超大型積體電路測試

VLSI testing

Homework I

國立清華大學

系所:電子所碩一

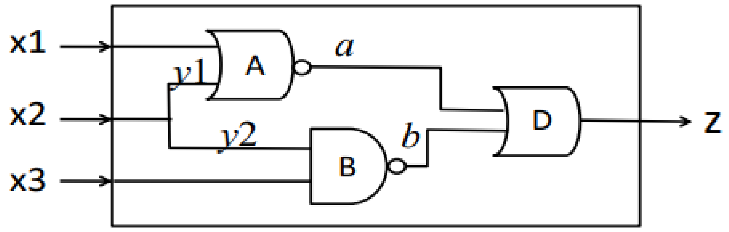
中文姓名:李聖謙

英文姓名:Sheng-Qian-Li

學號:111063517

授課老師:黃錫瑜

1.Consider the testing of a gate-level circuit as shown below. The primary input signals are {x1, x2, x3} and the primary output signal is {z}. The output signals of logic gates {A, B} are denoted as {a, b}, and the branches of primary input signal x2 is called y1 and y2, respectively.



(a) Write a software program (using C, C++, or any other programing language) that can exhaustively simulate the logic behavior of the given circuit under each of the 2 3 = 8 input vectors). Note that this can be done by executing the following Boolean equations in sequence in your program:

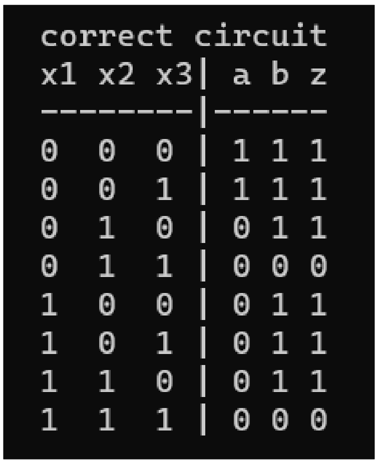
a = ~ (x1 or x2);

b = ~ (x2 and x3);

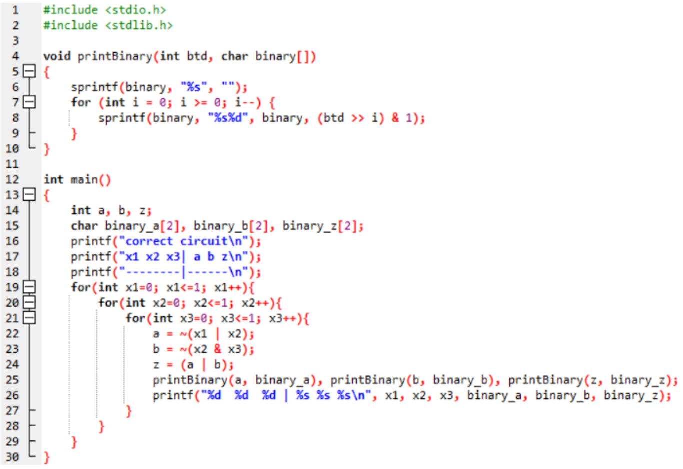
z = (a or b);

List the results as a truth table for output signal z. (Note this truth table contains 8 entries, one for each input combination).

Truth table & Code



```
correct circuit
x1 x2 x3| a b z
--------|------
0  0  0 | 1 1 1
0  0  1 | 1 1 1
0  1  0 | 0 1 1
0  1  1 | 0 0 0
1  0  0 | 0 1 1
1  0  1 | 0 1 1
1  1  0 | 0 1 1
1  1  1 | 0 0 0
```



