LAB1

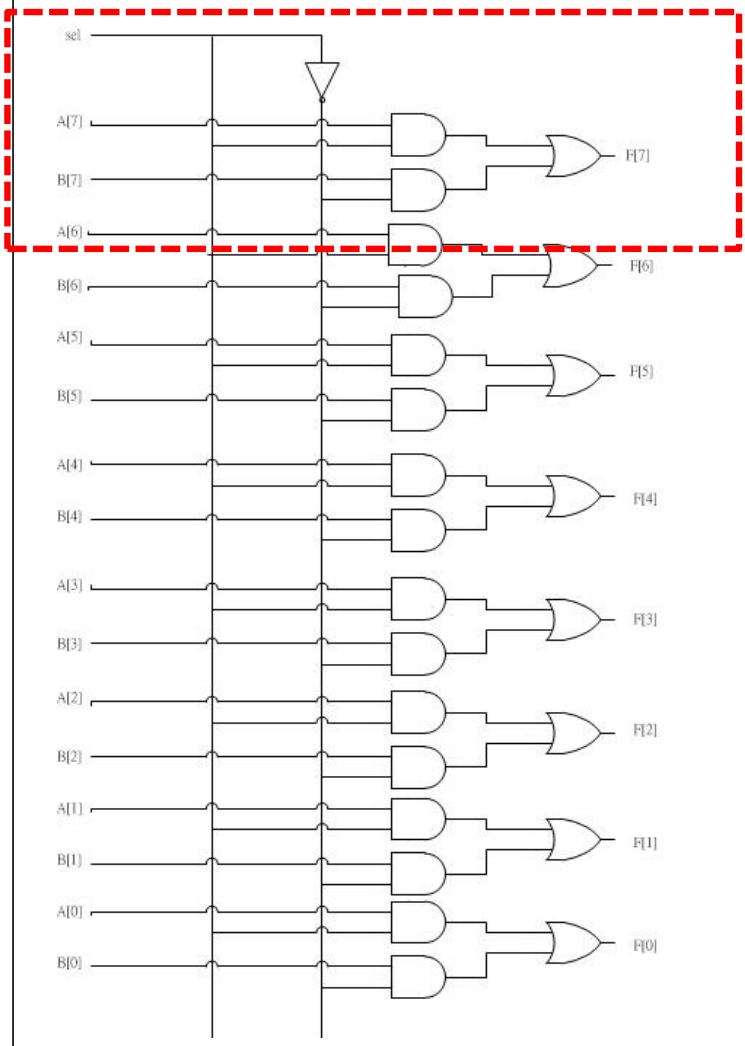
第七組

黎佑廷 105030009

郭家偉 105030015

VERILOG QUESTION 1:

8-BIT 2-TO-1 MUX



這跟 basic question 是相同的作法，不過變成 8 組。由圖，會有 NOT gate 是可以想像的。因為 sel 輸入一個訊號後，勢必只能走 A 或 B，所以要屏蔽掉一個。

假設 sel=1，那麼 A[7]會和 1 做交集，恆等於 A[7]本身，此時 B[7]會和 0 做交集，恆等於 0。最後將兩個 AND gate 做 OR，因為信號有可能從 A 或 B 出來，並且不會同時。同樣的作法，做 8 次，就是 8bit 的 mux。

驗證的方法是讓 A、B、sel 跑完所有的可能，但因為所有的可能有 2 的 17 次方這麼多。故我一次加了比較大的數，{sel, A, B}={sel, A, B}=7'b1111111。

心得：mux 這題較為容易，只是做 8 次 1bit 的 mux 而已。不過用 gate level 寫 comparator 卻是比想像中難，可能因為它的 behavior level 很簡單，我們就忽略了它的基本邏輯。於是，這題寫起來，是還蠻訓練邏輯與思考的，相當有趣。

