Verilog 硬體描述語言

Verilog HDL

A Guide to Digital Design and Synthesis

IEEE 1364-2001 Compliant

第10章 時序與延遲

- 針對設計硬體所做的功能驗證，通常只是在功能上來看電路的正確性，而不會去檢查真正的相關時序的正確性。
- 隨著科技進步，設計的電路越來越小，運作頻率越來越高，相對的時序正確性也越來越重要。
- 為了模擬速度的考量，通常會採用靜態時序模擬的方法，設計工程師先做功能性的模擬，再做靜態時序模擬。

10.1 延遲的模型

- 在Verilog中，一般有三種延遲的模型:分散式、整組式與接腳對接腳(路徑)延遲模型。
- **10.1.1 分散式延遲模型**
  - 本模型是以每一個元件為單位來指定延遲參數，圖10-1可看出在模組M如何指定分散式延遲。

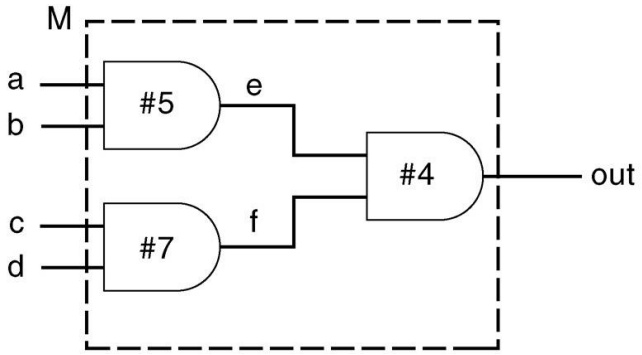


圖10-1　分散式延遲模型

10.1 延遲的模型

- 10.1.1 分散式延遲模型
  - 分散式延遲模型可藉由指定個別邏輯閘的延遲數值，或是使用assign敘述來建立使用。當輸入改變時，輸出會經過指定的延遲時間後，才會有所變化。
  - 範例10-1 分散式延遲

```
// 如何在邏輯閘層次中使用分散式延遲
module M(out, a, b, c, d);
output out;
input a, b, c, d;

wire e, f;
// 每一個邏輯閘都有不同的延遲時間設定
and #5 a1(e, a, b);
and #7 a2(f, c, d);
and #4 a3(out, e, f);
endmodule
```

```
// 如何在資料流敘述中，來設定延遲時間。
module M (out, a, b, c, d);
output out;
input a, b, c, d;

wire e, f;
// 在每一個敘述中指定延遲
assign #5 e = a & b;
assign #7 f = c & d;
assign #4 out = e & f;
endmodule
```

10.1 延遲的模型

- 10.1.2 整組式延遲模型
  - 整組式延遲模型主要是針對各個模組去指定，只在最後輸出端上做指定，或者可以結合所有的延遲，一次指定在最後輸出端。
  - 範例10-2 整組延遲

```
// 整組延遲模型
module M ( out , a, b, c, d);
output out;
input a, b, c, d;
wire e, f;
and a1(e, a, b);
and a2(f, c, d);
and #11 a3(out, e, f);   // 只有在最後一級才指定延遲時間
endmodule
```

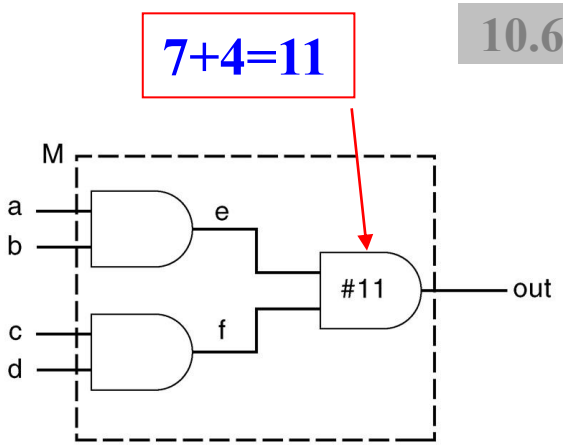


圖10-2 整組式延遲模型

10.1 延遲的模型

• 10.1.3 接腳對接腳延遲模型

- 本模型主要是根據每一條輸入到輸出的路徑，來指定延遲的時間。
- 從圖10-3可以看到如何計算每一條路徑的延遲時間，接腳對接腳的延遲時間，通常可從資料手冊查閱。這些資料必須依照線路特性，透過詳細的 SPICE 模擬分析所得到。
- 若不採用此模型，當面對的邏輯閘、資料流或行為描述改變，或加入了混合設計，要描述延遲時間將需一一改變，而難以完成。
- 採用接腳對接腳的模型，不用擔心模組內的敘述如何變化，只要接腳不變就好。

