An Integrated Software Development Framework for PLC & FPGA based Digital I&Cs

Junbeom Yoo[1], Eui-Sub Kim[2], Dong Ah Lee[3], and Jong-Gyun Choi[4]

1. Computer Science and Enginering Konkuk Univeristy Republic of Korea (jbyoo@konkuk.ac.kr)
2. Computer Science and Enginering Konkuk Univeristy Republic of Korea (atang34@konkuk.ac.kr)
3. Computer Science and Enginering Konkuk Univeristy Republic of Korea (ldalove@konkuk.ac.kr)
4. MMIS Lab. Korea Atomic Energy Research Institute Republic of Korea (choijg@kaeri.re.kr)

Abstract: *NuDE 2.0* (Nuclear Development Environment) is a model-based software development environment for safety- critical digital systems in nuclear power plants. It makes possible to develop PLC-based systems as well as FPGA-based systems simultaneously from the same requirement or design specifications. The case study showed that the *NuDE 2.0* can be adopted as an effective method of bridging the gap between the existing PLC and upcoming FPGA-based developments as well as a means of gaining diversity.
Keyword: software development, PLC, FPGA, nuclear power plants

1 Introduction

A safety-grade PLC (Programmable Logic Controller) has been used as an implementation platform of safety-critical digital systems in nuclear power plants, such as RPS (Reactor Protection System) and ESF-CCS (Engineered Safety Features-Components Control System). While complexity of newly developed systems and maintenance cost of the old ones have increased rapidly, alternative platforms for the PLC are widely being researched. The solution of [1,2,3] proposes to use FPGA (Field-Programmable Gate Array), which can provide powerful computation with lower hardware cost.

The platform change from PLC to FPGA, however, is not so straightforward. It gives rise to a paradigm shift from the CPU-based software development to FPGA-based hardware development. All PLC software engineers in nuclear domain should give up all experience, knowledge and practices accumulated over decades, and start a new FPGA-based hardware development from the scratch. The platform change may result in potential causes leading to safety-related problems. It is now strongly required to transit to the new development approach safely and seamlessly.

The loss and potential risk can be reduced if we can use the requirements and design specifications of the PLC-based systems as those of the FPGA-based systems, since the specifications are the fruit of the state-of-the-art PLC-based systems. The *NuDE 2.0* (Nuclear Development Environment) [4,5,6] makes us possible to develop the software systems of the PLC and FPGA platforms simultaneously from the same requirements or design specifications. The '*FBDtoVerilog 2.0/2.1*' translator [7], in particular, can translate an FBD program of a PLC-based RPS into a behaviorally equivalent Verilog program of FPGA platform which is the starting point of the mechanical FPGA synthesis process. We expect that the NuDE 2.0 can reduce the semantic gap between the PLC-based and FPGA-based developments (*i.e.,* software vs. hardware) and also be used as a means of gaining diversity of software design and implementation.

In order to demonstrate the possibility and effectiveness of the *NuDE 2.0*, we performed a case study with a preliminary FBD program of the KNICS APR-1400 RPS BP [8]. From the FBD program, C programs for PLC and Verilog/EDIF programs for FPGA were synthesized mechanically, and an exhaustive simulation tried to validate their behavioral equivalence. The organization of the paper is as

follows: Section 3 introduces the *NuDE 2.0* and various supporting tools, and Section 4 explains the case study in details. Section 5 concludes the paper and provides remarks on future research extension.

2 *NuDE 2.0*

The *NuDE 2.0* (Nuclear Development Environment) is a formal method- based software development environment, specialized for safety-critical digital systems in nuclear power plants. It starts from a formal requirements specification and transforms/synthesizes more concrete models subsequently across the whole SDLC (Software Development Life- Cycle). It now supports for PLC and FPGA platforms, simultaneously and seamlessly. It also encompasses various formal verification and safety analysis as well as the MBD (Model Based Development)-based code generation. (Fig.1) depicts the whole process in details, and the following subsections briefly explain each phase around supporting tools.

