数字集成电路静态时序分析基础

邸志雄 博士

西南交通大学信息科学与技术学院

Part-1:TCL语言

第一部分 概述

TCL简介

功能	工具	语言
数字综合	DC/Genus	TCL
DFT	Tessent/Dftmax	TCL
布局布线	Innovus/ICC2	TCL
静态时序仿真	PT/Tempus	TCL
功耗分析	Redhawk/Voltus	TCL
版图工具	Calibredrv	TCL
物理验证	Calibre/PVS	SVRF/TCL
形式验证	LEC/Formality	TCL

TCL、Perl、Python、csh语言的异同

	CSH	TCL	PERL	PYTHON
处理简单问题	****	*	**	*
EDA兼容性	*	****	*	*
EDA流程控制	*	****	*	*
语言功能扩展和高级用法(IC设计方向)	*	**	****	*
运算性能	*	*	*	****

如何启动TCL

- Linux 系统下
  - 输入 tclsh
  - 在文本第一行 “#! /user/bin/tclsh” （TCL的安装路径）

- Windows系统下

  安装active tcl并双击wish.exe

置换

TCL 解释器运用规则把命令分成一个个独立的单词，同时进行必要的置换。

TCL置换分为以下三类。

- 变量置换 $
- 命令置换 []
- 反斜杠置换 \

变量置换

- 用$表示变量置换

  TCL解释器会将认为$后面为变量名，将变量置换成它的值。

```
(Tcl) 84 % set a "snow"
snow
(Tcl) 85 % puts $a
snow
(Tcl) 86 % puts a
a
```

命令置换

- 用[]表示命令置换
  []内是一个独立的TCL语句

```
(Tcl) 94 % set a [expr 3 + 4]
7
(Tcl) 95 % puts $a
7
```

反斜杠置换

- 用\表示反斜杠置换

换行符、空格、[、$等被TCL解释器当作特殊符号对待的字符，加上反斜杠后变成普通字符。

```
(System32) 10 % puts "[expr $X + $Y]"
2.5
(System32) 11 % puts "\[expr $X + $Y\]"
[expr 1.2 + 1.3]
(System32) 12 % puts "\[expr \$X + \$Y\]"
[expr $X + $Y]
```

反斜杠置换

- 用\t表示TAB。
- 用\n表示换行符。

```
(Tcl) 96 % puts "a\tb"
a       b
(Tcl) 97 % puts "a\nb"
a
b
```

其他符号

- “” TCL解释器对双引号中 $ 和[]符号会进行变量置换和命令置换。

```
(System32) 3 % puts "\t[expr $X + $Y]"
        2.5
```

- {} 而在花括号中，所有特殊字符都将成为普通字符，TCL解释器不会对其作特殊处理。

```
(System32) 4 % puts {\t[expr $X + $Y]}
\t[expr $X + $Y]
```

- # 表示注释。

```
(System32) 5 % #[expr $X + $Y]
(System32) 6 %
```

第二部分 变量、数组、列表

变量

- 变量就是某个容器的名称，可以存储一个值。变量的名称在程序运行期间保持不变，但是变量的值通常会不断改变。
- 定义：set 变量名 变量值
- 取值：$变量名

```
(Windows) 11 % set cell "bufx2"
bufx2
(Windows) 12 % puts $cell
bufx2
(Windows) 13 % set cell "ivtx2"
ivtx2
(Windows) 14 % puts $cell
ivtx2
(Windows) 15 %
```

变量

例题：假设我们想打印变量varible，后面跟一个"_1"，会发生什么呢?

```
(Windows) 4 % set a 2
2
(Windows) 5 % puts $a_1
can't read "a_1": no such variable
(Windows) 6 %
```

```
(System32) 1 % set a  2
2
(System32) 2 % puts ${a}_1
2_1
(System32) 3 %
```

数组

- 数组：TCL中数组可以存储很多值，通过元素名来进行检索。类似于某件事物（数组名）几种不同属性（元素名），每一种属性有其独立的值。
- 定义：set 数组名（元素名） 值

```
(System32) 5 % set cell_1(ref_name) "bufx2"
bufx2
(System32) 6 % set cell_1(full_name) "top/cell_1"
top/cell_1
(System32) 7 % set cell_1(pins) "A B C"
A B C
```

