Tcl Scripting in Vivado

**1 Introduction**

The Vivado®IDE uses Xilinx Design Constraints (XDC) to specify the design constraints. XDC is

based on a subset of all the Tcl commands available in Vivado and is interpreted exactly like Tcl.

The Vivado tools write a journal file called vivado.jou into the directory from which Vivado

was launched. The journal is a record of the Tcl commands run during the session that can be used as a starting point to create new Tcl scripts.

Tk是Tcl的图形界面工具集。对应于wish（windowing shell）运用程序。

**2 A Brief Overview of Tcl**

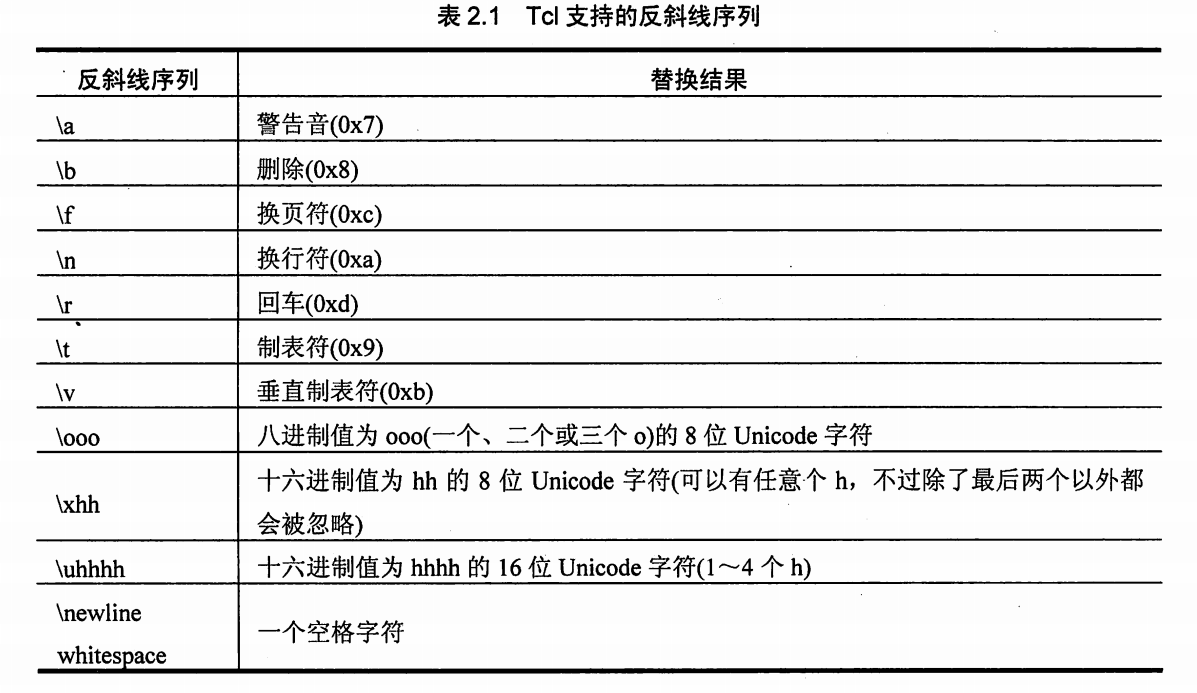
A Tcl script is a series of Tcl commands, separated by new-lines or semicolons. A Tcl command is a string of words, separated by blanks or tabs.

A word is a string that can be a single word, or multiple words within braces, {}（其中字符完全按照字面意义）, or multiple words within quotation marks, ””（其中字符或单词按照tcl语言关键字意义）.the backslash, \, is treated as a special character even within braces and quotation marks.

Commands can also be nested inside other commands within brackets, [], which are evaluated from left to right in a bottom-up manner.

[] Square brackets are treated as standard characters in Verilog and VHDL names (net, instances, etc.), and usually identify one or more elements of vectors, such as buses or arrays of instances.

