FBDtoVerilog 2.0: An automatic translation of FBD into Verilog to develop FPGA

Dong-Ah Lee, Eui-sub Kim, Junbeom Yoo
Division of Computer Science and Engineering
Konkuk University, Seoul, Republic of Korea
Email: {ldalove, atang34, jbyoo}@konkuk.ac.kr

Jang-Soo Lee, Jong Gyun Choi
Korea Atomic Energy Research Institute
Daejeon, Republic of Korea
Email: {jslee, choijg}@kaeri.re.kr

***Abstract*—The PLC (Programmable Logic Controller) is a digital computer which has been widely used for nuclear RPSs (Reactor Protection Systems). There is increasing concern that such RPSs are being threatened because of its complexity, maintenance cost, security problems, etc. Recently, nuclear industry is developing FPGA-based RPSs to provide diversity or to change the platform. Developing the new platform, however, is challenge for software engineers in nuclear domain because the two platform, PLC-based and FPGA-based, are too different to apply their knowledge. This paper proposes an automatic translation of FBD (Function Block Diagram: a programming language of PLC software) into HDL (Hardware Description Language). We implemented an automatic translation tool, '*FBDtoVerilog 2.0,*' which helps software engineers design FPGA-based RPSs with their experience and knowledge. Case study using a prototype version of a real-world RPS in Korea shows '*FBDtoVerilog 2.0*' translates FBD programs for PLC into HDL reasonably.**

I. Introduction

The past decade has seen the rapid development of nuclear power plants. As a consequence, the nuclear industry modernizes existing analog I&C (Instrumentation and Control) systems to digital I&C, as well as implements new digital I&C systems in new plants. As I&C systems have been digitalized, it allows software programs and network as a part of the systems. For example, a PLC (Programmable Logic Controller) has been widely used to implement real-time controllers in nuclear RPSs (Reactor Protection Systems). It includes software written in PLC programming languages [1], such as FBD (Function Block Diagram) and LD (Ladder Diagram), and communicates with other devices through networks. Digital systems offer higher reliability, better plant performance and additional diagnostic capabilities [2]. Despite the advantages, CCFs (Common Cause Failure) and security problems are rising in the field of the digital I&C systems in nuclear power plant. Furthermore, increasing complexity and maintenance cost are being brought up recently.

Diversity, which includes two or more redundant systems or components with different attributes, for the RPSs is important to prevent the threats [3]. Implementing the diversity using FPGA (Field Programmable Gate Array) with PLC is one of solutions [4], [5]. FPGA provides a powerful computation with lower hardware cost; however, it is a challenge for software engineers in nuclear domain to develop an all new platform based on FPGA from the scratch. Not only experience, knowledge and practice based on PLC are useless, but also safety certification is too costly.

This paper proposes an automatic translation of FBD, a programming language of PLC software, into behaviorally equivalent [6] Verilog [7], one of HDLs (Hardware Description Languages). The translation provides a starting point of typical FPGA developments while not giving up the PLC engineers' knowledge. Furthermore, all V&V activities and safety analyses applied PLC software are still valid in the new platform. We have developed an automatic translation tool, '*FBDtoVerilog 2.0,*' which conforms to *de factor* standard PLCopen [8].

This paper organized as follow: Section 2 briefly introduces the integrated development environment of nuclear RPSs based on both PLCs and FPGA and role of '*FBDtoVerilog 2.0*' in the environment. It also included introduction to the FBD and Verilog programming languages. Section 3 explains rules of the translation and '*FBDtoVerilog 2.0*' in detail. A case study of the translation using the proposed tool are in Section 5, and Section 6 concludes the paper and gives remarks on future research.

II. Background

A. NPP's safety-critical software Development Environment

Software in NPP (Nuclear Power Plant) such as a RPS is safety-critical software where it is essential that the system's operation is always safe [9]. RPS makes decisions for emergent reactor shutdown. Therefore, RPS software should be verified strictly and throughout entire development life-cycle. However, it is hard to apply these process and techniques, because the techniques are difficult to understand, the tools often work only in isolation, and the output is difficult to extract meaningful information. In order to overcome these difficulties, we have developed NPP's software development environment, '*NuDE* [10],' based on a formal-methods-based process [11]. We are now extending the environment from PLC-based RPS development to FPGA-based RPS development. Fig. 1 shows NuDE process for software of PLC and hardware design of FPGA.