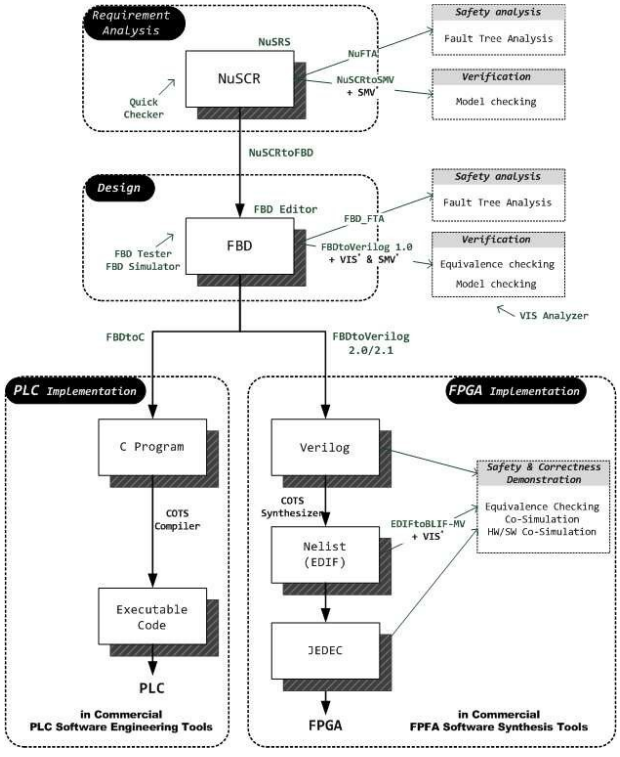


Figure 1 An overview of the *NuDE 2.0* framework

2.1 The Requirements Analysis Phase

(Fig.2) is an example of the *NuSCR* specification modeled in '*NuSRS 2.0*.' *NuSCR* [9] is a data-flow based requirements specification language, specialized for the safety-critical systems in the nuclear domain. The NuSCR modeling environment, NuSRS 2.0, includes static grammar checker '*Quick Checker*' and the '*NuSCRtoSMV*' [10] translator to generate the SMV input program and execute the Cadence SMV model checker [11], seamlessly. '*NuFTA*' [12] also generates software fault trees for the NuSCR specification mechanically. The NuSCR formal requirements specification is then translated into a behaviorally-equivalent FBD program by '*NuSCRtoFBD*' [13].

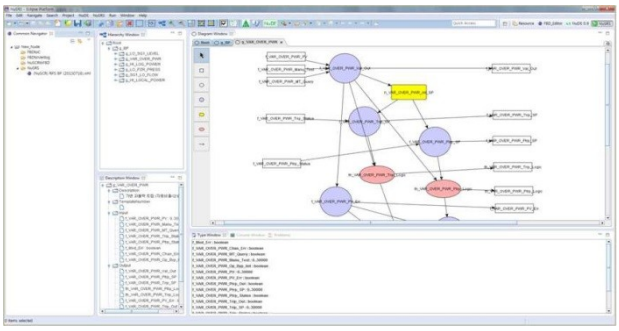


Figure 2 An *NuSCR* formal specification modeled in NuSRS 2.0

2.2 The Design Phase

'*FBD Editor*' in (Fig.3) shows the FBD program, which is mechanically translated from an NuSCR specification. We can also model it directly on the tool [14]. 'FBD Simulator' executes an FBD program with predefined inputs or randomly, while '*FBD Tester*' [15] enables us to do test the FBD programs directly with data-flow based coverage criteria for FBDs. Formal verification with the VIS verification system [16] and the SMV model checker is also possible through the '*FBDtoVerilog 1.0*' translator [17]. The FBD design phase often include hardware-dependent modifications on the FBDs, the formal verification are required additionally. The NuDE also provides '*VIS Analyzer*' [18] to assist the VIS verification graphically and seamlessly. '*FBD FTA*' [19] is a fault tree generation and analysis tool for FBD programs.

The FBD program modeled in the '*FBD Editor*' can be transformed into different implementation codes for PLC and FPGA. '*FBDtoC*' [20] translates FBDs into behaviorally- equivalent C programs for PLC, while '*FBDtoVerilog 2.0/2.1*' [7], [21] transforms FBDs into Verilog programs for FPGA. We are working on the transformation from FBDs into VHDL programs.

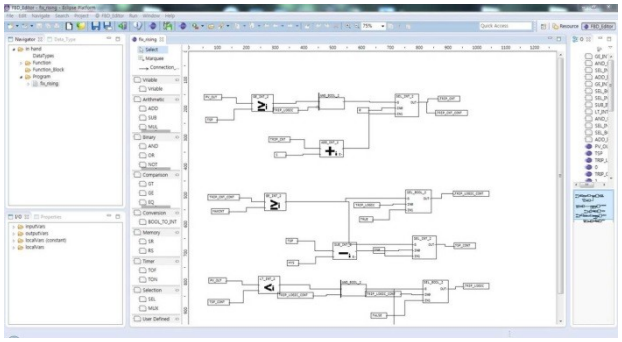


Figure 3 An example FBD program in '*FBD Editor*'

2.3 The PLC Implementation Phase

The C programs transformed by the 'FBDtoC' can be compiled into executable codes for a specific target PLC. Most commercial software engineering tools, however, translates FBDs into equivalent C and executable codes subsequently, and also downloads them into specific target PLCs. Most PLC vendors typically use COTS (Commercial Off-the-Shelf) software such as 'TMS320C55x' of Texas Instruments for the C compilers. The COTS compilers were well verified and certified enough to be used without additional verification effort. However, the vendor-provided automatic translators from FBD to C should demonstrate its functional safety and correctness rigorously, as we proposed in [22].

