Nuclear Engineering and Technology 53 (2021) 131–141

Contents lists available at ScienceDirect

Nuclear Engineering and Technology

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

Original Article

Quantitative measures of thoroughness of FBD simulations for PLC-based digital I&C system

Dong-Ah Lee, Eui-Sub Kim, Junbeom Yoo*

Konkuk University, Republic of Korea

ARTICLE INFO

Article history:
Received 9 February 2020
Received in revised form
19 May 2020
Accepted 11 June 2020
Available online 22 June 2020

Keywords:
Simulation coverage
FBD simulation
Scenario generation
Coverage estimation

ABSTRACT

Simulation is a widely used functional verification method for FBD programs of PLC-based digital I&C system in nuclear power plants. It is difficult, however, to estimate the thoroughness (*i.e.*, effectiveness or quality) of a simulation in the absence of any clear measure for the estimation. This paper proposes two sets of structural coverage adequacy criteria for the FBD simulation, *toggle coverage* and *modified condition/decision coverage*, which can estimate the thoroughness of simulation scenarios for FBD programs, as recommended by international standards for functional safety. We developed two supporting tools to generate numerous simulation scenarios and to measure automatically the coverages of the scenarios. The results of our experiment on five FBD programs demonstrated that the measures and tools can help software engineers estimate the thoroughness and improve the simulation scenarios quantitatively.

1. Introduction

Simulation plays an important role in understanding and verifying the function of programmable logic controller (PLC) programs [1–3] written in the function block diagram (FBD) [4], when developing safety-related digital instrumentations and control systems (I&Cs) in nuclear power plants (NPPs). Since PLC application software, such as an executable program like Bistable Processor, is too primitive to apply any verification methods, its system test can often be completed after testing the PLC that is configured or downloaded with the application software. Therefore, functional verification of PLC application software has tended to focus on FBD programs in the design phase, in order to ensure the functional correctness of the application as soon as possible, before downloading it into specific PLC hardware.

An inherent weakness of simulation-based verification methods is that they do not provide any quantitative measure to estimate the thoroughness, effectiveness, or quality of the simulation performed. They infamously suffer from a lack of completeness with respect to the coverage of system behaviors. The sufficiency of the simulations remains doubtful, even if they correspond to all functional requirements and were performed correctly. Practical experience has shown that many design errors cannot be revealed by a number of successful simulations [5,6].

This paper proposes two sets of structural coverage adequacy criteria [7] for FBD simulations, where the term simulation coverage indicates the structural coverage adequacy criteria in the paper, *i.e.* the measures to estimate the thoroughness of functional FBD simulation scenarios according to the structural elements of FBD programs. *Toggle coverage (TC)* and *modified condition/decision coverage (MC/DC)* are the two coverages used to measure or estimate how much a set of simulation scenarios satisfies the simulation coverages. They also provide invaluable information to improve and refine the scenarios, quantitatively. If the achieved simulation coverage is considered insufficient, either additional simulation scenarios shall be specified, or a rationale shall be provided.

We developed two supporting tools, *FBDSim* and *FBDCover*, for automatic measurement of numerous simulation scenarios. They can read and measure the simulation coverage upon any FBD program written according to the PLCopen TC6 standard format [8]. Three experiments were run using the supporting tools to simulate five FBD programs. The first simulation takes a processing value as an input which is randomly generated. The second takes values with an increasing or decreasing tendency. The last takes not only the processing values but also other inputs for testing or error signals. Both coverages increased as the experiment progressed. Our experimental results demonstrated the capability of the measures and tools to help software engineers refine and improve the

* Corresponding author. C308, Engineering Building, Konkuk University, Seoul, 05029, Republic of Korea.
E-mail address: jbyoo@konkuk.ac.kr (J. Yoo).

https://doi.org/10.1016/j.net.2020.06.017

simulation scenarios quantitatively.

Our approach corresponds to the recommendation by the IEC standards for the functional safety of safety-related systems [9–11]. They recommend that "*To evaluate the completeness of test cases and to demonstrate that there is no unintended functionality, the coverage of requirements at the software unit level shall be determined and the structural coverage (e.g., statement coverage, branch coverage and MC/DC) shall be measured.*" We extended the recommendation to the functional simulation of the whole FBD program, consisting of numerous execution cycles, unlike the software unit test limited to performing only a single or several execution cycles [12].

The remainder of the paper is organized as follows: Section 2 provides the background for the study including a literature survey of the relevant research. The quantitative measures for FBD simulation are proposed in Section 3 and the supporting tools we developed are explained in Section 4. The experimental results are analyzed in Section 5, and finally section 6 concludes the paper and provides remarks on future research extension and direction.

2. Related work

2.1. The PLC software development

The typical software development process for PLC-based digital I&Cs (*e.g.* a reactor protection system (RPS) and an engineered safety features-components control system (ESF-CCS)) is in Fig. 1. The software requirements specification (SRS) is first written in natural languages, and then the design specification is manually modeled with PLC programming languages such as FBD or ladder diagram (LD). Commercial PLC vendors provide PLC SW engineering tools (*e.g.* TriStation 1131 of Invensys for TriStation 1131 PLC, SIMATIC-Manager of Siemens for SIMANTIC Controller PLC, pSET of PONU-Tech for POSAFE-Q PLC, and SPACE of AREVA for TELEPERM XS PLC) that mechanically translate FBD or LD programs into ANSI-C programs and executable codes subsequently for specific target PLCs.

Simulation is the first functional verification method for FBD programs. Engineers execute an FBD program (usually a whole FBD program, not a unit) with a set of predefined simulation scenarios that execute 100 to 10,000 PLC execution cycles, and check whether the program works as intended. Preparing meaningful simulation scenarios is a crucial factor for success in this functional verification with simulation.

PLC SW engineering tools often produce supplementary C programs to perform both control flow graph (CFG)-based structural testing [7] and simulation, since the PLC executable codes are too primitive to do the system/integration/unit testing. Conventional software testing tools for C programs such as *LDRA* [13] and the one embedded in *SCADE* [14] can perform various test on the C programs. Measuring CFG-based structural coverages like all statements and MC/DC can also be used to assess the quality of the test cases [15,16].

This test approach suffers a problem in that the translated/intermediate C programs lack sufficient control flows to check the CFG-based structural coverages [17]. FBD or LD are dataflow-based programming languages for PLC, and programs written in FBD or LD include almost no control flows, except for a few blocks containing a selection function like SEL and MUX. On the other hand [18], developed three new dataflow-based structural coverages for FBD programs, and proposed the direct testing of FBD programs [12,19]. They are all, however, structural testing techniques for software units with a single (or several) execution(s). They cannot be simply applied to simulation scenarios of massive execution cycles (*e.g.*, 100–10,000 cycles) for a whole FBD program.

2.2. Coverage adequacy criteria in software testing

Many approaches are available in software testing [20]. Reviews, walk-throughs, and inspections are referred to as *static testing*, whereas executing programmed code with a given set of test cases is referred to as *dynamic testing*, which is pertinent to our discussion and we refer to simply as *testing*. To describe the viewpoint taken by test engineers when designing test cases, software testing methods are traditionally divided into black- and white-box testing [21–24], in which test cases are derived from the structure of programs and from program specifications, respectively, as shown in Fig. 2.

