High-level fault injection to assess FMEA on critical systems

Julie Roux, Vincent Beroulle, Katell Morin-Allory, Régis Leveugle, Lilian Bossuet, Frédéric Cézilly, Frédéric Berthoz, Gilles Genévrier, François Cerisier

Version of Record: https://www.sciencedirect.com/science/article/pii/S0026271421001013
Manuscript_54d9acc784e9e8f65eb0e168c240f813

High-Level Fault Injection to Assess FMEA on Critical Systems

Julie Roux[1, 2], Vincent Beroulle[1], Katell Morin-Allory[2], Régis Leveugle[2], Lilian Bossuet[3], Frédéric Cézilly[4], Frédéric Berthoz[4], Gilles Genévrier[4], François Cerisier[5]

[1]*Univ. Grenoble Alpes, CNRS, Grenoble INP*, LCIS*, 26000 Valence, France, firstname.name@univ-grenoble-alpes.fr
[2]*Univ. Grenoble Alpes, CNRS, Grenoble INP*, TIMA*, 38000 Grenoble, France, firstname.name@univ-grenoble-alpes.fr
[3]*Laboratoire Hubert Curien, Université de Lyon*, 42000 Saint Etienne, France
[4]*THALES*, 26000 Valence, France
[5]*AEDVICES*, 38430 Moirans, France
*Institute of Engineering Univ. Grenoble Alpes

***Abstract*—Embedded systems in critical applications are constrained by very strict standards, but safety analysis (*e.g.*, Failure Mode and Effects Analysis, or FMEA) of these systems is often empirical and mainly relies on the experience of engineers. Performing empirical analyses on complex designs is a major challenge that leads engineers to make very pessimistic assumptions and consequently to over-design multiple countermeasures. Many fault injection techniques have been developed to evaluate the robustness of Register Transfer Level (RTL) hardware designs. With these techniques, robustness is evaluated by comparing the faulty circuit outputs with the circuit specifications or golden RTL fault-free simulations. However, these techniques are too circuit centered, and therefore do not account for the overall system specifications. In addition, with complex hardware designs, fault simulations become very time-consuming. In this paper, we present a new high-level fault injection approach taking into account the overall critical system specifications to extract acceptable circuit parameter ranges while speeding up the evaluation process. We describe a case study of a real airborne system. The critical parameter ranges are determined for the circuit. Then, these ranges are used to rapidly evaluate the robustness of each RTL block in the circuit.**

***Index Terms*—safety, fault simulation, SystemC, TLM, embedded system, FMEA**

I. Introduction

For critical system development, Original Equipment Manufacturers (OEMs) must perform Failure Mode and Effects Analysis (FMEA) to demonstrate the robustness (ability to successfully complete a mission despite the occurrence of faults) of each component. Standards recognized by certification authorities have been produced to guide the development of these critical systems. For example for airborne systems, ARP5580 [1] describes the basic procedure by which to perform an FMEA. Standard DO254 [2] specifically relates to the safety requirements for electronic embedded systems in aircraft hardware, including programmable devices (*e.g.*, FPGAs). These standards often recommend that robustness analysis be started early in the design flow (with functional descriptions) and that robustness requirements be verified throughout the whole top-down design flow. Due to the decreasing size of transistors, systems have become increasingly vulnerable to perturbations. Thus, designers must perform detailed FMEA considering Single Event Upsets (SEUs, corresponding to single bit-flips), Multiple Bit Upsets (MBUs), and Multiple Cell Upsets (MCUs). This FMEA, called SEU FMEA (even if it also relates to MBU and MCU), is often performed empirically without executable models and based solely on engineer experience. Thus, the number of scenarios analyzed is limited, and accurate fault propagation analysis in complex systems is difficult to perform.

FMEA can be used to analyze the failure effects of a single component on the whole system. It consists of 3 steps:

- definition of possible failures and their probability of occurrence,
- determination of the effects of failure on the corresponding functional block,
- determination of the effect on the whole system and computation of the probability of occurrence.

These steps are generally performed at different levels of abstraction. In this paper, we focus on the SEU FMEA of a circuit used in an embedded system for a critical application. The three steps involved must be applied at the transistor level or gate level, the RT-level, and the system level, respectively. A failure is the result of fault propagation within the system.

Since the late 1990s, an efficient means to evaluate the robustness of electronic circuits is by fault injection. Three main categories of fault injection techniques exist:

- Physical: The circuit is exposed to an external source of perturbation like particle flux or laser. Since the ’60s, numerous studies have attempted to reproduce SEU effects ([3], [4]). Physical fault injection can be used to inject realistic faults and to access some locations that are not accessible with other techniques. However, this approach does not allow early analysis because the target devices must be available. Circuit redesign at this stage is expensive and time-consuming, as are the experiments themselves.
- Emulation: The design is emulated on a hardware platform where faults can be injected. Thus, robustness can be evaluated earlier in the design flow, but the hardware platform must be available and efforts deployed to im-

plement the system to be evaluated on it.
- Simulation: The system and the physical defects are simulated. This approach allows circuit robustness to be evaluated at different levels of abstraction, but is much slower than emulation. In addition, low-level simulations can become intractable within an affordable time frame.

The main goal of the study presented in this paper was to propose an approach to perform an early evaluation of the robustness of hardware devices included in complex critical systems. This approach complies with the top-down flow and cross-layer analysis recommended in safety standards, and takes advantage of both low- and high-level fault simulations. The first steps in the evaluation can be performed before the whole system is described at the RT-Level. Then, the final step is completed when RTL descriptions are available to verify the quality of this early evaluation. This paper is an extension of the approach presented in [5]. Here, we take into account interactions with external components of the system (*e.g.*, microcontroller) and complete critical system specifications. In the first step, the method proposes to extract the critical ranges for circuit parameters using high-level fault models. These ranges give the limit values for the parameters leading to the observation of system-critical behavior. Early results and implementation of the high-level fault injection (HLFI) step are presented in [6]. These ranges also make it possible to establish assertions for RTL fault simulation. The second step in the methodology reuses these assertions to evaluate the robustness of each RTL block. The third step is used to validate the accuracy of the high-level fault models applied in the first step.

This paper is organized as follows. The current state of knowledge closely related to our work is further detailed in Section II. In Section III, we describe the three steps involved in our methodology. Section IV presents the case study. In Section V, we present and discuss the results of the proposed method in the current case study. Section VI concludes the paper.

II. State of the art

a) Simulated fault injection: Simulation tools can be used to evaluate circuit robustness when the simulations include some kind of signal perturbation. Different fault model levels exist, from functional to gate or transistor level. High-level modeling at the transaction level (Transaction Level Modeling or TLM) is compatible with simulations of complex systems. Lower levels (gate level, RT-level) can be used to check parameters such as timing, accurate error propagation, or even power consumption. They provide more accurate results, that are closer to reality, but require increased simulation efforts.

