Reliability Engineering and System Safety 205 (2021) 107237

Contents lists available at ScienceDirect

Reliability Engineering and System Safety

journal homepage: www.elsevier.com/locate/ress

Model checking reveals design issues leading to spurious actuation of nuclear instrumentation and control systems☆

Antti Pakonen*,[a], I Buzhinsky[b,c], K Björkman[a]

[a] VTT Technical Research Centre of Finland Ltd., P.O. Box 1000, FI-02044 VTT, Finland
[b] Department of Electrical Engineering and Automation, Aalto University, P.O. Box 11000, FI-00076 Aalto, Finland
[c] Computer Technologies Laboratory, ITMO University, 197101 St. Petersburg, Russia

ARTICLE INFO

Keywords:
Model checking
I&c
Spurious failure
Model-based system engineering

ABSTRACT

A spurious actuation of an industrial instrumentation and control (I&C) system is a failure mode where the system or its component inadvertently produces an operation without a justified reason to do so. Design issues leading to spurious failures are difficult to analyse, but pose a high risk for safety. Model checking is a formal verification method that can be used for exhaustive analysis of I&C systems. In this paper, we explain how formal properties that address spurious failures can be specified, and how model checking can then be used to verify I&C application logic designs based on vendor-specific function block diagrams. Based on over ten years of successful practical projects in the Finnish nuclear industry, we present 21 real-world design issues (representing 37% of all detected issues), each involving a systemic failure that could lead to spurious actuation of nuclear safety I&C. We then describe how random failures of the underlying hardware architecture—another cause for spurious actuation—can also be included in the models. With an experimental evaluation based on real-world nuclear industry models, we demonstrate that our method can be effectively used for the verification of single failure tolerance.

1. Introduction

Spurious actuation is defined as a failure mode where an actuation of an instrumentation and control (I&C) system function occurs without a real demand [1]. The terms “inadvertent operation” or “active failure” [2] are also used. (In contrast, “passive failure” [2] means that the system fails to produce the required response.) In a nuclear power plant, a spurious failure can limit the ability of safety systems to function properly, and challenge the safety of the plant [3]. Nuclear regulatory bodies agree that spurious actuation is a particular safety concern [4], and that safety of systems “cannot be discussed and shown to exist” without considering unintended behaviour of both hardware and software [5].

Active failures can be further divided into random and systematic failures [2]. Random failures [6] can occur at any time, and the probability of their occurrence can increase due to aging of hardware components. Systematic failures [6], on the other hand, are deterministically related to a cause that can be eliminated by a modification of the design, e.g., a software design error.

By their nature, spurious failures are more complex to analyse than passive failures [1]. It is more straightforward to think of test cases for the intended functionality. Spurious failures are also a multidisciplinary issue, as the failure can be caused by any component between the process measurement sensors and the actuators [1], but also by support systems (power supply, cooling), environmental effects, human actions, or plant transients [3].

One of the safety design principles in nuclear power plants (NPPs) is defence-in-depth—establishment of several successive physical barriers for containing accidents. Still, spurious actuation of a safety I&C system is a hazard that can potentially challenge more than one barrier simultaneously [7]. As more complex I&C architectures are more difficult to design and verify, adding numerous defence-in-depth levels can actually increase the risk of spurious actuation [2].

Another NPP design principle that can be used to deal with spurious actuation from random failure is redundancy–adding redundant subsystems, and voting on control actions. Together, the subsystems are

☆ This work was funded by the Finnish Research Programme on Nuclear Power Plant Safety 2018–2022 (SAFIR 2022). We wish to thank Mika Johansson, Kim Wahlström, Matias Halinen and Nina Lahtinen of STUK for valuable help. We also wish to thank VTT’s clients in the Finnish nuclear industry for permitting the use of highly confidential customer project data for our research.

* Corresponding author.
E-mail addresses: antti.pakonen@vtt.fi (A. Pakonen), igor.buzhinskii@aalto.fi (I. Buzhinsky), kim.bjorkman@vtt.fi (K. Björkman).

https://doi.org/10.1016/j.ress.2020.107237
Received 7 October 2019; Received in revised form 27 August 2020; Accepted 7 September 2020
Available online 08 September 2020

then capable of performing the desired tasks even if any single component in them fails. The same solution is also used for safety critical applications in, e.g., aviation [8,9], aerospace [10–12], railway [13], or automotive [14] industries. Further failure tolerance is achieved with diversity—by using different technologies or design principles in redundant back-up systems.

Nevertheless, aside from emphasizing good design principles like defence-in-depth, single failure tolerance, quality, independence and qualification [7], the challenge remains: How to ensure that I&C systems do not contain design issues that might lead to spurious actuation? Modern I&C systems are so complex in terms of both hardware and software (platform and application) that 100% test coverage is practically impossible. The risk of systemic failure remains.

Model checking [15] is a formal verification method where a stated formal property of a system is verified through exhaustive exploration of all the reachable states of a model of that system. It is a proven method for exhaustive verification of a nuclear I&C system's application logic, regardless of whether the logic is implemented with programmable logic controller (PLC) software or field-programmable gate array (FPGA) configuration [16]. Since the formal properties can also address unwanted behaviour, the method can also be used to identify systemic failures leading to spurious actuation. However, the focus is often on the logic specification (e.g., a function block diagram), alone. In reality, the application logic operates on hardware components subject to random failure. Verifying that the system and the intended functionality are also fault tolerant calls for modelling of both the application logic and the failure modes of the underlying hardware architecture, in unison.

This paper is an extended version of [17], and includes a practical evaluation of ideas presented in [18]. The contribution is fourfold. First, we discuss the types of formal properties needed for analysing spurious actuation, the relationships between them, and the challenges in their formalisation. Second, we introduce a modelling approach and a practical tool for verifying I&C application logics, and discuss their inherent limitations. Third, we present data and discuss the characteristics of the 21 design issues VTT[1] has revealed in practical nuclear industry projects, each example involving a scenario where the application logic design causes a spurious actuation. Fourth, we show how the application logic model can be supplemented with hardware failures, and demonstrate using real-world nuclear industry models that the modelling method can effectively be used to verify single failure tolerance.

The rest of the paper is structured as follows. In Section 2, we introduce the basics of model checking, and discuss the use of temporal logic languages for specifying formal properties that deal with spurious actuation. In Section 3, we describe an example of a fault-tolerant, four-redundant nuclear I&C safety system, and describe how a graphical frontend for the NuSMV [19] model checker—called MODCHK—processes nuclear industry specific aspects of application logics. We then list design issues revealed in practical customer projects, and include a real example. In Section 4, we describe a method for modelling failures of the underlying I&C hardware components, and then evaluate the method–again using real nuclear industry application logic models. We analyse related research in Section 5, discuss our results in Section 6, and present our conclusions in Section 7.

2. Formal verification

2.1. Model checking

Model checking [15] is a formal verification method where a software tool called a model checker is used to specify a formal model and analyse whether a desired property holds for it through exhaustive exploration of all its reachable states. The desired properties are formalised using temporal logic languages like Linear Temporal Logic (LTL), Computation Tree Logic (CTL) [15] or Property Specification Language (PSL) [20]. If an execution path that violates the property is found, it is returned to the user as a counterexample scenario, possibly revealing a design issue.

Formally, the model of the system is a Kripke structure, a tuple $(S, S_0, T, \mathrm{AP}, L)$, where S is a set of states, $S_0 \subseteq S$ is a set of initial states, $T \subseteq S \times S$ is the transition relation, AP is a set of atomic propositions, and $L: S \rightarrow 2^{\mathrm{AP}}$ is a labeling function. In addition, we assume that the model has a number of Boolean or integer state variables such that each state corresponds to a unique assignment of these variables. In this case atomic propositions can be thought of as either Boolean variables or bits of integer variables, and the labeling function essentially returns the assignment of variables to a state. An execution path (or, simply, a path) of a model is a finite or infinite sequence $(s_0, s_1, \ldots)$ such that $s_0 \in S_0$ and $(s_i, s_{i+1}) \in T$ for all $i \geq 0$. A state $s \in S$ is reachable if it belongs to some path. Below, speaking of a state space, we will mean the set of reachable states.

A fundamental challenge in model checking is to avoid state space explosion, where the number of model states to enumerate through becomes enormous [15]. Symbolic model checkers—like the popular open source tool NuSMV [19]—employ Binary Decision Diagrams (BDD), which provide a canonical representation for Boolean formulae. BDD processing allows avoiding explicit state enumeration [21]. Another solution to make the analysis faster is to use Boolean (or propositional) satisfiability (SAT) solvers to perform bounded model checking (BMC), where the allowed length of checked state transition sequences is limited [22]. SAT solvers are also used to check whether a Boolean formula holds in all reachable states with inductive methods [23].

NuSMV is based on synchronous processing of model components over discrete time, where time corresponds to the number of executed transitions, but continuous model checkers are also available (e.g., UPPAAL [24]), as well as tools like HyComp [25] for hybrid systems.

2.2. Formal property specification

Most (but not all) formal properties can be cast into one of two types: safety properties dictate that something shall not happen, and liveness properties dictate that something shall eventually happen [26]. More formally, a finite execution cannot violate a liveness property–a counterexample must be lasso-shaped [15] (with a loop at the end), instead. Conversely, an execution path that violates a safety property can always be truncated to a finite one, and there is an identifiable time step where the undesired state occurs. Therefore, the properties that are specifically written to prove the absence of spurious actuation are safety properties.

Temporal logic languages provide a formalism to formulate statements over execution paths, not just individual states [15] (typically, only infinite execution paths are considered). For this purpose, LTL and CTL utilize so-called temporal operators in addition to Boolean connectives. The following temporal operators are defined in LTL (using notation from [15]):

- **X** p: p is true in the next state of the path ("neXt").
- **G** p: p is true at every state on the path ("Globally").
- **F** p: p is true at some future state on the path ("Finally").
- p **U** q: q is true at some future state, and at every preceding state on the path, p is true ("Until").

In the above expressions, p and q can be Boolean statements over state variables as well as nested temporal formulae.

In addition to describing future behaviour, being able to refer to past states is often convenient. Past LTL operators, as suggested in, e.g., [27], include:

[1] VTT Technical Research Centre of Finland Ltd. is a state owned company providing research and innovation services. https://www.vttresearch.com/.

- **Y** p: p holds in the previous state on the path. **Y** p is false in the initial state ("Yesterday").
- **Z** p is equivalent to **Y** p, except that it is true in the initial state.
- **H** p: p is true at every preceding (and the current) state on the path ("Historically").
- **O** p: p is true at some past (or the current) state on the path ("Once").
- p **S** q: q is true at some past state, and for every state that has then followed on the path, p has been true ("Since").

CTL is based on branching time, adding path quantifiers **A** ("for all execution paths") and **E** ("for some execution path").

Property Specification Language (PSL) [28] is an extension of LTL and CTL designed to be compatible with hardware description languages. In addition to aiming at human readability [20], PSL offers an "LTL style" called Sequential Regular Expressions (SERE), which is convenient for describing multi-cycle behaviour. As an example [20], the property "if `req` is true, then, on the next cycle, `ack` is true for one cycle, then `busy` is true for three cycles, and then `done` is true for one cycle" is written as:

```
always {req} |=>{ack ;busy[*3] ;done}!;
```

In the example, "`always`" stands for **G**, and the repetition operator `[*n]` replaces nested **X** expressions. The SERE style of PSL can be useful in specifying I&C system properties related to timing and control sequencing [29].

2.3. Specifying properties for spurious failures

To detect a spurious actuation scenario, the analyst does not necessarily have to specify any property for that specific purpose. Any specified property can reveal a counterexample where a spurious actuation occurs. For example, if the property addresses a requirement for opening a valve on high pressure, the counterexample can show the system model sending a close command on high pressure, instead.

