Dynamic fault injection into digital twins of safety-critical systems

Thomas Markwirth, Roland Jancke, Christoph Sohrmann
Fraunhofer IIS/EAS
Dresden, Germany
{Thomas.Markwirth, Roland.Jancke, Christoph.Sohrmann}@eas.iis.fraunhofer.de

***Abstract* — In this work we present a technology for dynamically introducing fault structures into digital twins without the need to change the virtual prototype model. The injection is done at the beginning of a simulation by dynamically rewiring the involved netlists. During the simulation on a real-time platform, faults can be activated or deactivated triggered by sequences, statistical effects or by events from the real world. In some cases the fault structures can even be auto-generated directly from a formal specification, which further automates the development process for safety-relevant systems. The approach is demonstrated at a SystemC/ SystemC AMS virtual prototype of a safety-critical sub-systems which runs on a dSPACE real-time hardware.**

Keywords—Digital Twin, Safety Critical Systems, Fault injection, SystemC, Verification, Validation, HiL

I. INTRODUCTION

Today's embedded systems often exhibit a great deal of complexity. Especially for safety-critical systems the efforts for system validation are huge. One of the objectives of the ISO standard 26262 is the impact analysis of faults on the system behavior. Performing these tests only at the end of the design phase might be too late. Early simulation-based testing on system level offers a solution. Simulations are performed at an early point in the design process, possibly already in the conceptual phase, but also during later stages of the entire design flow. For this purpose, system models are used in the form of executable specifications. They serve as a reference for the implementation as well as a virtual prototype for the development of firmware and application software. These are typical use cases of system models since they support the design process of products significantly.

The concept of digital twins picks up this aspect and augments the approach in the sense that digital twins are not only a tool for design support, but rather represent the product itself, therefore being a part of it. As a consequence, the life span of such a digital twin lasts during optimization at the product phase as well as over the complete product life cycle. At the same time they offer new possibilities of application, for instance the construction of entire supply chains by passing on the digital twin to partners and customers. In consequence, this allows for virtual design and test of complex systems without having to disclose in-house know-how and IP. Furthermore, the use of digital twins permits virtual examination of the final products, which would be either too complex, too dangerous, or technically impossible to be done on the real system.

Special focus is on injection of faults provoking potential critical system states. This makes it possible to study the impact of faults, to test safety concepts and therefore to evaluate the robustness of a system. Therefore digital twins are highly interesting for safety-critical products. However while creating digital twins for safety-critical systems, a variety of challenges need to be solved. One of the most critical challenges is to achieve a sufficiently high simulation performance to handle the complexity of the examined system. At the same time the level of abstraction needs to be chosen constructively to cover the relevant details.

In general, classical approaches to fault injection for evaluating safety concepts can become very tedious since all faults have to be integrated into the respective simulation models directly, in addition to the fault switching mechanisms for activating them. This in turn leads to the issue of mixing functional modelling and modelling of test functionalities. Functions for fault injection as well as the faults itself, however, are part of the test functionalities and should therefore not be element of the DUT implementation. According to this, in the following we will present an approach for dynamical fault injection into digital twins that solves this dilemma and is therefore easily applicable for safety-critical systems.

For the creation of a digital twin it is essential to use an efficient language for modelling and simulation. From the author's perspective, this is best accounted for by the SystemC/ SystemC AMS language. Modelling of a digital twin usually is done in an iterative process which starts with a specification and a concept phase. In order to support the respective use case, it is convenient to provide model implementations on multiple abstraction levels for each system component and function. To guarantee not only efficient flows but also consistency with the requirement documents, it is convenient to deploy an automatized generation of testbenches, fault injections and model conversions.

II. GENERAL METHODOLOGY

The focus of this paper is the introduction of an approach for dynamical fault injection based on requirement documents for the validation of safety concepts for safety-critical systems. Thereby a flow will be sketched including:

a) requirement management containing test scenarios for evaluation of safety concepts,
b) a document describing the faults that are to be injected,
c) the simulation environment, and
d) dynamical fault injection into the digital twin.

