Lab 2
Combinational Circuit Design using Gate-Level Modeling and Synthesis on FPGA

Module ID : CX-203

Digital System Design

Instructor : Dr. Abid Rafique

Version 1.2

Document History

The changes and versions of the document are outlined below:

Version	State / Changes	Date	Author
1.0	Initial Draft	Dec, 2023	Qamar Moavia
1.1	Modified with new exercises	August, 2024	Dr. Abid Rafique Qamar Moavia

Table of Contents

Objectives

By the end of this lab, students will:

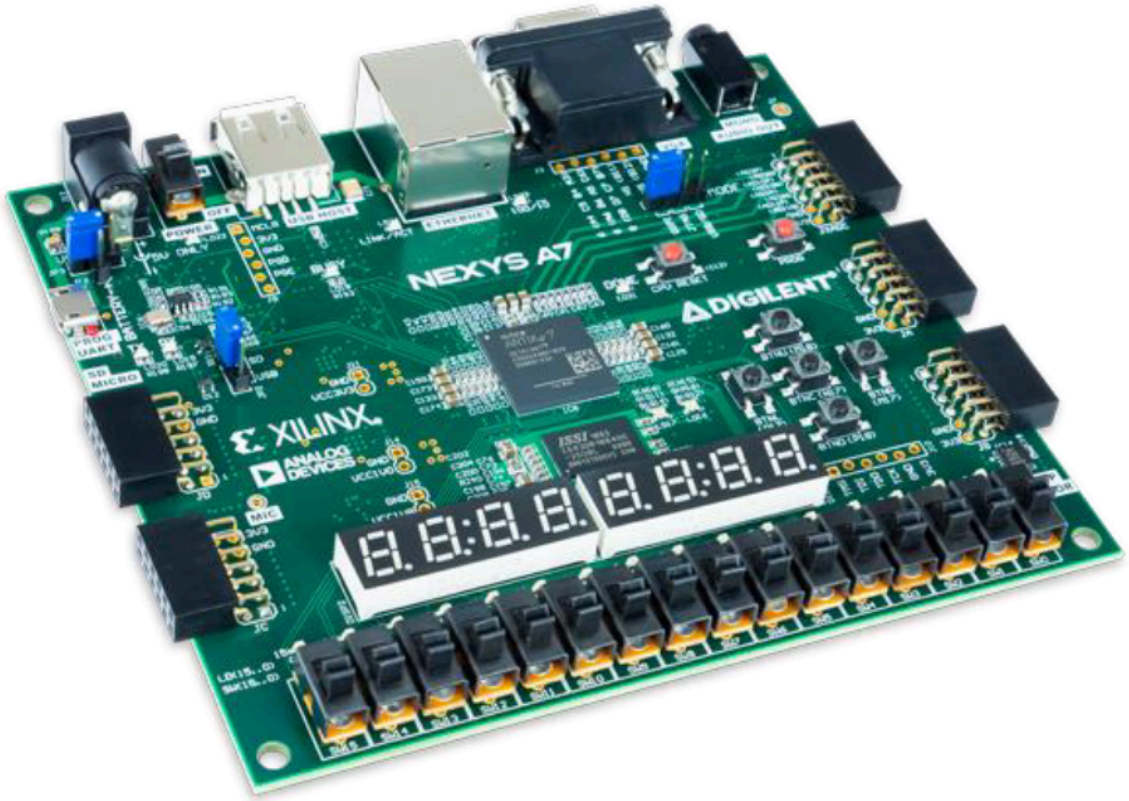
- Learn the process of synthesizing a design to an FPGA in Vivado.
- Understand pin assignment and constraints in FPGA design.
- Implementing basic designs on FPGA.

Deliverables

A single PDF document containing the following:

- Source code (code or screenshots)
- Simulation results (waveforms) screenshots
- Link to git repository
- FPGA Implementation results (multiple images for each task)

1. FPGA Implementation and Pin Assignment



Once you've modeled your circuit using gate-level modeling and simulated it as well, then the next step is to implement it on an FPGA. This involves:

1. **Synthesis**:

   Convert your SystemVerilog design into a gate-level netlist that can be mapped onto the FPGA. In Vivado, after writing your SystemVerilog code, you can synthesize your design by selecting the "Synthesis" option. This step checks for errors and generates a detailed report on the design.

