Nuclear Engineering and Technology 50 (2018) 570–581

Contents lists available at ScienceDirect

Nuclear Engineering and Technology

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

Original Article

Development of simulation-based testing environment for safety-critical software

Sang Hun Lee [a], Seung Jun Lee [b], Jinkyun Park [c], Eun-chan Lee [d], Hyun Gook Kang [a, *]

[a] Department of Mechanical Aerospace and Nuclear Engineering, Rensselaer Polytechnic Institute (RPI), 110 8th Street, Troy, NY, 12180, USA
[b] School of Mechanical, Aerospace and Nuclear Engineering, Ulsan National Institute of Science and Technology (UNIST), 50 UNIST-gil, Ulsan, 44919, Republic of Korea
[c] Integrated Safety Assessment Division, Korea Atomic Energy Research Institute (KAERI), 111 Daedeok-daero, 989beon-gil, Yuseong-gu, Daejeon, 34057, Republic of Korea
[d] Korea Hydro & Nuclear Power Co., Ltd., 1655 Bulguk-ro, Gyeongju-si, Gyeongsangbuk-do, 38120, Republic of Korea

ARTICLE INFO

Article history:
Received 30 January 2018
Received in revised form
28 February 2018
Accepted 28 February 2018
Available online 27 March 2018

Keywords:
Digital Instrumentation and Control System
Nuclear Power Plant
Software Reliability Quantification
Software Testing

ABSTRACT

Recently, a software program has been used in nuclear power plants (NPPs) to digitalize many instrumentation and control systems. To guarantee NPP safety, the reliability of the software used in safety-critical instrumentation and control systems must be quantified and verified with proper test cases and test environment. In this study, a software testing method using a simulation-based software test bed is proposed. The test bed is developed by emulating the microprocessor architecture of the programmable logic controller used in NPP safety-critical applications and capturing its behavior at each machine instruction. The effectiveness of the proposed method is demonstrated via a case study. To represent the possible states of software input and the internal variables that contribute to generating a dedicated safety signal, the software test cases are developed in consideration of the digital characteristics of the target system and the plant dynamics. The method provides a practical way to conduct exhaustive software testing, which can prove the software to be error free and minimize the uncertainty in software reliability quantification. Compared with existing testing methods, it can effectively reduce the software testing effort by emulating the programmable logic controller behavior at the machine level.

1. Introduction

With a shift in technology to digital systems as analog systems are approaching obsolescence and because of functional advantages of digital systems, existing nuclear power plants (NPPs) have begun to replace analog instrumentation and control (I&C) systems, while new plant designs fully incorporate digital systems [1]. Compared with the analog I&C systems, the digital systems provide advanced performance in terms of accuracy and computational capabilities and have potential for improved capabilities such as fault tolerance and diagnostics [2]. However, the use of microprocessor-based digital systems in NPP safety I&C systems has triggered a big challenge in incorporating their characteristics into the probabilistic risk assessment (PRA) model of NPPs used to evaluate the digital system reliability and its risk effect on the NPP safety.

A comprehensive review of the risk issues of digital I&C systems that should be considered in the NPP PRA model has been conducted by Kang and Sung [3]. Among various issues, estimation of the software failure probability was identified as one of the important factors in terms of NPP risk, and a sensitivity study was conducted to analyze the relationship between the system reliability and the software failure probability for a typical digital reactor protection system (RPS). A report on operation and maintenance experience described how software error was a major cause of digital system failures during 1990–1993 [4]; during this time, 30 failures were caused by software error, compared with nine random component failures, among a total of 79 digital system failure events. Several reports also stated the importance of software-based errors, which are considered to be a credible source of the common-mode or common-cause failure of the digital systems [5,6], that can lead to significant safety threats of NPPs. Therefore, quantification of software reliability plays a very

* Corresponding author.
E-mail addresses: lees35@rpi.edu (S.H. Lee), sjlee420@unist.ac.kr (S.J. Lee), kshpjk@kaeri.re.kr (J. Park), eclee@khnp.co.kr (E.-c. Lee), kangh6@rpi.edu (H.G. Kang).

https://doi.org/10.1016/j.net.2018.02.007

important role in ensuring the safety of NPPs, and the verification of a very low software failure probability is crucial for the PRA of a digitalized NPP.

In response, quantitative software reliability methods such as the software reliability growth model (SRGM), Bayesian belief network (BBN) model, and test-based method have been proposed and adopted in the nuclear field. The SRGM method [7] has been widely used in the software engineering field to assess software reliability by estimating the increment of reliability as a result of fault removal over time. By applying a software reliability model and using existing software failure data to estimate its parameters, the software reliability is assessed and predicted based on extrapolation. However, the SRGM method was found to be not applicable to safety-critical software [8] because of its high sensitivity in estimating the number of faults to time-to-failure data and the rare software failure sets in NPP safety-critical applications that are developed under a strict development and verification and validation life cycle.

The BBN method has also been extensively applied to estimate the software reliability of NPP safety systems [9,10]. The method models and aggregates disparate information about the software, such as software failure data and the quality of software life cycle activities. However, the limitations of the BBN method in quantifying the software reliability include the need to develop a credible BBN model, which requires identification of a complete and independent set of software attributes and the qualification of experts to estimate model parameters and qualitative evidence. Owing to those limitations, the uncertainty in the estimated software residual faults and failure probability from the BBN model may be very large, which makes it difficult to verify the very low failure probability of 10^{-4} to 10^{-5} required for safety-critical safety integrity level (SIL) level 4 software [11].

The test-based approach is another method that can be used to assess the reliability of NPP safety-critical software; this method applies standard statistical methods to the results of software testing, in a manner similar to that in which the reliability of hardware components is analyzed [12]. The studies relevant to the test-based approach conducted in the nuclear field are mainly divided into two testing methods: 1) black-box testing methods [13–15] and 2) white-box testing methods [16,17]. The black-box testing methods consider a software program as a black box, take random samples from its input space, determine if the outputs are correct, and use the results for statistical analyses to estimate the software reliability. However, because the black-box testing methods are conducted without knowledge on the program's internal logic or structure, the limitations of black-box testing include limited coverage and completeness of the test cases [18]. On the other hand, the white-box testing methods have an advantage in that they take into consideration the internal structures of the software; so, the tests are performed to ensure that certain parts of the software are functioning correctly, with full coverage. However, because the white-box testing methods aim to test all possible paths and nodes of the software, the number of tests that must be carried out for exhaustive testing is often very large [17] when the operational profile of the software encountered in an actual use is neglected. Therefore, an efficient and effective software testing framework for the safety-critical software used in NPP digital I&C systems must be developed to prove the correctness of the software and further quantify the software reliability based on software test results.

The objective of this study is to develop a simulation-based software test bed for white-box testing of NPP safety-critical software. The test bed is developed by emulating the microprocessor architecture of a safety-critical programmable logic controller (PLC) used in an NPP digital I&C system and capturing its behavior at each machine instruction line while the software executes its dedicated safety function. The effectiveness of the proposed software testing framework is demonstrated with the safety-critical trip logic software of a fully Integrated Digital Protection System-Reactor Protection System (IDiPS-RPS), developed under the Korea Nuclear Instrumentation & Control Systems (KNICS) project [19]. Given specific software input and internal states, the proposed method can effectively reduce software testing efforts by emulating the software behavior at a machine language level; this is in contrast to existing black-box testing, which uses trajectory inputs for software testing. The test results of safety-critical software from the suggested method can be used to support the software reliability quantification of NPP digital I&C systems and can be applied to the PRA of an NPP to analyze the effect of software failure on the digital system availability or the NPP risk.

2. Target system

In this section, an overview of the NPP safety-critical digital I&C system in which the test bed is developed is provided. The basic architecture and operation mechanism of the safety-grade PLC used in the target system are reflected in the test bed.

2.1. IDiPS-RPS configuration

The IDiPS-RPS is a digitalized RPS developed in the KNICS project for newly constructed NPPs and for upgrading existing analog-based RPS [19]. It has the same function as an analog RPS to automatically generate a reactor trip signal and engineered safety feature actuation signals whenever demand comes. Fig. 1 illustrates the architecture of the IDiPS-RPS, which has four redundant channels of processors for its dedicated safety functions.

As a part of the IDiPS-RPS, the bistable processors (BPs) determine the trip state by comparing the process variables measured from the plant sensors with the predefined pretrip or trip setpoints; coincidence processors (CPs) generate a final hardware-actuating trip signal by voting logic. The processors are configured based on the safety-grade PLC platform (POSAFE-Q) [20], and the function of each processor is implemented as software in the PLC platform.

