Lab#1

Name: 張嵩禾
StudentID: B083040041
Subject: Test Pattern Generation
Date: 3/13

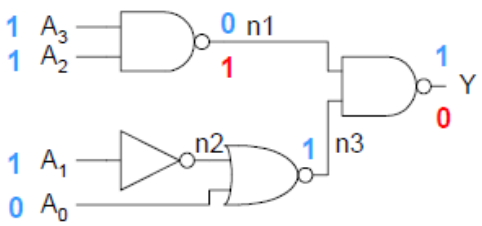
I. 實驗內容

目標：測試電路圖各輸入及輸出 stuck-at-fault(S1~S16)的情形，並依結果找出 Minimum Set。

Step 1：根據圖（一）建立邏輯閘，並將正確的輸出記錄在 out。
宣告一個陣列 flagger[16] 並初始為 0，定義 0000~1111 各輸入是否需 test。
宣告一個陣列 stk[16] 並初始為 0，定義為 0000~1111 各輸入的測試權重。
宣告變數 jumper 並初始為 0，定義為 stuck-at-fault(S1~S16)是否測試完成。

```
bool A0,A1,A2,A3,n1,n2,n3,y;
int outA3sa1[16],outA3sa0[16],ou
int out[16];

int flagger[16]={0};
int stk[16]={0};

int jumeper=0;
```

```
for(int aa=0;aa<16;aa++){
    int a[4]={0};
    num = aa;
    i=0;
    while (num > 0)
    {
        c = (num % n);
        a[i] = c;
        num = num / n;
        i++;
    }

    A3=a[3];
    A2=a[2];
    A1=a[1];
    A0=a[0];

    n1=!(A2&&A3);//NAND A3 A2
    n2=!(A1);//invert A1
    n3=!(n2||A0);//invert or
    y=!(n3&&n1);
    out[aa]=y;
}
```



▲圖（一）

Step 2：逐一測試 0000~1111 共 16 種輸入在各個輸入及輸出 stuck-at-0 或 stuck-at-1 的情形，並將答案記錄在 out*sa*的陣列中。以 A3 程式碼說明：

```
//A3
printf("\n");
for(int aa=0;aa<16;aa++){
    int a[4]={0};
    num = aa;
    i=0;
    while (num > 0)
    {
        c = (num % n);
        a[i] = c;
        num = num / n;
        i++;
    }
```

利用 for 迴圈測試 0000~1111 共 16 種輸入。

```
    A3=1;
    A2=a[2];
    A1=a[1];
    A0=a[0];
    //printf("\n?%d%d%d%d\n\n",A3,A2,A1,A0);
    n1=!(A2&&A3);//NAND A3 A2
    n2=!(A1);//invert A1
    n3=!(n2||A0);//invert or
    y=!(n3&&n1);
    outA3sa1[aa]=y;
}
```

將 A3 stuck-at-1，其餘不變，並將結果存在 outA3sa1。

```
for(int aa=0;aa<16;aa++){
    int a[4]={0};
    num = aa;
    i=0;
    while (num > 0)
    {
        c = (num % n);
        a[i] = c;
        num = num / n;
        i++;
    }
```

利用 for 迴圈測試 0000~1111 共 16 種輸入。

```
    A3=0;
    A2=a[2];
    A1=a[1];
    A0=a[0];
    //printf("\n?%d%d%d%d\n\n",A3,A2,A1,A0);
    n1=!(A2&&A3);//NAND A3 A2
    n2=!(A1);//invert A1
    n3=!(n2||A0);//invert or
    y=!(n3&&n1);
    outA3sa0[aa]=y;
}
```

