**Lab1 Report**



**Ⅰ. Lab1\_1 (BCD to excess 3 code converter)**

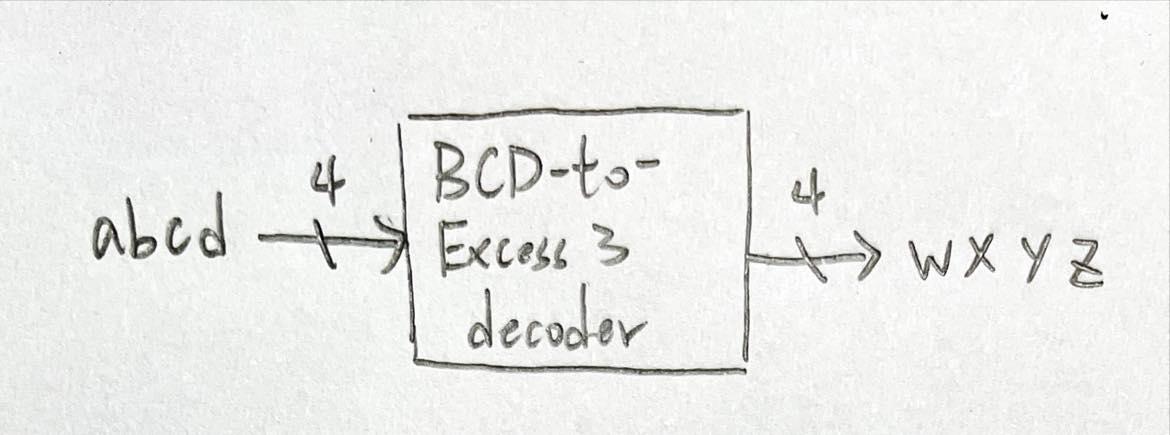
**Design Specification**

IO:

Input: a,b,c,d.

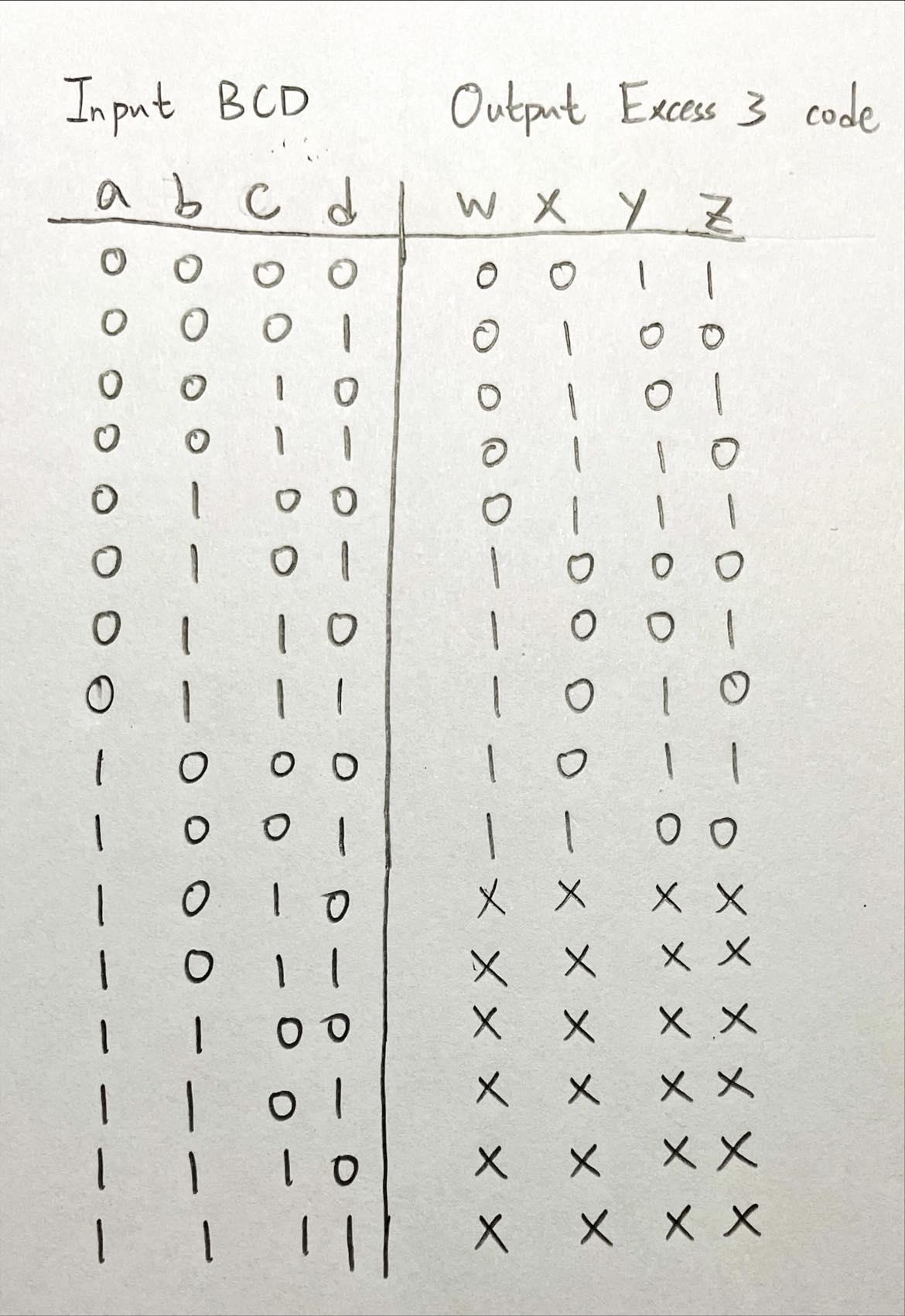
Output: w,x,y,z.

