A Virtual Fault Injection Framework for Reliability-Aware Software Development

Andrea Höller, Georg Macher, Tobias Rauter, Johannes Iber, and Christian Kreiner
Institute for Technical Informatics
Graz University of Technology, Austria
{andrea.hoeller, georg.macher, tobias.rauter, johannes.iber, christian.kreiner}@tugraz.at

***Abstract*—Ever more dependable embedded systems are built with commercial off-the-shelf hardware components that are not intended for highly reliable applications. Consequently, software-based fault tolerance techniques have to maintain a safe operation despite underlying hardware faults. In order to efficiently develop fault tolerant software, fault injection is needed in early development stages. However, common fault injection approaches require manufactured products or detailed hardware models. Thus, these techniques are typically not applicable if software and hardware providers are separate vendors. Additionally, the rise of third-party OTS software components limits the means to inject faults.**

In this paper, we present a virtual fault injection framework that simulates safety-standard aligned fault models and supports OTS software components as well as widely-used embedded processors such as ARM cores. Additionally, we show how to integrate the framework into various software development stages. Finally, we illustrate the practicability of the approach by exemplifying the integration of the framework in the development of an industrial safety-critical system.

I. Introduction

Dependability is of the utmost importance for many embedded and cyber-physical systems. For example, a reliable and safe operation is essential for systems whose failures could result in loss of life, significant property damage, or damage to the environment. Since such systems have to realize ever more features they have increasing demands on computing performance. In some domains this lead to the trend of using commercial off-the-shelf (OTS) processors that are not intended for highly reliable applications. By using OTS processors, the designers can use state-of-the-art hardware that offers a high performance. Moreover, costs can be reduced significantly, since OTS components come at lower costs than their certified counterparts [1], even though they may imply in additional redundant units. However, note that such components are not suited for highly-critical domains where it is only allowed to use certified hardware (e.g. railway).

At the same time, the semiconductor industry continues with structure and voltage downscaling, which leads to highly integrated but also highly sensitive devices. Reliability issues arise from permanent hard errors due to manufacturing, process variations, aging, and wear out [2]. Furthermore, there are soft errors caused by energetic radiation particles, capacitive coupling, electromagnetic interference, and other sources of electrical noise [3]. Historically, soft errors were mostly of concern for space applications. However, it is expected that failures in time for a chip will increases with Moore's law and thus handling soft errors will be a key challenge in future embedded system design.

Consequently, software should be effective in coping with faults in the underlying hardware. In order to master this challenge, we propose to create systems that are inherently resilient. Today, the issue of hardware fault tolerance is typically assigned to specialized persons. To enhance the resilience of the entire software system, we propose to introduce reliability-awareness throughout all stages of software development. For example, the vulnerability to hardware faults can be significantly reduced by exploiting inherent fault masking properties of software algorithms and defensive programming.

However, to establish this approach, software developers need to understand the impact of hardware faults on their software. Even though analysis techniques exists to evaluate software behavior in the presence of hardware errors (e.g. Hardware-Software Interaction Analysis), there still exists a major need for a fault injection tool to evaluate the vulnerability to hardware faults in early development stages. The tool should provide an understanding and measurement of how hard and soft errors affect the behavior of the software. This allows the identification of software parts that are especially vulnerable to hardware faults in early stages. Thus, software developers can early adapt their software in order to fulfill the reliability targets. This could prevent late stage redesigns. To ensure a high level of feasibility, a simple integration into an existing tool chain is desired.

Another trend is to use third-party OTS software. Typically, their source code is not available. Thus, to assess the reliability of such software components and software systems including these components, a fault injection technique that does not rely on knowing the source code is needed.

This paper contributes towards filling these gaps of reliability assessment with the following main contributions.

- Introduction of a software-based virtual fault injection (VFI) framework that is applicable without knowing hardware or software implementation details.
- Proposal how to integrate the VFI framework in a typical software development process in order to establish reliability-awareness throughout the implementation and test stages.
- An industrial case study illustrating the integration of the VFI framework in the development of a safety-critical controller for hydro-electrical power plants.

