ROCK N ROLL WITH TCL

By

Vikas Sachdeva

What is Tcl

- Tool Command Language
- Tcl is
  - A scripting language
  - An interpreter language for tools
- Designed and crafted by
  - Prof John Oosterhout
  - Of University of California Berkeley

Why Tcl

- Simple syntax
    - Easy to understand
- Ability to easily add Tcl interpreter in applications
- Provides enough programmability for complex scripts

Tcl Fundamentals

Tcl Fundamentals

- Tcl is a string-based command language
- Tcl is interpreted when application runs

Tcl commands

- Basic syntax for a Tcl command
  - command arg1 arg2 ……..
- White space to separate command name and its arguments
- Newline or semicolon to terminate a command
- # for a comment
- Tcl is case sensitive

Tcl commands

- Everything in Tcl is a command
- Basic syntax for a Tcl command
  - command arg1 arg2 ……..
- Command is the name of
  - Tcl built in command
  - Or user defined procedure

Hello World

```
➢puts "Hello World!"
➢puts {hello world}
➢ puts 1
➢ puts “we don’t need no education”
➢ puts “she is buying stairway to heaven” ;
➢# This is a comment
```

- curly baces are not printed
- Note quotes are not printed
- Why??

Variables and substitution

- Basic variable in Tcl is string
- Not necessary to define variable
- Setting a variable

```
➢ set var 5
➢ set my_string “I have become comfortably numb”
```

- Variable substitution

```
➢ set b $var
➢ set b
```

Command substitution

- Using result of the command

```
set var 5
set b [set var]
```

- string length <string>
    - returns the length of string

Command substitution

- If there are several cases of command substitution within a single command, the interpreter processes them from left to right. As each right bracket is encountered, the command it delimts is evaluated.

```
➢ set a [string length [string length
   [string length “its a long way to the top”]]]
```

Backslash substitution

- Replaces backslash with literal value

```
➢set a 5
➢ puts \$a
```

- Replaces \newline with a space and merges following line into current line

```
➢ puts \
      “There is a lady who’s sure”
```

Grouping

- Why grouping???
- Try –

```
➢ puts The length of guitar is [string length guitar]
```

- So we need grouping
- Folloing ways
    - Grouping with double quotes
    - Grouping with curly braces

Grouping with double quotes

- puts “The length of guitar is [string length guitar]”
- set s guitar
- puts “The length of $s is [string length $s]”

- What if we do not want the variable values to be substituted??

Grouping with curly braces

```
puts {The length of guitar is [string length guitar].}

set s guitar

puts {The length of $s is [string length $s]}
```

Grouping

- With quotes
  - Allows substitution
  - Quotes are not included in the value

- With Curly braces
  - Does not allow substitution
  - Braces are not included in the value

Grouping with quotes

- With quotes
  - Allows substitution
  - Quotes are not included in the value

```
set Z “Welcome”
set Z_LABEL “to the hotel california”
puts “$Z $Z_LABEL”
puts “\$Z $Z_LABEL”
puts “$Z [string length $Z_LABEL]”
```

Grouping with curly braces

- With Curly braces
  - Does not allow substitution
  - Braces are not included in the value

```
➢ set Z “Welcome”
➢ set Z_LABEL “to the hotel california”
➢ puts {$Z $Z_LABEL}
➢ puts {\$Z $Z_LABEL}
➢ puts {$Z [string length $Z_LABEL]}
```

Grouping with quotes, curly braces

- puts “{$Z $Z_LABEL}”
- puts {“$Z $Z_LABEL”}

Tcl commands again

- Basic syntax for a Tcl command
  - command arg1 arg2 ……..
- Steps of command evaluation in Tcl
  - Step1 - Command identification
  - Step2 - Grouping
  - Step3 - Substitution (Only one pass for substitution)
  - Step4 - Argument passing
  - Step5 - Command evaluation

Grouping types

- Grouping with double quotes
- Grouping with curly braces
- Grouping with square brackets

```
➢set a [string length guitar]
```

Substitution types

- Variable substitution
- Command substitution
- Backslash substitution

Tcl fundamentals summary

```
➢set my_string  [string length \
        [string length “Smells like teen spirit”]]
```

- Step1 - Command identification
- Step2 - Grouping
- Step3 - Substitution (Only one pass for substitution)
- Step4 - Argument passing
- Step5 - Command evaluation

Math Expression in Tcl

Math expressions

- Tcl interpreter itself does not evaluate math expressions.
- expr command is used for this

- expr 4 / 5
- Save result of 4/5 in some variable ???
- Add one to length of string and store in some variable in one line ??

Basic Arithmatic operators

- +,-,*,/,%,!
- <,>,<=,>=
- ==,!=,eq,ne
- &&,||

Builtin math functions

- Sin(x), cos(x)
- floor(x),log(x),log10(x)
- sqrt(x),abs(x)
- int(x),round(x)

String Processing in Tcl

String command

- First argument to string command specifies the operation
  - String length str

    Returns number of characters in the string

```
➢string length “Its my life”
```

  - String equal ?-nocase? str1 str2

    returns one if two strings are same

```
➢ string equal “Pink” “Led”
➢ string equal “Pink” “Pink”
```

String command

- String map charMap string

  returns a new string created by mapping characters in the string according to input output list in charMap

```
➢string map {b l o e n d} “bon”
```

String command

- String tolower str

  returns string in lower case

```
➢string tolower “PiNk”
```

- String toupper str

  returns string in upper case

Tcl Lists

Tcl List

- A Tcl list is a sequence of values
- Each value is indexed in numbers
- The index starts from 0

Constructing List

- List command

```
set integer_list [list 1 2 3]
```

- Split command

```
set string_list [split “just another brick in the wall “ “]
```

Useful List Commands

- llength

```
llength $integer_list
```

- lindex

```
lindex $integer_list 0
```

- lappend

```
lappend integer_list 5
```

Useful List Commands

- lsearch

```
➢lsearch $integer_list 2
```

- join

```
➢join $string_list “ “
```

Tcl control structures

Control Structures

- Like everything in Tcl control structures are also commands
- Useful Control Structures
    - if , else , elseif
    - Switch
    - While
    - Foreach
    - For

If, else, elseif

```
If {$x == 0} {
        puts “Divide by zero”
} else {
        puts [expr 1 / x]
}
```

If, elseif, else

```
If {$x == 0} {
        puts “Divide by zero”
} elseif {$x ==1}  {
        puts [expr 1 / x]
}
```

- Note : Curly brace positioning is important

```
If {$x == 0}
{
        puts “Divide by zero”
}
```

Switch

- To branch many one of many commands

```
➢set value 4
➢switch –exact $value {
                    “1” {puts “its one”}
                    “2” {puts “its two”}
          “default” {puts “it matches nothing”
    }
```

While

```
➢set i 1
      ➢   While {$i < 10} {
                     puts $i
                     incr i
                 }

      ➢    While $i < 10 {
                      puts $i
                             incr i
                      }
```

For

```
for {set i 0} {$i < 10} {incr i 2} {
  puts $i
 }
```

Always useful Foreach

- Foreach loop used over list

```
➢    set my_list [list “When” “you” “say” “nothing” “at” “all”]

➢    foreach element $my_list {
                    puts $element
          }
```

Multiple foreach loop variables

- set my_list [list “When” “you” “say” “nothing” “at” “all”]

-
  ```
  foreach {element1 element2} $my_list {
              puts “$element1 $element2”
  }
  ```

break and continue

- Break – come out of the innermost loop
- Continue – continue next iteration of the loop
- Loop commands – while,for,foreach

break

```
for {set i 0} {$i < 10} {incr i 2} {
  if {$i == 4} {
                 break
  }
  puts $i
}
```

continue

```
➢ for {set i 0} {$i < 10} {incr i 2} {
    if {$i == 4} {
              continue
    }
    puts $i
  }
```

Tcl Regular expression

Basic Regular Expression

- **^ Matches the beginning of a string**
- **$ Matches the end of a string**
- **. Matches any single character**
- *** Matches greater than equal to zero of the previous character**
- **+ Matches any count, but at least 1 of the previous character**
- **[...] Matches any character of a set of characters**
- **[^...] Matches any character *NOT* a member of the set of characters following the ^.**

regexp command

- regexp pattern string ?match sub1 sub2 …?

```
set sample “sweet child of mine”
regexp {[a-z]+} $sample
regexp {[a-z]+} $sample match
```

regexp command

- regexp {[a-z]+ [a-z]+} $sample match
- regexp {([a-z]+) ([a-z]+)} $sample match one two

regsub command

- regsub ?switches? pattern string subspec varname
  - -all
  - -nocase

```
➢regsub {[a-z]+} $sample “new” newvar
➢regsub {([a-z]+) ([a-z]+)} $sample “new” newvar
```

Procedure and scope

The proc command

- Proc name params body

```
➢ proc test_zero {a} {
          If {$a == 0} {
                    puts “a is 0”
          } else {
                    puts “a is not zero”
          }
  }
```

Deafult Argument for a procedure

```
➢ proc test_zero {{a 3}} {
          If {$a == 0} {
                    puts “a is 0”
          } else {
                    puts “a is not zero”
          }
  }
```

Returning a value from a procedure

```
➢ proc test_zero {{a 3}} {
            If {$a == 0} {
                        return 1
            } else {
                        return 0
            }
  }
```

- Note the code procedure is ended on return command

Procedure works just like a command

```
set a [test_zero 0]
```

Tcl Arrays

Tcl arrays

- Tcl variable with string value index
- A list is integer indexed
- Array is not ordered
- Tcl arrays are associative arrays
    - Key – the index of the array
    - Value – the value stored in the array

Setting an array

- set arr(index) value
- set my_array(1,2) 5

Accessing an array

```
puts $arr(index)

puts $my_array(1,2)

set alias “index”

puts $arr($alias)
```

Array commands

- Array exists arr
    - Returns 1 if arr is an array variable

```
➢ array exists my_array
```

- array set arr list
    - Sets an array on the basis of key value pair in the list

```
➢ array set my_array {one 1 two 2 three 3}
```

Array commands

- array get arr
    - Returns a list of key value pairs

```
➢ array get my_array
```

- array names arr
    - Return the keys of the array

```
➢ array names my_array
```

Foreach and arrays

```
➢ foreach key [array names my_array] {
          puts "key is $key and value is $my_array($key)
  }

➢ foreach {key value} [array get my_array] {
             puts "key is $key and value is $value
  }
```

Tcl File handling

Opening a file

- open fileName ?access? ?permission?
- Opens a file and returns a filehandle to be used when accessing the file
- ?FileName is the name of the file to open.
- ?access is the file access mode
- ?r......Open the file for reading. The file must already exist.
- ?r+...Open the file for reading and writing. The file must already exist.
- ?w.....Open the file for writing. Create the file if it doesn't exist, or set the length to zero if it does exist
- ?w+..Open the file for reading and writing. Create the file if it doesn't exist, or set the length to zero if it does exist.

