Injecting Intermittent Faults for the Dependability Assessment of a Fault-Tolerant Microcomputer System

Daniel Gil-Tomás, Joaquín Gracia-Morán, J. Carlos Baraza-Calvo, Luis J. Saiz-Adalid and Pedro J. Gil-Vicente, Member, IEEE

Abstract—As scaling is more and more aggressive, intermittent faults are increasing their importance in current deep submicron complementary metal-oxide-semiconductor (CMOS) technologies. This work shows the dependability assessment of a fault-tolerant computer system against intermittent faults. The applied methodology lies in VHDL-Based Fault Injection, which allows the assessment in early design phases, together with a high level of observability and controllability. The evaluated system is a duplex microcontroller system with cold stand-by sparing. A wide set of intermittent fault models have been injected, and from the simulation traces, coverages and latencies have been measured. Markov models for this system have been generated and some dependability functions, such as reliability and safety, have been calculated. From these results, some enhancements of detection and recovery mechanisms have been suggested. The methodology presented is general to any fault-tolerant computer system.

Index Terms—Fault injection, Hardware description languages, Intermittent faults, Dependability, Markov models

ACRONYMS AND ABBREVIATIONS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VLSI</td>
<td>Very large scale of integration</td>
</tr>
<tr>
<td>CMOS</td>
<td>Complementary metal-oxide-semiconductor</td>
</tr>
<tr>
<td>CISC</td>
<td>Complex instruction set computing</td>
</tr>
<tr>
<td>RISC</td>
<td>Reduced instruction set computing</td>
</tr>
<tr>
<td>FT</td>
<td>Fault tolerant</td>
</tr>
<tr>
<td>EDMs</td>
<td>Error detection mechanisms</td>
</tr>
<tr>
<td>ERMs</td>
<td>Error recovery mechanisms</td>
</tr>
<tr>
<td>CPU</td>
<td>Central processing unit</td>
</tr>
<tr>
<td>RT</td>
<td>Register-transfer (related to abstraction level)</td>
</tr>
<tr>
<td>VHSIC</td>
<td>Very high speed integrated circuit</td>
</tr>
<tr>
<td>VHDL</td>
<td>Very high speed integrated circuit</td>
</tr>
<tr>
<td>VFI</td>
<td>VHDL-based Fault Injection Tool</td>
</tr>
<tr>
<td>WDT</td>
<td>Watchdog timer</td>
</tr>
<tr>
<td>CP</td>
<td>Checkpointing</td>
</tr>
</tbody>
</table>

μs microseconds (10^-6 seconds)
ns nanoseconds (10^-9 seconds)
R reliability
S safety
SURE Semi-Markov Unreliability Range Evaluator

I. INTRODUCTION

RELIABILITY has become a major challenge in current computer systems. The feature size scaling of very-large-scale-of-integration (VLSI) integrated circuits, together with the reduction of the supply voltage, the increase of transistor speed and the significant sensitivities to temperature and electromagnetic noise, have led to the apparition of new faults. In this context, intermittent faults are increasingly important in current deep submicron complementary metal-oxide-semiconductor (CMOS) technologies [1]. New defects, residues, process variations and wear-out mechanisms may provoke intermittent faults. These faults occur non-deterministically at the same location due to unstable or marginal hardware. Although errors induced by transient and intermittent faults manifest in a similar way, intermittent faults are activated repeatedly in the same place, and hence, they are usually grouped in bursts. Additionally, intermittent faults may be activated or deactivated by changes in temperature, voltage or frequency [2].

Transient and permanent faults have been deeply studied and their models are well established [3], [4]. Permanent faults are provoked by irreversible physical defects caused by manufacturing defects and wear-out mechanisms. Transient faults are commonly generated by environmental conditions, like electromagnetic interferences or cosmic radiation. Instead, intermittent faults have been traditionally much less studied, and they have been typically considered as the prelude of permanent faults provoked by wear-out processes. In this way, intermittent fault models have been usually assimilated to those corresponding to permanent faults.

Nevertheless, their characterization is very complex, as they appear randomly and manifest in high rate bursts that can disappear and appear later. In recent works, deeper studies have been carried out using logged errors provoked by intermittent faults in real computing systems [1], [2], [5]. In these works, frequent sources of errors and their manifestation have been analyzed, and some mitigation techniques are suggested.
In former works, we have studied the impact of intermittent faults in different microprocessors (with both complex instruction set computing (CISC) and reduced instruction set computing (RISC) architectures) at gate and register-transfer abstraction levels [6], [7], [8]. From the analysis of real logged faults and the study of the physical mechanisms involved, we have generated new fault models for intermittent faults. Then, we have injected these intermittent fault models into the VHDL models of these systems. Finally, from the simulation traces obtained in the injection experiments, failures and latent errors have been registered, and the influence of different fault parameters has been analyzed. Other works in the literature have studied the impact of intermittent faults at higher abstraction levels (application programs) [9], [10], and they all show similar results as in [6], [7] and [8] regarding the influence of fault parameters and sensitivities.

In the present work, we go a step further and assess the dependability of a fault-tolerant (FT) computer system against intermittent faults. This is related to another research area in intermittent faults: the design and validation of fault tolerance mechanisms to cope with intermittent faults. In early tentative works, a FT microcomputer system was partially assessed. In [11], transient and permanent faults were injected, and coverages and latencies were calculated. In [12], intermittent faults were injected in some prospective targets, and their impact on the behavior of the detection and recovery mechanisms was analyzed. The objective of the present work is to complete the evaluation process against intermittent faults, as we explain in the following paragraphs.

Fig. 1 summarizes the methodology followed. First, fault models for intermittent faults are deduced, and injected in the VHDL model of the FT system (left branch of the graph). From the analysis of the faulty simulation traces, coverages and latencies are measured. This information can be used, in a feedback process (represented with dashed lines), to improve the Fault Tolerance Mechanisms and update the system, and hence, its VHDL model. Next, coverage values are introduced in the Markov model of the system, generated to represent the behavior of the FT system in the presence of intermittent faults. Some dependability functions, such as reliability and safety, are obtained by solving the Markov model. Finally, these results can be compared to those obtained using other fault tolerance approaches, for instance applying alternative redundancy techniques. The overall process can be repeated iteratively in a feedback path to enhance the design. We have applied a VHDL-Based Fault Injection technique for two main reasons: i) it allows the assessment in early phases of the system design; and ii) it permits high controllability and observability of the experiments.

