Verilog和SystemVerilog中的变量类型

Variable Type in Verilog and SystemVerilog

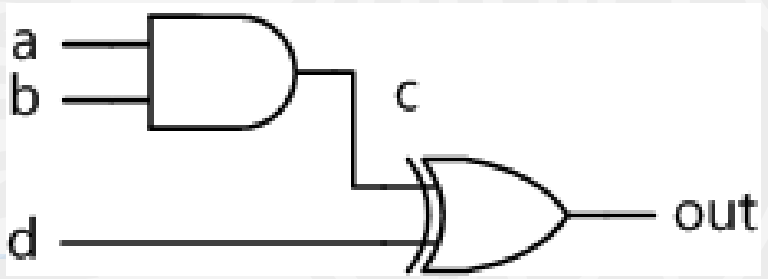
Verilog

- 线网类型（wire）：表示一条连线。
  - wire类型的信号只能在持续赋值语句（assign）中被赋值
  - wire类型的赋值被综合为组合逻辑电路
  - wire类型可用于声明输入端口、输出端口和中间信号，也是端口的默认类型
- 寄存器类型（reg）：表示存储结构，条件触发（如时钟沿）时改变内部状态。
  - reg类型的信号只能在过程块（always、initial、task）中被赋值
  - reg类型的赋值既可被综合为时序逻辑，也可被综合为组合逻辑，取决建模方式
  - reg类型可用于声明输出端口和中间信号，不能用于声明输入端口

Verilog和SystemVerilog中的变量类型

Variable Type in Verilog and SystemVerilog

Verilog（cont.）



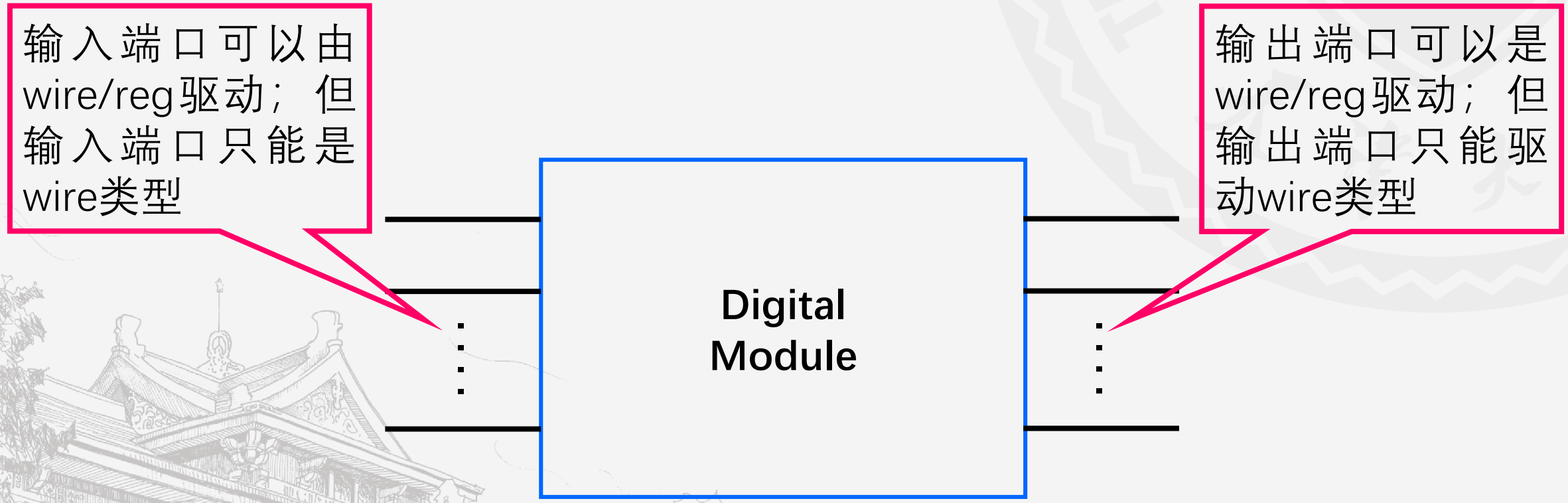

```
wire a, b, d;
reg out;
always @(*)
begin
   out = (a & b) ^ d;
end
```

```
wire a, b, d;                    错误!
wire out;
always @(*)
begin
   out = (a & b) ^ d;
end
```

```
wire a, b, d;                    错误!
reg out;
assign out = (a & b) ^ d;
```

```
wire a, b, d;
wire out;
assign out = (a & b) ^ d;
```

Verilog（cont.）



Verilog（cont.）

```
module DUT (out, a, b, d);

output out;  //默认wire类型

input a, b, d; //默认wire类型

assign out = (a & b) ^ d;

endmodule
```

```
module DUT (out, a, b, d);

output reg out;

input a, b, d; //默认wire类型

always @(*)

   out = (a & b) ^ d;

endmodule
```

