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

TA: 曾柏豪 r13943123@ntu.edu.tw **Due Tuesday, Sep. 30th, 13:59**

TA: 陳柏任 d13943013@ntu.edu.tw

Data Preparation

1. Unpack 1141_hw1.tar with the following command

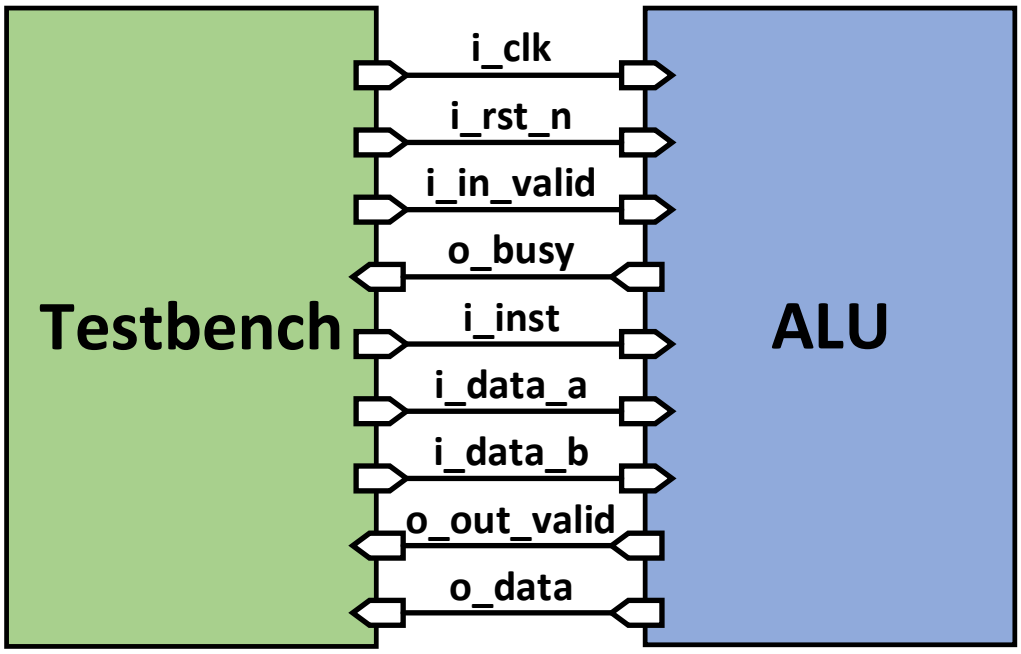
```
tar -xvf 1141_hw1.tar
```

Folder	File	Description
00_TESTBED	testbench.v	File to test your design
00_TESTBED /pattern	INST*_I.dat	Input instruction patterns
	INST*_O.dat	Output golden patterns
01_RTL	alu.v	Your design
	rtl.f	File list for RTL simulation
	01_run	VCS command for simulation
	99_clean	Command for cleaning temporal files

Introduction

An arithmetic logic unit (ALU) is one of the components of a computer processor. It performs arithmetic and bit-level logical operations in a computer. In this homework, you are going to design an ALU with some special instructions.

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_in_valid	I	1	The signal is **high** if input data is ready
o_busy	O	1	Set **low** if ready for next input data. Set **high** to pause input sequence.
i_inst	I	4	Instruction for ALU to perform
i_data_a	I	16	Signed input data with 2's complement representation 1. For instructions 0000~0011, fixed-point number (6-bit signed integer + 10-bit fraction) 2. For instructions 0100~1001, integer (for instruction 0100, it is unsigned integer!)
i_data_b	I	16	
o_out_valid	O	1	Set **high** if ready to output result
o_data	O	16	Signed output data with 2's complement representation 1. For instructions 0000~0011, fixed-point number (6-bit signed integer + 10-bit fraction) 2. For instructions 0100~1001, integer

3. Active low asynchronous reset is used only once.

4. All inputs are synchronized with the **negative** clock edge.
5. All outputs should be synchronized with the **positive** clock edge. That is, flip-flops should be added before all outputs.
6. New pattern (i_inst, i_data_a and i_data_b) is ready only when i_in_valid is high.
7. At each negative clock edge, i_in_valid will be randomly pulled high only if o_busy is low.
8. o_out_valid should be pulled high for only one cycle for each o_data.
9. The testbench will sample o_data at **negative** clock edge if o_out_valid is high.
10. You can raise o_out_valid at any moment.

