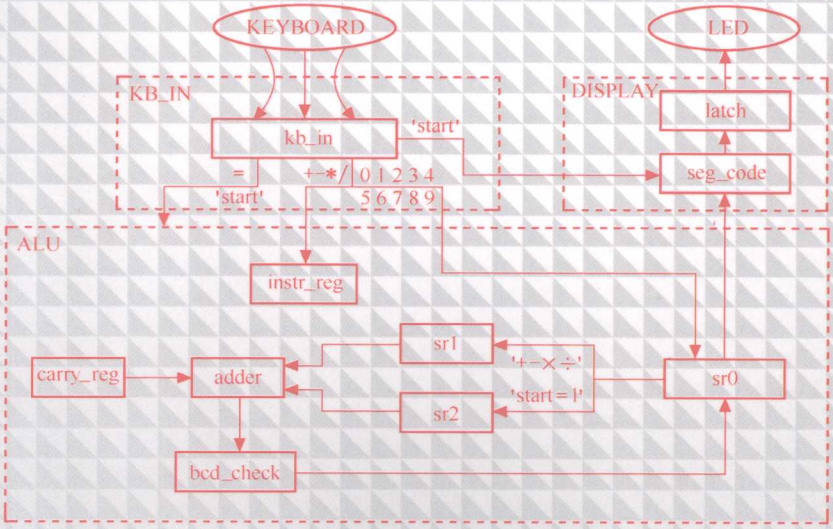
Verilog HDL 与数字系统设计简明教程

吴戈 编著



人民邮电出版社
POSTS & TELECOM PRESS

Verilog HDL与数字系统设计简明教程

吴戈 编著

人民邮电出版社
北京

图书在版编目（CIP）数据

Verilog HDL与数字系统设计简明教程 / 吴戈编著.
北京：人民邮电出版社，2009.2
ISBN 978-7-115-19366-7

Ⅰ. V… Ⅱ. 吴… Ⅲ. ①硬件描述语言，Verilog HDL－程序设计－教材②数字系统－系统设计－教材 Ⅳ. TP312
TP271

中国版本图书馆CIP数据核字（2008）第196314号

内 容 提 要

本书分 3 部分，第 1 部分（第 1～6 章）是语法部分，详细讲解 Verilog HDL 语法知识和基本应用；第 2 部分（第 7～9 章）是实例部分，通过从已公布的成熟源代码中精选的 50 多个最具代表性的建模实例，辅以框图和详细注释帮助读者理解程序，从而学习典型电路单元的建模方法；第 3 部分（第 10 章）是系统设计实战，为初学者展示了一个小型系统的详细设计流程。

本书内容丰富、实用性强，适合高等院校相关专业高年级学生和研究生学习和阅读，对参加相关工作的工程技术人员也有很强的参考价值。

Verilog HDL 与数字系统设计简明教程

◆ 编　　著　吴　戈
责任编辑　刘　浩

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街 14 号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京顺义振华印刷厂印刷

◆ 开本：787×1092　1/16
印张：18.75
字数：469 千字　　2009 年 2 月第 1 版
印数：1－3 000 册　　2009 年 2 月北京第 1 次印刷

ISBN 978-7-115-19366-7/TP

定价：35.00 元

读者服务热线：(010)67132692　印装质量热线：(010)67129223
反盗版热线：(010)67171154

前　言

HDL（Hardware Description Language，硬件描述语言）是伴随着集成电路设计复杂度和集成度的急剧上升而出现的。

硬件工程师们总要画电路图，用一个个小元件搭出或大或小的系统原理图。当电路图中的元件多达百个以上时，无论是画图还是分析都会有一定的难度。同样的情形也发生在集成电路设计中，当一个芯片内必须包含上万个甚至几十万、几百万个晶体管时（目前面市的频率最高的 CPU 芯片内已经集成了近 2 亿个晶体管），用原理图的方式来设计和管理显然是“不可能的任务”。于是出现了 HDL 设计方法，就像 20 世纪 70 年代高级编程语言迅速取代汇编语言一样，从 20 世纪 90 年代以来，HDL 逐渐取代了门级原理图设计方法。

Verilog HDL 是 1983 年由位于英格兰阿克顿市的 GDA 公司开发出来的。Verilog HDL 的具体物理建模能力强，语法类似 C 语言，容易学习。Verilog 在工业界被应用广泛，美国大多数公司的 RTL 级代码都是用 Verilog 写的。

