The 13th IEEE International Conference on Dependable Systems, Services and Technologies, DESSERT'2023
13-15 October, 2023, Athens, Greece

Application of Formal Verification Methods in a Safety-Oriented Software Development Life Cycle

Oleg Odarushchenko
Radics LLC
Kropyvnytskiy,Ukraine
oleg.odarushchenko@radics-ua.com

Oleksiy Striuk
Radics LLC
Kropyvnytskiy,Ukraine
oleksii.striuk@radics-ua.com

Viacheslav Shamanskyi
Radics LLC
Kropyvnytskiy,Ukraine
viacheslav.shamanskyi@radics-ua.com

Oleksandr Letychevskyi
V.M. Glushkov Institute of Cybernetics of the NAS of Ukraine
Kyiv, Ukraine
oleksandr.letychevskyi@litsoft.com.ua

Aleksandr Ivasiuk
International Scientific & Technical University
Kyiv, Ukraine
o.ivasiuk@istu.edu.ua

Elena Odarushchenko
Poltava State Agrarian University
Poltava, Ukraine
elena.odarushchenko@gmail.com

***Abstract*— This article delves into the growing significance of ensuring reliability and functional safety in hardware and embedded software of programmable controllers amidst rapid technological advancement. With the rise of FPGA-based digital Instrumentation and Control Systems (ICS), the need for dependable solutions has led to the development of the RadICS FSC (Functional Safety Controller) Platform by LLC RPC Radiy. While programmable controllers are pivotal across industries, vulnerabilities in their embedded software can lead to dire consequences for safety, economy, and the environment, particularly in safety-critical applications. Given this backdrop, testing embedded software becomes integral to ensuring reliability and safety. Tailored testing approaches are essential due to the unique characteristics of hardware, software, and specific controller applications. These approaches must encompass functional aspects as well as potential vulnerabilities exploitable by malicious actors.**

International standards like IEC 61508, IEC 61513, and IEC 60880 outline regulatory frameworks for testing, verification, validation, and safety in embedded software, guiding its development and implementation within the system safety lifecycle. This article focuses on applying formal verification methods to embedded software within the context of the safety lifecycle for programmable controllers used in safety-related applications. Through this study, valuable insights are garnered regarding the efficacy of formal verification techniques in bolstering the reliability and safety of programmable controllers' embedded software. The results shed light on the benefits and challenges associated with integrating formal verification within the safety lifecycle, paving the way for enhanced safety and functionality in critical systems.

Keywords—formal verification, reliability, safety, safety lifecycle

I. INTRODUCTION

In the context of rapid technological advancement and the increasingly widespread utilization of programmable controllers across various industrial sectors, the issue of ensuring the reliability and security of controller's hardware and embedded software is gaining critical significance. Using FPGA-based digital Instrumentation and Control Systems (ICS) is a possible way to increase the dependability and diversity of newly developed solutions. However, changing a platform from microprocessors to FPGA requires adopting FPGA-specific tools and design techniques in the development of application-specific software, such an adaptation is not always a straightforward task. LLC RPC Radiy has developed FPGA-based industrial controllers - RadICS FSC (Functional Safety Controller) Platform [1]. RadICS FSC Platform preserves a traditional approach to developing application-specific embedded software. Programmable controllers play a pivotal role in managing diverse systems, such as production lines, transportation networks, medical equipment, and power installations. Inconsistencies, defects, errors, or vulnerabilities in the embedded software of such systems can have severe repercussions for safety, economy, and the environment. Therefore, for these applications, embedded software is referred to as safety-critical software.

In this context, testing embedded software of programmable controllers becomes an integral part of the process of ensuring high reliability and safety of these systems. The characteristics of hardware and software, as well as the specific applications of programmable controllers, necessitate the development and implementation of specialized testing approaches. These approaches must take into account both the functional aspects of software operation and potential vulnerabilities that malicious actors could exploit to disrupt system functionality or cause harm.

To achieve the required level of quality and security for embedded software of programmable controllers, international and governmental bodies develop pertinent regulatory documents and standards. These standards provide guidelines and recommendations for testing, verification, and validation processes of embedded software. Additionally, they establish requirements for its reliability and resistance to attacks.

