Model-Based Safety Analysis for Vehicle Guidance Systems

Majdi Ghadhab[1], Sebastian Junges[2], Joost-Pieter Katoen[2], Matthias Kuntz[1], and Matthias Volk[2](✉)

[1] BMW AG, Munich, Germany
[2] RWTH Aachen University, Aachen, Germany
matthias.volk@cs.rwth-aachen.de

Abstract. This paper considers the design-phase safety analysis of vehicle guidance systems. The proposed approach constructs dynamic fault trees (DFTs) to model a variety of safety concepts and E/E architectures for drive automation. The fault trees can be used to evaluate various quantitative measures by means of model checking. The approach is accompanied by a large-scale evaluation: The resulting DFTs with up to 300 elements constitute larger-than-before DFTs, yet the concepts and architectures can be evaluated in a matter of minutes.

1 Introduction

Motivation. Cars are nowadays equipped with functions, often realised in software, to e.g., improve driving comfort and driving assistance (with a tendency towards autonomous driving). These functions impose high requirements on functional safety. To meet these requirements, it is crucial to execute these functions with a sufficiently low probability of undetected dangerous hardware failures. ISO 26262 [1] is the basic norm for developing safety-critical functions in the automotive setting. It enables car manufacturers to develop safety-critical devices—in the sense that malfunctioning can harm persons—according to an agreed technical state-of-the-art. The safety-criticality is technically measured in terms of the so-called Automotive Safety Integrity Level (ASIL). This level takes into account driving situations, failure occurrence, the possible resulting physical harm, and the controllability of the malfunctioning by the driver. The result is classified from ASIL QM (no special safety measures required) up to ASIL D (with ASIL A, B, C in between). This paper considers the design-phase safety analysis of the *vehicle guidance* system, a key functional block of a vehicle with a high safety integrity level (ASIL D, i.e., 10^{-8} residual hardware failures per hour). The crux of our approach is to: (1) construct dynamic fault trees [2] (DFTs) from system descriptions and combine them (in an automated manner) with hardware failure models for several partitionings of functions on hardware, and (2) analyse the resulting overall DFTs by means of probabilistic model checking [3].

S. Tonetta et al. (Eds.): SAFECOMP 2017, LNCS 10488, pp. 3–19, 2017.
DOI: 10.1007/978-3-319-66266-4_1

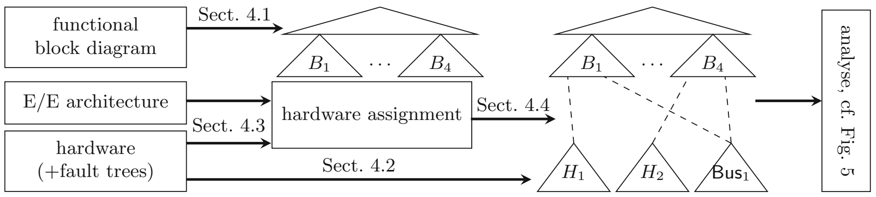


Fig. 1. Overview of the model-based safety approach

A Model-Based Approach. Figure 1 summarises the approach of this paper. The failure behaviour of the functional architecture, given as a functional block diagram (FBD), is expressed as a two-layer DFT: the upper layer models a system failure in terms of block failures B_i while the lower level models the causes of block failures. The use of DFTs rather than static fault trees allows to model warm and cold redundancies, spare components, and state-dependent faults; cf. [4]. Each functional block is assigned to a hardware platform for which (by assumption) a DFT is given that models its failure behaviour. Depending on the partitioning, the communication goes via different fallible buses. From the partitioning, and the DFTs of the hardware and the functional level, an overall DFT is constructed (in an automated manner) consisting of three layers: (1) the system level; (2) the block level; and (3) the hardware level. Details are discussed in Sect. 4.

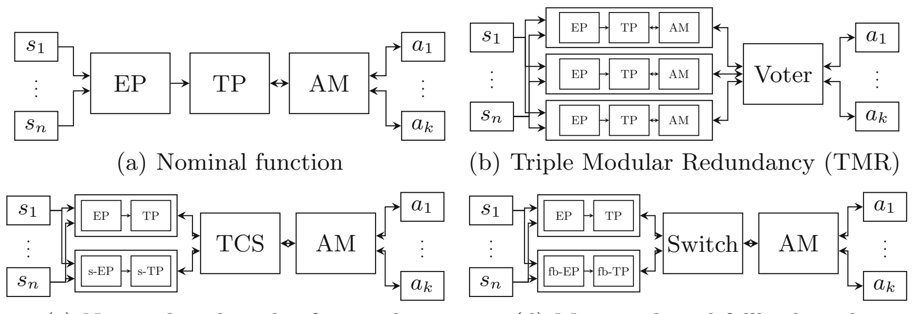
Analysis. We exploit probabilistic model checking (PMC) [3] to analyse the DFT of the overall vehicle guidance system. It can be used as a black-box algorithm—no expertise in PMC is needed to understand its outcomes—and supports *various metrics* that go beyond reliability and MTTF [5]. In contrast to simulation, where results are obtained with a given statistical confidence, PMC provides *hard guarantees* that the safety objectives are met. This is important as ISO 26262 requires metrics to be objectively assessable: "metrics are verifiable and precise enough to differentiate between different architectures" [1, 5:8-2]. Whereas most ISO 26262-based analyses focus on single and dual-point of failures, PMC naturally supports the analysis of *multi-point of failures* of the vehicle guidance system's DFT. This is highly relevant, as "[for systems where the] concept is based on redundant safety mechanism, multiple-point failures of a higher order than two are considered in the analysis" [1, 5:9-4].