In [1] a new methodology for dynamical fault injection was presented, which allows fault injection into arbitrary pre-existing SystemC/SystemC AMS models during runtime. For this, the faults are not part of the DUT model but part of the testbench instead. Faults are only injected during the execution of test cases for fault simulation, while the DUT can remain in its faultless form. This strongly simplifies the handling of the DUT and its different fault modes, since the nominal design and the failure modes can be stored separately. For very complex, hierarchical designs, the injection of different fault types and multiple faults scattered over the entire design, our approach is the only feasible solution. Often,

fault injection is done during the design phase, where the design still changes rapidly. In this paper, we will give a brief summary of the methodology. For more details on the underlying approach we refer the reader to [1]. Figure 1 shows the general principle of our injection approach.

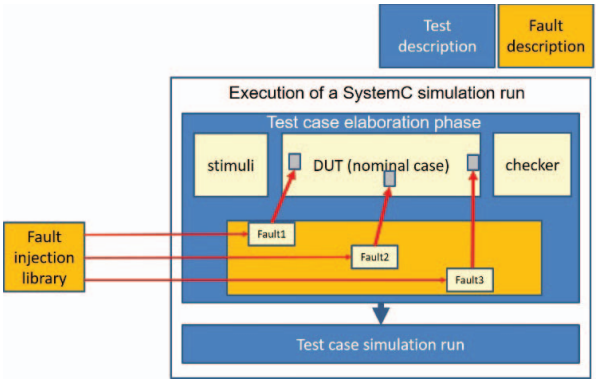

Figure 1: Principle of the dynamic fault injection approach

During the elaboration phase of the fault simulation, runtime in-memory connections in the netlist are disconnected and new structures are inserted which allow to switch between faultless and faulty behavior. This methodology is applicable for all MoCs in SystemC/SystemC AMS, while the respective MoC is detected automatically and the corresponding injection structure is used. For an easy application by the user, a model library is available. This model library contains a range of predefined fault models and user templates as well as functions for easy usage and a wide spectrum of configuration possibilities.

III. Fault Injection in the Application Context

In the following, the dynamical fault injection will be demonstrated on a battery management system (BMS) model as an example of a highly safety-critical system.

A. Overview of the BMS

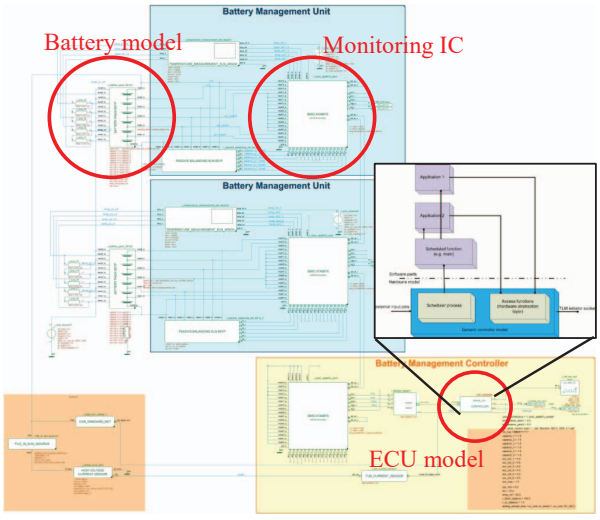

Figure 2: Toplevel of the digital twin of a safety-critical BMS

Figure 2 shows the top level view of the BMS twin which is designed as a heterogeneous system model. The battery cells are physical models on the basis of measurements on real cells. In the digital twin, they are permanently monitored regarding cell voltages and package temperatures by a chain of monitoring ICs. In addition, these ICs are used for the activation of balancing and bypassing. They are connected with the ECU model through a standard interface. The software applications embedded into this model are taken from a real battery management ECU.

Furthermore, the ECU model measures the load current which is provided by an current metering model. Thereby the load current is taken from a real driving cycle. The loads of the system model contain uncritical as well as safety-critical consumers of an fictitious board net.

