Bonus Project
Automatic Test Pattern Generation

ECE 586, Fall 2013

Due Date: 10:00am 12/06 Chicago time
Late submissions will NOT be graded

1 Introduction

In this project, you will practice the knowledge we have learned in the lectures to create a software program for automatic test pattern generation (ATPG). We will focus on ATPG for combinational circuits, especially the combinational parts of synchronous sequential circuits, to support the Design for Testability (DFT) methodology.

You can implement the program in any language you are familiar with, e.g. C, C++, Java, and Python. This project should be done individually. You can discuss the project with other students but:

- The source code and the report must be your OWN.
- You must ACKNOWLEDGE other students for the discussion.

Any violation will call for DISCIPLINARY ACTION. The consequences may include to assign 0 for your project score, or to remove you from the class.

Please note that since this is a bonus project, we cannot provide further help beyond this project instruction.

2 The ATPG Algorithm

The flow chart of the ATPG algorithm is shown in Fig. 1. The algorithm starts by reading a benchmark synchronous sequential circuit and then builds the netlist data structure to represent its combinational part. For this project, we will use a set of the synchronous sequential circuits from the ISCAS'89 benchmark, which are widely used for experimenting new VLSI CAD algorithms. The circuits can be downloaded from the course webpage and are all in a textual format that is self-explaining. Please note that certain circuits may contain gates that are isolated or in the clock distribution network and those gates need to be removed first.

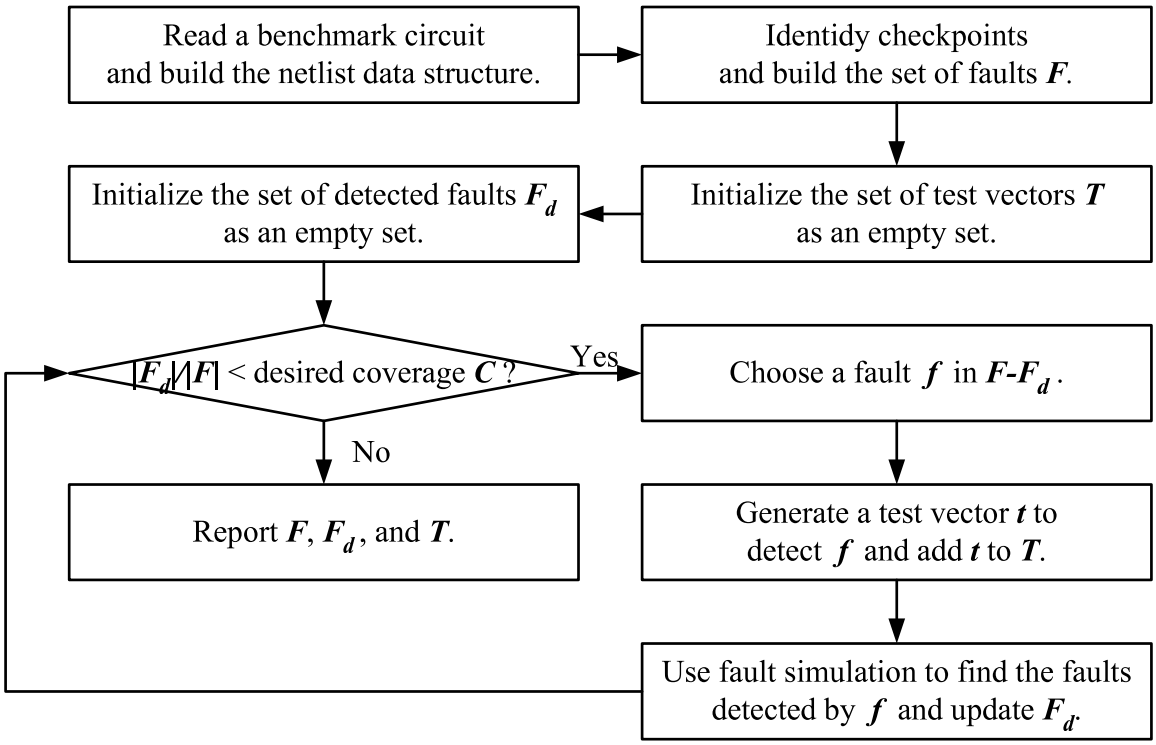

Figure 1: The ATPG Algorithm

The algorithm then proceeds to identify the checkpoints and to build the set of faults F that should be detected (s-a-0 and s-a-1 at each checkpoint). It is optional for you to apply structural equivalence in order to further reduce the number of faults in F.

The test generation loop starts by initializing both the set of test vectors T and the set of detected faults F_d (with respect to T) to empty sets. Then it iterates until the fault coverage $\frac{|F_d|}{|F|}$ is no less than the desired coverage C, which will be given as a constant.

In each iteration of the test generation loop, a fault f that is not detected yet is first chosen (from the set $F - F_d$). Then, a test vector t should be generated to detect f. For this purpose, you may utilize the algorithms introduced in Chapter 6 for line justification, or simply leverage any existing SAT solvers, e.g. MiniSAT (available at `http://minisat.se/`). Finally, you should perform fault simulation to decide what faults in $F - F_d$ can be detected by t and update F_d accordingly. Note that for fault simulation, you can either use the algorithms that decide exactly the faults in $F - F_d$ detectable by t, e.g. serial fault simulation, or use critical path tracing to discover the faults approximately.

3 Project Evaluation

You are required to prepare a report of no more than 6 pages to describe your implementation details and *to explain why the results generated by your program are correct.* The project will be worth 40 points and be graded according to the correctness of your implementation and the quality of your report.

We will evaluate your project using the following four criteria.

- Checkpoints (10 points): your program is able to identify the checkpoints correctly.
- Fault simulation (10 points): your program is able to identify most, if not all faults detectable for a given test vector.
- Test vector generation (10 points): your program is able to generate a test vector to detect a given fault correctly.
- System integration (10 points): your program works as a whole to cover $C = 80\%$ faults.

In the case your program may work for some but not all benchmark circuits, partial credits will be given – however, your program should work correctly for at least one circuit in order to be considered for partial credits.