Equivalence Checking between Function Block Diagrams and C Programs Using HW-CBMC

Dong-Ah Lee[1], Junbeom Yoo[1], and Jang-Soo Lee[2]

[1] Division of Computer Science and Engineering, Konkuk University
1 Hwayang-dong, Gwangjin-gu, Seoul, 143-701, Republic of Korea
{ldalove,jbyoo}@konkuk.ac.kr
http://dslab.konkuk.ac.kr
[2] Korea Atomic Energy Research Institute, 150 Deokjin, Yuseong
Daejeon, 305-335, Republic of Korea
jslee@kaeri.re.kr
http://www.kaeri.re.kr

Abstract. Controllers in safety critical systems such as nuclear power plants often use Function Block Diagrams (FBDs) to design embedded software. The design program are translated into programming languages such as C to compile it into machine code for particular target hardware. It is required to verify equivalence between the design and the implementation, because the implemented program should have same behavior with the design. This paper introduces a technique about verifying equivalence between a design written in FBDs and its implementation written in C language using HW-CBMC. To demonstrate the effectiveness of our proposal, as a case study, we used one of 18 shutdown logics in a prototype of Advanced Power Reactor's (APR-1400) Reactor Protection System (RPS) in Korea. Our approach is effective to check equivalence between FBDs and ANSI-C programs if the automatically generated Verilog program is translated into appropreate one of the HW-CBMC.

Keywords: Equivanelce Checking, Behavioral Consistency, FBDs, Verilog, ANSI-C, HW-CBMC.

1 Introduction

Controllers in safety critical systems such as nuclear power plants use Function Block Diagrams (FBDs) to design embedded software. The design is implemented using programming languages such as C to compile it into a particular target hardware. The implementation must have the same behavior with the design's one and it should be verified explicitly. For example, Korea Nuclear Instrumentation & Control System R&D Center (KNICS) [1] has developed a loader software, POSAFE-Q Software Engineering Tool (pSET) [2], to program POSAFE-Q Programmable Logic Controller (PLC). It provides Integrated Development Environment (IDE) including editor, compiler, downloader, simulator and monitor/debugger. It uses FBD, Ladder Diagram (LD) and Sequential Function Chart (SFC) to design a program of PLC software. The pSET translates

F. Flammini, S. Bologna, and V. Vittorini (Eds.): SAFECOMP 2011, LNCS 6894, pp. 397–408, 2011.

FBDs program into ANSI-C program to compile it into machine code for PLC. The ANSI-C program must have same behavior with the FBDs program.

Language difference between the design and the implementation (i.e., FBDs and ANSI-C) makes preserving behavioral consistency difficult. There are several studies and products to guarentee the behavioral consistency. Mathematical proof or verification of compiler, including code generator and translator, can help guarentee the behavioral consistency between two programs written in different languages. Those techniuqes have weaknesses, which are high expenditure [3] and repetitive fulfillment whenever the translator is modified. On the other hand, RETRANS [4] which is a verification tool of automatically generated source code, dosen't consider transformation rules of a specific translator. It analyzes only the generated source code to reconstruct its inherent functionality and compares it with its underlying specification to demonstrate functional equivalence between both. This approach requires additional analysis to reconstruct useful information from the generated source code. Our approach is verification of equivalence between design program and its implementation program without additional analysis or transformation of the implementation program.

