COMP311: Logic Circuit Design

Spring 2022, Prof. Taigon Song

Project 1. Due: May 12 7:59pm [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: ov (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. (For subtractor, inputs and outputs are assumed to be 2's complements.)

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 (e.g., using +, -, * in div) 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.

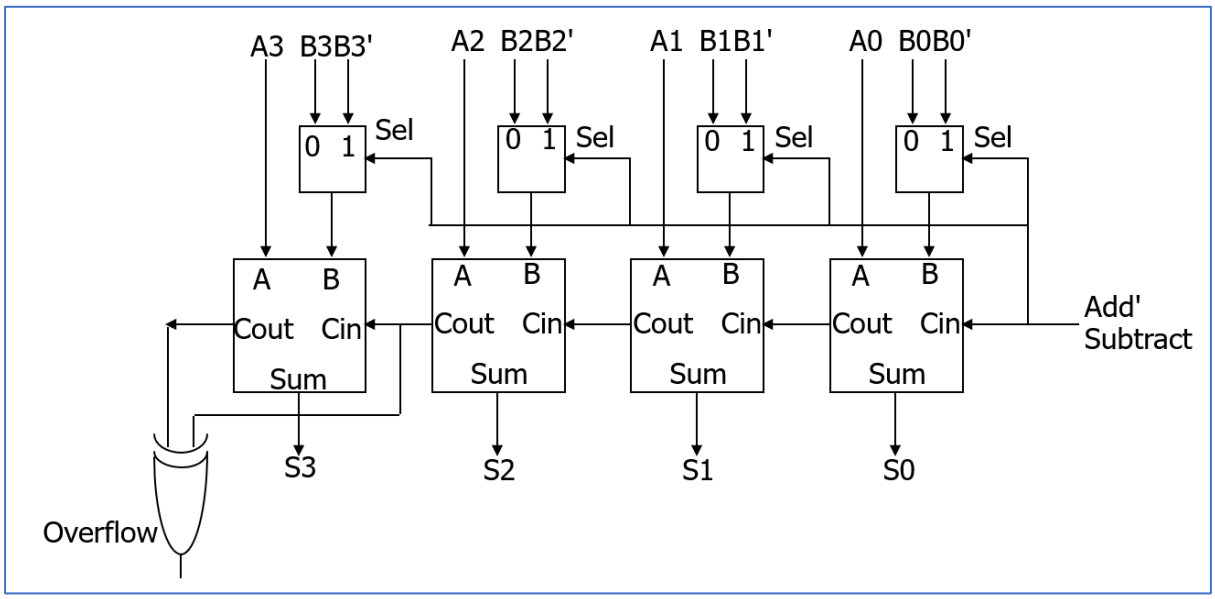

Fig. 1 – Hint for subtractor

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 for multiplier.

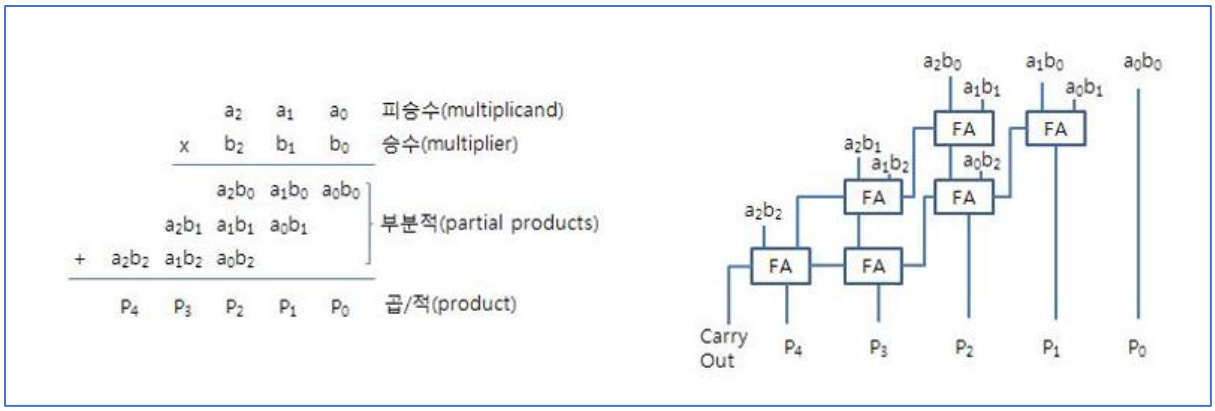

Fig. 2 – Hint for multiplier

(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.

[Regarding overflow]

Note that if you have incorrect results, then 'ov' will be 1. Otherwise, ov will be 0.

[testbench design]

At every 10ns, insert random numbers (of your choice) 10 times for each addition, subtraction, multiplication, and division and verify your functionality of your 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 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 (details will be announced by your TA. Make sure you screen capture your QOR results)

[Submission]

- Submit your report in pdf, and submit your source codes compressed in a .zip file.
- In order to verify if your codes are synthesizable, please screen capture the synthesis report in your .zip file (and in your report pdf).
- Present your results in decimal (radix option in iVerilog)

[Example of your QOR report]

```
****************************************
Report : qor
Design : aes_cipher_top
Version: O-2018.06-SP4
Date   : Tue Mar  2 12:09:32 2021
****************************************


  Timing Path Group 'myCLK'
  -----------------------------------
  Levels of Logic:               16.00
  Critical Path Length:           6.13
  Critical Path Slack:           -0.16
  Critical Path Clk Period:       6.00
  Total Negative Slack:          -3.59
  No. of Violating Paths:        48.00
  Worst Hold Violation:           0.00
  Total Hold Violation:           0.00
  No. of Hold Violations:         0.00
  -----------------------------------


  Cell Count
  -----------------------------------
  Hierarchical Cell Count:          22
  Hierarchical Port Count:         612
  Leaf Cell Count:               11836
  Buf/Inv Cell Count:             2090
  Buf Cell Count:                  314
  Inv Cell Count:                 1776
  CT Buf/Inv Cell Count:             0
  Combinational Cell Count:      11306
  Sequential Cell Count:           530
  Macro Count:                       0
  -----------------------------------


  Area
  -----------------------------------
  Combinational Area:    24581.061924
  Noncombinational Area:  3508.966321
  Buf/Inv Area:           2986.446182
  Total Buffer Area:           713.13
  Total Inverter Area:        2273.32
  Macro/Black Box Area:       0.000000
  Net Area:               6746.052167
  -----------------------------------
  Cell Area:             28090.028244
  Design Area:           34836.080412


  Design Rules
  -----------------------------------
  Total Number of Nets:          12146
  Nets With Violations:              0
  Max Trans Violations:              0
  Max Cap Violations:                0
  -----------------------------------


  Hostname: knuee-srv1

  Compile CPU Statistics
  ---------------------------------------
  Resource Sharing:                   0.61
  Logic Optimization:                 7.35
  Mapping Optimization:              22.44
  ---------------------------------------
  Overall Compile Time:              33.07
  Overall Compile Wall Clock Time:   33.48
```