**Lab01 (109061256 陳立萍)**

1. **lab1\_1: a BCD-to-Excess-3 code converter**

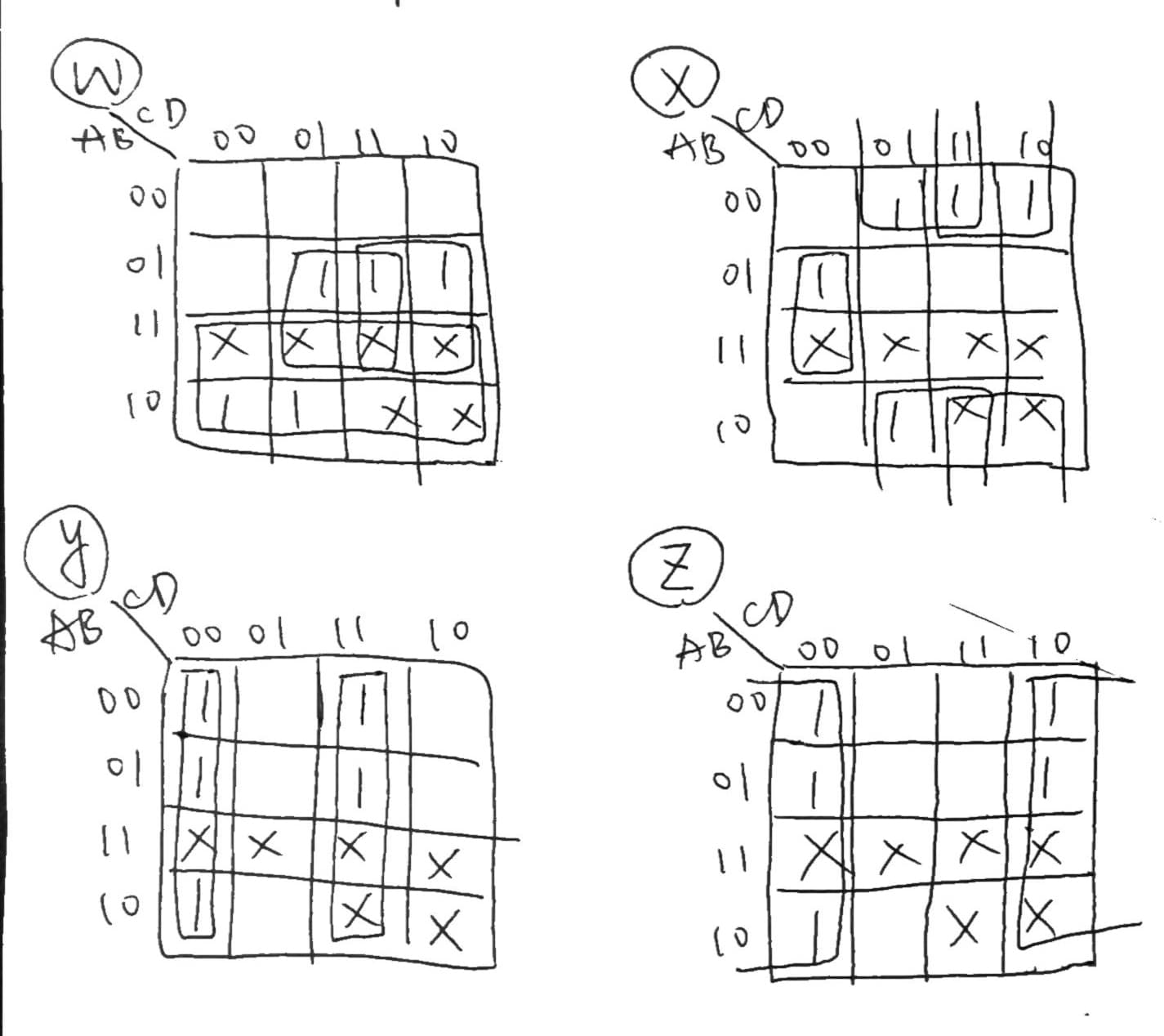
**Design Specification**

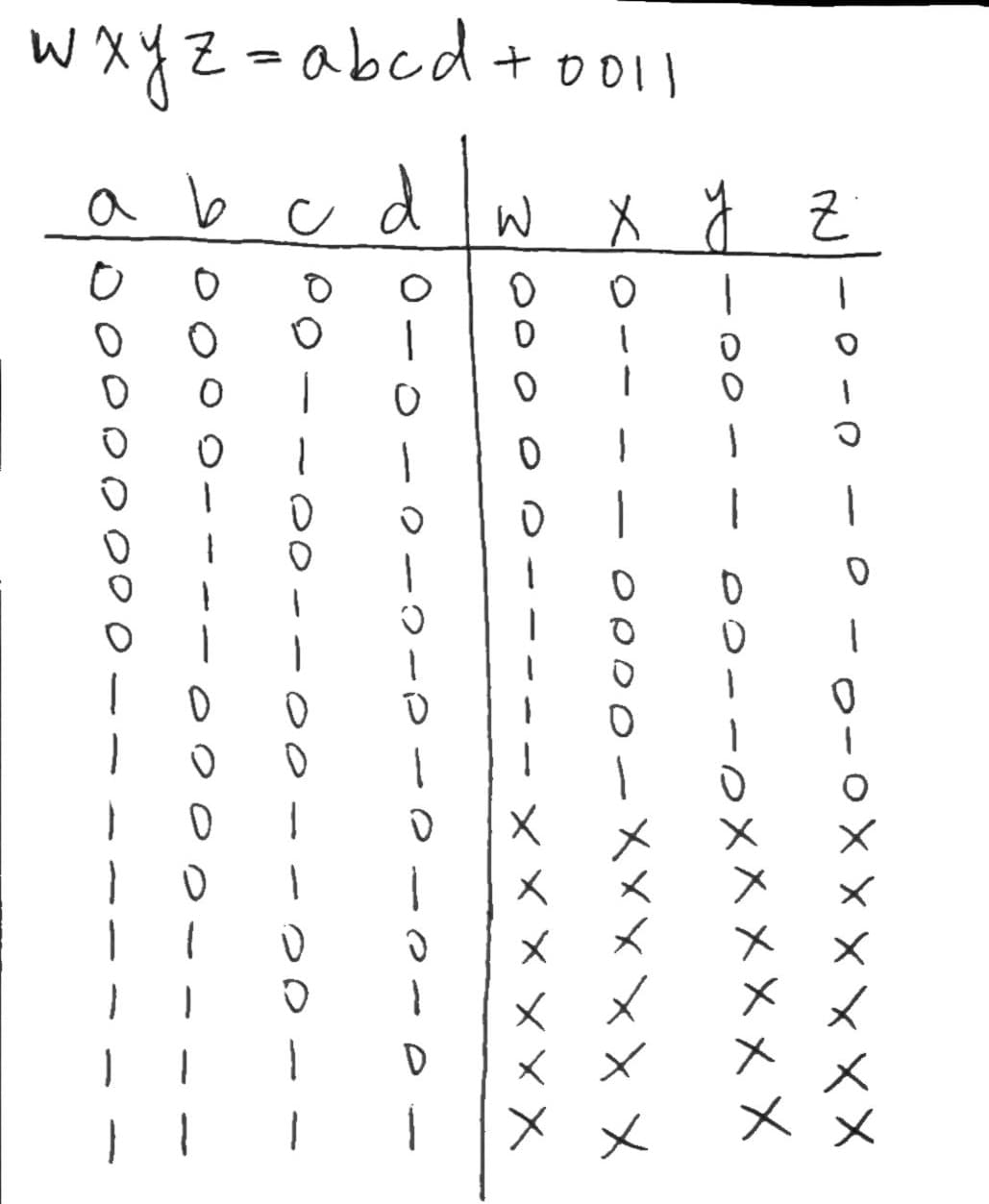
