Verilog 硬體描述語言

Verilog HDL

A Guide to Digital Design and Synthesis

IEEE 1364-2001 Compliant

第12章 自定邏輯閘

- Verilog語言不需特別的指定，就有基本邏輯閘的且(and)、非且(nand)、或(or)、非或(nor)與反相(inv)等功能。
- 我們有時候也需特別定義一些邏輯閘，這些自行定義的獨一無二邏輯閘，其地位高於自己定義的模組(module)，且同等於基本邏輯閘。
- 兩種自己定義的邏輯閘
  - **組合邏輯閘**:輸出只與當時的輸入有關
  - **循序邏輯閘**:下一個時間的輸出值與當時的輸入值及輸出值都有關係

12.1 自己定義邏輯閘的基本概念

- **12.1.1 自己定義的邏輯閘的可用基本關鍵字**
- 自行定義的邏輯閘都是以關鍵字primitive開始，依序給予名字、輸出訊號名以及輸入訊號名。如果設計的是循序電路，輸出端還要宣告reg的資料型態。

```
// 定義邏輯閘的名字與輸出入埠
primitive < 名字>(
<輸出端訊號名>(只可以有一個)
<輸入端訊號名>);(不限個數)

// 輸出入埠宣告
output <輸出埠名字>;
input  <輸入埠名字>;
reg <輸出埠名字>;(通常只在循序邏輯中來使用)

// 初始化邏輯閘(通常只在循序邏輯中來使用)
initial <輸出埠名字>=<指定初始值;>

// 邏輯閘狀態表
table
    <列表 >
endtable

// 結束宣告
endprimitive
```

圖12-1　自己定義的邏輯閘可用的關鍵字

12.1 自己定義邏輯閘的基本概念

- **12.1.2 自己定義的邏輯閘使用規則**
  - 自己定義的邏輯閘可以接受一個或多個的輸入埠。
  - 自己定義的邏輯閘只能有一個輸出埠，而且必須寫在埠定義的最前面，不允許有多個輸出埠。
  - 輸出埠使用關鍵字output來定義，在循序邏輯中必須配合上關鍵字reg的宣告。
  - 輸入埠使用關鍵字input來定義。
  - 在循序邏輯中可以使用關鍵字initial來定義狀態初始值。
  - 狀態表只處理0、1、x這三種邏輯值的關係，而不處理z的值，z的輸入則當作x來看待。
  - 自己定義的邏輯閘不能等到在模組內才定義，只能在模組內使用。
  - 自己定義的邏輯閘內並沒有輸出入雙向埠的定義。

12.2 自己定義的組合邏輯電路

- 組合邏輯電路通常是看輸入值的變化，參考狀態真值表來決定輸出值的變化。
- **12.2.1 自訂組合邏輯邏輯閘的定義**
  - 在組合邏輯中，最重要的就是輸出入之間的狀態真值表。
  - 解釋狀態真值表的最佳方式是以邏輯且(and)作為範例
  - **範例12-1 upd_and**

```
// 名字與埠定義
primitive udp_and(out, a, b);

// 宣告
output out;   // 輸出宣告在先
input a, b;

// 定義狀態真值表
table
    // 底下為了容易閱讀起見，依照順序來排列。
    // a   b  : out ;
       0   0  : 0;
       0   1  : 0;
       1   0  : 0;
       1   1  : 1;

endtable // 結束宣告

endprimitive // 結束定義
```

12.2 自己定義的組合邏輯電路

- 12.2.1 自訂組合邏輯邏輯閘的定義
  - 範例12-1 upd_and
  - 與圖12-1不同處是輸出並未宣告為reg格式，以及與initial敘述並未用到，這些只會在循序的自定邏輯閘中用到。
  - 自訂邏輯閘也支援ANSI C形式的宣告，允許一個埠的定義與其列表結合在一起。
  - 範例12-2 ANSI C形式的組合邏輯閘宣告

```
// 名稱與輸出入埠列表
primitive udp_and(output out,
                  input a,
                  input b);
...
endprimitive // 結束 udp_and 的宣告
```

12.2 自己定義的組合邏輯電路

- **12.2.2 狀態真值表**
  - 狀態真值表的設定功能必須先了解該如何寫，參考udp_and內容排列:

1. 必須依照輸出入埠的定義順序，如果順序錯了並不會有警告，但是會有不可預期的輸出結果。
2. 輸入與輸出的設定中間相隔一個 ":" 符號。
3. 每一行設定結尾必須以 ";" 符號終結。
4. 所有預先知道的輸出入對應關係，都必須清楚地寫進真值表，不然遇到不存在的對應關係，輸出都會對應到 x 輸出值。

