First Start with Vivado

Introduction

This tutorial shows you how to install Vivado and set up the license. This tutorial uses **Xilinx Vivado 2016.1 WebPACK edition** on **Windows 7**.

Vivado Installation

The installation of Xilinx Vivado is really simple. Download the installation file from the Xilinx Website (http://www.xilinx.com/support/download.html) and run the downloaded file. In this tutorial we will use Xilinx Vivado WebPACK Edition which has a free (but limited) license. You can select it during the installation process. It is also interesting to note that you can select Vivado HL Design Edition and Vivado HL System Edition and then use a WebPACK license. But in this case, you will install elements that you won't be able to use. Vivado HL WebPACK Edition supports the Artix®-7 (7A35T - 7A200T), Kintex®-7 (7K70T, 7K160T) and Zynq®-7000 All Programmable SoC Devices (XC7Z7010 - XC7Z7030) devices.

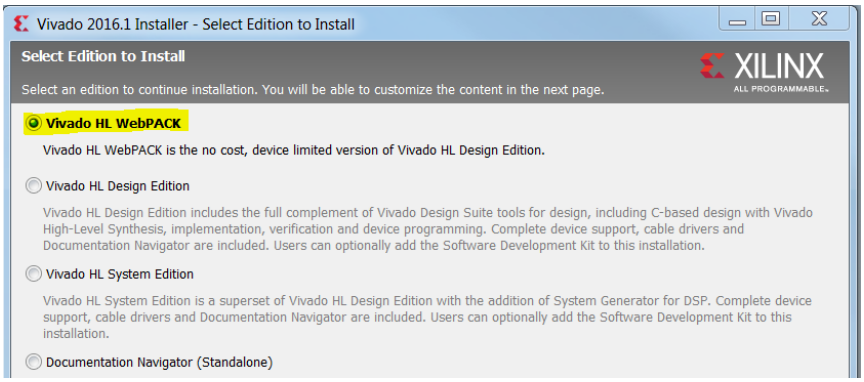

Figure 1 - Vivado HL WebPACK installation

In future tutorials, we will use the Xilinx Software Development Kit (SDK). To use it, we will need to select it during the installation process.

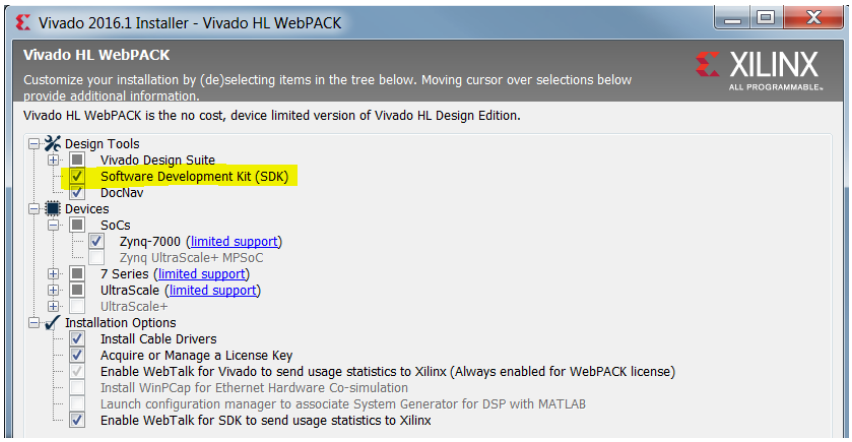

Figure 2 - Xilinx SDK

Obtain a license

Even if the WebPACK edition is free, we need to request a license from Xilinx. To request a license you will need to create an account on Xilinx website.

To obtain the Vivado WebPACK edition license, open Vivado License Manager (VLM) and select "*Get Free ISE WebPACK, ISE/Vivado IP or PetaLinux Licenses*" in the "*Obtain License*" tab. Then click on "*Connect Now*" to access your account.

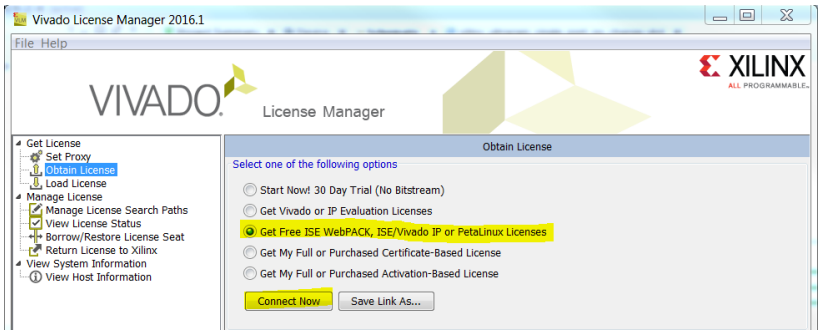

Figure 3 - Select Vivado WebPACK license in VLM

Once in your account, select the Vivado WebPACK license in the Certificate Based (CB) licenses part.

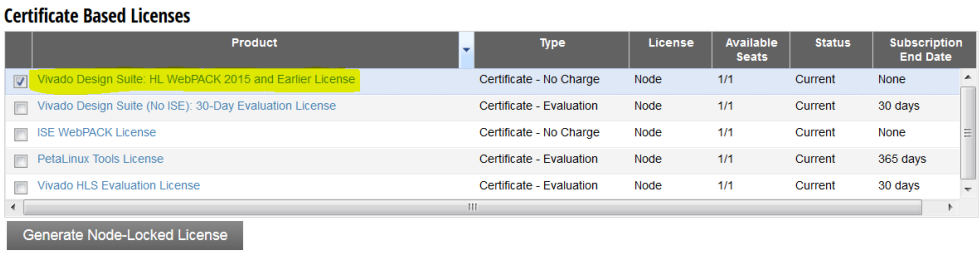

Figure 4 - Select the Vivado WebPACK license

Complete all the steps on the website. You should then receive an email with the license file attached. Save this .lic file on your computer. In VLM, enter the .lic file path in the "*XILINXD_LICENSE_FILE*" part in the "*Manage License Search Paths*" tab.

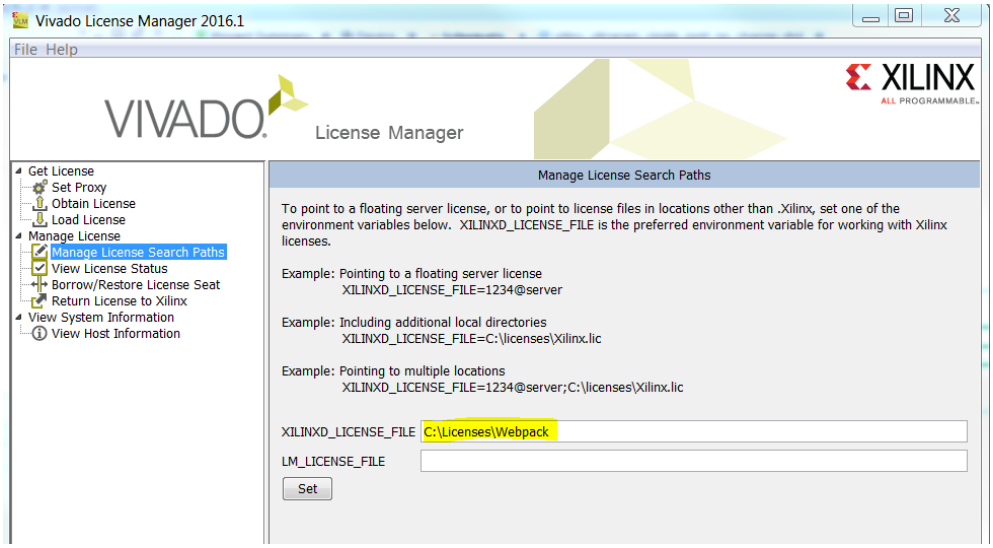

Figure 5 - Path to the .lic file

In the "*View License Status*" tab of VLM, you should see the WebPACK license in the "*Certificate Based Licenses*" part.

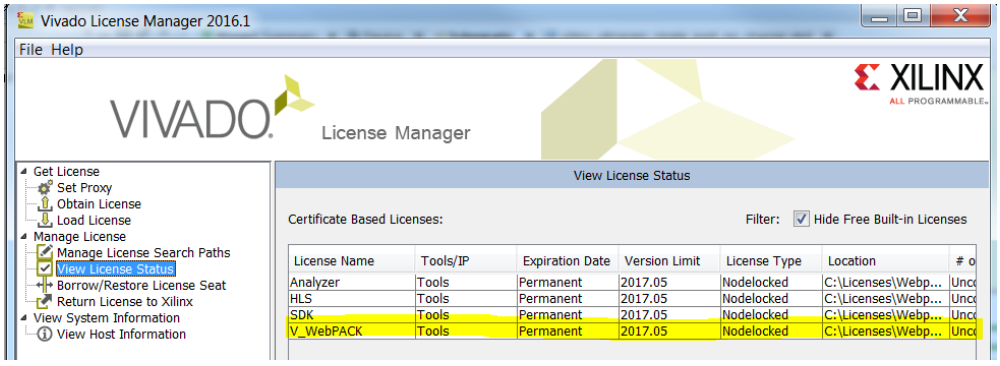

Figure 6 - View License Status

Launch Vivado

To launch Vivado you can simply click on the Desktop icon created during the installation. But I think that a better method is to run it from a command line.

Create a folder from which you will start Vivado (for example C:\2016.1). This will be you Current Working Directory (CWD).

In the command Prompt, go into the created folder, set up the environment running the C:\Xilinx\Vivado\2016.1\.setting64-Vivado.bat file. Then, to launch Vivado, enter the command "*Vivado*". This will open Xilinx Vivado in GUI mode.

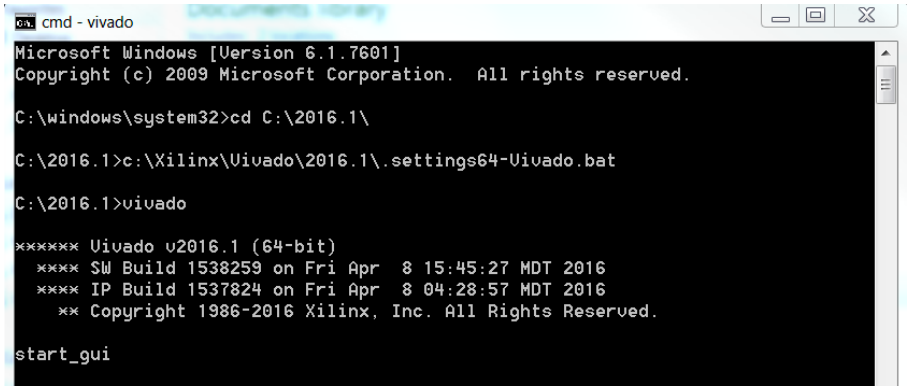

Figure 7 - Starting Vivado from the command prompt

Start an example project

For this tutorial, to discover Vivado we will use a Vivado Xilinx Example project. To open an example project, click on "*Open Example Project*" in the Vivado Home Page.

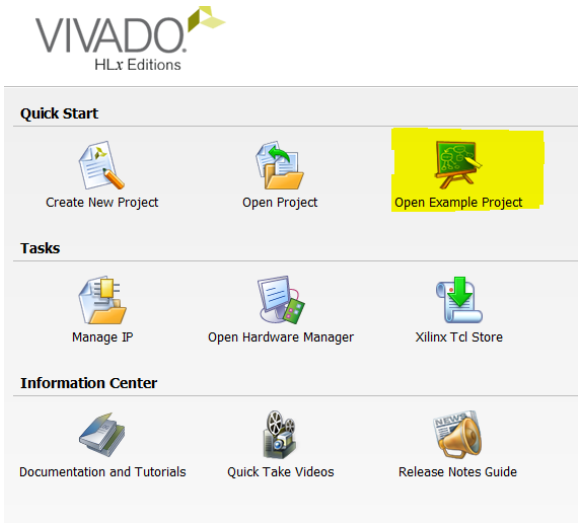

Figure 8 - Open Example Project

In this tutorial we will use the Wavegen example project.

