11

Validating a Safety Critical Railway Application Using Fault Injection

Ivano Irrera[1], András Zentai[2], João Carlos Cunha[1,3] and Henrique Madeira[1]

[1]CISUC, Department of Informatics Engineering, University of Coimbra, Portugal
[2]Prolan Process Control Co., Szentendrei út 1-3, H-2011 Budakalász, Hungary
[3]ISEC – Coimbra Institute of Engineering, Polytechnic Institute of Coimbra, Portugal

The need for safety assurance in critical systems demand for new tools and techniques which are able to provide the required confidence while maintaining the costs relatively at a low level. Fault Injection (FI) is a technique extensively used in several domains, such as space, but sporadically used in the railways. In this chapter, we present a fault-injection tool able to complement the traditional verification and validation procedures, to validate the safety of ProSigma, a Safety Integrity Level (SIL) 4 safety-critical system for railway signaling, implementing a Triple Modular Redundancy (TMR) architecture. This tool is based on the Joint Test Action Group (JTAG) technology, and allows emulating the effects of hardware faults. Results from the FI campaigns show the ProSigma system exhibiting a high degree of tolerance to most of the injected faults, and unexpected behavior in some cases. The results also confirm the efficacy of the proposed technique to help understand worst-case scenarios for validating safety of such a critical system.

11.1 Introduction

The products in all technical and societal domains are required to be certified against hidden design and implementation defects that may induce malfunctioning, which may cause critical damage to the system itself, including

the environment and humans. Several accidents caused by malfunctioning systems are sadly known, going along with the human technological rise.

A safe system is a system that will not cause harm to its users and the environment, in case a malfunctioning occurs. *Safety*, thus, comes to be an attribute of systems, which corresponds to guarantee the absence of catastrophic consequences on the user(s) and the environment to a certain extent. A system whose malfunctioning is likely to cause harm to users or the environment is named a *safety-critical system*.

Since their first realization, railway system fall into the class of safety-critical systems. For assuring the safety of such systems, best practices and standards have been proposed and used along with their technological evolution. In the last decades, a new series of standards have been proposed, namely the CENELEC standards (e.g., EN 50126, EN 50128, EN 50129, EN 50159), for regulating the development and safety assessment of software and hardware. In particular, the EN 50128 describes methods to be used in order to provide software that meet the demands for safety integrity. Although not directly referring fault-injection as a possible technique for verification and validation (V&V) processes, this is an approach extensively used in other domains, such as space.

Fault injection (FI) consists of the deliberate insertion of faults (i.e., realistic perturbations) in computer systems components in order to evaluate the dependability and safety properties of systems or to validate specific fault handling mechanisms. As typically FI tools perform FI campaigns with minimal user intervention (ideally, the process is fully automatic), it is possible to perform very large number of experiments (very often thousands or even millions), which makes FI a valuable method to anticipate worst-case scenarios or rare failure modes that are very hard to anticipate using analytical modeling or simulations techniques.

In this chapter, we present a FI tool and report some preliminary results meant to validate a TMR system for railway signaling. In the field of railway interlocking systems copper-based, long-distance connections exists between relay switches and remote equipment. In the case of constructing a new system, such as the one reported in this paper, the state-of-the-art solution is to apply Internet Protocol (IP) based signal transmission using Global System for Mobile communications (GSM) or fiber-optic communication. These new technologies pose significant safety challenges, which constitutes a relevant scenario for using FI.

The rest of the chapter is organized as followed: Section 11.2 presents background on V&V processes, certification and standards of railway systems, and FI; Section 11.3 presents the ProSigma system railway signaling

system; the proposed FI tool-based on On-Chip Debugging (OCD) technology is described in Section 11.4. The experimental application of the proposed FI tool for the validation of the safety of the ProSigma signaling system is presented in Section 11.5, where results obtained are discussed. Section 11.6 concludes the chapter.

11.2 Fault Injection for V&V and Certification

Several approaches have been proposed to assess and guarantee the correct functioning of a given product. Among these systems V&V are activities that allow verifying whether a product meets its own requirements (Verification), and that the product does what is expected to be done (Validation). However, the application of V&V is challenging, as the definition of methods, strategies, and tools for verifying and validating a system adequately, while simultaneously keeping the cost and delivery time reasonably low, is inherently complex. Companies, in fact, are often, on one hand, pushed towards meeting predefined quality goals, and on the other hand, required to deliver systems at acceptable cost and time to market. It is not rare to find companies following a brute-force approach, by focusing large volume investments into tooling and in-house training, especially when coming down to the development of mission- and safety-critical systems.

Validation and verification are time-consuming activities in traditional software engineering even for non-critical applications. In the case of safety-critical systems, which are often embedded, the complexity of V&V and certification procedures are exacerbated by the need of keeping properties such as safety or availability, and by involving custom and Commercial Off-The-Shelf (COTS) hardware elements and application dependent-interfaces, resulting in an extremely large number of potential factors.

