Formal Verification of FBD-Based PLC Software at Software Design Phase

Han Seong Son, Kee-Choon Kwon, Junbeom Yoo*, Sungdeok Cha*

Korea Atomic Energy Research Institute
150 Deokjin-dong, Yuseong-gu, Daejeon, 305-353, Korea
Email: hsson, kckwon@kaeri.re.kr
**Korea Advanced Institute of Science and Technology and AITrc/SPIC*
373-1 Kusong-dong, Yusong-gu, Daejeon, 305-701, Korea
Email: jbyoo, cha@salmosa.kaist.ac.kr

ABSTRACT

This article suggests an approach to formal verification of FBD-based PLC software at design phase. The approach involves model checking and equivalence check which is a function some of model checkers offer. In this approach, a part of software requirements is translated into FBD in a systematic way and SDS includes FBD, which are designed by a software designer. Compliance between SRS and SDS is checked through model checking and equivalence check on the FBDs.

INTRODUCTION

As Programmable Logic Controllers (PLC) have been increasingly being used for safety critical systems, such as nuclear power plant control systems, there have been many research projects in the field of formal verification of PLC software[1]. Most of the researches, however, did not deal with Function Block Diagrams (FBD), which is one of the PLC languages defined in the IEC 61131-3 standard [2]. This article addresses a formal verification of FBD using a model checker, Verification Interacting with Synthesis (VIS) [3]. VIS is a tool that integrates the verification, simulation, and synthesis of finite-state systems. It uses a Verilog front end and supports Computational Tree Logic (CTL) model checking. The Verilog front end makes it easy to perform formal verification of FBD disregarding its lack of formality. Son and Kwon demonstrated that VIS is useful for model checking on FBD in their research [4].

Yoo, et al. proposed a method translating a formal software requirements specification into FBD [5]. Assuming that FBD are generally used in the design of PLC based software system, the method may be very useful for the verification of the design in view of correctness, consistency, and completeness as well as the design activity itself. This is because we can design and/or verify a system using the same terms and languages by virtue of the method. Extending the method and utilizing the equivalence checking feature of VIS, we propose a formal design verification method, with which makes it easy and systematic to verify consistency between software requirements and software design. Figure 1 depicts the verification approach of the proposed method.

TRANSLATING SRS TO FBD

In the area of nuclear power plant control systems, the software safety becomes more important with the replacement of existing analog systems, which is based on RLL, by digital systems composed of software process controllers. Therefore, formal software requirements specification methods [6, 7] are required to preserve the safety of such systems in the early phase of the software development process. Also software requirements and design specifications, which are suitable for the characteristics of nuclear power plants systems, are becoming a new research issue by many researchers. NuSCR [8] is a formal specification method specialized for this purpose, and it is being used to formally specify the software requirements specification of DPPS RPS, which is presently being developed at KNICS in Korea. Figure 2 illustrates the constructs of NuSCR, which are FOD in (a), which shows the overall dataflow and their inter-relationships, and other three variable nodes in (b),(c),(d), which define all variable nodes in FOD. NuSCR improves the readability and specifying ability by supplying different notations on the basis of their typical operation categories, such that state-dependent, timing constrained, and functional operations.

