學號:105062227 姓名:黃敬堯 系級:資工 20

1. K-map 與架構圖

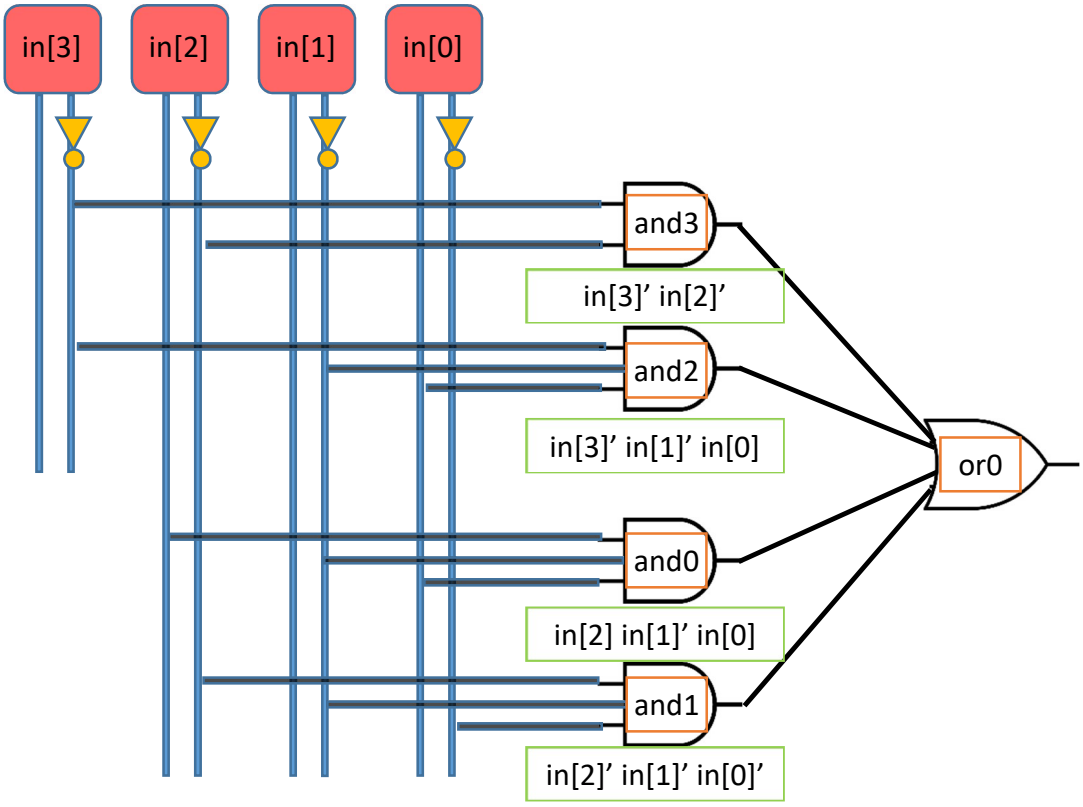
在畫電路的架構圖之前，先利用 K-map 確定電路連接的正確方式和利用的 gate 數，並確定避免 hazard 的接法；在此下面的 map 中，利用英文字母 a,b,c,d 分別代二進位數字中的十進位數字 0 至 15 的四個位元(abcd)。

cd / ab	00	01	11	10
00	1	1	1	1
01	0	1	0	0
11	0	1	0	0
10	1	0	0	0

1. 最上面那列的 0000,0001,0011,0010 可以化簡為 a’b’。
2. 第二行中第二、三列的 0101,1101 可以化簡為 bc’d。
3. 因為要考慮 hazard 的問題，所以最左下角的 1000 不能只有自己一個 gate，需要與最左上角的 0000 一起考慮，方可避免 hazard，因此可以化簡為 b’c’d’。
4. 因為要考慮 hazard 的問題，第二行中第一、二列的 0001,0101 需要再使用一個 gate，才可以避免 hazard 的產生，因此可以化簡為 a’c’d。

接著將上述的結果畫成架構圖——

為方便表示二進位數字 abcd，我將 a,b,c,d 分別表示成 in[3]、in[2]、in[1]、in[0]，



(圖中 gate 的名稱皆對應到程式碼中 gate 的名稱，因此在此圖中不會照順序排)

