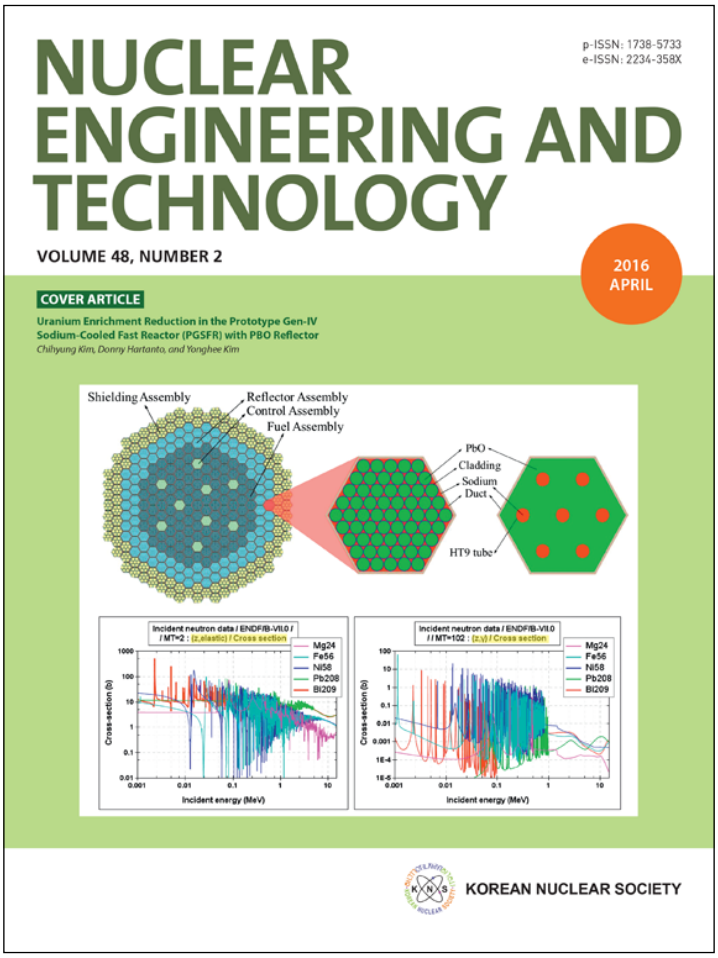
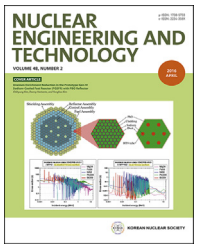


Nuclear Engineering and Technology 48 (2016) 470–481

Available online at ScienceDirect

Nuclear Engineering and Technology

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



Original Article

An Integrated Software Testing Framework for FPGA-Based Controllers in Nuclear Power Plants☆

Jaeyeob Kim [a], *Eui-Sub Kim* [a], *Junbeom Yoo* [a,*], *Young Jun Lee* [b], *and Jong-Gyun Choi* [b]

[a] *Division of Computer Science and Engineering, Konkuk University, 1 Hwayang-dong, Gwangjin-gu, Seoul, 143-701, Republic of Korea*

[b] *MMIS Lab., Korea Atomic Energy Research Institute, 989-111 Deadeok-daero, Yuseong-gu, Daejeon, 305-353, Republic of Korea*

ARTICLE INFO

Article history:
Received 1 September 2015
Received in revised form
28 November 2015
Accepted 3 December 2015
Available online 21 January 2016

Keywords:
Co-simulation
FPGA
Simulation
Testing
Verification

ABSTRACT

Field-programmable gate arrays (FPGAs) have received much attention from the nuclear industry as an alternative platform to programmable logic controllers for digital instrumentation and control. The software aspect of FPGA development consists of several steps of synthesis and refinement, and also requires verification activities, such as simulations that are performed individually at each step. This study proposed an integrated software-testing framework for simulating all artifacts of the FPGA software development simultaneously and evaluating whether all artifacts work correctly using common oracle programs. This method also generates a massive number of meaningful simulation scenarios that reflect reactor shutdown logics. The experiment, which was performed on two FPGA software implementations, showed that it can dramatically save both time and costs.

1. Introduction

Programmable logic controllers (PLCs) [2] are widely used to implement safety-critical systems for digital instrumentation and control (I&C) of Nuclear power plants (NPPs). The increasing complexity of newly developed systems and maintenance costs are now demanding more powerful and cost-effective implementation, such as field-programmable gate arrays (FPGAs) [3]. The nuclear industry is now eagerly researching FPGA-based digital I&Cs [4–6] to replace PLC-based systems. In turn, international standards [7–9] require more rigorous demonstrations of the safety of these new systems.

The FPGA software, which this paper concerns, was first modeled with hardware description languages (HDLs), such as Verilog and VHSIC HDL (VHDL), by software designers manually, and then subsequently synthesized into gate-level designs and physical layouts by software synthesis tools provided by FPGA vendors (e.g., ISE Design Suite (Xilinx, San Jose, CA, USA) [10], Quartus Prime (Altera, San Jose, CA, USA) [11], and Libero SoC (Microsemi, Aliso Viejo, CA, USA) [12]).

☆ A preliminary version of this paper was presented at the Transactions of the Korean Nuclear Society Autumn Meeting, Pyeongchang, Korea, in 2014 [1].

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

http://dx.doi.org/10.1016/j.net.2015.12.008

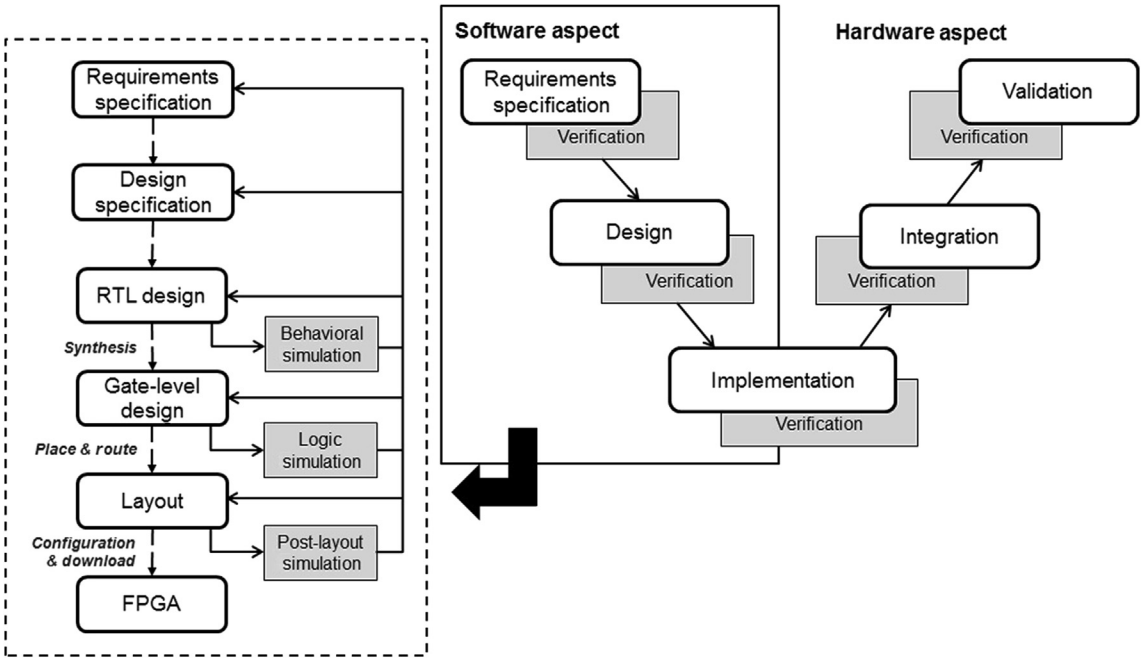


Fig. 1 – The V-shaped life cycle and a typical FPGA development process. FPGA, field-programmable gate arrays; RTL, register-transfer level.

FPGA tools make the synthesis process fully automatic, and software designers largely focus on HDL designs to implement FPGA requirements correctly.

FPGA software designers also use verification techniques, such as "simulation" [13–15], in order to check if high-level designs are correctly synthesized into low-level ones. At each step [i.e., register-transfer level (RTL), gate-level, and layout], designers perform three common activities. They first develop test scenarios, then simulate each target in a test bench, and finally evaluate (i.e., observe) the simulation results against specified requirements. The problem on which this study focused is that the verification activity should be performed at each step individually and repetitively. Furthermore, the individual preparation for each verification step, such as developing test scenarios and test benches, takes considerable time and money.

