Fault Insertion Testing of a Novel CPLD-based Fail-Safe System

Gerhard Grießnig
AVL LIST GMBH
Graz, Austria
gerhard.griessnig@avl.com

Roland Mader, Christian Steger, Reinhold Weiß
Graz University of Technology
Institute for Technical Informatics (ITI)
Graz, Austria
rmader@sbox.tugraz.at, steger@tugraz.at,
rweiss@tugraz.at

***Abstract*—According to the standard IEC 61508 fault insertion testing is required for the verification of fail-safe systems. Usually these systems are realized with microcontrollers. Fail-safe systems based on a novel CPLD-based architecture require a different method to perform fault insertion testing than microcontroller-based systems. This paper describes a method to accomplish fault insertion testing of a system based on the novel CPLD-based architecture using the original system hardware. The goal is to verify the realized safety integrity measures of the system by inserting faults and observing the behavior of the system. The described method exploits the fact, that the system contains two channels, where both channels contain a CPLD. During a test one CPLD is configured using a modified programming file. This file is available after the compilation of a VHDL-description, which was modified using saboteurs or mutants. This allows injecting a fault into this CPLD. The other CPLD is configured as fault-free device. The entire system has to detect the injected fault using its safety integrity measures. Consequently it has to enter and/or maintain a safe state.**

Keywords-IEC 61508; fail-safe system; safety integrity; fault insertion testing; fault injection; CPLD; VHDL

I. INTRODUCTION

A new architecture for fail-safe systems [10] has been developed recently. The architecture is homogenously redundant and consists of two channels. In contrast to other fail-safe system architectures [1] this novel architecture contains no microcontrollers. Instead the safety functions and the safety integrity measures are exclusively implemented using one CPLD (complex programmable logic device) per channel.

A fail-safe system based on this novel CPLD-based architecture is being developed by SIEMENS. The application domain of this system is industrial automation. The fail-safe system is able to execute safety functions if demanded. Each channel is able to perform the safety functions independently. Additionally each channel realizes measures to detect faults. This guarantees sufficiently high safety integrity. The functionality of the system is entirely implemented in hardware.

The certification of the fail-safe system in adherence with the standard IEC 61508 [6] is aspired. Thus it is necessary to fulfill the requirements of this standard. One of the verification measures, which are required by the IEC 61508, is fault insertion testing. The accomplishment of fault insertion tests is necessary to verify the realized safety integrity measures of the fail-safe system. This paper presents the new approach of fault insertion testing of a novel CPLD-based fail-safe system.

The rest of this paper is organized as follows. Section II describes the state of the art, reviews related work and introduces the standard IEC 61508, which requires fault insertion testing for safety related systems. In Section III the CPLD-based fail-safe system and its novel architecture are briefly described. The proposed approach to inject faults using a modified VHDL-description and a manual switch is described in Section IV. The procedure to verify the safety integrity measures of the fail-safe system is described in Section V. Section VI describes how synthesis results can be influenced using VHDL-constants. Section VII presents experimental results. Finally Section VIII concludes.

II. STATE OF THE ART

In the domain of industrial automation fault insertion testing is usually applied for the verification of microcontroller-based fail-safe systems. A fault can be injected into a microcontroller by modifying the executed program, which is usually implemented using C or C++. In this case the program can be modified with preprocessor commands, to inject a fault. Then the program needs to be recompiled and loaded into the microcontroller, which has to emulate the fault.

In contrast to microcontroller-based fail-safe systems the fail-safe system based on the novel CPLD-based architecture requires a new method to perform fault insertion testing, because these devices do not execute programs implemented in C or C++. To develop the presented method we reviewed related work and examined the relevant requirements of the standard IEC 61508.

A. Related Work

In [7] VHDL-based fault injection techniques are discussed. A saboteur is a component that is added into a VHDL-description to alter the timing characteristic or the value of a signal, if the saboteur is activated. During the normal operation

978-3-9810801-5-5/DATE09 © 2009 EDAA

of the system the saboteur is inactive. Saboteurs can be used to inject various fault types like stuck-at faults, bit-flips, bridging-faults or delay faults.

