Combining Graph-based Guidance with Error Effect Simulation for Efficient Safety Analysis

Jo Laufenberg*,Sebastian Reiter†,Alexander Viehl†,Oliver Bringmann*,
Thomas Kropf*, Wolfgang Rosenstiel*

* Universität Tübingen
[laufenbe, bringman, rosenstiel, kropf]@informatik.uni-tuebingen.de

† FZI Forschungszentrum Informatik
[sreiter, viehl]@fzi.de

***Abstract*—The increasing number of complex embedded systems used in safety relevant tasks produces major challenges in the field of safety analysis. This paper presents a simulation-based safety analysis that will overcome these challenges. The presented approach consists of two parts: an Error Effect Simulation (EES) and a graph-based specification. The EES is composed of a system simulation with fault injection capability and a generic fault specification. The graph-based specification approach guides systematically the EES and enables a very efficient exploration of the analysis space. Inherent in the graph-based specification is the documentation of the safety analysis and a coverage approach to assess the executed safety analysis. Combining these parts leads to an efficient and automatable framework for safety analysis. A use case of an interconnected electronic control system shows the application of the approach and highlights the benefits for a safety analysis, such as a failure mode and effect analysis.**

I. Introduction

In the last decades the amount of software and embedded systems has increased exponentially. In the automotive domain software has become the driving factor for innovations. Current premium segment cars include more than 70 connected interacting embedded platforms. The same trend can be seen in other domains, where amount, complexity and interaction of electronic-based systems are rapidly increasing. These systems are often involved in safety relevant tasks, like active breaking intervention. With this significant increased complexity the probability of operational errors is increasing too. An erroneous delivered service of these safety relevant applications could result in disastrous accidents. Therefore a safety assessment of the overall system is an obligatory task. The complexity of the complete system and all its variability has to be handled.
In the area of system verification, simulation-based assessments have been established. On system-level they are used to verify functional, timing and power requirements [1]. These so called virtual prototypes (VPs), behavioral models of the system to be verified, provide a promising approach for safety assessment [2]. They characterize the interdependency between components and the explicit and inherent error tolerance of the system.
This paper presents a comprehensive approach for safety analysis based on system simulations. The work covers a modularized, configurable system simulation approach extended with fault injection capabilities. The simulation framework supports both design decisions in early phases with regard to multiple system variations and detailed analysis with integrated software prototypes in later phases. An integral fault description enables the injection of diverse fault behaviors and therefore supports a variety of abstraction levels. The usability of the Error Effect Simulation (EES) is significantly enhanced by a graph-based specification approach, which is used to specify, control and supervise the complete safety verification as well as each single EES. This enables to cope with the increasing complexity caused by the combination of multiple system alternatives, faults and system environments. In early design phases a variety of system alternatives and parametrizations are specified with the help of a graph-based format and automatically explored by the framework. In later phases a comprehensive safety assessment is executed, with regard to manifold fault cases in different combinations and locations. The complete set of evaluations is guided by a graph-based specification, enabling an easy quantitative assessment of the overall system safety. Fig. 1 depicts the presented framework.
The remainder of this paper is structured as follows. After an overview of existing work the Section III introduces the used EES framework with a generic fault specification approach. Section IV shows the graph-based specification of a verification plan. A close integration of the graph-based specification within the EES, enabling a fine-grained control of the simulation-based safety analysis is presented in Section V. An industrial use case is presented in Section VI.

