

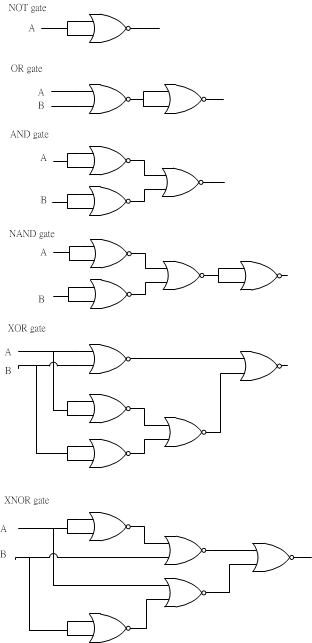
第七組

黎佑廷 105030009

郭家瑋 105030015

lab2

BASIC NOR UNIVERSAL GATE



Verilog Question 1:

(Gate Level) Binary code to Grey code

1. Dout[0] 的K-map

din[1], din[0]

din[3], din[2]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 00 | 01 | 11 | 10 |
| 00 | 0 | 1 | 0 | 1 |
| 01 | 0 | 1 | 0 | 1 |
| 11 | 0 | 1 | 0 | 1 |
| 10 | 0 | 1 | 0 | 1 |

1. Dout[1] 的K-map

din[3], din[2]

din[1], din[0]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 1 | 1 |
| 01 | 1 | 1 | 0 | 0 |
| 11 | 1 | 1 | 0 | 0 |
| 10 | 0 | 0 | 1 | 1 |

1. Dout[2] 的K-map

din[3], din[2]

din[1], din[0]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 0 | 0 |
| 01 | 1 | 1 | 1 | 1 |
| 11 | 0 | 0 | 0 | 0 |
| 10 | 1 | 1 | 1 | 1 |

1. Dout[3] 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 | 1 | 1 | 1 | 1 |
| 10 | 1 | 1 | 1 | 1 |

由K-map的結果可以發現:

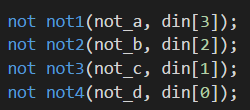
Dout[0] = ( not(Din[1]) & Din[0] ) | ( Din[1] & not(Din[0]) )

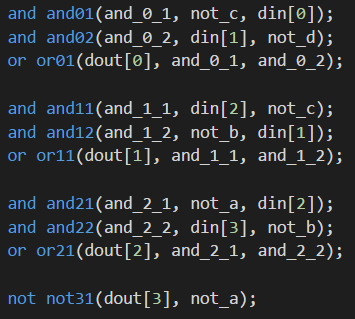
Dout[1] = ( not(Din[1]) & Din[2] ) | ( Din[1] & not(Din[2]) )

Dout[2] = ( not(Din[3]) & Din[2] ) | ( Din[3] & not(Din[2]) )

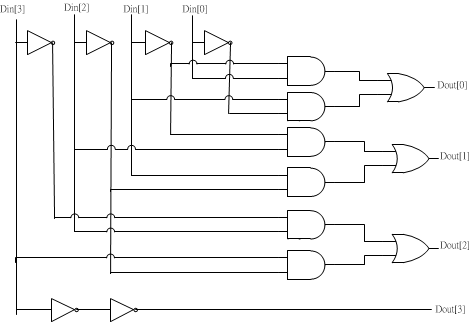
Dout[3] = Din[3]

它們所對應的code是這樣:





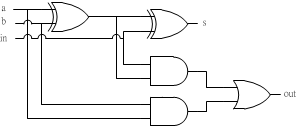
邏輯圖如下：



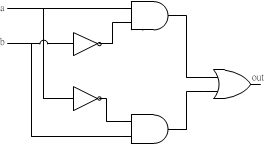
這題我所使用的驗證方法是將input din 從4’b0000 加到 4’b1111 以驗證所有的可能性。