Nevertheless, it is important for the analyst to consider unwanted system behaviour separately. Requirement specification documents do not typically include self-evident statements such as "the system shall not end up in a deadlock / send contradictory commands / actuate spuriously". Such requirements might be omitted due to, e.g., the difficulty in their verification. For any property that captures a desired ("good") behaviour, there may be several complementary properties that capture unintended ("bad") behaviour. Thankfully, spurious actuation properties share a type of symmetry with the intended-behaviour properties.

Below, we consider four types of desired behaviours for I&C systems, and the associated properties for addressing spurious actuation (see Fig. 1). First, immediate response means that the system shall give the actuation order (response) if and only if the actuation criteria (request) is true. Second, delayed response means that the response shall occur a time after the request. In the bounded variant, the response follows the request after a set amount of time steps. In the unbounded variant, the response occurs after an unspecified time. Finally, triggered response means that the response shall occur immediately upon request, but may last longer.

For immediate response, the intended property "a request shall lead to a response" can be written in LTL as the property:

$$\mathbf{G}(\text{request} \rightarrow \text{response}). \tag{1}$$

The counterexample would then contain a state where the request holds but the response is not true. However, (1) holds in a scenario where the response is true but the request is not. In order to address spurious actuation, let us turn (1) around to state: "a response implies that there is a request", or:

$$\mathbf{G}(\text{response} \rightarrow \text{request}). \tag{2}$$

For delayed response–bounded, the intended property is expressed

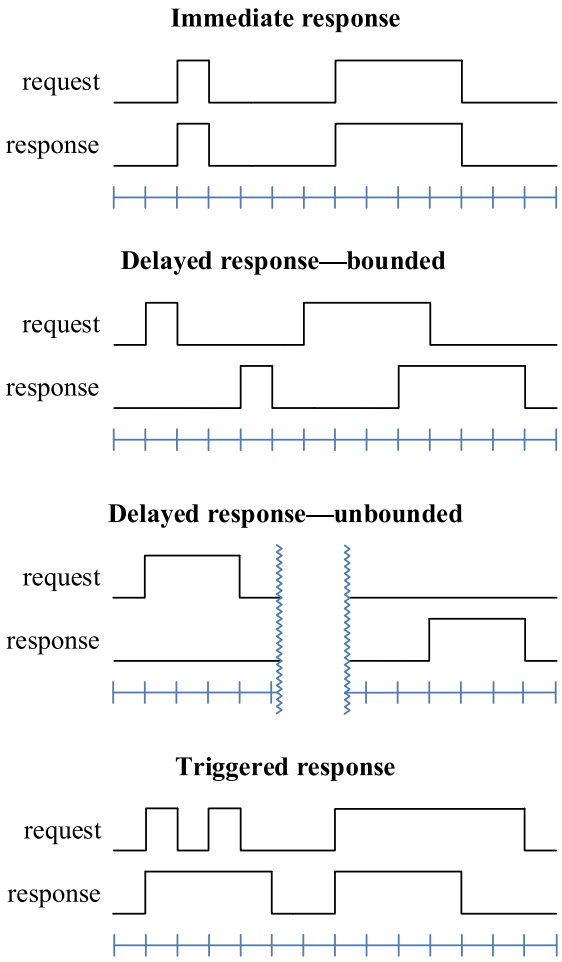

Fig. 1. Types of intended response to an actuation request.

using nested **X** operators, e.g., for a delay of three time steps:

$$\mathbf{G}(\text{request} \rightarrow \mathbf{X}(\mathbf{X}(\mathbf{X}\,\text{response}))). \tag{3}$$

The complementary spurious property, using nested **Y** operators, reads:

$$\mathbf{G}(\text{response} \rightarrow \mathbf{Y}(\mathbf{Y}(\mathbf{Y}\,\text{request}))). \tag{4}$$

For delayed response—unbounded, we formulate the more general liveness property "a request shall eventually lead to a response", or:

$$\mathbf{G}(\text{request} \rightarrow \mathbf{F}\,\text{response}). \tag{5}$$

In order to address spurious actuation, the counterpart safety property is:

$$\mathbf{G}(\text{response} \rightarrow \mathbf{O}\,\text{request}). \tag{6}$$

The symmetry of the above formulas exemplifies why, as stated in [27], **Y** and **O** are the "temporal duals" of **X** and **F**, respectively. If past temporal operators cannot be used, (6) can be rewritten using the less intuitive but equivalent expressions: $\neg(\neg\,\text{request}\ \mathbf{U}\,(\text{response} \wedge \neg\,\text{request}))$ [30], or: $(\mathbf{G}\,\neg\,\text{response}) \vee (\neg\,\text{response}\ \mathbf{U}\,\text{request})$ [31].

For triggered response, (1) and (2) do not apply, and an intended property "response shall be true when request changes from false to true" is written as:

$$\mathbf{G}((\neg\,\text{request} \wedge \mathbf{X}\,\text{request}) \rightarrow \mathbf{X}\,\text{response}). \tag{7}$$

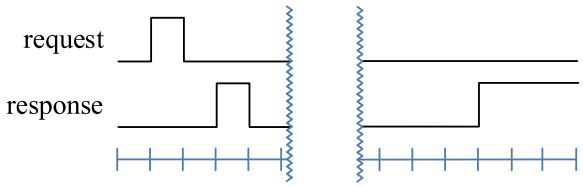

Fig. 2. Even if the latter occurrence of response is considered a spurious actuation, the property **G**(response → **O** request) will not reveal the issue.

A corresponding spurious property would then read:

$$\mathbf{G}((\neg \text{response} \wedge \mathbf{X}\,\text{response}) \rightarrow (\neg \text{request} \wedge \mathbf{X}\,\text{request})). \quad (8)$$

If request being true at the initial state should count as a triggering event, (7) can be modified to:

$$\mathbf{G}((\mathbf{Z}\ \neg \text{request} \wedge \text{request}) \rightarrow \text{response}). \quad (9)$$

and the spurious property (8) to:

$$\mathbf{G}((\mathbf{Z}\ \neg \text{response} \wedge \text{response}) \rightarrow (\mathbf{Z}\ \neg \text{request} \wedge \text{request})). \quad (10)$$

The desired behaviour can also be a combination of delayed response and triggered response, in which case (4), (8) or (10) will not apply as such, but (6) can still be used. (6) is also true if (2) is true.

While (6) can be thought of as a "universal" spurious property (see also the properties listed in Appendix A), it should be noted that a single past occurrence of request satisfies (6) even if there are several future occurrences of response. If the latter occurrence of response in Fig. 2 is considered a spurious actuation, the property will nevertheless hold.

In order to detect the spurious actuation scenario in Fig. 2, we can specify that a response shall be preceded by a request, and there has not been a (finished) response since that request:

$$\mathbf{G}(\text{response} \rightarrow (\neg((\mathbf{Y}\,\text{response}) \wedge \neg\text{response})\,\mathbf{S}\,\text{request})). \quad (11)$$

2.4. Open-loop vs. closed-loop modelling

I&C systems can be verified using open-loop or closed-loop models. Open-loop models only include the I&C logic, and do not account for feedback from the controlled process. In closed-loop modelling, feedback from the plant helps in filtering out irrelevant model behaviours, and can therefore reduce the state space [32].

Analysing architecture-level requirements (including non-functional requirements such as failure tolerance) is only possible if the plant is modelled as a whole. Model-based Systems Engineering (MBSE) is an approach that can be described as "the formalized application of modelling principles, methods, languages, and tools to the entire lifecycle of large, complex, interdisciplinary, sociotechnical systems" [33]. The key artefact in MBSE is a unified, coherent system model.

However, generating an accurate plant model for closing the loop can be challenging. Limiting the model behaviour can accidentally eliminate model executions relevant for safety, and the analysis times can actually increase [34]. We therefore continue to work with open-loop models of the I&C application logic, but, in Section 4, introduce hardware failures in order to verify single failure tolerance.

3. Verification of nuclear i&c systems

3.1. Nuclear power plant i&c systems and failure tolerance

For obvious reasons, the I&C systems of a nuclear power plant need to be failure tolerant. Single failure criterion means that the system shall be able to perform its function even if any single component designed for the function fails. Protection against single failure is achieved using several (perhaps identical) redundant subsystems placed in physically separated divisions.

Consequential failure refers to "a failure caused by a failure of another system, component or structure or by an internal or external event at the facility" [35]. For example, a failure of a power supply or ventilation system can result in the subsequent total failure of several I&C devices, but is still considered a single failure that shall be tolerated.

Common cause failure (CCF) refers to a "failure of two or more structures, systems and components due to the same single event or cause" [35]. Protection against CCF can be achieved using diverse backup systems (e.g., a different supplier, technology, or operating principle).

Single failure criterion is in the Finnish nuclear safety requirements referred to as N + 1. An even stricter criterion called N + 2 is used for the most critical systems (e.g., the reactor trip system). N + 2 means that in addition to the single failure, the system still needs to perform its function even if "any other component or part of a redundant system—or a component of an auxiliary system necessary for its operation—is simultaneously out of operation due to repair or maintenance" [35]. N + 2 can be fulfilled with a three-redundant structure where one operating division is sufficient for performing the function (3 × 100%), or a four-redundant structure where two operating divisions are needed (4 × 50%). In Finland, digital I&C systems of the highest safety class[2] are in practice always four-redundant.

As an example of a four-redundant I&C system, let us consider the Protection System (PS) of the proposed U.S. version of the European Pressurized Water Reactor (EPR) nuclear plant [36]. PS is based on Areva NP's TELEPERM XS technology, and each of the four independent divisions is located in a separate building [37] (see Fig. 3).

The PS utilizes different types of functional units [36]. The Acquisition and Processing Units (APUs) acquire signals from the process sensors and monitoring systems via the Signal Conditioning and Distribution System (SCDS) using a hardwired connection, perform calculations and setpoint comparisons, and distribute the results to the Actuation Logic Units (ALUs) for voting. The ALUs perform voting over processing results and issue actuating results, taking into account operator commands from the Safety Information and Control System (SICS). The actuation orders are then sent from the ALUs to the Priority and Actuator Control System (PACS) via a hardwired connection.

The Monitoring and Services Interfaces (MSI) units provide status monitoring, and information for display to operators via the Process Information and Control System (PICS). The Service Unit (SU) is used for system diagnosis and periodic testing.

Single failures that occur before the voting logic are handled by the voting logic in the ALU. Single failures at the voting logic level are handled by either redundancy within each division or redundancy across the four divisions. In the application logic (processed by the APUs and the ALUs), each signal has a status, which is set to "fault" upon failures detected by input modules of function processors. The status is then used to exclude invalid signals in selection (e.g., second-maximum, second-minimum[3]) and voting (n-out-of-m) blocks [36].

The notion of status or validity is not specific to Areva NP's TELEPERM XS technology, but is also used in a similar manner in Rolls-Royce's digital I&C platform Spinline [38].

[2] Finnish Safety Class 2 for I&C systems and equipment.

[3] These blocks select the second-largest and second-smallest of their input values, so that a single measured value alone crossing a limit would not lead to (potentially spurious) actuation.