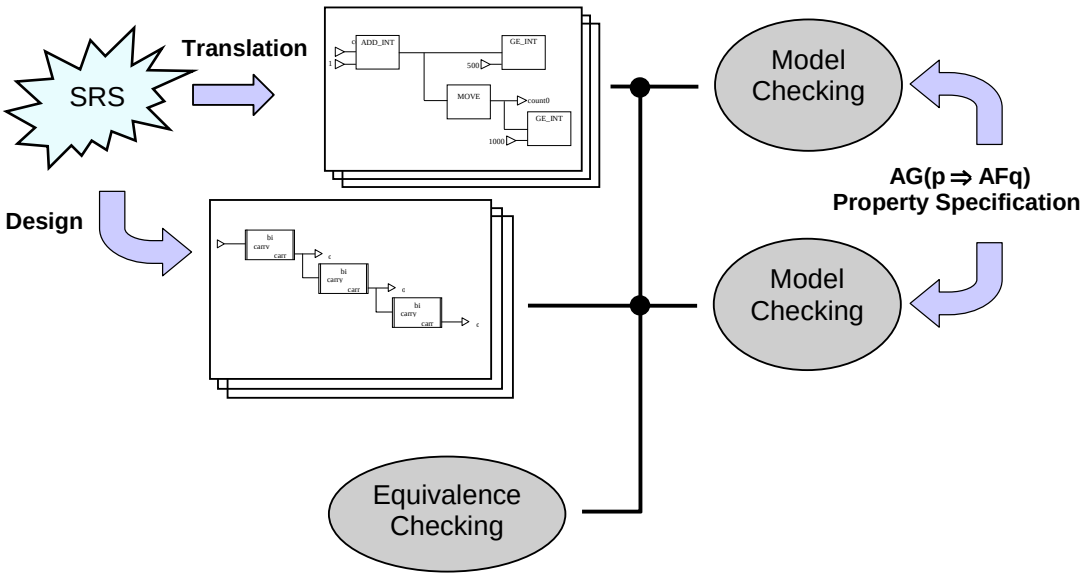


Figure 1. Schematic Diagram of the Proposed Method

Software in nuclear power plant control systems is embedded software that is implemented on PLC. Therefore, in the design phase, we specify software design requirements with PLC languages, such that FBD, which PLC can interpret and compile with. The developer usually writes the PLC programs manually from the requirements specification, as the requirements specification is written in a natural language. On the other hand, if a formal software specification method, s. t. NuSCR, is used, it has a formal syntax and semantics unlike natural languages, so it allows to generate PLC programs form the formal requirements specification, which has a same behavioral aspect with the requirements specification. The method [5] allows the systematic generation of PLC programs from NuSCR, and it can reduce the possible errors occurring in the manual design specification, and also the software development cost and time.

On the other hands, the method [5] increases the generated FBD program by 3 times and its execution also takes much time proportionately. To be appropriate for the characteristics of PLC programs, s.t. periodic operation with a strict time bound, the additional modification and optimization manually conducted by expert developers to reduce the size of generated FBD program from NuSCR specification are required. It should be noted that an analysis method to check whether the original generated FBD program and the subsequently modified ones are behaviorally equivalent. The authors currently focus on the analysis on the behavioral preservation of modifications between FBD programs.

TRANSLATING FBD TO VERILOG

A specification in Verilog consists of one or more modules. Component modules normally have input and output *ports*. Events on the input ports cause changes on the outputs. Modules can be specified either behaviorally or structurally, or by a combination of the two. A behavioral specification defines the behavior of a module using programming language constructs. A structural specification expresses a module as a hierarchical interconnection of sub modules. These facts on Verilog enable us to directly translate FBDs to Verilog modules because FBD has the same properties as the Verilog modules.

