



Tcl

The Good, The Bad, and The Ugly

Gregg D. Lahti
Steve J. Brown
Intel Corporation



Introduction

- Tcl language created by Dr. John Ousterhout in 1987
- Tcl is rapidly becoming a widely used, EDA industry language for tool control
- Tcl language is simple to use, offers powerful programming features with lightweight implementation overhead
- This paper is the result of the authors' experiences in using Tcl to replace DCSH scripts for synthesis
 - **The Good:** features of using Tcl for synthesis
 - **The Bad:** quirks that trap experienced programmers
 - **The Ugly:** Synopsys-specific implementations of Tcl that cause Tcl torture
 - What we'd like to see for improvements
 - Tcl References

Gregg D. Lahti & Steve J. Brown



The Good:

Tcl Makes Synthesis Easier

The Good

- Tcl adds the missing elements for a true programming environment
 - Procedures
 - Documenting user-defined procedures
 - Variables
 - Hashes and associative arrays
 - Regular expressions
 - Time and I/O functionality
 - Sockets



Procedures

The Good

- Allow grouping and modularization of code
- Can pass items to procedures and return values from procedures

Procedure

```
proc Pass {{arg1 foo} {arg2 bar} args} {  
  puts "arg1=$arg1, arg2=$arg2,  
  args=$args"  
}
```

Procedure results

```
%Pass  
arg1=foo, arg2=bar, args=  
%Pass the  
arg1=the, arg2=bar, args=  
%Pass the good bad ugly  
arg1= the, arg2= good, args= bad ugly
```



Documenting User Procedures

The Good

- Attributes can be attached to user-defined procedures to provide a help context
- Use the `define_proc_attribute` to setup the information

```
define_proc_attributes P_get_license \  
  -info "TOPS: Procedure to get a license" \  
  -define_args {  
    {ln "license name" ln string required}  
  }
```

Defining user help info

```
dc_shell-t> help -v P_get_license  
P_get_license #TOPS: Procedure to get a license  
  ln          (license name)  
dc_shell-t>
```

Help results



Variables

The Good

- Tcl variables are evaluated as strings
 - Different when compared to other programming languages
 - Variables are local in scope to procedures
 - Errors can occur when variables are referenced before declared

Fails, `fist_full_of_dollars`
is not declared

```
#!/usr/bin/perl
set outlaw $fist_full_of_dollars
```

Works by declaring
variable first

```
set fist_full_of_dollars ""
#!/usr/bin/perl
set outlaw $fist_full_of_dollars
```

Better method using
the `info` command

```
if {[info exists for_a_few_dollars_more]} {
    expr $for_a_few_dollars_more + 1
} else {
    set for_a_few_dollars_more 1
}
```

Gregg D. Lahti & Steve J. Brown

Hashes and Associative Arrays

The Good

- Like Perl, Tcl incorporates the use of associative arrays that use a string as the index or key to the array elements
- Arrays are stored internally to the Tcl interpreter as hash tables
- Tcl provides useful commands to manipulate or describe arrays:

exists	endsearch	set	search	size
names	startsearch	get	nextelement	

```
%set myarray(actor)
eastwood
%set myarray(tv)
rawhide
%
```

Array example

```
%array exists myarray
1
%array get myarray
actor eastwood tv rawhide
%array names myarray
actor tv
```

Useful commands to manipulate
and describe an array

- Regular expressions in Tcl like UNIX

- Different than Perl, but still powerful

```
# Get the instance name of the violating cell
regexp {(.)/. *} $reg_cell var1 var2
set inst_name [file tail $var2];
# Get the hierarchy path to the violating cell
regexp {(.)/. *} $var2 var1 hier_path
```

- Tcl has great time utilities

- Can easily format time to the user's requirements

```
proc P_timestamp {} {
    set c [clock format [clock seconds]];
    set h [sh hostname];
    set u [sh whoami];
    echo "#TIMESTAMP: " $c " " $h " " $u;
}; # end P_timestamp;
```

- File operations in Tcl can be OS-neutral

Using file join to create paths

```
set my_library [file join libs $my_lib]
```

Poor use of setting paths

```
set my_library libs/$my_lib
```




Sockets

(How to Fix the Lack of Tk)

The Good

- Sockets in Tcl are network-like communication channels based upon TCP based on server/client programming model
- Sockets can fix the lack of Tk in Design Compiler (DC)
- What is Tk?
 - Tk is the sister-language of Tcl which allows the user to program custom Graphical User Interfaces (GUIs)
 - Tk is usually built in with Tcl, except for the Synopsys implementations
- Paper contains code example that creates a window of scrollable text and displays information from the DC session
 - Server script executes using `wish` that accepts information over socket
 - Server script executes the Tk windowing (GUI) functions
 - Info passes between Design Compiler and server script using `puts [socket] [string]` commands and is displayed in window

Gregg D. Lahti & Steve J. Brown



The Bad:

Tcl Quirks That Trap Programmers

The Bad

- Tcl has non-mainstream language syntax and operation issues which cause problems to experienced programmers
 - The use of “ ”, { }, and []
 - Global variables and scope issues
 - Procedures hide operations
 - Line terminations
 - Oddities of conditionals and braces



The Use of “ ”, { }, and []

The Bad

- Braces { } and quotes “ ” group objects
- Braces do not expand variables, quotes expand variables
- Brackets [] allow command grouping
 - commands in brackets executed first

```
set movie {High Plains Drifter}
set movie "High Plains Drifter"
```

Using braces and quotes

```
%set movie {$bar}
$bar
%set outlaw {Jose Wales}
Jose Wales
%set movie "The Outlaw $outlaw"
The Outlaw Jose Wales
```

Braces do not expand variables
Quotes expand variables

Using brackets to
group commands

```
%set mystring {Paint Your Wagon}
Paint Your Wagon
%set line_length [string length $mystring]
16
```



Global Variables and Scope Issues

The Bad

- Tcl is not like C/C++ with regards to global variables
- Global variables must be declared “global” in the **procedure**
 - Declaring variables outside the procedure does **NOT** make them global
 - Opposite from C/C++, Perl

```
proc printit {} {
    echo "mystring is: $mystring"
}

%set mystring {Pale Rider}
Pale Rider
%printit
can't read "mystring": no such
variable
%
```

Incorrect use of global variables

```
proc setit { myvar } {
    global mystring
    set mystring $myvar
}

proc printit {} {
    global mystring
    echo "mystring is: $mystring"
}

%setit {Pale Rider}
Pale Rider
%printit
my string is: Pale Rider
```

Using global variables

Procedures Hide Everything!

The Bad

- Procedures have a completely new variable space
 - Synopsys variables are subject to this rule!
 - Must declare variables as “global” before **usage**

```
set bus_naming_style {%s[%d]};

proc proj_change_names {} {
  global bus_naming_style;
  # echo commands to log file
  set proc_body "info body P_proj_change_names";
  echo [eval $proc_body];
}
```

Required in order
to use new variable
value!

Code used to
echo procedure body
to STDOUT

Variable scoping example

- Procedures inherently hide execution info from the user
 - Output to STDOUT from commands get “squelched” by procedure
 - Errors within a procedure terminate the procedure, no easy method to track offending command (Irritating when debugging errors)

Gregg D. Lahti & Steve J. Brown



Line Terminations

The Bad

- Comment characters don't always get understood correctly by the Tcl interpreter
 - Get swallowed by parser as another argument to command

```
DCSHell-t> set bar [expr 3 * 4]    # test mult
Error: wrong # args: should be "set varName
?newValue?"
      use error_info for more info. (CMD-013)
```

No semi-colon before comment character #, fails parser

- Use a semi-colon (line termination character) to tell parser that the command is finished and a comment follows

```
DCSHell-t> set bar [expr 3 * 4]    ;# test multiply
12
```

A semi-colon precedes the comment character #



Oddities of Conditionals and Braces

The Bad

- The Tcl parser is very picky where you put beginning and ending braces in conditional statements

```
set death_valley 1
if {$death_valley == 1}
{
    echo "death_valley was set to 1"
}
```

Incorrect brace position, fails parser

```
set death_valley 1
if {$death_valley == 1} {
    echo "death_valley was set to 1"
}
```

Correct brace position

- Synopsys error messages are really cryptic!

```
Error: wrong # args: no script following
"$death_valley == 1" argument
    use error_info for more info.  (CMD-013)
Error unknown command `
    echo "death_valley was set to 1"
` (CMD-005)
```

Synopsys error message from incorrect brace position

Gregg D. Lahti & Steve J. Brown



The Ugly: Synopsys-Specific Tcl Implementations

The Ugly

- Synopsys added extensions to the Tcl interface which are not Tcl-specific but solve issues with Design Compiler and PrimeTime integration
- Some of these commands require greater understanding and more Tcl code workarounds for effective use
 - Collections
 - Attributes
 - Accessing Collections and Attributes
 - Collection Gotchas
 - Tcl-Tortured, Synposys-Specific Commands



Collections

The Ugly

- Collections are Synopsys-specific programming items that allow attributes to be grouped into a single variable reference
 - Must be accessed differently than variables of strings, lists, or arrays
 - Most Synopsys commands return a collection type

```
% set in_list [all_inputs]
{"clk", "rst_n", "capture"}
% set item0 [lindex $in_list 0]
{"clk", "rst_n", "capture"}
% set item0
{"clk", "rst_n", "capture"}
%
```

Incorrect method of accessing a collection

```
% set in_list [all_inputs]
{"clk", "rst_n", "capture"}
% set item0 [index_collection \
$in_list 0]
{"clk"}
% set item0
{"clk"}
%set port_name [get_object_name \
$item0]
clk
% set port_name
clk
```