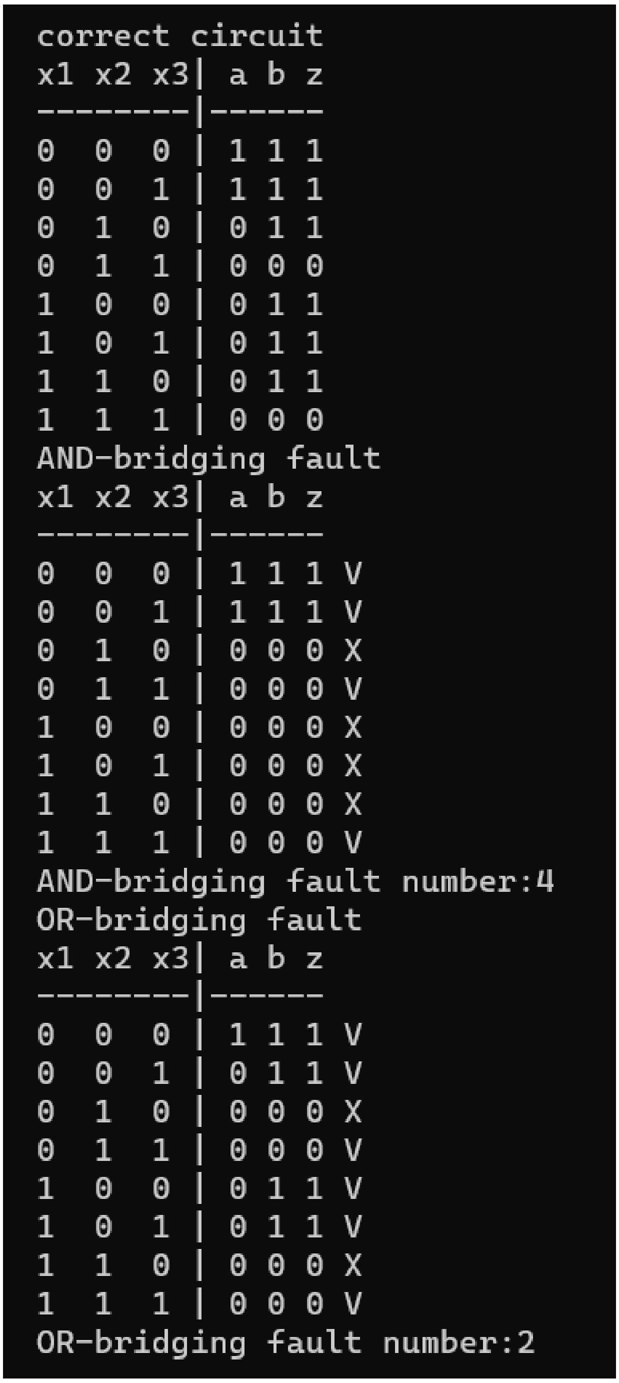
```c
#include <stdio.h>
#include <stdlib.h>

void printBinary(int btd, char binary[])
{
    sprintf(binary, "%s", "");
    for (int i = 0; i >= 0; i--) {
        sprintf(binary, "%s%d", binary, (btd >> i) & 1);
    }
}

int main()
{
    int a, b, z;
    char binary_a[2], binary_b[2], binary_z[2];
    printf("correct circuit\n");
    printf("x1 x2 x3| a b z\n");
    printf("--------|------\n");
    for(int x1=0; x1<=1; x1++){
        for(int x2=0; x2<=1; x2++){
            for(int x3=0; x3<=1; x3++){
                a = ~(x1 | x2);
                b = ~(x2 & x3);
                z = (a | b);
                printBinary(a, binary_a), printBinary(b, binary_b), printBinary(z, binary_z);
                printf("%d  %d  %d | %s %s %s\n", x1, x2, x3, binary_a, binary_b, binary_z);
            }
        }
    }
}
```

(b) Enhance your program so that it can perform exhaustive fault simulation for bridging faults, **{AND-bridging between a and b}** and **{OR-bridging between y1 and x3}**. Note that you need to report the total number of possible test patterns for each of the above two bridging faults. (Hint: perform fault injection, run exhaustive simulation on the faulty circuit, and then compare the results with those of the fault-free circuit. The fault injection can be done manually by changing the compiled code.)

