IL2234

Digital Systems Design and Verification using Hardware Description Languages

Project Milestone 1

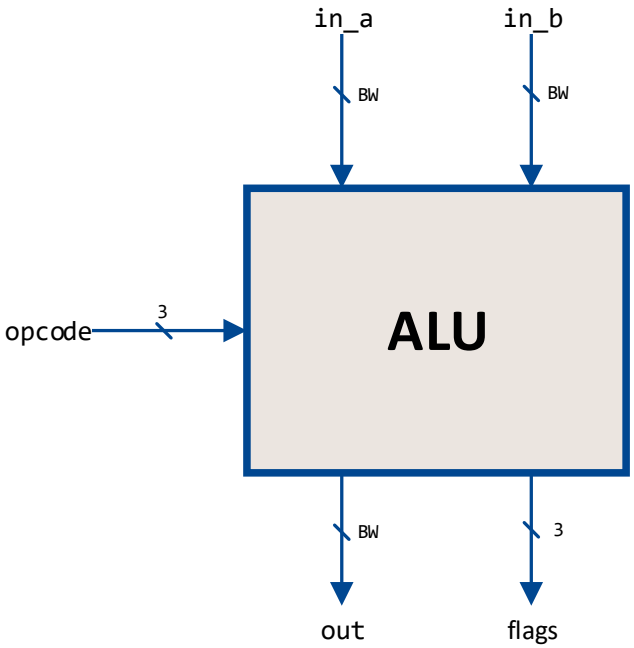
Combinational Circuits

Introduction

In this milestone, we will build the first component of our microprocessor: the Arithmetic Logic Unit (ALU). We will design and model the ALU circuit based on the SystemVerilog modelling concepts learned in the class. Additionally, we will develop a simple testbench to test the basic functionality of the ALU.

Overview

Below is a simple diagram of the ALU



The inputs and outputs are described in the table below

Name	Direction	Type	Bitwidth	Description
`in_a`	Input	`logic signed`	BW [1]	Operand A
`in_b`	Input	`logic signed`	BW [1]	Operand B
`opcode`	Input	`logic`	3	Operation code
`out`	Output	`logic signed`	BW [1]	Output result
flags	Output	`logic signed`	3	Flags of the result (overflow, negative, and zero detect)

[1] BW is a configurable parameter that defines the bitwidth of the ALU operands

The ALU behaviour is defined according to the following table.

Opcode	Name	Output	Description
000	ADD	in_a + in_b	Addition
001	SUB	in_a - in_b	Substraction
010	AND	in_a **AND** in_b	Logic AND
011	OR	in_a **OR** in_b	Logic OR
100	XOR	in_a **XOR** in_b	Logic XOR
101	INC	in_a + 1	Increment
110	MOVA	in_a	Passthrough A
111	MOVB	in_b	Passthrough B

Note that the inputs and outputs for arithmetic operations are in 2's complement format.

The `flag` output consists of 3 active high flags (overflow, negative, and zero detect) and works as follows:

- **Overflow**: asserted (1) if the result of AND or SUB gives the incorrect sign, i.e., the operation has an overflow. Deasserted (0) otherwise.
- **Negative**: asserted (1) if the result of **any** operation is a negative number. Deasserted (0) otherwise.
- **Zero detect**: asserted (1) if the result of **any** operation is zero. Deasserted (0) otherwise.

The three flags are combined in a 3-bit output as follows:

```
assign flags = {overflow,negative,zero};
```

Tasks

1. Design the ALU according to the specifications presented earlier. Write your code in System Verilog using **only synthesizable** constructs. Follow the same naming for the signals indicated in the diagrams and tables.
2. Write a testbench that generates inputs for the operands and opcodes; you may use randomisation functions to generate these inputs.

3. Simulate the ALU (DUT) in the testbench and check that the results are correct. Generate the necessary stimuli so all operations are checked, with multiple operands for each operation.

Deliverables

Use GitHub Classroom to submit your RTL and testbench files.

The GitHub Classroom repositories contain a skeleton of the RTL and testbench files for the ALU that you may use for your development.