Integrating Model Checking with HiP-HOPS in Model-Based Safety Analysis

Septavera Sharvia [a*], Yiannis Papadopoulos [b]

University of Hull, Cottingham Road HU6 7RX, Hull, United Kingdom
[a] s.sharvia@hull.ac.uk
[b] y.i.papadopoulos@hull.ac.uk

[*]Corresponding author. Contact details: Department of Computer Science, University of Hull, Cottingham Road HU6 7RX, Hull, United Kingdom. +44 (0)1482 465253.

Abstract

The ability to perform an effective and robust safety analysis on the design of modern safety-critical systems is crucial. Model-Based Safety Analysis (MBSA) has been introduced in recent years to support the assessment of complex system design by focusing on the system model as the central artefact, and by automating the synthesis and analysis of failure-extended models. Model checking and Failure Logic Synthesis and Analysis (FLSA) are two prominent MBSA paradigms. Extensive research has placed emphasis on the development of these techniques, but discussion on their integration remains limited. In this paper, we propose a technique in which model checking and Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) - an advanced FLSA technique - can be applied synergistically with benefit for the MBSA process. The application of the technique is illustrated through an example of a brake-by-wire system.

Keywords

HiP-HOPS; Fault Tree Analysis; Model Checking; Model-Based Safety Analysis; Failure-Logic Synthesis Analysis; Fault Tolerance

1 Introduction

Modern safety-critical systems frequently involve various engineering disciplines, numerous embedded control components, and employ complex architectures. These systems are widely employed in various industries including the automotive, aerospace, and nuclear industries. However, failures in these systems may result in catastrophic consequences, including the loss of human life. With the increasing scale and complexity, the analysis of the design and development of these systems have become increasingly challenging. Classical safety assessment techniques such as Hazards and Operability Analysis (HAZOP) [1], Fault Tree Analysis [2], and Failure Modes and Effects Analysis (FMEA) [3] have been used for many years to analyze the safety and reliability of safety-critical systems. However, these techniques are traditionally manual applied, and often considered to be no longer adequate to cope with the complexity of modern safety-critical systems because in the context of a large, complex system, manual processes become laborious and error-prone. The safety information and analyses in classical techniques are also often disconnected or loosely connected to system design models, making safety analysis in the context of design iterations difficult. Safety analyses are therefore often performed in the later stage of the design process, thus missing the opportunity to influence the system design.

Model-Based Safety Analysis (MBSA) techniques have been developed in recent years to address the challenges in analyzing and verifying complex safety-critical systems. MBSA focuses on developing effective and robust safety assessment techniques through the automation of the safety analysis process. Semi-formal and formal models are used as the foundation for various development activities, like prototyping, simulation, and code generation [4], making the process integrated and coherent. Various general-purpose modeling languages have been developed over the years to support the development of system design models. These include Systems Modelling Language (SysML) [5], Architecture Analysis and Design Language (AADL) [6], and Embedded Automotive System Technology - Architecture Description Language (EAST-ADL) [7]. Analysis on these models is performed to understand how a system behaves not only in its normal working condition (represented in the nominal model), but also in the presence of failure (represented in the error model [8]). The term error model refers to a model which contains failure information, and describes potential failures of design elements and possibly local effects. The automated analysis in MBSA typically determines global effects of failure of design elements, and the automated nature of the technique means that the results can be more easily generated to reflect changes in the system design, and can be incorporated as part of an iterative design process. This makes the process more efficient and contributes to more reliable results.

In general, the various MBSA techniques proposed and developed over the years gravitate towards two prominent paradigms, Failure Logic Synthesis and Analysis (FLSA) and model checking [9]. FLSA uses a process of composition to construct the system error models from the topology of a system and the local error models of its components. Model checking, on the other hand, uses exhaustive exploration of behavioral models of the system to assess satisfaction of safety requirements.

FLSA and model checking have emerged with little integration. Both techniques are fundamentally different in their objectives of assessment, working mechanisms, and application process. We aim to investigate the combined application of these emerging techniques, and believe that the exploitation of each technique's strengths can bring substantial value to the development process, particularly at the early design stages. Here we propose a technique through which we aim to demonstrate how a systematic, combined application of these two technologies can be performed, and how this may lead to a more robust design assessment process. In this work, HiP-HOPS [10] has been selected to facilitate FLSA, and NuSMV [11] has been selected to facilitate model checking. HiP-HOPS is a state-of-the-art tool that enables dependability analysis and optimization of design models. NuSMV is a model checker tool which enables formal verification. In addition to the higher-level conceptual integration of techniques, this paper also investigates and discusses the integration of HiP-HOPS and NuSMV.

The rest of the paper is structured as follows: Section 2 presents the background to this work and discusses relevant safety analysis techniques. Section 3 presents the proposed technique which combines application of FLSA and model checking in the context of an improved MBSA process. This process involves failure severity analysis, local failure logic definition and annotation, translation of FLSA results into finite state machines, and application of formal verification through model checking. This section also discusses how relevant failure information can be obtained and translated between different models of FLSA and model checking. Section 4 presents an example of a brake-by-wire system which illustrates the proposed technique. We show how failure information and safety artefacts from FLSA are used as the basis for the construction of finite state machines, which can then be verified. This section also discusses how a simple example of control and mitigation procedure of a diagonal-locking mechanism in car wheels can be assessed and verified. Section 5 presents a summary of this paper and outlines future work.

2 Background

2.1 Failure Logic Synthesis and Analysis (FLSA)

In FLSA techniques, individual components in a system model are extended with descriptions of their failure behavior, which are then used to construct the system-level error model. System error models are automatically constructed by establishing how the local component failures combine as effects propagate through the topology of the system. These error models are predictive models of system failure, and are typically produced in the form of well-known safety artefacts like fault trees.

Safety artefacts produced from FLSA are generally familiar to safety engineers, making the process more intuitive. These artefacts allow identification of potential failures and design weaknesses which guide useful design iterations, and help to derive and refine requirements. The process is flexible and adaptable to different stages of model development. Because of its compositional nature, it is easy to determine what the effects of changing one component will have on the rest of the system. This speeds up the analysis and enables effective evaluation on speculative changes to the design, which is important as it enables the assessment to be started early in the design process when concrete system details are still minimal.

FLSA is also often employed to facilitate reliability engineering. For example, with techniques like HiP-HOPS [10], it is possible to not only effectively perform qualitative assessment to obtain root causes of a failure, but also to perform advanced temporal analysis [12], probabilistic quantitative analysis, as well as architectural optimization [13].

Despite these benefits, analyses with FTA and FMEA are generally static, and do not take into consideration the changes in system states and therefore unable to capture dynamic behavior accurately. Techniques which are based upon FLSA approach include: Failure Propagation and Transformation Notation (FPTN) [14], Component Fault Trees (CFT) [15], State-Event Fault Tree (SEFT) [16], and HiP-HOPS [10].

2.1.1 HiP-HOPS

Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) [10] is a safety analysis method which automatically constructs fault trees and FMEAs from system model topologies annotated with component failure information. HiP-HOPS was inspired by FPTN but extends FPTN [14] with more advanced features and automation. FPTN was not originally intended for synthesis and analysis of artefacts like fault trees and FMEAs as in HiP-HOPS and the early papers that preceded HiP-HOPS do not describe such features and algorithms. FPTN was rather intended as a modular specification of failure logic.

HiP-HOPS works in conjunction with commonly-used system modeling tools, such as Matlab Simulink or Simulation X, and has recently been extended to facilitate analysis of AADL models. Failure editors are integrated into these modelling tools to allow system designers to annotate the components with failure information. This failure information includes failure modes (internal malfunction) and output failure expressions, and describes how the component fails and its relationship with other component failures, i.e. whether and how the component responds or not to effects of failure received at the component inputs. HiP-HOPS takes this information and examines how component failures propagate through the system topology, producing sets of interrelated fault trees and eventually an FMEA. This approach also enables the hierarchical structure of the system to be captured neatly in the fault trees.

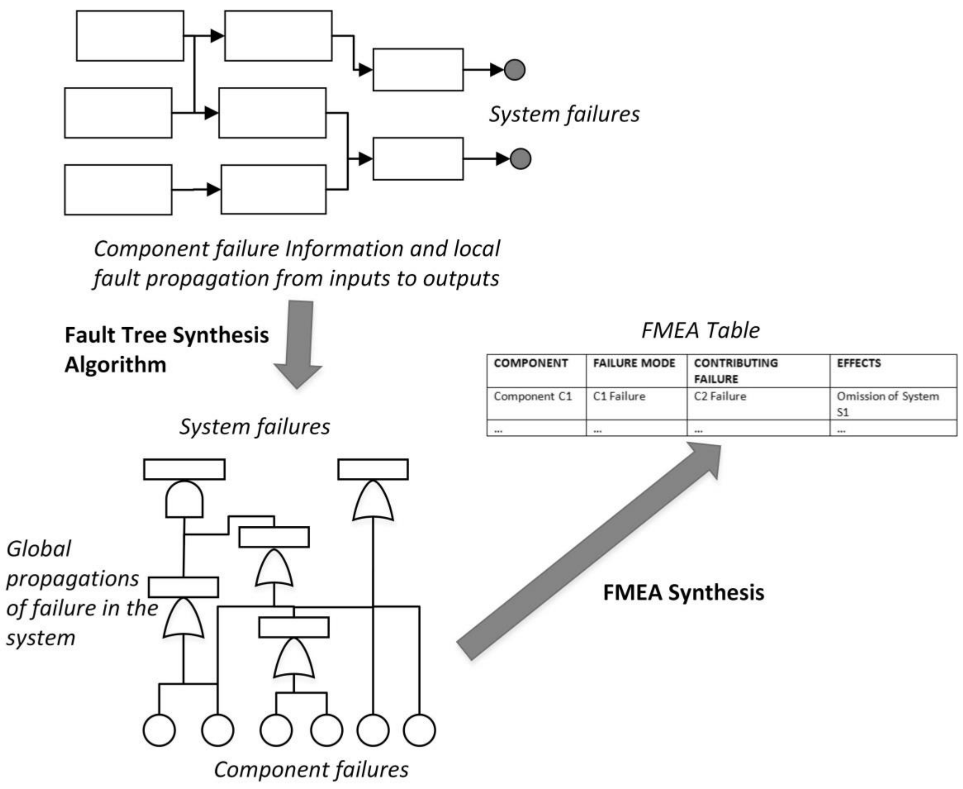
There are three main phases in HiP-HOPS: model annotation, fault tree synthesis, and fault tree and FMEA analysis phase.

The model annotation phase provides information to HiP-HOPS on how the component can fail. It takes the form of a set of expressions which are manually added. These local failure expressions describe how failures of the component outputs can be caused by a combination of failures received at the component's inputs and/or by internal malfunctions of the component itself. Common cause failures are also supported, as are failures propagated via other means, e.g. from allocated components. In this way it is possible to model more sophisticated scenarios — for instance, the effects on a software function when the hardware processor executing that function fails.

The synthesis phase produces an interconnected network of fault trees which link system-level failures (i.e., failures of the system's output functions) to component-level internal failures by using the model topology and component failure information. These fault trees show how the component failures propagate from one component to another and how ultimately they may affect the wider system, whether individually or in combination with other component failures.

In the analysis phase, the synthesized fault trees are analyzed via automated algorithms to generate minimal cut sets. Minimal cut sets describe the necessary and sufficient combination of events which lead to the undesired events. Eventually the data is combined into a multiple failure mode FMEA which shows both direct effects of failure modes on the system, as well as the further effects of the failure modes caused in conjunction with other failure modes occurring in the system. The resultant FMEA is presented in tables which are conveniently displayed through a web browser.