Functional (black-box) testing can be performed in any step of the software development process when functional specifications are in preparation, whereas structural (white-box) testing can only be applied to units or components, due to the limitation of required code analysis such as data-flow or control-flow. They both test the functionality of a program according to its requirements specification, but their measures of the thoroughness of test cases differ.

Several software metrics or measures are frequently used to assist in determining the adequacy of the testing or test cases. *Adequacy criterion* is a set of test obligations [7] used to investigate test cases. Each requirement in a requirements specification or code elements, such as *statement*, *branch*, *condition*, and *path*, are examples of the obligations. A test suite satisfies an adequacy criterion, if and only if all the tests succeed (pass), and every test obligation in the criterion is satisfied by at least one of the test cases

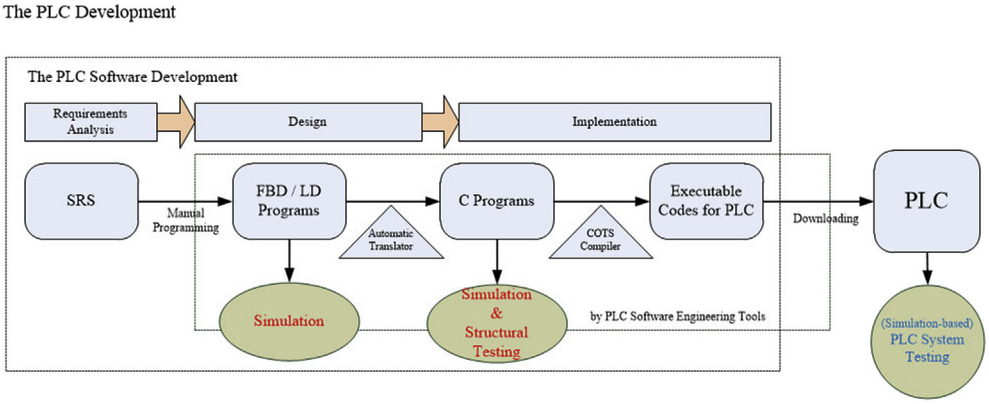

Fig. 1. A typical software development process for PLC.

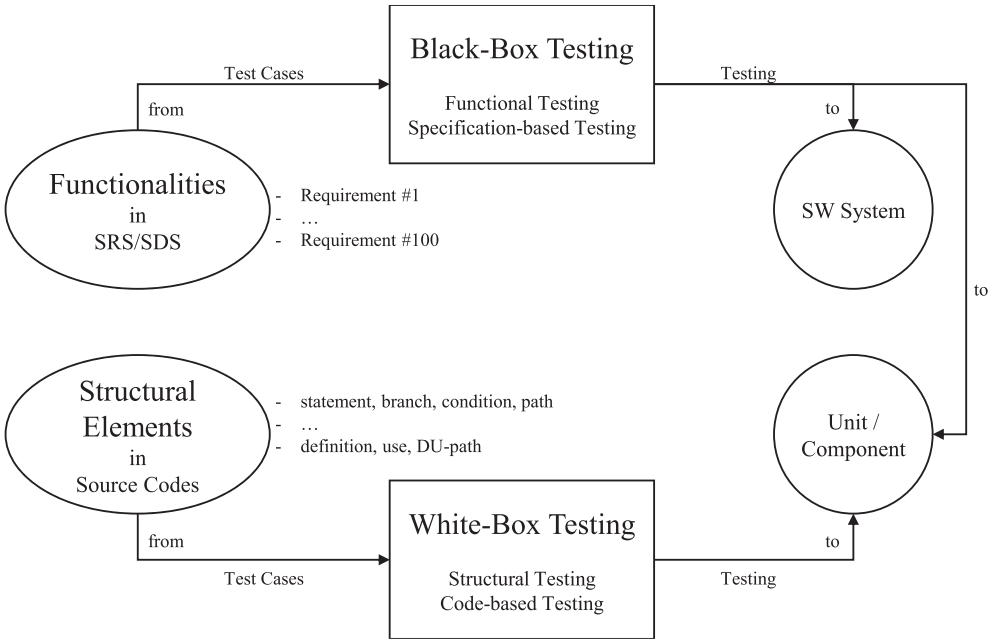

Fig. 2. Black-box testing *vs.* White-box testing.

in the test suite. For example, the statement coverage adequacy criterion is satisfied by test suite *S* for program *P* if each executable statement in *P* is executed by at least one test case in *S* and the outcome of each test execution has been passed.

However, often no test suite can satisfy a criterion for a given program, due to defensive programming or code reuse. The unsatisfiability of adequacy criteria is managed by measuring to the extent to which a test suite approaches an adequacy criterion, which is called *coverage adequacy criterion* and often called 'OOO coverage,' for brevity, *e.g.*, statement coverage and branch coverage. Measuring coverage (*i.e.*, the percentage of satisfied test obligations) can be a useful indicator of progress toward a thorough test suite. Trouble spots requiring more attention in testing can also be identified.

Coverage adequacy criterion can be used as a measure for the both testing methods, *i.e.*, functional and structural. Functional testing should achieve 100% of requirements coverage at all levels of functional test suites. If not, some requirements will not be tested by the test suite, which may lead to serious consequences. However, due to the difficulty in achieving 100% of structural coverages in structural testing, appropriate levels (%) are used. In practice, we develop functional test suites first, and then measure the structural coverage on the basis of the test suites to identify what is missing, in line with the recommendations of international standards on functional safety [9,10].

2.3. Coverage adequacy criteria for FBD

Many studies on coverage adequacy criteria include very few on FBD. Jee et al. [18] proposed three different coverage criteria for the FBD test: basic coverage (BC), input condition coverage (ICC), and complex condition coverage (CCC). They interpret an FBD program as a directed data flow graph in order to identify a data flow path. The three coverage criteria use the data flow path to generate test cases and measure the adequacy of the test cases quantitatively. The measures and testing technique are useful to evaluate the coverage on the FBD program. Their work is similar to ours in terms of a coverage criteria proposal, except that their focus on one cycle test and on finding a critical path did not include verification of an FBD program as PLC software in continuous operation.

Maruchi et al. [25] proposed MC/DC for data flow languages (MC/DC4d) and propagation toggle coverage (PTC) for data flow language. The MC/DC4d is an improved version of traditional MC/DC coverage such that "*Each input connected to outputs is shown to independently affect the values of the outputs.*" The PTCis also improved to support the MC/DC4d: "*Each edge (including inputs) in a program is shown to independently affect the values of the outputs.*" Both coverages are based on affecting-edge (fan-in condition) and affected-edge (fan-out). It is useful to evaluate variable changes, but it is hard to find a continuously setting value or more complex condition such as a fixed value after changing from zero to one or reverse. Further continuous operation such as simulation was not considered, even though one cycle or limited composition cycle is inappropriate to find such complex logic.

