Outline

- Verilog語法架構
- Verilog四大模型
- 組合電路

Verilog語法架構

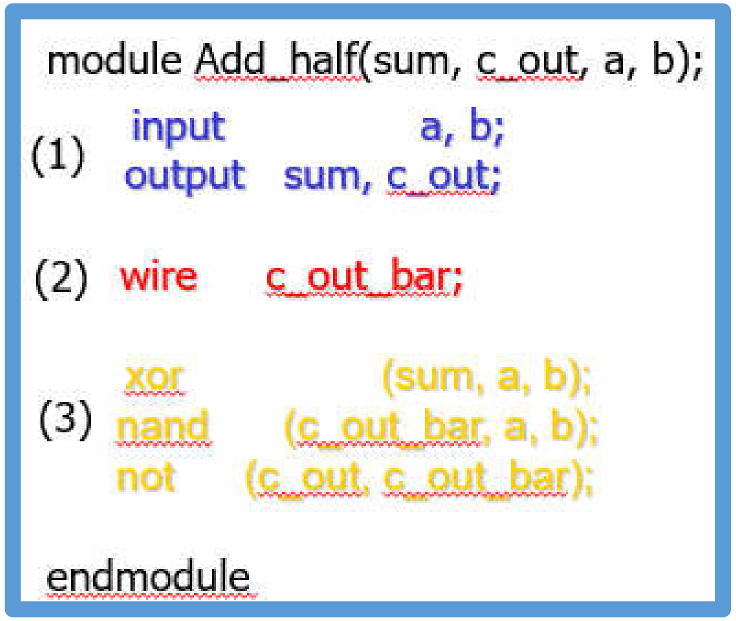
Verilog module (1/2)

```
module module_name(port_name);
        (1)Port declaration
        (2)Data type declaration
        (3)Module functionally or structure
endmodule
```

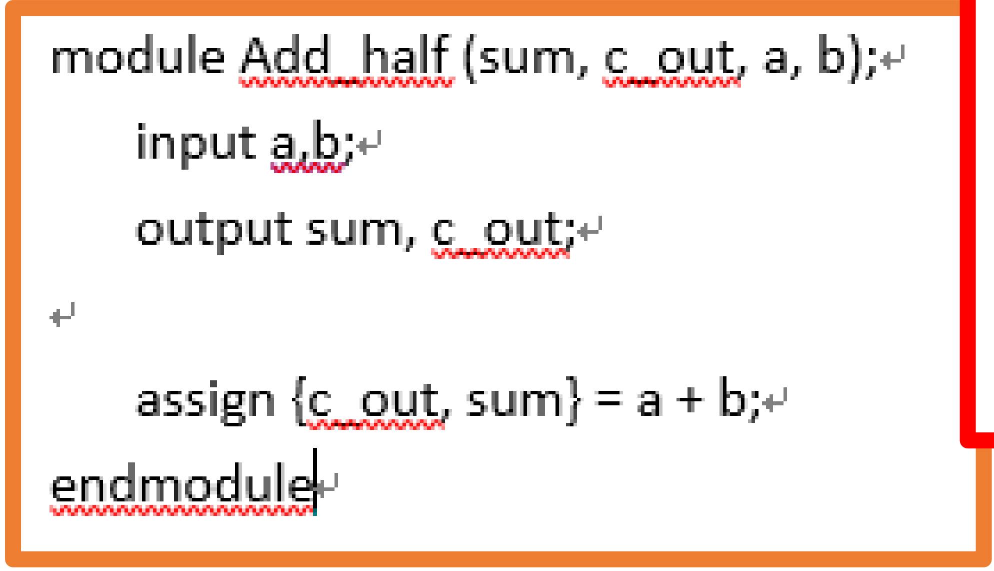
module_name:為電路的名稱。

port_name:為電路input和output所有的名稱。

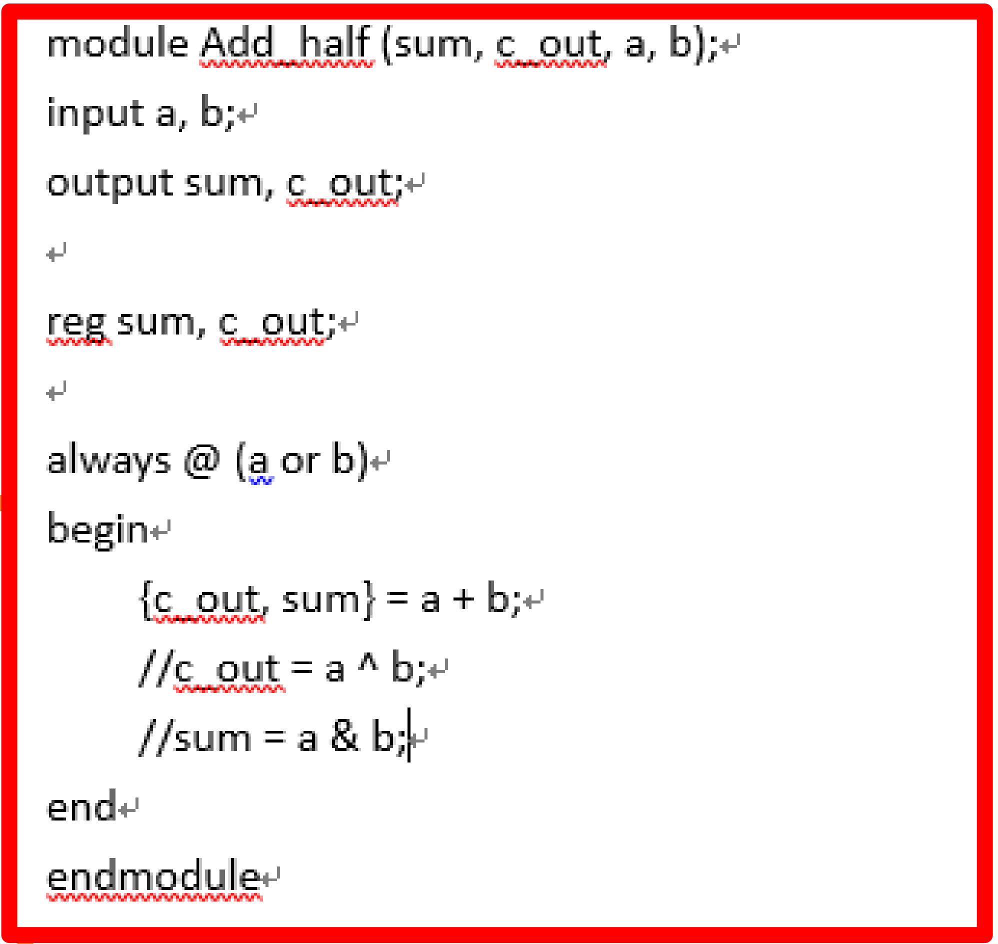
Verilog module (2/2)



```
module Add_half(sum, c_out, a, b);
(1)   input            a, b;
      output   sum, c_out;

(2)   wire     c_out_bar;

      xor            (sum, a, b);
(3)   nand     (c_out_bar, a, b);
      not     (c_out, c_out_bar);

endmodule
```



```
module Add_half (sum, c_out, a, b);
    input a,b;
    output sum, c_out;

    assign {c_out, sum} = a + b;
endmodule
```



```
module Add_half (sum, c_out, a, b);
input a, b;
output sum, c_out;

reg sum, c_out;

always @ (a or b)
begin
        {c_out, sum} = a + b;
        //c_out = a ^ b;
        //sum = a & b;
end
endmodule
```

關鍵字（keywords）

- 是一組特殊定義名稱。
- Keywords皆為小寫。

識別字(Keywords)

always	and	assign	Begin
case	casex	casez	default
else	end	endcase	endmodule
for	function	endfunction	if
initial	inout	input	output
integer	module	nand	negedge
nor	not	or	parameter
posedge	reg	wand	Wire

識別字(Identifiers)

- 給宣告的變數命名。
- 第一個字必須為字母，之後的字可以為字母、數字、底線(_)、錢號($)。
- 識別字的大小寫是有區別的。

Example:

```
input   out;
wire    register;
```

out和register為識別字

input和wire為關鍵字

Wire(1/2)

- wire(接線)特性
  - 是連接硬體元件之連接線。
  - 必須要被驅動才能改變它的數值。
  - 描述的關鍵字為wire。
  - 除非有被宣告成一個向量，否則接線是內定為一個位元的值，而且內定值是Z(high impedance，高阻抗)。
  - 邏輯準未有四狀態:高準位、低準位、高阻抗和不確定值。

Example:

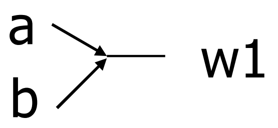
```
        input      b, c;         //宣告2個input，且命名為b和c
        wire       a = b & c;//宣告1條接線a，並指定他為b & c
```

Wire (2/2)

wire, wand(wire and), wor(wire or)

```
wire k;                // single-bit wire       (//為註解的意思)
wire [0:31] w1, w2;    // two 32-bit wires，numbered 0-31 from left to right
```

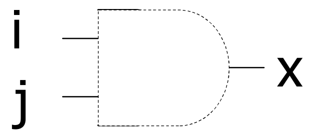
```
  wire w1;
   assign w1=a;
   assign w1=b; (error)
```



(error)

Method 1:

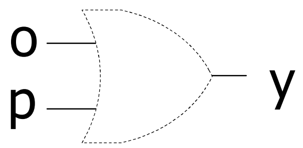
```
  wand x;
  assign x=j;     assign x=i;
```



(ok)

