可编程逻辑器件设计基础

课程 编号：1713013170
学时/学分：24/1.5
考核 方式：考查

2021~2022学年第一学期

第2章 Verilog HDL基础

课程目标

课程目标2：能应用硬件描述语言，实施程序编写。

课程目标4：能设计基于可编程逻辑器件的数字系统。

第2章 Verilog HDL基础

2.1 Verilog语言概述

2.2 Verilog程序结构

2.3 Verilog语言要素

2.4 Verilog常量

2.5 Verilog数据类型

2.6 Verilog名字空间

2.7 Verilog 语言表达式

2.1 Verilog语言概述

2.1.1 硬件描述语言HDL概念

- **Hardware Description Language**：硬件设计人员和**EDA**工具之间的界面，主要目的是用来编写设计文件，建立电子系统行为级的仿真模型。
- 任务：
  - 对所建模型的复杂数字逻辑进行仿真；
  - 自动综合，生成符合要求且在电路结构上可以实现的数字逻辑网表（**Netlist**）；
  - 根据网表和某种工艺器件自动生成具体电路；
  - 最后生成可下载到**PLD**具体电路的时延模型。

2.1 硬件描述语言HDL

2.1.2 硬件描述语言特点

① HDL既包含一些高层程序设计语言的结构形式，同时也兼顾描述硬件线路连接的具体构件。

② 实际硬件具有在同一时刻执行多任务的能力，HDL具有并发的特征；编程语言是非并行的。

③ 硬件电路中从输入到输出会有延迟存在，为此HDL需要建立时序的概念。编程语言是没有时序概念的。

④ HDL通过使用结构级或行为级描述可在不同的抽象层次进行描述设计，采用自顶向下的数字电路设计方法。

软件描述语言与硬件描述语言的区别

软件语言设计目标流程

C、asm
源程序

软件程序
编译器
Compile

编译

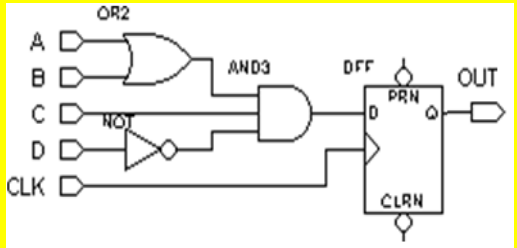
CPU指令、数据代码流
01000101
11001101
……

硬件语言设计目标流程

VHDL/Verilog
程序

HDL综合器
Synthesizer

综合



电路结构网表文件

HDL抽象层次描述表

	行为领域	结构领域	物理领域
系统级	性能描述	部件及它们之间的逻辑连接方式	芯片、模块、电路板和物理划分的子系统
算法级	I/O应答算法级	硬件模块数据结构	部件之间的物理连接、电路板、底盘等
寄存器传输级	并行操作寄存器传输、状态表	算术运算部件、多路选择器、寄存器总线、微定序器、微存储器之间的物理连接方式	芯片、宏单元
逻辑级	用布尔方程叙述	门电路、触发器、锁存器	标准单元布图
电路级	微分方程表达	晶体管、电阻、电容、电感元件	晶体管布图

传统的硬件抽象级的模型类型

① 系统级（**system**）用语言提供的高级结构实现系统运行的模型；

② 算法级（**algorithm**）用语言提供的高级结构实现算法运行的模型；

③ **RTL**级（**Register Transfer Level**）描述数据如何在寄存器之间流动和如何处理、控制这些数据流动的模型；

④ 门级（**gate-level**）描述逻辑门以及逻辑门之间的连接模型，与逻辑电路有确切的连接关系；

⑤ 开关级（**switch-level**）描述器件中三极管和存储结点以及它们之间连接的模型，开关级与具体的物理电路有对应关系。

2.1 硬件描述语言HDL

2.1.3 Verilog HDL硬件描述语言

- **Verilog HDL**：一种从算法级、门级到开关级多种抽象设计层次的、用于数字系统建模的硬件描述语言。
- 特点：
  - 既可进行电路设计，又能进行电路与系统仿真；
  - 能在多个层次上，采用多种方式对所设计的系统加以描述；
  - 行为描述语句类似于软件高级语言；
  - 内置多种逻辑门；
  - 支持用户定义原语（UDP）创建。

Verilog 语言功能

① 语言中内置基本逻辑门，如and、or和nand等。

② 允许用户自己定义元件UDP（User Defined Primitives）。通过UDP，可以把一块组合逻辑电路或者时序逻辑电路封装在一个UDP内，作为一个基本的元件来使用。注意：UDP不能综合，只能用于仿真。

