
# Verilog HDL语言规范

主 讲：何宾

**Email：hebin@mail.buct.edu.cn**

**2018.06**

# Verilog HDL系统任务和函数

根据系统任务和系统函数实现的功能不同，可将其分为以下几类：

- 显示任务
- 文件输入/输出任务
- 时间标度任务
- 仿真控制任务
- 仿真时间系统函数
- 可编程逻辑阵列建模任务
- 随机分析任务
- 变换函数
- 概率分布函数
- 命令行输入
- 数学函数

# Verilog HDL系统任务和函数
# --显示任务

显示系统任务用于信息显示和输出。这些系统任务进一步分为：

- 显示和写入任务
- 探测监控任务
- 连续监控任务

# Verilog HDL系统任务和函数
# --显示任务

## 显示和写入任务

- 显示任务将特定信息输出到标准输出设备，并且带有行结束字符。

  而写入任务输出特定信息时不带有行结束符。

- $display和$write系统任务的语法如下：

```
task_name (format_specification 1, argument_list 1 ,
           format_specification 2 , argument_list 2 ,
           …,
           format_specification N , argument_list N) ;
```

# Verilog HDL系统任务和函数
# --显示任务

其中：

task_name是如下编译指令的一种：

- $display
- $displayb
- $displayh
- $displayo
- $write
- $writeb
- $writeh
- $writeo

# Verilog HDL系统任务和函数 --显示任务

## 用于特殊字符的转义序列

- 如下表所示，转义序列用于打印特殊的字符。

| 参数 | 描述 |
| --- | --- |
| \n | 换行 |
| \t | 制表符 |
| \\ | 字符\ |
| \" | 字符” |
| \ddd | 3位八进制数表示的ASCII值 |
| %% | 字符% |

# Verilog HDL系统任务和函数
# --显示任务

转义序列用于打印特殊字符Verilog HDL描述的例子。

```
module disp;
initial begin
        $display("\\t\\n\"\123");
end
endmodule
```

仿真这个例子，将输出：

```
\   \
 "S
```

# Verilog HDL系统任务和函数 --显示任务

- 用于指定格式的转义序列
  - 下表给出转义序列用于指定格式。

| 输出格式符 | 格式说明 |
| --- | --- |
| %h 或%H | 以十六进制显示 |
| %d 或%D | 以十进制显示 |
| %o 或%O | 以八进制显示 |
| %b 或%B | 以二进制显示 |
| %c 或%C | 以ASCII字符形式显示 |

# Verilog HDL系统任务和函数
# --显示任务

| | |
|---|---|
| %v 或%V | 显示网络信号强度 |
| %l或%L | 显示库绑定信息 |
| %m 或%M | 显示分层名 |
| %s 或%S | 以字符串显示 |
| %t 或%T | 显示当前时间格式 |
| %u或%U | 未格式化的二值数据 |
| %z或%Z | 未格式化的四值数据 |

# Verilog HDL系统任务和函数
# --显示任务

- 如果没有指定参数格式说明，默认值如下：
  - $display与$write　：十进制数。
  - $display b与$writeb ：二进制数。
  - $display o与$writeo ：八进制数。
  - $display h与$writeh ：十六进制数。

# Verilog HDL系统任务和函数 --显示任务

- 下表给出了用于实数显示的指定格式。

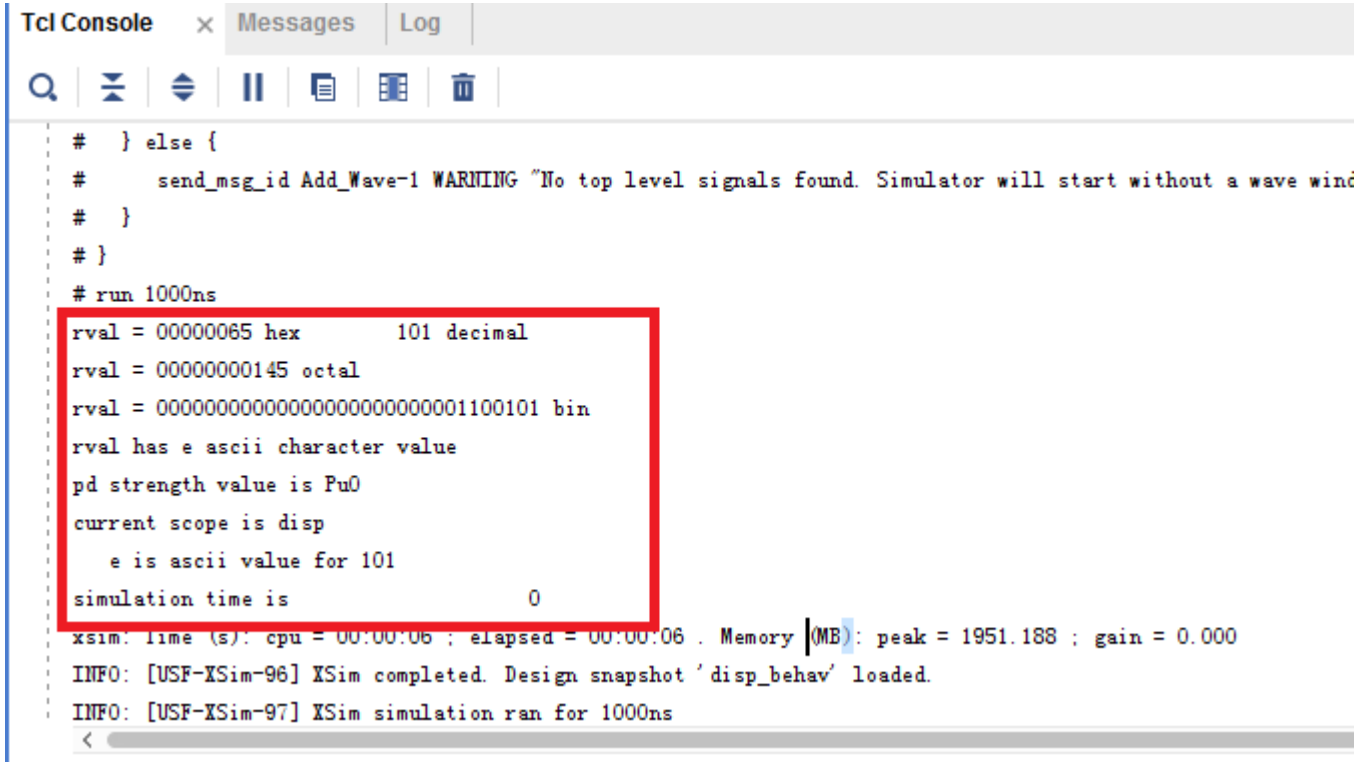
| 参数 | 描述 |
| --- | --- |
| %e 或%E | 指数格式显示实数 |
| %f 或%F | 十进制格式显示实数 |
| %g 或%G | 以上两种格式中较短的格式显示实数 |

# 显示任务
# --display任务的例子

```
module disp;
reg [31:0] rval;
pulldown (pd);
initial begin
        rval = 101;
        $display("rval = %h hex %d decimal",rval,rval);
        $display("rval = %o octal\nrval = %b bin",rval,rval);
        $display("rval has %c ascii character value",rval);
        $display("pd strength value is %v",pd);
        $display("current scope is %m");
        $display("%s is ascii value for 101",101);
        $display("simulation time is %t", $time);
    end
endmodule
```

# Verilog HDL系统任务和函数
# --显示任务

## 仿真这个例子，将显示下面的结果：



# Verilog HDL系统任务和函数
# --显示任务

## 所显示数据的位宽

- 对于表达式参数，写到输出文件（或者终端）时，自动的调整值的位宽。例如：12位表达式的结果：
  - 当以十六进制显示时，分配3个字符；
  - 当以十进制显示时，分配四个字符；

