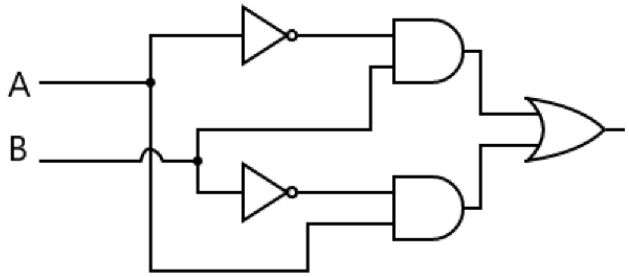
EECS 2070 02 Digital Design Labs 2019 Lab 1
學號：107062314　姓名：陳柏均

1. 實作過程

這次的 lab 主要要我們實作 ALU 的問題,我覺得這次 lab 不只幫我複習了 verilog 的主要語法，更是幫助我訓練嚴謹的邏輯思考。前三題的 testbench 皆相同，但卻分別用了 gate level, dataflow level 以及 behavior level, 由於有一段時間沒又去碰 verilog 了，一開始有一點生疏，但在經過認真思考和查資料後，慢慢地才拾回打 code 的熟悉感。

第零題我覺得真的出的很好，畢竟後面要用到，很感謝有這題並且提供了 testbench，雖說這題很簡單也不是主要要學習的部分，但我認為這題很關鍵地讓我熟悉了 gate level 的用法以及如何看著圖拼出一個自己想要的 module，我相信邏輯閘的用法和連結在硬體中是不可或缺的部分,而這題雖是基礎,但卻是十分重要且必要的。

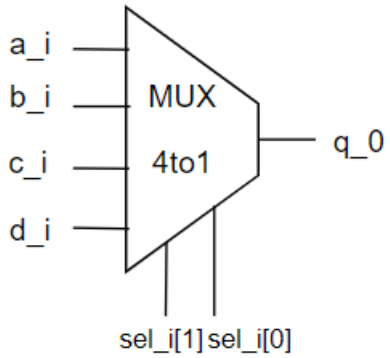


(註:第零題我選擇了用兩個 not,兩個 and 及一個 or 拼出 xor)

```
module myxor(out,a,b);
    input a,b;
    output out;
    wire and1,and2;
    wire nota,notb;
    not not_1(nota,a);
    not not_2(notb,b);
    and and_1(and1,nota,b);
    and and_2(and2,notb,a);
    or or_1(out,and1,and2);
endmodule
```

(註:用 and,or,not 所拼出的 myxor)

第一題我用了四個 module 去計算然後在主要的 module(lab1_1)用「one, two, three, four」這幾個 wire 去記住四種題目條件分別算出來的答案，然後 aluctr 再去選出應該給的答案。而在選的部分,我便是利用了助教所給的 mux4_to_1 去做選擇，如下圖。



(註:mux4_to_1 示意圖)

而由於 q_0 只有一個 bit，因此我用了兩次的 mux4_to_1 分別去判斷及給 d,e 值；另外在這一題的加法中我也使用了之前在邏輯設計課所學的 fulladder 去進行相加的動作。

```
first first_1(.out(one),.a(a),.b(b),.c(c));
second second_1(.out(two),.a(a),.b(b));
third third_1(.out(three),.a(a),.b(b));
fourth fourth_1(.out(four),.a(a),.b(b));
mux4_to_1 choose_1(.q_0(d),.a_i(one[0]),.b_i(two[0]),.c_i(three[0]),.d_i(four[0]),.sel_i(aluctr));
mux4_to_1 choose_2(.q_0(e),.a_i(one[1]),.b_i(two[1]),.c_i(three[1]),.d_i(four[1]),.sel_i(aluctr));
```

(註:用 MUX 去選擇的程式)

第二題我自己認為比第一題直覺多了，我用了 assign 並且使用 verilog 中 「?:」的功能去直接判斷並給值，我認為這樣十分地像在打 C 語言，對於我個人而言，軟體還是比硬體好打，軟體比較像用人類的語言去命令電腦去做甚麼，而硬體語言則比較偏向人類去用電腦的方式思考，但此題我就有一種回到打軟體語言的感覺，不用想太多，直接做自己想要編譯器去跑的程式,如下。

```
module lab1_2(a,b,c,aluctr,d,e);
    input a,b,c;
    input [1:0] aluctr;
    output d,e;
    assign {e,d}=(aluctr[1]==1'b0 && aluctr[0]==1'b0)? a+b+c:
           (aluctr[1]==1'b0 && aluctr[0]==1'b1)? {1'b0,a&b}:
           (aluctr[1]==1'b1 && aluctr[0]==1'b0)? {1'b0,!(a|b)}:
           {1'b0,a^b};
endmodule
```