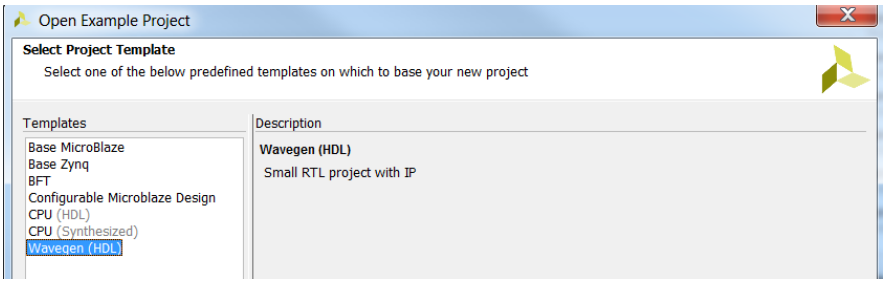

Figure 9 - Wavegen example project

On the next widows choose the project name and location. In the "*Default Part*" page, select the xc7k70tfbg676-1 device (which is included in the Vivado WebPACK license).

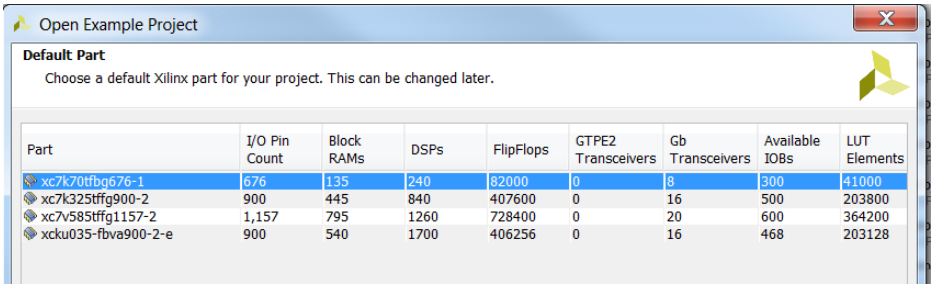

Figure 10 - Select the part

We will first analyse the different parts of the Vivado GUI. In the Figure 11, I have divided the Vivado GUI into 4 blocks.

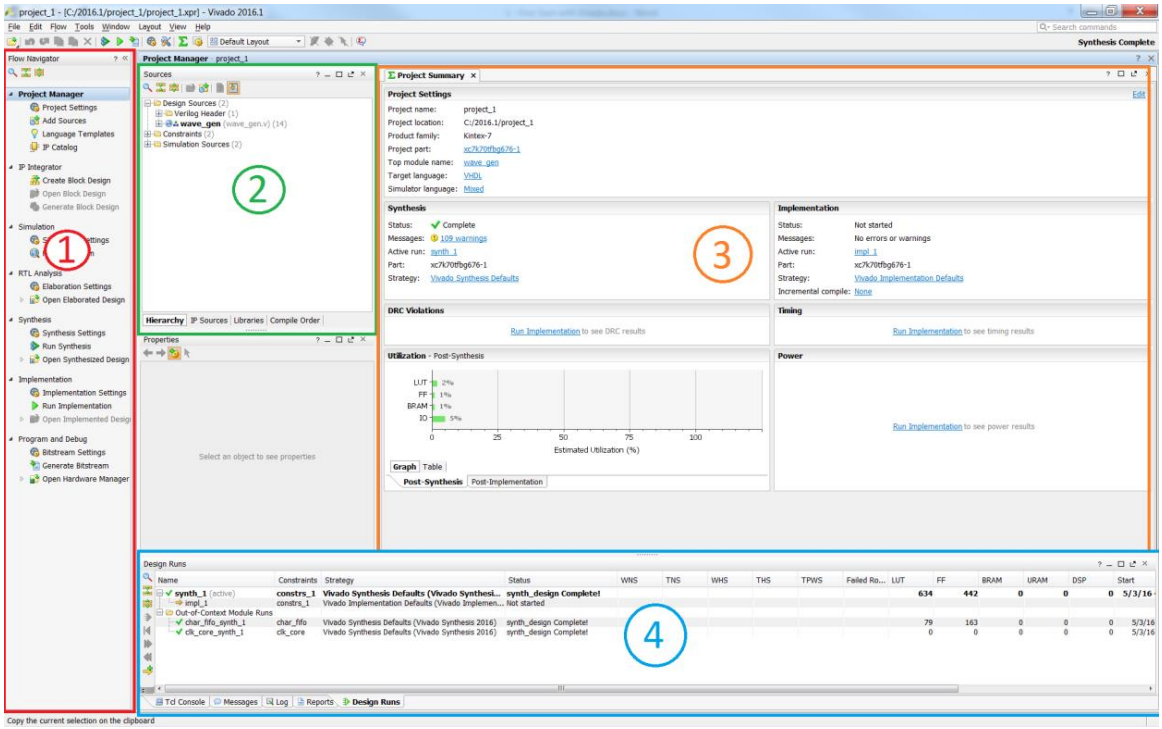

Figure 11 - Vivado GUI

The block 1 is the Flow Navigator. The Flow Navigator provides access to the main commands and tools to take a design from design entry to Bitstream Generation. The block 2 is the Data Windows Area. By default, this area of the Vivado IDE displays information related to design sources and data. The block 3 is the Workspace which displays windows with a graphical interface and those that require more screen space. The block 4 is the Results Windows Area.

Basic steps to generate a Bitstream

A Bitstream is a .bit file that is used to program a FPGA. To generate the Bitstream for this example project, you just have to click on "*Generate Bitstream*" in the flow navigator and click yes if a "*No Implementation Results Available*" window appears.

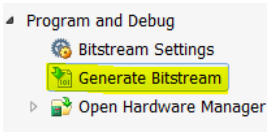

Figure 12 - Generate Bitstream

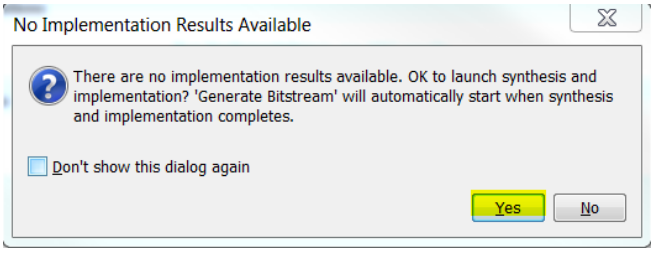

Figure 13 – No Implementation Results Available window

The tool will first run the synthesis process, then the implementation and will finally generate the Bitstream.

When the Bitstream is generated, a "*Bitstream Generation Completed*" window will appear. For this tutorial, you can simply close it. You can also close Vivado.

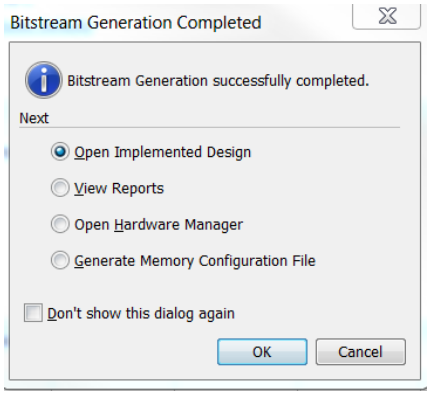

Figure 14 - Bitstream Generation Completed window

Vivado Journal and Log Files

Introduction

In the tutorial 1 (First Start with Vivado) we have used an example design to generate a Bitstream using Xilinx Vivado 2016.1. During this process from Project Creation to Bitstream Generation, .jou and .log files have been created by Xilinx Vivado. In this tutorial we will analyse these files.

vivado.jou and vivado.log files

If we look at the folder from which we have launched Vivado, two files have been created: vivado.jou and vivado.log.

The vivado.jou file is a journal file which contains tcl commands. Each GUI (Graphical User Interface) operation we have made results in one (or more) tcl command. The Figure 1 shows the vivado.jou file obtained from the previous tutorial. The line 12 contains the first tcl command: start_gui. This tcl command has been executed when we have enter the command "*vivado*" in the command prompt. By default, Vivado is launched in GUI mode, so the GUI is started. The lines 13 to 17 correspond to the project creation (we can see the part we have selected and the example project). Finally, the lines 18 and 19 correspond to the "Generate Bitstream" operation.

vivado.jou

```
1  #-----------------------------------------------------------
2  # Vivado v2016.1 (64-bit)
3  # SW Build 1538259 on Fri Apr  8 15:45:27 MDT 2016
4  # IP Build 1537824 on Fri Apr  8 04:28:57 MDT 2016
5  # Start of session at: Wed May 04 17:08:35 2016
6  # Process ID: 2548
7  # Current directory: C:/2016.1
8  # Command line: vivado.exe
9  # Log file: C:/2016.1/vivado.log
10 # Journal file: C:/2016.1\vivado.jou
11 #-----------------------------------------------------------
12 start_gui
13 create_project project_1 C:/2016.1/project_1 -part xc7k70tfbg676-1
14 set_property target_language VHDL [current_project]
15 instantiate_example_design -template xilinx.com:design:wave_gen:1.0
16 update_compile_order -fileset sources_1
17 update_compile_order -fileset sim_1
18 launch_runs impl_1 -to_step write_bitstream -jobs 2
19 wait_on_run impl_1
```

Figure 1 - vivado.jou file

The vivado.log file also contains the tcl commands captured from the GUI operations but also contains all the messages returned by Vivado.

If you want to use the desktop icon to run Vivado and define where the .log and .jou files will be created, right click on the icon and click on "*properties*". Enter the path from where you want to start Vivado in the "*start in*" field.

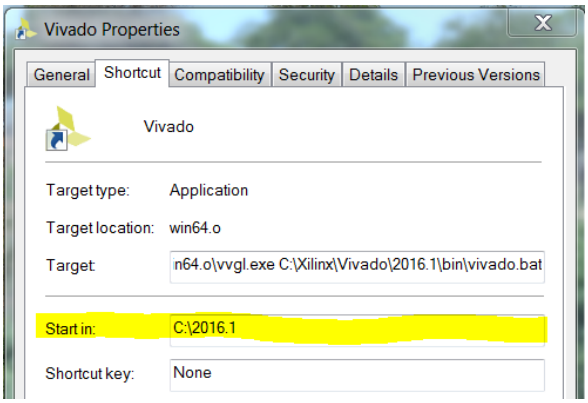

Figure 2 - Desktop icon properties

Finding Xilinx documentation and using DocNav

Introduction

In this tutorial we will see how to get information about Xilinx products using the Xilinx Documentation. There are three way to get the Xilinx documentation: using the Xilinx Website (www.xilinx.com) and using the Xilinx DocNav tool.

Find the documentation using the Xilinx website

If you know the document name or number you are looking for, you can go on the Xilinx Website main page and type it in the search bar. For example, if you want the datasheet of the 7 series (DS180) you can enter “DS180” in the search bar. A page with all the corresponding document will be displayed.

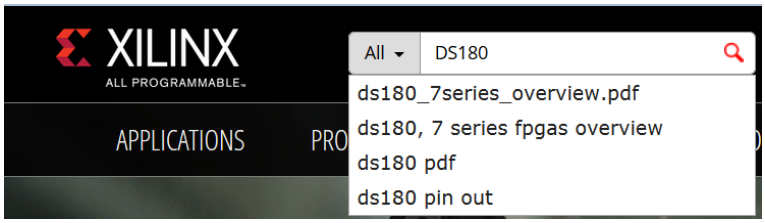

Figure 1 - Document search using the Xilinx Search Bar

If you don’t know the document name but you know the type of document you are looking for, you can go to the Xilinx Support page (http://www.xilinx.com/support.html#documentation) and use the categories to find your document.

