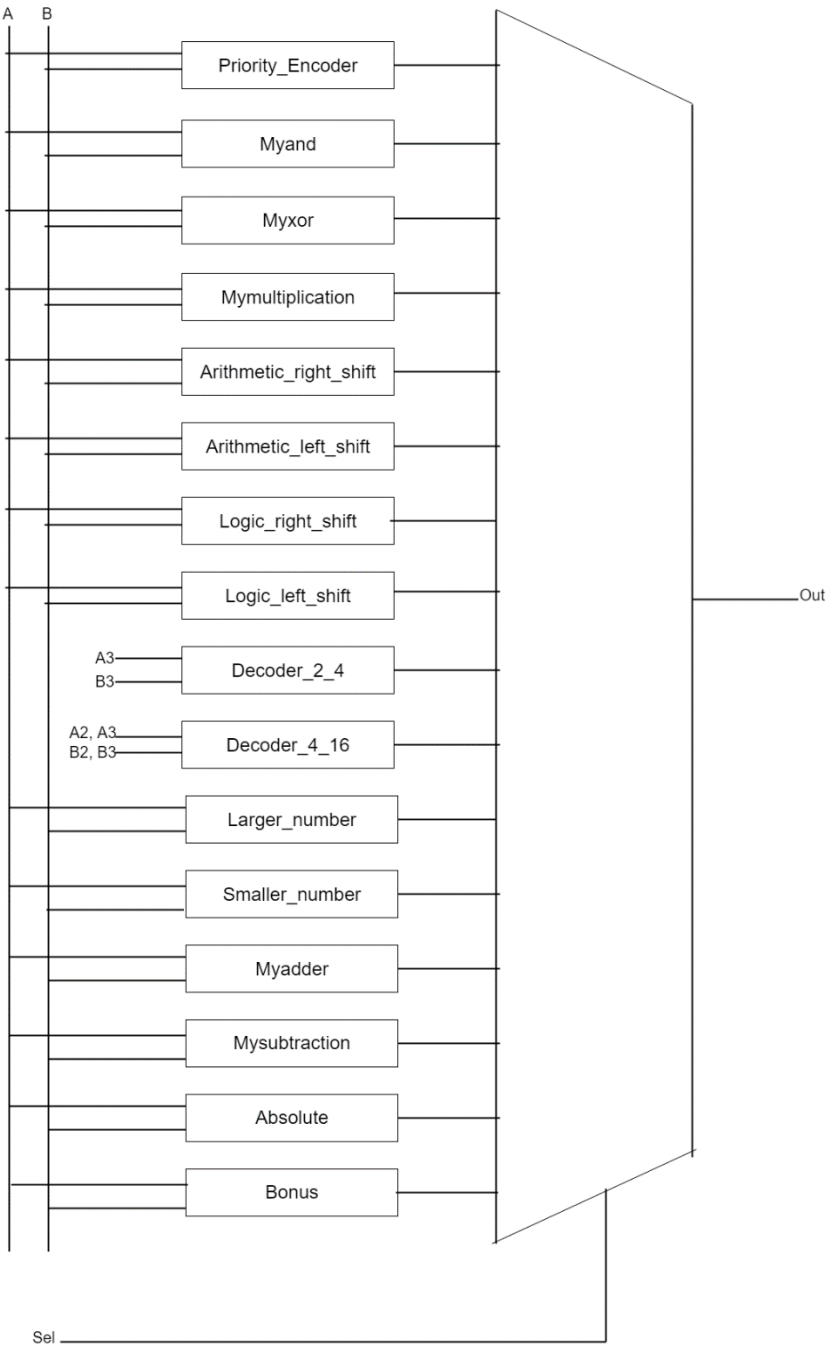
Logic Design

210510210 詹其佐

Lab2

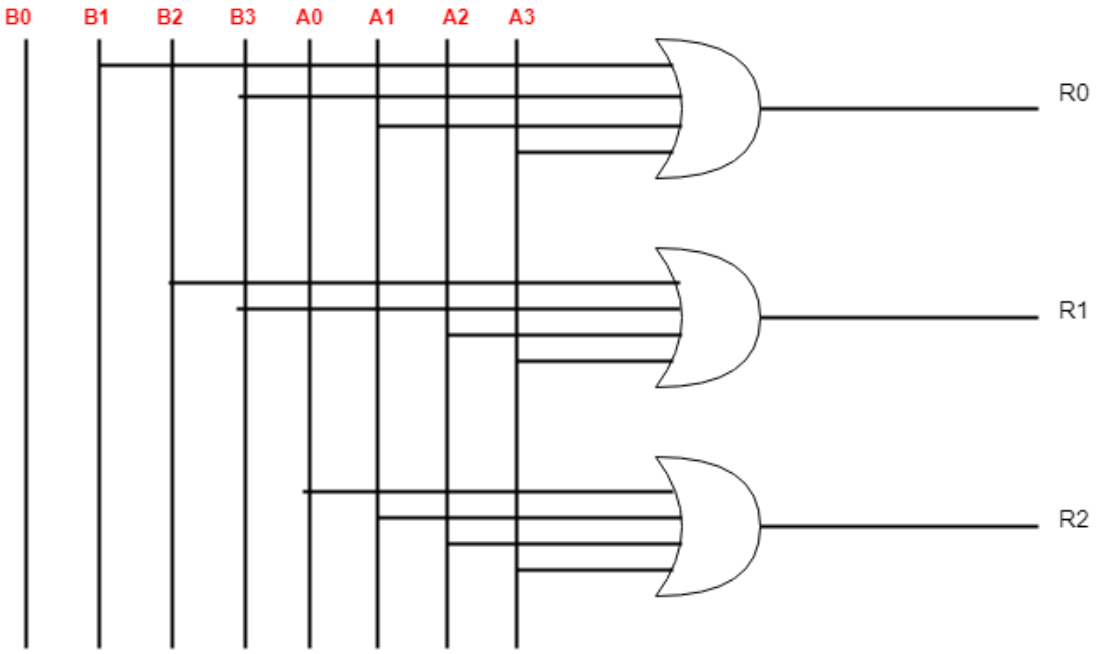
1.Block diagram



這是我這次設計的主要架構，裡面有幾個 **module**

像是 **myand, myxor, mymultiplication, Arithmic_right_shift, Arithmic_left_shift, logic_right_shift, logic_right_shift** 都是用 **verilog** 內建的語法，所以下面會列出幾個在實作的時候有碰到問題的 **module**，或者是比較複雜的 **module**。

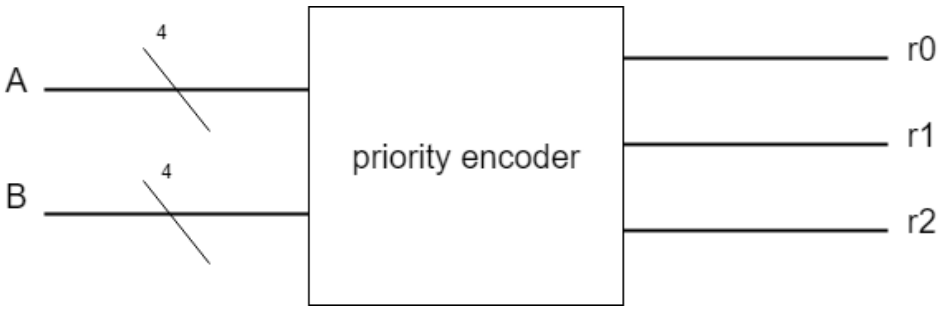
2.Simple_Priority_Encoder



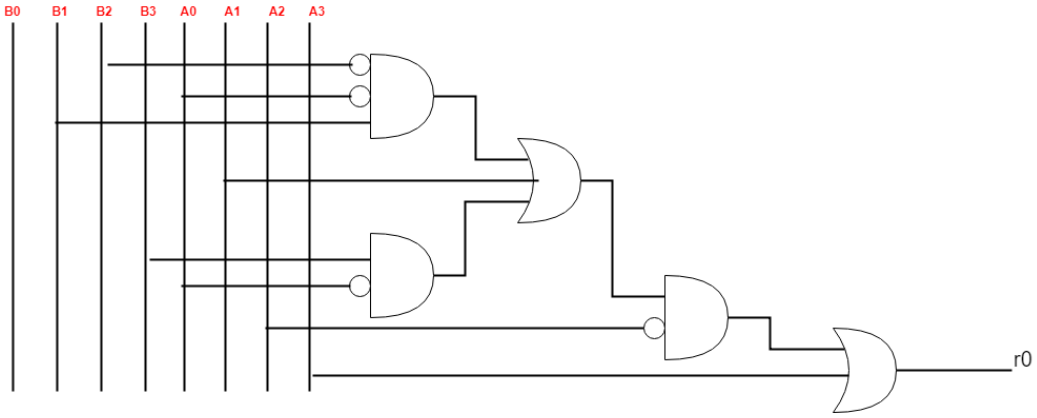
這是我直接用 **TT** 做出來的 **PE**，可是這個只針對傳進來的數只有一個 **1** 其他都是 **0** 的時候有用（根據我觀察的結果），所以我上網查了一下，發現這是簡單優先編碼器。之後我查到解決的辦法就是要把它前