Mutants are components which replace corresponding components. If a mutant is activated, it behaves like the corresponding component in presence of a fault. If the mutant is inactive, it behaves like the fault-free corresponding component. A mutant can either be created by adding saboteurs to a structural model description, by replacing subcomponents of a structural model description or by modifying the syntactical structures of a behavioral description [8]. It is possible to inject various fault types like assignment control, stuck-else or stuck-then using mutants [7].

Fault simulations [2] can be used to analyze the behavior of fault tolerant circuits in presence of faults. In [3] the use of PLDs is proposed to accelerate a fault simulation. A PLD is connected to a host computer. The circuit, which has to be simulated, is entirely or partly mapped on the PLD. The host computer executes a simulation program, which applies input vectors to the circuit. The responses are read back by the host computer.

The authors distinguish between dynamic and static fault injection. In case of dynamic fault injection faults are injected during run-time. Multiplexers are inserted into the circuit to emulate stuck-at faults. The multiplexers contain two inputs. One input is connected to the correct input value and the other input is connected to a 1 or a 0 (stuck-at fault). The host computer controls the selector wire of each multiplexer. Thus it can determine if the correct input value or the stuck-at fault is switched to the output of the multiplexer. The advantage of this approach is that the circuit description has to be compiled only once. Also the PLD has to be configured only once. The disadvantage is that the multiplexers require additional PLD resources. Also the delay of the circuit increases, limiting the maximal clock speed.

An alternative approach is static fault injection. In this case faults are injected into the circuit during compile-time. Thus whenever the circuit has to be simulated using another set of faults, the circuit description needs to be recompiled and the PLD needs to be reconfigured. The disadvantage is that the frequent recompilation and reconfiguration increases the duration of the simulation. The advantage is that no additional multiplexers are inserted. Thus no additional PLD resources are needed and the delay of the circuit is not increased.

In [4] the use of various kinds of fault injection elements is proposed to inject faults. These elements contain either one or two test inputs. Depending on the test inputs either correct inputs or stuck-at-faults are switched to the outputs of the fault injection elements. The test inputs of the fault injection elements can be connected to the flip-flops of a fault injection scan chain or to the outputs of a decoder to inject faults.

In [5] the use of a single FPGA as fast simulation environment is described. A faulty version of a circuit and a fault-free version of the same circuit are emulated concurrently. A comparator compares the outputs of the faulty circuit to the outputs of the fault-free circuit. If the outputs differ, a fault has been detected. A LFSR (linear feedback shift register) is used to generate test vectors for the circuits. An additional state machine controls the fault simulation.

B. IEC 61508

The IEC 61508 [6] is a basic functional safety standard, which defines a safety life cycle. All activities from the initial concept, through specification, design, implementation, operation to the disposal of the system are covered. The standard IEC 61508 applies, when an E/E/PES (electrical/electronic/programmable electronic system) carries out safety functions.

This standard defines four safety integrity levels (SIL). While SIL 4 is the highest achievable level of safety integrity, SIL 1 is the lowest achievable level. The higher the safety integrity level of a system, the higher the probability that an E/E/PES performs its safety functions correctly if demanded.

Requirements for preventing failures during the development process and requirements for controlling failures, if they are present, are defined by this standard. Techniques and measures, which are necessary to reach a certain level of safety integrity, are specified.

The IEC 61508 requires fault insertion testing for the verification of safety related systems. Thus fault insertion testing is an important prerequisite for the successful certification of a system in adherence with the IEC 61508. The aim of fault insertion tests is to simulate faults in the system hardware [6]. Then the responses of the faulty system are analyzed. These kind of tests are used to assess the dependability of the system in case of a fault.

III. THE NOVEL CPLD-BASED FAIL-SAFE SYSTEM

The application domain of the presented fail-safe system is industrial automation. Assume an electric motor in a manufacturing plant, which is controlled by a PDS (power drive system). This PDS realizes a number of different standard functions, which should be able to stop the electric motor, if demanded. To reduce the risk of failing standard functions, corresponding safety functions have to be implemented. A safety function is able to stop the electric motor, even if the corresponding standard function fails.