This paper introduces a technique verifying equivalence between design program written in FBDs and its implementation program written in ANSI-C using HW-CBMC [5]. The HW-CBMC is formal verification tool, verifying equivalence between hardware and software description. It requires two inputs for checking equivalence, Verilog for hardware and ANSI-C for software. We used it for verifying equivalence between design program and its implementation program. We first translated the design program written in FBDs into semantically equivalent Verilog program [6][7] to use as an input program of HW-CBMC, and verified equivalence between the Verilog program and the ANSI-C implementation of the FBD designs. We performed a case study to demonstrate its feasibility with one of 18 shutdown logics in a prototype of Advanced Power Reactor's (APR-1400) Reactor Protection System (RPS) in Korea. We translated FBDs program of the shutdown logic program into Verilog program. The ANSI-C implementation of the shutdown logic was implemented manually, because it hadn't prepared yet. We found specific features of the HW-CBMC that input program should follow specific rules related with naming variables or function calls of Verilog program. We, therefore, should modefy the Verilog program to be appropriate of the rules, and verified equivalence between the design and implementation of the shutdown logic. We founded that our approach is feasible to verify equivalence between design program written in FBD and its implementation program written in ANSI-C using ANSI-C.

The remainder of the paper is organized as follows: Section 2 explains equivalence checking, and translation rules from FBDs into Verilog briefly. Section 3 describes our technique to verify equivalence between design program and its implementation program. Section 4 explains a case study which verify equivalence between one of 18 shutdown logics in RPS of APR-1400 and its implementation. Section 4 also explains modification for the Verilog program and results of the checking. We conclude the paper at Section 5.

2 Related Work

2.1 Equivalence Checking

Equivalence checking is a technique to check behavioral consistency between two programs. The VIS (Verification Interacting with Synthesis) [9] is a widely used tool for formal verification, synthesis, and simulation of finite state systems. It uses Verilog as a front-end, and also provides combinational and sequential equivalence checking of two Verilog programs. The combinational equivalence of the VIS provides a sanity check when re-synthesizing portions of a network, and its sequential verification is done by building the product finite state machine. The checking whether a state where the values of two corresponding outputs differ, can be reached from the set of initial states of the product machine. On the other hand, there is a study for equivalence checking between two different discriptions. [8] presents a formal definition of equivalence between Transaction Level Modeling (TLM) and Register Transfer Level (RTL) is presented. The TML is the reference modeling style for hardware/software design and verification of digital systems, and the RTL is a level of abstraction used in describing the operation of a synchronous digital circuit. The definition is based on events, and it shows how such a definition can be used for proving the equivalence between both.

2.2 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 [11] defined 10 categories and all function blocks as depicted in Fig.1. For example, the function block ADD performs arithmetic

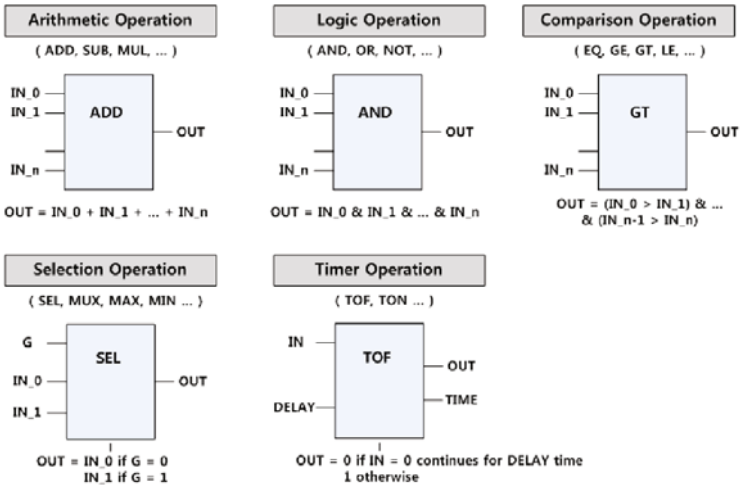

Fig. 1. A part of function blocks and categories defined in IEC 61131-3

addition of n+1 IN values and stores the result in OUT variable. Others are interpreted in a similar way. The FBD described in Fig. 4 which is our case study consists of a set of interconnection.

2.3 Transformation from FBDs into Verilog

Our approach of equivalence checking using HW-CBMC first translates FBDs into Verilog. Translation rules were proposed in [6]. The rule consists of three parts, corresponding to unit, component and system FBDs respectively. It defines the IEC 61131-3 [11] FBDs as state transition systems.

