FPGA-based fault injection architecture for real time software dependability testing

Andre Corsetti
INPE
S.J.Campos, Brazil
andrecor7@gmail.com

***Abstract*: Fault injection is a useful technique for supporting system validation. However, the non intrusiveness of fault injection mechanisms is a challenge for the test architecture. This paper presents a study of fault injection architectures for real time systems testing with focus on dependability attributes. In order to support testing of embedded real-time systems for space applications regarding the use of a complete test methodology, from model building to test generation and automatic test case execution, a FPGA-based fault injection architecture is proposed. The advantages of FPGA-based fault injection architecture are highlighted in a case study which uses a fault injection prototype developed to emulate failures in the communication channel. Fault tolerance is an essential requirement for systems that operate in the harsh space environment. The conception and execution of fault scenarios supported by fault injection mechanisms help the validation of the system behavior regarding dependability attributes like robustness.**

Space subsystem testing; fault injection; robustness; FPGA based architecture;

I. INTRODUCTION

Software validation is an important activity in the development of any software-intensive system that needs to be reliable. Fault injection is an effective mechanism to evaluate the system behavior facing emulated undesirable conditions. Software embedded in space systems are dependable computing systems which require special attention regarding fault tolerance characteristics. This motivates the use and extension of many validation techniques during different software development phases. Fault Injection (FI) imposed itself as a viable solution to the above problems [16]. Several FI techniques have been proposed and practically experimented; they can basically be grouped into simulation-based techniques, software-implemented techniques, and hardware-based techniques [16]. Each have pros and cons, what make them best applicable in different project context and for different objective.

In the Software document of the engineering branch of European Cooperation for Space Standardization – ECSS, it is stated: “5.6.4.2 b) The validation tests shall be “black box”, i.e. performed on the final software product to be delivered, without any modification of the code or of the data” [2]. This widely used guideline in space projects sets a requirement that make most of the fault injection methods unfeasible at latter projects phase of black box software validation testing, since it is denied the possibility for source code modification, source code adding, or software data modification.

For the final software product validation as single black box software or for validation of the dependability attributes of communicating space systems, one should only use the provided interfaces of the system. In this scenario a FPGA-based fault injection architecture is proposed. It supports the use of a formal test methodology based on automatic test generation and fault injection test case execution. This architecture integrates and extends the methodologies and tools used in [3] and [4] for testing space systems.

This paper aims at presenting a fault injector prototype for interoperability robustness testing, considering deviations in communications, developed at INPE and discuss an extension of the testing architecture to encompass chip level faults. The paper is organized as follow. Section 2 presents a brief discussion about fault injection and existing architectures. Section 3 presents a fault injector prototype developed and in use at INPE, section 4 discuss possible extension of the presented fault injector considering the existing test methodology and execution environment, and section 5 presents an extension of the fault injection architecture for dependability testing using FPGA. Section 4 concludes the paper with future work.

II. FAULT INJECTION

Fault injection is done to upset the system under test, in a controlled manner, with possible conditions, events or data that are undesirable in normal operation, aiming at the validation of the system behavior. Test scenarios for fault injection must mimic possible occurrences of real undesirable conditions, events or data that the system may endure in operation. The source of the faults detected in the software operation may be: hardware or software.

Hardware faults occurring during system operation are categorized mainly by duration. Permanent faults are caused by irreversible device failures within a component due to damage, fatigue, or improper manufacturing. Once a permanent fault has occurred, the faulty component can be restored only by replacement or, if possible, repair. Transient faults, on the other hand, are triggered by environmental disturbances such as voltage fluctuations, electromagnetic interference, or radiation. These events typically have a short duration, returning the affected circuitry to a normal operating state without causing any lasting damage (although the system state may continue to be erroneous). Transients can be up to 100 times more frequent than permanents, depending on the system's particular operating environment. Intermittent faults, which tend to oscillate between periods of erroneous activity and dormancy, may also surface during system operation. They are often attributed to design errors that result in marginal or unstable hardware. [5]

Software faults are caused by the incorrect specification, design, or coding of a program. Although software does not physically "break" after being installed in a computer system, latent faults or bugs in the code can surface during operation especially under heavy or unusual work-loads and eventually lead to system failures. [5]

The methods and architectures to mimic these faults are, as said, normally distinguished as: simulation fault injection; software fault injection; and hardware fault injection.