Design Description

1. The following table shows the operations you need to implement on your ALU:

i_inst[3:0]	Operation	Description
4'b0000	Signed Addition	o_data = i_data_a + i_data_b
4'b0001	Signed Subtraction	o_data = i_data_a - i_data_b
4'b0010	Signed Multiplication and Accumulation	o_data = i_data_a * i_data_b + $\text{data\_acc}_{\text{old}}$
4'b0011	Taylor Expansion of Sin Function	o_data = $\sum_{n=0}^{2} \frac{(-1)^n}{(2n+1)!}(\text{i\_data\_a})^{2n+1}$
4'b0100	Binary to Gray Code	Encode the gray code result
4'b0101	LRCW	Encode the CPOP result
4'b0110	Right Rotation	Rotate i_data_a right by i_data_b bits
4'b0111	Count Leading Zeros	Count leading 0's in i_data_a
4'b1000	Reverse Match4	Custom bit-level operation
4'b1001	Matrix Transpose	Transpose an 8*8 matrix

2. Specifications on number representation:
   a. For instructions 0000~0011, saturation must be applied to the final result. That is, if the final result exceeds the maximum (minimum) representable value of 16-bit representation (6-bit integer + 10-bit fraction), use the maximum (minimum) value as output.
   b. For instructions 0010 and 0011, rounding must be applied to the final result before saturation. The rounding mode used is **rounding to the nearest**[2]. That is, if the value of the remaining bits under LSB is greater than or equal to half the value representable by LSB, it should be rounded up.
3. For instruction 0010, The accumulator, data_acc, is cleared to 0 when the active-low reset i_rst_n is applied. The value in data_acc is first clamped (saturated) to its maximum or minimum threshold (16 bits int, 20 bits fraction) before the next accumulation cycle begins. Noted that any **intermediate value or accumulation**

value to be accumulated cannot be rounded

4. For instruction 0011, please use the following Taylor Expansion to compute Sin [3], a non-linear function. i_data_a is guaranteed to be between 1.0 and -1.0. It is not allowed to use the division operator (/) in Verilog code.

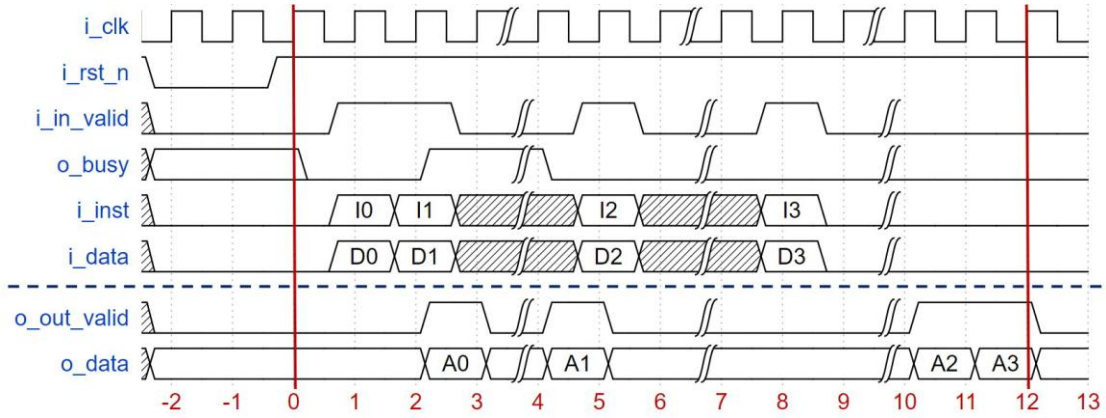
$$\sin(\text{i\_data\_a}) \cong \sum_{n=0}^{2} \frac{(-1)^n}{(2n+1)!} (\text{i\_data\_a})^{2n+1}$$

5. For instructions 0110, i_data_b is guaranteed to be from 0 to 16 (inclusive).
6. For instructions 0111, 1000 and 1001, it is recommended to use for loops.
7. For all instructions, there cannot be any combinational loop. Otherwise, the instruction will not be scored.
8. For instruction 1000, implement the following custom bit-level operation.