2. **Pin Assignment**:

   For the FPGA to interact with external components, you must map your design's inputs and outputs to specific physical pins on the FPGA board. This is done through pin assignment:

   - Open the **I/O Planning** feature in Vivado.
   - Identify the pins on your FPGA board (e.g., switches, LEDs, buttons).
   - Assign the inputs and outputs of your Verilog module to these FPGA pins by editing the constraints file (.xdc file).

   For example, if **a** and **b** are inputs connected to switches, and **f** is the output connected to an LED, you need to define these connections in the **.xdc** file.

   Here is a part of pin assignment file for the Nexys A7 FPGA board:

```
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { i_sw[0] }]

set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { i_sw[1] }]

.

set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { o_led[0] }]

set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { o_led[1] }]
```

   In this example:

   - J15 is connected to i_sw[0] on the Nexys A7 board, which can be used as input to your top level design.
   - Similarly, H17 is connected to an LED (o_led[0]) to display the output.

   You can rename the signals in the constraints file to match your Verilog module for clarity. For instance, rename i_sw[0] to a and i_sw[1] to b and so on. This way, when you reference a in your top module,

   Vivado will automatically associate it with the correct physical pin on the FPGA board.

3. **Run the Implementation Phase**

- After you have assigned the FPGA's physical pins to the inputs and outputs in your design, start the Implementation phase in Vivado.
- During this phase, Vivado verifies that your design can be correctly mapped onto the FPGA's architecture, making sure it fits the device's hardware resources and follows any timing or design constraints you set.
- If any issues arise (such as resource conflicts or timing problems), Vivado will notify you, and you may need to adjust your design or constraints.

4. **Generate the Bitstream File**

- Once your design passes the Implementation phase, the next step is to create a bitstream file. This file is essential because it contains the binary data needed to configure the FPGA to perform your specific design.
- To generate it, select the Bitstream Generation option in Vivado. This process compiles your implemented design into a configuration file compatible with the FPGA hardware.

5. **Program the FPGA**

- After generating the bitstream file, the final step is to load the design onto the FPGA.
- Use Vivado's Hardware Manager to program the FPGA. Connect your FPGA board to the computer, and ensure that Vivado detects it. Then, select the bitstream file you generated and program the FPGA.
- Once programming is complete, your design will be active on the FPGA, and you can test its functionality using the inputs and outputs you assigned.

2. Implementing 2-bit AND Gate on FPGA

This lab will guide you through the steps required to implement your 2-bit AND gate design on an FPGA, building upon the project created in Lab 1. If you haven't completed Lab 1, please follow those steps to create and simulate a 2-bit AND gate project in Vivado before proceeding.

Step 1: Open Your Vivado Project

1. Launch Vivado and open the 2-bit AND gate project from Lab 1.
2. Ensure that you have a working SystemVerilog file (**andgate.sv**) and a testbench (**tb_andgate.sv**) in your project.
3. In case you didn't create a separate project for simulation of 2-bit AND gate, then create a project first.

Step 2: Synthesis

1. In the Vivado Flow Navigator, locate and click on Synthesis under Project Manager.
2. Vivado will check your design for any errors. After synthesis completes, review the Synthesis Report to verify your design.
    - Tip: This report provides valuable feedback on the resource usage and any warnings in the design.

Step 3: Pin Assignment

1. Access the Constraints File:
    - Open the **.xdc** constraints file provided in your lab materials.
    - The file contains commented lines specifying connections to physical pins on the FPGA board.
2. Uncomment the Required Pins:
    - For this AND gate example, we'll use two switches and one LED:
        - Locate the lines in the **.xdc** file that specify **i_sw[0]** and **i_sw[1]**:

```
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { i_sw[0] }]

set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { i_sw[1] }]
```

Uncomment these lines and rename **`i_sw[0]`** to **a** and **`i_sw[1]`** to **b** to match your SystemVerilog design.

Next, find the line that specifies the **output LED (`o_led[0]`)**:

```
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { o_led[0] }]
```

        - Uncomment this line and rename **`o_led[0]`** to **f** to match your SystemVerilog design.
    - Save the **`.xdc`** file to apply these pin assignments.

Step 4: Implementation

1. In the Flow Navigator, select **Run Implementation**.
    - During implementation, Vivado will ensure that the design fits within the FPGA's architecture and adheres to timing constraints.
2. If any issues are reported, review them and make necessary adjustments to your design or constraints file.