Fault simulation methods can be divided into two categories: without code modification (simulation commands modify values of signals or variables) or with code modification (mutants or saboteurs). The MEFISTO tool [7] was one of the first to apply fault injection with or without code modification at the RT-Level. Fault injection using simulation commands is the least intrusive and easiest way to inject faults. Nevertheless, fault injection possibilities are restricted and depend on the tool. Fault injection with code modification allows many possible mutants and saboteurs but some studies [8] have defined fault models corresponding to a subset of realistic faults. Some VHDL language extensions were later proposed to avail of the advantages of simulation commands, mutants, and saboteurs [9].

Safety analysis at the gate or even RT-level is expensive because it is applied late in the design flow. RTL fault simulations are often very time consuming because of the growing complexity of the RTL descriptions. Furthermore, as Champon et *al.* [10] showed, some errors observed using RTL fault simulations have no real effect on the overall critical system. Indeed, these errors are filtered out or detected by the system. Thus, interactions with other parts of the system and system specifications must be taken into account. Fault injection can also be performed at system level with TLM to accelerate fault simulation and to take the whole system into account. However, result accuracy will be reduced since fewer details of the final implementation are available, and more abstract fault models must be used.

b) Efficiency of High-Level Fault Simulation: High-level simulation with adequate fault modeling accelerates fault simulations and makes it possible to account for the whole system. Nevertheless, high-level modeling requires details (such as a clock or data types) to be removed, which causes some realism issues. Many groups have studied the problem of high-level fault realism:

- The methodology proposed by Miele [11] injects faults into a SoC modeled in SystemC at transaction level. The realism of the injected faults is verified by comparing the high-level fault injection with the RTL fault simulation. The case study shows that some of the RTL faults injected have no high-level equivalent and conversely, some high-level faults have no RTL equivalent.
- Tabacaru *et al.* [12] used prototypes to perform fault effect analysis on a circuit. The methodology consists of performing gate-level fault injection into the circuit without activating its safety mechanisms to obtain a realistic high-level fault library. These faults are then injected into the high-level model with the safety mechanisms activated to perform safety analysis. This approach does not take system specifications into account, and it requires access to the RTL code.
- Herdt [13] studied the correspondence between RTL and TLM faults through a formal approach. The formal approach can be used to extract the TLM equivalent set of faults for each RTL fault. The authors concluded that some RTL faults have no equivalent at transaction level because they propagate and create several TLM faults.

Based on these studies, the two main problems with high-level fault injection are the following:

- Some RTL faults propagate multiple high-level faults throughout the system.

- Many high-level faults are not realistic.

c) Cross-Layer Approaches: Cross-layer methods aim to take advantage of both RTL and TLM fault injection. Some studies proposed ways to follow these approaches:

- The CLERECO project [14] proposed a cross-layer reliability evaluation. The evaluation performs system reliability analysis by isolating the individual layers, thus simplifying the analysis. This approach focuses on microprocessor reliability analysis but is incompatible with study of complex systems composed of hardware blocks connected to the microprocessor.
- Perez *et al.* [15] performed fault simulations at different levels of abstraction in a critical embedded system to guide architecture and RTL hardware designers. This approach allows early decision making and avoids time-consuming design iterations. However, the realism of the high-level faults injected was not verified. Indeed, as computation models are different in a block modeled at high- or RT-level, it is often hard to find correspondences between high-level faults and RTL (or lower level) faults. Furthermore, although the approach takes into account interactions with other parts of the system, it does not consider overall system specifications.
- Mueller *et al.* [16] injected faults into a processor at RT and behavioral levels, simulating short scenarios at the RT-level and longer scenarios at the transaction level. This cross-layer study did not establish correspondences between different fault injections.

The cross-layer approach allows design to be explored and fault injection performed at different levels, but no study has yet proven the relevance of the high-level faults injected. High-level fault injection guides architecture choices, whereas low-level fault injection provides information for implementation choices.

d) Safety analysis tools: Some solutions to assist FMEA already exist. Bernard [17] describes how the AltaRica project can assist safety analysis for airborne systems. AltaRica is a language that provides system description capabilities thanks to states and transitions. A formal approach identifies critical scenarios. This tool requires failure modes to be defined for each state of the system. It is a very high-level approach, mainly useful for validating the architecture during early design phases. Mariani [18] proposed a tool that complies with certification standards. From the RTL description, it extracts sensitive zones and performs exhaustive fault injection in these zones. The tool extracts probabilities and data that are useful for the FMEA. Only the hardware part of the system is studied, and no system specifications are considered. Those tools are useful for safety analysis but do not allow early estimations of the robustness of full embedded systems.

e) Summary: Simulated fault injection makes it possible to evaluate circuit robustness at various levels. High-Level fault simulation allows early robustness analysis to be performed, accelerating fault simulation and taking the whole system into account. However, high-Level faults and high-level models have some realism issues. Cross-layer approaches aim to deal with low-level accuracy and avail of high-level advantages, but no study has yet proposed an approach allowing a statistical estimation of the probability of critical faults. In the aeronautics industry, some safety analysis tools exist at a higher level. Fault modeling still requires experienced designers to model accurate high-level faults. The methodology proposed aims to perform a robustness analysis in compliance with the safety recommendations.

III. Methodology

Our methodology [5] relies on a 3-step cross-layer evaluation, which is illustrated in Figure 1:

- Step 1 involves high-level fault injection on all the block interface signals of the design. This fault injection allows the critical parameter ranges of each block interface signal to be extracted. System specifications are taken into account to identify critical system behaviors.
- In Step 2, RTL fault injection is performed on the blocks leading to critical behaviors. This RTL fault injection aims to identify RTL faults inducing the critical parameter ranges identified for each block's interface signal in Step 1. It can then be used to compute critical SEU probabilities.
- Step 3 is a multi-abstraction simulation (both high-level and low-level simulation) which involves replacing (one at time) each critical block in the high-level description with its RTL description. This step allows the accuracy of Steps 1 and 2 to be verified by mixing two abstraction levels.

This methodology performs both the functional analysis and the fault simulation for the hardware architecture, as recommended by the two aeronautics standards ARP5580 and DO254, respectively. The HLFI method used to extract the parameter ranges leading to critical behaviors differs from classical HLFI methods in several ways:

- High-level faults are injected only in the component interfaces that exist in the RTL description. It provides a more realistic high-level analysis.
- By taking into account system specifications, it is possible to select erroneous behaviors leading to critical behaviors and to remove others.