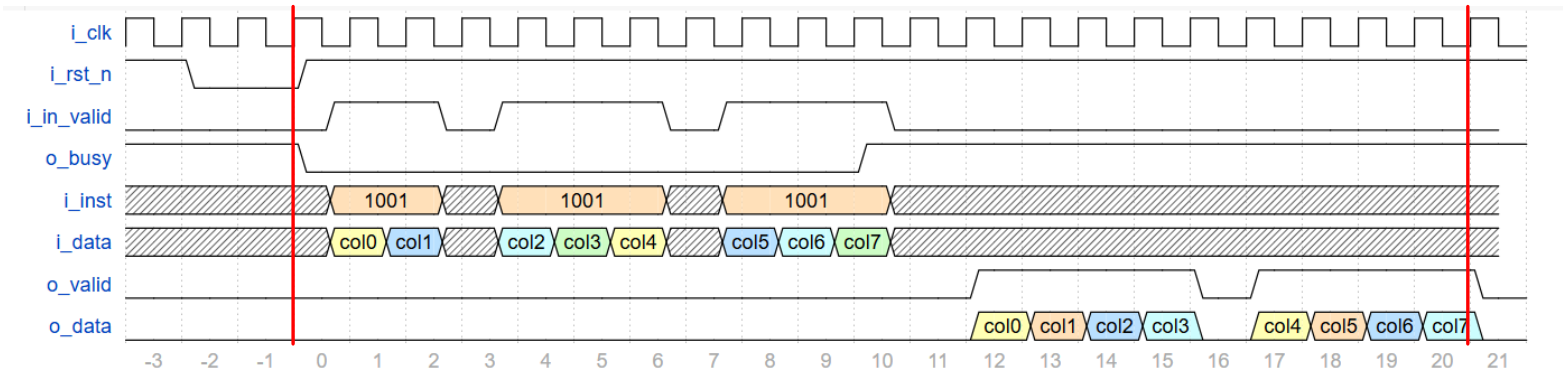
$$\text{o\_data[i]} = \begin{cases} (\text{i\_data\_a}[i+3:i] == \text{i\_data\_b}[15-i:12-i]), & i = 0 \sim 12 \\ 0, & i = 13 \sim 15 \end{cases}$$

9. For instruction 1001, a matrix transpose operation requires collecting 8 cycles of valid input data. These cycles may arrive non-consecutively, but the entire 8-cycle transfer is guaranteed to complete before the next instruction is issued.
10. For instruction 1001, the output data is transmitted over 8 cycles. The o_valid signal must be asserted during each cycle that contains valid output data. Besides, all valid results for the current matrix transpose operation must be output before the results from the **subsequent instruction** are transmitted
11. You **CANNOT** implement any operation by look up tables (the scaling factors of reciprocal in instruction 0011 are allowed).
12. You are **NOT** allowed to use DesignWare.

Sample Waveform



Instruction 1001 Waveform



Submission

1. Create a folder named **studentID_hw1** and follow the hierarchy below.

```
r13943000_hw1
    └── 01_RTL
        ├── alu.v
        ├── xxx.v (other verilog files you wrote)
        └── rtl.f
```

   Note: Use **lowercase** for all the letters. (e.g. r13943000_hw1)

2. Pack the folder **studentID_hw1** into a **tar file** named **studentID_hw1_v*k*.tar (*k* is the number of version, *k* =1,2,…)**. TA will only check the last version.

```
tar -cvf studentID_hw1_vk.tar studentID_hw1
```

   Note:

   a. Use **lowercase** for all the letters. (e.g. r13943000_hw1_v1.tar)

   b. Pack the folder on IC Design LAB server to avoid OS related problems.

3. Submit to NTU Cool

Grading Policy

1. TA will run your code with the following format of command. Make sure to run this command with no error message on IC Design LAB server.

```
vcs -full64 -R -f rtl.f +v2k -sverilog -debug_access+all +define+$1
```

2. Pass all the instruction tests to get full score.
   - Released patterns: **70%**

i_inst[3:0]	Operation	Score
4'b0000	Signed Addition	5%
4'b0001	Signed Subtraction	5%
4'b0010	Signed MAC	10%
4'b0011	Taylor Expansion of Sin Function	10%
4'b0100	Binary to Gray Code	5%
4'b0101	LRCW	5%
4'b0110	Right Rotation	5%
4'b0111	Count Leading Zeros	8%
4'b1000	Reverse Match4	8%
4'b1001	Matrix Transpose	9%

- Hidden patterns: **30%**
  - Mixture of all instructions

3. SpyGlass check (goal: lint_rtl and lint_rtl_enhanced) with error: -20%
4. Lose **5 points** for any incorrect naming or format.
   - It is your responsibility to ensure that the files can be correctly unpacked and executed on IC Design LAB server.
5. No late submission
   - 0 point for this homework
6. No plagiarism
   - **Plagiarism in any form, including copying from online sources, is strictly prohibited.**

References

1. Reference for fixed-point representation
   https://www.allaboutcircuits.com/technical-articles/fixed-point-representation-the-q-format-and-addition-examples/
2. Reference for rounding to the nearest
   https://www.mathworks.com/help/fixedpoint/ug/rounding.html
3. Reference for Taylor Expansion function
   https://en.wikipedia.org/wiki/Taylor_series
4. Reference for reciprocal multiplication
   https://homepage.divms.uiowa.edu/~jones/bcd/divide.html
5. Reference for LRCW
   Comparing fast implementations of bit permutation instructions