PLC-Based Safety Critical Software Development for Nuclear Power Plants

Junbeom Yoo[1], Sungdeok Cha[1],
Han Seong Son[2], Chang Hwoi Kim[2], and Jang-Soo Lee[2]

[1] Korea Advanced Institute of Science and Technology(KAIST) and AITrc/SPIC/IIRTRC
Department of Electrical Engineering and Computer Science,
373-1, Kusong-dong, Yusong-gu, Taejon, Korea,
{jbyoo,cha}@salmosa.kaist.ac.kr
[2] Korea Atomic Energy Research Institute(KAERI) MMIS team,
150, Deokjin-dong, Yusong-gu, Taejon, Korea,
{hsson,chkim2,jslee}@kaeri.re.kr

Abstract. This paper proposes a PLC(Programmable Logic Controller)-based safety critical software development technique for nuclear power plants' I&C software controllers. To improve software safety, we write the software requirements specification using a formal specification notation named NuSCR [1]. NuSCR specification is then mechanically transformed into semantically equivalent Function Block Diagram(FBD), a widely used PLC programming language. Finally, we manually refine the FBD programs so that redundant function blocks are identified and removed. As CASE tool supplied by PLC vendors automatically compiles the resulting FBD programs into PLC machine code, PLC software development is completed when the final FBD programs are essentially tested.
Proposed development technique offers several advantages. Requirement errors are reduced as we use the formal specification notation. Consistency and completeness checks are automated, and model checking can be performed on the NuSCR specification. Safety critical errors are less likely to be introduced to the synthesized FBD programming. As a consequence, cost of developing and validating the PLC-based software can be also reduced. The proposed approach is currently being applied in developing safety-critical control software for a Korean nuclear power plant, and experience to date has been positive.

1 Introduction

PLC [2] is widely used in industry to implement real-time safety critical software [3]. Such trend is especially true in the area of nuclear power plant's I&C(Instrumentation and Control) systems as aged RLL(Relay Ladder Logic)-based analog systems are being replaced by PLC-based digital systems [4].

Software development process for control software in nuclear power plants generally consists of analysis, design, and implementation phases. Software requirements are initially written in natural language, and a formal specification is developed on which various formal analysis techniques are applied. As embedded software controlling nuclear power plants usually run on PLCs, PLC programs, written in Ladder Diagram(LD) or Function Block Diagrams(FBDs) [2], are developed and documented in software design specification(SDS). In implementation phase, the hardware configuration for PLC, i.e. number of I/O cards, CPU speed, network communication, is decided, and then PLC programs are translated into machine code. This translation process is conducted automatically by an engineering tool, which is provided by PLC vendors. Therefore, the actual software development for PLC software is completed at the end of the design phase.

If software requirements specification is written in a natural language, much effort would be needed to certify its safety using inspection, simulation, and other safety analysis techniques. Likewise, manual programming of PLC programs is inefficient and a potentially error-prone task.

In this paper, we propose a PLC-based software development method. It consists of three phases: formal requirements specification, synthesis, and refinement phase. It uses the formal software requirements specification language, NuSCR [1], to express and analyze the software requirements. Formal software requirements specification allows developers to specify all requirements explicitly and completely while avoiding inconsistency in logic. Mechanical and formal verification methods, such as model checking [5] and mechanized theorem proving [6], can also be applied to NuSCR formal requirements.

In synthesis phase, we mechanically transform the NuSCR formal specification into FBD program using an intermediate notation called the *2C-Table*. Comparison of synthesized FBD against the manually developed FBDs revealed that experts could reduced the number of required FBD blocks by up to 50%. While the scan cycle of PLC is between 30 - 50ms, the total execution time of the manually programmed FBD program operating on PLC is usually at most 5 - 10ms. Therefore, the two times increase of the number of function blocks does not seriously affect timely execution of PLC code.

While FBD code can be automatically generated, domain engineers are still most likely to modify or manually optimize synthesized FBD code. In refinement phase, we are working on formal method support so that the semantic equivalence between the two FBD codes can be verified.

The remainder of the paper is organized as follows: Section 2 briefly introduces the PLC. In Section 3, we explain the proposed PLC based safety critical software development method. To aid the understanding we use the real case study, which is presently being developed in Korea. Conclusion and future work are in Section 4.