Method 2:

```
  wor y;
  assign y=o;   assign y=p;
```

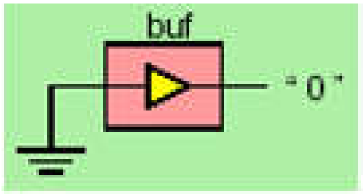


(ok)

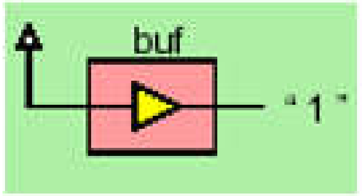
四種數值準位(1/2)

數值位準	實際電路狀態
0	邏輯0、zero、false、low、ground
1	邏輯1、one、true、High、power
x	不確定值(Unknown Value)
z	高阻抗(High Impedance)、浮接狀態(Floating State)、Disabled Driver(Unknow)

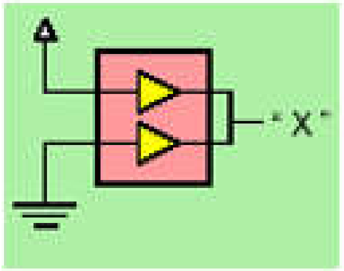
四種數值準位(2/2)

- 

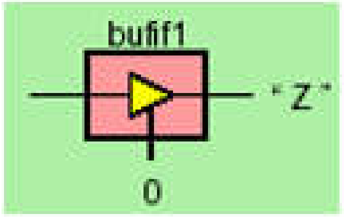
0: logic 0 / false

- 

1: logic 1 / true

- 

X: unknown logic value

- 

Z: high-impedance

reg(暫存器)

- 暫存器的功能很像變數，可以直接給定一個數值，主要的功能在於保持住電路中的某個值
- 暫存器描述的關鍵字為reg
- 除非有被宣告成一個向量(vector)，否則內定都為一個位元值，而且內定值是x(don’ t case)

向量表示法(1/2)

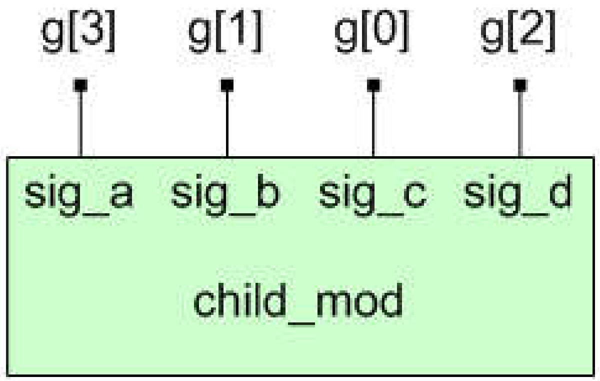
- wire和reg皆可定義為向量，若無定義位元長度，則視為被宣告的變數期位元長度為1，也就是一個位元。
- 向量是一個多位元的元件。
- 向量可定義為[Hight:Low]或[Low:Hight]。其中，”:”左邊為最高有效位元，”:”右邊為最低有效位元。

向量表示法(2/2)

- wire even; // 宣告變數名even為1條接線
- wire [15: 0] bus; //宣告變數名bus為1組由左至右編號為15-0的16條接線
- reg bit; //宣告變數名稱bit為1個位元的reg
- reg [31:0] Result //宣告變數名稱Result為1個32個reg所組成之32位元暫存器，其bit編號由左至右為31-0。
- reg [7:0] RegFile [1023:0] //1024個Register，每個Register為8bits

Port Mapping by Position Association

```
module parent_mod;
   wire        [3:0] g;

   child_mod G1(g[3], g[1], g[0], g[2]);
endmodule

module child_mod(sig_a, sig_b, sig_c, sig_d);
   input       sig_c, sig_d;
   output      sig_a, sig_b;

    module description
endmodule
```

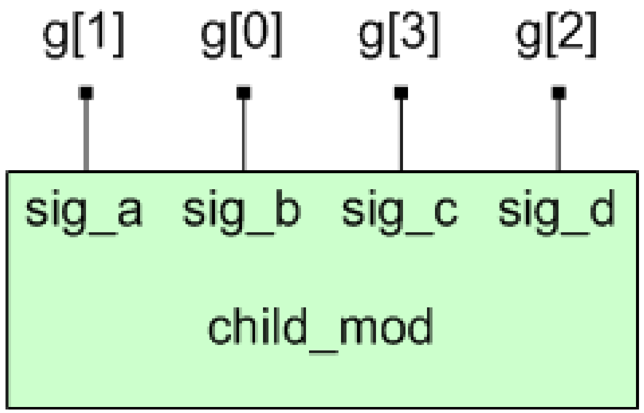


in-order port mapping

Post Mapping by Name Association

```
module parent_mod;
   wire        [3:0] g;

   child_mod G1(.sig_c(g[3]), .sig_d(g[2]), .sig_b(g[0]),
                     .sig_a(g[1]));
endmodule

module child_mod(sig_a, sig_b, sig_c, sig_d);
   input       sig_c, sig_d;
   output      sig_a, sig_b;

   module description
endmodule
```

naming port mapping



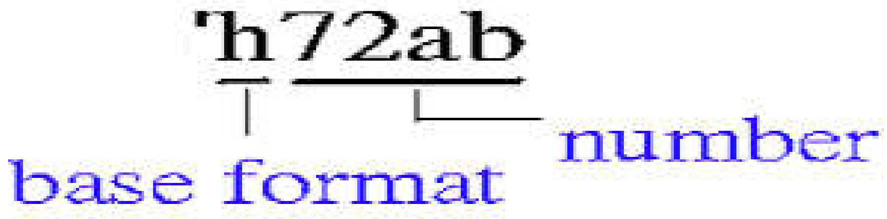
Number(1/2)

- Numbers are integer or real constants.
  - Integer constants are written as
  - <size>"<base format><number>
- Real number can be represented in decimal or scientific format.
- A number may be **sized** or **unsized**

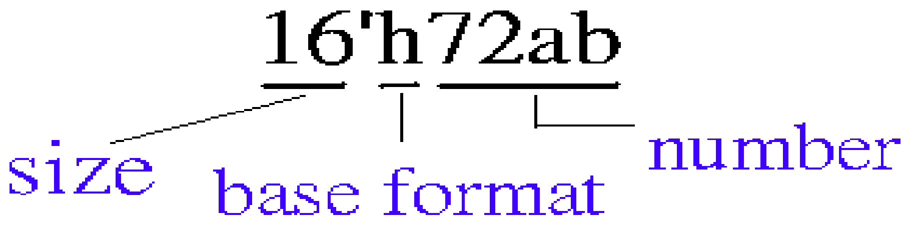
Number(2/2)

- The base format indicates the type of number
  - Decimal (d or D)
  - Hex (h or H)
  - Octal (o or O)
  - Binary (b or B)

ex: unsize



size



Parameter

- Parameter declaration
  - parameter<range>?<list_of_assignments>
- You can use a parameter anywhere that you can use a literal.

Ex1:

```
module mod (in,out);
.................................
parameter    m1    = 8 ;
..................................
wire        [m1:0]     w1;
endmodule
```

Ex2:

```
 modul mod (in,out);
…………
parameter [1:0] m1 = 2'b00 ;
…………
endmodule
```

運算子(1/3)

Name	Operator
bit-select or part-select	[ ]
parenthesis	( )
Arithmetic Operators	
multiplication	*
division	/
addition	+
subtraction	-
modulus	%
Sign Operators	
identity	+
negation	-

運算子(2/3)

Name	Operator
Relational Operators	
less than	<
less than or equal to	<=
greater than	>
greater than or equal to	>=
Equality Operators	
logic equality	==
logic inequality	!=
case equality	===
case inequality	!==
Logical Comparison Operators	
NOT	!
AND	&&
OR	||
Logical Bit-Wise Operators	
unary negation NOT	~
binary AND	&
binary OR	|
binary XOR	^
binary XNOR	^~ or ~^

運算子(3/3)

Name	Operator
Shift Operators	
logical shift left	<<
logical shift right	>>
Concatenation & Replication Operators	
concatenation	{ }
replication	{{ }}
Reduction Operators	
AND	&
OR	|
NAND	~&
NOR	~|
XOR	^
XNOR	^~ or ~^
Conditional Operator	
conditional	?:

算術(Arithmetic)運算子

```
module ARITHMETIC(A, B, Y1, Y2, Y3);
    input [2:0]        A, B;
    output [3:0]       Y1;
    output [4:0]       Y3;
    output [3:0]       Y2;
    reg [3:0] Y1;
    reg [4:0] Y3;
    reg [3:0] Y2;

 always @(A or B)
 begin
    Y1 = A + B;
    Y2 = A - B;
    Y3 = A * B;  ┐
    Y4 = A / B;  │(不建議)
    Y5 = A % B;  ┘
 end
endmodule
```

Arithmetic operators:
(1) +
(2) -
(3) *
(4) /   (++)non-syn.
(5) % (++)non-syn.

