正確學會 改訂新版 ディジタル回路と Verilog HDL

Verilog 的16堂課

第 4 章

組合電路 I

本章重點

邏輯電路，可分為組合電路以及序向電路。第 4、5 章會把具有代表性的組合電路逐一做個說明。

- 4.1 互斥或電路
- 4.2 多工電路文件
- 4.3 比較電路

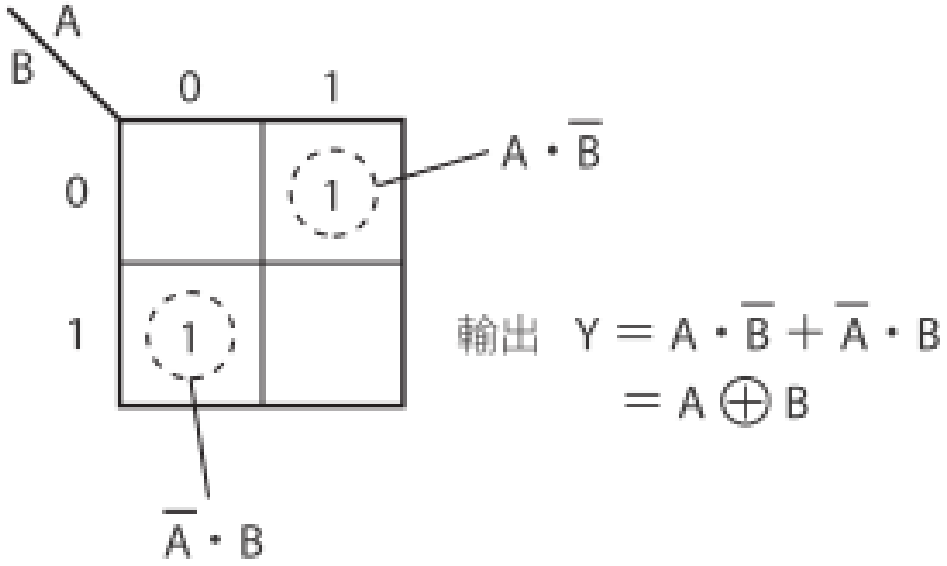
4.1 互斥或電路

4.1.1 真值表與卡諾圖

- 「互斥或」 eXclusive-OR 電路 (簡稱為 XOR) 只有輸入 A, B 的值不同的時候才會輸出 '1'

B	A	輸出
0	0	0
0	1	1
1	0	1
1	1	0

▲ 表 4.1　互斥或的真值表



▲ 圖 4.1　互斥或的卡諾圖

- XOR邏輯式

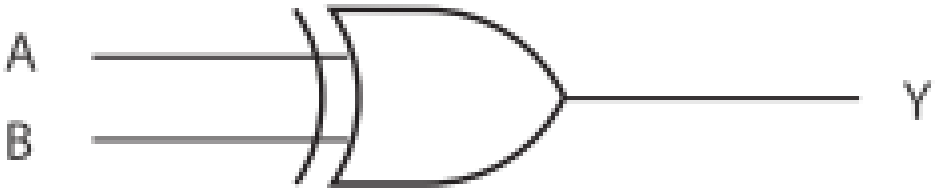
$$Y = A \cdot \overline{B} + \overline{A} \cdot B$$

- 也可簡化為：

$$Y = A \oplus B$$

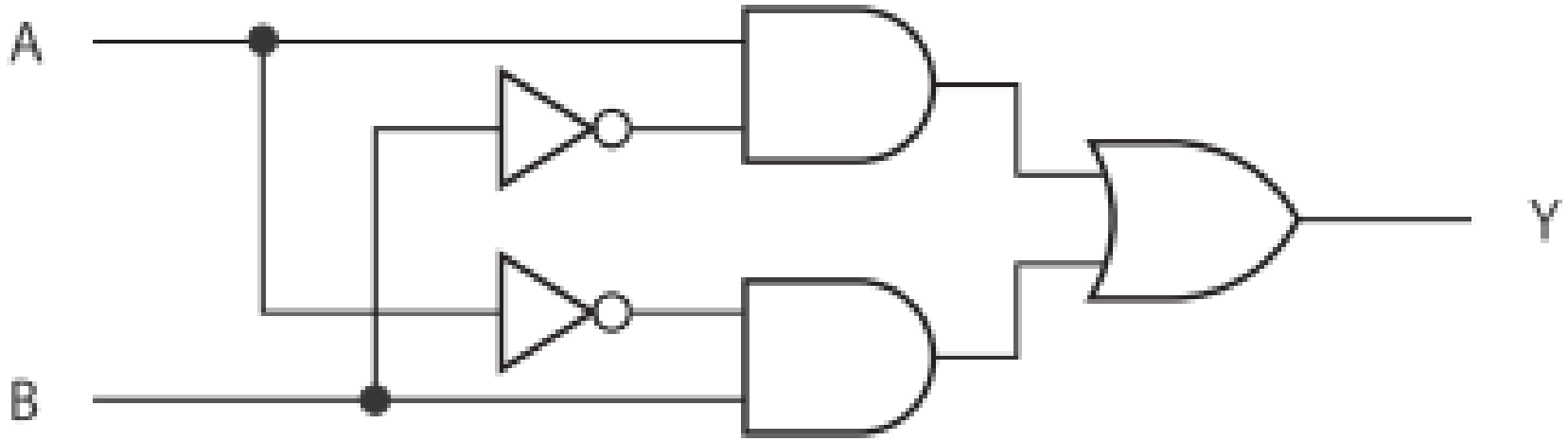
4.1.2 動作與電路符號

(a)



一般來說都會使用這個電路符號

- 等效電路如下：



4.1.3 XOR 電路的 Verilog HDL 描述

- 邏輯閘層

程式 4.1　互斥或電路的 Verilog HDL 描述 (邏輯閘層)

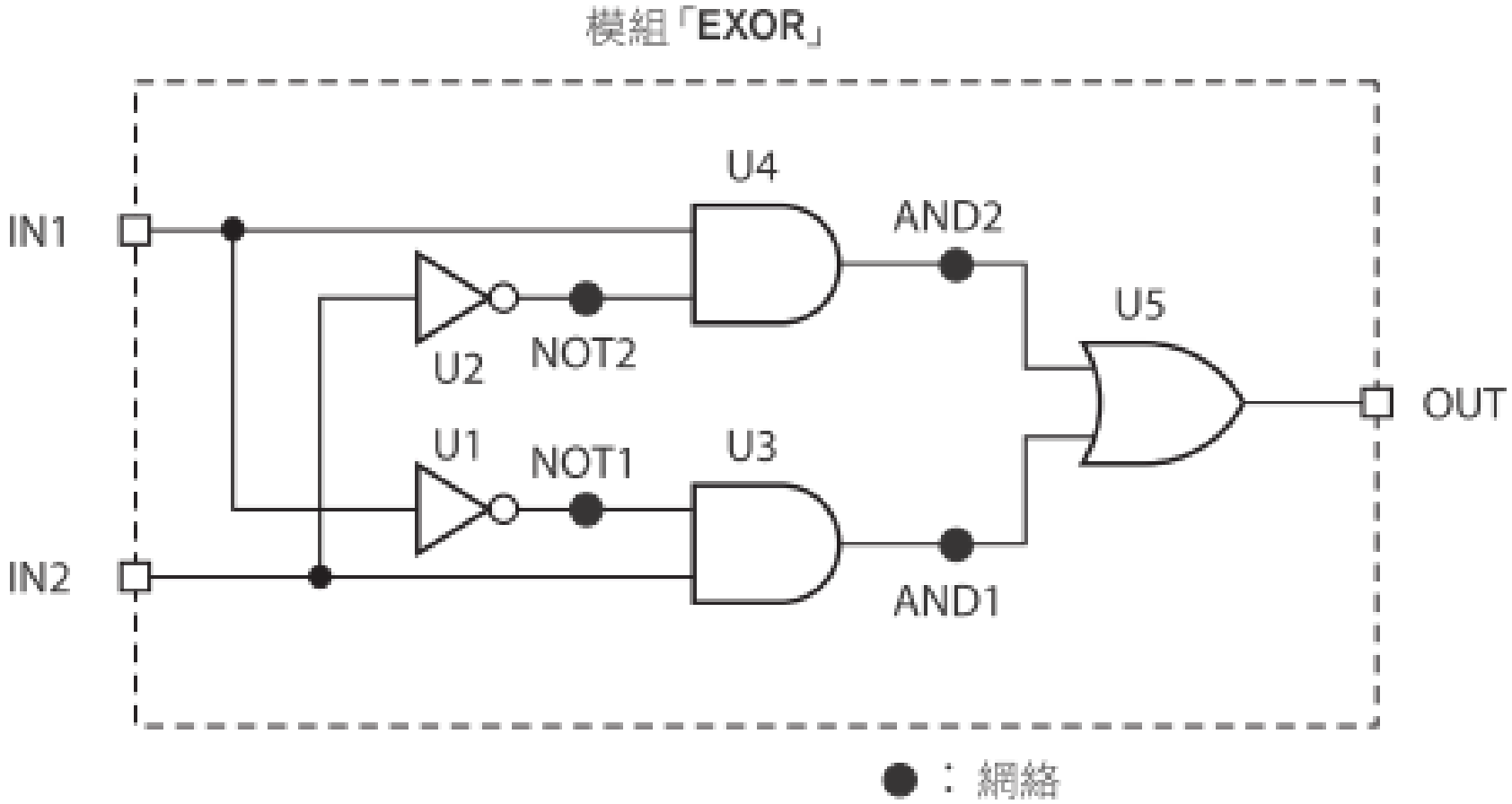
```
/*          EXOR          */
module      EXOR          ( IN1,  IN2,  OUT );
input       IN1,  IN2;
output      OUT;
wire        NOT1,NOT2,AND1,AND2;   宣告網絡
            not           U1            ( NOT1,  IN1 ),
                          U2            ( NOT2,  IN2 );
使用程式    and           U3            ( AND1,  NOT1,  IN2 ),
原生功能                  U4            ( AND2,  NOT2,  IN1 );
            or            U5            ( OUT ,  AND1,  AND2 );
endmodule
```

