GB

国家标准汇编

2001年制定

中国国家标准汇编

280

GB 18349～18359

（2001 年制定）

中国标准出版社

2002

中国国家标准汇编

280

GB 18349～18359

（2001 年制定）

中国标准出版社总编室　编

*

中国标准出版社出版

北京复兴门外三里河北街 16 号

邮政编码：100045

电话：68523946　68517548

中国标准出版社秦皇岛印刷厂印刷

新华书店北京发行所发行　各地新华书店经售

*

开本 880×1230　1/16　印张 46　字数 1 409 千字

2003 年 1 月第一版 2003 年 1 月第一次印刷

*

ISBN 7-5066-2927-5/TB・884

印数 1—2 000　定价 120.00 元

网址 www.bzcbs.com

*

科目 627—446

ISBN 7-5066-2927-5

9 787506 629270 >

出 版 说 明

1.《中国国家标准汇编》是一部大型综合性国家标准全集。自1983年起，按国家标准顺序号以精装本、平装本两种装帧形式陆续分册汇编出版。本《汇编》在一定程度上反映了我国建国以来标准化事业发展的基本情况和主要成就，是各级标准化管理机构，工矿企事业单位，农林牧副渔系统，科研、设计、教学等部门必不可少的工具书。

2. 本《汇编》收入我国正式发布的全部国家标准。各分册中如有顺序号缺号的，除特殊情况注明外，均为作废标准号或空号。

3. 由于本《汇编》的出版时间与新国家标准的发布时间已达到基本同步，我社将在每年出版前一年发布的新制定的国家标准，便于读者及时使用。出版的形式不变，分册号继续顺延。

4. 由于标准不断修订，修订信息不能在本《汇编》中得到充分和及时的反应，根据多年来读者的要求，自1995年起，在本《汇编》汇集出版前一年发布的新制定的国家标准的同时，新增出版前一年发布的被修订的标准的汇编版本，视篇幅分设若干分册。这些修订标准汇编的正书名、版本形式与《中国国家标准汇编》相同，但不占总的分册号，仅在封面和书脊上注明“20××年修订-1，-2，-3，…”字样，作为本《汇编》的补充。读者配套购买则可收齐前一年制定和修订的全部国家标准。

5. 由于读者需求的变化，自第201分册起，仅出版精装本。

本分册为第280分册，收入国家标准GB 18349～18359的最新版本。

中国标准出版社

2002年8月

目　　录

前　言

本标准等同采用(美国)电气与电子工程师协会 IEEE Std 1364—1995《基于硬件描述语言 Verilog 的标准硬件描述语言》,其技术内容与 IEEE Std 1364—1995 完全一致,标准结构也无任何改动,只是增加了附录 I"专用术语中英文对照表",以利于本标准专业术语的统一。

IEEE Std 1364—1995 标准已被世界各工业发达国家普遍接受,将其等同采用为我国国家标准将使我国的集成电路 CAD 软件技术在使用方法上与国际保持一致,有利于我国与国际集成电路设计技术标准化接轨,也有利于我国硬件描述语言按国际惯例规范化。

Verilog HDL(Verilog Hardware Description Language, Verilog HDL)是一项重要的集成电路 CAD 软件技术,是当前 HDL(硬件描述语言)设计方法学的基础,受到半导体及集成电路设计行业的普遍重视,世界各大半导体公司大多采用了该标准。当前国际微电子技术正在迅猛发展,我国正处在突破集成电路产业落后局面的关键时刻,而集成电路设计是有可能首先取得成功的行业,这一点在国内已取得共识。因此,本标准的发布实施必将会对我国的 CAD 发展有重要的推动作用。

硬件描述语言 Verilog 是 Philip R. Moorby 于 1983 年在英格兰阿克顿市的 Gateway Design Automation 硬件描述语言公司设计出来的,该公司在同一年发布了"Verilog HDL"及其模拟器,并于 1985 年推出改进产品 Verilog－XL。1989 年 12 月,Cadence 公司并购了这家公司,并于 1990 年将 Verilog HDL 公开,成立了 OVI(Open Verilog International)。OVI 是由 Verilog HDL 用户和 CAE 供应商组成的行业协会,负责推动和制定工业标准。Verilog HDL 在 1995 年 12 月 12 日被 IEEE 接纳为 IEEE std 1364—1995 工业标准(下文简称 Verilog)。

