

Using Fundamental Gates Lab

Overview:

In this lab you will learn how to model simple gates using Verilog HDL and use them to create a more complex design. You will use fundamental gates using language supported primitive gates. After building the basic models you will create a hierarchical design.

Outcome:

You will understand how to use Verilog primitive gates. You will learn how to create a model using ISE Create Project wizard. You will instantiate lower-level models to create a bigger model. You will use ISE simulator to simulate the design. You will add user constraint file (ucf) to assign pins so the design can be targeted to National Instruments (NI) Digital Electronics FPGA Board. You will implement the design and create a bitstream file using ISE's implementation tools. Once bitstream is created, you will download using ISE's iMPACT program and verify the design functionality.

Background:

Verilog HDL is a hardware description language that can be used to model a digital system at many levels of abstraction ranging from algorithmic- to the gate- to the switch-level. The complexity of the digital system being modeled could vary from a simple gate to a complete system. Various levels of abstractions can be used in modeling the digital system based on its functionality and complexity. The language supports constructs and means to model the digital system in a hierarchical fashion. It also allows designer to describe timing explicitly. The richness of the language constructs is exploited by using same language constructs to test the system.

A system- from a simple gate to a complex circuit- typically will have some input signals and some output signals to interact with either other digital devices or external board, and will have some functionality for which it has been designed. The basic unit of description in Verilog is the module. A module describes the functionality of a design and also describes the ports through which it communicates. The basic syntax of the module is:

```
module module_name (port_list);  
    Declarations  
    Statements  
endmodule
```

The Verilog language is case sensitive. In the above example **module** and **endmodule** are keywords describing beginning of a module definition and ending of the module definition. You can not have nested module definitions, i.e. you can not have another module keyword within a module-endmodule pair. In the language, a statement is delimited by a semicolon. You can have multiple statements on a given line. The declarations in the above example can be definition of data types such as wire and reg, can be parameter definition, ports direction, functions, and tasks to name few. The Statements can be initial, always, and continuous assignment statements as well as module, gate, UDP (User Defined Primitives) instantiations. The Statements describe the

actual functionality of the module. The identifiers must be defined using Declarations before they can be used.

The language defines three fundamental modeling styles. In a given module all or subset of these styles can be used. The three modeling styles are: Structural, Dataflow, and Behavioral. This lab exercise uses Structural style modeling. Structure can be described in Verilog using Built-in gate primitives, Switch-level primitives, User-Defined Primitives (UDP), and module instances. The Switch-level primitives are used to model fundamental gate functionality or a system built with switches or transistors. The UDP are used to define a unit as a black-box with providing functionality in truth-table form and explicit timing relationships between input and output ports. In this lab exercise you will use gate primitives and module instantiations. The language defines the following gates:

Gate Types		Component
Gates	Allows strengths	and, nand, or, nor, xor, xnor buf, not
Three State Drivers	Allows strengths	buif0, buif1 notif0, notif1
MOS Switches	No strengths	nmos, pmos, cmos, rnmos, rpmos, rcmos
Bi-directional switches	No strengths, non resistive	tran, tranif0, tranif1
	No strengths, resistive	rtran, rtranif0, rtranif1
	Allows strengths	pullup pulldown

[Verilog Quick Reference]

Here is an example of instantiating a nor gate:

```
nor X1 (S1, A, B);
```

where X1 is the instance name. It is optional for the gate- or switch-level instantiation. The instance name is required for a module instantiation. S1 is output, and A and B are input.

References:

1. National Instruments' Digital Electronics FPGA Board user manual
2. Verilog HDL books
Stephen Brown, Zvonko G. Vranesic, "Fundamentals of Digital Logic with Verilog Design", 2002
Zainalabedin Navabi, "Verilog Digital Systems Design: RT Level Synthesis, Testbench, and Verification", 2005
Samir Paltinkar, "Verilog HDL: A Guide to Digital Design and Synthesis", 2003
Joseph Cavanagh, "Verilog HDL: Digital Design and Modeling", 2007
Michael D. Ciletti, "Modeling, Synthesis, and Rapid Prototyping with Verilog HDL", 2003
Douglas J. Smith, "HDL Chip Design: A Practical Guide for Designing, Synthesizing and Simulating ASICs and FPGAs using VHDL or Verilog", 1996
3. On-line references:
Verilog HDL Reference Card: http://www.stanford.edu/class/ee183/handouts_win2003/VerilogQuickRef.pdf
Verilog Quick Reference: http://frank.harvard.edu/~howard/pulsenet/docs/verilog_quikref.pdf
Tutorial on Verilog: <http://faculty.kfupm.edu.sa/COE/about/COE%20202%20Verilog%20Guidelines.pdf>

Problem Statement:

Design a minority gate that has three inputs and one output. The output is logic 1 whenever the numbers of inputs which are logic 1 are zero or less than half of the input (i.e. 1 in this case).

Implementation:

The circuit to be designed consists of three inputs and one output. Typically such circuit can be implemented using a combinational network. A truth table is created and then some Boolean minimization technique (e.g. K-Map) may be used to reduce the number of logic gates used. The truth table for the design at hand is shown next along with K-Map and minimized Boolean expression.

Inputs			Output
A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

		AB			
		00	01	11	10
C	0	1	1	0	1
	1	1	0	0	0

$$F = \overline{A} \overline{C} + \overline{B} \overline{C} + \overline{A} \overline{B}$$

