Verilog HDL

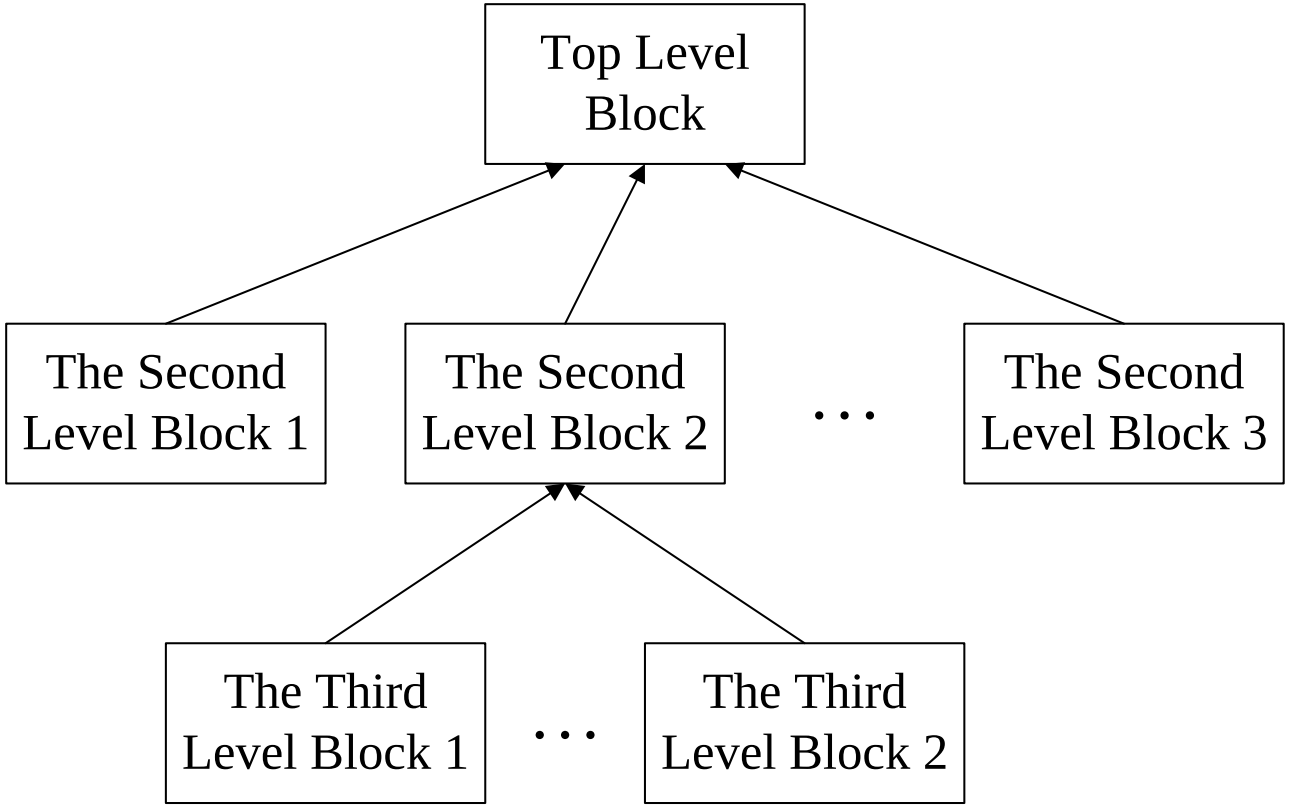
Mao-Hsu Yen

Verilog 硬體描述語言簡介

- Verilog 硬體描述語言的特性：
  - Verilog HDL 為一般性的硬體描述語言、易學易用。
  - Verilog HDL 的語法與 C 語言相似。
  - Verilog HDL 允許在同一個模組中有不同層次的表示法共同存在。
  - 一般的邏輯合成工具普遍都支援 Verilog HDL。
  - 許多的製造商有提供 Verilog HDL 的函式庫。

