ISTANBUL TECHNICAL UNIVERSITY

Digital System Design Applications

Experiment I
SSI COMPONENTS

Preliminary

Students should cover chapters 3, 5, 6, 7 and 8 from ISE In-Depth Tutorial. Skip VHDL, Synplify and Modelsim related topics. This tutorial is going to teach you how to create an ISE project, simulate and implement your HDL-based design, and configure your FPGA. Make sure that you understand and learn every concept given in ISE In-Depth Tutorial.

Objectives

- Become familiar with Xilinx ISE Software
- Create a library which contains SSI Components

Requirements

Students are expected to be able to

- define hardwares with Verilog
- create project on ISE
- synthesize, simulate, implement designs, generate bitstreams and configure FPGA

References:

1. Nexys2 Reference Manual
2. Spartan-3E 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

Creating SSI Library

Create a new ISE project with the settings below:

- Family: **Spartan 3E**
- Device: **XC3S500E**
- Package: **FG320**
- Speed: **-4**
- Synthesis Tool: **XST (VHDL\Verilog)**
- Simulator: **ISim (VHDL\Verilog)**
- Preferred Language: **Verilog**

Add a new Verilog module named **SSI_ Library** to your project. Clear all lines in your module.

AND Gate (60 Minutes)

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. Check syntax.
3. Synthesize your design if there is no problem with syntax.
4. Investigate **Synthesis Report**, and add the following informations to your report:
   - Low Level Synthesis - area consumption
   - Final Report - design statistics, cell usage, device utilization summary, maximum combinational path delay
   - Timing Detail - discuss each path delay.
5. View both **RTL** and **Technology** schematic of your module.
6. Verify that schematic of LUT2 in Technology schematic and RTL schematic are same, and they are both and gates. Add the followings to your report:
   - **Truth table** and **Karnaugh map** of LUT2
7. Generate **Post-Synthesis Simulation Model**.
8. Investigate the generated Verilog module, add the following informations to your report:
   - Primitives and their functions
   - Initial value of LUT primitive, and the meaning of it
9. **Translate** your design.
10. Generate **Post-Translate Simulation Model**.
11. Investigate the generated Veriolog module, and compare it with the one generated in the 8th step. Add your findings to your report.

12. **Map** your design.
13. Investigate **Map Report**, and add the following informations to your report:
    - Number of 4 input LUTs
    - Number of bonded IOBs
    - Removed Logic
    - IOB properties
14. Generate **Post-Map Static Timing Report**. Investigate it, and add the following to your report:
    - Delays from source pads to destination pad.
15. Generate **Post-Map Simulation Model**.
16. Compare **Post-Translate Simulation Model** and **Post-Map Simulation Model**, and add the following to your report:
    - Number of inputs of LUT
    - Input signals of LUT
    - Initial value of LUT
17. **Place & Route** your design.
18. Investigate **Place and Route Report** and **Post-PAR Static Timing Report**, add followings to your report:
    - Number of external IOBs
    - Number of external input IOBs
    - Number of external output IOBs
    - Delays from source pads to destination pad
19. Run **FPGA Editor**, and add the followings to your report:
    - SLICE that contains your module and its coordinates
    - Internal view of this SLICE
    - Logical function of LUT which is inside this SLICE
    - RPM grids of inputs and output
20. Generate **Post-Place & Route Simulation Model**.
21. Investigate the generated Verilog module, and add the followings to your report:
    - Input signals of LUT
    - Initial value of LUT
    - LOC information of primitives
22. Generate **Programming File**. You can program your FPGA with the generated file, but you will not be able to test if it works or not. Discuss why?
23. Add a new Verilog module named **Top_ Module** to your project. This module is going to have 1-bit inputs **A**, **B**, and 1-bit ouput **C**.

24. Add an **AND** module with instance name **GATE** to your **Top_ Module**. Connect A to I1, B to I2, and C to O.
25. Add a new UCF file(Implementation Constraints File) named **constraints**. Using this UCF file, connect **A** input to **SW1**, **B** input to **SW0**, and **C** output to **LD0** on your board.
26. Generate programming file, and program your FPGA. Observe your design on the FPGA.