input: abcd(MSB to LSB)

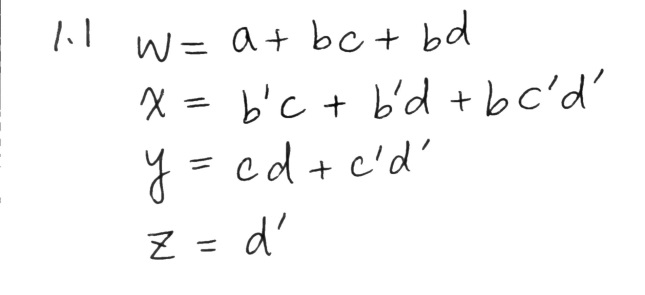
output: wxyz(MSB to LSB)

abcd: 0000 ~ 1001

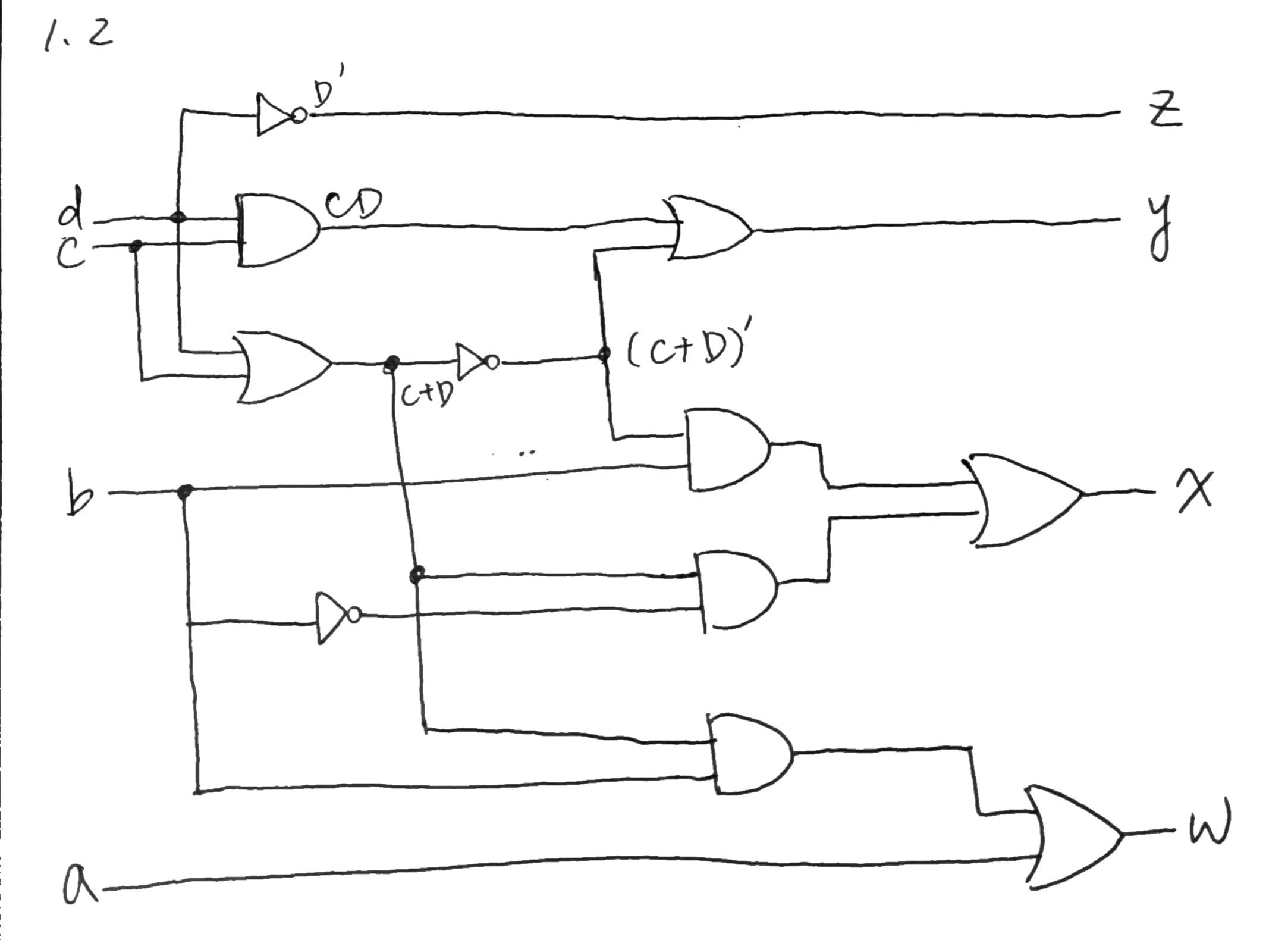
1.1 Write the Boolean function/logic equation.