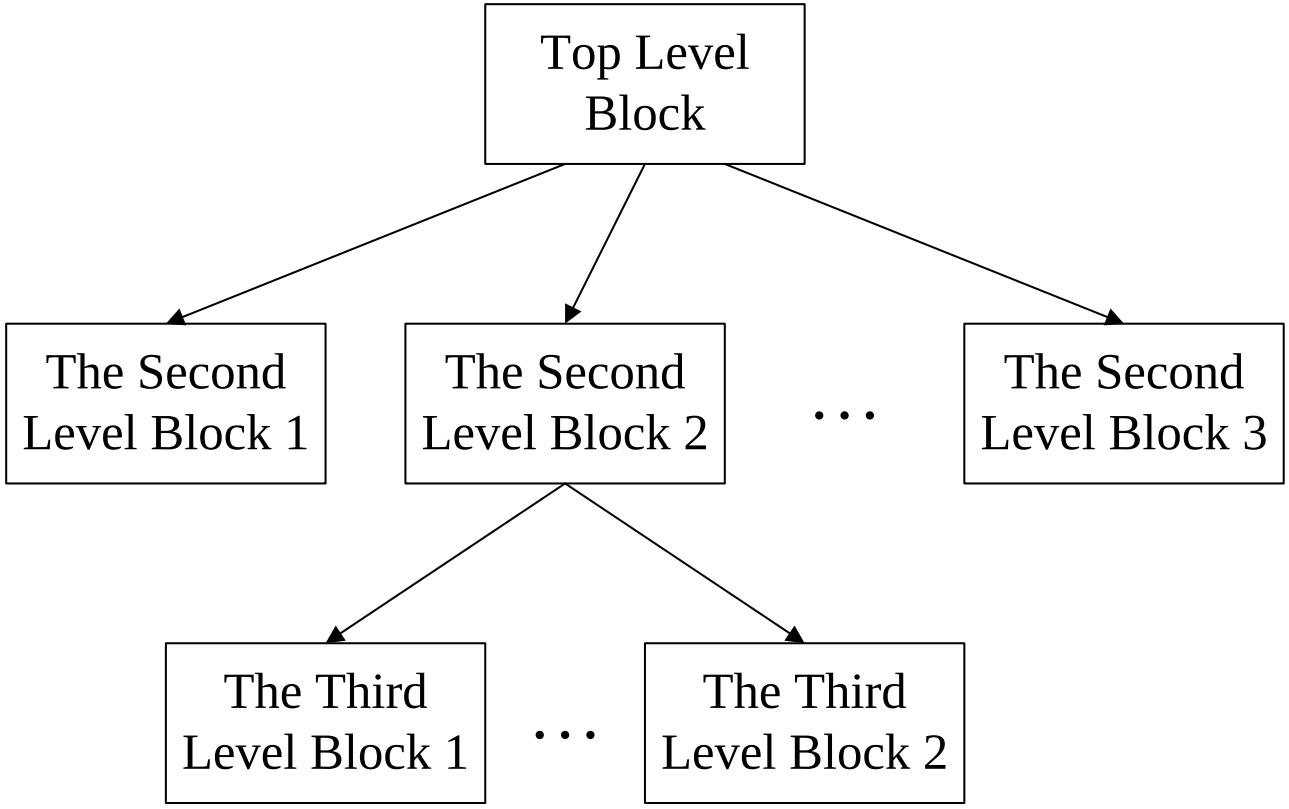
Verilog 的模組與架構

- 由低至高的設計方式



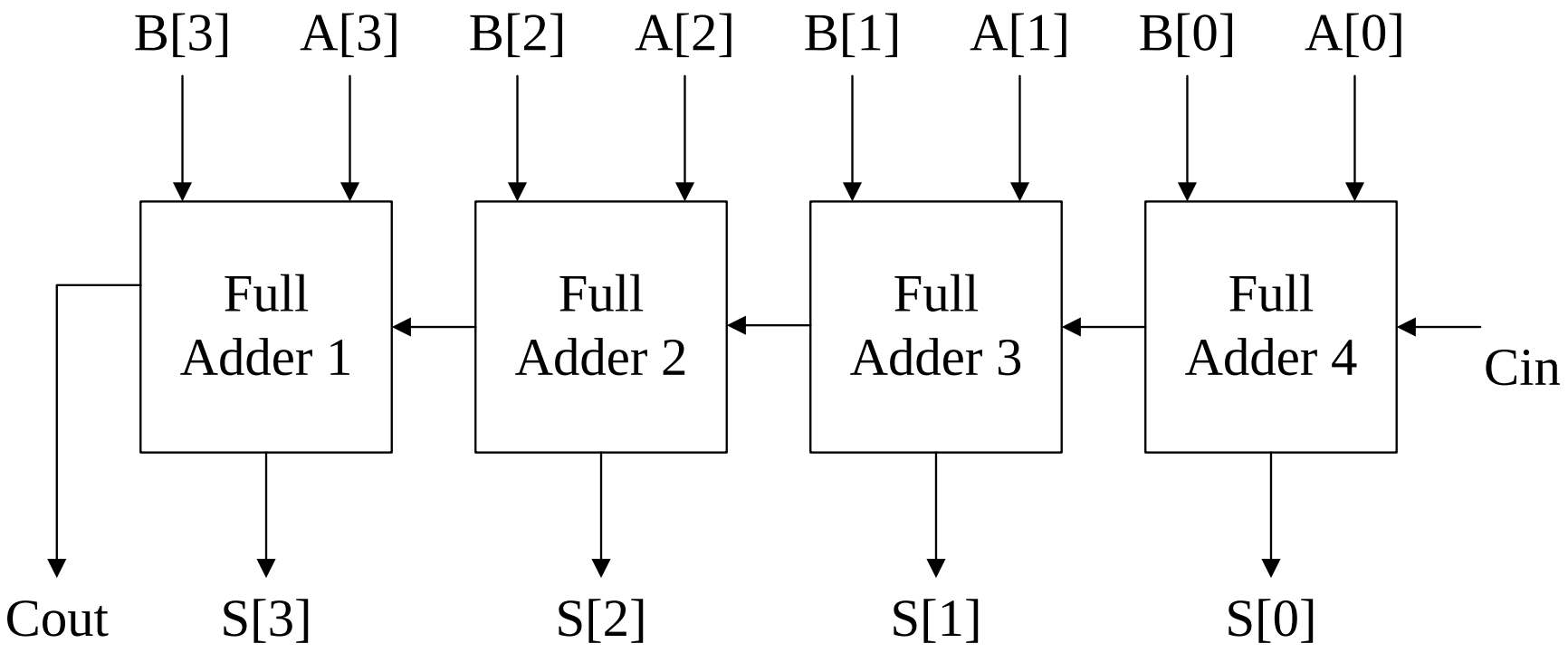
Verilog 的模組與架構

- 由高至低的設計方式