Block diagram:

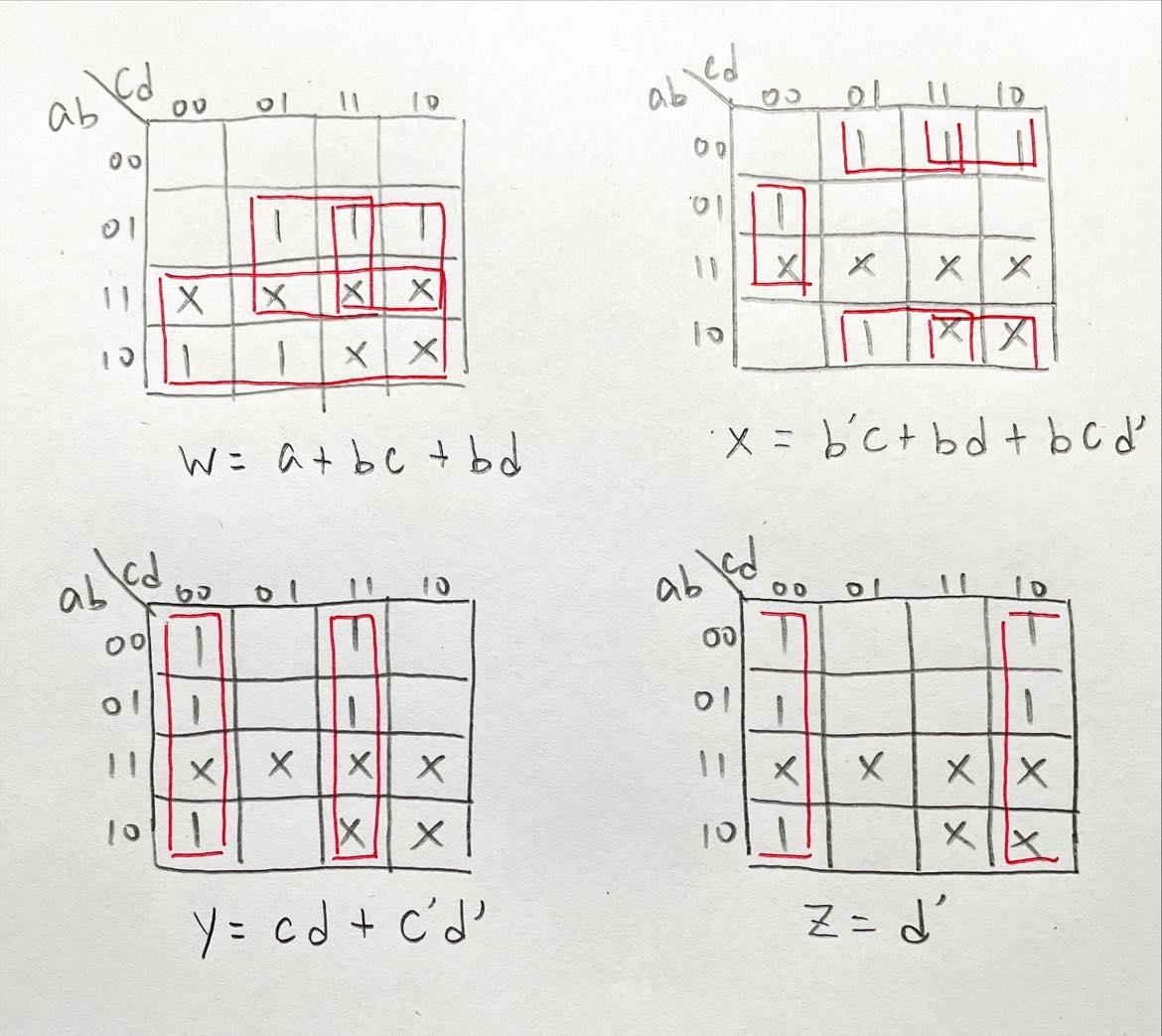


**Design Implementation**

First, draw the truth table of converter as follows:



Then, use K-map to derive logic function:



Boolean function:

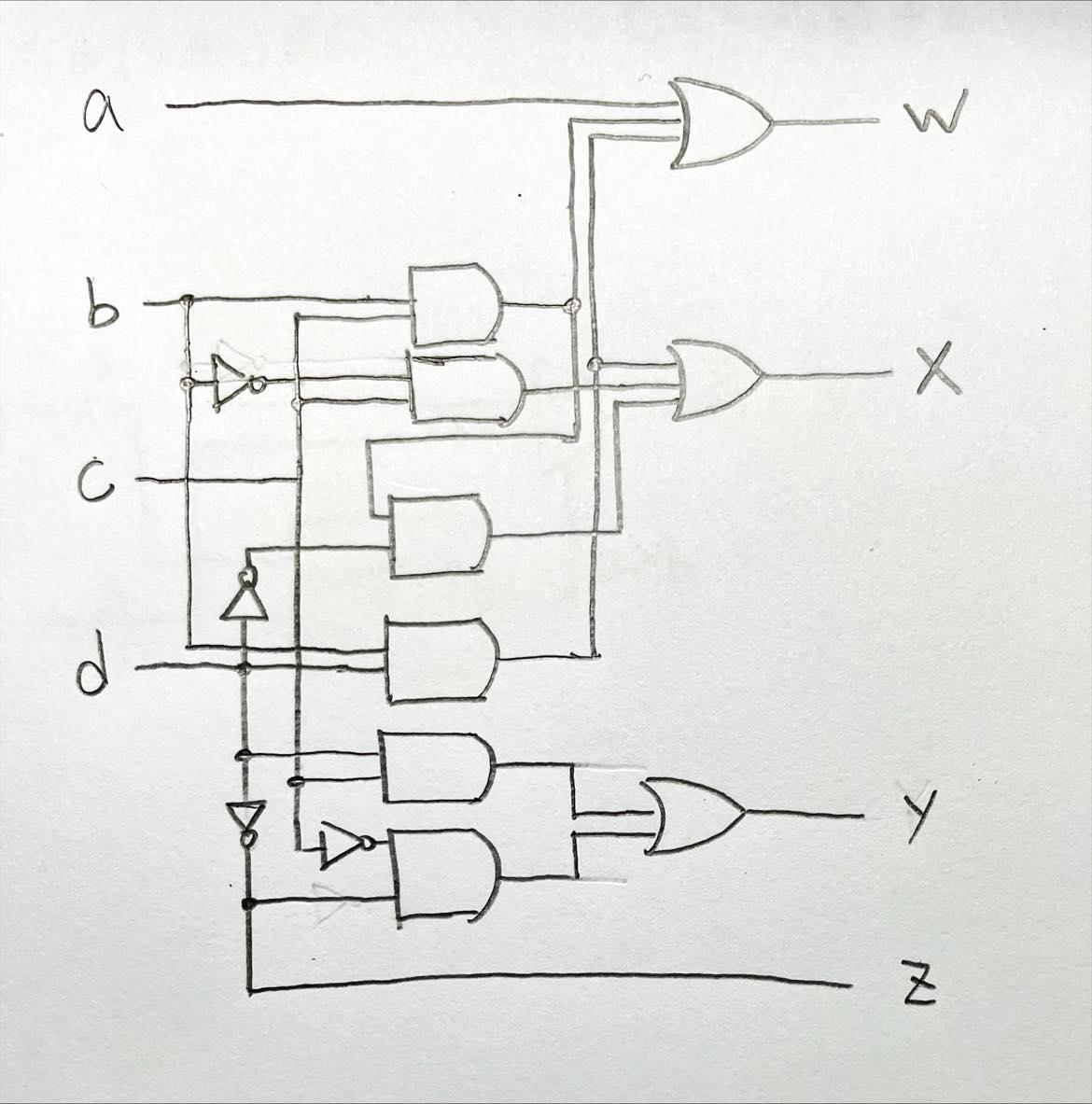
w = a+bc+bd

x = b’c+bd+bcd’

