Lab 2 report

組員：110062221 李品萱

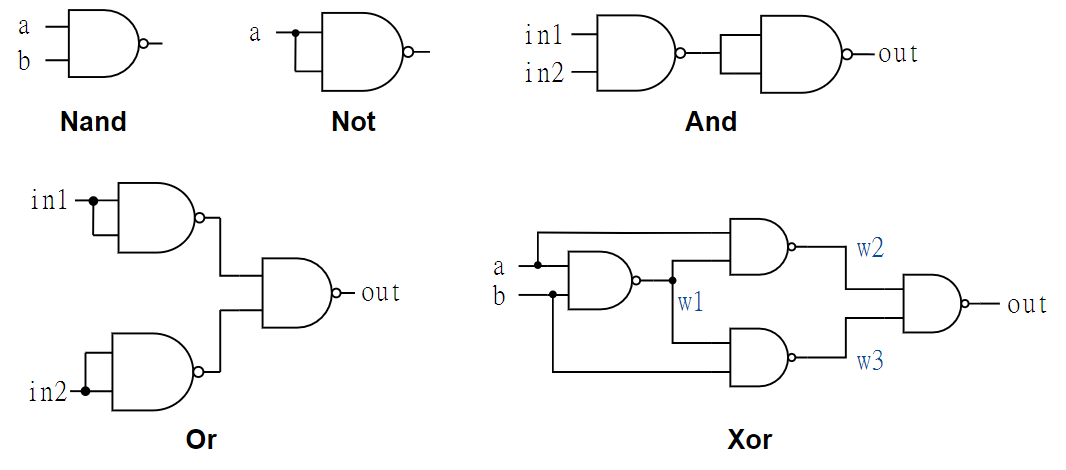
110062213 唐翊雯

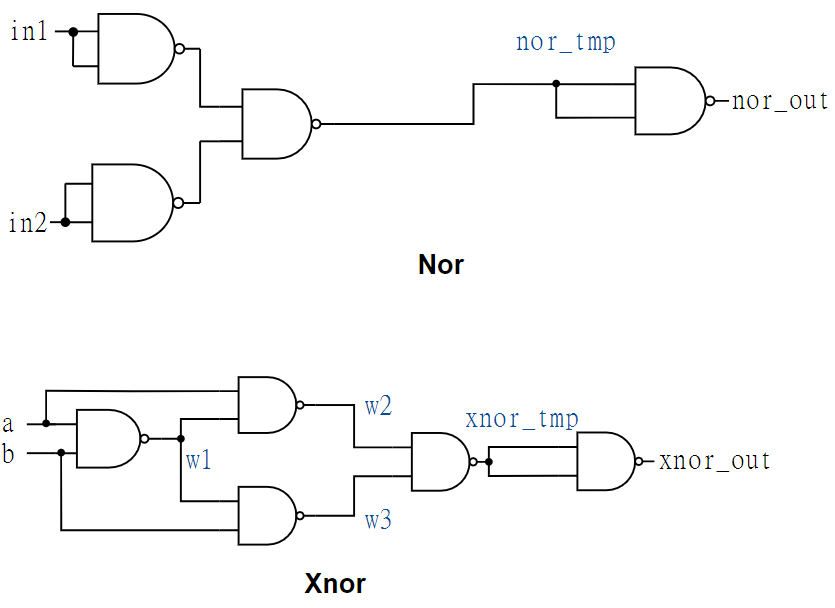
1. **(Gate Level) 8-bit ripple carry adder (RCA)**

首先，這題我們使用了Basic question 1中以NAND Gate實作的basic logic gates，以及在Basic question 3中以NAND Gate實作的Full Adder。

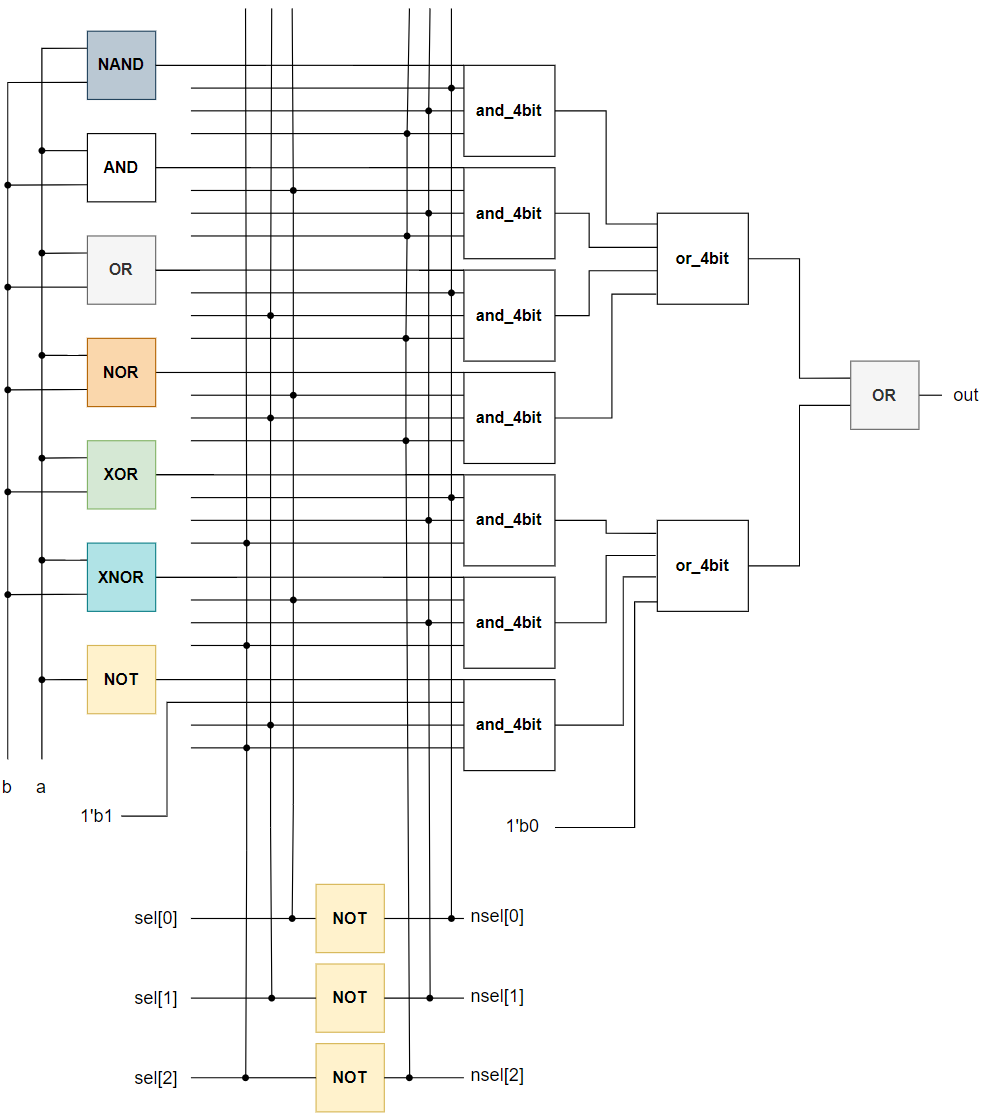
* Basic question 1：

以下為用NAND Gate實作的Basic Logic Gates:

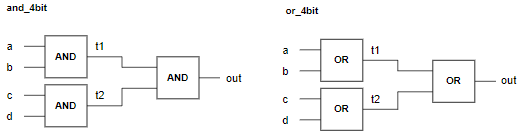




Select 的部分，我們沿用上面實作出來的basic logic gate，並用mux的想法實作，circuit如下:

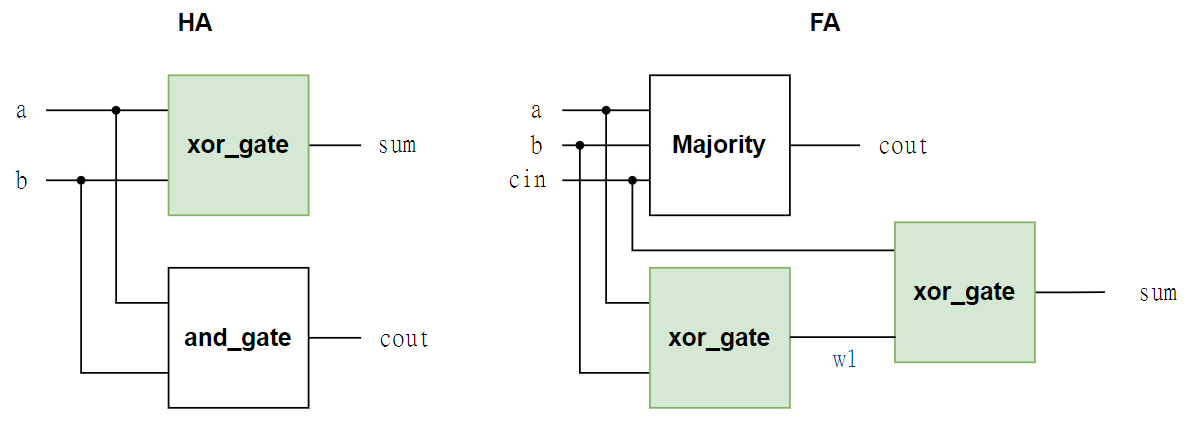


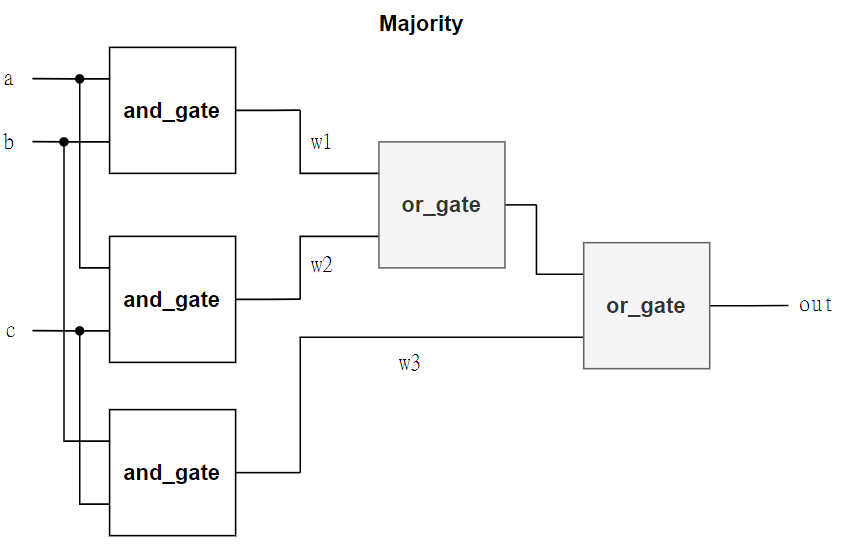
其中上面使用到and\_4bit與or\_4bit兩個module，功能分別為對四個bit取and和對4個bit取or，circuit如下：



* Basic question 3：

Half Adder、Full Adder的Circuit分別如下：



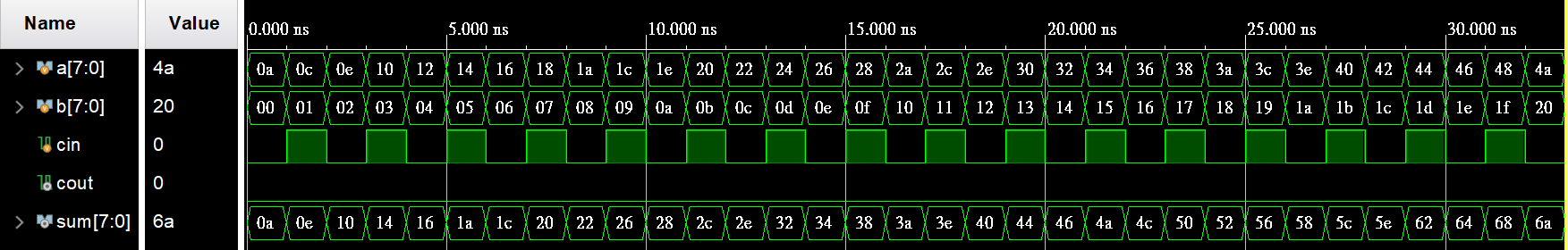


兩個Adder的差別在於，Half Adder只需要處理兩個bit相加，而Full Adder要處理三個，因此兩者在處理sum與cout的時候做法不太相同。

處理sum的時候，如果要相加的bit中，1的數量為奇數個，則sum = 1，否則sum = 0（因為被進位了，或是原本的input皆為0）。因此在實作Half Adder時，我們使用Xor來得到sum（若a != b表示有奇數個1，否則有偶數個）。而在實作Full Adder時，我們使用了兩個Xor Gate來得到sum。第一個Xor Gate若output 1，則表示a、b中有奇數個1，這時再跟cin做Xor，若cin為1則總共有偶數個1，output為0，反之總共有奇數個1，output為1；而若第一個Xor Gate output 0，則表示a、b中有偶數個1，這時再跟cin做Xor，若cin為1則總共有奇數個1，output為1，反之總共有偶數個1，output為0。因此這樣的寫法可以得到sum的值。

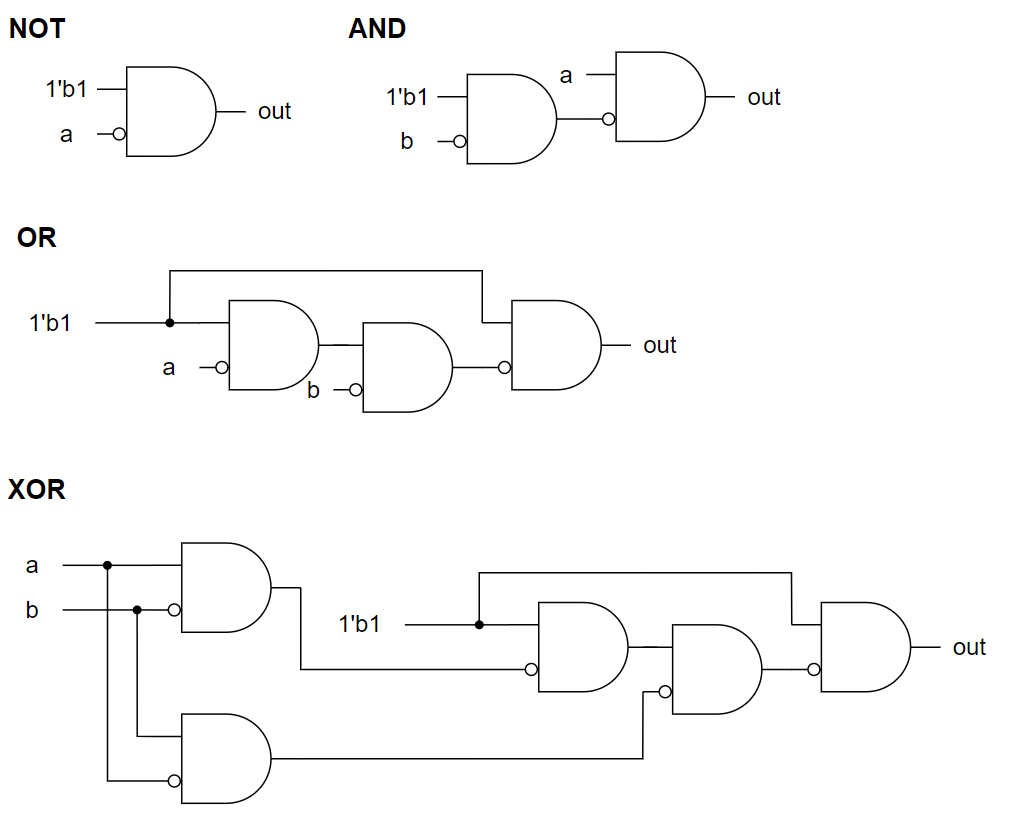
cout的部分，如果input中1的數量2，則cout為1，否則為0。因此Half Adder的cout只需要將a、b做and，即為所求。至於Full Adder的cout，只要a & b、a & cin、b & cin任一個成立，cout的值便為1，我們只要將這三項的值做or即為cout的值，而這恰好是Basic Question 2實作的circuit，可以直接拿來使用。

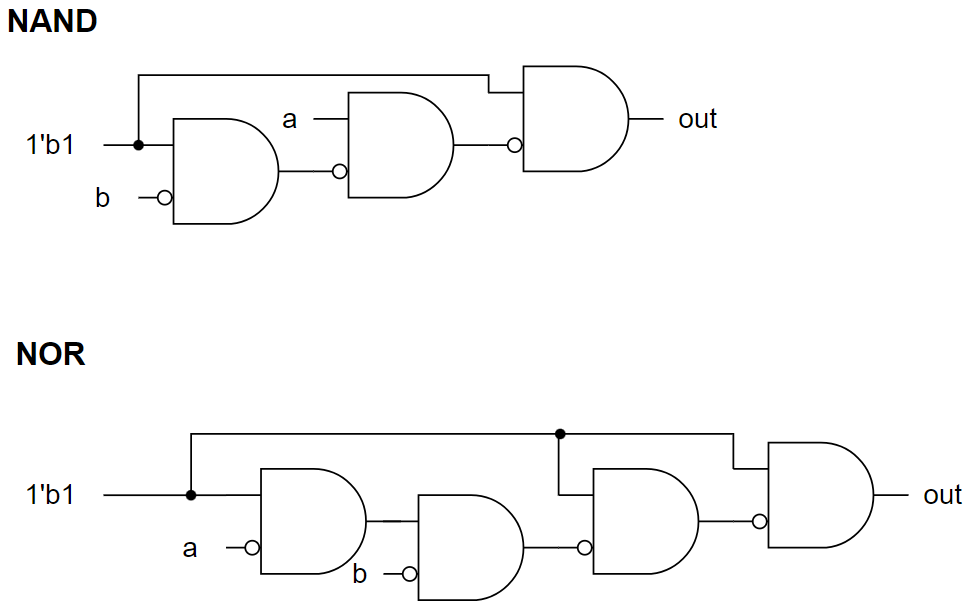
有了Basic Question 1與Basic question 3的module，在做8-bit ripple-carry adder時，我們只要像題目給定的circuit一樣，使用8個Basic question 3的Full Adder並全部接起來就完成了。Testbench的部分，我們讓a的值每次加2、讓b與cin的值每次加1，並賦予a和b不同的初始值，確認output是否正確。