2.2. POSAFE-Q architecture

The POSAFE-Q consists of various modules, such as a processor module, communication module, and I/O module [21]. The processor module consists of a TI C32 digital signal processor, central processing unit (CPU), and various types of memory, such as flash memory and static random access memory (SRAM). The application programs in the IDiPS-RPS, such as BP trip logic and CP voting logic, are downloaded into the memory embedded within the processor module. The application software is developed based on function block diagram and ladder diagram (FBD/LD) programming. In the implementation, the FBD/LD programs are compiled to machine instruction codes, which are loaded into the PLC memory area and executed by the PLC microprocessor [22]. Fig. 2 shows the safety-grade PLC compile procedure used to generate the machine code from the user application program, written in FBD/LD language.

3. Test bed development

In this section, the test bed development processes are described. The microprocessor architecture and operation mechanisms of the safety-grade PLC are emulated in the simulated environment. The methods of test bed verification are also described.

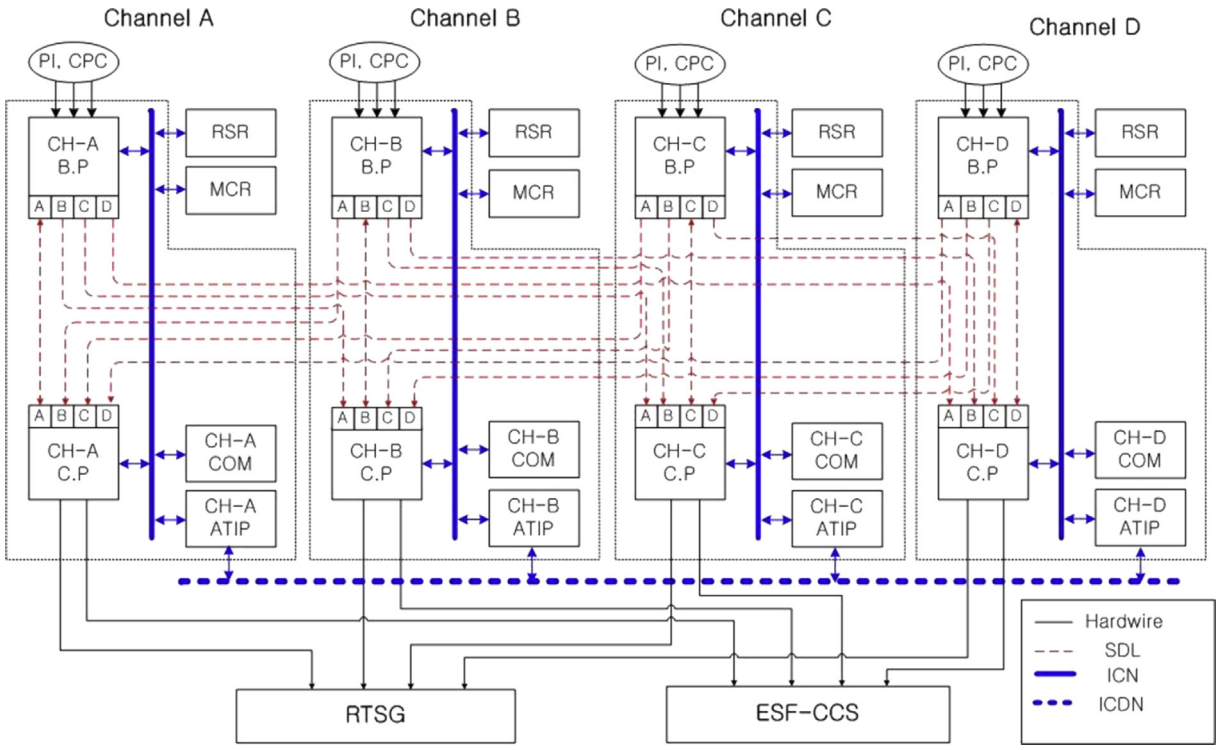

Fig. 1. Block diagram of IDiPS-RPS.
IDiPS-RPS, Integrated Digital Protection System-Reactor Protection System.

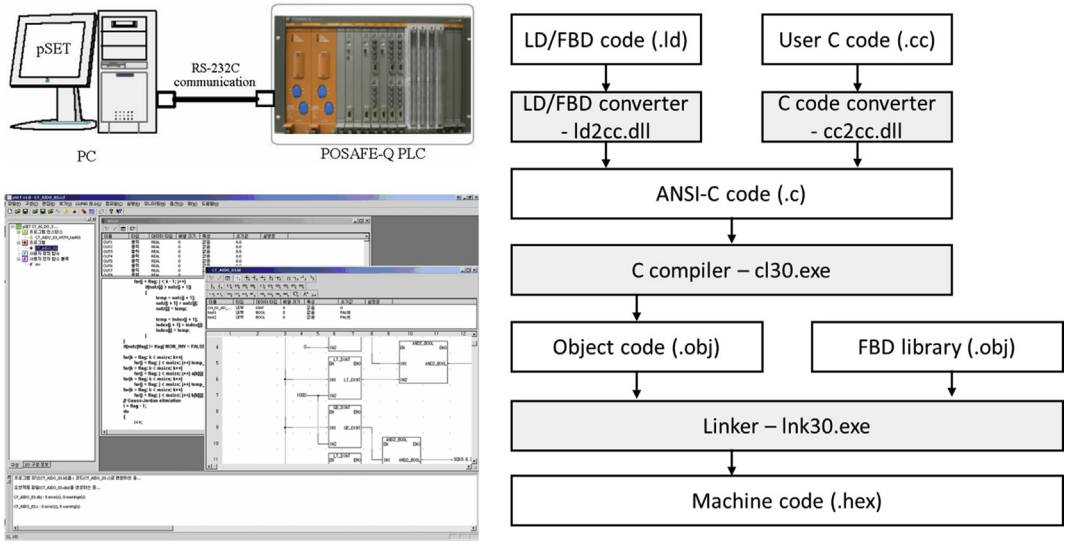

Fig. 2. FBD/LD compile procedure of safety-grade PLC [22].
FBD/LD, function block diagram and ladder diagram; PLC, programmable logic controller.

3.1. Development of software test bed

The most fundamental characteristic of PLC operation is the cyclic operation mode [23]. Each iteration of the cyclic operation of the PLC, called a scan cycle, consists of several operation stages that are sequentially repeated. After checking its own status, the PLC will copy all the software input values into the RAM, where input/internal/output variable data and user programs are stored. Then, the CPU executes the application program implemented in the PLC memory map, and the output of the software is updated based on the execution result. In each scan cycle, the aforementioned operations are repeated at a fixed interval of time called a scan time.

In this study, to simulate the software behavior given the states of software input and internal variables and to check whether the correct output is generated by the target software application program, a software test bed is developed that captures both the internal (CPU and memory architecture) and external (states of program input and output variables) aspects of the PLC scan cycle. The test bed is developed in C code by emulating the PLC microprocessor architecture, such as the CPU register and memory map [24] and the assembly language instructions. Fig. 3 shows an overview of the developed test bed structure. The test bed is composed of four major modules; the description of each module is as follows:

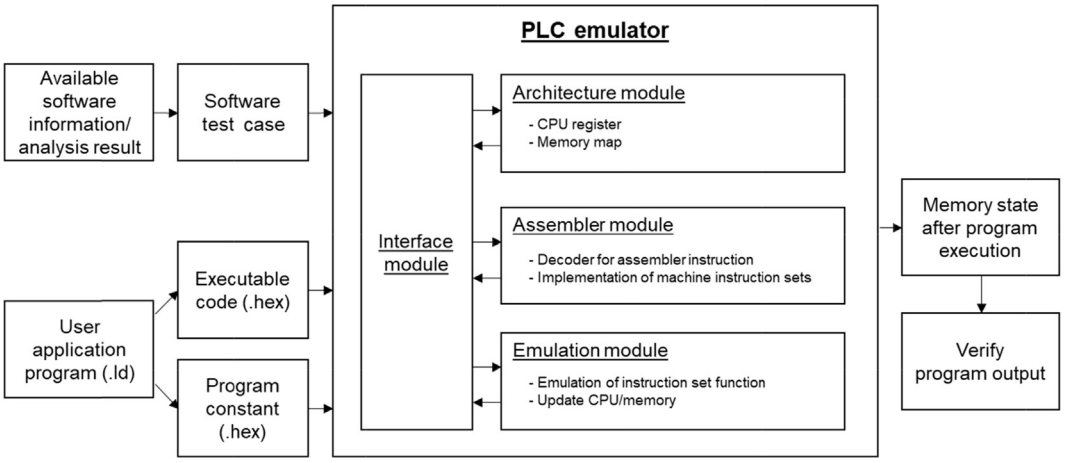

Fig. 3. An overview of the simulation-based test bed for safety-critical PLC software testing.
CPU, central processing unit; PLC, programmable logic controller.