The implementation of these recommendations is feasible within the prescribed stages of the system security lifecycle. Examples of standards that recommend the implementation of the system security lifecycle include the following [2-8]:

- IEC 61508: Functional safety of electrical/electronic/programmable electronic safety related systems;
- IEC 61513: Nuclear power plants – Instrumentation and control for systems important to safety – General requirements for systems programmable electronic (E/E/PE) safety-related systems;

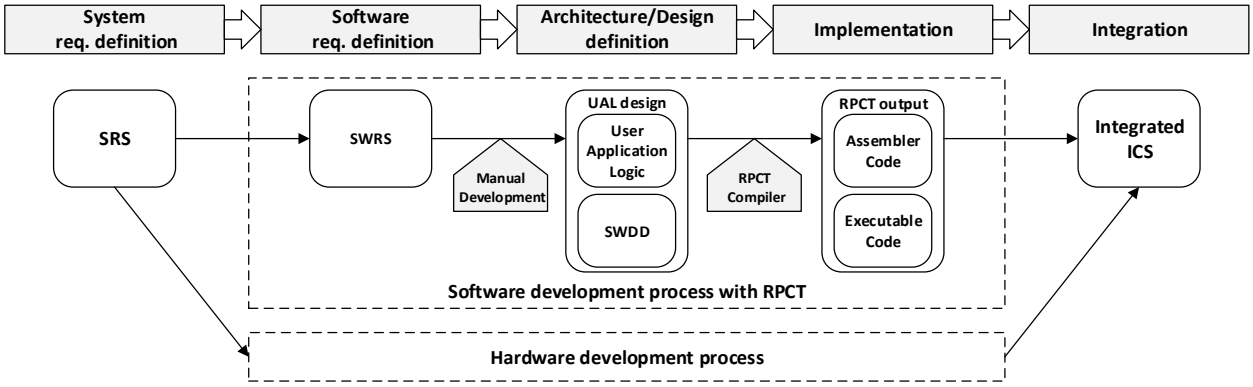


Fig. 1. A typical system development process for FSC-based ICS

- IEC 60880: Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category A functions.

The aim of this article is to present the results of applying formal verification methods to the embedded software of programmable controllers for safety-related applications within the context of the applied safety lifecycle.

II. State-of-the-Art

The integration of formal verification methods into the development process of embedded software for programmable controllers in safety-critical applications represents a cutting-edge approach to enhancing the reliability and safety of such systems within the broader framework of the applied safety lifecycle. This section provides an overview of the current state of the art in this field.

In recent years, there has been a significant advancement in the development of formal verification tools and techniques. These tools, including model checking, theorem proving, and abstract interpretation, have matured and become more accessible. They offer the ability to rigorously analyze software designs and detect critical issues such as data races, buffer overflows, and deadlocks, which are of paramount concern in safety-related systems [9-12].

The domain of safety-critical systems adheres to stringent standards and regulations, such as ISO 26262 for automotive systems or IEC 61508, for general industrial applications. Formal verification is increasingly recognized and accepted as a means of achieving compliance with these standards. This recognition has paved the way for the adoption of formal methods in safety-related software development.

Various industries, including automotive, aerospace, and healthcare, have started to incorporate formal verification methods into their safety-critical software development processes. For instance, in the automotive sector, companies are using formal methods to verify control software for autonomous vehicles [14], while in healthcare, formal verification plays a vital role in the development of medical devices and health information systems [15].

Despite the progress made in applying formal verification to safety-related software, there are still challenges and research gaps that need to be addressed. These challenges include scalability issues, the need for standardized safety-critical libraries, and the integration of formal methods into existing development processes. Researchers and practitioners are actively working on these challenges to further advance the field [16].

In summary, the current state of the art in applying formal verification methods to the embedded software of programmable controllers for safety-related applications within the context of the applied safety lifecycle is characterized by significant progress, growing industrial adoption, and a commitment to addressing existing challenges. This approach holds promise for enhancing the safety and reliability of critical systems in various industries and is poised to play a crucial role in the future of safety-critical software development.