This paper proposed an integrated software testing framework for FPGA software developments (IST-FPGA). It allows us to perform the three activities of the simulation-based verification only once and in one step. For all design artifacts at every step, it generates common and meaningful test scenarios mechanically, simulates all designs simultaneously, and finally evaluates the simulation results against expected ones altogether. If any one of the designs show different (i.e., incorrect) behavior from the expected one (i.e., a comparison oracle program), IST-FPGA analyzes and compares the incorrect case in detail. IST-FPGA is also supported by CASE tools, such as Verilog/VHDL Scenario Generator and Co-simulator.

In order to demonstrate the effectiveness of IST-FPGA, we performed an experiment with two FPGA-based I&C systems that are under development by the Korea Atomic Energy Research Institute (Daejeon, Korea). This experiment successfully demonstrated how IST-FPGA can reduce the time and cost for the simulation-based verification of FPGA software. The remainder of the paper is organized as follows: Section 2 provides background information on FPGA verification and simulation techniques. Various standards and guidelines for developing and verifying FPGA-based digital I&Cs are briefly surveyed. Section 3 proposes the integrated software testing framework for FPGA as well as assisting tools we developed. Experiment results are presented in Section 4, and Section 5 surveys related research. Section 6 concludes the paper and provides remarks on future research extension and direction.

2. Background

2.1. *FPGA development and software verification*

The system development life cycle of FPGA-based I&Cs should follow IEC-61513 [9]. An FPGA-based system has a specific feature that the portion of the development life cycle that uses HDL be classified as software, then once it is downloaded to a chip, it is classified as hardware. FPGA, therefore, should be developed to meet both IEC-60880 [8] in terms of software and IEC-60987 [16] in terms of hardware. Fig. 1 depicts the V-shaped life cycle of FPGA development explained in IEC-62566 [17], consisting of software and hardware aspects. The software aspect also has a typical development life cycle [18] presented on the left-hand side of the figure.

At each step of the FPGA software development life cycle, designers perform a simulation-based verification in order to confirm that each artifact satisfies its required specification. The first simulation on RTL designs, called behavioral simulation, aims to confirm that all requirements are implemented

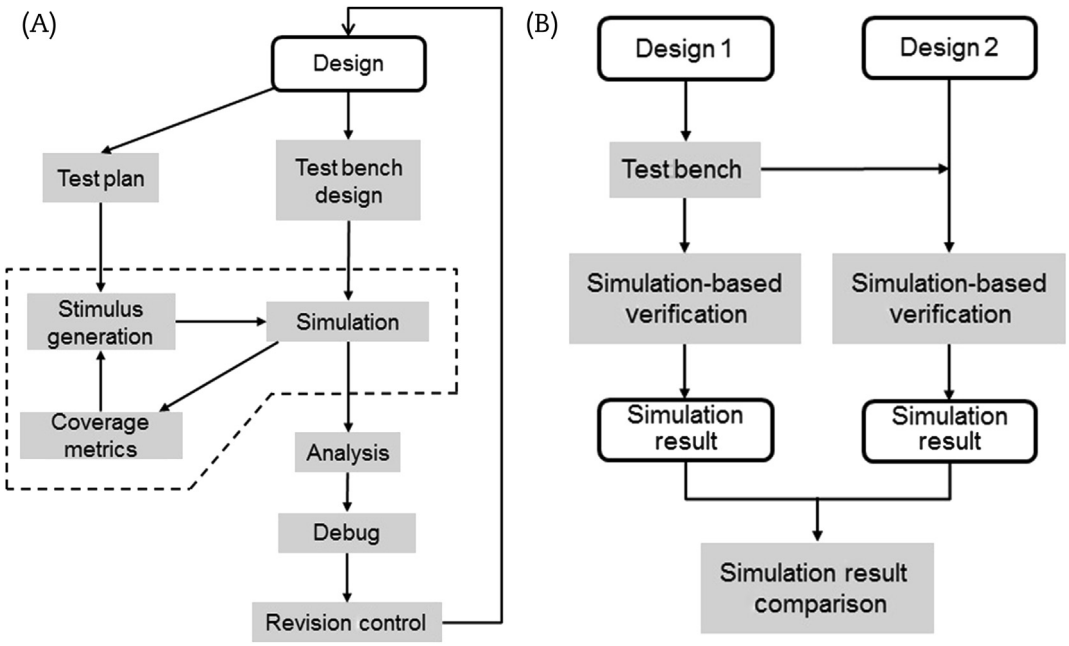


Fig. 2 – The simulation-based verification technique. (A) Typical simulation-based verification and (B) typical co-simulation.

into the RTL design correctly. As most designers develop RTL designs manually, this takes a significant amount of time for the behavioral simulation. After logic synthesis from RTL to gate-level design, designers perform a logic simulation in order to confirm that functionalities were preserved during synthesis. After place and route, they can validate the layout via a post-layout simulation to check that the layout meets all timing requirements. All simulation-based verifications at each step are performed individually and repetitively by experienced engineers, and are also considered to be one of key factors for efficient FPGA development.

2.2. *Simulation-based verification*

"Simulation-based verification" is a traditional method [15] that can be applied at any design level, be it RTL, gate-level, or layout. Fig. 2A depicts a typical flow of the simulation-based verification. Static error, potential error, and coding style guideline violations are checked through the linter program [19] beforehand. We then need to prepare a test plan determining what scenarios are used in the simulation. Simulations are performed using a test bench, and we also need to measure code coverages [20] in order to evaluate the quality of the simulations performed. If the coverage value measured does not reach an expected value, the portion of the dotted line in Fig. 2A needs to be performed repeatedly. Debugging can be performed on the basis of the analyzed information. This entire verification process is repeated until the simulation result has no debugging issues.

"Co-simulation" is a verification method that is extended from that described above. The basic idea of co-simulation is that if two designs generate the same outputs with the same inputs, both designs will perform the same operations [21]. Fig. 2B depicts a typical co-simulation process. The two designs are simulated in a suitable way and then all simulation results are compared. Two designs at any level can be used, e.g., two Verilog programs [22], a Verilog and C program [23], or a Verilog and FPGA hardware implementation [24]. Questa ADMS (Mentor Graphics, Wilsonville, OR, USA) [25] extends the Questa verification platform to provide a unified simulation environment that can co-simulate various designs at any level. If significant outputs in all simulation results are the same in every case, it can be considered that the two designs perform the same actions, at least for the simulation scenarios.

2.3. *Standards and guidelines for FPGA design verification*

The development and verification of FPGA-based safety-level controllers in nuclear power plants is an active research topic around the globe. For now, the standard discussing our concern is IEC 62566 [17]: "Nuclear Power Plants - Instrumentation and control important to safety - Development of HDL programmed integrated circuits for systems performing category A functions." Various organizations and research groups are now working to define more detailed guidelines and standards [26,27].

System safety assessment and the design life cycle are addressed in several regulations, such as DO-254 [28], IEEE 1012 [29], and IEC 61508 [30], and they can be used for FPGA-based systems as a general guide. More specific verification and validation processes, configuration management, and the development life cycle used in safety systems in nuclear power plants are addressed in IEEE 7-4.3.2 [31], IEEE 603 [32], IEC 62566 [17], and IEC 60880 [8]. Several Electric Power Research Institute (EPRI; Palo Alto, CA, USA) guidelines (TR-1019181/109390/1022983 [33–35]) try to guide the use of FPGA in NPP I&C systems, while NUREG/CR-7006 [18] provides a review guideline that can support acceptance or licensing processes.