- NOT、OR、AND 等電路的程式原生功能的語法：

not 元件名　(輸出, 輸入)

```
and   元件名        (輸出, 輸入 1, 輸入 2);
```

```
or    元件名        (輸出, 輸入 1, 輸入 2, 輸入 3);
```



▲ 圖 4.4　互斥或電路的模組定義

- 邏輯閘另種寫法 (使用原生“互斥或”功能)

程式 4.2　互斥或電路的 Verilog HDL 描述 (邏輯閘層)

```
/*          EXOR       */
module      EXOR       ( IN1, IN2, OUT );
input       IN1, IN2;
output      OUT;
            xor        U1    ( OUT, IN1, IN2 );
endmodule
```

使用程式原生功能的互斥或就變得很簡單了

- Verilog HDL 的程式原生功能
  - not
  - buf
  - and
  - nand
  - or
  - nor
  - xor
  - xnor

資料流層

- 用 assign 這個保留字描述組合電路邏輯式

```
OUT = ~IN1 & IN2 | IN1 & ~IN2; (要注意優先順序的高低是 ~ > & > |)
```

- 邏輯模擬器會一直觀察 assign 這行等號右邊的變化, 有任何的變化會反映到等號左邊

程式 4.3　互斥或電路的 Verilog HDL 描述 (資料流層)　LST

```
/*          EXOR        */
module      EXOR        ( IN1, IN2, OUT );
input       IN1, IN2;
output      OUT;
            assign   OUT = ~IN1 & IN2 | IN1 & ~IN2;
endmodule
```

資料流層是根據 assign 來描述邏輯式

- 資料流層也可以直接使用 XOR 位元運算子 '^'：

程式 4.4　互斥或電路的 Verilog HDL 描述 (資料流層)

```
/*          EXOR      */
module      EXOR      ( IN1, IN2, OUT );
input       IN1, IN2;
output      OUT;
            assign    OUT = IN1 ^ IN2;
endmodule
```

- Verilog HDL 的位元運算子的種類與功能

~：NOT　　&：AND

|：OR　　^：XOR

~^：XNOR（XOR、NOT ）

行為層

- 行為層的描述不需要在意電路圖或邏輯式，直接描述電路的動作即可
- 會使用到資料流層用到的 assign 與新的 function 語法

function 的語法

```
function <範圍> 函數名;
    <各種宣告>
    <狀態>
endfunction
```

▲ 圖 4.5 function 的語法

- 以「function」開頭「endfunction」結尾
- 「函數名」為「EXOR_FUNC」
- 「各種宣告」則用來宣告輸入,至少要用 input 寫一個引數
- 「狀態」則是由 if 來描述互斥或的功能

程式 4.5　互斥或電路的 Verilog HDL 描述 (行為層)　　LST4_5.V

```
/*              EXOR      */
module          EXOR ( IN1, IN2, OUT );
input           IN1, IN2;
output          OUT;
                assign   OUT = EXOR_FUNC ( IN1, IN2 );

function        EXOR_FUNC;
input           IN1, IN2;
                if              ( IN1 ^ IN2 )
                                EXOR_FUNC = 1;
                else
                                EXOR_FUNC = 0;
endfunction

endmodule
```

用 assign 呼叫 function

函數名

宣告輸入

用 if 來描述動作

(2) If 的語法

If (<條件>) <狀態 1>

else <狀態 2>

- 滿足條件的話執行狀態 1

  不滿足條件的話就執行狀態 2

- else 之後可省略

  (但是若用在組合電路則不能省略)

- 要用 if 來描述組合電路的話，一定不要忘記 else，否則會自動生成記憶電路

- 條件用到的運算子

邏輯運算子	!	邏輯 NOT
	&&	邏輯 AND
	||	邏輯 OR
相等運算子	==	等於
	!=	不等於
	===	等於 (也會比較 x, z)
	!==	不等於 (也會比較 x, z)
比較運算子	<	小於
	<=	小於等於
	>	大於
	>=	大於等於
移位運算子	<<	左移
	>>	右移

(3) 呼叫函數

- 要在模組裡面呼叫函數要使用 assign, 然後在 '=' 後面寫上「函數名稱」
- 要注意「引數的順序」與「函數內宣告輸入的順序」必須要有對應關係

測試平台

程式 4.6　為了模擬互斥或電路所建的測試平台　LST4_6.V

```
/*          EXOR _ TEST          */
`timescale          1ns/1ns
module      EXOR _ TEST;
reg         [1:0]  IN;
wire        OUT;
integer i;
            EXOR       EXOR       ( IN[0], IN[1], OUT );
            initial  begin
                                   IN = 0;
                  for          ( i = 0; i <= 4; i = i + 1 )
                  begin
                               #100       IN = IN + 1;
                  end
                  $finish;
      end
endmodule
```

匯流排表示法

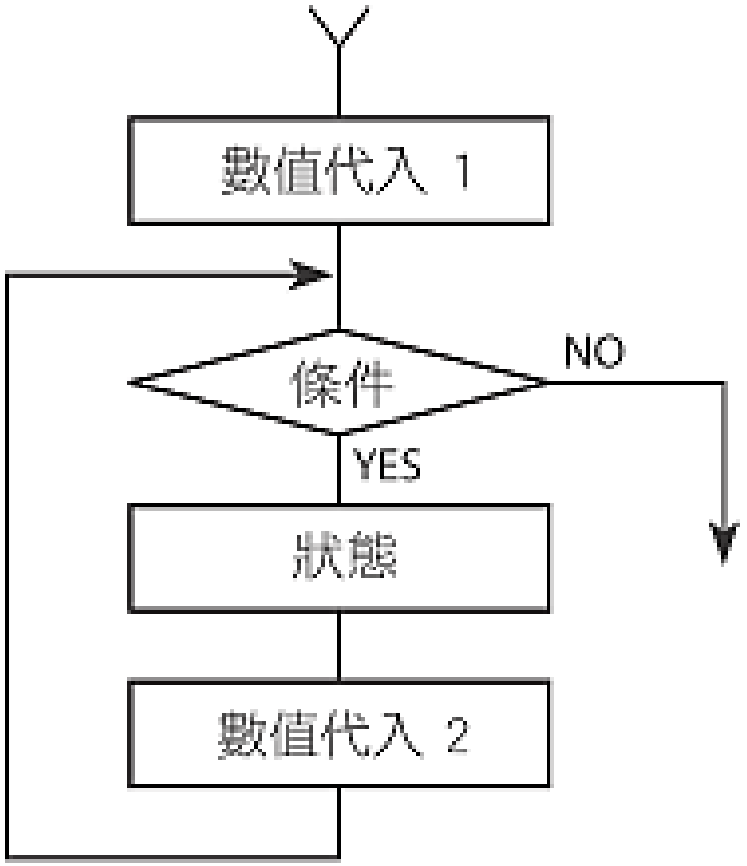
宣告 for 語法用到的 i

for 語法裡面的狀態

接下頁

(1) for 的語法定義

```
for  (<數值代入 1>;<條件>;<數值代入 2>)
                                <狀態>
```



- 從 initial 開始, 把 '0' 代入「IN」
- 在 for 的「數值代入 1」這邊, 把 '0' 代入 'i'
- 因為 i= '0', 所以「條件」的判斷結果為「YES」, 執行「狀態」
- 狀態這邊做的是, 每 100 時間單位之後把 IN+ '1', 所以現在 IN= '1'
- for 語法的狀態部分包含在「begin-end」裡面, 所以接下來要執行「數值代入 2」
- 「數值代入 2」這邊要 i+ '1', 所以現在 i= '1'
- 因為 i= '1', 所以「條件」的判斷結果仍舊是「YES」, 再一次執行「狀態」