面的 0 也列進去考慮，後面的 **don’t care** 就不用管了。下面是改良後的結果。

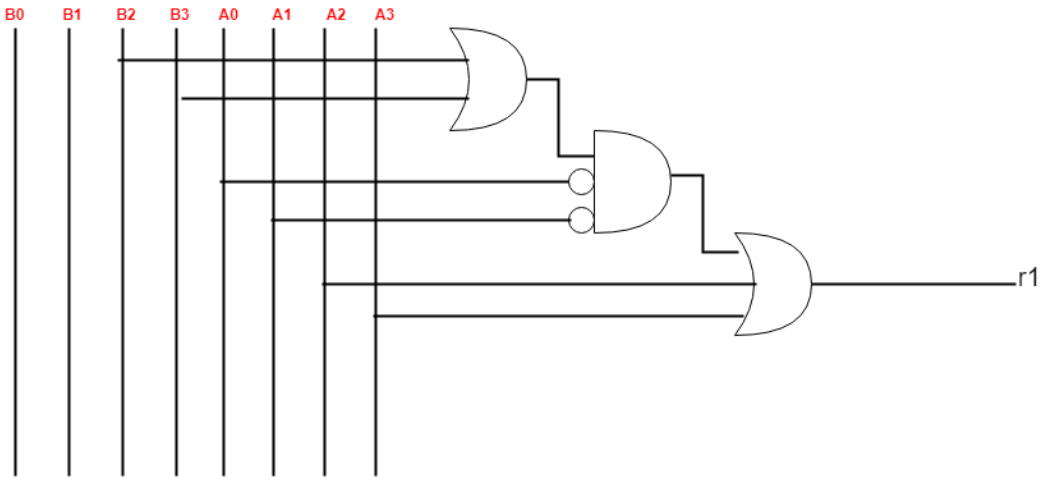
3.Priority_Encoder



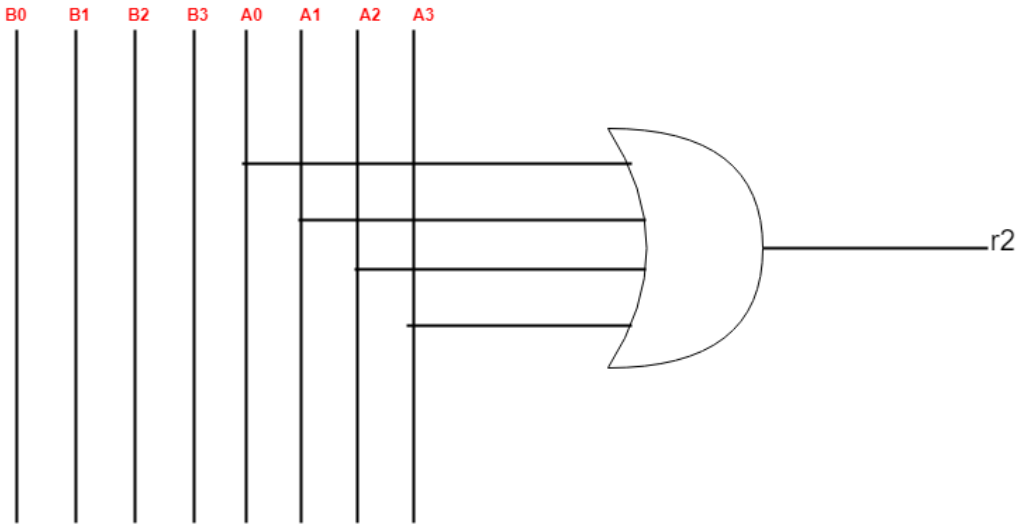
R0



R1



R2



改良之後的 **R0, R1, R2** 是根據前面的 **R0, R1, R2** 來改的，拿 **R0** 當例子，原本的 **R0** 是 **B1+B3+A1+A3**，現在把前面的 **0** 都補上去，所以就會變成 $\overline{A3}\ \overline{A2}\ \overline{A1}\ \overline{A0}\ \overline{B3}\ \overline{B2}\ B1$ + $\overline{A3}\ \overline{A2}\ \overline{A1}\ \overline{A0}$ B3 +$\overline{A3}\ \overline{A2}$ A1 + A3 接著化簡他，將$\overline{A3}\ \overline{B3}\ \overline{A1}\ \overline{B1}$消掉，剩下$\overline{A2}\ \overline{A0}\ \overline{B2}\ B1$ + $\overline{A2}\ \overline{A0}$ B3 + $\overline{A2}$ A1 + A3 最後把$\overline{A2}$提出來$\overline{A2}(\overline{A0}\ \overline{B2}\ B1 + \overline{A0}$ B3 + A1) + A3 這樣就可以畫出上面改良版的 **R0** 了。

經過改良後就可以克服掉同時有很多個 **1** 輸入進來的狀況，比如說 **00010110** 輸入進來就可以顯示出 **100** 了。下面兩張圖比較改良前和改良後的差異。

改良前

```
test_case No.    1: A = 0000, B = 0001, Sel = 0000, Out = 0000000000000000, Ovf = 0
test_case No.    2: A = 0001, B = 0110, Sel = 0000, Out = 0000000000000111, Ovf = 0

------------------<Wrong answer>------------------

Correct answear: Out = 0000000000000100, Ovf = 0.
```

