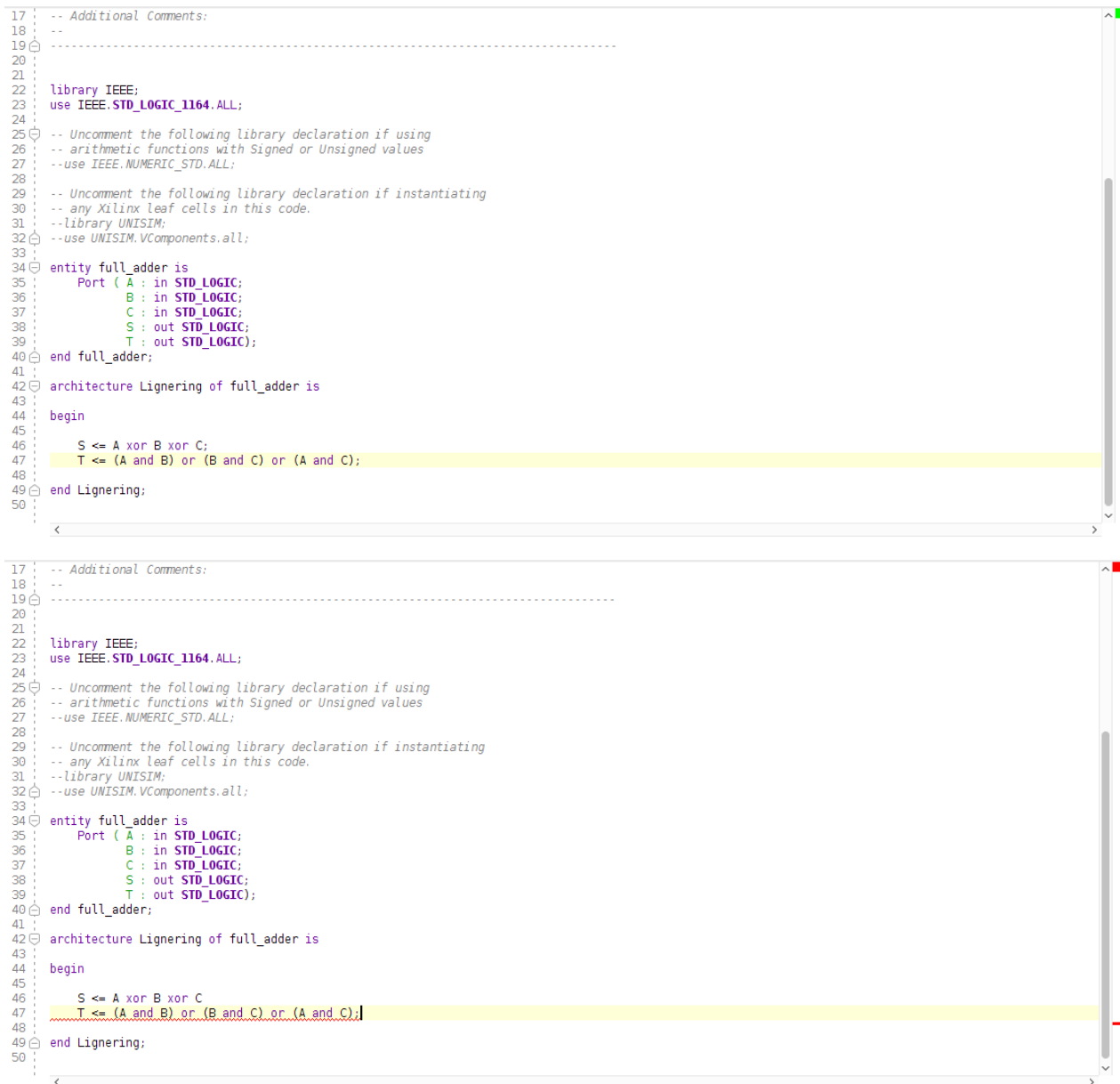


Figure 12 Correct (top-green) and erroneous (bottom-red) automatic syntax check indications

### 2.1.3 Define design constraints

At this point it is possible to simulate the circuit logic (see Section 2.2) or even synthesize and implement the design on the board (see Section 2.4). However, 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 file called **Xilinx Design Constraints** (with the extension .xdc). The XDC-file is provided in the file-sharing of DTU Inside found in here: <https://cn.inside.dtu.dk/cnnet/filessharing/download->

[load/d2e634d8-fb29-431b-a540-bec15e570cc7](https://github.com/Digilent/Basys3/blob/master/Projects/XADC_Demo/src/constraints/Basys3_Master.xdc), otherwise it can be downloaded from Digilent's GitHub repository: [https://github.com/Digilent/Basys3/blob/master/Projects/XADC\\_Demo/src/constraints/Basys3\\_Master.xdc](https://github.com/Digilent/Basys3/blob/master/Projects/XADC_Demo/src/constraints/Basys3_Master.xdc).

1. To add this XDC file to your project click on "File -> Add Sources..." and select "Add or create constraints". Then click "Next>". In the "Add Sources" window click on "Add Files" and select the Basys3.xdc file from the place where you've saved it. Next make sure that the checkbox "Copy constraints files into project" is ticked and click "Finish" as shown in Figure 13.