B. The necessity for a safety concept

In principle a variety of potentially safety-critical faults with possibly severe consequences can occur. The designer of safety-critical systems therefore is responsible for examining which faults have a high probability of occurrence as well as impact. On this basis, corresponding counter strategies have to be developed and evaluated.

Examples for potentially safety-critical faults in a BMS include:

- Battery cells run out of specified voltage reach values outside of specified temperature ranges
- Communication channels between monitor ICs and the software of the safety concept are disturbed or broken
- Bypass switches are damaged

On closer inspection, the latter two faults concern a double fault, since they only become critical in case of an additional fault such as the first one. When it comes to inspecting safety concepts, the examination needs at first to focus on the impact of single faults. The effort of examining multiple faults quickly becomes impractical. Therefore it is reasonable to verify the single fault case "*Battery cell is running out of the specified range*" first.

Now the safety concept has to guarantee compliance with the safety requirements at any time. However, some reaction time may be allowed. In fact, additional criteria are defined, allowing for a short-time violation without risking uncontrollable states. The ISO gives clear specifications on this, as shown in Figure 3.

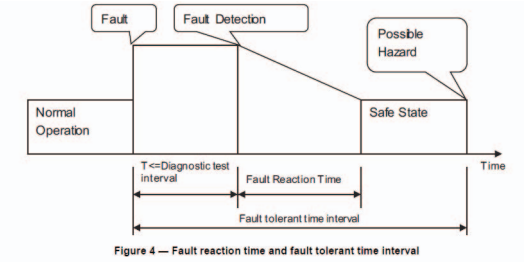

Figure 3: Fault reaction time as specified in the ISO 26262 standard

For example a "*fault tolerant time interval*" has to be defined. Inside of this time interval a fault status shall be detected as well as its implications be resolved, leading the system back to a safe state. Only then the safety criteria is fulfilled. Referring to our example, a global safety goal could serve for excluding an overheating of battery cells with the possibility of fire or explosion. The safety concepts has to guarantee that the sum of "*fault diagnostic time*" und "*fault reaction time*" is less than the "*fault tolerant time interval*". This can be controlled to some extend by the "*diagnostic test interval*", i.e. the time period for testing the respective fault. In order to meet the requirements, this interval must not be too long.

In general it is not sufficient to elaborate and implement safety concepts. They need to be tested as well. The most efficient way for doing so is by using a digital twin in combination with an approach for dynamical fault injection. With this approach many fault states due to single or multiple faults can be tested.

Because of the high simulation performance of the digital twin, even complex regressions can be performed within a reasonable time.

C. *From requirements to faults*

Usually there are different possibilities of how to inject faults into a DUT. For instance, for electrical battery faults, one can switch virtually between the faultless battery cell model and an externally controlled voltage source. The original connection between battery cell and connected node gets disconnected. Now switchable resistors are inserted, allowing for reproduction of the original cell voltage in the deactivated fault case while bridging the cell with a voltage source connected in parallel in the activated fault case. This is illustrated in Figure 4.

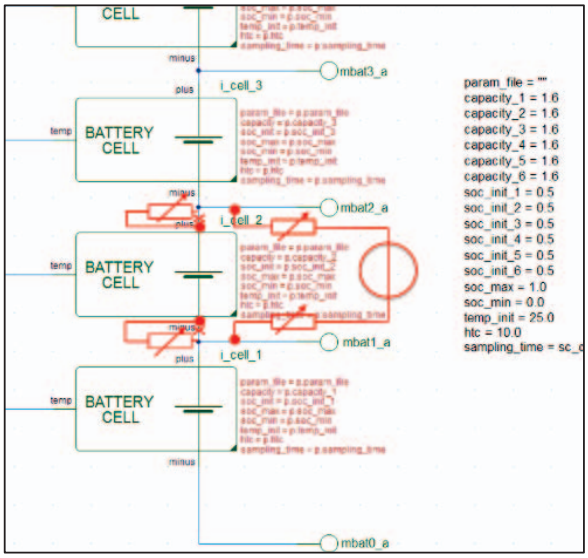

Figure 4: Visualization of the bridging fault injection principle into battery

It should be noted that the red part of the netlist is generated only in memory during the simulation run. For the description of this fault, there exists a separate form in the requirement document which is mapped to the specific test case, as indicated in Figure 5.