輸入 **00010110** 結果跳出 **111**

改良後

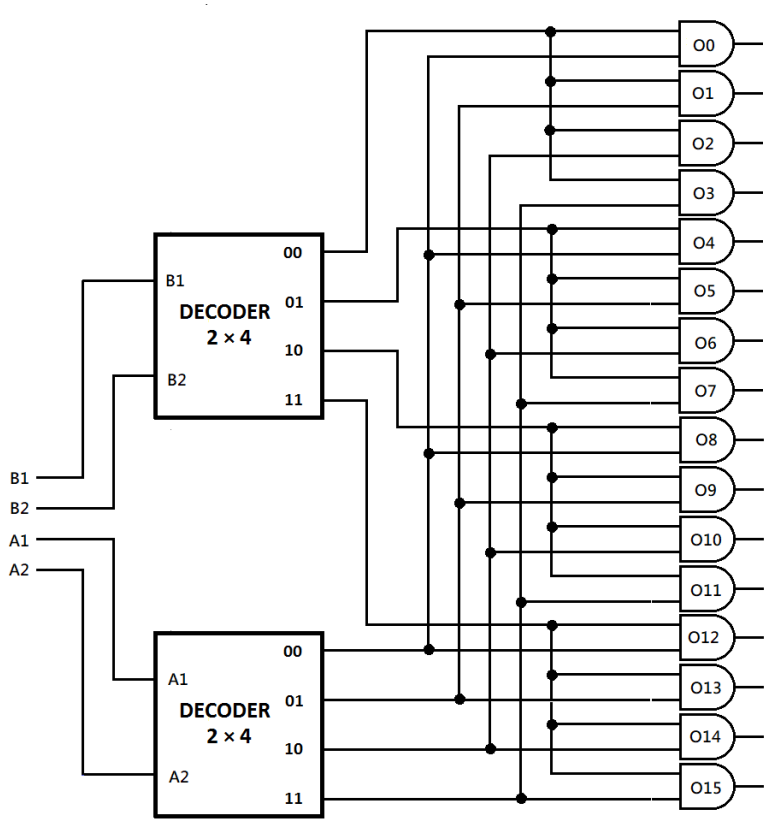
```
test_case No.    1: A = 0000, B = 0001, Sel = 0000, Out = 0000000000000000, Ovf = 0
test_case No.    2: A = 0001, B = 0110, Sel = 0000, Out = 0000000000000100, Ovf = 0
test_case No.    3: A = 0111, B = 0001, Sel = 0001, Out = 0000000000000000, Ovf = 0

------------------<Wrong answer>------------------

Correct answear: Out = 0000000000000001, Ovf = 0.
```

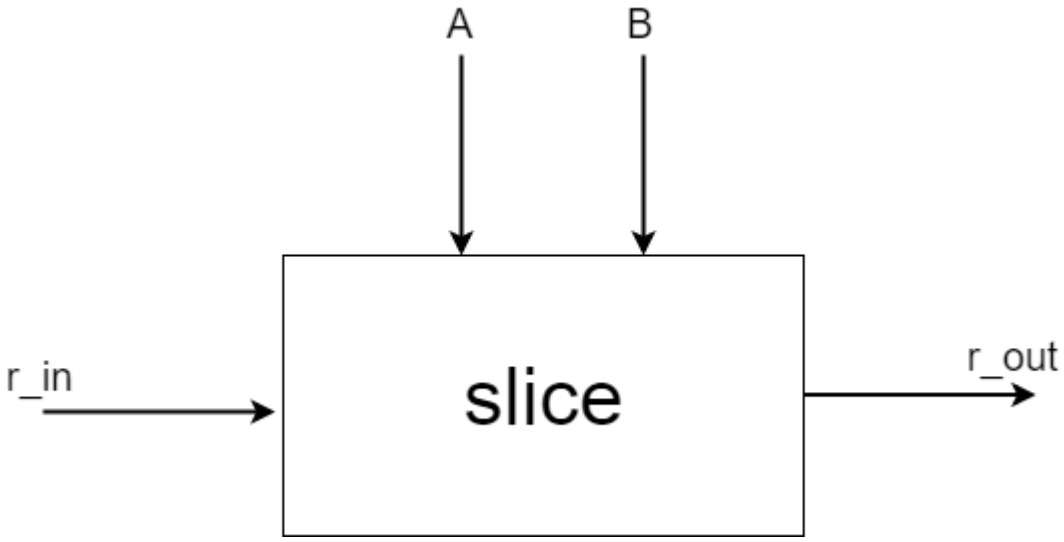
成功解決這個問題

4.Decoder_4_16



Reuse 兩個 **Decoder_2_4** 在輸入的地方，我嘗試用上次助教說的格式，可是還是不太對，所以這次我還是一樣開 **4** 個 **input** 讓他輸入，可能是我忘記上次正確的格式，所以打錯了。