2 Programmable Logic Controller

Programmable Logic Controller(PLC), which is widely used in the real-time and safety-critical systems industry, has features as follows [7]:

Concise Hardware Architecture. PLC has a relatively simple hardware architecture that facilitate the input/output configurations. This architectural characteristics makes the exact analysis of the program execution time possible. Operating system for PLC is provided by PLC vendors, and the application programs operating on the real-time OS are programmed with SFC, LD, or FBD using CASE tool supplied by PLC vendors.

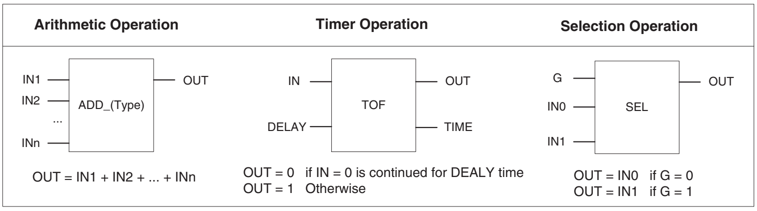


Fig. 1. IEC 61131-3 FBD samples

Program Execution Mechanism. PLC programs are executed in a permanent loop. In each iteration of loop, a scan cycle, inputs are read. The program computes a new internal state and output, and the outputs are updated. There exists an upper time bound for each cycle, which typically is in the order of milli-seconds.

Programming Languages. IEC 61131-3 standard include five PLC programming languages: ST(Structured Text), LD, IL(Instruction List), SFC, and FBD. In practice, FBD and LD are most widely used. FBD, similar to electrical circuit diagram in appearance, consists of a network of function blocks and regards the system as the flow of information expressed with primitive function blocks. (See ⟨Fig.1⟩ for samples of widely used basic function blocks.) Basic FBs can be classified into logical, arithmetic, selection, and timing operations, and it is known that the RPS(reactor protection system), which is presently being developed in Korea, can be programmed using 14 different types of FBD blocks belonging to the four groups mentioned above.

3 PLC-Based Software Development

An overview of the PLC-based software development process, starting with NuSCR formal specification and ending with validation of refined FBD code, is shown in ⟨Fig.2⟩.

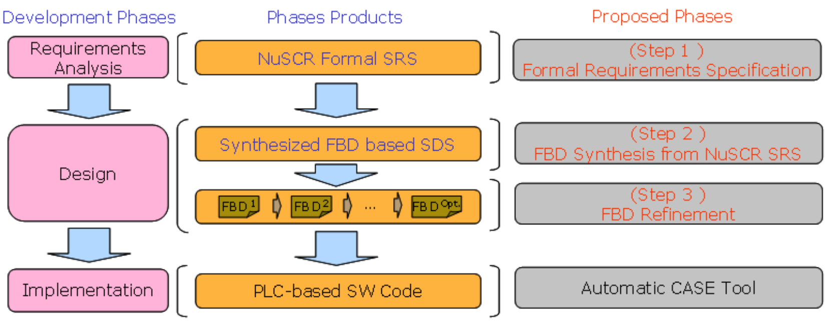


Fig. 2. Overview of formal method guided PLC based software development

3.1 Phase I: Formal Requirements Specification

NuSCR [1] was developed with active participation of and consultation by nuclear engineers who are familiar with software engineering knowledge in general and formal methods in particular. Readability of the specification to domain experts was a key concern when deciding which notation to use to capture various aspects of requirements.