Safety-critical systems also required, over time, the creation of a field of study particularly aimed at focusing on safety-related issues: safety engineering. Safety engineering is a well-established field, including several techniques for the assurance and assessment of safety in a system. Among these, Failure Modes and Effects Analysis (FMEA), Preliminary Hazard Analysis (PHA), and Fault-Tree Analysis (FTA) are some of the most used techniques. In particular, FMEA is a technique that aims at collecting the known system's failure modes, and studying its propagation paths through the system and its effects. Failure Modes, Effects and Criticality Analysis (FMECA) is a version of FMEA in which criticality is taken into account, aiming to identify all critical and catastrophic subsystem or system failure modes.

FMEA is performed mainly manually, even though several works for an automated FMEA have been proposed [1].

Such techniques are expected to be part of the V&V process of a safety-critical system, and even be mandatory. In this direction, standards started to rise with the aim of reducing risks related to the use of safety-critical systems.

11.2.1 Standards for Safety-critical Railway Applications

Standards have been proposed for developing safety-critical systems, both general and domain specific and suggesting strategies, processes, and techniques to adopt along the entire development cycle.

The specification, design and validation of dependability-related aspects concerning *railway* applications are regulated by the CENELEC standards. The most important European standard concerning robustness in this field is the standard EN 50128:2011 – Railway applications – Communication, signaling, and processing systems – Software for railway control and protection systems (EN 50128) [2]. The EN 50128 gives indication about the lifecycle that has to be followed, the techniques and measures to be applied, the necessary competences, and the expected documents and their content. The Software Requirements Specification shall express the required properties of the software being developed. These properties, which are all (except safety) defined in ISO/IEC 9126 series, shall include (among others) robustness and maintainability. The Software Verification Plan shall address (among other properties) the evaluation of the safety and robustness requirements (defined in the Software Requirements Specification). Several techniques and methodologies are also indicated for ensuring the software robustness properties, as Software Error Effect Analysis (i.e., SW-FMEA).

Furthermore, the EN 50128 concentrates on methods that need to be used in order to provide software that meet the demands for safety integrity. The EN 50128 defines robustness as the "*ability of an item to detect and handle abnormal situations*". The most important of software techniques to assess and increase the robustness are the following: Defensive Programming, Information Encapsulation, Fault Detection and Diagnosis, Error Detecting and Correcting Codes, Diverse Programming, Software Error Effect Analysis, Control Flow Analysis, Common Cause Failure Analysis, FI, Boundary Value Analysis, and Coding Standard.

Generally, Railway Safety Cases shall provide evidences that the consideration of Robustness (error cases, abnormal inputs, etc.) is provided together with the system validation and verification.

According to the standard CENELEC EN 50129:2003 [15], safety-related software has been classified into five safety integrity levels, where 0 is the lowest and 4 is the highest. To be conforming to SIL 4 requirements, the safety availability of the equipment must be over 99.999%. From the safety functionality point of view (CENELEC EN 50129:2003 [15]), SIL is a number that indicates the required degree of confidence that a system will meet its specified safety functions with respect to systematic failures. From the software point of view, CENELEC EN 50128:2011 [14] defines software safety integrity level as a classification number that determines the techniques and measures that have to be applied to software.

11.2.2 Fault Injection

Fault injection is a technique consisting in deliberately injecting faults (e.g., bombarding devices with radiations) or modifying parts of the system in a way that emulates the presence of such faults.

Fault injection has been used extensively in research and also already recommended by several standards, such as space [3] and automotive [4] industry standards, in addition to Information and Communication Technology (ICT) industry in general [5]. The space industry, in particular, has a long tradition of using FI as part of the V&V activities, namely to simulate the effects of cosmic radiation in on-board systems. As mere examples, here are some references for the interested reader [6–8]. There are also some examples of the use of FI in the railway industry [9, 10].

Faults are the hypothesized cause of an error (an unexpected internal state of a system) that can lead to a system failure (e.g., crash, performance degradation, or any interruption of the service provided by the system) [11]. Hardware faults, such as bit-flip and stuck-at, occur in hardware components, while software faults are defects in a piece of software that exist due to some issue during the development phase, such as a missing system specification or poor testing. FI consists of deliberately inserting faults into a system in a way that emulates real faults [12]. It is a well-known approach used in many works, where the observation of systems in the presence of faults is needed, such as for fault tolerance and dependability validation [13, 14], estimation of fault-tolerance parameters [12], and benchmarking [15].

The type of faults injected typically fall into three kinds: hardware faults (e.g., bit flips), software faults (i.e., bugs), or input corruption at component interface level (often named as robustness testing). Although the initial FI tools are used to hardware approaches to inject (hardware) faults, including pin-level, heavy-ion radiation, and electromagnetic disturbances, modern FI

tools use software approaches to inject the faults (actually, faults are emulated by software by mimicking the fault effects through the injection of errors). As modern FI tools use software to inject/emulate the faults, a key issue is the precision of the fault models. That is, the injected faults should be representative of the real faults that affect systems in the field. This is not a problem for the hardware faults, as the classic bit-flip or bit stuck-at models (at the processor register or memory level) are widely accepted, but the injection of realistic software faults (i.e., bugs) is far more complex. In software FI, the goal is to inject software faults (bugs) in a given software component to emulate the erroneous behavior that may result from the activation of residual bugs that may exist in that component. In this way it is possible to evaluate whether the system can cope up with the failures in the target software component or not, or to perform an experimental estimation of the risk of (re-) using software components.

