

Verilog

硬體描述語言

VerilogHDL

A Guide
to Digital
Design
and
Synthesis



IEEE
1364-2001
Compliant

第3章 使用Verilog 的基本概念

3.1 語法協定

3.2 資料型態

3.3 系統任務和編譯指令

3.4 總 結

3.5 習 題

3.1 語法協定(Lexical Conventions)

3.1

3.2

3.3

3.4

3.5

- Verilog的語法協定與C語言非常類似。
- Verilog是由一串的標記(token)組成，這些標記包含:註解(Comments)、數值(Numbers)、字串(Strings)、定義名稱(Identifiers)、和關鍵字(Keywords)。
- 標記之大小寫是不同的，所有的關鍵字皆由小寫所組成。
- **3.1.1 空白(Whitespace)**
 - 空白包含:空格(Blank spaces, \b)、欄位(Tabs, \t)、和換行(Newlines, \n)，除了分隔標記的空白外，其他地方使用的空白皆會被Verilog忽略。

3.1 語法協定(Lexical Conventions)

3.1

3.2

3.3

3.4

3.5

• 3.1.2 註解(Comments)

- 註解是為了使程式易讀(Readability)或文件化(Documentation)。
- 寫註解的兩種方法:以“//”開頭為單行註解，以“/*”開頭與“*/”結尾為多行註解，多行註解不允許有巢狀結構。

```
a = b && c; // This is a one-line comment.這是一個單行註解
```

```
/* This is a multiple line  
comment.這是一個多行註解 */
```

```
/* This is /* an illegal comment 這是一個錯誤的註解寫法 */
```

```
/* This is // a legal comment 這是一個合語法的註解寫法 */
```

3.1 語法協定(Lexical Conventions)

3.1

3.2

3.3

3.4

3.5

• 3.1.3 運算子(Operators)

- 運算子有三種形式: **一元**(Unary)、**二元**(Binary)和**三元**(Ternary)。
- 一元運算子放在運算元之前，二元運算子放在兩個運算元之間，三元運算子有兩個運算子來分隔三個運算元。

`a = ~b;` // ~是一元運算子，b 是運算元

`a = b && c;` // &&是二元運算子，b 和 c 是運算元

`a = b ? c : d;` // ?: 是三元運算子，b,c和d 是運算元

3.1 語法協定(Lexical Conventions)

3.1

3.2

3.3

3.4

3.5

•3.1.4 數字規格(Number Specification)

- Verilog有規定長度(Sized)，不定長度(Unsized)兩種數字規格。

規定長度之數字(Sized numbers)

- 規定長度之數字以<size>'<base format><number>來表示
- <size>以十進位來表示數字的位數(bits)，<base format>用以定義此數為十進位('d or 'D)、十六進位('h or 'H)、二進位('b or 'B)、八進位('o or 'O)，數字亦可用大寫表示。

```
a = 4'b1111; // 這是一個 4-bit 二進位數
```

```
a = 12'habc; // 這是一個 12-bit 十六進位數
```

```
A = 16'd255; // 這是一個 16-bit 十進位數
```

3.1 語法協定(Lexical Conventions)

3.1

3.2

3.3

3.4

3.5

•3.1.4 數字規格(Number Specification)

不定長度之數字(Unsigned numbers)

- 不定長度之數字不使用 **<size>** 來規定數字之位元數大小，而使用模擬器或硬體內定規格(必須大於32位元)。
- 若無 **<base format>** 則定義為十進制。

```
a = 23456; // 這是一個 32-bit 二進位數
```

```
a = 'hc3; // 這是一個 32-bit 十六進位數
```

```
A = 'o21; // 這是一個 32-bit 八進位數
```

3.1 語法協定(Lexical Conventions)