2.4 The FPGA Implementation Phase

The Verilog program translated by 'FBDtoVerilog 2.0/2.1' is the starting point of the fully-automated FPGA synthesis procedure provided by commercial tools. On the other hand, nuclear regulation authorities require more considerate demonstration of the correctness and even safety of the mechanical synthesis processes of FPGA synthesis tools, even if the FPGA industry have acknowledged them empirically as correct and safe processes and tools. While the synthesis process can be formally verified with the compiler verification techniques [23], [24], it is hard to apply them to the works of 3rd-party developers. It must be the most important obstacle for FPGAs to be used as a new platform of nuclear I&C systems. We are trying to overcome the obstacle through the safety and correctness demonstration technique proposed in [21].

2.5 Auxiliary Support for the Compiler Verification

The formal verification of compiler, translator and synthesizer is an important issue, and should be fully demonstrated whenever new PLC compilers or FPGA synthesis tools are proposed to use to develop new safety-critical digital systems in nuclear power plants. These are typically developed by 3rd-parties, and we have no information to perform the in-depth analysis on them with typical compiler verification techniques. We have proposed an indirect demonstration technique [21], which uses the VIS equivalence checking and (HW/SW) co-simulation [25]. It is our current on-going research issue.

3 Case Study

We performed a case study with a preliminary version of FBD programs [8] of the KNICS APR-1400 RPS BP. Starting from the FBD program, the *NuDE 2.0* seamlessly transformed the C and Verilog programs for the PLC and FPGA platforms, respectively. (Fig.4) depicts an overview of the case study we performed.

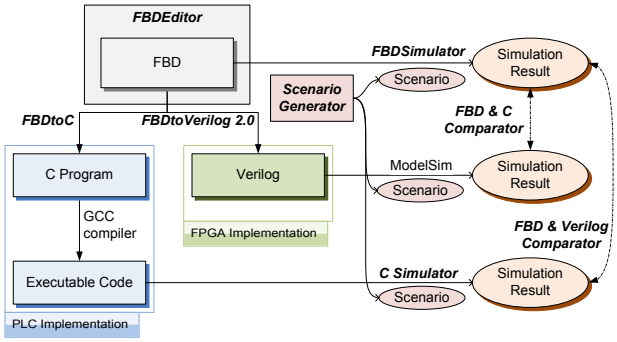


Figure 4 An overview of the case study

We also performed an exhaustive simulation of the two implementation programs in order to validate the transformations. We have developed a co-simulator which can execute C and Verilog programs simultaneously and confirm their sequential equivalence as [26].

The preliminary FBD programs are two of 18 independent logics of the RPS BP, which read sensor inputs and decide shutdown of the reactor periodically. The two are fixed set- point logics; one is a rising trip logic; and the other is a falling one. (Fig.5) shows a partial FBD program of the rising trip logic, which is designed using 'FBD Editor.' The case study performs translation from the FBD programs into C programs and Verilog programs using '*FBDtoC*' and '*FBDtoVerilog 2.0*' respectively. After the translation, we simulate the programs using simulators—FBD simulator, C simulator, and ModelSim—to demonstrate sequential equivalence between the programs. The simulations have to take the same input sequence to confirm the equivalence. The data formats, however, are different because language and simulator are difference. '*Scenario Generator*' generates virtual input data of sensors in three different formats for FBD simulator, C simulator, and ModelSim. Two comparators, '*FBD & C Comparator*' and '*FBD & Verilog Comparator*,' compare the simulation results which are output sequences of the each program

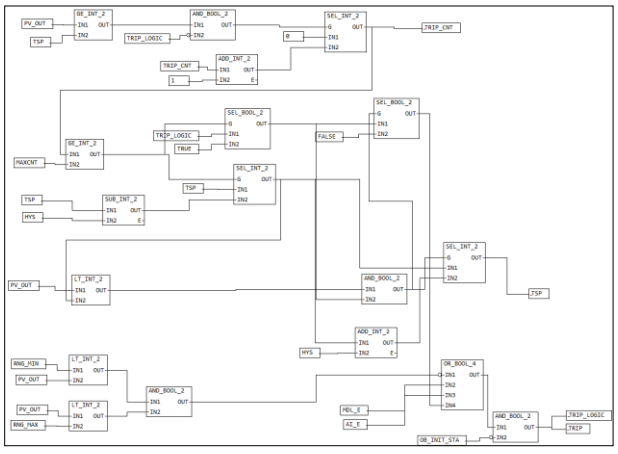


Figure 4 A part of the FBD program, 'fixed set-point rising trip'

3.1 The PLC Implementation

'*FBDtoC*' mechanically transforms FBD programs into C programs to implement PLC's programs. We transformed the two FBD programs into C programs using the '*FBDtoC*.' It generated 5 files - one is a header file and the others are C code files. The header file defines basic information, such as a data structure or interfaces. The four C code files, Function_Block.c, Component_FBD.c, System_FBD.c and Software_FBD.c, are hierarchically organized. Function_Block.c includes basic functions, such as addition or selection, and Software_FBD.c includes top functions which implement operational function for the PLC. (Fig.6) represents the transformed 'fixed set-point rising trip logic.' Only two of the transformed files are meaningful in the example - it depends on the structure of the FBD program.