These main phases of HiP-HOPS are illustrated in Figure 1.


System failures
Component failure Information and local fault propagation from inputs to outputs
Fault Tree Synthesis Algorithm
FMEA Table
System failures
Global propagations of failure in the system
FMEA Synthesis
Component failures

COMPONENT	FAILURE MODE	CONTRIBUTING FAILURE	EFFECTS
Component C1	C1 Failure	C2 Failure	Omission of System S1
...	...	...	...

Figure 1. Main Phases in HiP-HOPS

Quantitative data can also be entered for the component to represent the probability of internal failures occurring and the severity of output deviations. This data can then be used in the quantitative analysis phase to calculate the unavailability of the top event. HiP-HOPS assists reusability by enabling failure-annotated components to be stored in a library. This allows other components of a similar type to reuse the failure data, and avoids the designer having to enter the same failure data multiple times. Recently, HiP-HOPS has also been extended with advanced features, including the capability to accommodate temporal analysis and perform multi-objective optimization.

2.2 Model Checking

Model checking [17] [9]is often used to facilitate system formal verification. The system-level effects of failures are established by injecting faults into the formal specification of the system, and the effects on system behavior are observed. The technique is often employed to allow formal verification by ensuring that the system model conforms to specified safety properties. Formal models are expressed as state automata (or finite state machine), while the safety properties are expressed in temporal logic. A model checker tool is used to perform exhaustive exploration of the state space in order to assess whether a specified property holds for the system. When the safety property does not hold, a counterexample is produced to show the traces of simulation on how the breaching situation is reached.

The ability to facilitate automated formal verification and capture system dynamic behavior makes model checking a powerful and rigorous approach in assuring that certain critical properties are guaranteed in a complex system. It is also possible to differentiate between transient and permanent failures and model the temporal ordering of failures.

However, limitations of model checking include the fact that model checker tools generally require the system model to be expressed in the particular model checker input language. Useful safety artefacts like fault trees which are produced from model checkers in this approach generally have 'flat' structures, effectively a two level "OR-AND" structure representing the disjunction of all minimal cut sets. Therefore the tree is only two levels deep and can be very broad, and this can hamper the understanding of fault propagation through the system. Formal models - used as the input to the model checker – are usually developed only at the later stage where designs are more mature and stable. Approaches which are based on model checking also tend to be more computationally expensive and inductive in nature. This means that the exhaustive assessment of the effects of failure combinations can be potentially infeasible in larger systems.

Examples of techniques which are based on model checking, include Altarica [18] and Formal Safety Analysis Platform/New Symbolic Model Verifier (FSAP/NuSMV) [19]. The NuSMV model checker has been selected here as a good example of this technique to facilitate the integrated analysis with HiP-HOPS.

2.2.1 NuSMV

The New Symbolic Model Verifier (NuSMV) [11] is a symbolic model checking tool which checks a finite state system against specifications in the temporal logic Computation Tree Logic (CTL), and recently, Linear Temporal Logic (LTL). NuSMV is a newer version of Symbolic Model Verifier (SMV [20]).

The NuSMV language allows components of the system to be described in a modular hierarchical way. This means that each of the component can be described as a separate module in NuSMV, and links between these modules preserve the hierarchical structure (i.e. system and subsystem can be described clearly). System behavior is described through variables within the modules. Modules are generally used to define or distinguish separate physical (sub) systems or components.

A NuSMV `Module` can consist of a set of variable declarations, assignments of variable initial values and definitions, and property assertions. Data types used are confined to finite type (for example, Boolean, scalar, enumerated or fixed array of basic data types). The `Assignment` section contains a set of assignments of `Variables` into their initial value or the value in the next execution step. Various operators are available for variable assignments, including Boolean logic, conditional, arithmetic and comparison operators. The transition relation is described using expressions in propositional logic which outlines the variable update rules. The set of reachable states (if the activation condition in the update rules is true) is non-deterministic. An `Assertion` section is used to define CTL properties, which should hold over all executions.

NuSMV has been selected as a technique to complement HiP-HOPS based on the following considerations. Firstly, NuSMV is fundamentally a symbolic model checker. Symbolic model checkers are generally more scalable and therefore, are recommended for larger real-life systems. NuSMV is also more suitable for Boolean-based data (as opposed to enumerated type) [21]. Considering that most of the failure data obtained from HiP-HOPS are Boolean-based, NuSMV is a logical choice. The NuSMV support tool is also available as an open-source program which allows it to be tailored more effectively into a future integrated support tool.

3 Integrated Application of HiP-HOPS and Model Checking

The integration between HiP-HOPS and model checking aims to exploit the quick iterative nature of FLSA and the verification capability of model checking. The initial analysis results from FLSA are used to help derive a more-robust model prior to the application of model checking. The method can be iterated until a satisfactory design that fulfills safety requirements is reached.

The key steps involved in this process are illustrated in Figure 2. The method starts with the construction of a system model and a set of safety specifications. This model can be an early functional model or a more detailed architectural model depending on the stage of the system development. In the early functional design, the method is applied to an early model where design details are not mature, but sufficient enough to show input, processing and output functions, as well as the dependencies. In the next step, this model is examined further to evaluate the severity of failures in the output functions (or components). Local failure logic for each function (or component) is obtained and, once components have been augmented with that logic, one or more system fault trees for system level failures recording causes of these failures are synthesized and analyzed. A result of this process is also the synthesis of a system FMEA. This FMEA is then studied and interpreted, usually leading to recommendation for design improvement or additional safety measures. With the introduction of new safety measures, the requirements and system model may be updated. The severity of the failures and the failure data annotation are revised, and the next iteration of FTA and FMEA may be performed. When deemed satisfied, the results of FMEA can be analyzed and used to assist the construction of abstract state machines to represent the system dynamic behavior. Failure modes and effects in the FMEA are used to determine in a state machine transitions from normal states or modes of the system to degraded and failed states and modes. Model checking is then used to verify whether this dynamic system model conforms to the requirements. If the conformity is verified, the process proceeds to either further refinement of the model and iteration of the above process or its implementation. Otherwise, counter examples are produced to show how the model fails to fulfill certain requirements.

In this paper, focus is drawn to the benefits yielded by the method in enabling systematic construction of behavioral models from the FLSA output, and then useful application of model checking on such models. This technique aims to show how the integrated analyses and their results can help transform a basic initial functional (or early architecture-allocated functional) model of the system into a more robust, better prepared model which eventually becomes the foundation of the development of the system and its operating logic.

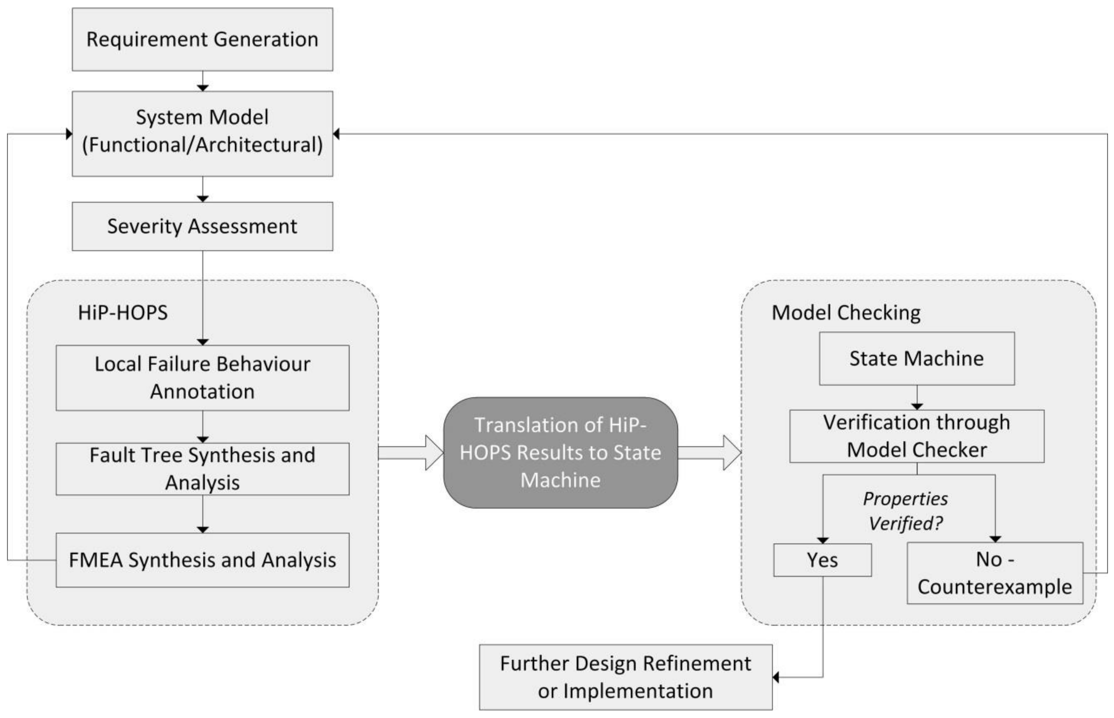

Requirement Generation
System Model (Functional/Architectural)
Severity Assessment
HiP-HOPS
Local Failure Behaviour Annotation
Fault Tree Synthesis and Analysis
FMEA Synthesis and Analysis
Translation of HiP-HOPS Results to State Machine
Model Checking
State Machine
Verification through Model Checker
Properties Verified?
Yes
No - Counterexample
Further Design Refinement or Implementation

Figure 2. Integrated Application of HiP-HOPS and Model Checking

3.1 Modelling and Failure Severity Assessment

HiP-HOPS can be performed on any model of a system which identifies components and the data, material or energy flows among these components. The model can be a hierarchy of subsystems enclosing architectures of lower level elements to manage complexity, and it may be in the form of a functional or architectural model, i.e. components can represent functions or elements like sensors processors or actuators. In the early stage of the development process, a functional model is typically constructed to reflect how a series of 'processes', which need to be performed to fulfill the system requirements, are organized. The model becomes increasingly detailed as the development progresses and architectural information emerges.

Once a system model is constructed, the severity of failures in the system outputs needs to be examined. Components delivering these outputs interact directly with the environment of the system. Environment of the system may refer to the users or other external elements outside the system boundaries.

The categorization of failures in terms of severity is based upon the guidelines presented in the IEC 61508 [22], as shown in Table 1. The severity of failures can be classified into the following categories according to their consequences to human stakeholders or in general the quality of service provided: *Catastrophic*, *Critical*, *Marginal*, and *Negligible*.

Table 1: Classification of Failure Severity

Description	**Consequence to human stakeholders**	**Consequence to service**
Catastrophic	Fatalities and/or multiple severe injuries	-
Critical	Single fatality or severe injury	Loss of major system
Marginal	Minor injury	Severe system damage
Negligible	Possible minor injury	System damage

The understanding of the effects a failure, and hence the assignment of the severity class, can be obtained from simulation, testing, experience or consultation with domain experts. The classification can be assigned as part of the information presented in the Functional Failure Analysis (FFA). FFA primarily aims to identify hazardous functional failures and presents this information in a tabulated format, where the analyzed functions, failure type, effects on the system, severity, detection, recovery plan, and recommendations are organized.

The assessment of failure severity allows the analysis process and resources to be focused effectively, particularly when conflicting priorities exist in the design. System failures which have been classified under 'catastrophic' and 'critical' categories naturally need to receive higher priority than those with 'marginal' or 'negligible' effects.

