**Lab 1: Verilog HDL**

Experiments

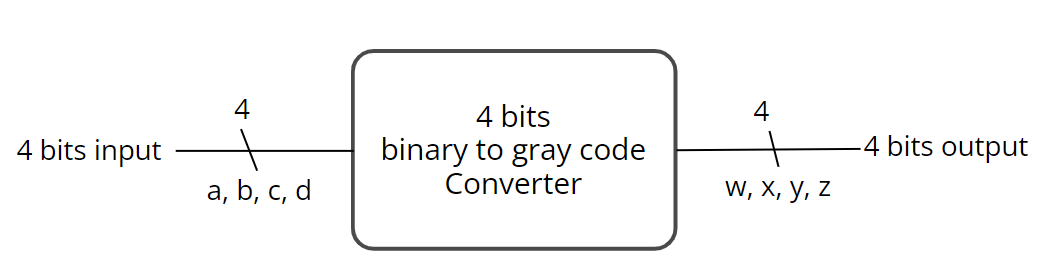
1. Design and verify a binary-to-Gray-code converter for a Gray code sequence with 10 code words (input: abcd, output: wxyz, a and w are the MSB).

Design Specification

IO 輸出入設定:

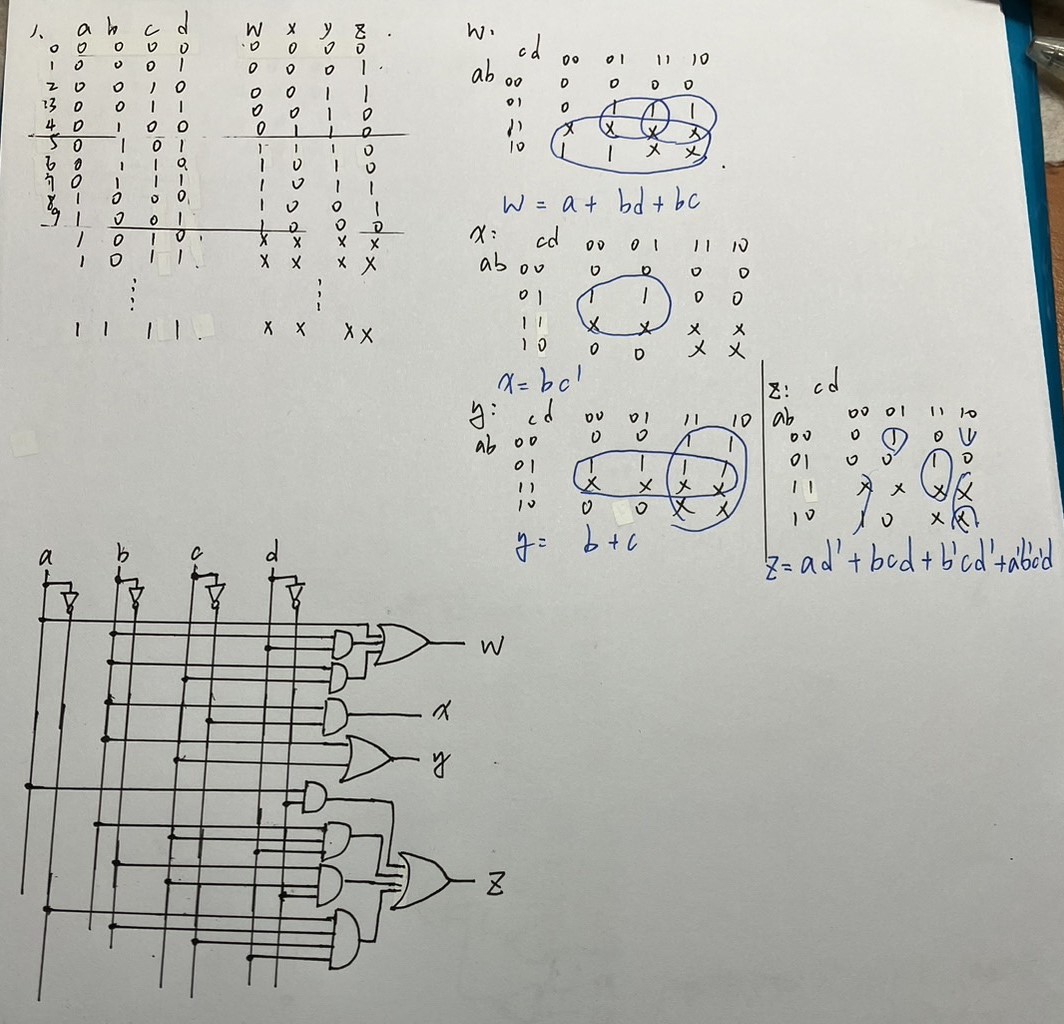
輸入: a, b, c, d, 每個大小皆為 1bit (a為MSB

輸出: w, x, y, z, 每個大小皆為 1bit (w為MSB



Design Implementation

1. Logic diagram and truth table:

見右圖🡪

abcd 10~15為don’t care condition

1. Logic Func.

用卡諾圖進行化簡

w = a + bd + bc

x = bc’

