Verilog

目錄

1. 簡介

- Verilog 是一種高階且模組化的硬體描述語言，其基本特點如下：
  - 功能強大：已達電路描述、合成與模擬驗證垂直整合設計功能。
  - 彈性設計 (Flexibility)：可模組化，易做重新組合，使設計更快速，維護更容易。
  - 多種描述風格：如連線關係，順序性與共時性敘述，布林代數式等。
  - 可攜性 (Portability)：是工業標準，可用不同編譯軟體編譯去分別適用不同的工作平台與製程。
  - 容易學習：語法與 C 語言相似。

2. Verilog 的模型

- Verilog 的四大模型 (model)：
  - 開關階層 (switch level) 或電晶體 (transistor) 模型。
  - 邏輯閘階層 (gate level) 模型。
  - 資料流 (data flow) 或暫存器轉移階層 (register transfer level) 模型。
  - 行爲 (behavioral) 模型。

2.1 邏輯閘階層模型

- 在這個階層中，電路模組是由最基本的邏輯閘所連接形成的。

```
// 2-input AND gate
module and2(in1, in2, out);
input  in1, in2;
output out;

       and u1(out, in1, in2);

endmodule
```

2.2 資料流模型

- 在這個階層中，電路的設計重點在於說明資料如何在電路中的傳送過程。

```
// 2-input AND gate
module and2(in1, in2, out);
input  in1, in2;
output out;

       assign out = in1 & in2;

endmodule
```

2.3 行為模型

- 這個階層是Verilog HDL中的最高階層。
- 在這個階層中，我們只需考慮電路模組的功能，而不需考慮其硬體的詳細內容。

```
// 2-input AND gate
module and2(in1, in2, out);
input  in1, in2;
output out;
reg    out;
       always@(in1 or in2)
       begin
              out = in1 & in2;
       end
endmodule
```

3. Verilog 的模組

- Verilog 中的模組 (module) 是組成一個電路的基本單位，它描述了
  - 引用模組的檔案名稱。
  - 模組的名稱。
  - 所用到的輸出入埠名稱、個數與大小。
  - 電路所需的接線與暫存器。
  - 引用之較低階的模組別名。
  - 電路所需功能的指定敘述 (assign)。
  - 電路所需功能的行爲層級的描述。
  - 函數 (functions) 與任務 (task)。

3.1 模組架構

編譯程式指引 // 'include & 'define

module module_name(port list);

Port 的宣告 // input, output, inout

變數資料型態宣告 // wire, reg, …

引用較低階的模組

邏輯閘階層之描述

資料流階層之描述

function 或 task 的宣告

行為階層之描述區塊

endmodule

3.2 電路檔案結構

- 如 C 語言的函數一般，Verilog 的模組中不能再有其他的模組存在。
- 在一個 Verilog 檔案中，可以同時存在多個模組。
- 模組宣告的順序可以是任意的。
- 模組名稱的命名規則與一般識別字相同。

3.3 Verilog 語法協定

- Verilog 語言的語法單元 (token) 包括：
  - 空白 (whitespace)
  - 註解 (comment)
  - 關鍵字 (keyword)
  - 識別字 (identifier)
  - 運算子 (operator)
  - 數字 (number)
  - …

3.3 Verilog 語法協定

- 註解
  - Verilog 所提供的「註解」格式有
    - 單行註解
      - 使用「//」作爲開始符號。
      - 結束符號爲換行符號 (end_of_line)。
    - 多行註解
      - 使用「/*」作爲開始符號。
      - 使用「*/」作爲結束符號。

3.3 Verilog 語法協定

- 關鍵字
  - 所有的關鍵字必須使用英文小寫字母來表示。
  - 常見的關鍵字有

```
always    negedge   posedge
begin     end       assign    wire
integer   function  endfunction
module    endmodule
for       if        else
inout     input     output
and       buf       nand      nor
not       or        xnor      xor
```

3.3 Verilog 語法協定

- 在 Verilog 電路描述中，識別字可用於定義變數名稱、函數名稱、模組名稱與物件實例 (instance) 名稱。
- 識別字的命名規則：
  - 第一個字元必須是英文字母。
  - 第二個之後的字元可以是英文字母、數字、底線 (_)、或是錢字號 ($)。
  - 識別字的長度沒有限制。
  - 識別字有區分英文大小寫。

3.3 Verilog 語法協定

- 運算子
  - Binary bit-wise operators: ~, &, |, ^, ~^, ^~
  - Unary reduction operators: &, ~&, |, ~|, ^, ~^, ^~
  - Logical operators: !, &&, ||
  - 2's complement operators: +, -, *, /, %
  - Relational operators: >, <, >=, <=, ==, !=, ===, !==
  - Logical shift operators: >>, <<
  - Conditional operators: ? :
  - Duplication operators: {n{ <exp> <,<exp>> *}}
  - Concatenation operators: {}

3.3 Verilog 語法協定

- 數字
  - 固定長度的數字
    - 語法：`<size>'<base><number>`
    - <size>：表所使用的 bit 數，十進位表示法
    - <base>：可以是 B、O、D、H
    - 範例：`1'B0, 4'O7, 8'HF, 10'D9`
  - 不定長度的數字
    - 不使用 `<size>` 來指定 bit 數目，而是使用 HDL 編譯程式的內定值。
    - 範例：`'h89ab, 128, 'O377, 'b1001`

4. MAX+plus II 的環境

- MAX+plus II 之 Verilog 電路設計與模擬

- 利用文字編輯器建立新的 Verilog 檔案。
  - 副檔名為 v。
- 指定專案名稱。
- 輸入 Verilog 電路模組。
- 儲存檔案。
- 建立內定符號。
- 執行功能編譯。

- 建立新的波形檔案。
  - 副檔名為 scf。
- 選取欲觀察的輸入、輸出接點。
- 定義輸入接點的波形。
- 儲存波形檔案。
- 啟動電路模擬功能。
- 檢視模擬結果的波形。

4. MAX+plus II 的環境

- MAX+plus II 的限制
  - 每個 Verilog 檔案只能有一個 module 的宣告。
  - Module 名稱需與檔案名稱相同。
  - 不支援 task。
  - 並非每個在 IEEE 標準中的資料型態、運算子、運算式、邏輯閘等均有支援，而部分功能也只是有限制的支援。
  - 建議：使用 MAX+plus II 撰寫 Verilog 程式時，可利用其 on-line help 來了解所支援的部分。

5. 基本資料型態

- Verilog 的基本資料型態：
  - wire：代表一條接線
  - wand：Wired-AND
    - MAX+plus II 不支援
  - wor：Wired-OR
    - MAX+plus II 不支援
  - reg：暫存器
    - 主要功能爲用於保持住電路中的某個值。
  - integer, real, …：用於計算的過程。
    - MAX+plus II 僅支援 integer，且只能用於 for 敘述。

5.1 四種數值準位

- Verilog 所提供的四種數值準位 (value level)：
  - 0：
    - 邏輯 0, Zero, False, Low, Logic Low, Ground, V_{SS}, Negative Assertion
  - 1：
    - 邏輯 1, One, True, High, Logic High, Power, V_{DD}, V_{CC}, Positive Assertion
  - X：
    - Unknown value
  - Z：
    - High impedance, Floating state, Tri-state, Disable driver

5.2 wire 的特性

- 資料型態爲 wire 的變數爲連接硬體元件之連接線。
- 變數必須被驅動，才能改變它的內容。
- 除非被宣告爲一向量，否則 wire 型態的變數內定爲一個位元的值，且其內定值爲 Z。

```
wire  a, b, c;

assign c = a & b;
```

5.3 reg 的特性

- 宣告爲資料型態 reg 之變數的功能和一般程式語言中的變數類似，可以直接給定一個數值。
- 除非被宣告爲一向量，否則 reg 型態的變數內定爲一個位元的值，且其值爲 X。

```
reg   a, b, c;

always@(…)
      c = a & b;
```

5.4 使用 wire 或 reg 的時機

- 使用 wire 所宣告的變數必須配合 assign 敘述來改變其值，且不能在 always 區塊中作爲敘述的左值 (*l*-value) 。
- 使用 reg 所宣告的變數必須使用在 always 區塊中作爲敘述的左值。

5.5 向量的宣告方式

- 向量 (vector) 的宣告方式
  - wire 與 reg 均可用於定義向量。
  - 向量是一個多位元的元件。

```
wire   [0:3] in;
wire   [0:3] out;
wire   [3:0] a, b;

reg    [0:3] c;
reg    [3:0] d, e;
```

5.6 陣列表示法

- 陣列表示法
  - 陣列的內容可以是 wire、reg、integer 或向量。
  - 陣列是多個 1 位元或多位元的元件集合。
  - 編譯程式只能使用一維陣列的表示法。

```
wire   in[0:100];
reg    out[0:100];
```

  - MAX+plus II 沒有支援。

5.7 位元選擇與部分選擇

- 位元選擇
  - 選擇向量表示法中，變數的某個位元。
- 部分選擇
  - 選擇向量表示法中，變數的某一些連續的位元。

```
wire  [7:0] a, b, c;

assign c[0] = a[0] & b[0];
```