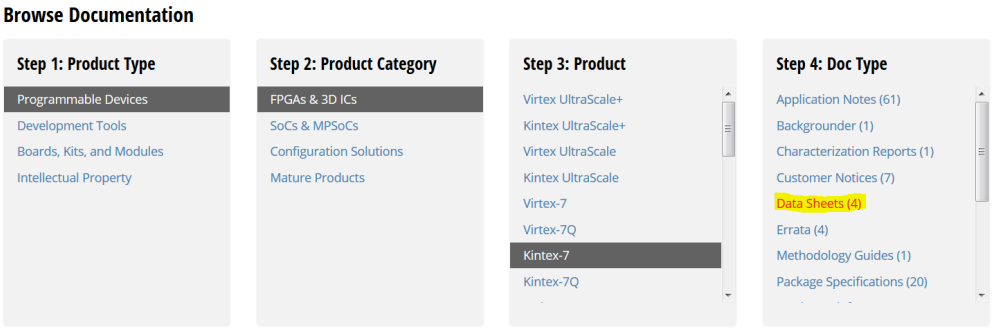

Figure 2 – Xilinx Document Search using Browse Documentation

If you are looking for a content but you don’t know in which type of document you can find it, you can enter keywords in the Xilinx search bar and all the documents containing the keywords will be displayed.

Getting a different version of the document:

Sometimes, you will want to get a different version of the document. For example, if you are using Vivado 2015.2 and the current version is 2016.1 you will need to get the correct version of the User

Guide (UG910) because there are several changes between the two tool releases. To get a different version, click “See All Versions” in the result page.

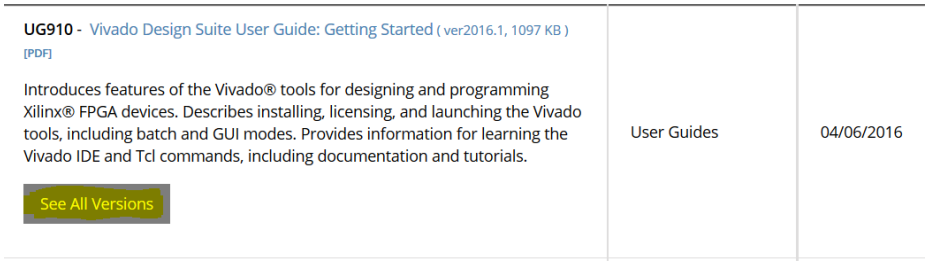

Figure 3 - Getting a different version of a Xilinx document

Find the documentation using Xilinx DocNav

The Xilinx Documentation Navigator is a free tool you can install on your computer to access the Xilinx Documentation.

In DocNav, you can use the Search Bar to find for a document name or number.

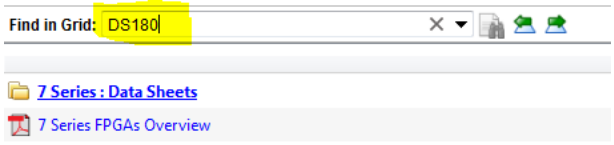

Figure 4 - Xilinx DocNav Search Bar

In the basic settings, the Search Bar only look for the keywords in the document name or number. If you want to search for the keywords inside the document, select “Search Docs”.

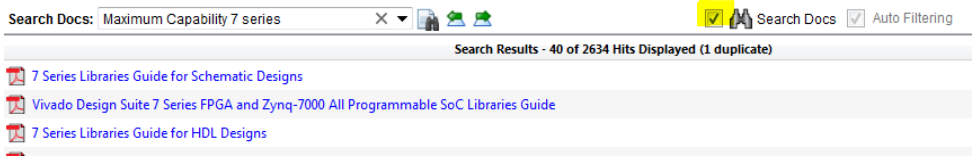

Figure 5 - DocNav Search - Search Docs option

If you want to search for a particular version of a document, you can use the version selector.

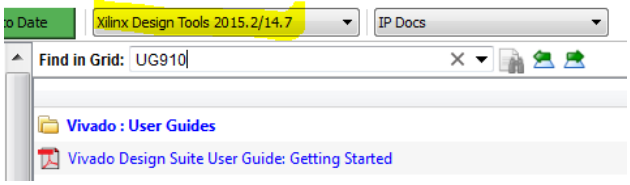

Figure 6 - Version Selector

Simple RTL (VHDL) project

Introduction

In this tutorial we will create a simple VHDL project using the text editor of Xilinx Vivado 2016.1.

The design will have 4 1-bit inputs and 1 1-bit output. The 2 first inputs, which we will name A and B, will be connected to an AND gate and the two last inputs, C and D, will be connected to an OR gate. The two gates outputs will enter in another AND gate. The project is schematized in the Figure 1.

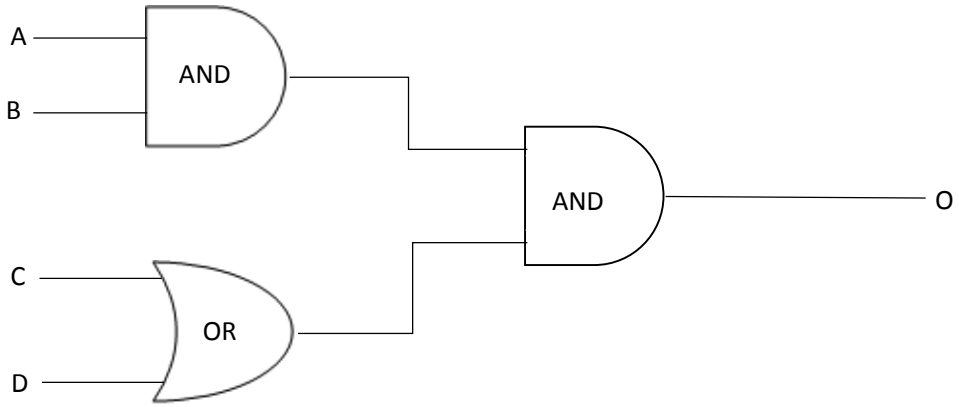

Figure 1 - Project Schematic

Create a VHDL project

Start Xilinx Vivado 2016.1 in a GUI mode using the command line as explained in the post 1 or using the desktop icon. On the welcome page click on "*Create New Project*". Enter a project name and location.

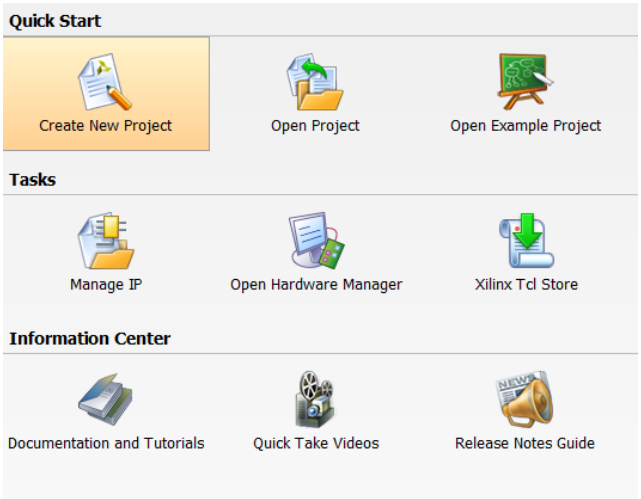

Figure 2 - Create a new project

In the "*Project Type"* window, select "*RTL project*" and check the "*Do not specify sources at this time*".

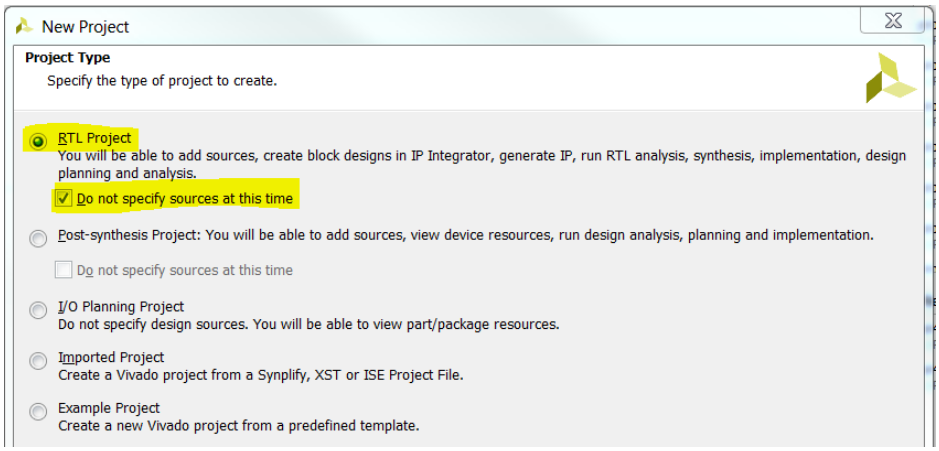

Figure 3 - RTL Project Selection

Select the Kintex-7 xc7k70tfbg676-1 as default part.

Create the VHDL file

In the Flow Navigator, click on "*Add Sources*" (alternatively you can click on "*File* > *Add Sources*").

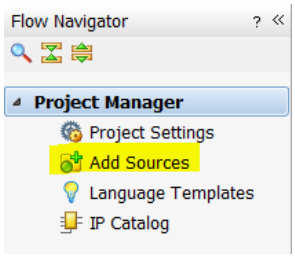

Figure 4 - Add Sources

In the first page of the "*Add Sources*" window, select "*Add or create design sources*".

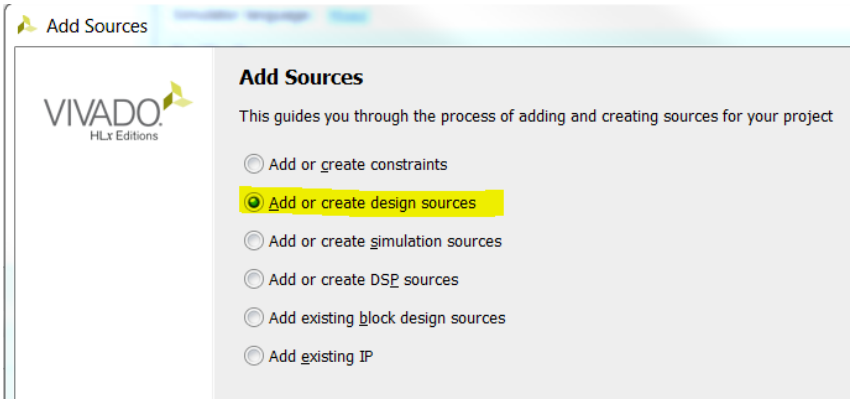

Figure 5 - Add or create design sources

In the "*Add or Create Design Sources*" page, click on the + button on the left and click on "*Create File*" or click on the "*Create File*" button on the bottom.

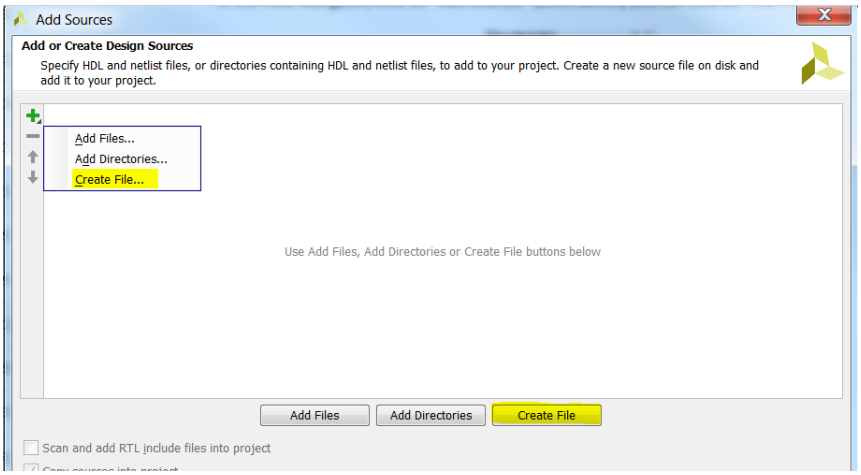

Figure 6 - Create Design Source File

In the "*Create Source File*" window, select "*VHDL*" for the "*File type*" field and add a file name. Click "*OK*" and click "*Finish*" on the "*Add Sources*" window.

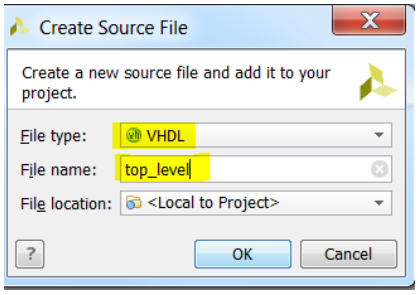

Figure 7 - Create a new VHDL file

A "*Define Module*" window will appear. It allows you to define the ports of your new module in order to have a pre-filed VHDL file. Fill it as below.