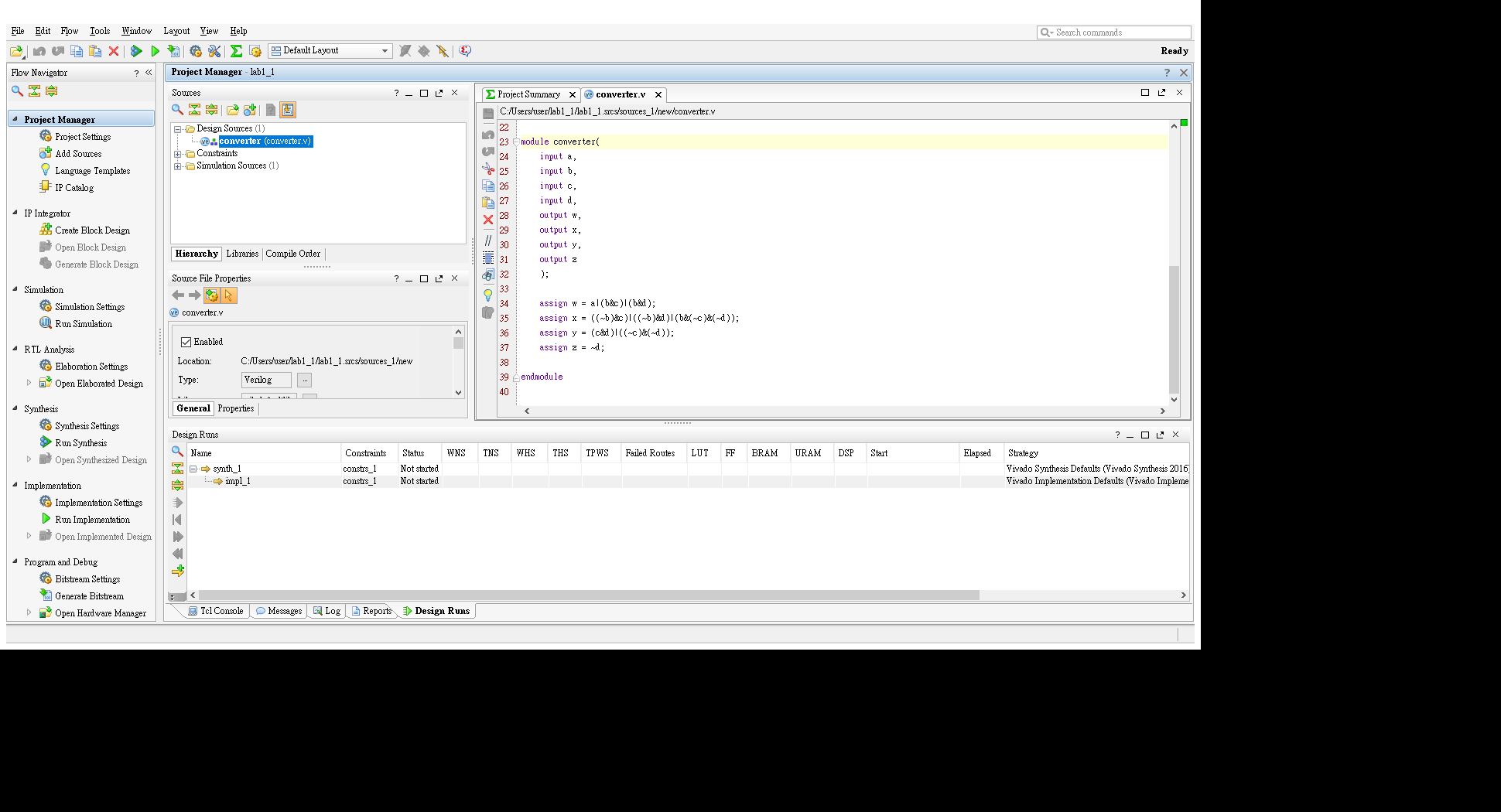
y = cd+c’d’

z = d’

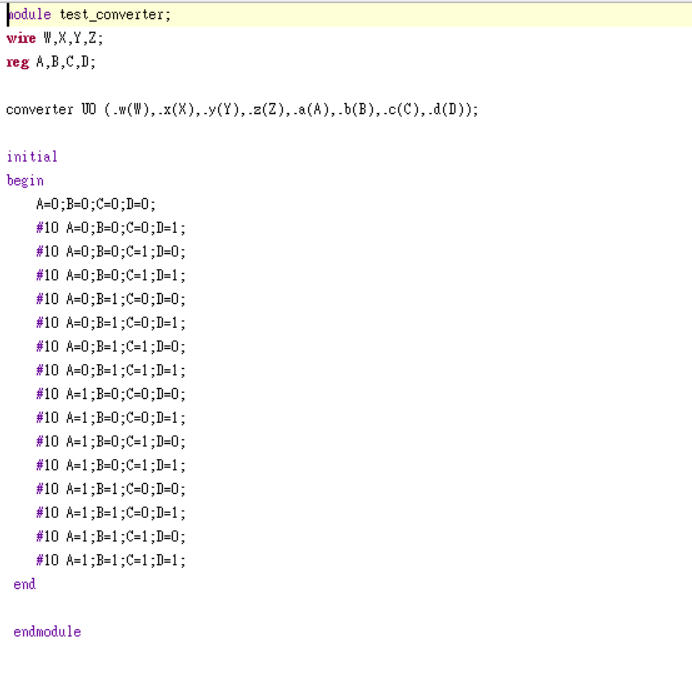
Logic diagram:



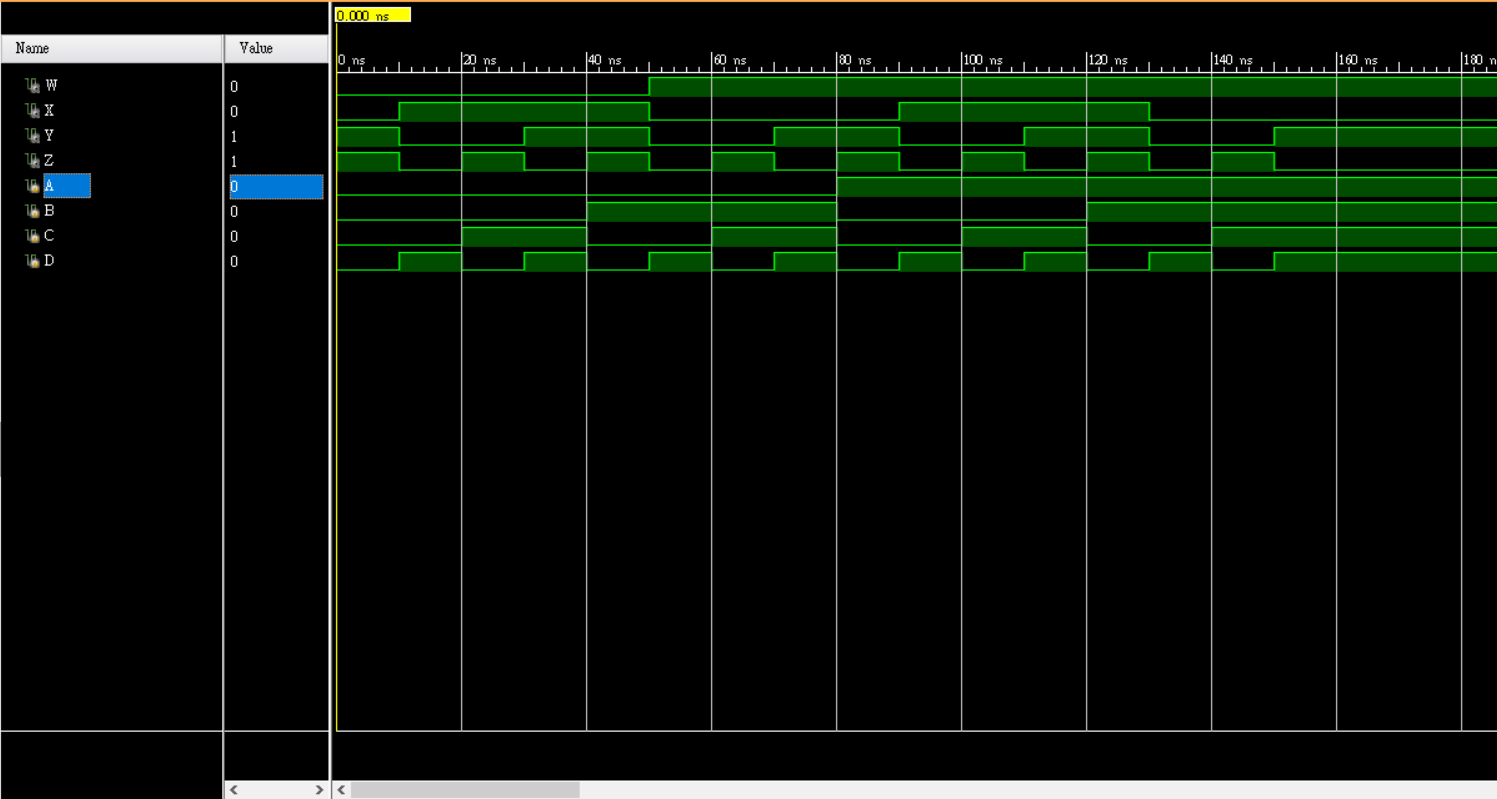
Finally, use logic function to construct Verilog RTL code for the converter:



Testbench:



Simulation:



**Ⅱ. Lab1\_2 (unsigned 2-bit x 2-bit binary multiplier)**

**Design Specification**

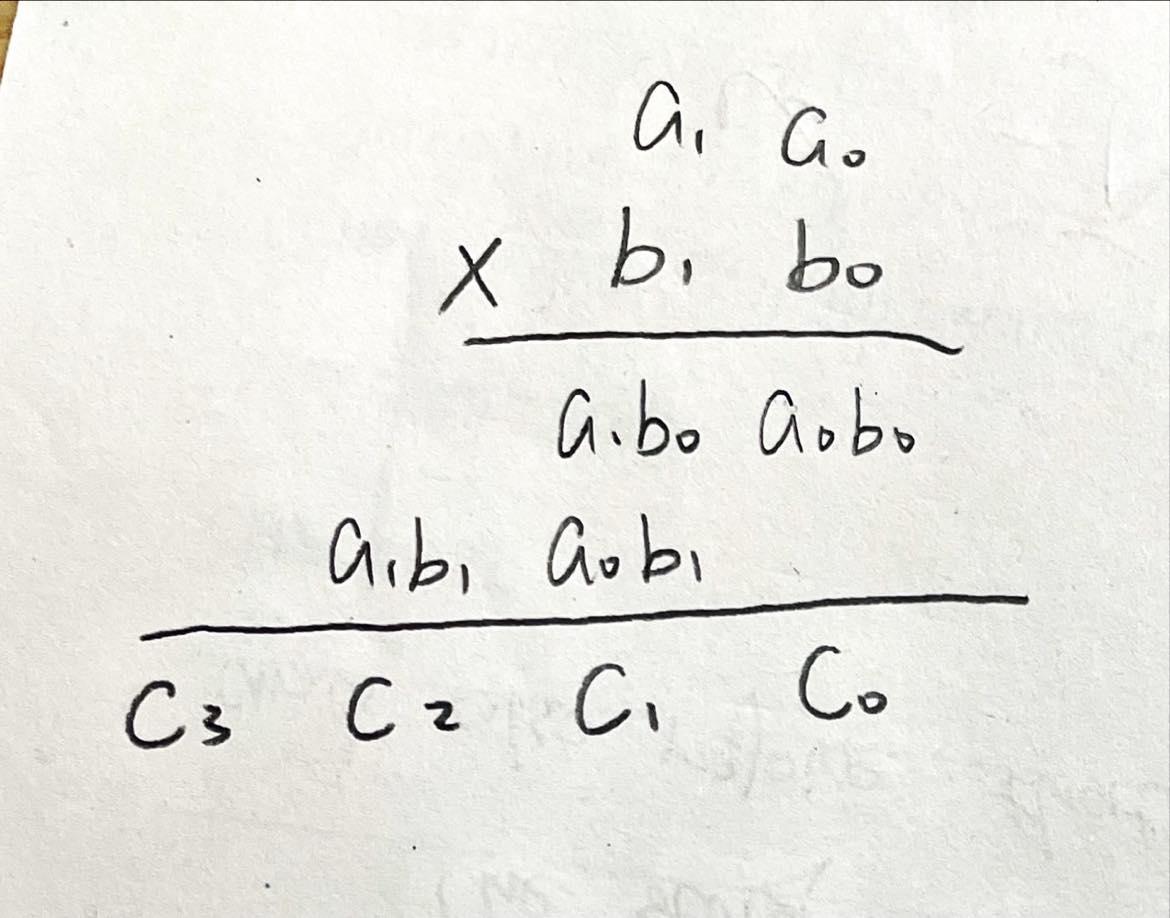
IO:

Input: a (a1,a0), b (b1,b0).

