Background

In VLSI design, electronic design automation (EDA) tools are used for different design stages, including logic synthesis, placement, routing, extraction and timing verification. Some of these stages can potentially take several days depending on design complexity. These tools support interactive commands with/without graphic user interface (GUI). Interactive commands along with GUI environment are easier to navigate through and allow intuitive understanding of the design and impact of each design decision on potential outcomes. However, usually design choices and flow details are already decided, and only need to be executed in a sequential manner. Hence, using GUI mode is inefficient and is prone to errors since it requires human interaction. Well-designed command scripts relieve such efforts and prevent human errors, by automating the set of all tasks through a command script. The purpose of this document is to show some of the practical exercises to make well-designed command scripts.

Tip1: Prior to running a tool

To run a tool, the location of binary and license files must be set by shell commands. Location of binary file must be stored in the shell variable “path”.

```
set path=(Full_path_that_contains_the_binary_file  $path)
```

and the location of license file must be specified as follows.

```
setenv LM_LICENSE_FILE port1@address1:port2@address2:...
```

The following is an example of setting up the basic environment variables for Cadence SOCEncounter within "ieng6" machines.

```
set path=(/software/nonrdist/cadence-soc71/bin $path)
setenv LM_LICENSE_FILE 1704@ieng9.ucsd.edu
```

Unless path and license file are correctly set, you cannot run the tools. Therefore, make sure your shell command script points to correct environments.

Tip2: Command line helps to find valid options

Most tools support both command line and GUI modes. You can select one of them by setting command options. To find appropriate options, you can run the binary with or without “-help”, “-h” options. Please refer to the user guide of each corresponding tool for the exact syntax.

```
csh> binary_name -help | -h
```

For example, to see the options for SOCEncounter, you can try the following.

```
csh> encounter -help
```

As with most of the tools, you can run SOCEncounter in either text (batch-mode) or GUI modes:

With GUI:
```
csh> encounter
```
Without GUI:
```
csh> encounter -nowin
```

*NOTE: In SOCEncounter, the user can turn the GUI on and off using "`win`" and "`win off`" commands respectively in the SOCE shell prompt.

Tip3: Launch with log files

Some tools automatically generate their own log files, but some others do not. In addition, some tools allow the user to specify the log file name or the location of the log file.

If tools do not support automatic creation of log files, you can redirect the standard output log to a file. Try the following and observe the differences.

```
csh> command > filename
csh> command >> filename
csh> command |& tee filename
```

Tip4: Saving intermediate solutions and flags

If you are not sure about the quality of your batch scripts, it will be safer to save any intermediate solutions after major steps to avoid re-running them. Commands for saving intermediate results are different for each tool. Hence, you need to refer to the user guide of each tool. The following is a sample batch script for SOCEncounter. "`saveDesign`" command saves intermediate results and you can load them at any time.

```
loadConfig ./jpeg_encoder.conf 0
commitConfig
…
setPlaceMode …
placeDesign …
saveDesign placement.enc
…
ckSynthesis …
…
```

```
saveDesign clock_synthesys.enc
…
```

If you find any error in the batch script after generating "`placement.enc`", you do not need to run "`placeDesign`" again. Instead, you can load the latest successful result (`placement.enc`), fix the errors and then continue with the rest of the tasks. If you figure out a problem with the original batch script while some steps that create some intermediate databases are still running, you can check for the creation of the appropriate database and branch a new/correct script from that point on. Even if the existing scripts/runs are proceeding smoothly, they can fork-off the later tasks with a different tool (e.g., use an external STA tool, instead of an internal STA engine, or run multiple jobs with different options from the same intermediate stage) based on creation of necessary files.

Sometimes, you want to monitor the progress of the job. To do this, you can make a routine to output the appropriate flags. There are many possible methods to report the progress. The following are two possible methods that can be used.

Method 1:

```
echo "Synthesis start: `date`"  > process.log
rc -file synth.rc
echo "Synthesis finish: `date`" >> process.log
echo "PnR start: `date`"  >> process.log
encounter -nowin -init run_pnr.cmd
echo "PnR finish: `date`"  >> process.log
…
```

Just opening the "`process.log`", you can see what was done when and what is being doing now.