- 1) *Architecture module*: The components of the safety-grade PLC microprocessor consist of CPU register files, such as 40-bit extended registers, 32-bit auxiliary registers, and other registers, and the memory units that are accessible to the CPU, which contains the total memory space of 16-Mbyte 32-bit words. Within the 16-Mbyte word address space, the program, data, and I/O space are contained, allowing the program code or data of the user application software to be stored in the memory map. In the test bed, the major components of the microprocessor are emulated to capture the state of the CPU instruction line of the application software. To simulate reading or writing of the values from/to the memory space, several different memory addressing modes, including the register, direct, indirect, and immediate addressing modes, are implemented.
- 2) *Assembler module*: The instruction set of the safety-grade PLC microprocessor contains a total of 113 instructions. All instructions are defined as a single machine word long (32-bit), and most instructions require one cycle to be executed. The categories of instruction sets include the instructions for load and store, 2-/3-operand arithmetic, program control, interlocked, and parallel operations. The syntax of each instruction set contains its specific 9-bit opcode, the addressing mode, and operands. To emulate the execution of application software in the 32-bit binary format, the functions and syntaxes of instruction sets are implemented in the test bed.
- 3) *Emulation module*: Based on the instruction set decoded from the binary program file by the *Assembler module*, the operands of the instruction set from the register files are read, and the set performs its specific operation. The operation result is written to the CPU registers or to a specific memory element, depending on the operands and addressing modes. The CPU contexts such as the system stack, the condition flags stored in the CPU status register, and the data in the memory area are updated at every machine instruction line.
- 4) *Interface module*: The *Interface module* provides an interface between each module. For example, the instruction set decoded from the *Assembler module* is transferred to the *Emulation module* to conduct its specific operation. In addition, the result of instruction set execution by the *Emulation module* is updated to the CPU register and the memory elements emulated in the *Architecture module*.

The test bed is executed based on four basic operation processes of the PLC microprocessor: 1) fetch, 2) decode, 3) read, and 4) execute [24]. In the fetch phase, the executable code, which is uploaded to the memory map, is fetched from the *Architecture module*. In the decode phase, the fetched executable code, in binary form, is decoded into a specific instruction set by the *Assembler module*. In the read phase, the address generation is performed, and the operands are read from the CPU registers. In the execute phase, the operation of the decoded instruction set is performed by the *Emulation module,* and the operation results are stored in the CPU register or the memory. If necessary, the registers that represent the status of the microprocessor, such as stack management, are updated during the execute phase.

To conduct software testing using the developed test bed, the program executable code compiled from the user application FBD/LD program and the program constant file, which contains the memory map of the input (e.g., pressure, water level in NPP) and the internal variable (e.g., counter, test parameters) used in the application program, are loaded into the test bed. Then, the software test cases are uploaded to the memory area emulated in the *Architecture module*. After all machine instruction lines of the application program are executed, the final status of CPU registers and memory map is automatically saved as an output file for every software test case. By checking the specific memory area that corresponds to a dedicated safety function of the application program, such as the trip signal, generated output files are used to verify whether correct output is generated given the test case.

3.2. Verification of software test bed

To validate the developed simulation-based software test bed, unit testing and functional testing were conducted for the instruction sets emulated in the test bed.

3.2.1. Unit testing of software test bed

Unit testing is a software testing method in which the individual units of the source code, such as the associated functions, are tested to determine whether each unit of the code generates the precise expected output [25]. In this study, the unit test cases for every PLC microprocessor machine instruction set were developed and used to verify the correctness of the instruction set operations emulated in the test bed. Fig. 4 shows the procedure of software test bed verification using the instruction unit test cases.

The instruction unit test cases for test bed verification were developed in consideration of all possible addressing modes and operands of the instruction sets based on the specification documents of the safety-grade PLC microprocessor [24] and converted into an equivalent 32-bit binary representation. The initial states

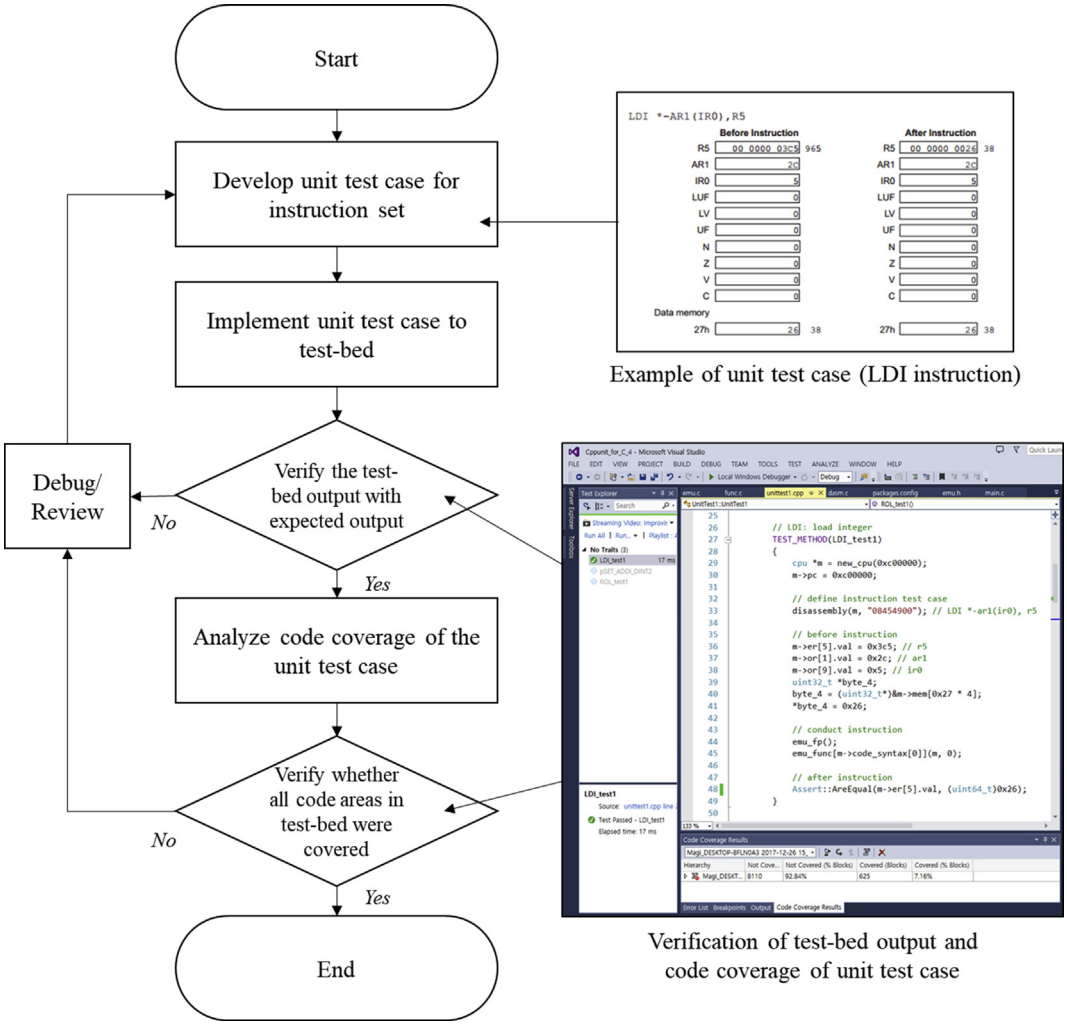

Fig. 4. Verification of the software test bed with instruction set unit test cases.
LDI, load integer.

(before instruction execution) of the CPU register and memory map are defined based on the unit test cases, and the final state of the microprocessor (after instruction execution) is captured to verify the result by comparing it with the expected final state of the CPU register or memory element. The code coverage analysis result was also used to verify that all source code areas of the instruction set operation in the test bed were correctly executed with full coverage.

3.2.2. Functional testing of software test bed

Functional testing is a type of software testing in which the source code is tested by checking the correctness of the program via comparison of the results for a given specific input [26]. In this study, the standard FBDs defined in IEC61131-3 [27], such as addition (ADD), logical conjunction (AND), and logical equality (EQ) function blocks, were used to test the functionality and correctness of the test bed by verifying the generated output in the test bed with the expected output of each function block.