An example of a survey of the earlier FI methods can be found in [16] and a very recent and extensive survey (57 pages) covering software FI is in [17], where the issue of defining realistic software fault models is explained in detail.

11.3 The ProSigma Safety-critical Railway Interlocking System

ProSigma [18] is a versatile Hardware–Software (HW–SW) system designed primarily for railway trackside signaling and communication purposes. It is a Safety Signal Transmitter (SST), which provides fail-safe signal transmission with high availability. It captures the analog signal outputs of the railway interlocking system, processes and transmits this information to a remote control center (DaKo). The ProSigma system is designed to be SIL 4 certified according to CENELEC EN 50126-1, 50126-2, 50128, and 50129 standards [12–15].

In case of disconnection or system failure, the outputs move into a safety position. The system is built from modular cards installed in racks, which enables system designers to scale the system according to the application needs.

11.3.1 Concepts of Generic Product, Generic Application and Specific Application

To ease the certification process, the system software is designed to have a three-layered architecture as it can be seen in Figure 11.1.

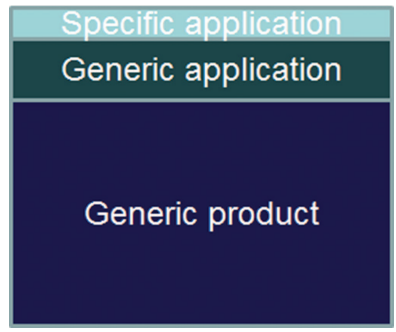

Figure 11.1 The ProSigma abstraction layers.

The bottom layer, called Generic Product (GP), implements the common functionalities of the system, including time synchronization, handling Controller Area Network (CAN) communication and other HW interfaces. The GP is quite complex, but it has to be certified only once, as it is common to all applications.

The middle layer, called Generic Application (GA), is a lightweight software component running on the top of the GP. Each GA handles one railway object (e.g. railway traffic signal, switch, etc.). Because of the simplicity of the code, the certification process of GA is relatively easy.

In the deployment phase of the system the GAs has to be parameterized with the actual values of the specific environment (e.g., voltage comparator thresholds, sampling frequency, etc.), which result in Specific Applications (SAs), which are the top layer of the software architecture. In the ProSigma system, the Logic and Input (LI) cards implement these three-layers design architecture.

11.3.2 The System Architecture and Functionality

A ProSigma test system was built in a pilot project to assess the benefits and drawbacks of FI, whose experimental results are presented in this chapter. The system has identical functionality but limited number of components compared to the one which is deployed trackside. The system adds a networking layer on top of a conventional relay based interlocking system. This network layer transmits the railway object states – represented by the relay outputs – to a remote control center (DaKo). The system architecture (Figure 11.2) consists of the following components:

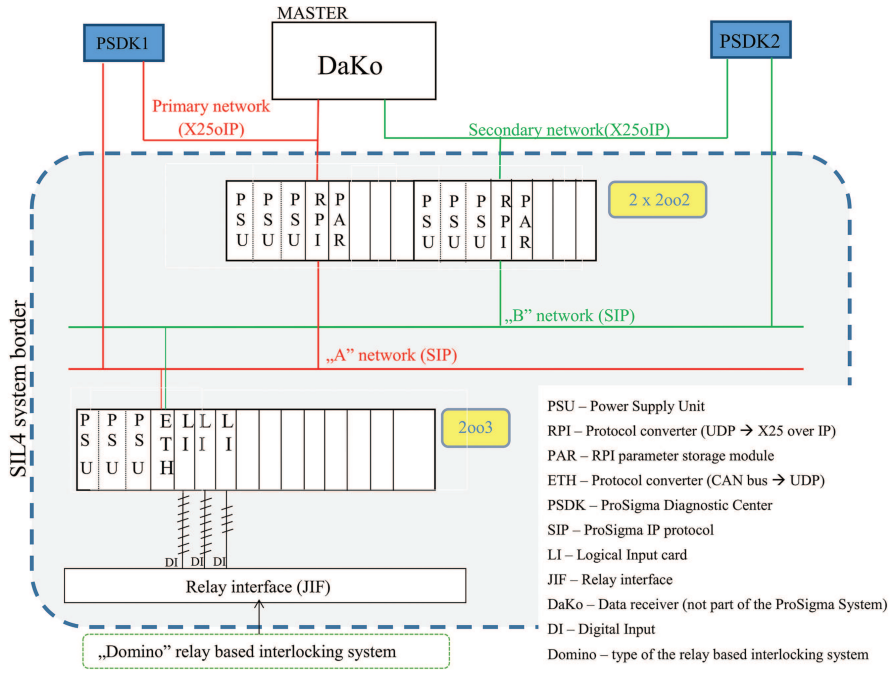

Figure 11.2 System architecture.

- Power Supply Unit (PSU), which supply 3.3 and 24 V of DC voltage to the cards;
- An analog signal conditioning unit (JIF) which filters and down-scales the relay output voltages from 0–48 V to 0–3 V range.
- Logic and Input card (LI) which are sampling the input voltages. They also contain the railway logic.
- CAN to UDP protocol converter cards (ETH), which convert CAN messages to UDP packets.
- UDP to X25 over IP protocol converter cards (RPI), which convert UDP packets to X25 over IP telegrams.
- Two diagnostic centers, which are responsible to log status and communication information and to provide diagnostic data to the operator.