另外，第一次使用 fpga 板，遇到了一些問題。像是[get_ports {fanout_eq[6]}]中，get_ports 與〝{〞之間的空格是必要的、若不是 module 的 output，比如說，某個 wire 信號，是不能當作控制 LED 的 port、並且只有 1 個 bit 的 output 信號不能同時給很多個 LED 當 port，就是不能一對多，只能一對一，因此要在原 module 上方加上一個 fanout module，用反向再反向的方式，達到由一個信號轉為多個信號的目的。

VERILOG QUESTION 2

(GATE-LEVEL) 4X16 DECODER

1. din = 4'b1111 時，dout[0] 的 K-map

din[3], din[2] \ din[1], din[0]	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	1	0
10	0	0	0	0

由 K-map 可以發現 dout[0] = din[3] & din[2] & din[1] & din[0]。

以此類推:

dout[1] = din[3] & din[2] & din[1] & not(din[0])

dout[2] = din[3] & din[2] & not(din[1]) & din[0]

dout[3] = din[3] & din[2] & not(din[1]) & not(din[0])

dout[4] = din[3] & not(din[2]) & din[1] & din[0]

dout[5] = din[3] & not(din[2]) & din[1] & not(din[0])

dout[6] = din[3] & not(din[2]) & not(din[1]) & din[0]

dout[7] = din[3] & not(din[2]) & not(din[1]) & not(din[0])

dout[8] = not(din[3]) & not(din[2]) & not(din[1]) & not(din[0])

dout[9] = not(din[3]) & not(din[2]) & not(din[1]) & din[0]

dout[10] = not(din[3]) & not(din[2]) & din[1] & not(din[0])

dout[11] = not(din[3]) & not(din[2]) & din[1] & din[0]

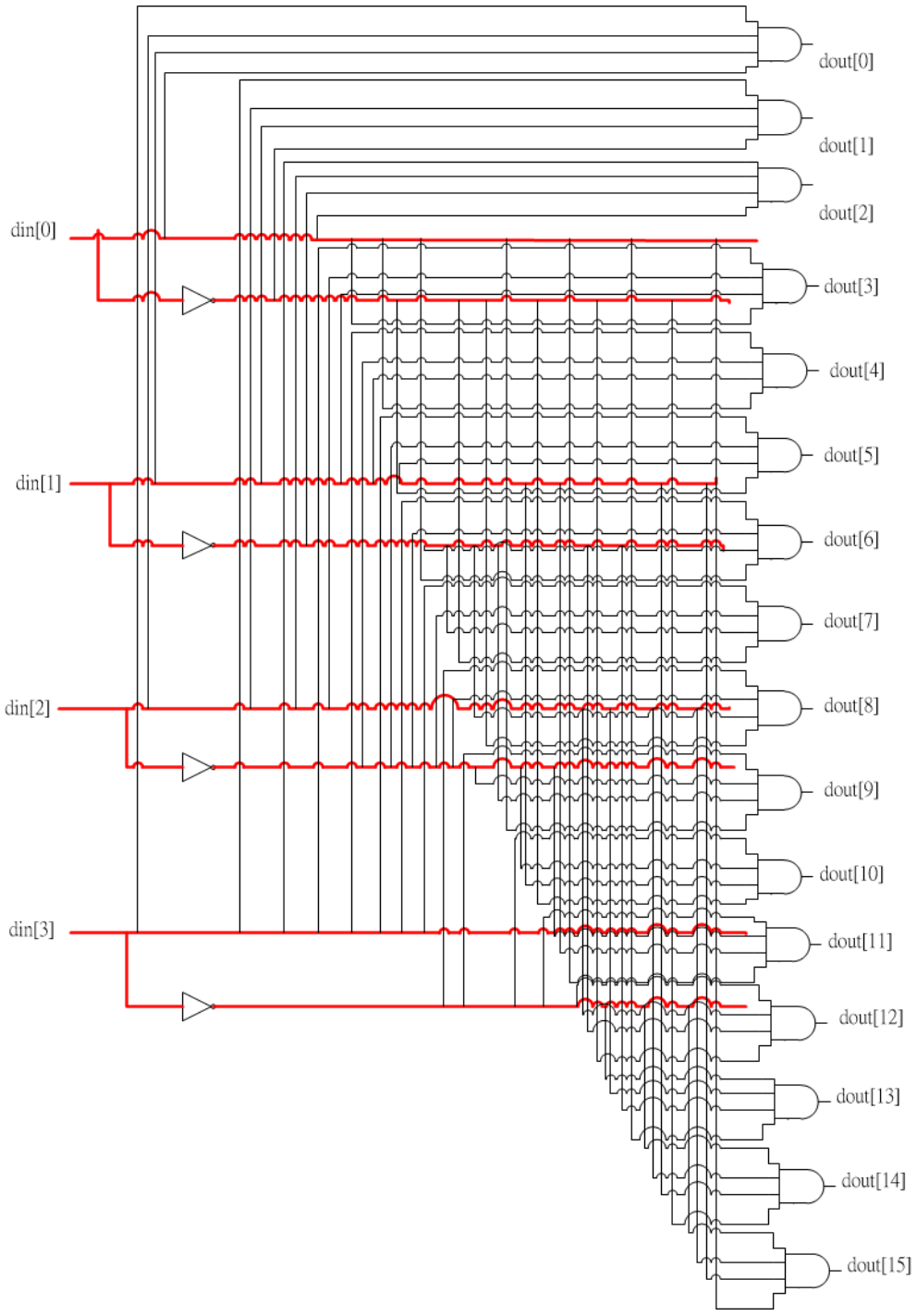
dout[12] = not(din[3]) & din[2] & not(din[1]) & not(din[0])