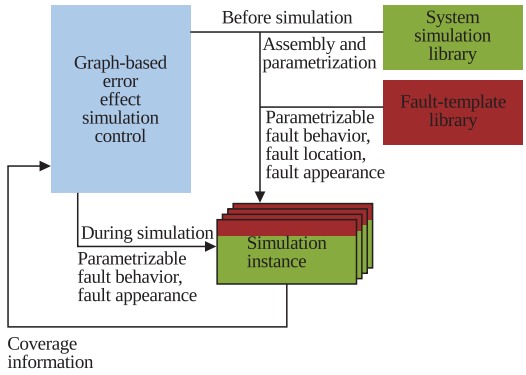

Fig. 1. Graph guided error effect simulation

II. Related Work

In industry functional verification is a widely used approach for testing systems [3]. Simulation-based techniques are part of the dynamic functional verification approaches and are the leading technique. In the development process verification plans are used to specify which components are verified and how [4]. These plans are often formulated in natural language with a high level of abstraction, given a wide scope of interpretation and limited potential for automation. To overcome these lacks, a graph-based approach is used, which is more restrictive to the scope of interpretation and gives a higher support to automation. Industrial applications like Trek designed by Breker [5] use such graphical descriptions to derive test cases.
Fault injection as part of the dynamic verification is an important and established approach for safety analysis. It targets hardware fault injection, software fault injection and simulation fault injection (SFI). Considering only SFI there exits already a variety of approaches. The group of non-invasive approaches [6], [7] use compiler or simulator modifications to inject faults. Our approach should be simulator and compiler independent to enable the usage of third party tools. Another category of approaches [8], [9], [10] use the so called saboteur method. They modify the structure of the model and introduces additional modules that inject faults in the data path. Although the change within the simulation structure

978-3-9815370-6-2/DATE16/

is best suited with our configuration approach, we did not use this approach because of its limitation to inject faults only in the exchanged data. Functional, abstract simulations often provide a monolithic structure that is not suited with the saboteur approach. Additionally the non-functional properties are most likely specified within the internal state. Therefore we chose the mutation approach. The references [11], [12], [13] present different mutation approaches. Especially the approach in [12] is similar to the presented injection technique. Our main contribution is in the specification of the fault behavior. None of the mentioned approaches has a comprehensive fault specification approach. Most approaches use a fixed injection behavior in the injector, a predefined set of fault behavior with simple trigger conditions such as time dependent fault triggers. A comprehensive fault specification approach, where both the injected value and the fault trigger depend on the current state of the system simulation is not presented.
This shortcomings lead to the development of our own injection and simulation framework. Combining this framework with the graph-based analysis specification provides a holistic approach for safety analysis that is not targeted by previous works.

III. Error Effect Simulation

EES enables the execution of what-if analyses in presence of faults and safety mechanisms. One focus of the presented framework is the applicability across the design process and for various levels of abstraction. Ranging from the exploration with abstract models to the detailed analysis with the actual Software-prototypes. To reduce the overhead the EES has to reuse already specified system simulations as much as possible. The system simulation consists of a library of parameterizable modules that are interconnected to create a simulation instance, similar to the approach in [14]. By using these basic building blocks, it is possible to create different system configurations by just changing the assembly, e.g. to exchange the used communication bus. Based on the parametrization of the basic building blocks it is possible to extend the system exploration space by changing the module parameters, such as the clock frequency of a microcontroller. The assembly and the parametrization are applied at simulation runtime by an IP-XACT configuration file. This enables the simulation of multiple system configurations without the need of re-compilation, reducing the analysis effort.
Besides the modular, configurable structure of the system simulation, EES provides an infrastructure to inject faults within the system simulation. The fault injection infrastructure provides injection and monitoring probes that are added to the system simulation. The injection probes provide an interface to change the value of the associated simulation variable, by an injection control module. The injection probes are added to the system simulation by replacing data types with an injectable probe data type. This requires a modification of existing simulation models. By using a template based data type structure that reflects the calling conventions of the reference counting `shared_ptr` of C++11, the changes are mostly limited to replacing the variable declaration. Behavioral modifications of the simulation modules are not required. To minimize the manual user modifications, especially for abstract simulation models, different extensions are provided by the injector probe, such as the possibility to specify a system simulation specific re-evaluation. Injecting a fault affects the surrounding components. With simulation directives in low level models, such as `sc_signal` this is handled by the simulation kernel, because events signaling a changed value trigger process executions. This is not the case with abstract functional models or transaction level models. In this case the user has to specify which functions have to be called or which events have to be notified after a fault is injected. In the presented framework the user can externally specify a re-

