**National Cheng Kung University**

**Department of Electrical Engineering**

***Introduction to VLSI CAD (Spring 2021)***

**Lab Session 3**

**Multiplexers, ALUs and the Conversion between RGB and Grayscale**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Student ID | | |
| 陳旭祺 | E24099059 | | |
| Practical Sections: | | Points | Marks |
| Prob A | | 20 |  |
| Prob B | | 30 |  |
| Prob C | | 30 |  |
| Report | | 20 |  |
| Notes | | | |

**Due Date: 15:00, March 17, 2021 @ moodle**

**Deliverables**

1. All Verilog codes including testbenches for each problem should be uploaded.

NOTE: Please **DO NOT** include source code in the paper report!

1. All homework requirements should be uploaded in this file hierarchy or you will not get the full credit.

NOTE: Please **DO NOT** upload waveforms!

1. Important! TA will use the command in Appendix A to check your design under SoC Lab environment, if your code can not be recompiled by TA successfully using the commands, you will not get the full credit.
2. **If you upload a dead body which we can’t even compile you will get NO credit!**
3. **All Verilog file should get at least 90% superLint Coverage.**
4. **File hierarchy should not be changed; it may cause** your code can not be recompiled by TA successfully using the autograding commands

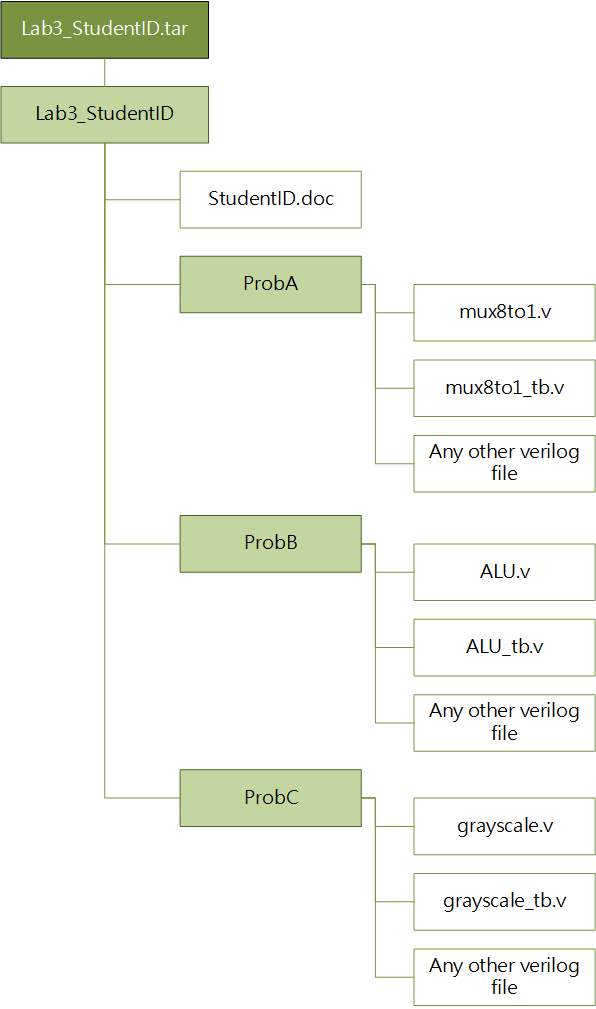


Fig.1 File hierarchy for Homework submission

**Objectives:**

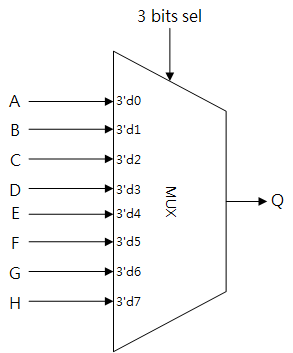
**To make you be familiar with some designs of combinational logic, like adders, decoders and some operation that uses multiplication and addition. You can follow this document to practice, or you have a cleverer way. Please show your best.**

Note that you can extend the spacing if it is not enough for you to answer.

Prob A: A 8-to-1 multiplexer and testbench

1. **Complete a 8-to-1 multiplexer.**

Design a 8-to-1 multiplexer. Design the testbench as well. Testbench need to test all selected inputs and print results.