本书主要针对初学者，在内容安排上主要以语法讲解和程序分析为主，并没有介绍复杂系统的设计方法。初学者应当从本书的基本结构和方法出发，打好基础，一旦成为一个熟练的设计师，就会发现 Verilog 只是一个工具，而对系统功能的考虑和时序的精细设计才是最需要设计人员考虑的。

本书的写作目标是让初学者顺利入门，希望您在阅读本书的过程中逐渐了解到 Verilog HDL 是什么、如何写、如何用，面对一段写好的程序能够做出正确分析，最终掌握设计流程和建模方法。

Verilog HDL 发展至今 20 多年积累的文档、书籍和各种资料可谓不计其数，本书作者试图从这些资料中，找到更好的也是更适合初学者学习的内容，用一种更好的组织方式，呈现给读者，让读者从中受益。

大量使用实例并带有详细的注释和分析是本书的最大特点。书中提供大量程序实例，目的不仅仅是让读者在学习阶段理解它们的建模方法和技巧，而且可以在设计阶段把其中很多程序片段直接拿来使用，从而节省时间。

实践是学好编程语言的最佳途径，所以建议读者在学习之余尽量能多动手编程并做仿真。如果您能把本书给出的实例全都调试一次，一定可以事半功倍。

本书由吴戈编写，参加资料整理的还有何伟、张兵、刘兆宏、季建华、刘福刚、赵文革、黄弦、邓玉春、曾庆华、石昀、朱元斌、钱文杰、陈功杰、汪洪、刘超、钟晓媛等，在此一并表示感谢！

本书的课件可在 http://www.ptpress.com.cn/Resources.aspx 处下载！

由于时间有限，书中难免存在不足，恳请广大读者朋友批评指正（book_better@sina.com）。

编者

2009 年 1 月

目　录

第 1 章　初识 Verilog HDL

硬件描述语言的出现彻底改变了数字电路的设计方法，使得工程师们可以像写 C 程序那样设计电路，从而把更多的精力集中到系统结构和算法实现上。Verilog HDL 是一门优秀的硬件描述语言，直观易学，在工业界获得广泛应用。本章将介绍 Verilog HDL 的基本概念和结构。

1.1　什么是 Verilog HDL

Verilog HDL 是一种硬件描述语言，可以在算法级、门级到开关级的多种抽象设计层次上对数字系统建模。

Verilog HDL 可以描述设计的行为特性、数据流特性、结构组成以及包含响应监控和设计验证方面的时延和波形产生机制。此外，Verilog HDL 提供了编程语言接口，通过该接口用户可以在模拟、验证期间从外部访问设计，包括模拟的具体控制和运行。

Verilog HDL 不仅定义了语法，而且对每个语法结构都定义了清晰的模拟、仿真语义。因此，用这种语言编写的模型能够使用 Verilog HDL 仿真器进行验证。Verilog HDL 从 C 语言中继承了多种操作符和结构，所以从形式上看 Verilog HDL 和 C 语言有很多相似之处。虽然 Verilog HDL 有一些不太容易理解的扩展功能，但是 Verilog HDL 的核心子集非常易于学习和使用，而且对大多数建模应用来说核心子集已经足够用了。完整的 Verilog HDL 足以对最复杂的芯片和完整的电子系统进行描述。

1.2　主要功能

作为一种硬件描述语言，Verilog HDL 可以直接描述硬件结构，也可以通过描述系统行为实现建模。Verilog HDL 的主要特点和功能有：

- 描述基本逻辑门，如 and、or 和 nand 等基本逻辑门都内置在语言中，可以直接调用。
- 描述基本开关模型，如 nmos 、pmos 和 cmos 等基本开关都可以直接调用。
- 允许用户定义基元（UDP），这种方式灵活而有效，用户定义的基元既可以是组合逻辑也可以是时序逻辑。
- 可以指定设计中的端口到端口的时延、路径时延和设计的时序检查。
- 可采用多种方式进行建模。这些方式包括顺序行为描述方式——使用过程化结构建模，数据流行为方式——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。
- Verilog HDL 中有两类数据类型，线网数据类型和寄存器数据类型。线网类型表示构