(2) 以匯流排表示信號

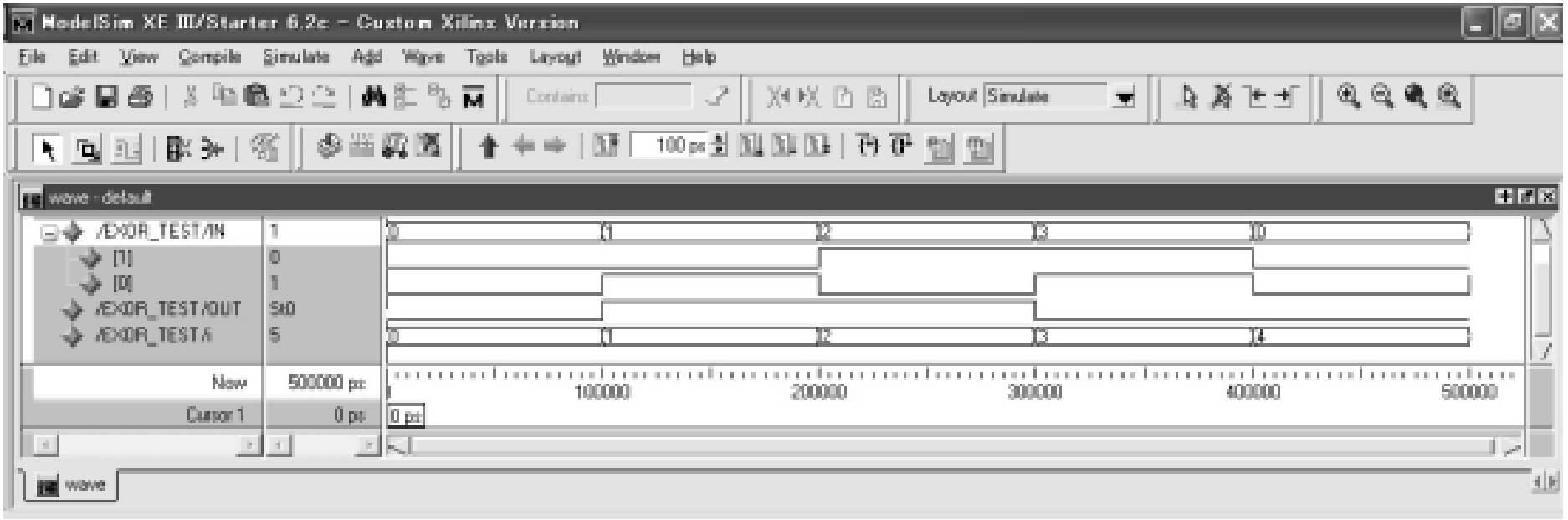
- Verilog HDL 可以透過範圍來同時宣告好幾個位元
- 範圍是以「MSB:LSB」的格式來表示；本例寫成「1:0」，表示包含有「bit_1」與「bit_0」
- 用範圍來表示匯流排的信號, 如果使用 '**信號名「位元數」**' 就可以一次控制一個位元。稱為位元分割

(3) Integer

- 在 Verilog HDL 裡面, 必須把電路及其對應的資料(信號) 與純粹用來描述的資料分開
- 電路及其對應的資料稱為**物質資料型態**, 用 reg 或者 wire 來描述
- 抽象資料型態

integer	整數型態
time	時間型態
real	實數型態
event	事件型態
parameter	參數型態

- 模擬結果



- 模擬結果確認重點：
  - (1)「IN」或者「IN[0]」,「IN[1]」的變化
  - (2) 確認輸出「OUT」的結果

總結

- 邏輯閘層是一個跟實際電路構造很接近的模組。
- 行為層是不考慮電路的內部構造，直接描述電路動作的模組。
- 資料流層介於上述兩種模組之間，是描述組合電路邏輯式的模組。
- 測試平台的目的是「給下層模組必須的信號，觀察輸出的變化」，所以在撰寫的時候不用考慮電路構造

4.2 多工電路

- 多工電路也稱為多工器 (selector)，有選擇信號與多個輸入信號，根據選擇信號的值來決定輸出要對應到哪個輸入。

4.2.1 真值表與邏輯式

2-1 多工器

- 2-1 多工器是最簡單的多工器

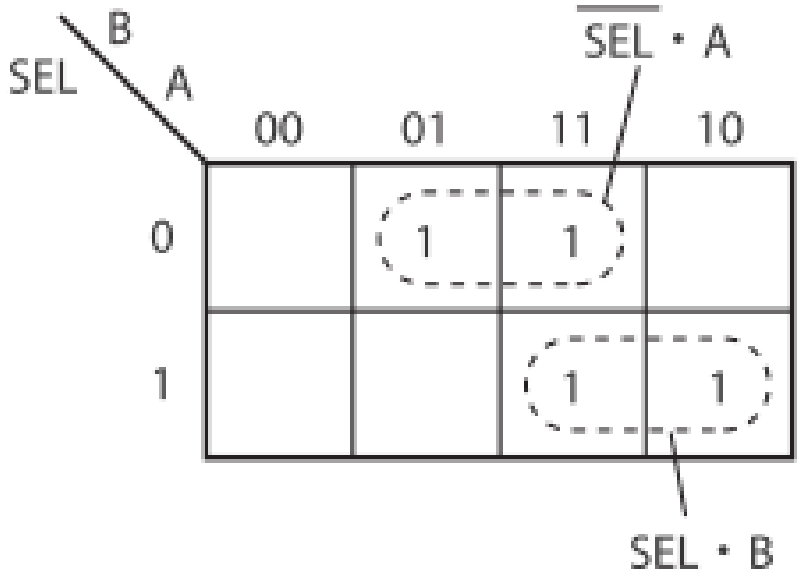
$$OUT = \overline{SEL} \cdot A + SEL \cdot B$$

