Reliability Engineering and System Safety 202 (2020) 107029

Contents lists available at ScienceDirect

Reliability Engineering and System Safety

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

A practical application of NUREG/CR-6430 software safety hazard analysis to FPGA software☆

Sejin Jung[a], Junbeom Yoo[a,*], Young-Jun Lee[b]

[a] Konkuk University, Division of Computer Science and Engineering, 120 Neungdong-ro, Gwangjin-gu, Seoul, 05029, Republic of Korea
[b] Korea Atomic Energy Research Institute, Man-Machine Interface System Team, 989-111 Deadeok-daero Yuseong-gu, Daejeon, 34057, Republic of Korea

ARTICLE INFO

Keywords:
NUREG/CR-6430
Software hazard analysis
FPGA
Digital I&C

ABSTRACT

Hazard analysis is a widely-used technique to achieve the system/software safety by analyzing hazards systematically. While programmable logic controller-based digital instrumentation and control systems have been replaced with field programmable gate array (FPGA)-based ones, hazard analysis on FPGA software as well as FPGA-based controllers becomes one of the prerequisites of operational approval. The NUREG/CR-6430 provides applicable processes/methods of software safety hazard analysis (*e.g.,* guide phrases and analysis techniques). Hazard analysis of FPGA software is different from typical software hazard analysis, since the FPGA is a hardware-based platform. This paper proposes a refined process and guide phrases at the software requirement analysis part in NUREG/CR-6430, tailored for the new target - FPGA software. We performed hazard analysis on FPGA software for a prototype version of an FPGA-based controller in Korea to show feasibility of the refined process and guide phrases.

1. Introduction

Digital instrumentation and control system (I&Cs) in nuclear power plants should be analyzed and evaluated to ensure that the systems are acceptably safe from hazards/risks/failures [2,3]. Hazard analysis is a method for identifying potential hazardous portions of a system. Eliminating, reducing, or avoiding the impact of identified hazards should be appropriately followed to achieve the freedom from the hazards [4]. Software, which is a part of systems, can also be a cause of system hazards, and software hazard analysis should be performed rigorously [5,6].

There are several standards/guidelines for software safety/hazard analysis [4,7], and safety plan [8] for nuclear safety system software. The NUREG/CR-6430 suggests methods (processes) for analyzing software-affected hazards during whole software development life cycle (SDLC) like safety plan. It provides an analysis process for each phase of software development, and also provides guide phrases and several techniques that can be applied to software hazard analysis. The analysis process consists that are identify a high-level hazard, analyze each element of requirements with guide phrases, and so on. The details of the NUREG/CR-6430 are explained in the next section.

Field-programmable gate array (FPGA) has received much attention from the nuclear industry to develop digital I&C systems as an alternative platform of programmable logic controller (PLC). There are several standardization efforts for using FPGA in nuclear systems [9,10]. FPGA-based digital controllers should be evaluated/analyzed that the systems are acceptably safe to operate, too. Since the typical FPGA development includes two different aspects of development, such as software and hardware, we need to apply hazard analysis hierarchically and compositionally [10,11]. Although the NUREG/CR-6430 might provide a useful approach to perform hazard analysis against FPGA software, we need extensions or refinement methods to analyze FPGA software throughly. Nevertheless, there is no hazard analysis result reported for FPGA software used in digital I&Cs. There are only a few approaches concerning FPGA software verification, simulation [12,13], and FPGA hardware reliability [14–16], to the best of our knowledge.

This paper proposes a refined hazard analysis process at the

Acronyms and Abbreviations: FPGA, Field-Programmable Gate Array; PLC, Programmable Logic Controller; DFLC-N, Digital FPGA Logic Controller-Nuclear; IEC, International Electrotechnical Commission; HDL, Hardware Description Language; EDA, Electronic Design Automation; RTL, Register-Transfer Level; P&R, Place & Route; SDLC, Software Development Life Cycle; HAZOP, HAzard and OPerability; RPS, Reactor Protection System; PHL, Preliminary Hazard Lists; PLD, Programmable Logic Device; BP, Bistable Processor

☆ This paper was originally published in *Korean Nuclear Society Autumn Meeting* 2016 [1].

* Corresponding author.
E-mail address: jbyoo@konkuk.ac.kr (J. Yoo).

https://doi.org/10.1016/j.ress.2020.107029
Received 7 September 2018; Received in revised form 19 April 2020; Accepted 12 May 2020
Available online 19 May 2020
0951-8320/

software requirements phase of the NUREG/CR-6430 that is applicable to FPGA software. It extends the steps of "identifyng software responsible hazards" and "applying guide phrases" in the NUREG/CR-6430 and also guide phrases to incorporate the hardware aspects of FPGA software requirements. The proposed refined process and guide phrases support to check for hardware aspects of software requirement hazard analysis on the FPGA software. We performed hazard analysis upon FPGA software in accordance with the NUREG/CR-6430 and proposed process in a case study. We used the hazard and operability (HAZOP) technique and one version of the FPGA software requirements specification of a process module in the digital FPGA logic controller-nuclear (DFLC-N) [17], which is an FPGA-based I&C controller under development in Korea. We also discussed the applicability and feasibility of the refined process presented in this paper through comparative analysis of the analysis aspects and analysis results.

The remainder of this paper is organized as follows. Section 2 introduces the FPGA development process and hazard analysis as a background. Section 3 presents the refined process of the NUREG/CR-6430 proposed in this paper, and we explain the performing hazard analysis upon the FPGA software as a case study in Section 4. Section 5 concludes this paper and provides remarks on future research extensions and directions.

2. Background

2.1. The FPGA development process

FPGA-based digital I&Cs should follow the development life cycle described in the IEC (International Electrotechnical Commission)-61513 standard [3]. FPGA-based systems, however, have specific features that developing part using hardware description language (HDL) is classified into software, while after downloading to a chip is classified into hardware. Therefore, FPGA should be developed to comply with both the IEC-60880 standard [18] in terms of software and IEC-60987 standard [19] in terms of hardware. 〈Fig. 1〉 depicts the V-shaped life cycle of FPGA development explained in IEC-62566 [9], consisting of both software and hardware aspects. The software aspect follow the typical development life cycle [11] presented on the left side of the figure.

FPGA software development is fully automated by the FPGA logic synthesis tools and commercial electronic design automation (EDAs) tools provided by FPGA vendors. After programming an register-transfer level (RTL) design using HDLs, synthesis software such as "*Synopsys Synplify Pro*", "*Precision RTL*" and "*Encounter RTL Compiler*" can be used to transform the design into a gate-level design (*i.e.,* netlist). The EDA tools of FPGA vendors such as "*Xilinx ISE Design Suit,*" "*Altera Quartus 2*" and "*Microsemi Libero SoC*" perform place and route (P&R) operations to place and map all netlist elements physically and prepare a downloadable file through configuration. At each step of the FPGA SDLC, designers often perform *simulation-based verification* to confirm that each artifact satisfies its requirements specification, such as *behavioral simulation* for RTL designs, *logic simulation* for gate-level design, and *post-layout simulation* on layout design.

