2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

A Meta-Model-Based Approach for Semantic Fault Modeling on Multiple Abstraction Levels

Michael Schwarz, TU Kaiserslautern, Kaiserslautern, Germany (schwarz@eit.uni-kl.de)

Moomen Chaari, Bogdan-Andrei Tabacaru, Wolfgang Ecker, Infineon Technologies AG, Munich, Germany (*Firstname.Lastname@infineon.com*)

Abstract — Technology scaling has led to devices which are increasingly susceptible to faults arising from manufacturing variations or environmental radiation. Hence, identifying critical weaknesses in SoCs is essential to maintain product quality and robustness. Taking emerging cross-layer dependability approaches into consideration, we propose a new fault-modeling concept for fault injection at the register transfer and system level by adding a new layer of semantic information. The new domain specific models are based on a common meta-model and allow for fault injection with respect to functionality rather than structure. During experimental evaluation of exemplary fault models on a 32-bit processor architecture full coverage was achieved.

Keywords — dependability; semantic; fault injection; cross-layer reliability; meta-model; abstraction;

I. INTRODUCTION

As electronic devices have become crucial in many aspects of modern society, such as mobility, health care and finance, they must meet strict dependability requirements. Higher system-integration density, resulting from technology scaling, and the practice of combining and integrating an increasing number of heterogeneous blocks on a single chip contribute considerably to an overall functionality and quality improvement of electronic products. The downside is a tremendously more complex design and verification task.

Supposing a bug free design, a computing system may still fail to provide its correct service. Mainly, this is due to modern transistors being increasingly prone to failures caused by manufacturing variations, degradation and environmental radiation. With a rising transistor count per design, it is reasonable to assume that transistor failures will be a common event. Prospectively, this increasing unpredictability must be addressed in order to mitigate these failures, maintain the quality and lifetime currently demanded in safety-critical or economically decisive applications as well as to uphold customer satisfaction. To effectively do so, identifying critical areas in the design is paramount. In order to gauge the impact of these on the quality of service provided by the system, the concept of dependability is generally applied, depicting the measure of trust which can be justifiably placed in it. It has grown to be an integrating concept encompassing several attributes such as reliability, availability, and safety, which have to be evaluated by qualitative or quantitative means under the occurrence of threats [1].

Fault injection is a technique that has been widely used over the last decades in order to assess the dependability of a system under test. The basic principle of it is the deliberate insertion of faults into a system whose behavior is then monitored in order to determine its response to the introduced faults. Divergences in the system's behavior are identified via comparison with the previously monitored behavior of a fault free system under the same workload [2]. Once critical fault weaknesses are detected, countermeasures must be taken. While correcting an error directly at the level of its occurrence may be possible, it may not be the most efficient mean. Hence, it is necessary to be able to model and evaluate faults and errors at and across multiple abstraction layers to preserve performance.

Therefore, the goal of this paper is to present a structured approach to define fault models at various higher abstraction levels. These can be incorporated earlier into the design process, reducing time-to-market and allow an evaluation of the effectiveness of countermeasures implemented at their respective abstraction level.

The paper is further structured as follows. Section II elaborates on the concept of abstraction and its significance in fault modeling. In section III, the meta-model-based fault-library concept as well as a set of example models are presented. Section IV gives an application example and the corresponding results. Related work is shown in section V. Finally, the paper's conclusions are stated in section VI.

II. ON THE SUBJECT OF ABSTRACTION

The definition of abstraction depends on the context where it is used. In the context of our paper abstraction is defined as the action of factoring out unnecessary details, identifying relevant similarities between objects and synthesizing those facts into a concept or class. This concept acts as a super-categorical noun for all subordinate concepts and connects any related concepts as a group, field, or category. The primary goal of abstraction is to get a better understanding of complex structures and to simplify the tasks that must be performed on them. This in turn leads solving problems in less time and less cost. Roughly speaking, abstraction is the process which allows us to consider what is relevant and to forget a lot of irrelevant details that will otherwise get in the way. Ironically, reducing the characteristics of abstraction in this particular way already constitutes an abstraction by itself.