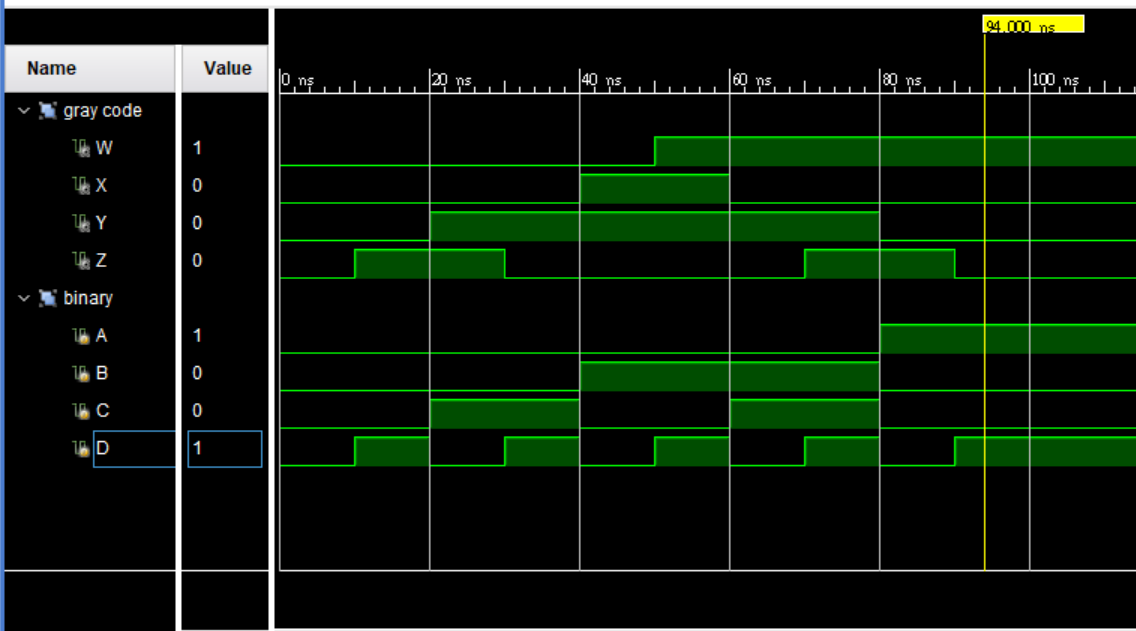
y = b + c

z = ad’ + bcd + b’cd’ + a’b’c’d

1. Code:

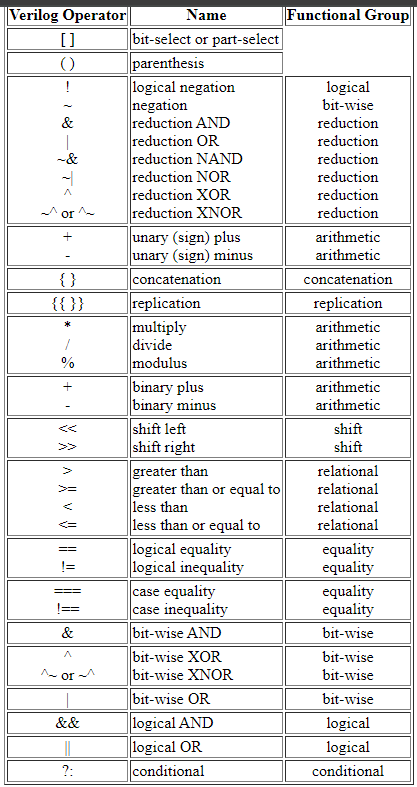
(請見附件程式碼檔案)

Discussion

圖中黃線所在的位置是最後一筆輸入和輸出。

在輸入abcd為0~9時都符合題目要求的gray code，其餘的輸入(10~15)由於題只要求10個gray code便沒有額外執行。在實作過程中全部都是用continue assignment處理，設計電路也是直接用上學期學到的組合電路的思維去做。由於Truth table和化簡過程皆沒有出錯，因此輸出的wavetable也符合我的預期。

想法:

在寫程式的過程中，我在想這些邏輯運算子能不能支援三元運算，還是只能支援二元，且彼此之間的優先度會不會影響。為了保險起見，我用括號來處理logic func.。這部分可能還是得去查一下Verilog的運算子優先權順序。(優先度見右圖🡪

另外一個想法是在寫完後仍覺得程式有個缺點，就是我覺得程式寫太死板了，缺乏彈性。若今天改為要求12個gray code，整個程式碼便要打掉重寫。但我還未想出解法，仍嘗試找出其他想法來解決這樣的問題。

Conclusion

在經過這實驗後，我學到了

* Assign的使用
* 基本邏輯運算子的使用
* Module輸出入的設定
* Testbench的寫法

Assign是基本的組合邏輯會用到的寫法，而邏輯運算子的結構也蠻簡單的。我覺得大致上和C相同，只多了一個~negative的運算，和C的 ! 寫法不同。

Module是我覺得較難的地方，和C一樣都要先宣告變數型態。但這變數型態和C就有很大的不同了，尤其是reg的應用還需要多點時間摸索。

Testbench遇到的問題大致上跟前面提到的一樣，另外麻煩的是寫Testbench要寫每個condition，這部分就由點枯燥繁瑣。

雖遇到許多問題和疑惑，但整體上還是學到了很多，且也很能感受到C和Verilog的不同。

References

<https://class.ece.uw.edu/cadta/verilog/operators.html>

如同前面提到的運算子優先權，經查詢後才發現原來OR的優先層級較AND低，所以在處理時要注意。還有NOT的運算，雖然是一元且優先層級很高，但我認為加個括號還是比較好，且能增加程式可讀性。

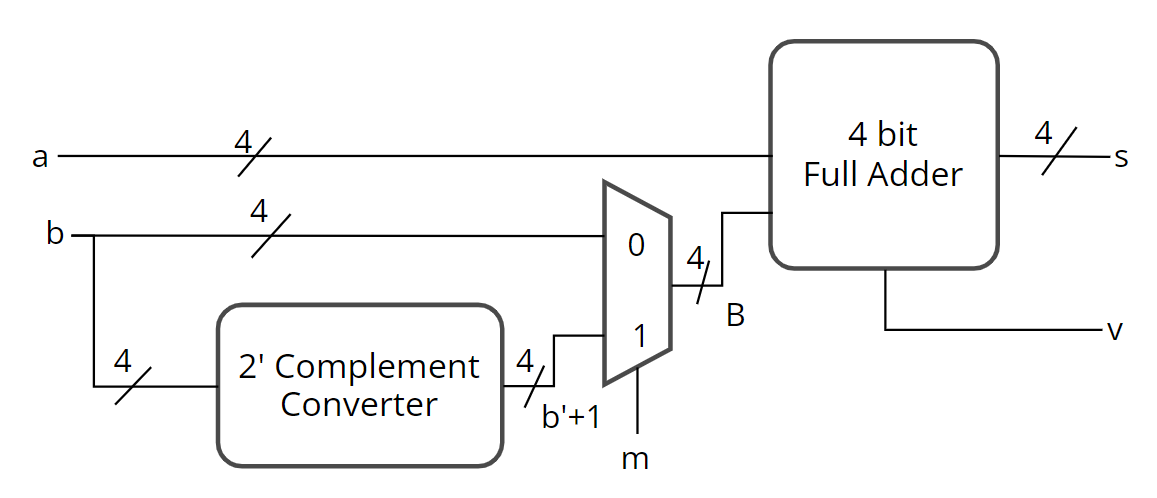
1. Design a signed 4-bit binary adder/subtractor with input a (a3a2a1a0), b (b3b2b1b0), m as theoperator control (0 for addition and 1 for subtraction); output s (s3s2s1s0), v as overflow indicator.