The test cases are developed by modeling the standard FBDs, including their input and output ports, and generating an equivalent 32-bit binary program using the digital signal processor compiler. Then, the program file and constant file, which includes the memory map of the input ports and internal variables used in the modeled FBD, are loaded into the test bed. The final state of the output port (after program file execution) is then checked to verify whether the output generated by the test bed is the same as that expected from the FBD model. Fig. 5 shows the procedure of test bed verification using the standard FBDs.

4. Case study

As a case study, the proposed software testing method was applied to the target safety-critical software program of KNICS IDiPS-RPS. The test cases were developed based on the profile of the software input and internal variables. For each test case, the test results are generated by capturing the final state of the output variable after the application program is executed in the developed test bed.

4.1. Target safety-critical software

In the IDiPS-RPS system, the BP compares the process variables transmitted from the measurement instruments in the NPP with the predefined trip setpoints, and the CPs perform two-out-of-four voting logic with the signals transmitted from the four redundant channels of the BP to determine whether the system should generate a trip signal. The function of each module is implemented as a software logic in the PLC memory map in binary format. Among the BP software modules, 19 modules for the trip logics are defined,

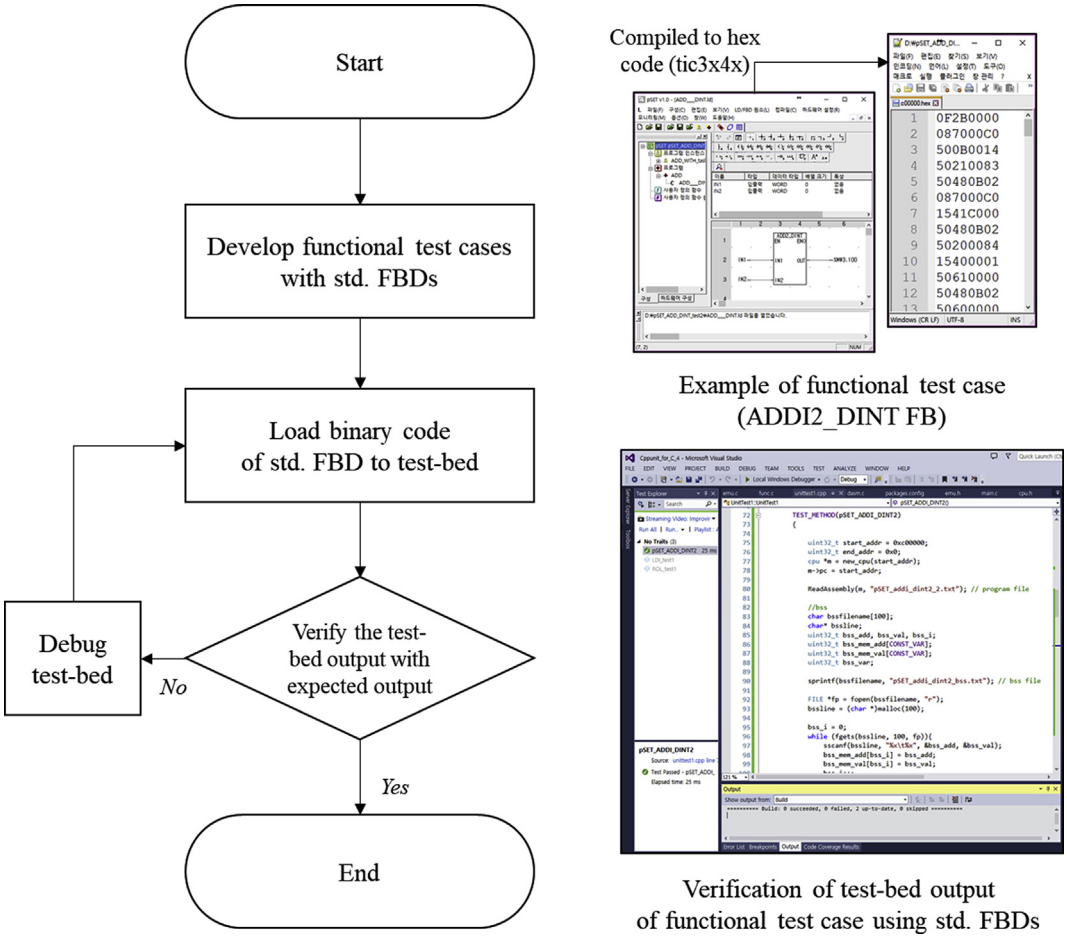

Fig. 5. Verification of the software test bed with standard FBD/LD test cases.
FBD/LD, function block diagram and ladder diagram.

and the process variable of each module is compared against its predefined threshold values [28]. These trip logics are categorized into four types: 1) fixed set-point trip (10 modules); 2) variable set-point trip (3 modules); 3) manual reset trip (3 modules); and 4) digital trip (3 modules). Table 1 shows the BP trip logics of the IDiPS-RPS [29]; the description of each trip setpoint (TSP) type is as follows:

- Fixed set-point logic: As the process input signal rises or falls through the fixed pretrip or trip setpoint, the BP generates the

Table 1
KNICS IDiPS-RPS BP application software modules [29].

Software modules	Description	OB	TSP type
VA_OVR_PWR_HI Trip (_1_)	Variable Over Power Hi Trip	—	RR, Rising
LOG_PWR_HI Trip (_2_)	Log Reactor Power Hi Trip	Y	Fixed, Rising
LPD_HI Trip (_3_)	Local Power Density Hi Trip	—	Digital
DNBR_LO Trip (_4_)	Departure from Nucleate Boiling Ratio Low Trip	—	Digital
PZR_PR_HI Trip (_5_)	Pressurizer Pressure Hi Trip	—	Fixed, Rising
PZR_PR_LO Trip (_6_)	Pressurizer Pressure Low Trip	Y	MR, Falling
SG1_LVL_LO_RPS Trip (_7_)	SG-1 Low Level Trip	—	Fixed, Falling
SG2_LVL_LO_RPS Trip (_8_)	SG-2 Low Level Trip	—	Fixed, Falling
SG1_LVL_LO_ESF Trip (_9_)	SG-1 Low-Low Level Trip	—	Fixed, Falling
SG2_LVL_LO_ESF Trip (_A_)	SG-2 Low-Low Level Trip	—	Fixed, Falling
SG1_LVL_HI Trip (_B_)	SG-1 Hi Level Trip	—	Fixed, Rising
SG2_LVL_HI Trip (_C_)	SG-2 Hi Level Trip	—	Fixed, Rising
SG1_PR_LO Trip (_D_)	SG-1 Low Pressure Trip	—	MR, Falling
SG2_PR_LO Trip (_E_)	SG-2 Low Pressure Trip	—	MR, Falling
CMT_PR_HI Trip (_F_)	Containment Hi Pressure Trip	—	Fixed, Rising
CMT_PR_HH Trip (_G_)	Containment Hi-Hi Pressure Trip	—	Fixed, Rising
SG1_FLW_LO Trip (_H_)	SG-1 Low Coolant Flow Trip	—	RR, Falling
SG2_FLW_LO Trip (_I_)	SG-2 Low Coolant Flow Trip	—	RR, Falling
CWP Trip (_J_)	CPC-CWP	—	Digital

*BP, bistable processor; Digital, On/Off trip; Fixed, fixed trip setpoint; KNICS, Korea Nuclear Instrumentation & Control Systems; MR, variable trip setpoint by manual reset; OB, operator bypass; RPS, reactor protection system; RR, variable trip setpoint by automatic rate limiting; TSP, trip setpoint.

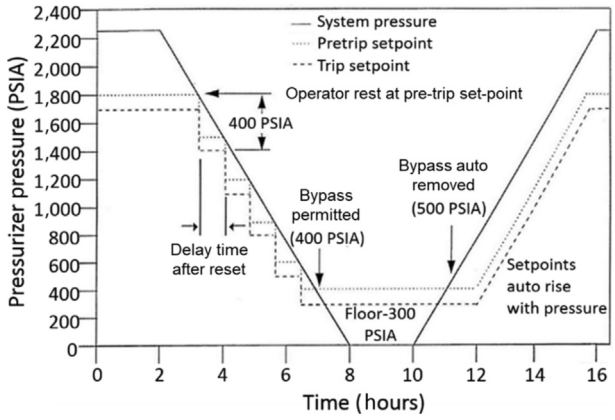

Fig. 6. An overview of the pressurizer-pressure-low trip logic [30].

