Verilog 硬體描述語言

Verilog HDL

A Guide to Digital Design and Synthesis

IEEE 1364-2001 Compliant

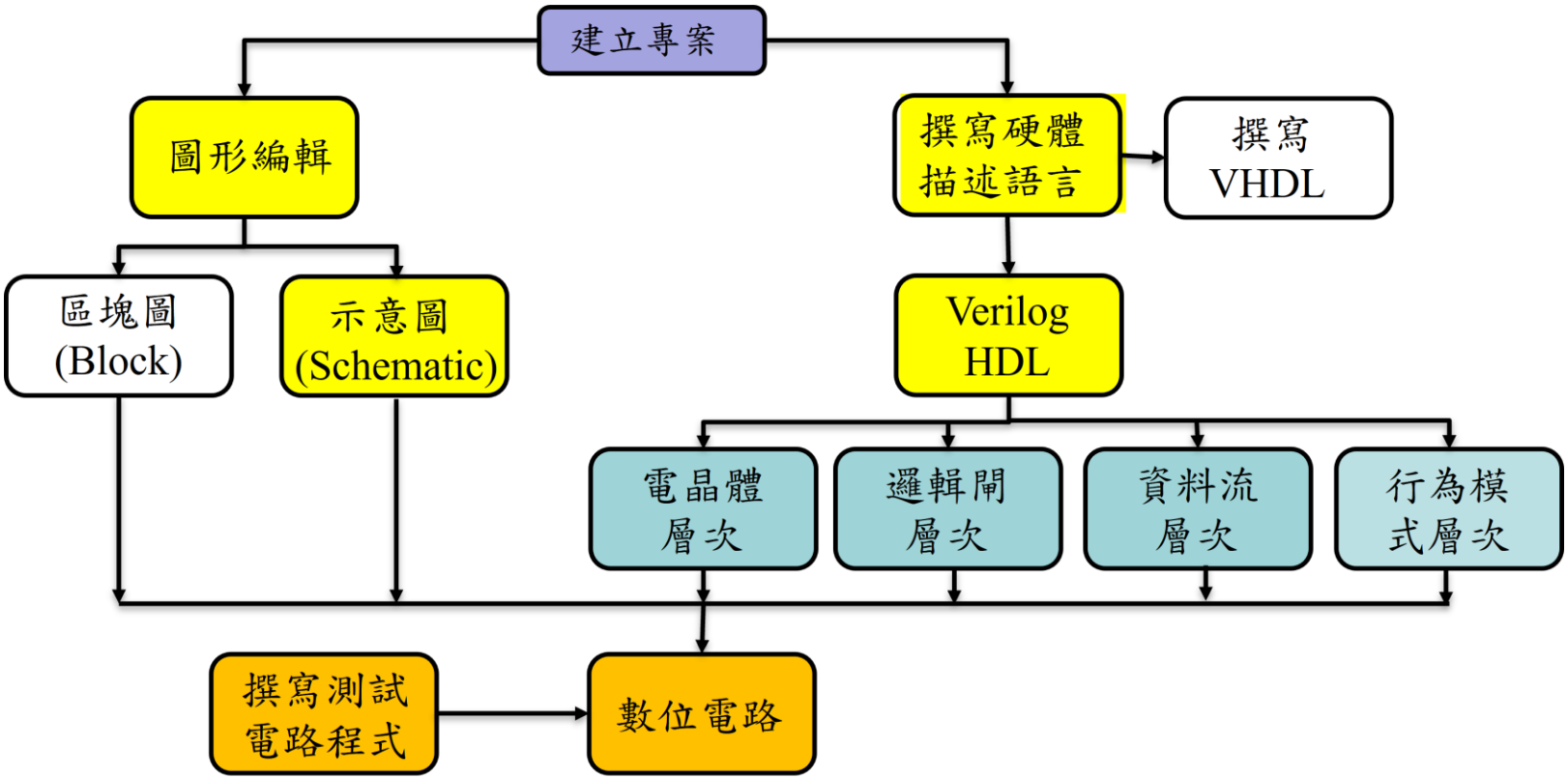
第5章 邏輯閘層次模型

5.0 Verilog 四種描述風格

- 電晶體層次 (transistor mode)
- 邏輯閘層次 (gate level mode)
- 資料流層次 (data flow mode)
- 行為模式層次 (behavior mode)

5.0 Verilog 四種描述風格

- 使用Quartus® 設計數位電路選項架構圖



5.0 Verilog 四種描述風格

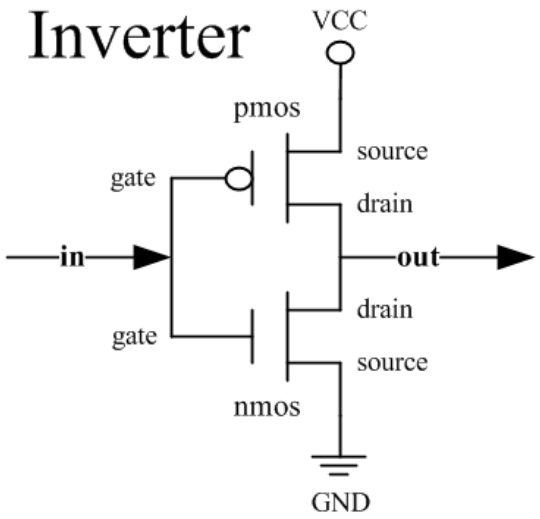
- 電晶體層次 (transistor mode)
  - 開關準位模式
  - 明白電晶體元件的特性
  - 最低階的描述層次

5.0 Verilog 四種描述風格

- 電晶體層次 (transistor mode)
  - 範例:用 PMOS (P-channel Metal Oxide Semiconductor) 及 NMOS (N-channel Metal Oxide Semiconductor) 來建立反向器 NOT (Inverter) 閘

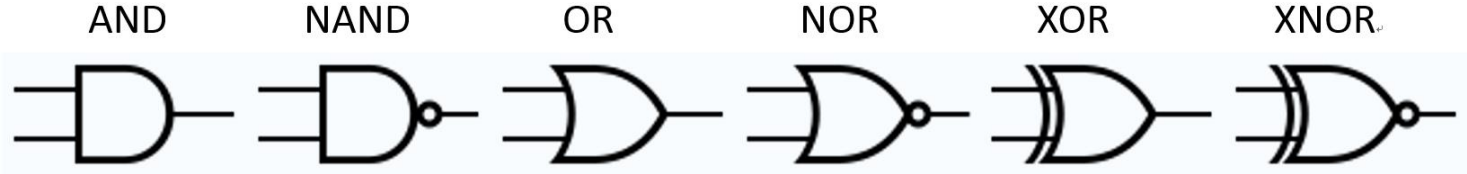
```
module  P_N_MOS_NOT (in, out);
input   in;   output  out;   wire   out;
supply1  vcc;   supply0  gnd;

  pmos( out, vcc, in );
  // drain, source, gate
  nmos( out, gnd, in );
  // drain, source, gate
endmodule
```



5.0 Verilog 四種描述風格

- 邏輯閘層次 (gate level mode)
  - 基本邏輯閘
  - 使用方法：<閘名稱> <閘編號> (輸出埠, 輸入埠1, 輸入埠2...);



True Table：

Input		Output					
In_1	In_2	and	nand	or	nor	xor	xnor
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

5.0 Verilog 四種描述風格

- 邏輯閘層次 (gate level mode)
  - 由基本邏輯閘組合而成，如：

    not、and、or、nand、nor、xor、xnor

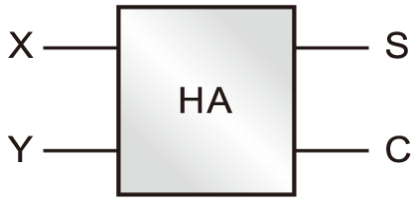
範例	檔名：HALF_ADDER_GATE_LEVEL_MODE
以邏輯閘層次的描述風格，設計一個半加器電路。	

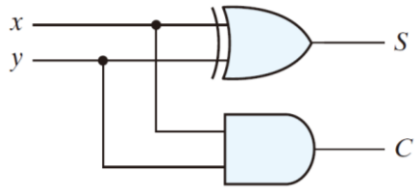
```
1  /*****************************
2   *        half adder         *
3   *   using gate level mode   *
4   *  Filename : HALF_ADDER.v  *
5   *****************************/
6
7  module HALF_ADDER(X, Y, C, S);
8
9    input   X, Y;
10   output  C, S;
11
12   xor SUM (S, X, Y);
13   and CARRY (C, X, Y);
14
15 endmodule
```