Verilog和SystemVerilog中的变量类型

Variable Type in Verilog and SystemVerilog

SystemVerilog

- 逻辑类型（logic）：是在Verilog基础上新增的信号类型，是reg类型的改进
  - 既可被过程赋值也能被持续赋值
  - 综合器自动推断logic是reg还是wire，取决于综合器分析能力
  - 对于输入端口，logic被推断为wire类型
  - 对于输出端口和中间信号，logic类型的推断取决于建模描述方式
  - 如果输出端口在过程块中被赋值，需要显示声明为logic类型

Verilog和SystemVerilog中的变量类型

Variable Type in Verilog and SystemVerilog

SystemVerilog（cont.）

错误！

```
module (
        input a, b, d,
        output out
);

always_comb
        out = (a & b) ^ d;

endmodule
```

out端口默认为wire类型，而wire类型不能在过程块中赋值

正确！

```
module (
        input a, b, d,
        output logic out
);

always_comb
        out = (a & b) ^ d;

endmodule
```

out端口被推断为reg类型，而reg类型可以在过程块中赋值

SystemVerilog（cont.）

```
module (                          正确!
        input a, b, d,
        output out
);


assign out = (a & b) ^ d;

endmodule
```

out端口默认为wire类型，而wire类型可以被持续赋值

```
module (                          正确!
        input a, b, d,
        output logic out
);

assign out = (a & b) ^ d;


endmodule
```

out端口被推断为wire类型，而wire类型可以被持续赋值

SystemVerilog（cont.）

```
module (                                    正确!
        input a, b, d,
        output logic out
);

logic a, b, d;

always_comb
        out = (a & b) ^ d;

endmodule
```

在头歌平台上这样的写法报错，其实这是正确写法，这是由编译器（iverilog）不完善，分析能力不足造成的。

建议：输入端口采用默认声明
输出端口声明为logic类型

基于持续赋值语句

- 基于**持续赋值语句**的建模（assign）只要“=”右侧表达式中的任意变量发生变化，该表达式立即重新计算并赋值给左边的变量，因此，只能用于描述组合逻辑电路，**关键字assign不能省略**。
- 如果定义了“<#延迟量>”，是**不可综合的！**

```
logic [3 : 0] out1, out2, A, B;
assign out1 = A + B;          // 正确，可综合
assign #5 out2 = ~(A & B);    // 正确，但不可综合
out2 = ~(A & B);              // 错误，缺少了关键词assign
```

基于过程块

- 关键词**always**或**initial**定义，通过**begin…end**包围起来的过程块对电路进行描述。可用于描述**组合逻辑电路**，也可用于描述**时序逻辑电路！**
- **always_comb**用于描述组合逻辑电路（另一种描述方式为**always @(*)**）。
- 过程块中的赋值语句**不能使用关键字assign**，并且**“=”左边的信号必须是logic类型**，不能是线网类型。
- 描述组合逻辑电路时，赋值语句称为**阻塞赋值**，即赋值立即发生。

分支结构

- 分支结构（包括if...else、case）要**包含所有分支可能**，这样才能确保产生**组合逻辑电路**；否则，将综合出带有锁存器的时序电路，除非设计者可以确定这是需要的电路类型。

```
always_comb begin
    case (select)
        2'b00 : out = a;
        2'b01 : out = b;
        2'b10 : out = c;
    endcase
end
```

不完整分支

```
always_comb begin
    case (select)
        2'b00 : out = a;
        2'b01 : out = b;
        2'b10 : out = c;
        2'b11 : out = d;
    endcase
end
```

完整分支

循环结构

- 循环语句块将被展开复制为多个并行的电路结构，占用较大面积，循环次数越多，占用面积越大，一定要**慎用循环结构！**除非设计者确定只能使用这种结构。

```
. . .
for(i = 1; i < 10; i = i + 1)
  C[i] = A[i] & B[9-i];
. . .
```

展开

```
. . .
C[0] = A[0] & B[9];
C[1] = A[1] & B[8];
. . .
C[9] = A[9] & B[0];
. . .
```

循环结构（cont.）

请使用for语句设计一个数字逻辑电路，能够统计8位输入信号din中1的个数。

```
module loop_demo(input logic [7 : 0] din,
                 output logic [2 : 0] out);
    int num_bits;
    always_comb begin
        int i;
        num_bits = 0;
        for (i = 0; i < 8; i = i+1) begin
            if (din[i] == 1)
                num_bits = num_bits + 1;
            else num_bits = num_bits;
    end
    assign out = num_bits;
endmodule
```

不推荐在组合逻辑电路建模中出现类似反馈的结构！容易产生震荡！

模块实例化
Module Instantiation

端口关联

```
module mux4 (input logic D0, D1, D2, D3,
             input logic [1 : 0] s,
             output logic y);
  logic low, high

  mux2 lowmux (D0, D1, sel[0], low);
  mux2 highmux (D2, D3, sel[0], high);
  mux2 finalmux (low, high, sel[1], y);
endmodule
```

