

Verilog HDL简单回顾

② 主讲人：蔡建

德以明理 学以精工

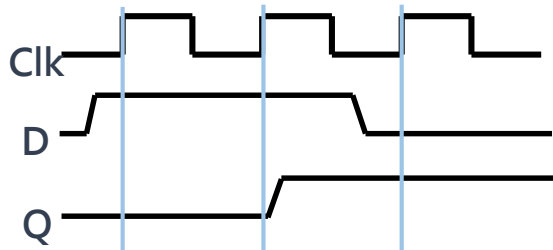
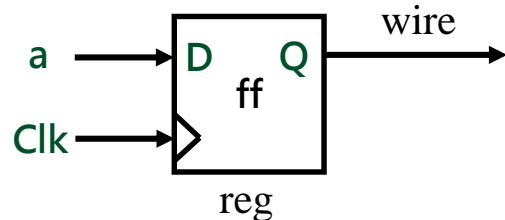


Verilog几个基本要素

写Verilog代码本质上是在搭硬件电路。

wire型变量与reg型变量

- 一位wire对应数字电路中的一条线
- 一位reg变量多数情况对应数字电路中的一个D触发器



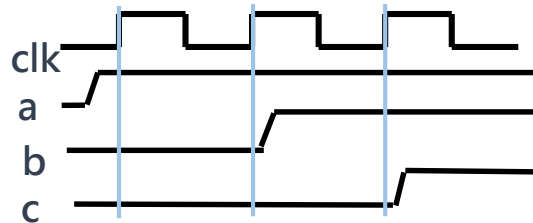
两种基本语句结构

- assign赋值：用来描述线之间的组合逻辑关系，针对wire型变量
- always语句块：用来描述电路中的时序状态转换，针对reg型变量

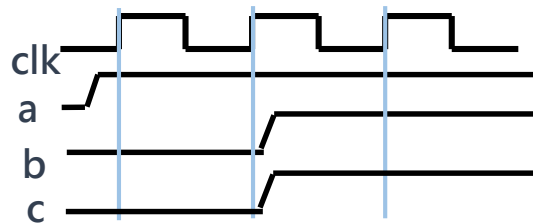
赋值语句


- 持续赋值：assign-主要用于对wire型变量的赋值
- 过程赋值：多用于always块内，分为
 - { 阻塞赋值 = : 该语句结束时就立即完成赋值操作
 - { 非阻塞赋值 <= : 整个过程块结束时才完成赋值操作

```
wire a;  
reg b, c;  
always @(posedge clk) begin  
    b <= a;  
    c <= b;  
end
```



```
wire a;  
reg b, c;  
always @(posedge clk) begin  
    b = a;  
    c = b;  
end
```



类 别	运 算 符	优先级
逻辑非、按位取反	! ~	高  低
算术运算符	* / %	
	+ -	
移位运算符	<< >>	
关系运算符	< <= > >=	
等式运算符	= = ! = == = ! ==	
缩减运算符	& ~&	
	^ ^~	
	~	
逻辑运算符	&&	
条件运算符	? :	

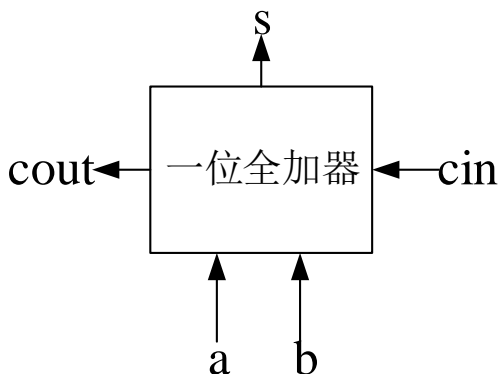
如果不确定优先级，请尽量使用小括号；如果想要将两个信号的位进行拼接可以用大括号。



组合逻辑与时序逻辑

组合逻辑

全加器
多路选择器
ALU
...

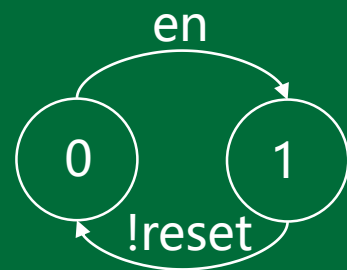


```
wire a, b, cin, s, cout;  
assign s = a^b^cin;  
assign cout = (a&b) | (b&cin) | (cin&a);
```

wire 变量; assign 赋值语句; 阻塞赋值

时序逻辑

D 触发器
计数器
各类状态机
...

