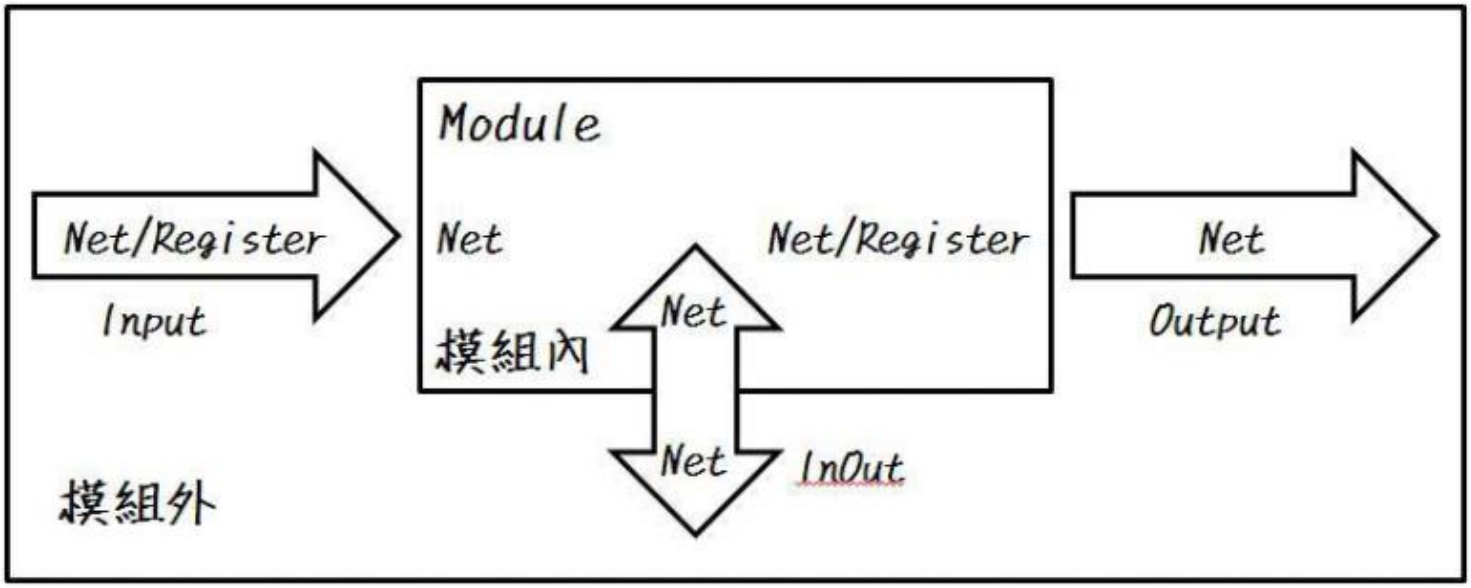
# Computer Architecture

Verilog tutorial

# Verilog 基本架構




```
module 模組名稱( 輸出入埠名稱 );
    輸出入埠 敘述
    資料型態 敘述
    內部電路 敘述
endmodule
```

# Verilog 資料型態

- **資料狀態**
  - 0　　邏輯0
  - 1　　邏輯1
  - x或X　未知的值( Unknow )或浮接( Floating )
  - z或Z　高阻抗( High Impendence )

# Verilog 資料型態

## 連接線Net ( wire)

```
module 模組名稱( a, b, c, d, e );

    input a, b;
    output c, d, e;

    wire c;
    wand d;
    wor e;

    // wire接一起 → 錯誤
    assign c = a;
    assign c = b;

    // wire-and → d = a&b
    assign d = a;
    assign d = b;

    // wire-or → e = a|b
    assign e = a;
    assign e = b;

endmodule
```

## 暫存器Register ( reg )

```
module 模組名稱( a, b, c );
    input a;
    output b, c;

    reg b, rTmp;

    // 範例1
    always @(*) begin
        b = a;
    end

    // 範例2
    assign c = rTmp;

endmodule
```

# 行為層次 Behavior Level

- always敘述

```
always @( 事件1, 事件2, … )
    begin
    敘述1;
    敘述2;
    … … …
end
```

- if-else敘述

```
if( 判斷條件1 )
    begin
        敘述1;
    end
else if( 判斷條件2 )
    begin
        敘述2;
    end
else
    begin
        敘述3;
    end
```