III. Existing Life Cycle of Testing User Applications Logic

A. Life cycle of a critical safety level software development

Figure 1 shows a typical system development process for an FSC-based ICS [1,17-21]. User Application Logic – application–specific embedded software developed with a Radiy-produced Integrated Design Environment (IDE) – RPCT (Radiy Platform Configuration Tool). RPCT allows designers to define a system configuration, to design a project-specific UAL in the graphically based design environment, to test the developed UAL, to compile and to upload the developed UAL into FSC. The software development chain consists of typical activities for PLC software development and does not require FPGA-specific knowledge. The input for software development is a System Requirements Specification (SRS). Software Requirements Specification (SWRS) is derived from SRS and specifies software-specific requirements. The UAL design is developed to satisfy the requirements of SWRS and consists of the UAL as a project in RPCT followed by Software Detailed Design – a document that describes the design with necessary details. Compilation of the UAL project produces an output that will be directly uploaded into FSC. Mainly the RPCT output consists of the following instructions for FSC:

- Reading data from the dedicated memory cells and putting it into a particular Application Functional Block (AFB).
- Execution of the AFB function.
- Writing results of the AFB function execution to the dedicated memory cells.

By the data structure, the RPCT output is a hex-byte code of Radiy's proprietary assembler for FSC.

B. Verification activities

Figure 2 shows the UAL life-cycle that was adopted from IEC 60880.

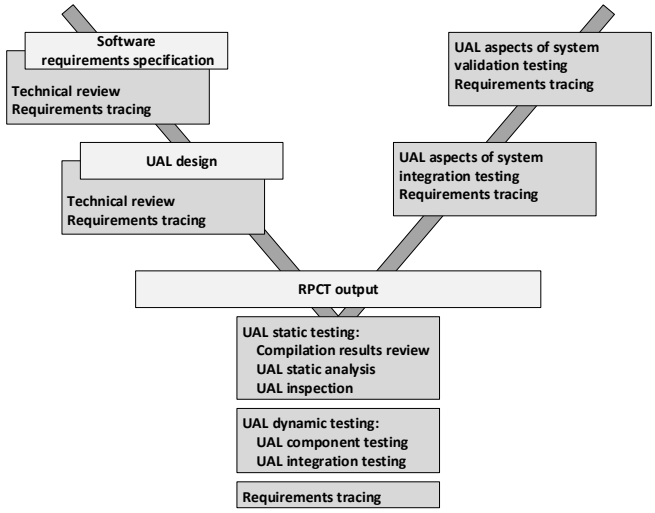


Fig. 2 Adaptation of the UAL Life-Cycle

Table 1 lists software tools supported verification activities through the UAL life-cycle. RPCT Outputs Verification Tool (ROVT) is a tool that was independently developed to facilitate the static testing of RPCT outputs. TraceTrend is Radiy's proprietary tool to perform requirements tracing. Although, some of the verification activities are performed as a review process, others are using software tools to do part of the activity (or whole) automated. Nevertheless, the existing UAL life-cycle requires a lot of complicated human-driven activities and is potentially vulnerable to human errors.

TABLE 1. SOFTWARE TOOL-SET FOR UAL DESIGN AND VERIFICATION

Activity	**Tool**
SWRS development	There are no specific tools
SWRS review	There are no specific tools
UAL design	RPCT
UAL design review	There are no specific tools
UAL static testing	ROVT
UAL dynamic testing	RPCT Simulator component
Requirements tracing	TraceTrend

IV. THE INCLUSION OF FORMAL VERIFICATION TOOLS IN THE EXISTING SOFTWARE LIFE CYCLE

In the previous chapters, the critical role of verification at different stages of product development, intended for use in high-safety systems, was emphasized. In this regard, the earlier an error is detected, the lower the cost of resources required to correct it. And the more alternative methods of verification can be applied to the object at the same stage, the higher the probability of detecting various types of defects.

Thus, by applying as many alternative methods of verification as possible, at the earliest stages of product development, we maximize the number of detected defects, and minimize the probability of detecting a defect at later stages of the life cycle, when the cost of their correction increases exponentially. It is with this purpose that we included the use of formal verification tools in the existing software life cycle, described in the previous chapter [1-5].

It should be noted that, although formal verification can potentially be applied to the development of a wide range of different systems, in our work we relied on some specific features of the product RadICS FSC Platform, and its life cycle of development, mentioned below.