因为表达式的最大的可能值为FFF（十六进制）和4095（十进制）。

# Verilog HDL系统任务和函数
# --显示任务

- 当以十进制显示时，将前面的零去掉，并且用空格代替。对于其它基数，总是显示前面的零。
- 通过在%字符和表示基数的字符之间插入一个0，覆盖所显示数据自动的调整位宽。

```
$display("d=%0h a=%0h", data, addr);
```

# Verilog HDL系统任务和函数
# --显示任务

## 不同数据大小显示的Verilog HDL描述的例子

```
reg [11:0] r1;
initial begin
  r1 = 10;
  $display( "Printing with maximum size - :%d: :%h:", r1,r1 );
  $display( "Printing with minimum size - :%0d: :%0h:", r1,r1 );
end
endmodule
```

- 第一个$display是标准的显示格式。
- 第二个$display使用%0的格式。

# Verilog HDL系统任务和函数
# --显示任务

**仿真后的结果显示为：**

- **Printing with maximum size - : 10: :00a:**
- **Printing with minimum size - :10: :a:**

# Verilog HDL系统任务和函数
# --显示任务

当一个表达式的结果包含一个未知值或者高阻值的时候，下面的规则应用于显示值：

- 对于%d的格式，规则如下：
  - 如果所有位是未知值，显示单个小写x字符。
  - 如果所有位是高阻值，显示单个小写z字符。
  - 如果某些位，而非全部的位为未知值时，显示大写的X字符。
  - 如果某些位，而非全部的位为高阻值时，显示大写的Z字符。除非当某些位为未知值时，则显示大写字符X。
  - 在一个固定宽度的区域内，总是向右对齐。

# Verilog HDL系统任务和函数
# --显示任务

- 对于%h和%o的格式，规则如下：
  - 每个4比特位组表示一个十六进制数字；每个3比特位组表示一个八进制数字。
  - 如果一个组内的所有位都是未知值时，该进制的某个数字显示小写字母x。
  - 如果一个组内的所有位都是高阻值时，该进制的某个数字显示小写字母z。
  - 如果一个组内的某些位为未知值时，该进制的某个数字显示大写字母X。
  - 如果一个组内的某些位为高阻值时，该进制的某个数字显示大写字母Z。除非有一些位为未知值时，为该进制的某个数字显示大写字符X。

# Verilog HDL系统任务和函数
# --显示任务

## 显示未知值和高阻值Verilog HDL描述的例子。

| 描述 | 结果 |
| --- | --- |
| $display("%d", 1'bx); | x |
| $display("%h", 14'bx01010); | xxXa |
| $display("%h %o", 12'b001xxx101x01,12'b001xxx101x01); | XXX 1x5X |

# Verilog HDL系统任务和函数
# --显示任务

## 强度格式

- %v格式用于显示标量网络的强度。
- 对于每个%v来说，以字符串方式显示。
- 用三个字符格式，报告一个标量网络的强度。
- 前两个字符表示强度，第三个字符便是标量当前的逻辑值，其值为下表给出的值。

# Verilog HDL系统任务和函数
# --显示任务

| 参数 | 描述 |
| --- | --- |
| 0 | 用于逻辑0值 |
| 1 | 用于逻辑1值 |
| X | 用于一个未知值 |
| Z | 用于一个高阻的值 |
| L | 用于一个逻辑0或者高阻值 |
| H | 用于一个逻辑1或者高阻值 |

- 前两个字符-强度字符，或者是两个字母助记符或者是一对十进制数字。
- 通常，使用两个字符注记符表示强度信息。
- 然而，少量情况下使用一对十进制数字表示各种信号强度。

# Verilog HDL系统任务和函数
# --显示任务

## 下表给出用于表示不同强度级的助记符

| 注记符 | 强度名字 | 强度级 |
|---|---|---|
| Su | Supply驱动 | 7 |
| St | 强驱动 | 6 |
| Pu | Pull驱动 | 5 |
| La | 大的电容 | 4 |
| We | 弱驱动 | 3 |
| Me | 中电容 | 2 |
| Sm | 小电容 | 1 |
| Hi | 高阻 | 0 |

# Verilog HDL系统任务和函数
# --显示任务

- 提供了四种驱动强度和三种电荷存储强度。驱动强度与门输出和连续分配输出关联。电荷存储强度和trireg类型网络相关。
- 对于逻辑0和1来说，如果信号没有强度范围，则使用一个助记符。否则，逻辑值用两个十进制数字引导，表示最大和最小的强度级。
- 对于未知值，当0和1强度元件在相同的强度级时，使用一个注记符。否则，未知值X由两个十进制数字引导，分别用于表示0和1强度级。
- 高阻强度没有一个已知的逻辑值；用于这个强度的逻辑值是Z。
- 对于值L和H，使用一个助记符表示强度级。

# Verilog HDL系统任务和函数
# --显示任务

强度级显示Verilog HDL描述的例子。

```
always
#15 $display($time,"group=%b signals=%v %v %v",{s1,s2,s3},s1,s2,s3);
```

下面给出这样一个调用可能的输出：

```
 0  group=111 signals=St1 Pu1 St1
15  group=011 signals=Pu0 Pu1 St1
30  group=0xz signals=520 PuH HiZ
45  group=0xx signals=Pu0 65X StX
60  group=000 signals=Me0 St0 St0
```

# Verilog HDL系统任务和函数
# --显示任务

## 下表解释了输出中不同的强度格式

| | |
|---|---|
| St1 | 强驱动1值 |
| PuO | 一个pull驱动0值 |
| HiZ | 高阻状态 |
| MeO | 一个中电容强度的0电荷存储 |
| StX | 一个强驱动未知值 |
| PuH | 1或者高阻值的pul驱动强度 |
| 65X | 带有一个强驱动0元件和一个上拉驱动1元件的未知值 |
| 520 | 范围从pull驱动到中电容的0个值 |

# Verilog HDL系统任务和函数
# --显示任务

## 层次化名字格式

- %m格式标识符不接受一个参数。它使得显示任务打印模块、任务、函数或者命名块的层次名字。
- 它调用包含格式标识符的系统任务。
- 当有很多模块例子调用系统任务时，这是非常有用的。
- 一个明确的应用是一个触发器或者锁存器模块内的时序检查消息，%m格式标识符精确的找到模块例化，该模块例化负责时序检查消息。

# Verilog HDL系统任务和函数
# --显示任务

## 字符格式

- %s格式标识符用于打印ASCII码的字符。
- 对于每个%s标识符，以一个字符串显示。
- 参数列表中，相应的参数将跟随字符。将相关的参数理解为一个8位十六进制ASCII码，每8位表示一个字符。
- 如果参数是一个变量，它的值右对齐，最右的值是字符串最后字符的最低有效位。在字符串的末尾不要求终止符，不打印前面的0。在字符串的末尾不要求终止符，不打印前面的0。

# Verilog HDL系统任务和函数
# --显示任务

## 探测任务

- 探测任务包含：
  - $strobe
  - $strobeb
  - $strobeh
  - $strobeo

# Verilog HDL系统任务和函数
# --显示任务

- 这些系统任务在一个选择的时间显示仿真数据。
- 但是，执行这种任务是在选择时间结束时才显示仿真数据。时间结束意味着完成选择时间内处理的所有事件。

# Verilog HDL系统任务和函数
# --显示任务

## $strobe任务的Verilog HDL描述

```
forever @(negedge clock)

$strobe ("At time %d, data is %h",$time,data);
```

该例子中，在时钟的每个下降沿，$strobe写时间和数据信息到标准的输出和日志文件。

# Verilog HDL系统任务和函数
# --显示任务

## 监控任务

- 监控任务有：
  - $monitor
  - $monitorb
  - $monitorh
  - $monitoro

# Verilog HDL系统任务和函数
# --显示任务

- $monitor任务提供了监控和显示任何变量或者表达式的值，这些值作为任务指定的参数。
- 该任务的参数和$display系统任务指定参数的行为一样，包括：用于特殊字符的转义序列和格式说明。
- 在任意时刻，对于特定的变量，只激活一个监控任务。可以用如下两个系统任务打开和关闭监控。

```
$monitor off;     //禁止所有监控任务。

$monitor on;      //使能所有监控任务。
```

