Real Time Fault Injection Using Logic Emulators

Reza Sedaghat-Maman Erich Barke

Institute of Microelectronic Systems
University of Hanover
Callinstr. 34, D-30167 Hanover, Germany
sedaghat@ims.uni-hannover.de

ABSTRACT

A hardware based approach to Fault Emulation independent of the logic emulation system in use has been developed and is presented in this paper. Fault injection into a targetted circuit is made possible with the introduction of additional logic. The stuck-at-fault model is simulated with the method described here, which may also be used for multiple faults and bridging faults [1]. This discussion will focus only on SSF. Using a commercial emulator, speeds for combinational circuits of several MHz were reached allowing for system testing at real time speed.

1. Introduction

At present, fault simulation for large as well as increasingly complex circuits is unrealistic due to extremely long computing times. Various methods exist which attempt to minimize simulation passes by processing faults concurrently, such as parallel [2] and concurrent [3]. However, these methods still do not result in an acceptable reduction in computing time for complex circuits [4,5]. Logic emulation is a new approach to design verification involving the development of a reprogrammable prototype of a digital circuit. In many cases a "real-time" speed which is 10^3 to 10^6 faster than software simulation [6] can be reached with logic emulators. In contrast to logic simulators, a logic emulator can be connected to the target system as a prototype of the digital circuit. With "real-time" fault injection a system can be tested to verify its function and its reaction to the inserted fault. Three methods of fault simulation with logic emulators i.e. Fault Emulation have recently been introduced and are referred to as a new approach to Fault Emulation [7], the Fault Grading Method [8] and Serial Fault Emulation SFE [9]. In contrast to the two latter methods, the technique discussed in this paper involves no reprogramming of the logic blocks and allows a faster fault injection into any node of the circuit without requiring a specific logic emulator or additional hardware.

This paper is organized as follows: In Section 2 an overview of Fault Emulation with Fault Injectors is presented. Section 3 defines the function of the Fault Injectors for stuck-at-faults. Section 4 explains the control of Fault Injectors and the Fault Activator. In Section 5 the Fault Emulation Hardware is described. Section 6 details the experimental results and discusses Fault Emulation runtime. Section 7 concludes this paper.

2. Fault Emulation Overview

As illustrated in Fig. 1, a circuit represented as a netlist is expanded with Fault Injectors. The faultlist for this circuit indicates the nodes where the Fault Injectors should be inserted. Each Fault Injector has a corresponding logical address and is controlled by the Fault Activator.

After the expanded circuit has been mapped both the faulty and the fault-free circuit are present in the emulator. If the Fault Injectors are deactivated the circuit is fault-free indicating a Good Emulation. A Good Emulation (MUX=0) is performed for each test pattern set T and the Good-results are stored. The actual Fault Emulation (MUX=1) starts with setting the first test pattern and injecting the first fault. Then, the same test pattern activates the next fault. This process is repeated until all faults are activated. Next step is the setting of the second test pattern and activation of the first Fault Injector. This procedure is repeated until all test patterns have been set. The Fault Emulation results are compared with the results of a Good Emulation. A test pattern t will detect a fault when the results of a Good Emulation G differ from those of Fault Emulation F ($G_{(t)} \neq F_{(t)}$) [1]. When $G_{(t)} \oplus F_{(t)} = 1$, the fault has been detected and the fault detection counter is incremented. Once the fault list has been processed fault coverage can be calculated. In contrast to previous Fault Emulation approach [7], this method requires no additional hardware modules.

