Verilog 硬體描述語言

Verilog HDL

A Guide to Digital Design and Synthesis

IEEE 1364-2001 Compliant

第9章 有用的程式技巧

9.1 程序持續指定

- 程序指定用來指定一個數值給暫存器，程序持續指定則使用有限週期的指定方式，指定數值到暫存器或線路。
- **9.1.1 assign 與 deassign**
  - 關鍵字assign 與 deassign 是用來表示第一種型態的程序持續指定。
  - 程序持續指定的左邊，可以是暫存器或是連續的暫存器，不能是一個接點的部分位元或暫存器的陣列。
  - 程序持續指定複寫正規程序指定，一般運用在週期控制。

9.1 程序持續指定

- 9.1.1 assign 與 deassign
  - 範例6-8的非同步設定負緣觸發D型正反器，在範例9-1中將使用 assign 與 deassign 來完成。
  - **範例9-1 程序持續指定D型正反器**

```
// 非同步設定負緣觸發 D 型正反器
module edge_dff(q, qbar, d, clk, reset);

// 輸出輸入
output q,qbar;
input d, c.k, reset;
reg q, qbar; // 宣告 q 和 qbar 是暫存器變數

always @(negedge clk) // 在 clock 負緣指定 q 和 qbar 的值
begin
    q = d;
    qbar =~d;
end
```

9.1 程序持續指定

- 9.1.1 assign 與 deassign
  - 範例9-1 程序持續指定D型正反器(續)

```
always @(reset) // 當reset為高電位使用程序持續指定複寫q和qbar
    if(reset)
    begin  // 如果 reset 為高電位，使用程序持續指定複寫 q 和 qbar
           assign q=1'b0;
           assign qbar=1'b1;
   end

   else
   begin  // 如果 reset 為低電位，移除複寫
          // 在移除複寫後， q = d and qbar=~d 將要等到下一個
             clock 負緣才能更改暫存器變數的值
          deassign q;
          deassign qbar;
    end

endmodule
```

9.1 程序持續指定

- **9.1.2 force 與 release**
  - 關鍵字force 與 release 是用來表示第二種型態的程序持續指定，其可複寫在暫存器(register)和接點(net)。
  - force 與 release 一般是用在互動式除錯程序中，強制(forced)給一個數值於暫存器或線路，會影響其他暫存器或線路。
  - 建議force 與 release 不要使用在設計區塊中，最好僅使用在模擬和除錯中。

暫存器的 force 與 release

```
module stimulus;
…
…
// 取別名 D 型正反器
edge_dff(Q, Qbar, D, CLK, RESET);
…
…
```

```
initial
begin // 在模擬時間 50 至 100 強制 dff.q 的值
   #50  force dff.q = 1'b0; // 強制 q =1 在 50
   #50  release dff.q;       //解強制 q 值在100
end
…
…
endmodule
```

9.1 程序持續指定

- **9.1.2 force 與 release**

接點的 force 與 release

- force 一個接點，則任何程序持續指定將被複寫，直到release為止。
- force 一個接點，可為一個表示式或數值，當接點被release時，馬上還原為原來的驅動值。

```
module top;
…
…
assign out = a&b&c;     // 持續指定 out
…
initial
    #50   force out = a|b&c;
    #50   release out;
end
…
…
endmodule
```

9.2 複寫參數

- 參數可以定義在模組中，在不同的編譯中可給予參數不同的值，而忽略先前所給予的值。
- **9.2.1 defparam 敘述**
  - 關鍵字defparam 可用來定義模組別名(instance)中之參數，模組別名之階層名可用來複寫參數值。
  - **範例 9-2 defparam敘述**

```
// 定義模組 hello_world
module hello_world;
parameter id_num = 0; // 定義一模組識別數 = 0

initial // 顯示識別數
        $display("Displaying hello_world id number = %d",
        id_num);
endmodule

// 定義最高層次模組
module top;

// 改變在這取別名模組的參數值
// 使用 defparam 敘述
defparam w1.id_num = 1,  w2.id_num=2;

// 取別名二個 hello_world 模組
hello_world w1();
hell0_world w2();

endmodule
```

9.2 複寫參數

- 9.2.1 defparam 敘述
  - 範例 9-2 defparam敘述
  - 模擬結果

```
Displaying hello_world id number = 1
Displaying hello_world id number = 2
```

  - 一個模組可以有許多 defparam 敘述，如今defparam 語法已被認定是不好的程式風格，建議以其他的Verilog HDL的程式風格來取代。
  - hello_world 中參數的定義也可以使用ANSI C形式來宣告

