

SystemVerilog

語法學習

目錄

1. Data_Type
2. Control_Flow
3. Processes
4. Communication
5. Interface
6. Constraints
7. Functional Coverage

Chapter 1

Data_Type

資料型態總類

- SystemVerilog 有分成 2-state 跟 4-state 的資料型態
 - 2-state: 指資料只有 0 或 1
 - 4-state: 指資料分成 0、1、Z(高阻抗)、X(don't care)

資料型態列表

| | Data-type | 2-state/4-state | # Bits | signed/unsigned | C equivalent | |
|---------------------------------|-----------|-----------------|----------|-----------------|--------------|---------------------------------|
| S Y S T E M | reg | 4 | ≥ 1 | unsigned | | V E R I L O G |
| | wire | 4 | ≥ 1 | unsigned | | |
| | integer | 4 | 32 | signed | | |
| | real | | | | double | |
| | time | | | | | |
| | realtime | | | | double | |
| V E R I L O G | logic | 4 | ≥ 1 | unsigned | | |
| | bit | 2 | ≥ 1 | unsigned | | |
| | byte | 2 | 8 | signed | char | |
| | shortint | 2 | 16 | signed | short int | |
| | int | 2 | 32 | signed | int | |
| | longint | 2 | 64 | signed | long int | |
| | shortreal | | | | float | |

Logic (4-State) (1/3)

- 它是一個四狀態型別（0、1、X、Z）
- 可以在程序區塊（如initial、always）和連續賦值（如assign）中都被驅動
- Logic 不能被使用在多個驅動源（drivers），
需使用**網路型別（net-type）**，例如wire，來代替。
因為SystemVerilog需要透過**強度解析（strength resolution）**
來決定該信號的最終值

Logic (4-State) (2/3)

- 連續賦值（如assign）
 - `en = 1`，因為`my_data[0] = 1`

```
module tb;
  logic [3:0] my_data;
  logic en;
  assign en = my_data[0]; // 連續賦值給logic型別

  initial begin
    my_data = 4'h3; // 二進位 0011
    #10;
    $display("en = %b", en);
  end
endmodule
```

Logic (4-State) (3/3)

- 多重驅動（使用 wire 示範）
 - strong強度的1勝過weak強度的0，所以signal = 1。

```
module tb;
  wire signal;

  assign (strong1, weak0) signal = 1'b1; // 第一個驅動源，強度為strong
  assign (weak1, weak0) signal = 1'b0;  // 第二個驅動源，強度為weak

  initial begin
    #10;
    $display("signal = %b", signal);
  end
endmodule
```


Bit, Byte, Int (2-State) (1/2)

- 它們是一個二狀態型別（0、1）
- 它們減少了模擬期間的記憶體使用量，因為每個位元只需要一位存儲
- 由於記憶體開銷較低且處理狀態的複雜性降低，因此模擬效能更快。

| Data Type | Size | Signed Range | Unsigned Range | Description |
|-----------|--------|-------------------------|-----------------|--|
| bit | 1-bit | | 0 and 1 | 2-state variable |
| byte | 8-bit | -2^7 to 2^7-1 | 0 to 2^8-1 | 2-state variable, similar to C char |
| shortint | 16-bit | -2^{15} to $2^{15}-1$ | 0 to $2^{16}-1$ | 2-state variable, similar to C short int |
| int | 32-bit | -2^{31} to $2^{31}-1$ | 0 to $2^{32}-1$ | 2-state variable, similar to C int |
| longint | 64-bit | -2^{63} to $2^{63}-1$ | 0 to $2^{64}-1$ | 2-state variable, similar to C long int |

Bit, Byte, Int (2-State) (2/2)

- 宣告方式 示範

```
module tb;
  bit          var_a;    // Declare a 1 bit variable of type "bit"
  bit [3:0]    var_b;    // Declare a 4 bit variable of type "bit"
  byte         m_var_byte;
  shortint     m_var_shortint;
  int          m_var_int;
  longint      m_var_longint;

  byte unsigned u_byte; // Byte is set to unsigned
  shortint unsigned u_var_shortint;

  initial begin
    var_b = 4'h3; // 二進位 0011
    $display("var_b=0x%0h ", var_b);
  end
endmodule
```

String (1/3)

- 宣告方式 示範

```
module tb;  
  // Declare a string variable called "dialog" to store string literals  
  // Initialize the variable to "Hello!"  
  string  dialog = "Hello!";  
  
  initial begin  
    // Display the string using %s string format  
    $display ("%s", dialog);  
  
    // Iterate through the string variable to identify individual characters and print  
    foreach (dialog[i]) begin  
      $display ("%s", dialog[i]);  
    end  
  end  
endmodule
```

模擬結果:
Hello!
H
e
l
l
o
!