Verilog 从诞生起就与生产实际紧密结合在一起,具有结构清晰、文法简明、功能强大、高速模拟和多库支持等优点,并获得许多工具的支持,深受用户的喜爱。虽然另一种硬件描述语言 VHDL 于 1987 年首先成为 IEEE std 1076—1987 标准,并于 1993 年扩展为 IEEE std 1076—1993 标准,使它得到迅速发展,但是 Verilog 实际上是 IC 行业标准,特别是在 1995 年 12 月被 IEEE 接纳为正式标准后,使它成为一种很有竞争力的硬件描述语言。

Verilog IEEE std 1364—1995 标准包含的内容十分丰富,主要有:词法约定、数据类型、表达式、语义调度、赋值语句、门级建模和开关级建模、用户定义的基元、行为级建模、任务和函数、有名称的块和任务的停用、层次化结构、指定块、系统任务和函数、值变转储文件、编译指令、PLI TF 和 ACC 接口机制、ACC 程序定义及 ACC 程序的使用、TF 程序定义及 TF 程序的应用、VPI 程序定义及 VPI 程序的使用等内容。

本标准中黑体字和斜体字的说明见 1.3。

本标准的附录 A、附录 B、附录 C、附录 D、附录 E 都是标准的附录。

本标准的附录 F、附录 G、附录 H 和附录 I 都是提示的附录。

本标准由中国标准研究中心和北京理工大学提出。

本标准由全国信息技术标准化技术委员会归口。

本标准起草单位:北京理工大学和中国标准研究中心。

本标准主要起草人:刘明业、蒋敬旗、董连续、石峰、胡燕翔、叶梅龙、董国华、樊孝忠。

中华人民共和国国家标准

集成电路/计算机硬件描述语言 Verilog

GB/T 18349—2001

Integrated Circuit/Computer Hardware Description Language Verilog

1 概述

1.1 本标准的目的

本标准旨在作为 Verilog®硬件描述语言(HDL)的完整规范。本文档包括：

——Verilog HDL 所有结构的形式化语法和语义；

——模拟系统的任务和函数，例如文本输出显示命令；

——编译指令，例如文本替代宏和模拟的时间刻度；

——编程语言接口(PLI)的连接机制；

——存取例行程序、任务例行程序/函数例行程序以及 Verilog 的过程化接口例行程序的形式化语法和语义；

——应用实例；

——PLI 头文件列表。

1.2 本标准使用的一些约定

本标准分为多章，每章集中阐述该语言的某个特定问题。每章中各条论述单独的结构和概念。论述中首先是对结构或概念的介绍和基本原理，接着是语法和语义描述，以及一些举例和注释。

本标准中使用的动词“shall”表示强制要求，而动词“can”表示可选择性。这些动词对于不同的读者表示不同的含义：

a) 对于 Verilog HDL 的工具开发人员来说，动词“shall”表示标准强加的要求。要求最终实现的结果能够实现这些要求，如果要求不能得到满足就报错。

b) 对于 Verilog HDL 模型开发人员来说，动词“shall”表示 Verilog HDL 的特性是语言定义的自然结果。要求模型开发人员能够遵循这些特性所隐含的约束。动词“can”表示可选择性，模型开发人员可以随意操作。如果采用，则要求模型开发人员遵循语言定义所阐明的要求。

c) 对于使用 Verilog HDL 模型的用户来说，动词“shall”表示模型的特性是语言定义的自然结果。模型用户可以依赖 Verilog HDL 源文本隐含的模型特性。

