

# FBDtoVerilog: A Vendor-Independent Translation from FBDs into Verilog Programs

Junbeom Yoo , Jong-Hoon Lee  
Div. of Computer Science and Engineering  
Konkuk University  
Seoul, Republic of Korea  
{jbyoo , kirdess}@konkuk.ac.kr

Sehun Jeong , Sungdeok Cha  
Dept. of Computer Science and Engineering  
Korea University  
Seoul, Republic of Korea  
{gifaranga , scha}@korea.ac.kr

**Abstract**—FBD (Function Block Diagram) is one of the widely used PLC (Programmable Logic Controller) programming languages in plant automation industry. Many vendors and products have their own forms and formats, which are not compatible with others. Formal verification techniques and tools for FBDs should have provided vendor- and product-specific versions. PLCopen, a vendor/product independent worldwide association, provides a standardized way to define FBDs in an XML format. This paper proposes a CASE tool, FBDtoVerilog, which translates the PLCopen-FBDs into Verilog programs. Verilog is an input programming language for formal verification tools such as VIS (Verification with Interaction and Synthesis). It had been efficiently used as an input front-end of formal verifications, when developing software controllers of nuclear power plants in Korea. We demonstrate its usefulness and effectiveness with a prototype version of FBDs which had developed for APR-1400 nuclear power reactor in Korea.

**Keywords**—Translation; PLCopen; FBD; Verilog; CASE

## I. INTRODUCTION

FBD is one of the five widely used PLC programming languages defined by International Electrotechnical Commission (IEC) [1]. It visually expresses PLC controller's behavior as sequentially interconnected function blocks. The KINCS project [2] developed a new RPS (Reactor Protection System) for Korean nuclear power plants and implemented its software in FBDs. Rigorous quality demonstration of RPS software was also required by the regulation agency (e.g., KINS [3] in Korea) prior to issuing operational approval. Automated and formal verification techniques such as model checking [4, 5] and equivalence checking [6] was applied to the FBDs in order to ensure adequate quality assurance.

Formal verification techniques have their own input front-ends. For example, the VIS verification system [7] needs Verilog program, while the SMV [8] model checker does SMV input program or Verilog program. Translation from FBDs into these front-ends is therefore the first step to applying various formal verification techniques into FBD programs. Our former researches on FBD verifications, 'FBD Verifier' and 'PLC Verifier' [9, 10] had to use a FBD format specific to POSCO ICT [11], which generated from its PLC engineering tool 'pSET' [12]. Some changes in the format, however, made us difficult to keep consistency and correctness of the automatic

translators and verification tools. This paper proposes a CASE tool, 'FBDtoVerilog' translating FBDs into Verilog programs, but uses a de facto standard XML format of FBD, proposed by PLCopen [13]. PLCopen is a vendor- and product-independent worldwide association. FBDtoVerilog can translate into Verilog programs FBDs from any vendors complying with the association's standard.

We demonstrated correctness and effectiveness of the proposed translator through a case study, formal verification of FBD programs using the SMV and the VIS. We used a prototype version [14] of FBD programs which had developed for a nuclear reactor protection system in Korea. The remainder of the paper is as follows. Section 2 introduces the FBD and PLC open association briefly. It also introduces relevant features of Verilog programming language, which are pertinent to our discussion. Section 3 introduces the CASE tool FBDtoVerilog. Section 4 explains a case study of formal verification using the proposed tool. Section 5 concludes the paper.

## II. BACKGROUND

### A. Function Block Diagram

An FBD (Function Block Diagram) consists of an arbitrary number of function blocks, 'wired' together in a manner similar to a circuit diagram. The international standard IEC 61131-3 defined 10 categories and all function blocks. For example, the function block ADD performs arithmetic addition of  $n+1$  IN values and stores the result in OUT variable. Others are interpreted in a similar way.

Fig.1 shows a part of preliminary FBD programs for the KNICS RPS BP (Bistable Processor) logic. The former was generated mechanically [15] from a formal requirements specification [14], while the latter was developed by domain experts. Even though they look different in appearance, they show the same behavior. We used these FBDs as examples to keep consistent with our former work and aid understanding of FBD programs. These FBDs both creates a warning signal 'th\_X\_Pretrip' when the pre-trip condition (i.e., reactor shutdown) remains true for  $k\_Trip\_Delay$  time units as implemented in the TOF function block. The number in parenthesis above each function block denotes its execution

order. The output ‘*th\_Prev\_X\_Pretrip*’ from MOVE stores current value of ‘*th\_X\_Pretrip*’ in order to use in the next execution cycle. A large number of FBDs similar to Fig.1 and Fig.2 are assembled hierarchically and executed according to a predefined sequential execution order.

