Lab 0: Introduction to CMC Cloud VCAD, Linux, and Tcl

COMPENG 4DV4: Very Large Scale Integration System Design

(Ungraded)

1 Learning Objectives

By the end of this laboratory, students will be able to:

- Access and use the CMC Cloud VCAD Linux environment.
- Navigate and manage files using standard Linux commands.
- Write and execute Tcl scripts used in ASIC EDA tools.
- Create a structured project workspace suitable for ASIC flows.
- Capture logs and outputs for reproducibility.

2 Environment Rules

1. Perform all work within the assigned CMC Cloud VCAD Linux instance.
2. Capture command outputs and logs for later verification.

3 Creating and Connecting to a CMC Cloud VCAD Remote Machine

Before starting the lab exercises, each student must create and connect to a Linux virtual machine hosted on the CMC Cloud VCAD platform. All subsequent steps in this laboratory assume that you are working inside this remote Linux environment.

3.1 Step-by-Step Procedure

1. Open a web browser and navigate to:

   `https://www.cmc.ca/cmccloud/`

2. Click **Access CMC Cloud VCAD**.
3. Select **Create My Instance** from the top menu.
4. Under **My Linux CAD Workstation (CentOS 7)**, click **Create Instance**.
5. In the instance configuration window:

- Accept the usage agreement.
- Select an instance duration appropriate for your lab session (e.g., 1 hour or multiple days).

6. Click **Create Instance** and wait for the instance to be initialized.
7. Once the instance is ready, go to **Manage My Instance** and click **Connect**.

3.2 Connecting to the Remote Desktop

CMC Cloud VCAD supports multiple connection methods. In this course, you will use the graphical remote desktop.

1. Download and install the NoMachine client from:

   `https://downloads.nomachine.com/`

2. From the CMC Cloud interface, click **Open Desktop (NoMachine)**.
3. Use the provided:
   - Remote IP address
   - Username
   - Temporary password
4. After login, you should see a Linux desktop environment.

4 Part A: CMC Cloud VCAD Workspace Setup

A1. Create Project Directory Structure

Execute the following commands in a terminal:

```
mkdir -p ~/asic_labs/{bin,docs,lab0,lab1,lib,logs,results,tmp}
cd ~/asic_labs
pwd
```

4.1 Save Directory Tree

```
tree -L 2 | tee project_tree.txt
```

If `tree` is unavailable:

```
find . -maxdepth 2 -type d | sort | tee project_tree.txt
```

5 Part B: Linux Fundamentals for ASIC Workflows

5.1 Directory Listing with `ls`

The `ls` command in Linux is used to list the contents of a directory. Understanding its options is essential for navigating and debugging ASIC workflows.

Before proceeding, read the manual page:

```
man ls
```

Experiment 1: Basic Listing Options

Run the following commands and observe the differences in output:

```
ls -l
ls -a
ls -t
ls -r
```

Lab Question 1:

- What additional information does `ls -l` display?
- What files appear when using `ls -a` that do not appear with plain `ls`?
- How does `ls -t` order files?
- What is the effect of `ls -r`?

Experiment 2: Combining Options

Linux commands can combine multiple options. Run:

```
ls -ltr
```

Lab Question 2:

- Explain what each option in `ls -ltr` does.
- Describe the final ordering of files produced by this command.

5.2 Create a File

Create the file `hello.txt` using `touch` command.

```
touch hello.txt
```

5.3 Edit a File

To edit the file `hello.txt` use `gedit` text editor.

```
gedit hello.txt
```

and write the phrases:

```
Hello ASIC
Hello World
Hello ASIC
```

5.4 Text Search with `grep`

The `grep` command is used to search for text patterns within files. It is frequently used in ASIC workflows to locate signals, modules, errors, and warnings in large log files.

Before proceeding, consult the manual page:

```
man grep
```

Experiment 3: Basic Pattern Search

Using the file `hello.txt` created earlier, run:

```
grep ASIC hello.txt
grep -n ASIC hello.txt
grep -i asic hello.txt
```

Lab Question 3 (GREP):

- What is the difference between `grep ASIC` and `grep -n ASIC`?
- What does the `-i` option change?
- In an ASIC design flow, give one example of when the `-n` option would be useful.