3.1

3.2

3.3

3.4

3.5

•3.1.4 數字規格(Number Specification)

x 或 z 值

- x 是代表不確定的值，z 是代表高阻抗。一個 x 在十六進制代表四位元的不確定值，八進制代表三位元的不確定值。
- 一個 z 在十六進制代表四位元的高阻抗，八進制代表三位元的高阻抗。
- 當最大位元數為 0、x、z 時，該數以其數值延伸至最高位元，若最大位元數為 1 時，該數以 0 延伸至最高位元。

```
a = 12'h13x; // 這是一個 12-bit 十六進位數; 最小四位元為不確定之值
```

```
a = 6'hx; // 這是一個 6-bit 十六進位數
```

```
A = 32'bz; // 這是一個 32-bit 高阻抗數
```

3.1 語法協定(Lexical Conventions)

3.1

3.2

3.3

3.4

3.5

•3.1.4 數字規格(Number Specification)

負數(Negative numbers)

- 將負號放在<size>之前即代表該數為負數。

```
a = -8'd3; // 用 8-bit 二補數來表示 -3
```

```
a = -6'sd3; // 用在有號整數(Signed Integer)的運算上
```

```
A = 4'd-2; // 不正確表示法
```

底線(Underscore characters)和問號(Question marks)

- 將負號放在<size>之前即代表該數為負數。

```
a = 12'b1111_0000_1010; // 與 12'b111100001010 相同
```

```
a = 4'b10??; // 與 4'b10zz 相同
```

3.1 語法協定(Lexical Conventions)

3.1

3.2

3.3

3.4

3.5

•3.1.5 字串(Strings)

- 字串之所有字元必須在同一行上。

```
a = "Hello Verilog World"; // 這是一個字串
```

```
a = "a/b"; // 這是一個字串
```

•3.1.6 定義名稱(Identifiers)與關鍵字(Keywords)

- **關鍵字**是一組特殊的定義名稱，其功用是為了定義程式語言架構。
- **定義名稱**是在程式語言中所給予物件的名稱，定義名稱可由**字母****數字**、**底線**與**錢號(\$)**所組成。定義名稱的大小寫是有分別的，不能以錢號做開頭。

```
reg value; // reg 是關鍵字 value 是定義名稱
```

```
input clock; // input 是關鍵字 clock 是定義名稱
```

3.2 資料型態

•3.2.1 數值組(Value Set)

- Verilog 提供四種數值和八種強度(Strengths)。
- 四種數值位準(Level)如表 3-1 所示

表3-1 數值位準

數值位準	實際電路狀態
0	邏輯 0，假(False)
1	邏輯 1，真(True)
x	不確定值(Unknown Value)
z	高阻抗(High Impedance)，懸接(Floating State)

3.1

3.2

3.3

3.4


3.5

3.2 資料型態

•3.2.1 數值組(Value Set)

- 數值位準為0、1有八種強度，如表 3-2 所示。
- 目的是為了解決在實際邏輯電路中，兩個不同強度的驅動源(Driver)衝突(Conflicts)的問題。

表3-2 強度位準

強度位準	型態	程度
supply	Driving	 最弱
strong	Driving	
pull	Driving	
large	Storage	
weak	Driving	
medium	Storage	
small	Storage	
highz	High Impedance	最強

如果strong1和weak0接在同一條線上，結果會是strong1。

如果strong1和strong0接在一起，結果會是x。

只有三態暫存接線(Trireg Nets)有 **large**、**medium**、**small**三種強度。

3.1

3.2

3.3

3.4

3.5

3.2 資料型態

• 3.2.2 接線(Nets)

- 接線是連接硬體元素之點。
- 在圖3-1的接線 a 是及閘g1的輸出，其值為接線b AND 接線c 的結果。
- 接線的最主要關鍵字為 **wire**，接線預設為一個位元，除非特別宣告成一個向量(vector)接線。
- 接線的預設值是 **z** (除了triereg接線之預設值是 x)，接線的值由它的驅動信號輸出而來。
- 若接線沒有驅動信號，則其值為 **x**。
- 接線關鍵字還有: wire, wand, wor, tri, trior, triereg...

```
wire a;           //宣告右邊電路中一個接線 a
wire b, c;        // 宣告右邊電路中接線 b, c
wire d = 1'b0;    // 在宣告時設定接線 d 為一固定值 0
```

