ISTANBUL TECHNICAL UNIVERSITY

Digital System Design Applications

Experiment 2
MSI COMPONENTS

2024

Preliminary

Students should know basic MSI elements needed for combinatorial circuits. Basic MSI elements can be listed as following: Decoders, Encoders, Priority Encoders, Multiplexers. For the experiment, students should also know simplifying methods for Combinatorial Circuits like Karnaugh Maps.

Objectives

- Create a library which contains MSI Components using SSI library generated in the previous experiment.
- Implement the MSI elements on the FPGA board using LEDs and 7-Segment Display.

Requirements

Students are expected to be able to

- define hardwares with Verilog
- create project on Vivado
- synthesize, simulate, implement designs, generate bitstreams and configure FPGA
- write truth tables of all MSI elements
- simplify combinatorial circuits by karnaugh maps.

Experiment Report Checklist

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

1. **DECODER**
   - DECODER Verilog code, testbench code and behavioral simulation wave screenshots.
   - RTL and Technology schematic of DECODER. How many LUTs are there in the Technology schematic? What are the logic statements of the LUTs? How do these logic statements implement the decoding operation.
   - Timing report and the **greatest delay** of implemented DECODER design.
2. **PRIORITY ENCODER**
   - Truth table of Priority Encoder and simplified logic statements of outputs OUT[0], OUT[1] and V using Karnaugh Map
   - Hand drawn schematic of priority encoder.
   - ENCODER Verilog code, testbench code and behavioral simulation wave screenshots.
   - RTL and Technology schematic of ENCODER. Add the difference between the RTL and Technology schematic of ENCODER regarding to counts of the components (LUTs, gates).
   - Verilog code of the ENCODER using **always** and **case** structure. Add the comparison of the implementation results (timing and space) for both structural and behavioral designs.
3. **MULTIPLEXER**
   - MULTIPLEXER Verilog code, testbench code and behavioral simulation wave screenshots.
   - RTL and Technology schematic of MUX.
   - Verilog code of the MUX using **always** and **case** structure. Add the comparison of the implementation results (timing,space,etc.) for both structural and behavioral designs.
4. **DEMULTIPLEXER**
   - DEMULTIPLEXER Verilog code, testbench code and behavioral simulation wave screenshots.
   - RTL and Technology schematics of DEMUX.
5. **Research**
   - Research about how to avoid inferring **latches**.
   - Research about **Leading Zero Counter** circuits.
   - Research about **fan-in** and **fan-out** concepts 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 MSI 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 **MSI_ 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.

Decoder

1. Write the truth table of a **4x16 Decoder** and add it your report.
2. Write down a module which is called **DECODER** into your **MSI_Library.v** file. This module is going to have 4-bit input **IN** and 16-bit output **OUT**. Obtain the **DECODER** behaviour using an **always** and **case structure**.
3. Create a new file named **top_module.v** and set this file as top module. Please look at the **Basic I/O part in the User Manual** of your FPGA board.
4. Add the inputs to your top module given below
   8-bit **sw** (switches)
   4-bit **btn** (buttons).
5. Add the outputs to your top module given below
   8-bit **led** (LEDs)
   7-bit **cat** (cathodes of the seven-segment display)
   4-bit **an** (anodes of the seven-segment display)
   1-bit **dp** (dot point of the seven-segment display)
6. Add the master constraint file of your board to your project.
7. Instantiate the **DECODER** module with the name **decoder1** in the **top module**.

8. Connect the least significant 4-bits of input **sw** to the input **IN** of the **decoder1**. Connect the outputs **dp**,**cat** and **led** to the output **OUT** of the **decoder1**, respectively.
most significant(dp) → least significant(led)

9. Connect least significant bit of the **an** to logic 0 and other digits to logic 1.

10. Create a testbench for your design then make **behavioral simulation**.

11. Synthesize your design and add the following informations to your report:
    - Verilog code, testbench code and behavioral simulation wave screenshots.
    - RTL and Technology schematic.
    - How many LUTs are there in the Technology schematic? What are the logic statements of the LUTs and how does these logic statements implement the decoding operation? ? Just explain.