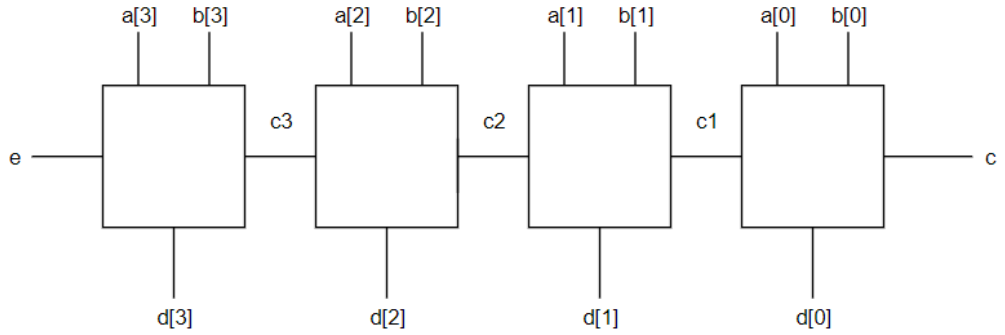
(註:直接使用 ?: 去判斷及給值)

第三題我用了 always block 並且在之中使用了 if else 去給值, 跟上一題的解法差不多, 主要差別是在 always 裡面我才可以使用 if else 這一個功能,

```
if(aluctr[1]==1'b0 && aluctr[0]==1'b0)
    begin
        {e,d}=a+b+c;
    end
```

(註:lab1_4 跟上一題差不多,只有差在 always 及 if else 的使用方式)

第四題的題目有稍做修正,跟前三題明顯不同, 但只要仔細的思考,便可以想出此題要使用串接的方式去做 4 個 bits 的題目, 就舉加法而言第一個結果的 cout 是第二個的 cin 而以此類推。



(註:lab1_4 想法示意圖)

```
lab1_2 alu0(.a(a[0]),.b(b[0]),.c(c),.aluctr(aluctr),.d(d[0]),.e(c1));
lab1_2 alu1(.a(a[1]),.b(b[1]),.c(c1),.aluctr(aluctr),.d(d[1]),.e(c2));
lab1_2 alu2(.a(a[2]),.b(b[2]),.c(c2),.aluctr(aluctr),.d(d[2]),.e(c3));
lab1_2 alu3(.a(a[3]),.b(b[3]),.c(c3),.aluctr(aluctr),.d(d[3]),.e(e));
```

(註:使用 lab1_2 進行指令的動作及運算,而這個 module 只負責做串接)

Bonus1 一開始我看了很久搞不清楚要幹嘛，後來才發現只要將之前的 code 稍微改一下並加上一個判斷式去給值就好，而 bonus2 則是依照 lab1_4 依樣的方法去做。

```
else if(aluctr[1]==1'b1 && aluctr[0]==1'b0)
    begin
        d=1'b0;
        if(a>b) e=1'b1;
            else if(a<b) e=1'b0;
        else
            begin
                if(c==1'b0) e=1'b0;
                else e=1'b1;
            end

    end
```

(註解:bonus1 使用 lab1_3 依題目要求稍做修改)

2. 學到的東西與遇到的困難

(i)reg 使用的問題

在 lab1_3 中,我一度忘記在 always block 之中左邊的值必須要為 reg，而產生了不該犯的錯誤, reg 跟 wire 的用法無疑是 verilog 之中一個很重要的觀念,而經過數次的犯錯,希望下次能更快發現錯誤或是說不會再遇到這樣的語法不正確。

```
Error: procedural assignment to a non-register e is not permitted, left-hand side should be reg/integer/time/genvar
Error: procedural assignment to a non-register d is not permitted, left-hand side should be reg/integer/time/genvar
```

(註:沒有使用 reg 導致的錯誤)