11.3.2.1 Logic and Input (LI) card

The input signals of the system are the analog output voltage signals of a relay-based railway interlocking system: "Domino 70". These voltage signals are passed through a relay interface unit (JIF), which performs the voltage

level interfacing for the Digital Inputs (DI) of the LI card. The Logic and Input (LI) card is a TMR system composed of three microcontrollers from different controller families. Red, Green and Blue (R, G, B) are the codenames for the three channels.

The Logic and Input (LI) cards are reading the analogous input signals and interpret them according to the rules of the specific railway object, which they are connected to. Finally LI cards transmit the status of the railway object states via CAN bus. The three channels (R, G, B) communicate on separate CAN buses, which are located on the back-panel of the mounting rack. See Figure 11.3 for the LI card.

The firmware (FW) of the controllers has been developed by different SW teams to avoid common mode faults. LI cards follow the three layered SW architecture described before consisting of two different FWs: GP and GA are parameters for the SA. On each channel, the FW of GP and GA are running on the microcontroller in a time and space partitioning architecture. On all channels, FW of the GPs handle the A/D conversion of the input signals. The raw data of the converted signals are filtered with a SW implemented debouncing algorithm in the GP to filter out the high frequency glitches of the relays. The GP FW calls the GA FW every 32 ms and the de-bounced values of the input signals are passed to the GA. The GA implements the railway object.

The railway object used in this case study is called block direction, which contains the information of the direction of traffic on the actual railway

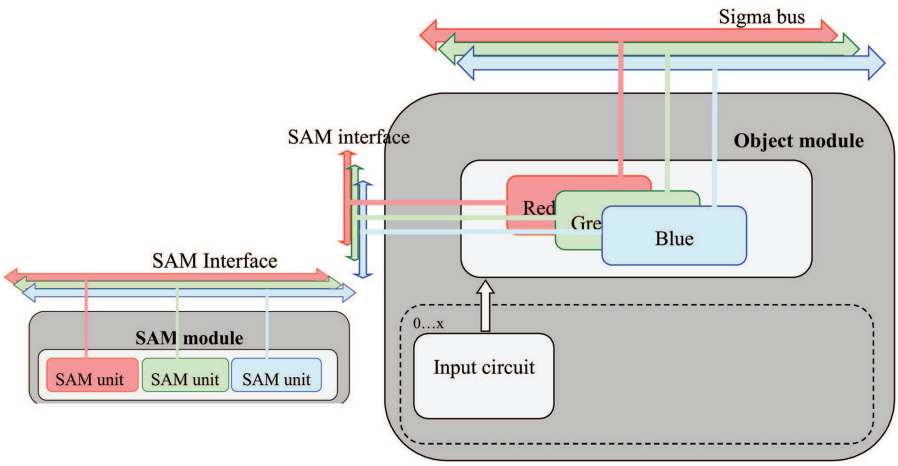

Figure 11.3 LI card interfaces.

Table 11.1 Railway object outputs

Valent Input	Antivalent Input	Meaning in Case P1 = 0	Meaning in Case P1 = 1
0	0	Transient (0 × 20) or invalid (0 × 80) state	Transient (0 × 20) or invalid (0 × 80) state
0	1	Direction = Exit (0 × 02)	Direction = Entry (0 × 01)
1	0	Direction = Entry (0 × 01)	Direction = Exit (0 × 02)
1	1	Transient (0 × 20) or invalid (0 × 80) state	Transient (0 × 20) or invalid (0 × 80) state

segment. The object has one input encoded by a pair of valent-antivalent input signals. Depending on the input signals and the value of parameter P1 the meaning of the direction could be entry, exit, transient or invalid as it is described in Table 11.1. The valent-antivalent signal pair does not change simultaneously so for a short period of time invalid input patterns (00 or 11) are accepted as transients. After that time period is passed, the signals became invalid.

The interpreted railway object state, encoded in the hexadecimal numbers indicated in Table 11.1, is transmitted on the CAN bus. The Sigma bus in Figure 11.3 indicates a proprietary application layer protocol implemented on top of the CAN bus. Specific Application Module (SAM) contains the parameters for the Generic Application. In the SAM module, 3 Flash memory chips contain the parameters for the three channels R, G, B. The LI card reads the parameter values from the memory via Serial Peripheral Interface (SPI) bus.

Interfaces of a LI card can be seen in Figure 11.3.

Up to 10 railway object modules could be inserted in one rack. In case there are more than 10 railway objects in a system, then the extra object modules are inserted into multiple racks. The racks are connected together to form a Local Area Network (LAN) using ETH cards.

11.3.2.2 ETH card

Primary function of the CAN to UDP protocol converter (ETH) card is to collect the railway object state information of the three channels from the CAN bus and transmit these messages as UDP datagrams on the Ethernet network. As it can be seen from Figure 11.4, ETH cards are connected to the CAN buses of all the three channels of the LI cards. This connection is physically realized through the back panel of the modular racks. Each ETH card contains two identical HWs. The inputs from both HWs are the

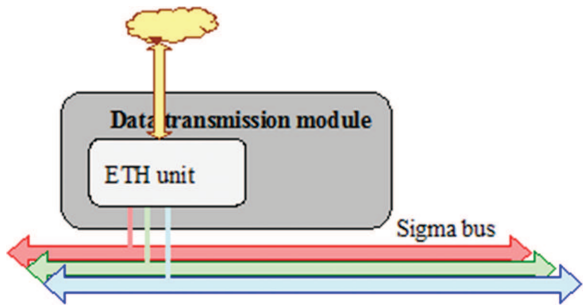

Figure 11.4 ETH card architecture.

same CAN channels, while the outputs are connected to two distinct LAN networks.

