COMP311-1: Logic Circuit Design

Fall 2019, Prof. Taigon Song

Project 1. Due: Nov. 15 9:59am [Total: 50 points]

Goal: Design an ALU that supports two 16-bit inputs and performs addition, subtraction, multiplication, and division. You are to have one 16-bit output and a bit of overflow (underflow).

Specifications:

- Two 16-bit inputs: ina, inb
- One 2-bit input: sel
- One 16-bit output: out???? (last 4 digits of your student ID)
- One 1-bit output: overflow

"sel" is the signal that controls whether you wish to perform addition (00), subtraction (01), multiplication (10), or division (11). Therefore, inside the top-module ALU, you should design four sub-modules: add, sub, mul, div. Since the outputs of these sub-modules should not merge to the top-module output, the outputs of these four modules are controlled by a multiplexer. Assumed inputs are two unsigned integer numbers, and when an unallowed output is executed, overflow bit becomes 1.

Handicap: The designs should not use any arithmetic operators (+,-,*). ALU design using these operators will receive 0 credit. However, using arithmetic operators when designing 'div' module will be allowed. The ALU to be designed is purely combinational. This means that no clocks can be involved in the ALU design.

[Hints of the four sub-modules]

Adder: Use the ripple-carry adder that is described in your textbook. You are to start from a 1-bit full-adder, then merge it to 4-bit adder, then 16-bit adder. No 'assign' statements or if-else, case statements allowed. Only primitive gates, and modules that you designed can be used.

Subtractor: Figure 1 is a way to implement a subtractor by recycling the adder you

designed.

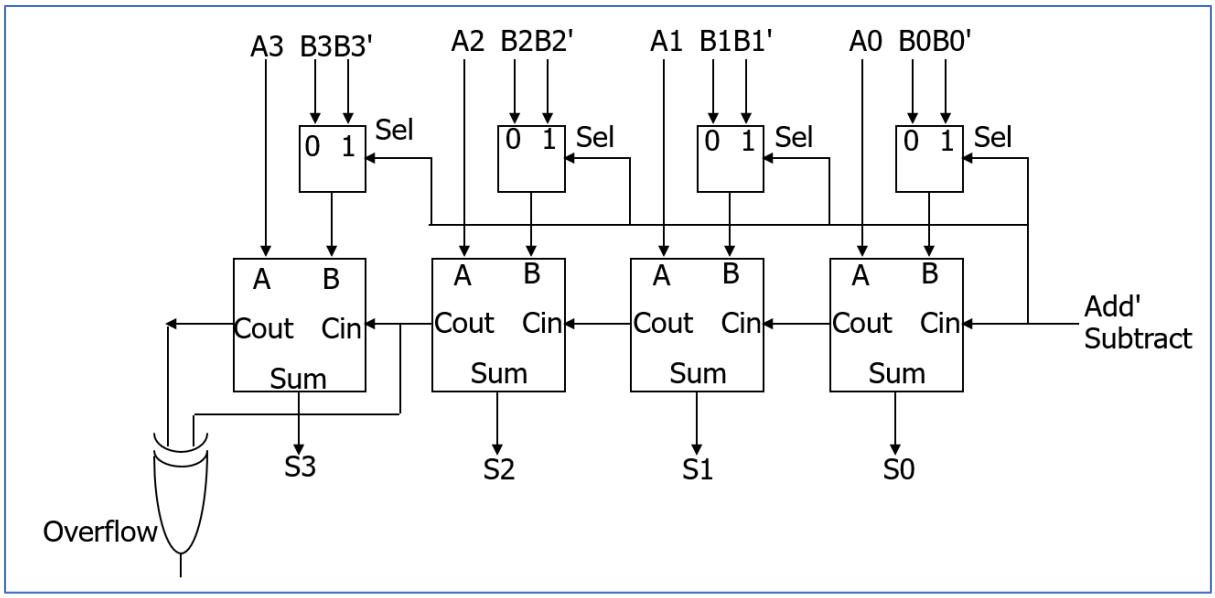

Fig. 1 – Hint for subtractor

Multiplier: The idea of multiplication is to perform multiple addition based on the numbers inserted. For example, a 3-bit multiplication performed, and the implementation would be something similar to Fig. 2. Recycle the adder you designed and implement your multiplier.

Note: Do not use any arithmetic (+,-,*,/) operators in any circumstances.

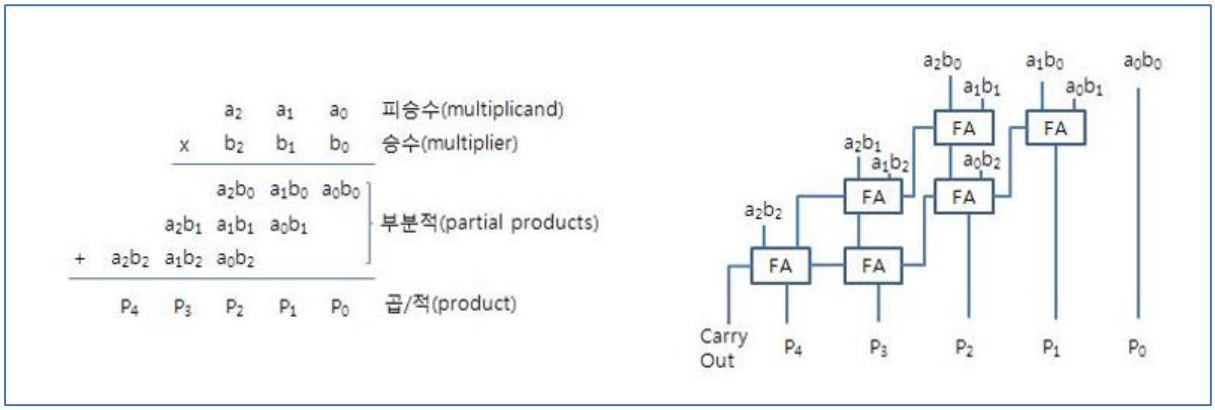

Fig. 2 – Hint for multiplier

(image source: http://www.ktword.co.kr/abbr_view.php?m_temp1=4891)

Divider: There are many ways to implement a divider, however, it is very challenging to implement a divider by primitive gates and modules. Find a way via web and implement the idea to your design. You may not use the '/' operator, but you may use other arithmetic operators by your choice. Below are some links that could provide you some hints to implement the divider.

https://blastic.tistory.com/136

http://bwrcs.eecs.berkeley.edu/Classes/icdesign/ee141_s04/Project/Divider%20Background.pdf

Note: If you can't clearly describe your divider implementation in your report, you will receive 0 credit. Explain your divider implementation line-by-line.

[testbench design]

At every 10ns, insert random numbers 10 times for each addition, subtraction, multiplication, and division and verify your functionality at ALU. Justify why your design is correct. If 10 random input is not enough to justify your functionality, you may have more. However, you need to have at least 6 regular outputs and 2 overflow outputs.

Format should be something like the below:

Input1	Input2	Correct output	Your output

Hint: for multiplication input, you may try something like ($random % 256) for verification.

Scores:

You will receive credit when your modules are validated via your testbench.

Adder: 5 points

Subtractor: 10 points

Multiplier: 10 points

Divider: 10 points

Top-module ALU: 5 points

Synthesizable: 10 points

[Submission]

- Submit your report in word/hwp/pdf, and submit your source codes compressed in a .zip file.
- In order to verify if your codes are synthesizable, please attach the synthesis report in your .zip file.