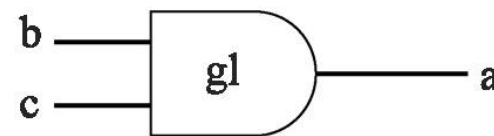


圖3-1 接線範例

3.1

3.2

3.3

3.4

3.5

3.2 資料型態

3.1

3.2

3.3

3.4

3.5

•3.2.3 暫存器(Registers)

- 暫存器用來表示資料儲存的元素，除非給定新數值，否則暫存器內的數值會一直維持。
- Verilog中的暫存器就是一個可以持留(Hold)數值的變數。
- 暫存器不用像接線一樣需要驅動，Verilog暫存器也不需要時脈。
- 在模擬過程中，可以在任意時間指定新的數值給暫存器。
- 暫存器的關鍵字是 **reg**，其預設值是 **x**。

```
reg reset;    // 宣告一個變數來持留數值
initial      // 這一個架構將於後面章節討論
begin
    reset = 1'b1;    // 設定reset之初始值為 1 來重設電路
    #100 reset = 1'b0; // 在 100 個模擬時間後，設定 reset 為 0
end
```

3.2 資料型態

3.1

3.2

3.3

3.4

3.5

•3.2.3 暫存器(Registers)

- reg 可以宣告一個有號數，用來做有號數的算數運算使用。

```
reg signed [63:0] m; //宣告一個 64 位元的有號數
integer i;          //一個 32 位元的有號整數
```

•3.2.4 向量(Vectors)

- 接線與暫存器皆可定義為向量，若無定義位元長度，則以一個位元(純量)來計。
- 向量可定義為[high#:low#] or [low#:high#]，但是中括號左邊為最大位元。

```
wire a; // 定義為純量暫存器變數
wire [7:0] bus; // 8-bit 匯流排
wire [31:0] busA, busB, busC; // 3 個 32-bit 寬度的匯流排
reg clock; // 定義為純量暫存器變數
reg [0:40] virtual_addr; // 向量暫存器變數，41 bits 寬度的虛擬位址
```

3.2 資料型態

• 3.2.4 向量(Vectors)

• 向量子集合選取

```
busA[7]      // 匯流排 A 之第七個位元
bus[2:0]      // 匯流排之最末三個位元，若使用 bus[0:2] 是不合法的
virtual_addr[0:1] // 虛位址之最高二個位元
```

• 固定寬度向量子集合選取

```
reg [255:0] data1;      // Little endian notation
reg [0:255] data2;      // Big endian notation
reg [7:0] byte;
// 利用上面宣告好的向量變數，來做固定寬度向量子集合選取
byte = data1[31 -: 8]; // 從第31位元開始遞減選取8位元 data1[31:24]
byte = data1[24 +: 8]; // 從第24位元開始遞增選取8位元 data1[31:24]
byte = data2[31 -: 8]; // 從第31位元開始遞減選取8位元 data2[31:24]
byte = data2[24 +: 8]; // 從第24位元開始遞增選取8位元 data2[31:24]
for (j=0; j<=31; j=j+1)
    byte = data1[(j*8)+:8]; // 選取到的結果如同 [7:0], [15:8], ...
data1[(byteNum*8)+:8] = 8'b0; // 如果byteNum=1, 清除[15:8]的8位元
```

3.1

3.2

3.3

3.4

3.5

3.2 資料型態

3.1

3.2

3.3

3.4

3.5

• 3.2.5 整數、實數、和時間暫存資料型態

• 整數

- 整數以關鍵字 **integer** 做宣告，雖然可以用 **reg** 做一般變數宣告，但整數變數以 **integer** 做宣告是較便利的，如計數(counting)。
- 整數是以主機的內定長度為長度，但最少要大於32位元。**reg**是宣告為**無號(Unsigned)**數，**integer**是宣告為**有號(Signed)**數。

```
integer counter;      // 一般用途的變數
initial
  counter = -1;       // 將 -1 存在 counter 中
```

