Oops! Examples of I&C design issues detected with model checking

Pakonen, Antti

Published in:
International Symposium on Future I&C for Nuclear Power Plants, ISOFIC 2021

Published: 15/11/2021

Document Version
Publisher's final version

Oops! Examples of I&C design issues detected with model checking

Antti PAKONEN

VTT Technical Research Centre of Finland Ltd., P.O. Box 1000, FI-02044 VTT, Espoo, Finland (antti.pakonen@vtt.fi)

***Abstract*—Since 2008, VTT has used a formal verification method called model checking to verify instrumentation and control (I&C) application logic design in practical projects in the Finnish nuclear industry. In this paper, we present seven examples of the 66 confirmed design issues that we have detected. We then discuss potential causes why only formal verification revealed the otherwise hidden issues. We hope the examples will be useful in case studies related to verification and quality assurance of I&C.**

Keywords—instrumentation and control; model checking; verification and validation, function block diagram

I. INTRODUCTION

Model checking [1] is a formal verification method, where a software tool (called a model checker) is used to prove if a model of a system satisfies stated formal properties. In Finland, the method has been applied for over a decade to verify instrumentation and control (I&C) application logic design in two new-build projects (Olkiluoto 3 EPR, and the functional design for the Hanhikivi-1 AES-2006) and an I&C renewal project (for the two Loviisa VVER-440 units). To date, in all these projects combined, VTT has detected 66 confirmed design issues of varying probability and safety relevance. We discuss the tools, the work process, and the limitations in [2], and the customer projects in more detail in [3].

In this paper, we reveal seven examples of the design issues we have detected. We have modified and simplified the designs in order to mask their origin, and focus on the parts that caused the issues. Similar examples can also be found in our previous publications [2][3][4][5][6][7][8]. We have detected all of the issues using the free, open source model checker NuSMV [9].

II. EXAMPLES OF DESIGN ISSUES

We recorded the analysis times below on an Intel Core i7-6600U CPU with a clock rate of 2.6 GHz.

A. Example 1: Exact timing of input signals produces unwanted response

The intended functionality for the "design pattern" in Fig 1. is that an active CRITERION leads to a minute long pulse of ACT, but if CRITERION is reset, so is ACT. However, if ACT only flashes on very quickly, and then becomes permanently active exactly one minute later, the Pulse element will not react to the latter rising edge, and the function is not actuated (save for the very short pulse at the start). The "design pattern" was used in several functions of the verified system.

In just 0.04 seconds, NuSMV generates a counter-example for the LTL [1] property: $\mathbf{G}\ (\neg\text{CRITERION} \wedge \mathbf{X}\ \text{CRITERION} \rightarrow \mathbf{X}\ \text{ACT})$.

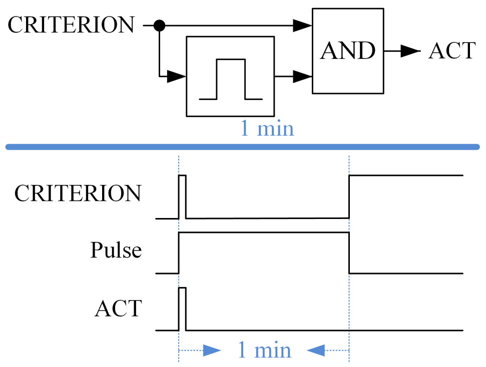


Fig 1. The logic and the counterexample (as a timing diagram) for example 1

B. Example 2: Two connected safety functions permanently frozen

The intended functionality for the two safety functions A and B, as simplified in Fig 2., is that if A is set, and then (within 5 seconds) B is set, then function A will be reset. However, in the (counterintuitive, in its original context) scenario where B is not active in the 5-second time window after A's actuation, the logic ends up in state where A is permanently set, and B permanently reset. (To reactivate the Pulse and set ACT_B to TRUE, ACT_A would first need to be reset to FALSE, which is not possible as long as ACT_B remains FALSE.)