II. Related Work

Fault injection (FI) allows better understanding of the system behavior, if faults are present. Table I gives an overview of FI techniques and their applicability for OTS-based systems. Depending on the design stage, faults can be injected with hardware FI techniques into finalized parts (e.g., by using radiation or manipulation), or during earlier phases using adaptable prototyping approaches [4]. During concept phases system-level description languages (e.g., SystemC) could be used the trigger of faulty hardware behavior [5]. Further established approaches are simulation- and emulation-based techniques that manipulate high-level hardware descriptions (e.g., VHDL models) [6]. A considerable disadvantage of these methods is that the detailed design of the processor (e.g., RTL model, netlist) is required. This information is typically not available when using third-party processors.

An approach to perform FI for OTS processors-based platforms is to use their on-chip debug features (e.g., JTAG) as proposed in [7]. However, these approaches are very platform dependent, since they rely on specific hardware features. Another approach is software-based FI that executes additional software to modify the state of the system [8]. These methods can represent faults in hardware components, which are accessible by the software, such as registers, memory, etc. However, software-based FI offers only limited levels of observation and control and it is very challenging to model permanent faults. Furthermore, it requires to modify source code. Consequently, it does not support closed-source OTS software components. Thus, the Islam et al. [9] proposed a binary-level fault injection technique. However, their approach only supports input errors and does not model specific hardware faults.

To overcome these limitations of traditional FI techniques, there are proposals to adapt emulators performing hardware virtualization to simulate faults. The Quick EMUlator (QEMU) [10] is open source and targets the emulation of hardware for embedded systems. It features the fast emulation of several CPU architectures (e.g., ARM, x86, Sparc, Alpha) on several host platforms (e.g., ARM, x86, PowerPC). Recently, QEMU-based tools realizing the injection of soft errors have been proposed [11], [12]. Permanent memory-related faults are considered in [13]. For our approach, we extend the QEMU-based FI tool presented in [14]. It supports advanced realistic memory and processor fault models that go beyond simple bit-flips and stuck-at faults (SAF).

TABLE I
Comparison of fault injection techniques

FI technique	*Support of* *OTS HW*	*OTS SW*	*No external HW required*
Hardware FI	-	✓	-
Simulation-based FI	-	✓	✓
Emulation-based FI	-	✓	-
JTAG-based FI	✓	✓	-
Software-based FI	✓	-	✓
Proposed VFI	✓	✓	✓

However, it is not possible to simulate faults that can not be mapped to the processor components represented by QEMU (e.g. instructions, memory cells/addressing). Such faults include for example, voltage glitches, clock drifts, short circuits, etc. Another drawback of the QEMU-based approach is the relatively slow simulation speed (e.g. compared to emulation-based approaches). Thus, we propose to design the fault library carefully, so that only representative faults are tested.

Although, many researchers have focused on FI, only a very few studies show how to apply them during various development stages in domains with high dependability requirements. To the best of our knowledge, only Pintard et al. [15] present an initial approach for automotive development according to ISO 26262. However, their approach depends on techniques that are not applicable when using OTS hard- and software.

III. Virtual Fault Injection Framework

A. Fault Modeling

The proposed VFI framework supports the fault model shown in Table II. This allows to fulfill IEC 61508 SIL 3 requirements regarding fault modeling to assess fault tolerance techniques for processor and RAM [14]. According to their duration, the simulated faults can be permanent, transient, and intermittent. The fault can be triggered if a certain program counter (PC) is reached or whenever the target resource is accessed. Furthermore, the accuracy of the fault models is increased by taking particularities of memory components into account. The framework features functional fault models describing the deviation of an observed and a specified behavior after a certain number of memory operations have been performed [16]. The framework supports static and dynamic functional faults that are sensitized by performing operations on a single cell or on two cells. The fault model includes faults that are triggered after one or two operations. The faults can be defined in a user-friendly XML format. For a detailed description of the fault model the reader is reffered to author's previous work [14].