Especially noteworthy in the previous definition are the terms relevant, unnecessary and/or irrelevant. Their interpretation is the sole guideline of the abstraction process, thus effectively determining its course and outcome. Unfortunately, this interpretation highly depends on the targeted usage of the concepts. Therefore, abstraction is a task that can only be performed manually, due to the inability of machines to discern between necessary and negligible information without a specific, predefined metric. In case such a metric exists, the actual abstraction is performed during its creation, not its application.

When abstraction is performed, it is done so with a given field of application in mind, e.g., a simplified representation and design of logic circuits by modeling physical transistors as mere switches. Such a field of application is further referred to as a domain. A domain can be subdivided into a multitude of new domains, each being the product of a new layer of abstraction, e.g., physical transistors to logic gates, logic gates to synchronous digital circuits, and those to functionally dedicated hardware like memories or processors. This is graphically represented in Figure 1. As a consequence of this process, the domain width, i.e., the number of domains at one level of abstraction, increases. This makes it much harder to cover an abstraction level with a set of fault models that is on one side universally applicable, and on the other specific enough to provide information detailed enough to be of significance. A possible approach to address this issue is to provide a set of fault models which is tailored specifically to the requirements of a certain domain, e.g., arithmetic errors in the datapath of a processor. Restricting the application of such a fault library to single domain allows thus for a much higher granularity of the defined models.

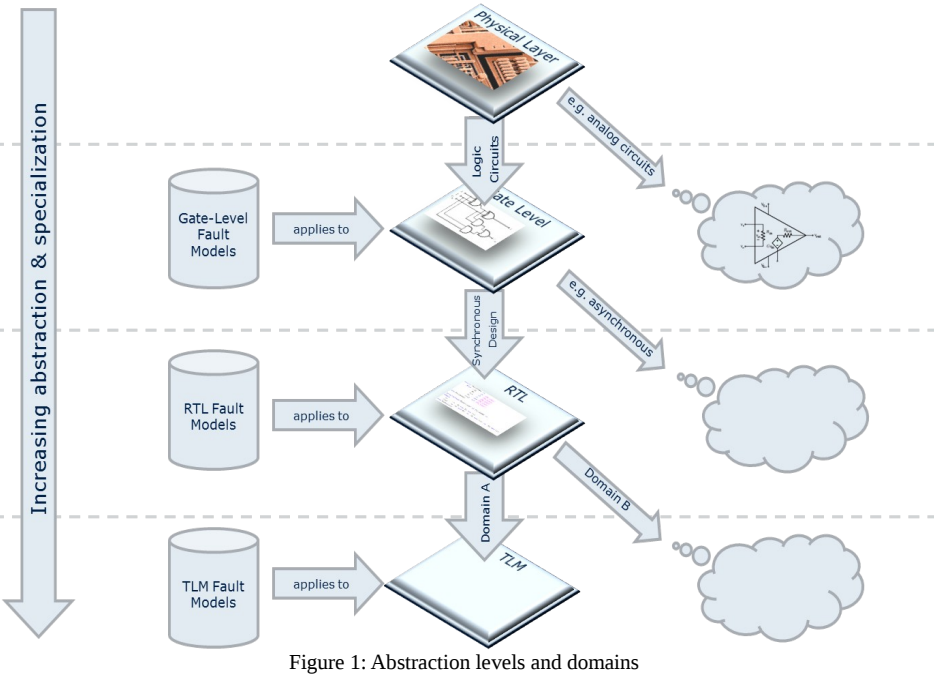

Figure 1: Abstraction levels and domains

III. THE LIBRARY AND EXAMPLES

To promote reuse and facilitate the generation of such libraries, a common ontological meta-model is used. Ontological meta-modeling is concerned with describing which concepts exist in a certain domain, and what properties they have. This is done via ontological instance-of relationships, which enable the definition of domain meta-types [3]. To be widely applicable, the meta-model must be as abstract as possible, such that it can serve as foundation for libraries targeting not only different domains, but also different levels of abstraction.