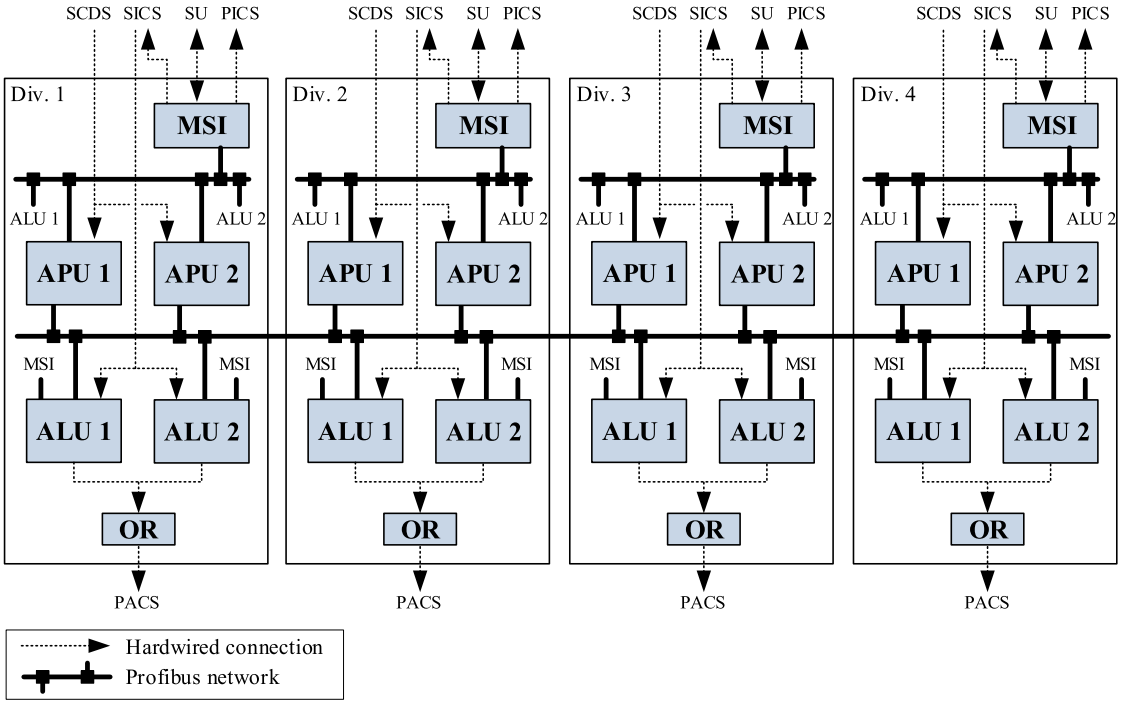

Fig. 3. Simplified architecture of the U.S. EPR Protection System (modified from [33]).

3.2. Work process for model checking

Based on previous research [39] and practical experience [40], VTT has developed a work process for the verification of vendor-specific, non-standard I&C logics[4] (see Fig. 4).

The first tasks are model boundary definition and requirement elicitation. Functional requirements can be found in system requirement specifications, or other documents containing suitable descriptions (e.g., a user manual).

The modelling work begins with the construction of a library of basic function blocks, using the input language of NuSMV [40]. Manual specification of the block library is necessary, since nuclear I&C system suppliers use vendor-specific solutions (such as signal validity). The proprietary source code for the blocks might not be available, so specification is based on functional descriptions (e.g., a function block user manual). Confidence can be gained by verifying the blocks' implementation against formal properties, checking the equivalence of differing implementations created by two analysts independently, or synthesis-aided methods [42].

The analyst then models the selected I&C functions using the library of basic function block code elements, and specifies formal properties in LTL, CTL and/or PSL [29] based on the collected requirements.

A NuSMV counterexample can reveal an error in the model or an incorrectly specified property (causing a "spurious counterexample" [43]), in which case the analyst fixes the error, and runs NuSMV again [39]. If NuSMV produces a counterexample that—after close inspection of the source documents—can only be explained by a problem in the design, the issue is documented and reported to relevant stakeholders.

The only fully automated part of the process is verification with NuSMV. In the next section, we discuss tool support for I&C function modelling and counterexample interpretation.

3.3. MODCHK—A practical i&c model checking tool

VTT has developed a graphical tool [44] called MODCHK (Fig. 5) for verifying I&C application logics based on function block diagrams with NuSMV.

MODCHK is used to:

1. Specify a library of vendor-specific basic function blocks with a text editor.
2. Model the block diagrams with a graphical editor. Composite blocks can also be specified, allowing for multilevel hierarchy [44]. The composite elements are especially useful for modelling distributed nuclear applications with redundant, identical divisions (see Fig. 5).
3. Specify the properties with a text editor.
4. Generate the necessary input files and run NuSMV.
5. Visualize the counterexamples produced by NuSMV with an animated view [43] of the block diagram (see Fig. 5). Signals with an invalid status are shown with a dashed line.

Each signal between the blocks carries both a Boolean or integer value V, and the associated validity status (as a Boolean variable V_FAULT). Inside the basic blocks, each input is also assigned a variable V_CONNECTED, allowing the analyst to specify how unconnected inputs affect the processing logic.

The status processing logic is explicitly defined for each basic block. As an example, in TELEPERM XS systems, "passive status processing" means that the status of the output signal is formed by simply OR-gating the status of each related input [40]. In "active status processing", the output value is only calculated from valid input signals [45].

To demonstrate "active" status processing, we show below the code for an exemplar function block. The 1-out-of-2 voting block will output TRUE if at least one of the two input signals is TRUE and valid, or if both inputs are invalid.

[4] While function block diagrams are a "programming language" [41], we discuss the verification of "logic" rather than "software". First, the actual source code (automatically generated based on the diagrams) is typically unavailable in nuclear industry projects. Second, we also work with FPGA logics, which cannot be called software.

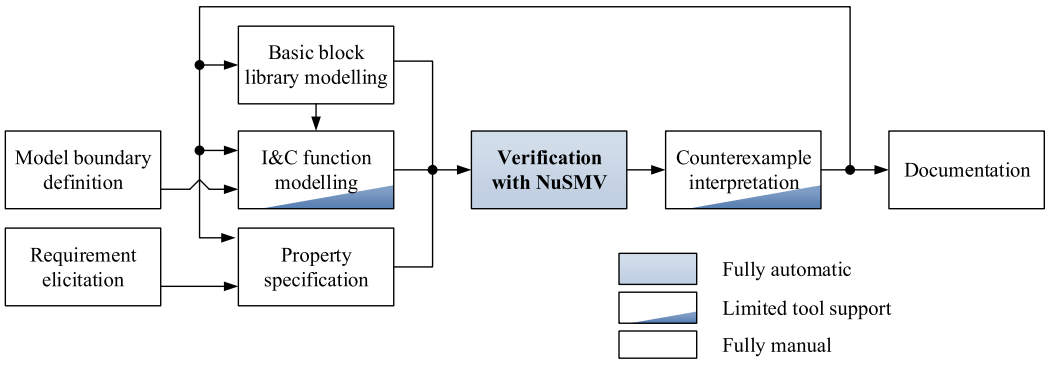

Fig. 4. Work process for the verification of vendor-specific I&C application logics.

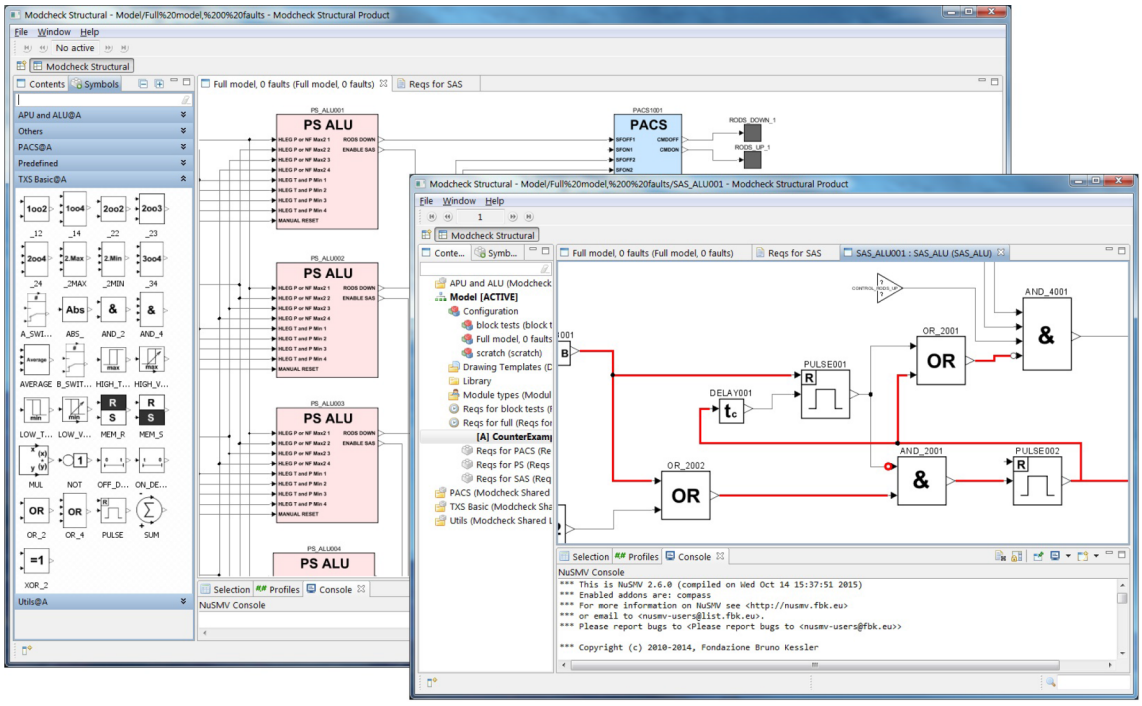

Fig. 5. MODCHK is a graphical tool for verifying composite function block diagrams with NuSMV.

```
MODULE _1oo2(
    BI1, BI1_FAULT, B1_CONNECTED,
    BI2, BI2_FAULT, BI2_CONNECTED)
DEFINE
  BO1 := case
    BI1_FAULT & BI2_FAULT : TRUE;
    (BI1 & !BI1_FAULT) |
    (BI2 & !BI2_FAULT) : TRUE;
    TRUE : FALSE;
  esac;
```

3.4. Practical results from the nuclear industry

Since 2008, VTT has applied model checking in practical customer projects in the Finnish nuclear industry [16]. The clients include (but are not limited to) the nuclear regulatory body and two plant operators.

Olkiluoto 3 is an EPR under construction. On commission from the Finnish Radiation and Nuclear Safety Authority (STUK), VTT has evaluated the application logic of the Protection System and the Priority and Actuation Control System. Both systems are based on the TELEPERM XS platform, PS being a software-based system, and PACS based on Field Programmable Gate Array (FPGA) technology.

The Loviisa NPP includes two reactors of the type VVER-4400. Old analogue I&C systems have been replaced with modern digital technology (based on Rolls-Royce's Spinline platform) in a renewal project. On commission from the plant operator Fortum, VTT has performed independent, third-party verification of the application logic of seven different I&C systems. Based on the results, design modifications have been made to the Reactor Trip System and the Reactor Power Control System [16].

Hanhikivi 1 is an NPP planned to be built in Pyhäjoki. The utility Fennovoima has submitted a construction license application for an AES-2006 type reactor. On commission from Fennovoima, VTT has evaluated the Hanhikivi 1 functional architecture, which uses function block diagrams to describe the safety functions of the plant in an early design stage. VTT has detected design issues that could lead to spurious actuation, contradictory commands, or otherwise incorrect response.