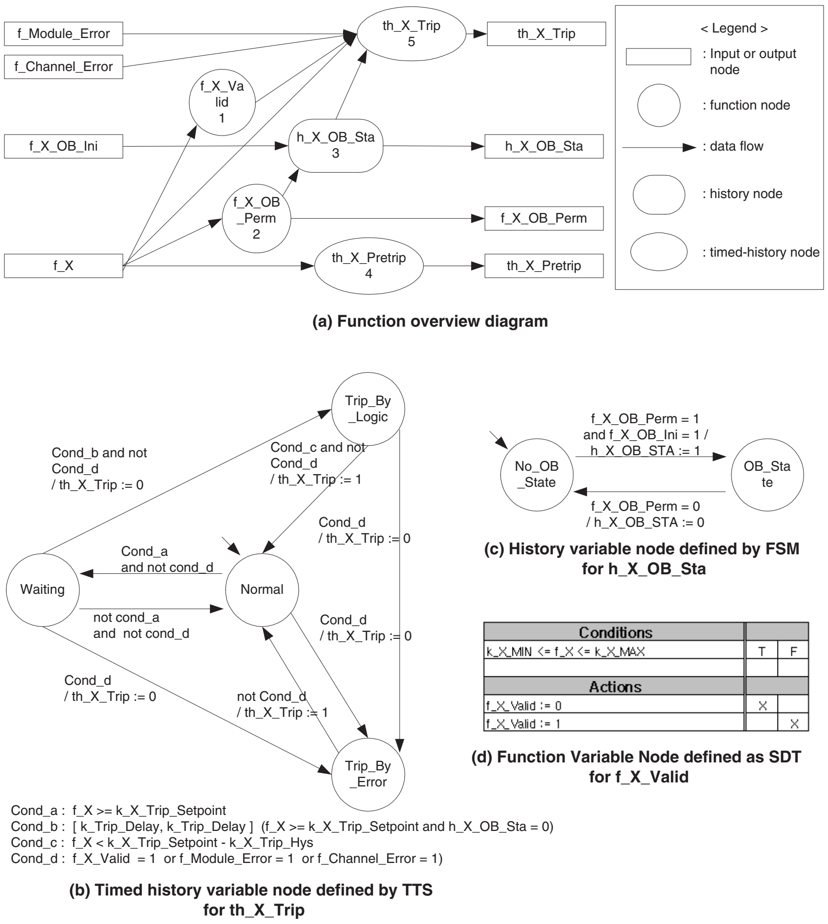
It uses FOD(Function Overview Diagram) for the overview of data flows. In addition, it introduces three basic constructs,*function variable*, *history variable*, and *timed history variable*. These constructs are written in SDT(Structured Decision Table), FSM(Finite State Machine), and TTS(Timed Transition System) [8] notations respectively. *Function variables* specify mathematical functional behavior of system, and they are defined as SDT, which is a condtion/event table. *History variables* describe state-based behavior in finite state machine where transitions capture triggering events or conditions as well as generated actions. *Timed-history variables* express timing constraints in extended FSM notations.

$\langle$Fig.3$\rangle$ describes the basic constructs of NuSCR. $\langle$Fig.3 (a)$\rangle$ is a FOD for *g_Fixed_Setpoint_Rising_Trip_with_OB* logic for fixed set-point rising trip in RPS(Reactor Protection System) BP(Bistable Logic), which is currently being developed at KNICS in Korea. *_g* means that it is a group of node in the FOD hierarchy and that details are further captured in a separate FOD diagram. It is composed of five internal nodes, and they are all defined individually. The prefixes *"f_"*, *"h_"*, and *"th_"* denote function variable nodes, history variable nodes, and timed history variable nodes, respectively. Arrows denote data-flow dependency relation.

$\langle$Fig.3 (b)$\rangle$ is an TTS definition for timed history variable node *th_X_Trip* appearing in the FOD. TTS is a FSM extended with time duration constraint $[a, b]$ in transition conditions. TTS defines the time-related behavior of nuclear power plants control systems. The TTS definition for *th_X_Trip* is interpreted as follows: "If condition *f_X* $\geq$ *k_X_Trip_Setpoint* is satisfied in state *Normal*, it transits to *Waiting* state. In this state, if the condition is lasted for *k_Trip_Delay* then it fires the trip signal 0. If *f_X_Valid*, *f_Module_Error*, or *f_Channel_Error* occur, then trip signal is fired at once. In the state *Trip_By_Error* or *Trip_By_Logic*, if the trip conditions are canceled, then it comes back to the state *Normal* and the output is 1." The TTS expression in *Cond_b*, *[k_Trip_Delay,k_Trip_Delay]* means that the condition has to remain true for *k_Trip_Delay* time units.

$\langle$Fig.3 (c)$\rangle$ is an FSM definition for history variable node *h_X_OB_Sta* in the FOD. It is interpreted as follows: " In initial state *No_OB_State*, if condition*f_X_Perm = 1 and f_X_OB_Ini = 1*is satisfied , it transits to *OB_State* with setting *h_X_OB_STA* is 1. In state *OB_State*, if condition*f_X_Perm = 0* is satisfied, then it transits back to *No_OB_State* with setting *h_X_OB_STA* is 0 again.