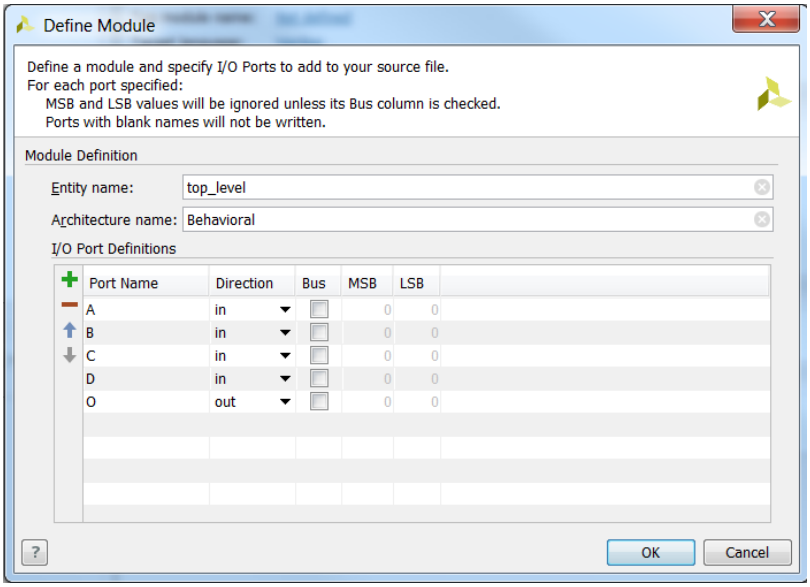

Figure 8 - Define Module window

Modify the VHDL file

In the Data Windows Area, in the "*Sources*" window, we can see the VHDL file in the "*Design Sources*". We can double click on this file to open the text editor.

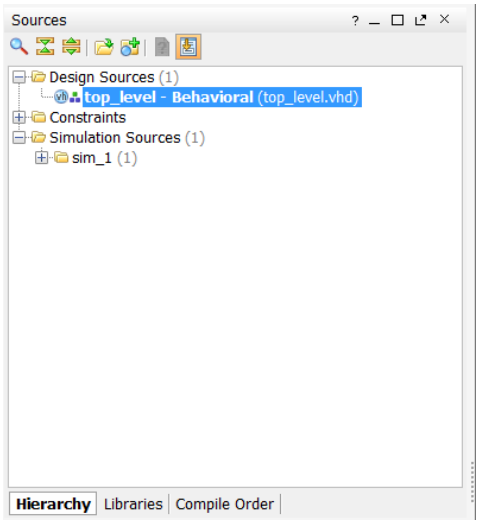

Figure 9 - Sources window

We can see that Vivado has already created the structure of the VHDL file.

```
34 entity top_level is
35     Port ( A : in STD_LOGIC;
36            B : in STD_LOGIC;
37            C : in STD_LOGIC;
38            D : in STD_LOGIC;
39            O : out STD_LOGIC);
40 end top_level;
41
42 architecture Behavioral of top_level is
43
44 begin
45
46
47 end Behavioral;
```

Figure 10 - VHDL structure created by Vivado

In the text editor we can code our block.

```
42 architecture Behavioral of top_level is
43
44 signal AND_1_out_sig  :  std_logic;
45 signal OR_out_sig    :  std_logic;
46
47 begin
48
49 AND_1_out_sig   <= A AND B;
50 OR_out_sig      <= C OR D;
51
52 O <= AND_1_out_sig AND OR_out_sig;
53
54
55 end Behavioral;
```

Figure 11 - VHDL code of the block

When you save your file, if there are syntax errors, the file will be displays under "*Syntax Error Files*" in the "*Sources*" window and the errors will be displayed in the messages console.

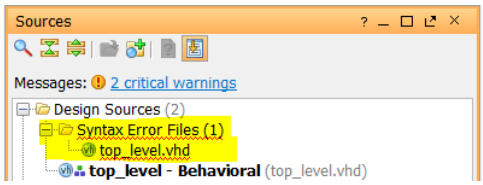

Figure 12 - Syntax Error Files

View the Schematic of the RTL (VHDL) block

In the "*Flow Navigator*", in "*RTL Analysis*", if we expend "*Elaborated Design*" we can click on "*Schematic*".

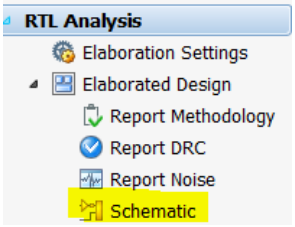

Figure 13 - Open the Elaborated Design Schematic

This will open the schematic of our RTL bloc. From this schematic we can check that we have correctly coded our bloc.

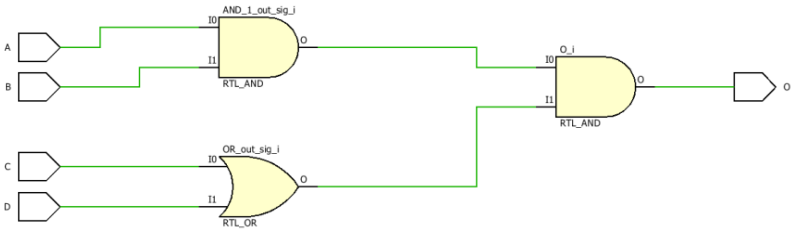

Figure 14 - Elaborated Design Schematic

```
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 top_level is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           C : in STD_LOGIC;
           D : in STD_LOGIC;
           O : out STD_LOGIC);
end top_level;

architecture Behavioral of top_level is

signal AND_1_out_sig    : std_logic;
signal OR_out_sig       : std_logic;

begin

AND_1_out_sig   <= A AND B;
OR_out_sig      <= C OR D;

O   <= AND_1_out_sig AND OR_out_sig;

end Behavioral;
```

Behavioral Simulation with the Vivado Simulator (XSIM)

Introduction

In the previous tutorial (4 - Simple RTL (VHDL) project) we have created a simple RTL project. In this tutorial we will use the Vivado Simulator (XSIM) to validate the behavior of our design.

Run the simulation

To launch the Vivado Simulator for behavioral simulation, click on "*Run Simulation*" under "*Simulation*" in the "*Flow Navigator*" and then click "*Run Behavioral Simulation*". This will open a waveform window.

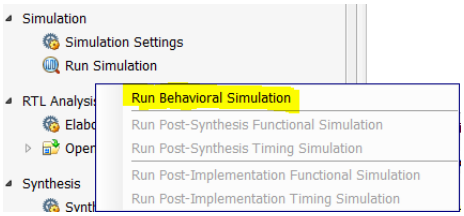

Figure 1 - Run Behavioral Simulation

In the waveform window, on the left, we can see the 4 inputs and the output of our design and the intermediate signals we have created for the output of the OR gate and of the first AND gate. We can see that the value of all the signal is 'U' which means Uninitialized. This is because we haven't given a value to our input signals.

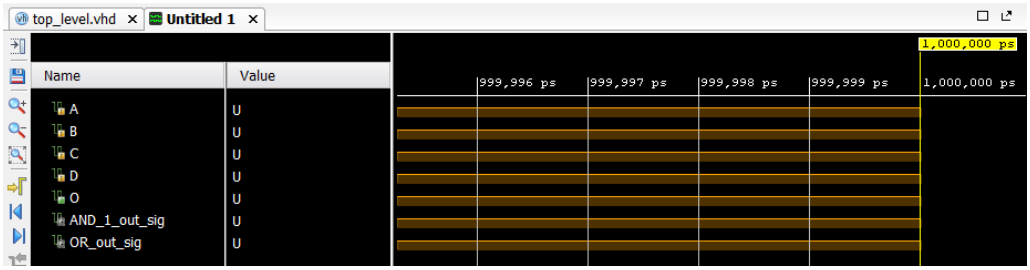

Figure 2 - Waveform window in Vivado

We will give a value to our input signals by forcing their value in the simulation. First, click on "*Run > Restart the Simulation*" to restart the simulation.

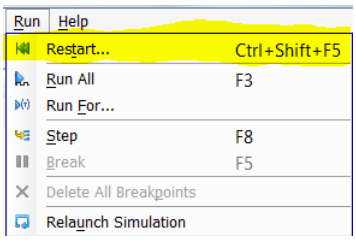

Figure 3 - Restart the simulation - Vivado Simulator

Then we will give a value to our inputs. To test all the possibilities of our system, we will force these signals to have a clock signal. Right click on the signal A and click on "*Force Clock".*

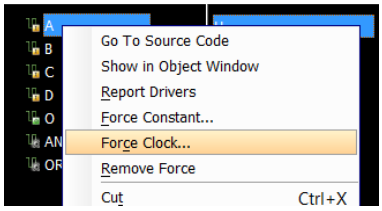

Figure 4 - Force Clock

Set the "*Value radix"* to "*Binary*", the "*Leading edge value"* to '0', the "*Trailing edge value*" to '1' and the period to "*10ns*".

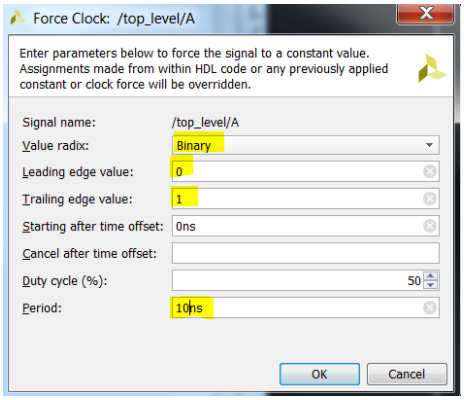

Figure 5 - Force clock on input A

Do the same for the inputs B, C and D but with period value respectively "*20ns*", "*40ns*" and "*80ns*".

Then run the simulation by clicking on "*Run > Run For… "*. Set the simulation time to 160ns.

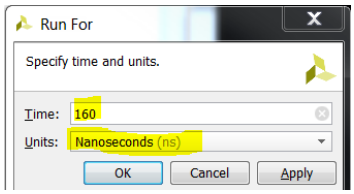

Figure 6 - Run the simulation for 160ns

On the waveform, we can see that O is high in 3 cases:

- When A, B and C are high and D is low

- When A, B and D are high and C is low
- When A, B, C and D are high

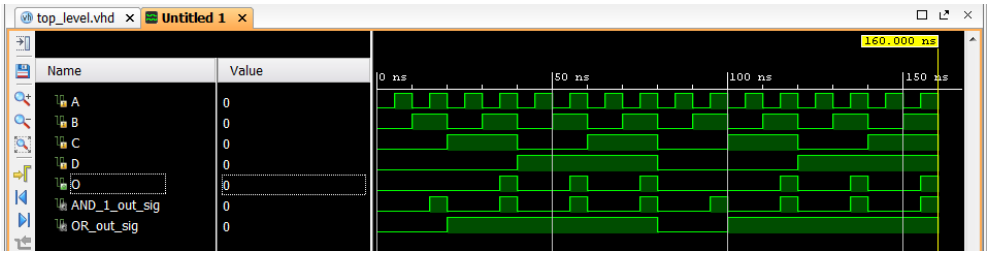

Figure 7 - Vivado Simulator Waveform

This is the expected behavior of our design. Close the simulation window.

Create a test bench (VHDL)

Another way to give a value to our input signals is to create a test bench to simulate our design.

What is a test bench? A test bench is a file written as an HDL file (VHDL, Verilog…) which generally provides a stimuli (inputs, clocks) to a Unit Under Test (UUT).

To create the test bench file in Vivado, click on "*Add Sources*" in the "*Flow Navigator*" and select "*Add or create simulation sources*".

Figure 8 - Add or create simulation sources

Then, click on "*Create File*", select VHDL, enter "*tb_top_level*" as file name, make sure the file type is VHDL and click on finish. Generally, test benches don't have inputs or outputs, so if the "*Define Module*" window appears, remove the first line by selecting it and clicking the "-" button and click on "*OK*".

