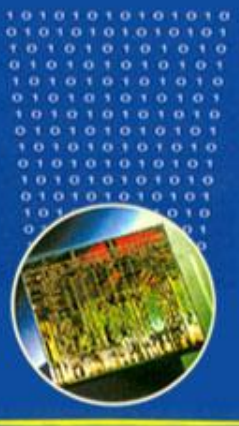
Verilog 硬體描述語言

Verilog HDL

A Guide to Digital Design and Synthesis



IEEE 1364-2001 Compliant

第8章 任務與函數

8.1 任務與函數的不同之處

- 在行為模式的設計中，通常程式中存在著許多具有相同描述的重複程式碼。在一般的程式語言裡，會將這些常用程式碼，寫成函數或是副程式的形式再去引用。
- 在Verilog中，則提供了任務(Task)與函數(Function)，讓相同程式碼可以在程式中被引用，省去一再撰寫相同的程式碼。
- 在任務當中，我們可以宣告input、output 與 inout，函數則可以宣告 input 作為輸出與輸入。

8.1 任務與函數的不同之處

表8-1　任務與函數

函數	任務
一個函數可以引用其他的函數，但不能引用其他的任務。	一個任務可以引用其他的任務與函數
函數永遠在時間等於零的時候開始執行	任務可以不從時間等於零的時候開始執行
函數不能包含有延遲、事件或是控制時間的任何陳述。	任務可以包含有延遲、事件與時間控制的陳述。
函數至少要有一個input的宣告，並能多於一個。	任務可以擁有零個或是更多的 input、output 或是 inout。
函數永遠回傳單一個值，並且不能有 output 與 inout 的宣告。	任務並沒有傳回的值，但可以藉由 output、inout 將值輸出來。

8.1 任務與函數的不同之處

- 任務與函數都必須在一個模組中定義，並僅屬於此模組。
- 任務通常用於取代一個具有延遲、時間或是事件的控制敘述，或是輸出的數目多於一個的Verilog程式碼。
- 函數則是應用在組合邏輯，執行時間為零與只有一個輸出的情況，因此函數常適用於轉換與計算。
- 任務與函數都可以有自己的區域變數、暫存器、時間變數、整數、實數或是事件，但不能有wire型態的變數。
- 任務與函數只能用於行為模式的敘述中，且本身不能有 initial 或 always 的敘述，但通常是在一個 initial 或 always 區塊中被引用。

8.2 任務(Tasks)

- 任務是以關鍵字task與endtask來宣告，適用的情況如下:
  - 需要用到延遲、時間或是事件控制指令。
  - 需要有零個或是多於一個輸出。
  - 沒有輸入。
- 8.2.1 任務宣告與引用
  - 一個任務的輸出與輸入訊號是用關鍵字 input、output 與 inout 來宣告。
  - input 與 inout 是將資料輸入到任務中，output 與 inout 則是將結果輸出到外面。
  - 與一般模組不同的是，任務當中的 input、output 與 inout 工作只是將訊號傳出或傳入一個任務，實際上並不是一個埠(port)。

8.2 任務(Tasks)

- 範例8-1 任務結構

```
// 宣告任務的語法
        task_declaration ::=
        task [automatic] task_identifier;
        { task_item_declaration }
        statement
        endtask
      | task[automatic] task_identifier(task_port_list);
        { block_item_declaration }
        statement
        endtask
task_item_declaration ::=
     block_item_declaration
   | { attribute_instance } tf_input_declaration;
   | { attribute_instance } tf_output_declaration;
   | { attribute_instance } tf_inout_declaration;
task_port_list ::= task_port_item { , task_port_item }
```

```
task_port_item ::=
     { attribute_instance } tf_input_declaration
   | { attribute_instance } tf_output_declaration
   | { attribute_instance } tf_inout_declaration
tf_input_declaration ::=
      input[reg][signed][range]list_of_port_identifiers
   |  input[task_port_type]list_of_port_identifiers
tf_output_declaration ::=
      output[reg][signed][range]list_of_port_identifiers
   |  output[task_port_type]list_of_port_identifiers
tf_inout_declaration ::=
      inout[reg][signed][range]list_of_port_identifiers
   |  inout[task_port_type]list_of_port_identifiers
task_port_type ::=
     time | real | realtime | integer
```

8.2 任務(Tasks)

