FMEA on Critical Systems: A Cross-Layer Approach Based on High-Level Models

Julie Roux[1,2], Katell Morin-Allory[2], Vincent Beroulle[1], Lilian Bossuet[3], Frederic Cezilly[4], Frederic Berthoz[4], Gilles Genevrier[4], Francois Cerisier[5], and Regis Leveugle[2](✉)

[1] Univ. Grenoble Alpes, CNRS, Grenoble INP Institute of Engineering Univ. Grenoble Alpes, LCIS, 26000 Valence, France
[2] Univ. Grenoble Alpes, CNRS, Grenoble INP Institute of Engineering Univ. Grenoble Alpes, TIMA, 38000 Grenoble, France
regis.leveugle@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

Abstract. Designing embedded systems for critical applications requires meeting strict safety constraints according to official standards. In current practice, safety analysis (e.g., Failure Mode and Effects Analysis) is often only relying on human experience and therefore lacks detailed data. Performing more detailed analyses on complex systems is a major challenge to avoid pessimistic assumptions and consequently to avoid over-design of the system, i.e., adding too many protections with respect to the system specifications and risk. Many fault injection techniques have been previously proposed to better evaluate the robustness of circuit designs described at various abstraction levels. However, very few take into account the global system constraints. Also, fault injection experiments become very time-consuming for complex designs. At the highest levels of abstraction (e.g., Transaction level), simulations are faster but suffer of the lack of realism of high-level models. Our contribution is to propose both an increase in safety analysis precision and a fault injection flow improving the analysis duration. The flow is based on an iterative process, taking into account the global system specifications and allowing improvements of high-level models to achieve both precision and efficiency. Improvements are based on metrics, and results are shown on a real airborne system.

Keywords: Safety · Embedded system · FMEA · Fault simulation · Cross-layer

1 Introduction

When developing embedded systems for critical applications, Original Equipment Manufacturers (OEMs) must perform Failure Mode and Effects Analysis (FMEA) to demonstrate the robustness of each component. Standards guide the development of these critical systems. Standards such as ARP5580 [1] for airborne systems and DO254 [2] for

V. Grimblatt et al. (Eds.): VLSI-SoC 2021, IFIP AICT 661, pp. 113–133, 2022.
https://doi.org/10.1007/978-3-031-16818-5_6

electronic embedded systems in aircraft recommend that robustness analysis starts early in the design flow.

FMEA aims et analyzing the failure effects of a component on the system and consists in 3 steps:

- Definition of possible failures and their probability of occurrence,
- Determination of the failure effects on the functional block,
- Determination of the global effect on the whole system and computation of the probability of occurrence of unacceptable events.

Designers must perform detailed FMEA considering:

- Single Event Upsets (SEUs), i.e., single bit-flips,
- Multiple Bit Upsets (MBUs), i.e., multiple bit-flips in the same logic word,
- Multiple Cell Upsets (MCUs), i.e., multiple bit-flips in the several logic words.

This FMEA, called SEU FMEA (even if it also relates to MBUs and MCUs) is often based solely on engineer experience. Some solutions to assist FMEA have been proposed [3, 4] but these tools do not allow early estimations of the robustness of complex systems.

Evaluating the robustness of a circuit can be based on different fault models, at different levels from functional to gate or transistor level. Low levels (transistor or gate level), or even Register Transfer level (RTL) can be used, but are very time-consuming to perform the robustness evaluation of complex systems. Furthermore, some errors observed using RTL fault simulations have no real effect on the overall system because these errors are filtered out or detected by the system. Thus, the interactions of the circuit with other parts of the system must be taken into account. High-level modeling at the Transaction level (Transaction Level Modeling or TLM) allows the quick simulation of complex systems. Such high-level simulations with adequate fault modeling accelerate fault simulations and make possible to account for the whole system. Nevertheless, high-level modeling requires details to be removed (such as a clock or internal signals), which causes some realism issues.

Previous works [5, 6] have studied these issues. For example, results in [6] show that some of the faults injected at RT-Level have no high-level equivalent and conversely, some high-level faults have no RTL equivalent. Cross-layer methods aim to take advantage of fault injections in both RTL and TLM (or e.g., Matlab models) to take into account the whole system [7–11]

However, as far as we know, no previous study proposed a way to build relevant high-level models of the system and of the faults. Safe-Air approach presented in [11] is a cross-layer fault simulation approach allowing speeding up SEU FMEA on circuits used in critical applications. This approach is composed of three steps: (1) system high-level fault simulations, (2) block-level RTL fault simulations, and (3) evaluation of the high-level models. It allows the high-level model realism validation but this validation happens late in the process. In particular, no improvement of the high-level models related to the obtained results was proposed in [11].

As proposed in [12], an iterative process can lead to both perform an early robustness analysis and to improve the realism of the high-level models using the robustness analysis results. The method makes use of “quick and dirty” iterations to speed up the validation of the high-level models. Each round allows improving the high-level models that are simulated with a small number of randomly injected faults. The first rounds are performed on approximate models with a small number of randomly injected faults (dirty), in order to get a quick simulation. Then, the quality of the high-level fault simulation is evaluated through several metrics. The main novelty of this approach relies on these “quick and dirty” rounds with randomly injected faults and on the definition of several metrics.

Our main contributions, in particular with respect to [12], is to present a more detailed discussion with respect to the state-of-the-art, and to show a more detailed analysis of the results with a comparison between the results obtained and data from the initial case study.

This paper is organized as follows. More details on the state-of-the art are presented in Sect. 2. The Safe-Air methodology is briefly reminded in Sect. 3. The proposed iterative flow based on the “quick and dirty” evaluations is presented in 4. Section 5 presents and discusses results applied to a case study. Section 6 concludes the paper.

2 State-of-the-Art and Discussion

2.1 Fault Injections

For more than twenty years, an efficient technique to evaluate the robustness of electronic circuits is fault injection. The results are used to complete FMEAs performed by human specialists, specially to quantify the risks. There are mainly three categories of fault injection techniques: simulation, emulation and physical injection.