1.3 语法描述

使用 Backus Naur 范式(BNF)进行 Verilog HDL 的形式化描述。使用以下约定：

a) 使用小写字母、下划线表示语法范畴。例如：

module_declaration

b) 使用黑体字表示保留的关键词、操作符，标点符号作为语法要求的一部分。这些字以较大的字体出现以便于区别。例如：

module => ;

c) 使用竖线将可选项分开，除非它以黑体字出现，此时它表示其本身。例如：

国家质量技术监督局 2001-04-09 批准　　2001-10-01 实施

```
unary _ operator ::=
        +|-|! |~|& |~& ||| ~||^ |~^ |^~
```

d) 使用方括号将选择项括起来。例如：

```
input _ declaration ::= input [range] list _ of _ variables ;
```

e) 使用大括号将重复项括起来，除非它以黑体字出现，此时它表示其本身。可选项可以出现 0 次或多次；重复项按从左到右的顺序出现，遵循左递归的规则。因此，下面的两条规则是相同的：

```
list _ of _ param _ assignments ::= param _ assignment { , param _ assignment }
list _ of _ param _ assignments ::=
                param _ assignment
                | list _ of _ param _ assignment , param _ assignment
```

f) 如果语法范畴的名称以斜体字开始，那么它与没有斜体字部分的范畴名称是相同的。斜体字部分是为了传达一些语义信息。例如，*msb* _ constant _ expression 和 *lsb* _ constant _ expression 等效于 constant _ expression。

当定义一个术语时，例如定义常量宽度字体或文件名时，或者当引用常数时，特别是引用 0、1、x 和 z 值时，主文本使用斜体字。

1.4 本标准的内容

各章和附录的大纲以快速参考的形式呈现。总共有 23 章和 9 个附录。所有章条和附录 A 到附录 E 都是本标准的正式内容。附录 F 到附录 I 只是为了提供一些信息。

第 1 章 概述

本章讨论本标准使用的约定及其内容。

第 2 章 词法约定

本章描述如何指定和解释词法符号。

第 3 章 数据类型

本章描述数据类型 net 和 reg。本章也讨论用作常数值的参数数据类型，并描述 net 上数值的驱动和电荷强度。

第 4 章 表达式

本章描述表达式中使用的操作符和操作数。

第 5 章 语义调度

本章描述 Verilog HDL 的语义调度。

第 6 章 赋值语句

本章对 Verilog HDL 中两种主要的赋值语句——持续赋值语句和过程赋值语句进行比较，描述将数值驱动到 net 上的持续赋值语句。

第 7 章 门级建模和开关级建模

本章描述门级建模、开关级基元和逻辑强度建模。

第 8 章 用户定义的基元(UDPs)

