Computer Organization Project
Lab2 – ALU & Shifter

1. 目標 Goal

在單元二中，請同學以 Verilog 語言實作一個算術邏輯單元(Arithmetic Logic Unit, ALU)和移位器(Shifter)，以了解如何設計 function unit 來符合處理器所支援的指令。此二電路均須設計成組合電路(combinational circuit)，並請採 gate-level 的 Verilog 描述方式。在之後的硬體設計 Lab 中，仍會用到 ALU 以及 shifter，因此本次所設計出的模組可以沿用到以後的實驗單元。

In Lab 2, we are going to implement an ALU (Arithmetic Logic Unit) and a shifter by Verilog. Through this lab, students will learn how to design the function unit of a processor to support its instruction set. Note that you should design these circuits in gate-level as combinational logic instead of sequential logic. The ALU and shifter designed in this lab may also be used in the succeeding lab units.

2. 實驗說明 Lab Description

A. 附加檔案 Attached files

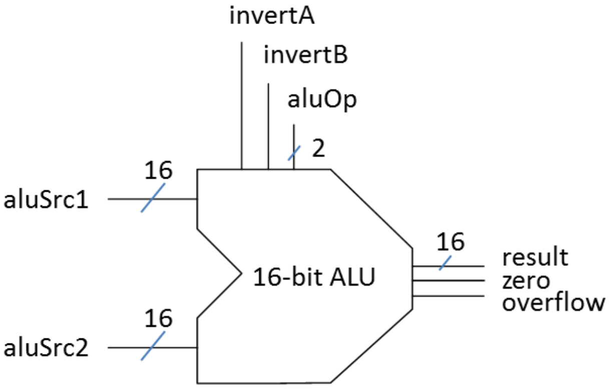
本單元所附的 Lab2.zip 中，包含基本的電路模組 ALU.v、ALU_1bit.v、Full_adder.v、Shifter.v，測試模組 t_ALU.v、t_Shifter.v；以及加分題所需的電路模組 ALU_adv.v、Shifter_Barrel.v，測試模組 t_ALU_adv.v、t_Shifter_Barrel.v。在實作中，如有需要，可新增額外的模組(.v 檔)。

The attached file“Lab2.zip”is composed of the basic circuit modules (ALU.v”, “ALU_1bit.v”, “Full_adder.v”, “Shifter.v”), advanced circuit modules (ALU_adv.v and Shifter_Barrel.v) and the testbenches (“t_ALU.v”, “t_Shifter.v”, “t_ALU_adv.v” and “t_Shifter_Barrel.v”). Please use these files to accomplish the design of your ALU and shifter and to test your design. You may create additional module (.v file) for your design, if necessary.

B. 算術邏輯單元 Arithmetic Logic Unit (ALU)

一個完整的 ALU 必須能處理指令集中所需要的算術與邏輯運算。圖一為本次 Lab 中將設計之 16-bit ALU 的方塊圖，表一則為其需處理的基本運算與相對應之控制訊號。

A well-designed ALU should be able to handle all the arithmetic and logic operations included in the instruction set of a processor. The block diagram of the 16-bit ALU for this lab is shown in Figure 1, and the operations and corresponded control signals for this ALU are described in Table 1.



圖一 (Figure 1)：16-bit ALU

表一 (Table 1)：ALU 之控制訊號及輸出 (Control signals and the output of ALU)

Operation	Control line			Value of result
	invertA	invertB	aluOP	
addition	0	0	10	aluSrc1 + aluSrc2
subtraction	0	1	10	aluSrc1 – aluSrc2
and	0	0	00	aluSrc1 & aluSrc2
or	0	0	01	aluSrc1 | aluSrc2
nor	1	1	00	~(aluSrc1 | aluSrc2)
set on less than	0	1	11	(aluSrc1 < aluSrc2) ? 1 : 0

各個運算除了要產生 16-bit result 外，還需要產生 overflow 以及 zero 這兩條輸出訊號。以下是這兩條輸出訊號的說明：

- zero : 為 1-bit 的輸出訊號；當運算結果為 0 時該值為 1，反之為 0。
- overflow: 為 1-bit 的輸出訊號；當 ALU 中的加減法運算結果產生溢位時為 1，反之為 0。

Besides computing “result”, generate two output signals which are described as follows:

- zero: A 1-bit output control signal. It is set to 1 when the computing result of ALU, “result”, is 0; else, is clear to 0.
- overflow: A 1-bit output control signal. It is set to 1 when the computing result of ALU is overflow; else, is clear to 0.