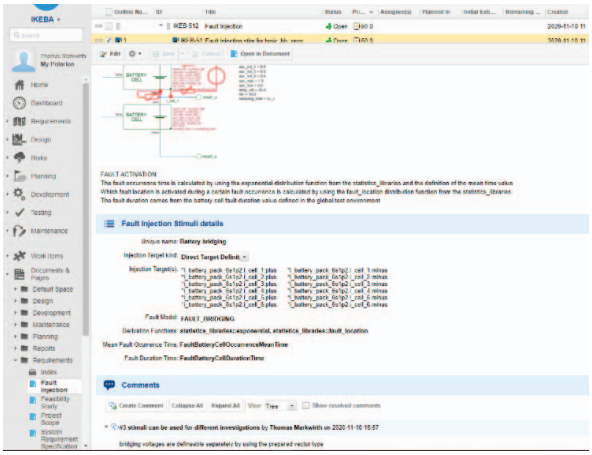

Figure 5: Definition of the battery fault in the requirement document

The source code of a SystemC module can be automatically generated from the requirement description, as shown in Figure 6. This module is instantiated only at execution of this particular test case, while for any other test case the instance is not generated.

```
// constructor
battery_bridging_fault_injection(sc_core::sc_module_name nm, params pa = params() ) : p(pa)
{

    // definition of fault injection targets
    port.push_back("*i_battery_pack_6s1p2.i_cell_1.plus");
    port.push_back("*i_battery_pack_6s1p2.i_cell_2.plus");
    port.push_back("*i_battery_pack_6s1p2.i_cell_3.plus");
    port.push_back("*i_battery_pack_6s1p2.i_cell_4.plus");
    port.push_back("*i_battery_pack_6s1p2.i_cell_5.plus");

    port_b.push_back("*i_battery_pack_6s1p2.i_cell_1.minus");
    port_b.push_back("*i_battery_pack_6s1p2.i_cell_2.minus");
    port_b.push_back("*i_battery_pack_6s1p2.i_cell_3.minus");
    port_b.push_back("*i_battery_pack_6s1p2.i_cell_4.minus");
    port_b.push_back("*i_battery_pack_6s1p2.i_cell_5.minus");

    // vector of fault models instances
    szenario1 = new fault_scenario_template<double >(std::make_pair(port, port_b), fault_injection_base::FAULT_BRIDGING);
    szenario1->stat_failure_time_function   = exponential;
    szenario1->location_function            = fault_location;
    szenario1->set_mean_fault_occurrence(   sc_core::sc_time(5000.0, SC_MS) );
    szenario1->set_fault_duration(          sc_core::sc_time(5000.0, SC_MS) );
```

Figure 6: Code snippet of the fault injection stimuli module

When the constructor is called during the elaboration phase, the signals get disconnected, the fault models get instantiated and the switch functionalities are inserted automatically. For doing so, the injection targets as given in the requirements document are used, for example via their hierarchical paths starting from the top level. It is also possible to pass in target vectors, leading to the creation of as many fault model instances simultaneously. As an alternative to fixed injection targets, the design can be scanned for predefines characteristics where an injection is done for every match. Further details of this approach can be found in [1].

Figure 7 shows a comparison of results of simulations with either nominal, i.e. faultless, or faulty, i.e. with injected faults, state. It can be clearly seen how the voltage of single a battery cells is reduced as the result of fault activation. Subsequently, an undervoltage is detected by the monitor IC and forwarded to the controller. The software application containing safety mechanisms is capable of recognizing faults and respond to it with a small delay.

