Lab6

Introduction to Verilog

1. Objective

The main objective of this lab is to give hands on experience with Verilog and the Xilinx Vivado.

2. Pre-requisite

For this lab you are expected to know some basic Verilog programming and understand the Xilinx Vivado. You are advised to go through the Verilog tutorial posted on your course website.

3. Using Xilinx Vivado

The objective is to familiarize you with the Vivado development environment. Please take note of what it is you are doing as you will be expected to perform similar exercises on your own in future experiments.

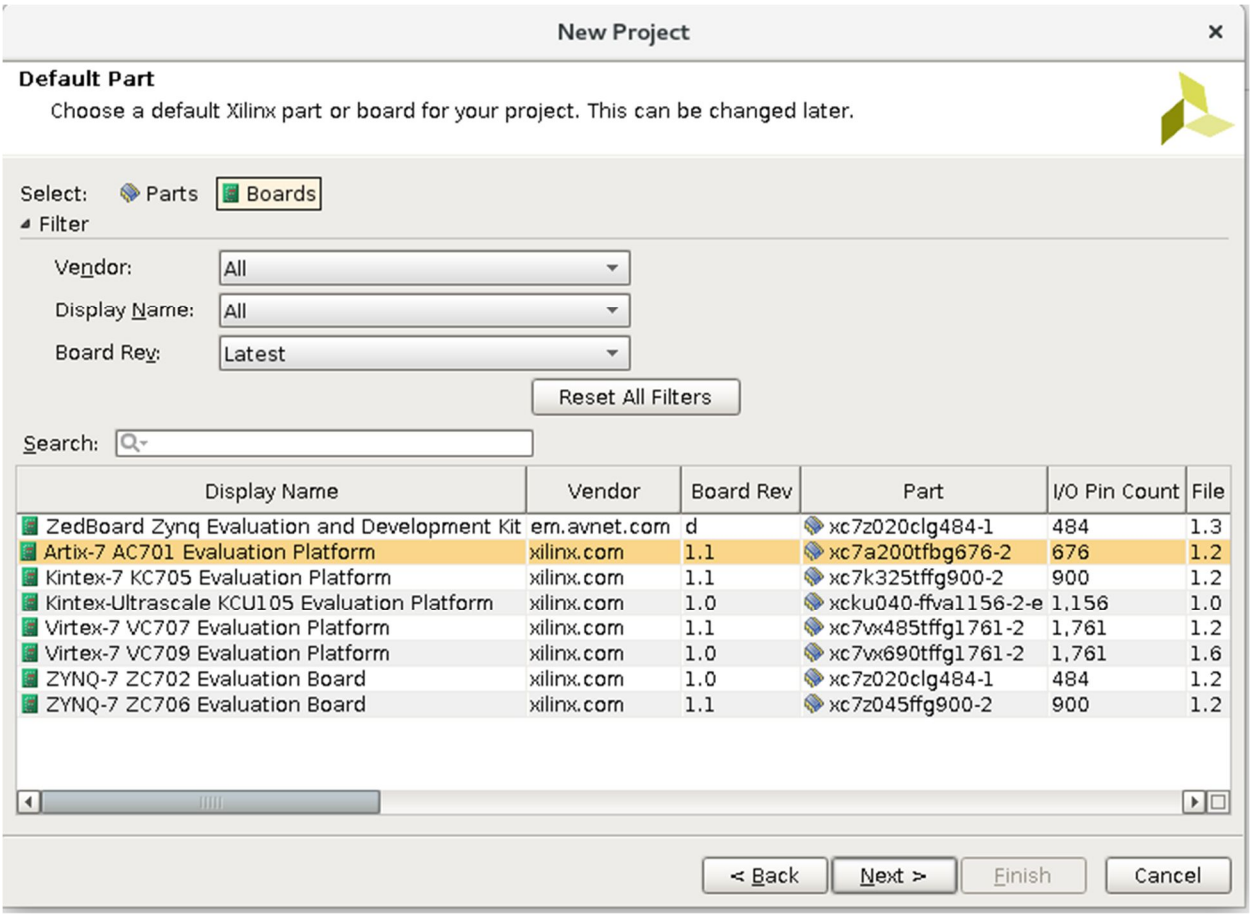
3.1 Launch Vivado and create a new design project.

3.1.1 Open a terminal window and type the following command and hit 'Enter' to create a work directory:
>mkdir $HOME/ecen350

3.1.2 Now execute the following commands to run Vivado:
>source /opt/coe/Xilinx/Vivado/2015.2/settings64.sh
>vivado
The first sets up the environment in order to run Vivado and the second command starts the Vivado Suite.