SEL	B	A	OUT
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

SEL = '0' 的話 OUT = A

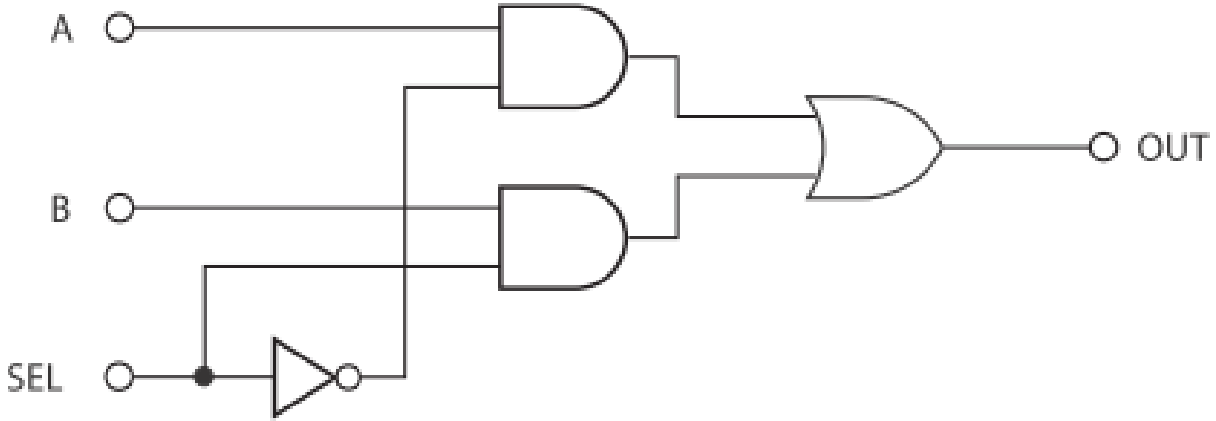
SEL = '1' 的話 OUT = B

▲ 表 4.2 多工器的真值表

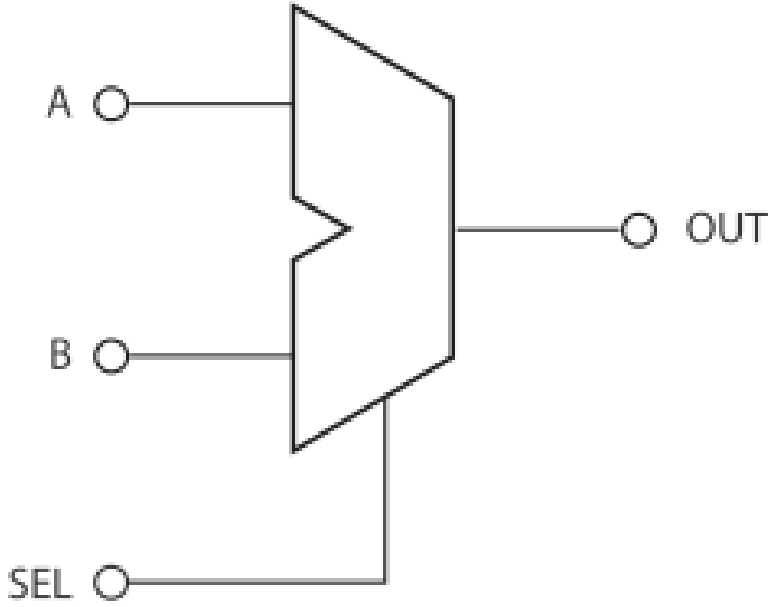


$$OUT = \overline{SEL} \cdot A + SEL \cdot B$$

▲ 圖 4.10 多工器的卡諾圖



▲ 圖 4.11　多工器的電路圖



▲ 圖 4.12　2-1 多工器的電路符號

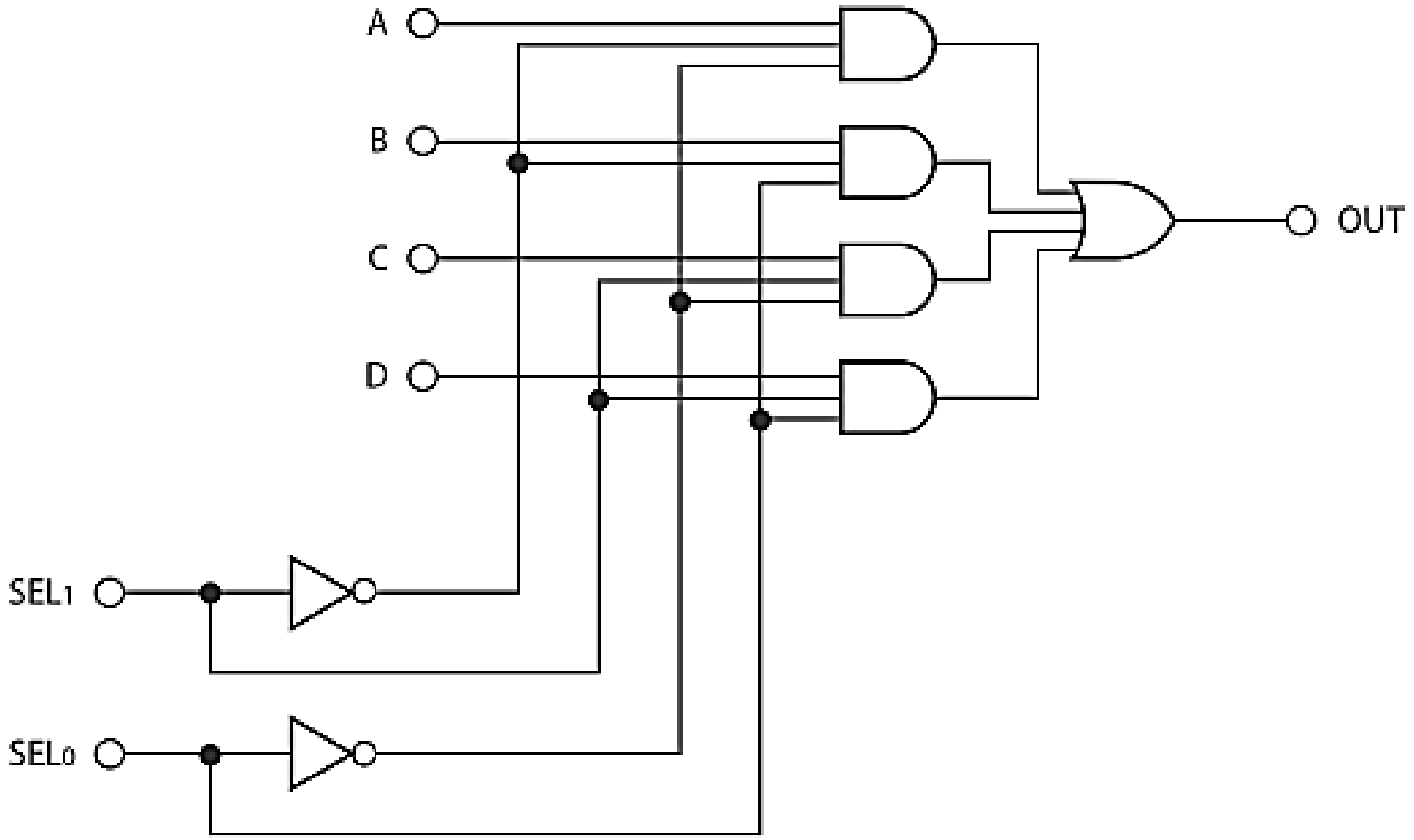
4-1 多工器

- 4-1 多工器的選擇信號需要 2 位元, 分別是「SEL_0」,「SEL_1」, 輸入為 A~D；因為輸入有 6 位元, 2_6 所以有 64 列之多

SEL_1	SEL_0	D	C	B	A	OUT
0	0	0	0	0	0	0
0	0	0	0	0	1	1
1	1	1	1	1	1	1

- SEL1 = '0', SEL0 = '0' 的話 OUT = A
- SEL1 = '0', SEL0 = '1' 的話 OUT = B
- SEL1 = '1', SEL0 = '0' 的話 OUT = C
- SEL1 = '1', SEL0 = '1' 的話 OUT = D
- 將上述 4 個邏輯關係, 全部 OR 起來的結果：

$$OUT = \overline{SEL_1} \cdot \overline{SEL_0} \cdot A + \overline{SEL_1} \cdot SEL_0 \cdot B + SEL_1 \cdot \overline{SEL_0} \cdot C + SEL_1 \cdot SEL_0 \cdot D$$



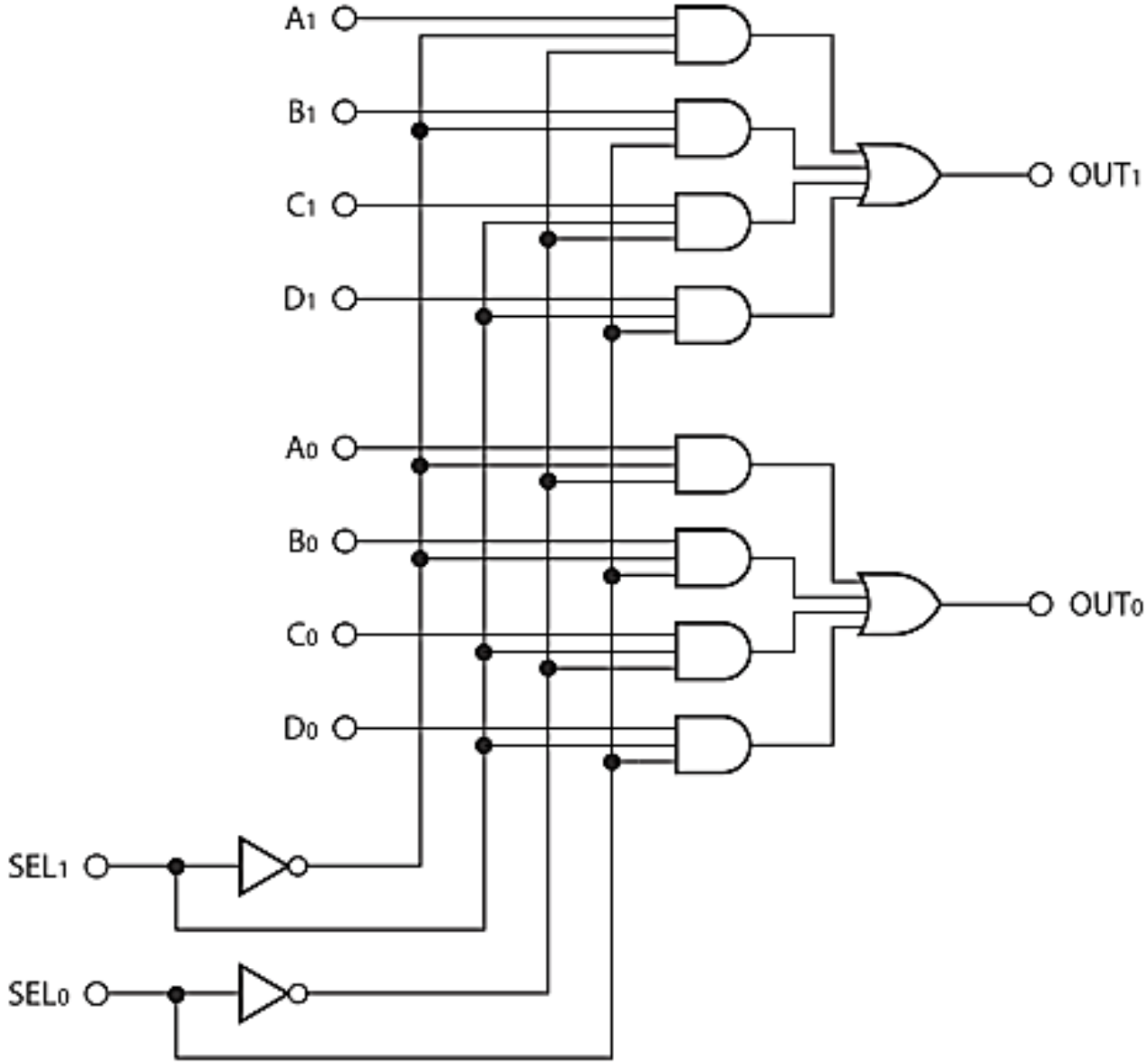
▲ 圖 4.13　4-1 多工器

4-2 多工器

- 從 4 組各 2 位元的輸入中選出 2 位元給輸出
- 兩個輸出的邏輯式如下：

$$OUT_0 = \overline{SEL_1} \cdot \overline{SEL_0} \cdot A_0 + \overline{SEL_1} \cdot SEL_0 \cdot B_0 + SEL_1 \cdot \overline{SEL_0} \cdot C_0 + SEL_1 \cdot SEL_0 \cdot D_0$$

$$OUT_1 = \overline{SEL_1} \cdot \overline{SEL_0} \cdot A_1 + \overline{SEL_1} \cdot SEL_0 \cdot B_1 + SEL_1 \cdot \overline{SEL_0} \cdot C_1 + SEL_1 \cdot SEL_0 \cdot D_1$$