The FPGA software development includes both software and hardware aspects as requirement analysis, design, automatic synthesis, and P&R and software requirement specifications for the FPGA software are defined in the form of hardware aspect modules by FPGA board units. Hazard analysis can also be performed to analyze potential hazards at each phase, which is pertinent to our research and will be discussed in a subsequent section.

2.2. Software hazard analysis of the NUREG/CR-6430

The NUREG/CR-6430 "*Software Safety Hazard Analysis*" [4] was proposed by united states nuclear research commission (NRC) to provide software hazard analysis guidelines for nuclear power plants. The NUREG/CR-6430 adopts basic concepts of the software safety plan in IEEE-1228 standard [8] and provides useful guide phrases that can help to perform hazard analysis at each phase of development. Hazard analysis is performed at each phase of the SDLC on different artifacts and the hazard analysis in later phases repeadtedly uses analysis artifacts from earlier phases.

Software hazard analysis in the NUREG/CR-6430 consists of two main steps. The prerequisites of the software hazard analysis, that are a preliminary process for finding PHLs (Preliminary Hazard List), are followed by software requirements hazard analysis, which finds software-responsible hazards/requirements and their evaluates criticality, as described in 〈Fig. 2〉. Each software requirement can then be analyzed using guide phrases provided by the NUREG/CR-6430.

〈 Table 1 〉 lists some example guide phrases that can help analysts perform hazard analysis on development elements in the SDLC. The guide phrases is provided by four clauses about "quality," "aspect," "phase," and "guide phrases" as shown in the 〈Table 1〉. The "quality" and "aspect" refer to goal and target of the guide phrases. The "Phase" refers to the phases of the SDLC in which analysis is applied, and "guide phrases" clause is to help the analysis start and guide analysis. It is worth noting that the NUREG/CR-6430 designates no specific hazard analysis techniques, but HAZOP [20] would be best suited. The NUREG/CR-6430 method has been used to software hazard analysis on a PLC development process [21,22].

The NUREG/CR-6430 provides a hazard analysis process in accordance with the software safety plan in the IEEE-1228 standard and guide phrases at each phase of SDLC. It is, however, not straightforward to apply into FPGA software due to the hardware-related aspects of

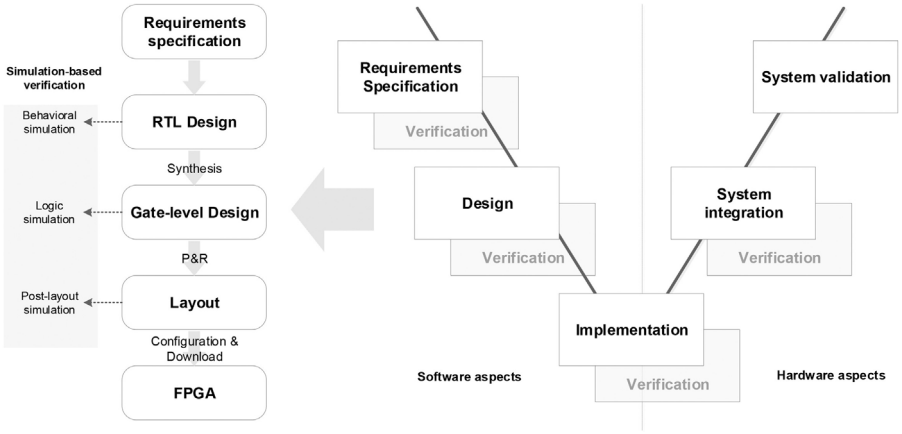

Fig. 1. Typical development life-cycle for FPGA-based platforms.

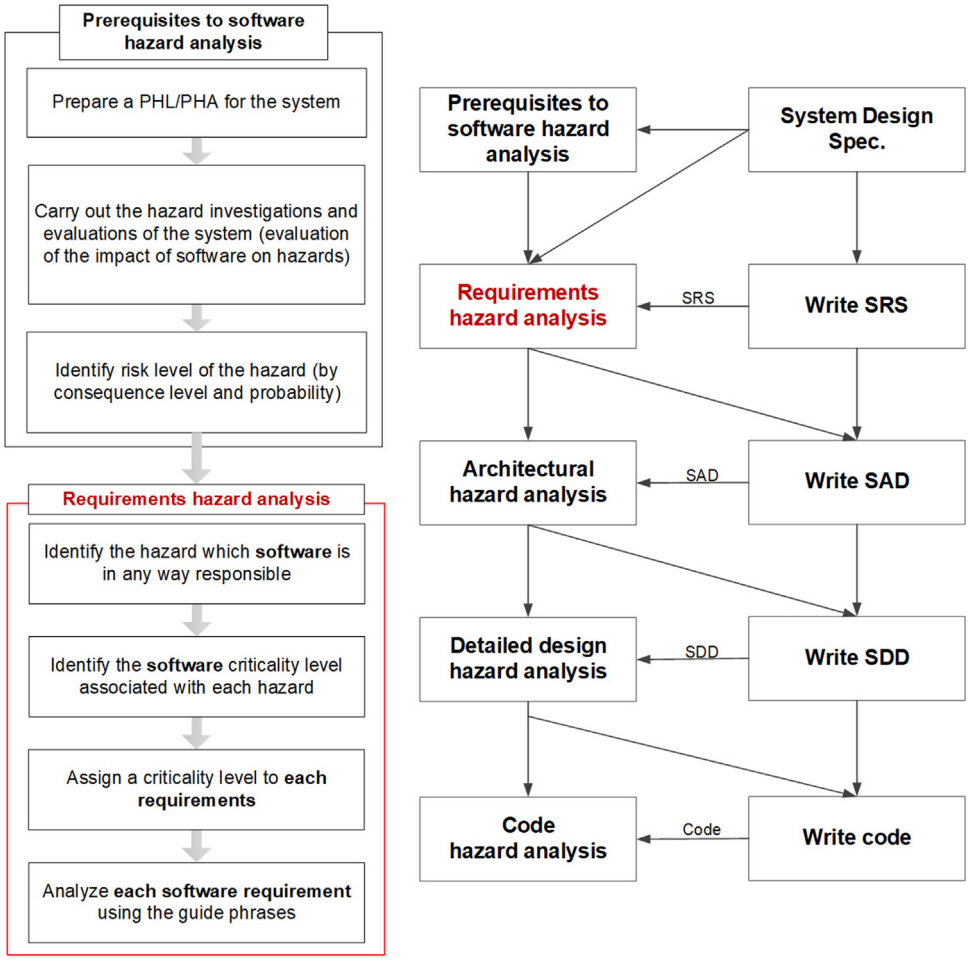

Fig. 2. The software hazard analysis process on the requirements specification of the NUREG/CR-6430.

