Available online at www.sciencedirect.com
SCIENCE DIRECT®
Reliability Engineering and System Safety 87 (2005) 287–294
RELIABILITY ENGINEERING & SYSTEM SAFETY
www.elsevier.com/locate/ress

Synthesis of FBD-based PLC design from NuSCR formal specification

Junbeom Yoo[a,*], Sungdeok Cha[a], Chang Hwoi Kim[b], Duck Yong Song[c]

[a]*Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology (KAIST) and AITrc/SPIC/IIRTRC, 373-1, Kusong-dong, Yusong-gu, Taejon, South Korea*
[b]*I&C-HMI team, Korea Atomic Energy Research Institute (KAERI), 150, Deokjin-dong, Yusong-gu, Taejon, South Korea*
[c]*Nuclear Research Division, Atomic Creative Technology Ltd, 1688-5, Sinil-dong, Daedeok-gu, Taejon, South Korea*

Received 15 August 2003; accepted 21 May 2004

Abstract

NuSCR is a formal specification language to document requirements for real-time embedded software with nuclear engineering applications in mind. Domain experts actively participated in selecting how to best represent various aspects. It uses tabular notations to specify required computations and automata to document state- or time-dependent behavior. As programmable logic controllers (PLCs) are widely used to implement real-time embedded software, synthesis of PLC code from a formal specification is desirable if transformation rules can be rigorously defined. In addition to improved productivity, results of safety analysis performed on requirements remain valid. In this paper, we demonstrate how NuSCR specification can be translated into semantically equivalent function block diagram (FBD) code. The process, except the initial phase where user provides information on missing or implicit details, is automated. Since executable code can be automatically generated using CASE tools from FBD, much of software development is automated. Proposed technique is currently being used in developing reactor protection system (RPS) for nuclear power plants in Korea, and experience to date has been positive. We demonstrate the proposed approach using the fixed set-point rising trip which is one of the most complex trip logics included in the RPS.
© 2004 Elsevier Ltd. All rights reserved.

Keywords: Formal requirements specification; Design specification; PLC; FBD; Nuclear power plant controller

1. Introduction

Software safety became an important issue for embedded control systems as traditional relay-based analog systems are being replaced by software. When developing safety-critical software such as emergency shutdown system for nuclear power plants, regulation authorities require safety demonstrations throughout life-cycle phases. Requirement engineering is well-known to play critical roles to software quality, and formal specification techniques are often used to facilitate unambiguous documentation and rigorous analysis of requirements. As active participation of domain experts is essential when performing safety analysis, several domain-specific specification languages have been developed. NuSCR, based on SCR [6] notation, is a specification language designed to serve nuclear engineering industry. While adopting notations familiar to domain experts such as function overview diagrams (FODs) and structured decision tables (SDTs), state- and time-dependent functionalities are visually specified in automata notation. As domain experts who are also familiar with research on formal methods actively participated when deciding how to best capture various requirements, nuclear engineers find NuSCR easy to use and intuitive [2]. In fact, when used to document requirements for bistable processor (BP) logic as a part of the KNICS[1] project, several errors such as ambiguities in

* Corresponding author. Address: Division of Computer Science, EECS Department and Advanced Information Technology Research Institute (AITrc), Korea Advanced Institute of Science and Technology (KAIST), 373-1 Kusong-dong, Yusong-gu, Taejon 305-701, South Korea.
E-mail addresses: jbyoo@salmosa.kaist.ac.kr (J. Yoo), cha@salmosa.kaist.ac.kr (S. Cha), chkim2@kaeri.re.kr (C.H. Kim), dysong@act.actbest.com (D.Y. Song).

0951-8320/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.ress.2004.05.005

[1] Goal of the KNICS project is to develop a suite of instrumentation and control software for use in next generation Korean nuclear power plants.