1.2 logic diagram



**Design Implementation**

* 1. wave diagram



**設計方法：**

BCD轉換成Excess-3的式子如上1.1 ，wxyz = abcd + 0011，接著寫出對應的truth table，再用k-map化簡output (wxyz)，得到wxyz各自的Boolean function.

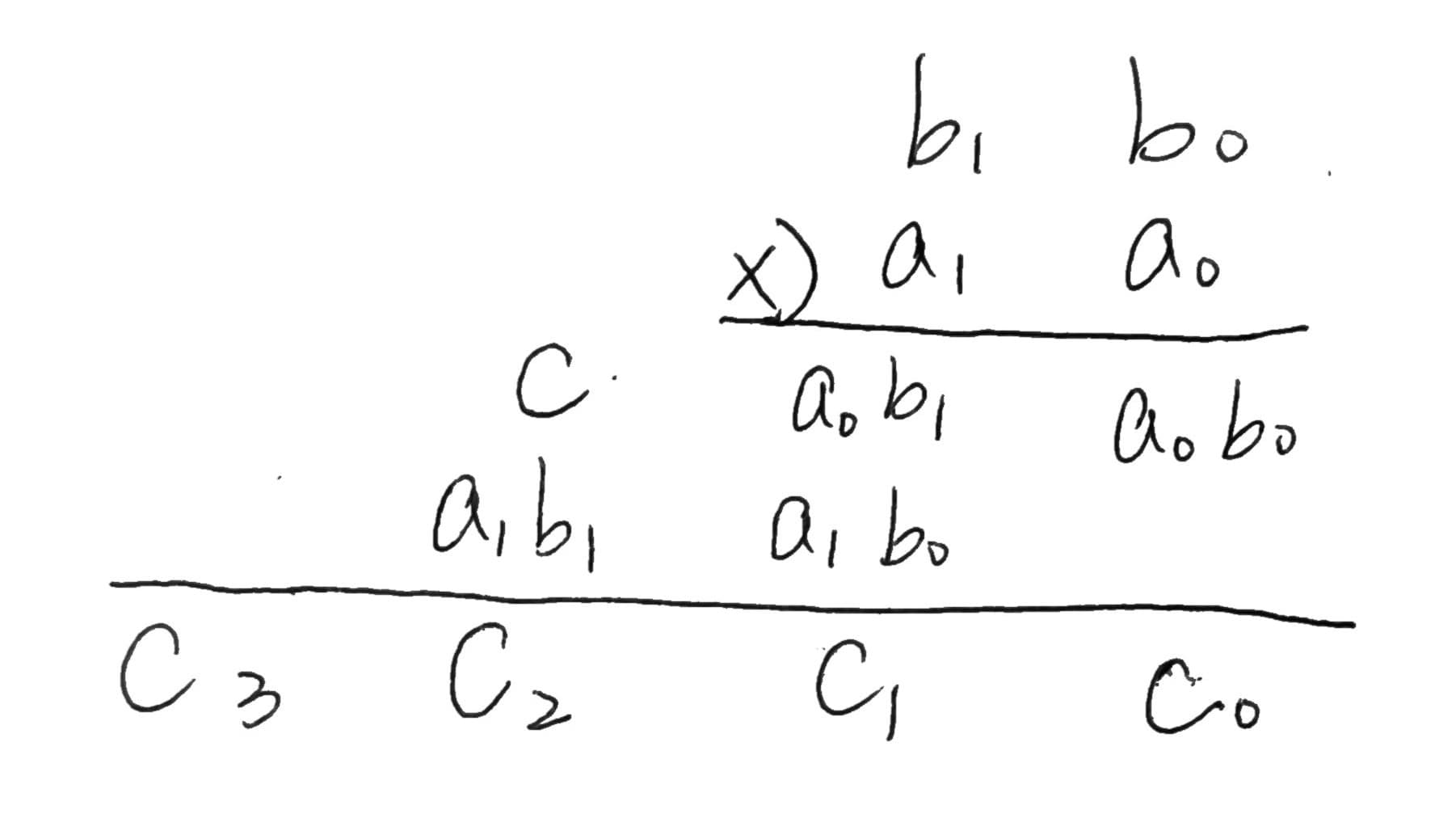
1. **lab1\_2: an unsigned 2-bit x 2-bit binary multiplier**