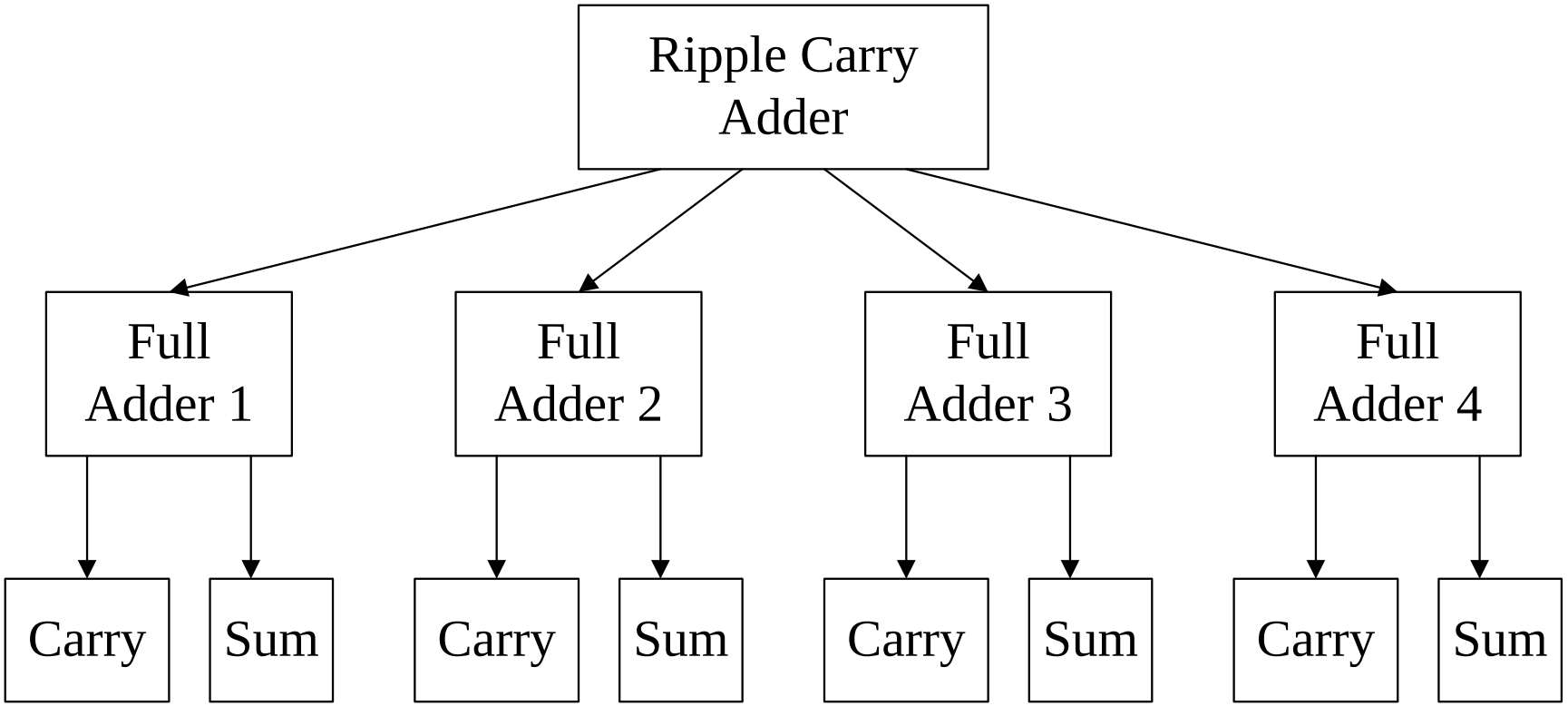
Verilog 的模組與架構

- 4-bit Ripple Carry Adder



Verilog 的模組與架構

- 階層式設計(Hierarchy Design)



