3 Verilog

1. **Verilog HDL**

是一种**硬件描述语言**，用于从**算法级、RTL 级、门级**到**开关级**的多种**抽象设计层次**的**数字系统建模**。

✧ **行为级描述：**数据结构和过程**类似 C**；用于描述**算法级**和 **RTL 级**的 Verilog 模型。

✧ **结构级描述：**用于描述**门级**和**开关级**电路；特点：支持**门级延时**信息和**驱动能力**等的描述。

VHDL 侧重于系统级描述，从而更多的为**系统级**设计人员所采用；

Verilog 侧重于电路级描述，从而更多的为**电路级**设计人员所采用。

2. **Verilog HDL 设计入门**

模块（module）

模块是 Verilog 的**基本描述单位**

模块的**定义**从关键词 **module 开始**，到关键词 **endmodule 结束**

每条 Verilog HDL **语句以分号“；”作为结束**

模块的基本结构

⑴、**模块定义行**

以 **module 开头**

接着给出所定义模块的**模块名**

括号内给出**端口名列表**（端口名等价于硬件中的**外接引脚**，模块**通过这些端口与外界发生联系**）

以**分号结束**

⑵、**端口类型说明**

端口类型只有 **input、output、inout** 三种

⑶、**数据类型说明**

支持的**数据类型**有**连线类（wire）**和**寄存器（reg）**类两个大类

一位宽的 wire 类可被缺省外，其它凡将**在后面的描述中出现的变量都应给出相应的数据类型说明**

⑷、**描述体部**

具体展开对模块的**描述**

⑸、**结束行**

用关键词 **endmodule 标志**模块定义的**结束**

它的**后面没有分号**

行为描述与结构描述

✧ **行为描述**（Behavior）描述**行为**或**功能**特性

✧ **结构描述**（Structure）描述通过什么样的**结构方式将不同的实体连接起来**用以实现所要求的**行为或功能**。

测试与仿真

✧ **测试平台（Test Bench）：**在输入端口加入测试信号，从输出端口检测其输出结果是否正确。

✧ **通常将需要测试的对象**称之为 **DUT**（Device Under Test）。

✧ **测试模块：**要调用 **DUT**；包含用于测试的**激励信号源**；能够实施**对输出信号的检测**，并**报告检测的结果**。

过程语句

✧ **Initial：**只**顺序**地**执行一次**；**没有触发条件**。

✧ **Always：**

当顺序**执行到最后一条语句后**，会**自动返回**到第一条语句**重新开始执行**，是一条**没有穷尽的循环语句**

往往带**有触发条件**

3. **Verilog HDL 基础知识**

基本词法定义

✧ **空白符**

- 空白符是以下**几种字符与控制符的总称：**①空格、②TAB 键、③换行符、④换页符。
- **空白符起分隔符的作用：允许**在**一行内写多条 Verilog HDL 语句**

✧ **注释行**

- **单行注释：**以“//”开始到行末结束，**不允许续行**
- **多行注释：**以“/*” 开始，到“*/”结束，可以**跨越多行**，但**不允许嵌套**

- **注释行中的内容只是为了便于阅读理解**
- 在必要的地方增加适当的注释说明，在 HDL 编程中尤为重要
- **有些操作控制命令以注释行的方式出现在 HDL 描述中**，它**不影响 HDL 的仿真**，但**其它工具可以识别这些控制命令**

✧ **四种逻辑状态**

- Verilog HDL 需要用**数字或字符**去表达在数字电路中**存储与传送**的**逻辑状态。**

0	**逻辑零、逻辑非、低电平**	x 或 X	**不确定的逻辑状态**
1	**逻辑 1、逻辑真、高电平**	z 或 Z	**高阻态**

✧ **整数及其表示**

- **整数可以表示成**：十进制、十六进制、八进制、二进制等
- **表示方法有两种：**

① 直接由 **0~9 的数字串组成的十进制数**

② +/-<位宽>‘<基数符号><按基数表示的数值>

位宽是指所要表示的整数用**二进制展开时所需的二进制位的个数**（bit 数），其**值用十进制数表示**

位宽在表示中**可以被缺省**

✧ **基数符号**

数制	基数符号	合法的表示值	数制	基数符号	合法的表示值
二进制	b or B	0, 1, x, X, z, Z, ?, _	十进制	d or D	0~9, _
八进制	o or O	0~7, x, X, z, Z, ?, _	十六进制	h or H	0~9, a~f, A~F, x, X, z, Z, ?, _

