ISTANBUL TECHNICAL UNIVERSITY

Digital System Design Applications

Experiment I
SSI COMPONENTS

2024

Preliminary

Students should know Xilinx Vivado Tutorial(Logic Simulation-Synthesis-Implementation-Using Constraints). These tutorials is going to teach you how to create an Vivado project, simulate and implement your HDL-based design, and configure your FPGA. Make sure that you understand and learn every concept given in Xilinx Vivado Tutorial.

Objectives

- Become familiar with Xilinx Vivado
- Understand FPGA Design Steps
- Create a library which contains SSI Components
- Use Verilog primitive gates

Requirements

Students are expected to be able to

- define hardwares with Verilog
- create project on Vivado
- synthesize, simulate, implement designs

Experiment Report Checklist

Each student is going to prepare his/her own report. Reports should include:

1. **AND Gate**
   - Verilog code,testbench code and behavioral simulation wave screenshots.
   - RTL Schematic.
   - Technology Schematic.
   - Add **Synthesis Report**
     - Truth Table of the LUT,
     - Usage of the FPGA resources (utilization summary),
     - Combinational path delays,
     - Maximum combinational path delay,
   - Add **Post-synthesis simulation model** (file).
   - Add **Implementation Report**:
     - Usage of the FPGA resources (utilization summary),
     - Combinational path delays,
     - Maximum combinational path delay,
     - comparison of the path delays generated in the synthesis step,
2. **Other Gates**
   - Verilog codes, testbench codes and behavioral simulation wave screenshots.
   - RTL Schematic.
   - Technology Schematic.
3. **Research**
   - Research about **basic resources (primitives)** of an FPGA.
   - Research about **Look-up Tables (LUTs)**.
   - Research about **glitches and hazards** in digital logic design.

- **Projects and reports are to be done INDIVIDUALLY. High amounts of points will be deducted from similar works.**
- **Reports must be written in a proper manner. Divide your text to sections and subsections if needed, label your figures and connect your sections with proper explanations of your works. Reports filled with imprecisely placed tables and figures, with no verbal explanations in workflow, will not be graded well.**
- **Check homeworks section in Ninova for submission dates.**

Creating SSI Library

1. Create a new Vivado project with the Project Type: **RTL Project**, Target Language: **Verilog** and Simulator Language: **Mixed**. After that, select your FPGA part:
   - Family: **Artix-7**
   - Package: **csg324**
   - Speed: **-1**
   - Part: **xc7a100tcsg324-1** or **xc7a50tcsg324-1**

   Create a new design resource with a Verilog module named **SSI_ Library** to your project.
2. Another method is adding FPGA board files **(Nexys4 DDR or Nexsys A7)**:
   - Download and extract the Zip Archive which is shared from the link below https://digilent.com/reference/programmable-logic/guides/install-board-files
   - Copy the files in this directory "vivado-board-master>new>boards-files"
   - Paste the files to C:Xilinx>Vivado>2024.1>data>boards>board-files
   - Restart Vivado
   - You are now ready to select your Digilent board for the Vivado project.

AND Gate for SSI Library

1. Write down a module which is called **AND** into your **SSI_ Library.v** file. This module is going to have 1-bit inputs **I1**, **I2** and 1-bit output **O**. This module should behave like an **AND gate**. Obtain this behaviour using only **logic operators**.
2. Create a **testbench** for your design and then run **Behavioral Simulation**.
3. Synthesize your design if there is no problem.
4. Investigate your **Synthesized Design**, and add the following details to your report:
   - Show the Truth Table of the LUT2 from the netlist.
   - Utilization Report - generate utilization report and show the consumption of the FPGA resources (utilization summary).
   - Timing Report - show each combinational path delays (generate Timing Report), find maximum combinational path delay.
5. View both **RTL** (obtain from RTL Analysis part) and **Technology** schematics (obtain from Synthesis part) of your design.
6. Verify and explain that **Technology** schematic and **RTL** schematic have the same functionality (AND gate).
7. Run **Post Synthesis Timing Simulation**. Find generated **Post-Synthesis Simulation** file from the sim directory of your project folder and add it to your report.
8. Add **Constraint File** to your design. Uncomment the **Switches** and **LEDs** lines in the **Constraint File**.
9. Add a new Verilog module named **Top_ Module** to your project. This module is going to have 16-bit input **IN** and 8-bit output **OUT**.

10. Add an **AND** module with instance name **AND_GATE** to your **Top_ Module**. Connect IN[0] to I1, IN[1] to I2, and OUT[0] to O.

11. Change the name of the ports in constraint file. Assign IN[0] to SW[0], IN[1] to SW[1], O[0] to LED[0].

12. Synthesize and implement your **Top_ Module**.

13. **Generate Bitstream** and program your FPGA and show the design works as expected.

Other Gates for SSI Library

1. Write down another module which is called **OR** into your **SSI_ Library.v** file. This module is also have 1-bit inputs **I1**, **I2** and 1-bit output **O**. Obtain the **OR gate** behaviour using **logic operators**.

2. Write down a **NOT** gate using **logic operators** to your **SSI_ Library.v** file. This module have 1-bit input **I**, and 1-bit output **O**.

3. Write down a **NAND** gate using an **always block** to your **SSI_ Library.v** file. This module have 1-bit inputs **I1**, **I2** and 1-bit output **O**. Remember that blocking assignments are used for combinatorial logic.

4. Write down a **NOR** gate using an **always block** as in the NAND gate.

5. Write down a **EXOR** gate into your **SSI_ Library.v** file. This module have 1-bit inputs **I1**, **I2** and 1-bit output **O**. Obtain EXOR behaviour using a **LUT2 primitive**. **LUT2 primitive** is a Xilinx pre-defined primitive that directly implements a LUT2 by instantiation.

6. Write down a **EXNOR** gate using a **LUT2 primitive** as in the EXOR gate.

7. Write down the final module which is called **TRI** into your **SSI_ Library.v** file. This module is going to have 1-bit inputs **I**, **E** and 1-bit output **O**. When E input is equal to 1, O output is going to have the same value as input I. When E input is equal to 0, O output will be high impedance (Z). Obtain this behaviour using **conditional operator (?)**.

8. Add instances for all of the gates above to your **Top_ Module**.

9. Make the connections for these instances in your **Top_ Module**, e.g. for your **OR_Gate** instance assign IN[2] to I1, IN[3] to I2 and OUT[1] to O and then for your **NOT_Gate** instance assign IN[4] to I and OUT[2] to O and similar for the other instances.

10. Change the name of the ports in constraint file. Assign IN[15:0] to SW[15:0] and OUT[7:0] to LED[7:0].

11. Create a **testbench** for your **Top_ Module** and then make a **Behavioral Simulation**.

12. Synthesize and implement your **Top_ Module** if there is no problem.

13. View both **RTL** (obtain from RTL Analysis part) and **Technology** schematics (obtain from Synthesis part) of the **Top_ Module**.

14. **Generate Bitstream** and program your FPGA and show the design works as expected.

Research

1. Make a research about **basic resources** (primitives, such as look-up tables) of an FPGA. Explain the functionalities of these primitives briefly.
2. Make a research about **LUTs (look-up tables)**, show their logic diagram and explain how they operate with an example.
3. Make a research about the **glitches and hazards** in digital logic design. Explain the types of hazards with examples, and how to avoid them.

References:

1. Nexys4DDR Reference Manual
2. Artix-7 Libraries Guide for HDL Designs
3. Constraints Guide
4. Brown&Vrasenic, "Fundamentals of Digital Logic with Verilog Design", McGraw-Hill, p.17-47, 87-107