Other related works evaluate intermittent error recovery configurations at a high level. Some of these configurations are similar to those of our paper. For instance, [13] models a multicore system at the functional level, while [14] models a multiprocessor chip based on Stochastic Activity Networks. In these works, no dependability functions (such as reliability, safety, etc.) are obtained. Instead, some performance estimators (such as throughput, latency, overhead, useful work, etc.) are calculated. On the other hand, in [15] the tolerance to intermittent faults is designed at the algorithm level and applied to a distributed system. A formal proof establishes its correctness. From this point of view, our paper may strengthen dependability assessment, introducing some novelties with respect to the related literature.

In summary, the contribution of our paper with respect to related literature consists in: i) Calculating coverages and latencies for intermittent faults; ii) Generating Markov models of the FT system in the presence of intermittent faults; iii) Obtaining dependability functions to assess the FT system. The methodology presented in this work is general, and it can be applied to any FT system at an early phase of the design.

This work is organized as follows. Section II depicts the fault models for intermittent faults. Section III introduces the fault injection technique used. Section IV describes the fault-tolerant system and the main components of the VHDL model. Sections II to IV match with the left branch in Fig. 1. Section V explains the parameters of the fault injection experiments. Section VI discusses the results obtained regarding the coverages and latencies. Sections V and VI correspond to the “Detection/recovery coverages and latencies” box in Fig. 1. Section VII describes the Markov models generated and some results obtained about the reliability and safety of the system. This section covers the right branch in Fig. 1. Finally, Section VIII provides some conclusions.
II. INTERMITTENT FAULT MODELING

An intermittent fault is defined as a fault that appears sporadically at the same hardware location, and lasts for one or more clock cycles [14]. Intermittent faults can be activated by environmental changes such as temperature, voltage, or frequency alterations. In addition, manufacturing defects, process variations and special wear-out processes can also lead to such faults. In some cases, intermittent faults can evolve to permanent faults due to aging mechanisms. The introduction of new deep submicron technologies accentuates the occurrence of intermittent faults and makes necessary to study their new fault causes and mechanisms.

Whereas transient and permanent fault models have been traditionally well established, modeling intermittent faults is a pending issue [1]. The unpredictable behavior of intermittent faults makes it difficult to define fault models. In previous works we have deduced a set of fault models at the logic and memory level which can be simulated into VHDL models [6], [7]. These fault models are summarized in Table I.

As stated previously, intermittent faults manifest in bursts. To model these faults, the following parameters must be specified (see Fig. 2) [8]: the number of fault activations in the burst (or burst length, $L_{\text{burst}}$), the duration of each activation (or activity time, $t_A$), and the separation between two consecutive activations (or inactivity time, $t_I$). For the sake of simplicity, all three parameters have been generated according to uniform distribution functions.

III. FAULT INJECTION ENVIRONMENT

In this paper, we use a VHDL-based fault injection technique, which is a simulation-based technique. This technique allows the assessment of the system in early design phases. And due to the features of the VHDL modeling language, this technique provides high controllability and observability of the injection experiments, as it permits the modification and monitoring of every element in the system. Fig. 3 shows different ways to implement the VHDL-based fault injection techniques [16] [4]:

![Fig. 2. Main elements of a burst of an intermittent fault.](image)

![Fig. 3. VHDL-based fault injection techniques.](image)

The simulator-commands-based fault injection (or simply simulator-commands) technique consists in modifying the value or timing of the signals and/or variables of the VHDL model at simulation time, by using special commands of the simulator.

The saboteurs-based fault injection (or simply saboteurs) technique requires modifying the VHDL code of the system by inserting injection components called saboteurs between the components of the model. While inactive, a saboteur does nothing, but when activated it can alter the value or timing characteristics of one or more signals, simulating the

<table>
<thead>
<tr>
<th>Causes</th>
<th>Targets</th>
<th>Fault mechanisms</th>
<th>Type of fault</th>
<th>Fault models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Residues in cells</td>
<td>Memory and registers</td>
<td>Intermittent contacts</td>
<td>Manufacturing defect</td>
<td>Intermittent stuck-at</td>
</tr>
<tr>
<td>Solder joints</td>
<td>Buses</td>
<td>Intermittent contacts</td>
<td>Manufacturing defect</td>
<td>Intermittent pulse</td>
</tr>
<tr>
<td>Electromigration</td>
<td>Buses</td>
<td>Variation of metal resistance</td>
<td>Wearout-Timing</td>
<td>Intermittent delay</td>
</tr>
<tr>
<td>Delamination</td>
<td>I/O connections</td>
<td>Voids</td>
<td>Wearout-Timing</td>
<td>Intermittent short</td>
</tr>
<tr>
<td>Crosstalk</td>
<td>I/O connections</td>
<td>Electromagnetic interference</td>
<td>Internal noise</td>
<td>Intermittent stuck-at</td>
</tr>
<tr>
<td>Gate oxide soft breakdown</td>
<td>NMOS transistors in SRAM cells</td>
<td>Leakage current fluctuation</td>
<td>Wearout-Timing</td>
<td>Intermittent delay</td>
</tr>
<tr>
<td>Negative bias-temperature instability (NBTI)</td>
<td>PMOS transistors in combinational logic</td>
<td>Increase of transistor threshold voltage $V_{TH}$</td>
<td>Wearout-Timing</td>
<td>Intermittent delay</td>
</tr>
<tr>
<td>Hot-carrier injection (HCI)</td>
<td>NMOS transistors in combinational logic</td>
<td>Increase of transistor threshold voltage $V_{TH}$</td>
<td>Wearout-Timing</td>
<td>Intermittent delay</td>
</tr>
<tr>
<td>Low-k dielectric breakdown</td>
<td>Buses</td>
<td>Leakage current fluctuation</td>
<td>Wearout-Timing</td>
<td>Intermittent delay</td>
</tr>
<tr>
<td>Doping profile and gate length deviations</td>
<td>MOS transistors in combinational logic and memory</td>
<td>Deviations in $V_{TH}$</td>
<td>Manufacturing variations</td>
<td>Intermittent delay</td>
</tr>
</tbody>
</table>

The simulator-commands-based fault injection (or simply simulator-commands) technique consists in modifying the value or timing of the signals and/or variables of the VHDL model at simulation time, by using special commands of the simulator.