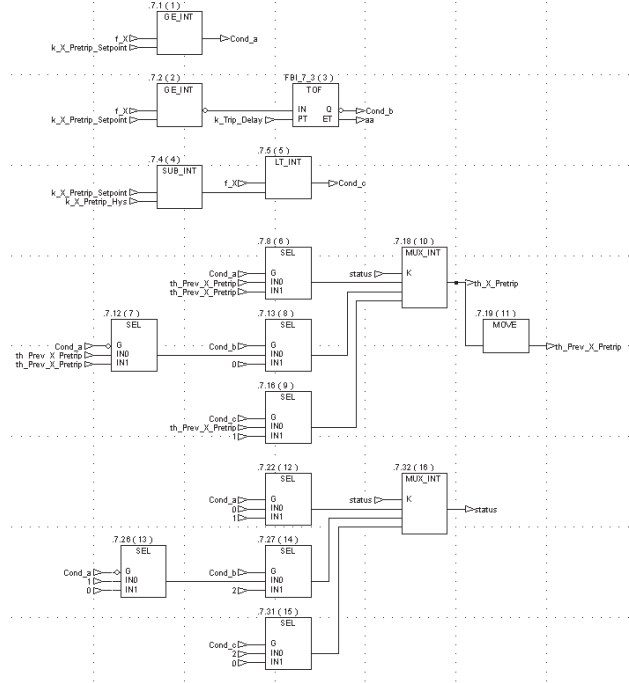


Figure 1. An FBD for *th\_X\_Pretrip* logic, generated mechanically

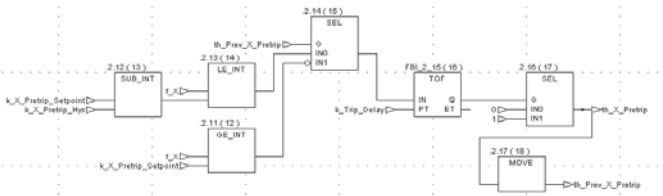


Figure 2. An FBD for *th\_X\_Pretrip* logic, developed by domain experts

### B. PLC open

PLCopen [13] is a vendor- and product-independent worldwide association, aiming to resolve topics related to control programming and to support the use of international standards IEC 61131-3 [1]. A working group named TC6 for XML (eXtended Markup Language) in PLCopen has defined an open interface between all different kinds of software tools, which provides the ability to transfer one’s information to other platforms. This paper used the XML specification defining FBD programming languages. The format unfortunately does not include all items which we need to translate FBDs into Verilog programs, so we used a few items in the specification for our specific purpose. The details will be introduced in Section 4.

### C. Verilog Programming

Verilog is one of the most common Hardware Description Languages (HDLs) used by Integrated Circuit (IC) designers.

Many verification and analysis techniques and tools widely use Verilog as an input programming language.

Fig.3 shows a Verilog program translated from the FBD described in Fig.2 according to the translation rules [15]. There are two inputs and two outputs. As input prefixes “*k*” indicate constants variables. *th\_Prev\_X\_Pretrip* is used as both input and output. Since it stores the value of *th\_X\_Pretrip* using the MOVE function block, we defined it as a *reg* variable in lines (8) and (32). The FBD’s output is produced in the *assign* statements (12) ~ (18) by composing several function blocks in the FBD. It also uses the variable *timer* to emulate the TOF function block, which we emulate with *procedural* assignments using *always* statements (19) ~ (31). We restricted the number of TOF internal states to six in this example as defined in (1). In addition, we used the *clk* variable, reserved for simulation purposes in the VIS verification system, to simulate cyclic executions of PLCs.

```

(1) typedef enum {T0, T1, T2, T3, T4, T5} timer_state;
(2) `define k_X_Pretrip_Setpoint 30;
(3) `define k_X_Pretrip_Hys 30;

(4) module th_X_Pretrip(clk, f_X, th_X_Pretrip);
(5)   input clk;
(6)   input [0:6] f_X;
(7)   output th_X_Pretrip;

(8)   reg th_prev_X_Pretrip;
(9)   timer_state reg timer;

(10)  initial th_prev_X_Pretrip = 1;
(11)  initial timer = T0;

(12)  assign th_X_Pretrip =
(13)    ((th_prev_X_Pretrip == 0) && (f_X <= (k_X_Pretrip_Setpoint - k_X_Pretrip_Hys))) ? 1 :
(14)    ((th_prev_X_Pretrip == 0) && (f_X > (k_X_Pretrip_Setpoint - k_X_Pretrip_Hys)) && (timer == T5)) ? 0 :
(15)    ((th_prev_X_Pretrip == 1) && (f_X > (k_X_Pretrip_Setpoint - k_X_Pretrip_Hys)) && (timer != T5)) ? 1 :
(16)    ((th_prev_X_Pretrip == 1) && (f_X < k_X_Pretrip_Setpoint)) ? 1 :
(17)    ((th_prev_X_Pretrip == 1) && (f_X >= k_X_Pretrip_Setpoint) && (timer == T5)) ? 0 :
(18)    ((th_prev_X_Pretrip == 1) && (f_X >= k_X_Pretrip_Setpoint) && (timer != T5)) ? 1 : 0;

(19)  always @(posedge clk) begin
(20)    if(f_X >= k_X_Pretrip_Setpoint) begin
(21)      case(timer)
(22)        T0: timer = T1;
(23)        T1: timer = T2;
(24)        T2: timer = T3;
(25)        T3: timer = T4;
(26)        T4: timer = T5;
(27)        T5: timer = T5;
(28)      endcase
(29)    end
(30)    else
(31)      timer = T0;

(32)    th_prev_X_Pretrip = th_X_Pretrip;
(33)  end

(34) endmodule