- 问号**“?”是高阻态 z** 的另一种表示
- 下划线**“_”**的引入**只是为了增加可读性**
- 在数值表示中，**左边为最高有效位**（MSB），**右边为最低有效位**（LSB）
- 在**二进制表示**中，**x, z 只代表相应位的逻辑状态**；在**八进制表示**中，**一位的 x, z 代表的是三个二进制位都处于 x 或 z 态**，在**十六进制**中则对应**代表四个二进制位都处于 x 或 z 态**
- 对于**带符号的整数**，**正、负号**的表示应**在最左边**

基数表示实例

数值表示	位宽	数制	等效二进制值及其解释	数值表示	位宽	数制	等效二进制值及其解释
10	缺省	十进制	0...0_1010(32位或以上)	6'hf3	6 位	十六进制	11_0011,高位部分被舍去
4ac	缺省	十进制	非法	6'hf	6 位	十六进制	00_1111,高位部分由0补足
'h4ac	缺省	十六进制	0...0_0100_1010_1100(32位或以上)	3'b10x	3 位	二进制	10x
9'o671	9 位	八进制	110_111_001	12'h2x6	12 位	十六进制	0010_xxxx_0110
9'o-671	9 位	八进制	非法	6'hx	6 位	十六进制	xx_xxxx,高位部分由x补足

✧ **实数及其表示**

- 实数可以用**十进制数表示**，也可用**指数表示**。
- 在 verilog HDL 中，实数可以**参与的运算是受限制的**。
- 当实数**被转换为整数**时，是按**四舍五入**的方式进行的。

特殊字符表示	意义
\n	换行
\t	Tab键
\\	反斜杠\
\"	引号"
\ddd	由三位八进制数表示的ASCII值
%%	%

✧ **字符串**

- **定义：**为两个双引号“ ”之间的字符；字符串不允许跨行。
- 可通过**前导的控制键**（**反斜杠及百分号**）**引入一些特殊字符**

✧ **取名规则**

- 必须是由**字母**（a~z, A~Z）或**下划线开头**，长度**小于 1024 字符**的**一串字符序列**
- **后续部分**可以是**字母**（ a~z, A~Z ）、**数字**（0～9）、**下划线**、**$**
- 还可以是**反斜杠“\”开头**，并**以空白符结尾**的**任何字符序列**。但**反斜杠本身及空白符**都**不属标识符**组成部分

✧ **系统命令**

- 以$开头的标识符用以代表系统命令（系统任务与系统函数）
- $display
- $monitor
- $finish

✧ **关键词**

- Verilog HDL 语言**内部已经使用的词**称为**关键词或保留词**，应避免使用
- 所有的**关键词都是小写**
- **always，and，assign，attribute，begin，buf，case，default，else……**

数据类型

✧ **连线类型和寄存器类型**

- **驱动方式**（或**赋值方式**）不同
- **保持方式**不同
- 对应**硬件实现**不同

✧ **wire 连线类型**

- 对应**硬件电路中**的**物理信号连接**
- **驱动有两种方式：**
  在结构描述中**把它连接到一个门或模块的输出端**
  用**连续赋值语句 assign 对其进行赋值**
- **没有电荷保持作用**，当**没有被驱动**时，将**处于高阻态 z**

连线类型	连线功能
wire, tri	标准连线（缺省）
wor, trior	多重驱动时，具有线或特性的连线
wand, triand	多重驱动时，具有线与特性的连线
trireg	具有电荷保持特性的连线
tri1	上拉电阻（pullup）
tri0	下拉电阻（pulldown）
supply1	电源线，逻辑1
supply0	电源线，逻辑0

补充

- 连线主要**出现在**模块的**结构描述中**，对应硬件电路中物理信号连接。
- 在对连线进行描述时，必须**用连线类型定义语句进行说明**，当说明**被缺省时**，表示的是**位宽为 1bit** 的 wire 型连线。Wire 是标准的，不附带其它逻辑功能的连线。
- **Tri 与 wire** 的功能是**完全一致的**。
- **Wire 与 wor 以及 wand** 三者之间的**差别**体现在**有多重驱动时**连线所**具有的不同逻辑特性**

✧ **reg 寄存器类型**

寄存器类型	功 能 说 明
reg	用于行为描述中对寄存器类的说明，由过程赋值语句赋值
integer	32位带符号整型变量
real	64位浮点、双精度、带符号实型变量
time	64位无符号时间变量

- 对应的是具**有状态保持作用**的硬件电路元件，如触发器、锁存器等
- 驱动可以通过**过程赋值语句实现**
- 过程赋值在接受下一次的过程赋值之前，将**保持原值不变**
- 当寄存器类型**没有被赋值前**，将**处于不定态 X**
- 所有**寄存器类的量**，都有“寄存”性，即在**接受下一次赋值前**，将**保持原值不变**。
- 所有寄存器类都**必须给出类型说明**（无缺省状态）。
- 寄存器类的量，必须通过**过程赋值语**句进行赋值。
- **Integer、real、time** 都是**纯数学的抽象描述**，不对应于任何具体的硬件电路实现。