The saboteurs-based fault injection (or simply saboteurs) technique requires modifying the VHDL code of the system by inserting injection components called saboteurs between the components of the model. While inactive, a saboteur does nothing, but when activated it can alter the value or timing characteristics of one or more signals, simulating the

<table>
<thead>
<tr>
<th>TABLE I. INTERMITTENT FAULT MECHANISMS AND MODELS [7]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Causes</td>
</tr>
<tr>
<td>-------------------------------</td>
</tr>
<tr>
<td>Residues in cells</td>
</tr>
<tr>
<td>Solder joints</td>
</tr>
<tr>
<td>Electromigration</td>
</tr>
<tr>
<td>Delamination</td>
</tr>
<tr>
<td>Crosstalk</td>
</tr>
<tr>
<td>Gate oxide soft breakdown</td>
</tr>
<tr>
<td>Negative bias-temperature instability (NBTI)</td>
</tr>
<tr>
<td>Hot-carrier injection (HCI)</td>
</tr>
<tr>
<td>Low-k dielectric breakdown</td>
</tr>
<tr>
<td>Doping profile and gate length deviations</td>
</tr>
</tbody>
</table>
occurrence of faults.

The *mutants-based fault injection* (or *mutants*) technique also requires modifying the VHDL code of the system. In this case, altered versions of the existing components, called *mutants*, are created. These mutated versions of the components present different wrong behaviors of the component. If no mutant is activated, the system behaves normally. By activating a combination of mutants, the occurrence of faults is simulated.

Finally, *Other techniques* are implemented by extending the VHDL language, adding new data types and signals and modifying the VHDL resolution functions. The new elements defined include descriptions of faulty behaviors.

Comparing the cost of implementing each method, it is worth to mention that:

- The simulator-commands technique is the easiest one to implement, and it introduces the lowest overhead.
- The main drawbacks of the saboteurs technique are two: i) it is more complex to implement than the simulator-commands; ii) it adds spatial overhead to the model. As an important advantage, it allows injecting a wider set of fault models.
- The mutants technique has the same drawbacks and advantages as saboteurs. Moreover, it has an additional issue: the fault modeling becomes hard to apply at lower abstraction levels, because it is very difficult to associate VHDL code mutations to hardware faults.
- The other techniques require the introduction of ad-hoc compilers and control algorithms to manage the language extensions, which makes them the most difficult to implement.

Our research group has developed a fault injection tool called VFIT (VHDL-based Fault Injection Tool) [4], which is able to inject faults by means of *simulator commands*, *saboteurs* and *mutants* techniques.

Recent works on fault injection show other platforms that perform simulation-based fault injection. For example, [9] injects faults in a model of a simple five-stage pipeline RISC processor by using the *Sim-Outorder* processor simulator, while [17] injects faults on Verilog models and allows FPGA accelerated full-system simulation on prototype hardware.

IV. SYSTEM UNDER STUDY

The different fault injection experiments were carried out on the VHDL model of a fault-tolerant microcomputer system, whose block diagram is shown in Fig. 4. The system is duplex with cold stand-by sparing, parity detection and a watchdog timer [11]. Although the system under study is an academic FT system, its structure is common in non-critical FT systems, such as long-life and high availability systems [3].

Both the main and the spare processors are an enhanced version of the MARK2 processor, developed by J.R. Armstrong in 1989 for academic purposes [18]. It has been extended to 16 bits, and several FT mechanisms have been added.

The structural architecture of the model is composed of the main and spare CPUs (CPUA and CPUB, respectively), the random access memory (MEM), the output parallel port (PORTOUT), the interrupt controller (SYSINT), the clock generator (CLK), the watchdog timer (WD), the pulse generator (GENINT), two back-off cycle generators (TRGENA, TRGENB) and two AND gates (PAND2A, PAND2B). Each component is modeled by a behavioral architecture with usually one or more concurrent processes.

Fig. 4. Block diagram of the FT computer system [11].
To increase the system dependability, both Error Detection Mechanisms (EDMs) and Error Recovery Mechanisms (ERMs) have been incorporated. The EDMs include a parity check in the memory and a program control flow check performed by a watchdog timer. The watchdog timer technique implements a timing check of a process. When the timer goes off, the system can assume that either the processor is hung or a control flow error in the code of the process has occurred [19].

The ERMs include the introduction of a back-off cycle (instruction retry) after parity error detection, checkpointing when errors are detected by the watchdog timer, and starting the spare processor in case of permanent errors. The number of successive error detections required to activate the spare CPU can be configured in the system.

V. FAULT INJECTION EXPERIMENTS

A. Fault injection technique

Faults have been injected by using VFIT, the fault injection tool developed by our research group. The different injection experiments presented in this paper have been carried out using the simulator-commands-based technique, because this technique is the easiest to apply and allows injecting all the fault models selected in the experiments. The fault injection process can be summarized in the following steps:

1. Setup of parameters related to the VHDL system model and the fault types.
2. Fault injection in the VHDL system model.
3. Analysis of the experiments by comparing the golden run (that is, the simulation trace of the model without faults) with all the faulty simulation traces.

B. Injection targets

Intermittent faults have been injected in the main processor and system buses (data, address and control). In this way, combinational logic and storage components of the arithmetic and control unit have been perturbed. No faults have been injected in the spare processor, because it is the backup unit and it remains inactive while the system is not reconfigured. When the system reconfigures, the spare processor becomes the main processor and then it can be faulty. On the other hand, we have not considered faults in memory because this study focuses on processor faults.

C. Fault models

From the set of intermittent fault models shown in Table I, we have chosen the following models for the experiments:

- **Intermittent stuck-at in storage elements.** We refer to intermittent contacts produced by manufacturing residues and observed in storage cells (registers and memory). This provokes bursts of Single Bit Errors (SBE) [2].
- **Intermittent pulse in buses.**
- **Intermittent {pulse, open, stuck-at, indetermination} in combinational logic.** Intermittent contacts in the input/output (I/O) connections of the combinational logic can manifest also as intermittent stuck-at [7]. These contacts can be provoked by solder joints or aging processes, such as electromigration or delamination.

To select these fault models, we have taken into account:

- A set of intermittent faults observed in real computer systems by means of error logging [2].
- An analysis of representative fault mechanisms related to manufacturing defects and variations, as well as wear-out processes that can provoke intermittent faults [7].
- The characteristics of the VHDL model of the system. For instance, we have not injected time-related faults (such as the Intermittent Delay fault model) due to the lack of temporal specifications in the VHDL model.

As mentioned in Section V.A, all the selected fault models have been injected using the simulator-commands-based fault injection technique.

D. Workloads

To activate the main components of the model, two typical and moderate-duration workloads have been executed: Arithmetic Series and Bubblesort algorithm.

E. Number of faults

1000 faults per experiment have been injected, being classified as single (one burst in a single target) or multiple (simultaneous bursts in different targets). It is interesting to stand out that multiple faults are increasingly important. Due to technology scaling, intermittent faults will likely affect multiple locations. In total, more than 60,000 faults have been injected.