We can see our created file under the "*Simulation Sources*" in the "*Sources*" window.

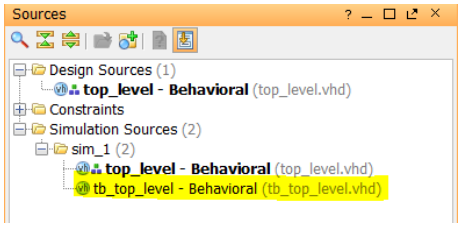

Figure 9 - Simulation Sources in the Sources window

Modify the test bench

Double click on the file in the "*Sources*" window to open it in the text editor. Vivado has already created a template for our file.

1. Declare the top_level bloc

First declare the top_level bloc in the declarative part (between the architecture and the begin keywords):

```
38  architecture Behavioral of tb_top_level is
39
40  component top_level is
41      Port (
42          A : in STD_LOGIC;
43          B : in STD_LOGIC;
44          C : in STD_LOGIC;
45          D : in STD_LOGIC;
46          O : out STD_LOGIC
47      );
48  end component top_level;
49
50  begin
```

Figure 10 - top_level bloc declaration

2. Declare the signals

Then declare all the signals we will use. We will connect all the top_level inputs and outputs so declare these signals in the declarative part and set the initial inputs values to '0':

```
49
50      signal A : STD_LOGIC := '0';
51      signal B : STD_LOGIC := '0';
52      signal C : STD_LOGIC := '0';
53      signal D : STD_LOGIC := '0';
54      signal O : STD_LOGIC;
55
56  begin
```

Figure 11 - Signals declaration

3. Instantiate the Unit Under Test (UUT)

Then instantiate the top_level bloc, that we will call UUT (for Unit Under Test), in the statement part (after the begin keyword) by connecting the signals:

```
58      UUT : top_level
59      Port map(
60          A => A,
61          B => B,
62          C => C,
63          D => D,
64          O => O
65      );
```

Figure 12 - Instantiate the UUT

4. Generate the stimuli

Then assign value to the input signals. A will be toggled every 10 ns, B every 20 ns, C every 40 ns and D every 80 ns. All the possible combinations will be tested after 160ns (simulation time).

```
66
67      A <= not A after 10 ns;
68      B <= not B after 20 ns;
69      C <= not C after 40 ns;
70      D <= not D after 80 ns;
71
72  end Behavioral;
```

Figure 13 - Stimuli generation

Run the behavioral simulation with test bench

Run the behavioral simulation by clicking "*Run Simulation > Run Behavioral Simulation*" under "*Simulation"* in the "*Flow Navigator*".

We can see the same waveform as in the first part of this tutorial.

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

entity tb_top_level is
--  Port ( );
end tb_top_level;

architecture Behavioral of tb_top_level is

component top_level is
    Port (
        A : in STD_LOGIC;
        B : in STD_LOGIC;
        C : in STD_LOGIC;
        D : in STD_LOGIC;
        O : out STD_LOGIC
    );
end component top_level;

    signal A : STD_LOGIC := 0;
    signal B : STD_LOGIC := 0;
    signal C : STD_LOGIC := 0;
    signal D : STD_LOGIC := 0;
    signal O : STD_LOGIC;

begin
    UUT : top_level
    Port map(
        A => A,
        B => B,
        C => C,
        D => D,
        O => O
    );

    A <= not A after 10 ns;
    B <= not B after 20 ns;
    C <= not C after 40 ns;
    D <= not D after 80 ns;

end Behavioral;
```

Introduction To RTL Synthesis

Introduction

In the tutorial 4 (Simple RTL project) we have created a new RTL project using Vivado and we have validated its behavior in simulation using the Vivado Simulator (XSIM) in the tutorial 5 (Behavioral Simulation with the Vivado Simulator). In this tutorial we will run the synthesis on this project to understand this process.

What is synthesis?

According to the Xilinx Glossary (http://www.xilinx.com/company/terms.htm#S), synthesis is "*a process that starts from a high level of logic abstraction (typically Verilog or VHDL) and automatically creates a lower level of logic abstraction using a library containing primitives*". To be simpler, the synthesis is a process that try to realize the behavior described in the sources files (Verilog, VHDL…) using FPGA element. It converts your RTL code into FPGA elements.

Synthesize the design

Open the project and synthesize the design clicking on "*Run Synthesis*" in the "*Flow Navigator*".

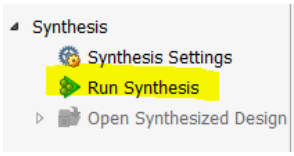

Figure 1 - Run Synthesis

When the Synthesis is done a "*Synthesis Completed*" window should appears. Select "*Open Synthesized Design*" and click "*OK*".

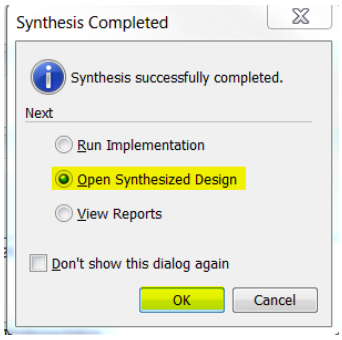

Figure 2 - Synthesis Completed window

In the "*Flow Navigator*", click on "*Schematic*" under "*Synthesized Design*" to open the synthesized design schematic.

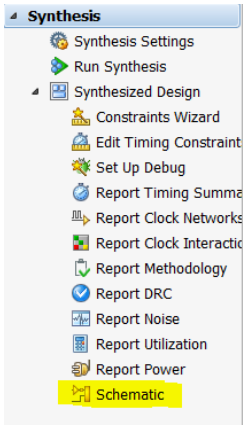

Figure 3 - open the synthesized design schematic

We can now see the synthesized design schematic. In this tutorial, we won't talk about the IBUF and OBUF elements.

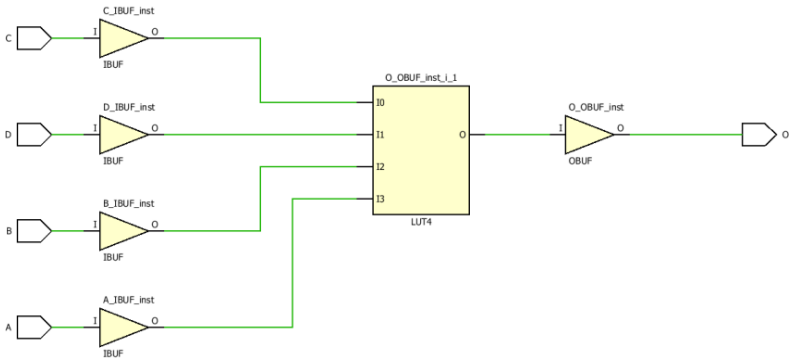

Figure 4 - synthesized design schematic

On this schematic, we can see that our AND and OR gates has been replaced by a LUT4 (Look Up Table). This is because the FPGA does not have these simple gates, so LUTs, which are basic FPGA elements, will be used to replace it. If we select the LUT4 in the schematic, we can see its "*Truth Table*" in the "*Cell Properties*" window.

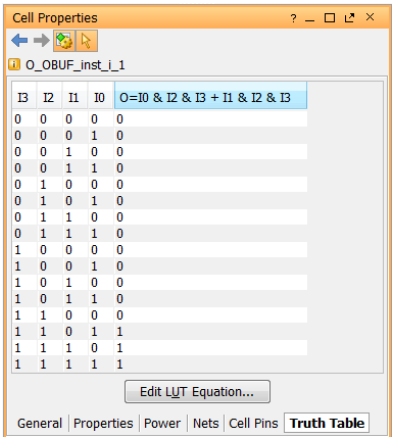

Figure 5 – LUT Truth Table

In this "*Truth Table"* we can see that the Output O is high in 3 cases. As the A, B, C and D inputs are connected respectively to the ports I3, I2, I0 and I1 of the LUT, the output is high:

- When A, B and C are high and D is low
- When A, B and D are high and C is low
- When A, B, C are D are high

This is what we expect.

In the "*Cell Properties*" we also can see the equation of the truth table:

- O = I0 & I2 & I3 + I1 & I2 & I3
- So O = C & B & A + D & B & A with our port names which is a development of our initial equation ((A & B) & (C + D))

Connected users can download this tutorial in pdf

Creating a custom IP in Vivado

Posted by Florent - 03 October 2017

Introduction

This tutorial shows how to package a RTL project (VHDL) to create a custom IP in Vivado 2017.2.

Create the module

Open Vivado 2017.2 and create a new project. Create a new VHDL file called logic_function.vhd.

We will first create a 1-bit Logical AND. In the "*Define Module*" window, define two 1-bit inputs (A_inp and B_inp) and one 1-bit output (P_outp).

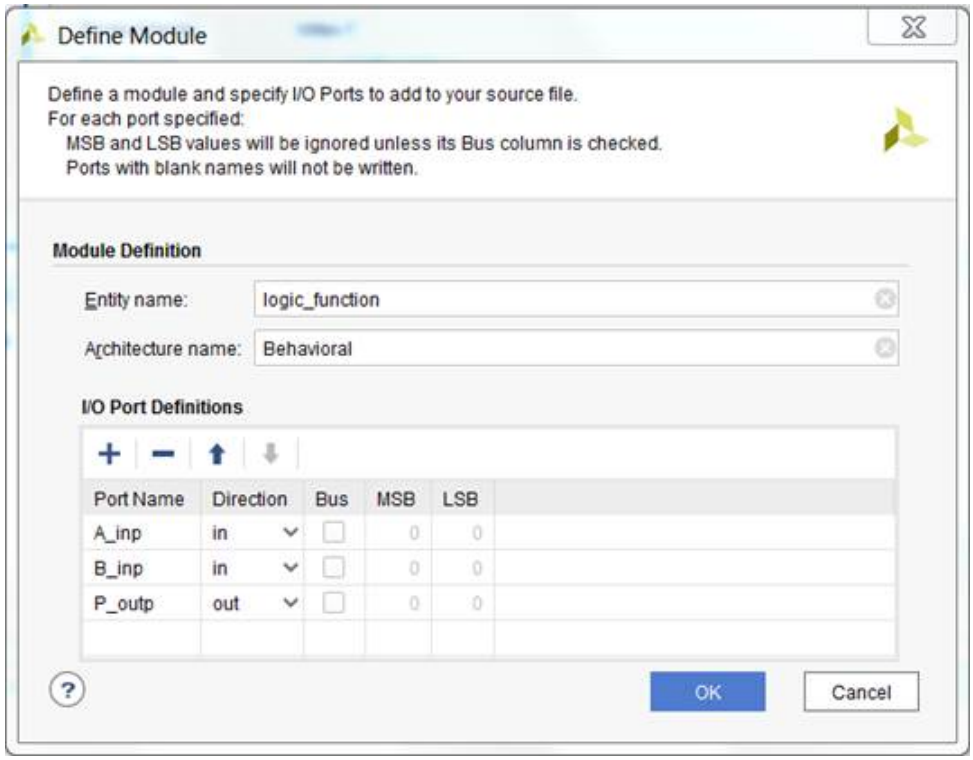

Figure 1 - Define the module

In the architecture of the module, just add the following line:

```
P_outp <= A_inp AND B_inp;
```

Package the module in an IP

Click on "*Tools > Create and Package New IP…*" to create a new project to package an IP. In the "*Create Peripherals, Package IP or Package a Block Design*" page, select "*Package your current project*".

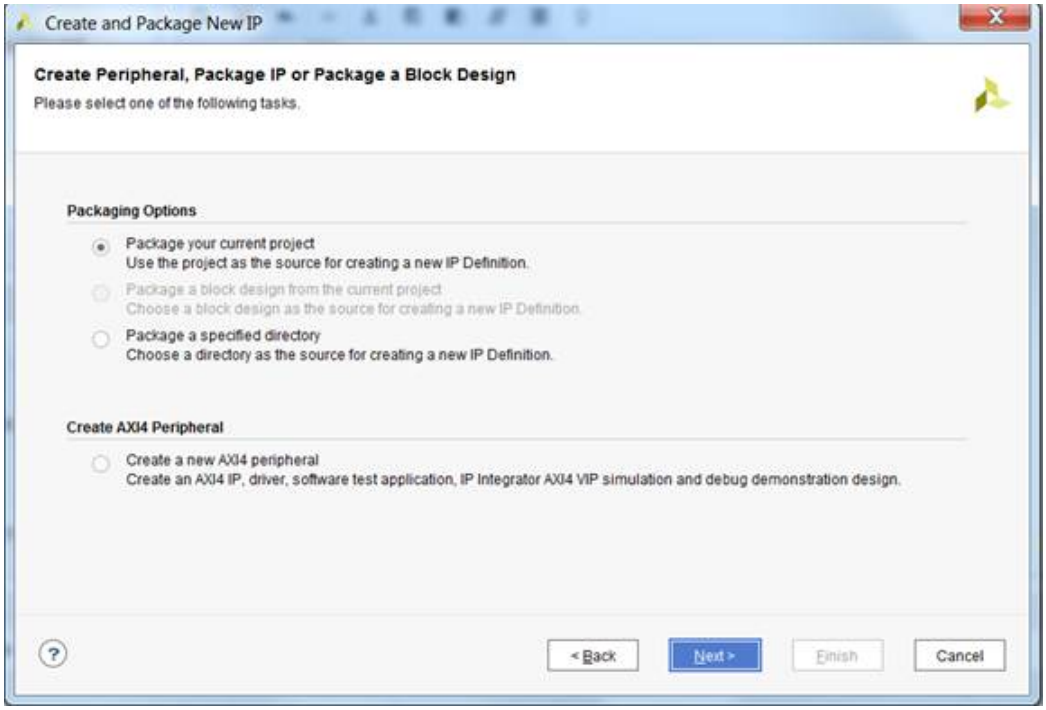

Figure 2 - Package your current project

In the "*Package your current project*" page, choose where you want the IP to be created and select "*Include .xci files*". If you are using revision control (GIT, SVN, etc…) with your custom IP definition, the UG1118 recommends to use an external repository location (outside the initial project) for your custom IP and to place the entire custom IP directory into the revision control system to preserve all the necessary outputs from the IP packager.

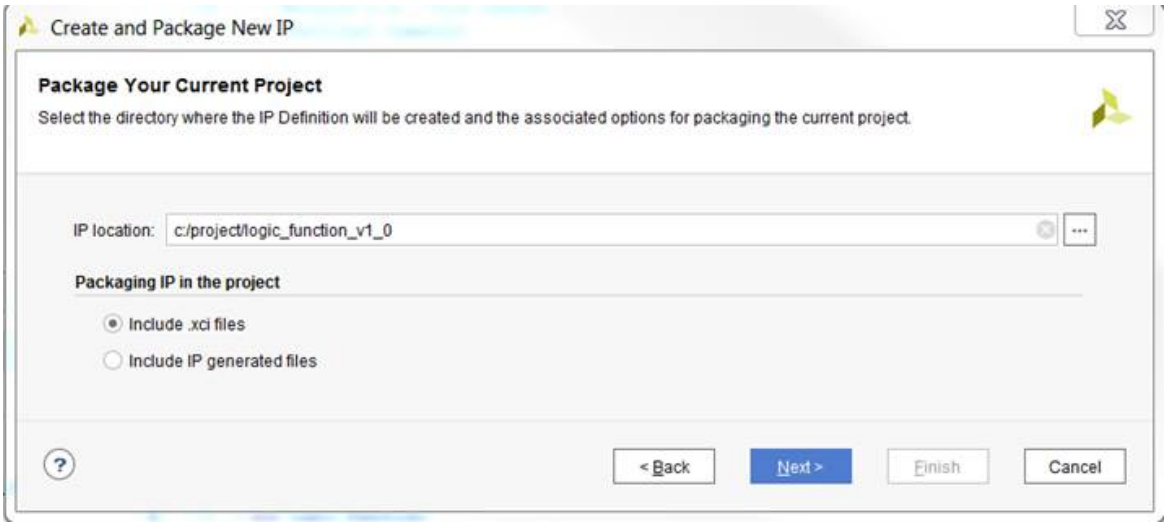

Figure 3 - Select Package IP Project Location

This create a temporary project where you can edit the IP configuration. You should have a window called "*Package IP – logic_function*" opened with various sections to configure the IP. The section "*Identification*" for example allow you to change how the IP will be identified in the Vivado IP catalog.

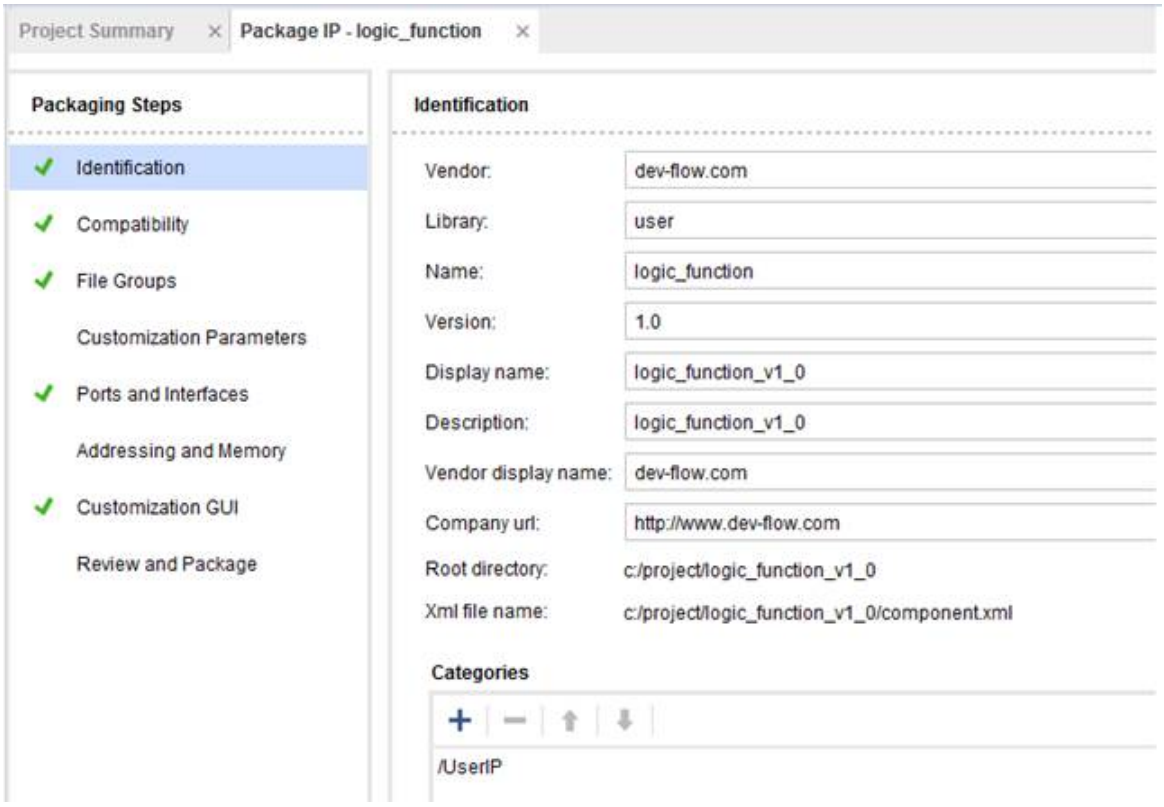

Figure 4 - Package IP – Identification

If you want to keep the IP project after packaging the IP (when using a revision control system for example), click on *Edit Packaging settings* (or go to *project settings > IP > Packager*) and disable *Delete project after packaging*.

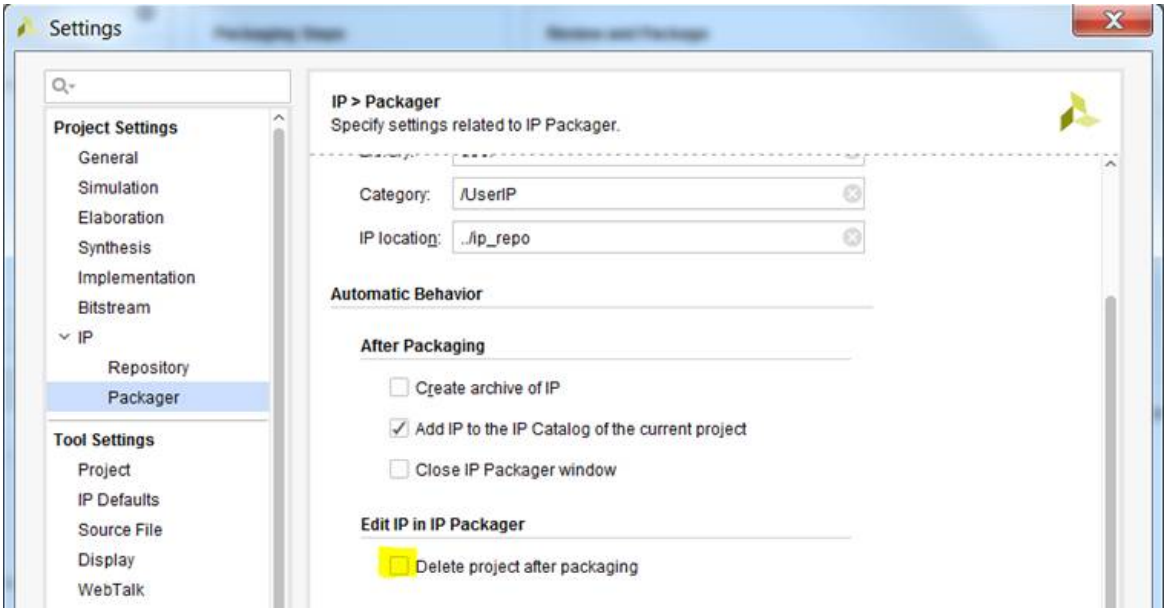

Figure 5 - Keep project after packaging

We won't do more changes for the moment. Go in the section "*Review and Package*". Click on "*Package IP*". Vivado will ask you if you want to close the project, click "*Yes*".

Then, in your initial project, if you go in the IP catalog ("*Window > IP catalog*") you should see the IP *logic_function* under *UserIP*.

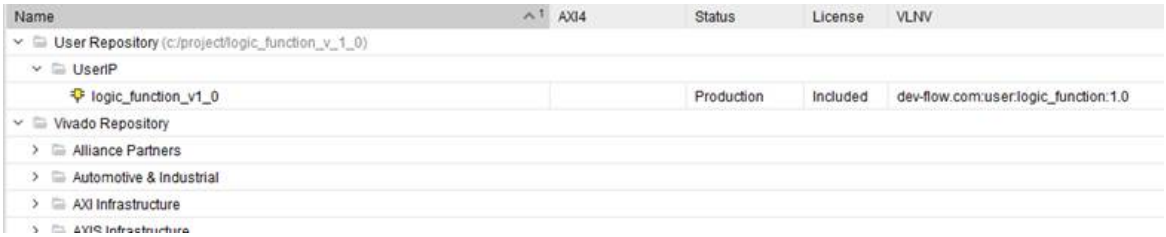

Figure 6 - User IP

Note: If you create a new Vivado project, you won't see the IP in the user catalog. This is because the folder where the IP is located is not scanned by Vivado. You need to add this folder to the "IP repositories" in the Project Settings (IP > Repository).

If we select the IP, we can see the "*IP Properties*" window with the properties of the IP. We can see that the version of the IP is 1.0 with the Rev. 1. The revision will be incremented automatically each time we package the IP. The version can be changed in the IP packaging project.

Parameterize the size of the ports

We will modify the IP to do a logical AND on buses (not only on 1-bit signals). To modify the IP:

- If you have kept the IP packager project, open this project
- If you haven't kept the IP project, open the IP catalog and search for the IP. Right click on the IP and click on "*Edit in IP Packager*".

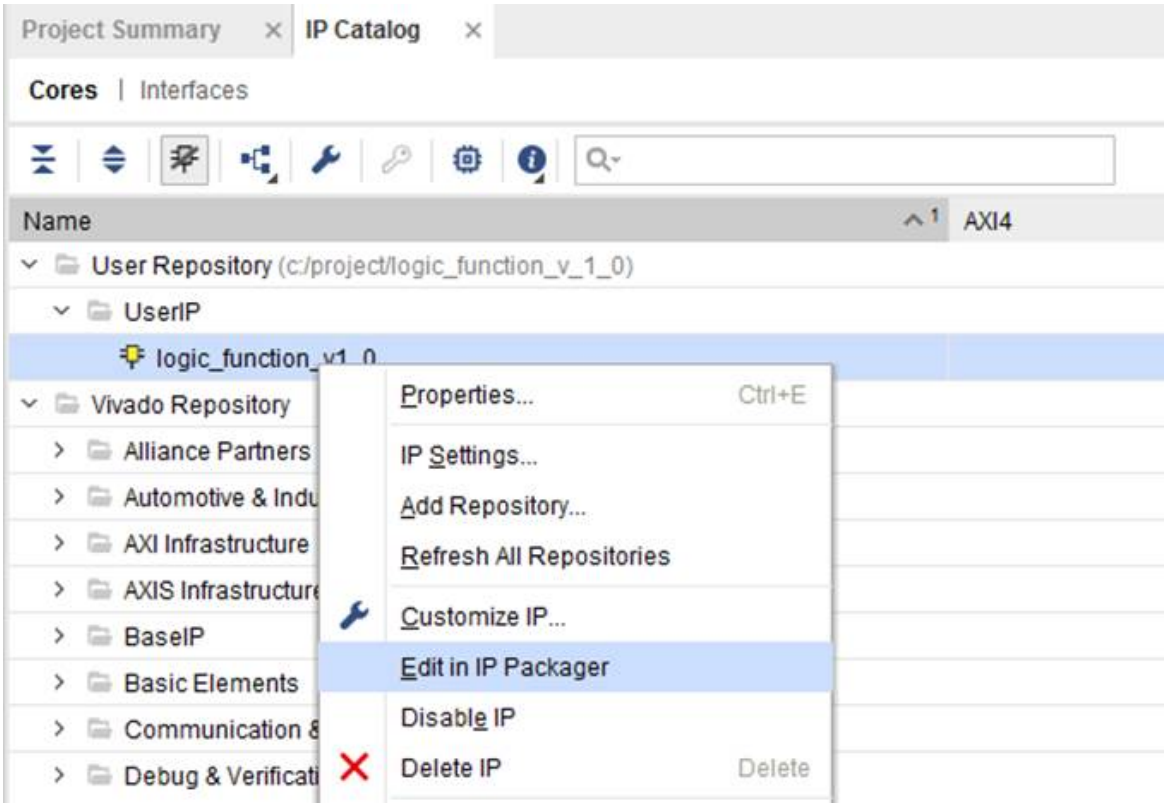

Figure 7 - Edit in IP Packager

In the design sources, doubled click on the source file (logic_function.vhd) to modify it.

