Lab 0 - Digital Systems Design Flow

ENGIN 341 – Advanced Digital Design

University of Massachusetts Boston

Introduction

This lab guides you through the process of using the **Vivado Design Suite** to create a simple **VHDL** design for the **ZYBO Development Board**. You will go through the Vivado installation and Zybo Board setup. You will also be introduced to several tools in Vivado that aid the design process.

Objectives

After completing this lab, you will be able to:

- Create a new project in Vivado using **VHDL**
- Use the **ZYBO Master Constraint file** to constrain the pin locations on the ZYBO Board
- **Simulate** the design and observe Input to Output behavior
- **Synthesize** and **Implement** the design
- Program the completed design onto the **ZYBO Development Board FPGA**

Design Description

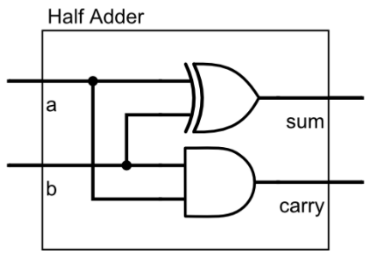
For this introductory lab, a simple Half-Adder will be designed and implemented. A Half-Adder consists of 2 inputs, and 2 outputs. The truth table of a Half-Adder is found in the table below, along with the schematic diagram.

Resulting in the expressions:

Sum = A xor B;

Carry = A and B

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Outline

- Part 0: Preliminary Setup
- Part 1: Creating a New Project in Vivado
- Part 2: Adding/Editing Design Sources and Constraints
- Part 3: Elaborating and Synthesizing the Design
- Part 4: Synthesizing, Implementing, and Programming the ZYBO Board
- Part 5: Adding a second half-adder

PART 0. PRELIMINARY SETUP

Before we start a new project, there are a few preliminary steps to take. We will install Vivado, setup the ZYBO hardware, and add the ZYBO support libraries to Vivado. References for the ZYBO boards are below – you may have the original ZYBO board or the ZYBO Z7.

- https://reference.digilentinc.com/reference/programmable-logic/zybo/start
- https://reference.digilentinc.com/reference/programmable-logic/zybo-z7/start

Step 0-1: Your first step is to install the Vivado development environment. For this, you can follow the instructions in the *LAB0 – Vivado Installation* file. The installation process could take anywhere from 20 minutes to an hour depending on your computer and network speed.

- **NOTE:** You may also follow the installation instructions here

Step 0-2: Next, you will need to prepare the ZYBO Board. After taking the board out of its packaging look at the top right corner of the board. There you will see a set of 4 jumpers with a blue cap covering the center 2. These jumpers control the boot mode for the ZYBO board. To set the board to wait for a program from the computer to be sent to it, ***move the cap to the right most pins***, labeled on the board as the JTAG setting, as shown in the image below (*Figure 1*).

Step 0-3: The next step is to connect the board to the computer:

- In the top left corner along the left side of the board is a MicroUSB port. Use the USB to MicroUSB cable provided in the ZYBO kit to connect from this port to your computer.
- You will also see another set of jumpers next to the MicroUSB port (Figure 2). These determine how the Board is powered. The default setting allows the board to be powered through the USB. You should *use the USB power for lab assignments* in this class; but if you wanted to use the wall adapter to power the board you would shift the blue cap from the bottom 2 pins to the top 2 pins.
- Finally, turn the board on using the power switch (also in the top left corner).

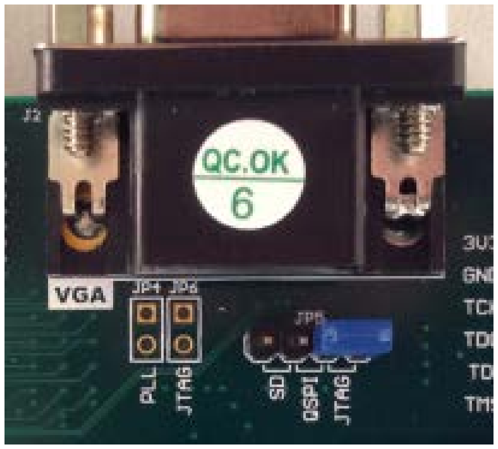


Figure 2: Step 0-2

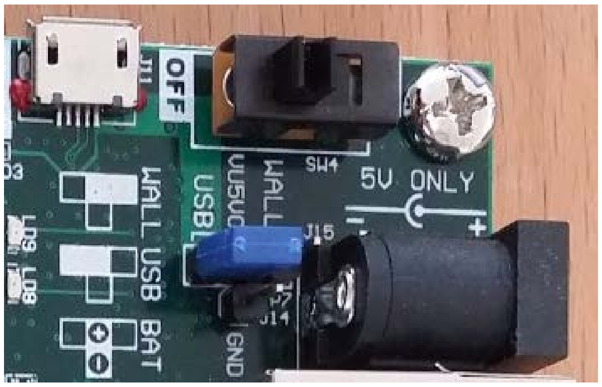


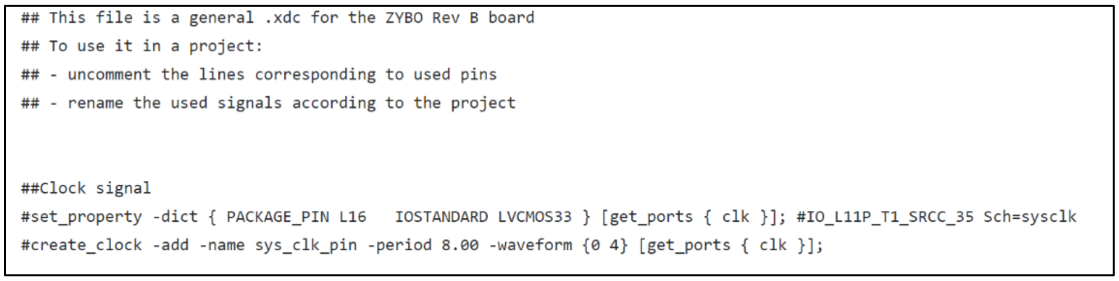
Figure 1: Step 0-3

Step 0-4: Next, you will Download the ZYBO Board Files and add them to the Vivado library. These files allow Vivado to know what peripheral devices and components are available on the board that you are using. The ZYBO is a product of Digilent; therefore, Digilent provides instructions for accessing the board files and making them accessible in Vivado. You can find detailed instructions here that describe how to download the Board files and make them available in Vivado.

- Find the *board_files* directory on your computer.
    - Most likely in *C:\Xilinx\Vivado\2020.2\data\boards\board_files*
- Download the board files from Digilent's github page.
    - https://github.com/Digilent/vivado-boards/archive/master.zip
- Extract the contents of the zip drive to your computer.
- Copy the desired directories from the *new/board_files* folder (extracted from the zip file) to the *board_files* directory on your computer (from the first step above).
    - You should add the board files for *zybo*, *zybo-z7-10*, and *zybo-z7-20*.

Step 0-5: Finally, download the *ZYBO Master XDC File for Vivado Designs*. This file contains a list of all the pins on the ZYBO Board, making the process of assigning pin constraints much simpler.

- Find the .xdc file for your specific board (Zybo or Zybo-Z7) here:
    - https://github.com/Digilent/digilent-xdc/
- Download the file and save it somewhere accessible on your local drive.
    - Note: Make sure to download the xdc file and ***not the webpage***! If you right-click and "save as" from github, you will end up with the html for the link rather than the .xdc file. You can download the directory from github with the "Code -> Download ZIP" option in the upper right corner of the page. If you download a .zip file of the full directory, make sure to extract the XDC files before trying to use it in a project. The .xdc file should look something like the image below (Figure 3).

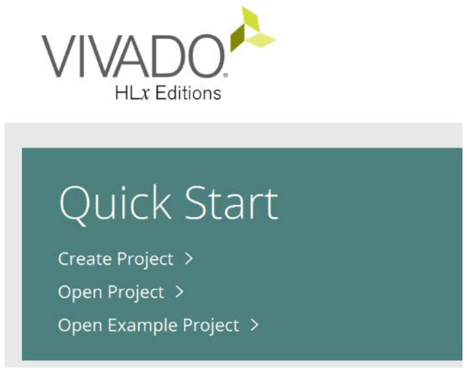


```
## This file is a general .xdc for the ZYBO Rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used signals according to the project


##Clock signal
#set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L11P_T1_SRCC_35 Sch=sysclk
#create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];
```

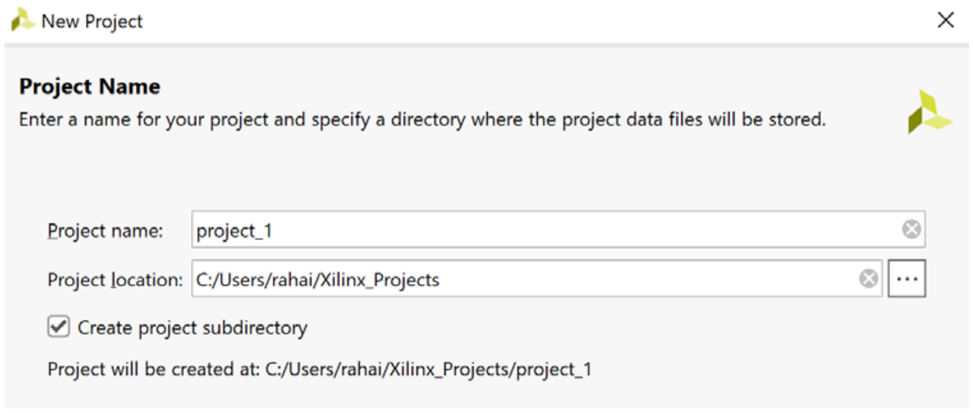
Figure 3: First 7 lines of Zybo-Master.xdc

PART 1. CREATING A NEW VIVADO PROJECT

Step 1-1: To create a new project, launch Vivado (not Vivado HLS) and select *Create New Project*.



Step 1-2: Select *Next*, then in the *Project Name* window, name your project whatever makes sense to you. Also make sure the *Create project subdirectory* box is checked. Then select *Next*.