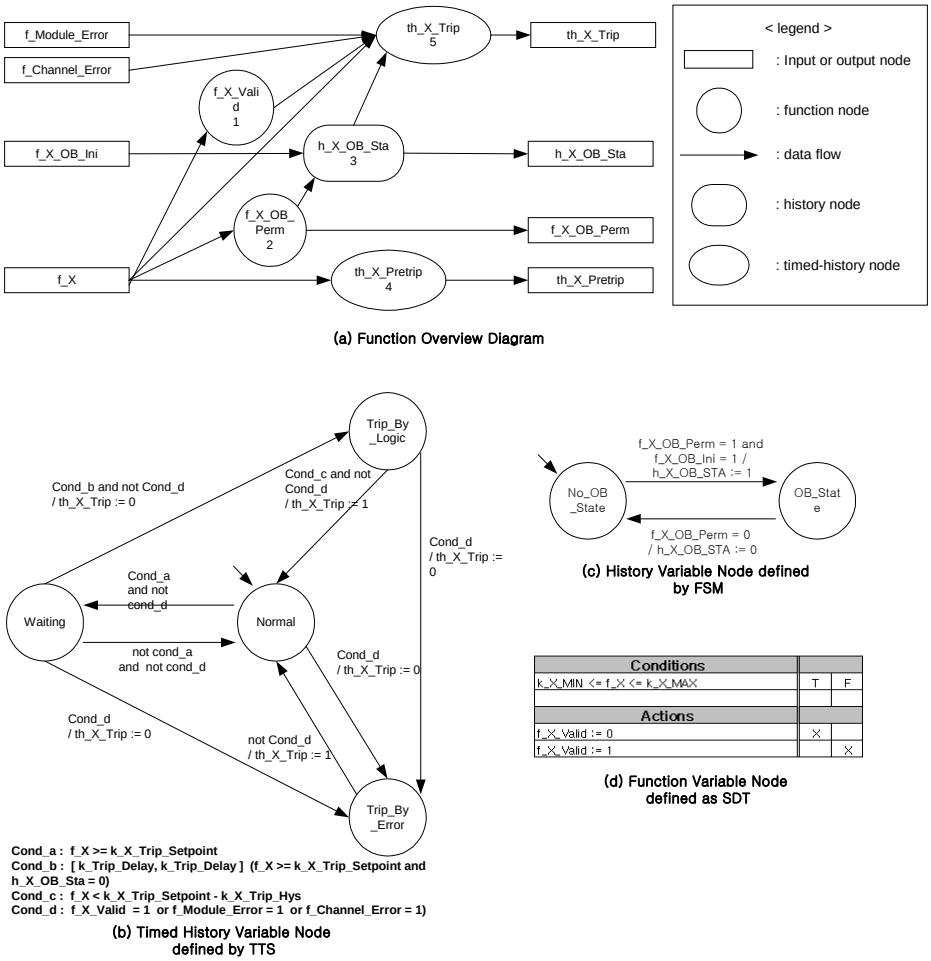
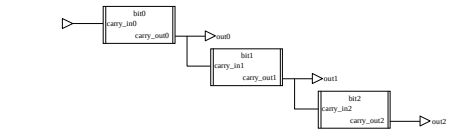
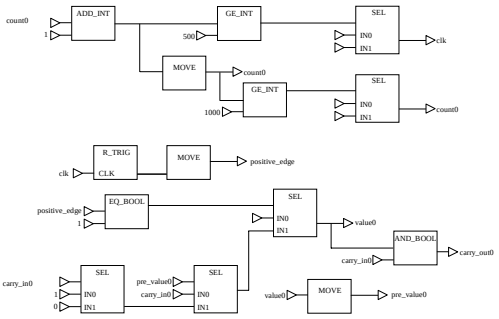


Figure 2. NuSCR specification

On the other hand, we need to abstract FBD in view of events to translate it to Verilog. In Verilog, events can be either changes in the values of *wire* variables (i.e., combinational variables) or in the values of *reg* variables (i.e., register variables), or can be explicitly generated abstract events. Generally, FBD-based PLC software distinguishes between combinational variables and register variables. Unfortunately, in some cases, it is difficult to represent them as Verilog variables without abstraction although most of them can be directly translated. Obviously, explicitly generated abstract variables must be defined in terms of Verilog through understanding the meaning of FBD and/or the connections between FBDs. Figure 3 shows the counter function blocks and their translated Verilog descriptions.



(a) FBDs for counter



(b) FBDs for counter cell

```
module counter(clk);
input clk;
wire out0, out1, out2;
counter_cell bit0 (clk, 1, out0);
counter_cell bit1 (clk, out0, out1);
counter_cell bit2 (clk, out1, out2);
endmodule

module counter_cell(clk, carry_in, carry_out);
input clk, carry_in;
output carry_out;
reg value;
assign carry_out = value & carry_in;
initial value = 0;
always @(posedge clk) begin
      case(value)
            0: value = carry_in;
            1: if (carry_in ==0)
                  value = 1;
            else value = 0;
      endcase
end
endmodule
```

(c) Verilog scripts for counter and counter cell

Figure 3. FBDs and their Verilog expressions for a counter cell

The correspondences between Figure 3 (a) and the Verilog script for 'module counter(clk)' can be easily traced because a function block in Figure 3 (a) corresponds to a functional statement in the script. Note that the clock variable, clk, was implemented within the FBDs for counter cell to emulate a real clock. In Figure 3 (b), we can see the emulating part in the upper portion. In the figure, we can easily have the pick of *wire* (combinational) variables and *reg* (register) variables. In other words, carry_i0 of function block 8 and carry_o0 of function block 10 are input and output *wire* variables, respectively. On the other hand, value0 of function block 9 and pre_value0 of function block 8 correspond to an identical *reg* variable. The lower portion of Figure 3 (b) describes the function of a counter cell, which can be directly translated into Verilog scripts in the *always* block of 'module counter_cell(clk, carry_in, carry_out)'.