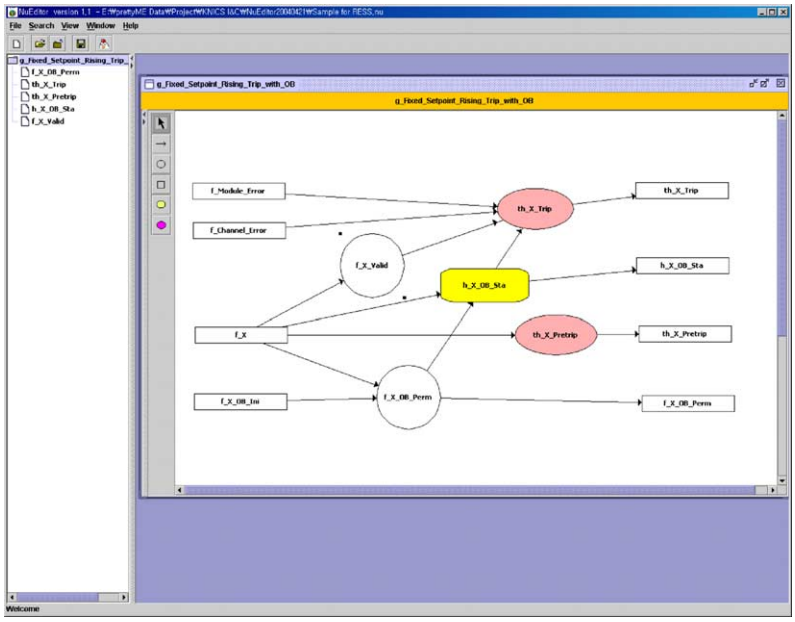


Fig. 1. FOD for *g_Fixed_Setpoint_Rising_Trip_with_OB*.

trip logics and missing initialization routines were found in documents written in English.

In the KNICS project [3], as is often the case with many embedded software projects, programmable logic controller (PLC) is used as an implementation platform. There are several widely used PLC programming languages, and function block diagram (FBD) and ladder diagrams are frequently used in industry. CASE tools like *SIEMENS TELEPERM XS/XP* [1] can automatically generate executable code from FBD. While C code can also be generated for testing purpose, such step is unnecessary if (1) adequate safety analysis can be applied on NuSCR; (2) FBD code can be synthesized from NuSCR specification; (3) transformation rules between the two are rigorously defined and verified; and (4) adequate techniques exist in testing PLC code.

This paper describes a systematic process of generating FBD-based PLC program from NuSCR specification using *fixed set-point rising trip with operating bypass* example included in the requirements for reactor protection system (RPS). After briefly introducing the trip logic in Section 2, we introduce key features of NuSCR and FBD programming. Section 3 explains NuSCR-to-FBD transformation rules, and Section 4 concludes the paper.

2. Background

2.1. RPS

KNICS digital plant protection system consists of three subsystems: RPS, engineering safety features-component control system (ESF-CCS), and automatic test and interface processor (ATIP). RPS, further divided into BP and coincidence processor (CP), implements trip logics so that the plant can be safely shut down in emergency situations such as increased reactor temperature or loss of coolant. EFS-CCS, similar in features to RPS, attempts to reduce influence of other reactor accidents while ATIP periodically tests RPS and EFS-CCS so that plant safety is always ensured. BP accepts inputs from 18 different sensors and performs required computations to monitor reactor's safety status. Among several trip logics, *fixed set-point rising trip with operating bypass* generates the trip signal (*th_X_Trip*) if input value *f_X* rises above the predefined threshold value. However, trip logic must be bypassed (*h_X_OB_Sta*) if an operator pushes the bypass button (*f_X_OB_Ini*) while the input value remains in the normal and permitted range. Different and preliminary trip logic (*th_X_Pretrip*), whose setpoint is slightly lower than that of the trip, warns an operator of potentially imminent trip. Figs. 1–3 show FOD as well as requirements for *f_X_Valid* and *th_X_Trip* in NuSCR, respectively. The system is quite complex in

Conditions		
k_X_MIN <= f_X <= k_X_MAX	T	F
Actions		
f_X_Valid := 0	X	
f_X_Valid := 1		X

Fig. 2. SDT for *f_X_Valid*.