Output: c (c3,c2,c1,c0).

**Design Implementation**

First, write down straight formula as follows:

****

Then, we can obviously derive the logic function for multiplier

Boolean function:

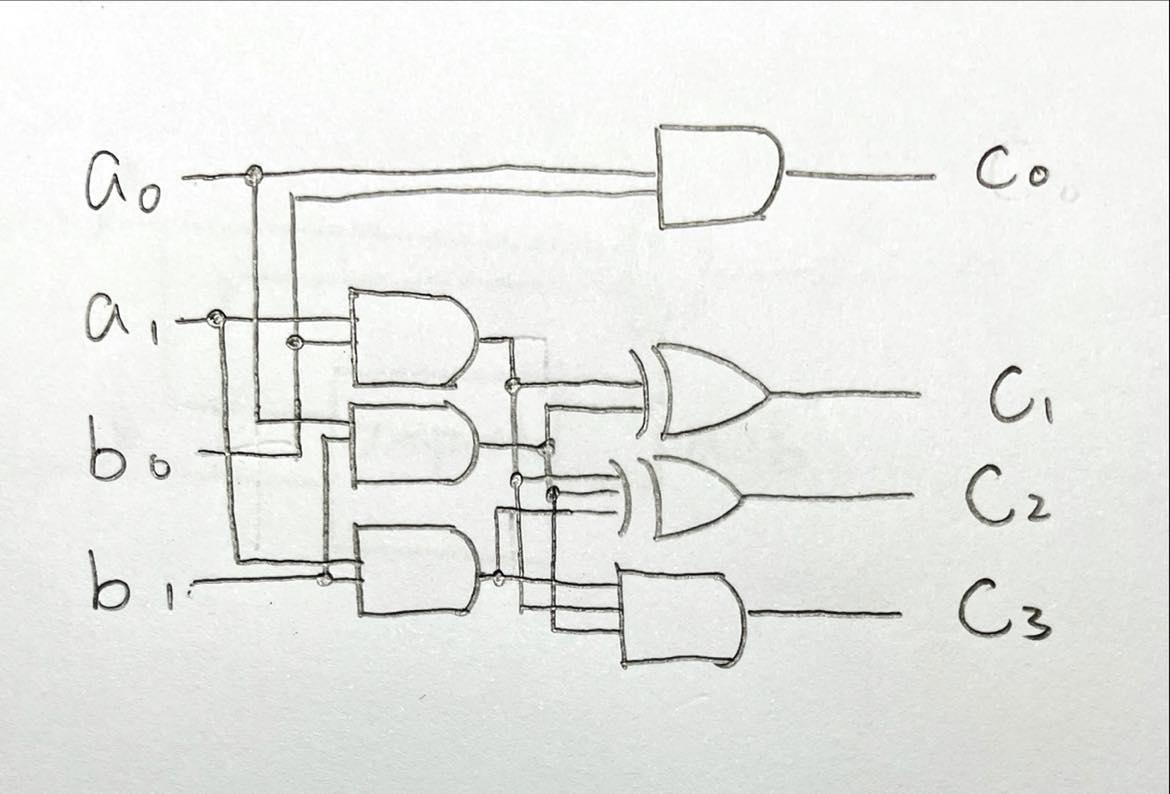
c3 = (a1b0\*a0b1)\*a1b1

c2 = (a1b0\*a0b1)⊕a1b1

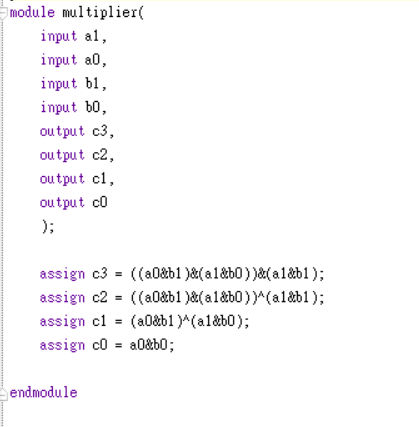
c1 = a1b0⊕a0b1

c0 = a0b0

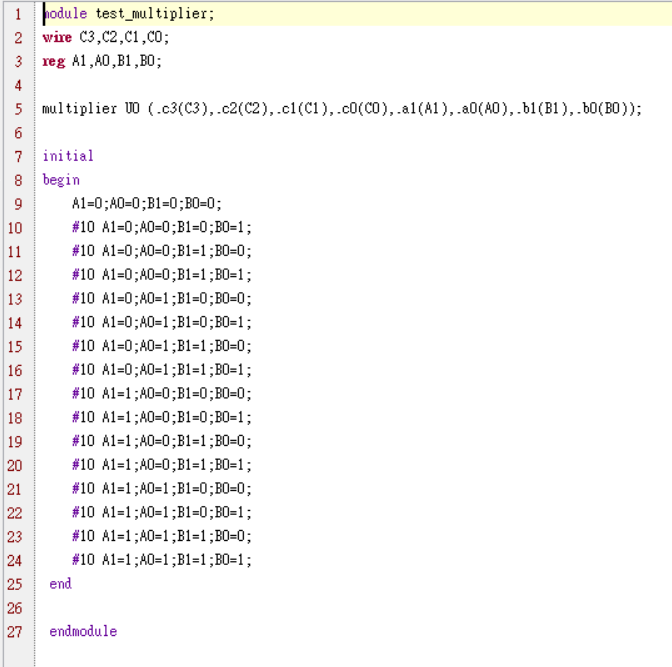
Logic diagram:



Finally, use logic function to construct Verilog RTL code for the multiplier:



Testbench:



Simulation:



**Ⅲ. Lab1\_3 (3-bit binary adder/subtractor)**

**Design Specification**

IO:

Input: a (a2a1a0), b (b2b1b0), m(operator control).