✧ **标量与矢量**

- 线宽只有一条的连线，以及位数只有一位的寄存器称之为标量。
- 线宽大于一条的连线，或位数大于一位的寄存器称之为矢量。
- **矢量的说明：**矢量的范围由括在方括号中的一对数字表示，中间一个冒号相隔。形式为：[msb:lsb]。

参数定义语句 parameter

✧ 用于对**延时、线宽、寄存器位数**等**物理量**的定义

✧ 用一个**文字参数**来**代替一个数字量**

✧ **优点**：增加描述的**可读性**；为以后的**修改带来方便**。

✧ **形式描述：参数定义表项**给出**具体的各个参数与数字量之间所谓对应关系**，相互间用**逗号“,”相隔**

宏替换 `define

✧ 宏替换的**形式描述: `define**<宏名><进行**宏替换的文本内容**>

✧ 宏替换是在编译时告知编译器，**用宏替换定义中的文本内容来直接替代模块描述文件中出现的宏名**。

✧ **一条宏定**义语句**只能定义一个宏替换**，且**定义结束时无分号**。

✧ 宏替换定义**本身以及用到宏替换的地方**必须**有撇号“`”作开头**

模拟时间定标

✧ 对**模拟器**的**时间单位**及**时间计算的精度**进行**定标**

✧ **定义：`timescale<计时单位>/<计时精度>**

- **计时单位**与**计时精度**都由**整数**及相应的**时间单位**两部分组成
- **时间单位：**S、ms（毫秒）、us（微秒）、Ns(纳秒) 、 ps（微微秒）、Fs（毫微微秒）
- **时间精度：延迟时间**的**最小分辨率**

✧ 计时单位**必须大于等于**精度单位

✧ 对 **timescale** 的定义**必须在模块描述的外部进行**

✧ 模拟器允许**对不同模块定义不同的时标**，但**以最小的精度进行模拟计算**

运算符

✧ **运算符的分类**

- **单目运算符：**只有**一个操作数**，且运算符位于操作数的**左边**
- **双目运算符：**有两个操作数，各位于运算符的**两边**
- **三目运算符：**属于这一类的**只有条件运算符（? :）**一个

✧ **运算符的优先级顺序**

运算符优先级顺序	
! ~ + - (unary)	最高优先级
* / %	
+ - (binary)	
<< >>	
< <= > >=	
== != === !==	
& ~&	
^ ~^	
| ~|	
&&	
||	
? :	最低优先级

运算符分类	所含运算符
算术运算符	+, -, *, /, %
位运算符	~, &, |, ^, ^~ or ~^
缩位运算符	&, ~&, |, ~|, ^, ^~ or ~^
逻辑运算符	!, &&, ||
关系运算符	<, >, <=, >=
相等与全等运算符	==, !=, ===, !==
逻辑移位运算符	<<, >>
连接运算符	{ }
条件运算符	? :

✧ **算术运算符：**

1、加法运算符； ＋，实现加法运算

2、减法运算符； －，实现减法运算

3、乘法运算符； ×，实现乘法运算

4、除法运算符； /， 实现除法运算

5、取模运算符； %，实现取模运算

- 在**算术运算操作中**，操作数是作为一个整体参与运算的，因而，如果某个操作数的**某一位为不定态**（x 值），则整个表达式的运算**结果也为不定态。**

✧ **位运算符：**

- **按位取反**运算符：～
- **按位与**运算符：&
- **按位或**运算符：|
- **按位异或**运算符：^
- **按位同或**运算符：^～或～^

✧ **缩位运算符：**

- **单目运算符，按位进行逻辑运算，最后结果为一位的二进制数**
- **缩位运算符包括：**

☺ & AND ☺ ^ XOR（异或） ☺ ~| NOR

☺ | OR ☺ ~& NAND ☺ ~^ or ^~ XNOR（同或）

✧ **逻辑运算符：**

- 逻辑**与**运算符：**&&**（双目运算符）
- 逻辑**或**运算符： || （双目运算符）
- 逻辑**非**运算符： ! （单目运算符）

A = 6; A && B → 1 && 0 → 0

B = 0; A || !B → 1 || 1 →1

C = x; C || B → x || 0 → x

✧ **关系运算符：**

1、小于：< 2、大于：> 3、小于等于：<= 4、大于等于：>=

- **结果是一位值：0，1 或 x**

✧ **相等与全等运算符：**

- **共有四种：**

  1、相等 运算符：==　　4'b 1z0x == 4'b 1z0x → x

  2、不等 运算符：!=　　4'b 1z0x != 4'b 1z0x → x

  3、全等 运算符：===　　4'b 1z0x === 4'b 1z0x → 1

  4、不全等运算符：!==　　4'b 1z0x !== 4'b 1z0x → 0

- 都是**双目运算符，**得到的**结果是 1 位的逻辑值**
- 相等算符：**逐位比较**两个操作数相应位的值是否相等，**只有当每一位都相等时，相等关系才满足**。如果任何一个操作数中的**某一位存在不定态或高阻态，则将得到一个不定态的结果**。
- 全等算符：将不定态或高阻态看作是逻辑状态的一种而参与比较