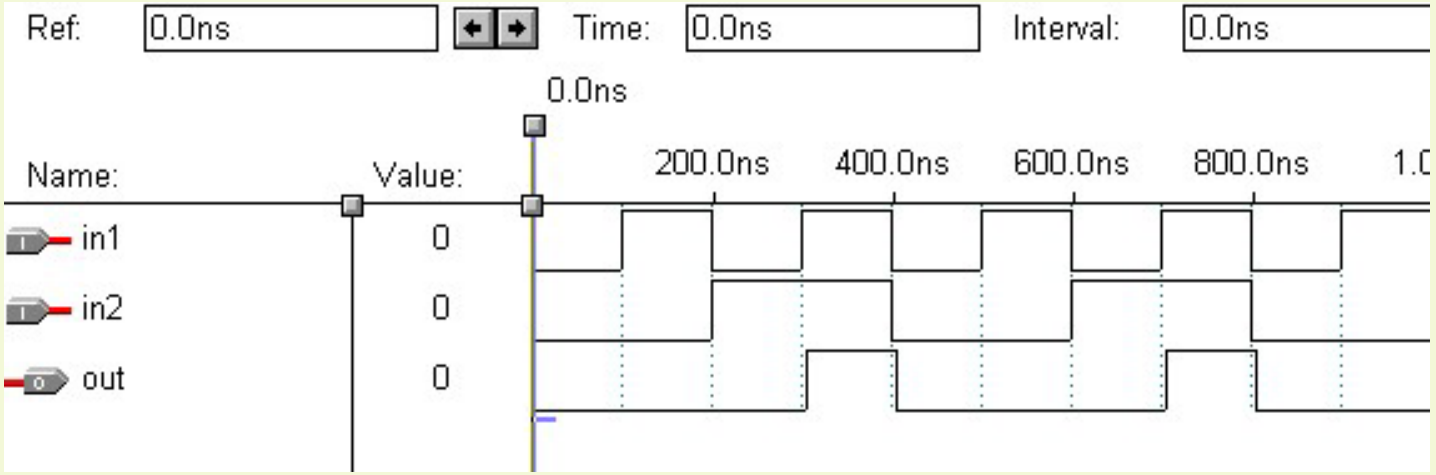
```
wire  [7:0] a, b, c;

assign c[3:0] = a[5:2]
                  & b[7:4];

assign c[0:3] = a[3:0]; // X
```

6. 輸出入埠的宣告

- Verilog 的輸出入埠宣告包括：
  - 輸入埠 (input)
    - 所定義的埠具有輸入訊號的特性。
  - 輸出埠 (output)
    - 所定義的埠具有輸出訊號的特性。
  - 雙向埠 (inout)
    - 所定義的埠同時具有輸入與輸出訊號的特性。

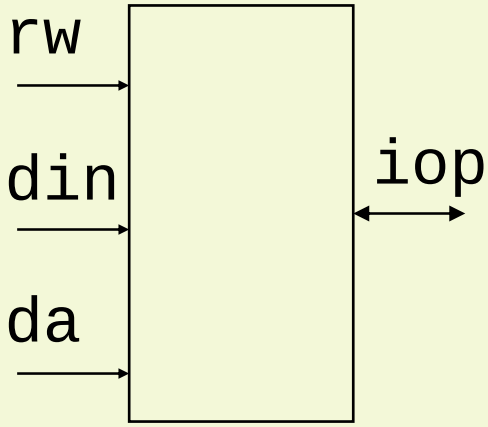
6.1 輸入埠與輸出埠

```
module and_2_1(in1, in2, out);
input  in1, in2;
output out;
       assign out = in1 & in2;
endmodule
```



6.2 雙向埠

```
module bidir(rw, din, da, iop);
input  rw, din, da;
inout  iop;

   assign iop = rw ?
        ((din) ? da : iop)
        : 1'bZ;

endmodule
```



* 在 MAX+plus II 中，電路有時序問題。

6.3 輸出入埠的規定

- 在 Verilog 中，所有輸出入埠的內定資料型態爲 wire。
- 若需將訊號的值存起來，則必須將其資料型態宣告爲 reg。
- 由於輸出入埠可用於與其他模組進行連接，因此可以將其視爲在模組內外相互連接的兩個部分。

7. 邏輯閘階層模型的敘述

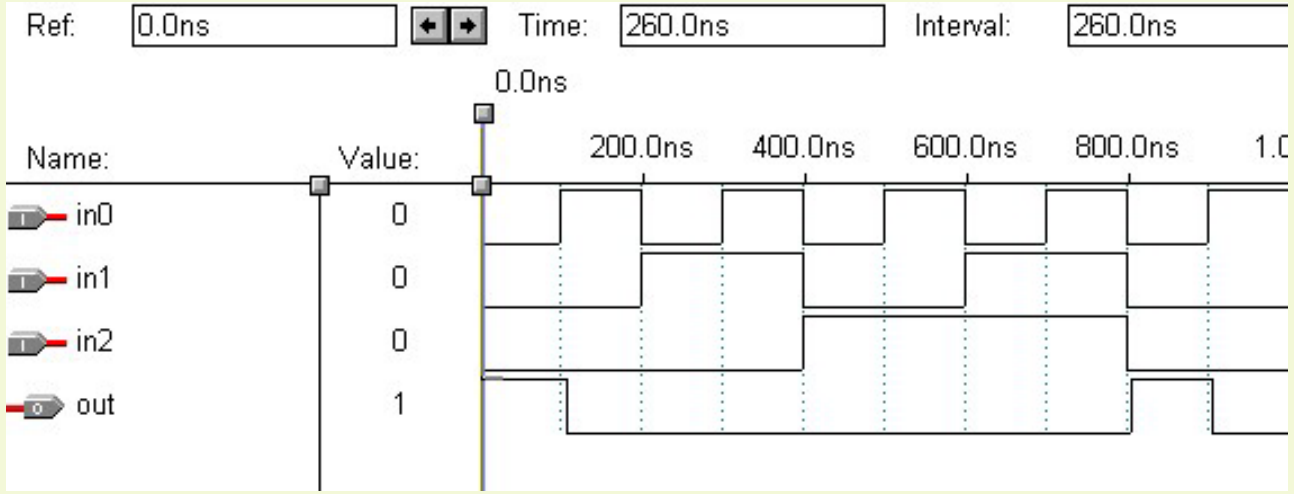
- 在這個階層中，電路模組是由最基本的邏輯閘所連接形成的。
- Verilog 所提供的基本邏輯閘有
  - 多輸入埠邏輯閘
    - and, nand, or, nor, xor, xnor
    - MAX+plus II 最多支援 12 個輸入埠，且不支援 xor 與xnor
  - 多輸出埠邏輯閘
    - buf, not
  - 致能邏輯閘
    - bufif0, bufif1, notif0, notif1

7.1 多輸入埠邏輯閘範例 - 1

```
module test(in, out);
input    [2:0] in;
output   out;
     and u1(out, in[0], in[1], in[2]);
endmodule
```

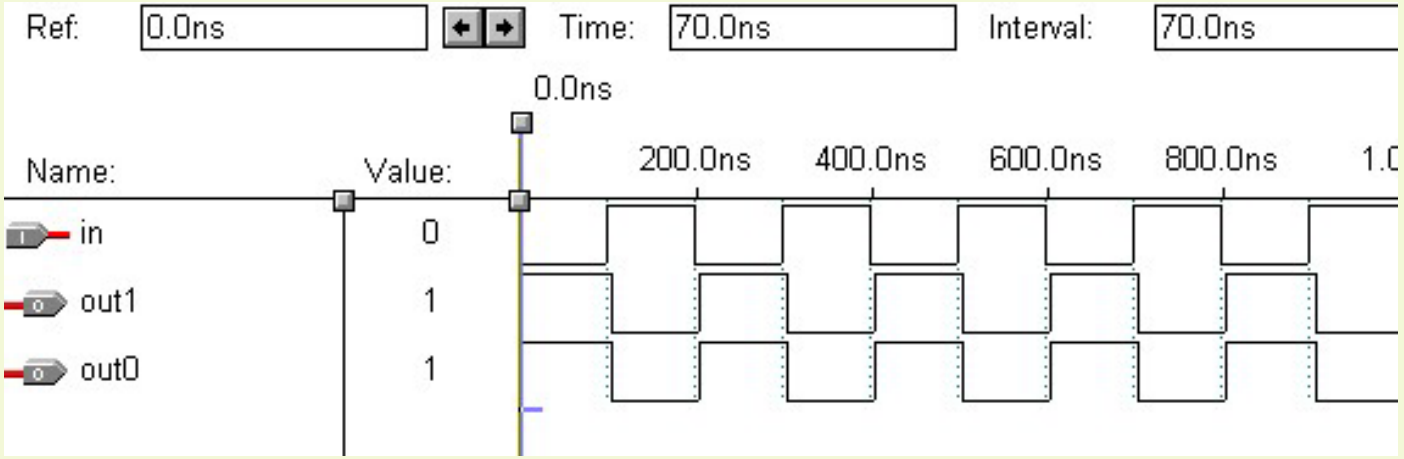
7.1 多輸入埠邏輯閘範例 - 2

```
module test(in, out);
input    [2:0] in;
output   out;
     nor u1(out, in[0], in[1], in[2]);
endmodule
```