Half Adder

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

表3-2 半加器真值表



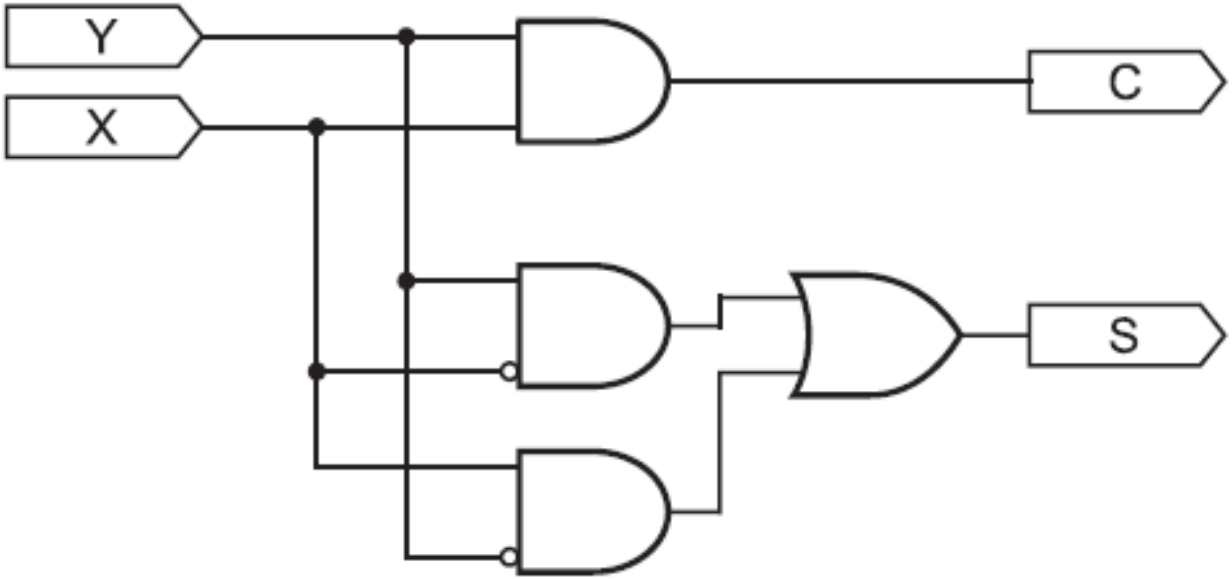


布林代數：

$$S = \bar{X}Y + X\bar{Y}$$
$$= X \oplus Y$$
$$C = XY$$

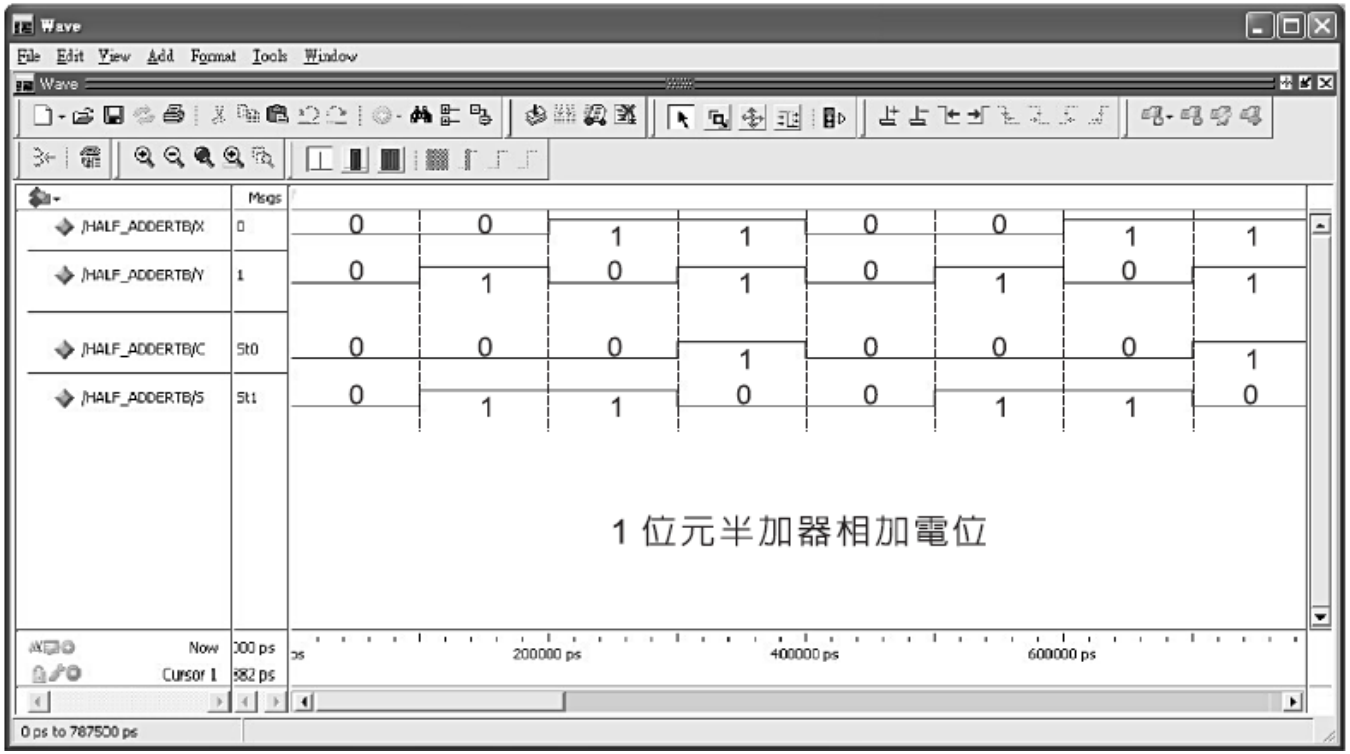
5.0 Verilog 四種描述風格

- 邏輯閘層次 (gate level mode)
  - 合成電路(Quartus II: RTL Viewer)



5.0 Verilog 四種描述風格

- 邏輯閘層次 (gate level mode)
  - 功能模擬結果波形圖(Quartus II: 需撰寫測試程式)



5.0 Verilog 四種描述風格

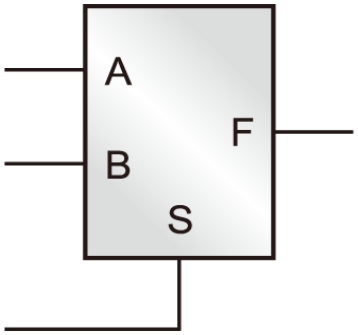
- 資料流層次 (data flow mode)
  - 又稱暫存器轉移層次
  - 以指定方式 (assign) 去描述訊號在電路內部的傳送

範例	檔名：MUL2_1_DATA_FLOW_MODE
於 verilog 語言中，反相閘、及閘與或閘的位元運算符號依序為“~”、“&”與“|”，試以資料流描述的風格，設計一個 1 位元 2 對 1 多工器。	

5.0 Verilog 四種描述風格

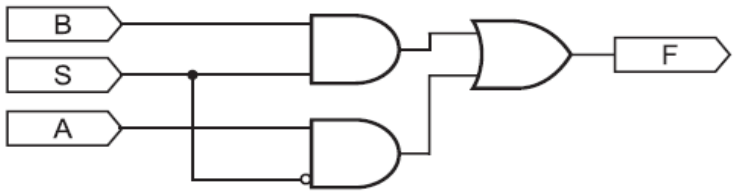
- 資料流層次 (data flow mode)
  - 原始程式 (source program)

```
 1  /***************************
 2   *   2 to 1 MULTIPLEXER    *
 3   *  using data flow mode   *
 4   *  Filename : MUL2_1.v    *
 5   ***************************/
 6
 7  module MUL2_1(A, B, S, F);
 8
 9    input   A, B, S;
10    output  F;
11
12    assign  F = ~S & A | S & B;
13
14  endmodule
```