Verilog 的模組與架構

- Verilog 的四大模型：是指用以描述電路功能或是電路架構的四種表示方式。
  - 低階交換模型(Switch Level Model) 或 電晶體模型(Transistor Model)。
  - 邏輯閘層次模型(Gate Level Model)。
  - 資料處理模型(Data Flow Model)。
  - 行為模型(Behavioral Model)。

Verilog 的模組與架構

```
module module_name(port_name);
        input, output … 等 埠列與埠的宣告;
        wire, reg … 等 資料型態變數的宣告;

        引用較低階之模組的別名;

        四個模組層次的描述;

        function 函數 與 task 任務的宣告;
endmodule
```

Verilog 的模組與架構

- 使用邏輯閘層次描述一個 2 input AND 閘

```
module GateAND2( out, in1, in2 );
        input     in1, in2;
        output    out;

        and  a1( out, in1, in2 );
endmodule
```

Verilog 的模組與架構

- 使用資料處理模型層次描述一個 2 input AND 閘

```
module GateAND2( out, in1, in2 );
        input     in1, in2;
        output    out;

        assign out = in1 & in2;
endmodule
```

Verilog 的模組與架構

- 使用行為模型層次描述一個 2 input AND 閘

```
module GateAND2( out, in1, in2 );
        input     in1, in2;
        output    out;
        reg       out;

        always @( in1 or in2 )
        begin
                  out = in1 & in2;
        end
endmodule
```

Verilog 的語法協定

- Verilog 是由一連串的標記(token) 所組成，這些標記可以是：
  - 空白(Whitespace)
  - 註解(Comments)
  - 運算子(Operators)
  - 數值(Numbers)
  - 識別字(Identifiers) 與 關鍵字(Keyword)
  - 底線(Underscore characters) 和 問號(Question Marks)

Verilog 的語法協定

- 空白：只要是空格(Space, ‘ ’)、跳格(Tab, ‘\t’)、換行(New Line, ‘\n’)，皆屬於空白。
- 單行註解用// …

  多行註解用/* … */
- 單元運算子：assign a = ~b;　　//放在運算元前

  二元運算子：assign a = b & c; //放在兩運算元間

  三元運算子：assign out = select ? a : b;

  //條件運算子，如果 select = 1，

  則 out = a，否則 out = b

Verilog 的語法協定

- 數值(Number Specification)：
  - <size>’<base><value>
  - <size>是以十進位來表示數字的位元數(Bits)
  - <base>用來定義此數值為：
    - 十六進位(‘h 或 ‘H)
    - 十進位(‘d 或 ‘D)
    - 八進位(‘o 或 ‘O)
    - 二進位(‘b 或 ‘B)
  - <value>0~f, x, z, ?
  - Ex:4’d15

Verilog 的語法協定

- Number Specification
  - 16 32-bit decimal number by default
  - ‘hc3 32-bit hexadecimal number
  - 8’b1100_0001 8-bit binary
  - 64’hff01 64-bit hexadecimal
  - 9’O17 9-bit octal
  - 2’B1? 2-bit binary number with LSB at high impedance
  - 32’bz 32-bit high-impedance number
  - 32’bx 32-bit binary unknown

Verilog 的語法協定

- 識別字是在Verilog電路描述中所給予物件的名稱。
- 識別字的第一個字元必須為字母，第二個字元之後可以為字母、數字、底線“_”、錢號“$”所組成。
- 識別字大小寫是有分別的：Even和even是不同的標記名稱。
- 關鍵字名稱必須使用小寫來表示。

Verilog的運算子

- 二元逐位元運算子(Binary bit-wise operator)
  - ~ (NOT), & (AND), | (OR), ^ (XOR), ~^或^~ (XNOR)
- 單元運算子(Unary reduction operator)
  - & (AND), ~& (NAND), | (OR), ~| (NOR), ^ (XOR), ~^或^~ (XNOR)
- 邏輯運算子(Logic operator)
  - ! (NOT), && (AND), || (OR)
- 二的補數算數(2's complement arithmetic)
  - + (加法), - (減法), * (乘法)
  - / (除法), (取餘數)：只適用電路模擬，不能合成。

Example

- a=4'b1011 b=4'b0010
- Bit-wire Operator
  - a|b => 4'b1011
  - a&b => 4'b0010
  - ~a => 4'b0100
- Unary reduction Operator
  - |a => 1'b1
  - &b => 1'b0