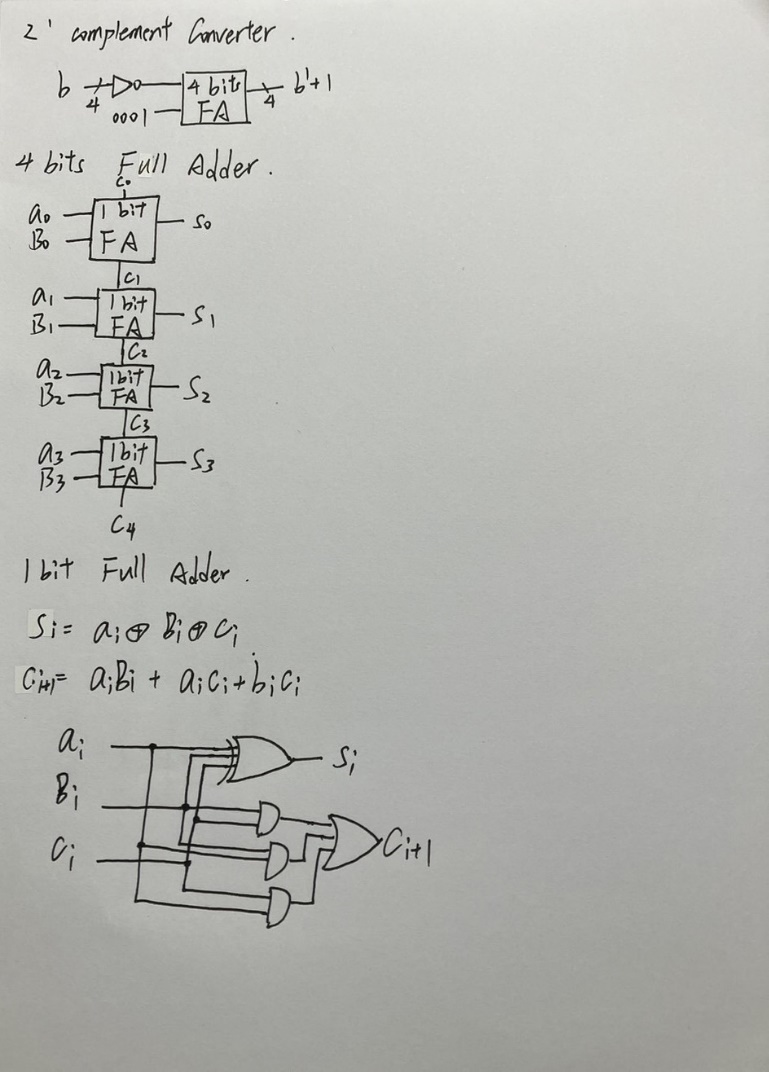
Design Specification

IO輸出入設定

輸入: [3:0]a, [3:0]b, m 前面兩個大小皆為4 bits(最左邊a[3], b[3]為MSB) , m: 1 bit

輸出: [3:0]s, v 大小分別為s: 3 bits(最左邊 s[3]為MSB), v: 1 bit



Design Implementation

1. Logic diagram and detaily design

見右圖🡪

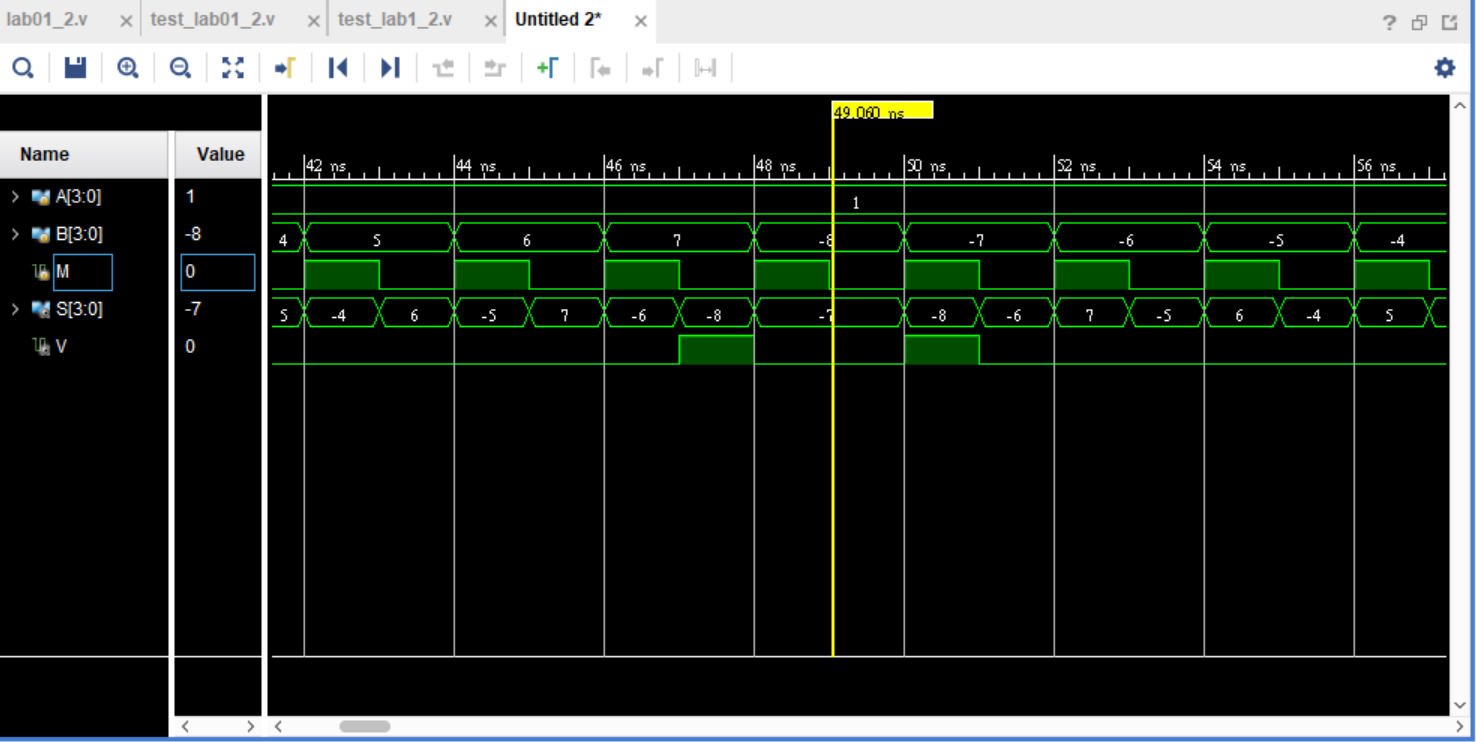
程式內我另外宣告兩個變數分別為B和c，前者紀錄經過多功器(if else)的b，後者記錄加法中每一位的進位。