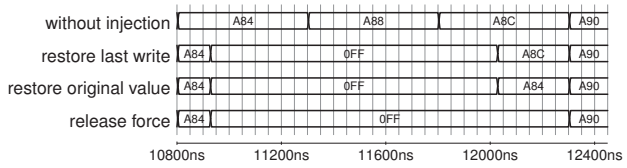

Fig. 2. Different recovery strategies

evaluation strategy with full access to the affected module. This approach was chosen, to prevent the re-writing of the model with sensitive processes. Another feature of the framework is to support different restore strategies. This is required because errors are restored in different ways. A service technician that removes a paper jam in a printing machine restores the system state before the error. Transient errors in memory cells are restored with a subsequent write. After a transient short-to-ground the last driven value, during the error period is restored. The simulation framework has to cope with these different scenarios. Fig. 2 shows the different restore strategies.

While the injection probes enable a write access to the system simulation, monitoring probes enable read access to the system simulation. Because both public and private variables can be replaced, the internal state of modules and the exchanged data or signal values can be modified. This enables the altering of functional and nonfunctional aspects of the system simulation, such as timing aspects. The probes are not associated with any injection behavior; they only provide an interface for the injection control module. This module reads an interpretable fault specification during runtime, using the probes to inject the faults. This extends the configuration file approach and enables the simulation of different injection behaviors and faults without re-compilation. The specification format is called Behavioral Threat Model (BTM) and is based on Timed Automata (TA). Similar to a classic TA the guards depend on local clocks. Additionally the guards depend on the current simulation state, accessed via monitoring probes. To synchronize multiple BTMs local events are provided to synchronize transitions of different BTMs. Besides the local, resettable clocks the BTM provides local variables to store intermediate information, e.g., a local counter to limit the injection amount. The actions and guards are specified, using Python expressions. The injection control module provides an interpreter to evaluate guard statements and to execute actions, enabling the usage of Python libraries such as the pseudo random number generator. Fig. 3 highlights an example BTM that injects randomly distributed, transient faults into an array. In this case it injects faults into a register set of 32 registers. It uses BTM local variables to store the injection mask and Python expressions to handle the array. The injection triggering is time dependent. Using a state-based specification format that is

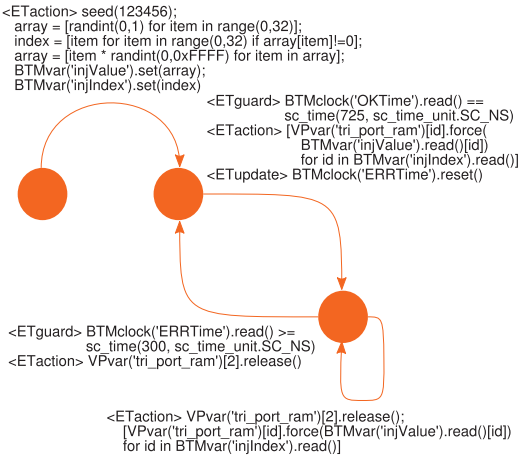

Fig. 3. Example of a Behavioral Threat Model (BTM)

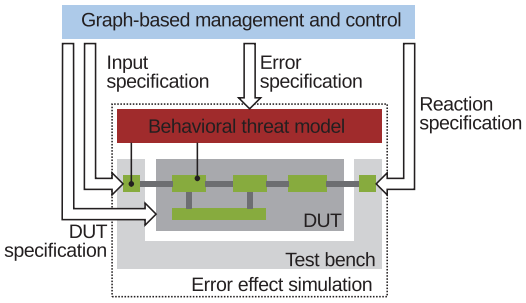

Fig. 4. Error effect simulation with graph-based control

often used to model complex real-time systems offers different advantages. It is best suited to describe complex fault behavior, e.g. it is easily possible to differentiate between permanent, transient and intermittent faults. Another advantage is that it is used to bridge the gap between low-level and abstract system fault models. Well-known low-level fault models could target system parts that are abstracted in system-level models. The neglected system parts would propagate this fault, e.g. the communication controller that propagates noise on the channel to complex transmission errors such as data corruption or transmission delay. To enable the injection of such faults into abstract models this propagation has to be specified within the fault description. Protocol faults are specified in the BTM using the original bit error probability and distribution. Fig. 4 shows the EES, with the required information to execute a simulation. The configuration file has to specify the assembly and parametrization of the Design Under Test (DUT), the system input, the injection behavior that is stimulated during simulation and the expected system reactions. In the following an approach is presented that allows automatically generating all this information.

