Project 1

ECE-470 Digital Design II

Project outline

In this project you will implement a logic and fault simulator on top of which you will then implement the D-Algorithm or PODEM algorithm. You are given a preliminary C++ framework implementation that contains basic classes and the circuit netlist parser. You are encouraged to develop this project on top of the given framework. However, you are free to use any other programming language you want. Your implementation will be verified as a logic simulator as well as a test generator for single stuck-at faults.

Part 1: C++ project framework (10 points)

You must read and understand the given C++ framework on top of which you will implement this project. You will write a 1-2 page report to describe the architecture code: data structures/classes and their relationship with a digital circuit netlist.

Part 2: Logic simulator (25 points)

In the first part, the simulator will read-in a circuit netlist, in mapped-blif format, and a set of input vectors (generated by a binary counter). The output should be the logic values at the primary outputs (POs), which will be saved into a text file, in a format similar to the file containing the primary inputs (PIs) vectors. The main elements of this part of the project are as follows:

- **Mapped-blif circuit netlist**. You will be given several circuit netlists, as text files (example: *circuit_1.map*). You will also be given a list with all the logic-gates used for the mapping of these circuits. You will need such a list to create your own library of logic gates, internal to your simulator.
- **Input file** with binary-counted primary input vectors (example *pi_circuit_1.txt*). You will have to create this file, for each given circuit, based on how many PIs the circuit has. For example, for a circuit with two PIs, this input file would contain four lines:

0 0
0 1
1 0
1 1

- **Output file** with the logic output; on a separate line for each input vector (example *po_circuit_1.txt*).

One should be able to run your logic-simulator executable using a command line like this:

simulator –mode logic_simulator circuit_1.map pi_circuit_1.txt po_circuit_1.txt

Where *simulator* is the name of your executable, *mode* is an argument which selects the logic-simulator mode, and the rest of arguments are the input and output files. The grading, of this part, will be done by running your simulator on the given netlists as well as additional netlists (not available to you). You will deliver your code and demonstrate your simulator directly to the instructor. You will also have to answer questions about your code/implementation. These questions are intended to verify that you are the one who did the implementation. Failure to answer these questions will result in a grade of zero for this part of the project.

Part 3: Fault simulator (15 points)

You will enhance the logic simulator to be able to simulate faults; thereby, implement a fault simulator. The main elements of this part of the project are as follows:

- **Mapped-blif circuit netlist**.
- **Input file** with single stuck-at faults. For example, such an input file with two faults would look like:

SignalLine3 0
SignalLine5 1

One should be able to run your fault-simulator executable using a command line like this:

simulator –mode fault_simulator circuit_1.map pi_circuit_1.txt faults_1.txt

Where *simulator* is the name of your executable, *mode* is an argument which selects the fault-simulator mode, and the rest of arguments are the input files. The grading, of this part, will be done by running your simulator on the given netlists as well as additional netlists (not available to you). You will demonstrate your simulator directly to the instructor. Again, you will also have to answer questions about your code/implementation.

Part 4: Test generator (40 points)

In this case your simulator will work as a test generator. It will implement a fault simulator and will have to generate, if the fault is detectable, test vectors, using either the D or PODEM algorithm, for given single stuck-at faults. The main elements of this part of the project are as follows:

- **Mapped-blif circuit netlist**.
- **Input file** with single stuck-at faults. For example, such an input file with two faults would look like:

SignalLine3 0
SignalLine5 1

- **Output file** with the test vectors.

One should be able to run your simulator executable using a command line like this:

simulator –mode test_generator circuit_1.map faults_1.txt tests_1.txt

You will have to deliver both your code and your executable. You will also have to prepare a written report describing the implementation of your project and results. The grading, of this part, will be done by running your simulator as a test generator for the given faults as well as other single stuck-at faults. The written report will also be part of the grade. Extra credit will be given for additional elements such as parallel implementation for shorter computational runtime, GUI, etc.

Part 5: Written report (10 points)

The final report will have to describe what the project is about and the code architecture of your implementation. Also, you must describe the main problems that you faced and how you resolved them. Finally, you must include a table with results (including CPU runtimes) for the provided circuit-netlists.

Deliverables

(Part 1) Initial report with description of the given C++ code (**Due on Sep. 1 2011**)
(Part 2) Logic simulator (**Due on Sep. 20 2011**): (a) Your code, (b) Demonstration on given circuits and input vectors.
(Part 3) Fault simulator (**Due on Oct. 4 2011**): (a) Demonstration on given circuits and stuck-at faults.
(Part 4) Test generator (**Due on Oct. 25 2011**): (a) Your updated code, (b) Demonstration on given circuits and stuck-at faults.
(Part 5) Final written report (**Due on Oct. 27 2011**)