```
entity logic_function
   Generic (
      DATA_SIZE : INTEGER := 32
   );
   Port (
      A_inp : in STD_LOGIC_VECTOR(DATA_SIZE-1 downto 0);
      B_inp : in STD_LOGIC_VECTOR(DATA_SIZE-1 downto 0);
      P_outp : out STD_LOGIC_VECTOR(DATA_SIZE-1 downto 0)
   );
end logic_function;
```

Change the entity by adding generics values to allow different size for the ports.

In the "*Package IP*" window, you can see that some sections don't have a green mark. This is because we made some changes to the RTL which are not taken in account in the package IP project. To update the project, go in the section "*Review and Package*". Click on one of the "*Merge Changes"* links under *Edit IP Project Changes*.

Note: If you do not see the package IP tab in vivado in you packaging IP project, you can open it by clicking on "Package IP" in the Flow Navigator

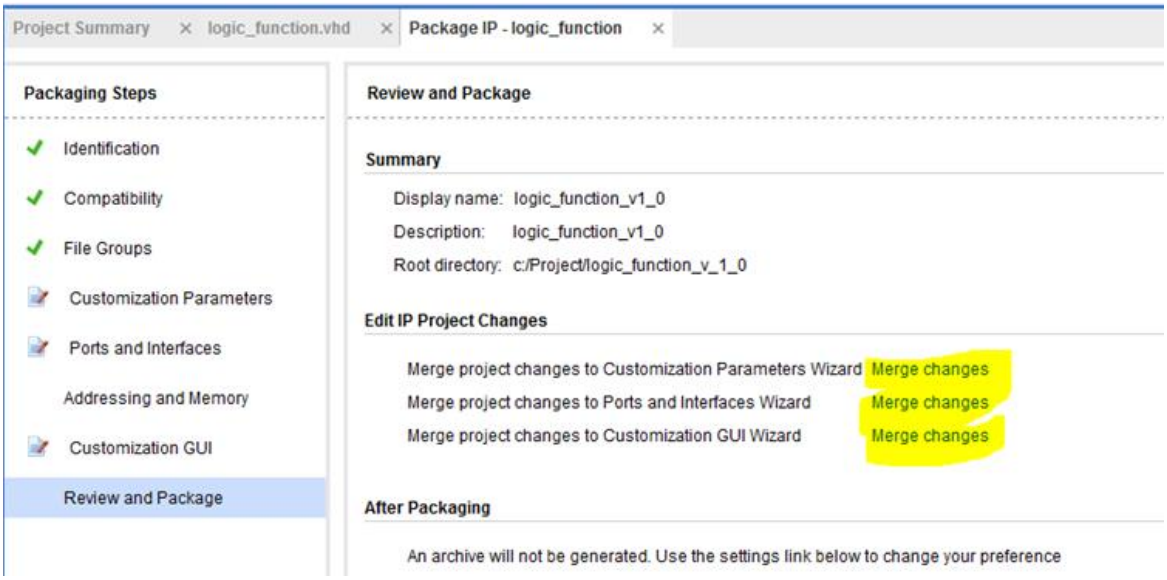

Figure 8 - Merge Changes

Then in *Customization Parameters*, under "*Hidden Parameters*", double click on "*DATA_SIZE*". In the window "*Edit IP Parameter*", enable "*Visible in customization GUI*" to be able to change the value from the configuration GUI when adding the IP to a project from the IP catalog.

Figure 9 - Customization Parameters tab

In the "*Edit IP Parameter*", enable "*Specify Range*" and select "*Range of Integer*" for "*Type*". Enter "*32*" as maximum value and click OK.

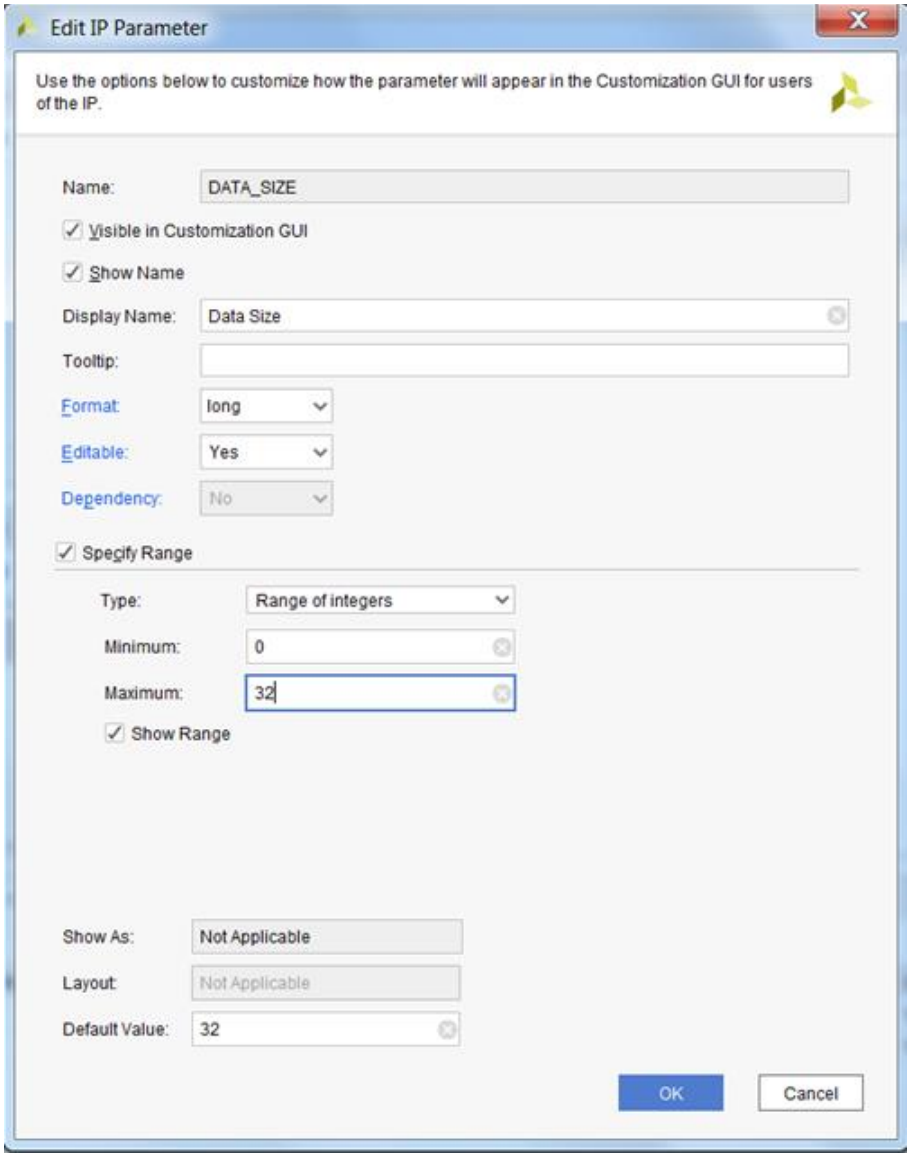

Figure 10 - Edit IP Parameter

In the "*Customization GUI*" section, move the parameter "*Data Size*" under "*Page 0*".

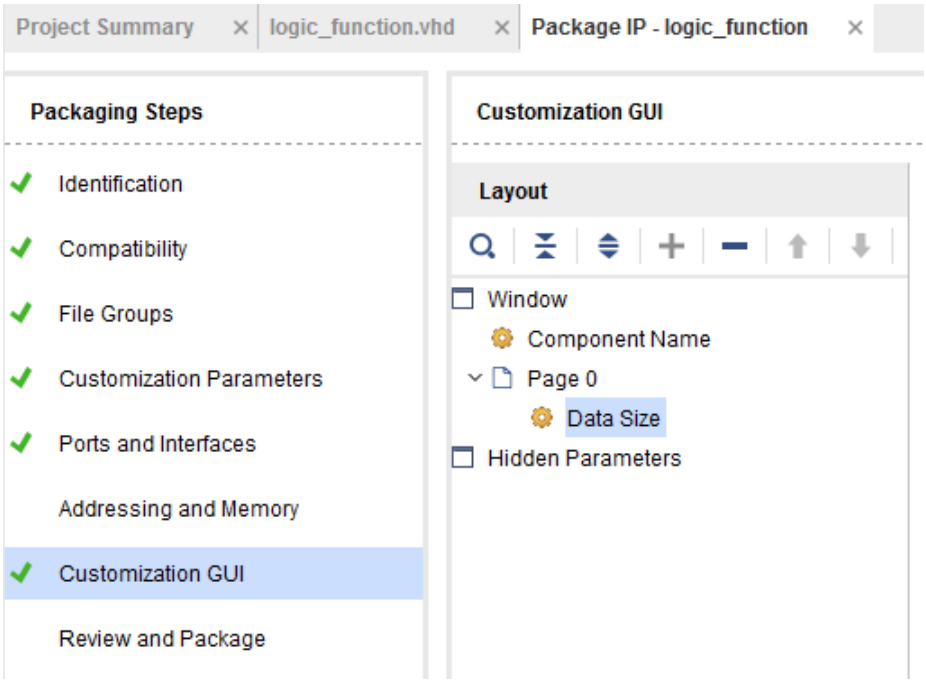

Figure 11 - Customization GUI

Then re-package the IP in the "*Review and Package*" section.

In the IP catalog, we can see that the IP revision value has been incremented (Rev. 2).

Now, if we add double click on the IP in the IP catalog, we can see that we can select the size of the data in the GUI.

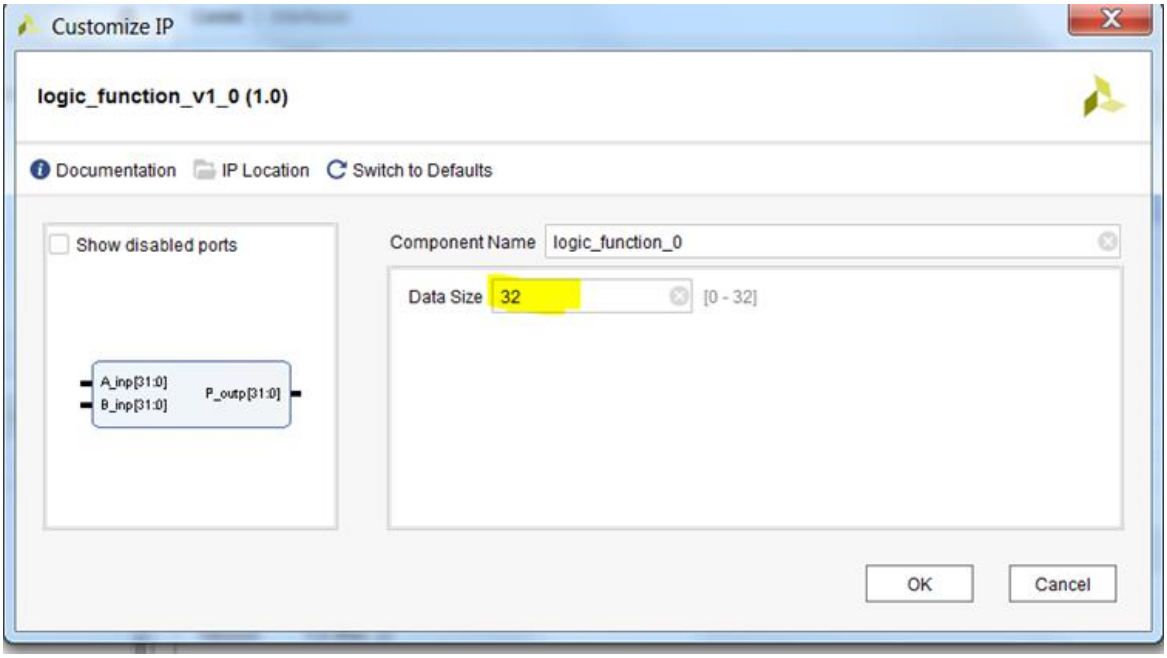

Figure 12 - IP GUI

Adding new functions

Now, we will add new functions to our IP: logical OR, logical NOR and logical NAND. And the user will be able to select the functionality he wants to use when adding the IP to a project.