Reading/writing a file

- set fileid [open my.txt r]
- gets $fileid line
    - gets command fetches the line from file
- puts $line
- Set new_fileid [open new.txt w]
- puts $new_fileid $line

eof and closing a file

- 1 if eof else 0

```
eof $fileid
```

- Closes file

```
close $fileid
```

Tcl scripts

Running a tcl script

- source script.tcl
- tclsh script.tcl

Exec command

- Running a shell command

```
exec ls
```

Getting command line arguments

- Script command line arguments are stored in a list argv
- argv0 is the name of the script

Accessing environment variables

- Environment variable values can be assessed by array env whose keys are names of the variables

```
puts $env(PATH)
```

Checking existence of a variable

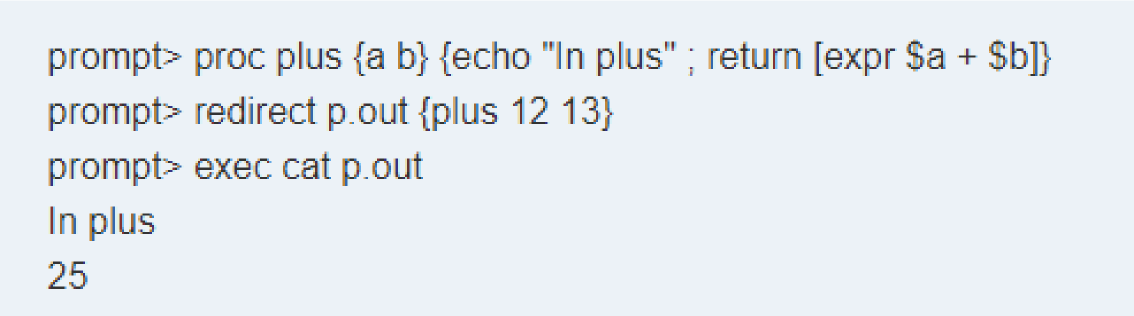
- info exists varname
- array exists arr

Unsetting a variable

- unset varName

redirect command

- redirect
  Redirects the output of a command to a file.

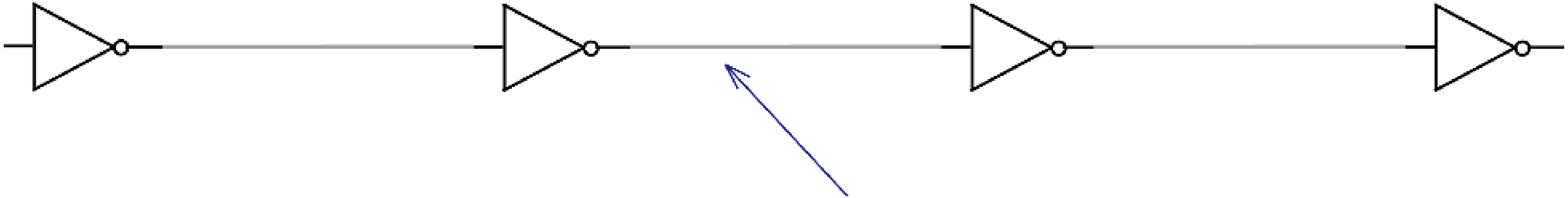


```
prompt> proc plus {a b} {echo "In plus" ; return [expr $a + $b]}
prompt> redirect p.out {plus 12 13}
prompt> exec cat p.out
In plus
25
```

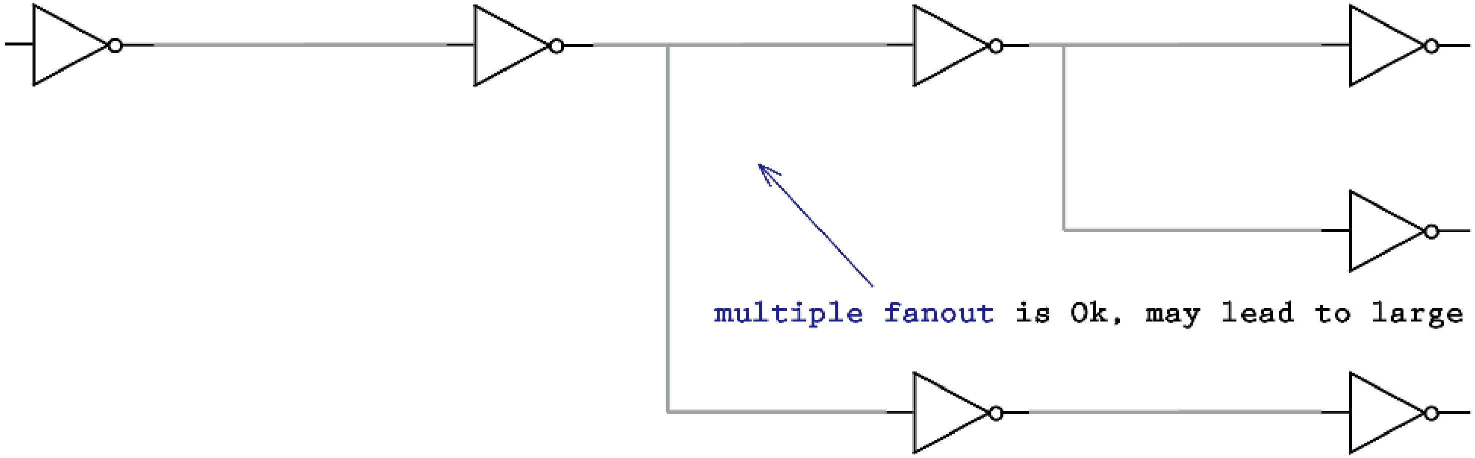
redirect -variable ret {find_transistor m*} puts $ret

