A Homogeneous Framework for AMS Languages Instrumentation, Abstraction and Simulation

Enrico Fraccaroli, Luca Piccolboni and Franco Fummi
Department of Computer Science, University of Verona, Italy, name.surname@univr.it

***Abstract*—In the last years an inversion of trend has brought new interest to the analog domain. Its integration within modern digital designs has led to the birth of the so called Analog and Mixed-Signal (AMS) systems. Functional safety assessment of such systems must be evaluated by instrumenting both the analog and digital parts. Such an activity can be simplified if these parts can be considered as an unique layer. Based on such an idea, this work brings them to a common ground by unifying the description language. Such a process is performed through settled procedures that abstract the AMS description to a common abstraction level (behavioral) and to a homogeneous high-level language (C++). This provides a speedup of two orders of magnitude in the fault simulation of an AMS platform.**

***Index Terms*—Homogeneous fault injection, fault coverage, digital, analog and mixed-signal, safety**

I. Introduction

Nowadays, complex analog and digital systems are largely used in safety-critical domains. Functional safety of these systems is provided through Analog and Mixed-Signal Systems-on-Chip (AMS-SoCs) very often composed of many third-party Intellectual Property (IP) cores [1]. As a consequence, ensuring dependability of such systems is becoming a complex and critical task, especially when dealing with safety critical systems [2]. It is needed an effective and reliable procedure which can assess their correctness. One solution is to move the safety verification to a higher abstraction level and earlier in the development process [3]. This would enable the use of well-established procedures to coordinate the processes of fault injection. For *digital* systems solutions can be found at different levels of abstraction, *e.g.*, Gate-Level [4], Register-Transfer Level (RTL) [1] or also at Transaction-Level Modeling (TLM) [2]. For *analog* systems the main challenges are, the lack of a standardized analog fault model [5] and the high complexity of performing analog fault testing. This work proposes a methodology for functional safety assessment at high abstraction level, taking into account the growing complexity and the high heterogeneity of modern safety-critical systems. It focuses on AMS descriptions where analog and digital sub-parts are described respectively at RTL and circuit level as shown in Figure I. The contributions of this work are: ① an innovative method of code instrumentation for high-level digital descriptions and ② a unified flow for the homogeneous fault simulation of AMS descriptions.

II. Homogeneous Fault Simulation

The flow of the proposed methodology is depicted in Figure I. First, the heterogeneous AMS descriptions are transformed into a homogeneous behaviorally equivalent one, then the code is injected with saboteurs and mutants.

A. Analog abstraction

The methodology proposed in [6] has been used to perform this step. Its aim is the abstraction of analog models for their integration inside virtual platforms for smart systems. It takes as input an AMS model described by means of differential equations and it generates as output a simplified model described with a high-level language (*e.g.*, C++). Such a model is able to reduce the simulation complexity by preserving only the input/output relation describing the behavior of the original AMS description. All the other internal details are not evaluated, since they are not visible from outside the model and not necessary to ensure its correctness. As such, the abstraction process generates a high-level description that is a signal-flow representation of the original model. By referring to the work in [6] the following assumption can be made:

Assumption I (Analog behavioral equivalence)**.** *The description generated by the analog abstraction process preserves the behavior of the original description w.r.t. the values of interest.*

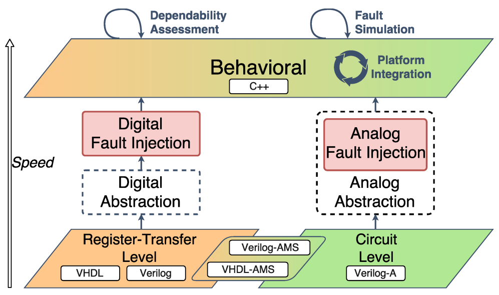

Figure I: Overview of the injection methodology.