```

Figure 3. A Verilog program translated from the FBD in Fig.3

### III. FBDTOVERILOG

We have used the proposed, but not fully refined, FBD definition and translation rules [16] to formally verify FBD programs in the KNICS project. Fig.4 briefly shows how we have used them to verify the FBD programs with various verification techniques and tools. It is a part of PLC-based software development framework we proposed in [15]. We planned to apply two formal verification techniques into the FBDs, the model checking and the equivalence checking. While the former can prove mathematically whether the FBD satisfies important properties, the latter can conclude whether two different FBDs show the same behavior or not.

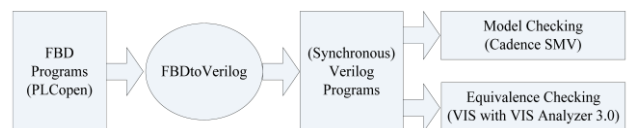


Figure 4. The use of the proposed translator in formal verifications

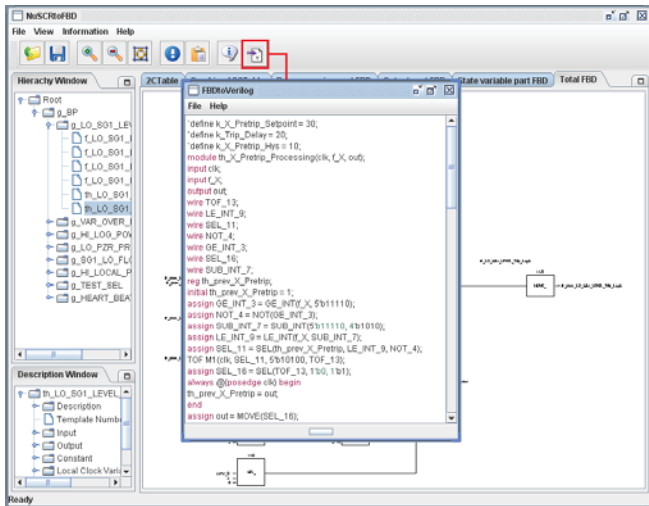


Figure 5. FBDtoVerilog v1.0 Screen-dump

We had developed automatic translator and verification assisting tool *FBD Verifier* [9], and applied them into the KNICS project in part [17]. However, our former work started with a specific version of FBDs specialized for POSCON ICT. In order to apply useful formal verification techniques with no hindrance from the compatibility problem, we decided to separate the translator from the specific FBD and used standard XML format of FBD. Fig.5 depicts a screen-dump of *FBDtoVerilog 1.0* CASE tool which we have developed. It is embedded in *NuSCRtoFBD 3.0* and reads standard FBDs of PLCopen and produces (synchronous) Verilog programs.