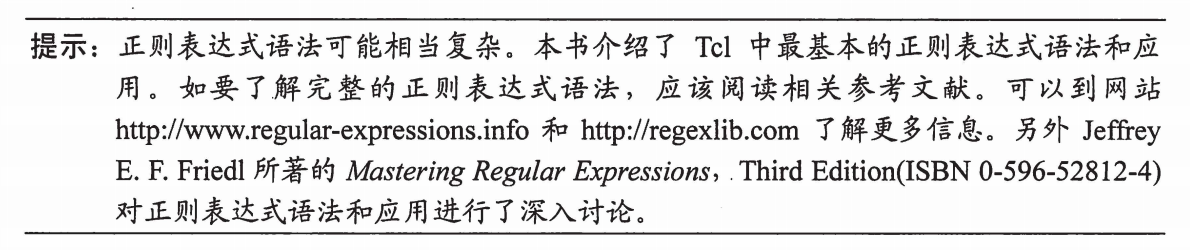
\替换。



\n 表示换行符。在双引号”字符串”中出现’\n’字符时，若输出该字符串将被分为两行。

\空格 表示空格字符。

\换行 表示空格字符，一个’\’紧接一个’换行’，中间没有空格。’\换行’字符通常用于命令行的分行书写，避免一行太长。在双引号”字符串”中出现’\换行’字符时，若输出该字符串，’\换行’字符将被替换成一个空格，且输出为一行，而不是两行。



glob \* 命令相当于ls ./ 意思是将通配符‘\*’匹配的字符串展开。

encoding system -- 列出系统支持的字符编码，比如unicode

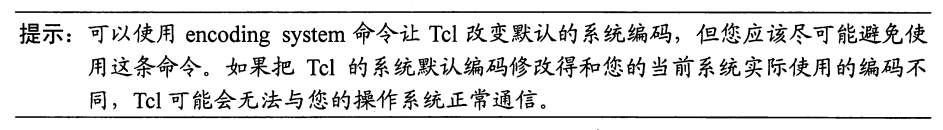
encoding convertfrom gb2312 “字符串” -- 将gb2312字符串转换为utf-8 unicode

encoding convertto gb2312 “str” -- 将utf-8字符串转换为gb2312

source -encoding gb2312 “script.tcl” 以gb2312编码方式执行script.tcl

set fd [open $file r]

Fconfigue $fd -enconding gb2312 调整文件的编码方式



**3 Getting Help**

You can get help directly from the Tcl console. Every Vivado command supports the -help

command line argument that can be used anywhere in the line.

In addition, there is a help command that provides additional information. Providing a command name to the help command (that is, help <command>) reports the same help information as <command> -help.

The help command can also just return a short description of the arguments using the -args

option: help <command> -arg

A short summary of the syntax of a command is also available with the -syntax option:

help <command> -syntax

A list of categories can be obtained by executing the “help” command without any argument or option. The categories is:

ChipScope

DRC

FileIO

Floorplan

GUIControl

IPFlow

Object

PinPlanning

Power

Project

PropertyAndParameter

Report

SDC

Simulation

TclBuiltIn

Timing

ToolLaunch

Tools

XDC

The list of commands available under each category can be also reported with the -category

option. For example, reporting all the commands under the Tools category:

help -category tools

数组是一组变量的集合。由，数组变量名（元素名） — 值，表达。$数组变量（$元素变量）可求出值。

set {capital(R china)} beijing #设置数组变量capital，元素R china 的值为beijing

>beijing

set capital(R\_china) beijing #设置数组变量capital，元素R\_china 的值为beijing

>beijing

上述capital(R china) 与 capital(R\_china)相当于两个变量。由于前一个包含空格，所以要用{}或””包含。

puts "$capital(R china)" #字符串中的变量需要替换

>beijing

Puts ${capital(R china)} #变量范围由{}界定

>beijing

puts {$capital(R china)} #字符串原样输出不替换

>$capital(R china)

