Chapter 5 :
Additional Reading Materials

- DC-Tcl

DC-Tcl

Tcl = Tool Command Language (tickle):

- DC-Tcl is the command interface for DC in XG mode
- Built on the “open” industry-standard shell programming language Tcl
- DC-Tcl an interpreted scripting language

Many Synopsys tools support Tcl for consistency, e.g. Design Compiler, Formality, PrimeTime, Physical Compiler and more.

Tcl was originally developed by John K. Ousterhout at UC Berkeley.

There are many books on the topic of Tcl programming, here are a few:

- Tcl and the Tk Toolkit, John K. Ousterhout
- Practical Programming in Tcl and Tk, Brent B. Welch

Some Tcl web sites for reference and further information:

www.tcl.tk (documentation and advanced packages for Tcl, same as www.scriptics.com)

www.tclforeda.net (many DC script examples and other useful tools for Logic Designers)

For training on Tcl, please refer to three additional workshops from CES.

The Power of Tcl 1: Becoming a Proficient Tcl User

The Power of Tcl 2: Creating High-Impact Procedures

The Power of Tcl 3: Direct Access through Collections and Attributes

- Many Synopsys tools support Tcl for consistency, e.g. Design Compiler, Formality, PrimeTime, Physical Compiler and more.
- Tcl was originally developed by John K. Ousterhout at UC Berkeley.
- There are many books on the topic of Tcl programming, here are a few:
  - Tcl and the Tk Toolkit, John K. Ousterhout
  - Practical Programming in Tcl and Tk, Brent B. Welch
- Some Tcl web sites for reference and further information:
  - www.tcl.tk (documentation and advanced packages for Tcl, same as www.scriptics.com)
  - www.tclforeda.net (many DC script examples and other useful tools for Logic Designers)

Converting from dcsh to DC-Tcl

A program is avilable for users to migrate from “old” dcsh to DC-Tcl.

```
UNIX% dc-transcript  my_script.scr  my_script.tcl
```

- Will convert most commands in existing scripts to Tcl
- Only goes from DCSH to DC-Tcl
- Called from the UNIX prompt

The dc-transcript utility accurately translates most existing dcsh mode scripts.

The dc-transcript does not do the following:

- Does not check the syntax of your dcsh mode scripts, although serious syntax errors will stop the translation
- Does not, in general, check the semantics of your commands
- Does not optimize your scripts
- Does not, in general, teach you how to write Tcl scripts
- Does not always update your dcsh mode commands to the most current and preferred Tcl mode commands

- The dc-transcript utility accurately translates most existing dcsh mode scripts.
- The dc-transcript does not do the following:
  - Does not check the syntax of your dcsh mode scripts, although serious syntax errors will stop the translation
  - Does not, in general, check the semantics of your commands
  - Does not optimize your scripts
  - Does not, in general, teach you how to write Tcl scripts
  - Does not always update your dcsh mode commands to the most
    current and preferred Tcl mode commands

Executing DC-Tcl Scripts

- Commands can be typed:
  - Interactively in DC Tcl

```
dc_shell-xg-t> echo "Running my.tcl..."
dc_shell-xg-t> source –echo –verbose my.tcl
```

  - Executed in batch mode

```
UNIX% dc_shell-xg-t -f my.tcl | tee –i my.log
```

The tee command displays the results on the screen and writes them into the specified log file.

The `tee` command displays the results on the screen and writes them into the specified log file.

Tcl Basics

- Tcl command =
  - One or more words separated by white space
  - First word is command name, others are arguments
  - Returns string result
- Tcl script =
  - Sequence of commands
  - Commands are separated by newlines and/or semi-colons

Examples:

`set a 22`	set the variable ‘a’ to 22
`echo "Hello World!“`	world’s most common program

Variable Substitution

- Syntax: *$varName*
- Variable name is letters, digits, underscores
- Substitution may occur anywhere within a word:

Sample commands	Results
`set b 66`	`66`
`set a b`	`b`
`set a $b`	`66`
`set a $b+$b+$b`	`66+66+66`
`set a $b.3`	`66.3`
`set a $b4`	`no such variable`

To remove a variable, use the command unset, example:

```
unset b
```

Variables can be concatenated with strings in many ways, e.g. to get the contents of the variable b concatenated with the string "test", you type:

`set a ${b}test` -> "66test"

Variables do not need declaration as in languages like C, Pascal, etc., since there is only one "type" of variable – a string. The string may be interpreted in different ways by the command itself, e/g/ the `expr` command (shown later) may interpret the string as an integer or as a floating point number.

