# Computer-Aided VLSI System Design
# Homework 1: Arithmetic Logic Unit

TA: 陳定揚 r09943115@ntu.edu.tw **Due Tuesday, Oct. 4th, 13:59**

TA: 羅宇呈 f08943129@ntu.edu.tw

## Data Preparation

1. Decompress 1111_hw1.tar with following command

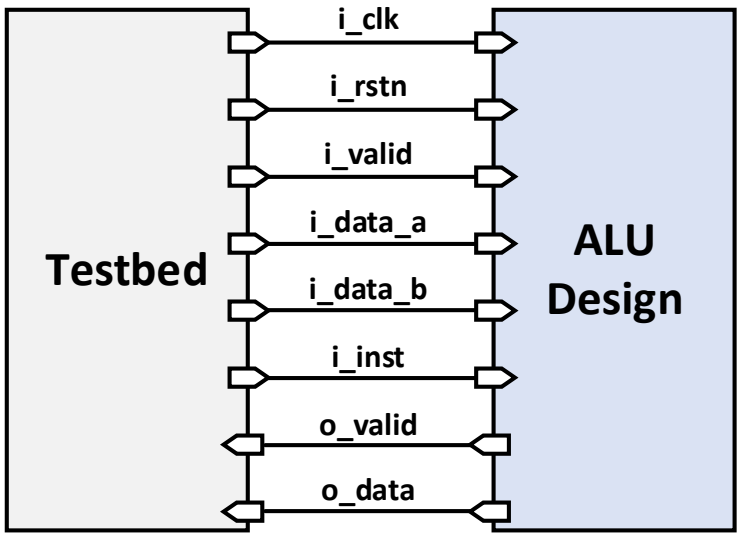
```
tar -xvf 1111_hw1.tar
```

| Files/Folder | Description |
|---|---|
| alu.v | Your design |
| testbench.v | File to test your design |
| pattern/ | 9 instruction data for testing |
| 01_run | NCverilog simulation command |
| 09_clean | Command for cleaning temporal files |

## Introduction

The Arithmetic logic unit (ALU) is one of the components of a computer processor. In this homework, you are going to design an ALU being able to compute special operations

## Block Diagram



## Specifications

1. Top module name: alu
2. Input/output description:

| Signal Name | I/O | Width | Simple Description |
|---|---|---|---|
| i_clk | I | 1 | Clock signal in the system |
| i_rst_n | I | 1 | Active **low** asynchronous reset |
| i_valid | I | 1 | The signal is **high** if input data is ready |
| i_data_a | I | 8 | Signed input data with 2's complement representation (3b integer + 5b fraction) |
| i_data_b | I | 8 | |
| i_inst | I | 3 | Instruction for ALU to operate |
| o_valid | O | 1 | Set **high** if ready to output result |
| o_data | O | 8 | Result after ALU processing with 2's complement representation (3b integer + 5b fraction). |

3. All inputs are synchronized with the negative edge clock.
4. All outputs should be synchronized at clock **rising** edge (Flip-flops are added at outputs)
5. Active low asynchronous reset (You should set all your outputs to be zero when i_rst_n is **low**)
6. i_valid will be pulled **high** for one cycle for ALU to get i_data_a, i_data_b and i_inst
7. i_valid will be pulled **high** anytime
8. o_valid should be pulled **high** for only **one cycle** for every o_data
9. When o_valid is **high**, the testbench will get your output at negative clock edge to check the answer
10. You can raise your o_valid anytime. TA will check your answer when o_valid is high