F. Injection instant

For the sake of simplicity, the injection instant has been selected randomly along the workload duration, according to a uniform distribution. In real computer systems, other fault distributions have been observed, such as Exponential or Weibull [3]. For instance, a Weibull distribution with increasing fault rate can be used to emulate a wear out process that increases the frequency of intermittent faults, before eventually becoming permanent. Nevertheless, it can be challenging to set up the distribution parameters, because they depend on system technology and environmental factors such as the temperature.

G. Burst parameters

The burst parameters have been described in Section II. In this way, the number of fault activations, or burst length ($L_{burst}$), has been generated according to a random uniform distribution in the range $[1.0T–10.0T]$. The activity and inactivity times ($t_a$ and $t_i$) have been generated according to a random uniform distribution in the time interval $[0.1T–1.0T]$, where $T$ is the clock cycle (whose value in our experiments is 1µs). This is an intermediate interval between $[0.01T–0.1T]$ and $[1.0T–10.0T]$, used in other works [6]–[8].
H. Analysis of results

For each injection experiment, the simulation traces generated are compared to the golden run. From this comparison, the estimated coverages and latencies of the EDMs and ERMs are automatically calculated. Fig. 5 shows the fault pathology graph, that represents the evolution of faults from their injection to the detection and possible recovery by the FT mechanisms.

From the experimental results, the following measures are obtained:

First, we obtain the percentage of activated errors, defined as

\[ Act = \frac{N_{act}}{N_{inj}} \]  

(1)

where \( N_{inj} \) is the number of injected faults and \( N_{act} \) is the number of activated errors. A fault is called activated when it produces a change on a signal or variable of the model. If it also propagates to signals of the external structural architecture, then it is logged as an activated error.

Another measure is the error detection coverage. We distinguish between two types of coverage estimators:

- The coverage of the detection mechanisms, defined as

\[ C_d(mech) = \frac{N_{det}}{N_{act}} \]  

(2)

where \( N_{det} \) is the number of errors detected by the EDMs. Similarly, we define the detection coverages of each EDM as the errors detected by the individual mechanism (the parity or the watchdog timer):

\[ C_d(Parity) = \frac{N_{det(Parity)}}{N_{act}} \]

\[ C_d(WDT) = \frac{N_{det(WDT)}}{N_{act}} \]

Since \( N_{det} = N_{det(Parity)} + N_{det(WDT)} \), then

\[ C_d(mech) = C_d(Parity) + C_d(WDT) \]  

(3)

- The global system coverage, defined as

\[ C_d(sys) = \frac{N_{det} + N_{non\_effect}}{N_{act}} \]  

(4)

where \( N_{non\_effect} \) is the number of non effective errors, that is, the errors that do not affect the result of the running application. They are overwritten or remain latent in an unused part of the system. \( C_d(sys) \) extends \( C_d(mech) \) by including non effective errors (from (2) and (4)):

\[ C_d(sys) = C_d(mech) + \frac{N_{non\_effect}}{N_{act}} \]  

(5)

Similarly, we calculate the error recovery coverage. Again, we distinguish two types of coverage estimators:

- The coverage of the recovery mechanisms, defined as

\[ C_r(mech) = \frac{N_{det\_rec}}{N_{act}} \]  

(6)

where \( N_{det\_rec} \) is the number of detected errors that are subsequently recovered by the ERMs. As in the case of the EDMs, we also calculate the recovery coverages of each individual ERM (in this case, the back-off cycle, the checkpointing or the spare):

\[ C_r(Boff) = \frac{N_{det\_rec(Boff)}}{N_{act}} \]

\[ C_r(CP) = \frac{N_{det\_rec(CP)}}{N_{act}} \]

\[ C_r(Spare) = \frac{N_{det\_rec(Spare)}}{N_{act}} \]

And the following relationship is accomplished:

\[ C_r(mech) = C_r(Boff) + C_r(CP) + C_r(Spare) \]  

(7)

- The global system coverage, defined as

\[ C_r(sys) = \frac{N_{det\_rec} + N_{non\_effect}}{N_{act}} \]  

(8)

\( C_r(sys) \) extends \( C_r(mech) \) by including non effective errors (from (6) and (8)):

\[ C_r(sys) = C_r(mech) + \frac{N_{non\_effect}}{N_{act}} \]  

(9)

Finally, we calculate the propagation, detection and recovery latencies, defined as:

\[ L_p = t_p - t_{inj} \]  

(10)

\[ L_d = t_d - t_p \]  

(11)

\[ L_r = t_r - t_d \]  

(12)

where \( t_p \) is the time instant when the fault is visible at the signals of the external structural architecture, \( t_{inj} \) is the injection time instant, \( t_d \) is the time instant when the error is detected by the detection mechanisms, and \( t_r \) is the time instant when the recovery mechanisms finish the recovery process.

Similarly to the detection and recovery coverages, we can calculate the latencies of each detection and recovery mechanism: \( L_d(Parity), L_d(WDT), L_r(Boff), L_r(CP) \) and \( L_r(Spare) \).

VI. FAULT-INJECTION EXPERIMENTS RESULTS

Table II and Table III contain the results of the experiment. The first column of Table II shows the percentage of activated errors for single and multiple faults, and for both workloads. It can be observed that multiple faults have much more impact than single faults, with values near 100%. This is an expected result, as multiple faults affect several places at the same time.

This trend is fulfilled in the two workloads, with small differences between them.

Table II also shows the detection coverages (columns \( C_d(mech) \) and \( C_d(sys) \)). We make several observations. First, \( C_d(sys) \) is very high, over 90%. This means that most activated errors are detected or non effective (see Fig. 5). Second, \( C_d(mech) \) is lower, especially in single faults. In multiple faults, \( C_d(mech) \) present values near \( C_d(sys) \). And finally, as expected, detection coverages are higher for multiple faults than for single faults, as multiple faults provoke a higher impact. The percentage of non detected errors that provoke a failure (1 – \( C_d(sys) \)) is very low. Thus, we can conclude that the detection process works quite well. Furthermore, Table II presents the detection coverage of the different EDMs. As it can be seen, \( C_d(Parity) \) is much greater than \( C_d(WDT) \). This indicates that most errors are detected by the parity mechanism.
Last, Table II also contains the average detection latencies, both global and separated by mechanisms. We point out that $L_{d(Parity)}$ is much lower than $L_{d(WDT)}$, because WDT involves the timer overflow, and that the global detection latency is reduced by the influence of the parity mechanism, as it is the most frequently activated EDM.