B. Digital abstraction

The digital abstraction flow first transforms the description from RTL to a behavioral equivalent model by using the state-of-the-art methodology originally proposed in [7]. The input description can be written with an arbitrary Hardware Description Language (HDL). First, the HDL data types contained inside the description are replaced by standard and much more efficient C++ built-in data types. Then, an analysis of the dependency graphs of the digital processes is performed in order to infer their activation order. This allows to implement an event-driven simulation directly inside the C++ model. The proof that the RTL faults are preserved into the model generated by abstraction is given in [8]. In particular the following assumption can be made:

Assumption II (Preservation of faults behavior)**.** *For all the sets of event sequences inputs, the behavior of the faulty RTL model obtained through injection is equal to the one of the faulty TLM model generated after the abstraction.*

C. Analog fault injection

As shown in Figure I, the process of analog fault injection is highly integrated inside the abstraction process. The process of analog injection has been extensively presented in [9]. Analog faults can be injected inside a Verilog-AMS model by adding new differential equations or by drastically altering an existing process parameter. Given the Assumption I, the physical values

```
#define MUTATE_8u(I, RHS)\
  ((RHS & ~masks[I])|(−stuckAt & masks[I])) & uint8_t(255U)
```

Listing I: Mask for unsigned eight bit variable.

associated with the injected fault must be considered as values of interest. This is imperative in order to preserve the faulty behavior through the abstraction process. In conclusion, *the proposed approach can use every type of fault that can be modeled by means of new equations or by manipulating the existing ones*. Once the type of fault models has been chosen, the injection process iterates over all the possible fault locations depending on the type of fault. For each location it executes two actions:① injects the equation that models the chosen fault and ② executes the analog abstraction. At each new iteration, the previous injected equation is removed from the model, otherwise the process would produce a design that contains multiple faults. At the end of the process, a series of abstracted faulty descriptions are produced. Also the fault-free description goes through the abstraction process.

D. Digital fault injection

The digital fault injection instruments the code with mutants that simulate fault models at the bit level (*i.e.*, stuck-at bit). The output description contains all the mutations that are able to reproduce the faulty design behavior. A previous methodology presented in [8] proposed to inject mutants by means of mutation functions. The first source of overhead is due to the time required to execute the function call. Furthermore, such functions must be called each time an event is generated in the corresponding logic and thus many times. This is another source of overhead. All these operations could be avoided by means of the function-like macros provided by C++. The technique developed in this work takes advantage of this feature and it assigns a bit mask to each mutation location identified by means of an *index*. When a mask is set to the value 0 the corresponding location is fault-free. If a mask has a bit set to 1, then the mutation location has that bit mutated. This mechanism is able to simulate the stuck-at behavior by means of bitwise operations that apply the mask to the original value. Bitwise operations are much more faster than a function call and can be easily used with macros. Listing I shows an example of mask which takes two parameters: the **I**ndex of the mutation location and the **R**ight-**H**and **S**ide original value. `stuckAt` is a global variable which indicates which value of stuck-at must be applied (*i.e.*, zero or one).

E. Homogeneous fault simulation

After both the analog and digital sub-parts have been abstracted and injected with all the faults, the resulting description is enriched with an array of masks. Each mask is mapped to a mutation location and progressively associated with the corresponding index. For what concerns the analog faults, each set of equations produced by the abstraction process is associated with a progressive index starting from the last one used for indexing the digital faults. This allows to enable both digital and analog faults without discrimination.

III. Experimental results

A complex hardware description has been identified with the purpose of showing the effectiveness of the proposed approach on a real case study. The design comprises the hierarchical Verilog description at RTL of the 8-bit microcontroller *M6502* and an *Accelerometer* written in Verilog-AMS. The total number of digital faults injected in every version is 4151, among 536 mutation locations found. For the analog sub-part, the choice of fault models fell on *open* and *short* circuits. The total number of analog faults is 50. The evaluation of the diagnostic coverage metric exploits three safety mechanisms:

- *Hardware (Watchdog)* – It checks if the CPU executes the workload in a fixed number of clock cycles.
- *Software (FIR Filter)* – It checks if the computation is correct by comparing each value generated by the program with a reference value known a priori.
- *Software (Analog)* – It is a Mixed-Signal Built-In Self-Test (MS-BIST) that checks whether the accelerometer is behaving correctly or not.

