4DV4-MiniProject-01
Arithmetic Logic Unit

COMPENG 4DV4: Very Large Scale Integration System Design

Deadline

Last day to submit this project is Tuesday Feb. 10 (10/2/2026) by 11:59 p.m

1 Data Preparation

1. Extract the assignment package using:

```
tar -xvf 1101_hw1.tar
```

2. The decompressed directory will contain:
   - `alu.v`: Design module to be implemented
   - `testbench.v`: Verification testbench for your design
   - `pattern/`: Nine instruction patterns for verification
   - `01_run`: Script for running NCVerilog
   - `99_clean`: Script for removing temporary files

2 Introduction

The Arithmetic Logic Unit (ALU) is a core computational block within a processor, responsible for executing both arithmetic and logical operations. In this assignment, you will implement an ALU that supports several specialized instructions and must correctly compute all provided inputs according to functional requirements.

3 Block Diagram

The following signals form the interface between the ALU and the host system:

- Operational clock and reset signals

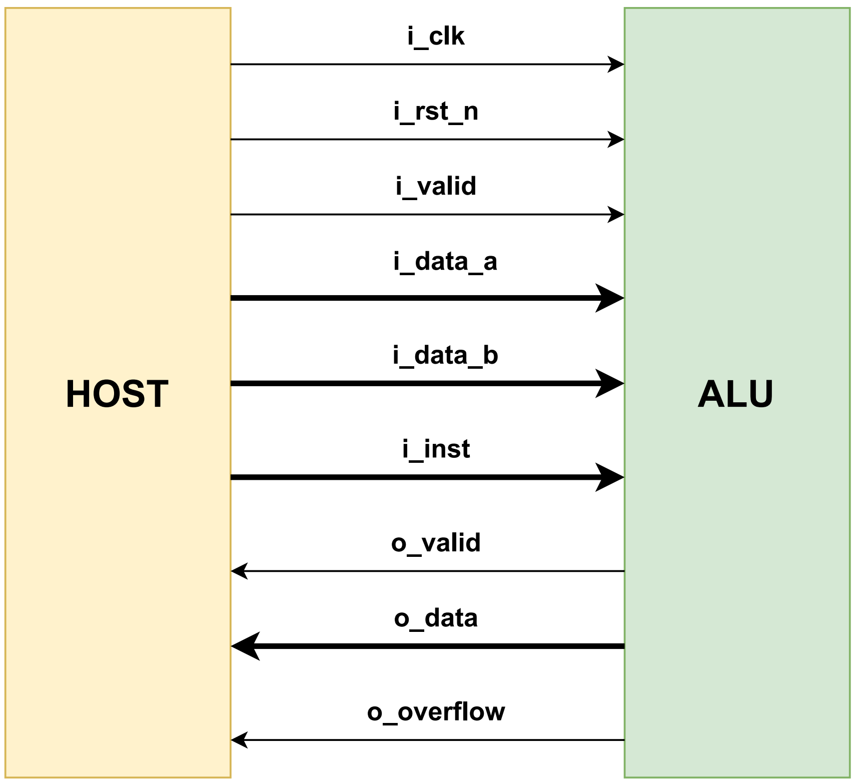

Figure 1: Interface between the ALU and the host system

- Signed fixed-point operand inputs
- Instruction command input
- Computed result, validity indicator, and overflow flag

4 Specifications

1. Module name: `alu`
2. I/O interface description:

Signal Name	I/O	Width	Description
`i_clk`	I	1	System clock input
`i_rst_n`	I	1	Active-low asynchronous reset
`i_valid`	I	1	High when operand and instruction inputs are valid
`i_data_a`	I	12	Signed fixed-point operand (7-bit integer + 5-bit fraction)
`i_data_b`	I	12	Signed fixed-point operand (7-bit integer + 5-bit fraction)
`i_inst`	I	3	Instruction code for operation selection
`o_valid`	O	1	High for a single cycle when output result is valid
`o_data`	O	12	ALU output in same fixed-point format
`o_overflow`	O	1	Asserted if computation result exceeds representable range

3. Inputs are sampled on the falling edge of the system clock.
4. All outputs must be registered and updated on the rising clock edge.
5. Assert all output signals to zero when `i_rst_n` is low.
6. `i_valid` is high for exactly one cycle indicating new input data.
7. `i_valid` will be randomly asserted throughout execution.
8. `o_valid` must assert high for only one cycle per result.
9. The testbench checks outputs on the falling clock edge only when `o_valid` is high.
10. If overflow occurs, `o_overflow` should be asserted and the testbench ignores `o_data` that cycle.
11. You may output results at any cycle as long as the valid signal is high.