NuDE has 3 activities—development, safety analysis, and verification and validation (V&V)—to develop RPS software. *NuSRS* uses a formal specification language, NuSCR, to specify requirements of RPS software. *NuFTA*, which generates fault tree of the requirement, helps analysts perform safety analysis. *NuSCRtoSMV* translates NuSCR formal specification into the SMV language to perform V&V using a model checking tool, Cadence SMV. FBD, in which *NuSCRtoFBD* translates NuSCR formal specification, is design language

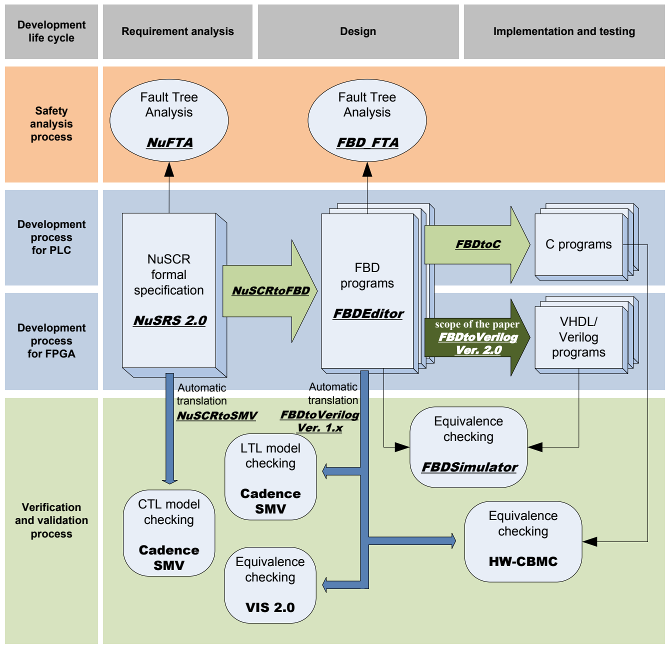


Fig. 1. NuDE (NPP's safety-critical software Development Environment): Software development, verification, and safety analysis for nuclear RPSs

for PLC software Safety analysis of the FBD design uses *FBD_FTA* generating fault trees. Former version of FBDtoVerilog, *FBDtoVerilog 1.0* [12] and *FBDtoVerilog 1.1* [13] translates FBD designs into Verilog to verify the design. Cadence SMV and VIS 2.0 are used for model checking, and HW-CBMC is used for equivalence checking with C program generated by *FBDtoC*.

'*FBDtoVerilog 2.0*' translates FBD programs into Verilog programs to develop FPGA. The most superior advantage of the translation is that PLC engineers can design hardware components (FPGA) with their experience and knowledge about development, V&V, and safety analysis of PLC software. As a result, developing FPGA-based RPS and certifying safety grade become shorter and cost effective.

B. Function Block Diagram

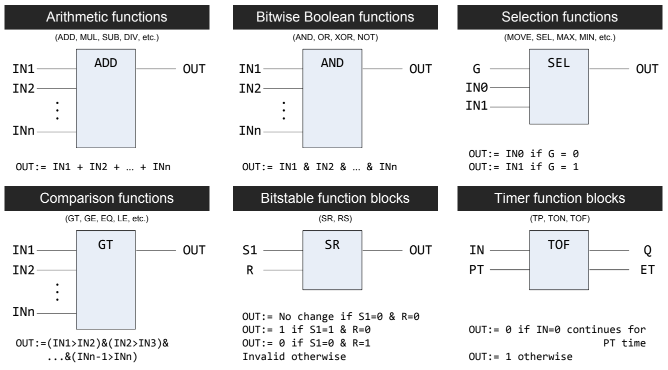


Fig. 2. An example of FBD program

The IEC 61131-3 standard includes five PLC programming languages: Structured Text (ST), Function Block Diagram (FBD), Ladder Diagram (LD), Instruction List (IL), and Sequential Function Chart (SFC). FBD's graphical notations and support for networks of software blocks "wired" together in a manner similar to circuit diagrams has lead to its widespread use. Each function block is depicted as a rectangle and is connected to other input/output variables. Among the 14 standard function groups and 4 standard function block groups defined in IEC 61131-3 Std., 6 examples are illustrated in Fig. 2. The behavior of a function block is intuitive as their names imply: `ADD`, `AND`, `SEL`, etc.