String Operators (2/3)

| | Operator | Semantics |
|---------------|--|---|
| Equality | Str1 == Str2 | Returns 1 if the two strings are equal and 0 if they are not |
| Inequality | Str1 != Str2 | Returns 1 if the two strings are not equal and 0 if they are |
| Comparison | Str1 < Str2 Str1 <= Str2 Str1 > Str2 Str1 >= Str2 | Returns 1 if the correspondig condition is true and 0 if false |
| Concatenation | {Str1, Str2, ..., StrN} | All strings will be concatenated into one resultant string |
| Replication | {multiplier{Str}} | Replicates the string N number of times, where N is specified by the multiplier |
| Indexing | Str[index] | Returns a byte, the ASCII code at the given index. If given index is out of range, it returns 0 |
| Methods | Str.method([args]) | The dot(.) operator is used to call string functions |

String Methods (3/3)

| | | |
|----------------|-----------------------------------|--|
| str.atoi() | function integer atoi(); | Returns the integer corresponding to the ASCII decimal representation in str |
| str.atohex() | function integer atohex(); | Interprets the string as hexadecimal |
| str.atooct() | function integer atooct(); | Interprets the string as octal |
| str.atobin() | function integer atobin(); | Interprets the string as binary |
| str.atoreal() | function real atoreal(); | Returns the real number corresponding to the ASCII decimal representation in str |
| str.itoa(i) | function void itoa (integer i); | Stores the ASCII decimal representation of i into str |
| str.hextoa(i) | function void hextoa (integer i); | Stores the ASCII hexadecimal representation of i into str |
| str.octtoa(i) | function void octtoa (integer i); | Stores the ASCII octal representation of i into str |
| str.bintoa(i) | function void bintoa (integer i); | Stores the ASCII binary representation of i into str |
| str.realtoa(r) | function void realtoa (real r); | Stores the ASCII real representation of r into str |

| Usage | Definition | Comments |
|-------------------|--|--|
| str.len() | function int len() | Returns the number of characters in the string |
| str.putc() | function void putc (int i, byte c); | Replaces the i th character in the string with the given character |
| str.getc() | function byte getc (int i); | Returns the ASCII code of the i th character in str |
| str.tolower() | function string tolower(); | Returns a string with characters in str converted to lowercase |
| str.compare(s) | function int compare (string s); | Compares str and s, as in the ANSI C strcmp function |
| str.icompare(s) | function int icompare (string s); | Compares str and s, like the ANSI C strcmp function |
| str.substr (i, j) | function string substr (int i, int j); | Returns a new string that is a substring formed by characters in position i through j of str |

Enumeration (1/1)

- Enum 範例

```
module tb;
    // "e_true_false" is a new data-type with two valid values: TRUE and FALSE
    typedef enum {TRUE, FALSE} e_true_false;

    initial begin
        // Declare a variable of type "e_true_false" that can store TRUE or FALSE
        e_true_false answer;

        // Assign TRUE/FALSE to the enumerated variable
        answer = TRUE;

        // Display string value of the variable
        $display ("answer = %s", answer.name);
    end
endmodule
```

Array (1/8)

- 分成

1. Packed array
2. Unpacked array

```
module tb;
    bit [7:0]          m_data;          // A vector or 1D packed array
    bit [3:0][7:0]     m_data0;         // multidimensional packed array, 4 bytes
    bit [2:0][3:0][7:0] m_data1;        // multidimensional packed array, 12 bytes

    bit               m_mem [10];       // Unpacked
    byte              stack0 [2][4];     // Unpacked 2 rows, 4 cols

    bit [3:0][7:0]     stack1 [2][4];   // packed + unpacked array.
endmodule
```

Array (2/8)

```
module tb;
    bit [3:0][7:0]    stack [2][4];           // 2 rows, 4 cols

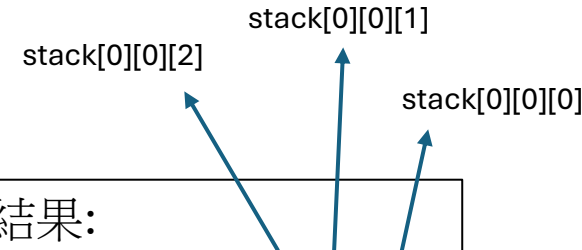
    initial begin
        // Assign random values to each slot of the stack
        foreach (stack[i])
            foreach (stack[i][j]) begin
                stack[i][j] = $random;
                $display ("stack[%0d][%0d] = 0x%0h", i, j, stack[i][j]);
            end

        // Print contents of the stack
        $display ("stack = %p", stack);

        // Print content of a given index
        $display("stack[0][0][2] = 0x%0h", stack[0][0][2]);
    end
endmodule
```