Verilog Question 2

(Gate Level) Multiplier



FullAdder



XOR gate

以下FullAdder腳位對應為

b

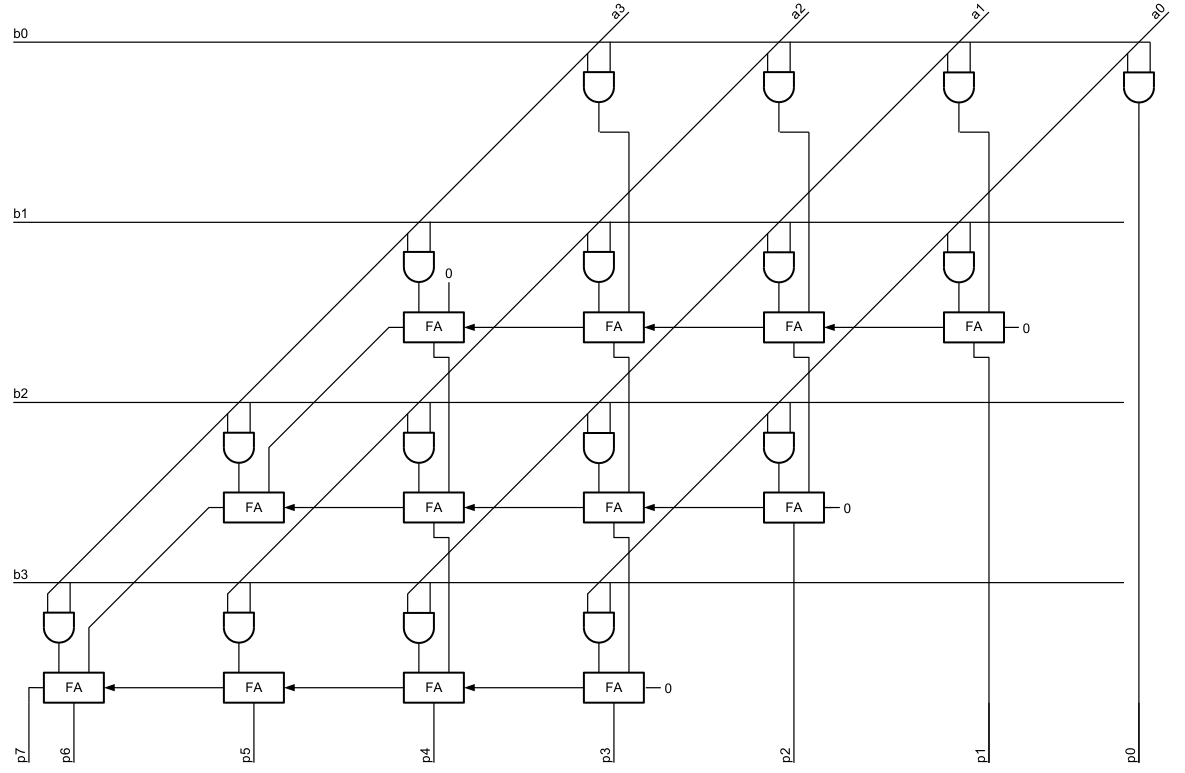
FA

a

in

s

out



(圖片來自W. J. Dally Digtal Design: A System Approach)

基本上，用gate-level寫出4\*4的乘法器是跟數學直式完全相同。

首先，將a[0]\*b[0]放在最右bit，並逐漸向左邊的bit移動，分別乘出a[1]\*b[0]、a[2]\*b[0]、a[3]\*b[0]。用a[0]乘完一輪b的各bit後，要換成a[1]乘b的各bit，此時如同直式乘法，要整列向左移一bit。每個bit相乘時，只有1\*1才會得到1，否則都是0，所以用AND-gate。

每個bit都乘完後，只需處理行的加法與進位問題。此時用上1 bit Full Adder。因為1 bit Full Adder的input只有a、b、cin，故一次只能處理兩個1 bit的相加。如圖上紅框處，若兩個AND-gate經第一個FA相加後，因為下方還有運算未結束，於是將sum作為下方FA的input b。持續相同動作，直到一整行完成運算，將sum輸出成p[3]。

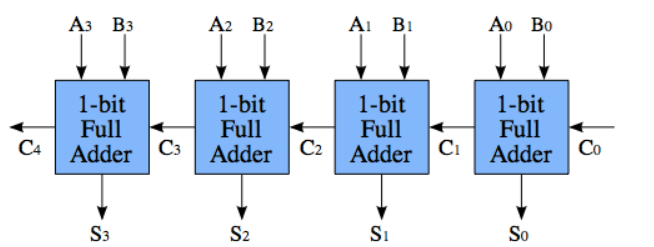
若某FA的input a、b、cin都為1，此時除了sum=1還會有cout=1，跟數學進位法相同，要將cout帶進左方bit的FA的cin。在最右方的FA因為不會有cin，所以直接用0代cin。