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

用于文件操作的系统任务和函数分为下面的类型：

- 打开和关闭文件的函数和任务。
- 输出值到文件的任务
- 输出值到变量的任务
- 从文件中读取值，然后加载到变量或者存储器的任务和函数

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

## 打开和关闭文件

- 在Verilog HDL中，系统函数$fopen用于打开一个文件，其语法格式如下：

<file_descriptor>=$fopen(“<file_name>”，type);

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

**其中:**

- **file_name**

**指定被打开的文件名及其路径。是一个字符串，或者是一个reg，包含一个用于命名所要打开的文件名。**

- **type**

**文件类型。下表给出了文件描述符的类型。类型b用于区分打开的是文本文件还是二进制文件。**

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

| 参数 | 描述 |
| --- | --- |
| “r”或者“rb” | 打开文件，用于读 |
| “w”或者“wb” | 截断到长度零，或者创建文件用于写 |
| “a”或者“ab” | 添加，打开用于在文件的末尾写或创建用于写 |
| “r+”,“r+b”或者“rb+” | 打开用于更新（读和写） |
| “w+”,“w+b”或者“wb+” | 截断或者创建用于更新 |
| “a+”,“a+b”或者“ab+” | 添加，打开或者创建用于在文件末尾更新 |

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

- file_descriptor

文件描述符。如果路径与文件名正确，则返回一个32位的多通道描述符或者一个32位的文件描述符。

- 如果没有指定文件类型，打开文件用于写，返回一个多通道的描述符mcd。mcd是32位reg，设置其中的一位表示打开哪个文件。mcd的LSB总是引用标准输出。多个用多通道符打开的文件，通过对mcd按位OR，将其写到结果的值中。保留mcd的MSB，总是为0。用多通道描述符限制打开最多31个文件，用于输出。

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

- 如果指定type，打开指定类型的文件，返回文件描述符fd。fd是一个32位的值，保留fd的MSB，总是设置为1。这允许实现文件的输入和输出功能，以确定文件打开的方式。fd剩下的比特位用于表示打开什么文件。

  不像多通道描述符那样，文件描述符不能按位OR。

  如果不能打开文件，则返回0。通过调用$error，以确定不能打开文件的原因。

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

- 3个预先打开的文件描述符，分别是：
  - STDIN

    其值为32’h8000_0000，用于读。
  - STDOUT

    其值为32’h8000_0001，用于写。
  - STDERR。

    其值为32’h8000_0002，用于添加。

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

在Verilog HDL提供了$fclose系统任务，用于关闭文件，其语法格式为：

```
$fclose(file_handle)；
```

- 当使用多个文件时，为了提高速度，可以将一些不再使用的文件关闭。
- 一旦关闭某个文件，则不能再向它写入任何信息，并且打开的其他文件可以使用该文件的句柄。

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

## 文件输出系统任务

- 显示、写入、探测和监控系统任务都有一个用于向文件输出的相应副本，该副本可用于将信息写入文件。
- Verilog HDL中用来将信息输出到文件的系统任务有：
  - $fdisplay
  - $fwrite
  - $fstrobe
  - $fmonitor

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

它们具有如下相同的语法格式：

```
task_name(file_handles,format_specifiers);
```

其中：

- task_name是上述四种系统任务中的一种。
- file_handles是文件句柄描述符，与打开文件所不同的是，可以对句柄进行多位设置。
- format_specifiers用来指定输出格式。

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

## 设置多通道描述符Verilog HDL描述的例子

```
integer
   messages, broadcast,cpu_chann, alu_chann, mem_chann;
initial begin
      cpu_chann = $fopen("cpu.dat");
      if (cpu_chann == 0) $finish;
            alu_chann = $fopen("alu.dat");
      if (alu_chann == 0) $finish;
            mem_chann = $fopen("mem.dat");
```

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

```
    if (mem_chann == 0) $finish;
        messages = cpu_chann | alu_chann | mem_chann;
    // broadcast包含标准的输出
    broadcast = 1 | messages;
end
endmodule
```

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

## 文件输出系统任务Veriog HDL描述的例子

```
$fdisplay( broadcast, "system reset at time %d", $time );

$fdisplay( messages, "Error occurred on address bus",
                " at time %d, address = %h", $time, address );

forever @(posedge clock)
   ( alu_chann, "acc= %h f=%h a=%h b=%h", acc, f, a, b );
```

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

格式化数据到一个字符串的$swrite的命令格式如下：

string_output_task_name ( output_reg , list_of_arguments ) ;

- string_output_task_name

  输出任务名字，包括：$swrite、$swriteb、$swriteh、$swriteo。

- output_reg

  输出寄存器变量名字。

- list_of_arguments

  参数列表。$swrite的第一个参数是一个寄存器类型的变量，用于保存写入的字符串。

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

**$sformat的命令格式如下：**

**$sformat ( output_reg , format_string , list_of_arguments ) ;**

**这两个命令的主要一个区别：$sformat总是理解第二个参数，只有第二个参数作为一个格式化字符串。这个格式参数是一个静态字符串，比如：“data is %d”，或者用于保存一个格式化串的reg。它支持$display支持的所有格式描述符。比如：**

**$sformat(string, "Formatted %d %x", a, b);**

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

## 从文件中读取数据

- 一次读一个字符

一次读取一个字符Veriog HDL描述的例子1

```
c = $fgetc ( fd );
```

如果读取发生错误，则将c设置为EOF（-1）。调用$ferror，可以确定读取错误的原因。

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

## 一次读取一个字符Veriog HDL描述的例子2

```
code = $ungetc ( c, fd );
```

- 将c指定的字符插入到文件描述符fd指定的缓冲区。字符将在对该fd的$fgetc的调用时返回。文件本身并不变化。
- 如果发生错误，则将一个字符仍给一个文件描述符，并且将code设置为EOF。否则，将code设置为0。

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

### 一次读一行

一次读一行Verilog HDL描述的例子

```
integer code ;

code = $fgets ( str, fd );
```

- 从fd指定的文件中，将字符读入寄存器类型的str，直到将str填充满，或者读到新的一行，或者遇到EOF条件。
- 如果str不是整数字节，不使用最高有效的部分字节，这样用于确定宽度。
- 如果发生错误，则将code设置为0。否则，将code设置为读取的字符个数。

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

- 读格式化数据
  - 读格式化数据的Verilog HDL描述的例子

```
integer code ;

code = $fscanf ( fd, format, args );

code = $sscanf ( str, format, args );
```

  - $fscanf从文件描述符fd指定的文件中读。
  - $sscanf从reg str中读取。

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

所有的函数读取字符，然后根据格式理解字符，并保存结果。如果参数太小，不能保存转换后的输入，通常情况下，则传输最低有效位。

- Verilog HDL支持任何长度的参数。
- 然而，如果目标是real或者realtime，传输+Inf/-Inf。格式可以是字符串常数，也可以是保存一个字符串内容的reg。字符串保存转换的规范。

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

**格式符包括：**

%、b、o、d、h/x、f/e/g、v、t、c、s、u、z、m。

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

- 读二进制数据

```
integer code ;
code = $fread( myreg, fd);
code = $fread( mem, fd);
code = $fread( mem, fd, start);
code = $fread( mem, fd, start, count);
code = $fread( mem, fd, , count);
```