First, the execution of user application logic in RadICS FSC Platform is sequential and clearly deterministic. The order of execution of functional elements of the logic algorithm is fixed and remains constant in each cycle of execution of the algorithm. In this case, the execution of the next functional element begins only after the completion of the previous functional element, and feedback elements are optimized by the compiler in such a way that the functional block, dependent on the data, will be executed after all the blocks that generate these data. This implementation of user application logic allows to avoid potential problems, typical for systems, in which the sequence of execution of functional elements can change between cycles of execution of the algorithm, or systems, where parallel execution of two or more functional elements is possible. Determinism and sequential order of algorithm execution allows to significantly simplify the formalization of individual components of user application logic.

Secondly, the number of types and variants of functional elements that can be used when programming user application logic is finite and clearly defined. We have 104 functional elements - components of user logic. They can have a variable (in a limited range) number of input parameters, can have additional (predefined) parameters that affect the calculation of the output value, and can also have additional output results, but all these characteristics for each block are known. Within the capabilities of FSC Platform, the user can construct their own functional elements, but in their implementation, there will remain the basic functional elements of the platform. This approach makes it theoretically possible to formalize user logic of any complexity (if there are formal algebraic models for all basic functional blocks).

As the point of application of formal verification methods on the model of software development life cycle for FSC platform, the stage "UAL design" was chosen (see Fig. 2 "Adaptation of UAL life cycle"). The arguments in favor of choosing this stage were the following:

- This stage is quite early, and therefore the cost of correcting the errors found will be relatively low.
- At this stage, the representation of most algorithms of user logic goes from abstract formulation by means of functional requirements to combinations of specific basic functional blocks, and therefore they can be formalized with a high degree of accuracy.
- Finally, unlike the following stages, where part of the verification is performed using software tools, the UAL design stage, as can be seen in the same figure, is checked (not counting Requirements tracing, which checks only one of many characteristics of requirements – their integrity) by means of Technical review, and therefore leaves a potential possibility of human error. The use of more automated verification tools at this stage can significantly increase the probability of detecting errors in the design.

The formalization of user logic is based on the principle of decomposition. First of all, subsystems and individual diagrams (algorithms) are formalized. Certain algorithms can use the results of calculations of other algorithms, some can be not related to other calculations, but for all the same approach is applied.

Each diagram (algorithm) of user logic is divided into agents. Each functional element of the diagram that exchanges data with at least one other functional element of this or another diagram is an agent. Also, elements of the external environment that cannot be described using basic functional elements but can either send data to other agents or receive data from them, will be considered agents. In case a functional element has some configuration parameters, they are also considered input parameters for calculating the output data of the agent.

If a certain agent needs the results of calculations of other agents for its calculations, then it is written as a function of the specified agents. For convenience, complex functions can be denoted by variables, and in further calculations use the names of these variables instead of the full substitution of all agents involved in the calculations.

Thus, at a certain stage, all the output data of a separate algorithm of user logic can be written as a set of functions of input data, while preserving the order of execution of logic and dependencies between functional blocks. Moving up a level, individual complex algorithms can be formalized as agents - functions of whole algorithms, and so on until the highest level of user logic can be formalized as one or several agents. Having user logic in the form of a fully formalized algebra of interaction between agents, one can perform an analysis of the model corresponding to these agents, for such important characteristics of the system as incompleteness, non-determinism, compliance with the library of functional blocks, safety properties, etc. Having models of the same system formalized at different stages of the life cycle, one can perform a comparative analysis to make sure that no elements were lost or added in the process of development.

V. THE APPLICATION OF FORMAL VERIFICATION ON THE EXAMPLE OF A PRACTICAL PROBLEM

For the demonstration of the practical application of the proposed approach, a formalized description and verification of a fragment of the project's user application logic developed in the RPCT IDE [17] have been performed. The algorithm (ALG) is presented as a main scheme representing the composition of three schemes (*S1_ALG, S2_ALG, S3_ALG)*, each of which embodies a combination of elements; figure 3 (a, b, c).

The task of formalizing the description of algorithm schemes and verification can be categorized as a problem of checking the correctness of the design.