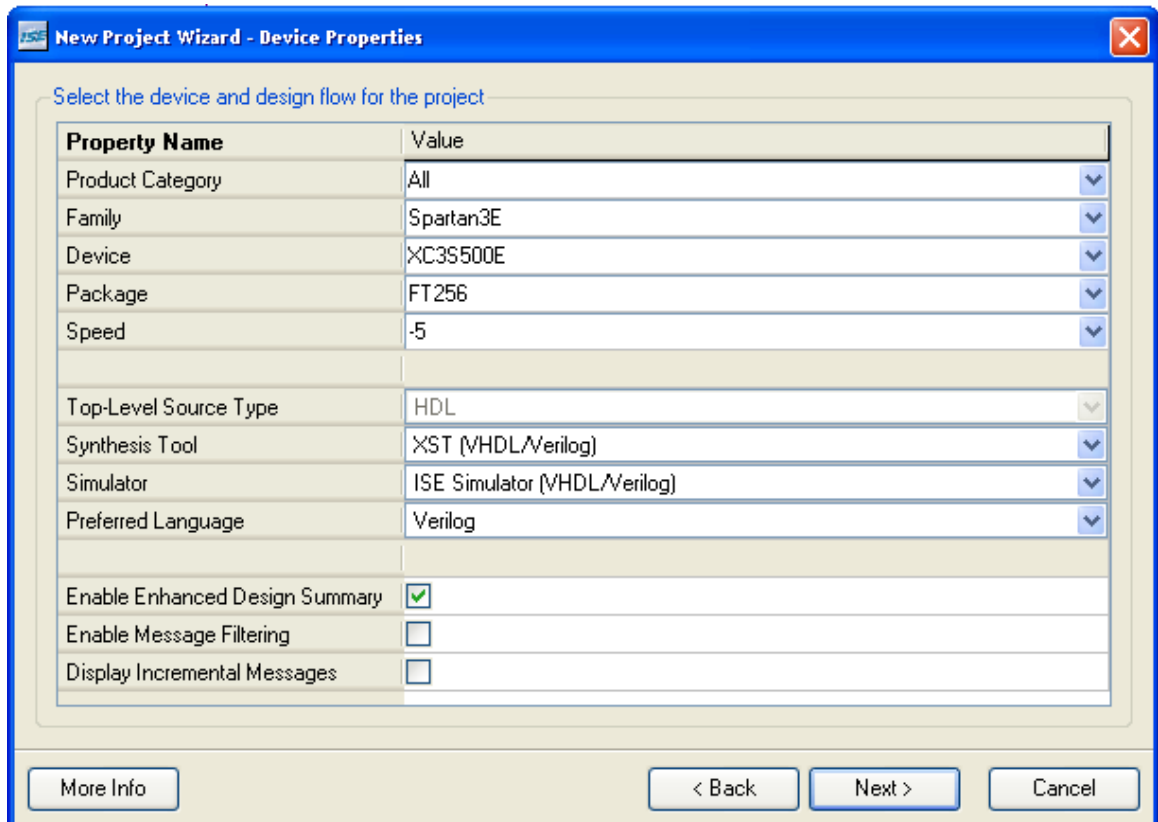
$$F = (\overline{A} + \overline{C}) * (\overline{B} + \overline{C}) * (\overline{A} + \overline{B})$$

The two expressions are equivalent. We will use the 2nd expression in this design.

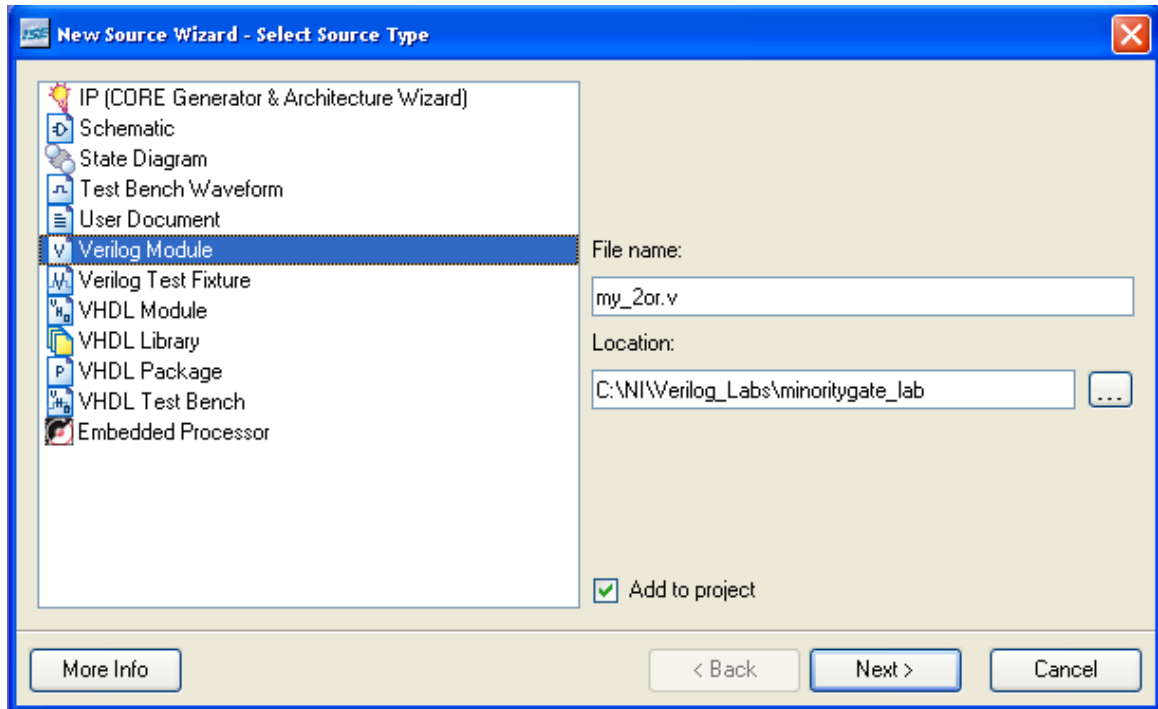
Procedure:

1. Create a ISE project

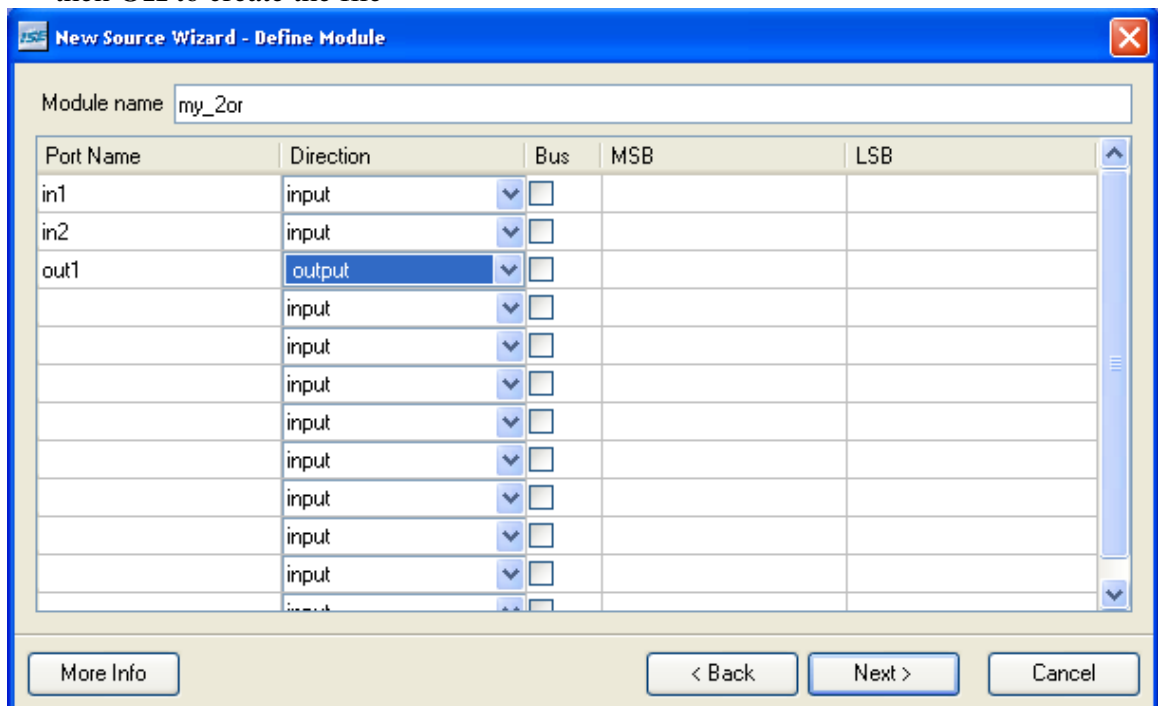
- Launch ISE: Select **Start** → **Programs** → **Xilinx ISE Design Suite 10.1** → **ISE** → **Project Navigator**
- In the Project Navigator, select **File** → **New Project**. The New Project Wizard opens
- For Project Location, use the “...” button to browse to C:\NI\Verilog_labs, and then click **OK**
- For Project Name, type *minoritygate_lab*
- Click **Next**
- Select the following options and click **Next**
 - ✓ Device Family: **Spartan3E**
 - ✓ Device: **xc3s500E**
 - ✓ Package: **ft256**
 - ✓ Speed Grade: **-5**
 - ✓ Synthesis Tool: **XST (VHDL/Verilog)**
 - ✓ Simulator: **ISE Simulator (VHDL/Verilog)**
 - ✓ Preferred Language: **Verilog**