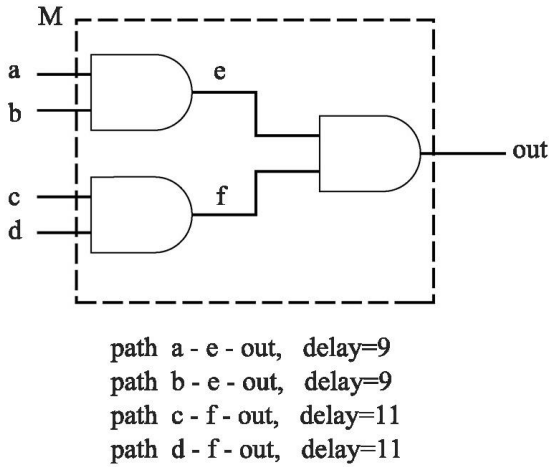


圖10-3　接腳對接腳延遲模型

10.2 路徑延遲模型

- 10.2.1 指定區塊
  - 從來源端(輸入埠或輸出入埠)到目的端(輸出埠或輸出入埠)，所指定的路徑延遲統稱為模組路徑延遲。
  - 在Verilog語言中來指定路徑延遲，必須配合specify與endspecify兩個關鍵字來使用，這兩個字所包圍的敘述會組成一個特定區塊，具有三種功能:
    - 指定由接點到接點的延遲時間
    - 設定線路時脈檢查參數
    - 定義specparam常數

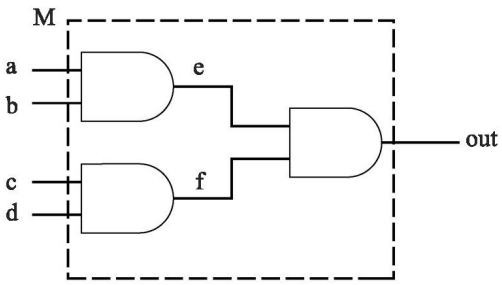
10.2 路徑延遲模型

- 10.2.1 指定區塊
  - 以圖10-3為例，若以指定區塊觀念來寫，就如範例10-3所示。
  - 特定區塊不可以包含在其他區塊中，例如包含在以always或是initial開始的區塊就是不合法的。特定區塊中的敘述必須是很明確的敘述。
  - **範例10-3 接點到接點的延遲**

```
// 接點到接點的延遲
module M ( out , a, b, c, d);
output out;
input a, b, c, d;

wire e, f;
// 特定區塊
specify
    ( a = out ) = 9;
    ( b = out ) = 9;
    ( c = out ) = 11;
    ( d = out ) = 11;
endspecify

// 邏輯閘別名
and a1(e, a, b);
and a2(f, c, d);
and a3(out, e, f);
endmodule
```



path a - e - out, delay=9
path b - e - out, delay=9
path c - f - out, delay=11
path d - f - out, delay=11

10.2 路徑延遲模型

- 10.2.2 在指定區塊內
  - 本節將說明有那些指令可以寫進特定區塊中。
  - 平行連接
    - 所有的路徑延遲敘述都有一個來源端與一個目的端。
    - 一個平行連接的敘述包含來源端與目的端，中間以=>符號做連接。
    - 用法: (來源端) => (目的端) = <延遲時間>
    - 在一個平行連接的敘述中，來源端目的端的位元數必須一致，可以是一對一或是多對多。

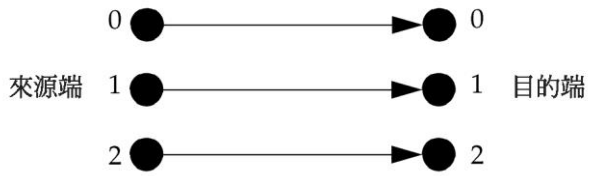


圖10-4　平行連接一對一

10.2 路徑延遲模型

- 10.2.2 在指定區塊內
  - 平行連接
  - 範例10-4 平行連接

```
// 一個位元對一個位元的連接
(a = out ) = 9;
// 向量的連接，兩邊都是四位元的向量，分別是 a[3:0]與 out[3:0]
// a[3:] 來源端，out 目的端
(a ⇒ out ) = 9;
// 如果上一行敘述分解成一各個位元的敘述
( a[0] ⇒ out[0] ) = 9;
( a[1] ⇒ out[1] ) = 9;
( a[2] ⇒ out[2] ) = 9;
( a[3] ⇒ out[3] ) = 9;

// 接著底下是一個錯誤的示範，a [4:0]是一個五個位元的向量，
// out[3:0] 是一個四位元的向量。這兩者的位元數不一致。
( a⇒ out ) =9;     // bit width does not match
```

