Enhanced Fault Coverage Analysis Using ABVFI

Scott Bingham and John Lach
Charles L. Brown Department of Electrical and Computer Engineering
University of Virginia
351 McCormick Road, Charlottesville, VA 22904-4743
Phone: {434-295-7086, 434-924-6086}
{sfb5y, jlach}@virginia.edu

Keywords: fault injection, fault coverage, formal verification

Abstract

Fault injection is an effective way to determine the fault coverage of an integrated circuit design and is usually accomplished through simulation. Simulation is time intensive, making it impossible to simulate all possible input and fault combinations in a complex circuit under a realistic fault model. ABVFI is a fault injection methodology that uses assertion-based verification (ABV), a variant of model checking analysis, to perform fault injection. Like simulation-based fault injection, ABVFI has high observability and controllability, but is advantageous over simulation because it can prove properties about the design through an exhaustive analysis without prohibitive time and computational requirements. In this paper, we present ABVFI and two major advantages that ABVFI provides over related techniques – the inclusion of fault locations in combinational logic, and the use of multiple fault models during analysis.

1. Introduction

As manufacturing technologies advance to smaller transistor sizes, hardware components become more and more susceptible to single event upsets (SEUs) and other types of failures [7, 17]. This is especially disconcerting for safety- and mission-critical systems, but even consumer electronics and high performance computing systems that have not had to worry about transient faults in the past are now faced with similar concerns.

Traditionally, systems have been hardened against the effects of faults by protecting memory structures through redundancy or data protection techniques. However, with smaller transistor sizes, decreased propagation delays, and higher clock frequencies, the likelihood of an SEU causing an erroneous value to be latched has risen. Studies have shown that the soft error rate (SER) in combinational logic will be nearly equal to that of SRAM at the 50nm technology generation [2, 16]. Furthermore, it is expected that with future technology generations, neutron- and alpha-induced multiple bit upsets will increase [14]. Therefore, attention needs to be given to combinational logic as well as sequential logic during design and verification.

Systems must be analyzed to validate that the fault mitigation techniques are sufficient to achieve a desired dependability. A common method for determining the resiliency of a design against faults is fault injection. Through fault injection, the system is subjected to a series of tests in which perturbations in the system are induced that mimic the behavior of faults. The main way way that fault injection is performed on integrated circuits (ICs) is through simulation-based techniques.

In simulation-based fault injection, signals or elements within the design are altered during the simulation to represent the occurrence of a fault. This provides good controllability and observability of the design, and many fault models can be used, as opposed to post-manufacturing techniques. Furthermore, simulation-based fault injection can be performed earlier in the design process, making design modifications much cheaper.

However, a major drawback of simulation in general is that due to the amount of time it takes to simulate a large design, it is impossible to cover all possible combinations of inputs, and therefore only a subset is tested. This is especially true with simulation-based fault injection due to the increase in state-space [5]. The simulation-based fault coverage analysis is therefore incomplete and must rely on estimates from statistical analysis.

Emerging techniques involve the use of formal methods in the analysis of a design's fault coverage. These techniques replace simulation with formal verification engines, such as model checking, in a fault injection campaign. Like simulation-based fault injection, formal verification-based

fault injection provides high controllability and observability, and can model many types of faults. However, the advantages of using formal methods are that the analysis performs an exhaustive search of the input and fault space, ensuring that no corner case fault goes uncovered, and in less time than an exhaustive analysis with simulation.

This paper presents a methodology we are developing, called ABVFI, that allows fault injection to be performed at the register transfer level (RTL) of a design. ABVFI extends an emerging technique called assertion based verification (ABV), a variant of model checking, to mathematically analyze design behavior in the presence of faults. Given a defined fault model, this method considers all possible combinations of faults across both time (when the fault occurs) and space (where the fault occurs), for a complete analysis of fault coverage.

The following section provides background information on fault injection and model checking techniques. ABVFI is presented in section 3. We describe a case study used to test ABVFI concepts in section 4. Related work is presented in section 5. Major contributions of the ABVFI approach are described in section 6, followed by future work and the conclusion.