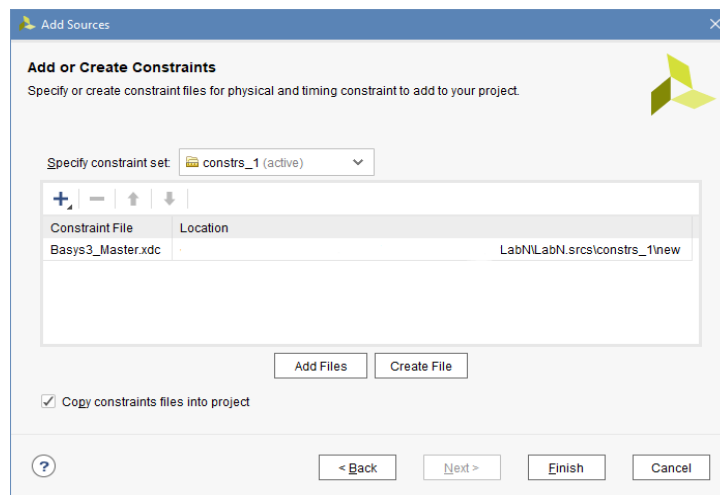


Figure 13 Add design constraints file

2. Under the window "Sources" open the folder "Constraints->constrs\_1" and double-click on the newly added file "Basys3\_Master.xdc" to edit it as shown in Figure 14.

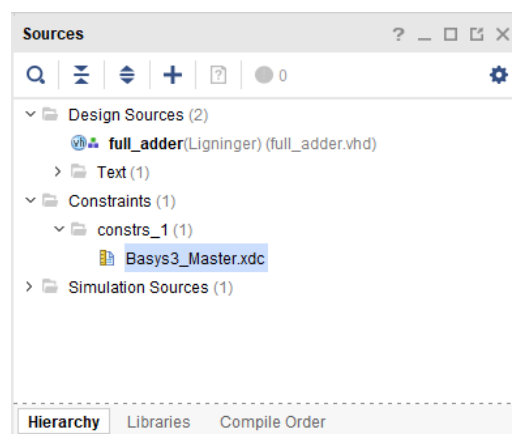


Figure 14 Constraints file location

3. 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 and then connect the outputs “S” and “T” to leds 0 and 1 respectively. First we need to identify in the XDC file where are the constraints related to the Switches/LEDs.

4. Un-comment the 3 lines where **sw[0]**, **sw[1]** and **sw[2]** are defined and then replace the generic names (for example **sw[2]**) with the actual signal names of our Full Adder logic circuit (for example “A”). Do the same for the output signals “S” and “T” which will be connected to the first two LEDs, **led[0]** and **led[1]**. Finally comment the all the lines in the clock signal as this design is asynchronous and does not use a clock. You should end up with the following lines uncommented in your XDC file as shown in Figure 15 (all the other lines in the file are commented out). Make sure you save the XDC file before continuing to the next step.

```
6 set_property IOSTANDARD LVCMOS33 [get_ports *]
7
8 ## Clock signal
9 #set_property PACKAGE_PIN W5 [get_ports clk]
10 #create_clock -add -name sys_clk_pin -period 10.00
11
12 # Switches
13 set_property PACKAGE_PIN V17 [get_ports {A}]
14 set_property PACKAGE_PIN V16 [get_ports {B}]
15 set_property PACKAGE_PIN W16 [get_ports {C}]
16 #set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
17 #set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
18 #set_property PACKAGE_PIN V15 [get_ports {sw[5]}]
19 #set_property PACKAGE_PIN W14 [get_ports {sw[6]}]
20 #set_property PACKAGE_PIN W13 [get_ports {sw[7]}]
21 #set_property PACKAGE_PIN V2 [get_ports {sw[8]}]
22 #set_property PACKAGE_PIN T3 [get_ports {sw[9]}]
23 #set_property PACKAGE_PIN T2 [get_ports {sw[10]}]
24 #set_property PACKAGE_PIN R3 [get_ports {sw[11]}]
25 #set_property PACKAGE_PIN W2 [get_ports {sw[12]}]
26 #set_property PACKAGE_PIN U1 [get_ports {sw[13]}]
27 #set_property PACKAGE_PIN T1 [get_ports {sw[14]}]
28 #set_property PACKAGE_PIN R2 [get_ports {sw[15]}]
29
30
31 # LEDs
32 set_property PACKAGE_PIN U16 [get_ports {S}]
33 set_property PACKAGE_PIN E19 [get_ports {T}]
34 #set_property PACKAGE_PIN U19 [get_ports {led[2]}]
```

Figure 15 XDC modified for Full Adder

## 2.2 Step 2 – Simulation

Using the Vivado design flow, two simulation ways can be identified, (a) an interactive mode where the user can drive the IO signals of the design-under-test using the UI and (b) using a formal test bench that instantiates the design and performs an automated test sequence. Following up, we will present the two main ways of simulation in Vivado.