OR Gate (15 Minutes)

1. Write down another module which is called **OR** 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 **or gate**. Obtain this behaviour using only **logic operators**.
2. Observe **OR** module is now present in hierarchy.
3. Change **AND** instance to **OR** in **Top_ Module.v**.
4. Synthesize **Top_ Module**, view its RTL Schematic, and add it to your report.
5. View Technology schematic, and add followings to your report:
   - Truth table and Karnaugh map of LUT
6. **Place & Route** your design.
7. Run **FPGA Editor**, and add the design view to your report.
8. Generate programming file, and program your FPGA. Observe your design on the FPGA.

NOT Gate (15 Minutes)

1. Write down another module which is called **NOT** into your **SSI_ Library.v** file. This module is going to have 1-bit input **I**, and 1-bit output **O**. This module should behave like an **not gate**. Obtain this behaviour using only **logic operators**.
2. Change **OR** instance to **NOT** in **Top_ Module.v**. Make proper connections!
3. Synthesize **Top_ Module**, and add Technology Schematic to your report. Observe that your design does not contain LUT.
4. **Place & Route** your design.
5. Run **FPGA Editor**, and observe that your design does not contain SLICE. Add the design view to your report.
6. Generate programming file, and program your FPGA. Observe your design on the FPGA.

NAND Gate (15 Minutes)

1. Write down another module which is called **NAND** 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 **nand gate**. Obtain this behaviour using **always** block.
2. Change **NOT** instance to **NAND** in **Top_ Module.v**. Make proper connections!

3. Synthesize **Top_ Module**, and add the followings to your report:
   - Technology Schematic
   - Truth table and karnaugh map
4. **Place & Route** your design.
5. Run **FPGA Editor**, and compare it with the **OR** gate.
6. Generate programming file, and program your FPGA. Observe your design on the FPGA.

NOR Gate (10 Minutes)

1. Write down another module which is called **NOR** 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 **nor gate**. Obtain this behaviour using **always** block. Repeat the steps from NAND gate.

EXOR Gate (20 Minutes)

1. Write down another module which is called **EXOR** 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 **exor gate**. Obtain this behaviour using **LUT2 primitive**. Repeat the steps from NAND gate.

EXNOR Gate (10 Minutes)

1. Write down another module which is called **EXNOR** 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 **exnor gate**. Obtain this behaviour using **LUT2 primitive**. Repeat the steps from NAND gate.

TRI Component (20 Minutes)

1. Write down another 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 gong to have the same value as input I. When E input is equal to 0, O output will be high impedance Obtain this behaviour using **?** operator. Repeat the steps from NAND gate.

Ring Oscillator (60 Minutes)

1. Add a new Verilog module named **ring_oscillator**, and assign it as a top module. This module is going to have 1-bit inputs **A**, **B**, 1-bit output **C**, and 200-bit wire **W**. Cascade 199 inverters (NOT gates in your SSI_Library) using **generate** block.
2. Add a **TRI** module to **ring_oscillator.v**. Connect **I** input to wire **W[199]**, and **O** output to wire **W[0]**. E girisinin nereye baglanacagi belli degil.
3. Connect **W[10]** to output **C**.

4. Synthesize your design, and view Technology schematic, and investigate Design Summary. How many LUTs does your design use?
5. Use **KEEP** constraint so that wires do not reduced.
6. Synthesize your design, and view Technology schematic, and investigate Design Summary. How many LUTs does your design use? Compare it with the result obtained in step 4, and discuss why different results are obtained. Add your results and opinions to your report.
7. **Place & Route** your design, and determine the delay between **A** and **C**.
8. Run **FPGA Editor**, and add the design view to your report.
9. Generate programming file, and program your FPGA. Observe the output of ring oscillator on the oscilloscope. Determine the frequency of output signal, and add it to your report.

Experiment Report

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

- Screenshots and results asked above
- Your comments on results
- Definition of **fan-out**

Reports are to be submitted before next experiment. Submission includes reports and project files.