dout[13] = not(din[3]) & din[2] & not(din[1]) & din[0]

dout[14] = not(din[3]) & din[2] & din[1] & not(din[0])

dout[15] = not(din[3]) & din[2] & din[1] & din[0]

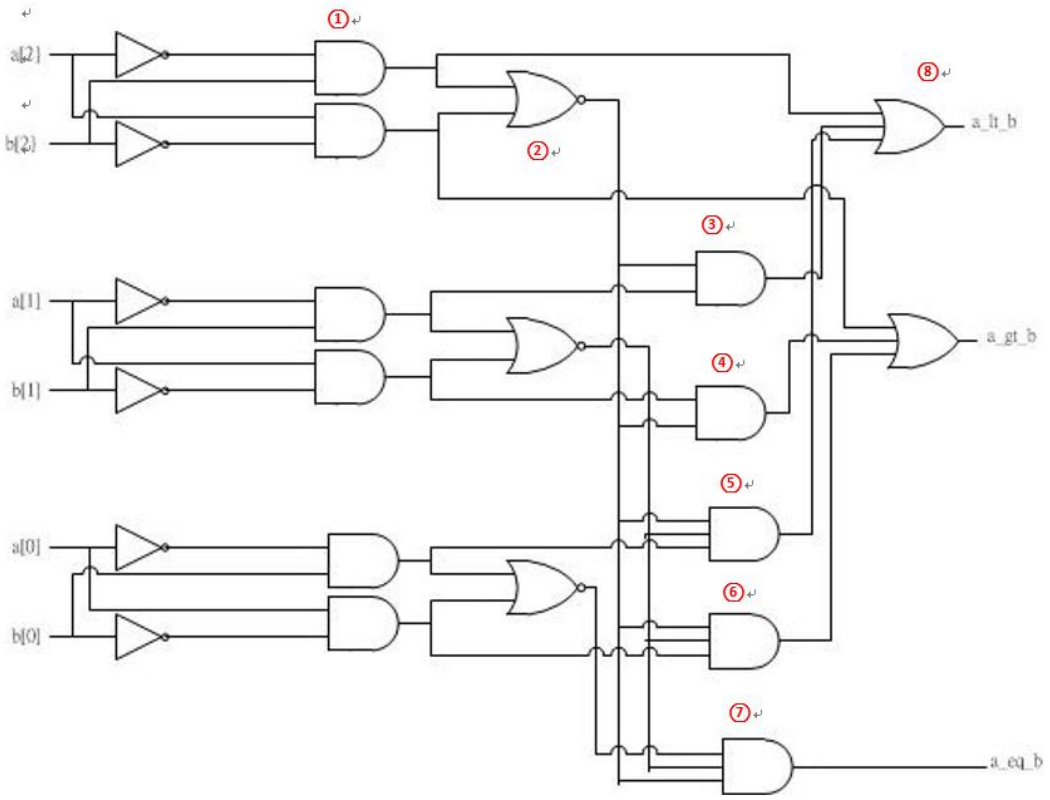
Gate Level Circuit:



這題我所使用的驗證方法是將 input din 從 4'b0000 加到 4'b1111 以驗證所有的可能性。

VERILOG QUESTION 3

3-BIT COMPARATOR



說明：

①在這個 AND gate，只有在 b[2]=1，a[2]=0，也就是 b[2]>a[2]時，才會 output=1，其餘情況都是 output=0。在①下方的 AND gate 則是在 a[2]>b[2]時才會 output=1。下方的四個 AND gate 也是同義，不過分別是 a[1]與 b[1]、a[0]與 b[0]。

②在這個 NOR gate，只有在兩個 input 都是 0 時，才會 output=1。回到①的 AND gate 看，要兩個 AND gate 都 output=1 代表 b[2]不大於 a[2]，且 a[2]也不大於 b[2]，這代表 a[2]==b[2]。所以這個 NOR gate 輸出 1 時，代表在這個 bit 還不分勝負，必須往下一個 lsb 檢查，故連結到③~⑦的 AND gate。必須要是 1 才能啟動 lsb 的檢查，反之，如果是 0，代表在目前比較的 bit 已經能分出勝負，故會以連接的 AND gate 關閉後面的 lsb 檢查。

③在這個 AND gate，如果在 a[2]與 b[2]分不出勝負，且在目前的 bit，也就是 b[1]>a[1]時，會 output=1，並且連結到 a_lt_b 的 OR gate。④也是相同道理，不過是 a[1]>b[1]，並且連結到 a_gt_b 的 OR gate。⑤與⑥亦同，不過這次 AND gate 有三個 input，要當 a[2]與 b[2]、a[1]與 b[1]都分不出勝負時，且在 b[0]>a[0]，或 a[0]>b[0]時分別連結到 a_lt_b、a_gt_b。

⑦若是在 a[2]與 b[2]、a[1]與 b[1]、a[0]與 b[0]都分不出勝負時，會使這個 AND gate output=1，也就是 a_eq_b=1。

⑧此 OR gate 有三個 input，分別代表在三個 bit 中哪一個 bit 發現 a<b，導致 a_lt_b=1。這三個 input 只會有一個是 1，其餘為 0。這是因為在稍微前端的三個 NOR gate，如果在目前的 bit 發現 a<b 就會關閉後面的檢查。a_gt_b 的 OR gate 也是相同道理。

Testbench 的寫法主要是讓 a、b 所有的可能性都跑過一次，也就是{a, b}={a, b}+1'b1，藉以檢查 code。

VERILOG QUESTION 4

(GATE-LEVEL) 4-BIT RIPPLE-CARRY ADDER (RCA)

