浙江大学
ZHEJIANG UNIVERSITY

数字逻辑设计

Verilog HDL设计概要

浙江大学计算机学院　王总辉
zhwang@zju.edu.cn
http://course.zju.edu.cn
2021年10月

提 纲

- EDA与硬件描述语言
- Verilog设计入门
- Verilog HDL基础知识
- Verilog行为描述

提 纲

- EDA与硬件描述语言
- Verilog设计入门
- Verilog HDL基础知识
- Verilog行为描述

EDA与硬件描述语言

- EDA设计流程
- 硬件描述语言

EDA设计流程（1）

- 第一步：行为级描述
  - 在完成系统性能分析与功能划分的基础上，对于各个电路功能模块，用HDL语言（Verilog HDL/VHDL）完成行为级（Behavior Level）描述。
- 第二步：行为级优化与RTL级描述的转化
  - 行为级算法优化与功能仿真
  - 完成向RTL级描述的转化
- 第三步：选定工艺库，确定约束条件，完成逻辑综合与逻辑优化
  - 将RTL级HDL代码映射到具体的工艺上加以实现
  - 设计过程与实现工艺相关联

EDA设计流程（2）

- 第四步：门级仿真
  - EDA设计过程的每一个阶段都需要进行模拟仿真
  - 门级仿真包含了门单元的延时信息，需要相应工艺的仿真库的支持
- 第五步：测试生成
  - 功能测试，检测线路的逻辑、时序等是否正确
  - 制造测试，实现高的故障覆盖率，通常称之为测试向量，可自动生成
- 第六步：布局布线（P&R：Place & Routing）
  - 借助于版图综合的自动布局布线工具，在对应工艺的版图库支持下完成的，通常称之为后端设计

EDA设计流程（3）

- 第七步：参数提取
- 第八步：后仿真
  - 进行时序模拟，考察在增加连线延时后，时序是否仍然满足设计要求
- 第九步：制版、流片

硬件描述语言

- Verilog HDL（Hardware Description Language）是一种硬件描述语言，用于从算法级、门级到开关级的多种抽象设计层次的数字系统建模。
  - 最初是1983年由Gateway Design Automation公司(后被Cadence收购)为其模拟器产品开发的硬件建模语言
  - 1990年，Cadence公司成立OVI （Open Verilog International）组织来负责推广Verilog
  - 1995年，IEEE制定了Verilog HDL标准，即IEEE Std 1364 - 1995
  - 2001年，发布了IEEE Verilog 1364-2001
  - 2005年，发布了IEEE Verilog 1364-2005

Verilog与VHDL

- 目前，设计者使用Verilog和VHDL的情况
  - 美国：Verilog：60%，VHDL：40%
  - 台湾：Verilog：50%，VHDL：50%
- 两者的区别：
  - VHDL侧重于系统级描述，从而更多的为系统级设计人员采用
  - Verilog侧重于电路级描述，从而更多的为电路级设计人员采用

Verilog与C语言

- 虽然Verilog的某些语法与C语言接近，但存在本质上区别
  - Verilog是一种硬件语言，最终是为了产生实际的硬件电路或对硬件电路进行仿真
  - C语言是一种软件语言，可以用来控制硬件来实现某些功能
  - **利用Verilog编程时，要时刻记得Verilog是硬件语言，要时刻将Verilog与硬件电路对应起来**

提 纲

- EDA与硬件描述语言
- Verilog设计入门
- Verilog HDL基础知识
- Verilog行为描述

Verilog设计入门

- 模块概念
- 两种描述方式
  - 行为描述
  - 结构化描述
- 测试与仿真
- 数据选择器例子

模块的基本结构（1）

- 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
- 端口类型说明
  - 端口类型只有input、output、inout三种