$\langle$Fig.3 (d)$\rangle$ is an SDT definition for function variable node*f_X_Valid*. It is interpreted as follows: "If the value of *f_X* is between *k_X_MIN* and *k_X_MAX*, the output value *f_X_Valid* is 0, which means it is a normal case. Otherwise output value of *f_X_Valid* is 1." NuSCR recommends multiple correlated condition statements per row. In this way, NuSCR can resolve a large part of the table-size explosion problems, and also can increase the readability of SDTs [9].


f_Module_Error
f_Channel_Error
f_X_OB_Ini
f_X
f_X_Va lid 1
f_X_OB _Perm 2
h_X_OB_Sta 3
th_X_Pretrip 4
th_X_Trip 5
th_X_Trip
h_X_OB_Sta
f_X_OB_Perm
th_X_Pretrip
< Legend >
: Input or output node
: function node
: data flow
: history node
: timed-history node
(a) Function overview diagram
Trip_By _Logic
Waiting
Normal
Trip_By _Error
Cond_b and not Cond_d / th_X_Trip := 0
Cond_c and not Cond_d / th_X_Trip := 1
Cond_d / th_X_Trip := 0
Cond_a and not cond_d
not cond_a and not cond_d
Cond_d / th_X_Trip := 0
Cond_d / th_X_Trip := 0
not Cond_d / th_X_Trip := 1
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)
(b) Timed history variable node defined by TTS for th_X_Trip
No_OB _State
OB_Sta te
f_X_OB_Perm = 1 and f_X_OB_Ini = 1 / h_X_OB_STA := 1
f_X_OB_Perm = 0 / h_X_OB_STA := 0
(c) History variable node defined by FSM for h_X_OB_Sta
(d) Function Variable Node defined as SDT for f_X_Valid

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

Fig. 3. Basic constructs of NuSCR

3.2 Phase II: FBD Synthesis from NuSCR SRS

In this phase, we derive PLC-based FBD program from the requirements specification written in NuSCR. See [10,11] for detailed discussion on the formal definitions of rules, algorithms, and procedures. FBD generation process has 4 steps. First we perform consistency and completeness analysis of selected nodes, which are defined as SDT, FSM, or TTS. After modifying all nodes to be complete and consistent, we produce 2C-Table for corresponding FSM and TTS. 2C-Table is an intermediate notation that are used to facilitate the FBD generation process for FSM and TTS. In the next step, FBDs are generated from SDTs of 2C-Tables. After generating the individual FBDs, we analyze the dependency among nodes in FOD and decide appropriate execution orders for all nodes in the FOD. As PLC executes its application programs sequentially, proper selection of execution order is essential. Details of each step, along with an example, is explained below:

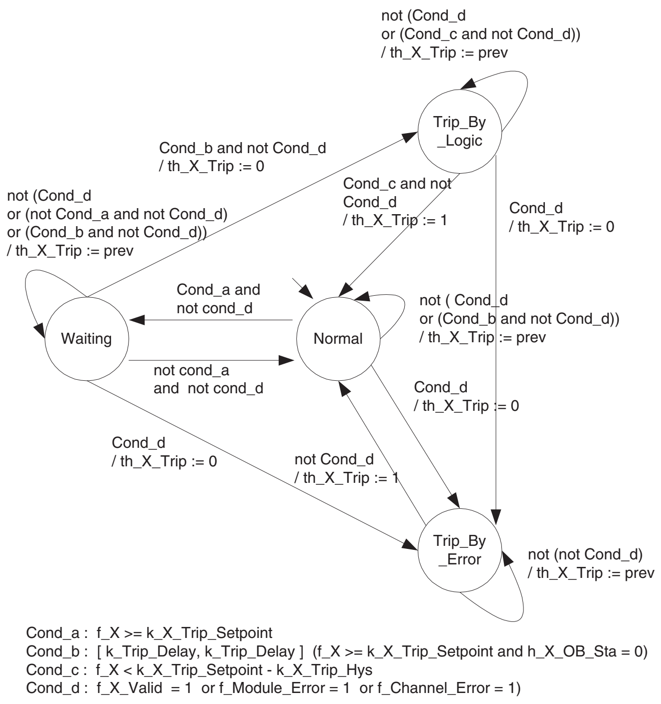
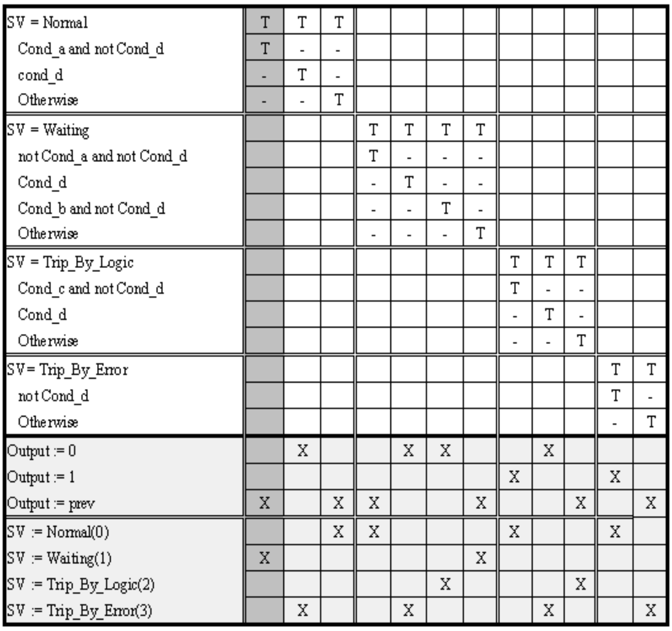