Output: s (s2s1s0), v(overflow indicator).

Block diagram:

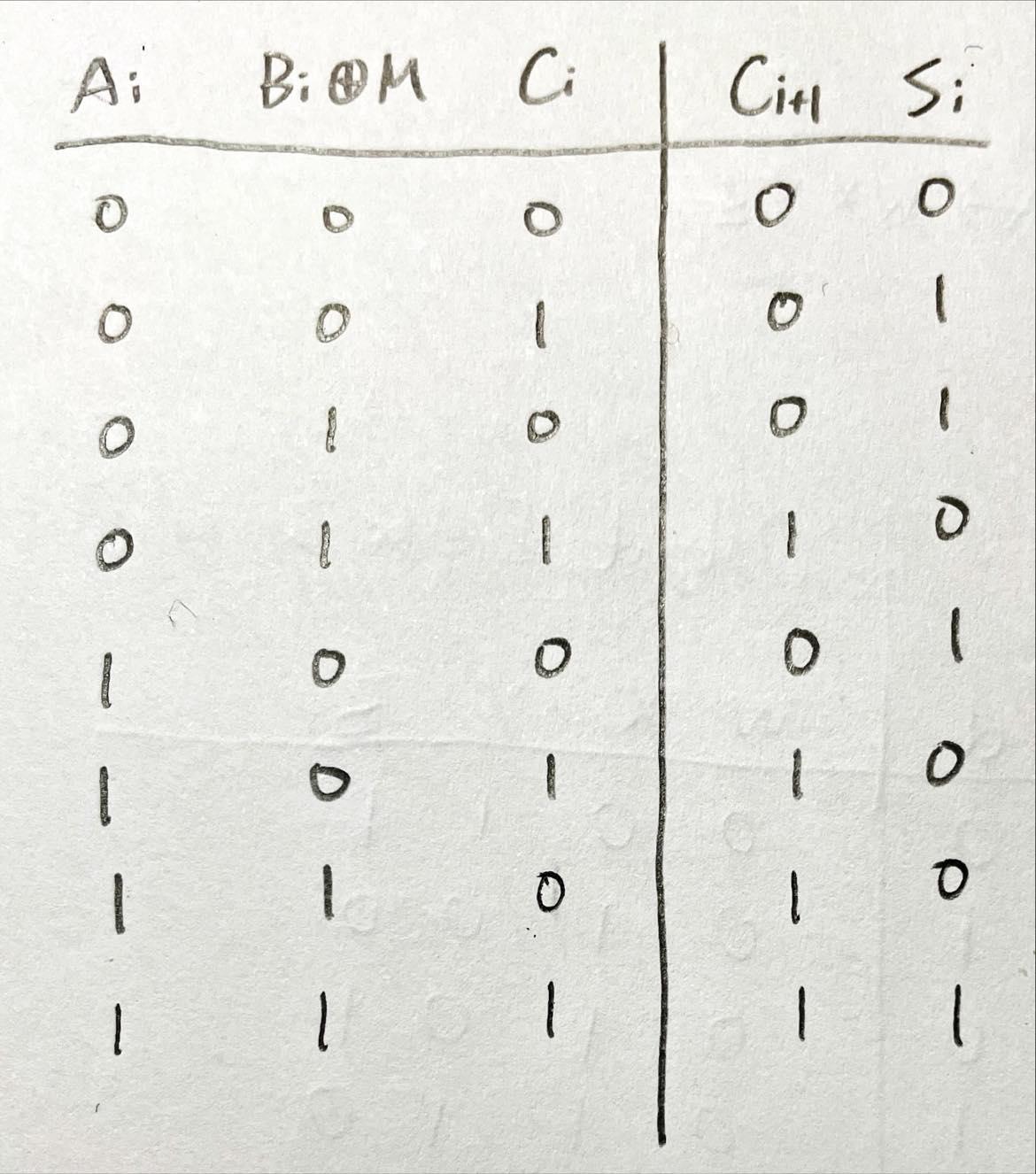


**Design Implementation**

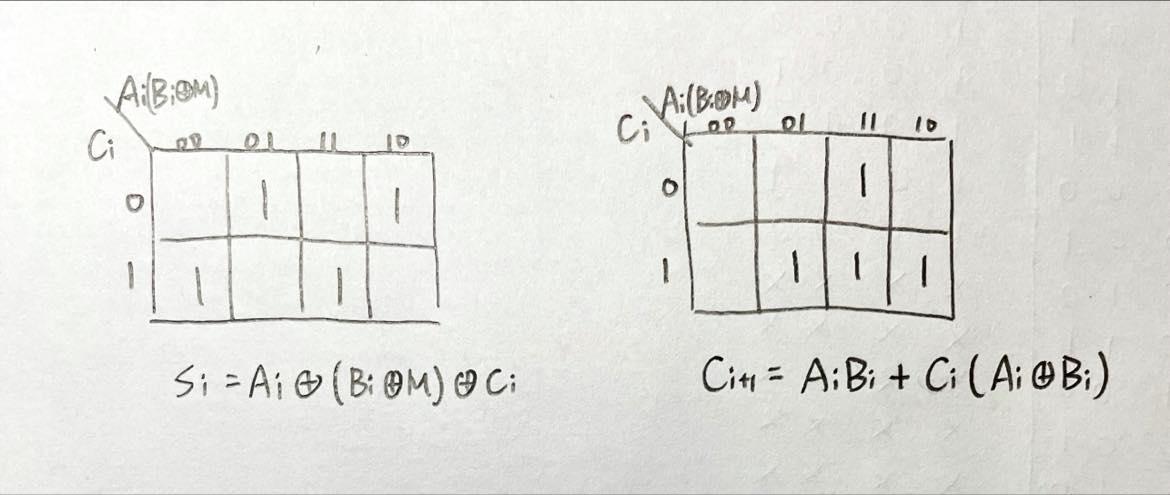
First, because of readability, we add a inout c[3:0] for carry.

Then, we can clearly realize the structure of adder/subtractor,

so we draw truth table for the full adder.