3.2 Local Failure Information

In addition to evaluating the severity of system output failures, it is also important to determine the causes which lead to these failures (particularly the catastrophic and critical ones). Qualitative analysis identifies the root causes and can provide valuable feedback towards improvement of the design architecture by pinpointing weak parts in the system model.

As mentioned earlier, for the purpose of the analysis, each component is annotated with failure information in the format suitable for HiP-HOPS. This failure information defines each component's failure behavior including the failure logic. Failure logic describes how the component itself can fail, and how it reacts to failures propagated from other components. Essentially, this information defines how failure in the component output can occur due to the internal failure within the component itself or corresponding deviations received at component inputs. This is done by annotating the model with failure expressions which define the failure logic that causes component output deviations.

The classification of different types of deviation is typically assisted through the use of guide words similar to those used in HAZOP. These include: *omission*, *commission*, *value* and *timing* failure. The *omission* refers to failure of the component to provide the intended output; a *commission* failure refers to unintended provision of output; a *value* failure refers to incorrect output value provided; and a *timing* failure refers to output provided later or earlier than intended.

In its basic form a failure expression contains information of the failure type and the location where it occurs (i.e. the input/output ports). The dash symbol "-" is used to separate the failure type from the input or output parameters, and the failure causes are connected by logical operators. Commonly used logical operators in HiP-HOPS include the disjunctive operator and the conjunctive operator but extensions also allow temporal operators such as "before" and at the "same time".

As an example, a failure expression in HiP-HOPS can be expressed as the following:

```
Omission-OutputPort = InternalFailure OR Omission-InputPort
```

This expression defines that a deviation in the component output port (`Omission-OutputPort)` can be caused by an internal malfunction of the component (`InternalFailure`) or an omission in the component input port (`Omission-InputPort)`. Component failure expressions are provided manually at the component level and focus on the output ports through which a component provides services to other components in the system. In the analysis process, each output port is systematically examined for potential deviations of port parameters from the intended normal behavior.

This component failure behavior annotation, in addition to the topology of the system architecture, allows system-level failure logic to be developed and analyzed.

3.3 Fault Tree and FMEA Synthesis and Analysis

The global view of the failure propagation in the system architecture can be captured by traversing and following the causal links defined in each component local failure logic. The process starts from a failure in a system output and systematically moves backwards towards system input components. The fault tree is incrementally generated by progressively replacing the input deviations for each component with the corresponding output deviations propagated by other components. The logical structure of the tree is determined by local failure logic and interconnections between components. This process produces a set of fault trees that represent the relationships between system output failures and their root causes in the system.

Fault trees in HiP-HOPS can be analyzed qualitatively and quantitatively, and the results can be summarized in an automatically generated FMEA table. Traditional FMEA table shows only the *direct effects* of a single failure on the output components. But because of the way the FMEA is constructed by HiP-HOPS from a series of fault trees, it also captures the effects of a component failure when it occurs in conjunction with failures from other components. These are termed *further effects* of the component failure.

Details on HiP-HOPS can be found in [10, 23, 24] . To facilitate readability, and without going into too much detail, here we discuss a small example. Figure 3 illustrates a simple standby-recovery system. It contains one input function called Input, and two processing functions called Primary and Standby. Primary receives and processes the value generated by Input. A redundant function Standby monitors the output of Primary, and when an omission failure at the output Primary is detected, Standby is initiated to replace Primary. Primary and Standby functions supply to the Normal and Backup system outputs respectively.

For simplicity, Input, Primary and Standby functions have one internal malfunction each, called InputFailure, PrimaryFailure, StandbyFailure respectively. Table 2 presents the failure expression which describes the failure logic for functions Input, Primary and Standby. Failure logic has been simplified for the purpose of the example, therefore there is no analysis of commission or timing failures.

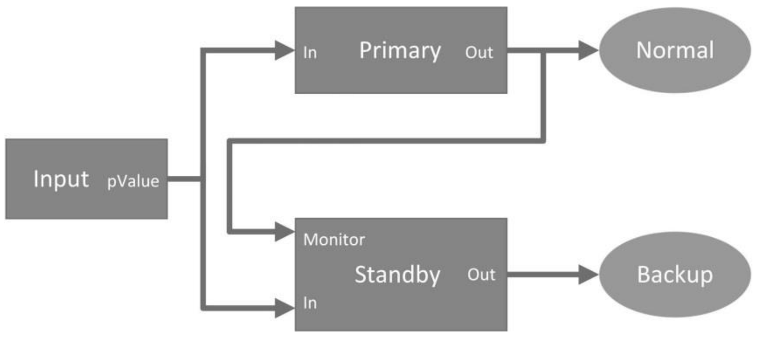


Figure 3. Standby-recovery System

Table 2: Failure Expressions for Standby-Recovery System

Function	**Output Deviations**	**Failure Expressions**
Input	Omission-pValue	InputFailure
Primary	Omission-Out	Omission-In OR PrimaryFailure
Standby	Omission-Out	Omission-Monitor AND (Omission-In OR StandbyFailure).

Components of the model are augmented with their respective failure logic within the modelling environment (e.g. this could be Matlab Simulink). From these expressions, and the structure of the model, the HiP-HOPS safety analysis tool generates the fault trees shown in Figure 4 and FMEA illustrated in Table 3. The FMEA shows that a failure of the input function Input causes omission of both the Normal and Backup outputs of the system, and it is therefore, a critical failure. On the other hand, a single failure of Primary causes only omission of the Normal output and therefore can be deemed as less critical. Finally, the failure of Standby does not have any direct effect on the system. It becomes significant only in conjunction with failure of Primary, which in this design provides the condition that precisely triggers the need to deploy Standby in the first place.

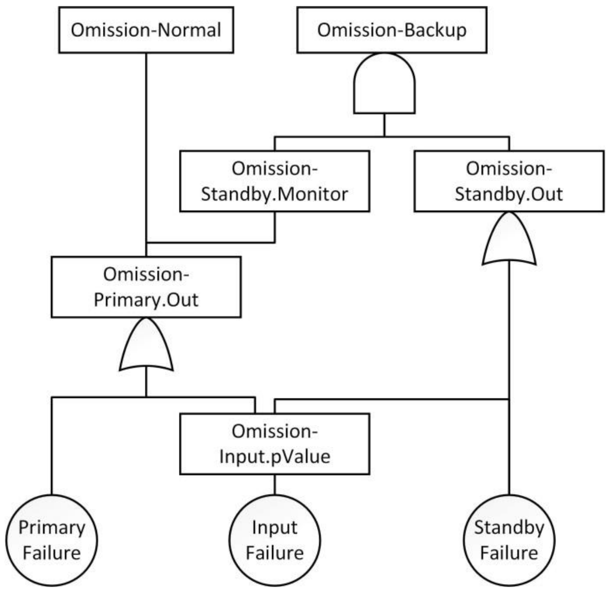


Figure 4. Fault Tree for Standby System

Table 3: FMEA Table for Standby-Recovery System

Component	**Failure Mode**	**Direct Effects**	**Effects in Conjunction with [Other Events]**
Input	InputFailure	Omission-Normal Omission-Backup	-
Primary	PrimaryFailure	Omission-Normal	Omission-Backup [StandbyFailure]
Standby	StandbyFailure	-	Omission-Backup [PrimaryFailure]

The FMEA indicates that failure of Input is indeed a direct cause of a critical system failure and should be prevented by design. The analysis shows that an independent failure of either Primary or Standby cannot cause a critical system failure. Emphasis of the design therefore has to be placed on how to protect these two functions from common cause failures (for example, failures caused by electromagnetic interference).

This small example demonstrates the ability of the synthesis tool to detect hazardous dependencies in the model, i.e. component failures that may cause simultaneous failure of hypothetically independent system functions. This may seem a trivial task in this example largely because the source of the dependency is very close to the affected functions but also because the model and associated failure logics are very simple. However, hazardous dependencies are not always as simple to detect especially those originating from remote data sources which are deeply embedded in the hierarchy of complex designs. The detection of such dependencies is, indeed, a hard task which justifies the need for efficient automated support.

The FMEA table generally provides a list of components, failure modes, effects of the component failures on system outputs, and other contributing failures that in conjunction cause further effects. It is also possible to include additional information, such as the severity of the affected output component, recommended treatments, and other general comments. The FMEA table essentially shows how internal failures of components can contribute to hazardous system output failures. Interpretation of the table allows failure information of the system to be checked against safety requirements, and safety measures to be devised when necessary. Focus is placed on the components whose failures contribute to catastrophic system failure, as they need to be prevented by design, or have their impacts minimized. While the design solution ultimately relies on sound engineering judgment, the identification of the criticality of each component assists the management of effort allocation and design modification. Modification of the system structure, for example through incorporation of redundant components, is often necessary. This brings about new components to the system, changes to the architecture, as well as introduction of new failure behavior modes and changes to the failure logic of the system. Subsequently, there is a need to iterate the process to regenerate and update FTAs and FMEAs.

From this process, more-refined design and safety requirements can be derived. The identification of lower level failures leading to output failures can be evaluated, and this helps derive more refined design requirements. This will, in turn, result in fewer late design changes in comparison to traditional practice.

3.4 Translation of HiP-HOPS Results into State Machine

3.4.1 Abstract State Machine

To integrate the application of FLSA and model checking techniques constructively, we need to establish an effective association between the output of HiP-HOPS (i.e. FTA and FMEA results) and the input of model checking (i.e. state machine). Iterative application of the proposed process starts from an *abstract* state machine and gradually becomes more *refined* as additional details become available and incorporated to behavioral models that can be subject to verification.

Abstract state machines (sometimes termed ‘mode charts’) are typically constructed at early stages, where dynamic behavior can be expressed as a set of different functional states of the system and transitions between them. A functional state (or *mode*) [25] is defined by the set of functions delivered by the system in this state. Generally, types of states used in the modeling of an abstract state machine in our approach include *normal, degraded* and *failed* states. The system is said to be in normal state when it delivers its predefined set of functions. A degraded state describes the condition where a system delivers part of the intended functionality safely, whereas a failed state refers to the condition where there is complete loss of function or the system behaves in an unpredictable or hazardous manner. This implies that in cases when the system loses even only part of its functions, if the lost function happens to be critical and has catastrophic effects on the system as a whole, the system is said to be in the failed state.

If necessary, degraded and failed states can be further decomposed into subcategories according to the general fault classification criteria. For example, different types of faults can be categorized based on criteria such as activity (latent and active), duration (transient and permanent) or causes (random and generic) [26]. A latent fault is dormant, whereas an active fault is actively generating errors. A transient fault is temporary, whereas a permanent fault is lasting. A random fault may affect only a certain component, whereas a generic fault is systemic. Here, our interest leans towards time-based classification, where degraded and failed states can be categorized based on whether they are *temporary* or *permanent*. A degraded or failed state is temporary when action can be taken to restore the system back to a normal state, while permanent states occur when recovery is no longer possible.

The FMEA results gathered from application of FLSA essentially show the effects of component failures, and then help decide whether the failures can be tolerated. Intolerable failures must be managed by appropriate modification of design or external detection, and recovery mechanisms must be put in place. Tolerable failures may be allowed to happen, often resulting in the loss of functionality, but always in the context of a graceful transition to degraded-but-safe states of operation for the system. In the case of tolerable failures, FMEA results lead to design solutions that introduce, or reinforce, these new degraded states. Abstract state machines can therefore be constructed to show how graceful transition to the identified degraded states could be achieved, and how they can act as potential buffers to divert hazardous failures.

An FMEA-ModeChart Assistance Table is used to organize the construction of abstract state machines by identifying and organizing the key elements of the state machine – which include system states, severity of each state, output functions delivered, failure event(s) triggering the transition, and target states(s). Table 4 shows an example of simple FMEA-ModeChart Assistance Table on the basis of which an abstract state machine can be constructed .