- 8.2.2 兩個任務範例
  - 範例1: 運用 input 與 output
  - 範例8-2 任務中的input與output

```
// 定義一個名為 operation 的模組，並包含一個名為 bitwise_oper 的
   任務。
 module operation;
 ...
 ...
 parameter delay = 10 ;
reg[15:0]  A, B ;
reg[15:0]  AB_AND,  AB_OR,  AB_XOR ;

always  @(A or B)// 每當 A or B 改變其值
begin
    // 呼叫任務 bitwise_oper 並提供兩個數入引數 A、B
    // 提出三個輸出引數 AB_AND，AB_OR，AB_XOR
    // 引數的順序必需要跟定義任務時的順序相同
    bitwise_oper(AB_AND, AB_OR, AB_XOR, A, B);
end
```

```
 ...
 ...
// 定義任務 bitwise_oper
task bitwise_oper;
output [15:0] ab_and, ab_or, ab_xor; // 任務的輸出
input [15:0] a, b; // inputs to the task
begin
    #delay ab_and = a & b ;
    ab_or = a | b ;
    ab_xor = a ^ b ;
 end
 endtask
 ...
 endmodule
```

8.2 任務(Tasks)

- 8.2.2 兩個任務範例
  - 範例1: 運用 input 與 output
  - 範例8-3 利用ANSI C的慣用方式來定義任務

```
// 定義一個任務bitwise_oper
task bitwise_oper(output [15:0] ab_and, ab_or, ab_xor,
                  input [15:0] a, b);
begin
    #delay ab_and = a & b;
    ab_or = a | b;
    ab_xor = a ^ b;
end
endtask
```

8.2 任務(Tasks)

- 8.2.2 兩個任務範例
  - 範例2: 非對稱計時器訊號產生器
  - 範例8-4 直接作用於reg變數上的任務

```
// 定義一個包括任務 asymmetric_sequence 的模組
module sequence;
...
reg clock;
...
initial
    init_sequence; // 呼叫任務 init_sequence
...
always
begin
    asymmetric_sequence;  // 呼叫任務 asymmetric_sequence
end
...
...
```

```
// 初始值
task init_sequence;
begin
    clock = 1'b0;
end
endtask
// define task to generate asymmetric sequence
// operate directly on the clock defined in the module
task asymmetric_sequence;
begin
      #12 clock = 1'b0;
      #5  clock = 1'b1;
      #3  clock = 1'b0;
      #10 clock = 1'b1;
end
end task
...
endmodule
```

8.2 任務(Tasks)

- 8.2.3 自動任務
  - 傳統的任務使用方式是靜態配置，不論該任務被呼叫多少次，執行的都是同一段程式碼，使用同樣的記憶體空間。
  - 優點是當任務在多次呼叫中，對於需要保留的相關資訊，不用另外定義新的變數與佔用記憶體空間。
  - 缺點是如果同時間此任務被重複呼叫使用，將會造成變數值內容的錯亂。
  - 解決方案可以在宣告任務時，加一個關鍵字automatic，這樣在每次呼叫任務時，任務內定義的變數值將被重新初始化，程式也會在獨立的記憶體空間執行。

8.2 任務(Tasks)

- 範例8-5 自動任務

```
// 本範例中，只列出相關的任務使用部分。
// 本範例中有兩組時脈訊號，一組時脈 clk2 是操作於另一組時脈 clk 的
   兩倍速度。
module top;
reg [15:0] cd_xor, ef_xor; //variables in module top
reg [15:0] c, d, e, f; //variables in module top
-
task automatic bitwise_xor;
output [15:0] ab_xor; //output from the task
input [15:0] a, b; //inputs to the task
begin
    #delay ab_and = a & b;
    ab_or = a | b;
    ab_xor = a ^ b;
end
endtask
...
-
```

```
// These two always blocks will call the bitwise_xor task
// concurrently at each positive edge of clk. However,
   since
// the task is re-entrant, these concurrent calls will
   work correctly.
always @(posedge clk)
    bitwise_xor(ef_xor, e, f);
-
always @(posedge clk2) // twice the frequency as the
                          previous block
    bitwise_xor(cd_xor, c, d);
-
-
endmodule
```

8.3 函數(Functions)

- 函數是以關鍵字function與endfunction來宣告，適用的時機如下:
  - 程序中沒有延遲、時間或是事件控制指令。
  - 程序傳回單一個值時。
  - 至少有一個傳入參數時。
  - 沒有輸出或是輸出入埠。
  - 沒有阻礙程序。
- 在Verilog宣告一個函數時，同時也內定宣告了一個以函數名稱為名的暫存器，當函數執行完畢時，函數輸出會經由此暫存器傳回。

8.3 函數(Functions)

- 範例8-6 宣告函數的語法

```
function_declaration ::=
    function [ automatic ] [ signed ][ range_or_type ]
    function_identifier ;
    function_item_declaration{function_item_declaration}
    function_statement
    endfunction
  | function [ automatic ] [ signed ] [ range_or_type ]
    function_identifier(function_port_list);
    block_item_declaration { block_item_declaration }
    function_statement
    endfunction
function_item_declaration ::=
    block_item_declaration
  | tf_input_declaration ;
function_port_list ::= { attribute_instance } tf_input_
                       declaration {,{ attribute_instance
                       } tf_input_declaration }
range_or_type ::= range | integer | real | realtime | time
```

8.3 函數(Functions)

