A Verification Framework for FBD based Software in Nuclear Power Plants

Junbeom Yoo
Div. of Computer Science and Engineering
Konkuk University
Seoul, Republic of Korea
Email: jbyoo@konkuk.ac.kr

Sungdeok Cha
Dept. of Computer Science and Engineering
Korea University
Seoul, Republic of Korea
Email: scha@korea.ac.kr

Eunkyoung Jee
Dept. of Electrical Engineering and Computer Science, KAIST
Daejeon, Republic of Korea
Email: ekjee@dslab.kaist.ac.kr

Abstract

Formal verification of Function Block Diagram (FBD) based software is an essential task when replacing traditional relay-based analog system with PLC-based software in nuclear reactor protection system (RPS). FBD programs are developed manually and revised frequently in process of development. There are a set of properties to be verified formally, which all FBD releases should satisfy. Whenever FBDs are modified, there is also a need to verify behavioral equivalence of subsequently modified FBDs. This paper proposes a software verification framework for FBD software in nuclear power plants. It uses SMV model checker for verifying whether an FBD meets its required properties, and VIS verification system for checking behavioral equivalence between modified FBDs. A case study, conducted using a nuclear power plant shutdown system being developed in Korea, demonstrated that the proposed verification framework is effective and useful.

1. Introduction

Software safety [1] became an important issue for embedded real-time control systems. When verifying safety-critical software, formal methods [2] play critical roles in demonstrating compliance to several regulatory requirements. KNICS [1] [3] project used NuSCR [4], a formal specification language and toolset based on SCR [5,6], to formally specify and verify software requirements for nuclear reactor protection system (RPS) of APR-1400 nuclear power reactor. Formal verification techniques such as model checking [7, 8] were also used to verify critical system properties [21].

PLCs (Programmable Logic Controllers) are widely used to implement safety-critical control software, and IEC [9] defined five programming languages as international standards for PLCs. KNICS decided to use FBD as a standard representation of software design, because it can visually expresses controller behavior as interconnected operation of function blocks. It is common for FBD engineers to develop it manually from requirements specification and to revise it frequently to reflect new or modified requirements in process of development.

Safety demonstration is the most important quality aspect that must be rigorously demonstrated throughout entire life-cycle phases of safety-critical software systems. Therefore all modifications made to FBDs require safety demonstration. RPS software of APR-1400 advanced nuclear power reactor, in development in Korea, is such an example. While inspection technique is useful, it alone is inadequate to meet rigorous regulatory requirements. This paper proposes a software verification framework for FBD software used in nuclear power plant's reactor protection systems.

The software verification framework uses two different verification techniques to verify FBD software thoroughly. It uses Cadence SMV model checker [10] for verifying whether an FBD meets its required properties, and also uses VIS verification system [11] for checking behavioral equivalence between subsequently modified FBDs. For these we first translate FBD programs into Verilog [12] programs according to the translation rules proposed in [13, 14] and using automatic translation program we developed in [26].

[1] Goal of KNICS consortium project (2001~2008) is to develop a suite of I&C software for use in the next generation Korean nuclear power plants.

1530-1362/08 $25.00 © 2008 IEEE
DOI 10.1109/APSEC.2008.26

The remainder of the paper is organized as follows: Section 2 introduces background information on FBD programming, Verilog, SMV model checking, and VIS equivalence checking. Section 3 shows the software verification framework for FBD software in nuclear power plants. Application of the proposed technique is demonstrated in Section 4. Section 5 presents related work, and we conclude the paper at Section 6.

2. Background

2.1. FBD Programming in PLC

Programmable Logic Controller (PLC), widely used in real-time and embedded control applications [16], has relatively simple architecture. Sensors and actuators are plugged in via input and output channels, relatively. Operating system manages periodic execution of PLC applications by reading all input values at the beginning of each execution cycle and updating values of system variables. It then performs predefined computation and generates output values at the end of the cycle. Simplified architecture and processing mechanism make PLC an attractive platform for implementing embedded application software, i.e. chemical processing plants, nuclear power plants and traffic control systems.