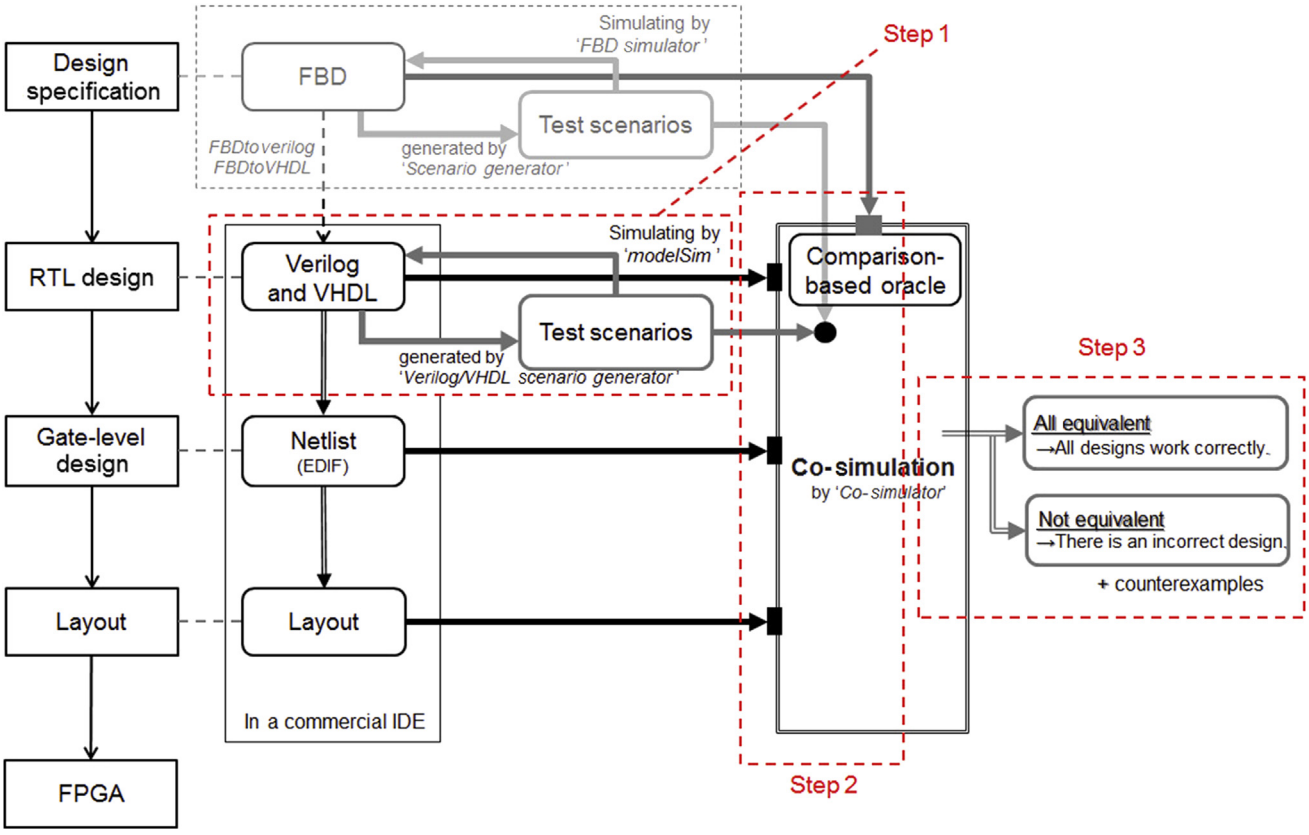


Fig. 3 – An integrated software testing framework for FPGA (IST-FPGA). EDIF, electronic design interchange format; FBD, function block diagram; FPGA, field-programmable gate arrays; IDE, integrated development environment;IST, integrated software testing; RTL, register-transfer level; VHDL, VHISC hardware description languages.

3. IST-FPGA

3.1. *Overview*

IST-FPGA is an integrated software testing framework for FPGA-based digital I&Cs in NPPs. As illustrated in Fig. 3, it suggests three steps of co-simulation-based software verification, namely: Step 1 - Preparation, preparing a common oracle program with scenarios; Step 2 - Co-simulation, co-simulating all designs simultaneously; and Step 3 - Evaluation: evaluating simulation results all together.

3.1.1. *Step 1—Preparation*

IST-FPGA first prepares one common oracle program (i.e., a correct answer) and confirms its correct functioning against required specifications. It co-simulates all designs simultaneously in order to check their behavioral equivalence instead of performing simulations and evaluating the results individually at each level. Therefore, it requires designers to evaluate simulation results only once. IST-FPGA supports three kinds of common oracles, such as a function block diagram (FBD) [2] and Verilog and VHDL programs. If we use Verilog or VHDL programs, IST-FPGA mechanically generates a number of meaningful scenarios through the Verilog Scenario Generator and the VHDL Scenario Generator. It also generates a test bench that can execute the scenarios upon Verilog/VHLD-Netlist-Layout programs seamlessly with ModelSim (Mentor Graphics, Wilsonville, OR, USA) [36]. We then simulate the Verilog/VHDL program with the scenarios through ModelSim in order to verify that it works correctly. This is a manual verification performed by experienced engineers and often constitutes a large part of the entire FPGA software verification process. After confirming correct functioning, we use it as a common oracle program with which to co-simulate.

3.1.2. *Step 2—Co-simulation*

IST-FPGA then co-simulates all designs (i.e., Verilog/VHDL, Netlist, and Layout) with the test scenarios simultaneously with support of the Co-simulator. As a Verilog/VHDL program is used as a correct answer (a common test oracle), all other designs should show the same behaviors as the Verilog/VHDL program. IST-FPGA uses ModelSim to simulate each design, and the test benches generated by Verilog/VHDL Scenario Generator are used to execute the simulator with a large number of scenarios systematically and seamlessly.

3.1.3. *Step 3—Evaluation*

IST-FPGA finally evaluates all co-simulation results and judges whether all designs work correctly or not. If all designs show the same behaviors, i.e., the same outputs against all test scenarios, it concludes that all designs work correctly. If any design shows different behavior from the common oracle program, it shows the case in detail to analyze when the design works differently.