模块的基本结构（2）

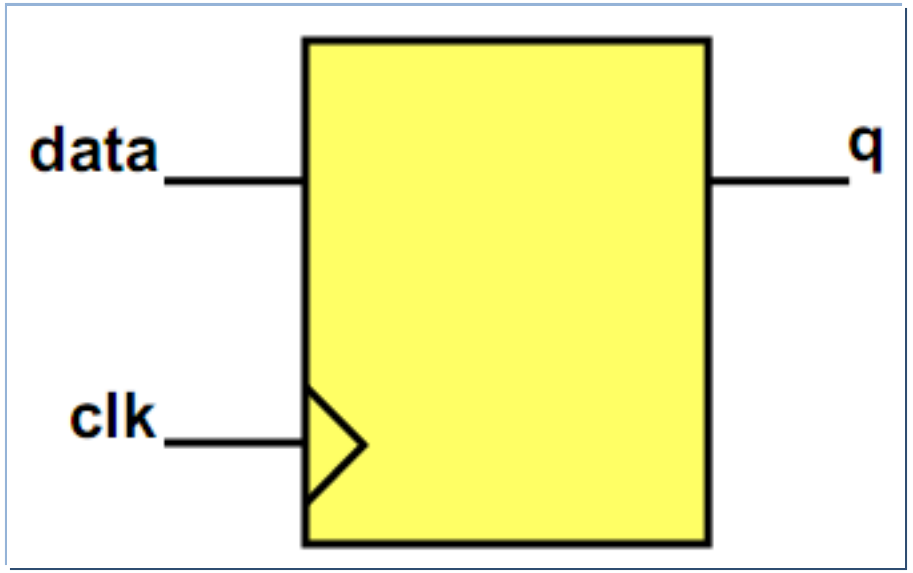
- 数据类型说明
  - 数据类型有连线类和寄存器类两个大类
  - 一位宽的wire类可被缺省外，其它凡将在后面的描述中出现的变量都应给出相应的数据类型说明
- 描述体部
  - 具体展开对模块的描述
- 结束行
  - 用关键词endmodule标志模块定义的结束

模块例子-D触发器

用Verilog HDL语言描述一个上升沿D触发器，其中clk为触发器的时钟，data，q分别为触发器的输入、输出

```
module dff_pos(data, clk, q);
    input data, clk;
    output q;
    reg q;

    always @(posedge clk)
        q = data;
endmodule
```



Verilog 语言描述

- 行为描述：描述行为或功能特性
  - 用法与C语言类似
  - 有条件语句，循环语句等。
  - 有算术操作，逻辑操作，位操作等表达式。
  - 时序控制
- 结构描述：描述通过什么样的结构方式将不同的实体连接起来
  - 用来描述门级或开关级电路
  - 一整套组合逻辑元器件
  - 支持门延迟

测试与仿真

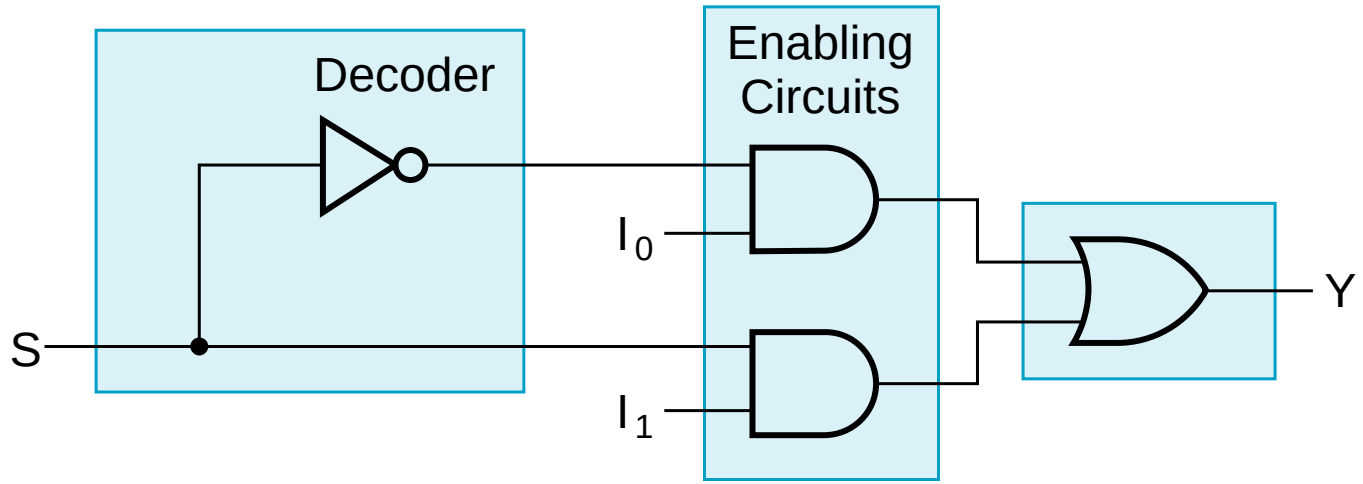
- 测试平台（Test Bench）
  - 在输入端口加入测试信号，从输出端口检测其输出结果是否正确
- 通常将需要测试的对象称之为DUT（Device Under Test）
- 测试模块
  - 要调用DUT
  - 包含用于测试的激励信号源
  - 能够实施对输出信号的检测，并报告检测的结果