布林代數：

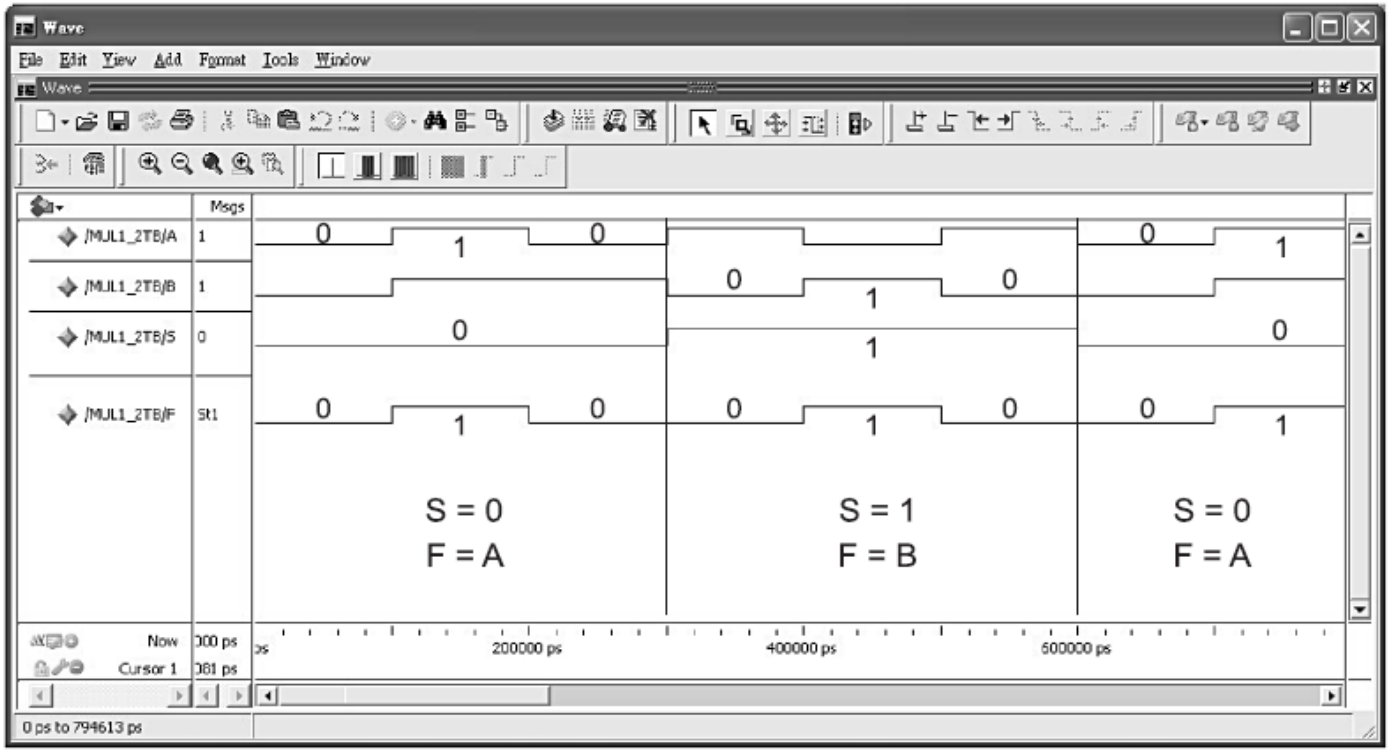
$F = \overline{S}A + SB$



- 合成電路

5.0 Verilog 四種描述風格

- 資料流層次 (data flow mode)
  - 功能模擬結果波形圖(Quartus II: 需撰寫測試程式)



5.0 Verilog 四種描述風格

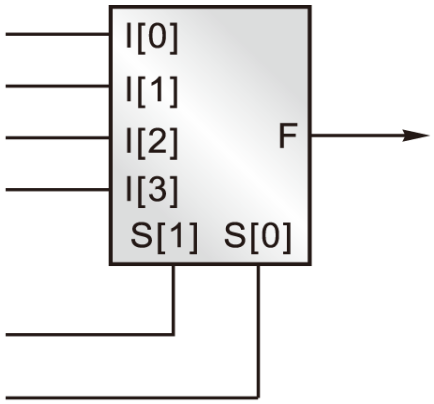
- 行為模式層次 (behavior mode)
  - 與一般的高階語言一樣 (如 C 語言)
  - 將電路的實際動作狀況 (即行為模式),以 Verilog 語言描述出來

範例	檔名：MUL4_1_BEHAVIOR_MODE
於 verilog 語言中，反相閘、及閘、或閘的位元運算符號依序為 “~” 、“&” 、“|” ，試以行為模式描述的風格，設計一個 1 位元 4 對 1 多工器。	

5.0 Verilog 四種描述風格

- 行為模式層次 (behavior mode)
  - 原始程式 (source program)

```
 1  /************************
 2   *  4 to 1 multiplexer   *
 3   *  using behavior mode  *
 4   *  Filename : MUL4_1.v  *
 5   ************************/
 6
 7  module MUL4_1(I, S, F);
 8
 9    Input   [3:0] I;
10    Input   [1:0] S;
11    Output  F;
12
13    reg F;
14
15    always @(I, S)
16     F = ~S[1] & ~S[0] & I[0] |
17         ~S[1] &  S[0] & I[1] |
18          S[1] & ~S[0] & I[2] |
19          S[1] &  S[0] & I[3];
20
21  endmodule
```

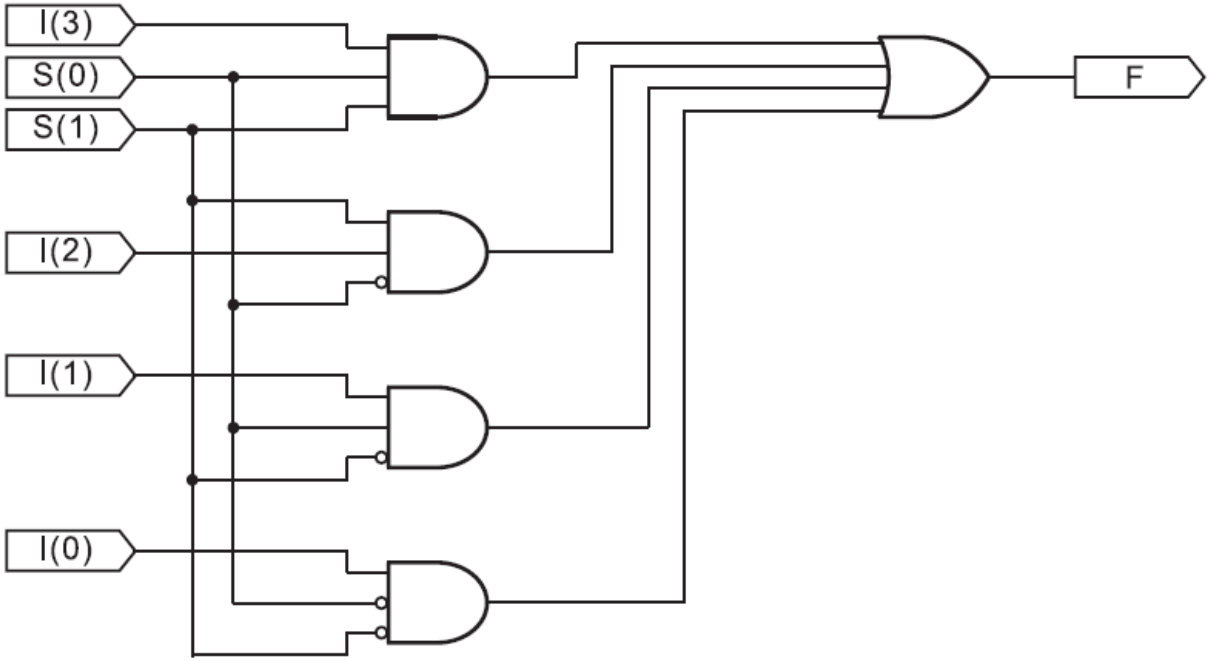


布林代數：

$$F = \overline{S[1]}\,\overline{S[0]}\,I[0] + \overline{S[1]}\,S[0]\,I[1] + S[1]\,\overline{S[0]}\,I[2] + S[1]\,S[0]\,I[3]$$