3. Quantitative measures for FBD simulations

This paper proposes two FBD simulation coverages, *toggle coverage* and *MC/DC*, which can quantitatively measure the thoroughness of the functional FBD simulations according to the structural elements of FBD programs. We applied the idea of the

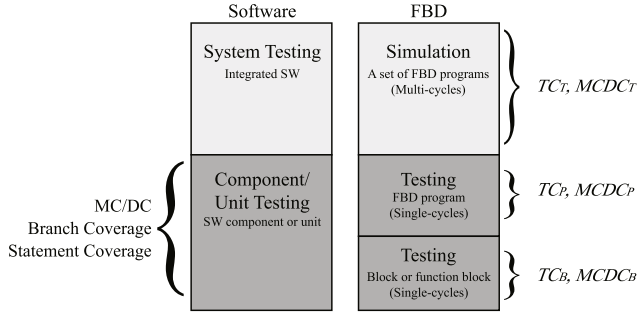

Fig. 3. Relation among structural test adequacy criteria in SW testing and FBD simulation.

structural coverage adequacy criteria of software testing, which could be applied to a software unit or component with a single execution cycle, to the FBD simulation to estimate the structural coverage of FBD simulation scenarios in a whole FBD program with multiple execution cycles. The FBD simulation coverage checks each block directly, whereas software testing measures the coverages according to CFG and DFG transformed from source codes.

The measurement starts with the coverages of a basic element, a block, and gradually expands to a software program and then to massive simulation scenarios. Toggle coverage for a block (TC_B) is the basic measure, and it expands to the toggle coverage for a program (TC_P) and massive simulation scenarios (TC_T). The MC/DC expands in the same way ($MCDC_B \rightarrow MCDC_P \rightarrow MCDC_T$). Basic coverages for a block and a program such as TC_B, TC_P, $MCDC_B$, and $MCDC_P$ correspond to the structural coverages of software testing. However, total coverages, TC_T and $MCDC_T$, which are supersets of basic coverages, can measure the structural coverages of all sets of simulation scenarios upon a whole FBD system.

Fig. 3 shows the correspondence between software testing and the FBD simulation in terms of structural coverages. While the structural coverages such as statement, branch, and MC/DC are applicable only to software units or components, the FBD simulation coverages can be used for both an FBD unit and the whole FBD system. The toggle coverage corresponds to the branch and statement coverage of software testing and the MC/DC corresponds to MC/DC of software testing. Other structural coverages, for example path coverage, that are not applicable to an FBD program are not shown in the figure.

3.1. Toggle coverage of FBD simulation

Toggle coverage [26] is one of the oldest coverage measures in circuit design verification. It measures bits of wires or registers in the circuit design that have toggled. The measurement focuses on how many bits are changed from a value of zero to one (0-to-1) and back from one to zero (1-to-0) during simulation. A bit is said to be fully covered, *i.e.* 100% coverage, when it toggles back and forth at least once. The toggle coverage of the FBD simulation shows that how many outputs of blocks toggle during simulation.

- **Adequacy criterion**: An output of each Boolean block should be toggled back and forth at least once by a simulation scenario.
- **Rationale**: A fault in a function block can only be revealed by executing the faulty function block.
- **Corresponding SW testing coverages**: statement coverage.

The FBD program in Fig. 4 is a part of the simplified trip (shutdown) logic in RPS BPs. It has 5 blocks (2 LT, 2 AND, and 1 OR blocks), 7 inputs (3 integer (I) and 4 Boolean (B) input variables), and 1 output (TRIP). RNG_MIN and RNG_MAX are constants fixed at 10 and 20,000 respectively. The two FBD simulation coverages use the small example to explain how to measure the thoroughness of simulation scenarios in detail.

Table 1
Estimation of TC_B for the block (1) in Fig. 4

Cycle	Input values		Block (1)LT_INT			TC_B (%)
	RNG_MIN	PV_OUT	OUT	T_P	T_N	
1	10	5	0			0
2	10	15	1	X		50
3	10	25	1			50
4	10	5	0		X	100
5	10	35	1	X		100

3.1.1. TC_B: toggle coverage for a block

The basic measure for the toggle coverage begins with a block (TC_B). A Boolean block has two toggles, a positive toggle (0-to-1, T_P) and a negative toggle (1-to-0, T_N). The block is fully covered where the two toggles are detected at least once during a simulation scenario. If a simulation catches only T_P, then the toggle coverage of the block is 50% TC_B. If neither T_P nor T_N is detected, *i.e.*, the block produces a constant value such as 0 or 1 while executing the simulation, it is 0% TC_B.

$$TC_B = \frac{the\ number\ of\ toggles\ executed}{the\ number\ of\ possible\ toggles} \times 100(\%)$$

Table 1 shows how to estimate TC_B with a simulation scenario having 5 execution cycles. The initial values of the inputs, RNG_MIN and PV_OUT, at the first cycle are 10 and 5 respectively. The block, (1)LT_INT, toggles at the 2nd (T_P), 4th (T_N), and 5th(T_P) cycles. TC_B is calculated at every execution cycle of a scenario, and duplicate toggles in the shade columns do not count. Therefore, the TCB of (1) LT_INT becomes 50% at the 2nd cycle and 100% at the 4th cycle.

3.1.2. TC_P: toggle coverage for an FBD program

TC_P is a set of TC_B of blocks that organize an FBD program. An FBD program is organized with blocks connected each other. Simulating an FBD program measures the TC_B of all blocks in the program. We can estimate all the TC_B and calculate the coverage by using the program, TC_P.

$$TC_P = \frac{\sum TC_B\ by\ a\ scenario}{the\ number\ of\ blocks\ in\ a\ program}(\%)$$

The FBD program in Fig. 4 takes a value measurement of the input, PV_OUT. The PV_OUT is valid when it only has a value

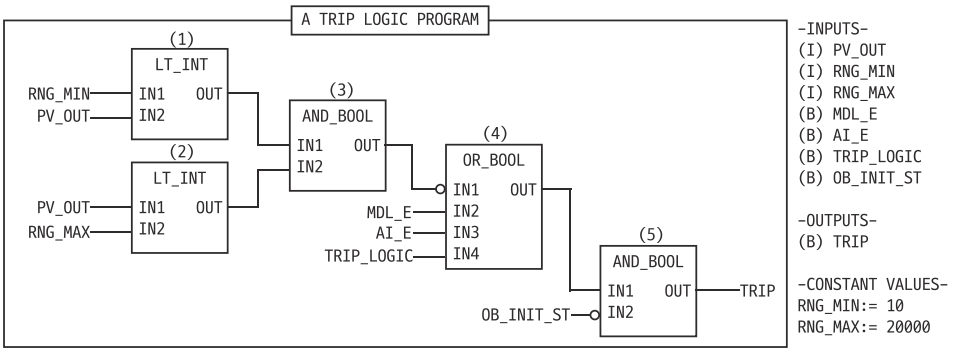

Fig. 4. An example FBD program called trip logic.