• 實數

- 實數以關鍵字 **real** 做宣告，其可以是科學記號(例如，3e6表示 3×10^6)，或者十進位數(例如，3.14)，內定值為 0，當實數指定給一整數時，會取最接近該數之整數值。

3.2 資料型態

3.1

3.2

3.3

3.4

3.5

• 3.2.5 整數、實數、和時間暫存資料型態

• 實數

```
real delta;          // 一般用途的變數
initial begin
    delta = 4e10;      // delta 被指定一科學表示式
    delta = 2.13       // delta 被指定一數值 2.13
end
integer i;           // 宣告一整數
initial
    i = delta; // i 的數值為 2 (rounded value of 2.13)
```

• 時間

- 時間是以關鍵字 **time** 做宣告，功用是儲存模擬時間，最少要為 **64-bits** 的資料。**\$time** 是系統函數，功能在取得目前的模擬時間。

```
time save_sim_time;    // 定義時間變數 save_sim_time
initial
    save_sim_time = $time; // 儲存目前的模擬時間
```

3.2 資料型態

• 3.2.6 陣列

- 陣列之內容可以是整數、暫存器、時間、實數和向量，陣列的維度不限。陣列的表示法為 `<array_name>[<subscript>]`，不論使用一維陣列或是多維陣列，每一個維度大小都必須是指定好的常數。

```
integer count[0:7]; // 8 個整數組成的陣列
reg bool[31:0];      // 32 個一位元布林暫存器變數組成的陣列
time chk_point[1:100]; // 100 個時間變數組成的陣列
reg [4:0] port_id[0:7]; // 8 個 5 位元組成的陣列
integer matrix[4:0][0:255]; // 二維陣列
reg [63:0] array_4d [15:0][7:0][7:0][255:0]; // 四維陣列
wire [7:0] w_array2 [5:0]; // 六個八位元接線組成的陣列
wire w_array1[7:0][5:0]; // 一個 48 個元素組成的二維陣列，每一元素是一個接線
```

3.1

3.2

3.3

3.4

3.5

3.2 資料型態

• 3.2.6 陣列

- 向量與陣列是不相同的，向量是一個多位元的元素，陣列是多個一位元或多個多位元的元素。

```
count[5] = 0; // 清除 count 陣列中第五個元素
chk_point[100] = 0; // 清除 chk_point 陣列中第100個元素
port_id[3] = 0; // 清除 port_id 陣列中第三個元素(內含 5 個位元)

matrix[1][0] = 33559; // 設定 matrix 陣列中[1][0]的元素為 33559
array_4d[0][0][0][0][15:0] = 0; // 清除四維度的 array_4d 陣列中
// [0][0][0][0]元素的16位元[15:0]

port_id = 0; // 錯誤的用法，意圖清除整個陣列
matrix [1] = 0; // 錯誤的用法，意圖將二維陣列中，[1][0] ~ [1][255]
// 的元素都設定為零
```

3.1

3.2

3.3

3.4

3.5

3.2 資料型態

3.1

3.2

3.3

3.4

3.5

• 3.2.7 記憶體(Memories)

- 在Verilog中，記憶體是一個暫存器的陣列，陣列中的每一個物件就像是個字元(word)，每個字元可以是一個位元或多個位元。
- n 個一位元暫存器和一個 n 位元的暫存器是不同的。
- 陣列中的註標(subscript)像是記憶體中的位址，可以指定我們想要的字元。

```
reg mem1bit[0:1023];      // 記憶體 mem1bit 是 1k 個一位元的字元
reg [7:0] membyte[0:1023]; // 記憶體 membyte 是 1k 個八位元的字元

membyte[511] // 提取位址為 511 的一位元組(byte)的字元
```