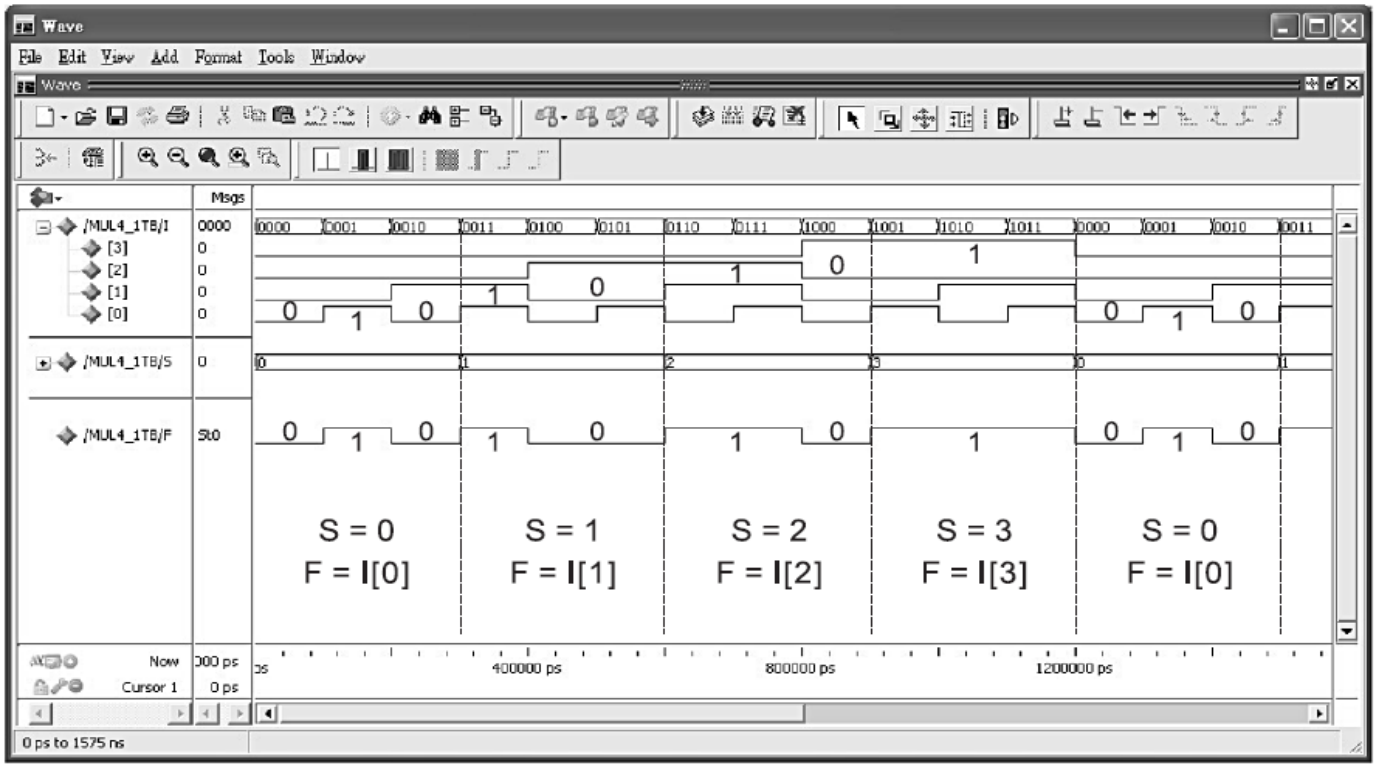
5.0 Verilog 四種描述風格

- 行為模式層次 (behavior mode)
  - 合成電路



5.0 Verilog 四種描述風格

- 行為模式層次 (behavior mode)
  - 功能模擬結果波形圖(Quartus II: 需撰寫測試程式)



5.1 閘的種類

- 5.1.1 and/or 閘
  - 閘有一個純量輸出與多個純量輸入，而且閘的第一項為輸出，其他項皆為輸入。
  - Verilog可使用的and/or閘有六種: and or xor nand nor xnor
  - 這些閘可以不用別名名稱(instance name)，若輸入多於兩個以上時，可直接加上這些輸出入埠，Verilog會自動找到相符的閘。

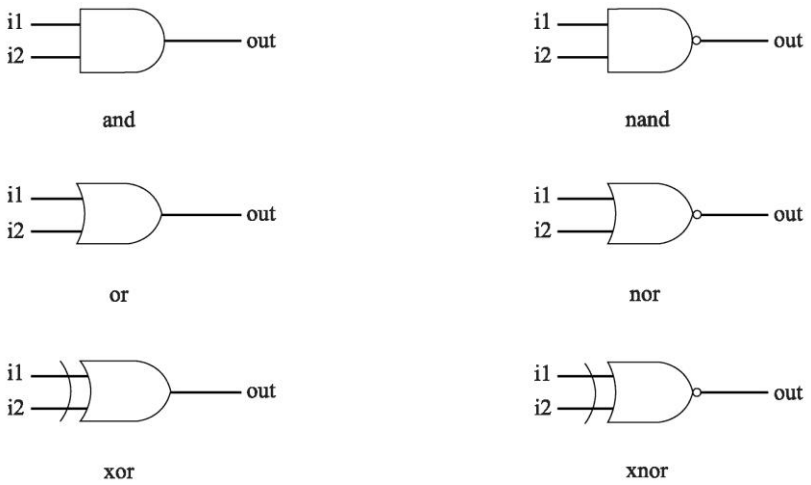


圖5-1　基本閘

5.1 閘的種類

- 範例5-1 and/or閘的取別名

```
wire OUT, IN1, IN2;

// 基本閘的取別名
and a1(OUT, IN1, IN2);
nand na1(OUT, IN1, IN2);
or or1(OUT, IN1, IN2);
nor nor1(OUT, IN1, IN2);
xor x1(OUT, IN1, IN2);
xnor nx1(OUT, IN1, IN2);
// 輸入多於二、三個輸入的及閘。
nand na1_3inp(OUT, IN1, IN2, IN3);

// 閘的取別名不需別名名稱
and (OUT, IN1, IN2); // 合法閘的取別名
```

5.1 閘的種類

and	i1=0	i1=1	i1=x	i1=z
i2=0	0	0	0	0
i2=1	0	1	x	x
i2=x	0	x	x	x
i2=z	0	x	x	x

nand	i1=0	i1=1	i1=x	i1=z
i2=0	1	1	1	1
i2=1	1	0	x	x
i2=x	1	x	x	x
i2=z	1	x	x	x

or	i1=0	i1=1	i1=x	i1=z
i2=0	0	1	x	x
i2=1	1	1	1	1
i2=x	x	1	x	x
i2=z	x	1	x	x

nor	i1=0	i1=1	i1=x	i1=z
i2=0	1	0	x	x
i2=1	0	0	0	0
i2=x	x	0	x	x
i2=z	x	0	x	x

xor	i1=0	i1=1	i1=x	i1=z
i2=0	0	1	x	x
i2=1	1	0	x	x
i2=x	x	x	x	x
i2=z	x	x	x	x

xnor	i1=0	i1=1	i1=x	i1=z
i2=0	1	0	x	x
i2=1	0	1	x	x
i2=x	x	x	x	x
i2=z	x	x	x	x

表5-1　真值表

5.1 閘的種類

- 5.1.2 buf/not 閘
  - buf/not 閘有一個純量輸入與多個純量輸出，而且閘的最末項為輸入，其他項皆為輸出。

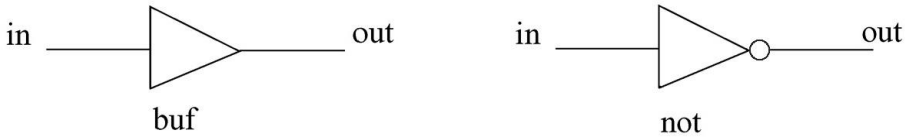


圖5-2　Buf/Not閘

- 範例5-2 Buf/Not 閘的取別名

```
// 基本閘的取別名
buf b1(OUT1, IN);
not n1(OUT1, IN);
// 輸出多於二
buf b1_2out(OUT1, OUT2, IN);
// 閘的取別名不需別名名稱
not (OUT1, IN); // 合法閘的取別名
```

buf	in	out
	0	0
	1	1
	x	x
	z	x

not	in	out
	0	1
	1	0
	x	x
	z	x

表5-2　真值表

5.1 閘的種類

- bufif/notif 閘
  - bufif/notif 閘是buf/not 閘加上一個控制訊號。
  - 四種閘: bufif1 bufif0 notif1 notif0