Verilog的運算子

- 關係運算子(Relational operator)
  - > (大於), < (小於), >= (大於等於), <= (小於等於)
- 等式運算子(Equality operator)
  - == (相等), != (不相等)
  - ===, !==：只適用電路模擬，不能合成。
- 移位運算子(Logical shift operator)
  - >> (右移), << (左移)
- 條件運算子(Conditional operator)
  - ?:

Verilog的運算子

- 重複運算子(Duplication operator)
  - assign out = {2{4'b110x}}; //out = 8'b110x110x
  - assign out = {3{2'b1x}, 2'b00}; //out = 8'b1x1x1x00
- 連結運算子(Concatenation operator)
  - 用以連結二個或以上的值到一個變數中，或將一個變數的值劃分到二個或以上的變數中
  - assign out = {2'b1x, 6'hf}; //out = 2'b1x001111
  - assign out = {4'b1010,4'b1010} //out = 8'b10101010
  - assign {part1, part2} = out; //劃分

Verilog 資料型態

- Verilog所提供的四種數值位準：
  - 0：邏輯0, Zero, False, Low, Ground, V_{SS}
  - 1：邏輯1, One, True, High, Power, V_{CC}, V_{DD}
  - X：不確定(Unknown)
  - Z：高阻抗(High Impedance)

Verilog 資料型態

- wire接線的特性：
  - 接線是連接硬體元件之連接線
  - 接線必須要被驅動才能改變它的內涵值
  - 接線描述的關鍵字為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高阻抗(Z)值
  - wire需配合assign使用，不能放在always區塊裡
    - wire b, c; //宣告2條接線，且命名為 b 和 c
    - wire a = b &c; //宣告1條接線a，並指定為 b & c
    - wire d = 1'b0; //宣告1條接線d，並給它固定值0

Verilog 資料型態

- reg暫存器特性
  - 與變數很像，可以直接給定一個數值
  - 主要功能在於保持住電路中的某個值，不必像wire要被驅動才能改變它的內涵值
  - 暫存器描述的關鍵字為reg
  - 除非有被宣告成一個向量，否則暫存器是內定為一個位元的don’t care(X)值
  - 在always和initial區塊中的變數，都必須宣告成reg

Verilog 資料型態

```
module test(zero, sum);
        input       sum;
        output      zero;
        wire [3:0] sum;
        reg zero;

        always @ (sum)
        begin
                if( sum == 0 )      zero = 1;
                else                zero = 0;
        end
endmodule
```

Verilog 資料型態

- 向量表示多位元的元件：
  - wire 和 reg 皆可以定義為向量，若無定義位元長度，則被視為宣告為一個位元
  - 向量可以定義為[#High:#Low]或[#Low:#High]
    - 以[#High:#Low]來說，中括號內之冒號左邊為最高位元，右邊為最低位元
  - 可以用向量表示法，取得向量內之部分訊號
    - wire a[7:0];　　//宣告有一八位元的Bus
    - reg [0:15] out; //宣告有一16位元寬度的向量暫存器變數out

Verilog輸出入埠的敘述

- input (輸入埠)
- output (輸出埠)
- inout (雙向埠)

```
module test( d, a, b, c );
        input a, b, c;
        inout d;
        wire d;
                assign d = a ? ( b ? c : d ) :1'bz
endmodule
```

Modules

module *module_name*(port_name);

Definitions

Module Instantiations

Module Statements & Constructs

endmodule

```
module test(z, a, b);
//port declarations
  input a,b;
  output z;

  wire and_out;
  and I1(and_out,a,b);
  not I2(z,and_out);
endmodule
```

Modules

```
module SR_latch(q, qbar, sbar, rbar);
// Port definition
output q, qbar;
input sbar, rbar;
// Module instantiations
nand n1(q, sbar, qbar);
nand n2(qbar, rbar, q);
endmodule
```

Gate Types

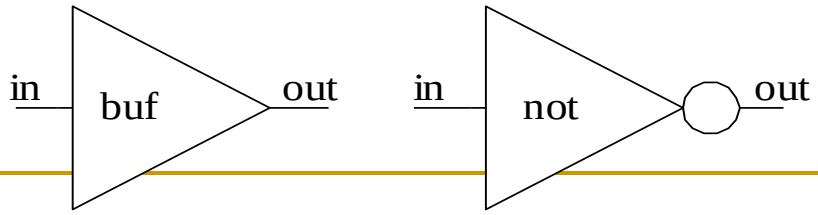
- **Buf** / **Not** Gates have ***one*** scalar input and one or ***more*** scalar outputs.