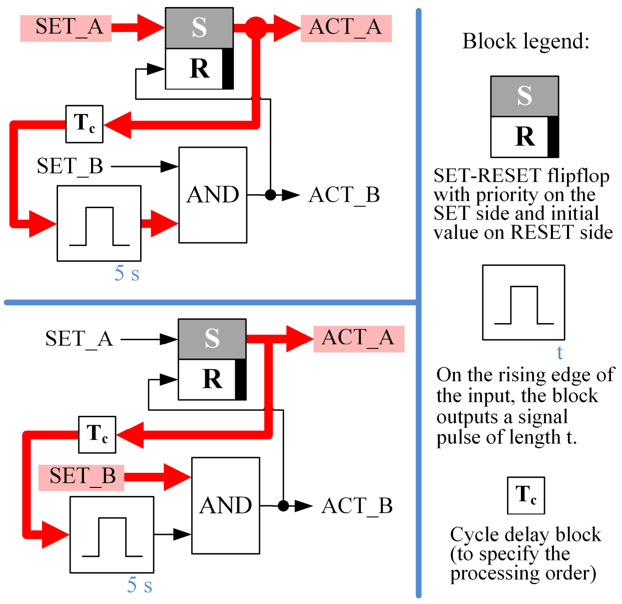

SET_A
S
R
ACT_A
T_c
SET_B
AND
ACT_B
5 s
SET_A
S
R
ACT_A
T_c
SET_B
AND
ACT_B
5 s
Block legend:
S
R
SET-RESET flipflop with priority on the SET side and initial value on RESET side
t
On the rising edge of the input, the block outputs a signal pulse of length t.
T_c
Cycle delay block (to specify the processing order)

Fig 2. The logic and the counterexample for example 2

The originally verified model consisted of 17 function blocks. NuSMV takes 0.38 seconds to produce the counterexample for the LTL property: $\mathbf{G}\ (\text{ACT\_A} \rightarrow \mathbf{X}\ \mathbf{X}\ \mathbf{X}\ \mathbf{X}\ ((\neg\text{SET\_A} \wedge \text{SET\_B}) \rightarrow \neg\text{ACT\_A}))$.

C. *Example 3: Instantaneous process variable drop leaves the function unactuated*

The intended functionality for the logic in Fig 3. is to initiate a function (ACT) when a certain process variable (VAR) drops below 10, and then stop the order when the variable proceeds to drop below 8 (actual values masked). However, in the physically very unlikely scenario where the variable would drop instantaneously (within a processing cycle) from above 10 to below 8, the function is never actuated.

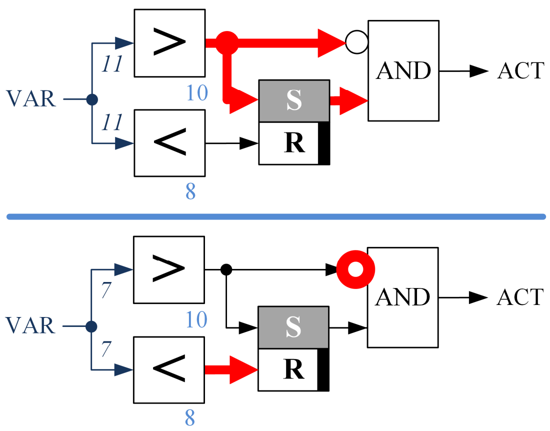


Fig 3. The logic and the counterexample for example 3

The originally verified model consisted of 60 function blocks. NuSMV takes 8.6 seconds to generate the counterexample for the LTL property: **G** ((VAR > 10) ∧ **X** (VAR < 10) → **X** ACT).