New Project

Project Name

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

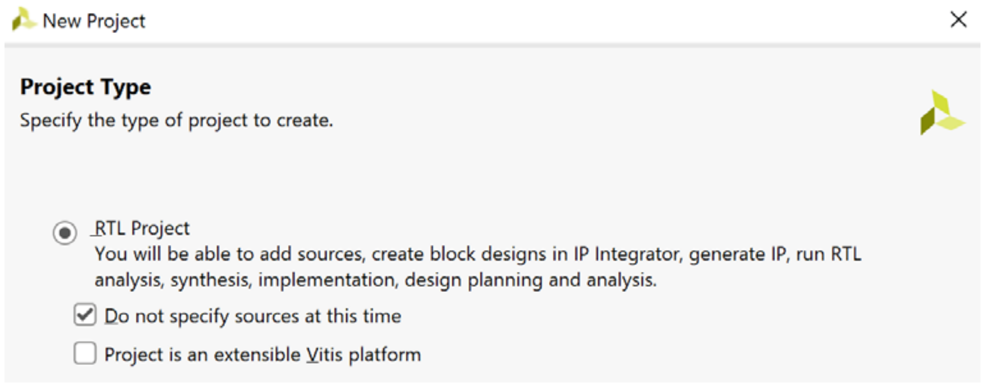
Project name: project_1

Project location: C:/Users/rahai/Xilinx_Projects

☑ Create project subdirectory

Project will be created at: C:/Users/rahai/Xilinx_Projects/project_1

Step 1-3: In the *Project Type* window, select *RTL Project*, and for the tutorial, check the *Do not specify sources at this time* box. Select *Next*.


New Project

Project Type

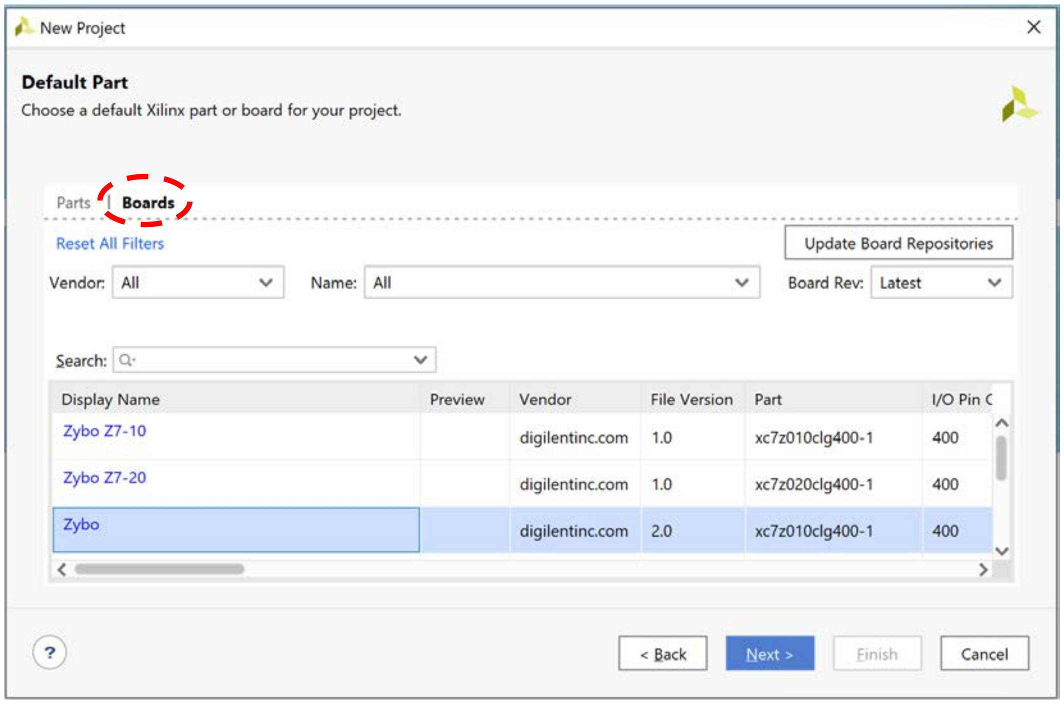
Specify the type of project to create.

● 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

☐ Project is an extensible Vitis platform

Step 1-4: In the *Default Part* window, select *Boards*, then find the *ZYBO* board and select it (or Zybo Z7-10 if you have this board). This will only work if you have downloaded the ZYBO board files and added them to the Vivado library as instructed in the preliminary part of this tutorial.



New Project

Default Part

Choose a default Xilinx part or board for your project.

Parts | **Boards**

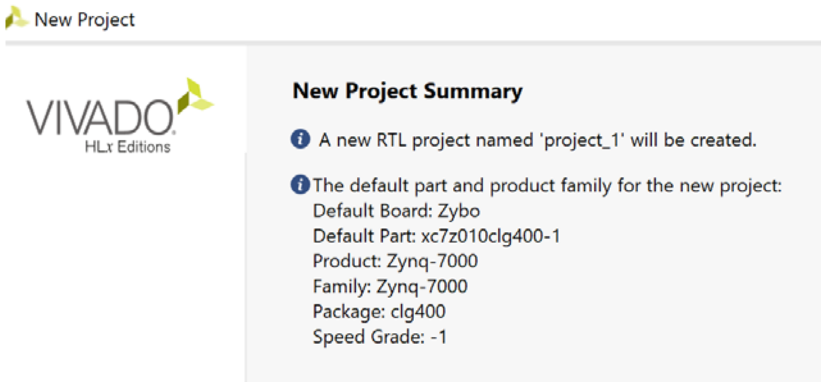
Reset All Filters

Vendor: All | Name: All | Board Rev: Latest

Search:

Display Name	Preview	Vendor	File Version	Part	I/O Pin C
Zybo Z7-10		digilentinc.com	1.0	xc7z010clg400-1	400
Zybo Z7-20		digilentinc.com	1.0	xc7z020clg400-1	400
Zybo		digilentinc.com	2.0	xc7z010clg400-1	400

Step 1-6: In the next window is the *New Project Summary*. The details in your summary should match the ones in the image below. If they do, then go ahead and select *Finish*.



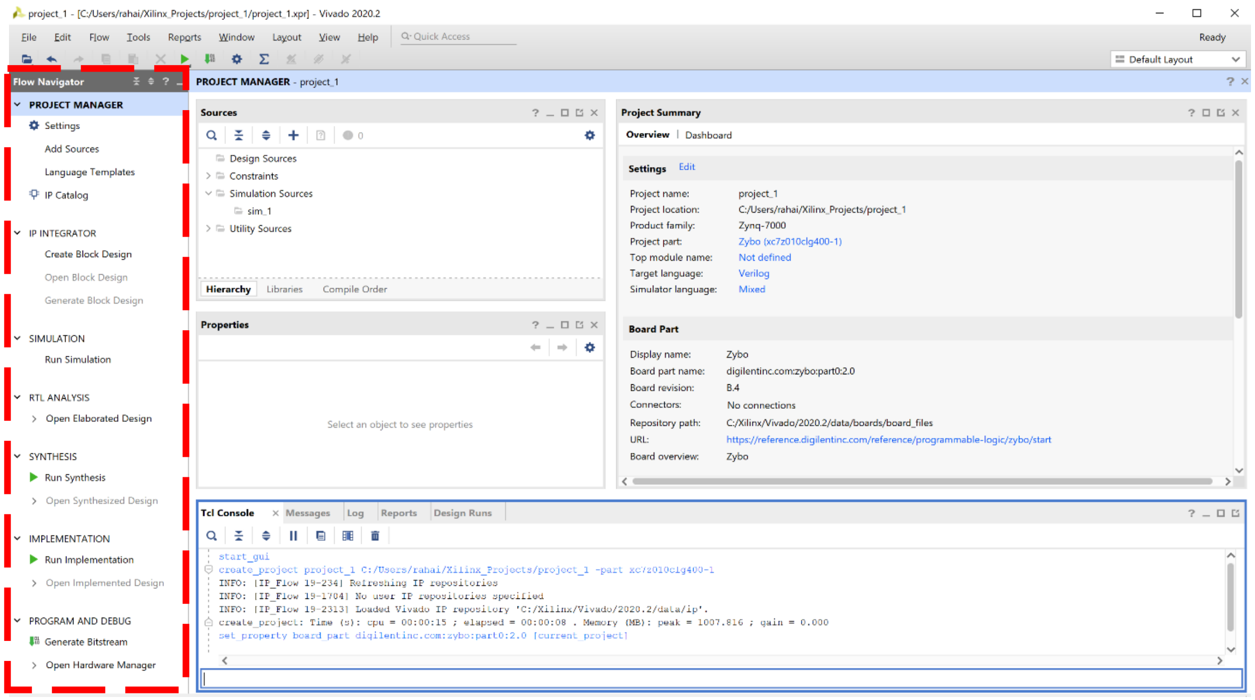
New Project

New Project Summary

A new RTL project named 'project_1' will be created.

The default part and product family for the new project:
Default Board: Zybo
Default Part: xc7z010clg400-1
Product: Zynq-7000
Family: Zynq-7000
Package: clg400
Speed Grade: -1

Step 1-7: Now we are at the main window. On the left is the *Flow Navigator* where you can access various tools that will allow you to analyze your design and program it onto the ZYBO board.



Settings Edit

Project name: project_1
Project location: C:/Users/rahai/Xilinx_Projects/project_1
Product family: Zynq-7000
Project part: Zybo (xc7z010clg400-1)
Top module name: Not defined
Target language: Verilog
Simulator language: Mixed

Board Part

Display name: Zybo
Board part name: digilentinc.com:zybo:part0:2.0
Board revision: B.4
Connectors: No connections
Repository path: C:/Xilinx/Vivado/2020.2/data/boards/board_files
URL: https://reference.digilentinc.com/reference/programmable-logic/zybo/start
Board overview: Zybo

```
start_gui
create_project project_1 C:/Users/rahai/Xilinx_Projects/project_1 -part xc7z010clg400-1
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository 'C:/Xilinx/Vivado/2020.2/data/ip'.
create_project: Time (s): cpu = 00:00:15 ; elapsed = 00:00:08 . Memory (MB): peak = 1007.816 ; gain = 0.000
set_property board_part digilentinc.com:zybo:part0:2.0 [current_project]
```