(在實作Converter時是直接進行+算術運算子)

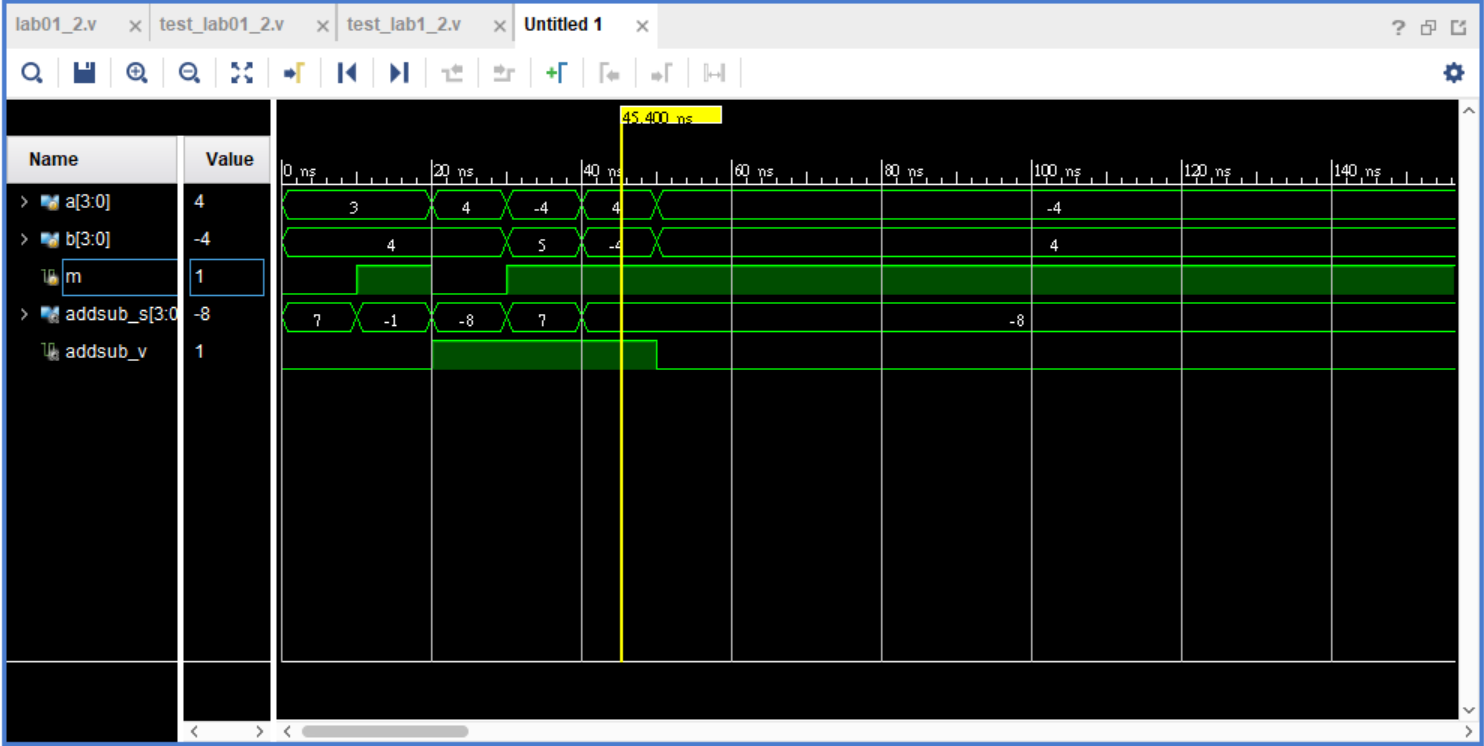
1. Logic func.
2. Code

(請見附件程式碼檔案)