IV. Safety Verification Plan

During EES one dedicated system instance is evaluated by applying a single injection scenario, consisting of BTMs and system stimulation. To achieve a comprehensive system analysis, different simulation runs with a variety of injection scenarios and system stimulations have to be executed. In the domain of system verification, graph-based verification plans are established to document and partially guide the verification process. Fig. 5 shows a verification plan with the different parts of EES: the system instance (DUT) and its parametrization (DUT-C), the system input, generated by the test bench, the used injection behavior and a set of checks to verify the system reaction. A path through the graph presents one system simulation. Each evaluated path creates one configuration file and executes the simulation. With the help of alternative paths a comprehensive set of analyses is specified.

In this work, Brekers Trek [5] is used as part of the tooling. Trek is a graph-based constraint solver developed for creating functional verification tests for digital designs. Trek provides two types of nodes: diamonds specify a selection of the subsequent nodes, rectangles an unordered sequence. The path evaluation can be executed randomly or guided via forcing and masking nodes, to fulfill e.g. coverage criteria. Considering the huge degree of flexibility, different coverage mechanisms are required to assess the efficiency and level of the executed safety analyses. Based on the verification plan it is possible to highlight the evaluated paths and nodes and calculate the coverage. By using this information the analyzed safety scenarios can be summarized. The execution of BTMs can not be monitored by Trek, therefore a mechanism is implemented to trace the coverage of a BTM during simulation and back-annotate the results into Trek. With this combined approach a detailed assessment of the executed EES and the injected behavior is created.

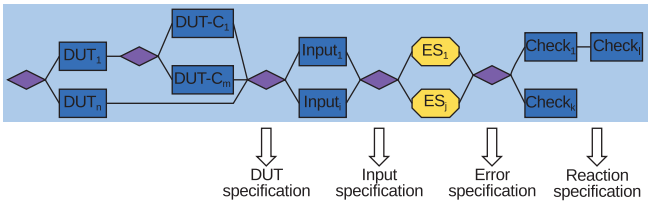

Fig. 5. Verification plan for error effect simulation

V. Simulation Control

Besides the global control of the safety analysis, Trek can be used to realize dynamic injection campaigns. The BTM's state-based specification offers a first degree of flexibility and enables the modeling of the injection scenario with different states and a variety of injection behaviors. Specifying a whole injection campaign with a sequence of multiple injection behaviors would be possible but for each new assembly a new BTM has to be generated. Trek enables assigning basic BTMs to nodes and adding different parametrization alternatives. To get the required flexibility the graph-based specification is evaluated in parallel to the system simulation and can therefore react on events during the simulation. Synchronization between Trek and the SystemC-based simulation is realized with dedicated statements in the BTM. The BTM interpreter enables registering callback functions that can be called by guarded state transitions. This way a callback function to the Trek graph-evaluation can be registered, forwarding information from the system simulation to Trek. The Trek environment evaluates this information and calculates the next BTM. In this case different modes are possible: In the first mode the graph is further evaluated, so only BTMs for components are chosen, which may already be affected by the actual active BTM. The second mode restarts Trek from the beginning (holding the chosen system specification and input), so every defined BTM can be calculated as next. In this case we can use constraints to limit the possibilities. In a third mode we can choose to repeat the actual BTM with a different parametrization. The EES provides an interface for Trek to substitute the currently active BTMs. Therefore, Trek can dynamically change the injection behavior during the simulation. Fig. 6 shows an injection scenario with multiple BTMs, BTM configurations and the synchronization points between Trek and the EES. At each synchronization point the simulation is executed until the callback condition is met and Trek is activated again.

