Computer Architecture HW2: Simple ALU Unit

Deadline: 14:20 on Nov. 26, 2024

In this homework, you will implement a simple arithmetic logic unit (ALU) in Verilog. If you are not familiar with the language, a tutorial video with a basic introduction to Verilog can be found on NTU COOL.

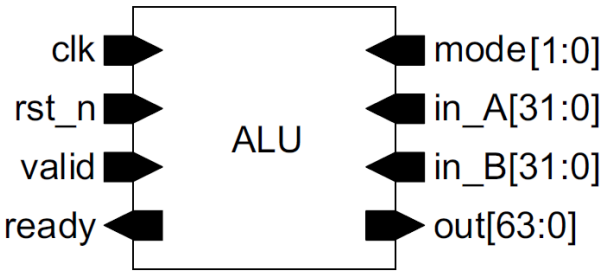
Goal

In this assignment, you will implement an **unsigned multiplier**, an **unsigned divider**, as well as a few **arithmetic**, **logical**, **comparison**, and **shift** operations. You will then combine these operations into a single unit. Our goal is to help you learn:

- How to use assignments with wire.
- How to use always blocks with reg.
- How to describe combinational and sequential circuits.
- How to control a state machine.
- How to develop a good coding style.

Specification

The following are the specifications of the ALU:



Name	I/O	Width	Description
clk	I	1	Positive edge-triggered clock
rst_n	I	1	Asynchronous negative edge reset
valid	I	1	Valid input data when valid = 1
mode	I	4	See the table below
in_A[31:0]	I	32	Input A
in_B[31:0]	I	32	Input B
ready	O	1	Ready output data when ready = 1
out[63:0]	O	64	Output

Mode	Operation	Description	Note
0	Add	in_A + in_B	Signed operation
1	Sub	in_A - in_B	Signed operation
2	AND	in_A & in_B	Bit-wise
3	OR	in_A | in_B	Bit-wise
4	XOR	in_A ^ in_B	Bit-wise
5	Equal	out[0]=1 if (in_A == in_B)	Signed operation
6	Greater than	out[0]=1 if (in_A >= in_B)	Signed operation
7	Shift right logical	in_A >> in_B	Unsigned operation
8	Shift left logical	in_A << in_B	Unsigned operation
9	MUL	Use your multiplier	Unsigned operation
10	DIV	Use your divider	Unsigned operation

Operations

The 32-bit ALU should provide arithmetic, logical, comparison, and shift operations. Also, the hardware can be shared between multiplication and division operations. The following are the descriptions of the operations and the regulations of the input-output format:

Addition and Subtraction

In these two modes, the unit performs 32-bit signed addition and subtraction.

- Both in_A and in_B are considered as 32-bit numbers in 2's complement representation.
- If an overflow occurs, the 32-bit output should be set to the largest positive value or the smallest negative value that a 32-bit number can represent.
- Since out has 64 bits, out = {32'd0, in_A ± in_B }.

AND, OR, XOR

In these three modes, the unit performs bitwise AND, OR, XOR operation for 32-bit numbers.

- Since out has 64 bits, out = {32'd0, in_A & in_B }

Equal and Greater Than

In these two modes, the unit performs a 32-bit signed comparison operation.

- Both in_A and in_B are considered as 32-bit numbers in 2's complement representation.
- Since out has 64 bits, out = {63'd0, (in_A == in_B) }.