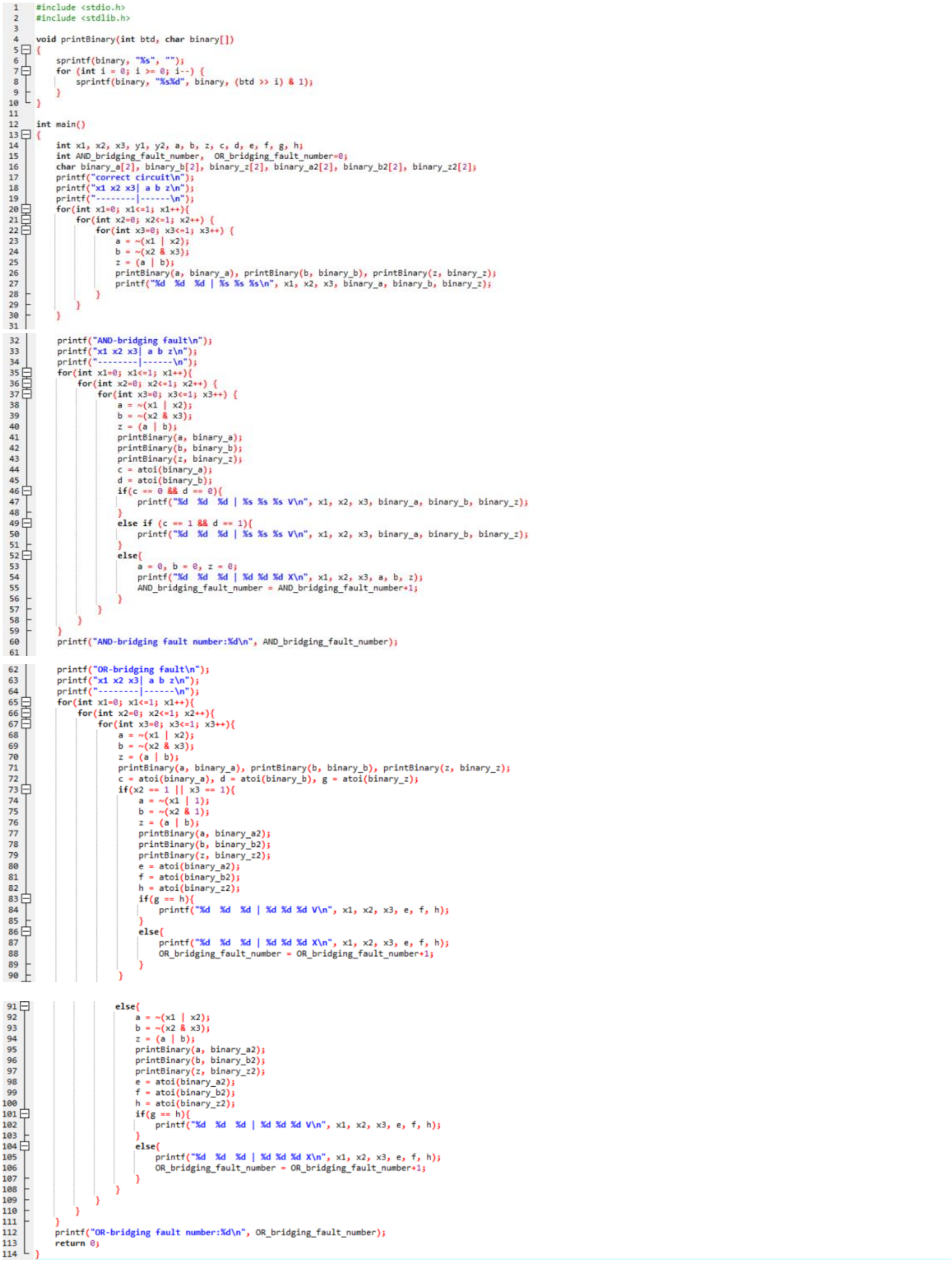
Truth table



```
correct circuit
x1 x2 x3| a b z
--------|------
0  0  0 | 1 1 1
0  0  1 | 1 1 1
0  1  0 | 0 1 1
0  1  1 | 0 0 0
1  0  0 | 0 1 1
1  0  1 | 0 1 1
1  1  0 | 0 1 1
1  1  1 | 0 0 0
AND-bridging fault
x1 x2 x3| a b z
--------|------
0  0  0 | 1 1 1 V
0  0  1 | 1 1 1 V
0  1  0 | 0 0 0 X
0  1  1 | 0 0 0 V
1  0  0 | 0 0 0 X
1  0  1 | 0 0 0 X
1  1  0 | 0 0 0 X
1  1  1 | 0 0 0 V
AND-bridging fault number:4
OR-bridging fault
x1 x2 x3| a b z
--------|------
0  0  0 | 1 1 1 V
0  0  1 | 0 1 1 V
0  1  0 | 0 0 0 X
0  1  1 | 0 0 0 V
1  0  0 | 0 1 1 V
1  0  1 | 0 1 1 V
1  1  0 | 0 0 0 X
1  1  1 | 0 0 0 V
OR-bridging fault number:2
```