- The **Create New Source** dialog will appear. You can use this dialog to create a new HDL source file by defining the module name and ports. You will do this once to get experience. Subsequent files creation will be done using a text file rather than using the dialog. Click **New Source** button
- A new source wizard will appear. Select *Verilog Module* as a source type and enter **my_2or** in the *File name* field and click **Next**



- A *Define Module* form will appear. Enter **in1**, **in2** as the *input* in *Port Name* field and **out1** as the *output*. Change the input Direction to output by clicking and drop-down button and selection output as the direction. Click **Next** and then **OK** to create the file



- Click **Next** and *Add Existing Sources* form will be displayed. Click **Finish** as we do not have anything to add
- Click **Finish**. A project will be created and *my_or2.v* file be created and added to the project. The file will be opened and displayed in the editor

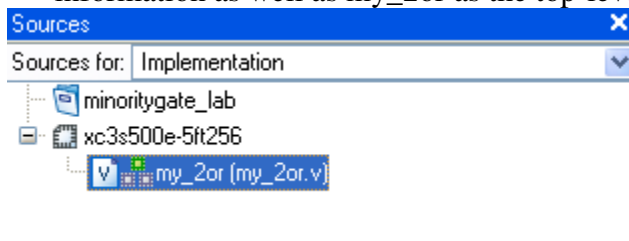
window. You can enter related information regarding project name, target devices, etc. You will also see a module and endmodule statements are created. The only thing that is needed is to add the functionality

```

1  `timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date:    00:26:32 01/11/2009
7  // Design Name:
8  // Module Name:   my_2or
9  // Project Name:
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 ///////////////////////////////////////////////////////////////////
21 module my_2or(
22     input in1,
23     input in2,
24     output out1
25 );
26
27
28 endmodule
29

```

You will notice that the project hierarchy view window showing the part information as well as my_2or as the top-level model



- Enter a gate-level instantiation statement that uses primitive gate to perform *or* function on the two inputs and output the result with 2 ns delay


```

21 module my_2or(
22     input in1,
23     input in2,
24     output out1
25 );
26
27     or #2 O1 (out1, in1, in2);
28
29 endmodule
30

```

- Save and close the file



- Click *New* button () and select *text file* and click **OK**
- Enter the **module**, **endmodule**, and gate-level instantiation statement to model 2-input *and* function using the port names as **in1**, **in2**, and **out1**

```
1 module my_2and(  
2     input in1,  
3     input in2,  
4     output out1  
5 );  
6  
7     and #2 A1 (out1,in1,in2);  
8  
9 endmodule  
10
```

- Save and close the file, giving **my_2and.v** as the filename
- Similarly, create a new model for the minority gate functionality (3 inputs [in1, in2, and in3] and one output [out1]), instantiating *my_2or* and *my_2and* models, and using necessary number of fundamental gates instantiations and having 2 ns as the delay. Use named-mapped port convention. The named-port convention requires a port name of the parent module be listed first preceded with “.” followed by a net name that is connecting that port in the current module. In the below figure, .in1 is the port name of my_or2 module (parent module) where as in1_n is the net name connecting to that port. In named-port mapping convention, one need not mention all the ports of the parent module. The unlisted parent module ports are drive logic 0 if they input and left unconnected if they are output.