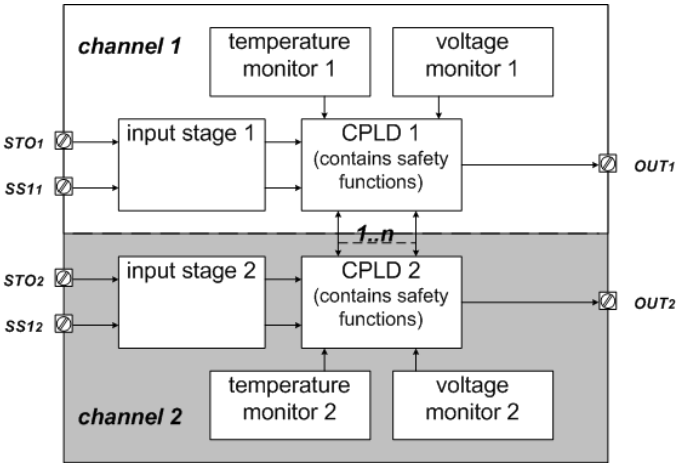

Figure 1. Architecture of the F-module

A. Architecture of the Fail-Safe System

The safety functions are realized using a separate module (F-module), which can be connected to the PDS. To communicate with the PDS, the CPLDs exchange messages with a DSP (digital signal processor), which is a component of the PDS. The architecture of the F-module is illustrated by Figure 1 schematically.

The architecture consists of two channels. Each channel is able to perform the safety functions independently. Thus the safety functions can still be carried out, if one channel is faulty. The F-module consists of the following components:

- **Input stages:** The input stages of the F-module are used to activate the safety functions. Each channel contains a separate input stage.
- **CPLDs:** The CPLDs carry out the safety functions, if demanded. Additionally the CPLDs realize measures to increase the safety integrity of the system. Thus they exchange a number of corresponding signal pairs. The CPLDs can deactivate the electric motor via the safety critical outputs (OUT_1 and OUT_2).
- **Voltage monitors:** There are separate voltage monitors, which are able to detect undervoltage and overvoltage. Additionally each channel is protected against dangerous overvoltage.
- **Temperature monitors:** One temperature monitor per channel checks, if the temperature is within the specified range.

B. Behavior of the F-module

The behavior of the F-module is illustrated by Figure 2. The system can enter a safe state and an unsafe state. The safe state can be divided into the substates start-up, hard-error and motor stopped. When the system is in the safe state, no power is applied to the electric motor. Thus the motor is not able to cause harm to people, the environment or property. When the system is in state motor running (unsafe), power can be applied and the PDS is able to control the motor.

When the F-module is switched on, the system enters state start-up (safe). In this state the CPLDs initialize all their registers and flip-flops. Both CPLDs receive parameters for a safety function from the DSP. Before the system leaves this state, the CPLDs have to perform an initial test, which ensures that the CPLDs are able to perform the safety functions correctly.

When the initial test is finished successfully by both CPLDs, they tell the PDS via the DSP, that state start-up can be left. With a defined DSP message the system enters state motor running (unsafe).

If a safety function is activated, the system enters state motor stopped (safe). If the safety functions have been performed properly and if they are not activated any more, the DSP can tell the F-module to enter state motor running again.

During the operation of the F-module various safety integrity measures are performed to detect faults. If a fault is detected, the system enters state hard-error (safe). This state cannot be left any more as long as the system is operating.

C. Safety Integrity Measures

To guarantee sufficiently high safety integrity, both CPLDs perform diagnostic checks to detect faults as long as the F-module is switched on. If a safety integrity measure detects a fault, the system enters the safe state. Various safety integrity measures to detect faults are realized:

- **Discrepancy monitors:** The CPLDs exchange corresponding signal pairs. Some of them contain information about the safety functions. If these signal pairs are discrepant for a specified time, a fault is signaled.
- **Init-test:** When the system is in state start-up, a test is performed, which verifies that both CPLDs are able to perform the safety functions correctly.
- **SS1-test:** When the Init-test is finished, the CPLDs perform a periodic background test. Components, which are relevant for the correct execution of the safety functions, are tested.
- **Shut down test:** The CPLDs get information about the state of the electric motor via feedback signals. If the deactivation of the motor does not occur in a specified time, a fault is signaled.
- **Short circuit test:** This test can detect short circuits between the safety critical outputs of the F-module.
- **Temperature monitor:** If the temperature of the environment exceeds a certain limit, the system enters the safe state.
- **Voltage monitor:** If the supply voltage of the CPLDs is too high or too low, the system enters the safe state.