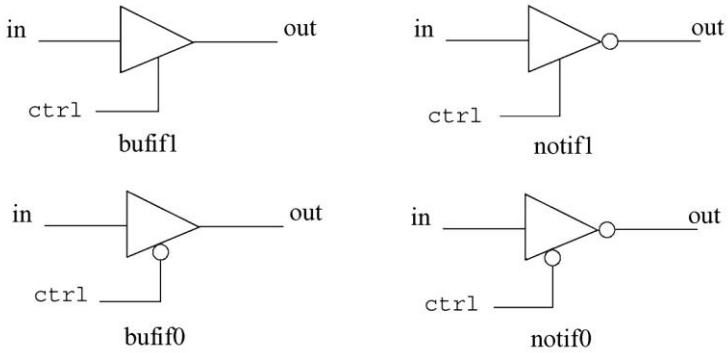


圖5-3 Bufif/notif閘

- 範例5-3 取別名 bufif/notif 閘

```
// 取別名 bufif 閘
bufif1 b1 (out, in, ctrl);
bufif0 b0 (out, in, ctrl);

// 取別名 notif 閘
notif1 n1 (out, in, ctrl);
notif0 n0 (out, in, ctrl);
```

5.1 閘的種類

- 表5-3　真值表

bufif1	ctrl 0	1	x	z
in 0	z	0	L	L
1	z	1	H	H
x	z	x	x	x
z	z	x	x	x

bufif0	ctrl 0	1	x	z
in 0	0	z	L	L
1	1	z	H	H
x	x	z	x	x
z	x	z	x	x

notif1	ctrl 0	1	x	z
in 0	z	1	H	H
1	z	0	L	L
x	z	x	x	x
z	z	x	x	x

notif0	ctrl 0	1	x	z
in 0	1	z	H	H
1	0	z	L	L
x	x	z	x	x
z	x	z	x	x

5.1 閘的種類

- 5.1.3 別名陣列(array of instances)
  - 有重複的模組需要一再使用時，Verilog 2001標準可使用別名陣列
  - 範例5-4 別名陣列

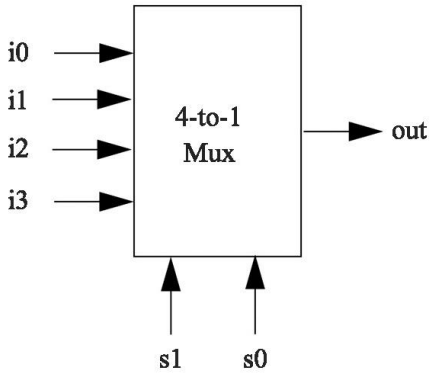
```
wire [7:0] OUT, IN1, IN2;

// basic gate instantiations.
nand n_gate[7:0](OUT, IN1, IN2);

// 上面的敘述相當於一次宣告八個模組別名，特別注意輸出入埠的定址關係。
nand n_gate0(OUT[0], IN1[0], IN2[0]);
nand n_gate1(OUT[1], IN1[1], IN2[1]);
nand n_gate2(OUT[2], IN1[2], IN2[2]);
nand n_gate3(OUT[3], IN1[3], IN2[3]);
nand n_gate4(OUT[4], IN1[4], IN2[4]);
nand n_gate5(OUT[5], IN1[5], IN2[5]);
nand n_gate6(OUT[6], IN1[6], IN2[6]);
nand n_gate7(OUT[7], IN1[7], IN2[7]);
```

5.1 閘的種類

- 5.1.4 範例一
  - 邏輯閘層次的多工器



s1	s0	out
0	0	I0
0	1	I1
1	0	I2
1	1	I3

圖5-4　4-to-1多工器

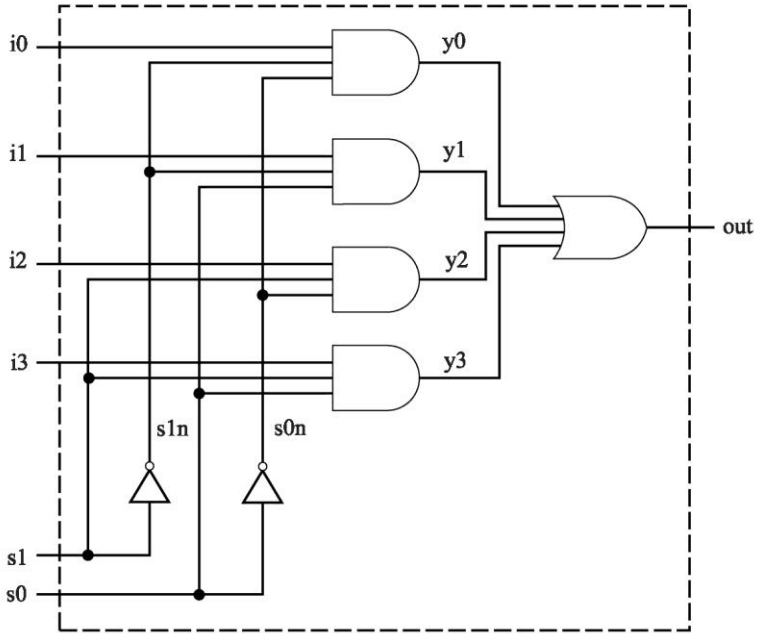


圖5-5　多工器的邏輯電路

5.1 閘的種類

- 範例5-5 多工器的Verilog程式

```
// 從 I/O 圖示中輸出入埠表示，建立 4-to-1 多工器的模組。
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0);

// 宣告輸出入埠
output out;
input i0, i1, i2, i3;
input s1, s0;

// 宣告內部接線
wire s1n, s0n;
wire y0, y1, y2, y3;

// 閘的取別名

// 產生 s1n 和 s0n 信號
not(s1n, s1);
not(s0n, s0);

// 三輸入及閘之取別名
and (y0, i0, s1n, s0n);
and (y1, i1, s1n, s0);
and (y2, i2, s1, s0n);
and (y3, i3, s1, s0);

// 四輸入或閘之取別名
or (out, y0, y1, y2, y3);

endmodule
```

5.1 閘的種類

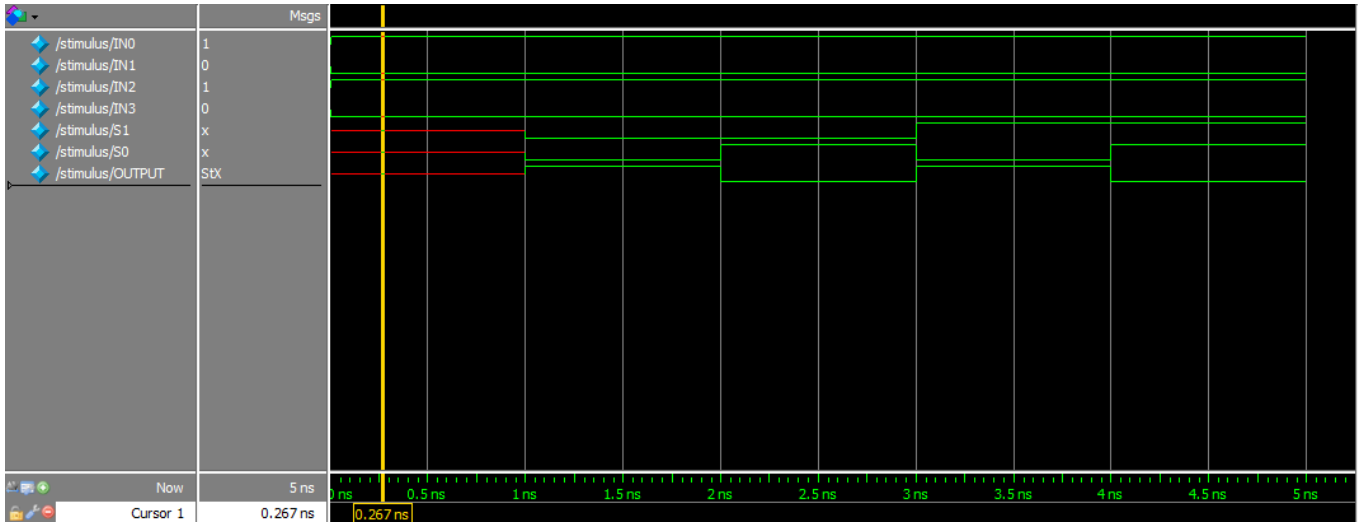
- 範例5-6 模擬程式

```
1  // Define the stimulus module (no ports)
2  `timescale 1ns/100ps
3  module stimulus;
4  // Declare variables to be connected
5  // to inputs
6  reg IN0, IN1, IN2, IN3;
7  reg S1, S0;
8  // Declare output wire
9  wire OUTPUT;
10 // Instantiate the multiplexer
11 mux4_to_1 mymux(OUTPUT, IN0, IN1, IN2, IN3, S1, S0);
12 // Stimulate the inputs
13 initial
14 begin
15    // set input lines
16    IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;
17    #1 $display("IN0= %b, IN1= %b, IN2= %b, IN3= %b\n",IN0,IN1,IN2,IN3);
18    // choose IN0
19    S1 = 0; S0 = 0;
20    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b \n", S1, S0, OUTPUT);
21    // choose IN1
22    S1 = 0; S0 = 1;
23    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b \n", S1, S0, OUTPUT);
24    // choose IN2
25    S1 = 1; S0 = 0;
26    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b \n", S1, S0, OUTPUT);
27    // choose IN3
28    S1 = 1; S0 = 1;
29    #1 $display("S1 = %b, S0 = %b, OUTPUT = %b \n", S1, S0, OUTPUT);
30 end
31 endmodule
```