Table 2
Estimation of TC_T for the FBD program in Fig. 4

SS	C	I PV_OUT	Blocks															TC_T (%)
			(1)			(2)			(3)			(4)			(5)			
			O	T_P	T_N	O	T_P	T_N	O	T_P	T_N	O	T_P	T_N	O	T_P	T_N	
1st	1	10,000	1			1			1			0			0			0
	2	15,000	1			1			1			0			0			0
	3	25,000	1			0		X	0		X	1		X	1		X	40
	4	25,000	1			0			0			1			1			40
	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
2nd	1	10,000	1			1			1			0			0			40
	2	0	0		X	1			0		X	1	X		1	X		70
	3	10,000	1	X		1			1	X		0		X	0		X	90
	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…

SS: Simulation Scenario, C: Cycle, I: Input value, O: Output.

between RNG_MIN and RNG_MAX, but is invalid when its value is less than or equal to RNG_MIN or larger than or equal to RNG_MAX. If a simulation scenario of the program includes only a range of the valid values (i.e. 10 < PV_OUT <20,000), the Boolean block (1) LT_INT will never toggle, but it gets stuck at 1. The block (2)LT_INT has the same result too. In other words, the block (1)LT_INT executed by the simulation scenario never gets both of TP and TN. Thus, the simulation scenario will have 0% of TCB for the (1)LT_INT.

We may get the 100% of TCB in two ways. Refining a simulation scenario make the TCB increasing. For example, if PV_OUT in the scenario has rising values from below the RNG_MIN to above RNG_MIN, changing the value downward in the middle of the scenario increases the TC_B. However, this is not straightforward, since we have to refine the simulation scenario with several execution cycles. The block toggle coverage can also be improved by giving the task to other simulation scenarios. The FBD simulation coverage is measured from a set of simulation scenarios, and others can satisfy the coverage for the block. Total toggle coverage measures the toggle coverage in a block by a whole set of simulation scenarios.

3.1.3. TC_T: total toggle coverage for massive simulation scenarios

Total toggle coverage (TC_T) measures the TC_B of all Boolean blocks in an FBD program for a set of simulation scenarios. An FBD program has 100% of TC_T, if all Boolean blocks have been toggled by a set of simulation scenarios at least once. Individual toggle coverages (TC_B, TC_P) can be used to measure and improve a single simulation scenario, while TC_T can be used to measure the thoroughness of the whole set of FBD simulation scenarios.

$$TC_T = \frac{\sum TC_B \text{ by a set of scenarios}}{\text{the number of blocks in a program}} (\%)$$

TC_T continues measuring the coverage until the simulation executes all the scenarios in the set. Table 2 shows the measurement of TC_T with 2 consecutive simulation scenarios. The toggles in the shade of the 2nd scenario are not measured repeatedly, because they have been already checked when the toggles occurred the first time in the 1st scenario.

3.2. Modified condition/decision coverage (MC/DC) for FBD simulation

MC/DC is the most widely used structural coverage in software testing [27,28]. It tries to effectively test important combinations of conditions, without exponentially increasing. Important combinations mean a set of test inputs for a compound condition that each basic condition independently affects the outcome of each decision. Measuring MC/DC for the FBD simulation begins with measuring the MC/DC of a block which is the basic element in an FBD program. The MC/DC for a block is measured by the same procedure as that for MC/DC [7], but the detail is beyond the scope of this paper.

- **Adequacy criterion**: Each important condition should be executed at least once by a simulation scenario.
- **Rationale**: Multiple condition coverage is impractical in practice. MC/DC was developed to achieve many of the benefits of multiple-condition testing while retaining the linear growth in required test cases of condition/decision testing.
- **Corresponding SW testing coverages**: MC/DC coverage, multiple condition coverage

3.2.1. $MCDC_B$: MC/DC for a block

$MCDC_B$ measures how many important conditions are covered

Table 3
Important conditions for $MCDC_B$.

Blocks	Important Conditions
AND	(IN1, IN2): {(0, 1), (1, 0), (1, 1)}
AND	(IN1, IN2, IN3): {(0, 0, 1), (0, 1, 0), (1, 0, 0), (1, 1, 1)}
OR	(IN1, IN2): {(0, 0), (0, 1), (1, 0)}
OR	(IN1, IN2, IN3): {(0, 0, 0), (0, 0, 1), (0, 1, 0), (1, 0, 0)}
GT	(IN1 > IN2), (IN1 ≤ IN2)
GE	(IN1 ≥ IN2), (IN1 < IN2)
LT	(IN1 < IN2), (IN1 ≥ IN2)
LE	(IN1 ≤ IN2), (IN1 > IN2)
EQ	(IN1 = IN2), (IN1 ≠ IN2)
SR	(SET, RESET, prev): {(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 1, 0), (1, 1, 1)}
RS	(SET, RESET, prev): {(0, 0, 0), (0, 0, 1), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)}
SEL	(G): {0, 1}
MUX	(K): {0, …, N} where N is the number of inputs

by a simulation scenario in a block. $MCDC_B$ of a Boolean block is fully covered by a simulation scenario, if all important conditions are covered by the simulation scenario. If a simulation does not achieve 100% $MCDC_B$, it implies that the block has been executed without one or more important conditions.

$$MCDC_B = \frac{\text{the number of important condition executed}}{\text{the number of importnat conditions}}(\%)$$

Table 3 summarizes the important conditions for widely used function blocks in order to achieve 100% of $MCDC_B$. For example, of the four combinations of inputs of AND block ((0,0), (0,1), (1,0), (1,1)), only three (*i.e.*, (0,1), (1,0), (1,1)) are important for 100% of the $MCDC_B$ of the block. Blocks, such as AND and OR, may have more than two inputs. The number of combinations increases in line with the increase in the number of inputs, and the number of important combinations increases too.

3.2.2. $MCDC_P$: MC/DC for an FBD program

$MCDC_P$ is a set of the $MCDC_B$ of blocks that organize an FBD program, which has the same relation between the toggle coverages for a block and a program. Simulating an FBD program measures the $MCDC_B$ of all blocks in the program. We use all the $MCDC_B$ to calculate the MC/DC of the program, $MCDC_P$.

$$MCDC_P = \frac{\sum MCDC_B \text{ by a scenarios}}{\text{the number of blocks in a program}}(\%)$$

If the PV_OUT in Fig. 4 only has the valid value (*i.e.*, 10 < PV_OUT < 20,000), the (3)AND_BOOL block reaches only 33% of $MDCD_B$. A simulation scenario including invalid values (10 ≥ PV_OUT or PV_OUT ≥ 20,000) improves the $MCDC_B$ of the (3) AND_BOOL, therefore $MCDC_P$ also increases. The $MCDC_P$ can also be increased by giving the task to other simulation scenarios. The FBD simulation coverage is measured from a set of simulation scenarios, and others can satisfy the $MCDC_P$ for the function block. The total MC/DC, $MCDC_T$, measures the $MCDC_B$ by a whole set of simulation scenarios.