In all the above projects combined, between 2008 and 2019, VTT has identified 57 design issues in I&C application logic. The reader should note that the issues are about a single system not behaving according to its stated requirements in some particular scenario, however

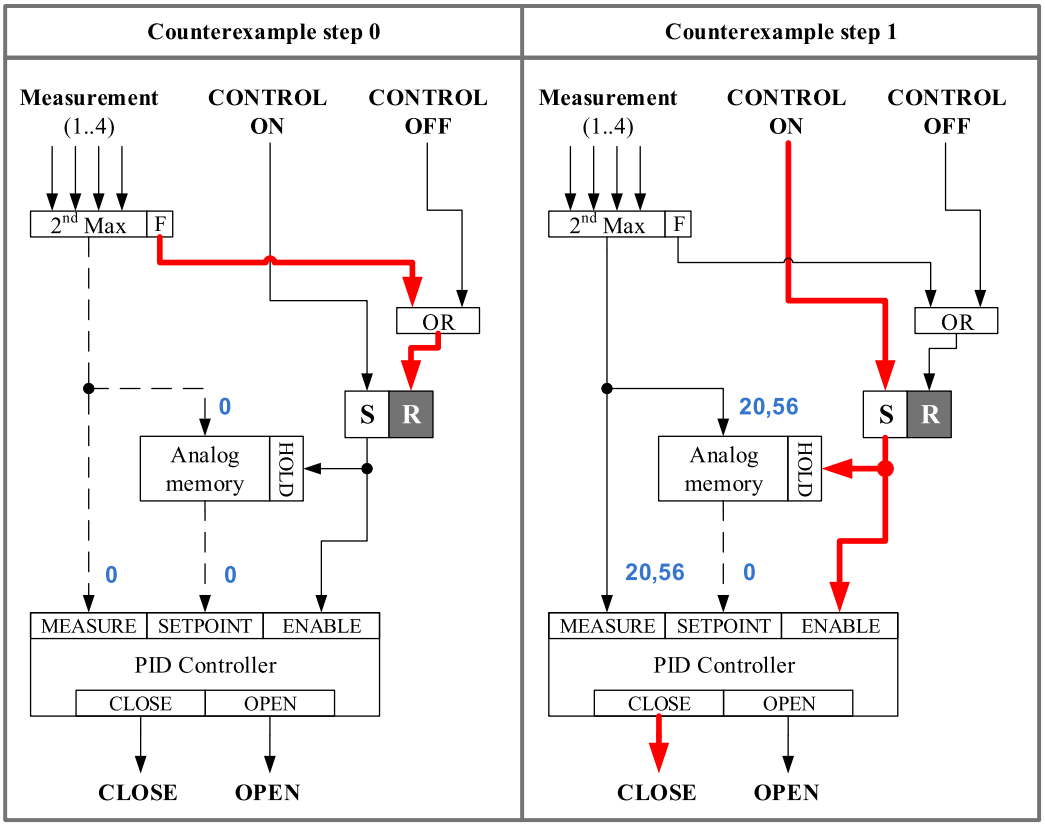

Fig. 6. A counterexample scenario for spurious actuation (actual industry example).

unlikely. We wish to emphasise that we do not discuss the safety relevance of any issue, which may be purely theoretical[5] Some of the systems or functions have been considered in isolation–not accounting for, e.g., the characteristics of the controlled process, or safeguards built into other related systems–and any issue might therefore be practically irrelevant in its actual context. VTT only modelled the application logic, and no hardware failures were assumed. All the issues can therefore be attributed to human error in the application logics' design processes.

Of the 57 design issues, 21 (37%) deal with spurious actuation. We can further divide these scenarios in two types:

1. Spontaneous spurious failure occurs when the criteria for actuation has not been valid before the actuation command.
2. Stuck-on spurious failure occurs when the actuation command has first been given based on valid criteria, but the command remains set while the criteria no longer justify or allow it.

Sixteen of the detected issues are of the spontaneous type, and five of the stuck-on type. The characteristics of the issues are described in Appendix A. For each issue, we present a short generalized description, and then list the property that revealed the issue, elements in the design that introduce complexity, the number of function blocks and NuSMV variables, the number of reachable states in the model, and time it took NuSMV to verify the failing property.

The analysis times are based on experiments with an Intel Core i7-6600U CPU with a clock rate of 2.6 GHz. NuSMV 2.6.0 was run on a single core with the options "`-int -dynamic`".

Some of the issues were found with a composite model containing several different safety functions of the same system(s), which explains the model characteristics that are shared between issues 1–3, 6–7, and 11–12. In total, the issues in Appendix A were found in six different I&C systems.

Notably, the "universal" spurious property (formula (6) in Section 2.3) occurs in five of the examples, in two cases preceded by $\mathbf{G}(p) \rightarrow$, which is usable for filtering out irrelevant executions, looking for more counterexamples, or seeking to better understand model behaviour [29].

The PSL property `never{p[*n]}` is obviously convenient for verifying that signal `p` does not get stuck on (for `n` cycles, at least). In LTL, the equivalent property is expressed with nested **X** operators.

3.5. A practical example of spurious actuation

We now present an example of a real design issue resulting in spurious actuation, revealed using model checking in a practical industry project (issue 8 in Appendix A). The originally verified application logic consisted of 73 function blocks. Here, we only include the five blocks that are needed for reproducing the problematic scenario. The appearance and the detailed processing logic of the function blocks have been modified to obscure their origin. The simplified, masked application logic can be found in Fig. 4. The PID Controller element is represented in the NuSMV model by a simple abstraction.

The intended (here, simplified) functionality is PID control of a safety actuator. When the control mode is switched on, the current measurement value (based on second-minimum voting over four redundant signals) is memorized, and then used as the setpoint for the controller. When the control mode is switched off, the controller is disabled.

The analyst discovered the issue by verifying a property not specifically addressing spurious actuation, but the selection of the controller setpoint. The property read, simplified: **G**

[5] Discussion on the potential plant level effects for some of the issues can be found in [46].

$\neg((\text{measurement} = 0) \wedge \text{measurement\_FAULT}) \rightarrow \mathbf{G}\,\neg\,((\text{setpoint} = 0) \wedge \text{enable}))$,

"measurement" meaning the output of the "2nd Max" block. In other words, "assuming that the valid (actual) measurement is never zero, then zero shall never be selected as the setpoint". The issue would also have been revealed by the spurious property (type (6) in Section 2.3): $\mathbf{G}(((\text{setpoint} = n) \wedge \text{enable}) \rightarrow \mathbf{O}((\text{meas} = n) \wedge \neg\ \text{measurement\_FAULT}))$, i.e., for any allowed integer value n, "n can only be selected as the setpoint if a valid measurement has at some point had the value n." The counterexample output by NuSMV is visualised in Fig. 6.

The counterexample begins at the initial state, where the measurements have invalid status (dashed line in Fig. 6). The "2nd Max" block will in such a case output a user-configurable initial value, for which the pre-set default value is zero. The value is output as such (including validity) by the "Analog memory" block.

In the next state, the control mode is switched on at the exact same processing cycle where the measurements turn valid. The "Analog memory" block is switched to hold mode, and then keeps outputting the previous input value for as long as the hold mode is on.

As the now valid measurement is almost certainly more than zero, the simplified PID Controller element then issues the "close" command spuriously. The analyst is aware that the PID element is not accurately modelled, but also realises that a realistic PID component would—given the incorrect set-point zero—also issue the "close" command.

The scenario would be difficult to detect using testing or simulation, because:

1. situations where the system is rebooted to its initial state are very rare, and here the reboot also coincides with invalid measurement data,
2. two unrelated events (measurements turn valid, and control mode is switched on) need to occur at the exact same processing cycle, and/or
3. the analyst should know to focus on intricate details on how two of the 73 blocks process validity.

With MODCHK, locating the cause of the spurious actuation is typically straightforward. The animated view highlights when the signal targeted in the property is actuated, and the analyst can easily find the states in the counterexample where that is the case. As the counterexample can also contain states where the signal is actuated on demand (e.g., the spurious actuation is of the "stuck-on" variety), the analyst then needs to check the states to find the one(s) where the signal is active without demand. Visualisation of the signal values helps the analyst in navigating the diagram to see if the proper criteria are fulfilled. For LTL properties, we have also developed a counterexample explanation tool that can locate the state where the failure occurs [43], which is particularly useful if the counterexample is long.

4. Single failure tolerance in model checking

4.1. Modelling approach for single failures

So far, we have only considered application logic issues, with the assumption that the underlying hardware does not fail. Below, we broaden our scope by including hardware failures.

Previous work on similar I&C application logic models [47] has shown that modelling hardware failure modes to each processor and communication link results in excessive verification times. We therefore have to simplify the failure model, which is possible if we focus on single failure in open-loop models. To illustrate the concept, we use the U.S. EPR PS (see Fig. 7), but the same approach is applicable to any system with redundancies and a voting unit.

For the N + 2 criterion (see Section 3.1), we assume that the division that is out of operation due to repair or maintenance will not drive the actuators. We assume that an inhibition function is built into the hardware and/or application logic. (For example, when a sensor is placed in a maintenance bypass, a lockout attaches a faulty status to the sensor's signal [37].) The out-of-operation division will thus only issue actuation orders if it fails. Below, we therefore focus on single failure tolerance, in general.

As shown in Fig. 7, we use several simplifications. A single failure can disable an entire division (e.g., consequential failure due to power supply). That also means that the other three divisions cannot have any failures (at the same time), otherwise the single failure criterion no longer holds. Therefore, it follows that:

1. It is sufficient to model the failures for one division only.
2. We can pick any non-failing division, and if the divisions are identical, any verification result for that division (in Fig. 7, div. 1) will hold for at least two other divisions. Therefore, there is no need to model the outputs of the other divisions, which is why it is sufficient to model the voting logics (ALUs) for just one division. The processing logic before the selected ALU is needed, which is why we include all the redundant APUs, but not the other ALUs. Such a simplification is routinely used in VTT's work to reduce the models' complexity.

The included ALU cannot fail, because we have assumed that the fully modelled division is not the failing one. In open-loop analysis, we are not interested in the output of the potentially failing ALU, which is why it can be excluded from the model altogether.

Another significant simplification we use is the injection of failure points to a limited number of locations. At those locations, we assume that the HW failure can have occurred in any part of the system that has processed the signal by that point, and the signal is faulty from that location on. Since we are primarily interested in how the application logic running in the APU and ALU function processors interacts with hardware failures, we inject the failures in two locations: (1), for each signal entering the APUs of the failing division, and (2), for each signal leaving the APUs of the failing division (see Fig. 7), i.e., between the APU and ALU.

The final simplification has to do with how the failures are modelled on the signal level. The model is not based on detailed analyses of realistic hardware component failure modes. Instead, at the specified failure points, the correct value of each signal is simply replaced with a nondeterministic variable. The validity (status) of each signal is also nondeterministic, i.e., the failure can be self-announcing or non-self-announcing. Due to the nondeterminism, each signal may fail independently, or all signals may fail simultaneously (some passively, some actively).

The two failure injection points cover all conceivable failure modes of the hardware components [36,37] listed in Table 1. We assume a network topology where any failure in a failed division cannot affect the APU—ALU communication between the remaining redundant divisions.