Discussion



▲這張是我另外做的testbench，我窮舉了所有可能並檢查。



▲這張為老師提供的測試檔。

大致上皆符合預期的成果，若出現數學上的錯誤也會顯示Overflow溢位，運行後的結果也符合題目的要求。實作的過程因為有用到for loop迴圈，所以使用always block。在做Converter時可以直接用運算子+1來進行。這裡另外要注意的是input 變數只能是wire，所以我另外設一個變數 B為暫存器reg，才能在always block裡計算b的2’ complement。

至於4 bits FA的部分，由於要判斷是否Overflow，所以我得手動寫一個加法器，而不是直接用+運算子進行。但我後來有想到另一個寫法便是利用if else 判斷，若 正+正=負 或 負+負=正 那就可以知道有Overflow，這樣或許就可以省去建立c的空間。

另外一個困難的點就是在寫窮舉所有可能的testbench。一開始想要用for loop做，但就得必須寫成巢狀。且出來的結果會看到多兩行輸出i, j，且i和j都會變成陣列的形式。經查訊後發現for loop在Verilog裡是一次展開並一次執行，除非用task包裝後在initial block呼叫便可達到像C語言的效果。

所以我後來改用repeat的形式，並用if else 判斷當b是1111時，a加1進位，而b重新設成0。值得注意的是，延遲時間必須要設小一點，否則會跑不完。

Conclusion

在經過這實驗後，我學到了

* always block的使用
* For loop, if else的使用
* repeat的應用
* integer

經過上網查詢後，我發現if else, for loop這些流程控制都必須寫在always block裡，而always裡的變數都要設定成reg暫存器。但a, b這種變數陣列可以統一操作便令我感到新奇，和C有很大的不同。還有在用for loop時，要另外建立一個integer。這個integer就跟C的int很像，可以設定有號無號、bit長度。

最後，在經過上網查詢後才發現連for loop、if else 都要加begin, end，這樣讓我可以輕鬆地看懂某些指令運行的範圍，也大大增加了可讀性。

References

<https://ithelp.ithome.com.tw/articles/10192465>

在討論裡提到的for loop和C有很大的不同，但用task包裝後就方便許多，verilog也不會在每次執行迴圈時重新宣告i。

<https://www.chipverify.com/systemverilog/systemverilog-data-types-integer-byte>

這篇提到了更多關於int的操作，有分short , long int，且變數可以用bit\_wise()函式來查詢變數所佔的記憶體空間。

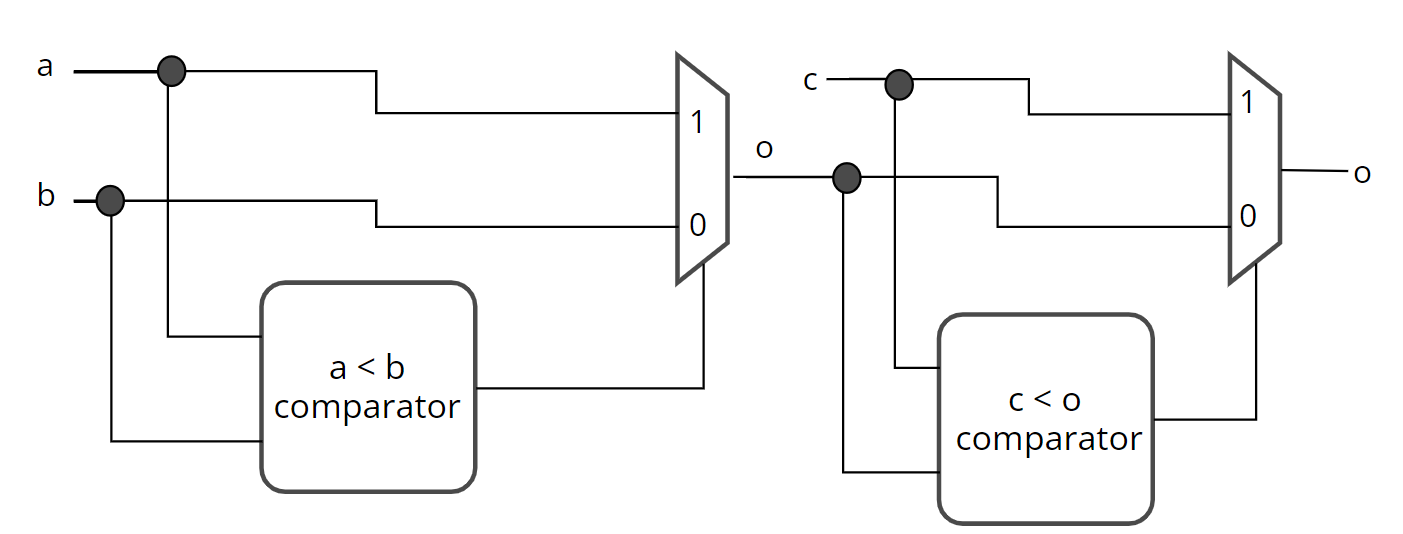
3 (Bonus) For three 3-bit signed numbers a (a2a1a0), b (b2b1b0), and c (c2c1c0), build a logic circuit to output o(o2o1o0) as the smallest number and use a given testbench for verification.