5.1 閘的種類

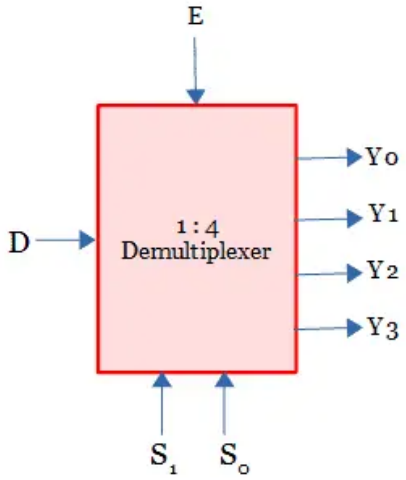
- 範例5-6 模擬程式輸出結果

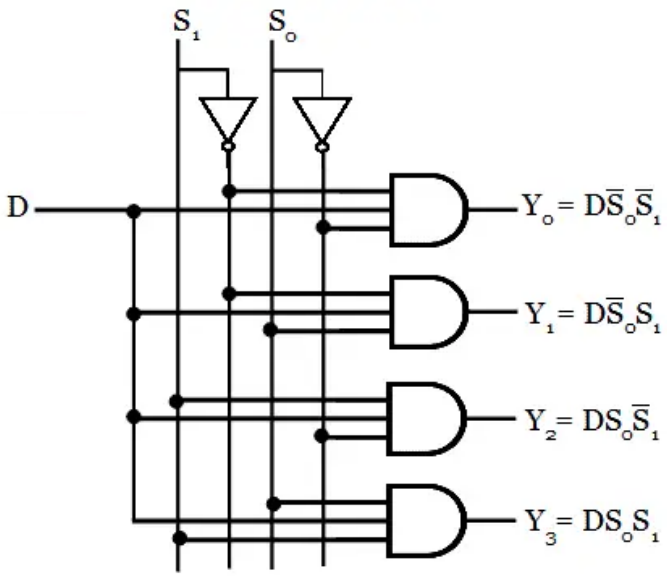
```
IN0 = 1, IN1 = 0, IN2 = 1, IN3 = 0
S1 = 0, S0 = 0, OUTPUT = 1
S1 = 0, S0 = 1, OUTPUT = 0
S1 = 1, S0 = 0, OUTPUT = 1
S1 = 1, S0 = 1, OUTPUT = 0
```



5.1 閘的種類

- 5.1.4 作業一
  - 邏輯閘層次的1-to-4 解多工器
  - 請仿照 4-to-1多工器範例與步驟，完成1-to-4 解多工器專案(邏輯閘層次的Verilog程式與模擬程式)





5.1 閘的種類

- 5.1.4 範例二
  - 邏輯閘層次的一位元全加器

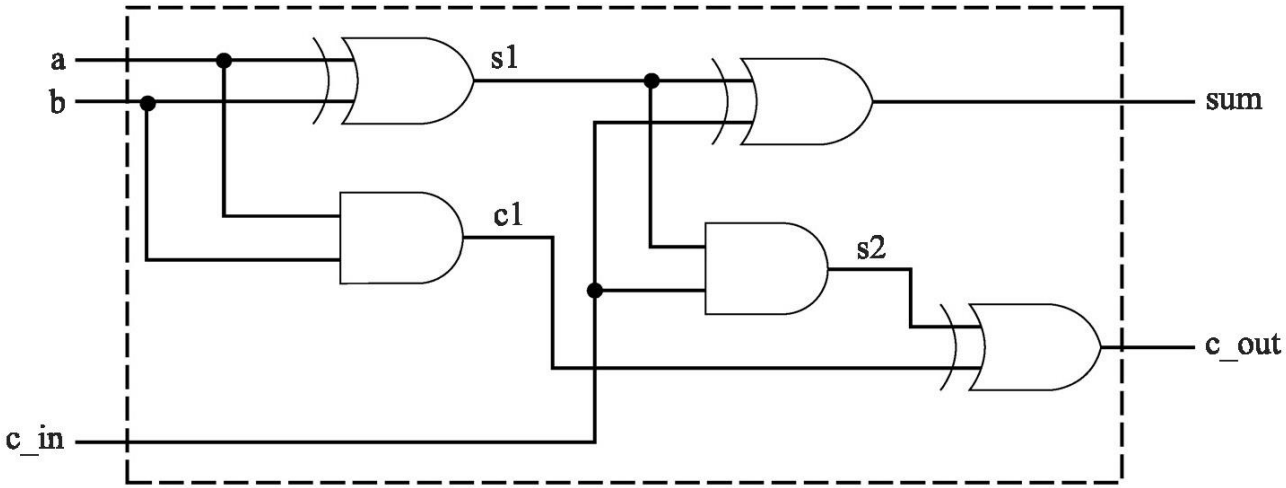


圖5-6　一位元全加器

5.1 閘的種類

- 範例5-7 一位元全加器的Verilog程式

```
// 定義一位元全加器
module fulladd(sum, c_out, a, b, c_in)

// 宣告輸出入埠
output sum, c_out;
input a, b, c_in;

// 宣告內部接線
wire s1, c1, c2;

// 閘的取別名
xor(s1, a, b);
and (c1, a, b);
xor (sum, s1, c_in);
and (c2, s1, c_in);
xor (c_out, c2, c1);
endmodule
```

5.1 閘的種類

- 範例5-8 四位元漣波進位全加器

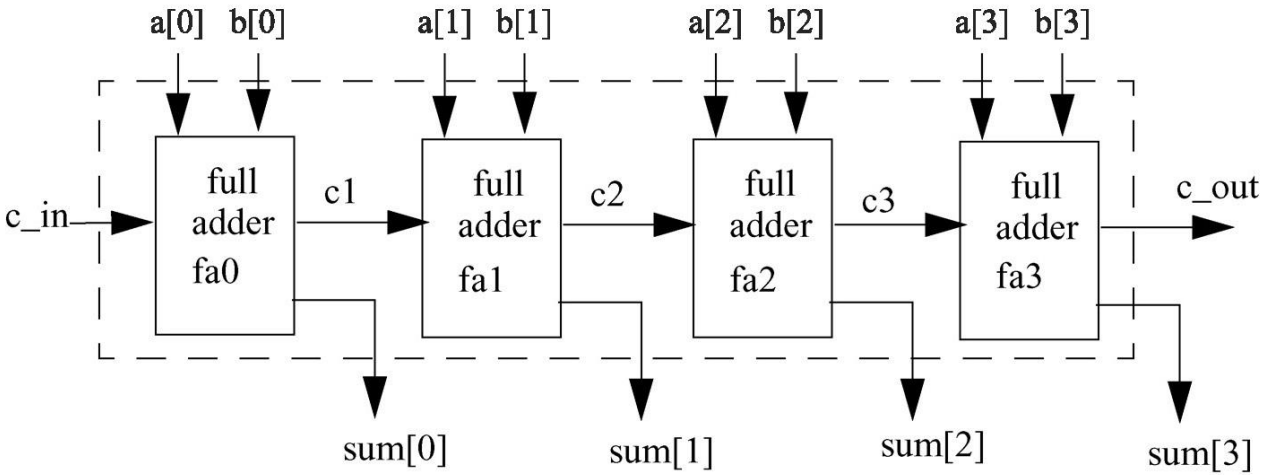


圖5-7　四位元全加器

5.1 閘的種類

- 範例5-8 四位元漣波進位全加器的Verilog程式

```
// 定義四位元全加器
module fulladd4(sum, c_out, a, b, c_in);

// 宣告輸出入埠
output [3:0] sum;
output c_out;
input [3:0] a, b;
input c_in;

// 宣告內部接線
wire c1, c2, c3;

// 四個一位元全加器的取別名
fulladd fa0(sum[0], c1, a[0], b[0], c_in);
fulladd fa1(sum[1], c2, a[1], b[1], c1);
fulladd fa2(sum[2], c3, a[2], b[2], c2);
fulladd fa3(sum[3], c_out, a[3], b[3], c3);

endmodule
```