模擬結果:

stack[0][0] = 0x12153524
stack[0][1] = 0xc0895e81
stack[0][2] = 0x8484d609
stack[0][3] = 0xb1f05663
stack[1][0] = 0x6b97b0d
stack[1][1] = 0x46df998d
stack[1][2] = 0xb2c28465
stack[1][3] = 0x89375212
stack = '{'h12153524,
'hc0895e81, 'h8484d609,
'hb1f05663}, {'h6b97b0d,
'h46df998d, 'hb2c28465,
'h89375212}'
stack[0][0][2] = 0x15



Dynamic Array (3/8)

- Dynamic Array 類似 C++ 的 new (動態記憶體配置)
- 可以在run time時，create出array的大小 or 新增大小

```
module tb;
    // Create a dynamic array that can hold elements of type int
    int      array [];

    initial begin
        // Create a size for the dynamic array -> size here is 5
        // so that it can hold 5 values
        array = new [5];

        // Initialize the array with five values
        array = '{31, 67, 10, 4, 99};
    end
endmodule
```

| Function | Description |
|--------------------------|--|
| function int size (); | Returns the current size of the array, 0 if array has not been created |
| function void delete (); | Empties the array resulting in a zero-sized array |

Dynamic Array (4/8)

```
module tb;
    // Create two dynamic arrays of type int
    int array [];
    int id [];

    initial begin
        // Allocate 5 memory locations to "array" and initialize with values
        array = new [5];
        array = '{1, 2, 3, 4, 5};

        // Point "id" to "array"
        id = array;

        // Display contents of "id"
        $display ("id = %p", id);

        // Grow size by 1 and copy existing elements to the new dyn.Array "id"
        id = new [id.size() + 1] (id);

        // Assign value 6 to the newly added location [index 5]
        id [id.size() - 1] = 6;

        // Display contents of new "id"
        $display ("New id = %p", id);

        // Display size of both arrays
        $display ("array.size() = %0d, id.size() = %0d", array.size(), id.size());
    end
endmodule
```

模擬結果

id = '{1, 2, 3, 4, 5}'

New id = '{1, 2, 3, 4, 5, 6}'

array.size() = 5

, id.size() = 6

Associative array(6/8)

| Function | Description |
|---|--|
| function int num (); | Returns the number of entries in the associative array |
| function int size (); | Also returns the number of entries, if empty 0 is returned |
| function void delete ([input index]); | <i>index</i> when specified deletes the entry at that index, else the whole array is deleted |
| function int exists (input index); | Checks whether an element exists at specified index; returns 1 if it does, else 0 |
| function int first (ref index); | Assigns to the given index variable the value of the first index; returns 0 for empty array |
| function int last (ref index); | Assigns to given index variable the value of the last index; returns 0 for empty array |
| function int next (ref index); | Finds the smallest index whose value is greater than the given index |
| function int prev (ref index); | Finds the largest index whose value is smaller than the given index |

Array Manipulation Methods(7/8)

| Method name | Description |
|--------------------|--|
| find() | Returns all elements satisfying the given expression |
| find_index() | Returns the indices of all elements satisfying the given expression |
| find_first() | Returns the first element satisfying the given expression |
| find_first_index() | Returns the index of the first element satisfying the given expression |
| find_last() | Returns the last element satisfying the given expression |
| find_last_index() | Returns the index of the last element satisfying the given expression |

| Methods | Description |
|----------------|--|
| min() | Returns the element with minimum value or whose expression evaluates to a minimum |
| max() | Returns the element with maximum value or whose expression evaluates to a maximum |
| unique() | Returns all elements with unique values or whose expression evaluates to a unique value |
| unique_index() | Returns the indices of all elements with unique values or whose expression evaluates to a unique value |

| Method | Description |
|-----------|--|
| reverse() | Reverses the order of elements in the array |
| sort() | Sorts the array in ascending order, optionally using <code>with</code> clause |
| rsort() | Sorts the array in descending order, optionally using <code>with</code> clause |
| shuffle() | Randomizes the order of the elements in the array. <code>with</code> clause is not allowed here. |

Array Manipulation Methods(8/8)