First part of the rules describes how a unit of FBD is translated into a function in Verilog language. It first determines Verilog function type, and each input and its type are declared. Behavioral description of the function is then followed, such as arithmetic, logic or selection operations. Second part explains translation rules for component FBDs. The component FBD is a logical block of independent function blocks which a number of function blocks are interconnected with to generate meaningful outputs. The rules of the second part declare a name of component FBD, ports and register type variables. If an output variable is also used as input, it is declared as reg type as its value is to be used in the next cycle. Every Verilog function is called if there is a function according to its execution order to generate outputs of the component FBD. Every function block is separately translated as a Verilog function and included in the definition of module for the component FBD. The last part describes how a system FBD is translated into a Verilog program. A system FBD contains a number of component FBDs and their sequential interconnections. While translation rules for system FBDs look similar to the rules of component FBDs, it calls Verilog Modules instead of Verilog function. Verilog modules are instantiated and called according to their execution order with outputs communicated.

2.4 HW-CBMC

The HW-CBMC [5] is a testing and debugging tool for verifying behavioral consistency between two implementations of the same design: one written in ANSI-C, which is written for simulation, and one written in register transfer level HDL, which is the actual product. Motivation of the HW-CBMC is to reduce additional time for debugging and testig of the HDL implementation in order to produce the chip as soon as possible. The HW-CBMC reduces cost by providing automated way of establishing the consistency of HDL implementation using the ANSI-C implementation as a reference, because debugging and testing cost of the ANSI-C implementation is usually lower.

The HW-CBMC verifies the consistency of the HDL implementation written in Verilog using the ANSI-C implementations as a reference. The data in the Verilog modules is available to the C program by means of global variables, and the Verilog model makes a transition once the function *next_timeframe()* is called in C program. The HW-CBMC provides counterexample when a condition of the

function *assert(condition)* in C program is not satisfied with two trace: One for the C program and a separate trace for the Verilog module. The values of the registers in the Verilog module are also shown in the C trace as part of the initial state.

3 Equivalence Checking

3.1 Equivalence Checking Process

This section introduces how the equivalence checking works in a software develop-menet process for nuclear power plant's reactor protection system. A part of exisiting software development process for KNICS's ARP-1400 RPS is described in a upper part of Fig. 2 devided by dotted line. Each development phase has verification or testing techniques to guarantee its correctness. The design phase uses model checking techniques to verify it against important design properties. The design written in FBDs is translated into an input language of specific model checker such as SMV [10] and the model checker verifies that the design program satisfies its properties. After the model checking, the code generator generates an implementation written in ANSI-C from the design program. The ANSI-C program is compiled into executable machine code for PLCs, and the testing of the executable machine code is performed after being loaded on PLCs. The code