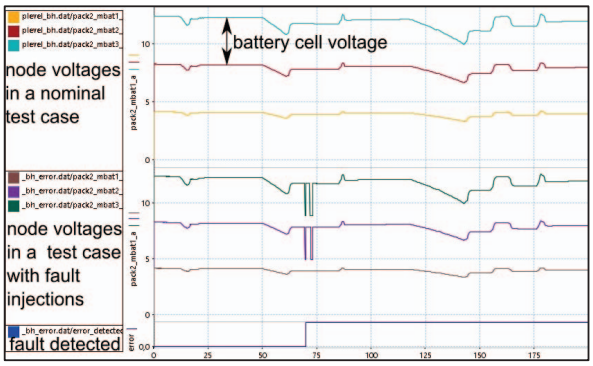

Figure 7: Comparison of faulty and faultless simulation results

An injection of a fault structure does not automatically imply the activation of the corresponding fault. For doing so, there also exist various options. In the test case, fault activation functions can be called at specific points in time. The fault injection control also allows periodic activations in the form of sequences. By including a statistics library [3], even pseudo-random activations become possible. The time as well as the location of a fault activation can be chosen randomly. User-specified distribution functions can be used to select fault location and fault time. Moreover, the mean value for the time distribution and a fixed value for the fault duration can be defined. Reproducible results are assured by the use of the same seed values. Figure 8 shows the simulation results of the same test case when using different seeds. As a result, varying values for the activation of faults in the simulation process are obtained. This approach enables a highly realistic analysis.

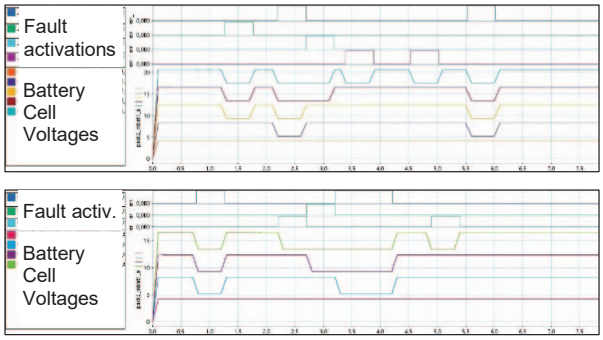

Figure 8: Comparison of random fault activation for different seed values

Additional safety goals for the battery management system are related to safety-critical components such as steering or braking systems. It is absolutely mandatory to supply a stable voltage to these systems at any time during driving, even in case of main battery failure.

A safety requirement derived from this safety goal states that whenever the supply voltage drops below a critical level for a pre-defined time, the critical consumers shall be disconnected from the battery and instead be supplied by a backup or emergency battery. This is shown in Figure 9 as an example board net. The input terminal *vbat* is connected with the battery model of Figure 2. It contains line impedances as well as several load impedances which are uncritical with regard to the system safety. Another load impedance whose failure is possibly critical for the system function needs to be supplied with voltage under all circumstances.

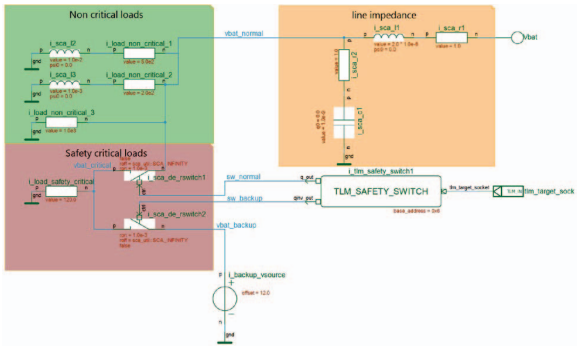

Figure 9: Board net example with critical and non-critical loads

A safety requirement contains the activation of an emergency or backup battery in this case. The corresponding ECU application needs to monitor the supply voltage permanently, while short-term threshold crossing have to be ignored. However, the emergency battery needs to be activated in the range of the "*fault tolerant time interval*" defined for the safety goal in the requirements document.

The Figure 10 shows the simulation result of a test case which serves as verification of the safety requirements. It shows a strong drop of the supply voltage at the safety-critical consumer as a result of the battery fault activation. After fault detection by the safety mechanism and activation of the emergency supply, the safety-critical system component is supplied in the tolerated time interval such that the overall system can be transferred into a safe state.