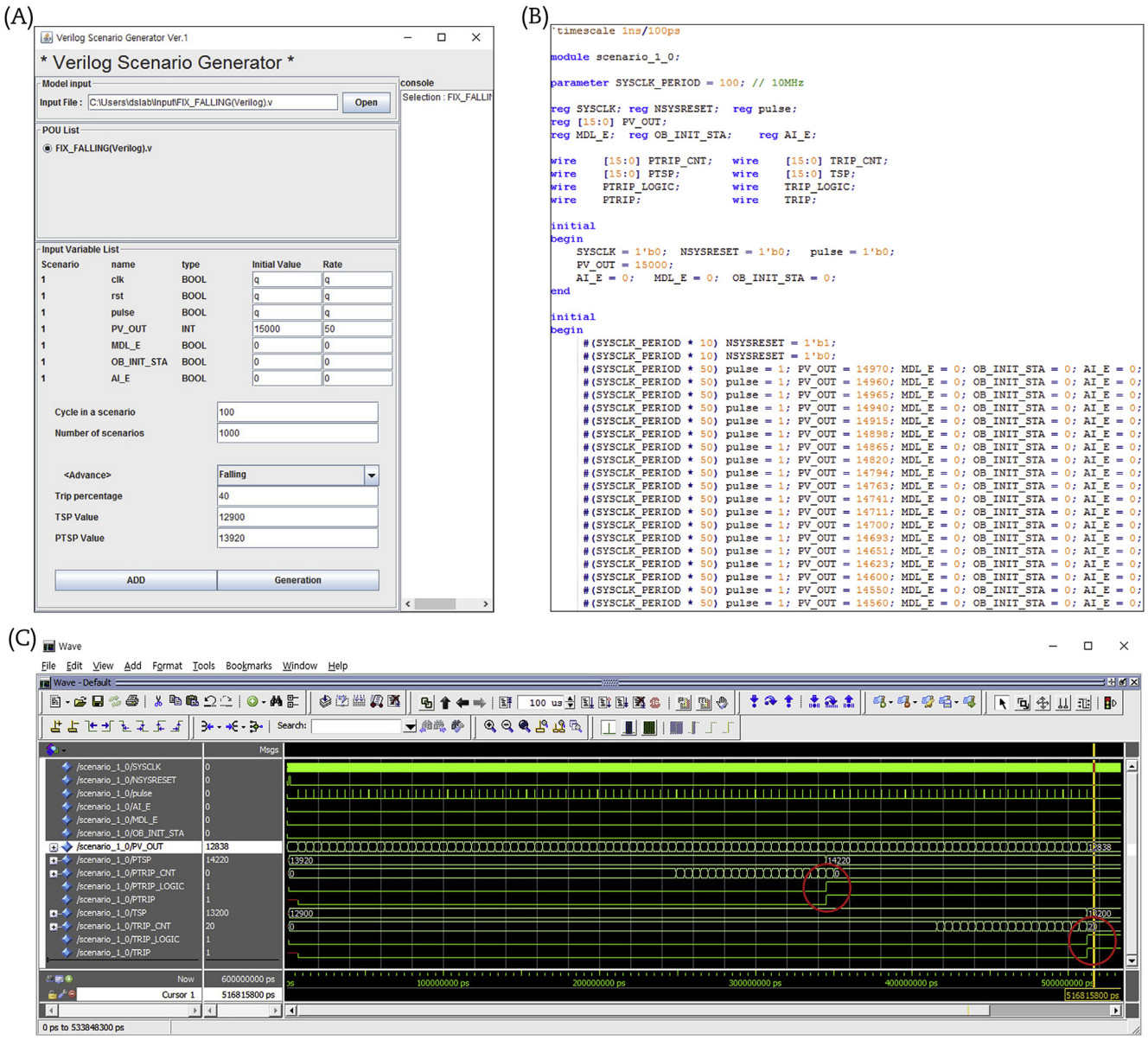


Fig. 4 – The simulation using Verilog Scenario Generator and ModelSim. (A) Verilog Scenario Generator. (B) Scenario (test bench) generated. (C) The Verilog simulation with ModelSim.

It is worth noting that we can use an FBD program at the level of design specification as a common oracle program, as proposed by the NuDE framework [3]. An FBD program is more intuitive to understand and analyze as compared with RTL languages, and we also have many systems designed with FBD working correctly. The NuDE framework, therefore, proposes to use FBD programs as design specifications and to generate RTL programs mechanically from the FBD programs. We can generate a number of meaningful test scenarios (i.e., as Scenario Generator, which this paper proposes) from the FBD program with support of FBD Scenario Generator, and also generate two test benches that can execute the scenarios on FBD programs and Verilog/Netlist/Layout programs, respectively. We then simulate them with FBD Simulator [37] to confirm that they work correctly and can co-simulate them simultaneously.

3.2. *Supporting tools for IST-FPGA*

Verilog/VHDL Scenario Generator generates a number of meaningful scenarios mechanically from Verilog/VHDL programs and prepares test benches to seamlessly simulate the programs with generated scenarios through ModelSim. Fig. 4A shows the Verilog Scenario Generator, generating 1,000 scenarios of 100 cycles from a Verilog program implementing Fixed Set-point Falling Trip Logic in the reactor protection system (RPS). The tool first reads a Verilog (or VHDL) program and requests more information about the program from designers, such as initial values and rates of change for all input

variables, a trip set-point, and the percentage of trip situations in all generated scenarios (this has now been customized into the features of the RPS). The number of execution cycles for each scenario and the total number of scenarios to generate are also required. Fig. 4B shows an example of the scenario (i.e., test bench) generated by Verilog Scenario Generator that a simulator ModelSim can simulate seamlessly, as shown in Fig. 4C.

We can confirm the correctness of the Verilog program against the generated scenarios by inspecting the simulation results carefully. For example, the fixed set-point falling trip logic simulated in Fig. 4C has an important input (PV_OUT) that starts from 15,000 and changes at the maximum rate of 50. If the trip condition PV_OUT $\leq$ TSP is satisfied for 20 cycles (TSP_CNT), then the shutdown signal will be initiated (i.e., TRIP = 1), as shown in the simulator. The simulation in Fig. 4C shows the situation when the pre-trip was first activated, followed by the trip. The current input value of PV_OUT is 12,838, while PTSP is 14,220 = 13,920 + 300 (hysteresis) and TSP is 13,200 = 12,900 + 300. We modified the real value of trip/pre-trip set-points to aid understanding. The pre-trip (PTRIP) is a warning signal before the actual alarm, TRIP, and also shows the same behavior as the trip signal, but works earlier in advance of the trip signal. After confirming its correct behavior through careful reviews of simulation results by experts, the program (i.e., a Verilog program in this example) will be used as a correct answer (i.e., a comparison oracle program) for all other design artifacts.

It is worth noting that the scenarios generated by Verilog/VHDL Scenario Generator try to reflect the physical conditions for the RPS trip logic. Fig. 4A shows that we set information on all input variables, as well as auxiliary information, such as falling trip and trip set-point (12,900). It also set the tool to generate scenarios, in which ~40% would generate the trip signal. Fig. 5 shows the trajectories of 1,000 scenarios generated, i.e., 1,000 trajectories of the input variable PV OUT during 100 cycles.

The Co-simulator plays the role of an integrated controller in IST-FPGA, and allows designers to perform the three steps of IST-FPGA seamlessly and mechanically. Step 1 - Preparation for a common oracle program with massive and meaningful scenarios is the same as the Verilog/VHDL Scenario Generator, except that it can also generate and execute scenarios for FBD programs. Three designs, such as Verilog, VHDL, and FBD programs, can be used as candidates for the comparison oracle program at the first step, as shown in Fig. 6A.

Step 2 - Co-simulation of all designs simultaneously requires all input designs, such as FBD, Verilog, VHDL, netlist (EDIF), and post-layout, as well as required libraries, as presented in Fig. 6B. It then performs the co-simulation of all input designs with the simulation scenarios generated in the previous step by executing ModelSim or FBD Simulator in the background.

Co-simulator also supports Step 3 - Evaluation of simulation results altogether. If it detects any inequivalent case against the oracle program in co-simulation, it explains all simulation traces graphically (i.e., produces a counterexample of model-checking techniques [38] to aid the analysis, such as, "Where and when does the inequivalent occur?"). For example, the graph in Fig. 6C shows that the netlist shows different/incorrect behavior in the 59th cycle. The graphical counter example helps designers analyze the incorrect behavior of a specific design and localize errors in the design (this example used a seeded error in netlist design).

FBD Scenario Generator and FBD (Massive) Simulator are specialized tools for FBD programs. The NuDE framework [3,39] proposes the use of FBD programs as design specifications for RTL designs, and the FBD Scenario Generator generates a number of meaningful scenarios, while the FBD (Massive) Simulator simulates them independently from the software engineering tools of PLC vendors [38]. We notice that the trace of the FBD program shown in Fig. 7 is easier to understand as compared to the wave of ModelSim in Fig. 4C, while they all show the same behavior.

4. Experimental results

We performed an experiment with two examples of software implementations (i.e., Verilog and VHDL programs) that are under development as new FPGA-based digital I&Cs in Korea. We demonstrated how IST-FPGA can reduce the time and cost of the simulation-based verification of FPGA software. Fig. 8 provides an overview of the entire process and the experimental targets. The Verilog program of Case I was developed to confirm the functionalities (trip logics) of the new FPGA-based RPS [5], and consisted of 18 independent shutdown logics/modules with no hardware-dependent implementation details. The VHDL program of Case II, however, was the program used to synthesize, configure, and download into the FPGAs of the diverse protection system. It includes four shutdown logics as well as FPGA HW details such as I/Os.

For both the Verilog/VHDL programs, we performed the FPGA software development using Libero SoC and Synplify Pro (Synopsys, Mountain View, CA, ISA) [40]. As shown in Fig. 8, IST-FPGA performed the simulation-based verification with support of Co-simulator. We estimated the time taken for the IST-FPGA based verification and also compared it with the calculated/estimated times of traditional approaches performed individually and independently, as shown on the left-hand side of Fig. 8. The equation for calculating the estimated time taken for each verification approach is as follows:

$$E_{IST-FPGA} = \sum_{i=1}^{N} E_{IST-FPGA}(i), \tag{1}$$

where:

$$E_{IST-FPGA}(i) = E_{Step\ 1}(i) + T_{Step\ 2+Step\ 3}(i)$$

$$E_{Step\ 1}(i) = T_{Scenario\ Generation}(i) + E_{MO}(i)$$

$$E_{MO}(i) = eMO(i) \times NS$$

$$T_{Step\ 2+Step\ 3}(i) = T_{Co\text{-}Simulation}(i)$$

where: N = the number of logics/modules ($1 \leq i \leq N = 18$); $E(i)$ = estimated time; NS = the number of scenario (NS = 1,000); and e = meaning that we measured it with average time to estimate.