2. 模擬結果

```
Synopsys licenses have set!
Cadence license have set!
[dld10076@ic21 ~]$ cd lab1
[dld10076@ic21 ~/lab1]$ ncverilog majority.v testbench.v
ncverilog: 14.10-s005: (c) Copyright 1995-2014 Cadence Design Systems, Inc.
Loading snapshot worklib.testbench:v ..................... Done
*Verdi3* Loading libsscore_ius141.so
*Verdi3* : Enable Parallel Dumping.
ncsim> source /usr/cad/cadence/INCISIV/cur/tools/inca/files/ncsimrc
ncsim> run
time =  5, in = 0000, out_G = 1, out_D = 1, out_B = 1
time = 10, in = 0001, out_G = 1, out_D = 1, out_B = 1
time = 15, in = 0010, out_G = 1, out_D = 1, out_B = 1
time = 20, in = 0011, out_G = 1, out_D = 1, out_B = 1
time = 25, in = 0100, out_G = 0, out_D = 0, out_B = 0
time = 30, in = 0101, out_G = 1, out_D = 1, out_B = 1
time = 35, in = 0110, out_G = 0, out_D = 0, out_B = 0
time = 40, in = 0111, out_G = 0, out_D = 0, out_B = 0
time = 45, in = 1000, out_G = 1, out_D = 1, out_B = 1
time = 50, in = 1001, out_G = 0, out_D = 0, out_B = 0
time = 55, in = 1010, out_G = 0, out_D = 0, out_B = 0
time = 60, in = 1011, out_G = 0, out_D = 0, out_B = 0
time = 65, in = 1100, out_G = 0, out_D = 0, out_B = 0
time = 70, in = 1101, out_G = 1, out_D = 1, out_B = 1
time = 75, in = 1110, out_G = 0, out_D = 0, out_B = 0
time = 80, in = 1111, out_G = 0, out_D = 0, out_B = 0
Simulation complete via $finish(1) at time 80 NS + 0
./testbench.v:17                $finish;
ncsim> exit
```

3. 遇到的問題與解決方法

1. Q:在剛剛寫程式之時，並不確定當需要使用到如 in[0]’ 這種東西時，是要用一個 not gate 並多使用一個 wire 型別的東西去接，再使用它，或是說我可以直接在 and gate 之中使用布林運算符號如 ~，上網查資料，網路上也多沒有提到相關問題的處理方法。

   A:結果我就自己嘗試看看，本來的方法是將[3:0]in 也就是我的 input 分別丟進四個 not gate 中，並用 wire 型別的[3:0]in_bar 分別去接，最後再放進四個 and 中進行計算；之後我試著直接在 and gate 中直接進行布林運算，譬如寫成 and and0(temp0,in[2],~in[1],in[0]);，結果答案是一樣的，因此可以使用這個方法。

2. Q:有時候工作站會當掉，畫面卡住沒辦法按:wq 做記憶儲存。

   A:剛開始都直接把整個程式關掉，前面更改過的 code 就都沒有儲存

到；但是之後有在網路上查到指令:wq!，可以強制儲存並退出，就算畫面卡住也沒關係，可以按 esc 並打入指令就可以存，問題就解決了。

3. Q:在命名 input 的時候，因為超過 1 個 bit，要利用[3:0]這種方法命名，但是在熟悉 c 語言之後卻很不習慣將最大的數字放在最左邊，因此很好奇能不能將 0 和 3 倒過來，寫成[0:3]，這樣比較習慣。

   A:經過嘗試，試著將原本宣告為 input[3:0]in 的 input 改寫為 input[0:3]in，並更改 gate 中的 in[]，在嘗試後，發現答案是一樣的，推測方法可行。但是經過考慮後，還是決定維持原來的寫法，一方面是因為助教給的 example 是這樣寫，另一方面因為二進位數字最右邊的數字是 1 也就是 2 的 0 次方，而數字往左皆對應到 2 的次方，我認為對往後建構比較大的程式，這種 coding style 應該有所助益，所以決定維持原來寫法。