==	0	1	X	Z
0	1	0	X	X
1	0	1	X	X
X	X	X	X	X
Z	X	X	X	X

===	0	1	X	Z
0	1	0	0	0
1	0	1	0	0
X	0	0	1	0
Z	0	0	0	1

✧ **逻辑移位运算符：**

1、逻辑左移：<<　　**2**、逻辑右移：>>

- Result is **same size** as first operand, always **zero filled**

  例：a = 4'b1010;　d = a >> **2**;　// d = 0010　　c = a << **1**;　// c = 0100

✧ **连接运算符：**

- 将两组或两组以上的信号**用大括号括起来，拼接成一组新的信号**。
- 对于一些**重复信号的连接**，可以用它的**简化表示方法{n{a}}**，表示**将信号 a 重复连接 n 次。**
- Operands must be sized !!（**数的进制相同**）

✧ **条件运算符：**

- <**条件**运算符>?<条件为**真时的表达式**>:<条件为**假时的表达式**>：Assign out = **(sel == 0) ? a :b**

允许实型量参与的运算符　　不允许实型量参与的运算符

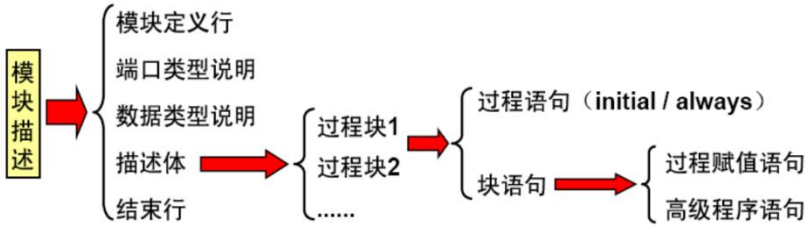
算符类型	运算符
算术算符	+ - * /
关系算符	> >= < <=
逻辑算符	! && ||
相等算符	== !=
条件算符	? :

算符类型	运算符
连接算符	{ }
取模算符	%
位运算符	~ & |
全等算符	=== !==
缩位算符	^ ~^ & ~& | ~|
移位算符	<< >>

4. Verilog 行为描述

Verilog 行为描述的构成框架

✧ **模块的基本结构**



过程块

- Verilog HDL 对**模块的行为描述**以**过程块为基本组成单位**，一个模块的**行为描述**由一个或多个**并行运行的过程块**组成。

✧ **过程块的形式定义：**

- **过程语句 @ (事件控制敏感表)**
  - ☺ 块语句**开始标识符：块名**
  - ☺ **块内局部变量说明**
  - ☺ 一条或多条**过程赋值或高级程序语句**
  - ☺ 块语句**结束标识符**

- 斜体表示的是**可缺省的部分**
- **过程语句**是指 **Initial 或 always**
- **事件控制敏感表只在 always 过程语句中**出现，用于**激活过程语句的执行**
- **块语句标识符**分 **begin-end**（**串行**块）与 **fork-join**（**并行**块）两类
- 块名和块内**局部变量说明均为可选项**

✧ **过程语句 Initial 与 always**

1、 都是**从模拟的 0 时刻开始执行**，但 **initial 过程语句**后面的块语句**沿时间轴只执行一次**，而 **always** 则**循环地重复执行**其后的块语句。

2、 initial 过程语句不带触发条件，因而**必定从模拟的 0 时刻开始**执行它后面的块语句；**always 过程语句**则**通常带有触发（激活）条件，只有当触发条件被满足时**，其后的**块语句才真正开始执行**。如果**触发条件被缺省，则认为触发条件始终被满足**。

3、 一个模块的行为描述中**可以有多个 initial 与 always 语句**，代表多个过程块的存在，它们之间**相互独立，并行运行**。

- **Initial 过程语句**
  - ☺ 在**实际的描述过程**中，**最经常地应用于测试模块**中**对激励向量的描述**。
  - ☺ 在**对硬件功能模块**的**行为描述**中，**仅在必要时给寄存器变量赋以初值**。
  - ☺ 是一条**主要面向模拟的过程语句**，通常**不为逻辑综合工具接受**。
- **Always 过程语句**
  - ☺ 在测试模块中一般**用于对时钟的描述**，但**更多地用于对硬件功能模块的行为描述**。
  - ☺ 功能模块的行为描述是由过程块构成的，每个过程块都要由过程语句所引导，因而**每个功能模块的行为描述中，至少存在一个 always 过程语句**。