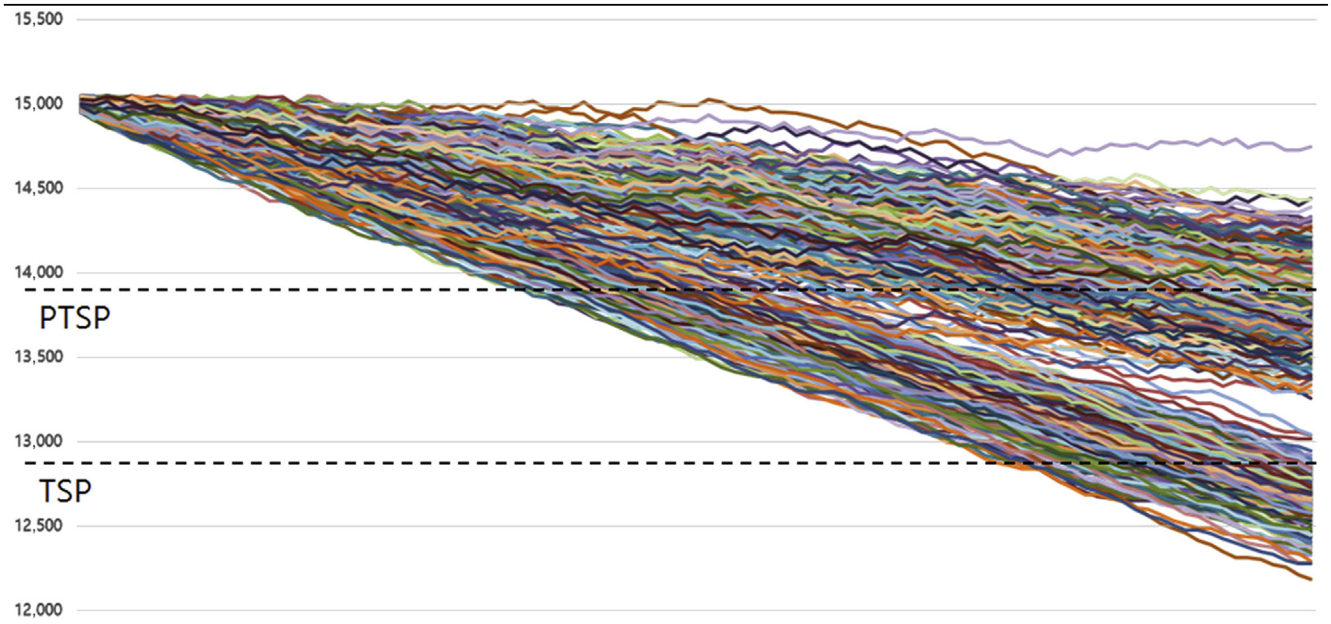


Fig. 5 – Trajectories of the 1,000 scenarios generated for a fixed set-point falling trip logic. PTSP, pretrip setpoint; TSP, trip setpoint.

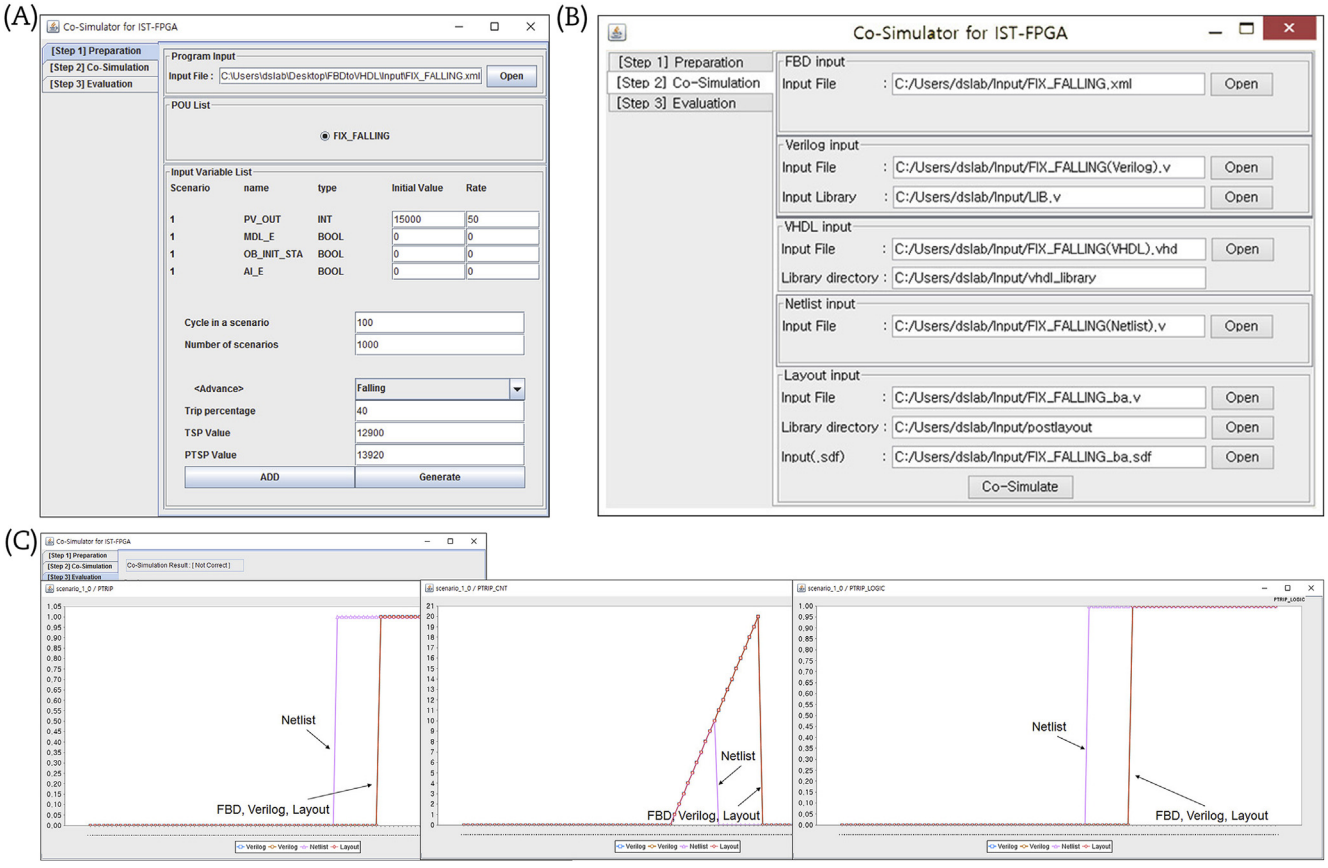


Fig. 6 – Co-simulation for IST-FPGA. (A) Step 1: Preparation. (B) Step 2: Co-simulation. (C) Step 3: Evaluation. FPGA, field-programmable gate arrays; IST, integrated software testing.