Tcl Application Scripts and Synopsys Extension

Netlist

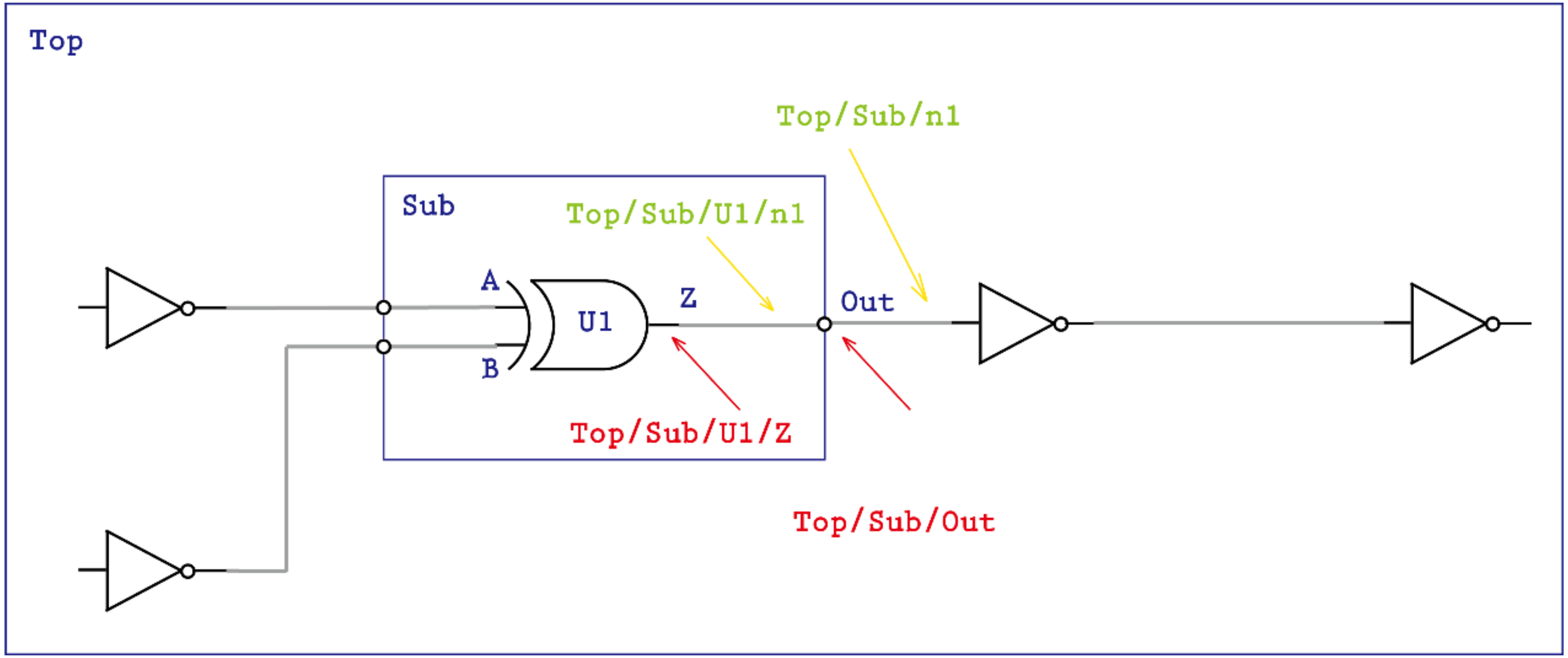


single fanout is pretty awesome, simplest structure

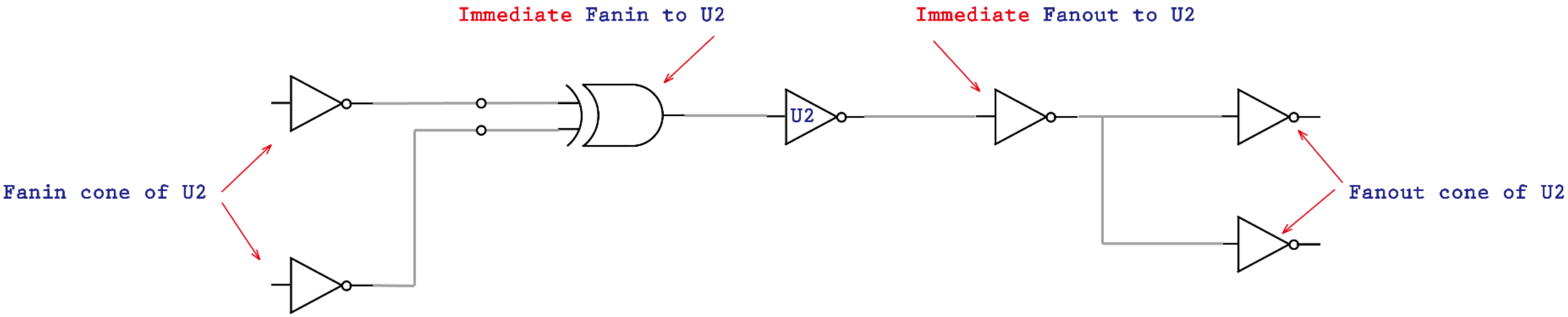


multiple fanout is Ok, may lead to large load on net

Hierarchical (cell/pin) v.s. Leaf (cell/pin)



Immediate fanin/fanout



Immediate Fanin to U2

Immediate Fanout to U2

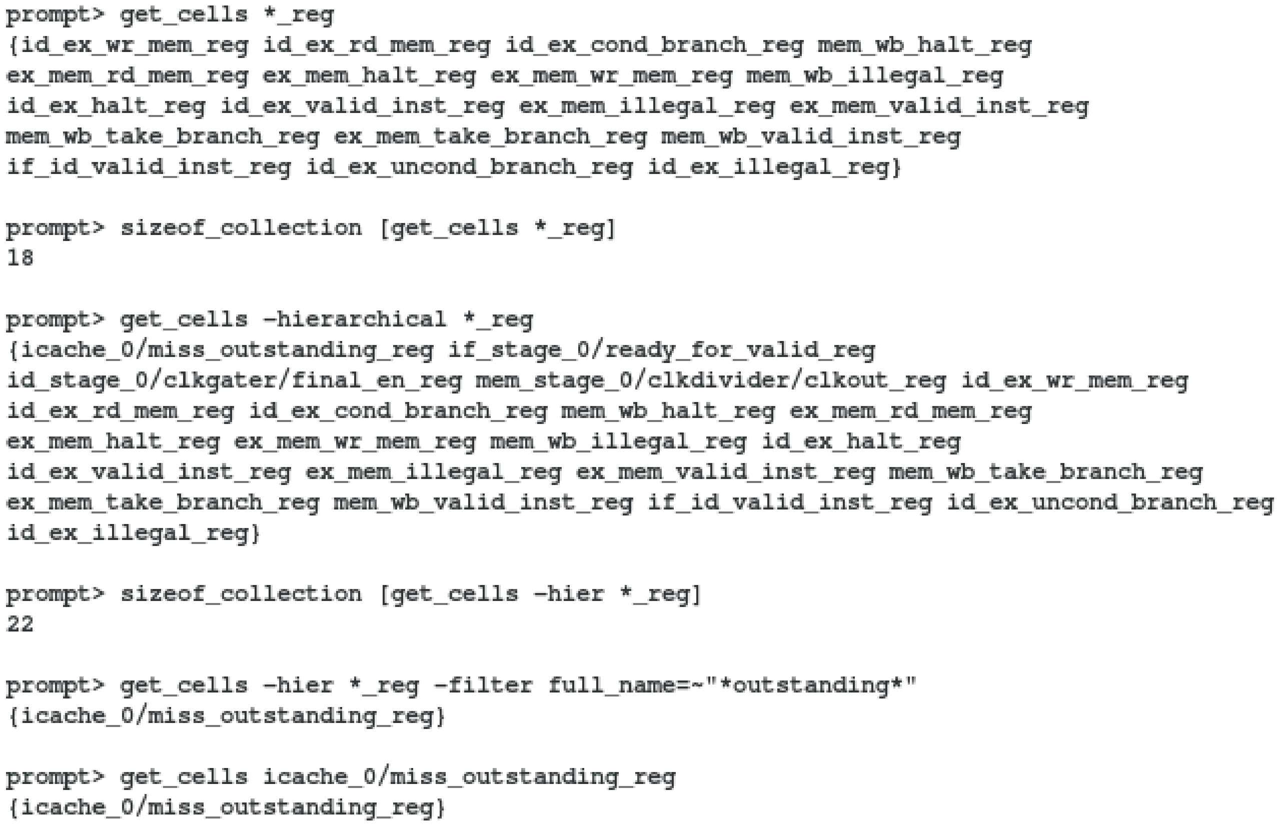
Fanin cone of U2

Fanout cone of U2

Basic Query Commands

get_cells

- ➢*Broaden search into all hierarchy with -hier option*
- ➢*Refine the results with -filter option*

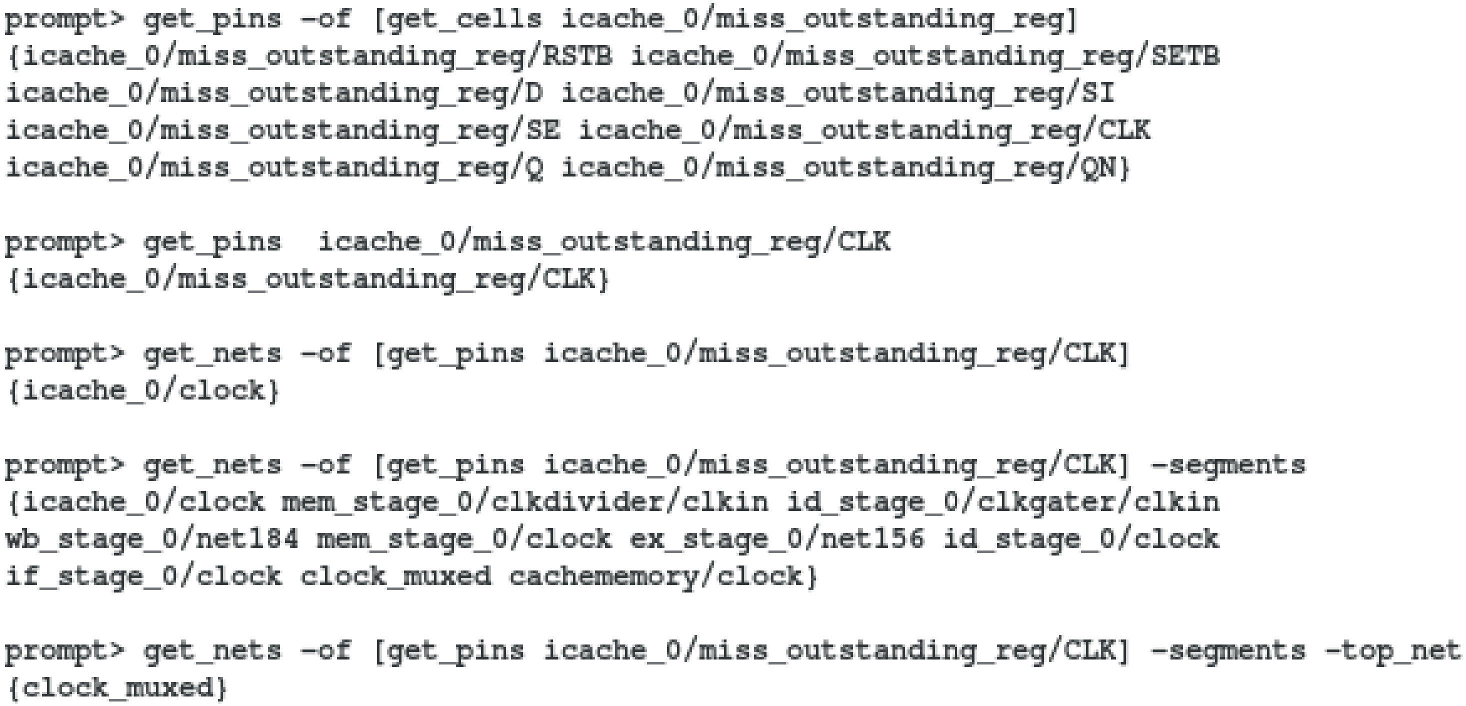


```
prompt> get_cells *_reg
{id_ex_wr_mem_reg id_ex_rd_mem_reg id_ex_cond_branch_reg mem_wb_halt_reg
ex_mem_rd_mem_reg ex_mem_halt_reg ex_mem_wr_mem_reg mem_wb_illegal_reg
id_ex_halt_reg id_ex_valid_inst_reg ex_mem_illegal_reg ex_mem_valid_inst_reg
mem_wb_take_branch_reg ex_mem_take_branch_reg mem_wb_valid_inst_reg
if_id_valid_inst_reg id_ex_uncond_branch_reg id_ex_illegal_reg}

prompt> sizeof_collection [get_cells *_reg]
18

prompt> get_cells -hierarchical *_reg
{icache_0/miss_outstanding_reg if_stage_0/ready_for_valid_reg
id_stage_0/clkgater/final_en_reg mem_stage_0/clkdivider/clkout_reg id_ex_wr_mem_reg
id_ex_rd_mem_reg id_ex_cond_branch_reg mem_wb_halt_reg ex_mem_rd_mem_reg
ex_mem_halt_reg ex_mem_wr_mem_reg mem_wb_illegal_reg id_ex_halt_reg
id_ex_valid_inst_reg ex_mem_illegal_reg ex_mem_valid_inst_reg mem_wb_take_branch_reg
ex_mem_take_branch_reg mem_wb_valid_inst_reg if_id_valid_inst_reg id_ex_uncond_branch_reg
id_ex_illegal_reg}

prompt> sizeof_collection [get_cells -hier *_reg]
22

prompt> get_cells -hier *_reg -filter full_name=~"*outstanding*"
{icache_0/miss_outstanding_reg}

prompt> get_cells icache_0/miss_outstanding_reg
{icache_0/miss_outstanding_reg}
```