5.Bit_slice_larger&smaller



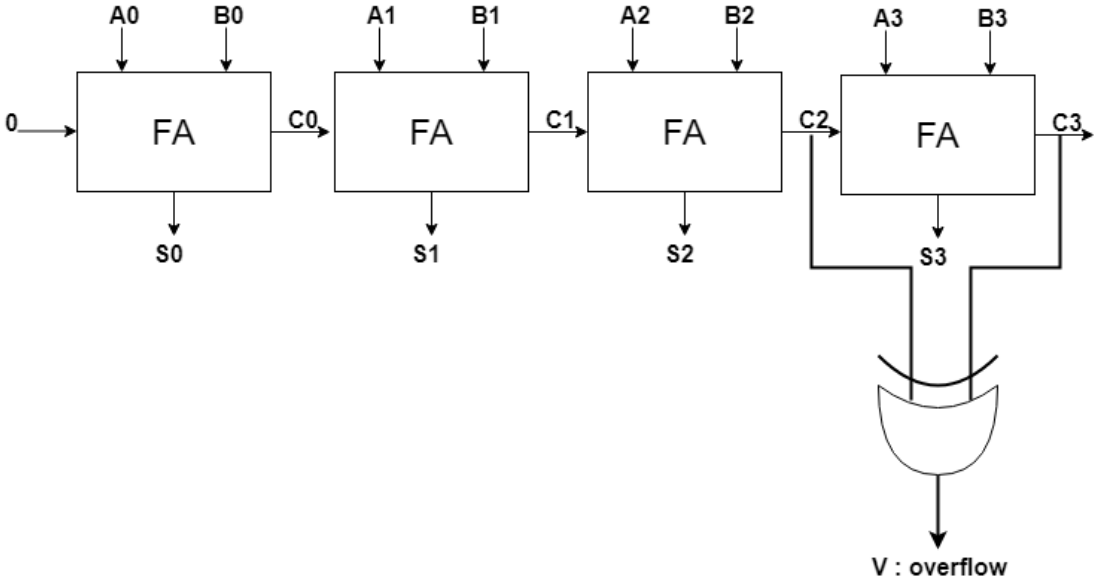
找大找小，權重都是從小到大，剛開始我以為一個是從小到大，另一個就要從大到小，可是後來想過才發現，如果不是將大於、等於、小於，分開判斷的話，都要從小比到大，這樣才不會有他無法判斷的情況。

Case_larger 如果 **A>=B r_out=1, r0** 要傳 **1** 進去因為一開始的 **A** 跟 **B** 都是 **0**。輸出的部分，如果最後的 **r**

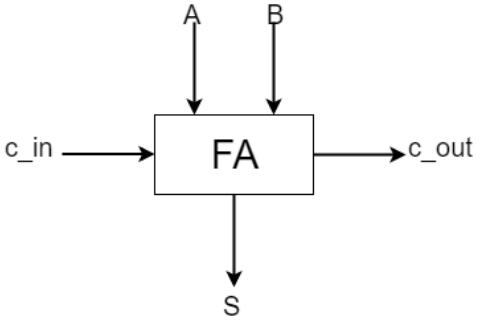
是 **1** 就輸出 **A** 反之就輸出 **B**，原因是 **r=1** 代表 **A>=B** 那不管是大於還是等於，輸出 **A** 都沒問題。

Case_smaller 如果 **A<=B r_out=1, r0** 要傳 **1** 進去因為一開始的 **A** 跟 **B** 都是 **0**。輸出的部分，如果最後的 **r** 是 **1** 就輸出 **A** 反之就輸出 **B**，原因是 **r=1** 代表 **A<=B** 那不管是小於還是等於，輸出 **A** 都沒問題。

6.Myadder&Mysubtractor



Myadder&Mysubtractor_Bit Slice



Myadder&Mysubtractor _truth table

A	B	C_in	C_out	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

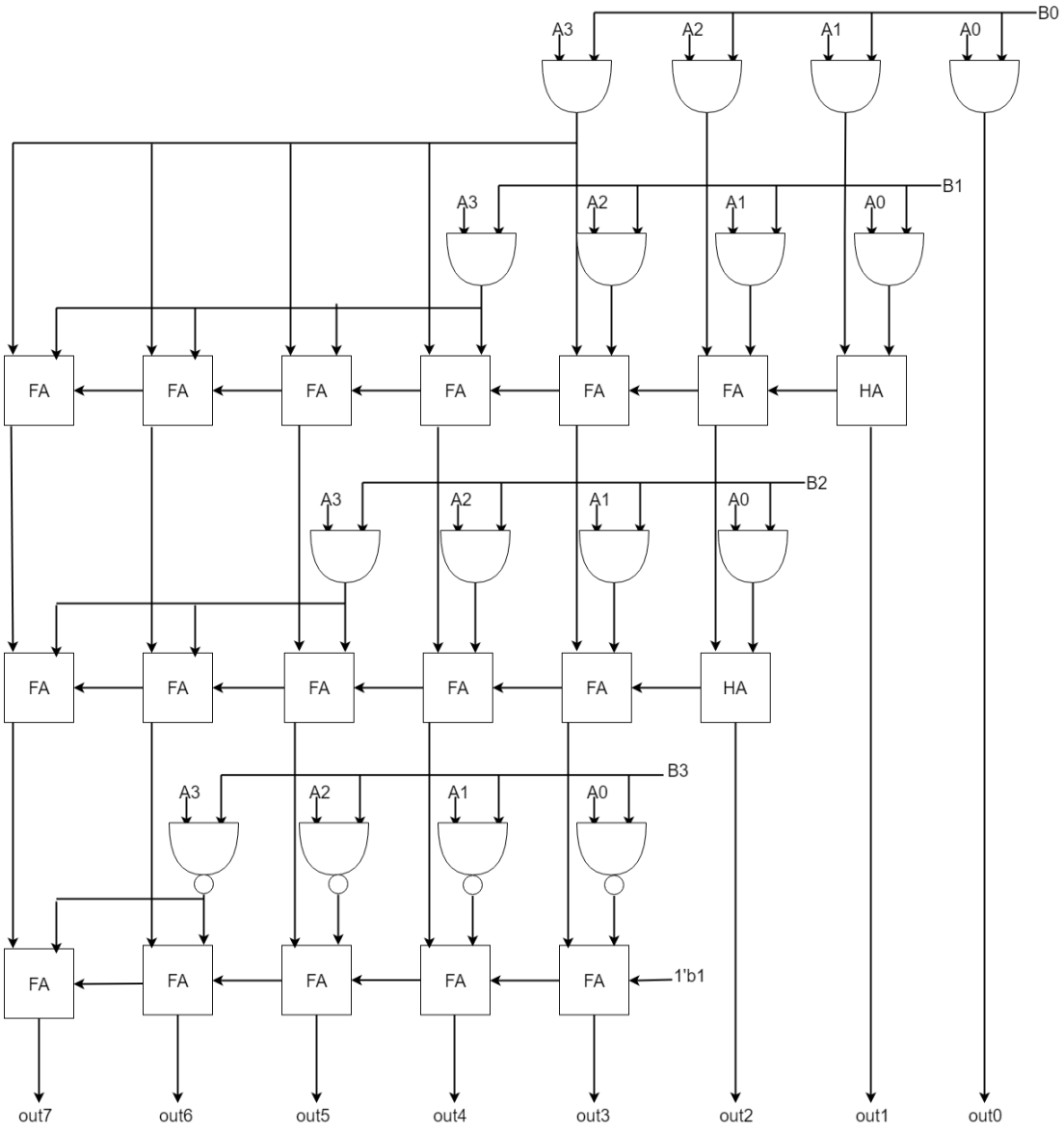
Add&Sub 一開始要傳 **0** 進去，因為前面沒有進位。我判斷 **overflow** 的地方是在最後一個 **bit slice** 的 **c_in** 跟 **c_out**，兩個不相同，就是 **overflow**。