將 A3 stuck-at-0，其餘不變，並將結果存在 outA3sa0。

其餘 A2、A1、A0、n1、n2、n3 及 Y 仿照相同方法，利用迴圈將 16 種不同組合輸入，並依序指定 stuck-at-1/stuck-at-0 將結果記錄在各個陣列中，將結果與 step1 計算出的正確輸出相比，若不相等則判定此輸入為可找出 stuck-at-fault 的 input pattern，並將 out*sa* 中的輸出改為 1，相等時則為 0；依此方法便可完成 S1~S16 逐項對應的所有 pattern，並可完成表格。
其中，在測試輸出是否相等時，需同時判斷 flagger 的條件，若輸出不相等且 flagger 等於 1，即此輸入必須被選為 Minimum set，因此可跳過此次 stuck-at-fault 的測試（jumper=1），並將 out*sa* 陣列清除為 0，代表已測試過；若輸出不相等但 flagger 等於 0，即代表此輸入仍為被選入 Minimum set，則必須計算此輸入的權重存在 stk 中，並將 out*sa* 陣列修正為 0/1，1 代表當前輸入可為 test pattern。

以 A3 程式碼說明：

```
for(int x=0;x<16;x++)
    printf("%4d",x);
printf("\n");
for(int x=0;x<16;x++){
    if(outA3sa1[x]!=out[x]&&flagger[x]==1){
        jumeper=1;
    }
}
if(jumeper==1){
    for(int x=0;x<16;x++)
        outA3sa1[x]=0;
    jumeper=0;
}
else{
    for(int x=0;x<16;x++)
        if(outA3sa1[x]!=out[x]&&flagger[x]==0){
            stk[x]++;
            outA3sa1[x]=1;
            printf("%4d",1);
        }
        else{
            outA3sa1[x]=0;
            printf("%4d",0);

    }
}
```

逐一測試是否答案不相等，並同時判斷是否已選入 Minimum set

若已選入 Minimum set 中即可略過當前測試目標，並將陣列清除為 0

若未選入 Minimum set 則需計算當前輸入的權重，將值記錄在 stk 中。此外將答案不相等、發生 stuck-at-fault 的地方紀錄為 1。

Step 3: 為了找出可偵測到所有 Stuck-at-fault 的 Minimum Set，我採用 Greedy 演算法，即為當儲存於 stk 變數中的測試權重越大時，優先將此輸入納入 Minimum set 中（當回合的 solution），並將此輸入的 flagger 設為 1，使得在重複 Step2 時將不必測試的情況排除。

```
//output

printf("\ninput: ");
for(int x=0;x<16;x++){

    printf("%4d",stk[x]);
}
```

```
int maxx=0;
int flag;
for(int x=0;x<16;x++){
    if(stk[x]>maxx){
        maxx=stk[x];
        flag=x;
    }
}
```