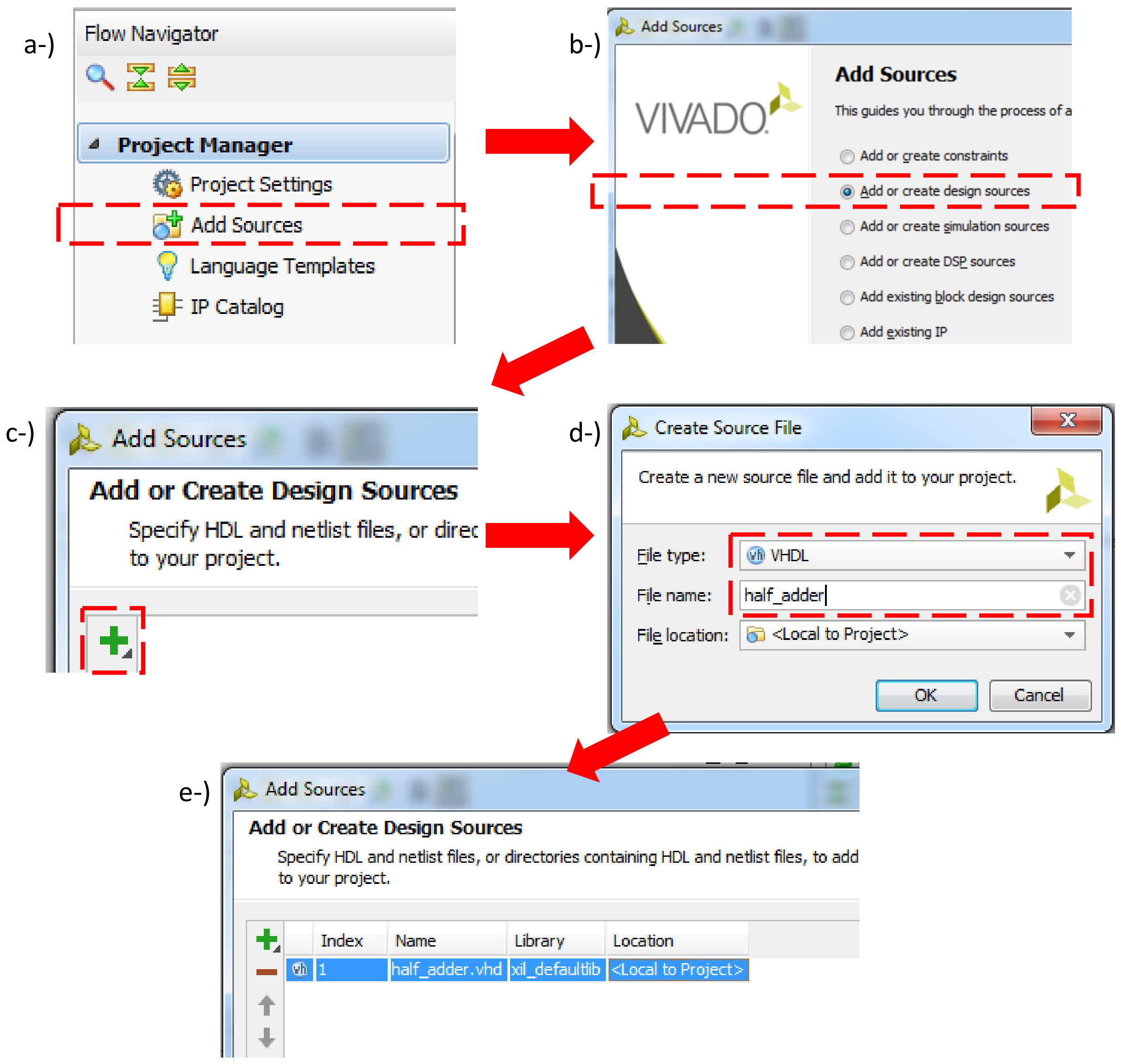
The *Flow Navigator* mimics the general FPGA design process. A Vivado "flow" is an inner window arrangement based on the functionality of whatever is selected in the flow navigator. Starting from top to bottom, the different Flows are:

1. *Project Manager* – A flow where you can add/manage project source files and hierarchy.
2. *IP Integrator* – A flow used to incorporate turnkey Intellectual Property (IP) designs from other sources.
3. *Simulation* – A flow used to simulate the operation of the design.
4. *RTL Analysis* – A flow that enables the analysis of your design's functionality at the RTL (Register Transfer Level) design abstraction level.
5. *Synthesis* – A flow used for mapping the RTL design onto specific hardware where the user can specify further constraints to be met in the *Implementation* phase.
6. *Implementation* – A flow where the *Synthesis* hardware and timing constraints are elaborated.
7. *Program and Debug* – A flow used for generating and transferring a bit stream of the design and optionally a debug core to the target device.

PART 2. ADDING/EDITING DESIGN SOURCES AND CONSTRAINTS

Step 2-1: Now we can begin designing the Half-Adder and assigning the board pins. First, we will add the source file.

a) In *the Flow Navigator* under *Project Manager* select *Add Sources*.
b) Select *Add or create design sources* then select *Next*.
c) Now click on the *plus sign* and select *Create File*.
d) In the *Create Source File* window, name the source file **half_adder**, and make sure the *File type* is *VHDL*.
e) The source file should now be listed in the *Add or Create Design Sources* window. Select *Finish*.



Add Sources

This guides you through the process of a

- ( ) 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

Add or Create Design Sources

Specify HDL and netlist files, or direc
to your project.

Create a new source file and add it to your project.

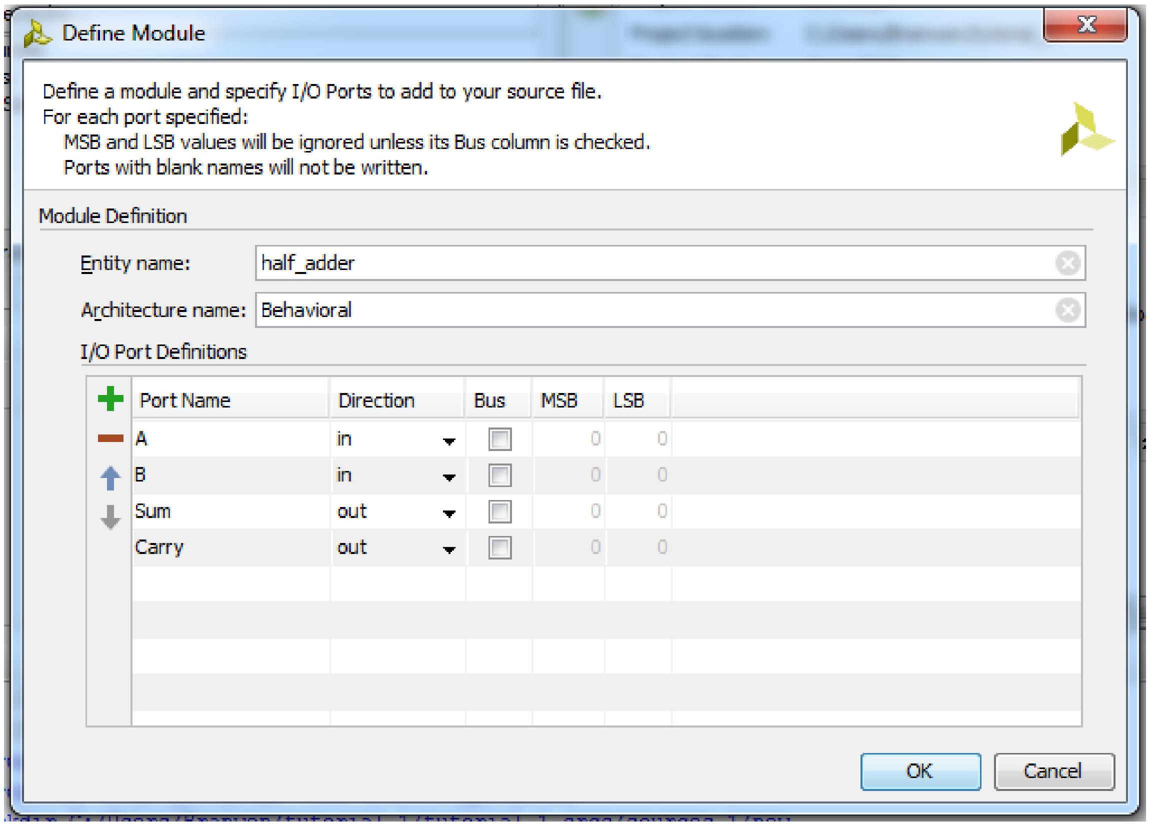
File type: VHDL
File name: half_adder
File location: <Local to Project>

Add or Create Design Sources

Specify HDL and netlist files, or directories containing HDL and netlist files, to add to your project.

Index	Name	Library	Location
1	half_adder.vhd	xil_defaultlib	<Local to Project>

Step 2-2: Now the *Define Module* window pops up. Here is where you can enter the input/output ports for our design. Click the *plus sign* to add additional ports and name them as shown below. Make sure to change the *Direction* of **Sum** and **Carry** to *out*, then click *OK*.



Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

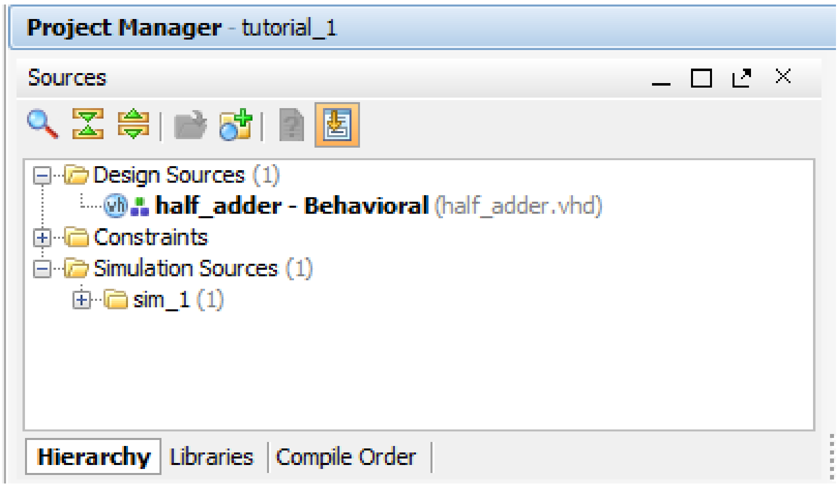
Entity name: half_adder

Architecture name: Behavioral

I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
A	in	☐	0	0
B	in	☐	0	0
Sum	out	☐	0	0
Carry	out	☐	0	0

Step 2-3: Now the design source file we just created is listed in the *Sources* window of the *Project Manager*. Next to the file name is a small symbol, a green block on top of two blue blocks. This symbol means that this file is the *Top Level file*. It is not important for this design but later on when we get into designs with multiple sources, this will be important.