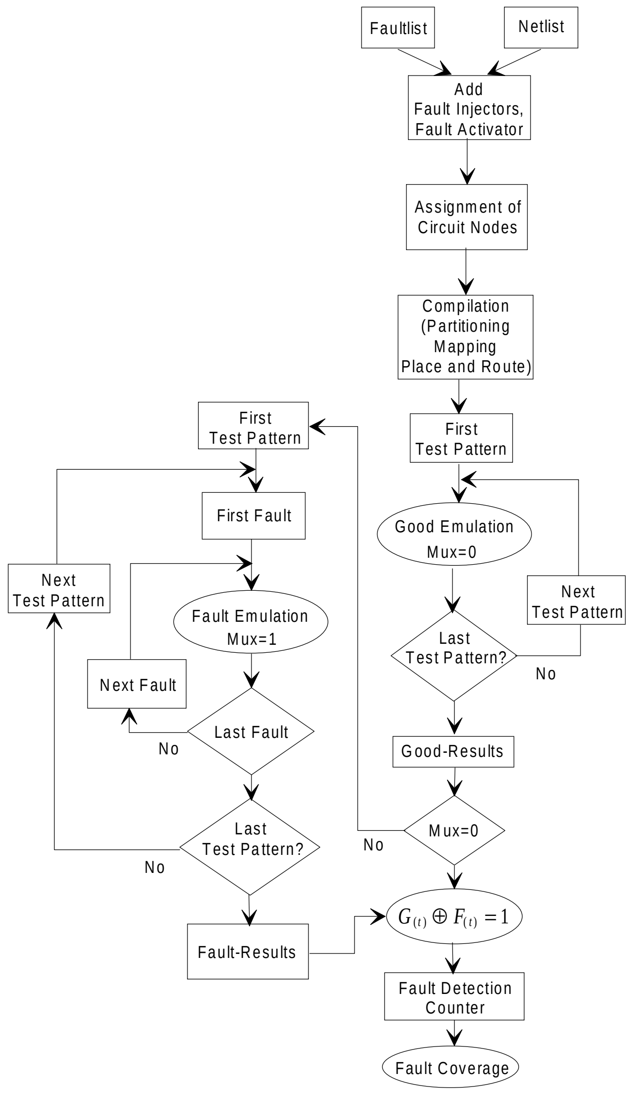
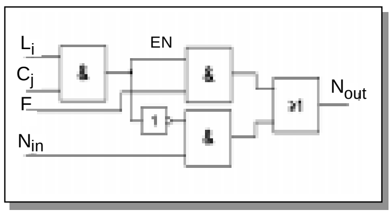


Fig. 1: Fault Emulation Flow Diagram

3. Fault Injectors

Three diferent Fault Injectors [7] are used for modeling stuck-at-zero and stuck-at-one faults. The injection of a fault into a net is done by cutting the net and inserting a Fault Injector. Fault Injectors are controlled by two signals, L_i and C_j; The data in/out variables are N_{in} and N_{out}, representing the net in the circuit into which the fault should be inserted. Activation/deactivation of the injectors is determined by the EN ($EN = L_i \cdot C_j$) signal.

In order to model both stuck-at faults (Fig. 2) an additional variable F is needed. If F=1, then $N_{out} = (L_i \cdot C_j) \cdot F + N_{in} \cdot (\overline{L_i \cdot C_j}) = 1$ and the Fault Injector simulates the stuck-at-1 fault. When F=0, then $N_{out} = (L_i \cdot C_j) \cdot F + N_{in} \cdot (\overline{L_i \cdot C_j}) = 0$ and the stuck-at-0 fault is modelled.



$$N_{out} = EN \cdot F + N_{in} \cdot \overline{EN}$$

Fig. 2: Stuck-at-0/1 Fault Injector

An example of fault injection is illustrated in Fig. 3 and 4. In order to model a Sa-0 fault in net F, the corresponding Fault Injector is inserted. The circuit displayed below has been mapped in a CLB with two LUTs (Look-Up-Table) [11]. Gates G1 and G2 have been mapped with four variables $S_1 = A.B.C.D$ in LUT 1 and G2 with two variables $S_2 = F + E$ in LUT2. As a result of Fault Injection in net F the function of LUT2 is affected such that net F is replaced by the function $F \cdot \overline{L_i \cdot C_j}$. In this case no additional resources are required for the application of Fault Injectors.

