ECE 697VS: Hardware Verification using Symbolic Computation

Prepared by *Priyank Kalla*
Spring 2017, Homework # 1
Due Date: Thu, Feb 16.

Note: I would like that this HW be uploaded by the students electronically on Moodle. I'm still trying to figure out how to give you permissions to upload your assignment. I'll let you know as soon as I figure that out. Please have the HW completed by Feb 16, when I'll assign you similar experiments with BDDs.

In this assignment, you will get introduced to two types of tools that are used in design verification: i) SAT solvers; and ii) AIG-based synthesis and verification tool. You should download, compile and install these tools on your personal and/or (UMASS ECE) lab machines (in your home directories, of course). *The URLs to access these tools are available on the class website under the "CAD, Verification, and Computer Algebra Tools and Resources" section.*

SAT solvers: 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, available from Alan Mishchenko of UC Berkeley. 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 in compiling the too). *Please also go through the ABC manual and tutorials available on Alan's website.*

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.

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

Now, let us get to the assignment:

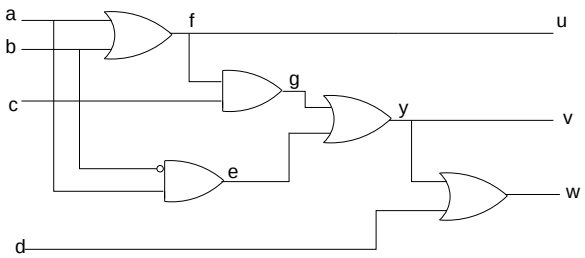

Fig. 1: The circuit for equivalence checking

1) **[Simulation vector generation, 10 points]** Consider the circuit of Fig. 1. You are asked to identify an input vector that excites $u = 1, v = 1, w = 0$ at the primary outputs. Formulate and solve the problem using SAT.
   a) Write the CNF file corresponding to the above problem, and execute a SAT solver on this file. Use any SAT solver.
   b) Repeat the experiment for $u = v = w = 1$.
   c) Submission: Describe your problem formulation and attach your solver's output.
2) **[Equivalence checking, 25 points]** Consider, again, the circuit of Fig. 1. Assume that this is the specification circuit.
   a) Design another implementation of this circuit with only AND and XOR gates (and inverters if needed). No OR gates are allowed. Consider this as your "optimized" implementation.
   b) Draw the circuit schematic. Use a drawing package such as `xfig` or `inkscape`.
   c) You are asked to check the equivalence between these two circuits using SAT. *Miter* these two circuits, write the CNF formula in a file, and solve SAT on the CNF file using the solvers. How many decisions, implications, conflicts, restarts, etc. are encountered or executed by the solver?
   d) You should execute multiple solvers on the same CNF file, and record these runtime stats to compare the performance of these solvers. Display a comparison of these results as a graph or a table.

e) Introduce a bug (of your choice!) in the design, and catch the bug using SAT.

f) Make sure that internal and output node names in both circuits are different. Only the primary inputs (PIs) are tied together.

g) Submission: Briefly describe your setup, show your circuit schematic, and attach the solver's output.

3) **[AIG-based equivalence check, 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 (`miter file1.blif file2.blif`); 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 issue is that the miter command of ABC already does some FRAIGing and reduces the CEC instance].

h) **Extra Credit (25 points!)**: If you want, you can write a Perl/Python/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; actually, it can be found in the CAD resources section on Moodle, under the BLIF database. 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.

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 them from someone else ☺.

You will notice that I'm not asking you to attach CNF files or BLIF files of your designs. However, I want you to preserve those source files, as I may ask to see those files later. You may also need them for later assignments.

Have fun! In the next assignment, I'll give you some programming exercises with BDDs and the VIS verification tool to solve similar problems.