****

* 1. Name your design file *mux8to1.v* and your testbench file *mux8to1\_tb.v.*
  2. The frame code is given.
  3. Inputs: **A**, **B, C, D, E, F, G, H, sel**
  4. Outputs: **Q**
  5. Include all needed Verilog files in your testbench.

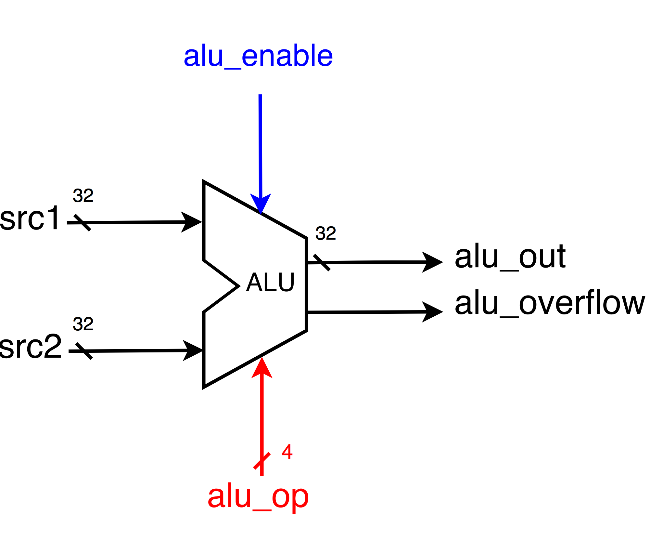
1. **Please attach your design waveforms.**

|  |
| --- |
| Your simulation result on the terminal  ( You should $monitor the I/O port like the example TB code does! ) |
| 一張含有 文字 的圖片  自動產生的描述一張含有 文字 的圖片  自動產生的描述 |
| Your waveform |
|  |
| Explanation of your waveform |
| A multiplexer is a device that selects between several input signals and forwards the selected input to a single output line. The selection is directed a separate set of digital inputs known as select lines. Because I use in testbench, the waveform looks regular and periodical. Look into detail, all the output signal is 0 except the condition when input signal A-H is 1 meanwhile the selection signal selects that line, which meets my expectation. |
| SuperLint Coverage |
| 一張含有 文字 的圖片  自動產生的描述 |

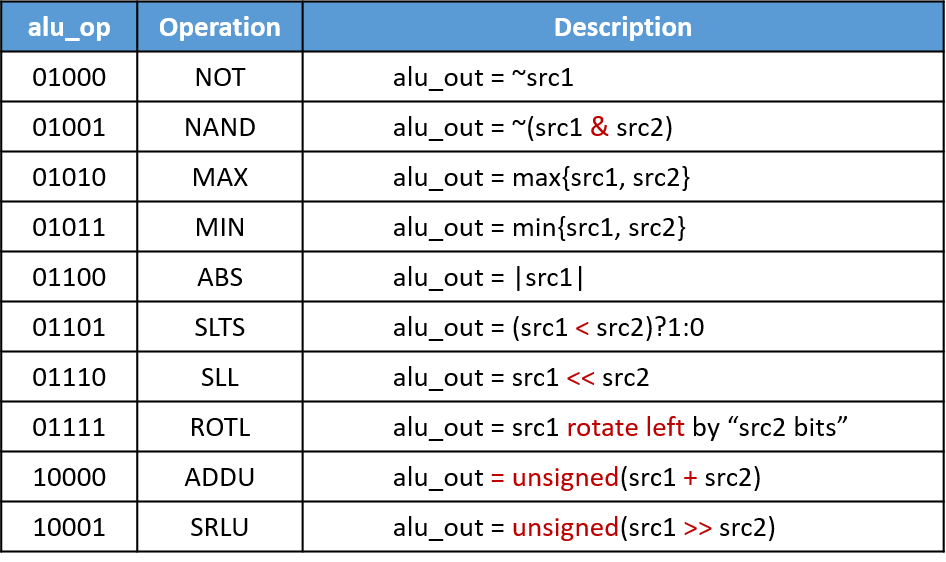
**Objectives:**

**Learn how to design an ALU and Register file which are two main components in CPU. This homework you are allowed to use behavioral description to speed up your design process.**