- 以udp_and內容為例，若遇到a=x且b=0，正常運作下的輸出值應該是0，但因真值表未定義，所以輸出值是x。

12.2 自己定義的組合邏輯電路

- 12.2.2 狀態真值表
  - 範例12-3 udp_or

```
primitive udp_or(out, a, b);

output out;
input a, b;

table
    // a  b  :  out;
       0  0  :  0;
       0  1  :  1;
       1  0  :  1;
       1  1  :  1;
       x  1  :  1;
       1  x  :  1;
endtable

endprimitive
```

12.2 自己定義的組合邏輯電路

- 12.2.3 可以忽略值的速記符號
  - 在**範例12-3**可以看到，只要有一個輸入埠值為1，另一個輸入埠值不論為何，輸出值都是1。此時應該有一個符號來直接代表0、1與x。在Verilog語言中以**?**符號來代表此設定。

```
primitive udp_or(out, a, b);

output out;
input a, b;

table
    // a  b  :  out;
       0  0  :  0;
       1  ?  :  1;
       ?  1  :  1;
       0  x  :  x;
       x  0  :  x;
endtable

endprimitive
```

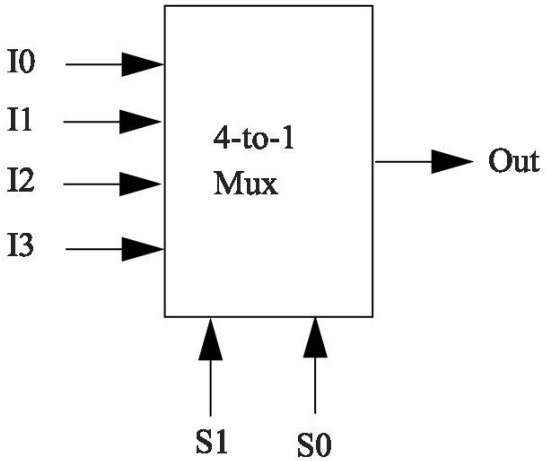
12.2 自己定義的組合邏輯電路

- 12.2.4 使用自己定義的邏輯閘
  - 以全加器為例，利用前面upd_and與upd_or，再配合其他相關指令一起來寫程式。
  - 範例12-4 使用自定邏輯閘

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

// 輸出入埠定義
output sum, c_out;
input a, b, c_in;

// 內部連線
wire s1, c1, c2;

// 混合使用原有與自定邏輯閘
xor(s1, a, b); // 原有邏輯閘
udp_and(c1, a, b); // 自定邏輯閘
xor(sum, s1, c_in); // 原有邏輯閘
udp_and(c2, s1, c_in); // 自定邏輯閘
udp_or(c_out, c2, c1); // 自定邏輯閘

endmodule
```

12.2 自己定義的組合邏輯電路

- 12.2.5 一個使用自己定義的組合邏輯閘範例
  - 使用4對1多工器為例，將第五章5.1.3範例以自定邏輯閘方式來重新改寫。
  - 多工器的輸出通常只有一個輸出埠，非常適合使用自訂邏輯的方式來描述。



S1	S0	Out
0	0	I0
0	1	I1
1	0	I2
1	1	I3

圖12-2　4對1 UDP多工器

12.2 自己定義的組合邏輯電路

- 12.2.5 一個使用自己定義的組合邏輯閘範例
  - 這一個多工器有六個輸入埠與一個輸出埠，自訂邏輯閘敘述如下。
  - 範例12-5 4對1多工器的自訂邏輯閘

```
// 4對1多工器
primitive mux4_to_1(out, i0, i1, i2, i3, s1, s0);
//
output out;
input i0, i1, i2, i3;
input s1, s0;
table
  // i0 i1 i2 i3  ,  s1 s0 : out;
     1  ?  ?  ?      0  0  : 1  ;
     0  ?  ?  ?      0  0  : 0  ;
     ?  1  ?  ?      0  1  : 1  ;
     ?  0  ?  ?      0  1  : 0  ;
     ?  ?  1  ?      1  0  : 1  ;
     ?  ?  0  ?      1  0  : 0  ;
     ?  ?  ?  1      1  1  : 1  ;
     ?  ?  ?  0      1  1  : 0  ;
     ?  ?  ?  ?      x  ?  : x  ;
     ?  ?  ?  ?      ?  x  : x  ;
endtable

endprimitive
```

12.2 自己定義的組合邏輯電路

- 12.2.5 一個使用自己定義的組合邏輯閘範例
  - 要指定一個4對1多工器，已經用了一個很大的真值表，如果要再增加輸入埠的數目，記憶體的需求將以指數次方的趨勢增加。何時用自訂邏輯閘比較適合?只有當你知道輸出入對應關係，但又不是很清楚該用甚麼邏輯閘來處理，就可以使用自訂邏輯閘直接開始設計。
  - **範例12-6 4對1多工器的測試模組**