2. Background

2.1. Simulation-based Fault Injection

Simulation-based fault injection is performed while running a set of test vectors. At some point during the simulation, the execution is paused, a design signal is overridden with a new value, simulation proceeds, and the design's response is monitored. Generating fault injection tests consists of identifying four parameters: the location of the fault, the time the fault occurs, the duration of the fault, and the design's input stimulus. It should be apparent that the input space for a fault injection campaign is extremely large and accentuates the impossibility of testing all possible fault scenarios with simulation.

While approaches that make use of simulator commands to control fault injection have been proposed [12], the more common approach is through code modification [8, 3]. The design is modified such that faults are injected into the design by intercepting signals or by altering the behavior of a component through modified logic. These actions are performed by saboteurs and mutants [1]. When activated, a saboteur intercepts and alters the value or timing characteristics of a signal. Components are replaced by mutants that behave normally while inactive and abnormally when activated. The flexibility of saboteurs and mutants allows many types of faults to be modeled. The main drawback is that many control signals must be added to the design to control the saboteurs and mutants.

2.2. Model Checking

With model checking, properties the design should adhere to are described in a formal language [9]. These properties describe states and sequences of events that the design "should always" or "should never" reach. For example, a property could state that "signal *a* and signal *b* should never be asserted at the same time." Using mathematical proofs these properties are then verified against a model of the design that describes all the possible states the design can reach. If a property is found to be true, it is an exhaustive proof that a design will always adhere to the property [6].

ABV is a model checking methodology tailored to hardware design verification. With ABV, the HDL code, e.g. Verilog or VHDL, at the register transfer level (RTL) serves as the modeling language. Properties, called assertions, are embedded into the hardware design giving them access to all the signals and variables of interest, providing exceptional observability and controllability.

In addition to the design under test (DUT) and properties, ABV requires an environmental model to be defined for analysis. The environmental model is a definition of the operating environment for the component. For example, when using ABV to analyze a processor, the environmental model would provide a definition of valid instructions for the processor. The assertions are verified against the DUT under these constraints.

3. The ABVFI Methodology

The ABVFI methodology takes ABV and extends it to include fault injection. Essentially, the input space is expanded to include scenarios outside normal operating conditions. The analysis then determines if the design's properties still hold in the presence of faults. Given the fault model defined, this method considers all possible combinations of faults across both space and time, for a complete analysis of fault coverage.

Figure 1 shows the typical steps followed during ABV with the ABVFI augmentation. The HDL design and an operational profile define the DUT and its environmental model, respectively. Properties the design should adhere to are obtained from the specification and converted to property specification language (PSL) assertions. These include properties that define fault coverage. Design extension is the process of making modifications to the test environment such that it is possible to mimic the behavior of faults in the design. Saboteurs are contained in the environmental model that describe the behavior of faults. Modest changes are made to the DUT to enable the saboteurs do inject faults into its logic.

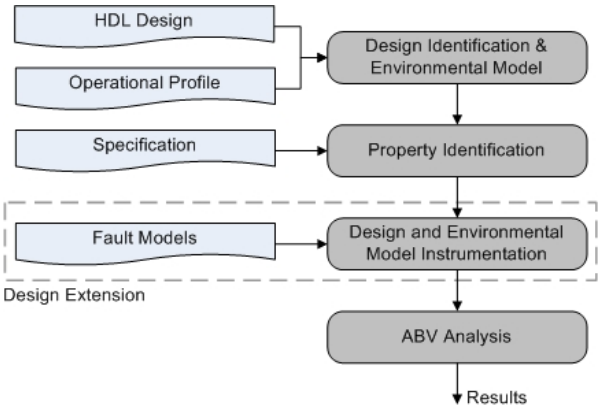
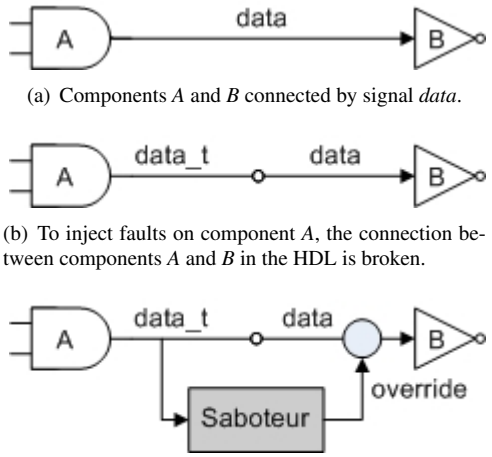


Figure 1. ABV process augmented with the design extension step (shown in the dashed box) for fault injection analysis.