块语句

- 由块标**识符 begin-end** 或 **fork-join 界定**的一组行为描述语言。
- 过程块所要完成的具体操作内容，都是通**过块语句中**所包含的**描述语句的执行而得以实现的**。

✧ **串行块 begin-end**

- 位于**串行块**中的各条**语句按串行方式顺序执行**
- **特点：**

1、 串行块中的每条**语句依据块中的排列次序，先后逐条顺序执行**。块中每条语句**给出的延时都是相对于前一条语句执行结束的相对时间**。

2、 串行块的**起始执行时间**就是**串行块中第一条语句开始执行的时间**。串行块的**结束时间**就是块中**最后一条语句执行结束的时间**。

3、 串行块的行为描述可以形象地理解为硬件电路中，**数据在时钟及控制信号作用下，沿数据通道的各级寄存器之间的传送过程**。

✧ **并行块 fork-join**

- 位于并行块中的**各条语句按并行方式同时执行**
- **特点：**

1、 并行块中的**每条语句都是同时开始并行执行**的，各条语句的**执行过程与语句在块中的先后排列顺序无关**。块中每条语句**给出的延时都是相对于并行块开始执行时的绝对时间**。

2、 并行块的**起始执行时间**就是**流程控制转入并行块的时间**，并行块中的**每条语句**都是**相对于这一时间同时开始执行的。**并行块的**结束时间**就是并行块中按执行时间排序**最后执行的一条语句结束的时间**。

3、 并行块的行为描述可以形象的**理解为硬件电路上电后**，各**电路模块同时开始工作的过程**。

✧ **有名块 Named-block**

- **块是可以取名的**，方法是在块语句**开始标识符后面加上一个冒号，之后给出一个名字**。
- **作用：①、**便于实现对块语句**执行过程**的**有效控制； ②、**允许在块语句**内部引入局部变量。**

赋值语句

- **位于过程块中的赋值语句**称之为**过程赋值语句**
- 过程赋值语句**只能对寄存器类的量进行赋值**

✧ 过程赋值语句的**两种延时模式**

- 依据**定时控制**在过程赋值语句中**的不同位置**，分为**外部模式**和**内部模式**两类
- **外部模式**
  - <**定时控制**><**寄存器变量**>=<**表达式**>;
  - 经“**定时控制**”所**确定的延时后**，再**计算右端表达式的值**，并**把结果赋给左端**的**寄存器变量**
  - **定时控制**

    1、**延时控制**：就是直接给出所需**延时的时间**，如：#delay a=b;

    2、**事件控制**：以符号**“@”**开头，后面**紧跟的是事件控制敏感表**

    ①@（信号名）：

    信号名所指定的**信号通常是一位的**，但**也可以是多位的**，并且**对数据类型没有限制；**只有**等到检测到信号名所确定的信号发生变化**（一般是指**上升沿**或**下降沿**）时，后面的赋值语句才被执行。

    ②@（posedge 信号名）

    只关心信号发生**上升沿跳变**情况的出现（Positive Edge）

    ③@（negedge 信号名）

    只关心信号发生**下降沿跳变**情况的出现（negative edge）

    ④@（敏感事件 1 or 敏感事件 2 or 敏感事件 3……）

    敏感事件是指**上面三类事件控制**中的任一种事件
- **内部模式**
- <**寄存器变量**>=<**定时控制**><**表达式**>;
- “**定时控制**”的表现形式与外部模式中的**完全一致**。
- **差别：**

  在**外部模式**中，**定时控制位于过程赋值语句之前**，直接**体现为对过程赋值语句执行时间的延期上**，只有当**延时时间被满足**，或**其它类型的激发条件被满足后**，过程赋值语句**才能被计算和赋值**。

  在**内部模式**中，**先**完成对表达式的**求值**过程，**再等待**延时的到期，等到**条件满足时**，再把前面求得的**结果赋给**左边的寄存器变量。

✧ **阻塞型过程赋值**

- **阻塞过程赋值算符：＝**
- 在**串行块**的执行过程中，**前一条**语句**没有完成赋值过程**之前，**后面**的语句**不可能被执行**。也可以解释为，由于前一条赋值语句的执行，使得**后面的赋值语句都被阻塞住了**。这样的赋值过程称之为阻塞型赋值过程，相应的赋值语句被称为阻塞型赋值语句（Blocking Assignment Statement）