找出測試權重最大的值。

```
printf("\nsol: %d\n",flag);
flagger[flag]=1;
}
```

將最大的權重輸出為 solution，並將該輸入的 flagger 設為 1。

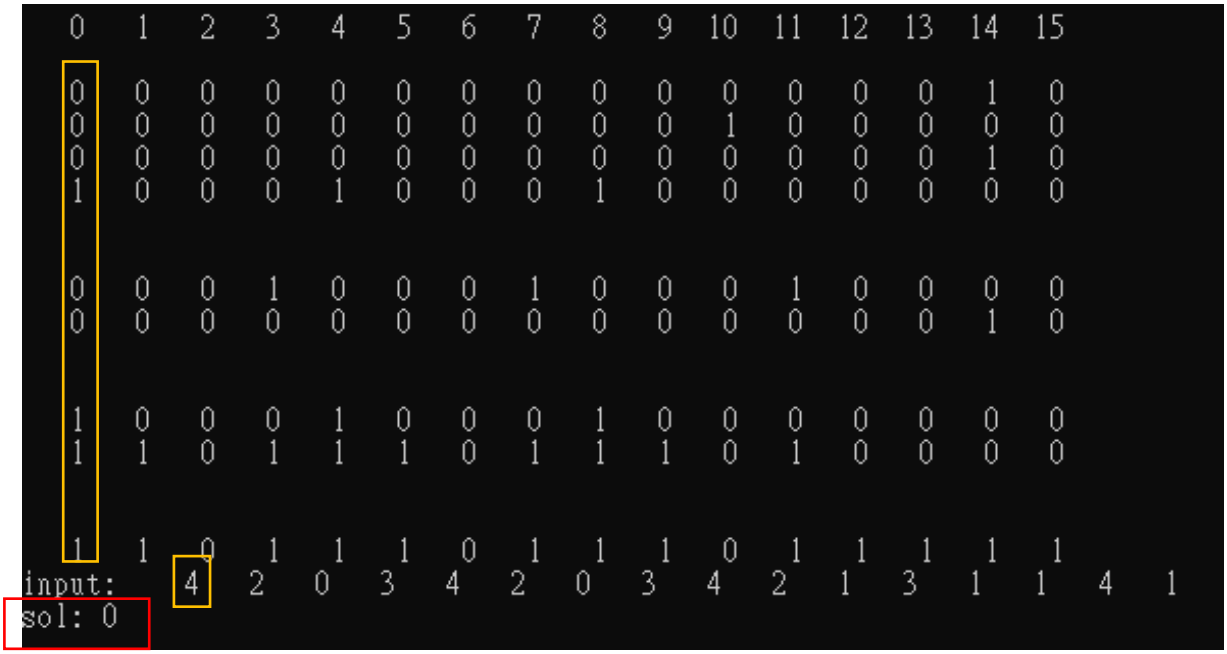
II. 實驗過程及結果分析

根據以上步驟可找出每一個 stuck-at-fault(S1~S16)對應的 input pattern。

並且第一輪時所有輸入的 flagger 都為 0，因此計算所有輸入的測試權重，顯示在 input 中。由 input 中可判斷 $6 = (0110)_2$ 有最大的測試權重，因此將 6 選入為 solution 中。

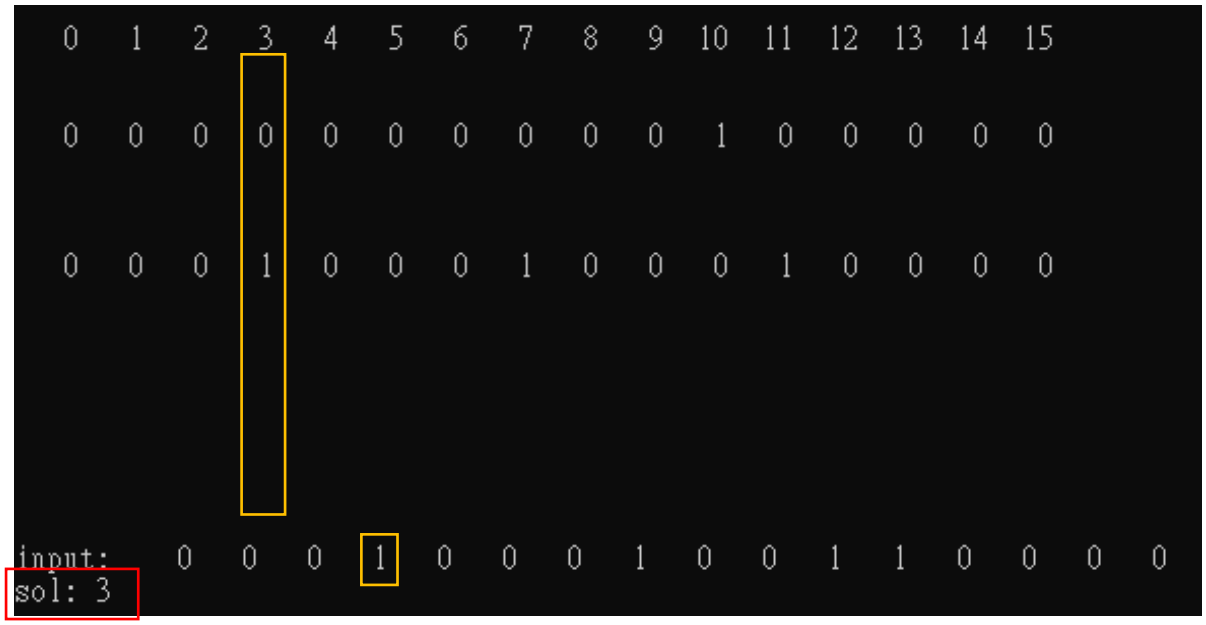
```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
  0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0
  0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0
  1   0   0   0   1   0   0   0   1   0   0   0   0   0   0   0
  0   0   1   0   0   0   1   0   0   0   1   0   0   0   0   0
  0   0   1   0   0   0   1   0   0   0   1   0   0   0   0   0
  0   0   0   1   0   0   0   1   0   0   0   1   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0
  0   0   1   0   0   0   1   0   0   0   1   0   0   0   0   0
  0   0   1   0   0   0   1   0   0   0   1   0   0   0   0   0
  1   0   0   0   1   0   0   0   1   0   0   0   0   0   0   0
  1   1   0   1   1   1   0   1   1   1   0   1   0   0   0   0
  0   0   1   0   0   0   1   0   0   0   1   0   0   0   0   0
  0   0   1   0   0   0   1   0   0   0   1   0   0   0   0   0
  1   1   0   1   1   1   0   1   1   1   0   1   1   1   1   1
input:    4   2   6   3   4   2   7   3   4   2   7   3   1   1   4   1
sol: 6
```