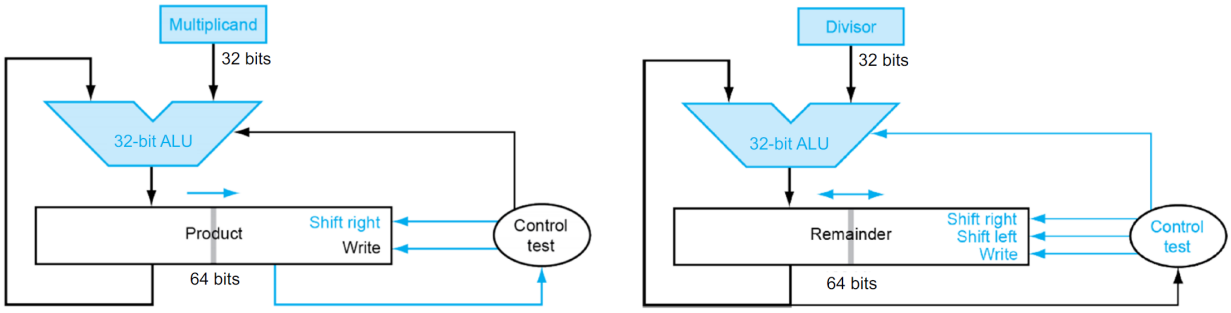
Shift Right and Shift Left

In these two modes, the unit performs a 32-bit unsigned logical shift operation.

- in_A and in_B are considered 32-bit unsigned numbers.
- Be careful with the difference between logical shifts and arithmetic shifts in Verilog.
- Since out has 64 bits, out = {32'd0, (in_A >> in_B) }.

Multiplier and Divider

In these two modes, the unit performs multiplication and division with 32-bit inputs. Think about the optimized multiplier and divider you have learned in Chapter 3. They can both reduce the register consumption and the size of the ALU.



(a) Multiplier (b) Divider

Figure 1

The structures of both multiplier and divider are similar, including operand register, ALU and shift register. You can reuse the hardware and reduce the total area.

- Multiplication
  - in_A and in_B are considered as unsigned 32-bit numbers.
  - out = in_A × in_B (32b × 32b → 64b)
- Division
  - in_A and in_B are considered as unsigned 32-bit numbers.
  - quotient = in_A / in_B (quotient is 32b)
  - remainder = in_A % in_B (remainder is 32b)
  - out = {remainder, quotient}

Design issue

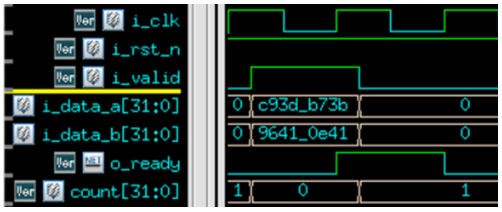
In this ALU, we assume you will need 1 cycle to load data. Then 32 cycles are required to do multiplication/division, and modes 0 to 8 need to be done immediately.

We consider CPU is issuing a task to ALU, so inputs are valid to ALU for only 1 cycle.

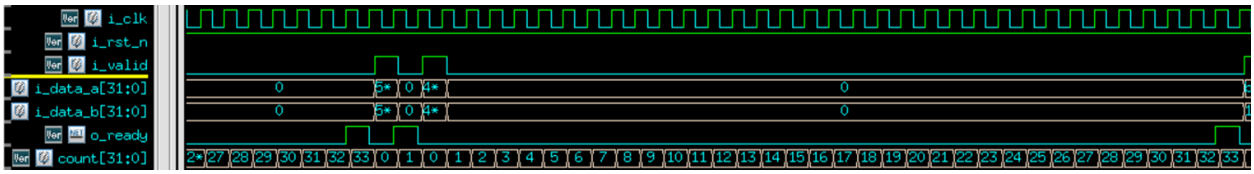
Waveform

- The testbench checks the result when the ready signal goes high.
- For operations with mode 0 to mode 8, the ready signal needs to raise at the same cycle when the inputs are loaded.
- For operations with mode 9 and 10, the ready signal needs to raise after 33 cycles when the input valid is detected.

(a) Reset waveform



(b) Other operations' waveform



(c) MUL and DIV waveform

Figure 2

Simulation

There are two files in the folder named "code":

- ALU.v
- testbench.v

To run simulation, you should run source command in advance.(Use the given license.cshrc)

```
$ source license.cshrc
```

Then run the Verilog simulation

```
$ vcs testbench.v ALU.v -full64 -R -debug_access+all +define+p0
```

If you want to test the other public patterns, change `+define+p0` in the command to `+define+p1` or `+define+p2` .

Check your design

In your ALU, all sequential elements must be flip-flops. Use **Design Compiler** to check if there's any latch in your design. The commands are as follows:

```
$ dc_shell

dc_shell> read_verilog ALU.v
```

All the sequential elements in your design should be flip-flop.