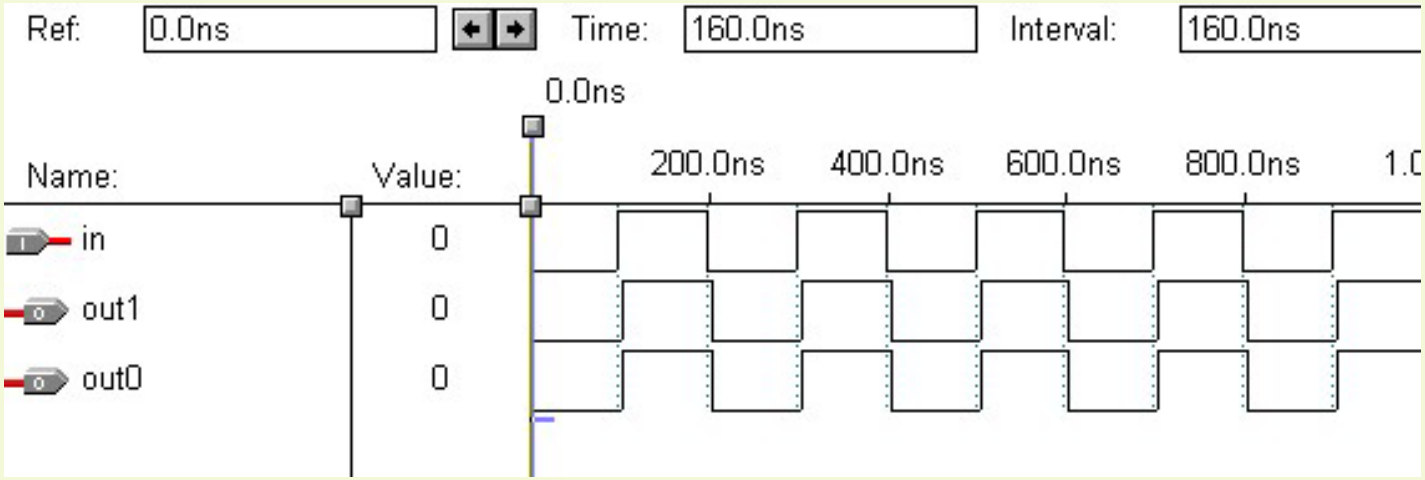
7.2 多輸出埠邏輯閘範例 - 1

```
module test(in, out);
input    in;
output   [1:0] out;
    not u1(out[0], out[1], in);
endmodule
```



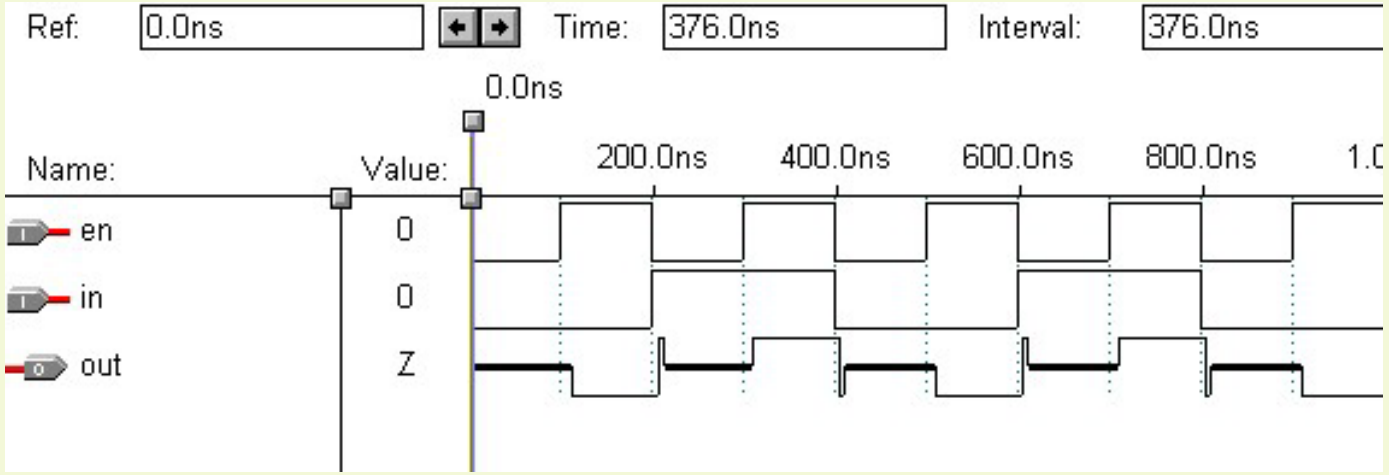
7.2 多輸出埠邏輯閘範例 - 2

```
module test(in, out);
input    in;
output   [1:0] out;
    buf u1(out[0], out[1], in);
endmodule
```



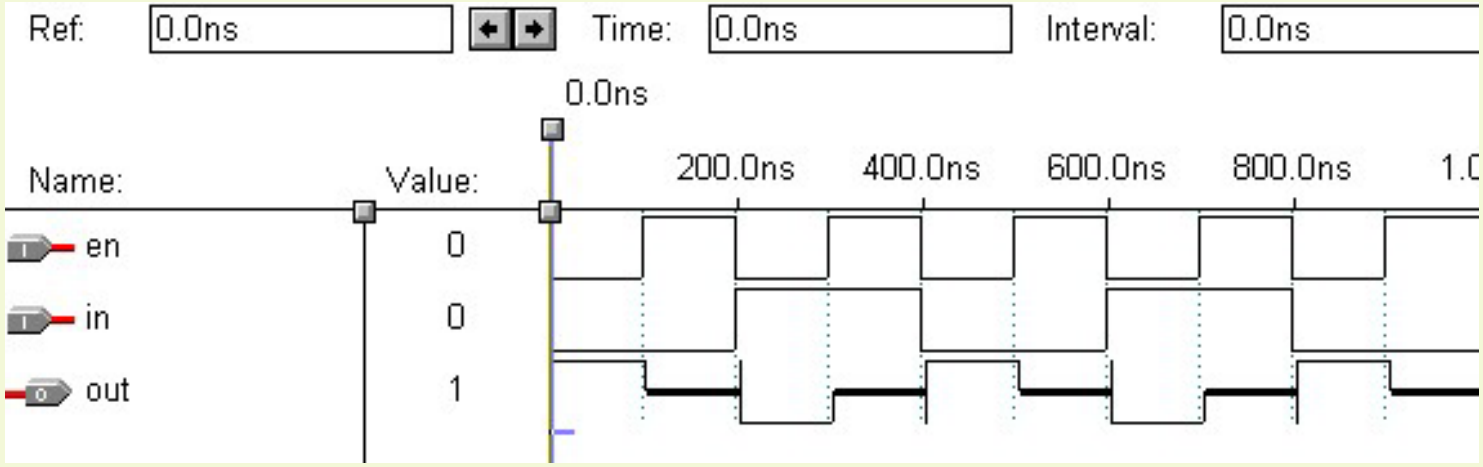
7.3 致能邏輯閘範例 - 1

```
module test(in, out, en);
input    in, en;
output   out;
    bufif1 u1(out, in, en);
endmodule
```



7.3 致能邏輯閘範例 - 2

```
module test(in, out, en);
input    in, en;
output   out;
    notif0 u1(out, in, en);
endmodule
```



8. 資料流模型的敘述

- 在這個階層中，電路的設計重點在於說明資料如何在電路中的傳送過程。
- 在此模型中，常用的敘述有 `assign`：
  - 驅動某個值到 wire、wand、wor 或 tri。
  - 用於描述組合邏輯電路。
  - 必須避免使用迴路式的寫法：

```
assign a = b + a;
```

8.1 組合邏輯電路範例

$X = A \cdot C \cdot D' + B \cdot C' + B \cdot D + C \cdot D$
$Y = A' + B + C$

```
module boolean(a, b, c, d, x, y);
input   a, b, c, d;
output  x, y;
   assign x = (a & c & (!d))
         | (b & (!c)) | (b & d) | (c & d);
   assign y = (!a) | b | c;
endmodule
```

8.2 可用於電路合成的運算子

- Binary bit-wise operators
  - ~ : NOT
  - & : AND
  - | : OR
  - ^ : XOR
  - ~^, ^~ : XNOR

```
module test(in1, in2, out);
input  in1, in2;
output [0:4] out;
    assign out[0] = ~in1;
    assign out[1] = in1 & in2;
    assign out[2] = in1 | in2;
    assign out[3] = in1 ^ in2;
    assign out[4] = in1 ~^ in2;
endmodule
```

8.2 可用於電路合成的運算子

- Unary reduction operators
  - &：AND
  - ~&：NAND
  - |：OR
  - ~|：NOR
  - ^：XOR
  - ~^, ^~：XNOR

```
module test(in, out);
input    [0:3] in;
output   [0:5] out;
     assign out[0] = &in;
     assign out[1] = ~&in;
     assign out[2] = | in;
     assign out[3] = ~| in;
     assign out[4] = ^ in;
     assign out[5] = ~^ in;
endmodule
```