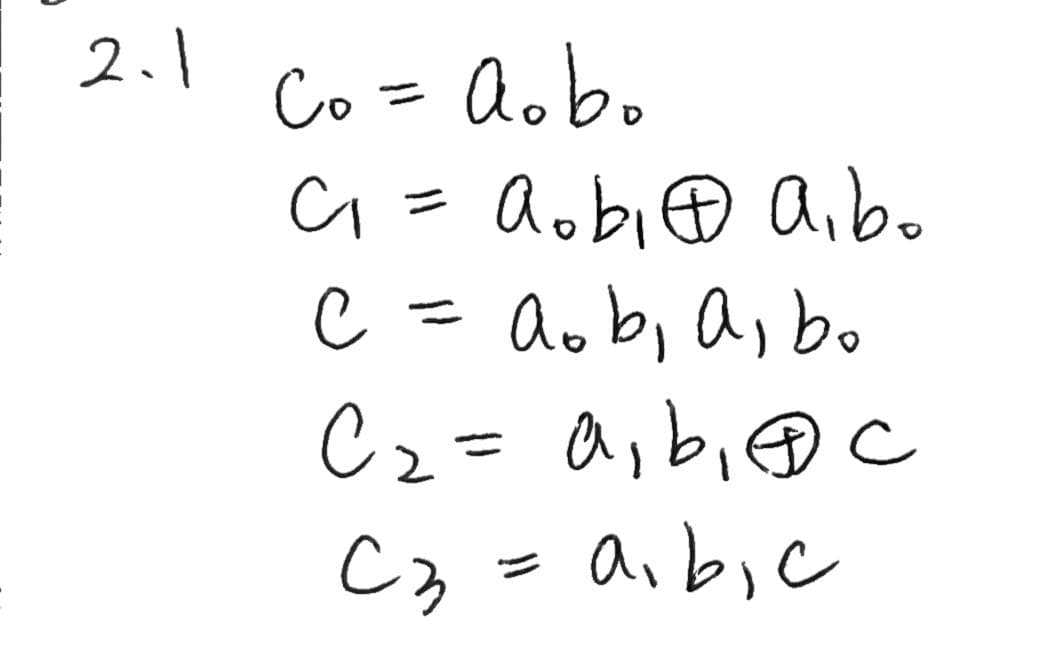
**Design Specification**

input: a1, a0, b1, b0

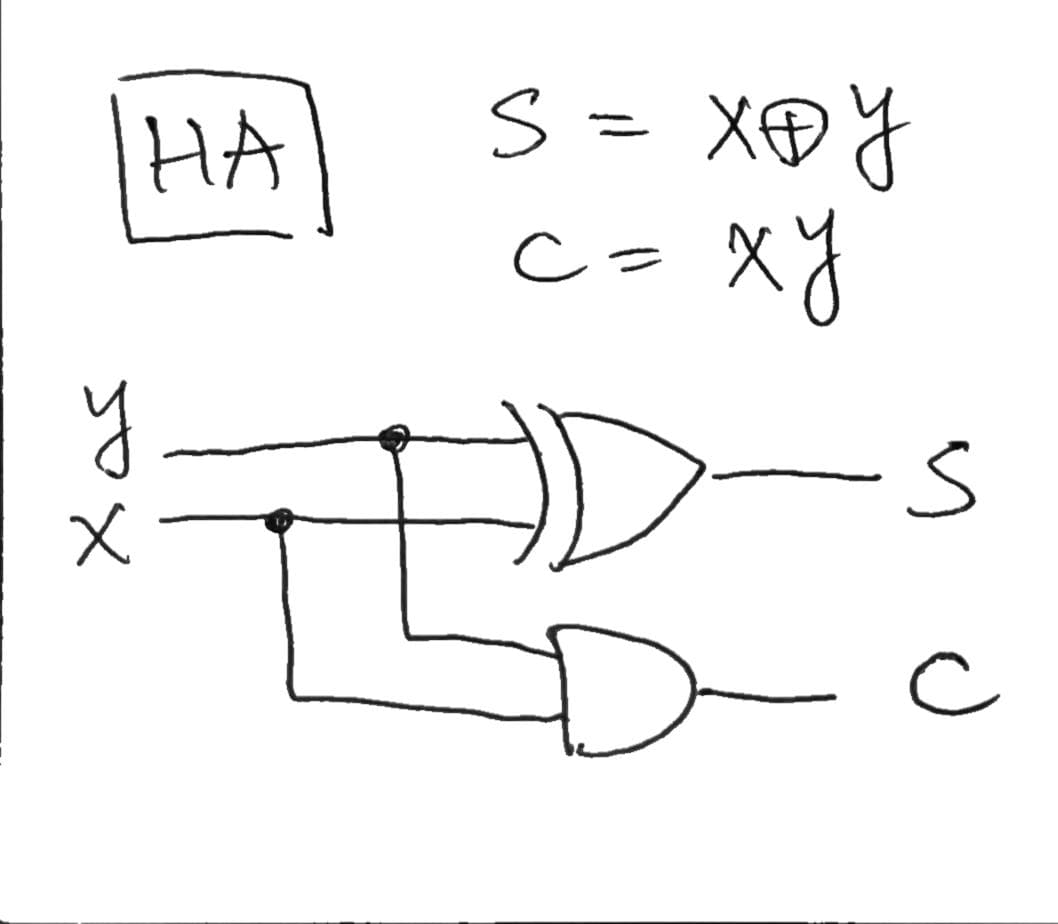
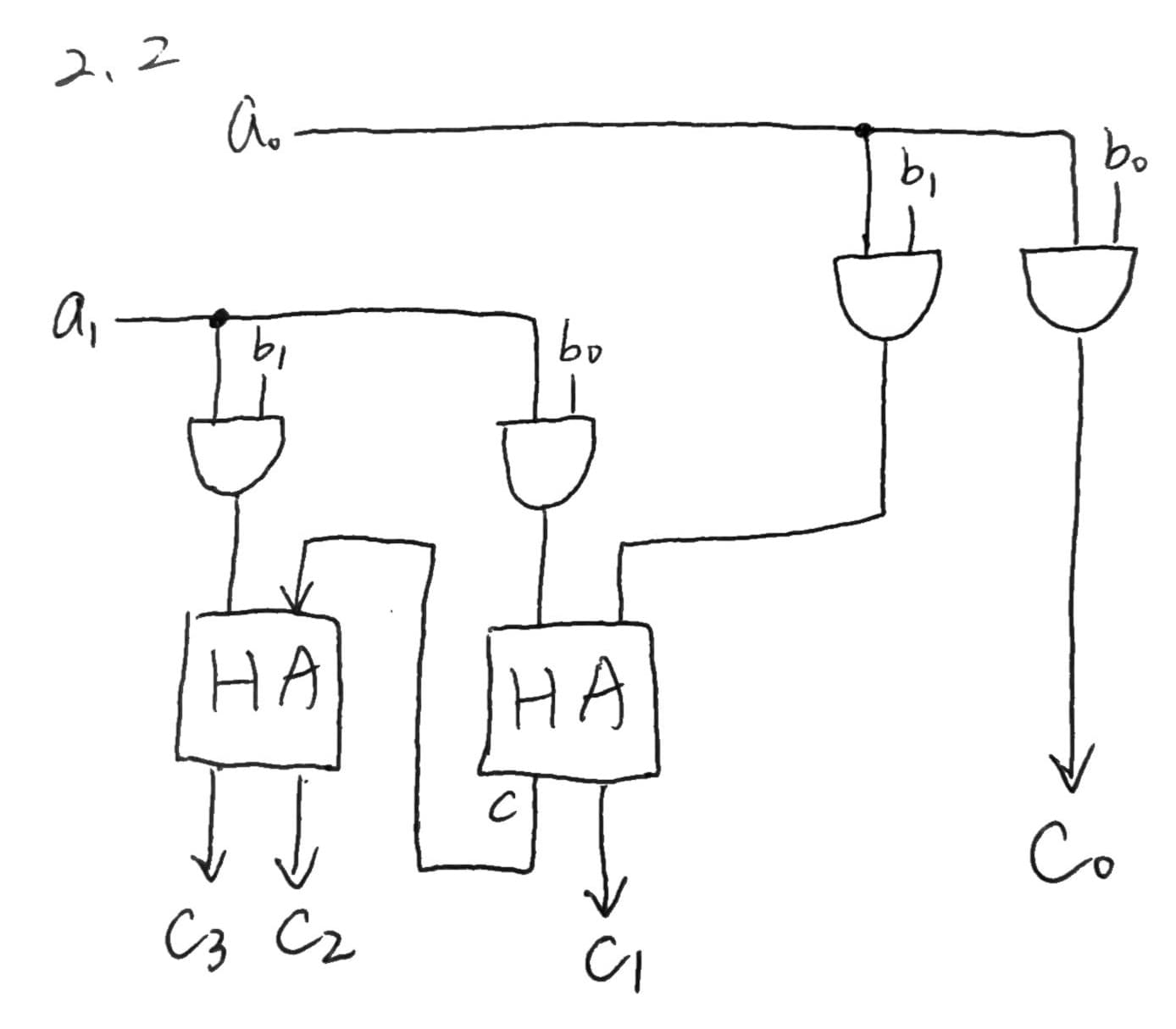
output: c3, c2, c1, c0, c