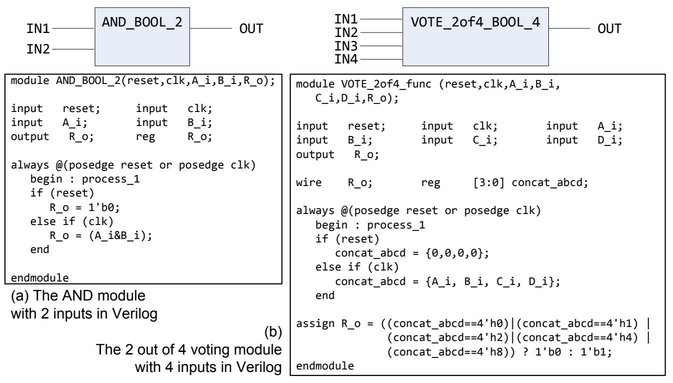
C. Verilog

Verilog is one of the most common HDLs (Hardware Description Languages) used by IC (Integrated Circuit) designers. Designs modeled in Verilog are technology independent, easy to develop and debug, and considered more readable than schematics. For this reason, Verilog is being increasingly used to specify software logic for process control systems. Verilog has several variable types. A `wire`, similar to a physical wire in a circuit, is used to connect modules in software development. A wire does not store its value and must be driven by a continuous assignment statement or by connecting it to an output of a module. On the other hand, a `reg`, used in a procedural assignment block beginning with `always`, represents a data object which holds its value from the current execution cycle to the next.

III. FBDtoVerilog 2.0

A. Translation of FBDs into Verilog

1) Verilog Library: IEC 61131-3 Std. defines standard functions and function blocks; and it specifies their behaviors. *FBDtoVerilog 2.0* uses a basic function block library which includes Verilog modules developed by experts in KAERI (Korea Atomic Energy Research Institute) in advance. The library is one-to-one which means a function or a function block of the Std. is translated into a module of Verilog. For instance, the `AND` function is translated into a `AND` module; and the `TOF` function block is translated into a `TOF` module.



```
module AND_BOOL_2(reset,clk,A_i,B_i,R_o);

input   reset;     input   clk;
input   A_i;       input   B_i;
output   R_o;      reg     R_o;

always @(posedge reset or posedge clk)
   begin : process_1
   if (reset)
      R_o = 1'b0;
   else if (clk)
      R_o = (A_i&B_i);
   end

endmodule
```

(a) The AND module with 2 inputs in Verilog

```
module VOTE_2of4_func (reset,clk,A_i,B_i,
   C_i,D_i,R_o);

input   reset;     input   clk;       input   A_i;
input   B_i;       input   C_i;       input   D_i;
output   R_o;

wire    R_o;       reg     [3:0] concat_abcd;

always @(posedge reset or posedge clk)
   begin : process_1
   if (reset)
      concat_abcd = {0,0,0,0};
   else if (clk)
      concat_abcd = {A_i, B_i, C_i, D_i};
   end

assign R_o = ((concat_abcd==4'h0)|(concat_abcd==4'h1) |
              (concat_abcd==4'h2)|(concat_abcd==4'h4) |
              (concat_abcd==4'h8)) ? 1'b0 : 1'b1;
endmodule
```

(b) The 2 out of 4 voting module with 4 inputs in Verilog

Fig. 3. `AND` and `VOTE` modules in the 'pre-translated' Verilog library

Fig. 3 shows examples of the pre-translated Verilog modules—`AND` and `VOTE`[1]—in the library. The name of functions and function blocks includes a type and an input number

[1]IEC 61131-3 Std. does not include `VOTE` functions, however, nuclear engineers put them to frequent use. NuDE also provide the functions as basic functions for efficient designing.

which is a '`[NAME]_[TYPE]_[NUM]`' form. For example, the `AND` function with two boolean inputs, (a) in Fig. 3, has the name as '`AND_BOOL_2`'; the `VOTE` function with 4 boolean inputs, (b) in Fig. 3, has the name as '`VOTE_2of4_BOOL_4`.' Names of the modules in Library are exactly same respectively. The all modules have a reset, `reset`, and a clock, `clk`, inputs. The reset initializes all output ports; and the clock synchronizes all modules.