Step 5: Bitstream Generation

1. Once implementation is successful, select **Generate Bitstream** in the Flow Navigator.
    - Vivado will convert the implemented design into a bitstream file (.bit file) compatible with the FPGA hardware.
2. After the bitstream generation completes, a prompt will confirm success.

Step 6: Programming the FPGA

1. **Connect Your FPGA Board**:
    - Attach your FPGA board to the computer via USB and ensure Vivado detects the device.
2. **Open Hardware Manager**:
    - In Vivado, go to **Open Hardware Manager** and **Open Target** to establish a connection with your FPGA.
3. **Load the Bitstream**:
    - Select **Program Device**, and choose the **.bit** file generated in the previous step. Vivado normally automatically selects the **.bit** file.
    - Click **Program** to load the design onto the FPGA.

Step 7: Test Your Design

1. Use the physical switches (SW0 and SW1) on the FPGA board to set the inputs **a** and **b**.
2. Observe the LED (LD0) to see the output **f**:
    - The LED should light up only when both switches are ON, representing the AND gate functionality.

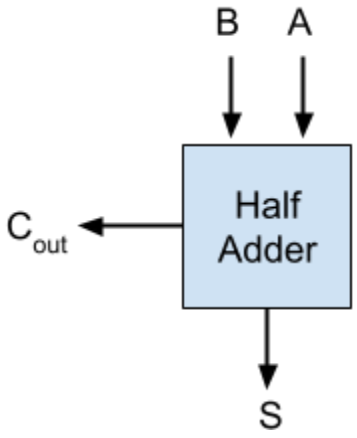
TASK 1 : 4-input AND gate Implementation on FPGA

Remember the 2-input AND gate you designed and simulated in Lab 1.
Now, implement a 4-input AND gate on the FPGA. Follow all the steps we used for the 2-input AND gate. **Use switches SW[3:0] as inputs and LED LD0 as the output.**

TASK 2 : Half Adder Implementation on FPGA

Remember the 2-input AND gate you designed and simulated in Lab 1.
Now, implement a Half Adder on the FPGA. Follow all the steps we used for the 2-input AND gate. **Use switches SW[0] and SW[1] as inputs A and B respectively, with LED LD0 for the Sum output and LED LD1 for the Cout output.**

A	B	S	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

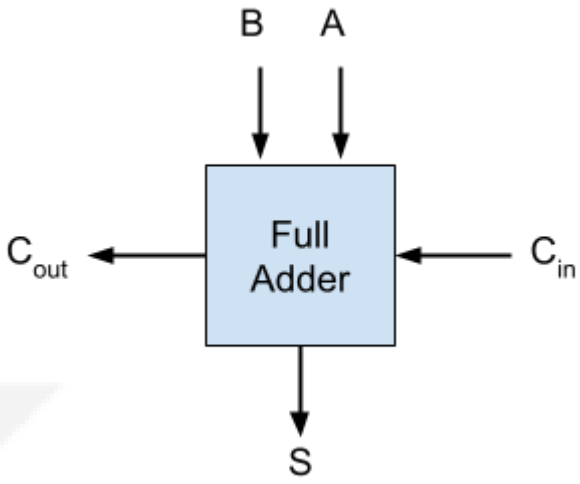


TASK 3 : Full Adder Implementation on FPGA

Implement Full Adder from Lab1 on FPGA.

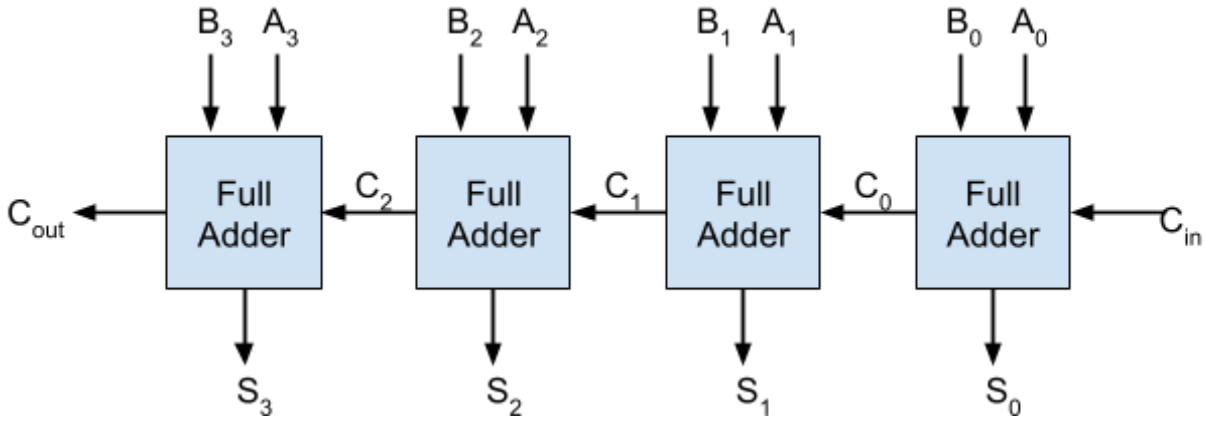
Use switches SW[0], SW[1], and SW[2] as inputs, with LED LD0 for the Sum output and LED LD1 for the Carry output.

