Final Project

You will implement an automatic test generator for single stuck-at faults. It will include (a) fault collapsing; (b) a circuit simulation algorithm, and (c) a test generation algorithm. We assume a single-stuck-fault (SSF) model. ***Python is strictly required for this project for the ease of testing of your code.*** The following are the requirements for the program:

1. The program must run in an interactive mode.
2. The input to the program is a gate-level net-list.
3. The generator must parse the gate-level netlist and then report the following:
   (a) The final list of fault classes after performing fault collapsing
   (b) For each detectable fault, show the list of test vectors that will detect such faults.
   (c) The list of undetectable faults (if any).
   (d) The list of detectable faults for a given test vector revealed by circuit simulation.

Figure 1 shows the general flow of the test pattern generator.

On program invocation, the following interactive menu should be presented to the user:

```
[0] Read the input net-list
[1] Perform fault collapsing
[2] List fault classes
[3] Simulate
[4] Generate tests (D-Algorithm)
[5] Generate tests (PODEM)
[6] Generate tests (Boolean Satisfaibility)
[7] Exit
```

These options are explained with more details below.

- First it reads the circuit (Option `0`) and generates appropriate data structures.
- Option `1` performs fault collapsing and creates the fault classes.
- By selecting `2`, the found fault classes are displayed, one for each line.
- Option `3` simulates the circuit. It takes an input test vector and a set of faults, and show faults propagated to one of the outputs at the end of simulation. If no faults are provided, it simulates the circuit with the provided test vector to show the output values.
- Options `4` to `6` provide different test generation algorithms. You choose to implement one of them.

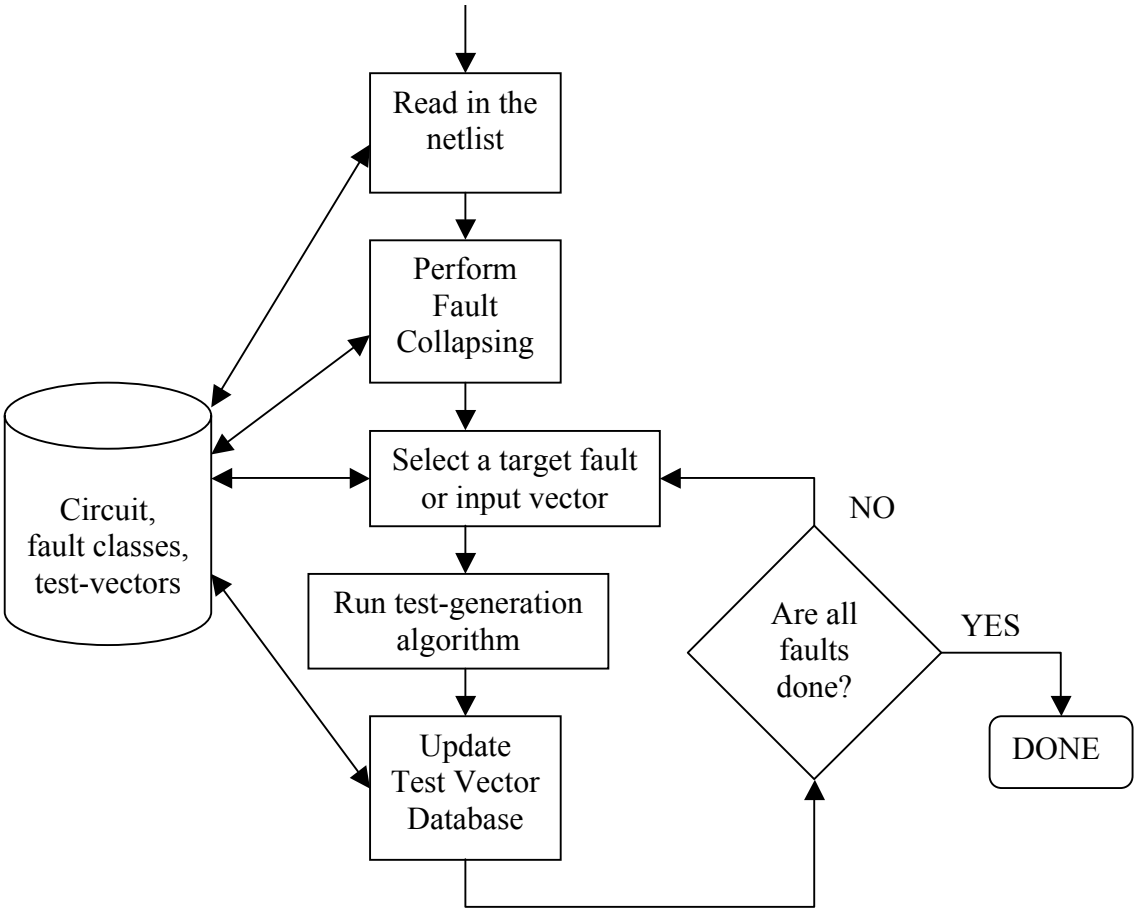

Figure 1: High level task flow for the ATPG.

If you choose to implement option `6`, you need to convert a test generation program into the corresponding Boolean satisfiability problem, and then use an existing Boolean SAT solver to find the tests. Some of the modern efficient solvers can be found at `http://www.satlive.org/solvers/`. *Make sure you pick one of the CDCL solvers.*. Another interesting and good solver is the `Z3` solver which also has a Python distribution (`https://pypi.org/project/z3-solver/`).

Teaming. You can work in a team with **up to** 2 people.

Evaluation

This project carries 35% of the final grade, and it is evaluated as follows.

1. **Correctness** Whether your code produces correct results for a set of test cases. A set of 5 circuits will be provided to you. You make sure that your code should run

correctly for them. Additional 5 circuits (not provided to you) will be run to check your code during your project demonstration. So make sure your code implements the test generation algorithm correctly instead of specializing your code for those provided test circuits.

2. **Coding style** Your code should have good *Readability* and *Modularity*.
3. **Error handling** Your code should exits gracefully in presence of wrong input entry (such as mistype, wrong option, non-existent file name, etc.)

0.1 Deliverables

1. Submit a final report summarizes your project, *i.e.* code structure, compile instructions, results from applying your generator to the five benchmark circuits.
2. Upload archived code in a single zipped file including a above final report, Makefile, and other necessary files.

0.2 Input Netlist format

The input netlist (self-explanatory) will be given in a format illustrated by the following example. Note that "$" denotes the start of a comment. An example is given below.

```
$c17 iscas example (to test conversion program only)
$-------------------------------------------------
$
$  total number of lines in the netlist ..............    17
$  simplistically reduced equivalent fault set size =     22
$        lines from primary input  gates .......     5
$        lines from primary output gates .......     2
$        lines from interior gate outputs ......     4
$        lines from **     3 ** fanout stems ...     6
$
$        avg_fanin  =  2.00,     max_fanin  =  2
$        avg_fanout =  2.00,     max_fanout =  2
$
1gat                                    $... primary input
2gat                                    $... primary input
3gat                                    $... primary input
6gat                                    $... primary input
7gat                                    $... primary input
                                        $... primary input
$
$

22gat                                   $... primary output
23gat                                   $... primary output
                                        $... primary output
$
$
$       Output  Type    Inputs...
$       ------  ----    ---------
        10gat   nand    1gat    3gat
        11gat   nand    3gat    6gat
        16gat   nand    2gat    11gat
        19gat   nand    11gat   7gat
        22gat   nand    10gat   16gat
        23gat   nand    16gat   19gat
```