Contributions. The main contribution of this paper is two-fold: We report on the usage of dynamic fault trees for safety analysis in a potential automotive setting. While standard fault tree analysis is part of the ISO 26262, the usage of DFTs in this field is new. The paper shows how additional features help to create faithful models of the considered scenarios. These models are then used to analyse the given scenarios. To increase the applicability of DFTs as a method for probabilistic safety assessment in an industrial setting, we give concrete building blocks to work with, e.g. redundancy and faults covered by fallible safety mechanisms.

A clear benefit of the usage of DFTs is that all these methods are integrated in existing off-the-shelf analysis tools, which provide sound error bounds. This reduces the amount of domain specific knowledge in the analysis, and thus supports a more model-oriented approach. In this paper, we utilise this to investigate the effect of different hardware partitioning on a range of metrics. The generated fault trees are to the best of our knowledge the largest real-life ones in the literature – larger trees have only been artificially created for scalability purposes [6]. In particular, this paper is the first to consider the model-checking based approaches for DFT analysis on real-life case studies.

Related Work. Earlier work [7] considers an automotive case study where functional blocks are translated to static fault trees without treating the partitioning on hardware architectures. The effect of different topologies of a FlexRay bus has been assessed using FTA in [8]; and identified the need for modelling dynamic aspects. The analysis of architecture decisions under safety aspects has been considered in e.g. [9] using a dedicated description language and an analytical evaluation. Safety analysis for component-based systems has been considered in [10], using state-event fault trees. Qualitative FTA has been used in [11] for ISO 26262 compliant evaluation of hardware. Different hardware partitionings are constructed and analysed using an Architecture Description Language (ADL) in [12]. ADL-based dependability analysis has been investigated for several languages, e.g., AADL [13], UML [14], Arcade [15], and HiP-HOPS [16]. These approaches typically have a steeper learning curve than the use of DFTs. The powerful Möbius analysis tool [17] has recently been extended with dynamic reliability blocks [18]. Model checking for safety analysis has been proposed by, e.g., [19]; which focuses on AltaRica, and does not cover probabilistic aspects.

Remark. The proposed concepts and architectures are exemplary. No implication on actual safety concepts or E/E architectures implemented by BMW AG can be derived from these examples. The same remark applies on any quantity (failure rates, obtained metrics, ...) presented in this paper.



(a) Nominal function (b) Triple Modular Redundancy (TMR)

(c) Nominal path and safety path (d) Main path and fallback path

Fig. 2. Different functional block diagrams for vehicle guidance

2 Vehicle Guidance

The most challenging safety topic in the automotive industry is currently the driving automation, where the driving responsibility is moving partly or even entirely from the driver to the embedded vehicle intelligence. Rising liability questions make it crucial to develop functional safety concepts adequately to the intended automation level and to provide evidence regarding the integrity and the reliability of these concepts.

2.1 Scenario

As a real-life case study from the automotive domain, we propose to consider the functional block diagram (FBD) in Fig. 2(a) representing the skeletal structure of automated driving. Data collected from different sensors (cameras, radars, ultrasonic, etc.) are synthesised and fused to generate a model of the current driving situation in the Environment Perception (EP). The model is used by the Trajectory Planning (TP) to build a driving path with respect to the current driving situation and the intended trip. The Actuator Management (AM) ensures the control of the different actuators (powertrain, brakes, etc.) following the calculated driving path. Thus, the blocks in the FBD fulfil tasks: The tasks are realised by (potentially redundant) functional blocks, connected by lines to depict dataflow. These diagrams are *not* reliability block diagrams in which the system is operational as long as a path through operational blocks exist. According to the automation level, the vehicle guidance function must be designed as *fail-operational*, that is, the system should safely continue to operate for a certain time after a failure of one of its components.

2.2 Modelling of Safety Concepts

Functional Safety Concepts. Based on the criticality of the vehicle guidance function, especially when the driver is out-of-the-loop, ASIL D applies to it. Different design patterns have been developed and implemented in safety-critical systems with fail-operational behaviour and high safety levels, cf. e.g. [20]. The variety of possibilities is illustrated by the following three concepts:

SC1- Triple Modular Redundancy (TMR), Fig. 2(b): The nominal function for vehicle guidance is replicated into three paths each fulfilling ASIL B. A Voter, fulfilling ASIL D, ensures that any incorrect path is eliminated.

SC2- Nominal path and safety path, Fig. 2(c): Consists of two different paths, a nominal path (n-Path) and a safety path (s-Path) in hot-standby mode. The n-Path provides a full extent trajectory with ASIL QM and the s-Path a reduced extent trajectory but with highest safety and integrity level ASIL D. The safety trajectory is generated from a reduced s-EP (safety Environment Perception) and s-TP (safety Trajectory Planning). The Trajectory Checking and Selection (TCS) verifies whether the trajectory calculated by the n-Path is within the safe range calculated by the s-Path or not. In the case of failure, the s-Path takes over the control and the safe trajectory with reduced extent is followed by the AM. In this case, we consider the system to be *degraded*.

SC3- Main path and fallback path, Fig. 2(d): Similar to SC2 although the main path (m-Path) is now developed according to ASIL D in order to detect its own hardware failures and signalise them to the Switch. The Switch then commutates the control of the AM to a fallback path (fb-Path) with ASIL B. In this case, we consider the system to be *degraded*.

Technical Safety Concepts and Partitioning on E/E Architecture. The next design step consists of extending the nominal E/E architecture for vehicle guidance and partitioning the blocks of every safety concept on its elements. The nominal E/E architecture is represented in Fig. 3(a). The vehicle guidance function is implemented on an ADAS-platform (Advanced Driver Assistance System) which is connected to all used sensors. A number of dedicated ECUs (Electronic Control Unit) control the actuators. On an I-ECU (Integration ECU), additional, non-dedicated actuation functions can be implemented. Naturally, implementing all blocks from the safety concepts on the ADAS in Architecture A defeats the purpose of the redundant paths.