Testbench的寫法，是把全部的可能數字a、b都帶入乘法，驗證結果。

Verilog Question 3

(Gate Level) 4-bit Carry-Lookahead (CLA) Adder

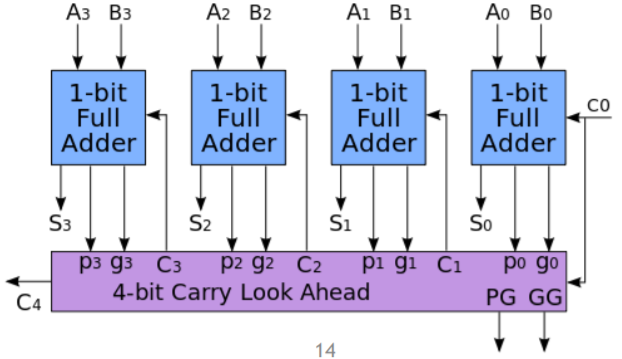
讓我們先從一般的ripple開始說起:



從右數來第二個fulladder必須等待第一個fulladder計算完並傳出C1後它才能正確計算出C2的值，

以此類推，C4要等到4倍的propagation delay才能被正確的計算，而carry-lookahead adder 解決了這樣的問題。

接下來我們來看看carry-lookahead adder



其中Si=Ai⊕Bi⊕Ci

https://gss0.bdstatic.com/-4o3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D294/sign=66928dd719178a82ca3c78a9c202737f/8b13632762d0f703753be2c70ffa513d2697c524.jpg  = Ai \*Bi+（Ai⊕Bi）\* C i

Gi = Ai \* Bi  (generate)

Pi = Ai⊕Bi (propagate)

所以Ci+1= Gi+ Pi \*Ci ，也就是說:

C1 = G0 + P0 \* C0 ，

C2 = G1 + P1 \* C1 ，

C3 = G2 + P2 \* C2 ，

C4 = G3 + P3 \* C3 ，

經過一些替代後，我們得到:

C0 = Cin，

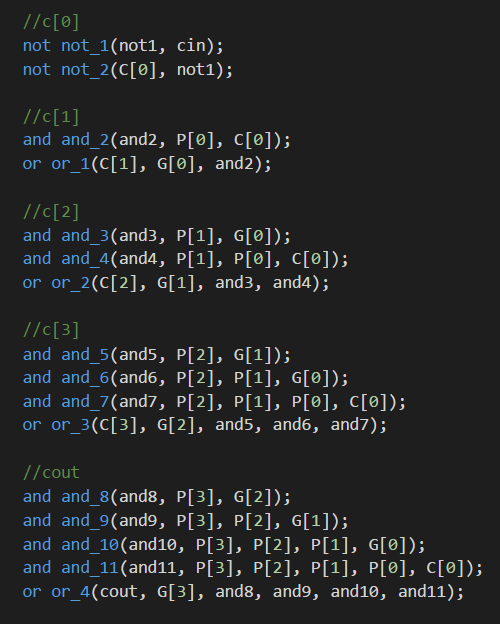
C1 = G0 + P0 \* C0 ，

C2 = G1 + G0 \* P1 + C0 \* P0  \* P1，

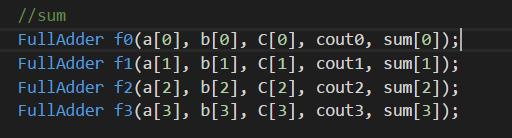
C3 = G2 + G1 \* P2 + G0 \* P1 \* P2 + C0 \* P0  \* P1 \* P2，