It is necessary for execution to compile the transformed program. GNU Compiler Collection (GCC) is one of the most popular compilers for C programs. We used the GCC compiler of the transformed C programs.

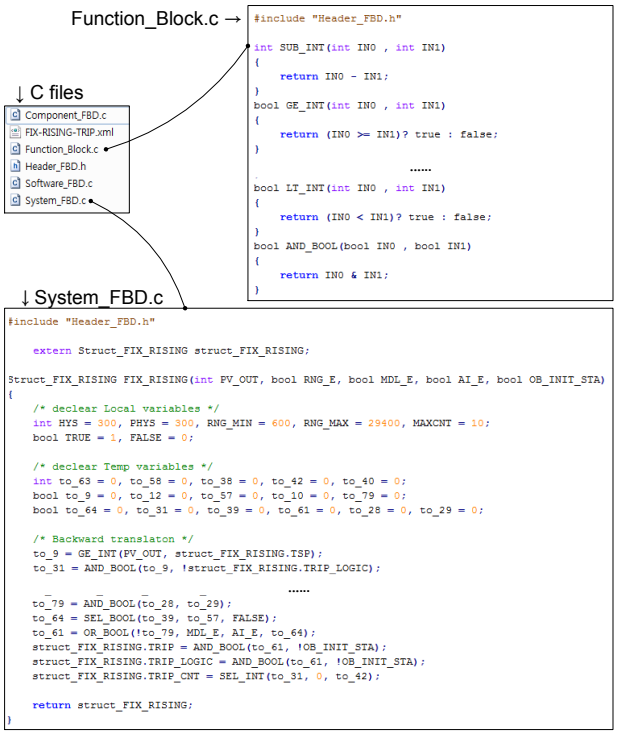


Figure 5 The result of the translation from FBD into C

3.2 The FPGA Implementation

'FBDtoVerilog 2.0' is a translator which the translated Verilog program needs pre-translated library modules, while 'FBDtoVerilog 2.1' translates all elements on-the-fly. We used 'FBDtoVerilog 2.0' with the library modules developed by experts in KAERI for the case study. Using the library modules helps the translator only focuses on the translation about the programs' interface and blocks' connections.

(Fig.7) shows translation result of the 'fix set-point rising trip logic' using the 'FBDtoVerilog 2.0.' Module call statements, which refer modules in the library, are at the middle of the transformed code, such as statements start with GE_INT_2 and LT_INT_2. The pulse signal is a unique feature to copy cyclic execution behavior of FBD programs. Verilog programs wait to store and read values of former execution result as input values synchronizing with the pulse signals.

3.3 The Equivalence Validation

We validated the behavioral equivalence between FBD versus C and FBD versus Verilog using simulation. It consists of three steps for the validation of FBD versus C programs: step-1) simulation of FBD programs; step-2) simulation of C programs; step-3) comparison of results of the two simulations from step-1 and step-2. The validation of FBD versus Verilog is a similar method. Verilog programs take the second step instead of C programs.

It is essential that pairs of the programs have to take the same input sequences to validate if they perform equivalent behavior. 'Scenario Generator' mechanically generates input sequences for FBD, C, and Verilog programs. (Fig.8) depicts screen dump of 'Scenario Generator' and a single scenario. Scenarios are able to be fully random or take several constraints, e.g., initial value, rate of change and maximum/minimum values.

'FBD Simulator,' which we developed, performs simulation of FBD programs automatically. The simulation executes tons of scenarios in a way of batch processing. We performed the simulation of the two FBD programs with 1000 scenarios for each. The scenarios are automatically generated

```
module FIX_RISING (rst, clk, pulse, PV_OUT, RNG_E, MDL_E, AI_E,
                   OB_INIT_STA, TSP, TRIP_CNT, TRIP_LOGIC, TRIP);

    input clk;
    input rst;
...
    input    OB_INIT_STA;
    output  [15:0] TSP;     reg     [15:0] TSP;
    output  [15:0] TRIP_CNT;    reg     [15:0] TRIP_CNT;
...
    parameter   [15:0] MAXCNT = 10;

    wire     GE_INT_2_wire_9_OUT;
    wire     GE_INT_2_wire_10_OUT;
    wire     LT_INT_2_wire_12_OUT;
    wire     LT_INT_2_wire_28_OUT;
    wire     LT_INT_2_wire_29_OUT;
...
    wire    [15:0] SEL_INT_2_wire_63_OUT;
    wire     SEL_BOOL_2_wire_64_OUT;
    wire    [15:0] SEL_INT_2_wire_65_OUT;
    wire  AND_BOOL_2_wire_71_OUT;
    wire  AND_BOOL_2_wire_79_OUT;

    GE_INT_2 GE_INT_2_9(rst, clk, PV_OUT, TSP, GE_INT_2_wire_9_OUT);
    GE_INT_2 GE_INT_2_10(rst, clk, SEL_INT_2_wire_63_OUT, MAXCNT, GE_INT_2_wire_10_OUT);
    LT_INT_2 LT_INT_2_12(rst, clk, PV_OUT, SEL_INT_2_wire_58_OUT, LT_INT_2_wire_12_OUT);
    LT_INT_2 LT_INT_2_28(rst, clk, RNG_MIN, PV_OUT, LT_INT_2_wire_28_OUT);
    LT_INT_2 LT_INT_2_29(rst, clk, PV_OUT, RNG_MAX, LT_INT_2_wire_29_OUT);
...
    assign TRIP = AND_BOOL_2_wire_71_OUT;

    always @(posedge rst or posedge clk or posedge pulse)
    begin
        if(rst) begin
            TSP <= 27870;
            TRIP_CNT <= 0;
            TRIP_LOGIC <= 0;
        end else if (clk) begin
        end
        if (pulse) begin
            TSP <= SEL_INT_2_wire_65_OUT;
            TRIP_LOGIC <= AND_BOOL_2_wire_71_OUT;
            TRIP_CNT <= SEL_INT_2_wire_63_OUT;
        end
    end
endmodule
```

