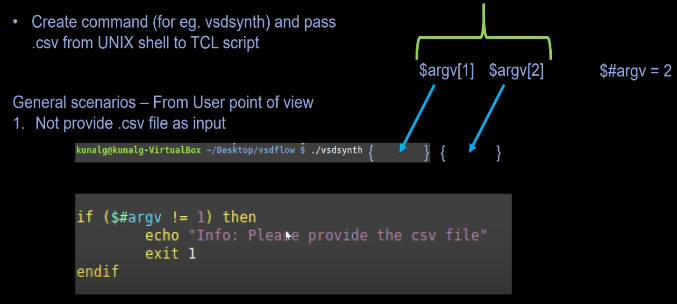
**TCL for synthesis scripting in VLSI Design**

**Tool Command Language (TCL)** is a high-level scripting language widely used for electronic design automation (EDA) for scripting tasks in tools like Synopsys and Cadence.

**Module 1: Introduction to TCL and VSDSYNTH Toolbox Usage**

Task which involves building a TCL box which takes a csv file as an input and gives a timing results as an output.

1. Sub-task is to create a command (for example, vsdsynth) and pass .csv files from UNIX shell to TCL script.



1. The next sub-task involves converting all inputs to format [1]and SDC format,
2. Then passing them to the synthesis tool 'Yosys'. Yosys tool cannot understand the csv file so we need to convert it into foramt [1]
3. Convert openMSP430\_design\_constraints.csv into the sdc format which can be passed to Yosys tool for synthesis.
4. Convert format [1] and SDC to format [2] and pass them to the timing tool 'Opentimer'. foramt[2] can be understandable by Opentimer tool for STA.
5. Final sub-task is to generate an output report with the timing results as shown below.

## Module 2: Variable Creation and Processing Constraints from CSV(Subtask 2 highlighted in Module1 )

## Sub-task 2: Converts .csv file into format[1] and constraints.csv file into sdc format

1. Create the variables using tcl script by reading the first column elements of openMSP430\_design\_details.csv file like DesingName, OutputDirectory etc...

*Converting csv file into matrix*

set filename [lindex $argv 0]

package require csv

package require struct::matrix

struct::matrix m

set f [open $filename]

csv::read2matrix $f m , auto

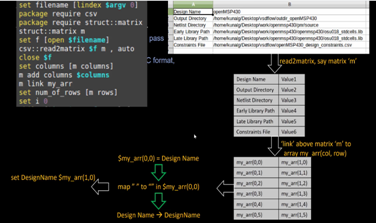
close $f

set columns [m columns]

#m add columns $columns

m link my\_arr

set rows [m rows]



1. Check if the files/directory provided by user in a csv file ( second column) does exists or not.

if { [file isdirectory $OutputDirectory]} {

puts "\nInfo : Output directory exists and found in path $OutputDirectory "

} else {

puts "\n Info : Cannot find the output directory $OutputDirectory. Creating $OutputDirectory"

file mkdir $OutputDirectory

}

if { [file isdirectory $NetlistDirectory]} {

puts "\nInfo : Netlist directory exists and found in path $NetlistDirectory "

} else {

puts "\n Info : Cannot find the Netlist directory $NetlistDirectory. Creating $NetlistDirectory"

file mkdir $NetlistDirectory

}

1. Read the constraints file for above and convert it into a SDC format which is acceptable for synthesis and PNR purposes.
2. Read all the files from "NetlistDirectory" i.e. all verilog files and write it in script for Yosys.
3. Lastly, create a synthesis script and pass it to Yosys tool.

## Module 3 Processing clock and input constraints (Subtask 3 highlighted in Module2)

## Clock and input/output constraints in csv file processed into the Synopsys Design Constraints (SDC) format which can be used for synthesis.