```
// 測試模組，不需輸出入埠。
module stimulus;

// 宣告使用的變數
reg IN0, IN1, IN2, IN3;
reg  S1, S0;

// 宣告多工器的輸出埠
wire OUTPUT;

// 使用 mux4_to_1 自定邏輯閘
mux4_to_1 mymux(OUTPUT, IN0, IN1, IN2, IN3, S1, S0);
```

12.2 自己定義的組合邏輯電路

- 12.2.5 一個使用自己定義的組合邏輯閘範例
  - 範例12-6 4對1多工器的測試模組

```
// 測試輸入
initial
begin
    // 預先給入四個輸入埠值
    IN0 = 1; IN1 = 0; IN2 = 1 ; IN3 = 0;
    #1 $display(" IN0= %b, IN1= %b, IN2= %b, IN3= %b \n",
     IN0, IN1, IN2, IN3);
// 選擇 IN0
    S1=0; S0=0;
   #1 $display(" S1= %b, S0= %b, OUTPUT= %b \n",S1,S0,OUT-
    PUT);

    // 選擇 IN1
    S1=0; S0=1;
    #1 $display(" S1= %b, S0= %b, OUTPUT= %b \n",S1,S0,
    OUTPUT);
```

12.2 自己定義的組合邏輯電路

- 12.2.5 一個使用自己定義的組合邏輯閘範例
  - 範例12-6 4對1多工器的測試模組

```
        // 選擇 IN2
        S1=1; S0=0;
        #1 $display(" S1= %b, S0= %b, OUTPUT= %b \n",S1,S0,
        OUTPUT);

        // 選擇 IN3
        S1=1; S0=1;
        #1 $display(" S1= %b, S0= %b, OUTPUT= %b \n",S1,S0,
        OUTPUT);
    end

    endmodule
```

  - 測試結果：

```
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
```

12.3 循序自定邏輯閘

- 循序邏輯與組合邏輯不論是在定義上或使用上都有一些不同點:
  - 循序邏輯的輸出埠必須再加上**reg**的定義。
  - 可以利用**initial**關鍵字來做輸出值初始化。
  - 狀態真值表的定義不完全相同。
  - 在狀態真值表中分成三個部份:輸入埠、現在狀態與下一個狀態，彼此以符號"**:**"區隔。
  - 輸入埠可以利用訊號的位準，或是位準改變的邊緣觸發(edge-trigger)作為參考值。
  - 現在狀態表示輸出值。
  - 下一個狀態由輸入埠與現在輸出埠的值所決定，代表下一個時間輸出埠的值。
  - 所有可能發生的狀態最好都寫進去，避免發生未知的狀態。

12.3 循序自定邏輯閘

- 12.3.1 訊號位準敏感循序自定邏輯
  - 位準敏感(level-sensitive)循序邏輯在輸入位準改變時會改變輸出值，正反器就屬於這一類。
  - 以下為一個有清除功能的簡單正反器

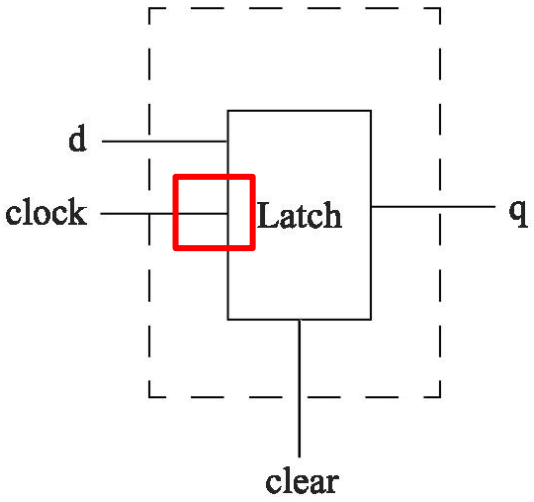


圖12-3　正反器

12.3 循序自定邏輯閘

- **12.3.1 訊號位準敏感循序自定邏輯**
  - 以上圖來說，當清除訊號(clear)為1時，輸出(output)永遠是0，如果清除訊號為0，當clock為1時，q會等於d的值，當clock為0時，q會保持在以前的值不變。
  - **範例12-7 訊號位準敏感循序自定邏輯**

