**EE533 NETWORK PROCESSOR DESIGN & PROGRAMMING  
LAB#2: Xilinx ISE 10.1 SCHEMATIC (32-BIT ALU)**

**Instructor: Prof. Young Cho, PhD**

CREATED AND COMPILED BY:  
SARTHAK JAIN

(MS EE, UNIVERSITY OF SOUTHERN CALIFORNIA)

***GITHUB LINK FOR MY REPOSITORY:***  
You’ll find all the codes and the executables on this repository.   
 <https://github.com/SARTHAK-JAIN-ASIC/EE533/tree/main/LAB2>

We first start with a one-bit adder   
Shown for all subsequent circuits is:  
1. Circuit  
2. Testbench waveform  
3. Actual simulation waveform

**1. ONE BIT ADDER**  
A computer screen shot of a drawing

Description automatically generated

A computer screen shot of a computer

Description automatically generated

A computer screen shot of a black screen

Description automatically generated

**2. I then turned this adder into a SYMBOL and utilized it to make an 8-bit combinational adder**

A computer screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

3. Finally, I made use of 8 bit FFs to turn my combinational adder into a synchronous one

A computer screen shot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated4. Next was to turn my 8-bit adder into a 32-bit one. As follows:

A computer screen shot of a computer

Description automatically generated

A computer screen shot of a computer

Description automatically generated

SYMBOL FOR THE SAME:

A screenshot of a computer

Description automatically generated

5. Next I wanted to create a **32-bit adder**/**subtractor** out of the adder that I already had. For that, I just had to utilize a few xor gates (made a symbol out of the intermediate stage) and gave the user an option to decide if the subtractor was required or the adder.

A screenshot of a computer

Description automatically generated

A computer screen shot of a diagram

Description automatically generated

A screenshot of a computer

Description automatically generated

A computer screen shot of a computer

Description automatically generated  
The waveform at the bottom shows the “subtract” signal, toggling addition and subtraction

A screenshot of a video game

Description automatically generated

A diagram of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

6. Next, it was time to make a shifter. For this, I gave a LINE/INPUT to decide if the user wanted to shift the number left or shift it right. First

A computer screen shot of a computer screen

Description automatically generated

This was a very basic unit I used to make my shifter. Created a SYMBOL out of it and utilized that to create a 32-bit shifter.

A computer screen shot of a blueprint

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

The following shows the mapped implementation and the overall LUT and cell usage

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A computer screen shot of a computer screen

Description automatically generated

**VERILOG CODE IMPLEMENTATIONS:**

The following is the waveform I got for the ALU when I wrote the Verilog code for the modules I implemented

A screenshot of a computer

Description automatically generated

A white background with black and white clouds

Description automatically generated

We realize from this that the number of gates in the schematic version is **more** compared to the actual synthesis from the verilog code we created (as here, it the tool that decides what to ‘optimize’). In the schematic, we use explicit DFFs and gates and other miscellaneous logic, while the verilog version uses sophisticated algorithms to optmize the “data paths” in the circuits described using HDL. Therefore corresponds to fewer LUTs and DFFs and logic cells as compared to the schematic version.

FINAL ALU:  
Waveform:

A screenshot of a computer

Description automatically generated

SCHEMATIC:  
A computer screen shot of a computer

Description automatically generated

8-bit 2x1 MUX

A diagram of a circuit

Description automatically generated

32-bit 2x1 MUX:  
A computer screen shot of a diagram

Description automatically generated

FINAL ALU MUX (32-bit 4x1)

A computer screen shot of a computer

Description automatically generated