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

Prepared by *Priyank Kalla*

Fall 2023, Homework # 3

Due Date: Wed Oct 18, 2023, by midnight

Note: The HWs should be uploaded by the students electronically on Canvas. Please have the HW completed by midnight Oct 16.

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 ABC. You should download, compile and install these tools on your personal and/or (CADE) 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. All the SAT solvers use the DIMACS Conjunctive Normal Form (CNF) format, which write one clause per line.

The ABC tool: ABC is a And-Invert-Graph (AIG) based synthesis and verification tool, available from Alan Mishchenko of UC Berkeley. Downloading and compiling is quite painless. Go to Alan's ABC website – the URL is also given on the class website https://people.eecs.berkeley.edu/~alanmi/abc/ – download the ABC source and compile. I've also uploaded an older version of my personal compiled copy for you (in case you are having problems in compiling the tool). *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.

BLIF to CNF Conversion: The ABC tool can convert a blif file to a CNF file: `read_blif, write_cnf`. I have also uploaded a Perl Script blif2cnf.pl, which can convert a BLIF file to a CNF file for you. Note that a BLIF file has net names, whereas a CNF file uses numbers to denote a literal or its complement. The blif2cnf.pl script can also write out a file with info on how the net names were mapped to numbers: `perl blif2cnf.pl -m fname <file>`.

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, 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 the discussion forum on our Canvas website for issues with tool usage.*

Now, let us get to the assignment:

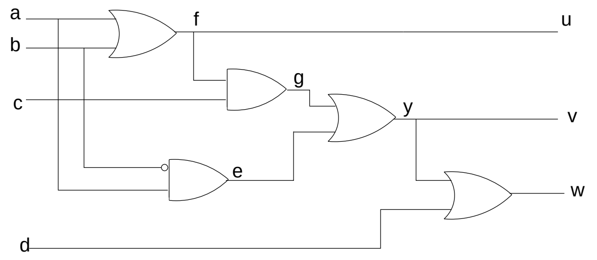

Fig. 1: The circuit for equivalence checking

1) **[Simulation vector generation, 30 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 (your own CNF file!) corresponding to the above problem, and execute a SAT solver on this file. Use any SAT solver.
   b) Make sure to associate the names of the nets of the circuit to the variable number in the CNF file. Read the CNF-SAT solver's output to identify the simulation vector. Write the obtained simulation vector.
   c) Repeat the experiment for $u = v = w = 1$.
   d) Submission: Describe your problem formulation and attach your solver's output. Explain how you converted the logic gates to CNF clauses. Which SAT solver did you use?

2) **[AIG-based equivalence check, 20 points]** This experiment will help you gain some experience with the usage and the capabilities of AIG-based tool ABC. In this question, you will use ABC to perform combinational equivalence checking (cec) between two designs: `C7552.blif`

and `C7552_opt.blif`. Then you will introduce a bug in the design and use ABC to detect the presence of the bug(s). Proceed as follows:

a) To 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 (builds a miter between two BLIF files),
cec, (combinational equivalence checking)
write_blif,
write_cnf, etc.
```

You will use these commands to explore the use of ABC.

b) The -h option tells you how to use a command. Eg: 'ifraig -h', 'miter -h', etc. Most of the commands have a '-v' option, which denotes high verbosity level. Try to use -v where available, so you can get more information from the tool.

c) First of all, create a miter between file1 and file2 (`miter file1.blif file2.blif`).

d) Then, use the following sequence of commands: `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) Then perform combinational equivalence checking between `C7552.blif` and `C7552_opt.blif` using the `cec` command.

f) To experiment with both bug-free and buggy designs, modify some logic function in the `C7552_opt.blif` file to introduce a bug – e.g. change a gate, or change inputs to a gate, etc. Does ABC provide you with a counter-example that excites the bug in the design?

g) To compare the performance of ABC-CEC with SAT-based CEC, you can perform the following experiment: i) miter file1.blif file2.blif; ii) write_cnf miter.cnf; iii) then invoke a SAT solver on miter.cnf. [The only issue is that the miter command of ABC already does some FRAIGing and reduces the AIGs in the CEC instance, so the miter.cnf file is already kind of optimized]. Nevertheless, try to run SAT on miter.cnf and compare the run-time stats.

h) Submission: Briefly describe your experiments, and also your conclusions from these experiments. Print the AIG stats for the circuits, the number of AIG nodes removed by FRAIGing, and attaching the script/output of your ABC run.

i) 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.

3) **(Rectification of buggy circuits with Craig Interpolation – 50 points.)**

a) Consider the circuit of Fig. 2. Assume that this is a “specification” model (*Spec*). Write the corresponding BLIF file for this circuit: `spec.blif`.

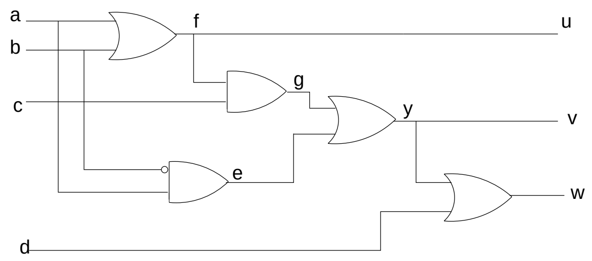

Fig. 2: The specification circuit for the rectification problem

b) Consider the net $e = a \cdot \overline{b}$ in the circuit. Introduce a bug by changing this gate in the circuit to an OR gate, i.e. introduce the buggy gate: $e = a \vee b$. Write the corresponding buggy implementation blif file: impl.blif.

c) Perform equivalence checking using ABC: `cec spec.blif impl.blif`. Confirm that the gate change is indeed a bug, i.e. confirm that CEC generates a counter-example to equivalence.

d) This bug should be rectifiable at the same net e, of course. Using the single-fix rectification theorem (Thm 1 in our slides on rectification), confirm if the circuit can be rectified at net e.

- **Explain how are you performing this experiment.** From our slides, you may recall that the circuit is rectifiable at a net x_i if and only if the intersection of the miters is UNSAT, i.e. $M_0 \wedge M_1 = \perp$. How are you performing this check using ABC and SAT?
- Hint: You could write out M_0.$blif$ and M_1.$blif$ as BLIF files using ABC, and then combine them with an AND gate $thm1$.$blif$, and then run SAT on them. If you do so, make sure to simplify the BLIF files: e.g., `miter spec.blif impl.blif; strash; ifraig -sv; collapse; resyn; write_blif M0.blif`.
- Also, remember to constrain the output of the final AND gate to 1.
- You can also use the BLIF 2 CNF Perl Script.

e) Subsequently, compute the rectification functions using both the smallest and the largest interpolants, and validate your result by re-running CEC with the rectified functions.

f) For the bug $e = a \vee b$, can the circuit be rectified at net g? If so, compute the corresponding rectification function at g for the bug at e. Keep in mind the fanout at net f.

g) **Important Note:** Note that the computations for the rectification check as well as for the rectification function can be performed manually (of course), but also using ABC. Since the rectification function can be computed as an interpolant of appropriate M_0 and M_1 miters (see slides pp. 12), **I want you to compare your results of rectification functions computed using smallest and the largest interpolants against the one provided by ABC**. Recall, ABC has a heuristic procedure to compute **any** interpolant between (and including) the smallest and the largest one.

h) To compute interpolants using ABC, see command: "`inter -h`". I have created two blif files "`ci_A.blif, ci_B.blif`" corresponding to the example given on my rectification slides on page 15. Run the commands: "`inter ci_A.blif ci_B.blif; write_blif inter.blif`", and see in the file `inter.blif` which interpolant is computed. These files are uploaded with this HW assignment.