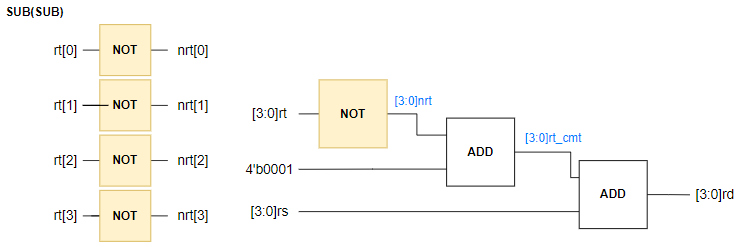
1. **(Gate Level) Decode and execute**

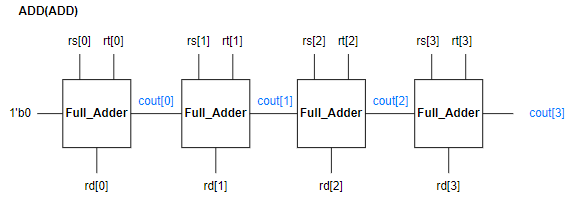
這題首先要使用給定的universal gate，實作出其他所有的basic logic gate，它們的circuit分別如下：

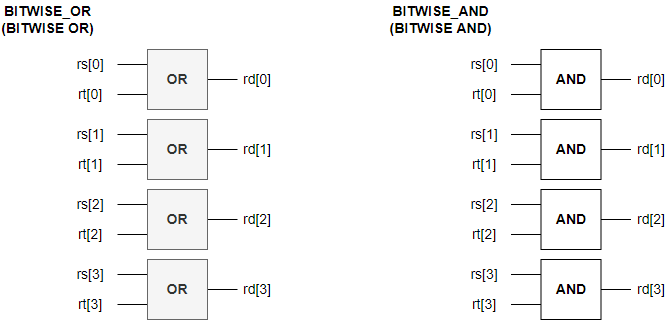


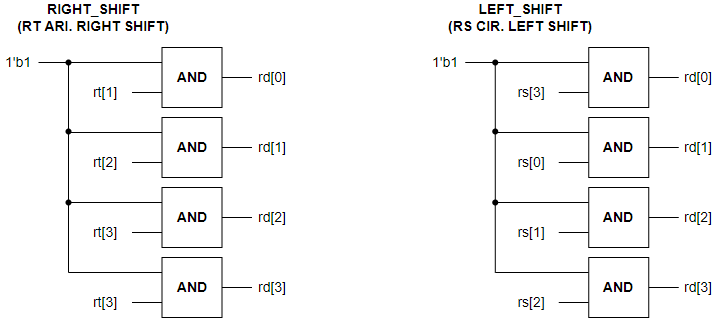


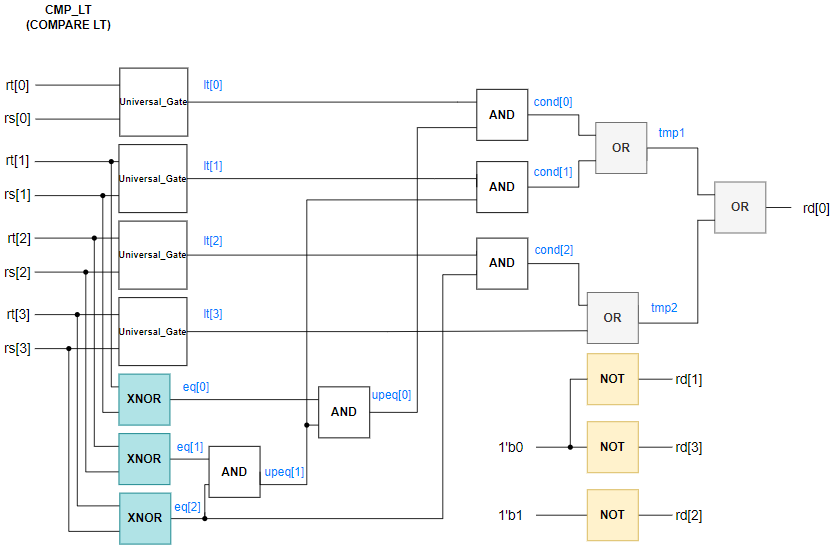
我們基於這些basic logic gate，實作了題目所要求的各個function，它們的circuit分別如下：

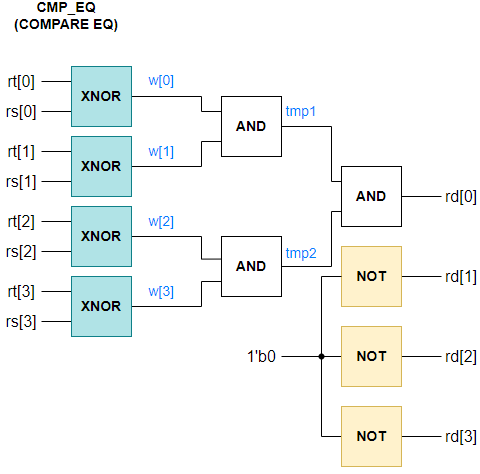












SUB 的實作使用到了二補數：首先，這個function等同於。因此我們先對rt取負號，即計算它的二補數，接著再利用ADD這個module將rs與rt相加，得到我們要的輸出。計算二補數的方法是將所有的bit都倒過來再加一，因此我們先使用之前實作的NOT gate，對rt所有的bit都取not，再將它與4’b0001，一樣利用ADD這個module將它們相加起來。

ADD這個function的實作則是使用到了4個Full Adder。基本上，每一個Full Adder的實作的方式與basic question 3的實作相同，只是把所有basic logic gates的實作從使用NAND改為使用題目規定的universal gate。如上圖把4個Full Adder接在一起，便可以實作出這個function。

BITWISE OR的實作相對單純：將rs與rt的每一個bit分別作OR，便是這個function要的輸出。BITWISE AND的實作也很相似：將rs與rt的每一個bit分別作AND即可。

RT ARI. RIGHT SHIFT的部分，題目已經說明rd的每一個bit分別要是rt哪一個bit的值，這邊我們的做法是對這些值與1’b1取AND，輸出即為所求。RS CIR. LEFT SHIFT也很相似，一樣，題目已經說明rd的每一個bit分別要是rs哪一個bit的值，對這些值與1’b1取AND，輸出即為所求。

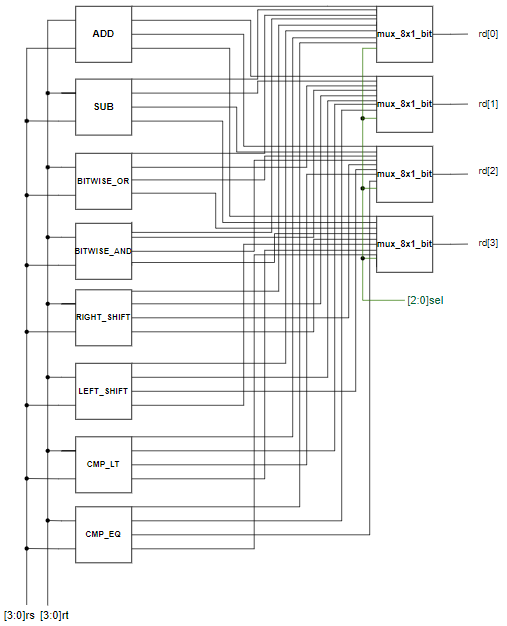
COMPARE LT的部分，我們實作的想法如下：比較兩個二進位數字的大小，首先要比較最高位的bit，若不同則該bit比較出來的大小關係即為兩數的大小關係。若相同，則繼續往下一個較低位的bit比較，依此類推。因此若要符合，必須符合下列條件的其中一項：

因此，我們分別用logic gate得到上述關係式的結果，再將所有結果取OR，output即為rd[0]。 rd[3:1] 的部分，題目已經指定每個bit的值，這邊我們直接使用三個NOT gate：輸入分別放題目要求的值的相反；輸出分別放rd[0]、rd[1]、rd[2]，便能達到題目要求。

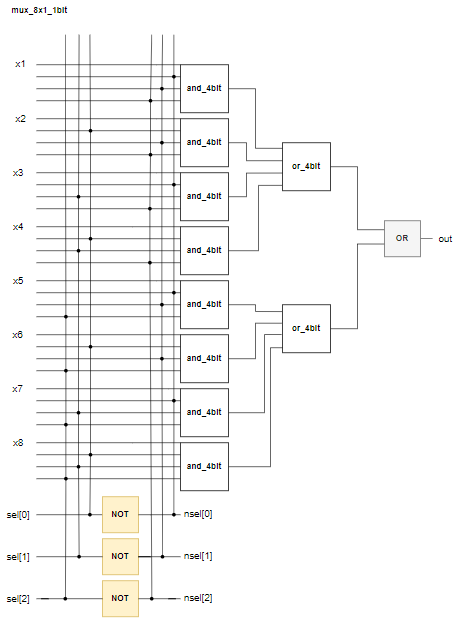
COMPARE EQ的實作，我們的想法如下：比較兩個二進位數字的是否相等，要比較兩數的每一個bit是否都相同。也就是說，若，下列的值皆須為true：