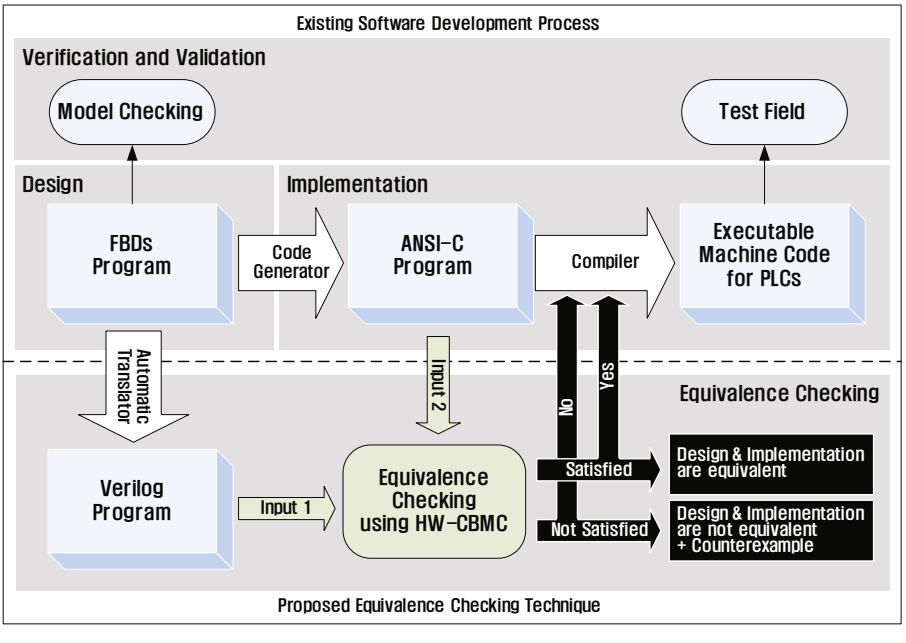

Fig. 2. A part of proposed software development process using equivalence checking with POSAFE-Q Software Engineering Tool

generator must guarantee the correctness of its behavior, since it has to translate all of behavior and feature of FBDs program to ANSI-C program precisely. It, therefore, is necessary to use a certified code generator or prove the correctness of a code generator mathematically. The mathematical proof is difficult to apply and requires high expenditure. It also requires additional proofs whenever the code generator is upgraded.

Our approach of verifying consistency between design and implementation programs includes the equivalence checking using the HW-CBMC, described in a lower part of Fig. 2, without considering the code generator. The HW-CBMC for equivalence checking requires two input programs, Verilog and ANSI-C programs. The language of implementation program, the ANSI-C program, is the same with one of input programs of the HW-CBMC, while the design program is different. We, therefore, should translate the design program written in FBDs into semantically equivalent Verilog program. If the equivalence checking is satisfied, then the ANSI-C program will be compiled into executable machine code for PLCs. If the equivalence checking, on the other hand, is not satisfied, then we will have to look into the code generator in depth.

3.2 Verilog Program for HW-CBMC

We found that the HW-CBMC has specific rules for input programs. Verilog program as an input language of the HW-CBMC should keep the rules as following:

- A variable's name should be different from the module's name which defines and uses it.
- Function calls are not allowed.

Our research team has developed automatic *FBDtoVerilog* translators [6][12], and the current version uses the rule that a module name should be the same as that of its output variable name, in accordance with the commonly accepted usage of FBDs. The HW-CBMC maps variables which are defined as global variables in ANSI-C program onto the data in Verilog modules. If there is a variable which has same name with its module, the HW-CBMC maps variables incorrectly. The variables of module, therefore, must have a different name with the name of its module.

The *FBDtoVerilog* translates each function block in FBDs into a Verilog function, and call it according to the execution order in the Verilog module definition. However, we found that the HW-CBMC does not allow to use function calls. In order to allow Verilog program as one input of HW-CBMC, we must translate a function into a module and the effect of the modification should be analyzed further. Fig. 3 shows an example of the translation. We first translated a function *GE_INT* into a module *GE_INT()*, moved it out from *main_module()*. Next, we declared input arguements and an additional output arguement. We replaced the function call, *GE_INT(IN, 7'b0011110)*, with the module call, *GE_INT M_GE_INT(IN, 7'b0011110, M_GE_INT_OUT)*. The module calls must follow order of function blocks of FBDs in order to make behavior