Table 1
Example guide phrases in the NUREG/CR-6430 and this work.

Quality	Aspect	Phase	Guide Phrases	Note
Functionality		RA	Function is not carried out as specified	NUREG/CR-6430 suggests.
		RA	Function is not initialized properly before being executed	
		RA	Trigger conditions are satisfied but function fails to execute	
		R	Function uses incorrect inputs	
Accuracy	Sensor	RADC	Stuck at all zeroes	
		RADC	Stuck at all ones	
		RADC	Stuck elsewhere	
		RADC	Below minimum range	
		RADC	Above maximum range	
Accuracy	Circuit	RADC	Stuck at all zeroes	This paper refines.
		RADC	Stuck at all ones	
		RADC	Stuck elsewhere	
	Memory	RDC	Stuck at all zeroes or ones	
		RDC	Stuck elsewhere	
Security		R	SW are not encoded	

FPGA-based reactor protection system (RPS) software. First, the process of identifying software responsible hazards and assigning software criticality levels to each requirement element differ slightly from the one against the PLC-based software requirements specifications because FPGA software requirement specifications are defined in hardware-dependent FPGA module units independently of the hardware. Specification starts by dividing the entire system into board and component modules, rather than the functional ones as typical software requirement specifications. This feature makes it difficult to identify software responsible hazards lists from system-level hazards and assign criticality levels to each software requirement elements, since almost hazards in PHL care about system functions not board and components. Aadditionally, using guide phrases for hazard analysis must also account for hardware-specific aspects. This paper proposes an extended process and guide phrases that can incorporate such hardware-related concerns efficiently.

2.3. Related work

The safety/hazard analysis of FPGA-based digital I&C systems has been researched several approaches. The FPGA component failure analysis proposed in [14] is used to quantify error propagation at the design level by calculating failure rates from FPGA logical information. Neto et al. [12] and Vismari et al. [23] proposed practical approaches to the safety analysis of PLD (Programmable Logic Device)-based safety systems. They uses HDL descriptions to perform safety analysis on the PLD, especially, a code inspection, which are based on checklists from [24], is applied to analyze software. Checklists are similar to the guide phrases at the code-level analysis in the NUREG/CR-6430, however, it only focuses on code-level analysis not other phases of SDLC.

Reliability analysis about FPGA-based triple modular redundancy systems have also been proposed in several papers. Jung et al. [15] proposed a mathematical model to estimate/predict the failure rates of on-board processor systems, which is based on an SRAM (Static Random-Access Memory)-based FPGA. They presented an on-board processor system adopting triple modular redundancy and an external

scrubber mitigation process, and calculated system failure rates using single-event upsets rates and information of system configurations. Benites et al. [25] also proposed a reliability calculation process for the SRAM-based FPGA mitigated by triple modular redundancy and memory scrubbing. They represent experimental results of reliability calculations with and without mitigation designs with fault injection and heavy ion irradiation. These papers presented calculations of failure rates and reliability levels of FPGA-based systems adopting a triple modular redundancy designs to mitigate failures in terms of hardware.

McNelles et al. [26,27] compared and contrasted and the results of fault tree analysis and dynamic flowgraph methodology for FPGA-based safety-critical systems. They performed hazard analysis on logic-level block diagrams and focused on comparing quantitative results by differecnes of static and dynamic approach. A probabilistic model checking-based analysis method has been proposed for the quantitative analysis of triple modular redundancy partitioning in the FPGA of design phases [16]. It proposes a formal model for triple modular redundancy system irrespective of the partition size with capturing single and double-cell upsets. Model checking is used to perform quantitative analysis of the model about availability and reliability. Various papers have proposed FPGA analysis methods based on failure rate calcuation or reliability at the board or component level. Analysis of desing phases has also focused on hardware-level design. Such analysis may serve as a useful basis for prbability or risk analysis at the software implementation phases.

FPGA software analysis has also been studied in various ways, including FPGA SDLC model [28], verification processes [29], and simulation-based approaches [13,30,31]. The authors of [28] proposed the W model, which reflects the verification of FPGA software development during the development life-cycles. The W model covers testing activities ranging from requirement analysis to system integration. Verification based on simulation and testing considering develpment phases [13,30] and the application of functional validation and system assessment through simulating with pre-defined failure scenarios [31] are simulation-based approaches to FPGA verification. However, such methods only focus on the functional verification and validation of the FPGA. Huang et al. [32] presented a systematic literature review of studies on failure mode and effect analysis. However, very few hazard analysis of FPGA software were contained in their review. McNelles et al. [33] proposed a failure taxonomy for assessing the reliability of the FPGA-based I&C systems. The structure of the taxonomy consists of possible failure modes (failure categories), uncovering, mitigation, and effects at the FPGA decomposition level combined with existing information regarding safety analysis results, functional safety standards, or fault categories. It may be more helpful to classify hazard analysis results by detection, effect, mitigation information in the taxonomy when analyzing software requirements.

3. A refined process for hazard analysis of software requirement specifications

This paper proposes a refined process for the hazard analysis of FPGA software requirement specifications. It extends the hazard analysis process of the NUREG/CR-6430 to incorporate the hardware-specific features of FPGA software. We also extend the guide phrases of the NUREG/CR-6430 to handle the circuit and memory aspects of FPGA software. ⟨Fig. 3⟩ presents the refined process for FPGA software requirement hazard analysis. We compose the process into six steps and also change the order of certain steps. FPGA software requirement specifications are often written in components (hardware) modules units since FPGA-based controllers consist of multiple components, such as a set of FPGA boards. The proposed refined process is composed by applying these characteristics of the FPGA software requirement.

The "*identify the hazards which software is in any way responsible*" step in ⟨Fig. 2⟩ is the first step of requirement hazard analysis in the NUREG/CR-6430. This step identifies system hazards for which software is responsible, and assigns a criticality levels of the software requirements. However, this step has some difficulties in identifying software responsible hazards directly in the PHL as discussed in Section 2.3. Since hazards in a PHL and related software components are profoundly different in a hierarchical structure, FPGA software requirement specifications, which are defined in a hardware-module specific manner, are not directly connected to system hazards. Therfore, we add an additional step "*Identify hazards of software overall function aspects*" before "*Identify the hazards which software is in any way responsible in PHL (with software funcionality level hazards)*" to lower the gap.