▲ 圖 4.14　4-2 多工器

4.2.2 多工電路的 Verilog HDL 描述

邏輯閘層

- (1) 2-1 多工器

程式 4.7　2-1 多工器的 Verilog HDL 描述 (邏輯閘層)

```
/*          2-1 SELECTOR          */
module      SEL  ( A, B, SEL, OUT );
input       A, B, SEL;
output      OUT;
wire SEL_NOT, AND1, AND2;
            not   U1          ( SEL_NOT, SEL );
            and   U2          ( AND1, B, SEL ),
                  U3          ( AND2, A, SEL_NOT );
            or    U4          ( OUT , AND1, AND2 );
endmodule
```

- (2) 4-1 多工器

```
wire     SEL1_NOT, SEL0_NOT, AND1, AND2, AND3, AND4;
         not       U1        ( SEL1_NOT, SEL[1] ),
                   U2        ( SEL0_NOT, SEL[0] );
         and       U3        ( AND1, A, SEL1_NOT, SEL0_NOT ),
                   U4        ( AND2, B, SEL1_NOT, SEL[0] ),
                   U5        ( AND3, C, SEL[1]  , SEL0_NOT ),
                   U6        ( AND4, D, SEL[1]  , SEL[0] );
         or        U7        ( OUT , AND1, AND2, AND3, AND4 );
```

- (3) 4-2 多工器

```
wire    SEL1_NOT, SEL0_NOT;
wire    AND_A1, AND_B1, AND_C1, AND_D1;
wire    AND_A0, AND_B0, AND_C0, AND_D0;
        not         U1          ( SEL1_NOT, SEL[1] ),
                    U2          ( SEL0_NOT, SEL[0] );
        and         UAND_A1     ( AND_A1, A[1], SEL1_NOT, SEL0_NOT ),
                    UAND_B1     ( AND_B1, B[1], SEL1_NOT, SEL[0] ),
                    UAND_C1     ( AND_C1, C[1], SEL[1]  , SEL0_NOT ),
                    UAND_D1     ( AND_D1, D[1], SEL[1]  , SEL[0] );
        and         UAND_A0     ( AND_A0, A[0], SEL1_NOT, SEL0_NOT ),
                    UAND_B0     ( AND_B0, B[0], SEL1_NOT, SEL[0] ),
                    UAND_C0     ( AND_C0, C[0], SEL[1]  , SEL0_NOT ),
                    UAND_D0     ( AND_D0, D[0], SEL[1]  , SEL[0] );
        or          OUT1        ( OUT[1] , AND_A1, AND_B1, AND_C1, AND_D1 ),
                    OUT0        ( OUT[0] , AND_A0, AND_B0, AND_C0, AND_D0 );
```

資料流層

- (1) 2-1 多工器

程式 4.10　2-1 多工器的 Verilog HDL 描述 (資料流層)

```
/*          2-1 SELECTOR          */
module      SEL         ( A,  B,  SEL,  OUT );
input       A,  B,  SEL;
output      OUT;

            assign    OUT = ~SEL & A |  SEL & B;

endmodule
```

- (2) 4-1 多工器

程式 4.11　4-1 多工器的 Verilog HDL 描述 (資料流層)

```
/*          4-1 SELECTOR          */
module   SEL        ( A, B, C, D, SEL, OUT );
input    A, B, C, D;
input    [1:0] SEL;
output   OUT;

         assign   OUT = ~SEL[1] & ~SEL[0] & A
                        |~SEL[1] &  SEL[0] & B
                        | SEL[1] & ~SEL[0] & C
                        | SEL[1] &  SEL[0] & D;

endmodule
```

- (3) 4-2 多工器

程式 4.12　4-2 多工器的 Verilog HDL 描述 (資料流層)

```
/*          4-2 SELECTOR          */
module   SEL  ( A, B, C, D, SEL, OUT );
input    [1:0] A, B, C, D, SEL;
output   [1:0]OUT;

         assign   OUT[1] = ~SEL[1] &~SEL[0] & A[1]
                          |~SEL[1] &  SEL[0] & B[1]
                          | SEL[1] & ~SEL[0] & C[1]
                          | SEL[1] &  SEL[0] & D[1];

         assign   OUT[0] = ~SEL[1] &~SEL[0] & A[0]
                          |~SEL[1] &  SEL[0] & B[0]
                          | SEL[1] & ~SEL[0] & C[0]
                          | SEL[1] &  SEL[0] & D[0];

endmodule
```

條件運算子

- (1) 2-1 多工器
  - 條件運算子的語法是「(條件)? 狀態 1: 狀態 2;」
  - 應用到 2-1 多工器可寫成：

```
assign   OUT = ( SEL == 0 )? A: B;
```

- (2) 4-1 多工器
  - 若使用條件運算子表示 4-1 多工器，需要使用到巢狀 (nesting) 結構

```
OUT = (SEL[0] == 0)? 狀態 1: 狀態 2 ;
    狀態 1: (SEL[0] == 0)? A : B
    狀態 2: (SEL[0] == 0)? C : D
```

  - 實際寫法為：

```
OUT = ( SEL[1] == 0 ) ?
        (( SEL[0] == 0 ) ? A: B ): (( SEL[0] == 0 )? C: D );
```

- (3) 4-2 多工器
  - 如同 4-1 多工器
  - 省略表示匯流排的「範圍」的話就表示信號整體
  - 實際寫法如下：

```
OUT = ( SEL[1] == 0 ) ?
        (( SEL[0] == 0 ) ? A: B ): (( SEL[0] == 0 )? C: D );
```

行為層

- (1) 2-1 多工器
  - 用 if 語法, 根據「條件的真/假」來執行「狀態 1」或者「狀態 2」
  - 實際寫法：

```
if ( SEL == 0 ) SEL2_1_FUNC = A;
          else  SEL2_1_FUNC = B;
```

- 用 case 語法描述, 將 SEL 當成「判斷式」, case item 為 0 的時候為將 A 丟給回傳值, case item 為 1 的時候將 B 丟給回傳值。

```
function      SEL2_1_FUNC;
input         A, B, SEL;
              case ( SEL )
                            0:SEL2_1_FUNC = A;
                            1:SEL2_1_FUNC = B;
              endcase
endfunction
```

```
case (<判斷式>)  case item  endcase
case item
        <項目> : 狀態
        default : 狀態
        case item 可以寫超過一個
```

▲ 圖 4.15　case 的語法定義

- (2) 4-1 多工器
  - If 語法

```
function     SEL4_1_FUNC;
input        A, B, C, D;
input        [1:0] SEL;
             if ( SEL[1] == 0 )
                      if ( SEL[0] == 0 )
                               SEL4_1_FUNC = A;
                      else
                               SEL4_1_FUNC = B;
             else
                      if ( SEL[0] == 0 )
                               SEL4_1_FUNC = C;
             else
                               SEL4_1_FUNC = D;
endfunction
```

- (2) 4-1 多工器
  - case 語法

```
function      SEL4_1_FUNC;
input         A, B, C, D;
input         [1:0] SEL;
              case ( SEL )
                              0:SEL4_1_FUNC = A;
                              1:SEL4_1_FUNC = B;
                              2:SEL4_1_FUNC = C;
                              3:SEL4_1_FUNC = D;
          endcase
endfunction
```

- (3) 4-2 多工器
  - 和 4-1 多工器略有不同
  - Input 用匯流排方式表示輸入 A~D
  - Output 用匯流排方式表示輸出 OUT
  - Function 用範圍表示

- if 語法

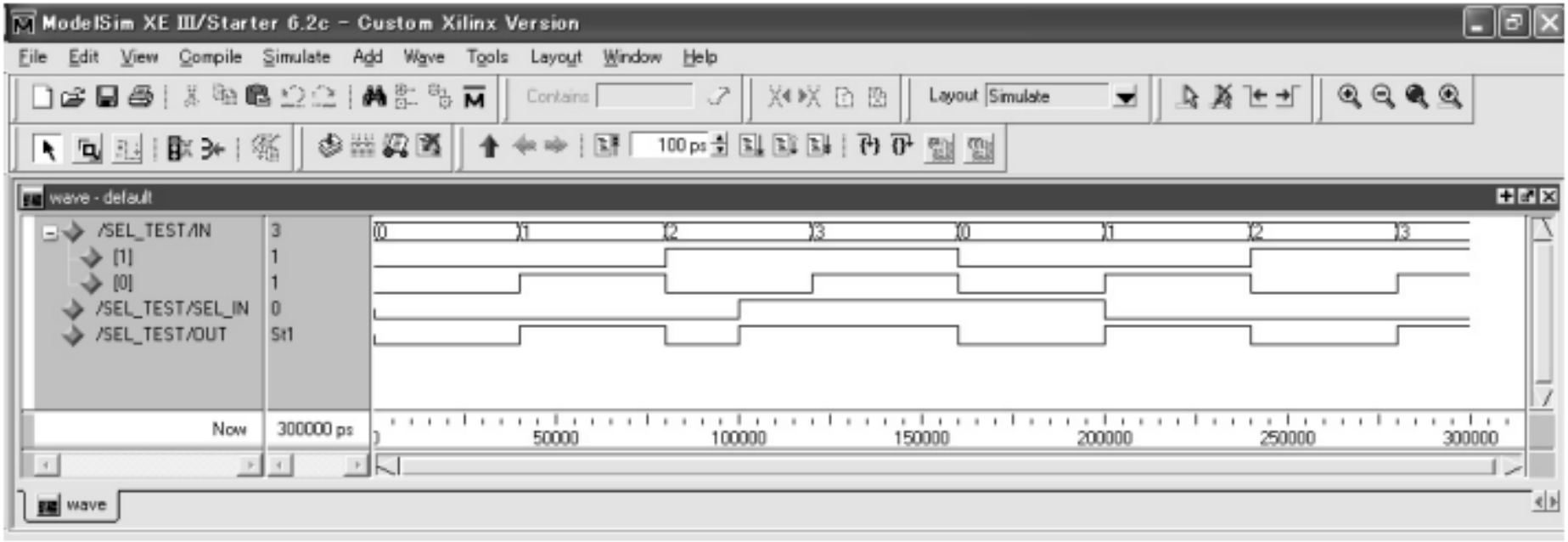
```
function     [1:0] SEL4_2_FUNC;
input        [1:0] A, B, C, D;
input        [1:0] SEL;
             if ( SEL[1] == 0 )
                          if ( SEL[0] == 0 )
                                       SEL4_2_FUNC = A;
                          else
                                       SEL4_2_FUNC = B;
             else if ( SEL[0] == 0 )
                                       SEL4_2_FUNC = C;
                     else
                                       SEL4_2_FUNC = D;
endfunction
```