5.1 閘的種類

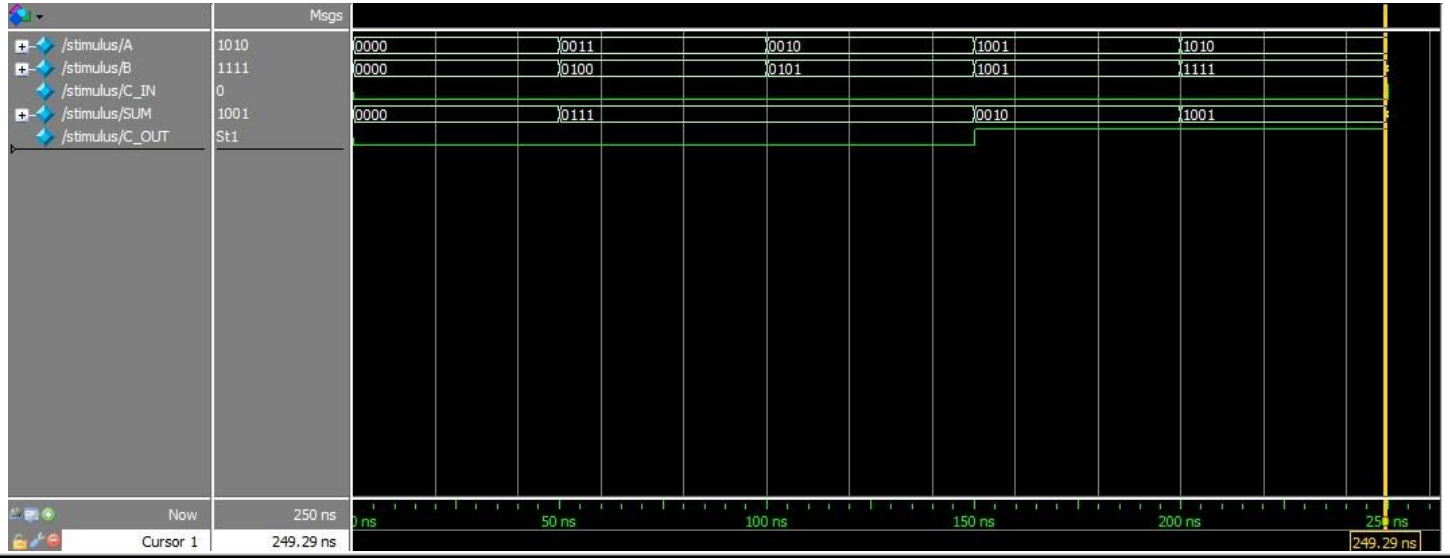
- 範例5-9 模擬程式

```
1  // Define the stimulus (top level module)
2  `timescale 1ns/100ps
3  module stimulus;
4  // Set up variables
5  reg [3:0] A, B;
6  reg C_IN;
7  wire [3:0] SUM;
8  wire C_OUT;
9  // Instantiate the 4-bit full adder. call it FA1_4
10 fulladd4 FA1_4(SUM, C_OUT, A, B, C_IN);
11 // Setup the monitoring for the signal values
12 initial
13 begin
14    $monitor($time," A= %b, B=%b, C_IN= %b,, C_OUT= %b, SUM= %b\n",
15                                          A, B, C_IN, C_OUT, SUM);
16 end
17
18 // Stimulate inputs
19 initial
20 begin
21    A = 4'd0; B = 4'd0; C_IN = 1'b0;
22    #50 A = 4'd3; B = 4'd4;
23    #50 A = 4'd2; B = 4'd5;
24    #50 A = 4'd9; B = 4'd9;
25    #50 A = 4'd10; B = 4'd15;
26    #50 A = 4'd10; B = 4'd5; C_IN = 1'b1;
27 end
28
29 endmodule
```

5.1 閘的種類

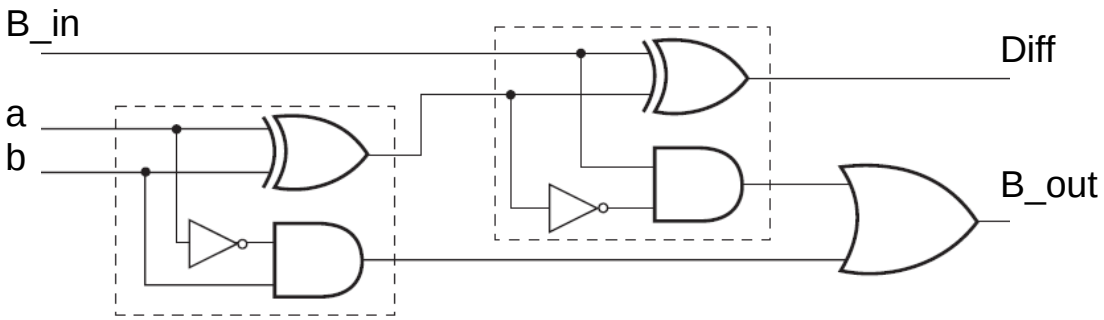
- 範例5-9 模擬程式輸出結果

```
0 A = 0000, B = 0000, C_IN = 0, --- C_OUT = 0, SUM = 0000
5 A = 0011, B = 0100, C_IN = 0, --- C_OUT = 0, SUM = 0111
10 A = 0010, B = 0101, C_IN = 0, --- C_OUT = 0, SUM = 0111
15 A = 1001, B = 1001, C_IN = 0, --- C_OUT = 1, SUM = 0010
20 A = 1010, B = 1111, C_IN = 0, --- C_OUT = 1, SUM = 1001
25 A = 1010, B = 0101, C_IN = 1, --- C_OUT = 1, SUM = 0000
```



5.1 閘的種類

- 5.1.4 作業
  - 邏輯閘層次的四位元漣波進位全減器
  - 請仿照四位元漣波進位全加器範例與步驟，完成四位元漣波進位全減器專案(邏輯閘層次的Verilog程式與模擬程式)



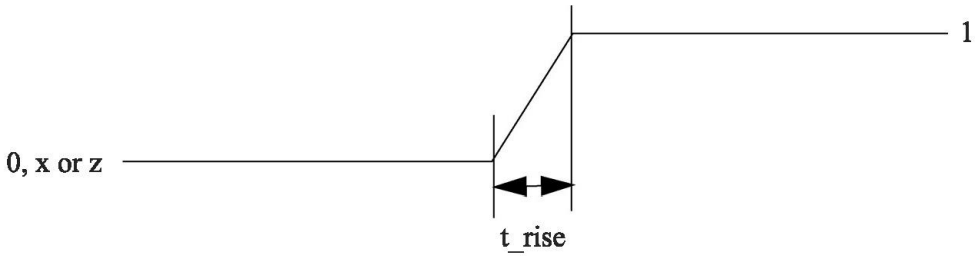
一位元全減器

a	b	B_in	B_out	Diff
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

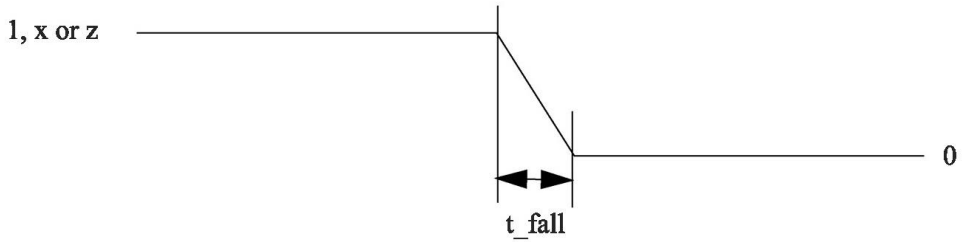
全減器真值表

5.2 閘的延遲(Gate Delays)

- 5.2.1 上升、下降和關閉延遲(Rise, Fall and Turn-off Delays)
  - **上升延遲**:指輸出由其他值上升至 **1** 所經過的延遲



  - **下降延遲**:指輸出由其他值下降至 **0** 所經過的延遲



5.2 閘的延遲(Gate Delays)

