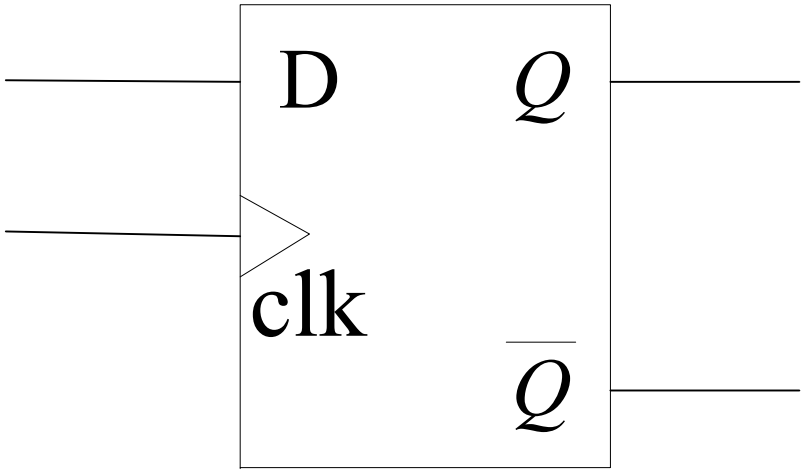
Verilog class 2

D型正反器



D型正反器

C:/verilog_pgm/example/d_ff.v

```
ln #
 1    module d_ff(d, clk, q, q_bar);
 2         input d, clk;
 3         output q, q_bar;
 4         reg q, q_bar;
 5         always@(posedge clk)
 6         begin
 7              q <= d;
 8              q_bar <= ~d;
 9         end
10    endmodule
```

Always Block

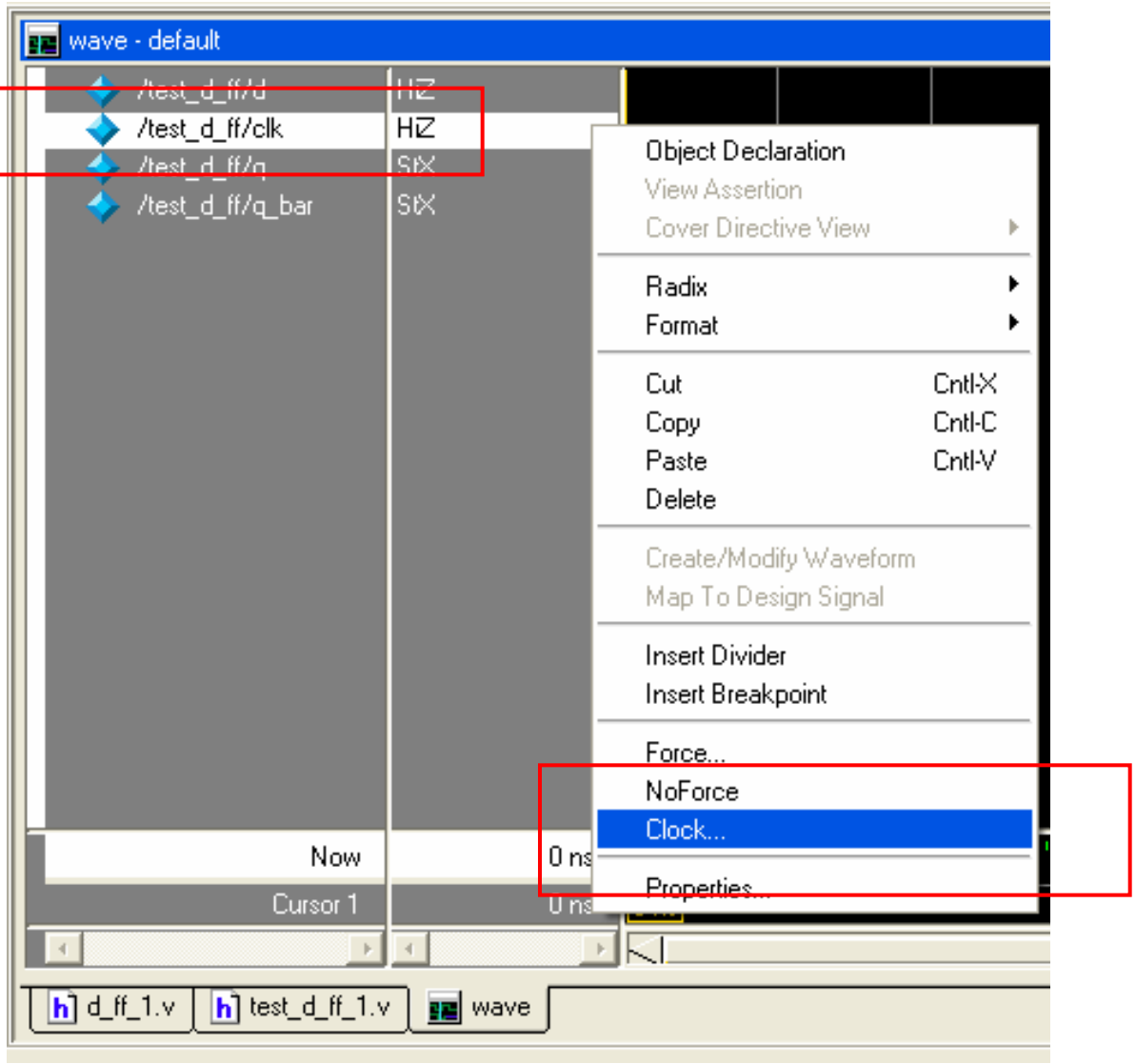
- always blocks ：當觸發訊號啓動後，**always block** 就會執行一次。
- always@(posedge clk)

```
begin
    q <= d;
    q_bar <= ~d;
end
```