# 行為層次 Behavior Level

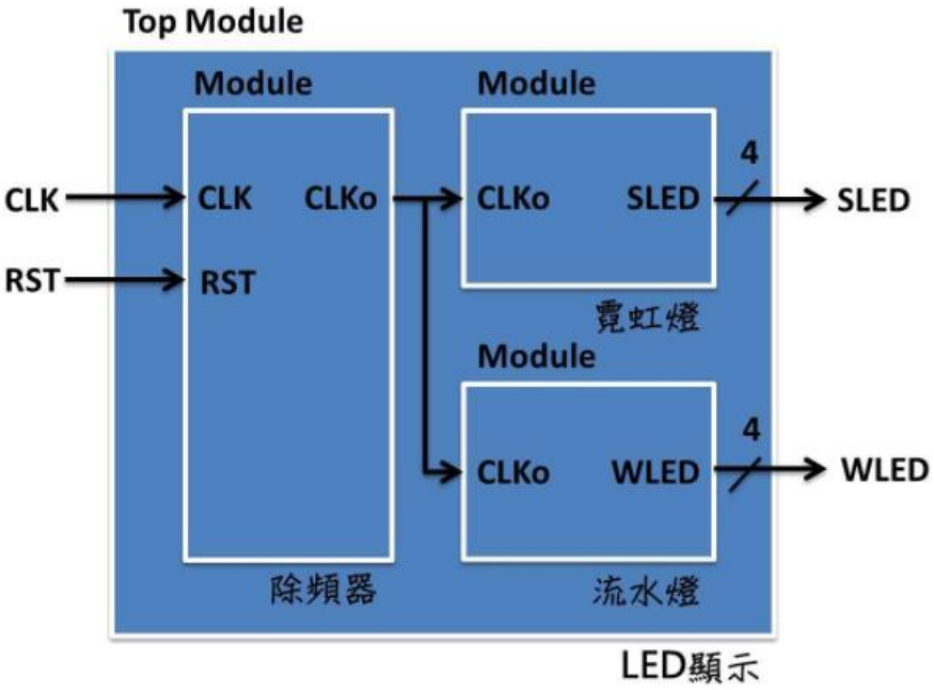
- case敘述

```
case( expr )
  item 1:
    begin
      敘述1;
    end
  item 2:
    begin
      敘述2;
    end
  … … …
  default:
    敘述n;
endcase
```

- Blocking/
  Non-Blocking敘述

```
input In;
reg [3:0] A, B, C;

always @( posedge CLK ) begin

    /* Blocking */       // 有順序執行，同C概念
    A[0] = In;           // 1CLK後A[0] = In
    A[1] = A[0];         // 1CLK後A[1] = A[0] = In
    A[2] = A[1];         // 1CLK後A[2] = A[1] = A[0] = In
    A[3] = A[2];         // 1CLK後A[3] = A[2] = A[1] = A[0] = In

    /* Non-Blocking */   // 同時執行，此範例有資料平移的效果
    B[0] <= In;          // 1CLK後B[0]存進In, B[1]存進B[0](存In之前的值)
    B[1] <= B[0];        // 2CLK後B[1]存進In
    B[2] <= B[1];        // 3CLK後B[2]存進In
    B[3] <= B[2];        // 4CLK後B[3]存進In

    /* 混合 */           // 盡量不要常用
    C[0] <= In;          // 1CLK後C[0]存進In, C[1], C[2]存進C[0](存In之前的值)
    C[1] = C[0];         // 2CLK後C[1] = C[2] = C[3] = In
    C[2] = C[1];         //
    C[3] <= C[2];        //

end
```

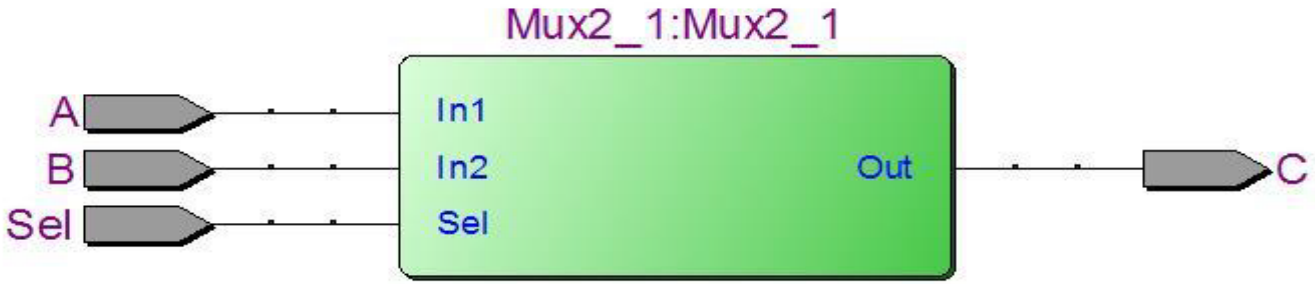
# 模組化與階層化




```
// 連接除頻器module
wire _CLK, _RST, _CLKo;

Freq_Divider FD_1Hz(
    .CLK( _CLK ),
    .CLKo( _CLKo ),
    .RST( _RST )
);
```

```
// 連接除頻器module
wire _CLK, _RST, _CLKo;

Freq_Divider FD_1Hz(
    _CLK,
    _RST,
    _CLKo
);
```