In simulation fault injection the system under test is immersed in a computational environment that may mimic the faults, the system under test is normally a simulated model prototype. The simulation injection has the benefit of been feasible at early development stages of a project and providing ease of use and control over the test. The disadvantage is that simulation systems are complex by their own, so normally it is needed a degree of abstraction in the tests and so forth a differentiation from the real system.

In software fault injection, the objective of software fault/error injection techniques is to modify the hardware software state of the system under software control, thus causing the system to behave as if a hardware fault were present. The argument here is that since hardware functionality is largely visible through software, faults at various levels of the system can be emulated. Hence, this method of fault injection is quite versatile because of the ability to alter the state of registers and memory, is less expensive in terms of time and effort than hardware implemented techniques, and the system under study is never damaged during the injection. [6]

Advantages of software fault injection are flexibility and controllability. Code may be altered, inserted or coupled in the system under test to deterministically provide the mean of the fault. Disadvantage is intrusiveness to enable the software injector system. The altered software may not correspond to the behavior of the unaltered one. Common methods include offline fault injection, where piece of codes are altered to provide a faulty code, and online fault injection, where piece of codes are inserted or coupled in the software under test to provide the mechanism of the fault.

Hardware fault injection uses separate hardware components to stimulate the fault in the system under test. Faults may be induced by special hardware, as radiation exposition [7], faults may be inserted by hardware signals, as in pin signalizing faults [8], or faults may be inserted in the system under test by special fault injector hardware in the test target interfaces [3]. The advantage of hardware injection is its intrusiveness compared to software injection, hardware injection may need less or none altered/inserted code, and it does pass to the external hardware most of the fault mechanism computational load. The disadvantage with these approaches is that the fault and error injection requires special hardware. In addition, these approaches require accessibility to the hardware of the target system which may be difficult or extremely expensive to accomplish. Furthermore, this approach has the possibility of damaging the system under study. [9]

III. PROTOTYPE UNDER USE

Embedded software robustness is a vital requirement in space domain applications. A fault injector for robustness testing of communicating systems [3] was developed at INPE, as an extension of the QSEE project [15]. This fault injector is a hardware fault injector that acts at the communication channels of the target system via its interfaces with the test system. This fault injector, named FEM (Failure emulator mechanism), emulate failures in the communication channel of interoperable space subsystems without modifying any source code of the System Under Testing (SUT).

The injector acts in the middle of a communication channel and can apply corruption of data and timing faults into the messages exchanged by the sub-systems (S_1 and S_2). Figure 1 shows the FEM as part of the Test System, improving the execution of interoperability test cases with controllability and observability [3] in the lower interfaces

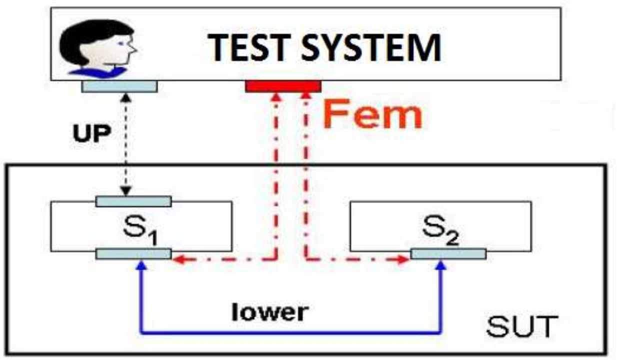

Figure 1. FEM architecture

FEM was implemented as a hardware fault injector using a dedicated 16 bits micro-controller running at 16MHz and discrete electronic circuitry. It operates counting messages that are transferred in the communication channel in each communication way. When injecting faults, the mechanism is aware of the structure of the protocol used in the communicating channel, and can work byte wise in applying faults.