When using simulation, faults derived from a given model (e.g., SEUs) are voluntarily introduced in some elements at different execution times. The main advantage of simulation is the possibility to use system models designed with different abstraction levels, from TLM (Transaction Level Modeling) down to transistor-level or even physical device descriptions. Of course, the fault models have to be adapted to each level. Simulations at the highest levels are less precise but when coming to lower levels the simulation times are much higher and quickly become intractable, even when using statistical fault injections [13]. The lowest levels are therefore used to characterize small cells and the results are used to make the highest fault models more realistic.

Even simulations at RT-Level can lead to very long simulation times, thus requiring the use of emulation (or hardware prototyping). In that case, the system is implemented on a hardware platform where faults are injected using logic modifications during the application execution. The advantage is to speed-up the execution for each injected fault, compared to simulation, but at the expense of a non-negligible time to prepare the hardware set-up. Also, such emulations cannot be performed for all description levels of the system; in general, they are limited to RT-Level descriptions.

The third type of method that can be used is physical injection. There are mainly two techniques used to mimic the effects of environmental disturbances: laser [14, 15], or particle accelerators (e.g., [16]). Experiments in natural conditions (so-called life-time experiments) are also possible, in space or on the earth e.g., on mountains or underground, depending on the type of particle that has to be considered. Such experiments are in all cases very costly, especially when particles are used, and a lot of effort is required to prepare them. Furthermore, they need of course to have the final circuit available and any required improvement of the robustness implies a long and expensive process. Such experiments should therefore be restricted to the final product characterization and cannot be considered in early development phases.

Combining laser-based experiments, statistical fault injections and beam experiments was also proved worthwhile in [17]. However, once again, the final circuit must be available to complete the evaluation process.

As mentioned in the introduction, standards for critical embedded systems require an early evaluation of the robustness. Also, reducing costs and time to market imply having a quick identification of weak points in the system. We will therefore focus in the sequel on approaches based on behavioral simulation, aiming at reducing the time required to perform the fault injections while achieving an efficient early analysis of the system-level robustness.

Within this context, many authors have made proposals. We will cite the main approaches related to our proposal.

2.2 Simulated Fault Injections

Injection of faults during a simulation can use two types of approaches: with or without modifying the system description. Most works target RTL descriptions, but similar approaches can be used with TLM descriptions or at lower levels.

Avoiding modifications of the model requires the use of simulator commands to modify signals or variables at a given time. It is therefore not intrusive but depends on the simulator used. To avoid this dependency, it is possible to add control signals to the model, that command internal injection logic. This logic can be added to internal interconnections between blocks or gates in order to force the value of a signal; this is called saboteur insertion. Some types of errors cannot be injected using saboteurs and the behavioral model of the block must then be modified; this is called mutant generation. The pioneer work in [18] proposed both simulator commands and saboteurs. An example of mutant generation can be found in [19]. Of course, modifying the model of the system can become very complex and may also lead to functional errors. We will therefore avoid model modifications in our methodology.

As previously mentioned simulations even at RT-Level are very expensive when the system complexity increases. Furthermore, some errors observed at the circuit level have no real effect on the global system due to intrinsic tolerance properties on some parameters, to detection/tolerance logic or to masking effects during the error propagation. Thus, the global interactions within the system and the system specifications have to be taken into account to avoid a too pessimistic safety evaluation [7, 20].

In order to speed-up the evaluation, TLM models can be used. However, in that case many system characteristics are not yet defined (e.g., communication protocols or timing). Consequently, results are less accurate. This paper will show how it is possible to achieve both accelerated simulations and accuracy.

2.3 Limitations of High-Level Fault Models

Since most of the registers in the system are not yet defined, injecting faults in a TLM description is generally done using specific high-level fault models rather than for example SEU/MBU/MCU. Of course, saboteurs can be used but since the description is quite far from the following RTL model the results obtained are in general not very useful.

High-level modeling removes for example clocks or detailed data types, so some RTL faults have no high-level equivalent as shown in the case study presented in [21], based on a TLM SoC model written in SystemC.

Other works have studied this problem of fault model realism at high-level.

In [22], gate-level fault injections are used to define a realistic high-level fault library. The library is obtained from the circuit without safety mechanisms and used to perform high-level simulations after adding such mechanisms. Such an approach has several limitations. First the RTL description (and even the gate-level description) must be available, thus the high-level fault models are obtained very late in the design process. Then, gate-level simulations are very slow. Finally, the system specifications are not taken into account.

In [5], RTL and TLM faults are compared using a formal approach. More precisely, the TLM faults equivalent to an RTL fault are extracted. The study shows again that some RTL faults have no high-level representation, but also that a single RTL fault can have several equivalences at high-level.

In conclusion, there is no TLM fault model allowing a complete and realistic safety evaluation with this level of description.

2.4 Cross-Layer Approaches

Since a complete evaluation is not possible on high-level models only, many authors considered Cross-layer approaches. This can be used for example to co-simulate an RTL description and the gate-level description of some circuit blocks, but we will focus here on earlier evaluations.

The CLERECO project [8] aimed at performing cross-layer reliability evaluations in order to accelerate the analysis. However, it was focused on microprocessors and software.

Simulations are also performed at several levels in [9], to guide architecture and RTL hardware designers of a critical embedded system. The approach allows early decision making and avoids time-consuming design iterations. However, the problem mentioned in the previous section is not addressed and no verification is made about the realism of the results obtained at high-level. Also, the global system specifications are not taken into account to decide what errors are actually critical.

In [23], faults are injected at both high-level and RT-Level in a processor. The high-level description is used to simulate longer scenarios but there is no comparison between the results obtained at the two levels.

These examples show that there is usually no proof of relevance of the results obtained at high-level with respect to the results obtained at lower levels, and the scope is sometimes limited to a part of the system, in particular microprocessors.

2.5 Safety Analysis Tools

Tools have also been proposed to assist engineers when doing FMEA, in particular:

The AltaRica project [24] aimed at safety analyses in airborne systems. Systems are described at very high level with states and transitions, and critical scenarios are identified with a formal approach. This level of description is useful for the global system validation, but no link is made with hardware implementations.

A tool was proposed in [3], to help extracting quantitative information useful for FMEA. Here some specific parts of an RTL description are chosen and exhaustive fault injections are performed. The RTL description must therefore be available, and the complete system and its specifications are not taken into account.