A. Step 1: High-Level fault simulation

a) System modeling: Transaction-Level Modeling is a high-level approach to model systems. Communication interfaces and functional units are separated. At the transaction level, we focus on functionality and data transfer. TLM can be used to experiment with different system architectures. After exploring the architecture, each functionality can be designed using hardware or software components, and the information obtained on the communication interfaces remain unchanged through the whole design flow. Thus, TLM coupled with fault injection on communication interfaces provides an early evaluation of the robustness of critical functionalities in

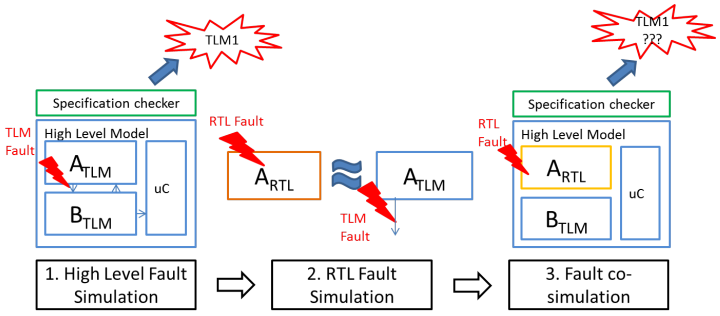

Fig. 1. Three steps in the methodology.

```
Main.cpp
Module declaration
Signal declaration
Binding
Start (simulation_time) ;
```

Fig. 2. Top module of the system

```
Main.cpp
Module declaration
Signal declaration
Binding
Start (i) ;
Inject_fault (l, t) ;
Start (simulation_time – i);
```

Fig. 3. Modified top module

embedded systems. In this study, as we were interested in analyzing only the robustness of the hardware devices, faults were only injected into the communication interfaces among the blocks belonging to these hardware devices.

To model the critical systems, it is essential to take computation times into consideration. Indeed, system specifications often include some time-related properties (*e.g.*, throughput or data flow rate), and a Time-To-Alarm (TTA) may be added to define the acceptable time before an error should be reported to the user. TLM Approximately Timed (TLM AT) is a level of abstraction that allows embedded systems to be modelled taking into account each transaction or internal function computation time. The SystemC library is based on C++ classes and macros that allow TLM AT modeling. It is *de facto* the standard for SoC virtual prototyping.

b) Fault injection: In our case, fault injection into the high-level model uses simulator commands [19]. This is the least intrusive solution since no code modification is required. In our HLFI method, a fault F is represented by a 4-tuple (L, T, I, D) where:

- L is the location and represents an interface signal (faults are injected into all the communication interfaces).
- T is the fault type. The fault type is a function that corrupts the original value of the signal. The signal can be modified by a proportion (signal +/- $X\%$) or a fixed value (signal +/- X), where X is the amplitude of the fault injected.
- I is the instant at which the fault is injected.
- D is the injection duration. As transient faults are injected, the value is corrupted either for a duration (corresponding to a perturbation burst) or until the signal is written once again (corresponding to a SEU or a MBU).

The SystemC top module of the original system description was modified to provide fault injection capabilities. A fault was then injected by inserting the 4-tuple(L, T, I, D) into the fault injection command. Figures 2 and 3 illustrate one code modification that provided fault injection capabilities. This implementation allows faults to be injected until the signal is written once again.

c) Evaluation of Critical System Parameter Ranges: Critical behaviors are erroneous behaviors (*i.e.* that do not respect the system specifications) which are not detected by the internal detection mechanisms (detected behaviors are not considered critical). An additional module (the specification checker) is used to filter the erroneous behaviors to extract those leading to critical behavior. This module uses assertions written using Property Specification Language (PSL). All system specifications are translated into PSL assertions. PSL is a formal language through which temporal properties for hardware designs can be specified. PSL properties are interpreted by a simulation tool or a formal verification tool. They can be used in particular with RTL description languages such as VHDL or Verilog, and with high-level description languages such as SystemC or SystemVerilog. System specifications allowed us to define PSL assertions from tolerance intervals on output signals (*e.g.*, min/max signal amplitude or delay). For example, TTA allows erroneous behaviors to be filitered to extract only those that must be notified in order to avoid false alarms. Our goal was to extract parameter ranges (for each interface signal L) leading to one of the following system behaviors: silent fault (S), quasi silent fault (Q), detected error (D), and critical error (C). These system behaviors are defined below.

Figure 4 shows how fault simulations were generated and results processed. The SystemC design is the TLM AT model of the system. It includes both the hardware components analyzed and all the remaining components of the embedded system (including the specification checker). The `Fault injection campaign` file corresponds to a set of faults to be injected into the TLM AT model. The `Scenarios` files contains scenarios to simulate. Fault injection into the TLM AT design is simulated in a SystemC environment. Algorithm 1 illustrates how HLFI results were sorted. The `Golden_output` file contains the fault-free simulation results (Line 2). The simulation results for each fault are written

in a temporary `output` file (Line 6). A script was used to implement this algorithm and to automatically process simulation results, sorting them into the following categories:

- Silent fault (S): HLFI output `output` is equal to `Golden_output` (Lines 7, 8).
- Quasi silent fault (Q): simulation output `output` is not equal to `Golden_output` but remains within the specification requirements (Lines 10, 11).
- Detected Error (D): simulation output `output` is outside the specification requirements and at least one of the detection mechanisms triggered an error (Lines 13, 14).
- Critical error (C): simulation output `output` is outside the specification requirements and is not detected by the [implemented] mechanisms implemented (Line 15).

Once the results of HLFI have been sorted into these categories, faulty ranges can be extracted (respectively detected or critical error ranges determined) according to ranges of X or I that lead to a critical error.

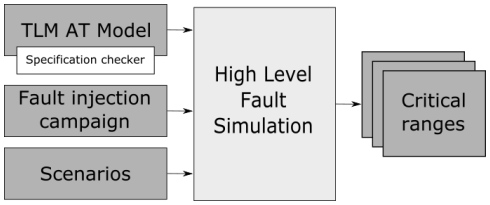

Fig. 4. Overview of HLFI analysis.