本章描述如何在 Verilog HDL 中定义基元以及这些基元如何包含在 Verilog HDL 模型中。

第 9 章 行为建模

本章描述过程赋值语句、过程持续赋值语句和行为语句。

第 10 章 任务和函数

本章描述任务和函数——在行为模型中可以从一个以上的地方进行调用的过程，描述如何像子程序一样使用任务，以及如何使用函数定义新的操作符。

第 11 章 有名称的块和任务的停用

本章描述如何停止使用任务和具有指定名称的语句块。

第 12 章　层次化结构

本章描述如何在 Verilog HDL 中建立层次，如何重载在模块中说明的参数值。

第 13 章　指定块

本章描述如何指定模块中输入端口和输出端口之间的时序关系。

第 14 章　系统任务和函数

本章描述系统任务和函数。

第 15 章　值变转储(VCD)文件

本章描述与值变转储(VCD)文件有关的系统任务和文件格式。

第 16 章　编译指令

本章描述编译指令。

第 17 章　PLI TF 和 ACC 的接口机制

本章描述一种接口机制，该接口机制提供一种方法使用户将 PLI 任务/函数(TF)例行程序和存取例行程序连接到 Verilog 软件工具上。

第 18 章　ACC 例行程序的使用

本章主要描述 ACC 例行程序，包括如何使用以及为什么使用这些 ACC 例行程序。

第 19 章　ACC 例行程序的定义

本章描述特殊的 ACC 例行程序，解释它们的功能、语法和使用方法。

第 20 章　TF 例行程序的使用

本章描述 TF 例行程序使用的操作类型。

第 21 章　TF 例行程序的定义

本章描述专有的 TF 例行程序，解释它们的功能、语法和使用方法。

第 22 章　VPI 例行程序的使用

本章描述 Verilog 编程接口(VPI)例行程序使用的操作类型。

第 23 章　VPI 例行程序的定义

本章描述 VPI 例行程序。

附录 A　形式化语法定义

本附录使用 BNF 描述 Verilog HDL 的语法。

附录 B　关键词列表

本附录列出 Verilog HDL 的关键词。

附录 C　acc _ user.h 文件

本附录列出 acc _ user.h 文件的内容。

附录 D　veriuser.h 文件

本附录列出 acc _ veriuser.h 文件的内容。

附录 E　vpi _ user.h 文件

本附录列出 vpi _ user.h 文件的内容。

附录 F　系统任务和函数

本附录描述经常用到的系统任务和函数，但这些并不是标准的一部分。

附录 G　编译指令

本附录描述经常用到的编译指令，但这些并不是标准的一部分。

附录 H　参考文献

本附录包括本标准的参考书目。

附录 I　专用术语中英文对照表

本附录包括本标准用到的中英文专用术语。

1.5 头文件清单

头文件清单包含在附录C、D和E中，分别是veriuser.h、acc_user.h和vpi_user.h，这些是本标准的正式内容。所有匹配的软件工具都应使用这些头文件中包含的相同的函数说明、常数定义和结构定义。

1.6 举例

本标准中给出一些用Verilog HDL和C编程语言描述的举例。这些举例用于说明在简单的环境下Verilog HDL结构和PLI函数的使用方法，并没有给出全部的语法定义。

1.7 先决条件

第17章至第23章和附录C至附录E以具有C编程语言的应用知识为先决条件。

2 词法约定

本章描述Verilog HDL源文本中使用的词法标记和它们的约定。

2.1 词法标记(**Lexical tokens**)

```
number ::=
      decimal_number
    | octal_number
    | binary_number
    | hex_number
    | real_number
decimal_number ::=
      [sign] unsign_number
    | [size] decimal_base unsigned_number
binary_number ::=
    [size] binary_base binary_digit { _ | binary_digit }
octal_number ::=
    [size] octal_base octal_digit { _ | octal_digit }
hex_number ::=
    [size] hex_base hex_digit { _ | hex_digit }
real_number ::=
      [size] unsigned_number.unsigned_number
    | [size] unsigned_number [.unsigned_number] e [sign] unsigned_number
    | [size] unsigned_number [.unsigned_number] E [sign] unsigned_number
sign ::=
    +|-
size ::=
    unsigned_number
unsigned_number ::=
    decimal_digit { _ | decimal_digit }
decimal_base ::=
    'd | 'D
binary_base ::=
    'b | 'B
octal_base ::=
    'o | 'O
hex_base ::=
    'h | 'H
decimal_digit ::=
    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
binary_digit ::=
    x | X | z | Z | 0 | 1
octal_digit ::=
    x | X | z | Z | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
hex_digit ::=
    x | X | z | Z | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f | A | B | C | D | E | F
```

语法2-1 整数和实数的语法

Verilog 源文本文件应当是一个词法标记流。一个词法标记应当包括一个或多个字符。源文件中的标记格式应当是自由格式——即除了转义标识符以外，空格和换行除了作为分隔符外，在语句构成上不应当具有特殊意义(见 2.7.1)。

Verilog 语言中词法标记有以下类型：

——空白；

——注释；

——操作符；

——数字；

——字符串；

——标识符；

——关键词。

2.2 空白(White space)

空白包括空格符、制表符、换行符、走纸符。这些字符除了用于分隔其它词法标记外没有另外意义。但是，间隔符和制表符在字符串中应看作是具有特殊意义的字符(见 2.6)。

2.3 注释(Comments)

Verilog HDL 具有两种注释形式。*单行注释*以符号//开始，以换行符结束。*块注释*以/* 开始，以 */结束。块注释不应当嵌套。单行注释标记//在块注释中没有特殊含义。

2.4 操作符(Operators)

在表达式中使用的操作符有一元字符、二元字符和三元字符。第 4 章讨论表达式中操作符的使用。

*一元操作符*应出现在操作数的左边。*二元操作符*出现在操作数之间。*条件操作符*具有两个操作符，用于将三个操作数分开。

2.5 数字(Numbers)

*常数*可表示为整常数和实常数。见语法 2-1。

2.5.1 整常数(Integer constants)