Verilog仿真器

- Verilog-XL
- VCS
- NC-Verilog
- Active HDL
- ModelSim
- ……

数据选择器

- 二选一MUX功能模块
  - 行为描述模块
  - 结构描述模块
  - 测试模块



数据选择器-行为描述

```
module mux_beh(
    output wire out,
    input wire a,
    input wire b,
    input wire sel
    );

    assign out = ( sel == 0 ) ? a : b;

endmodule
```

数据选择器-结构描述

```
module mux_str(
    output wire out,
    input wire a,
    input wire b,
    input wire sel
    );

    not gate1( net1, sel );
    and gate2( net2, a, net1 );
    and gate3( net3, b, sel );
    or gate4( out, net2, net3 );
endmodule
```

数据选择器-测试模块

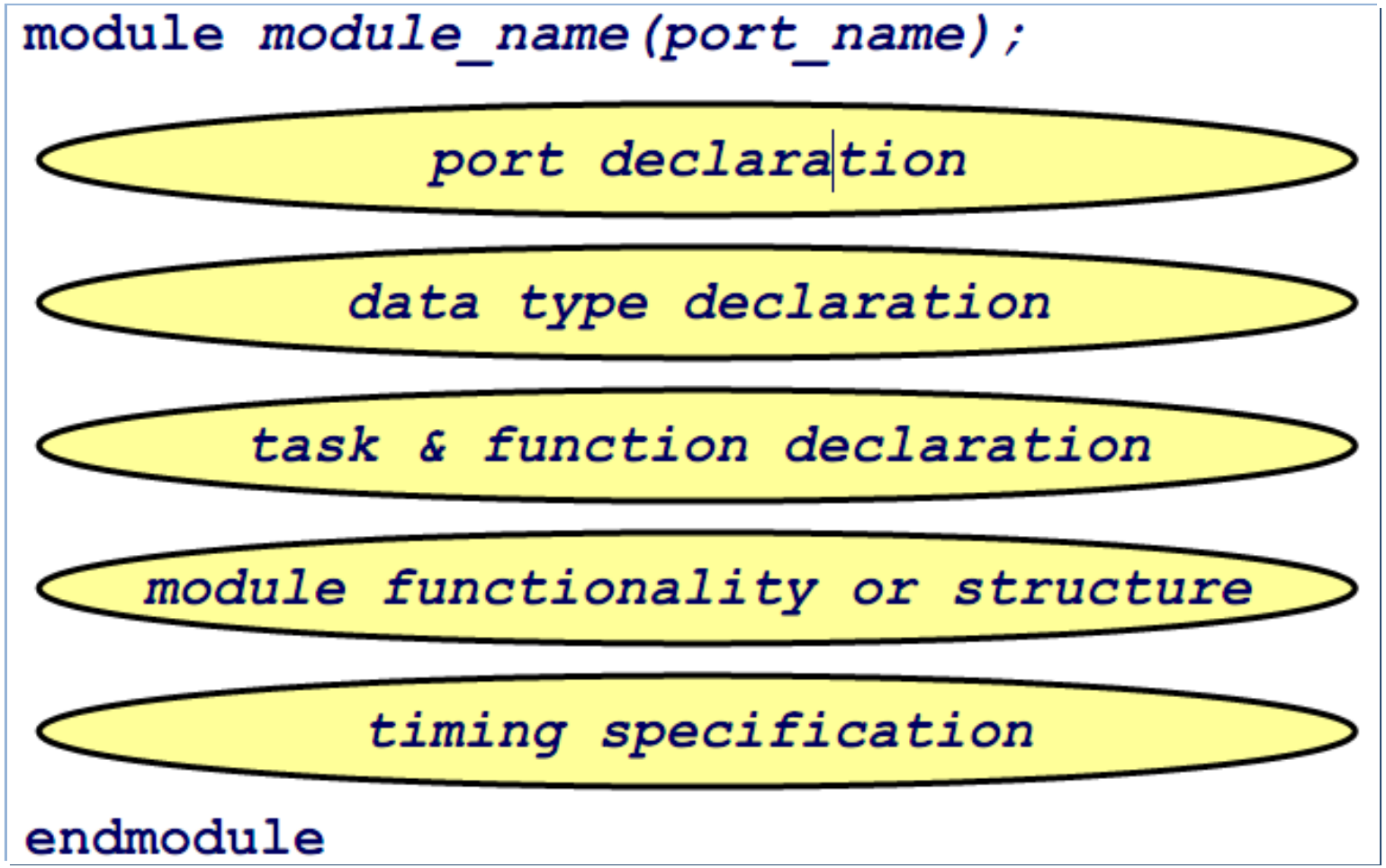
```
module  test_for_mux;
    reg a, b, s;
    // 调用DUT
    mux_str mux1( out, a, b, s );
    // 产生测试激励信号
    initial begin
        a = 0; b = 1; s = 0;
        #10 a = 1;
        #10 b = 0;
        #10 s = 1;
        #10 b = 1;
        #10 a = 0;
        #10 $finish;
    end
    // 检测输出信号
    initial
        $monitor( $time, " a=%b b=%b s=%b out=%b", a, b, s, out );
endmodule
```

```
 0 a=0 b=1 s=0 out=0
10 a=1 b=1 s=0 out=1
20 a=1 b=0 s=0 out=1
30 a=1 b=0 s=1 out=0
40 a=1 b=1 s=1 out=1
50 a=0 b=1 s=1 out=1
```

