2014 IEEE International Symposium on Software Reliability Engineering Workshops

A Technique for Demonstrating Safety and Correctness of Program Translators: Strategy and Case Study

Eui-Sub Kim , Junbeom Yoo
Computer Science and Enginering
Konkuk Univeristy
Republic of Korea
Email: {atang34, jbyoo}@konkuk.ac.kr

Jong-Gyun Choi , Young Jun Lee , Jang-Soo Lee
MMIS Lab.
Korea Atomic Energy Research Institute
Republic of Korea
Email: {choijg, yjlee426, jslee}@kaeri.re.kr

***Abstract*—The safety and correctness demonstration of program translators plays a critical role in software certification of digital I&C (Instrumentation & Control) systems in nuclear power plants. This paper proposes a strategy for the demonstration of the *FBDtoVerilog* translator, which translates FBD programs into Verilog programs to synthesize FPGAs. It uses safety case to explain the strategy precisely and also implemented several supporting tools to derive evidences efficiently. A case study of a Korean nuclear power plant found the efficiency of the proposed demonstration strategy and supporting tools.**

I. INTRODUCTION

Program translators are widely used to develop embedded software more efficiently and safely. When developing a new I&C system in Korean nuclear power plants with PLCs (Programmable Logic Controllers) [1], [2], the correctness and safety demonstration of the FBD-to-C [3] and C-to-PLC translators was strongly requested by regulation authorities for approving the operation of the I&C system [4]. We are now developing an I&C system [5] with FPGAs (Field Programmable Logic Controller) [6] in place of PLCs, due to the increasing maintenance cost of PLCs and the higher performance of FPGAs.

The platform change of I&C from PLC to FPGA, however, involves an element of risk. The software development of PLC should shift to the hardware development of FPGA. The new development paradigm requires not only a whole new hardware developing process, but also abandon the accumulated experience and knowledge to develop the PLC software. In order to reduce the risk and preserve the experience and knowledge, we did propose a hybrid development process [7] which uses the '*FBDtoVerilog*' translator. The '*FBDtoVerilog*' [8] mechanically translates FBD(Function Block Diagram) programs of the PLC-based development into Verilog programs of the FPGA-based development, while preserving behavioral equivalence. It is also required to demonstrate its safety and correctness sufficiently before being used in earnest.

This paper proposes a strategy for demonstrating the safety and correctness of the program translator '*FBDtoVerilog*.' The conventional verification techniques [9], [10] for translators and compilers are hard to apply because of the outrageous cost and performance time. Instead, this paper tries to use an indirect approach; we do not prove the translators against all possible input cases, but only against specific inputs which are under development for a target I&C system. The indirect demonstration can assure that the translator works safely and correctly at least for the FBD programs under development.

We use the safety case technique [11] to explain the proposed strategy more precisely and systematically. We first set the top goal and find necessary evidences and then connect the evidences to the goal with logical arguments. It is important to connect between evidences and goal with the technique since a proof how the evidences contribute to the goal is more important than how many the evidence we are founded. It sets goals first and provides evidences and logical arguments which connect the evidences to the goals. This paper also developed several CASE tools to support the derivation of evidences, *e.g.,* '*FBDtoCadenceSMV*,' '*Scenario Generator*,' '*FBD Simulator*' and '*FBD-Verilog Comparator*.' A case study with an FBD program of a Korean nuclear power plant could also find the efficiency of the proposed indirect demonstration strategy and supporting tools.

The paper is organized as follows: Section 2 proposes the strategy for demonstrating the safety and correctness of the '*FBDtoVerilog*' translator. Section 3 explains 4 supporting tools which we developed to derive evidences of arguments efficiently. In Section 4, we performed a case study with an FBD program of an I&C system in a Korean nuclear power plant. Section 5 concludes the paper and provides remarks on future research extension.

II. THE DEMONSTRATION STRATEGY

This section explains our approach to demonstrate the safety and correctness of the '*FBDtoVerilog*' translator, indirectly. The specific input is not means that an inputted program must have restrictions in regard of language, but means that we do not prove the translators against all possible input cases, but only against specific inputs which are under development for a target I&C system. We intend it for a specific input, *i.e.*