3 Methodology Based on Cross-Layer Fault Simulation

As shown in the previous section, there are currently strong limitations to available approaches. The goal of the work presented hereafter is to reduce some of them. First, fault injections in high-level TLM models allow early robustness evaluations, but are not completely realistic. The other advantage of simulations at that level is to be able to take into account the whole system and its global specifications (no matter if it is completely implemented as a circuit or designed with several circuits, and even other types of components). It is however necessary, when a lower description level is made available, to check the accuracy of the results obtained at high-level. This can be done using a cross-layer approach, with RTL simulations made at the block level and keeping the TLM model for the whole system, thus noticeably reducing the simulation times compared to a full RTL simulation while being able to take into account the global system specifications. None of the approaches and tools presented in the state of the art has such characteristics. Our methodology allows extracting critical ranges at high-level, then verifying the relevance of the obtained statistics, and also refining the high-level fault models.

The cross-layer fault simulation methodology has been detailed in [25]. We give here an overview of it to keep the paper self-contained. In the sequel of the paper, we will insist more on the extensions of the approach that were presented in [12] and on the comparison with a classical FMEA for one case study.

Our methodology can be divided into three steps illustrated in Fig. 1.

Step 1: We perform a fault injection on a high-level system model at each interface signal of each block: a function corrupts the original value of signals by a fixed amplitude. We analyze the results according to the system specification. The signal corruptions that propagate to outputs and are not detected by any specific mechanism are analyzed to extract critical parameter ranges on the interface signals. Critical parameter ranges are converted into assertions to observe the behavior of these signals: if the assertion is not verified, the value of the signal is not in the range that can be tolerated and leads to a critical behavior of the whole system. This step allows detecting critical blocks.

Step 2: We statistically inject faults [13] in each RTL block leading to critical behaviors at the system level. Statistical fault injections are based on computing the number of faults required to achieve robustness evaluation with a predefined accuracy. The number

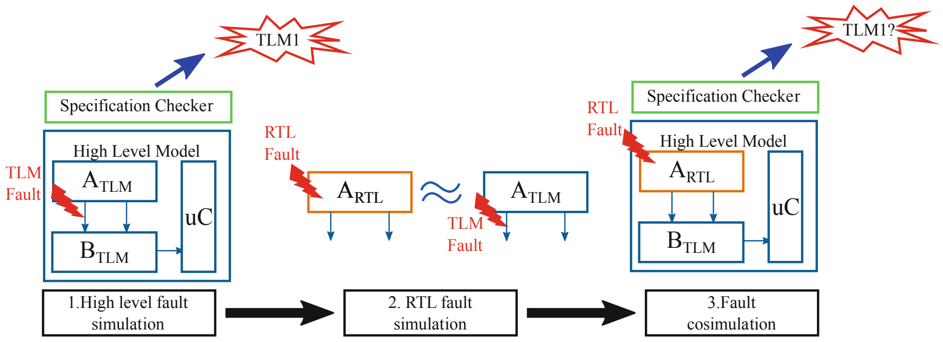

Fig. 1. The three steps in our cross-layer methodology.

of faults depends on three parameters: the number of possible faults (i.e., number of targets multiplied by the number of execution cycles), the margin of error, and the level of confidence (the probability that the exact value is in the margin of error). The usually chosen level of confidence is at least 90%. This method highly reduces the number of faults to inject while the confidence level remains high and the error margin can be reduced to a small percentage. Injected faults are simulated along with the assertions. The values of the assertions allow sorting faults into two categories: critical faults (if assertions are violated) and silent faults (if assertions are satisfied).

Step 3: It verifies the relevance of the high-level models used in step 1. Each RTL block is co-simulated within the high-level system model, and we inject the same faults on it as in step 2. Since the simulation is on the whole system but with a component modeled at lower level with more precise faults to inject, this step requires much more simulation time. Faults are sorted according to the observed behavior of the system. For each block, we obtain a confusion matrix (see Table 1). 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 as the reference behavior.

Table 1. Confusion matrix.

Co-simulation / Predicted	Critical	Silent
Critical	True Critical (TC)	False Critical (FC)
Silent	False Silent (FS)	True Silent (TS)

From this confusion matrix, we extract three metrics: the precision, the true silent rate, and the accuracy.

The precision is the proportion of true critical behaviors:

$$\text{Precision} = \text{TC} / (\text{TC} + \text{FC}) \quad (1)$$

A low precision indicates that too many faults are critical in step 2. The assertions are too restrictive, and critical parameter ranges are too large. Since critical parameter ranges are computed according to the fault amplitude, the low precision may be due to the high-level fault model used in step 1 that is not accurate enough: the amplitude of the fault should be modified.

The True Silent Rate (TSR) is the proportion of true silent behaviors:

$$\text{TSR} = \text{TS} / (\text{TS} + \text{FS}) \tag{2}$$

A low TSR means that assertions have not been able to detect correctly the critical faults. Since the critical behaviors are detected during the co-simulation, but not in the RTL simulation, it may mean that no assertion was generated to detect it. If the assertion is not created, it means that the critical behavior was not generated in step 1. This critical behavior is present at the RT-Level but not in the high-level model; it may be due to a bad high-level system modeling.

The accuracy defines the proportion of true results among injected faults:

$$\text{Accuracy} = (\text{TC} + \text{TS}) / (\text{TC} + \text{TS} + \text{FC} + \text{FS}) \tag{3}$$

This metric is a kind of trade-off between TSR and precision. It allows us to know the weight of each of the two other metrics. If the accuracy is high, but one of the two others is low, it indicates that the number of faults (either critical or silent) not correctly sorted in step 2 is small according to the whole number of faults. It may not be useful to modify the fault model (low precision) or the high-level model (low TSR).

4 Quick and Dirty Flow

The “quick and dirty” flow aims at highly speeding up the validation of high-level models and the overall evaluation of the system robustness. It is based on successive “quick and dirty” evaluations of the cross-layer fault simulation. It allows obtaining early evaluations of:

- the blocks leading to critical behaviors (step 1),
- the approximate probability of critical SEU (step 2),
- the realism of the high-level system model (step 3),
- the realism of the high-level fault model (step 3).