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

Verilog模块的典型结构



测试模块的典型结构

```
module testfixture;
    // Data type declaration

    // Instantiate modules

    // Applying stimulus

    // Display results

endmodule
```

提 纲

- EDA与硬件描述语言
- Verilog设计入门
- Verilog HDL基础知识
- Verilog行为描述

Verilog HDL基础知识

- 基本词法定义
- 数据类型
- 参数定义、宏替换及模拟时间单位的定标
- 运算符

空白符

- 空白符是以下几种字符与控制符的总称
  - 空格
  - TAB
  - 换行符
  - 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

Verilog HDL的四种逻辑状态

- 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
1	逻辑1、逻辑真、高电平
x 或 X	不确定的逻辑状态
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
  - 直接由0～9的数字串组成的十进制数
  - +/-〈位宽〉’〈基数符号〉〈按基数表示的数值〉

基数符号

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

整数表示实例

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)
4ac	缺省	十进制	非法
'h4ac	缺省	十六进制	0…0_0100_1010_1100(32位或以上)
9'o671	9 位	八进制	110_111_001
9'o-671	9 位	八进制	非法
6'hf3	6 位	十六进制	11_0011,高位部分被舍去
6'hf	6 位	十六进制	00_1111,高位部分由0补足
3'b10x	3 位	二进制	10x
12'h2x6	12 位	十六进制	0010_xxxx_0110
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

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

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
%%	%

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、$
- 还可以是以反斜杠“\”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

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

寄存器类型

- 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

连线类型及其功能

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

说明

- 连线主要出现在模块的结构描述中，对应硬件电路中的物理信号连接。
- 在对连线进行描述时，必须用连线类型定义语句进行类型说明，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。

寄存器类型及其说明

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

说明

- 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。

标量与向量

- 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（Scalar quantity）。
- 线宽大于一条的连线，或位数大于一位的寄存器称之为向量（vector）。
- 向量的范围由括在方括号中的一对数字表示，中间用一个冒号相隔。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

参数定义语句的形式描述

- parameter <参数定义表项>;

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。

```
module module_name(……);
…
parameter msb=7, lsb=0, delay=1;
reg [msb:lsb] reg_a;
and #delay (x, y, z);
…
endmodule
```

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。
- 一条宏替换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。

例子

```
`define msb 8 //用宏名msb来替代常数8
`define lsb 0 //用宏名lsb来替代常数0
`define delay_and and #1 /*用宏
                         delay_and来替代单元
                         名and及其延时参数*/
reg [`msb : `lsb] a;
`delay_and (x, y, z);
```

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二部分组成
  - 时间单位：
    - s（秒） ms(毫秒10^{-3}s) us(微秒10^{-6}s)
    - ns(纳秒10^{-9}s) ps（皮秒10^{-12}s） fs（飞秒10^{-15}s）

说明

- 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

运算符的分类

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	? :

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（？：）

算术运算符

- 加法运算符：　＋，实现加法运算
- 减法运算符：　－，实现减法运算
- 乘法运算符：　＊，实现乘法运算
- 除法运算符：　／，实现除法运算
- 取模运算符：　％，实现取模运算

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

- 缩位运算符是单目运算符，按位进行逻辑运算
- 缩位运算符包括
  - 与（&）
  - 或（|）
  - 异或（^）
  - 及其相应的非操作（~&，~|，~^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）

