ECE 522/622 - Spring 2025
Modeling and Verification of Embedded Systems
Lab #1 Reachability Analysis

Objectives of this Lab:

1) To understand symbolic reachability analysis
2) To gain familiarity with Boolean satisfiability problem
3) Practice programming skills

Sources:

1) **Minisat** SAT solver: http://minisat.se/
2) **Picosat** SAT solver: http://fmv.jku.at/picosat/
3) **DIMACS CNF** syntax: http://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html
4) **Icarus Verilog** simulator: Find installation instructions for your respective machine.
   - For Linux/Ubuntu: "**sudo apt-get update"** followed by "**sudo apt-get install iverilog**" in the terminal.
   - For Windows: https://bleyer.org/icarus/
   - For Mac: https://ee.sonoma.edu/resources/computer-hardware-and-software/icarus-and-gtkwave-mac
5) **Guide to getting Started with Icarus Verilog**: https://steveicarus.github.io/iverilog/usage/getting_started.html
6) **Further reading** about the correspondence between gates and CNF clauses: (on Canvas)
   T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 1, pp. 4–15, 1992

General lab report instructions:

1) You can work alone, or in groups of two.
2) Lab reports must be typed and submitted in PDF format, with name(s) and ID(s) at the top of the lab report.
3) All screenshots and text should be clearly readable.
4) Only groups that have one or more ECE622 students are required to complete part C.
   - ECE522 students can complete part C for extra credit (total not to exceed 100 points).
   - ECE522 students that skip part C will have the scores for A and B scaled up proportionally.
5) Office hours are the best resource for detailed help if you get stuck.
6) Use Piazza to ask and answer questions. We will monitor and comment on Piazza as well.
7) Other than the SAT solver, you must generate all the code you use to complete the lab. You can use tools to help you generate code, but cannot use code from other people, or code found online. If you use tools to help you generate code, you must state this in the report and explain how you used them. For example, if you use ChatGPT to generate a graph, or to generate code that converts Verilog to CNF, you must provide the sequence of prompts used. If you want to use any existing code or API, you must ask first on Piazza to make sure it is allowed.

Introduction:

Sequential systems have memory elements with values that evolve according to inputs and combinational logic. The behavior of sequential systems is modeled by states and transitions. Reachability analysis deals with determining which states can be reached from a given initial state or states. In this lab, the initial state is 0 for each benchmark. Reachability analysis of large embedded systems is a complex task attracting significant research efforts.

The examples used in this lab are provided as sequential Verilog design modules with combinational logic and state updates on each positive clock edge. These benchmark designs are posted on Canvas with the lab assignment. For simplicity, the only combinational gate types used in the designs are 2-input AND gates and NOT gates (inverters). Fig 1 shows a graphical schematic of the smallest benchmark (ex1) with 4 gates and 2 flip flops. The correspondence between logic gate types and CNF clauses are as shown in Fig 2, and more details can be found in the Larrabee paper listed above. For more details, and especially for information about unrolling the CNF transition relation formulas in symbolic reachability, please see the reachability slides from class.

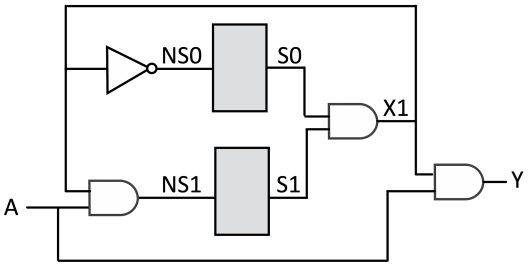

Fig. 1: Gate-level schematic of ex1.v

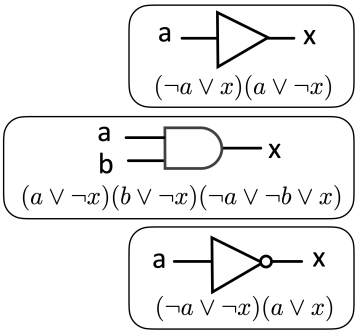

Fig. 2: CNF formulas for gate types needed in

design	target state bits (i.e. 31 indicates NS31/S31)																													
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ex1																															1	1
ex2																		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ex3				0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0	1
ex4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
ex5																								0	0	0	1	1	0	1	1	1
stoplight1																													0	1	0	0
stoplight2																												0	1	0	0	0

Fig. 3: Target states to use for checking reachability in each design

Part A (Reachability by Graph Traversal): *[25 points]*

In this part of the lab, you will explore how to check reachability using random execution (with Verilog testbench) and using explicit reachability by graph traversal (on paper).

A.1) Draw the state transition graph for ex1 and stoplight1. You can use any method that you prefer to determine which state transitions are possible (pen and paper, simulation, etc). Explain your approach for determining which transitions are possible.

- The graph should have a node for each of the (4 or 16, respectively) states, and a directed edge for each possible state transition, as in slides.
- Annotate nodes with their corresponding state, e.g. the node of initial state for ex1 will be annotated as 00.
- The graph should be generated on a computer (pen on tablet does not count). You can create the graph manually (Visio, Powerpoint, draw.io, etc) or generate it automatically from text (TikZ, Graphviz, etc.).