Basic Query Commands

get_pins / get_nets

- *List same net with different hierarchical net name -segment*
- *Get top level net name -top_hierarchical_group*



```
prompt> get_pins -of [get_cells icache_0/miss_outstanding_reg]
{icache_0/miss_outstanding_reg/RSTB icache_0/miss_outstanding_reg/SETB
icache_0/miss_outstanding_reg/D icache_0/miss_outstanding_reg/SI
icache_0/miss_outstanding_reg/SE icache_0/miss_outstanding_reg/CLK
icache_0/miss_outstanding_reg/Q icache_0/miss_outstanding_reg/QN}

prompt> get_pins  icache_0/miss_outstanding_reg/CLK
{icache_0/miss_outstanding_reg/CLK}

prompt> get_nets -of [get_pins icache_0/miss_outstanding_reg/CLK]
{icache_0/clock}

prompt> get_nets -of [get_pins icache_0/miss_outstanding_reg/CLK] -segments
{icache_0/clock mem_stage_0/clkdivider/clkin id_stage_0/clkgater/clkin
wb_stage_0/net184 mem_stage_0/clock ex_stage_0/net156 id_stage_0/clock
if_stage_0/clock clock_muxed cachememory/clock}

prompt> get_nets -of [get_pins icache_0/miss_outstanding_reg/CLK] -segments -top_net
{clock_muxed}
```

Collections and Objects

- Think of a collection as a place to hold things.

- Let's say you want to manage 4 sheep named Larry, Moe, Curly and Shemp.

- You could just keep referring to them by their names, but that means that every time you wanted to bring them up in conversation, you'd have to say, "Larry, Moe, Curly and Shemp".

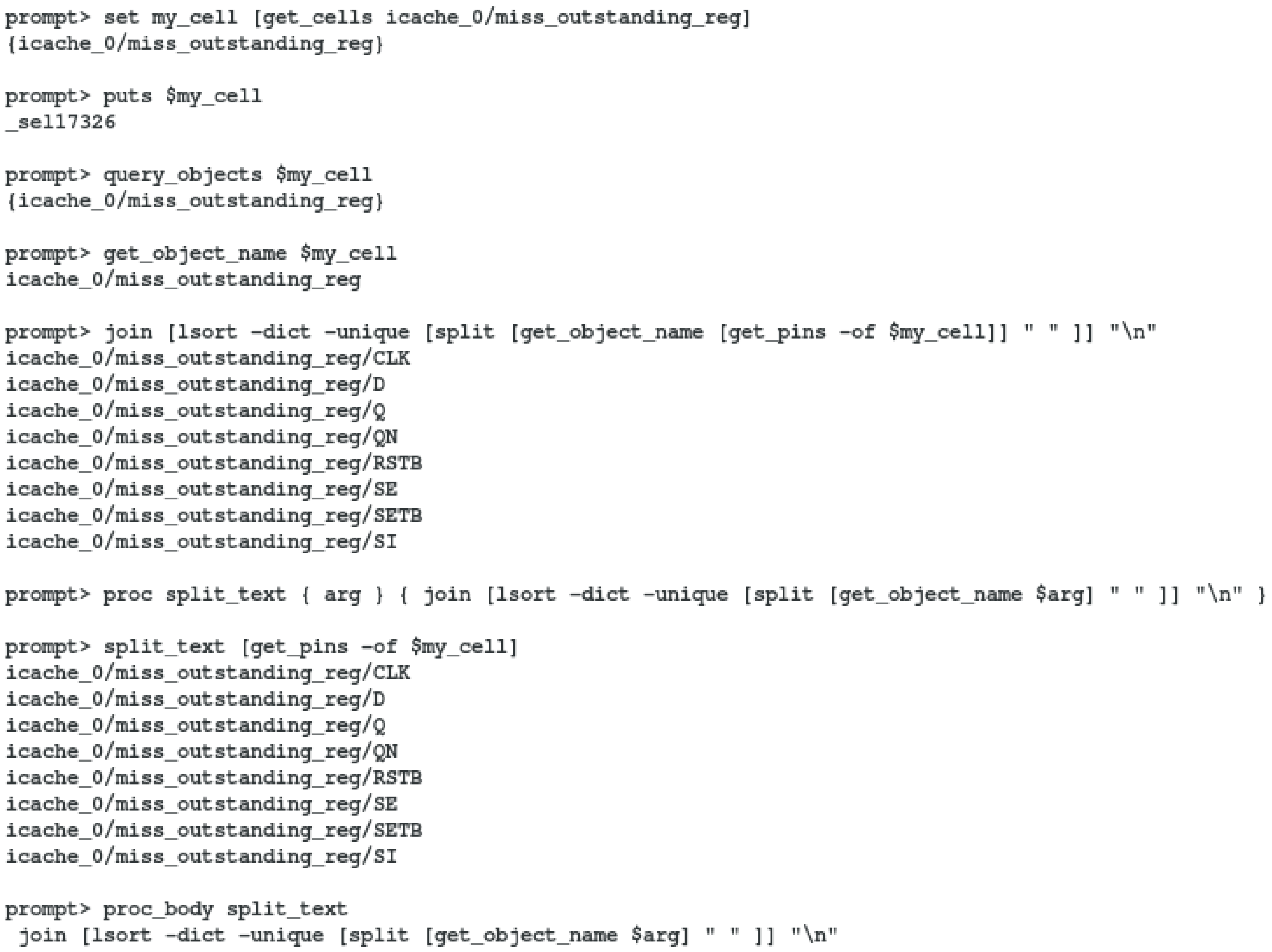
- An easier way to deal with the sheep would be refer to the herd, eg "MyHerd".

- So now rather then having to call each of the sheep out, you can just say, "MyHerd". MyHerd is a symbolic reference to the sheep, Larry, Moe, Curly and Shemp.

Collections and Objects

- This is how collections work in Tcl.
- Once you create a collection of objects, you need only refer to it by it's reference.
- As it happens this reference is called a handle.
- It is easier to keep track of this single handle than it is to keep passing around the individual names of the sheep.
- That said, collection is a group of objects referenced by a string identifier.
- There is a set of commands to create and manipulate collections.
  - Creating collection commands: find, all_inputs, all_outputs, etc., get_cells, get_nets, etc...
  - Manipulating collection commands: sizeof_collection, foreach_collection, add_to_collection, remove_from_collection, copy_collection, query_object