关系运算符

- 小于：　　＜
- 大于：　　＞
- 小于等于：＜＝
- 大于等于：＞＝

相等与全等运算符

- 共有四种：
  - 相等运算符：　　＝＝
  - 不等运算符：　　！＝
  - 全等运算符：　　＝＝＝
  - 不全等运算符：　！＝＝
- 都是双目运算符，得到的结果是1位的逻辑值

相等算符

- 逐位比较二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。

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

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

移位运算符

- 逻辑移位
  - 逻辑左移：<<
  - 逻辑右移：>>
- 算术移位
  - 算术左移：<<<
  - 算术右移：>>>

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。

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〉

运算符的优先级顺序

	Op	Meaning
Highest	~	NOT
	*, /, %	MUL, DIV, MOD
	+, –	PLUS, MINUS
	<<, >>	Logical Left/Right Shift
	<<<, >>>	Arithmetic Left/Right Shift
	<, <=, >, >=	Relative Comparison
	==, !=	Equality Comparison
	&, ~&	AND, NAND
	^, ~^	XOR, XNOR
	|, ~|	OR, NOR
Lowest	?:	Conditional

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	? :

不允许实型量参与的运算符

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	<< >>

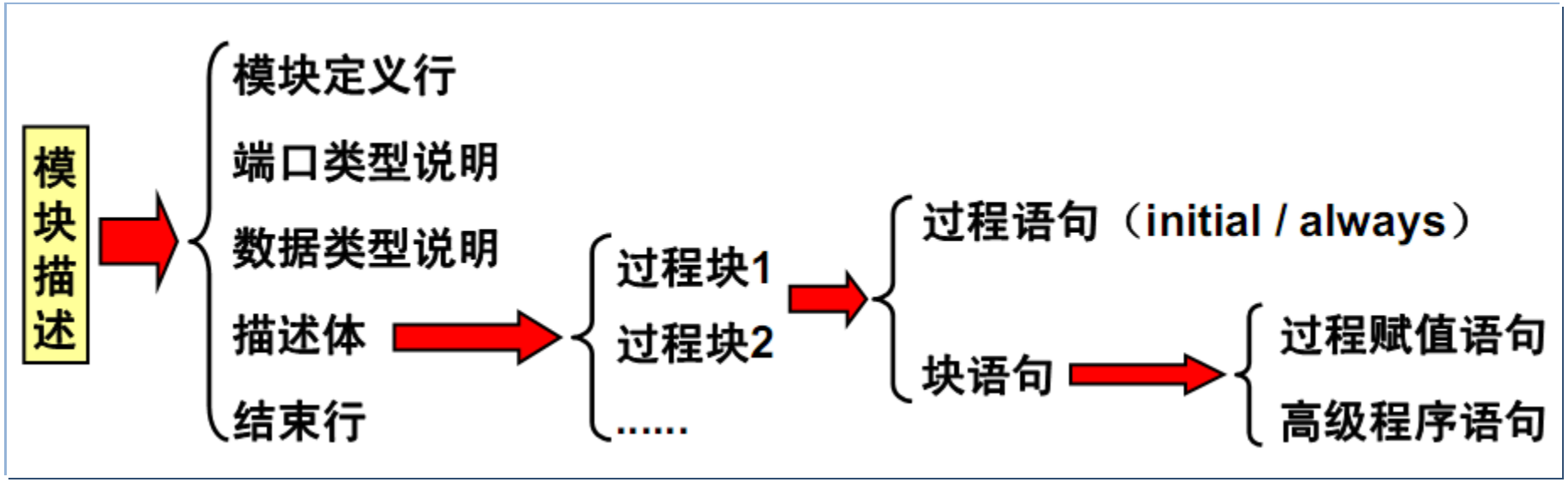
提 纲

- EDA与硬件描述语言
- Verilog设计入门
- Verilog HDL基础知识
- Verilog行为描述

Verilog行为描述

- Verilog行为描述的构成框架
- 赋值语句
- 高级程序语句
- 模块调用

模块的基本结构


模块描述
模块定义行
端口类型说明
数据类型说明
描述体
结束行
过程块1
过程块2
.....
过程语句（initial / always）
块语句
过程赋值语句
高级程序语句

过程块

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

initial过程语句

- 从模拟的0时刻开始执行，后面的块语句沿时间轴只执行一次，不带触发条件。
- 常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。

always 过程语句

- always过程语句通常带有触发（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。
- 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。
- 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。