Project Manager - tutorial_1

Sources

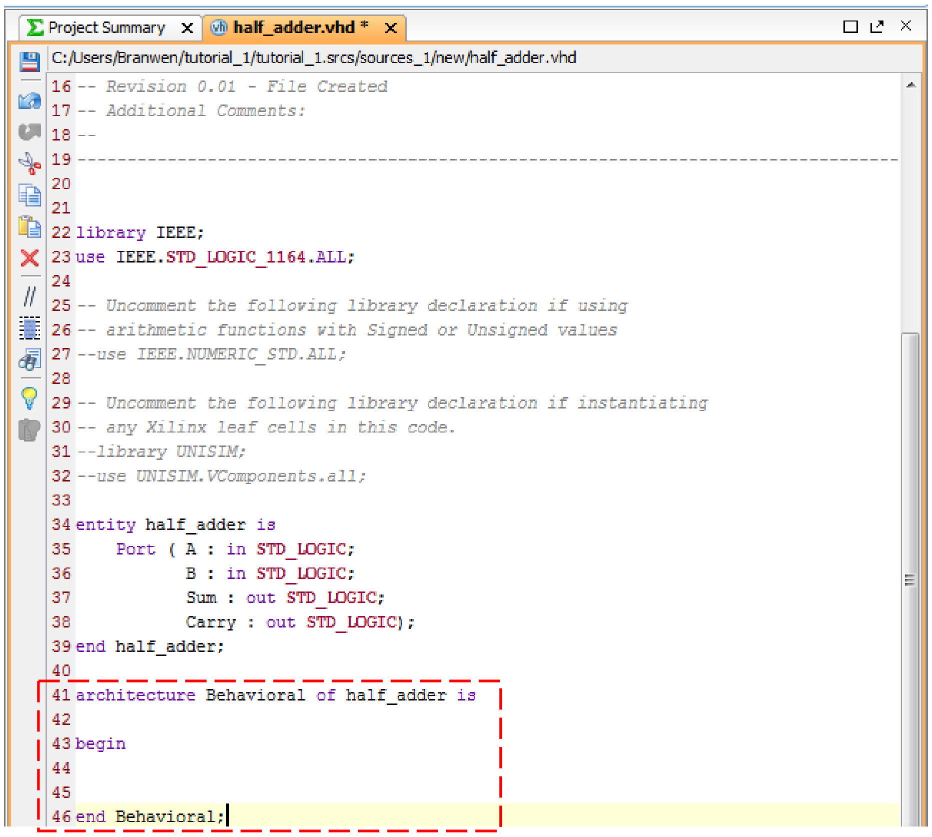
Design Sources (1)
half_adder - Behavioral (half_adder.vhd)
Constraints
Simulation Sources (1)
sim_1 (1)

Step 2-4: Now, we can edit the Source file.

a) Double click on the source file to bring it up in the window on the right. There are a few parts to this VHDL design source file, but we will cover those details in a later tutorial.

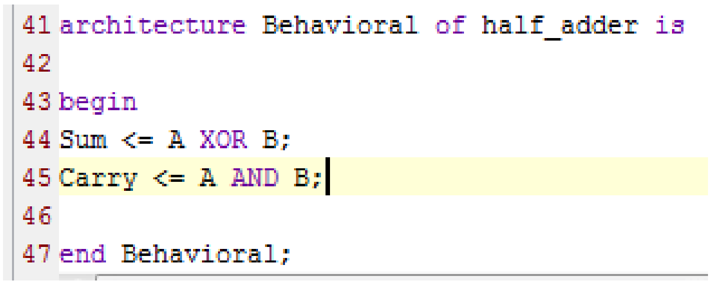
b) For now, we are just adding some simple *Architecture* statements to setup up the expressions for the Half-Adder. Add the statements shown in the image below between the *begin* and *end Behavioral;* statements in the *Architecture* portion of the source file.

c) Save this file using the *floppy disk* symbol in the top left corner of the editing window.



C:/Users/Branwen/tutorial_1/tutorial_1.srcs/sources_1/new/half_adder.vhd

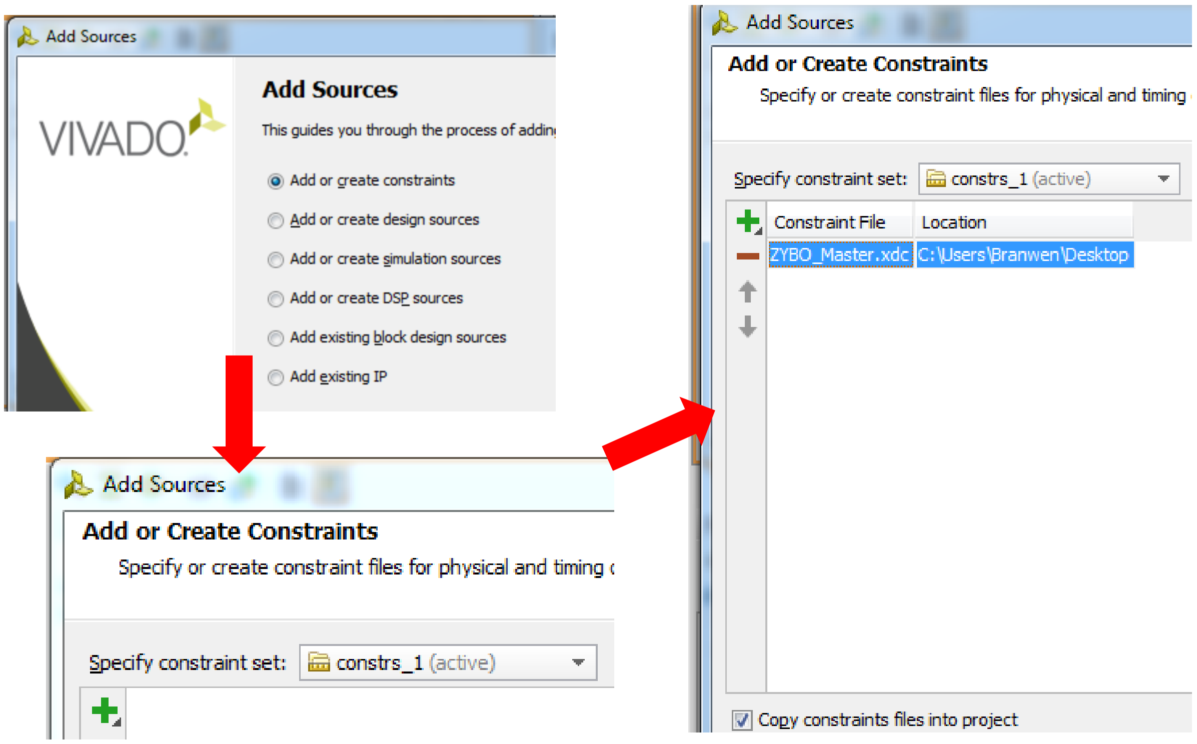
```
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity half_adder is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           Sum : out STD_LOGIC;
           Carry : out STD_LOGIC);
end half_adder;

architecture Behavioral of half_adder is

begin


end Behavioral;
```



```
architecture Behavioral of half_adder is

begin
Sum <= A XOR B;
Carry <= A AND B;

end Behavioral;
```

Step 2-5: The next step is to create a constraints file.

a) Go back to the *Flow Navigator* and select *Add Sources* again. This time select *Add or create constraints*.
b) In the *Add or Create Constraints* window click on the *plus* sign and select *Add Files*.
c) Browse to where you saved the *ZYBO Master XDC* file you downloaded at the beginning of this tutorial and select it.
d) Now the *Master XDC* file is listed in the *Add or Create Constraints* window. Make sure that the *Copy constraints files into project* box is checked, otherwise any edits you make to the file will be saved to the original file. This would affect other projects linked to this file and could cause serious issues, since the constraint file is used to map the ports from your design to the physical pins on the *ZYBO*.


Add Sources

VIVADO

Add Sources

This guides you through the process of addin

(●) 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

Add Sources

Add or Create Constraints

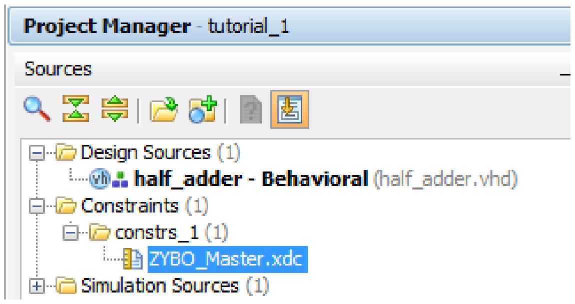
Specify or create constraint files for physical and timing

Specify constraint set: constrs_1 (active)

Constraint File	Location
ZYBO_Master.xdc	C:\Users\Branwen\Desktop

☑ Copy constraints files into project

Step 2-6: Now the constraint file is listed under the *Constraints folder* in the *Source* window of the *Project Manager*.