判斷相等的部分我們使用XNOR實作，若兩bit作XNOR的值為true，即代表兩bit相等。最後把上述4個值取AND，即為rd[0] 的值。rd[3:1] 的部分與COMPARE LT的部分相同，使用三個NOT gate給rd[3:1] 題目所要求的值。

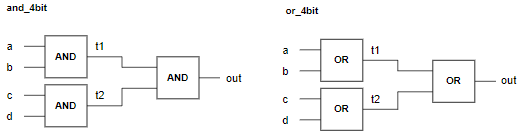
最後，根據sel的值，我們要給出相對應的output。在得到各個function的output之後，我們使用了4個8 to 1的MUX (1 bit)，分別得到output的每個bit的值，circuit如下：



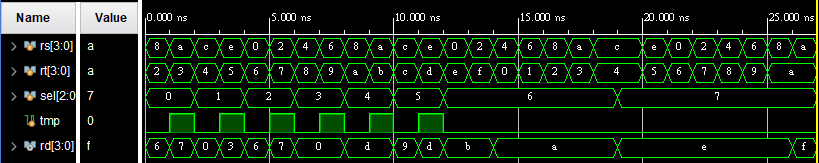
而8 to 1的MUX (1 bit)，其circuit如下：

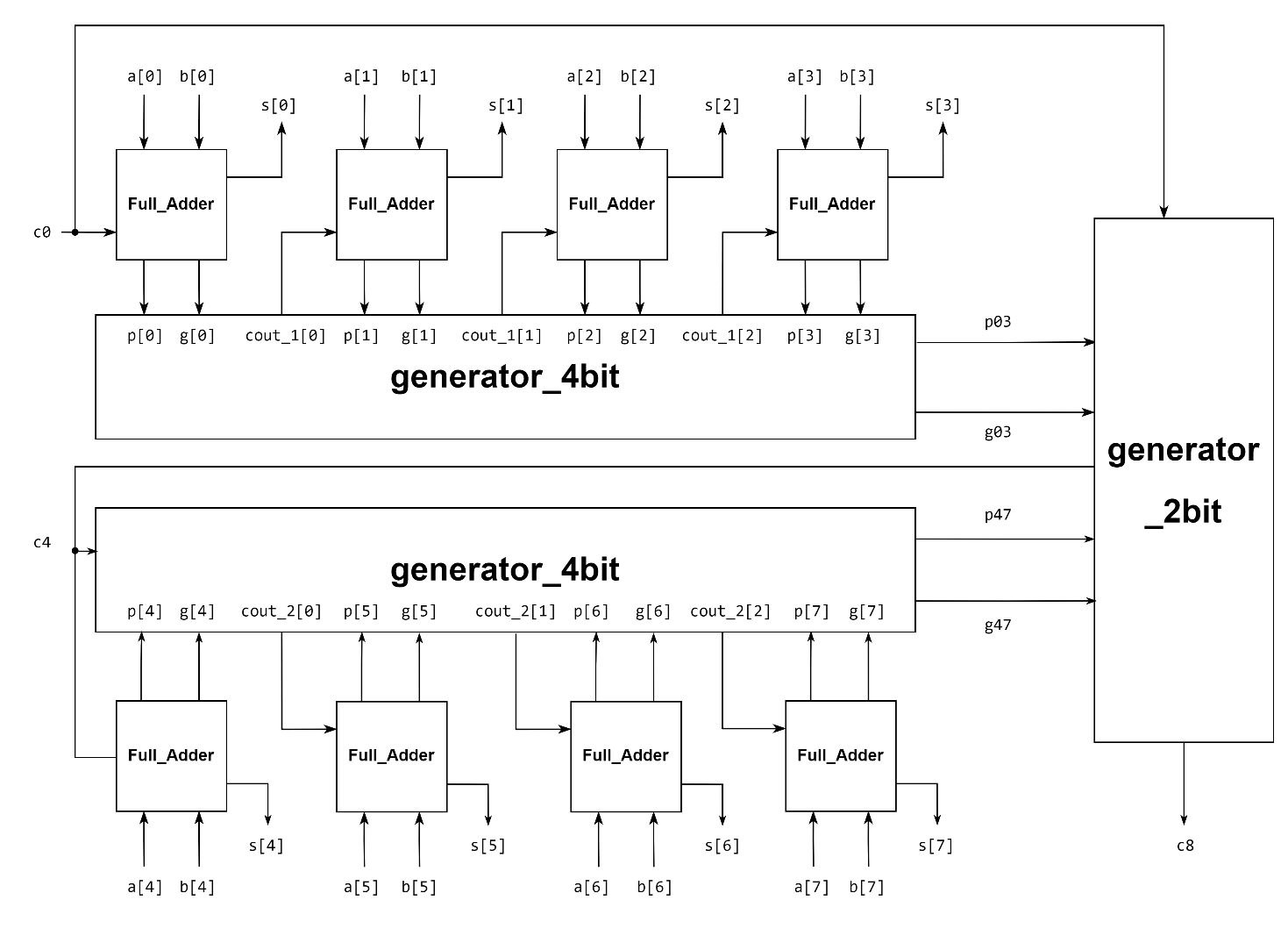


其中使用到了4 bit的AND以及4 bit的OR，circuit分別如下：

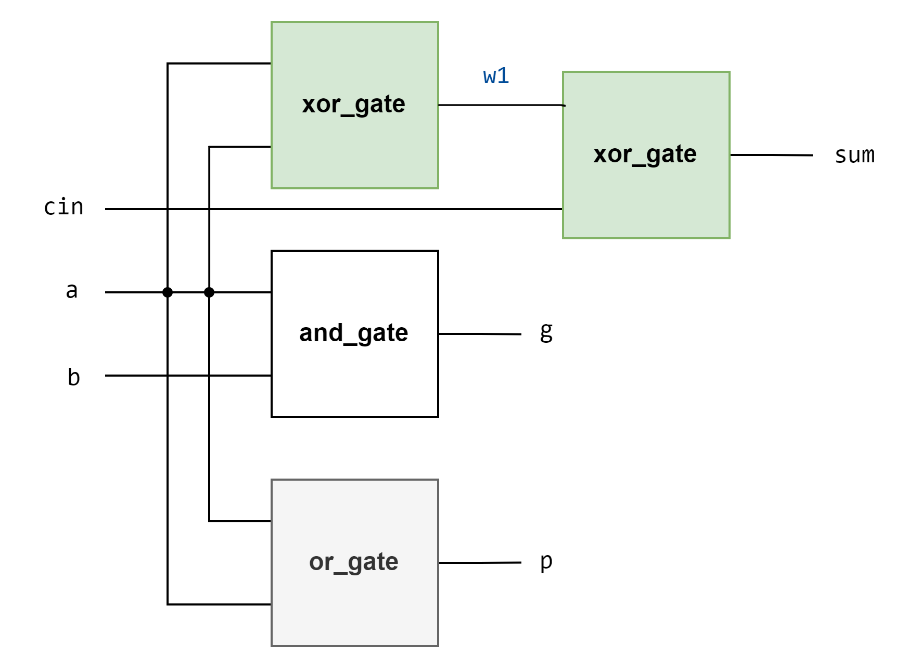


Testbench的部分，對於前6個function，我們讓rs的值每次加2、讓rt的值每次加1，並賦予不同的初始值下去確認每個function的output是否正確。而對COMPARE LT的部分，我們多跑了幾次，以確認在、以及的情況下，output 都會是正確的。COMPARE EQ的部分，我們也特別assign了rs與rt的值，以確保在以及的情況下，output都是正確的。