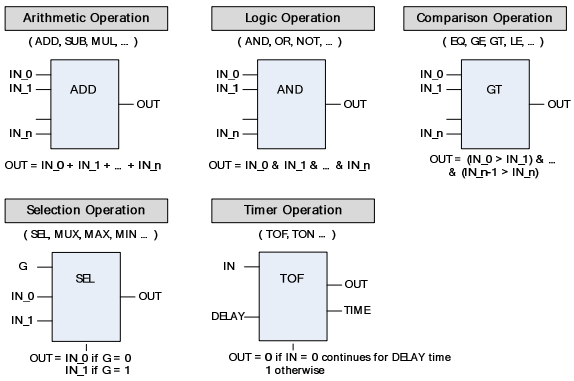

Figure 1. Representative function blocks defined in IEC 61131-3

There are several PLC programming languages. The IEC 61131-3 [9] standard includes five languages: Structured Text (ST), Function Block Diagram (FBD), Ladder Diagram (LD), Instruction List (IL) and Sequential Function Chart (SFC). FBD is frequently used due to its graphical notations and support for a network of function blocks "wired" together in a manner similar to a circuit diagram. Function blocks, each of which is depicted as a rectangle and connected to input/output variables, are classified into several categories as shown in Fig.1. We present five in ten categories, which are pertinent to our discussion.

Fig.2 shows a partial implementation of RPS BP (Bistable Processor) logic for APR-1400. It creates a warning signal *th_X_pretrip* when any safety-threatening situation occurs. Sequential combination of seven function blocks produces an output *th_X_Pretrip*, and the number in parenthesis above each function block denotes execution order. For example, GE_INT function block, numbered (11) is executed first and produces 1 as its block output if input value *f_X* is greater than or equal to *k_X_Pretrip_Setpoint*. Otherwise, 0 is the block's output value. The output is then negated as denoted as small circle attached to input of the SEL block numbered 14. Next, SUB_INT numbered (12) subtracts *k_X_Pretrip_Hys* value from *k_X_Pretrip_Setpoint* and the result is compared against *f_X* in the LE_INT numbered (13). Computations take place sequentially according to the defined behavior of each block (see Fig.1). Of the two FBD outputs, *th_Prev_X_Pretrip* value is used to store current value of *th_X_Pretrip*.

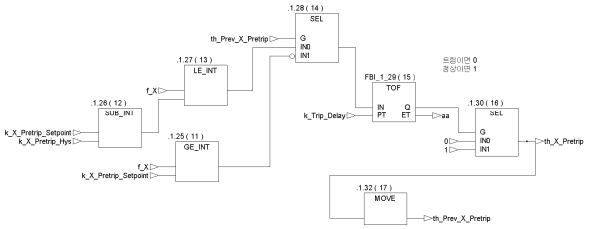

Figure 2. An FBD for *th_X_Pretrip* logic[2]

2.2. Verilog

Verilog [12] is one of the most popular HDL (Hardware Description Language) used by IC (Integrated Circuit) designers. Verilog allows software design to be simulated earlier in design cycle to correct errors and experiment with different hardware architecture. Several characteristics of Verilog make it possible to use it easily in software design too. Procedural assignment statements, module and function calls, and the manner of I/O of Verilog are very similar to those of C/C++, basic software design languages. These characteristics make software engineers who design and verify FBD programs feel more comfortable with learning and training. Software engineers familiar with procedural programming

[2] The FBD in Fig. 2 was developed using Concept ver. 2.2 XL SR2, a PLC programming assistant tool marketed by Schneider Automation GmbH.

languages like C/C++ can use the Verilog without much effort. It is one of major advantage of Verilog compared to other HDLs.

2.3. SMV Model Checking

The proposed verification framework uses model checking technique to verify FBD programs formally. Model checking is a technique to prove whether a formal specification satisfies required properties or not. Cadence SMV [10] is a model checker based on symbolic model checking technique [17]. It can verify a model programmed in Synchronous Verilog (SV) [18], a slight variation of the Verilog language with cycle-based behavior. Cadence SMV's *vl2smv* function converts the Synchronous Verilog into SMV input language, and then performs model checking. *True* is returned if Verilog model meets given properties, otherwise, a counter-example is produced to demonstrate the existence of errors in the Verilog model.