From the results of detection coverages and latencies, WDT has demonstrated to be a poorly efficient and slow detection mechanism in our system. This suggests that detection can be improved by replacing it with other detection techniques, such as parity in CPU registers and buses, as well as implementing CPU exceptions. In this way, $C_{d(mech)}$ for single faults can also be increased.

Let us now see the recovery coverages and latencies shown in Table III ($C_{r(mech)}$ and $C_{r(sys)}$) for single and multiple faults, and for both workloads. We observe that recovery coverages ($C_r$) are lower than detection coverages ($C_d$), because a fraction of the detected errors cannot be recovered by the recovery mechanisms (see Fig. 5). Most of them provoke failures and a small portion is recovered by the intrinsic recovery mechanisms (see Fig. 5). Most of them provoke failures but non recovered errors (see Fig. 5), mainly for multiple faults. We can conclude that the intermittent faults ($C_{r(sys)}$) includes non effective errors. The difference is more pronounced in single faults than in multiple faults. Finally, we can observe that $C_{r(mech)}$ in multiple faults is greater than $C_{r(mech)}$ in single faults. Although multiple faults are more difficult to recover once they have been detected, they are more frequently detected. For this reason, $C_r$ is bigger. This fact is not accomplished in $C_{r(sys)}$, where we can observe lower values for multiple faults.

In addition, we have verified a non-negligible percentage of failures provoked by detected but non recovered errors (see Fig. 5), mainly for multiple faults. We can conclude that the recovery process does not work as well as the detection process.

On the other hand, Table III shows also the recovery coverage of the different ERMs. From the table, the spare and the backoff (retry) cycle are the most activated mechanisms, while the checkpointing (CP) is much less activated because it is related with the watchdog timer. The activation of the spare is especially high in multiple faults, because these faults are the most harmful and the system interprets them as permanent faults.

Finally, Table III presents the average recovery latencies, both global and separated by mechanisms. Some points can be remarked. First, $L_r$ is much greater than $L_{d(r)}$, because $L_r$ includes longer duration processes. Second, $L_{r(Boff)}$ is smaller than $L_{r(CP)}$ and both are much lower than $L_{r(Spare)}$. CP includes the reading of the checkpoint from stable memory. Spare also adds the reconfiguration time to enable the backup CPU. And finally, $L_r$ for single faults is lower than $L_r$ for multiple faults. Single faults recover quicker than multiple faults because when multiple faults are present, the activation of the spare CPU is more frequent.

From previous results, retry proves to be the ERM that presents the best coverage/latency compromise in our system. Also, its implementation cost is much lower than the spare’s. In fact, it is considered one of the best methods to tolerate intermittent faults [20].

Recovery latency must be a key factor to handle intermittent faults. High frequency intermittent faults may lead to a near coincident fault scenario, i.e. a new fault arrives before the handling of the previous one is completed, leading to the failure of the recovery process. In the following paragraphs, some proposals to reduce $L_r$ are presented.

![Fig. 5. Fault pathology graph [4].](image)

---

### Table II.

**ERROR ACTIVATION, DETECTION COVERAGES AND LATENCIES**

<table>
<thead>
<tr>
<th>Arith. Series</th>
<th>Act (%)</th>
<th>$C_{d(Parity)}$ (%)</th>
<th>$C_{d(mech)}$ (%)</th>
<th>$C_{d(WDT)}$ (%)</th>
<th>$C_{d(Parity)}$ (%)</th>
<th>$C_{d(WDT)}$ (%)</th>
<th>$L_d$ (µs)</th>
<th>$L_{d(Parity)}$ (µs)</th>
<th>$L_{d(WDT)}$ (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>63.8</td>
<td>94.04</td>
<td>48.28</td>
<td>41.54</td>
<td>6.74</td>
<td>16.47</td>
<td>2.89</td>
<td>100.21</td>
<td></td>
</tr>
<tr>
<td>Multiple</td>
<td>99.4</td>
<td>99.70</td>
<td>93.06</td>
<td>77.77</td>
<td>15.29</td>
<td>13.48</td>
<td>1.18</td>
<td>76.04</td>
<td></td>
</tr>
<tr>
<td>Bubblesort</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single</td>
<td>57.5</td>
<td>92.35</td>
<td>36.70</td>
<td>33.22</td>
<td>3.48</td>
<td>10.34</td>
<td>3.21</td>
<td>78.47</td>
<td></td>
</tr>
<tr>
<td>Multiple</td>
<td>98.9</td>
<td>98.28</td>
<td>90.39</td>
<td>75.13</td>
<td>15.26</td>
<td>14.35</td>
<td>1.98</td>
<td>75.23</td>
<td></td>
</tr>
</tbody>
</table>

### Table III.

**RECOVERY COVERAGES AND LATENCIES**

<table>
<thead>
<tr>
<th>Arith. Series</th>
<th>$C_{r(mech)}$ (%)</th>
<th>$C_{r(Parity)}$ (%)</th>
<th>$C_{r(CP)}$ (%)</th>
<th>$C_{r(Spare)}$ (%)</th>
<th>$L_r$ (µs)</th>
<th>$L_{r(Parity)}$ (µs)</th>
<th>$L_{r(CP)}$ (µs)</th>
<th>$L_{r(Spare)}$ (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>87.62</td>
<td>41.85</td>
<td>17.71</td>
<td>2.51</td>
<td>96.26</td>
<td>10.1</td>
<td>37.83</td>
<td>173.59</td>
</tr>
<tr>
<td>Multiple</td>
<td>81.19</td>
<td>74.55</td>
<td>12.47</td>
<td>2.21</td>
<td>132.11</td>
<td>7.14</td>
<td>33.32</td>
<td>161.81</td>
</tr>
<tr>
<td>Bubblesort</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single</td>
<td>89.91</td>
<td>34.26</td>
<td>19.83</td>
<td>0.17</td>
<td>91.82</td>
<td>9.95</td>
<td>0.78</td>
<td>172.2</td>
</tr>
<tr>
<td>Multiple</td>
<td>77.65</td>
<td>69.77</td>
<td>19.21</td>
<td>1.01</td>
<td>119.82</td>
<td>13.63</td>
<td>18.93</td>
<td>163.06</td>
</tr>
</tbody>
</table>
The first one is to increase the threshold that activates the spare CPU. In this way, some intermittent faults can be recovered by the retry or checkpoint techniques, instead of activating the spare CPU. Fig. 6 shows this idea, with different fault types and different threshold (T) values. For instance, T1 and T2 thresholds activate erroneously the spare unit, as the system interprets long transient faults as well as some intermittent faults that disappear as permanent faults. On the other hand, T3 threshold activates the spare unit correctly, recovering thus permanent faults and long intermittent faults that become permanent or do not disappear. The difficulty of this approach is that the programming of the threshold depends on the transient duration and the intermittent burst length, and they are not easy to know a priori. They depend on the hardware technology and the environment. Nevertheless, at least we can estimate the maximum number of successive attempts before activating the spare CPU, that is related with the maximum value of T:

\[ n \leq L_d + L_{(\text{Boff, CP})} \]