- 取值：$数组名（元素名）

```
(System32) 8 % puts $cell_1(ref_name)
bufx2
```

数组

使用array指令获取数组信息

```
(System32) 9 % array size cell_1
3
(System32) 10 % array names cell_1
ref_name pins full_name
```

列表

- 列表是标量的有序集合。
- 定义 set 列表名 {元素1 元素2 元素3……}
- 取值 $列表名

```
(System32) 12 % set ivt_list {ivtx2 ivtx3 ivtx8}
ivtx2 ivtx3 ivtx8
(System32) 13 % puts $ivt_list
ivtx2 ivtx3 ivtx8
```

列表

TCL中有一系列十分方便的列表操作命令

命令	功能
concat	合并两个列表
lindex	选取列表中的某个元素
llength	列表长度
lappend	在列表末端追加元素
lsort	列表排序

列表指令-concat

- 语法格式： concat 列表1 列表2
- 功能: 将列表1和列表2合并

```
(Windows) 1 % set list1 {bufx1 bufx2 bufx4}
bufx1 bufx2 bufx4
(Windows) 2 % set list2 {ivtx1 ivtx2 ivtx4}
ivtx1 ivtx2 ivtx4
(Windows) 3 % concat $list1 $list2
bufx1 bufx2 bufx4 ivtx1 ivtx2 ivtx4
(Windows) 4 %
```

实例讲解

例题：　　concat 后面接三个list会怎么样呢？

```
(Windows) 1 % set a {1 2}
1 2
(Windows) 2 % set b {3 4}
3 4
(Windows) 3 % set c {5 6}
5 6
(Windows) 4 % concat $a $b $c
1 2 3 4 5 6
```

列表指令-llength

- 语法格式： llength 列表
- 功能: 返回列表中的元素个数

```
(Windows) 9 % set list1 {bufx1 bufx2 bufx4}
bufx1 bufx2 bufx4
(Windows) 10 % llength $list1
3
(Windows) 11 %
```

实例讲解

例题： list1为{bufx1 bufx2 bufx4}，那么 llength [concat $list1 $list1] 会得到多少呢?

```
(Windows) 6 % set list1 {bufx1 bufx2 bufx4}
bufx1 bufx2 bufx4
(Windows) 7 % llength [concat $list1 $list1]
6
(Windows) 8 %
```

列表指令-lindex

- 语法格式：　　lindex 列表 n
- 功能:　　返回列表中第n个元素（从0开始计数）

```
(Windows) 4 % set list1 {bufx1 bufx2 bufx4}
bufx1 bufx2 bufx4
(Windows) 5 % lindex $list1 1
bufx2
(Windows) 6 %
```

实例讲解

例题：　　如何得到列表list1 {a b c d e f}的最后一个元素？

```
(System32) 11 % set list1 {a b c d e f}
a b c d e f
(System32) 12 % llength $list1
6
(System32) 13 % expr [llength $list1] - 1
5
(System32) 14 % lindex $list1 [expr [llength $list1] - 1]
f
(System32) 15 %
```

列表指令-lappend

- 语法格式： lappend 列表 新元素
- 功能：　　列表末尾加入新元素

```
(Windows) 8 % set a {1 2 3}
1 2 3
(Windows) 9 % lappend a 4
1 2 3 4
(Windows) 10 % puts $a
1 2 3 4
(Windows) 11 %
```

实例讲解

例题：　　如果我们lappend一个列表会怎么样？

```
(Windows) 11 % set a {1 2 3}
1 2 3
(Windows) 12 % set b {4 5}
4 5
(Windows) 13 % lappend a $b
1 2 3 {4 5}
```

例题：　　如果我们想得到4 用什么样的命令？

```
(Windows) 14 % lindex [lappend a $b] 3
4 5
(Windows) 15 % lindex [lindex [lappend a $b] 3] 0
4
(Windows) 16 %
```

列表指令-lsort

- 语法格式：　　lsort 开关 列表
- 功能：　　将列表按照一定规则排序
- 开关：　　缺省时默认按照ASCII码进行排序。
  - –real　　按照浮点数值大小排序
  - -unique唯一化，删除重复元素

列表指令-lsort

- 按照ASICC码排序

```
(Windows) 16 % set list1 {c d a f b}
c d a f b
(Windows) 17 % lsort $list1
a b c d f
```

- 按照数字大小排序

```
(Windows) 18 % set list2 {-2 3.1 5 0}
-2 3.1 5 0
(Windows) 19 % lsort -real $list2
-2 0 3.1 5
```