Experiment 4: Recursive Search

Run the following command from your project root:

```
grep -r "ASIC"
```

Lab Question 4 (GREP):

- What does the `-r` option do?
- Why is recursive search important when debugging large EDA projects?

5.5 Counting with `wc`

The `wc` (word count) command reports the number of lines, words, and characters in a file. It is commonly used to quantify logs, reports, and source files.

Consult the manual page:

```
man wc
```

Experiment 5: Counting Lines and Words

Run the following commands:

```
wc hello.txt
wc -l hello.txt
wc -w hello.txt
wc -c hello.txt
```

Lab Question 5 (WC):

- What does each column of `wc hello.txt` represent?
- How do `-l`, `-w`, and `-c` differ?
- In an ASIC flow, why might counting lines in a report file be useful?

Experiment 6: Combining Commands

Combine `grep` and `wc` to count occurrences of a pattern:

```
grep ASIC hello.txt | wc -l
```

Lab Question 6 (PIPELINE):

- What does the pipe symbol (`|`) do?
- Explain what this command computes.
- Give one practical ASIC-related example where this type of command pipeline would be useful.

5.6 File Search with `find`

The `find` command searches for files and directories based on name, type, size, and other attributes. It is essential for navigating large design repositories.

Consult the manual page:

```
man find
```

Experiment 7: Finding Files by Name

From your project root, run:

```
find . -name "hello.txt"
find . -type f
find . -type d
```

Lab Question 7 (FIND):

- What is the difference between `-type f` and `-type d`?
- Why is it important to distinguish files from directories in automated scripts?

Experiment 8: Searching by Extension

Search for all txt scripts in your project:

```
find . -name "*.txt"
```

Lab Question 8 (FIND):

- What does the `*` wildcard represent?
- How could this command be used to audit all Tcl scripts in a design flow?

Experiment 9: Advanced Find Usage

Run:

```
find . -type f -mtime -1
```

Lab Question 9 (FIND):

- What does the `-mtime -1` option select?
- Why might this be useful when tracking recent design changes or debugging tool failures?

5.7 Search and Inspection

```
grep -n "ASIC" hello.txt
wc -l hello.txt
head -n 5 hello.txt
tail -n 5 hello.txt
```

5.8 Environment Inspection

```
echo $SHELL
echo $PATH
which tclsh || echo "tclsh not found"
```

5.9 Save Command History

```
history | tail -n 200 > linux_basics.log
```

6 Part C: Tcl Fundamentals

6.1 Create Tcl Script

Create `tcl_basics.tcl` with the following contents:

```
# tcl_basics.tcl
set student_id "REPLACE_WITH_YOUR_ID"
set clk_period_ns 1.0
set freq_ghz [expr {1.0 / $clk_period_ns}]

puts "Student: $student_id"
puts "Clock period (ns): $clk_period_ns"
puts "Frequency (GHz): $freq_ghz"

set corners {TT SS FF}
foreach c $corners {
    puts "Analyzing corner: $c"
}

proc mhz_from_period_ns {p_ns} {
    return [expr {1000.0 / $p_ns}]
}

puts "Frequency (MHz): [mhz_from_period_ns $clk_period_ns]"

set fp [open "tcl_generated_report.txt" "w"]
puts $fp "Student: $student_id"
puts $fp "Clock period (ns): $clk_period_ns"
puts $fp "Frequency (GHz): $freq_ghz"
close $fp
exit
```

6.2 Run Tcl Script

To run a .tcl script we require a TCL compiler. Fortunately most ASIC design tools use TCL as their scripting language. We will use a tool called dc_shell to run our script.

```
source /CMC/scripts/synopsys.syn.2022.12.csh

dc_shell -f tcl_basics.tcl | tee tcl_basics_output.txt

cat tcl_generated_report.txt
```

7 Packaging Your File

```
cd ~/asic_labs
tar -czvf <studentID>_lab0_v1.tar.gz \
  linux_basics.log \
  tcl_basics.tcl \
  tcl_basics_output.txt \
  project_tree.txt
```