- 以上 Method 可以搭配 with()，去篩選出想要的結果

```
module tb;
  int array[9] = {4, 7, 2, 5, 7, 1, 6, 3, 1};
  int res[$];

  initial begin
    res = array.find(x) with (x > 3);
    $display ("find(x)      : %p", res);

    res = array.find_index with (item == 4);
    $display ("find_index   : res[%0d] = 4", res[0]);

    res = array.find_first with (item < 5 & item >= 3);
    $display ("find_first    : %p", res);

    res = array.find_first_index(x) with (x > 5);
    $display ("find_first_index: %p", res);

    res = array.find_last with (item <= 7 & item > 3);
    $display ("find_last     : %p", res);

    res = array.find_last_index(x) with (x < 3);
    $display ("find_last_index : %p", res);
  end
endmodule
```

模擬結果:

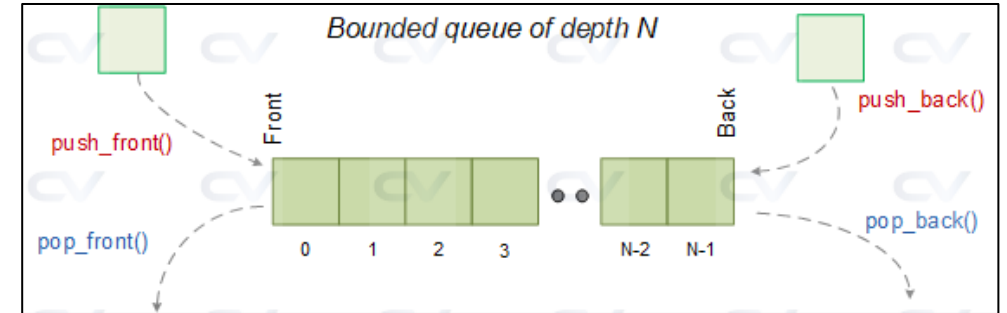
```
find(x)      : {4, 7, 5, 7, 6}
find_index   : res[0] = 4
find_first    : {4}
find_first_index: {1}
find_last     : {6}
find_last_index: {8}
```

Queue (1/1)

- 使用 [\$] 來宣告成queue

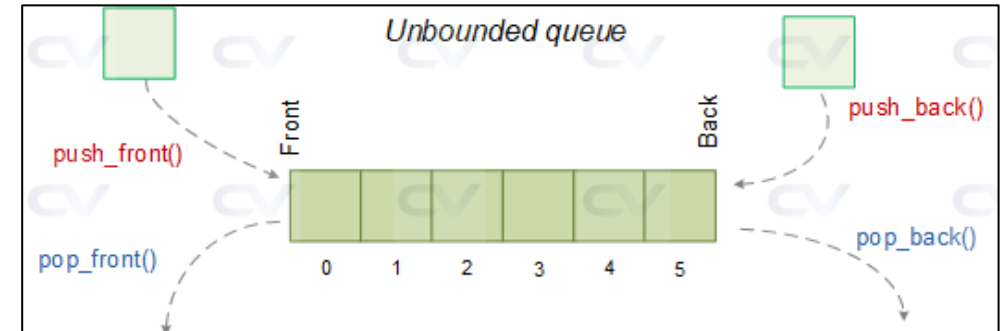
- Bounded queue

- [data_type] [name_of_queue] [\$:N];
 - int bounded_queue [\$:10]; // Depth 10



- Unbounded queue

- [data_type] [name_of_queue] [\$];
 - int unbounded_queue [\$]; // Unlimited entries



Structure (1/1)

- 分成 packed & unpacked struct
 - packed struct
 - 使用 packed 去宣告 struct，預設情況下它是無號的
 - 資料以連續的位元方式儲存，沒有記憶體間隙(padding)
 - 所有成員緊密排列，類似於一個連續的位向量(bit vector)
 - 適合用於硬體設計中需要精確控制位元對應的場景，例如通訊協定封包或暫存器映射。
 - Packed Struct：適合硬體設計，記憶體連續，支援位元操作，合成效率高，但限制於位元型別。
 - Unpacked Struct：適合測試平台或軟體風格的資料結構，靈活但可能有記憶體間隙，合成效率較低。

Chapter 2

Control_Flow

Types of looping constructs (1/1)

| | |
|----------|--|
| forever | Runs the given set of statements forever |
| repeat | Repeats the given set of statements for a given number of times |
| while | Repeats the given set of statements as long as given condition is true |
| for | Similar to while loop, but more condensed and popular form |
| do while | Repeats the given set of statements at least once, and then loops as long as condition is true |
| foreach | Used mainly to iterate through all elements in an array |

Types of if-else statement (1/2)