Table 4: Example of FMEA-ModeChart Assistance Table

State	Severity	Functions Delivered	Failure Events Causing Transition	Target State
Normal	-	Function F1	Omission F1	Degraded
Normal	-	Function F2	Omission F2	Failed
Degraded	Marginal	Function F2	Omission F2	Failed
Failed	Critical	-	-	-

When further behavioral information on low level components becomes available in an incremental development process, it is possible to refine the transition events (particularly *guards*) in the abstract state machine. This allows good traceability between abstract and refined state machines as it improves the understanding on how transitions in a more refined level affect the higher level state of the system.

The refined state machine can often be constructed directly from a HiP-HOPS model, as most information required for failure-relevant transitions can be obtained from the HiP-HOPS component failure annotations. Behavioral information, e.g. normal states and normal transitions between them which is not obtained from the HiP-HOPS model can be independently included.

It is important to note that our state machines are not tied to any commercial state chart tools, and both abstract and refined state machines are directly represented as NuSMV model. Commercial state chart tools, for example Matlab Stateflow or Statemate, can be used to provide graphical description. Alternatively, converter tools, for example stm2smv [27], or mdl2smv [28] are available to convert state machines from these commercial tools into model checking input models (particularly, SMV models).

Each component is represented as a module in NuSMV, and the information flow between components can be represented through the use of module parameters. These parameters provide links between the output (port) of a source component to the input (port) of a target component. In a similar manner, these input parameters are also used to relay the failure propagation between components. Internal malfunctions too are defined within modules. Once an internal malfunction occurs, it is assumed to be persistent throughout the entire run, unless a restoration event is explicitly specified and triggered. All internal malfunctions are initialized as false (components are assumed to be in normal operating condition on the start of the run). Example of initialization of the internal malfunction called "Component1Failure" is `init(Component1Failure):=0;`. A failure expression defining the output deviation (obtained from a HiP-HOPS failure annotation) can be assigned within the module. Input deviations are specified in the module input parameters. The mapping between HiP-HOPS and NuSMV elements is summarized in Table 5:

Table 5: Elements Mapping Between HiP-HOPS and NuSMV

HiP-HOPS	NuSMV
System, Component	`Module`
Input deviation	`Module` parameters
Internal malfunction	`Variables` declarations
Failure expression	`Assign` Constraint
State, Transition	`Assign` Constraint

As a summary, the process of translating FTA/FMEA results into state machines is illustrated in Figure 5.

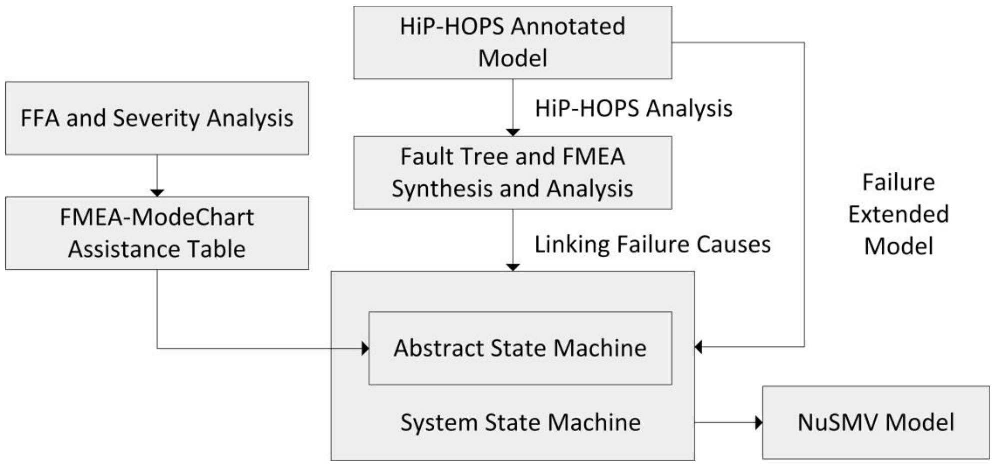


Figure 5. Translation of FMEAFTA Results into State Machine

3.4.2 Transition Events Refinement

Refinement refers to the process of providing a system solution with more details or precision in an incremental development process. This includes the process of adding constraints and developing details of system and component attributes. The refinement of a model may affect both structural and behavioral elements of a system. The refinement of a design often traverses abstraction levels and captures subsystems. A systematic management of decomposition can help to provide good traceability. This in turn, allows designers and analysts to follow the evolution of the design more closely and establish connections between earlier and later design models. One way to achieve this is through clear communication and linking between events in the earlier abstract design and those in the later design. Well-established traceability is particularly useful in situations where errors are discovered through model checking later on. In this case it is possible to trace errors back to an earlier design decision and eventually investigate and re-evaluate their effects on high-level design assumptions and goals. A systematic refinement of transition events (particularly events relating to failure logic) contributes to a better traceability.

One way to refine a transition event is by replacing it with the logical expression of its causal events. As the failure-related transition events are generally loss of functions or component malfunctions which form the top events of fault trees in HiP-HOPS, causal events can be effectively obtained and mapped from HiP-HOPS FTA/FMEA results. For each top event, its minimal cut sets essentially form the replacement expressions. This approach works well in a situation where focus is placed more on the verification of the abstract behavioral states of a system compared to the behavior of the refined individual subsystems enclosed within the system. This usually means that the verification process aims to explore the effects of the lower level subsystem failures on the modes changes at the higher level. One advantage of adopting the results from FLSA is the easy representation of both deviations and component basic events in NuSMV as Boolean data types.

Another approach to the refinement of transition events is through compositional annotations – i.e. by modeling each subsystem behavior in its own module. This approach captures the functional hierarchy by constructing independent state charts and NuSMV modules for each subsystem, which in turn allows nominal behaviors of each subsystem to be effectively modeled and considered in their contribution to system failures. To effectively link failure behavior to input modules and capture the structural information, transition events are expressed in a structure similar to HiP-HOPS failure annotations.

3.4.3 Translation from HiP-HOPS to NuSMV Models

The construction of NuSMV models from HiP-HOPS models can be achieved by mapping the failure information as discussed in section 3.4.1 (mapping between HiP-HOPS and NuSMV elements), a process which results in a failure-extended NuSMV model. The following steps outline the translation process from HiP-HOPS to a NuSMV model.

For each HiP-HOPS component, a NuSMV module is created within which the following steps are performed:

Step 1: Identify input parameters

The input parameters of a NuSMV module are the input deviations of that component. The publically accessible HiP-HOPS XML specification of the system with its failure annotations and the data structures of the tool readily provide this information.

Step 2: Declare the internal variables

Internal variables which can be assigned automatically from HiP-HOPS models typically include the internal malfunctions, output ports, and output deviations. These are declared as Boolean data type.

Step 3: Specify initial values for the internal variables

The initial value of internal malfunction and output deviations are set to false by default, reflecting the assumption that the system starts from a normal initiating state.

Step 4: Define output deviations

Each output deviation is defined according to the failure expression provided in HiP-HOPS annotation. It is described in terms of a logic that connects basic events and input deviations.

Step 5: Specify next value for internal variables

The `next` notion in NuSMV relates the current and next state variables to express transitions. As mentioned previously, once an internal failure occurs (set to true), the `next` value stays at true as it is persistent throughout the entire run. The `next` value of an output deviation can also be defined here in relation to current value of an internal malfunction and input deviations.

In addition to modules which represent components in the system, a `Main` module is also constructed for each NuSMV model to:

- Construct instances of all component modules.

- Define the connections between component modules. This is achieved by connecting the parameters of each component's input ports (and supplying them as input parameters) to the corresponding output ports of other modules which are connected to it.

The refinement of the state machine transitions through minimal cut sets works by constructing one `Main` NuSMV module to include the internal malfunctions of all components. The initial state of the system is set to Normal, and all internal malfunctions are set to be false. The output deviation is defined in terms of its minimal cut sets generated from the FTA, as opposed to defining it in terms of input deviation and internal malfunctions.

The refinement through compositional annotation can be achieved using the algorithm described above where one NuSMV module is constructed for each component. This way, the structural, hierarchical and failure propagation information is retained. This is particularly useful for refined state machine where establishing failure connections between components is crucial.

The above process can be fully automated. However, to fully utilize the analysis capabilities of model checking, manual intervention may be required to include additional information which is not captured in the initial HiP-HOPS model. This additional information generally pertains dynamic behavior, and may include:

- Description of states and transitions: different system states will need to be manually specified, e.g. various different normal states of the system that may entail different failure logic associated with them. These states are important in the understanding of system behavior and failure logic. Such states can be separately analysed via HiP-HOPS as explained earlier. Normal transitions between these states, and their corresponding trigger events will need to be defined.
- Requirement specification: the requirement specification needs to be manually provided by the analysts in terms of CTL or LTL.

3.5 Model Checking and Verification of Safety Properties

Once the state machines have been automatically or semi-automatically constructed and translated into a model checking input language, system requirements can be verified. A Model checker confirms whether the system model satisfies its properties, and produces counter examples if a property is breached. Based on the results of the model checking, analysis takes place to determine whether modifications are required in the design of the model or in the formulation of desired properties.

System specifications and requirements can be expressed in temporal logic, in our case CTL (Computation Tree Logic). These requirements can be classified into different categories of properties, for example: reachability, safety, liveness and fairness properties [29]. Reachability properties define that a particular configuration φ (a state in the Kripke structure) of the model can be reached. Safety properties define that under certain conditions, configuration φ shall not occur. Liveness properties require that something 'good' must eventually happen. Fairness properties define that under certain conditions, states where configuration φ holds will occur infinitely often. Fairness properties are expressed in LTL, and in NuSMV, fairness constraints can be introduced with the inclusion of "`FAIRNESS φ`".

Considering the fact that the general application of model checking has been primarily targeted at mature design models, it is important to understand its value at earlier design stages. Case studies have been presented in [21] and [30] to demonstrate that formal models can be used effectively to find errors before the implementation of the system. One common error found through model checking is the inaccuracy in the original requirement (or how it was expressed). This generally leads to the modification of the requirement into a more accurate form. The importance of model checking in detecting hazardous interactions between system features (e.g. software components) is also highlighted in [28]. These software components which control mechanical components are often developed in isolation, and their combination can sometimes result in unexpected or undesired system behavior. Model checking can be used to detect these hazardous combinations in the design stage.

The benefit of the proposed integration of techniques in the above scheme is that HiP-HOPS analysis can guide the design of the system and its behavioral models by helping to identify degraded and failed states in response to failures and possible responses to such situations. Models of this behavior can then be subject to analysis via model checking. In our approach, more than one normal states and transitions between them can be the subject of the proposed analysis. Such states can in theory model the operating logic of the subsystems. We should note that this (normal) operating logic can also be faulty by design, i.e. contain bugs, and this is also subject to examination through model-checking. However, work on the verification of normal operating logic has already been the topic of extensive investigation and is therefore out of the scope of this paper, which is focused on how to extend such models with degraded and failed states and transitions to and from these states.

3.6 A Generic and Abstract High-Level Example of Translation

This section presents a generic and abstract high-level example to illustrate the translation from FTA/FMEA results to a state machine. Figure 6 shows a functional architecture which produces three output functions: F7, F8 and F9. For simplicity, we assume that every function has a single output failure - omission - and that this is caused by internal failure of the function or the omission of its input. Severity assessment is assumed to have identified that the severity of omission failure in output function F8 is hazardous, while the severities of failures in F7 and F9 are marginal.

