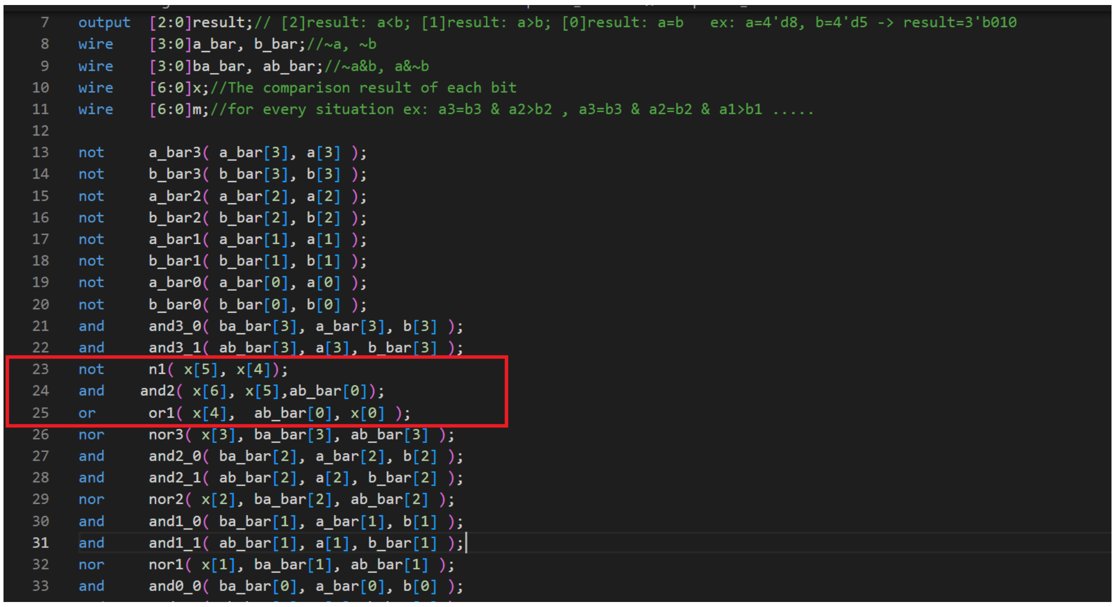
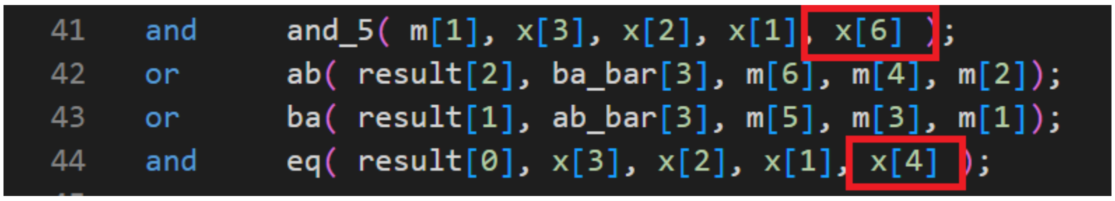
一、電路偵測與改善

Case1

Verilog Code Trojans:



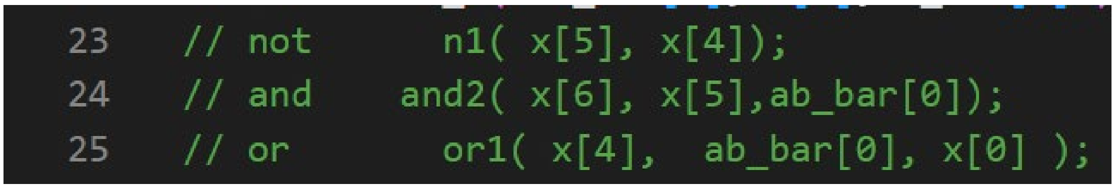
```
output  [2:0]result;// [2]result: a<b; [1]result: a>b; [0]result: a=b   ex: a=4'd8, b=4'd5 -> result=3'b010
wire    [3:0]a_bar, b_bar;//~a, ~b
wire    [3:0]ba_bar, ab_bar;//~a&b, a&~b
wire    [6:0]x;//The comparison result of each bit
wire    [6:0]m;//for every situation ex: a3=b3 & a2>b2 , a3=b3 & a2=b2 & a1>b1 .....

not     a_bar3( a_bar[3], a[3] );
not     b_bar3( b_bar[3], b[3] );
not     a_bar2( a_bar[2], a[2] );
not     b_bar2( b_bar[2], b[2] );
not     a_bar1( a_bar[1], a[1] );
not     b_bar1( b_bar[1], b[1] );
not     a_bar0( a_bar[0], a[0] );
not     b_bar0( b_bar[0], b[0] );
and     and3_0( ba_bar[3], a_bar[3], b[3] );
and     and3_1( ab_bar[3], a[3], b_bar[3] );
not     n1( x[5], x[4]);
and    and2( x[6], x[5],ab_bar[0]);
or      or1( x[4],  ab_bar[0], x[0] );
nor     nor3( x[3], ba_bar[3], ab_bar[3] );
and     and2_0( ba_bar[2], a_bar[2], b[2] );
and     and2_1( ab_bar[2], a[2], b_bar[2] );
nor     nor2( x[2], ba_bar[2], ab_bar[2] );
and     and1_0( ba_bar[1], a_bar[1], b[1] );
and     and1_1( ab_bar[1], a[1], b_bar[1] );
nor     nor1( x[1], ba_bar[1], ab_bar[1] );
and     and0_0( ba_bar[0], a_bar[0], b[0] );
```