Stage 1

The algorithm implements the parallel composition of two schemes, the outputs of which feed into a third scheme that evaluates their results. Agents operating within the scheme exchange signals: *cmpc_fp_gr_1, or2_11, or2_12, and_1, mul_fp_1, tctc_filter_1, switch_fp_1, or4_1, cmpc_fp_gr_2, or2_21, or2_22, and_2, mul_fp_2, tctc_filter_2, switch_fp_2, or4_2, not, or3, cmpc_fp_eq, xor_1, xor_2, cnt_up_1, cnt_up_2, cnt_up_3*.

Additionally, there are agents such as the Monitoring and Tuning System (MATS) and the external environment (ENV).

Each agent functions is an element of the scheme and can receive and transmit a certain number of signals to and from other agents. The types of agents correspond to the types of scheme elements, namely: *cmpc_fp_gr, or2, or3, or4, and, mul_fp, tctc_filter, switch_fp, not, cmpc_fp_eq*.

Stage 2

Next, the description of parameterized behaviors and actions defined in the previous stage for the agents is performed. The agents' names and specific parameter values serve as the parameters.

The description of top-level behavior takes the following form:

$$B = (S1\_ALG \parallel S2\_ALG \parallel S3\_ALG \parallel BTime), \quad (1)$$

where B is the behavior identifier, and the right-hand side of expression (1) represents a behavioral expression, which in turn is a parallel || composition of actions and behaviors. Parallel composition considers various possibilities of sequences of behavior components B.

As an example, let's provide the description of the behavior of the first scheme, *S1_ALG*:

$$BTime = (A\_timer\_start + A\_timer\_expired) \quad (2)$$

$$S1\_ALG = P11 \parallel P12 \parallel P13, \quad (3)$$

where, BTime, A_timer_start = (TIMER == 0) -> <> (TIMER == 1), and A_timer_expired = (TIMER == 1) -> <> (TIMER == 0) are environment variables, P11, P12, P13 represent behaviors of algorithmic schemes, respectively.

The description of behaviors *P11, P12, P13* is as follows:

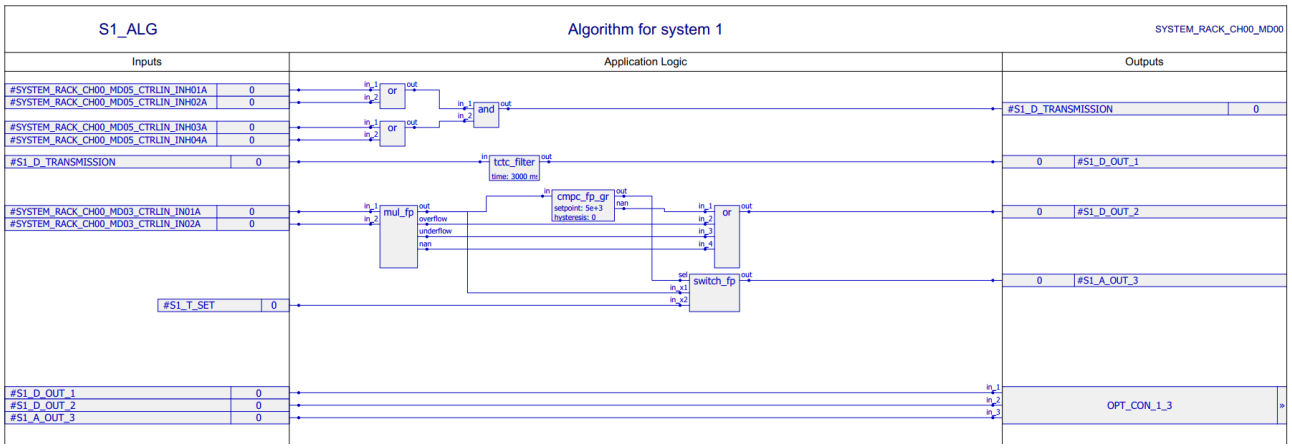
P11 = ((Aor2(or2_11, MATS, MATS, and1) || Aor2(or2_12, MATS, MATS, and1)); Aand (and1, or2_11, or2_12, tctc_filter_1)),