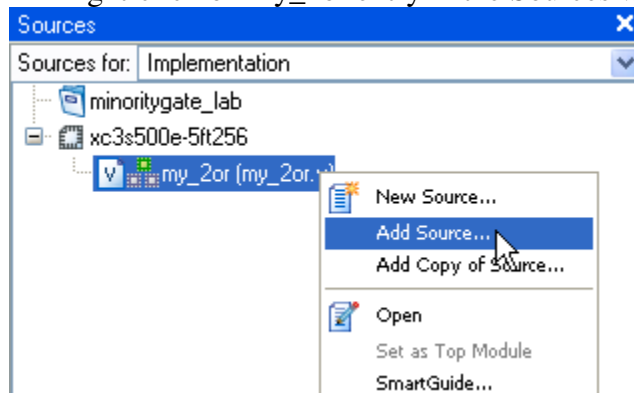

```

1  module minoritygate( input in1, input in2, input in3, output out1);
2
3  wire in1_n, in2_n, in3_n, net1, net2, net3, net4;
4
5  not #2 N1 (in1_n,in1);
6  not #2 N2 (in2_n,in2);
7  not #2 N3 (in3_n,in3);
8
9  my_2or U1(
10     .in1(in1_n),
11     .in2(in2_n),
12     .out1(net1)
13 );
14
15  my_2or U2 (
16     .in1(in1_n),
17     .in2(in3_n),
18     .out1(net2)
19 );
20
21  my_2or U3 (
22     .in1(in2_n),
23     .in2(in3_n),
24     .out1(net3)
25 );
26
27  my_2and U4(
28     .in1(net1),
29     .in2(net2),
30     .out1(net4)
31 );
32
33  my_2and U5(
34     .in1(net3),
35     .in2(net4),
36     .out1(out1)
37 );
38
39  endmodule
40

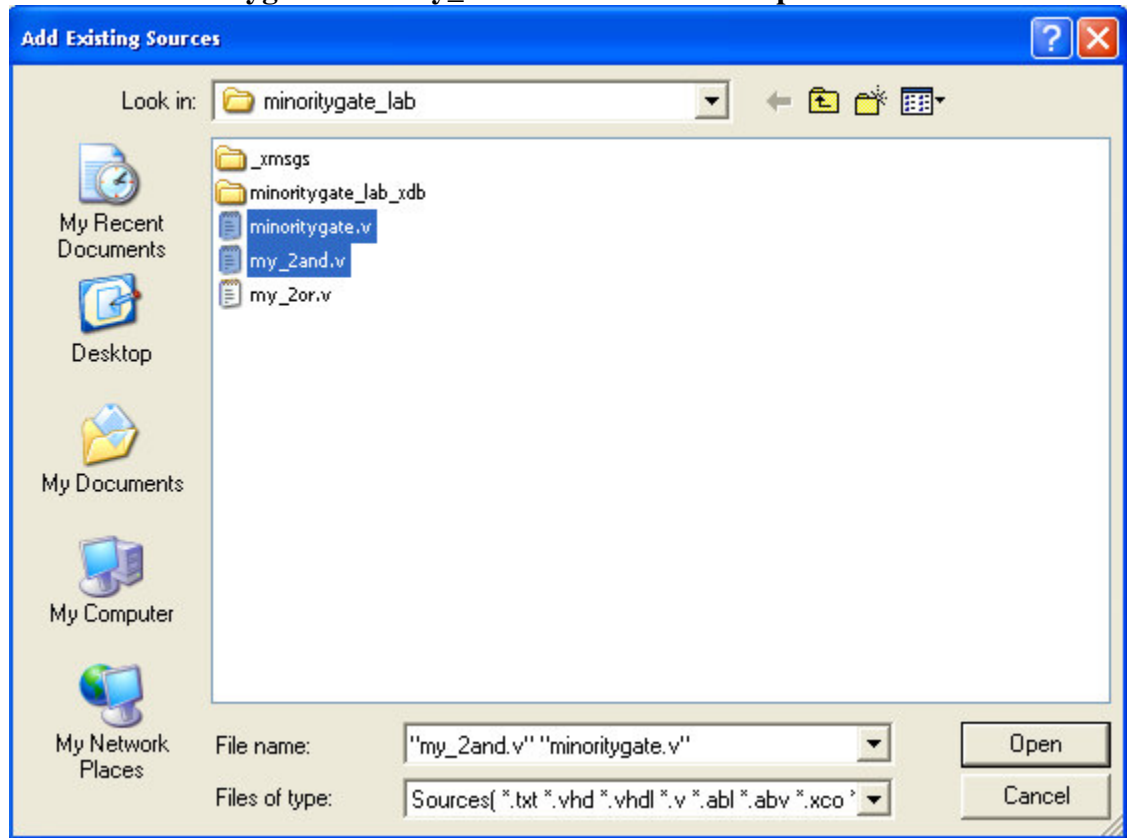
```

Named Port Association

- Right-click on my_2or entry in the Sources window and select Add Source ...



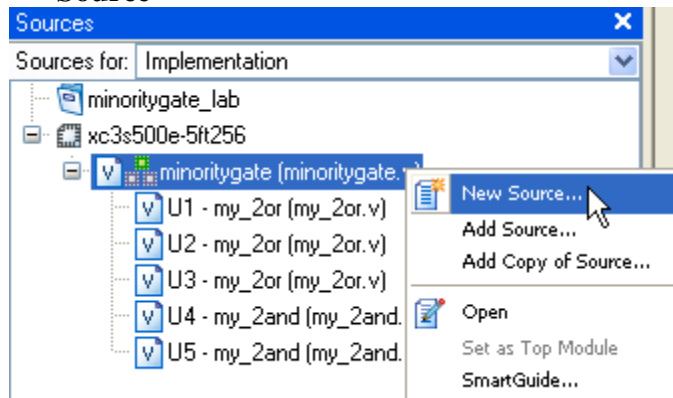
- Select **minoritygate.v** and **my_2and.v** files and click **Open**



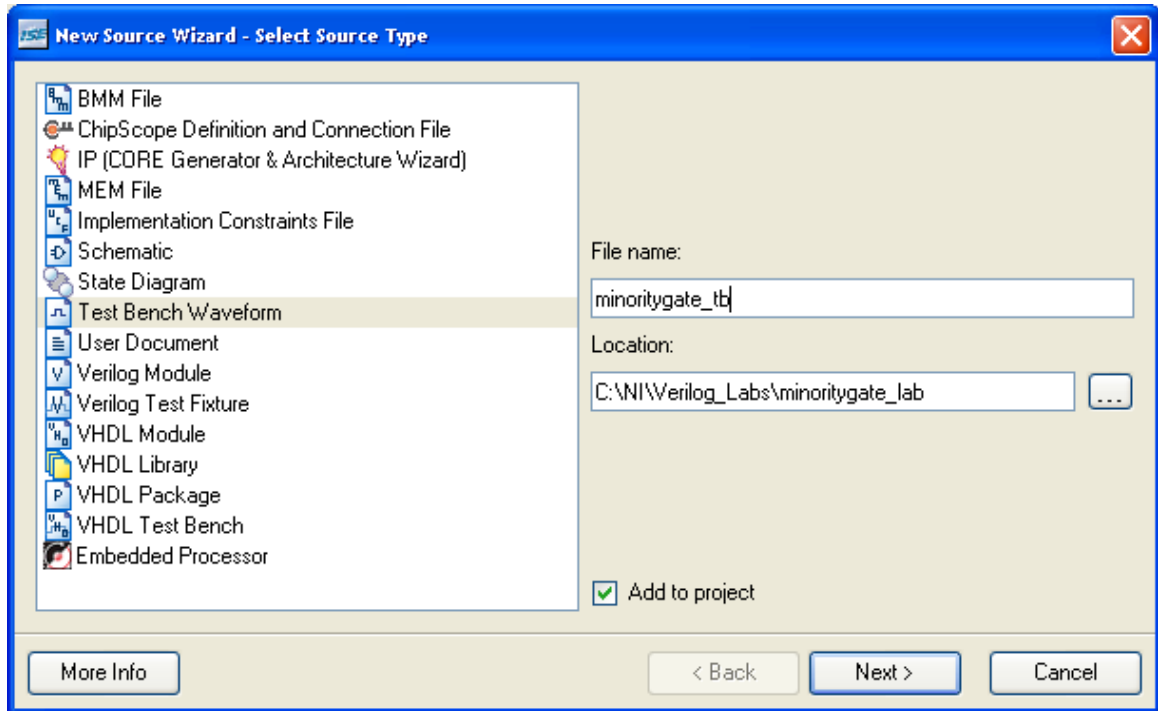
- Click **OK** to finish the addition

2. Simulate the design using ISIM

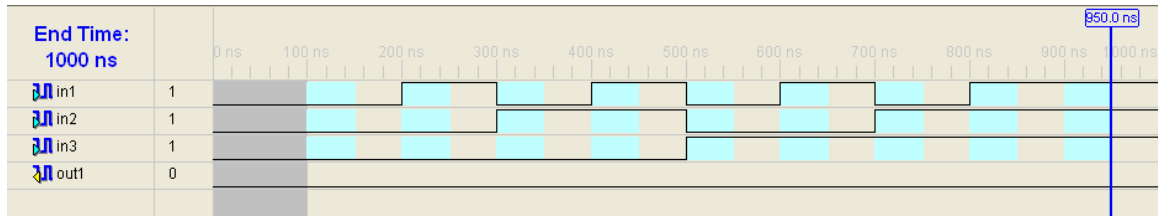
- Right-click on the **minoritygate** entry in *Sources* window and select **New Source**