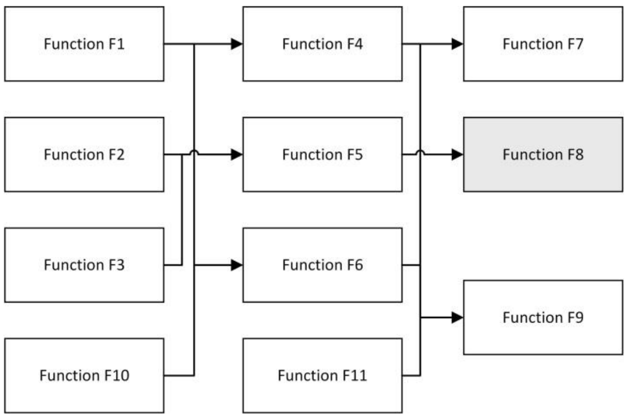


Figure 6. Example of Functional Architecture

HiP-HOPS analysis of the model produces the fault tree shown in Figure 7 and identifies the root causes of omission failure in F8. The fault tree is analyzed and an FMEA table (as partly shown in Table 6) is generated. The FMEA table identifies those functions (function F2, function F3 and function F5) whose failures contribute to omission failure of function F8.

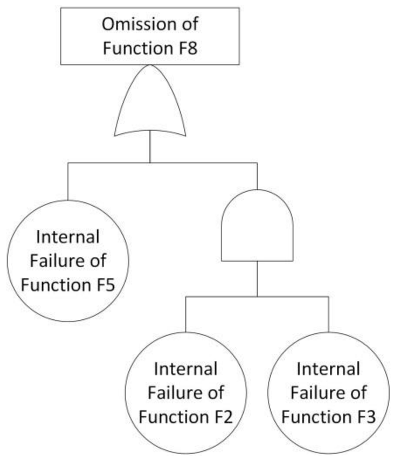

Omission of Function F8
Internal Failure of Function F5
Internal Failure of Function F2
Internal Failure of Function F3

Figure 7. Example Fault Tree

Interpretation of the table allows the failure logic behavior of the functional architecture to be checked against safety requirements. By examining the FMEA table, safety measures can be devised. While the design solution ultimately relies on the engineer's decision and experience, this identification of criticality for each function offers assistance in the management of effort allocation and design modification. For example, apart from identifying that focus should be placed on function F2, F3 and F5 due to their failure criticality, the fault tree also shows that function F5 is a single-point of failure that might need additional attention.

Table 6: Excerpt of FMEA Table

Function	Failure Mode	Effects	Contributing Failure	Severity
Function F2	Internal Failure	Omission of Function F8	Internal Failure in Function F3	Catastrophic
Function F3	Internal Failure	Omission of Function F8	Internal Failure in Function F2	Catastrophic
Function F5	Internal Failure	Omission of Function F8	-	Catastrophic

As previously explained in section 3.4.1, an FMEA-ModeChart Assistance Table can guide the construction of the state machine. The FMEA-ModeChart Assistance Table for this example is shown in Table 7. To construct the state machine, abstract states need to be identified. As an example for this simple system, the first state identified is System_Normal, where all output functions (function F7, F8 and F9) are delivered. Each output function is susceptible to an omission failure which results in the inability of the system to deliver the particular function. As function F8 has been identified as a critical function, this brings us to the second state, System_Degraded. In System_Degraded mode, output function F8 is delivered regardless of the condition of function F7 or F9. System state goes to System_Failed when omission in function F8 occurs. It is also possible to include other degraded states to further define the delivery (or not) of function F7 or function F9 if necessary.

Table 7: FMEA-ModeChart Assistance Table for the Example

State	Severity	Functions Delivered	Failure Events Causing Transition	Target Mode
System_Normal	-	Function F7	Omission of Function F7	System_Degraded
		Function F8	Omission of Function F8	System_Failed
		Function F9	Omission of Function	System_Degraded

			F9	
System_Degraded	Marginal	Function F8	Omission of Function F8	System_Failed
System_Failed	Hazardous	-	-	-

The resulting state machine is shown in Figure 8 (a). As previously explained, the transition event for "Omission of F8" can be refined with its root causes (as identified through the FTA). This is illustrated in Figure 8 (b). Please note that emphasis of the example is placed on the root causes of F8. Omission of function F7 and F9 too can be refined accordingly when FTA is performed on these functions. As the system design develops and complexity grows, state machines can be enhanced and refined. Various components and subcomponents may be allocated to fulfill each of these functions. In turn, each of the components may have multiple states. The fault trees will develop to be more complex and so do the transition triggers. Representation of the state machine in a model checking input language allows safety properties to be verified throughout different levels of hierarchy. Model checking will play an important role in ensuring that safety properties are upheld throughout the interactions between system and component state changes.

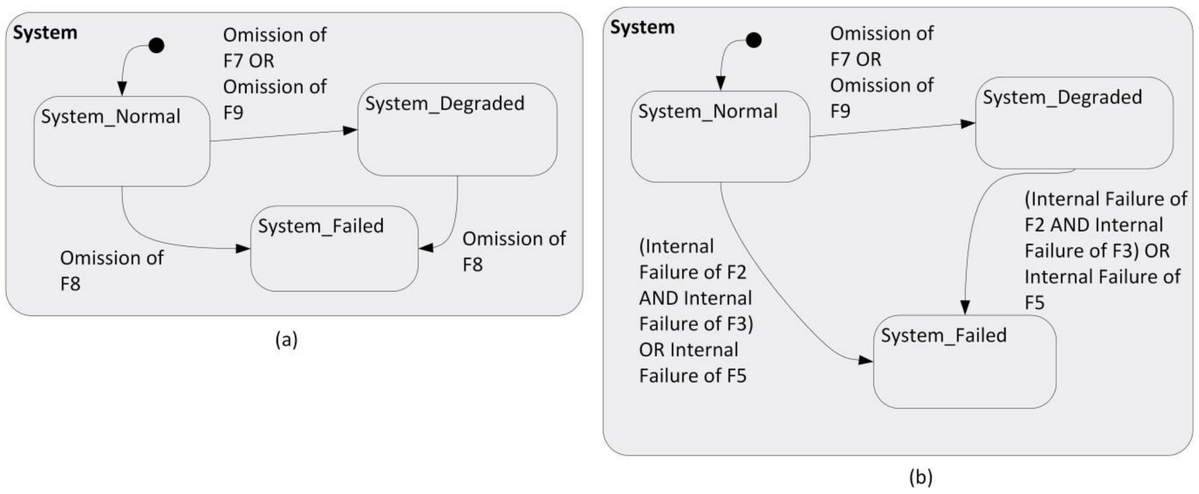

System
Omission of F7 OR Omission of F9
System_Normal
System_Degraded
System_Failed
Omission of F8
Omission of F8
(a)
System
Omission of F7 OR Omission of F9
System_Normal
System_Degraded
(Internal Failure of F2 AND Internal Failure of F3) OR Internal Failure of F5
(Internal Failure of F2 AND Internal Failure of F3) OR Internal Failure of F5
System_Failed
(b)

Figure 8. Example of State Machines

4 Case Study: Brake-by-Wire System

This case study demonstrates the application of the proposed integrated technique in the design of a brake-by-wire system. Brake-by-wire systems replace traditional automotive braking components with electronic sensors and actuators. The hypothetical brake-by-wire system used in this example is based upon a model presented in [31], but also draws from designs in [32] and [33]. The system consists of one vehicle-level processing function and four local-level wheel processing functions. The vehicle-level processing function reads in the braking command input from the driver, communicated through a braking pedal, and subsequently generates braking command for each local-level wheel processing function utilizing advance braking functions such as an Anti-Lock Brake System (ABS). Local-level wheel processing functions are located physically close to the wheels and control the provision of the braking energy. Upon receiving the braking command from the vehicle-level processing function, each local-level processing function calculates the value of the braking pressure, taking into consideration various local information, including actuator position and speed. The value of braking pressure is subsequently fed to an actuator which applies the pressure on the corresponding wheel of the vehicle.

The initial discussion of the system and an early functional analysis are presented in [34]. There, omission and commission failure for basic breaking and ABS-assisted braking functions are analyzed. FLSA is used to identify critical components whose failure leads to hazardous system failure. The analysis results lead to design recommendations and introduction of backup mechanisms. An abstract state machine is constructed – supported by FLSA results - to model and analyze system functional degradation.

In this paper, we extend the analysis to illustrate the iterative application of the proposed method in a more detailed phase, more specifically in an architecture-allocated functional modeling phase. An architecture-allocated functional model extends a purely functional model by taking into account system architecture and representing the allocation of functions to architectural elements without going into fine details of the architecture. Figure 9 illustrates the architecture-allocated model of the system for the corresponding four wheels of the vehicle.

This model extends the earlier functional model [34], and allocates functions to components. The Vehicle-level processing function is assigned to an ECU (Electronic Control Unit). Each local processing function is allocated to a BCU (Brake Control Unit) which, together with an actuator, are assigned to each wheel. It is common that multiple architectural components are assigned to perform a single function, or for a single component to be shared between multiple functions.

Figure 9 illustrates the system model. It consists of input components Input_brakeDemand, Input_wheelSpeedSensor and Input_externalSensors; they provide the driver demand from the brake pedal, readings for the wheel speed, and external variable reading respectively. This information is passed to the ECU, which calculates and generates independent braking and ABS commands for each BCU. BCUs read the corresponding wheel actuator position and speed sensor, and provide the correct value of braking pressure applied by each actuator. Abbreviation FL indicates front-left wheel, FR indicates front-right wheel, RL indicates rear-left wheel, and RR indicates rear-right wheel. The HydraulicBackup component has been introduced to provide emergency braking in the presence of failure that affects electrical-based functions. Table 8 summarizes the input and output flow explanation for the system.

