ECE 597MB/622 - Spring 2022
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 introduction:** http://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html
4) **Icarus Verilog (simulator):** http://iverilog.icarus.com
5) **Further reading about the correspondence of gates and CNF clauses:** (on Moodle)
   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) Lab reports must be typed and submitted in pdf format, with name and ID at the beginning of the Lab report.
2) All screenshots and text should be clearly readable.
3) If you complete the lab by yourself, then you can skip Part C and still be awarded the 15 points of credit for that question. If you work in a team of 2, then you must complete Part C to earn the 15 points.
4) JD's weekly office hours are the best resource for detailed help if you get stuck
5) Use Piazza to ask and answer questions. We will monitor and comment on Piazza as well.
6) You must write all the code you use to complete the lab. If you want to use an existing resource, 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 Moodle with the lab assignment. For simplicity, the only combinational gate types used in the designs are 2-input AND gates and NOT gates (inverters). Figure 1 shows a graphical depiction of the smallest benchmark (ex1.v) with 4 gates and 2 flip flops. The correspondence between gate types and CNF clauses are as shown below, 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.

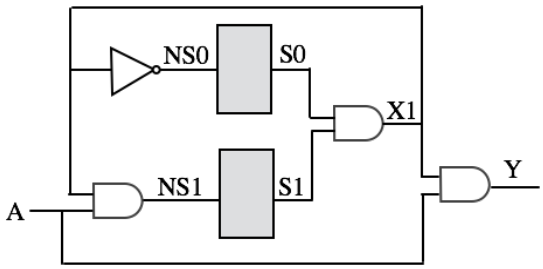
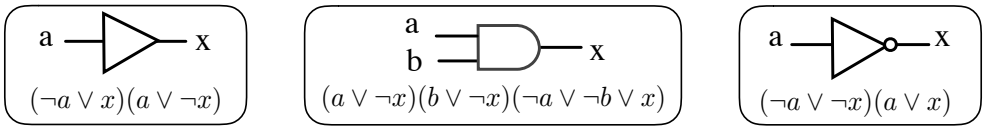


Fig. 1 Drawing of ex1.v

CNF formulas for the gates needed in this lab:



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
stoplight1																													0	1	0	0
stoplight2																												0	1	0	0	0

Part A [20 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).

(1) Draw legibly the state transition graph for ex1.v and stoplight1.v. The graph will include states and edges to mark the transitions, as we've seen in class. Your graph must include all states, even unreachable ones. You can use any method that you prefer to determine which state transitions are possible (pen and paper, simulation, etc), but you must include all transitions. Explain your approach.

(2) Use these two 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.

(3) For the benchmarks given (ex1, ex2, ex3, ex4, stoplight1, 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 execution can terminate and the result can be reported as "timed out". If the target state is reached, report how many cycles of random simulation were performed before first reaching it. Include your stoplight2 testbench in your report.

(4) If you have made a mistake on A.3 or A.2, you may find that these two methods give results that are incompatible with each other. Describe what type of simulation results from A.3 would be incompatible with your finding from A.2. Did you find such a result? If so, fix it.

Part B (Symbolic Reachability) [65 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, C++, Java, Perl 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. For example, to check whether stoplight1.v can in 10 steps reach the target state shown in the table above, if you use python then you might call your program like this:

```
python3 solveReachability.py stoplight1.v 10 0100
```

(1) Use your program to generate the CNF files for ex1 with the transition relation unrolled twice. Be sure to add the initial state of 0 and be sure to enforce that the next state of each transition relation is equal to the starting state of the next transition relation. Use the SAT solver to check whether state 11 is reachable as the 2nd state after the initial one. 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. Be sure to fix any bugs here before trying the larger examples that follow!

(2) For each benchmark, perform symbolic reachability analysis, with 10 unrollings. Note that you are checking whether the target state is reachable in exactly the 10th state after the initial one. In other words, you are checking whether the next state in the 10th unrolling can be the target state. Describe your program and any data structures used. For each design, indicate whether or not the target state is reachable as the 10th state. How long does the SAT solver take to perform its search for a satisfying assignment?

(3) Compare your results using SAT-based reachability to the results from random simulation. Explain any differences in results and what causes them. Are there any limitations in SAT-based reachability due to only checking whether a state is reachable in the last unrolling and not intermediate ones? How might you improve the symbolic search to avoid this limitation?

(4) Submit the source code of your program as a separate file with your report and include instructions for how to run it for testing.

(5) For the benchmark stoplight1.v, check whether the target state is reachable as the ith state after the initial state 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 unrollings. You can do this using a script. Report the values of i for which the target state is reachable. How does this compare with your results from A.2?

(6) Repeat previous question using benchmark stoplight2.v. How does this compare to your finding from random execution of stoplight2.v in A.3?

(7) Given the number of state bits in the stoplight2.v, how many states (including unreachable ones) does the system have? How many of those states can be reached as exactly the 17th state from the initial state. Explain your approach.

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

This question may take a substantial amount of work, and you only need to complete it if you choose to work in a group of 2, so choose carefully.

(1) In design ex2.v, starting from the initial state of all 0 values, what is the total number of states that are reachable within i steps. Answer this question for all values of i from 1 to 20. Give your answer as both a table and plot (y-axis should be number of reached states, and x-axis should be the value of i).

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