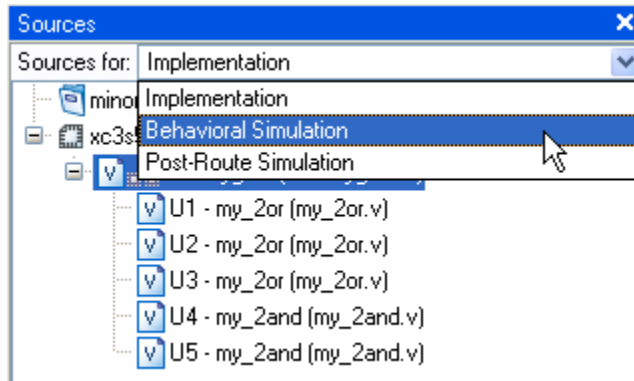
- Select Test Bench Waveform as the source type and enter **minoritygate_tb** in the File name field. Click **Next**



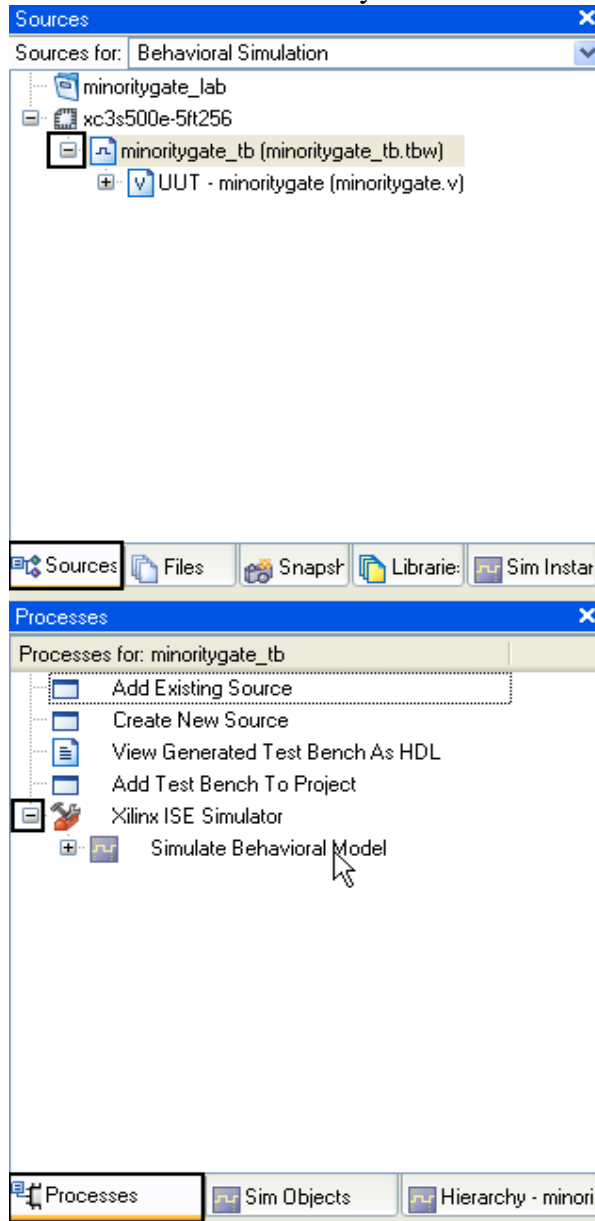
- Select **minoritygate** as the UUT to associate the testbench to the source model and click **Next** followed by clicking **Finish** button
- Select combinatorial (internal clock) and click **Finish** to display the input and output signals. Click appropriately in the waveform window to generate input stimulus as shown below



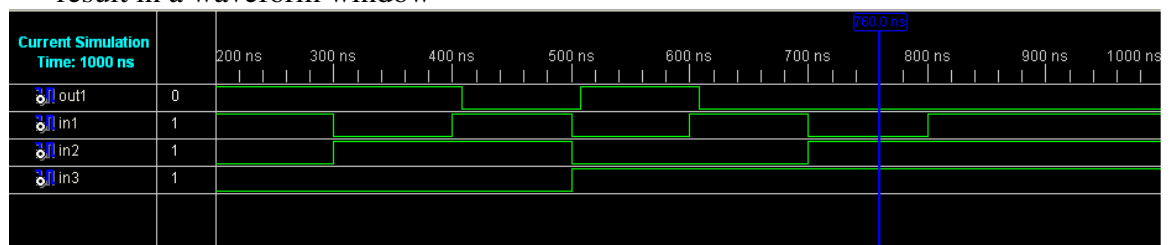
- Save the file
- Select *Sources* tab and then click **minoritygate** in *Sources* window, and click on the drop-down button of the **Sources for** window and select **Behavioral Simulation**



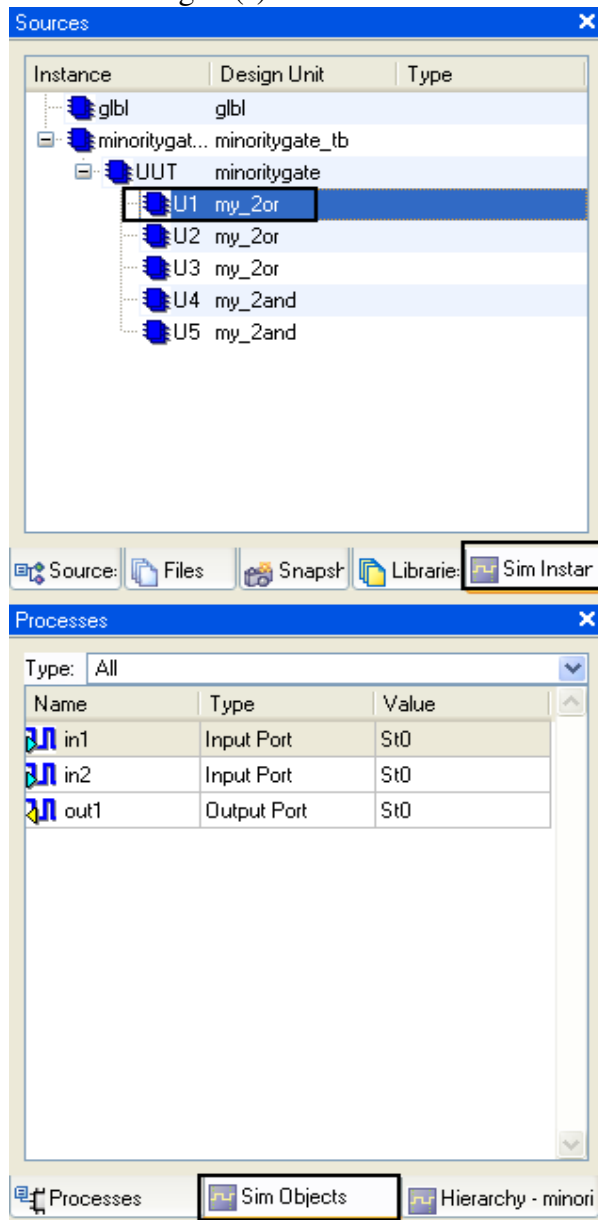
- Select **minoritygate_tb** in *Sources* tab, expand its entry to see UUT. Select *Processes* tab, expand **Xilinx USE Simulator**, and double-click on **Simulate Behavioral Model** entry to run the simulator