## The below mentioned script creates a .sdc file in OutputDirectory and writes the clock parameters into .sdc file in a particular manner as mentioned below.

set sdc\_file [open $OutputDirectory/$DesignName.sdc "w"]

set i [expr {$clock\_start+1}]

set end\_of\_ports [expr {$input\_ports\_start -1}]

puts "\n Info-SDC: Working on clock constraints..."

while {$i< $end\_of\_ports} {

puts "Working on clock [constraints get cell 0 $i ]"

puts -nonewline $sdc\_file "\ncreate\_clock -name [constraints get cell 0 $i ] -period [constraints get cell 1 $i ] -waveform \{0 [expr { [constraints get cell 1 $i ]\*[constraints get cell 2 $i ]/100 }]\} \[get\_ports [constraints get cell 0 $i ]\]"

puts -nonewline $sdc\_file "\nset\_clock\_latency -source -early -rise [constraints get cell $clock\_early\_rise\_delay\_start $i ] \[get\_clocks [constraints get cell 0 $i]\]"

puts -nonewline $sdc\_file "\nset\_clock\_latency -source -early -fail [constraints get cell $clock\_early\_fall\_delay\_start $i ] \[get\_clocks [constraints get cell 0 $i]\]"

puts -nonewline $sdc\_file "\nset\_clock\_latency -source -late -rise [constraints get cell $clock\_late\_rise\_delay\_start $i ] \[get\_clocks [constraints get cell 0 $i]\]"

puts -nonewline $sdc\_file "\nset\_clock\_latency -source -late -fail [constraints get cell $clock\_late\_fall\_delay\_start $i ] \[get\_clocks [constraints get cell 0 $i]\]"

puts -nonewline $sdc\_file "\nset\_clock\_transition -rise -min [constraints get cell $clock\_early\_rise\_slew\_start $i ] \[get\_clocks [constraints get cell 0 $i]\]"

puts -nonewline $sdc\_file "\nset\_clock\_transition -fall -min [constraints get cell $clock\_early\_fall\_slew\_start $i ] \[get\_clocks [constraints get cell 0 $i]\]"

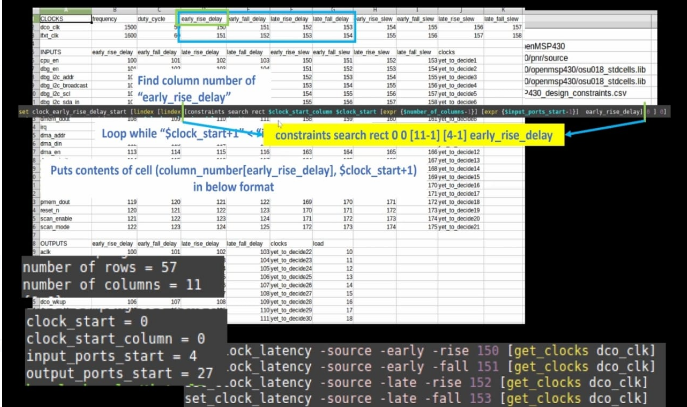
puts -nonewline $sdc\_file "\nset\_clock\_transition -rise -max [constraints get cell $clock\_late\_rise\_slew\_start $i ] \[get\_clocks [constraints get cell 0 $i]\]"

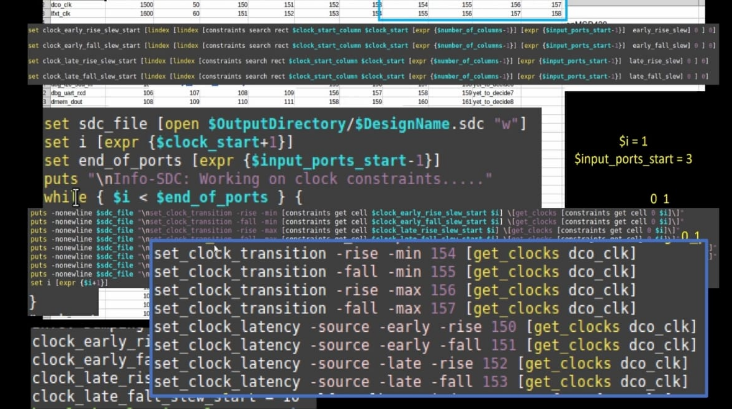
puts -nonewline $sdc\_file "\nset\_clock\_transition -fall -max [constraints get cell $clock\_late\_fall\_slew\_start $i ] \[get\_clocks [constraints get cell 0 $i]\]"

set i [expr {$i+1}]

}

## 





Some of the input port have multiple spaces in between them. The below mentioned script removes the spaces in between them and writes them in a temporary file.