件间的物理连线，而寄存器类型表示抽象的数据存储元件。

- 能够描述层次设计，可使用模块实例结构描述任何层次。
- 设计的规模可以是任意的，语言不对设计的规模（大小）施加任何限制。
- Verilog HDL 是 IEEE 标准而不再是某些公司的专有语言，语言标准都是公开的。
- 人和机器都可阅读 Verilog HDL 语言，因此它可作为 EDA 的工具和设计者之间的交互语言。
- Verilog HDL 语言的描述能力可以通过使用编程语言接口（PLI）进一步扩展。PLI 是允许外部函数访问 Verilog HDL 模块内信息，允许设计者与模拟器交互的例程集合。
- 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。
- 能够使用内置开关级原语在开关级对设计完整建模。
- 同一语言可用于生成模拟激励和指定测试的验证约束条件，例如指定输入值。
- Verilog HDL 能够监控模拟验证的执行，即模拟验证执行过程中设计的值能够被监控和显示。这些值也能够用于与期望值进行比较，在不匹配的情况下，打印报告消息。
- Verilog HDL 不仅能够在 RTL 上进行设计描述，而且能够在体系结构级和算法级行为上进行设计描述。
- 能够使用门和模块实例化语句在结构级进行结构描述。
- Verilog HDL 具有混合方式建模能力，即设计中每个模块均可以在不同设计层次上建模。
- Verilog HDL 具有内置逻辑函数，例如&（按位与）和 |（按位或）。
- Verilog HDL 内有很多高级编程语言结构，例如条件语句、情况语句和循环语句。
- Verilog HDL 可以对并发行为和定时行为进行建模。
- Verilog HDL 提供强有力的文件读写能力。
- Verilog HDL 在某些特定情况下是非确定性的，即相同的程序在不同的模拟器（仿真工具）上可能产生不同的结果。

1.3 设计流程

图 1-1 所示是一个典型的 FPGA/CPLD 设计流程，如果是 ASIC 设计，则不需要 STEP5 这个环节，而是把综合后的结果交给后端设计组（后端设计主要包括版图、布线等）或直接交给集成电路生产厂家。

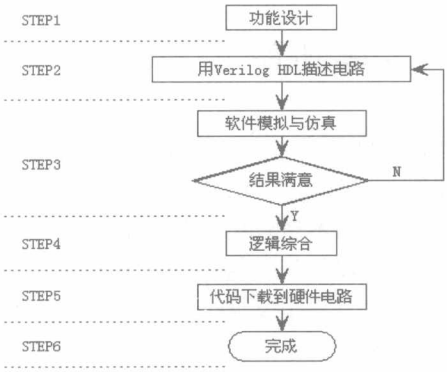


图 1-1 Verilog HDL 设计流程

1.4 基本结构

Verilog HDL 程序文件的后缀都是“.v”，假如为加法器建模时创建了一个名为 adder 的文件，那么这个文件就是 adder.v。每个.v 文件里可以有一个或几个模块的描述程序。

1.4.1　模块的概念

模块（moudle）是 Verilog HDL 最基本的概念，也是 Verilog HDL 设计中的基本单元。每个 Verilog HDL 设计的系统都是由若干模块组成的。所以有必要在学习语法之前了解模块的概念。

- 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。
- 模块的实际意义是代表硬件电路上的逻辑实体（即实现特定逻辑功能的一组电路），其范围可以从简单的门到整个大的系统，比如计数器、存储子系统、微处理器等。
- 每个模块都实现特定的功能（一般来说最底层模块的功能都比较简单）。
- 模块的描述方式有行为建模和结构建模（或二者结合）之分。
- 模块之间是并行运行的。
- 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功能。
- 各模块连接完成整个系统需要用一个顶层模块（Top-module）。

可以这样理解模块与系统：用 Verilog HDL 做设计有点像搭积木，每个模块都是一个积木块。不同的是，我们首先要“制造”出很多积木块。每块积木用在不同的地方有不同的功能要求（不同的模块要实现不同的功能），制作每个积木块的过程就是给每个模块编程，最后把这些积木（模块）搭在一起完成整个系统的设计。