B. Fault Injection Procedure

We propose a reliability assessment based on four steps as shon in Fig. 1. First, the hardware usage characteristics of the application are profiled. Based on this information a fault library is created containing fault models for an efficient fault injection campaign. Next, the framework injects the defined faults and the resulting application outputs are then saved. Finally, these outputs are interpreted and a clear and detailed report is provided. We implemented the framework for an ARM9 architecture. However, the approach could be relatively easiliy adapted for other architectures.

1) Application Profiling: To improve the efficiency of fault injection campaigns, the VFI framework first collects execution statistics for the tested application. A golden run is executed to record the memory and register usage, similar to that proposed in [17]. QEMU dynamically translates currently processed guest instructions to host instructions. The execution statistics are collected before the translation takes place. More

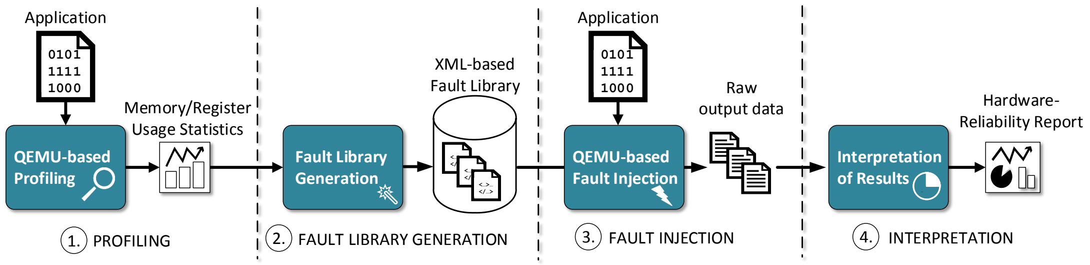


Fig. 1. Overview of VFI framework showing the steps of hardware reliability assessment of an application-software.

specifically, if the instruction performs a memory or register access, the current PC and the address is logged.

If the assessed software is non-functional and purely implements fault tolerance mechanisms, this step can be skipped. Then the fault library generation is not based on an execution profile, but on a definition of faults that should be handled by these mechanisms.

2) Fault Library Generation: The fault library consists of multiple XML files, where each file defines a fault that should be injected. Although, the FI tool allows to inject multiple faults at the same time, we only consider single faults to limit the execution time. Based on user-provided information a script generates the fault library. The user can define the target components, fault types and number of injected faults. For example, to achieve a quick overview of the behavior of the application under faulty circumstances SAF and bit-flips are often sufficient.

Instruction Decoder Faults: Random instructions chosen from the binary are replaced with a NOP instruction to simulate 'inactive decoder faults' or by another randomly chosen instruction to simulate 'wrong instruction decoder faults' as defined in [18].

Memory/Register Cell and Address Decoder Faults: The tested fault modes and the number of cells that are affected are user-defined. For example, the fault can have an effect on only one bit or on a whole word.

To increase the efficiency of permanent fault injections, the hardware parts that are used more frequently by the application are more likely to become victims. Therefore the register and memory usage statistics are used. Target cells are chosen randomly by using probabilities that are weighted according to the number of accesses of these cells.

Transient faults are generated by randomly chosing PC/address pairs from the execution statistics. To ensure that transient faults are injected at a point in time when the target address is used, the fault is triggered whenever the given PC is reached.

3) Fault Injection: To inject faults during the execution of a binary, QEMU is extended. A fault injector reads the currently processed instruction and a fault library. This information is used by a controller to decide whether and where a fault should be injected. For a more detailed description about the FI extension, we refer to [14].

4) Interpretation of Results: Finally, the consequences of the injected faults are evaluated by analyzing the generated outputs. We consider the following three fault effects. First, the fault could be masked, which means that the output is identical to the result of the golden fault-free execution.