978-1-4799-7377-4/14 $31.00 © 2014 IEEE
DOI 10.1109/ISSREW.2014.71

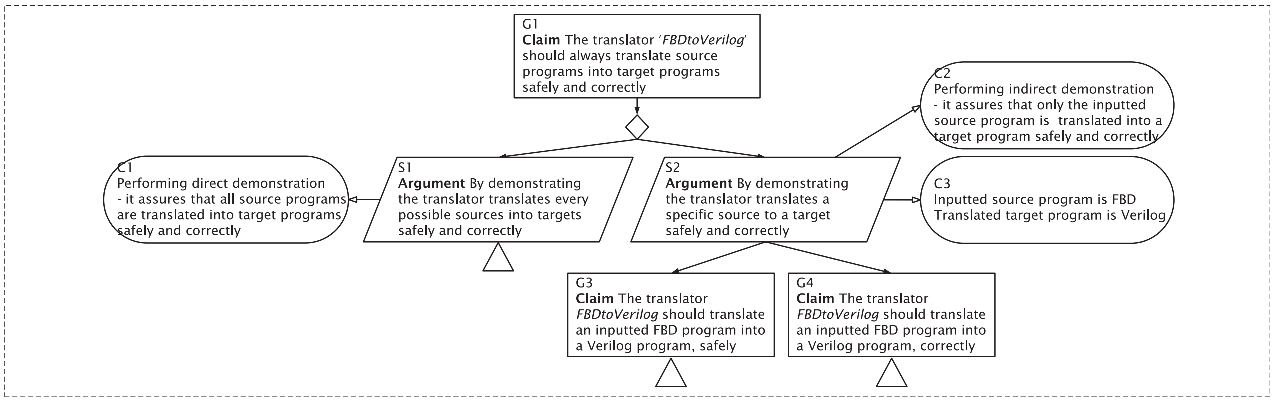
G1
Claim The translator 'FBDtoVerilog' should always translate source programs into target programs safely and correctly
C2
Performing indirect demonstration - it assures that only the inputted source program is translated into a target program safely and correctly
C1
Performing direct demonstration - it assures that all source programs are translated into target programs safely and correctly
S1
Argument By demonstrating the translator translates every possible sources into targets safely and correctly
S2
Argument By demonstrating the translator translates a specific source to a target safely and correctly
C3
Inputted source program is FBD Translated target program is Verilog
G3
Claim The translator FBDtoVerilog should translate an inputted FBD program into a Verilog program, safely
G4
Claim The translator FBDtoVerilog should translate an inputted FBD program into a Verilog program, correctly

Fig. 1. The top-level GSN for the safety and correctness demonstration of translators

an FBD program, not for all possible inputs. The GSN(Goal Structure Notation) in ⟨Fig.1⟩ shows our goals, arguments and required evidences clearly. We first set up the top-level goal (G1). There are two strategies such as (S1) and (S2) to accomplish the goal (G1). The (S1) is a direct demonstration, which uses compiler verification techniques [9], [12], [13], [14], and its outrageous cost made us hard to use it. The (S2), on the other hand, is an indirect approach as explained in (C2). It needs a specific input source program (*i.e.,* FBD) and a specific output target program (*i.e.,* Verilog) as assumed by (C3). This paper proposes to use the indirect strategy (S2). (G3) and (G4) are the sub-goals for the (S2), which are responsible for the safety and the correctness, respectively. The following subsections expand the sub-goals (G3) and (G4) in details.

A. The Safety Demonstration Strategy

The GSN in ⟨Fig.2⟩ illustrates how we can demonstrate the safety of the translator '*FBDtoVerilog*.' When demonstrating the safety goal (G3), we first need to define the safety of the translator '*FBDtoVerilog'* clearly as suggested by (C4). (Definition 1) below defines the safety. According to the definition, (S3) suggests that a translated program is safe for an input program if the translated program satisfies the safety properties, which were already satisfied with the source program. (A1) assumes that we use the model checking technique [15] to support the (S3). Model checker can check whether a program reaches a state violating safety properties through searching all possible input sets and states exhaustively.

There are ambiguous points in practice using the model checking with same safety property such as potential hazards in source, new introduced failures in target one and so on. However, it is no concerns of our scope how deal with potential hazards in source, how the potential hazards are translated to target, how failures in target is generated from no failure source and so on. The purpose of the model checking technique proves whether both programs satisfy the same safety property or not. The potential hazards should be handled by safety analysis in early phase and the new introduced failures are a translator error, which is the point we must find with model checking technique.

Definition 1. *A translator is safe, if safety properties are satisfied with the input and output programs simultaneously.*

The argument (S3) can be demonstrated by three sub-goals (G5), (G6) and (G7). Before performing model checking upon the input and output programs against safety properties, we have to be sure that the safety properties are reflecting important safety features of the target input/output programs (S4) and well formed with the formalism which the model checking technique requires (S5). These are all about the appropriateness of the safety properties (G5). Reviews by domain experts and formalism experts (A2, Sn1, A3) are required to support the (G5).