无论多复杂的系统，总能划分成多个小的功能模块。因此系统的设计可以按下面 3 个步骤进行：

（1）把系统划分成模块。

（2）规划各模块的接口。

（3）对模块编程并连接各模块完成系统设计。

每个步骤都涉及到模块，可见模块是整个设计中最基本且非常重要的单元。

模块的结构是这样的：

```
module<模块名>(<端口列表>);
    <定义>
    <模块条目>
endmodule
```

其中：

- <模块名>是模块惟一的标识符（模块的名称）。
- <端口列表>是输入、输出和双向端口的列表，这些端口用来与其他模块进行连接（不妨理解为集成电路的引脚，是用来和其他电路连接的）。
- <定义>是一段程序，用来指定数据对象为寄存器型、存储器型、线型以及过程块，诸如函数块和任务块（说明这个模块内部都有什么）。
- <模块条目>是说明这个模块要做什么的语句。
- 标识模块结束的 endmodule 之后没有分号。

[例 1-1]这是一个 NAND 与非门模块的建模程序，输出 out 是输入 in1 和 in2 相与后求反的结果。图 1-2 所示为本例示意图。

```
module NAND(in1, in2, out);        //NAND 模块
    input in1, in2;                //两个输入端口 in1 和 in2
```

```
    output out;                    //一个输出端口 out
    assign out = ~(in1 & in2);     //连续赋值语句
endmodule                          //NAND 模块结束
```

图 1-2　NAND 模块

在上面这段程序中，module 之后的 NAND 是模块的名称，如果高层模块想调用本模块（实例化），就需要使用这个名称；端口列表里的 in1、in2 和 out 是模块和外界通信的途经；“input in1, in2; ”和“output out;”这两句就是<定义>，规定 in1 和 in2 是输入端口，out 是输出端口；“assign out = ~(in1 & in2);”这句就是<模块条目>，assign 连续赋值语句不间断地监视等式左右端的变量，一旦其中任意一个发生变化，右端表达式被重新赋值并将结果传给等式左端进行输出；“//连续赋值语句”是注释，单行注释以符号“//”开头；endmodule 表明这个模块描述完毕。

1.4.2　模块调用

在做模块划分时经常会出现这种情形：某个大的模块中包含了一个或多个功能子模块。Verilog HDL 是通过“模块调用”或称为“模块实例化”的方式实现这些子模块与高层模块的连接的。下面通过一个与门的建模实例来说明模块调用的方法。

[例 1-2] 为与门建模。通过将一个 NAND 门的输出连到另一个 NAND 门的两个输入上可得到一个与门，可以把这个与门看作“顶层”模块，例 1-1 中的 NAND 模块可以看作是本例的子模块。图 1-3 所示为本例示意图。

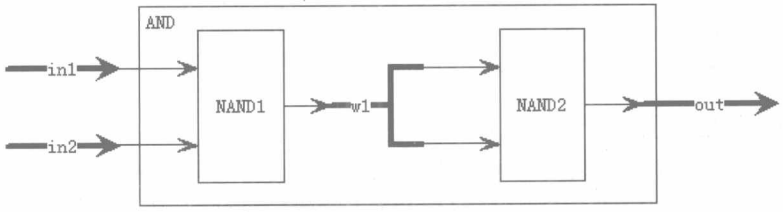


图 1-3　AND 模块

```
module AND(in1, in2, out);          // AND 模块
    input in1, in2;                 // 两个输入端口 in1 和 in2
    output out;                     // 一个输出端口 out
    wire w1;                        // 一个模块内部连线 wire
    NAND NAND1(in1, in2, w1);       //调用（实例化）一个 NAND 子模块
    NAND NAND2(w1, w1, out);        //再调用（实例化）一个 NAND 子模块
endmodule                           // AND 模块结束
```

这个 AND 模块含有两个 NAND 模块实例，分别是 NAND1 和 NAND2，通过内部连线 w1 连接起来，就可以实现与门的逻辑功能。

调用模块实例的一般形式为：

```
<模块名><参数列表><实例名>(<端口列表>);
```

其中，<模块名>是要调用子模块的名称，本例中要调用的是 NAND；<参数列表>是传输到子模块的参数值，参数传递的典型应用是定义门级时延，但是本例并未使用；<实例名>是把子模块调用过来后对它的命名，本例分别给两个子模块命名为 NAND1 和 NAND2；<端口列表>是实现子模块连接并实现高层模块功能的关键，如图 1-3 所示，AND 模块的两个输入 in1 和 in2 分别作为子模块 NAND1 的两个输入，AND 模块内部连线 w1 成为子模块 NAND1 的一个输出和 NAND2 的两个输入，AND 模块的输出端 out 作为子模块 NAND2 的输出。