In both ways to start the simulation click “Run Simulation” under “Flow Navigator” and select “Run behavioral simulation”. After compilation you should be presented with the view shown in Figure 16. Vivado supports different levels of simulation including post-implementation and timing analysis simulations but this advanced features will not be covered in this simple guide.

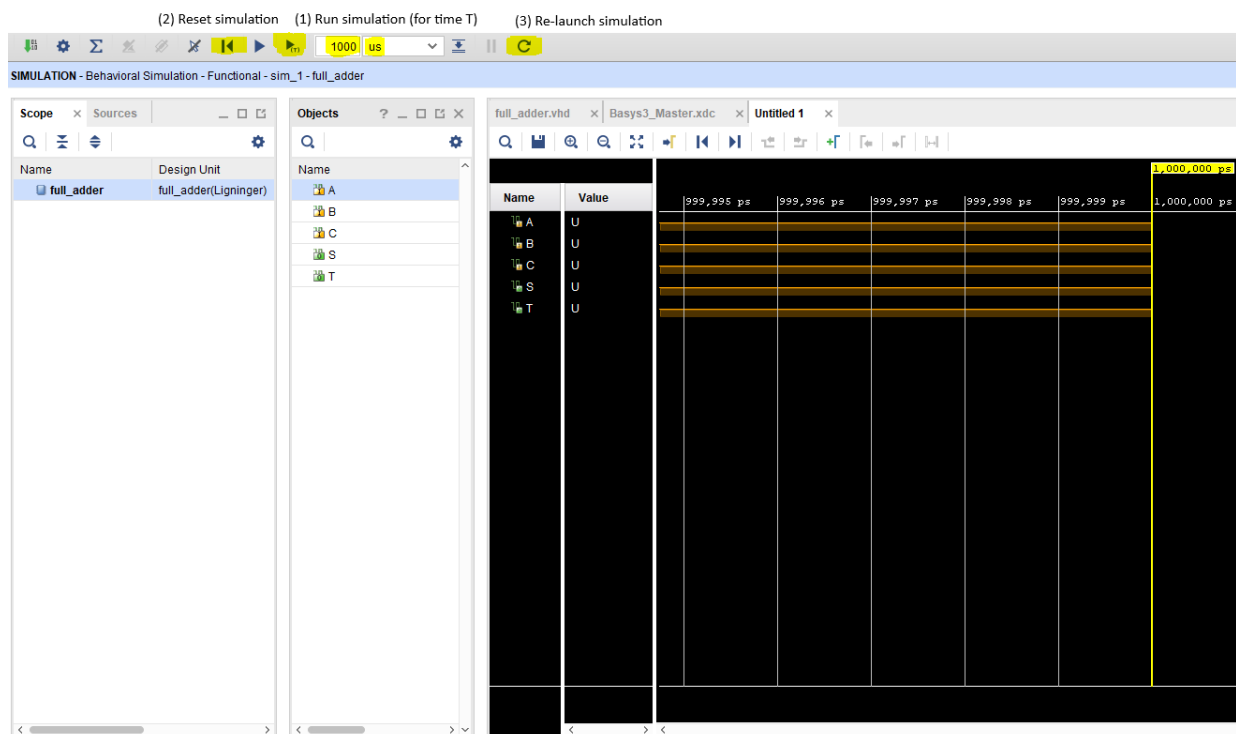


Figure 16 Vivado simulation view (annotated)

The simplest flow of simulation involves the following three (annotated with in the Figure 16) buttons:

- 1) Run simulation (for time T), which simulates your circuit for, the specified in the textbox, amount of time.
- 2) Reset simulation, which resets the simulation. It zeros every simulated signals and reverts the simulation time to zero.
- 3) Re-launch simulation, which is useful when it is needed to re-compile the simulation after performing changes in the code.

### 2.2.1 Interactive simulation

1. To simulate interactively the Full Adder design we first click on “Run Simulation” and then we select “Run behavioral simulation”.
2. After the simulation view opens, we proceed to manual drive click on our input signals that we want to drive. First right-click on signal “A” and then click on “Force constant...” to drive a signal with a constant value throughout the simulation. Enter the value **0** in the field “Force value”. Proceed to do the same for input signal “B” and enter the value **1** as shown in Figure 17.

Enter parameters below to force the signal to a constant value. Assignments made from within HDL code or any previously applied constant or clock force will be overridden.

Signal name: /full\_adder/A  
Value radix: Hexadecimal  
Force value: 1  
Starting after time offset: 0ns  
Cancel after time offset:   
OK Cancel

Signal name: /full\_adder/B  
Value radix: Hexadecimal  
Force value: 1  
Starting after time offset: 0ns  
Cancel after time offset:   
OK Cancel

Figure 17 Force constant signal A and B

3. Finally, to demonstrate another powerful feature of the interactive simulation, click on the input signal “C” and select “Force clock...” to drive the signal with a periodic change of values throughout the simulation. Enter the value **0** in the field “Leading edge value”, the value **1** in the field “Trailing edge value” and finally the value **100ns** in the field “Period” to create a periodic signal of 10MHz as shown in Figure 18. This may give some more interesting results in simulation 😊.

Enter parameters below to force the signal to a constant value. Assignments made from within HDL code or any previously applied constant or clock force will be overridden.

