

# 常見語法差異

Verilog 與 C 語言

# Verilog 實現類似 for loop 的功能

```
12 // Verilog
13 reg [31:0] cnt;
14 always @(posedge clk or negedge rst_n) begin
15     if(!rst_n) begin
16         cnt <= 32'b0;
17     else if(cnt < 32'd64)
18         cnt <= cnt + 32'b1;
19 end
```

```
3 // C
4 int i;
5 int cnt = 0;
6 for(i=0; i<64; i++) {
7     cnt = cnt + 1;
8 }
```

- 只要 Simulator (波形模擬) 沒有停止，則 always block 內的程式碼永遠都會進行；意指硬體語言沒有所謂「執行結束」的概念，因為硬體會一直存在於你的電路中。
- result: cnt = 63
- 程序執行結束後，記憶體從系統中被釋放。
- result: cnt = 63

# "if" 在 C program 與 Verilog 上的差異

```
39 // Verilog
40 reg [3:0] a;
41 reg [3:0] b;
42 reg [3:0] c;
43 always @(posedge clk or negedge rst_n) begin
44     if(!rst_n) begin
45         a <= 4'd4;
46         b <= 4'd2;
47         c <= 4'd5;
48     end
49     else begin
50         if(a > b)
51             a <= 4'd6;
52         if(a > c)
53             c <= 4'd9;
54     end
55 end
```

```
28 // C
29 int a = 4;
30 int b = 2;
31 int c = 5;
32 if(a > b)
33     a = 6;
34 if(a > c)
35     c = 9;
```

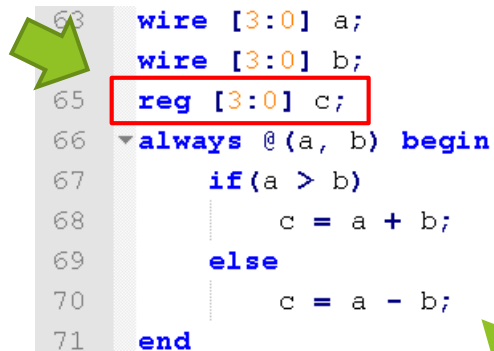
- $a > b$  與  $a > c$  同時進行判斷。
- 經過第一個 cycle 之後，  
result:  $a = 6$ ,  $b = 2$ ,  $c = 5$

- 先判斷  $a > b$ ，再判斷  $a > c$ 。
- 程式執行結束後，  
result:  $a = 6$ ,  $b = 2$ ,  $c = 9$

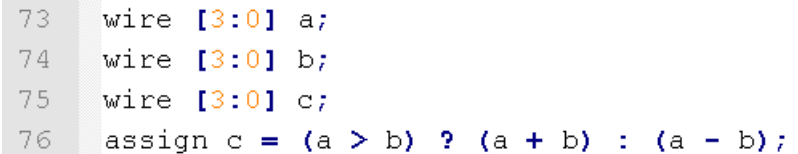
# Verilog 的 behavior 與 data flow 描述

以判斷式描述為例：

注意資料型態



```
63 wire [3:0] a;  
64 wire [3:0] b;  
65 reg [3:0] c;  
66 always @(a, b) begin  
67     if(a > b)  
68         c = a + b;  
69     else  
70         c = a - b;  
71 end
```



```
73 wire [3:0] a;  
74 wire [3:0] b;  
75 wire [3:0] c;  
76 assign c = (a > b) ? (a + b) : (a - b);
```

兩者描述法等效

# Verilog 容易犯錯的寫法

```
89 // Verilog
90 wire [3:0] a, b;
91 reg [3:0] c;
92 always @(a, b) begin
93     c = a + b;
94     c = a - b;
95 end
```

或

```
99 wire [3:0] a, b;
100 wire [3:0] c;
101 assign c = a + b;
102 assign c = a - b;
```

```
80 // c
81 int a = 4, b = 2, c = 0;
82 c = a + b;
83 c = a - b;
```

- c 被 multiple assignment。
- result: c = "unknown"

- 最終只有  $c = a - b$ ; 為有效程式碼。
- result: c = 2