Experiment 3: Design, Simulation, and Synthesis of an 8-bit ALU

Objective

1. To design a parameterized 8-bit Arithmetic Logic Unit (ALU) in Verilog HDL.
2. To verify its functionality using a suitable testbench.
3. To perform logic synthesis using Synopsys Design Compiler (DC) and analyze area, power, and timing reports.

Background

An Arithmetic Logic Unit (ALU) is a core computational block of any digital processor that performs arithmetic and logical operations on binary data. Typical operations include addition, subtraction, bitwise logic functions (AND, OR, XOR, NOT), and shift/rotate operations.

The ALU plays a vital role in CPUs, digital signal processors (DSPs), and hardware accelerators, executing the arithmetic and logical computations required in instruction-level or algorithmic processing.

In this experiment, you will design both behavioural and structural versions of an ALU, verify them using simulation, and then synthesize them using the Synopsys Design Compiler with an SCL 180 nm standard-cell library to compare their implementation results.

Tools and Library Used

- Synopsys Design Compiler (DC)
- SCL 180 nm CMOS Standard Cell Library

Experimental Procedure

Part 1 – Behavioural ALU Design

1. Design a Parameterized ALU
    - Create a Verilog module with the following specifications:
        - **Inputs**: Two $n-$bit operands A, B, and a 3-bit operation code op.
        - **Outputs**: Result Y, carry flag (CF), and zero flag (ZF).
2. The ALU should support at least the following operations:

Operation Code	Operation	Description
000	ADD	A + B
001	SUB	A – B
010	AND	A & B
011	OR	A | B
100	XOR	A ^ B
101	NOT	~A
110	SHL	A << 1
111	SHR	A >> 1

3. **Status Flags**
   - Carry Flag (CF): Indicates carry or borrow for arithmetic operations.
   - Zero Flag (ZF): Asserted when the output result is zero.
4. **Verification**
   - Write a Verilog testbench to apply all combinations of operations and verify correctness of Y, CF, and ZF. Simulate the design using any HDL simulator. Observe and record the output waveforms.
5. **Synthesis**
   - Synthesize the behavioural ALU using Synopsys DC with the SCL 180 nm library. Take a note of the netlist, area, power and the slack obtained in the timing report.

Part 2 – Structural ALU (RCA-based Design)

Reuse the RCA module developed in the previous experiment. The RCA should be parameterized and return both sum and carry-out.

1. **Modify ALU for RCA-based Arithmetic**
   - Replace the behavioural addition/subtraction with the RCA module.
   - Use control logic to perform both addition and subtraction using the same RCA:
     - For subtraction: invert B and set carry-in = 1.

   - *Keep other logical operations identical to the behavioural version.*

2. **Simulation**
   - Develop a corresponding testbench. Verify that both versions (behavioural and structural) produce identical functional outputs.
3. **Synthesis and Comparison**
   - Synthesize the RCA-based ALU using Synopsys DC and compare area, power, and timing with the behavioural ALU.

Part 3 – Gate Level Simulation

Perform a gate level simulation to verify the functionality of the synthesized netlist.