3.2.3. $MCDC_T$: total MC/DC for massive simulation scenarios

$MCDC_T$ measures the $MCDC_B$ of all Boolean function blocks in an FBD program for a set of simulation scenarios. An FBD has 100% of $MCDC_T$, if all important combinations of input conditions for all Boolean function blocks in an FBD program are covered by a set of simulation scenarios at least once, *i.e.*, $MCDC_B$ for all blocks are covered by a set of simulation scenarios at least once. Individual $MCDC_P$ can be used to measure and improve a single simulation scenario, while $MCDC_T$ can be used to measure the thoroughness of the whole set of FBD simulation scenarios.

$$MCDC_T = \frac{\sum MCDC_B \text{ by a set of scenarios}}{\text{the number of blocks in a program}}(\%)$$

4. The supporting tools development

We developed two supporting tools, *FBDSim* and *FBDCover*, to automate the measurement for numerous simulation scenarios. They can read and measure the simulation coverages of any FBD program written according to the PLCopen TC6 standard format [8]. The tools are used in conjunction with the *NuDE 2.0* [29,30], a software development framework based on formal methods.

4.1. FBDSim

Commercial PLC SW engineering tools are not compatible with others, and it is often impossible to simulate an FBD program independently from its PLC SW engineering tools. We developed FBDSim to simulate an FBD program, PLC-independently, as depicted in Fig. 5. It reads an FBD program written according to the PLCopen TC6 standard format [8] and a set of FBD simulation scenarios, and then automatically simulates the FBD program with the scenarios in batches.

The FBD program for simulation is provided by an FBD Editor [31], which can read, edit, and store an FBD program of the PLCopen TC6 XML format. The FBD simulation scenarios are produced by an automatic scenario generator, FBD Scenario Generator [32], which produces numerous simulation scenarios according to the RPS trip (shutdown) logics. It requests auxiliary information from the FBD program in order to make the scenarios meaningful. Initial values and the rate of change in all input variables, trip/pre-trip set-points, the overall percentage of trip situations, and the number of PLC execution cycles for each scenario are requested. The results of the FBD simulation are transferred to FBDCover to measure coverages and provide quantitative information in a concise manner.

4.2. FBDCover

FBDCover, as depicted in Fig. 6, is a visualization tool to provide quantitative information about the FBD simulation coverages.

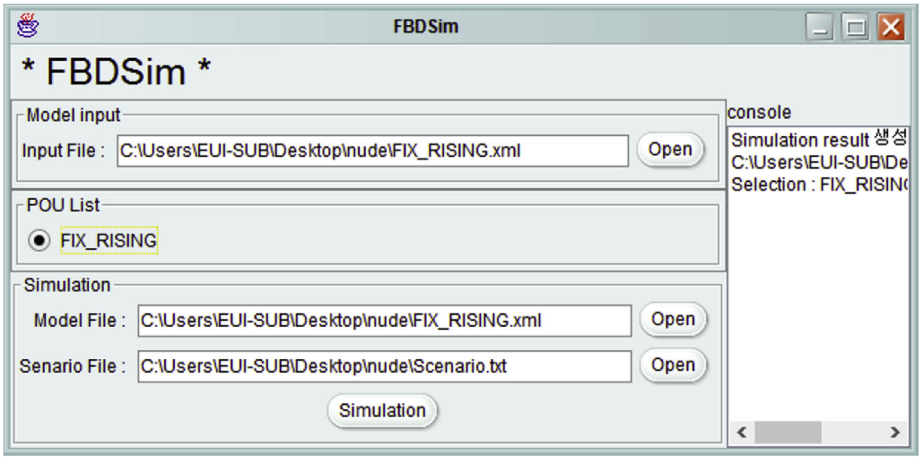

Fig. 5. A screen-dump of *FBDSim*.

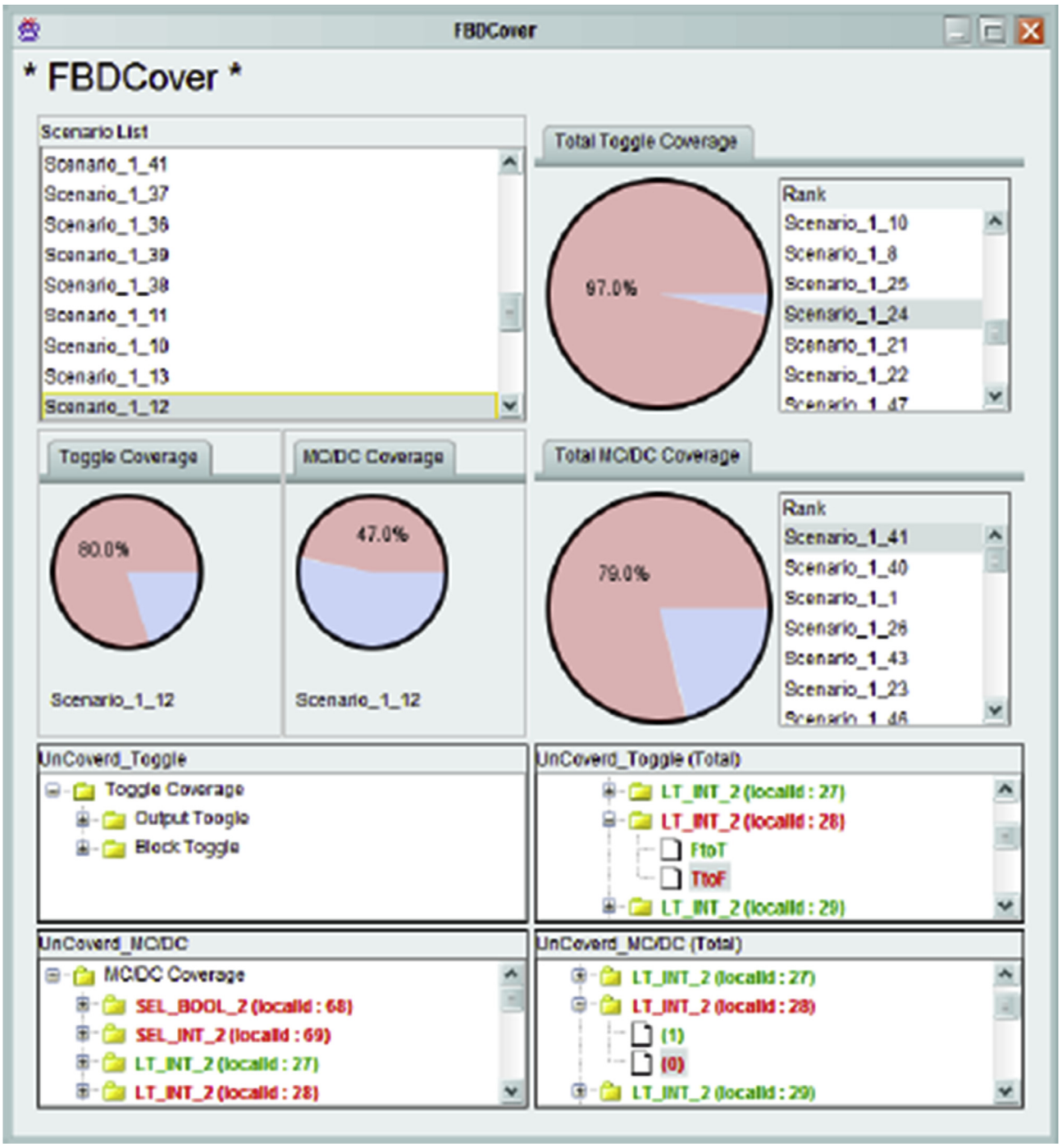

Fig. 6. A screen-dump of *FBDCover*.