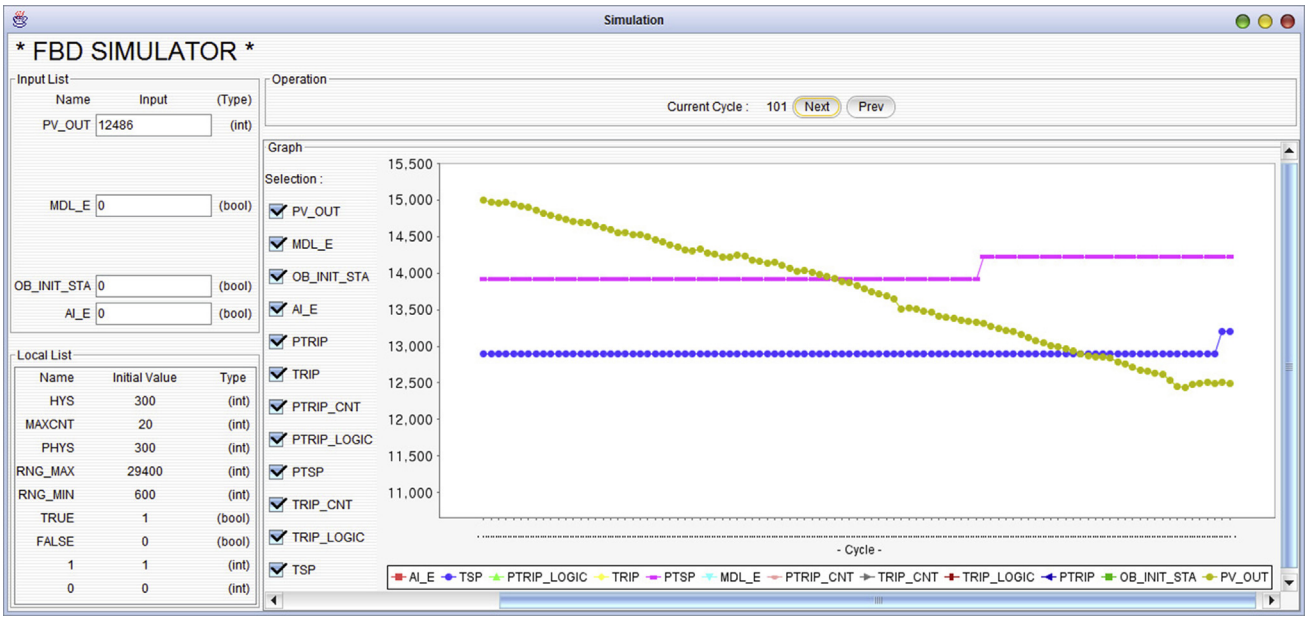


Fig. 7 – FBD simulator. FBD, function block diagram.

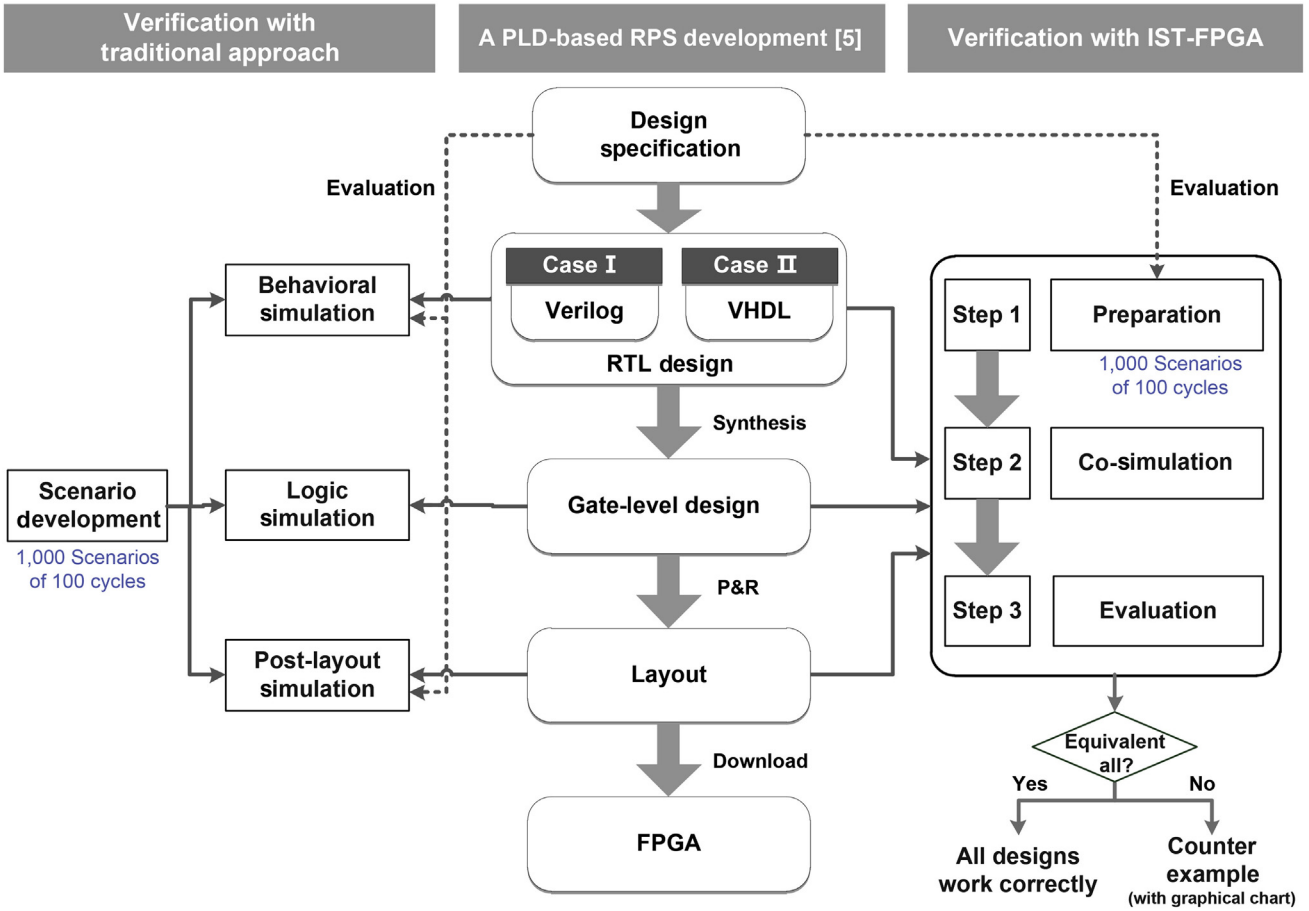


Fig. 8 – Experimental overview. FPGA, field-programmable gate arrays; IST, integrated software testing; P&R, place and route; PLD, programmable logic device; RPS, reactor protection system; RTL, register-transfer level; VHDL, VHISC hardware description languages.

In IST-FPGA, Step 1 consists of scenario generation and making the oracle (MO). Verilog/VHDL Scenario Generator or Step 1 of the Co-simulator can perform the scenario generation, and we can measure the exact executing time. However, MO is performed manually and depends upon the complexity of the test scenario generation and the experience of the experts. Therefore, we estimated $eMO(i)$ as the average time of 10 samples. Steps 2 and 3 are supported by the Co-simulator, and the executing time can be measured accurately.

$$E_{TA} = \sum_{i=1}^{N} E_{TA}(i), \tag{2}$$

where:

$$E_{TA}(i) = E_{SD}(i) + T_{Simulation}(i) + E_{Evaluation}(i)$$

$$E_{SD}(i) \approx T_{Scenario\ Generation}(i)$$

$$T_{Simulation}(i) = T_{BS}(i) + T_{LS}(i) + T_{PS}(i)$$

$$E_{Evaluation}(i) \approx E_{MO}(i) \times 3.$$

The time for the traditional approach (TA) can be estimated as shown above. Scenario development (SD) time is regarded in the same way as with the IST-FPGA, since Verilog/VHDL Scenario Generator can do it mechanically. The simulation time, including behavioral simulation (BS), logic simulation (LS), and post-layout simulation (PS) for three steps are measured by ModelSim. The evaluation time for three simulation results can reasonably be regarded as the three times of $E_{MO}(i)$ in IST-FPGA.

4.1. *Case I—The Verilog program*

The Verilog program consists of 18 logics/modules, as summarized in Table 1. For example, the Hi_CNY_PRS_FSR logic (module) has 34 inputs and two outputs while using 36 reg variables to store information for later steps. As most modules were developed to confirm the functionality of shutdown logics, they have two outputs: trip and pre-trip. Implementation details, such as operational bypass and heartbeat, are not implemented in the Verilog program, while they are in the VHDL program.

Table 1 – Summarized information for each module and measured time in Case I. SD, scenario development; pi, primary input; po, primary output; reg, register.