③ 语言中内置开关级基本结构模型，例如pmos和nmos等。

④ 提供显式语言结构指定设计中的端口到端口的时延及路径时延和设计的时序检查。

⑤ 三种设计建模方式：

- 行为描述方式—使用过程化结构建模；
- 数据流方式—使用连续赋值语句方式建模；
- 结构化方式—使用门和模块实例语句描述建模。

⑥ 网络数据类型表示构件间的物理连线；寄存器数据类型表示抽象的数据存储元件。

⑦ 语言不对设计的规模（大小）施加任何限制。

⑧ 设计能够在多个层次上加以描述，从开关级、门级、寄存器传送级（RTL）到算法级，包括进程和队列级。

2.1 HDL硬件描述语言

2.1.4 Verilog和VHDL比较

① VHDL是为了实现美国国防部VHSIC计划所推出的各个电子部件供应商具有统一数据交换格式标准的要求，从推出的过程来看，偏重于标准化的考虑。

② Verilog HDL则和EDA工具结合得更为紧密，是在全球最大的EDA/ESDA供应商Cadence公司的支持下针对EDA工具专门开发的硬件描述语言。已有20多年历史的Verilog HDL拥有广泛的设计群体，成熟的资源远比VHDL丰富。

2.1 HDL硬件描述语言

③ Verilog HDL是从高级设计语言C语言发展而来的，相比VHDL而言更容易上手，其编码风格也更为简洁明了，是一种非常容易掌握的硬件描述语言。

④ 在行为级抽象建模的覆盖范围方面不同。一般认为Verilog HDL在系统抽象方面比VHDL要强一些，Verilog HDL比较适合算法级（Algorithm）、寄存器传输级（RTL）、逻辑级（Logic）以及门级（Gate）的设计，而VHDL更适合特大型系统级（System）的设计。

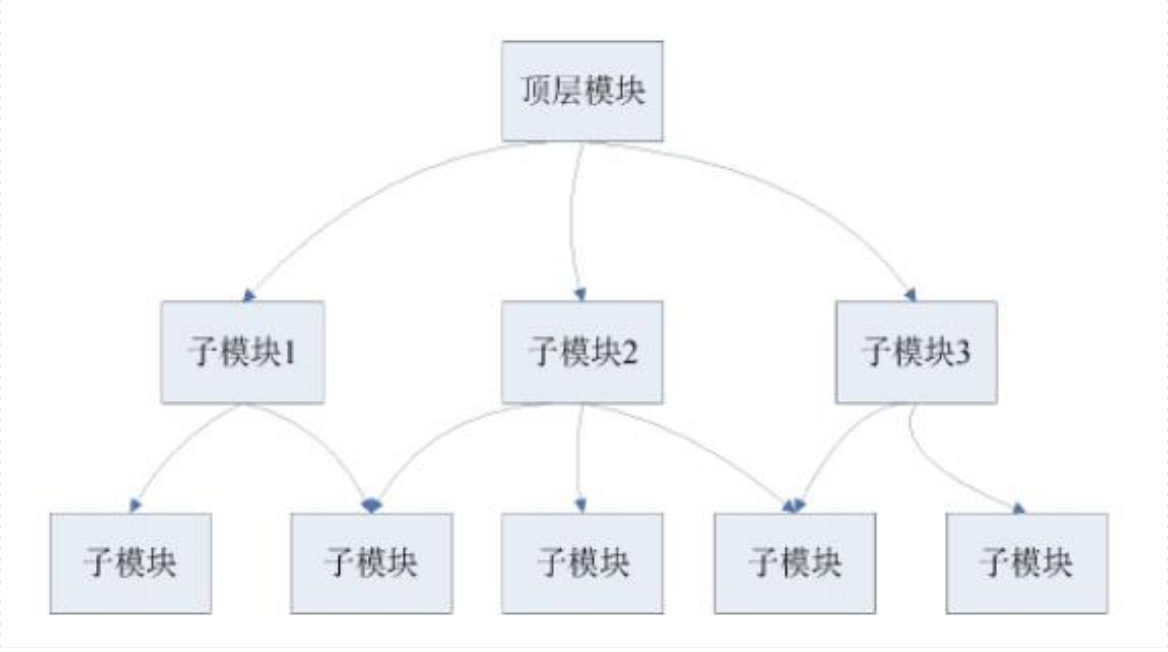
2.2 Verilog程序结构

2.2.1 Verilog程序结构

- 一个复杂电路系统的完整VerilogHDL模型由若干个Verilog HDL模块构成，每一个模块又可以由若干个子模块构成。
- 模块是Verilog的基本描述单位，一般来说是一个元件或者是一个更低层设计模块的集合。
- 模块是并行运行的，通常需要一个高层模块通过调用其他模块的实例来定义一个封闭的系统，包括测试数据和硬件描述。
- Verilog 程序源文件以.v作为文件扩展名。