範例 9-3 ANSI C形式的參數宣告

```
// 定義 hello_world 模組
module hello_world #(parameter id_num = 0);   // ANSI C 形式的參數

initial  // 顯示模組的識別值
        $display("Displaying hello_world id number=%d", id_num);

endmodule
```

9.2 複寫參數

- 9.2.2 模組別名參數指定
  - 當模組取別名後，就可以複寫其參數，只要改寫範例 9-2 即可說明。新參數將傳送至模組別名中，最高層次模組(Top)可傳送參數至別名 w1、w2 如下。

```
// 定義最高層次模組
module top;
// 取二個 hello_world 模組的別名，傳送新的參數值
// 依照列表順序指定參數值
hello_world #(1) w1;     // 傳遞 1 到模組 w1

// 依照名稱指定參數值
hello_world #(.id_num(2)) w2;     // 傳遞 2 給模組 w2 中的 id_num參數

endmodule
```

9.2 複寫參數

- ## 9.2.2 模組別名參數指定
  - 若在一個模組中有多個參數，只要依照參數的順序指定，就可以在此模組中給定這些參數新的數值，若未指明覆蓋的數值，則使用預設的參數值，也可以用指定名稱方式來覆蓋參數數值。
  - **範例 9-4 模組別名參數數值**

```
// 定義包含延遲的模組
module bus_master;
parameter delay1 = 2;
parameter delay2 = 3;
parameter delay3 = 7;
…
// <模組內部>
…
endmodule
```

```
//最高層次模組，取二個 bus_master 模組的別名。
module top;
//取具有新的延遲數值的模組別名
//依照列表順序指定參數數值
bus_master #(4, 5, 6) b1(); //b1: delay1 = 4, delay2 = 5,
    delay3 = 6
bus_master #(9, 4) b2(); //b2: delay1 = 9, delay2 = 4, de-
    lay3 = 7(default)
//依照列表順序指定參數數值
bus_master #(.delay2(4), delay3(7)) b3(); // b2: delay2
                                          // =4,delay3=7
                                        // delay1=2(預設值)
// 建議使用指定名稱的方法設定參數數值，如此可以將錯誤的機會減到最
// 少，而且增加或減少參數的時候，可以不需擔心改變他們的順序。
endmodule
```

9.3 有條件的編譯與執行

- Verilog程式的部份可能只適用某些環境，因而需要設計兩種版本的程式，設計者可以定義這些部份的程式，在某些旗標被設定時才被編譯，這就是有條件的編譯。若是希望部份程式在某些旗標被設定時才被執行，這就是有條件的執行。
- **9.3.1 有條件的編譯**
  - 有條件的編譯是由編譯指令: ‘ifdef、‘ifndef、‘else、‘elsif和‘endif來完成。
  - ‘ifdef以及‘ifndef敘述可以放在設計的任一地方，設計者可以有條件地編譯敘述、模組、區塊、宣告和其他的編譯指令。
  - ‘else 伴隨著 ‘ifdef 或‘ifndef 選擇性的使用，最多一個‘ifdef或‘ifndef可以有一個‘else敘述。

9.3 有條件的編譯與執行

- 9.3.1 有條件的編譯
  - 範例 9-5 有條件的編譯
    - 在Verilog檔案中可使用'define敘述來設定條件編譯的旗標。

```
// 有條件的編譯
// 範例 1
`ifdef TEST // 如果文字巨集 TEST 被定義，編譯模組 test。
module test;
...
...
endmodule
`else    // 否則預設編譯模組 stimulus
module stimulus;
...
...
endmodule
`endif    // 完成 `ifdef 條件敘述
```

```
//範例 2
module top;

bus_master b1();    // 無條件取別名模組
`ifdef ADD_B2
  bus_master b2(); // 如果文字巨集ADD_B2 被定義，取別名模組b2。
`elsif ADD_B3
  bus_master b3(); // 如果文字巨集ADD_B3 被定義，取別名模組b3。
`else
   bus_master b4(); // 預設取別名模組 b4
`endif

`ifndef IGNORE_B5
   bus_master b5(); // 如果文字巨集 ADD_B3 未被定義，取別名
                    // 模組 b5。
`endif
endmodule
```