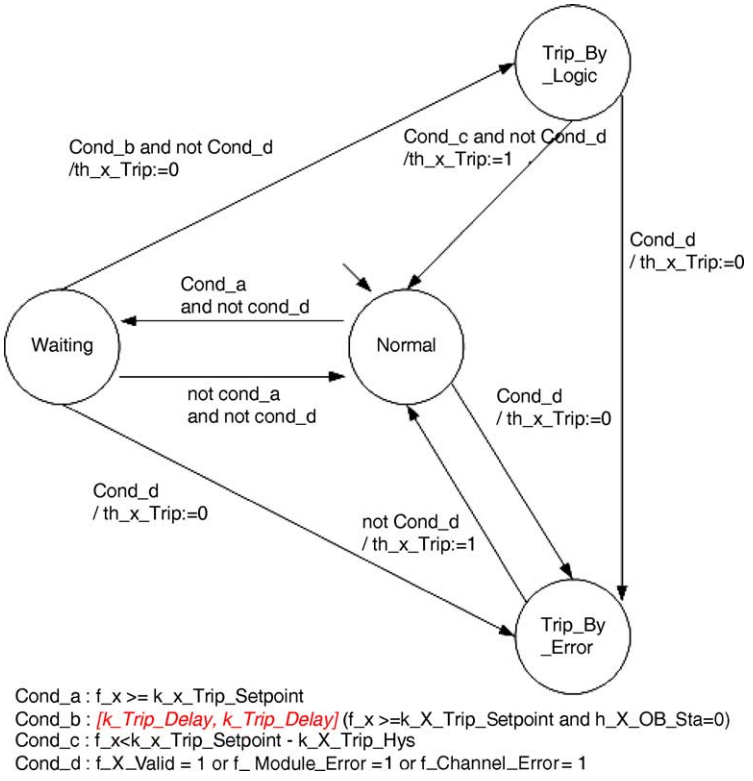


Fig. 3. TTS for *th_X_Trip*.

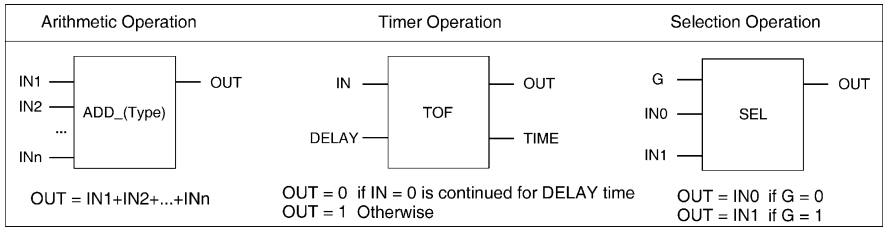


Fig. 4. IEC 61131-3 FBD samples for nuclear DPPS Software.

that requirements for BP and CP alone in NuSCR exceed 400 pages.[2] All four redundant copies of BPs and a CP are implemented on separate PLCs and executed periodically.

2.2. NuSCR

NuSCR improves expressiveness of the notation and readability of the SCR specification [6,7]. Ref. [4] describes details NuSCR syntax and formal semantics in detail, and Ref. [8] reports a NuSCR-based requirements engineering environment named NuEditor. In this section, we briefly review only the features necessary to understand how FBD code is synthesized from NuSCR.

Function overview diagram (FOD), in notation similar to the data-flow diagrams, captures dependency relation among various nodes hierarchically so that complex requirements can be specified in a divide-and-conquer fashion. FOD for *g_Fixed_Setpoint_Rising_Trip_with_OB* (Fig. 1) shows the overall structure for fixed set-point falling trip logic. It is a screen-dump of NuEditor. Rectangular nodes indicate inputs and outputs, and different types of nodes are drawn in different shape and color so that their roles are visually apparent.

Function variables use SDT as used in SCR. Fig. 2 describes how *f_X_Valid* computes the required output. NuSCR allows an arbitrarily complex composition Boolean expression whereas SCR notation requires it to be

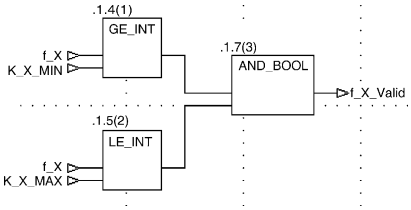


Fig. 5. FBD for *f_X_Valid*.

[2] Requirements written in natural language, during preliminary phase of requirements engineering is about 40 pages.