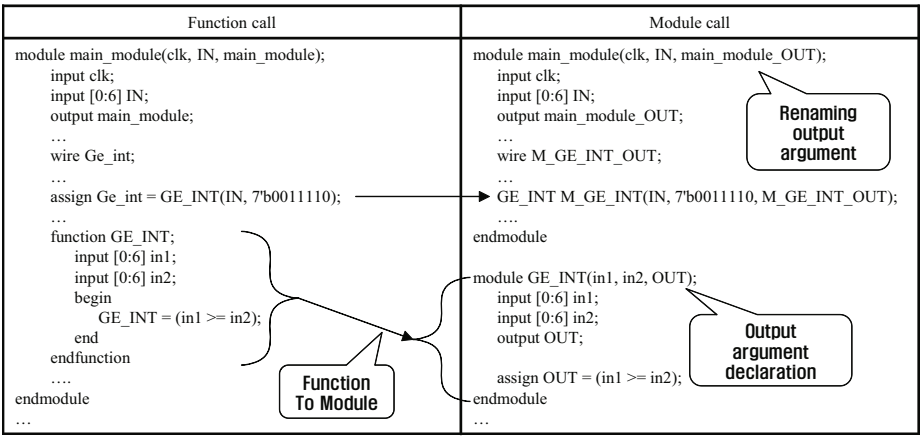

Fig. 3. An example of translation from a function to a module

of the Verilog program same with the FBDs. Fig. 3 also shows remaning output variable of *main_module*.

4 Case Study

We applied the proposed equivalence checking approach to one of 18 shutdown logic programs, named *th_X_Pretrip*, in ARP-1400 RPS developed in Korea. We first describe the FBD of *th_X_Pretrip* and Verilog program which was automatically generated by the *FBDtoVerilog 1.0* in Subsection 4.1. Then we introduce the desirable modifications required to use the HW-CBMC. Subsection 4.2 shows the details of C programs generated by the C code generator. Subsection 4.3 shows the equivalence checking of the Verilog and C program in details.

4.1 *th_X_Pretrip* Program

The *th_X_Pretrip* logic consists of 8 Function Blocks as depicted in Fig. 4. It creates a warning signal, *th_X_Pretrip* (name of logic and output could be same), when the pretrip condition (e.g., 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* for using in the next execution cycle. A large number of FBD is assembled hierarchically and executed according to predefined sequential execution order.

As described in Fig. 5, the Verilog program for the *th_X_Pretrip* logic has two inputs, *clk* and *f_x*, and one output, *th_X_Pretrip*. 7 functions and one module match with function blocks of the *th_X_Pretrip* FBDs. The output, *th_X_Pretrip*, will become 0 when the trip condition remains true for 5 time units which TOF module counts. We modified automatically generated Verilog program. We first