Verilog语言的模块层次结构

- 一个复杂的系统，可划分成多个小的功能模块。



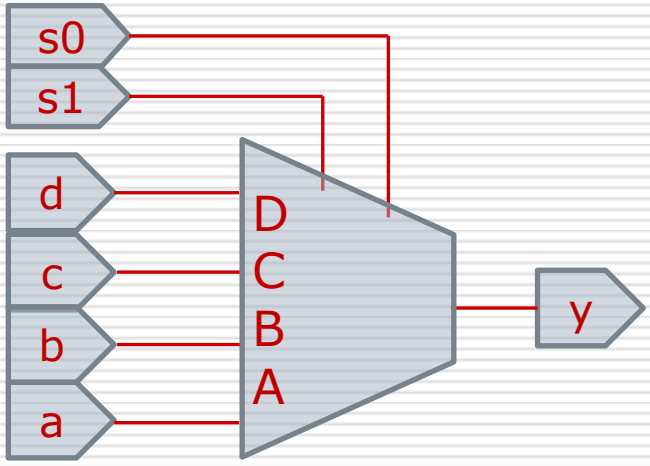
- 顶层模块可以对多个子模块调用，子模块也可以嵌套调用子模块。
- 一般情况下子模块的调用在5级以内。

- FPGA有许多逻辑门，Verilog的编程就是将这些门连接、组合起来成为具有功能性的电路；
- 每个子模块相当于一个芯片，实现编写的功能；
- 顶层模块把“芯片”连接构成一个完整的电路。

- 系统设计的三个步骤：
  - 把系统划分成模块；
  - 规划各模块的接口；
  - 对模块编程并连接各模块完成系统设计。

2.2.2 模块结构

【例2-1 】4选1多路选择器



- I/O端：
  - 输入端—a,b,c,d
  - 输出端—y
  - 控制端—s1,s0
- 功能：四选一

真值表	
控制端s1 s0	输出端y
00	a
01	b
10	c
11	d

Verilog程序示例

```
module MUX41a(a,b,c,d,s1,s0,y);
    input a,b,c,d;
    input s1,s0;
    output y;
    reg y;
    always@(a or b or c or d or s1 or s0)
        begin: MUX41 //块语句开始
          case ({s1,s0})
              2'b00: y<=a;
              2'b01: y<=b;
              2'b10: y<=c;
              2'b11: y<=d;
            default: y<=a;
          endcase
        end
endmodule
```