Sub 的做法是先將 **B** 變成**(-B)**，**reuse myadder** 將 **A** 跟**(-B)**加起來。

7.Absolute

我先 **reuse** 減法，算出來的結果存到一個 **tmp[3:0]** 裡面，再用一個 **wire S2[3:0]**存負的 **tmp**，如果 **tmp[3]**是 **1** 說明 **tmp** 是負的，這時候就把 **output** 設成 **S2**，反之就直接把 **tmp** 當成 **output**。

8.Bonus



用 **and gate** 可以達到相乘的作用之後再用 **FA** 和 **HA** 把每一項相乘的結果按照權重加起來。最下面有 **4** 個 **nand gate**，因為 **2's complement** 的第一位是負的，所以經過 **nand gate** 在加 **1** 的轉換後，就等價於將 **A** 轉成負的之後再跟 **B** 相乘，在這裡，如果是兩個正數相乘，那他經過 **nand gate** 的轉換，因為 **B3=0**

所以結果一定是都是 **1**，在加 **1** 就會全部變成 **0** 了，所以也不會有錯。

9.Ncverilog Simulation Result

```
test_case No.    1: A = 0000, B = 0001, Sel = 0000, Out = 0000000000000000, Ovf = 0
test_case No.    2: A = 0001, B = 0110, Sel = 0000, Out = 0000000000000100, Ovf = 0
test_case No.    3: A = 0111, B = 0001, Sel = 0001, Out = 0000000000000001, Ovf = 0
test_case No.    4: A = 0001, B = 0110, Sel = 0001, Out = 0000000000000000, Ovf = 0
test_case No.    5: A = 0111, B = 0001, Sel = 0010, Out = 0000000000000110, Ovf = 0
test_case No.    6: A = 0001, B = 0110, Sel = 0010, Out = 0000000000000111, Ovf = 0
test_case No.    7: A = 0111, B = 0011, Sel = 0011, Out = 0000000000010101, Ovf = 0
test_case No.    8: A = 0001, B = 0110, Sel = 0011, Out = 0000000000000110, Ovf = 0
test_case No.    9: A = 0111, B = 0000, Sel = 0100, Out = 0000000000000011, Ovf = 0
test_case No.   10: A = 0001, B = 0000, Sel = 0100, Out = 0000000000000000, Ovf = 0
test_case No.   11: A = 0111, B = 0000, Sel = 0101, Out = 0000000000001110, Ovf = 0
test_case No.   12: A = 0001, B = 0000, Sel = 0101, Out = 0000000000000010, Ovf = 0
test_case No.   13: A = 0111, B = 0000, Sel = 0110, Out = 0000000000000011, Ovf = 0
test_case No.   14: A = 0001, B = 0000, Sel = 0110, Out = 0000000000000000, Ovf = 0
test_case No.   15: A = 0111, B = 0000, Sel = 0111, Out = 0000000000001110, Ovf = 0
test_case No.   16: A = 0101, B = 0000, Sel = 0111, Out = 0000000000001010, Ovf = 0
test_case No.   17: A = 0000, B = 0000, Sel = 1000, Out = 0000000000000001, Ovf = 0
test_case No.   18: A = 0000, B = 1000, Sel = 1000, Out = 0000000000000010, Ovf = 0
test_case No.   19: A = 0000, B = 0000, Sel = 1001, Out = 0000000000000001, Ovf = 0
test_case No.   20: A = 0000, B = 1000, Sel = 1001, Out = 0000000000000100, Ovf = 0
test_case No.   21: A = 0100, B = 0011, Sel = 1010, Out = 0000000000000100, Ovf = 0
test_case No.   22: A = 0000, B = 1000, Sel = 1010, Out = 0000000000001000, Ovf = 0
test_case No.   23: A = 0100, B = 0011, Sel = 1011, Out = 0000000000000011, Ovf = 0
test_case No.   24: A = 0000, B = 1000, Sel = 1011, Out = 0000000000000000, Ovf = 0
test_case No.   25: A = 0100, B = 0011, Sel = 1100, Out = 0000000000000111, Ovf = 0
test_case No.   26: A = 0000, B = 1000, Sel = 1100, Out = 1111111111111000, Ovf = 0
test_case No.   27: A = 0100, B = 0011, Sel = 1101, Out = 0000000000000001, Ovf = 0
test_case No.   28: A = 0000, B = 1100, Sel = 1101, Out = 0000000000000100, Ovf = 0
test_case No.   29: A = 0100, B = 0011, Sel = 1110, Out = 0000000000000001, Ovf = 0
test_case No.   30: A = 0000, B = 0100, Sel = 1110, Out = 0000000000000100, Ovf = 0
test_case No.   31: A = 1111, B = 0100, Sel = 1110, Out = 0000000000000101, Ovf = 0
test_case No.   32: A = 1100, B = 0101, Sel = 1110, Out = 0000000000000111, Ovf = 1
All Correct.

Test bonus...

Bonus No.   33: A = 0111, B = 0011, Sel = 1111, Out = 0000000000010101, Ovf = 0
Bonus No.   34: A = 0001, B = 0110, Sel = 1111, Out = 0000000000000110, Ovf = 0

Get bonus.
```