2.4. VIS Equivalence Checking

VIS (Verification Interacting with Synthesis) [11] is a tool that integrates verification, simulation and synthesis of finite state hardware system. It uses Verilog as a front end and supports fair Computational Tree Logic (CTL) model checking, language emptiness checking, combinational and sequential equivalence checking, cycle-based simulation, and hierarchical synthesis. As VIS has the capability to interface with SIS [19] to optimize logic modules, it is an integrated system for hierarchical synthesis as well as verification. More detailed introduction to its structure and functions goes out of scope, so we introduce equivalence checking briefly what we concern in this paper.

VIS provides the capability to test sequential and combinational equivalence of two designs. An important usage of combinational equivalence is to provide a sanity check for re-synthesizing portions of a combinational logic. Sequential equivalence checking is done by building the product of finite state machine, and checking whether a state where the values of two corresponding outputs differ can be reached from the set of initial states of the product machine. If this happens, a debug trace is provided. Unfortunately, VIS has no graphical UI support yet.

3. A Verification Framework

This section introduces a software verification framework for FBD software in nuclear power plant's reactor protection system. A typical safety-critical system, RPS in APR-1400 advanced power reactor, is being developed and implemented on PLCs by KNICS project. Its whole software development process is described in Fig.3. Details are introduced in [20, 14].

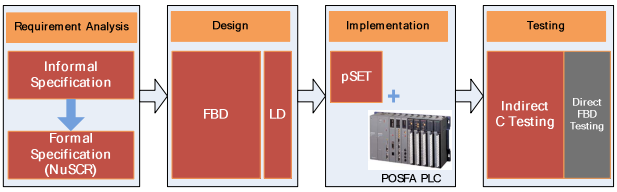

Figure 3. A software development process for KNCIS's APR-1400 RPS

In the software development process for APR-1400 RPS, software requirements are derived from an informal natural language specification first and then rewritten in NuSCR formal specification language [4]. Formal specification mandates developers to specify all requirements explicitly and completely without any assumptions or omissions, so use of formal specification is strongly recommended by government authorities like KINS [24]. A formal verification, model checking was performed on NuSCR formal specification to verify important properties as presented in [21].

In design phase, FBD programs are manually developed by FBD engineers from the requirement specification. A synthesis technique proposed in [22] could support mechanical synthesis of FBD programs from NuSCR formal specification, but its lack of automatic tool-support made it difficult to be used in the development process.

In implementation phase, an engineering tool named *pSET* [15] translates FBD programs into executable codes for PLC. The engineering tool also generates intermediate C code for testing purposes. There is our on-going research on direct FBD testing [25]. PLC-based software development is finished when FBD program has been adequately tested using the generated C code.

Initial FBD design is revised when additional requirements are introduced or optimization is performed to avoid redundant implementation. Although FBD modifications might be minor in scope, they might still give rise to subtle behavioral changes and result in safety critical errors. Therefore, one must

always demonstrate that required behavior is preserved as illustrated in Fig.4. The proposed verification framework depicted in Fig.4 uses two different formal verification techniques. While VIS verification system can check behavioral equivalence of subsequently modified FBD programs, SMV model checker verifies the FBD program whether it satisfies properties or not. These different verification techniques works together to demonstrate safety of the final FBD program, FBD_N.