电路模块端口说明和定义段

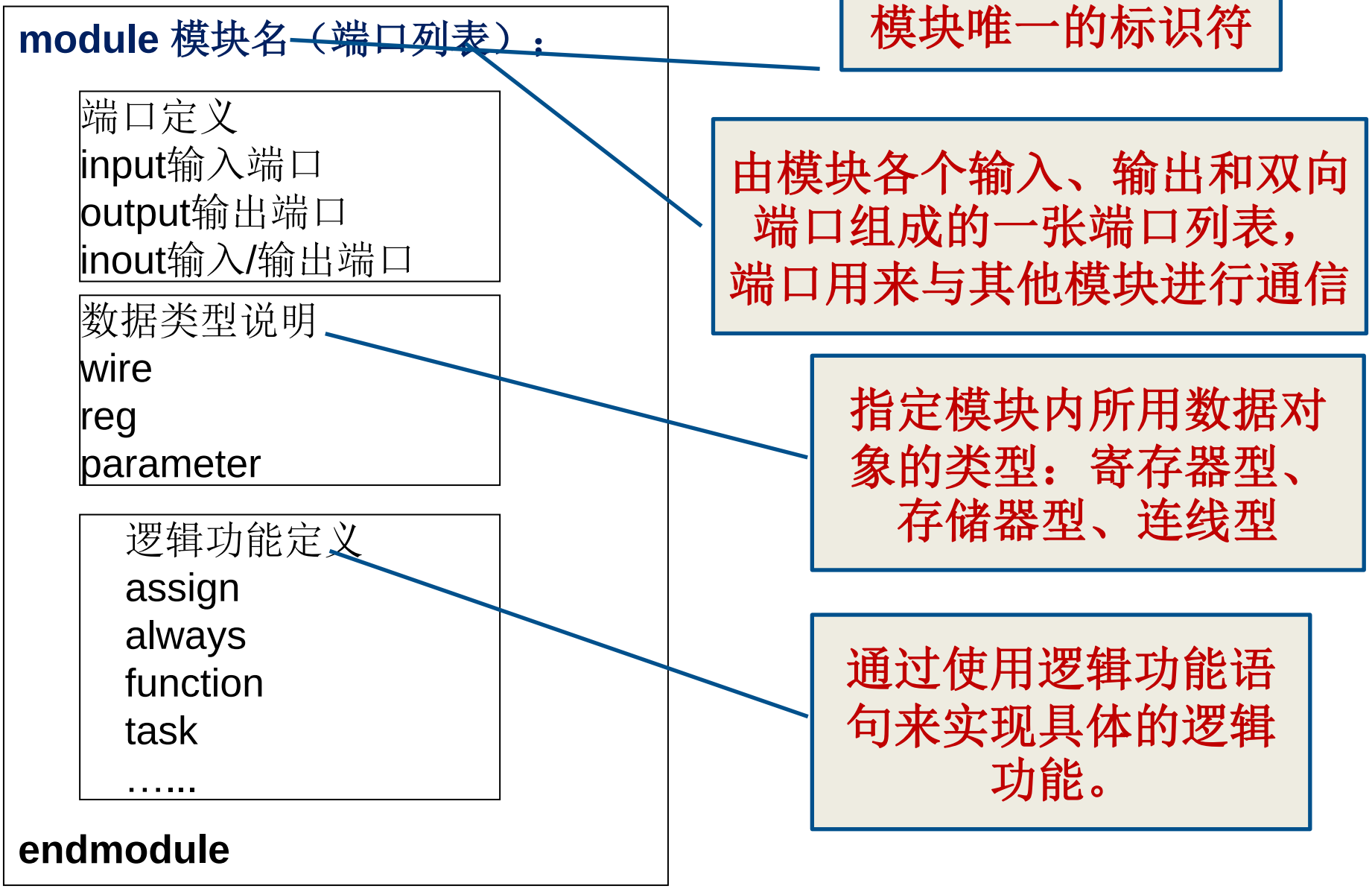
信号类型定义段

具体描述电路功能的Verilog语句

电路模块功能描述段

Verilog表述的可综合的完整电路模块

Verilog程序结构



说明

- Verilog HDL区分标识符中的大小写字母。
- Verilog HDL程序书写与C语言类似，一行可以写多条语句，也可以一条语句分成多行书写。
- 每条语句以分号结束。
- endmodule语句后不加分号。
- 白空（新行、制表符和空格）没有特殊意义。

（1）模块定义

- 模块定义格式：

```
module 模块名 (port_name1,…,port_namen);
….
endmodule
```

- 其中：
  - 模块名是模块的唯一标识。
  - port_name为端口名，以“,”分隔。
  - 模块在语言形式上是以关键词**module**开始、以关键词**endmodule**结束的一段程序。

- 特点：

① 模块是verilog设计中最基本的概念和基本单元，每个v设计的系统中都由若干模块组成。

② 模块的实际意义是代表硬件电路上的逻辑实体，实现某一特定功能。

③ 模块的描述方式有行为建模和结构建模之分。

④ 模块之间是并行运行的。

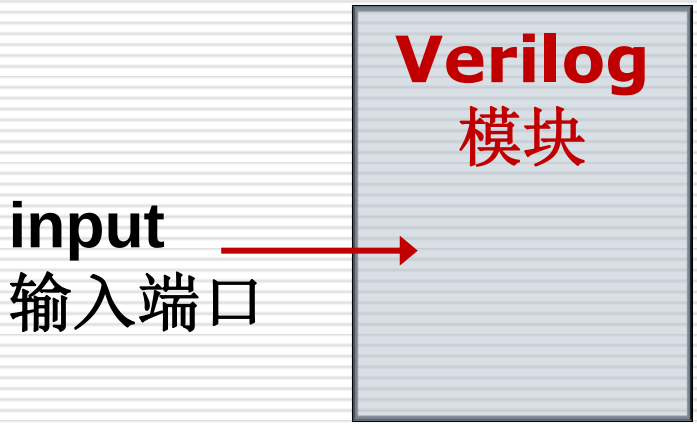
⑤ 模块是分层的，高层模块通过调用、连接低层模块的实例来实现复杂的功能。

⑥ 各模块连接完成整个系统需要一个顶层模块（top-module）。

（2）模块端口定义

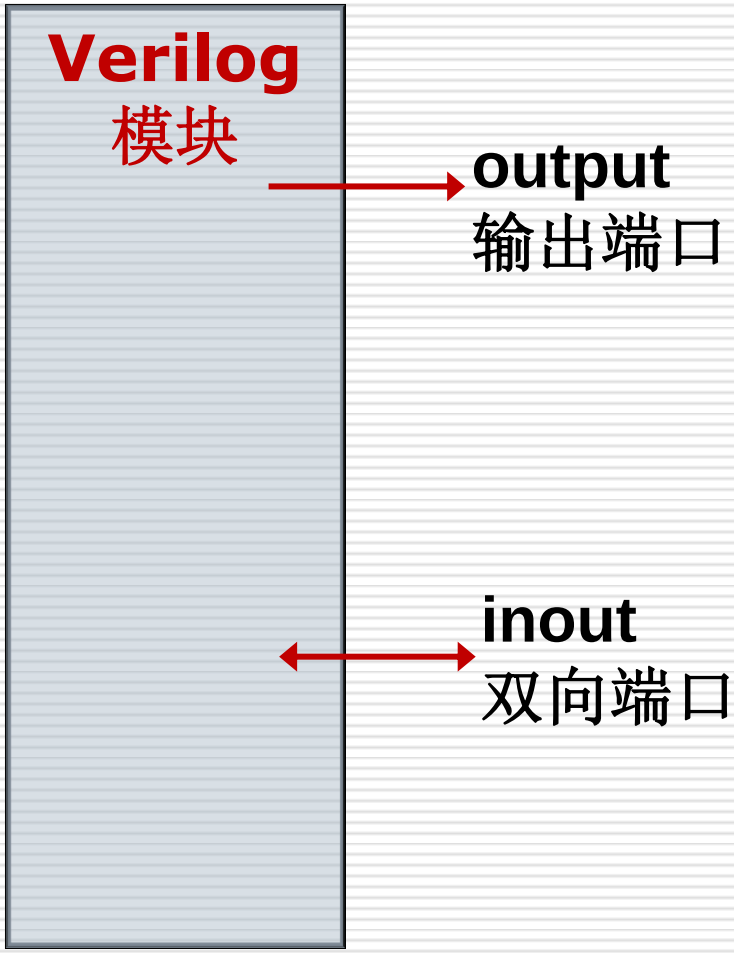
- 端口：模块与外部其它模块进行信号传递的通道（信号线或芯片引脚）。
- 定义格式：

