

SIFU! - A Didactic Stuck-at Fault Simulator

Vinícius P. Correia Marcelo Lubaszewski André I. Reis
vincor@inf.ufrgs.br luba@iee.ufrgs.br andreis@inf.ufrgs.br
Instituto de Informática – UFRGS - Caixa Postal 15064
CEP 91501-970 – Porto Alegre – RS – Brazil

Abstract

This paper presents a didactic simulator for stuck-at (sa) faults on logic circuits. The tool has a set of features that helps to understand the concepts of single and multiple stuck-at faults, being these faults testable or not, and how to generate test vectors in order to test the detectable fault subset. An interface was developed to allow the edition of a circuit, the injection of faults and the fault simulation. The tool performs two simulations concurrently, one for the original circuit and another for the faulty circuit considering the injected faults. When the two simulations differ, for a given input vector, the tool shows the error (detection of the fault) graphically.

1. Introduction

Logic simulation of a combinational circuit is able to obtain the logic response of the circuit to a given input vector. Fault simulation consists of simulating a circuit in the presence of faults. Faults can be modeled according to several models. One of the classic fault models is the stuck-at (sa) fault model, in which faults are modeled as circuit nodes permanently connected to the power supply (stuck-at 1 fault) or to the ground (stuck-at-0 fault). In order to perform a fault simulation, it is necessary to inject some faults in the circuit to be simulated. This way, when using the stuck-at model, the fault injection consists in stating that some internal nodes (wires) are permanently connected either to the ground or to the power supply. Comparing the fault simulation results with those of the fault-free simulation of the same circuit simulated with the same applied input vector, it is possible to determine the faults detected by this input vector [1].

Despite there are many logic simulators (didactic or not) available, the authors of this paper ignore the availability of didactic fault simulators. This way a tool to graphically perform fault simulation was implemented. It is named SIFU! [2], which is an acronym for fault simulator in portuguese. The motivation to build this software comes from the Test Course (CMP 116) of the Master Program (PPGC UFRGS-Brazil). The remaining of this paper will present this tool and how it is able to deal with the concepts related to stuck-at fault simulation.

2. Related Concepts

The tool has been designed to help the study of test concepts in the context of combinational digital circuits. These concepts are briefly described in this section.

Error – An unexpected behavior in an output or in an internal node that can be observed, probably caused by a fault.

Fault – A defect in the circuit that can lead to an error or not. Physical faults can be permanent, transient or intermittent. These faults are modeled as logical faults that represent the effect of the physical faults on the circuit. The faults can be detectable or undetectable. A detectable fault can be observed by an error that it causes.

Stuck-at-level Fault – Consists on a signal being stucked at a logic value, zero or one. Represents a physical fault like an improper soldering.

Faulty Circuit – A circuit with faults, detectable or not.

Single Fault – A single fault in the faulty circuit under test.

Single Stuck-Fault Model – The most used fault model, also know as standard fault model. This model assumes that the faulty circuit has only a single stuck-at-level fault.

Fault Injection – An indication of the presence of a fault in the circuit to the data structure of the simulator. This indication is used in the simulation of the faulty circuit.

Logic Simulation – Analysis of the functional behavior of the circuit, where the signals are propagated and operated from the inputs to the outputs, in a fault-free circuit.

Fault Simulation – The same kind of computation done for the logic simulation, but considering the presence and effects of injected faults. It is performed on the faulty circuit.

Fault Coverage – It is the ratio between the number of detected faults and the total number of injected faults, for a given set of input vectors.

Test Generation – The action of generating a set of input stimuli to detect faults. The quality of the resulting test set is given by its fault coverage.

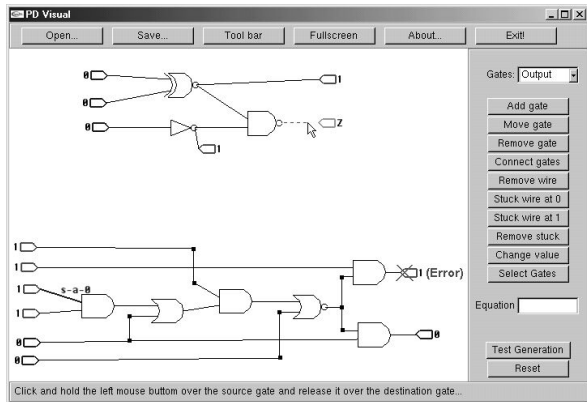


Fig. 1 – The main editing window. Screenshot of the last wire being connected to an output.