With this approach the different injection strategies can be easily assembled from basic BTMs and the overall injection campaign is well documented. Combining the verification plan generation of the previous section with the injection control presented in this section, results in the following procedure: First the DUT and its parametrization is created, then the used system input is selected. The first BTM specification and the synchronization points are created and all information is stored in the EES configuration file. The EES is executed and every time a synchronization point is reached the control is given to Trek to evaluate the current simulation state and calculate the next injection behavior. This behavior overwrites the currently active injection strategy and the EES is continued until the next synchronization point. After the EES is finished, the specified checks are asynchronously validated with the recorded trace files. Another approach, where assertions are generated due to the checks and validated during simulation, is evaluated too. In this case the assertions are added to the configuration file. After simulation of one path, Trek will calculate the remaining possible paths, each time executing an EES simulation. With this tooling environment a complete safety analysis with multiple system alternatives, different injection scenarios and system stimuli can be easily specified and automatically executed.

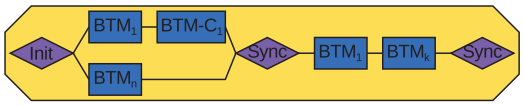

Fig. 6. Error injection plan during simulation

VI. Use case evaluation

The assessment of different system alternatives with a variety of fault injections is demonstrated with the help of a motion-control-system. It consists of an engine and a brushless DC electric motor, which drives a conveyor belt. The conveyor belt is monitored by an optical sensor, measuring its velocity and sending these data as additional information to the programmable logic controller. Fig. 7 shows the motion-control-system with special attention to the engine. For this paper the analyzed system is limited to the virtual prototype of the engine with an abstract modeled rotor part of a permanent magnet synchronous motor, which are the primarily used motors in motion-control-systems.

The power module, the control unit and the motor itself are modeled in SystemC. For each of these components a node in the graph-based description is specified containing the variables for parametrization of the SystemC model. Three different system configurations are developed. The main differences between these configurations are the voltage levels (230V vs 400V) at which the motor is driven, and the granularity of the calculation of the motor reaction depending on the input. At each component node BTMs for every variable are attached which mutate randomly the value at the current state, e.g. multiply a double with a scalar.

To demonstrate the different abstraction levels at which BTMs could take place a BTM is attached, which inject effects of a torn V-belt into the motor model. This scenario is part of a failure scenario analyzed in a Failure Modes, Effects and Diagnostic Analysis (FMEDA) of an industrial application.

Each system configuration is simulated without fault injection and afterwards with different injection scenarios. The monitored system behaviour without fault injection is used to determine the effects of the injected faults.

Each system configuration is simulated exactly for two minutes to determine the error effects. For each system configuration every usable mode in Trek, described in Section V, was executed. For more interesting results, a random evaluation strategy in the graph-based approach was chosen, because in such a small example with coverage guided evaluation only very little differences in the different modes could occur.

With a random execution strategy not every BTM is executed and also not every fault injection leads to a failure.

Experiments with different system configurations during one simulation run are made in the same way, but they are not meaningful in the area of EES, because they should never occur in real life. While attending BTMs to some of the configuration parameters could be useful to model fault scenario like current fluctuations.

VII. Conclusion and further work

In this work an approach is presented that executes a safety analysis with the help of simulation-based fault injection. A modular, reconfigurable system simulation with injection capability builds the core of the analysis and enables the easy evaluation of multiple system alternatives. A graph-based tooling environment reduces the analysis effort significantly, by automatically deriving multiple simulation runs. The user specifies the atomic parts of the analysis and connects them with choice and sequence operations in a graph-based editor. The tool calculates all possible paths, each representing one system simulation. This way the complete safety analysis can be derived from one graph-based specification. Different coverage approaches process the executed simulation runs and