P12 = Btctc_filter(tctc_filter_1, and1, xor_1, 3000),

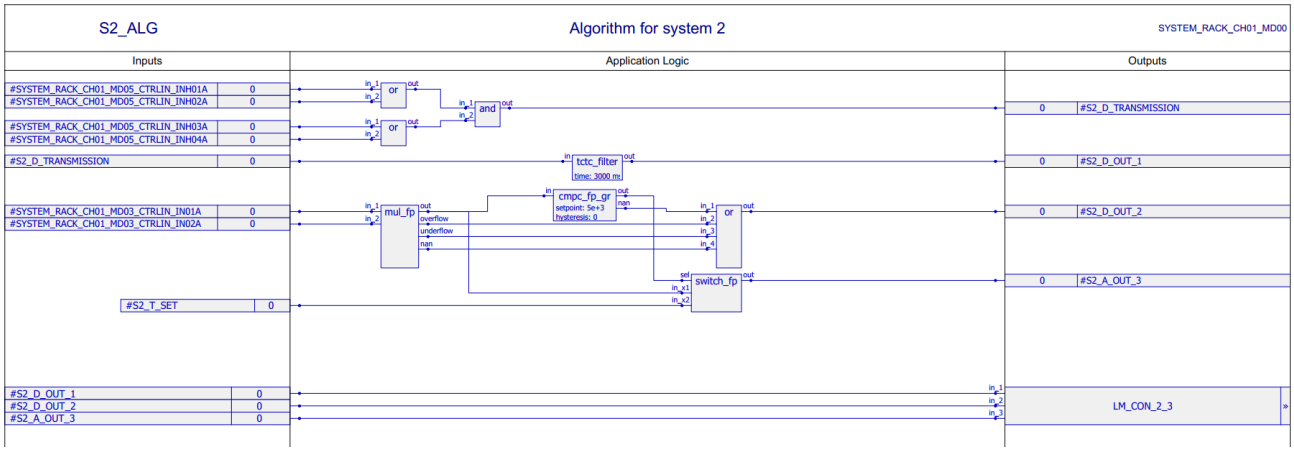
P13 = (Amul_fp(mul_fp_1, MATS, MATS, cmpc_fp_gr_1, switch_fp_1, or4_1, or4_1, or4_1); Acmpc_fp_gr (cmpc_fp_gr_1, mul_fp_1, switch_fp_1, or4_1, 5000, 0); (Aor4(or4_1, cmpc_fp_gr_1, mul_fp_1, mul_fp_1, mul_fp_1, xor_2) || Aswitch_fp(switch_fp_1, cmpc_fp_gr_1, mul_fp_1, MATS, cmp_fp_eq))).

An example of a formalized description of behavior actions is as follows:

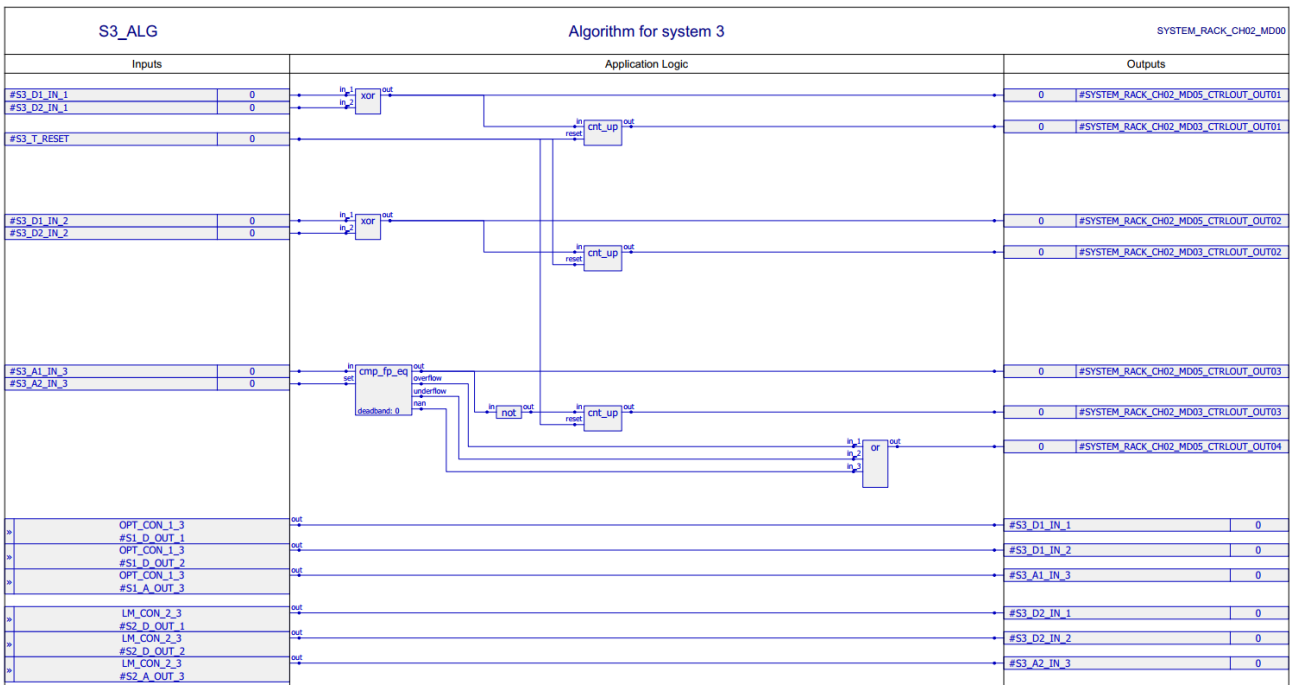
Acmpc_fp_gr(s, x1, x2, y, setpoint:real, hysteresis:real) = (in:real, out:bool, nan:bool) 1 -> <receive(x1 : signal(in)), send(x2 : signal (out)), signal (y : nan)> out = (in > setpoint – hysteresis), where the semantics of actions are as follows.