```
entity logic_function is
   Generic (
      DATA_SIZE : INTEGER := 32;
      FUNC_SEL : INTEGER := 0
   );
   Port (
      A_inp : in STD_LOGIC_VECTOR(DATA_SIZE-1 downto 0);
      B_inp : in STD_LOGIC_VECTOR(DATA_SIZE-1 downto 0);
      P_outp : out STD_LOGIC_VECTOR(DATA_SIZE-1 downto 0)
   );
end logic_function;

architecture Behavioral of logic_function is

begin

   IF_AND: IF(FUNC_SEL = 0) GENERATE
      P_outp <= A_inp AND B_inp;
   END GENERATE;

   IF_OR: IF(FUNC_SEL = 1) GENERATE
      P_outp <= A_inp OR B_inp;
   END GENERATE;

   IF_NOR: IF(FUNC_SEL = 2) GENERATE
      P_outp <= NOT(A_inp OR B_inp);
   END GENERATE;

   IF_NAND: IF(FUNC_SEL = 3) GENERATE
      P_outp <= NOT(A_inp AND B_inp);
   END GENERATE;

end Behavioral;
```

Open the IP in "*IP Packager*" and open the source code and modify it with the following code.

Save the source file and merge the changes in the "*Package IP*" window. Edit the parameter *FUNC_SEL* in *Customization parameters*:

- enable "*Visible in customization GUI*"
- *Specify range:*
  - *Pairs*
  - Key: And, Value: 0
  - Key: Or, Value: 1
  - Key: Nor, Value: 2
  - Key: Nand, Value: 3

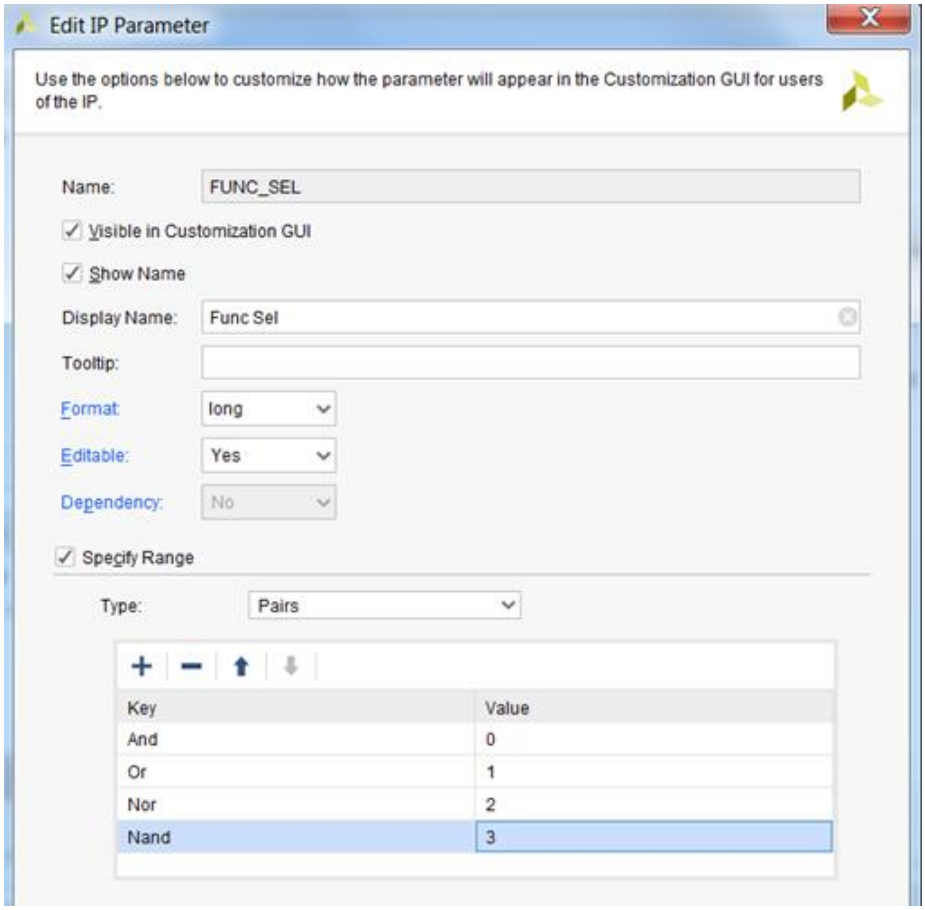

Figure 13 - Customization of parameter FUNC_SEL

In the "*Customization GUI*", move the parameter "*Func Sel*" under *Page 0* and Re-package the IP.

Now, from the IP GUI, you can select which function you want.

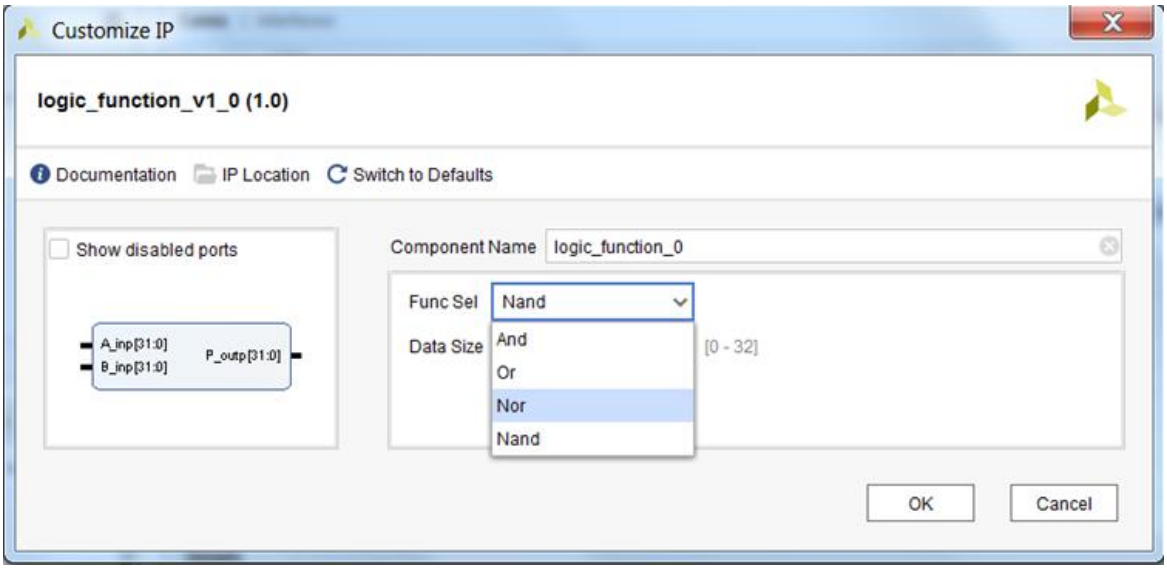

Figure 14 - Function selection in the IP GUI

Set a port as unused port

We will now add a new function to our IP: a logic NOT. For this we only need one input port. The other one will be unused so we will disable it from the IP when the function is selected in the customization GUI.

```
IF_NOT: IF(FUNC_SEL = 4) GENERATE
   P_outp <= NOT(A_inp);
END GENERATE;
```

Open the IP in "*IP Packager*" and open the source code and modify by adding the following code.

Save the source file and merge the changes in the "*Package IP*" window. Edit the parameter *FUNC_SEL* by adding a new pair (key: Not, Value: 4).

In the "*Ports and Interfaces*" section, double click on *B_inp* to open the "*Edit Port*" window. If the user want to use the logical NOT function, the input port *B_inp* will be useless. Thus we need to change *Port Presence* to "*Optional*", set *Driver Value* to "*0*" and write "$FUNC_SEL != 4" in the box below as shown in the Figure 13.

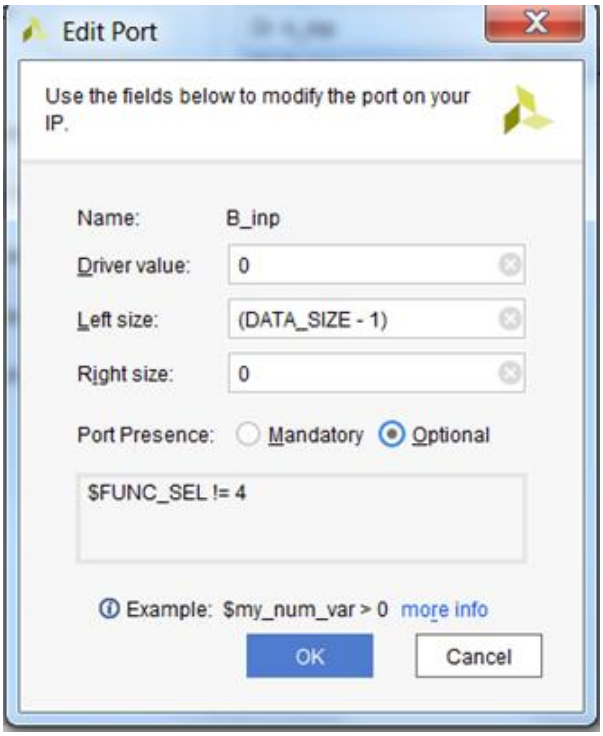

Figure 15 - Edit port B_inp

Re-package the IP.

Now in our original project, we can see that if we select the function *NOT*, the input port B_inp will be disabled for the IP.

Figure 16 - Port B disabled

Post a comment

Only connected users can post comments