10.2 路徑延遲模型

- 10.2.2 在指定區塊內
  - 完全連接
    - 一個完全連接的敘述包含來源端與目的端，中間以 * 符號做連接。　　用法: (來源端) *> (目的端) = <延遲時間>
    - 在一個完全連接的敘述中，每一個在來源端的位元都與每一個在目的端的位元有關係，對於來源端是一對多的關係指定，對於目的端則是多對一的關係指定。
    - 與平行連接不同處在於，完全連接來源端的位元數量與目的端的位元數量不一定要一致。

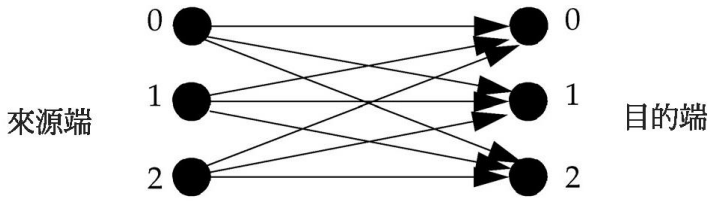


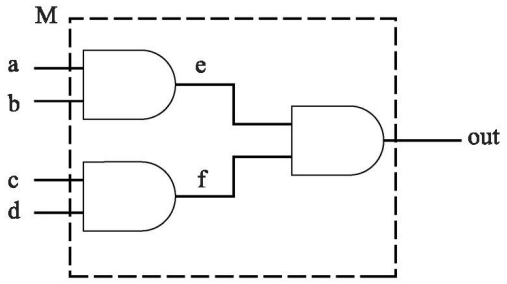
圖10-5　完全連接

10.2 路徑延遲模型

- 10.2.2 在指定區塊內
  - 完全連接
  - 將範例10-3的指定轉換成此模型的寫法，就如範例10-5所示。
  - 範例10-5 完全連接

```
// 完全連接
module M ( out , a, b, c, d);
output out;
input a, b, c, d;
wire e, f;
// 完全連接
specify
( a ,b *> out ) = 9;
( c,d  *> out ) = 11;
endspecify
```

```
// 邏輯閘別名
and a1(e, a, b);
and a2(f, c, d);
and a3(out, e, f);
endmodule
```



path a - e - out, delay=9
path b - e - out, delay=9
path c - f - out, delay=11
path d - f - out, delay=11

10.2 路徑延遲模型

- **完全連接**
- 用完全連接的方式非常有效率，當輸出入的位元數目都很大時，幾乎不可能一一指定它們之間的所有關係。

```
// a[31:0] 是一個32位元的向量，out[15:0]是一個16位元的向量
// 他們之間的延遲時間是 9 單位

specify
  (a *> out) = 9;   // 如果使用平行指定的方式，
                    // 將需要 32*16 = 352 行敘述才做得到
endspecify
```

- **訊號緣敏感路徑(Edge-Sensitive Paths)**
- 此路徑可定義從輸入到輸出的延遲時序，此延遲只在輸入端特定信號緣出現才存在。

```
// 當clock出現上升緣時，由clock訊號延續到out訊號的路徑，
// 具有 10 單位的上升延遲，以及 8 單位的下降延遲。此時路徑
// 是由 in 到 out，並且 in 訊號傳遞到 out 訊號時不會被反相
(posedge clock +> (out +: in)) = (10 : 8);
```

10.2 路徑延遲模型

- **specparam 敘述**
- 特別的參數可以利用 specparam 關鍵字，宣告在特定區塊中。
- 將相關意義的時間參數，利用此關鍵字宣告成一組文字，如此既易讀也有益於設計的維護。這種指定方式只能用在他所在的小區塊中。
- **範例10-6 Specparam**

```
// 在指定區塊中使用 specparam
specify
    // 定義參數
    specparam d_to_q =9;
    specparam clk-to_q =11;
    (d = q) = d_to_q;
    (clk = q ) = clk_to_q;
endspecify
```