set i [expr {$input\_ports\_start+1}]

set end\_of\_ports [expr {$output\_ports\_start-1}]

puts "\nInfo-SDC: Working on IO constraints....."

puts "\nInfo-SDC: Categorizing input ports as bits and bussed"

while { $i < $end\_of\_ports } {

set netlist [glob -dir $NetlistDirectory \*.v]

set tmp\_file [open /tmp/1 w]

foreach f $netlist {

set fd [open $f]

#puts "reading file $f"

while {[gets $fd line] != -1} {

set pattern1 " [constraints get cell 0 $i];"

if {[regexp -all -- $pattern1 $line]} {

#puts "\npattern1 \"$pattern1\" found and matching line in verilog file \"$f\" is \"$line\""

set pattern2 [lindex [split $line ";"] 0]

#puts "\ncreating pattern2 by splitting pattern1 using semi-colon as delimiter => \"$pattern2\""

if {[regexp -all {input} [lindex [split $pattern2 "\S+"] 0]]} {

#puts "\nout of all patterns, \"$pattern2\" has matching string \"input\". So preserving this line and ignoring others"

set s1 "[lindex [split $pattern2 "\S+"] 0] [lindex [split $pattern2 "\S+"] 1] [lindex [split $pattern2 "\S+"] 2]"

#puts "\nprinting first 3 elements of pattern as \"$s1\" using space as delimiter"

puts -nonewline $tmp\_file "\n[regsub -all {\s+} $s1 " "]"

#puts "\nreplace multiple spaces in s1 by space and reformat as \"[regsub -all {\s+} $s1 " "]\""

}

#else { " \"$pattern2\" didnt have first term as 'output'"}

}

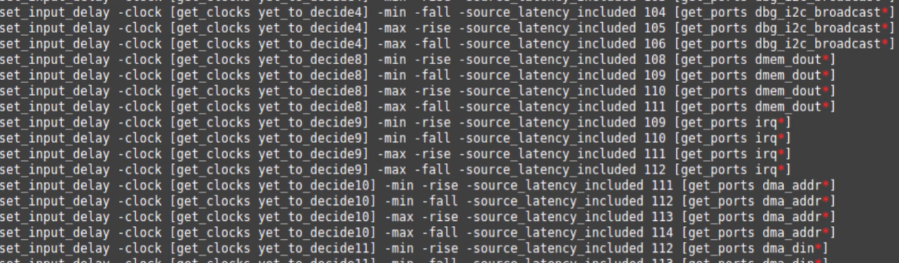
}

close $fd

}

close $tmp\_file

Input ports are bussed and some are not. But from the cosntraitns.csv file we can't differentiate which ports are bussed or which are not. Need to expand the bussed ports into single ports for understanding purposes for Opentimer tool. If it is a bus we need to append a **"\*"** at the end of the port name in the .sdc file as shown below.



Same process for output constraints into sdc format.

## Module 4: Complete Scripting and Yosys Synthesis Introduction

#### Creating Scripts for Hierarchy check

## The script mentioned below creates a .heir.ys and writes the script for the hierarchy check.

puts "\n Info: Creating hierarchy check script to be used by Yosys"

set data "read\_liberty -lib -ignore\_miss\_dir -setattr blackbox ${LateLibraryPath}"

#puts "data is \"$data\""

set filename "$DesignName.hier.ys"

#puts "\nfilename is \"$filename\""

set fileId [open $OutputDirectory/$filename "w"]

#puts "open \"$OutputDirectory/$filename"\ in write mode"

puts -nonewline $fileId $data

#puts "netlist is \"$netlist\""

set netlist [glob -dir $NetlistDirectory \*.v]

foreach f $netlist {

set data $f

#puts "data is \"$f\""

puts -nonewline $fileId "\n read\_verilog $f"

}

puts -nonewline $fileId "\nhierarchy -check"

close $fileId

Whenever **"exec"** command uses in the tcl script, it runs the command in terminal.