Fig. 4. Modified complete and consistent TTS for *th_X_Trip*

(Step 1) Completeness and Consistency Analysis. NuSCR allows inclusion of arbitrarily complex expressions and macros in SDT and automata such as FSM and TTS. 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, ⟨Fig.4⟩ illustrates the results of performing consistency and completeness analysis applied on ⟨Fig.3 (b)⟩. FSM, which defines the history variable node, can be modified in the same way so that automata remains in the current state if no conditions initiating a transition to other states are satisfied. In this case, we need to specify the implicit transitions explicitly . If there is no action statement in the transition label in FSM and TTS, then NuSCR regards that the output is set to the same value in the previous scan cycle (i.e. *th_X_Trip := prev*).

(Step 2) 2C-Table Generation. While format is similar to that of SDT, 2C-Table has an additional action part capturing changes made to state variables. ⟨Fig.5⟩ is the 2C-Table obtained from modified automata shown in ⟨Fig.4⟩. *SV* is the state variable, and Output denotes the output of *th_X_Trip* node. For example, the second column of ⟨Fig.5⟩, shaded for the purpose of illustration, denotes that if condition "*Cond_a and not Cond_d*"



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. 5. 2C-Table for TTS of *th_X_Trip*

is 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 and state values are determined. ⟨Fig.6⟩ is a generated from the SDT shown in ⟨Fig.3 (c)⟩. Complex conditions are internally decomposed into a collection of primitive predicates, and Boolean operators are replaced by the corresponding FBD blocks. [11] describes the details of the algorithm. The FBD is made from *Concept version 2.2 XL SR2*, a PLC programming assistant tool by *Schneider Automation GmbH* [12], and the numbers above the function blocks indicate the generation and execution orders. 2C-Table is also transformed, as shown in ⟨Fig.7 ⟩, into FBDs using the same procedure. SEL and MUX function block allows one of several inputs to be chosen as outputs and updated value of state variables.

⟨Fig. 7 (b)⟩ is the whole output processing part FBD for *th_X_Trip*. The selection among many condition statements and their corresponding action statements in 2C-Table or SDT are implemented in FBD with SEL function block. Each condition and output statements are preprocessed previously, and the results are used in the output calculation. The current state of automata node(FSM and TTS), *Status*, provides the basis of the

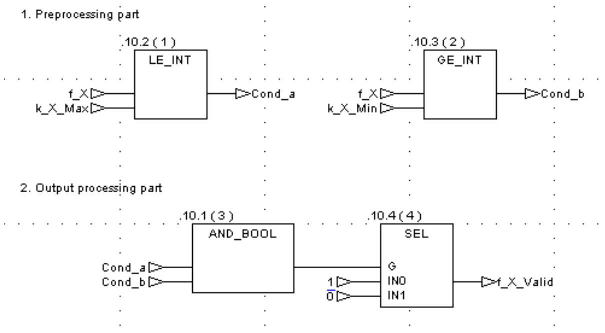


Fig. 6. FBD generated from SDT of *f_X_Valid*

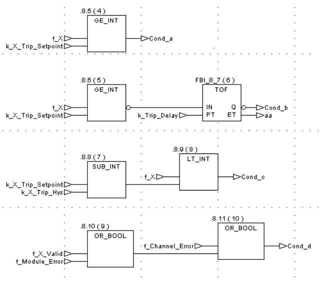
decision about which one in SEL function blocks we have to select as an output, and this selection process is implemented using MUX function block with *status*. MUX function block generates a selected output according to *status*, and the variable *status* is implemented as ⟨Fig.7 (c)⟩ from the lower part of 2C-Table in ⟨Fig.5⟩. Variable *th_Prev_X_Trip*, which is the other output in the FBD, is the internal variable used in only the FBD. After these processes are finished, we get the individual FBDs for all nodes, s.t. function variable nodes, history variable nodes, and timed-history variable nodes, in FOD.