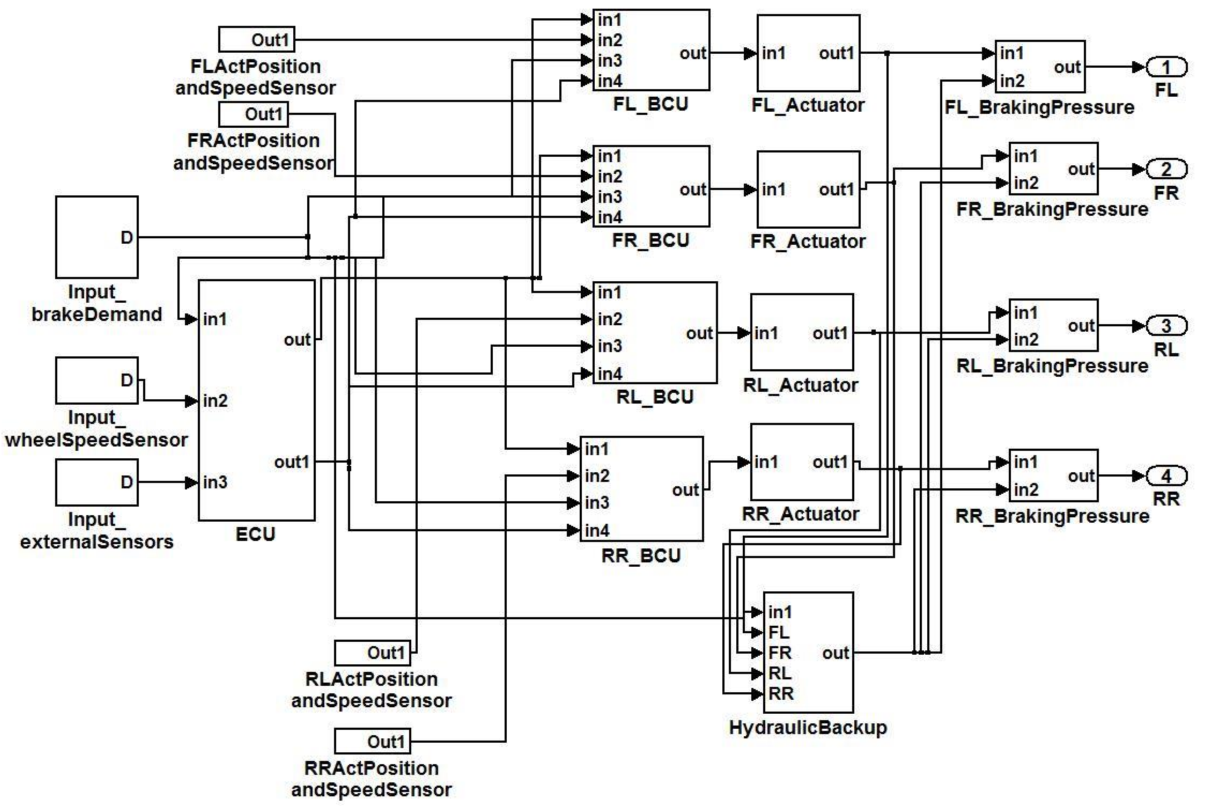


Figure 9. Architecture-allocated Functional Model for BBW System

Table 8: BBW Input and Output Flow Explanation

Component	Input	Input Explanation	Output	Output Explanation
Input Brake Demand	-	-	D	Provide input brake demand from driver to ECU, Hydraulic Backup and all BCUs
Input Wheel Speed Sensor	-	-	D	Provide information on wheel speed to ECU
Input External Sensor	-	-	D	Provide information on external variables to ECU
ECU	in1	Input brake demand from driver	out	Provide braking command to BCUs
	in2	Information on wheel speed	out1	Provide ABS command BCUs
	in3	Information on external variables		
BCU (for all four wheels)	in1	Braking command from ECU	out	Provide local wheel braking command to actuators
	in2	Actuator position and speed		
	in3	Braking command from brake pedal		
	in4	ABS command from ECU		
Actuator Position and Speed Sensor (for all four wheels)	-	-	out1	Provide information on local actuator position and speed to BCUs
Actuator (for all four wheels)	in1	Information on braking pressure to be applied	out1	Application of braking pressure
Hydraulic Backup	in1	Input brake demand from driver	out	Provide braking pressure from hydraulic component
	FL,FR, RL, RR	Monitoring of each actuator output		

In earlier work [34], the omission and commission failure of the basic braking function and ABS-assisted braking function are analyzed. In this paper, focus is placed on the BCU and the delivery of each wheel braking pressure as part of the entire brake-by-wire structure, as well as on exploring the relationship between the delivery (and failure of the delivery) of this function from different wheels.

The system delivers four braking functions, which are handled by a BCU delivering commands to the actuator of each wheel. The longitudinal symmetry of the design means that the potential single functional failures on each side of the car and their effects on the system are similar to those on the other side of the car. For this analysis, single functional failure and multiple combinatorial failures are investigated.

4.1 FFA and Severity Assessment

In line with the proposed technique, the process starts from failure severity assessment. The following section explores the analysis of functional failures.

4.1.1 Analysis of Single Functional Failure

The initial severity assessment of omission and commission failures are presented in [34], and here we introduced a new type of failure, LockedWheel, and investigate the effects of this failure on the system. LockedWheel failure occurs when a wheel experiences rapid deceleration (causing it to lock) and stop much more quickly than the vehicle could. This is usually prevented by the ABS anti-lock function which alternately reduces the pressure to the brake until it sees acceleration, and increases pressure until it sees deceleration again. This is performed within a very short period of time, resulting in the slowing down of the wheel matching the deceleration rate of the vehicle.

Table 9 presents the FFA for the braking of the front-left wheel where the effects of the wheel locking with or without braking intention are examined.

Table 9: FFA for Front-left Wheel

Component	Failure Type	Effects on System	Severity	Detection	Recommendation
FL_BrakingPressure	Locking-Com. Permanent wheel lock when there is no braking intention	Vehicle tends to drift to side. Severe loss of control as maximum brake is applied	Critical	Comparison of pedal input and pressure sensor feedback	Transform commission failure of brake pressure into omission of brake pressure
FL_BrakingPressure	Locking-Om. Permanent wheel lock when there is braking intention	Vehicle tends to drift to side. Severe loss of control as maximum brake is applied	Critical	Comparison of pedal input and pressure sensor feedback	ABS algorithm to prevent permanent locking

The table shows that the wheel locking failure is of critical severity. As it is assumed that it occurs when ABS feature is absent and recovery of the single wheel is not possible, in the following section we explore how simultaneous wheel locking can, in some cases, potentially reduce the instability of the vehicle.

4.1.2 Analysis of Multiple Functional Failure

The conjunction of two to four locking failures is examined next. As the system incorporates four braking functions (one for each wheel) and there are six corresponding failure modes for each function, there appears to be a large number of possible combinations. However, a systematic analysis of unique combinations yields a relatively small number, due to the symmetry of the brake-by-wire system.

Certain failure combinations are also inapplicable because they can only occur in mutually exclusive conditions. Here, analysis of the L-Braking failure is performed in a scenario where the locking occurs when brake pressure is required. A summary of FFA presented in [35] focuses on the analysis regarding wheel locking and concluded that:

- Severity of single wheel locking failure is critical and affects the stability and steerability of the vehicle.
- Severity of two locking failures in diagonal wheels is less severe because stability is improved.

- Severity of three locking failures is critical.
- Locking in all four wheels is identified as less severe than locking in three, or in some cases, two wheels.

This is based on the assumption that the vehicle is driven on a straight road. From these FFA results, hypothetical recovery mechanisms can potentially be implemented by incorporating the ability to perform intentional locking. The intentional locking of a diagonal wheel can be performed in response to a single wheel locking failure, and intentional locking of all four wheels can be used as a recovery mechanism to reduce the severity of a failure of three wheels.

4.1.3 Intentional-Locking Mechanism

Working with the assumption that the recovery plan to unlock the wheel is not possible, the ability to intentionally lock the wheel can be potentially incorporated as an additional function to each BCU. An additional module, Diagonal Locking (DL), is used to monitor the output of each BCU to detect locked wheel (L-Braking) and subsequently activate the locking of the corresponding diagonal wheel. This DL module can be implemented as part of the ECU or as a separate independent component. It is also possible to further analyze the failure to provide this intentional locking (e.g. omission and commission failure of DL). To maintain the simplicity of this example, however, we assume that DL only propagates failures, and focus the analysis on degradation phases that the system experiences in the occurrence of wheel locking, ensuring that required safety properties hold during these phases. Figure 10 illustrates the data flow from component DL to BCU.

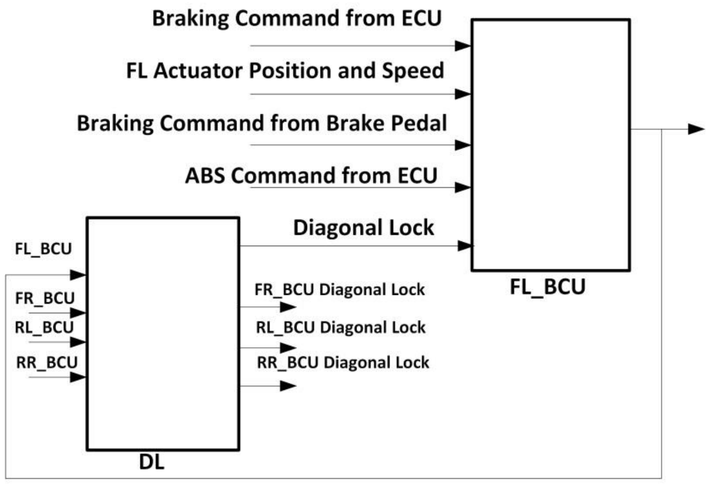


Figure 10. Diagonal Locking (DL) Controller

4.2 Local Failure Logic

Information of the internal malfunction and output deviation is provided for each component. BDBE, WSBE, and ESBE represent the internal malfunctions which cause omission failure for components Input_brakeDemand, Input_wheelSpeedSensor, and Input_externalSensor respectively. Internal malfunctions for ECU include ECUBE which causes omission failure, ECUBEc which causes commission failure, ECUBEabs which causes omission of ABS feature, and ECUBEabsC which causes commission of ABS feature.

The internal malfunction which causes omission failure for components ActPositionSpeedSensor is represented by LSBE. The internal malfunctions of BCU include BCUBE which causes omission failure, BCUBEc which causes commission failure, and LockBE which causes the locked wheel failure. LockBE represents an internal malfunction in the BCU algorithm causing maximum brake pressure to be applied.

The newly introduced component DL, and the actuator and BrakingPressure components are assumed to only propagate failures.

In this example, as it is assumed that the supervision of relevant parameters (e.g. wheel speed) and the processing of ABS commands are performed by the ECU, failure in ECU is sufficient to cause failure in producing the correct ABS command. The LockedWheel (L-Braking) failure occurs when LockBE occurs in the BCU and, at the same time, the ECU fails to produce the necessary command/information to instruct the ABS to prevent locking (O-ABScmd). Of course, sudden hard braking from the driver may also cause wheels to lock. It is possible to introduce a new event to model this, and include the event in the analysis. However, to maintain simplicity in this example, only LockBE is included. Subsequently, the failure logic for the LockedWheel failure in BCU can be expressed as:

L-Braking = LockBE AND O-ABScmd

4.3 FTA and FMEA

Following the introduction of intentional-locking mechanism through DL, it is important to understand the dynamics between failures of different wheels - as DL is activated in response to failure(s). The application of FTA and FMEA aims to derive root causes of the locking failure (L-Braking) of each wheel. This in turns enables us to study how failures from different wheels propagate to cause the intentional-locking of the diagonal wheel.

Based on the failure annotations, Table 10 shows a list of the minimal cut sets for L-Braking failure for front left wheel. The basic event is preceded with the component name. As part of the analysis result in the earlier work, ECU is assumed to be equipped with a redundant backup mechanism to prevent single-point failure. Therefore ECU consists of component PrimaryECU and StandbyECU.

Table 10: Fault Tree Analysis Results

Minimal cut sets for L-Braking	Explanation
FL_BCU.LockBE **AND** Input_wheelSpeedSensor.WSBE	Internal locking malfunction in BCU **AND** Internal malfunction in Wheel Speed Sensor
FL_BCU.LockBE **AND** Input_externalSensors.ESBE	Internal locking malfunction in BCU **AND** Internal malfunction in External Sensors
ECU.PrimaryECU.ECUBEabsC **AND** ECU.StandbyECU.ECUBEabsC **AND** FL_BCU.LockBE	Internal ABS-commission malfunction in Primary ECU **AND** Internal ABS-commission malfunction in Standby ECU **AND** Internal locking malfunction in BCU
ECU. PrimaryECU.ECUBEabsC **AND** ECU. StandbyECU.ECUBEabs **AND** FL_BCU.LockBE	Internal ABS-commission malfunction in Primary ECU **AND** Internal ABS-ommission malfunction in Standby ECU **AND** Internal locking malfunction in BCU
ECU. PrimaryECU.ECUBEabs **AND** ECU. StandbyECU.ECUBEabsC **AND** FL_BCU.LockBE	Internal ABS-ommission malfunction in Primary ECU **AND** Internal ABS-commission malfunction in Standby ECU **AND** Internal locking malfunction in BCU
ECU. PrimaryECU.ECUBEabs **AND** ECU. StandbyECU.ECUBEabs **AND** FL_BCU.LockBE	Internal ABS-ommission malfunction in Primary ECU **AND** Internal ABS-ommission malfunction in Standby ECU **AND** Internal locking malfunction in BCU

4.4 Construction of State Machine

In the early functional analysis [34], the identification of root causes of omission of braking function led to the introduction of backup mechanisms (e.g. hydraulic backup component, and redundancy in vehicle-level processing function). This information, along with information on the delivery of functions (e.g. basic braking pressure and assisted braking), are used to guide the construction of an abstract state machine. The state machine records the delivery of main function and the degraded states where non-critical function is loss or sacrificed. System abstract states can be categorized into: Normal, Permanent_Degraded1, Permanent_Degraded2, and Failed.