*FBDtoVerilog* used an *addData*, general-purpose element of the PLCopen XML specification [13]. *NuSCRtoFBD 3.0* generates PLCopen specific XML that every single function block element belongs to an externally visible output which *addData* element stores its name. Fig.6 shows *LE\_INT* block cooperate with computing output *th\_X\_Pretrip*. *FBDtoVerilog* uses the information to translate an FBD’s flat structure into a Verilog module’s hierarchy structure.

```
<block instanceName="LE_INT" localId="9" typeName="LE_INT" executionOrderId="15">
  <inputVariables>
    ...
  </inputVariables>
  <outputVariables>
    ...
  </outputVariables>
  <addData>
    <data name="th_X_Pretrip" handleUnknown="discard">
      <BOOL/>
    </data>
  </addData>
</block>
```

Figure 6. Usage of *addData* element in *th\_X\_Pretrip* FBD specification

The current version of *FBDtoVerilog 1.0* has some room to improve. First, it produces incomplete Verilog code that requires manual post-process to supply variable size in bit vectors. Performing formal verification activities such as equivalence checking and model checking require complete size determination. Second, it translates every function block, even though they are too simple to be defined as a Verilog function. We suggest practically possible translation option in

Table 1. These aspects will implement in next version of *FBDtoVerilog*.

Table 1. Alternative optimized function block translation rule

	Current rule	Optimized rule
<b>SEL</b>	<pre>var = SEL(a, b, c); ... function SEL;   input in1;   input in2;   input in3;   begin     SEL = (in1 == 1) ? in3 :           in2;   end endfunction</pre>	<pre>var = (a == 1) ? b : c;</pre>
<b>ADD</b>	<pre>var = ADD(a, b); ... function [0:6] SUB_INT;   input [0:6] in1;   input [0:6] in2;   begin     SUB_INT = (in1 - in2);   end endfunction</pre>	<pre>var = a + b;</pre>

#### IV. CASE STUDY

We performed a case study as described in Fig.7 to validate correctness of *FBDtoVerilog 1.0*. We translated the system FBD *g\_LO\_SG1\_Level* depicted abstractly in Fig.1 and Fig.2 into Verilog programs. And we applied manual post-processing on the translated code with preserving its original semantic as we mentioned in Section 3 (see Fig.8). We had plan performing Cadence SMV model checking and the VIS equivalence checking against the Verilog program. When preparing the case study, we only focused on checking the validity of the CASE tool.

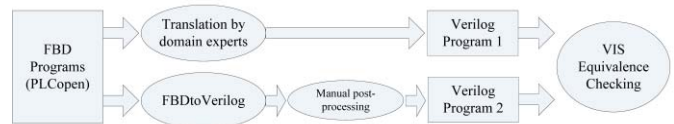


Figure 7. Case study plan

The VIS equivalence checking result shows "sequentially equivalent" message as we can see in Fig.9, which means two Verilog programs have same output behavior against same inputs. We also conducted flawless examination of two source codes to validate our tool’s correctness, since source codes have quite different coding style. For example, original domain expert generated code doesn’t contain user-define functions that our code has.

Cadence SMV model checker cannot read the Verilog program which the current version of *FBDtoVerilog* produced. We found out that the model checker forbid the reuse of functions such as *SEL* or *ADD* in our code. We are working on this issue with more refined translation rules. From the results, we can say that our proto-type *FBDtoVerilog* archived its main purpose at minimum that the translated Verilog code has same behavior with the original code developed and certified by domain experts.

```

module main(clk, f_X, th_X_Pretrip);
input clk;
input [0:6] f_X;
output th_X_Pretrip;

//wires and regs
wire tof_out;
wire Sel1;
wire Sel2;
...

reg th_prev_X_Pretrip;
initial th_prev_X_Pretrip = 1;

// assign temp wire variables
assign Ge_int = GE_INT(f_X, 7'b0011110);
assign Sel1 = SEL(th_prev_X_Pretrip, Le_int, Not);
...
TOF M1(clk, Sel1, 7'b0000101, tof_out);
assign th_X_Pretrip = MOVE(Sel2);

always @(posedge clk) begin
th_prev_X_Pretrip = th_X_Pretrip;
end

//function blocks
function GE_INT;
input [0:6] in1;
...

```

Figure 8. Translated Verilog code from the FBD in Fig.3

```

vis release 2.3 (compiled 2011 년 3 월 14 일 월요일 22 시 50 분 06 초 KST)
vis> read_blif_mv ./th_X_Pretrip_tr_v4.mv
Warning: Some variables are unused in model NOT.
Warning: Some variables are unused in model GE_INT.
Warning: Some variables are unused in model SEL.
Warning: Some variables are unused in model MOVE.
Warning: Some variables are unused in model LE_INT.
vis> init_verify
vis> seq_verify ./th_X_Pretrip_custom_v3.mv
Networks are sequentially equivalent.