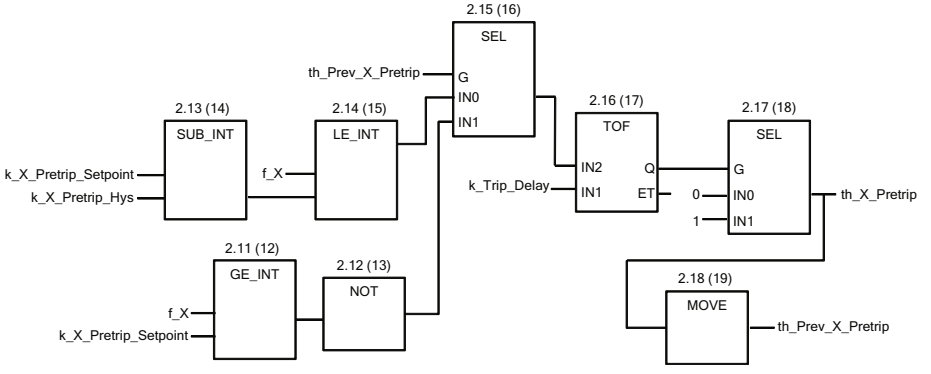

Fig. 4. Function block diagrams of *th_X_Pretrip*

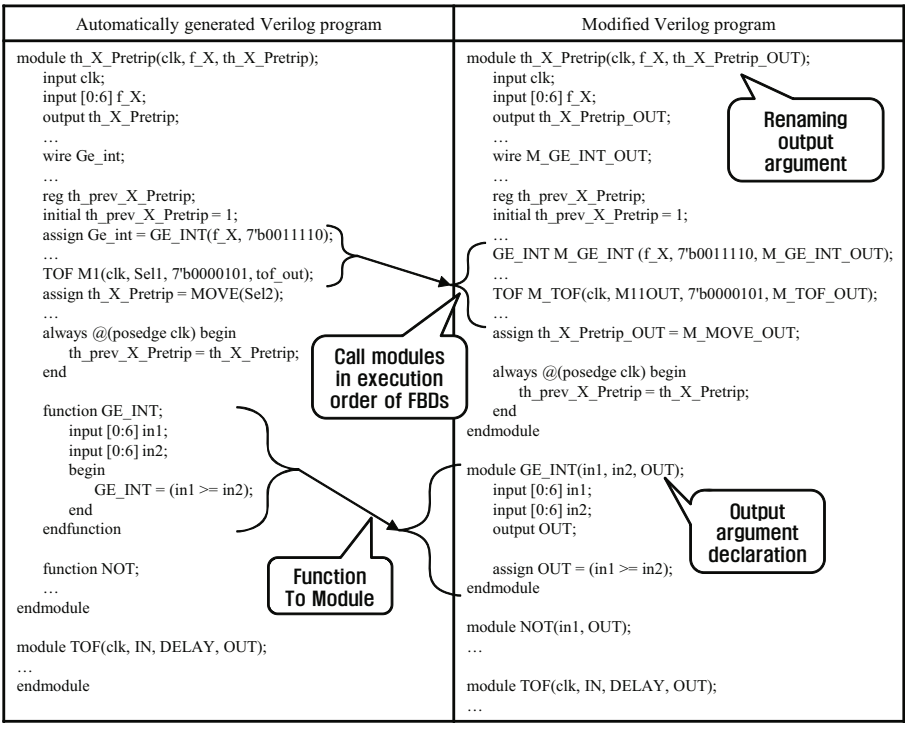

Fig. 5. The automatically generated Verilog program from *th_X_Pretrip* FBDs and its modified one

translated functions into modules. Next we changed function call to module call. The order of module call must follow the execution order of FBDs. Fig. 5 shows difference between the two Verilog programs.

4.2 Implementation of ANSI-C Program

We implemented ANSI-C program which has same behavior with *th_X_Pretrip* logic (Fig. 6). The program includes input value generation, synchronization with the Verilog program and equivalence checking property. The implemented ANSI-C program works according to the following steps:

Step 1 generate input value nondeterministically, *f_X = nondet_int()*, and synchronize inputs with Verilog program using *set_input()* statement.
Step 2 update conditions, *Cond_{a,b,c}_1* (corresponding FB from 2.11 to 2.15), and a output signal, *th_X_Pretrip_OUT*.
Step 3 count time unit (corresponding FB 2.16), *timer*, where the input variable is over the limitation named *k_Pretrip_Setpoint*.
Step 4 control state of the program, *state*, by checking the conditions.
Step 5 check equivalence of output of ANSI-C and Verilog programs using *assert(th_X_Pretrip.th_X_Pretrip_OUT == th_X_Pretrip_OUT)* statement and make a transition of the Verilog program using *next_timeframe()* function.

One loop of for statement means one transition of the Verilog program, because the for loop statement has one *next_timeframe()* function. We, therefore, could check equivalence between output of Verilog and variable of C program every transitions. The *assert(th_X_Pretrip.th_X_Pretrip_OUT == th_X_Pretrip_OUT)* statement means that the checking will stop if the output of Verilog program is not same with the variable of ANSI-C program. If the condition is not satified, then the HW-CBMC will make two counterexamples of ANSI-C and Verilog program.