Figure 6 The result of the translation from FBD into Verilog

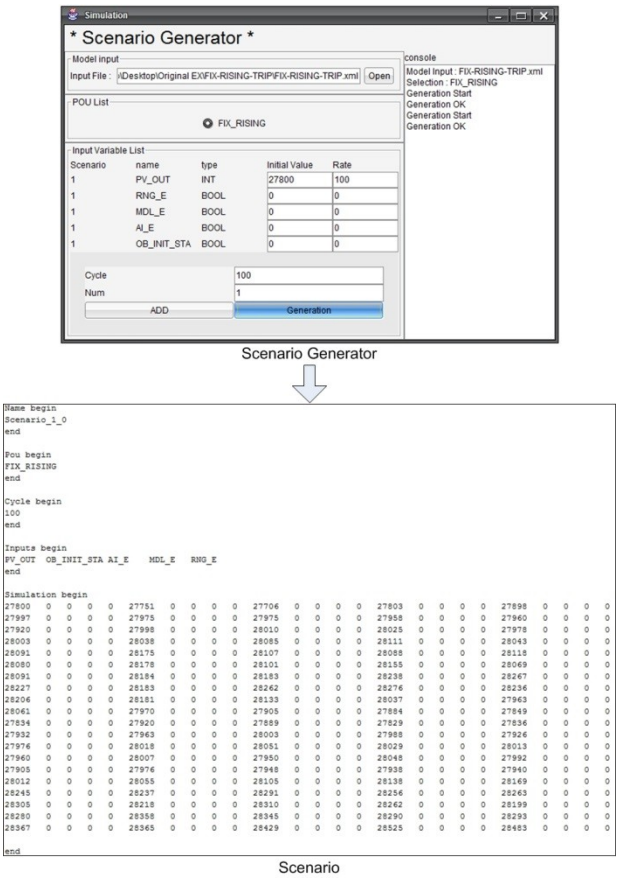


Figure 7 'FBD Simulator' and simulation results

using '*Scenario Generator*' under some constraints about the logic. (Fig.9) shows the screen dump of '*FBD simulator*' and a part of simulation results in text.