Signal name: /full\_adder/C  
Value radix: Hexadecimal  
Leading edge value: 0  
Trailing edge value: 1  
Starting after time offset: 0ns  
Cancel after time offset:   
Duty cycle (%): 50  
Period: 100ns  
OK Cancel

Figure 18 Force clock signal C

4. To start the simulation enter the value **200 ns** and press the **Play** button. The simulation should run for 200 ns and present the waveform shown in Figure 19.



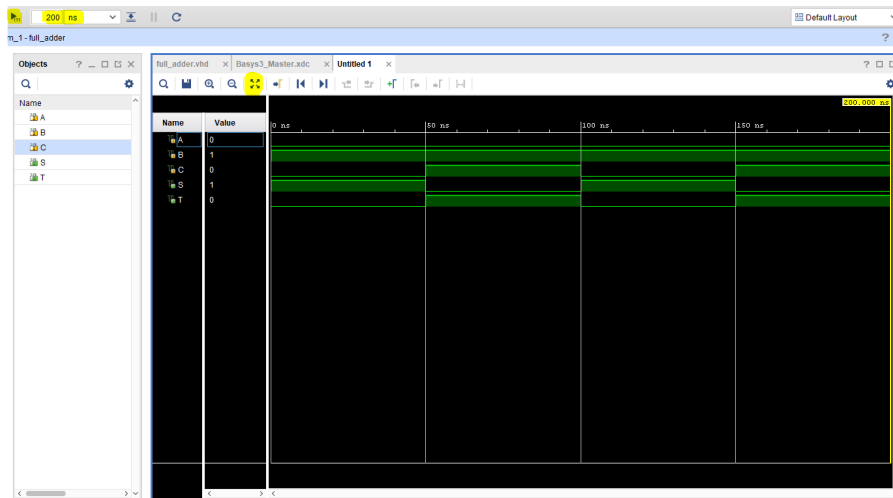



Figure 19 Full adder simulation run: 200 ns

NOTE: If your waveform is not visible or needs to be scaled a quick-and-easy way is to press the  button to zoom and fit the waveform to the window size.

## 2.2.2 Test bench driven simulation

A more formal way to verify the functionality of our circuit is to provide a test bench which drives the input ports of our design and evaluates its outputs. Although verification is a complex engineering field which is out-of-scope of this simple guide, a simple test bench is presented in the context of demonstrating a formal method of simulation in Vivado.

1. On the window “Sources”, right-click on the folder “Simulation Sources”, click “Add sources...” and select “Add or create simulation sources”. Press “Next >”. Click on “Create File” and enter the name “full\_adder\_tb” in the field “File name” as shown in Figure 20. Press “OK” and then “Finish”. Optionally, in the next view fill in the field “Architecture name” with the value “testbench”. Press “OK”.

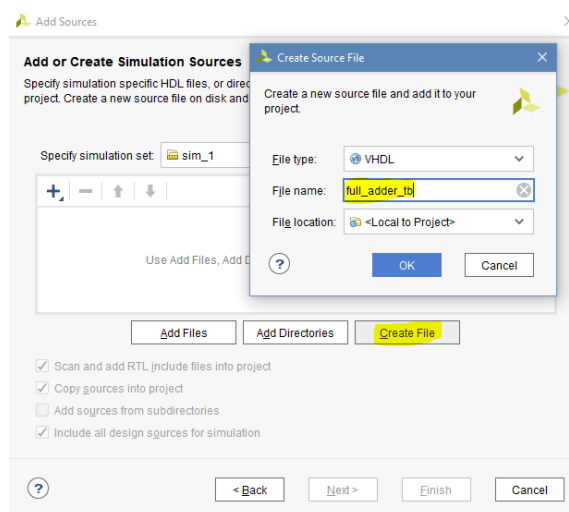


Figure 20 Create simulation source

- Proceed to edit the test bench file “full\_adder\_tb.vhd” by adding the code presented in Figure 21.

