正確學會

改訂新版 ディジタル回路と Verilog HDL

Verilog 的16堂課

第 6 章

正反器 (Flip Flop)

本章重點

- 6.9 非同步 R + 同步 JK 正反器
- 6.10 閂鎖電路
- 6.11 assign 與 always 的執行時間
- 6.12 正反器的傳遞延遲時間

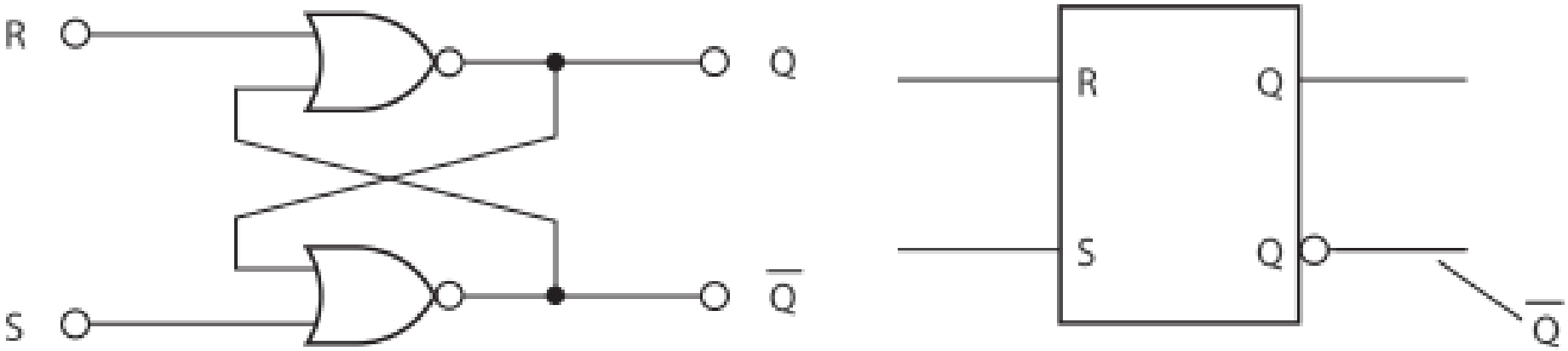
組合電路與序向電路的差異

- 組合電路
  - 由邏輯閘電路所構成，根據給定的輸入有固定的輸出。
  - 代表電路：多工器與解碼器
- 序向電路
  - 由邏輯閘電路與記憶電路所構成，輸出與電路的下一個狀態會由輸入與記憶電路的值來決定。
  - 代表電路：計數器、位移器
- 記憶電路是由正反器所組成，分為沒有時脈的非同步型，以及有時脈的同步型兩種

6.1 非同步型 RS 正反器

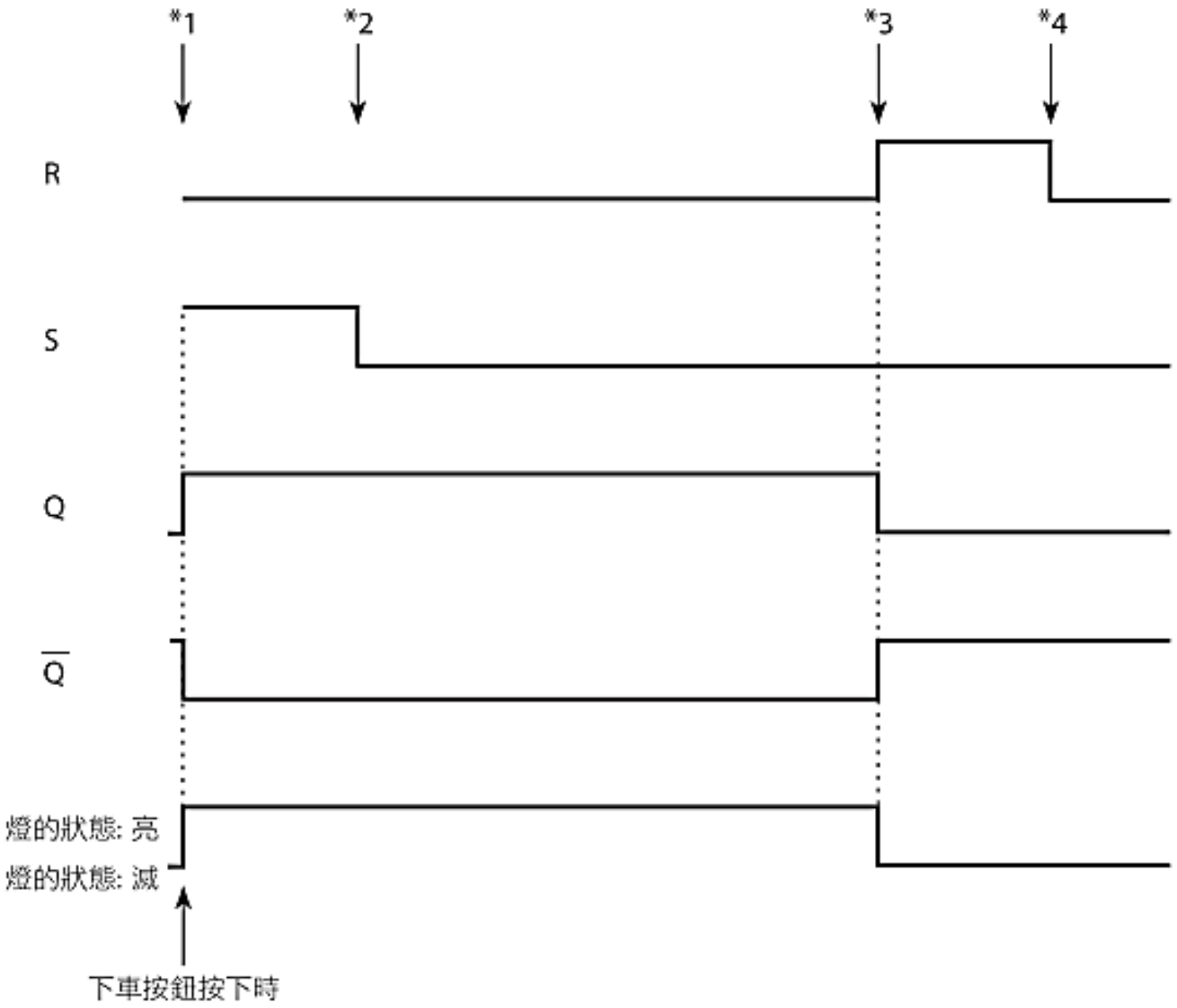
6.1.1 動作與電路符號

- 由兩個 NOR 電路的輸出接到彼此的輸入所構成



▲圖 6.1　非同步型 RS 正反器

1. R = '1', S= '0' 時,輸出 Q 被重置為 '0'
2. R = '0', S= '1' 時,輸出 Q 被設置為 '1'
3. R = '0', S= '0' 時, Q 與 $\overline{Q}$ 都沒有變化



▲圖 6.3　非同步型正反器的時序圖

- 結論：非同步 RS 正反器
  - R 為 ON 的狀態即會重置 Q
  - S 為 ON 的狀態即會設置 Q
  - R 跟 S 都 OFF 的話則會保持之前的狀態
- RS 正反器就是「Reset · Set Flip Flop」的簡稱
- 但R與S不能同時為'1'，否則輸出Q與$\overline{Q}$會同時為'0'。所以，R與S同時為'1'的情況為禁止輸入。

6.1.2 特性表與特性方程式

- 特性方程式 (characteristic equation) 是「為求得正反器下一個狀態的邏輯式」

(1) 特性表的作成

- 左側填入 R 與 S 以及現在正反器的狀態
- 右側填入正反器下一個狀態

S	R	Q	Q'	
0	0	0	0	沒有變化
0	0	1	1	
0	1	0	0	重置
0	1	1	0	
1	0	0	1	設置
1	0	1	1	
1	1	0	—	禁止輸入
1	1	1	—	