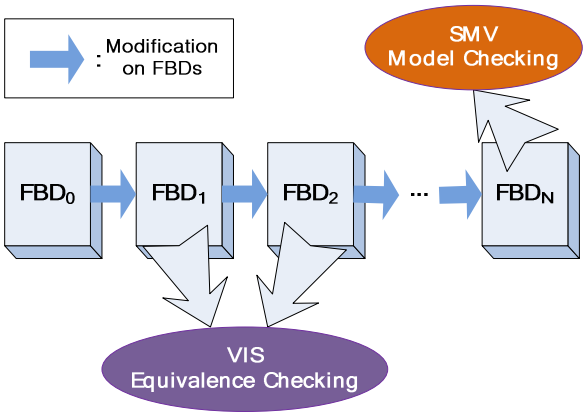

Figure 4. A verification framework for FBD software

3.1. VIS Equivalence Checking

The verification framework uses VIS's equivalence checking to preserve behavioral equivalence between subsequently modified FBD programs. For this purpose, first we translate FBDs into equivalent Verilog programs, and perform equivalence checking using VIS as described in Fig.5 below.

FBD programs are stored in *.ld* format in the engineering tool *pSET*, and *FBD Verifier 1.0* [26] we developed translates them into equivalent Verilog programs in *.v* format. As VIS verification system has no graphical user interface, we execute the VIS in *Cygwin* environment and check their equivalence. A program named *vl2mv* [27] in VIS verification system translates Verilog program in *.v* into *.mv* format which VIS can read and analyze.

If any "*NOT equivalence*" occurs, VIS shows a counter example describing the situation - sequences of changed value of variables. Up to now, we analyze the counter-examples manually with FBD engineers to find a precise cause of the not equivalence. However, the automation and visualization of VIS analysis on which we are currently focusing will promote efficiency of the manual analysis.

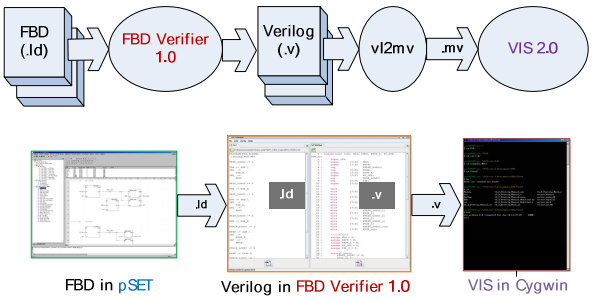

Figure 5. An overview of VIS equivalence checking

3.2. SMV Model Checking

SMV model checking in the verification framework verifies Verilog program translated from FBD program whether it satisfies important properties or not. *FBD Verifier 1.1* reads FBD programs in *.ld* format and translates them into Verilog programs in *.v* format. *FBD Verifier 1.1* also translates the Verilog program into SMV input program (*.smv*) automatically using *vl2smv* program in *Cadence SMV*. Fig.6 describes the SMV model checking process in the framework.

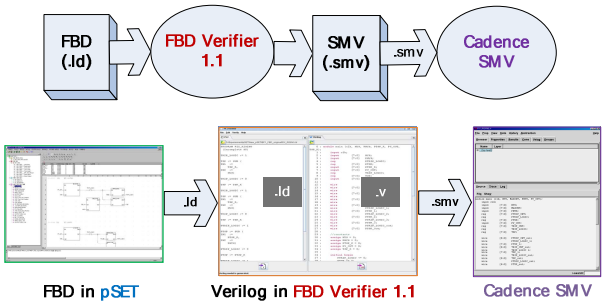

Figure 6. An overview of SMV model checking

Properties to be checked are written in LTL formula within translated Verilog programs. The Cadence SMV returns *TRUE* if Verilog meets give properties, otherwise, returns a counter- example. In addition to translation function, *FBD Verifier 1.1* also supports analysis of counter examples by visualizing graphically changing status of variables [26].

4. Case Study

We applied the proposed verification technique to APR-1400 RPS [23] which is being developed in Korea. The RPS is composed of Bistable Processor (BP) and Coincidence Processor (CP). While we applied VIS equivalence checking to a part of BP, we

applied SMV model checking to whole FBDs of RPS BP and CP.

4.1. VIS Equivalence Checking

VIS Equivalence checking aims for verifying behavioral equivalence between two different FBDs. Therefore, there is no official experimental result left for demonstrating its usefulness, unfortunately. VIS's non-graphical interface also made it difficult for FBD engineers to use the technique easily and efficiently. In spite of its lack of official experimental result, VIS equivalence checking technique's usefulness was well recognized by domain engineers.