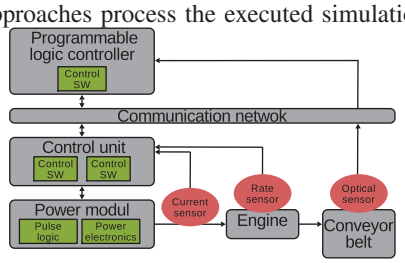

Fig. 7. Engine model as part of the motion-control-system

give the user an assessment of the executed safety analyses. In further work the parametrization of the BTMs should be guided by combinatorial and mutation-based techniques, combined with constraints, to limit the exploration space. The results of the EES should be translated into constraints, so the exploration space could be minimized for further evaluation. Considering the motion-control-example, the framework should be expanded for using SystemC-AMS. The graph-based approach gives the possibility of a high automation, which should also be addressed further.

Acknowledgment

This work has been partially supported by the German Ministry of Science and Education (BMBF) in the project EffektiV under grant 01IS13022.

References

[1] J. Zimmermann, S. Stattelmann, A. Viehl, O. Bringmann, and W. Rosenstiel, "Model-driven virtual prototyping for real-time simulation of distributed embedded systems," in *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on*, 2012, pp. 201–210.

[2] J.-H. Oetjens, N. Bannow, M. Becker, O. Bringmann, and B. et al., "Safety evaluation of automotive electronics using virtual prototypes: State of the art and research challenges," in *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*, June 2014, pp. 1–6.

[3] H. D. Foster, "Trends in functional verification: A 2014 industry study," in *Proceedings of the 52Nd Annual Design Automation Conference*, ser. DAC '15. New York, NY, USA: ACM, 2015, pp. 48:1–48:6. [Online]. Available: http://doi.acm.org/10.1145/2745404.2744921

[4] B. Wile, J. Goss, and W. Roesner, *Comprehensive Functional Verification: The Complete Industry Cycle (Systems on Silicon)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[5] J. Behrend, G. Dittmann, K. Keuerleber, J. Grosse, and F. Krampac, "Graph-Based Verification Patterns," in *ACM/IEEE Design Automation Conference (DAC) – Designer Track (poster)*, ser. DAC '13, 2013.

[6] P. Lisherness and K.-T. Cheng, "SCEMIT: A SystemC error and mutation injection tool," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, 2010, pp. 228–233.

[7] D. Lee and J. Na, "A novel simulation fault injection method for dependability analysis," *Design Test of Computers, IEEE*, Nov 2009.

[8] K.-C. Chang, Y.-C. Wang, C.-H. Hsu, K.-L. Leu, and Y.-Y. Chen, "System-bus fault injection framework in systemc design platform," in *Secure System Integration and Reliability Improvement, 2008. SSIRI '08. Second International Conference on*, 2008, pp. 211–212.

[9] K. Rothbart, U. Neffe, C. Steger, R. Weiss, E. Rieger, and A. Muehlberger, "High level fault injection for attack simulation in smart cards," in *Test Symposium, 13th Asian*, Nov 2004, pp. 118–121.

[10] S. Misera, H. Vierhaus, L. Breitenfeld, and A. Sieber, "A Mixed Language Fault Simulation of VHDL and SystemC," in *Digital System Design: Architectures, Methods and Tools*, 2006.

[11] N. Bombieri, F. Fummi, and G. Pravadelli, "A Mutation Model for the SystemC TLM 2.0 Communication Interfaces," in *Design, Automation and Test in Europe, 2008. DATE '08*, March 2008, pp. 396–401.

[12] R. Shafik, P. Rosinger, and B. Al-Hashimi, "SystemC-based Minimum Intrusive Fault Injection Technique with Improved Fault Representation," in *International On-line Test Symposium (IOLTS)*, April 2008.

[13] A. Fin, F. Fummi, and G. Pravadelli, "AMLETO: a multi-language environment for functional test generation," in *Test Conference, 2001. Proceedings. International*.

[14] S. Reiter, A. Burger, A. Viehl, O. Bringmann, and W. Rosenstiel, "Virtual Prototyping Evaluation Framework for Automotive Embedded Systems Engineering," in *SIMUTools '14*, 2014.