C4 = G3 + G2 \* P3 + G1 \* P2 \* P3 + G0 \* P1 \* P2 \* P3 + C0 \* P0  \* P1 \* P2 \* P3 = Cout，

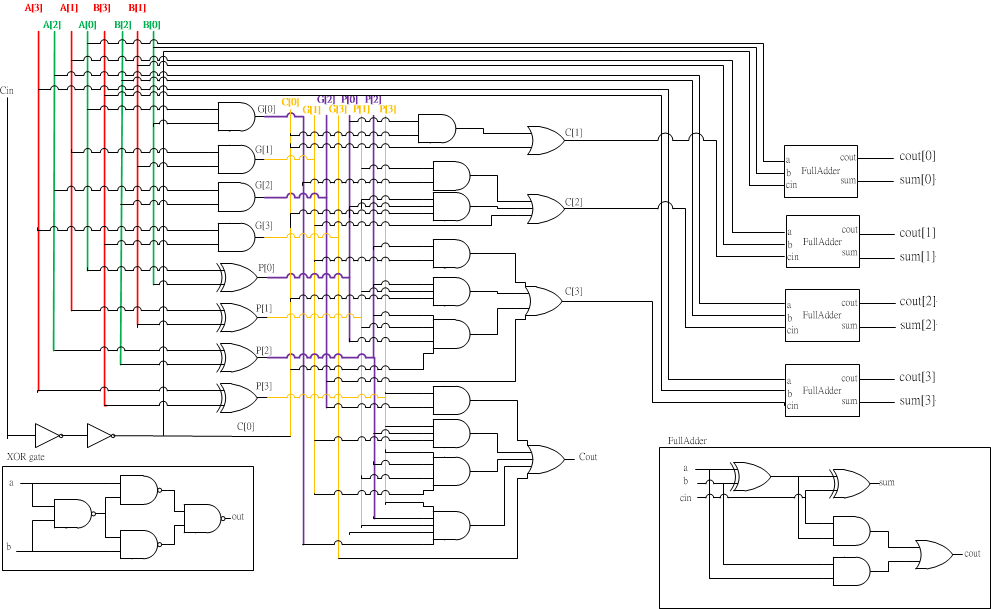
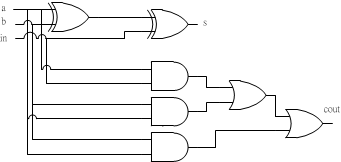
C0, C1, C2, C3, C4(cout) 所對應的code如下:



Sum的部分應slide得要求，我使用了lab1的fulladder module 來算出它，code如下:



邏輯圖如下：

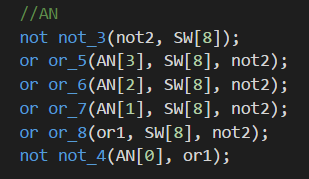


從這邊我們可以看出，每個位數所產生的進位都是獨立生成的，跟前面一個位元的運算沒有關聯，只和input以及Cin相關，因為這樣它減少了一般ripple carry adder進位所產生的延遲。

這題我所使用的驗證方法是賦予input a, b, cin不同的值來進行確認，但因為a跟b都有16種可能性, cin 有兩種可能性，所以有512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以及cin數值的方式，進行抽驗。

FPGA的部分:

我將AN[3] 、AN[2]、AN[1] 設成 SW[8] + not (SW[8]) 這樣一來不管 SW[8] 是多少它們都會等於1



至於seg我是用Decoder的方式將它從sum中decode出來，如表格:

|  |  |
| --- | --- |
| Sum[3:0] | Seg[0:6] |
| 0000 | 0000001 |
| 0001 | 1001111 |
| 0010 | 0010010 |
| 0011 | 0000110 |
| 0100 | 1001100 |
| 0101 | 0100100 |
| 0110 | 0100000 |
| 0111 | 0001111 |
| 1000 | 0000000 |
| 1001 | 0000100 |
| 1010 | 0001000 |
| 1011 | 1100000 |
| 1100 | 0110001 |
| 1101 | 1000010 |
| 1110 | 0110000 |
| 1111 | 0111000 |