Figure 3 gives further illustrative examples for E/E architectures for the different safety concepts: For SC1, Architecture B (Fig. 3(b)) allows an implementation of the three redundant paths on separate ADAS-cores. The Voter is implemented on the I-ECU. For SC2, the following two implementations both yield ASIL D for the safety path, each with TCS and AM on the I-ECU: (1) Executing the nominal path on one ADAS and redundant execution of the s-Path on two ADAS-cores in *lock-step mode*, using Architecture B. (2) *Encoded execution* [21] of the s-Path on a single ADAS+-core in Architecture C (Fig. 3(c)), the + refers to the additional hardware resources to run an encoded s-Path. An E/E architecture for SC3 could run on Architecture C, where the m-Path is implemented on ADAS$_1$ and the fb-Path on ADAS$_2$. Alternatives are considered in our experiments in Sect. 5.

Hardware Platforms and Faults. We assume that all hardware platforms can completely recover from transient faults (e.g. by restarting the affected path),

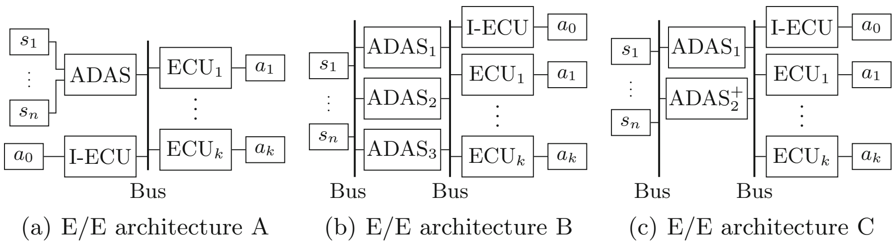


Fig. 3. Different E/E architectures

so that only transient fault directly leading to a failure of the system are of importance. As transient faults disappear quickly, the probability for another fault occurring in presence of a transient fault is negligible. It is thus reasonable to assume that during a transient fault, no other faults occur [1].

2.3 Measures

The safety goal for the considered systems is to avoid wrong vehicle guidance. As the system is designed to be fail-operational, the system should be able to maintain its core functionality for a certain time. The safety goal is violated, if e.g. two out of three TMR paths or both the n-Path and the s-Path fail. The goal is also violated if e.g. a failure of the n-Path is not detected. The safety goal is classified as ASIL D. We stress that safe faults do not need to be considered. For ease of notation we define the *complement* of probability p as $1 - p$.

Several measures allow insights in the safety-performance of the different safety concepts: *System integrity* refers to the probability that the system safely operates during the considered operational lifetime. To obtain the average failure-probability per hour, the complement of the integrity is scaled with the lifetime (to determine the failures in time, FIT). Besides the integrity, the mean time to failure (MTTF) is a standard measure of interest. We expect that only a reduced functionality is provided in the degraded states: The following measures reflect insights also relevant for customer satisfaction: (1) the probability at time t that the system provides the full functionality, (2) the fraction of system failures which occur without going to a degraded state first, (3) the expected time to failure upon entering the degraded state, (4) the criticality of a degraded state, in terms of the probability that the system fails within e.g. a typical drive cycle, and (5) the effect on the overall system integrity when imposing limits on the time a system remains operational in a degraded state. It is important to consider the robustness or *sensitivity* of all measures w.r.t. changes in the failure rates.

3 Technical Background

3.1 Fault Trees

Fault trees (FTs) are directed acyclic graphs (DAG) with typed nodes. Nodes of type T are referred to as "a T". Nodes without children (successors in the DAG), are *basic events* (BEs, Fig. 4(a)). Other nodes are *gates* (Fig. 4(b)–(h)). A BE *fails* if the event occurs; a gate fails if its *failure condition* over its children holds. The *top level event* ($\mathsf{TLE}(F)$) is a specifically marked node of a FT F. $\mathsf{TLE}(F)$ fails iff the FT F fails.

Static Fault Trees. The key gate for static fault trees (SFTs, gates (b)–(d)) is the *voting* gate (denoted VOT_k) with *threshold* k and at least k children. A VOT_k-gate fails, if k of its children have failed. A VOT_1 gate equals an OR-gate, while a VOT_k with k children equals an AND-gate.

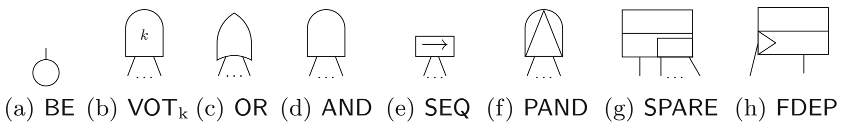


Fig. 4. Node types in ((a)–(d)) static and (all) dynamic fault trees

Dynamic Fault Trees. As SFTs are not expressive enough for a faithful model, cf. e.g. [22], dynamic fault trees (DFTs) additionally allow the following gates:

Sequence-Enforcers. The sequence enforcer (SEQ, Fig. 4(e)) restricts the order in which BEs can fail; its children may only fail from left-to-right. Contrary to a widespread belief, SEQs cannot be modelled by SPAREs (introduced below) [22]. SEQs appear in [1, 10-B.3], where they are indicated by the boxed L.

Priority-and. The (binary) priority-and (PAND, Fig. 4(f)) fails iff all children failed and the right child did not fail before the left child.

Spare-Gates. Spare-gates (SPARE, Fig. 4(g)) model spare-management and support warm and cold standby. Warm (cold) standby corresponds to a reduced (zero) failure rate. Likewise to an AND, a SPARE fails if all children have failed. Additionally, the SPARE *activates* its children from left to right: A child is activated as soon as all children to its left have failed. By activating and therefore using a child the failure rate is increased. The children of the SPAREs are assumed to be roots of independent subtrees, these subtrees are called *modules*. Upon activation of the root of a module, the full module is activated.

