ATPG (DRAFT)

Group Project Phase #3

In the final phase of the group project you will implement a complete *automatic test pattern generation* (ATPG) system. Many of the required modules are already implemented in the first two phases of the project. In this phase, you will focus on the implementation of two ATPG algorithms, D-Algorithm and PODEM, and later connect the available modules to generate set of input patterns to provide a high fault coverage test.
Just like the other phases, this new version of your code should be compatible with the older one, in other words, it should preserve its correct functionality and format for previous versions.

D-Algorithm:
Considering your circuit has been already read by the simulator, D-Algorithm generates test vector for one single fault. Add a new command called DALG that takes two arguments, the node number, and the stuck at fault, separated by a space:
DLAG <node-number> <fault 1/0>. Here is an example:
DALG 15 1 *//doing D-algorithm for node 15 stuck at 1.*
The generated test vector can have 1, 0, and X values and should be stored in a files named:
<circuit-name>_DALG_<node>@<fault>.txt
for the fault mentioned earlier the output name will be:
c17_DALG_15@1.txt
The format of the output pattern file is similar to the input of DFS, or logic simulation with just one pattern: the top row is the PI node numbers, followed by a line of PI values, all values separated by a comma.

PODEM:
The flow is similar to D-algorithm part, using PODEM command.

ATPG_DET:
The ATPG_DET (deterministic algorithms) operates stand alone, meaning there is no guarantee that the operator read the circuit, levelized it, etc.
ATPG_DET <circuit-name> <alg-name DALG/PODEM>

- ATPG operation should result in a **small** set of test vectors that is required for testing this circuit.
- Do not use any random patterns in this command.
- You may need to call fault simulation to make sure about the efficiency of your test.
- All faults are targets for test. However, many of them share equivalence or dominance relationship, which fault simulation will take care of it. Moreover, some faults are more important (refer to phase). Considering priority for specific faults will helps you with a better test outcome.
- Two output files will be generated by running this command:
    1. Input patterns required for testing the circuit should be written in <ckt-name>_<DALG/PODEM>_ATPG_patterns.txt with this format: first row has the name of the nodes separated by comma. The next rows are assigned to each test pattern, and the values are in the same order as the order of nodes in the first row, separated by comma. The values are binary (0 or 1 and there should not exist any X value).
    2. Report of the ATPG process in <ckt-name>_<DALG/PODEM>_ATPG_report.txt which is a log file of the process, including the system time required for running ATPG, from the time the ATPG command is issued till the time the report is written down in the file, measure in seconds. In addition, the fault coverage percentage should be reported. You need to report this metric based on all the faults of the circuit, and not just the reduced fault list. An example of the format of the report is shown below:

c345_DLAG_ATPG_report.txt

```
Algorithm: DALG
Circuit: c345
Fault Coverage: 85.12%
Time: 741 seconds
```

ATPG report example

ATPG:
This is your final product: ATPG <circuit-name>
Consider all the modules you implemented so far (PFS, DFS, DALG, PODEM, random pattern fault simulation, checkpoints, etc.) or other techniques (e.g. using testability measurements such as SCOAP) to design a **strategy** for developing a test process for circuit netlists. This is the only part of your project that needs a written report (in any format, slides, document, etc.) that you will use for your project demo.

- The command operates stand alone, reads the circuit, levelizes it, etc.
- Your strategy can/should be **dynamic**. In other words, you can modify your process based on the specifications of the circuit, or the run-time condition. As an example, changing from random-pattern algorithm to deterministic, or the frequency of running fault simulation, etc. can be changed within runtime.
- Explore different methods by designing experiments, and report them. We will ask about the logic behind making decisions and it's best if you have some intermediate results in your report to provide during the demo. Your demo will be focused on these strategies.
- Two output files will be generated by running this command which which are similar to the outputs of ATPG_ALG:
  1. Input patterns <ckt-name>_ATPG_patterns.txt
  2. Report of the ATPG process in <ckt-name>_ATPG_report.txt

General guidelines for submission:

- Team work! Team work! Team work!
- You know about compile.sh, simulator, `viterbi-scf`, members.txt
- Submit one ".zip" folder: `"EE658_Phase3_⟨GroupNumber⟩.zip"` and only one person per group should submit the project.
- Do not submit executable files, netlists, report, etc.
- We will only use your submitted code during the demo.
- The report is for you to document and present your intermediate experiments and results during the demo, it will not be graded separately.

Don't forget team work!
Good luck!