注意{}或””的用法区别。

set nation "R china" #创建变量nation，其值是R china，是一个元素名

>R china

puts $capital($nation)

>beijing

set nati R\_china #创建变量nati，其值是R\_china，是一个元素名

>R\_china

puts $capital($nati)

>Beijing

puts $capital($R\_china)

can't read "R\_china": no such variable #元素变量与元素名不是一回事！

puts $capital(${R china})

can't read "R china": no such variable #同上，看来数组比后面的列表，字典麻烦

set matrix(1,1) 140 #多维（二维）数组

set matrix(1,2) 218

set matrix(1,3) 84

set i 1

set j 2

set cell $matrix($i,$j)

> 218

列表list {字符串0 字符串1，…… } 是字符串组。其索引从0开始的自然数。

字典dict {key0 value0 key1 value1 ……} 是key-value字符串对的组合，还是字符串集合。字符串value的索引是key，不是自然数。字典是具有偶数个元素的列表。

字典和数组有根本的不同，数组是变量的无序集合，其元素是变量；而字典是一组具有索引名称的值的有序集合。数组通常用于动态传递具备同类属性的数值；字典通常用于静态保存具备同类属性的数值。

set example {first joe sub schmoe title mr}

>first joe sub schmoe title mr

lindex $example 1

>joe

dict get $example sub

>schmoe

**4 Platform Specific Tcl Behaviors**

win32, win64, lnx32: sizeof(int) is 4bytes

lnx64: sizeof(int) is 8bytes

**5 Compilation and Reporting Example Scripts**

**5.1 Compilation with a Non-Project Flow**

**5.2 Compilation with a Project Flow**

**6 Loading and Running Tcl Scripts**

**6.1 Initializing Tcl Scripts**

The Vivado Design Suite can automatically load Tcl scripts defined in **Vivado\_init.tcl** file.

When you start the Vivado tools, it looks for a Tcl initialization script in several locations with the following precedence:

1. In the software installation: <installdir>/Vivado/<VivadoVersion>/scripts/Vivado\_init.tcl

Where <installdir> is the installation directory where the Vivado Design Suite is installed.

2. In the local user directory (Vivado tools version dependent):

• For Windows 7: %APPDATA%/Xilinx/Vivado/<VivadoVersion>/Vivado\_init.tcl

For example: %APPDATA%/Xilinx/Vivado/2017.1/Vivado\_init.tcl

• For Linux: $HOME/.Xilinx/Vivado/<VivadoVersion>/Vivado\_init.tcl

For example: $HOME/.Xilinx/Vivado/2017.1/Vivado\_init.tcl

3. In the local user directory (Vivado tools version independent):

• For Windows 7: %APPDATA%/Xilinx/Vivado/Vivado\_init.tcl

• For Linux: $HOME/.Xilinx/Vivado/Vivado\_init.tcl

If Vivado\_init.tcl exists in several locations, the Vivado tool sources the file following the

precedence explained above.The Vivado\_init.tcl file in the home directory allows each user to specify additional commands, or to override commands from the software installation to meet their specific design requirements.

**6.1 Sourcing Tcl Scripts**

source *<filename>*

Where <filename> specifies both the name of the file, as well as the relative or absolute path to the file.

Within the Vivado IDE you can also source a Tcl script from the Tools → Run Tcl Script menu command.

source <filename> -notrace

No echos.

**6.2 Executing a Tcl Script at Startup**

vivado -source myscript.tcl

source myscript.tcl

A checkpoint can also be specified on the command line along with a Tcl script:

vivado design.dcp -source myscript.tcl

open\_checkpoint design.dcp

source myscript.tcl

**6.3 Using Tcl Scripts in a Constraints Set**

Tcl scripts can be added to project constraint sets like any regular XDC file, except that the XDC files are managed by the tool, and not Tcl scripts.

