Lab 1 - Arithmetic and Logic Unit (ALU)

Introduction

Welcome to cs161L. This lab will be an introduction to microprocessors and the Verilog language. You will also be getting familiar with the Xilinx design environment. The goal of this lab is to implement a simple Arithmetic and Logic Unit (ALU) in Verilog.
First, follow this Xilinx ISE Tutorial. Additional Verilog examples here as a guide.

ALUs are hardware circuits that perform the arithmetic computations within a processor. They support multiple operations like addition, subtraction, multiplication, division, square roots, etc. The hardware logic to perform these operations can vary widely based on the approach used (Carry-Lookahead vs Ripple-Carry adder) or the data types supported (Integer, Float, Double). An ALU could even supply multiple versions of the same operation. They are not limited to only arithmetic operations. They can support bitwise operations, like AND OR and NOT, as well. Input data and control bits are sent to the ALU. The control bits specify an operation (OPCODE), and the ALU redirects the inputs to the corresponding functional circuit. When the computation completes the result is output along with extra data about the operation (overflow, underflow, carryouts, etc.)

Prelab

Write the test-bench (myalu_tb.v) for the ALU you will be designing. The test-bench should include tests for the boundary conditions and edge cases (as well as typical cases). Describe the test cases in comments.

Deliverables

For this lab you are expected to build an ALU that supports 8 arithmetic operations using the template provided (myalu.v) in the zip file. The ALU should be designed in such a way that the user can specify the operation's width (`N`) **without** modifying the source code (use parameters). In addition to the ALU you are also expected to build a test-bench that sufficiently verifies its correctness using the template provided (myalu_tb.v).

- The module name **must** be named "`myalu`"
- The module **must** use a parameter, called "`NUMBITS`", specifying the bit width
- The module **must *register*** all outputs (`reg`)
- The module **must** have input/output ports with the **EXACT** names listed below

Inputs	**Size**	**Outputs**	**Size**
`clk`	1-bit input	`result`	N-bit output

`reset`	1-bit input	`carryout`	1-bit output
`A`	N-bit input	`overflow`	1-bit output
`B`	N-bit input	`zero`	1-bit output
`opcode`	3-bit input		

- The module **must** support the operations listed below
  Hint: use switch (case) statements for the opcodes.

Operation	**Opcode**
unsigned add	`000`
signed add	`001`
unsigned sub	`010`
signed sub	`011`
bit-wise AND	`100`
bit-wise OR	`101`
bit-wise XOR	`110`
Divide A by 2	`111`

The **`zero`** port should be '`1`' when the **`result`** port is all zeros.
The specification for the overflow and the carry out signals is as follows.

Overflow

	A	***B***	***Result***
signed add	>= 0 < 0	>= 0 < 0	< 0 (Overflow) >= 0 (Overflow)
signed sub	>= 0 < 0	< 0 >= 0	< 0 (Overflow) >= 0 (Overflow)

Carryout

unsigned add	‘1’ when MSB's carryout is '1'
unsigned sub	‘1’ when MSB's carryout is '0'

‘0’ all other times

Submission:

Each student **must** turn in one zip file to Gradescope. The contents of which should be:

- A README file with your name and email address, and any incomplete or incorrect functionality
- All Verilog file(s) used in this lab (implementation and test benches).

If your file does not synthesize or simulate properly, you will receive a 0 on the lab.