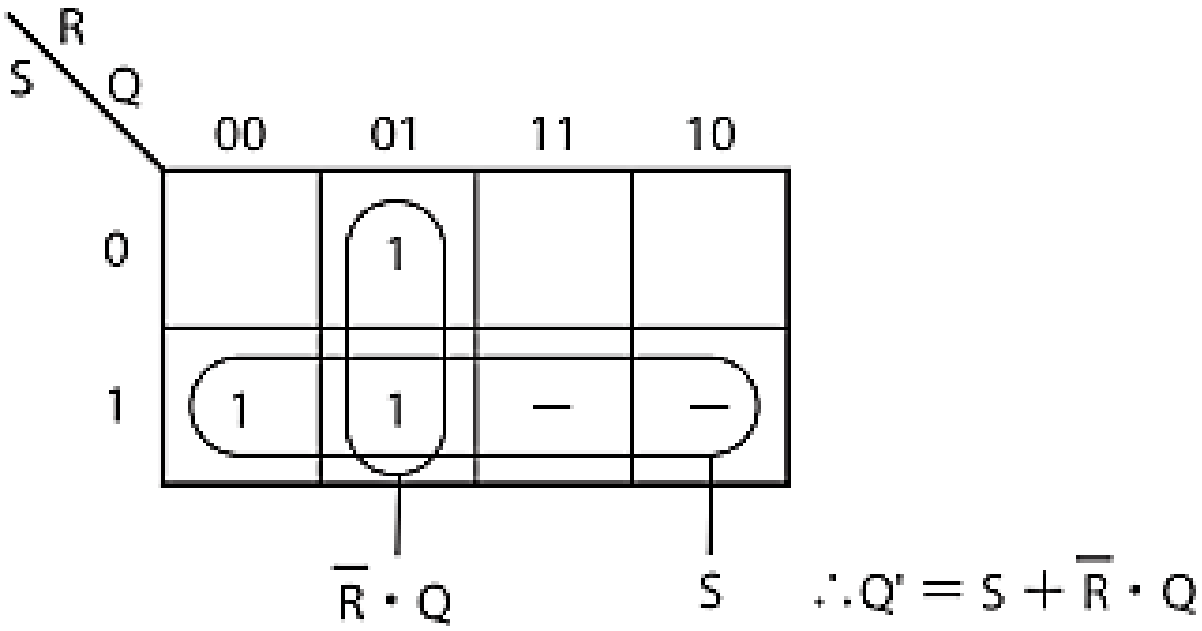
▲ 表 6.1　RS 正反器的特性表

(2) 特性方程式的算法

- 用卡諾圖算出特性方程式

RS 正反器的特性方程式

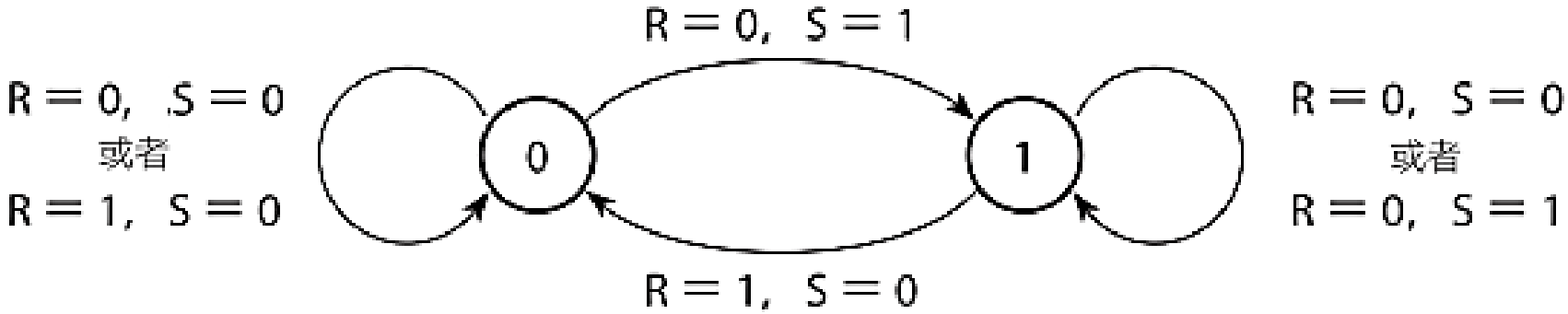
$Q' = S + \overline{R} \cdot Q$



▲ 圖 6.5 RS 正反器的特性方程式

6.1.3 狀態圖

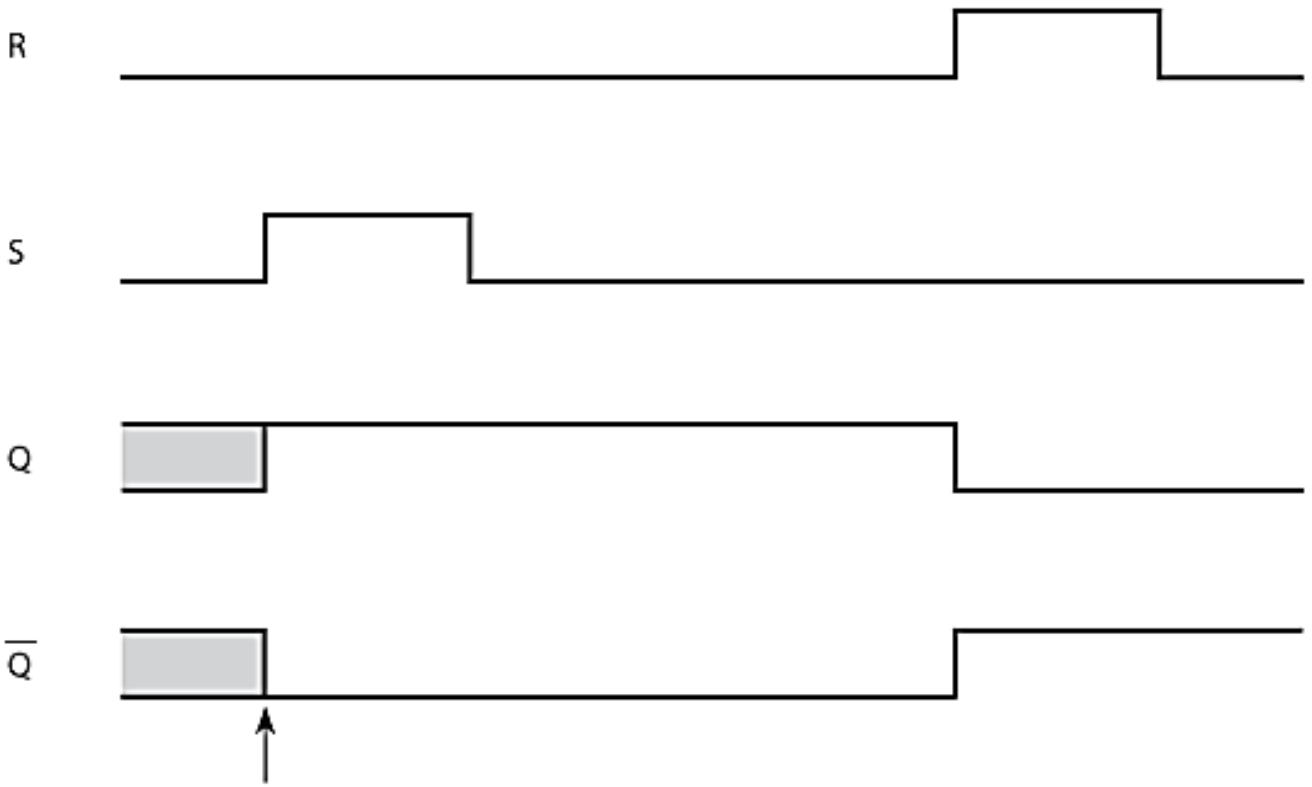
- 狀態圖就是圖形化正反器狀態變化（遷徙）的原因與移動狀態的圖形



▲ 圖 6.6 RS 正反器的狀態圖

6.1.4 正反器的初始狀態

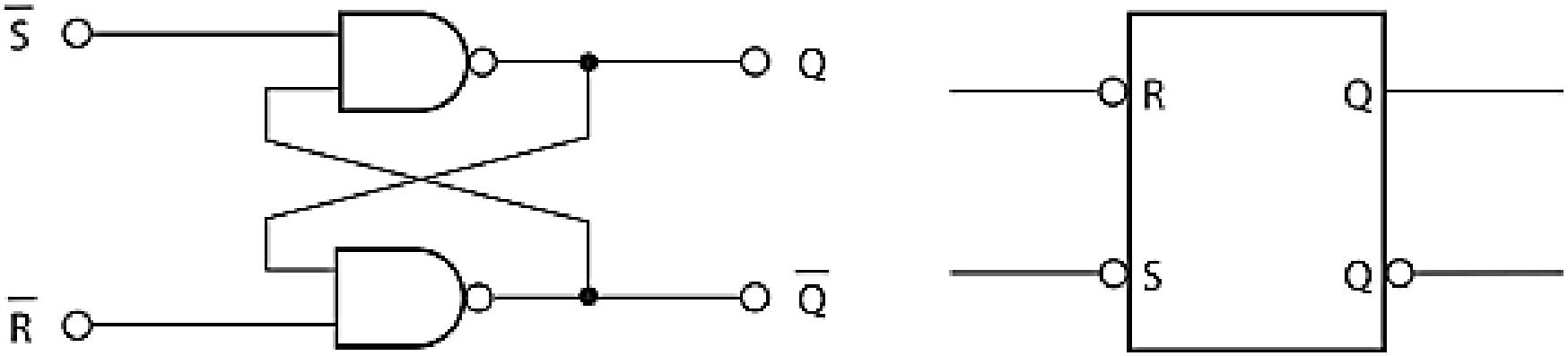
- 剛開始給正反器電路電源的時候，Q 跟 $\overline{Q}$ 值可能是 '0' 或 '1'。但不會兩個都是同樣的值



▲ 圖 6.7　非同步 RS 正反器的時序圖

6.1.5 用 NAND 電路構成非同步 RS 正反器

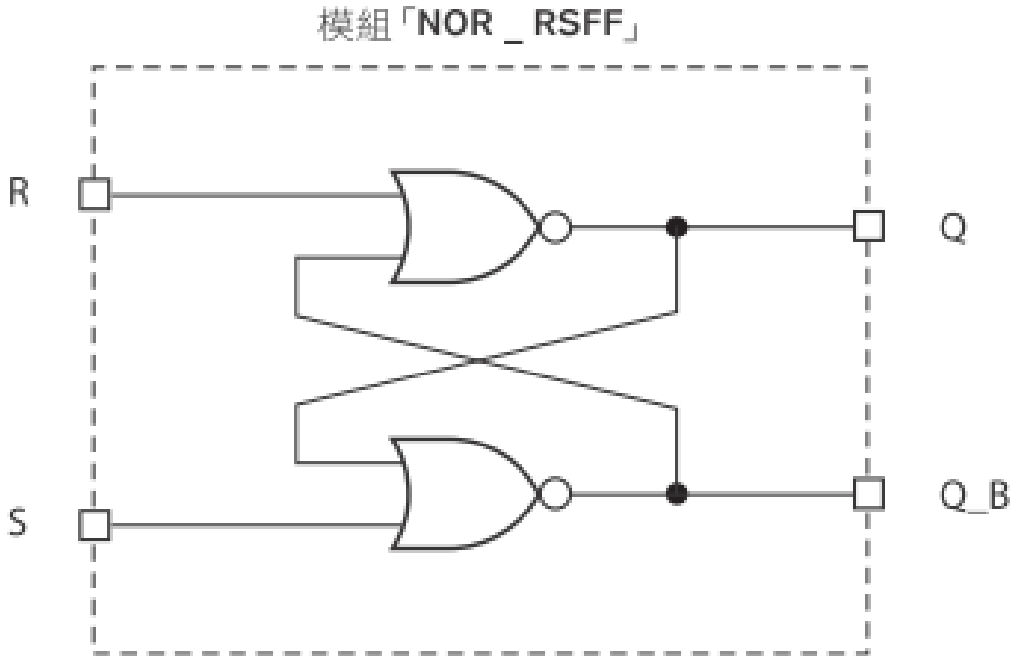
- 非同步 RS 正反器可以用 NAND 電路構成
- 輸出會連接到另一邊的輸入, 輸入有加上表示負邏輯的圈圈, Q, $\overline{Q}$ 跟 R, $\overline{S}$ 的位置關係都相反



▲圖 6.8 非同步 RS 正反器

6.1.6 Verilog HDL 描述

- (1) 邏輯閘層的模組化



▲ 圖 6.10　模組的定義

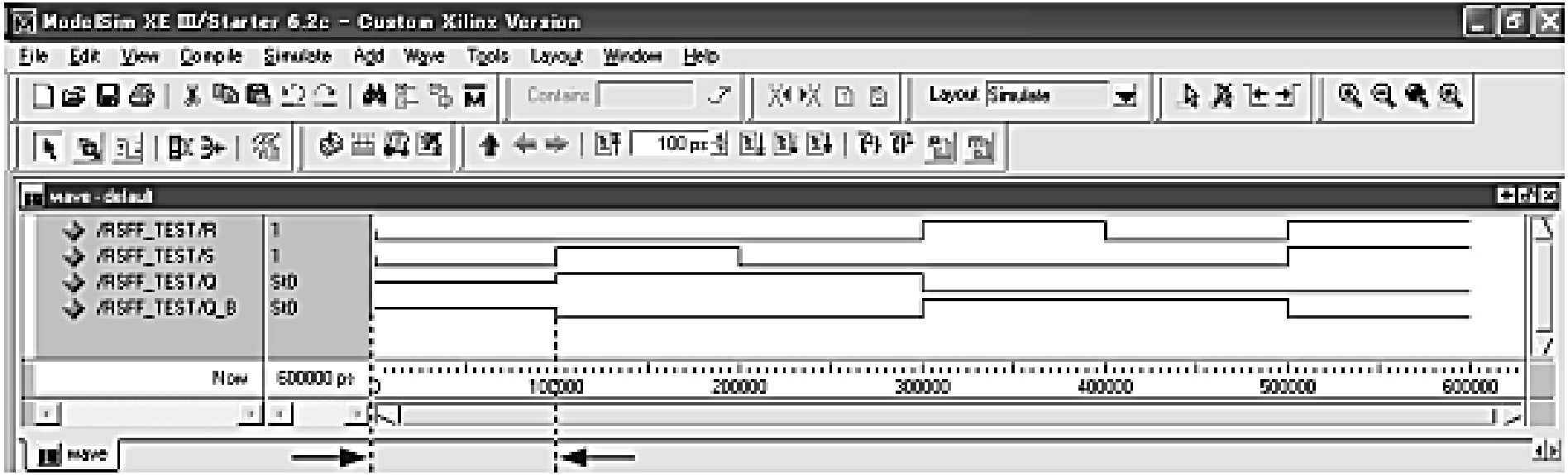
- 所有的「負邏輯的網絡名」為了方便會加上「_B」識別

程式 6.1　非同步 RS 正反器的測試平台

```
/*              RSFF_TEST       */
`timescale              1ns/1ns
module          RSFF_TEST;
reg             R, S;
wire            Q, Q_B;
```

程式 6.2　非同步 RS 正反器的 Verilog HDL 描述 (邏輯閘層)

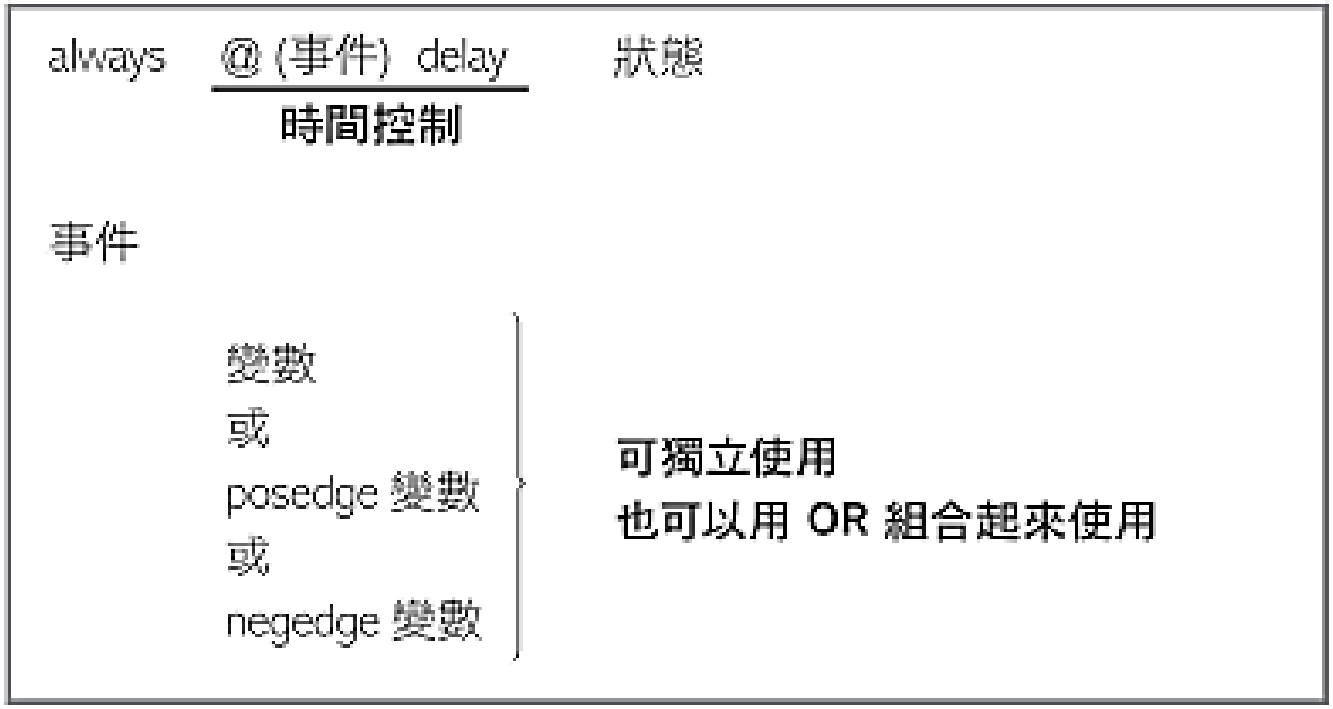
```
/*              NOR_RSFF        */
module          RSFF            ( R, S, Q, Q_B );
input           R, S;
output          Q, Q_B;
                nor             ( Q  , R, Q_B );
                nor             ( Q_B, S, Q   );
endmodule
```