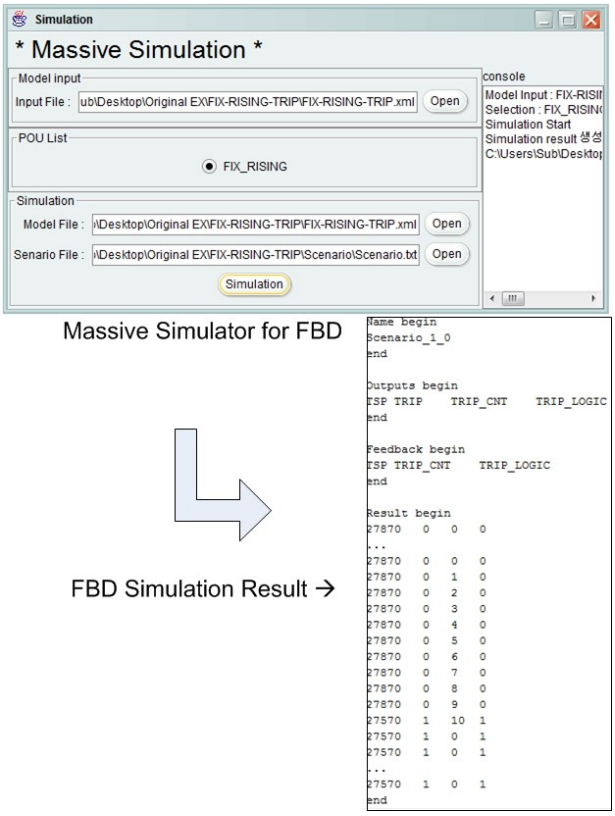


Figure 8 'FBD Simulator' and simulation results

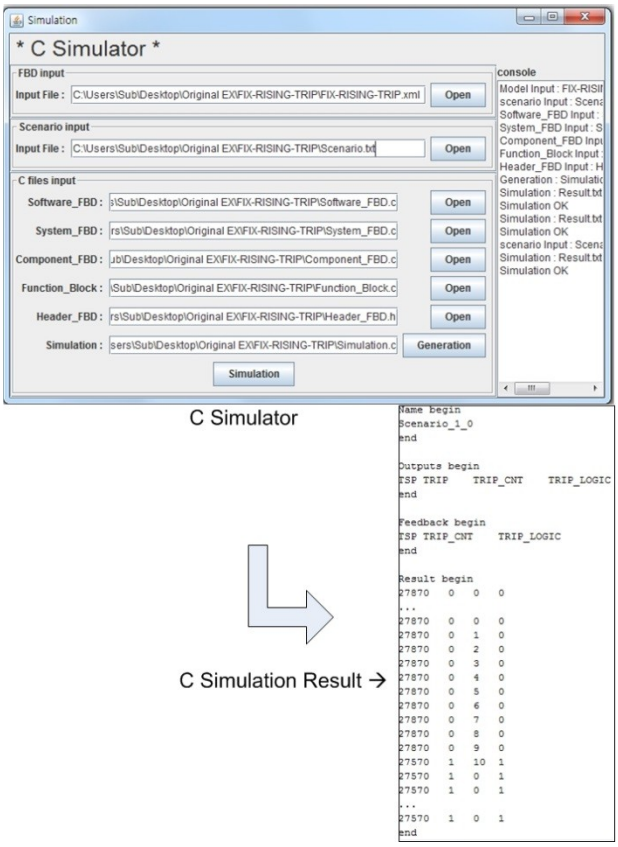


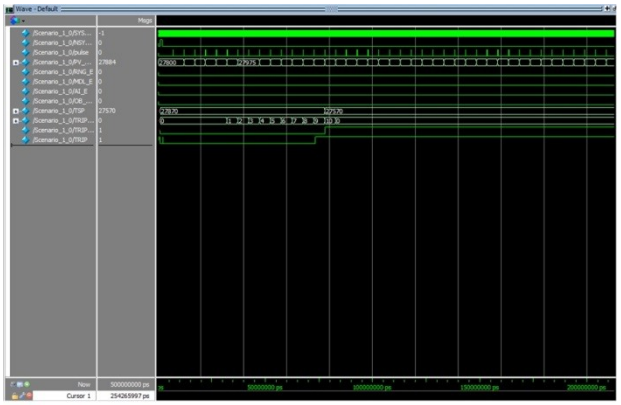
Figure 9 'C Simulator' and simulation results

'*C Simulator*' performs simulation of C programs automatically. It simulates compiled executable code not C programs just as it is. We compiled the transformed C programs using GCC compiler and simulated them using '*C Simulator*.' The simulator takes exactly the same scenario files that '*FBD Simulator*' does. The simulation, therefore, also executed 1000 scenario and generated results also in text. (Fig.10) shows the screen dump of '*C simulator*' and a part of simulation results.

Figure 10 A screen dump of 'FBD & C Comparator'

(Fig.11) is a screen-dump of the tool, '*FBD & C Comparator*,' to compare the simulation results of the two programs, FBD and C, with the same scenarios. It read a number of simulation results executed '*FBD Simulator*' and '*C Simulator*,' and compares them. The comparison makes results in True (sequentially equivalent) or False (sequentially NOT equivalent). If all simulation results are sequentially equivalent then it will make a graph of the last comparison result. On the other hand, if there is a simulation result which is not equivalent then the simulator will make a graph of it. We performed simulations with 1000 scenario for each and found that the all simulation results are sequentially equivalent.

We also validated sequential equivalent between FBD and Verilog. We used the ModelSim, which is a simulator developed by Mentor Graphics, to simulate Verilog programs. Simulation of Verilog programs, however, takes a different form of input scenarios called a test bench. '*Scenario Generator*' also provides test benches, which is the same input scenarios that '*FBD Simulator*' and '*C Simulator*' takes, for ModelSim. Naturally, we performed the simulation of the two Verilog programs with 1000 scenarios. The simulator provides the simulation results in wave and text forms. (Fig.12) shows an example of the simulation results in the two different forms.