10.2 路徑延遲模型

- 條件式路徑延遲
- 使用時機: 如果延遲的時間會隨著一些輸入的組合而改變時。
- 條件式路徑延遲可縮寫為SDPD(state dependent path delays)
- 範例10-7 條件式路徑延遲

```
module M ( out , a, b, c, d)
output out;
input a, b, c, d;
wire e, f;
// 使用條件式接腳對接腳延遲模型
specify
//由 a 的狀態來決定延遲時間
if ( a ) ( a= out ) = 9;
if (~a) ( a = out) = 10;
// 由 b 與 c 的狀態，來決定延遲時間。
if (b&c) ( b = out ) = 9;
if(~ (b&c) ) ( b= out ) = 13;
// 使用集合運算元，以及完全連接。
if({c,d} == 2'b01 ) (c,d *> out 0 )= 11;
if(~{c,d} == 2'b01 ) (c,d *> out 0)= 11;

endspecify

// 邏輯閘別名
and a1(e, a, b);
and a2(f, c, d);
and a3(out, e, f);
endmodule
```

10.2 路徑延遲模型

- **上升延遲，下降延遲與關閉延遲**
- 在接腳對接腳的延遲模型中，我們可以針對這三種時間參數做設定。
- 這三種參數配合四種位元表示法(包括0、1、x與z)，可以有很多組合，但只有一、二、三、六或十二可以使用，其他都是不允許的組合。表示的順序也必須嚴格地遵守上升，下降與關閉延遲的順序。
- **範例10-8 上升延遲，下降延遲與關閉延遲時間**

```
// 一個延遲時間
specparam t_delay = 11;
(clk = q) = t_delay;

// 指定兩個延遲時間，包含上升與下降。
// 上升時間 0->1, 0->z, z->1
// 下降時間 1->0, 1->z, z->0
specparam t_rise = 9, t_fall = 13
(clk => q) = ( t_rise , t_fall )
```

```
// 指定三個延遲時間，上升，下降與關閉。
// 上升時間 0 -> 1, z -> 1
// 下降時間 1 -> 0, z -> 0
// 關閉時間 0 -> z, 1 -> z
specparam t_rise=9, t_fall=13, t_turnoff = 11;
( clk => q ) = ( t_rise, t_fall, t_turnoff);
```

10.2 路徑延遲模型

- 上升延遲，下降延遲與關閉延遲
- 範例10-8 上升延遲，下降延遲與關閉延遲時間(續)

```
// 指定六個延遲，依照 0->1, 1->0, 0->z, z->1, 1->z, z->0 的順序
specparam t_01= 9, t_10= 13, t_0z=11;
specparam t_z1= 9, t_1z= 11, t_z0=13;
( clk= q ) = (t_01, t_10, t_0z, t_z1, t_1z, t_z0);
// 指定十二個延遲，順序分別是 0->1,1->0,0->z,z->1,1->z,z->0,
// 0->x, x->1, 1->x,  x->0, x->z, z->x
specparam t_01=9,t_10=13,t_0z=11;
specparam t_z1=9,t_1z=11,t_z0=13;
specparam t_0x=4,t_x1=13,t_1x=5;
specparam t_x0=9,t_xz=11,t_zx=7;
(clk => q) = (t_01, t_10, t_0z, t_z1, t_1z, t_z0, t_0x,
              t_x1, t_1x, t_x0, t_xz, t_zx);
```

10.2 路徑延遲模型

- 最小值、最大值與標準值
- 在接腳對接腳的模型中，一樣可以指定最小值、最大值與標準值的延遲時間。
- 範例10-9 配合著最小值、最大值與標準值的路徑延遲

```
// 指定上升，下降與關閉的延遲時間。
// 每一個值都分別有最小值、最大值與標準值。
specparam t_rise=8:9:10, t_fall = 12:13:14,
          t_turn_off = 10:11:12;
(clk => q ) = ( t_rise, t_fall, t_turnoff);
```

10.2 路徑延遲模型

- 處理不明值(x)的狀況
- 在Verilog中，如果狀態值的變化與不明狀態有關，這時Verilog會採取一個很悲觀的方式來處理相關的延遲時間，尤其是當與不明狀態有關的時間參數未被設定的時候。
- 兩種狀態之處理:
  - 由不明狀態切換到已知狀態，這時候將參考會發生的**最大延遲時間**。
  - 由已知狀態切換到不明狀態，將會取用**最小的可能延遲時間**。