The flow is illustrated in Fig. 2. According to the design, we empirically define a threshold on metrics that indicates confidence in the high-level models. This threshold constrains the accuracy; we recommend at least 90%. Below, the quality of the high-level system model may not be sufficient. The next rounds depend on the value of the threshold:

- If the threshold is achieved, we perform a final statistical fault injection (arrow from the metric computations to step 2) with a very high confidence level and a low margin of errors to get an accurate evaluation of the probability of critical events. This last round may be very long but it is performed only once.

- If the threshold is not reached, the designer manually improves the high-level models. If the precision is low, he/she modifies the fault model. If the TSR is low, he/she analyzes the location of false silent faults to understand the origin of the model inaccuracy and to correct the system model. Then, a new round is performed (arrow from the metric computations to high-level models).

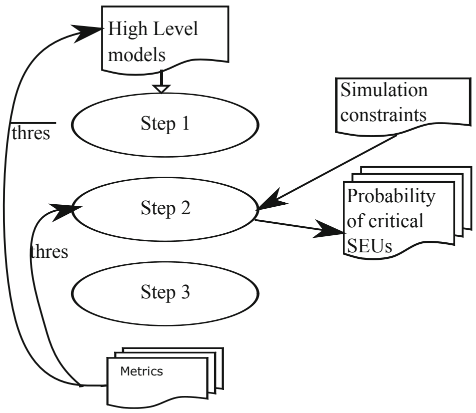

Fig. 2. Quick and dirty flow.

For each round, we define a maximum simulation time. This maximum time impacts the number of faults that can be injected in each block and therefore the couple (confidence level, margin of errors) that can be chosen. We evaluate the RTL simulation time of all blocks (simulation time of each block multiplied by the number of injected faults) for three confidence levels usually used in statistical fault injections (90%, 95%, and 99.8%) and several values of the margin of error. We then select a couple (confidence level, margin of error) that is below the maximum simulation time but still presents a correct robustness evaluation accuracy. During the first rounds, as the high-level model realism is not known, it is useless to obtain a very accurate robustness evaluation. Unaccurate (or dirty) evaluations allow us to quickly validate or improve high-level models. The simulation time per round can be progressively increased with the progress of the model quality.

5 Analysis Results on a Case Study

In this section, we apply the quick and dirty flow on a system converting the frequency of the oscillations of a sensor into a digital value. This design is used in aeronautic. For confidentiality reasons and without loss of generality, all names have been concealed, and all values normalized. The fault model used during the simulations is the SEU fault model (single bit-flips).

5.1 Case Study: System Description

We study an embedded system SYS_X that measures a physical value X, which is sent to the main airplane ECU (Electronic Control Unit). From other collected information, the ECU deduces the flight information. The system inputs the oscillations of a sensor which depend on the physical value X and converts the oscillation frequency into a digital value. Since SensorX is sensitive to temperature, a second sensor SensorT is used to capture the temperature, also generating oscillations. The architecture of the whole system is presented in Fig. 3. It contains a FPGA, a microcontroller, and a memory. Dividers and counters are used to obtain the digital values and are implemented in the FPGA. The microcontroller periodically reads the number of cycles counted during the measurement window of 100 ms (an interrupt occurs when new values are written in the output registers of CounterX and CounterT). The value of X is then computed based on a sensor table (series of points containing the values X corresponding to a given couple of periods of SensorT and SensorX) characterizing each type of sensorX. An interpolation method is used by the microcontroller 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 in charge of the communication. The communication block then formats the frame in order to transmit the data to the ECU.

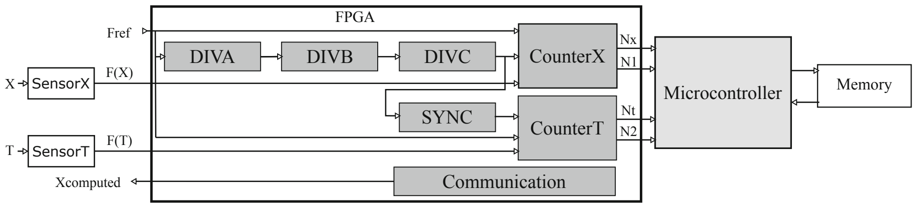

Fig. 3. Architecture of the system used for the case study.

Table 2. System specifications.

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	<100 ms
Computation period	<100 ms
Transport delay	300 ms ± 3.3 ms

The system specifications are summarized in Table 2. “Data Flow Rate” is the time between two value transmissions. The “Time to Alarm” property corresponds to the minimum error duration before notifying the error during data measurement. The “Transmission Period” is the time spent by the communication block to transmit the data.

"Computation Period" is the maximum time spent by the microcontroller to compute the final value and to transmit it to the communication block when an interrupt occurs. "Transport Delay" is the time between the beginning of a measure window and the data transmission by the communication block to the ECU. Several error detection mechanisms are also implemented in the system: counters overflow, detectors of periods out of the sensor table, detection of X value out of the possible range, etc. These system specifications are taken into account in our approach to classify an error as critical or not. As an example, the exact computation time of the controller is not critical; what is critical is to be under the maximum time allowed by the system. Also, a detected error is managed by the ECU or other redundant elements in the airplane and is therefore not critical.

5.2 Case Study: Initial FMEA

The usual FMEA process, managed mainly thanks to experienced engineers, is illustrated in Fig. 4. The global analysis involves several steps.

The first step is performed considering the architecture description (equivalent to our use of a high-level model, but in current practice it is most often a written description). At that step, a functional FMEA is performed in compliance with the ARP5580 standard (since it is an airborne system). This step early identifies the level of criticality of the different parts in the system. It only takes into account a subset of potential faults and all those are not necessarily realistic.

The second step goes into more details with respect to the implementation. It evaluates the probability of critical errors if a perturbation occurs in one block, especially in one block identified as critical during the first step. One part of this second step is focused on permanent failure of the block (or component) and is very global because this type of event is less frequent than the others, especially with the regular maintenance processes. The second part is focused on transient disturbances and in particular atmospheric particle effects for airborne systems. These events have a much higher probability than permanent failures. This is the SEU FMEA mentioned in introduction.