- To remove a variable, use the command unset, example:
  - `unset b`

- Variables can be concatenated with strings in many ways, e.g. to get the contents of the variable b concatenated with the string “test”, you type:
  - `set a ${b}test` -> “66test”
- Variables do not need declaration as in languages like C, Pascal, etc., since there is only one “type” of variable – a string. The string may be interpreted in different ways by the command itself, e.g. the `expr` command (shown later) may interpret the string as an integer or as a floating point number.

Nested Commands

- Syntax: [***commands…***]
- Evaluate command, return result
- May occur anywhere within a word:

```
Sample command                    Result
set b 8                           8
set a [expr $b+2]                 10
set a "b-3 is [expr $b-3]"        b-3 is 5
```

Command substitution produces:
`set a "b-3 is 5"`
Then, the command "set" is executed

Note: "expr" is a Tcl function that performs math operations.

"`expr`" is a Tcl function that performs math operations.

Defining Words

- Words end or break at *white space* and *semi-colons*, except:
  - Double-quotes prevent breaks

    `set a "x is $x; y is $y"`
  - Curly braces prevent breaks and substitutions

    `set a {[expr $b*$c]}`
  - Backslashes escape special characters

    `set a word\ with\ \$\ and\ space`
  - Backslashes can escape newline (line-continuation)

    `report_constraint \`
    `  -all_violators`

```
set x 3
set y 5
set a “x is $x; y is $y”        ; #Sets the variable a to “x is 3; y is 5”
set a {[expr $b*$c]}            ; #Sets the variable a to “[expr $b*$c]”

set a word\ with\ \$\ and\ space ; #Sets variable a to “word with $ and space”

report_constraint \
  all_violators
```

Make sure that there is no space after the backslash. “Line-continuation” means “backslash – newline.”

Notice that a \+newline is evaluated as a space. e.g.

```
set a "1 2\
3 4“
```

sets a to “`1 2 3 4`” – with a space between the 2 and the 3!

- `set x 3`
- `set y 5`
- `set a “x is $x; y is $y”;` #Sets the variable a to “`x is 3; y is 5`”
- `set a {[expr $b*$c]} ;` #Sets the variable a to “`[expr $b*$c]`”

- `set a word\ with\ \$\ and\ space;` #Sets variable a to “`word with $ and space`”

- `report_constraint \`
  `all_violators`

  Make sure that there is no space after the backslash. “Line-continuation” means “backslash – newline.”

- Notice that a \+newline is evaluated as a space. e.g.
- `set a "1 2\`
  `3 4“`
  sets a to “`1 2 3 4`” – with a space between the 2 and the 3!

Comments in DC-Tcl

```
# Comments in Tcl

# If you want to comment on the same line, be sure
# to use a semicolon before the comment:

set header_str "Output Header" ; # Same line comment
```

This semicolon is required!

Comment a line in a DC-Tcl script using the '#' character

Using Wildcards

- DC-Tcl supports two wildcard characters:
  - ***** will match zero to ‘n’ characters
  - **?** matches exactly 1 character

Examples:

```
dc_shell-xg-t> help create*
dc_shell-xg-t> set_input_delay 5 -clock CLK \
                              [get_ports BUS*]
```

Arithmetic Expressions

To evaluate arithmetic expressions use the `expr` command.

```
dc_shell-xg-t> set period 10.0
10.0

dc_shell-xg-t> set freq [expr  1 / $period]
0.1

dc_shell-xg-t> echo "Freq = " [expr $freq * 1000] "MHz"
Freq = 100.0 MHz

dc_shell-xg-t> set_load [expr [load_of \
               ssc_core_slow/and2a0/A] * 5] [all_outputs]
```

To have the result of `expr` represented as a floating point number, at least one of the numbers involved in the calculation has to be a float. The number 7 becomes 7.0 if floating point is required.

e.g. the command:

```
expr 5/2
```

will return 2.