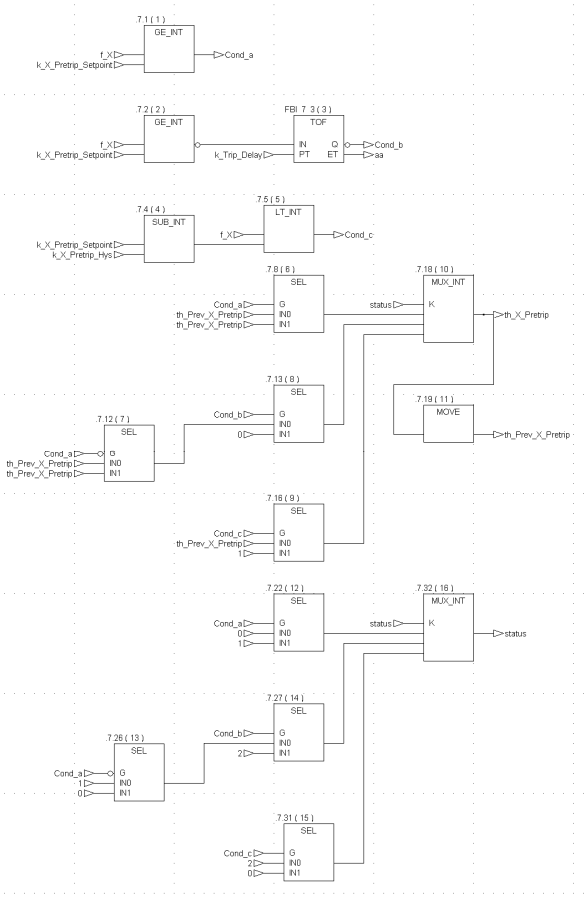

Figure 7. A mechanically synthesized FBD for *th_X_Pretrip* logic

In this subsection, we introduce some experiments on early version of BP FBDs, Rev.00 draft [36]. The VIS equivalence checking technique was originally developed for the purpose of supporting FBD synthesis technique proposed in [22]. We could synthesize FBDs mechanically from NuSCR formal specification, but it was not applied to the actual development of RPS yet. No automatic tool support for the FBD synthesis technique was a huge obstacle to apply it in earnest. Therefore, there were two kinds of FBDs, manually developed FBDs and mechanically synthesized FBDs, from the same NuSCR formal requirements specification. Fig.7 depicts an FBD which was synthesized mechanically from the same NuSCR requirement specification as the manually developed FBD depicted in Fig.2. The FBD in Fig.2 is an optimized one which was manually modified by domain experts. While the mechanically synthesized FBD in Fig.7 is composed of 15 function blocks, the optimized FBD in Fig.2 has only 7 function blocks.

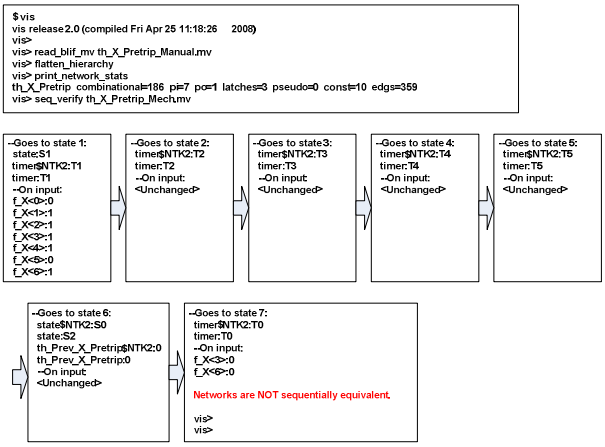

Figure 8. A counter-example of VIS equivalence checking

As VIS counter-example does not show the different output values explicitly in final state, we must use simulation facility of VIS to investigate the cause more precisely and fix the errors. We are expecting that the VIS analysis supporting tool which we are currently focusing will solve the inefficiency and inconvenience well. Details are beyond the scope of this paper, but domain engineers found an error in the position of TOF function block, and modified it as shown in Fig.9. VIS equivalence checking proved that these two FBDs have the same behavior.