	Name	pi	po	reg	IST-FPGA : Step 1 / Step 2 + Step 3 Traditional approach : SD / Simulation / Evaluation (min)
1	Hi_CNT_PRS_FSR	34	2	36	396.53 / 1076.31
2	Hi_Local_Power_Density_OF	5	2	5	305.85 / 817.51
3	Hi_Log_Power_FSR	37	3	45	413.93 / 1124.93
4	Hi_PZR_Pressure_FSR	34	2	40	403.31 / 1096.26
5	Hi_SGL1_NR_FSR	34	2	40	407.32 / 1101.49
6	Hi_SGL2_NR_FSR	34	2	40	403.97 / 1098.43
7	HIHI_CPRS_NR_FSR	34	2	40	409.70 / 1110.61
8	Lo_DNBR_OF	5	2	5	303.68 / 815.53
9	Lo_PZR_Pressure_MRF	38	4	212	575.83 / 1585.79
10	Lo_RC1_FLW_VSF	50	2	200	515.77 / 1426.59
11	Lo_RC2_FLW_VSF	50	2	200	516.78 / 1427.52
12	Lo_SG1_PRS_MRF	36	2	203	554.77 / 1531.76
13	Lo_SG2_PRS_MRF	36	2	203	557.95 / 1534.67
14	Lo_SGL1_ESF_FSF	34	2	40	407.53 / 1101.68
15	Lo_SGL1_RPS_FSF	34	2	40	408.58 / 1102.64
16	Lo_SGL2_ESF_FSF	34	2	40	407.47 / 1101.63
17	Lo_SGL2_RPS_FSF	34	2	40	407.64 / 1101.78
18	Variable_OverPower_VSR	50	2	152	735.76 / 2079.64
	Average	34.06	2.17	87.83	451.80 / 1235.27

Table 2 – Summarized information for each module and measured time in Case II. SD, scenario development; DPS, diverse protection system; pi, primary input; po, primary output; reg, register.

	Name	pi	po	reg	IST-FPGA : Step 1 / Step 2 + Step 3 Traditional approach : SD / Simulation / Evaluation (min)
1	DPS_BP1	50	104	2815	1605.68 / 4611.46
2	DPS_BP2	50	104	2816	1608.58 / 4620.49
	Average	50	104	2815.50	1607.13 / 4615.98

We performed two simulation-based verifications, the IST-FPGA and the TA, and measured the time taken for each according to the formulae above. We generated 1,000 scenarios of 100 cycles, and the values of $eMO(i)$ were estimated by experts as an average of 10 real evaluation times of simulation results. For example, $eMO(1)$ for the Hi_CNY_PRS_FSR module was estimated as 0.33 minutes, and $E_{MO}(1)$ will be 0.33 * 1,000.

Since $E_{MO}(i)$ takes a larger portion of total verification time (up to 90%), the number of times the action has to be executed plays an important role in reducing the total verification time. While the IST-FPGA performs the activity only one time at Step 1 (Preparation), TAs have to do it for each evaluation step and will take threefold longer than IST-FPGA. In summary, IST-FPGA could perform the simulation-based test 2.73-times faster than TAs, even when executing and evaluating the same set of test scenarios.

4.2. *Case II—The VHDL program*

The VHDL program includes four shutdown logics, as well as implementation details, such as interfaces for FPGA hardware, as summarized in Table 2. For example, the DPS_BP 1 module uses 50 inputs (bits), 104 outputs (bits), and 2,815 reg variables (bits). Implementation details, such as operational bypass and heartbeat, were set appropriately beforehand.

We performed two simulation-based verifications and measured (estimated) the time using the same methods as in Case I. We used VHDL Scenario Generator instead of Verilog Scenario Generator. The results of Case II were similar to those of Case I, but it took much more time, as the VHDL program consists of more complex and larger scales of logics. In summary, IST-FPGA could perform the simulation-based test 2.87-times faster than TAs under the same conditions.

4.3. *Experiment analysis*

The experiment claimed that IST-FPGA is approximately threefold faster than traditional simulation-bases testing approach. However, the time and cost of the TA greatly depend on the experience and ability of developers and test engineers. As we assume that the two approaches use the same simulation scenarios mechanically generated by Co-simulator, the time taken to develop test scenarios for TAs may be underestimated. It must take a considerable amount of time for developers to develop such test scenarios. However, they use a massive number of simulation scenarios (e.g., 1,000), and the time taken to evaluate simulation results may be overestimated, even if not multiplied by three. Such massive numbers of test scenarios cannot be developed nor used in practice.

The heart of the question is how to guarantee the quality of simulation scenarios generated by Co-simulator or Verilog/VHDL Scenario Generator mechanically. If the scenarios are meaningful ones reflecting the RPS trip logics, the experiment will go close to common agreement. We now try to reflect the fixed set-point trip logic to generate simulation scenarios as shown in Figs. 4 and 5. Other trip logics, such as manual reset and variable set-point, should be dealt with, and we are working on these.

Structural testing techniques [41], as well as functional testing, should be considered to improve the quality of simulation scenarios. We need to check the code coverage [20] of simulation scenarios generated against RTL programs to determine whether they meet the code coverage requirements for safety-critical I&C systems, which unfortunately are not yet defined. DO-254 [28] in avionics requires 100% statement and branch coverages for complex hardware systems that use FPGAs, complex programmable logic devices, and application-specific integrated circuits. It is also required that Verilog/VHDL Scenario Generator generates test scenarios that can meet specific predefined code coverages, and vice versa. We are now planning an approach to this problem.

5. Related work

There are several methods performing design verification in the FPGA software development process. Simulation-based verification is a widely used way to confirm the correctness of designs. The co-simulation [42] technique can also be used to execute two designs in parallel. Yang et al [43] proposed RTL scan flow that is similar to our framework. It uses simulation scenarios to perform co-simulation with an RTL-level design and a mapped design in an FPGA environment (iPOVE) [44]. It is more closely related to hardware/software co-simulation. Zheng et al [14] proposed an FPGA software verification method using test bench and assertion analysis. It simulates an RTL program and a gate-level design with generated scenarios, including assertions. The simulation with assertions checks specific design features and can be used to show the authenticity and efficiency of FPGA. Neither support automated scenario generation.

There are also several methods used to generate simulation scenarios from HDL designs [45–47]. Vemuri et al [45] presented a method for generation of input sequences from VHDL programs. It generates input sequences to execute desired control-flow paths through enumeration, constraint generation, and constraint-solving techniques. Gharebaghi et al [46] presented two generation algorithms that work on the VHDL process, representing combinational logic and sequential logic. The goal of both algorithms is to test all portions of the process body by traversing all the feasible paths in order to achieve high coverages. Hari et al [47] proposed a methodology for automatic extraction of word-level model constraints from Verilog. The scenarios are also expressed as constraints, which can be solved using an integer solver to arrive at the necessary functional test.

6. Conclusions and future work

This paper proposed an integrated software testing framework (IST-FPGA) for FPGA-based digital controllers in NPPs. It simulates all designs in each level of FPGA software development simultaneously and evaluates whether all designs work correctly against common oracle programs. It also generates a massive number of input scenarios with the guide of domain experts in order to produce meaningful scenarios reflecting reactor shutdown logics. It is also supported by assistant tools, such as Verilog/VHDL Scenario Generator and Co-simulator. We performed experiments with two FPGA software implementations of RPS and showed that the proposed framework can save time and costs associated with verifying FPGA software.

We are now trying to improve the quality of simulation scenarios, which are automatically generated by our tools, in two ways. From the aspect of functional testing, more delicate scenario generation reflecting all reactor shutdown logics are required. From the aspect of structural testing, code coverages for RTL programs should be measured and also be used to generate simulation scenarios, and vice versa. We are also planning to analyze the correspondence between the code coverages in RTL and Netlists at the gate-level in order to clarify the impact of high-quality FPGA software designs on FPGA hardware implementations.

Conflicts of interest

All authors have no conflicts of interest to declare.

Acknowledgments

This research was supported, in part, by a grant from the Korean Ministry of Science, ICT, and future planning under the I&C Safety Conformance Assessment Platform. It was also supported, in part, by a grant from the Korea Atomic Energy Research Institute under the development of the core software technologies of the integrated development environment for FPGA-based controllers.