2.1





2.2 logic diagram



**Design Implementation**

2.3 wave diagram



**設計方法：**

經由乘法直式計算（在2.1）可以得到input與output之間的關係，而其中c為a0b1與a1b0做加法後所得之進位，而其中的加法運算利用half adder作加法，得到的每一個output cx即為每次加法所的到的S，c0為最低位，c3為最高位。

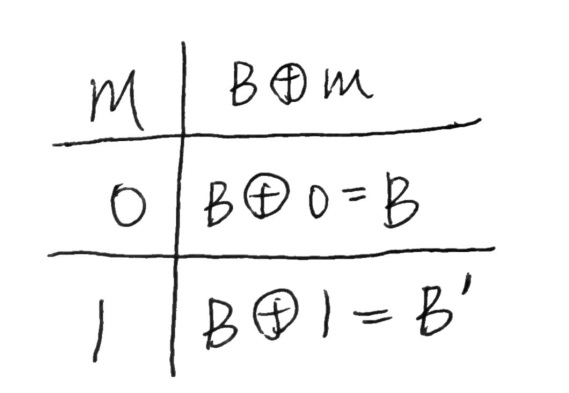
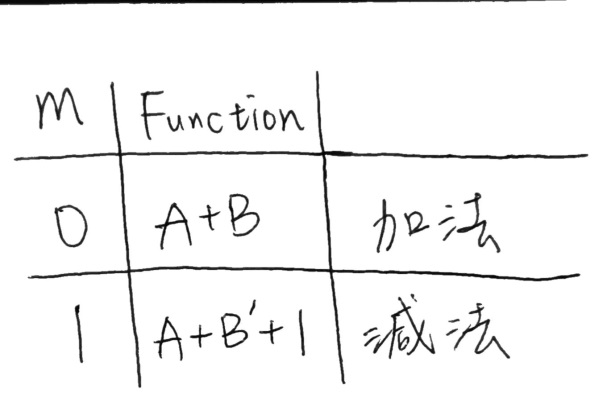
1. **lab1\_3: a 3-bit binary adder/subtractor**