Design Specification

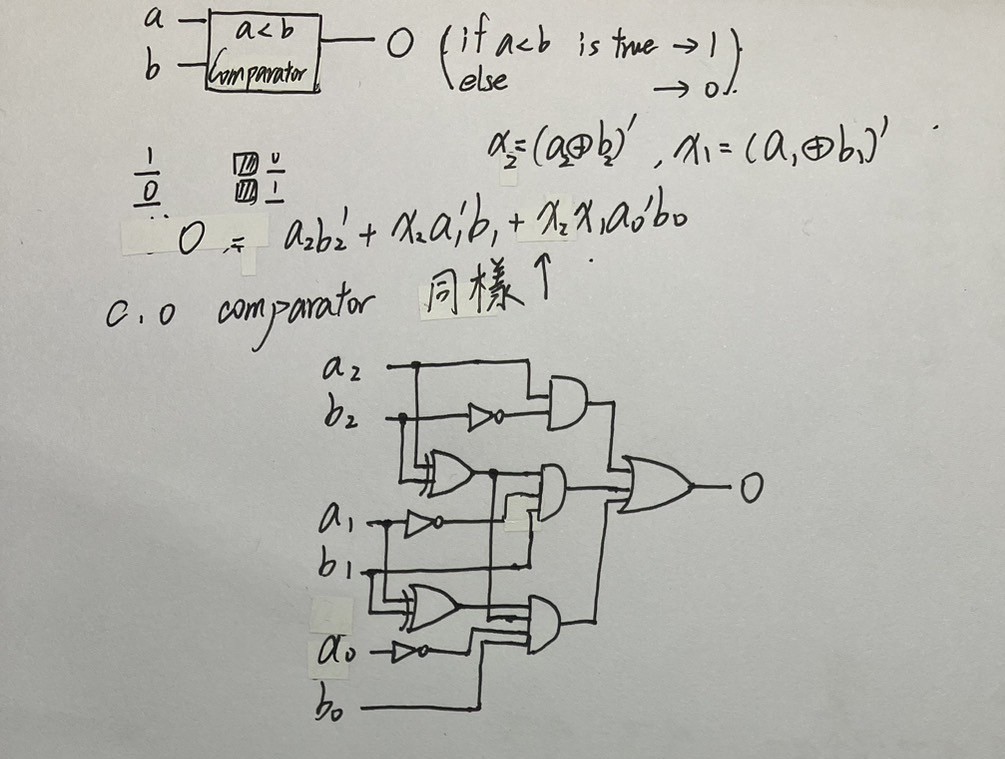
IO輸出入設定

輸入: [2:0]a, [2:0]b, [2:0]c 三個大小皆為3 bits(皆為最左邊為MSB)

輸出: [2:0]o 大小為3 bits(最左邊為MSB)