- 唯一化

```
(Windows) 20 % set list3 {a c c b a d}
a c c b a d
(Windows) 21 % lsort -unique $list3
a b c d
```

实例讲解

例题：如何得到列表list1 {0 1.2 -4 3 5}中的最小值?

```
(System32) 15 % set list1 {0 1.2 -4 3 5}
0 1.2 -4 3 5
(System32) 16 % lsort -real $list1
-4 0 1.2 3 5
(System32) 17 % lindex [lsort -real $list1] 0
-4
```

运算

- 数学运算

```
a + b
a – b
a * b
a / b
```

- 逻辑运算

```
a <= b
a >= b
a  == b
a != b
```

数学运算指令-expr

- 语法格式：　　expr 运算表达式
- 功能：　　将运算表达式求值

```
(Windows) 24 % expr 6 + 4
10
(Windows) 25 % expr 6 - 4
2
(Windows) 26 %
```

实例讲解

例题：我们在TCL中经常会遇到下面的现象

```
(Windows) 26 % expr 5/2
2
```

其原因是表达式5/2中5和2都是整数型参数，默认运算结果也是整数型。
如果想要进行浮点运算，只要将其中任意一个数值，写成浮点形式（有小数点）即可

```
(Windows) 27 % expr 5/2.0
2.5
(Windows) 28 % expr 5.0/2
2.5
```

控制流-if

- 语法格式：

```
if {判断条件} {
脚本语句
} elseif {判断条件} {
脚本语句
} else {
脚本语句
}
```

```
(windows) 31 %  if {$a > $b} {
> puts $a
> } else {
> puts $b
> }
3
```

判断语句

脚本语句

- 注意，上例中脚本语句的'{'一定要写在上一行，因为如果不这样，TCL 解释器会认为if命令在换行符处已结束，下一行会被当成新的命令，从而导致错误

实例讲解

例题：我们如何判断一个列表{0 1 2 3 4}的长度是大于3，还是等3，还是小于3?

```
(System32) 24 % set list1 {0 1 2 3 4}
0 1 2 3 4
(System32) 25 % set length [llength $list1]
5
(System32) 26 % if {$length > 3} {
puts "The length of list1 is larger than 3"
} elseif {$length == 3} {
puts "The length of list1 is equal to 3"
} else {
puts "The length of list1 is less than 3"
}
The length of list1 is larger than 3
(System32) 27 %
```

第三部分 控制流

循环指令-foreach

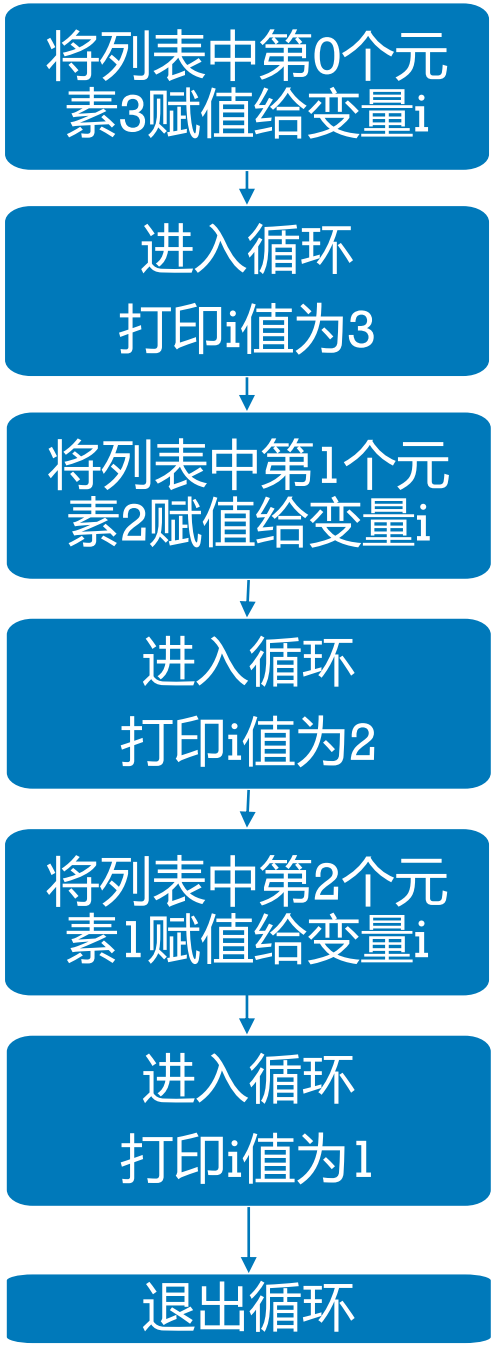
- 语法格式： foreach 变量 列表 循环主体
- 功能：从第0个元素开始，每次按顺序取得列表的一个元素，将其赋值给变量，然后执行循环主体一次，直到列表最后一个元素

```
(Windows) 36 % set list1 {3 2 1}
3 2 1
(Windows) 37 % foreach i $list1 {
> puts $i
> }
3
2
1
```