  buf b1(out1,in); //one output

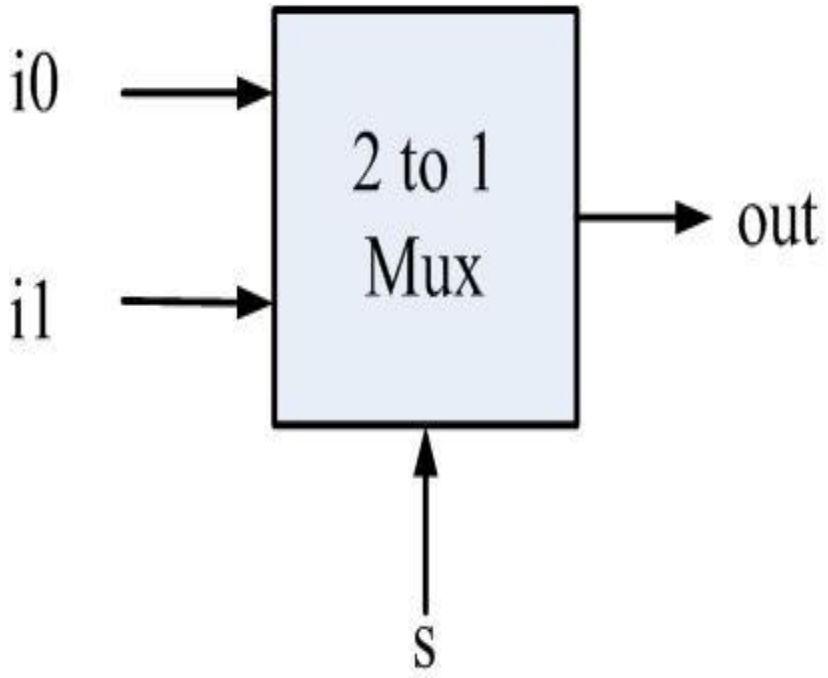
  not n1(out2,in); //one output

  buf b1_2out(out3,out4,in); //2 outputs

  not (out1, in);