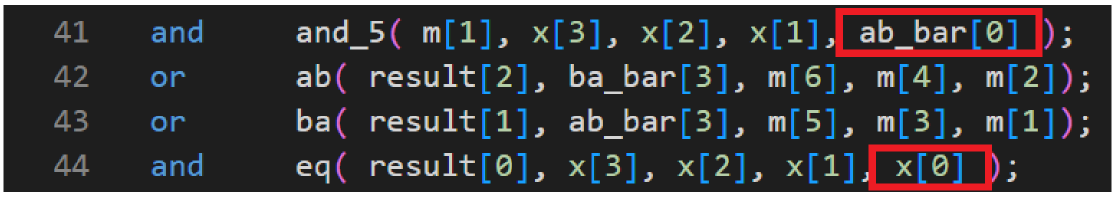
```
and     and_5( m[1], x[3], x[2], x[1], x[6] );
or      ab( result[2], ba_bar[3], m[6], m[4], m[2]);
or      ba( result[1], ab_bar[3], m[5], m[3], m[1]);
and     eq( result[0], x[3], x[2], x[1], x[4] );
```

Golden circuit:

對下列幾行 code 做更改即可使電路功能正常。



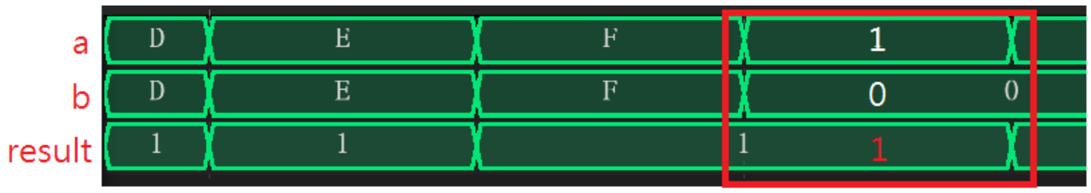
```
// not      n1( x[5], x[4]);
// and     and2( x[6], x[5],ab_bar[0]);
// or       or1( x[4],  ab_bar[0], x[0] );
```

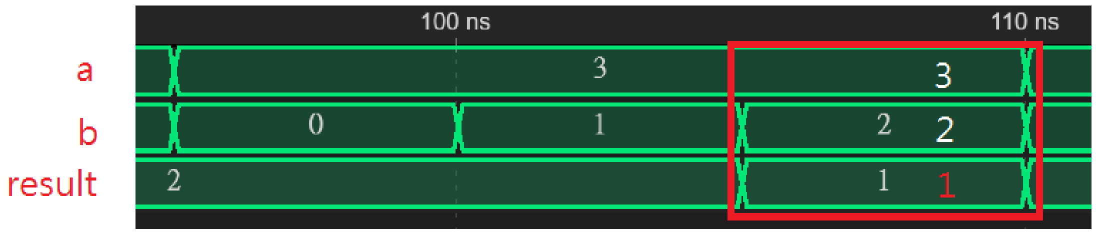


```
and     and_5( m[1], x[3], x[2], x[1], ab_bar[0] );
or      ab( result[2], ba_bar[3], m[6], m[4], m[2]);
or      ba( result[1], ab_bar[3], m[5], m[3], m[1]);
and     eq( result[0], x[3], x[2], x[1], x[0] );
```

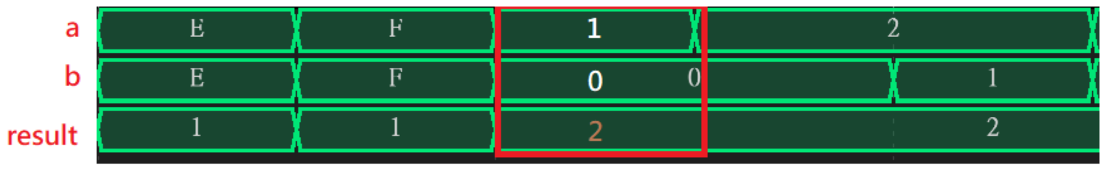
result:

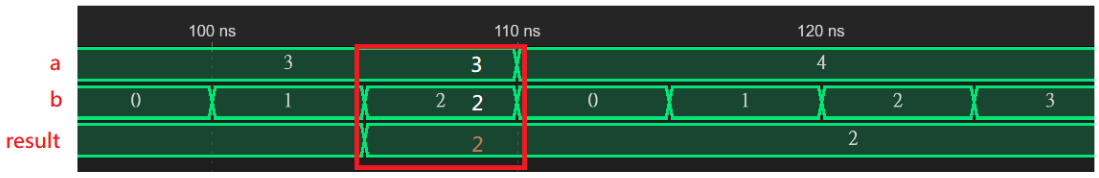
1. **Infected circuit:**





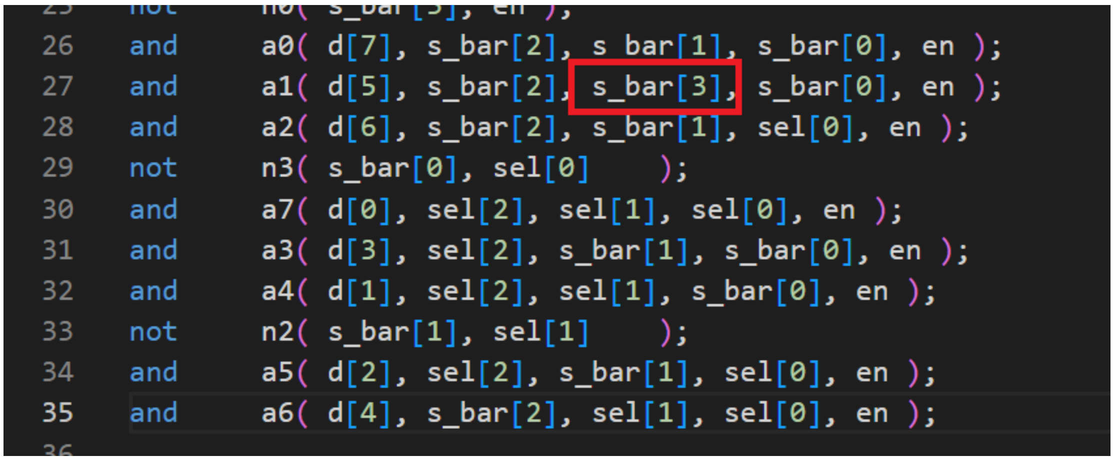
2. **Golden circuit:**





Case2

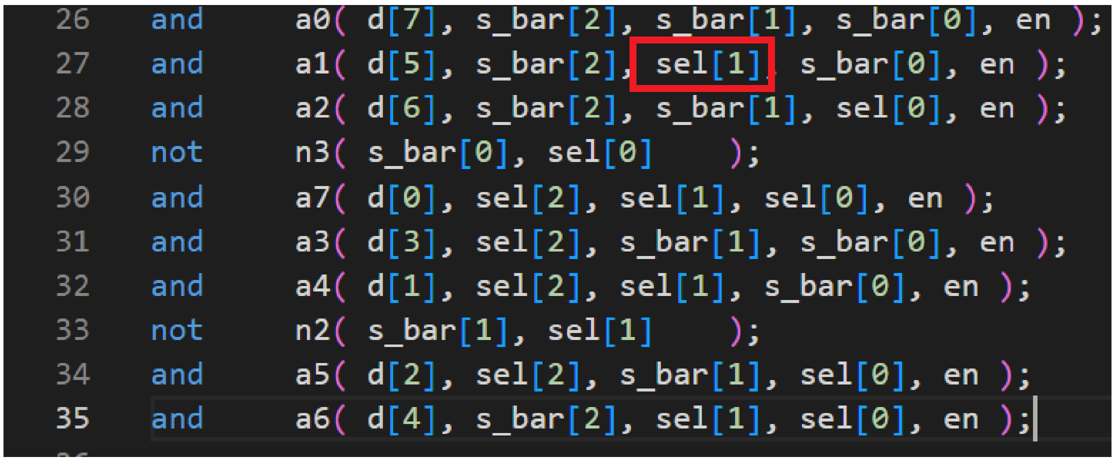
Verilog Code Trojans:



```
and     a0( d[7], s_bar[2], s_bar[1], s_bar[0], en );
and     a1( d[5], s_bar[2], s_bar[3], s_bar[0], en );
and     a2( d[6], s_bar[2], s_bar[1], sel[0], en );
not     n3( s_bar[0], sel[0]     );
and     a7( d[0], sel[2], sel[1], sel[0], en );
and     a3( d[3], sel[2], s_bar[1], s_bar[0], en );
and     a4( d[1], sel[2], sel[1], s_bar[0], en );
not     n2( s_bar[1], sel[1]     );
and     a5( d[2], sel[2], s_bar[1], sel[0], en );
and     a6( d[4], s_bar[2], sel[1], sel[0], en );
```

Golden circuit:

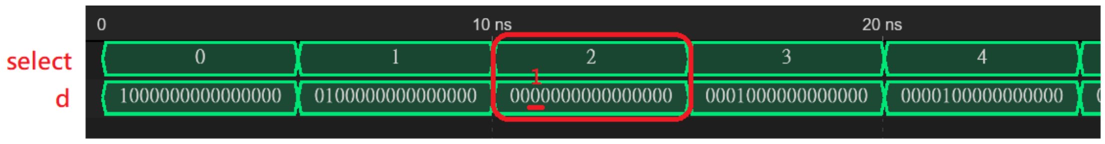
將 s_bar[3]改成 sel[1]即可使電路功能正常。



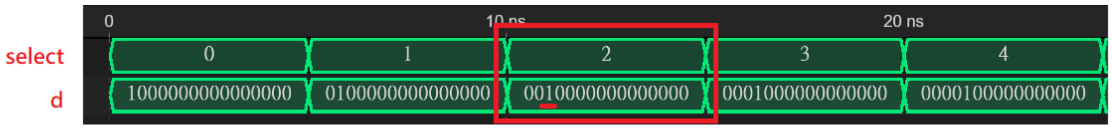
```
and     a0( d[7], s_bar[2], s_bar[1], s_bar[0], en );
and     a1( d[5], s_bar[2], sel[1], s_bar[0], en );
and     a2( d[6], s_bar[2], s_bar[1], sel[0], en );
not     n3( s_bar[0], sel[0]     );
and     a7( d[0], sel[2], sel[1], sel[0], en );
and     a3( d[3], sel[2], s_bar[1], s_bar[0], en );
and     a4( d[1], sel[2], sel[1], s_bar[0], en );
not     n2( s_bar[1], sel[1]     );
and     a5( d[2], sel[2], s_bar[1], sel[0], en );
and     a6( d[4], s_bar[2], sel[1], sel[0], en );
```

result:

1. **Infected circuit:**

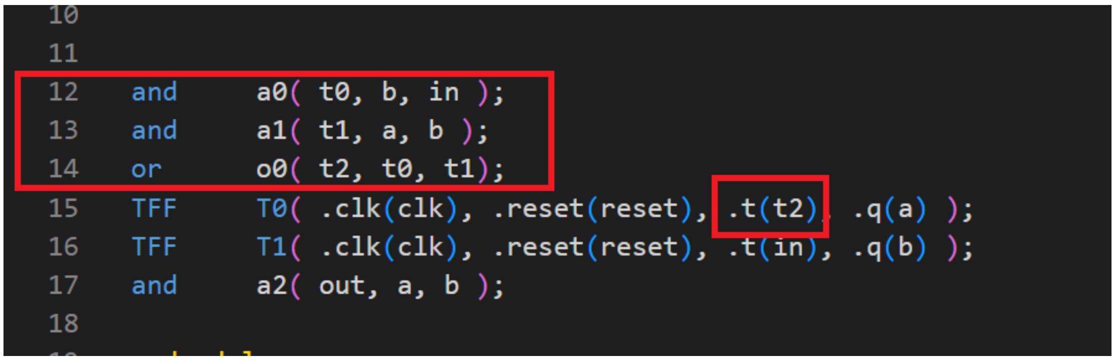


2. **Golden circuit:**



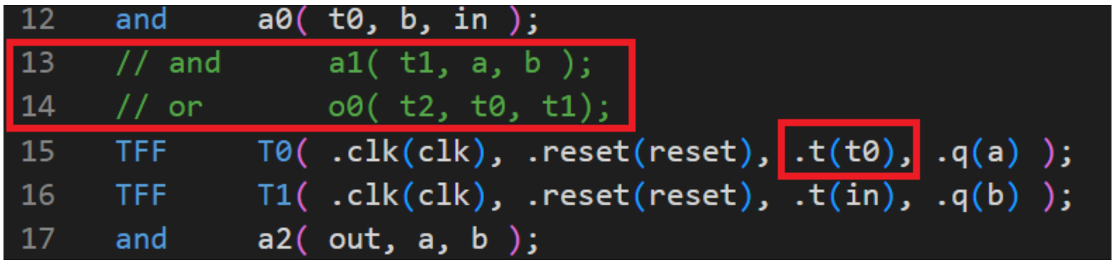
Case3

Verilog Code Trojans:



```
and     a0( t0, b, in );
and     a1( t1, a, b );
or      o0( t2, t0, t1);
TFF     T0( .clk(clk), .reset(reset), .t(t2), .q(a) );
TFF     T1( .clk(clk), .reset(reset), .t(in), .q(b) );
and     a2( out, a, b );
```

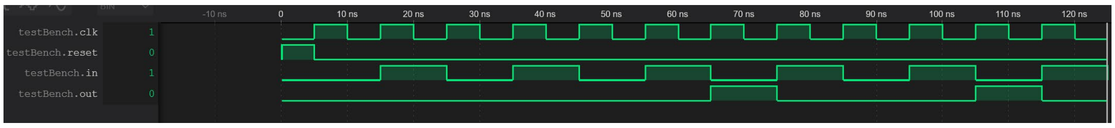
Golden circuit:



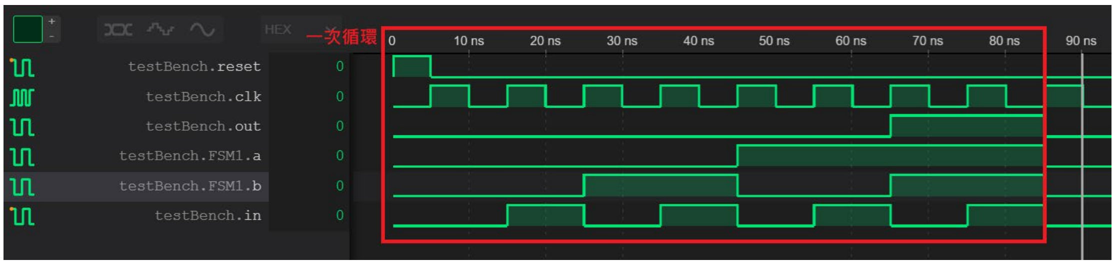
```
and     a0( t0, b, in );
// and     a1( t1, a, b );
// or      o0( t2, t0, t1);
TFF     T0( .clk(clk), .reset(reset), .t(t0), .q(a) );
TFF     T1( .clk(clk), .reset(reset), .t(in), .q(b) );
and     a2( out, a, b );
```

result:

1. Infected circuit:

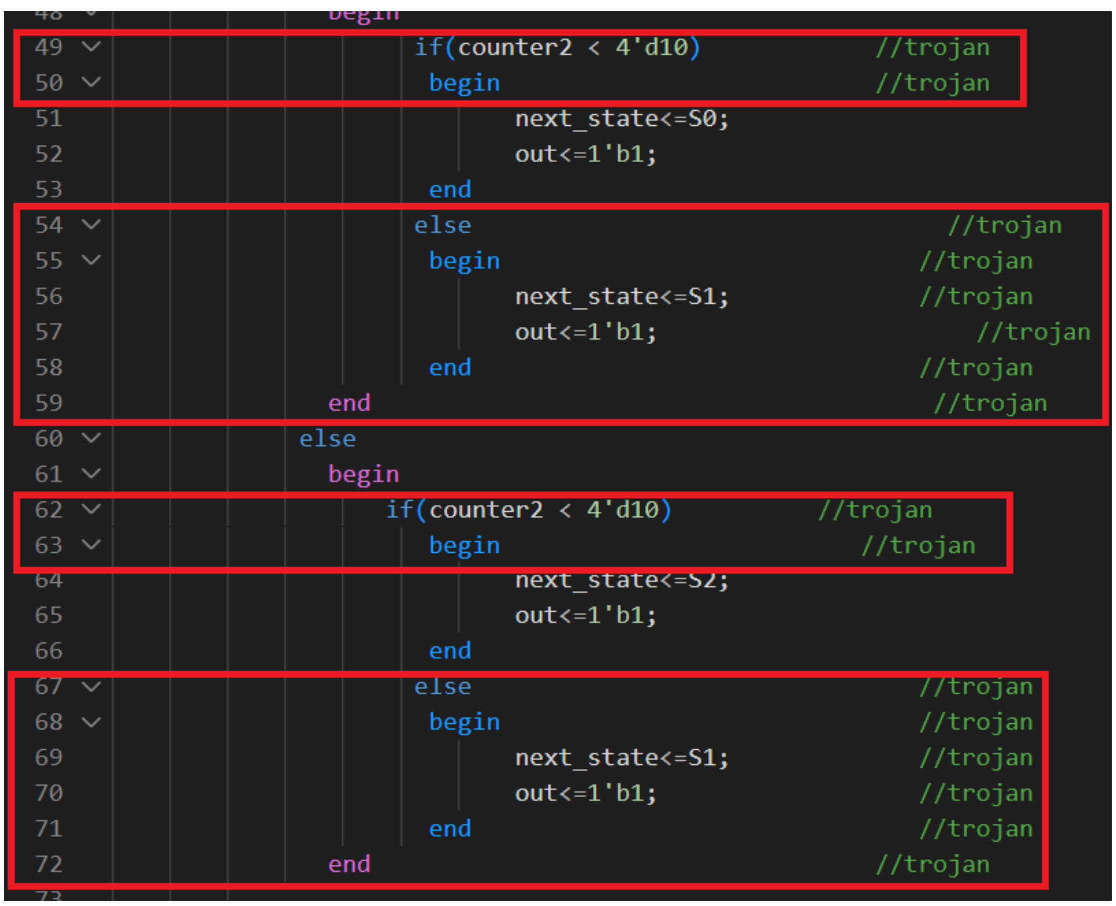


2. Golden circuit:



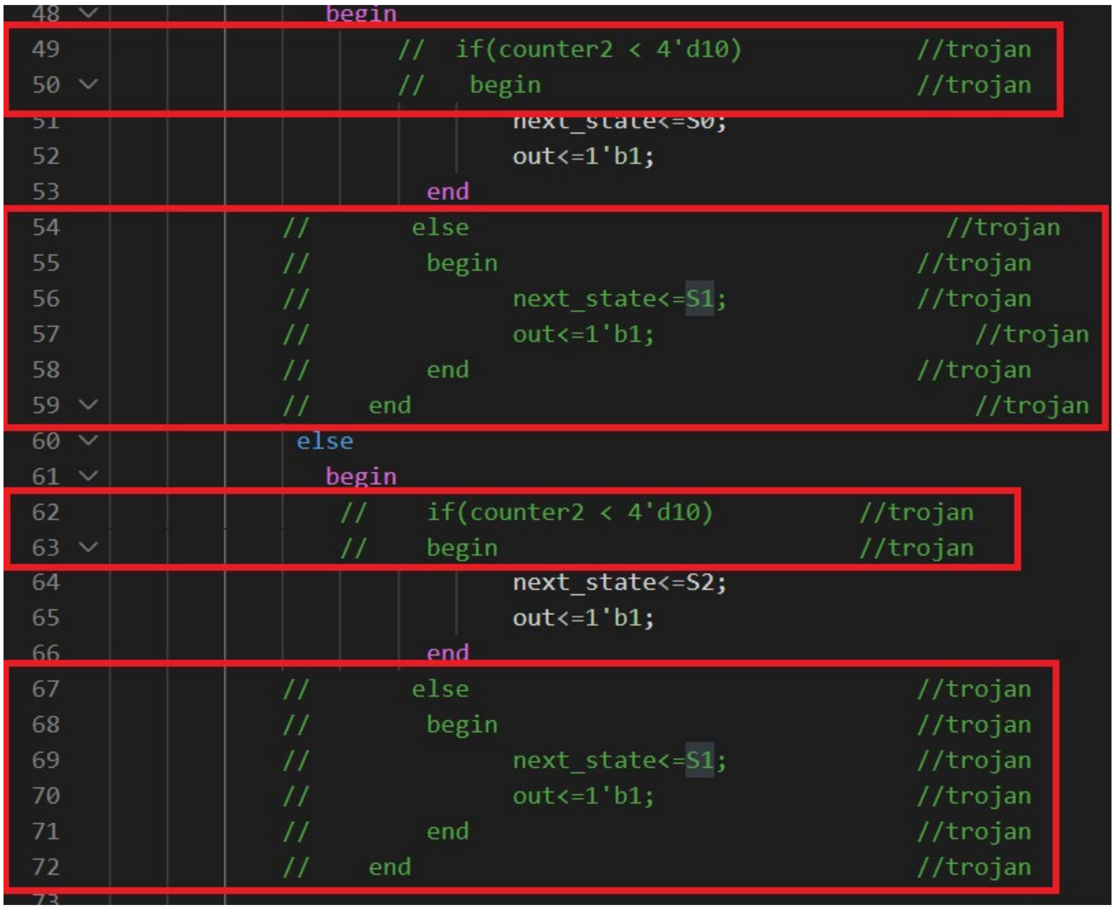
Case4

Verilog Code Trojans:



```
               begin
                     if(counter2 < 4'd10)            //trojan
                      begin                          //trojan
                            next_state<=S0;
                            out<=1'b1;
                      end
                     else                                 //trojan
                      begin                             //trojan
                            next_state<=S1;             //trojan
                            out<=1'b1;                      //trojan
                      end                               //trojan
               end                                       //trojan
             else
               begin
                   if(counter2 < 4'd10)          //trojan
                      begin                         //trojan
                            next_state<=S2;
                            out<=1'b1;
                      end
                     else                               //trojan
                      begin                             //trojan
                            next_state<=S1;             //trojan
                            out<=1'b1;                  //trojan
                      end                               //trojan
               end                                   //trojan