Algorithm 1: High-Level Fault Injection Analysis

```
1  Golden_output ← simulate(Golden_model, input, No_Fault)
2  for F ∈ Fault_injection_location do
3      Faulty_output ← simulate(Faulty_model, input, F)
4      if Faulty_output = Golden_output then
5          S ← F
6      else
7          if Faulty_output ∈ specification_requirements then
8              Q ← F
9          else
10             if Faulty_output ∈ Detected_Error then
11                 D ← F
12             else
13                 C ← F
```

d) Illustration: Figure 5 illustrates Step 1. Let S be a system, A and B two blocks of S. We denote A_{TLM} and B_{TLM} the TLM description of A and B. System S is simulated with fault injection at the transaction level. Let $F = (L, T, I, D)$ be a fault injected on signal sig. The value of sig is corrupted to X at time I. If the output signals out do not satisfy the specifications and remain undetected, F is considered to lead to critical behavior. Table VII presents an example of results obtained from the high-level fault simulation. Results are sorted in a Table to identify ranges. For example, the critical parameter range is:

$$[+1\%; +5\%] \times [50ms; 250ms]$$

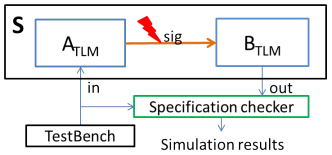

Fig. 5. Step 1: High-level fault injection.

TABLE I
EXAMPLE OF HIGH-LEVEL FAULT INJECTION RESULTS

T \ I	0ms	50ms	100ms	150ms	200ms	250ms	300ms
Injection on sig							
-50%	D	D	D	D	D	D	D
...%	...	...	...	...	...	...	...
-20%	D	D	D	D	D	D	D
-10%	Q	Q	Q	Q	Q	Q	Q
...%	...	...	...	...	...	...	...
-1%	Q	Q	Q	Q	Q	Q	Q
+1%	Q	Q	C	C	C	Q	Q
...%	...	...	...	...	...	...	...
+5%	Q	Q	C	C	C	Q	Q
+10%	D	D	D	D	D	D	D
...%	...	...	...	...	...	...	...
+50%	D	D	D	D	D	D	D

B. Step 2: RTL fault simulation

The blocks leading to critical system behaviors are identified, and, when their RTL description is available, are simulated with RTL fault injections. Simulation at this level allows more accurate fault simulation, and make it possible to identify the RTL faults (*e.g.,* Flip-Flop single bit flips, or register multiple bit flips) propagating and creating signal interfaces in the critical ranges identified in Step 1. We defined assertions on interface signals corresponding to those parameter ranges.

When an RTL fault violates the assertion, the RTL fault is considered critical for the system. The ratio of RTL faults satisfying or violating assertions can be used to extract statistics.

a) Fault injection: The fault model at the RT-level is very similar to the fault model at the system level: a fault f is defined by a 4-tuple (l, t, i, d) where:

- l is the location: faults are injected into all the Flip-Flops and registers.
- t is the type: It defines the type of faults injected. Single bit flips were injected in the experiments described here (multiple bit flips in each block may also be considered in future studies).

- i is the instant of injection: Faults are injected at different simulation times.
- d is the fault duration: the value is corrupted until the register or the Flip-Flop is written once again.

Fault injection at the RT-level is performed using simulator commands.

b) Fault equivalence checking: To determine whether a RTL fault propagates and creates a high-level fault, each range is first translated into a PSL property.

Let $[x1; x2] \times [t1; t2]$ be a critical range. Let $window$ be a signal that defines the validity time window of the property; $window$ is given by the characteristic function of the set $[t1; t2]$ (1 if current time is in the range, 0 otherwise). We define a PSL property that checks whether the faulty behavior of the RTL block in presence of RTL faults is equivalent to the critical range $[x1; x2] \times [t1; t2]$ identified in step 1:

$$\textbf{Assert always } window \quad \textbf{->} \quad Q \bigtriangledown \ s$$

where s is the value of signal l without corruption, s' its corrupted value, and $\bigtriangledown$ can be any relational operator (*e.g.*, $=$, $\geq$ and $\leq$, ...).

Block verification in the RTL model is performed based on these formal properties (equivalent to the high-level ranges leading to critical behaviors). If a property holds despite fault injection and an incorrect output, it means that the fault is not relevant and that the block robust to this fault.

c) Illustration: Step 2 (see Fig. 6) is illustrated based on the example given in Step 1. The temporal property φ corresponds to the critical range identified in Step 1:

$$\textbf{Assert always } window \quad \textbf{->}$$
$$((sig' \geq \ sig * 1.05) | (sig' \leq \ sig * 1.01))$$

where $window$ is equal to one between 50 ms and 250 ms, and 0 otherwise.

Faults are now injected into the RTL description of A (A_{RTL}), and we verify the correctness status of φ. The fault simulation is only performed on A_{RTL}: faults are randomly injected in single bits belonging to the same registers of A_{RTL}. If φ never holds despite the fault injections, it means that high-level faults associated with the critical range are not relevant and the RTL block is robust. If φ holds for some fault injections, then some RTL bits or registers of the block are identified as the most sensitive bits or registers of the block. Furthermore, based on the ratio of RTL faults for which φ holds, statistics can be extracted.

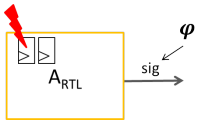

Fig. 6. Step 2: RTL fault injection.

C. Step 3: System fault co-simulation

In Step 3, each RTL block is successively co-simulated in the high-level system description developed in Step 1 with RTL fault injection. Hereafter, we call this a system fault co-simulation. Step 3 aims to validate the critical ranges extracted in Step 1. The same RTL faults used in Step 2 are used once again (when RTL fault injection was done in Step 2; i.e., when critical parameter ranges for at least one output parameter of the block were identified in Step 1). When no RTL fault injection was done in Step 2, then a first RTL fault injection campaign is performed on a block during Step 3. If the system specification holds despite RTL fault injection, it means that the fault is silent or quasi-silent and the overall system is robust to it. These multi-abstraction simulations allow the high-level fault models used in step 1 and the results obtained in step 2 to be evaluated based on the ranges extracted with these models. Using multi-abstraction decreases the duration of the whole system simulation as it uses the most accurate faulty models (*e.g.*, RTL fault model) for the critical blocks. However, simulating even a single RTL block in a complex system remains much more time consuming than performing a high-level system simulation. Thus, system fault co-simulation can only be used to validate the previous fault simulation results based on less time-consuming fault simulations.

Table II is a confusion matrix [20]. It allows us to define metrics to evaluate the results obtained in the previous steps. The "Predicted" column in this matrix corresponds to the results from Steps 1 and 2. The "Co-simulation" line corresponds to the results of Step 3 and is considered the reference behavior. The accuracy is the proportion of true results compared to the number of injected faults. It is calculated as follows:

$$Accuracy = \frac{TC + TO}{TC + TO + FC + FO}$$

The precision is defined by the proportion of true critical behaviors identified in the two first steps:

$$Precision = \frac{TC}{TC + FC}$$

TABLE II
CONFUSION MATRIX

Predicted \ Co-simulation	Critical	Others
Critical	True Critical (TC)	False Critical (FC)
Others	False Others (FO)	True Others (TO)

Illustration: As illustrated in Figure 7 system S is now simulated with A_{RTL} and B_{TLM}. Faults extracted from Step 2 are injected in A_{RTL} and we verify the correctness of the output signals of the system using the verification checker.