- 分成三種 if-else
 1. unique-if
 - 如果設計要求條件互斥且必須有一個分支執行
 2. unique0-if
 - 如果設計要求條件互斥但允許無匹配的情況
 3. priority-if
 - 如果設計需要明確的優先級（條件可能重疊）

Types of if-else statement (2/2)

| 特性 | unique-if | unique0-if | priority-if |
|---------|-------------------------|------------|-------------------------|
| 條件互斥性 | 必須互斥 | 必須互斥 | 不要求互斥 |
| 多條件同時為真 | 報錯（違規） | 報錯（違規） | 按優先級執行第一個為真的分支 |
| 無條件為真 | 允許（執行 else 或無動作） | 允許（無動作） | 允許（執行 else 或無動作） |
| 優先級 | 無優先級（假設互斥） | 無優先級（假設互斥） | 有明確優先級（從上到下） |
| 典型應用 | 解碼器、互斥控制邏輯 | 靈活的互斥邏輯 | 優先級編碼器、仲裁器 |
| 綜合優化 | 假設互斥，可能生成更小電路 | 假設互斥，允許無匹配 | 生成優先級邏輯，可能更大電路 |

Blocking & Non-Blocking assignment statement (1/2)

- 賦值分為兩類
 1. 阻塞賦值 (Blocking Assignment)
 - 阻塞賦值是立即執行的，當前語句完成賦值後，才會執行下一條語句。
 2. 非阻塞賦值 (Non-Blocking Assignment)
 - 非阻塞賦值是延遲執行的，賦值操作在當前模擬時間步（time step）的調度階段完成。

Blocking & Non-Blocking assignment statement (2/2)

| 特性 | 阻塞賦值 (=) | 非阻塞賦值 (<=) |
|------|---------------------------|--------------------|
| 符號 | = | <= |
| 執行時機 | 立即執行 | 延遲到模擬時間步結束 |
| 模擬行為 | 順序執行，立即更新變數 | 並行執行，所有更新同時發生 |
| 變數影響 | 後續語句使用更新後的值 | 後續語句使用舊值 |
| 典型應用 | 組合邏輯 (always_comb) | 時序邏輯 (always_ff) |
| 硬體對應 | 模擬連線或組合邏輯 | 模擬寄存器或觸發器 |
| 潛在風險 | 在時序邏輯中可能導致 race condition | 在組合邏輯中可能導致不必要延遲 |

Function & Task (1/2)

- Task
 - 可以消耗時間，使用input, output, inout 來傳遞參數
 - 不能使用@(posedge)和@(negedge)。
 - 沒有return值
- Function
 - Function 不可以消耗時間，所以不能有以下時間控制statements
 - @, #, fork join, or wait
 - Function不能呼叫 task，因為 task 可以消耗時間

Function & Task (2/2)

| 特性 | Function | Task |
|-------|---------------------------------------|---|
| 模擬時間 | 不消耗模擬時間（立即執行） | 可以消耗模擬時間（支持延遲、等待） |
| 返回值 | 可以有返回值（默認為 void ） | 無返回值（僅通過 output/inout 傳遞結果） |
| 語法結構 | 使用 function 和 endfunction | 使用 task 和 endtask |
| 時序控制 | 不能包含時序控制語句（如 #delay, @(posedge clk) ） | 可以包含時序控制語句 (delay、wait、@) (不能使用 posedge、negedge) |
| 賦值類型 | 使用阻塞賦值 (=) | 可以使用阻塞(=) 或非阻塞(<=) 賦值 |
| 輸入/輸出 | 支持 input、output、inout 參數 | 支持 input、output、inout 參數 |
| 調用場景 | 用於組合邏輯計算（例如數學運算、邏輯處理） | 用於時序邏輯或行為模擬（例如測試平台、協議實現） |
| 執行範圍 | 必須在單個模擬時間步內完成 | 可以跨多個模擬時間步執行 |

Chapter 3

Processes

fork (1/2)