The first step, "*Identify hazards of software overall function aspects*," takes into account all hardware-specific features of FPGA software, such as memory and signals, as shown in ⟨Table A2⟩. The next step, "*Identify the hazards which software is in any way responsible in PHL*," identifies and connects the consequences of software hazards to system-level hazards, which is the same as NUREG/CR-6430. While the NUREG/CR-6430 uses this step to assign a criticality level to each software component, the refined process connects analysis results to system elements, additionally. We recommend using these two steps hierarchically according to the system structure.

The "*Analyze each software requirement using the guide phrases*" step extends the "*Connect hazards for analyzing consequences to higher level component*" step. When analyzing each software requirement element using guide phrases, analysis should be performed according to two conceptual ways: the functional units of a requirement and the hardware components aspect on software in the ⟨Fig. 3⟩. The former refers to using guide phrases to analyze deviations in designated requirement element and the latter refers to using guide phrases to all elements in a specification. For these two steps, we tailored some guide phrases of the NUREG/CR-6430 to provide meaningful aspect for FPGA software requirements. ⟨Table 1⟩ shows some supplemented guide phrases and descriptions. We incorporate some additional guide phrases for the hazard analysis of FPGA software requirement specifications to reflect the specific characteristics of FPGA. These revisions mainly focus on the additional categories of "quality" and "aspect."

The step "*Connect hazards for analyzing consequences to higher level component*" progressively analyzes the hazard consequences from software into the higher component. It is necessary to analyze the effects of software hazards on hardware, boards, or system-level hazards either simultaneously or pregressively. This step makes connections between software hazards identified from the deviations using guide phrases and software functional-aspect level hazards. The final two steps in ⟨Fig. 3⟩ identify the criticality levels of software hazards and assign criticality levels to each requirement. These two steps proceed step by step based on results of the connections from before step.

⟨ Table 2 ⟩ is a sample worksheet table suitable for the refined process proposed in this paper. The clauses qualities and aspects are related to the guide phrases applied and the other entries are related to hazard analysis. The "*hazard on SW concern*" and "*hazard on PHL*" clauses are expressions of the refined process about connecting hazards. Im summary, this section introduced the refined process and guide phrases to support filling gaps and performing software hazard analysis more efficiently.

4. Case study

We conducted hazard analysis on an FPGA software requirement specification using the refined process and guide phrases proposed in this study. The target specification is one version of a process module in DFLC-N [17], which is an FPGA-based I&C controller developed in Korea. This case study introduces the feasibility and efficiency of the proposed process and guide phrases to software hazard analysis of FPGA software in accordance with the NUREG/CR-6430. Because of space limitations, we only focus on the most important and relevant

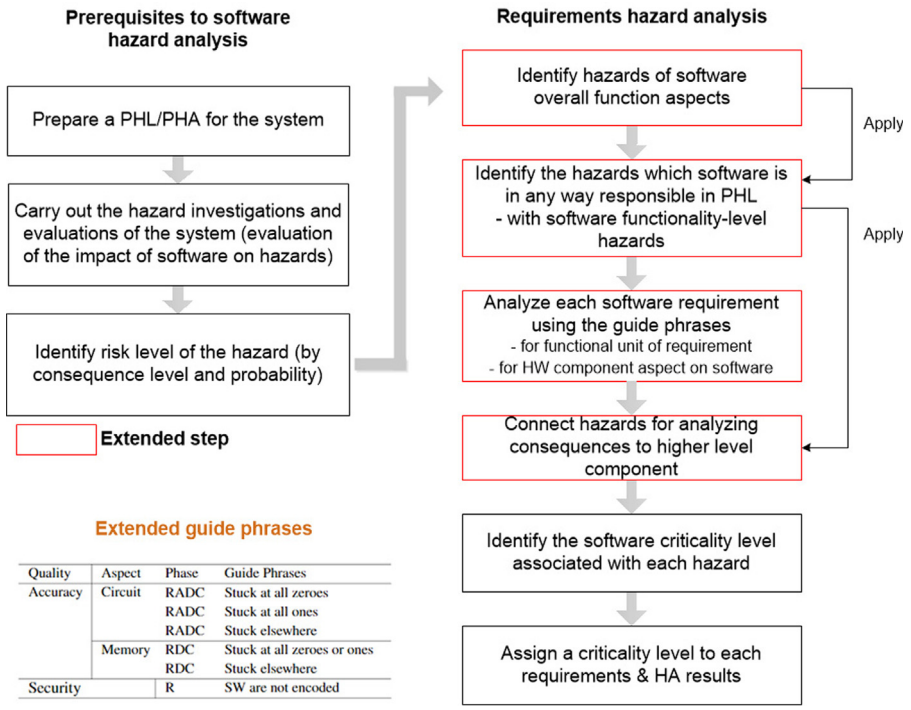

Fig. 3. A refined process for the hazard analysis of FPGA software requirement specifications.

results.

4.1. Target system software

DFLC-N is a safety-related FPGA controller for the I&Cs in nuclear power plants (NPPs). It consists of several modules/components, such as sub-rack, bus module, process module, and input/output module. FPM-01 is a general process module consisting of several sub-modules and FPGA boards, that performing core functions such as self-diagnosis, input/output control, and application control logic. CLFPM01 is a control logic for controlling and monitoring FPGA modules belonging to a FPM-01. The primary functions of a CLFPM01 module are "*reset and clock signal generation*," "*operation voltage monitoring*," "*diagnosis/send/receive of an input and output data*," and "*diagnosis/send/receive of data link/network communication data*," and "*status indication*." ⟨Fig. 4⟩ presents a part of the structures of the CLFPM01 of FPM-01. As shown in the figure, CLFPM01 logic consists of hardware-based structure, and software is also defined by such features.

4.2. A summary of requirements hazard analysis results

⟨ Fig. 5 ⟩ presents an analysis process with directions to corresponding results from our case study. The analysis results are summarized in the five tables in ⟨Appendix A⟩ according to a sequence of the proposed process. ⟨Table A1⟩ contains hazard lists that are a module level of the DFLC-N, which represent the reprequisites for software hazard analysis. The hazard category, which consists of "high," "middle," and "low," priority levels, is defined by domain experts according to the consequences and probabilities of the hazards. ⟨Table A2⟩ shows the results of first two steps that are the software function aspects hazards and its connection for identifying software responsible hazards. When analyzing the hazards in these steps, we considered the characteristics of FPGA which include hardware-related aspects such as viewing the signals of the PM software.