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
  - 每条语句依据在块中的排列次序，先后**逐条顺序执行**。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。
  - 起始执行时间就是串行块中第一条语句开始执行的时间。结束时间就是块中最后一条语句执行结束的时间。
  - 可以理解为**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
  - 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顺序无关。语句给出的延时都是相对于开始执行时的绝对时间。
  - 起始执行时间就是流程控制转入的时间，每条语句都是相对于这一时间同时开始执行的。结束时间就是按执行时间排序最后执行的一条语句结束的时间。
  - 可以理解为**硬件电路上电后，各电路模块同时开始工作的过程**。

串行块-例子

- 用串行块行为描述产生一段周期为100时间单位、占空比为1:1的信号波形。

```
module wave_gen_seri(output wire wav);
    reg wav;
    event end_wave;
    parameter delay=50;
    initial
    begin
        wav=0;
        #delay wav=1;
        #delay wav=0;
        #delay wav=1;
        #delay wav=0;
        #delay ->end_wave;
    end
endmodule
```

并行块-例子

- 用并行块行为描述产生一段周期为100时间单位、占空比为1:1的信号波形。

```
module wave_gen_para(output wire wav);
    reg wav;
    event end_wave;
    initial
    fork
        wav=0;
        #50 wav=1;
        #100 wav=0;
        #150 wav=1;
        #200 wav=0;
        #250 ->end_wave;
    join
endmodule
```

赋值语句

- 连续赋值语句
  - 连续赋值语句用于对连线类变量
- 过程赋值语句
  - 过程赋值语句完成对寄存器类变量的赋值

过程赋值语句

- 位于过程块中的赋值语句称为过程赋值语句。
- 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-例子

```
reg_a=8’b1011_1100;     //对一个8位寄存器的赋值

reg_a[3]=1’b0;          //对寄存器的某一位赋值

reg_a[7:4]=4’b1010;     //对寄存器的其中几位赋值

mem_a[address]=8’h5d;  //对由address地址指定的存储器单元赋值

{carry, sum}=reg_a+reg_b; //通过连接算符构成一个整体进行赋值
```

两种定时控制模式

- 外部模式
  - <定时控制> <寄存器变量>=<表达式>;
- 内部模式
  - <寄存器变量>=<定时控制> <表达式>;

外部模式

- 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。
  - **延时控制**：就是直接给出所需延时的时间，如#delay　a=b;
  - **事件控制**：以符号“@”开头，后面紧跟的是事件控制敏感表
    - @（信号名）
    - @（posedge 信号名）/ @（negedge 信号名）
    - @（敏感事件1 or 敏感事件2 or 敏感事件3 ……）

内部模式

- 先完成对表达式的求值过程，再等待延时的到期，再完成赋值过程。

阻塞型过程赋值

- 阻塞型过程赋值算符：＝
- 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，后面的语句不能被执行。

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，等同于并行块

连续赋值语句 VS 过程赋值语句(1)

- 对象不同
  - 连续赋值语句用于对连线类变量
  - 过程赋值语句完成对寄存器类变量的赋值
- 赋值过程实现方式不同
  - 连续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

连续赋值语句 VS 过程赋值语句(2)

- 语句出现的位置不同
  - 连续赋值语句不能出现在过程块中
  - 过程赋值语句只能出现在过程块中
- 语句结构不同
  - 连续赋值语句以关键词assign为先导
  - 过程赋值语句分阻塞型和非阻塞型两类

高级程序语句

- 直接借用高级语言中的程序设计语句对模块进行描述。
- 出现在对模块进行行为描述的过程块中
- 分三类
  - if-else条件语句
  - case语句
  - 循环语句
    - forever
    - repeat/while/for

模块调用

- 顶层模块
  - 必定有一个，也只有一个
  - 名称可以自己定义
- 模块调用
  - 基本门调用（ Primitive Instantiation ）
  - 模块调用（ Module Instantiation ）
- 语句
  - 模块名 调用名（端口名表项）；

端口关联方式

- 位置关联
  - 调用时的信号名按照模块定义时确定的端口顺序排列，中间用逗号分隔
  - 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，逗号不能省略
- 名称关联
  - 模块定义时的端口名和调用时的信号名之间一一对应：.定义时的端口名(调用时与之相连的信号名)
  - 调用时端口名的排列顺序可以随意改变。

端口关联-例子

- 位置对应
  - full-adder m1(a,b,in,s,out);
  - full-adder m2(a,b,,s,cout);
- 名称关联
  - full-adder
    m3(.x(a),.cin(in),.y(b), .s(c),.cout(out));

Thank You !