3.2 資料型態

3.1

3.2

3.3

3.4

3.5

• 3.2.8 參數(Parameters)

- 在Verilog的模組中可利用 **參數** 關鍵字 **parameter** 來定義一個固定常數。參數不能當作 **變數** 使用，但可以在 **初始化模組** 時重新定義。
- 參數型態或是位元數都可以在 **宣告** 時設定。
- 利用 **defparam** 可以在模組宣告時改變模組內的參數值。
- 在Verilog 2001標準中，新增了 **localparam** 關鍵字，此關鍵字所定義的參數，將不能透過 **defparam** 的方式來修改。

```
parameter port_id = 5;          // 宣告一常數 port_id，其值為 5
parameter cache_line_width = 256;
// 宣告一常數 cache_line_width，其值為 256
parameter signed [15:0] WIDTH;
// 宣告一參數 WIDTH 為一個十六位元有號數

localparam state1 = 4'b0001, state2 = 4'b0010,
              state3 = 4'b0100, state4 = 4'b1000;
```

3.2 資料型態

• 3.2.9 字串(Strings)

- 字串可以指定給**暫存器(reg)**，若字串長度大於暫存器長度，則左邊的位元會被**移除**，若字串長度小於暫存器長度，則左邊的位元會被**補零**。

```
reg [8*18:1] string_value;    // 宣告一變數 有18 位元組的寬度
initial
string_value = "Hello Verilog World"; // 將字串存於變數中
```

表3-3 特殊字元

特殊字元	顯示字元
\n	換行
\t	跳欄
%%	%
\\	\
\"	"
\ooo	1-3 個八位元數

3.1

3.2

3.3

3.4

3.5

3.3 系統任務和編譯指令

3.1

3.2

3.3

3.4

3.5

• 3.3.1 系統任務(System tasks)

- 系統任務包含了顯示、監視數值、模擬暫停與工作完成等等，皆以 `$<keyword>` 的格式來表示。

資訊顯示

- `$display` 是最有用的系統任務，主要用來顯示變數值或是字串內容。
- 格式： `$display(p1,p2,p3,...,pn);`
- 參數 `p1,p2,p3,...,pn` 可以是字串、變數或說明，類似於C語言的 `printf` 功能。只是執行每一個 `$display` 指令後會自動換行，若之後無接任何參數，就會輸出一個空行。

3.3 系統任務和編譯指令

• 3.3.1 系統任務(System tasks)

資訊顯示

表3-4 格式定義表

格式	顯示
%d or %D	十進制變數
%b or %B	二進制變數
%s or %S	字串
%h or %H	十六進制變數
%c or %C	ASCII 字元
%m or %M	階層名(不需引數)
%v or %V	強度
%o or %O	八進制變數
%t or %T	目前時間
%e or %E	實數科學記號
%f or %F	十進制實數變數
%g or %G	選擇實數科學記號和十進制實數變數較短者

3.1

3.2

3.3

3.4

3.5

3.3 系統任務和編譯指令

3.1

3.2

3.3

3.4

3.5

• 3.3.1 系統任務(System tasks)

資訊顯示

• 範例3-3 顯示任務

```
// 顯示雙引號中之字串
%display("Hello Verilog World");
--Hello Verilog World

// 顯示目前時間
%display($time);
--230

// 顯示虛擬位址為 1fe0000001c 的值和目前時間 200
reg [0:40] virtual;
$display("At time %d virtual address is %h", $time, virtual_adder);
--At time 200 virtual address is 1fe0000001c
```

3.3 系統任務和編譯指令

3.1

3.2

3.3

3.4

3.5

• 3.3.1 系統任務(System tasks)

資訊顯示

• 範例3-3 顯示任務

```
// 使用二進制顯示 port_id 其值為 5
reg [4:0] port_id;
    $display("ID of the port is %b", port_id);
-- ID of the port is 00101

// 顯示 x 特性
// 使用二進制顯示四位元匯流排其值為 10xx
reg [3:0] bus;
$display("Bus value is %b",bus);
--Bus value is 10xx

// 顯示較高層次模組 top 的別名 p1 的階層名，不需任何引數。
$display("This string is displayed from %m level of hier-
    archy")
--This string is displayed from top.p1 level of hierarchy
```