```

Figure 9. VIS equivalence checking result

Our future work will focus on implementing next version of *FBDtoVerilog*. First issue is fully automatic Verilog code generation feature that includes variable size determination algorithm. Second issue is Cadence SMV compatible code generation feature. And we will plan the case study that verifies further correctness of the *FBDtoVerilog* through VIS equivalence checking and Cadence SMV model checker using all FBDs used in the KNICS project.

## V. CONCLUSION

As safety critical systems are using FBD as standard representation of software design, software verification on FBDs becomes indispensable. Our former researches on FBD verifications used a vendor-specific format of FBD, and it made us difficult to keep consistency and correctness of the automatic translator and verification tools. This paper proposes a CASE tool, '*FBDtoVerilog*' translating FBDs into Verilog programs, but uses a de facto standard XML format of FBD, proposed by PLCopen. We demonstrated correctness and effectiveness of the assisting tool through a case study, formal verification of FBD programs using the VIS. We used a prototype version of FBD programs developed for a nuclear reactor protection system in Korea. The case study demonstrated that the CASE tool, *FBDtoVerilog* translates standard FBDs into Verilog programs correctly and efficiently.

## ACKNOWLEDGMENT

This research was partially supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency (NIPA-2011-(C1090-1131-0008) and NIPA-2010-(C1090-1031-0003)). This research was also supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2010-0002566).

## REFERENCES

- [1] IEC (International Electrotechnical Commission), International standard for programmable controllers: Programming languages: Part 3 (IEC 61131-3), 1993.
  - [2] KNICS (Korea Nuclear Instrumentation & Control System R&D Center), <http://www.knics.re.kr/english/eindex.html>.
  - [3] KINS (Korea Institute of Nuclear Safety), <http://www.kins.re.kr>.
  - [4] 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 Systems*, Vol. 8, No. 2, pp.244-263, 1986.
  - [5] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled, *Model Checking*, MIT Press, 1999.
  - [6] Shi-Yu Huang and Kwang-Ting(Tim) Cheng, *Formal Equivalence Checking and Debugging*, chapter 4, Kluwer Academic Publishers, 1998.
  - [7] Robert K. Brayton, Gary D. Hachtel, Alberto Sangiovanni-Vincentelli, Fabio Somenzi, Adnan Aziz, Szu-Tsung Cheng, Stephen A. Edwards, Sunil P. Khatri, Yuji Kukimoto, Abelardo Pardo, Shaz Qadeer, Rajeev K. Ranjan, Shaker Sarwary, Thomas R. Shiple, Gitanjali Swamy, and Tiziano Villa, "VIS : A system for verification and synthesis," In the Eighth International Conference on Computer Aided Verification, CAV '96, pages 428-432, 1996.
  - [8] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
  - [9] Eunyoung Jee, Seungjae Jeon, Sungdeok Cha, Kwanyong Koh, Junbeom Yoo, Geeyong Park, and Poonghyun Seong, "FBD Verifier: Interactive and Visual Analysis of Counterexample in Formal Verification of Function Block Diagram," *Journal of Research and Practice in Information Technology*, Vol.42, No.3, pp.255-272, August, 2010.
  - [10] Junbeom Yoo, Sungdeok Cha, and Eunyoung Jee, "Verification of PLC Programs written in FBD with VIS," *Nuclear Engineering and Technology*, Vol.41, No.1, pp.79-90, 2009.
  - [11] POSCO ICT, <http://www.poscoict.co.kr>.
  - [12] S. Cho, K. Koo, B. You, T.-W. Kim, T. Shim, and J.S. Lee, "Development of the loader software for PLC programming," In *Conference of the Institute of Electronics Engineers of Korea*, Vol.30, pp.959-960, 2007.
  - [13] PLCopen for efficiency in automation, <http://www.plcopen.org>.
  - [14] KAERI (Korea Atomic Energy Research Institute). *Formal SRS for Reactor Protection System*, KNICS-RPS-SVR131-01 Rev.00, 2005.
  - [15] Junbeom Yoo, Sungdeok Cha, Chang Hwoi Kim, and Duck Yong Song, "Synthesis of FBD-based PLC Design from NuSCR Formal Specification," *Reliability Engineering and System Safety*, Vol.87, No.2, pp.287-294, 2005.
  - [16] Junbeom Yoo, Eunyoung Jee, and Sungdeok (Steve) Cha, "Formal Modeling and Verification of Safety-Critical Software," *IEEE Software*, Vol.26, No.3, pp.42-49, May/June 2009.
- Junbeom Yoo, Eunyoung Jee, and Sungdeok Cha, "A Verification Framework for FBD based Software in Nuclear Power Plants," In *The 15th Asia Pacific Software Engineering Conference (APSEC)*, pages 385-392, 2008