变量

列表

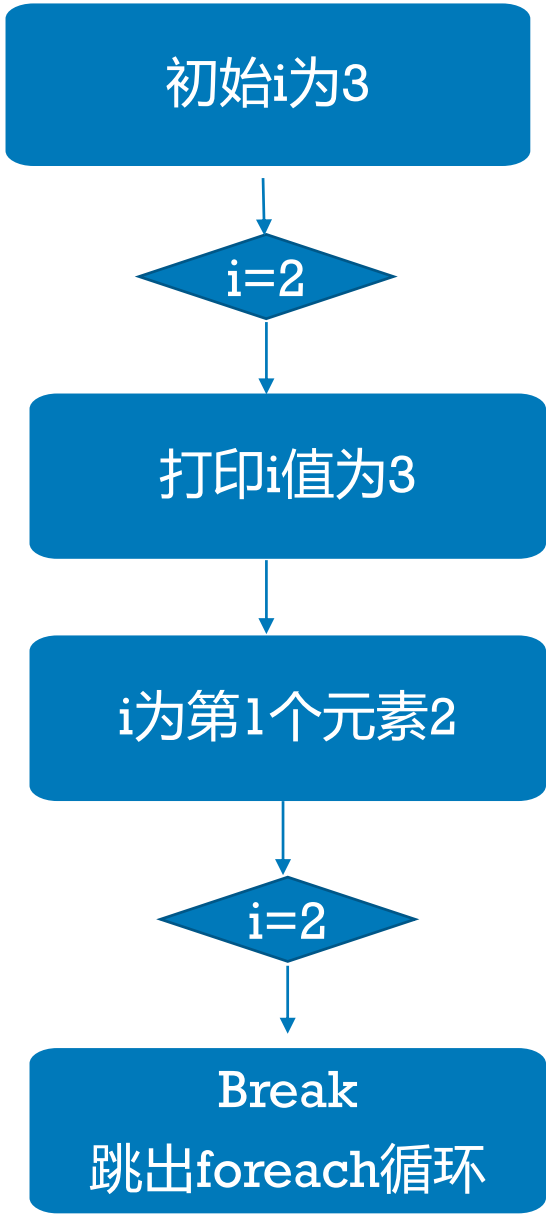
循环主体



循环控制指令-break

- 语法格式： break
- 功能:　　结束整个循环过程，并从循环中跳出

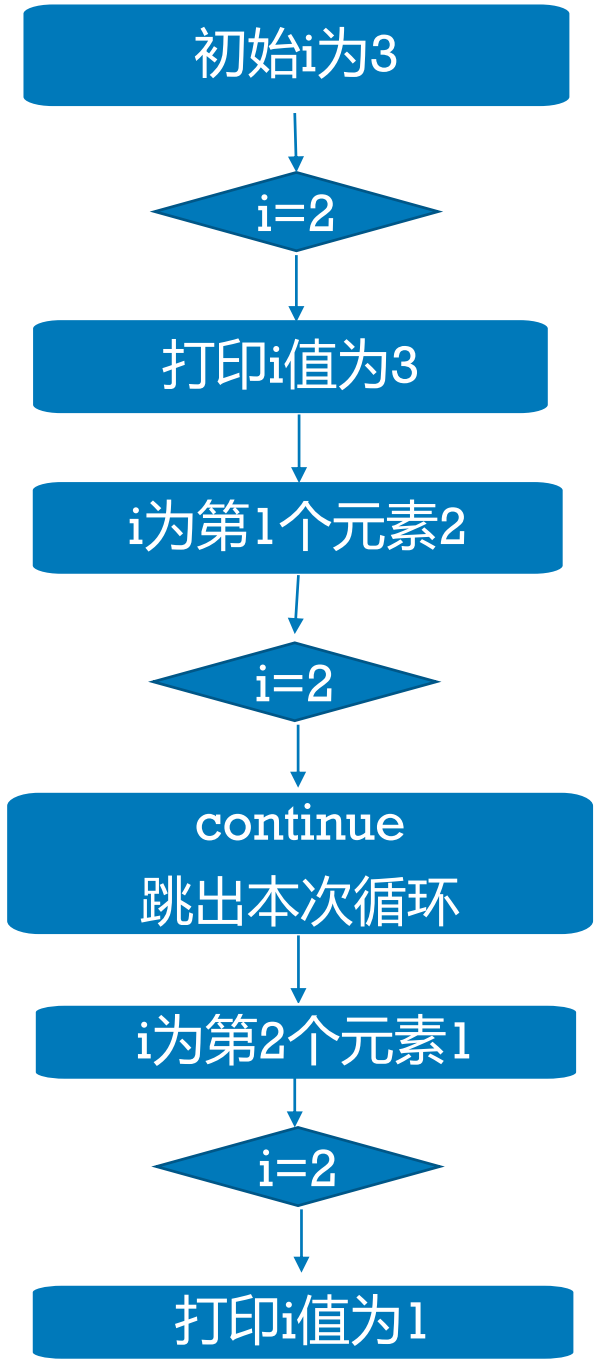
```
(Windows) 44 % set list1 {3 2 1}
3 2 1
(Windows) 45 % foreach i $list1 {
if {$i == 2} {
break
}
puts $i
}
3
```



循环控制指令-continue

- 语法格式： continue
- 功能:　　仅结束本次循环

```
(Windows) 46 % set list1 {3 2 1}
3 2 1
(Windows) 47 % foreach i $list1 {
if {$i == 2} {
continue
}
puts $i
}
3
1
```