Finally, the results of analyzing each software requirement using guide phrases and final steps are listed in ⟨Table A3⟩ and ⟨Table A4⟩. These tables list portions of the hazard analysis results for the FPGA software requirement specifications of a CLFPM01 module. The worksheet in ⟨Table 2⟩ was used to perform analysis. Analysis was conducted to check requirement elements such as item, function/purpose and, parameter, and identify deviation/consequence by ascertaining what happens when situation of guide phrases occur. For example, guide phrases pertaining to a "sensor" can be combined with the function of the "*operating voltage monitoring function*" in our target software, which relates to sensor input. We express risk based on criticality levels only because calculating the failure probability of software is difficult in the requirement phase.

As shown in ⟨Table A3⟩, five potential hazards were identified by the analysis. For example, "*operating voltage monitoring*" function may be affected by deviations in the sensor for the stuck condition because the requirement specification does not specify any requirements for that sensor. Several deviations by guide phrases related to the range concept do not have any practical implications in terms of FPGA software. ⟨Table A4⟩ lists a portion of the analysis results for the accuracy-circuit pair, which uses the tailored guide phrases proposed in this paper. It also shows three potential hazards and these hazars are un-happended because the preventing function is already specified in the software requirement specifications. Hazards caused by ambiguous definitions of requirements also exist and software designers/developers must consider such hazards carefully when the next step of the development

Table 2
An example of an analysis worksheet.

Software hazard analysis worksheet												
No.	Qualities	Aspects	Item	Function/ Purpose	Parameter	Guide Phrases	Deviation	Conse-quence	Hazard	Risk	Hazard on SW functionality aspect	Hazard on PHL

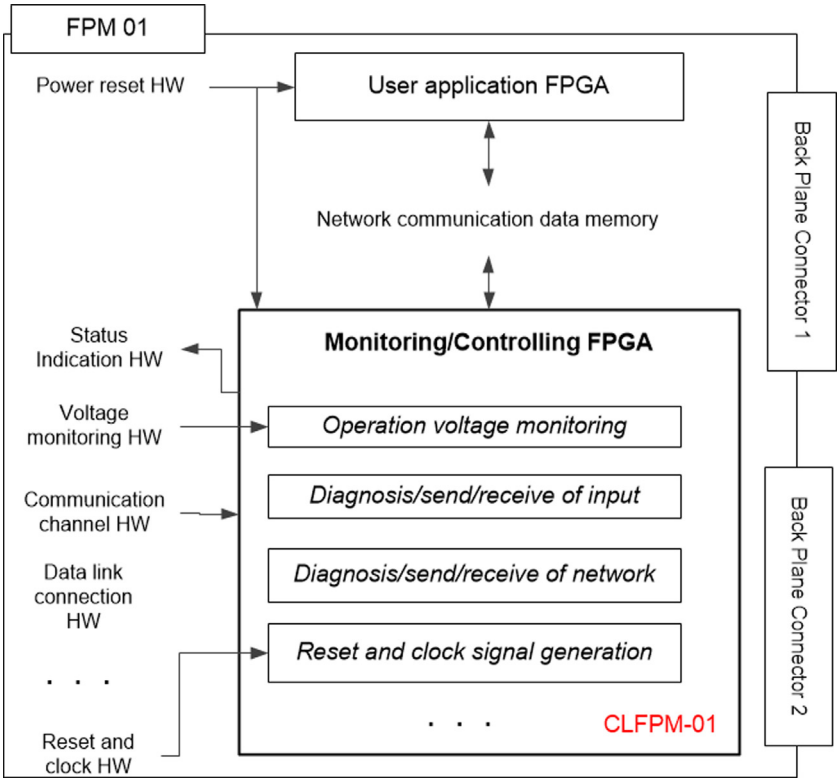

Fig. 4. Example structure of CLFPM01/FPM-01 modules.

proceed. As shown in the ⟨Table A3⟩ and ⟨Table A4⟩, an analyst can analyze the consequences of deviations in requirement elements in terms of software functionality aspect hazards and module-level hazards progressively.

Table A5 provides a summary of hazard analysis results that includes a number of hazards identified in the case study. The table shows that identified hazards can differ according to the guide phrases. Guide phrases provide a variety of perspective for the analysis of hardware aspects. Such hazards may or may not represent significant threats to the system according to the results of later development

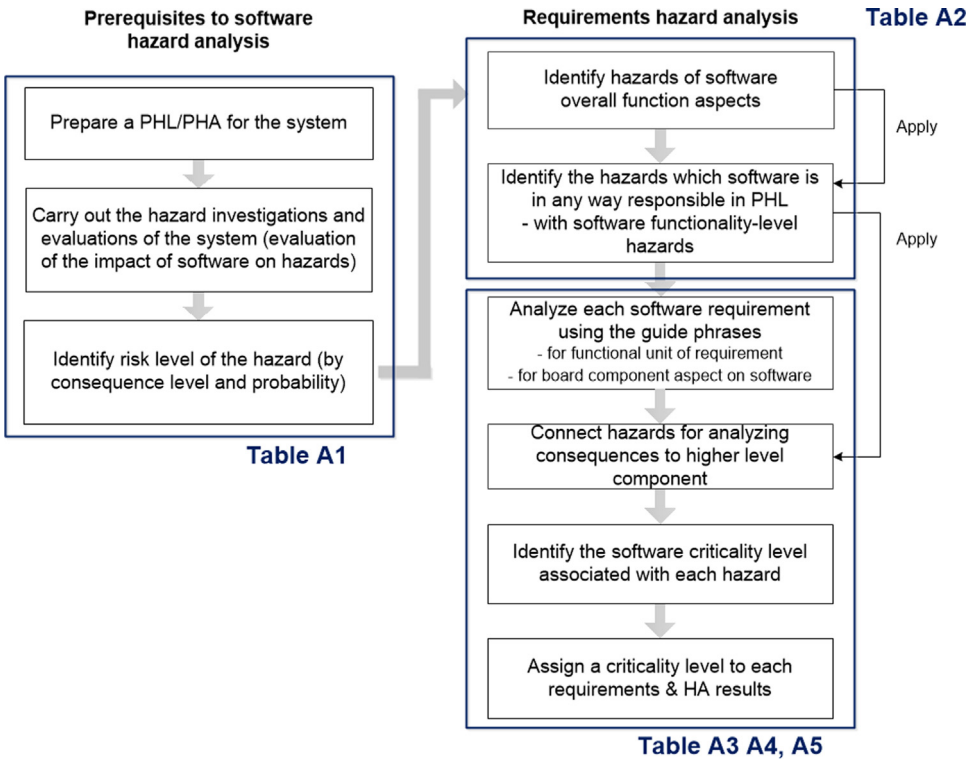

Fig. 5. A results of the case study along the proposed analysis process.

Table 3
Analysis aspects with requirement points in the analysis results.