IV. CASE STUDY

A. System description

Hereafter, for confidentiality reasons, all names have been concealed and all values normalized. We studied an embedded

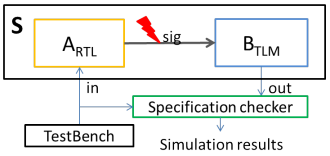

Fig. 7. Step 3: System fault co-simulation.

system SYS_X that measures a physical value X, which is sent to the main airplane ECU (Electronic Control Unit). Based on other information, the ECU deduces the flight information from this value.

The embedded system SYS_X is based on two sensors $\mathrm{S_X}$ and $\mathrm{S_T}$. Sensor $\mathrm{S_T}$ measures the temperature T and vibrates at a frequency $F(\mathrm{T})$. Sensor $\mathrm{S_X}$ measures the value X; it vibrates at a frequency $F(\mathrm{X})$ determined by both the value X and the temperature T. The system is composed of an FPGA, a microcontroller and a memory containing the sensor table. The FPGA part contains a frequency meter to measure the frequencies emitted by sensors and a communication block that transmits data to the ECU.

Figure 8 illustrates the detailed architecture of the system. The system operates with two counters, CounterX and CounterT, which count the number of cycles performed by the sensors $\mathrm{S_X}$ and $\mathrm{S_T}$, with respect to a reference frequency Fref. Two lower frequencies $F(D)$ and $F(\mathrm{SYNC})$, for $\mathrm{S_X}$ and $\mathrm{S_T}$ respectively, give the measurement windows of CounterX and CounterT. The microcontroller periodically reads the number of cycles counted during the measurement window (an interrupt occurs when new values are written in the output registers for CounterX and CounterT). Based on these values, the microcontroller computes the corresponding periods T_T and T_X. The value of X is then computed from data in a sensor table (series of points containing the values X corresponding to a given pair of periods T_T and T_X) characterizing each sensor $\mathrm{S_X}$ (see Fig. 8). The microcontroller uses an interpolation method to calculate an accurate value of X. The microcontroller then transmits the calculated value and some maintenance information (error detection mechanism outputs) to the FPGA block in charge of communication. The communication FPGA block then formats the data for transmission to the ECU.

The system constraints are described in the airplane specifications (see Table III). "Data Flow Rate" is the time between two value transmissions. The "Time to Alarm" corresponds to the minimum time before an error is notified during data measurement. The "Transmission Period" is the time required for the communication block to transmit the data. The "Computation Period" is the time spent by the microcontroller computing the final value and transmitting it to the communication block when an interrupt occurs. "Transport Delay" is the time between the beginning of a measurement window and transmission of the data by the communication block to the ECU. Several error detection mechanisms are also implemented in the system: counter overflow, detectors of periods out of the sensor table, detection of X values outside the relevant range, *etc.* Some of these mechanisms are described in Table IV. The first column corresponds to the block in which the error detection mechanism is implemented.

TABLE III
SYSTEM SPECIFICATION

Property	Specification
Data Flow Rate	100 ms +/- 3.3 ms
Time to Alarm	300 ms +/-3.3 ms
Accuracy	+/- 0.8 U (confidential unit)
Transmission Period	$\leq$ 100 ms
Computation Period	$\leq$ 100 ms
Transport Delay	300 ms +/- 3.3 ms

TABLE IV
ERROR DETECTION MECHANISMS

	Error Detection Mechanism	Execution Every
FPGA	output is looped back	100 ms
μC	Hardware reset at the start of each calculation cycle	100 ms
μC	check if X (or T) period is out of range	100 ms

B. FMEAs on the case study

Two safety analyses were performed on the whole system.

The first analysis at the functional level is based on one FMEA (in compliance with the ARP5580 standard). It allows early identification of the critical system paths and evaluation of block criticality. It is not exhaustive (takes only a few faults into account), and the critical faults found are often not realistic.

The second, more detailed, analysis determines the probability of critical events. It contains two FMEAs:

- One which studies the effect of component failures on the system. These component failures are permanent. As they are rare, an overall analysis of their effect on the system is sufficient.
- One which examines the effect of Single Event Upsets (SEUs) and Multiple Bit Upsets (MBUs). It needs to be more accurate because of the high probability of particle impacts on the system. Figure 9 illustrates the principles of this analysis. The particle flow and the area of each block allow the particle flow on each block of the system to be deduced. The critical fault ratio (*i.e.*, SEU/MBU rate leading to a critical fault according to each block of the system) can be used to calculate the probability that a critical event will occur. This ratio is empirically determined based on engineer experience. To be conservative, ratios are pessimistic: in case of doubt SEU/MBU are considered critical for the system.

As for this case study, safety analyses are often not based on executable models. In addition, the certification standards highlight the need for functional FMEA. High-level system models could be used to perform some more accurate functional FMEA.

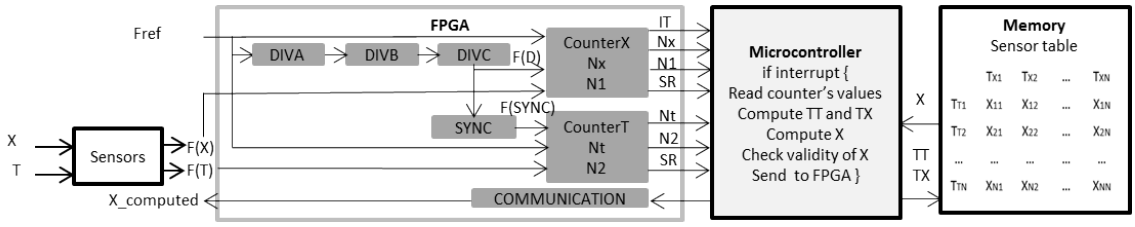

Fig. 8. Detailed system architecture.

Fig. 9. Principles of the Upset effects analysis.

V. Simulation Results

Here, we will now apply the first two steps of the methodology to the case study. Our methodology based on fault simulation techniques aims to obtain more realistic results than those provided by empirical functional FMEA and empirical SEU/MBU FMEA.

