# ECE201D W '22 TCL Tutorial

WOJCIECH ROMASZKAN

NANOCAD LABORATORY

ELECTRICAL AND COMPUTER ENGINEERING, UCLA

# TCL Basics

- Material is based on:
  - https://tcl.tk/man/tcl8.5/tutorial/tcltutorial.html
  - https://www.tutorialspoint.com/tcl-tk/index.htm
  - Internal Cadence tutorials (not bundled with documentation).

# TCL Basics

- All TCL in this class will be run within the tools (Genus, Innovus etc.).
  - Either through interactive shell or scripts.

```
@genus:root: 2> puts "Hello World!"
Hello World!
@genus:root: 3> source synthesize.tcl
```

- Comments start with "#" (full line) or ";#" (after a command).

```
@genus:root: 4> # Comment line
@genus:root: 5> puts "Hello World!" ;# Comment
Hello World!
```

- TCL command are one or more words, comma separated.

```
<command_name> <arg1> <arg2> ...
```

- Use man <command_name> to get help on a given command.

# TCL Variables

- Variables need a name (letters, digits, underscores), and a value (optional).
  - TCL is case sensitive (clk != CLK).

```
set <var_name> ?<var_value>? ;# Create/retrieve value
unset <var_name>             ;# Deletes a variable
info exists <var_name>       ;# Check if exists
```

- Variables hold string values.
  - But can be interpreted as numbers (TCL is dynamically typed).

```
@genus:root: 8> set a "Some string"
Some string
@genus:root: 9> set a
Some string
@genus:root: 10> info exists a
1
```

# Variable Substitutions

- "$" is used to substitute variable value.
  - Escaped "\$" disables substitution.

```
@genus:root: 11> set tt 10
10
@genus:root: 12> puts "Time = $tt ns" ;# weak quote
Time = 10 ns
```

- Variables within "{}" are not substituted.

```
@genus:root: 15> puts {Time = $tt ns} ;# rigid quote
Time = $tt ns
```

- Standard wildcard substitution is enabled (*/?).

```
@genus:root: 17> get_nets *clk*
0x2
hnet:s1494_bench/blif_clk_net
```

# Command Substitutions & Expressions

- [<command>] substitutes the result of a command.
  - [] can be nested.

```
@genus:root: 22> set c [expr $tt + 5]
15
@genus:root: 23> puts "New time = [expr $tt + 5]"
New time = 15
```

- "expr" is used to evaluate expressions.
  - Logical, arithmetic, relational, bit-wise etc.

```
@genus:root: 25> expr {$tt == $c}
0
@genus:root: 26> expr {$tt != $c}
1
```

# TCL Numbers

- By default, numbers are treated as integers.
  - decimal, octal (starts with 0), hex (starts with 0x).

```
@genus:root: 27> expr {$tt == 0xA}
1
```

- Floating point numbers must follow one of the following formats:
  - 2.1, 3., 6E4, 7.91e-16, .0001.
  - Integer by integer operations return an integer.

```
@genus:root: 28> expr {2.1*10}
21.0
@genus:root: 29> expr {1/2}
0
```

# Operators

- Operators:
  - +,-,*,/,% - add, subtract, multiply, divide, reminder. Integer by integer operation returns an integer (e.g. 1 / 2 = 0).
  - ** - exponent.
  - &, | , ^ - bitwise AND, OR, XOR.
  - &&, | | - logical AND, OR.
  - <, >, <=, =>, == - relational operators.

```
@genus:root: 31> set a 2
2
@genus:root: 32> expr $a > 0 && $a <= 3
1
@genus:root: 33> set b 0x7
0x7
@genus:root: 34> expr ~$b
-8
@genus:root: 35> expr $b & 0x4
4
```

# Math Functions

- Available functions: `abs acos asin atan atan2 bool ceil cos cosh double entier exp floor fmod hypot int isqrt log log10 max min pow rand round sin sinh sqrt srand tan tanh wide`

```
@genus:root: 42> set c 2
2
@genus:root: 43> expr exp($c)
7.38905609893065
```

# String Manipulation

- Use "string" command.
  - string <option> <arg1> …

```
@genus:root: 47> set str1 "wire_1"
wire_1
@genus:root: 48> string compare $str1 "wire_10"
-1
@genus:root: 49> string length $str1
6
@genus:root: 50> string cat $str1 "_int"
wire_1_int
```

- Remember about "man" (e.g. man string), to get help.