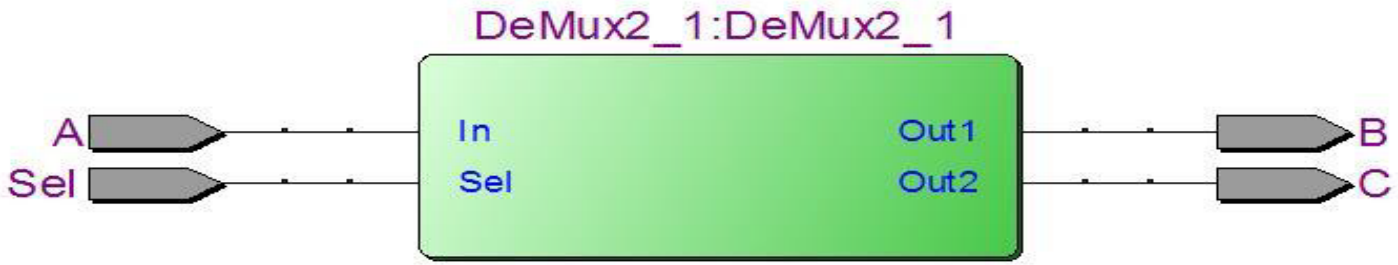
# 多工器( Multiplexier )




程式( 2 to 1 多工器 )：

```
module Mux2_1( In1, In2, Sel, Out );

    input   In1, In2, Sel;
    output  Out;

    wire    In1, In2, Sel;
    reg     Out;

    always @( In1, In2, Sel ) begin
        if( !Sel )
            Out <= In1;
        else
            Out <= In2;
    end

endmodule
```

# 解多工( DeMultiplexier )




程式( 1 to 2解多工)：

```
module DeMux2_1( In, Sel, Out1, Out2 );

    input   In, Sel;
    output  Out1, Out2;

    wire    In, Sel;
    reg     Out1, Out2;

    always @( In, Sel ) begin
        case( Sel )
            1'b0: Out1 <= In;
            1'b1: Out2 <= In;
        endcase
    end

endmodule
```

# 編碼器( Encoder )


程式( 3 to 8解碼器)：

```
module DeCoder( In, Out );

    input   [2:0] In;
    output  [7:0] Out;

    wire    [2:0] In;
    reg     [7:0] Out;

    always @( In ) begin
        case( In )
            3'b000:     Out <= 8'b0000_0001;
            3'b001:     Out <= 8'b0000_0010;
            3'b010:     Out <= 8'b0000_0100;
            3'b011:     Out <= 8'b0000_1000;
            3'b100:     Out <= 8'b0001_0000;
            3'b101:     Out <= 8'b0010_0000;
            3'b110:     Out <= 8'b0100_0000;
            3'b111:     Out <= 8'b1000_0000;
            default:    Out <= 8'bxxxx_xxxx;
        endcase
    end

endmodule
```

# 比較器( Comparator )


程式( 比較器 )：

```
module Comparator( In1, In2, Out );

    input   In1, In2;
    output  [2:0] Out;

    wire    In1, In2;
    reg     [2:0] Out;

    always @( In1, In2 ) begin
        Out[0] <= ( In1 > In2 );
        Out[1] <= ( In1 == In2 );
        Out[2] <= ( In1 < In2 );
    end

endmodule
```