A. Step 1

A HLFI campaign was performed on the TLM AT test case model: 1,176 high-level faults were injected. Each fault corresponds to a 4-tuple (L, T, I, D):

- 8 interface signals (L): targeting all the TLM block interface signals in the hardware part of the system (*e.g*, diva_to_divb, divb_to_divc, freq_out). The goal was to analyze the impact of the faults on the FPGA behavior.
- 21 faulty values (T): Each signal was modified to cover a deviation range from -50% to +50% in 10% increments. The step size may be reduced to refine the analysis in future experiments.
- 7 injection times (I): 10 ms jection intervals were chosen. Injection times were 0 ms, 10 ms, 20 ms, ..., 60 ms. No faults were injected after 60 ms because the overall computation process takes 60 ms. Nevertheless, to refine the analysis, the interval between injections may be reduced.
- Injection duration (D): faults on each signal were injected until the signal is written once again.

The high-level simulation time for injecting and simulating all faults lasted only 3 minutes. Table V presents the results of the fault simulations for some of the signals of the high-level test case model. Overall simulation results were as follows:

- 509 quasi-silent faults (Q: different from the reference file but still within the specifications),
- 343 detected errors (D),
- 227 critical errors (C).

As the probability of occurrence of each high-level fault is unknown, it is impossible to deduce the probability of critical events from these results. This deduction will be possible using the RTL descriptions. Nevertheless, we can observe the considerable interest of taking system specifications into account to identify the fault simulations that should be analyzed. In subsequent steps, this allows the tolerable faults T to be considered no different from the reference file as they do not produce erroneous system behaviors.

TABLE V
Fault injection in divider and counter A

T \ I	0ms	10ms	20ms	30ms	40ms	50ms	60ms
Injection on diva_to_divb							
-50%	D	D	D	D	D	D	D
...%	...	...	...	...	...	...	...
-20%	D	D	D	D	D	D	D
-10%	Q	Q	Q	Q	Q	Q	Q
...%	...	...	...	...	...	...	...
-1%	Q	Q	Q	Q	Q	Q	Q
+1%	S	S	S	S	S	S	S
...%	...	...	...	...	...	...	...
+5%	S	S	S	S	S	S	S
+10%	D	D	D	D	D	D	D
...%	...	...	...	...	...	...	...
+50%	D	D	D	D	D	D	D
Injection on divb_to_divc							
-50%	D	D	D	D	D	D	D
...%	...	...	...	...	...	...	...
+50%	D	D	D	D	D	D	D
Injection on freq_out							
-50%	D	D	D	D	D	D	D
...%	...	...	...	...	...	...	...
-1%	D	D	D	D	D	D	D
+1%	Q	Q	Q	Q	Q	Q	Q
...%	...	...	...	...	...	...	...
+5%	Q	Q	Q	Q	Q	Q	Q
+10%	C	C	C	C	C	C	C
...%	...	...	...	...	...	...	...
+50%	C	C	C	C	C	C	C
Injection on sync_out							
-50%	S	S	S	S	S	S	S
...%	...	...	...	...	...	...	...
+10%	S	S	S	S	S	S	S
+20%	D	D	D	D	S	S	S
...%	...	...	...	...	...	...	...
+50%	D	D	D	D	S	S	S

a) Divider fault analysis: The upper part of Table V illustrates the results of applying HLFI to the diva_to_divb interface signal. For each injected fault, the results of the simulation are reported in the table to identify some ranges. Results are sorted in three categories as defined in Section III. The

TABLE VI
FAULT INJECTION IN NX, NT, N1 AND N2

T \ I	0ms	10ms	20ms	30ms	40ms	50ms	60ms
Injection on Nx							
-50%	S	S	S	D	S	S	D
...%	...	...	...	...	...	...	...
-3%	S	S	S	D	S	S	D
-2%	S	S	S	C	S	S	C
...%	...	...	...	...	...	...	...
+10%	S	S	S	C	S	S	C
+20%	S	S	S	D	S	S	D
...%	...	...	...	...	...	...	...
+50%	S	S	S	D	S	S	D
Injection on Nt							
-50%	D	D	D	D	D	D	D
-40%	C	C	C	C	C	C	C
...%	...	...	...	...	...	...	...
+5%	C	C	C	C	C	C	C
+10%	D	D	D	D	D	D	D
...%	...	...	...	...	...	...	...
+50%	D	D	D	D	D	D	D
Injection on N1							
-50%	S	S	S	D	S	S	D
...%	...	...	...	...	...	...	...
-3%	S	S	S	D	S	S	D
-2%	S	S	S	C	S	S	D
-1%	S	S	S	C	S	S	C
+1%	S	S	S	C	S	S	C
+2%	S	S	S	C	S	S	D
+3%	S	S	S	D	S	S	D
...%	...	...	...	...	...	...	...
+50%	S	S	S	D	S	S	D
Injection on N2							
-50%	D	D	D	D	D	D	D
...%	...	...	...	...	...	...	...
-10%	D	D	D	D	D	D	D
-5%	C	C	C	C	C	C	C
...%	...	...	...	...	...	...	...
+5%	C	C	C	C	C	C	C
+10%	D	D	D	D	C	C	C
+20%	D	D	D	D	D	D	D
...%	...	...	...	...	...	...	...
+50%	D	D	D	D	D	D	D

table shows that fault injections create detected errors (D) for faulty values between $[-50\%; -20\%]$ and [in] $[+10\%; +50\%]$. These ranges are independent of the fault injection instant. Faulty values between $[-10\%; +5\%]$ produce silent faults (S). For faulty values between $[-50\%; -20\%]$, the mechanism detecting the fault is $M1$ (the microcontroller missed a value on the Nx and $N1$ registers). Furthermore, the microcontroller also detects a period computed outside the sensor table ($M3$). Finally, two mechanisms always detect these errors. For the faulty values between $[+10\%; +50\%]$, the watchdog of the microcontroller ($M4$) indicates that the timing is not respected.

The parameters diva_to_divb, divb_to_divc, and sync_out only produce faults leading to detected errors. Consequently, there is no critical range for these parameters. However, the parameter freq_out corresponding to the divider DIVC has a critical range. This first analysis shows which dividers are more sensitive to critical faults.

b) Counter fault analysis: The upper part of Table VI presents the results of HLFI on the Nx signal for CounterX. Fault injections create critical errors for faults injected at 30 ms and 60 ms. At these injection instants, faults were critical for all the values in the $[-2\%; +10\%]$ range. At the same instants, errors outside this range were detected by the detection mechanisms implemented in the microcontroller because they create some periods outside the sensor table. Fault injection on Nx and $N1$ produced similar results. Thus, injecting a fault only had an impact at 30 ms and 60 ms because it corresponds to the time slot between writing a new value on the register and it being read by the microcontroller. Nt and $N2$ are more sensitive because they correspond to continuous counters

c) Discussion of results: These results show the following advantages of our approach:

- The HLFI method achieves fast fault simulations early in the design flow.
- Specification checking can be used to filter out the high-level behaviors leading to critical behaviors. It allows 79% of erroneous behaviors to be eliminated.
- The HLFI method allows easy extraction of some critical ranges. These ranges are defined either as simulation time intervals or as absolute or relative parameter value intervals. Critical ranges allow RTL faults to be filtered. Indeed, only RTL faults involving parameter values in these ranges will be considered critical faults.
- Redundant error detection mechanisms can be identified. For example, in the divider, two error detection mechanisms detected the error created by the fault injection between -50% and -20%. Further investigations would allow us to validate the most efficient mechanism.