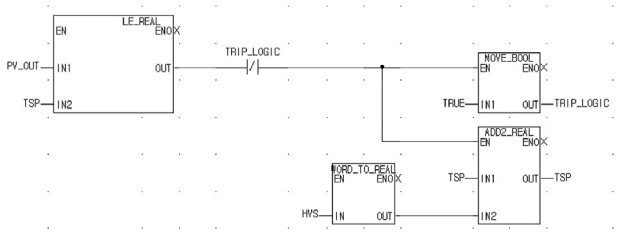

Fig. 7. A part of RESET_FALLING logic in BP PZR_PR_LO trip logic.
BP, bistable processor; PZR_PR_LO, pressurizer-pressure-low.

pretrip or trip signal, and the trip setpoint is decreased by hysteresis. When the BP is untripped, it restores the trip set-point value.

- Variable set-point logic: The BP generates a pretrip or trip signal when the process input signal reaches the level of the trip or pretrip setpoint. In this logic, the set-point value can change depending on the rising or falling of the process input signal.
- Manual reset logic: The operation is identical to that of variable set-point logic, but the operator can delay the trip by moving the trip setpoint to an upper or lower value by pushing a reset button.
- Digital logic: The BP generates a pretrip or trip signal based on the digital input signal (0 or 1) from other RPS modules, such as the core protection calculator.

Among 19 trip logics, the pressurizer-pressure-low (PZR_PR_LO) trip logic, which has a variable TSP and operator bypass function, was chosen as a case study to demonstrate the effectiveness of the proposed software test method. The PZR_PR_LO trip logic is one of the most complicated logics among BP trip logics; it includes various functions, such as operator bypass, reset delay timer, and set-point reset by the operator.

Fig. 6 shows the operation logic of the PZR_LO_PR trip [30]. The process variables of the PZR_LO_PR trip logic, which include the pressurizer pressure obtained from the measuring instruments (0–3,000 psi), are processed into analog voltage signals (0–10 voltage direct current (VDC)) that are converted into digital signals (0–30,000 counts) by a 15-bit analog–digital converter [31]. The trip logic generates a trip signal if the process variable decreases below the trip setpoint. When the plant is in full-power mode, the trip setpoint is fixed at 1,779 psi. The trip setpoint ranges between 1,779 psi and 300 psi during shutdown and start-up processes. The operator should manually decrease the trip setpoint while the pressure slowly decreases during the plant shutdown phase. When the pretrip alarm occurs, where the pretrip setpoint is at 70 psi above the trip setpoint, the operator has to push the reset button, after which the trip setpoint decreases to 400 psi below the current pressure. Further decrease of the trip setpoint is not permitted within a certain delay time, and bypass is permitted under 400 psi. When the pressurizer pressure increases as the plant starts up, the trip setpoint is automatically set to 400 psi below the current pressure, and the trip set-point reset bypass is canceled from 500 psi.

4.2. Test case generation of target software

The generation of the software test cases that cover all possible states of the software input and internal variables is one of the key steps in the software test–based method. Previous test-based approaches conducted in the nuclear field [12,13] have involved developing an input set as a trajectory form (a series of successive values for the input variables of a program that occur during the operation of the software over time) by random sampling of test sets from the software input profile. However, the limitations of those approaches include the uncertainty caused by random sampling, the ambiguity in the necessary length of a trajectory, and a long execution time per test case.

Because software failure is basically a deterministic process, i.e., the software will follow the same execution path and generate the same output for the same input and internal state of the software, it is possible to test the software by constructing a test set as a combination of possible profiles of the software input and internal variables and verifying whether the correct output was generated by the software for each test set. Therefore, there is no need for a long input test trajectory as in previous test-based methods; this improvement allows the software testing time to be drastically reduced. Furthermore, compared with the existing black-box testing methods, the total number of test cases for exhaustive software testing which covers all possible software states can be mathematically derived.

In this study, the software test cases were developed by identifying the variables that contribute to generating the output signal of the target PZR_PR_LO trip logic software and deriving a possible profile of the input and internal variables in consideration of the operating profile of the software.

4.2.1. Variables and states of the target software

As previously discussed, the BP trip logics are programmed with FBD/LD language. For example, Fig. 7 shows a part of the RESET_FALLING logic, which is one of the FBD components in the PZR_PR_LO trip logic. The value of the output variable (TRIP_LOGIC) is generated from the combined execution of several function blocks, as shown in Fig. 7. The LE_REAL function block in the leftmost position receives the process variable (PV_OUT) and the internal variable (TSP) as inputs and computes the output. The output of LE_REAL function block combined with the TRIP_LOGIC variable is used in the next function blocks as input. If the enable (EN) values of both function blocks are true, the TRIP_LOGIC variable is set as true by the MOVE_BOOL function block, and the value of the TSP variable is increased by the value of the hysteresis (HYS) variable by the ADD2_REAL function block.

The functions of the whole BP trip logic are configured by the network of function blocks in the form of a circuit as a function between the input variables and the output variables, similar to the above example. By inspecting the FBD/LD program of the PZR_PR_LO trip logic, the input and internal variables that determine the state of the output variable of the software (pretrip or trip signal) were investigated. There are a total of 143 variables in the logic. By excluding the variables for the constants and the temporary variables that are automatically calculated based on software input and internal values between scan intervals, the remaining

variables that contribute to generating the pretrip or trip signal of the PZR_PR_LO trip logic were identified, as shown in Fig. 8.

The status of the BP module, whether it is in normal operation mode or manual or automated test mode, is determined by the BP scan flag variable (T_SCAN_FLAG), the BP test status variable (BP_INTEST), and the periodic automated test start signal (BP_PAT_-START) transmitted from the automatic test and interface processor (ATIP). The process variable (_6_PV_OUT_AI) is obtained from the plant sensors and processed through the analog-to-digital converter (ADC) of the BP input module. The pretrip and trip set-point values (_6_PTSP_R, _6_TSP_R) of the trip logic change depending on the process variable and manual set-point reset signal generated by the operator (_6_RST_REQ_MCR_DI, _6_RST_REQ_RSR_DI) when the reset delay counter (_6_RST_DELAY_CNT) exceeds the predefined maximum count value. When its process variable reaches below the level of the trip setpoint that exists at that time, the BP software will generate a trip signal for the PZR_PR_LO trip logic. The trip signal can also be generated if there is an error signal from the analog input module or channel (_6_AI_CH_ERR, AI_2_MDL_ERR, AI2_ch6_6). In specific conditions in which operator bypass is permitted, (_6_OB_PERM) a trip signal can be bypassed if the operator provides a bypass signal (_6_OB_REQ_MCR_DI, _6_OB_REQ_RSR_DI). Detailed description of the selected variables used for the test case generation is shown in Table 2.

4.2.2. Obtaining the profile of the variables

From the viewpoint of NPP safety, the software testing of the BP trip logic needs to focus on the failure of its dedicated safety function, that is, the failure of trip signal generation when demand comes. In this study, the test cases that include the states of input and internal variables that cover all possible safety signal demand situations of the target software were developed based on the profile of each variable encountered in actual use during plant operation.

As the states of internal variables represent a certain state of running the software, the ranges of each internal variable that generates the trip signal were identified by inspecting the software logic and the other available information, such as the software

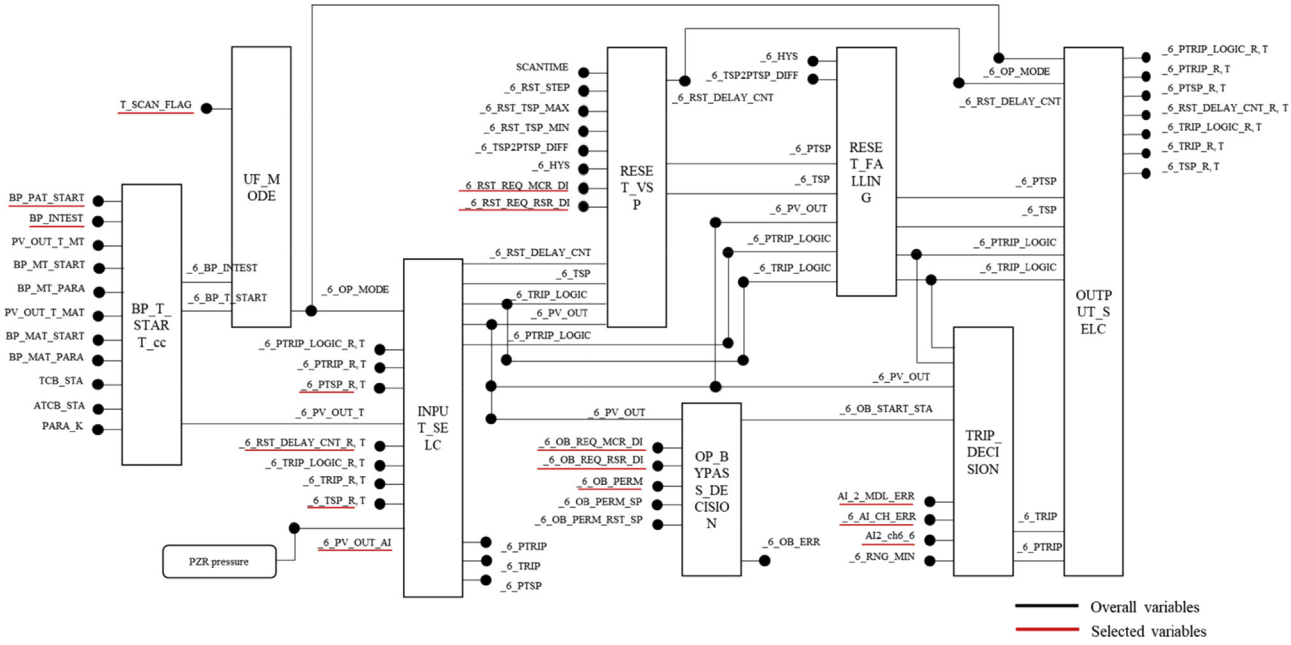

Fig. 8. An overview of the function block diagrams and variables of the BP PZR_PR_LO trip logic.