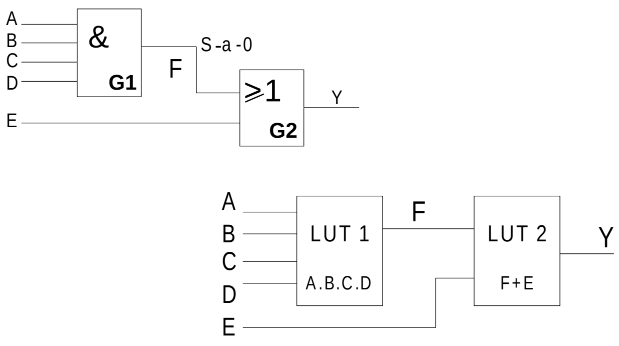


Fig. 3: Example of a Circuit in a CLB

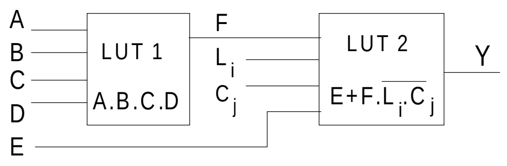


Fig. 4: Fault Injector S-a-0 in a CLB

4. Fault Activator

Fault Injectors are controlled by a Fault Activator [7] whose two-dimensional matrix form allows more Fault Injectors to be addressed with fewer connections. Fault Injectors are addressed and controlled by x- and y-decoders. The matrix consists of a line decoder and a column decoder which can be activated/deactivated with an additional variable *MUX* thereby controlling Fault and Good Emulation.

5. Fault Emulation Hardware

The process of hardware-based Fault Emulation is illustrated in Fig. 5.

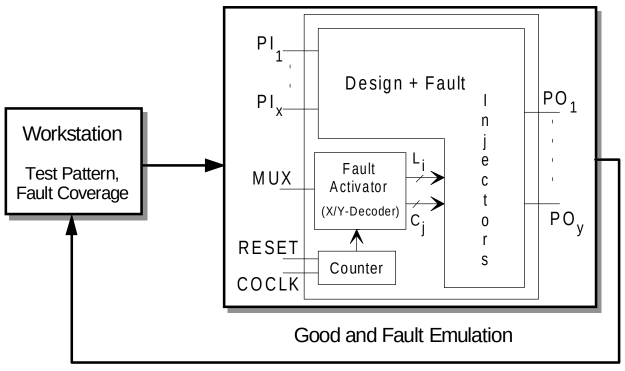


Fig. 5: Fault Emulation Hardware

The complete system consists of emulator hardware and a workstation, which loads the test pattern in the Emulator. After an emulation process the results of a Good Emulation are compared with those of a Fault Emulation for the calculation of fault coverage in the workstation. In our previous approach [7] a fault was activated and all test patterns were examined to determine whether the fault had been detected. The advantage of this method was that after n test patterns the fault emulation process could be interrupted and the next fault activated. In contrast to the previous approach, the complete fault set must be calculated for a single test pattern during the fault emulation process with the logic emulator used for this research. This results in an increase in runtime (see section 6).

The Fault Activator requires three primary input pins: *MUX* (Good/Fault Emulation), *COCLK* (Address Counter Clock) and *RESET* (Address Counter Reset) as shown in Fig. 5. COUNTER is a n+1-bit address counter for the x/y decoder. The signal *RESET* is used to reset the counter before the start of a Fault Emulation.

6. Experimental Results

Fault Emulation results were evaluated using the FPGA (4013 series) from XILINX and Quickturn´s M250 emulation system. Table 1 describes the ISCAS ´85 benchmark circuits and a decoder (c32k) for which a test evaluation was performed. When the circuits were emulated with the M250 the maximum emulation speed of the Logic Emulator (11.12 MHz) was reached.

Table 1: Data and Logic Emulation of Original Circuit

Circuit	# Gate	# CLBs	Freq [MHz]
c1908	880	387	11.12
c2670	1192	598	11.12
c3540	1669	732	11.12
c5315	2307	978	11.12
c6288	2416	1047	11.12
c7552	3512	1562	11.12
32k	32754	7423	11.12

Fault Emulation results are shown in Tables 2 and 3 with stuck-at-0 (Sa-0) and stuck-at-1 (Sa-1) Fault Injectors. Using fault collapsing, Fault Injectors were built into the circuits resulting in a reduction in the number of faults [10].