B. Step 2

Step 2 of the methodology looks for RTL faults producing the parameter ranges identified in Step 1. These ranges were first translated into PSL properties as explained in III.B.b. Then, an RTL fault simulation campaign was performed. Table VII presents the results of the RTL fault simulations for DIVC, CounterX, and CounterT RTL blocks. Exhaustive or statistical fault simulations were performed depending on the complexity of each block. Fault simulation results are sorted into 3 categories: critical faults (*i.e.*, assertion violations), tolerable faults (*i.e.*, outputs differing from the golden model outputs but that are not critical), and silent faults. The simulation times for each block and the number of flip-flops occurring in each block is also reported with the proportion relative to the whole system.

a) DIVC RTL analysis: An exhaustive RTL fault simulation [has been] was performed on the RTL description of DIVC. Single bit-flips were injected into all the FF elements in this RTL block, at each clock cycle. Only 9.8% of these faults lead to the freq_out critical parameter range identified for in Step 1.

b) CounterX and CounterT RTL analysis: CounterX outputs are Nx and $N1$. For each output, a PSL assertion was

defined from the critical ranges obtained in Step 1. As the CounterX counting cycle lasts 30 ms, each fault simulation must last longer. The simulation of only one counting cycle is too long to perform an exhaustive fault simulation campaign in an affordable time frame. We therefore choose to perform a statistical fault simulation campaign. This campaign lasts 2.5 hours for 10,098 simulated faults among which 6,362 (63%) violate one assertion. The same statistical fault simulation was applied to CounterT. The CounterT counting cycle lasts 1.5 s. Consequently, the simulation lasted 10 hours, with 2,018 faults injected, of which 54.4% violated one assertion.

c) Computation of the circuit critical fault probability: Previous results allowed the SEU Failure Rate (FR_SEU) to be estimated. This metric corresponds to the circuit critical fault probability. Since the DIVA, DIVB and SYNC blocks have no critical range, they do not affect the FR_SEU computation. Algorithm 2 illustrates computation of FR_SEU. We note α the SEU rate (*i.e.*, the number of SEUs per bit per time unit). This rate is a parameter of the FMEA. This algorithm first computes the number of bit-flips leading to critical behaviors for the entire circuit (line 3). Then, the FR_SEU is obtained by multiplying the proportion of critical bit-flips for the circuit by α.

Algorithm 2: SEU Failure Rate computation

```
1 Nb_critical_faults = 0
2 for Blocks ∈ System do
3     Nb_critical_faults = Nb_critical_faults +
        (Probability_critical_fault(Block) *
        Nb_bits(Block))
4 FR_SEU = Nb_critical_faults / (Total number of bits) * α
```

The FR_SEU failure rate computed for this case study using Algorithm 2 is more accurate than the empirical FMEA originally estimated by the designers. We obtained a 3-fold lower FR_SEU than the FR_SEU from the empirical FMEA.

C. Step 3

For DIVA, DIVB, and SYNC, which had no critical ranges identified in Step 1, results were consistent with the analysis performed in Step 1: no RTL fault led to critical behaviors within the system fault co-simulation. For DIVC, the system fault co-simulation takes 5 hours; and 37 hours for CounterX. These times are around 10 times longer than individual RTL fault simulation times.

Since the faults simulated in Step 3 are the same as those in Step 2, we can compute a confusion matrix for each block. For example, Table VIII shows the confusion matrix for DIVC. This table highlights the lack of false silent or quasi-silent faults (FO), and reveals numerous false critical faults (FC). However, compared to the empirical FMEA which assumed all faults to be critical, our approach can rapidly estimate a small subset of critical faults from among the entire list of faults (3,794 critical faults over 39,053 faults). In addition, this subset contains all the real critical faults. From this confusion matrix, we obtained an accuracy of 0.92 and a precision of 0.25 (as defined in III.C). These values show that Steps 1 and 2, which are much faster than Step 3 give accurate but not precise results. The confusion matrix allows us to evaluate the quality of our high-level fault modeling approach, but this evaluation is very time-consuming. Indeed, to compute the accuracy and the precision with a confusion matrix we had to inject the same faults as those injected in Step 2. Of course, this complete evaluation is not required for each new FMEA, but only when an FMEA involves new high-level blocks. For casual systems, a system fault co-simulation using a subset of the faults would allow faster verification of the critical parameter ranges.

In Step 1, only high-level single fault injections are performed on high-level parameters. However, SEU also affect blocks with multiple outputs (such as CounterX and CounterT) which are related to multiple parameters (such as Nx, N1, SR for CounterX). RTL faults can propagate through these blocks and simultaneously create multiple errors in their outputs. For blocks produced with multiple outputs, it is particularly important to evaluate the accuracy. In Step 3, we verify whether these multiple errors (quasi-silent faults) produce additional critical faults. In our test case, the percentage of multiple errors corresponding to quasi-silent faults (generated by RTL fault injections) on CounterX and CounterT outputs is less than 8%. It follows that we will observe less than 8% additional critical faults. For other applications, if too many additional critical faults occur, the high-level fault model must be improved to improve the evaluation performed in Step 1.

TABLE VII
RTL FAULT SIMULATION RESULTS

	DIVC	**CounterX**	**CounterT**
Number of Flip-Flops	51 (16.2%)	102 (32.4%)	102 (32.4%)
Number of injected faults (exhaustive/statistic)	39,052 (exhaustive)	10,098 (statistical)	2,018 (statistical)
Number of critical faults	3,794 (9.8%)	6,362 (63%)	1,098 (54.4%)
Number of quasi-silent faults	8,346 (21.3%)	979 (9.7%)	293 (14.5%)
Number of silent faults	26,912 (68.9%)	2,757 (27.3%)	627 (31.1%)
Simulation time	45 min	2.5 h	10 h

TABLE VIII
CONFUSION MATRIX FOR DIVC

Predicted \ Co-simulation	Critical	Others
Critical	TC: 960	FC: 2,900
Others	FO: 0	TO: 33,171

VI. CONCLUSION

This paper presents a cross-layer fault simulation approach which speeds up SEU FMEA on circuits used in critical applications. This approach involves three steps:

- Step 1 extracts critical parameter ranges by injecting high-level faults and filtering results with respect to the overall system specifications.
- Step 2 reuses the critical ranges identified in Step 1 to filter the results of an RTL fault injection simulation. This step provides statistical information.
- Step 3 highlights some modeling defects.