2-to-1 Multiplexer



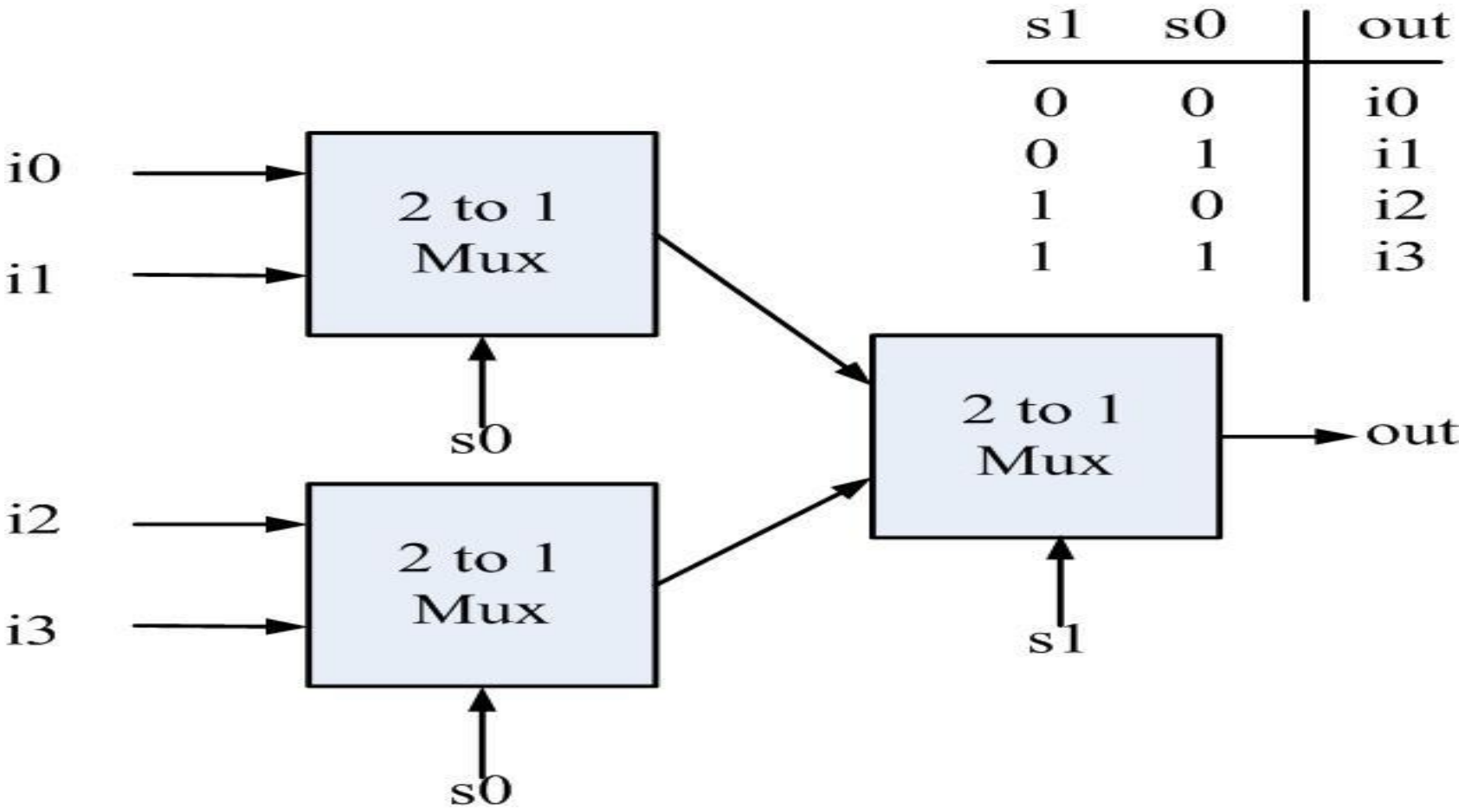
s	out
0	i0
1	i1

2-to-1 Multiplexer

```
module Mux2to1(out,i0,
    i1,s);
input i0,i1,s;
output out;
not a1(sbar,s);
and a2(net1,i0,sbar);
and a3(net2,i1,s);
or a4(out,net1,net2);
endmodule
```

```
module Mux2to1(out,i0,
    i1,s);
input i0,i1,s;
output out;
assign sbar= ~s;
assign net1=i0 & sbar;
assign net2=i1 & s;
assign out=net1 | net2;
endmodule
```

4-to-1 Multiplexer

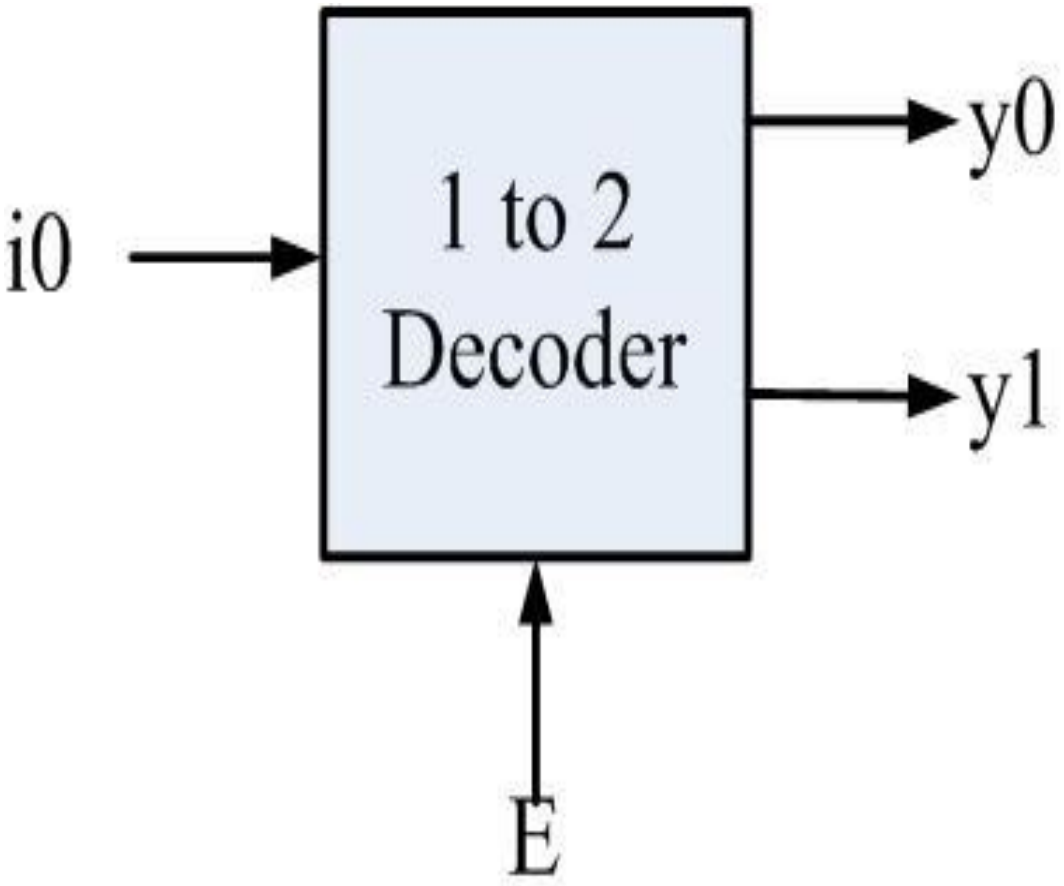


s1	s0	out
0	0	i0
0	1	i1
1	0	i2
1	1	i3

4-to-1 Multiplexer

```
module Mux4to1(out,s1,s0,i0,i1,i2,i3);
input s1,s0,i0,i1,i2,i3;
output out;
wire net1,net2;
// module Mux2to1(out,i0,i1,s);
Mux2to1 a1(net1,i0,i1,s0);
Mux2to1 a2(net2,i2,i3,s0);
Mux2to1 a3(out, net1, net2,s1);
endmodule
```