我這次也是寫好一個 **module** 後就傳一次，慢慢把他們全部寫完，在做的過程中，**mymultiplication** 讓我想很久，想說要怎麼不用 **gatelevel** 來做，因為 **bonus** 才會用到，大概花了 **1** 個小時，後來在滑助教

給的講義”Verilog HDL 教學講義”的資料流層次時，看到乘法原來可以直接用，瞬間豁然開朗，下面的 **shift** 也順勢寫出來，謝謝助教。

10.碰到的問題與解決的方法

問題 **1**：在做 **sign extension** 的時候，我不想要用 **if else** 來判斷他前面的時 **1** 還是 **0**。

解決方法：後來發現可以強制轉換，下圖程式碼裡面的**$signed()**就是將括號裡面的 **wire** 變成可以自動 **sign extension** 的型態，如果等號左邊的矩陣比較長，他就會自動 **sign extension**。

```
4'b1100: begin
  //Out[3:0] = out1100[3:0];
  Out[15:0] = $signed(out1100);
  Ovf = Ovf1100;
  end
4'b1101: begin
  Out[15:0] = $signed(out1101);
  Ovf = Ovf1101;
  end
4'b1110: begin
  Out[15:0] = $signed(out1110);
  Ovf = Ovf1110;
  end
```

問題２：在做絕對值的 **module** 時，我原本是直接傳 **S** 進去 **mysubtraction**，可是這樣會有形態問題，

因為我下面有用 **always** 所以 **S** 得變成 **reg** 可是這裡又要傳 **wire** 進去。

解決方法：在 **473** 行我設了一個 **wire tmp** 他先記住 **mysubtraction** 的結果，下面是用 **S2** 記住負的 **tmp** 如此一來，便可以解決這個問題。

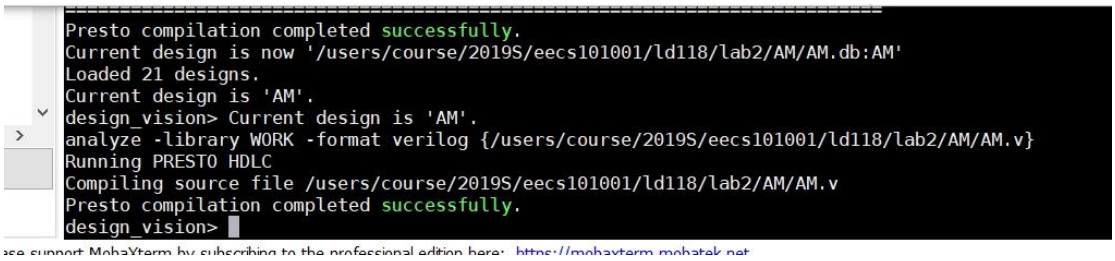
```
473   wire [3:0] tmp;
474
475   //assign tmp[3:0] = S[3:0];
476   mysubtraction mysubtraction_1(.A(A), .B(B), .S(tmp), .Ovf(Ovf));
477
478   //because 's' is negative so I take -s
479   wire [3:0] S1;
480   wire [3:0] S2;
481
482   not not_0(S1[0], tmp[0]);
483   not not_1(S1[1], tmp[1]);
484   not not_2(S1[2], tmp[2]);
485   not not_3(S1[3], tmp[3]);
486
487   assign S2[3:0] = S1[3:0] + 4'b0001;
488
```