9.3 有條件的編譯與執行

- 9.3.2 有條件的執行
  - 條件執行旗標允許設計者在執行過程中，用來控制敘述的執行流程。
  - 條件執行旗標僅能使用在行為模型敘述，系統任務中用來處理條件執行的關鍵字為 $test$plusargs 。
  - **範例 9-6 使用$test$plusargs進行有條件的執行**

```
// 條件執行
module test;
...
...
initial
begin
    if($test$plusargs("DISPLAY_VAR"))
        $display("Display = %b", {a,b,c}); // 如果 flag 被
                                           // 設定則顯示
    else
        $display("No Display"); // 反之，則不顯示。
end
endmodule
```

如果執行時設定了DISPLAY_VAR旗標，則這些變數會顯示出來。設定方法是在執行時，指定執行選項 +DISPLAY_VAR 。

9.3 有條件的編譯與執行

- 9.3.2 有條件的執行
  - 範例 9-7 使用$value$plusargs的條件執行

```
// 使用$value$plusargs 的條件執行
module test;
reg [8*128-1:0] test_string;
integer clk_period;
…
…
initial
begin
    if($value$plusargs("testname=%s", test_string))
       $readmemh(test_string, vectors); // 讀取測試向量
```

9.3 有條件的編譯與執行

- 9.3.2 有條件的執行
  - 範例9-7 使用$value$plusargs的條件執行(續)

```
        else
            // 否則顯示錯誤訊息
            $display("Test name option not specified");

        if($value$plusargs("clk_t=%d", clk_period))
            forever #(clk_period/2) clk =~clk; // 設定時脈
        else
            // 否則顯示錯誤訊息
          $display("Clock period option name not specified");
end

//  在這個例子中，若要利用上述的選項，可以在執行模擬器的時候加上下
    列選項。
// +testname=test1.vec +clk_t=10
// 如此測試檔名 = "test1.vec" 而且 clk_period = 10
endmodule
```

9.4 時間刻度

- 模擬當中經常需要在不同模組中定義不同延遲數值的時間單位，如1us 與 100 ns等。
- Verilog 使用編譯指令 ‘timescale 來參照時間單位。
- 語法: ‘timescale <reference_time_unit>/<time_precision>
- <reference_time_unit> 是時間與延遲的單位大小
- <time_precision> 則指定其精確度，只有1、10、100是合法的整數值

9.4 時間刻度

- 範例9-8 時間刻度

```
// 定義模組 dummy1 的時間刻度
// 時間單位為 100 奈秒，精確度為 1 奈秒。
`timescale 100 ns /1 ns

module dummy1;

reg toggle;

// 設定 toggle 初始值
initial
    toggle = 1'b0;

// 每 5 個時間單位，反轉 toggle 暫存器變數值。
// 在模組中，每 5 個時間單位= 500 ns = .5us
always #5
    begin
        toggle =~ toggle ;
       $display("%d, In %m toggle = %b ", $time ,toggle);
    end

endmodule
```

9.4 時間刻度

- 範例9-8 時間刻度(續)

```
// 定義模組 dummy2 的時間刻度
// 時間單位為 1 微秒，精確度為 10 奈秒。
`timescale 1us/10ns

module dummy2;
reg toggle;

// 設定 toggle 初始值
initial
    toggle = 1'b0;
// 每 5 個時間單位，反轉 toggle 暫存器變數值。
// 在模組中，每 5 個時間單位 = 5us = 5000ns。
always #5
    begin
        toggle =~toggle;
        $display("%d, In %m toggle = %b ", $time, toggle);
    end

endmodule
```

9.4 時間刻度

- 範例9-8 時間刻度(續)
- 在dummy1與dummy2模組中，除了時間單位分別為100ns和1us外，其餘皆相同。
- 所以，dummy1中$display敘述執行十次，dummy2中$display敘述才執行一次。
- $time任務依據參照模組中之時間單位報告模擬時間。

```
// 模擬結果
 5, In dummy1 toggle = 1
10, In dummy1 toggle = 0
15, In dummy1 toggle = 1
20, In dummy1 toggle = 0
25, In dummy1 toggle = 1
30, In dummy1 toggle = 0
35, In dummy1 toggle = 1
40, In dummy1 toggle = 0
45, In dummy1 toggle = 1
 5, In dummy2 toggle = 1  <=========
50, In dummy1 toggle = 0
55, In dummy1 toggle = 1
```