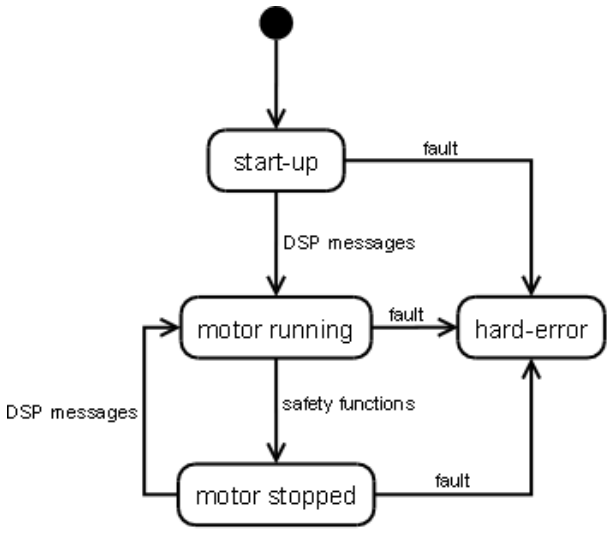

Figure 2. Behavior of the F-module

IV. Fault Insertion using System Hardware

The IEC 61508 requires fault insertion tests to be performed using the system hardware. That is why it is not sufficient to perform fault insertion tests using software simulations. Instead we use the system hardware (F-module) to perform the tests.

The behavior of the CPLDs on the F-module is defined by a synthesizable VHDL-description. After the compilation of the VHDL-description a programming file is available, which can be used to configure each CPLD via a JTAG interface. Usually both CPLDs are configured using the same programming file. Nevertheless it is possible to configure the CPLDs using different programming files. The presented method to perform fault insertion testing exploits this fact.

To inject a fault we create a modified version of the synthesizable VHDL-description. This modified description makes it possible to inject a certain permanent or transient fault into one CPLD via an input pin of this CPLD. After compilation a modified programming file is available. We configure one CPLD (fault emulating device) with this modified programming file. The other CPLD (golden device) is configured with the original programming file. Thus it is possible to inject a fault into one CPLD but not into the other one.

One input pin of the fault emulating device is connected to a manual fault enable switch. When the input pin is not switched to V_{DD}, no fault is injected and the fault emulating device has the same behavior as the golden device. If the input pin is switched to V_{DD}, a fault is injected into the fault emulating device.

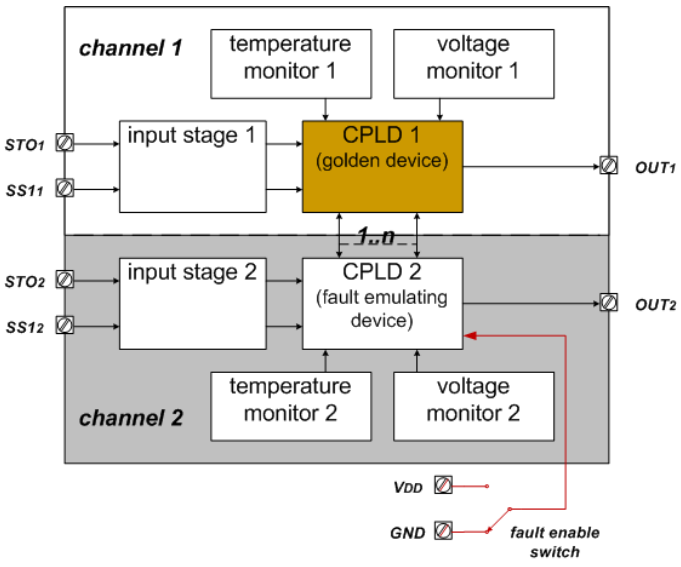

Figure 3. Fault insertion using the F-module

To modify the VHDL-description saboteurs can be added. This allows the injection of transient faults like bit-flips. Also permanent and intermittent faults like stuck-at faults, bridging-faults and delay faults can be injected. Also mutants can be added to the VHDL-description to inject faults. Saboteurs and mutants can be activated and deactivated via the fault enable switch. Various fault models can be considered using the presented approach. The only restriction for the creation of saboteurs and mutants for this approach is the use of synthesizable VHDL-constructs.