A.2) Use the state transition graphs to determine whether the target states are reachable in ex1 and stoplight1. If so, give the sequence of states visited along the shortest path to the target state. Shortest path means the path with the fewest transitions, or equivalently, fewest clock cycles. Describe your approach.

A.3) Include a table showing how many states of ex1 and stoplight1 are reachable within i transitions, for all values of i from 1 to 20. "Reachable within i transitions" is cumulative, meaning that any state reachable within j transitions will also be listed as reachable within j+1 transitions, regardless of whether it is possible to reach that state on transition j+1 itself.

A.4) For ex1, ex2, stoplight1, and stoplight2, write a Verilog testbench to initialize the state to 0, apply random inputs in each cycle, and notify you if/when it reaches the target state. If the target state is not reached within 100,000 cycles, the result can be reported as "timed out". If the target state is reached, report how many cycles were simulated before first reaching it. See included file demo.v for an example of a Verilog testbench.

- Include a screenshot of the simulation result where the target state is reached.
- Include your testbench for stoplight1 as part of your report.

A.5) Explain whether/how your result for stoplight1 A.4 is consistent with your result from A.2. Be specific about what finding in A.4 would have shown that at least one of the results (A.2, A.4, or both) must be wrong.

Part B (SAT-Based Symbolic Reachability): *[60 points]*

This part of the lab solves the same problems as above, but now uses SAT-based symbolic reachability. Essentially, you are writing your own simple verification tool for Verilog that uses SAT solving as a backend. You must write a program that will check reachability by parsing Verilog modules, unrolling the transition relation and converting it to dimacs formatted CNF, and calling a SAT solver. The SAT solver that your program calls (e.g. picosat, minisat, or any other that you choose to install on your computer) will check whether the formula is satisfiable, and the result will reveal whether the target state is reachable in some number of transitions from the initial state. You can write your program in C++, Java, or Python (check with us on Piazza if you want to use another language). Your program must take three inputs from command line: (1) the name of the Verilog file to read; (2) the number of times to unroll the transition relation in the symbolic search; and (3) the target state as a string of 0 and 1 values ordered by descending index as listed in the table. To check whether stoplight1 can, in 10 transitions, reach the target state in the table above, then you might call your program like this:

```
python3 lab1_src_<lastname>.py stoplight1.v 10 0100
```

B.1) Use your program to generate the CNF files for ex1 with the transition relation unrolled 2 times. "unrolled 2 times" means there are 2 instances of the transition relation. Be sure to add the initial state of 0, and to enforce that the next state of each transition relation is equal to the starting state of the subsequent transition relation. Use the SAT solver to check whether state 11 is reachable in 2 transitions (as the 2nd state after the initial). Is your finding consistent with the state transition graph from question A.1? Explain your answer. Now that you've checked whether "11" is reachable in 2 cycles, repeat the analysis to check whether the other 3 states (00,01,10) are reachable in 2 cycles, and again explain whether your findings are consistent with question A.1. Include screenshots of results from the SAT solver for each case. Be sure to fix any bugs here before trying the larger examples that follow!

B.2) For each design, perform symbolic reachability analysis with the transition relation unrolled 15 times, to check if the target state can be reached on the 15th transition. In other words, you are checking whether the target state can be written into the flip-flops on the 15th rising clock edge. Describe your program and any data structures used. For each design, indicate whether the target state is reachable, and include screenshots of results from the SAT solver. Measure the SAT solver runtime for each design and provide the runtimes in the report.

- Upload the source code of your program as a text file when submitting your report. Name the file as `lab1_src_<lastname>.cpp` (substitute appropriate extension, use lastname of one team member).
- Upload the dimacs CNF for ex2 with transition relation unrolled 15 times, and appropriate constraints added for initial and target state. Name the file `lab1_ex2_<lastname>.dimacs`
- **ECE622 only**: Resolve the same seven cases using a different SAT solver. Include the name of the solver, its runtime, and the results for each design.

B.3) For the stoplight1, from initial state, check whether the target state is reachable on the ith transition for i=1,2,…,32. This should be done by invoking your program (and the SAT solver) 32 times while changing the input argument that specifies the number of times to unroll the transition relation. You can do this using a script. Report the values of i for which the target state is reachable.

B.4) Repeat the previous question using stoplight2. How does this compare to your finding from random execution of stoplight2 in A.4?

B.5) Find all states that stoplight2 could reach on the 17th transition from initial. Explain your approach.

Part C (Counting Reachable States): *[15 points]*

ECE622 only: You are free to choose your own approach to solving this problem. Some will be easier than others, so it is advisable to consider multiple options before you commit to implementing one approach.

C.1) In design ex5, starting from the initial state, what is the total number of states that are reachable within i transitions (defined to be cumulative, as in A.3). Answer this question for all values of i from 1 to 20. Repeat to find the same information for ex1. Give your answers as table. You can compare the ex1 result against the result from A.3 to gain confidence that your part C approach is producing correct answers.

C.2) Describe your approach and submit as a separate file any code that you used to answer the question.