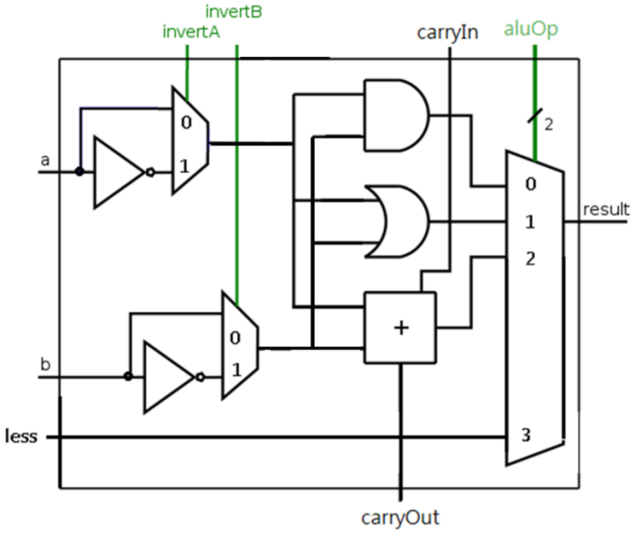
請注意，表一中的每一道運算皆為 16-bit 的運算。因此，需要用 16 個 1-bit ALU 來組出此 16-bit ALU。

Note that all operations in Table 1 are 16-bit operations. Therefore, in this lab, you should implement a typical 1-bit ALU first and then build the 16-bit ALU by 16 1-bit ALUs.

a. 1-bit ALU

圖二為本次 Lab 所需設計的 1-bit ALU 之電路圖：

The logic diagram of a typical 1-bit ALU to be implemented in this lab is shown in Figure 2.



圖二 (Figure 2)：1-bit ALU

ALU_1bit.v 之各輸入、輸出訊號的定義如下：

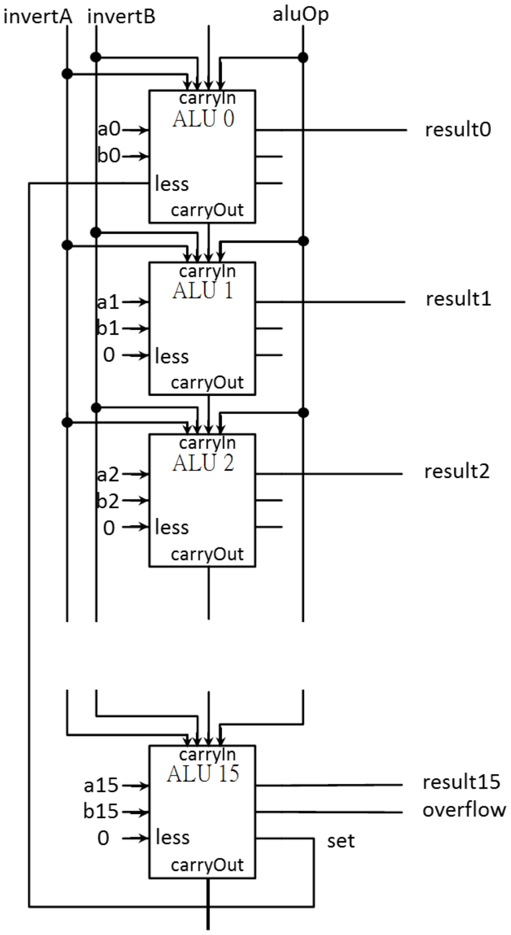
- a: 為 1-bit 的輸入資料，作為運算的第一個運算子。
- b: 為 1-bit 的輸入資料，作為運算的第二個運算子。
- invertA: 為 1-bit 的輸入控制訊號線；其值為 1 時，會將 a 的 invert 值當作運算的輸入資料。
- invertB: 為 1-bit 的輸入控制訊號線；其值為 1 時，會將 b 的 invert 值當作運算的輸入資料。
- aluOp: 為 2-bit 的輸入控制訊號線，用來決定輸出那個運算結果。
- less: 為 1-bit 的輸入資料，用來設定“set on less than”運算的輸入資料。
- carryIn: 為 1-bit 的輸入資料，為全加器的進位輸入。
- carryOut: 為 1-bis 的輸出資料，為全加器的進位輸出。
- result: 為 1-bit 的輸出資料，為此 1-bit ALU 的運算結果。