Functional Dependencies. Functional dependencies (FDEP, Fig. 4(h)) ease modelling of feedback-loops. FDEPs have a *trigger* (a node) and a *dependent event* (a BE). Instead of propagating failure upwards, upon the failure of the trigger, the dependent event fails. FDEPs are syntactic sugar in SFTs, but not in DFTs [6].

Activation Dependencies. To overcome syntactic restrictions induced by SPAREs and to allow greater flexibility with activation, we use *activation dependencies* (ADEPs), as proposed in [22, Sect. 3E]. If the *activation source* is activated, this is propagated to the *activation destination*. ADEPs are typically used in conjunction with an FDEP, where the activation sources are the dependent events and the activation target is the trigger.

3.2 Analysing DFTs by Model Checking

Both the measures of interest (Sect. 2.3) and the dynamic extensions to fault trees refer to different states in the model. It is therefore natural to make these states explicit with a state-based model. Markov models have been long used in model checking and performance evaluation as such state based models [23].

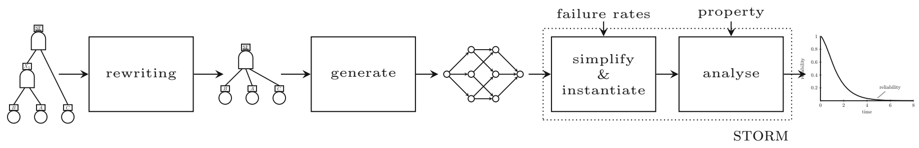


Fig. 5. Overview of the STORM-dft approach

STORM (see http://www.stormchecker.org) is the state-of-the-art tool for the automatic analysis of DFTs via a conversion to Markov Automata (MA) [5]. The tool chain is depicted in Fig. 5. As a first step, we *rewrite* the given FT: Whereas an FT has a lot of structural requirements that make it a good and readable fault tree, cf. [24], these requirements are counterproductive in terms of automated analysis: Consider e.g. superfluous levels of ORs in the FT. DFTs can automatically be rewritten using the techniques from [6] to make them more suitable for analysis purposes. STORM translates the rewritten DFT into a (deterministic) parameterised MA: This is the computationally most expensive step. STORM then reduces the parameterised MA to a parametric continuous time Markov chain (CTMC). Next, the parametric CTMC is instantiated to a CTMC by considering concrete values for all failure rates: Thus, the model construction is invoked only once for multiple sets of failure rates. The last step executes the model-checking routine with a property to produce the required result.

Properties. STORM takes a wide variety of measures, corresponding to continuous stochastic logic properties with reward extensions [3]. Most relevant are:

Reach-Avoid Probability. Given sets of target states and bad states, this measure gives the probability to reach the target state without visiting a bad state meanwhile. If the bad set is empty, this is *reachability.*

Time-Bounded Reach-Avoid Probability. Given an additional deadline, it gives the probability to reach a target state (avoiding bad states) within the deadline.

Expected Time. Given a set of target states that are eventually reached, it gives for each state the expected time to reach the target.

Evidence. Some measures require the evaluation of degraded system states on the earliest moment that certain nodes in the FT already failed. Typically, the original DFT is also analysed, and the complete state space has been constructed before. Thus, it is beneficial to deduce measures for the degraded system on the original state space. Using *evidence* allows defining degraded states as initial states in the underlying model to ease analysis. Given evidence in the form of a set of BEs considered as already failed, all possible orderings (traces) in which they failed are considered. Following these traces from the initial state of the model yields a set of states S; based on the evidence, the system is in one of these states. For all states in S, we obtain values for the measures via model checking. Evidence in the form of gates is conceptually equivalent, but realised via backward-search.

Transient Faults. After the occurrence of a transient fault, the system either fails directly, or the fault disappears and the system returns to its previous state. Transient events are thus considered in each state, but are only added to the underlying MA if they lead to the failure of the TLE.

Approximate Reliability. To alleviate the state space explosion problem, we lifted the idea for sound over- and under-approximation of the MTTF [5] to reliability. The central observation is that typically only a fraction of the state space is relevant to obtain tight bounds for the property at hand. Thus, only a fraction of the state space needs to be constructed: A safe over-approximation of the complement of the reliability consists in assuming that each terminal state in the partial state space corresponds to the failure of the TLE, whereas an under-approximation simply assumes that the TLE cannot fail if it has not failed in one of the terminal states.

4 Methodology

This section describes details for the approach from Fig. 1.

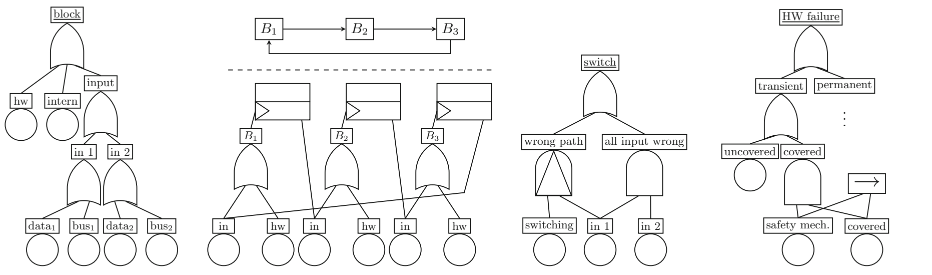
4.1 From Functional Block Diagrams to Functional Fault Trees

Definition 1 (Block diagram). *A* block diagram $\mathcal{D} = (\mathsf{Blc}, \rhd)$ *is a finite directed graph. The vertices are called* blocks*, the edges are called* channels*.*

Formally, the input for this step is any block diagram, the output is a FT F with *dummy events* for each hardware piece: $\{f_x \mid x \in \mathsf{Blc} \cup \rhd\} \subseteq F_{\mathsf{BE}}$.