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

**其中：**

- start是可选项，说明在存储器中加载的第一个元素的地址。
- count是可选项，存储器中可以加载的最大的位置的数目。
- 如果加载的是reg类型，则忽略start和count。
- 读回的数据是大端方式。
- 从文件加载的是二值数据。
- 如果读取错误，将code设置为0。

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

## 文件定位

$ftell的Verilog HDL描述的例子

```
integer pos ;
pos = $ftell ( fd );
```

返回fd所指向文件从开始到当前的位置。将被后面的文件描述符读或者写。$fseek用于重新定位到该位置。任何重定位可以通过$ungetc操作取消。如果发生错误，返回EOF。

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

## $fseek的Verilog HDL描述的例子

```
code = $fseek ( fd, offset, operation );
code = $rewind ( fd );
```

设置fd指向文件的下一个输入/输出的位置。下一个位置是从开始，从当前，或者从文件结束的有符号的距离偏置。由operation确定：

- 当设置为0时，位置等于偏置字节。
- 当设置为1时，位置等于当前位置加偏置。
- 当设置为2时，位置到EOF加偏置。

$rewind等价于$fseek（fd，0，0）。

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

## 刷新输出

刷新输出的Veriog HDL描述的例子

```
$fflush ( mcd );

$fflush ( fd );

$fflush ( );
```

将任何缓冲的输出写入到mcd或者fd指向的文件。如果没有参数，则写到所有打开的文件。

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

## I/O错误状态

I/O错误状态的Verilog HDL描述的例子

```
integer errno ;
errno = $ferror ( fd, str );
```

最近的文件I/O操作的错误类型的字符串描述，写入到str中，它应该至少为640比特宽度。错误的代码值保存在errno中。如果最近的操作没有错误，则error返回0，将str清空。

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

## 检测EOF

- 检测文件结束Verilog HDL描述的例子。

```
integer code;

code = $feof ( fd );
```

当检测到文件结束时，返回非0的值；否则，返回0。

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

## 一个对文件进行读操作Verilog HDL描述的例子

```
module readFile(clk,reset,dEnable,dataOut,done);
parameter size = 4;
//to Comply with S-block rules which is a 4x4 array will
   multiply by
// size so row is the number of size bits wide
parameter bits = 8*size;
input clk,reset,dEnable;
output dataOut,done;
```

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

```
wire [1:0] dEnable;
reg dataOut,done;
reg [7:0] addr;
integer file;
reg [31:0] c;
reg eof;
always@(posedge clk)
begin
        if(file == 0 && dEnable == 2'b10)begin
                file = $fopen("test.kyle");
        end
end
```

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

```
always@(posedge clk) begin
      if(addr>=32 || done==1'b1)begin
         c <= $fgetc(file);
         eof <= $feof(file);
         addr <= 0;
      end
end
always@(posedge clk)
begin
       if(dEnable == 2'b10)begin
          if($feof(file))
```

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

```
 done <= 1'b1;
    else
        addr <= addr+1;
  end
end
//done this way because blocking statements should not
  really be used
always@(addr)
begin:Access_Data
        if(reset == 1'b0) begin
              dataOut <= 1'bx;
              file <= 0:
```

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

```
        end
      else
   if(addr<32)
         dataOut <= c[31-addr];
end
endmodule
```

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

## Verilog HDL中提供两个系统任务：$readmemb和$readmemh

- 用于从指定的文本文件中读取数据，并将数据加载到指定的存储器中。
- 这两个系统任务的区别在于：
  - $readmemb要求以二进制数据格式存放数据文件。
  - $readmemh要求以十六进制数据格式存放数据文件。

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

它们具有相同的语法格式，格式为：

```
task_name(file_name,memory_name,start_addr,end_addr);
```

其中：

- task_name

  用来指定系统任务，为$readmemb或$readmemh。

- file_name

  读出数据的文件名。

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

- **memory_name**

  为要读入数据的存储器的名字。

- **start**

  存储器的起始地址。实际就是建模存储器数组的索引值。

- **end**

  存储器的结束地址。实际就是建模存储器数组的索引值。

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

## 从文件中读取存储器数据Verilog HDL描述的例子

```
reg [7:0] mem[1:256];

initial $readmemh("mem.data", mem);
initial $readmemh("mem.data", mem, 16);
initial $readmemh("mem.data", mem, 128, 1);
```

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

- 第一句话，没有显式声明地址。在仿真时间0时刻，开始在存储器地址1开始加载数据。
- 第二句话，声明起始地址，但没有声明结束地址。在地址16开始加载数据，连续向上到地址256。
- 第三句话，声明开始地址和结束地址。如果开始地址大于起始地址，则地址递减。在地址128开始加载数据，连续向下到地址1为止。

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

## 从SDF文件中加载时序数据

## $sdf_annotate系统任务的语法格式：

```
$sdf_annotate ("sdf_file" [ , [ module_instance ] [ , [ "config_file" ]
                [ , [ "log_file" ] [ , [ "mtm_spec" ]
                [ , [ "scale_factors" ] [ , [ "scale_type" ] ] ] ] ] ] ] );
```

**其中：**

- **sdf_file**

要打开的sdf文件名，由字符串表示，或者保存在包含文件名字字符串的寄存器类型中。

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

## module_instance

可选参数，说明在SDF文件中的注解的范围。SDF注解器使用指定例化的层次级，运行注解，并且允许数组索引。如果没有指定该参数，SDF注解器使用包含该系统任务调用的模块作为module_instance。

## config_file

可选字符串参数。提供了一个配置文件的名字。这个文件的信息可以用于提供对注解很多方面的详细控制。

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

- **log_file**

可选字符串参数。提供了在SDF注解期间，日志文件的名字。来自SDF文件的时序数据的每一个注解，导致在日志文件中的一个入口。

- **mtm_spec**

可选字符串参数。指定min/typ/max的哪个将要被注释。下表给出了mtm_spec的参数。

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

| 关键字 | 描述 |
| --- | --- |
| **MAXIMUM** | 注解最大值 |
| **MINIMUM** | 注解最小值 |
| **TOOL_CONTROL (默认)** | 注解由仿真器选择的值 |
| **TYPICAL** | 注解典型的值 |

# Verilog HDL系统任务和函数
# --文件输入-输出系统任务和函数

- **scale_factors**

可选的字符串参数。当注解时序值的时候使用scale_factors。比如：“1.6:1.4:1.2”，将使得最小值乘以1.6，典型值乘以1.4，最大值乘以1.2。默认值，“1.0:1.0:1.0”。scale_factors覆盖配置文件中的任何SCALE_FACTORS。

- **scale_type**

可选的字符串参数。指定将scale_factor如何用于min/typ/max。下表给出了scale_type的参数。

# Verilog HDL系统任务和函数
## --文件输入-输出系统任务和函数

| 关键字 | 描述 |
| --- | --- |
| **FROM_MAXIMUM** | 将scale_factor应用到最大值 |
| **FROM_MINIMUM** | 将scale_factor应用到最小值 |
| **FROM_MTM (默认)** | 将scale_factor应用到min/typ/max值 |
| **FROM_TYPICAL** | 将scale_factor应用到典型值 |