Figure 3 illustrates the F-module, which is prepared for fault insertion testing. In this case CPLD 1 is configured as golden device and CPLD 2 is configured as fault emulating device. Generally it does not matter which CPLD is configured as golden device and which CPLD is configured as fault emulating device.

V. Verification of Safety Integrity Measures

According to the requirements of IEC 61508 the synthesizable VHDL-description is modular and consists of a number of interconnected components. We perform a system FMEA (failure modes and effects analysis) and a design FMEA to determine possible sources of failure of the components and their interconnections and identify the consequences in terms of system behavior.

If a fault of a component or interconnection leads to the safe state of the entire system, the fault can be considered to be safe. In this case no fault insertion test case needs to be planned. If a fault potentially leads to the unsafe sate of the entire system, we expect the implemented safety integrity measures to detect the fault and the system has to enter and/or maintain the safe sate. Consequently a test case needs to be planned to verify that the safety integrity measures are able to detect the fault.

It is necessary to verify, that each of the two channels is able to detect injected faults. Thus CPLD 1 is configured as golden device for one half of the test cases, while CPLD 2 is configured as golden device for the other half of the test cases.

For every test case another fault has to be injected into the fault emulating device. Consequently another modified programming file needs to be generated and the fault emulating device needs to be reconfigured for every test case. Thus a new modified synthesizable VHDL-description has to be created. For every test case we insert a saboteur between components or create a mutant of a component of the VHDL-description. This allows injecting a single fault into the fault emulating device, while the golden device remains fault-free. Considered types of faults are stuck-at, bit-flip, stuck-then, stuck-else, assignment control and stuck-data. If the entire system (golden device in cooperation with the fault emulating device) detects the fault and consequently enters and/or maintains the safe state, the test is successful. Otherwise the test fails.

The synthesizable VHDL-description can be parameterized before compilation. The parameters are constants which are defined in a separate VHDL-package. There is one constant per fault insertion test case. If all constants are set to 0, it is not possible to inject a fault into the CPLD after compilation and configuration via the fault enable switch. The synthesized circuit contains no fault injection logic. Thus in the VHDL-description for the golden device, all constants are set to 0. After compilation and configuration not a single logic element is required for mutants or saboteurs in this case.

If the constant for a certain fault injection test case is set to 1, it is possible to inject a corresponding fault after compilation and configuration using the fault enable switch. In this case the synthesized description of the fault emulating device contains a saboteur or a mutant, which can be activated via the fault enable switch. Thus for every fault insertion test case a single constant is set to 1 in the VHDL-description of the fault emulating device. The remaining constants are set to 0. Consequently there is only a little area overhead caused by a little fault injection logic required for a single saboteur or a single mutant for each test case.

There is no guarantee that the mapping of the modified VHDL-description on the logic elements of the CPLD is similar to the mapping of the unmodified VHDL-description on the logic elements. That is why the presented approach is limited to systems which contain at least two channels where each channel contains a CPLD or FPGA, which can be configured independently. There must always be at least one channel that contains a CPLD or FPGA that is configured with the compiled, unmodified VHDL-description.

For every test case a certain fault is injected, when the system is in one of two states. In the first case a fault is injected, when the F-module is in state start-up and in the second case a fault is injected, when the F-module is in state motor running. A test case is successful if the F-module either does not leave state start-up (safe) or if it enters and maintains the safe state.