Graphs indicate the results of the FBD simulation after the simulation by FBDSim. A list of all simulation scenarios is provided. Total coverages and basic coverages are depicted separately, and clicking an individual scenario shows the basic, i.e., block/output toggle coverages and block MC/DC coverage for the scenario. It also identifies a set of uncovered points in regard with the basic coverages.

Software engineers can use the quantitative information to refine an individual simulation scenario or to improve the total coverages of the whole set of FBD simulation scenarios.

Fig. 7 describes the toolchain of two supporting tools within the NuDE 2.0 framework. FBD Editor reads, edits and stores an FBD program of PLCopen TC6 XML format, and FBD Scenario Generator reads the FBD program and produces a set of simulation scenarios. FBDSim reads the FBD program and FBD simulation scenarios and executes the program with the scenarios in batches. FBDCover is seamlessly executed to display the quantitative information of coverages in graphics.

The toolchain in Fig. 7 is an extension of the integrated software testing (simulation) framework (IST-FPGA) for FPGA-based digital

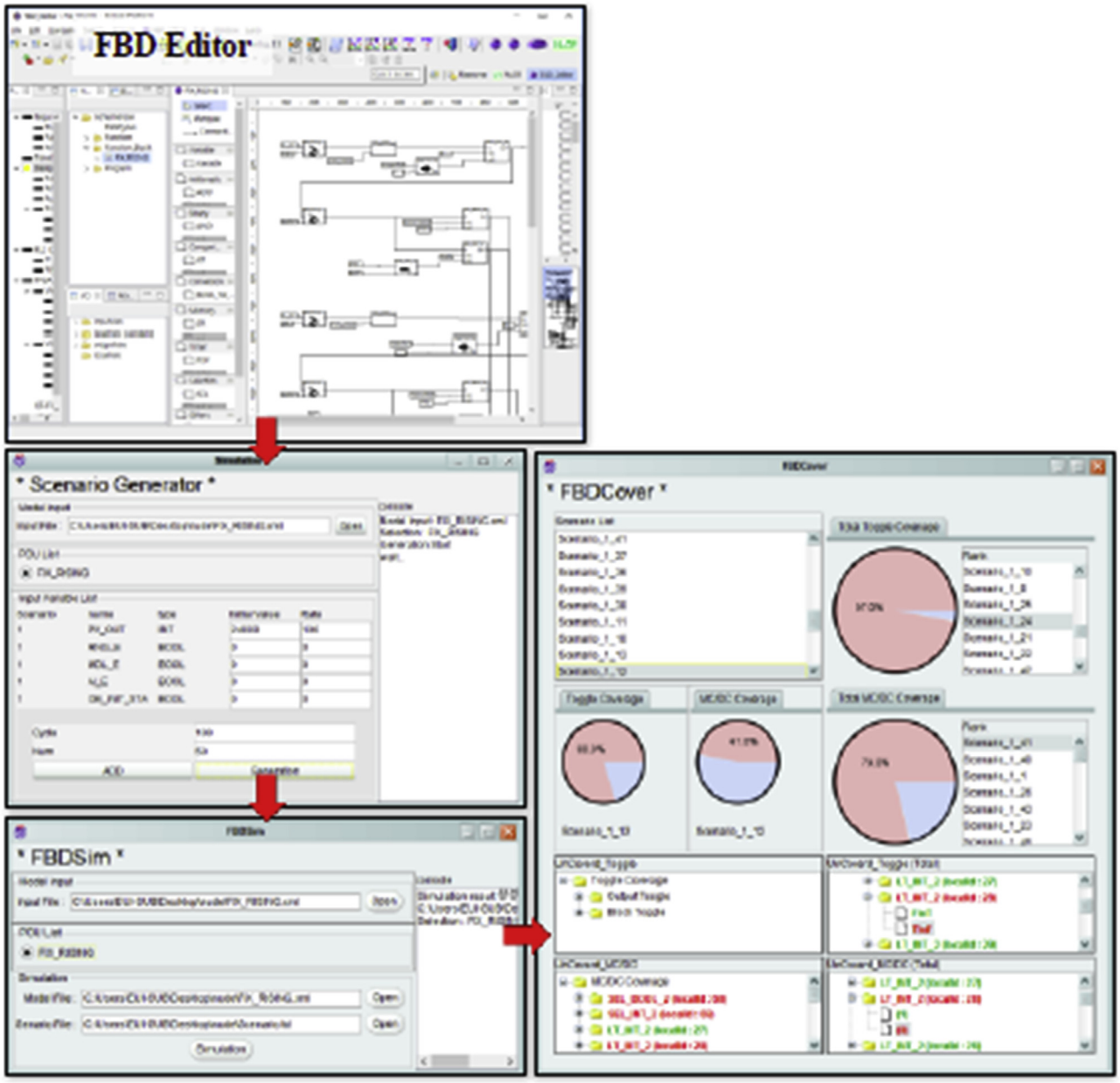

Fig. 7. A toolchain of the supporting tools with the FBD Simulator.

controller development [33], as shown in Fig. 8. The IST-FPGA suggests integrating the three independently performed simulation-based software test into one, and step 1 tries to develop common test oracles of Verilog/VHDL programs for other steps. Step 1 for Verilog/VHDL programs in the RTL design phase can be extended up to FBD programs as a design specification, and the toolchain which this paper proposes can make the integrated software testing framework start from the FBD programs of the design specification.

5. Experimental results

We performed experiments with FBD programs [34] for the second phase of KNICS APR-1400 RPS BP [35]. It was excerpted from an NPP (but, not officially final version) that is almost in commercial operation. It consists of 18 shutdown logics of FBD programs, of which we used 5 representative trip logics: *fixed set-point falling trip* (FFT), *fixed set-point rising trip* (FRT), *manual reset falling trip* (MFT), *variable set-point falling trip* (VFT), and *variable set-point rising trip* (VRT).

We initially generated scenarios so that each input variable, processing value (PV_OUT), has random values. Each program has a set of 1000 simulation scenarios, with 100 execution cycles, that are automatically generated by the FBD Scenario Generator. FBDSim simulated the FBD programs with the sets of simulation scenarios in batches. The scenarios having random PV_OUT values only achieved 11–15% of TC_T and 40–45% of $MCDC_T$. The second attempt designs the PV_OUT with a tendency to imitate the actual processing values. Fig. 9 shows 1000 value sequences of PV_OUT for FFT, which have a decreasing tendency. The FBD Scenario Generator can also automatically generate scenarios for which the input sequence has such tendencies. About 40% of the simulation

