ECE/CS 5745/6745: Testing and Verification of Digital Circuits
Hardware Verification Using Symbolic Computation

Prepared by *Priyank Kalla*
Fall 2025, Homework # 4
Due Date: Monday Oct 27, 2025. Upload on Canvas by midnight.

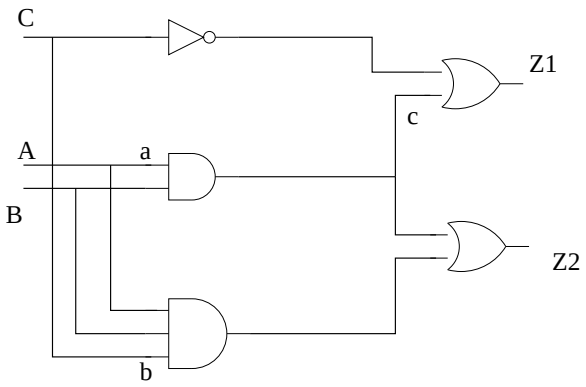

Fig. 1: Test for faults b/1, tests for distinguishing a/0 from c/0 and test for multi-fault {a/0, b/1}.

1) **[ATPG: 15 points]** For the circuit shown in Fig. 1:
   a) Find the set of all tests that detect single stuck-at fault b *s-a-1*.
   b) Find the set of all tests that distinguish between single stuck-at faults (i.e. their fault effects) a *s-a-0* and c *s-a-0*.
   c) Derive a test for the multi-fault { a *s-a-0*, b *s-a-1* }.
   d) Based on the outcome of the above three tests, state your observations regarding the (un)testability of single and multi-faults. Refer to the appropriate class slides on multiple stuck-at faults, and the concept of test invalidation for multi-faults under the presence of redundancies.

2) **[ATPG Checkpoints: 25 points]** For the circuit shown in Fig. 2, solve the following:
   a) (10 pts) Suppose we wish to derive a set of tests to *distinguish* between all distinguishable single stuck-at faults in the circuit of Fig. 2. Identify a set of faults for which tests *need not*

be derived.

b) (5 pts) List all the checkpoint faults of this circuit.

c) (10 pts) Find a *smallest/minimal* subset of checkpoint faults that must be targeted for test generation if the *detection* of all single stuck-at faults is the goal.

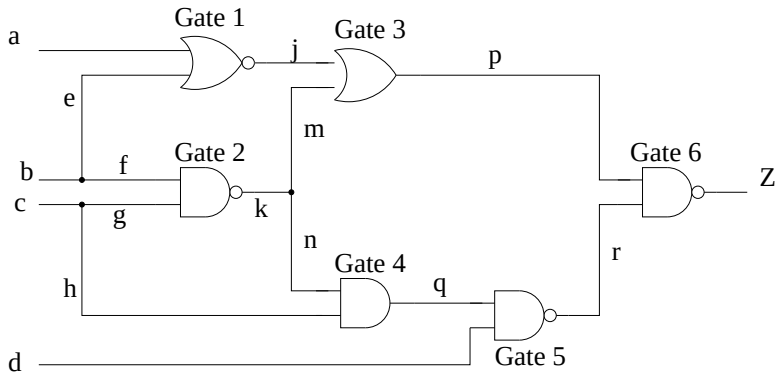

Fig. 2: The circuit diagram related to Checkpoint faults

3) **[Stuck-at faults at fanout stems and branches: 20 points]** For the circuit shown in Fig. 2:

a) (15 pts) Derive a test that detects the following single faults (i) k/1; (ii) m/1; and (iii) n/1. You should use path-sensitization to derive these tests. Of course, you can check your answer by setting up the problem as a miter between a fault-free and a faulty circuit, and use the 'cec' command of ABC!

b) (5 pts) If any of the above faults is undetectable, remove the redundancy by removing redundant gates and/or lines, and draw the simplified circuit.

4) **[20 points] This question is for ECE/CS 6745 students. ECE/CS 5745 students may solve it for extra credit.** Let N be a combinational circuit composed only of NAND gates. Assume that all the primary inputs of the circuit have a fanout of exactly one (1). Show that any test set T that detects <u>all</u> single **stuck-at-1** faults in the circuit, detects all single **stuck-at-0** faults as well. *Note: Do not assume that the circuit is fanout-free. Only the primary inputs (PIs) are fanout free. (A fanout of 1 means that the gate output is connected to only 1 other gate input, and this gate is also referred to as being fanout-free).*

5) **[10 points: Equivalence checking versus bug-detection in arithmetic circuits]**. On the class website, along with this HW, I have uploaded two BLIF files. They correspond to a 16-bit Mastrovito GF multiplier (`MastrovitoF_q16.blif`) and another 16-bit Montgomery GF multiplier (`MontgomeryF_q16.blif`). As described in my book chapter, (which we will study in the next few lectures) these architectures perform modulo-multiplication, but are based on different mathematical concepts — these designs do not exhibit any internal structural or functional equivalences. As a result, SAT/AIG-based techniques are unable to prove equivalence between them because the AIG of the miter cannot be functionally reduced (fraig'ed). Instead of taking my word for it, you will gain a first-hand experience for yourself.

- Input the two designs into the ABC tool, and miter them.
- Using `print_stats, strash, ifraig, print_stats`, identify the structural similarity in the design. Let N_1 be the number of AIG nodes in the miter before *fraiging*, and N_2 be the number of AIG nodes after *fraiging*. Then $\frac{N_1 - N_2}{N_1}$ roughly depicts the structural similarity as a percentage.
- Solve sat on the miter (or equivalently run the `'cec'` command) to perform the combinational equivalence check. Is it feasible to prove equivalence of (fairly small) 16-bit datapath circuits? ☺
- Now introduce a bug in any one of these circuits, by making any modification to any one of the BLIF files. Now run ABC-CEC and report whether bug catching or correctness proofs for arithmetic circuits is easier.

Describe your experiments and state your conclusions from these experiments.