The proposed library meta-model is build using the UML standard, as it is an essential standard in the visualization, storage and exchange of software designs and models, that is widely used in today's industry. It is structured as shown in Figure 2. The library class represents the root-node of the meta-model. A library must be named and may contain an arbitrary number of library elements. A library element can be either one of the following classes: threat, target, and domain.

- *Threat:* The threat class is used to specify the catalog of faults, errors and failures deemed necessary by the library creator. A threat might be constrained to domains via the *at* association and/or a subset of the defined targets via the *has* association.
- *Target:* Objects instantiated from the target class represent valid application points for threats. A target is associated to some domain, due to otherwise being possibly ambiguous .
- *Domain:* Instances of this class depict different areas of interest and or functionality. The domain determines in particular which systems are covered by the library and specifies the viable targets via the contains association.

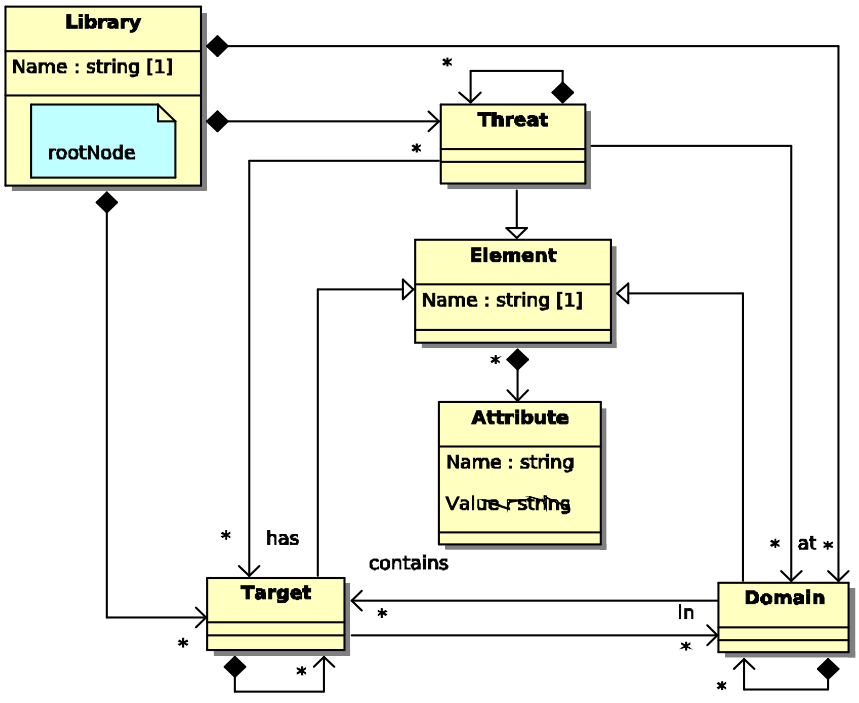

Figure 2: Fault model library meta-model

Each instantiated object of one of these classes has to be identifiable by a name, which should reflect its purpose, and might be characterized by an arbitrary number of attributes. Furthermore, each element is self-aggregating, i.e., it can be an aggregation of elements of the same class. The attribute class is a simple pair of two strings, the first representing the name or identifier of this attribute, the second specifying its value. The way this value is represented and interpreted is to be defined by the library creator, giving him nearly complete freedom in his decisions, enabling him to tailor the attributes according to the intended application. Table I and II show excerpts of targets and threats defined in an exemplary fault model library designated for RTL.

Table I: Target Examples

Name	Description
Address	The bits representing a memory or register address
Clock	The clock signal of system / module
Control	A register or bits controlling the program flow, e.g. instruction counter, zero bit etc.
Guard	Guard bits, e.g., parity, overflow etc.
Operation	The bits specifying the operation performed by an instruction

Table II: Threat Examples

Name	Description
Clock Glitch	Rapid triggering of the clock signal for a short period of time
Sign Error	Bit flip of the sign bit of a signed number
Address Error	Bit flips changing a memory or register address
Guard Error	An arbitrary number of bit flips at a guard target

Figure 3 and Figure 4 depict the structural hierarchy of targets and domains, respectively. The associations *in* and *contains* are not shown.