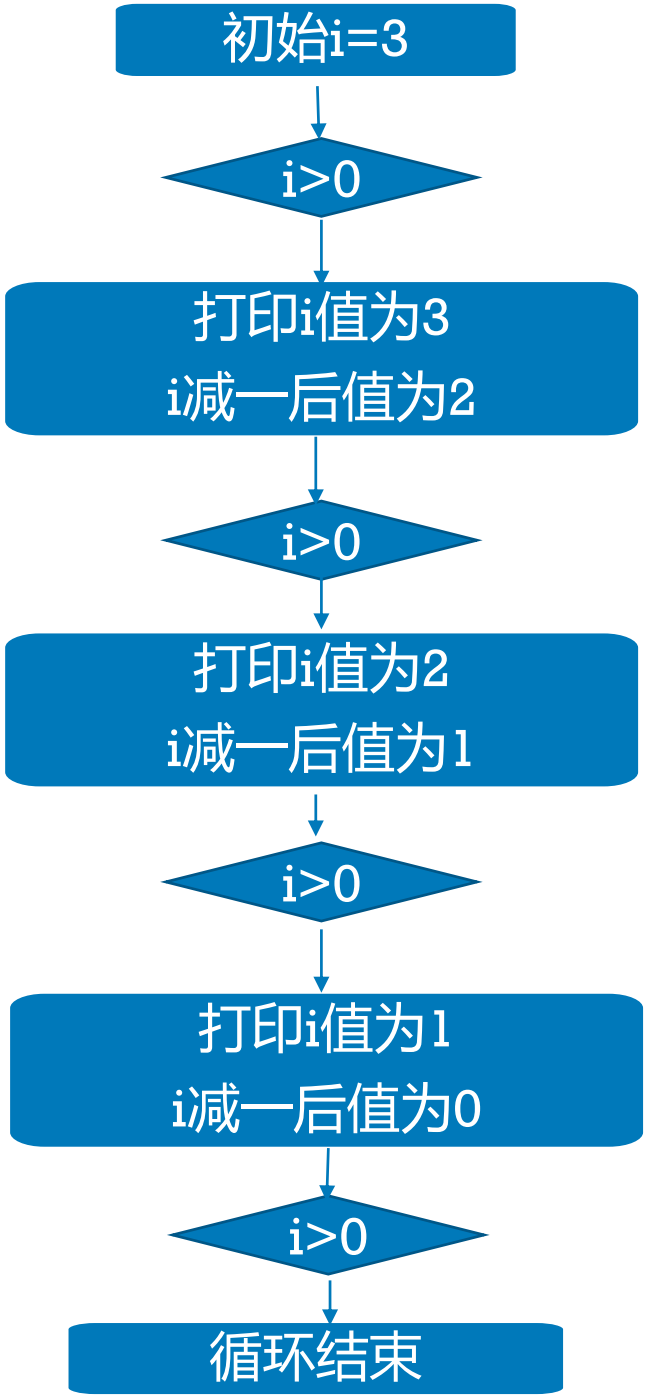
循环控制指令-while

- 语法格式： while 判断语句 循环主体
- 功能: 如果判断语句成立（返回值非0），就运行脚本，直到不满足判断条件停止循环，此时while命令中断并返回一个空字符串。

```
(Windows) 33 % set i 3
3
(Windows) 34 % while {$i > 0} {
> puts $i
> incr i -1; #set i [expr $i -1]
> }
3
2
1
```

判断语句

循环主体



循环控制指令-for

- 语法格式：

  for 参数初始化 判断语句 重新初始化参数 循环主体