3.3 系統任務和編譯指令

3.1

3.2

3.3

3.4

3.5

• 3.3.1 系統任務(System tasks)

資訊監視

- Verilog提供一個監控訊號變化的系統任務，只要監控訊號有變化，便會輸出新的值，這一個系統任務便是\$monitor。
- 格式：\$monitor(p1,p2,p3,...,pn);
- 參數 p1,p2,p3,...,pn 可以是字串、變數或說明，但是\$monitor 不僅僅是顯示一次而已，而是每一次信號變化都顯示一次。
- 每一次僅允許有一個\$monitor 可以執行，若有多次\$monitor 的使用，只有最後一個有效。
- 不同的\$monitor 工作可以利用 \$monitoron 與 \$monitoroff 做切換
- 格式：\$monitoron;
\$monitoroff;

3.3 系統任務和編譯指令

3.1

3.2

3.3

3.4

3.5

• 3.3.1 系統任務(System tasks)

資訊監視

• 範例3-5 監視敘述

```
// 監視信號 clock 與 reset 的值和時間
// Clock 每 5 時間單位觸發一次而 reset 在 10 時間單位轉為低電位
initial
begin
    $monitor($time, "Value of signals clock = %b reset = %
    b, click, reset);
end
```

Partial output of the monitor statement:

```
-- 0 Value of signals clock = 0 reset = 1
-- 5 Value of signals clock = 1 reset = 1
-- 10 Value of signals clock = 0 reset = 0
```

3.3 系統任務和編譯指令

3.1

3.2

3.3

3.4

3.5

• 3.3.1 系統任務(System tasks)

中止(Stopping)和完成(Finishing)模擬

- 格式: `$stop;` 可以暫時中止模擬運算的進行，並進入交談模式，設計者可以利用交談模式來檢驗當時信號的值是否正確。
- `$stop`可以讓設計者在任一個想停止的時刻來做觀察除錯的工作。
- 格式: `$finish;` 結束模擬運算
- 範例3-6 中止和完成模擬運算

```
// 中止模擬在 100 時間單位並檢驗結果
// 在 1000 時間單位完成模擬
initial // 模擬時間為 0
begin
  clock = 0;
  reset = 1;
  #100 $stop; // 中止模擬在 100 時間單位
  #900 $finish; // 在 1000 時間單位完成模擬
end
```

3.3 系統任務和編譯指令

3.1

3.2

3.3

3.4

3.5

• 3.3.2 編譯指令(Compiler Directives)

- Verilog 一樣有編譯指令，編譯指令皆以 '**<keyword>**' 來表示，此處只兩種最常用的語法。

`define

- 用來定義文字巨集(text macro)，類似C語言的**#define**
- 每次編譯Verilog檔案時，編譯器會將定義好的文字巨集，搜尋代換有使用的程式片段。

• 範例3-7 `define的用法

```
// 用文字巨集定義字的寬度
// 在程式碼中使用`WORD_SIZE
`define WORD_SIZE 32

// 定義一個別名，當`s出現時用$stop取代
`define s $stop;

// 定義一個經常使用的文字字串
`define WORD_REG reg [31:0]
// 然後可以用`WORD_REG reg32; 定義一個 32-bit 暫存器變數
```

3.3 系統任務和編譯指令

3.1

3.2

3.3

3.4

3.5

• 3.3.2 編譯指令(Compiler Directives)

`include

- `include可以帶入其他檔案的內容，插入到include使用的位置，類似C語言的**#include**

• 範例 3-8 `include的用法

```
//包含 header.v 檔案，其包含一些在主程式 design.v 檔案中之宣告。  
`include header.v  
...  
...  
<Verilog code in file design.v>  
...  
...
```