10.2 路徑延遲模型

- 從範例10-8中的六個延遲時間設定例子來看:

```
// 指定六個延遲，依照 0->1, 1->0, 0->z, z->1, 1->z, z->0 的順序
specparam t_01= 9, t_10= 13, t_0z=11;
specparam t_z1= 9, t_1z= 11, t_z0=13;
( clk= q ) = (t_01, t_10, t_0z, t_z1, t_1z, t_z0);
// 指定十二個延遲，順序分別是 0->1,1->0,0->z,z->1,1->z,z->0,
// 0->x, x->1, 1->x,  x->0, x->z, z->x
```

- 從下表可以看出，當切換到不明狀態時會產生的可能情況(取最小):

切換狀態	延遲時間
0->x	min(t_01, t_0z) = 9
1->x	min(t_10, t_1z) = 11
z->x	min(t_z0, t_z1) = 9
x->0	max(t_10, t_z0) = 13
x->1	max(t_01, t_z1) = 9
x->z	max(t_1z, t_0z) = 11

10.3 檢查時間設定

- 在前面學到，能使用路徑延遲時間的設定方式，會比針對每一個邏輯閘來設定延遲時間的方式好。
- 本節將學會如何去設定模擬執行時期的時間檢查參數。
- 在微處理器的設計中，要能確保高速，而且不允許時序錯誤的順序邏輯中，唯有做好時間的驗證，才能有正確的操作。
- 在Verilog中，我們只介紹三種比較簡單常用的設定方法，包括$setup、$hold與$width，而且這三種方式都只用在指定區塊中。

10.3 檢查時間設定

- 10.3.1 $setup與$hold的檢查方式
  - $setup的指令設定，主要是針對訊號來設定時間， 而$hold指令則是設定訊號的保持時間。
  - $setup指的是，在設定的觸發訊號到達前，應該在多久時間內，原規劃進入暫存器的值，就要先送到輸入端。
  - $hold指令則是指，在設定的觸發訊號到達後，已進入暫存器的值應該在輸入端保持多久時間。

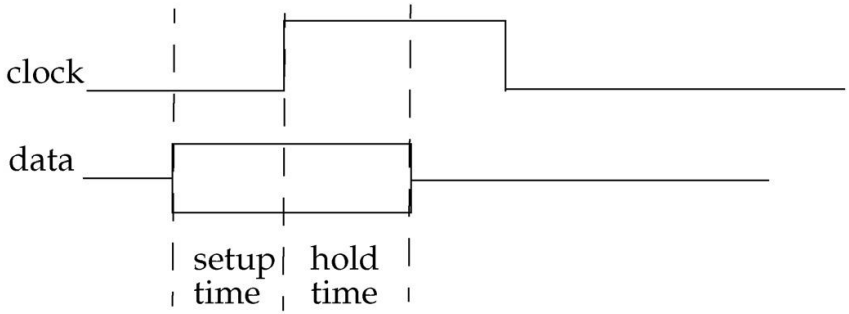


圖10-6　設定與保持時間

10.3 檢查時間設定

- 10.3.1 $setup與$hold的檢查方式
  - **$setup功能**
  - 指令用法: $setup(待檢查的訊號、參考訊號、時間限制)
  - **待檢查的訊號**: 要檢查是否違反時序的訊號
  - **參考訊號**: 檢查時序需要一個參考訊號，才能開始檢查時脈的正確性
  - **時間限制**: 檢查待檢查訊號的最小設定時間值
  - 如果(參考時間 – 待觀察時間) < 時間限制，時脈的設定值違反這一個設定時間限制，則系統將會出現錯誤訊息，告知錯誤的發生。

```
// 檢查設定時間
// 使用時脈輸入當作參考訊號
// 如果 (T_reference_even – T_data_even) < 3 則發出警告)
specify
   $setup (data, posedge clock, 3);
endspecify
```