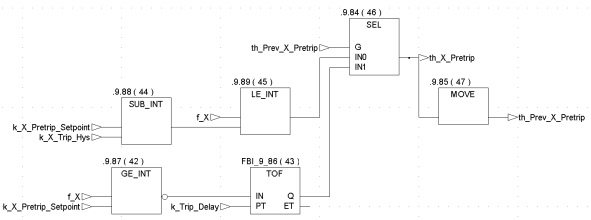

Figure 9. A modified FBD for *th_X_Pretrip* logic

We applied the VIS equivalence checking to several kinds of trip (shutdown of nuclear reactor) logic in RPS BP, early version of BP FBDs Rev.00 draft [36], and Table 1 depicts the result. We, surprisingly, found several critical logic errors in both the manual and synthesized FBDs. Such errors might be difficult to detect using other techniques such as testing or inspection, and later releases of specification were corrected. Domain engineers felt that FBD verification made an important contribution in ensuring safety of PRS implementation.

TABLE 1. VIS equivalence checking result for parts of BP

Trip Logic for BP	Error Type	Mechanically Synthesized FBDs	Manually Developed FBDs
Fixed set-point rising trip without operating bypass	Syntax	0	0
	Logic	0	1
Manual reset variable set-point trip with operating bypass	Syntax	0	3
	Logic	6	2

4.2. SMV Model Checking

In contrast to VIS equivalence checking, we applied SMV model checking to the whole RPS BP and CP, FBD Rev.02 [23]. Table 2 describes their size and complexity. We used Cadence SMV model checker to verify whether the Verilog program meets important properties or not. *FBD Verifier 1.1* translates FBD programs (*.ld*) in *pSET* into Verilog programs (*.v*), and then executes *vl2smv* function in Cadence SMV to translate it into SMV input programs (*.smv*). *FBD Verifier 1.1* also executes Cadence SMV automatically using the translated SMV programs as inputs. Properties to be verified for RPS were developed by cooperation of nuclear engineers and software engineering engineers. Domain experts provided important properties specified in natural language, and software engineering engineers encoded them into proper logical expressions. These properties belong to safety properties which can be specified with LTL easily as classified in Table 3.

TABLE 2. RPS system information

System	# of pages of requirements specification (Natural lang.)	# of function blocks	# of variables	# of lines in Verilog model
BP	190	1,335	1,038	7,862
CP	163	1,623	820	3,085

TABLE 3. Examples of verification properties for RPS BP and CP

No.	Properties
1	When the trip condition is satisfied, a trip should occur.
2	When the trip release condition is satisfied, a trip should release.
3	Trip set-point value should be in valid range.
4	When trip and pretrip did not occur, trip set-point and pretrip set-point should keep the specified difference.
5	When the processing value is in invalid range, a range error should occur.
6	When the heartbeat of the other system is unsound, a heartbeat error should occur.

We verified the BP thoroughly with suggested properties by domain engineers depicted in Table 3, and as a result, 47 errors were found. With the exclusion of repeated errors with the same cause, 10 distinct errors were found. In addition to the BP system, we also verified the CP successfully. Table 4 summarizes the verification result. Detailed analysis of the verification result is beyond the scope of this paper [37], but it is worth to introduce typical causes of errors analyzed from the verification result.

TABLE 4. Verification result for RPS

System		BP	CP
# of Properties		216	83
Found Errors	**Incorrect Logic**	14	6
	Omission	0	2
	Ambiguous Logic	4	0
	Incorrect FBD	13	5
	Incorrect Design	16	0
Total # of Errors		47	13
Distinct # of Errors		10	3

Typical causes of errors in RPS system were as follows: misused variable name (e.g. use of *TRIP_LOGIC* variable instead of *1_TRIP_LOGIC* variable), misused operator (e.g. use of >= instead of >), omitted range check, undetermined values at initial phase, unmatched specification between natural language specification and FBDs, and not removed temporary test logic. Most of these errors had not been detected with other activities such as inspection, traceability analysis, and safety analysis. We reported found errors to RPS developing engineers, and they were satisfied with the verification result and modified the RPS program based on our verification result. Topical report for the RPS was submitted to the regulation authority KINS in order to get safety approval and the approval result is about to come out.