2) Translation of Programmable Organization Units: POUs (Programmable Organization Units) are function, function block, and program. The standard functions and funcion blocks are pre-defined POUs. Users are able to define the POUs to implement specific behavior. It includes interface and body—the interface has information about inputs, outputs, and other variables; and the body implements POU's functionality using connections among variables, functions, and function blocks. POUs is a hierarchy structure: a program can include functions and function blocks; a function block can include functions and function blocks; and a function can include functions (functions and function blocks can be both standard ones and user defined ones).

FBDtoVerilog 2.0 translates a user defined POU into a module. Fig. 4 indicates a form of the translation results. The translation starts from defining interface of a module (line 1). `POUName` is the name of a target POU; and `INPUT` and `OUTPUT` are external input and output ports in the POU—the plus symbol '$^+$' means that an element in brackets '[ ]' can be repetitive as much as POU's. Line 2–10 show definition of input ports, output ports, feedback variables, constant variables, and connector/continuation pairs. The top three are mandatory inputs for all POUs, and the others are defined by interface of the POU.

```
1:  module [POUName] (rst, clk, pulse[, INPUT]^+[, OUTPUT]^+);
2:      input clk;
3:      input rst;
4:      input pulse;
5:
6:      INPUTs:[input [BitSize] Name;]^+
7:      OUTPUTs:[output [BitSize] Name;]^+
8:      FEEDBACKS:[output [BitSize] Name; reg [BitSize] Name;]^+
9:      CONSTANTs:[parameter [BitSize] Name = Value;]^+
10:     Connectors/continuations(CON):[wire [BitSize] Name;]^+
11:
12:     POUs:[ModuleName ModuleName_[LocalId](rst, clk, pulse,
13:             [, INPUT|CON|ModuleWire^1]^+ [, ModuleWire^2]^+);]^+
14:     Wiring POUs:[wire [BitSize] ModuleWire^2;]^+
15:
16:
17:     Wire to CON|OUTPUT:[assign [CON|OUTPUT] = ModuleWire^3;]^+
18:
19:     always @posedge rst or posedge clk or posedge pulse)
20:     begin
21:         if(rst) begin
22:             Output initializations:[OUTPUT <= initialValue;]^+
23:         end else if (clk) begin
24:         end
25:         if (pulse) begin
26:             Feedback assignments:[FEEDBACK <= ModuleWire^4;]^+
27:         end
28:     end
29: endmodule
```

Fig. 4. The frame of translation results from FBD to Verilog

POUs defines its behavior with wiring POUs. After defining interface, it translates a POU's behavior using module calls. All POU in a POU are translated into module calls (line 12–13). The call have connections with pre-elements, such as inputs (`INPUT`), connectors (`CON`), or output wire of other POUs (`ModuleWire`^{1}). A POU has at least one output, therefore a translated module also has ones which is defined at line 13 and 14 (`ModuleWire`^{2}). If the POU defined at line 12–13 is connected to a continuation or an output, then its output (`ModuleWire`^{3}) is assigned to the continuation or the output (line 17). `always` statement models repetition continuously in a digital circuit (line 19–28). It plays two roles—initializing storable variables (line 21–22) and modeling cycles of FBD (line 25–27). Every storable variable has its own initial value. When `rst` is a positive edge which changes the signal from FALSE to TRUE, it initializes the variables with their initial value. `clk` is criterion for operations of modules in the library, so modules receive the signal (line 12–13) and the modules use it.