觸發訊號**: clk** 的正緣 **(posedge clk)**

clk的負緣 = negedge clk

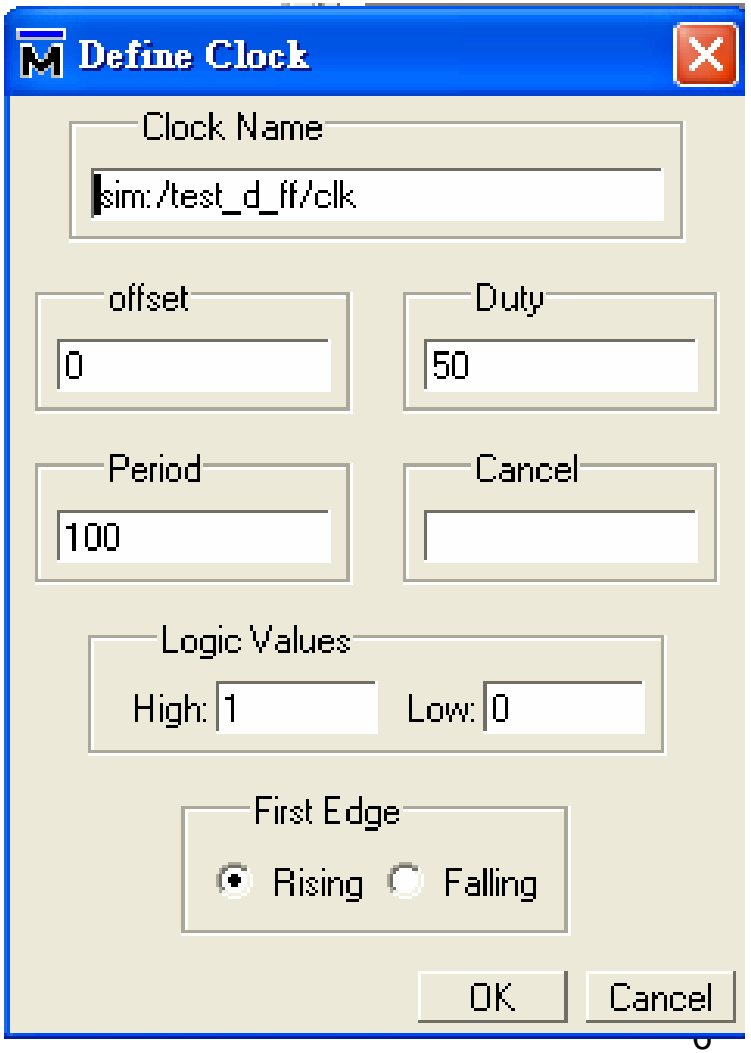
Clock 訊號設定


wave - default
/test_d_ff/clk
HiZ
/test_d_ff/q_bar
StX
Object Declaration
View Assertion
Cover Directive View
Radix
Format
Cut Cntl-X
Copy Cntl-C
Paste Cntl-V
Delete
Create/Modify Waveform
Map To Design Signal
Insert Divider
Insert Breakpoint
Force...
NoForce
Clock...
Properties...
Now
Cursor 1
d_ff_1.v
test_d_ff_1.v
wave

Clock 訊號設定(cont.)