Table 2
Summarized variables for PZR_PR_LO (_6_) trip logic test case generation.

Variable	Description	Format	Type*
T_SCAN_FLAG	Flag for PLC scan operation (operation/test)	BOOL	SV
BP_INTEST	BP test status	BOOL	SV
_6_PTSP_R	PZR_PR_LO pretrip setpoint	WORD	SV
_6_TSP_R	PZR_PR_LO trip setpoint	WORD	SV
_6_RST_DELAY_CNT_R	PZR_PR_LO reset delay count	WORD	SV
AI_2_MDL_ERR	Analog input module error signal	BOOL	IV
_6_AI_CH_ERR	Analog input channel error signal	BOOL	IV
AI2_CH6_6	Analog input channel high over range error signal	BOOL	IV
_6_OB_PERM	Operator trip bypass permission	BOOL	IV
_6_OB_REQ_MCR_DI	Operator trip bypass request (from MCR)	BOOL	IV
_6_OB_REQ_RSR_DI	Operator trip bypass request (from RSR)	BOOL	IV
_6_RST_REQ_MCR_DI	Trip setpoint reset signal (from MCR)	BOOL	IV
_6_RST_REQ_RSR_DI	Trip setpoint reset signal (from RSR)	BOOL	IV
BP_PAT_START	Periodic automatic test start signal	WORD	IV
_6_PV_OUT_AI	PZR_PR_LO process parameter (PZR pressure)	WORD	IV

*BP, bistable processor; IV, input variable; MCR, main control room; PLC, programmable logic controller; PZR, pressurizer; RSR, remote shutdown room; SV, state (or internal) variable.

Table 3
Truth table of BPscan mode decision [31].

Case	BP_T_START	BP_INTEST	T_SCAN_FLAG	Result[a]
1	F	0	F	(1)
2	F	0	T	(4)
3	F	1	F	(5)
4	F	1	T	(4)
5	F	3	F	(1)
6	F	3	T	(6)
7	F	6	F	(1)
8	F	6	T	(6)
9	T	0	F	(1)
10	T	0	T	(4)
11	T	1	F	(2)
12	T	1	T	(4)
13	T	3	F	(1)
14	T	3	T	(3)
15	T	6	F	(1)
16	T	6	T	(3)

BP, bistable processor.
[a] (1): Operational scan mode, (2): Manual test (MT) scan mode, (3): Automatic scan (AT) mode, (4): Idle scan mode, (5): Restore from MT scan mode, (6): Restore from AT scan mode.

requirement specification and the software design specification documents [31]. For example, the scan mode of the BP software is determined by the state of two internal variables (BP_INTEST, and T_SCAN_FLAG) and one input variable (BP_T_START), as shown in Table 3. Because the focus of this study is to derive the test cases that represent the trip initiation condition by the software in normal operation, the possible combination of software internal variables of cases 1, 5, 7, 9, 13, and 15 in Table 3 which results in operational scan mode of the BP was derived as the profile of those variables. The profiles of other internal variables, shown in Table 2, that generate trip signals were derived in a similar way.

The input variables represent the software input from various sources, such as the pressure or temperature signals from the measurement instruments in the NPP, the operator action from the main control room (MCR) or from the remote shutdown room (RSR), and error signals from other modules. The profiles of the input variables were also derived; these represent the software inputs that are encountered in actual use.

Fig. 9 provides an illustration to explain a possible profile of the plant process parameter, which depends on the scan time (how often the digitalized system detects the trip demand) and the plant dynamics (how fast the plant transient is). If a deviation happens in an NPP, the plant process parameter will deviate from its normal value, and a reactor trip signal will be generated if a process parameter goes beyond its setpoint. In the real world, the process parameter exceeds the trip setpoint at points A and B. However, as the digitalized system reads the digital value from the transmitted analog signal converted via the ADC, the digitalized system detects the demand at points A_1, A_2, B_1, and B_2. Points A_1 and A_2 denote trip demand in cases of slow transient (deviation A), and points B_1 and B_2 denote cases of fast transient (deviation B).

As shown in Fig. 9, the trip demand is generated at point A_1 ($i = -3$), which is the 3rd digital value below the trip setpoint in case of slow transient; the trip demand is generated at point B_1 ($i = -9$) for fast transient when the scan time is t_1. As the process parameter moves faster in the case of a fast transient, the profile of the process variable that will generate the trip demand is larger than in the case of slow transient. The profile of the process variable that represents the trip demand condition also depends on how often the digital system scans the input signals. For example, in deviation B in Fig. 9, the trip demand is generated at point B_1 ($i = -9$) for scan time t_1, whereas it is generated at point B_2 ($i = -15$) for scan time t_2. Because the deviation of the plant process parameter is limited to a certain time interval, the deviation of the process variable increases as the scan time of the digital system increases.

In this study, the profile of the process variable, which is the pressurizer pressure for the PZR_PR_LO trip, was obtained by plant thermo-hydraulic simulation. As a representative pressure transient accident, a loss-of-coolant accident (LOCA) was selected for trip demand condition. Plant model APR-1400 was used to estimate the plant responses using the Multi-dimensional Analysis of Reactor Safety code [32], which was developed in the Korea Atomic Energy Research Institute for thermo-hydraulic analysis of an NPP. As the design requirements of the IDiPS-RPS limit the scan time to less than 50 ms, the pressure deviation during the time interval between operational scan modes (100 ms) at the point of trip demand was derived based on the simulation results.

Fig. 10 shows the deviation of pressurizer pressure before trip demand for various LOCA groups. To derive conservative test cases for trip signal generation by the trip logic software, the plant simulation result for the hypothetical double-ended guillotine break accident case was used to derive the profile of the process variable, as shown in Fig. 10. Table 4 shows D_{max} which is the maximum i (i^{th} digital value below trip setpoint) given 15-bit ADC resolution obtained from plant simulation results for various LOCA groups.

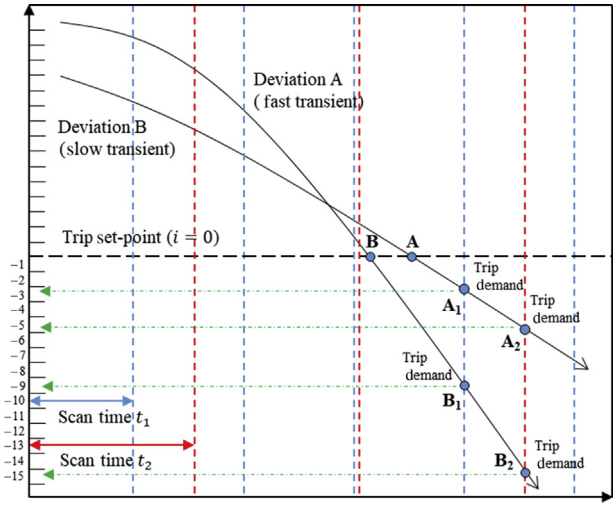

Fig. 9. Illustration of the process parameter profile for trip demand generation in consideration of the plant dynamics and scan time of digital system.