Second, the execution could crash. For example, it the fault could delaying the output (e.g., due to an infinite loop) or cause a segmentation fault. There are well-known techniques to manage these kind of faults, such as a watchdog or fault handling mechanisms provided by the operating system. To identify time-related faults, we execute the faulty application with a timeout. If the execution has not finished after a time twice as long as the time needed for the golden run, the fault is considered as a timing fault. Other faults that are detected by the system (e.g., segmentation faults) are identified by searching error messages in the output files.

Third, we consider silent corruptions that lead to incorrect outputs. Such faults are especially hard to handle by the application, since the system seems to work correctly, although it produces erroneous results. Thus, it is of utmost importance for the developer of a reliable system to be aware of these kind of faults. The VFI simulates faults that potentially cause silent-corruptions and gives feedback about the resulting erroneous output. Then, the developer can design appropriate mechanisms according to the impact of the error.

TABLE II
DETAILS ABOUT FAULT LOCATION AND MODES OF SUPPORTED FAULTS

Comp.	*Target*	*Fault modes*	*Implementation*
CPU	Instruction decoder	New value	Replaces current instruction with a given instruction
	Register cell	Bit-flip, SAF, new value	Changes the data of the register according to the fault mode
	Register address decoder	Bit-flip, SAF, new value	Changes the address of the register according to the fault mode
RAM	Address decoder	Bit-flip, SAF, new value	Changes the address according to the fault mode
	Memory cell or R/W logic	Bit-flip, SAF, static and dynamic fault models	Changes the data according to the fault mode

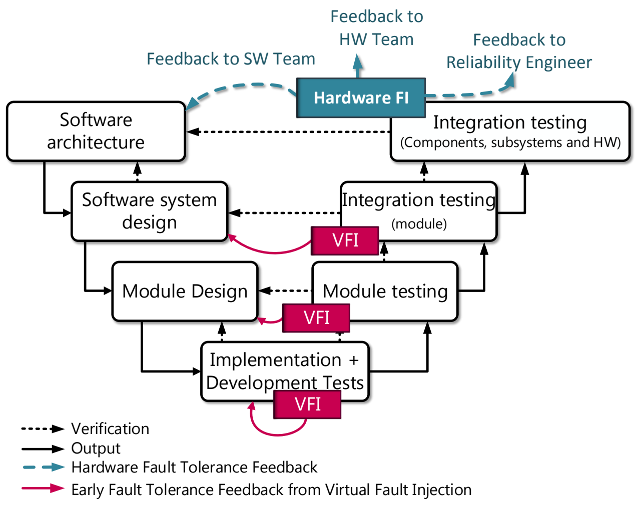

Feedback to HW Team
Feedback to SW Team
Feedback to Reliability Engineer
Hardware FI
Software architecture
Integration testing (Components, subsystems and HW)
Software system design
Integration testing (module)
VFI
Module Design
Module testing
VFI
Implementation + Development Tests
VFI
Verification
Output
Hardware Fault Tolerance Feedback
Early Fault Tolerance Feedback from Virtual Fault Injection

Fig. 2. Proposed Fault Injection in Development Process

In order to provide clear feedback to the developer, we summarize the frequency of these fault effects in an XML report. For more detailed information, the report also includes details of the injected faults and links to the corresponding results generated by the application.

C. Tool Chain Integration

The VFI framework is well suited for the integration in an existing tool chain. It does not require to adapt the software or the binary that should be assessed. In many tool chains QEMU is already used for cross-platform development. Since the VFI framework is an extension of QEMU, the standard QEMU installation routines can be applied and various operating systems are supported (e.g. Windows, Linux, OSx). For assessing a binary with the VFI framework, it is only necessary to execute a script managing the above described steps of the FI campaign. Finally, the results of the FI experiments are represented in an XML format that allows an easy parsing and representation by existing tools.