# Verilog HDL系统任务和函数
## --时间标度系统任务

Verilog HDL提供了两种时间标度任务函数：

- $printtimescale
- $timeformat。

# Verilog HDL系统任务和函数
## --时间标度系统任务

## $printtimescale

- $printtimescale系统任务显示用于特殊模块的时间单位和精度。
- 其语法格式如下：

$printtimescale（module_hierarchical_name）；

其中：

- module_hierarchical_name：

为模块层次化名字。如果没有指定参数，则输出包含该任务调用的所有模块的时间单位与精度。

# Verilog HDL系统任务和函数
## --时间标度系统任务

以下面的格式显示时间标度信息：

Time scale of (module_name) is unit / precision

# Verilog HDL系统任务和函数
## --时间标度系统任务

### $printtimescale的Verilog HDL描述的例子

```
`timescale 1 ms / 1 us
module a_dat;
initial
      $printtimescale(b_dat.c1);
endmodule

`timescale 10 fs / 1 fs
module b_dat;
        c_dat c1 ();
endmodule
```

```
`timescale 1 ns / 1 ns
module c_dat;
.
.
.
endmodule
```

运行后的显示结果为：
Time scale of (b_dat.c1) is 1ns / 1ns

# Verilog HDL系统任务和函数
## --时间标度系统任务

### $timeformat

系统任务函数$timeformat指定以%t格式定义如何报告时间信息。

该任务语法格式如下：

```
$timeformat(units,precision,suffix,numeric_field_width);
```

其中：

- units

用于指定时间单位，其取值范围0~-15，各值所代表的时间单位如下表所示。

# Verilog HDL系统任务和函数
## --时间标度系统任务

| 取值 | 时间单位 | 取值 | 时间单位 |
|---|---|---|---|
| 0 | 1 s | -8 | 10 ns |
| -1 | 100 ms | -9 | 1 ns |
| -2 | 10 ms | -10 | 100 ps |
| -3 | 1 ms | -11 | 10 ps |
| -4 | 100 us | -12 | 1 ps |
| -5 | 10 us | -13 | 100 fs |
| -6 | 1 us | -14 | 10 fs |
| -7 | 100 ns | -15 | 1 fs |

# Verilog HDL系统任务和函数
## --时间标度系统任务

- **precision**

  指定所要显示时间信息的精度。

- **suffix**

  诸如“ms”、“ns”之类的字符。

- **numeric_field_width**

  说明时间信息的最小字符数。

# Verilog HDL系统任务和函数
## --时间标度系统任务

### $timeformat的Verilog HDL描述的例子

```
`timescale 1 ms / 1 ns
module cntrl;
initial
      $timeformat(-9, 5, " ns", 10);
endmodule

`timescale 1 fs / 1 fs
module a1_dat;
reg in1;
integer file;
buf #10000000 (o1,in1);
```

# Verilog HDL系统任务和函数
## --时间标度系统任务

```
initial begin
file = $fopen("a1.dat");
#00000000 $fmonitor(file,"%m: %t in1=%d o1=%h", $realtime,in1,o1);
#10000000 in1 = 0;
#10000000 in1 = 1;
end
endmodule
`timescale 1 ps / 1 ps
module a2_dat;
reg in2;
integer file2
buf #10000 (o2,in2);
```

# Verilog HDL系统任务和函数
## --时间标度系统任务

```
initial begin

file2=$fopen("a2.dat");

#00000 $fmonitor(file2,"%m: %t in2=%d o2=%h",$realtime,in2,o2);

#10000 in2 = 0;

#10000 in2 = 1;

end

endmodule
```

# Verilog HDL系统任务和函数
## --时间标度系统任务

执行完后，

- 文件a1.dat的内容：

a1_dat: 0.00000 ns in1= x o1=x

a1_dat: 10.00000 ns in1= 0 o1=x

a1_dat: 20.00000 ns in1= 1 o1=0

a1_dat: 30.00000 ns in1= 1 o1=1

# Verilog HDL系统任务和函数
## --时间标度系统任务

- 文件a2.dat的内容：

```
a2_dat: 0.00000 ns in2=x o2=x
a2_dat: 10.00000 ns in2=0 o2=x
a2_dat: 20.00000 ns in2=1 o2=0
a2_dat: 30.00000 ns in2=1 o2=1
```

# Verilog HDL系统任务和函数
# --仿真控制任务

## Verilog HDL提供了两个仿真控制系统任务：

- $finish
- $stop

# Verilog HDL系统任务和函数
# --仿真控制任务

## $finish

- 系统任务$finish使仿真器退出，并将控制返回到操作系统。
- 语法格式为：

$finish [ ( n ) ] ;

其中：

- n=0，不打印任何信息。
- n=1，打印仿真时间和位置。
- n=2，打印仿真时间、位置，以及仿真时，CPU和存储器的利用率。

# Verilog HDL系统任务和函数
# --仿真控制任务

## $stop

- 系统任务$stop挂起仿真。
- 在这一阶段，可能将交互命令发送到仿真器。
- 语法格式为：

$stop [ ( n ) ] ;

# Verilog HDL系统任务和函数
## --可编程逻辑阵列建模系统任务

Verilog HDL提供了一组系统任务用于对可编程逻辑阵列（Programmable Logic Array，PLA）进行建模。

- 语法格式为：

```
$array_type$logic$format ( memory_identifier , input_terms ,
                                    output_terms ) ;
```

其中：

- $array_type$logic$format

  表示PLA系统建模任务类型。

# Verilog HDL系统任务和函数
## --可编程逻辑阵列建模系统任务

下表给出了PLA建模系统任务

| | | | |
|---|---|---|---|
| $async$and $array | $sync$and $array | $async$and $plane | $sync$and $plane |
| $async$nand $array | $sync$nand $array | $async$nand $plane | $sync$nand $plane |
| $async$or $array | $sync$or $array | $async$or $plane | $sync$or $plane |
| $async$nor $array | $sync$nor $array | $async$nor $plane | $sync$nor $plane |

# Verilog HDL系统任务和函数
## --可编程逻辑阵列建模系统任务

- array_type：

  包含：sync、async

- logc：

  包含：and、or、nand、nor

- format：

  包含：array、plane。当使用array时，只有来自memory_identifier比特组合中带有‘1’的值和输入数据进行比较。如果使用plane时，‘0’和‘1’都是重要的。可以通过‘z’和‘?’字符指定没有考虑 的值。

# Verilog HDL系统任务和函数
## --可编程逻辑阵列建模系统任务

- 0-输入数据的补。
- 1-输入的真值。
- x-输入值的最坏情况。
- z/?：不考虑。

# Verilog HDL系统任务和函数
## --可编程逻辑阵列建模系统任务

- memory_identifier

存储器标识符。将其声明为寄存器类型。它和输入项集合的宽度一样， 与输出项的深度一样。

可以通过使用系统任务$readmemb或$readmemh，从文本文件中将内容加载到存储器中。也可以通过过程分配语句，将内容分配给memory_identifier。声明格式为：

```
reg [1:n] mem[1:m];
```

# Verilog HDL系统任务和函数
## --可编程逻辑阵列建模系统任务

- input_terms

  输入项集合。

- output_terms

  输出项集合。

# Verilog HDL系统任务和函数
## --可编程逻辑阵列建模系统任务

### 异步系统和同步系统调用Verilog HDL描述的例子

```
wire a1, a2, a3, a4, a5, a6, a7;
reg b1, b2, b3;
wire [1:7] awire;
reg [1:3] breg;
$async$and$array(mem,{a1,a2,a3,a4,a5,a6,a7},{b1,b2,b3});
```

或者

```
$async$and$array(mem,awire, breg);
```

# Verilog HDL系统任务和函数
## --可编程逻辑阵列建模系统任务

同步系统调用的例子：

```
$sync$or$plane(mem,{a1,a2,a3,a4,a5,a6,a7}, {b1,b2,b3});
```

同步形式用于控制时间，在这个时间时，将重新评估逻辑阵列，然后更新输出。异步形式，当输入项改变的时候，就自动地执行评估。

# Verilog HDL系统任务和函数
## --可编程逻辑阵列建模系统任务

### PLA建模的Verilog HDL描述的例子1

```
module pla(a0, a1, a2, a3, a4, a5, a6, a7, b0, b1, b2);
input a0, a1, a2, a3, a4, a5, a6, a7;
output b0, b1, b2;
reg b0, b1, b2;
reg [7:0] mem[0:2];
initial begin
        mem[0] = 8'b11001100;
        mem[1] = 8'b00110011;
        mem[2] = 8'b00001111;
        $async$and$array(mem, {a0,a1,a2,a3,a4,a5,a6,a7},
                              {b0,b1,b2});
end
endmodule
```