1.4.3　测试模块

完成程序描述之后为了确认这个模型是否正确，应当对模块进行测试，即从模块输入端输入信号，再从输出端得到输出信号，对这些输入输出信号进行分析可以检查模型是否正确。Verilog HDL 提供了一种灵活有效的产生输入信号波形的方式——写测试程序（testbench），就是用一段程序产生激励信号，用语言描述信号的变化。testbench 应当是被测模块的高层模块，它没有 I/O 端口，而且内部有被测模块的实例。

下面通过为例 1-2 编写测试模块进行说明。

[例 1-3] 检查所设计的与门模块是否能实现与门的功能。把激励信号的变化送入 AND 模块的输入端，再把处理过的信号从 AND 模块的输出端取出，通过对这些输入输出信号进行分析（主要是分析波形）可以检查模块的功能是否满足设计要求。图 1-4 所示为本例示意图。

```
module test_AND;
    reg a, b;  //定义两个寄存器变量 a 和 b
    wire out1, out2;  //定义两个线网 out1 和 out2

    initial    //产生测试数据（激励信号）
      begin
        a=0; b=0;
        #1 a=1;
        #1 b=1;
        #1 a=0;
      end

    initial  //监测功能
      begin
        $monitor("Time=%0d a=%b b=%b out1=%b out2=%b", $time, a, b, out1, out2);
      end
    AND gate1(a, b, out2);      /*模块 AND 实例，激励信号通过 a 和 b 端进入 AND 模块，
                                  测试结果从 out2 输出*/
```

```
    NAND gate2(a, b, out1);    /*模块 NAND 实例，激励型号通过 a 和 b 端进入 NAND 模块，
                                 测试结果从 out1 输出*/
endmodule
```

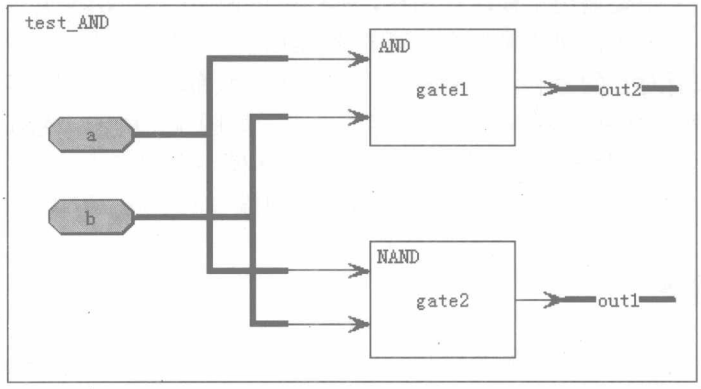


图 1-4　test_AND 模块

程序中有两个 initial 引导的程序段，每个 initial 语句后都有一个 begin-end 结构，第一个 begin-end 之间的语句是测试信号，描述了 a 和 b 的值的变化：先把 a 和 b 置为 0，然后“#1”延迟一个时间单位（这个时间单位可由系统默认或自己定义）之后，把 a 置为 1，再用“#1”延迟一个时间单位后，将 b 置为 1，再经过一个时延后将 a 置为 0。这样就完成了激励信号的产生，延迟时间和信号值改变次数可以根据自己仿真的需要用如此简单的语句完成。第二个 begin-end 之间的语句实现监测信号的功能，可以让程序随时向屏幕输出 a、b、out1、out2 的值。

上述程序仿真后将产生如下结果：

```
Time=0 a=0 b=0 out1=1 out2=0
Time=1 a=0 b=0 out1=1 out2=0
Time=2 a=0 b=1 out1=0 out2=1
Time=3 a=0 b=1 out1=1 out2=0
```

仿真器执行所有的事件后自行停止，因此不需要指定仿真结束时间。

从形式上看，模块的定义和调用（实例化）类似于软件中函数的定义和调用，其实二者有本质的区别。软件中的函数调用只是对同一段程序的使用，不管调用多少次，这个函数在程序中只有这么一段。但是在 HDL 硬件编程中，模块的调用是硬件的实现。每一次调用（实例化）都将产生实现这个模块功能的一组电路。如图 1-3 所示，调用了两次 NAND 模块，就在 AND 模块中产生了两块实实在在的 NAND 电路。

1.5　程序设计基础

Verilog HDL 其实是最容易学会的一种编程语言，因为它的结构和语法都比较简单而且和 C 语言相似。本节将介绍一些 Verilog HDL 程序设计的基础知识，主要是为了让读者初步了