- fork 分成三種

1. fork join

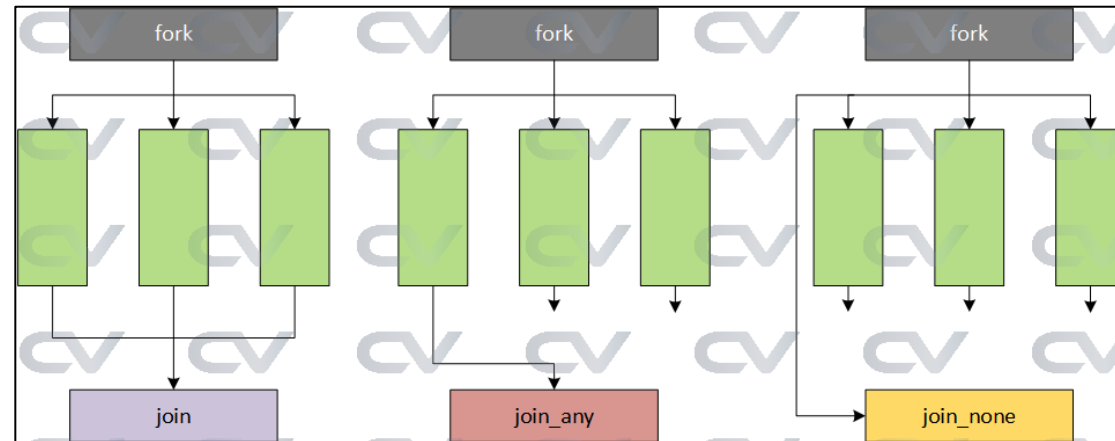
- 需要所有 thread 完成後才會離開 fork 區塊往下執行

2. fork join_any

- 只要有其中一個 thread 完成，就會離開 fork 區塊往下執行

3. fork join_none

- 不管 thread 是否完成，都直接離開fork區塊往下執行



fork (2/2)

- fork 兩種後續指令
 1. disable fork
 - 關閉 fork 內所有 thread
 2. wait fork
 - 等待所有 thread 完成後，再繼續執行

Chapter 4

Communication

Communication (1/6)

- SV 分成三種 Communication

| | |
|------------|--|
| Events | Different threads synchronize with each other via event handles in a testbench |
| Semaphores | Different threads might need to access the same resource; they take turns by using a semaphore |
| Mailbox | Threads/Components need to exchange data with each other; data is put in a mailbox and sent |

Event (2/6)

- Create event
 - `event eventA; // Creates an event called "eventA"`
- Trigger event
 - `-> eventA;`
- Wait for event to happen
 - `@eventA; // Use "@" operator to wait for an event`
 - `wait (eventA.triggered); // Or use the wait statement with "eventA.triggered"`
 - 使用 `.triggered` 可以避免 race condition
- `wait_order()`
 - 用來判斷多個 event 有沒有照順序trigger
 - Ex: `wait_order(a, b) // 觸發順序 -> a -> b`

Semaphore (3/6)

- Create semaphore
 - semaphore key;
key = new (1); // Argument to new () defines the number of keys.
- Use semaphore
 - get ()
 - get the key by using get (), the keyword which will wait until a key is available
 - put()
 - put the key back using the put () keyword

Semaphore (4/6)

- Semaphore Methods

| Name | Description |
|--|--|
| function new (int keyCount = 0); | Specifies number of keys initially allocated to the semaphore bucket |
| function void put (int keyCount = 1); | Specifies the number of keys being returned to the semaphore |
| task get (int keyCount = 1); | Specifies the number of keys to obtain from the semaphore |
| function int try_get (int keyCount = 1); | Specifies the required number of keys to obtain from the semaphore |

Mailbox (5/6)

- Create mailbox
 - mailbox mbx;
- Use mailbox
 - mbx.put (trns); // 放入物件
 - mbx.get (trns); // 取出物件

Mailbox (6/6)

- Mailbox Methods

| Function | Description |
|--|--|
| function new (int bound = 0); | Returns a mailbox handle, bound > 0 represents size of mailbox queue |
| function int num (); | Returns the number of messages currently in the mailbox |
| task put (singular message); | Blocking method that stores a message in the mailbox in FIFO order; message is any singular expression |
| function int try_put (singular message); | Non-blocking method that stores a message if the mailbox is not full, returns a postive integer if successful else 0 |
| task get (ref singular message); | Blocking method until it can retrieve one message from the mailbox, if empty blocks the process |
| function int try_get (ref singular message); | Non-blocking method which tries to get one message from the mailbox, returns 0 if empty |
| task peek (ref singular message); | Copies one message from the mailbox without removing the message from the mailbox queue. |
| function int try_peek (ref singular message); | Tries to copy one message from the mailbox without removing the message from queue |

Chapter 5

Interface

Interface (1/3)

- **Interface**是一種將訊號封裝到區塊中的方法
- 所有相關訊號被組合在一起形成一個**Interface**區塊
- **Interface** 可以重複用於其他項目。
- 且與 **DUT** 和其他驗證組件的連接也變得更加容易

```
// 以下是一個 APB bus protocol 的 interface
interface apb_if (input pclk);
    logic [31:0] paddr; // address 變數
    logic [31:0] pwrdata; // write data 傳輸變數
    logic [31:0] prdata; // read data 傳輸變數
    logic penable;
    logic pwrite;
    logic psel;
endinterface
```

modport (2/3)