這段時間內, 因為 Q 與 Q_B 的值不確定, 所以會介於 H level 與 L level 的中間

▲ 圖 6.11　非同步 RS 正反器的模擬結果

- (2) 行為層的模組化
  - always 語法是滿足「事件」，且在「delay」定義的時間結束之前中斷執行中的「狀態」

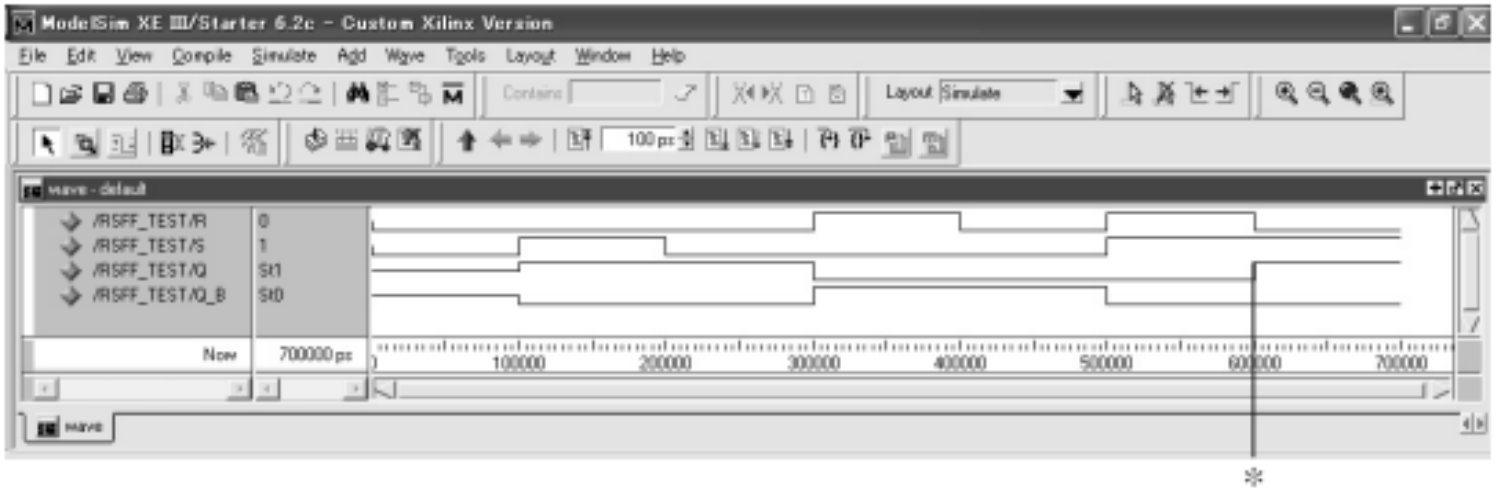


always 的語法定義

程式 6.3　非同步 RS 正反器的 Verilog HDL 描述 (行為層)　LST6_3.V

```
/*          RSFF          */
module      RSFF          ( R, S, Q, Q_B );
input       R, S;
output      Q, Q_B;
reg  *1     Q, Q_B;
            always  *2    @( R or S )
                          case ({ R , S }) *3
                               1:begin Q <= 1; Q_B <= 0; end *4
                               2:begin Q <= 0; Q_B <= 1; end
                               3:begin Q <= 0; Q_B <= 0; end
                          endcase
endmodule
```

- 1. 用 reg 宣告為正反器的輸出
- 2. always 與事件的部分,根據 R 或 S 的變化來執行
- 3. case 描述, 用連接運算子「{}」將 R 與 S 連結在一起,就變成 case item ‘0’ ~ ‘3’
- 4. case item ‘0’ 時, Q 與 Q_B 沒有變化



▲ 圖 6.12　非同步 RS 正反器的模擬結果（事件為「R or S」）

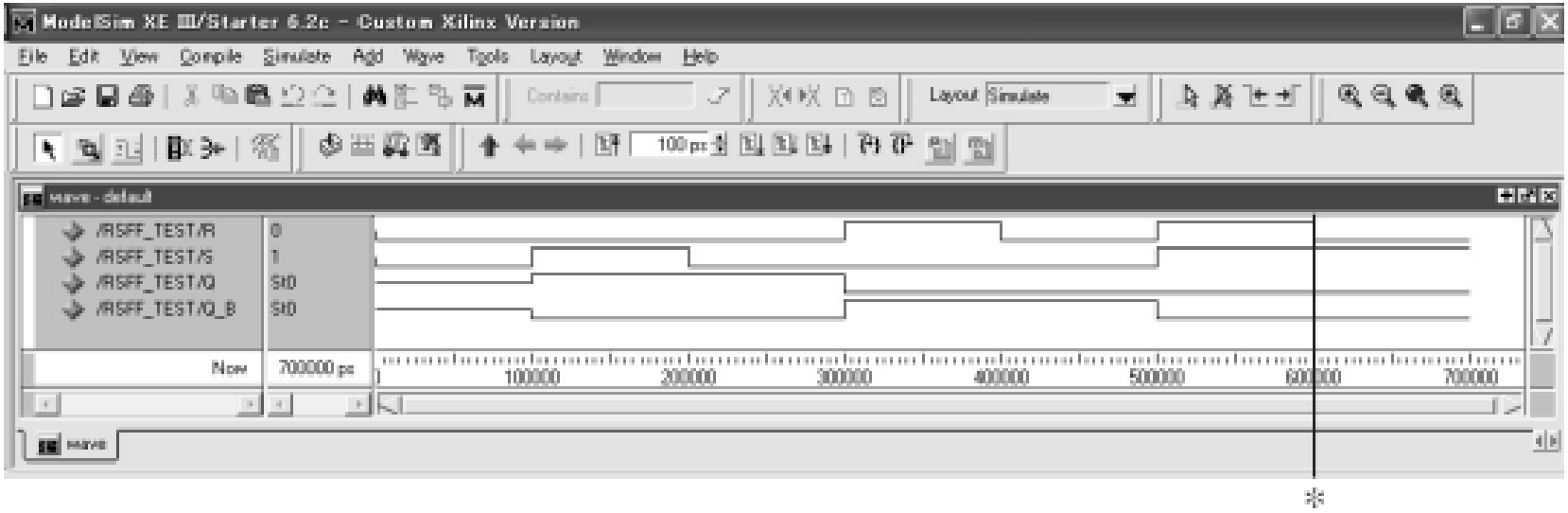
- R='1', S='1' 這樣的禁止輸入, 之後就變成 R='0'
- 之後當R降為'0'十

- 若將 always @(R or S)改為

  always @(posedge R or posedge S)

  會在*處無法將'0'帶入而讓Q升為'1'

  (因是negedge)

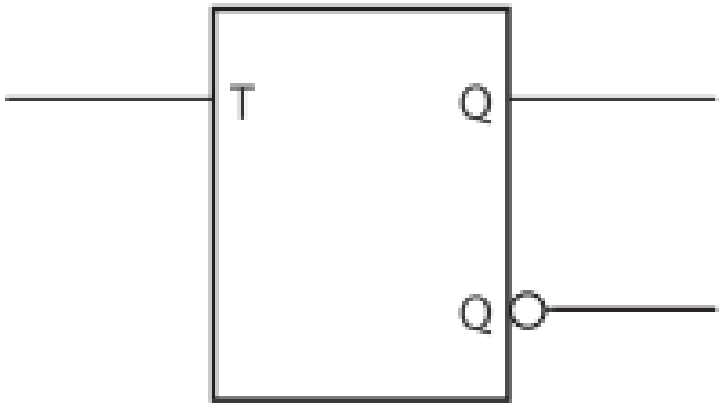


▲ 圖 6.13　非同步 RS 正反器的模擬結果（事件為「**posedge R or posedge S**」）

6.2 非同步 T 型正反器

6.2.1 動作與電路符號

- 非同步正反器的電路符號
  - 輸入:同正反器名稱的 T
  - 輸出:Q,$\overline{Q}$



▲圖 6.14　非同步 T 型正反器

6.2.2 特性表, 特性方程式, 狀態圖

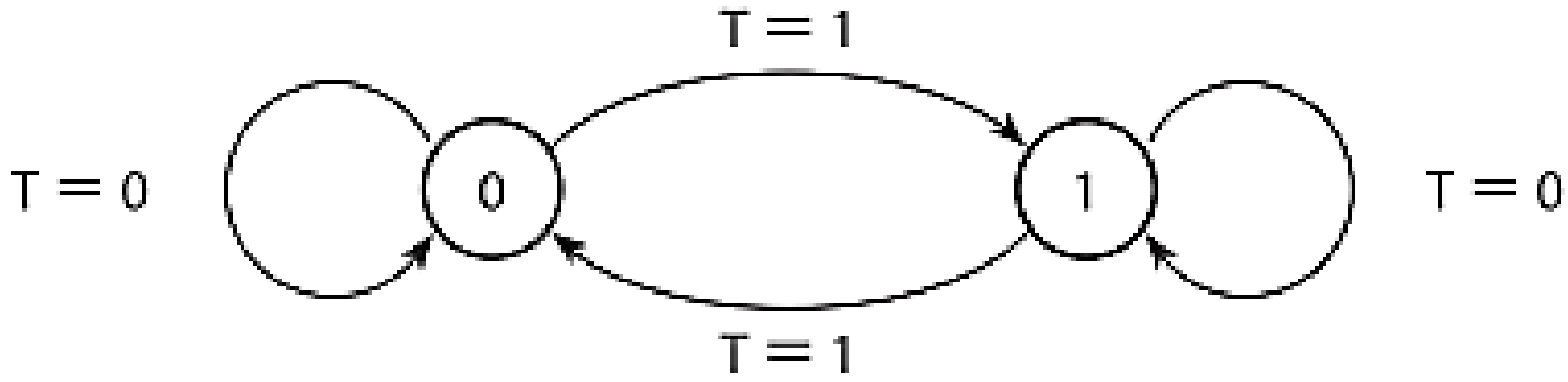
- 1. 特性表

T	Q	Q'	
0	0	0	沒有變化
0	1	1	
1	0	1	反轉
1	1	0	

- 2. 特性方程式

$$Q' = T \cdot \overline{Q} + \overline{T} \cdot Q$$
$$= T \oplus Q$$

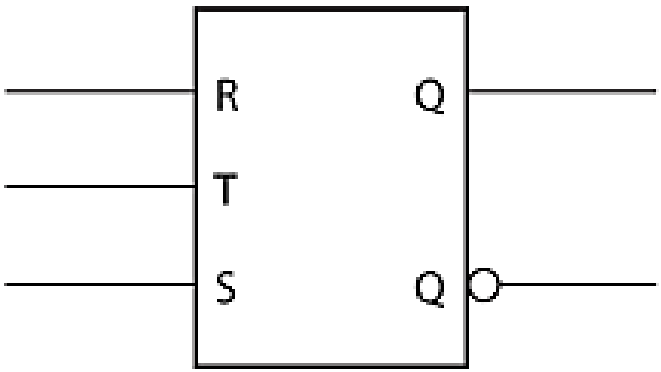
- 3.狀態圖
  - T='0' 的話 Q 沒有變化, '1' 的話就反轉



▲ 圖 6.16　非同步 T 型正反器的特性表, 特性方程式, 狀態圖

6.2.3 初始狀態

- T 型正反器的初始狀態跟其他的正反器一樣，Q 跟 $\overline{Q}$ 都是未知的
- T 型正反器通常會增加 R 或 S，叫做 RST 正反器
- RST 正反器動作是當 R 為 ON 的時候重置，S 為 ON 的時候設置，T 為 ON 的時候切換。



▲ 圖 6.17 非同步 RST 正反器

6.2.4 Verilog HDL 描述

程式 6.7　非同步 T 型正反器的 Verilog HDL 描述　LST6_7.V

```
/*              TFF          */
module          TFF          ( R, T, Q, Q_B );
input           R, T;
output          Q, Q_B;
reg             Q; *1

          *2    assign       Q_B = ~Q;
          *3    always       @( posedge R or posedge T )
                             Q <= ( R )? 0: ~Q;
endmodule
```