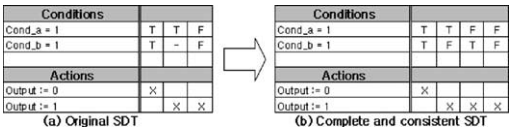


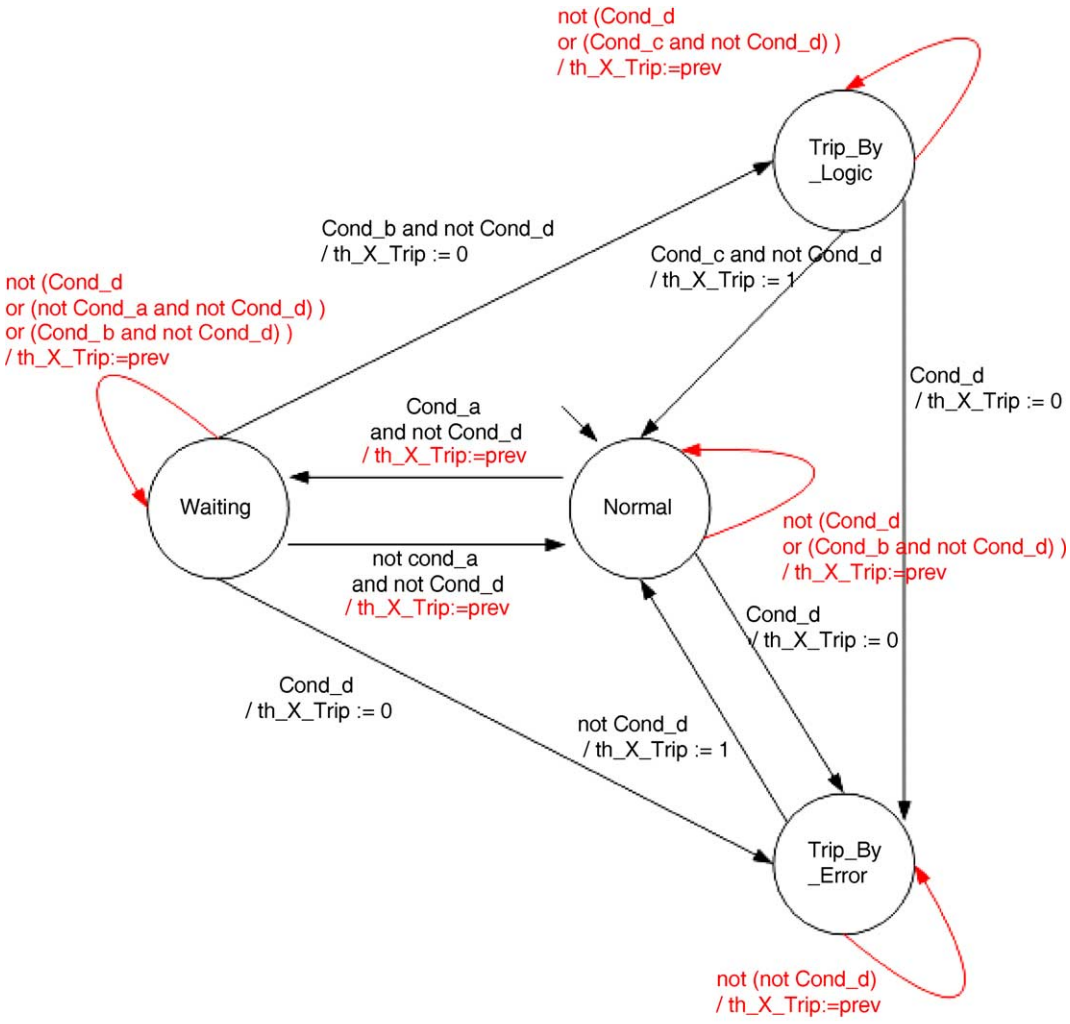
Fig. 6. Example of the completeness and consistency for SDT.

decomposed into primitive units (e.g. no Boolean operators). Example shown in Fig. 2 is too simple to convincingly demonstrate the difference in expressiveness of SDT used in NuSCR and SCR. However, domain experts have repeatedly emphasized that there is no practical advantage to gain by forcing fine-grained breakdown of complex mathematical equations that are well-understood as a coherent unit. Revised syntax of SDT is a critical factor contributing to conciseness and reviewability. For example, in Wolsung SDS2, written in SCR, some tables were so complex as to contain 16 rows and 12 columns, and requirements for KNICS RPS is significantly more complex than that of Wolsung SDS2.

History variables, specifying state-dependent behavior, use finite state machine notation whose labels represent triggering events and conditions. *Timed-history variables* are similar to history variables in notation, but time constraints (e.g. durations such as [1,5]) are associated with labels. Macros are also supported to prevent automata from becoming too crowded to be easily reviewed.

2.3. PLC programming in FBD

PLC has relatively simple architecture, compared to modern microprocessors, where sensors and actuators are plugged in via input and output channels, respectively. Simplified architecture and processing steps make PLC an attractive platform for implementing embedded application



Cond_a : f_X >= k_X_Trip_Setpoint
Cond_b : [k_Trip_Delay,k_Trip_Delay] (f_X >= k_X_Trip_Setpoint and h_X_OB_Sta = 0)
Cond_c : f_X < k_X_Trip_Setpoint - k_X_Trip_Hys
Cond_d : f_X_Valid = 1 or f_Module_Error = 1 or f_Channel_Error = 1)

Fig. 7. Modified complete and consistent timed automata for *th_X_Trip*.