Table I: Simulation times for the diagnostic coverage evaluation.

Code Version	Times			SpeedUp (x)
Verilog-AMS	532h	48m	31.9s	reference
Functions	6h	12m	45.0s	85.7
Masks	5h	9m	49.8s	103.1

The execution times of this experiment are reported in Table I. Let us define the following set of variables and their values:

$$\begin{aligned}\lambda_{WD} &= \text{\#errors flagged by Watchdog} &&= 1641\\ \lambda_{FIR} &= \text{\#errors flagged by FIR filter} &&= 22\\ \lambda_{MSBIST} &= \text{\#errors flagged by MSBIST} &&= 46\end{aligned}$$

The Diagnostic Coverage metric evaluated for each safety mechanism [10] is as follows:

$$\begin{aligned}DC_{WD} &= \frac{\lambda_{WD}}{\text{\# faults}} &= \frac{1641}{4151} &= 0.3953\\ DC_{FIR} &= \frac{\lambda_{FIR}}{\text{\# faults}} &= \frac{22}{4151} &= 0.0052\\ DC_{MSBIST} &= \frac{\lambda_{MSBIST}}{\text{\# faults}} &= \frac{46}{50} &= 0.9200\end{aligned}$$

While the overall Diagnostic Coverage is:

$$DC = DC_{WD} + DC_{FIR} + DC_{MSBIST} = 1.3205$$

IV. Concluding remarks

An innovative automatic flow has been proposed that abstracts the whole AMS system to behavioral level and then performs a fault injection campaign. Results show how this flow can dramatically reduce the efforts needed to assess the functional safety of an analog mixed-signal platform.

Acknowledgments: We would like to thank Nicoló Danzi for an initial analysis and implementation of the framework.

References

[1] R. Mariani, G. Boschi, and F. Colucci, "Using an innovative SoC-level FMEA methodology to design in compliance with IEC61508," in *Proc. of IEEE/ACM DATE 2007*, apr, pp. 492–497.

[2] Y. Y. Chen, C. H. Hsu, and K. L. Leu, "SoC-level risk assessment using FMEA approach in system design with systemC," in *Proc. of IEEE SIES 2009*, pp. 82–89.

[3] A. Sherer, J. Rose, and R. Oddone, "Ensuring functional safety compliance for ISO 26262," in *Proc. of IEEE/ACM DAC 2015*, pp. 1–3.

[4] A. Fin and F. Fummi, "A VHDL error simulator for functional test generation," in *Proc. of IEEE/ACM DATE 2000*, pp. 390–395.

[5] M. Soma, "Challenges in analog and mixed-Signal fault models," *IEEE Circuits and Devices Magazine*, vol. 12, no. 1, pp. 16–19, 1996.

[6] E. Fraccaroli, M. Lora, S. Vinco, D. Quaglia, and F. Fummi, "Integration of mixed-signal components into virtual platforms for holistic simulation of smart systems," in *Proc. of IEEE/ACM DATE 2016*, pp. 1586–1591.

[7] S. Vinco, V. Guarnieri, and F. Fummi, "Code Manipulation for Virtual Platform Integration," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2694–2708, sep 2016.

[8] N. Bombieri, F. Fummi, and V. Guarnieri, "FAST: An RTL fault simulation framework based on RTL-To-TLM abstraction," *Journal of Electronic Testing: Theory and Applications*, vol. 28, no. 4, pp. 495–510, 2012.

[9] E. Fraccaroli and F. Fummi, "Analog Fault Testing Through Abstraction," in *Proc. of IEEE/ACM DATE 2017*, pp. 1–4.

[10] I. E. Commission *et al.*, "IEC 61508 : Functional safety of electrical/-electronic/ programmable electronic safety-related systems," pp. 7–13, 2010.