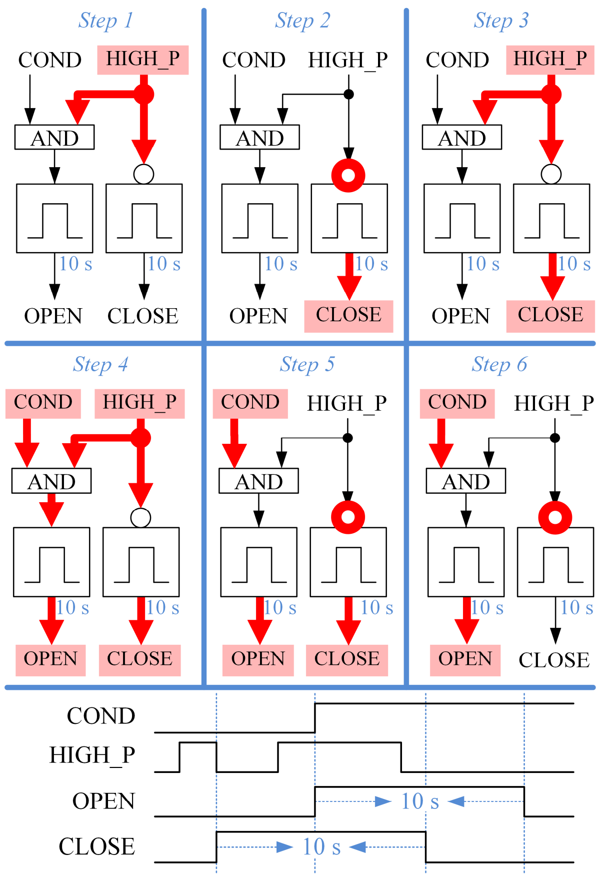


Fig 4. The logic and the counterexample for example 4

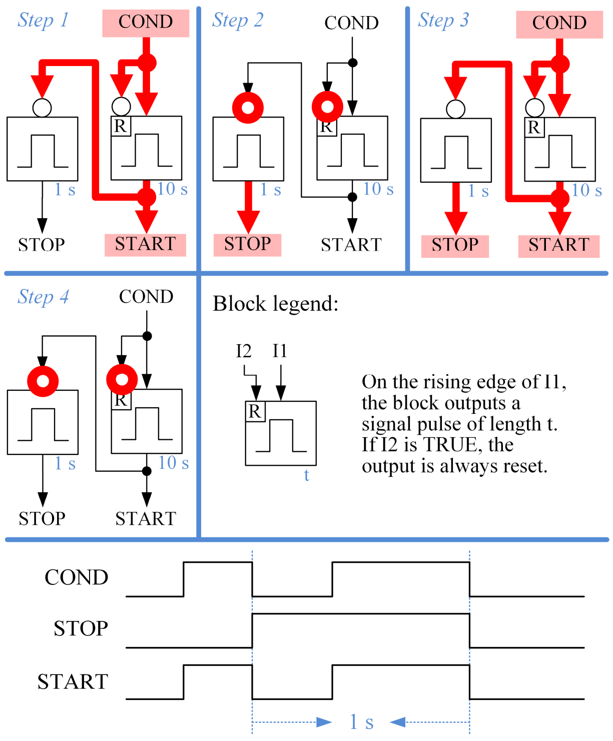
D. *Example 4: Valve left in incorrect position after fluctuating inputs*

The intended functionality for the logic in Fig 4. is to open a certain valve if the pressure is high (HIGH_P) and the process conditions (COND) otherwise allow for it. When the pressure returns to normal (¬HIGH_P), the logic shall again close the valve. However, Fig 4. shows a scenario where (1) HIGH_P is first active for a short time while COND is not, (2) HIGH_P is then again active while COND is also TRUE, and (3) HIGH_P then again resets, before 10 seconds have passed since the last time it was FALSE. In this scenario, the closing-side Pulse block does not respond to the second falling edge of HIGH_P (as it still processing the earlier 10-second pulse), and the logic leaves the valve open.

NuSMV takes 0.17 seconds to generate the counterexample for the LTL property: **G** ((COND ∧ HIGH_P) ∧ **X** (COND ∧ ¬HIGH_P) → **X** (CLOSE ∧ ¬OPEN)).

E. *Example 5: Contradictory commands to an actuator (on fluctuating inputs)*

The intended functionality for the logic in Fig 6. is that when the process condition (COND) changes to TRUE, the START output is active for 10 seconds. When the process condition then resets, STOP is active for one second. However, Fig 5. shows a scenario, where (1) COND is first set, (2) COND is then reset, (3) soon afterwards, COND is again set, and (4) exactly one second after step 2, COND is again reset. In this scenario, the left-side (STOP) Pulse does not react to the rising edge of its input on step 4, as it is just at the end of the previous pulse. The actuator receives contradictory commands, and no separate STOP at the end.



Step 1
Step 2
Step 3
Step 4
COND
R
1 s
10 s
STOP
START
Block legend:
I2
I1
t
On the rising edge of I1, the block outputs a signal pulse of length t. If I2 is TRUE, the output is always reset.

Fig 5. The logic and the counterexample for example 5

The originally verified model consisted of 156 function blocks. NuSMV takes 5.1 seconds to generate the counterexample for the LTL property: **G** (COND ∧ **X** ¬COND → **X** STOP).

F. *Example 6: Several channels inhibited simultaneously*

The intended functionality for the logic in Fig 6. is that when an operator inhibits a channel by turning a switch (SWT), a signal is sent to the other channels, preventing them from being inhibited at the same time. However, if many switches are turned at the exact same time (which the operators are not supposed to do), each channel is inhibited, as the signals from the other channels are received on the next cycle.