The *functional FT*, i.e. the system and the block level, is created manually: This has several advantages, most importantly: (1) There is no need to formalise the semantics of the FBD and its implicit assumptions, e.g. different failure behaviour of voters or edges with different meaning in FBD, and adaptions can be made by hand. (2) Constructing the FT is an important step in the development-*process* of safety-critical systems [1].

For the considered scenarios, the TLE is assumed to represent the safety-critical failure of the system. A *task-based* partitioning of the FT is helpful: The TLE is assumed to fail if any of the tasks can no longer be executed. The tasks fail if no block can realise the task anymore. Dataflow is encoded by encoding an error in the input as a fault. The failure behaviour of a single block is precisely expressed as a fault tree (referred to as *block fault tree*). A typical example is depicted in Fig. 6(a): The hardware-failure is a dummy event, to be



(a) Block FT (b) Feedback loop and FT (c) Switch FT (d) Hardware FT

Fig. 6. Illustrative fragments of the fault trees

connected with the hardware FT, and internal failures can be used for sub-blocks or for additional descriptions. The input fails if either of the two input channels fail: The channels fail if either the hardware for the channel fails or the block creating the input fails, dummies for bus hardware and data flow, respectively. It is important to support feedback loops, which are realised by the use of FDEPs, as in [24]. The structure (simplified) is depicted in Fig. 6(b): For three blocks as shown on top, the three FTs are connected via FDEPs. If e.g. B_1 fails, the failure is propagated to the input of B_2, etc. The use of FDEPs prevents that cyclic dependencies yield cyclic FTs, and is very flexible. The scheme can be used as-is for hierarchical descriptions of both tasks and functional blocks. For the safety concepts, the following remarks apply.

Triple Modular Redundancy. For the Voter input failures, rather than an OR over the inputs as in Fig. 6(a), a VOT_2 is used. The block fails if two inputs fail.

Fallback and Cold-Redundancy. The fallback-path is in cold standby: Using a SPARE instead of an AND to encode the failure of all blocks fulfilling a task ensures the activation of the fallback-path.

Path Switching. The Switch (SC3) may fail. This only leads to failure when the path has to be switched. This is reflected by the FT in Fig. 6(c): the Switch fails if it either uses the wrong path or if all input is wrong. The path is wrong if the switching mechanism fails before the primary input fails, i.e., it can no longer switch to an operational path. The scheme can be extended to more paths.

4.2 Fault Trees for Hardware

We support arbitrary (D)FTs to model the hardware failures. We briefly illustrate how to integrate coverage and both transient and permanent faults, as in [1, 10:B]. The DFT is depicted in Fig. 6(d). Faults are either transient or permanent faults. Each type has its own corresponding safety mechanism. A transient fault occurs if either the fault is not covered by the safety mechanism or the fault is covered but the safety mechanism has failed before. The latter is modelled with use of a SEQ, it cannot be modelled faithfully by a static FT.

4.3 Hardware Assignment and E/E Architecture

Definition 2 (Hardware assignment). *Given a block diagram $\mathcal{D} = (\mathsf{Blc}, \rhd)$, a set of hardware-platforms H and a set of buses Bus, a* hardware-assignment *is a function $h \colon \mathcal{D} \to H \cup \mathsf{Bus}$ s.t. $h(\mathsf{Blc}) \subseteq H$ and $h(\rhd) \subseteq \mathsf{Bus}$.*

Typically, we only want to assign blocks to hardware-platforms; the channel-assignment then follows from the E/E architecture. This can be captured by a simple set of rules, exemplified below: Any channel between two blocks assigned to the same hardware platform is realised by the internal infrastructure on the platform; it requires no additional hardware. All other channels are realised via the same bus, as typical in CAN or FlexRay. Formally, assume $\mathsf{Bus} = \{\text{CAN}\} \cup \{c_p \mid p \in H\}$. For a fixed block-assignment, the channel-assignment between blocks s and t is thus $c_{h(s)}$ if $h(s) = h(t)$ and CAN otherwise.

4.4 Constructing a Complete Fault Tree

To merge the FTs, we take the disjoint union of the functional FT F with $\{f_x \mid x \in \mathsf{Blc} \cup \rhd\} \subseteq F_{\mathsf{BE}}$, and the hardware FTs $\{F_y \mid y \in H \cup \mathsf{Bus}\}$. For each $x \in \mathsf{Blc} \cup \rhd$, an FDEP with trigger $\mathsf{TLE}(F_{h(x)})$ and dependent event f_x is added, and an ADEP in the reverse direction. The ADEP ensures that the hardware FTs are correctly activated. For example, consider the block FT in Fig. 6(a) and the corresponding hardware FT in Fig. 6(d). The TLE "HW failure" of the hardware FT is connected to the dummy BE "hw" of the block FT with an FDEP.

4.5 Translating Measures

We translate the measures introduced in Sect. 2.3. The formal definition of the measures is given in Table 1. As we construct the FT such that the TLE corresponds to a violation of the safety goal, *(system) integrity* corresponds to reliability in FT analysis. Thus, the integrity is obtained by first computing the time-bounded probability to reach a state where the TLE has failed and then complementing this value. *FIT* computation is realised by post-processing. The *MTTF* corresponds to the expected time until a state is reached where the TLE has failed. In the considered DFTs, the expected time is always defined. For degraded states, for (1) we compute the complement of the time-bounded reachability probability for reaching a failed or degraded state and call this *Full Function Availability* (FFA), for (2) we compute the time-bounded reach-avoid probability avoiding degraded states and reaching the TLE failure and call this *Failure Without Degradation* (FWD), for (3) we compute the expected time from the moment of degradation to a failure. We scale these with the probability to reach such a state. The result is the *Mean Time from Degradation to Failure* (MTDF). Furthermore, for (4) we compute the reliability from the moment of degradation: we take the minimum over all states and call the complement the *Minimal Degraded Reliability* (MDR) and for (5) we compute the time-bounded reachability to a TLE failure with a drive cycle as deadline starting from the