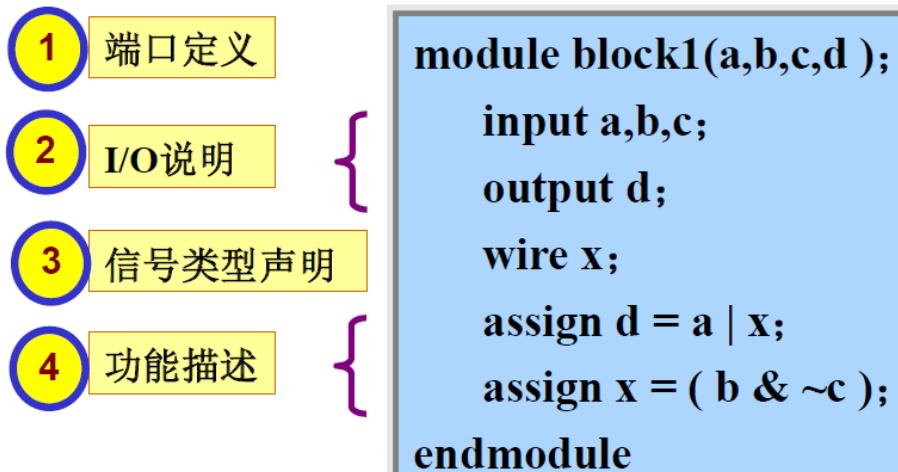


一位计数器

```
input clk, reset;  
reg count;  
wire en;  
always @(posedge clk) begin  
    if(reset==0) count <= 0;  
    else if(en) count <= 1;  
end
```

reg 变量;
always 过程语句;
非阻塞赋值

模块的结构



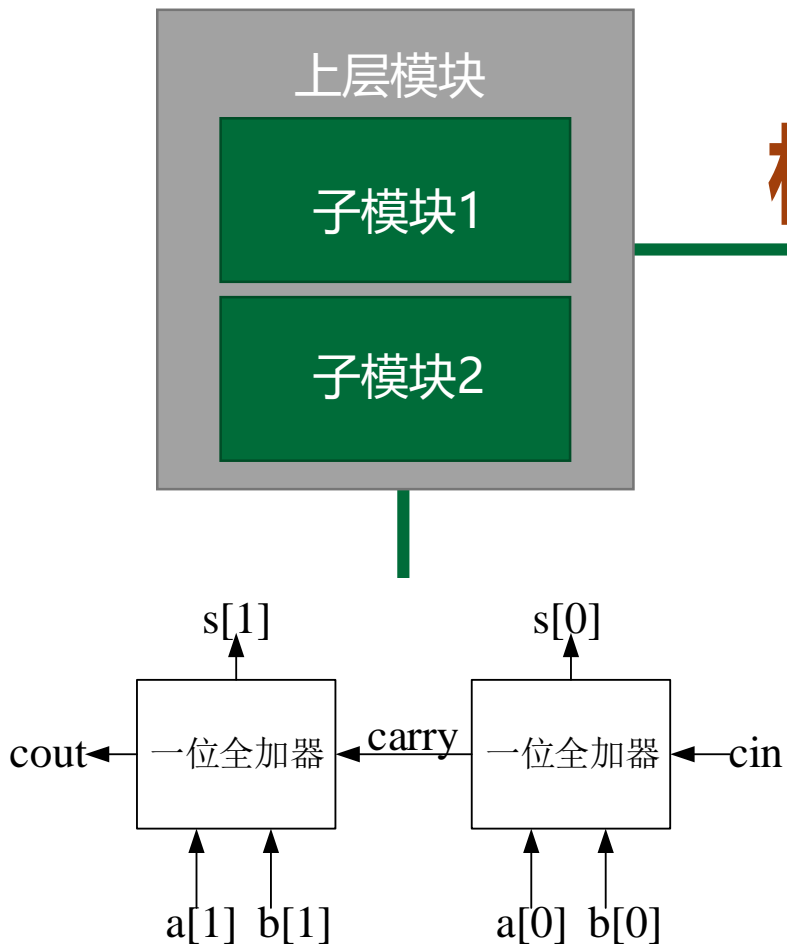
```
module full_adder(a,b,cin,s,cout)  
    input a, b, cin;  
    output s, cout;  
    assign s = a^b^cin;  
    assign cout = (a&b) | (b&cin) | (cin&a);  
endmodule
```