ALU_1bit.v contains the following inputs and outputs:

- a: A 1-bit input data, which represents the 1st operand of the 1-bit ALU.
- b: A 1-bit input data, which represents the 2nd operand of the 1-bit ALU.
- intvertA: A 1-bit input control signal; when it is set to 1, select the complement of “a” as operand.
- intvertB: A 1-bit input control signal; when it is set to 1, select the complement of “b” as operand.
- aluOp: A 2-bit input control signal, which controls the selection of the result of the 1-bit ALU.
- less: A 1-bit input data, is used to support operation “set on less than”.
- carryIn: A 1-bit input data, is used as the value of “carry in” for the full adder.
- carryOut: A 1-bit output data, is the value of “carry out” of the full adder.
- result: A 1-bit output data, which represents the computation result of the 1-bit ALU.

b. 16-bit ALU

圖三為本次 Lab 所需設計的 16-bit ALU 的方塊圖。此模組之檔案名稱為 ALU.v。
The block diagram of the 16-bit ALU to be implemented is shown in Figure 3. The file name of this module is ALU.v.



圖三 (Figure 3)：16 個 1-bit ALU 之連結 (Connection between 16 1-bit ALUs)

如圖三所示，16-bit ALU 之輸入訊號以及資料都會各自輸入到 1-bit ALU 之中。而 carryIn 以及 carryOut 則用以串接各個 1-bit ALU。

As shown in Figure 3, each input signal of the 16-bit ALU will be connected to the corresponding input of a 1-bit ALU. Moreover, carryOut of ALU$_i$ will be connected to carryIn of ALU$_{i+1}$.

ALU15 的設計與典型 1-bit ALU 略有不同，新增兩條訊號線之說明如下；

- set: 為 1-bit 的輸出訊號線，將由 ALU15 回傳至 ALU0。
- overflow: 為 1-bit 的輸出訊號；當其值為 1 時，表示運算產生溢位情形。

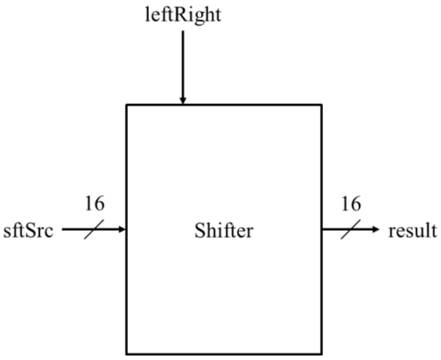
The design of ALU15 is different from that of the other typical 1-bit ALUs.There are two additional signals generated by ALU15 and are described as follows:

- set: A 1-bit control line, is generated by ALU15 and send to ALU0.
- overflow: A 1-bit output control signal, is set to 1 when overflow occurs.

C. Shifter

圖四為本次 Lab 應設計之 shifter 的方塊圖。表二則為此 Shifter 必須能夠處理的基本移位運算以及相對應之控制訊號。

The block diagram of the shifter to be implemented in this lab is shown in Figure 4, and the basic operations and the corresponded control signals of the shifter t are described in Table 2.



圖四 (Figure 4)：移位器 (Shifter)

表二 (Table 2)：Shifter 之控制訊號及輸出 (Control signals and the output of shifter)

Operation	LeftRight	Value of result
Shift right logical by 1 bit	0	sftSrc >> 1
Shift left logical by 1 bit	1	sftSrc << 1

此移位器為 16-bit shifter，只能執行邏輯左移或右移 1 位。請以組合電路的方式實作之。

This shifter is 16 bits, which may logical left/right shift by one bit position each time. Please implement it as a combinational circuit.

以下是 Shifter.v 之輸入輸出介紹：

- sftSrc: 為 16-bit 的輸入資料，作為被移動的資料。
- leftRight: 為 1-bit 的輸入控制訊號；其值為 0 時，執行右移，反之則左移。
- result: 為 16-bit 的輸出資料，為 sftSrc 位移後的結果

Shifter.v contains the following inputs and outputs:

- sftSrc: A 16-bit input data, is the source data of the shifter.
- leftRight: A 1-bit input control signal. When it is set to 1, the shifter perform logical left shift; else, does logical right shift.
- result: A 16-bit output data, which represents the shifting result of the shifter.