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity full_adder_tb is
5  -- Port ( );
6  end full_adder_tb;
7
8  architecture testbench of full_adder_tb is
9
10 component full_adder is
11     Port (
12         A : in STD_LOGIC;
13         B : in STD_LOGIC;
14         C : in STD_LOGIC;
15         S : out STD_LOGIC;
16         T : out STD_LOGIC
17     );
18 end component;
19 signal test_A, test_B, test_C, test_S, test_I : STD_LOGIC := '0';
20 begin
21
22 proc_test: process
23 begin
24     test_A <= '0'; test_B <= '0'; test_C <= '0'; wait for 50 ns;
25     test_A <= '0'; test_B <= '0'; test_C <= '1'; wait for 50 ns;
26     test_A <= '0'; test_B <= '1'; test_C <= '0'; wait for 50 ns;
27     test_A <= '0'; test_B <= '1'; test_C <= '1'; wait for 50 ns;
28     test_A <= '1'; test_B <= '0'; test_C <= '0'; wait for 50 ns;
29     test_A <= '1'; test_B <= '0'; test_C <= '1'; wait for 50 ns;
30     test_A <= '1'; test_B <= '1'; test_C <= '0'; wait for 50 ns;
31     test_A <= '1'; test_B <= '1'; test_C <= '1'; wait for 50 ns;
32     test_A <= '0'; test_B <= '0'; test_C <= '0'; wait for 50 ns;
33     wait;
34 end process;
35
36 fa_inst: full_adder port map(
37     A=>test_A,
38     B=>test_B,
39     C=>test_C,
40     S=>test_S,
41     T=>test_I
42 );
43
44 end testbench;

```

Figure 21 Full adder test bench simulation source

- Before simulating, we must select the file to use for the simulation. This is done by right-clicking the “Simulation sources” under the “Sources” window and select “Set as Top” as shown in Figure 22.

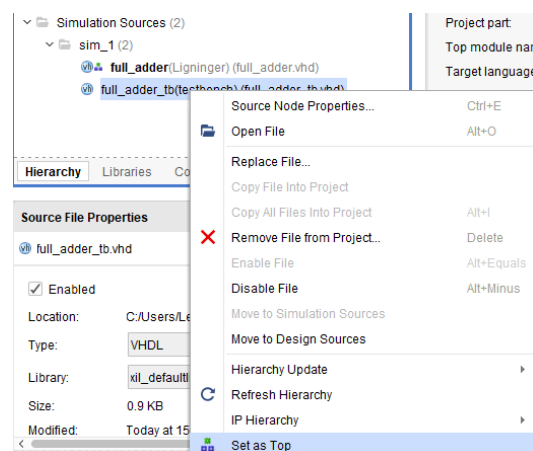


Figure 22 Selecting entity for simulation

4. Proceed by clicking on “Run Simulation” under the “Flow Navigator” window and selecting “Run behavioral simulation”. The simulation should now compile and you should be presented with the Vivado simulation view and the waveform generated from the test bench as seen in Figure 23.

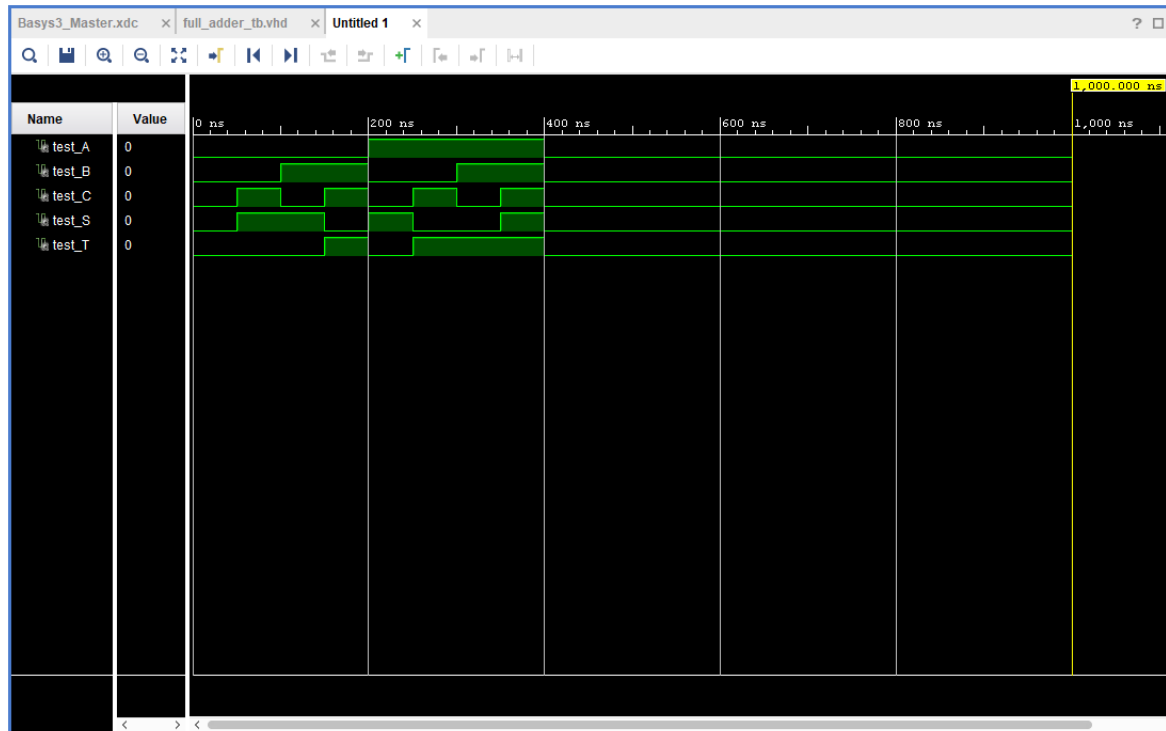


Figure 23 Test bench driven simulation

## 2.3 [OPTIONAL] RTL analysis

Sometimes it is useful to investigate our circuit logic visually at an early development stage using a schematic. After all, it is a circuit that we are building. To view the design register transfer level (RTL) schematic we simply click on “Open Elaborated Design” under “RTL Analysis” in the “Flow Navigator” as shown in Figure 24 to view the circuit RTL schematic as seen in Figure 25.