(a) Components *A* and *B* connected by signal *data*.

(b) To inject faults on component *A*, the connection between components *A* and *B* in the HDL is broken.

(c) The saboteur (contained in the environmental model) drives the input to component *B*.

Figure 2. HDL modifications required for design extension.

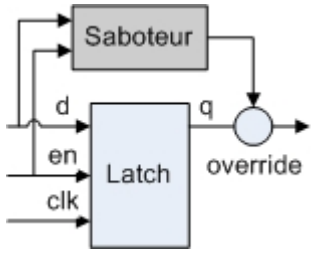


Figure 3. The saboteur overrides the sequential component.

Design extension involves minor modifications to the design HDL (as shown in figures 2 and 3), and fault injection logic is added to the environmental model. In the HDL, signals coming from a component that will be targeted for fault injection are split into two signals, e.g. *data* and *data_t* (see figure 2(b)).

In the environmental model, saboteurs, which are logic modules representing a fault model, are instantiated for each fault location in the design (shown in figure 2(c)) and are responsible for overriding the target signal when a fault is injected.

For sequential components that may hold their value for longer than one clock cycle, a saboteur is used that includes sequential behaviors. In this case the saboteur in the environmental model simply overrides the signal value, and thus does not require any HDL modifications, as shown in figure 3.

We are developing a toolset that automatically instruments VHDL design code and generates the necessary declarations required in the environmental model for the selected fault locations.

After the design and environmental model have been extended to include fault injection logic, the design is then verified against the specified functional and safety properties. The result is either 1) verification that a design will function as specified not only under ideal operating conditions but also under a comprehensive set of faults and fault combinations that could occur, or 2) identification of how a design is susceptible to faults. This is in contrast to traditionally fault coverage that yields a statistical probability.

3.1. Verification and Assessment

ABVFI is an assessment tool for fault coverage. During verification and validation, ABVFI can be used to ensure that the DUT meets its specification, particularly dependability requirements regarding fault coverage.

One benefit of model checking is that when a property is violated, a counter example is produced describing a scenario that led to the property's failure. The counter example can then be used as an aid in determining the vulnerabilities of the design.

ABVFI counter-example results can also be helpful at other stages of system verification. A compilation of uncovered faults and their error behaviors can be used to feed a fault injection analysis at the system level, as shown in figure 4. The benefit of this is that the system level fault list generation can be limited to the known failure scenarios of the low-level components.

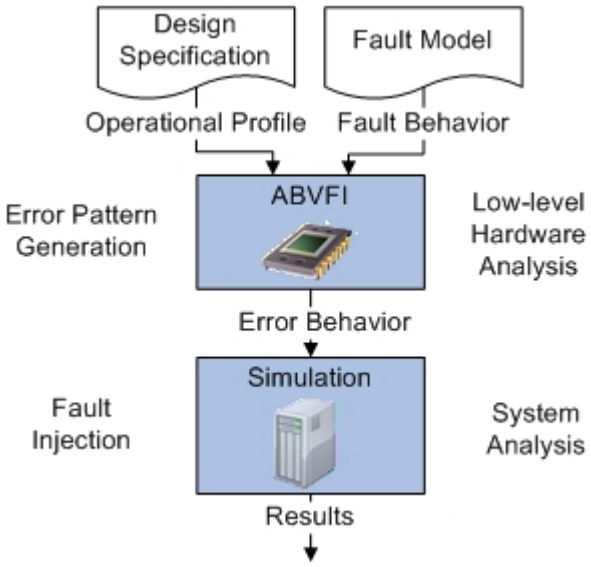


Figure 4. ABVFI within a system-level fault injection analysis.