```

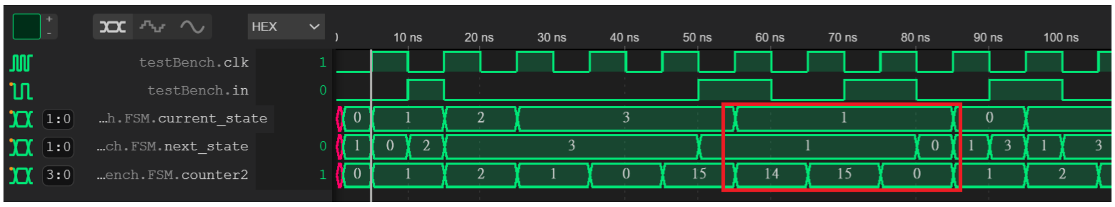
Golden circuit:



```
                begin
                    //  if(counter2 < 4'd10)            //trojan
                    //   begin                          //trojan
                            next_state<=S0;
                            out<=1'b1;
                        end
                //      else                                //trojan
                //       begin                          //trojan
                //              next_state<=S1;         //trojan
                //              out<=1'b1;                  //trojan
                //       end                            //trojan
                //    end                                   //trojan
            else
                begin
                    //     if(counter2 < 4'd10)         //trojan
                    //     begin                        //trojan
                            next_state<=S2;
                            out<=1'b1;
                        end
                //      else                            //trojan
                //       begin                          //trojan
                //              next_state<=S1;         //trojan
                //              out<=1'b1;              //trojan
                //       end                            //trojan
                //    end                               //trojan
```

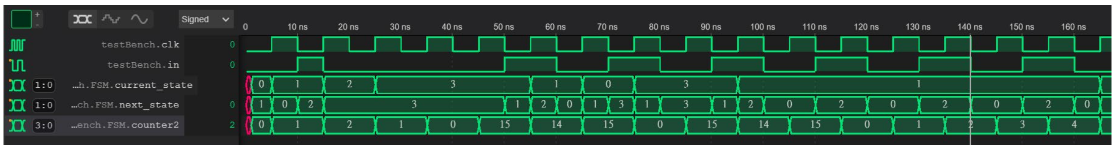
result:

1. Infected circuit:



在 Normal Mode 的階段，current_state=1，next_state 只能=0 or 2，由訊號圖紅框框的訊號可看出此時出現錯誤。

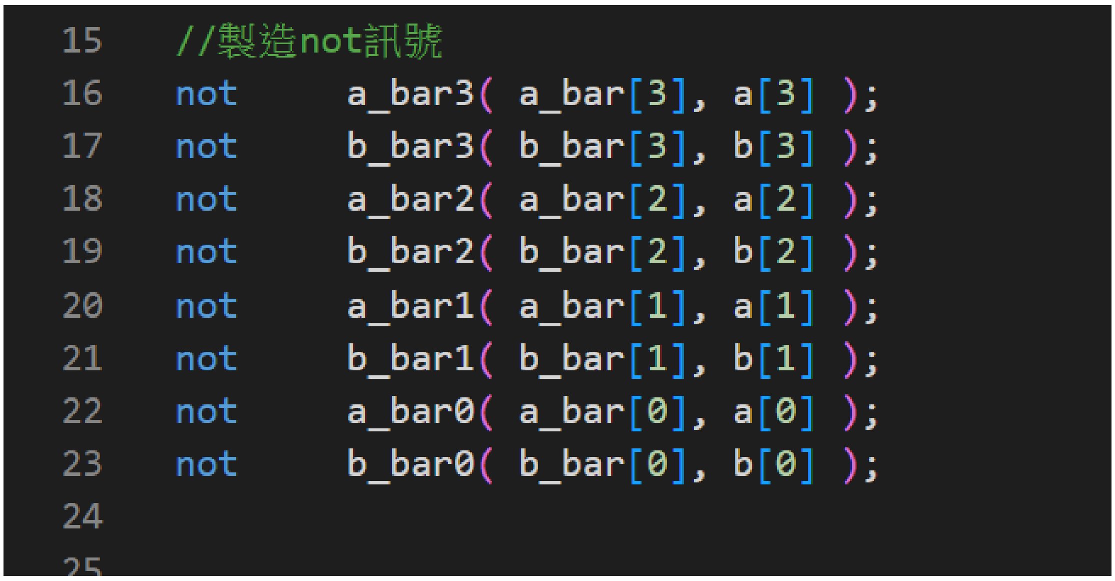
2. Golden circuit:



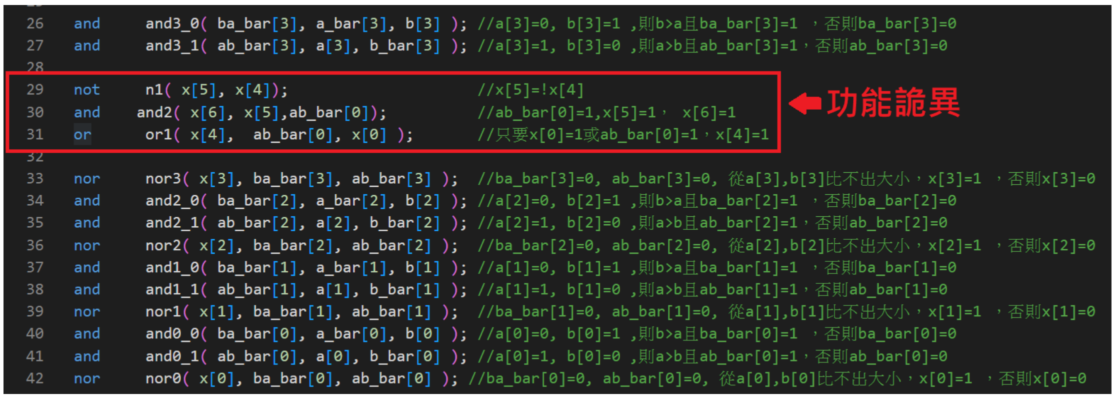
Current_state 與 Next_state 在 nomal mode 階段不會再出現問題。

二、我的方法

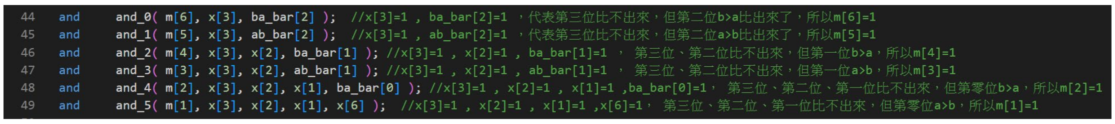
如下圖，將 code 一行一行註解其功能，發現有功能詭異且不必要之程式碼。



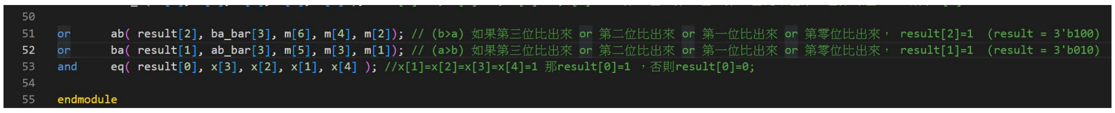
```
//製造not訊號
not     a_bar3( a_bar[3], a[3] );
not     b_bar3( b_bar[3], b[3] );
not     a_bar2( a_bar[2], a[2] );
not     b_bar2( b_bar[2], b[2] );
not     a_bar1( a_bar[1], a[1] );
not     b_bar1( b_bar[1], b[1] );
not     a_bar0( a_bar[0], a[0] );
not     b_bar0( b_bar[0], b[0] );