The failure model is implemented in MODCHK (and the corresponding NuSMV model) by inserting a module in the selected division to signal connection to/from the affected APUs. Module FAULT_BIN is used for Boolean and FAULT_ANA for integer signals. Both blocks have the output FAILURE that allows the analyst to also observe the non-self-announcing failures. The NuSMV source code for FAULT_BIN is shown

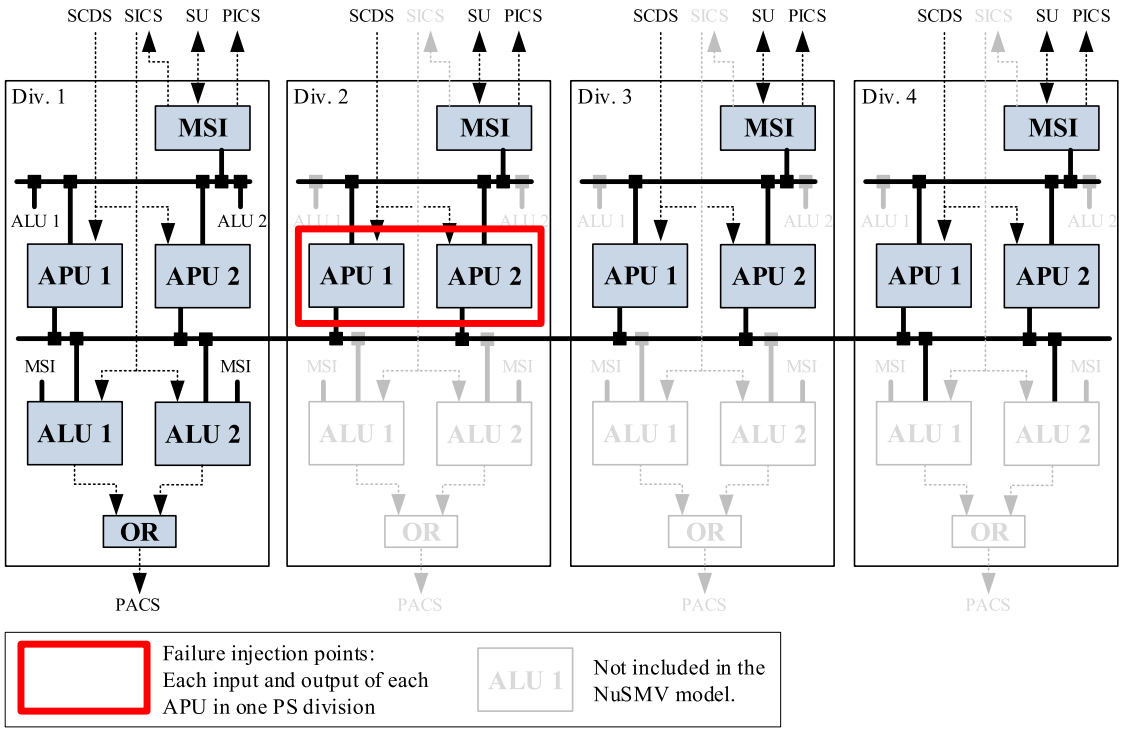

Fig. 7. Simplification of the NuSMV model based on symmetry.

below.

```
MODULE FAULT_BIN(BI1, BI1_FAULT,
    BI1_CONNECTED)
VAR
  fail : boolean;
  spurious : boolean;
  announcing : boolean;
DEFINE
  BO1 := fail ? spurious : BI1;
  BO1_FAULT := fail ?
    announcing : BI1_FAULT;
  FAILURE := fail;
  FAILURE_FAULT := FALSE;
```

For FAULT_ANA, the integer values the signal can have upon failure are given as a user-configurable parameter. To limit the state space growth, the analyst might have to select only a few possible values. For the input signals to APUs, the analyst can use the similarly discretized values as the model inputs. For APU—ALU communication, the choice might not be as straightforward, but the PS function diagrams in [37] show only binary signals exchanged between APUs and ALUs.

4.2. Experimental evaluation

We evaluated the modelling approach using actual industrial examples. We collected suitable models that VTT had constructed for both TXS and Spinline based four-redundant safety systems in practical projects. Failures were then manually modelled for a single division. For each model, between four and seven original properties were selected, including true and false properties, and both CTL and LTL/PSL. If needed, the properties were modified to account for the injected failures. NuSMV was then used to calculate the effect of the failure injection on the model state space and the analysis times.

The selected models did not represent the most complex functions that VTT has verified, because such functions often have to be verified by including only a single division, or one logic unit, or even just parts of a logic unit. Here, we included models where all four divisions were originally modelled without severe performance issues.

The results are listed in Table 2. The analysis time is the average for one to five different properties per language. NuSMV was executed as explained in Section 3.4. All the CTL properties were true, while a third of the LTL/PSL properties were false for both models. The numbers of function blocks and NuSMV variables are for the original model, and do not include the failure modelling elements.

From the results, it is apparent that computational overhead from single failure modelling is practically negligible. The analysis times might even decrease (see example 1). Despite a significant increase in the state space (particularly examples 8 and 11), the maximum absolute increase in analysis time was 16,3 s (example 2). Relatively, the highest

Table 1
I&C hardware components that have their failures covered in the model.

Failure point	Hardware components and systems
Before the APU	• Process sensors • SCDS – signal condition modules – signal distribution modules • Service Unit (SU) • Hardwired connections – sensor to SCDS – SCDS to APU – SU to MSI • Communication network – MSI to APU • APU – input module
After the APU	• APU – function processor – communication module – optical link module
Both points simultaneously (consequential failures)	• Power Supply (Class 1E uninterruptible power supply, EUPS) • Heating, ventilation, and air conditioning (HVAC) system

Table 2
Comparison of NuSMV performance for no-failure against single failure models.

			No HW failures			One HW failure			
	Function blocks	NuSMV variables	Reachable states	CTL (s)	LTL / PSL (s)	Reachable states	Normalized reachable states	CTL (s)	LTL / PSL (s)
1	51	230	$1,75 \cdot 10^{20}$	32,8	75,1	$3,14 \cdot 10^{23}$	$2,04 \cdot 10^{20}$	4,90	57,5
2	179	893	$3,96 \cdot 10^{25}$	11,8	63,7	$6,64 \cdot 10^{32}$	$1,62 \cdot 10^{29}$	17,2	80,0
3	78	354	$1,29 \cdot 10^{15}$	1,40	2,70	$2,52 \cdot 10^{18}$	$1,64 \cdot 10^{15}$	3,70	3,80
4	72	318	$5,20 \cdot 10^{20}$	1,20	0,92	$1,38 \cdot 10^{29}$	$2,06 \cdot 10^{21}$	2,30	1,50
5	76	388	$8,85 \cdot 10^{45}$	0,92	1,40	$3,12 \cdot 10^{59}$	$8,85 \cdot 10^{45}$	1,20	1,90
6	118	785	$2,18 \cdot 10^{24}$	0,45	1,70	$3,57 \cdot 10^{28}$	$8,73 \cdot 10^{24}$	0,92	2,50
7	140	994	$7,95 \cdot 10^{28}$	0,30	0,66	$6,65 \cdot 10^{35}$	$3,17 \cdot 10^{29}$	0,46	0,85
8	100	694	$6,56 \cdot 10^{8}$	0,23	0,47	$9,28 \cdot 10^{19}$	$3,00 \cdot 10^{15}$	0,34	0,63
9	85	484	$4,74 \cdot 10^{21}$	0,19	0,28	$9,68 \cdot 10^{24}$	$1,89 \cdot 10^{22}$	0,30	0,44
10	211	1229	$1,57 \cdot 10^{13}$	0,11	0,50	$7,21 \cdot 10^{16}$	$1,57 \cdot 10^{13}$	0,14	0,27
11	143	800	$4,50 \cdot 10^{15}$	0,10	0,17	$3,96 \cdot 10^{28}$	$4,50 \cdot 10^{15}$	0,95	0,46
12	73	385	$7,30 \cdot 10^{17}$	0,08	0,07	$9,18 \cdot 10^{24}$	$5,84 \cdot 10^{18}$	0,24	0,22
13	26	100	$5,80 \cdot 10^{9}$	0,03	0,26	$6,83 \cdot 10^{13}$	$6,67 \cdot 10^{10}$	0,08	0,39

increase can be seen in examples where the analysis time was still parts of a second. In example 2, the relative change is a 20% increase for CTL and a 46% increase for LTL/PSL. We further found that the increase of the state space is largely connected with the addition of non-determinism within failure blocks—for example, adding a failure block that can nondeterministically substitute the signal (as well as its validity status) by two different values expands the state space in at least four times. We removed this effect by dividing the number of states by the magnitude of this expansion, and the result is shown in the "normalized reachable states" column of the table. Notably, in three cases the result is the same as with no failures, meaning that the addition of failures only influences the output values of models, not their memory-stored state.

Finally, we visualized the connection between the number of reachable states (normalized in case of one failure) and model checking time in Fig. 8 where each data point corresponds to a pair of a model and a number of failures. In order to make the distributions of state space sizes and times close to normal, we took the logarithms of all the values. The logarithm of model checking time of each checked property was taken independently, and then the mean was calculated (thus, the obtained values do not strictly correspond to any of the columns in Table 2). Fig. 8 also shows linear regression trend lines for the cases of no failures and one failure separately.

5. Related research

5.1. I&c software risk assessment

A lot has been written about the risk that spurious actuations pose to plant safety, and the difficulty in analysing such failures. Proposed solutions are harder to find. Nuclear regulatory bodies are working towards consensus on regulatory guidance on evaluating spurious actuations [48].

Approaches for analysing the effects of spurious failures using probabilistic safety assessment (PSA) are explored in [1]. PSA is a well-established methodology for risk based-decision making. In nuclear applications, it is used during design and operation for, e.g., identifying weaknesses, and prioritizing components and systems for safety classification, testing, inspection, and maintenance [49]. Spurious actuation failure modes can be "quite well covered" [1] in modern PSA models, but there is no consensus on how to treat software reliability [1,50]. There is also limited operational data available that would support plausible reliability estimates for I&C application logics [46], since failures of safety system application software are rarely observed during operation [50].

Our work, in turn, shows how design issues causing spurious actuation can be detected (and therefore eliminated) through deterministic analyses. In addition to reducing the risk in practice, the results (that indicate existence or non-existence of design faults) have an impact on the estimated probability of a software failure [46]. Furthermore, the scenarios that model checking reveals (see Appendix A) could lead to new failure modes and/or fault propagation paths added to the PSA model [46], improving coverage.

In any case, a realistic assessment of overall safety calls for a balanced combination of both probabilistic and deterministic analysis methods [51].

5.2. I&c application logic verification

A great deal of the research on I&C system model checking has focused on the automatic generation of models based on standard PLC languages, either IEC 61131-3 (e.g., [52–54]), or IEC 61499 [55]—or, in the case of FPGAs, directly based on hardware description languages [56]. As we point out in Section 3.2, such approaches are not directly applicable in the nuclear industry, where vendors use proprietary function blocks with industry-specific, non-standard features like status processing.

Due to the difficulty in mastering temporal logic languages, another key research topic is user-friendly property specification. Dwyer et al. have published an influential collection of property specification patterns [31]. I&C domain specific patterns have been suggested in, e.g.,

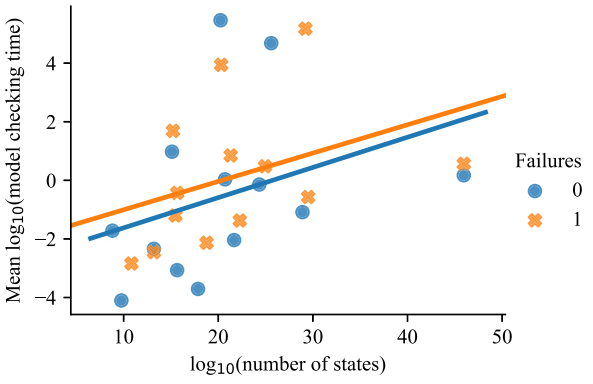

Fig. 8. Scatter plot showing the dependency between the number of states in the model and model checking time.

Table 3
Related work on model checking and fault tolerance.

Ref.	Concept	State space size	Analysis time	Model checker(s)	Case system(s)
[8]	Deviation analysis	–	–	NuSMV	Aircraft altitude switch
[47]	Functional verification incl. failures and plant transients	–	30–110 s	NuSMV	Nuclear safety systems
[9]	Functional verification incl. failures	–	–	SCADE, Simulink verifier	Aircraft wheel brake system
[10]	Functional verification incl. failures	196 608	30 s	Spin	Spacecraft controller
[13]	Functional verification incl. failures	$\leq$ 122 767	–	AMC/JACK	Railway interlocking system
[11]	Functional verification incl. failures	–	–	UPPAAL	Aerospace system
[12]	Functional verification incl. failures	$\leq 1{,}0 \cdot 10^{13}$	$\leq$ 12 min	NuSMV	Platform for satellite control systems
[14]	Functional verification incl. FTA + FMEA	–	–	NuSMV	Automotive break-by-wire system
[65]	Functional verification incl. FTA + FMEA	$\leq 3{,}8 \cdot 10^{25}$	$\leq$ 13 min	NuSMV	Aircraft power supply, braking system
[66]	Model checking for automated FMEA	$\leq 35 \cdot 10^{9}$	$\leq$ 4 days	SAL	Metal press, mine pump, drug infusion pump
[67]	Model checking for automated FMEA	–	4–10 min	Spin	Interface definition
[68]	Probabilistic model checking in FMEA	$\leq$ 615 600	$\leq$ 12h	PRISM	Airbag system

[57], and different graphical languages are listed in [29] and [58]. In [59], formal requirements for automotive applications have been collected in order to develop a specification language for I&C engineering.