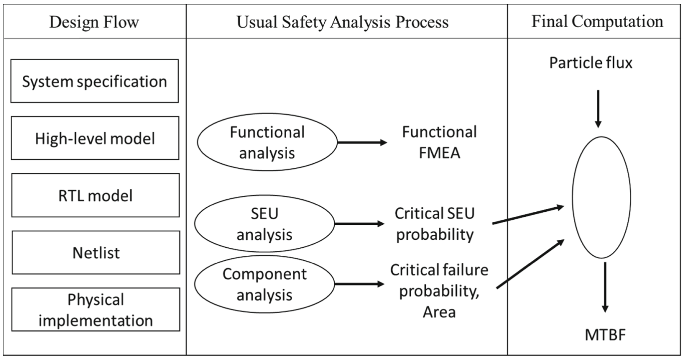

Fig. 4. Classical FMEA process.

Knowing the conditions in which the system will be used (e.g., flight altitude), a particle flux can be characterized and used to evaluate the flux per block (or component) with respect to the area of each block and to the sensitivity of the component in which it is implemented (depending on the technology). Taking into account the probability of critical errors if a SEU (or MBU/MCU) occurs in this block, it is possible to compute the probability of critical failures and, summing this over all the blocks, it is possible to derive the Mean Time Between Failures (MTBF) for the system.

Since the analysis is empirically managed by humans, any doubt about a potential SEU effect leads to the decision that it is critical in order to be conservative and the repartition between the types of failure modes is very rough. The result is often a pessimistic calculation of the MTBF, leading potentially to over-protect the system with consequences on cost and time-to-market. In addition, the analysis is only performed at the level of macro-components and with few failure modes, as illustrated in Table 3. This indicates the most critical components and the main failure modes but cannot give precisions on the criticality of each individual function in the architecture. In this case, the probability of faults per hour due to the particle flux is higher for the FPGA than for the microcontroller and the probability of each major failure mode is just estimated at 50%.

Table 3. Initial SEU FMEA for the case study.

Block	Particle flux per component (per hour)	Failure mode	Ratio
Microcontroller	0.03E−07	Erroneous X value	50%
		Reset	50%
FPGA	0.15E−07	Erroneous X value	50%
		No transmission	50%

For this system, taking into account before the actual implementation that about 25% of the logic resources should be used in the FPGA, and that all SEUs in these resources lead to a critical event, the critical error probability was evaluated around 1E-7 faults per flight hour for the functions in the FPGA.

5.3 Quick and Dirty Flow: First Round

We have modeled the whole system in SystemC at the transaction level approximately timed according to the architecture given in Fig. 3. Each block is implemented by a process. The interface signals between blocks are implemented with sc_fifo communication channels. When a new input is available on a communication channel, the process is notified. The input oscillations are directly modeled by the value of their frequency, i.e. an integer value.

Simulations allow identifying the level of disturbances that may be tolerated on the interface signals. This is illustrated in Table 4 for CounterX. Some signal corruptions can be tolerated thanks to the system specifications, but some others lead to characteristics out of range due to either amplitude or injection time. We can see for example that disturbances have no effect at 150 ms, but have a strong effect if at 100 ms or 200 ms, especially if the value is modified in the range −2% to +10% because in that case they are not detected by the available mechanisms. This leads to assertions, such as for this case:

Assert always window - > (Nx < Nx_gold * 0.98) or (Nx > Nx_gold * 1.1).
or (Nx = Nx_gold) report « ERR_ASSERTION»;

Table 4. Analysis at high-level of disturbances effects on CounterX (C in red means unacceptable, D in orange means detected, in green it is either silent or tolerable for the system).

	0 ms	50 ms	100 ms	150 ms	200ms
-50%	-	-	D	-	D
-40%	-	-	D	-	D
-30%	-	-	D	-	D
-20%	-	-	D	-	D
-10%	-	-	D	-	D
-5%	-	-	D	-	D
-4%	-	-	D	-	D
-3%	-	-	D	-	D
-2%	-	-	C	-	C
-1%	-	-	C	-	C
0%					
1%	-	-	C	-	C
2%	-	-	C	-	C
3%	-	-	C	-	C
4%	-	-	C	-	C
5%	-	-	C	-	C
10%	-	-	C	-	C
20%	-	-	D	-	D
30%	-	-	D	-	D
40%	-	-	D	-	D
50%	-	-	D	-	D

The first step of the cross-layer fault simulation leads to generate 8 assertions that are used in Step 2 to sort the behaviors. Those assertions monitor the signals on the communication interfaces. In blocks DIVA, DIVB, and SYNC, no fault leads to a critical behavior, so no assertion is generated for these blocks. Only blocks DIVC, CounterX, and CounterY lead to critical behaviors with respect to the system specifications.

To follow the quick and dirty flow, we chose a confidence level (c) greater than 90%, a margin of error (e) less than 10%, and a simulation time less than 3 h. In the third column of Table 5, we have the simulation time of each RTL block that allows us to compute the simulation time for a given number of injected faults. Figure 5 represents this simulation time according to the margin of error for different confidence levels. Three couples (c,e) are compliant with our constraints: either (90%, 4%), (95%, 5%) or (99.8%, 9%). Since a typical value for the confidence level is 95%, we choose the couple (95%, 5%). In Table 5 all the values are computed according to this couple. Columns 4 and 5 give the number of injected faults, then the corresponding simulation time. Columns 6 and 7 give the number of critical faults and the number of silent faults.

Table 5. RTL fault injection during the first round of the quick and dirty evaluation, for c = 95% and e = 5%.

Block	Number of possible faults	RTL simulation time per fault (s)	Number of faults to inject	Simulation time (min)	Number of critical faults	Number of silent faults
DIVA	2,500	1.1	333	6.1	0	333 (100%)
DIVB	1,500	1.1	305	5.6	0	305 (100%)
DIVC	18,750	1.1	376	6.9	37 (10.0%)	334 (90.0%)
SYNC	2,500	1.1	333	6.1	0	333 (100%)
CounterX	30E6	2	384	12.8	182 (47.5%)	202 (52.5%)
CounterT	150E6	21	384	134.4	157 (40.9%)	227 (59.1%)

The fault simulations of all RTL blocks may take 171 min. Since DIVA, DIVB, and SYNC were not considered critical at Step 1, all the faults are silent (there is no assertion) and the number of critical errors will be null. For this second step, it was therefore not useful to simulate them. For the other blocks, results show that some are more vulnerable than the others: in CounterX and CounterT, quite half of the faults lead to a critical error.