Running the yosys tool by passing **"openMSP430.heir.ys"** as input file and catches all the logs in **"openMSP430.hierarchy\_check.log"**

If there is an error, **"my\_error"** will be set to 1 and we have to find the error. In yosys when error occurs, then we will find a common pattern such as **"referenced in module"**. It differs across various tools. We have to search each lines of .log file and prints the error statement. The script mentioned below does the above purpose. If the hierarchy check passes then it displays a message saying **"Hierarchy check PASS".**

set my\_error [catch { exec yosys -s $OutputDirectory/$DesignName.hier.ys >& $OutputDirectory/$DesignName.hierarchy\_check.log} msg]

puts "Error flag is \"$my\_error\""

if { $my\_error } {

set filename "$OutputDirectory/$DesignName.hierarchy\_check.log"

puts "log file name is \"$filename\" "

set pattern "referenced in module"

#puts "pattern is $pattern"

set count 0

set fid [open $filename r]

while {[gets $fid line] != -1} {

# -- used to say end of command options. everything after this is args

incr count [regexp -all -- $pattern $line]

if {[regexp -all -- $pattern $line]} {

puts "\nError: module [lindex $line 2] is not part of design $DesignName. Please correct RTL in the path '$NetlistDirectory'"

puts "\nInfo: Hierarchy check FAIL"

}

}

close $fid

} else {

puts "\nInfo: Hierarchy check PASS"

}

puts "\n Info: Please find the hierearchy check details in [file normalize $OutputDirectory/$DesignName.hierarchy.check.log] for more info"

#### Module 5: Advanced Scripting Techniques and Quality of Results Generation

#### Develop script for synthesis and run yosys tool (Subtask 4 &5 highlighted in Module2).

puts "\nInfo: Creating main synthesis script to be used for yosys"

set data "read\_liberty -lib -ignore\_miss\_dir -setattr blackbox ${LateLibraryPath}"

set filename "$DesignName.ys"

#puts "\nfilename is \"$filename\""

set fileId [open $OutputDirectory/$filename "w"]

#puts "open \"$OutputDirectory/$filename\" in write mode"

puts -nonewline $fileId $data

#puts "netlist is \"$netlist\""

set netlist [glob -dir $NetlistDirectory \*.v]

foreach f $netlist {

set data $f

#puts "data is \"$f\""

puts -nonewline $fileId "\n read\_verilog $f"

}

puts -nonewline $fileId "\nhierarchy -top $DesignName"

puts -nonewline $fileId "\nsynth -top $DesignName"

puts -nonewline $fileId "\nsplitnets -ports -format \_\_\ndfflibmap -liberty ${LateLibraryPath}\nopt"

puts -nonewline $fileId "\nabc -liberty ${LateLibraryPath}"

puts -nonewline $fileId "\nflatten"

puts -nonewline $fileId "\nclean -purge\niopadmap -outpad BUFX2 A:Y -bits\nopt \nclean"

puts -nonewline $fileId "\nwrite\_verilog $OutputDirectory/$DesignName.synth.v"

close $fileId

puts "\nInfo: Synthesis script created and can be accesed from the path $OutputDirectory/$filename"

puts "\nInfo: Running Synthesis...."

if {[catch { exec yosys -s $OutputDirectory/$DesignName.ys >& $OutputDirectory/$DesignName.synthesis.log} msg]} {

puts "\nError: Syntesis failed due to errors. Please refer to log $OutputDirectory/$DesignName.synthesis.log for errors"

exit

} else {

puts "\nInfo: Synthesis finished sucessfully"

}

puts "\nInfo: Please refert olog $OutputDirectory/$DesignName.synthesis.log"

The above script creates a **"openMSP430.ys"** which can be passed to Yosys for synthesis purpose. It writes all the netlist and all scripts necessary for synthesis into **"openMSP430.ys".**

By using **"exec"** command, yosys runs via tcl command and all the logs are stored in openMSP430.synthesis.log. If there is an error, it displays a message "Error: Synthesis failed due to errors. Refer to log /home/vsduser/vsdsynth/outdir\_openMSP430/$openMSP430.synthesis.log for errors".

#### Procs

Procedures(procs) such as **reopenStdout.proc, set\_num\_threads.proc, read\_lib.proc, read\_verilog.proc, read\_sdc.proc** to convert the format[1] and sdc format to format[2].

The procedure has three main options:

* -late: Specifies the path to the late library file used for setup timing analysis
* -early: Specifies the path to the early library file used for hold timing analysis
* -help: Displays usage information for the procedure

Convert create\_clock constraints into format [2] which can be understandable by Opentimer tool.