(ii)給值的錯誤

在寫第二題的時候,本來以為會很快就過了或者是很少錯誤,沒想到丟上 Vivado 編譯後竟然跑出了一大堆 errors，我從 testbench 給的值看了許久還是不懂自己為何而錯。到後來當我再去看一次題目的時候才發現自己 d,e 的值給相反了，原本以為 e 是比較低的那一位,結果是相反的。另外在這裡我也學到了大括號的使用技巧，可以寫成 {d,e} 然後去給值,十分的方便。

```
assign {d,e}=(aluctr[1]==1'b0 && aluctr[0]==1'b0)? a+b+c:
        (aluctr[1]==1'b0 && aluctr[0]==1'b1)? {1'b0,a&b}:
        (aluctr[1]==1'b1 && aluctr[0]==1'b0)? {1'b0,!(a|b)}:
        {1'b0,a^b};
```

```
 20 Error: aluctr = 00, a = 0, b = 1, c = 0, d = 0, e = 1
 30 Error: aluctr = 00, a = 1, b = 0, c = 0, d = 0, e = 1
 40 Error: aluctr = 00, a = 1, b = 1, c = 0, d = 1, e = 0
 50 Error: aluctr = 00, a = 0, b = 0, c = 1, d = 0, e = 1
 60 Error: aluctr = 00, a = 0, b = 1, c = 1, d = 1, e = 0
 70 Error: aluctr = 00, a = 1, b = 0, c = 1, d = 1, e = 0
120 Error: aluctr = 01, a = 1, b = 1, c = x, d = 0, e = 1
130 Error: aluctr = 10, a = 0, b = 0, c = x, d = 0, e = 1
180 Error: aluctr = 11, a = 0, b = 1, c = x, d = 0, e = 1
190 Error: aluctr = 11, a = 1, b = 0, c = x, d = 0, e = 1
```

(註:d,e 給值錯誤導致出現諸多 errors)

(iii)串接的問題

剛開始讀第四題的題目的時候,我其實沒有很懂題目想要表達的意思, 還以為是一個一個 bit 做並且去給 output e 值,後來怎麼想都不合理, 思考了許久後才了解到此題要我所做的事; 同時也發現自己原本想法的荒謬,原本我的做法竟然是把它當作四個獨立的 alu 去給值,而非將其串接起來。

```
lab1_2 alu0(.a(a[0]),.b(b[0]),.c(c),.aluctr(aluctr),.d(d[0]),.e(e));
lab1_2 alu1(.a(a[1]),.b(b[1]),.c(c),.aluctr(aluctr),.d(d[1]),.e(e));
lab1_2 alu2(.a(a[2]),.b(b[2]),.c(c),.aluctr(aluctr),.d(d[2]),.e(e));
lab1_2 alu3(.a(a[3]),.b(b[3]),.c(c),.aluctr(aluctr),.d(d[3]),.e(e));
```

```
Error: aluctr=00a=0001   b=0001   c=0  d=0000   e=x
Error: aluctr=00a=0001   b=0011   c=0  d=0010   e=x
Error: aluctr=00a=0001   b=0101   c=0  d=0100   e=x
Error: aluctr=00a=0001   b=0111   c=0  d=0110   e=x
Error: aluctr=00a=0001   b=1001   c=0  d=1000   e=x
Error: aluctr=00a=0001   b=1011   c=0  d=1010   e=x
Error: aluctr=00a=0001   b=1101   c=0  d=1100   e=x
```

(註: 一開始錯誤的思考方向成了一個 e=x 的結果)

3. 想對老師或助教說的話

個人一直以來其實對硬體語言都不太熟悉, 尤其邏輯設計的課剛開始打 verilog code 的時候,每次看到題目後,我幾乎都是很慌張的狀態。對於剛學完軟體程式的那時候的我而言,這個語言真的是難上加難,常常我重複閱讀了指示卻仍坐在電腦前面, 不知從何下手。對於我而言,軟體真的比較直覺且熟悉多了,我可以想到什麼就打什麼,不用以機器的角度去思考。

但同時我不能反對的是熟能生巧,之前打了數次的 lab, 自己學習能力真的比較差, 很多次都是不斷地詢問助教及同學才能完成的。而這次卻比較不一樣,我發現其實我認真的去思考還是能打出一個成果的,但我相信這當然也是之前所累積下來的結果。

這次的 lab1 我複習了不少 verilog 的應用技巧及學習的一些新的用法, 這次的難度對我個人而言是剛剛好的, 其實也花了滿多時間在 debug 的。這次的題目對大部分人來講應該偏簡單, 我還有很大的進步空間, 相信以後

一定會有更佳有挑戰性的題目，即使難免會擔心打不出來,但我知道這就是學習的過程;沒有犯錯,豈能讓下次的的自己變更好?

我認為這次的出題方式十分地清晰有條理,我很喜歡,幫我複習了許多東西,同時也讓我更加地運用上課所學。未來無可避免的會有許多瓶頸,但我同時也希望透過這些困難及挑戰獲得一定的收穫,讓自己不斷地突破自我。