3.1.3 Once Vivado loads, select **File → New Project**
The New Project Wizard will open. Click 'Next' and a window as illustrated below will open.
Type a Project Name (ex. Project1) and a Project Location (ex. /ecen350/lab6 in your home directory). Click 'Next' and select 'RTL Project'. Click 'Next' and the 'Add Sources' window. Select Target

language as ‘Verilog’ and simulator language as ‘Verilog’. Click ‘Next’ until you reach the ‘Default Part’ window. Select ‘Board’ and select ‘Artix-7 AC701 Evaluation Platform’ as shown below. Click ‘Next’ and select ‘Finish’ to create new project.



3.2 Create a Verilog source file which describes the gates.

3.2.1 From within Vivado, select File → New File to create a new file

3.2.2 A window will appear as seen above. Type the name as ‘Gates.v’ and type the code below into the new file.

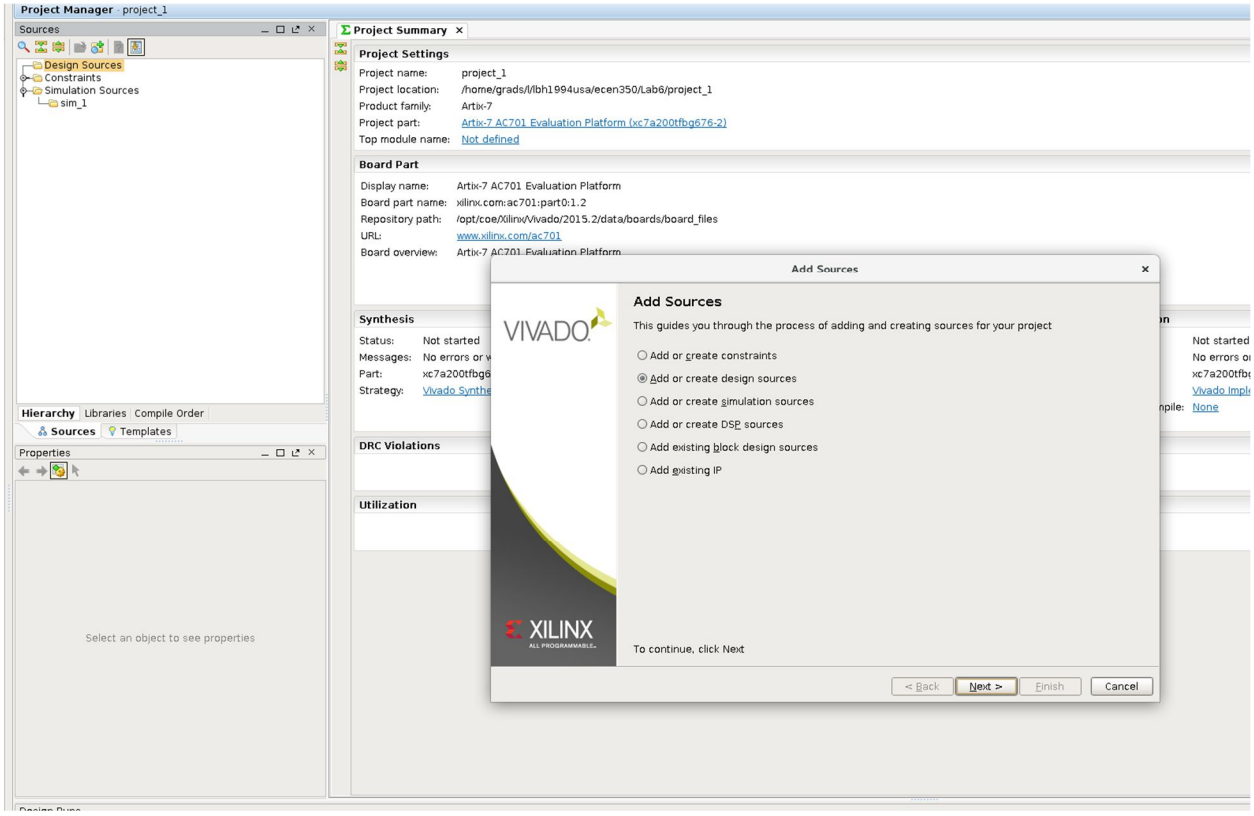
```
module Gates (out, in);
    input [0:1] in;
    output [0:2] out;

    and and0(out[0], in[0], in[1]);
    or or0(out[1], in[0], in[1]);
    xor xor0(out[2], in[0], in[1]);
endmodule
```

This creates a module which has one input comprising 2 bits, and one output comprising 3 bits. The first bit of the output is set to the AND of the two input bits, the second to the OR, and the third to the Exclusive OR.