畫完k-map後可以知道

Seg[0] = 

Seg[1] = 

Seg[2] = 

Seg[3] = 

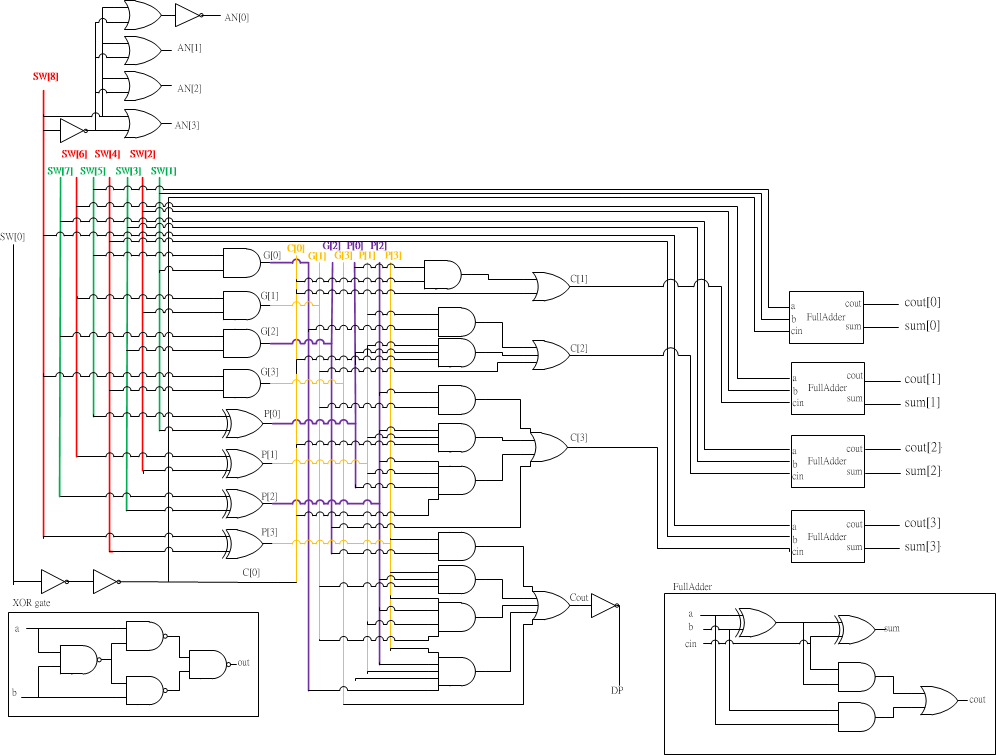
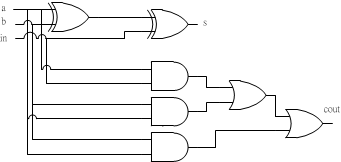
Seg[4] = 