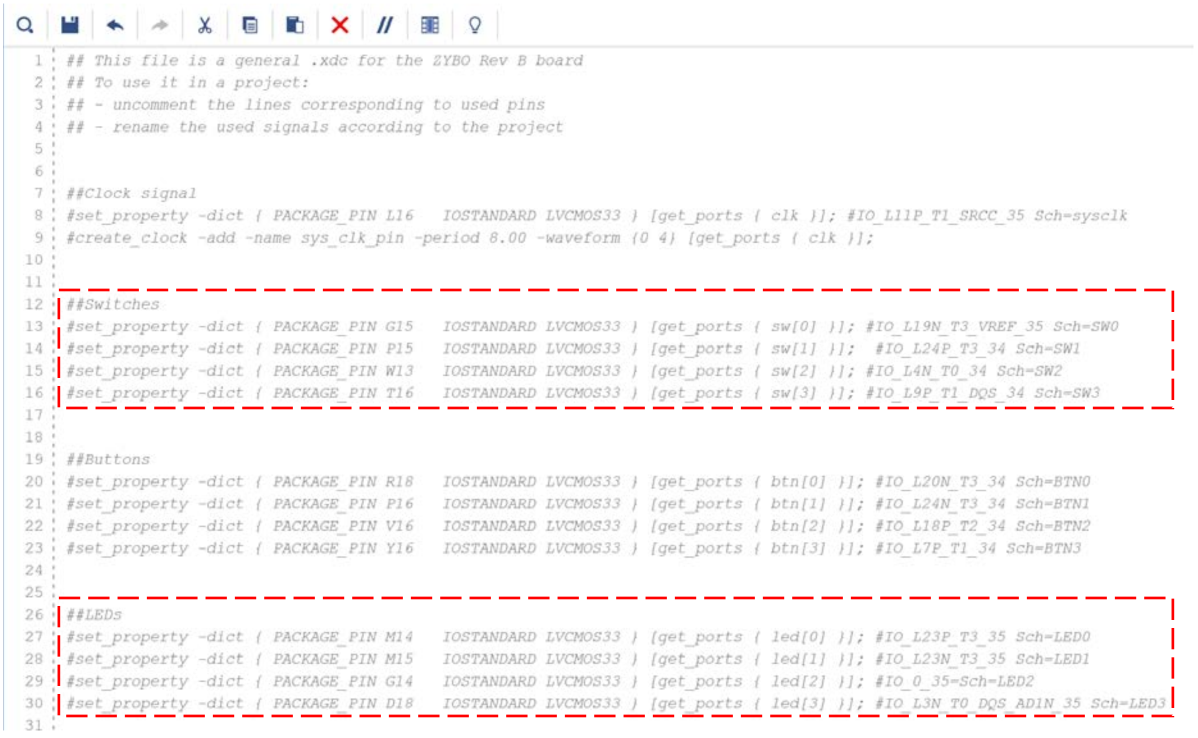

Project Manager - tutorial_1

Sources

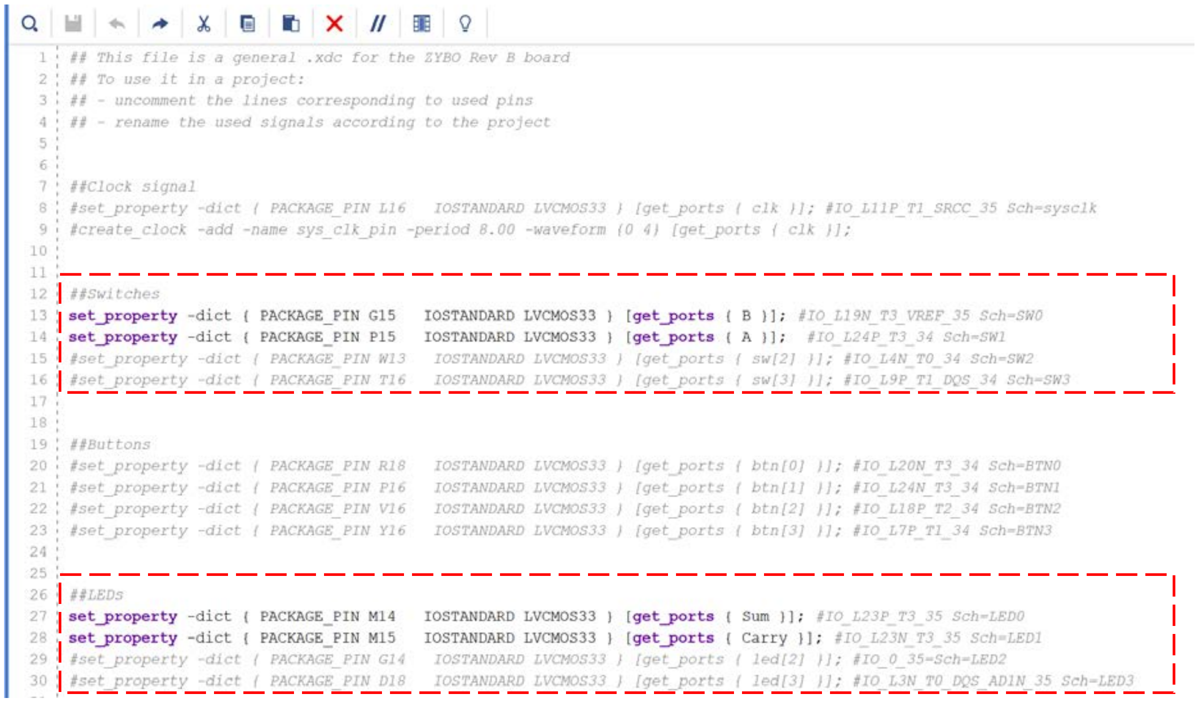
```
⊟ Design Sources (1)
    half_adder - Behavioral (half_adder.vhd)
⊟ Constraints (1)
  ⊟ constrs_1 (1)
      ZYBO_Master.xdc
⊞ Simulation Sources (1)
```

Step 2-7: We can now edit the constraint file to relate our design with the specific hardware.

a) Double click on the file to bring it up in the window on the right for editing.
b) Uncomment the lines for *sw[0]* and *sw[1]* by deleting the *hash symbols* and rename the ports *B* and *A* as shown.
c) Repeat the process with *led[0]* and *led[1]* as shown here with *Sum* and *Carry*.
d) *NOTE: Port Names are Case Sensitive.*



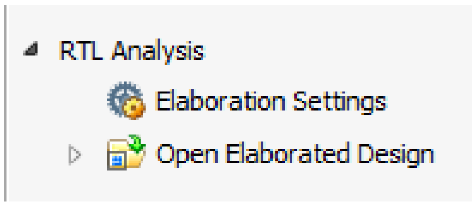
```
## This file is a general .xdc for the ZYBO Rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used signals according to the project


##Clock signal
#set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L11P_T1_SRCC_35 Sch=sysclk
#create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];


##Switches
#set_property -dict { PACKAGE_PIN G15   IOSTANDARD LVCMOS33 } [get_ports { sw[0] }]; #IO_L19N_T3_VREF_35 Sch=SW0
#set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { sw[1] }];  #IO_L24P_T3_34 Sch=SW1
#set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports { sw[2] }]; #IO_L4N_T0_34 Sch=SW2
#set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { sw[3] }]; #IO_L9P_T1_DQS_34 Sch=SW3


##Buttons
#set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { btn[0] }]; #IO_L20N_T3_34 Sch=BTN0
#set_property -dict { PACKAGE_PIN P16   IOSTANDARD LVCMOS33 } [get_ports { btn[1] }]; #IO_L24N_T3_34 Sch=BTN1
#set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { btn[2] }]; #IO_L18P_T2_34 Sch=BTN2
#set_property -dict { PACKAGE_PIN Y16   IOSTANDARD LVCMOS33 } [get_ports { btn[3] }]; #IO_L7P_T1_34 Sch=BTN3


##LEDs
#set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33 } [get_ports { led[0] }]; #IO_L23P_T3_35 Sch=LED0
#set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS33 } [get_ports { led[1] }]; #IO_L23N_T3_35 Sch=LED1
#set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { led[2] }]; #IO_0_35=Sch=LED2
#set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports { led[3] }]; #IO_L3N_T0_DQS_AD1N_35 Sch=LED3

```



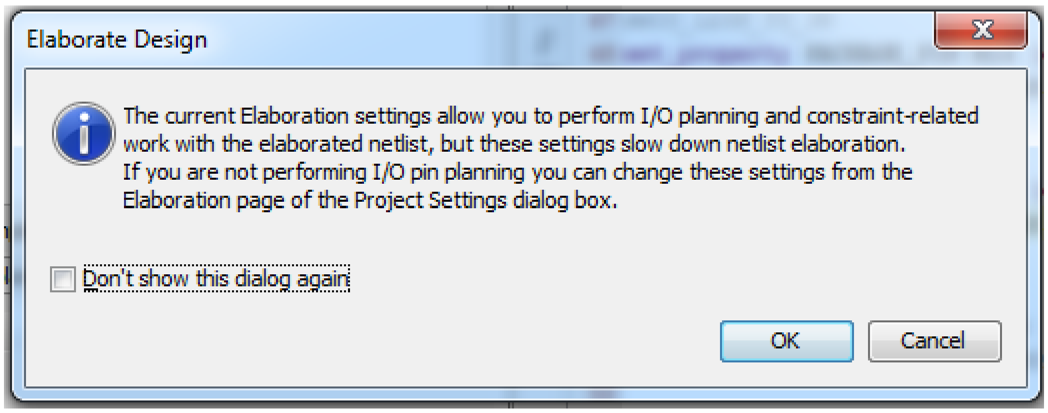
```
## This file is a general .xdc for the ZYBO Rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used signals according to the project


##Clock signal
#set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L11P_T1_SRCC_35 Sch=sysclk
#create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];


##Switches
set_property -dict { PACKAGE_PIN G15   IOSTANDARD LVCMOS33 } [get_ports { B }]; #IO_L19N_T3_VREF_35 Sch=SW0
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { A }];  #IO_L24P_T3_34 Sch=SW1
#set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports { sw[2] }]; #IO_L4N_T0_34 Sch=SW2
#set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { sw[3] }]; #IO_L9P_T1_DQS_34 Sch=SW3


##Buttons
#set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { btn[0] }]; #IO_L20N_T3_34 Sch=BTN0
#set_property -dict { PACKAGE_PIN P16   IOSTANDARD LVCMOS33 } [get_ports { btn[1] }]; #IO_L24N_T3_34 Sch=BTN1
#set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { btn[2] }]; #IO_L18P_T2_34 Sch=BTN2
#set_property -dict { PACKAGE_PIN Y16   IOSTANDARD LVCMOS33 } [get_ports { btn[3] }]; #IO_L7P_T1_34 Sch=BTN3


##LEDs
set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33 } [get_ports { Sum }]; #IO_L23P_T3_35 Sch=LED0
set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS33 } [get_ports { Carry }]; #IO_L23N_T3_35 Sch=LED1
#set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { led[2] }]; #IO_0_35=Sch=LED2
#set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports { led[3] }]; #IO_L3N_T0_DQS_AD1N_35 Sch=LED3
```

PART 3. ELABORATING AND SIMULATING THE DESIGN

Step 3-1: In the *Flow Navigator* under *RTL Analysis*, select *Open Elaborated Design*.



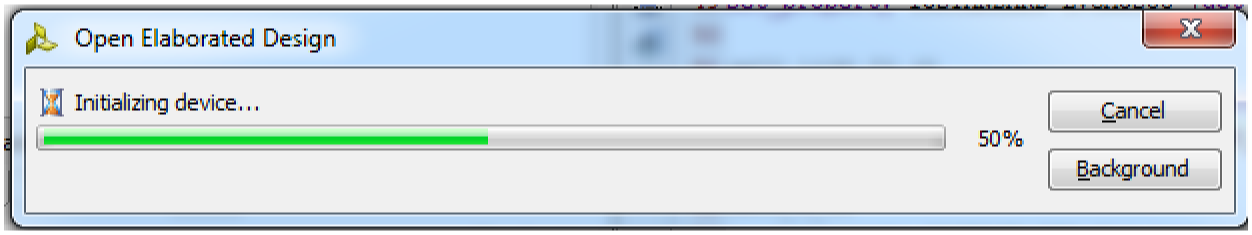
Step 3-2: Click *OK*.


Elaborate Design

The current Elaboration settings allow you to perform I/O planning and constraint-related work with the elaborated netlist, but these settings slow down netlist elaboration.
If you are not performing I/O pin planning you can change these settings from the Elaboration page of the Project Settings dialog box.