5. Related Work

In recent years, demand for PLC program verification on safety has been growing. Several research projects addressed FBD verification issues, and [16, 28] provide surveys on verification of PLC programming languages LD and ST.

[29] used higher order logic (HOL) to model specifications and implementations. Function blocks are modeled as relations on streams. According to the framework, there are no restrictions on data type, and time is treated implicitly on the contrary to the Verilog and VIS verification system. However, proofs are done with help of a theorem prover, *Isabelle/HOL* system [30], and it costs too much in comparison with automatic verification using VIS.

IEC 61499 [31] defined interactions between controllers and overall systems (plants) using FBDs, and [32] formalized it with Single-Net Systems (SNS) [33] model. The controller code is defined in FBD format and the overall system is organized in IEC 61499 function blocks. In this approach, the complete structure is automatically translated into Single-Net Systems (SNS) model using a tool, *VEDA*. On the combined model of plant and controller, model checking is performed using *SESA* (Signal/Event System Analyzer) [34].

In [35], a toolset called *PLCTOOLS* has been introduced. The FBD programs are modeled and described as High Level Timed Petri Nets (HLPTN), and HLTPN are used for validating the design and generating the code. *MATLAB/SIMULINK* provides means for specifying and simulating plants. *PLCTOOLS* focuses on designing, simulation, and PLC code generation, not formal verification.

6. Conclusion and Future Work

This paper proposed a verification framework for FBD-based software in nuclear power plant's reactor protection systems. Proposed framework suggests two different formal verification techniques, Cadence SMV model checker for verifying whether an FBD meets its required properties and VIS verification system for checking behavioral equivalence between subsequently modified FBDs. They work together to demonstrate safety of FBD programs.

We verified FBDs for KNICS APR-1400 RPS with proposed verification framework. A number of FBDs could be verified and analyzed effectively, and the verification result was successfully applied to official releases of FBDs. We are also currently focusing on developing VIS analysis automation tool for visualizing VIS equivalence checking process and result more efficiently. We have a plan to apply the proposed verification framework to other safety-critical systems too.

Acknowledgement

This research was supported by the Korean Research Foundation Grant funded by the Korean Government (KRF-2008-331-D00524), and by Defense Acquisition Program Administration and Agency for Defense Development under the contract. It was also partially supported by Foundation of ubiquitous computing and networking project (UCN) Project, the Ministry of Knowledge Economy (MKE) 21st Century Frontier R&D Program in Korea and a result of subproject UCN 08B3-S2-30S.

References

[1] N.G. Leveson. SAFEWARE, System Safety and Computers. Addison Wesley, 1995.

[2] D.A. Peled. SOFTWARE RELIABILITY METHODS. Springer, 2001.

[3] KNICS. http://www.knics.re.kr/english/eindex.html.

[4] 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.

[5] K.L. Heninger. Specifying software requirements for complex systems: New techniques and their application. IEEE Trans. Software Engineering, vol.SE-6, no.1, pp.2-13, 1980.

[6] A.J. Schouwen Van, D. Parnas, and J. Madey. Documentation of requirements for computer systems. In RE'93: IEEE International Symposium on Requirements Engineering, pp.198–207, 1993.

[7] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Programming Languages and Sysems, vol.8, no.2, pp.244-263, 1986.

[8] E.M. Clarke, Orna Grumberg, and Doron A. Peled. Model Checking. MIT Press, 1999.

[9] IEC(International Electrotechnical Commission). International standard for programmable controllers: Programming languages, part 3, 1993.

[10] Cadence SMV. http://www.cadence.com.

[11] R.K. Brayton, G.D. Hachtel, et. al. VIS : A system for verification and synthesis. In the 8th International Conference on Computer Aided Verification, CAV '96, pp.428–432, 1996.