An unwanted overhead is injected by FEM when analyzing bytes transferred. The overhead is less than two bytes time transmitted in the communication channel. So a communication channel of 9600 bits per second would have, in worst case, an unwanted overhead of about two milliseconds.

The fault injector is integrated at the automated test environment [4] and supports the execution of the test cases automatically generated following the model-based approach

presented in [3]. The benefit of this fault injector is the very small intrusiveness in the SUT. The fault injector is transparent when not injecting faults, and adds a negligible unwanted time overhead when injecting faults. The disadvantages of the injector is its capacity of space coverage, the fault injector can only inject faults in the system's communication interfaces, not been able to possible apply memory faults, processor faults or other internal system faults.
Possible faults injected by FEM violate two attributes of the messages, its timing and its data. When inserting timing deviations in the messages, FEM can manipulate the messages to cause: lost messages; messages delays; message ordering change; communication channel delay; and messages clutching. When manipulating the data of the message, FEM can: corrupt data; insert data; duplicate data; and delete data.
Lost messages are messages with theoretical infinite delay, it does not pass from the sender to the receiver. Message delay is a deviation delay inserted in the communication interfering one message, possibly causing communication timeouts or losing synchronism. Message ordering change is a case of a major delay deviation applied to a message, making it be received after other sent messages. Communication channel delay is a delay inserted in the channel, all messages that are transferred in it are delayed by some amount of time. Message clutching is a specific case of a delay injection in a message or a group of messages, making two or more messages clutch together and been passed to the receiver in a small interval.
Data corruption is the application of a single or series of bit flips in the message been transferred. Data insertion is the stuffing of new bytes in some point of the message. Data duplication is the duplication of the message and passing both to the receiver after some defined time interval between them. This time interval can even be enough to the duplicate message being received after other sent messages. Data deletion is the removal of some bytes from the message before passing it to the receiver.
In Figure 2 one can see the FEM added to the Test System facilities aiming at the validation of the expected robustness behavior of the target subsystem (SwPDC) facing delayed message received from EPP.