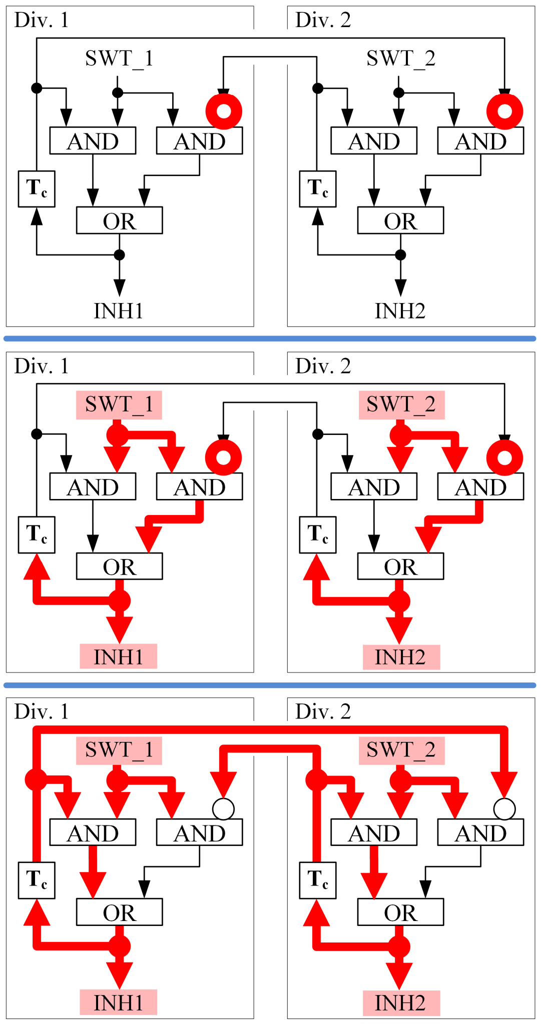


Fig 6. The logic and the counterexample for example 6

The originally verified model consisted of 118 function blocks. NuSMV takes 1.9 seconds to generate the counterexample for the LTL property: **G** ¬(INH_1 ∧ INH_2).

G. *Example 7: Contradictory commands to an actuator (on contradictory inputs)*

The intended functionality for the logic in Fig 7. is that an actuator is started when the measurement is below 8 units, and stopped when the measurement is above 10 units. The logic performs a type of majority vote by starting based on the second-lowest measurement, and stopping based on the second-highest measurement. However, on the kind of highly unlikely measurement data (perhaps attributable to multiple equipment failure) shown in Fig 7., the controlled actuator receives contradictory commands.

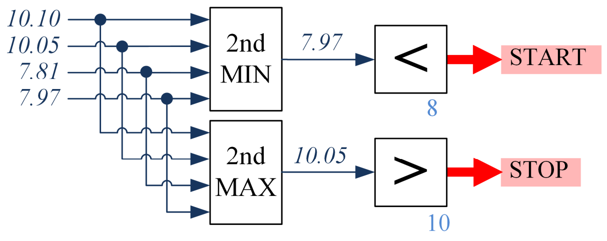


Fig 7. The logic and the counterexample for example 7

The originally verified model consisted of 19 function blocks. NuSMV takes 14 seconds to generate the counterexample for the LTL property: **G** ¬(START ∧ STOP).

III. Discussion

Most of the designs we verified in the practical projects had already been subjected to verification and validation (V&V) based on more conventional methods (e.g., testing). Therefore, the detected issues often have some features that make them hard to find without formal verification. A potential cause is that the scenario needed to reveal the issue is highly unlikely and/or counterintuitive. In Table 1, we list common features of the issues and the scenarios (counterexamples) that revealed them.

Table 1 Features or causes of the issues

Feature / cause	Issues	Share	Examples
Spurious actuation	22	33%	4,[2][8]
Exact timing	21	32%	1,5,6,[2][6][7][8]
Human user actions	19	29%	6,[2][3][6][7][8]
Uncharacteristic input	14	21%	2,3,7
Signal validity logic	6	9%	[2]
Frozen (deadlock)	5	8%	2

33% of the issues feature spurious actuation, which is otherwise hard to analyze [2]. In testing, for example, it is much easier to address the intended functionality. 32% of the issues feature exact (millisecond-level) timing, which might not be possible to reproduce in a test field or plant simulator [7]. In 29% of the scenarios, operators or maintenance personnel perform ill-advised or ill-timed actions.

By “uncharacteristic input”, we refer to scenarios where the inputs of the model that represent process measurements have values that are not likely (or even possible) when we consider the actual physical and chemical processes of the controlled plant. More exactly, we mean scenarios where (1) the process variables show a combination of values that is physically unlikely, or (2) a process variable changes its value faster than is physically likely. We do not model the controlled plant, so the model inputs can have arbitrary values at each counterexample step. (Sometimes, the analyst is able to alter the scenario first produced by the model checker into a more likely one, but here we refer to scenarios where such modification was not possible.) Such input data are one of the causes in 21% of the issues.