[12] IEEE. IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language (IEEE Std. 1364-2001), 2001.

[13] J. Yoo and S. Cha. A Definition and Semantics of FBD for Software Verification. Journal of ETRI, submitted. 2008.

[14] J. Yoo. Synthesis of Function Block Diagrams from NuSCR Formal Specification. PhD thesis, KAIST, 2005.

[15] S. Cho, K. Koo, B. You, T.-W. Kim, T. Shim and J.S. Lee. Development of the loader software for PLC programming. Proceedings of Conference of the Institute of Electronics Engineers of Korea, vol.30, no.1, pp959-960, 2007.

[16] A. Mader. A classification of PLC models and applications. In Discrete Event Systems Analysis and Control: WODES 2000, 2000.

[17] K.L. McMillan. Symbolic Model Checking. Kluwer Academic Publishers, 1993.

[18] C.-T. Chou. Synchronous Verilog: A Proposal. Fujitsu Laboratories of America, 1997.

[19] E.M. Sentovich, K.J. Singh, L.Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical Report UCB/ERL M92/41, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, May 1992.

[20] 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 the 23th International Conference on Computer Safety, Reliability and Security (SAFECOMP 2004), LNCS 3219, pp.155–165, 2004.

[21] J. Cho, J. Yoo, and S. Cha. NuEditor - a tool suite for specification and verification of NuSCR. In the Second ACIS International Conference on Software Engineering Research, Management and Applications (SERA 2004), pp.298-304, 2004.

[22] 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.

[23] KNICS-RPS-SDS231. Rev.02, Software Design Specification for Reactor Protection System. Korea Atomic Energy Research Institute (KAERI), 2006.

[24] KINS. http://www.kins.re.kr.

[25] E. Jee, S. Jeon, H. Bang, S. Cha, J. Yoo, G. Park, K. Kwon. Testing of Timer Function Blocks in FBD, In the 13th Asia Pacific Software Engineering Conference (APSEC), pp.243-250, 2006.

[26] S. Jeon. Verification of Function Block Diagram through Verilog Tranlsation. MS thesis, KAIST, 2007.

[27] S.-T. Cheng and R.K. Brayton.Compiling Verilog into Automata. UCB ERL Technical Report M94/37, 1994.

[28] G. Frey and L. Litz. Formal moethods in PLC programming. In IEEE Conference in System Man and Cybernetics: SMC 2000, 2000.

[29] B. J. Kramer and N. Volker. A higher dependable computer architecture for safety critical control applications. Real-Time Systems Journal, vol.13, no.3,pp.237–251, 1997.

[30] T. Nipkow, L.C. Paulson and M. Wenzel. Isabelle/HOL - A Proof Assistant for Higher-Order Logic, LNCS 2283, 2002.

[31] IEC(International Electrotechnical Commission) TC65 WG6. Function blocks for industrial process measurements and control systems, 2003. Committee Draft.

[32] V. Vyatkin and H.-M. Hanisch. Modelling of IEC 61499 function blocks - a cue to their verification. In XI Workshop on Supervising and Diagnostics of Machining System, pp.59–68, 2000.

[33] P.H. Starke. Symmetries of signal-net systems. In Workshop on Concurrency, Specification and Programming, pp.285–297, 2000.

[34] P.H. Starke and S. Roch. Tools for formal specification, verification, and validation of requirements. In the 12th Annual Conference on Computer Assurance, COMPASS ’97, pp.35–47, 1997.

[35] L. Baresi, M. Mandrioli, S. Morasca, and M. Pezz`e. Plctools: Design, formal verification, and code generation for programmable controllers. In the IEEE Conference on System, Man, and Cybernetics (SMC), pp.2437–2442, Nashville, USA, Oct. 2000.

[36] KNICS-RPS-SDS101. Rev.00 Draft, SDS for Reactor Protection Systemz, Korea Atomic Energy Rearch Institute, Sept. 2003.

[37] E. Jee, S. Jeon, S. Cha and J. Yoo. FBDVerifier: Formal Verification of Function Block Diagrams through Automatic Verilog Translation. Not published.