12. Implement your design. Add the timing results and find the **greatest delay** of the decoder.

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

Priority Encoder

1. Write the truth table of a **4 to 2 Priority Encoder with an Valid output**, and add it to your report.

2. Simplify the logic statements of the Outputs (OUT[0], OUT[1]) and Valid (V) using Karnaugh Map and add the results to your report.

3. Draw the logic diagram of the reduced statements with hand, and add it to your report.

4. Write down a module which is called **ENCODER** into your **MSI_Library.v** file. This module is going to have 4-bit input **IN**, 2-bit output **OUT** and 1-bit output **V**. Use your logic diagram of the reduced statements to obtain **ENCODER** behavior. Write a structural code using Primitive Gates.

5. Replace the **DECODER** in the **top_ module** with the **ENCODER**.

6. Connect the least significant 4-bits of input **sw** to the input **IN** of the ENCODER. Connect the least significant 2-bit of the output **led** to output **OUT** and the most significant bit of the output **led** to the output **V** of the ENCODER.

7. Create a testbench for your design then make **behavioral simulation**.

8. Synthesize your design and add the following information to your report:
    - Verilog code, testbench code and behavioral simulation wave screenshots.
    - RTL and Technology schematic.
    - Difference between the RTL and Technology schematic, regarding to counts of the components(LUTs,gates).

9. Implement your **top_ module** and add the implementation results (timing and utilization) to your report.

10. Write a new Verilog code for the **ENCODER** using an **always** and **case** structure.

11. Synthesize and Implement your new design.
12. Add the implementation results (timing and space) and make a comparison between structural and behavioral designs.
13. **Generate Bitstream** and program your FPGA and show one of your ENCODER design (structural or behavioral) works as expected.

Multiplexer

1. Write down another module which is called **MUX** into your **MSI_Library.v** file. This module is going to have 4-bit input **D**, 2-bit input **S**, and 1-bit output **O**. Obtain the **MULTIPLEXER** behaviour using **assign** and **logic operators**.
2. Replace the **ENCODER** in the **top_ module** with the **MUX**.
3. Connect the least significant 4-bits of input **sw** to the input **D** of the MUX. Connect the least significant 2-bit of the input **btn** to input **S** of the MUX. Connect least significant bit of the **led** to the output **O** of the MUX.
4. Create a testbench for your design then make **behavioral simulation**.
5. Synthesize your design and add the following informations to your report:
   - Verilog code, testbench code and behavioral simulation wave screenshots.
   - RTL and Technology schematic.
6. Implement your **top_ module** and add the implementation results (timing and utilization) to your report.
7. Write a new Verilog code for the MUX using **always** and **case** structure.
8. Synthesize and Implement your new design.
9. Add the comparison of the implementation results (timing and space) for both dataflow and behavioral designs.
10. **Generate Bitstream** and program your FPGA and show one of your MUX design works as expected.

Demultiplexer

1. Write down another module which is called **DEMUX** into your **MSI_Library.v** file. This module is going to have 1-bit input **D**, 2-bit input **S** and 4-bit output **O**. Obtain the **DEMULTIPLEXER** behaviour using **NOT**, **AND** and **TRI** gates from your SSI_library.
2. Connect the least significant bit of input **sw** to the input **D** of the DEMUX. Connect the least significant 2-bit of the input **btn** to input **S** of the DEMUX. Connect least significant 4-bit of the **led** to the ouput **O** of the DEMUX.
3. Create a testbench for your design then make **behavioral simulation**.
4. Synthesize your design and add the following information to your report:
   - Verilog code, testbench code and behavioral simulation wave screenshots.
   - RTL and Technology schematic.

5. Implement your design.
6. **Generate Bitstream** and program your FPGA and show your design works as expected.

Research

1. Make a research about why the **latches** should not be used and how to avoid inferring **latches** in FPGA designs.
2. Make a research about the **Leading Zero Counter** circuits and how to design a **Leading Zero Counter** circuit using priority encoders.
3. Make a research about **fan-in** and **fan-out** concepts in digital logic design.

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.297-319, 149-201