✧ **非阻塞型过程赋值**

- **非阻塞型过程赋值算符：<=**
- 在一个**串行块**中，一条非阻塞型赋值语句的执行，并**不影响块中其它语句的执行**。
- 当一个**串行块中的语句全部由非阻塞型赋值语句构成时，对这个串行块的执行构成的解释与对并行块的解释完全是一样的**。

✧ **本质区别在于：**

- **非阻塞赋值语句**右端表达式**计算完后并不立即赋值给左端**，而是**同时启动下一条语句继续执行**，所有的右端表达式**在进程开始时同时计算**，计算完后 ，等进**程结束时同时分别赋给左端变量**。
- 阻塞赋值语句在**每个右端表达式计算完后立即赋给左端变量**。前一条语句的执行结果直接影响到后面语句的执行结果。
- **非阻塞赋值不能用于“assign”持续赋值中。阻塞赋值则既能用于“assign”连续赋值，也能用于“initial”和“always”等过程赋值中**。
- 对于**时序逻辑描述和建模**，应**尽量使用非阻塞赋值方式**。

说明：串行块中**非阻塞型赋值语句**的执行过程与**并行块**中**阻塞型赋值语句**的执行过程是**完全一致的**。

作为一种**可综合化设计**，通常**一个块语句中不允许同时**出现**阻塞型过程赋值语句**与**非阻塞型过程赋值语句**。

✧ **连续赋值语句**与**过程赋值语句**的**差别：**

①、赋值**对象不同：连续赋值语句**用于对**连线类变量**赋值；**过程赋值语句**完成对**寄存器类变量**的赋值。

②、赋值过程**实现方式不同：**

连续变量一旦**被连续赋值语句赋值后**，赋值语句**右端表达式中的信号有任何变化，都将随时反映到左端的连线变量中**。

过程赋值语句只要**在语句被执行到时，赋值过程才进行一次**，且赋值过程的具体执行时刻还受到定时控制及延时模式等多方面的影响

③、语句出现的**位置不同：**

连续赋值语句**不能出现在**任何一个**过程块中；过程赋值**语句则**只能出现在过程块中。**

④、语句**结构不同：**

连续赋值语句以**关键词 assign 为先导**，语句中的赋值算符**只要阻塞型一种形式；**

过程赋值语句**不需要相应的先导关键词**，语句中的赋值算符**分阻塞型和非阻塞型**两类。

⑤、**冲突处理方式不同：**

一条连线可被多条连续赋值语句同时驱动，最后的结果依据连线类型的不同**有相应的冲突处理方式；**

寄存器变量在**同一时刻只允许一条过程赋值语句对其进行赋值。**

高级程序语句

- 所谓**高级程序语句**，就是**直接借用高级语言中的程序设计语句**对模块进行描述
- 高级程序语句**只能出现在**对模块进行**行为描述的过程块中**。

✧ **分成三类**：

- **If-else 条件语句**
- **Case 语句**
- **循环语句**：**Forever** 循环语句；**Repeat** 循环语句；**While** 循环语句；**For** 循环语句。

✧ **If（条件表达式）块语句**

- 当**条件表达式成立**时，执行后面的块语句，当条件表达式不成立时后面的块语句不被执行；
- 一条**没有 else** 选项的 if 语句映射到硬件上，形成的**是一个锁存器。**
- **If**（条件表达式 1） **块语句 1**

  Else if（条件表达式 1） **块语句 2**

  ……

  Else if（条件表达式 n） **块语句 n**

  Else
- 表示**多路选择控制**
- 条件**判断有先后次序**，隐含着一种优先级关系

✧ **Case 语句**

- 当**多路选择的控制条件**集中在**某个变量**的变化上时，用 case 语句加以**表达显得更为方便与直观；**
- Case 语句**最适宜于对 CPU 的译码**等部件的描述以及对**有限状态机**的描述
- Case 语句分为 **case**、**casez**、**casex** 共三种表示方式
- **Casez** 与 **casex** 语句
  - 在 case 语句中，**敏感表达式**与**值项**之间的比较是一种**全等比较；**
  - 在 casez 语句中，如果**比较的双方有一边的某一位的值是 z**，那么**这一位的比较就不予考虑；**
  - 在 casex 语句中，如果**比较的双方有一边的某一位的值是 z 或 x**，那么**这一位的比较就不予考虑**。

```
Case（敏感表达式）
值1：块语句1
值2：块语句2
……
值n：块语句n
Default:块语句n+1
endcase
```

✧ **Forever 循环语句**

- Forever 循环语句的形式定义：Forever 块语句；
- Forever 循环语句是一个**永无止境的循环语句**，即**执行完块语句后，接下来再重新执行**，永不停止。

```
initial
  begin
    clock=0;
    forever #50 clock=~clock;
  end
```