- case 語法

```
function      [1:0] SEL4_2_FUNC;
input         [1:0] A, B, C, D;
input         [1:0] SEL;
              case ( SEL )
                     0:SEL4_2_FUNC = A;
                     1:SEL4_2_FUNC = B;
                     2:SEL4_2_FUNC = C;
                     3:SEL4_2_FUNC = D;
              endcase
endfunction
```

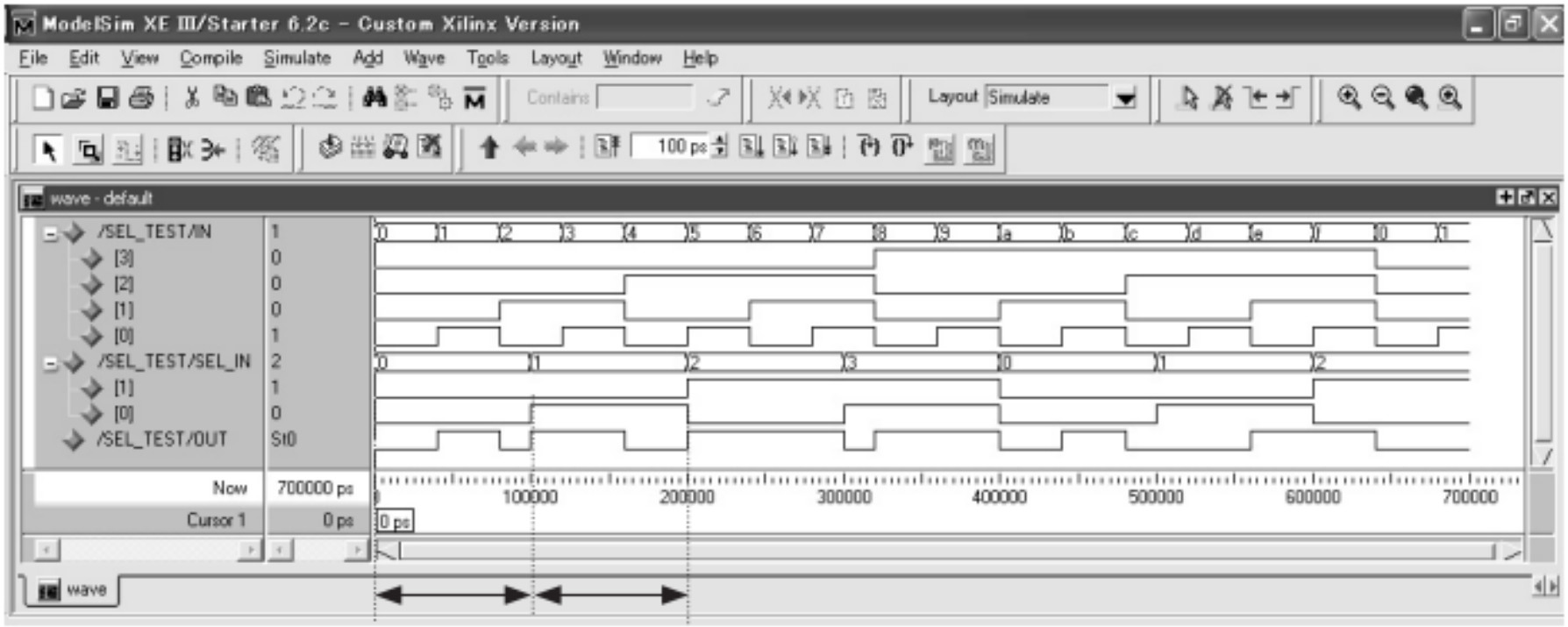
模擬結果

- (1) 2-1 多工器



▲ 圖 4.16　2-1 多工器的模擬結果

- (2) 4-1 多工器

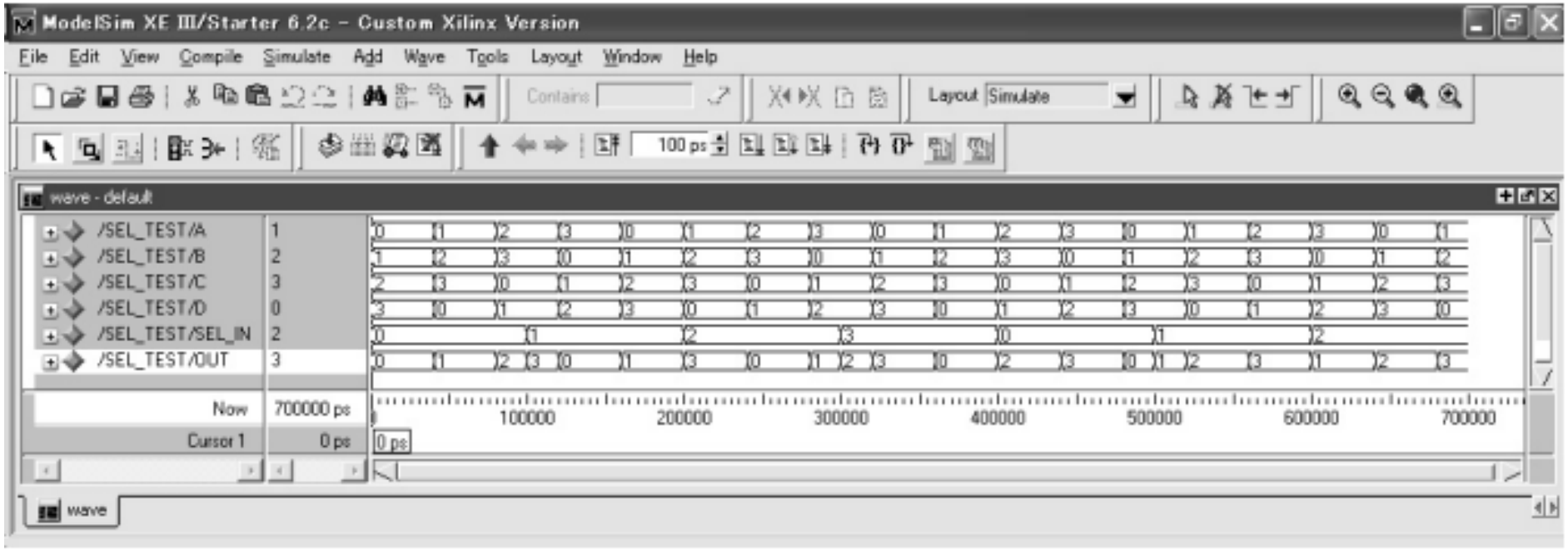


這一段時間因為 SEL_IN = '**0**' 所以 OUT = IN[0]

這一段時間因為 SEL_IN = '**1**' 所以 OUT = IN[1]

▲圖 4.17　4-1 多工器的模擬結果

- (3) 4-2 多工器



▲ 圖 4.18　4-2 多工器的模擬結果

4.3 比較電路
4.3.1 真值表與卡諾圖

- 比較電路 (comparator) 有分兩種，一種是比較兩個資料大小關係的電路，另外一種是檢查兩個資料是否一致的一致電路

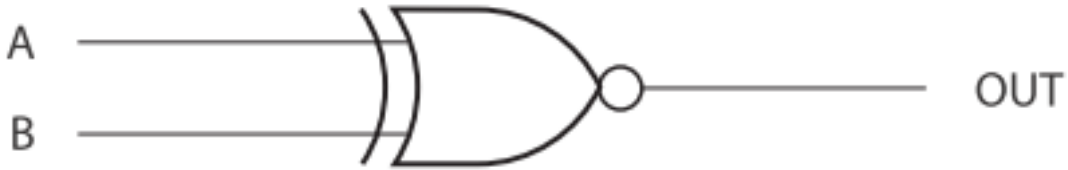
雙輸入的一致電路

- 與 XOR 剛好相反，相當於 XNOR 電路

B	A	OUT
0	0	1
0	1	0
1	0	0
1	1	1

這個真值表與 XOR $A \oplus B$ 的輸出剛好相反

▲ 表 4.5　一致電路的真值表



▲ 圖 4.19　一致電路

2 位元比較電路

- 輸出根據「X=Y」,「X>Y」與「X<Y」的狀況, 有以下三種結果：

X = Y: X = Y 的話輸出為 '1', X≠Y 的話輸出為 '0'
X > Y: X > Y 的話輸出為 '1', X≦Y 的話輸出為 '0'
X < Y: X < Y 的話輸出為 '1', X≧Y 的話輸出為 '0'