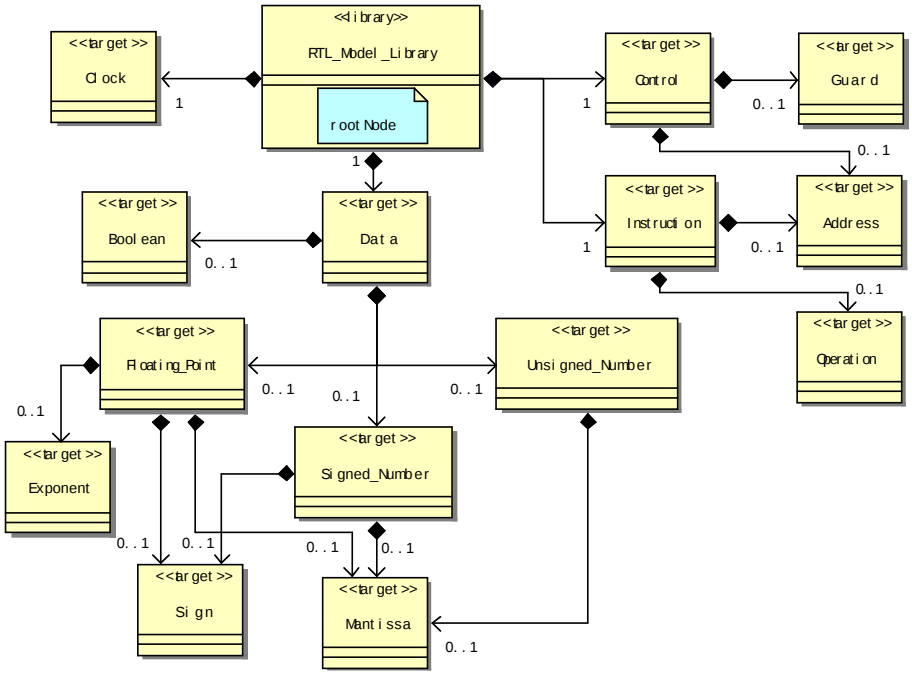

Figure 3: RTL target example structure

A simple example of a fault model is the sign error in an adder. The model itself specifically targets the sign bit in its functionality and is thus somewhat detached from the actual implementation of the hardware. Hence, the bit-width of the adder and other implementation details, such as using little or big endian encoding do not affect the model. Thus, once a plan for a fault injection campaign has been created, it can be easily ported to another architecture.

The construction and development of the meta-model-related frameworks are performed in Metagen, Infineon's meta-modeling and code generation environment. Based on the concepts of model-driven development, Metagen is a meta-modeling development framework which offers highly-automated code generation capabilities using model APIs and text templates. It can be defined as a “design building box for a meta-modeling infrastructure”, replacing classic manual typing of expected target code by a three-step approach:

Designing a meta-model, implementing a specification reader to fill the model (instance of the meta-model) and writing a corresponding code generator called template [4]. Further application examples of Metagen and meta-modeling can be found in [5],[6],[7] and [8].

