VLSI Design Flow: RTL to GDS

Dr. Sneh Saurabh
Department of Electronics and Communication Engineering
IIIT-Delhi

Tutorial 2
Lecture - 10
Introduction to TCL

Hello everyone. My name is Jasmine Kaur. I am a Ph.D. student at IIIT Delhi, and I will be your TA for the course VLSI Design Flow: RTL to GDS. In this tutorial, we will explore and gain an understanding of TCL commands. TCL stands for Tool Command Language, a powerful scripting language with programming features. TCL is extensively used in VLSI design automation tools and flows.

It allows the designer to automate repetitive tasks, to customize and integrate different design flows and interface with different EDA tools. So, let us look at a few examples to understand TCL commands.

So, moving on to the first example.

```
set List {0 1 2 3 4 5 6}
set index -1
foreach elem $List {
incr index
puts "Index: $index"
if {$elem % 2 == 0} {
lset List $index [expr {-$elem}]
}
puts "Updates list: $List"
}
```

So, in the first example, we will look at the commands for, for each, if, while, continue, break. They work in a similar fashion as in other programming languages. So, in this example, first, we are setting up a variable name using set List {0 1 2 3 4 5 6}. So, set command is used to define a variable (variable name List), and here we are providing 0, 1, 2, 3, 4, 5, and 6 as the list (defining variable value) to the List variable. Next, we are setting another variable, index, and its value is -1: set index -1.

Next is foreach elem $List. So, foreach is used to iterate over the elements in a list. So, we are taking the elements from the List we provided above. In this, it will increment the index: incr index. In the first iteration index was -1. So, after incrementing, it will change to 0. Then puts is used to print to a channel. So here we are, printing the index value: puts "Index: $index". Next, if is used to check a condition, so here, we will check if the elem is divisible by 2: if {$elem % 2 == 0}.

So, this is a modulus operation. So, if it is divisible by 2, it will move into this block. What is happening here is lset command is used: lset List $index [expr {-$elem}]. So, lset is used to change any element in a given list at the given index value. So, we are changing the List here. The list variable here is List, and at the index value of $index, we will put the new value, which is evaluated in the square bracket.

Another important thing to remember is that, in TCL, always the square bracket is first evaluated. So, in this, what is happening is it is evaluating the expression -$elem : [expr {-$elem}]. So what will happen is, for the first iteration at index 0, it will replace the element by its negation, and then puts is being used to print the updated List: puts "Updated list: $List". So, in the next iteration, the elem 1 will be taken from the List, as the index is now incremented to value 1. Since 1 is not divisible by 2, we will not move into this block (lset List $index [expr {-$elem}]) and move on to the next iteration again, i.e., 2. Since it is divisible by 2, it will be replaced by its negation.

We can see using the tclsh command (tclsh ex1.tcl). Also, all the files of TCL need to be saved in .tcl format, and once we run it, we can see that for the first iteration, the negation of 0 is 0, so there is no change. For the next iteration, the List remains unchanged. In the next iteration, for index 2, List element at index 2 will be changed from 2 to -2. Then, for the next iteration, it will be unchanged. For the next iteration, it will negate 4. In the next iteration, since 5 is not divisible by 2, there is no change, and in the last iteration, it is changing it to -6.

So, now let us look at the second example.

```
set fp [open "input.txt" w+]
puts $fp "test"
close $fp

set fp [open "input.txt" r]
set file_data [read $fp]
puts $file_data
close $fp
```

In this example, we will look at the commands open, close, read, and eof, which work with files. Using set command here: set fp [open "input.txt" w+], we are opening a channel with the variable name fp. In this, what is happening is we are opening the file input.txt in w+ mode. This means that the file is being opened in read-and-write mode. It will truncate the file if it already exists; if it does not, it will create the file. The next is puts $fp test. So, using this command, we are printing (puts command is used to print to a channel). So, we are writing test into this channel, which will write it into the input.txt file, then we will close this channel (set fp [open "input.txt" w+]).

For the next command, again, we are opening the channel fp: set fp [open "input.txt" r]. Now, we are opening it in read mode only. So, using a file_data variable, we will read the

contents of the file: set file_data [read $fp], and using puts $file_data, we can show the contents on the terminal and close the channel: close $fp. So, let us run this script using tclsh ex2.tcl. So, we can see that test has been written in the file.

Moving on to the next example.

```
proc printSumProduct {x y} {
set sum [expr {$x + $y}]
set prod [expr {$x * $y}]
puts "Sum is : $sum"
puts "Product is : $prod"
return
puts " This line will not be printed"
}

puts [printSumProduct 10 50]
```

So, in this example, the commands proc and return will be studied. So, these are the commands to create TCL procedures. How do we create a tcl procedure? Let us say we are creating a procedure that prints the sum and product of two numbers. So, the name of the procedure is printSumProduct, and it takes two arguments, x and y: proc printSumProduct {x y}. So, firstly, we are defining a variable sum in which we evaluate the expression that is the sum of the two inputs: set sum [expr {$x + $y}].

In the next variable prod, we are multiplying the two inputs: set prod [expr {$x * $y}]. So, after this, we can print the result using puts command, and return is used to exit the control from this procedure. Then, once the control is returned from this procedure, the next line will not be evaluated: puts " This line will not be printed". So, this line will not be printed in the output of the TCL script. Next, how do we invoke this procedure? We are invoking it here using the puts command: puts [printSumProduct 10 50].

We call it using printSumProduct and give the two inputs values. So, 10 and 50 here. So, let us see what the output will look like—running tclsh ex3.tcl. So, we can see the sum is 60, and the product is 500, which is accurate.

Moving on to the next example.

```
puts [exec ls]
puts [exec pwd]
```

So, in this example, we will show how we can run system commands in TCL. The command for that is exec. In this, we are using puts to show the command output in the terminal. So, we are executing the system command ls using exec command, and also we are using the exec command to run the system command pwd.

So, we will see what the output of these two commands will look like. So, tclsh ex4.tcl. So, here we can see the output shows the list of the files and directories, and the last line is the output of the pwd command.
So, this was all for today's tutorial. You can check the complete list of TCL commands in the TCL manual for a complete list of TCL commands with their detailed description, syntax, and command options. Thank you.