- Modport 可以用來定義 interface 內的訊號，用在不同 module 上時的輸入輸出關係

```
interface myBus (input clk);  
    logic [7:0] data;  
    logic    enable;  
  
    // 從 TestBench 來看, 'data' 是 input and 'write' 是 output  
    modport TB (input data, clk, output enable);  
  
    // 從 DUT 來看, 'data' 是 output and 'enable' 是 input  
    modport DUT (output data, input enable, clk);  
endinterface
```

Clocking block (3/3)

- Clocking block 可以用來降低 module 跟 module 連接時，所產生的訊號取樣 race condition

```
interface my_int (input bit clk);  
    // Rest of interface code  
  
    clocking cb_clk @(posedge clk);  
        // 定義 input 提前 3ns 取樣, output 延後 2 ns 取樣  
        default input #3ns output #2ns;  
        input enable;  
        output data;  
    endclocking  
endinterface
```

Chapter 6

Constraints

Random 變數 (1/1)

- rand
 - 會隨機產生數值
- randc
 - 會隨機產生數值，且在一週期內數值不重複，等所有可能都出現過後，才會結束週期

```
class Packet;  
    randc bit [2:0] data;  
endclass  
  
module tb;  
    initial begin  
        Packet pkt = new ();  
        for (int i = 0 ; i < 10; i++) begin  
            pkt.randomize ();  
            $display ("itr=%0d data=0x%0h", i, pkt.data);  
        end  
    end  
endmodule
```

```
// 執行結果  
itr=0 data=0x6  
itr=1 data=0x3  
itr=2 data=0x4  
itr=3 data=0x7  
itr=4 data=0x0  
itr=5 data=0x1  
itr=6 data=0x5  
itr=7 data=0x2 // 一個週期結  
束，以上數字不重複  
itr=8 data=0x5  
itr=9 data=0x0
```


Constraint blocks (1/1)

- 可以針對 random 變數做限制
- 在做randomize() 時，變數會依照 Constraint blocks 內容隨機化出符合規定的數值

```
class ABC;
    rand bit [3:0] mode;

    // 建立一個 constrain block 來限制 mode 的數值範圍，  $2 < mode \leq 6$ 
    constraint c_mode { mode > 2;
                        mode <= 6;
    };
endclass
```

Constraint Operator (1/5)

1. inside operator
2. weighted distributions
3. 在 constraint block 常見的用法
 1. `->`
 2. `if else`
 3. `foreach`
 4. `solve before`
4. 直接在 `randomize()` 後面加上 `with()` 來做限制

Constraint Operator (2/5)

1. inside operator

```
constraint my_range { typ > 32;  
                    typ < 256; }  
  
// typ >= 32 and typ <= 256  
constraint new_range { typ inside {[32:256]}; }  
  
// 選擇 32 or 64 or 128  
constraint spec_range { type inside {32, 64, 128}; }
```

Constraint Operator (3/5)

2. weighted distributions (:=)

```
class myClass;  
  rand bit [2:0] typ;  
  
  // 權重分配，使用 :=  
  // 0 有 20  
  // 1~5 各有 50  
  // 6 有 40  
  // 7 有 10  
  // 總共 320  
  // 產生 0 的機率為 20/320  
  constraint dist1 { typ dist { 0:=20, [1:5]:=50, 6:=40, 7:=10}; }  
endclass
```

Constraint Operator (4/5)

3. 在 constraint block 常見的用法

1. ->

```
// 當 mode == 2 時，len > 10  
constraint c_mode { mode == 2 -> len > 10; }  
  
// 意思跟上面一樣  
constraint c_mode { if (mode == 2)  
                    len > 10;  
}
```

Constraint Operator (5/5)

3. 在 constriant block 常見的用法

4. solve before

```
class ABC;
  rand bit  a;
  rand bit [1:0] b;

  // a == 0 時，b 的可能值為 0 ~ 3 這個部分的機率為 1/2 * 1/4
  // a == 1 時，b 的可能值為 3 這個部分的機率為 1/2
  constraint c_ab { a -> b == 3'h3;

                      // 告訴 solver
                      // 先決定 a，在依照 constraint 去決定 b
                      solve a before b;
  }

endclass
```

Static constraint (1/2)

- constraint 可以宣告成 static 型態
- 當 constraint 為 static，則此 constraint 為個物件共用的 constraint
- 有任一物件使用 `.constraint_mode();` 去開關 constraint 時，其他物件也會開關此 static constraint