If a floating point answer is required, use:

```
expr 5.0/2
```

This will return 2.5

- To have the result of `expr` represented as a floating point number, at least one of the numbers involved in the calculation has to be a float. The number 7 becomes 7.0 if floating point is required.
- e.g. the command:
  - `expr 5/2`
  - will return 2.
- If a floating point answer is required, use:
  - `expr 5.0/2`
  - This will return 2.5

Using Lists in DC-Tcl

Arrange *your* data as lists, example:

```
dc_shell-xg-t> set colors {red green blue}
red green blue
dc_shell-xg-t> echo $colors
red green blue
dc_shell-xg-t> set Num_of_Elements [llength $colors]
3
dc_shell-xg-t> set colors [lsort $colors]
blue green red
```

```
dc_shell-xg-t> set link_library {*}
*
dc_shell-xg-t> lappend link_library tc6a.db opcon.db
* tc6a.db opcon.db
dc_shell-xg-t> echo $link_library
* tc6a.db opcon.db
```

To manipulate lists, use Tcl built-in list commands:

`concat`	Concatenates two lists and returns a new list
`join`	Joins elements of a list into a string
`lappend`	Creates a new list by appending elements to a list
`lindex`	Returns a specific element from a list
`linsert`	Creates a new list by inserting elements into a list
`list`	Returns a list formed from its argument
`llength`	Returns the number of elements in a list
`lrange`	Extracts elements from a list
`lreplace`	Replaces a specific range of elements in a list
`lsearch`	Searches a list for a regular expression
`lsort`	Sorts a list
`split`	Splits a string into a list

- To manipulate lists, use Tcl built-in list commands:
- `concat` Concatenates two lists and returns a new list
- `join` Joins elements of a list into a string
- `lappend` Creates a new list by appending elements to a list
- `lindex` Returns a specific element from a list
- `linsert` Creates a new list by inserting elements into a list
- `list` Returns a list formed from its argument
- `llength` Returns the number of elements in a list
- `lrange` Extracts elements from a list
- `lreplace` Replaces a specific range of elements in a list
- `lsearch` Searches a list for a regular expression
- `lsort` Sorts a list
- `split` Splits a string into a list

Iterate through Lists

The following example iterates over a list:

```
set all_colors "red green blue"

foreach color $all_colors {
  echo $color is a nice color...
}
```

```
red is a nice color…
green is a nice color…
blue is a nice color…
```

Objects and Attributes

- Recall that designs consist of objects:
  - Designs, cells, ports, pins, clocks, and nets
- In order to keep track of circuit functionality and timing, DC attaches many attributes to each of these objects:
  - Ports can have the following attributes

    `direction` **`driving_cell`**

    `max_capacitance` others. . .
  - Designs can have the following attributes

    `area` **`operating_conditions_max`**

    `max_area` others…

Accessing the Synopsys Database

- Access to DC *objects* in DC-Tcl is achieved through collections - a DC extension to standard Tcl
- Collections are generally created by `get_` or `all_` commands:

Example:

```
get_ports clk*
set myclocks [all_clocks]
set hi_cap_pins [get_pins
busdriver/tristate*]
```

Partial list of get_* and all_* commands:

```
get_cells          # Create a collection of cells
get_clocks         # Create a collection of clocks
get_designs        # Create a collection of designs
get_libs           # Create a collection of libraries
get_nets           # Create a collection of nets
get_pins           # Create a collection of pins
get_ports          # Create a collection of ports

all_clocks         # Create a collection of all_clocks
all_designs        # Create a collection of all_designs
all_inputs         # Create a collection of all_inputs
all_outputs        # Create a collection of all_outputs
all_registers      # Create a collection of all_registers
```

When these commands are issued, DC **internally** creates a group of objects, along with all their attributes.

- Partial list of get_* and all_* commands:
- `get_cells` `# Create a collection of cells`
- `get_clocks` `# Create a collection of clocks`
- `get_designs` `# Create a collection of designs`
- `get_libs` `# Create a collection of libraries`
- `get_nets` `# Create a collection of nets`
- `get_pins` `# Create a collection of pins`
- `get_ports` `# Create a collection of ports`

- `all_clocks` `# Create a collection of all_clocks`
- `all_designs` `# Create a collection of all_designs`
- `all_inputs` `# Create a collection of all_inputs`
- `all_outputs` `# Create a collection of all_outputs`
- `all_registers` `# Create a collection of all_registers`

- When these commands are issued, DC **internally** creates a group of objects, along with all their attributes.