Verilog Simulation Result (Wave)

ps	delta	/Scenario_1_0/SYSCLK	/Scenario_1_0/NSYSRESET	/Scenario_1_0/pulse	/Scenario_1_0/PV_OUT	/Scenario_1_0/RNG_E	/Scenario_1_0/MDL_E	/Scenario_1_0/AI_E	/Scenario_1_0/OB_INIT_STA	/Scenario_1_0/TSP	/Scenario_1_0/TRIP	/Scenario_1_0/TRIP_CNT	/Scenario_1_0/TRIP_LOGIC
0	+0	0	0	0	27800	0	0	0	0	0	x	x	x
50000	+0	-1	0	0	27800	0	0	0	0	0	x	x	x
100000	+0	0	0	0	27800	0	0	0	0	0	x	x	x
150000	+0	-1	0	0	27800	0	0	0	0	0	x	x	x
200000	+0	0	0	0	27800	0	0	0	0	0	x	x	x
250000	+0	-1	0	0	27800	0	0	0	0	0	x	x	x
300000	+0	0	0	0	27800	0	0	0	0	0	x	x	x
350000	+0	-1	0	0	27800	0	0	0	0	0	x	x	x
400000	+0	0	0	0	27800	0	0	0	0	0	x	x	x
450000	+0	-1	0	0	27800	0	0	0	0	0	x	x	x
450000	+2	-1	0	0	27800	0	0	0	0	0	x	x	1
500000	+0	0	0	0	27800	0	0	0	0	0	x	x	1
550000	+0	-1	0	0	27800	0	0	0	0	0	x	x	1
600000	+0	0	0	0	27800	0	0	0	0	0	x	x	1
650000	+0	-1	0	0	27800	0	0	0	0	0	x	x	1
700000	+0	0	0	0	27800	0	0	0	0	0	x	x	1
750000	+0	-1	0	0	27800	0	0	0	0	0	x	x	1
800000	+0	0	0	0	27800	0	0	0	0	0	x	x	1
850000	+0	-1	0	0	27800	0	0	0	0	0	x	x	1
900000	+0	0	0	0	27800	0	0	0	0	0	x	x	1
950000	+0	-1	0	0	27800	0	0	0	0	0	x	x	1
1000000	+0	-1	*	0	27800	0	0	0	0	0	x	x	1
1000000	+2	0	*	0	27800	0	0	0	0	0	x	x	0
1000000	+3	0	*	0	27800	0	0	0	0	0	27870	0	0
1050000	+0	-1	*	0	27800	0	0	0	0	0	27870	0	0
1100000	+0	0	*	0	27800	0	0	0	0	0	27870	0	0
1150000	+0	-1	*	0	27800	0	0	0	0	0	27870	0	0
1200000	+0	0	*	0	27800	0	0	0	0	0	27870	0	0
1250000	+0	-1	*	0	27800	0	0	0	0	0	27870	0	0
1300000	+0	0	*	0	27800	0	0	0	0	0	27870	0	0
1350000	+0	-1	*	0	27800	0	0	0	0	0	27870	0	0
1400000	+0	0	*	0	27800	0	0	0	0	0	27870	0	0
1450000	+0	-1	*	0	27800	0	0	0	0	0	27870	0	0
1500000	+0	0	*	0	27800	0	0	0	0	0	27870	0	0
1550000	+0	-1	*	0	27800	0	0	0	0	0	27870	0	0
1600000	+0	0	*	0	27800	0	0	0	0	0	27870	0	0
1650000	+0	-1	*	0	27800	0	0	0	0	0	27870	0	0
1700000	+0	0	*	0	27800	0	0	0	0	0	27870	0	0
1750000	+0	-1	*	0	27800	0	0	0	0	0	27870	0	0
1800000	+0	0	*	0	27800	0	0	0	0	0	27870	0	0

Verilog Simulation Result (Text)

Figure 11 Verilog simulation using ModelSim

'*FBD & Verilog Comparator*,' which was developed for automatic comparison, takes the two simulation results from '*FBD Simulator*' and ModelSim. It also produces True or False whether the two are sequentially equivalent or not likewise '*FBD & C Comparator*.' (Fig.13) is a screen-dump of '*FBD & Verilog Comparator*' and comparison results. The results of the 1000 comparisons with the 1000 input scenarios were True.

Figure 12 A screen dump of 'FBD & Verilog Comparator'

4 Conclusion and Future Work

This paper introduced 'NuDE 2.0', which is an integrated software development framework for two kinds of digital I&C platform, PLC and FPGA. 'NuDE 2.0' includes various CASE tools not only for software development but also language translation, translation validation, etc. We performed a case study with two logics in a preliminary version of an FBD program in order to demonstrate the sequential equivalence between two programs - FBD and C; FBD and Verilog.

We are now planning to increase confidence and thoroughness of the process to implement FPGA from Verilog. Various techniques, such as formal verification, simulation, testing, etc., are in consideration to validate equivalence between development steps or to evaluate suitability of

COTS tools. We are also trying to demonstrating safety and correctness of '*FBDtoC*' using the proposed technique in [26]. The demonstration will perform simulation, model checking, and safety analysis using various CASE tools under development.

Acknowledgement

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. She, "Investigation on the benefits of safety margin improvement in candu nuclear power plant using an fpga-based shutdown system," Ph.D. dissertation, The University of Western Ontario, 2012.

[2] J.-G. Choi, "Survey of the CPLD/FPGA Technology for Application to NPP Digital I&C System," Korea Atomic Energy Research Institute, Tech. Rep., 2009.