Table 2: Fault Emulation with Stuck-at-0 Fault Injector

Circuit	# S-a-0	# CLBs	CLB Overhead	Freq [MHz]
c1908	288	539	1.39	11.12
c2670	705	954	1.59	11.12
c3540	980	3097	4.2	11.12
c5315	1353	1982	2	11.12
c6288	5744	4754	4.5	11.12
c7552	1646	2907	1.86	11.12
32k	36569	23146	3.6	11.12
Average			2.71	

Table 3: Fault Emulation with Stuck-at-1 Fault Injector

Circuit	# S-a-1	# CLBs	CLB Overhead	Freq [MHz]
c1908	1396	1309	3.38	11.12
c2670	1781	1667	2.78	11.12
c3540	2040	3234	4.4	11.12
c5315	3550	3435	3.51	11.12
c6288	560	1337	1.27	11.12
c7552	5149	6649	4.25	11.12
32k	36555	22123	3.44	11.12
Average			3.29	

The CLB-overhead of the expanded circuits is illustrated by the ratio #CLB(Sa-0) / #CLB(original) and depends on the number of Fault Injectors i.e. the number of faults and the structure of the circuit. Further examination is necessary to determine which structure parameters influence CLB-overhead. The average overhead for Stuck-at-0 is 2.7 and 3.3 for Stuck-at-1. The emulation speed Freq [MHz] can vary depending on the emulation system in use. Fig.6 portrays the quasi-linear increase in the number of CLBs in the FPGAs in relation to the number of faults in the expanded circuits of Tables 2 and 3.

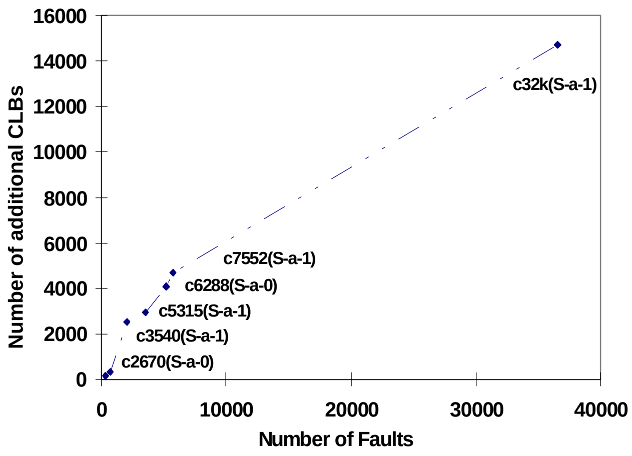


Fig.6: Number of additional CLBs in the FPGAs

The random assignment of circuit nodes to Fault Injectors results in high CLB-usage [11]. Fault Injectors are connected by a two-dimensional array interconnect structure, which reflects the matrix form of the Fault Activator. An optimized assignment of nodes to Fault Injectors is a crucial aspect of CLB-overhead and leads to improved FPGA partitioning, mapping, placement and routing. This is a topic of current research.

The Fault Activator with Fault Injectors is modelled as a two-dimensional *L x C* array [7]. The circuit nodes are assigned to all Fault Injectors where faults are to be inserted such that the neighboring nodes are always mapped to the neighboring Fault Injectors in the Fault Activator.

6.1 Fault Emulation Runtime

With Fault Emulation runtime $RT_{(F)}$ determined by the number of faults *f* and number of test patterns *P*, and Good Emulation runtime $RT_{(G)}$ calculated from the number of test pattern sets *T* and the emulation clock speed *Freq*, the total runtime *RT* can be defined as follows:

$$RT = RT_{(F)} + RT_{(G)} = \frac{(P \cdot f) + T}{Freq}$$

The fault simulator Comsim, a parallel pattern fault simulator from the University of Hanover was also tested with the listed circuits on a Sun workstation (Sparc10 with 512 MByte RAM) using the SSF fault model. Fault Emulation is performed with 20K test patterns. Fig. 7 displays the runtime of fault simulation and Fault Emulation. For Fault Emulation the runtimes for S-a-0 and S-a-1 have been added together. The speedup with the method described here when tested on circuits with 800 to 33k gates varies from 15 to 70.

6.2 Theoretical Complexity

Hardware emulation runtime is independent of circuit size, therefore the experimental results indicate a linear increase in Fault Emulation runtime. The results of fault simulation illustrate a linear to quadratic increase in runtime. With an increase in the quantity of gates, a higher speedup (Fig. 8) of Fault Emulation over fault simulation can be expected.