For the three blocks considered as critical at Step 1 and for which assertions have been generated, Table 6 illustrates the interest of taking the system-level specifications into account. In a classical fault injection process, all "tolerable faults" would be considered as critical, significantly reducing the MTBF and increasing the fault tolerance requirements.

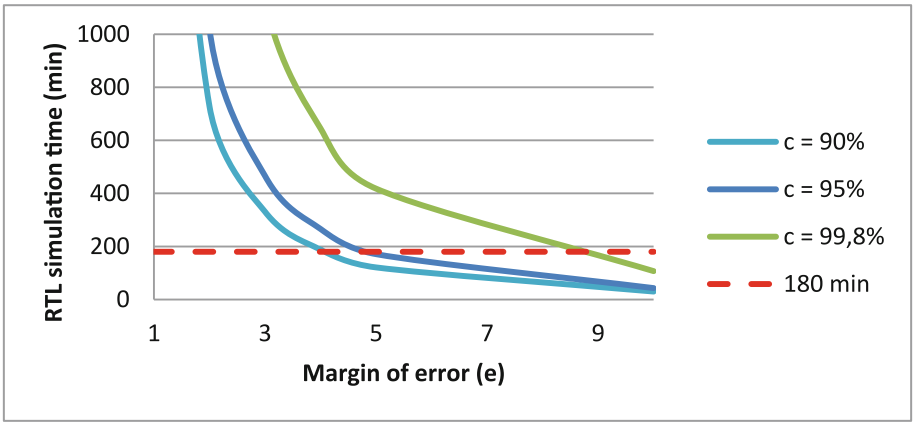

Fig. 5. RTL simulation time according to the margin of error.

Table 6. Classification of injected faults at RTL taking into account assertions with respect to system-level specifications ("tolerable faults"), for c = 95% and e = 5%.

Block	Number of critical faults	Number of tolerable faults	Number of silent faults
DIVC	37 (10%)	79 (21.0%)	260 (69.0%)
CounterX	182 (47.5%)	165 (42.9%)	37 (9.6%)
CounterT	157 (40.9%)	35 (9.1%)	192 (50%)

The co-simulation (Step 3) takes 278 min. The same faults as in Step 2 are injected, and in this step randomly chosen bit flips are also injected in DIVA, DIVB, and SYNC. We extract for each block the precision, the accuracy, and the True Silent Rate. The first column of Table 7 gives the name of the blocks. Simulation times for each block are displayed in the second column (co-simulation time). In the three last columns, the different metrics are reported, indicating the quality of the high-level models. Let us recall that we fix a threshold for the accuracy at 90%. The precision for blocks DIVA, DIVB, and SYNC is not applicable since they were classified as not critical at step 1. For these blocks, few fault injections led to errors and all of those were tolerated with respect to the system specifications. Block DIVC presents a good accuracy, a perfect TSR, but a very low precision (many critical faults identified in Step 2 turn out to be not critical in Step 3). Since the accuracy threshold is achieved, the low precision rate is not relevant. We do not need to change the high-level fault model.

The accuracy of block CounterX is below the threshold. The two other metrics are relevant. The TSR is very low: quite half the critical errors in Step 3 have not been correctly identified at Step 1. We must refine the high-level model to be closer to reality. We will not dwell on CounterT that is a different instantiation of CounterX.

Table 7. Results of the quick and dirty analysis at Step 3, 1st round.

Block	Co-simulation time (min)	Accuracy	Precision	True silent rate
DIVA	61	100%	N.A	100%
DIVB	45.75	100%	N.A	100%
DIVC	9.4	92%	2.8%	100%
SYNC	6.1	100%	N.A	100%
CounterX	96	71.6%	96.5%	55%
CounterT	70.4	78.1%	92.4%	69.8%

5.4 Quick and Dirty Flow: Next Rounds

CounterX (see Fig. 6) is a block that counts the number of impulsions on signals Fref and F(X) between two impulsions of signal F(D).

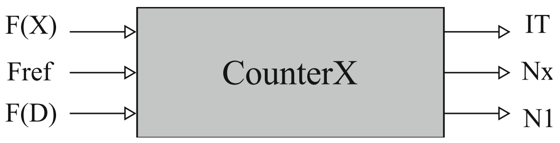

Fig. 6. CounterX RTL model.

Without loss of generality, we will focus only on signal F(X). Signal IT indicates that a fresh value on Nx is available and it interrupts the microcontroller. The behavior of CounterX is illustrated in Fig. 7. Between the two impulsions of F(D), there are 3 impulsions of F(X). On the last impulsion of F(D), signal Nx takes value 3, and signal IT is enabled.

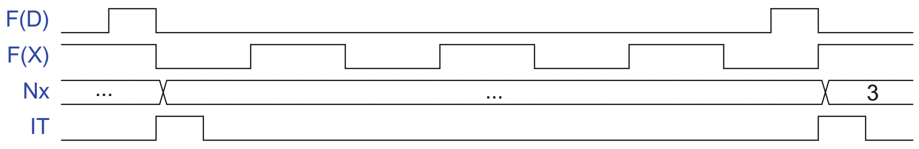

Fig. 7. CounterX behavior.

In the high-level system model, signals F(X) and F(D) were integers and not impulsions. Output Nx was the result of the division of F(X) by F(D). Since Nx is modeled with sc_fifo communication channels that notify the microcontroller when new data is available, signal IT is not modeled.

To improve the quality of the high-level CounterX model, we accurately analyzed the injection results of Step 3. Figure 8 shows the distribution of false silent faults per register. We see that less than 10 registers are sensitive to false silent faults. All these registers are used to compute signal IT.

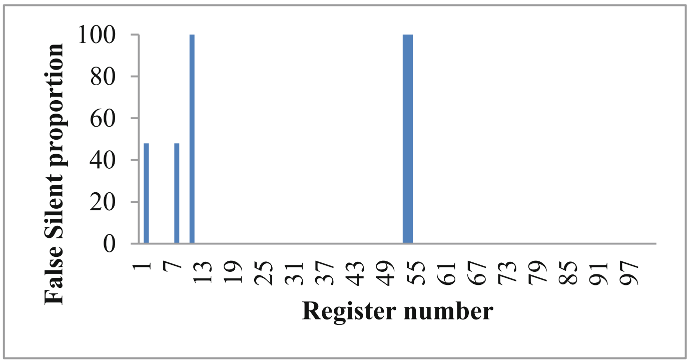

Fig. 8. False silent fault distribution per register.

