系統晶片驗證 (SoC Verification)

113 學年下學期 電機系電子所選修課程 *943 U0250*

TA email：ntueesocv@googlegroups.com

Homework #1【Getting Familiar with verification and GV framework】

(Due: 9pm, Tuesday, 3/4)

【Objectives】

1. Getting familiar with the verification framework "GV" (developed by Design Verification Lab in NTU).

2. Writing monitors (assertions) and performing simulation-based verification on a provided RTL design.

【Disclaimers】

We only support Linux platform. Please compile and test the homework on Linux platform only. Although it is possible to compile it on other platforms, we will test and grade your homework on a Linux environment. It will be of fewer problems if you implement and test your program on the same platform.

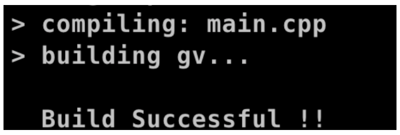
【Problems】

1. [Create gv repo with GitHub]

   "Please open the GitHub template and use it to create your own private repository, naming it **socv-1132**. Then, add **ntuee-socv** as a collaborator, and fill out the form to inform us of your GitHub ID and the URL of the repository.

2. [Installing GV]

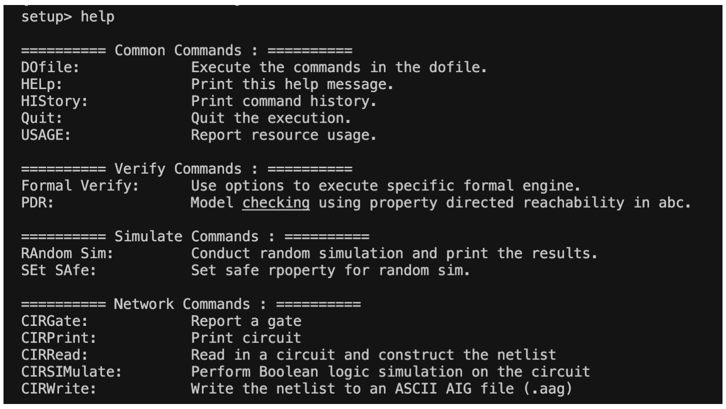
   Afterwards, git clone your repo to your local machine (i.e. git clone https://github.com/<your-github-ID>/socv-1132.git.) and install GV. You should be able to install the environment dependencies, and then compile GV by typing "sudo ./SETUP.sh" and "./INSTALL.sh". You may see the "Build Successful" message at the end of compilation, such as:



```
> compiling: main.cpp
> building gv...

  Build Successful !!
```

Upon the success of compilation, just type “*./gv*” under “*socv-1132/*”, and you should see GV running (with “setup> ” command prompt). Type “help” to see if it works properly.



```
setup> help

========== Common Commands : ==========
DOfile:             Execute the commands in the dofile.
HELp:               Print this help message.
HIStory:            Print command history.
Quit:               Quit the execution.
USAGE:              Report resource usage.

========== Verify Commands : ==========
Formal Verify:      Use options to execute specific formal engine.
PDR:                Model checking using property directed reachability in abc.

========== Simulate Commands : ==========
RAndom Sim:         Conduct random simulation and print the results.
SEt SAfe:           Set safe rpoperty for random sim.

========== Network Commands : ==========
CIRGate:            Report a gate
CIRPrint:           Print circuit
CIRRead:            Read in a circuit and construct the netlist
CIRSIMulate:        Perform Boolean logic simulation on the circuit
CIRWrite:           Write the netlist to an ASCII AIG file (.aag)
```

3. [Read and Write Designs Tutorial]

   Go through the “*Read and Write Designs*” tutorial in Chapter 1 of “*doc/GV_tutorial.pdf*” and repeat it for the “*vending.v*” design under the “*design/SoCV/vending/*” directory. In essence, do the following:

   (a) Read the spec of the design “*SPEC-vending_machine.pdf*” under the “*design/SoCV/vending/*” directory to understand architecture and design of the vending machine. Also, read the “*GV preliminary*” tutorial in Chapter 0 of “*doc/GV_tutorial.pdf*” to get further familiar with GV usage.

   (b) Read in the original RTL design (*vending/vending.v*) and write it out into “blif” and “aig” formats. Name them as “*vending.blif*” and “*vending.aig*”, and put them in the same (i.e. “*design/SoCV/vending/*”) directory.