Synopsys Object

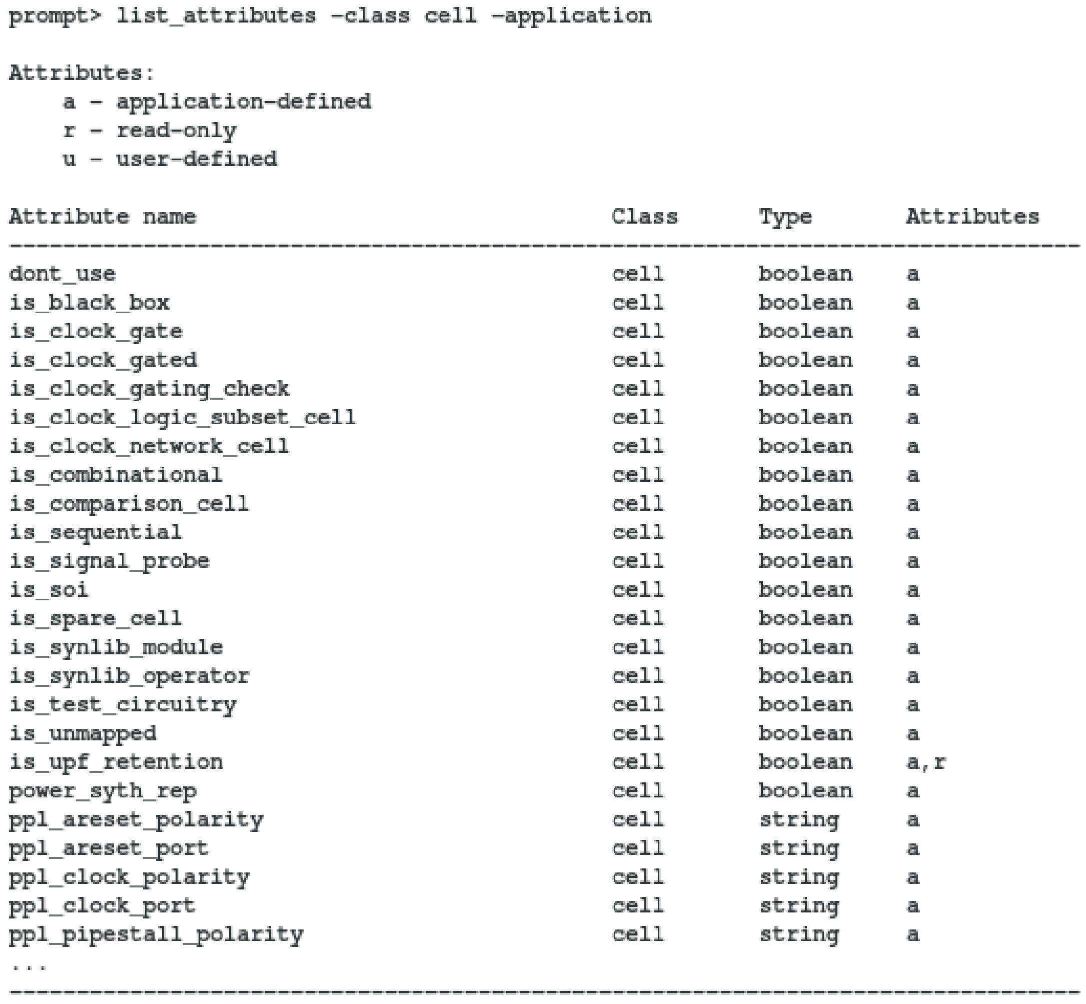


```
prompt> set my_cell [get_cells icache_0/miss_outstanding_reg]
{icache_0/miss_outstanding_reg}

prompt> puts $my_cell
_sel17326

prompt> query_objects $my_cell
{icache_0/miss_outstanding_reg}

prompt> get_object_name $my_cell
icache_0/miss_outstanding_reg

prompt> join [lsort -dict -unique [split [get_object_name [get_pins -of $my_cell]] " " ]] "\n"
icache_0/miss_outstanding_reg/CLK
icache_0/miss_outstanding_reg/D
icache_0/miss_outstanding_reg/Q
icache_0/miss_outstanding_reg/QN
icache_0/miss_outstanding_reg/RSTB
icache_0/miss_outstanding_reg/SE
icache_0/miss_outstanding_reg/SETB
icache_0/miss_outstanding_reg/SI

prompt> proc split_text { arg } { join [lsort -dict -unique [split [get_object_name $arg] " " ]] "\n" }

prompt> split_text [get_pins -of $my_cell]
icache_0/miss_outstanding_reg/CLK
icache_0/miss_outstanding_reg/D
icache_0/miss_outstanding_reg/Q
icache_0/miss_outstanding_reg/QN
icache_0/miss_outstanding_reg/RSTB
icache_0/miss_outstanding_reg/SE
icache_0/miss_outstanding_reg/SETB
icache_0/miss_outstanding_reg/SI

prompt> proc_body split_text
 join [lsort -dict -unique [split [get_object_name $arg] " " ]] "\n"
```

Basic Query Commands

list_attribute

- ➢*Find all available attribute of a class*

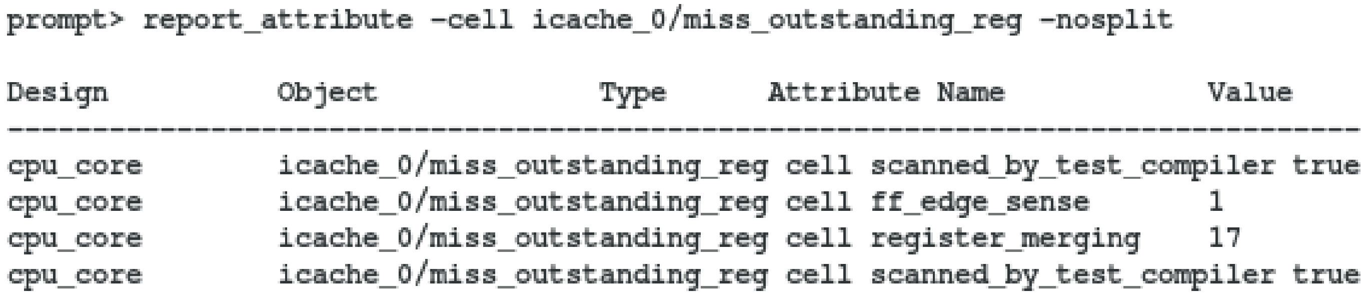


```
prompt> list_attributes -class cell -application

Attributes:
    a - application-defined
    r - read-only
    u - user-defined

Attribute name                              Class      Type       Attributes
--------------------------------------------------------------------------------
dont_use                                    cell       boolean    a
is_black_box                                cell       boolean    a
is_clock_gate                               cell       boolean    a
is_clock_gated                              cell       boolean    a
is_clock_gating_check                       cell       boolean    a
is_clock_logic_subset_cell                  cell       boolean    a
is_clock_network_cell                       cell       boolean    a
is_combinational                            cell       boolean    a
is_comparison_cell                          cell       boolean    a
is_sequential                               cell       boolean    a
is_signal_probe                             cell       boolean    a
is_soi                                      cell       boolean    a
is_spare_cell                               cell       boolean    a
is_synlib_module                            cell       boolean    a
is_synlib_operator                          cell       boolean    a
is_test_circuitry                           cell       boolean    a
is_unmapped                                 cell       boolean    a
is_upf_retention                            cell       boolean    a,r
power_syth_rep                              cell       boolean    a
ppl_areset_polarity                         cell       string     a
ppl_areset_port                             cell       string     a
ppl_clock_polarity                          cell       string     a
ppl_clock_port                              cell       string     a
ppl_pipestall_polarity                      cell       string     a
...
--------------------------------------------------------------------------------
```

Basic Query Commands

report_attribute

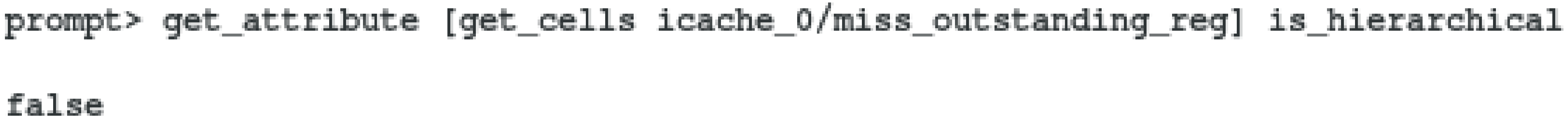
➢*Report active attribute on a cell*



```
prompt> report_attribute -cell icache_0/miss_outstanding_reg -nosplit

Design          Object                 Type      Attribute Name            Value
--------------------------------------------------------------------------------
cpu_core        icache_0/miss_outstanding_reg cell scanned_by_test_compiler true
cpu_core        icache_0/miss_outstanding_reg cell ff_edge_sense           1
cpu_core        icache_0/miss_outstanding_reg cell register_merging        17
cpu_core        icache_0/miss_outstanding_reg cell scanned_by_test_compiler true
```

get_attribute

➢*Get particular attribute of an object*

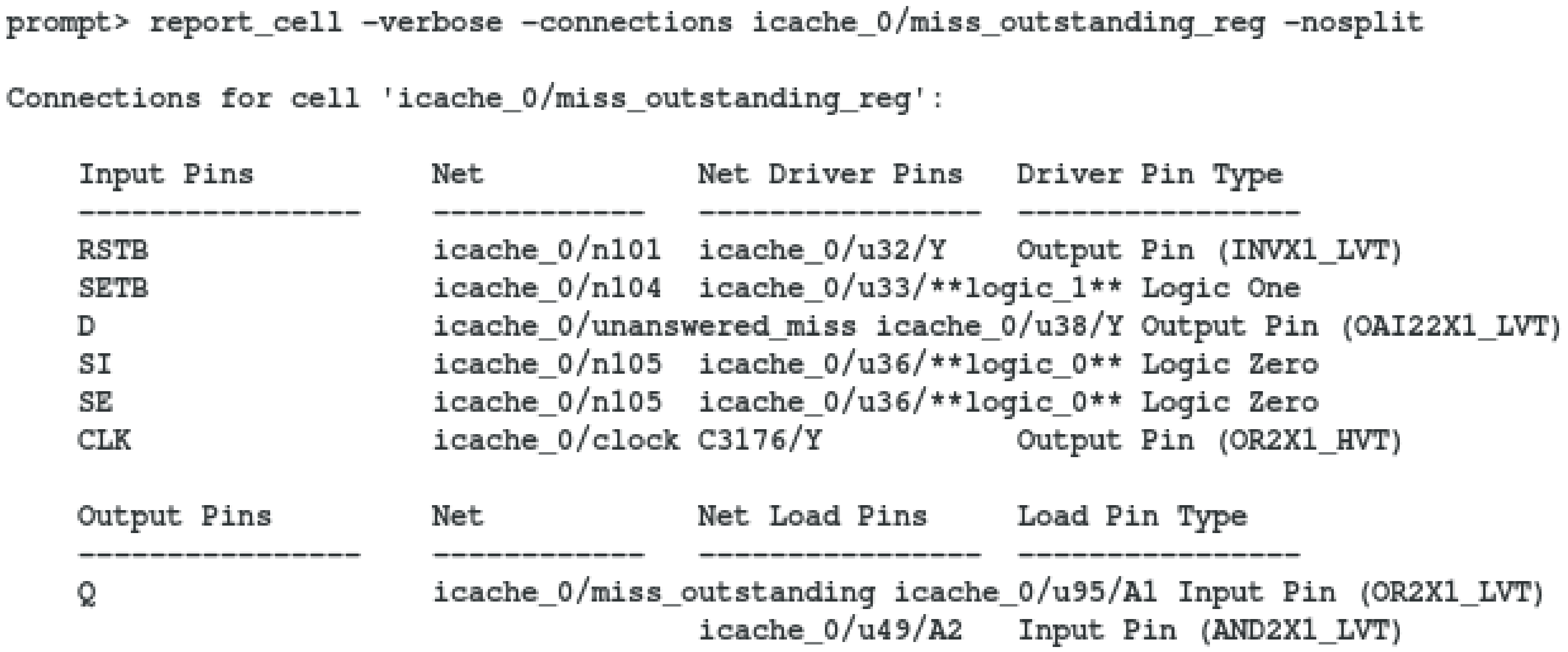


```
prompt> get_attribute [get_cells icache_0/miss_outstanding_reg] is_hierarchical

false
```