After assuring the appropriateness of the safety properties (G5), we can proceed to the claims (G6) and (G7), which check the safety properties against two programs - FBD programs (G10) and Verilog programs (G11). The model checking of Verilog programs (S7) can be conducted by using the SMV model checker [16]. It reads Verilog program as well as the SMV input program without modification and performs the CTL model checking [17]. On the other hand, model checking of FBD programs needs other techniques such as translating FBDs into equivalent SMV input programs (S6), since we cannot use our translator '*FBDtoVerilog*' itself recursively. We, therefore, tried to use other's translation technique [18] from FBDs into SMV input programs, and implemented a mechanical translator '*FBDtoCadenceSMV*' (Sn2). The details of tool will be introduced in Section 3.A. The comparison of model checking results of FBD and Verilog programs are defined in (G10) and (G11), and it can be performed by naive comparison as (Sn3) and (Sn4).

B. The Correctness Demonstration Strategy

The GSN in ⟨Fig.3⟩ explains how we can demonstrate the correctness of the translator. When demonstrating the correctness goal (G4), we also need to define the correctness

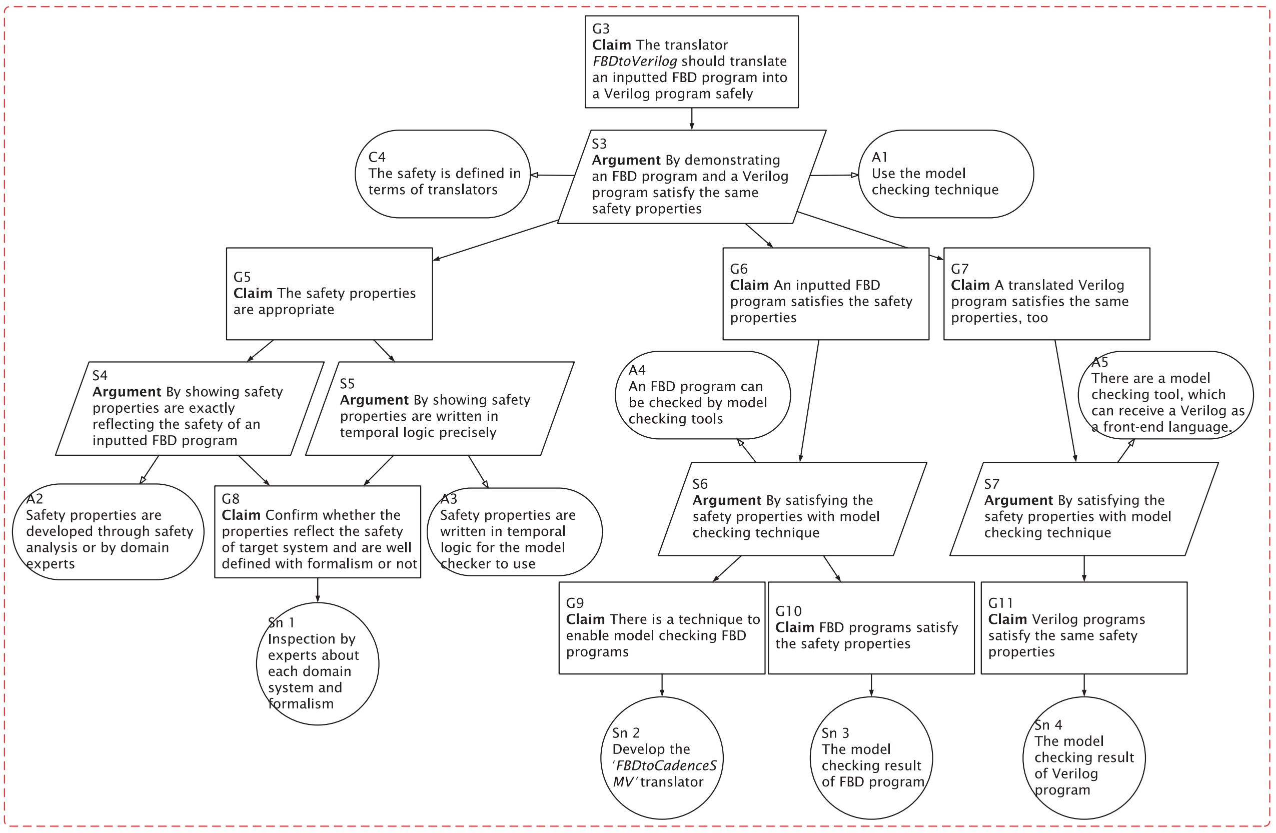
G3 Claim The translator FBDtoVerilog should translate an inputted FBD program into a Verilog program safely
C4 The safety is defined in terms of translators
S3 Argument By demonstrating an FBD program and a Verilog program satisfy the same safety properties
A1 Use the model checking technique
G5 Claim The safety properties are appropriate
G6 Claim An inputted FBD program satisfies the safety properties
G7 Claim A translated Verilog program satisfies the same properties, too
S4 Argument By showing safety properties are exactly reflecting the safety of an inputted FBD program
S5 Argument By showing safety properties are written in temporal logic precisely
A4 An FBD program can be checked by model checking tools
A5 There are a model checking tool, which can receive a Verilog as a front-end language.
A2 Safety properties are developed through safety analysis or by domain experts
G8 Claim Confirm whether the properties reflect the safety of target system and are well defined with formalism or not
A3 Safety properties are written in temporal logic for the model checker to use
S6 Argument By satisfying the safety properties with model checking technique
S7 Argument By satisfying the safety properties with model checking technique
Sn 1 Inspection by experts about each domain system and formalism
G9 Claim There is a technique to enable model checking FBD programs
G10 Claim FBD programs satisfy the safety properties
G11 Claim Verilog programs satisfy the same safety properties
Sn 2 Develop the 'FBDtoCadenceSMV' translator
Sn 3 The model checking result of FBD program
Sn 4 The model checking result of Verilog program