*整常数*可以用十进制、十六进制、八进制或二进制几种格式表示。

表示整常数的格式有两种。第一种格式是简单的十进制数，可表示为数字 0 到 9 的一个序列，可以用一元加或一元减操作符起始。第二种格式表示*长度固定的常数*，由三种标记组成——一个可选择的长度常数、一个后面跟随基数格式字符的单引号以及表示数值的多个数字。

第一个标记是一个长度固定的常数，它应当根据精确的位数表示常数的长度。它应当表示为一个无符号的十进制数。例如，两个十六进制数的长度指定是 8，因为一个十六进制数要求 4 位。

第二个标记是一个基数格式，它应当由表示数字基数的一个字母组成，前面加上一个单引号符(')。合法的基数指定为 d,D,h,H,o,O,b,B，分别作为十进制、十六进制、八进制和二进制的基数。

在定义一个数字的值时，x 和 z 的使用不区分大小写。

单引号和基数格式符号之间不应当有空格。

第三个标记是一个无符号的数字，它应当由合法基数格式的数字组成。无符号数字标记应直接跟在基数格式后面，前面可以有空格。十六进制数 a 到 f 应当不区分大小写。

不带有长度和基数格式的简单十进制数应视为*带符号的整数*，而由基数格式表示的数字应视为*无符号整数*。

在长度固定的常数前面的加号或减号是常数的符号；长度固定的常数不带符号。在基数格式和数字之间的加号或减号是非法的语法。

*负数*应当以 2 的补码的形式表示。

x 表示十六进制、八进制和二进制数中的*未知数*。*z* 表示*高阻*值。见 3.1 中对 Verilog HDL 数值集合的讨论。在十六进制中 *x* 应设置为 4 位未知数，在八进制中应设置为 3 位，在二进制中应设置为 1 位。同样，*z* 应分别设置为 4 位、3 位和 1 位的高阻值。

如果无符号数的长度比要表示的常数的长度小，则无符号数的左边应用 0 填充。如果无符号数最左边的位是 *x* 或者 *z*，则应当分别用 *x* 或者 *z* 填充到左边。

在 Verilog HDL 中，数字表示中使用的问号(?)可以替代表示字符 z。它在十六进制中设置为 4 位高阻值，在八进制中设置为 3 位，在二进制中设置为 1 位。在高阻值是无关项的情况下，可以使用问号增强可读性。见 9.5.1 中对 ***casez*** 和 ***casex*** 进行讨论。也可以在用户定义基元的状态表中使用问号。见 8.1.4。

下划线(_)除了作为第一个字符外，在一个数字的任何地方使用都是合法的。下划线字符可以忽略。可以用这个特性将很长的数字分隔开以便易于阅读。

举例：

长度不固定的常数

```
659          //  是一个十进制数
'h 837FF     //  是一个十六进制数
'o 7460      //  是一个八进制数
4af          //  非法(十六进制格式要求 'h)
```

长度固定的常数

```
4'b1001      //  是一个 4 位二进制数
5'D 3        //  是一个 5 位十进制数
3'b01x       //  是一个最低位为未知数的 3 位数
12'hx        //  是一个 12 位的未知数
16'hz        //  是一个 16 位的高阻数
```

使用符号的常数

```
8 'd -6      //  这是一个非法的语法
-8 'd 6      //  这定义出 6 的补码，
             //  是 8 位数，与 - (8'd b)等效
```

自动左填充

```
reg  [11:0] a, b, c, d;
initial begin
        a = 'h  x;    //  生成  xxx
        b = 'h  3x;   //  生成  03x
        c = 'h  z3;   //  生成  zz3
        d = 'h  0z3;  //  生成  0z3
```

在数字中使用下划线

```
27 _ 195 _ 000
16'b0011 _ 0101 _ 0001 _ 1111
32'h 12ab _ f001
```

注

1 当对一个寄存器数据类型进行赋值时，一个固定长度的负数是符号不扩展的数。

2 表示数字的三个标记都可用宏替换。

3 构成长度不固定数字的位数(可以是一个简单的十进制数或者是一个没有长度规范的数)应当至少为 32。

2.5.2 实常数(**Real constants**)