Seg[5] = 

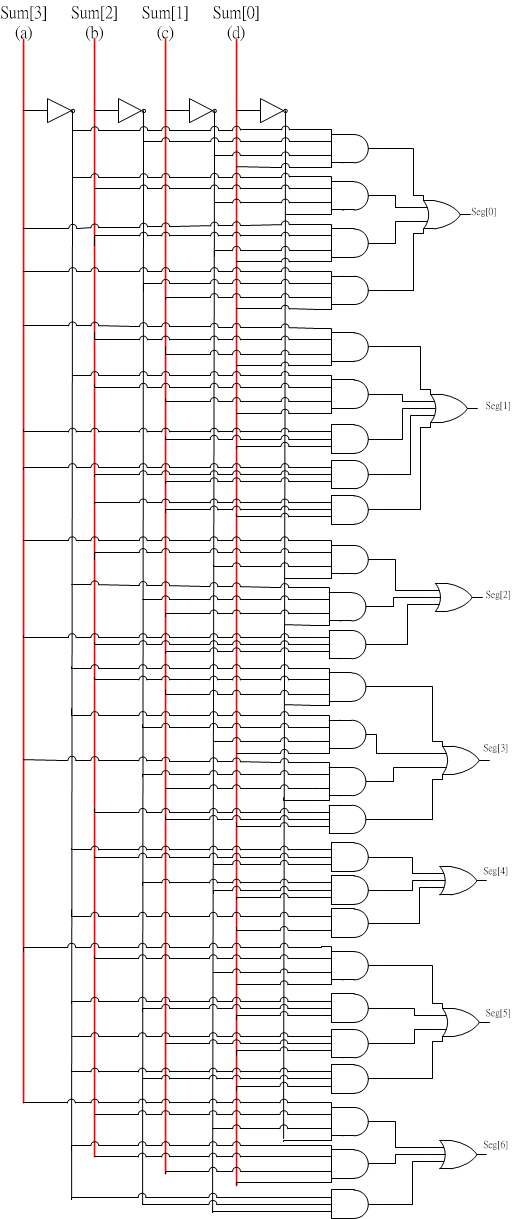
Seg[6] = 

DP則是Cout經過not運算的結果。

FPGA實作的邏輯圖與上圖類似，不過將input 的A、B換成SW。並且主要是多處理了七段顯示器的decode。邏輯圖如下：



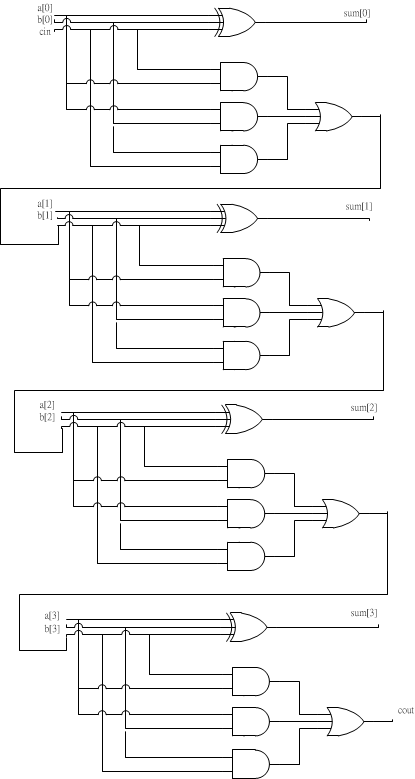
(篇幅原因，下圖接續上圖，處理SEG部分)



Verilog Question 4

(Gate-level) 4-bit ripple-carry adder (RCA)

**ADD MODULE**



**FA**

**FA**

**FA**

b

FA

a

in

s

out

**FA**

ADD是由四個FullAdder串接，a[0]和b[0]當a、b，輸出sum[0]與cout，並且將cout連接至下一個FullAdder (a、b input是a[1]和b[1]) 的cin。因為a、b input是a[0]和b[0]的FullAdder沒有cin故代0。

**SUB MODULE**

SUB的作法與ADD類似。a–b=a+(-b)，所以減法可以看成加一個負數。先把b用NOT-gate做二補數，再加1’b1即是-b。因此把b[0]、b[1]、b[2]、b[3]做not，放在上圖中b[0]、b[1]、b[2]、b[3]位置，再將FA的LSB的cin代入1(作加法時代入0)，代表加1’b1，即是答案。

**INC MODULE**

INC的作法與ADD類似。只需把b設定成4’b0001，並且LSB的FA的cin還是維持代0，即是答案。

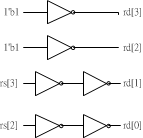
**BITWISE NOR MODULE**

BITWISE NOR的作法即是把各個bit分開用NOR-gate做。

**BITWISE NAND MODULE**

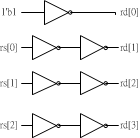
BITWISE NAND的作法即是把各個bit分開用NAND-gate做。

**RS DIV 4 MODULE**

****

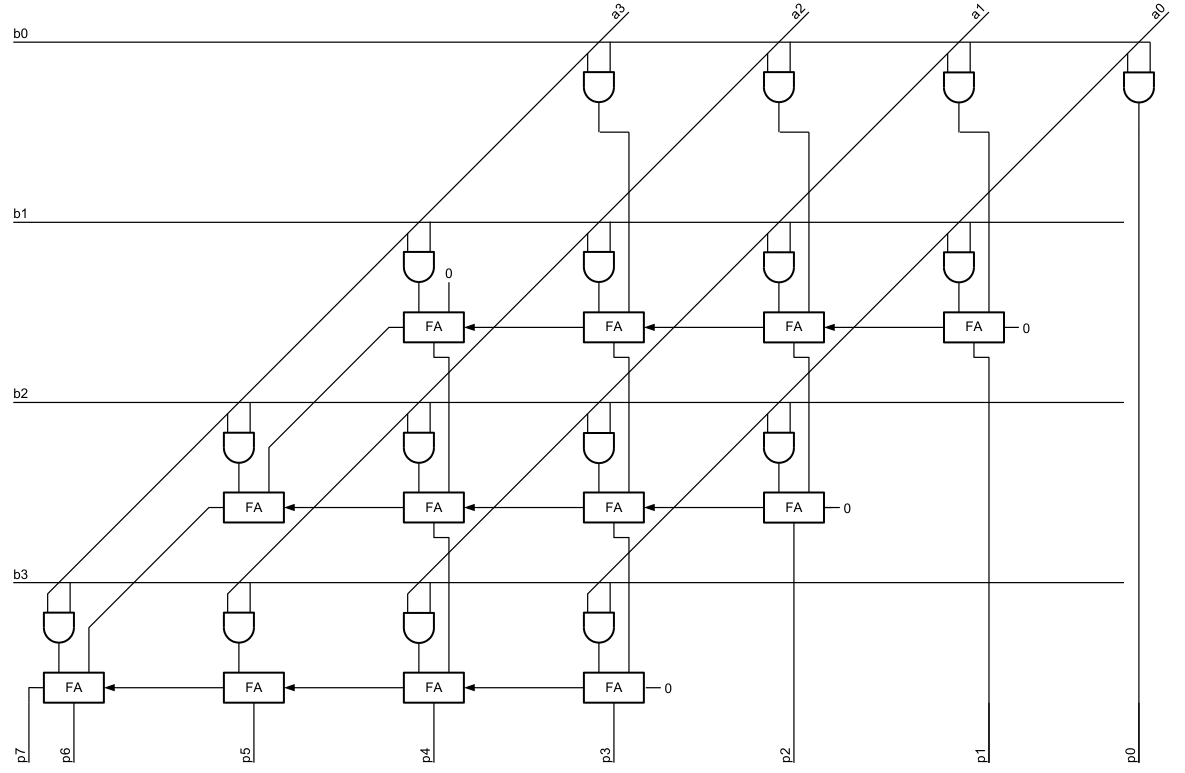
將RS除以4即是在二進位中，將RS的各個bit右移兩個bit，並且因為題目中所說為〝>>〞，代表用logic shift就好，所以在右移兩bit後，在最大的兩個bit直接補0。