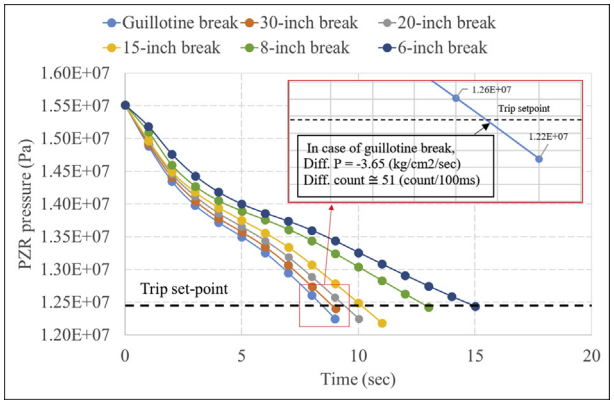

Fig. 10. Profile of process parameter for various LOCA groups.
LOCA, loss-of-coolant operation; PZR, pressurizer.

Table 4
D_{max} of the pressurizer pressure for various LOCA groups.

ID	Hole diameter (inch)	D_{max} (count)
1	30 × 2[a]	51
2	30	48
3	20	46
4	15	44
5	8	29
6	6	21

LOCA, loss-of-coolant accident; RCS, reactor coolant system.
[a] The scenario assumes that the 30-inch diameter pipe used in the reactor coolant system (RCS) undergoes a double-ended guillotine break (30-inch × 2) [33].

Trip bypass request and permission variables, which are inputs from the operators, are Boolean-type variables; so, they can have a value of either true or false. If an operator gives a trip bypass order, the system should not generate a trip signal; so, only the combinations of those variables that do not bypass the trip signal were examined in this study. The possible states of other input variables in Table 2 that generate the trip signal were derived by inspecting the software code and other available information, such as software requirement specification and software design specification.

Based on the obtained profiles of each input and internal variable, the test cases are formed as the combinations of the profiles of each software variable that will generate trip signal output as true. A total of 705,892,684 test cases were derived, as shown in Table 5. The test cases were used as input to the developed software test bed to verify whether the output variable updated by the BP trip logic software in the memory area matches the expected output.

4.3. Test procedure and results of target software

Based on the test cases derived from the possible states of each software input and internal variable, as described in the previous section, the test starts with initializing of the software test bed, which includes emulating the CPU registers and memory elements of the target digital processor. Then, the binary files of the target application program (PZR_PR_LO trip logic software), including the program file, which consists of the 32-bit–long binary code generated from the user application program written in FBD/LD programming language and the constant file that stores the memory map of the variables used in program, are loaded into the test bed. After reading the binary file of the target software, the test case file, which includes the memory address and the values of the software input and internal variables that should be tested, is loaded into test bed and overwrites the values in the emulated memory map. Then, the program executable file is executed by the test bed, and the value of the memory address, where the output variable (trip signal of PZR_PR_LO trip) is saved as an output file at the end of program execution. The output file is used to check whether the software output is the same as the expected output.

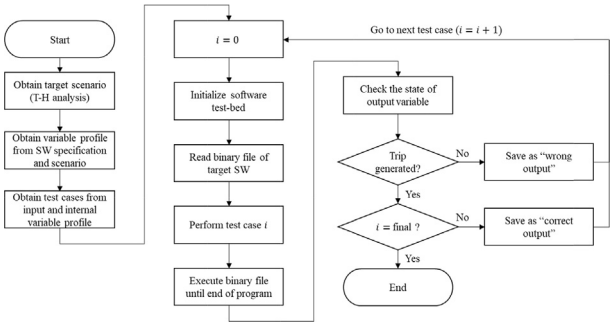

Fig. 11. An overview of the software testing procedure using the simulation-based software test bed and test cases.

Because the test cases are developed focusing on the trip initiation condition by the target software, the test case is verified as an error-free portion and saved as correct output if the value of the trip variable corresponds to true. However, if there is any test case that results in a value of trip signal variable of false, it is saved as wrong output and should be reviewed and debugged; the test should be restarted from the beginning, if necessary. Fig. 11 illustrates the procedure of software testing using the developed software test bed with the test cases.

The BP trip logic software consists of 32,566 lines of machine instruction; 98,755 lines were executed on average for a single test case. Among the executed instruction sets, LDIU (load integer unconditionally) and LDI (load integer) machine instructions were executed most frequently, 44,731 and 14,666 times, respectively. It was observed that 50.32% and 8.9% of the total execution time were spent by the LDIU and LDI instructions, where the internal CPU clocks used per instruction were 303 and 163 clocks in the developed software test bed, respectively. The longest internal CPU clocks used per instruction included instructions related to floating-point operation. For example, the CPU clocks used by CMPF (compare floating-point value) and LDFU (load floating-point unconditionally) were 2,406 and 1,104 clocks, respectively.

Fig. 12 shows a part of the test results using the test cases developed for the trip initiation condition of the PZR_PR_LO trip logic software as a case study. The output variable of the BP trip logic software is the TRIP_R_a variable, which is sent to CP as a trip signal for the voting logic. The TRIP_R_a variable is packed with trip signal output of various trip logics. For example, the _6_TRIP_R variable, which is the trip signal for the PZR_PR_LO trip logic, is packed at the 5th bit of the TRIP_R_a variable. As can be seen in Fig. 12, the test results showed that the state of the TRIP_R_a variable after the program execution is at 0x20, which indicates that the 5th bit of the trip signal (PZR_PR_LO trip signal) is set to 0x1, meaning that the software generated correct output for the given test case. All the 705,892,684 test cases developed from the

Table 5
An example of test cases developed for PZR_PR_LO (_6_) trip logic according to the profile of input and internal variable.

ID	Input, internal variable state of test case*	Description of test case
1	AI_2_MDL_ERR = 0x1, …	Trip generated because of error signal from analog input module
2	_6_AI_CH_ERR = 0x1, …	Trip generated because of error signal from analog input channel
3	AI2_CH6_6 = 0x1, …	Trip generated because of high over range error signal from analog input channel
4	_6_PV_OUT_AI = 0x256, …	Process variable is below its minimum range
5	_6_PV_OUT_AI = 0 × 454B, _6_TSP_R = 0x457E, _6_OB_REQ_MCR_DI = 0x1, …	Operator requested the trip bypass signal, but it is not permitted. Trip signal is generated because process variable is below the trip setpoint
6	_6_PV_OUT_AI = 0x3E5, _6_TSP_R = 0x3E7, _6_OB_REQ _MCR_DI = 0x0, _6_OV_REQ_RSR_DI = 0x0, …	Trip bypass is permitted, but the operator does not request the trip bypass signal. Trip signal is generated because process variable is below the trip setpoint

(in total of 705,892,684 test cases).

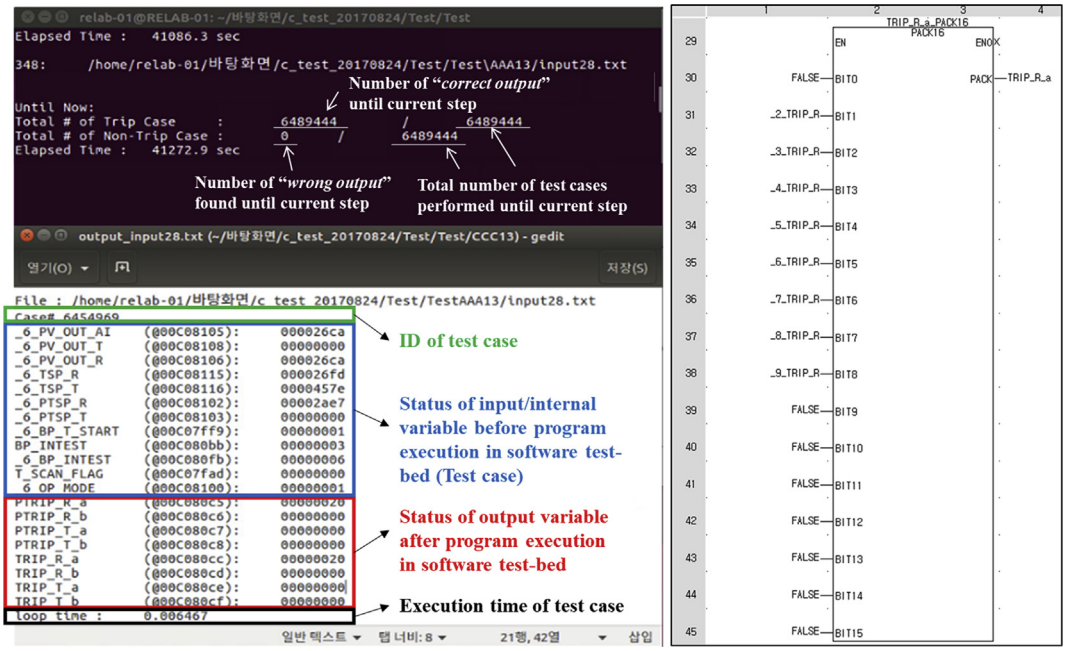

Fig. 12. An example of the test result for BP PZR_PR_LO trip logic software.

previous section generated trip signals for the PZR_PR_LO trip logic, and the test was conducted in 76.04 h using 16 3.60 GHz logical processors, that is, 6.205 ms were spent per test case on average in the software test bed.