At the system level, the faults we injected modified the amplitude of the values of Nx but did not modify the instant when they were available. Figure 9 illustrates this behavior: processes CounterX and uC are intrinsically synchronized through the sc_fifo communication channel. At the RT-Level, injected faults modify signal IT so no fresh data may be sent to the microcontroller. It is illustrated in Fig. 10: at T0, signal IT is correct and the microcontroller reads a value Nx1, but at time T1, signal IT is corrupted, CounterX has not finished its computation, and the microcontroller reads value Nx1 again.

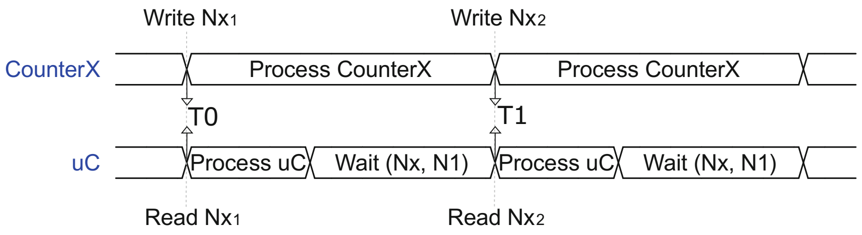

Fig. 9. CounterX TLM model synchronization with the microcontroller.

To avoid this problem, we changed the high-level model: an interruption signal was explicitly described. The second round of the quick and dirty method is then performed. At the end of this round, we get the metrics displayed in Table 8. We observe that in step 2, more faults are critical (column 2), both in CounterX and in CounterT. It means that

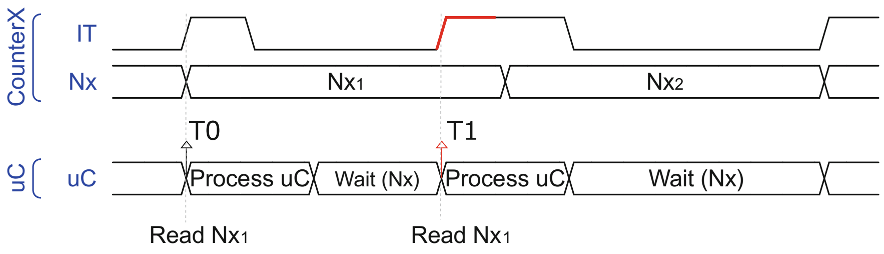

Fig. 10. CounterX RTL model with fault injection propagating on IT signal.

these blocks are more critical than initially assumed. TSR and accuracy are considerably improved. The threshold on the accuracy is reached, we are confident on the high-level model, and we can stop the quick and dirty rounds.

Table 8. Results of the quick and dirty analysis at Step 3, 2nd round, for c = 95% and e = 5%.

Block	Critical (step 2)	Accuracy	Precision	True silent rate
CounterX	73.4%	94.4%	93.7%	95.4%
CounterT	56.5%	95.4%	95.3%	95.3%

Table 9. Results of the quick and dirty analysis at 3rd round for c = 99.8% and e = 2%.

Block	Critical (1st round)	Critical (3rd round)	Silent (1st round)	Silent (3rd round)
DIVA	0%	0%	100%	100%
DIVB	0%	0%	100%	100%
DIVC	10.2%	10.2%	89.8%	89.8%
SYNC	0%	0%	100%	100%
CounterX	47.5%	73.4%	52.5%	26.6%
CounterT	40.7%	70.2%	59.3%	29.8%

The final step is to improve the accuracy of the robustness evaluation in Step 2. We increase the confidence level and decrease the margin of error. With a confidence level of 99.8% and a margin of error of 1%, we inject 23,855 faults in CounterX for a simulation time of 14 h. The percentage of critical faults is 73.8% (very close to the percentage obtained after the second round). More complete results are shown in Table 9, with a global simulation time of 41 h.

5.5 Comparison Between the Proposed Flow and the Initial FMEA

The comparison with the initial FMEA stands on two aspects: the functional analysis and the SEU analysis.

For the functional analysis, the simulations of high-level models allows getting a quick and automatic first insight about critical blocks or components in the architecture, more precise that what can be done by an expert from paper specifications. In addition, it is possible to get data about the criticality range of some blocks and timings as illustrated in Table 4.

For the SEU analysis, several strong points can be mentioned. First, with the "quick and dirty" flow, it is possible to keep simulation times acceptable and at the same time to refine the high-level models. Second, results can be obtained not only on components (e.g., FPGA) but also on the different blocks implemented inside the component, as illustrated in Table 10. This is much more precise than the type of analysis illustrated in Table 3 and can help a designer in focusing protection techniques on the most critical parts e.g., in that case, the counters. Also, comparing the figures in Table 10 and the initial estimation for the FPGA component (with the same flux of neutrons but with a conservative assumption that each SEU is a critical fault) the evaluated probability of critical errors is clearly reduced (4.62E−8 vs. 1E−7), leading to a better MTBF and potentially to a reduced need of additional design.

Table 10. Results after the last round assuming a particle flux of $\alpha = 7.2E-11$ neutrons/bit/h.

Block	Critical (3rd round)	Criticality per block (f/h)
DIVA	0%	0
DIVB	0%	0
DIVC	10.2%	1.59E−9
SYNC	0%	0
CounterX	73.4%	2.28E−8
CounterT	70.2%	2.18E−8
Total		4.62E−8

6 Conclusion

This contribution presents an iterative flow to quickly and accurately evaluate both circuit robustness with a cross-layer methodology and improving the realism of high-level models. This approach complies with aeronautics standards that recommend performing robustness analysis at different modeling levels and early in the design flow, from functional level models when possible. The method is based on iterative statistical fault injections that allow during first rounds quick fault simulations with an adapted level of confidence and improvement of the high-level models. When the high-level models are accurate enough, RTL fault simulations of each block with a high number of faults are performed. These RTL fault simulations determine if the faults are critical using the critical parameter ranges given by system high-level fault simulations. The proposed flow allows evaluating quickly and accurately the robustness of each block. It also allows identifying the most critical SEUs and then proposing fair cost countermeasures. This method has been applied to a critical airborne embedded system and demonstrated that, after improvement of high-level models, some RTL blocks were more critical than others. In comparison with the human FMEA analysis available as reference, much more details are available on the different blocks used in the application. The global MTBF is better and so should reduce the need for over-design, while the additional information allows focusing the required protections on the most critical blocks.