D. Bonus

以下列出 ALU 以及 Shifter 各一種進階的功能；如完成進階，則可得到額外的分數。

Additional functions for ALU and shifter are described in the following subsection with bonus.

a. Advanced ALU (5%)

在 ALU_adv.v 中修改“2. B”之設計，使其在執行“set on less than”時，不致因發生溢位而產生錯誤結果。所需之輸出入訊號與“2.B”中所定義的相同。

Modify your design of the basic 16-bit ALU (in section 2. B) to prevent the error when performing operation “set on less than” while overflow occurs, and save the modified code in “ALU_adv.v”. The input and output signals are the same as that of the basic ALU.

以下列運算為例，假設我們要對 27500 和-16000 做 set on less than：

(27500_{10} = 0110101101101100_2)
(-16000_{10} = 1100000110000000_2)

預期結果之輸出應為 0，因為 27500 比-16000 大；但實際上卻可能因為溢位問題而產生輸出為 1 的結果。

For example, perform “set on less than” for 27500 and -16000
(27500_{10} = 0110101101101100_2)
(-16000_{10} = 1100000110000000_2)
The correct result should be 0 (because 27500 is greater than -16000). However, the basic ALU might outputs 1 when overflow occurs.

b. Advanced shifter (20%)

在 Shifter_Barrel.v 中實作一個可左右位移任意距離 (0~15 bit positions) 之 logical shifter (稱為 Barrel shifter)，如圖五所示。所需之輸出入訊號為：

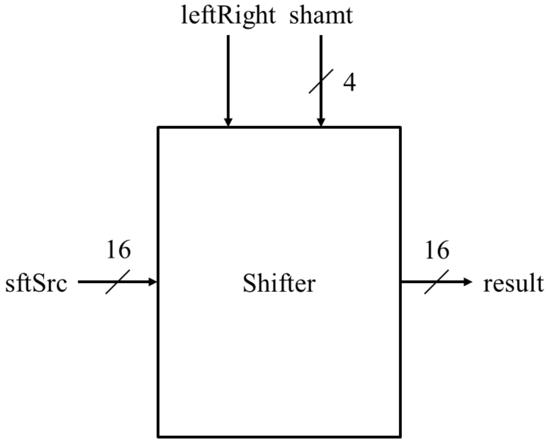
- sftSrc: 為 16-bit 的輸入資料，作為被移動的資料。
- leftRight: 為 1-bit 的輸入控制訊號；其值為 0 時，執行右移，反之則左移。
- shamt: 為 4-bit 的輸入資料，shifter 會根據這個值決定將 sftSrc 移動幾位。
- result: 為 16-bit 的輸出資料，為 sftSrc 位移後的結果。

※禁止預先產生 32 種可能的移位結果後，再用多工器去選擇輸出。

Complete“Shifter_Barrel.v” to implement a 16-bit barrel shifter as shown in Figure 5. It should be able to perform left/right logical shift by arbitrary length (from 0 to 15 bit positions). The input and output signals of the barrel shifter are as follow:

- sftSrc: A 16-bit input data, is the source data of the shifter.
- leftRight: A 1-bit input control signal. When it is set to 1, the shifter perform logical left shift; else, does logical right shift.
- shamt: A 4-bit input data, represents the number of bit positions to be shifted.
- result: A 16-bit output data, which represents the shifting result of the shifter.

※It is forbidden to use multiplexers to choose the result from the pre-generated 32 possible shifting results.