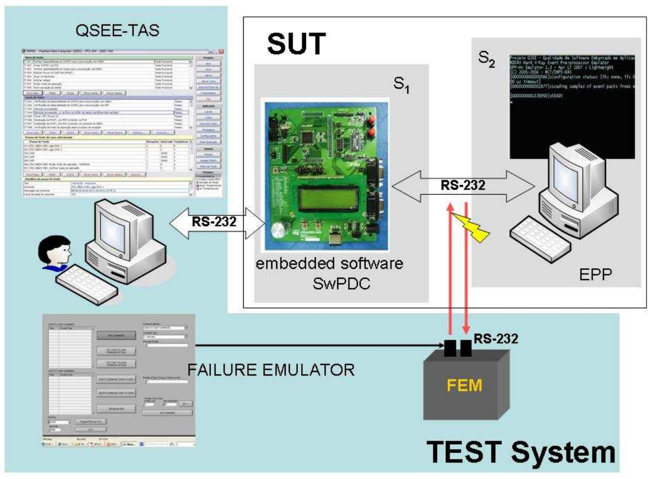

Figure 2. Test Environment

IV. Fault Injector Extension

Aiming at the generalization and extension of the fault injection mechanism presented in last section, a new architecture is proposed based on a FPGA infrastructure. The architecture takes into consideration the existing automated test executor presented in [4] and the model-based approach for test generation, named InRob, presented in [3]. InRob creates service-based interoperability models of the communicating systems and extend these models to cope with robustness testing. Based on the cause-effect rationale the InRob approach guides the addition of states and transitions in the nominal models of interoperability in order to represent the expected robustness behavior of each SUT subsystem facing delayed message received from the other communicating subsystem. This approach can be extended to be equally applicable to some other types of faults, even considering fault at chip level. The process of extending models to encompass fault behavior is presented in [1]. The models would be extended to represent system behavior related to undesirable events of memory and processor faults, such as bit flip. From those extended models, one can generate test cases automatically and automate the execution. of dependability testing.
The test execution automation using the new fault injector would be a challenge because the automatic test engine would have to have some control over the fault mechanism in order to guarantee traceable test results. Actually, the QSEE-Tas tool for automatic test execution [4], supports pin insertion emulated faults in a system under test, as it uses multiple ports for communication and to control the system under test. The tool can be extended to control not only the system under test but also an additional tool to inject faults in processor and memory levels, that would be the new proposed fault injector executor.
The extended test architecture should be able to use FEM fault injection capabilities, and also inject memory and processor level faults in the system under test, as well as be able to use the existing methodology and test execution tools.

V. Proposed Architecture

The new proposed FPGA-based architecture takes advantage of the available soft-cores computers to space systems, as LEON, and the capability of the FPGA to emulate micro-chip hardware. The architecture would use a FPGA to emulate the space system's target micro-chip, and also provide hardware circuitry provision to create into the FPGA a mechanism to generate faults in specific points of the LEON emulated chip or memory areas. Figure 3 depicts the use of the FPGA to emulate the target micro-chip and to create to fault injector mechanism that will have access to processor and memory points of the emulated chip.
In this scenario, some of the FPGA pins would be used by the emulated target chip as its IO ports, and some pins of the FPGA would be used to control the fault injection mechanism. Since both emulated micro-chip and the fault injector resides inside the same FPGA, an alternate route may be created from

the fault injector to points of memory or processor areas, to create controlled faults as bit flips at chip level.

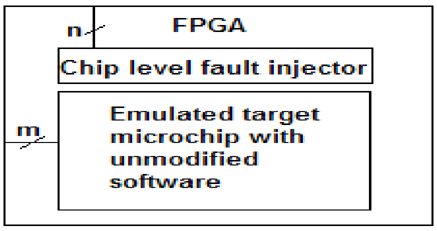
n
FPGA
Chip level fault injector
m
Emulated target microchip with unmodified software

Figure 3. FPGA circuit with emulated chip and fault injector

The real time embedded space software would be loaded in the FPGA emulated micro-chip and be able to run unmodified. The benefits of this architecture is that no modification is necessary in the software, and that the fault injector can be seen as a parallel running hardware, using the characteristics of parallel processing of FPGAs, with access to chip inside circuitry, the fault injection can be deterministic in time and space when injecting faults, that means that the fault would the triggered at a predicted time and in a known point of the emulated chip.

The fault injector circuit would be commanded by dedicated FPGA pins, making it possible to be automated by testing tool, and letting emulated target chip IO ports free to be manipulated by FEM.

Some limitation of this architecture is that the target micro-chip circuit is emulated by the FPGA. Even thou the logic of the micro-controller is the same as the real micro-controller, the circuit and its physical operation is not. In this case, it is the circuit and physical operation of the FPGA.

This architecture would provide an emulated micro-chip, capable of loading the developed space software, and ready for use in dependability tests, since it is not target software dependent. The test bed would extend the actual possible fault capabilities of the test environment integrating itself with a complete test methodology and realizing a general and ready to use test bed for validation of space software.

VI. FUTURE WORK

The use of FPGA for speeding up fault simulation can be found in [10], [11], [12], [13] and [14]. The closest one found is [16], which uses FPGA to emulate circuits, and implements a mechanism concurrent to the test emulated circuit to insert faults in it. An extension of his referred work is the use of a soft core for faults injection in operating space software, and the proposed FPGA-based architecture would be integrated to a complete testing process.