Understanding `pulse` needs more background about FPGA-based RPSs. FPGA-based RPSs consists of two parts—one is the I/O part and the other is the logic part. The I/O part has scan time which means that it reads input values from external memory and writes output values to the memory periodically. The logic part calculates its result using inputs from the I/O part and sends the result to outputs in I/O part. Because every modules in the library operate by the `clk` signal, the calculation time is in proportion to a number of function and function blocks from inputs to outputs. Therefore, output variables should wait to be assigned until the calculation finishes. `pulse` is a criterion of the calculation and models the scan cycle in FBD. When `pulse` is a positive edge, then all storable variable is assigned (line 25–27).

B. Implementation of FBDtoVerilog 2.0

FBDtoVerilog 2.0 is an eclipse plug-in to be integrated into NuDE which is based on eclipse plug-in environment. It translates FBD files which conform PLCopen TC6 XML version 2.01 scheme and generates Verilog files which has "*.v" file extension. We implemented *FBDtoVerilog 2.0* to operate independently; however, now it is on *FBDEditor* in 1 for convenience. *FBDEditor* takes charge of opening and checking FBD files; and *FBDtoVerilog 2.0* takes charge of the translation only with one-click.

IV. Case Study

We performed case study to demonstrate the translation using two bistable trip logics in a RPS [14]—FIX RISING TRIP and FIX FALLING TRIP DECISION. Fig. 5 shows one of the logics partially, which *FBDEditor* loads. Each logic has 5 inputs, 8 outputs, and 33 functions and function blocks. The detail design of the logics are omitted due to space matters.

The translated file from the FBD logic is about 300 lines of Verilog code except the pre-translated library. There are three steps mainly to implement FPGA—synthesis, compile, and P&R (Place and Route). We used '*Libero Soc v11.1* [15]' for the implementation and use '*ProASIC3 Start Kit*' as FPGA hardware. *Libero Soc v11.1* reports no errors and one warning. The warning is about a unused input port which the logic has before the translation. Fig. 6 indicates the report from

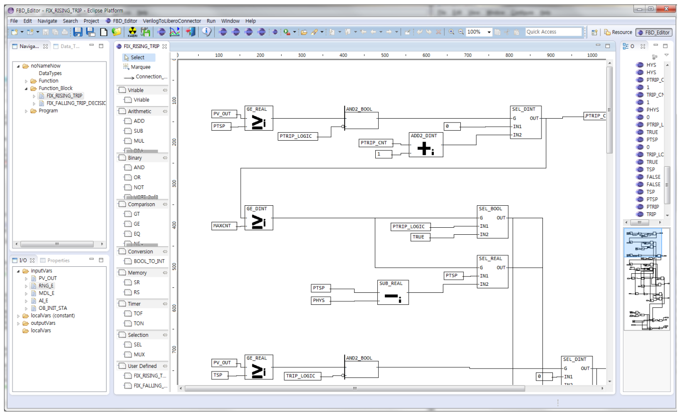
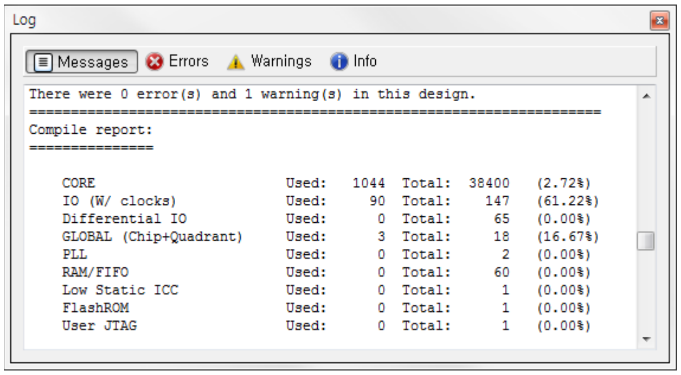


Fig. 5. A screen dump of a partial logic in BP

Libero Soc v11.1 after compile is finished. We confirmed the behavior performing simulation with *Modelsim ACTEL 10.1c* [16]. Simulation scenario is from the requirement in [14] and simulation result are equivalent.



```
There were 0 error(s) and 1 warning(s) in this design.
=====================================================================
Compile report:
===============

    CORE                       Used:   1044  Total:  38400    (2.72%)
    IO (W/ clocks)             Used:     90  Total:    147    (61.22%)
    Differential IO            Used:      0  Total:     65    (0.00%)
    GLOBAL (Chip+Quadrant)     Used:      3  Total:     18    (16.67%)
    PLL                        Used:      0  Total:      2    (0.00%)
    RAM/FIFO                   Used:      0  Total:     60    (0.00%)
    Low Static ICC             Used:      0  Total:      1    (0.00%)
    FlashROM                   Used:      0  Total:      1    (0.00%)
    User JTAG                  Used:      0  Total:      1    (0.00%)
```