Fig. 2. The GSN for the safety demonstration (G3)

of the translator *'FBDtoVerilog'* clearly as suggested by (C5) as the (Definition 2) below. If a source program (*i.e.,* FBD program) and its translated program (*i.e.,* Verilog program) show the same outputs for all possible scenarios of inputs (S7), the (G4) claims that the translator is correct. The (A6) suggests using the co-simulation techniques [19] to check their behavioral equivalence.

Definition 2. *A translator is correct, if the behavior of a translated program is the same with its source program for all possible input scenarios.*

The argument (S8) can be demonstrated by two sub-goals (G12) and (G13). Before performing the co-simulation as (G13), we need to be sure that the scenarios we use are sufficient for demonstrating the behavioral equivalence as (G12). (S9) suggests that we need to develop various scenarios as many as possible, while (S10) focuses on the appropriateness of the developed scenarios, *i.e,* whether they reflect domain-specific features. To support these arguments, we developed a mechanical '*Scenario Generator*' for FBD programs. It can generate random scenarios as well as user-defined scenarios, while preserving predefined constraints such as rate-of-changes of continuous input variables (A9) and the range and scale of variables. The tool '*Scenario Generator*' can generate various scenarios while preserving domain-specific constraints. The details of tool will be introduced in Section 3.B.

The sub-goal (G13) claims that the two programs (FBD and Verilog) should show the same behavior for all developed scenarios by the (G12). It can be satisfied by two arguments (S10) and (S11). The (S11) suggests using existing simulation tools for FBD and Verilog programs. While a commercial tool *ModelSim* [20] can simulate the input scenarios efficiently as (Sn9), it was hard to find an appropriate simulator for FBDs, which can read the developed scenarios and generate outputs efficiently. It is due to the custom that PLC vendors (*e.g.,* [21] and [22]) provide vendor-specific tools for FBD programming and simulation. We, therefore, developed an '*FBD Simulator*,' which can read FBD programs in XML format of the PLCopen TC6 standard [23] and simulate them with input scenarios of '*Scenario Generator*' as (Sn7). The details of the tool will be explained in Section 3.C.

The (S12) requires comparing the simulation results of the two input programs, FBD and Verilog, simultaneously for all scenarios. The input scenarios are developed by the tool '*Scenario Generator*,' while the execution of two input programs are simulated by the tools, '*FBD Simulator*' and