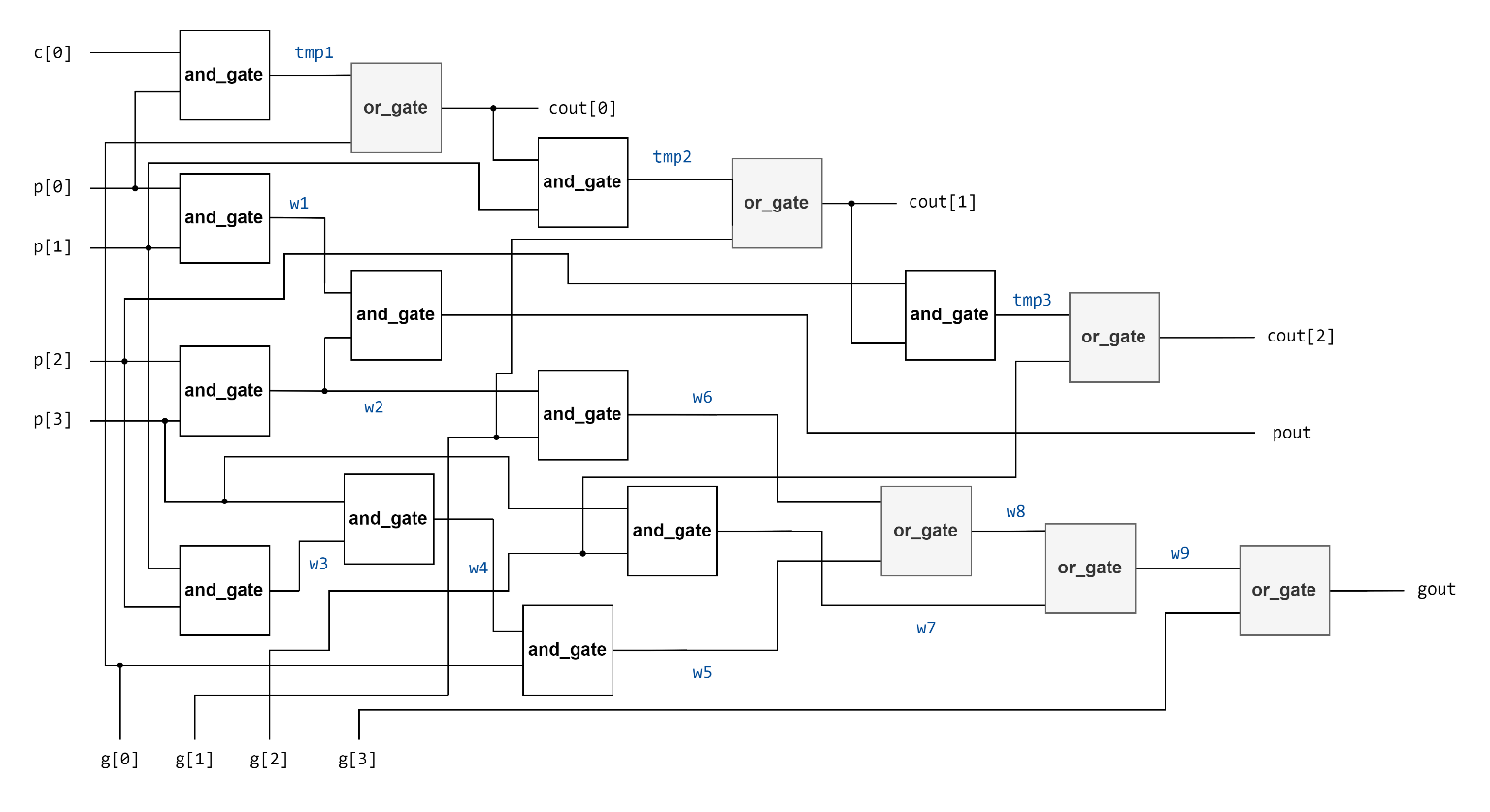


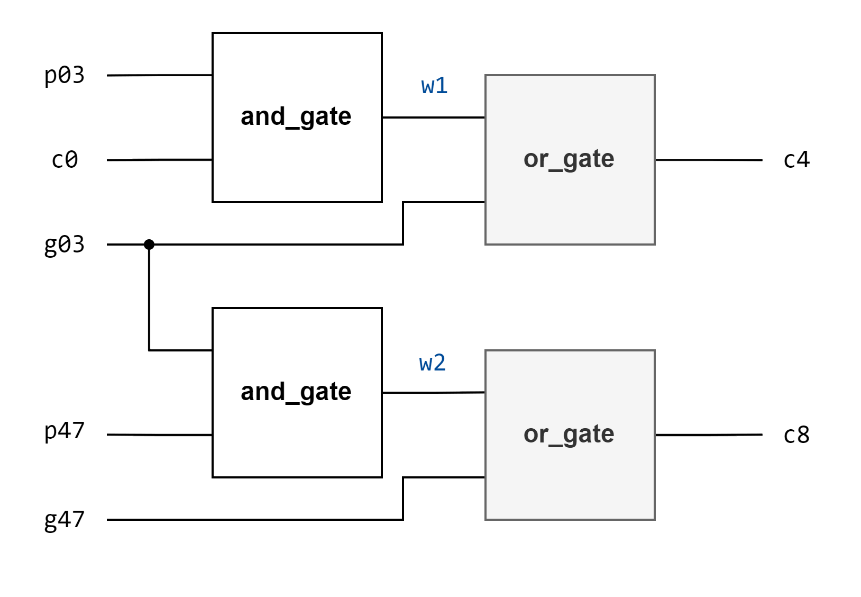
1. **(Gate Level) 8-bit carry-lookahead (CLA) Adder**

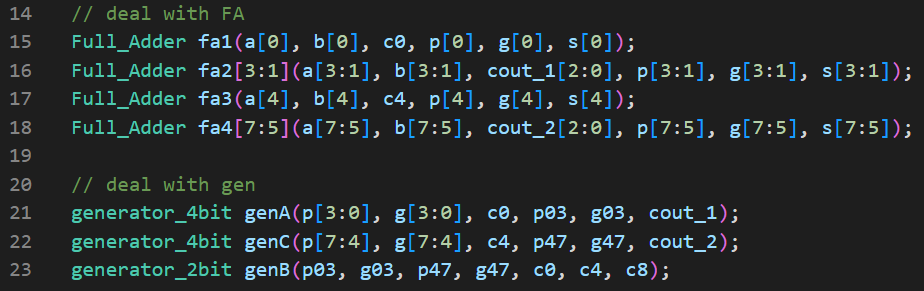
如上圖，這題我們需要八個Full Adder、兩個4-bit Carry-Look-Ahead Generator與一個2-bit Carry-Look-Ahead Generator。CLA特別的地方在於它的p, g及generator，讓它可以提前知道兩數相加時的carry，且每個bit的Full Adder可以直接拿自己的carry in同時進行運算，而不用像RCA需要等待上一個bit做完之後才知道carry out的值，能夠減少延遲時間，也不須依賴前面的計算結果來獲得值，因此較為有效率，這也能夠從電路圖中看出來，同樣是8bits的adder，RCA需要的and gate及or gate個數相較之下比CLA多，因此在運算上也會較為耗時。

為了實作generator，我們需要先做出p與g兩個訊號，p代表的是propagate，負責將兩個bit做or，也就是說如果兩個bit中至少有一個bit為1就會是1，此時當carry in 是1時就會將此carry往下傳遞；g代表的是generate，負責將兩個bit做and，意即當兩個bit都是1時必會出現carry out。而在設計的部分我們將input丟進Full Adder中讓它output, p, g, sum再將p, g傳入generator中，這部分的Full Adder如下圖。

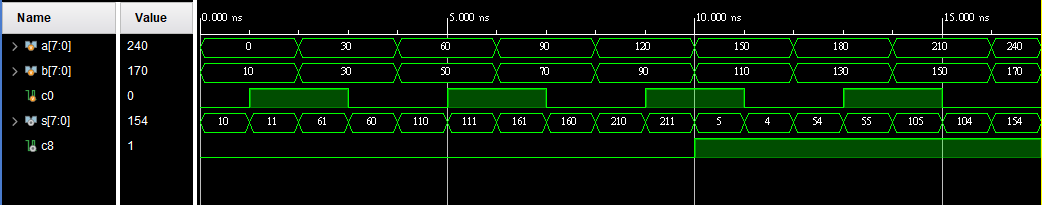
在計算進位的部分，它的equation為：，經過計算可得到原式 ，其中 即為 ，而 即為 ， 及 同理，於是我們可以知道c4即為，而c8即為 。因此在code的部分，我們傳入4bit的p及g，依照上面equation的框架做出 、 及cout，4bit generator的circuit diagram如下。

而後，4bit generator的pout及gout會再傳入2bit generator做上述c4及c8的equation再output出c4及c8，如下圖。

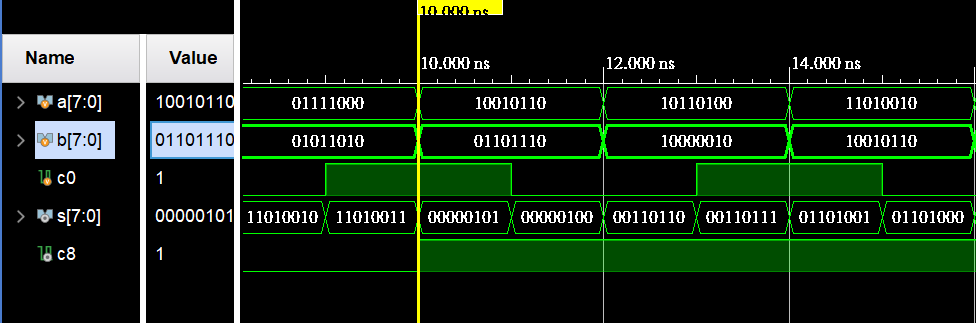
在code的部分，考慮到c4與其它cout接出來的地方不同，為了寫code上的方便，我們單獨處理c4，並將cout分為cout\_1及cout\_2兩部分，各為4bits，下圖為我們在Carry\_Look\_Ahead\_Adder\_8bit這個module中呼叫Full Adder及generator的部分。