Thus

\[ n_{\text{max}} = \frac{L_{(\text{Spare})}}{L_d + L_{(\text{Boff, CP})}} \]

As \( C_{(\text{Boff})} \) is much greater than \( C_{(\text{CP})} \), we can approximate \( L_{(\text{Boff,CP})} \approx L_{(\text{Boff})} \), and hence

\[ n_{\text{max}} \approx \frac{L_{(\text{Spare})}}{L_d + L_{(\text{Boff})}} \]

Another possibility is to implement hot sparing instead of cold sparing. In this way, the time overhead related with reading the CP from stable memory is eliminated, reducing the reconfiguration latency. The disadvantage of this option is the power consumption overhead introduced. For instance, dual systems with hot sparing are typical of self-checking flight computers in aircraft flight control systems [21], where the response time is critical.

It is also feasible to introduce Error Correcting Codes (ECC) into the memory and the critical registers of the CPU in order to improve the recovery from parity detection. Particularly, Flexible Unequal Error Control Codes (FUEC) [22] can be good candidates to cope with intermittent faults, as this type of fault can present unequal error rates in different parts of the data word.

Finally, applying triple modular redundancy (TMR) [3] [23] in selected components of the CPU would allow masking transient and short intermittent faults. Although this technique may introduce a high hardware overhead, it can take advantage of the increasing integration density of current deep sub-micron technologies.

Some authors suggest that hardware implemented error handling techniques are likely to provide the best solutions to mitigate the effects of intermittent faults [2], [13]. The high speed of silicon logic makes hardware implementations well suited for detection and correction of high rate errors. In addition, hybrid solutions, which combine hardware error detection and recovery with software implemented failure prediction and resource reconfiguration, may improve dependability significantly [2].

VII. MARKOV MODELS FOR DEPENDABILITY EVALUATION

In this section, we show Markov models generated for our fault-tolerant system in the presence of intermittent faults, in order to evaluate their dependability. To calculate the transition rates of the Markov chains, we have used the coverage values presented in Table II and Table III. The final objective is to obtain the reliability and safety of the FT system and compare it with other FT approaches (see Fig. 1). Particularly, we compare the original duplex-cold-sparing system with other configurations using warm and hot sparing.

An element with an intermittent fault is usually represented by a two-state Markov model [23], [24]. The two states are a failed state and a pseudo-failed state. In the first state, the fault is active, and using the element produces an incorrect output. In the second state, the fault is in a benign mode, and the output is not corrupted when using this element. An intermittent fault oscillates between these two states with a frequency that depends on the characteristics of the fault.

Taking the previous considerations into account, we have generated the Markov chain shown in Fig. 7 for a duplex system with cold sparing. The intermittent fault oscillation is observed between states 2 and 3, which corresponds to failed (active) and pseudo-failed (benign) states, respectively. The meaning of the different states and transition parameters is summarized in the figure. The expression of transition rates will be explained later in a simplified chain.

![Fig. 6. Configuration of the spare activation in order to reduce the recovery latency.](image-url)
A drawback of this type of model is that the on-off cycles of intermittent faults must be modeled, and the associated parameters have to be measured. It is very difficult to obtain realistic estimation of the rates $\alpha$ and $\beta$. The unpredictable behavior of intermittent faults may also impose an update on the parameters [20].

Another problem is stiffness, which appears in Markov models with transition rates that differ by several orders of magnitude. In our case, $\alpha$ and $\beta$ are usually much higher than the fault rate $\lambda$. Stiffness causes numerical difficulties when solving the differential equations that arise from the Markov chain [26].

To overcome these issues, some authors suggest the elimination of the intermittent cycles and the generation of a Markov chain only with coverages [24], [25], [26]. Although the resulting model is considerably simpler, it can be much more accurate because it relies only on parameters that are directly observable. Fig. 8 shows a simplified version of the chain after removing fast loops to solve the stiffness problem. As mentioned, the coverages have been obtained from the results of the fault injection experiments, analyzed in Section VI.

The activated error rate is calculated as $\lambda_{\text{activ}} = \lambda \times \text{Act}$, where $\text{Act}$ is the percentage of activated errors (see (1)) and $\lambda$ is the intermittent fault rate, that is to say, the arrival rate of intermittent faults.

The coverages of the chain were also defined in Section V.H, and $C_{r(Spare)} = C_{r(mech)} - (C_{r(Bofr)} + C_{r(CP)})$, as pointed in (3). From the expression of the coverages (Section V.H) and the fault pathology graph (see Fig. 5), we can demonstrate the transition rates between states in the Markov model of Fig. 8:

**Transition 1 $\rightarrow$ 2:**

$$\lambda_{\text{activ}} C_{r(Spare)} = \left( \frac{N_{\text{act}}}{u.t.} \right) \times \left( \frac{N_{\text{det,rec(Spare)}}}{N_{\text{act}}} \right) = \frac{N_{\text{det,rec(Spare)}}}{u.t.} \quad (u.t.: \text{unit of time}).$$

So, $\lambda_{\text{activ}} C_{r(Spare)}$ is the rate of activated errors that are detected and recovered by the spare mechanism.

![Fig. 7. Markov model for a duplex system with cold sparing and intermittent faults.](image-url)
From the results obtained in Section VI, and taking into account the variations due to the fault multiplicity (single/multiple faults), as well as the workload, for the Markov chain in Fig. 8 we have selected the coverage values shown in Table IV.

<table>
<thead>
<tr>
<th>Coverage Value Selection for the Markov Chain in Fig. 8</th>
<th>Sample values</th>
<th>Typical value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_d(mech)</td>
<td>0.6, 0.7, 0.8, 0.9</td>
<td>0.8</td>
</tr>
<tr>
<td>C_d(spare)</td>
<td>0.92, 0.94, 0.96, 0.98, 1.00</td>
<td>0.96</td>
</tr>
<tr>
<td>C_d(mech)</td>
<td>0.6, 0.65, 0.7, 0.75, 0.8</td>
<td>0.7</td>
</tr>
<tr>
<td>C_d(spare)</td>
<td>0.4, 0.5, 0.6</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Some comments about the coverage values should be made. On the one hand, all coverages are normalized with respect to the number of activated errors (N_acts). This fact explains that C_d > C_r >> C_s(spare). It can seem that 0.4 is an unacceptably low value for C_r(spare), but remember that C_r = C_r(Boff) + C_r(CP) + C_r(spare) (see (7)). So C_r(spare) is a portion of C_r(mech). It represents the fraction of all activated errors that are detected and recovered by one of the ERMs, the spare. Table III shows the values of C_r(spare).