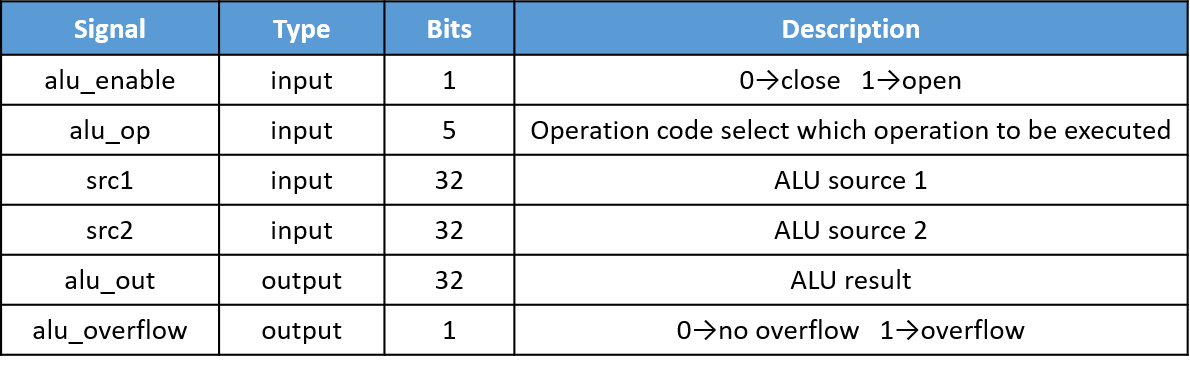
Prob B: Arithmetic Logic Unit



1. Based on the reference code we gave you, please implement another 8 operations which are listed as below.(signed : 2’s complement)



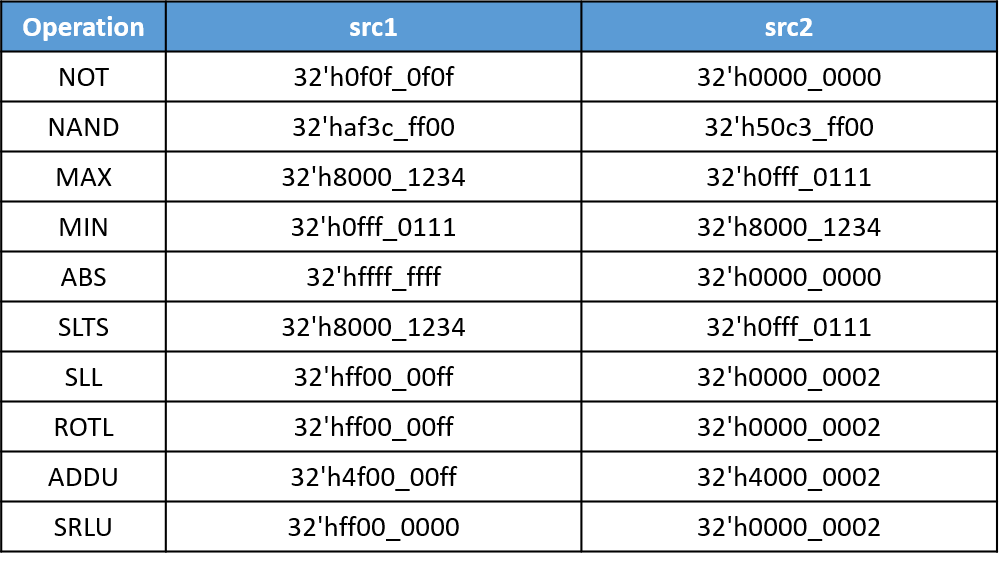
1. Port list



1. You should follow the file name rules as follow.

* ALU (provided!)
  + File name: **ALU.v**
  + Module name: **ALU**
* ALU testbench
  + File name: **ALU\_tb.v**
  + Module name: **ALU\_tb**

1. You should verify your code by the following test patterns.



1. You need to snapshot the waveform of the 10 operations you implemented and explain why they are correct. If you didn’t explain you will get no credit!
2. **Please attach your design waveforms.**

|  |
| --- |
| Your simulation result on the terminal  ( You should $monitor the I/O port like the example TB code does! ) |
| 一張含有 文字 的圖片  自動產生的描述一張含有 文字 的圖片  自動產生的描述  一張含有 文字 的圖片  自動產生的描述 |
| Your waveform |
|  |
| Explanation of your waveform |
| Frankly, I wondered [overflow condition on opcode SLL](https://stackoverflow.com/questions/26224684/guaranteeing-negative-result-when-left-shifting-a-negative-number-in-twos-compl#:~:text=Shifting%20left%20may%20trigger%20arithmetic,the%20sign%20bit%20(MSB).&text=Shifting%20left%20by%20m%20bits,the%20number%20by%202%5Em%20) and TA said overflow on SLL is relatively unimportant to my response. All the function is nothing new and is similar to the example code TA’s typed on top of my code. It’s worth mentioning that the last opcode ADDU, I use concatenation operator to get carry value , when this extra bit, carry is 1, overflow detection signal will set to 1 and vice versa. |
| SuperLint Coverage |
|  |