# Lists

- Three ways of creating a list: {}, "", list command.

```
@genus:root: 53> set list1 "a b c"    ;# string is a list
a b c
@genus:root: 54> set list2 {a b c}
a b c
@genus:root: 55> set list3 [list a b c]
a b c
```

- There are special commands to work with lists: lindex, llength, lappend, linsert, concat.

```
@genus:root: 56> lindex $list1 0
a
@genus:root: 57> llength $list2
3
@genus:root: 58> set list4 [concat $list2 $list3]
a b c a b c
```

# List ctd. & Arrays

- Lists can be nested (multi-dimensional).

```
@genus:root: 60> set list5 {a 1 10 {something 22} 55}
a 1 10 {something 22} 55
@genus:root: 61> lindex [lindex $list5 3] 0 ;# try to decode this
something
```

- Arrays are associative (think python dictionaries).
  - Key-value pairs: set <arr_name>(<key>) <value>
  - "array" function provides utilities for working with arrays.

```
@genus:root: 62> set students(1) "John Doe"
John Doe
@genus:root: 63> set students(2) "Jane Doe"
Jane Doe
@genus:root: 64> puts $students(2)
Jane Doe
@genus:root: 65> array size students
2
```

# Control Flow

- TCL supports standard if/for/while/switch/foreach statements.

```
if {expression1} {
script1
} elseif {expression2} {
script2
} elseif {expression3} {
script3
…
} else {
final_script
}
```

```
for {initialization_expr}
\
{loop_terminate_expr} \
{update_expr} {
Script
}
```

```
foreach var $any_list
{
Script
}
```

```
@genus:root: 71> for {set
i 0} {$i < 3} {incr i} {\
==> puts "wire_$i"}
wire_0
wire_1
wire_2
```

```
@genus:root: 72> foreach
val $list1 {\
==> puts $val }
a
b
c
```

- You can look up while/switch syntax if you need it.

# Working with Files

- "file" provides basic file operations.

```
@genus:root: 80> file exists s1494.v
1
@genus:root: 81> file extension s1494.v
.v
```

- "open" is used for opening files.
  - Returns file ID to use.
  - puts $<file_id> <value> to write to file.
  - gets $<file_id> <variable> to read a line from file. read $<file_id> to read whole file.
  - close $<file_id> to close file.

```
@genus:root: 83> set fid [open "test.txt" w+]
file16
@genus:root: 84> puts $fid "Logfile"
@genus:root: 85> close $fid
```

# Parsing Through a File

```
# Option 1: Read line by line
set fid [open “logfile.txt” r]
while {[gets $fid line] >= 0} {
# process lines
}
close $fid

# Option 2: Read the whole file
set fid [open "logfile.txt" r]
set file_data [read $fid]
close $fid
# Process data file
set data [split $file_data "\n"]
foreach line $data {
# process lines
}
```

# Genus/Innovus Collections

- get_* and all_* commands create collections.
  - Collections are sets of design objects.
  - Collections are not lists.

```
@genus:root: 6> set ports [get_ports *]
0x5
@genus:root: 7> lindex $ports 0
0x5
```

- Operating on collections requires special functions.

```
@genus:root: 10> query_objects $ports ;# queries all objects
port:s1494_bench/blif_clk_net port:s1494_bench/blif_reset_net ...
@genus:root: 11> get_object_name $ports ;# extracts object names
blif_clk_net blif_reset_net CLR v6 v5 v4 v3 v2 v1 v0 …
@genus:root: 12> sizeof_collection $ports
29
```

# Genus/Innovus Collections ctd.

- index_collection is used to access indexed elements.
  - Combined with query_objects can be used to extract specific objects.

```
@genus:root: 14> index_collection $ports 0
0x7
port:s1494_bench/blif_clk_net
@genus:root: 15> query_objects [index_collection $ports 0]
port:s1494_bench/blif_clk_net
```

- foreach_in_collection is used to iterate through collections.

```
@genus:root: 24> foreach_in_collection port $ports {puts
[get_object_name $port]}
blif_clk_net
blif_reset_net
CLR
V6
...
```

# Filtering Collections

- You can filter a collection using filter_collection:
  - filter_collection <collection> <expression>
  - -regexp option can be used for regular expressions.
    - Use =~ for regexp comparisons.

```
@genus:root: 60> filter_collection $ports
"name==blif_clk_net"
0x44
port:s1494_bench/blif_clk_net
@genus:root: 61> filter_collection -regexp $ports name=~".*clk.*"
0x45
port:s1494_bench/blif_clk_net
@genus:root: 62> set clocks [filter_collection -regexp $ports
name=~".*clk.*"]
0x54
```

# Querying Objects

- Genus/Innovus objects have properties.
  - You can find a list of properties for a given object type using list_property.

```
@genus:root: 77> list_property –type port
Property : address
Property : full_name
Property : base_name
Property : obj_type
...
```

- Object types: pin, port, cell, net, clock, lib_cell, lib_pin, design, lib, timing_path, timing_point, timing_arc, path_group, lib_timing_arc, si_victim, si_attacker, pg_pin, pg_net.
- You can retrieve properties using get_property, or get_db.

```
@genus:root: 82> get_property $clocks obj_type
port
@genus:root: 83> get_db [index_collection $clocks 0] .obj_type
port
```