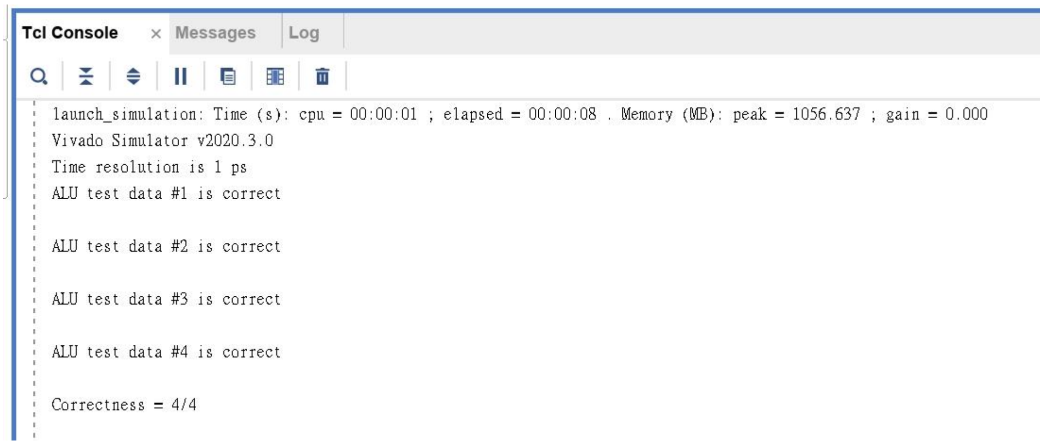
圖五 (Figure 5)： Barrel 移位器 (Barrel shifter)

3. Test Bench

記得把對應的 txt 檔加入 Simulation source。

Add corresponding txt files to Simulation source.

可以從視窗下半部的「Tcl Console」看到從*$display* 印出的訊息
You can find the message from *$display* at the 「Tcl Console」



```
launch_simulation: Time (s): cpu = 00:00:01 ; elapsed = 00:00:08 . Memory (MB): peak = 1056.637 ; gain = 0.000
Vivado Simulator v2020.3.0
Time resolution is 1 ps
ALU test data #1 is correct

ALU test data #2 is correct

ALU test data #3 is correct

ALU test data #4 is correct

Correctness = 4/4
```

倘若想要執行其他的測資，可更改 TestBench 中測資輸入檔的檔名；例如：ALU 的測試資料輸入檔如為 ALU_test1.txt，則在 testbench 中設定 *`define test_file_ALU "**ALU_test1.txt**"*。
You may change the path of the input data file in testbench and recompile the testbench module, if you want to perform other test. For example, if the input data file of ALU is ALU_test1.txt, the setting of the path is *`define test_file_ALU "**ALU_test1.txt**"* .

4. 測試方式 Test Method

作業繳交後，助教將會用類似的 testbench 所編譯的模組去連結 ALU.v 以及 Shifter.v 編譯出的模組(ALU&Shifter)來進行模擬測試。模擬中，會讀入不同的測試數據以驗證同學們程式的正確性。如果有實作加分題(ALU_adv.v, Shifter_Barrel.v)，也會作類似的測試。

After you hand in your code, TA will use the similar TestBench module and different test data to verify the correctness of your design of ALU and shifter.If you have implemented the advanced ALU and barrel shifter, they will also be tested by similar methods.

同學如果有新增其他的檔案，請確保其編譯出的模組不會影響 TestBench 對 ALU 或是 Shifter 的讀取。

If you have attached additional modules for your design, do ensure that those modules would not affect our testing.

- 以下是 TestBench 所讀取之 ALU/ALU_adv 測試資料的格式：
  (invertA)(invertB)(aluOP)(aluSrc1)(aluSrc2)
  例如：TestBench 讀取到
  0 0 10 0000000000000010 0000000000000011
  意即 *addition 2 3* 。

  The format of the test data for ALU/ALU_adv is shown as follows:
  (invertA)(invertB)(aluOP)(aluSrc1)(aluSrc2)
  For example:
  0 0 10 0000000000000010 0000000000000011

means *addition 2 3* .

- 以下是 TestBench 所讀取之 Shifter 測試資料的格式：

  (leftRight)(sftSrc)

  例如：TestBench 讀取到

  1 0000000000001001

  意即 *shift 9 left by 1* 。

  The format of the test data for Shifter is shown as follows:

  (leftRight)(sftSrc)

  For example:

  1 0000000000001001

  means *shift 9 left by 1* .

- 以下是 TestBench 所讀取之 Shifter_Barrel 測試資料的格式：

  (leftRight)(shamt)(sftSrc)

  例如：TestBench 讀取到

  0 0011 0000000000001001

  意即 *shift 9 right by 3* 。

  The format of the test data for Shifter_Barrel is shown as follows:

  (leftRight)(shamt)(sftSrc)

  For example:

  0 0011 0000000000001001

  means *shift 9 right by 3* .