Following steps are necessary to verify all safety integrity measures:

1. Perform systematic FMEAs on system- and design-level to identify necessary test cases
2. Define expected results for each test case
3. Repeat for every test case
    a. Insert a saboteur into the VHDL-description or create a mutant
    b. Add a VHDL-constant to be able to enable or disable the test case before compilation
4. Repeat for every test case
    a. Edit VHDL-constants to enable injection via the appropriate saboteur or mutant
    b. Compile modified VHDL-description
    c. Reconfigure the fault emulating device
    d. Inject fault when system is in state start-up
    e. Restart to inject fault in state motor running
    f. Decide if test was successful
    g. Analyze and document result

The effort for all fault insertion test cases is dominated by steps 1 and 2, which consume much time for analysis and documentation. The time required for inserting saboteurs or mutants, editing VHDL-parameters, compiling the VHDL-description and reconfiguring the fault emulating device is comparably small.

VI. SYNTHESIS

An example demonstrates how the synthesis results can be influenced by modifying VHDL-constants, to make fault insertion possible. Figure 4 and Figure 5 illustrate schematics of simple combinational circuits, which were synthesized. The circuits have an output that depends on the inputs A_INPUT, B_INPUT, C_INPUT and D_INPUT. If all constants in the corresponding VHDL-package are set to 0, the FAULT_ENABLE pin is not connected to the saboteurs (simple fault injection multiplexers for stuck-at faults). Thus the saboteurs always switch the correct input value to the output (Figure 4). Consequently these saboteurs require no CPLD-resources due to optimizations of the tool, we use for synthesis and fitting.

If the first constant in the corresponding VHDL-package is 1, while the second constant is 0, it is possible to inject a stuck-at-1 fault between the output of the AND-gate A_AND_B and one input of the OR-gate. In this case the FAULT_ENABLE pin is connected to a saboteur. Thus it is possible to inject a stuck-at-1 fault using the fault enable switch (Figure 5). Consequently the synthesized circuit requires an additional logic element.

