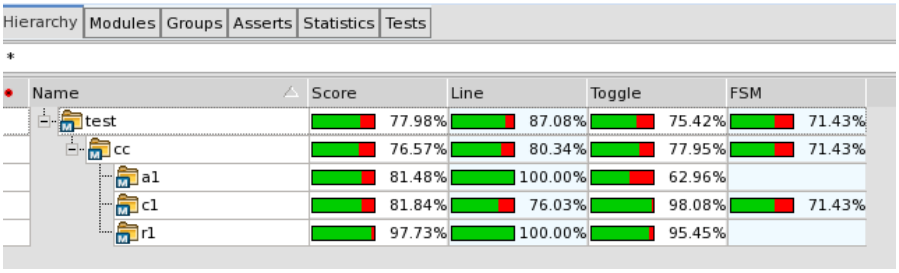
HW3 Coverage Analysis

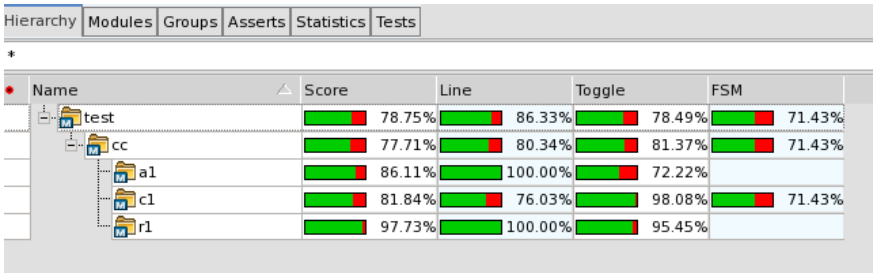
311510182 李旻臻

1. Why coverage rate can't reach 100% in problem 1(with coverage rate pictures)



因為電路有 BUG，所以導致一些 state 會進不去，導致部分 block、statement 會跑不到，且因為 pattern 是 random 產生，所以可能會因為 pattern 不夠豐富，導致 line、toggle 覆蓋不到。若要達到 100% coverage rate 需要電路正常且 pattern 豐富。

2. Analyze problem 2 results (with coverage rate pictures)



在刪除掉 5 個 pattern 後可看到%數提升，明顯改變的值多為 toggle，fsm 可能因為 cpu.v 裡有 bug，導致一些 state 根本不可能跑進去，所以數值不動，但 toggle 可以透過變換 pattern 的排序或是減少數量來改變 rate，因在不同的 pattern 的排序裡，每階段的 A,B,C 值皆不同，故 toggle 的覆蓋程度一定不一樣。而我減少了這 5 個 pattern 剛好導致 01 互換成功，toggle 覆蓋率上升。

3. Where is the bug in cpu_bug.v and how you correct it

```
always@(alu_out or in or mem_out or reg_c or sel)begin
    if (sel == DATA_IN)
        write_data = in;
    else if (sel == ALU_OUT)
        write_data = alu_out;
    else if (sel == MEM_OUT) //bug
        write_data = mem_out;
    else if(sel == REG_C_O) //bug
        write_data = reg_c;
    else //bug
        write_data = write_data;
end
```

↑(1)

```
if(ready)begin
    if (w_reg_c)
        reg_c = write_data;
    else if (w_reg_a)
        reg_a = write_data;
    else if (w_reg_b)
        reg_b = write_data;
    else if (w_out) //bug
        out = write_data;
    else
        out = 0;
end
```

↑(2)

```
always@(posedge clk)begin
    if (!reset)
        current_state = next_state;
    else
        current_state = ST_0;//bug
end
```

↑(3)

```
4'b0100:begin
        w_out=0; w_reg_a=0; w_reg_b=0; w_reg_c=1; w_mem=0; //bug
        sel=ALU_OUT; next_state=ST_0;
        end
4'b0101:begin
        w_out=0; w_reg_a=0; w_reg_b=0; w_reg_c=1; w_mem=0; //bug
        sel=ALU_OUT; next_state=ST_0;
        end
```