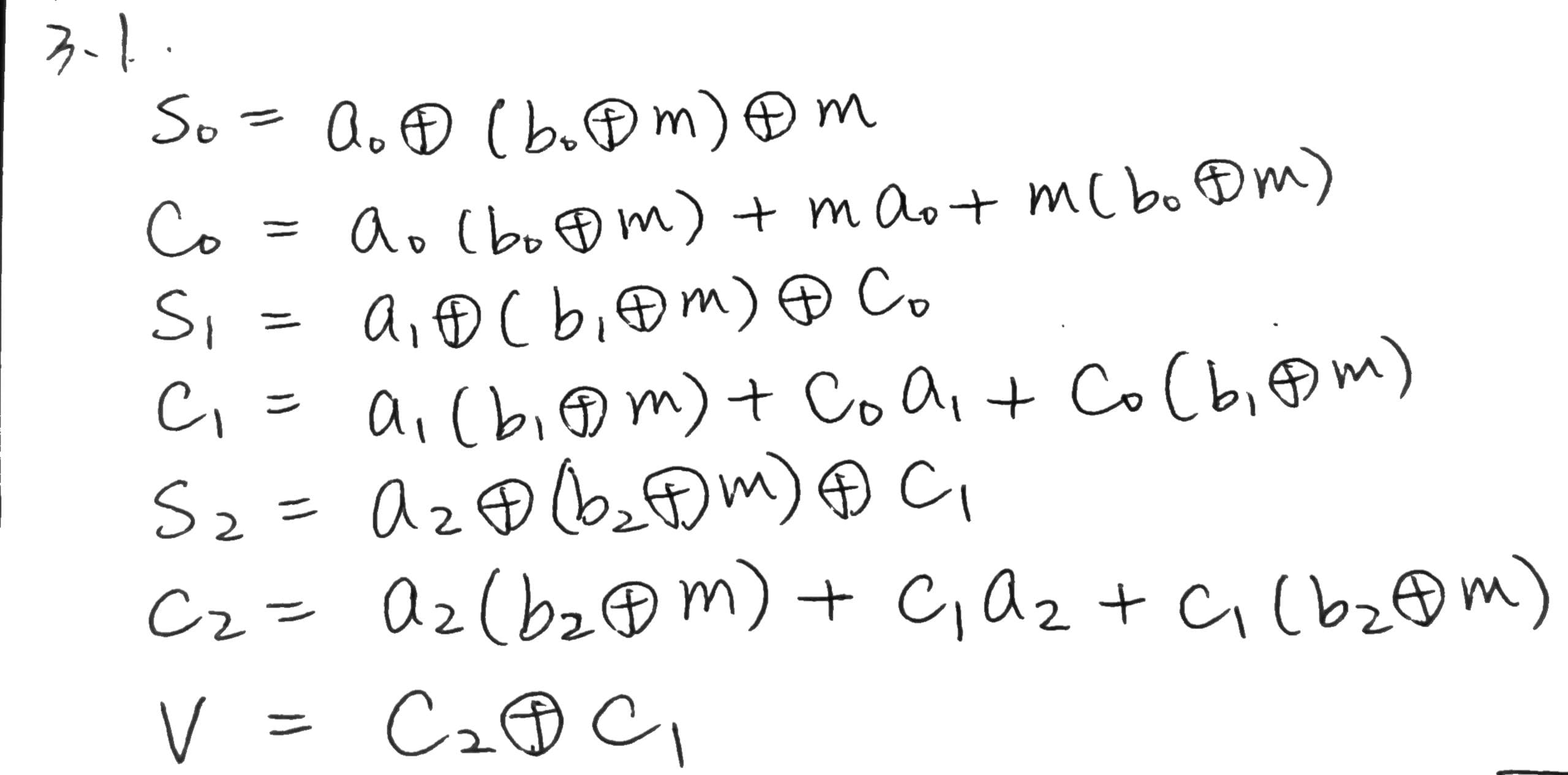
**Design Specification**

input: a[2:0], b[2:0], m

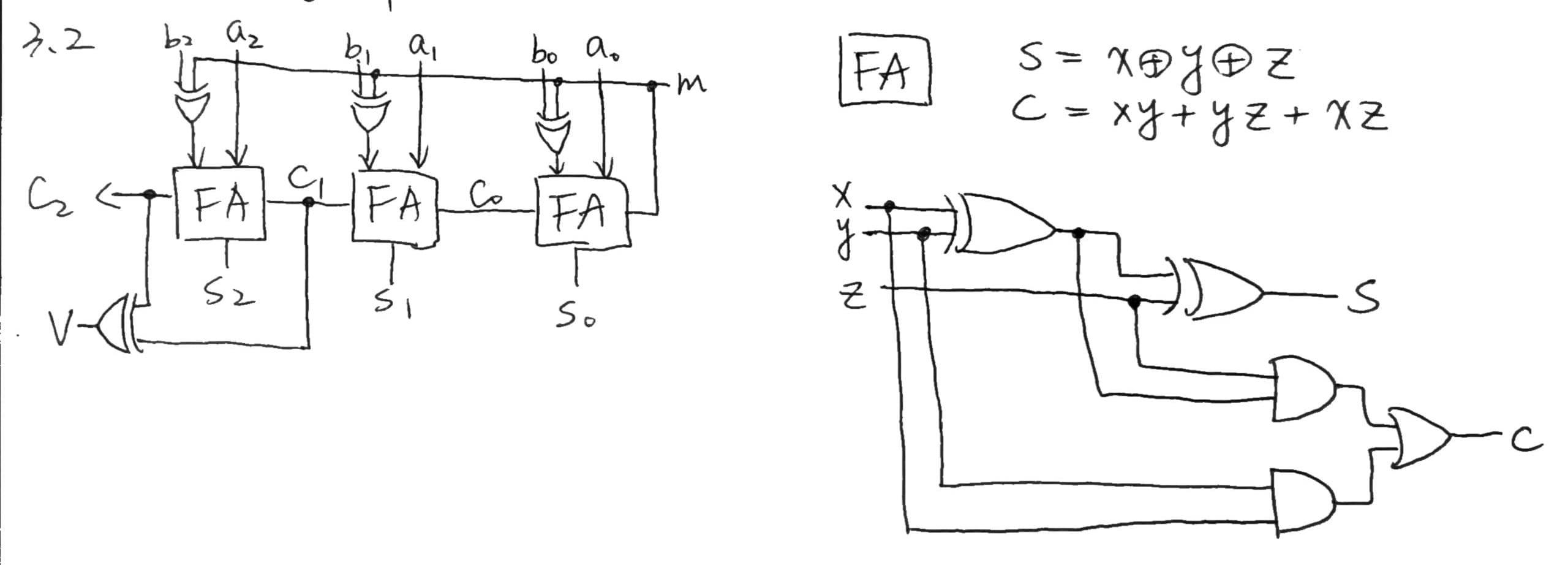
output: s[2:0], c[2:0], v

3.1



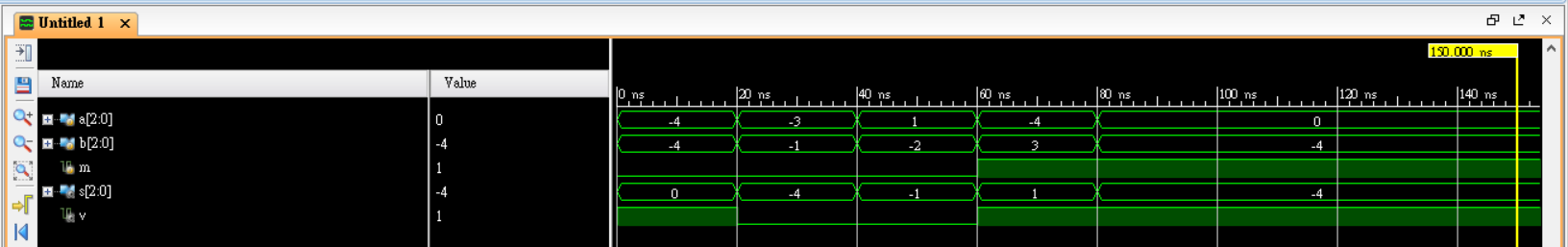


3.2 logic diagram



**Design Implementation**

3.3 wave diagram