4.3 Euqivalence Checking

In order to verify equivalence between the Verilog and ANSI-C programs using HW-CBMC, we executed following statement in Visual Studio command prompt:

```
>hw-cbcm.exe th_X_Pretrip.v th_X_Pretrip.c --module Pretrip
--bound 20
```

The *th_X_Pretrip.v* is the modified Verilog program of the automatically generated Verilog from FBDs. The *th_X_Pretrip.c* is the ANSI-C program which is implemented. The main module of the Verilog program is declared using the option *module*. The option *bound* specifies the number of times the transition relation of the module *Pretrip*. HW-CBMC result shows "VERIFICATION SUCCESSFUL" messssage which means that the Verilog and ANSI-C progmrams

```
1  ...
2  struct module_Pretrip {
3    unsigned int f_X;
4    unsigned int th_X_Pretrip_OUT;
5  };//definition of the variables that holds the value of the Verilog module
6  extern struct module_Pretrip th_X_Pretrip;
7  int main()
8  {
9    unsigned int f_X=0;
10   … //variable declaration
11
12   for(cycle=0; cycle<bound; cycle++)
13   {
14     //synchronizing Inputs
15     f_X = nondet_int(); th_X_Pretrip.f_X = f_X; set_inputs();
16
17     Cond_a_1 = (f_X >= k_Pretrip_Setpoint);
18     Cond_b_1 = ((f_X >= k_Pretrip_Setpoint) && (timer == 5));
19     Cond_c_1 = (f_X <= k_Pretrip_Setpoint - k_X_Pretrip_Hys);
20     th_X_Pretrip_OUT = (state==0 && Cond_a_1)?th_Prev_X_Pretrip:
21       (state==0 && !Cond_a_1)?th_Prev_X_Pretrip:
22       (state==1 && !Cond_a_1 && !Cond_b_1)?th_Prev_X_Pretrip:
23       (state==1 && Cond_a_1 && !Cond_b_1)?th_Prev_X_Pretrip:
24       (state==1 && Cond_b_1)?0:
25       (state==2 && Cond_c_1)?1:th_Prev_X_Pretrip;
26     if(f_X >= k_Pretrip_Setpoint) //count the time unit
27       if(timer == 5) timer == 5;
28       else timer++;
29     else
30       timer=0;
31     //assertion statement
32     assert(th_X_Pretrip.th_X_Pretrip_OUT == th_X_Pretrip_OUT);
33
34     th_Prev_X_Pretrip = th_X_Pretrip_OUT;
35     switch(state) //control a state by conditions
36     {
37       case 0:
38         if(Cond_a_1) state = 1;
39         else state = 0;
40       break;
41       case 1:
42         …
43     }
44     next_timeframe();
45   }
46 }
```

Fig. 6. Implemented ANSI-C program

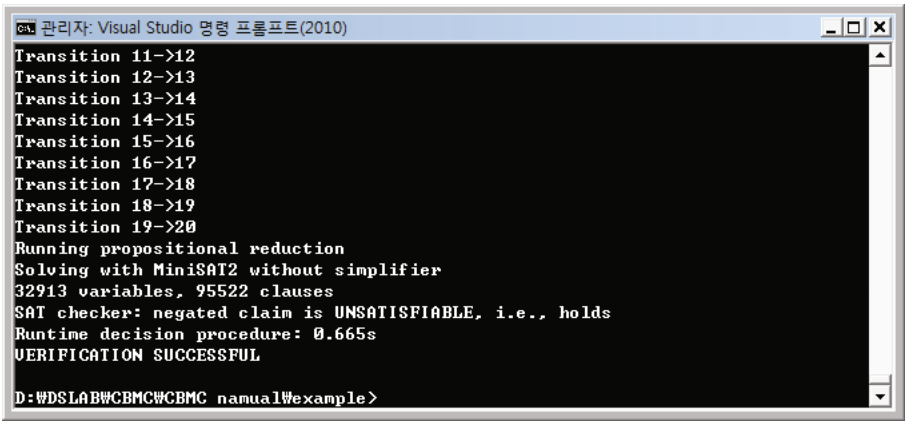

Fig. 7. A screen dump of equivalence checking result between *th_X_Pretrip.v* and *th_X_Pretrip.c* program

produced same output against the same input generated randomly. Fig. 7 shows a screen dump of the equivalence checking result.