A[2:0]	000	001	101	110
B[2:0]	101	111	100	001
Y1[3:0]	0101	1000	1001	0111
Y2[3:0]	1011 (-5)	1010(-6)	0001(+1)	0101(+5)
Y3[4:0]	00000	00111	10100	00110

標誌(Sign)運算子

```
module SIGN(A, B, Y1, Y2);
   input [2:0]        A;
   input [2:0]        B;
   output [3:0]       Y1;
   output [3:0]       Y2;
   reg [3:0]          Y1;
   reg [3:0]          Y2;

 always @(A or B)
 begin
   Y1 = A + -B;
   Y2 = -A + B;
 end
endmodule
```

Sign operators:
(1)+
(2)-

A[2:0]	000	001	100	101	110	111	000
B[2:0]	110	101	100	010	000	001	101
Y1[3:0]	1010 (-6)	1100 (-4)	0000 (0)	0011 (3)	0110 (6)	0110 (6)	1011 (-5)
Y2[3:0]	0110 (6)	0100 (4)	0000 (0)	1101 (-3)	1010 (-6)	1010 (-6)	0101 (5)

關係(Relational)運算子

```
module RELATIONAL_OPERATORS(A, B, Y);
    input [2:0]        A;
    input [2:0]        B;
    output [3:0]       Y;
    reg [3:0] Y;

 always @(A or B)
 begin
    Y[0] = A > B;
    Y[1] = A >= B;
    Y[2] = A < B;
    if ( A <= B)
      Y[3] = 1;
    else
      Y[3] = 0;
 end
endmodule
```

Relational operators:
(1) <
(2) <=
(3) >=
(4) >

A[2:0]	1	2	3	4	5	6	7	3	0
B[2:0]	3	5	6	7	2	1	0	3	6
Y[3]	1	1	1	1	0	0	0	1	1
Y[2]	1	1	1	1	0	0	0	0	1
Y[1]	0	0	0	0	1	1	1	1	0
Y[0]	0	0	0	0	1	1	1	0	0

平等(Equality)與 不平等(Inequality)運算子

```
module EQUALITY_OPERATORS(A, B, Y);
    input [2:0]        A;
    input [2:0]        B;
    output [4:0]       Y;
    reg [4:0] Y;

 always @(A or B)
 begin
    Y[0] = A != B;
    Y[1] = A == B;
    if ( A == B)
        Y[4:2] = A;
    else
        Y[4:2] = B;
end

endmodule
```

Equality operators:

(1)	==	(等於)
(2)	!=	(不等於)
(3)	===	(++) non-syn.
(4)	!==	(++) non-syn.

When comparison is true, the result is 1

When comparison is false, the result is 0

A[2:0]	0	7	4	1	0
B[2:0]	3	7	3	4	5
Y[0]	1	0	1	1	1
Y[1]	0	1	0	0	0
Y[4:2]	3	7	3	4	5

邏輯比較(Logical Comparison) 運算子

```
module COMPARISON(A, B, C1,C2,C3);
    input   [2:0]   A,B;
    output  [2:0]   C1, C2, C3;
    reg     [2:0]   C1, C2, C3;

 always @(A or B)
 begin
  if (( A == 1) && ( B>2 ) )
      C1= 2'b 00;   else   C1= 2'b 11;
  if (( A>3) || ( B>1 ) )
      C2= 2'b 00;   else   C2= 2'b 11;
  if (!A)
      C3= 2'b 00;   else   C3= 2'b 11;
  end
endmodule
```

Logic comparison operators:

(1) ! (NOT)

(2) && (AND)

(3) || (OR)

A[2:0]	000	001	010	101
B[2:0]	000	010	001	010
C1[2:0]	011	011	011	011
C2[2:0]	011	000	011	000
C3[2:0]	000	011	011	011

位元邏輯(Logical Bit-Wise) 運算子

```
module BITWISE(A, B, Y1, Y2, Y3, Y4, Y5);
      input [3:0]    A;
      input [3:0]    B;
      output [3:0]  Y1, Y2, Y3, Y4, Y5;

      reg [3:0]       Y1, Y2, Y3, Y4, Y5;

  always @(A or B)
  begin
   Y1 =   ~A;
   Y2 =  A & B;
   Y3 =  A | B ;
   Y4 =  A ^ B;
   Y5 =  A ^ ~ B;
  end
endmodule
```

Logic bit-wise operators:

(1) ~	(unary NOT)
(2) &	(binary AND)
(3) |	(binary OR)
(4) ^	(binary XOR)
(5) ^~ or ~^	(binary XNOR)

```
   0110
&  0011
   0010
```

```
   0110
|  0011
   0111
```

A[3:0]	0000	0001	0010	0110
B[3:0]	0000	0000	0001	0011
Y1[3:0]	1111	1110	1101	1001
Y2[3:0]	0000	0000	0000	0010
Y3[3:0]	0000	0001	0011	0111
Y4[3:0]	0000	0001	0011	0101
Y5[3:0]	1111	1110	1100	1010

移位(Shift)運算子

```
module SHIFT(A, Y1, Y2);
    input [7:0]        A;
    output [7:0]       Y1;
    output [7:0]       Y2;

    reg [7:0]          Y1;
    reg [7:0]          Y2;

    parameter          B=3;

 always @(A)
 begin
    Y1 = A << B;
    Y2 = A >> 2;

end

endmodule
```

Shift operators:

(1) <<   (left shift)

(2) >>   (right shift)

A[7:0]	00000000	00000001	00000011	00000100
Y1[7:0]	00000000	00001000	00011000	00100000
Y2[7:0]	00000000	00000000	00000000	00000001

連結(Concatenation)和 複製(Replication)運算子

```
module CONCATENATION  (A, B, Y);
   input [2:0]        A;
   input [2:0]        B;
   output [14:0]      Y;

   reg [14:0]         Y;

   parameter          C=3'b011;

 always @(A)
 begin
   Y = { B, A, { 2 { C } }, 3'b 001};
end
endmodule
```

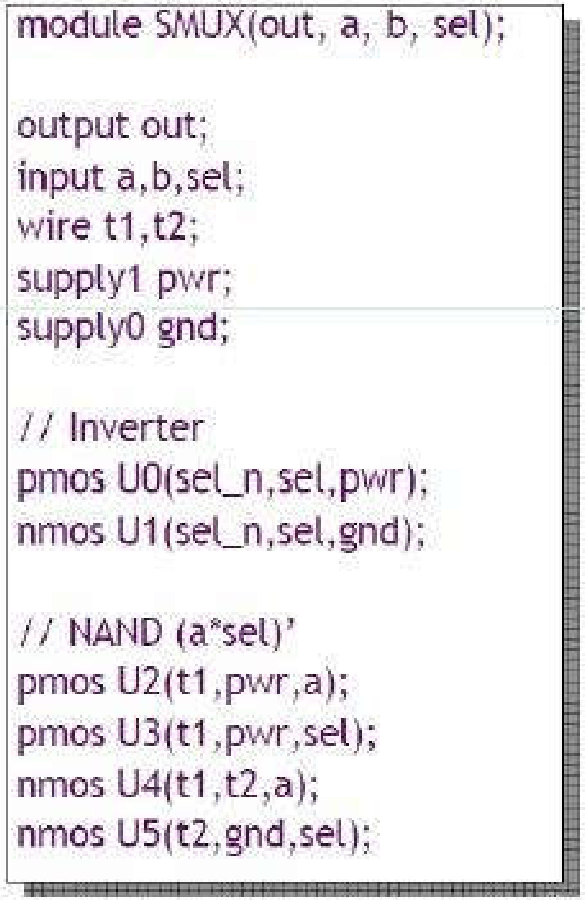
Concatenation	{}
Replication	{{}}

A[2:0]	000	001	010	011
B[2:0]	000	010	100	110
Y[14:0]	000 000 011 011 001	010 001 011 011 001	100 010 011 011 001	110 011 011 011 001

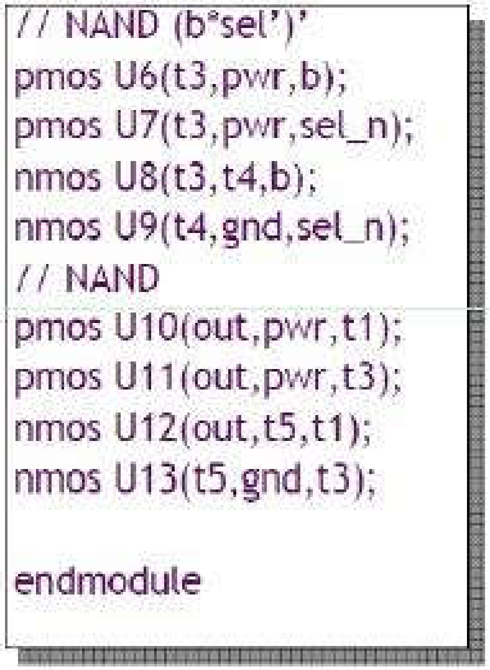
Verilog四大模型

開關階層(switch level)

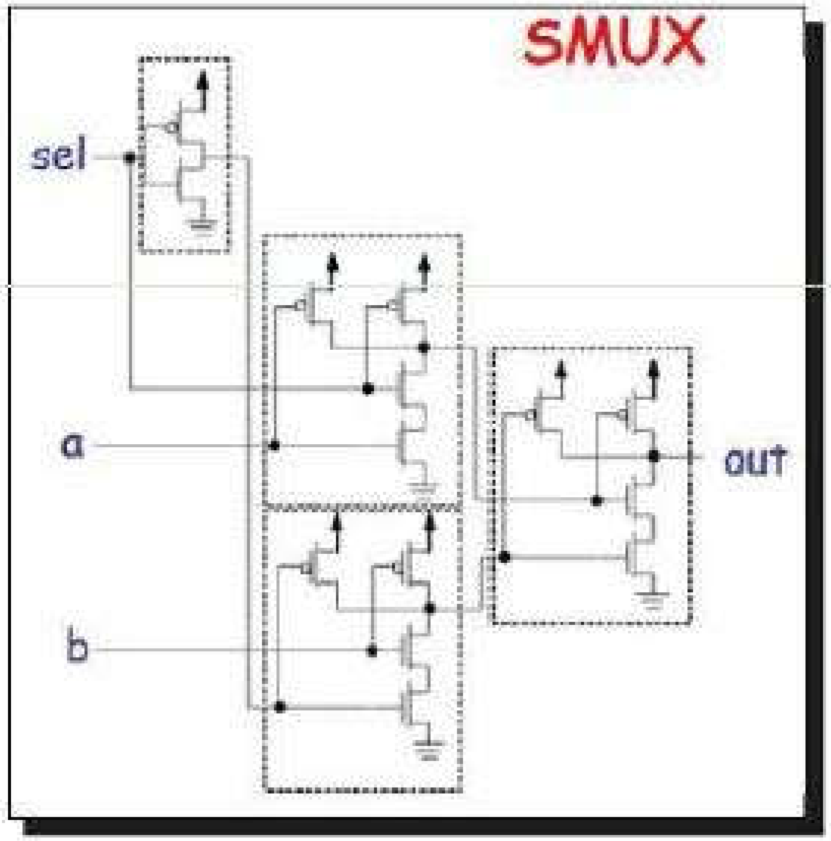
- 為Verilog最低階的層級，這層級中設計者必須知道電晶體元件的特性。



```
module SMUX(out, a, b, sel);

output out;
input a,b,sel;
wire t1,t2;
supply1 pwr;
supply0 gnd;

// Inverter
pmos U0(sel_n,sel,pwr);
nmos U1(sel_n,sel,gnd);

// NAND (a*sel)'
pmos U2(t1,pwr,a);
pmos U3(t1,pwr,sel);
nmos U4(t1,t2,a);
nmos U5(t2,gnd,sel);
```