  端口类型　**port_name, ...other_ports**;//端口列表
- 端口类型：



① input：定义通道为单向只读模式，规定数据只能由此端口被读入模块实体。

端口类型：



② output：定义通道为单向输出模式；规定数据只能通过此端口从模块实体向外流出，或可将模块中的数据向此端口赋值。

③ inout：定义通道为I/O双向端口，规定数据只能通过此端口从模块实体向外流出，或可将模块中的数据向此端口赋值。

【例2-2】端口说明实例

```
module addr(cout,sum,ina,inb,cin)
input cin;
input[7:0] ina,inb;
output[7:0] sum;
output cout;
……
endmodule
```

模块名：addr

cin：1位输入端口

ina、inb：8位输入端口

sum：8位输出端口

cout：1位输出端口

① 在声明端口I/O时，还要声明其数据类型；默认的端口类型为wire类型，端口可被显式地指定为网络类型。

② 网络或寄存器必须与端口说明中指定的宽度相同。

ANSI C风格的端口说明

- Verilog-2001标准规定可将端口定义和信号类型声明放在一起。

```
module add(
   output reg [3:0] sum,
   output reg  c_out,
   input [3:0] a,b,
   input c_in
 );
    ……
endmodule
```

（3）信号类型声明

- 模块中所用到的所有信号（包括端口信号、模块内部的网线信号或节点信号等）都必须进行数据类型的定义。
- Verilog 语言提供的各种信号类型，分别模拟实际电路中的各种物理连接和物理实体。

【例】信号类型声明

```
reg cout; //cout的数据类型为reg型
reg[7:0] out; //out的数据类型为8位reg型
wire A,B,C,D,F //A,B,C,D,F为wire型
```

（4）逻辑功能定义

- 描述模块的电路结构和具体功能。
- 方式：

① 赋值语句实现逻辑定义

```
assign F=~((A&B)|(~(C&D)));
```

② 在always过程赋值

- always (A,B,C,D)

```
        begin
            F=~((A&B)|(~(C&D)));
        end
```

- always块来描述时序逻辑电路

always过程实现计数器的过程。

```
always @(posedge clk, posedge reset)
begin
    if(reset) out<=0;
    else out<=out+1;
end
```

系统复位后，计数器回零

在**clk**上升是进行加**1**计数

③ 模块调用（实例化）

➢一个Verilog模块可以被任意多个其他模块调用。

④ 函数和任务调用

类似于模块调用。

二者差别：

•一个模块代表拥有特定功能的一个电路块，当被其他模块内调用一次，其所表示的电路结构就会在调用模块代表的电路内部被复制一次。

•模块调用不能像函数调用一样具有“退出调用”的操作，因为硬件电路结构不会随着时间而发生变化，被复制的电路块将一直存在。

2.3 Verilog语言要素

① 注释

在Verilog HDL中有两种形式的注释：

- /*第一种形式：可以扩展至多行* /
- //第二种形式：在本行结束。

② 空白符

包括空格字符（\b）、制表符(\t)、换行符(\n)以及换页符。综合时空白符被忽略。

③ 关键字（保留字）

Verilog HDL语言内部使用的词。所有关键字都使用小写字母。

2.3 Verilog语言要素

④ 标识符

Verilog HDL中的标识符(identifier)可以是任意一组字母、数字、$符号和_(下划线)符号的组合，是赋予一个对象唯一的名字。标识符的第一个字符必须是字母或者下划线。另外，标识符是区分大小写的。标识符分为简单标识符、转义标识符、生成标志符、关键字。