3.3 Add source file and test bench to the Vivado project.

3.3.1 Right-click on the Design Sources within the Hierarchy window under Sources and select ‘Add Sources..’ as shown in below



3.3.2 In the Add Sources window select ‘Add or create design sources’ and click ‘Next’.

3.3.3 Click on the green ‘+’ button and select ‘Create files’. Click ‘Finish’ to add the source file to the project.

3.3.4 Ensure the source file you just added appears under the Design Sources in the Hierarchy window.

3.3.5 In the Add Sources window select ‘Add or create simulation sources’ and click ‘Next’. Create a new test bench and click ‘Finish’. Add the stimulus below into the test bench file and name it as ‘Gates_tb.v’.

```
#0      in = 0;
#10     in = 2'b01;
#10     in = 2'b10;
#10     in = 2'b11;
```

The test bench utilizes advanced concepts in Verilog that we have not discussed yet, but some things should look familiar. A test bench file is nothing more than a Verilog module which instantiates the Unit Under Test (UUT) and stimulates the inputs for testing. Notice that the test bench module does not have any ports of its own. The input and output ports of the UUT are the focus, and when we simulate the test bench file, we will examine those signals.

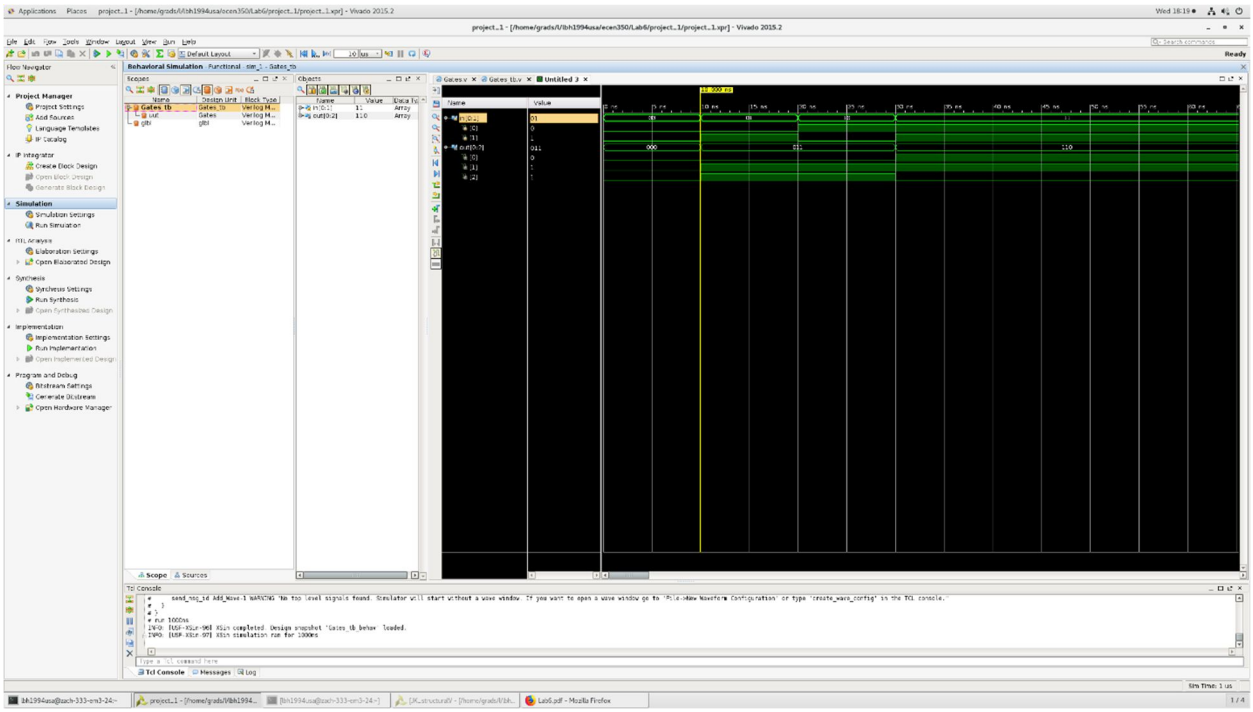
3.4 Simulate the Gates test bench.