a) S1_ALG



b) S2_ALG



c) S3_ALG

Fig. 3. Example schemes User Application Logic

The action parameters include an instance list. For example, in the action *Acmpc_fp_gr(s, x1, x2, y, setpoint:real, hysteresis:real)*, '*s*' is the key instance or an instance of the element where the action is being executed. '*s*' receives signals from instances '*x1*' and '*x2*', and forwards instances '*y*'. *'Setpoint:real*' and *'hysteresis:real*' are other parameters of the element. Signals are represented using the notification "*signal(x)*", which has a parameter '*x*' that can have types – real, int, bool. The action description is preceded by a list of local parameters that operate within the protocol. The process element of the action can receive signals: receive (*x:signal(in)).* Instance '*s*' received a signal with the parameter 'in' from instance 'x'. The process element of the action can send signals: *send(x:signal(out))*. Instance '*s*' sent a signal with the parameter 'out' to instance '*x*'. The postcondition out = (in > setpoint – hysteresis), where 'out' is a bool, means that 'out' is 1 if (in > setpoint – hysteresis) is true, and 0 otherwise. The set of behavior actions for the scheme *S1_ALG* is as follows:

MS1_ALG = {Aor2, Aand, Btctc_filter, Amul_fp, Acmpc_fp_gr, Aor4, Aswitch_fp}.

The semantics and description of behavior actions are similar to the description provided above.

VI. CONCLUSIONS

The article presents the results of applying formal verification methods to the embedded software of programmable controllers for safety-related applications within the context of the applied safety lifecycle.

An implementation of the developed approach is provided through an adaptation of the UAL life cycle of embedded software development using a programmable controller RadICS FSC Platform as an example.

The main research findings are as follows:

- the UAL life cycle was adapted through the incorporation of formal verification methods;

- the carried-out formalization was modeled on an algebraic server and explored for incompleteness and non-determinism. The tested UAL schemes do not exhibit deadlocks or non-determinism;

- experimental evidence confirms that the implementation of formal verification methods allows for a reduction in testing time and minimizes the impact of the tester's expertise level on verification results.

The following steps of research should include:

- the formalization of schemes was done manually, but this process should be automated;

- the developed algebraic models can be applied for generating verification tests of binary bitstream code;

- the developed algebraic models can be applied for verifying security properties or other specific requirements that may be considered for certain devices, such as timing correctness, signal propagation, and more.

REFERENCES

[1] Radiy. URL: http://www.radiy.com/en/.

[2] IEC 61508:2010. Functional safety of electrical/electronic/programmable electronic safety-related systems. Published. 2010 – 04. IEC Standards, 2010. 594 p.