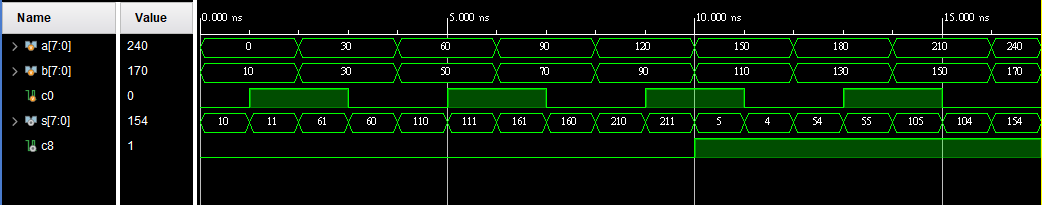


在test的部份我們用clock延遲的方式讓c0會在a與b沒有變動時出現0跟1兩種情況，以便確認carry這部分沒有問題，如下圖。



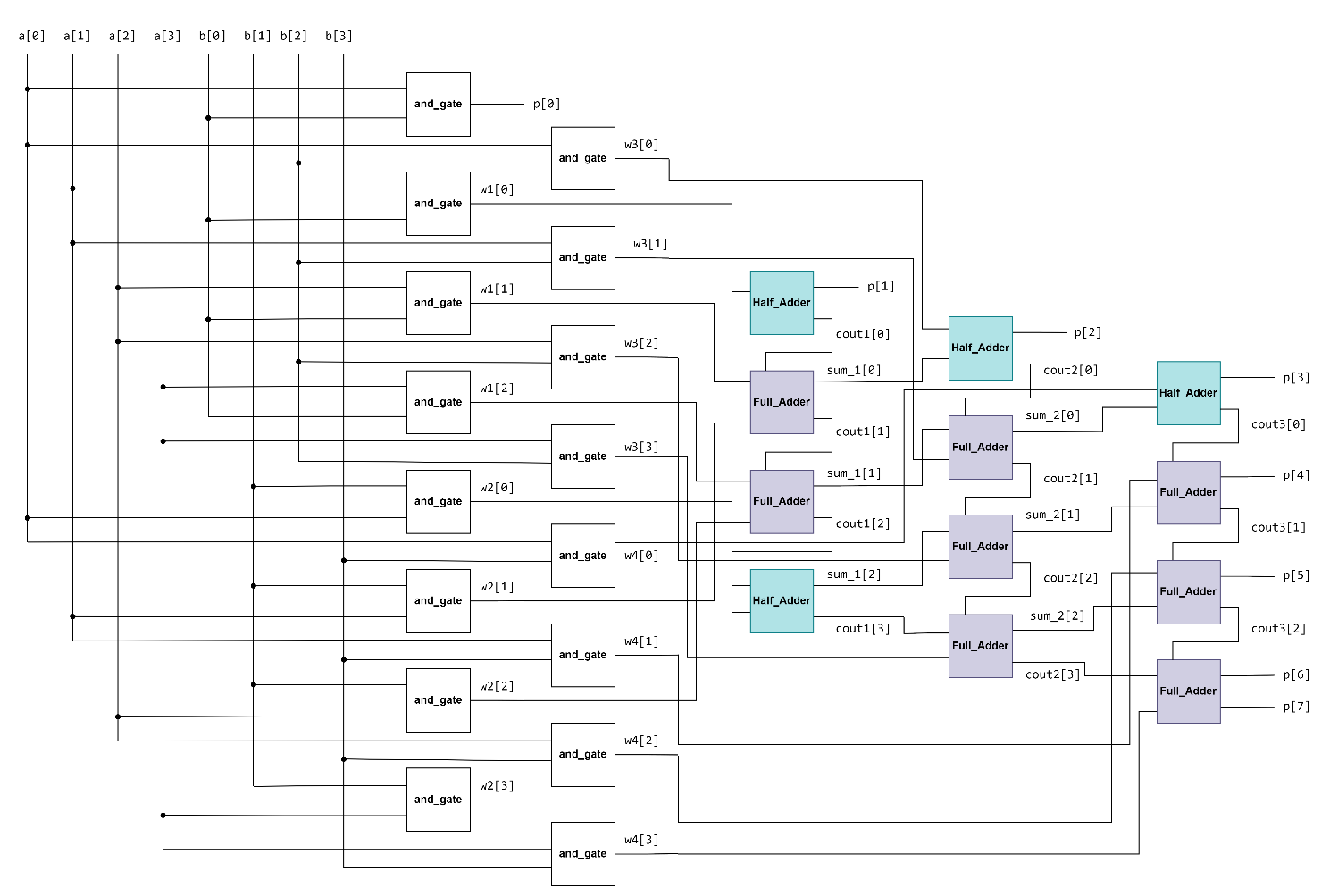
且為了觀察c8的狀況，我們這邊選擇讓a每次加30，b每次加20，而當s產生carry out，也就是c8為1時，可以發現s的數值無法完整呈現a、b相加該有的結果，這是因為s只有8bits，因此最大值以十進位表示為255，超過則會發生溢位的狀況，此時若以二進位觀察可以發現s是a+b的後面8個bits，如下圖。若將c8與s cascade起來就會得到正確的結果。

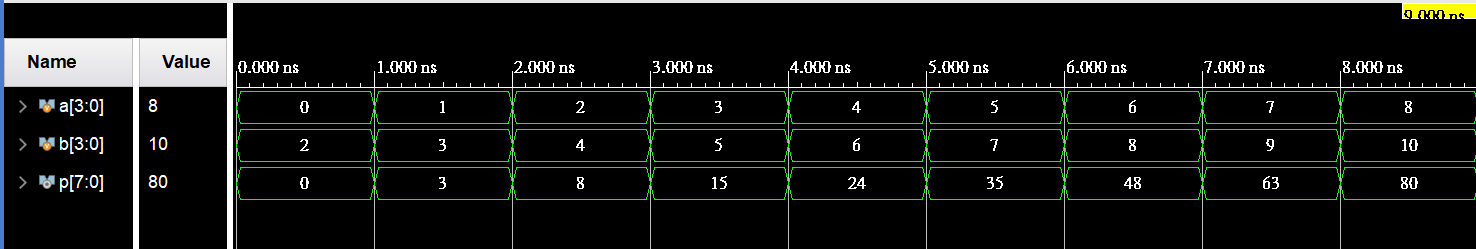




1. **(Gate Level) 4-bit multiplier**

這題我們需要設計一個4bit的multiplier，其中我們會用到前面時做過的Full Adder及Half Adder，它的原理與直式乘法類似，透過計算每個partial product再將他們送至adder中做運算得到最後的值。如下圖，首先我們將a與b每個bit互相and，其中，a[0] · b[0] 即為p[0]，其他bit做完and的結果我們以wire接出來，由於我們這邊的作法是將a作為被乘數，b作為乘數，因此我們將a[i] · b[j] 對應到wj+1，接下來先將w1接出來的訊號送入圖中最左邊那一個column的adder中，這邊我們採用兩個Half Adder及兩個Full Adder，因為該column最上方的Half Adder不會有carry in，我們用設計上比較簡單的Half Adder進行運算，這個Half Adder的output即為p[1]。而接下來兩個bits都會有carry in，所以都採用Full Adder；每個Full Adder會將sum送進右方的Adder中，並將carry out往下傳，類似Ripple Carry Adder；值得注意的是該column最下方的也是使用Half Adder，其原因並不是沒有carry in，而是將carry in視為Half Adder input的其中1 bit，將此carry in與partial product（也就是w2[3]）相加。第二個column與前述類似，最上方的Half Adder即會算出p[2]，而最下方我們使用的是Full Adder，因為這邊要接左方的cout1[3]、w3[2]及前一個Full Adder的carry out（即cout[2]）做運算。第三個column與左方相同，這邊的sum即為p的第3個bit到第7個bit。

這邊testbench我們就簡單的讓a跟b每次加1，觀察p的值是否符合a與b相乘的結果。



1. **An exhaustive testbench design**

這一題我們要design一個可以正確找出錯誤的testbench，所以我們設計了一個faulty的RCA與題目要求的testbench。

首先，在testbench的部分，由於題目要求要枚舉所有可能，參考template上的example後，我們用三層repeat來實作，首先固定cin為0、b為0，讓a先從0跑到15，依序對b為1到15的狀況讓a從0跑到15，最後在對cin為1時做相同的事，這樣就會覆蓋到a[3:0] + b[3:0]的所有可能情況。在最內層的repeat中，我們先延遲一個clock，在將error設為0，這樣做的用意是對每個要檢查的結果做error的初始化，且符合題目要求的，若檢查結果正確則在input進來1ns後將error設為0。接下來我們用if對RCA的output進行檢查，這邊的想法是既然我們要確認output的正確性，那我們就先用input做出正確的結果，再檢查output與結果是否相等，一開始我們是想到bitwise operation可以做出正確結果，但要對每個bit做太麻煩了，後來想到其實直接如下圖將a、b、cin加起來就好，此時我們再將cout與sum做串接，並用嚴格不等於確保他們是逐位相等且只會回傳0與1，在if條件成立即為偵測到錯誤，此時因為前面已經延遲過一個clock，所以會馬上raise error；if條件判斷完後，我們讓他維持4個clock再改變a的值，因此加上下一次進repeat後即會有維持5個clock的error及input訊號。