Report

Provide the snapshot of the following table in your report. (Only need this screenshot in your report.pdf)

Register Name	Type	Width	Bus	MB	AR	AS	SR	SS	ST
cnt_r_reg	Flip-flop	5	Y	N	Y	N	N	N	N
MulDiv_in_r_reg	Flip-flop	32	Y	N	Y	N	N	N	N
state_r_reg	Flip-flop	2	Y	N	Y	N	N	N	N
out_buffer_r_reg	Flip-flop	64	Y	N	Y	N	N	N	N
multiply_r_reg	Flip-flop	1	N	N	Y	N	N	N	N

You can name all registers in your design freely. Just make sure that all sequential elements in your design are flip-flops.
If there are latches, you will get -1 point as a penalty

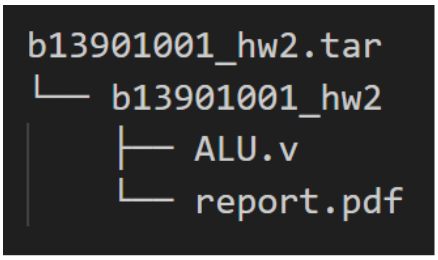
Scoring

- 3 public pattern and 3 hidden pattern
  - **each pattern worth 1 point.**

Submission

- Deadline: 14:20 on Nov. 26, 2024
  - No late submission allowed
- Upload <student_id>_hw2.tar to NTU COOL
  - **Wrong file or folder format will get -1 point as a penalty**
  - Compress your files with the following instruction

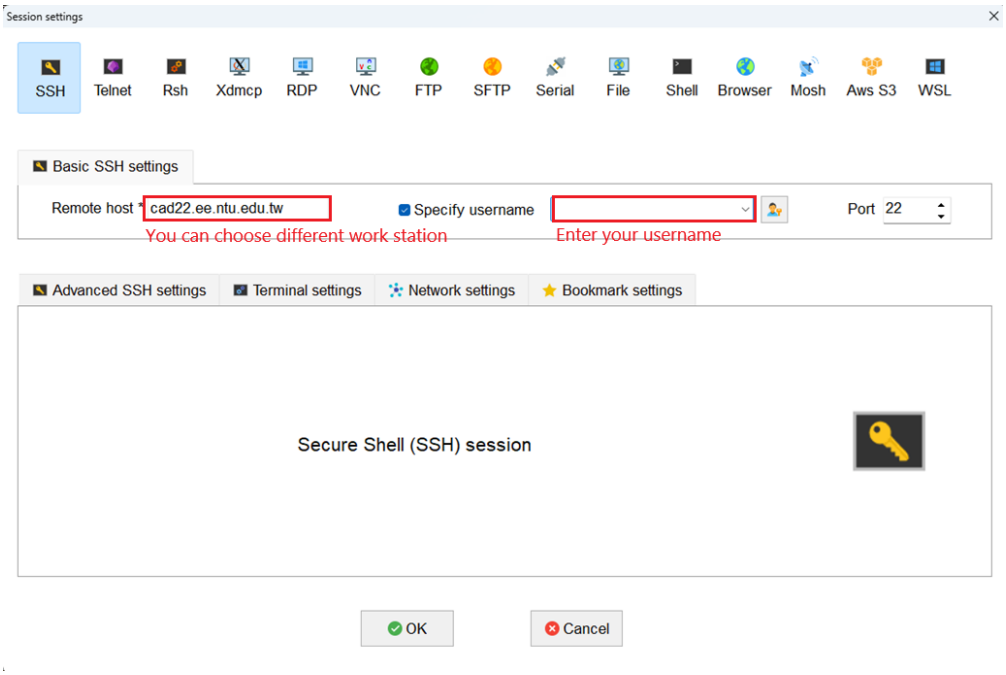
```
$ tar -cvf b13901001_hw2.tar b13901001_hw2/
```



Appendix

Connect to 231 workstation

- Apply an account on 231 work station.
- Use NTU's ip.
- Connect by MobaXterm



- Connect to different workstations

```
# Workstations: cad16,cad17,cad20,cad21,cad23,cad27,cad29,cad30,cad31,cad33,cad34,cad38,cad39,cad40
  Ex: [Username@cad]# ssh -X cad??
```