Don't show this dialog again

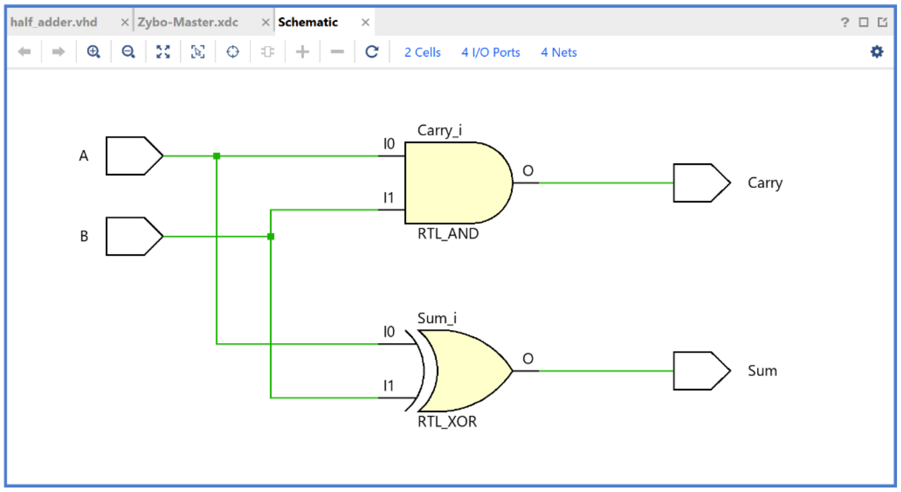
Step 3-3: This can take a few seconds to finish. This process is essentially Vivado's version of compiling your design.


Open Elaborated Design

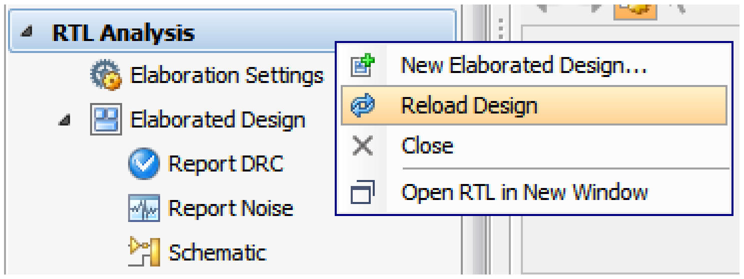
Initializing device...

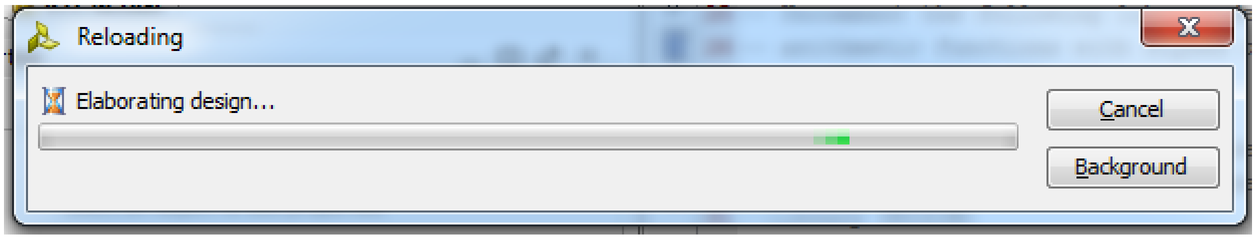
50%

Step 3-4: Now in the window on the right we see a schematic of our design, which matches the diagram from the beginning of this tutorial (NOTE: *In newer versions of Vivado, you might see the package or device view… to get to the schematic view, you can select* ***Schematic*** *under* ***RTL Analysis*** *in your* ***Flow Navigator***). This is put together based on our VHDL code and is a good way to verify if our logic is correct. Your schematic should look the same as this one.

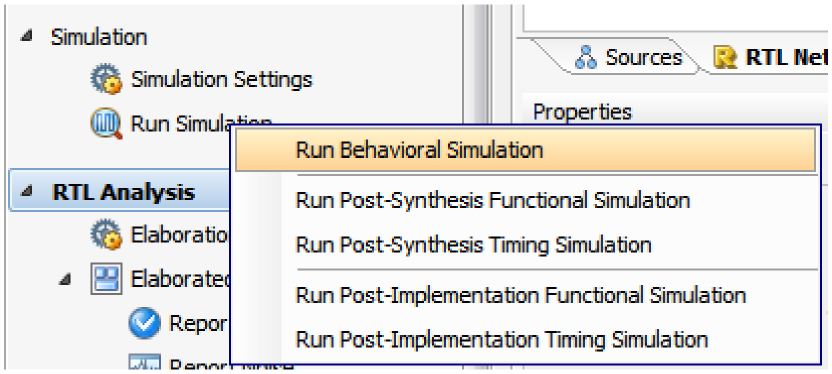


Step 3-5 (optional): If at this point, you were to edit any of your source files, you can then right click on *RTL Analysis* on *Elaborated design* and select *Reload Design* to 'recompile' your design.

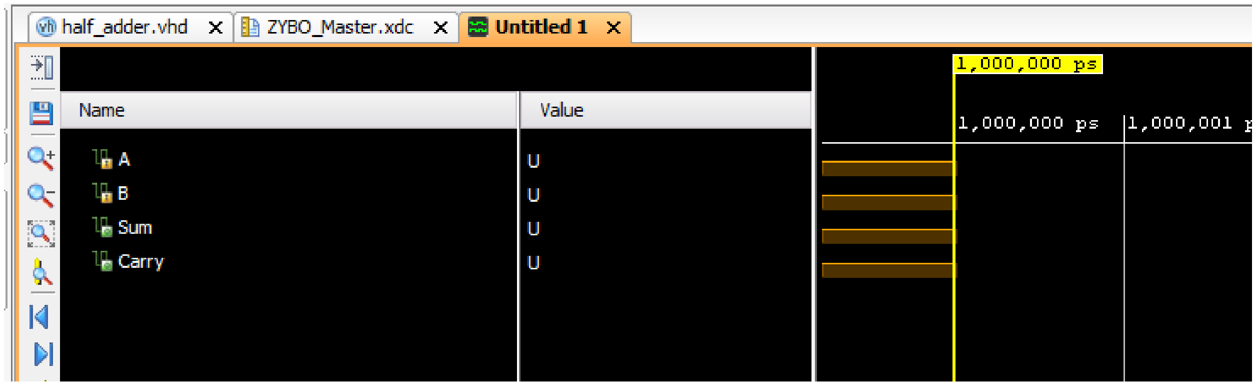




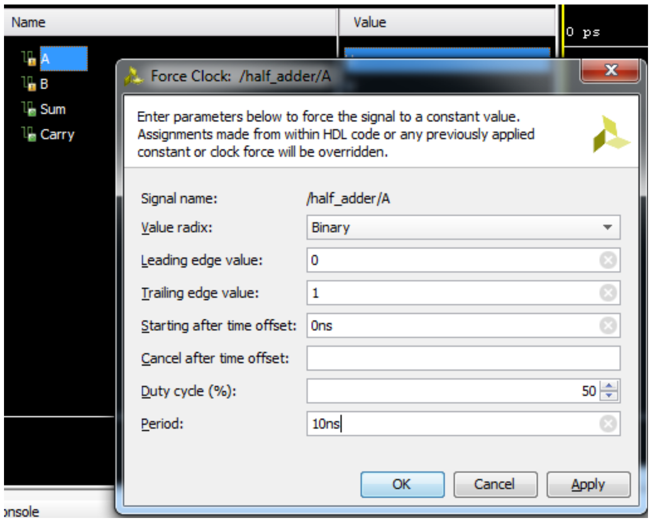
Step 3-6: In the *Flow Navigator* under *Simulation* select *Run Simulation*, then *Run Behavioral Simulation*.


Simulation
Simulation Settings
Run Simulation
RTL Analysis
Elaboratio
Elaborated
Repor
Sources
RTL Net
Properties
Run Behavioral Simulation
Run Post-Synthesis Functional Simulation
Run Post-Synthesis Timing Simulation
Run Post-Implementation Functional Simulation
Run Post-Implementation Timing Simulation

Step 3-7: Now in the window on the right we have the beginning of a timing diagram.



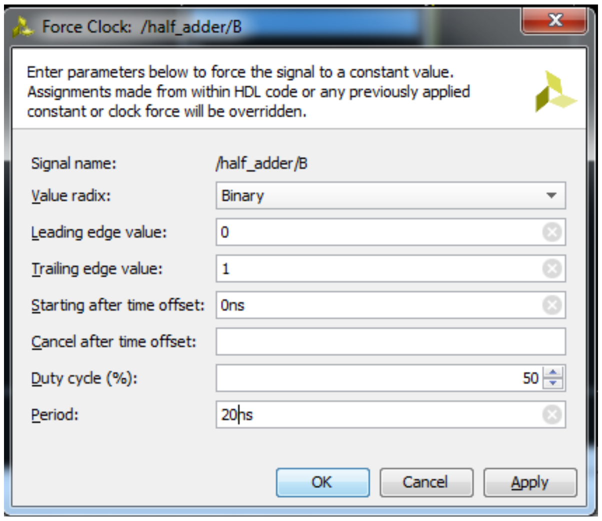
Step 3-8: Right click on port *A* under *Name* and select *Force Clock*, then enter the values shown below. Repeat the process with port *B* and enter the same values, except this time change the *Period* to *20ns*.



Force Clock: /half_adder/A

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:	/half_adder/A
Value radix:	Binary
Leading edge value:	0
Trailing edge value:	1
Starting after time offset:	0ns
Cancel after time offset:	
Duty cycle (%):	50
Period:	10ns

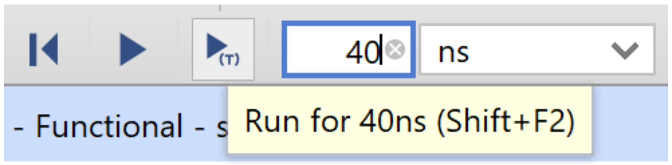


Force Clock: /half_adder/B

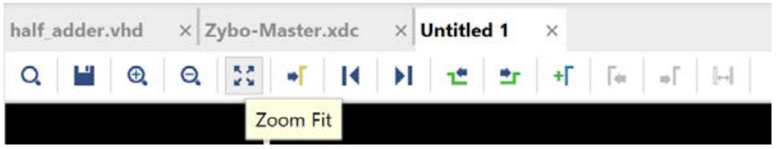
Enter parameters below to force the signal to a constant value. Assignments made from within HDL code or any previously applied constant or clock force will be overridden.

Signal name:	/half_adder/B
Value radix:	Binary
Leading edge value:	0
Trailing edge value:	1
Starting after time offset:	0ns
Cancel after time offset:	
Duty cycle (%):	50
Period:	20ns

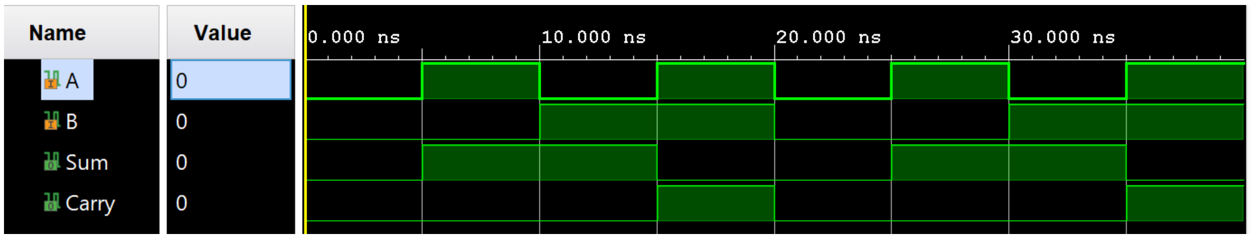
Step 3-9: Now in the *upper tool bar* first select the *Restart* button. Then change the time to *40 ns* and select the *Run* button with the small '*T*'. This runs the simulation for the set period of 40 ns.