**RS\_MUL\_2 MODULE**



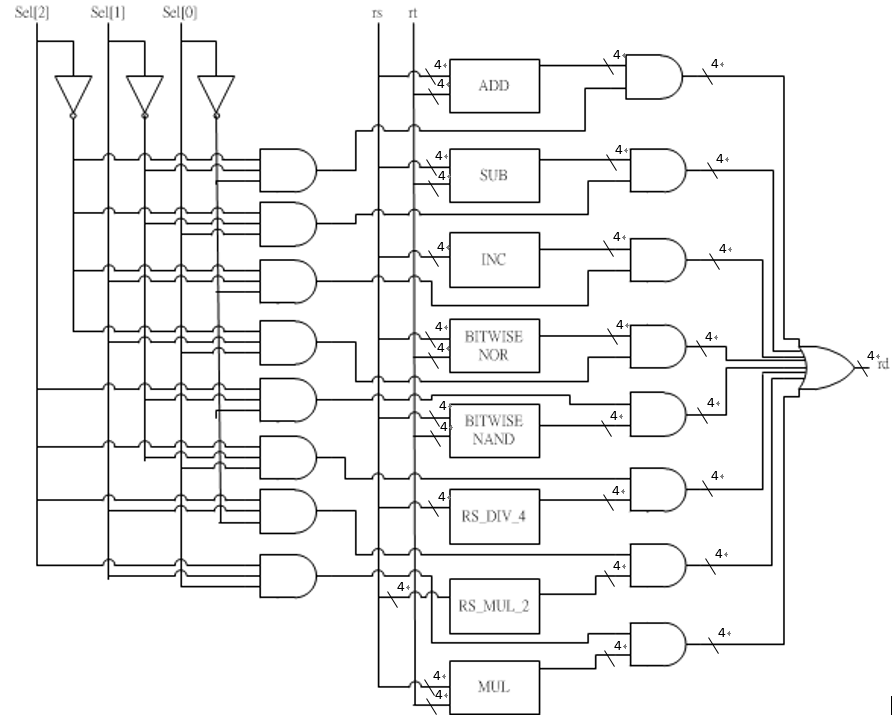
將RS乘以2即是在二進位中，將RS的各個bit左移一個bit，並且因為題目中所說為〝<<〞，代表用logic shift就好(雖然左移時的logic shift和asithmetic shift相同)。

**Multiplier MODULE**



這與進階題第二題的multiplier完全相同，邏輯圖與說明見第二題，故不贅述。

**Putting the modules altogether**

****

此題的testbench寫法是將所有可能的數值代入input，並檢查output。

**心得(BY 郭家偉)：**

這次lab我覺得比較有挑戰性與趣味的是Multiplier。如果用behavior level寫起來可能會很簡單，但用gate level寫起來就比較考驗邏輯了。不過用gate level寫起來也比較清楚程式實際的運作方式，而不是靠軟體去synthesis，也比較不會出問題，但是寫起來也相當費時費力。

**心得(BY黎佑廷)：**

我覺得這次lab第一題不難，只要畫畫k-map答案就呼之欲出了，必較麻煩的是第三題，在搞懂carry lookahead adder的過程就花了一些時間，至於fpga的部分，因為是第一次使用7-Segment Display，所以也花了一些時間摸索。

**成員:**

黎佑廷: 負責第1、3題以及整理report

郭家瑋: 負責第2、4題以及畫出gate level circuit