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



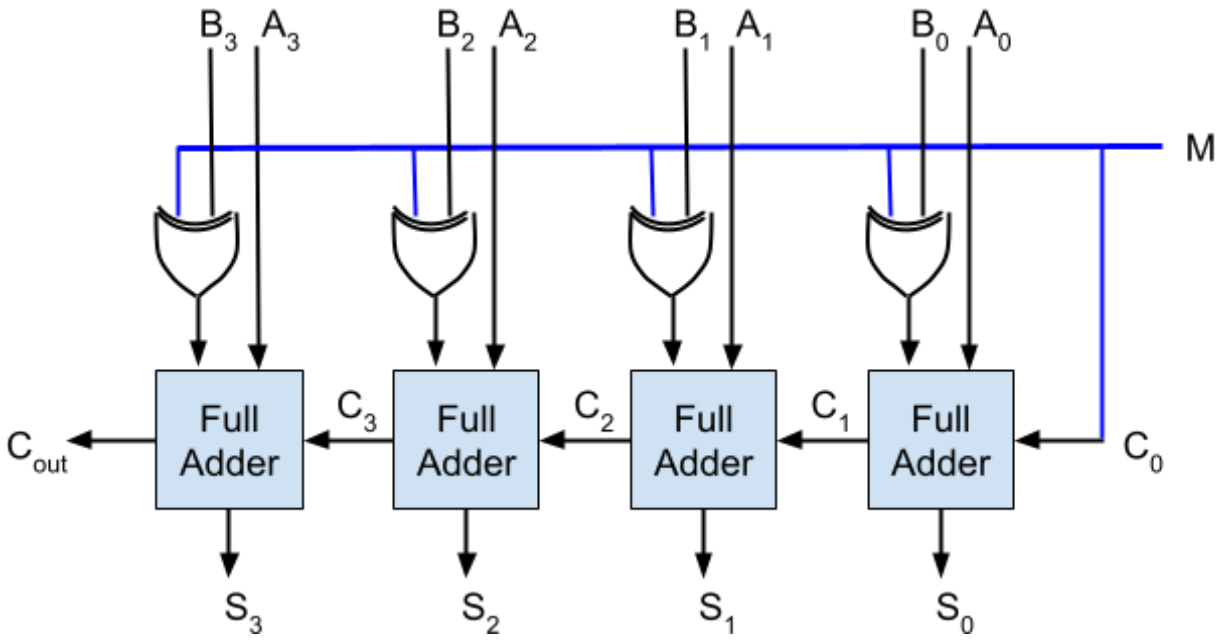
TASK 4 : 4 bit Adder

Design a 4-bit adder using gate level modeling in SystemVerilog. Reuse single-bit full adder instances. Simulate your design by writing a testbench that verifies the adder with at least 10 different test cases. Once the simulation is working then implement the design on the FPGA, using switches SW[7:4] for input A and SW[3:0] for input B, SW[8] for carry input. Use LEDs LD[3:0] for Sum and LD[4] for carry output.



TASK 5 : 4 bit Adder/Subtractor

Design a 4-bit adder/subtractor using gate level modeling in SystemVerilog. Use a control signal M to toggle between addition (M = 0) and subtraction (M = 1). Simulate your design by writing a testbench that verifies the adder with at least 10 different test cases. Once the simulation is working then implement the design on the FPGA, using switches for inputs and LEDs to display the outputs.



Good Luck 🙂