The results show the following advantages of the methodology:

- Faster fault simulation
- Filtering of erroneous behaviors taking system specificationsinto account
- Statistics on critical fault occurrence
- Verification of the realism of the high-level model.

The approach was applied to a programmable circuit used in a critical airborne system. It proposes a safety analysis considering real-time aspects.

Our evaluation is closer to real effects than the empirical FMEA originally performed by the designers. The simulation allowed us to identify and suppress from the critical fault list the errors for which consequences are naturally pruned by the complete system behavior. All faults identified as having no criticality at the system level implicate the implementation of less costly protections in the system, and reduced development time.

Additional studies on high-level model realism issues will need to be performed on real-time systems. Future works aim to develop more realistic high-level models and high-level faults. To validate the realism of these models, a whole-system quantitative analysis at Step 3 will be performed. In addition, the methodology will allow us to identify the robustness mechanisms that are truly useful, and those that may be redundant.

ACKNOWLEDGMENT

This work is part of the Safe-Air project, from the "Pack ambition recherche" program, funded by "La Région Auvergne-Rhône-Alpes" .

REFERENCES

[1] *Design assurance guidance for airborne electronic hardware*, 2000.

[2] SAE International, *Recommended failure modes and effects analysis (fmea) practices for non-automobile applications arp5580*, 2012.

[3] D. H. Habing, "The use of lasers to simulate radiation-induced transients in semiconductor devices and circuits," *IEEE Transactions on Nuclear Science*, vol. 12, no. 5, pp. 91–100, 1965.

[4] V. Pouget, D. Lewis, H. Lapuyade, P. F. R. Briand, L. Sarger, and M.-C. Calvet, "Validation of radiation hardened designs by pulsed laser testing and spice analysis," *Microelectron. Reliab.*, vol. 39, pp. 931–935, 1999.

[5] J. Roux, V. Beroulle, K. Morin-Allory, R. Leveugle, L. Bossuet, F. Cézilly, F. Berthoz, G. Genévrier, and F. Cerisier, "Cross layer fault simulations for analyzing the robustness of rtl designs in airborne systems," in *2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2020, pp. 1–4.

[6] ——, "High level fault injection method for evaluating critical system parameter ranges," in *2020 27th IEEE International Conference on Electronics Circuits and Systems (ICECS)*, 2020, pp. 1–4.

[7] E. Jenn, J. Arlat, M. Rimén, J. Ohlsson, and J. Karlsson, "Fault injection into vhdl models: The mefisto tool," in *Predictably Dependable Computing Systems*, B. Randell, J.-C. Laprie, H. Kopetz, and B. Littlewood, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 329–346, ISBN: 978-3-642-79789-7.

[8] Ghosh, S. Chakraborty, and T.J, "On behavior fault modeling for digital designs.," *J Electron Test 2*, pp. 135–151, 1991.

[9] H. Cho, S. Mirkhani, C.-Y. Cher, J. A. Abraham, and S. Mitra, "Quantitative evaluation of soft error injection techniques for robust system design," in *Proceedings of the 50th Annual Design Automation Conference*, ser. DAC '13, Austin, Texas: Association for Computing Machinery, 2013, ISBN: 9781450320719. DOI: 10.1145/2463209.2488859. [Online]. Available: https://doi.org/10.1145/2463209.2488859.

[10] R. Champon, V. Beroulle, A. Papadimitriou, D. Hély, G. Genévrier, and F. Cézilly, "Comparison of rtl fault models for the robustness evaluation of aerospace fpga devices," in *2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2016, pp. 23–24.

[11] A. Miele, "A methodology for the design and the analysis of reliable embedded systems," Ph.D. dissertation, Politecnico di Milano, 2010.

[12] B. Tabacaru, M. Chaari, W. Ecker, T. Kruse, and C. Novello, "Fault-effect analysis on system-level hardware modeling using virtual prototypes," in *2016 Forum on Specification and Design Languages (FDL)*, 2016, pp. 1–7.

[13] V. Herdt, H. M. Le, D. Grobe, and R. Drechsler, "On the application of formal fault localization to automated rtl-to-tlm fault correspondence analysis for fast and accurate vp-based error effect simulation - a case study," in *2016 Forum on Specification and Design Languages (FDL)*, 2016, pp. 1–8.

[14] A. Vallero, S. Tselonis, N. Foutris, M. Kaliorakis, M. Kooli, A. Savino, G. Politano, A. Bosio, G. Di Natale, D. Gizopoulos, and S. Di Carlo, "Cross-layer reliability evaluation, moving from the hardware architecture to the system level: A clereco eu project overview," *Microprocessors and Microsystems*, vol. 39, no. 8, pp. 1204–1214, 2015, ISSN: 0141-9331. DOI: https://doi.org/10.1016/j.micpro.2015.06.003. [Online].

Available: http://www.sciencedirect.com/science/article/pii/S0141933115000824.

[15] J. Perez, M. Azkarate-askasua, and A. Perez, "Codesign and simulated fault injection of safety-critical embedded systems using systemc," in *2010 European Dependable Computing Conference*, 2010, pp. 221–229.

[16] D. Mueller-Gritschneder, P. R. Maier, M. Greim, and U. Schlichtmann, "System c-based multi-level error injection for the evaluation of fault-tolerant systems," in *2014 International Symposium on Integrated Circuits (ISIC)*, 2014, pp. 460–463.

[17] R. Bernard, J.-J. Aubert, P. Bieber, C. Merlini, and S. Metge, "Experiments in model based safety analysis: Flight controls," *IFAC Proceedings Volumes*, vol. 40, no. 6, pp. 43–48, 2007, 1st IFAC Workshop on Dependable Control of Discrete Systems, ISSN: 1474-6670. DOI: https://doi.org/10.3182/20070613-3-FR-4909.00010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667015310934.

[18] R. Mariani, G. Boschi, and F. Colucci, "Using an innovative soc-level fmea methodology to design in compliance with iec61508," in *2007 Design, Automation Test in Europe Conference Exhibition*, 2007, pp. 1–6.

[19] S. Misera, H. T. Vierhaus, and A. Sieber, "Fault injection techniques and their accelerated simulation in systemc," in *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007)*, 2007, pp. 587–595.

[20] T. Fawcett, "An introduction to roc analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006, ROC Analysis in Pattern Recognition, ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2005.10.010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016786550500303X.