- Period (週期)=100 ns
- Duty = 50

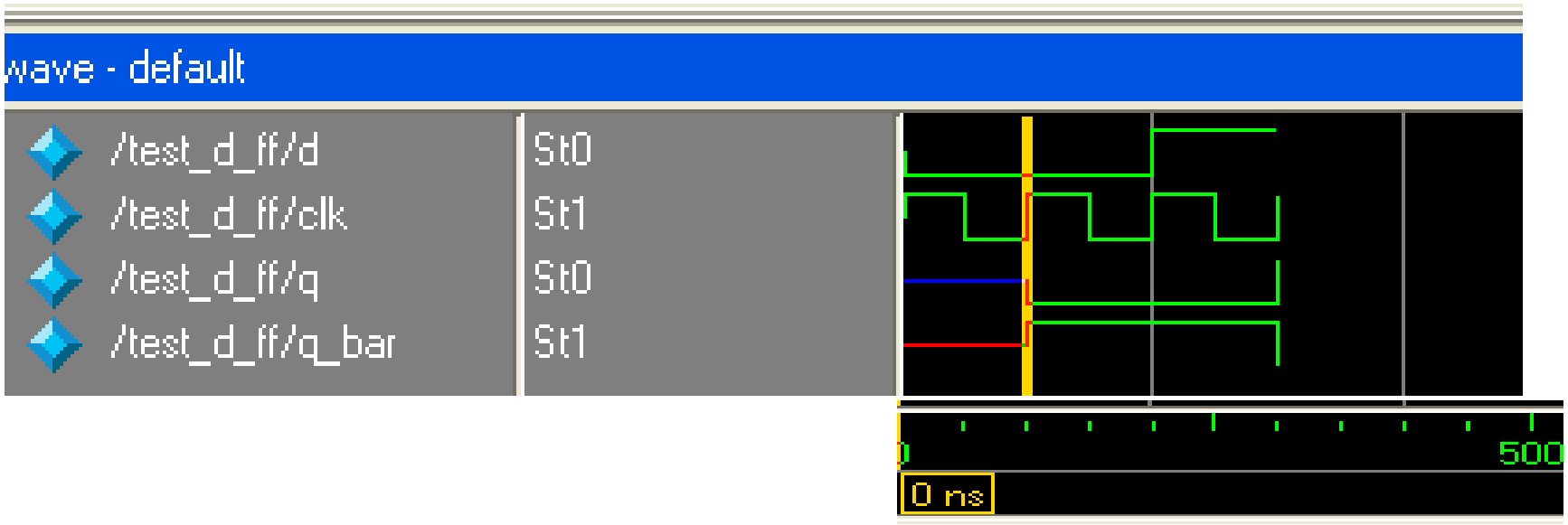
  (logic 1 與 logic 0各佔 50%)



D型正反器測試模組

```
1    include d_ff.v;
2
3    module test_d_ff(d, clk, q, q_bar);
4        input d, clk;
5        output q, q_bar;
6        d_ff u1(d, clk, q, q_bar);
7    endmodule
8
9
10
```

D型正反器


wave - default
/test_d_ff/d
/test_d_ff/clk
/test_d_ff/q
/test_d_ff/q_bar
St0
St1
St0
St1
500
0 ns

HW#3

- 製作一個負緣觸發的D型正反器，與其測試模組 test_d_ff
- 使用Simulation驗證電路正確

循序邏輯

- 顧名思意，依序執行邏輯運算
- 常見的方塊有： if, if_else, if_else_if, case, while_loop, for_loop.

if statement

```
always @ (觸發訊號)
begin
    if ( 條件  ) begin
        執行邏輯運算
    end
end
```

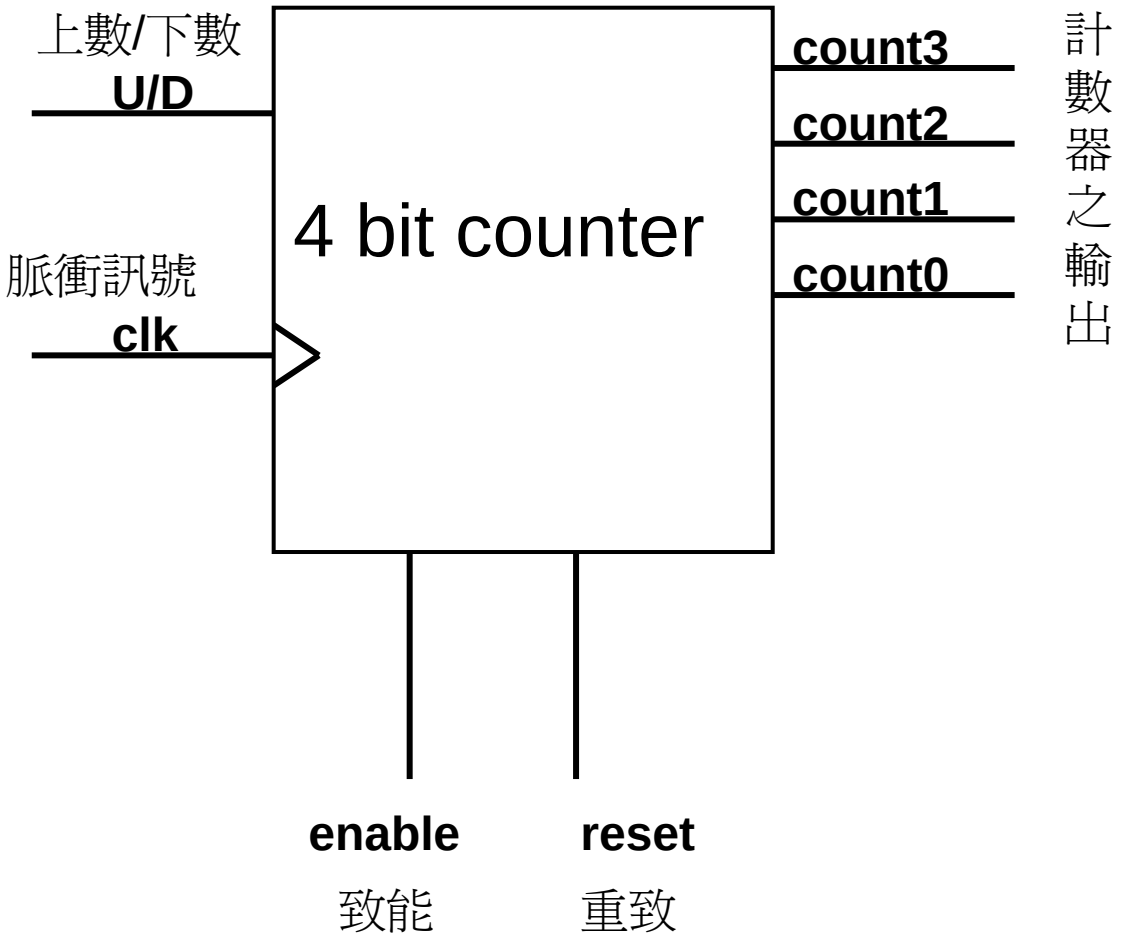
if _else statement

```
always @ (觸發訊號)
begin
   if ( 條件 ) begin
      執行邏輯運算1
   end  else begin
      執行邏輯運算2
   end
end
```