(Step 4) FBD Execution Order Decison. In the final step, the execution order for each FBDs in FOD is analyzed and then decided. The possible execution orders for the 5 nodes composed in FBD in ⟨Fig. 3 (a)⟩ is as follows: At first, there are two partial orders among the 5 nodes in FOD as their input/output relationships. As the node numbered 4 has no interaction with other nodes, it is considered independent of others.

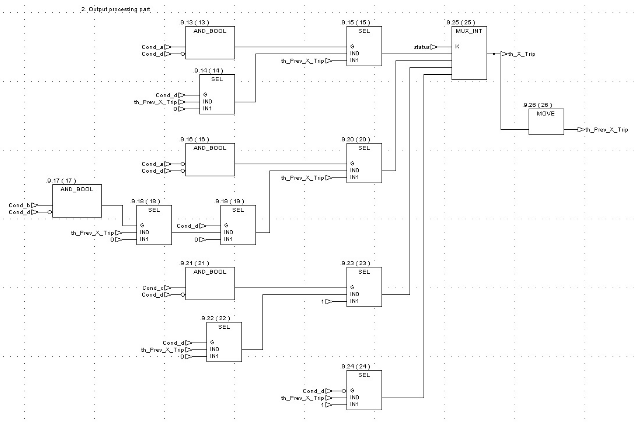
Partial execution order 1: $(1 \longrightarrow 5)$
Partial execution order 2: $(2 \longrightarrow 3 \longrightarrow 5)$
Independent execution: (4)

All possible combinations of execution orders are shown below, and one is picked nondeterministically as all are equivalent.

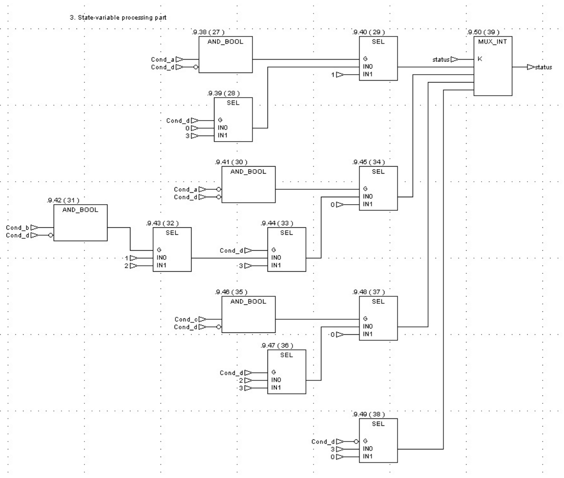
Execution order 1:
(Input) $\rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow (4) \rightarrow$(Output)
Execution order 2:
(Input) $\rightarrow 2 \rightarrow 1 \rightarrow 3 \rightarrow 5 \rightarrow (4) \rightarrow$(Output)
Execution order 3:
(Input) $\rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 5 \rightarrow (4) \rightarrow$(Output)