Basic Query Commands

report_cell

- ➢*Get a quick look of cell connectivity*

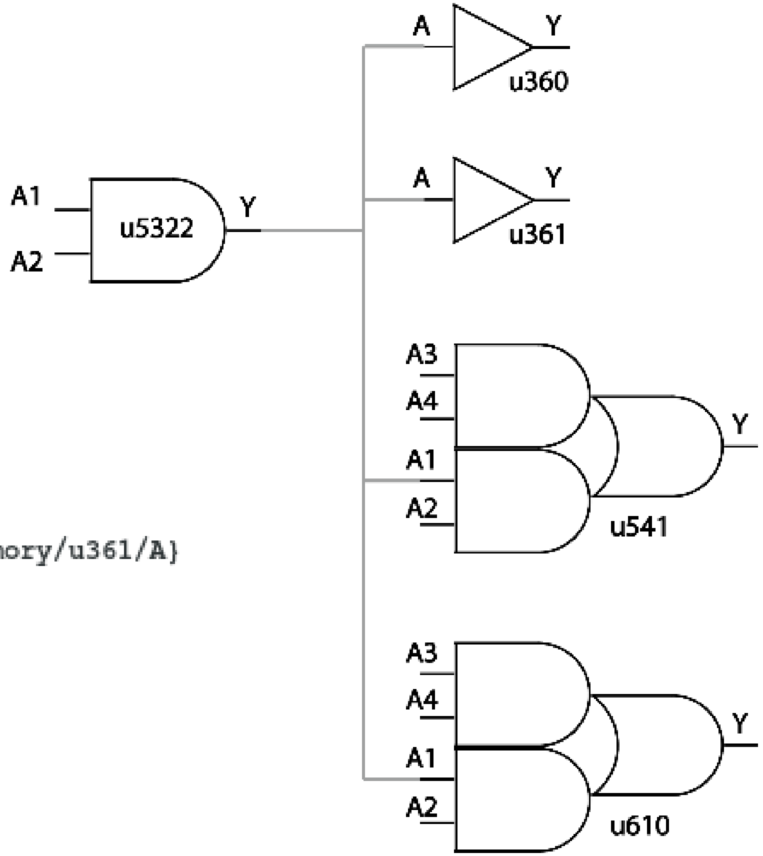


```
prompt> report_cell -verbose -connections icache_0/miss_outstanding_reg -nosplit

Connections for cell 'icache_0/miss_outstanding_reg':

    Input Pins          Net            Net Driver Pins   Driver Pin Type
    ----------------    ------------   ---------------   ----------------
    RSTB                icache_0/n101  icache_0/u32/Y    Output Pin (INVX1_LVT)
    SETB                icache_0/n104  icache_0/u33/**logic_1** Logic One
    D                   icache_0/unanswered_miss icache_0/u38/Y Output Pin (OAI22X1_LVT)
    SI                  icache_0/n105  icache_0/u36/**logic_0** Logic Zero
    SE                  icache_0/n105  icache_0/u36/**logic_0** Logic Zero
    CLK                 icache_0/clock C3176/Y           Output Pin (OR2X1_HVT)

    Output Pins         Net            Net Load Pins     Load Pin Type
    ----------------    ------------   ---------------   ----------------
    Q                   icache_0/miss_outstanding icache_0/u95/A1 Input Pin (OR2X1_LVT)
                                       icache_0/u49/A2   Input Pin (AND2X1_LVT)
```

Basic Tracing Commands

all_connected [<pin/net object>]

- ➢*Usually we only care about leaf cell, so get leaf cell with option –leaf*
- ➢*Argument must be a valid database object, not a plain text*
- ➢*Get immediate fanin/fanout*

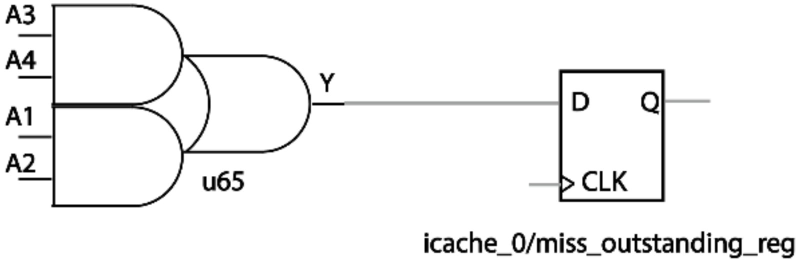


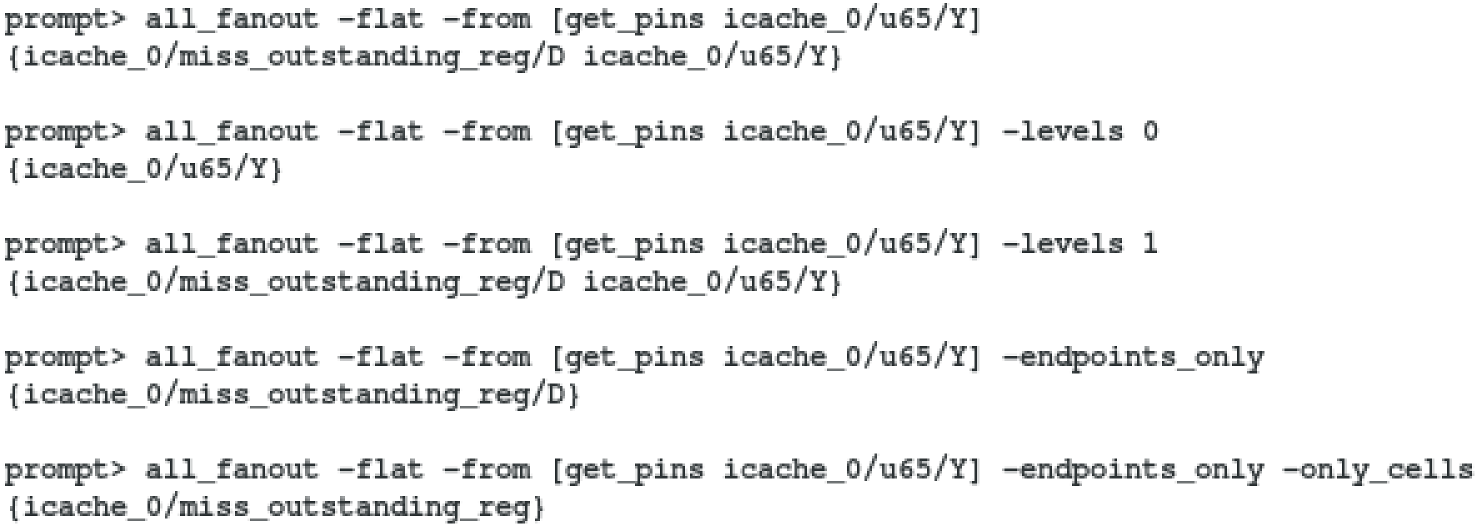
```
prompt> all_connected -leaf [get_nets cachememory/n323]
{cachememory/u5322/Y cachememory/u610/A1 cachememory/u541/A1 cachememory/u360/A cachememory/u361/A}

prompt> all_connected -leaf [get_pins cachememory/u5322/Y]
{cachememory/n323}
```

Basic tracing commands

all_fanout -flat -from

- *Trace entire fanout-cone of particular pin*
- *Refine results with –endpoints_only, -cell_only option*



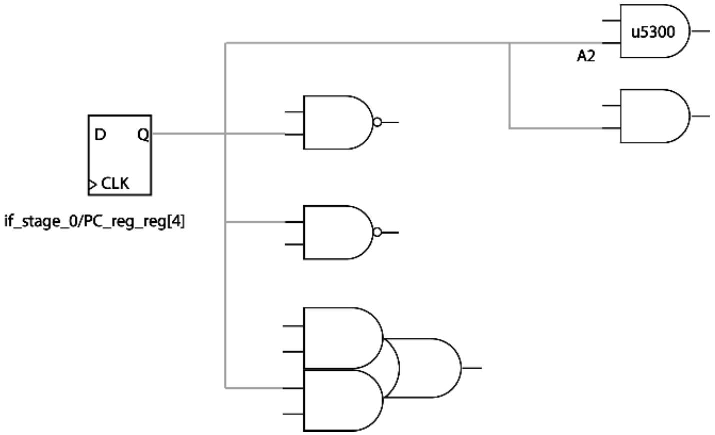


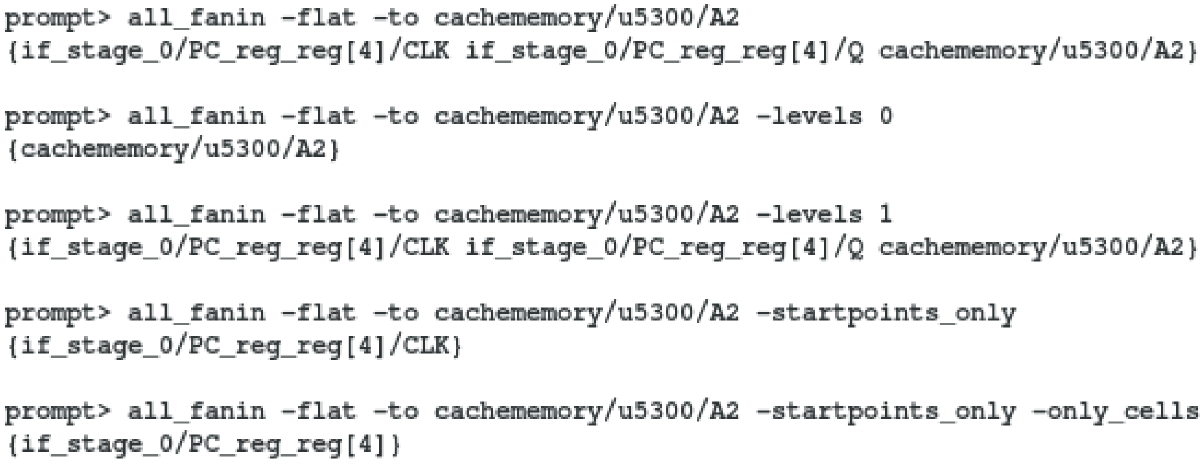
```
prompt> all_fanout -flat -from [get_pins icache_0/u65/Y]
{icache_0/miss_outstanding_reg/D icache_0/u65/Y}

prompt> all_fanout -flat -from [get_pins icache_0/u65/Y] -levels 0
{icache_0/u65/Y}

prompt> all_fanout -flat -from [get_pins icache_0/u65/Y] -levels 1
{icache_0/miss_outstanding_reg/D icache_0/u65/Y}

prompt> all_fanout -flat -from [get_pins icache_0/u65/Y] -endpoints_only
{icache_0/miss_outstanding_reg/D}

prompt> all_fanout -flat -from [get_pins icache_0/u65/Y] -endpoints_only -only_cells
{icache_0/miss_outstanding_reg}
```