**設計方法：**

使用exclusive or作加數與補數的轉換，如3.1表格所示，在每次作運算前，都先將bx的與m作exclusive or的動作，使其化為加數或是補數，再與a的每一個位元用full adder相加，每一級加法所產生的s即為答案，而c用來記住每一級加法所產生的carry，另外，v為overflow indicator為(c2 exclusive c1)

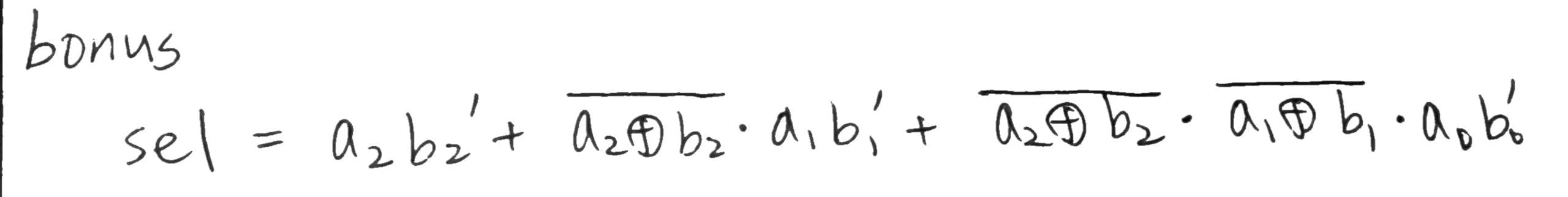
**Bonus. Lab1\_bonus:** for two 3-bit unsigned numbers a(a2a1a0) and

b(b2b1b0), build a logic circuit to output o as the smaller number.