只比較輸出 z，正確的後面為 V，錯誤的為 X

Code



```c
#include <stdio.h>
#include <stdlib.h>

void printBinary(int btd, char binary[])
{
    sprintf(binary, "%s", "");
    for (int i = 0; i >= 0; i--) {
        sprintf(binary, "%s%d", binary, (btd >> i) & 1);
    }
}

int main()
{
    int x1, x2, x3, y1, y2, a, b, z, c, d, e, f, g, h;
    int AND_bridging_fault_number,  OR_bridging_fault_number=0;
    char binary_a[2], binary_b[2], binary_z[2], binary_a2[2], binary_b2[2], binary_z2[2];
    printf("correct circuit\n");
    printf("x1 x2 x3| a b z\n");
    printf("--------|------\n");
    for(int x1=0; x1<=1; x1++){
        for(int x2=0; x2<=1; x2++) {
            for(int x3=0; x3<=1; x3++) {
                a = ~(x1 | x2);
                b = ~(x2 & x3);
                z = (a | b);
                printBinary(a, binary_a), printBinary(b, binary_b), printBinary(z, binary_z);
                printf("%d  %d  %d | %s %s %s\n", x1, x2, x3, binary_a, binary_b, binary_z);
            }
        }
    }

    printf("AND-bridging fault\n");
    printf("x1 x2 x3| a b z\n");
    printf("--------|------\n");
    for(int x1=0; x1<=1; x1++){
        for(int x2=0; x2<=1; x2++) {
            for(int x3=0; x3<=1; x3++) {
                a = ~(x1 | x2);
                b = ~(x2 & x3);
                z = (a | b);
                printBinary(a, binary_a);
                printBinary(b, binary_b);
                printBinary(z, binary_z);
                c = atoi(binary_a);
                d = atoi(binary_b);
                if(c == 0 && d == 0){
                    printf("%d  %d  %d | %s %s %s V\n", x1, x2, x3, binary_a, binary_b, binary_z);
                }
                else if (c == 1 && d == 1){
                    printf("%d  %d  %d | %s %s %s V\n", x1, x2, x3, binary_a, binary_b, binary_z);
                }
                else{
                    a = 0, b = 0, z = 0;
                    printf("%d  %d  %d | %d %d %d X\n", x1, x2, x3, a, b, z);
                    AND_bridging_fault_number = AND_bridging_fault_number+1;
                }
            }
        }
    }
    printf("AND-bridging fault number:%d\n", AND_bridging_fault_number);

    printf("OR-bridging fault\n");
    printf("x1 x2 x3| a b z\n");
    printf("--------|------\n");
    for(int x1=0; x1<=1; x1++){
        for(int x2=0; x2<=1; x2++){
            for(int x3=0; x3<=1; x3++){
                a = ~(x1 | x2);
                b = ~(x2 & x3);
                z = (a | b);
                printBinary(a, binary_a), printBinary(b, binary_b), printBinary(z, binary_z);
                c = atoi(binary_a), d = atoi(binary_b), g = atoi(binary_z);
                if(x2 == 1 || x3 == 1){
                    a = ~(x1 | 1);
                    b = ~(x2 & 1);
                    z = (a | b);
                    printBinary(a, binary_a2);
                    printBinary(b, binary_b2);
                    printBinary(z, binary_z2);
                    e = atoi(binary_a2);
                    f = atoi(binary_b2);
                    h = atoi(binary_z2);
                    if(g == h){
                        printf("%d  %d  %d | %d %d %d V\n", x1, x2, x3, e, f, h);
                    }
                    else{
                        printf("%d  %d  %d | %d %d %d X\n", x1, x2, x3, e, f, h);
                        OR_bridging_fault_number = OR_bridging_fault_number+1;
                    }
                }

                else{
                    a = ~(x1 | x2);
                    b = ~(x2 & x3);
                    z = (a | b);
                    printBinary(a, binary_a2);
                    printBinary(b, binary_b2);
                    printBinary(z, binary_z2);
                    e = atoi(binary_a2);
                    f = atoi(binary_b2);
                    h = atoi(binary_z2);
                    if(g == h){
                        printf("%d  %d  %d | %d %d %d V\n", x1, x2, x3, e, f, h);
                    }
                    else{
                        printf("%d  %d  %d | %d %d %d X\n", x1, x2, x3, e, f, h);
                        OR_bridging_fault_number = OR_bridging_fault_number+1;
                    }
                }
            }
        }
    }
    printf("OR-bridging fault number:%d\n", OR_bridging_fault_number);
    return 0;
}
```

Discussion

本次作業我使用 c 語言來完成此邏輯閘的真值表。

因為 AND-bridging fault 的原因，所以當 a 或 b 有任意一個為 0 時，就會將另一個也變為 0，就是 AND 閘的概念，所以本題的輸出只會有 0 或 1，在 bridging 後原本的 OR 閘也等於無效，就不會產生 a 和 b 不同但輸出 z 卻相同的情況。

因為 OR-bridging fault 的原因，所以當 x1 或 y1 有任意一個為 1 時，就會將另一個也變為等同於在中間加了 OR 閘，但這會產生一個問題，如果只單純看 z 輸出的值與原本相同，可能中間的 a 和 b 點也要檢測錯誤的話會不一樣，例如(x1,x2,x3)=(0,0,1)時，正確的(a,b,z)=(1,1,1)，但 OR-bridging 後會變成(a,b,z)=(0,1,1)，可以看出 a 從 1 變 0 了，但因為本題只做 z 判斷，故將這個邏輯中的差別寫在下面討論。