Collections Are Referenced by a Handle

Just like lists, collections have special access commands.

```
dc_shell-xg-t> set foo [get_ports p*]
{"pclk", "pframe_n", "pidsel", "pad[31]"...}
dc_shell-xg-t> sizeof_collection $foo
50
dc_shell-xg-t> query_objects $foo
{"pclk", "pframe_n", "pidsel", "pad[31]"...}
```

Collection commands return a collection handle, NOT a list!
A list, containing the names of all the objects returned by the **get**_ or **all**_ command is echoed to the screen.

Standard Tcl list commands (concat, llength, etc) will not work with the output of a collection command!

Collection commands return a collection handle, NOT a list!

A list, containing the names of all the objects returned by the **`get`**_ or **`all`**_ command is echoed to the screen.

Standard Tcl list commands (`concat`, `llength`, etc) will not work with the output of a collection command!

Manipulating Collections

```
dc_shell-xg-t> help *collection*
add_to_collection      # Add object(s)
remove_from_collection # Remove object(s) from a
                          collection
. . .
```

```
dc_shell-xg-t> set pci_ports [get_ports "DATA*"]

dc_shell-xg-t> set pci_ports [add_to_collection \
                     $pci_ports [get_ports CTRL*]]

dc_shell-xg-t> set all_inputs_except_clk \
              [remove_from_collection [all_inputs] \
               [get_ports CLK]]
```

```
dc_shell-xg-t> help *collection*

add_to_collection      # Add object(s)
compare_collections    # compares two collections
copy_collection        # Make a copy of a collection
filter_collection      # Filter a collection, resulting
                          in a new collection
foreach_in_collection  # Iterate over a collection
index_collection       # Extract object from collection
remove_from_collection # Remove object(s) from a
                          collection
sizeof_collection      # Number of objects in a
                          collection
sort_collection        # Create a sorted copy of a
                          collection
dc_shell-xg-t> help *object*
more collection related commands…
```

- **dc_shell-xg-t>** help *collection*

- add_to_collection # **Add object(s)**
- compare_collections # **compares two collections**
- copy_collection # **Make a copy of a collection**
- filter_collection # **Filter a collection, resulting in a new collection**
- foreach_in_collection # **Iterate over a collection**
- index_collection # **Extract object from collection**
- remove_from_collection # **Remove object(s) from a collection**
- sizeof_collection # **Number of objects in a collection**
- sort_collection # **Create a sorted copy of a collection**
- **dc_shell-xg-t> help *object***
- **more collection related commands…**

Filtering Collections

- Use the `filter_collection` command to get only objects you are interested in:

```
filter_collection [get_cells *] "ref_name =~ AN*"
filter_collection [get_cells *] "is_mapped != true"
```

- The `-filter` option is a nice short-cut:

```
get_cells * -filter "dont_touch == true"
set fastclks [get_clocks * -filter "period < 10"]
```

- Relational operators are:

```
==, !=, >, <, >=, <=, =~, !~
```

Description of the examples above:

1. Returns all cells starting with the name “AN”
2. Returns all unmapped cells
3. Returns all cells with the “dont_touch” attribute
4. Returns all clocks with a period smaller than 10

`filter_collection` creates a new collection, or an empty string if no objects match the expression.

The `-filter` option is more efficient, because the collection does not have to be read twice.

Other examples:

```
get_cells -hier -filter “is_unmapped != true”
get_cells -hier -filter “is_hierarchical == true”
```

To see all DC defined attributes:

```
dc_shell-xg-t> list_attributes -application
```

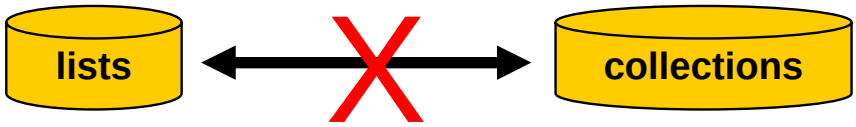
- Description of the examples in the previous slide:
  - 1. Returns all cells starting with the name “AN”
  - 2. Returns all unmapped cells
  - 3. Returns all cells with the “dont_touch” attribute
  - 4. Returns all clocks with a period smaller than 10

