ECE/CS 5745/6745: Testing and Verification of Digital Circuits

Fall 2014, Homework # 2
Due Date: Mon, September 29, 2014 (in class)

In this assignment, you will get introduced to three types of tools that are used in design verification: i) SAT solvers; ii) BDD-based verification tools; and iii) AIG-based synthesis and verification tool. You should download/compile/install these tools on your personal and/or CADE lab machines (in your home directories, of course). *The URLs for all these tools are available on the class website.*

SAT solvers: So many SAT solvers are freely available on the web, feel free to download and install any and all of them. Both Lingeling (recent SAT competition winner) and Picosat solvers from Prof. Armin Biere (Johannes Kepler Univ., Linz) can be downloaded and compiled. I have also uploaded my own compiled copy of the zChaff solver on the class website.

The ABC tool: ABC is a AIG-based synthesis and verification tool from Alan Mishchenko. Downloading and compiling is also painless. Go to Alan's ABC website, download the ABC source and compile. I've also uploaded and older version of my personal compiled copy for you (in case you are having problems). *Please also go through the ABC manual and tutorials available on Alan's website.*

The VIS tool: VIS stands for Verification interacting with synthesis. This is a BDD-based verification tool. Interest in VIS seems to have waned over the past few years, so there has not been a whole lot of updates and maintenance. There are two ways to get VIS:

- Get the VIS-2.4 version from Univ. Colorado's website (URL on the class webpage), read the manual and compile. Some students have had problems in compiling the tool recently, probably due to some newer compiler versions doing some stronger typecasting (maybe also some library issues).
- Download my personal, though quite a bit older copy of VIS. The tarball I'm providing already has a compiled binary that should run on the CADE lab linux machines.

Sample BLIF and CNF files: The ones needed for the experiments are uploaded along with this document.

It would be a good idea to install these tools in a 'tools' directory in your own home-dir, and also update your PATH environment variable to point to these executables. *Later on when you start working on the programming assignments and projects, you may have to modify the source-code of some of these tools. Therefore, it would be a good idea to try to compile them, and start reading their user and programmer's manuals. Just to get started for this assignment, you could use my pre-compiled binaries; however, you*

cannot rely on them forever. I will also request that you help each other in compiling and installing these tools. In case the newer versions of these tools require modification of the Makefiles or configure files (and you have figured it out!), please document those changes and share with the class and with me.

Installing and Running VIS: VIS actually requires the environment variable VIS_LIBRARY_PATH to be properly set. The tarball that I have uploaded on the class website is VIS.tar.gz. Say, you copy this file in a "tools" directory in your home-dir:

```
~/tools/VIS.tar.gz
```

Download, gunzip and untar:

```
prompt> gunzip VIS.tar.gz
prompt> tar -xvf VIS.tar
```

This will create a VIS/ directory, which will include a glu-2.x and a vis-2.x directory (x = version). The "vis" executable resides in the vis-2.x directory. To read the BLIF files, VIS makes use of some awk-scripts which are in the vis-2.x/share directory. The system needs to know where they are. So you set the environment variable VIS_LIBRARY_PATH as follows. If you use (t)csh:

```
setenv VIS_LIBRARY_PATH ~/tools/VIS/vis-2.x/share
```

In bash/bourne shells:

```
export VIS_LIBRARY_PATH="~/tools/VIS/vis-2.x/share"
```

If you don't know your shell, type:

```
echo $SHELL
```

Feel free to use Canvas discussion board to ask for advice from your fellow classmates. Good luck!

Now, let us get to the assignment:

1) (30 points) Consider the circuit of Fig. 1 that we have analyzed in class before. Assume that this is the specification circuit.

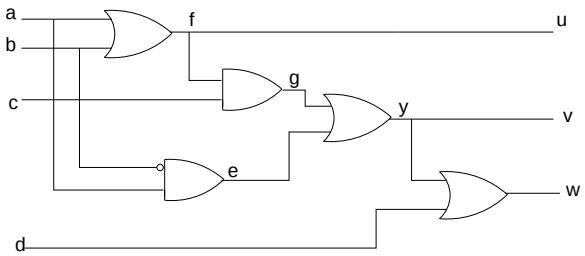

Fig. 1. The circuit for equivalence checking

a) Design another implementation of this circuit with only AND-XOR gates (and inverters if needed). No OR gates are allowed. Consider this as your "optimzed" implementation.
b) Draw the circuit schematic. Use a drawing package such as xfig or inkscape.
c) You are to check the equivalence between these two circuits using SAT. Miter these two circuits, write the CNF formula, and solve SAT using any solver. How many decisions, implications, conflicts, restarts, etc. are needed?
d) Introduce a bug (of your choice!) in the design, and catch the bug using SAT.
e) Make sure that internal and output node names in both circuits are different. Only PIs are tied together.
f) Submission: Briefly describe your setup, show your circuit schematic, and attach the solver's output.

2) (15 points) Now that you know how to use SAT for verification, re-run this experiment using ABC.
a) Write out the AND-XOR circuit that you designed in BLIF format.
b) Verify the two circuits using ABC. The commands that you need to explore in ABC are: help, read_blif, strash (builds AIG for the circuit), print_stats, ifraig -sv (does fraiging to identify and merge equivalent nodes in the network), miter, cec, write_blif, write_cnf, etc.
c) The -h option tells you how to use a command. Eg: 'ifraig -h'
d) To observe the power of FRAIGing: create a miter between file1 and file2; strash; print_stats; ifraig -sv; print_stats. This will tell you the number of AIG nodes in the miter-circuit prior to and post FRAIGing.
e) Experiment with both bug-free and buggy designs. Does ABC provide you with a counter-example that excites the bug in the design?
f) Re-run the experiment with the somewhat larger files C7552.blif and C7552_opt.blif, both uploaded on the class website along with this assignment. Also introduce a bug in C7552_opt.blif, and re-run the experiment, for both equivalence proof and bug-catching.
g) To compare the performance of ABC-CEC with SAT-based CEC, you could: i) miter file1.blif file2.blif; ii) write_cnf miter.cnf; and then invoke the SAT solver on miter.cnf. [The only problem is that the miter command of ABC already does some FRAIGing and reduces the CEC instance].
h) **Extra Credit (100 points): worth 2 extra HWs**: If you want, you can write a perl/shell/Ruby/whatever script that: i) takes two blif files as inputs; ii) writes out a "structural" miter between them (miter.blif), without any simplification. I can then give you my BLIF2CNF script. This can be used for a fair comparison against ABC's FRAIGed miter for SAT-based verification!

i) Submission: Print the network and AIG stats for the circuits, the number of AIG nodes removed by FRAIGing, and attaching the script/output of your ABC run.

3) (15 points) Re-run the same experiments now with VIS.

a) The commands to use in VIS are: help, read_blif, flatten_hierarchy, static_order, write_order, build_partition_mdds, print_bdd_stats, print_network_stats, and comb_verify, among others.

b) Submission: Attach the script/output of your VIS experiment, and mention the max number of BDD nodes in the manager.

c) If you really like to abuse your computer, download the C6288.blif file, and run: i) read_blif C6288.blif; ii) flt; iii) so; iv) part inout; and see how memory gets consumed by BDDs.

In all the experiments, try to use the verbose options (-v) of the tools/commands, and print out as much information (print_stats) that you can from the tools. It will give you an idea of various operations and their results. Also, create your own designs (BLIF/CNF files); don't just borrow it from someone else ☺.

Have fun! In the next assignment, I'll give you some programming with VIS/BDDs to solve a problem.