✧ **Repeat 循环语句**

- Repeat 循环语句的形式定义：Repeat（循环次数计算表达式）块语句；
- **先计算出循环次数**表达式的值，并**将它作为循环过程重复执行次数的基值**被保存起来，接着**执行后面的块语句**，再接着**重新执行下一次的块语句操作**，直到循环执行次数被减为 0 时，结束整个循环过程。

✧ **While 循环语句**

- While 循环语句的形式定义：While（循环执行条件表达式）块语句；
- **先判断循环执行条件表达式是否为真**，如果是，执行后面的块语句，接着再判断循环执行条件表达式是否仍为真，**直到表达式的值为非真时，结束循环过程**。

✧ **For 循环语句**

- For 循环语句的形式定义：For（表达式 1；表达式 2；表达式 3）块语句
- **表达式 1** 只在**第一次循环开始前计算一次**，通常是为了**给循环控制变量赋初值**。
- **表达式 2** 是一个**循环结束控制的条件表达式**。
- **表达式 3** 往往是**给循环变量增值（或减值）**，然后再对循环结束控制的条件表达式 2 重新计算和判断，若条件表达式的值仍为真，则继续执行上述循环过程，直到**表达式 2 不成立时，循环过程被终止**。
- For 循环语句可以看作是 while 循环语句的一种**简约表示形式。**

```
begin
  表达式1;
  while(表达式2)
    begin
      循环体语句;
      表达式3;
    end
end
```

✧ **Disable 循环中断控制语句**

- Disable 语句的形式定义：Disable 块名或任务名
- Disable 语句的引入，是为了**中止正在进行中的循环或任务**。

✧ **Wait 语句**

- Wait 语句的形式定义：Wait（条件表达式）块语句或空语句
- 代表一种**等待状态**，当**条件表达式非真时，过程处于等待状态**，直到等到条件表达式为真时，等待状态才告结束。

Verilog 任务与函数

✧ **任务（task）的形式定义**

- task **任务名**；
- **端口与类型**说明；
- **局部变量**说明；
- 块语句
- endtask

1、任务的**定义与引用**都在一个 **module 模块内部**

2、任务的**定义与 module 的定义有些类似**，同样**需要进行端口说明与数据类型说明**。另外，任务定义的**内部没有过程块**，但在**块语句中可以包含定时控制部分**。

3、当任务**被引用时，任务被激活**。

4、**一个任务可以调用别的任务或函数。**

✧ **函数（function）的形式定义**

- function<**位宽**说明>**函数名**；
- **输入端口与类型**说明；
- **局部变量**说明；
- 块语句
- endfunction

✧ **任务与函数的异同**

1、 任务和函数的**定义和引用都位于模块的内部**，而**不是一个独立的模块**。

2、 **函数不能调用任务**，而**任务可以调用任何其它的任务或函数**。

3、 **任务可以没有输入变量**或有任何类型的 I/O 变量，而**函数允许有输入变量且至少有一个，输出则由函数名自身担当**。

4、 **函数通过函数名返回一个值，任务名本身没有值的意义**，只是实现某种操作，**传值通过 I/O 端口实现**。

5、 任务和函数的定义中，**内部都没有过程块。**

6、 **函数**还可以出现在**连续赋值语句 assign 的右端表达式**中。

Verilog 模块调用

- 模块调用是 Verilog HDL **结构描述的基本构成方式**
- 一个**硬件系统的描述中**，必定**有而且只能有一个顶层模块。**
- **有两类：**

1、**基本门调用**（Primitive Instantiation）；　　2、通常的 **module 模块调用**（Module Instantiation）。

- 模块**调用的最基本形式：模块名 调用名（端口名表项）**

✧ **说明：**

1、 **位置对应调用法**

按照**定义时确定的端口顺序**，结合设计中实际的连接关系，将模块定义时的端口名对应地用与之相连

的信号线名称所替代。

2、 **端口名对应调用法**

模块**定义时的端口名**和**调用时的实际连接信号名**之间一一对应关系被显式的表示出来。调用时端口名的排列顺序可以随意改变。

定义时的端口名（调用时与之相连的信号名）

3、 允许出现不连接的端口

允许**调用时**在**不连接的端口名的相应位置不提供相连的信号名**，但为保证端口名之间的对应关系，逗号不能省略。

5. Verilog 系统函数与编译向导

Verilog 系统任务与系统函数

- **系统任务**和**系统函数**是**面向模拟的、嵌入到 Verilog HDL 语句中**的**模拟系统功能调用**。
- **调用系统函数时会返回一个值**，而**调用系统任务**则**只完成某项任务，没有值的返回**。
- 依据实现**功能的不同**，可分成以下几类：