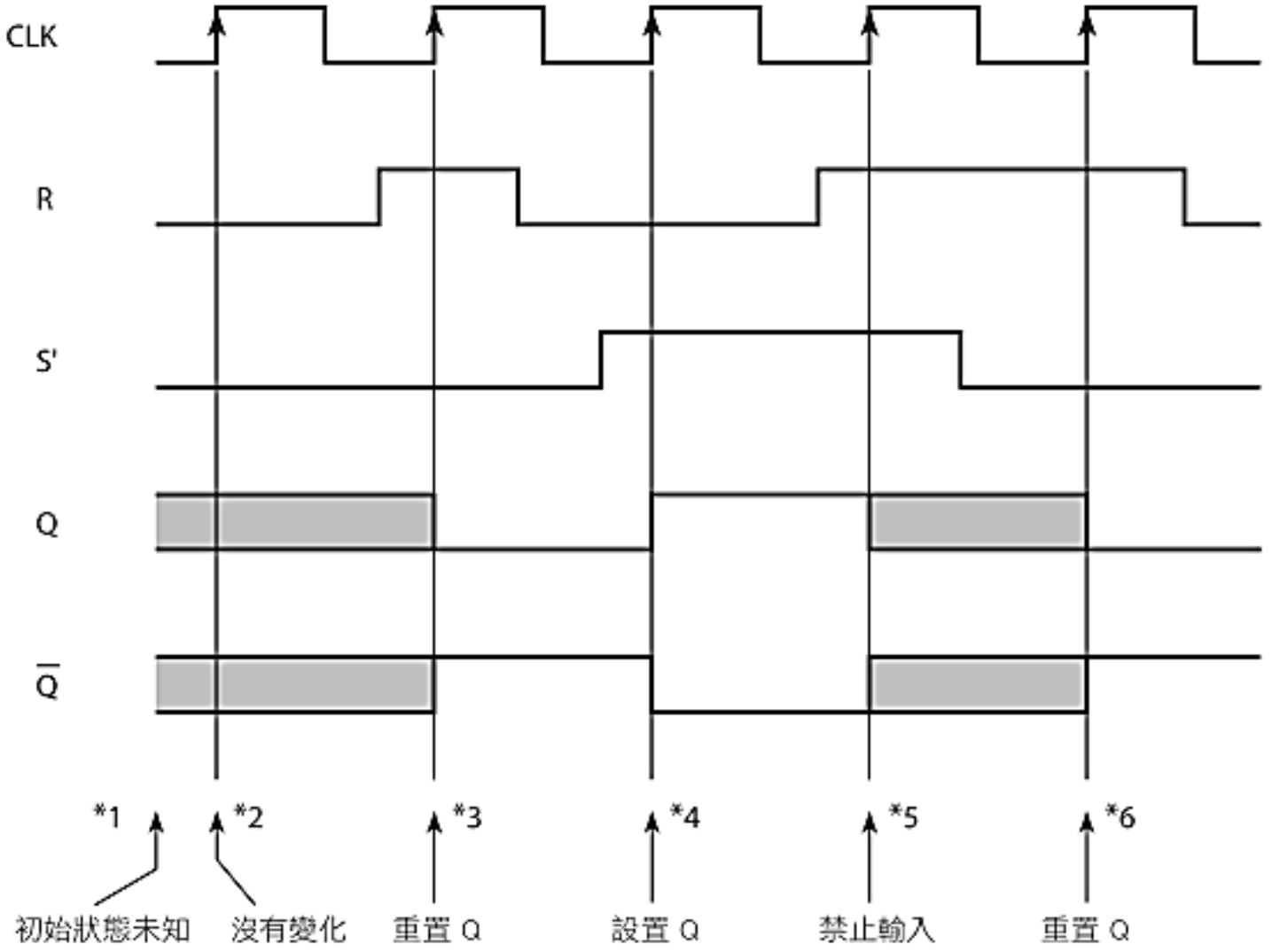
- 1.用 reg 單獨只宣告 Q
- 2.用 assign 將 Q_B 的值代入
- 3. always 的事件寫成「 posedge R or posedge T 」，當 R 或 T 正緣的時候就會執行狀態

6.3 同步 RS 正反器

- 同步正反器附有時脈輸入, 根據時脈的正緣或者負緣與 R 或 S 的值來決定輸出的結果

6.3.1 動作與電路符號

- 同步 RS 正反器動作根據 CLK 正緣時的 R 與 S 值來決定正反器的重置或者設置
- 正反器的「非同步」與「同步」的差異就在於有沒有 CLK
- 同步型正反器會視時脈的正緣或負緣決定 Q 與 $\overline{Q}$ 的值



▲ 圖 6.20　同步 RS 正反器的時序圖

6.3.2 特性表, 特性方程式, 狀態圖

- 同步 RS 正反器的特性表, 特性方程式, 狀態圖與非同步 RS 正反器相同
- 差在 CLK 的正緣與負緣的動作

6.3.3 Verilog HDL 描述

程式 6.8　同步 RS 正反器的測試平台　LST6_8.V

```
/*              SY_RSFF_TEST */
`timescale      1ns/1ns
module          SY_RSFF_TEST;
reg             R, S, CLK;
wire            Q, Q_B;
parameter       STEP = 50;
SY_RSFF         SY_RSFF         ( R, S, CLK, Q, Q_B );
            *1  always          #( STEP/2 )     CLK = ~CLK;
```

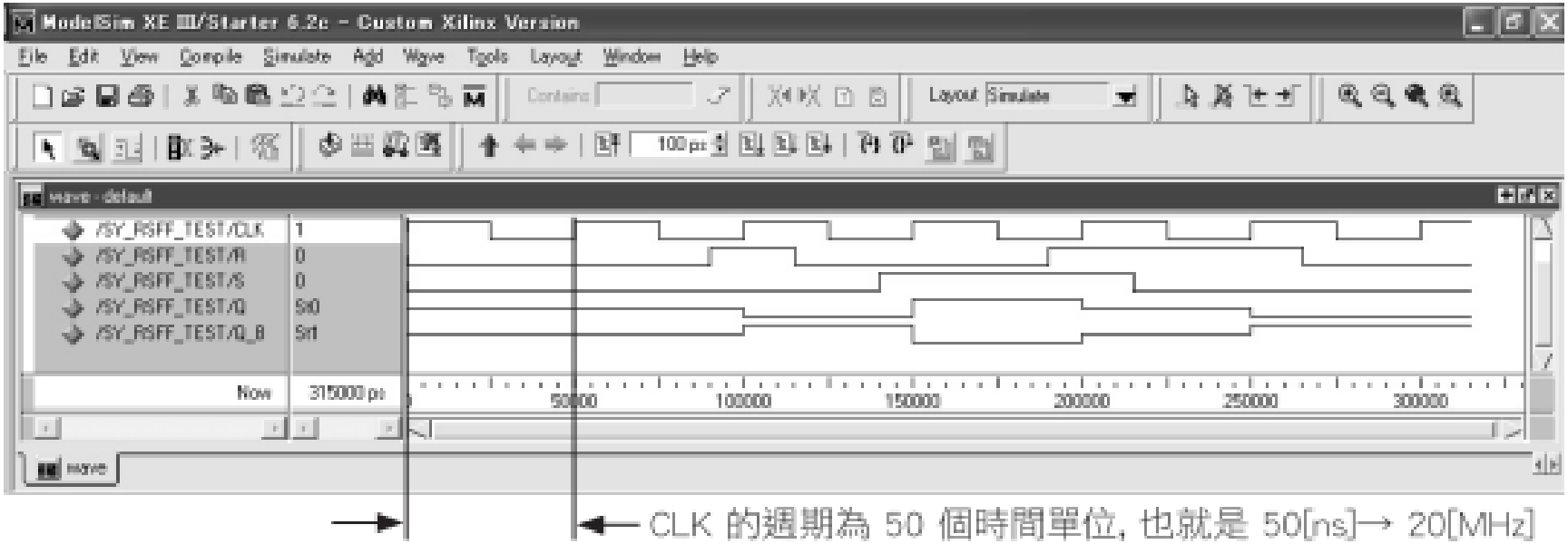
接下頁

- 1.測試平台對 CLK 的描述。每「STEP/2」為單位

程式 6.9　同步 RS 正反器的 Verilog HDL 描述　LST6_9.V

```
/*              SY_RSFF         */
module          SY_RSFF         ( R, S, CLK, Q, Q_B );
input           R, S, CLK;
output          Q, Q_B;
reg             Q;
                assign          Q_B = ~Q;
            *2  always          @( posedge CLK )
                                case ({ R ,S })
                                        1:Q <= 1;
                                        2:Q <= 0;
                                    *3  3:Q <= 1'bx;
                                endcase
endmodule
```

- 2. always 宣告的事件,狀態跟非同步 RS 正反器一樣, 但是禁止輸入的值為 'x' (未知)。
- 3. 將 'x' 代入到 reg 變數「Q <= 1'bx;」

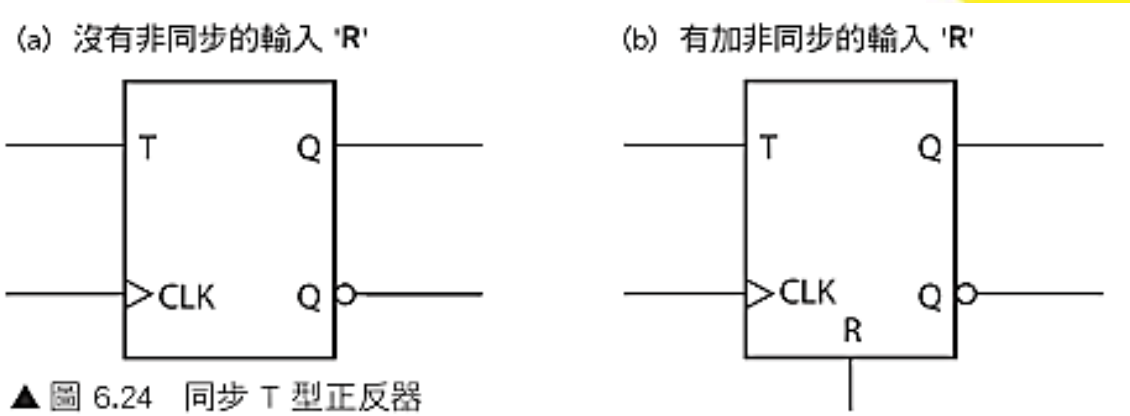


▲ 圖 6.22　同步 RS 正反器的模擬結果 (正緣觸發)

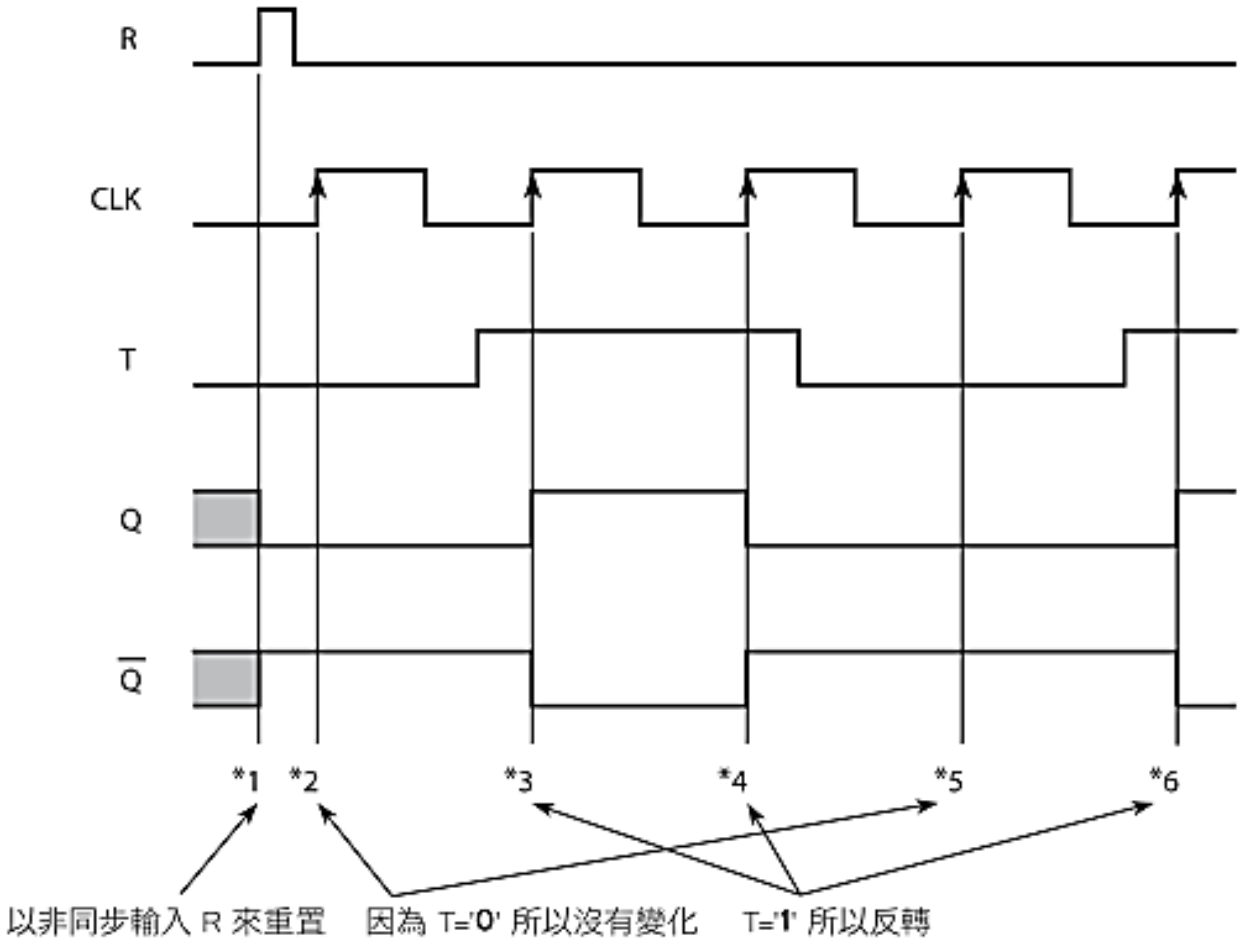
6.4 同步 T 型正反器

6.4.1 動作與電路符號

- 這個正反器的動作是當 CLK 正緣的時候，輸入 T 若為 '1' 則 Q 與 Q 反轉，'0' 的話則沒有變化。
- T 型正反器沒辦法決定初始值的，所以加入非同步的輸入 R