In 9%, the issue is at least partially caused by signal validity processing, a common feature [2] in fault-tolerant nuclear I&C logics (see [2] for an exemplar issue). Finally, in five cases, the logic permanently froze to some output state.

Another potential cause that makes the issues hard to detect without formal verification is the complexity of the logic design. Function block diagrams make it relatively easy to understand the "flow" of control from input to output values. Elements that interfere with this flow make it harder to figure the logic out, leaving room for the designer to make a choice that enables unintended behavior (and making it harder for a reviewer to detect the issue). In Table 2, we show the prevalence of such elements in the logics that then contained design issues.

Table 2 Design elements in the logics behind the issues

Element in logic	Issues	Share	Examples
Memory	43	65%	2,3,[2][3][5][6][7][8]
Delay	43	65%	1,2,4,5,[4]
Feedback loop	20	30%	2,6,[4][7]

65% of the logics contained a memory (bistable latch, flip-flop) block. 65% contained a delay element (with a configurable delay time). 30% contained a feedback loop. (A cycle delay block used to specify the processing order in a feedback loop also acts as a type of memory/delay.)

We have significantly simplified the original logics for this paper, and included in the examples only the minimum number of blocks needed to recreate the issues. As stated above, the original logics could contain up to 156 function blocks, and the problematic elements were not necessarily as close to one another on the diagram. What can look like an obvious mistake in this paper was not necessarily as apparent in the original context, at least for manual review.

IV. Conclusions

Designing fault-free I&C software is hard. What is also hard is to invent interesting, credible application software design errors. Observations of nuclear safety I&C systems failing during operation are actually so rare, that it is making software probabilistic safety assessment (PSA) hard [10]. The contribution of this paper is a set of real-world examples of actual design issues. We hope that the examples prove useful in different case studies to support research on testing, test automation, simulation, model checking, theorem proving, run-time verification, and other V&V activities.

In addition, we of course hope that our examples help prove the point that formal verification methods should already be in much wider use.

Acknowledgement

Our research on nuclear I&C model checking has been funded by Finnish Research Programme on Nuclear Power Plant Safety 2018–2022 (SAFIR 2022). 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. The NuSMV model checker has been developed by Fondazione Bruno Kessler (FBK).

References

[1] E. Clarke, O. Grumberg, D. Peled, Model checking, 2nd Ed., MIT Press, 2001.

[2] A. Pakonen, I. Buzhinsky, K. Björkman, Model checking reveals design issues leading to spurious actuation of nuclear instrumentation and control systems, *Reliability Engineering & System Safety*, Vol. 205, 107237, 2021.

[3] A. Pakonen, Model-checking of I&C logics — insights from over a decade of projects in Finland, Proceedings of the 12th ANS International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies (NPIC & HMIT 2021), Providence, RI, USA, pp. 792-801, 2021.

[4] A. Pakonen, Model-checking infinite-state nuclear safety I&C systems with nuXmv, Proceedings of the 19th IEEE International Conference on Industrial Informatics (INDIN 2021). Palma de Mallorca, Spain, 2021.

[5] A. Pakonen, P. Biswas, N. Papakonstantinou, Transformation of non-standard nuclear I&C logic drawings to formal verification models, Proceedings of the 46th Annual Conference of the IEEE Industrial Electronics Society (IECON 2020), Singapore, pp. 697-704, 2020.

[6] A. Pakonen, I., Buzhinsky, V. Vyatkin, Counterexample visualization and explanation for function block diagrams, Proceedings of the 16th IEEE International Conference on Industrial Informatics (INDIN 2018), Porto, Portugal, pp. 747-753, 2018.

[7] A. Pakonen, T., Tahvonen, M., Hartikainen, M., Pihlanko, Practical applications of model checking in the Finnish nuclear industry, Proceedings of the 10th ANS International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies (NPIC & HMIT 2017), San Francisco, CA, USA, pp. 1342-1352, 2017.

[8] A. Pakonen, K. Björkman, Model checking as a protective method against spurious actuation of industrial control systems, Proceedings of the 27th European Safety and Reliability Conference (ESREL 2017), Portoroz, Slovenia, 2017.

[9] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella, NuSMV 2: An OpenSource Tool for Symbolic Model Checking, Proceedings of the International Conference on Computer-Aided Verification (CAV 2002). Copenhagen, Denmark, pp. 359-364, 2002.

[10] M. Jockenhövel-Barttfeld, A. Taurines, C. Hessler, Quantification of application software failures of digital I&C in probabilistic safety analyses, Proceedings of the 13th International Conference on Probabilistic Safety Assessment and Management (PSAM13), 2016.