In the nuclear domain, model checking has been applied for I&C software verification in Korea [60], Hungary [61], and at the European Organization for Nuclear Research (CERN) [54]. In [60], successive revisions of IEC 61131-3 function block diagrams were checked for equivalence. In [54], IEC 61131-3 programs were transformed into a network of synchronised automata. After property-preserving reductions, the intermediate model was then verified with nuXmv [62]. In [61], the verified system is also based on TELEPERM XS technology, and the model includes signal status processing. The block diagram was first reinterpreted as an "almost identical" Petri net, and CTL properties were then verified with model checking. The authors however had to constrain possible input values to the "most significant failure scenario" in order to bring the number of states down to 46811. The use of symbolic model checkers is mentioned as a future option.

Our approach has built-in support for custom (non-standard) function blocks, omits intermediate model transformation steps, and requires no simplification of the model beyond the abstractions imposed by heuristics. The downside is that the library of basic blocks has to be specified manually.

5.3. Analysing hardware failure tolerance

Several studies address failure tolerance in the context of model checking (see Table 3 for examples). In [9] and [10], failures are added into a model of a two-redundant aerospace system, in order to verify tolerance against single failure. In [12], an error model is added to a nominal model of a satellite control system platform. In [13], the target is a railway system. In [11], real-time model checking is used to verify a three-redundant aerospace system. In [8], a system model duplicate is given incorrect measurement data, and its performance is then compared with the original model.

Failure modelling is related to failure mode and effects analysis (FMEA) [63]—a bottom-up method for reviewing potential component-level failure modes and their system-level effects–and fault tree analysis (FTA) [64]—a top-down method where an undesired system state is broken down to component-level events. In [14], FTA and FMEA are used to assist in the modelling of transitions from normal to failed states, and NuSMV is then used to verify system model conformity. In [65], a NuSMV add-on called NuSMV-SA—capable of dynamic FTA and FMEA–is used to analyse the behaviour of the system model in degraded conditions. In [66] and [67], model checking is used to partially automate FMEA by searching for system-level consequences of low-level failures, with [67] focusing on software issues. In [68], probabilistic model checking is used to identify the components that contribute the most to system-level failures.

A limitation in many of the proposed methods is that the system model has to be kept very abstract, or the model becomes too complex to be verifiable in reasonable time [10,47]. Instead of detailed design, the target of verification is the "functional behaviour" [66], "early" functional [14] or architecture level [12] model, "specified behaviour" [10], or other simplification. Model state spaces, when revealed, can be relatively modest, processing times still rather long (see Table 3), and there may exist "serious questions about the scalability" [9]. (It should of course be noted that the performance data is not directly comparable to ours[10]. and [13], for example, were published two decades ago, using tools and processing power available at the time.)

Closest to our work is [47], where nuclear I&C application logic models similar to ours are supplemented with detailed hardware failure modes, as well as accident and transient scenarios. A model containing seven different safety systems is then verified using NuSMV. However, the application logics have to be kept very simple, and even then, only BMC based verification of invariant properties is practically feasible [47].

Using our approach, we were able to include the hardware failure modes without having to simplify the application logic model.

6. Discussion

The obvious conclusion from VTT's customer work is that model checking truly is an effective method for detecting I&C application logic design issues that could result in spurious actuation of critical systems. In addition, the data collected from the industry projects reveals interesting statistics about what it is in the design of the logics that can cause them to fail.

From Section 3.4, we see that each of the failed designs contained either a memory or a delay element, or both. A memory element can be either a flip-flop switch or latch, or any element that stores a value (e.g., a block whose output is based on last valid input value). Feedback loops were found in four designs. (A feedback loop also introduces a memory/delay element out of necessity, since the processing order of the blocks needs to be explicit.)

One strength of function block diagrams as a programming language is that it is relatively easy to understand the "flow" of processing from inputs to outputs. However, memory and delay elements, feedback loops, and blocks processing signal validity in an active way, all interfere with the flow, making it harder to design and manually review the logic. Delay and memory elements are also very common in I&C application logics, due to, e.g., the dynamic characteristics of the processes being controlled [29].

Let us also consider the oft-occurring characteristics of the counterexample scenarios that revealed the design issues. First, in six cases, the issue involved very exact timing of external events (independent issues occurring on the same processor cycle). Second, four of the issues

Table 4
Oft-occurring characteristics in the failed designs and the scenarios that revealed the failures.

		Spurious actuation issues (count: 21)	All design issues (count: 57)
Elements in the design	Memory block (e.g., flip-flop switch)	76%	68%
	Delay block	76%	60%
	Feedback loop	19%	28%
Features in the counterexample scenario	Exact timing of external events	33%	33%
	Human user actions	24%	28%
	Interaction of several I&C systems	14%	14%
	Validity processing logic	19%	11%
	Permanently frozen state	–	5%

involved human actions, i.e., personnel (operation of maintenance) doing something ill-advised and/or ill-timed. Third, three issues required the interaction of several I&C systems for the problem to occur, and analysing the systems in isolation would not have revealed the issue. Validity processing logic was crucial in four issues.

In Table 4, we also list the prevalence of the above-mentioned characteristics in all the 57 design issues identified in the industry projects (including both active and passive failure). Out of the designs that failed to actuate, three failed in a way that a signal froze permanently to some value, requiring, e.g., system restart for recovery.

It is hard to draw further conclusions from the fact that 37% of the detected issues were of the spurious kind. Most of the designs analysed in the projects had already undergone verification and validation (V&V) based on methods that are more conventional. Since spurious failures are by their nature harder to analyse than passive failures [1], it is likely that the spurious cases are over-represented in our data. The "true" statistical share of hidden design issues potentially leading to spurious failures is difficult to estimate.

In Sections 3.4 and 4.2, the number of reachable states in the NuSMV model is shown to be as high as $8{,}85 \cdot 10^{45}$, or $3{,}12 \cdot 10^{59}$ when hardware failure modelling is included. Let us consider how such a high number is possible, when the modelled system consists of mostly binary logic. For n binary signals, there are obviously 2^n possible value combinations, or 2^{2n}, if we include the status of each signal as a free input. Internal memory elements also increase the state space. However, our models are not limited to binary logic. Since NuSMV can only handle integer numbers and discrete time, we have to discretise several model signals:

1. Analogue inputs are modelled as a range or enumeration of integer variables. To limit the growth of the state space, the analyst can limit the enumeration to a set of values that are sufficient to allow for all relevant executions [40], e.g., by selecting one value below a limit threshold, and another from above it.
2. If an analogue signal is memorized in the logic (e.g., a last-valid-signal logic, or a cycle step delay needed to implement a feedback loop), an integer variable needs to be added. If the analyst cannot narrow down the possible values to a small set, the computational cost increases significantly.
3. Each delay element needs its own internal clock variable, another integer variable, for which a suitable range needs to be specified [40].

When we introduce the hardware failures, not only do we add several free inputs (for each FAULT_BIN and FAULT_ANA element), we also allow the model to reach states that it would not reach if the signals inside the logic were deterministic (which, of course, is exactly our objective in failure modelling).

Nevertheless, it is important to note that the number of reachable states alone is not a sufficient (or necessarily even credible) measure of computational complexity in model checking. From Table 2 in Section 4.2, we can see that a model with $3,12 \cdot 10^{59}$ reachable states is analysed in less than two seconds, while a model with $1,75 \cdot 10^{20}$ reachable states takes more than a minute to process. It is obviously important to consider the analysis time when assessing practical applicability.

We made no effort to determine what the realistic failure modes for the underlying I&C hardware components could be. In our non-deterministic failure model, any kind of failure–conceivable or not—is allowed. From the point of view of safety, such a view might be preferable. Still, the most unlikely failure we can think of is the "chaotic" behaviour of a function processor (or an output module) where some of its outputs fail actively while others passively. Such a scenario can be feasible if the components overheat, which is possible if, e.g., the ventilation system fails—a single failure [3,18]. Risk-based analysis methods can be used to decide whether the cost of redesigning the application logic to tolerate a highly improbable failure mode is justified [49].

In our experiments, we were able to show that the failure modelling approach is applicable to models of real-world systems created in practical industry projects. The computational overhead is practically negligible. However, our experiments did not allow for a complete re-verification of the target systems, and in any case, we did not identify any previously unknown design issues. Whether design issues involving both the I&C application logic design and the underlying hardware component failures are actually identified (and how common they are) remains to be seen in future customer projects.

Regarding the validity of our experiments, the failure modules were inserted manually, using a graphical environment. Since our tools do not yet automate the process, verifying that the failures were correctly modelled was based on manual review. Second, any claims that we make about the complexity of verification problems focus on the practical applicability of using a specific tool. The heuristics of NuSMV may be biased, which prevents us from making objective assessments of the "actual" increase in complexity resulting from failure injection. Third, there are many ways to express temporal properties, and the selections used in our work may have a bias concerning the complexity of verification. However, we assume that model checking performance depends more on the model and the selected algorithm than the property.

When using any formal method for verifying safety critical systems, it is important to recognise the limitations in the formal descriptions, methods and tools used [5]. In our approach:

- What is verified is whether the way function blocks are connected in the diagram can result in unwanted behaviours. Modelling is based on functional descriptions, and no assumptions are made about the correctness of the function block source code, generated application logic source code, or compiled machine instructions [40] (or, in case of FPGA, the generated netlist or configuration).
- The model is representative of the actual application logic only to a certain degree. NuSMV can only handle elementary mathematical operations with integers. Discretisation of real numbers and other necessary abstractions are performed manually.

- It is difficult to guarantee that all relevant properties have been properly formalised. The requirement specification serving as input might not be complete [40]. In any case, effort is needed from the analyst to consider every aspect of a requirement (see Section 2.3).
- It is possible to make a human error both in specifying the model and in formalising the property, so that the (false) property holds for the (false) model, hiding both errors, and producing incorrect results [40].
- Correctness of the model checker cannot be exhaustively proved [40].

7. Conclusions

In this paper, we have shown that model checking can be used to detect issues that could lead to spurious actuation of industrial I&C systems, regardless of whether the cause of the actuation is a systemic failure due to human error in application logic design, or a random failure of hardware component. The way that we construct the model accounts for plant transients, human actions (operators or maintenance personnel), environmental conditions, and failures of support systems (e.g., power supply, ventilation, testing equipment). We can inject the hardware failure modes in the model without resorting to further abstraction and simplification in the way the application logic is modelled. Model checking enables exhaustive verification, but only for the properties that have been specified. The analysis results can only be conclusive if the human user has captured every relevant property. While direct errors in property specification are often revealed by spurious counterexamples [43], omissions are another matter. For any functional requirement addressing the intended response of the system, there may be several properties needed to rule out unwanted behaviour. To make matters worse, there is no silver bullet for the more general challenge of making formal property specification easy [43].

So far, we have focused on a single I&C system. When we consider the entire overall I&C architecture of a nuclear power plant, single failure tolerance becomes more complex to verify. If it takes several different I&C systems to carry out a safety function, the failure criterion must be applied to the entire chain of systems–from measurements to actuators, including all supporting systems—as a functional whole. In our future work, we therefore need to broaden the scope. After all, 14% of the design issues VTT has revealed result from the way several different I&C systems operate as a whole.

In our tools, further development is needed for automatic import of the block diagrams, property editing, and counterexample explanation [43]. We also aim to investigate whether tools such as nuXmv [62] or HyComp [25] could be used to avoid some of the modelling simplifications imposed by NuSMV.