Table 1. Definition of measures

	Measure	Model Checking Queries
System	integrity	$1 - P(\Diamond^{\leq t} \text{ failed})$
	FIT	$\frac{1}{\text{lifetime}} \cdot \big(1 - P(\Diamond^{\leq \text{lifetime}} \text{ failed})\big)$
	MTTF	$\mathsf{ET}(\Diamond \text{ failed})$
Degradation	FFA	$1 - P\big(\Diamond^{\leq t} \text{ (failed} \vee \text{degraded)}\big)$
	FWD	$P\big((\neg\text{degraded}) \ \mathsf{U}^{\leq t} \ (\neg\text{degraded} \wedge \text{failed})\big)$
	MTDF	$\Sigma_{s \in \text{degraded}}\big(P(\neg\text{degraded} \ \mathsf{U} \ s) \cdot \mathsf{ET}^{s}(\Diamond \text{ failed})\big)$
	MDR	$\text{argmin}_{s \in \text{degraded}}\big(1 - P^{s}(\Diamond^{\leq t} \text{ failed})\big)$
	SILFO	$1 - \Big(FWD + \Sigma_{s \in \text{degraded}}\big(P(\neg\text{degraded} \ \mathsf{U}^{\leq t} \ s) \cdot P^{s}(\Diamond^{\leq \text{drivecycle}} \text{ failed})\big)\Big)$

moment we reach a degraded state. We scale this with the probability to reach a degraded state in time, and add the FWD. We call the complement *System Integrity under Limited Fail-Operation* (SILFO). Sensitivity analysis is realised by a series of experiments with different failure rates.

5 Experiments

The goal of the presented experiments is to show the applicability of the proposed methodology on systems and concepts similar to those from Sect. 2.2.

5.1 Set-up

All experiments are executed on a 2.9 GHz Intel Core i5 with 8 GB RAM. We consider the three safety concepts. For each SC, we construct a functional FT. We include four sensors, of which two are required for safe operation, and four actuators, which are all required for safe operation.

Different Partitioning Schemes. For each concept, we vary the baseline partitions from Sect. 2.2. For each concept, we additionally use (1) the nominal architecture and (2) remove the I-ECU from the architecture; reassigning the components to ADAS$_2$. For some combinations, we vary the hardware, by e.g. considering a redundant bus or introducing more hardware platforms. We additionally scale the number of sensors and actuators and the required number of sensors for safe operation.

Failure Rates. For presentation purposes we assume the following failure rates, which do not necessarily reflect reality and especially do not reflect any system from BMW AG. We assume that functional blocks, e.g. EP, are free of systematic faults. Sensors, actuators, and ECUs have failure rates of 10^{-7}/h. In the ADAS hardware platforms transient (permanent) faults occur with rate 10^{-4}/h

(10^{-5}/h), respectively. All faults can be detected by a safety mechanism covering 99%/90%/60% (for ASIL D/B/QM) of the faults, which fails with rate 10^{-5}/h. For ADAS+, failure rates increase by a factor 10 and coverage becomes 99.9%.

Tool-Support. Both the generation of the FT corresponding to the safety concepts as well as their analysis are supported by a Python toolchain. For a FBD and block trees for each block, dependencies based on the data flow in the FBD are automatically inserted. Give an E/E architecture, a partitioning, hardware FTs and the functional FT, the complete FT is automatically generated. The analysis of DFTs as in Fig. 5 is completely automated. We additionally add templates for the hardware FTs, such that changes in coverage or failure rates require only single parameters to be changed.

5.2 Evaluation

An overview of (a selection of) considered scenarios and the corresponding DFTs and CTMCs can be found in Table 2. Each scenario is defined by the safety concept, the used architecture with possible adaptions and the fraction of sensors and actuators which have to be operational (columns 2–6). The next three columns give the number of basic events, dynamic gates and the total number of nodes in the DFT. The last three columns contain the number of states and transitions in the CTMC after applying reduction techniques and the percentage of degraded states in the CTMC. The analysis results for the measures from Sect. 4.5 are given in Table 3. Notice that SC1 does not have degraded states. The times for generating the model and computing each measure are given in Table 4. Figure 7 illustrates the obtained measures for a variety of concepts and architectures. The complement of the integrity is given in Fig. 7(a) and FIT in Fig. 7(b). Figure 7(c) considers SC2 and SC3 on Architecture C: The straight lines were obtained by the baseline failure rates for the hardware components, whereas the dashed (dotted) lines were obtained assuming an increased (decreased) coverage according to an increased (decreased) ASIL level. The graph in Fig. 7(d) displays the SILFO for the safety concepts with degraded states and a drive cycle of one hour. Moreover, computing an approximate integrity allowing a 3% relative error on the largest scenario could be computed within 22 s (requiring only 324,990 states).

5.3 Analysis of Results

Table 2 indicates that reduction techniques successfully alleviate the state space problem: Depending on the scenario, the generated state space remains small even for hundreds of elements. Naturally, latent faults increase the state space, but then the effectiveness of the approximation increases. Most of the considered measures can be computed within seconds even on the largest models. However, MTDF and SILFO are computationally more expensive, as model checking queries have to be performed for each degraded state. For SILFO, the failure