⑤ 运算符

Verilog提供了丰富的运算符。

2.4 Verilog常量

- 常量constants：

  在程序运行过程中，不能发生变化的量。

- Verilog HDL中常量类型：
  - 整数型；
  - 实数型；
  - 字符串型。

2.4.1 整数型integer

- 整数型常量描述方式：

① 十进制格式：

32　十进制数32

-15　十进制数-15

② 基数表示法：

+/-size′basevalue

- size：数据的二进制位数，采用十进制正整数。
- base：数的进制，以字母D、B、O、H表示。
- value：基于进制的数字符号序列。
- ※表示法中的字母不区分大小写。

【例2-3】基数表示法实例

表示法	说明
5'O37	5位B八进制数，即11111
4'D2	4位B十进制数，即0010
4'B1x_01	4位B二进制数，即1x01
7'Hx	7位Bx(扩展的x)，即xxxxxxx
4'hZ	4位Bz(扩展的z)，即zzzz
4'd-4	非法，数值不能为负
8_'h_2A	在位长和字符之间,以及基数和数值之间允许出现空格
3`_b001	非法，'和基数b之间不允许出现空格
(2+3)'b10	非法，位长不能够为表达式

说明：

- 为提高可读性，可以在较长的数据间加下划线 如：10′b10_1010_1011
- 若不注明位宽和进制（或仅用D标明），都是十进制数，默认32位。如：

  123 等价于 32′h0000_007B

  ′D12 等价于32′h0000_000C。
- 若不注明位宽，仅标明进制，数据宽度即为此进制规定的数值实际位数。如′o466=9′o466
- 数据的正负号放到最左边（宽度前），负数为补码形式。-7′d30=7′H62, -5=32′hFFFFFFFB。

- 符号x表示状态未知，z(?)表示高阻态。
- 若定义的位宽比实际要短，则多余位数从高位被截断。若长，则高位补零。
- ✓ 补零填充

  10'b10，左边添0占位,0000000010

  10'bx0x1，左边添x占位, xxxxxxx0x1
- ✓ 数据截断

  3'b1001_0011与3'b011相等

  5'H0FFF与5'H1F相等

2.4.2 实数型

- 描述中必须有小数点，且小数点两侧都有数字。

① 十进制计数法

2.0，5.678，11572.12，0.1 正确

2. 非法，小数点两侧必须要有1位数字

② 科学计数法

23_5.1e2其值为23510.0；忽略下划线

3.6E2其值为360.0e(与E相同)

5E-4其值为0.0005

- Verilog语言定义的实数可通过四舍五入被隐式地转换为整数。

2.4.3 字符串常量

- 字符串：双引号内的字符序列，用ASCⅡ码表示的一维字符数组。
- 作用：仿真时显示一些相关信息，或指定显示的格式
- 字符串被用作Verilog表达式或复制语句的操作数，则字符串被看做无符号整数序列。
- 字符串变量是reg型变量，位宽为字符数乘以8。

如存储字符串”Hello World!”，需要8*12（即96位bit）宽的寄存器。（无需结束符号/）

【例2-4】字符串操作

mo

量

s

```
$display("%s is stored as %h",stringvar,stringvar);
stringvar={stringvar,"!!!"};
$display("%s is stored as %h",stringvar,stringvar);
```

en

输出结果为：
Hello World is stored as
00000048656c6c6f20776f726c64
Hello World!!! is stored as
48656c6c6f20776f726c64212121

在操作过程中，
如果声明的字符串变量位数大于字符串实际长度，则在赋值操作后，字符串变量的左端（即高位）补0。
如果声明的字符串变量位数小于字符串实际长度，那么字符串的左端被截去，这些高位字符就丢失了。

特殊字符：

在某些字符之前加上一个引导性的字符（转意字符），只能用于字符串中。

特殊字符	意义
\n	换行符
\t	Tab键
\\	符号\
\\”	符号”
\ddd	3位八进制数表示的ASCII值（0<<d<<7）
%%	符号%

2.5 Verilog数据类型

- 数据类型：表示数电硬件中的物理连线、数据存储对象和传输单元等。
- 数据变量类型的基本种类：
  - net型
  - register型（1995标准）/variable型（2001标准）
- Verilog HDL四种基本的值：

① 0：逻辑0或“假”状态；

② 1：逻辑1或“真”状态；

③ x/X：未知状态，对大小写不敏感；

④ z/Z：高阻状态，对大小写不敏感。

2.5.1 net类型

- net型变量常被综合为硬件电路的物理连接，需要门和模块的驱动。
- net型数据是指输出值紧随输入值的变化而更新，（数值由驱动元件决定），若没有驱动元件连接，则其默认值为高阻（z）。
- 应用场合：
  - 用来表示assign引导的组合电路描述；
  - 在结构描述中将其连接到一个门元件或模块的输出端。

（1）net类型定义

- 声明格式：

  net rang delay 数据名列表;

- 其中：

  - net：指定网络型数据的类型。
  - range：指定数据宽度。若省略，默认数据类型为1位的标量；否则该项指定数据的矢量形式。
  - delay：指定仿真延迟时间。
  - 数据名列表：net变量名称，可定义多个变量，用逗号分开。