8.2 可用於電路合成的運算子

- Logical operators
  - ！：NOT
  - &&：AND
  - ||：OR

```
module test(in1, in2, out);
input  in1, in2;
output [0:2] out;
    assign out[0] = ! in1;
    assign out[1] = in1 && in2;
    assign out[2] = in1 || in2;
endmodule
```

8.2 可用於電路合成的運算子

- 2's complement arithmetic
  - + : Add
  - - : Subtract
  - * : Multiply
  - / : Divide
  - % : Module

```
module test(i1, i2, o1, o2, o3);
input    i1, i2;
output   [0:1] o1, o2, o3;
    assign o1 = i1 + i2;
    assign o2 = i1 - i2;
    assign o3 = i1 * i2;
endmodule
```

8.2 可用於電路合成的運算子

- Relational operators
  - >
  - <
  - >=
  - <=
  - ==
  - !=

```
module test(a, b, lt, eq, le);
input    [0:3] a, b;
output   lt, eq, le;
     assign lt = (a > b);
     assign eq = (a == b);
     assign le = (a < b);
endmodule
```

8.2 可用於電路合成的運算子

- Logical shift operators
  - >>
  - <<

```
module test(in, out1, out2);
input    [0:3] in;
output   [0:3] out1, out2;
     assign out1 = in << 1;
     assign out2 = in >> 1;
endmodule
```

* MAX+plus II 僅支援常數值的移位量。

8.2 可用於電路合成的運算子

- Conditional operator
  - `?:`

```
module test(a, b, sel, out);
input    a, b, sel;
output   out;
     assign out = sel ? a : b;
endmodule
```

8.2 可用於電路合成的運算子

- Duplication operator
  - {{}}
- Concatenation operator
  - {}

```
module test(in, out1, out2);
input    [0:7] in;
output   [0:7] out1, out2;
     assign out1 = {2{4'b1100}} + in;
     assign out2 = {{3{2'b10}}, 2'b00} + in;
endmodule
```

9. 行為模型的敘述

- 在這個階層中，我們只需考慮電路模組的功能，而不需考慮其硬體的詳細內容。
- Verilog 的時序控制爲以事件爲基礎的時序控制 (event-based timing control)：
  - 事件：
    - 接線或暫存器的值被改變。
    - 模組的輸入埠接收到新的值。
  - 可用於電路合成的控制方式：
    - 正規事件控制：正緣、負緣、訊號值改變
    - 多事件或訊號控制

9.1 always 敘述

- always 敘述的觀念有如監督程式一般，隨時監看著輸出入埠訊號的變化，然後告知模組內部進行相關的處理。
- 語法

```
always
   statement
```

```
always@(event_expression)
   statement
```

```
always
begin
   statements
end
```

```
always@(event_expression)
begin
   statements
end
```

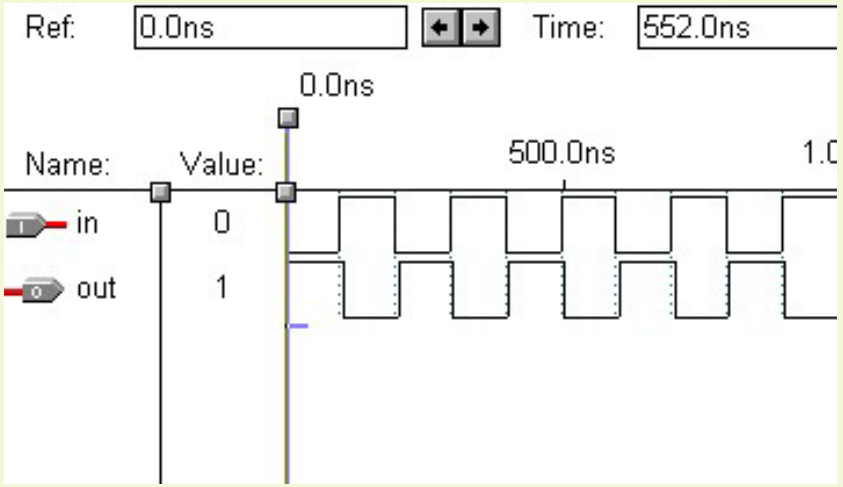
9.1 always@ 敘述

- always@(…) 括弧內的運算式稱之爲事件運算式 (event expression)，其可以是
  - 單一訊號
    - 屬於準位觸發 (level trigger)。
  - 多個訊號
    - 利用 or 關鍵字連接不同訊號。
    - 屬於準位觸發 (level trigger)。
  - 邊緣觸發
    - 正緣觸發：posedge
    - 負緣觸發：negedge

9.1 always@ 敘述

- 單一訊號範例

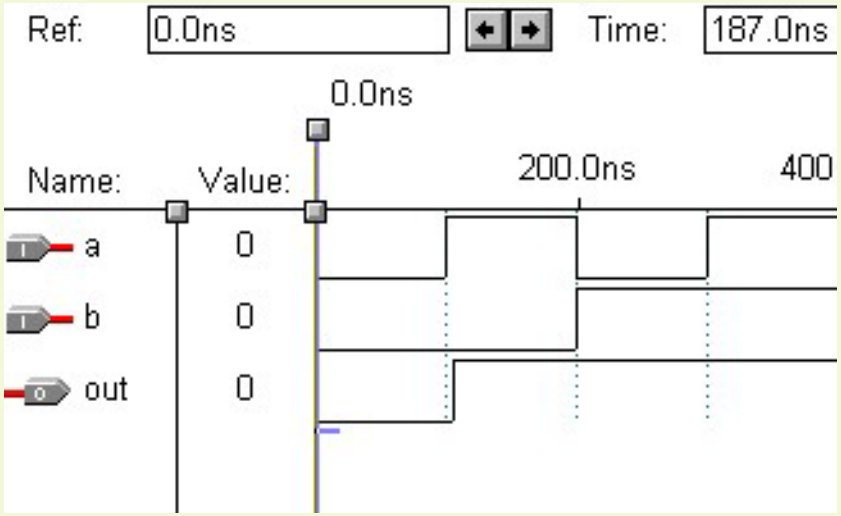
```
module test(in, out);
input   in;
output  out;
reg     out;
    always@(in)
        out = ~ in;
endmodule
```



9.1 always@ 敘述

- 多個訊號的範例

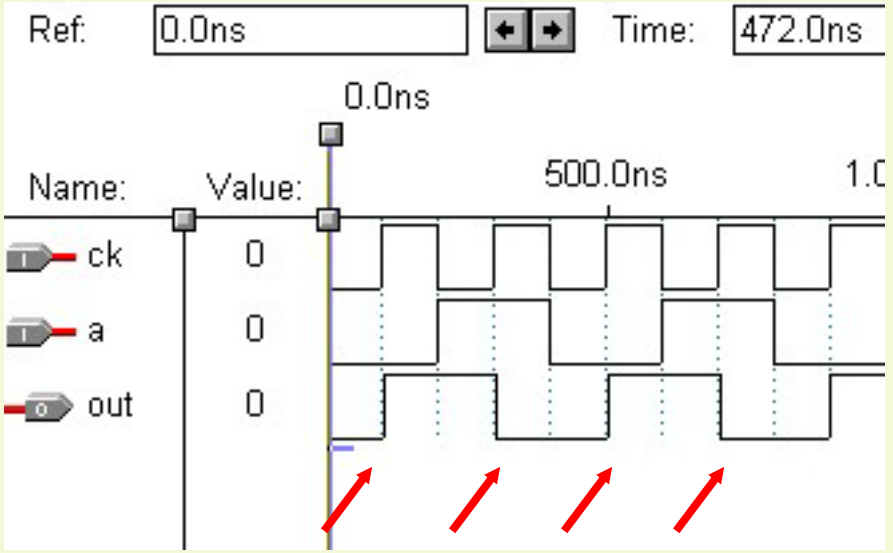
```
module test(a, b, out);
input   a, b;
output  out;
reg     out;
    always@(a or b)
    begin
        out = a | b;
    end
endmodule
```