Step 3-10: Finally click the *Zoom Fit* button in the simulation file window (NOTE: *it might be on the left hand side of the window in older versions of Vivado*).



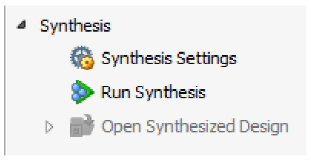
Step 3-11: Now we have a legible timing diagram that shows the simulated output of all possible input combination. Your simulation should have the same pattern as the one below.

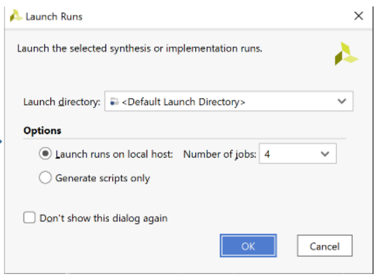


Note: I*f it does NOT show the values changing, repeat steps 3-8 through 3-10 to make sure that the force clock is set correctly. If you still do not see the changes, you can try selecting the Run button in step 3-9 rather than the Run for 40ns button. This will run the simulation indefinitely. You can also set an offset to the start of the simulation by opening the "force clock" again for A and B and set the value for "Starting after time offset" to 10ns.*

Part 4. Synthesizing, Implementing, Programming the ZYBO

Step 4-1: Now we can move onto programming the ZYBO board. In the *Flow Navigator* under *Synthesis* select *Run Synthesis*. If you get the "Laung Runs" Screen (newer versions of Vivado), keep the default values and select OK. This can take a few moments to complete (there is NOT a waiting notification, but the "Synthesis Complete" window will show up when it is done). Errors that show up here are usually syntax errors in the design source files (e.g., missing semi-colon).




Launch Runs

Launch the selected synthesis or implementation runs.

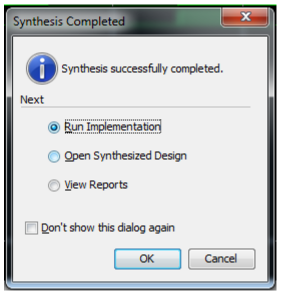
Launch directory: <Default Launch Directory>

Options

[✓ Launch runs on local host:] Number of jobs: 4

Generate scripts only

Don't show this dialog again


Synthesis Completed

Synthesis successfully completed.

Next

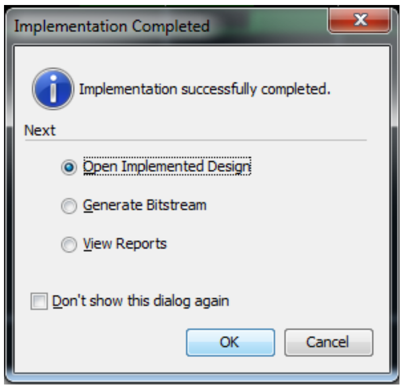
[✓ Run Implementation]

Open Synthesized Design

View Reports

Don't show this dialog again

Step 4-2: Once the *Synthesis* is complete move directly to the next step by selecting *Run Implementation*. Again, this can take a few moments to finish. At this stage, if the *Synthesis* finished without any problems, then any errors that come up during the *Implementation* process are generally related to the constraints file.


Implementation Completed

Implementation successfully completed.

Next

[✓ Open Implemented Design]

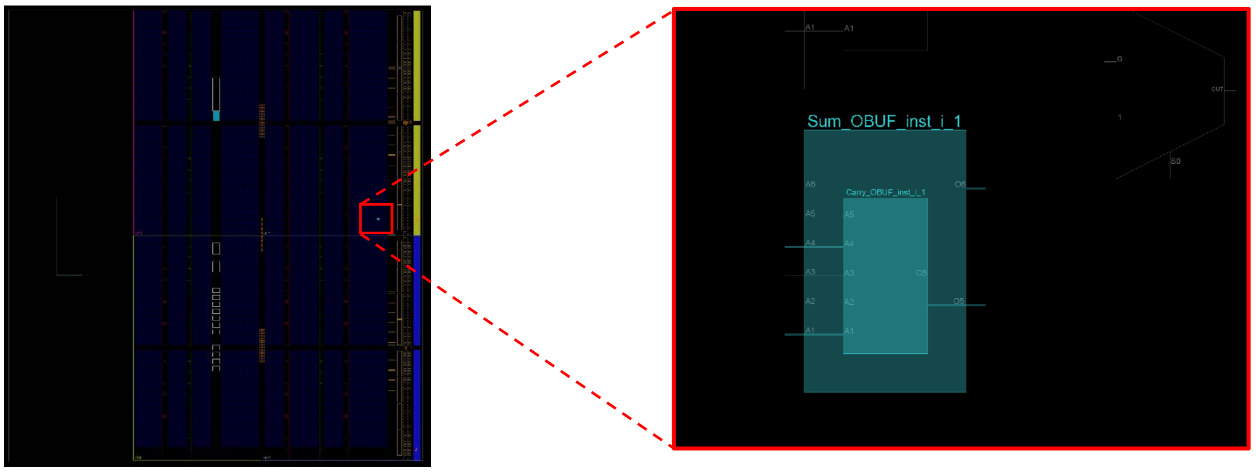
Generate Bitstream

View Reports

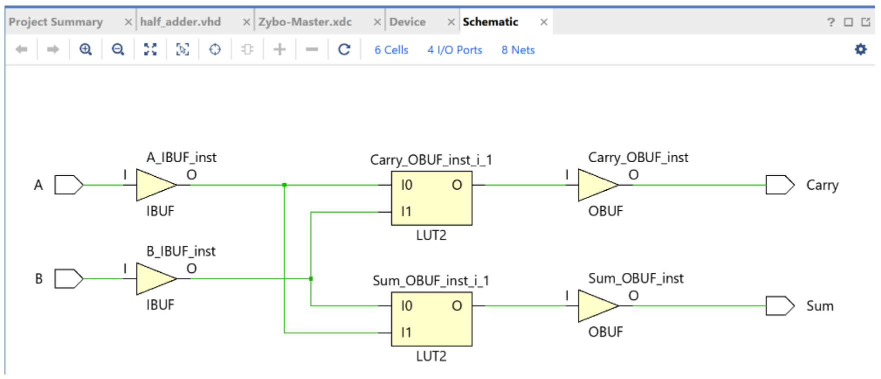
Don't show this dialog again

Step 4-3: Once the *Implementation* finishes, select *Open Implemented Design*.

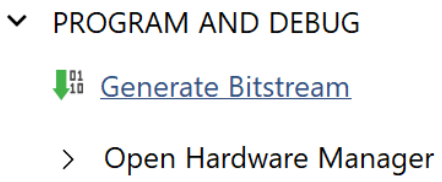
Step 4-4a: What is now shown in the window (seen below) is the internal layout of the *FPGA*. To zoom in on a part of the board, click and drag in the spot you want to magnify. Here you can look at details such as where the pins you are using are located. At this stage this is not very important, but later on when things like speed are critical, knowing where the pins you are using are in relation to others can be very helpful.



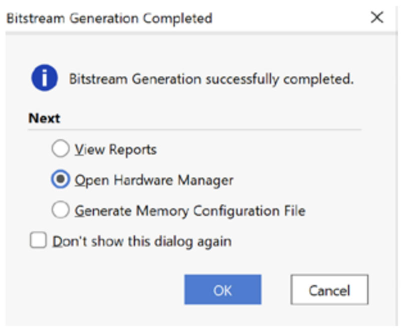
Step 4-4b: If you select "Schematic" under implementation, you can also find the implemented schematic (compared to the design schematic from the RTL analysis in step 3-4). You can find where these components are in the FPGA by zooming into the highlighted portions of the design view in 4-4a (e.g., the look-up table, or LUT, implementing Carry and Sum functionality is shown in the zoomed in image above).



Step 4-5: Now, in the *Flow Navigator* under *Program and Debug* select *Generate Bitstream*. This step can take a few moments as well.



Step 4-6: Once the *Bitstream* is generated, select *Open Hardware Manager*.


Bitstream Generation Completed

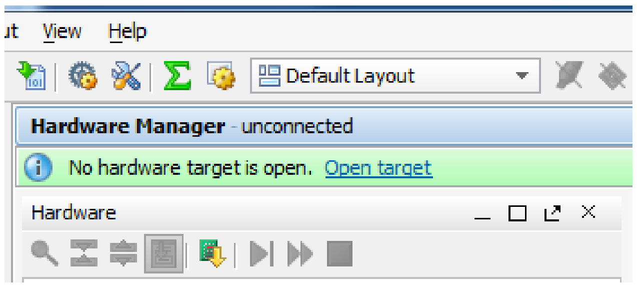
Bitstream Generation successfully completed.

Next

View Reports [✓ Open Hardware Manager] Generate Memory Configuration File

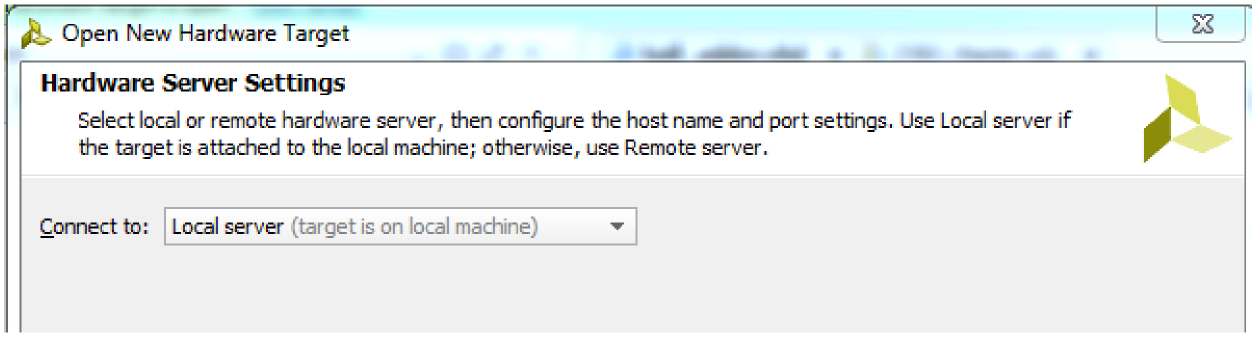
Don't show this dialog again

Step 4-7: Near the top of the window you will see *Open Target*. Select this, then *Open New Target*. This is how you direct Vivado where to send the *bitstream*.