常用的net型变量

类型	功能	可综合性
wire， tri	标准内部连接线	√
supply1， supply0	电源和地	√
wor， trior	多驱动源线或	
wand， triand	多驱动源线与	
trireg	能保存电荷的net	
tri1， tri0	无驱动时上拉/下拉	

① wire和tri类型

- wire用于连接单元，是最常见的网络类型。
- Verilog程序模块中的I/O信号类型默认为wire
- 作为方程式的输入，或作为assign语句、实例元件的输出。
- tri和wire功能和使用方法完全一样，仅为增加程序的可读性，表示信号综合后具有三态的功能。

【例2-5】网络的声明

```
tri [15: 0] bus;      // 16位三态总线
wire [0: 31] w1, w2;  // 两个32位wire，MSB为bit0
```

② supply0和supply1类型

supply0用于对“地”建模，即低电平0；

supply1网用于对电源建模，即高电平1。

【例】supply0和supply1网络类型描述

```
supply0 Gnd,ClkGnd;
supply1 [2:0] Vcc;
```

2.5.2 reg类型

- 具有寄存特性，即接受下一次赋值前保持原值不变。（具有记忆功能）
- **register（variable）**型变量必须放在过程语句中，通过过程赋值语句完成赋值操作。
- **reg**型变量与**wire**数据的主要区别：

类型	功能	用途	特点
reg	常用寄存器变量		可综合
integer	32位有符号实型变量		
time	64位无符号整型变量	对模拟时间的存储和处理	用于仿真
real	64位有符号实型变量	实数寄存器	

（1）reg类型变量

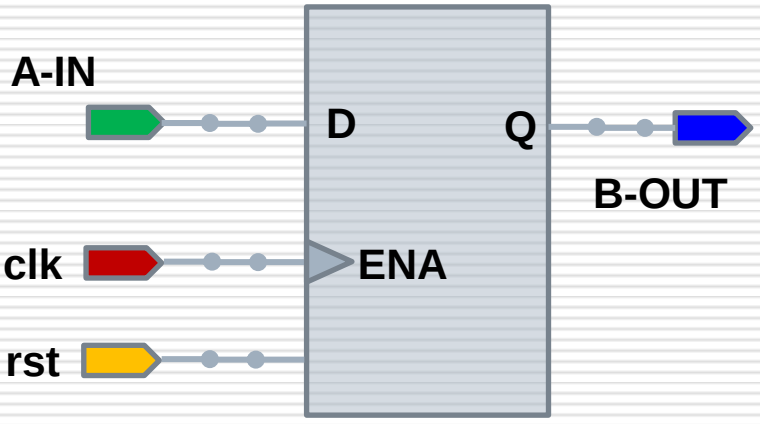
- 声明格式：

  reg range reg变量列表;

- 其中：
  - reg：表示寄存器类型；
  - range：矢量范围，采用[MSB：LSB]格式，只对reg类型有效；
  - reg变量列表：reg型变量名，一次可定义多个reg型变量，使用逗号分开。

【例2-6】reg型变量的声明及使用


```
module mult(clk, rst, A_IN, B_OUT);
    input clk,rst,A_IN;
    output B_OUT;
    reg Q = 1'b0;
    always @(posedge clk or posedge rst)
      begin
        if (rst)  Q<= 1'b1;
          else  Q<= A_IN;
        end
     B_OUT <= Q;
endmodule
```



（2）integer变量

- 整型变量常用做循环控制变量。
- 32位二进制补码形式的有符号数。
- 说明：integer与reg型数据在实际意义上相同，但reg数据被当做无符号数来处理。
- 【例】整数变量的声明

```
integer i,j;
integer[31:0] D;
```

不能进行位操作

※ 注意：虽然interger有位宽度的声明，但是不能作为位向量访问。D[6]和D[16:0]的声明都是非法的。在综合时，integer型变量的初始值是x。

（3）time型变量声明

time型变量与integer变量类似，只是它是64位的无符号数。时间型变量主要用于对仿真时间的存储与计算处理，常与系统函数$time一起使用。

（4）real型变量声明

real型数据在机器码表示法中是浮点型数值，可用于对延迟时间的计算。实数型变量是不可综合的。

2.5.3 向量

- 向量：线宽大于1位的变量，用于将已声明过类型的元素组合成多维的数据对象。
- 向量声明：

  数据类型 位宽 变量名 维度;

  - 位宽：在声明的数据标识符后面指定元素的地址范围，由常量表达式[msb_:lsb_]说明，msb_代表范围的左侧值，lsb_代表范围的右侧值。lsb_值可以大于、等于、小于msb_值。
  - 一个维度代表一个地址范围。数组可以是一维向量，也可以是多维向量。

- net和reg型向量若没有被声明为有符号量或链接到声明为有符号的数据端口，则该向量被隐含当做无符号的量。
- 可通过单独的赋值语句为一个数组元素赋值。
- 位选择和域选择：在表达式中可任意选中向量中的一位或相邻几位，称为位选择（bit-select）和域选择（part-select）。