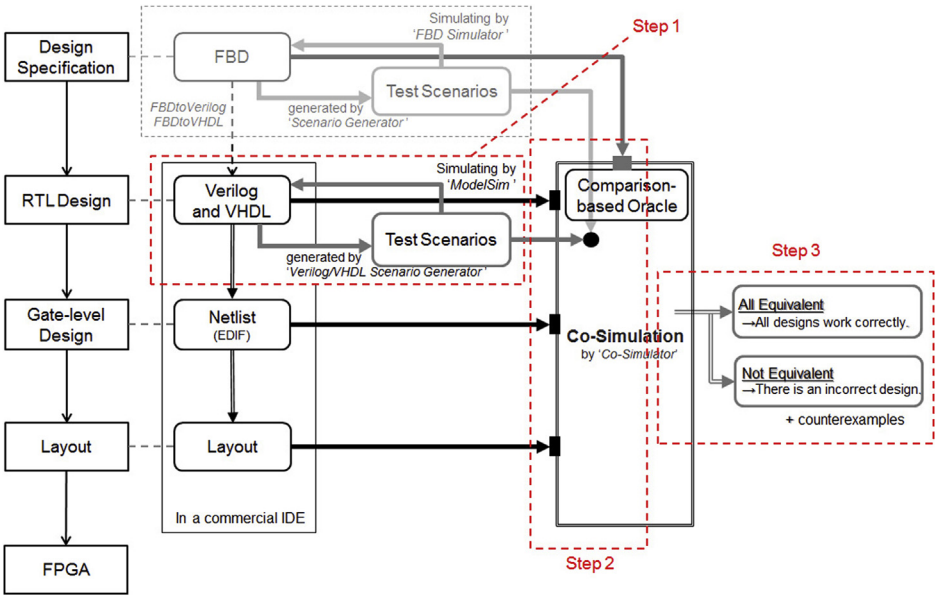

Fig. 8. IST_FPGA: an integrated software testing framework for FPGA [33].

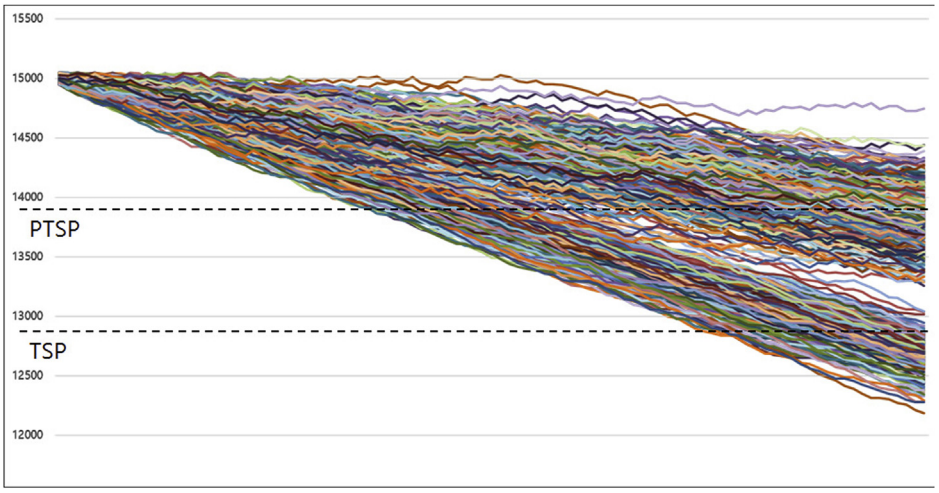

Fig. 9. Trajectories of the 1,000 scenarios generated for FFT [33].

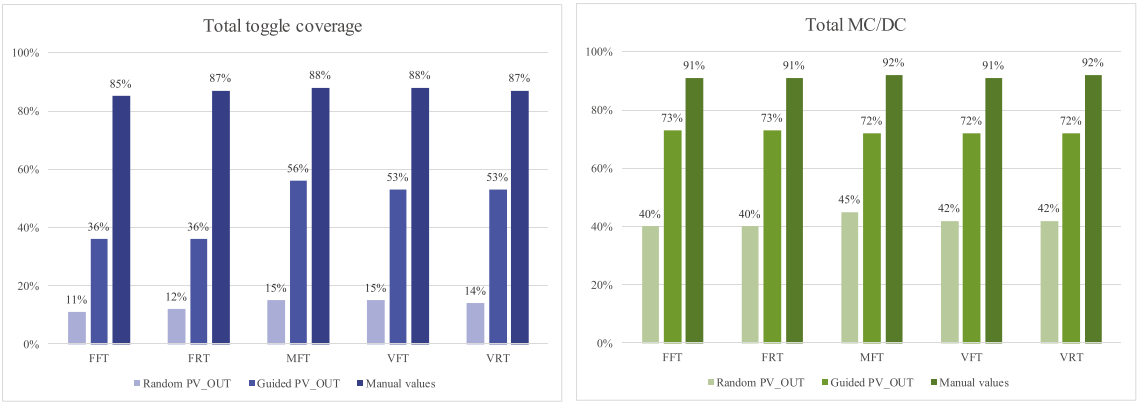

Fig. 10. TC_T and $MCDC_T$ of the 5 logics with 3 types of simulation scenarios.

scenarios generate the predefined trip signal. The scenarios including the given tendency achieved 36–56% of TC_T and 72–73%

of $MCDC_T$.

Finally, we performed an experiment with the simulation scenarios manually modified for more concrete evaluation. The trip logics have a processing value (PV_OUT) and also input variables for testing and error signals. The testing signal indicates the test run of system. A program never makes an output signal, such as a shutdown signal, having *true* value, even if the PV_OUT violates an allowable range in the case of a *true* testing signal. In addition, the system takes signals for module and channel error to notify that modules or channels are erroneous. It is difficult for engineers to simulate every possible situation, even if they understand the logics, requirements, and functions. The third experiment simulates the logics using the simulation scenarios of the second experiment in which the testing and error signals are modified manually.

Fig. 10 shows all the experimental results in which FBDCover measures the coverages according the five logics. The third experiment achieved 85–88% of TC_T and 91–92% of $MCDC_T$. The coverage gradually increased as the experiments proceeded. The first set of simulation scenarios may be ineffective, since the processing variable has meaningless values that are randomly generated. The second experiment simulated programs with the processing variable having values with a given tendency, such as increasing or decreasing, according to the programs. The third results achieved the highest percentage of both coverages, as we modified the simulation scenarios with specific purposes. TC_T and $MCDC_T$ could be improved because the simulation results demonstrated an insufficient number of simulation scenarios to execute the logics thoroughly. The simulation coverages indirectly indicate which cases are not simulated and how engineers need to improve the thoroughness of the simulation.

6. Conclusion and future work

This paper proposes two sets of structural coverage adequacy criteria for FBD simulations to indicate the structural coverage adequacy criteria., *i.e.*, the measures to estimate the thoroughness of functional FBD simulation scenarios according to the structural elements of FBD programs. The coverages are inspired by traditional software testing coverages, and we improve and modify the coverages for their application in multi-cycle simulation. We developed two supporting tools, *FBDSim* and *FBDCover*, to automate the measurement for numerous simulation scenarios. With the supporting tools, we performed the experimental study with FBD programs for RPS logics in NPPs. The experiment estimated the thoroughness of simulation scenarios. Simulation scenarios designed for verification purposes achieved 49% of TC_T and 72% of $MCDC_T$ on average, which are respectively 36 and 30 higher percent points than randomly generated ones. It was possible to increase them to 87% and 91% by manually modifying the simulation scenarios. The results of the experiment showed that both coverage criteria can be used to provide information on how thorough simulation scenarios are. Future research needs to carefully examine the effectiveness of the coverage criteria as measures to find faults or defects in an FBD program. We are presently conducting mutation analysis in order to demonstrate the effectiveness of the measures proposed herein.