11.3.2.3 RPI card

The UDP messages are transmitted to the UDP to X25 over IP protocol converter unit (RPI). RPI architecture is depicted in Figure 11.5. This unit is responsible for converting the UDP packages to X25 over IP telegrams and sending these to the data receiver (DaKo), which is not part of the system.

The RPI module also performs a voting on the data collected from the three channels (extracted from the UDP packets), thus being central to the correct functioning of the TMR schema. Moreover, it provides two times 2-out-of-2 fault tolerance schema applied to both received data and voting result: the information is analyzed from two separated nodes (here named node 0 and node 1), and differences among data cause the entire RPI node to fail. Each node has a 2-out-of-2 architecture.

The underlying hardware of the UDP to X25 over IP protocol converter card/RBC-Prolan Interface (RPI) card is identical to the ETH card.

The functionality of the RPI card includes:

- Managing X25 connection with the Radio Block Center;
- Voting about the object states;
- Transmitting object states to the RBC;
- Exchanging Heartbeat (HB) signals both on active and on the potentially active channel.

11.3.2.4 Power Supply Units

In each rack, three Power SUpply (PSU) cards provide the necessary energy for the operation of the system.

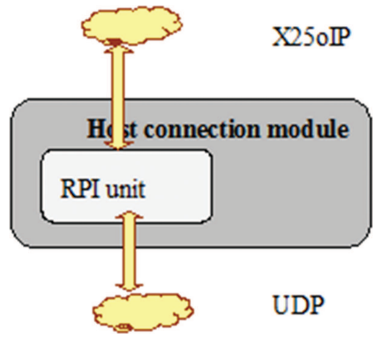

Figure 11.5 RPI card architecture.

11.3.2.5 Diagnostic centers

Two diagnostic centres (PSDK1 and PSDK2) are monitoring and logging the traffic on the internal and external networks.

11.3.2.6 Parameter modules

The parameter modules (PAR) contain the parameters of the GP and GA, which are required for the operation of the system.

11.3.3 System's Critical Aspects Worth to Study Using FI

Considering the block direction railway object, a dangerous situation occurs when the DaKo system's block direction information is the opposite direction than the actual block direction. This situation could occur when an opposite block direction information is sent to the DaKo or when the block direction changes but the system does not transmit this information to the DaKo. Thus the critical parts of the system are the input processing parts of the LI cards and the voting part of the RPIs. These are the parts where fault-injection should be applied to assess the system's robustness.

11.4 The ProSigma FI Framework

Hardware and software failures may both occur with non-negligible probability, especially in a complex safety-critical system operating in harsh environments, and both types have a potentially huge impact on the system and on the application (railway signaling in the ProSigma case). As presented, the FI technique aims at emulating situations in which the system and its fault tolerance mechanisms face the activation of hardware and software faults,

and, at the same time, collecting information on the fault activation, errors and failures caused.

The proposed FI framework has been designed to inject hardware and software faults, taking advantage of on-board scan-chain circuitry (or OCD), to emulate faults with controlled intrusiveness. The proposed framework also provides the infrastructure for collecting the experiment results automatically, allowing posterior validation of system safety requirements. In particular, the FI framework is based on the JTAG scan-chain circuitry, a *de-facto* standard implemented on a large variety of microcontrollers, including those used in safety-critical scenarios. The JTAG allows, for instance, reading values in RAM without interrupting the controller execution, and writing values in local controller registries. These are key features to both inject the faults and collect direct impact at CPU level. As an example, a bit-flip fault is injected by stopping the controller execution, reading the value of a CPU register, changing the value of a given bit (or bits), and writing back the new value to the register. The intrusiveness of such injection operation is just a few operation cycles.

11.4.1 Fault Injector Framework Architecture and Functionalities

The architecture of the FI framework, shown in Figure 11.6, is made up of several components, distributed on a host system and on the target system, namely:

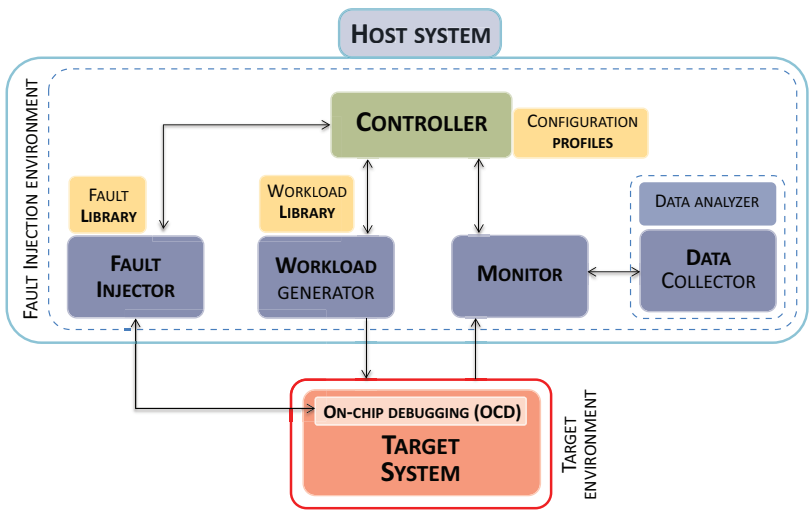

Figure 11.6 Fault injection structure and environment.