1. Verilog HDL程序是由模块构成的。每个模块嵌套在**module**和**endmodule**声明语句中。模块是可以进行层次嵌套的。
2. 每个Verilog HDL源文件中只准有一个顶层模块，其他为子模块。源文件名称与顶层模块同名（后缀为.v）。
3. 每个模块要进行输入输出端口定义，然后用语句对模块的功能进行描述。
4. 语句用分号结束，可用/*...*/和//...对程序的任何部分作注释。



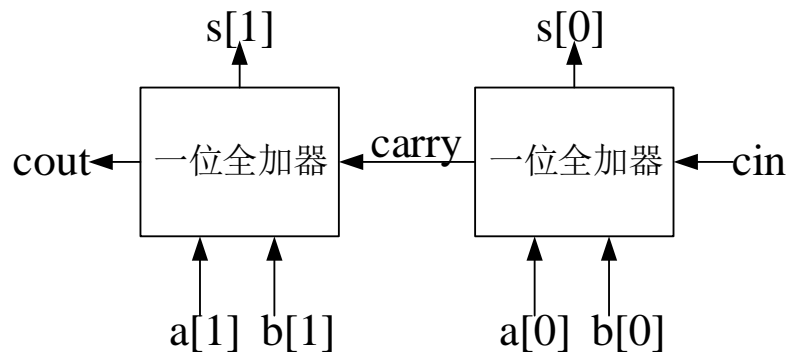
模块元件例化

FPGA逻辑设计中通常是一个大的模块中包含了一个或者多个小的模块。上层模块通过调用子模块的功能来实现更高层次的硬件逻辑封装。这里的模块调用就可以称之为模块确定特定输入输出连接后的实例化。同一个设计模块可以被同一个或者多个其它模块多次实例化。





2位串行加法器实例



一位加法器设计模块: full_adder.v

```
module full_adder(a,b,cin,s,cout)
    input a, b, cin;
    output s, cout;
    assign s = a^b^cin;
    assign cout = (a&b) | (b&cin) | (cin&a);
endmodule
```

• serial_adder_2bits.v

```
module serial_adder_2bits(
```

```
    input  [1:0] a,
    input  [1:0] b,
    input  cin,
    output [1:0] s,
    output cout
```

上层模块输入
输出信号定义

```
);
```

设计模块名

```
wire carry;
```

例化模块名

```
full_adder bit0(
```

```
    .a(a[0]),
```

```
    .b(b[0]),
```

.指定要对接的
设计模块信号

```
    .c(cin ),
```

```
    .s(s[0]),
```

```
    .cout(carry));
```

上层模块传给例化子
模块对应接口的信号

```
full_adder bit1(.a(a[1]),.b(b[1]),.c(carry),.s(s[1]),.cout(cout ));
```

```
endmodule
```





写代码时的注意点

避免逻辑回环

```
module wrong(a)
  output a;

  wire b,c;

  assign b = a;
  assign c = b;
  assign a = c;

endmodule
```

避免多驱动

```
module wrong(a,b,c)
  output a;

  input b,c;

  assign a = b;
  assign a = c;

endmodule
```

意外生成锁存器

```
module wrong(a,b,c)
  output reg a;

  wire b,c;

  always @(*)
  begin
    if(b)
      a = c;
  end

endmodule
```

其它要点

1. 注意合理使用阻塞赋值与非阻塞赋值。
2. 注意存在initial、# 等之类的语句只能用来仿真，无法综合。
3. 注意仿真时 Z 表示对应连线浮空，X 则表示当前信号值不确定，往往是由于 Z 造成的。



建议

- ◆ 除testbench外禁止使用initial语句
- ◆ 避免使用always @ (*)
- ◆ 避免使用always @ (a)
- ◆ 避免使用always @ (posedge clk or negedge resetn)
- ◆ 最好：一个always只对一个变量赋值
- ◆ 禁止：多个always对一个变量赋值
- ◆ 建议只用always @(posedge clk) ———时序电路
- ◆ 建议assign ———组合电路
- ◆ 请多多使用中间变量
- ◆ 常量请加宽度： 1' b0、32' hffff_ffff
- ◆ 良好的代码书写习惯
 - 命名规律
 - 适当缩进
 - 信号对齐

感谢各位

◎ 主讲人：蔡建

计算机学院

德以明理 学以精工