3. Working the concepts with SIFU!

The concepts related to fault simulation using the stuck-at model are easily worked out by using SIFU! fault simulator. Figure 1 shows a screenshot of SIFU!, running a fault simulation while the user is finishing to edit a circuit. The circuit is composed of logic gates, primary inputs and outputs, as well as wires connecting them. These elements may be freely positioned in the workspace. Simple logic gates of any kind (and, nand, or, nor, xor, xnor and inverters) are supported. Circuit nodes are treated as a special kind of gate that is used to organize circuit wires when a gate drives two or more other elements. This feature is shown in figure 1, for the wire in the output of the nor gate. Fault injection is done by sticking wires either to logic zero or to logic one. Every fault injected is indicated in the appropriate wire by a s-a-0 or s-a-1 tag. The circuit in figure 1 has a s-a-0 fault injected, as indicated by the specific tag.

The logic simulation is constantly being performed during the execution of Sifu!. The input vector applied to the circuit is shown graphically, where each current input value is shown on the left of the corresponding input gate. Similarly, output gates are used to graphically show the values resulting from the simulation. The fault simulation is also performed, considering the injected faults and their effects. As both simulations are performed concurrently, each time they differ from each other for an output gate of the circuit, an error occurs, and an error tag is shown. This way, it becomes clear that faults are detected by appropriate input stimuli that allow the fault to propagate an error to a primary output. Another important issue is that some faults are not testable and this way they will never produce an error that propagates to a primary output of the circuit. The user learns this by changing the input stimuli without ever produce an observable error. The user will acquire the concept of test pattern generation, while trying to observe an error at the output by randomly generating test vectors. With practice, the user begins to develop more clever methods to generate test vectors.

The test generation feature performs an exhaustive search for a minimal set of input vectors that detects the maximum number of testable faults. Obviously, this exhaustive search is not the best choice when working with larger circuits, but this is not goal of SIFU! as a didactic tool. This function was designed to produce an exact pattern cover, but the implementation of other methods, like the pseudorandom method and others used in commercial tools are being considered for the next version of the tool.

Finally, it is important to notice that the result of comparing the simulations of the faulty and fault-free circuits is only indicated by error tags on primary outputs. The same happens in a real circuit where not all the internal points can be probed. This leads to the concept of design-for-testability, where the circuit design process includes intermediate steps to make modifications on nodes with relevant information for the test of the system, in order to make them able to be externally probed. This way the addition of extra outputs for intermediate signals increase the overall testability of the circuit by adding more observable nodes. In the upper circuit of figure 1, every gate signal is observable.

4. Tool Features

The tool is cross platform, as it was coded to be operating system independent. It was written in C++, using the STL [3] and the OpenGL [4] libraries. Simple logic gates of any kind (and, nand, or, nor, xor, xnor and inverters) are supported, for any number of inputs.

The user is allowed to save and restore all configurations of the current workspace. The tool is also able to save the circuit using the mapped blif format, allowing other tools like SIS [5] to use the circuit.

5. Conclusion

This paper presented a didactic tool for fault simulation developed for student use. SIFU! tool works out several test concepts, allowing an easier understanding of them.

6. Acknowledgements

This project was partially supported by CNPq and FAPERGS Brazilian founding agencies.

7. References

- [1] Abramovici et al. Digital Systems and Testable Design, IEEE, 1990.
- [2] SIFU! site: <http://www.inf.ufrgs.br/gme/lagarto/sifu>
- [3] Stroustrup, B.. The C++ Programming Language. Wesley, 1997.
- [4] OpenGL site: <http://www.opengl.org>
- [5] E.M.Sentovich et al. "SIS : A system for sequential circuit synthesis". Technical report UCB/ERL M92/41, UC Berkeley, 1992.