**6.4 XDC Constraints: read\_xdc versus source**

When applying constraints to the design, the commands read\_xdc and source differ in

behavior. For more information, see the Vivado Design Suite User Guide: Using Constraints (UG903).

**6.5 Defining Tcl Hook Scripts**

In a Non-Project flow you have the ability to source a Tcl script at any point in the flow, such as before or after running the synth\_design command. You can also do this in a project-based flow, using the Vivado IDE, or by using the set\_property command to set a property on either a synthesis or implementation run. Tcl hook scripts allow you to run custom Tcl scripts prior to (tcl.pre) and after (tcl.post) synthesis and implementation design runs, or any of the implementation steps.

Right-click a run in the Design Runs window and select the Change Run Settings from the pop-up menu to open the Design Run Settings dialog box. The tcl.pre and tcl.post options can be used to specify a Tcl hook script.

**6.6 Sharing Hook Scripts Between Steps**

**6.7 Customizing the GUI**

**7 Writing a Tcl Script**

**7.1 Defining Tcl Procedures**

You can write Tcl scripts that can be loaded and run from the Vivado IDE, or you can write procedures, to act like new Tcl commands, taking arguments, checking for errors, and returning results.

A Tcl procedure is defined with the proc command which takes three arguments: the procedure name, the list of arguments, and the body of code to be executed. The following code provides a simple example of a procedure definition:

proc helloProc { arg1 } {

# This is a comment inside the body of the procedure

puts "Hello World! Arg1 is $arg1"

}

A procedure usually has predefined arguments. Each of them can optionally have a default value. When an argument has a default value, it does not need to be specified when calling the procedure if all the mandatory preceding arguments are specified. A procedure returns an empty string unless the return command is used to return a different value.

proc reportWorstViolations { nbrPaths corner delayType } {

report\_timing -max\_paths $nbrPaths -corner $corner -delay\_type $delayType

-nworst 1

}

proc reportWorstViolations { nbrPaths { corner Slow } { delayType Max } } {

report\_timing -max\_paths $nbrPaths -corner $corner -delay\_type $delayType

-nworst 1

}

The args keyword is a Tcl list that can have any number of elements, including none.

proc reportWorstViolations { nbrPaths args } {

eval report\_timing -max\_paths $nbrPaths $args （$args represent any arguments legally）

}

%> reportWorstViolations 2

%> reportWorstViolations 1 -to [get\_ports]

%> reportWorstViolations 10 -delay\_type min\_max -nworst 2

上述命令（过程）均能正常执行。

注意eval命令，eval是动态运行命令：

在shell中启动 tclsh

% puts abc

% eval puts abc

两条执行的结果是一样的，体会不到eval的好处，因为我们写死了这条命令"puts abc"

如果想执行多条命令，命令是变化的，在 \*.tcl 脚本中可以定义字符串变量，变量对应不同的命令，每次最后调用：

eval $cmd

就可以运行不同的命令，动态命令就是变化的命令。

此外eval命令能扫描后续参数，并将其分解为独立单词，可解决一些语法问题，参见Tcl\_Tk入门经典(第2版)第46页，参数展开章节。

**7.2、Parsing Command Line Arguments**

A single procedure that can handle multiple contexts is easier to use and maintain that multiple procedures that cover the same range of contexts with duplicated code. Tcl provides an easy way to do this through the args variable. The keyword args used inside the list of arguments of a procedure can match any number of elements, including none. The args variable is a regular Tcl list that can be processed and analyzed like any Tcl list.

通常有三种参数列表的分析方案。

**7.3、Local and Global Variables**

A local variable is a variable created inside a procedure. It is created at runtime inside the stack of the function. The variable is only accessible within the procedure and the variable name is not subject to name collision with variable names outside of the procedure.

A global variable is a variable created outside of a procedure and that belongs to the global

namespace. To refer to a global variable inside a procedure, the keyword global is used

followed by the variable name