1-to-2 decoder



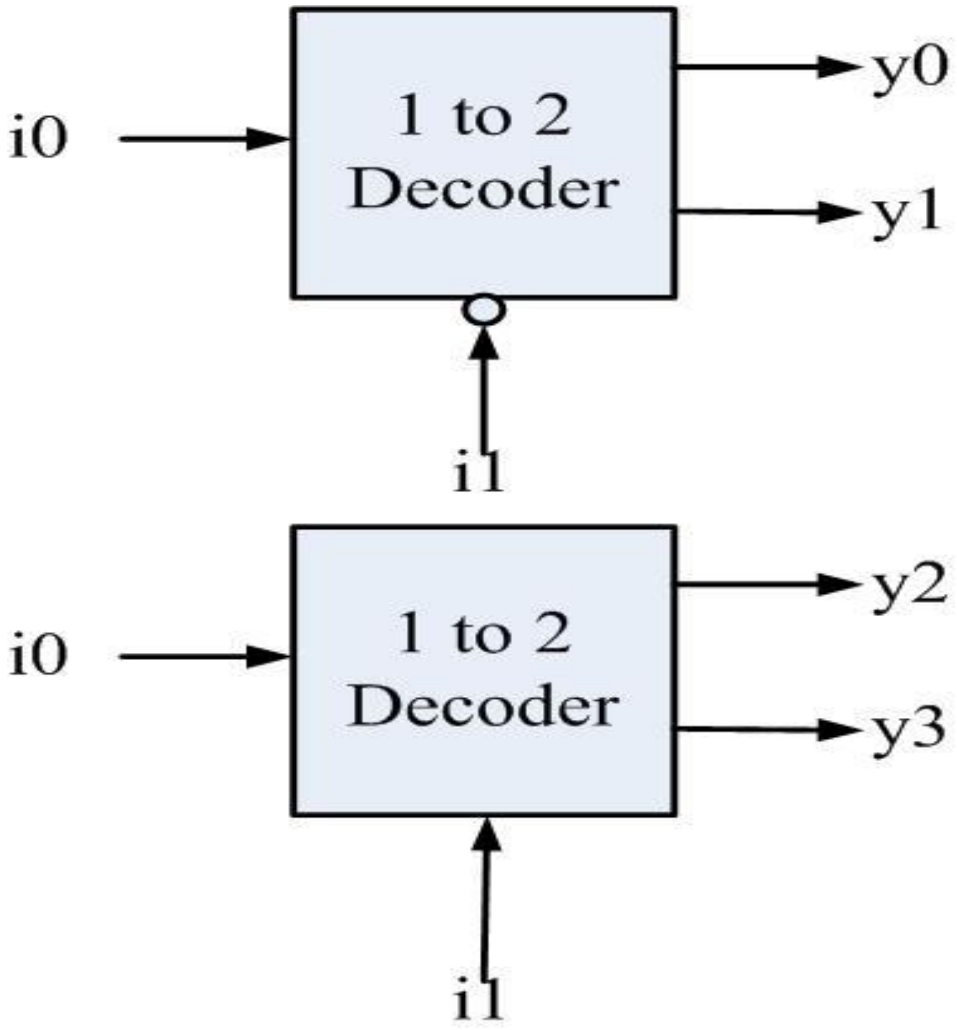
E	i0	y0	y1
0	X	0	0
1	0	i0	0
1	1	0	i0

1-to-2 decoder

```
module decoder1to2
    (y0,y1,E,i0);
input i0,E;
output y0,y1;
not a1(ibar,i0);
and a2(y0,ibar,E);
and a3(y1,i0,E);
endmodule
```

```
module decoder1to2
    (y0,y1,E,i0);
input i0,E;
output y0,y1;
assign ibar= ~i0;
assign y0= ibar & E;
assign y1= i0 & E;
endmodule
```

2-to-4 decoder



i1	i0	y0	y1	y2	y3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

2-to-4 decoder

```
module decoder2to4(y3,y2,y1,y0,i0,i1);
input i0,i1;
output y3,y2,y1,y0;
// module decoder1to2 (y0,y1,E,i0);
not a3(i1bar,i1);
//assign i1bar=~i1;
decoder1to2 a1(y0,y1,i1bar,i0);
decoder1to2 a2(y2,y3,i1,i0);
endmodule
```