承接第一輪的結果，當 6 被選入為 Minimum set 中時(flagger=1)，S1、S6、S7、S10、S11、S14、S15 即完成測試(jumper=1)；後續如同第一輪，再繼續計算每一個輸入的測試權重，在此輪時可判斷 $0 = (0000)_2$ 時有最大的測試權重，因此將 0 選入為 solution 中。

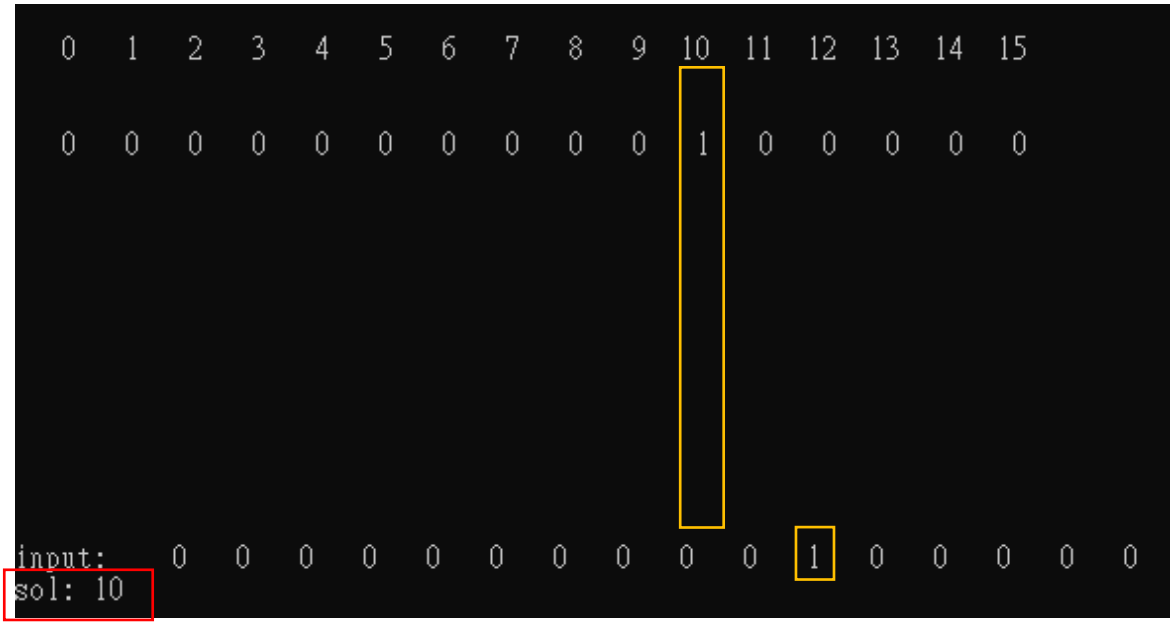


承接第二輪的結果，當 0 被選入為 Minimum set 中時(flagger=1)，S5、S12、S13、S16 即完成測試(jumper=1)；後續如同先前，再繼續計算每一個輸入的測試權重，在此輪時可判斷 $14 = (1110)_2$ 時有最大的測試權重，因此將 14 選入為 solution 中。

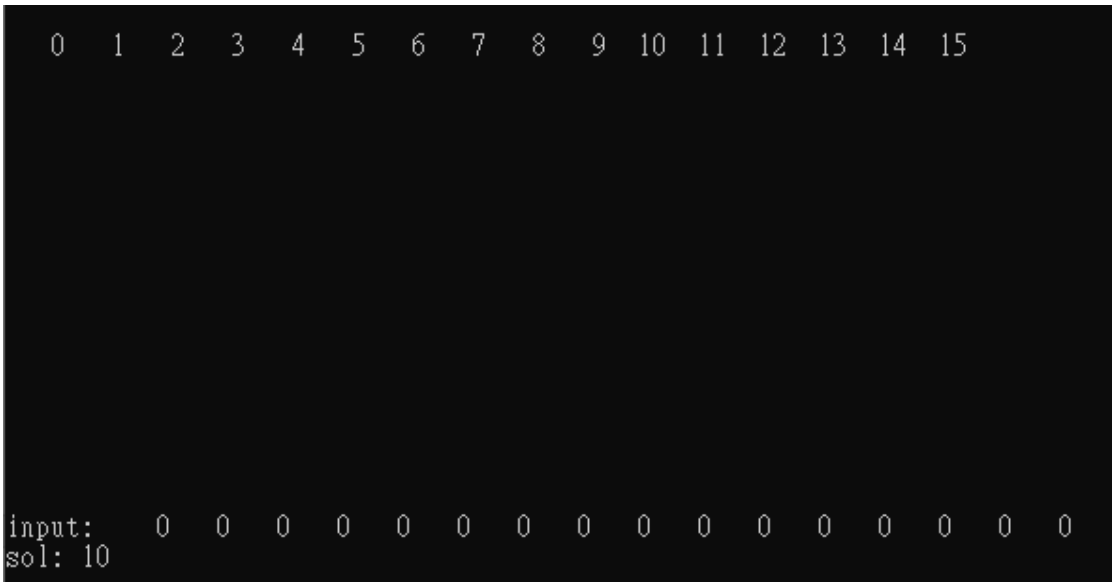
承接第三輪的結果，當 14 被選入為 Minimum set 中時(flagger=1)，S2、S4、S9 即完成測試(jumper=1)；後續如同先前，再繼續計算每一個輸入的測試權重，在此輪時可判斷 $3 = (0011)_2$ 時有最大的測試權重，因此將 3 選入為 solution 中。



承接第四輪的結果，當 3 被選入為 Minimum set 中時(flagger=1)，S8 即完成測試(jumper=1)；後續如同先前，再繼續計算每一個輸入的測試權重，在此輪時可判斷 $10 = (1010)_2$ 時有最大的測試權重，因此將 10 選入為 solution 中。



承接第四輪的結果，當 10 被選入為 Minimum set 中時(flagger=1)，S3 即完成測試(jumper=1)，此時所有測試權重皆為 0，代表 Minimum set 已經全數找出。



綜合各輪的結果，可找出 Minimum Set 為: **{6、0、14、3、10}**

III. 實驗心得

我覺得這次實驗滿有趣的，可以透過電子自動化設計來驗證電路的正確性，用 Greedy 演算法就可以找到不錯的 Minimum Set，在設計演算法時可以多多考慮，嘗試不同的方式來得到全部的解。