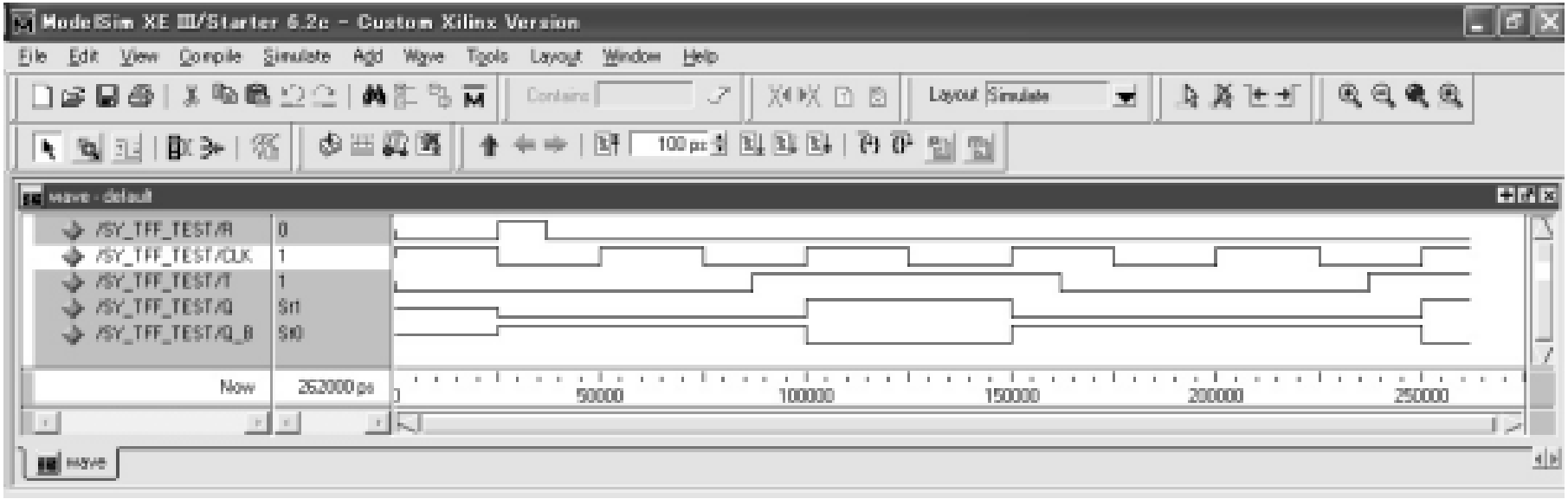


▲ 圖 6.24　同步 T 型正反器



6.4.3 Verilog HDL 描述

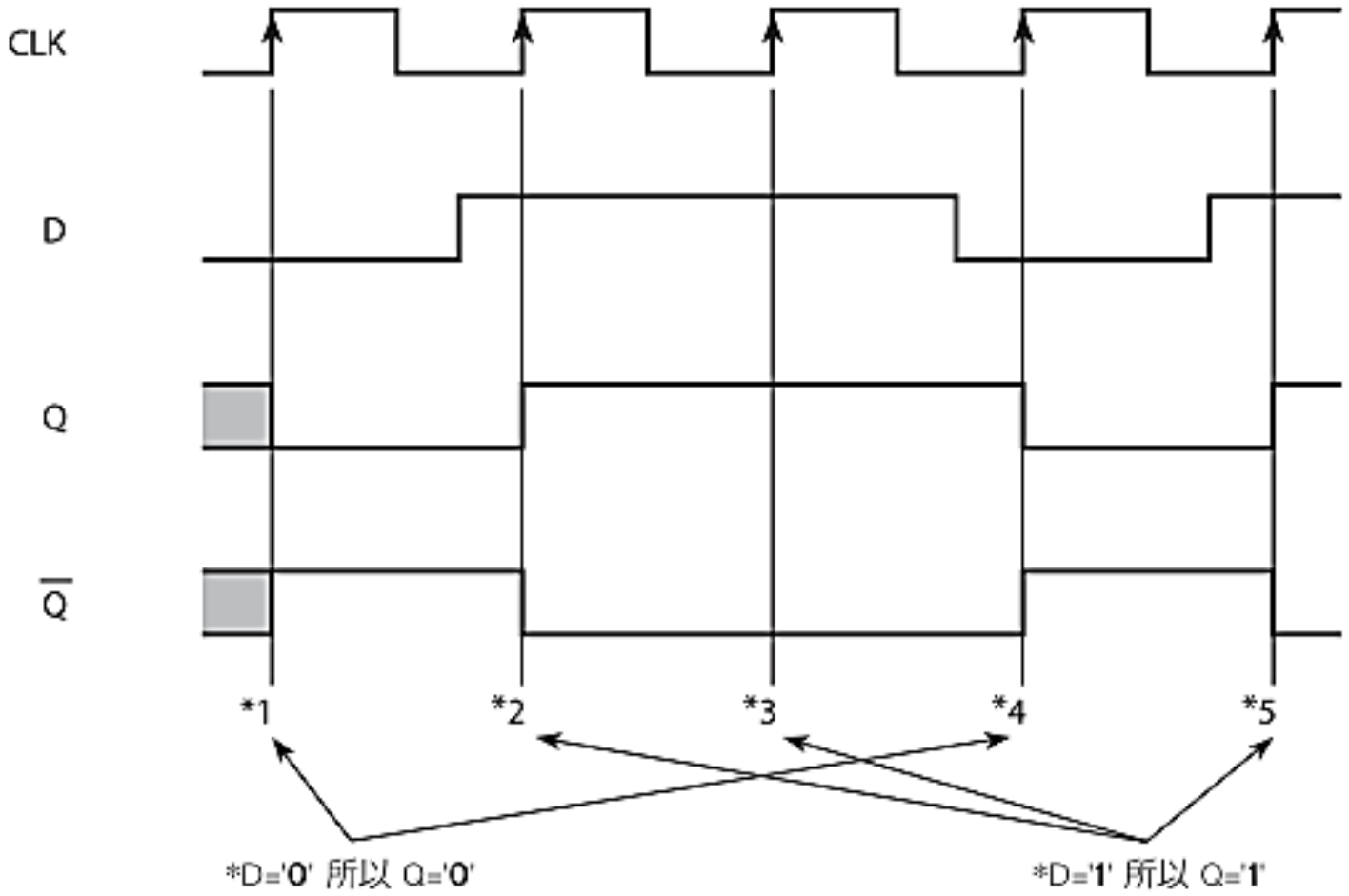
- 執行狀態的依據是 CLK 跟 R 的正緣, 所以寫作「posedge CLK or posedge R」；不要寫成「posedge CLK or R」