- The source files will be compiled and the executable file (minoritygate_tb_ism_beh.exe) be generated which is then run, displaying the result in a waveform window



- You can view lower-level signals by selecting *Sim Insulator* tab in **Sources** window, expanding hierarchy, selecting a particular instance, then clicking *Sim Objects* in tab in **Processes** window, and then selecting and dragging a desired signal(s) in the waveform window

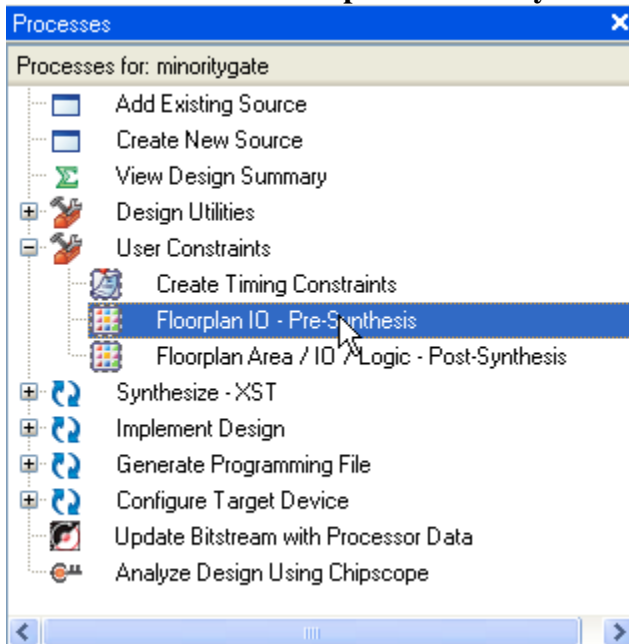


- You can zoom-in or zoom-out the waveform window by clicking on appropriate buttons (   )
- You can *restart* by clicking on () button and re-run by clicking on *run for the specified time* () button
- You can change the run-time by typing in the *new time* field ( ns ), clicking *restart* button, and then clicking on *run for the specified time* button

- Close the simulator by closing the waveform window and click OK to close the active simulator

3. Implement the design

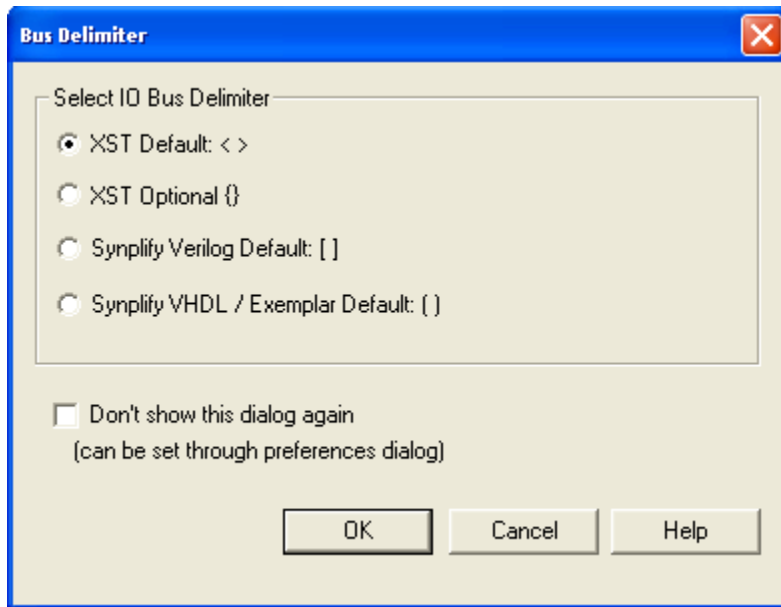
- Select **implementation** in *Sources for* window
- Select **minoritygate** module in *Sources* window
- Expand **User Constraints** processes in *Processes* window
- Double-click on **Floorplan IO Pre-Synthesis** to open *PACE* program



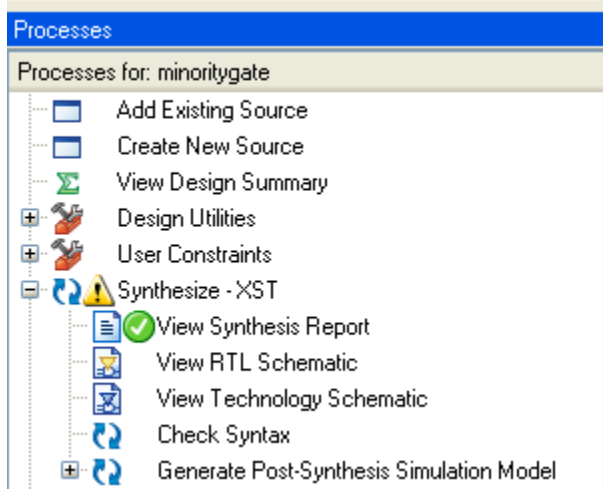
- Click **OK** and then **Yes** to add ucf file
- In *PACE* window assign pin *locations* and *I/O Std* as shown below

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcc
in1	Input	J11	BANK	LVCMD533	N/A	3.30
in2	Input	J12	BANK	LVCMD533	N/A	3.30
in3	Input	H16	BANK	LVCMD533	N/A	3.30
out1	Output	C11	BANK	LVCMD533	N/A	3.30