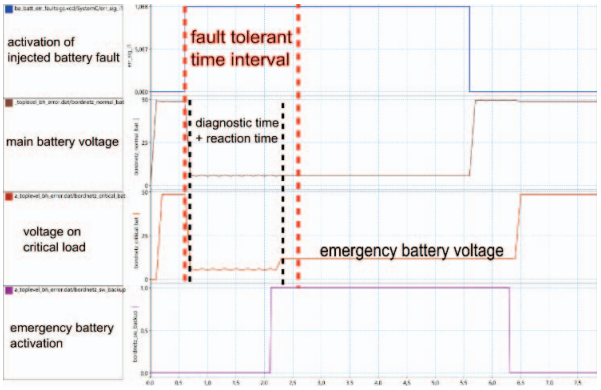

Figure 10: Simulation result for evaluation of fulfilled critical load safety goal

D. Final validation of the physical ECU

One differentiating feature of a digital twin is its application during the entire product life cycle. Therefore safety functions have to reexamined after every product patch or consumer adjustment. This is not only requested by the requirement management but also mandatory for the evaluation of safety concepts which are expected to reduce the impact of safety-critical events on the system behavior. Even though many tests are performed as simulations when using a digital twin, it is unavoidable to carry out corresponding tests also on the real system. Even here, the digital twin can support the process. However, for this type of application the digital twin requires real-time capability. In that case, it serves as a virtual

environment of the real system. An example of such a scenario is shown in Figure 11. Here, the control unit was separated from the digital twin of the battery management system and included into tests as a real-world component.

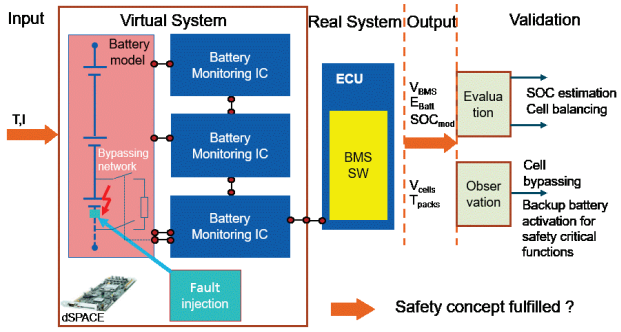

Figure 11: HiL scenario for the BMS, based on digital twin connected to real BMS-ECU

The virtual part of the overall system is running on a real-time platform. In our case dSPACE real-time hardware has been used. By the approach for dynamical fault injection outlined in section III.C, it is possible to trigger critical fault states even in such a test. For this, the generated test cases for the digital twin of the overall system is reused. This allows to obtain comparable results, reduces the test effort and provides consistency in the complete verification process. With these test cases, the safety mechanisms running on the real-world control unit can be tested realistically.

Also in this case the critical battery state has to be detected during the "*fault tolerant time interval*". In addition, the bypass for the corresponding cell hast to be activated, in order to resolve the critical state and to fulfill the safety goals.

IV. Outlook

Future work on this topic will be focusing on the design of a convenient, fault-tolerant generation flow from requirement document up to emulation.

V. Acknowledgements

This work has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 876852.
The JU receives support from the European Union's Horizon 2020 research and innovation programme and
Austria, Czech Republic, Germany, Ireland, Italy, Portugal, Spain, Sweden, Turkey.

References

[1] T. Markwirth, P. Ehrlich, and D. Matter, "Dynamic Fault Injection Library Approach for SystemC AMS", DVCon Europe, October 2016

[2] C. Ziebert, A. Melcher, B. Lei, M. Rohde, H.J. Seifert, M. Gulbins, T. Markwirth, J. Haase, "A Table-driven Li-Ion Battery Model for a BMS Development Platform – Modeling, Measurements, Implementation and Validation", European Battery Hybrid and Fuel Cell Electric Vehicle Congress, Geneva, March 2017

[3] T. Markwirth, J. Haase, K. Einwich; "Statistical Modeling with SystemC-AMS for Automotive Systems", FDL'08; 2008

[4] G. Pachiana et al., "Automated traceability of requirements in the design and verification process of safety-critical mixed-signal systems", DVCon US 2021, to be published