Method 2:

```
mkdir ./process
touch "./process/synthesis_start"
rc -file synth.rc
touch "./process/synthesis_done"
touch "./process/pnr_start"
encounter -nowin -init run_pnr.cmd
touch "./process/pnr_done"
```

"`touch`" command generates an empty file with user-specified name as an option. So, if you find the file in the "`./process`" directory, you can see which steps are already done. If you cannot see the specified files after some expected time has been elapsed, you need to check the other log files generated from each step. (If you list the content of a directory using "`ls -l`", then you can also see when the files were generated.)

In the example methods above, there are two separated jobs – the latter requires files generated from the former. If the first step does not work correctly and does not generate

correct output files, there is no point to start the second step. Between the first step and second step, you can add a routine to check file existence. The following example checks whether there is a netlist from the synthesis that will be used at P&R stage.

```
…
rc -file synth.rc
…
if (-e ${path_to_netlist}/mapped.v) then
  …
  encounter -nowin -init run_pnr.cmd
  …
else
  echo "no netlist for P&R"
end
```

Tip5: Diagnosis

Most tools generate their own log files. If there are any errors with respect to commands in the scripts, the tool will report possible reasons, and will suggest potential solutions with some level of severeness. Two major comments from tools are warnings and errors. Usually warnings can be ignored after checking their relevance, but errors must be fixed before proceeding any further. To collect warnings and errors, you can use the following shell commands along with the log files.

```
csh>  grep "ERROR|Error|error" log_filename > error.log
csh>  grep "WARN|Warn|warn" log_filename > warning.log
```

The warning and errors must be understood by referring to the tool user guide. Once you decide to ignore some type of errors or warnings, you can use the following with some regular expression for the text patterns.

```
csh>  grep "WARN|Warn|warn" log_filename | \
      grep -v "TECHLIB-436" > error.log
```

If you have verified all errors and warnings, the next step will be to check the expected outputs. You need to carefully examine all the output files from your runs. For example, if you want a netlist from synthesis that meets all timing constraints, you should see the timing report file generated by the synthesis tool.

Tip6: Making a batch script

Some tools automatically save command history during running interactive commands. For example, you can find "encounter.cmd" file which contains all the commands executed through command-line or GUI during the entire SOCEncounter job. You can

update the file, e.g., deleting unnecessary commands or adding additional commands, and save it as your own batch script file.

If a tool does not generate command history files, you can also find what you did during interactive mode from the log file. You can cut appropriate commands that you used and paste them to your own batch script file.

Tip7. Monitoring and automatic notification

Before running your jobs or during running them, you can see whether other people are using the same machine. You can also check to see whether your old jobs are still running.

```
csh> top
```

Please pay attention to available memory or computing power before launching new jobs. If a machine is already loaded with jobs that are consuming most of its memory and CPU resources then it is advised not to launch a new job on that machine. The reason is that usually EDA tools require significant amount of memory and CPU power, and if these are not met, the tool will crash incompletely.

In addition, before launching jobs, you must check if there is enough space to write the output data. To check the disk space, you can use the following command.

```
csh> df .
```

To monitor the progress, you do not need to sit in front of your monitor until all jobs are finished. If you expect that e.g., the synthesis job to be finished in 2 hours and you want to know when it runs and is finished, you can use the following techniques.

Write a script (`alarm.csh`) that checks whether the expected file (flags) is generated after 2 hours.

```
sleep 7200 ;
if (-e ./process/pnr_done) then
   mail -s "pnr was done" user@ucsd.edu < encounter.log
else
   mail -s "Error in pnr" user@ucsd.edu < encounter.log
end
```

or, you can use the following technique to get notified when the job is finished. The following script checks whether the flag file exists every minute, and once it is generated, a notification email will be automatically sent to the specified address with the log file. (Instead of only mailing the progress, you can add the next job in the following script.)

```
while (1)
  if (-e ./process/pnr_done) then
    break
  else
    sleep 60
  endif
end
mail -s "Job was finished" user@ucsd.edu < encounter.log
```

Once you have your notification script (i.e., alarm.csh), run it and wait for the incoming message.

```
csh>  source alarm.csh
```

If you do not get an email within the expected execution time, you need to come back to your terminal and check the scripts and log files.

Tip8: Before leaving your computer

As mentioned earlier, it is very often that batch scripts have bugs which either result in erroneous outcomes or incomplete runs. Hence, it is strongly recommended that you test each batch script line-by-line before running batch scripts.