Basic tracing commands

all_fanin -flat -to

- ➢ *Trace entire fanin-cone of particular pin*
- ➢ *Refine results with –startpoints_only, -cell_only option*





```
prompt> all_fanin -flat -to cachememory/u5300/A2
{if_stage_0/PC_reg_reg[4]/CLK if_stage_0/PC_reg_reg[4]/Q cachememory/u5300/A2}

prompt> all_fanin -flat -to cachememory/u5300/A2 -levels 0
{cachememory/u5300/A2}

prompt> all_fanin -flat -to cachememory/u5300/A2 -levels 1
{if_stage_0/PC_reg_reg[4]/CLK if_stage_0/PC_reg_reg[4]/Q cachememory/u5300/A2}

prompt> all_fanin -flat -to cachememory/u5300/A2 -startpoints_only
{if_stage_0/PC_reg_reg[4]/CLK}

prompt> all_fanin -flat -to cachememory/u5300/A2 -startpoints_only -only_cells
{if_stage_0/PC_reg_reg[4]}
```

1. Find out all the nets present in the design having length more than 200 micron.

```
foreach_in_collection net [get_flat_nets *] {
set dr_len [get_attribute [get_nets $net] dr_length]
if {$dr_len > “300”} {
echo “[get_object_name $net] $dr_len” >> nets_drLength.rpt
}
}
```

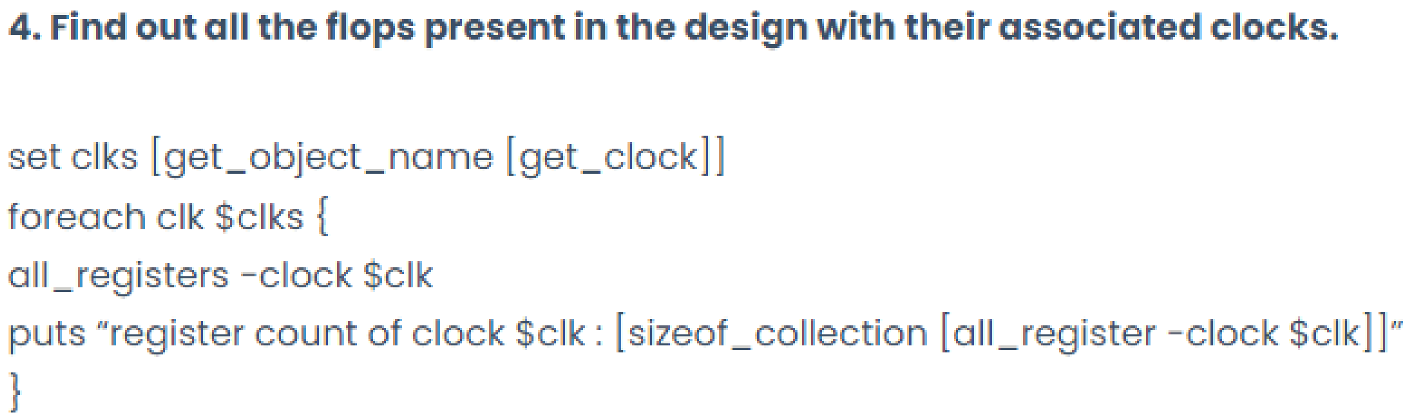
2. Find out the ports present in the design with direction as INPUT.

```
set OFILE [open ports_in.tcl “w”]
set ports “”

foreach_in_collection port [get_ports * -filter direction==in] {
puts $OFILE [get_object_name [get_ports $port]]
}
close $OFILE
```

3. Find out the nets present in the design which are data and clock and dump into a file. You may tweak the script according to your need.

```
set nets *

foreach net $nets {

set net_type [get_attribute [get_nets $net] net_type]
if {$net_type == “clock”} {
echo “[get_object_name $net] $net_type” >> nets_clk.rpt
} else {
echo “[get_object_name $net] $net_type” >> nets_data.rpt
}
}
```

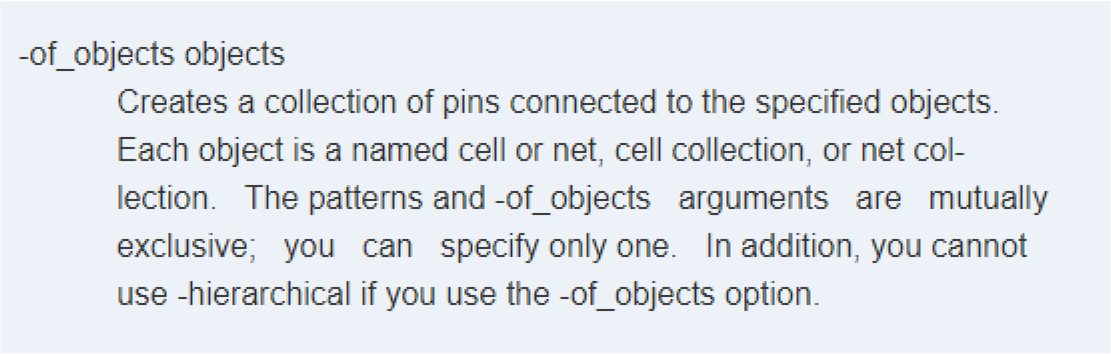
List of register sinks for a clock



4. Find out all the flops present in the design with their associated clocks.

```
set clks [get_object_name [get_clock]]
foreach clk $clks {
all_registers -clock $clk
puts “register count of clock $clk : [sizeof_collection [all_register -clock $clk]]”
}
```

Find high fanout nets

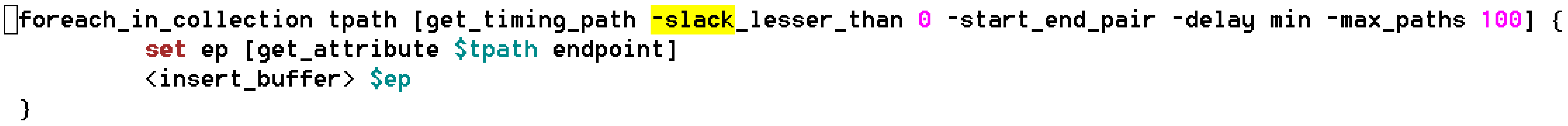


-of_objects objects

Creates a collection of pins connected to the specified objects. Each object is a named cell or net, cell collection, or net collection. The patterns and -of_objects arguments are mutually exclusive; you can specify only one. In addition, you cannot use -hierarchical if you use the -of_objects option.

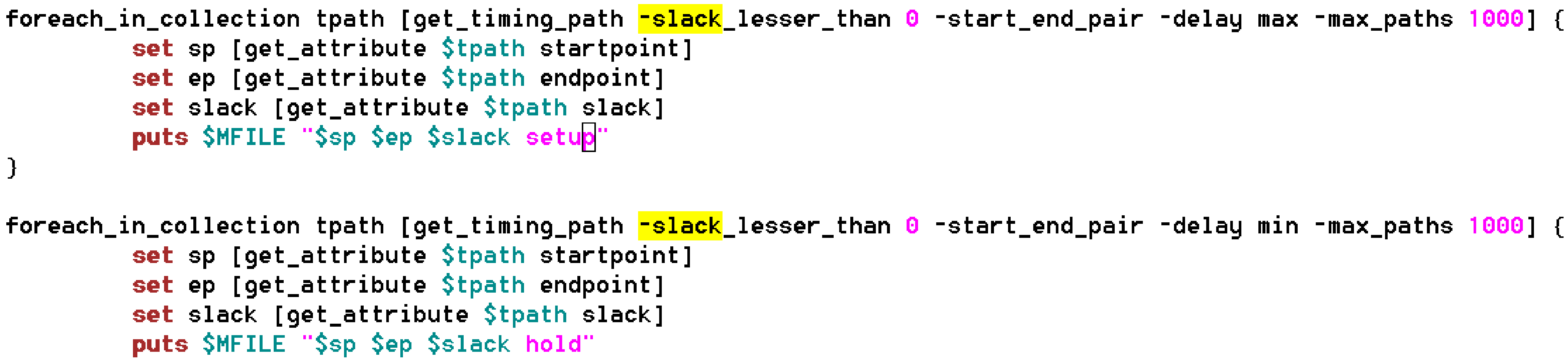
```
set all_nets [get_net -hier -top_net_of_hierarchical_group]
set total_high_fanout_nets 0

echo -n “Searching [sizeof_collection $all_nets] nets “
echo    “looking for fanouts > $HFN_FANOUT_THRESHOLD”

foreach_in_collection the_net $all_nets {

  set fanout [sizeof_collection \
                  [filter \
                       [get_pins -quiet -leaf -of_objects $the_net] \
                       “pin_direction!=out”]]

  if {$fanout > $HFN_FANOUT_THRESHOLD} {
```