   (c) Report the network statistics by the command “print info -verbose” for the three output designs in (b) (“*vending.v*”, “*vending.blif*”, “*vending.aig*”). Put the screen snapshot of the output message into a file "*hw1_3c.pdf*" under the "*hw/hw1/*" directory.

- Since GV utilizes the open source engine – Yosys, which cannot read over one design without replacement. Hence, remember to replace the original design by adding the argument "-replace" or "ctrl+C" when reading a new design.

(d) Plot the "rtl" network (vending.v) by the command "show", which can plot the high-level architecture of the RTL circuit. Take a screenshot, name the output file "*plot_vending-rtl.png*" and put them in the "*socv-1132/hw/hw1*" directory.

- Since the plotting package (graphviz) needs Xserver to display the window, you might need to use the following tools:
    - [windows] MobaXterm
    - [MAC] XQuartz: by "brew install xquartz". Open XQuartz on your Mac. You don't need to use the terminal of XQuartz for X11 forwarding, but you do need to have XQuartz open. If you need ssh, then establish an ssh connection that enables the X11 forwarding function by "ssh -X <username>@<server-name>"
- The warning messages can be ignored when plotting.
- Besides, *vending.v* is not a toy design, so it is hard to understand its architecture after plotting. It is good to start from a smaller design block and plot its netlist to realize how the "show" command works.

4. [Design Simulation Tutorial]

Go through the "*Design Simulation*" tutorial in Chapter 2 of "*doc/GV_tutorial.pdf*". Please note that the vending machine design inside this tutorial is the original (more complicated) one. In this problem, you are asked to perform a simulation for the simplified one (i.e. "*vending.v*" under the "*design/SoCV/vending/*" directory).

Please note there is at least one bug in the design, and the monitor "*p*" in line 112 is enough to reveal it. This monitor signified the bug that, when there is no transaction or the transaction fails (e.g. not enough changes), the output changes are NOT equal to the input changes. The input pattern file "*design/SoCV/vending/input.pattern*" is for your reference. You should try to generate more test patterns to reveal the bug and figure out the cause(s). Please refer to Appendix A of GV's tutorial for the format of the simulation pattern, and please pay attention to the order of the input variables in the pattern files.

- Note that the simulation by default runs 20 cycles, you can try to use option "-sim_cycle <cycle_times>" to tune the simulation cycles, which makes your design be able to finish running all patterns.
- Through simulation cycle tuning, you can learn that a design might need multiple cycles to finish running one pattern, which means as your design becomes larger, it might take thousands or even millions of cycles for simulation.

Create a subdirectory "*debug*" under the "*socv-1132/hw/hw1/*" directory, and put your input pattern file "input-cex.pattern" and output log "output-cex.pattern" in it. You should also include a file "*hw1_4.pdf*" to describe how you find the bug and fix it and put them in the "*socv-1132/hw/hw1*" directory.

5. [Monitor (assertion)]

   Save the rectified design as "*vending-fixed.v*" in the "*vending/*" directory. In order to verify the design, you should write at least two more "monitors" to guard the potential bugs and generate more patterns to check if any of the monitors can be triggered. If another bug is found, you need to debug. Note that sometimes it is not a bug in the design, but a bug in the monitors you write. Repeat the process (i.e. writing more monitors and creating more patterns) until you think the verification quality is good enough.

   Since the manual pattern creation process is very tedious, you are encouraged to write some "driver" program to automatically generate legal patterns for simulation.

   Put your input pattern file and output log in the subdirectory "*verify*" under "*socv-1132/hw/hw1*". You should also write a report "*hw1_5.pdf*" in the same directory to describe: (1) your monitors and how sound you think they can guard the design, and (2) the test patterns and their verification results.

【Submission of Homework】

Please name the files and place them in directories properly or as suggested by the description above. In summary, you should at least have the following new items for your homework1:

- *socv-1132/hw/hw1/*
  - *hw1_3c.pdf*
  - *hw1_4.pdf*
  - *hw1_5.pdf*
  - *plot_vending-rtl.png*
  - *debug/*
    - input-cex.pattern
    - output-cex.pattern
  - *verify/*
    - input-cex.pattern
    - output-cex.pattern
- *socv-1132/design/SoCV/vending/*
  - *vending.blif*
  - *vending.aig*
  - *vending-fixed.v*

Please be sure to push the files to the github repo before the deadline (i.e. git add .; git commit; git push). TAs will pull the files of the repo to our Linux local machine (Ubuntu 22.04 LTS) at the deadline to grade the assignments.

Failure to abide by the above rules may result in deduction of your HW grade.