5 Design Requirements

5.1 Instruction Definitions

`i_inst[2:0]`	Operation Description
000	Signed addition: $o\_data = i\_data\_a + i\_data\_b$
001	Signed subtraction: $o\_data = i\_data\_a - i\_data\_b$
010	Signed multiplication: $o\_data = i\_data\_a \times i\_data\_b$
011	MAC: multiply then accumulate continuously based on prior results
100	Bitwise XNOR: $o\_data = \sim (i\_data\_a \oplus i\_data\_b)$
101	ReLU: $o\_data = i\_data\_a$ if positive, else zero
110	Mean: $o\_data = \frac{i\_data\_a + i\_data\_b}{2}$
111	Absolute max: $\max(\lvert i\_data\_a \rvert, \lvert i\_data\_b \rvert)$

5.2 Additional Guidelines

- Use bitwise operations for instruction 100.
- For MAC (011), accumulation occurs only over uninterrupted MAC sequences; accumulation restarts when the instruction code changes.
- Overflow detection is required for opcodes 000, 001, 010, and 011.
- During MAC operations, once overflow is detected, maintain `o_overflow = 1` until the MAC sequence ends.
- Ensure correct signed extension when bit width variations occur.
- For fixed-point multiplication operations, properly manage the integer-fraction separation and apply rounding prior to output.
- For instruction 110 (Mean), truncate the least significant bit after computation.

6 Testing

To test you design run the following:

```
ncverilog testbench.v alu.v +define+I0 +access+rw
```

where you can change the tested instruction by changing I0 to I1 ... I8.

Submission

1. Create folder: `Proj1_studentID` that includes all your design files and a report showing your work with explanations and screenshots.
2. The report should also include screen shots using verdi (nWave) for all the different instructions.

```
nWave tb.vcd &
```

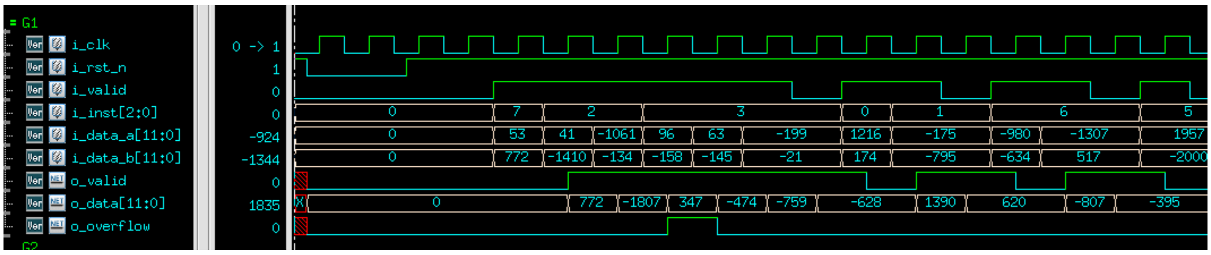

Figure 2: Sample Waveform

3. Make sure that all the signals in the I/O interface description are visible with the correct radix.
4. Compress and upload it to its corresponding Dropbox on avenue

7 Grading Policy

1. TAs will execute your design using different instruction:

```
ncverilog testbench.v alu.v +define+I0 +access+rw
.
.
ncverilog testbench.v alu.v +define+I8 +access+rw
```

   Your submission must compile and run with no errors.

2. Score breakdown:
   - Visible patterns: 80
     - I0, I1: 5
     - I2–I8: 10
   - Hidden patterns: 20
     - Based on accuracy over 10000 randomized instructions
3. Naming rule errors incur a 3-point deduction per violation.