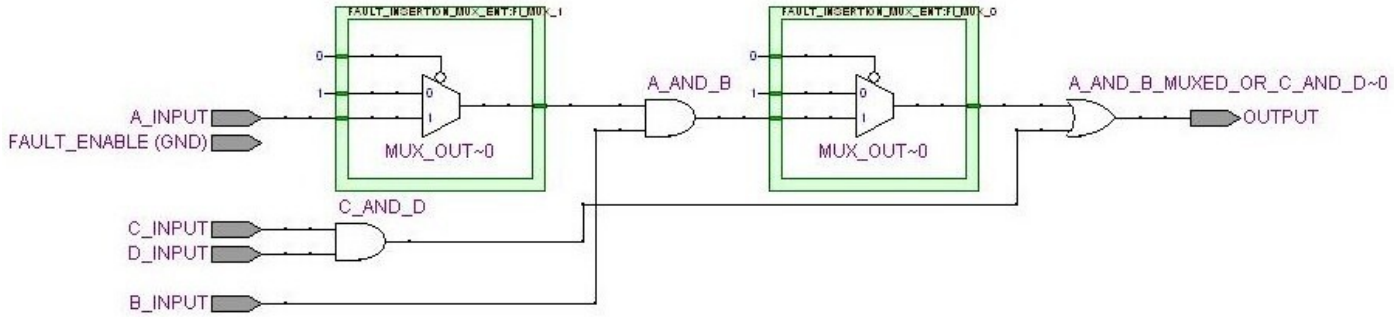

Figure 4. No fault can be injected

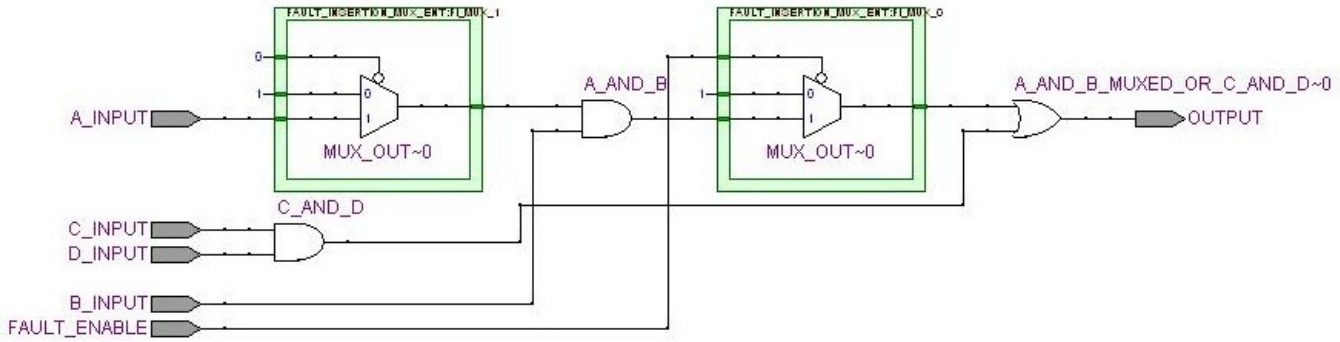

Figure 5. A stuck-at-1 fault can be injected between the AND-gate and the OR-gate

VII. Experimental Results

The compiled description requires approximately 95% of the 240 logic elements of the used type of CPLD [9]. There remain only a few logic elements, which can be used for fault injection logic. Thus we performed a number of test cases using mutants and saboteurs to prove that the proposed method is applicable. On average the synthesized circuit required 1.72 additional logic elements (less than 1% of the device resources) due to additional fault insertion logic. Although our design already requires approximately 95% of the available logic elements, the method is still applicable.

VIII. Conclusion

The presented method allows the application of a proven verification method (fault insertion testing) for a novel CPLD-based fail-safe system. Methods to verify comparable microcontroller-based systems using fault injection are not applicable for the fail-safe system, because it contains exclusively CPLDs. It is possible to verify the safety integrity measures of the fail-safe system using the system hardware (F-module). The successful verification of the realized safety integrity measures using fault insertion testing is an important prerequisite for the certification of the developed CPLD-based fail-safe system in adherence with the standard IEC 61508.

TÜV SÜD assessed a plan for validation and verification of the fail-safe system. This plan also describes the presented method to perform fault insertion testing. TÜV SÜD stated that the presented method is appropriate.

References

[1] P. Sundaram, and J.G. D'Ambrosio, "Controller integrity in automotive failsafe system architectures," SAE Transactions, 2006, vol. 115, pp. 370–377.

[2] M. Abramovici, A.D. Friedman, and M.A. Breuer, Digital Systems Testing and Testable Design. IEEE PRESS. 1994.

[3] W.L. Gallagher, H.H. Yao, and E.E. Swartzlander Jr., "Fault simulation with PLDs," In Proc. of the 31st Asilomar Conference on Signals, Systems & Computers, 1997, pp. 411–415.

[4] Shih-Arn Hwang, Jin-Hua Hong, and Cheng-Wen Wu, "Sequential circuit fault simulation using logic emulation," IEEE Transations on Computer-Aided Design of Integrated Circuits and Systems, 1998, vol 17, pp. 724–736.

[5] P. Ellervee, J. Raik, K. Tammemäe, and R. Ubar, "Environment for FPGA-based fault emulation," In Proc. of the Estonian Academy of Sciences. Engineering, 2006, vol 12, pp. 323–335.

[6] IEC 61508, Functional safety of electrical/electronic/programmable electronic safety-related systems.

[7] J. Gracia, J.C. Baraza, D. Gil, and P.J. Gil, "Comparison and. application of different VHDL-based fault injection techniques," In Proc. of the International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'01), 2001, pp. 233-241.

[8] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, "Fault injection into VHDL models: the MEFISTO tool.," In Proc. of the 24th International Symposium on Fault-Tolerant Computing (FTCS-24), 1994, pp. 66-75.

[9] P. Leventis, B. Vest, M. Hutton, and D. Lewis, "MAX II: A low-cost, high-performance LUT-based CPLD," In Proc. of the Custom Integrated Circuits Conference, 2004, pp. 443–446.

[10] G. Grießnig, C. Steger, and R. Weiß, "CPLD basierende homogen redundante fehlersichere Architektur," In Proc. of the Informationstagung Mikroelektronik (ME08), 2008, pp. 201–205.