4. Case Study

We have demonstrated the use of ABVFI on a processor we developed called the PHFT processor. The PHFT processor is a 32-bit, 5 stage, RISC-style, pipelined processor with hazard detection and data forwarding. It is based on the pipelined processor described in [13], but has been modified to include fault tolerant mechanisms designed to mitigate SEUs.

The PHFT processor components can be classified into two categories: 1) the combinational logic elements, which includes the pipeline stages that are responsible for accomplishing work, and 2) the storage elements, which includes the inter-stage latches necessary for storing instruction data between stages.

The storage elements are protected by a redundant register, called the PCDMR register (later described in section 6.2), capable of detecting and correcting an SEU. The combinational elements are dual modular redundant (DMR), but are only capable of SEU detection, and not correction. The detection of a fault in a pipeline stage causes the processor to stall while the fault propagates out of the combinational logic, after which correct execution can resume.

Analysis of the PHFT processor was performed using IBM's Rulebase PE, a commercially available ABV toolset. Each stage of the processor was independently verified using the SEU fault model and fault locations included all combinational and sequential components in the design. The only exception to this is that faults on known single-point-of-failure components, such as voters, were not considered. Assertions were defined for each stage that specify the correct progression of instructions through the pipeline and that faults are detected by the redundant logic.

Table 1. PHFT analysis results.

Processor Stage	Number of Assertions	Total Analysis Time (minutes)
Fetch	3	1.97
Decode	8	2.92
Execute	7	3.12
Memory	2	7.23
Write back	1	0.4
Inter-stage latch	1	6.27
Total	22	21.91

Table 1 shows the results of our analysis. Shown are the number of assertions and analysis times for all five pipeline stages and one 32-bit inter-stage latch.

Although somewhat expected due to the extensive use of redundancy, our analyses showed that all the faults considered are covered correctly. This proves that all possible faults that are represented by the SEU fault model and that occur in the fault locations we considered are properly covered by the PHFT processor. Furthermore, the analysis times were not prohibitive (about 22 minutes), thus ensuring that ABVFI could be adopted into practice.

5. Related Work

The use of formal methods for fault coverage analysis has been proposed previously and such work can be compared by three metrics: the formal model used to represent the DUT, fault models considered, and the locations of injected faults.

The FSAP/NuSMV-SA tool requires that a formal model be manually created that describes the behavior of the system [4]. Because formal models are typically abstract representations, it is difficult to know just how accurately the model describes the dynamics of the real system. Furthermore, developing formal models is challenging due the nature of the formal languages.

[10, 11, 15], and ABVFI all present methodologies that work from the design's HDL, either directly or indirectly. The benefit of this is that there is no need to translate the design into an abstract formal model.

The fault models for the approaches presented in [10, 11, 15] all focus on SEUs. However, ABVFI allows for the possibility to consider multiple types of faults during an analysis. Our approach goes even further by creating a flexible fault model framework, as discussed in section 6.2.

A typical approach to selecting fault locations is to only inject faults on sequential components. [10, 11, 15] follow

this approach, ignoring errors that result from faults that occur in combinational logic. ABVFI considers fault locations in both sequential and combinational logic, as discussed in section 6.1, providing a more accurate analysis.

6. ABVFI Advantages

6.1. Improved Fault Location Analysis

Many fault injection approaches assume that faults occurring in combinational components are unlikely enough that they can be ignored [18]. However, it has been shown that SERs in combinational logic are becoming more significant [2, 16]. Fault tolerance in memory structures will no longer be sufficient, but will also need to be employed in logic blocks as well to mitigate these effects.

It is also important to include fault models for combinational logic because of the degree to which errors can propagate to many locations in the design. Though an SEU may occur in a single location in a design, if the fault occurs in combination logic it can propagate and be latched into multiple bits in registers. Therefore, any analysis that only considers faults in sequential components are inaccurate and most likely optimistic. For these reasons, ABVFI supports fault injection into both combinational and sequential components.