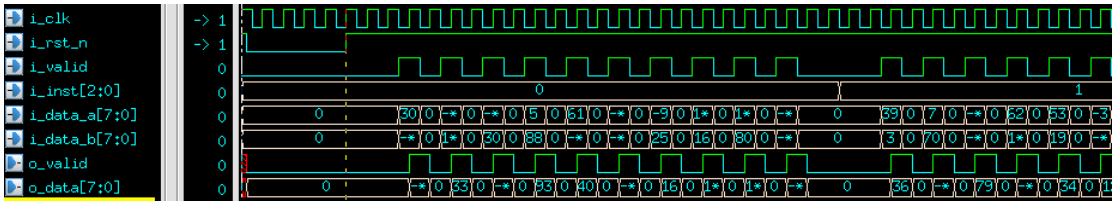
## Design Description

1. The followings are the functions of instructions you need to design for this homework:

| 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 * i_data_b |
| 011 | NAND | o_data = ( i_data_a · i_data_b )' |
| 100 | XNOR | o_data =( i_data_a ⊕ i_data_b)' |
| 101 | Sigmoid | o_data = $\sigma$( i_data_a ) |
| 110 | Right Circular Shift | See 111_HW1_note.pdf |
| 111 | Min | o_data = min ( i_data_a, i_data_b )<br>If i_data_a = i_data_b,<br>o_data = i_data_a |

2. Considerations between digital systems and signals:
   a. Bit-wise operation for instruction 011 and 100.
   b. For instruction 000, 001 and 010, output value might exceed the range of 8b representation (3b integer + 5b fraction).
   c. For instruction 000, 001 and 010, if the output value exceeds the maximum value of 8b representation, use the **maximum value** as output, and vice versa.
   d. For instructions 010, the result need to be **rounded to nearest number** before output
   e. For instructions 101, you need to implement a sigmoid function which is a popular activation function in in an artificial neural network. In order to have a easier implementation, we use **piecewise linear approximation** to compute sigmoid function. You can check 111_HW1_note.pdf for details.
   f. For instruction 110, you should implement **right circular shift** function and view i_data_b as rotation amount (8b integer). You can check 111_HW1_note.pdf for details.
   g. For instruction 111, if i_data_a = i_data_b, just output i_data_a
   h. Before and after the operation, if the total number of bits of the variable (reg or wire) is inconsistent, the system presets to complement the zero extension. Please use sign extension for the signal at the appropriate time.
   i. For fixed-point operation, please pay attention to the position of separation between integer and fraction after ALU operation.

## Sample Waveform




## Running RTL Simulation

1. To obtain file permissions for the entire folder have read/write/execute, run the following command under ./1111_hw1 before running simulation

```
chmod -R 777 ./
```

2. To run RTL simulation, you can execute 01_run directly

```
./01_run
```

3. To clean temporal file after simulation

```
./09_clean_up
```

## Submission

1. Create a folder named **studentID_hw1**, and put all below files into the folder
   - alu.v

   Note: Use **lower case** for the letter in your student ID. (Ex. r09943115_hw1)

2. Compress the folder **studentID_hw1** in a **tar file** named **studentID_hw1.tar**

```
tar -cvf studentID_hw1.tar studentID_hw1
```

3. Submit to NTU Cool

## Grading Policy

1. TA will run your code with following format of command. Make sure to run this command with no error message

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

2. Pass all the instruction test to get full score
   - Released patterns: **80%**

| i_inst [2:0] | Operation | Score |
|---|---|---|
| 000 | Signed Addition | 10% |
| 001 | Signed Subtraction | 10% |
| 010 | Signed Multiplication | 10% |
| 011 | NAND | 5% |
| 100 | XNOR | 5% |
| 101 | Sigmoid | 20% |
| 110 | Right Circular Shift | 10% |
| 111 | Min | 10% |

   - Hidden pattern: **20%**
     - Hidden pattern contains 9 different instructions
     - Only if you pass all patterns will you get full 20% score-
3. Delay submission
   - In one day: **(original score)*0.6**
   - In two days: **(original score)*0.3**
   - More than two days: **0 point** for this homework
4. Lose **3 point** for any wrong naming rule

## References

1. 2'complement: https://en.wikipedia.org/wiki/Two%27s_complement
2. To understand fixed-point representation with fraction number: https://reurl.cc/D3xQnR