```
// 使用自定邏輯來描述一個位準敏感的正反器
primitive latch(q, d, clock, clear);

// 宣告輸出入埠
output q;
reg q; // 暫存器 q 引起宣告儲存內部
input d, clock, clear;

// 初始化
initial
    q = 0;  // 初始化輸出值 0

// 狀態真值表
table
   //  d  clock  clear  : q : q+ ;
       ?  ?      1      : ? : 0  ;  // 清除狀態
       1  1      0      : ? : 1  ;  // q = d
       0  1      0      : ? : 0  ;  // q = d
       ?  0      0      : ? : -  ;  // clock = 0,
                                       輸出值不變化
endtable

endprimitive
```

12.3 循序自定邏輯閘

- **12.3.1 訊號位準(level)敏感循序自定邏輯**
  - 循序自訂邏輯閘也支援ANSI C形式的宣告，只是要注意在輸出埠的定義中進行**reg**的宣告，同時也可以做輸出埠數值的初始化。
  - **範例12-8 ANSI C形式的循序自定邏輯閘宣告**

```
// 用自定邏輯閘來定義訊號位準敏感的正反器
primitive latch(output reg q = 0,
                input d, clock, clear);
...
...
endprimitive
```

12.3 循序自定邏輯閘

- **12.3.2 訊號緣(edge)敏感循序自定邏輯**
  - 訊號緣敏感循序邏輯根據訊號緣的改變，與位準來決定輸出的變化。一般來說，暫存器通常是訊號緣敏感循序邏輯。
  - 以下為一個有清除輸入功能的負緣觸發暫存器

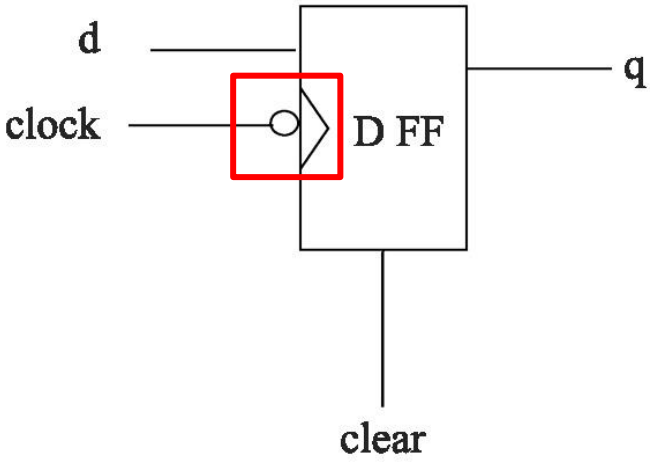


圖12-4　負緣觸發暫存器

12.3 循序自定邏輯閘

- **12.3.2 訊號緣(edge)敏感循序自定邏輯**
  - 以上圖來說，當清除訊號(clear)為1時，則輸出(output)永遠是0。如果清除訊號為0，則暫存器正常工作。在正常工作時，剛好clock有負緣觸發，也就是從1變化到0，則q將等於d的值。其他的狀態變化，q都不會變化。
  - **範例12-9 有清除輸入的負緣觸發暫存器**

```
primitive edge-dff(output reg q =0,input d, clock , clear);
table
   // d   clock clear : q : q+ ;
      ?     ?     1    : ? : 0  ; // 清除狀態
      ?     ?    (10)  : ? : -  ; // 忽略清除訊號的負緣變化
      1    (10)   0    : ? : 1 ; // 在 clock 訊號負緣時取資料
      0    (10)   0    : ? : 0 ; // 在 clock 訊號負緣時取資料
      ?    (1x)   0    : ? ; - ; // 其他狀態，都不予理會
      ?    (0?)   0    : ? ; - ; // 其他狀態，都不予理會
      ?    (x1)   0    : ? : - ; // 其他狀態，都不予理會
     (??)  ?     0    : ? : - ; // 其他狀態，都不予理會
endtable
endprimitive
```

12.3 循序自定邏輯閘

- 12.3.2 訊號緣(edge)敏感循序自定邏輯
  - 範例12-9 有清除輸入的負緣觸發暫存器
  - 以上例來說，我們可以看到邊緣的表示方法:
    - (10)表示一個負緣變化，數值從1變化到0。
    - (1x)表示從1變化到 x 未知狀態。
    - (0?)表示從0變化到 0、1或x 狀態，可能包含正緣觸發。
    - (??)表示任何有可能的變化組合，0、1、x變化到 0、1、x 。
  - 同樣地，我們也要仔細注意狀態真值表的指定情形，盡可能就所知狀態完整地指定所有的狀態。注意表中每一行同時只能有一個邊緣訊號，下面即是不允許的範例:

```
table
……
                (01)(10)0 : ? : 1 ; //錯誤的示範
……
endtable
```

12.3 循序自定邏輯閘

- **12.3.3 循序自定邏輯範例**
  - 這裡以一個四位元的漣波型(ripple)計數器為例，之前在6.5.3小節用T型正反器組成計數器，T型是由負緣觸發的D型正反器所組成。此處T型則用自訂邏輯閘來描述。
  - **範例12-10 使用自定邏輯的T型暫存器**

```
// 緣觸發 T 型暫存器
primitive T_FF(output reg q, input clk , clear);

// 宣告
output reg q;
input clk, clear;

// 不需要初始化

table
// clk   clear :   q   :   q+   ;
     // 非同步清除
       ?       1       :   ?   :   0   ;
       // 忽略清除訊號的負緣變化
       ?       (10)    :   ?   :   -   ;
      // 在 clock 訊號負緣的時候，將輸出反相。
     (10)     0       :   1   :   0   ;
     (10)     0       :   0   :   1   ;
      // 忽略 clock 訊號的正緣變化
      (0?)    0       :   ?   :   -   ;
endtable
endprimitive
```

12.3 循序自定邏輯閘

- 12.3.3 循序自定邏輯範例
  - 使用T型正反器(T_FF)來組成一個四位元漣波計數器， 必須用至少四個T型正反器，結果如以下範例所示。
  - 範例12-11 水波型計數器

```
// 水波型計數器
module counter(Q, clock, clear);

// 輸出入埠
output [3:0] Q;
input clock, clear;

// 使用 T_FF，別名部份可有可無。
T_FF tff0(Q[0],clock,clear);
T_FF tff1(Q[1],Q[0],clear);
T_FF tff2(Q[2],Q[1],clear);
T_FF tff3(Q[3],Q[2],clear);

endmodule
```

12.4 自定邏輯閘中的一些常用速記符號

- 使用(10)來表示負緣變化的方式其實蠻困難的，也不夠直覺。Verilog針對此一需求定義了一些速記符號來代表。

表12-1　UDP速記符號

速記符號	代表意義	說明
?	0, 1, x	不可用在輸出埠
b	0, 1	不可用在輸出埠
-	不改變	只能用在循序邏輯的輸出埠
r	(01)	正緣觸發
f	(10)	負緣觸發
p	(01),(0x),(x1)	有可能的正緣觸發
n	(10),(1x),(x0)	有可能的負緣觸發
*	(??)	所有可能的改變

12.4 自定邏輯閘中的一些常用速記符號

- 有了這些速記符號，就可以將範例12-7部份改寫如下。

```
table
   // d  clock  clear  :  q  :  q+ ;
      ?     ?      1    :  ?  :  0  ;  // 清除狀態
      ?     ?      f    :  ?  :  -  ;  // 忽略清除訊號的負緣變化
      1     f      0    :  ?  :  1  ;  // 在clock訊號負緣時取資料
      0     f      0    :  ?  :  0  ;  // 在clock訊號負緣時取資料
      ?    (1x)    0    :  ?  :  -  ;  // 其他狀態都不予理會
      ?     p      0    :  ?  :  -  ;  // 其他狀態都不予理會
      *     ?      0    :  ?  :  -  ;  // 其他狀態都不予理會
endtable
```

12.5 自定邏輯閘的使用方針

- 自訂邏輯閘只有功能部份，並不包含時脈或任何與製程有關的資訊，這種狀況就要用模組宣告來處理。
- 自訂邏輯閘只提供一個輸出埠，超過就不行使用。
- 自訂邏輯閘的輸入埠數目也不是無限的，而是根據所使用的模擬軟體所決定。組合邏輯至多有十個，循序邏輯則為九個。
- 因為自訂邏輯閘的處理方式是整個存入記憶體中，所以如果使用太多自訂邏輯，記憶體將會不夠。自訂邏輯的大小與輸入埠數目成指數成長的關係。
- 使用自訂邏輯的方式很方便，但不是一個很好的設計策略。

12.5 自定邏輯閘的使用方針

- 盡可能完整地來指定所有已知的狀態，若無指定的狀態，通通會是未知狀態，以簡化真值表。
- 使用速記符號有助於編寫一個正確的真值表，也能確保易於閱讀維護，還能簡化真值表的規模。但在模擬軟體中，整個真值表會存入記憶體中且展開，佔用的記憶體大小並不會因使用速記符號而縮小。
- 位準敏感的敘述決定權優先於位準邊緣敏感敘述，盡可能避免這樣的情況發生。