9.5 有用的系統任務

- 9.5.1 檔案輸出
  - 系統任務$fopen可開啟一個檔案
  - 語法: $fopen(“<name_of_file>”);
  - 語法: <file_handle>=$fopen(“<name_of_file>”);
  - 任務$fopen將傳回一個32位元的值稱為多通道描述符號(multichannel descriptor)，在此符號中僅有一個位元會被設定。
  - 範例9-9 檔案描述符

```
// 多通道描述符號
integer handle1, handle2,handle3; // integers是32-bit值

// 標準輸出開啓時，descriptor = 32'h0000_0001 (bit0 set)。
initial
begin
    handle1 = $fopen("file1.out"); // handle1 = 32'h0000
              _0002 (bit 1 set)
    handle2 = $fopen("file2.out"); // handle2 = 32'h0000
              _0004 (bit 2 set)
    handle3 = $fopen("file3.out"); // handle3 = 32'h0000
              _0008 (bit 3 set)
end
```

9.5 有用的系統任務

- 9.5.1 檔案輸出
  - 多通道描述符號最大的用處為，可同時選擇輸出至多個檔案。
  - **寫至檔案(Writing to file)**
    - 系統任務$fdisplay、$fmonitor、$fwrite、$fstrobe皆用來寫至檔案。
  - 語法: $fdisplay(<file_descriptor, p1, p2, …, pn);

```
// 所有的handle定義於範例9-9　寫至檔案
 integer desc1, desc2, desc3;  // 三個多通道描述符號
initial
begin
   desc1 = handle1 | 1; // desc1 = 32'h0000_0003
   $display(desc1, "Display 1");    // 寫至檔案 file1.out 和標準輸出
   desc2 = handle2 | handle1; // desc2 = 32'h0000_0006
   $display(desc2, "Display 2");    // 寫至檔案 file1.out 和file2.out
   desc3 = handle3; // desc3 = 32'h0000_0008
   $display(desc3, "Display 3");    // 寫至檔案 file3.out
end
```

關閉檔案(Closing files)
- 語法: $fclose(handlel);

9.5 有用的系統任務

- 9.5.2 顯示出階層
  - 顯示任務$display、$write、$monitor與$strobe皆可用%m來顯示階層。
  - 範例9-10 顯示出階層

```
// 顯示出階層
module M;
...
initial
    $display("Displaying in %m");
endmodule

// 取別名模組 M
module top;
...
M m1();
M m2();
M m3();
endmodule
```

模擬結果

```
Displaying in top.m1
Displaying in top.m2
Displaying in top.m3
```

9.5 有用的系統任務

- 9.5.3 閃控(Strobing)
  - 其關鍵字為$strobe，非常類似$display，但當多個敘述和$display在同一時間執行時，其執行順序是不可知的。
  - 在正緣觸發後，先執行a=b 和 c=d，再顯示數值。若為$display，則可能較a=b 和 c=d 先執行，顯示的數值可能不同。
  - 範例9-11 閃控

```
// 閃控
always @(posedge clock)
begin
    a = b;
    c = d;
end

always @(posedge clock)
   $strobe("Displaying a=%b,c=%b",a,c); // 在正緣觸發顯示值
```

9.5 有用的系統任務

- **9.5.4 亂數產生器(Random Number Generation)**
  - 亂數產生器可以用來產生隨機測試向量(random test vectors)。亂數測試在要抓出設計中潛在的bug時非常重要，產生亂數向量也可用在分析晶片架構的效能。
  - 語法: $random; or $random(<seed>);
  - **範例9-12 亂數產生**

```
// 產生亂數並送至一個簡單的 ROM
module test;
integer r_seed;
reg [31:0] addr;// 輸入到 ROM
wire [31:0] data;// 從 ROM 輸出
…
…
ROM rom1(data, addr);

initial
    r_seed = 2; // 任意定義 seed 為 2

always @(posedge clock)
    addr = $random(r_seed); // 產生亂數
…
// <比對 ROM 的輸出與預期的結果>
…
…
endmodule
```

9.5 有用的系統任務

- 9.5.4 亂數產生器(Random Number Generation)
  - 亂數產生器能夠產生有號整數，因此，依照$random任務的不同用法，可以產生正整數或負整數。
  - 注意: 由於$random使用的演算法已經標準化，如果給定相同的seed值，在不同模擬器中執行會產生一致的亂數值。

- **範例9-13 利用$random任務產生正整數以及負整數**

```
reg [23:0] rand1, rand2;
rand1 = $random % 60;    // 產生介於-59 以及 59 的亂數
rand2 = {$random} % 60; // $random 加上結合運算元，可以用來產
                         // 生介於 0 到 59 的正整數。
```