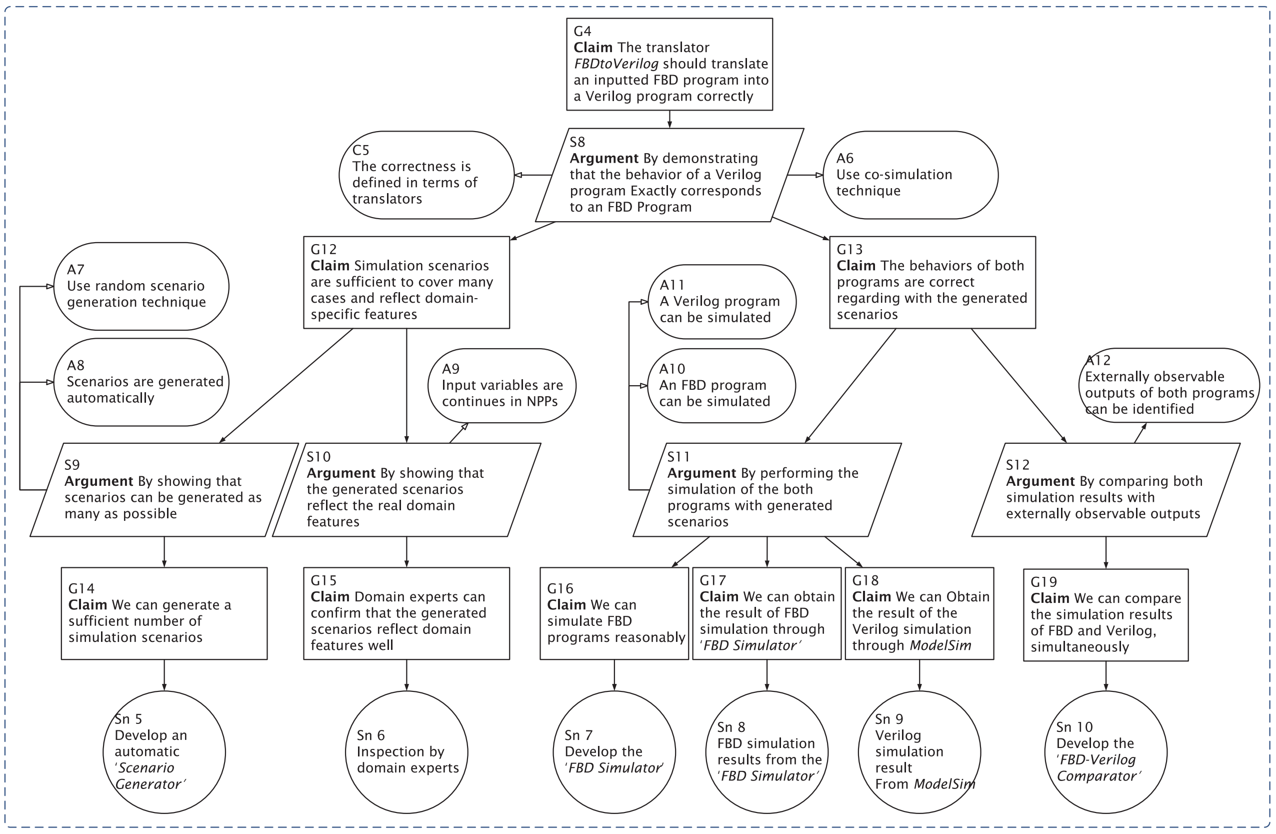
G4
Claim The translator FBDtoVerilog should translate an inputted FBD program into a Verilog program correctly
S8
Argument By demonstrating that the behavior of a Verilog program Exactly corresponds to an FBD Program
C5
The correctness is defined in terms of translators
A6
Use co-simulation technique
G12
Claim Simulation scenarios are sufficient to cover many cases and reflect domain-specific features
G13
Claim The behaviors of both programs are correct regarding with the generated scenarios
A7
Use random scenario generation technique
A8
Scenarios are generated automatically
A9
Input variables are continues in NPPs
A11
A Verilog program can be simulated
A10
An FBD program can be simulated
A12
Externally observable outputs of both programs can be identified
S9
Argument By showing that scenarios can be generated as many as possible
S10
Argument By showing that the generated scenarios reflect the real domain features
S11
Argument By performing the simulation of the both programs with generated scenarios
S12
Argument By comparing both simulation results with externally observable outputs
G14
Claim We can generate a sufficient number of simulation scenarios
G15
Claim Domain experts can confirm that the generated scenarios reflect domain features well
G16
Claim We can simulate FBD programs reasonably
G17
Claim We can obtain the result of FBD simulation through 'FBD Simulator'
G18
Claim We can Obtain the result of the Verilog simulation through ModelSim
G19
Claim We can compare the simulation results of FBD and Verilog, simultaneously
Sn 5
Develop an automatic 'Scenario Generator'
Sn 6
Inspection by domain experts
Sn 7
Develop the 'FBD Simulator'
Sn 8
FBD simulation results from the 'FBD Simulator'
Sn 9
Verilog simulation result From ModelSim
Sn 10
Develop the 'FBD-Verilog Comparator'

Fig. 3. The GSN for the correctness demonstration (G4)

ModelSim. We developed a mechanical comparator '*FBD-Verilog Comparator*' which compares the simulation outputs from the two tools, as explained in (Sn10). The Comparator is described in Section 3.D.

III. The Development of Supporting Tools

A. FBDtoCadenceSMV

The '*FBDtoCadenceSMV*' translator translates an FBD program of the format of the PLCopen TC6 [23] into a behaviorally-equivalent SMV input program, which is an input front-end of the Cadence SMV model checker [16]. It is intended for the goal (G9) and extended the translation algorithm proposed by [18] in order to handle SEL and TOF function blocks, which were not handled by the original research. In accordance with the translation algorithm, the tool needs to set the maximum execution cycle to avoid the state explosion problem [17].