The difficulty in analysing spurious actuation is tied to the challenge of achieving 100% test coverage. The obvious benefit of formal verification is exhaustive coverage, and the ability to address unwanted behaviour just as well as intended behaviour. The results of VTT's customer projects prove that systemic errors leading to spurious actuation of I&C systems are in practice detected with model checking. To the best of our knowledge, it is the only truly effective method available. We hope our example inspires more widespread adoption of the method, because there is no reason for the established use of industrial I&C system model checking to be only limited to the Finnish nuclear industry.

CRediT authorship contribution statement

Antti Pakonen: Conceptualization, Methodology, Software, Investigation, Writing - original draft. **I Buzhinsky:** Visualization, Writing - review & editing. **K Björkman:** Conceptualization.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. List of detected design issues with spontaneous (1–16) and stuck-on (17–21) spurious failure

	Desc.	Failed property type	Elements in design	Func. blocks	NuSMV vars	Reachable states	Analysis time (s)
1	Actuation criterion is signals *a* and *b* being active at the same time. However, *a* remains in an internal memory (which is supposed to reset automatically). Later, signal *b* alone can lead to actuation.						
		$\mathbf{G}(p \wedge \mathbf{X}q) \rightarrow (\mathbf{X}r \vee s \vee \mathbf{Y}s \vee \mathbf{YY}s)$	memory, delay, feedback loop	130	530	$4, 52{\cdot}10^{38}$	0,4[a]
2	Actuation of a safety function on a certain channel is supposed to be inhibited if signal *a* is true. However, if a first becomes active on channel 2, and then immediately for the channel 1, actuation can (under certain conditions) occur on channel 1.						
		$\mathbf{G}(p \leftrightarrow q)$	memory	130	530	$4, 52{\cdot}10^{38}$	0,2[a]
3	Actuation criterion is signals *a* and *b* being active at the same time. However, *a* remains in an internal memory (which is supposed to reset automatically). Later, signal *b* alone can lead to actuation.						
		$\mathbf{G}(p \rightarrow \mathbf{O}q)$	memory, delay, feedback loop	130	530	$4, 52{\cdot}10^{38}$	0,2[a]
4	If a channel is put to test mode, test inputs can set a memory element. When the channel returns to normal use, the stored signal leads to actuation.						
		$\mathbf{G}(p) \rightarrow \mathbf{G}(q \rightarrow \mathbf{O}r)$	memory, delay	118	779	$2, 18{\cdot}10^{24}$	23,7
5	If a channel is put to test mode, test inputs can set a delay element. If the channel is immediately returned to normal use, the delayed signal leads to actuation.						
		$\mathbf{G}(p) \rightarrow \mathbf{G}(q \rightarrow \mathbf{O}r)$	memory, delay	118	773	$5, 45{\cdot}10^{23}$	14,0
6	One of the criteria for an OPEN command is delayed in the logic, such that on demand, the CLOSE command is actuated instead.						
		`always {p[*n]}|->{q}!`	delay	60	254	$5, 39{\cdot}10^{18}$	35,7
7	Actuation is supposed to occur if measurement *a* decreases below a limit value. However, actuation occurs at a certain range above the limit, instead.						
	$\mathbf{G}(p \wedge \mathbf{X}q) \rightarrow \mathbf{X}r$	memory, delay	60 254	$5, 39{\cdot}10^{18}$	8,6		
8[b]	Invalid (e.g., off-scale) measurements on system initialisation lead to a default value 0 being stored into memory. If the measurements return to normal at the exact same moment the operator enables a controller, the controller uses the default 0 as the setpoint, and actuates a spurious "close" command.						
		$\mathbf{G}(p) \rightarrow \mathbf{G} \neg q$	memory, delay, feedback loop, controller (abstracted), filter (abstracted)	73	645	$4, 57{\cdot}10^{18}$	126
9	An inhibition signal is supposed to prevent actuation, but it does not.						
		$\mathbf{G}(p \rightarrow q)$	memory, delay	74	338	$1, 29{\cdot}10^{15}$	37,9

10	A delay logic is intended to return certain actuators to their normal position after a safety function actuation has successfully been carried out. However, the delay logic is automatically actuated at system initialisation.							
	$\mathbf{G}(p \rightarrow \mathbf{O}q)$	delay			41	346	3, $21{\cdot}10^{13}$	7,4

[a]BMC was used, since analysis times otherwise exceeded several hours.

[b]The issue is used as an example in Section 3.5.

	Desc.	Failed property type	Elements in design	Func. blocks	NuSMV vars	Reachable states	Analysis time (s)
11	A permissive signal can be set through maintenance actions, regardless of process state justification, by manually setting the state of a memory element. A function can then be actuated while it is supposed to be inhibited.						
		$\mathbf{G}(p) \rightarrow \mathbf{G}(q)$	memory	27	171	2, $81{\cdot}10^{13}$	0,6
12	A permissive signal is set without the required operator acknowledgement if the process state justification for the permissive is valid at system initialisation. A function can then be actuated while it is supposed to be inhibited.						
		$\mathbf{G}(p \rightarrow q)$	memory	27	171	2, $81{\cdot}10^{13}$	0,6
13	Invalid (e.g., off-scale) measurements lead to an "open" command (as intended), but also to a simultaneous "close" command.						
		$\mathbf{G}(p \rightarrow q)$	memory, delay	146	29	4, $29{\cdot}10^{11}$	0,2
14	Contradicting measurements lead to an "open" command (as intended), but also to a simultaneous "close" command.						
		$\mathbf{G} \neg p$	memory, delay	146	29	4, $29{\cdot}10^{11}$	0,2
15	Quick re-initialisation of a test by an operator, in combination with a very short signal pulse from an unrelated safety function, can interfere with the test sequence timing logic, and lead to actuation without need.						
		$\mathbf{G}(p \rightarrow q)$	memory, delay, feedback loop	93	399	1, $52{\cdot}10^{9}$	50,6
16	Measurement signal validity, in combination with how hysteresis is applied to a limit criterion, can lead to actuation even if there has never been a valid measurement above/below the limit.						
		$\mathbf{G}(p \rightarrow \mathbf{O}q)$	memory	13	86	1, $07{\cdot}10^{8}$	0,2
17	A startup sequence timing logic leaves the "start" command on for extended time, if the start criteria activate, the operator then resets the order, and the "start" command is then reactivated through (same or other) criterion.						
		`never {p[*n]}`	memory, delay	52	410	1, $11{\cdot}10^{19}$	50,8
18	A startup sequence timing logic leaves the "start" command on for extended time, if the start criteria activate, reset, and then reactivate with exactly the right timing.						
		`never {p[*n]}`	delay	52	300	9, $48{\cdot}10^{17}$	140
19[c]	If the process criteria activate at the same cycle the operator presses reset, and the process criteria then reset at the same cycle the reset signal ends, the safety function remains actuated (while no longer justified by the measurements).						
		$\mathbf{G}(p \rightarrow q)$	memory	33	146	4, $29{\cdot}10^{11}$	0,3
20	A delay logic is intended to return certain actuators to their normal position after a safety function actuation has successfully been carried out. However, if the process criteria are set, then reset, and then again set with specific timing, both the "on" and "off" commands are activated simultaneously.						
		`never {p[*n]}`	delay	29	168	1, $67{\cdot}10^{10}$	3,2
21	A fluctuating process measurement causes a timing logic to enter a state where one measurement change is not accounted for (due to another signal setting a delay element), resulting in a command that the measurement no longer justifies.						
		$\mathbf{G}(p \wedge \mathbf{X}q) \rightarrow \mathbf{X}r$	delay	6	34	3 940	1,2

[c]The issue is used as an example in [17].

References

[1] Authén S, Bäckström O, Holmberg J, Porthin M, Tyrväinen T. NKS-361, Modelling of DIgital I&C, MODIG – Interim report 2015. NKS-R 361. NKS; 2016.

[2] IAEA. Approaches for overall instrumentation and control architectures of nuclear power plants. Nuclear Energy Series NP-T-2.1. International Atomic Energy Agency; 2018. http://www-pub.iaea.org/MTCD/Publications/PDF/PUB1821_web.pdf.

[3] MDEP. Common position on spurious actuation. Generic Common Position CP-DICWG-13. Multinational Design Evaluation Programme; 2018. https://www.oecd-nea.org/mdep/common-positions/cp-dicwg-13.pdf.

[4] MDEP. Common position on hazard identification and controls for digital instrumentation and control systems. Generic Common Position DICWG-10. Multinational Design Evaluation Programme; 2016. https://www.oecd-nea.org/mdep/common-positions/MDEP_GCP-DICWG-10_HazardIDandControl.pdf.

[5] Bel VBfE, CNSC, CSN, ISTec, KAERI, KINS, NSC, ONR, SSM, STUK. Licensing of safety critical software for nuclear regulators, common position of international nuclear regulators and authorised technical support organisation. Common position Revision 2018. 2018. http://www.onr.org.uk/software.pdf.

[6] IEC. Nuclear power plants – instrumentation and control important to safety – general requirements for systems. IEC Standard 61513:2011. International Electrotechnical Commission; 2011.

[7] IAEA. Technical challenges in the application and licensing of digital instrumentation and control systems in nuclear power plants. Nuclear Energy Series NP-T-1.13. International Atomic Energy Agency; 2015. http://www-pub.iaea.org/MTCD/Publications/PDF/P1695_web.pdf.

[8] Heimdahl M, Choi Y, Whalen W. Deviation analysis: a new use of model checking. Automated Softw Engineering 2005;12(3):321–47. https://doi.org/10.1007/s10515-005-2642-x.

[9] Joshi A, Heimdahl MPE. Model-based safety analysis of Simulink models using SCADE design verifier. In: Winther R, Gran BA, Dahll G, editors. Computer Safety, Reliability, and Security Berlin, Heidelberg: Springer Berlin Heidelberg; 2005. p. 122–35. https://doi.org/10.1109/ICRE.1998.667803.

[10] Schneider F, Easterbrook SM, Callahan JR, Holzmann GJ. Validating requirements for fault tolerant systems using model checking. Proceedings of IEEE International Symposium on Requirements Engineering: RE '98. 1998. p. 4–13. https://doi.org/10.1109/ICRE.1998.667803.

[11] Zhang M, Liu Z, Morisset C, Ravn AP. Design and verification of fault-tolerant components. In: Butler M, Jones C, Romanovsky A, Troubitsyna E, editors. Methods, Models and Tools for Fault Tolerance Berlin, Heidelberg: Springer Berlin Heidelberg; 2009. p. 57–84. https://doi.org/10.1007/978-3-642-00867-2_4.

[12] Bozzano M, Cimatti A, Katoen J-P, Katsaros P, Mokos K, Nguyen VY, et al. Spacecraft early design validation using formal methods. Reliability Engineering & System Safety 2014;132:20–35. https://doi.org/10.1016/j.ress.2014.07.003.

[13] Bernardeschi C, Fantechi A, Gnesi S. Model checking fault tolerant systems. Software Testing, Verification and Reliability 2002;12(4):251–75. https://doi.org/10.1002/stvr.258.

[14] Sharvia S, Papadopoulos Y. Integrating model checking with HiP-HOPS in model-based safety analysis. Reliability Engineering & System Safety 2015;135:64–80. https://doi.org/10.1016/j.ress.2014.10.025.

[15] Clarke E, Grumber O, Peled D. Model checking. 2 Cambridge, Massachusetts, US: MIT press978-0-262-03270-4; 2001.

[16] Pakonen A, Tahvonen T, Hartikainen M, Pihlanko M. Practical applications of model checking in the Finnish nuclear industry. 10th International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies (NPIC & HMIT). 2017. p. 1342–52. http://www.vtt.fi/inf/julkaisut/muut/2017/OA-Practical_applications_of_model_checking.pdf.

[17] Pakonen A, Björkman K. Model checking as a protective method against spurious actuation of industrial control systems. 27th European Safety and Reliability Conference (ESREL). 2017. p. 3189–96. https://cris.vtt.fi/files/23031295/Pakonen_ESREL2017_Accepted_Manuscript.pdf.