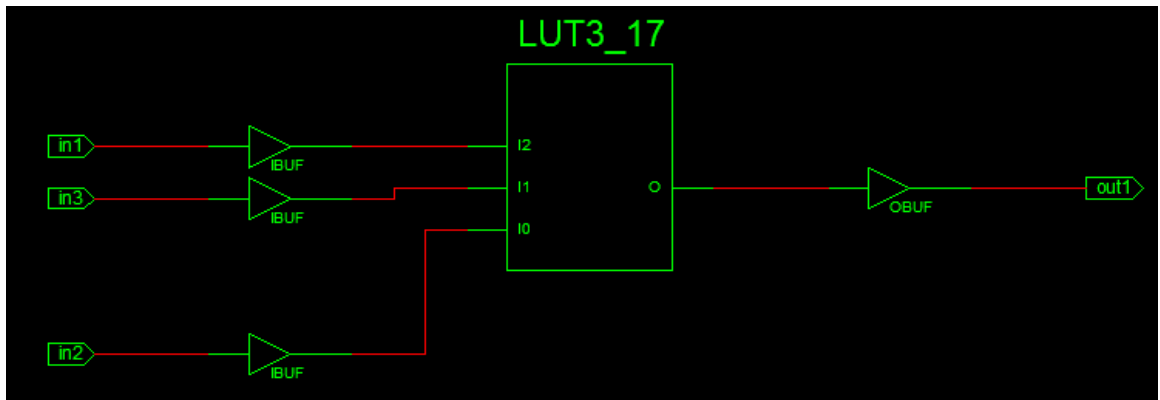
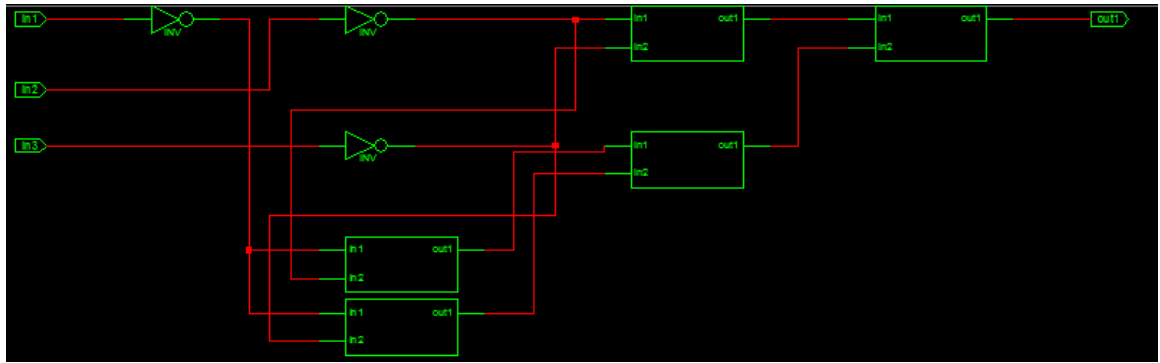
- Click **Save** button
- A dialog box will appear. Choose **XST Default: <>** option and click **OK** as we are using XST synthesis tools



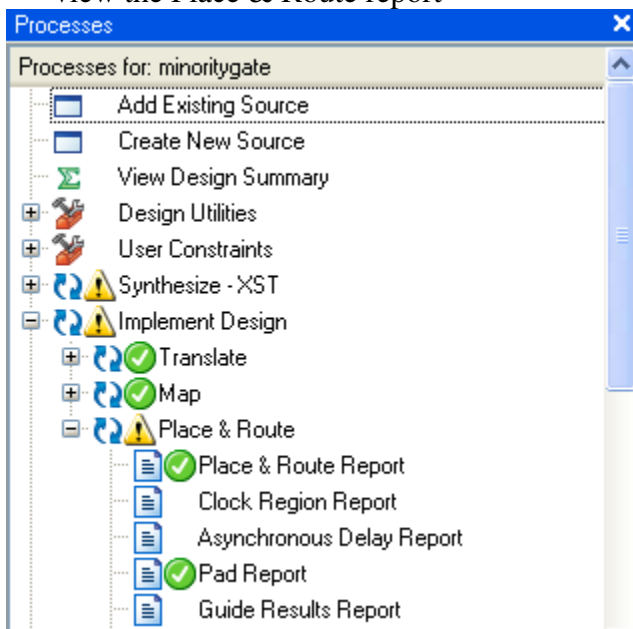
- Close the *PACE* program using **File** → **Close**
- A **minoritygate.ucf** file will be added to the project. Open that file and see how the constraints are written
- Close the file
- Select **minoritygate** in *Sources* window and double-click on **Implement Design** process in *Processes* window. This will go through Synthesis, and Implementation stages
- Expand Synthesis processes and double click on View RTL Schematic and View Technology Schematic processes to get different views



- You can push-in to a lower-level schematic by double-clicking on the top-level



- When the implementation is completed, expand Implement Design process to view the Place & Route report



- Double-click on the **Place & Route Report** to view the report. Look at the resource utilization and note that **1** slice is being used
- You can see similar information by clicking on Design Summary tab and looking at the various information

NO partition information was found.

Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(s)
Number of 4 input LUTs	1	9,312	1%	
Logic Distribution				
Number of occupied Slices	1	4,656	1%	
Number of Slices containing only related logic	1	1	100%	
Number of Slices containing unrelated logic	0	1	0%	
Total Number of 4 input LUTs	1	9,312	1%	
Number of bonded IOBs	4	190	2%	

Performance Summary			
Final Timing Score:	0	Pinout Data:	Pinout Report
Routing Results:	All Signals Completely Routed	Clock Data:	Clock Report
Timing Constraints:			

Detailed Reports					
Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report	Current	Sun Jan 11 08:56:50 2009	0	5 Warnings	0
Translation Report	Current	Sun Jan 11 08:57:04 2009	0	0	0
Map Report	Current	Sun Jan 11 08:57:22 2009	0	0	2 Infos
Place and Route Report	Current	Sun Jan 11 08:57:42 2009	0	1 Warning	1 Info
Static Timing Report	Current	Sun Jan 11 08:57:47 2009	0	0	3 Infos

Project Properties

- Enable Enhanced Design Summary
- Enable Message Filtering
- Display Incremental Messages

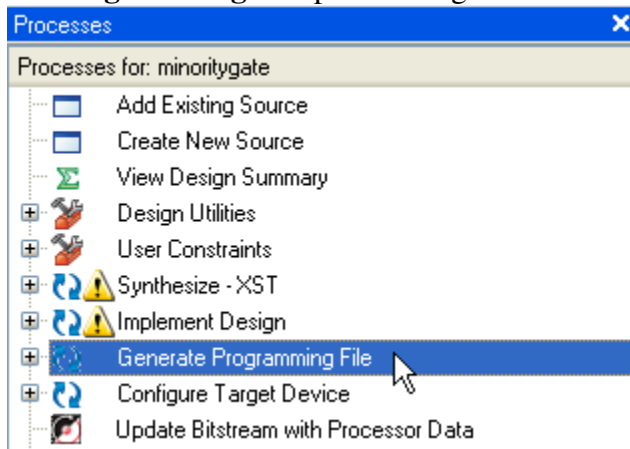
Enhanced Design Summary Contents

- Show Partition Data
- Show Errors
- Show Warnings
- Show Failing Constraints
- Show Clock Report