解 Verilog HDL 程序的形式，如果对于某些语法无法理解的话请暂时放下，随后的章节会有详细讲解。

1.5.1　程序格式

Verilog HDL 是一种书写格式非常自由的语言，可以把一段程序写在一行之内，形式如下：

```
initial begin Top=3'b001; #2 Top=3'b011; end
```

也可以跨越多行编写，形式如下：

```
initial
  begin
    Top=3'b001;
    #2 Top=3'b011;
  end
```

对 Verilog HDL 仿真器而言，上面这两种形式没有任何区别，都是正确的，但是后一种形式可以让设计者看起来更“舒服”些，程序结构也一目了然。所以，为了提高程序的可读性，建议采用第二种格式编写程序。

请注意，在 Verilog HDL 程序中，语句间的空格是没有任何意义的。在 Verilog HDL 中，只有出现在字符串中的空格才是有意义的，如字符串“Verilog HDL”中的空格会被当作一个字符对待，但是所有其他的空格都是被 Verilog HDL 仿真器忽略的无意义符号，这些空格一般只是用来提高程序的可读性。可以说，之所以 Verilog HDL 的书写格式非常自由，就是因为空格是无意义的，用户可以任意排版。有时为了让程序排列得更整齐或更利于阅读，可以在程序中任意插入空格，如上例中的赋值语句“Top=3'b001;”可以写成“Top = 3'b001;”，只是在等号两侧增添了两个空格，这样可以使得这条语句看起来不那么“拥挤”。

另外，Verilog HDL 是区分大小写的，如果定义的变量是 sum，但是引用变量时写成 Sum 的话，程序会出错。

1.5.2　注释语句

有的程序员宁肯自己从头开始写 1000 行代码，也不愿意去阅读别人写的 100 行代码。这是千真万确的，阅读别人写的程序是一件痛苦的事情，如果注释很少或者干脆没有注释，简直就是一种折磨。写程序不加注释是一种很不好的编程习惯，有效的注释不但能够让别人尽快理解你的程序，也能在程序需要修改时帮助你迅速找到相关的点。

Verilog HDL 的注释语句有两种形式。

（1）单行注释。用符号“//”表示注释的开始，从这个符号开始到本行的结束都被认为是注释，而且它只能注释到本行结束。如：

```
reg a, b;   //定义两个寄存器变量 a 和 b
```

（2）多行注释。以起始符“/*”开始，到终止符“*/”结束，可以跨越多行，在一对起始符与终止符之间的所有内容都被认为是注释。如：

```
AND gate1(a, b, out2);      /*模块 AND 实例，激励信号通过 a 和 b 端进入 AND 模块，
                                测试结果从 out2 输出*/
```

1.5.3 标识符和关键词

标识符是用户定义的各种名称，可以是模块、端口、寄存器、线网、实例和程序块等元素的名称，比如语句“module adder;”就定义了一个标识符 adder，而语句“reg abc;”则定义了标识符 abc。标识符可以是字母、数字和下划线“_”等符号的任意组合序列，但首字符不能是数字，而且单个标识符的总字符数不能多于 1024。

关键词是语言的保留字，有其特定的和专有的语法作用，用户不能再对这些关键词做新的定义。Verilog HDL 共有 102 个关键词。注意：关键词必须是小写的，如“module”是关键词，而“Module”不是。

1.5.4 参数声明

程序中经常多次出现某些数字，如延迟时间或变量的宽度，有时（如调用任务或实例化模块时）可能要改变这些值，这种情况下经常要用到参数。参数一经声明，就视其为一个常量，在整个仿真过程中不再改变。其声明形式如下：

```
parameter param1=const_expr1, param2 = const_expr2, ..., paramN = const_exprN;
```

parameter是用于声明参数的关键词；param1、param2…paramN是标识符；const_expr1、const_expr2…const_exprN分别是标识符要代表的数字，它们可以不仅仅是数字，也可以是计算表达式。下面是3条参数声明语句：

```
parameter LINELENGTH=132, ALL_X_S=16'bx;
parameter BIT=1, BYTE=8, PI=3.14;
parameter STROBE_DELAY=(BYTE + BIT)/2;
```