We are now planning to define the translation rules more formally and generally on the basis of [18] in order to deal with all categories of function blocks, which the IEC 61131-3 defined. As an alternative, we are also planning to develop our own translation rules and translator like our research about '*FBDtoVerilog*' [24], [8].

B. Scenario Generator

The '*Scenario Generator*' depicted in ⟨Fig.4. (a)⟩ generates various scenarios for executing FBD and Verilog programs simultaneously. It is intended for the goal (G14). It first takes several constraints on input values, *e.g.,* initial values, rate of change and maximum/minimum values. The tool then randomly generates a number of scenarios within predefined constraints on input values. The generated scenarios consist of two sets for FBD and Verilog programs respectively. The '*FBD Simulator*' in Section 3.C executes an FBD according to the scenarios, while the *ModelSim* simulator [20] reads and executes the scenarios for Verilog. We are now planning to extend it with more elaborate and systematic generation strategy, based on theories such as structural coverage criteria [25], [26] for co-simulation.

C. FBD Simulator

⟨Fig.4. (b)⟩ show '*FBD Simulator*' which we developed for the goals (G13) and (G16). While conventional/commercial tools such as the *ModelSim* can simulate Verilog programs efficiently, FBD programs are not easy to execute or simulate with scenarios of others. It is due to the fact that PLC software engineering tools provided by PLC vendors support for the FBD programming and simulation, working on their own tools

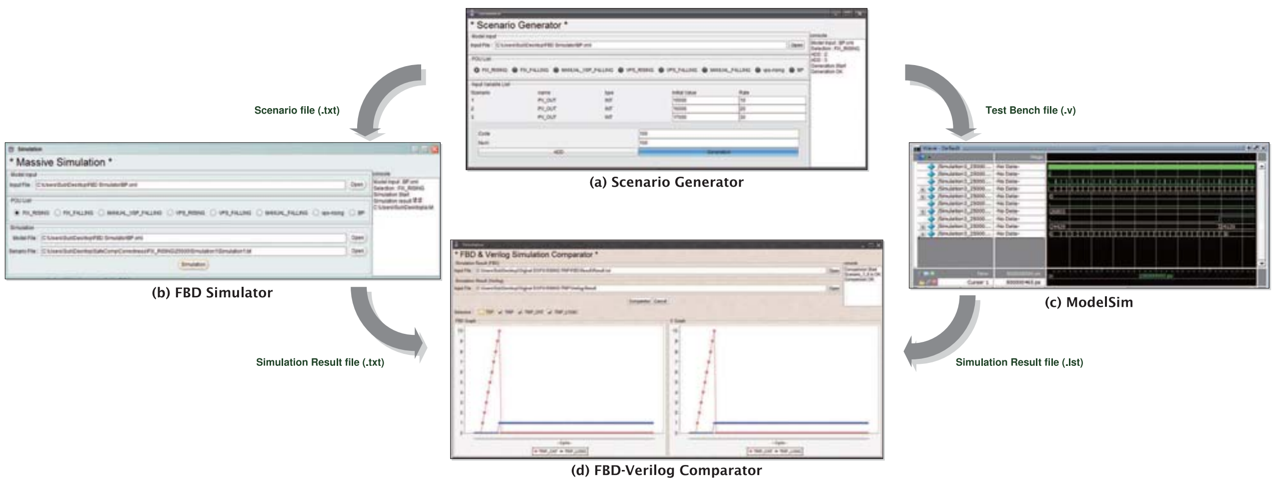

Fig. 4. Co-Simulation Framework

and PLC hardware. It first reads a set of FBD scenarios, which are generated from '*Scenario Generator*.' After simulation is done, it writes the simulation results into a text file. The '*FBD-Verilog Comparator*' introduced at the next subsection reads the text file and judges their equivalence.

D. *FBD-Verilog Comparator*

⟨Fig.4. (d)⟩ show '*FBD-Verilog Comparator*' which we developed for the goal (G19). It compares the execution results of two programs (*i.e.,* FBD and Verilog) with the same input scenarios. If all simulation results are equivalent, it produces 'True,' otherwise it produces counter example with graphical chart. The graphical chart make user to know how two programs reach to the different state with tracing the flow of variables.

IV. Case Study

We performed a case study with an FBD program of the KNICS APR-1400 RPS BP system [27] in order to demonstrate the correctness and safety of the '*FBDtoVerilog*' translator, indirectly. The BP reads 18 sensor values from a nuclear reactor and decides to generate trip/pre-trip signals out to shutdown the reactor immediately, if any value is out of safe range. This case study used two examples of the 18 logics, such as FIX-RISING and FIX-FALLING. The case study is aiming for demonstrating the safety and correctness of the '*FBDtoVerilog*' translator at least for the two logics. It performed the safety and correctness demonstration in accordance with the strategy in Section 2.