3.4.1 To simulate a file using a test bench, right click on the test bench file and select bench file and select 'Set as Top'. Here right click on 'Gates_tb.v' and make it top. Ensure the hierarchy appears as seen in below

3.4.2 Select the Gates test bench file in the hierarchy window and then click 'Run Simulation' from within the Flow Navigator panel. Select 'Run Behavioural Simulation'.

3.4.3 If there are any errors or warnings, they will show up in the Console panel. Correct any errors and warnings at this time. You may re-run the simulation process once you have fixed your source code.

3.4.4 The waveform panel shown below will open once your design successfully compiles. Take note of the waveform panel in the top-right corner and the console panel at the bottom. Notice that the test bench exercises the multiplexer through all of the possible input combinations. Ensure the waveform matches the logic. Please include a screenshot of the waveform in your lab report along with the console output of the test bench.



4. Testing Verilog Designs

For all modules listed below:

- Write down a Verilog implementation.
- Test your code using the provided testbenches and verify that your code is correct.
- Demonstrate the correct functionality to the TA.

4.1 JK-flip-flop

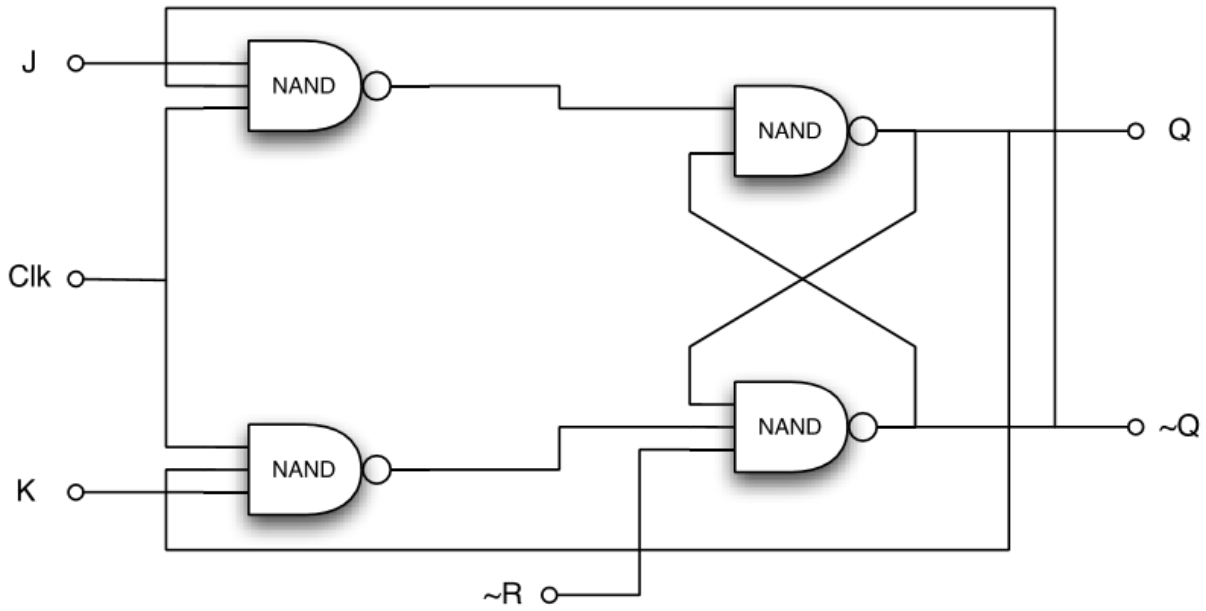


Fig. 5 JK-Flip-Flop With Reset

Use the structural model designed in the prelab with the following module definition. Give all your gates a delay of 2 and make sure that if reset is set to 1, the output is 0. See the designed in Fig. 5 to see how to implement the reset.

module JK(out, j, k, ck, reset);
input j, k, clk, reset;
output out;

4.2 2-to-1 MUX designed (structural model)

Use the structural model designed in the prelab with the following module definition:

module Mux21(out, in, sel);
input [1:0] in;
input sel;
output out;

4.3 2-to-4 decoder

Use behavioral model with the following module definition:

module Decode24(out, in);
input [1:0] in;
output [3:0] out;

5. Deliverables

At the end of the lab you are supposed to turn in:

- Your Verilog HDL for each module.
- Screenshot of the waveform for each module.