▲ 圖 6.26 同步 T 型正反器的模擬結果

6.5 同步 D 型正反器

- 當 CLK 正緣時, 把輸入 D 的值給 Q, 反向值給 $\overline{Q}$

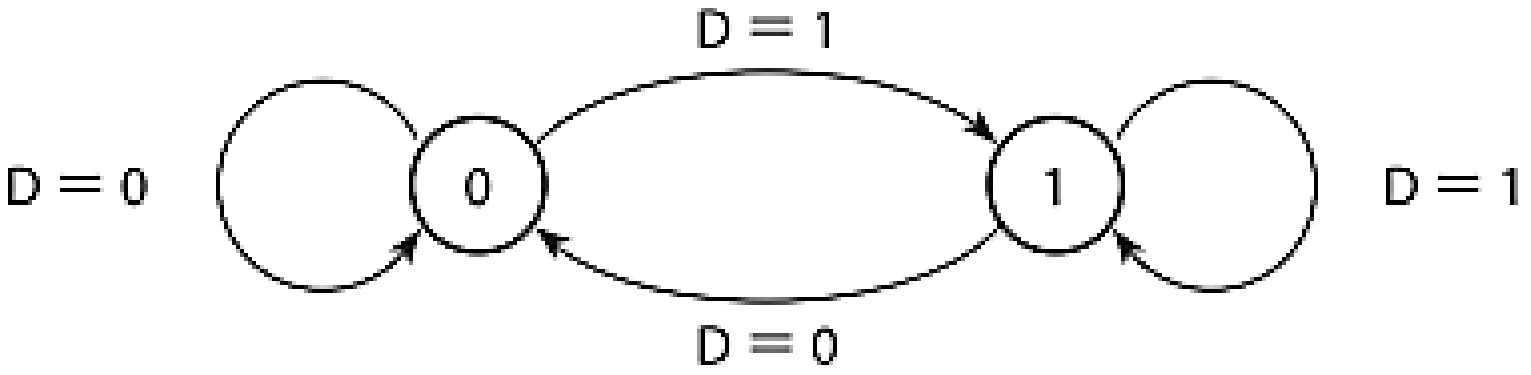


6.5.2 特性表, 特性方程式, 狀態圖

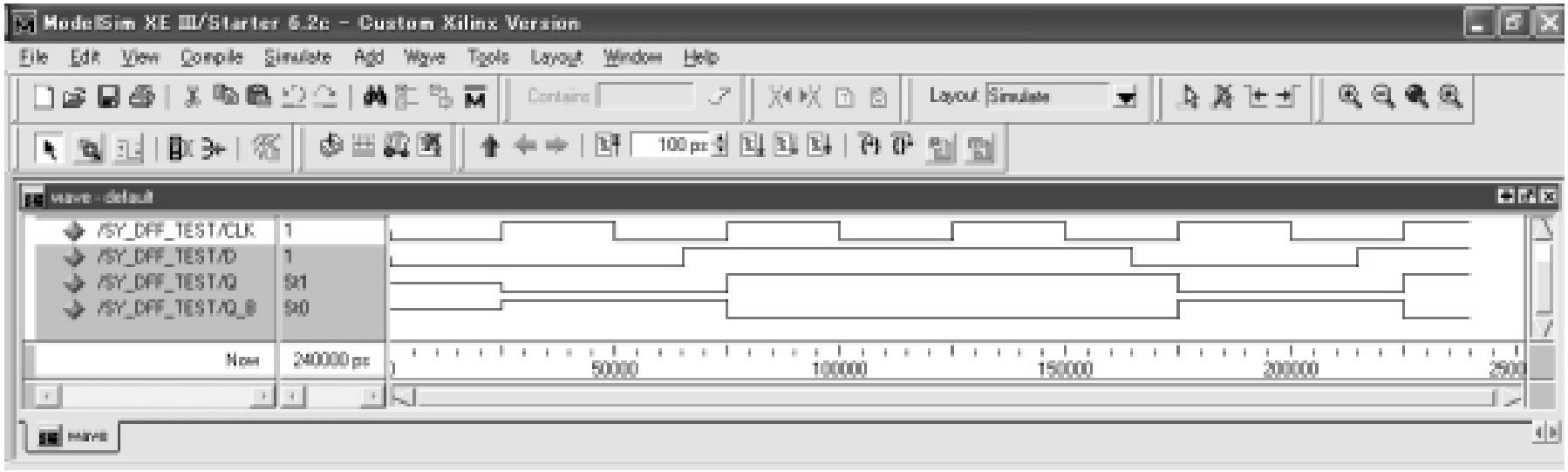
- 特性表

D	Q	Q'
0	0	0
0	1	0
1	0	1
1	1	1

- 特性方程式 $Q' = D$
- 狀態圖



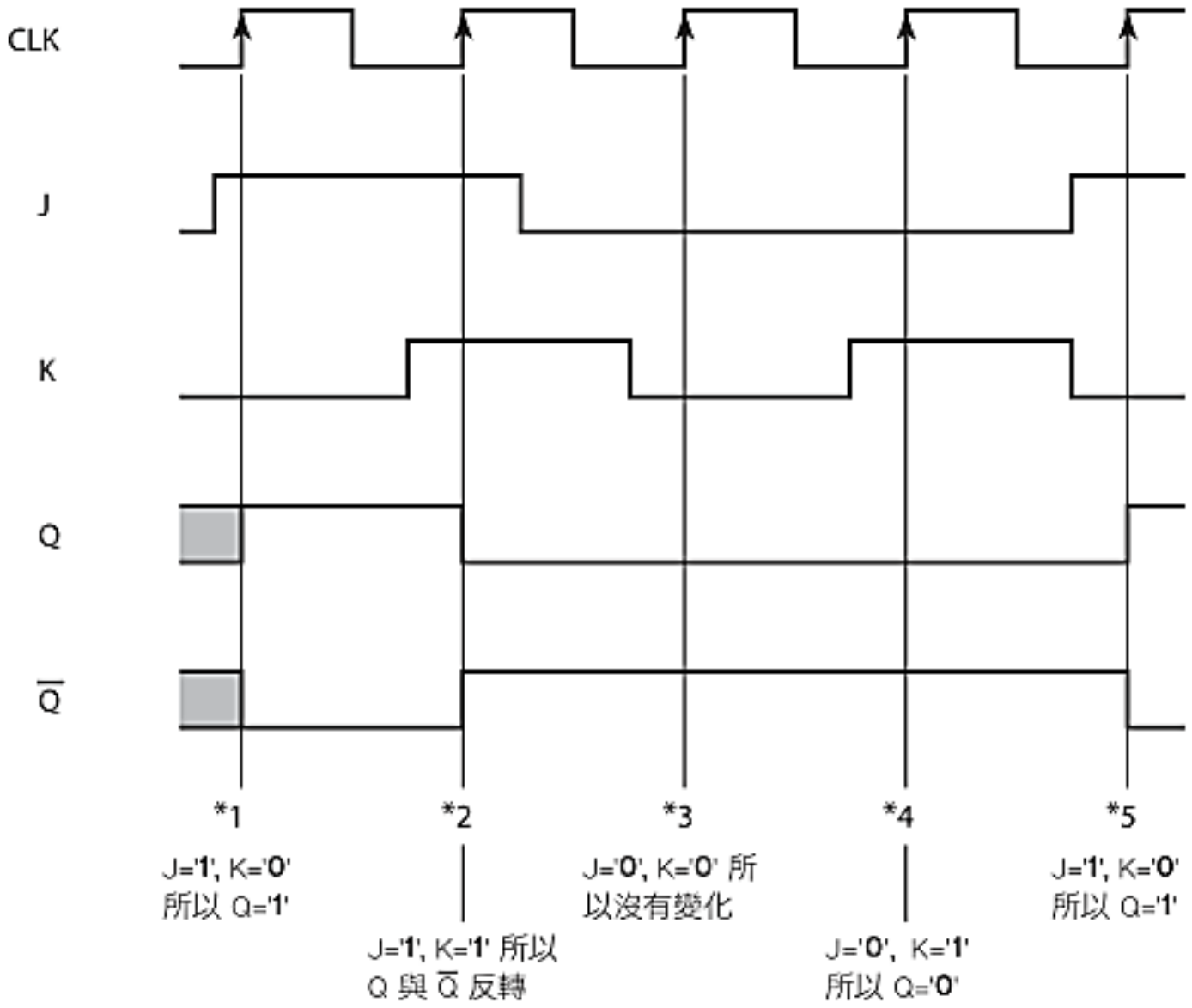
同步 D 型正反器的模擬結果



▲ 圖 6.30　同步 D 型正反器的模擬結果

6.6 同步 JK 正反器

- 當 CLK 正緣的時候會依照 J、K 的值決定輸出
  - J=' 0', K='0' 的話 Q 與 $\overline{Q}$ 沒有變化
  - J=' 1', K='1' 的話 Q 與 $\overline{Q}$ 反轉
  - J=' 1', K='0' 的話 Q='1', $\overline{Q}$ ='0'
  - J=' 0', K='1' 的話 Q='0', $\overline{Q}$ ='1'


CLK
J
K
Q
$\overline{Q}$
*1
*2
*3
*4
*5
J='1', K='0' 所以 Q='1'
J='1', K='1' 所以 Q 與 $\overline{Q}$ 反轉
J='0', K='0' 所以沒有變化
J='0', K='1' 所以 Q='0'
J='1', K='0' 所以 Q='1'

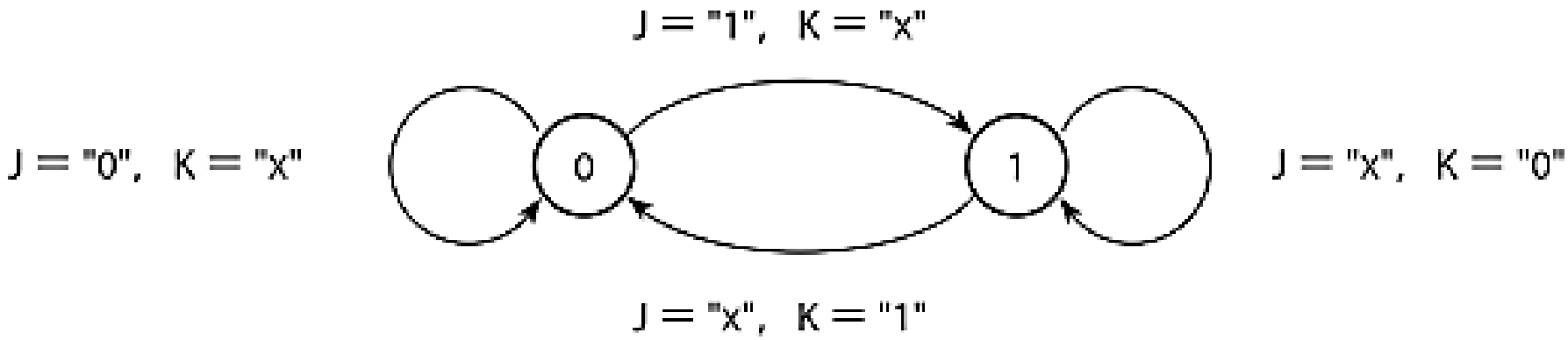
6.6.2 特性表, 特性方程式, 狀態圖

- 特性表

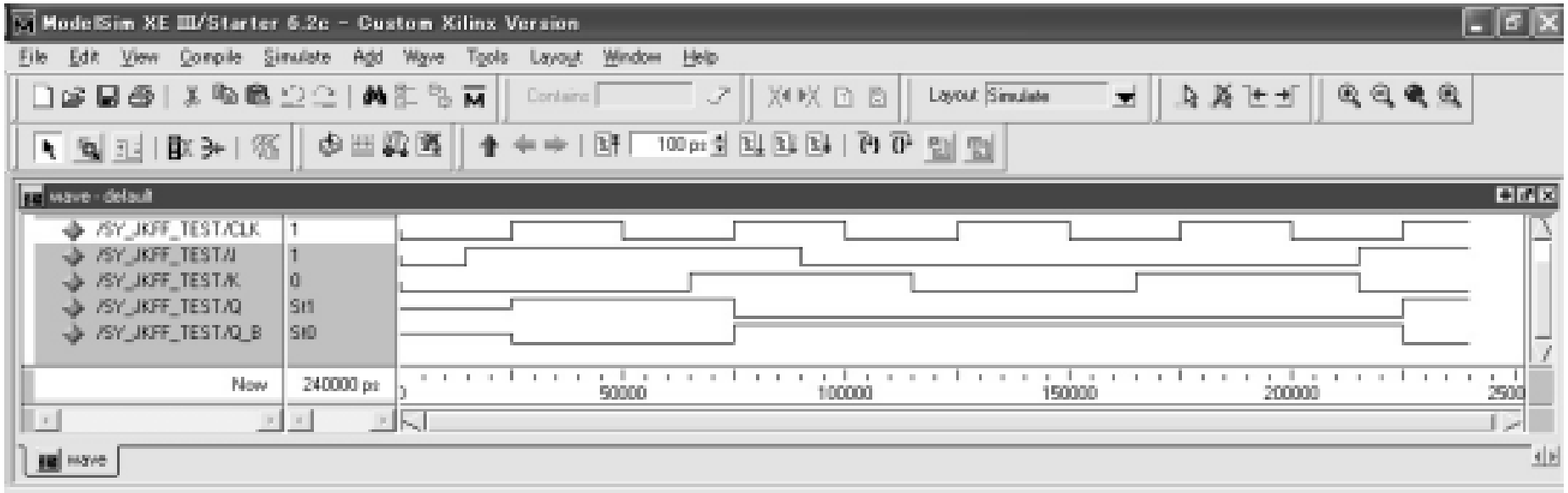
J	K	Q	Q'
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

- 特性方程式 $Q' = J \cdot \overline{Q} + \overline{K} \cdot Q$

- 狀態圖 (用 x: don't care 表示)