实常数 应表示为由 IEEE 标准——IEEE Std 754—1985[B1]描述的双精度浮点数。

实数可以用十进制计数法(例如，14.72)或者用科学计数法(例如，39e8 表示 39 乘以 10 的 8 次幂)表示。带小数点的实数在小数点的每一边至少有一个数字。

举例：

1.2
0.1
2394.26331
1.2E12（指数符号可以是 e 或 E ）
1.30e－2
0.1e－0
23E10
29E－2
23.6123 _ 763 _ e－12（下划线可以忽略）

下面的实数形式无效，因为在小数点的每一边至少应当有一个数字，而它们没有：

.12
9.
4.E3
.2e－7

2.5.3 转换(Conversion)

实数可以通过四舍五入转换成相近的整数，而不是将小数截断。当把实数赋值给整数时，应进行隐式转换。

举例：

当转换为整数时实数 35.7 和 35.5 都变成 36，而 35.2 变成 35。

－1.5 转换成整数时成为－2，而 1.5 转换成整数时成为 2。

2.6 字符串(Strings)

一个字符串是一个由双引号(“”)括起来并包含在单独一行的字符序列。在表达式和赋值语句中用作操作数的字符串应视为无符号整常数，由一个 8 位 ASCII 值的序列表示，而一个 8 位 ASCII 值表示一个字符。

2.6.1 字符串变量说明(String variable declaration)

字符串变量是寄存器类型(见 3.2)的变量，宽度等于字符串中的字符数乘以 8。

举例：

为了存储 12 个字符的字符串“Hello world!”，要求一个 8×12 或者 96 位宽的寄存器。

```
reg [8 * 12:1]  stringvar;
initial  begin
        stringvar = “Hello world!”;
end
```

2.6.2 字符串的操作(String manipulation)

字符串可以使用 Verilog HDL 操作符进行操作。操作符操作的值是 8 位 ASCII 值的序列。

举例：

```
module  string _ test;
reg  [8 * 14:1]  stringvar;
initial begin
        stringvar = “Hello world!”;
        $display ( “%s is stored as %h”, stringvar, stringvar);
        stringvar = { stringvar,“!!!”};
        $display (“%s is stored as %h”, stringvar, stringvar);
end
```

```
endmodule
```

输出是：

```
Hello world is stored as 00000048656c6c6f20776f726c64
Hello world!!! is stored as 48656c6c6f20776f726c64212121
```

注：当一个变量比要求保存的赋值还长时，那么赋值后左边要用 0 填充。这与非字符串赋值时的填充方法相同。如果一个字符串比目标字符串变量更长，那么字符串截断左边位，并丢失最左边字符。

2.6.3 字符串中的特殊字符(Special character in strings)

一些字符只能使用在字符串中，前面介绍的这样的字符称为转义字符。表 2-1 右栏列出这些字符，表示字符的转义序列在左栏。

表 2-1 字符串中特殊字符的说明

转义字符串	由转义字符串产生的字符
\n	换行字符
\t	制表符
\\	\字符
\"	"字符
\ddd	用 1～3 个八进制数字(0≤d≥7)表示的一个字符

2.7 标识符、关键词和系统名称(Identifiers, keywords and system names)

标识符 用于为一个对象给出一个唯一的名称以便引用。标识符应当是字母、数字、美元号($)和下划线(_)的任意序列。

标识符的第一个字符不能是数字或 $，可以是字母或下划线。标识符应当区分大小写。

举例：

```
shiftreg_a
busa_index
error_condition
merge_ab
_bus3
n$657
```

注：工具可以对标识符的最大长度进行限制，但至少应当为 1024 个字符。如果标识符超过工具规定的长度极限，应当报错。

2.7.1 转义标识符(Escaped identifiers)

转义标识符应当以反斜线符号(\)开始，以空白(空格、制表符、换行符)结束。这些转义标识符为打印标识符(十进制数 33 到 126，或者十六进制数 21 到 7E)中任何可打印的 ASCII 字符提供了一种方法。

前面的反斜线和后面的空白不是标识符的一部分。因此，转义标识符\cpu3 与非转义标识符 cpu3 是相同的。

举例：

```
\busa+index
\-clock
\* * *error-condition* * *
\net1/\net2
\{a,b}
\a*(b+c)
```