# 算術邏輯運算單元( ALU )


程式( ALU )：

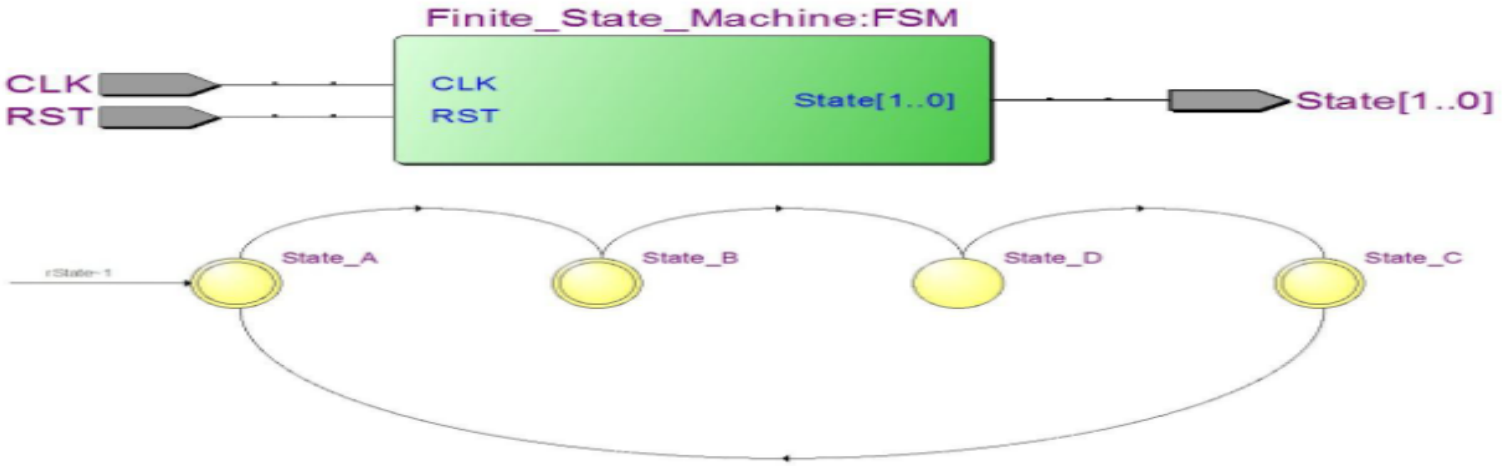
```
`define    ADD 2'b00
`define    SUB 2'b01
`define    AND 2'b10
`define    OR  2'b11

module ALU( Data_A, Data_B, OP_Code, Data_Out );

    parameter   Data_Size = 8;
    parameter   OP_Code_Size = 2;

    input   [Data_Size-1:0] Data_A, Data_B;
    input   [OP_Code_Size-1:0] OP_Code;
    output  [Data_Size-1:0] Data_Out;

    reg     [Data_Size-1:0] Data_Out;

    always @( Data_A, Data_B, OP_Code ) begin
        case( OP_Code )
            `ADD:       Data_Out <= Data_A + Data_B;
            `SUB:       Data_Out <= Data_A - Data_B;
            `AND:       Data_Out <= Data_A & Data_B;
            `OR:        Data_Out <= Data_A | Data_B;
            default:    Data_Out <= 0;
        endcase
    end

endmodule
```

# 計數器( Counter )


程式( 計數器 )：

```
module Counter( CLK, RST, Cnt_Num, Cnt_Data );

    parameter Cnt_Num_Size = 2;
    parameter Cnt_Data_Size = 16;

    input   CLK, RST;
    input   [Cnt_Num_Size-1:0] Cnt_Num;
    output  [Cnt_Data_Size-1:0] Cnt_Data;

    wire    CLK, RST;
    wire    [Cnt_Num_Size-1:0] Cnt_Num;
    reg     [Cnt_Data_Size-1:0] Cnt_Data;

    always @( posedge CLK, negedge RST ) begin
        if( !RST )
            Cnt_Data <= 0;
        else
            Cnt_Data <= Cnt_Data + Cnt_Num;
    end

endmodule
```

# 有限狀態機( Finite State Machine )




程式( FSM ) :

```
module Finite_State_Machine( CLK, RST, State );

     parameter    State_A = 2'b00, State_B = 2'b01,
                  State_C = 2'b10, State_D = 2'b11;

     input    CLK, RST;
     output   [1:0] State;
     reg      [1:0] State;

     always @( posedge CLK, negedge RST ) begin
          if( !RST )
               State = State_A;
          else
               case( rState )
                    State_A:     State <= State_B;
                    State_B:     State <= State_D;
                    State_C:     State <= State_A;
                    State_D:     State <= State_C;
                    default:     State <= State_A;
               endcase
     end

endmodule
```

# Reference


[1] Verilog硬體描述語言實務　出版社：全華, 作者：鄭光欽、周靜娟、黃孝祖、嚴培仁、吳明瑞

[2] Verilog硬體描述語言數位電路設計實務　出版社：儒林, 作者：鄭信源

[3] 真OO無雙之真亂舞書-由C語言學習Verilog的思維轉換

http://www.cnblogs.com/oomusou/archive/2008/06/17/c_verilog_mental_thinking.html

[4] 中原大學電機系Verilog HDL讀書會, 暑期教學講義 4th

https://legacy.gitbook.com/book/hom-wang/verilog-hdl/details

# Appendix

- Notepad++

https://notepad-plus-plus.org/zh/

- Iverilog

http://iverilog.icarus.com/

- GTK Wave

http://gtkwave.sourceforge.net/

- Tool 教學

https://darkblack01.blogspot.tw/2012/10/iverilog-gtkwave-notepadverilog.html