REFERENCES

[1] J. Kim, E.S. Kim, J. Yoo, A translator verification technique for FPGA software development in nuclear power plants, Transactions of the Korean Nuclear Society Autumn Meeting, Pyeongchang, Korea, 2014, pp. 1986–1988.
[2] International Electrotechnical Commission (IEC), International standard for programmable controllers: Programming languages 61131-Part 3, IEC, 1993.
[3] 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.
[4] J.G. Choi, Survey of the CPLD/FPGA technology for application to NPP digital I&C system, Technical Report, Korea Atomic Energy Research Institute, 2009.
[5] J. Choi, D. Lee, Development of RPS trip logic based on PLD technology, Nucl. Eng. Technol. 44 (2012) 697–708.
[6] J. Ranta, The current state of FPGA technology in the nuclear domain, Technical Report, VTT Technical Research Centre of Finland, Espoo (Finland), 2012.
[7] NS-G-1.2, Safety assessment and verification for nuclear power plants: safety guide, IAEA, Vienna, 2004.
[8] International Electrotechnical Commission (IEC), Nuclear power plants–Instrumentation and control systems important to safety–Software aspects for computer-based systems performing category A functions, IEC 60880, 2006.
[9] International Electrotechnical Commission (IEC), Nuclear power plants–Instrumentation and control important to safety–Hardware design requirements for computer-based systems, IEC 61513, 2011.
[10] Xilinx [Internet]. San Jose, CA, USA, 2013 [cited 2016 Feb 17]. Xilinx ISE Design Suite. Available from: http://www.xilinx.com/products/design-tools/ise-design-suite.html.
[11] Altera [Internet]. San Jose, CA, USA, 2015 [cited 2016 Feb 17]. Quartus Prime. Available from: https://www.altera.com/products/design-software/overview.html.
[12] Microsemi [Internet]. Aliso Viejo, CA, USA, 2015 [cited 2016 Feb 17]. Libero SoC. Available from: http://www.microsemi.com/products/fpga-soc/design-resources/design-software/libero-soc.
[13] D. Kim, M. Ciesielski, S. Yang, A new distributed event-driven gate-level HDL simulation by accurate prediction, Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, Grenoble, France, 2011, pp. 1–4.
[14] D. Zheng, W. Yichen, Z. Xueyi, The methods of FPGA software verification, 2011 IEEE International Conference on Computer Science and Automation Engineering (CSAE), Shanghai, China, Volume 3, IEEE, 2011, pp. 86–89.
[15] R.E. Bryant, A methodology for hardware verification based on logic simulation, JACM 38 (1991) 299–328.
[16] International Electrotechnical Commission (IEC), Nuclear power plants–Instrumentation and control important to safety–Hardware design requirements for computer-based systems, IEC 60987, 2013.
[17] International Electrotechnical Commission (IEC), Nuclear power plants–Instrumentation and control important to safety–Development of HDL-programmed integrated circuits for systems performing category A functions, IEC 62566, 2012.
[18] M. Bobrek, D. Bouldin, D.E. Holcomb, S.M. Killough, S.F. Smith, C. Ward, R.T. Wood, Review guidelines for field-programmable gate arrays in nuclear power plant safety systems, NUREG/CR-7006, U.S. NRC, 2010.
[19] R. Lissel, J. Gerlach, Introducing new verification methods into a company's design flow: an industrial user's point of view, Proceedings of the Conference on Design, Automation,

and Testing in Europe, EDA Consortium, Nice, France, 2007, pp. 689–694.
[20] J.Y. Jou, C. Liu, Coverage analysis techniques for HDL design validation, Proceedings of the 6th Asia Pacific Chip Design Languages, Fukuoka, Japan, 1999, pp. 48–55.
[21] S. Sjoholm, L. Lindh, The need for co-simulation in ASIC-verification, EUROMICRO 97. New Frontiers of Information Technology, Proceedings of the 23rd EUROMICRO Conference, IEEE, Budapest, Hungary, 1997, pp. 331–335.
[22] MathWorks [Internet]. Natick, MA, USA, 2015 [cited 2016 Feb 17]. HDL Verifier. Available from: http://kr.mathworks.com/products/hdl-verifier/features.html.
[23] C. Valderrama, F. Naçabal, P. Paulin, A. Jerraya, Automatic VHDL-C interface generation for distributed co-simulation: application to large design examples, Des. Autom. Embed. Syst. 3 (1998) 199–217.
[24] B. Oraw, V. Choudhary, R. Ayyanar, A co-simulation approach to model-based design for complex power electronics and digital control systems, Proceedings of the 2007 Summer Computer Simulation Conference, Society for Computer Simulation International, San Diego, CA, USA, 2007, pp. 157–164.
[25] Mentor Graphics [Internet] Wilsonville, OR, USA, 2016 [cited 2016 Feb 17]. Questa ADMS. Available from: https://www.mentor.com/products/fv/advance_ms/.
[26] SCC [Internet]. 2015, [cited 2016 Feb 17]. Software Certification Consortium. Available from: http://cps-vo.org/group/scc.
[27] J.K. Lee, Y.M. Kim, Design and verification of FPGA-based applications in nuclear power plants, J. Energy Power Eng. 7 (2013) 537–544.
[28] Radio Technical Commission for Aeronautics (RTCA), Design assurance guidance for airborne electronic hardware, DO-254, 2000.
[29] Institute of Electrical and Electronics Engineers (IEEE), IEEE standard for software verification and validation, IEEE 1012, 2005.
[30] International Electrotechnical Commission (IEC), Functional safety of electrical, electronic and programmable electronic (E/E/PE) safety-related systems, IEC 61508, 2000.
[31] Institute of Electrical and Electronics Engineers (IEEE), IEEE standard criteria for digital computers in safety systems of nuclear power generating stations, IEEE 7-4.3.2, 2003.
[32] Institute of Electrical and Electronics Engineers (IEEE), IEEE standard criteria for safety systems of nuclear power generating stations, IEEE 603, 2003.
[33] TR-1019181, Guidelines on the use of field-programmable gate arrays in nuclear power plant I&C systems, Electric Power Research Institute, 2009.
[34] TR-109390, Design description of a prototype implementation of three reactor protection system channel using field-programmable gate arrays, Electric Power Research Institute, 1998.
[35] TR-1022983, Recommended approaches and design criteria for application of field-programmable gate arrays in nuclear power plant instrumentation and control systems, Electric Power Research Institute, 2009.
[36] Mentor Graphics [Internet]. Wilsonville, OR, USA, 2015 [cited 2016 Feb 17]. ModelSim. Available from: http://www.mentor.com/products/fpga/simulation/modelsim.
[37] E.S. Kim, D.A. Lee, J. Yoo, The scenario generator for verifying the correctness of FBDtoVerilog translator Volume 1, Korea Information Processing Society (KCC 2014), Busan, Korea, 2014, pp. 599–602 [in Korean].
[38] E.M. Clarke, O. Grumberg, D. Peled, Model checking, MIT press, Cambridge, 1999.
[39] J. Yoo, E.S. Kim, D.A. Lee, J.G. Choi, An integrated software development framework for PLC- & FPGA-based digital I&Cs, International Symposium on Future I&C for Nuclear Power Plants/International Symposium on Symbiotic Nuclear Power System (ISOFIC/ISSNP), Jeju, Korea, 2014.
[40] Synopsys [Internet]. Mountain View, CA, USA, 2015 [cited 2016 Feb 17]. Synplify Pro. Available from: http://www.synopsys.com/Tools/Implementation/FPGAImplementation/FPGASynthesis/Pages/SynplifyPro.aspx.
[41] M. Pezze, M. Young, Software testing and analysis: process, principles, and techniques, John Wiley & Sons, New York, 2008.
[42] S. Sicklinger, V. Belsky, B. Engelmann, H. Elmqvist, H. Olsson, R. Wiichner, K.U. Bletzinger, Interface Jacobian-based co-simulation, Int. J. Numer. Meth. Eng. 98 (2014) 418–444.
[43] S. Yang, H. Shim, W. Yang, C.M. Kyung, A new RTL debugging methodology in FPGA-based verification platform, Proceedings of 2004 IEEE Asia-Pacific Conference on Advanced System Integrated Circuits 2004, Fukuoka, Japan, IEEE, 2004, pp. 180–183.
[44] Dynalith Systems, iPROVE: a block design and Verification platform, White Paper, 2003.
[45] R. Vemuri, R. Kalyanaraman, Generation of design verification tests from behavioral VHDL programs using path enumeration and constraint programming, IEEE T. VLSI Syst. 3 (1995) 201–214.
[46] A.M. Gharebaghi, Z. Navabi, High-level test generation from VHDL behavioral descriptions, Proceedings of the VHDL International Users Forum Fall Workshop (VIUF'00), Orlando, FL, USA, 2000, pp. 123–126.
[47] S.K.S. Hari, V.V.R. Konda, V. Kamakoti, V.M. Vedula, K.S. Maneperambil, Automatic constraint-based test generation for behavioral HDL models, IEEE T. VLSI Syst. 16 (2008) 408–421.