A. *The Safety Demonstration (G3)*

We performed the Cadence SMV model checking upon the two input/output of the '*FBDtoVerilog*' translator - the FBD and Verilog programs, to prove whether they all satisfy the same safety properties. We developed 28 safety properties with assistant from domain exports and referable papers [27], [28], [29] (Sn1). They are all concerning about only the safety of the target system. The below are the example of the safety properties and corresponding CTL formulae to be checked with the SMV model checker.

"*If PV_OUT (An input sensor value) is more than the TSP (Trip Set-Point) for a predefined time, then the trip signal should be fired (TRIP_LOGIC = 1) immediately.*"
: AG((PV_OUT > TSP) & (TRIP_CNT >= (MAXCNT - 1)) → AX(TRIP_LOGIC = 1))

The '*FBDtoCadenceSMV*' translated the FBD program into a behaviorally-equivalent SMV program(Sn2). We performed the SMV model checking upon the SMV programs against the 14 safety properties(Sn3). We could check that all 14 safety properties are satisfied. On the other hand, the model checking of Verilog programs is straightforward, since the model checker can read Verilog programs without modification (Sn4). The FBDs of two logics (*i.e.,* FIXED-RISING and FIXED-FALLING) and the Verilog programs translated by the '*FBDtoVerilog*' translator were satisfied with the whole safety properties. Therefore, we can assure that the '*FBDtoVerilog*' translator translated the FBD programs into the Verilog programs, safely (G3).

B. *The Correctness Demonstration (G4)*

We also performed the co-simulation of the FBD and Verilog programs to demonstrate the correctness of the '*FBDtoVerilog*' translator (G4). We used the CASE tools which we developed to support the co-simulation, as introduced in the Section 3. The '*Scenario Generator*' first generated 2,000 random scenarios mechanically (Sn5), complying with the predefined constraints (Sn6), as shown in ⟨Table I⟩.

The '*FBD-Verilog Comparator*' then executed the '*FBD Simulator*' (Sn7, Sn8) and the *ModelSim* (Sn9) simultaneously with the set of scenarios, and judged that they all showed the

TABLE I
THE CO-SIMULATION RESULTS

Name of Logic	Scenarios	Initial Values	Rate of Change	Cycles
FIX-RISING	10,000	27,000 - 28,000 (Stepwise: 100)	10 - 100 (Stepwise: 10)	100
FIX-FALLING	10,000	12,000 - 13,000 (Stepwise: 100)	10 - 100 (Stepwise: 10)	100

same outputs for all sequences of inputs (Sn10). Therefore, we can assure that the '*FBDtoVerilog*' translator translated the FBDs into the Verilog programs, correctly (G4).

It is worth to note that the correctness demonstration is valid under the 2,000 random scenarios. We are now trying to increase the confidence and thoroughness of the co-simulation scenarios thorough appropriate structural coverage criteria [25], [26] for co-simulation.

V. CONCLUSION AND FUTURE WORK

This paper proposed an indirect strategy for demonstrating the safety and correctness of the '*FBDtoVerilog*' translator. We used the safety case technique and GSN to explain the proposed strategy more precisely and systematically. We also developed several CASE tools to support for deriving evidences, such as '*FBDtoCadenceSMV*,' '*Scenario Generator*,' '*FBD Simulator*' and '*FBD-Verilog Comparator*.' We then performed a case study with an FBD program of the KNICS APR-1400 RPS BP in order to demonstrate the safety and correctness of the '*FBDtoVerilog*,' indirectly, according to the demonstration strategy proposed. We expect that the result is reasonable and sufficient to demonstrate the safety and correctness of the translator.

We are now trying to increase the confidence and thoroughness of the '*Scenario Generator*' on the basis of structural coverage criteria. We are also planning to improve the strategy through applying it to other translators which we developed, such as '*FBDtoC*' and '*NuSCRtoFBD*.' We expect to extend the proposed techniques into a safety and correctness demonstration framework for general translators and compilers.

ACKNOWLEDGMENT

This research was supported, in part, by a grant from the Korea Ministry of Science, ICT and Future Planning, under the development of the integrated framework of I&C dependability assessment, monitoring, and response for nuclear facilities. 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. Yoo, E. Jee, and S. S. Cha, "Formal Modeling and Verification of Safety-Critical Software," *IEEE Software*, vol. 26, no. 3, pp. 42–49, May/June 2009.

[2] J. Yoo, S. Cha, H. S. Son, C. H. Kim, and J.-S. Lee, "PLC-Based safety critical software development for nuclear power plants," in *23th International Confierence onf Computer Safety, Reliability, and Security (SAFECOMP 20014)*, 2004, pp. 155–165.