Correct method of accessing a collection



Attributes

The Ugly

- Attributes help define Synopsys items with special synthesis information that Design Compiler can utilize
- Synopsys supports these attributes:

cell	clock	design
library	net	port
read_only	reference	pin

- Attributes can be user-defined

```
set reset_name "sysrst";
echo "#RESET-Info: Setting reset attributes on port - $reset_name";
remove_driving_cell $reset_name;
set_ideal_net $reset_name;
set_drive 0 $reset_name;
set_false_path -from $reset_name;
set_attribute $reset_name is_reset true -type Boolean;
```

Using set_attribute to create a user-defined attribute

- Synopsys has provided access to the contents of collections through commands:

```
sort_collection          filter_collection
foreach_in_collection
```

```
# find flops called FD1S in design
filter_collection [get_cells *] "ref_name == FD1S";
```

Using the filter_collection command

- Some commands can optionally pass a **-sort** or **-filter** command as an argument

```
set unmapped_cells [get_cells -filter {@is_unmapped == true} "*"];
set clk_ports [get_ports -filter {@is_clock == true} "*"];
```

Using the -filter option with the get_cells command



Collection Gotchas

The Ugly

- In Tcl-mode of Design Compiler, there is no easy method of adding or subtracting items within a collection
 - In DCSH, just subtract output of one command from another

```
in_list = all_inputs() - all_clocks()
```

DCSH method of getting all inputs without any clocks

- in DCTcl mode, all_inputs and all_clocks return collections
- Instead, must operate on each element of collection

```
set inlist [all_inputs]      ;# inlist contains a collection!  
foreach_in_collection ic [all_clocks] {  
    set inlist [remove_from_collection $inlist [get_object_name $ic]]  
}
```

Tcl method to do equivalent operation



Tcl-Tortured, Synopsys-Specific Commands

The Ugly

- Setting variables on the command line of `dc_shell` using the `-x` facility must be Tcl-based, rather than DCSH-based

```
dc_shell-t -x "MOVIE=good_bad_ugly"
```

Supported in 1999.10

```
dc_shell-t -x "set MOVIE good_bad_ugly"
```

Newer method for 2000.05

- UNIX-style redirection must now use the `redirect` command

```
report_cell > cell.rpt
```

Supported in 1999.10

```
redirect cell.rpt {report_cell}
```

Newer method for 2000.05

- Some Synopsys commands return output to STDOUT only and cannot be set to a variable
 - This "feature" occurs in most Synopsys commands
 - Most notably, `list_license` and `report_constraint`



Tcl-Tortured, Synopsys-Specific Commands

The Ugly

- To overcome this “feature”, redirect output to a temporary file and then read the file back into a variable
 - Example shows `eval` usage to get Process ID of session

```
exec touch [eval pid].tmp;  
exec rm -rf [eval pid].tmp;  
redirect [eval pid].tmp {list_licenses};  
set licenses [exec cat [eval pid].tmp];  
exec rm -rf [eval pid].tmp;
```

exec, redirect and eval Example

```
redirect tmp { report_constraint -min_delay -all };  
set violator_list [exec sed -n -e "/VIOLATED/p" tmp];  
sh rm tmp;
```

Using redirect to get constraint violations into a variable



Items for Enhancement

- Fix commands like `list_licenses`, `report_constraint`, etc. to return string values that can be assigned to a variable rather than text output to the Design Compiler console
- Ability to display commands being executed within procedures to console (STDOUT)
- Inclusion of Tk into Design Compiler and PrimeTime
- Ability to add or subtract collection elements

```
in_list = expr [ [all_inputs] - [all_clocks] ]
```
- Adding `-example` to `define_proc_attributes` command
 - One info line isn't enough
- A built-in Tcl debugger
 - Command line like the perl debugger
 - Graphical version in Design Vision

Gregg D. Lahti & Steve J. Brown



Conclusions

- Tcl is a very powerful language that enhances the synthesis tasks
- Tcl does have quirks and oddities in its usage and syntax that conflict with more mainstream programming languages
- The Synopsys-specific Tcl features create some confusion in implementing Tcl and require extra attention
- Examples were used from the The TOPS synthesis environment. TOPS was a joint effort by:
 - Gregg Lahti
 - Steve Brown
 - Tim Wilson
 - Rodney Pesavento
 - Doug Hergatt (CX Design)

Gregg D. Lahti & Steve J. Brown



Reference

- Ousterhout, John K., Tcl and the Tk Toolkit, Addison-Wesley, 1994. ISBN: 020163337X
- Welch, Brent B., Practical Programming in Tcl & Tk, Second Edition, Prentice Hall, 1997. ISBN 0-13616830-2
- Nelson, Christopher, Tcl/Tk Programmer's Reference, Osbourne/McGraw Hill, 2000. ISBN 0-07-212004-5
- Wilson, Tim L, and Pesavento, Rodney, Using Tcl to Implement an Efficient Synthesis Environment (TOPS), Boston Synopsys Users Group Conference, September 2000
- Tcl Developer Exchange Website: <http://ajubasolutions.com>
- Synopsys Solvnet Website: <http://synopsys.com>
- Deepchip Website: <http://www.deepchip.com>

Gregg D. Lahti & Steve J. Brown