- 邏輯式如下：

$$\begin{aligned}
「X=Y」 =& \overline{X_1} \cdot \overline{X_0} \cdot \overline{Y_1} \cdot \overline{Y_0} \\
&+ \overline{X_1} \cdot X_0 \cdot \overline{Y_1} \cdot Y_0 \\
&+ X_1 \cdot \overline{X_0} \cdot Y_1 \cdot \overline{Y_0} \\
&+ X_1 \cdot X_0 \cdot Y_1 \cdot Y_0
\end{aligned}$$

$$「X>Y」 = X_0 \cdot \overline{Y_1} \cdot \overline{Y_0} + X_1 \cdot X_0 \cdot \overline{AY_0} + X_1 \cdot \overline{Y_1}$$

$$「X<Y」 = \overline{X_0} \cdot Y_1 \cdot Y_0 + \overline{X_1} \cdot \overline{X_0} \cdot Y_0 + \overline{X_1} \cdot Y_1$$

- 「X=Y」的目的只是要檢查 X 是否等於 Y, 可使用兩個一致電路來對「X1」與「Y1」,「X0」與「Y0」來做比較,致電路來對「X1」與「Y1」,「X0」與「Y0」來做比較, 再把其結果做 AND 即可：

$$「X=Y」= \overline{(X_1 \oplus Y_1)} \cdot \overline{(X_0 \oplus Y_0)}$$

4.3.2 比較電路的 Verilog HDL 描述

- 邏輯閘層的描述請直接使用程式原生功能

資料流層

$[X=Y] \rightarrow EQ, \quad [X>Y] \rightarrow LG, \quad [X<Y] \rightarrow SM$

```
assign    LG = X[0] & ~Y[1] & ~Y[0]
                | X[1] & X[0] & ~Y[0]
                | X[1] & ~Y[1];
          assign EQ = ( X[1] ~^ Y[1] ) & ( X[0] ~^ Y[0] );
          assign SM = ~X[0] & Y[1] & Y[0]
                | ~X[1] & ~X[0] & Y[0]
                | ~X[1] & Y[1];
```

行為層

- 使用 if 與關係運算子來輸出 X, Y 的大小關係到 [EQ], [LG] 與 [SM]

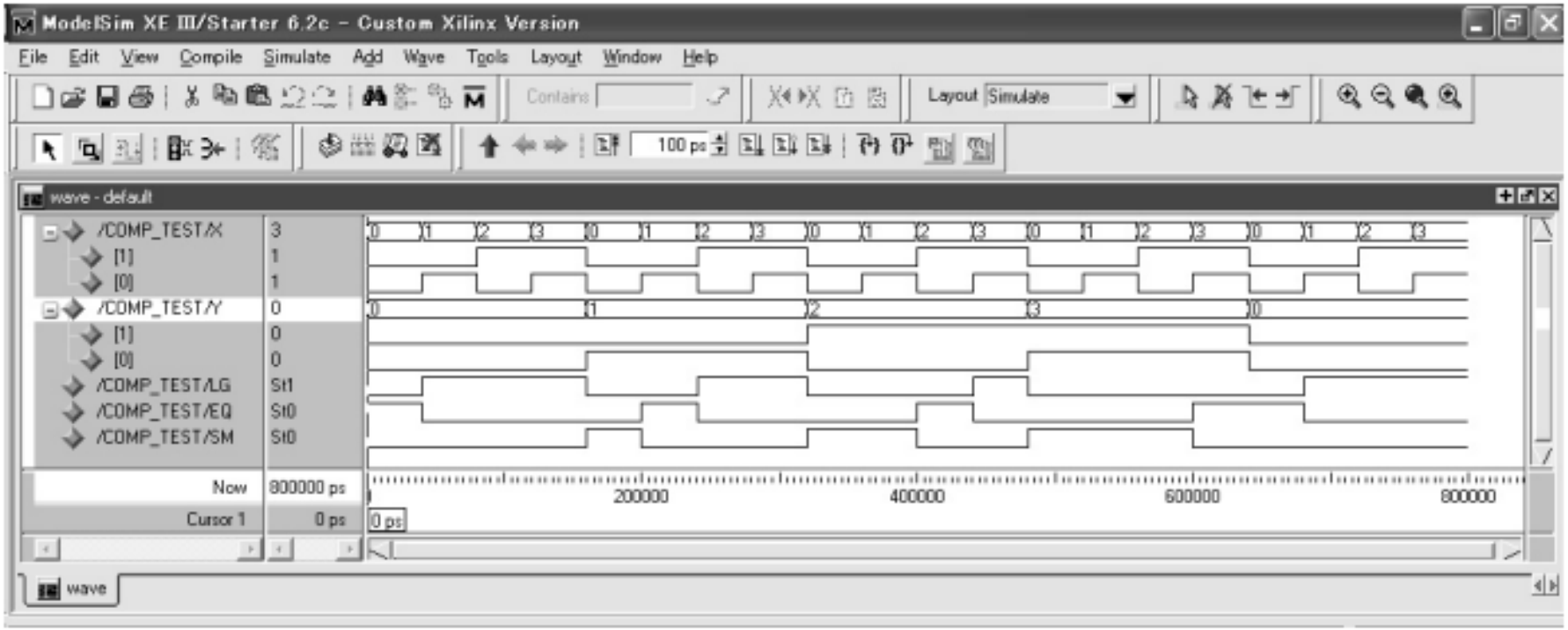
程式 4.25　2 位元比較電路的 Verilog HDL 描述 (行為層)　LST4_25.V

```
/*          2bit COMPARATER       */
module      COMP        ( X, Y, LG, EQ, SM );
input       [1:0] X, Y;
output      LG, EQ, SM;
            assign      { LG, EQ, SM } = FUNC _ COMP ( X, Y );
                          連接運算子
function    [2:0] FUNC _ COMP;
input       [1:0] X, Y;
            if ( X > Y )
                                  FUNC _ COMP = 3'b100;
                                                LG 為 '1'
            else if ( X < Y )
                                  FUNC _ COMP = 3'b001;
                                                SM 為 '1'
                    else
                                  FUNC _ COMP = 3'b010;
                                                EQ 為 '1'
endfunction

endmodule
```