[3] J. Yoo, E.-S. Kim, and J.-S. Lee, "A Behavior-Preserving Translation from FBD Design to C Implementation for Reactor Protection System Software," *Nuclear Engineering and Technology*, vol. 45, no. 4, pp. 489–504, 2013.

[4] International Electrotechnical Commission, "Nuclear power plants - instrumentation and control systems important to safety - Software aspects for computer-based systems performing category a function," 2006, iEC 60880.

[5] J. G. Choi and D. Y. Lee, "DEVELOPMENT OF RPS TRIP LOGIC BASED ON PLD TECHNOLOGY," *Nuclear Engineering and Technology*, vol. 44, no. 6, pp. 697–708, 2012.

[6] Wikipedia, "FPGA: Field-programmable gate array," *http://en.wikipedia.org/wiki/ Field-programmable_gate_array*.

[7] J. Yoo, E.-S. Kim, and J.-S. Lee, "A Behavior-Preserving Translation from FBD Design to C Implementation for Reactor Protection System Software," *Nuclear Engineering and Technology*, vol. 45, no. 4, pp. 489–504, 2013.

[8] D.-A. Lee, E.-S. Kim, and J. Yoo, "FBDtoVerilog 2.0: An automatic translation of FBD into Verilog to develop FPGA," in *International Conference on Information Science and Application (ICISA 2014)*, 2014, accepted.

[9] E.-S. Kim, D.-A. Lee, and J. Yoo, "A Survey of Verification Methods for Translator, Code Generator and Compiler," in *Korea Computer Software Engineering 2013 (KCSE 2013)*, 2013, pp. 43–51.

[10] M. A. Dave, "Compiler verification: a bibliography," *ACM SIGSOFT Software Engineering Notes*, vol. 28, no. 6, pp. 2–2, 2003.

[11] D. Jackson, M. Thomas, L. I. Millett *et al.*, *Software for Dependable Systems: Sufficient Evidence?* National Academies Press, 2007.

[12] T. Hoare, "The verifying compiler: A grand challenge for computing research," *Journal of the ACM*, vol. 50, no. 1, pp. 63–69, 2003.

[13] X. Leroy, "Formal Verification of a Realistic Compiler," *Communication of the ACM*, vol. 52, no. 7, pp. 107–115, 2000.

[14] J. McCarthy and J. Painter, "Correctness of a Compiler for Algorithmic Expressions," in *Symposium in Applied Mathematics 19, Mathematical Aspects of Computer Science*, 1967, pp. 33–41.

[15] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT press, 1999.

[16] K. McMillan, "Cadence SMV," *http://www.kenmcmil.com*.

[17] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen, *Systems and software verification: model-checking techniques and tools*. Springer Publishing Company, Incorporated, 2010.

[18] O. Pavlovic and H.-D. Ehrich, "Model checking PLC software written in function block diagram," in *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*. IEEE, 2010, pp. 439–448.

[19] S. Sicklinger, V. Belsky, B. Engelmann, H. Elmqvist, H. Olsson, R. Wuchner, and K.-U. Bletzinger, "Interface jacobian-based co-simulation," *International Journal for Numerical Methods in Engineering*, vol. 98, no. 6, pp. 414–444, 2014.

[20] Mentor Graphics, "Modelsim," http://www.mentor.com/products/fpga/model.

[21] Simens, http://www.siemens.com.

[22] Ponu-Tech, http://www.ponu-tech.co.kr.

[23] PLCopen, "PLCopen for efficiency in automation," *http://www.plcopen.org*.

[24] J. Yoo, S. Cha, and E. Jee, "Verification of PLC Programs Written in FBD with VIS," *Nuclear Engineering and Technology*, vol. 41, no. 1, pp. 79–90, 2009.

[25] S. Tasiran and K. Keutzer, "Coverage metrics for function validation of hardware designs," *IEEE Design and Test of Computers*, vol. 18, no. 4, pp. 36–45, 2001.

[26] M. Graphics, www.mentor.com/fv.

[27] KAERI, "KNICS-RPS-SRS101 Rev.00," 2003.

[28] J. Yoo, T. Kim, S. Cha, J.-S. Lee, and H. Seong Son, "A formal software requirements specification method for digital nuclear plant protection systems," *Journal of Systems and Software*, vol. 74, no. 1, pp. 73–83, 2005.

[29] G.-Y. Park, K. Y. Koh, E. Jee, and P. H. Seong, "Fault Tree Analysis of KNICS RPS Software," *Nuclear Engineering and Technology*, vol. 40, no. 5, pp. 397–408, 2008.