位置关联

```
module mux4 (input logic D0, D1, D2, D3,
             input logic [1 : 0] s,
             output logic y);
  logic low, high

  mux2 lowmux (.y(low), .D0(D0), .D1(D1), .sel(sel[0]));
  mux2 highmux (.D1(D3), .D0(D2), .sel(sel[0]), .y(high));
  mux2 finalmux (.D0(low), .sel(sel[1]), .D1(high), .y(y));
endmodule
```

名称关联

推荐使用名称关联法！！！

一些说明

Some explanations

- constant selects in always_* processes are not currently supported (all bits will be included)
  - 将**always_comb**换成**always @(*)**，开源综合器问题导致的。
- Part select expressions must be constant
  - 域选表达式必须是常量，如**Din[7 : i]**是错误的。
- 循环变量可以是int类型，也可以是logic类型，但**logic类型默认是无符号数。**
- 中间信号的声明**推荐**放在**always块之外**。此外，HDL中没有中间变量，全局变量的概念。

一些说明

Some explanations

写HDL代码要心中有电路结构，不要只关注功能，否则，综合出的电路可能不是你想要的，可能性能很差，甚至出现错误！

- 电路结构，不是电路细节
- 选择器、译码器、编码器、加法器、移位器、乘法器、寄存器、存储器，这些常见的电路结构描述方式基本固定，要准确掌握
- 现代EDA工具已经很强大，但毕竟做不到完全智能，HDL写的越规范，电路性能越好

00：心中无电路，代码无电路　　01：心中有电路，代码无电路

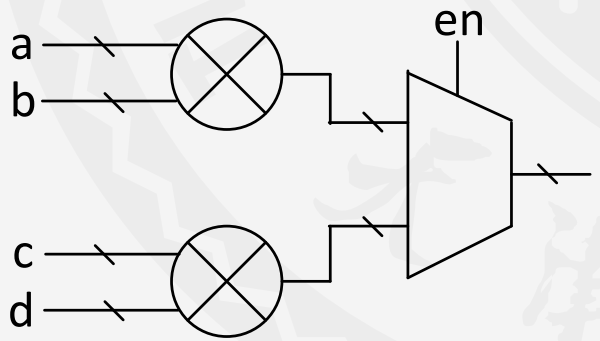
10：心中有电路，代码有电路　　11：心中无电路，代码有电路

一些说明

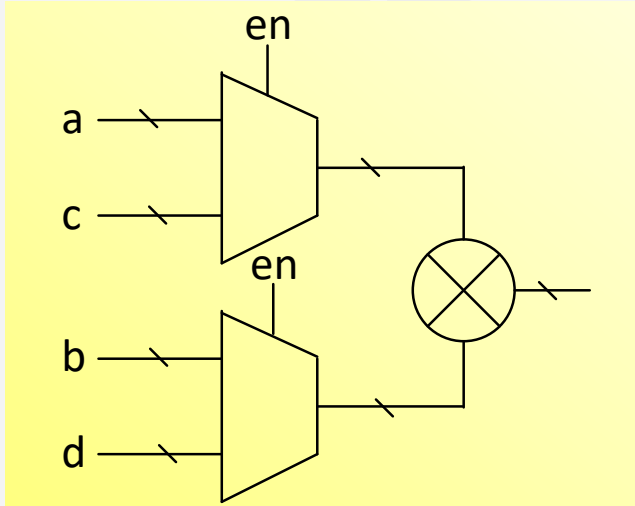
Some explanations

- 当信号en为高电平，输出out = a*b；否则输出out = c*d。

```
assign out = (en) ? a*b : c*d;
```




```
assign tmp0 = (en) ? a : c;
assign tmp1 = (en) ? b : d;
assign out = tmp0 * tmp1;
```



一些说明

Some explanations

- constant selects in always_* processes are not currently supported (all bits will be included)
  - 将**always_comb**换成**always @(*)**，开源综合器问题导致的。

- always @(*) begin

  . . . . .

  assign S=tt[0];

  assign Cout=tt[1];

  . . . . .

  end

错误信息：S is not a valid l-value，S is declared here as wire

过程块中不能使用持续赋值语句

一些说明

Some explanations

```
always_comb begin
        if (en == 0)    x = 1;
        else if (A[3] == 0)     _74LS138(A[2:0], 1,0, 0,Y[7:0]);
        else if (A[3] == 1)     _74LS138(A[2:0], 1,0, 0,Y[15:8]);
end
```

错误信息：Enable of unknown task _74LS138'

- 例化模块不是函数调用，**例化的模块**、**always块**、**持续赋值语句**都是平级关系，不存在调用关系。这些语法都是用于构建一个独立电路功能子模块的，电路模块间是靠信号进行信息交互，而不是靠调用关系进行信息交互。因此，不能在过程块中进行模块的实例化

一些说明

Some explanations

- 向量信号进行域选的时候，指定最低位和最高位一定是常量，不能是变量
- 持续赋值语句只能给wire类型赋值；块语句只能给logic类型赋值
- 循环语句中，while和forever是不可综合的，因为无法确定循环次数；for和repeat是可以综合的