5. Conclusion

In this study, a software test method using a simulation-based software test bed was proposed. The software test bed was developed considering the characteristics of the safety-critical PLC and the CPU architecture and memory map of the PLC microprocessor. Because the software test inputs for a safety-critical application, such as the RPS of an NPP, are inputs that cause activation of protective action, such as reactor trip, the software test case was developed in consideration of the digital signal processing features of the PLC and plant thermo-hydraulics data for plant transients or accidents in an NPP. As an application of the proposed software test method, software test cases were developed for a PZR_LO_PR trip of KNICS IDiPS-RPS BP software logic and were tested by capturing the state of output variables stored in the memory map after the end of the trip logic program.

An important characteristic of the proposed software test approach is that the test sets can be quantitatively derived to achieve exhaustive testing of the safety-critical software. In addition, compared with the existing black-box testing, this method can effectively reduce the software testing time per test case by emulating the software behavior given the software input and internal states at machine language level. Therefore, the proposed software test method can be used to support the software reliability quantification of NPP safety-critical I&C applications and further ensure the safety of software-based digital systems.

Although the proposed framework focuses on verifying that the software logic is error free when demand comes, other causes of software error should be investigated in consideration of the running environment. For example, the environment on which the software is running includes interaction with the operating system and hardware module. Although the application software can be tested as error free using the framework proposed in this study, the software will not generate a safety signal if the operating system kernel does not properly call the application software or if there is any error in the hardware module that affects the application or the operating system software. In addition, external causes of potential software error, such as wrong input by operator mistake or noise from sensors or the signal transmission path, also need to be considered to completely model the software failure.

Conflicts of interest

All authors have no conflicts of interest to declare.

Acknowledgment

This work was supported by the project of 'Evaluation of human error probabilities and safety software reliabilities in digital environment (L16S092000),' which was funded by the Central Research Institute (CRI) of the Korea Hydro and Nuclear Power (KHNP) company.

References

[1] M. Hassan, W.E. Vesely, Digital I&C Systems in Nuclear Power Plants. Risk-screening of Environmental Stressors and a Comparison of Hardware Unavailability with an Existing Analog System, NUREG/CR-6579, Brookhaven National Laboratory, 1998.
[2] National Research Council, Digital Instrumentation and Control Systems in Nuclear Power Plants: Safety and Reliability Issues, National Academies Press, 1997.
[3] H.G. Kang, T. Sung, An analysis of safety-critical digital systems for risk-informed design, Reliab. Eng. Syst. Saf. 78 (2002) 307–314.
[4] H. Ragheb, Operating and Maintenance Experience with Computer-based Systems in Nuclear Power Plants, in: International Workshop on Technical Support for Licensing Issues of Computer-based Systems Important to Safety, March 1996. München, Germany.
[5] U.S. Nuclear Regulatory Commission, Guidance for Evaluation of D3 in Digital Computer-based Instrumentation and Control Systems, 2012. BTP 7–19 (Rev. 6).
[6] K. Korsah, M.D. Muhlheim, R. Wood, A Qualitative Assessment of Current CCF Guidance Based on a Review of Safety System Digital Implementation Changes with Evolving Technology, ORNL/SR-2016/148, Oak Ridge National Lab, 2016.
[7] M.R. Lyu, Handbook of Software Reliability Engineering, McGraw-Hill, New York, 1996.

[8] M.C. Kim, S.C. Jang, J. Ha, Possibilities and limitations of applying software reliability growth models to safety critical software, Nucl. Eng. Technol. 39 (2007) 145–148.
[9] N. Fenton, M. Neil, W. Marsh, P. Hearty, D. Marquez, P. Krause, R. Mishra, Predicting software defects in varying development lifecycles using Bayesian nets, Inf. Software Technol. 49 (2007) 32–43.
[10] H.S. Eom, G.Y. Park, S.C. Jang, H.S. Son, H.G. Kang, V&V-based remaining fault estimation model for safety–critical software of a nuclear power plant, Ann. Nucl. Energy 51 (2013) 38–49.
[11] S. Brown, Overview of IEC 61508. Design of electrical/electronic/programmable electronic safety-related systems, Comput. Contr. Eng. J 11 (2000) 6–12.
[12] T.L. Chu, M. Yue, M. Martinez-Guridi, J. Lehner, Review of Quantitative Software Reliability Methods, BNL-94047–2010, Brookhaven National Laboratory, 2010.
[13] J. May, G. Hughes, A.D. Lunn, Reliability estimation from appropriate testing of plant protection software, Software Eng. J. 10 (1995) 206–218.
[14] T.L. Chu, Development of Quantitative Software Reliability Models for Digital Protection Systems of Nuclear Power Plants, NUREG/CR-7044, U.S. Nuclear Regulatory Commission, 2013.
[15] S. Kuball, J.H.R. May, A discussion of statistical testing on a safety-related application, Proc. Inst. Mech. Eng. O J. Risk Reliab. 221 (2007) 121–132.
[16] H.G. Kang, H.G. Lim, H.J. Lee, M.C. Kim, S.C. Jang, Input-profile-based software failure probability quantification for safety signal generation systems, Reliab. Eng. Syst. Saf. 94 (2009) 1542–1546.
[17] S.M. Shin, S.H. Lee, H.G. Kang, H.S. Son, S.J. Lee, Test based reliability quantification method for a safety critical software using finite test sets, in: Proceedings of the 9th International Topical Meeting on Nuclear Plant Instrumentation, Control & Human–machine Interface Technologies (NPIC & HMIT 2015), Charlotte, NC, February 2015.
[18] C.V. Ramamoorthy, W.T. Tsai, Advances in software engineering, Computer 29 (1996) 47–58.
[19] K.C. Kwon, M.S. Lee, Technical review on the localized digital instrumentation and control systems, Nucl. Eng. Technol. 41 (2009) 447–454.
[20] J.G. Choi, S.J. Lee, H.G. Kang, S. Hur, Y.J. Lee, S.C. Jang, Fault detection coverage quantification of automatic test functions of digital I&C system in NPPS, Nucl. Eng. Technol. 44 (2012) 421–428.
[21] M. Lee, S. Song, D. Yun, Development and Application of POSAFE-Q PLC Platform, IAEA-CN-194, International Atomic Energy Agency (IAEA), 2012.
[22] K. Koo, B. You, T.W. Kim, S. Cho, J.S. Lee, Development of Application Programming Tool for Safety Grade PLC (POSAFE-Q), Transactions of the Korean Nuclear Society Spring Meeting, May 2006. Chuncheon, Korea.
[23] J. Palomar, R.H. Wyman, The Programmable Logic Controller and its Application in Nuclear Reactor Systems, NUREG/CR-6090, U.S. Nuclear Regulatory Commission, 1993.
[24] Texas Instruments, TMS320C3x User's Guide, 1997.
[25] D. Huizinga, A. Kolawa, Automated Defect Prevention: Best Practices in Software Management, John Wiley & Sons, 2007.
[26] C. Kaner, J. Falk, Testing Computer Software, Wiley, 1999.
[27] International Electrotechnical Commission, Programmable Controllers – Part 3: Programming Languages, IEC, 1993, pp. 61131–61133.
[28] J. Yoo, J.H. Lee, J.S. Lee, A research on seamless platform change of reactor protection system from PLC to FPGA, Nucl. Eng. Technol. 45 (2013) 477–488.
[29] G.Y. Park, K.Y. Koh, E. Jee, P.H. Seong, K.C. Kwon, D.H. Lee, Fault tree analysis of KNICS RPS software, Nucl. Eng. Technol. 40 (2008) 397–408.
[30] J.G. Choi, D.Y. Lee, Development of RPS trip logic based on PLD technology, Nucl. Eng. Technol. 44 (2012) 697–708.
[31] Doosan Heavy Industries and Construction Co., Ltd, BP SDS for Reactor Protection System, 2008. KNICS-RPS-SDS231 (Rev. 3).
[32] J.J. Jeong, K.S. Ha, B.D. Chung, W.J. Lee, Development of a multi-dimensional thermal-hydraulic system code, MARS 1.3.1, Ann. Nucl. Energy 26 (1999) 1611–1642.
[33] U.S. Nuclear Regulatory Commission, Report of the US Nuclear Regulatory Commission Piping Review Committee, NUREG/1061, 1984.