SV = Normal	T	T	T									
Cond_a and not Cond_d	T	-	-									
cond_d	-	T	-									
Otherwise	-	-	T									
SV = Waiting				T	T	T	T					
not Cond_a and not Cond_d				T	-	-	-					
Cond_d				-	T	-	-					
Cond_b and not Cond_d				-	-	T	-					
Otherwise				-	-	-	T					
SV = Trip_By_Logic								T	T	T		
Cond_c and not Cond_d								T	-	-		
Cond_d								-	T	-		
Otherwise								-	-	T		
SV= Trip_By_Error											T	T
not Cond_d											T	-
Otherwise											-	T
Output := 0		X			X	X			X			
Output := 1								X			X	
Output := prev	X		X	X			X			X		X
SV := Normal(0)			X	X				X			X	
SV := Waiting(1)	X						X					
SV := Trip_By_Logic(2)						X				X		
SV := Trip_By_Error(3)		X			X				X			X

Fig. 8. 2C-table for *th_X_Trip*.

software. Operating system, managing periodic execution of PLC applications, reads all input values at the beginning of each cycle, generates required outputs, and stores system variables.

Among the five PLC programming languages included in the IEC 61131-3 standard [9], KNICS project chose to implement software in FBD. FBD, similar to electrical circuit diagram in notation, consists of a network of primitive function blocks. Fig. 4 includes some of the representative samples of function blocks performing logical, arithmetic, selection, and timing operations.

For example, SDT for *f_X_Valid*, shown in Fig. 2, can be easily translated into FBD shown in Fig. 5.[3] Numbers appearing in parenthesis appearing above the function blocks indicate the generation and execution sequences. That is, AND_BOOL computation is started when GE_INT and LE_INT function blocks produced their results.

3. FBD synthesis

This section describes a process of generating FBD code from NuSCR specification. Consistency and completeness of the tables and automata are first analyzed. Automata are then converted into an intermediate tabular notation which we call 2C-table. Basic FBD corresponding to each table notation is generated, and the final step determines an execution order of all the FBD nodes.

Step 1. *Consistency and completeness checking*. NuSCR allows inclusion of arbitrarily complex expressions and macros in SDT and automata. When specifying state- and time-dependent behavior using (timed) automata notation, not all edges are drawn explicitly. In addition, SDT may contain nondeterminism if specific ordering of execution sequences does not matter. While such features reduce complexity of requirements, all the missing details and exceptional situations must be made explicit, complete and consistent. For example, SDT shown in Fig. 6(a) is neither complete nor consistent. As '-' denotes 'don't care' condition, output can be either 1 or 0 if both conditions *Cond_a* and *Cond_b* are TRUE. On the other hand, output is unspecified when *Cond_a* is FALSE and *Cond_b* is TRUE. As there is no completely automated process through which one can transform tables to be complete, consistent, and semantically correct, user intervention is required. Similarly, state- and time-dependent requirements must also be verified, in isolation, for completeness and consistency. See Fig. 7 for an example where automata shown in Fig. 3 are made complete and consistent.

Step 2. *2C-table generation for FSM and TTS*. Requirements captured in automata are converted into an intermediate tabular notation called the 2C-table. While its format is similar to that of SDT, 2C-table has an additional action part capturing changes made to state variables. Fig. 8 is the 2C-table obtained from modified automata shown in Fig. 7. *SV* is the state variable, and *Output* denotes the output of *th_X_Trip* node. For example, the second column

[3] FBD shown in Fig. 5 is developed using Concept version 2.2 XL SR2, a PLC programming assistant tool marketed by Schneider Automation GmbH.

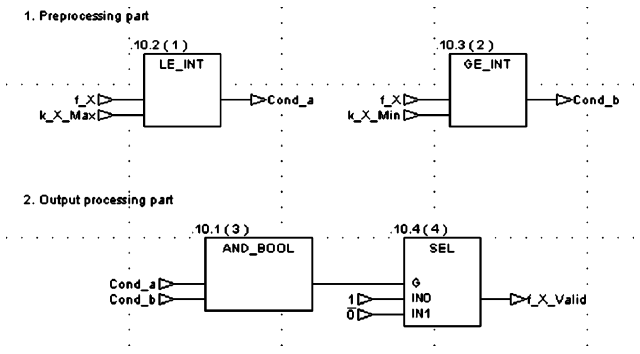


Fig. 9. FBD generated from SDT of *f_X_Valid*.

of Fig. 8, shaded for the purpose of illustration, denotes that if *Cond_a* and *not Cond_d* are satisfied in state *Normal*, the output value of *th_X_Trip* is the same as the previous one and the next state is *Waiting*.

Step 3. *Basic FBD generation*. The next step separately generates basic FBD from each SDT and 2C-table. Reflecting the characteristics of PLC, where input values are first read before output values are computed, FBD generated from SDT and 2C-table consists of two parts: (1) preprocessing routine in which conditions and macros used in the SDTs are first evaluated; and (2) computation routine where output values are determined. See Fig. 9, generated from SDT shown in Fig. 2, for an example. Complex conditions are internally decomposed into a collection of primitive predicates, and Boolean operators are replaced by the corresponding FBD blocks. Ref. [10] describes the details of the algorithm. 2C-table is transformed, as shown in Figs. 10–12, into FBDs using the same procedure. SEL block allows one of several inputs be chosen as outputs and updated value of state variables.