5. 注意事項 Notice

a. 請用 Vivado 2020.3(推薦)或 iverilog 做為開發環境。(硬體描述語言模擬器之安裝與設定請參閱 Lab2-0-Vivado_iverilog_gtkwave 說明文件)
Develop your lab in Vivado 2020.3 (recommended) or iverilog. (Refer to Lab2-0-Vivado_iverilog_gtkwave for the simulator of hardware description language, if necessary.)

b. 請務必使用附件提供的「ALU.v」、「ALU_1bit.v」、「Full_adder.v」、「Shifter.v」、「ALU_adv.v」、「Shifter_Barrel.v」及對應的 TestBench 來完成作業。
Use “ALU.v”, “ALU_1bit.v, “Full_adder.v”, “Shifter.v”, ALU_adv.v”, “Shifter_Barrel.v”, and corresponding testbench provided to implement your ALU and shifter.

c. **除非有特別說明，否則請勿更改**任何現有的程式碼以及變數名稱 (TestBench 中的測資檔名除外)。可以在 Testbench 增加 display 項目幫忙 debug。
Do not modify any existing code and variable names except the input filename in testbench.Additional *$display* items are accepted for debugging.

d. 請將你的程式碼寫在 module 中標示「/*your code here*/」的位置。
Write your code in “/*your code here*/” in each module.

e. **禁止使用順序邏輯**來設計本實驗單元之電路，違者 0 分計算！

Do not design the ALU or the shifter as a sequential circuit! Zero score will be given for sequential ALU or shifter.

6. 作業報告 Report

請回答以下問題：

Answer the following questions:

Q1: 請簡略說明你所新增的檔案以及其功能。倘若你沒有新增檔案，則請簡略說明你的 ALU.v 中設計了哪些電路。

Describe your additional file and its circuit briefly. If you have not attached additional file for ALU design, describe your ALU.v.

Q2: 假設在 ALU 設計中不需要實作減法運算，請問哪幾條控制訊號可以刪除?假設 ALU 設計中不需要 NOR 運算，請問哪幾條控制訊號可以刪除?

Which control lines could be deleted if “subtraction”operation is not required in the design of our ALU? If “nor” operation is not required, which control lines could be deleted?

Q3: 下表為各種邏輯閘及電路之 propagation delay：

The following table shows the propagation delay of each gate/unit:

	Propagation delay (in *ps*)
and	10
or	10
not	10
xor	20
2-to-1 mux	20
4-to-1 mux	40
Full adder	40

試算出你所設計的 16-bit ALU 之 propagation gate delay，並說明之。

Calculus the propagation gate delay of the basic 16-bit ALU designed by you, and explain your answer briefly.

Q4: 請寫出你在 Lab2 中所碰到的困難，以及實作完成後的心得。

Write down your comment for Lab2. If you have any difficulty while implementing the function units in this lab, express it also in your report.

Q5: (若有作 ALU 加分部分，請回答此題) 請簡單說明你是如何修改 ALU，使在執行“set on less than”時，不致因發生溢位而產生錯誤結果。

(Answer Q5 if you have implemented the advanced ALU) Describe your design for the advanced ALU.

Q6: (若有作 Shifter 加分部分，請回答此題) 請以你設計 Barrel shifter 電路的方法，

畫一張 4-bit Barrel shifter 的電路圖。
(Answer Q6 if you have implemented the Barrel shifter) Draw the logic diagram for a 4-bit barrel shifter by the same method used for the 16-bit barrel shifter designed by you.

7. 評分方式 Grading

模組與註解(80%)：助教會以多筆測資測試你的基本 ALU 與 Shifter 模組，並檢查你的模組是否符合作業要求。若有答錯或未符合作業要求，則視程式完成度斟酌給分。
Verilog modules and Comments (80%): We will test many test cases for grading.

報告(20%)：回答作業報告中所列的 Q1~Q4 四個問題。請依照自己的想法回答，回答只需正確或合乎邏輯即可得到分數。
Report (20%): You should answer the four (Q1~Q4) questions mentioned in "**6. Report**", and you will get the points if your answer is correct or reasonable.