問題 **3**：在合成電路圖的時候遇到的一系列問題

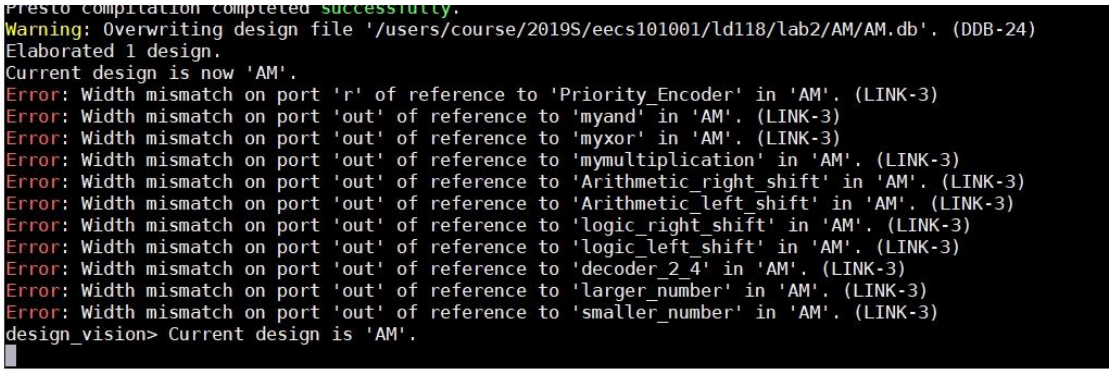
```
Running PRESTO HDLC
Compiling source file /users/course/2019S/eecs101001/ld118/lab2/AM/AM.v
Warning:  /users/course/2019S/eecs101001/ld118/lab2/AM/AM.v:540: the undeclared symbol 's6' assumed to have the default net type, which is 'wire'. (VER-936)
Warning:  /users/course/2019S/eecs101001/ld118/lab2/AM/AM.v:550: the undeclared symbol 's9' assumed to have the default net type, which is 'wire'. (VER-936)
Warning:  /users/course/2019S/eecs101001/ld118/lab2/AM/AM.v:551: the undeclared symbol 's10' assumed to have the default net type, which is 'wire'. (VER-936)
Warning:  /users/course/2019S/eecs101001/ld118/lab2/AM/AM.v:554: the undeclared symbol 's7' assumed to have the default net type, which is 'wire'. (VER-936)
Warning:  /users/course/2019S/eecs101001/ld118/lab2/AM/AM.v:555: the undeclared symbol 's8' assumed to have the default net type, which is 'wire'. (VER-936)
Presto compilation completed successfully.
design_vision>
```

a.警告我的一些 **wire** 沒有宣告就使用

解決方法：一個對一個的宣告完成。



分析成功，本以為結束了。



b.他又跳出這個警告，後來我上網查了發現是因為我的 **wire** 長的接到短的去。

解決方法：在主函數額外設 **out** 讓他可以接好，如下圖，之後再手動把前面補成 **0**。

```
//absolute
wire[3:0] out1110;
wire Ovf1110;
absolute absolute_0(.A(A), .B(B), .S(out1110), .Ovf(Ovf1110));

//bonus
wire[7:0] out1111;
bonus bonus_0(.A(A), .B(B), .out(out1111));

always @ ( * ) begin
  case (Sel)
    4'b0000: begin
      Out[2:0] = out0000[2:0];
      Out[15:3] = 13'b0_0000_0000_0000;
      Ovf = 1'b0;
      end
    4'b0001: begin
      Out[3:0] = out0001[3:0];
      Out[15:4] = 12'b0000_0000_0000;
      Ovf = 1'b0;
      end
```

因為實在太長了，所以我就取，宣告的後面一點點，和 **mux** 的前面一點點來截圖，但中間都是一樣的，上面依據輸出的 **out** 來宣告對應的 **out(編號)**，成功解決**(b)**的問題。

11.結論

這次的 **lab2** 剛開始看到的時候我完全沒頭緒，也不知道從何下手，因為 **lab1** 只有做一個函數而已，**lab2** 突然要做 **16** 個，又要把他們融合在一起，剛開始真的很頭大，可是助教很有耐心，在我完成這份報告前總共問了助教 **3** 次，第一次是在下課後有先問過助教這次是要做出全部的就好，還是要合在一起。第二次在 **demo** 的時候我是先跟助教說我的合成想法，再問助教可不可行。問完後那個假日我就把程式碼打完了，花了整整兩天都在打。第三次就是問一些報告要交的東西，跑出來的結果這樣是不是對的，這些問題。

這次 **lab2** 讓我學到很多，至少在實作過程中，我碰上的問題比上面列的還要多，很多都很零碎就忘記

了，但解決他們都會讓我對 **verilog** 更加熟悉。雖然期中考還是考不好，感覺突然要兜一些電路會兜不出來，可是我還是希望能在 **lab** 上面努力一點，把該學的學會。