if _else_if statement

```
always @ (觸發訊號)
begin
   if ( 條件 1  ) begin
      執行邏輯運算1
   end  else if (條件 2) begin
      執行邏輯運算2
   end
end
```

4 bit counter



4 bit counter (Cont.)

C:/verilog_pgm/example/four_bit_counter.v

```
ln #
 1    module four_bit_counter (clk, reset, enable, U_D, count);
 2        input clk, reset, enable, U_D;
 3        output[3:0] count;
 4        reg[3:0] count;
 5        always@(posedge clk)
 6        begin
 7            if(reset == 1'b1) begin
 8                count <= 4'b0000;
 9            end else if( enable == 1'b1 && U_D == 1 ) begin
10                count <= count + 1;
11            end else if( enable == 1'b1 && U_D == 0 ) begin
12                count <= count - 1;
13            end
14        end
15    endmodule
```

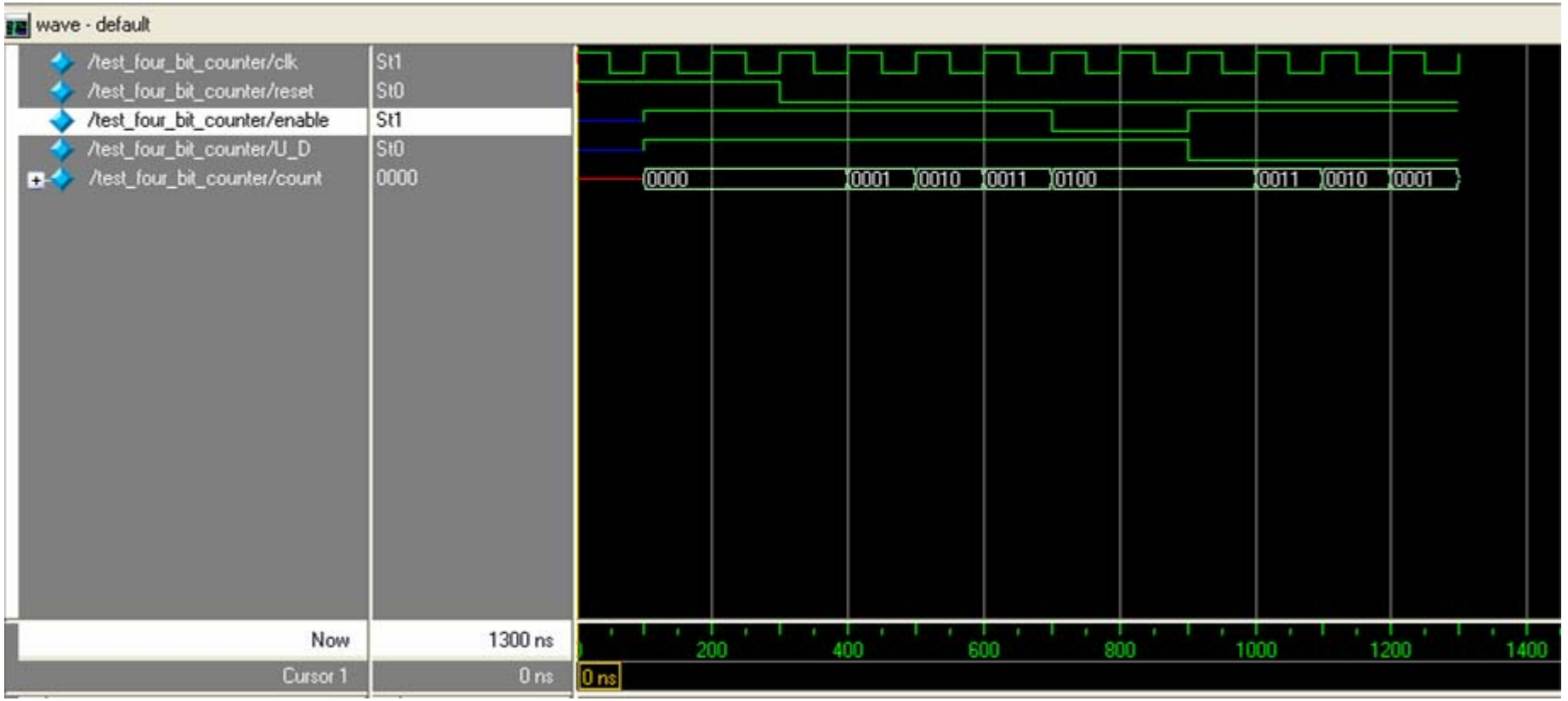
4 bit counter (Cont.)