Use K-map to derive logic function:



Boolean function:

c[0] = m

c[1] = a[0](b[0]⊕m)+c[0](a[0]⊕(b[0]⊕ m))

c[2] = a[1](b[1]⊕m)+c[1](a[1]⊕(b[1]⊕m))

c[3] = a[2](b[2]⊕m)+c[2](a[2]⊕(b[2]⊕m))

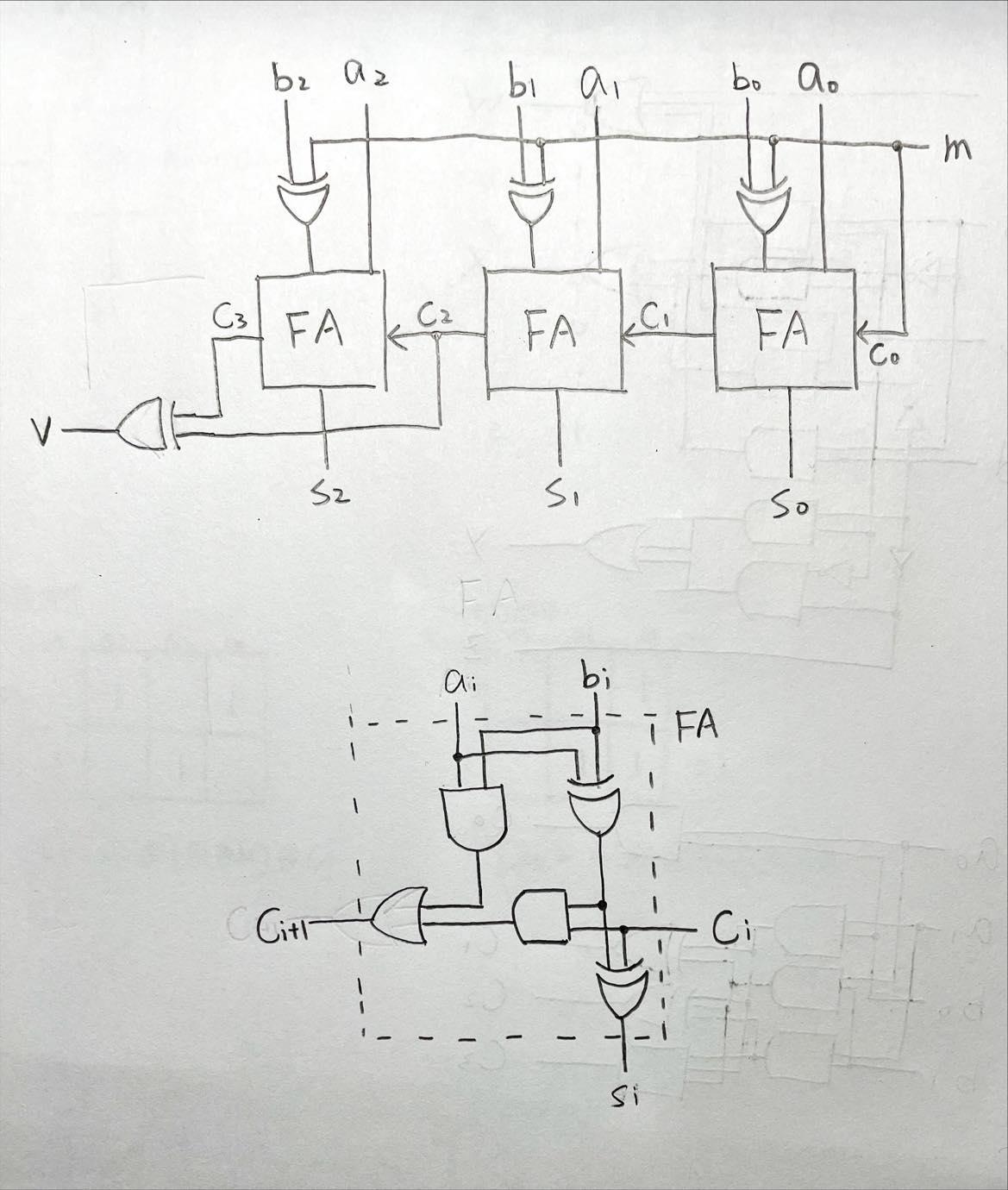
s[0] = a[0]⊕(b[0]⊕m)⊕c[0]

s[1] = a[1]⊕(b[1]⊕m)⊕c[1]

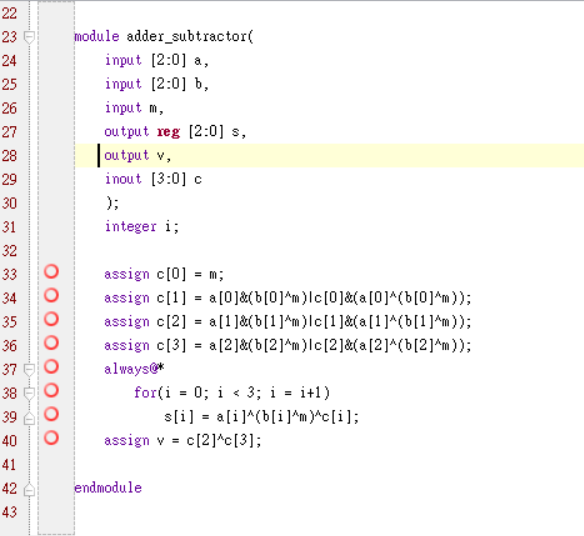
s[2] = a[2]⊕(b[2]⊕m)⊕c[2]

v = c[2]⊕c[3]

Logic diagram:



Finally, use logic function to construct Verilog RTL code for the adder/subtractor:



Simulation:



**Ⅳ. Lab1\_bonus (3-bit comparator)**

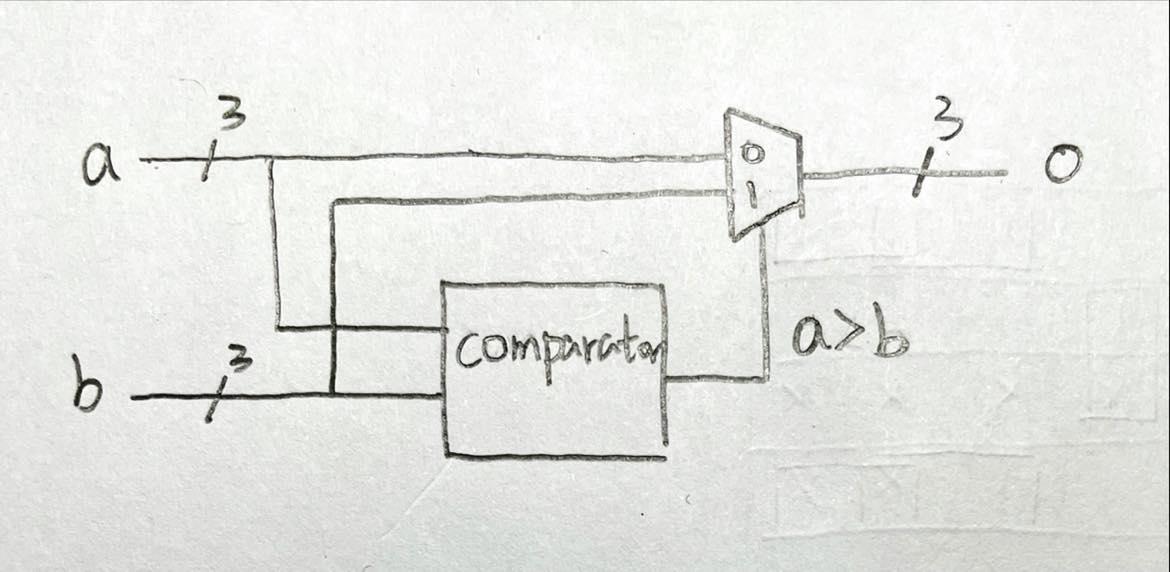
**Design Specification**

IO:

Input: a (a2,a1,a0), b (b2,b1,b0).

Output: o (o2,o1,o0).

Block diagram:



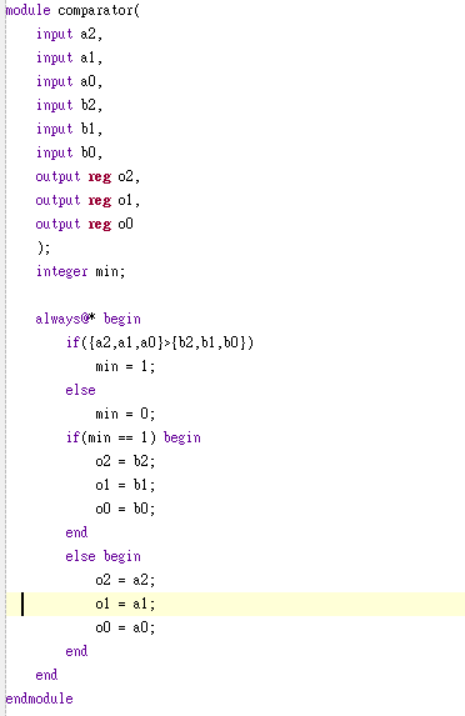
**Design Implementation**

First, according to block diagram, we design a comparator with if…else in Verilog .

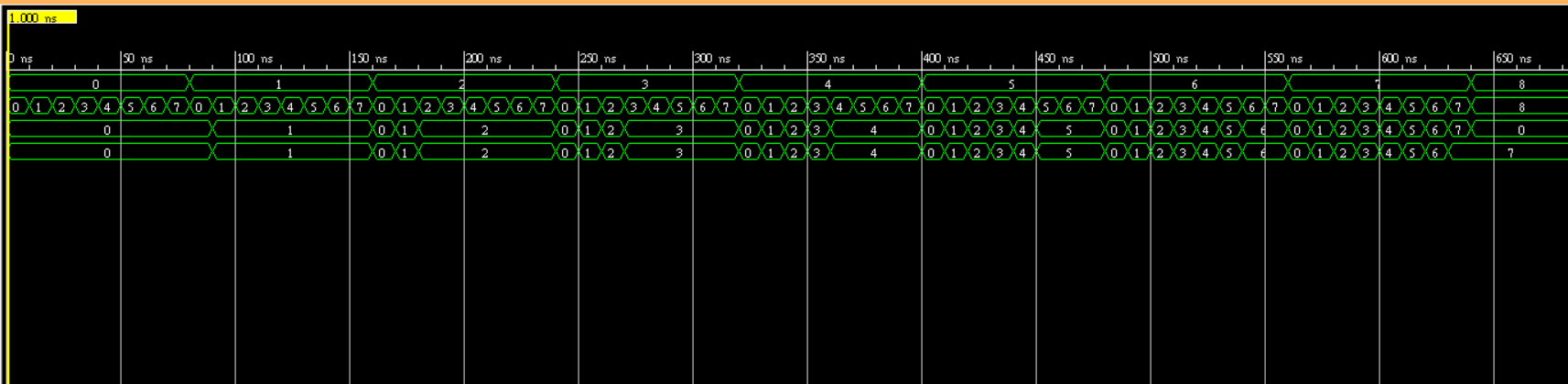
Then, if a>b, comparator outputs 1; if a≦b, comparator outputs 0.

So I declare an integer **min** to store comparator’s output , and o = smaller{a,b}.

Finally, use above information to construct Verilog RTL code for 3-bit comparator:



Simulation:



**Discussion**

**Lab1\_1:**

在實驗開始時，我使用truth table模擬出各種情形(什麼輸入導致什麼輸出)，最後logic function再使用K-map化簡，所以Verilog 模擬結果才符合預期。

1. 使用K-map化簡後的logic function是最精簡的嗎？

Ans：K-map化簡後的function一定是最簡單的and & or組合，雖然變數數量可能最為精簡，但消耗的邏輯可能並不是最少的，所以用K-map化簡完，再去整理才能確保用最小的消耗做出預期的功能。

**Lab1\_2:**

因為要做的是乘法，所以我直接寫出直式去模擬輸出，用最簡單且最好理解的方式推導出logic function，再丟到Verilog進行模擬。

1. 為何直式相加時，不是使用or而是使用exclusive–or？

Ans：因為2 bit相加時，如果兩個都是1，那出來的結果應該是0，1要進位，如果使用or，那出來的結果就是1了，與實際不相符，所以應要使用exclusive–or。

**Lab1\_3:**

在做加減法時，因為需要一個m去控制是加法還是減法，所以我把原本直接進入Ripple-carry Adder的B**i**換成B**i**⊕m，再照著Ripple-carry Adder的方式做出logic function。

**Lab1\_bonus:**

為了實現題目的功能，我先使用比較器比較出a是否大於b，再使用一個多功器去輸出較小的那個。

**Conclusion**

這次的lab雖然不難，但是我已經接近四個月沒碰verilog了，因此在語法等方面都不是那麼的熟練，尤其是always等號左邊的變數要宣告成reg就是我常常沒注意到的地方，也因此這次實驗還是花了不少時間debug和熟悉verilog，下次實驗就要開始接上電路板了，對我來說又是一個新的挑戰，這學期的邏設實驗讓我終於能將上學期所學的理論應用在實體上，也期望以後的實驗能夠更有效率地完成。

**References**

清華大學eeclass學習數位平台