Here, we extend the state machine to reflect the inclusion of intentional-locking mechanism, and to further study dynamic behavior, particularly of DL. The information to construct these refined state machines is directly obtained from the HiP-HOPS model.

The updated abstract states include the following states:

- Normal state: where both basic braking and ABS braking functions are delivered. Braking function in normal mode is delivered through the primary source, i.e. the Electrical module.
- Permanent_Degraded3 state: where braking function is delivered by the Electrical module, ABS braking function can no longer be delivered, but intentional-locking is available.
- Permanent_Degraded1 state: where braking function is delivered by the Electrical module, but the ABS braking and intentional-locking cannot be delivered.
- Permanent_Degraded2 state: where braking pressure is delivered by Hydraulic module, ABS function is not delivered.
- Failed state: where no braking pressure is delivered.

The transition from Normal to degraded Permanent_Degraded3 state occurs when omission failure in ABS braking occurs, where intentional-locking can be activated on locked wheel failure. The transition from Permanent_Degraded3 to Permanent_Degraded1 occurs when DL is unavailable, for example, when an omission failure occurs in DL itself. The system moves to Permanent_Degraded2 when braking pressure cannot be delivered by electrical components, activating the hydraulic backup module. Failure in both the hydraulic and electrical mechanisms will cause the system to enter the Failed state. Figure 11 illustrates the abstract state machine.

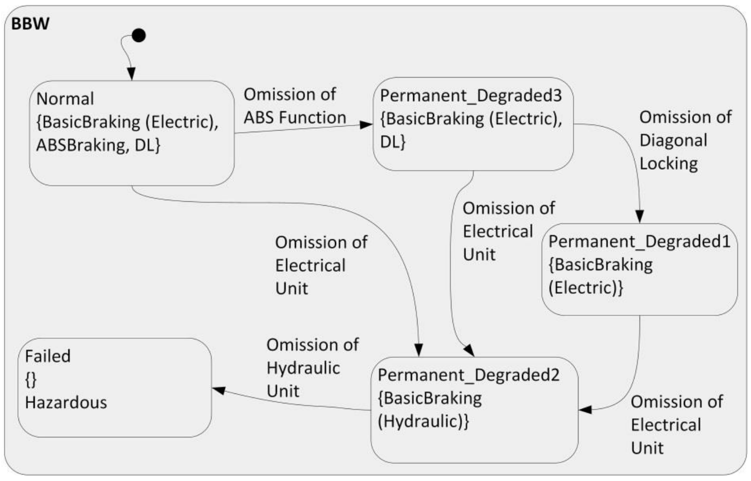

BBW
Normal {BasicBraking (Electric), ABSBraking, DL}
Omission of ABS Function
Permanent_Degraded3 {BasicBraking (Electric), DL}
Omission of Diagonal Locking
Omission of Electrical Unit
Permanent_Degraded1 {BasicBraking (Electric)}
Omission of Electrical Unit
Failed {} Hazardous
Omission of Hydraulic Unit
Permanent_Degraded2 {BasicBraking (Hydraulic)}
Omission of Electrical Unit

Figure 11. BBW Abstract State Machine

In addition to this, it is also possible to extend the analysis to include a component-level state machine which shows the dynamic behavior of the DL controller. This will allow us to understand how locking failures from different wheels - triggered by the root causes of L-Braking (identified from FTA/FMEA) - affect the changes in the system states and the activation of intentional diagonal locking. In its normal state, the DL module function is to monitor the occurrence of locking in any wheel and, when detected, to instantiate the locking of the diagonal wheel. It is possible for the DL module to have the following modes:

1. Normal: when there is no wheel locking

2. TD*n*_Critical_*X*: represents a Temporary Degraded state - when locking occurs in wheel(s) *X* with total *n* number of lockings occurring in the vehicle
3. PD*n*_*X*: represents a Permanent Degraded state - when locking occurs in wheel(s) *X* with total *n* number of lockings occurring in the vehicle

X represents vehicle wheel(s): FL, FR, RL, RR. $X \subseteq \{FL, FR, RL, R\}$. The states are mainly categorized based upon the *n* number of wheels locked (intentionally or not). Temporary Degraded (TD) modes are marked as critical because they only occur where either one or three of the wheels are locked, which has critical effects. These modes are temporary because the entry behavior (which is executed immediately once the mode is entered) triggers event "*X* DiagonalLock" which locks the corresponding diagonal wheel X, and therefore causes the system to move to a less-critical Permanent Degraded mode. Figure 12 describes this relationship and the transitions between the states.

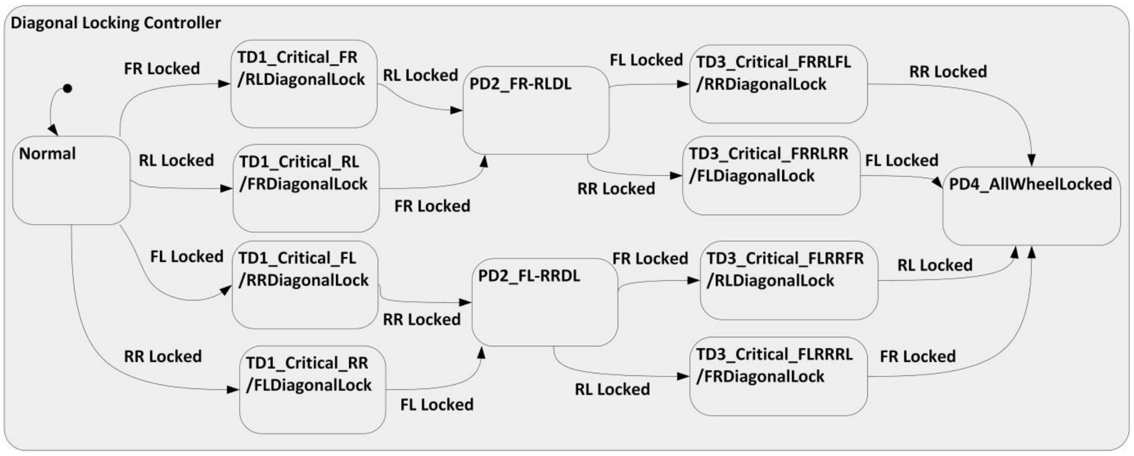

Diagonal Locking Controller
Normal
FR Locked
RL Locked
FL Locked
RR Locked
TD1_Critical_FR /RLDiagonalLock
TD1_Critical_RL /FRDiagonalLock
TD1_Critical_FL /RRDiagonalLock
TD1_Critical_RR /FLDiagonalLock
RL Locked
FR Locked
RR Locked
FL Locked
PD2_FR-RLDL
PD2_FL-RRDL
FL Locked
RR Locked
FR Locked
RL Locked
TD3_Critical_FRRLFL /RRDiagonalLock
TD3_Critical_FRRLRR /FLDiagonalLock
TD3_Critical_FLRRFR /RLDiagonalLock
TD3_Critical_FLRRRL /FRDiagonalLock
RR Locked
FL Locked
RL Locked
FR Locked
PD4_AllWheelLocked

Figure 12. State Machine for DL Controller

The events that trigger the state transitions are signals from individual BCUs to indicate wheel locking. This locking can be caused by an intentional locking command from DL or unintentionally as a result of L-Breaking.

State machines for other components were constructed. They reflect the hierarchical structure, and enable the generation of the NuSMV model which captures all the relevant modules that trigger corresponding transition events in BCU.

Figure 13 illustrates the state machine for component BCU of Front Left wheel which shows the connections between component state changes. The transition from Normal to Locked state in the wheel can be triggered by LockBE failure and omission of ABS command, or by intentional diagonal locking. The transition in ABS from Normal to Failed state may be caused by the omission of ECUabs failure, which is propagated from the ECU component. This then triggers the omission of ABS command. Transition within ECU from Normal state into the state where ABS is unavailable is shown to be caused by omission failure in its input, which is propagated from its input of external sensors and wheel speed sensor, and ABS related internal malfunctions within the ECU. External and Wheel Speed Sensor components change from their Normal state into a Failed state when their corresponding internal malfunctions ESBE and WSBE occur.

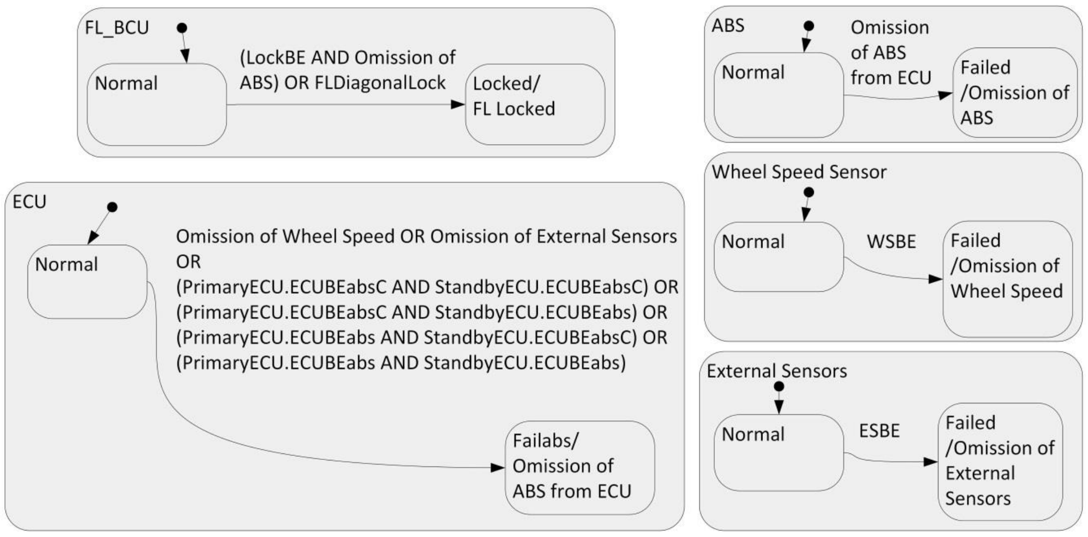

FL_BCU
Normal
(LockBE AND Omission of ABS) OR FLDiagonalLock
Locked/ FL Locked
ABS
Normal
Omission of ABS from ECU
Failed /Omission of ABS
ECU
Normal
Omission of Wheel Speed OR Omission of External Sensors OR (PrimaryECU.ECUBEabsC AND StandbyECU.ECUBEabsC) OR (PrimaryECU.ECUBEabsC AND StandbyECU.ECUBEabs) OR (PrimaryECU.ECUBEabs AND StandbyECU.ECUBEabsC) OR (PrimaryECU.ECUBEabs AND StandbyECU.ECUBEabs)
Failabs/ Omission of ABS from ECU
Wheel Speed Sensor
Normal
WSBE
Failed /Omission of Wheel Speed
External Sensors
Normal
ESBE
Failed /Omission of External Sensors

Figure 13. Hierarchical Failure Modelling in BCU FL

The construction of these state machines, particularly the refinement of the transitions events is systematically guided by HiP-HOPS models. To effectively link failure behavior to input modules and capture structural information, transition events are expressed in a structure similar to HiP-HOPS failure annotations.