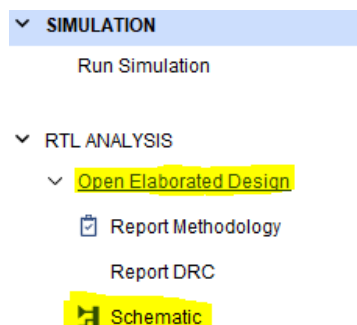


Figure 24 RTL analysis

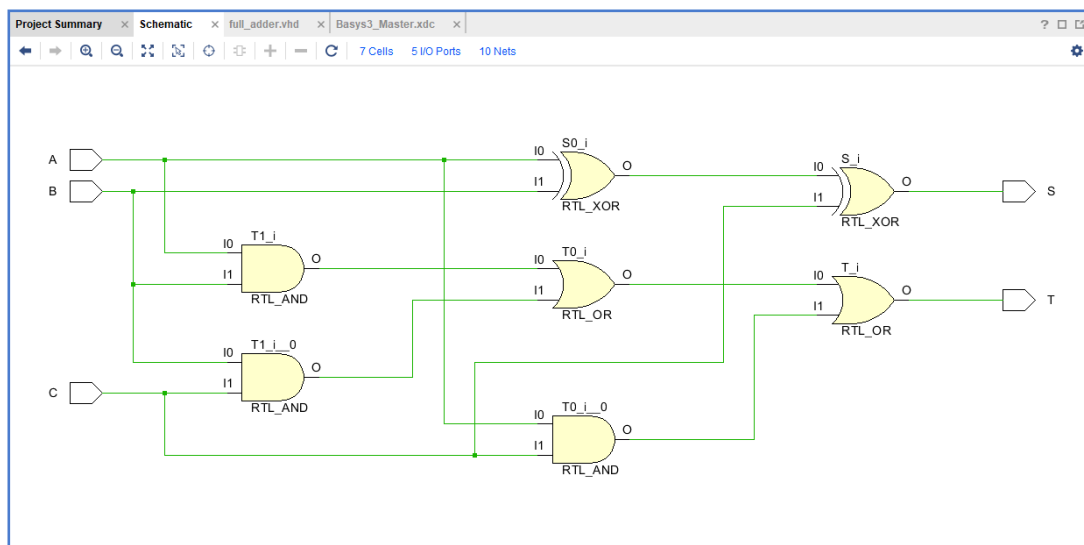


Figure 25 Full adder RTL schematic

## 2.4 Step 3 – Synthesis & Implementation

1. To synthesize the developed design, click on “Run Synthesis” under “Flow Navigator” to initiate the synthesis process. Leave the default values on the window in Figure 26 and press “OK”.

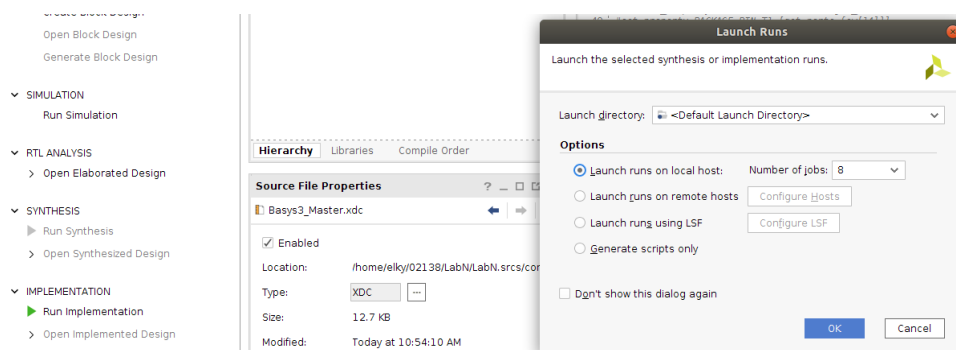


Figure 26 Run synthesis

2. After Synthesis is successfully complete, a popup dialog shows up (Figure 27). To initialize the implementation, simply click “OK”. Otherwise if further investigation is needed on the design such as, resource utilization, timing analysis, post-synthesis schematic, select “Open Implemented Design”.

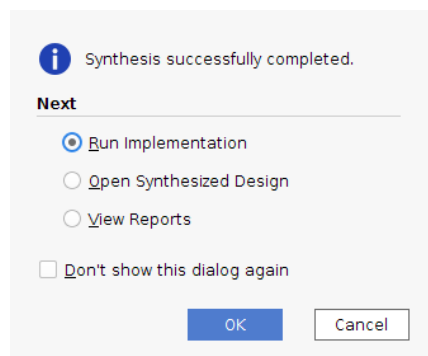
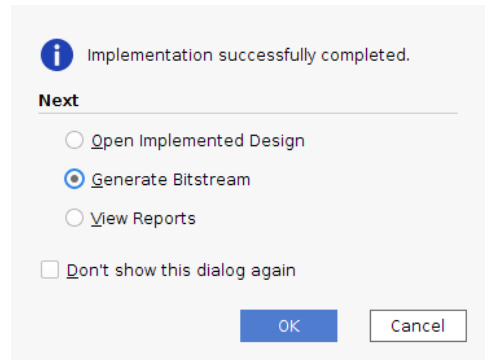


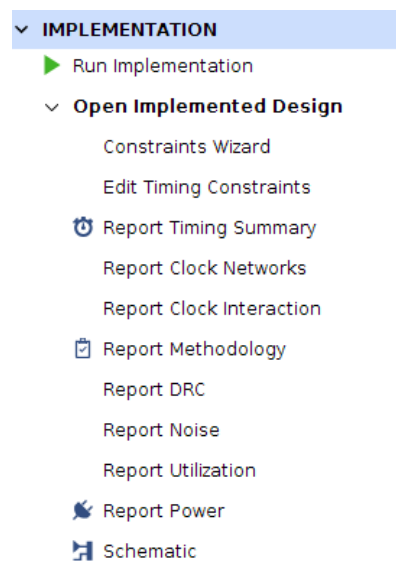
Figure 27 Synthesis completed – Run implementation

3. After Implementation is successfully completed, a popup dialog shows up like Figure 28. To initialize the generation of the programming file, click “Generate Bitstream”.



**Figure 28 Implementation completed - Generate Bitstream**

4. After bitstream generation completes successfully, a popup dialog shows up like Figure 29. Click “OK” to view the implemented design or otherwise select “Open Hardware Manager” and proceed to Section 0 of this guide to program the FPGA.  
Your design is now implemented for the selected device technology. From the “Flow Navigator” you can explore the available options for the implemented design as shown in Figure 29.



**Figure 29 Implemented design menu**

To view the schematic of the implemented design press “Schematic” and the window in Figure 30 will open. In the window bottom left you will find the “Cell Properties”. From there you can investigate different properties for each

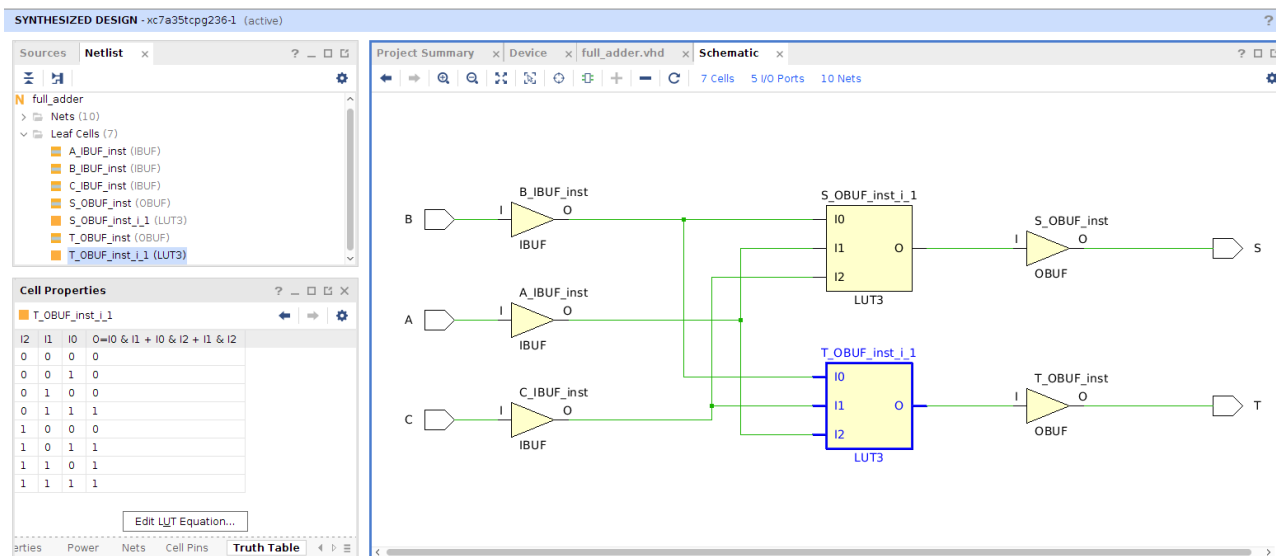


Figure 30 Post-implementation schematic (circuit mapped to FPGA LUTs)