一張含有 文字 的圖片

自動產生的描述

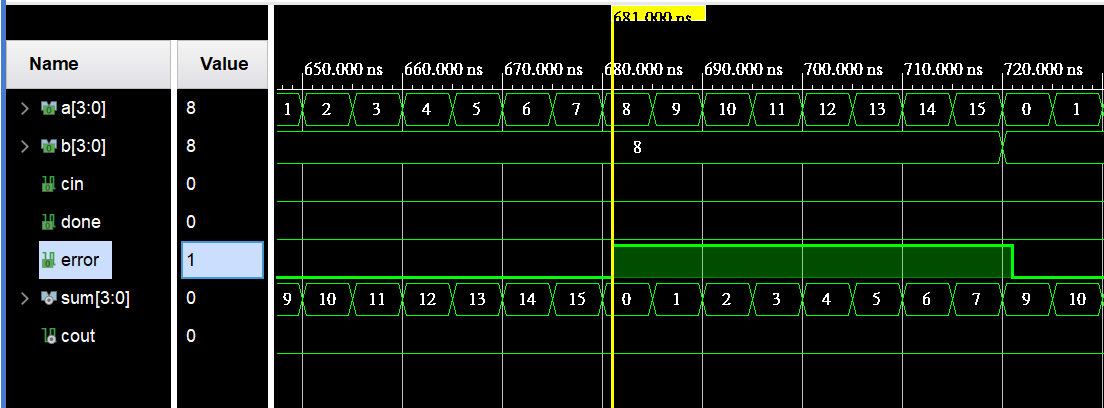
為了檢察我們的testbench設計是否正確，我們先用正確的RCA確定它不會raise error也有正確的呈現done訊號，再用一個會產生錯誤結果的RCA進一步檢查，我們假設了兩種錯誤情況。

第一種是cout出錯，如下圖，我們讓這個RCA只在c3及c4都為1時才會產生cout。

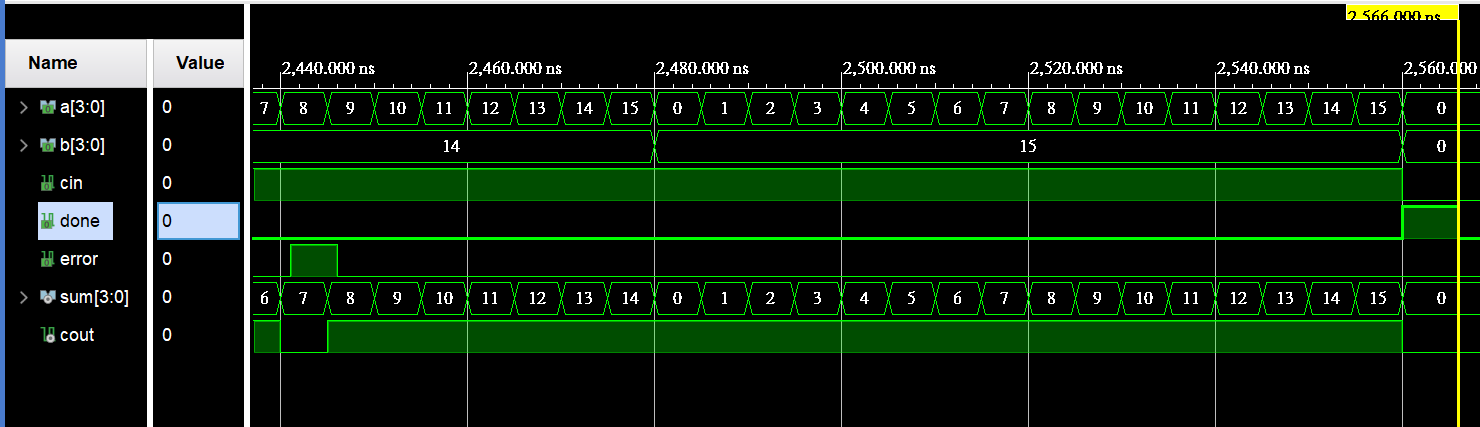
一張含有 文字, 螢幕, 電子用品 的圖片

自動產生的描述

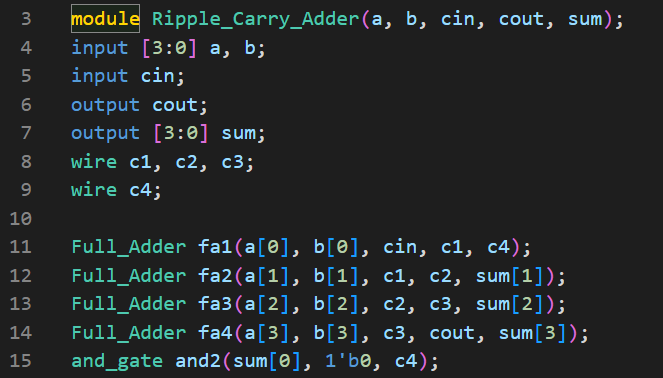
這個情況的simulation結果如下圖，在b = 8，a = 8到a = 15時cout沒有呈現1，所以在偵測到錯誤1ns後，error就會為1，且由於他在這段區間內都是錯的，因此error會一直呈現high的狀態，一直到偵測到正確值1ns後才會變為0。



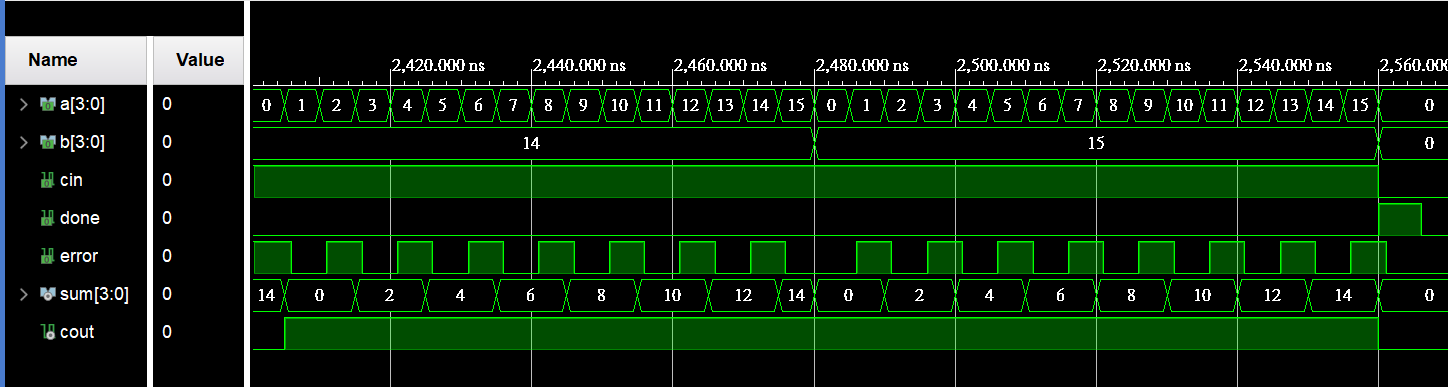
這邊我們也能看到在a = 8，b = 14時，cout輸出0，因此會raise 5 ns的error。在所有可能都跑過後，done會在偵測到最後一個input 5ns後變為1，並如同spec圖中在5ns後恢復成0。



第二種情況是sum出現錯誤，如下圖，我們將原本sum[0]的位子換成c4，並將它與0做and，造成它在應該輸出1時反而輸出0。



這個情況的simulation結果如下圖。由於我們在枚舉時是每次加1，而raise error與否決定在sum[0]，因此error的狀況多數時候會交替出現，這個RCA的設計也是為了可以更清楚的看到在correctness部分的波型呈現。

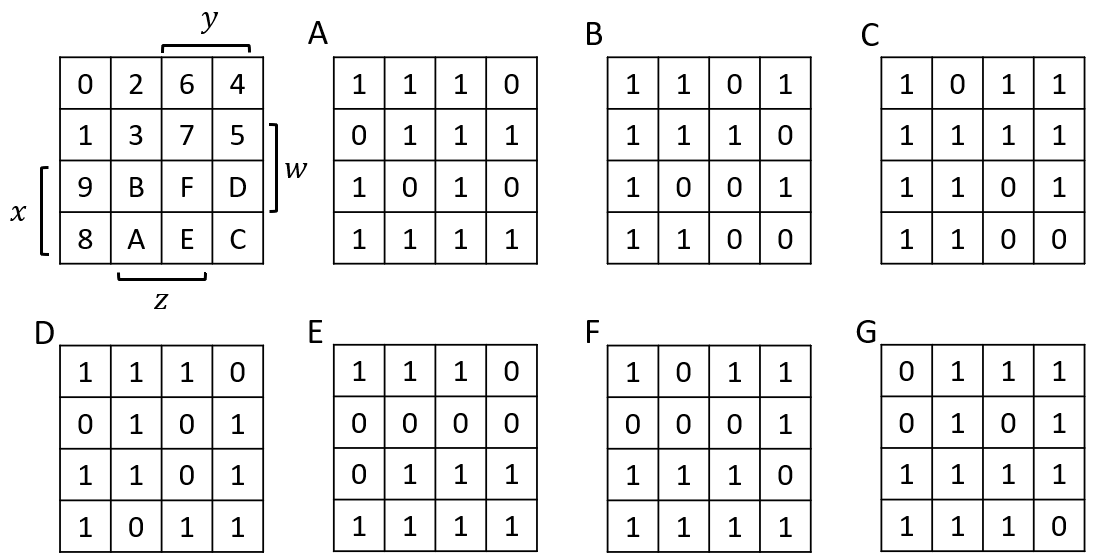


1. **FPGA: (Gate Level) Decode and execute**

這題是要沿用Advance question 2的module，並將結果顯示在FPGA板上，我們使用了gate-level circuit來完成這題。由於要用7-segment display來顯示我們的output，首先我們要對每個數字及各個segment做對應。對於0F的每一個數字，它們對二進位以及7-segment的對應分別如下：