使用参数可以提高程序的可读性，也利于修改，尤其是延迟时间和变量宽度这些在调试中可能经常修改的值。

1.5.5 预处理指令

和C语言类似，Verilog HDL也有预处理指令，预处理指令是以反引号“`”开始的某些标识符，它们指示编译器执行某些操作。预处理指令通常应当出现在Verilog HDL文件的最开始几行。

Verilog HDL共有8组预处理指令：

- `define, `undef
- `ifdef, `else, `endif
- `include
- `timescale
- `resetall
- `default_nettype
- `unconnected_drive, `nounconnected_drive
- `celldefine, `endcelldefine

其中最常用的是前4组，下面介绍这4组指令的含义及用法。

1．`define 和`undef

`define指令和C语言中的#define指令功能相同，是用一个指定的标识符来代表一个字符串，它的一般形式为：

```
`define 标识符 字符串
```

如：

```
`define MAX_BUS_SIZE 32
...
reg [ `MAX_BUS_SIZE - 1:0 ] AddReg;
```

在上面这段程序中，MAX_BUS_SIZE被`define指令指定代表字符串32，程序中所有的MAX_BUS_SIZE都被替换为32，所以寄存器AddReg的位宽就是[32-1:0]。

一旦`define 指令被编译，那么它在整个编译过程中都有效，除非遇到`undef 指令。`undef指令用于取消前面`define指令所做的定义。例如：

```
`define WORD 16                    //指定WORD的值为16
...
wire [ `WORD : 1] Bus;             //[16:1] Bus
...
`undef WORD                        //在`undef编译指令执行后, WORD不再代表16
```

2．`ifdef、`else 和`endif

这3个编译器指令通常一起出现，和普通的if-else结构类似，这些编译指令用于条件编译，如下例所示：

```
`ifdef WINDOWS
  parameter WORD_SIZE = 16
`else
  parameter WORD_SIZE = 32
`endif
```

上面这段程序可以这样理解：如果已经定义了WINDOWS，那么定义参数WORD_SIZE为16，否则定义WORD_SIZE为32。

`else程序指令对于`ifdef 指令是可选的，写成下面这样的结构也是可以的：

```
`ifdef WINDOWS
  parameter WORD_SIZE = 16
`endif
```

3．`include

`include指令用于嵌入“内嵌文件”的内容，这里的“内嵌文件”一般也是Verilog HDL文件。如下所示，假如在某个Verilog HDL文件adder.v中有如下内容：

```
`include " . . / . . /halfadder.v "    //双引号内是要内嵌的文件的路径和文件名
module adder;
  ......
```

```
endmodule
```

假设halfadder.v这个文件中的内容如下：

```
module halfadder;
  ……
endmodule
```

那么编译时，adder.v中的`include这一行将由文件“ . . / . . / primitives.v”的内容替代，adder.v的内容变成下面这样：

```
module halfadder;
  ……
endmodule
module adder;
  ……
endmodule
```

4. `timescale

在Verilog HDL模型中，所有时延都用单位时间表示。`timescale编译器指令定义时延单位和时延精度，其格式为：

```
`timescale time_unit / time_precision
```

其中，time_unit和time_precision由值1、10、和100以及单位s、ms、μs、ns、ps和fs组成，time_unit定义时延单位，time_precision定义时延精度。例如：

```
`timescale 1ns / 100ps
```

表示时延单位为1ns, 时延精度为100ps。

`timescale编译器指令在模块外部出现，并且影响后面所有的时延值。下面用一个带有时延定义的例子说明。

[例1-4]

```
`timescale 1ns/100ps
module AndFunc (Z,A,B) ;
  output Z;
  input A, B;
  and #(5.22, 6.17 ) Al(Z,A,B);          //规定了上升及下降时延值。
endmodule
```

编译器指令定义时延以ns为单位，并且时延精度为1/10ns（100ps）。因此，时延值5.22对应5.2ns（受精度限制，只精确到0.1ns），时延6.17对应6.2ns。如果把例1-4中的`timescale指令替换成下面这句：

```
`timescale 10ns /1ns
```

那么5.22对应52ns, 6.17对应62ns。

在编译过程中，`timescale指令影响这一编译器指令后面所有模块中的时延值，直至遇到另一个`timescale指令或`resetall指令。

当设计中多个模块带有自身的`timescale编译指令时将发生什么？在这种情况下，模拟器总是定位在所有模块的最小时延精度上，并且所有时延都相应地换算为最小时延精度。