## 2.5 Step 4 – Program & Debug

After successfully implementing your design you can at any point generate the related bitstream programming file and proceed to configure the FPGA. Connect your Basys3 board to the computer and proceed as follows:

1. Simply click on “Generate Bitstream” under “Flow Navigator”. After the bitstream generation is completed successfully, click on “Open Hardware Manager” and then “Open Target” either from the popup dialog of Figure or from the “Flow Navigator” or from the top banner of the main window as seen in Figure 31. Press “Auto connect”.

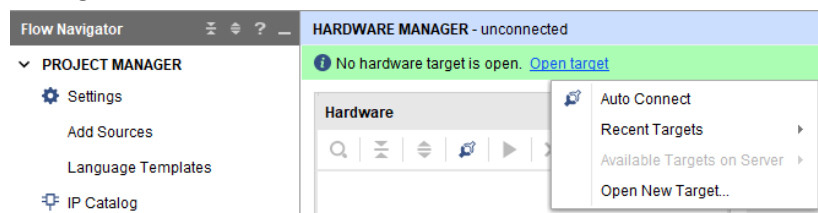


Figure 31 Hardware manager open target device

2. After you connect successfully to your device, press “Program Device” from the top green banner as shown in Figure 32.

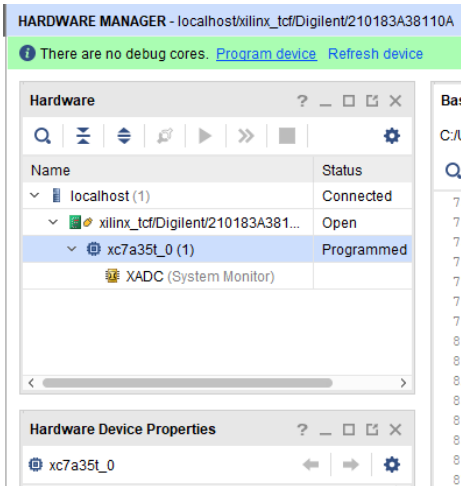


Figure 32 Connected to board - Program



In the next popup window, shown in Figure 33, press “Program”.

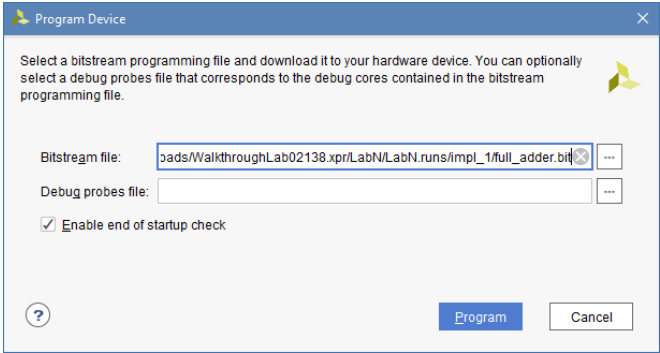


Figure 33 Program device window

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

### 3 Installing the Xilinx Vivado ML Standard tool on your PC

You can download and install the Xilinx Vivado tool on your PC for free. This section briefly lists how to install Vivado ML Standard Edition 2022.1 and add the necessary Basys3 board files.

The tools are large; You will need 89GB disk space during installation, and approximately 50GB remain after installation. The tools used are 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 with Xilinx as well as request license files. I suggest using your DTU student email account for this purpose.

#### 3.1 Register a personal account with Xilinx

- 1a: Go to <https://www.xilinx.com/> and select the "Login/Register" icon top-right.
- 1b: Use your DTU student email account (@student.dtu.dk) and fill in the required information.
- 1c: Activate your account using the instructions in the validation email sent to you.

#### 3.2 Download the Xilinx Vivado ML Standard edition installer

- 2a: Go to <https://www.xilinx.com/support/download.html> .
- 2b: Under "Vivado ML Edition - 2022.1 Full Product Installation", chose "Xilinx Unified Installer 2022.1: Windows Self Extracting Web Installer" to download the Xilinx installer (206MB).