- 測試模組 test_four_bit_counter

C:/verilog_pgm/example/test_four_bit_counter.v

```
ln #
1    include four_bit_counter.v;
2    module test_four_bit_counter (clk, reset, enable, U_D, count);
3        input clk, reset, enable, U_D;
4        output[3:0] count;
5        four_bit_counter u1 (clk, reset, enable, U_D, count);
6    endmodule
```

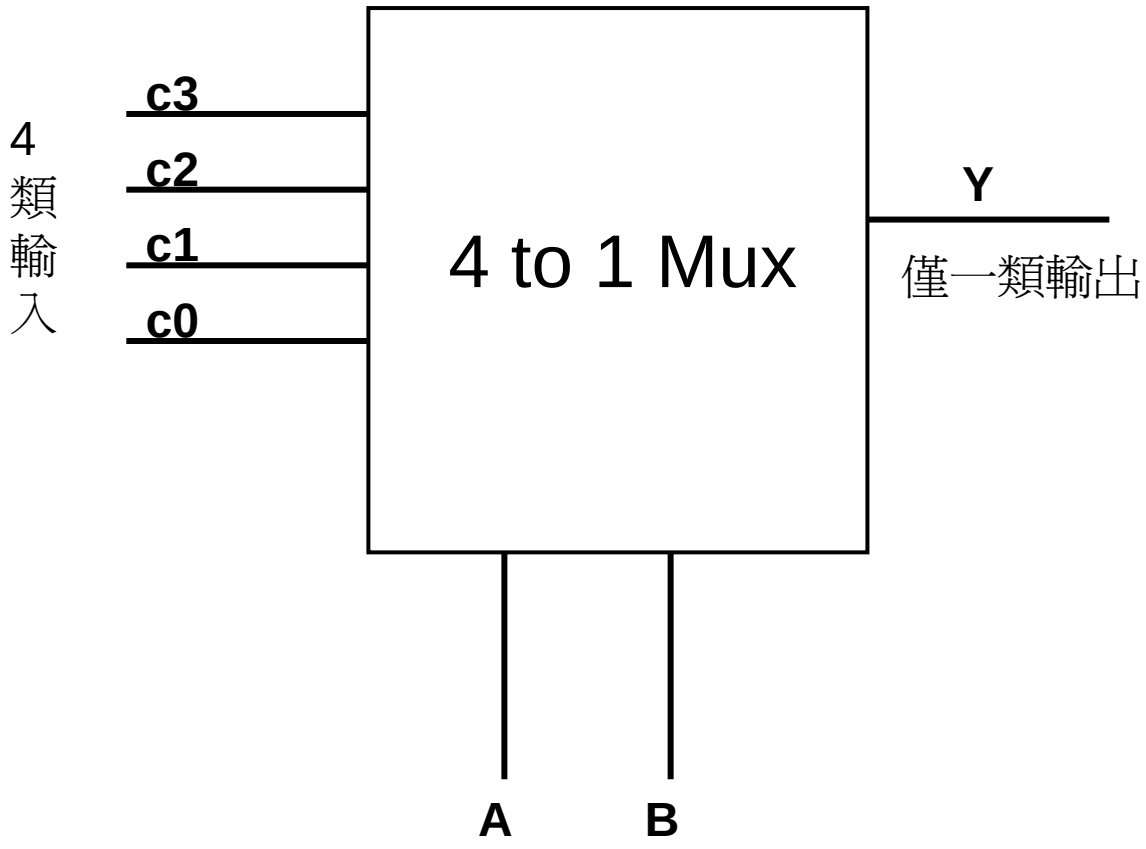
4 bit counter (Cont.)