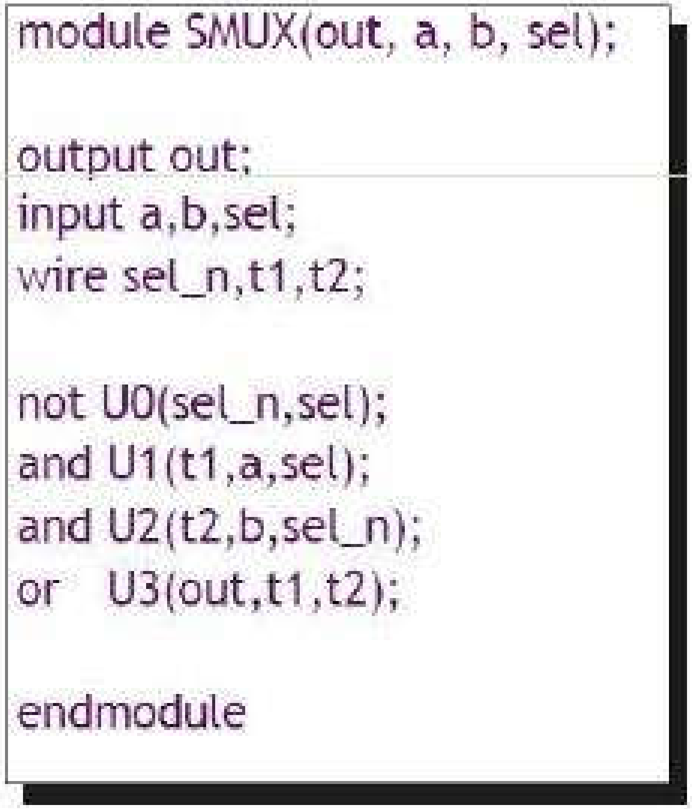


```
// NAND (b*sel')'
pmos U6(t3,pwr,b);
pmos U7(t3,pwr,sel_n);
nmos U8(t3,t4,b);
nmos U9(t4,gnd,sel_n);
// NAND
pmos U10(out,pwr,t1);
pmos U11(out,pwr,t3);
nmos U12(out,t5,t1);
nmos U13(t5,gnd,t3);

endmodule
```

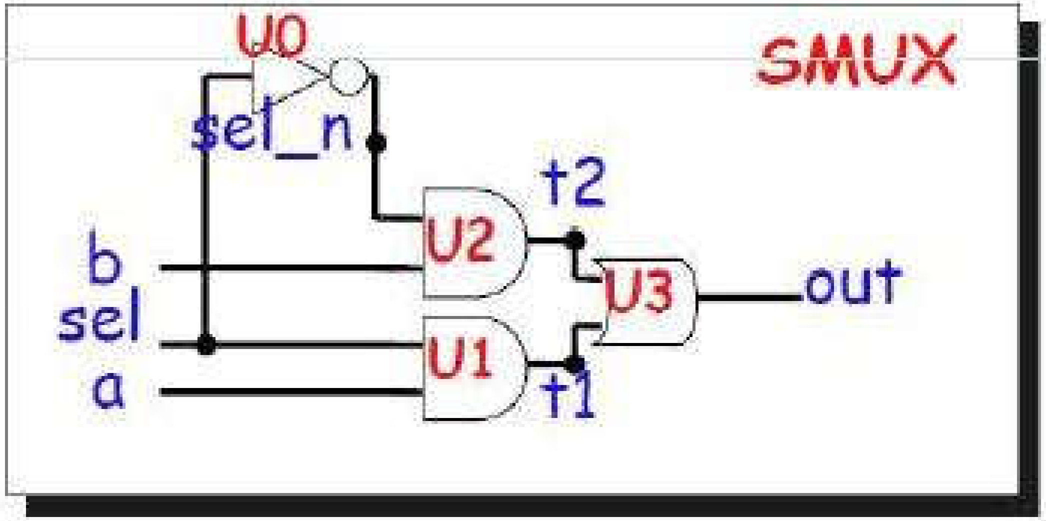


邏輯閘階層(Gate Level)

- 由邏輯閘所連接而成。

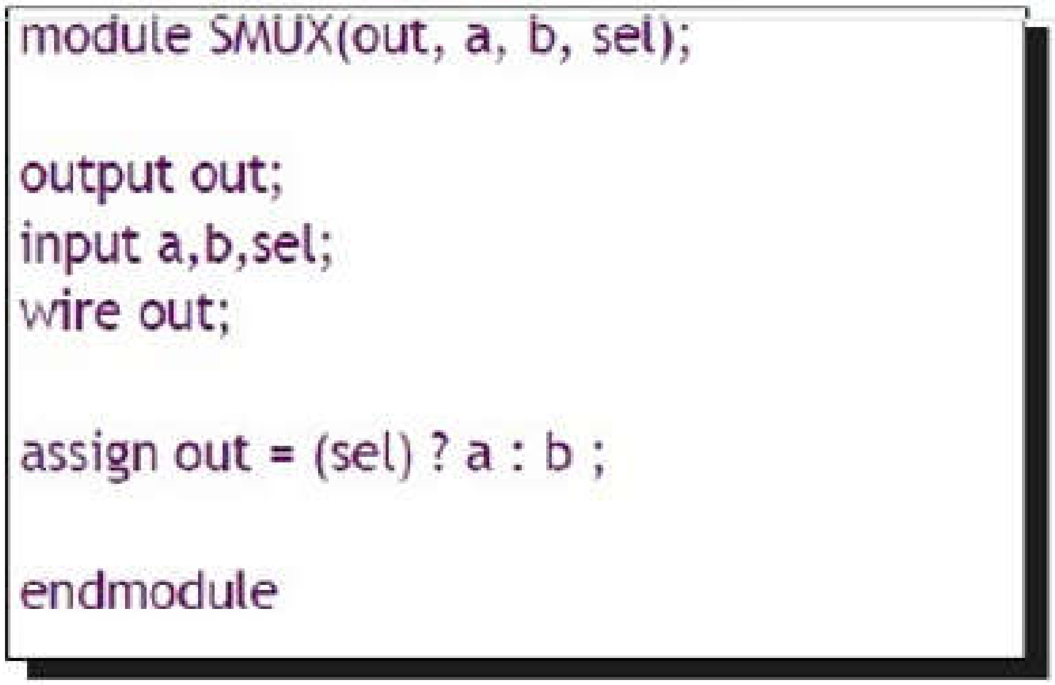


```
module SMUX(out, a, b, sel);

output out;
input a,b,sel;
wire sel_n,t1,t2;

not U0(sel_n,sel);
and U1(t1,a,sel);
and U2(t2,b,sel_n);
or   U3(out,t1,t2);

endmodule
```

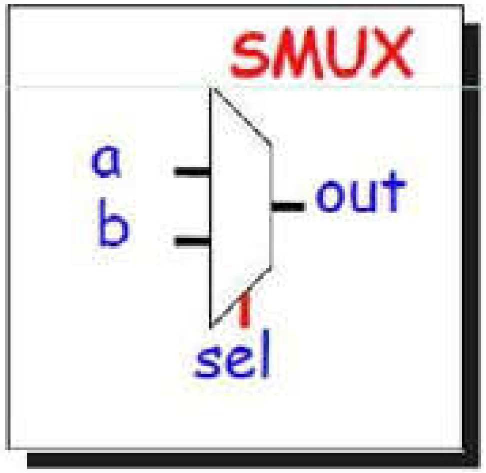


資料處理層級(Data Flow Level)