4-BIT RIPPLE-CARRY ADDER 可以由四個 1-BIT 的 FullAdder 組成，因此我翻了翻以前的筆記，複習一下。

1-BIT FullAdder 的運作原理如下:

a	b	Cin	Cout	sum
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

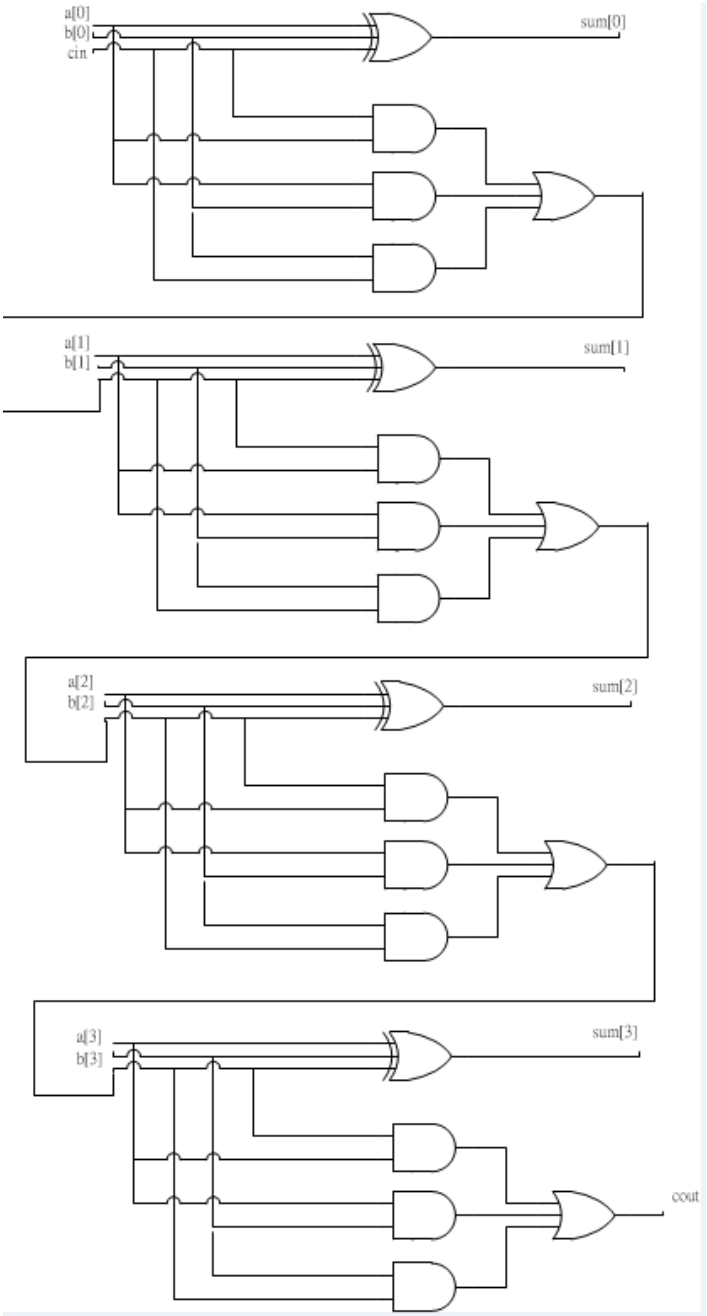
由表格可以發現 Cout 就是 a, b, Cin 的 majority，

即 Cout = (a&b) | (a&cin) | (b&cin)

而 sum 就是如果 a, b, cin 當中有奇數個 1 那 sum 就等於 1，也就是說 sum = a, b, cin 三個 input 做 XOR。

有了這樣的基礎知識，接下來我們只需要將四個 1-BIT FullAdder 的 Cout 以及 Cin 做前後連接，4-BIT RIPPLE-CARRY ADDER 就完成啦。

Gate Level Circuit:



這題我所使用的驗證方法是賦予 input a, b 不同的值來進行確認，但因為 a 跟 b 都有 16 種可性，所以有 256 種組合，要一一驗證會耗費大量時間與精力，因此我採用人工賦予 a, b 數值的方式，進行抽驗。

成員:

黎佑廷: 負責第 2、4 題以及整理 report

郭家偉: 負責第 1、3 題以及劃出 gate level circuit