←(4)

```
4'b0111:begin
        w_out=0; w_reg_a=0; w_reg_b=1; w_reg_c=0; w_mem=0;
        sel=DATA_IN; next_state=ST_1;//bug
        end
4'b1000:begin
        w_out=0; w_reg_a=0; w_reg_b=0; w_reg_c=0; w_mem=1;
        sel=DATA_IN; next_state=ST_1;
        end
4'b1001:begin
        w_out=0; w_reg_a=0; w_reg_b=0; w_reg_c=0; w_mem=1;
        sel=REG_C_O; next_state=ST_0;//bug
        end
4'b1010:begin
        w_out=0; w_reg_a=0; w_reg_b=0; w_reg_c=0; w_mem=0;
        sel=REG_C_O; next_state=ST_2;//bug
        end
4'b1011:begin
        w_out=1; w_reg_a=0; w_reg_b=0; w_reg_c=0; w_mem=0;
        sel=REG_C_O; next_state=ST_0;//bug
        end
4'b1100:begin
        w_out=0; w_reg_a=0; w_reg_b=0; w_reg_c=0; w_mem=0;
        sel=MEM_OUT; next_state=ST_3;//bug
        end
4'b1101:begin
        w_out=0; w_reg_a=1; w_reg_b=0; w_reg_c=0; w_mem=0;
        sel=REG_C_O; next_state=ST_0;//bug
        end
4'b1110:begin
        w_out=0; w_reg_a=0; w_reg_b=1; w_reg_c=0; w_mem=0;
        sel=REG_C_O; next_state=ST_0;//bug
        end
```

↑(5)

```
ST_2 : begin
        ready = 0; addr = 0;
        w_out = 0; w_reg_a = 0; w_reg_b = 0; w_reg_c = 1; w_mem = 0;
        sel = MEM_OUT; next_state = ST_0;//bug
        end
ST_3 : begin
        ready = 0; addr = 0;
        w_out = 1; w_reg_a = 0; w_reg_b = 0; w_reg_c = 0; w_mem = 0;
        sel = MEM_OUT; next_state = ST_0;//bug
        end
```

↑(6)

```
always@(reg_a or reg_b or sel_fun)begin
    case (sel_fun)
    3'b000 : alu_out = reg_a + reg_b;//bug
    3'b001 : alu_out = reg_a - reg_b;//bug
    3'b010 : alu_out = reg_a + 1;
    3'b011 : alu_out = reg_a - 1;
    3'b100 : alu_out = reg_a + reg_b + 1;//bug
    3'b101 : alu_out = ~reg_a;//bug
    default: alu_out = 8'bx;
    endcase
end
```

←(7)

(1) 未寫出所有 sel 的可能，導致值無法儲存

(2) 未寫出 w_out 導致無法 output

(3) else 裡的 current state 寫錯，導致初始的 state 在錯誤的 state

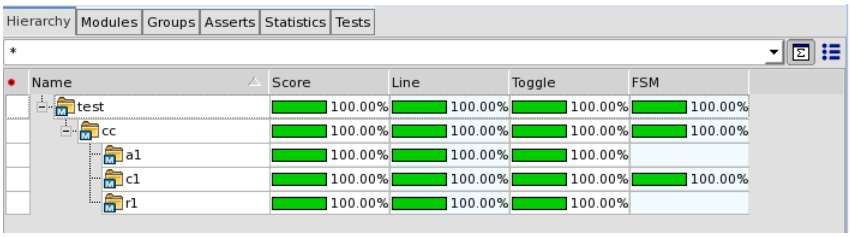
(4) Control 裡的 case 寫錯，可能導致值存到錯誤的地方，或者因為 sel 跟 state 錯誤，導致根本無法計算

(5) 同上述(4)

(6) 再 state2,3 裡的 next state 寫錯，導致無法回到原來初始 state，而跑到 state1，可能導致讀錯指令

(7) Alu 裡計算錯誤，對 cpu 運作不影響，但會導致 output 出的值是錯的，也會導致再設計 100% coverage rate pattern 時判斷有誤

4. <u>What did you do to increase the coverage rate (with 100% coverage rate pictures)</u>



我自己是先把所有的指令都先打上去，再慢慢排列，但因為還要顧及 toggle 的%數，所以一定會有指令是重複的，所以要盡可能地去拼湊出兩個一樣的指令但 output 的結果可以覆蓋到每個

bit 都有 0,1，基本上就還是不斷的思考跟嘗試。

5. What did you learn in this project

在這次作業中學會了如何使用此 tool，並運用它去 debug 電路，透過這個 tool 可以輕易地發現 FSM 有問題，且用這個 tool 可以輕易的設計 pattern，那其完整覆蓋整個 code，不過這個 tool 跟 nWave 一樣爛，真的很容易變成閃屏，我都還要想辦法跟它拚手速…