[3] IEC 61508:2010. Functional safety of electrical/electronic/programmable electronic safety-related systems. Published. 2010 – 04. IEC Standards, 2010. 594 p.

[4] IEC 61513: 2011. Nuclear power plants – instrumentation and control for systems important for safety – general requirements for systems. Published. 2011 – 08 – 25. IEC Standards, 2011. – II, 86 p.

[5] IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations, IEEE Standard 603-2009, 2009.

[6] IEEE Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations, IEEE Standard 7-4.3.2-2010, 2010.

[7] IEEE Standard for Software Safety Plans, IEEE Standard 1228-1994, 1994.

[8] IEEE 1012:2016 Standard for Software Verification and Validation.

[9] Nuclear power plants—Instrumentation and control systems important to safety—Software aspects for computer-based systems performing category A functions, International Electrotechnical Commission, IEC 60880:2006, 2006.

[10] Letichevsky A., Gilbert D. Interaction of agents and environments / D. Bert, C. Choppy (Eds.) // Resent trends in Algebraic Development technique, LNCS 182.27. – Berlin: Springer-Verlag, 1999. P. 311–328.

[11] Letichevsky A., Letychevskyi O., Peschanenko V., Insertion Modeling and Its Applications, Computer Science Journal of Moldova, Vol. 24, Issue 3, 2016, P. 357-370.

[12] Letichevsky O., Odarushchenko O., Peschanenko V., Kharchenko V., Moskalets V. Insertion Semantics of VHDL as Electronic Design languge. Cybernetics and Systems Analysis, Vol. 58, No. 2, March, 2022. P. 289-298. doi 10.1007/s10559-022-00461-2.

[13] Letychevskyi, O.O., Peschanenko, V.S., Kharchenko, V.S., Volkov, V.A., Odarushchenko, O.M. Modeling Method for Development of Digital System Algorithms Based on Programmable Logic Devices. *Cybernetics and Systems Analysis* this link is disabled, 2020, 56(5), стр. 710–717. ISSN 1019-5262.

[14] Chunxiao Li, Anand Raghunathan, N.K. Jha. Improving the Trustworthiness of Medical Device Software with Formal Verification Methods. September 2013. IEEE embedded systems letters 5(3), P.50-53.

[15] Vassil Todorov, Frederic Boulanger, Safouan Taha. Formal verification of automotive embedded software. In FormaliSE FormaliSE '18: 6th Conference on Formal Methods in Software Engineering , June 2, 2018, Gothenburg, Sweden. ACM, New York, NY, USA, 4 pages.

[16] Mario Glelrscher, Jaco van de Pol, Jim Woodcock. A manifesto for applicable formal methods. Springer. Software nad Systems Modeling. Published online:18 August 2023, 13 pages. https://doi.org/10.1007/s10270-023-01124-2.

[17] Mario Glelrscher, Jaco van de Pol, Jim Woodcock. A manifesto for applicable formal methods. Springer. Software nad Systems Modeling. Published online:18 August 2023, 13 pages. https://doi.org/10.1007/s10270-023-01124-2.

[18] Eui-Sub Kim, Dong-Ah Lee, Sejin Jung, Junbeom Yoo, Jong-Gyun Choi and Jang-Soo Lee. NuDE 2.0: A Formal Method-based Software Development, Verification and Safety Analysis Environment for Digital I&Cs in NPPs Journal of Computing Science and Engineering, Vol. 11, No. 1, March 2017, P. 9-23.

[19] Jaeyeob Kim, Eui-Sub Kim, Junbeom Yoo, Young Jun Lee, Jong-Gyun Choi. An Integrated Software Testing Framework for FPGA-Based Controllers in Nuclear Power Plants. Nuclear Engineering and Technology 48 (2016), P.470-481.

[20] Sang Hun Lee, Seung Jun Lee, Jinkyun Park, Eun-chan Lee, Hyun Gook Kang. Development of simulation-based testing environment for safety-critical software. Nuclear Engineering and Technology 50 (2018), P.570-581.

[21] Jiawen Xiong 1, Gang Zhu, Yanhong Huang , Jianqi Shi A User-Friendly Verification Approach for IEC 61131-3 PLC Programs. 1,2Electronics 2020, 9, 572; doi:10.3390/electronics9040572.