Static constraint (2/2)

Ex:

```
class ABC;  
  rand bit [3:0] a;  
  
  // "c1" is non-static, but "c2" is static  
  constraint c1 { a > 5; }  
  static constraint c2 { a < 12; }  
endclass
```

```
module tb;  
  initial begin  
    ABC obj1 = new;  
    ABC obj2 = new;  
  
    // Turn non-static constraint  
    // 當 obj1 去關閉 c2 constraint，則 obj2 的 c2 也會被關閉  
    obj1.c2.constraint_mode(0);  
  
    for (int i = 0; i < 5; i++) begin  
      obj1.randomize();  
      obj2.randomize();  
      $display ("obj1.a = %0d, obj2.a = %0d", obj1.a, obj2.a);  
    end  
  end  
endmodule
```


pre & post randomize (1/2)

1. pre_randomize()

- 物件呼叫 randomize() 後，先執行pre_randomize() 在去做 random 動作

2. post_randomize()

- 物件呼叫 randomize() 後，randomize() 執行成功後，執行 post_randomize()
- 如果 randomize() 沒成功，不執行 post_randomize()

pre & post randomize (2/2)

Ex:

```
class Beverage;
  rand bit [7:0] beer_id;

  constraint c_beer_id { beer_id >= 10;
                        beer_id <= 50; };

  function void pre_randomize ();
    $display ("This will be called just before randomization");
  endfunction

  function void post_randomize ();
    $display ("This will be called just after randomization");
  endfunction

endclass
```

```
module tb;
  Beverage b;

  initial begin
    b = new ();
    $display ("Initial beerId = %0d", b.beer_id);
    if (b.randomize ())
      $display ("Randomization successful !");
    $display ("After randomization beerId = %0d", b.beer_id);
  end
endmodule
```

Soft Constraints (1/1)

- `soft constraint` 是一種 弱約束條件
- 如果該變數沒有其他會與`soft constraint`牴觸的限制，`soft constraint` 會生效
- 如果有其他強制 `constraint`、直接指定值或使用 `.randomize() with {}`，那麼 `soft constraint` 就會被忽略。

Disable Constraints (1/2)

- `constraint_mode()` 可以用來關閉 **constraint**
- `.constraint_mode(0)` -> 關閉
- `.constraint_mode(1)` -> 開啟
- `.constraint_mode()` => 不帶任何參數，
回傳目前 `constraint` 狀態 0 -> disable, 1 -> enable
- 常搭配 `pre_randomize()` 來決定是否開啟 `constraint`

Disable Constraints (2/2)

Ex:

```
class Fruits;  
  rand bit[3:0] num;  
  
  constraint c_num { num > 4;  
                    num < 9; };  
  
endclass
```

```
module tb;  
  initial begin  
    Fruits f = new ();  
    $display ("Before randomization num = %0d", f.num);  
  
    // Disable constraint  
    f.c_num.constraint_mode(0);  
  
    if (f.c_num.constraint_mode ())  
      $display ("Constraint c_num is enabled");  
    else  
      $display ("Constraint c_num is disabled");  
  
    // Randomize the variable and display  
    f.randomize ();  
    $display ("After randomization num = %0d", f.num);  
  end  
endmodule
```

Disable Randomization (1/1)

- `rand_mode()` 用來關閉某個參數的隨機化

```
class Fruits;  
  rand bit [3:0] var1;  
  rand bit [1:0] var2;  
endclass
```

```
module tb;  
  initial begin  
    Fruits f = new();  
    $display ("Before randomization var1=%0d var2=%0d", f.var1,  
f.var2);  
  
    // Turn off randomization for var1  
    f.var1.rand_mode (0);  
  
    // Print if var1 has randomization enabled/disabled  
    if (f.var1.rand_mode())  
      $display ("Randomization of var1 enabled");  
    else  
      $display ("Randomization of var1 disabled");  
  
    f.randomize();  
  
    $display ("After randomization var1=%0d var2=%0d", f.var1, f.var2);  
  end  
endmodule
```

Random weighted case (1/1)

- 可以自己創造自己想要的 case，並給予他們權重

```
module tb;
  initial begin
    for (int i = 0; i < 10; i++)
      // 此 case 分母為 1+5+3 = 9
      // 出現 0 的機率為 1/9
      // 出現 5 的機率為 5/9
      // 出現 3 的機率為 3/9
      randcase
        0: $display ("Wt 1");
        5: $display ("Wt 5");
        3: $display ("Wt 3");
      endcase
    end
  endmodule
```

Chapter 7

Functional Coverage