Prob C: Design a circuit “grayscale conversion”



1. **Design your Verilog code with the following specifications:**

This conversion unit changes 24-bit RGB values into 8-bit grayscale values. The operation is as follows:

y = 0.3125r + 0.5625g + 0.125b

where r, g, b are the values of segments of the 24-bit input respectively, and y is the 8-bit output.

* 1. Number format: unsigned numbers.
  2. Name your design file *grayscale.v* and your testbench file *grayscale\_tb.v.*
  3. The frame code and testbench are given. Use the testbench to help you verify the result. You don’t need to write another testbench in this problem.
  4. Inputs: **color[23:0]**
  5. Outputs: **gray[7:0]**
  6. Signal **color** represents the RGB value**.**
  7. Signal **gray** represents the grayscale value.
  8. The result of the operation should round to an integer. If you find some of your result has a slight difference, like 1, from the expected, it is probable that the rounding is not correct.
  9. Do not use “\*” (multiply) directly in your code.

1. **Please attach your design waveforms.**

|  |
| --- |
| Your simulation result on the terminal  ( You should $monitor the I/O port like the example TB code does! ) |
| 一張含有 文字 的圖片  自動產生的描述 |
| Your waveform |
|  |
| Explanation of your waveform |
| First, put the decimal coefficient into fraction, i.e.    Second, use wire with larger bit space to store decimal place that is result of shifing bit from first step formula.  Finally, we have to round the number to fit the output . In fact, I had a hard time that how to round binary numbers and reading this [article](https://indepth.dev/posts/1017/how-to-round-binary-numbers), I learned the basic rule of rounding is that **rounding to the nearest or finding shortest distance** in other word. In binary, it will be   |  | | --- | | The general rule when rounding to the n-th place prescribes to check the digit following the n-th place in the number. If it’s 0, then the number should always be rounded down. If, instead, the digit is 1 and any of the following digits is also 1, then the number should be rounded up. If, however, all of following digits are 0’s, then a tie breaking rule must be applied and usually it’s the . This rule says that we should round to the number that has 0 at the n-th place. | | It’s supposed to be programmed as conditional statement ……(omit)  Final step is check the result, look into waveform, first one and second one is extreme case, 0 and ffffff and fortunately, there is no saturation case caused by carry. That is the formula is correct and I also check it by myself.   |  | | --- | | 2-bit shift 001111111100 | | 4-bit shift 000011111111 | | 1-bit shift 011111111000 | | 4-bit shift 000011111111 | | **+)** 3-bit shift 000111111110 | | 111111110000 |   The last four numbers from right side is 0 and thus it won’t carry bit. | |
| SuperLint Coverage |
|  |

**At last, please write the lessons learned from this lab session, or some suggestions for this lab session.**

Compare to previous homework, it is more challenging and interesting this time. As a matter of fact, I did a foolish coding error in ABSop in ProbB, if statement should be . Instead, I typed . As C#, index of array starts fromin Verilog. Unfortunately, I forgot it and keep debugging for a whole time. I’ve learnt this lesson in order to reduce my time on assignment. After all, efficiency also matters in coding!

Appendix A : Commands we will use to check your homework

|  |  |  |
| --- | --- | --- |
| **Problem** |  | **Command** |
| **ProbA** | Compile | % ncverilog mux8to1.v |
| Simulate | % ncverilog mux8to1\_tb.v +define+FSDB +access+r |
| **ProbB** | Compile | % ncverilog ALU.v |
| Simulate | % ncverilog ALU\_tb.v +define+FSDB +access+r |
| **ProbC** | Compile | % ncverilog grayscale.v |
| Simulate | % ncverilog grayscale\_tb.v +define+FSDB +access+r |