311591015 葉憶嫻

CAD HW3 MiniReport

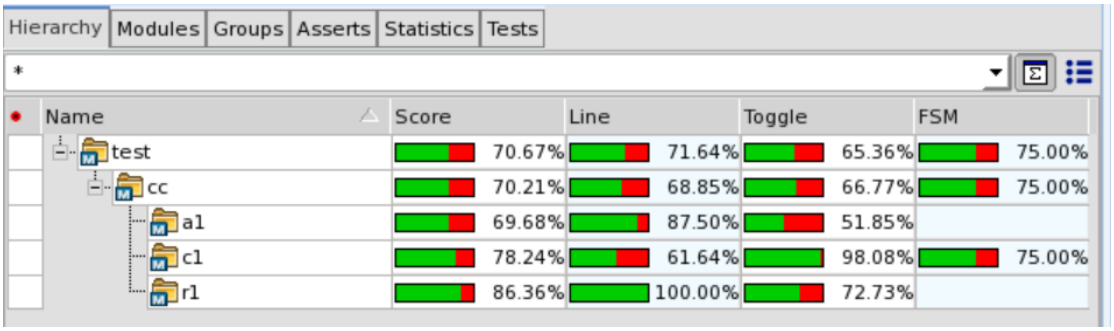
1. Generate 20 patterns randomly (test_random.v) to analyze coverage

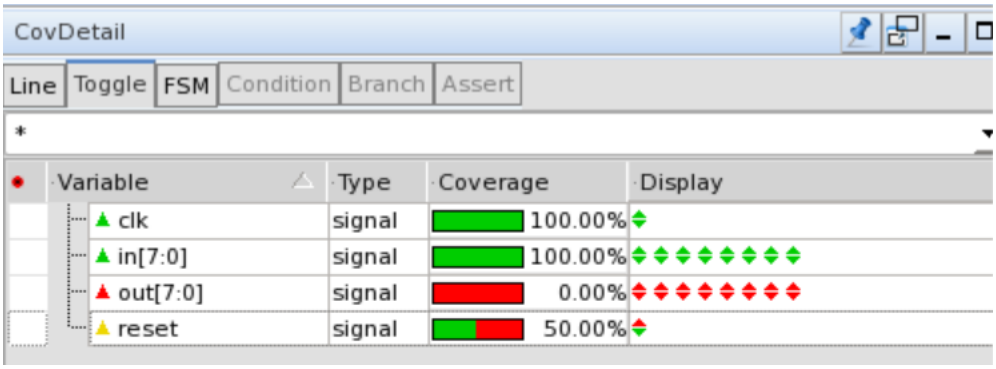
I. why coverage rate can't reach 100%

使用$random( )功能，並寫一個 for 迴圈來產生出 20 組 pattern。但是因為 input 的數值為隨機亂數產生的，所以無法保證生成的 pattern 沒有重複的情形。

故可以分析出沒有 100% coverage rate 的原因，就是因為隨機產生的 20 組 pattern 並沒有包括到全部的 15 種 opcode。

II. coverage rate pictures



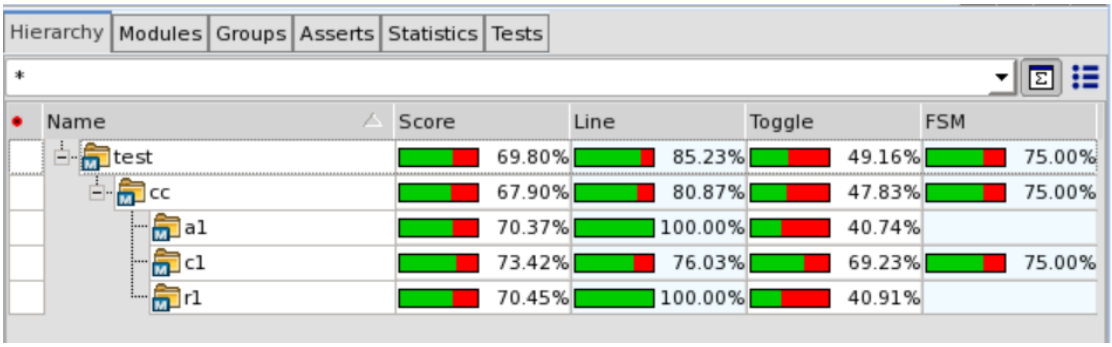


2. Base on CPU operation, create some patterns to test the original cpu_bug.v with coverage analysis. Can you use less number of test patterns than random approach?