Step 4. *FBDs execution order decision*. In the final step, FOD is analyzed, and execution order of each FBD is decided. Analysis of FOD shown in Fig. 1 reveals the following dependencies among nodes: $1 \rightarrow 5$ and $2 \rightarrow 3 \rightarrow 5$. Therefore, three interleaving sequences are possible when all the nodes except the node 4 are included, and 15 possibilities exist when all the nodes are included. Because all such cases are semantically equivalent, our procedure selects one of them in a nondeterministic manner.

While the procedure describe above is straightforward in concept and algorithm, it is not optimal in the number of FBD blocks used because FBDs computing outputs and updating state variables are separately but redundantly implemented (Table 1). Such redundancy is acceptable in nuclear applications where safety assurance is much more important than optimization of FBD blocks. As each FBD block is inexpensive and RPS is not mass produced in large quantity, as is the case with automotive control PLCs, optimization of the FBD is not the most important concern. In fact, experimental studies

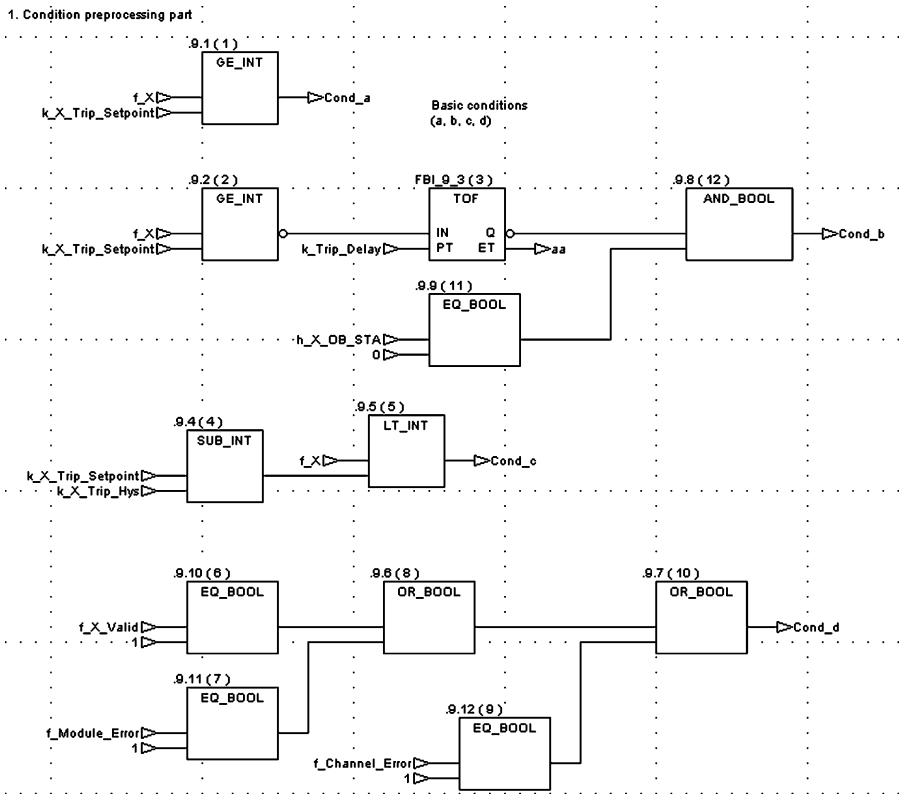


Fig. 10. Preprocessing part FBD for *th_X_Trip*.