D. Support of OTS-Based Systems

1) VFI for the Assessment of OTS Software Components: As the use of OTS-based software components in the context of safety-critical applications is increasing, the problem of profiling this software is of paramount importance. For example, the automotive AUTOSAR standard proposes a higher penetration of OTS software across product lines [9]. They should enhance the re-usability, reduce the time to market, and improve product quality [19]. Typically, these components are commercially available without source code that could be used for assessment purposes. However, before reusing a software component, the context it is intended to be build in has to be carefully evaluated [20]. This includes hardware and physical aspects. To assess the reliability and behavior of OTS software when faults appear in the underlying hardware, fault injection is needed. Additionally, this supports the development of component wrappers that should detect errors or suspicious activity of the OTS component.

The proposed VFI is well suited to analyze the reliability characteristics of OTS software, since it can be applied straight-forward without the need to change the source code or executable. Additionally, it offers a fault model that is able to realistically represent the target hardware.

2) VFI for Systems Based on OTS processors: The QEMU-based VFI framework provides a high-level emulation of typical embedded processor architectures, such as ARM cores. Based on this abstraction, the VFI framework supports realistic instruction-level hardware faults without the need for detailed hardware models. Thus, it is applicable for third-party processors without provided netlists or HDL models.

IV. Virtual Fault Injection Throughout the Development Process

Here, we describe how to use the VFI framework in various stages of the V-model, which is recommended by many safety standards (e.g., ISO 26262, IEC 62508).

A. Fault Injection from Specification to Design

Typically, the left side of the V-model is related to development activities. Here, the VFI framework is applicable as soon as an executable software is available. For example, if in an incremental approach is adapted, an executable version of the software is early available. However, then several gold runs may be needed along the process causing overhead. Nevertheless, early feedback about the reliability of the software can prevent expensive late redesigns.

B. Fault Injection during Implementation and Test

Often, a software system is divided into software modules, which are developed by individual teams. In parallel the hardware development takes place. After testing and integrating the software components, the software system is deployed on the hardware. Typically, FI campaigns are conducted after the integration of hard- and software. If they lead to the result that the fault tolerance of the system does not fulfill the requirements, usually the persons that are responsible for reliability take actions. This actions could involve changes in the hardware or software. For example, additional reliability is established by adding extra fault tolerance methods (e.g., increasing the level of redundancy). To prevent such overhead and late stage redesigns, we propose to raise the awareness regarding hardware faults throughout implementation and test stages.

1) VFI in the Implementation Stage: During the implementation stage, the programmer can use the VFI framework to receive reliability metrics of the newly created code. Developers of certain software parts, have deep knowledge of their structure and behavior. This can be helpful for improving the fault tolerance characteristics of the software. For example, programmers know the typical value domains of used variables, which helps to create efficient plausibility checks. Furthermore, the code could be modified in such a way that the inherent fault masking capabilities of software are efficiently exploited.

2) VFI in the Module and Integration Testing Stages: Hardware fault tolerance assessment using the VFI framework should complement regular functional testing activities to identify less reliable parts of the software. Software test engineers know each module and/or the entire software system. In contrast to the implementation stage, the modules and system are regarded more like black-boxes, which can pose new views on the resilience characteristics.

3) Fault Injection after Integrating Hard- and Software: We propose to apply traditional hardware FI after the integration of hard- and software. For example, the finalized parts can be analyzed regarding their behavior, if there is to high input voltage or if there are disruptive environmental influences (e.g. high temperature). However, it can be assumed that the identified reliability weaknesses will be reduced due to the prior reliability assessments.

V. Industrial Use Case

We integrated the VFI framework in the software development tool chain of our industrial partner. This company develops next-generation highly reliable controllers for hydroelectrical power plants. Due to the different nature and configurations of power plants, there can be many ways to realize the control functions. Thus, the principles of component-based software-engineering are used to realizing functions with functional block diagrams according to IEC 61131. Systems are built as compositions of components. These components could implement small functions (e.g. basic logic, arithmetic operations), but also complex functions (e.g. controllers, filters). Due to performance and economic reasons the system is based on a commercial OTS ARM926EJ-S processor. This CPU was designed and manufactured for multimedia applications without high reliability requirements. Thus, two processors are used redundantly in an 1oo2 architecture. The software implements several reliability functions. This includes mutual checking of the redundant hardware channels, periodically software-based self-tests, and fault handling mechanisms.