2.7.2 关键词(Keywords)

关键词是预定义的非转义标识符，用于定义语言的结构。Verilog HDL 关键词前面如果加上转义字符就不是关键词。

2.7.3 系统任务和函数(System tasks and functions)

$ 字符引进一种可以使用用户定义的任务和函数的语言结构。跟在 $ 后面的名称是系统任务或系统函数。

语法 2-2 给出 系统任务或函数的语法。

```
system_task_or_function ::=
        $system_task_identifier[(list_of_arguments)];
        | $system_function_identifier[(list_of_arguments)];
list_of_arguments::=
        argument{,[argument]}
argument::=
        expression
```

语法 2-2 系统任务和函数的语法

$ 标识符系统任务和函数可以在三种情况下定义：

—— $ 标识符系统函数和任务的标准集，如第 14 章所定义。

—— 附加的 $ 标识符系统函数和任务，可以使用 PLI 定义，如第 17、23、25 章中所描述。

—— 附加的 $ 标识符系统函数和任务，由软件工具定义。

除了这个结构外，任何有效的标识符，包括已经在上下文中使用的关键词，都可以用作系统任务或函数的名称。第 14 章中描述的函数和任务是本标准的一部分。带有 $ 标识符结构的附加的系统任务和函数不是本标准的一部分。

举例：

```
$display ("display a message");
$finish;
```

2.7.4 编译指令(Compiler directives)

`符号(ASCII 值为 60，称作打开符号(open quote)或重音符号(accent grave))引入 一个用于实现编译指令的语言结构。编译器一读到指令，由编译指令规定的编译行为立即生效。在编译过程中，指令一直保持有效，除非出现其它不同的编译指令。所以，一个描述文件中的编译指令可以控制多个描述文件中的编译行为。

`标识符编译指令结构可以在两种情况下定义：

—— `标识符编译指令的一个标准集，在第 16 章中定义。

—— 附加的 `标识符编译指令，由软件工具定义。

任何有效的标识符，除了这个结构外，包括已经在上下文中使用的关键词，都可以用作编译指令名称。在第 16 章中描述的编译指令是本标准的一部分。带有 `标识符结构的附加的编译指令不是本标准的一部分。

举例：

```
`define wordsize 8
```

3 数据类型

Verilog HDL 数据类型集的设计，是为了表示数字硬件中的数据存储和传输元件。

3.1 值集合

Verilog HDL 值集合由四种基本的值组成：

0—表示逻辑 0 或条件为假；

1—表示逻辑 1 或条件为真；

x—表示未知逻辑值；

z—表示高阻状态。

值 0 和 1 互为逻辑补码。

当 z 值出现在一个门的输入端或出现在一个表达式中时，其效果通常与 x 值的效果相同。金属氧化半导体(MOS)基元例外，它可以传递 z 值。

Verilog HDL 中几乎所有的数据类型均存储全部四种基本的值，只有*事件*(*event*)类型(见 9.7.3)例外，它没有存储。向量的每一位均可独立地设置成四种基本值中的一种。

除了线网变量的基本值信息外，本语言还包括强度信息。详细描述见本标准的第 7 章。

3.2 线网和寄存器

有两组主要的数据类型：寄存器(register)数据类型和线网(net)数据类型。这两组数据类型在赋值和保持值的方法上是不同的，它们表示不同的硬件结构。

3.2.1 线网

*线网*数据类型表示结构实体之间的物理连接，如门与门之间的连接。线网不存储值(trireg 线网除外)，它的值由其驱动器的值决定，例如一个持续赋值语句或者一个门。对这些结构的定义见第 6 章和第 7 章。如果没有驱动器连接到线网上，线网的值是高阻(z)，除非线网是一个 trireg，在这种情况下，它保持以前的驱动值。