# Verilog HDL系统任务和函数
## --可编程逻辑阵列建模系统任务

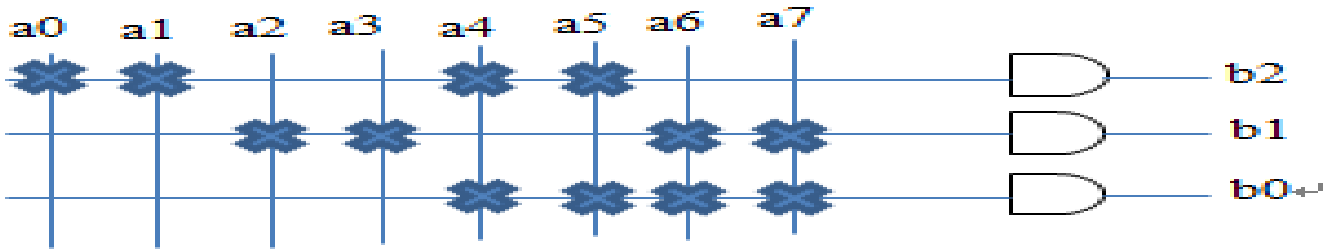
PLA有8位输入([7:0]) 总线和3位输出([0:2])总线，声明为reg (b0, b1, b2)。这个PLA 器件是一个异步的与操作。

| a0 | a1 | a2 | a3 | a4 | a5 | a6 | a7 | |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | b2 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | b1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | b0 |

b2=a0 & a1 & a4 & a5

b1=a2 & a3 & a6 & a7

b0=a4& a5 & a6 & a7



# Verilog HDL系统任务和函数
## --可编程逻辑阵列建模系统任务

**该器件的仿真结果表示为：**

**A = {a0, a1, a2, a3, a4, a5, a6, a7}**
**B = {b0, b1, b2}**
**mem[0] = 8'b11001100;**
**mem[1] = 8'b00110011;**
**mem[2] = 8'b00001111;**
**A = 11001100 -> B = 100**
**A = 00110011 -> B = 010**
**A = 00001111 -> B = 001**
**A = 10101010 -> B = 000**
**A = 01010101 -> B = 000**
**A = 11000000 -> B = 000**
**A = 00111111 -> B = 011**

# Verilog HDL系统任务和函数
## --可编程逻辑阵列建模系统任务

如果换成了$async$and$plane，则仿真结果变成：

A = {a0, a1, a2, a3, a4, a5, a6, a7}

B = {b0, b1, b2}

mem[0] = 8'b11001100;

mem[1] = 8'b00110011;

mem[2] = 8'b00001111;

A = 11001100 -> B = 100

A = 00110011 -> B = 010

A = 00001111 -> B = 001

A = 10101010 -> B = 000

A = 01010101 -> B = 000

A = 11000000 -> B = 000

A = 00111111 -> B = 000

# Verilog HDL系统任务和函数
## --可编程逻辑阵列建模系统任务

这个例子的同步版本表示为：

```
module pla(a0, a1, a2, a3, a4, a5, a6, a7, b0, b1, b2);
input a0, a1, a2, a3, a4, a5, a6, a7;
output b0, b1, b2;
reg b0, b1, b2;
reg [7:0] mem[0:2];
initial begin
        mem[0] = 8'b11001100;
        mem[1] = 8'b00110011;
        mem[2] = 8'b00001111;
forever @(posedge clk)
  $sync$and$array(mem, {a0,a1,a2,a3,a4,a5,a6,a7}, {b0,b1,b2});
end
endmodule
```

# Verilog HDL系统任务和函数
## --可编程逻辑阵列建模系统任务

### PLA建模的Verilog HDL描述的例子2

逻辑表达式：

```
b[1] = a[1] & ~a[2];
b[2] = a[3];
b[3] = ~a[1] & ~a[3];
b[4] = 1;
```

PLA的映射关系：

```
3'b10?
3'b??1
3'b0?0
3'b???
```

# Verilog HDL系统任务和函数
## --可编程逻辑阵列建模系统任务

PLA模型的模块描述：

```
module pla;
`define rows 4
`define cols 3
reg [1:`cols] a, mem[1:`rows];
reg [1:`rows] b;
initial begin
// PLA系统调用
        $async$and$plane(mem,a[1:3],b[1:4]);
        mem[1] = 3'b10?;
        mem[2] = 3'b??1;
        mem[3] = 3'b0?0;
        mem[4] = 3'b???;
```

# Verilog HDL系统任务和函数
## --可编程逻辑阵列建模系统任务

```
// 激励源和显示
        #10 a = 3'b111;
        #10 $displayb(a, " -> ", b);
        #10 a = 3'b000;
        #10 $displayb(a, " -> ", b);
        #10 a = 3'bxxx;
        #10 $displayb(a, " -> ", b);
        #10 a = 3'b101;
        #10 $displayb(a, " -> ", b);
end
endmodule
```

# Verilog HDL系统任务和函数
## --可编程逻辑阵列建模系统任务

**运行结果如下：**

111 -> 0101

000 -> 0011

xxx -> xxx1

101 -> 1101

# Verilog HDL系统任务和函数
# --随机分析任务

Verilog HDL提供了一个系统任务和函数集合，用于管理队列。这些任务便于实现随机队列模型。

# Verilog HDL系统任务和函数
# --随机分析任务

## $q_initialize

- 该系统任务创建一个新的队列。其语法格式为：

$q_initialize (q_id, q_type, max_length, status) ;

其中：

- q_id：是一个整数，用于标识一个新的队列。
- q_type：是一个整数输入。其值标识队列的类型。下表给出了$q_type值的类型。

# Verilog HDL系统任务和函数
## --随机分析任务

| q_type值 | 队列类型 |
| --- | --- |
| 1 | 先进先出 |
| 2 | 后进先出 |

- status：表示该操作成功或者错误状态。
- max_length：是整数输入，标识队列中允许入口的最大个数。

# Verilog HDL系统任务和函数
# --随机分析任务

## $q_add

- 该任务在队列添加入口。其语法格式为：

```
$q_add (q_id, job_id, inform_id, status) ;
```

其中：

- q_id：是一个整数，用于标识添加到入口的一个队列。
- job_id：是一个整数输入，标识工作。
- inform_id：是一个整数输入，和队列入口相关。它的含义由用户定义。比如：代表在一个CPU模型中，一个入口的执行时间。
- status：表示该操作成功或者错误状态。

# Verilog HDL系统任务和函数
# --随机分析任务

## $q_remove

- 该任务从一个队列中接收一个入口。其语法格式为：

$q_remove (q_id, job_id, inform_id, status) ;

其中：

- q_id：是一个整数，用于标识将移除的队列。
- job_id：是一个整数输入，标识正在移除的入口。
- inform_id：是一个整数输出，在$q_add时，由队列管理器保存它。它的含义由用户定义。
- status：表示该操作成功或者错误状态。

# Verilog HDL系统任务和函数
## --随机分析任务

### $q_full

该系统任务用于检查一个队列是否有空间用于其它入口，

其语法格式为：

```
$q_full (q_id, status) ;
```

其中：

- status：表示该操作成功或者错误状态。
- 当队列满时，返回1；否则返回0。

# Verilog HDL系统任务和函数
# --随机分析任务

## $q_exam

- 该系统任务提供队列q_id活动性的统计信息。其语法格式为：

  $q_exam (q_id, q_stat_code, q_stat_value, status) ;