At that time, the approach has been applied to only one "real-life" case study. Results are promising, but in future works, we will apply this approach to other test cases. Also, metrics and thresholds used to measure the realism of the high-level models will have to be assessed further.

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

References

1. Recommended failure modes and effects analysis FMEA practices for non-automobile aplications ARP5580, Standard (2012). www.sae.org/standards/content/arp5580/
2. Fulton, R., Vandermolen, R.:Airborne Electronic Hardware Design Assurance: A Practitioner's Guide to RTCA/DO-254. CRC Press, Boca Raton (2014). ISBN: 1482206056
3. Mariani, R., Boschi, G., Colucci, F.: Using an innovative SoC-level FMEA methodology to design in compliance with IEC61508. In: 2007 Design, Automation and Test in Europe Conference Exhibition Proceedings, pp. 1–6 (2007)
4. Bernard, R., Aubert, J.-J. , Bieber, P., Merlini, C., Metge, S.: Experiments in model based safety analysis: flight controls. In: 1st IFAC Workshop on Dependable Control of Discrete Systems Proceedings (2007)
5. Herdt, V., Le, H.M., Grosse, D., Drechsler, R.: 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) Proceedings, pp. 1–8 (2016)
6. Miele, A.: A methodology for the design and the analysis of reliable embedded systems. Ph. D. Dissertation, Politecnico di Milano (2010)
7. Leveugle, R., Cimonnet, D., Ammari, A.: System level dependability analysis with RT-level fault injection accuracy. In: 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems Proceedings, Cannes, France, pp. 451–458 (2004)
8. Vallero, A., et al.: Cross-layer reliability evaluation, moving from the hardware architecture to the system level: a CLERECO EU project overview. Microprocess. Microsyst. **39**(8), 1204–1214 (2015)
9. Perez, J., Azkarate-Askasua, M., Perez, A.: Codesign and simulated fault injection of safety-critical embedded systems using SystemC. In: 2010 European Dependable Computing Conference Proceedings, pp. 221–229 (2010)

10. Mueller-Gritschneder, D., Maier, P.R., Greim, M., Schlichtmann, U.: System C-based multi-level error injection for the evaluation of fault-tolerant systems. In: 2014 International Symposium on Integrated Circuits (ISIC) Proceedings, pp. 460–463 (2014)
11. Roux, J., et al.: High level fault injection method for evaluating critical system parameter ranges. In: 27th IEEE International Conference on Electronics Circuits and Systems (ICECS) Proceedings, pp. 1–4 (2020)
12. Roux, J., et al.: Cross-layer approach to assess FMEA on critical systems and evaluate high-level model realism. In: 29th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC) Proceedings, Singapore (2021)
13. Leveugle, R., Calvez, A., Maistri, P., Vanhauwaert, P.: Statistical fault injection: quantified error and confidence. In: 2009 Design, Automation Test in Europe Conference Exhibition Proceedings, pp. 502–506 (2009)
14. Habing, D.H.: The use of lasers to simulate radiationinduced transients in semiconductor devices and circuits. IEEE Trans. Nucl. Sci. **12**(5), 91–100 (1965)
15. Pouget, V., Lewis, D., Lapuyade, H., Briand, P.F.R., Sarger, L., Calvet, M.-C.: Validation of radiation hardened designs by pulsed laser testing and spice analysis. Microelectron. Reliab. **39**, 931–935 (1999)
16. Constantinescu, C.: Neutron SER characterization of microprocessors. In: 2005 International Conference on Dependable Systems and Networks (DSN) Proceedings, pp. 754–759 (2005)
17. Guibbaud, N., Miller, F., Molière, F., Bougerol, A.: New combined approach for the evaluation of the soft-errors of complex ICs. IEEE Trans. Nucl. Sci. **60**(4), 2704–2711 (2013)
18. Jenn, E., Arlat, J., Rimen, M., Ohlsson, J., Karlsson, J.: Fault injection into VHDL models: the MEFISTO tool. In: Randell, B., Laprie, JC., Kopetz, H., Littlewood, B. (eds) Predictably Dependable Computing Systems, pp. 66–75. Springer, Cham (1994). https://doi.org/10.1007/978-3-642-79789-7_19
19. Leveugle, R., Hadjiat, K.: Optimized generation of VHDL mutants for injection of transition errors. In: 13th Symposium on Integrated Circuits and Systems Design (SBCCI2000) Proceedings, pp. 243–248 (2000)
20. Champon, R., Beroulle, V., Papadimitriou, A., Hely, D., Genevrier, G., Cezilly, F.: Comparison of RTL fault models for the robustness evaluation of aerospace FPGA devices. In: IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS) Proceedings, pp. 23–24 (2016)
21. Miele, A.: A methodology for the design and the analysis of reliable embedded systems. Ph.D. Thesis, Politecnico di Milano (2010)
22. Tabacaru, B., Chaari, M., Ecker, W., Kruse, T., Novello, C.: Fault-effect analysis on system-level hardware modeling using virtual prototypes. In: 2016 Forum on Specification and Design Languages (FDL) Proceedings, pp. 1–7 (2016)
23. Mueller-Gritschneder, D., Maier, P.R., Greim, M., Schlichtmann, U.: System C-based multi-level error injection for the evaluation of fault-tolerant systems. In: 2014 International Symposium on Integrated Circuits (ISIC) Proceedings, pp. 460–463 (2014)
24. Bernard, R., Aubert, J.-J., Bieber, P., Merlini, C., Metge, S.: Experiments in model based safety analysis: flight controls. IFAC Proc. Vol. **40**(6), 43–48 (2007)
25. Roux, J., et al.: High-level fault injection to assess FMEA on critical systems. Microelectron. Reliab. **122**, 114–135 (2021)