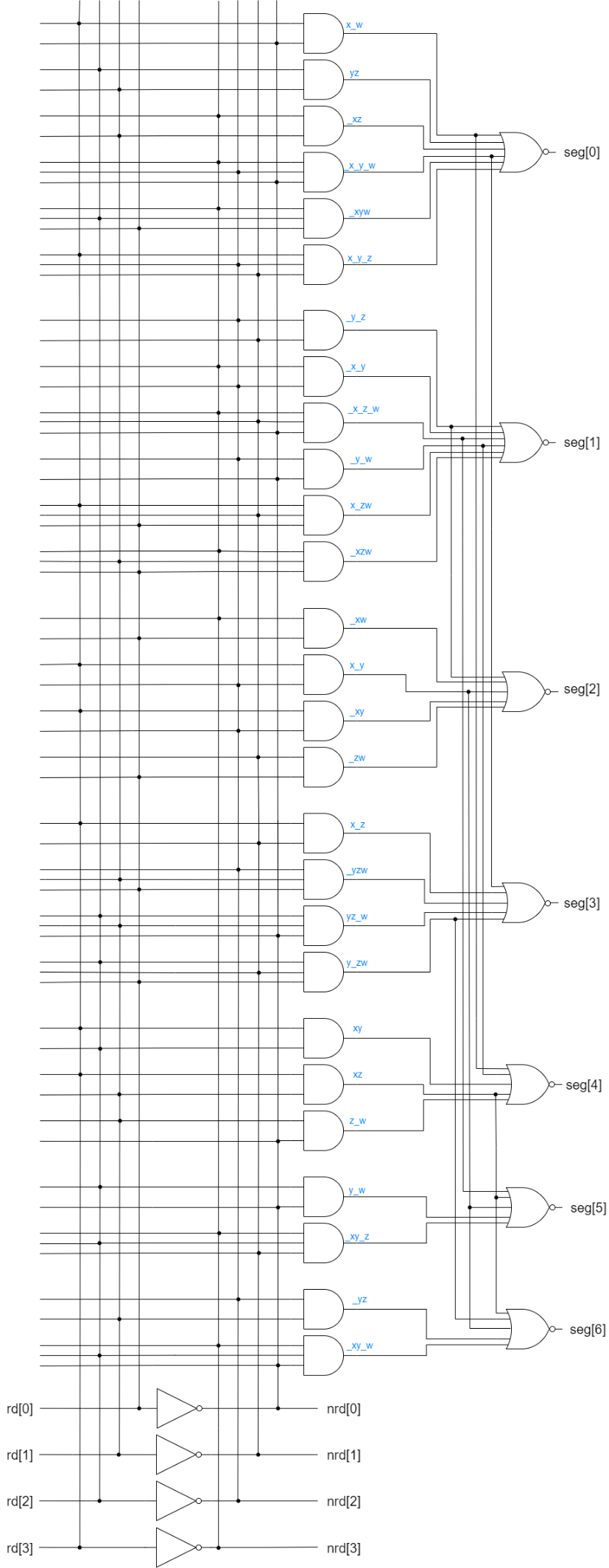
|  |  |  |
| --- | --- | --- |
| **Hex** | **Binary** | **7-segment** |
| 0 | 0000 | ABCDEF |
| 1 | 0001 | BC |
| 2 | 0010 | ABDEG |
| 3 | 0011 | ABCDG |
| 4 | 0100 | BCFG |
| 5 | 0101 | ACDFG |
| 6 | 0110 | ACDEFG |
| 7 | 0111 | ABC |
| 8 | 1000 | ABCDEFG |
| 9 | 1001 | ABCDFG |
| A | 1010 | ABCEFG |
| B | 1011 | CDEFG |
| C | 1100 | ADEF |
| D | 1101 | BCDEG |
| E | 1110 | ADEFG |
| F | 1111 | AEFG |

我們對7-segment的每一個線段，去紀錄它們分別在output為哪些值時會亮，並使用k-map去寫出function。下列的k-map，1表示要亮、0表示不亮。x、y、z、w 分別代表 rd[3]、rd[2]、rd[1]、rd[0] 的值，A~G則分別代表每一個segment：



將每個k-map化簡之後，可以得到如下sum of product的算式：

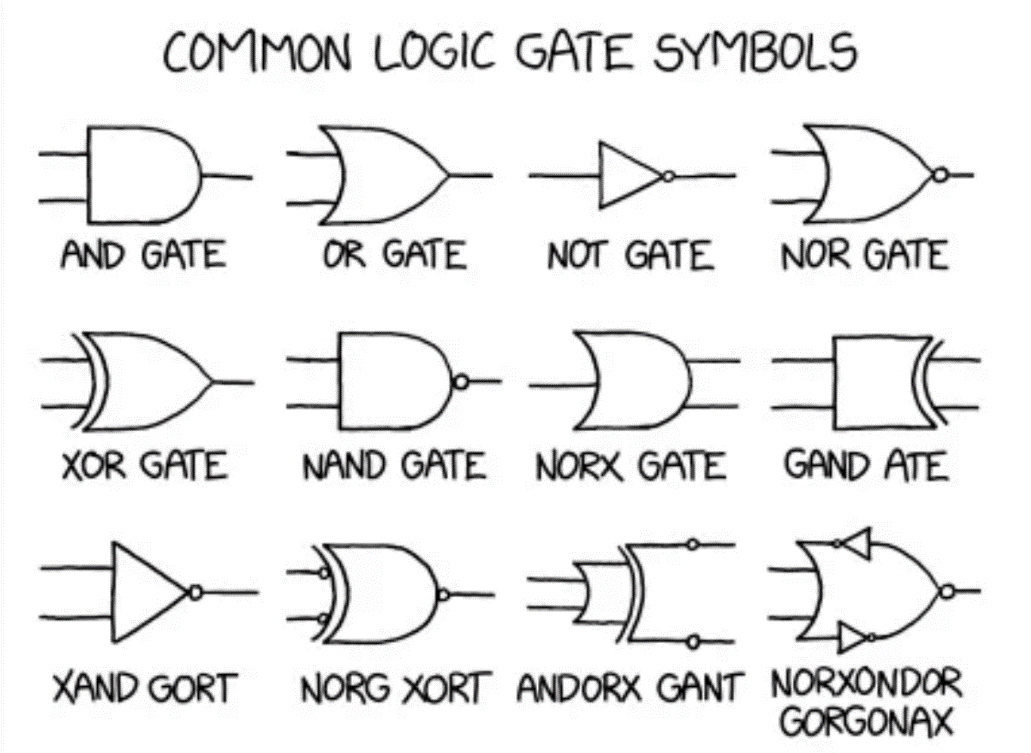
有了這些式子後，我們便可以用logic gate去實現它們，進而從output得知每個segment需要亮或不亮。由於前面使用k-map的時候，定義1表示要亮、0表示不亮，因此每個segment最後的output要再接一個NOT才是正確的值，整體的circuit如下。



1. **Summary**

這次的Lab實作量比較大，也比上一次的Lab需要思考，在這次的Lab中，我們對不同的adder有了更多的了解，並能夠利用基本的adder去接出可以執行較複雜運算的RCA、CLA及Multiplier，也了解到FA及HA的不同與應用；在Decode and execute中，我們運用universal logic gate組合出需要的logic gate，學習如何用logic gate設計出需要的運算方式，除此之外也用到了Mux去做select；在FPGA的部分也比上一次較為複雜，我們運用了在邏輯設計學過的k-map，將其實際應用在7-segment上；在設計testbench的部分，我們需要設計出正確的時間延遲，並測試自己的testbench是否能成功判斷Adder設計的對或錯，對testbench的操作也有更進一步的認識。

比起上次基本上只要做接線及組合logic gate的動做就能得出結果，這次我們需要自己依照題目要求去設計，logic gate的組合也更為複雜，所以多花了不少時間，但跟logic gate也有更緊密的接觸，如下圖。



1. **Contributions**

* **Code:**

**(Gate Level) 8-bit ripple carry adder (RCA)** by 李品萱

**(Gate Level) Decode and execute** by 李品萱

**(Gate Level) 8-bit carry-lookahead (CLA) Adder** by 唐翊雯

**(Gate Level) 4-bit multiplier** by 唐翊雯

**An exhaustive testbench design** by 唐翊雯

**FPGA: (Gate Level) Decode and execute** by李品萱

* **Report:**

兩人先描述各自在code部分負責的題目及畫電路圖，再由唐翊雯寫summary。