#### 3.3 Install the Xilinx Vivado ML Standard tool

- 3a: Run the downloaded installer.
  - 3b: Enter your account information.
  - 3c: Chose "Vivado", next chose "Vivado ML Standard edition".
  - 3d: Do not modify the installation. If you can't resist, remember that Basys3 uses the Artix-7 series.
  - 3e: Read and accept all user agreements
  - 3f: Chose installation folder "C:\Xilinx"
  - 3g: The Standard tool is now downloaded (28GB) and installed (duration 2 hours).
  - 3h: Final processing: install winpcap. The actual processing depends on the configuration of your pc.
- I got the error message 'Invalid switch "data"' and felt a strange sensation ignoring the message.
- At the end of the installation process the Xilinx License Configuration Manager (XCLM) will start automatically and guide you through licensing. Alternatively, visit <http://www.xilinx.com/getlicense> .

#### 3.4 Download and install the Basys3 board file

Vivado needs the Basys3 board file for generating machine code for the Basys3 board:

- 4a: Download a zip file from <https://github.com/Digilent/vivado-boards/archive/master.zip> .
- 4b: Navigate to folder "C:\Xilinx\Vivado\2022.1\data\boards\".
- 4c: In the zip file navigate to zip folder "vivado-boards-master/new\".
- 4d: Copy folder board\_files to "C:\Xilinx\Vivado\2022.1\data\boards\board\_files".

#### 3.5 Open Vivado and create a new project

Open "Vivado 2022.1" (not Vitis 2022.1) and "Create Project". Chose a project name and a project folder of type "RTL Project". Select "Next" with no source or constraints files. At "Default Part" chose menu "Boards" to select "Basys3" from the list. You should now be able to use Vivado for your Basys3 board.



The Basys 3 is an entry-level FPGA development board designed exclusively for [Vivado Design Suite](#), featuring Xilinx Artix-7 FPGA architecture. The Basys 3 board manual, schematic and XDC file can be found at DTU Learn.

For additional information about the board visit: <https://store.digilentinc.com/basys-3-artix-7-fpga-trainer-board-recommended-for-introductory-users/>

To find official examples as well as the Basys 3 XDC file visit the official repository of Digilent for Basys 3 board at: <https://github.com/Digilent/Basys3>