1、**输出控制**类**系统任务**：完成对**输出量的格式控制**，$display， $write， $monitor

2、**模拟时标**类**系统函数**：$time，$realtime

3、**进程控制**类**系统任务**：用于**对模拟进程的控制**，$finish， $stop

4、**文件读写**类**系统任务**：用于**控制对数据文件的读写方式**

5、其它类

✧ **系统任务$display 与$write**

- $display 与$write 属**输出控制类**系统任务
- **调用形式**
  - ☺ **$display**（“格式控制字符串”，输出变量名表项）
  - ☺ **$write**（“格式控制字符串”，输出变量名表项）
  - ☺ **输出变量名表项**就是指**要输出的变量**，各变量名之间**以逗号相隔。**
  - ☺ **格式控制字符串**的内容包括**两部分**：
  - · 需要与输出变量**一起在输出时一并显示的普通字符；**
  - · 对输出变量显示形式**进行控制的格式说明符。**
- **$display** 在**输出结束后会自动换行**，而$write 不会。

格式说明符	输出格式
%h 或 %H	以十六进制的格式输出
%d 或 %D	以十进制的格式输出
%o 或 %O	以八进制的格式输出
%b 或 %B	以二进制的格式输出
%c 或 %C	以ASCII字符形式输出
%s 或 %S	以字符串方式输出
%v 或 %V	输出连线型数据的驱动强度
%t 或 %T	输出模拟系统所使用的时间单位
%m 或 %M	输出所在模块的分级名（Hierarchical Name）
%e 或 %E	将实型量以指数方式显示
%f 或 %F	将实型量以浮点方式显示
%g 或 %G	将实型量以上面两种中较短的方式显示

输出格式说明符

✧ **系统任务$monitor**

- 属于**输出控制**类**系统任务**
- **调用形式：**

$monitor（“格式控制字符串”，输出变量名表项）

$monitoron；

$monitoroff；

- **$display 是每调用一次就执行一次**，而**$monitor 一旦被调用后，就相当于启动了一个后台进程**，将随时对输出变量名表项中列出的各个变量进行检测，如发现其中的任何一个变量在模拟过程中的某一时刻发生了任何形式的改变，则系统将按照调用$monitor 时所规定的格式，输出一次变量的结果。

✧ **系统函数$time 与$realtime**

- 属**模拟时标**类**系统函数**
- **调用形式：**

$time

$realtime

- 都返回从模拟程序开始执行到被调用时刻的时间，**$time 返回的是 64 位的整数，$realtime 返回的是一个实型数。**

✧ **系统任务$finish 与$stop**

- 用于**控制模拟进程**的**系统任务**

- **调用形式：**

  $finish;　　$stop;

  $finish(n);　　$stop(n);
- **$finish 的作用**就是**终止仿真器的运行，结束仿真过程。$stop 的作用**只相当于一个 **pause 暂停语句。**
- **n 只能取以下三个值：**

  0：**不输出**任何信息。

  1：**输出结束仿真的时间及模拟文件的位置**

  2：在 1 的基础上**增加对 CPU 时间、机器内存占有情况等统计结果的输出**。
- 当$finish **不带参数时，对应于缺省值 1**.

编译向导

✧ **文件包含 `include**

- 把语句中**指定的源文件全部包含到当前的文件中**
- 形式定义：**`include** “文件名”
- 几点说明：

  1、 每条`include 语句**只能用于对一个文件的包含**，多个文件的包含只需要多条语句分别注明即可。

  2、 文件包含**允许嵌套**。

  3、 **必要时**，`include 文件包含语句中的**文件名表项中应指明包含文件的目录路径**。

✧ **取消编译向导`resetall**

- 编译系统**碰到这条编译向导**语句后，将**取消前面所有由编译向导引入的定义**，并**恢复其原始的缺省态**。
- 这条语句通常**被放在一个 Verilog HDL 源文件的最开始**，取消可能的由于与别的文件一起编译而带来的一些不必要的编译控制。

6. 非 verilog 表达式

- 非法的自增算符“i++”，自减算符“--”;
- Verilog HDL 以**分号作为语句的结束符**，设计时会不经意中**用逗号代替，从而产生错误。**
- 在 verilog HDL 模块描述内部，如果**一个量被过程赋值语句所赋值**，那么**在模块的数据类型说明部分**，必须能够**找到它的寄存器类的类型说明语句。**
- 如果一个量被 **assign 连续赋值语句所赋值**，则**这个量必须是连线型**的。
- 在 verilog HDL 的结构描述中，对 **wire 型的连线可以缺省其连线类型说明**，但**线宽说明不能被缺省**。
- 如果**连线的位宽不是一位**，而又不能以其它方式说明线宽特征，则**必须用显式说明语句指明连线的宽度**。