9.1 always@ 敘述

- 邊緣觸發：posedge

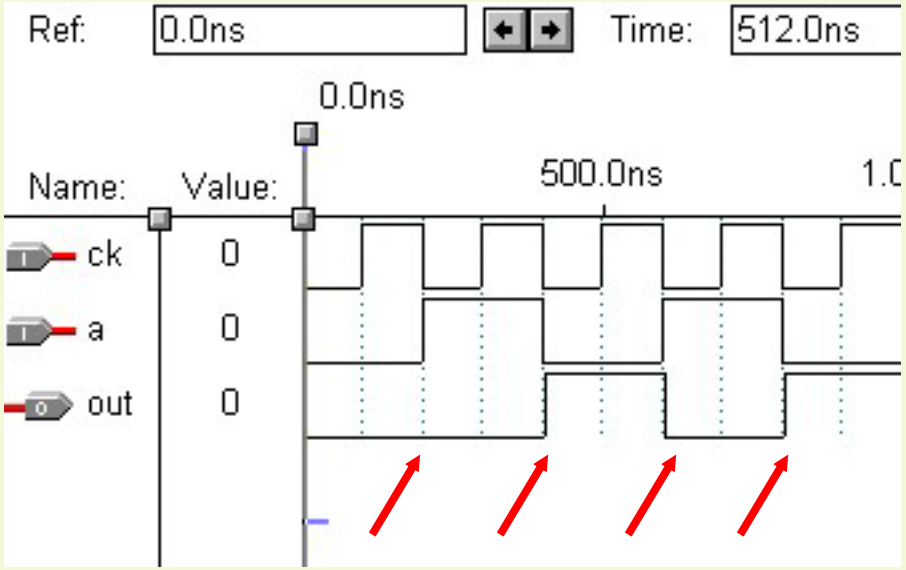
```
module test(a, ck, out);
input   a, ck;
output  out;
reg     out;
    always@(posedge ck)
    begin
        out = ~ a;
    end
endmodule
```



9.1 always@ 敘述

- 邊緣觸發：negedge

```
module test(a, ck, out);
input   a, ck;
output  out;
reg     out;
    always@(negedge ck)
    begin
        out = ~ a;
    end
endmodule
```



9.2 if 敘述

- 可用來進行訊號值的判斷，後根據判斷結果執行相關處理。
- if 敘述能處理正準位與負準位觸發兩種訊號。
- 語法：

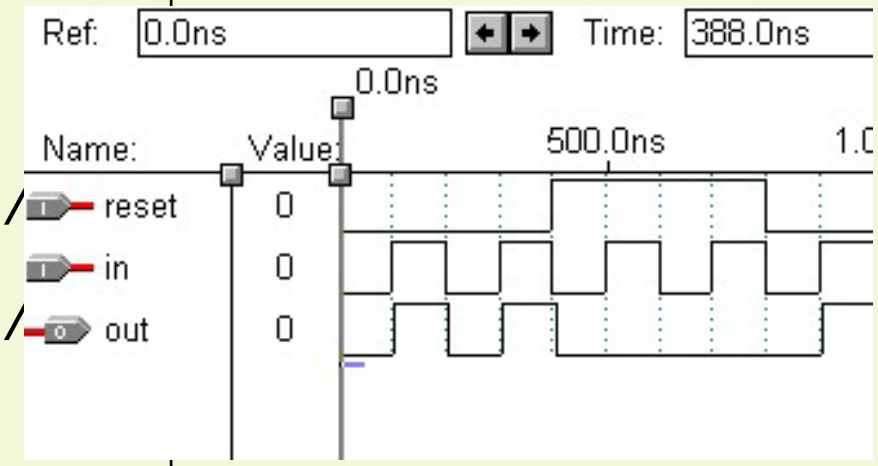
```
if (expression)
      statement
```

```
if (expression)
begin
      statements
end
```

9.2 if 敘述

- 準位觸發

```
module test(reset, in, out);
input   reset, in;
output  out;
reg     out;
    always
    begin
        if (reset == 1'b1) /
            out = 0;
        if (reset == 1'b0) /
            out = in;
    end
endmodule
```



9.2 if 敘述

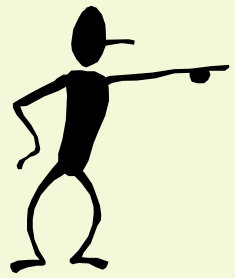
- 準位觸發 – 另一種寫法

```
module test(reset, in, out);
input   reset, in;
output  out;
reg     out;
    always
    begin
        if (reset)      // 正
            out = 0;
        if (! reset)    // 負
            out = in;
    end
endmodule
```

9.3 if-else 敘述

- 語法：

```
if (expression)
        statement
else  statement
```



```
if (expression)
begin
        statements
end
else
begin
        statements
end
```

9.3 if-else 敘述

- 範例：優先權編碼器 (priority encoder)

```
always@(X, Y, Z)
begin
     if (Z)                    // highest
          out = result4;
     else if (Y)
          out = result3;
     else if (X)
          out = result2;
     else out = result1; // lowest
end
```

9.4 case 敘述

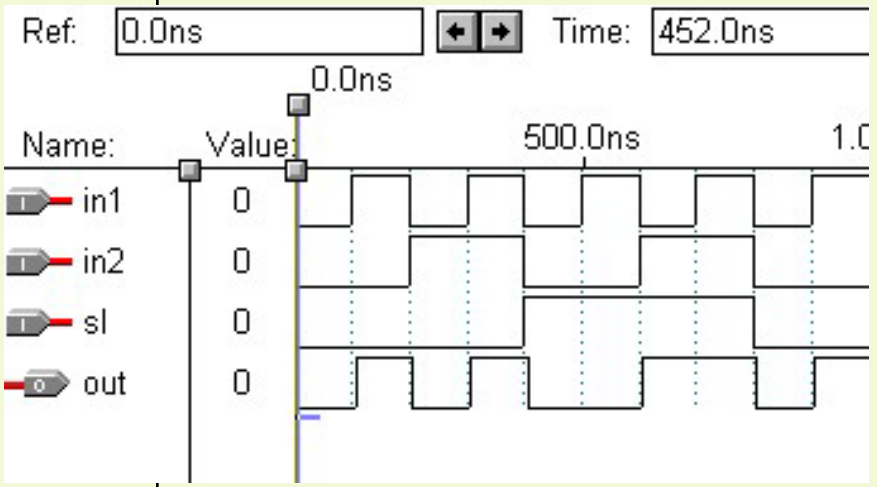
- case 敘述爲一多路分支選擇的敘述。
- 如果電路中所有可能的分支判別條件都被指定了，則稱爲 full case。
- 語法：

```
case (expression)
     alter_1, alter_2: stm_1;
     alter_3: stm_2;
     …
     default: default_stm;
endcase
```

9.4 case 敘述

- 範例：2X1 Multiplexor

```
module mux21(in1, in2, sl, out);
input   in1, in2, sl;
output  out;
reg     out;
   always@(in1 or in2 or sl)
   begin
      case (sl)
         1'b0: out = in1;
         1'b1: out = in2;
      endcase
   end
endmodule
```



9.4 case 敘述

- 範例：Priority encoder

```
module test(X, Y, Z, out);
input  [0:1] X, Y, Z;
output [0:2] out;
reg    [0:2] out;
always@(X or Y or Z)
begin
    case (2'b11)
        X: out = 3'b001;
        Y: out = 3'b010;
        Z: out = 3'b100;
        default: out = 3'b000;
    endcase
end
endmodule
```

當 X、Y、Z 均爲 2‘b11 時，out 的值爲何？

9.4 casex 與 casez 敘述

- Verilog 中還有 casex 與 casez 兩種 case 敘述。
  - casex
    - 允許 alter 選項中有未知值 (?) 出現。
    - experssion 不允許出現 Z 或 X。
  - casez
    - 允許 alter 選項中有未知值 (?) 出現。
    - experssion 允許出現 Z。
  - MAX+plus II 不支援 casex 與 casez。

9.5 迴圈敘述 -- for

- Verilog 提供有 for、while、repeat 和 forever 等迴圈敘述。
- MAX+plus II 僅支援 for 敘述而已。
- 語法：

```
for (statement; expression; statement)
begin
     statements;
end
```

- 所有迴圈敘述僅能在 always 敘述中執行。

9.5 迴圈敘述 -- for

```
module forloop(a, b, out);
input    [0:7] a, b;
output   [0:7] out;
reg      [0:7] out;
integer  i;
   always@(a or b)
   begin
      for (i = 0;i <=  7; i = i + 1)
         out[i] = a[i] | b[i];
   end
endmodule
```

9.6 function 敘述

- Verilog 中的 function 敘述的用途與 C 語言中只利用 function 傳回計算結果的函數是相同的。
- function 的傳回值是儲存在與函數名稱相同的變數中。
- 語法：

```
function <range> name;
input 參數宣告;
        函數主體
endfunction
```

9.6 function 敘述

```
module fun_as(a, b, Add,
              Sub, Inc, c);
input   [0:3] a, b;
input   Add, Sub, Inc;
output  [0:3] c;
reg     [0:3] c;
    always@(a or b or Add
              or Sub or Inc)
    begin
        if (Add)
            c = a + b;
        else
            c = ff(a, b, Sub, Inc);
    end
```

```
    function [0:3] ff;
    input   [0:3] a, b;
    input   Sub, Inc;
        begin
        if (Sub)
            ff = a – b;
        else if (Inc)
            ff = a + 1;
        else
            ff = a – 1'b1;
        end
    endfunction
endmodule
```