多维向量的声明实例

实例1—一维数组

wire [7:0] array2 [0:255]

声明一维的256个wire型数据，数据宽度8位。

实例2—二维数组

reg [63:0] regarray2 [255:0][7:0]

声明一个256x8的reg型数据，数据宽度是64位。

实例3—三维数组

wire [7:0] array3 [0:15][0:255][0:15]

声明表示一个三维的向量，表示16个256x16的wire型数据，8位数据宽度。

2.6 Verilog名字空间

- 全局名字空间（两类）

① 定义名字空间：包括所有**module**、**primitive**、**marcomodule**的定义。一旦名字被用于定义一个模块、基本元件或宏模块，则它将不能再用于声明其他模块、基本元件或宏模块，在定义名字空间具有唯一性。

② 文本宏名字空间：具有全局性。由于文本宏名由重音符号（`）引导，因此它与别的名字空间有明显的区别。文本宏名的定义逐行出现在设计单元源程序中，它可以被重复定义，即同一宏名后面的定义将覆盖其先前的定义。

- 局部名字空间（五类）
  - 包括：
  - block（语句块）
  - module（模块）
  - port（端口）
  - specify block（延时说明块）
  - attribute（属性）。
  - 某个名字在这5个名字空间中的任意一个空间内被定义，它就不能在该空间中被重复定义（即具有唯一性）

2.7 Verilog 语言表达式

- 表达式：采用操作符将操作数联合起来并通过运算可得结果，在出现数值的任何地方使用。

2.7.1 操作符

- 按功能分：算术操作符、关系操作符、相等操作符、逻辑操作符、按位操作符、归约操作符、移位操作符、条件操作符、连接和复制操作符；
- 按运算符所带操作数的个数可分：单目操作符、双目操作符和三目操作符

操作符

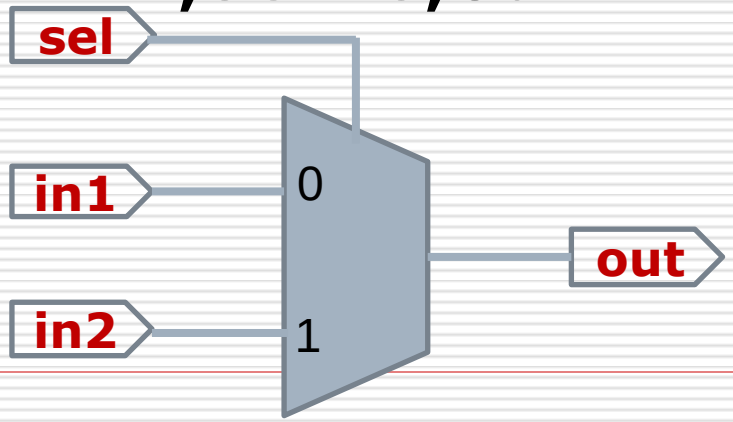
符号	意义	符号	意义	符号	意义	符号	意义
+	一元加	~	一元按位求反	<<	左移	===	全等
-	一元减	!	一元逻辑非	>>	右移	!==	非全等
+	二元加	|	归约或	<	小于	&	按位与
-	二元减	~|	归约或非	<=	小于等于	^	按位异或
*	乘	&	归约与	>	大于	^~ (~^)	按位异或非
/	除	~&	归约与非	>=	大于等于	|	按位或
%	取模	^	归约异或	==	逻辑相等	&&	逻辑与
?:	条件操作符	^~ (~^)	归约异或非	!=	逻辑不等	||	逻辑或

- 归约操作符
  - 归约操作符是一元操作符，是将单一操作数的所有位当成多个1bit的操作数进行操作符所规定的运算，并产生1位结果。
  - 例如：a=4’b1101，则 &a= 1&1&0&1 =0
  - 按位运算是二元操作符，是将操作符两边的操作数进行操作符所规定的运算 。
  - 例如：a & b(a=1,b=1),出来结果为1。
- 位拼接运算符{}
  - 将两个或多个信号的某些位拼接起来。
  - 如考虑进位的加法器

```
assign {cout,sum}=ina+inb+cin;
```

- 条件运算符？：
  - 信号=条件?表达式1:表达式2;
  - 当条件成立时，信号取表达式1值，否则取表达式2值。
  - 如二选一选择器

```
out=sel?in2:in1;
//sel=1,out=in2;sel=0,out=in1
```



2.7.2 延迟表达式

- 延迟表达式的格式：用圆括号括起、用冒号分隔开的三个表达式；表达式之值依次代表最小、典型、最大延迟时间值。

【例】延迟表达式的使用。

(a:b:c)+(d:e:f)

表示：

最小延迟值为a+d的和，

典型延迟值为b+e的和，

最大延迟值为c+f的和