4. Q:在創建 vim 的檔案時，不小心把名字打錯；而且會有太多打錯或是測試的檔案放在資料夾中，顯得雜亂無章，但卻無法看到總共有哪些檔案。

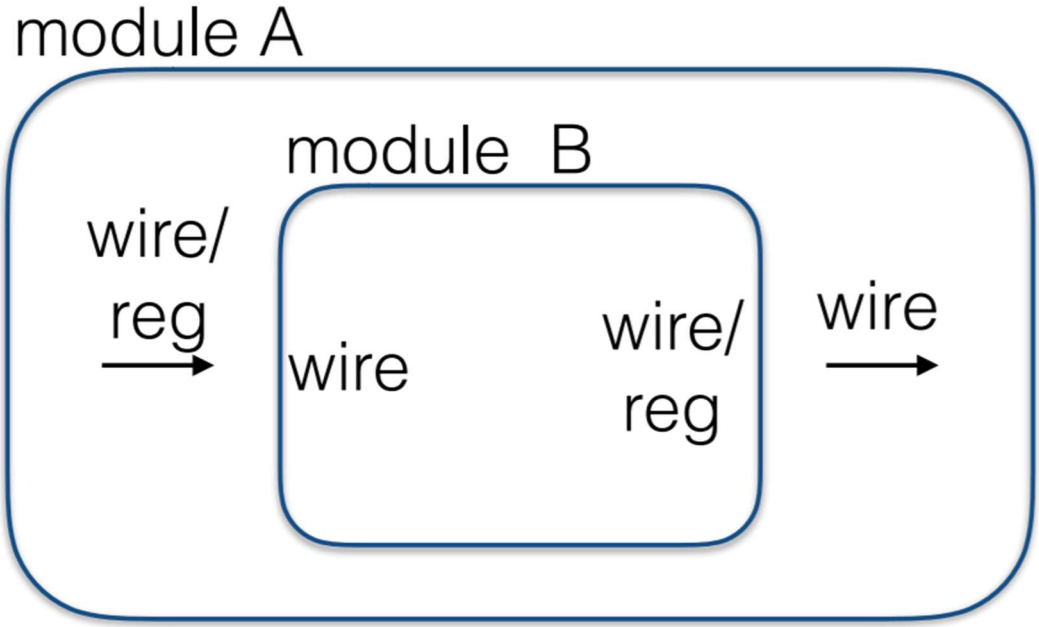
   A:上網查詢 linux 的指令表，發現可以使用 mv 加上舊檔案名稱再加上新檔案名稱，這樣就可以更改名稱。而將目錄中所有文件列出的方法，則是在目錄中，輸入 ls –a、或是 ls –all，此舉便可以顯示出目錄中所有檔案。

4. Advance

1. 在剛開始寫 verilog 時，有一個很好奇的問題就是 wire 和 reg 這兩個型別的差別到底在哪裡，它們並不如 int 或 float 等型別直觀，而且硬體語言對我來說是一種很新奇的東西，所以我認為自己需要更理解它，才能更進一步，並且寫出更正確的程式。
   但是上網查資料卻發現弔詭的地方，因為在資料中提及
   1. 「wire elements are used as inputs and outputs within an actual module declaration.」
   2. 「reg elements can be used as outputs within an actual module declaration.」
   3. 「reg elements cannot be used as inputs within an actual module declaration.」

   而在助教提供的講義中卻有說明到 module 的 input 可以是 wire 或是 reg

型別，而 output 則只能是 wire 型別，因此就產生了疑問，到底哪個說法是對的。
之後經過助教的解釋，還有這個關鍵的圖



(圖片來源:清大 ilms 討論區助教截圖)

才終於了解，文中所述的 module instantiation 是他的範例 module，對應到圖中的 module A，而助教提供的講義中所說的 module 則是在描述圖中的 module B，因此兩者的陳述皆是正確的，只是因為沒有讀懂而產生的誤會。
而 wire 和 reg 的實質差異在於前者無記憶性而後者有；wire 就像是線路，無法儲存內容也無法被指定；而 reg 則像是暫存器，可以儲存值，直到下次被指定為止；另外 wire 可以用於 combinational circuit，而 reg 可以用於 combinational 和 sequential circuit；當我們宣告 input 和 output 時，若沒有指定它們的型別，則預設為 wire。

2. 剛看到範例程式時很好奇 assign 的使用方法，在範例程式的第二種描述法中也有使用到，但是令我困惑的是，明明就用一般的布林運算給定值就好，為什麼還需要特別在前面使用加上 assign 呢。因此便上網查詢資料，就我目前的認知，assign 後面所描述的電路為永遠存在，只要輸入(等號右邊)有所變化，輸出(等號左邊)就會馬上改變，也因此在使用 assign 時，等號左邊的值只用是 wire 型別的。另外有時會同時使用很多個 assign，它們並不能使用軟體的思維思考，換句話說就是它們並沒有順序的對應關係，這是必須注意的一個點。

3. 在範例程式的第三種描述法中，output 所使用的型別為 reg，與前兩種描述法並不相同，更使用到了 always@(*)的寫法，上網查到一些資料，卻還是有點一知半解，因為好像與 blocking statement 與 nonblocking statement 有關。就目前的理解，只能知道在 always 中，在等號左邊的型別只能為 reg，而當右邊的數值有所變化，左邊的數值就會有所改變，也因此，如 if-else 或 case 等都應列入其中。另外就是(*)的使用方法，經過查詢後得知，括號中原本應該是放入右邊的變數，但是在變數過多時，這卻會造成 debug 的困難，因此可以在括號中寫上*，利用*來代表所以等式右邊的變數，因此只要變數有所改變，便會重新計算值。