(a) Preprocessing part FBD



(b) Output processing part FBD



(c) State-variable processing part FBD

Fig. 7. FBD generated from TTS of *th_X_Trip*

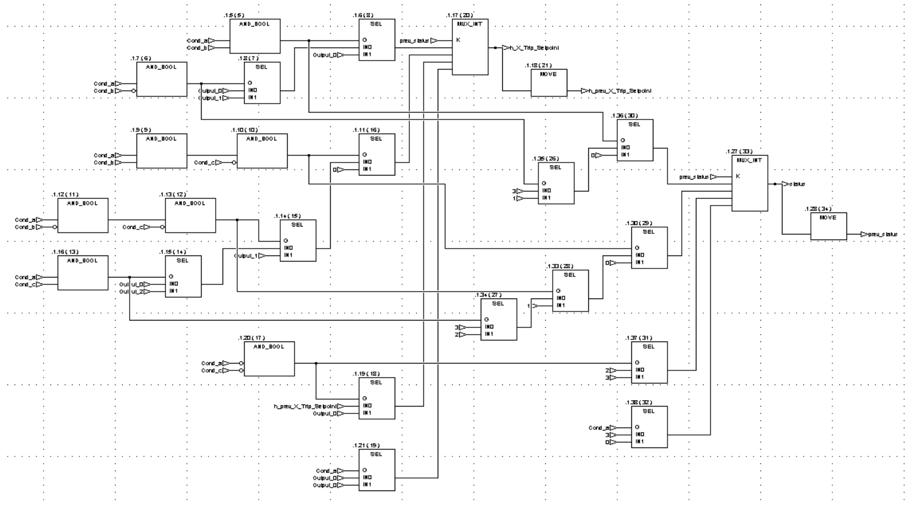


Fig. 8. An example of refined FBD from Fig.7(b) and (c)

3.3 Phase III: FBD Refinement

Finally in refinement phase, we allow domain experts to refine the generated FBD program. Our case study revealed that manually prepared FBD may contain fewer number of FBD blocks, and execution time takes less. Therefore, to increase chance that execution of PLC code satisfies the timing requirements, additional modification processes are provided to further reduce the size of generated FBD program. ⟨Fig.8⟩ is an example of refined FBD from the one in ⟨Fig.7 (b),(c)⟩.

However, the revised FBD code must be tested to demonstrate that it is still the same in its behavior. We are currently focusing on sequential equivalence [13] of two FBD programs to verify their behavioral equivalence. Some formal and automated techniques, such as VIS model checker [14] or COSPAN/FormalCheck [15], may be used to verify their behavioral equivalence.

4 Conclusion and Future Work

In this paper, we proposed a PLC based safety critical software development process in which formal methods played critical roles. It improves safety of embedded software and reduces time needed to develop safety-critical software.

We have been applying our proposed method successfully to develop the plant protection system to be deployed in a nuclear power plant in Korea. Experience by domain experts on the proposed approach has been positive, and we are currently working on development of analysis methods used to check the behavioral equivalence of subsequently modified FBDs. This analysis method is expected to accelerate the whole appliance of our proposed transformation-based PLC software development method.

Acknowledgements. This research was partially supported by Advanced Information Technology Research Center(AITrc), Software Process Improvement Center(SPIC), and Internet Intrusion Response Technology Research Center(IIRTRC) in Korea.

References

1. Yoo, J., Cha, S., Son, H.S., Kim, C.H., Lee, J.S.: A formal software requirements specification method for digital nuclear plants protection systems. Journal of Systems and Software to be published (2003)
2. Commission, I.E.: International standard for programmable controllers: Programming languages (1993) part 3.
3. Leveson, N.G.: SAFEWARE, System safety and Computers. Addison Wesley (1995)
4. NRC, U.: Digital Instrumentation and Control Systems in Nuclear Power Plants: safety and reliability issues. National Academy Press (1997)
5. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Programming Languages and Sysems **8** (1986) 244–263
6. Dalen, D.V. In: Logic and Structure. 3 edn. Springer-Verlag (1994)
7. Mader, A.: A classification of plc models and applications. In: Discrete Event Systems-Analysis and Control: WODES 2000. (2000)
8. Henzinger, T.A., Manna, Z., Pnueli, A.: Timed transition systems. In: REX Workshop. (1991) 226–251
9. Yoo, J., Cha, S., Kim, C., Oh, Y.: Formal software requirements specification for digital reactor protection systems. Journal of KISS(Korea Information and Science Society) to be published (2004)
10. Yoo, J., Cha, S., Kim, C., Song, D.Y.: Synthesis of FBD-based PLC design from NuSCR formal specification. Reliability Engineering and System Safety to be published (2004)
11. Yoo, J., Bang, H., 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)
12. Electric, S.: (http://www.modicon.com/)
13. Huang, S.Y., Cheng, K.T.: 4. In: Fromal Equivalence Checking and Debugging. Kliwer Academic Publishers (1998)
14. Brayton, R.K., Hachtel, G.D., Sangiovanni-Vincentelli, A., Somenzi, F., Aziz, A., Cheng, S.T., Edwards, S., Khatri, S., Kukimoto, Y., Pardo, A., Qadeer, S., Ranjan, R.K., Sarwary, S., Shiple, T.R., Swamy, G., Villa, T.: ((vis)
15. Kurshan, R.P.: Computer Aided Vrification of Coordinating Processes: The Automata-Theoretic Approach. Princeton University Press (1994)