9.5 有用的系統任務

- 9.5.5 從檔案來設定記憶體的初始值
  - Verilog提供兩個任務: $readmemb和$readmemh從檔案來設定記憶體的初始值，其分別為二進制與十六進制格式。
  - 範例9-14 設定記憶體的初始值

```
module test;
reg [7:0] memory [0:7]; // 宣告 8-byte 記憶體
integer i;
initial
begin
    // 讀取檔案 init.dat 至記憶體
    $readmemb("init.dat",memory);
    // 讀取記憶體的初始值
    for(i=0; i<8; i=i+1)
      $display("Memory [%0d] = %b", i, memory[i]);
end
endmodule
```

下面為一個簡單的int.dat的檔案

```
@002
11111111 01010101
00000000 10101010

@006
1111zzzz 00001111
```

模擬結果如下：

```
Memory [0] = xxxxxxxx
Memory [1] = xxxxxxxx
Memory [2] = 11111111
Memory [3] = 01010101
Memory [4] = 00000000
Memory [5] = 10101010
Memory [6] = 1111zzzz
Memory [7] = 00001111
```

9.5 有用的系統任務

- **9.5.6 數值變化轉儲檔案(Value Change Dump File)**
  - 數值變化轉儲檔案(VCD格式)是一個ASCII檔案，其內容包含:模擬時間、範圍(scope)、信號的定義與信號值的轉換。所有信號或一組被選擇的信號，在模擬時皆可被寫入VCD檔案中。
  - 後製處理器可以輸入VCD檔，並顯示階層資訊、信號值與波形。

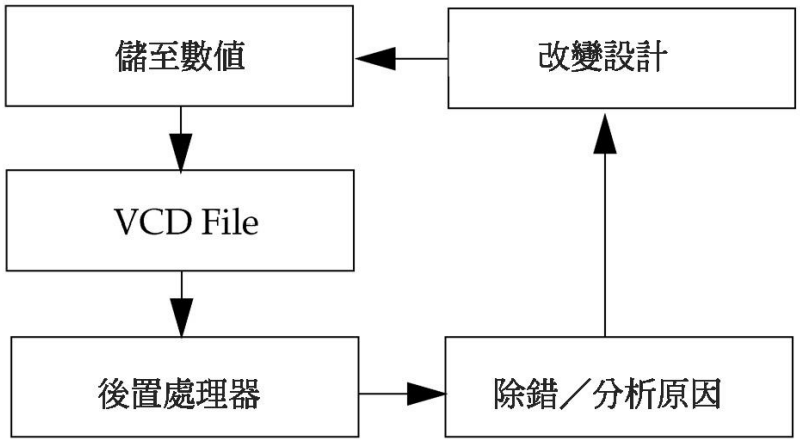

圖9.1　數值變化轉儲檔案除錯分析的流程

9.5 有用的系統任務

- **9.5.6 數值變化轉儲檔案(Value Change Dump File)**
  - 系統任務提供:選擇模組別名或模組別名信號被轉存($dumpvars)、VCD檔案的名字($dumpfile)、開始和結束轉儲處理($dumpop, $dumpoff)，和產生核對點($dumpall)。
  - 通常$dumpvars和$dumpfile放在開頭，其他三個控制轉儲處理。
  - **範例9-15 數值變化轉儲檔案的系統任務**

```
// 定義 VCD 檔案名稱，其他由模擬器內定名稱。
initial
    $dumpfile("myfile.dmp"); // Simulation info dumped to
    myfile.dmp

// 轉儲模組信號
initial
     $dumpvars; //若無引數，轉儲所有信號
initial
     $dumpvars(1,top); // 轉儲別名模組 top 的變數
```

9.5 有用的系統任務

- 9.5.6 數值變化轉儲檔案(Value Change Dump File)
  - 針對大的模擬，使用後置處理器的圖形顯示較為直接。
  - 通常VCD檔案有可能很龐大，所以要慎選部份的信號轉儲。
  - 範例9-15 數值變化轉儲檔案的系統任務(續)

```
                        // 數字代表階層層次，轉儲低於 top 一階層
                        // 層次的變數，也就是在 top 中之變數，
                        // 但不轉儲由 top 取別名之模組中的變數。
initial
    $dumpvars(2, top.m1); // 轉儲低於 top.m2 二階層層次的變數

initial
    $dumpvars(0, top.m1); // 數字零表示轉儲所有 top 階層的變數

// 啓始與結束轉儲
initial
begin
     $dumpon;            // 啓始轉儲
     #100000 $dumpoff; // 結束轉儲在 100,000 時間單位後
end

// 設置檢查點，轉儲所有目前 VCD 變數值。
initial
     $dumpall;
```