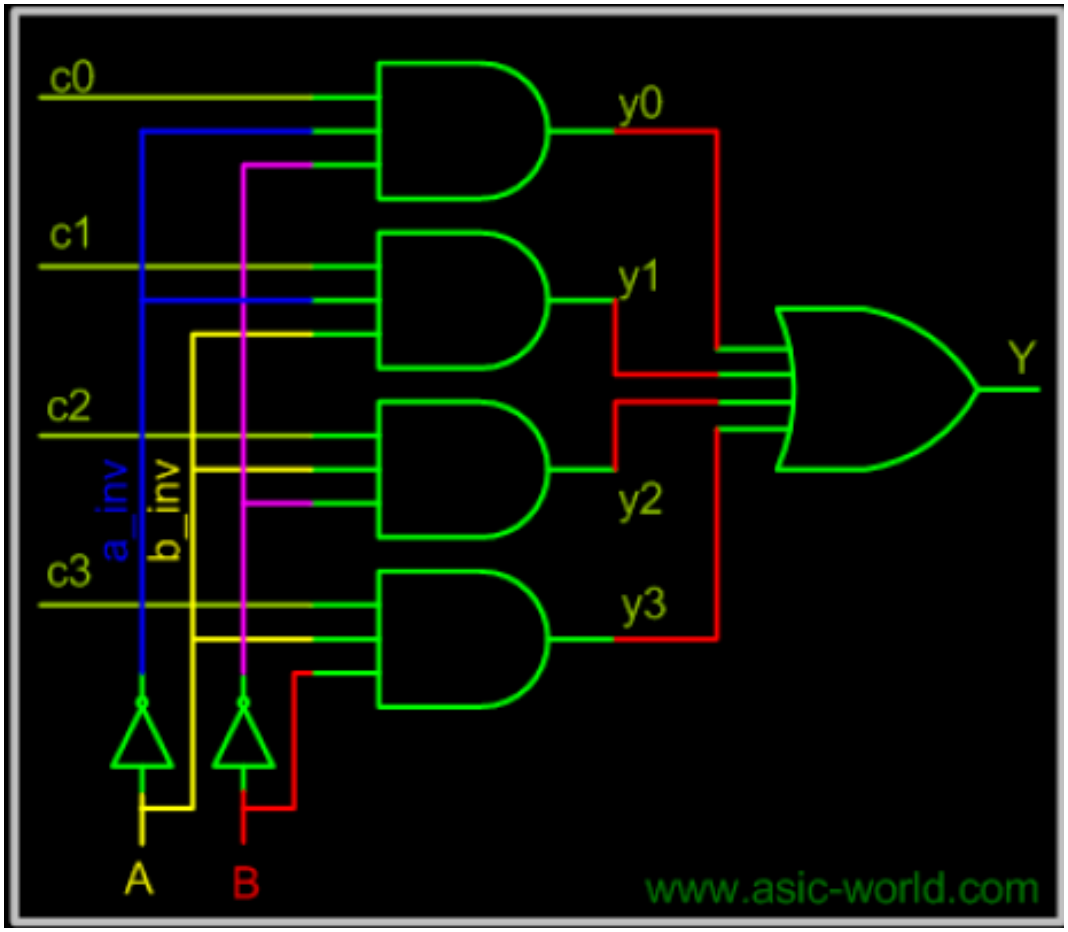
HW#4

- 製作一個負緣觸發的8 bit counter，與其測試模組 test_8_bit_counter
- 使用Simulation驗證電路正確

4-to-1 Multiplexer



4-to-1 Multiplexer



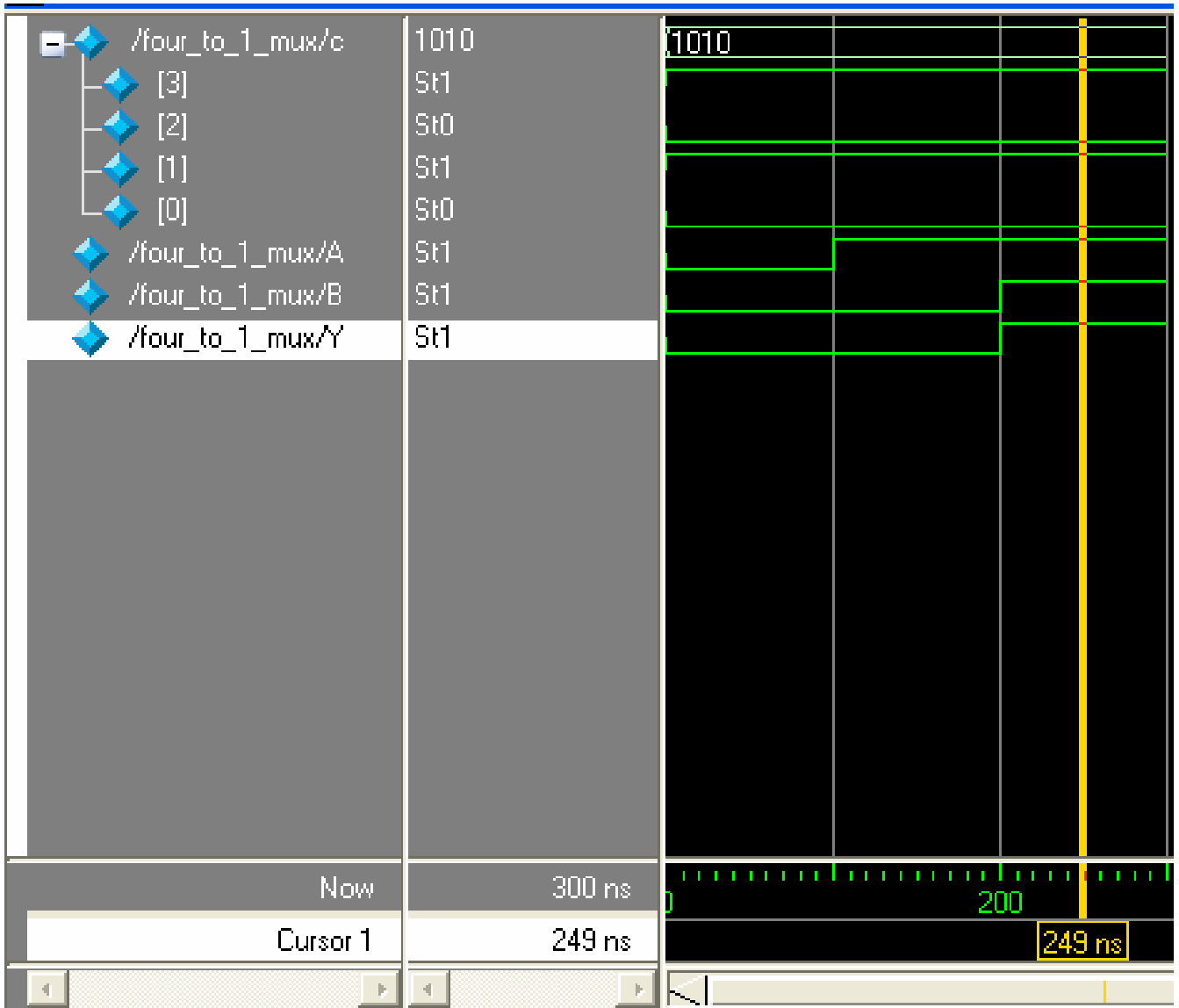
4-to-1 Multiplexer(Cont.)

```
module four_to_1_mux(c, A, B, Y);
     input[3:0] c;
     input A,B;
     output Y;

     not  (A_inv, A),
          (B_inv, B);
     and  (y0,c[0],A_inv,B_inv),
          (y1,c[1],A_inv,B),
          (y2,c[2],A,B_inv),
          (y3,c[3],A,B);
     or   (Y,y0,y1,y2,y3);

endmodule
```

4-to-1 Multiplexer(Cont.)



case statement

always @ (觸發訊號)
case(訊號)
    <case 1>: 執行邏輯運算1
    <case 2>: 執行邏輯運算2
        .....
    default: 執行邏輯運算n
end case

4-to-1 Multiplexer(Cont.)

```
module four_to_1_mux_1(c, A, B, Y);
     input[3:0] c;
     input A,B;
     output Y;
     reg Y;
     always@(A or B or Y)
      case({A,B})
           2'b00: Y = c[0];
           2'b01: Y = c[1];
           2'b10: Y = c[2];
           2'b11: Y = c[3];
      endcase
endmodule
```

4-to-1 Multiplexer(Cont.)