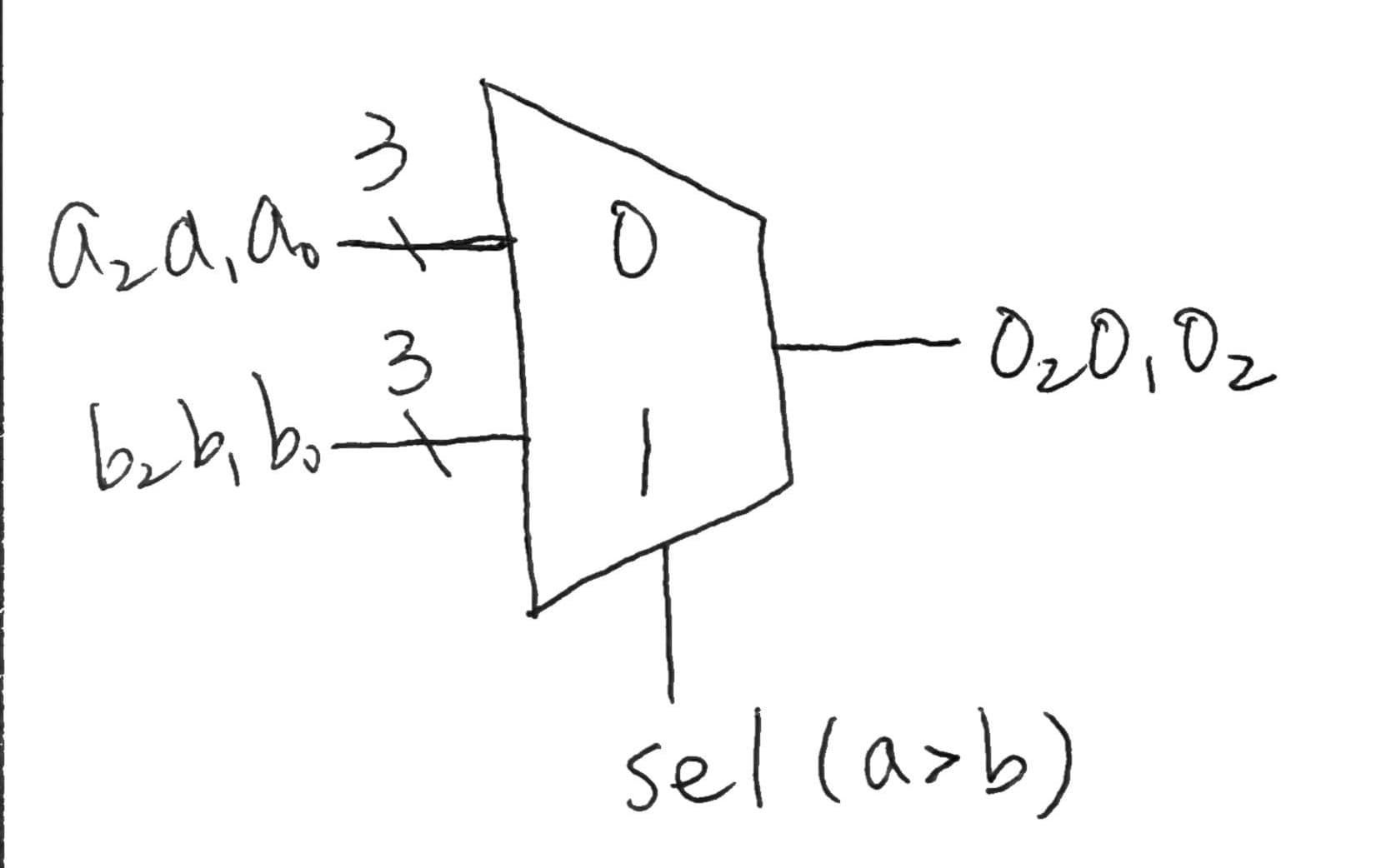
**Design Specification**

input: a2, a1, a0, b2, b1, b0

output: o2, o1, o0, sel

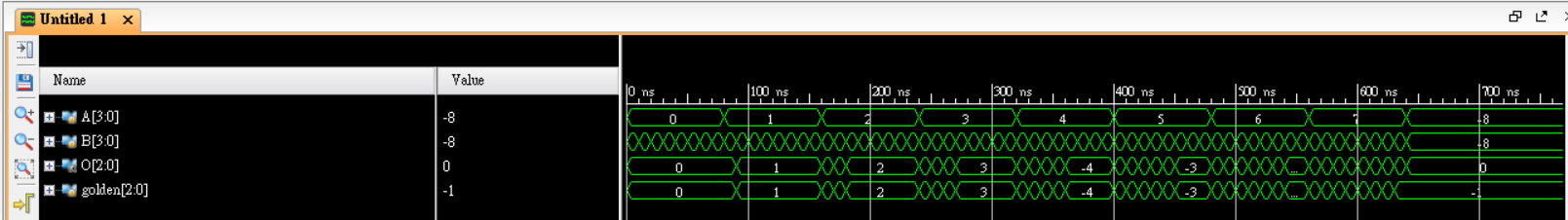


logic diagram



**Design Implementation**

wave diagram



**設計方法：**

狀況一：a2 = 1, b2 = 0

如果a>b要為true，則可得boolean function: (a2 & (~b2))

狀況二：a2 = b2, a1 = 1, b1 = 0

如果a>b要為true，則可得boolean function: ((~(a2 ^ b2)) & a1 & (~b1))

狀況三：a2 = b2, a1 = b1, a0 = 1, b0 = 0

如果a>b要為true，則可得boolean function: ((~(a2 ^ b2)) & (~(a1 ^ b1)) & a0 & (~b0))

綜合以上三點，將三狀況OR在一起，即可表達所有a>b的狀況，故表達a > b (a2, a1, a0, b2, b1, b0)的Boolean function如下: (a2 & (~b2)) | ((~(a2 ^ b2)) & a1 & (~b1)) | ((~(a2 ^ b2)) & (~(a1 ^ b1)) & a0 & (~b0))。

再用MUX以sel做選擇，如果sel為1，亦即a>b為true，那麼b會被選出並assign給o，若sel為0，亦即a>b為false，那麼a會被選出並assign給o。

**Discussion**

**Lab1\_1**

這個題目非常簡單，就是寫出每一個bit的boolean function再用k-map化簡即可得到簡化之答案，再一一assign給output即可。

**Lab1\_2**

這題在將乘法直式寫出來後，就非常清楚該如何寫出boolean function，而結果也與預期的相同。

**Lab1\_3**

在做這題時，起初忘記b的每一個bit要exclusive or m，因此導致輸出結果一直無法正確，除錯除了好久才發現是這個原因。

在Simulation的波形圖上可以調數值的顯示要是signed 或是unsigned ，而電腦預設為unsigned，導致一開始跑的時候，第一組數值是A=4,B=4,S=0

但是testbench上給的數值是A=-4,B=-4,S=0，一開始一直以為是程式寫錯，後來才發現旁邊可以調數值的有號與無號。

調整完後的輸出便與testbench給的值相符。

**Lab1\_bonus**

做這題的時候原先我用if else的語法一一比較a與b的每一個bit，但是在assign給o的每一個bit的時候，卻無法將單一一個ax或是bx assign給ox（ox宣告為output型態），因為宣告型態的問題，所以後來就改了方法，用mux下去寫，然而整個程式變得簡短許多，跑出來的波形圖也為正確答案。

**Conclusion**

經過lab01後上學期忘記的東西大部分都回來了。不過光是lab01就交了很多新的語法，像是always與 if else 和case等語法，都還在可以理解的範圍。希望之後可以學到更多不一樣的，尤其是一些設計想法，也希望自己可以理解這些新的知識。

**References**

教授授課頭影片：語法運用，符號運用。