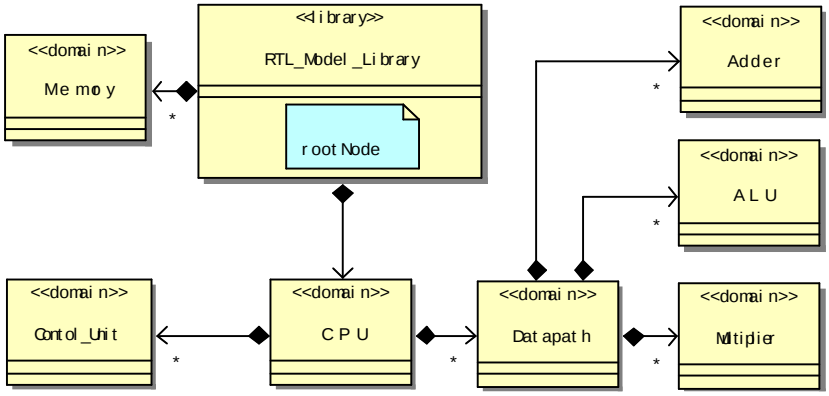

Figure 4: RTL domain example structure

IV. Application Example and Results

The experimental setup here is that of a normal fault injection campaign at gate level. As target system served an implementation of the widely known MIPS-architecture. The implementation was written in VHDL and thus being described at the register-transfer level. The gate-level description was synthesized using the Synopsys Design Compiler [9]. Each module of the design was subject to an independent fault injection campaign, followed by a combined campaign of the whole processor. The workload for each campaign was provided by a set of uncorrelated linear feedback shift registers (LFSR) instantiated in an automatically generated VHDL-testbench.

The simulations were performed in QuestaSim [10], injecting a single fault per run in the design by forcing a random internal signal at a random point of time to either a logical one or zero. An initial run without an injected fault serves as golden reference. Each of these fault-injection campaigns consists of up to 300 unique faulty runs, each being simulated for a simulation time of 1 ms at a frequency of 20 MHz, effectively monitoring 6 million cycles per campaign. Any observable deviation from the output behavior of the golden reference is logged with the attempt to categorize it accordingly to the respective fault library.

During all campaigns, no erroneous situation that is not covered by the defined fault models is observed. This is a promising sign for the coverage of the proposed library as well as the underlying creation methodology.

V. Related Work

Fault injection has been an established method in dependability analysis at the logic gate-level. In order to improve performance, increasing the level of abstraction seems a logical step. Thus, there exist a number of other works are already proposing different frameworks and fault models.

As observed in [11], the increasing independence of higher abstraction layers from the actual details of implementation and technology make it difficult to define such models. The most common methods applied are thus finding abstraction-level equivalents of the single bit stuck-at fault or creating implementation specific libraries of physical induced faults. A promising approach is to simulate the target system at RTL with exception of the module were the fault is injected. This module is, by invoking a respective tool, simulated with gate-level accuracy [12]. In the same work, the author shows that the single-bit stuck-at, applied at RTL level, is not able to fully capture the behavior of failures resulting from gate-level faults.

The recently proposed concept of the Resilience Articulation Point (RAP) model [13] is intended to serve as the logical interface point for resilience analysis between higher levels of abstraction and lower levels of system implementation. RAP assumes that physically induced faults at the technology or CMOS device layer will

eventually manifest themselves as a flip of a single or multiple bit(s). The provided framework is based on probabilistic error functions, which are specific for a given signal. These bit flip probability functions can further be accumulated to higher level functions. In combination with the Reliability Information Interchange Format (RIIF) [14], IP can be shipped with accurate reliability information for seamless integration for dependability analysis in new designs.

VI. CONCLUSION

In order to enhance the capabilities of fault injection at the more abstract layers of RTL and TLM a systematic approach for the definition of a new type of fault models was proposed. The implicit assumptions made during the act of abstraction were hereby identified as a kind of specialization, limiting those models to certain domains. The bit-flip was chosen as smallest observable change, acting therefore as resilience articulation point for the creation of accumulated models of higher order. By this it is compatible with RAP and can be used in conjunction with it. A novelty of these models is, that they are defined by their semantics rather than their structural build, adding a new layer of information to the analysis. At this time, this layer can be utilized in fault injection to identify injection points and target functionalities of a system rather than its underlying structure, simplifying testing for specific errors as well as portability to other architectures. In addition to this semantic aspect, the meta-modeling-based approach enables formalizing the definition of the semantic fault models and describing the correlations between the different abstraction layers. While this is already possible in a bottom-up fashion, future research will look into the possibility of correlating faults in higher abstraction layers with faults in lower abstraction layers in order to gather reusable information from fault injection campaigns, effectively reducing simulation effort spent at RTL and gate level.