No hardware target is open. Open target

Step 4-8: Click *Next*, then in the *Hardware Server Settings* window, make sure it's set to *Connect to Local Server*. Then click *Next*.

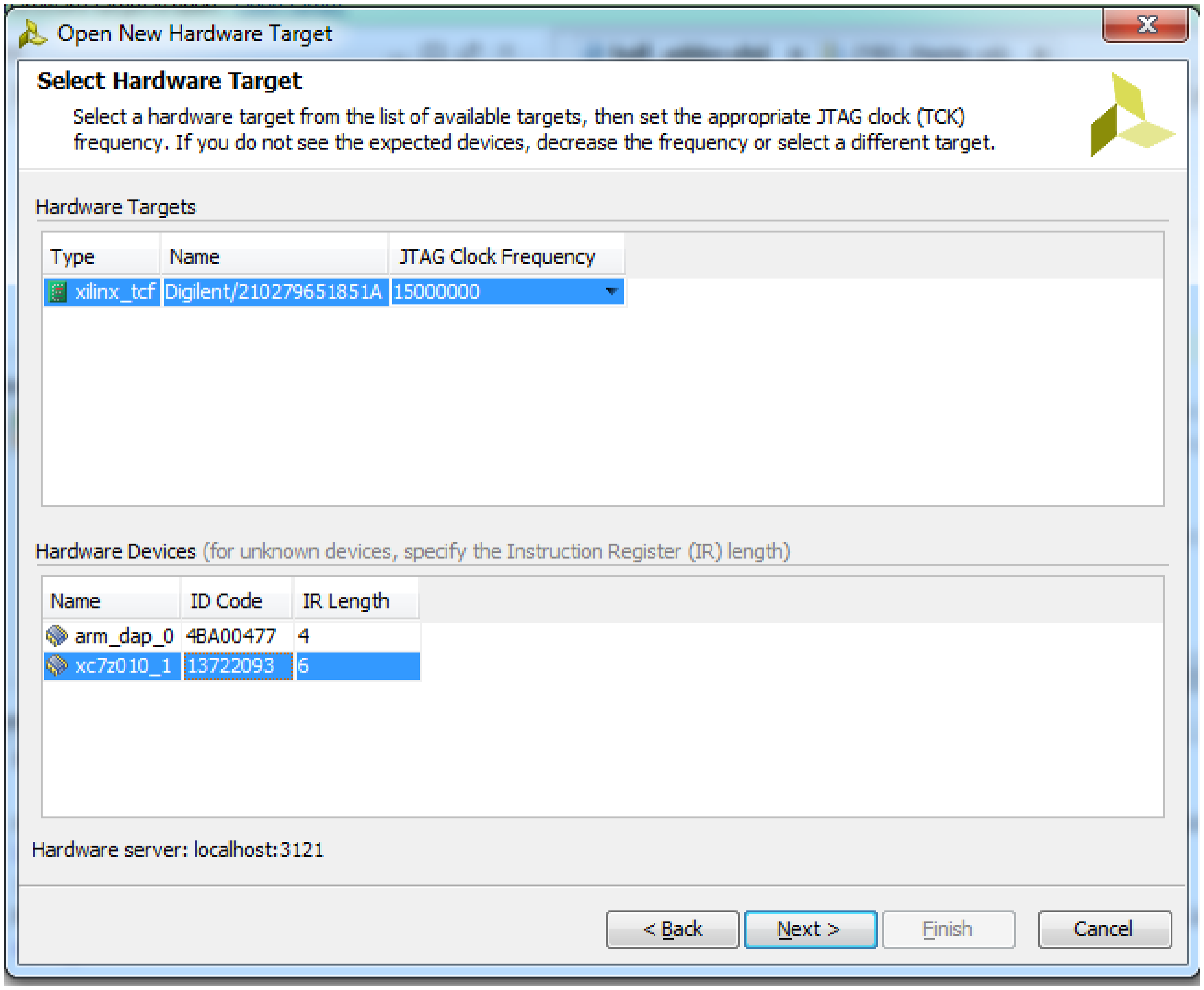

Open New Hardware Target

Hardware Server Settings

Select local or remote hardware server, then configure the host name and port settings. Use Local server if the target is attached to the local machine; otherwise, use Remote server.

Connect to: Local server (target is on local machine)

Step 4-9: In the *Select Hardware Target* window, select the **xc7z010_1** chip in *Hardware Devices*, then click *Next.*


Open New Hardware Target

Select Hardware Target

Select a hardware target from the list of available targets, then set the appropriate JTAG clock (TCK) frequency. If you do not see the expected devices, decrease the frequency or select a different target.

Hardware Targets

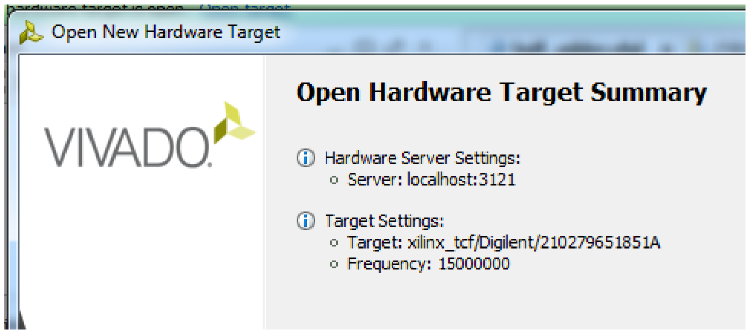
Type	Name	JTAG Clock Frequency
xilinx_tcf	Digilent/210279651851A	15000000

Hardware Devices (for unknown devices, specify the Instruction Register (IR) length)

Name	ID Code	IR Length
arm_dap_0	4BA00477	4
xc7z010_1	13722093	6

Hardware server: localhost:3121

Step 4-10: Verify that the details in the *Open Hardware Target Summary* match the ones below and click Finish. (Target will have a different number since this is indicating your specific device)

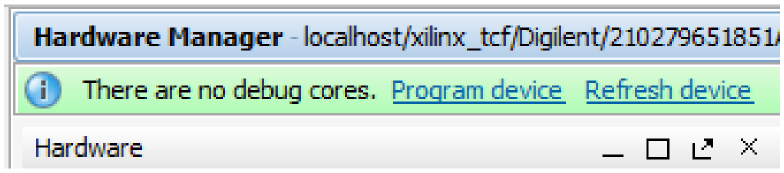


Open Hardware Target Summary

Hardware Server Settings:
- Server: localhost:3121

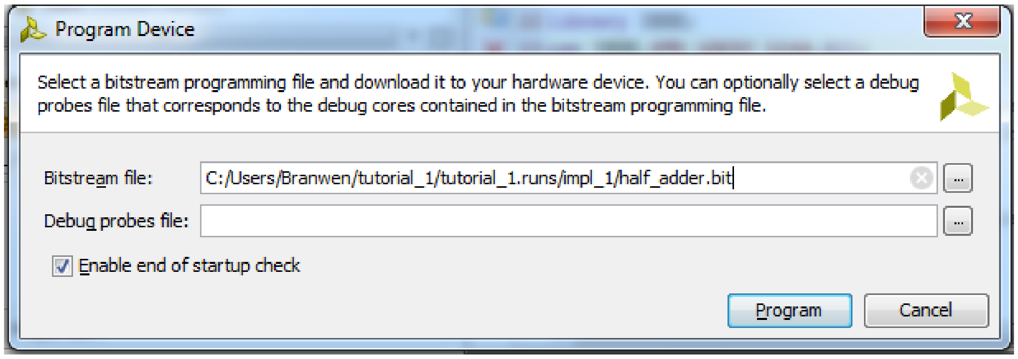
Target Settings:
- Target: xilinx_tcf/Digilent/210279651851A
- Frequency: 15000000

Step 4-11: Now click on *Program device*, then select the **xc7z010_1** chip (if given the option).


Hardware Manager - localhost/xilinx_tcf/Digilent/210279651851

There are no debug cores. Program device Refresh device

Step 4-12:. Finally click on *Program*.



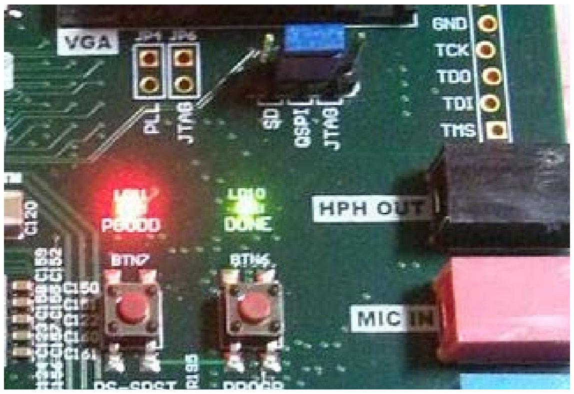
Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.

Bitstream file: C:/Users/Branwen/tutorial_1/tutorial_1.runs/impl_1/half_adder.bit

Debug probes file:

[✓] Enable end of startup check

Step 4-13: Once the Board is programmed, the green *DONE LED* will light up. Now you can test the Switches and LEDs to make sure they follow the design.



Part 5. Adding a Second Half-Adder

In this final part of the lab, you will follow the process above and add a second half-adder that uses two pushbuttons (BTN0 and BTN1) as input and LED2/LED3 as the output. This second half-adder should run alongside the half-adder that you have already implemented (i.e., you should be able to use both simultaneously).
Note: *Select PROJECT MANAGER in the Flow Navigator to return to the programming view*.

Step 5-1: Fill out the autogenerated heading at the top of your source file. This is a good practice that you should get used to doing. It is particularly important when working on larger projects and/or on projects with other engineers.

Company: UMass Boston
Engineer: <Your Name>
Project Name: ENGIN 341: Lab 0
(Also add a brief description and any comments you think are relevant)

Step 5-2: Add A2, B2, Sum2 and Carry2 to the I/O Port definitions in your source file:

```
entity half_adder is
    Port ( A, A2 : in STD_LOGIC;
           B, B2 : in STD_LOGIC;
           Sum, Sum2 : out STD_LOGIC;
           Carry, Carry2 : out STD_LOGIC);
end half_adder;
```

Step 5-3: Add the appropriate architecture statements that relate to the desired logic.

Step 5-4: Update the constraint file to include the following relations:

- btn[0] → A2
- btn[1] → B2
- led[2] → Sum2
- led[3] → Carry2

Step 5-5: Simulate and instantiate the board as described in Sections 3 and 4. Note that you should now be able to use BTN 0/1 as a half adder with output on led 2/3 IN ADDITION TO using switches 0/1 as a half adder with output on led 0/1.