[18] Buzhinsky I, Pakonen A. Model-checking detailed fault-tolerant nuclear power plant safety functions. IEEE Access 2019;7:162139–56. https://doi.org/10.1109/ACCESS.2019.2951938.

[19] Cimatti A, Clarke E, Giunchiglia E, Giunchiglia F, Pistore M, Roveri M, et al. NuSMV 2: An open source tool for symbolic model checking. In: Brinksma E, Larsen KG, editors. Computer Aided Verification. Berlin, Heidelberg: Springer Berlin Heidelberg978-3-540-45657-5; 2002. p. 359–64.

[20] Eisner C, Fisman D. A practical introduction to PSL. Springer US; 2006. https://doi.org/10.1007/978-0-387-36123-9.

[21] Burch J, Clarke E, McMillan K, Dill D, Hwang L. Symbolic model checking: 10^{20}states and beyond. Information and Computation 1992;98(2):142–70. https://doi.org/10.1016/0890-5401(92)90017-A.

[22] Clarke E, Biere A, Raimi R, Zhu Y. Bounded model checking using satisfiability solving. Formal Methods in System Design 2001;19:7–34. https://doi.org/10.1023/A:1011276507260.

[23] Eén N, Sörensson N. Temporal induction by incremental SAT solving. Electron Notes Theor Comput Sci 2003;89(4):543–60. https://doi.org/10.1016/S1571-

0661(05)82542-3.
[24] Behrmann G, David A, Larsen KG. A Tutorial on Uppaal. Berlin, Heidelberg: Springer Berlin Heidelberg978-3-540-30080-9; 2004. p. 200–36.
[25] Cimatti A, Griggio A, Mover S, Tonetta S. HyComp: An SMT-based model checker for hybrid systems. In: Baier C, Tinelli C, editors. Tools and Algorithms for the Construction and Analysis of Systems. Berlin, Heidelberg: Springer Berlin Heidelberg978-3-662-46681-0; 2015. p. 52–67.
[26] Lamport L. Proving the correctness of multiprocess programs. IEEE Trans Software Eng 1977;SE-3(2):125–43. https://doi.org/10.1109/TSE.1977.229904.
[27] Benedetti M, Cimatti A. Bounded model checking for past LTL. In: Garavel H, Hatcliff J, editors. Tools and Algorithms for the Construction and Analysis of Systems. Berlin, Heidelberg: Springer Berlin Heidelberg978-3-540-36577-8; 2003. p. 18–33.
[28] IEC. Property specification language. IEC Standard 62531:2012. International Electrotechnical Commission; 2012.
[29] Pakonen A, Pang C, Buzhinsky I, Vyatkin V. User-friendly formal specification languages - conclusions drawn from industrial experience on model checking. 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA). 2016. p. 1–8. https://doi.org/10.1109/ETFA.2016.7733717.
[30] Pradella M, San Pietro P, Spoletini P, Morzenti A. Practical model checking of LTL with past. International Workshop on Automated Technology for Verification and Analysis (ATVA03). 2003.
[31] Dwyer M, Avrunin G, Corbett J. Patterns in property specifications for finite-state verification. Proceedings of the 21st International Conference on Software Engineering. New York, NY, USA: Association for Computing Machinery1581130740; 1999. p. 411–20. https://doi.org/10.1145/302405.302672.
[32] Preuße S, Lapp H, Hanisch H. Closed-loop system modeling, validation, and verification. Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012). 2012. p. 1–8. https://doi.org/10.1109/ETFA.2012.6489679.
[33] Ramos AL, Ferreira JV, Barceló J. Model-based systems engineering: an emerging approach for modern systems. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 2012;42(1):101–11. https://doi.org/10.1109/TSMCC.2011.2106495.
[34] Buzhinsky I, Pakonen A, Vyatkin V. Explicit-state and symbolic model checking of nuclear I&C systems: A comparison. IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society. 2017. p. 5439–46. https://doi.org/10.1109/IECON.2017.8216942.
[35] STUK. Safety design of a nuclear power plant. YVL Guide B.1. Radiation and Nuclear Safety Authority; 2013. https://www.stuklex.fi/en/ohje/YVLB-1.
[36] Areva NP. U.S. EPR protection system. Technical Report ANP-10309NP. Areva NP; 2012. https://www.nrc.gov/docs/ML1216/ML121660317.html.
[37] Areva NP. U.S. EPR application documents. Final Safety Analysis Report. Areva NP; 2013. https://www.nrc.gov/reactors/new-reactors/design-cert/epr/reports.html.
[38] Rolls-Royce. Spinline TM, a Rolls-Royce modular I&C digital platform dedicated to nuclear safety. Technical Sheet. Rolls-Royce; 2013.
[39] Lahtinen J, Valkonen J, Björkman K, Frits J, Niemelä I, Heljanko K. Model checking of safety-critical software in the nuclear engineering domain. Reliability Engineering & System Safety 2012;105:104–13. https://doi.org/10.1016/j.ress.2012.03.021.
[40] Pakonen A, Valkonen J, Matinaho S, Hartikainen M. Model checking for licensing support in the Finnish nuclear industry. International Symposium on Future I&C for Nuclear Power Plants (ISOFIC). 2014.
[41] IEC. Programmable controllers – part 3: Programming languages. IEC Standard 61131-3:2013. International Electrotechnical Commission; 2013.
[42] Buzhinsky I, Pakonen A, Vyatkin V. Synthesis-aided reliability assurance of basic block models for model checking purposes. 2018 IEEE 27th International Symposium on Industrial Electronics (ISIE). 2018. p. 669–74. https://doi.org/10.1109/ISIE.2018.8433793.
[43] Pakonen A, Buzhinsky I, Vyatkin V. Counterexample visualization and explanation for function block diagrams. 2018 IEEE 16th International Conference on Industrial Informatics (INDIN). 2018. p. 747–53. https://doi.org/10.1109/INDIN.2018.8472025.
[44] Pakonen A, Mätäsniemi T, Lahtinen J, Karhela T. A toolset for model checking of PLC software. 2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA). 2013. p. 1–6. https://doi.org/10.1109/ETFA.2013.6648065.
[45] Siemens. TELEPERM XS: A digital reactor protection system. Technical Report EMF-2110(NP)(A). Siemens; 2012.
[46] Helminen A, Pakonen A. Potential applications of model checking in probabilistic risk assessments. VTT Technical Report VTT-R-00017-20. VTT Technical Research Centre of Finland Ltd.; 2020. https://cris.vtt.fi/files/27299692/VTT_R_00017_20.pdf.
[47] Lahtinen J. Hardware failure modelling methodology for model checking. VTT Technical Report VTT-R-00213-14. VTT Technical Research Centre of Finland Ltd.; 2014. https://www.vtt.fi/inf/julkaisut/muut/2014/VTT-R-00213-14.pdf.
[48] Garcia I. Spurious actuations in digital instrumentation and control systems - evaluation framework. 10th International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies (NPIC & HMIT). 2017. p. 275–82.
[49] Vaurio JK. Importance measures in risk-informed decision making: ranking, optimisation and configuration control. Reliability Engineering & System Safety 2011;96(11):1426–36. https://doi.org/10.1016/j.ress.2011.06.012.
[50] Jockenhövel-Barttfeld M, Taurines A, Hessler C. Quantification of application software failures of digital I&C in probabilistic safety analyses. Proceedings of 13th International Conference on Probabilistic Safety Assessment and Management (PSAM13). 2016.
[51] Martorell S, Martorell P, Martón I, Sánchez A, Carlos S. An approach to address probabilistic assumptions on the availability of safety systems for deterministic safety analysis. Reliability Engineering & System Safety 2017;160:136–50. https://doi.org/10.1016/j.ress.2016.12.009.
[52] Ovatman T, Aral A, Polat D, Ünver A. An overview of model checking practices on verification of PLC software. Softw & Systems Modeling 2016;15(4):937–60. https://doi.org/10.1007/s10270-014-0448-7.
[53] Pavlovic O, Ehrich H. Model checking PLC software written in Function Block Diagram. 2010 Third International Conference on Software Testing, Verification and Validation. 2010. p. 439–48. https://doi.org/10.1109/ICST.2010.10.
[54] Fernández Adiego B, Darvas D, Viñuela EB, Tournier J, Bliudze S, Blech JO, et al. Applying model checking to industrial-sized PLC programs. IEEE Trans Ind Inf 2015;11(6):1400–10. https://doi.org/10.1109/TII.2015.2489184.
[55] Cheng Pang, Vyatkin V. Automatic model generation of IEC 61499 function block using net condition/event systems. 2008 6th IEEE International Conference on Industrial Informatics. 2008. p. 1133–8. https://doi.org/10.1109/INDIN.2008.4618273.
[56] Déharbe D, Shankar S, Clarke EM. Model checking VHDL with CV. In: Gopalakrishnan G, Windley P, editors. Formal Methods in Computer-Aided Design. Berlin, Heidelberg: Springer Berlin Heidelberg; 1998. p. 508–14.
[57] Campos JC, Machado J. Pattern-based analysis of automated production systems. IFAC Proceedings Volumes 2009;42(4):972–7. https://doi.org/10.3182/20090603-3-RU-2001.0425. 13th IFAC Symposium on Information Control Problems in Manufacturing.
[58] Autili M, Inverardi P, Pelliccione P. Graphical scenarios for specifying temporal properties: an automated approach. Automated Softw Engineering 2017;14(3):293–340. https://doi.org/10.1007/s10515-007-0012-6.
[59] Ljungkrantz O, Åkesson K, Fabian M, Ebrahimi A. An empirical study of control logic specifications for programmable logic controllers. Empirical Softw Engineering 2014;19(3):655–77. https://doi.org/10.1007/s10664-012-9232-x.
[60] Yoo J, Cha S, Jee E. Verification of PLC programs written in FBD with VIS. Empirical Softw Engineering 2009;41(1):79–90.
[61] Németh E, Bartha T. Formal verification of safety functions by reinterpretation of functional block based specifications. In: Cofer D, Fantechi A, editors. Formal Methods for Industrial Critical Systems. Berlin, Heidelberg: Springer Berlin Heidelberg; 2009. p. 199–214.
[62] Cavada R, Cimatti A, Dorigatti M, Griggio A, Mariotti A, Micheli A, et al. The nuXmv symbolic model checker. In: Biere A, Bloem R, editors. Computer Aided Verification. Cham: Springer International Publishing; 2014. p. 334–42.
[63] IEC. Failure modes and effects analysis (FMEA and FMECA). IEC Standard 60812. International Electrotechnical Commission; 2018.
[64] IEC. Fault tree analysis (FTA). IEC Standard 61205:2006. International Electrotechnical Commission; 2006.
[65] Bozzano M, Cimatti A, Lisagor O, Mattarei C, Mover S, Roveri M, et al. Safety assessment of AltaRica models via symbolic model checking. Sci Comput Program 2015;98:464–83. https://doi.org/10.1016/j.scico.2014.06.003.
[66] Grunske L, Winter K, Yatapanage N, Zafar S, Lindsay PA. Experience with fault injection experiments for FMEA. Software: Practice and Experience 2011;41(11):1233–58. https://doi.org/10.1002/spe.1039.
[67] Molnár V, Majzik I. Model checking-based software-FMEA: assessment of fault tolerance and error detection mechanisms. Periodica Polytechnica Electrical Engineering and Computer Science 2017;61(2):132–50. https://doi.org/10.3311/PPee.9755.
[68] Aljazzar H, Fischer M, Grunske L, Kuntz M, Leitner-Fischer F, Leue S. Safety analysis of an airbag system using probabilistic FMEA and probabilistic counterexamples. 2009 Sixth International Conference on the Quantitative Evaluation of Systems. 2009. p. 299–308. https://doi.org/10.1109/QEST.2009.8.