加分(25%)(含模組與註解)：在作完進階電路後，必須回答作業報告中的 Q6 與 Q7。根據電路與問題回答情形，斟酌給分。

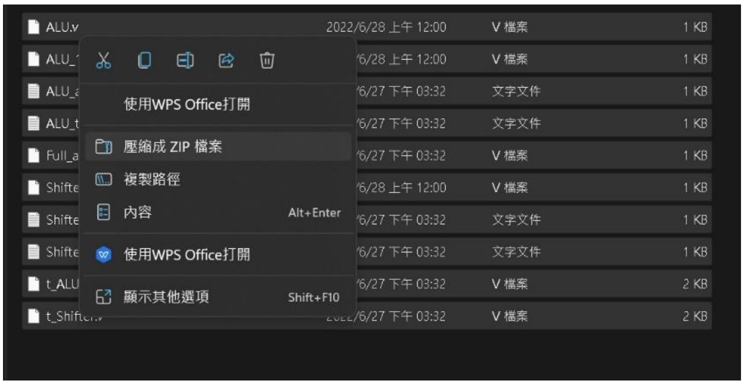
Bonus (25%)(including Verilog modules and Comments): If you have implemented the advanced ALU and shifter for bonus, please answer question Q5 and Q6, repeatedly, in your report. You will get bonus if your circuit design and question answer are good.

8. Deadline

a. 本實驗單元為一人一組，請將實驗報告及相關 Verilog 檔案上傳至 e3 平台。
This lab unit is one student per group. Please upload your Lab Report (pdf file) and the corresponding HDL code (.v files) onto e-Campus platform.

- **實驗報告**：pdf 檔，請命名為 **Lab2_學號**，如：「Lab2_110550600」。
Hand in the lab report as a pdf file, named **Lab2_StudentID**, for example, "Lab2_110551600".
- **Verilog modules 檔案**：
繳交下述 Verilog 模組檔案：
Hand in the following Verilog modules:
ALU.v、ALU_1bit.v、Full_adder.v、Shifter.v、t_ALU.v、t_Shifter.v、
ALU_adv.v、Shifter_Barrel.v、t_ALU_adv.v、t_Shifter_Barrel.v、
Any other verilog files you create
- 請將上述作業 pdf 檔及 Verilog 電路模組與測試模組檔案(.v)全部壓縮成一個 zip 檔 (禁止上傳 rar 檔或是其他檔案格式)請將檔案全選壓縮，勿多包一層資料夾，也不要繳交整個 project 檔案，並以「Lab2_學號」的方式命名，如：「Lab2_110551600」。檔案繳交格式錯誤(包含壓縮方式以及 pdf)將酌扣 10 分作為懲罰。
Please compress the pdf file of lab report and the Verilog circuit modules and testbench described above all into one zip file (rar file or other format is not accepted). Please compress your files only, without an additional directory. Also do not submit your project files. Name the zip file as "Lab2_StudentID", for example, "Lab2_110551600". Wrong format, including zip and pdf files, would lead to 10

points penalty.
Select all source files, report pdf and compress



ALU.v 2022/6/28 上午 12:00 V 檔案 1 KB
ALU_ /6/28 上午 12:00 V 檔案 1 KB
ALU_ /6/27 下午 03:32 文字文件 1 KB
ALU_t /6/27 下午 03:32 文字文件 1 KB
Full_a /6/27 下午 03:32 V 檔案 1 KB
Shifte /6/28 上午 12:00 V 檔案 1 KB
Shifte /6/27 下午 03:32 文字文件 1 KB
Shifte /6/27 下午 03:32 文字文件 1 KB
t_ALU /6/27 下午 03:32 V 檔案 2 KB
t_Shifter.v 2022/6/27 下午 03:32 V 檔案 2 KB
使用WPS Office打開
壓縮成 ZIP 檔案
複製路徑
內容 Alt+Enter
使用WPS Office打開
顯示其他選項 Shift+F10

b. 繳交日期為 2022/7/24 (日) 23:59，準時繳交者可獲得 10%的額外加分；最後繳交期限為 2022/8/15 (一) 23:59，之後不接受逾期繳交。
The due date is 2022/7/24 (Sunday) 23:59 and you will get extra 10% points for on-time award. Deadline is 2022/8/15 (Monday) 23:59, no late hand-in is allowed.

c. 禁止抄襲，違者(抄襲者與被抄襲者)以 0 分計算。
Any assignment work by fraud will get a zero point.