On the other hand, in the selection of typical values, we have chosen a representative “average” value from the variations of the fault multiplicity and the workload. We have taken into account the increasing trend of the probability of occurrence of multiple faults as feature size shrinks. This consideration makes the coverage values grow slightly.

For the intermittent fault rate \( \lambda \), we have used a typical value of \( 10^{-4} \) [25]. From Table II, we have selected an “average” value for the percentage of activated errors \( Act = 0.8 \). So, \( \lambda_{activ} = \lambda \times Act = 10^{-4} \times 0.8 \).

Next, some results obtained by solving the Markov model with WinSURE are shown. System reliability and safety have been calculated. Next we deduce the expressions for reliability (R) and safety (S) from Fig. 8.

The reliability is defined as the probability that the system works correctly at a given time [3]. Thus:

\[
R = P_1 + P_2
\]

The safety is defined as the probability that the system works correctly, or fails in a safe-controlled way. Then:

\[
S = P_1 + P_2 + P_{FS} = R + P_{FS}
\]

Fig. 9 shows the reliability and the safety as a function of time. We observe that:

- \( S \) is much greater than \( R \), and the difference grows with time. Remember that \( S = R + P_{FS} \).
- For low time values, \( R \) and \( S \) present acceptable values, over 0.99.
- However, \( R \) degrades excessively \( (R < 0.6) \) from about 26000 hours (almost 3 years). This shows the necessity of improving the reliability of the system for long time runs.

Table V shows the detailed values of \( R \) and \( S \) for some representative time values. Acceptable results are observed for 10h and 1 week. On the other hand, a large degradation of \( R \) is observed for high time values, with values under 0.5.

We have also analyzed how the variation of the coverages affects \( R \) and \( S \), in order to improve system dependability (see Fig.10). We have found that \( R \) is sensitive to both \( C_r(mech) \) and \( C_d(sys) \) with similar slopes. On the other hand, \( C_d(sys) \) is the coverage that most improves \( S \). Small increases in the detection coverage yield a significant variation of \( S \).

As \( C_d(sys) = C_d(mech) + \frac{N_{non_effect}}{N_{activ}} \), (see (5)), it is necessary to augment \( C_d(mech) \) (by improving the fault detection mechanisms) to enhance \( C_d(sys) \).

Let us suppose that the cold sparing system is included in a...
long-life unmanned spacecraft (such as a satellite or a deep-space planetary probe). Space probes typically use duplex systems with cold standby sparing in the Attitude Control Subsystem, and duplex systems with hot standby sparing in the Command and Control Subsystem [28]. In this case, it is mandatory to improve R. On the other hand, S is not a critical metric since no human life is involved. So effort should be focused on improving $C_{r(mech)}$ by the optimization of the EDMs. Some proposals have been commented in Section VI when analyzing the coverages results.

On the other hand, suppose that the cold sparing system is included in the flight control system of a space shuttle (with mission time about 1 week) or a civil aircraft (with flight time about 10h). They are safety critical systems with human life involved. Besides having a high reliability, special care with safety must be taken into account. In these cases, EDMs should be enhanced in order to improve $C_{d(sys)}$.

To sum up, the generic problem of dependability assessment can be formulated in this way: Given an application (system) with operation (life) time $t$, which values of $C_d/C_r$ are needed in order to achieve required levels for R/S? This can determine whether the EDMs/ERMs are effective enough or if they must be improved instead. In the last case, a feedback process is applied, as shown in Fig. 1.

Finally, we will compare the duplex cold sparing with other fault-tolerant reconfiguration approaches, such as warm and hot sparing. In Section VI we have commented that $L_r$ is a key possibility of near coincident scenarios. We have said also that a possible solution to reduce $L_r$ is to introduce sparing variants such as hot or warm sparing. Hot sparing is a typical configuration on critical systems where recovery must be done as soon as possible [28]. Let us see how R and S are affected by these variants. These techniques can be good alternatives to improve the tolerance against intermittent faults if R and S stay at acceptable values.

Fig. 11 shows the Markov model for warm sparing, including the explanation of states and transition parameters. This model has been generated taking as reference the cold sparing model in Fig. 8, adding the state 3 and some transitions. The key difference is the fault rate of the primary ($\lambda_p$) and the backup ($\lambda_b$) processors. In cold sparing, $\lambda_b = 0$, because the backup unit is inactive. $\lambda_b < \lambda_p$ in warm sparing, due to the fact that the backup unit is active, but less than the primary unit. In hot sparing, $\lambda_b = \lambda_p$, because both units run the same tasks simultaneously. Hence, the model is also valid for hot sparing, making $\lambda_b = \lambda_p$.

Different activated error rates are defined for both the primary (main) and the backup (spare) processor:

\[
\lambda_{p актив} = Act_p \times \lambda_p \\
\lambda_{b актив} = Act_b \times \lambda_b
\]

where $Act_p$ and $Act_b$ are the percentage of activated errors for the primary and the backup processors, respectively. They can be obtained by applying fault injection, as it has been done for $Act$.

The transition rates between states can be deduced from the expression of the coverages, as it was done for the cold sparing model.

**Transition 1 → 3:**

$\lambda_{p актив}(1-C_{d(sys)})$ represents the rate of activated errors in the backup unit that are non-detected and produce a failure in this unit.

Fig. 10. Sensitivity of reliability and safety to coverages. Duplex with cold sparing. Time = 168h.

Fig. 11. Markov model for a duplex system with warm sparing and intermittent faults.
Transition 3 → F:
\[ \lambda_{pactiv} \left[ (1 - C_{d(sys)}) + C_{r(Spare)} \right] \] has two contributions:
- \( (1 - C_{d(sys)}) \), the activated errors in the primary unit that are non-detected and provoke a failure.
- \( \lambda_{pactiv} C_{r(Spare)} \), the activated errors in the primary unit that would be detected and recovered by the spare mechanism. In this case, the system tries to reconfigure with the backup unit, but this unit is failed.

Transition 3 → FS:
\[ \lambda_{pactiv}(C_{d(mech)} - C_{r(mech)}) \] is the rate of activated errors in the primary unit that are detected but non recovered. This provokes a safe failure, because the errors are detected.