- 須明確的指明資料處理的方法。

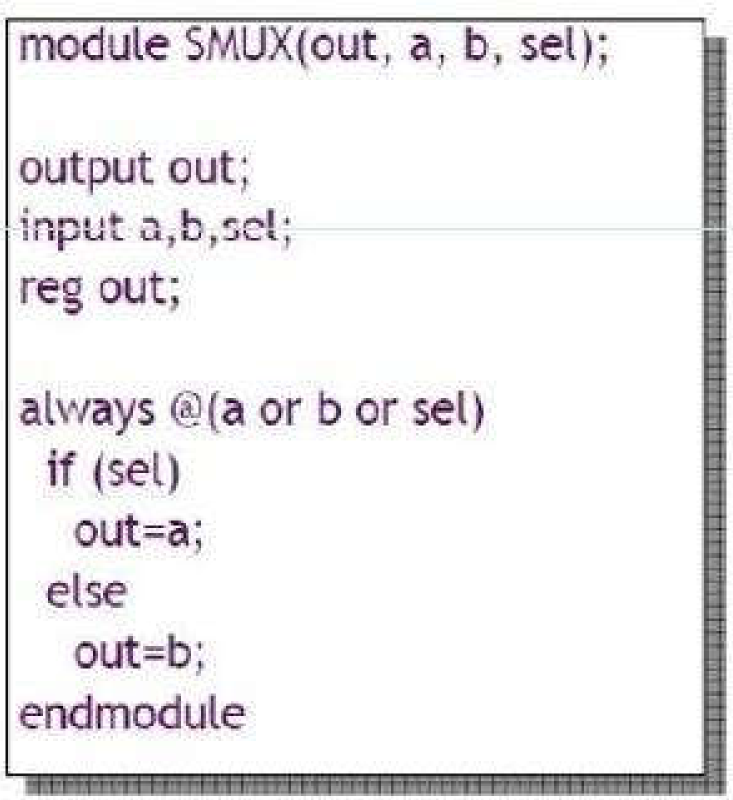


```
module SMUX(out, a, b, sel);

output out;
input a,b,sel;
wire out;

assign out = (sel) ? a : b ;

endmodule
```

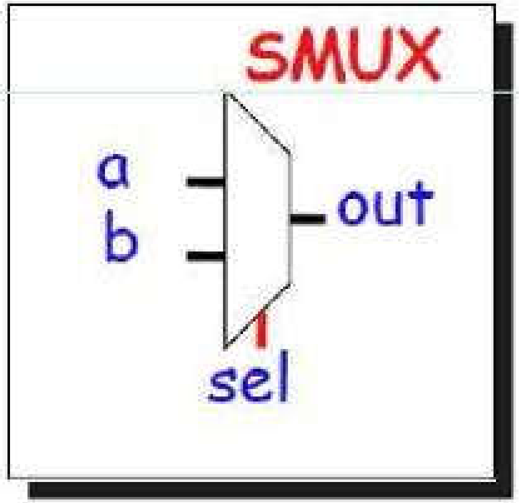


行為描述(Behavioral)模型

- Verilog HDL中的最高層級，只需考慮電路的功能，不須考慮硬體架構。



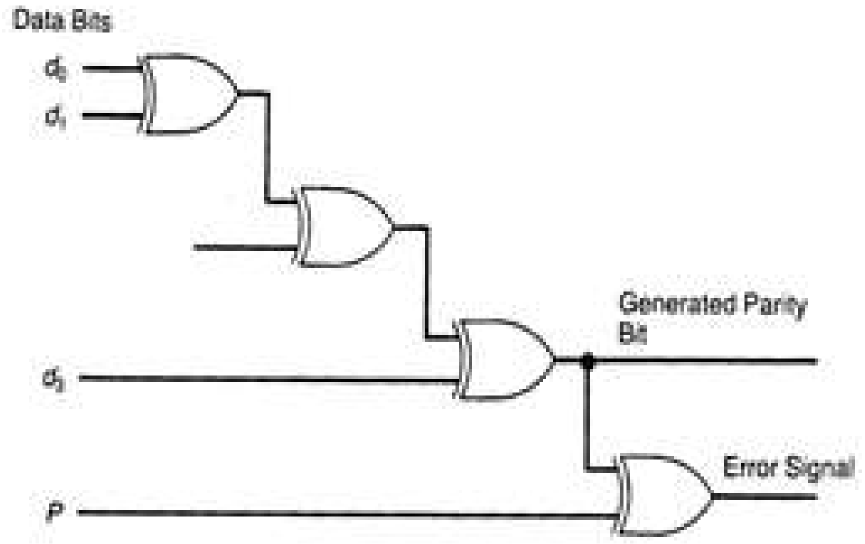
```
module SMUX(out, a, b, sel);

output out;
input a,b,sel;
reg out;

always @(a or b or sel)
  if (sel)
    out=a;
  else
    out=b;
endmodule
```

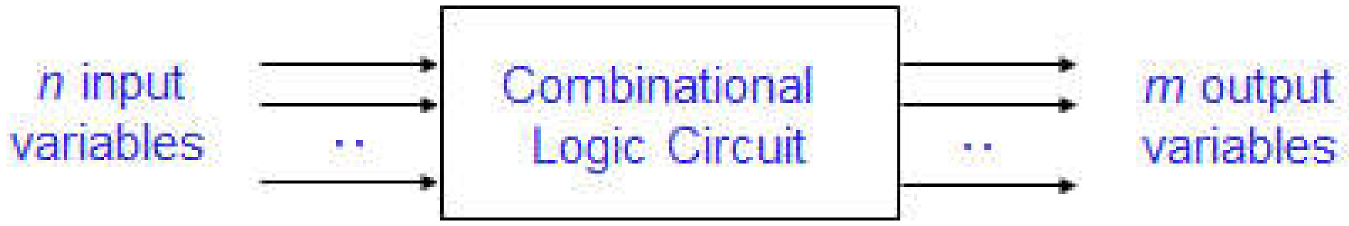


組合電路

組合電路

- 電路的output會因input不同的值改變，而不需考慮前一狀態的值。





組合邏輯設計步驟

- 設定input/output
- 劃出真值表
- 簡化布林代數
- 設計電路

組合電路的HDL模型

- Gate-level modeling使用原始閘以及使用者定義的模組進行例示 (instantiation)。
- Dataflow modeling使用具有關鍵字assign之連續指定敘述。
- Behavioral modeling使用具有關鍵字assign之連續指定敘述。

always敘述(1/2)

- always敘述用來描述一個重複工作的數位邏輯電路
- 使用always區塊時，其中指定的資料必須是reg型態。

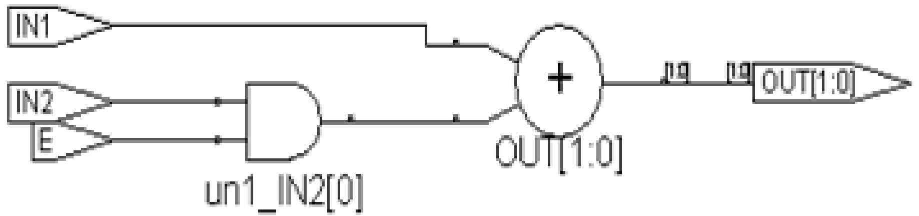
```
always @ (事件1 or 事件2)
begin
        敘述……
end
```

always敘述(2/2)

- Example
    - always @ (A or B)　　　　　　　//A或B事件觸發
    - always @ (posedge clk)　　　　//clk上緣觸發
    - always @ (negedge clk)　　　　//clk下緣觸發

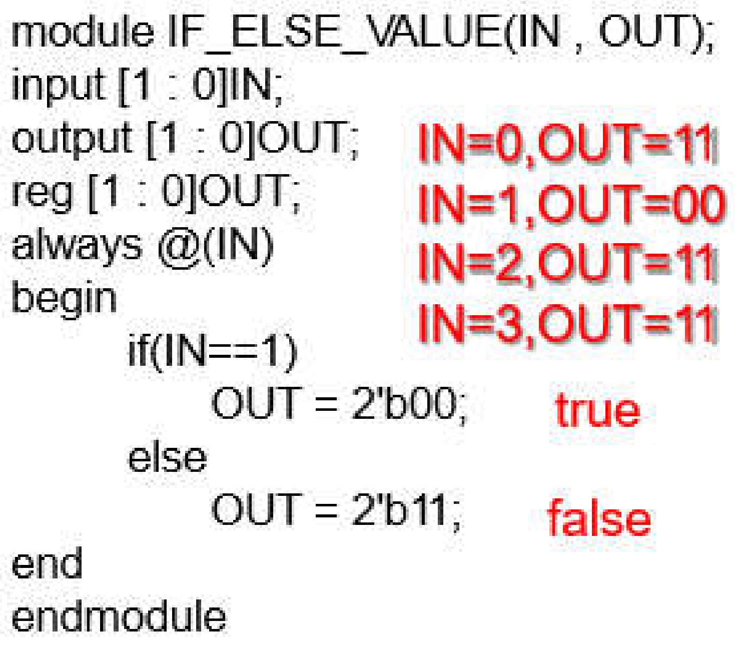
If-else敘述(1/3)

```
module IF_ELSE(IN1 , IN2 , E , OUT);
input IN1, IN2, E;
output [1 : 0]  OUT;   reg [1 : 0]OUT;
always @(IN1 or IN2 or E)
begin
        if(E == 1)
              OUT = IN1 + IN2;
        else
              OUT = IN1;        end
endmodule
```



```
If (expression)
      begin
      . . . statements . . .
      end
[else
      begin
      . . . statements . . .
      end]
```

If-else敘述(2/3)