- 8.3.2 兩個任務範例
  - 範例8-7 奇偶計數器

```
// 定義一個包含函數 calc_parity 的模組
module parity;
...
reg[31:0]  addr;
reg parity;
// 當 address 值改變的時候，即重新計算奇偶性。
always @(addr)
begin
    parity = calc_parity(addr); // 第一次呼叫 calc_parity
                                函數
   $display("Parity calculated = %b", calc_parity(addr));
                          // 第二次呼叫 calc_parity 函數
end
...
...
```

```
// 定義計算奇偶性質的函數
function calc_parity;
input [31:0] address;
begin
    // 運用內定的暫存器 calc_parity 來輸出值
    calc_parity = ^address; // 利用簡化的互斥或運算計算出
                            // address 的奇偶性
end
endfunction
...
...
endmodule
```

8.3 函數(Functions)

- 8.3.2 兩個任務範例
  - 範例8-8 使用ANSI C的方式來定義函數

```
// 使用 ANSI C 的方式來定義函數
function calc_parity(input [31:0] address);
begin
    // 運用內定的暫存器 calc_parity 來輸出值
    calc_parity = ^address; // 回傳所有位元互斥運算後的值
end
endfunction
```

8.3 函數(Functions)

- 8.3.2 兩個任務範例
  - 範例8-9 左/右移位器

```
// 定義一個包含有函數 shift 的模組
module shifter;
...
// 左 / 右移位器
'define LEFT_SHIFT          1'b0
'define RIGHT_SHIFT         1'b1
reg [31:0]  addr, left_addr, right_addr ;
reg control;

// 當 addr 值變動時，即重新計算新 addr 的左 / 右移位後的值。
always @(addr)
begin
       // 呼叫 shift 函數計算左移與右移
       left_addr = shift(addr, 'LEFT_SHIFT);
       right_addr = shift(addr, 'RIGHT_SHIFT);
end
...
...
```

```
...
// 定義一個輸出為 32 位元的 shift 函數
function [31:0] shift;
input [31:0] address;
input control;
begin
    // 視 control 訊號作左移或是右移運算
   shift=(control== LEFT_SHIFT) ?(address<<1): address>>1;
end
endfunction
...
...
endmodule
```

8.3 函數(Functions)

- 8.3.3 自動遞迴函數
  - 範例8-10 利用遞迴(Automatic)函數

```
// 利用遞迴呼叫來作階乘
module top;
...
// 定義函數
function automatic integer factorial;
input [31:0] oper;
integer i;
begin
if(operand >= 2)
  factorial = factorial(oper -1)* oper; // 遞迴呼叫
else
  factorial = 1 ;
end
endfunction

// 呼叫函數
integer result;
initial
begin
    result = factorial(4); // 計算 4 的階乘
    $display("Factorial of 4 is %0d", result); // 答案是 24
end
...
...
endmodule
```

利用關鍵字automatic來定義一個自動(允許遞迴)函數，讓每次函數被呼叫時，會動態分配獨立的空間給函數，彼此互相不干擾。

8.3 函數(Functions)

- 8.3.4 常數函數
  - 一般用於複雜值的計算，或是取代常數的位置，使用上多有限制
- 範例8-11常數函數(計算記憶體位址寬度)

```
// 定義記憶體 1
module ram(...);
parameter RAM_DEPTH = 256;
input [clogb2(RAM_DEPTH)-1:0] addr_bus; // 計算記憶體所需
                                        // 定址線的數目
                              // clogb2 的結果是 8
                              // 形同 input [7:0] addr_bus;
--
--
// 常數函數
function integer clogb2(input integer depth);
begin
    for(clogb2=0; depth >0; clogb2=clogb2+1)
        depth = depth >> 1;
end
endfunction
--
--
endmodule
```

8.3 函數(Functions)

- 8.3.5 有號函數
  - 有號函數允許回傳的值可以經過有號數的運算
- 範例8-12 有號函數

```
module top;
--
// 定義有號函數
// 回傳一個 64 位元有號數
function signed [63:0] compute_signed(input [63:0] vec-
    tor);
--
--
endfunction
--
// 呼叫 compute_signed 有號函數
if(compute_signed(vector)< -3)
begin
--
end

--
endmodule
```

8.3 總結

- 任務與函數是用來取代在行為模式中常用到的功能，免除重複鍵入相同的程式碼。藉由任務與函數可將程式分成一個個有特定功能的區塊，使程式更加地容易解讀。
- 任務可擁有任意數目的 input、output 與 inout，並且可以有延遲、時間或是事件的控制敘述，另一個任務可以呼叫其他的任務或函數。
- 函數通常應用在至少有一個輸入，並且只有一個輸出的情況下，函數中不可有延遲、時間或是事件的控制敘述，一個函數只能呼叫其他的函數，不能呼叫任務。
- 任務與函數皆包含在Verilog的階層化架構中，並擁有階層化的名稱。