- 功能:　　如果判断语句返回值非0就进入循环，执行循环主体后，再重新初始化参数。然后再次进行判断，直到判断语句返回值为0，循环结束。

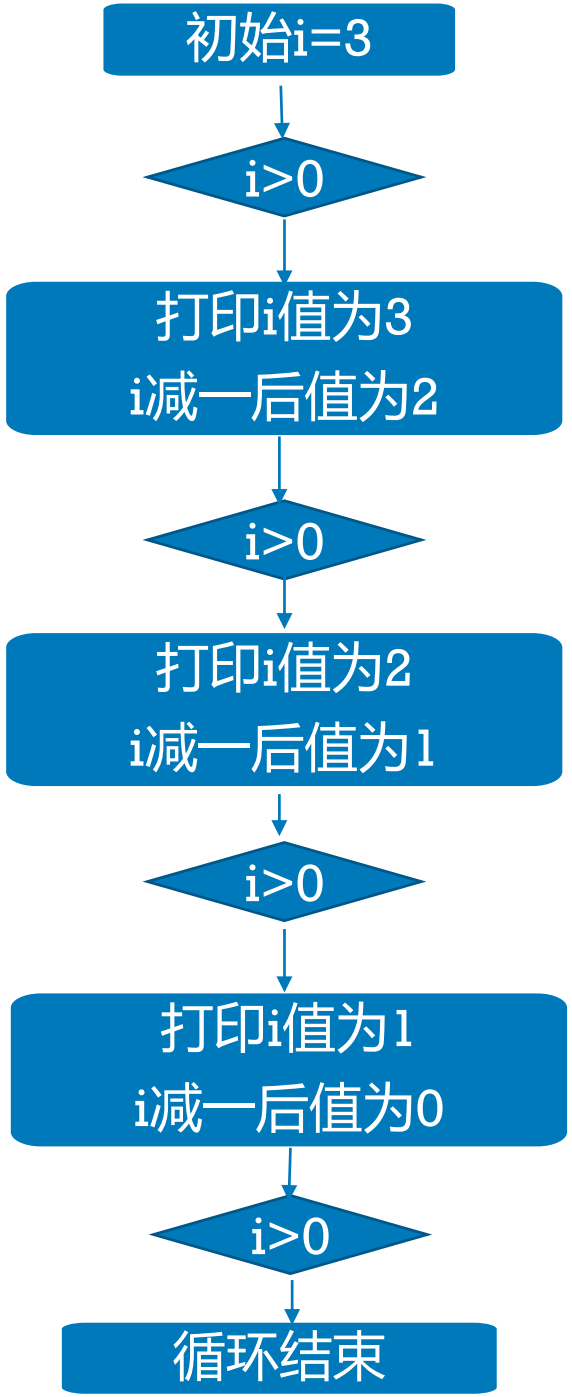
```
(Windows) 35 % for {set i 3} {$i >0} {incr i -1} {
puts $i
}
3
2
1
```

循环主体

初始化参数

判断语句

重新初始化参数



参考书目

- Using Tcl with Synopsys Tools. Version B-2008.09, March 2011. Synopsys.
- 集成电路静态时序分析与建模. 刘峰, 机械工业出版社.出版时间：2016-07-01.

谢谢聆听！