As a result of this case study, we concluded that the HW-CBMC is effective for our proposal. Ahthough there is gap between automatically generated Verilog program and an input program of the HW-CBMC, we could check equivalence between both if the Verilog program modified successfully.

5 Conclusion

In this paper, we have proposed equivalence checking approach between design written in FBDs and its implementation program written in ANSI-C using HW-CBMC. The FBDs should be translated into Verilog language in order to make the FBDs into an input of the HW-CBMC. The automatically translated Verilog program, however, was not exactly appropriate to be the input. We modefied some features of the Verilog program, and made it into the input of HW-CBMC. As a result of the case study, the equivalence checking between FBDs and ANSI-C programs using HW-CBMC is effective.

We are planning to verify equivalence between FBDs and its automatically generated ANSI-C program by pSET. We are also planning to modify translation rules from FBDs to Verilog in order to make automatically generated Verilog program appropriate to the HW-CBMC and develop a translator for automation of this process.

Acknowledgments. 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 the KETEP

(Korea Institute of Energy Technology Evaluation And Planning)"(KETEP-2010-T1001-01038), the Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2010-0002566) and the IT R&D Program of MKE/KEIT [10035708, "The Development of CPS(Cyber-Physical Systems) Core Technologies for High Confidential Autonomic Control Software"].

References

1. Korea Nuclear Instrumentation & Control System R&D Conter, http://www.knics.re.kr/
2. Cho, S., Koo, K., You, B., Kim, T.-W., Shim, T., Lee, J.S.: Development of the loader software for PLC programming. In: Proceedings of Conference of the Institute of Electronics Engineerers of Korea, vol. 30(1), pp. 595–960 (2007)
3. Hoare, T.: The Verifying Compiler: A Grand Challenge for Computing Research. Journal of the ACM 50, 63–69 (2003)
4. RETRANS, Institue for Safety Technology (ISTec), http://www.istec.grs.de/en/produkte/leittechnik/retrans.html?pe_id=54
5. Clarke, E., Kroening, D.: Hardware verification using ANSI-C programs as a reference. In: Proceedings of the 2003 Asia and South Pacific Design Automation Conference, pp. 308–311 (2003)
6. Yoo, J., Cha, S., Jee, E.: Verification of PLC programs written in FBD with VIS. Nuclear Engineering and Technology 41(1), 79–90 (2009)
7. IEEE: IEEE standard hardware dexcription language based on the Verilog hardware description language. (IEEE Std. 1364-2001) (2001)
8. Bombieri, N., Fummi, F., Pravadelli, G., Marques-Silva, J.: Towards Equivalence Checking Between TLM and RTL Models. In: 5th IEEE/ACM International Conference on Formal Methods and Models for Codesign, MEMOCODE 2007, pp. 113–122 (2007)
9. Sangiovanni-Vincentelli, A., Aziz, A., Cheng, S.-T., Edwards, S., Khatri, S., Kukimoto, Y., Qadeer, S., Shiple, T.R., Swamy, G., Hachtel, G.D., Somenzi, F., Pardo, A., Ranjan, R.K., Brayton, R.K.: VIS: A System for Verification and Synthesis. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 428–432. Springer, Heidelberg (1996)
10. McMillan, K.L.: Symbolic Model Checking. Kluwer Academic Publishers, Dordrecht (1993)
11. IEC (International standard for programmable controllers): Programming languages 61131- Part 3 (1993)
12. Jee, E., Jeon, S., Cha, S., Koh, K., Yoo, J., Park, G., Seong, P.: FBDVerifier: Interactive and Visual Analysis of Counterexample in Formal Verification of Function Block Diagram. Journal of Research and Practice in Information Technology 42(3), 255–272 (2010)