Table 2. Model characteristics

Scenario						DFT			CTMC		
	SC	Arch.	Adap.	Sens.	Act.	#BE	#Dyn.	#Elem.	#States	#Trans.	Degrad.
I	SC1	B	—	2/4	4/4	76	25	233	5,377	42,753	—
II	SC2	B	—	2/4	4/4	70	23	211	5,953	50,049	19.35%
III	SC2	C	ADAS+	2/4	4/4	57	19	168	1,153	7,681	16.65%
IV	SC3	C	—	2/4	4/4	57	21	170	385	1,985	12.47%
V	SC2	A	—	2/4	4/4	58	19	185	193	897	0.00%
VI	SC2	B	removed I-ECU	2/4	4/4	65	21	199	1,201	8,241	19.98%
VII	SC2	B	5 ADAS, 2 BUS	2/8	7/7	96	30	266	194,433	2,171,905	19.35%
VIII	SC2	B	8 ADAS, 2 BUS	6/8	7/7	114	36	305	3,945,985	66,225,665	10.90%

Table 3. Obtained measures with operational lifetime = 10,000 h and drive cycle = 1 h

Scenario		System			Degradation				
		1-integ.	FIT	MTTF	FFA	FWD	MTDF	MDR	SILFO
I	SC 1/B	1.58E-2	2.49E-6	85,658	–	–	–	–	–
II	SC 2/B	1.02E-2	1.13E-6	341,954	5.17E-2	9.98E-3	227,565	2.88E-1	9.98E-3
III	SC 2/C (ADAS+)	1.21E-2	1.66E-6	111,380	5.23E-2	1.07E-2	20,808	7.35E-1	1.07E-2
IV	SC 3/C	1.02E-2	1.13E-6	284,685	1.58E-2	1.02E-2	135,124	2.14E-1	1.02E-2
V	SC 2/A	5.99E-2	9.29E-6	69,177	5.99E-2	5.99E-2	0	0	5.99E-2
VI	SC 2/B (I-ECU)	1.12E-2	1.23E-6	344,309	5.27E-2	1.10E-2	230,976	2.05E-1	1.10E-2
VII	SC 2/B (5 ADAS)	1.71E-2	1.83E-6	280,228	5.83E-2	1.67E-2	173,305	3.66E-1	1.67E-2
VIII	SC 2/B (8 ADAS)	1.71E-2	1.83E-6	269,305	9.78E-2	1.64E-2	TO	4.34E-1	TO

Table 4. Timings

	I	II	III	IV	V	VI	VII	VIII
Model generation	1.02 s	1.02 s	0.38 s	0.33 s	0.34 s	0.40 s	25.13 s	632.89 s
System + FFA + FWD	0.02 s	0.02 s	0.00 s	0.00 s	0.00 s	0.00 s	1.46 s	46.67 s
MTDF	—	2.67 s	0.18 s	0.03 s	0.02 s	0.20 s	2892.42 s	>3600 s
MDR	—	0.60 s	0.11 s	0.02 s	0.02 s	0.11 s	26.07 s	781.93 s
SILFO	—	1.83 s	0.17 s	0.04 s	0.02 s	0.18 s	1694.91 s	>3600 s

probability within a drive cycle is orders of magnitude smaller than the FWD indicating that the duration of the drive cycle is insignificant for SILFO.

The variety of measures obtained allows some insights in the effect of different safety concepts that go beyond merely meeting specific targets. The MTTF indicates that system integrity of SC3/C and SC2/B are superior, with SC2/B slightly better than SC3/C. A similar claim can be deduced from Fig. 7(a). The differences between SC3/C and SC2/B are marginal. Figure 7(b) indicates that it is not always sufficient to look at the FIT as a measure as the value changes with the considered operation time. The sensitivity shows that the influence of the safety coverage in SC2 is higher than in SC3. Thus, the importance of fault coverage in platforms depends on the chosen architecture. SC2/B and SC3/C differ in their failure behaviour of degraded states as seen in Fig. 7(d). When limiting the driving time in the degraded state to one hour SC2/B offers a

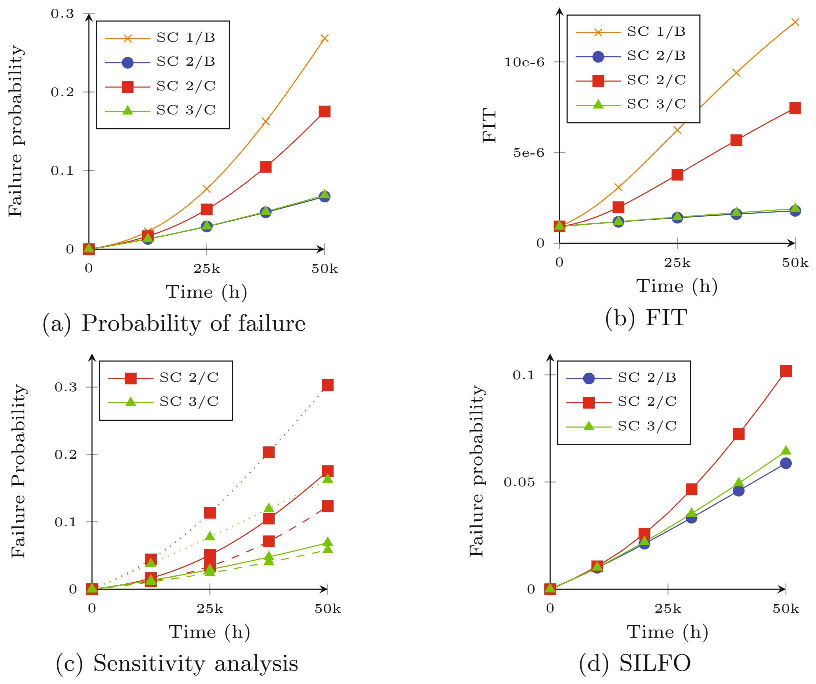


Fig. 7. Analysis results

better integrity than SC3/C, whereas in the overall integrity the difference is marginal.

6 Conclusion

We presented a model-based approach towards the safety analysis of vehicle guidance systems. The approach (see Fig. 1) takes the system functions and their mapping onto the hardware architecture into account. Its main benefit is the flexibility: new partitionings and architectural changes can easily and automatically be accommodated. The obtained DFTs were analysed with probabilistic model checking. Due to tailored state-space generation [5], the analysis of these DFTs—with up to 100 basic events—is a matter of minutes. Future work consists of involved error propagation schemes and a rigorous treatment of transient faults.