One potential issue that arises with respect to fault injection into combinational logic at the RTL is due to synthesis. During design synthesis, tools convert a design from an HDL into logic components that can be mapped onto a programmable logic device (PLD) or processed further for use in an ASIC. While the overall functionality of the design will not have changed during synthesis, the structure may have. The fault locations identified by ABVFI are made based on the structure defined by the RTL code. Therefore, synthesis can in some ways invalidate the results of combinational logic fault injection analysis. However, this issue can largely be mitigated by the use of structural RTL code, which is much more prescriptive of the actual logic structure than behavioral RTL.

Another consideration is that including combinational logic faults in an ABVFI analysis has the potential to greatly increase the input state-space, and thus have a negative impact on analysis execution times. In order for ABVFI to be adopted into a development process, the analysis times can't be prohibitive. In order to evaluate the additional execution time of fault injection into combinational logic, we ran several experiments on the PHFT Processor.

During our tests of the PHFT Processor's execute stage, which included 230 fault locations, the analysis times increased by 65.5%, from approximately 2 minutes to just over 3 minutes, over the analysis that did not include fault injection. Though the increase was large, the overall analysis time shows that it is feasible to perform an exhaustive analysis of an input space that includes a large number of combinational fault locations.

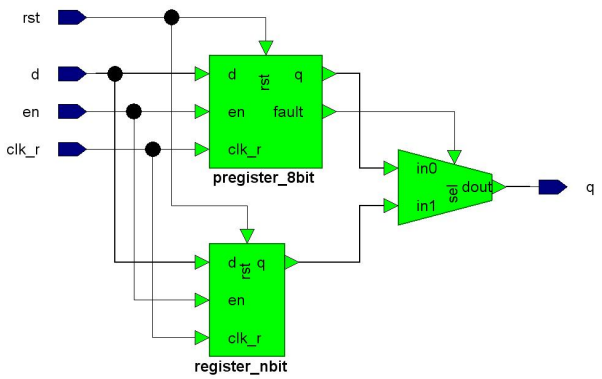


Figure 5. The PCDMR register.

6.2. Flexible Fault Model Structure

Another advantage of ABVFI over related work is the development of a flexible fault model framework for integrating fault models into an ABV analysis. The framework allows us to have a library of fault models that are specific to the manufacturing technology and application environment of the system. It also allows for custom designed fault models to be easily integrated into an ABVFI analysis.

To emphasize the importance of being able to use multiple fault models in an analysis, consider the design of a protected register we developed for the PHFT Processor termed the parity controlled dual modular redundant (PCDMR) register, shown in figure 5. The PCDMR register includes a redundant pair of registers, one of which is protected by a parity bit. A multiplexer, controlled by the parity check logic, selects the final output signal. The PCDMR register is designed to be able to detect and correct one transient SEU.

For this design, fault locations included all the register bits, the parity bit, and the control line for the multiplexor. When this design was tested with ABVFI using the SEU fault model, the functional assertions where shown to hold. However, when extending the fault model to include the possibility of a recurring transient, the analysis failed. The violation occurred when recurring transients occur during a period of time when a new value is not loaded into the register (i.e. the *en* signal is not asserted). It took 99 seconds for the Rulebase tool to verify the assertions under the first fault model, and 36 seconds to find a violation in the second fault model.

7. Current and Future Work

Currently we are in the process of using ABVFI on several case studies. Our case study using the PHFT Processor, mentioned earlier, is nearly complete. We will also be applying ABVFI to an IC design that employs temporal redundancy redundancy and a safety system design used in nuclear power plants to control fuel rod movement. These case studies will provide a broad range of designs to test ABVFI's merits.

Another ongoing task is the development of the automated toolset that implements ABVFI. This is an important objective of this research to make the ABVFI methodology accessible to design engineers, such that it can become a natural part of the development process.

8. Conclusion

Fault injection is an important technique for evaluating a design's fault coverage. Unfortunately, traditional methods involving simulation are unable to fully simulate a complex system.