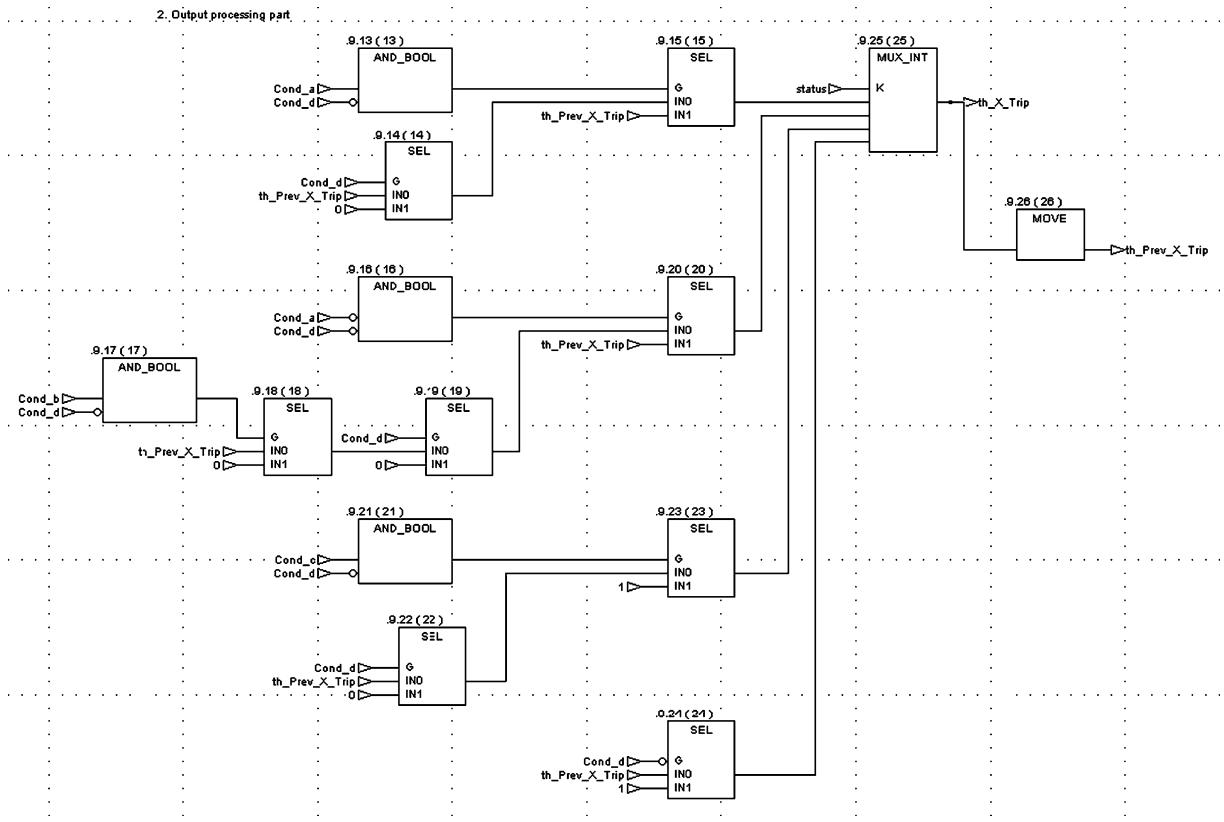


Fig. 11. Output processing part FBD for *th_X_Trip*.

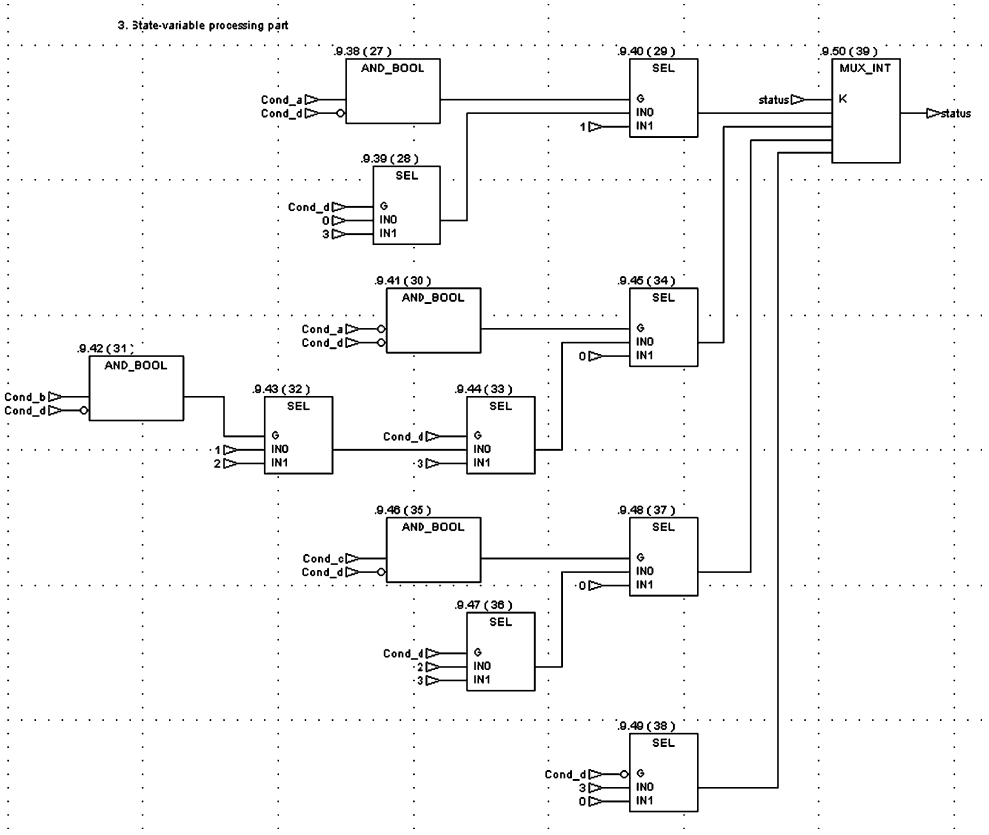


Fig. 12. State-variable processing part FBD for *th_X_Trip*.