Fig. 6. Compile report of a translated Verilog design using *Libero SoC v11.1*

V. Conclusion and Future Work

This paper introduces that *FBDtoVerilog 2.0* translator transforms FBD programs for PLC-based software into behaviorally-equivalent Verilog designs for developing FPGA. In order to confirm the suitability of the translation, we performed a case study with two logics of a preliminary prototype version of the RPS BP. We also performed simulation with scenario from its requirement and confirmed behavioral-equivalence.

Since a FBD program and a Verilog design are one-to-one correspondence, it is possible to manually identify structural equivalence between them. It, however, is hard to guarantee behavioral equivalence manually because theirs are not a one off but a continual ones. We are now performing verification and validation activities in diverse directions such as co-simulation between a FBD program and a Verilog design. Furthermore, we are planning to apply to *FBDtoVerilog 2.0* the 'safety/dependability case' approach [17].

Acknowledgment

This research was partially supported by a grant from the Korea Ministry of Strategy, under the development of the integrated framework of I&C conformity assessment, sustainable monitoring, and emergency response for nuclear facilities, and also partially supported by a grant from the Korea Atomic Energy Research Institute, the development of the core software technologies of the integrated development environment for FPGA-based controllers.

References

[1] "Programmable controllers - Part 3: Programming languages," *IEC 61131-3 ed3.0*, 2013.

[2] International Atomic Energy Agency, "Instrumentation and control (I&C) systems in nuclear power plants: A time of transition," 2008. [Online]. Available: http://www.iaea.org/About/Policy/GC/GC52/GC52InfDocuments/English/gc52inf-3-att5_en.pdf

[3] *Instrumentation and Control Systems Important to Safety in Nuclear Power Plants Safety Guide*, International Atomic Energy Agency, 2002.

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

[5] J.-G. Choi, et al., "Survey of the CPLD/FPGA technology for application to NPP digital I&C system," Korea Atomic Energy Research Institute (KAERI), Tech. Rep., 2009.

[6] "Functional safety of electrical/electronic/programmable electronic safety-related systems," *S+ IEC 61508*, 2005.

[7] "IEEE Standard for Verilog Hardware Description Language," *IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001)*, 2005.

[8] *XML Formats for IEC 61131-3*, PLCopen Technical Committee 6, 2009. [Online]. Available: http://www.plcopen.org

[9] I. Sommerville, *Software Engineering*. Pearson Education, 2011, pp. 299–302.

[10] J.-H. Lee and J. Yoo, "Nude: Development environment for safety-critical software of nuclear power plant," in *Transactions of the Korean Nuclear Society Spring Meeting 2012*, 2012, pp. 114–1155.

[11] J. Yoo, E. Jee, and S. Cha, "Formal modeling and verification of safety-critical software," *Software, IEEE*, vol. 26, no. 3, pp. 42–49, 2009.

[12] J. Yoo, S. Cha, and E. Jee, "Verification of plc programs written in fbd with vis," *Nuclear Eng. and Technology*, 2009.

[13] D.-A. Lee, J. Yoo, and J.-S. Lee, "Equivalence checking between function block diagrams and c programs using hw-cbmc," in *Computer Safety, Reliability, and Security*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, vol. 6894, pp. 397–408.

[14] J. G. Choi and D. Y. Lee, "Development of rps trip logic based on pld technology," *Nuclear Engineering and Technology*, vol. 44, no. 6, pp. 697–708, 2012.

[15] Microsemi. [Online]. Available: http://www.microsemi.com

[16] *ModelSim*, Mentor Graphics. [Online]. Available: http://www.mentor.com/products/fpga/model

[17] D. Jackson, "A direct path to dependable software," *Communications of the ACM*, vol. 52, no. 4, pp. 78–88, 2009.