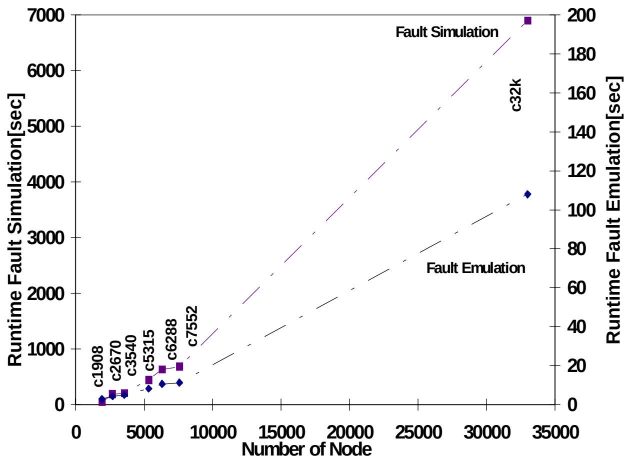


Fig. 7: Runtime of Fault Emulation and Comsim

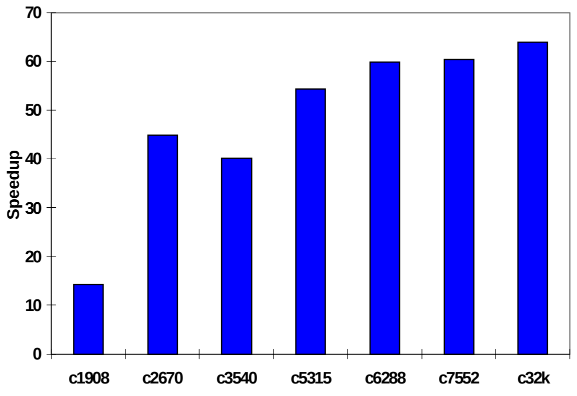


Fig. 8: Speedup of Fault Emulation over Comsim

7. Conclusions

A new Fault Emulation method has been presented in this paper which does not require additional hardware and can be implemented with every logic emulator available today. Fault injection with additional logic affects CLB overhead. Design overhead is not a matter of concern when considering the capacity of present and future logic emulators. As shown, Fault Injectors were easily mapped to FPGAs and primary circuit inputs were minimally expanded by three. The goal for a 1 million gate circuit is to reach a Fault Emulation runtime of several hours. In addition to already known applications of hardware emulators, the technique described in this paper provides a new approach to Fault Emulation and allows for "real-time" fault injection.

References

[1] M. Abramovici, M. A. Breuer, A. D. Friedman "Digital Systems Testing and Testable Design", New York, W.H. Freeman and Company, 1990, p. 131

[2] S. Seshu, "On an Improved Diagnosis Program", IEEE Transactions on Electronic Computers, Vol. EC 14, No. 1, p. 76-79, 1965.

[3] E. Ulrich, T. Baker, "The Concurrent Simulation of Nearly Identical Digital Networks", Proc. of the 10th Design Automation Conference, 1973, p. 145-150.

[4] P. Bottorff, "Test Generation and Fault Simulation", VLSI Testing, North Holland, 1985. p. 29

[5] B. Krishnamurthy, D Harel, "Is There Hope for Linear Time Fault Simulation?", Fault Tolerant Computing Symposium, 1987, p. 30

[6] U. R. Khan, H. L. Owen, J. L. Hughes, "FPGA Architectures for ASIC Hardware Emulator", Proc. 6. IEEE ASIC Conference, 1993, p. 336

[7] R. Sedaghat-Maman, E. Barke, "A New Approach to Fault Emulation", RSP'97 Proc. of the 8th International Workshop of Rapid System Prototyping, 1997, p. 173-179

[8] K. Cheng, S. Huang, W. Dai, "Fault Emulation: A new Approach to Fault Grading", ICCAD, 1995, p. 681-686

[9] L. Burgun, F. Reblewski, G. Fenelon, J. Barbier, O.Lepapa, "Serial Fault Emulation", Proc. of the 33th Design Automation Conference, 1996, p. 801-806

[10] B.R. Wilkins, "Testing Digital Circuits" London, Chapman & Hall, 1994, p. 54

[11] XILINX data book, " The Programmable Logic", 1994