FORMAL VERIFICATION USING VIS

Formal verification was performed automatically by virtue of VIS. Figure 4 shows the results of model checking.

```
VIS RELEASE 1.4 (COMPILED 3-SEP-02 AT 2:32 PM)
VIS> READ_BLIF_MV COUNTER.MV
WARNING: SOME VARIABLES ARE UNUSED IN MODEL COUNTER.
VIS> INIT_VERIFY
VIS> LANG_EMPTY -D 0
# LE: LANGUAGE IS NOT EMPTY
VIS> MODEL_CHECK -D 0 COUNTER.CTL
# MC: FORMULA PASSED --- AG(AF(BIT2.CARRY_OUT=1))
```

Figure 4. Results of formal verification

The property we wanted to check is "invariantly eventually the counter count till 8" which is expressed in CTL as "AG AF bit2.carry_out". The model checker VIS delivered a result that the formula has passed and thus the counter invariantly eventually the counter count till 8.

VIS makes it possible to check the equivalence of two software models (i.e., networks). The command *comb_verify* verifies the combinational equivalence of two networks. In particular, any set of functions,

defined over any set of intermediate variables, can be checked for equivalence between two networks. Two networks are declared combinationally equivalent if and only if they have the same outputs for all combinations of inputs and pseudo-inputs. The command *seq_verify* tests the sequential equivalence of two networks. In this case the set of intermediate variables has to be the set of all primary inputs. This produces the constraint that both networks should have the same number of primary inputs. The set of functions can be an arbitrary subset of node of the networks. The command verifies whether any state, where the values of two corresponding functions differ, can be reached from the set of initial states of the product machine. If this happens, a debug trace is provided.

CONCLUSIONS

FBD-based PLC programs take an important role in the actual software development for nuclear power plants control systems. In the design phase, the developers usually write the PLC programs manually from the requirements specification. It is mainly due to that the requirements specification is written in natural language. This paper proposes a formal design verification method, with which makes it easy and systematic to verify consistency between software requirements and software design. We have adopted a systematic FBD-based PLC program generation technique from NuSCR formal requirements specification and utilized the model checking and equivalence checking features of VIS.

This article demonstrated that VIS can make it systematic to formally verify FBD-based PLC software. For further researches, we suggest that defining formal semantics of FBD would produce more sophisticated rules that are needed to completely translate FBD to Verilog.

REFERENCES

1. M. RAUSCH, B. H. KROGH, “Formal Verification of PLC programs,” in Proc. American Control Conference, Philadelphia, PA, USA, pp. 234-238 (1998).
2. IEC (International Electrotechnical Commission), IEC Standard 61131-3: Programmable controllers-Part 3, (1993).
3. VIS Home Page: http://www-cad.eecs.berkeley.edu/~vis.
4. Han Seong Son, Kee-Choon Kwon, “A Usability Review of a Model Checker VIS for the Verification of NPP I&C System Safety Software,” in Proc. Autumn Meeting of Korea Nuclear Society, Yongpyeong, Korea (2002).
5. Junbeom Yoo, Sungdeok Cha, Changhui Kim, Duck Yong Song, “From Formal Software Requirements to PLC-based Design,” Reliability Engineering and System Safety submitted.
6. D.Harel “ Statecharts: A Visual Formalism for Complex Systems,” Science of Computer Programming, Vol. 8, pp. 231-274, (1987).
7. K.L. Heninger, “Specifying Software Requirements for Complex Systems: New Techniques and Their Application,” IEEE Transaction on Software Engineering SE-6, No.1, pp2-13, (1980).
8. Junbeom Yoo, Taihyo Kim, Sungdeok Cha, Jang-Soo Lee, Han Seong Son, “A Formal Software Requirements Specification Method for Digital Nuclear Plant Protection Systems,” Journal of Systems and Software submitted.