Requirement point	Analysis aspects		
	The proposed process	NUREG/CR-6430	HAZOP analysis
Sensor	Deviation analysis	Deviation analysis	Cause
Input/Output	Cause, Deviation analysis	Cause, Deviation analysis	Cause
Timing	Deviation analysis	Deviation analysis	Cause, Deviation analysis
Function	Deviation analysis	Deviation analysis	Deviation analysis
Circuit	Deviation analysis	Cause	Cause
Security	Deviation analysis	Deviation analysis	N/A
Memory	Cause, Deviation analysis	N/A	Cause, Deviation analysis
Data bus	Deviation analysis	Deviation analysis	Cause, Deviation analysis
Network	Deviation analysis	Deviation analysis	Cause, Deviation analysis

phases. The proposed refined analysis process is helpful for analyzing hazards from software to system by bridging the gaps between software/hardware/system hazards at an identification of the clause of "Hazard on SW PHL" and "Hazard on PM PHL."

4.3. Discussions

The case study presented in this paper demonstrated that the proposed refined process and guide phrases can be used for software hazard analysis for FPGA software requirements specifications. Well-defined guide phrases provide an opportunity to explore various aspects of hazard analysis, as shown in the case study. We performed additional analysis on the case study results as comparing them to the results of HAZOP analysis of the same software to identify the applicability of the proposed process. Firstly, we classified analysis aspects for each requirement element in the case study results. The left and middle columns in ⟨Table 3⟩ shows the classification results by analysis aspects when performing requirement hazard analysis using the proposed process and the original NUREG/CR-6430 process, respectively.

"*Deviation analysis*" indicates that a requirement item is used to analyze its deviations by guide phrases directly, whereas, "*Cause*" indicates that an requirement item causes deviations in other requirements. According to the table above, many requirement points, which are related to hardware aspects, can be analyzed deviations directly. It shows that the proposed process and guide phrases may be helpful for analyzing deviations in FPGA software requirement specifications directly. ⟨Table 3⟩ also compares analysis aspects between software hazard analysis results from using the proposed process, original NUREG/CR-6430 and HAZOP analysis. The HAZOP analysis result in the table were generated using the HAZOP technique and general guide words [1], which have been introduced in several studies and books [34].

HAZOP analysis with general guide words focuses on the functionality of the software itself, therefore some requirement points, that are directly written in the specification of a software function, are analyzed deviations directly. Other points that are classified to only cause in the table are related to hardware which supports the software functionality indirectly. This results show that the perspective of hazard analysis can be changed by changing the analysis approach and guide words/phrases. If guide phrases contain appropriate items/contents for the characteristics of the target system, they can be useful for analysis. The process and guide phrases proposed in this paper may be more suitable for the requirements hazard analysis of FPGA software, when analysts need to analyze each software requirement directly on a hardware-based platform.

4.4. More considerations on requirement hazard analysis of FPGA software

The step of identifying software-responsible hazards, which is in part of the original NUREG/CR-6430 process, allows one to assign software criticality level. However, this step cannot be directly applied to FPGA software requirement specifications which are defined in module units, because a gap exists between software elements and hazards of software/hardware/system. Therefore, we instituted an additional step to bridge this gap, but the connections which are shown in the ⟨Table A2⟩ do not provide exact one-to-one matching between hazards. According to system theory, a system is not a simple combination of components [35] and consists of a hierarchical structure with interactions.

We believe that a more structural approach may be realized by considering the hierarchical structure of a system. A chain of cause-consequence, hazard propagations between different level of a system, and traceability-based hazard identification are conceivable elements in the hierarchy. For instance, the system generally can be divided into a hierarchical structure such as sub-system, component, hardware, and software. Nuclear systems also have hierarchical structures. An example hierarchical structure of NPP is discussed below. If information regarding hazards in the software requirement specifications phase is well-organized, such as the information provided in ⟨Table A2⟩, such information can behelpful for analyzing software hazards efficiently. Traceability between system hierarchy structures is also a conceivable method for supporting preliminary hazard analysis.

- SW - SRS, SDD, Code
- PLC, FPGA
- Bistable Processor
- RPS/Plant Protection System
- NPP system

Although the requirement hazard analysis of FPGA software has several differences in terms of analysis aspects, it is not completely different from the hazard analysis of common software. Deriving consequences, effects, and hazards from deviations by guide phrases is similar substances, for example, the hazards caused by sensor deviations are the same as software failures caused by misreading value. However, software hazard analysis of later development phases should account for the hardware-based development of FPGA. For example, FPGA development proceeds to design, implementation, synthesis, and P&R hazard analysis of after the implementation is entirely different from existing hazard analysis method. There may be need to another approach of hazard analysis for FPGA software. Various studies on reliability analysis for FPGA may be useful for the hazard analysis of such later development phases.

5. Conclusions and future work

This paper proposes a refined process and guide phrases for the hazard analysis of FPGA software requirement specifications. The proposed process extends the hazard analysis process of the NUREG/CR-6430 to incorporate the hardware features of FPGA software. The proposed process consists of six steps for requirement hazard analysis and extended guide phrases to handle the circuit and memory aspects of FPGA software. We performed a practical application of the proposed

Table A1
Hazard lists of processor module of the DFLC-N.

No.	Preliminary hazard list	Hazard category (H/M/L)
1. power	a. loss of power	H
	b. loss of current	H
	c. over-voltage	H
2. Physical effects of internal/external	a. Conflagration	H
	b. Physical impact (*e.g. earthquake*)	H
	c.Radioactivity	H
3. Calculation error	a. Application error	H
	b. Memory error - stuck	H
	c. Response time error	H
	d. Monitoring function error	H
	e. Transmit capacity error	H
	f. Invalid LED operation	M
	g. Network failure	H
4. Operator	a. Operation error/missed bypass by operator	M

Table A2
Software responsible hazards in PHL from software functionality-level hazards.

No.	Hazards of software concerns	PHL with concerns	Hazard category (H/M/L)
1	PM software cannot send qualified information of its status	f. Invalid LED operation 4. Operator d. Monitoring function error	M
2	PM software transmit incorrect signal	a. Application error d. Monitoring function error f. Invalid LED operation c. Response time error	H
3	PM software transmit signal when the signal is not occurred	a. Application error c. Response time error d. Monitoring function error	H
4	PM software cannot transmit signal when the signal is occurred	a. Application error c. Response time error d. Monitoring function error	H
5	PM software transmit invalid length signal	a. Application error c. Response time error	H
6	PM software transmit incorrect data	a. Application error	H

process in a case study on a prototype version of an FPGA-based controller operating in Korea. This paper also discussed results of the hazard analysis about comparing analysis aspects and hierarchical structure of the system. The proposed process and guide phrases are efficient for analyzing the hazards of FPGA software requirement specifications. It may also be applied to FPGA software in other domains when software requirement specifications are developed using an appropriate development process. If the extension of guide phrases is considered for other PLDs, the proposed process may be a useful approach to hazard analysis. We are now planning to develop software hazard guidelines, such as templates for generalizable hazard analysis. In the future, we also plan to complement our integrated development process by incoporating asoftware hazard analysis process[36] and an integrated safety analysis process.