I. Analyze results

使用 opcode 規則去生成隨機的 pattern，並刪到剩下 15 組，可能是因為 test pattern 不夠豐富，coverage rate 的數據不是很好。

II. coverage rate pictures



3. Find the bugs in the cpu_bug.v (if any) and correct them

總共有 20 個需要修改的 bug。

- Bug 1, 2

```
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;
end
```

correction

```
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 == REG_C_O)
       write_data = reg_c;
    else
        write_data = mem_out;
```

- Bug 3

```
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
        out = 0;
end
```

correction

```
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)
        out = write_data;
    else
        out = 0;
end
```

- Bug 4

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

correction

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

- Bug 5

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

correction

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

- Bug 6

```
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_0;
        end
```

correction

```
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; //
        end
```

- Bug 7

```
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_1;
        end
```

correction

```
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;//
        end
```

- Bug 8

```
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_3;
        end
```

correction

```
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;//
        end
```

- Bug 9

```
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_2;
        end
```

correction

```
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;//
        end
```

- Bug 10

```
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_3;
        end
```

correction

```
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; //
        end
```

- Bug 11

```
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_1;
        end
```

correction

```
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; //
        end
```

- Bug 12

```
default:begin
        w_out=0; w_reg_a=0; w_reg_b=0; w_reg_c=0; w_mem=1;
        sel=ALU_OUT; next_state=ST_0;
        end
```

correction

```
default:begin
        w_out=0; w_reg_a=0; w_reg_b=0; w_reg_c=0; w_mem=0;
        sel=ALU_OUT; next_state=ST_0;
        end
```

- Bug 13, 14

```
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_1;
        end
```

correction

```
ST_2 : begin
        ready = 1; 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; //
        end
```

- Bug 15, 16

```
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_2;
        end
```

correction

```
ST_3 : begin
        ready = 1; 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; //
        end
```

- Bug 17, 18, 19 20

```
3'b000 : alu_out = reg_a - reg_b;
3'b001 : alu_out = reg_a + reg_b;
3'b010 : alu_out = reg_a + 1;
3'b011 : alu_out = reg_a - 1;
3'b100 : alu_out = reg_a + reg_b - 1;
3'b101 : alu_out = reg_a;
```

correction

```
3'b000 : alu_out = reg_a + reg_b;//
3'b001 : alu_out = reg_a - reg_b;//
3'b010 : alu_out = reg_a + 1;
3'b011 : alu_out = reg_a - 1;
3'b100 : alu_out = reg_a + reg_b + 1;//
3'b101 : alu_out = ~reg_a;//
```

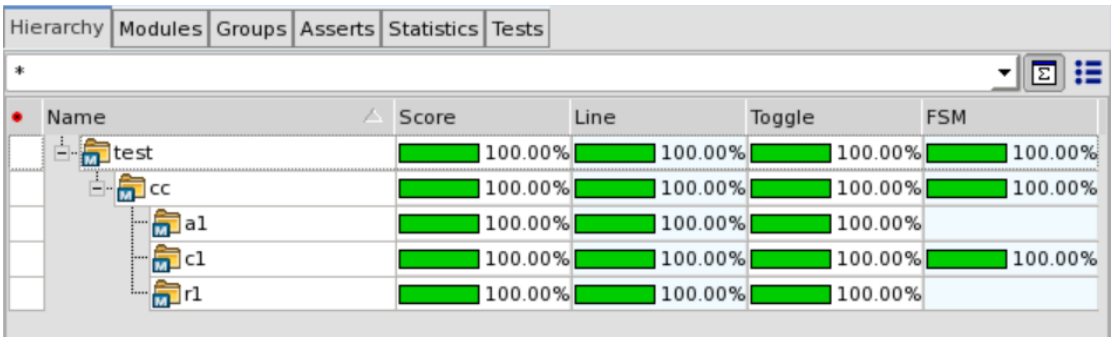
4. Improve your test.v to increase the coverage rate to 100%, no restriction about the number of pattern

I. increase the coverage rate

我設計 pattern 的方法是將所有的 operate code 都執行過一次，然後將 0 儲存到 register 中，來增加 toggle rate。

接著，會再把 1 存入先前的那個 register 中，最後一步操作則是將 0 存入 register。這樣一來那個 register 就會完整地歷經從 1 到 0 再從 0 變化到 1 的過程，就能達到 100% coverage rate。

II. 100% coverage rate pictures



5. What did you learn in this project

這次的作業讓我學到如何利用 debug tool 去查看出現紅色標底的程式碼，是否有 ALU 邏輯錯誤、FSM state 轉換錯誤、旗標設定錯誤，一個一個修正並使用 Verdi 測試 coverage。

也了解到設計一個好的 pattern 的重要性，這次作業能實際體會到在做 verification 時用 random pattern 和設計過的 pattern，跑出來的結果差異會非常大。有計畫性地生成 pattern，效率會更好。因為通過針對不同 case 來設計能避免重複的問題，即可達到 100% coverage。