A. Current and Proposed Application of Fault Injection

Several teams are involved in the development as shown in Fig. 3. For creating the hardware platform and basic software (e.g., real-time operating system, network communication, component container) individual hardware and embedded software teams are working in parallel. Additionally, there are engineers developing components using the C program language. They have deep knowledge of how to develop IEC 61131 compliant components. Furthermore, a team focuses on project-specific control engineering aspects and implements control algorithms with functional block diagrams. Currently, only test teams apply fault injection and only consider high-level hardware faults such as DC offsets, current changes, etc.

However, we propose that multiple development teams consider reliability aspects by using the VFI framework. The framework supports the emulation of the used SoC (ARM9

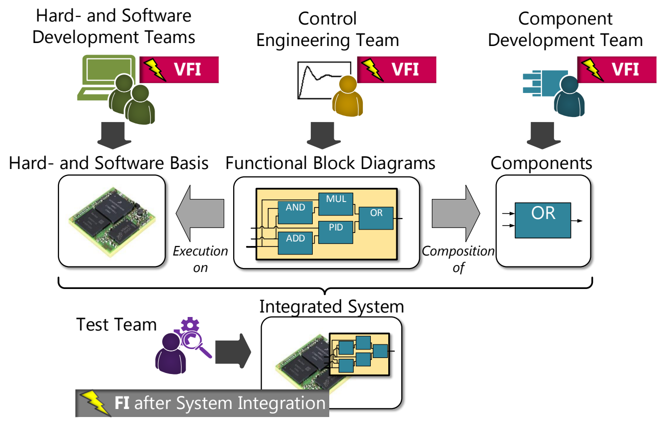


Fig. 3. Teams and their workproducts for the development of a power-plant controller. Currently, fault injection is only applied after system integration. We propose the use of VFI in all teams to assess the reliability.

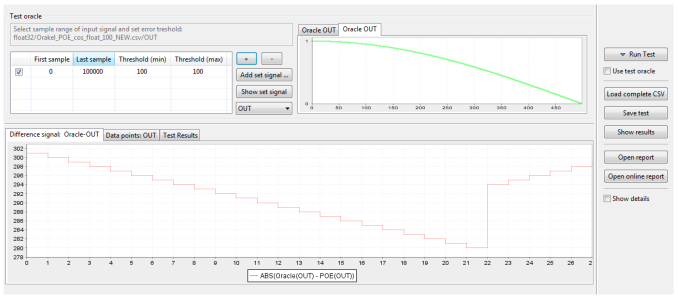
processor, RAM, in- and outputs, etc.) and does not require external hardware. Thus, it can be used to analyze the basic software. Furthermore, the component development team can test the reliability characteristics of each software component. After software integration, also the control engineering team can use the VFI framework to analyze the resilience of the overall control algorithm regarding hardware faults. To exemplify the VFI integration, we present below an extension of an existing component test framework.

B. Integration of VFI in Component Testing

The existing industrial partner test framework allows to remotely test components and is based on techniques described in [21]. The framework consists of two main components: the development host and the embedded target. The development host is realized as an Eclipse plugin that allows to define the component under assessment, input stimuli, and expected output. Then, to represent the embedded target, the given component is executed remotely with QEMU in isolation. This means that each component can be assessed individually without the need for integration. Each component is available as a binary that executes functions processing input stimuli provided by the execution context. This context corresponds to a simple middleware or container that implements the lifecycle management for a single software component. To communicate with the rest of the framework, it exposes the interfaces of the component as a webservice.