- 5.2.1 上升、下降和關閉延遲(Rise, Fall and Turn-off Delays)
  - **關閉延遲**:指輸出由其他值轉至 **z** 所經過的延遲。如果是轉至 **x** 的延遲，則取上述延遲中之最小值。如果有二個延遲被設定，則分別為上升和下降延遲，關閉延遲為取兩者中最小值。若只有一個延遲設定，則三個延遲皆為同值，若無設定則皆為零。
  - 範例 5-10 延遲設定的型態

```
// 所有轉換延遲 delay_time 時間
and #(delay_time) a1(out, i1, i2);

// 上昇和下降延遲的設定
and #(rise_val, fall_val) a2(out, i1, i2);

// 上昇，下降和關閉延遲的設定
bufif0 #(rise_val, fall_val, turnoff_val) b1(out, in, con-
    trol);

and #(5) a1(out, i1, i2); // 所有轉換延遲 5 單位時間
and #(4,6) a2(out, i1, i2); // 上升 = 4、下降 = 6。
bufif0 #(3,4,5) b1(out, in, control); // 上升 = 3、下降 = 4、
  // 關閉 = 5。
```

5.2 閘的延遲(Gate Delays)

- 5.2.2 Min/Typ/Max 數值
  - 每一種型態的延遲皆有三種數值
    - Min: 設計者所期望的最小延遲數值
    - Typ: 設計者所期望的典型延遲數值
    - Max:設計者所期望的最大延遲數值
  - 這三種數值在Verilog執行時用+maxdelays、+typdelays與 +mindelays去控制，內定值是typdelays。
  - 範例 5-11 最大，典型，最小延遲數值

```
// 一個延遲參數
// if +mindelays, delay=4
// if +typdelays, delay= 5
// if +maxdelays, delay=6
and #(4:5:6) a1(out, i1, i2);
```

5.2 閘的延遲(Gate Delays)

- 5.2.2 Min/Typ/Max 數值
  - 範例 5-11 最大，典型，最小延遲數值(續)

```
// 二個延遲參數
// if +mindelays, rise=3, fall=5, turn-off = min(3,5)
// if +typdelays, rise=4, fall=6, turn-off = min(4,6)
// if +maxdelays, rise=5, fall=7, turn-off=min(5,7)
and #(3:4:5, 5:6:7) a2(out, i1, i2);

// 三個延遲參數
// if +mindelays, rise=2, fall=3, turn-off=4
// if +typdelays, rise=3, fall=4, turn-off=5
// if +maxdelays, rise=4, fall=5, turn-off=6
and #(2:3:4, 3:4:5, 4:5:6) a3(out, i1, i2);
```

5.2 閘的延遲(Gate Delays)

- 5.2.2 Min/Typ/Max 數值
  - 假設含有延遲模組的檔案為test.v

```
// 用最大延遲模擬
verilog test.v +maxdelays

// 用最小延遲模擬
verilog test.v +mindelays

// 用典型延遲模擬
verilog test.v +typdelays
```

5.2 閘的延遲(Gate Delays)

- 5.2.3 延遲的範例(Delay Examples)
  - D是一個簡單的模組，功能為out=(a·b)+c

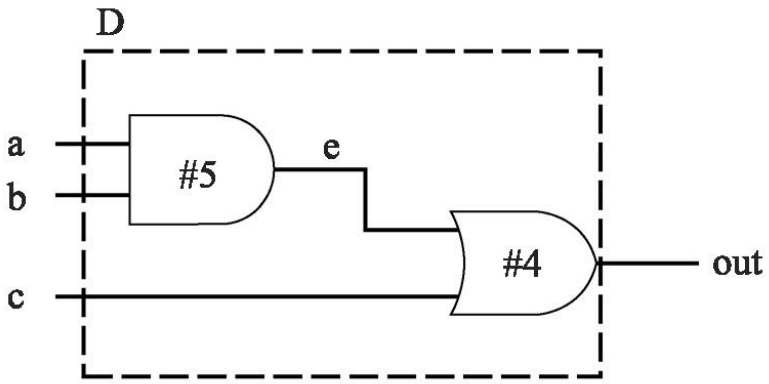


圖5-8　D模型

5.2 閘的延遲(Gate Delays)

- 5.2.3 延遲的範例(Delay Examples)
  - 範例5-12 用Verilog定義模組D

```
// 定義一簡單組合邏輯模組 D
module D (out, a, b, c);

// 宣告輸出入埠
output out;
input a, b, c;

// 宣告內部接線
wire e;

// 使用基本閘的取別名架構電路
and #(5) a1(e, a, b); // 閘 a1 延遲 5
or #(4) o1(out, e,c); // 閘 o1 延遲 4
endmodule
```

5.2 閘的延遲(Gate Delays)

- 5.2.3 延遲的範例(Delay Examples)
  - 範例5-13 模擬D模組

```
// 模擬 (最高階層模組)
module stimulus;

// 宣告變數
reg A, B, C;
wire OUT;

// 取別名module D
D d1 (OUT, A, B, C);

// 輸入模擬，模擬結束於40單位時間。
initial
begin
  A= 1'b0; B= 1'b0; C= 1'b0;
  #10A= 1'b1; B= 1'b1; C= 1'b1;
  #10A= 1'b1; B= 1'b0; C= 1'b0;
  #20 $finish;
end

endmodule
```

5.2 閘的延遲(Gate Delays)

- 5.2.3 延遲的範例(Delay Examples)
  - D模組模擬延遲時間波形圖

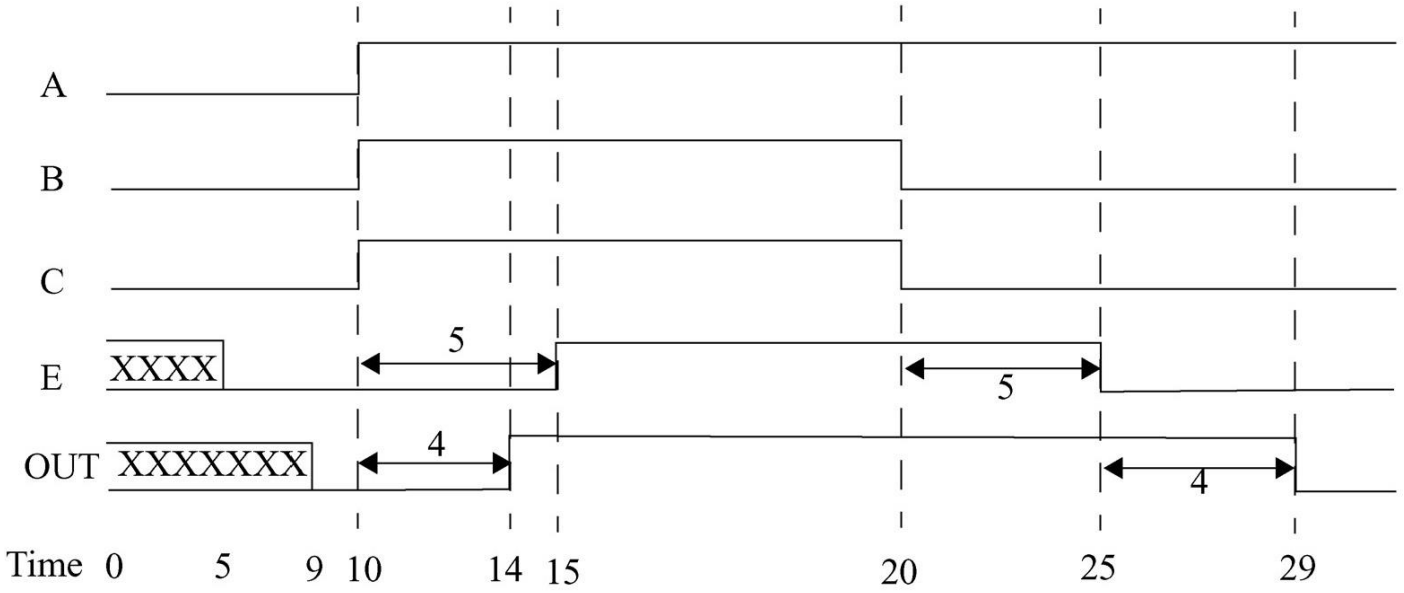


圖5-9 模擬延遲時間波形圖

5.2 閘的延遲(Gate Delays)

- 5.2.3 延遲的範例(Delay Examples)
  - D模組模擬延遲時間波形圖