ABVFI is a methodology that provides a formal analysis of a design's fault coverage. ABVFI improves upon previous work by expanding the fault locations considered during analysis to combinational logic and providing a fault model framework that is flexible enough to include many types of fault models simultaneously. This is important given the new fault models that are emerging with technology scaling. Using ABVFI, designers will have the ability to make important design decisions based on fault coverage data, and verification engineers will be given more accurate error behavior data.

References

[1] J. C. Baraza, J. Gracia, D. Gil, and P. J. Gil. Improvement of fault injection techniques based on vhdl code modification. In *Tenth IEEE International on HLDVT '05: Proceedings of the High-Level Design Validation and Test Workshop.*, pages 19–26, Washington, DC, USA, 2005. IEEE Computer Society.

[2] R. Baumann. Soft errors in advanced computer systems. *IEEE Design & Test of Computers*, 22(3):258–266, 2005.

[3] A. Benso and P. Prinetto, editors. *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation*. Kluwer Academic Publishers, 2003.

[4] M. Bozzano and A. Villafiorita. The fsap/nusmv-sa safety analysis platform. *International Journal on Software Tools for Technology Transfer*, 9(1):5–24, 2007.

[5] J. A. Clark and D. K. Pradhan. Fault injection - a method for validating computer-system dependability. *Computer*, 28:47–56, 1995.

[6] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[7] C. Constantinescu. Impact of deep submicron technology on dependability of vlsi circuits. *International Conference on Dependable Systems and Networks*, pages 205–209, 2002.

[8] E. Jenn, J. Arlat, M. Rimbn, J. Ohlsson, and J. Karlsson. Fault injection into vhdl models: the mefisto tool. In *Twenty-Fourth International Symposium on Fault-Tolerant Computing*, 1994.

[9] C. Kern and M. R. Greenstreet. Formal verification in hardware design: A survey. *ACM Trans. Des. Autom. Electron. Syst.*, 4(2):123–193, 1999.

[10] U. Krautz, M. Pflanz, C. Jacobi, H. W. Tast, K. Weber, and H. T. Vierhaus. Evaluating coverage of error detection logic for soft errors using formal methods. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 176–181, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

[11] R. Leveugle. A new approach for early dependability evaluation based on formal property checking and controlled mutations. In *IOLTS '05: Proceedings of the 11th IEEE International On-Line Testing Symposium*, pages 260–265, Washington, DC, USA, 2005. IEEE Computer Society.

[12] B. Parrotta, M. Rebaudengo, M. S. Reorda, and M. Violante. New techniques for accelerating fault injection in vhdl descriptions. In *IOLTW '00: Proceedings of the 6th IEEE International On-Line Testing Workshop (IOLTW)*, page 61, Washington, DC, USA, 2000. IEEE Computer Society.

[13] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design: The Hardware/software Interface*. Morgan Kaufmann, third edition, 2005.

[14] N. Seifert, P. Slankard, M. Kirsch, B. Narasimham, V. Zia, C. Brookreson, A. Vo, S. Mitra, B. Gill, and J. Maiz. Radiation-induced soft error rates of advanced cmos bulk devices. *Reliability Physics Symposium Proceedings, 2006. 44th Annual., IEEE International*, pages 217–225, March 2006.

[15] S. A. Seshia, W. Li, and S. Mitra. Verification-guided soft error resilience. In *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, pages 1442–1447, San Jose, CA, USA, 2007. EDA Consortium.

[16] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi. Modeling the effect of technology trends on the soft error rate of combinational logic. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 389–398, Washington, DC, USA, 2002. IEEE Computer Society.

[17] N. J. Wang, A. Mahesri, and S. J. Patel. Examining ace analysis reliability estimates using fault-injection. In *ISCA '07: Proceedings of the 34th Annual International Symposium on Computer Architecture*, pages 460–469, New York, NY, USA, 2007. ACM Press.

[18] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. patel. Characterizing the effects of transient faults on a high-performance processor pipeline. In *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks*, page 61, Washington, DC, USA, 2004. IEEE Computer Society.