The development host shows a graphical comparison of the resulting and expected output values as exemplified in Fig. 4. The expected outputs are generated with a golden model in Matlab. Frequently, the outputs of the embedded system are slightly different due to limited precision. Thus, an accepted deviation from the ideal output curve can be defined. The framework highlights violations regarding this specification.

We extended this test framework with VFI (see Fig. 5). The victim hardware components, number of faults, and fault types can be defined in the Eclipse framework. Then, this information is forwarded to the platform simulating the


Test oracle

Select sample range of input signal and set error treshold:
float32/Orakel_POE_cos_float_100_NEW.csv/OUT

	First sample	Last sample	Threshold (min)	Threshold (max)
☑	0	100000	100	100

Fig. 4. Screenshot of component test framework showing the difference of the expected and resulting output.

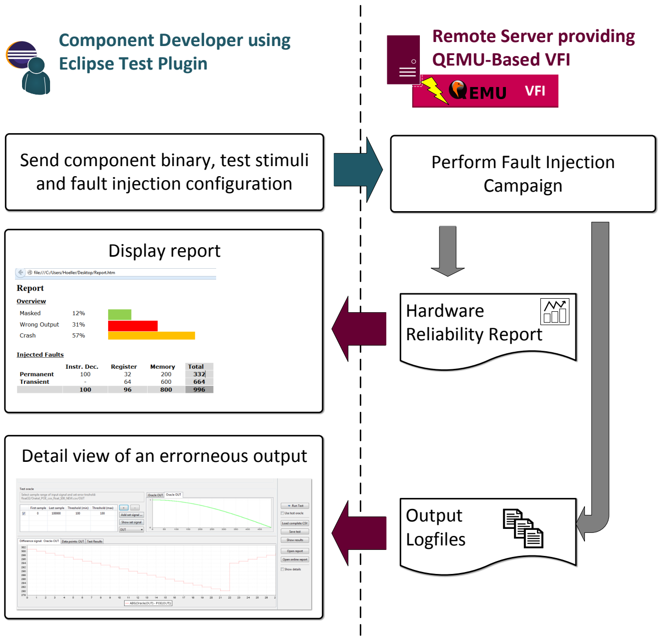

Component Developer using Eclipse Test Plugin
Remote Server providing QEMU-Based VFI
QEMU VFI
Send component binary, test stimuli and fault injection configuration
Perform Fault Injection Campaign
Display report
file:///C:/Users/Hoeller/Desktop/Report.htm
Report
Overview
Masked 12%
Wrong Output 31%
Crash 57%
Injected Faults
Instr. Dec. Register Memory Total
Permanent 100 32 200 332
Transient - 64 600 664
100 96 800 996
Hardware Reliability Report
Detail view of an errorneous output
Output Logfiles

Fig. 5. Component Test Framework Extension with VFI.

embedded target. For the simulation of the embedded target under faulty conditions, the VFI framework executes the four steps described above. Finally, the results of the FI campaign are graphically represented on the development host. The proportion of faults causing crashes, and silent corruptions causing outputs beyond the defined value range is shown. To support the component developer in analyzing the behavior of the components under faulty conditions, critical erroneous results can be inspected by analyzing the corresponding output curve graphically.

VI. Conclusion

In order to create highly reliable systems with low development cost overhead, we propose a tool that allows to introduce reliability-awareness in multiple stages of software development. Software developers need to understand the impact of hardware faults on their software modules. This raises the need for a FI tool that can be easily integrated into existing tool chains without requiring extra hardware equipment. The trend to use OTS hard- and software components further limits the number of applicable FI techniques.

In this paper, we presented a VFI framework that can be applied during various development stages. Furthermore, we proposed the integration of this tool in a typical software development process and illustrated the practicability of the approach with an industrial use case. In the future, we plan to provide the presented framework as open source.

References

[1] O. Goloubeva, M. Rebaudengo, M. M. S. Reorda, and M. Violante, *Software-Implemented Hardware Fault Tolerance*. Springer, 2006.