- the **fault injector component**, executing a set of instructions directly on the target system, using the OCD interface of the target system. The fault to be injected are defined in a specific module of the injector, called fault library;
- the **workload generator**, controlling the inputs to the target system. The stimuli are stored in a workload library;
- the **monitor**, which collects information about the correct functioning of the target system from the target system and its environment. The data is stored in a collection module, including a data analyzer for the user;
- the **controller module**, which orchestrates the several modules of the FI tool according to parameters specified in the form of configuration profiles.

Fault Injection campaigns consist of five phases, ranging from the definition of the faults to their injection, ending in the analysis of the results. In details:

- **Definition phase**: the user defines the faults to inject and their locations, the workload details and profiles, and the information to be monitored;
- **Set-up phase**: in this phase the user connects the FI environment installed in the host system to the target system, configure the profile of the FI campaign(s), and defines the target system requirements to be validated automatically by the system;
- **Execution phase**: in this phase the user launches one FI campaign at a time, which can be paused and resumed at any moment. A FI campaign is made of several runs, each run executing the target system (in this context the ProSigma system, or part of it) and injecting a fault (FI run, or FIR). Alternatively, runs with no fault injected are called Golden runs (GR), which are useful to observe the nominal behavior of the system;
- **Analysis phase**: this phase serves for analyzing the data collected for possible errors and failure events collected. Depending on the target system, a huge variety of analysis can be carried out;
- **Validation phase**, finally, correlating the errors and failure events, if any, to the target system requirements defined.

11.4.2 The ProSigma FI Tool (ProSigma-FIT)

The proposed framework was implemented into the ProSigma FI tool for the ProSigma system (ProSigma-FIT). A representation of the implemented FI environment is depicted in Figure 11.7. The tool can inject "bit-flips" hardware faults, i.e., emulating the flip from 0 to 1 or viceversa, in one of

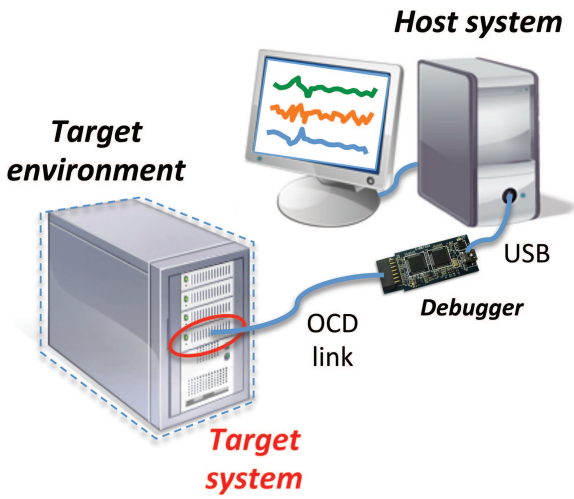

Figure 11.7 Fault injection structure and environment.

the positions of a given registers, These kinds of faults are usually caused by environmental conditions, as charged particles passing through the circuitry. ProSigma-FIT injects faults in the microcontrollers that constitute the ProSigma system, using a host system running Windows 7. The host performs the injection using a debugger communicating through USB port, an external electronic board equipped with circuitry for communicating with a given set of microcontrollers using an OCD port (JTAG in the current case).

ProSigma-FIT is developed as a Java application, and it uses an external library named OpenOCD, which eases the use of JTAG protocol by offering a set of high-level command to a user of the host system. OpenOCD is a project developed at University of Applied Sciences Augsburg [19]. The tool is made of core classes, which include objects for injecting faults and saving data into a MySQL database (*Fault Injector package*), a package for managing the FI environment and the target environment (*ProSigma Environment package*), and objects for monitoring the status of the target system and its environment (*Monitor package*).

11.5 ProSigma Safety Assessment Through FI: Experiments and Results

The ProSigma-FIT was used to assess the safety mechanisms implemented by the ProSigma system, both at hardware and software level, as a whole. The ProSigma-FIT was setup to target both CPU registers and RAM memory

locations of both the targets locations (G channel of the LI card and RPI), and to performed several FI campaigns.

11.5.1 Safety Assessment of the Prosigma System: Experimental Setup

We injected hardware bit flips in the ProSigma system, namely in the G channel of the LI card (target system: TI LM3S2948 microcontroller), which is one of the TMR channels, and in the RPI, node 0 (target: TI TMS570LS3137 microcontroller), which is one of the two modules contained in a RPI card performing the voting functionality. The faults are injected using two JTAG debuggers, namely the Texas Instruments LM3S8962 and the Texas Instruments XDS100. Figure 11.8 shows a photo of the complete experimental setup.