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.

CRediT authorship contribution statement

Sejin Jung: Methodology, Validation, Visualization, Writing - original draft, Writing - review & editing. **Junbeom Yoo:** Conceptualization, Methodology, Validation, Writing - original draft, Writing - review & editing. **Young-Jun Lee:** Resources, Validation, Writing - review & editing.

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2017R1D1A1B03030065) and Next-Generation Information Computing Development Program through the National Research Foundation (NRF) of Korea funded by the Ministry of Science, ICT (NRF-2017M3C4A7066479).

Appendix A. Hazard Analysis Results of FPGA Software Requiremetns Specification for DFLC-N

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.ress.2020.107029

Table A3
A part of the analysis results of FPGA software requirements of CLFPM01 about sensor.

No.	Qualities	Aspects	Item	Function/Purpose	Parameter	Guide Phrases	Deviation	Consequence	Hazard	Risk	Hazard on SW PHL	Hazard on PM PHL
1	Accuracy	Sensor	9.2 Operating voltage monitoring function hardware	Monitoring the operating voltage of the FPGA module and sending the state value to operating voltage monitoring function.	9.2.3.2 swr4 operating voltage monitoring	Stuck at all zeroes	All received data from sensor generates stuck-at zero	Monitoring function receives all 0 regardless of the current state. And it can change the state to err when zero value continues with ten cycles	Display the error state value when operating voltage has normal value	M	1.	3.f., 4., 3.d.
2						Stuck at all ones	All received data from sensor generates stuck-at one	Monitoring function receives all 1 regardless of the current state. When this fault continues, error state of the operating voltage is not received from the sensor	Display the normal state value when FPGA module receives err state voltage of the operation	M	1.	3.f., 4., 3.d.
3						Stuck elsewhere	All received data from sensor generates stuck-at fault	Receiving an opposite state value	The operating voltage display shows different state of the current	M	1.	3.f., 4., 3.d.
5						Below minimum range	Monitoring function receives the state value of the above maximum	It does not occur in practice, because sensor sends a one-bit data	-	-	-	-
6						Above maximum range	Monitoring function receives the state value of the above maximum	It does not occur in practice, because sensor sends a one-bit data	-	-	-	-
7						Within range, but wrong	Monitoring function receives the error value from the sensor	Receiving an opposite state value	The operating voltage display shows different state of the current	M	1.	3.f., 4., 3.d.
8						Physical units are incorrect	Physical fault occurs in monitoring sensor	Monitoring function does not receive any state value of the operating voltage or receive an error value	State value for the display does not exist in	M	1.	3.f., 4., 3.d.
9						Wrong data type or data size	Monitoring function receives wrong type of sensor data	It does not occur in practice, because sensor sends a one-bit data	-	-	-	-

Table A4
A part of the analysis results of FPGA software requirements of CLFPM01 about circuit.

No.	Qualities	Aspects	Item	Function/Purpose	Parameter	Guide Phrases	Deviation	Consequence	Hazard	Risk	Hazard on SW PHL	Hazard on PM PHL
1	Accuracy	Circuit	9.1 Reset and clock signal generation function	Generating clock signal	Generating clock signal	Stuck elsewhere	Stuck fault occurs in circuits which performs software function	Generating incorrect period clock signal	Timing error issues in memory and registers by incorrect clock period	H	6.	3.a.
2									Output cycle changes to incorrect by clock	H	3., 4.	3.a., 3.c., 3.d
3			9.11 State expression function	Showing the current state of the system with LED	LED lighting circuit	Stuck elsewhere	Stuck fault occurs in circuits which performs software function	State expression LED shows the state of stuck	LED function expresses the state value different from current state	H	6.	3.a.
4			Other functions of the requirements	Each function of the requirements item	Calculating circuits for each function	Stuck at all zeroes	All values are stuck to zero	Wrong value is reached to functional unit. Shared memory in function module saves wrong value	Memory surveillance function exist to prevent these situation	-	-	-
5						Stuck at all ones	All values are stuck to one	Wrong value is reached to functional unit. Shared memory in function module saves wrong value	Memory surveillance function exist to prevent these situation	-	-	-
6						Stuck elsewhere	Some values are stuck	Wrong value is reached to functional unit. Shared memory in function module saves wrong value	Memory surveillance function exist to prevent these situation	-	-	-
7	Functionality		All of elements in requirements	Each function of elements	Function operation	Function is not initialized properly before being executed	What happened if each function is not initialized first	Execution with uninitialized memory, option, register	Initialization requirements already exist	-	-	-
8			9.1 Reset and clock signal generation function	Generating clock signal	Generating clock signal	Function is not carried out as specified	What happened if software are not performed to its function	It generates incorrect cycle clocks	Timing error issues in memory and registers by incorrect clock period	H	6.	3.a.
9									Output cycle changes to incorrect by clock	H	3., 4.	3.a., 3.c., 3.d
8			Each elements of 9.3 ~ 9.10 in requirements spec.	Each function of elements	Function operation	Function is not carried out as specified	What happened if software are not performed to its function	The incorrect results are written to memories in FPGA module	Memory surveillance function exist to prevent these situation	-	-	-
8			9.7 Input/output data transmit/receive and surveillance (9.8, 9.9 also)	Data transmit/ receives by network channel	9.9.4.1 SWR 17	Function is not carried out as specified	What happened if software are not performed to its function by ambiguous requirements	Unintended software design is generated by ambiguous requirements	Unintended signal transmit/receive by unintended design results	H	2.	3.a., 3.d., 3.f., 3.c.

Table A5
A number of hazards from hazard analysis of the case study.