* 在 MAX+plus II 中，function 必須在使用之前被宣告。

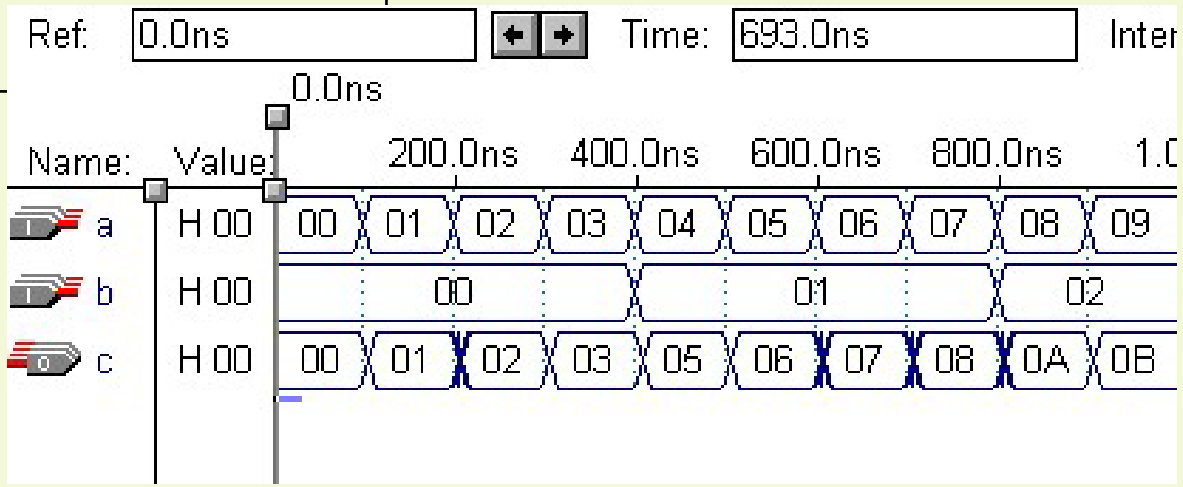
9.7 parameter 敘述

- parameter 敘述可用於定義一個常數供模組用來定義輸出入埠的寬度、向量的大小等。
- 語法：

```
parameter name = number;
```

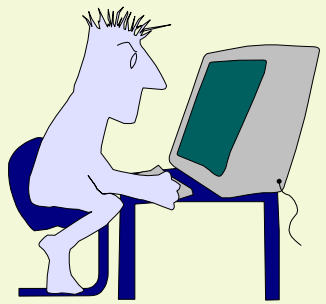
9.7 parameter 敘述

```
module add8(a, b, c);
parameter width=8;
input  [1:width] a, b;
output [1:width] c;
    assign c = a + b;
endmodule
```



10. 編譯命令

- 如 C 語言之編譯命令 (compiler directives) 一般，Verilog 也提供了一些編譯命令供使用者利用，以便指示編譯程式進行一些編譯的前置作業。
- 常見的編譯命令有：
  - `` `include ``
  - `` `define ``



10.1 `include

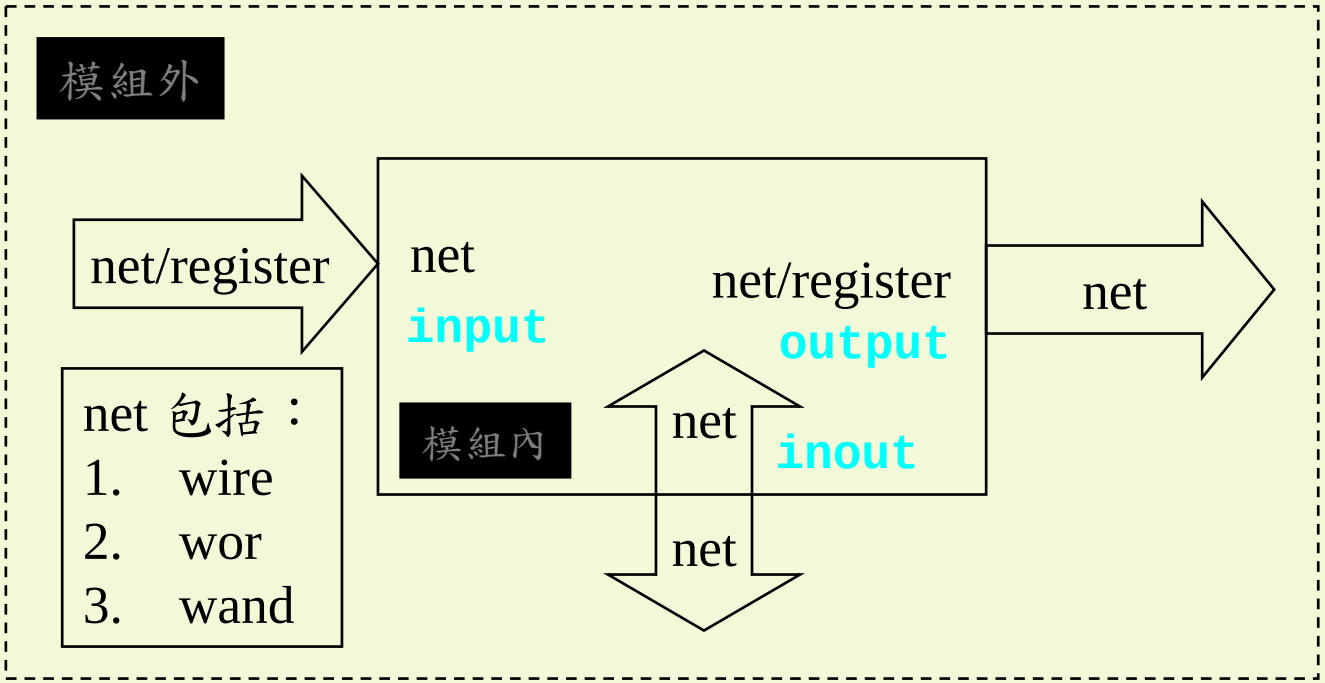
- `include 可將其他 Verilog 檔案引入至目前的檔案一起編譯、合成。

```
module fa1(a, b, ci, s, co);
input    a, b, ci;
output   s, co;
   assign {co, s} = a + b + ci;
endmodule
```

```
`include "fa1.v"
module fa2(a, b, ci, s, co);
input    [1:0] a, b;
input    ci;
output   [1:0] s;
output   co;
wire     c1;
   fa1 ffa0(a[0],b[0],ci,s[0],c1);
   fa1 ffa1(a[1],b[1],c1,s[1],co);
endmodule
```

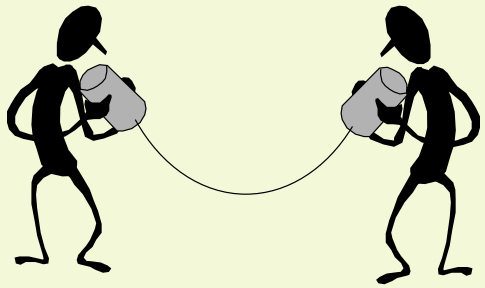
10.2 模組間埠對應的方式

- 當模組引用其他模組的時候，必須將訊號透過該模組的輸出入埠傳送給該模組。



10.2 模組間埠對應的方式

- 在 Verilog 中，模組間埠的對應方式可分爲：
  - 依模組定義之輸出入埠的順序來連接。
    - In-Order
  - 依「指定輸出入埠名稱」的方式來連接。
    - By-name



10.2 模組間埠對應的方式

- 依模組定義之輸出入埠的順序來連接：

```
module or_2(a, b, c);
input    a, b;
output   c;
    assign c = a | b;
endmodule
```

```
`include "or_2.v"
module or_8(a, b, c);
input    [3:0] a, b;
output   [3:0] c;
   or_2  o1(a[0], b[0], c[0]);
   or_2  o2(a[1], b[1], c[1]);
   or_2  o3(a[2], b[2], c[2]);
   or_2  o4(a[3], b[3], c[3]);
endmodule
```

10.2 模組間埠對應的方式

- 依「指定輸出入埠名稱」的方式來連接：

```
`include "or_2.v"
module or_8(a, b, c);
input    [3:0] a, b;
output   [3:0] c;
   or_2  o1(.a(a[0]), .b(b[0]), .c(c[0]));
   or_2  o2(.c(c[1]), .b(b[1]), .a(a[1]));
   or_2  o3(a[2], b[2], c[2]);
   or_2  o4(a[3], b[3], c[3]);
endmodule
```

10.3 `define

- `define 可用來定義模組中所用到的常數。

```
`define SI  4
module buf4(in, out, clock);
input   [0:`SI-1] in;
input   clock;
output  [0:`SI-1] out;
reg     [0:`SI-1] out;
integer i;
    always@(posedge clock)
    begin
        for (i=0; i<`SI; i=i+1)
            out[i] = in[i];
    end
endmodule
```

10.4 defparam 敘述

- 可用於取代引入模組內之以 parameter 敘述所定義之常數值。
- 語法：

```
defparam instance_name.id1 = number1;
```

10.4 defparam 敘述

```
module or8(a, b, c);
parameter width=8;
input  [1:width] a, b;
output [1:width] c;
    assign c = a | b;
endmodule
```

```
`include “or8.v”
module or4(a, b, c);
parameter size = 4;
input  [1:size] a, b;
output [1:size] c;
    or8o4(a, b, c);
    defparam o4.width=4;
endmodule
```