VII. ACKNOWLEDGMENT

This work is partly supported by the German Federal Ministry of Education and Research (BMBF) within the project EffektiV under contract no. 01IS13022.

REFERENCES

[1] Avizienis, A.; Laprie, J.-C.; Randell, B.; Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. Dependable and Secure Computing, IEEE Transactions on, 1(1):11–33.

[2] Hsueh, M.-C.; Tsai, T. K.; Iyer, R. K. (1997). Fault injection techniques and tools. Computer, 30(4):75–82.

[3] Atkinson, C.; Kuhne, T. (2003). Model-driven development: a metamodeling foundation. Software, IEEE, 20(5):36–41.

[4] Ecker, W.; Velten, M.; Zafari, L.; Goyal, A. (2012): Metamodeling and code generation - The Infineon approach, ESWeek Workshop MeCoES, S. 1–4.

[5] Ecker, W.; Velten, M.; Zafari, L.; Goyal, A. (2014). "The metamodeling approach to system level synthesis," Design, Automation and Test in Europe Conference and Exhibition (DATE), vol., no., pp. 1(2):24-28

[6] Ecker, W.; Velten, M.; Zafari, L.; Goyal, A. (2014). Metasynthesis for Designing Automotive SoCs. DAC '14 Proceedings of the 51st Annual Design Automation Conference, 71(1):1-6

[7] Chaari, M.; Ecker, W.; Kruse, T.; Tabacaru, B.-A. (2015). Automation of Failure Propagation Analysis through Metamodeling and Code Generation. ITG/GI/GMM-Workshop Testmethoden und Zuverlässigkeit von Schaltungen und Systemen (TuZ).

[8] Chaari, M.; Ecker, W.; Novello, C.; Tabacaru, B.-A.; Kruse, T. (2015) A Model-Based and Simulation-Assisted FMEDA Approach for Safety-Relevant E/E Systems. In Proceedings of the 52nd Annual Design Automation Conference, pages 1–6. ACM.

[9] Synopsys Inc., Design Compiler 2010 (2010), Retrieved April 30, 2015, from Design Compiler 2010. (2010, January 1). Retrieved April 30, 2015, from http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DesignCompiler/Pages/default.aspx

[10] Mentor Graphics Corporation. Questa® SIM User's Manual 10.1c, 2012.

[11] Bengtsson, T.; Kumar, S.; Bengtsson, T.; Kumar, S. (2004). A survey of high level test generation methodologies and fault models. Technical Report ISSN 1404-0018;04:5, Jönköping University, School of Engineering.

[12] Li, M.-L.; Ramachandran, P.; Karpuzcu, U. R.; Hari, S. K. S.; Adve, S. V. (2009). Accurate microarchitecture-level fault modeling for studying hardware faults. In HPCA, pages 105–116. IEEE Computer Society.

[13] Herkersdorf, A.; Aliee, H.; Engel, M.; Glaß, M.; Gimmler-Dumont, C.; Henkel, J.; Kleeberger, V. B.; Kochte, M. A.; Kühn, J. M.; Mueller-Gritschneder, D.; Nassif, S. R.; Rauchfuss, H.; Rosenstiel, W.; Schlichtmann, U.; Shafique, M.; Tahoori, M. B.; Teich, J.; Wehn, N.; Weis, C.; Wunderlich, H.-J. (2014). Resilience articulation point (rap): Cross-layer dependability modeling for nanometer system-on-chip resilience. Microelectronics Reliability, 54(6-7):1066 – 1074

[14] Evans, A.; Nicolaidis, M.; Wen, S.-J.; Alexandrescu, D.; Costenaro, E. (2012). Riif - reliability information interchange format. In On-Line Testing Symposium (IOLTS), 2012 IEEE 18th International, pages 103–108.