[2] K. Gruttner, A. Herrholz, U. Kuhne, D. Grosse, A. Rettberg, W. Nebel, and R. Drechsler, "Towards Dependability-Aware Design of Hardware Systems Using Extended Program State Machines," *Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, 2011.

[3] G. P. Saggese, N. J. Wang, Z. T. Kalbarczyk, S. J. Patel, and R. K. Iyer, "An Experimental Study of Soft Errors in Microprocessors," *IEEE Micro*, 2005.

[4] A. Benso and P. Prinetto, *Fault Injection Technqiues and Tools for Embedded Systems Reliability Evaluation*, 2003.

[5] J. Oetjens, N. Bannow, M. Becker, O. Bringmann, A. Burger, and M. Chaari, "Safety Evaluation of Automotive Electronics Using Virtual Prototypes : State of the Art and Research Challenges," in *Design Automation Conference*, 2014.

[6] A. Krieg, C. Preschern, J. Grinschgl, C. Steger, C. Kreiner, R. Weiss, H. Bock, and J. Haid, "Power And Fault Emulation for Software Verification and System Stability Testing in Safety Critical Environments," *IEEE Transactions on Industrial Informatics*, 2013.

[7] A. V. Fidalgo, M. G. Gericota, G. R. Alves, and J. M. Ferreira, "Real-Time Fault Injection Using Enhanced On-Chip Debug Infrastructures," *Microprocessors and Microsystems*, 2011.

[8] H. Ziade, R. Ayoubi, and R. Velazco, "A Survey on Fault Injection Techniques," 2004.

[9] M. M. Islam, N. M. Karunakaran, J. Haraldsson, F. Bernin, and J. Karlsson, "Binary-Level Fault Injection for AUTOSAR Systems," 2014.

[10] F. Bellard, "QEMU, a Fast and Portable Dynamic Translator," in *USENIX Annual Technical Conference*, 2005.

[11] Q. Guan, N. Debardeleben, S. Blanchard, and S. Fu, "F-SEFI: A Fine-Grained Soft Error Fault Injection Tool for Profiling Application Vulnerability," *International Parallel and Distributed Processing Symposium*, 2014.

[12] F. d. A. Geissler, F. Kastensmidt, and J. Souza, "Soft Error Injection Methodology based on QEMU Software Platform," in *Latin American Test Workshop*, 2014.

[13] J. Xu and P. Xu, "The Research of Memory Fault Simulation and Fault Injection Method for BIT Software Test," *Conference on Instrumentation, Measurement, Computer, Communication and Control*, Dec. 2012.

[14] A. Höller, G. Schönfelder, N. Kajtazovic, and C. Kreiner, "FIES: A Fault Injection Framework for the Evaluation of Self-Tests for COTS-based Safety-Critical Systems," in *Microprocessor Test and Verification Workshop*, 2014.

[15] L. Pintard, J. C. Fabre, K. Kanoun, M. Leeman, and M. Roy, "Fault Injection in the Automotive Standard ISO 26262: An Initial Approach," in *European Workshop on Dependable Computing*, 2013.

[16] Z. Al-Ars and A. Van de Goor, "Static and Dynamic Behavior of Memory Cell Array Opens and Shorts in Embedded DRAMs," *Conference on Design, Automation and Test in Europe*, 2001.

[17] S. Chyek, "Collecting program execution statistics with Qemu processor emulator," 2009.

[18] D. Brahme and J. A. Abraham, "Functional Testing of Microprocessors," *IEEE Transactions on Computers*, 1984.

[19] R. Barbosa, N. Silva, and J. Duraes, "Verification and validation of (real time) COTS products using fault injection techniques," *Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems*, 2007.

[20] F. Belli, "Dependability and software reuse - Coupling them by an industrial standard," *International Conference on Software Security and Reliability Companion*, 2013.

[21] N. Kajtazovic, A. Höller, T. Rauter, and C. Kreiner, "A Lightweight Framework for Testing Safety-critical Component-based Systems on Embedded Targets," in *ModComp*, 2014.