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.

Acknowledgement

This paper was supported by Konkuk University in 2017.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.net.2020.06.017.

References

[1] E.-S. Kim, S. Jung, J. Kim, J. Yoo, MC/DC and toggle coverage measurement tool for FBD program simulation, in: Transactions of the Korean Nuclear Society Spring Meeting, Jeju, Korea, May 11-13, 2016.
[2] International Atomic Energy Agency (IAEA), Verification and Validation of Software Related to Nuclear Power Plant Instrumentation and Control, 1999. Technical reports series No. 384.
[3] R. Alur, A. Kanade, S. Ramesh, K. Shashidhar, Symbolic analysis for improving simulation coverage of simulink/stateflow models, in: Proceedings of the 8th ACM International Conference on Embedded Software, New York, USA, Oct 19-24, 2008.
[4] International Electrotechnical Commission (IEC), IEC 61131-3, Programmable Controllers: Part 3—Programming Languages, second ed., 2003.
[5] R.G. Sargent, Verification and validation of simulation models, J. Simulat. 7 (2013) 12–24.
[6] C. Schnakenbourg, J.-M. Faure, J.-J. Lesage, Towards IEC 61499 function blocks diagrams verification, in: IEEE International Conference on Systems Man and Cybernetics, Yasmine Hammamet, Tunisia, Oct 6-9, 2002.
[7] M. Pezze, M. Young, Software Testing and Analysis: Process, Principles and Techniques, Wiley, 2008.
[8] PLCopen Technical Committee 6, XML formats for IEC 61131-3, Ver. 2.01, Available: http://www.plcopen.org, 2009.
[9] IEC, IEC 61508 Functional Safety of Electrical, Electronic and Programmable Electronic (E/E/PE) Safety-Related Systems, 2000.
[10] IEC, IEC 60880 Nuclear Power Plants - Instrumentation and Control Systems Important to Safety—Software Aspects for Computer-Based Systems Performing Category A Functions, 2006.
[11] International Organization for Standardization (ISO), ISO 26262 Road Vehicles—Functional Safety, 2011.
[12] E. Jee, D. Shin, S. Cha, J.-S. Lee, D.-H. Bae, Automated test case generation for FBD programs implementing reactor protection system software, Softw. Test. Verif. Reliab. 24 (8) (2014) 608–628.
[13] Liverpool Data Research Associates (LDRA), LDRA tool suite, Available: http://www.ldra.com/.
[14] Esterel Technologies, SCADE - IEC 60880 compliant, Available: http://www.esterel-technologies.com/.
[15] J.H. Kim, D.Y. Oh, N.H. Lee, C.H. Kim, J.H. Kim, A nuclear safety system based on industrial computer, in: Transactions of the Korean Nuclear Society Spring Meeting, Taebaek, Korea, May 26-27, 2011.
[16] C. Park, C. Choe, S. Jin, An effective application process for code coverage analysis, in: Proceedings of the International Symposium on Future I&C for Nuclear Power Plants/International Symposium on Symbiotic Nuclear Power System 2014, Jeju, Korea, Aug 24-28, 2014.
[17] E. Jee, J. Yoo, S. Cha, Control and data flow testing on function block diagrams, in: Proceedings of the 24th International Conference on Computer Safety, Reliability and Security, Fredrikstad, Norway, Sep 28-30, 2005.
[18] E. Jee, J. Yoo, S. Cha, D. Bae, A data flow-based structural testing technique for FBD programs, Inf. Software Technol. 51 (7) (2009) 1131–1139.
[19] D. Shin, E. Jee, D.-H. Bae, Comprehensive analysis of FBD test coverage criteria using mutants, Software Syst. Model 15 (3) (2016) 631–645.
[20] Wikipedia, Software testing. https://en.wikipedia.org/wiki/Software testing.
[21] G.J. Myers, C. Sandler, T. Badgett, The Art of Software Testing, John Wiley & Sons, 2011.
[22] Institute of Electrical and Electronics Engineers (IEEE), IEEE Standard 1012 - IEEE Standard for System and Software Verification and Validation, 2012.
[23] IEEE, IEEE Standard 1028 - IEEE Standard for Software Reviews and Audits, 2008.
[24] ISO/IEC/IEEE, ISO/IEC/IEEE 29119-4 Software and System Engineering Software Testing Part 4: Test tTchniques, 2015.
[25] K. Maruchi, H. Shin, M. Sakai, MC/DC-like structural coverage criteria for function block diagrams, in: 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops, OH, USA, Mar 31-Apr 4, 2014.
[26] S. Tasiran, K. Keutzer, Coverage metrics for functional validation of hardware designs, IEEE Design Test Comput. 4 (2001) 36–45.
[27] Radio Technical Commission for Aeronautics, DO-178B Software Considerations in Airborne Systems and Equipment Certification, 1992.
[28] K.J. Hayhurst, D.S. Veerhusen, J.J. Chilenski, L.K. Rierson, A Practical Tutorial on Modified Condition/decision Coverage, NASA, 2001. TM-2001-210876.
[29] J. Yoo, E.-S. Kim, D.-A. Lee, J.-G. Choi, Y.J. Lee, J.-S. Lee, Nude 2.0: a model-based software development environment for the PLC & FPGA based digital systems

in nuclear power plants, in: Proceedings of the 2014 International Symposium on Integrated Circuits, Singapore, Dec 10-12, 2014.
[30] E.-S. Kim, D.-A. Lee, J. Kim, S. Jung, J. Yoo, J.-G. Choi, J.-S. Lee, NuDE 2.0: a formal-methods based software development, verification and safety analysis environment for digital I&Cs in NPPs, J. Comp. Sci. Eng. 11 (1) (2017) 9–23.
[31] D.-A. Lee, E.-S. Kim, Y.-J. Seo, J. Yoo, FBDEditor: an FBD design program for developing nuclear digital I&C systems, in: Proceedings of the 16th Korea Conference on Software Engineering, PyeongChang, Korea, Feb 12-14, 2014
[32] E.-S. Kim, D.-A. Lee, J. Yoo, The scenario generator for verifying the correctness of FBDtoVerilog translator, in: Proceedings of the Korea Information Processing Society, vol. 21, Ajou University, Korea, April 24-25, 2014.
[33] J. Kim, E.-S. Kim, J. Yoo, Y.J. Lee, J.-G. Choi, An integrated software testing framework for FPGA-based controllers in nuclear power plants, Nucl. Eng. Technol. 48 (2) (2016) 470–481.
[34] Korea Atomic Energy Research Institute (KAERI), Software Design Specification for Reactor Protection System, Rev.02, 2006. KNICS-RPS-SD231.
[35] Rev.00, KAERI, Software Requirements Specification for Reactor Protection System, 2005. KNICS-RPS-SRS221.