在 MAX+plus II 中，defparam 所重新定義的值必須小於原來 parameter 所定義的值。

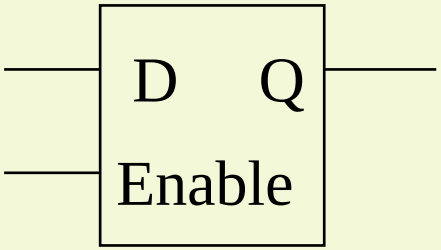
11. 循序邏輯電路範例

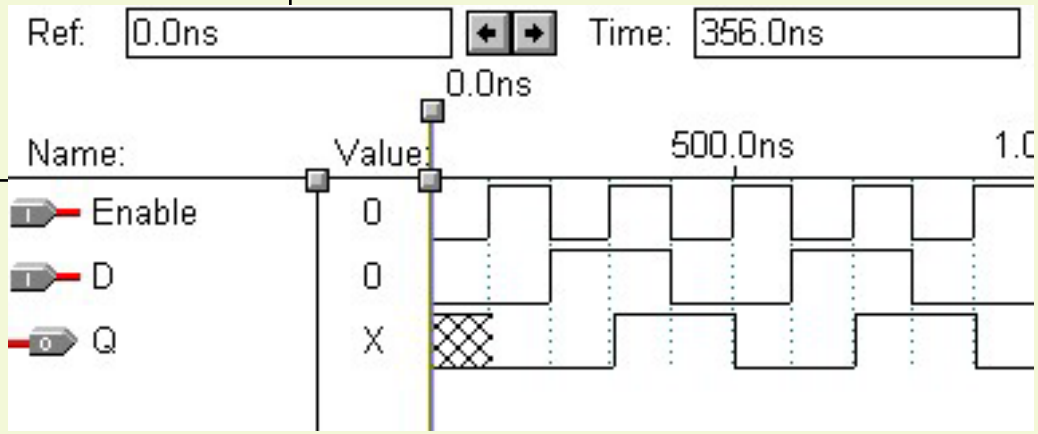
- 循序邏輯電路最主要包含：
  - 記憶元件 (memory)
    - 閂鎖 (latch)
    - 正反器 (flip-flop)
    - 暫存器 (register)
  - 計數器 (counter)
    - 同步計數器 (synchronous counter)
    - 漣波計數器 (ripple counter)
  - 有限狀態機器 (finite state machine)

11.1 記憶元件 -- 閂鎖

- 具有 output enable 控制的 D-type latch：

```
module latch_d(Enable, D, Q);
input  Enable, D;
output Q;
reg    Q;
    always@(Enable or D)
        if (Enable)
            Q = D;
endmodule
```

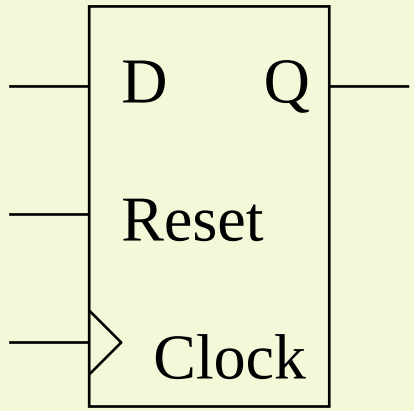




11.1 記憶元件 -- 正反器

- 具有同步重置的 D-type flip/flop：

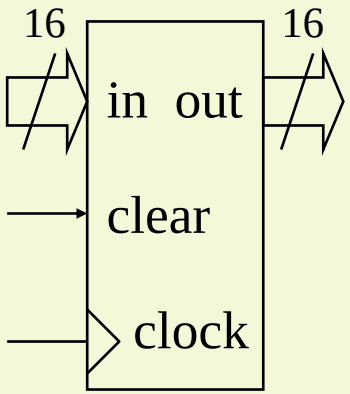
```
module D_FF_SR(clock, reset, D, Q);
input  clock, reset, D;
output Q;
reg    Q;
    always @(posedge clock)
    begin
        if (reset == 1'b1)
                Q = 1'b0;  // reset
        else    Q = D;
    end
endmodule
```



11.1 記憶元件 -- 暫存器

- 具有 clear 控制訊號的同步 16 位元暫存器：

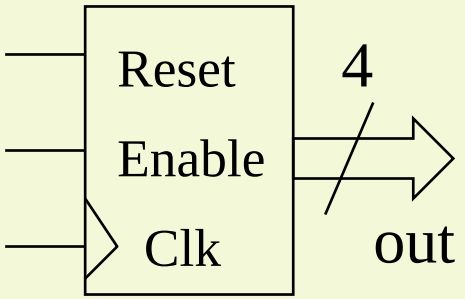
```
module reg_16(clear, clock, in, out);
input      clear, clock;
input      [0:15] in;
output     [0:15] out;
reg        [0:15] out;
     always@(posedge clock)
     begin
          if (clear == 1'b1)
               out = 16'b0;
          else out = in;
     end
endmodule
```



11.2 計數器 -- 同步計數器

- 具有 reset 與 enable 控制訊號的 4 位元同步計數器：

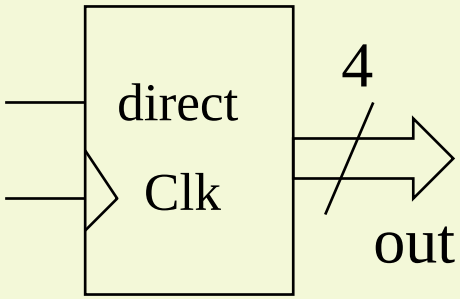
```
module counter1(reset, enable, clk, out);
input   reset, enable, clk;
output  [0:3] out;
reg     [0:3] out;
    always@(posedge clk) begin
        if (reset)
            out = 0;
        else if (enable)
            out = out + 1;
    end
endmodule
```



11.2 計數器 -- 同步計數器

- 上下數計數器

```
module counter1(direct, clk, out);
input       direct, clk;
output      [0:3] out;
reg         [0:3] out;
integer     value;
      always@(posedge clk) begin
            if (direct)
                  value = 1;
            else  value = -1;
            out = out + value;
      end
endmodule
```

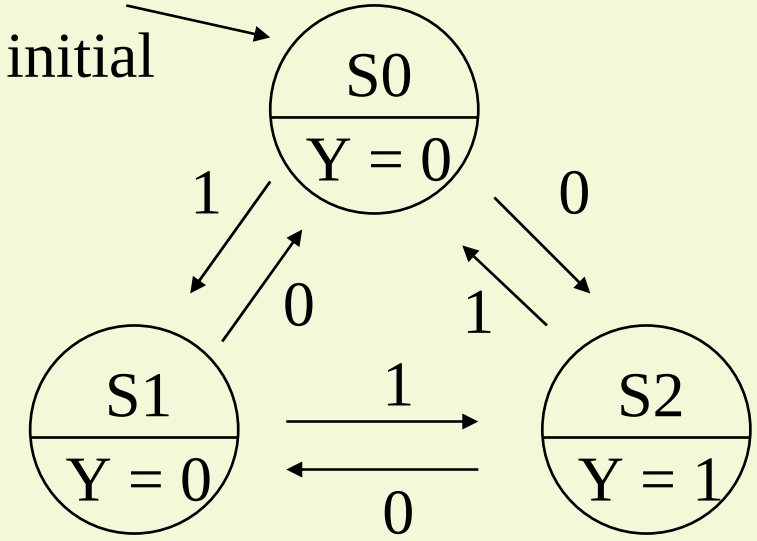


11.3 有限狀態機器

- 有限狀態機器是循序電路中變化性最多的一種電路，常被應用在一些具有特殊要求的控制電路中。
- 有限狀態機器依其輸入訊號、輸出訊號、以及狀態之間的關係，可分成：
  - Moore machine
    - 輸出訊號僅與目前的狀態有關，而與輸入訊號無關。
  - Mealy machine
    - 輸出訊號與輸入訊號及狀態均有關係。

11.3 有限狀態機器

- Moore machine



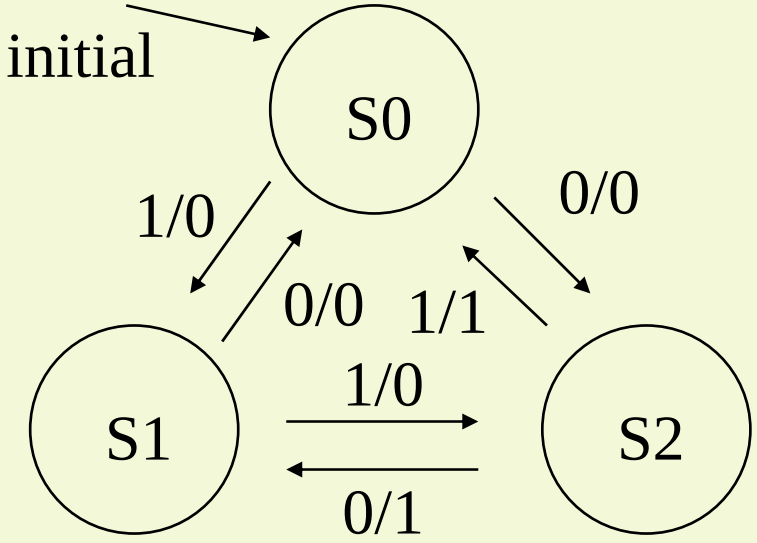
目前狀態	次一狀態		輸出 Y
	輸入 0	輸入 1	
S0	S2	S1	0
S1	S0	S2	0
S2	S1	S0	1