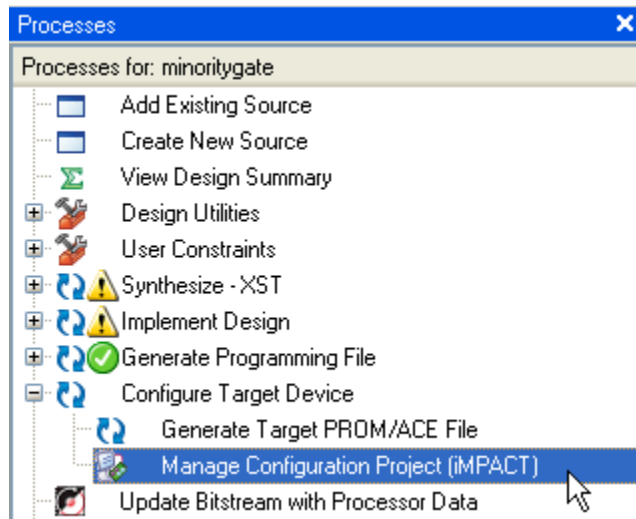
What's New in ISE Design Suite 10.1 | Design Summary | Place and Route Report

4. Verify the design in hardware

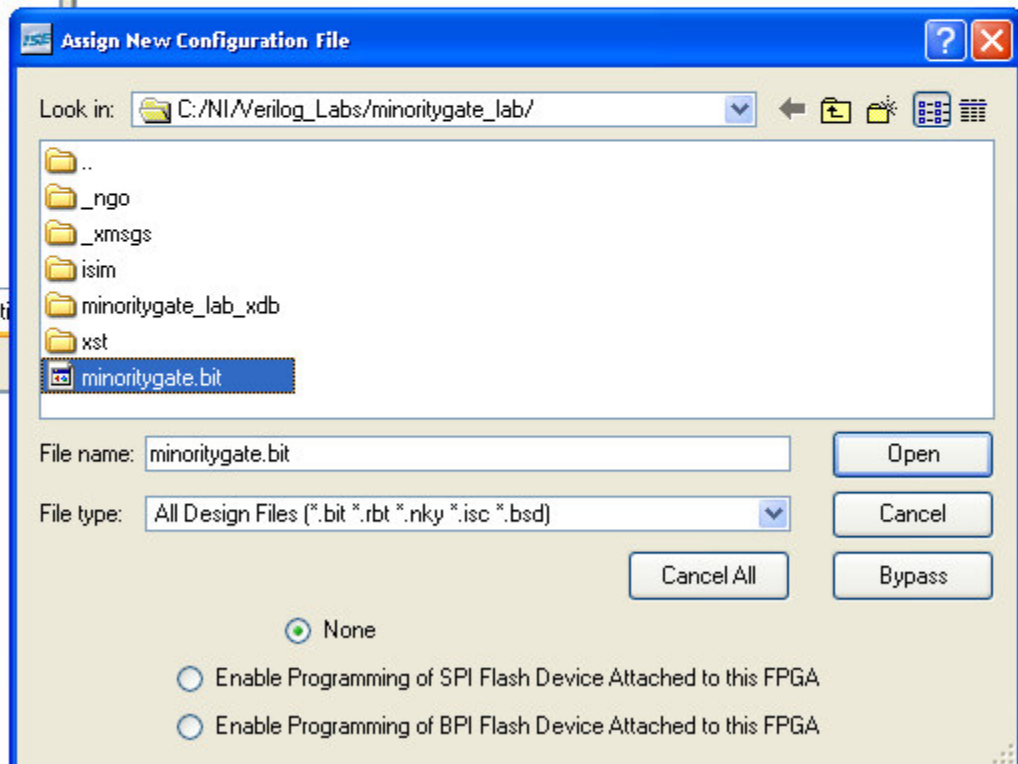
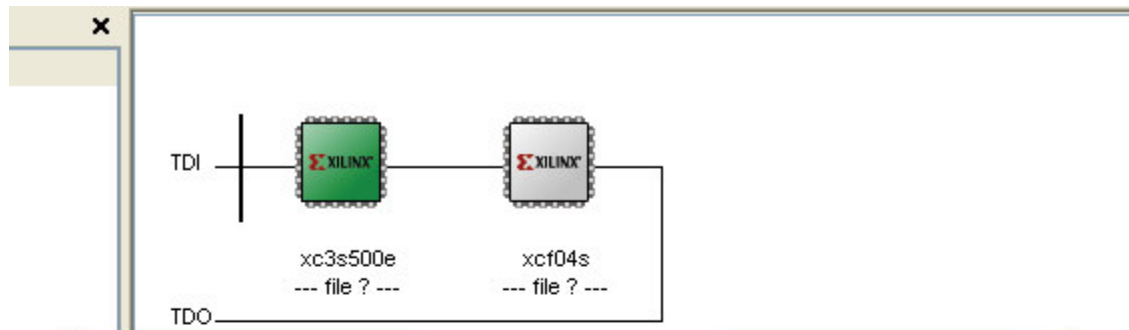
- Select **minoritygate** in *Sources* window and double-click on **Generate Programming File** process to generate the bit file for the design



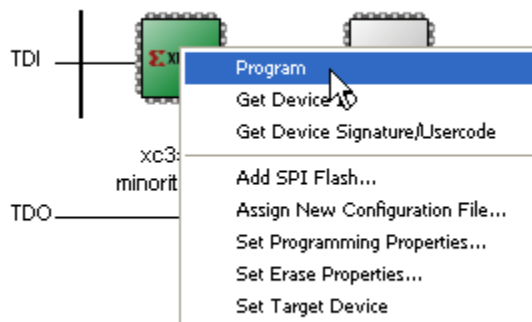
- Expand **Configure Target Device** process and double-click on **Manage Configuration Project (iMPACT)** process



- Connect the board with the USB-JTAG cable
- Power ON the board
- Click Finish to use the JTAG chain
- Select *minoritygate.bit* file to be assigned to xc3s500e device and click **Open**



- Click **Bypass** button for *xcf04s* and then **OK** to use FPGA device programming
- Right-click on the FPGA and select **Program**



- This will program the FPGA and DONE light will lit on the board

- Once programmed successfully, verify the functionality by using SW0 thru SW2 and monitoring LD0 output. Verify that when more than one switch is turned ON, LD0 turns OFF
- Once confirmed the functionality, power down the board and close ISE saving project changes

Conclusion:

In this lab exercise you learned how to design a hierarchical system. You also learned how to model a circuit using gate-level as well as module-instantiations. You were able to simulate the design at each level and then verify the complete design in hardware board.