No.	Software contributable hazards in PHL	Number of hazard	Aspect category of guide phrases
1	PM software cannot send qualified information of its status	8	Sensor, circuit, calculator, and functionality
2	PM software transmit incorrect signal	4	Functionality, timing
3	PM software transmit signal when the signal is not occurred	9	Circuit, input & output, calculator, timing, functionality
4	PM software cannot transmit signal when the signal is occurred	9	Circuit, input & output, calculator, timing, functionality
5	PM software transmit invalid length signal	1	Timing
6	PM software transmit incorrect data	6	Circuit, input & output, calculator, timing, functionality

References

[1] Jung S, Kim E-S, Yoo J, Keum JY, Lee J-S. Hazard analysis of software requirements specification for process module of FPGA-based controllers in NPP. Transactions of the Korean nuclear society autumn meeting, Gyeongju, Korea. 2016.
[2] Design of Instrumentation and control systems for nuclear power plant. Tech. Rep.. International Atomic Energy Agency; 2016.
[3] Nuclear power plants - instrumentation and control important to safety - general requirements for systems (IEC 61513). Tech. Rep.. International Electrotechnical Commission (IEC); 2011.
[4] Software safety hazard analysis (NUREG/CR-6430). Tech. Rep.. United States Nuclear Regulatory Commission (NRC); 1995.
[5] Leveson NG. Software safety in embedded computer systems. Commun ACM 1991;34(2):34–46.
[6] McDermid J. Software hazard and safety analysis. International symposium on formal techniques in real-time and fault-tolerant systems. Springer; 2002. p. 23–34.
[7] Software reliability and safety in nuclear reactor protection systems (NUREG/CR-6101). Tech. Rep.. United States Nuclear Regulatory Commission (NRC); 1993.
[8] IEEE Standard for Software Safety Plans (IEEE 1228). Tech. Rep.. Institute of Electrical and Electronics Engineers (IEEE); 1994.
[9] Nuclear power plants - instrumentation and control important to safety - development of HDL-programmed integrated circuits for systems performing category A functions (IEC 62566). Tech. Rep.. International Electrotechnical Commission (IEC); 2012.
[10] Application of field programmable gate arrays in instrumentation and control systems of nuclear power plants. Tech. Rep.. IAEA (International Atomic Energy Agency); 2016.
[11] NUREG/CR-7006 : review guidelines for field-programmable gate arrays in nuclear power plant safety systems. Tech. Rep.. U.S. Nuclear Regulatory Commision (NRC); 1996.
[12] da Silva Neto AV, Vismari LF, Gimenes RAV, Sesso DB, de Almeida JR, Cugnasca PS, et al. A practical analytical approach to increase confidence in PLD-Based systems safety analysis. IEEE Syst J 2017;12(4):3473–84.
[13] Zheng D, Wang Y, Xueyi Z. The methods of FPGA software verification. 2011 IEEE international conference on computer science and automation engineering. 3. 2011. p. 86–9.
[14] Karimi MM, Anzagira A, Taylor W, Nelson J, Osareh A. Component failure analysis (CFA): a new method for implemented FPGA design failure analysis. SoutheastCon 2018, IEEE. IEEE; 2018. p. 1–8.
[15] Jung S, Choi JP. Predicting system failure rates of SRAM-based FPGA on-board processors in space radiation environments. Reliabil Eng Syst Saf 2019;183:374–86.
[16] Hoque KA, Mohamed OA, Savaria Y. Dependability modeling and optimization of triple modular redundancy partitioning for SRAM-based FPGAs. Reliabil Eng Syst Saf 2019;182:107–19.
[17] KAERI. Digital FPGA Logic Controller - Nuclear(NTIP-FLC-SRS201). Tech. Rep.. Korea Atomic Energy Research Institute; 2016. In Korean
[18] Nuclear power plants - Instrumentation and control systems important to safety - Software aspects for computer-based systems performing category A functions (IEC 60880). Tech. Rep.. International Electrotechnical Commission (IEC); 2006.
[19] Nuclear power plants - Instrumentation and control important to safety - Hardware design requirements for computer-based systems (IEC 60987). Tech. Rep.. International Electrotechnical Commission (IEC); 2007.
[20] Hazard and operability studies (HAZOP studies) - application guide (IEC 61882). Tech. Rep.. International Electrotechnical Commission (IEC); 2016.
[21] Park G-Y, Lee J-S, Cheon S-W, Kwon K-C, Jee E, Koh KY. Safety analysis of safety-critical software for nuclear digital protection system. International Conference on computer safety, reliability, and security September 18–21, Germany. 2007. p. 148–61.
[22] KAERI. Safety analysis report of reactor protection system software requirements specification (KNICS-RPS-SVR122). Tech. Rep.. Korea Atomic Energy Research Institute; 2007. Rev.01
[23] Vismari LF, Camargo JB, de Almeida JR, da Silva Neto AV, Gimenes RAV, Cugnasca PS. A practical analytical approach to increase confidence in software safety arguments. IEEE Syst J 2017;11(4):2072–83.
[24] et. al. ML. est Practice VHDL coding standards for DO-254 Programs. Tech. Rep.. DO-254 User Groups; 2010.
[25] Benites LA, Benevenuti F, Oliveira ÁBD, Kastensmidt FL, Added N, Aguiar VA, et al. Reliability calculation with respect to functional failures induced by radiation in tmr arm cortex-m0 soft-core embedded into SRAM-based fpga. IEEE Trans Nucl Sci 2019;66(7):1433–40.
[26] McNelles P, Zeng ZC, Renganathan G, Lamarre G, Akl Y, Lu L. A comparison of fault trees and the dynamic flowgraph methodology for the analysis of fpga-based safety systems part 1: reactor trip logic loop reliability analysis. Reliabil Eng Syst Saf 2016;153:135–50.
[27] McNelles P, Renganathan G, Zeng ZC, Chirila M, Lu L. A comparison of fault trees and the dynamic flowgraph methodology for the analysis of fpga-based safety systems part 2: theoretical investigations. Reliabil Eng Syst Saf 2019;183:60–83.
[28] Li W, Hao Z. FPGA software testing process management. 2015 IEEE international conference on grey systems and intelligent services (GSIS). IEEE; 2015. p. 600–3.
[29] Maerani R, Mayaka JK, Jung JC. Software verification process and methodology for the development of FPGA-based engineered safety features system. Nucl Eng Des 2018;330:325–31.
[30] Ahmed I, Jung J, Heo G. Design verification enhancement of field programmable gate array-based safety-critical I&C system of nuclear power plant. Nucl Eng Des 2017;317:232–41.
[31] Lu J-J, Hsu T-C, Chou H-P. System assessment of an FPGA-based RPS for ABWR nuclear power plant. Prog Nucl Energy 2015;85:44–55.
[32] Huang J, You J-X, Liu H-C, Song M-S. Failure mode and effect analysis improvement: asystematic literature review and future research agenda. Reliabil Eng Syst Saf 2020;199.
[33] McNelles P, Zeng ZC, Renganathan G, Chirila M, Lu L. Failure mode taxonomy for assessing the reliability of field programmable gate array based instrumentation and control systems. Ann Nucl Energy 2017;108:198–228.
[34] Ericson CA. Hazard analysis techniques for system safety. John Wiley & Sons; 2015.
[35] Leveson NG. Safeware: system safety and computers. Addison Wesley; 1995.
[36] Kim E-S, Lee D-A, Jung S, Yoo J, Choi J-G, Lee J-S. NuDE 2.0: a formal method-based software development, verification and safety analysis environment for digital I&Cs in NPPs. J Comput Sci Eng 2017;11(1):9–23.