- 根据q_stat_code所要求的信息，返回q_stat_value。下表给出了$q_exam系统任务的参数值。

# Verilog HDL系统任务和函数
## --随机分析任务

| q_stat_code所要求的值 | 从q_stat_value返回值的信息 |
|---|---|
| 1 | 当前队列长度 |
| 2 | 平均到达时间 |
| 3 | 最大队列长度 |
| 4 | 最短等待时间 |
| 5 | 用于队列内工作的最长等待时间 |
| 6 | 队列中的平均等待时间 |

# Verilog HDL系统任务和函数
# --随机分析任务

## 状态编码

- 所有的队列管理任务和函数返回一个输出状态码。下表给出了状态码的值。

| 状态码值 | 含义 |
|---|---|
| 0 | OK |
| 1 | 队列满，不能添加 |
| 2 | 未定义的q_id |
| 3 | 队列空，不能移除 |
| 4 | 不支持的队列类型，不能创建队列 |
| 5 | 指定的长度，小于等于0不能创建队列 |
| 6 | 重复的q_id，不能创建队列 |
| 7 | 没有足够的存储器，不能创建队列 |

# Verilog HDL系统任务和函数
## --随机分析任务

### 随机分析任务Verilog HDL描述的例子

```
always @(posedge clk)
begin
    // 检查队列是不是满
$q_full(queue1, status);
   //如果满，则显示信息和移除一个条目
if (status) begin
     $display("Queue is full”);
     $q_remove(queue1, 1, info, status);
end
```

# Verilog HDL系统任务和函数
## --随机分析任务

```
// 添加一个新的条目到队列queue1
$q_add(queue1, 1, info, status);
// 如果有错误，显示消息
if (status)
    $display("Error %d”,status);
  end
end
```

# Verilog HDL系统任务和函数
# --仿真时间函数

Veriog HDL提供系统函数，用于访问当前仿真时间。

$time

- 该系统函数，用于返回64位的整数。与调用该函数模块的时间尺度有关。

# Verilog HDL系统任务和函数
# --仿真时间函数

## $time的Verilog HDL描述的例子。

```
`timescale 10 ns / 1 ns
module test;
reg set;
parameter p = 1.55;
initial begin
        $monitor($time,,"set=",set);
        #p set = 0;
        #p set = 1;
end
endmodule
```

# Verilog HDL系统任务和函数
## --仿真时间函数

该例子的输出：

// 0 set=x

// 2 set=0

// 3 set=1

在该例子中，在仿真时间16ns时，给寄存器类型变量分配值0。在仿真时间32ns时，分配值1。由下面的步骤决定了$time系统函数返回的时间值：

# Verilog HDL系统任务和函数
## --仿真时间函数

- 仿真时间16ns和32ns，被标定到1.6和3.2，因为用于模块的时间单位是10ns。因此，这个模块报告的时间值是10ns的乘法。
- 因为$time系统函数返回一个整数，值1.6四舍五入到2，3.2四舍五入到3。时间精度不引起这些值的四舍五入。

# Verilog HDL系统任务和函数
# --仿真时间函数

## $stime

- 该系统函数，用于返回无符号的32位整数时间。与调用该函数模块的时间单位相关。

## $realtime

- 该系统函数，向调用它的模块返回实时仿真时间。

# Verilog HDL系统任务和函数
## --仿真时间函数

**$realtime的Verilog HDL描述的例子。**

```
`timescale 10 ns / 1 ns
module test;
reg set;
parameter p = 1.55;
initial begin
        $monitor($realtime,,"set=",set);
        #p set = 0;
        #p set = 1;
end
endmodule
```

```
//输出结果为:
// 0 set=x
// 1.6 set=0
// 3.2 set=1
```

# Verilog HDL系统任务和函数
# --转换函数

有时需要将整数转换成实数，或将实数转换成整数，或者用向量形式来表示实数等，Verilog HDL提供了下列数字类型转换的功能函数：

- $rtoi(real_value)
  - 通过截断小数值将实数转换为整数。
- $itor(integer_value)
  - 将整数转换为实数。

# Verilog HDL系统任务和函数 --转换函数

- $realtobits(real_value)
  - 将实数转换为64位的实数向量表示法（实数的IEEE 745表示法）。
- $bitstoreal(bit_value)
  - 将位模式转换为实数（与$realtobits相反）。

# Verilog HDL系统任务和函数
# --转换函数

## $realtobits和$bitstoreal的Verilog HDL描述的例子。

```
module driver (net_r);
output net_r;
real r;
wire [64:1] net_r = $realtobits(r);
endmodule

module receiver (net_r);
input net_r;
wire [64:1] net_r;
real r;
initial assign r = $bitstoreal(net_r);
endmodule
```

# Verilog HDL系统任务和函数
## --概率分布函数

Verilog HDL提供了系统函数，根据标准的概率函数，返回整数值。

# Verilog HDL系统任务和函数
## --概率分布函数

### $random函数

- 系统函数$random提供了生成随机数的机制。该函数返回一个后符号的32位随机数。其语法格式为：

$random[(seed)]

其中：

- seed

  种子变量。

# Verilog HDL系统任务和函数
## --概率分布函数

**注：**

**1. 种子变量（必须是寄存器、整数或时间寄存器类型）控制函数的返回值，即不同的种子将产生不同的随机数。**

**2. 如果没有指定种子，每次调用$random函数时，根据默认种子产生随机数。**

# Verilog HDL系统任务和函数
## --概率分布函数

### 概率分布函数的Verilog HDL描述1

```
integer Seed, Rnum;
wire Clk ;
initial seed = 12;
always@ (Clk) Rnum= $random (seed) ;
```

在Clk的每个边沿，调用$random，并返回一个32位有符号随机整数。

# Verilog HDL系统任务和函数
## --概率分布函数

### 概率分布函数的Verilog HDL描述2

```
Rnum = $random(seed)%11;
```

如果数字在取值范围内，上面的模运算符可产生 - 10 ~ +10之间的随机数。

# Verilog HDL系统任务和函数
# --概率分布函数

## 概率分布函数的Verilog HDL描述3

Rnum = $random /2;

注：

- 数字产生的顺序是伪随机排序的，即对于一个初始种子值产生相同的数字序列。
- 并置操作符（{ }）将$random函数返回的有符号整数变换为无符号数。

# Verilog HDL系统任务和函数
## --概率分布函数

### $dist_函数

根据在函数名中指定的概率函数，下列系统函数产生伪随机数。

- $dist_uniform (seed,start,end)
- $dist_normal(seed,mean,standard_deviation,upper)
- $dist_exponential(seed,mean)
- $dist_poisson(seed,mean)
- $dist_chi_square(seed,degree_of_freedom)
- $dist_t(seed,degree_of_freedom)
- $dist_erlang(seed,k_stage, mean)

# Verilog HDL系统任务和函数
## --概率分布函数

其中：

- 系统函数的所有参数都是整数值。
- 对于函数exponential、poisson、chi_square、t和erlang来说，参数mean、degree_of_freedom和k_stage的值应该大于0。
- 每个函数返回一个伪随机数，它们的分布特征由函数名字确定。
- 对于$dist_uniform函数来说，start和end参数是整数输入。start的值小于end的值。

# Verilog HDL系统任务和函数
## --概率分布函数

$dist_函数Verilog HDL描述的例子1。

```
reg [15:0] a ;
initial begin
        a = $dist_exponential(60, 24);
end
```

# Verilog HDL系统任务和函数
## --概率分布函数

$dist_函数Verilog HDL描述的例子2。

```
reg [15:0] a ;
initial begin
        a = $dist_erlang(60, 24, 7) ;
end
```