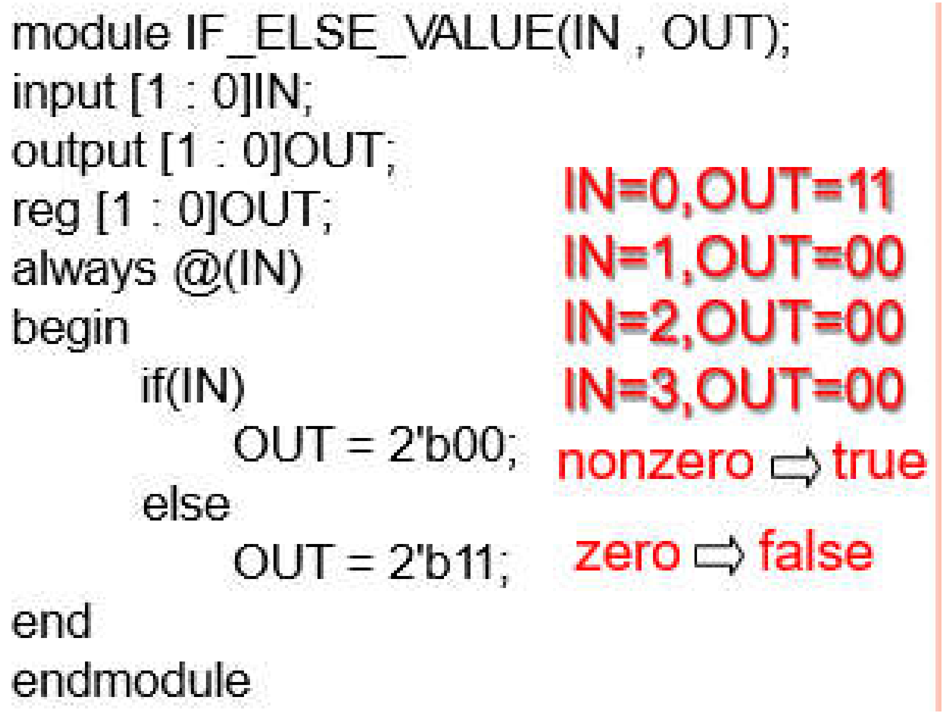


```
module IF_ELSE_VALUE(IN , OUT);
input [1 : 0]IN;
output [1 : 0]OUT;
reg [1 : 0]OUT;
always @(IN)
begin
        if(IN==1)
              OUT = 2'b00;
        else
              OUT = 2'b11;
end
endmodule
```

IN=0,OUT=11
IN=1,OUT=00
IN=2,OUT=11
IN=3,OUT=11

true

false



```
module IF_ELSE_VALUE(IN , OUT);
input [1 : 0]IN;
output [1 : 0]OUT;
reg [1 : 0]OUT;
always @(IN)
begin
        if(IN)
              OUT = 2'b00;
        else
              OUT = 2'b11;
end
endmodule
```

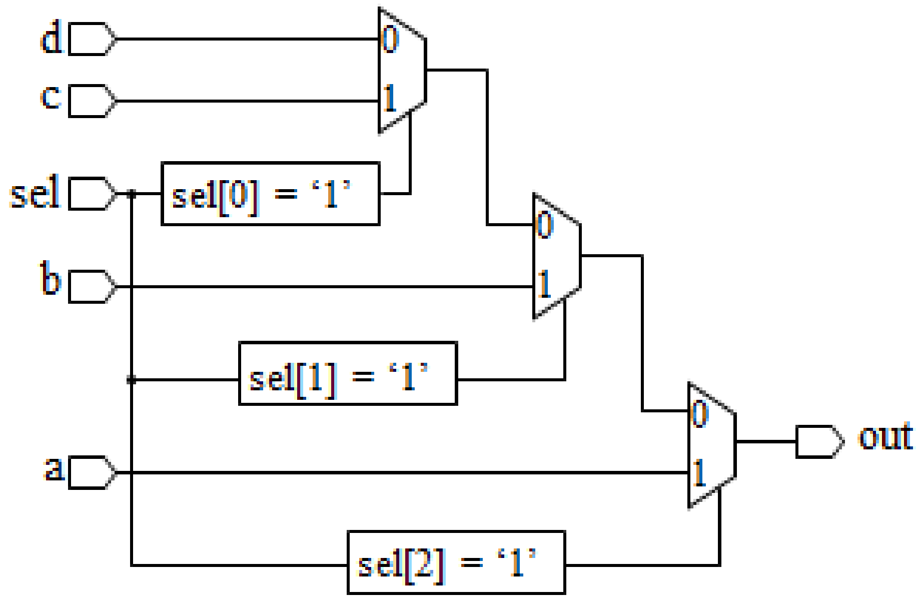
IN=0,OUT=11
IN=1,OUT=00
IN=2,OUT=00
IN=3,OUT=00

nonzero ⇨ true

zero ⇨ false

If-else敘述(3/3)

```
always @(sel or a or b or c or d)
    if (sel[2] == 1'b1)
        out = a;       //sel=1XX
    else if (sel[1] == 1'b1)
        out = b;       //sel=01X
    else if (sel[0] == 1'b1)
        out = c;       //sel=001
    else
        out = d;       //sel=000
```



Case敘述

```
module FULL_CASE(IN , OUT);
input  [1 : 0]IN;output [3 : 0] OUT;
reg [3 : 0]OUT;
always @(IN)
begin
        case(IN)
        2'b00: OUT = 4'b0001;
        2'b01: OUT = 4'b0010;
        2'b10: OUT = 4'b0100;
        2'b11: OUT = 4'b1000;
        endcase
end
endmodule
```

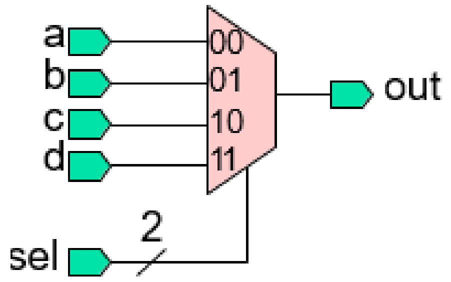
```
module FULL_CASE(IN , OUT);
input  [1 : 0]IN;output [3 : 0] OUT;
reg [3 : 0]OUT;
always @(IN)
begin
        case(IN)
        2'b00: OUT = 4'b0001;
        2'b01: OUT = 4'b0010;
        2'b10: OUT = 4'b0100;
        endcase
end
endmodule
```

```
case (expression)

    case_item 1:
        begin
        . statements . .
        end
    case_item 2:
        begin
        . statements. .
        end
    . . . . .
    default:
        begin
        . statements. .
        end
endcase
```

Case vs if-else敘述

```
always @ (sel or a or b or c or d)
begin
    case (sel)
        2’b00: out = a;
        2’b01: out = b;
        2’b10: out = c;
        default : out = d;
    endcase
end
```

```
always @(sel or a or b or c or d)
    if (sel == 2’b00)
        out = a;
    else if (sel == 2’b01)
        out = b;
    else if (sel == 2’b10)
        out = c;
    else
        out = d;
```



Loop

for loop are "unrolled", and then synthesized.

```
integer i;
always @(a r b)
begin
    for (i = 0; i < 6; i = i+1)
        example[i] <= a[i] & b[5-i];
end
```