11.5.2 Results

The ProSigma-FIT injected a total of 10,702 faults in few days. Table 11.2 presents the failure modes (system's modules level) monitored by the observers in the LI and the RPI cards. Table 11.3 shows the FI campaigns performed and presents a summary of key results. Figure 11.9 shows the distribution of the failure modes in each FI campaign.

As an example, we selected one of the ProSigma system requirements (R1) to be at validated. Due to the page limit, this chapter does not address the validation of other requirements. The requirement selected is the following:

> ***R1*** *– AFTER the INPUT status is set, the system's client must EVENTUALLY receive a message indicating the SWITCHING STATE and the CORRECT OBJECT STATE.*

Most of the faults injected in the channel G of the LI card (i.e., one of the channels of the TMR) caused effects in the ProSigma system, as shown in Table 11.3. However, as expected, the system managed to tolerate all the faults injected in the LI card. In addition to the more detailed analysis of the fault effects (especially for the ones that caused Crashes and Performance Failures) in the LI card, more comprehensive FI campaigns are needed to gain additional confidence in the system. Previous FI experiments performed for space application [3] have shown that unexpected error propagation due to shared resources such as memory may cause common mode failures.

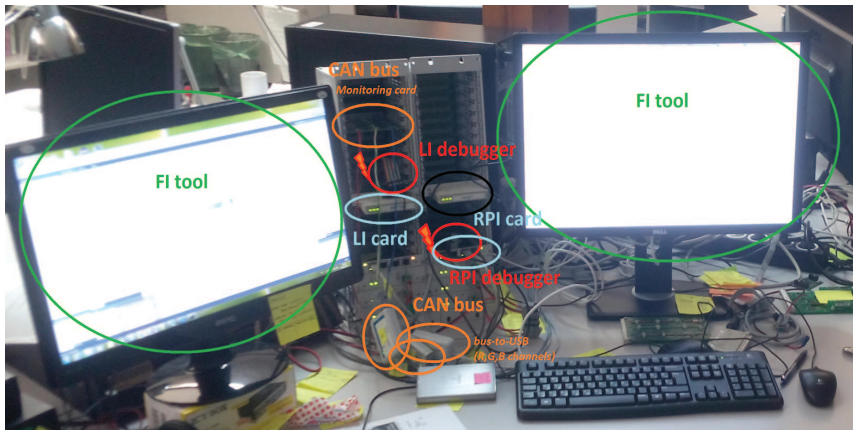

Figure 11.8 The ProSigma system and the FI tool and environment.

Table 11.2 Failure modes

Target	Observer	Failure Mode	Conditions
LI card, G channel	CAN bus	No PIT* messages (NPm)	No PIT messages from the G channel on the CAN bus for more than 3 seconds
LI card, G channel	CAN bus	No CONN** messages (NCm)	No CONN messages from the G channel on the CAN bus for more than 3 sec.
LI card, G channel	CAN bus	Performance failure (P)	PIT or CONN messages appear late on the CAN bus (latency between 1 and 3 seconds)
LI card, G channel	CAN bus	Crash (C)	No PIT and CONN messages for more than 3 sec.
RPI card, module 0	CAN bus	All the same failure modes defined for the LI card	

*PIT is a high-level protocol implemented by the ProSigma system in the LI card.
**CONN is a low-level protocol (right above the CAN messages) implemented by the ProSigma system in the LI card.

Concerning the faults injected in the RPI (voting) card, a single campaign was enough to observe Crash failures that caused the system to stop working, entering in a fail-safe state. Next FI campaigns will be focused on the comprehensive evaluation of the SW voting elements.

As shown in Figure 11.9, the faults injected caused a significant percentage of failures in the target (G channel and RPI). In particular, faults injected in the LI card caused failures in the G channel in about 30% of the

Table 11.3 Summary of FI campaign results

		Target Failures				
Campaign	# FI Runs	NCm	NPm	P	C	ProSigma Behavior
#1 (LI, registers)	674	0	15	5	152	Failure tolerated
#2 (LI, registers)	618	0	2	0	159	Failure tolerated
#3 (LI, registers)	720	0	5	1	172	Failure tolerated
#4 (LI, registers)	721	0	6	0	171	Failure tolerated
#5 (LI, RAM)	2,116	0	10	0	828	Failure tolerated
#6 (LI, RAM)	2,950	0	0	0	854	Failure tolerated
#7 (LI, RAM)	2,150	0	23	1	828	Failure tolerated
#8 (RPI, registers)	753	0	28	1	472	Safety state (Crash)
Total	10,702	0	61	7	3164	

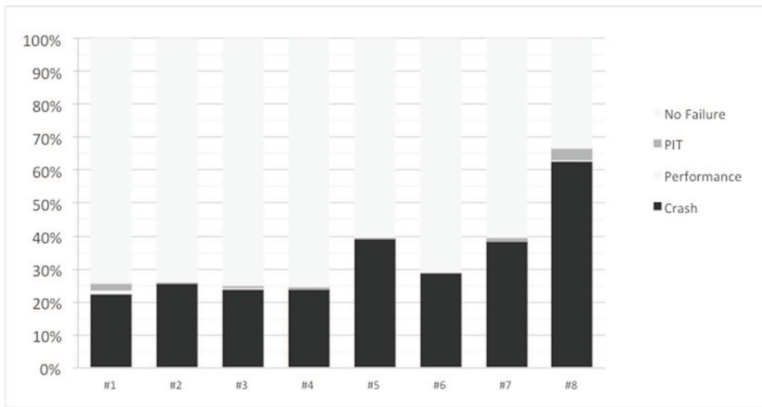

Figure 11.9 Fault injection campaign: failure modes distribution.