[3] J. Yoo, J.-H. Lee, and J.-S. Lee, "A Research on Seamless Platform Change of Reactor Protection System from PLC to FPGA," Nuclear Engineering and Technology, vol. 45, no. 4, pp. 477–488, 2013.

[4] 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.

[5] J.-H. Lee and J. Yoo, "NuDE: Development Environment for Safety- Critical Software of Nuclear Power Plant," in Transactions of the Korean Nuclear Society Spring Meeting 2012, 2012, pp. 1154–1155.

[6] J. Yoo, E.-S. Kim, D.-A. Lee, J.-G. Choi, Y. J. Lee, and J.-S. Lee, "NuDE 2.0: A Model-based Software Development Environment for the PLC & FPGA based Digital Systems in Nuclear Power Plants," in ISIC, 2004, submitted.

[7] D.-A. Lee, E.-S. Kim, and J. Yoo, "FBDtoVerilog 2.0: An automatic translation of FBD into Verilog to develop FPGA," in 5th International Conference on Information Science and Application (ICISA 2014), 2014, pp. 447–450.

[8] KAERI, "Software design specification for reactor protection system KNICS-RPS-SD231," Korea Atomic Energy Research Institute, Tech. Rep., 2006, rev.02.

[9] J. Yoo, T. Kim, S. Cha, J.-S. Lee, and H. S. Son, "A Formal Software Requirements Specification Method for Digital Nuclear Plants Protection Systems," Journal of Systems and Software, vol. 74, no. 1, pp. 73–83, 2005.

[10] E. Jee, S. Jeon, S. Cha, K. Koh, J. Yoo, G. Park, and P. Seong, "FBD Verifier: Interactive and Visual Analysis of Counterexample in Formal Verification of Function Block Diagram," Journal of Research and Practice in Information Technology, vol. 42, no. 3, pp. 255–272, August 2010.

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

[12] T. Kim, J. Yoo, and S. Cha, "A Synthesis Method of Software Fault Tree from NuSCR Formal Specification using Templates," Journal of the Korean Institute of Information Scientists and Engineers - Software and Application (in Korean), vol. 32, no. 12, pp. 1178–1191, 2005.

[13] J. Yoo, S. Cha, C. H. Kim, and D. Y. Song, "Synthesis of FBD-based PLC Design from NuSCR Formal Specification," Reliability Engineering and System Safety, vol. 87, no. 2, pp. 287–294, 2005.

[14] D.-A. Lee, E.-S. Kim, Y.-J. Seo, and J. Yoo, "FBDEditor: An FBD Design Program for developing Nuclear Digital I&C Systems," in Korea Conference on Software Engineering (KCSE 2014), 2014, pp. 315–318, in Korean.

[15] E. Jee, J. Yoo, S. Cha, , and D. Bae, "A Data Flow-based Structural Testing Technique for FBD Programs," Information and Software Tech- nology, vol. 51, no. 7, pp. 1131–1139, 2009.

[16] R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S.-T. Cheng, S. A. Edwards, S. P. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. K. Ranjan, S. Sarwary, T. R. Shiple, G. Swamy, and T. Villa, "VIS : A system for verification and synthesis," in the Eighth International Conference on Computer Aided Verification, CAV'96, 1996, pp. 428–432.

[17] 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.

[18] S. Jung, J. Yoo, and S. Cha, "VIS Analyzer : A Visual Assis- tant for VIS Verification and Analysis," in The 13th IEEE Com- puter Society symposium dealing with the rapidly expanding field of object/component/service-oriented real-time distributed computing (ORC) technology, ISORC 2010 Symposium, 2010.

[19] Y.-J. Seo, D.-A. Lee, and J. Yoo, "A Mechanical Fault Tree Construction and Analysis for Function

Block Diagrams," in APSEC, 2004, submitted.
[20] 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.
[21] E.-S. Kim, , J. Yoo, J.-G. Choi, J.-Y. Kim, and J.-S. Lee, "A Correctness Verification Technique for Commercial FPGA Synthesis Tools," in ISSRE, 2004, submitted.
[22] D.-A. Lee, J. Yoo, and J.-S. Lee, "A Systematic Verification of Behav- ioral Consistency between FBD Design and ANSI-C Implementation Using HW-CBMC," Reliability Engineering and System Safety, vol. 120, no. 12, pp. 139–149, 2013.
[23] T. Hoare, "The verifying compiler: A grand challenge for computing research," Journal of the ACM, vol. 50, no. 1, pp. 63–69, 2003.
[24] X. Leroy, "Formal Verification of a Realistic Compiler," Communication of the ACM, vol. 52, no. 7, pp. 107–115, 2000.
[25] 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 Engineer- ing, vol. 98, no. 6, pp. 414–444, 2014.
[26] E.-S. Kim, D.-A. Lee, J. Yoo, J.-G. Choi, and J.-S. Lee, "The Safety and Correctness Demonstration of the FBDtoVerilog Translator: Strategy and Case Study," in ISSRE, 2004, submitted.