- 測試模組 test_four_to_1_mux_1:

```
module test_four_to_1_mux_1(c, A, B, Y);
    input[3:0] c;
    input A,B;
    output Y;
    four_to_1_mux u1(c, A, B, Y);
endmodule
```

HW#5

- 製作一個 2_to_1 multiplexer，與其測試模組 test_2_to_1_multiplexer
- 使用Simulation驗證電路正確

Assign Statement

Assign 訊號A = (訊號B) ? 數值X:數值Y

=

if (訊號B == 1) 訊號A = 數值X

else 訊號A = 數值Y

訊號A 需被宣告爲 wire, 如 wire signal A;

3-to-8 decoder

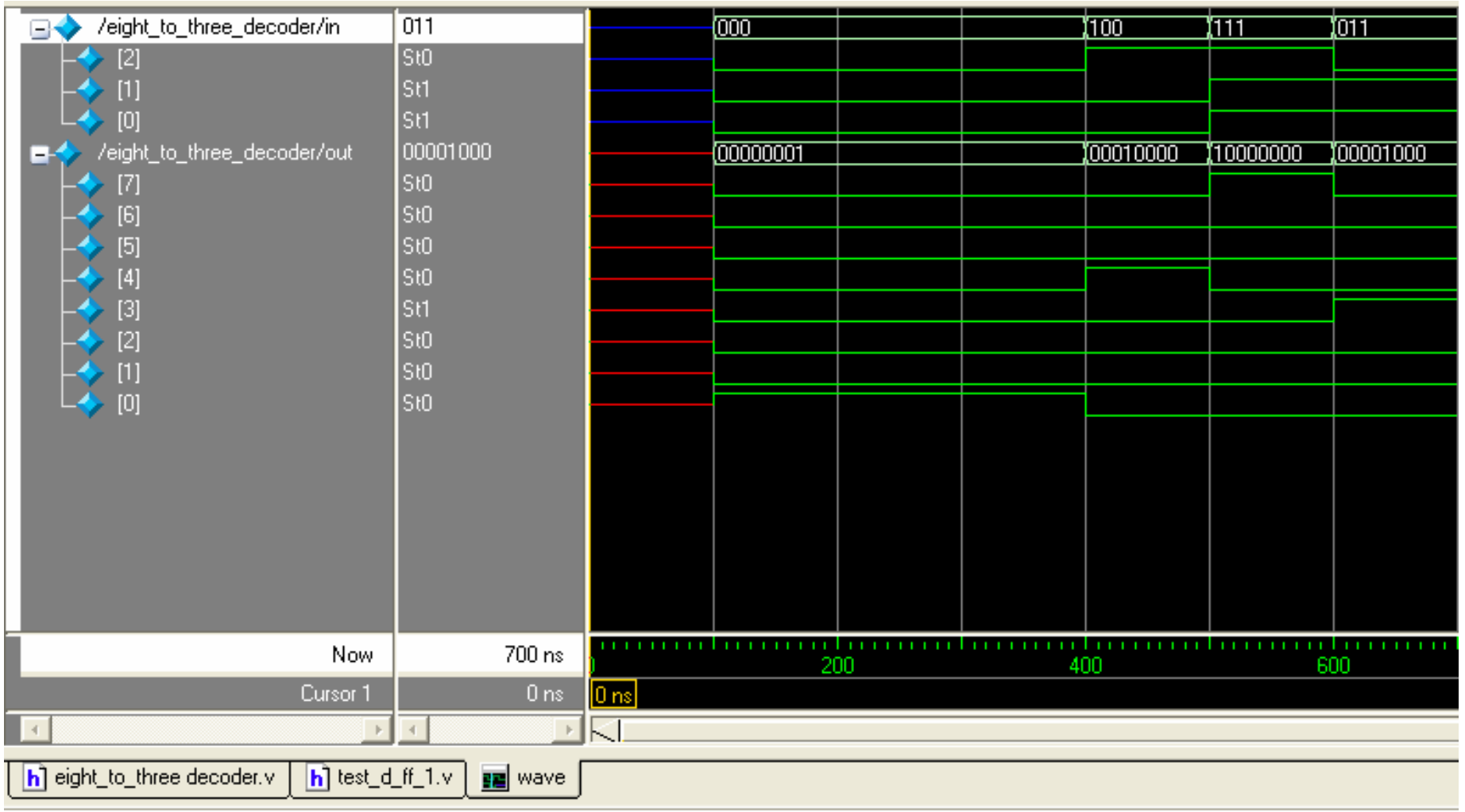
input	output
000	00000001
001	00000010
010	00000100
011	00001000
100	00010000
101	00100000
110	01000000
111	10000000

3-to-8 decoder

```
module eight_to_three_decoder (in, out);
  input [2:0] in;
  output [7:0] out;
  wire[7:0] out;
  assign out = (in == 3'b000) ? 8'b00000001:
               (in == 3'b001) ? 8'b00000010:
               (in == 3'b010) ? 8'b00000100:
               (in == 3'b011) ? 8'b00001000:
               (in == 3'b100) ? 8'b00010000:
               (in == 3'b101) ? 8'b00100000:
               (in == 3'b110) ? 8'b01000000:
               (in == 3'b111) ? 8'b10000000: 8'b00000000;

endmodule
```

3-to-8 decoder(Cont.)



HW#6

- 使用assignment 指令，製作一個 2-to-4 decoder，
- 使用Simulation驗證電路正確

input	output
00	0001
01	0010
10	0100
11	1000