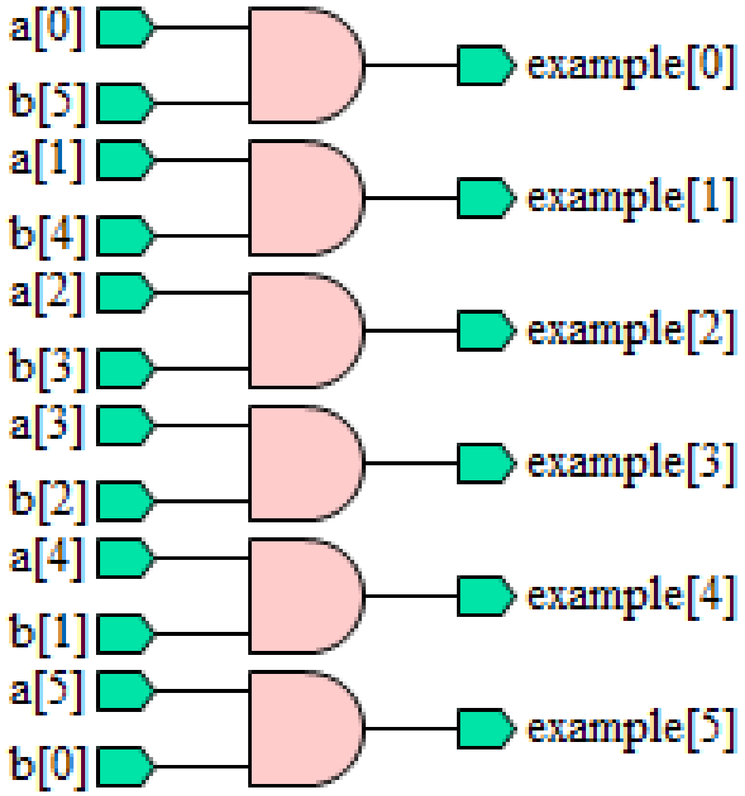
Verilog for loop

↓

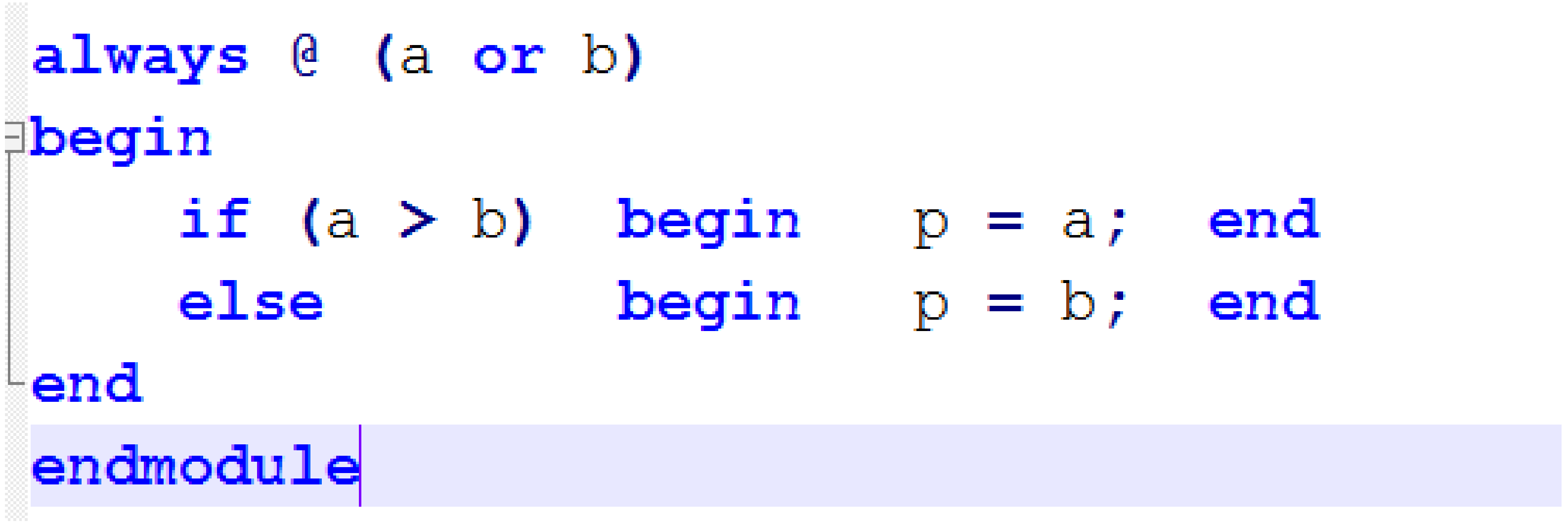
```
example [0] <= a[0] and b[5];
example [1] <= a[1] and b[4];
example [2] <= a[2] and b[3];
example [3] <= a[3] and b[2];
example [4] <= a[4] and b[1];
example [5] <= a[5] and b[0];
```

for loop unrolled

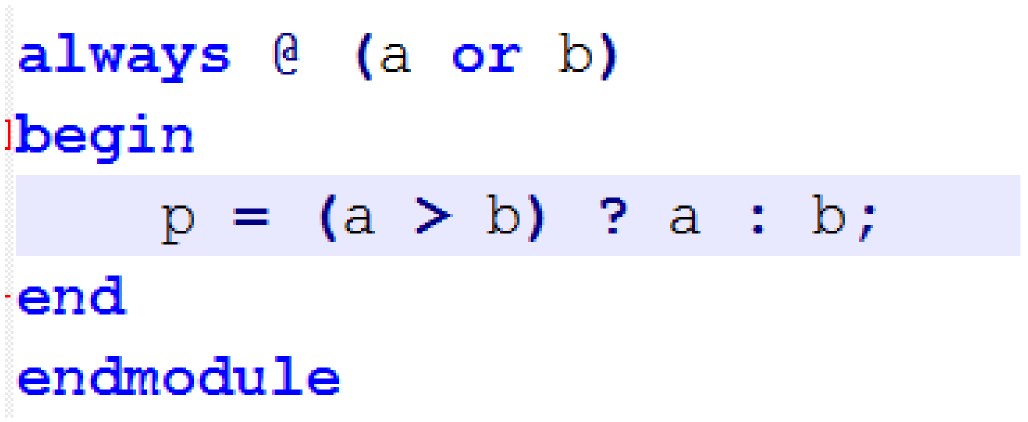
→



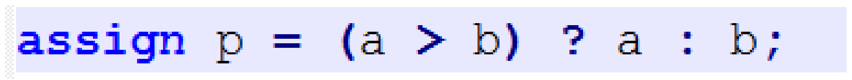
條件運算子



```
always @ (a or b)
begin
    if (a > b)  begin   p = a;  end
    else        begin   p = b;  end
end
endmodule
```



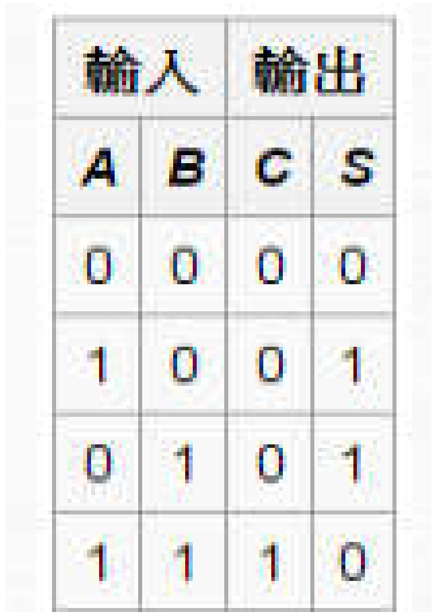
```
always @ (a or b)
begin
    p = (a > b) ? a : b;
end
endmodule
```



```
assign p = (a > b) ? a : b;
```

半加器

- 設計一個半加法器
- 設定輸入和輸出繳
  - Input: A, B
  - Output: C, S
- 真值表



輸入		輸出	
A	*B*	*C*	*S*
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

布林代數表示

$$S = A'B + AB'$$

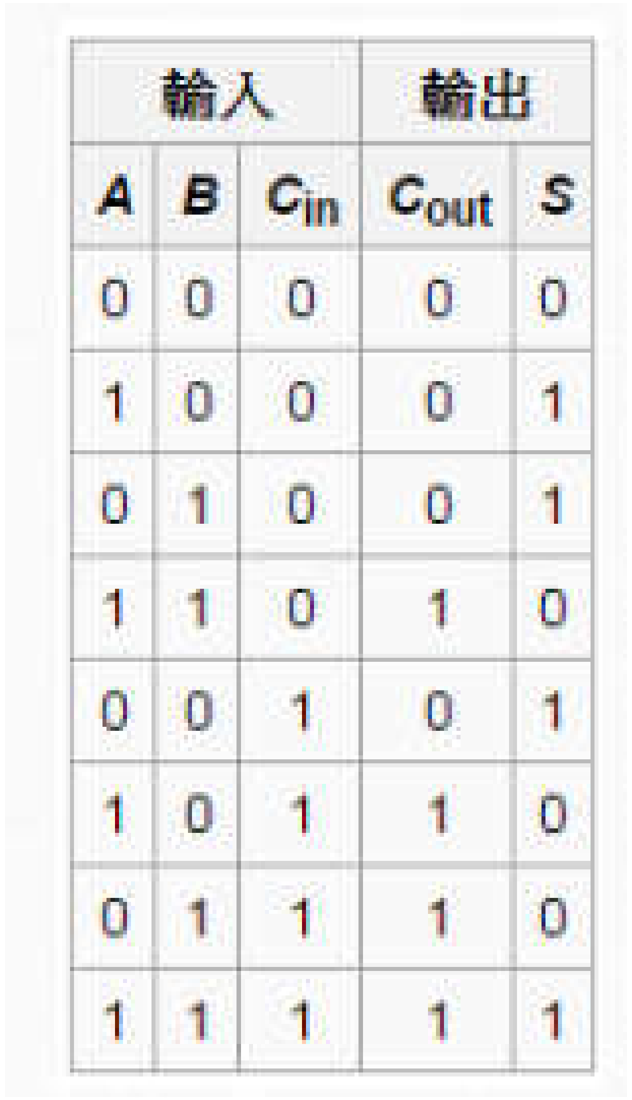
$$C = AB$$

程式碼

```
module HA (s, c, x, y);
input x, y;
output s, c;

    assign s = x ^ y;
    assign c = x & y;
endmodule
```

全加器(1/3)

- 設定input/output
  - Input A, B, Cin
  - Output S Cout
- 真值表



輸入			輸出	
A	B	C_{in}	C_{out}	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1

全加器(2/3)

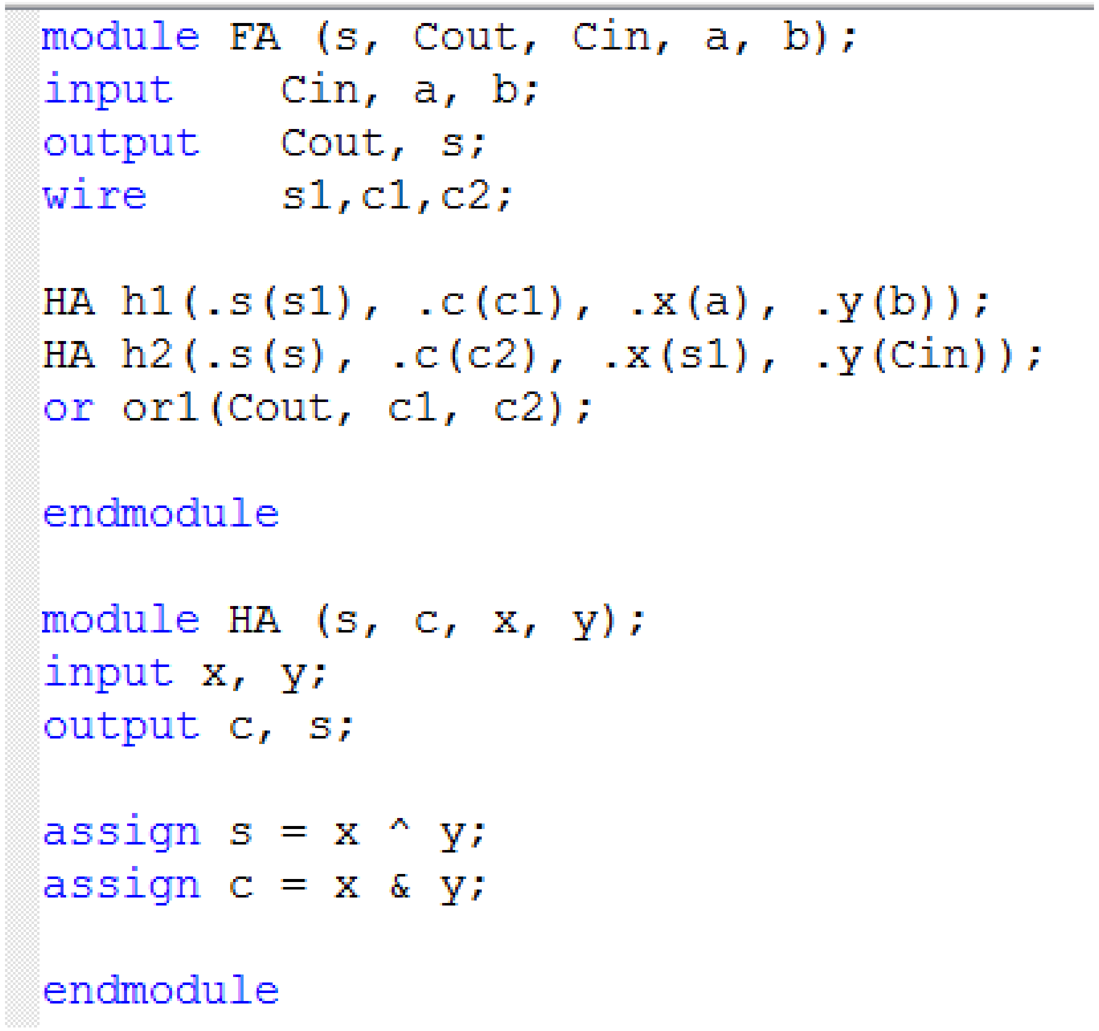
- 化簡布林代數
  - $S = A\ xor\ B\ xor\ C$
  - $Cout = AB + BC + CA = A(B + C) + BC$

