ECE-571: Introduction to SystemVerilog (Design and Verification)
Fall-2025: Venkatesh Patil
Homework #2

In this assignment, you are going to build an **4-bit ALU**, utilizing submodules that performs operations as listed in the opcode. Make sure you utilize parameterized modules, package and enum data types where it make sense. Apply both implicit (.name) and implicit (.*) port connection styles. You will apply SystemVerilog constructs you learnt in the class so far. This assignment will give you an opportunity to:

1. practice different HDL modeling styles
2. Practice utilizing hierarchical development using sub-modules and instantiations
3. Writing self-checking testbenchs
4. Different styles of port-connection methods used in SystemVerilog

Build submodules (Design):

1. Design a structural model of 1-bit signed full adder using logic gates.
2. Design 4-bit ripple-carry adder using your 1-bit full adder
3. Design 4-bit inputs multiplier (this can be a simple behavioral model), but if you would like to challenge yourself you can create structural more efficient multiplier.
4. Design 2:4 decoder block with any RTL coding style

Verify submodules:

1. Write a conventional self-checking testbench to verify all the above submodules. You can create testbench for each of the submodule separately or one big testbench that verifies all the sub-blocks at once

Build and Verify 4-bit ALU

1. Design a digital ALU (Arithmetic Logic Unit) that will perform the following logic operations based the specification provided below: Ripple carry adder will only perform addition operation. For AND and SUB, you can use behavioral assignement.

opcode	operation	description	output
000	ADD	4-bit addition	8 bits
001	SUB	4-bit subtraction	8 bits
010	MUL	4x4 bit multiplication	8 bits
011	AND	bitwise and	8 bits
100	DEC	2-to-4 decoder on A[1:0]	8 bits
101-111	unused	unused	8 bits

2. Write a self-checking testbench to verify the ALU and its operations.
   - Follow the format for self checking displays input, actual output, expected out
     when reporting errors as shown here:
     $display("Error: A=%b B=%b Expected=%b Got=%b", A, B, expected, actual);

Deliverables: Follow the instructions

1. **Submit all the design files with the following naming style**
- Filename onebitFA.sv - module onebitFA (input logic a, b, cin, sum, output logic carry);
- fourbitFA.sv: module fourbitFA (input logic [3:0] a, b, output logic [3:0] sum, output logic carry);
- fourbitMUL.sv: module fourbitMUL (input logic [3:0] a, b, output logic [7:0] product);
- fourbitDEC.sv: module fourbitDEC (input logic [1:0] a, output logic [3:0] y);
- fourbitALU.sv: module fourbitALU (input logic [3:0] a, b, input logic [2:0] opcode, output logic [7:0] result);
2. **Testbench Files (Module names for testbenches should match their respective file names. Example: tb_onebitFA.sv --> module tb_onebitFA;**
- tb_onebitFA.sv
- tb_fourbitFA.sv
- tb_fourbitMUL.sv
- tb_fourbitDEC.sv
- tb_fourbitALU.sv
- **Self-Checking Output Format :** For self checking displays input, actual output, expected output when reporting errors
  $display("Error: A=%b B=%b Expected=%b Got=%b", A, B, expected, actual);
3. **Note on ALU operations:** When producing an 8-bit output from 4-bit operations:
- Use **zero-padding** for (bitwise AND and Decoder) — pad the upper bits with zeros.
- Use **sign extension** for **signed arithmetic operations** (ADD, SUB) — extend the sign bit (MSB) of the 4-bit result into the upper bits to preserve the correct signed value.
4. **Submission Format:** All the design, testbench files, waveforms, do files, and transcripts should be **zipped as a single file called ECE572f25_HW2.zip**
5. Follow these instructions for module declarations, naming conventions, and submission format, points will be deducted otherwise. All names are case-sensitive, including module names, filenames, port names, and parameter names.

Examples of padding and sign extensions:
4-bit inputs: A = 1010 (10), B = 0111 (7)
A & B = 0010 (2) // 4-bit result
Extend to 8-bit: 00000010 // upper 4 bits padded with 0
4-bit signed numbers: A = 0110 (6), B = 1011 (-5)
ADD: 0110 + 1011 = 0001 (1) // 4-bit result
Sign extension to 8-bit:
0001 -> 00000001 // MSB is 0 → positive → upper bits = ***0***
A = 0101 (5), B = 0110 (6)
SUB: 0101 - 0110 = 1111 (-1) // 4-bit result
Sign extension to 8-bit:
1111 -> 11111111 // MSB is 1 → negative → upper bits = 1
Input: A = 2'b10
Decoder output (4-bit): Y = 0100
Extended to 8-bit: Y_ext = 00000100