同步 JK 正反器的模擬結果



▲ 圖 6.34　同步 JK 正反器的模擬結果

6.7 特性方程式的應用

RS 正反器　　$: Q' = S + \overline{R} \cdot Q$

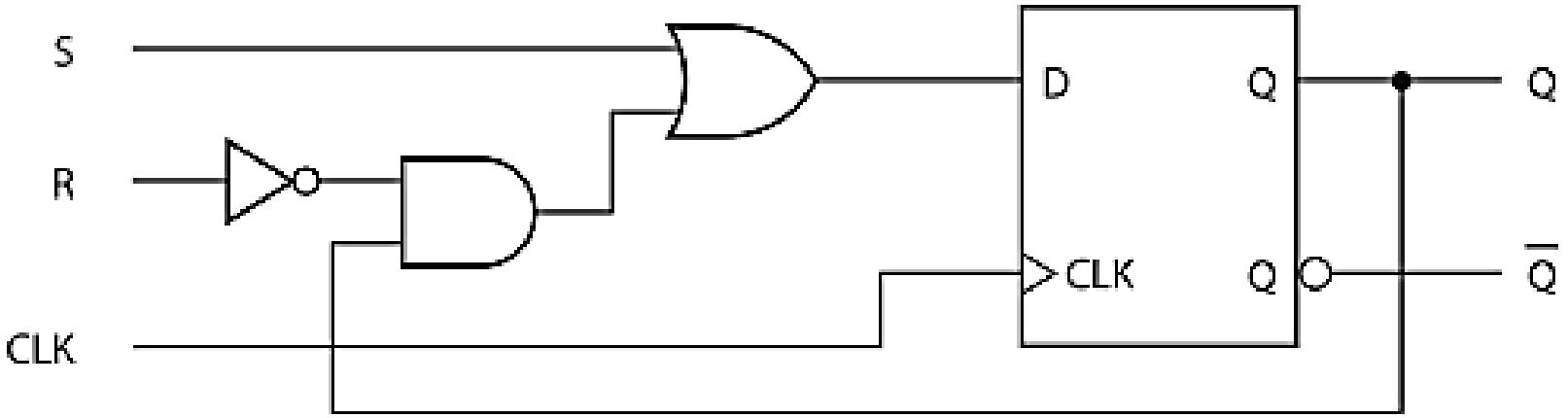
T 型正反器　　$: Q' = T \oplus Q$

JK 正反器　　$: Q' = J \cdot \overline{Q} + \overline{K} \cdot Q$

D 型正反器　　$: Q' = D$

6.7.1 同步 RS 正反器

- RS 正反器 ：$Q' = S + \overline{R} \cdot Q$
- 根據 RS 正反器的特性方程式來連接組合電路，

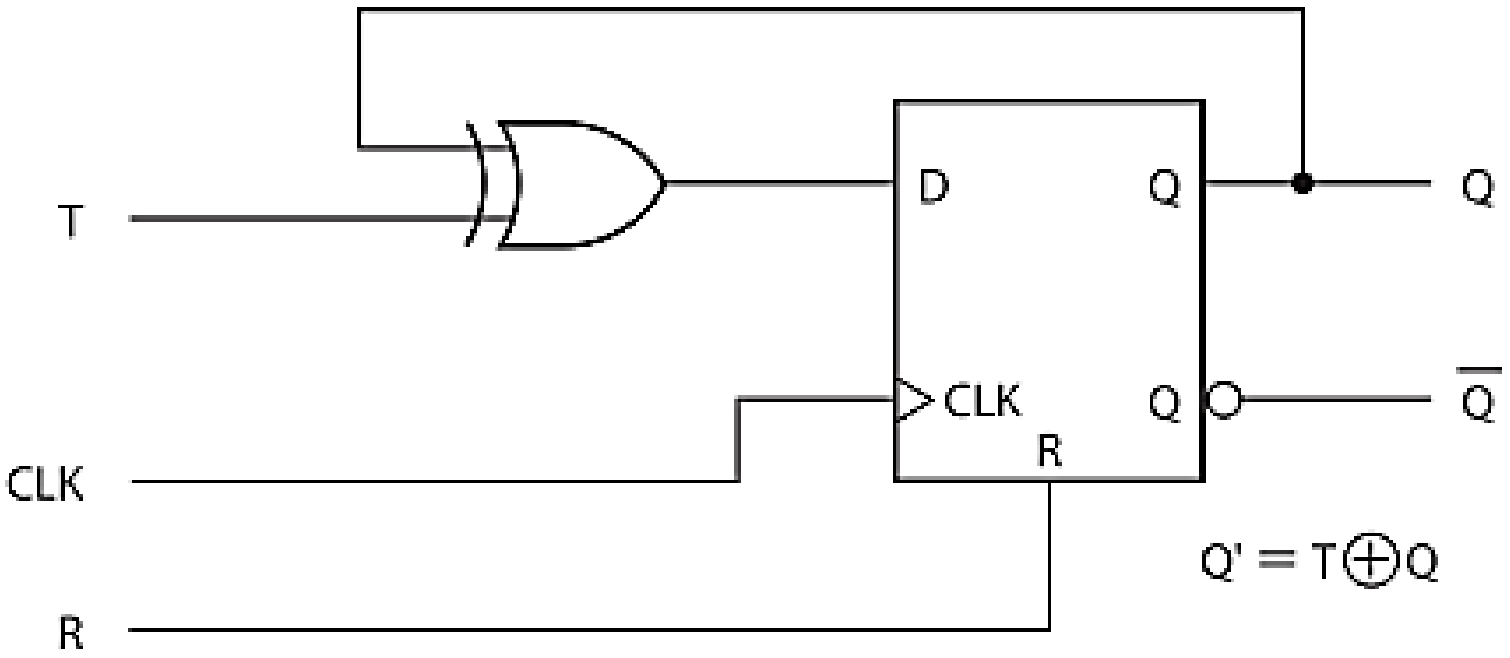


$$Q' = S + \overline{R} \cdot Q$$

▲圖 6.35 用 D 型正反器實現 RS 正反器

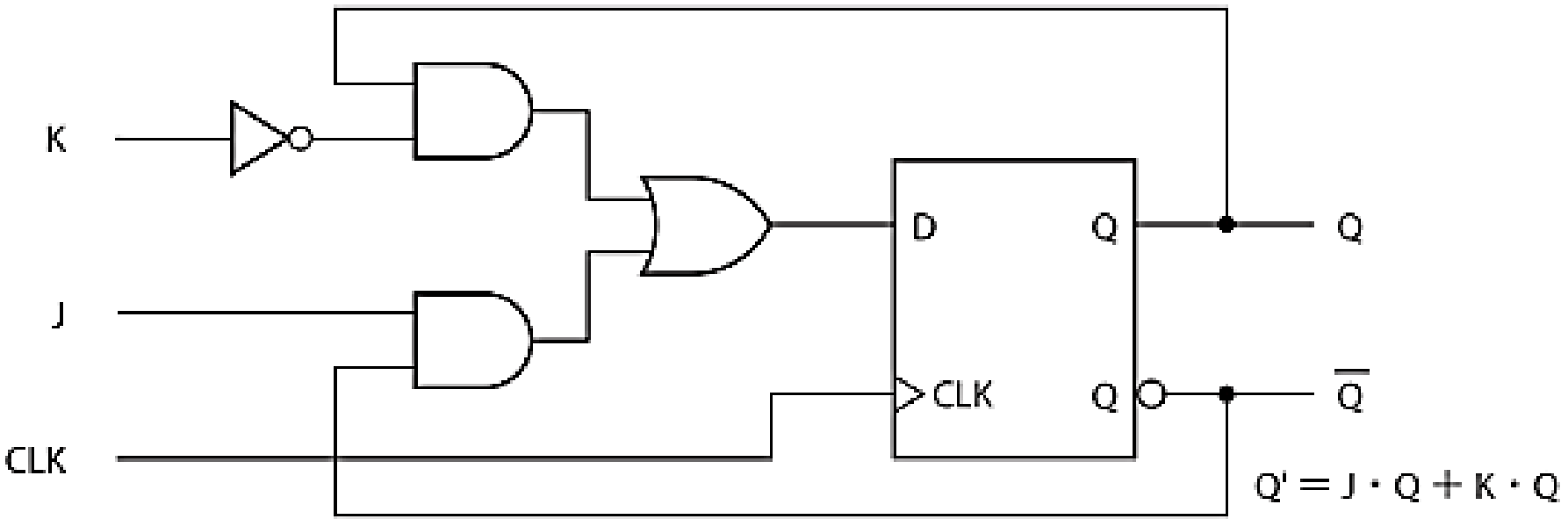
6.7.2 同步 T 型正反器

- T 型正反器 ：$Q' = T \oplus Q$
- 根據特性方程式來連接組合電路, 最後再接上 D 型正反器的輸入



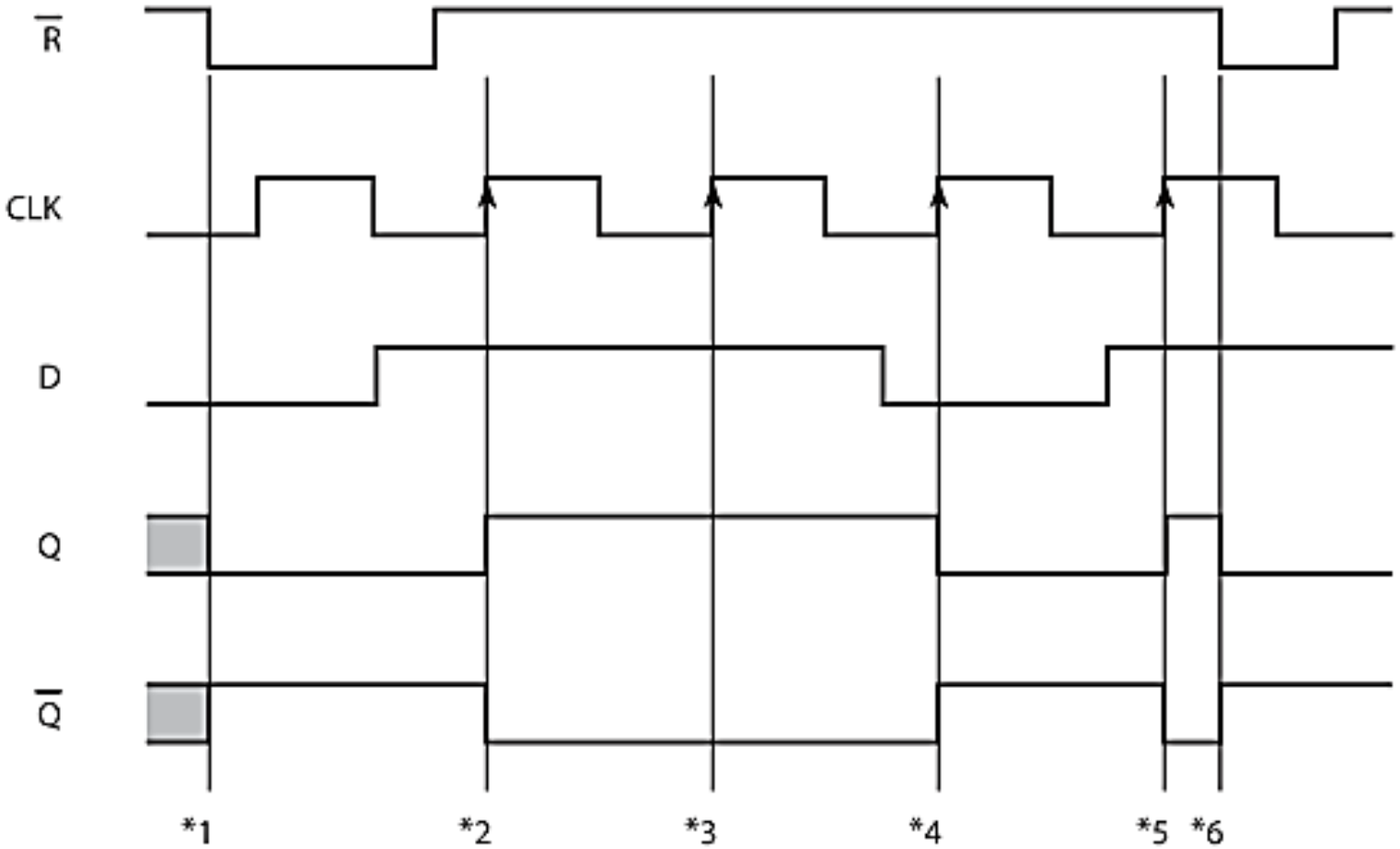
6.7.3 同步 JK 正反器

- JK 正反器 ：$Q' = J \cdot \overline{Q} + \overline{K} \cdot Q$
- 同樣根據特性方程式來連接組合電路, 最後再接上 D 型正反器的輸入



- 利用特性方程式做的 JK 正反器的模擬, 在重置 Q='0' 之前無法保證其動作

6.8 非同步 R + 同步 D 型正反器



▲圖 6.39 (b) 非同步 R + 同步 D 型正反器

- $\overline{R}$ 是負邏輯所以為 '0' 的時候會清除正反器
- $\overline{R}$ 為 '1' 的話, CLK 正緣的時候把 D 的值給 Q, 把反轉的值給 Q