程式碼(作法一)

```
        module FA (s, Cout, cin, a,b);
        input a,b, Cin;
        output s, Cout;

                assign s = a ^ b ^ Cin;
                assign Cout = ab + bc + ca;
        module
```

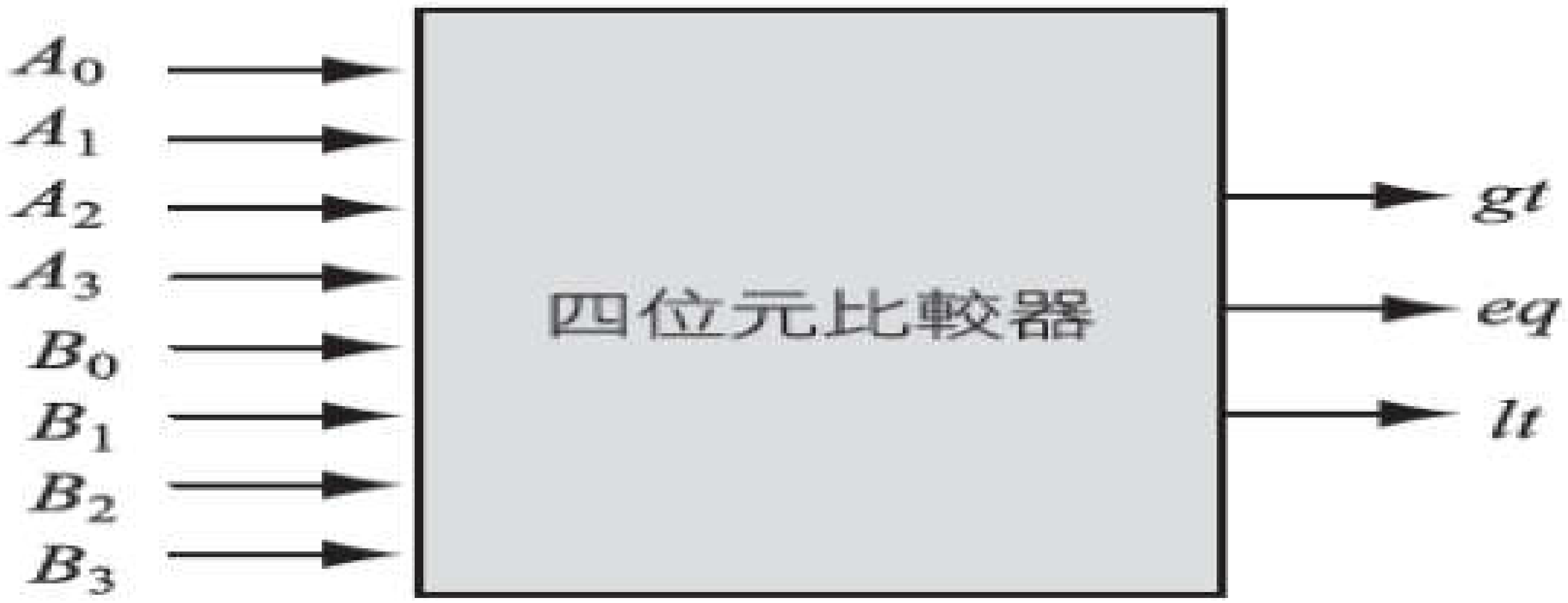
全加器(3/3)

- 全加器 = 兩個半加器 + 一個OR閘

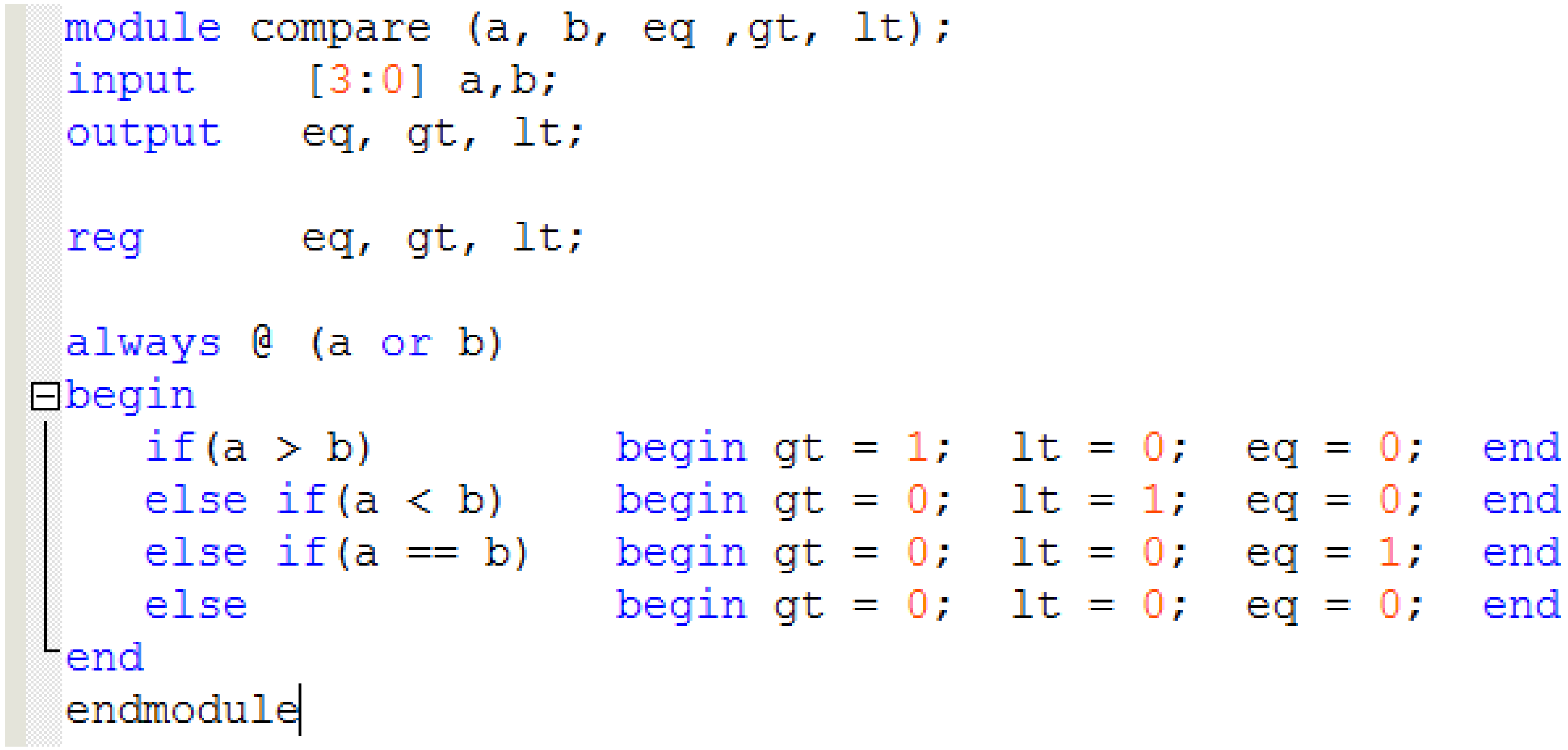


```
module FA (s, Cout, Cin, a, b);
input    Cin, a, b;
output   Cout, s;
wire     s1,c1,c2;

HA h1(.s(s1), .c(c1), .x(a), .y(b));
HA h2(.s(s), .c(c2), .x(s1), .y(Cin));
or or1(Cout, c1, c2);

endmodule

module HA (s, c, x, y);
input x, y;
output c, s;

assign s = x ^ y;
assign c = x & y;

endmodule
```

比較器(1/2)



比較器(2/2)



```
module compare (a, b, eq ,gt, lt);
input     [3:0] a,b;
output    eq, gt, lt;

reg       eq, gt, lt;

always @ (a or b)
begin
    if(a > b)           begin gt = 1;  lt = 0;  eq = 0;  end
    else if(a < b)      begin gt = 0;  lt = 1;  eq = 0;  end
    else if(a == b)     begin gt = 0;  lt = 0;  eq = 1;  end
    else                begin gt = 0;  lt = 0;  eq = 0;  end
end
endmodule
```