11.3 有限狀態機器

```
module moore(reset, clk, x, y);
input reset, clk, x;
output   y;
reg      y;
parameter   s0=2'b00,
            s1=2'b01, s2=2'b10;
reg   [0:1] ps, ns;

   always@(reset or ps)
   begin
      if (reset)
      begin
         ns = s0; y = 0;
      end
      else
      begin
         case(ps)
            s0: begin
                  if (x == 0)
                     ns = s2;
                  else  ns = s1;
                  y = 0;
                end
            s1: begin
                  if (x == 0)
                     ns = s0;
                  else ns = s2;
                  y = 0;
                end
                ……
         endcase
   end end // else & always

   always@(posedge clk)
      ps = ns;

endmodule
```

11.3 有限狀態機器

- Mealy machine



目前狀態	次一狀態		輸出	
	X=0	X=1	X=0	X=1
S0	S2	S1	0	0
S1	S0	S2	0	0
S2	S1	S0	1	1

11.3 有限狀態機器

```
module mealy(reset, clk, x, y);
input reset, clk, x;
output   y;
reg      y;
parameter   s0=2'b00,
            s1=2'b01, s2=2'b10;
reg   [0:1] ps, ns;

   always@(reset or ps)
   begin
      if (reset)
      begin
         ns = s0; y = 0;
      end
      else
      begin
```

```
      case(ps)
         s0: begin
               if (x == 0)
                  ns = s2;
               else  ns = s1;
               if (x == 0)
                  y = 0;
               else  y = 0;
             end
            ……
      endcase
   end end // else & always

   always@(posedge clk)
      ps = ns;

endmodule
```

LAB

LAB 1

● 題目：

請使用verilog設計一全加器(Full Adder)

● 說明：

-輸入腳：

– in1：加法運算元1
– in2：加法運算元2
– carryin ：進位輸入值

-輸出腳：

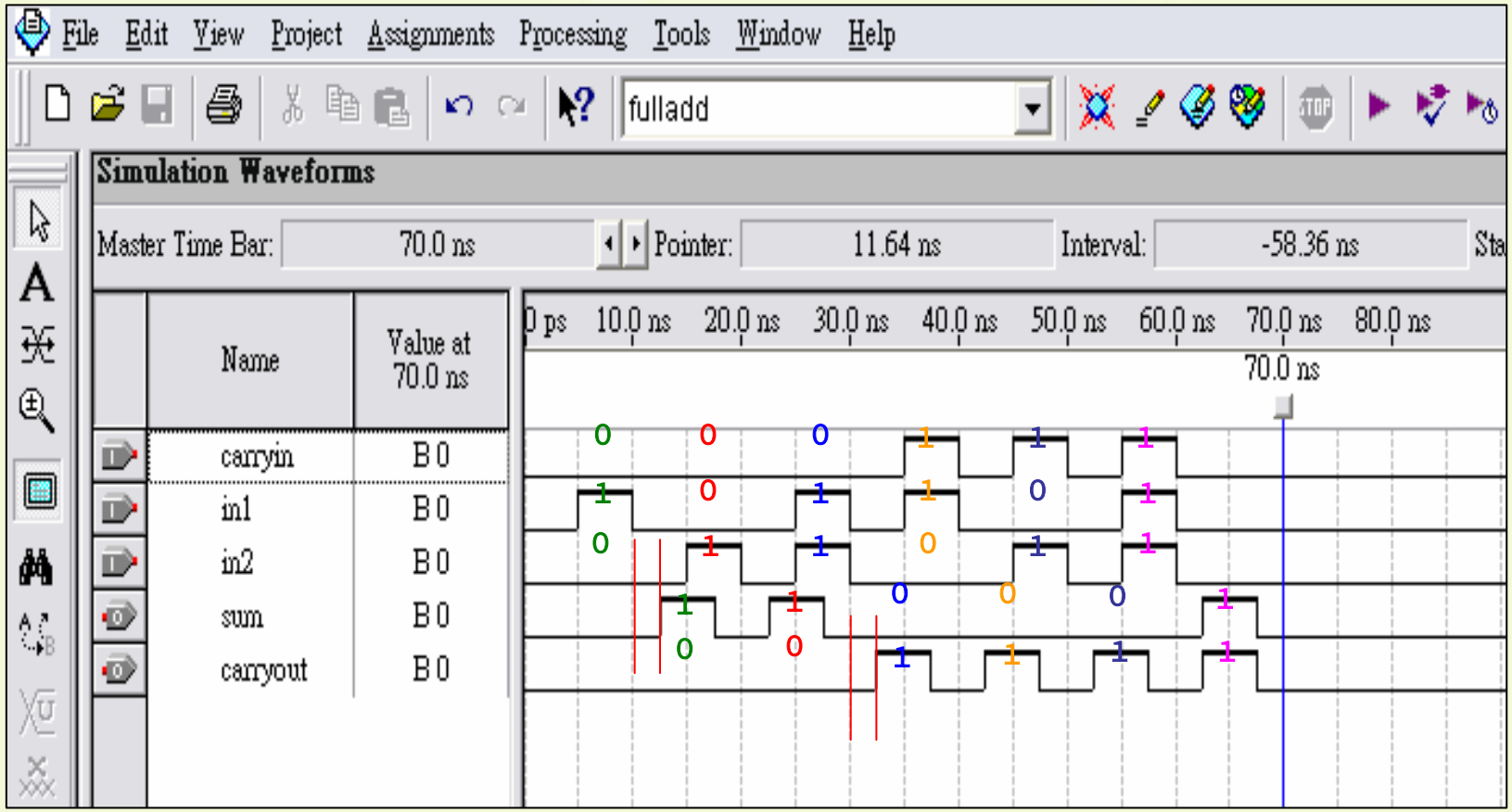
– sum：加法總和
– carryout：進位輸出值

布林方程式：

– sum=in1⊕in2 ⊕ carryin
– Carryout=in1・in2+in1・carryin+in2・carryin

*寬度皆為一位元

模擬波形圖：



說明：輸出結果sum、carryout都有半週期的延遲(delay)。

LAB 2

●題目：

設計一個四對一多工器

●腳位：

腳位：

控制線2條：S0、S1

輸出線4條：Y[3..0]

S1	S2	Y[3..0]
0	0	0001
0	1	0010
1	0	0100
1	1	1000

LAB 3

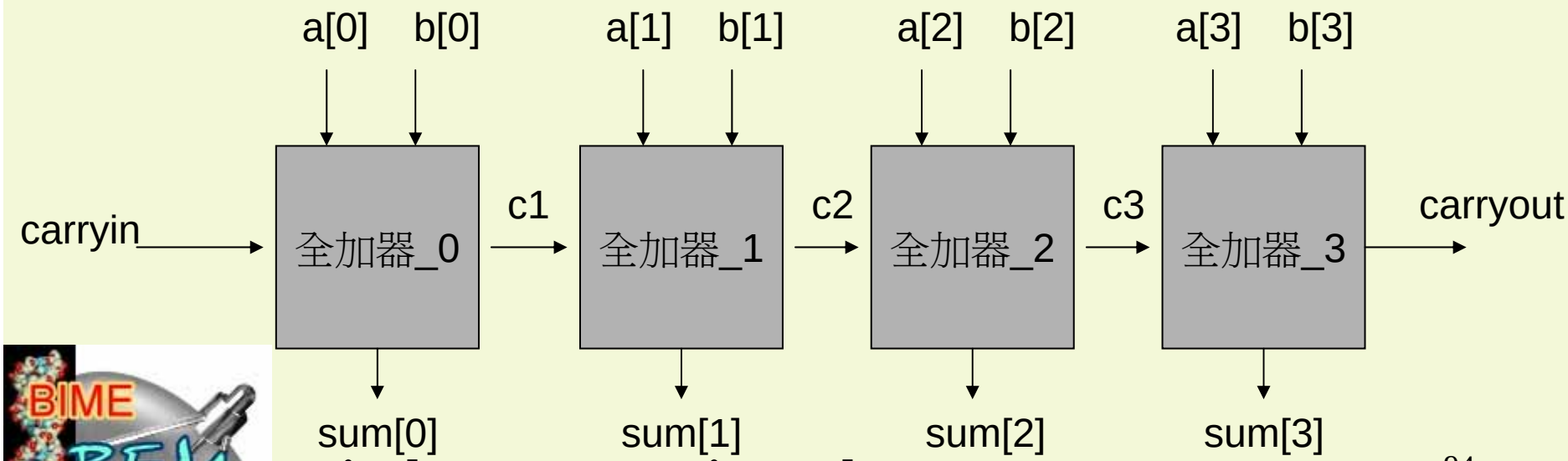
●題目：

利用hw1設計一個四位元進位加法器

●腳位：

輸入：a[3..0] 、b[3..0] 、carryin

輸出：sum[3..0] 、carryout



Thanks for your attention.