4.5 Verification of safety properties

Once the mode charts and NuSMV models have been constructed, requirements can be verified via model checking. Among the requirements, safety properties are of primary concern in safety-critical systems. The process in this example aims to ensure that the extended model and the introduction of intentional locking mechanism uphold the system safety requirements.

Possible safety requirements which can be verified in this example are:

SR1: Intentional locking of diagonal wheel shall not always lead to hazardous state

This safety property aims to ensure that the activation of diagonal locking will not, in itself, cause unintentional locking which leads to hazardous states.

To do this, the "hazardous" state is defined as the condition either where one wheel locks or where three wheels lock, i.e. the occurrence of either TD1_Critical_X or TD3_Critical_X respectively:

```
Hazardous := case
States = TD1_Critical_FR |States = TD1_Critical_RL | States = TD1_Critical_FL | States = TD1_Critical_RR |States = TD3_Critical_FRRLRR |States = TD3_Critical_FLRRFR | States = TD3_Critical_FLRRRL : 1;
1: 0;
esac;
```

The activation of DL (DLActive) is defined as the condition when any of the wheels has been diagonally locked intentionally:

```
DLActive := FLdiagonalLock | RLdiagonalLock | FRdiagonalLock | RRdiagonalLock;
```

We aim to verify that the diagonal locking function itself will not always lead to hazardous state. This can be expressed in CTL as the following, and this has been verified to be true by the model checker:

```
SPEC !AG(DLActive -> Hazardous);
```

Another example of property which can be investigated by model checker is

SR2: In situations where the number of locked wheel failures is not one or three, the system shall not enter the hazardous mode

```
SPEC AG((!(Counter = 1) & !(Counter = 3))->!Hazardous));
```

This property does not hold and the model checking traces demonstrate that the locking failure of two non-diagonal wheels at one point leads to the locking of three wheels, which is hazardous. With the current arrangement of DL, however, this means that the locking of two non-diagonal wheels should always eventually lead to the locking of all four wheels. In the NuSMV model, variable TwoParallelWheelsLocked is assigned to represent the locking of two non-diagonal wheels. This state of reachability can be verified through the following modified properties:

SR3: In situations where two non-diagonal wheels are locked, all four wheels shall eventually be locked

```
AG(TwoParallelWheelsLocked -> AF(States = PD4_AllWheelsLocked));
```

This property is verified to be true by the model checker, and therefore we are assured that the non-diagonal locking of two wheels will also lead to locking of all four wheels (which is less severe).

The next property aims to ensure that the intentional locking should only be instantiated when the ABS function is not working, as in its presence the ABS would be expected to manage the prevention of wheel locking:

SR4: Intentional locking of diagonal wheel shall not be instantiated when ABS function is working

For this, we define a condition ALLOFF where no diagonal locking is taking place. Every time when the ABS is present (omission of ABS does not occur), ALLOFF should be true.

```
SPEC AG ((O-ABScmd = 0) -> ALLOFF);
```

This is verified to be true by the model checker.

Another example of property that one could check is the system failure recoverability, for instance, whether intentional diagonal locking will always eventually result in the system moving from the hazardous state to a less-severe (non-hazardous) state. This aims to ensure that DL fulfills its function as a fail-safe mechanism. Less-severe states refer to the condition where either only two diagonal wheels are locked or all four wheels are locked, which brings us to the next requirement – which has been verified to be true:

SR5: Intentional locking of wheels shall always lead the system to a less-severe state

```
SPEC AG (DLActive -> !Hazardous);
```

The verification of these properties helps highlight the ability to assess dynamic behavior in the DL controller and to verify safety properties which cannot be managed with FLSA.

It is important to note that the example and modification are hypothetical and simplified. The example aims to illustrate the application of the proposed integration, and does not cover the detailed design. The discussion on safe and optimum intentional-locking timing and rate is out of the scope. Other advanced feature which assist braking, for example, Electronic Brake Distribution (EBD) as part of the ABS, are not covered. Instead, the example shows how the proposed integrated technique can *in principle* assist the design, development, and safety assessment process to achieve a more robust and fault-tolerant design.

5 Conclusion

The ability to perform early safety analysis which can guide the design of safety critical systems has become increasingly important. In this paper, we presented a method which facilitates the combined application of two prominent MBSA techniques, namely FLSA in the form of HiP-HOPS and model checking, from an early stage. The method exploits the strengths of both techniques, and systematically utilizes the results of FLSA analysis for construction of design models for the system that are subject to formal verification. This allows identification of design weakness and verification of safety requirements, and contributes to a more-robust design process. An example of a brake-by-wire system was presented to demonstrate the application of the proposed method. We illustrated how the application of HiP-HOPS identified root causes for locked wheel failures, and how HiP-HOPS models and results also systematically assist the construction of the state machines for a control component, which can then be verified using NuSMV.

This paper has demonstrated that the output of FLSA can guide the construction of input for model checking. In early design, abstract state machines created via FLSA can be used to assist systematic design of degraded modes and fault tolerant strategies. In more detailed design, state machines can be used to model and analyze detailed dynamic behavior of system elements, and allow detailed safety requirements to be verified (which is a challenge in FLSA). This essentially brings the capabilities of model checking into an earlier design stage, allowing the results to have a greater influence in design decisions. The degree of automation enabled by the underpinning techniques allows the analysis to be iterated, and contributes to a more-rigorous safety assessment. Future work includes application of the proposed concept in the context of design using emerging architecture description languages such as AADL [6] and EAST-ADL [7].

6 Acknowledgments

This work has been supported by the British government through the HEIF5 project.

7 References

[1] IEC. Hazard and Operability Studies (HAZOP studies). International Electrotechnical Commission (IEC); 2000.

[2] Vesely WE, Goldberg FF, Roberts NH, Haasl DF. Fault Tree Handbook Washington D.C., USA: Nuclear Regulatory Commission; 1981.

[3] IEC60812. Functional Safety of Electrocal/Electronic/Programmable Electronic Safety/ Related Systems, Analysis Techniques for System Reliability - Procedure for Failure Mode and Effect Analysis (FMEA). 1991.

[4] Heimdahl MPE. Formal model-based development in aerospace systems: challenges to adoption. Lecture notes Minnesota: University of Minnesota Software Engineering Centre Critical System Research Group; 2007.

[5] Weilkiens T. Systems Engineering with SysML/UML: Modeling, Analysis, Design. CA, USA: Morgan Kaufmann; 2008.

[6] Feiler PH, Gluch DP, Hudak JJ. The Architecture Analysis and Design Language (AADL): An Introduction. 2006.

[7] Cuenot P, Chen DJ, Gerard S, LöNN H, Reiser MO, Servat D, et al. Towards improving dependability of automotive systems by using the EAST-ADL architecture description language. Lecture Notes in Computer Science. 2006;4615:39-65.

[8] Walker M, Mahmud N, Papadopoulos Y, Tagliabò F, Torchiaro S, Schierano W, et al. ATESST: Review of relevant Safety Analysis Techniques. 2010.

[9] Clarke E.M, Wing JM. Formal Methods: state of the art and future directions ACM Computer Surveys 1996;28:626-43.

[10] Papadopoulos Y, Maruhn M. Model-based automated synthesis of fault trees from Matlab-Simulink models. International Conference on Dependable Systems and Networks2001. p. 77-82.

[11] Cimatti A, Clarke E, Giuchiglia F, Roveri M. NuSMV: a new symbolic model verifier. Computer-aided Verification. London: Springer-Verlag; 1999. p. 495-9.

[12] Walker M, Papadopoulos Y. PANDORA: The time of Priority-AND gates. INCOM. France2006. p. 237-42.

[13] Parker D, Papadopoulos Y. Optimization of networked controlled systems using model-based safety analysis techniques. IEEE International Conference on Networking, Sensing and Control. London2007. p. 425-30.

[14] Fenelon P, McDermid JA. An integrated toolset for software safety analysis. Journal of Systems and Software. 1993;21(3):279-90.

[15] Kaiser B, Liggesmeyer P, Mackel O. A new component concept for fault tree. 8th Australian Workshop on Safety Crucial Systems and Software2003.

[16] Grunske L, Kaiser B, Papadopoulos Y. Model-driven safety evaluation with state-event-based component failure annotation. Lecture Notes in Computer Science 2005;3489:33-48.

[17] Clarke EM, Emerson A. Characterizing correctness properties of parallel programs using fixpoints. Automata, Languages and Programming1980. p. 169-81.

[18] Arnold A, Gerald P, Griffault A, Rauzy.A. The Altarica formalism for describing concurrent systems. Fundamenta Informaticae. 2000;34:109-24.

[19] Bozzano M, Villafiorita A. The FSAP/NuSMV-SA safety analysis platform. International Journal on Software Tools for Technology Transfer (STTT). 2006;9:5-24.

[20] McMillan KL. Symbolic model checking Kluwer; 1993.

[21] Miller SP, Tribble A, Whalen M, Heimdahl M. Proving the shalls: early validation of requirements through formal models. International Journal on Software Tools for Technology Transfer (STTT). 2003;8:303-19.

[22] IEC61508. Functional safety of electrical/electronic/programmable electronic safety-related systems. Geneva: International Electrotechnical Commission; 1998.

[23] Papadopoulos Y, Grante C. Evolving car designs using model-based automated safety analysis and optimisation techniques. Journal of Systems and Software. 2005;76:77-89.

[24] Papadopoulos Y, Walker M, Parker D, Rüde E, Hamann R, Uhlig A, et al. Engineering failure analysis and design optimisation with HiP-HOPS. Journal of Engineering Failure Analysis. 2011;18:590-608.

[25] Papadopoulos Y. Safety-directed system monitoring using safety cases: University of York; 2000.

[26] Suri N, Walter C, Hugue MM. Advances in ultra-dependable distributed systems: IEEE Computer Society Press 1995.

[27] Loer K. Model-based automated analysis for dependable interactive systems: University of York; 2003.

[28] Juarez-Domiguez AL, Nancy AD, Joyce JJ. Modelling feature interactions in the automotive domain. 2008 International Workshop on Models in Software Engineering (MiSE) 2008. p. 45-50.

[29] Berard B, Bidoit M, Finkel A, Laroussinie F, Petit A, Petrucci. L, et al. System and software verification: model-checking techniques and tools. Germany: Springer-Verlarg; 1999.

[30] Tribble AC, Miller SP. Software safety analysis of a flight management – a status report. 22nd Digital Avionics Systems Conference2003.

[31] Papadopoulos Y. Safety analysis of a distributed brake by wire system for cars. ESPRIT 23396 (TTA) Deliverable. University of York 1998.

[32] Colombo D. Brake-by-wire system development: Technology and development process. Torino: FIAT Group; 2007.

[33] Hedenetz B, Belschner R. Brake-by-wire without mechanical backup using a TTP-communication network. Society of automotive engineering International; 1998.

[34] Sharvia S, Papadopoulos Y. Integrated application of compositional and behavioural safety analysis. Dependability and Complex System 2011. Wroclaw: Springer; 2011.

[35] Papadopoulos Y, McDermid JA. Hierarchically Performed Hazard Origin and Propagation Studies. 18th International Conference on Computer Safety, Reliability and Security. Toulouse: Springer-Verlag; 1999. p. 139-52.