# Verilog HDL系统任务和函数
## --命令行输入

在仿真中，取代读取文件得到使用信息的另一种方法是用带有命令的指定信息来调用仿真器。

- 这个信息是提供给仿真的一个可选参数格式。
- 通过用一个“+”字符开始，这样这些参数就可以区别于其它仿真器参数。

# Verilog HDL系统任务和函数
## --命令行输入

### $test$plusargs (string)

- $test$plusargs系统函数为用户指定的plusarg_string查找plusargs列表。
- 这个字符串不包含命令行前面的“+”号。
- 如果匹配则返回非零的整数；否则，返回0。

# Verilog HDL系统任务和函数
## --命令行输入

### $test$plusargs 的Verilog HDL描述的例子

```
initial begin
    if ($test$plusargs("HELLO")) $display("Hello argument
       found.")
    if ($test$plusargs("HE")) $display("The HE subset string is
       detected.");
    if ($test$plusargs("H")) $display("Argument starting with H
        found.");
    if ($test$plusargs("HELLO_HERE"))$display("Long
       argument.");
    if ($test$plusargs("HI")) $display("Simple greeting.");
    if ($test$plusargs("LO")) $display("Does not match.");
end
```

# Verilog HDL系统任务和函数
## --命令行输入

用命令+HELLO运行仿真器。运行结果如下：

Hello argument found.

The HE subset string is detected.

Argument starting with H found.

# Verilog HDL系统任务和函数
## --命令行输入

### $value$plusargs (user_string, variable)

- $value$plusargs系统函数为用户定义的plusarg_string寻找plusargs。
- 系统函数内的第一个参数指定的字符串作为一个字符串或者一个非实数变量（将其理解为一个字符串），字符串不包含命令行参数前面的“+”号。

# Verilog HDL系统任务和函数
## --命令行输入

在命令行所提供的plusargs，按照所提供的顺序进行查找。如果提供的plusargs其中一个字头匹配所提供字符串的所有字符，则函数返回一个非零的整数，字符串的剩余部分转换为use_string内指定的类型，结果值保存在所提供的变量中。如果没有找到匹配的字符串，函数返回一个整数0，不修改所提供的变量。当函数返回0的时候，不产生警告信息。

# Verilog HDL系统任务和函数
## --命令行输入

user_string是下面的格式

“plusarg_stringformat_string”，格式化字符串和$display系统任务一样。下面是合法的格式：

- %d，十进制转换
- %o，八进制转换
- %h，十六进制转换
- %b，二进制转换

# Verilog HDL系统任务和函数
## --命令行输入

- %e，实数指数转换
- %f，实数十进制转换
- %g，实数十进制或者指数转换
- %s，字符串（没有转换）

# Verilog HDL系统任务和函数
## --命令行输入

来自plusargs列表的第一个字符串提供给仿真器，匹配user_string指定的plusarg_string部分，将是用于转换的可用的plusarg字符串。

进行匹配plusarg的剩余字符串将从一个字符串转换为格式字符串指定的格式，并保存在所提供的变量中。如果没有剩余的字符串，保存到变量的值为0或者空的字符串的值。

# Verilog HDL系统任务和函数
## --命令行输入

如果变量的宽度大于转换后的值，则在保存的值前面补零。如果变量不能保留转换后的值，则把值截断。如果值是负数，值被认为大于所提供的变量。如果在字符串中用于转换的字符是非法的，则将变量的值设置为‘bx。

# Verilog HDL系统任务和函数
## --命令行输入

### $value$plusargs的verilog HDL描述的例子

```
`define STRING reg [1024 * 8:1]
module goodtasks;
    `STRING str;
     integer int;
     reg [31:0] vect;
     real realvar;
     initial
         begin
             if ($value$plusargs("TEST=%d",int))
                 $display("value was %d",int);
```

# Verilog HDL系统任务和函数
## --命令行输入

```
    else
        $display("+TEST= not found");
    #100 $finish;
  end
endmodule

module ieee1364_example;
    real frequency;
    reg [8*32:1] testname;
    reg [64*8:1] pstring;
    reg clk;
    initial
```

# Verilog HDL系统任务和函数
## --命令行输入

```
begin
    if ($value$plusargs("TESTNAME=%s",testname))
        begin
            $display(" TESTNAME= %s.",testname);
            $finish;
        end
if (!($value$plusargs("FREQ+%0F",frequency)))
            frequency = 8.33333; // 166 MHz
            $display("frequency = %f",frequency);
            pstring = "TEST%d";
if ($value$plusargs(pstring, testname))
            $display("Running test number %0d.",testname);
 end
endmodule
```

# Verilog HDL系统任务和函数
## --命令行输入

- 添加到工具的命令行plusarg

+TEST=5

输出结果是：

value was 5

frequency = 8.333330

Running text number x.

# Verilog HDL系统任务和函数
## --命令行输入

- 添加到工具的命令行plusarg

+TESTNAME=bar

输出结果是：

+TEST= not found

TESTNAME=bar.

# Verilog HDL系统任务和函数
## --命令行输入

- 添加到工具的命令行plusarg

+FREQ+9.234

出结果是：

+TEST= not found

frequency = 9.234000

# Verilog HDL系统任务和函数
## --命令行输入

- 添加到工具的命令行plusarg

+TEST23

输出结果是：

+TEST= not found

frequency = 8.333330

Running test number 23.

# Verilog HDL系统任务和函数
# --数学函数

Verilog HDL提供了整数和实数函数。数学系统函数可以用在常数表达式中。

# Verilog HDL系统任务和函数
# --数学函数

## 整数数学函数

## 整数数学函数Verilog HDL描述的例子

```
integer result;
result = $clog2(n);
```

系统函数$clog2将返回基于2的对数的计算结果。参数可以是一个整数或者是一个任意宽度的向量值。将参数看作是无符号的数。如果参数值为0，则产生的结果也为0。

# Verilog HDL系统任务和函数
# --数学函数

## 实数数学函数

下表给出了Verilog到C实数数学函数的交叉列表。这些函数接受实数参数，返回实数结果。这些行为匹配等效的C语言标准数学库函数。

# Verilog HDL系统任务和函数
## --数学函数

| Verilog函数 | 等效的C函数 | 描述 |
|---|---|---|
| $ln(x) | log(x) | 自然对数 |
| $log10(x) | log10(x) | 基10对数 |
| $exp(x) | exp(x) | 指数函数 |
| $sqrt(x) | sqrt(x) | 平方根 |
| $pow(x,y) | pow(x,y) | x的y次幂 |
| $floor(x) | floor(x) | 向下舍入 |
| $ceil(x) | ceil(x) | 向上舍入 |
| $sin(x) | sin(x) | 正弦 |

# Verilog HDL系统任务和函数
## --数学函数

| | | |
|---|---|---|
| $cos(x) | cos(x) | 余弦 |
| $tan(x) | tan(x) | 正切 |
| $asin(x) | asin(x) | 反正弦 |
| $acos(x) | acos(x) | 反余弦 |
| $atan(x) | atan(x) | 反正切 |
| $atan2(x,y) | atan2(x,y) | (x/y)反正切 |
| $hypot(x,y) | hypot(x,y) | (x*x+y*y)的平方根 |
| $sinh(x) | sinh(x) | 双曲正弦 |

# Verilog HDL系统任务和函数
## --数学函数

| | | |
|---|---|---|
| $cosh(x) | cosh(x) | 双曲余弦 |
| $tanh(x) | tanh(x) | 双曲正切 |
| $asinh(x) | asinh(x) | 反双曲正弦 |
| $acosh(x) | acosh(x) | 反双曲余弦 |
| $atanh(x) | atanh(x) | 反双曲正切 |