times, with "Crash" failures being the most frequent type, followed by "No PIT messages" and "Performance", and with "No CONN messages" failures only occurred when a Crash occurred, without any isolated occurrence. Conversely, more than 60% of the faults injected in the RPI card caused failures, most of which were "Crash"-type. We believe that such behavior is due to additional fault tolerance mechanisms contained in the TMS570LS3137 microcontroller, as the lock-step schema.

Finally, during the campaigns we measured an average injection time below 1ms (round-trip-time host-controller-host). The injection operation is hence quite invasive, being the period of the fastest microcontroller of 6.25 ns. However, the impact of the introduced latency can be tolerated by the single target system. We aim at implementing dedicated module to reduce the injection time in a future work.

11.6 Conclusion

This chapter presented a FI tool based on the JTAG technology proposed for validating a safety-critical railway signaling system, called ProSigma, a TMR system for railway trackside signaling and communication purposes. The ProSigma system has been developed at Prolan Zrt., and has been designed for being certified by the CENELEC standards as SIL 4, the most demanding level in terms of Safety availability.

The FI tool demonstrated to potentially reduce costs related to V&V activities, as it is able to highlight critical situations in which the system under test acts in a hazardous manner. The use of automated FI campaigns, focused on several components of the target system, allows to expose the system to a very large number of fault scenarios, helping gaining confidence in the safety properties of the system under validation. Results from a thorough FI campaigns are presented, illustrating the effectiveness of the FI tool and the approach in general, which confirms to be a valid instrument to help on the V&V of safety-critical system.

References

[1] Bonfiglio, V., Montecchi, L., Irrera, I., Rossi, F., Lollini, P., and Bondavalli, A. (2015). "Software Faults Emulation at Model-Level: Towards Automated Software FMEA," in *Proceedings of 2015 IEEE International Conference on Dependable Systems and Networks Workshops (DSN-W)* (New York. NY: IEEE), 133–140.

[2] European Committee for Standardization. Available at: www.cen.eu

[3] NASA. (2004). *NASA Software Safety Guidebook, NASA-GB-8719.13*.

[4] ISO. (2011). *Product development: software level. ISO 26262: Road vehicles – Functional safety 6*.

[5] Microsoft Corporation. (2014). *Resilience by design for cloud services*.

[6] Barbosa, R., Costa, D., and Madeira, H. (2006). "An empirical approach to assess software off-the-shelf components using fault injection," in *International Conference on Data Systems in Aerospace, DASIA 2006*, Berlin, Germany.

[7] Madeira, H., Some, R. (NASA), Moreira, F., Costa, D., (Critical Software), and Rennels, D. (UCLA). "Experimental evaluation of a COTS system for space applications," in *IEEE/IFIP Int. Conf. on Dependable Systems and Networks* (New York. NY: IEEE), DSN, USA, June 2002.

[8] Silva, A., Sánchez, S., Polo, O. R., and Parra, P. (2014). Injecting faults to succeed. Verification of the boot software on-board solar orbiter's energetic particle detector. *Acta Astronautica*, 95.

[9] Wei, S., Cai Bai-gen, Chen-xi, G., Jian, W., Jing-jing, W. (2010). "Research on reliability evaluation of high-speed railway train control system based on fault injection," in *Int. Conf. Environmental Science and Information Application Technology (ESIAT)*, Vol. 3, 288–293, Wuhan, China, 17–18 July.

[10] Benso, A., and Prinetto, P. (2006). *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation.* Berlin: Springer Science & Business Media.

[11] http://www.prolan.hu/en/divisions/railway-automation/prosigma/

[12] CENELEC. (1999). *EN 50126-1:1999 Railway application, The specification and demonstration of Reliability. Availability, Maintainability and Safety (RAMS), Part 1: Basic requirements and generic process*.

[13] CENELEC. (2007). *EN 50126-2:2007 Railway applications. The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS) – Part 2: Guide to the application of EN 50126-1 for safety*.

[14] CENELEC. (2011). *EN 50128:2011 Railway applications: Communication, signaling and processing systems – Software for railway control and protection systems*.

[15] CENELEC. (2003). *EN 50129:2003 Railway applications: Communication, signaling and processing systems – Safety related electronic systems for signaling*.

[16] Hsueh, M. C., Tsai, T. K., and Iyer, R. K. (1997). Fault injection techniques and tools. *IEEE Comput. J*. 30, 75–82.

[17] Natella, R., Cotroneo, D., and Madeira, H. (2016). "Assessing dependability with software fault injection: a survey", in *ACM Computing Surveys* (ACM: New York, NY), Vol. 48.

[18] ProSigma. *Prolan Process Control Co*. Available at: http://www.prolan.hu/en/divisions/railway-automation/prosigma/

[19] OpenODC. *University of Applied Sciences Augsburg*. Available at: http://openocd.org/documentation/