```



```
and     and3_0( ba_bar[3], a_bar[3], b[3] ); //a[3]=0, b[3]=1 ,則b>a且ba_bar[3]=1 ，否則ba_bar[3]=0
and     and3_1( ab_bar[3], a[3], b_bar[3] ); //a[3]=1, b[3]=0 ,則a>b且ab_bar[3]=1，否則ab_bar[3]=0

not     n1( x[5], x[4]);                    //x[5]=!x[4]
and    and2( x[6], x[5],ab_bar[0]);         //ab_bar[0]=1,x[5]=1， x[6]=1
or      or1( x[4],  ab_bar[0], x[0] );      //只要x[0]=1或ab_bar[0]=1，x[4]=1

nor     nor3( x[3], ba_bar[3], ab_bar[3] );  //ba_bar[3]=0, ab_bar[3]=0, 從a[3],b[3]比不出大小，x[3]=1 ，否則x[3]=0
and     and2_0( ba_bar[2], a_bar[2], b[2] ); //a[2]=0, b[2]=1 ,則b>a且ba_bar[2]=1 ，否則ba_bar[2]=0
and     and2_1( ab_bar[2], a[2], b_bar[2] ); //a[2]=1, b[2]=0 ,則a>b且ab_bar[2]=1，否則ab_bar[2]=0
nor     nor2( x[2], ba_bar[2], ab_bar[2] );  //ba_bar[2]=0, ab_bar[2]=0, 從a[2],b[2]比不出大小，x[2]=1 ，否則x[2]=0
and     and1_0( ba_bar[1], a_bar[1], b[1] ); //a[1]=0, b[1]=1 ,則b>a且ba_bar[1]=1 ，否則ba_bar[1]=0
and     and1_1( ab_bar[1], a[1], b_bar[1] ); //a[1]=1, b[1]=0 ,則a>b且ab_bar[1]=1，否則ab_bar[1]=0
nor     nor1( x[1], ba_bar[1], ab_bar[1] );  //ba_bar[1]=0, ab_bar[1]=0, 從a[1],b[1]比不出大小，x[1]=1 ，否則x[1]=0
and     and0_0( ba_bar[0], a_bar[0], b[0] ); //a[0]=0, b[0]=1 ,則b>a且ba_bar[0]=1 ，否則ba_bar[0]=0
and     and0_1( ab_bar[0], a[0], b_bar[0] ); //a[0]=1, b[0]=0 ,則a>b且ab_bar[0]=1，否則ab_bar[0]=0
nor     nor0( x[0], ba_bar[0], ab_bar[0] ); //ba_bar[0]=0, ab_bar[0]=0, 從a[0],b[0]比不出大小，x[0]=1 ，否則x[0]=0
```



```
and     and_0( m[6], x[3], ba_bar[2] );  //x[3]=1 , ba_bar[2]=1 ，代表第三位比不出來，但第二位b>a比出來了，所以m[6]=1
and     and_1( m[5], x[3], ab_bar[2] );  //x[3]=1 , ab_bar[2]=1 ，代表第三位比不出來，但第二位a>b比出來了，所以m[5]=1
and     and_2( m[4], x[3], x[2], ba_bar[1] ); //x[3]=1 , x[2]=1 , ba_bar[1]=1 ， 第三位、第二位比不出來，但第一位b>a，所以m[4]=1
and     and_3( m[3], x[3], x[2], ab_bar[1] ); //x[3]=1 , x[2]=1 , ab_bar[1]=1 ， 第三位、第二位比不出來，但第一位a>b，所以m[3]=1
and     and_4( m[2], x[3], x[2], x[1], ba_bar[0] ); //x[3]=1 , x[2]=1 , x[1]=1 ,ba_bar[0]=1， 第三位、第二位、第一位比不出來，但第零位b>a，所以m[2]=1
and     and_5( m[1], x[3], x[2], x[1], x[6] );  //x[3]=1 , x[2]=1 , x[1]=1 ,x[6]=1， 第三位、第二位、第一位比不出來，但第零位a>b，所以m[1]=1
```



```

or      ab( result[2], ba_bar[3], m[6], m[4], m[2]); // (b>a) 如果第三位比出來 or 第二位比出來 or 第一位比出來 or 第零位比出來， result[2]=1  (result = 3'b100)
or      ba( result[1], ab_bar[3], m[5], m[3], m[1]); // (a>b) 如果第三位比出來 or 第二位比出來 or 第一位比出來 or 第零位比出來， result[1]=1  (result = 3'b010)
and     eq( result[0], x[3], x[2], x[1], x[4] ); //x[1]=x[2]=x[3]=x[4]=1 那result[0]=1 ，否則result[0]=0;

endmodule
```

三、本次作業的困難及如何克服

這次作業最困難的地方無疑是找出到底是哪幾行 code 為 trojan，如果只是丟 test patterns 進去測試，只能達到檢測是否有錯誤存在的功能，無法明確的達到診斷(diagonose)，不過把錯誤的 response 和 golden response 去做比較，可以從中發現一些蛛絲馬跡，最後再從這些線索去檢查原始 code，找出是哪些 code 功能詭異且非必要。

備註: TestBench 會以 verilog 檔另外上傳。