數值定數

模擬結果



▲ 圖 4.22　2 位元比較電路的模擬結果

4.3.3 更多輸入的比較電路

- 位元數更高的比較電路，無法使用先前「真值表 → 卡諾圖 → 邏輯式 → 電路圖」的順序來做設計
- 可從以下的 1 位元比較電路來做延伸

(1) 1 位元比較電路

- 輸出有「LG(X>Y)」,「EQ(X=Y)」與「SM(X<Y)」三個位元的結果

$$LG = \overline{Y} \cdot X$$

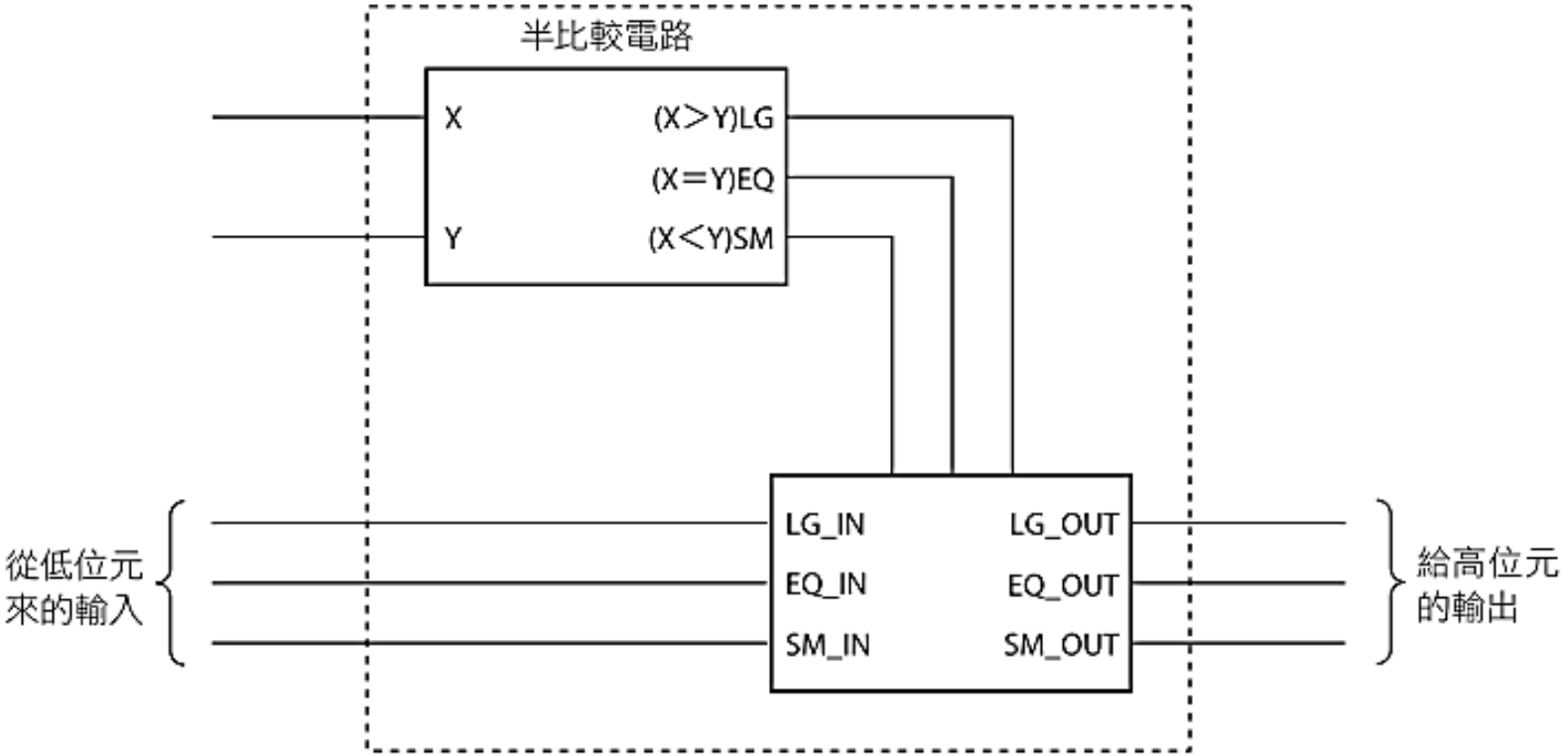
$$EQ = \overline{Y \oplus X}$$

$$SM = Y \cdot \overline{X}$$

- 以下將1位元比較電路先稱為半比較電路

(2) 半比較電路的功能強化

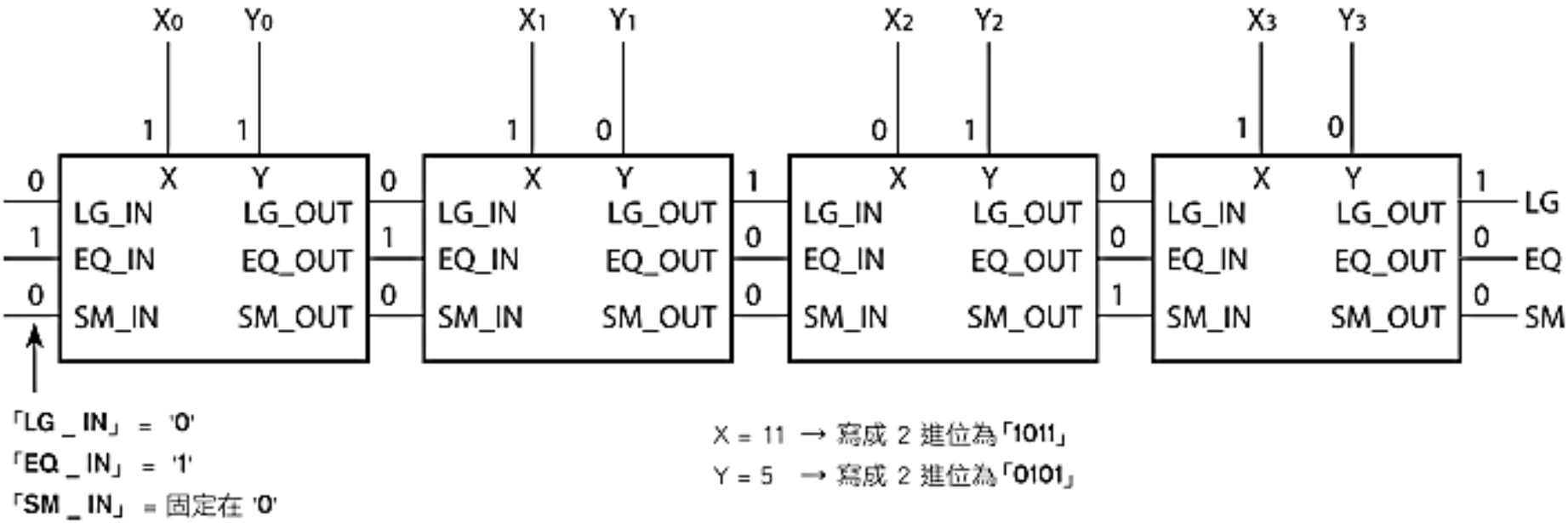
- 由低位元將大小關係輸入, 由這些信號與半比較電路的輸出作為給高位元的信號,讓半比較電路可以支援到複數位元



▲圖 4.24　全比較電路

(3) 用全比較電路實現 4 位元比較電路

- 4 位元比較電路可以用全比較電路串接的構成



▲ 圖 4.26 4 位元比較電路

4.3.4 全比較電路及 4 位元比較電路的 Verilog HDL 描述

- 模組「COMP」透過下層模組的「FULL_COMP」改變元件名且同時做了 4 次元件宣告

```
module   COMP          ( X, Y, LG_OUT, EQ_OUT, SM_OUT );
input    [3:0] X, Y;
output   LG_OUT, EQ_OUT, SM_OUT;
wire     [2:0] LG, EQ, SM;
         FULL_COMP COMP0 ( X[0], Y[0], 1'b0, 1'b1, 1'b0,
                                       LG[0], EQ[0], SM[0] ),
                   COMP1 ( X[1], Y[1], LG[0], EQ[0], SM[0],
                                       LG[1], EQ[1], SM[1] ),
                   COMP2 ( X[2], Y[2], LG[1], EQ[1], SM[1],
                                        LG[2], EQ[2], SM[2] ),
                   COMP3 ( X[3], Y[3], LG[2], EQ[2], SM[2],
                                        LG_OUT, EQ_OUT, SM_OUT );
endmodule
```

模組「FULL_COMP」元件宣告

- 如果不對全比較電路做元件宣告, 4 位元比較電路跟 2 位元比較電路只差在 X, Y 的範圍不同而已

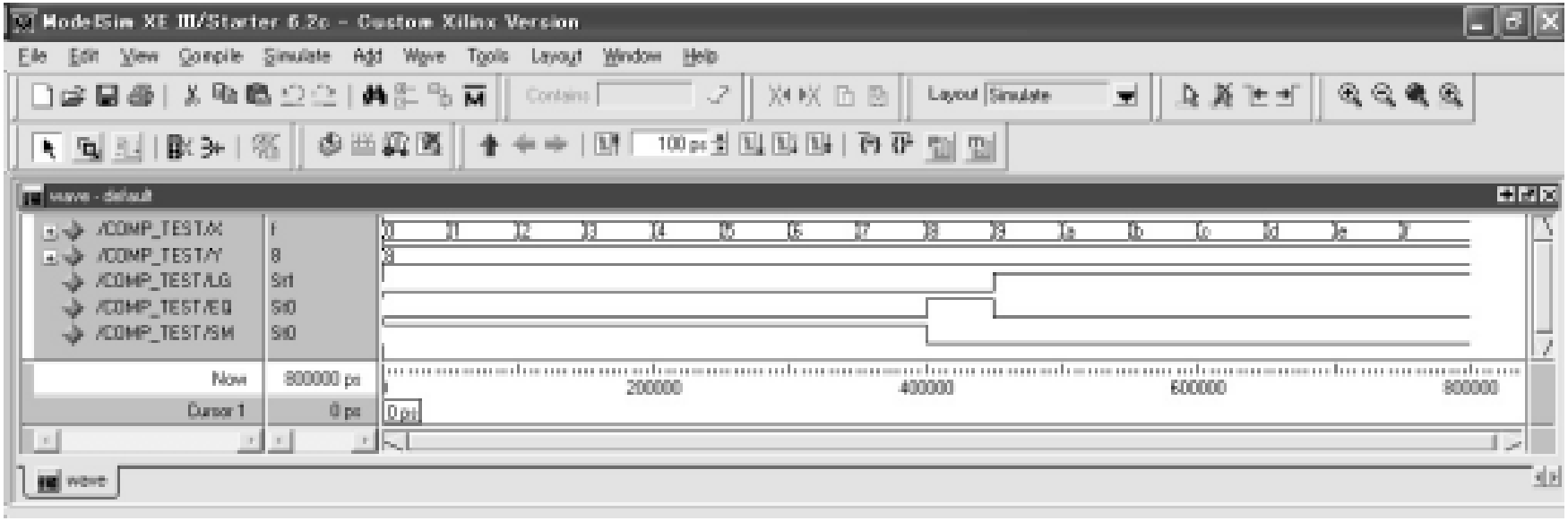
程式 4.28　4 位元比較電路的 Verilog HDL 描述 (行為層)　LST4_28.V

```
/*          4bit COMPARATER                */
module      COMP            ( X, Y, LG, EQ, SM );
input       [3:0] X, Y;
output      LG, EQ, SM;
            assign          { LG, EQ, SM } = FUNC_COMP ( X, Y );

function    [2:0] FUNC_COMP;
input       [3:0] X, Y;
            if ( X > Y )
                                    FUNC_COMP = 3'b100;
            else            if ( X < Y )
                                    FUNC_COMP = 3'b001;
                            else
                                    FUNC_COMP = 3'b010;
endfunction

endmodule
```

模擬結果



▲圖 4.27　4 位元比較電路的模擬結果