Design Implementation

1. Logic diagram

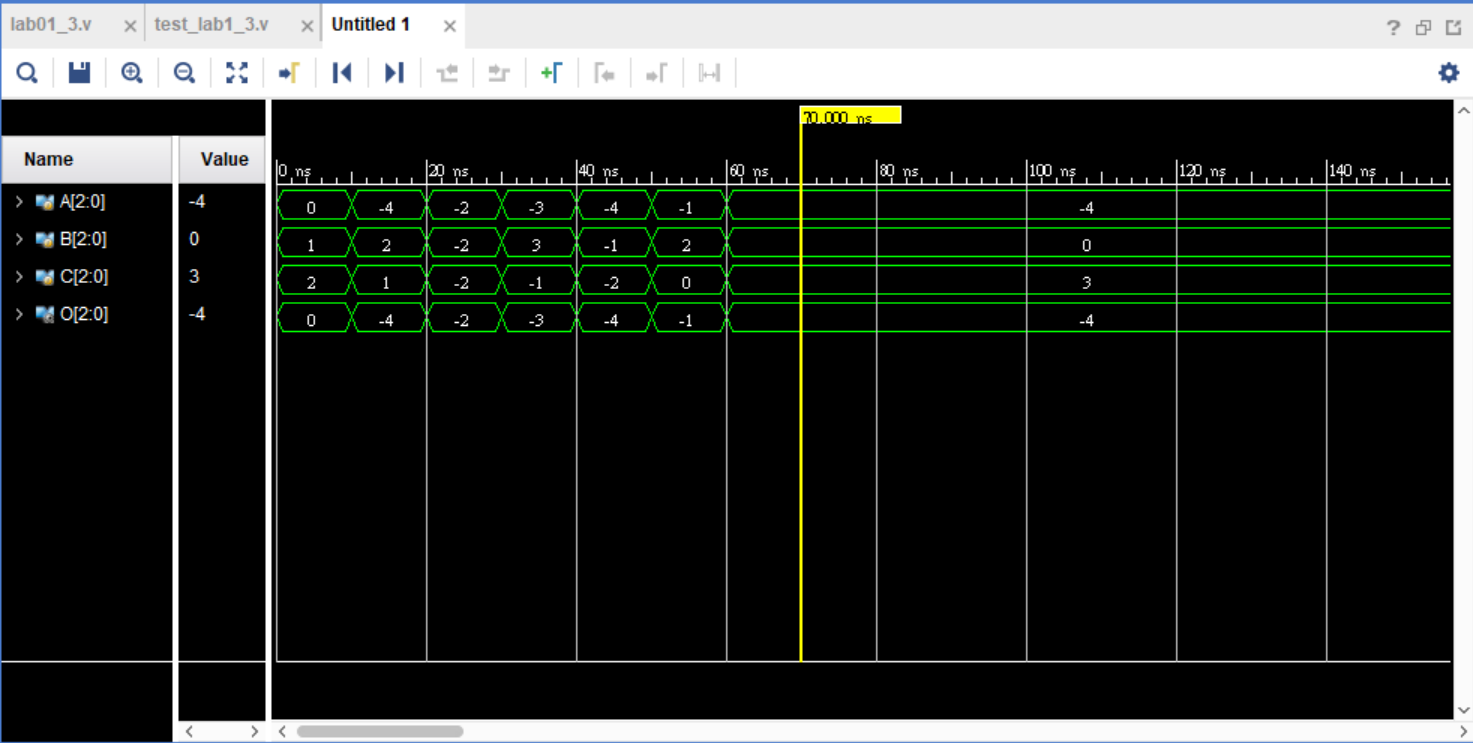
如右圖，但實際操作時直接以 < 運算子判斷。

1. Logic func.

判斷小於

1. Code(請見附件程式檔)

Discussion



▲老師提供的testbench

程式跑完符合預期的成果。能正確地在正數負數之間判斷大小關係。實作的過程我是直接用if else 加上 < 的運算子做判斷。一開始我寫成-4\*a[2]+a[1:0]的形式，但經過display檢查後發現數字出來是一個INT\_MAX。所以我懷疑是因為乘以-4的關係，但a預設為unsigned int，所以碰到負號變成溢位。後來直接在宣告input時就也同時宣告a為signed，這樣一來便能輕鬆地進行大小判斷。

Conclusion

經過實驗後，我學到了

* 注意宣告變數時的正負號運算

經過上網查詢後，我發現若一開始宣告unsigned，但在賦值給了一個負號，例如: -4’d12，對於有號數來說不會影響，但對無號數就有。-4’d12是直接賦值10100給變數，就如同老師上課講的，這樣的賦值是直接從bit設定。

但正負號的變數在進行同樣地加減運算時可能就會影響(參見References)。

這次的bonus感覺算簡單，透過signed的宣告就可以避免掉寫一大段Logic func和宣告一堆變數。希望可以學到更多好讓我們可以做實驗更輕鬆一點。

References

<https://stackoverflow.com/questions/12399991/how-does-verilog-behave-with-negative-numbers>

這裡提到了關於正負號數在進行除法時的差異。留言也提到更多相關的問題，例如: /2 和 >> 1在實際運算上的不同。