It searches a pattern

**create\_clock** and gets the clock\_port\_name, clock\_period and calculates duty cycle. And then writes all the above values in /tmp/2 file.

**set\_clock\_transition** and gets all the parameters. Then writes all the above values in /tmp/2 file which is further written in .timings file.

**set\_input\_delay** and gets all the parameters. Then writes all the above values in /tmp/2 file which is furher written in .timings file.

**set\_input\_transition** and then gets all the parameters and convert into Opentimer format. Then writes all the above values in /tmp/2 file.

**set\_output\_delay** and then gets all the parameters and convert into Opentimer format. Then writes all the above values in /tmp/2 file.

#### Creating scripts for Opentimer

if {$enable\_prelayout\_timing == 1} {

puts "\nInfo: enable prelayout\_timing is $enable\_prelayout\_timing. Enabling zero-wire load parasitics"

set spef\_file [open $OutputDirectory/$DesignName.spef w]

puts $spef\_file "\*SPEF \"IEEE 1481-1998\""

puts $spef\_file "\*DESIGN \"$DesignName\""

puts $spef\_file "\*DATE \"Sun Jun 11 11:59:00 2023\""

puts $spef\_file "\*VENDOR \"VLSI System Design\""

puts $spef\_file "\*PROGRAM \"TCL Workshop\""

puts $spef\_file "\*DATE \"0.0\""

puts $spef\_file "\*DESIGN FLOW \"NETLIST\_TYPE\_VERILOG\""

puts $spef\_file "\*DIVIDER /"

puts $spef\_file "\*DELIMITER : "

puts $spef\_file "\*BUS\_DELIMITER [ ]"

puts $spef\_file "\*T\_UNIT 1 PS"

puts $spef\_file "\*C\_UNIT 1 FF"

puts $spef\_file "\*R\_UNIT 1 KOHM"

puts $spef\_file "\*L\_UNIT 1 UH"

}

close $spef\_file

#### Quality of results (QOR) generation algorithm

This is the final sub-task which involves output generation as a datasheet.

set time\_elapsed\_in\_us [time {exec /home/vsduser/OpenTimer-1.0.5/bin/OpenTimer < $OutputDirectory/$DesignName.conf >& $OutputDirectory/$DesignName.results} ]

set time\_elapsed\_in\_sec "[expr {[lindex $time\_elapsed\_in\_us 0]/100000}]sec"

puts "\nInfo: STA finished in $time\_elapsed\_in\_sec seconds"

puts "\nInfo: Refer to $OutputDirectory/$DesignName.results for warings and errors"

The above script is used to executed the Opentimer tool by passing openMSP430.conf file as an input the results are stored in openMSP430.results. It also stores the time elapsed during STA in microseconds and seconds.

puts "\n"

puts " \*\*\*\*PRELAYOUT TIMING RESULTS\*\*\*\* "

set formatStr "%15s %15s %15s %15s %15s %15s %15s %15s %15s"

puts [format $formatStr "----------" "-------" "--------------" "---------" "---------" "--------" "--------" "-------" "-------"]

puts [format $formatStr "DesignName" "Runtime" "Instance Count" "WNS Setup" "FEP Setup" "WNS Hold" "FEP Hold" "WNS RAT" "FEP RAT"]

puts [format $formatStr "----------" "-------" "--------------" "---------" "---------" "--------" "--------" "-------" "-------"]

foreach design\_name $DesignName runtime $time\_elapsed\_in\_sec instance\_count $Instance\_count wns\_setup $worst\_negative\_setup\_slack fep\_setup $Number\_of\_setup\_violations wns\_hold $worst\_negative\_hold\_slack fep\_hold $Number\_of\_hold\_violations wns\_rat $worst\_RAT\_slack fep\_rat $Number\_output\_violations {

puts [format $formatStr $design\_name $runtime $instance\_count $wns\_setup $fep\_setup $wns\_hold $fep\_hold $wns\_rat $fep\_rat]

}

puts [format $formatStr "----------" "-------" "--------------" "---------" "---------" "--------" "--------" "-------" "-------"]

puts "\n"

The above script can be used to show the results in Horizontal format like shown in below figure.