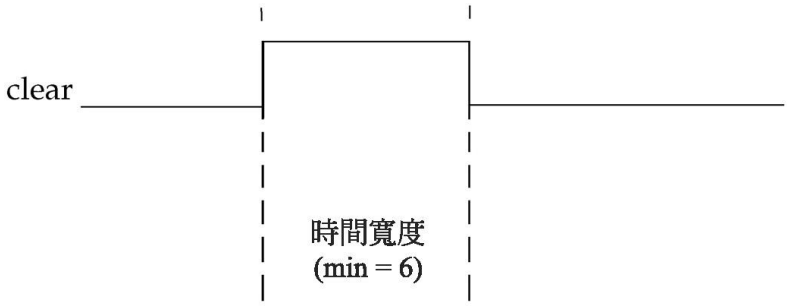
10.3 檢查時間設定

- 10.3.1 $setup與$hold的檢查方式
  - **$hold功能**
  - 指令用法: $hold(參考訊號、待檢查的訊號、時間限制)
  - **參考訊號**: 檢查時序需要一個參考訊號，才能開始檢查時脈正確性
  - **待檢查的訊號**: 要檢查是否違反時序的訊號
  - **時間限制**: 檢查待檢查訊號的最小保持時間值
  - 如果(待觀察時間–參考時間) < 時間限制，時脈的保持值違反這一個保持時間限制，則系統將會出現錯誤訊息，告知錯誤的發生。

```
// 檢查保持時間
// 使用時脈輸入當作參考訊號
// data 是待觀察值
// 如果 (待觀察時間–參考時間 ) < 5 則系統會發出警告)
specify
   $hold ( posedge clock, data, 5);
endspecify
```

10.3 檢查時間設定

- 10.3.2 $width 檢查
  - 有時我們會去檢查一個波形的寬度，是否符合我們的要求。



  - 用法: $width(參考訊號，最小時間寬度限制)
  - **參考訊號**: 必須是一個邊緣觸發的訊號
  - **最小時間寬度限制**: 檢查待檢查訊號的最小時間寬度值
  - 如果(參考訊號的指定邊緣出現到再度改變值的時間差) < 時間限制，表示時脈的寬度值違反這一個時間寬度限制，則系統將會出現錯誤訊息，告知錯誤的發生。

10.3 檢查時間設定

- 10.3.2 $width 檢查
  - 使用範例:

```
// 檢查寬度
// 使用時脈輸入當作參考訊號
// 5 個單位的時間當作限制值
// 時脈輸入在正緣觸發後，必須保持五個時間單位不變
specify
   $width ( posedge clear, 5);
endspecify
```

10.4 延遲時間加入原有的設計

- 如何將延遲時間加入原有的設計，對整個模擬的過程是非常重要的一環。細節可參考開放Verilog基金會所出的OVI Standard Delay File (SDF) Format Manual。
- **步驟**:

1. 此設計工程師使用暫存器轉換導向(RTL)的語法來設計，並作功能上的模擬驗證。
2. 設計被邏輯合成工具轉換到邏輯閘層次的設計。
3. 有了邏輯閘層次的設計後，設計工程師使用具有時脈關係的模擬分析，來檢查時脈前後彼此的關係，並利用固定的延遲時間來檢查時脈的正確性。
4. 接著，使用放置與佈線工具，將所有邏輯閘放好及完成接線。產生的電路佈局可得出細節的電阻值與電容值。

10.4 延遲時間加入原有的設計

- **步驟**:

5. 將這些電阻值與電容值轉換成對應的延遲時間細節，並加入原有的設計中，這時可以作更實際的時間相關模擬與驗證。
6. 如果線路設計不符合要求，必須回到步驟 2 到步驟 5 重作一次。如果還是不能符合要求，則必須回到步驟 1，重新來設計程式。

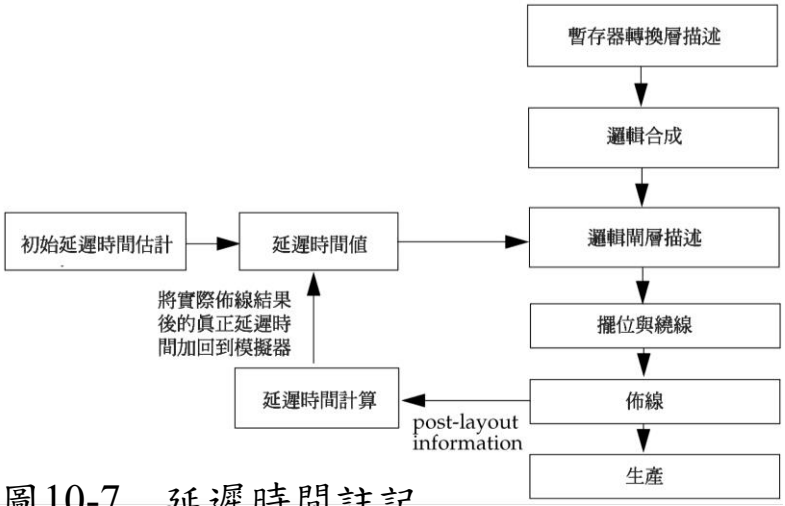


圖10-7　延遲時間註記