Theoretical analysis was done for the proposed architecture, and it was considered feasible. It is important now to evaluate FPGA development environments for creating experiments.

REFERENCES

[1] Mattiello-Francisco, F.; Martins, E.; Corsetti, A.; Cavalli, A.R.; Yano, E.;" Extended interoperability models for timed system robustness testing". *LATINCOM '09 : IEEE Latin-American Conference on Communications*, Colombie (2009)

[2] ECSS-E-ST-40C, "Software", Third issue, 06 March 2009. European Cooperation for Space Standardization

[3] Mattiello-Francisco, Maria de Fátima. InRob – Uma abordagem para testes de Interoperabilidade e de Robustez de subsistemas de tempo-real intesivos em software. 2009. 212f. Tese de doutorado em Engenharia Eletrônica e Computação, na área de Informática – Instituto Tecnológico de Aeronáutica, São José dos Campos.

[4] Silva, Wendell Pereira da. QSEE-TAS: execução automatizada de casos de teste para software embarcado em aplicações espaciais. São Jos_e dos Campos: INPE, 2009. 103p. ; (INPE-15662-TDI/1438). Dissertação (Computação Aplicada) Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 20/11/2008.

[5] Clark, J. A., Pradhan, D. K.. Fault Injection – A method for validating computer-system dependability. Computer, New York, v. 28, n.6, p. 47-56, June, 1995.

[6] G.A. Kanawati, N.A. Kanawati, and J.A. Abraham, "FERRARI: A Flexible Software-Based Fault and Error Injection System," IEEE Trans. Computers, vol. 44, no. 2, pp. 248-260, Feb. 1995.

[7] J. Karlsson, P. Lide´n, P. Dahlgren, R. Johansson, and U. Gunneflo, "Using Heavy-Ion Radiation to Validate Fault-Handling Mechanisms," IEEE Micro, vol. 14, no. 1, pp. 8-23, Feb. 1994.

[8] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, and D. Powell, "Fault Injection for Dependability Validation—A Methodology and Some Applications," IEEE Trans. Software Eng., vol. 16, no. 2, pp. 166-182, Feb. 1990.

[9] G.A. Kanawati, N.A. Kanawati, J.A. Abraham, "FERRARI: A Flexible Software-Based Fault and Error Injection System", IEEE Trans. on Computers, Vol 44, N. 2, February 1995, pp. 248-260

[10] S. A. Hwang, J. H. Hong, C. W. Wu, "Sequential circuit fault simulation using logic emulation", IEEE Trans. On CAD, Vol. 17, No. 8, Aug. 1998, pp. 724 -736

[11] K. T. Cheng, S. Y. Huang, W. J. Dai, "Fault emulation: A new methodology for fault grading", IEEE Trans. On CAD, Vol. 18, No. 10, Oct. 1999, pp. 1487 -1495

[12] L. Antoni, R. Leveugle, B. Fehér, "Using Run-time reconfiguration for Fault Injection in Hardware Prototypes", IEEE Int.l Symp. on Defect and Fault Tolerance in VLSI Systems, 2000, pp. 405-413

[13] C. Hungse, E.M. Rudnick, J.H. Patel, R.K. Iyer, G.S. Choi, "A gate-level simulation environment for alphaparticle-induced transient faults", IEEE Trans. on Computers, Vol. 45, No. 11, Nov. 1996 , pp. 1248-1256

[14] B. Parrotta, M. Rebaudengo, M. Sonza Reorda, M. Violante, "New Techniques for Accelerating Fault Injection in VHDL descriptions", IEEE Int.l On-Line Test Workshop, 2000

[15] QSEE – Quality of embedded software for space applications. http://www.cea.inpe.br/qsee

[16] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante, "Exploiting FPGA for Accelerating Fault Injection Experiments," ioltw, pp.0009, Seventh International On-Line Testing Workshop, 2001