Insert buffer on all endpoints having hold violation

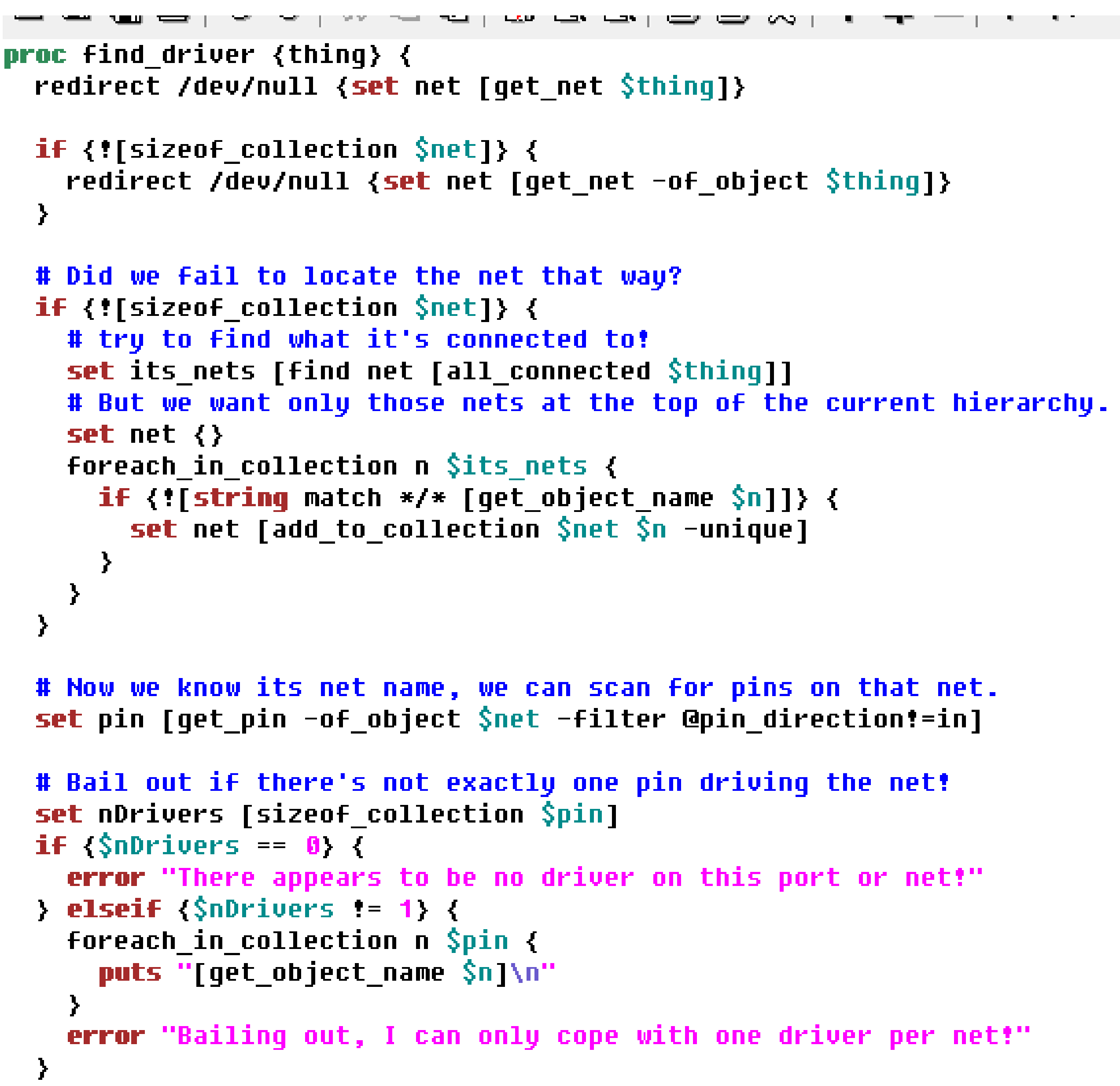


```
foreach_in_collection tpath [get_timing_path -slack_lesser_than 0 -start_end_pair -delay min -max_paths 100] {
         set ep [get_attribute $tpath endpoint]
         <insert_buffer> $ep
}
```

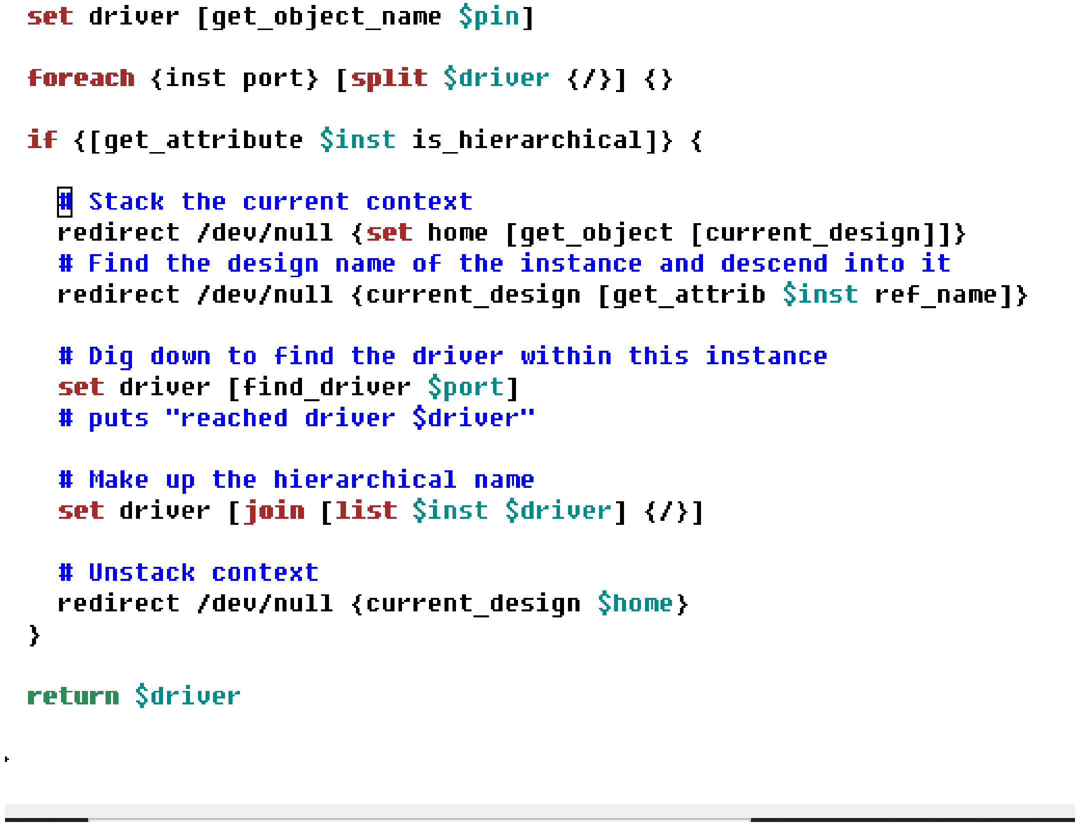
Report endpoints and startpoints slack of top 1000 failing paths



```
foreach_in_collection tpath [get_timing_path -slack_lesser_than 0 -start_end_pair -delay max -max_paths 1000] {
         set sp [get_attribute $tpath startpoint]
         set ep [get_attribute $tpath endpoint]
         set slack [get_attribute $tpath slack]
         puts $MFILE "$sp $ep $slack setup"
}

foreach_in_collection tpath [get_timing_path -slack_lesser_than 0 -start_end_pair -delay min -max_paths 1000] {
         set sp [get_attribute $tpath startpoint]
         set ep [get_attribute $tpath endpoint]
         set slack [get_attribute $tpath slack]
         puts $MFILE "$sp $ep $slack hold"
```



```
proc find_driver {thing} {
  redirect /dev/null {set net [get_net $thing]}

  if {![sizeof_collection $net]} {
    redirect /dev/null {set net [get_net -of_object $thing]}
  }

  # Did we fail to locate the net that way?
  if {![sizeof_collection $net]} {
    # try to find what it's connected to!
    set its_nets [find net [all_connected $thing]]
    # But we want only those nets at the top of the current hierarchy.
    set net {}
    foreach_in_collection n $its_nets {
      if {![string match */* [get_object_name $n]]} {
        set net [add_to_collection $net $n -unique]
      }
    }
  }

  # Now we know its net name, we can scan for pins on that net.
  set pin [get_pin -of_object $net -filter @pin_direction!=in]

  # Bail out if there's not exactly one pin driving the net!
  set nDrivers [sizeof_collection $pin]
  if {$nDrivers == 0} {
    error "There appears to be no driver on this port or net!"
  } elseif {$nDrivers != 1} {
    foreach_in_collection n $pin {
      puts "[get_object_name $n]\n"
    }
    error "Bailing out, I can only cope with one driver per net!"
  }
```



```
set driver [get_object_name $pin]

foreach {inst port} [split $driver {/}] {}

if {[get_attribute $inst is_hierarchical]} {

  # Stack the current context
  redirect /dev/null {set home [get_object [current_design]]}
  # Find the design name of the instance and descend into it
  redirect /dev/null {current_design [get_attrib $inst ref_name]}

  # Dig down to find the driver within this instance
  set driver [find_driver $port]
  # puts "reached driver $driver"

  # Make up the hierarchical name
  set driver [join [list $inst $driver] {/}]

  # Unstack context
  redirect /dev/null {current_design $home}
}

return $driver
```