语法 3-1 给出线网说明的语法。

本章描述线网说明的前两种形式。第三种形式称为线网赋值，在第 6 章描述。

3.2.2 寄存器

寄存器是数据存储元件的一种抽象。寄存器数据类型的关键词是 **reg**。寄存器存储从一个赋值语句到下一个赋值语句之间的值。在程序中，一个赋值语句相当于一个触发器，可以改变数据存储元件中的值。**reg** 数据类型的缺省的初始化值是未知值 x。

```
net_declaration ::=
    net_type [vectored | scalared][range][delay3] list_of_net_identifiers;
    | trireg [vectored | scalared][charge_strength][range][delay3]
    list_of_net_identifiers;
    | net_type [vectored | scalared][drive_strength][range][delay3]
                                        list_of_net_decl_assignments;
net_type ::= wire | tri | tri1 | supply0 | wand | triand | tri0 | supply1 | wor | trior
range ::= [msb_constant_expression : lsb_constant_expression ]
drive_strength ::=
     (strength0, strength1)
    | (strength1, strength0)
    | (strength0, highz1)
    | (strength1, highz0)
    | (highz1, strength0)
    | (highz0, strength1)
strength0 ::= supply0 | strong0 | pull0 | weak0
strength1 ::= supply1 | strong1 | pull1 | weak1
charge_strength ::= ( small ) | ( medium) | ( large)
delay3 ::= #delay_value | # (delay_value[,delay_value[,delay_value]])
delay_value ::= unsigned_number | parameter_identifier |
      constant_mintypmax_expression
list_of_net_decl_assignments ::= net_decl_assignment {,net_decl_assignment}
net_decl_assigment ::= net_identifier=expression
```

语法 3-1　线网说明的语法

```
reg _declaration ::= reg [range] list _of _register _identifiers ;
time _declaration ::= time  list _of _register _identifiers ;
integer _declaration ::= integer list _of _register _identifiers ;
real _declaration ::= real  list _of _real _identifiers ;
realtime _decalaration ::= realtime  list _of _real _identifiers ;
list _of _register _identifiers ::= register _name {, register _name}
register _name ::=
      register _identifier
       | memory _identifier [ upper _limit _constant _expression :
      lower _limit _constant _expression ]
```

语法 3-2 ***reg*** 说明的语法

语法 3-2 给出 reg 说明的语法。

如果一组线网或一组寄存器具有相同的特征，那么可以使用相同的说明语句对它们进行说明。

注意

可以把负值赋给寄存器，但是当寄存器是表达中的操作数时，其值可看作是一个无符号的(正)值。例如，如果寄存器是一个表达式的操作数，则 4 位寄存器中的负 1(−1)起到数字 15 的作用。有关表达中的数字约定的详细信息请见 4.1.3。

3.3 向量

无范围指定的线网或寄存器说明是 1 位位宽，是一个*标量*。具有多位的线网数据类型和寄存器数据类型通过指定一个范围进行说明，它是一个*向量*。

3.3.1 向量说明

范围指定给出多位线网或多位寄存器中的各个位的地址。由 *msb* 常数表达式指定的最高位是范围域值中的左边值，由 *lsb* 常数表达式指定的最低位是范围域值中的右边值。

msb 常数表达式和 lsb 常数表达式都是常数表达式。msb 常数表达式和 lsb 常数表达式可以是任意值——正值、负值或零。lsb 常数表达式可以大于、等于或小于 msb 常数表达式。

向量线网和向量寄存器遵守 n 次幂模数为 2 的算术规则(2^n)，其中，n 是向量中的位数。将向量线网和寄存器作为无符号量处理。

举例：

```
wand   w;                          //"wand"类型的标量线网
tri[15:0] busa;                    //三态 16 位总线
trireg(small) storeit;             //小强度的电荷存储节点
reg a;                             //标量寄存器
reg[3:0] v;                        //4 位向量寄存器，
                                   //由 v[3]、v[2]、v[1]和 v[0]
                                   //(从最高有效位到最低有效位)组成
reg[-1:4] b;                       //6 位向量寄存器
wire w1,w2;                        //说明两根连线
reg[4:0] x,y,z;                    //说明 3 个 5 位寄存器
```

注

1 不同的实现工具可以对向量的最大长度进行限制，但其最大长度至少为 65536(2^{16})位。

2 不同的实现工具没有必要诊断整数操作的溢出。

3.3.2 向量线网的可访问性

在向量线网或寄存器说明中，*vectored* 和 *scalared* 是可选的关键词。如果实现这些关键词，可以限