Transition 1 → 2:
It has two contributions:
- \( \lambda_{pactiv} C_{r(Spare)} \), that has the same meaning as in the cold system. It indicates the rate of activated errors that are detected and recovered by the spare mechanism.
- \( \lambda_{pactiv}(C_{d(mech)} - C_{r(mech)}) \), that indicates the activated errors in the backup unit that are detected but non recovered. In this case, they produce a failure in the backup unit while the primary unit is correct.

The remaining transitions coincide with the cold sparing model, using \( \lambda_{pactiv} \). This model is also valid for transient and permanent faults, by simply changing the corresponding coverage values and the activated error rates.

To solve the model, the following parameters values have been applied:
- \( \lambda_{pactiv} = Act_p \times \lambda_p = 0.8 \times 10^{-4} \)
- \( \lambda_{pactiv} = Act_b \times \lambda_b = 0.5 \times 10^{-5} \)
Notice that \( \lambda_b < \lambda_p \) and \( Act_p < Act_b \).

According to the system to analyze, the values of the coverages can be different. Warm and hot sparing usually include additional EDMs, such as monitoring between the processors as well as comparison in hot sparing. In this case, the detection coverages increase. We have assumed that:
- \( C_{d(mech)} \) hot > \( C_{d(mech)} \) warm > \( C_{d(mech)} \) cold
- \( C_{d(sys)} \) hot > \( C_{d(sys)} \) warm > \( C_{d(sys)} \) cold

Table VI summarizes the values of the coverages used for the three systems.

<table>
<thead>
<tr>
<th>Coverages</th>
<th>hot</th>
<th>warm</th>
<th>cold</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_{d(mech)} )</td>
<td>0 95</td>
<td>0 85</td>
<td>0 8</td>
</tr>
<tr>
<td>( C_{r(mech)} )</td>
<td>0 7</td>
<td>0 7</td>
<td>0 7</td>
</tr>
<tr>
<td>( C_{d(sys)} )</td>
<td>0 99</td>
<td>0 97</td>
<td>0 96</td>
</tr>
<tr>
<td>( C_{r(sys)} )</td>
<td>0 4</td>
<td>0 4</td>
<td>0 4</td>
</tr>
</tbody>
</table>

Fig. 12 shows the reliability (R) as a function of time, for the three types of sparing. We observe:
- An asymptotic exponential decrease, with \( R(\infty) \rightarrow 0 \), in the three systems.
- \( R_{cold} > R_{warm} > R_{hot} \). This is an expected result, since the fault probability is bigger when the spare unit is active (hot) or pseudo-active (warm).

Differences increase with time. Prior to 4500 hours (about 6 months), \( R > 0.9 \) with small differences between the three systems. After about 18000 hours (about 2 years), \( R_{hot} \) degrades below 0.6.

Fig. 13 shows the safety (S) as a function of time, for the three types of sparing. We notice:
- An asymptotic exponential decrease, but with \( S(\infty) \rightarrow C_{d(mech)} \). The same trend has been observed for cold, warm and hot sparing. S values are much greater than R values. Values near or higher than 0.9 are observed for all mission times.
- \( S_{hot} > S_{warm} > S_{cold} \). This is because the detection coverage is bigger in hot and warm.
- Differences grow with time, although they are not high.

From these results, we can conclude that for short-medium times hot (or warm) sparing may be acceptable solutions to reduce \( L_n \), because R does not degenerate excessively. In addition, S shows better values than in cold sparing. For long times, R degrades too much. It should be necessary to improve the ERMs (\( C_{r(mech)} \)), as stated previously for cold sparing.
It is important to notice that, independently of the particular results obtained for reliability and safety, the methodology presented is general, and it can be applied to any fault-tolerant system at an early phase of the design.

VIII. CONCLUSIONS

In this work, we have presented a study of the dependability assessment of a fault-tolerant system against intermittent faults. This study is motivated by the increasing incidence of intermittent faults in deep submicron technologies. The fault-tolerant microcomputer system studied is duplex with cold sparing. This redundancy technique is often used in both long-life unmanned spacecraft systems and high-availability transactional processing systems. We have applied VHDL-based fault injection due to its flexibility, as well as the high observability and controllability of all the modeled components.

Detection and recovery coverages and latencies have been calculated in order to assess the fault-tolerance mechanisms. Among the applied detection/recovery mechanisms, parity and retry have shown the best coverage-latency compromise. Finally, some Markov models have been generated to evaluate the dependability of the fault-tolerant system. Coverages have been introduced in the Markov model and dependability attributes (reliability and safety) have been calculated. Results have been compared with warm and hot sparing. From the results obtained, and considering the reconfiguration latency as a key factor to cope with intermittent faults, it is suggested to use hot sparing for short-medium operation times. The Markov models generated are flexible and general, and they can be applied also to manage transient and permanent faults with no modification of the chain; instead, only coverage values must be changed.

The methodology presented is general, and it can be applied to any fault-tolerant system at an early phase of the design. An iterative improvement of the detection and recovery mechanisms can be achieved, following a feedback process.

REFERENCES


Daniel Gil-Tomás is an associate professor at the Department of Computer Engineering (DISCA) of the Universitat Politècnica de València (UPV). He is also affiliated to the Fault-Tolerant Systems Research Line (STF-ITACA). His research interests include Design and Validation of Fault-Tolerant Systems, Reliability Physics and Reliability of Emerging Nanotechnologies.

Joaquín Gracia-Morán is a senior lecturer at the DISCA-UPV. He is also a member with the STF-ITACA. His research interests include Design and Implementation of Digital Systems, Design and Validation of Fault-Tolerant Systems and Fault Injection.

J.-Carlos Baraza-Calvo is an associate professor at the DISCA-UPV. He is also affiliated to the STF-ITACA. His research interests include Design and Implementation of Digital Systems, Design and Validation of Fault-Tolerant Systems and Fault Injection.

Luis-J. Saiz-Adalid is a Ph.D. candidate and a lecturer at the DISCA-UPV, and he is affiliated to the STF-ITACA. His research interests include Design and Implementation of Digital Systems, Design and Validation of Fault-Tolerant Systems and Fault Injection.

Pedro-J. Gil-Vicente (M’93) is professor at the DISCA-UPV, and co-director of the STF-ITACA. His research focuses on the Design and Validation of Real-Time Fault-Tolerant Distributed Systems, Dependability Validation using Fault Injection, Design and Verification of Embedded Systems, and Dependability and Security Benchmarking. He has authored more than 100 research papers on these subjects.