- `filter_collection` creates a new collection, or an empty string if no objects match the expression.
- The `-filter` option is more efficient, because the collection does not have to be read twice.
- Other examples:
- `get_cells -hier -filter “is_unmapped != true”`
- `get_cells -hier -filter “is_hierarchical == true”`

- To see all DC defined attributes:
- `dc_shell-xg-t> list_attributes -application`

Summary – Lists/Collections

- *Lists* are structures to store *YOUR* data
- *Collections* are used to access *DB* data
- List commands should not be used on collections and vise versa



The above is a strong recommendation. DC does allow some mixing of lists and collections, this does not mean that it should be done.
The following is allowed:
set port_col [list [get_ports a*] [get_ports b*]]
port_col: is a list with two collections. This list may be passed to other collection manipulation commands.
It is better to convert the command to this:
set port_col [get_ports "a* b*"]

The above is a strong recommendation. DC does allow some mixing of lists and collections, this does not mean that it should be done.

The following is allowed:

```
set port_col [list [get_ports a*] [get_ports b*]]
```

`port_col:` is a list with two collections. This list may be passed to other collection manipulation commands.

It is better to convert the command to this:

```
set port_col [get_ports "a* b*"]
```

Recommendations

- Avoid using aliases and abbreviating command names in scripts
- Use common extensions:
  `e.g. foo.tcl`
- Use full option names in commands:
  `create_clock -period 5 [get_ports clk]`
- Avoid “snake scripts”
  - “Snake scripts” are scripts that call scripts, that call scripts: Very hard to debug.
- Avoid sourcing scripts from your .synopsys_dc.setup file, since these scripts will be executed automatically every time you start the tool.

“Snake scripts” are scripts that call scripts, that call scripts: Very hard to debug.

Avoid sourcing scripts from your .synopsys_dc.setup file, since these scripts will be executed automatically every time you start the tool. This of course excludes scripts that only define procedures for later use.

Need Help?

DC Tcl Help:

- Commands:

```
help create*
help –verbose create_clock
create_clock –help
man create_clock
```

- Variables:

```
printvar *_library
echo $target_library
man target_library
```

```
dc_shell-xg-t> help *clock
 clock                    # Builtin
 create_clock             # create_clock
 create_test_clock        # create_test_clock
 remove_clock             # remove_clock
 remove_propagated_clock # remove_propagated_clock
 report_clock             # report_clock
 set_propagated_clock     # set_propagated_clock

dc_shell-xg-t> help -verbose create_clock
 create_clock       # create_clock
   [-name clock_name]     (name for the clock)
   [-period period_value] (period of the clock)
   [-waveform edge_list]  (alternating rise, fall times for
                           1 period)
   [port_pin_list]               (list of ports and/or pins)
```

```
dc_shell-xg-t> help *clock
 clock                    # Builtin
 create_clock             # create_clock
 create_test_clock        # create_test_clock
 remove_clock             # remove_clock
 remove_propagated_clock # remove_propagated_clock
 report_clock             # report_clock
 set_propagated_clock     # set_propagated_clock

dc_shell-xg-t> help -verbose create_clock
 create_clock  # create_clock
    [-name clock_name]     (name for the clock)
    [-period period_value] (period of the clock)
    [-waveform edge_list]  (alternating rise, fall times for 1 period)
    [port_pin_list]  (list of ports and/or pins)
```

Command Summary (Lecture, Lab)

`dc-transcript`	UNIX utility used to translate DCSH script to DC-Tcl script
`set`	Read and write variables
`echo`	Display a value of a variable
`help`	Display command help information
`foreach`	Iterate through a list
`llength`	Returns the number of elements in a list
`sizeof_collection`	Returns the number of elements in a collection
`query_objects`	Returns object names of a collection
`add_to_collection`	Add objects to a collection
`remove_from_collection`	Remove objects from a collection
`get_attribute`	Returns the value of an attribute on a list of design or library objects
`filter_collection`	Filter an existing collection
`man`	Displays reference manual pages
`printvar`	Prints the values of one or more variables