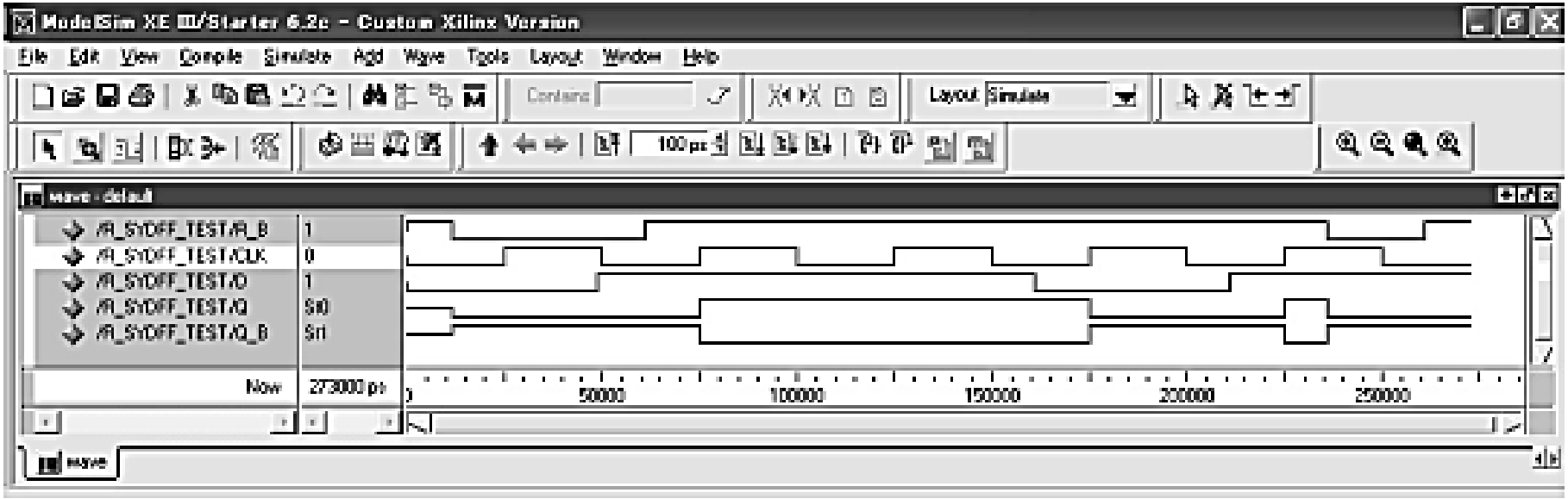
- 特性表

R	CLK	D	Q	Q'
0	—	—	—	0
1	1	—	1	1
1	1	—	0	0
1	0	—	1	1
1	0	—	0	0
1	↑	1	—	1
1	↑	0	—	0

▲ 表 6.2　非同步 R + 同步 D 型正反器的特性表

6.8.3 Verilog HDL 描述

- 注意 Verilog HDL 中的條件為「posedge CLK or negedge R_B」



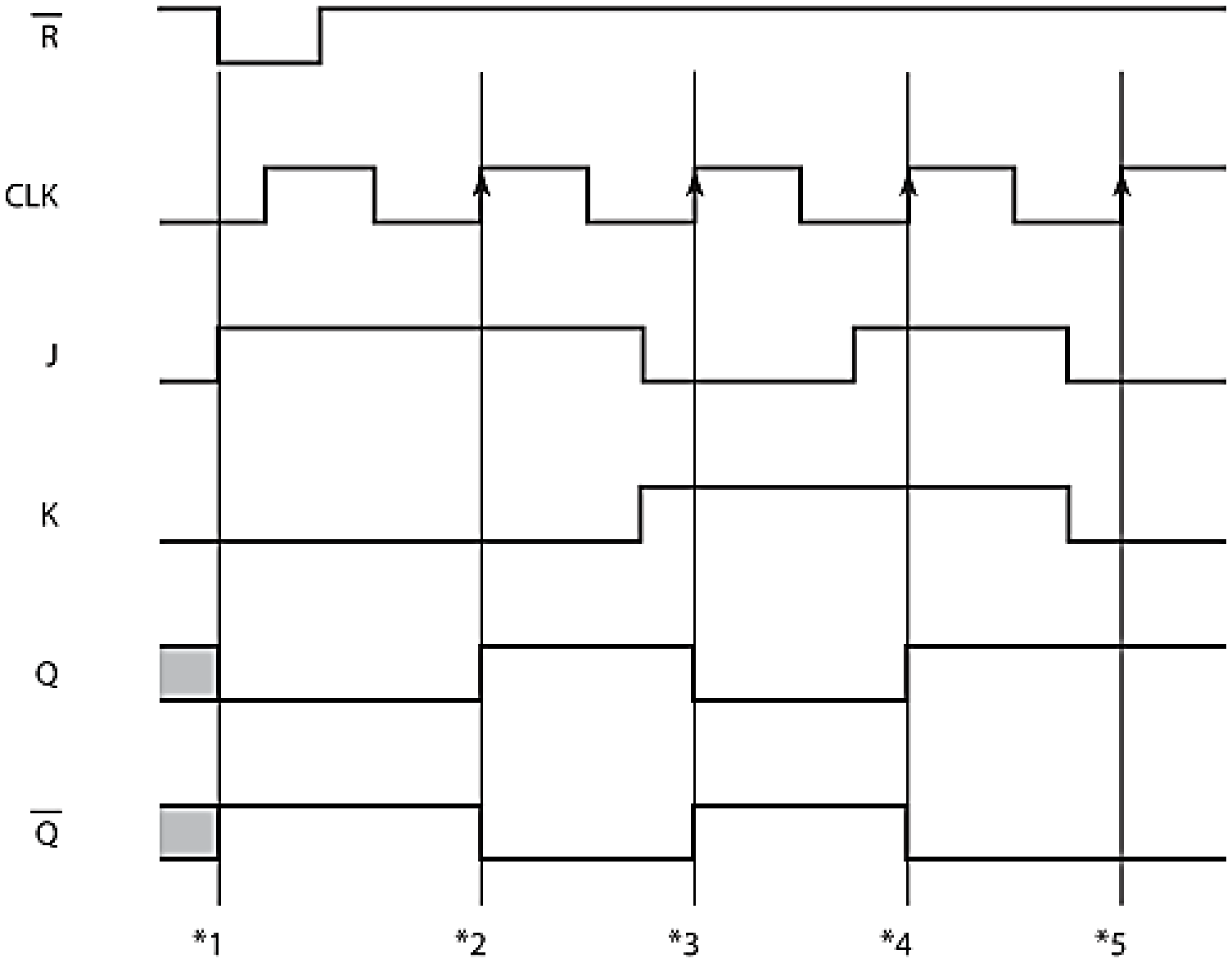
▲ 圖 6.40 非同步 R+ 同步 D 型正反器的模擬結果

6.9 非同步 R + 同步 JK 正反器

- $\overline{R}$ 為負邏輯輸入，當值為 '0' 的時候會清除正反器。為 '1' 的話，當 CLK 為正緣時, Q 與 Q 會根據 J 與 K 的值輸出
- 特性表

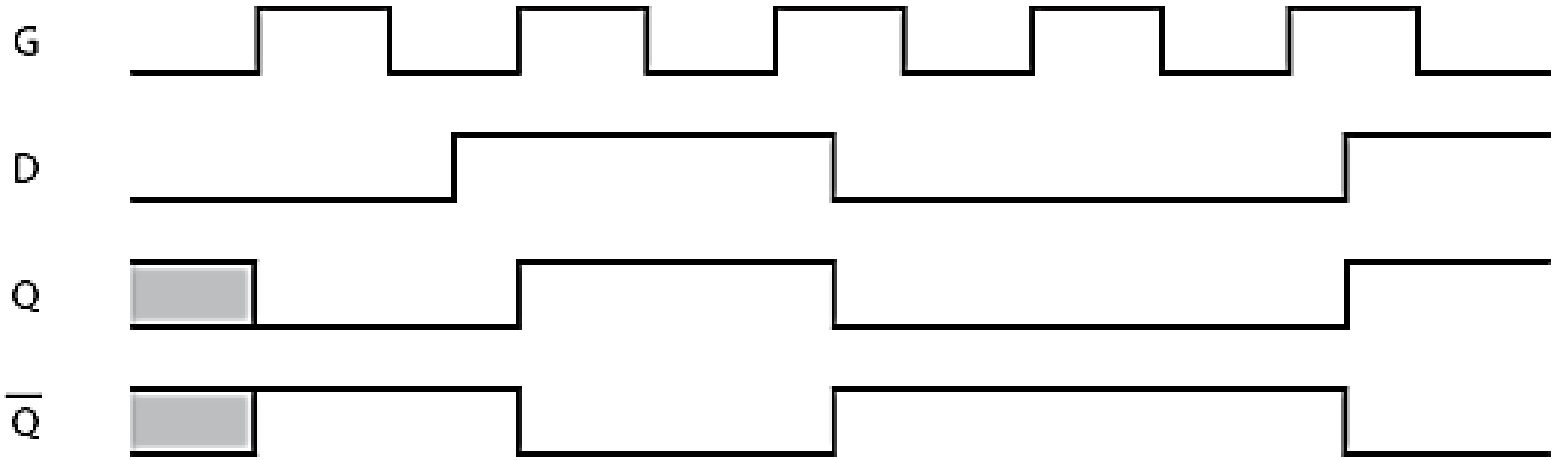
$\overline{R}$	CLK	J	K	Q	Q'
0	—	—	—	—	0
1	1	—	—	0	0
1	1	—	—	1	1
1	0	—	—	0	0
1	0	—	—	1	1
1	↑	0	0	0	0
1	↑	0	0	1	1
1	↑	0	1	—	0
1	↑	1	0	—	1
1	↑	1	1	0	1
1	↑	1	1	1	0

- 時序圖



6.10 閂鎖電路

- G 為 '1' 的時候 D 的值給 Q, 反轉的值給 $\overline{Q}$。在這一段期間中如果 D 一有變化就會馬上反映到 Q 與 $\overline{Q}$
- 上述動作與正反器不同。有這種動作的閂鎖電路稱為透明閂鎖 (transparent latch)

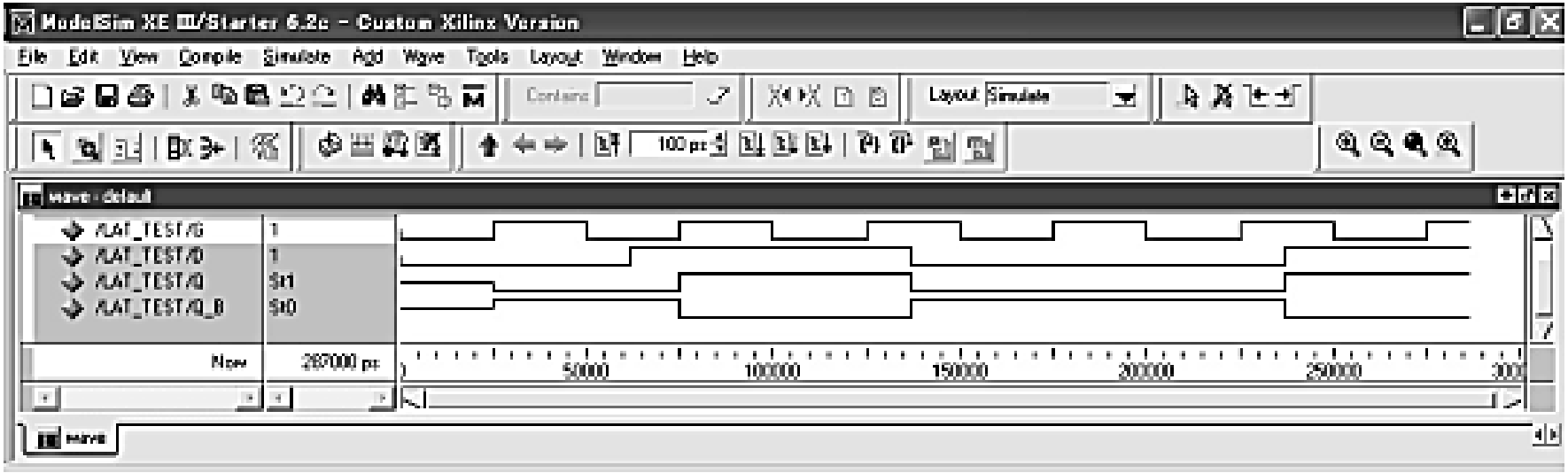


G='1' 的時候 Q=D, $\overline{Q}=\overline{D}$

G 為負緣的時候輸出保持為 D 的值

6.10.2 Verilog HDL 描述

- 狀態執行的條件是當「G 正緣或負緣」與「G='1' 且 D 有變化的時候」，因此事件寫成「D or G」



▲ 圖 6.44　閂鎖電路的模擬結果

6.11 assign 與 always 的執行時間

- assign右邊的變數只要有變化, 就會經過運算之後把最新的結果代入到左側；有指定 delay, 會在 delay 的時間內中斷正在執行的程序

  assign #10 OUT = A & B;

  上述 A 或 B 只要有一個信號有變化, 就會中斷 10 個時間單位之後再把「A and B」算出來的結果代入到 OUT

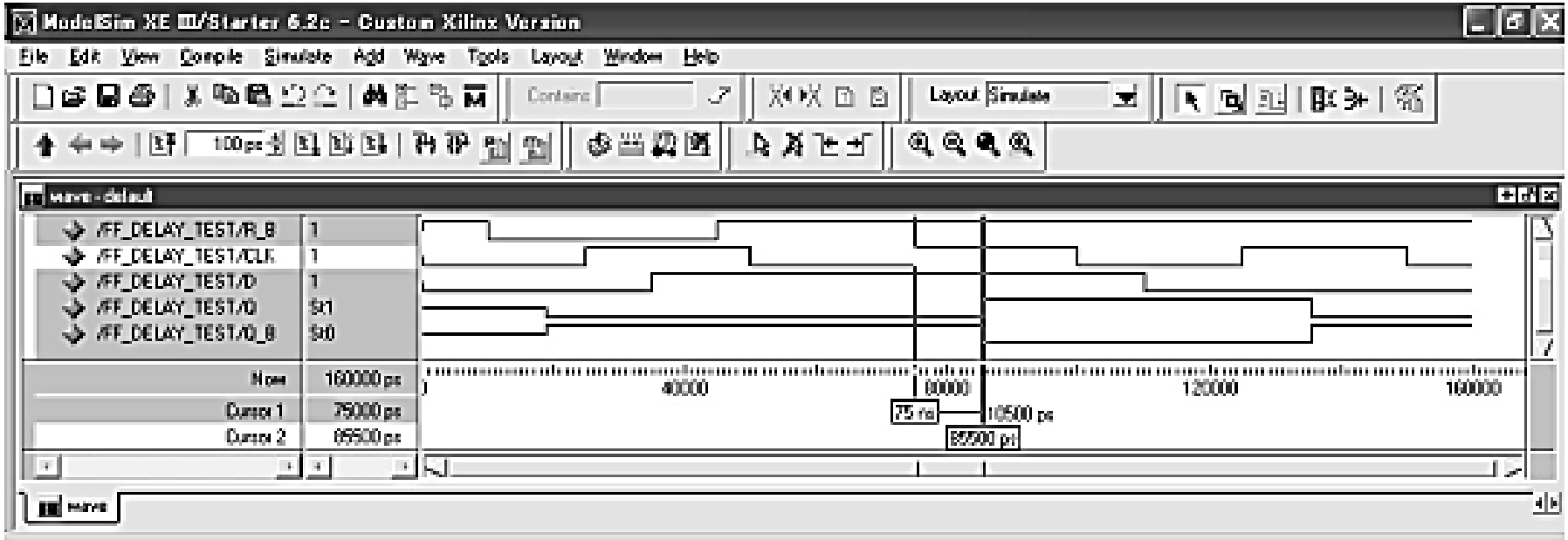
- always 是用來描述序向電路的語法；指定的事件發生時, 在 delay 指定的時間內中斷目前的執行

  always @ (posedge CLK) #10 OUT <= A & B;

  只要 CLK 有正緣變化的話會先中斷執行 10 個時間單位, 之後再把「A and B」的結果代入到 OUT

6.12 正反器的傳遞延遲時間

- 傳遞延遲時間，就是「從造成輸出變化的原因發生，一直到輸出變化為止的時間」
- 正反器的傳遞延遲時間可使用 if 的 delay 來描述傳遞延遲時間



▲ 圖 6.47 正反器的傳遞延遲時間的模擬結果