Table 1
Comparison of the number of FBD blocks included in the fixed set-point rising trip logic

	f_X_Valid	*th_X_Trip*	*th_X_Pretrip*	*F_X_OB_Perm*	*h_X_OB_Sta*	Total
System atically generated from NuSCR	3	39	16	2	11	71
Manually generated by experts	3	12	8	9		32

Number of function blocks used.

comparing synthesized FBDs against manually developed and optimized FBDs by domain experts revealed that there is about 2:1 ratio in the number of FBD blocks required. Table 2 compares the number of necessary FBD blocks in implementing three representative trip logics for KNICS RPS BP: *fixed set point trip*, *auto-limited rate variable set point trip*, and *manual reset variable set point trip*.

4. Conclusion and future work

In this paper, we proposed a systematic procedure of generating FBD-based PLC code from NuSCR specification. While the procedure is not fully automated in that user interactions are needed when filling in implicit information or making sure that SDTs are complete and consistent, the rest is automated. While the transformation algorithm is not yet fully refined to generate optimized FBD code, the case studies applied to nuclear instrumentation and control applications found the approach effective in several ways. First, correctness of the synthesis procedure is easy to validate, and straightforwardness of the algorithm carries an important advantage when demonstrating software safety. Second, in nuclear applications, we found that synthesized FBD code was efficient enough in meeting the deadlines. Whereas application required cycle times in the range of 30–50 ms, synthesized FBDs completed its computation in less than 20 ms.

While FBD code can be automatically generated, domain engineers are still most likely to modify or manually optimize synthesized FBD code. In order to provide adequate support when developing safety-critical software, one must provide techniques where two different FBD designs are proven to be semantically equivalent. We are working on developing techniques to accomplish formal verification of FBD equivalence.

Table 2
Comparison of the number of function blocks used for the representative trip logics in BP

Trip logic for BP	Mechanically generated from NuSCR	Manually generated by experts
Fixed set-point rising trip with operating bypass	71	32
Fixed set-point rising trip without operating bypass	53	24
Auto-limited rate variable set point trip without operating bypass	95	40
Manual reset variable set point trip with operating bypass	117	67
Total	336 2.06:1	163

Number of function blocks used.

References

[1] SIEMENS, TELEPERM XP/XS, http://www.powergeneration.siemens.com/en/processcontrol/index.cfm.
[2] Yoo J, Cha S, Kim CH, Oh Y. Formal software requirements specification for digital reactor protection systems. J KISS: Software and Application June 2004;31(6):750–9.
[3] KNICS. Korea Nuclear Instrumentation and Control System Research and Development Center. http://www.knics.re.kr/english/eindex.html.
[4] Yoo J, Kim T, Cha S, Lee J-S, Son HS. A formal software requirements specification method for digital nuclear plants protection systems. J Syst and Software 2003 in press.
[5] Parnas D, Madey J. Functional documentation for computer systems engineering (version 2). CRT 237. Hamilton, Ont.: Telecommunications Research Institute of Ontario (TRIO), McMaster University; 1991.
[6] Heninger KL. Specifying software requirements for complex systems: new techniques and their application. IEEE Trans Software Eng 1980; SE-6(1):2–13.
[7] Schouwen Van AJ, Parnas D, Madey J. Documentation of requirements for computer systems In: RE'93: IEEE International Symposium On Requirements Engineering 1993 pp. 198–207.
[8] Cho J, Yoo J, Cha S. NuEditor—a tool suite for specification and verification of NuSCR In: Second ACIS International Conference on Software Engineering Research, Management and Applications (SERA 2004) 2004. LA, USA, May 5–7, pp. 298–304.
[9] IEC (International Electrotechnical Commission). International standard for programmable controllers: programming languages 61131-3, 1993.
[10] Yoo J, Bang HJ, Cha S. Procedural transformation from formal software requirement to PLC-based design. Technical Report CS/TR 2004-198. Korea Advanced Institute of Science and Technology (KAIST), 373-1, Kusong-dong, Yusong-gu, Taejon, Korea, 2004.