References

1. ISO: ISO 26262: Road Vehicles - Functional Safety (2011)
2. Dugan, J.B., Bavuso, S.J., Boyd, M.: Fault trees and sequence dependencies. In: Proceedings of RAMS, pp. 286–293 (1990)

3. Katoen, J.P.: The probabilistic model checking landscape. In: Proceedings of LICS, pp. 31–45. ACM (2016)
4. Ruijters, E., Stoelinga, M.: Fault tree analysis: a survey of the state-of-the-art in modeling, analysis and tools. Comput. Sci. Rev. **15–16**, 29–62 (2015)
5. Volk, M., Junges, S., Katoen, J.P.: Fast dynamic fault tree analysis by model checking techniques. IEEE Trans. Ind. Inform. (2017, to appear)
6. Junges, S., Guck, D., Katoen, J.P., Rensink, A., Stoelinga, M.: Fault trees on a diet: automated reduction by graph rewriting. Formal Asp. Comput. **29**(4), 651–703 (2017)
7. McKelvin, M.L., Sangiovanni-Vincentelli, A.: Fault tree analysis for the design exploration of fault tolerant automotive architectures. SAE International, SAE Technical Paper, pp. 1–8 (2009)
8. Leu, K.L., Chen, J.E., Wey, C.L. Chen, Y.Y.: Generic reliability analysis for safety-critical flexray drive-by-wire systems. In: Proceedings of ICCVE, pp. 216–221 (2012)
9. Rupanov, V., Buckl, C., Fiege, L., Armbruster, M., Knoll, A., Spiegelberg, G.: Employing early model-based safety evaluation to iteratively derive E/E architecture design. Sci. Comput. Program. **90**, 161–179 (2014)
10. Grunske, L., Kaiser, B., Papadopoulos, Y.: Model-driven safety evaluation with state-event-based component failure annotations. In: Heineman, G.T., Crnkovic, I., Schmidt, H.W., Stafford, J.A., Szyperski, C., Wallnau, K. (eds.) CBSE 2005. LNCS, vol. 3489, pp. 33–48. Springer, Heidelberg (2005). doi:10.1007/11424529_3
11. Adler, N., Otten, S., Mohrhard, M., Müller-Glaser, K.D.: Rapid safety evaluation of hardware architectural designs compliant with ISO 26262. In: Proceedings of RSP, pp. 66–72. IEEE (2013)
12. Walker, M., Reiser, M., Piergiovanni, S.T., Papadopoulos, Y., Lönn, H., Mraidha, C., Parker, D., Chen, D., Servat, D.: Automatic optimisation of system architectures using EAST-ADL. J. Syst. Softw. **86**(10), 2467–2487 (2013)
13. Bozzano, M., Cimatti, A., Katoen, J.P., Nguyen, V.Y., Noll, T., Roveri, M.: Safety, dependability and performance analysis of extended AADL models. Comput. J. **54**(5), 754–775 (2011)
14. Leitner-Fischer, F., Leue, S.: QuantUM: quantitative safety analysis of UML models. In: Proceedings of QAPL. EPTCS, vol. 57, pp. 16–30 (2011)
15. Boudali, H., Crouzen, P., Haverkort, B.R., Kuntz, M., Stoelinga, M.: Architectural dependability evaluation with Arcade. In: Proceedings of DSN, pp. 512–521. IEEE (2008)
16. Chen, D.J., Johansson, R., Lönn, H., Papadopoulos, Y., Sandberg, A., Törner, F., Törngren, M.: Modelling support for design of safety-critical automotive embedded systems. In: Harrison, M.D., Sujan, M.-A. (eds.) SAFECOMP 2008. LNCS, vol. 5219, pp. 72–85. Springer, Heidelberg (2008). doi:10.1007/978-3-540-87698-4_9
17. Courtney, T., Gaonkar, S., Keefe, K., Rozier, E., Sanders, W.H.: Möbius 2.3: an extensible tool for dependability, security, and performance evaluation of large and complex system models. In: Proceedings of DSN, pp. 353–358. IEEE (2009)
18. Keefe, K., Sanders, W.H.: Reliability analysis with dynamic reliability block diagrams in the Möbius modeling tool. ICST Trans. Secur. Saf. **3**(10), e3 (2016)
19. Bozzano, M., Cimatti, A., Lisagor, O., Mattarei, C., Mover, S., Roveri, M., Tonetta, S.: Safety assessment of AltaRica models via symbolic model checking. Sci. Comput. Program. **98**, 464–483 (2015)
20. Armoush, A., Salewski, F., Kowalewski, S.: Design pattern representation for safety-critical embedded systems. JSEA **2**(1), 1–12 (2009)

21. Ghadhab, M., Kaienburg, J., Süßkraut, M., Fetzer, C.: Is software coded processing an answer to the execution integrity challenge of current and future automotive software-intensive applications? In: Schulze, T., Müller, B., Meyer, G. (eds.) Advanced Microsystems for Automotive Applications 2015. LNM, pp. 263–275. Springer, Cham (2016). doi:10.1007/978-3-319-20855-8_21
22. Junges, S., Guck, D., Katoen, J.P., Stoelinga, M.: Uncovering dynamic fault trees. In: Proceedings of DSN, pp. 299–310. IEEE (2016)
23. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.P.: Performance evaluation and model checking join forces. Commun. ACM **53**(9), 76–85 (2010)
24. Stamatelatos, M., Vesely, W., Dugan, J.B., Fragola, J., Minarick, J., Railsback, J.: Fault Tree Handbook with Aerospace Applications. NASA Headquarters (2002)