

第七組

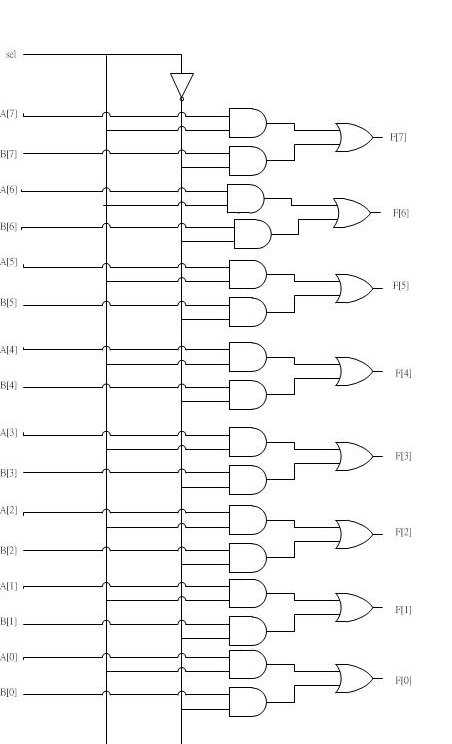
黎佑廷 105030009

郭家偉 105030015

lab1

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[1], din[0]

din[3], din[2]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 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]

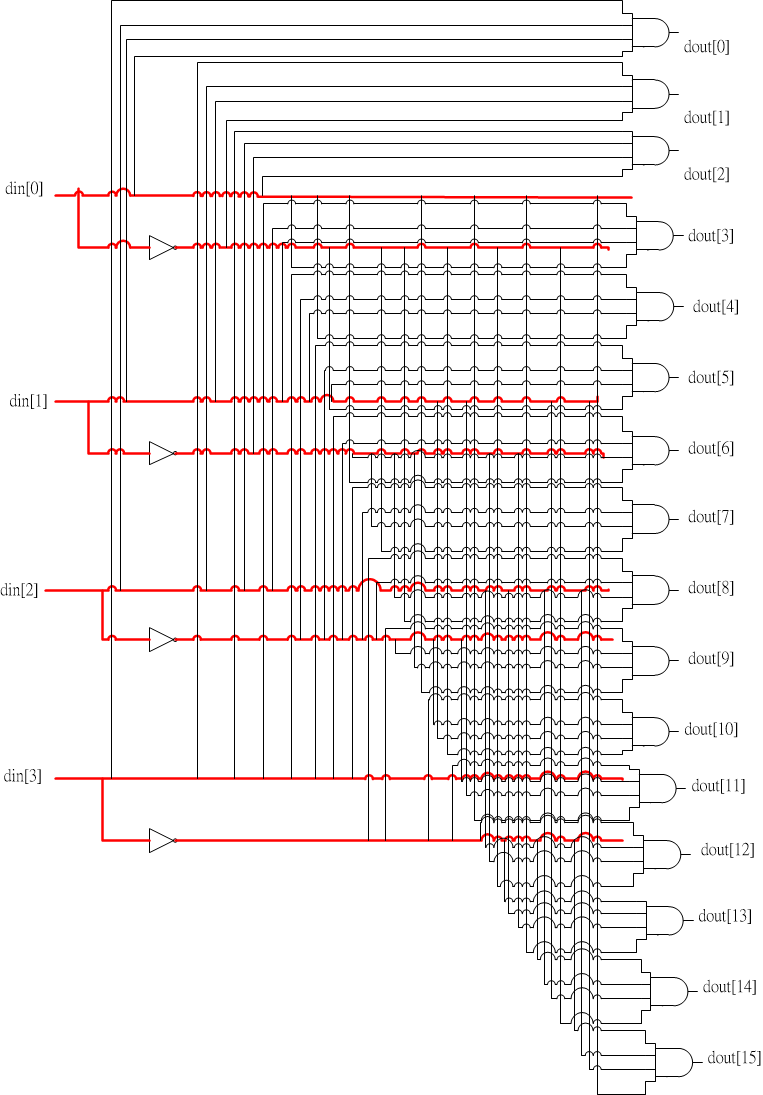
dout[12] = not(din[3]) & din[2] & not(din[1]) & not(din[0])

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]

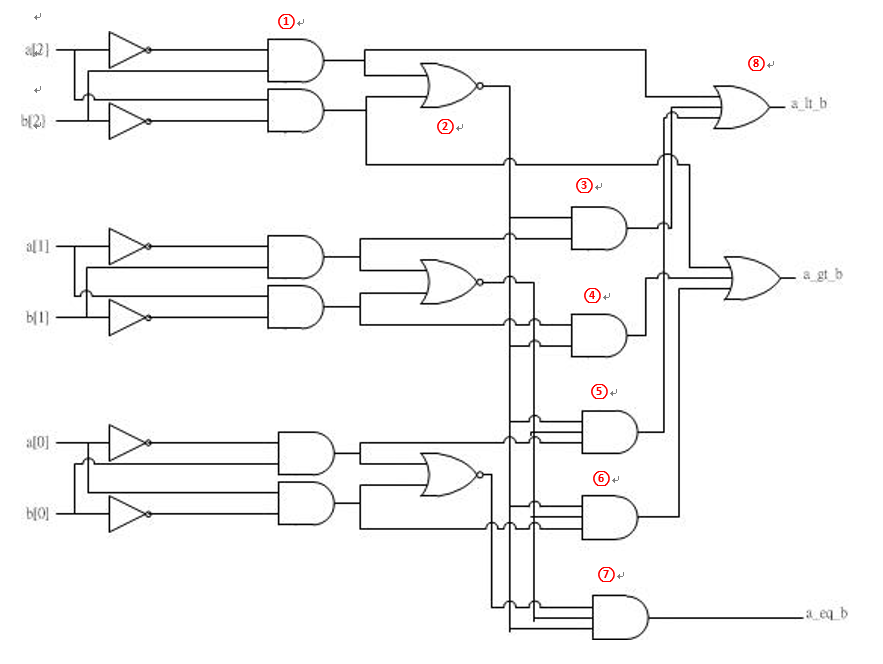
Gate Level Circuit:



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：

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]。

在這個NOR gate，只有在兩個input都是0時，才會output=1。回到的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~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。

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。也是相同道理，不過是a[1]>b[1]，並且連結到a\_gt\_b的OR gate。與亦同，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。

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。

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 |

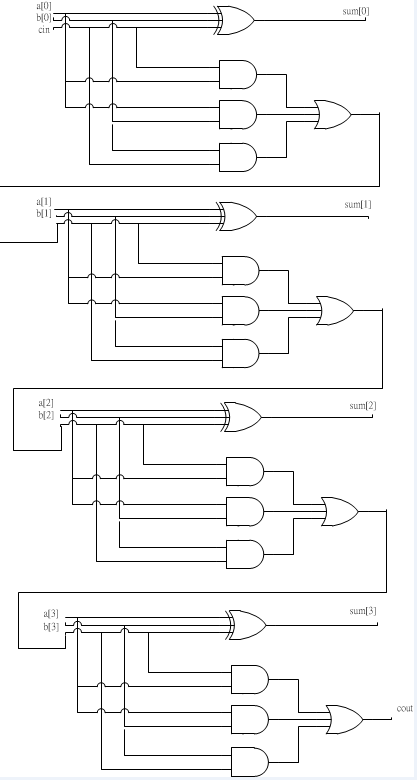
由表格可以發現 Cout 就是a, b, Cin 的majority，

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