# A Correctness Verification Technique for Commercial FPGA Synthesis Tools

Eui-Sub Kim [a], Junbeom Yoo [a*], Jong-Gyun Choi [b], Jang-Yeol Kim [b], Jang-Soo Lee [b]
[a]*Department of Computer Science and Engineering, Konkuk University, Seoul, 143-701, Korea*
[b]*Korea Atomic Energy Research Institute, Deadeok-daero 989-111, Yuseong Daejeon, Korea*
[*]*Corresponding author: jbyoo@konkuk.ac.kr*

## 1. Introduction

The PLCs (Programmable Logic Controller) have been widely used to implement the safety-critical system such as RPSs (Reactor Protection System) in Korean nuclear power plants. Recently, there have been attempted to implement the software in RPSs by FPGAs (Filed-Programmable Gate Array) [1][2], due to the increasing maintenance cost of PLCs and the higher performance of FPGAs.

The FPGAs are typically modeled with HDLs (Hardware Description Languages) such as Verilog and VHDL by software designers manually, and then subsequently synthesized into gate-level design and physical layout by software synthesis tools of FPGA vendors (e.g., 'Synopsis *Synplify Pro*' [3] and 'Cadence *Encounter RTL Compiler*' [4]). Once the FPGA designers designs Verilog programs, the commercial synthesis tools automatically translate the Verilog programs into EDIF programs so that the designers can have largely focused on HDL designs for correctness of functionality.

Nuclear regulation authorities, however, require more considerate demonstration of the correctness and safety of mechanical synthesis processes of FPGA synthesis tools, even if the FPGA industry have acknowledged them empirically as correct and safe processes and tools. In order to assure of the safety, the industry standards for the safety of electronic/electrical devices, such as IEC 61508 [5] and IEC 60880 [6], recommend using the formal verification technique. There are several formal verification tools (i.e., 'FormalPro' [7], 'Conformal' [8], 'Formality' [9] and so on) to verify the correctness of translation from Verilog into EDIF programs, but it is too expensive to use and hard to apply them to the works of 3rd-party developers.

This paper proposes a formal verification technique which can contribute to the correctness demonstration in part. It formally checks the behavioral equivalence [10] between Verilog and subsequently synthesized Netlist with the VIS verification system [11]. A Netlist is an intermediate output of FPGA synthesis process, and EDIF [12] is used as a standard format of Netlists. If the formal verification succeeds, then we can assure that the synthesis process from Verilog into Netlist worked correctly at least for the Verilog used.

In order to support the formal verification, we developed the mechanical translator 'EDIFtoBLIF-MV,' which translates EDIF into BLIF-MV [13] as an input front-end of VIS system, while preserving their behavior equivalence. It consists of three-steps – Parsing, Pro-processing and Translation. On other hands, the translation from Verilog to BLIF-MV is straightforward because the VIS provides an in-house translator 'vl2mv' [14], which translates Verilog into BLIF-MV automatically.

We performed the case study with an example of a preliminary version of RPS [15] in a Korean nuclear power plant in order to provide the efficiency of the proposed formal verification technique and implemented translator. It uses the 'Actel Libero IDE' [16] (internally, 'Synopsys Synplify Pro' [3]) to synthesize Netlist from the Verilog program, and also uses the 'EDIFtoBLIF-MV' to translate Netlist into BLIF-MV. The VIS verification system is then used to prove the behavioral equivalence.

This paper is organized as follows: Section 2 provides background information. Section 3 explains the developed tool, which translates EDIF to BLIF-MV. A case study with Verilog examples of a Korean nuclear power plant is presented in Section 4 and Section 5 concludes the paper and provides remarks on future research extension.

## 2. Background

### *2.1 An FPGA Development Process*

Fig.1 depicts a whole process of FPGA development [17]. Software requirements are analyzed and refined in requirements analysis and design phases, similar to the conventional software development. The process provides no standard form of requirements and design specifications. In HDL coding phase, we need to program the designs with HDLs such as Verilog or VHDL, manually. Some FPGA vendors provide own high-level design tools [18]. These tools use flow-charts, state machines or block diagrams to model design specification graphically, and generate HDL codes mechanically.

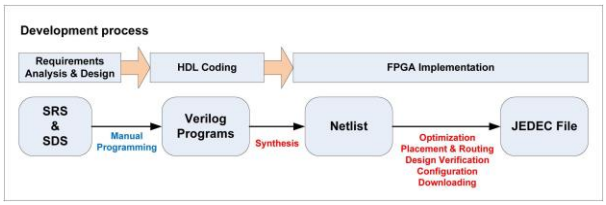


Fig. 1. An FPGA development process

After programming Verilog (or VHDL) programs, an FPGA is implemented mechanically thorough several steps, such as gate-level synthesis, optimization, placement & routing, design verification, configuration and downloading. Software synthesis tools provided by FPGA vendors such as 'Xilinx ISE Design Suite' [19], 'Altera Quartus II' [20] and 'Actel Libero IDE' [16] support all these steps seamlessly and mechanically. They also provide systematic verification and simulation facilities for each synthesis step.

### *2.2 Hardware Description Language*

An HDL (Hardware Description Language) is a specialized computer language for describing the design and operation of ICs (Integrated Circuits). It is a C-like programming language, and easy to learn and use. It enables HDL designers to design various levels of modeling, such as gate-level, data-flow and behavioral-level modeling, in combination. It can also make FPGA designers focus on functional verification at the early/beginning step of FPGA development.

Verilog [21] and VHDL [22] are the widely used HDLs in industry. A number of electronic vendors and EDA (Electronic Design Automation) tools use the HDLs to synthesize gate level design. HDLs are also widely used for logic verification as an input front-end of formal verification and analysis tools such as SMV [23], VIS [11] and HW-CBMC [24].

### *2.3 EDIF*

EDIF (Electronic Design Interface Format) [12] is a vendor neutral format in which to store Netlists and schematics. It was one of the first attempts to establish a neutral data exchange format for the EDA industries. The latest version of EDIF is 4.0.0, but most FPGA vendors still have used the version 2.0.0 which was first approved as the standard ANSI/EIA-548-1998. Nevertheless the effort for the neutral data exchange, FPGA vendors keep modifying the EDIF format slightly and appropriately for their own tools. The EDIFs of various FPGA vendors are now not compatible with each other. This paper uses the EDIF of 'Actel Libero IDE.'

### *2.4 VIS Verification System*

The VIS [11] is a verification system integrating formal verifications, simulation and synthesis of finite states hardware systems. It uses Verilog as an input front-end and supports fair CTL (Computational Tree Logic) model checking [25], [26], language emptiness checking, combinational equivalence checking, sequential equivalence checking, cycle-based simulation and hierarchical synthesis. It provides 'vl2mv' tool [14], which translates a subset of Verilog into an intermediate format BLIF-MV.

## 3. EDIFtoBLIF-MV

The translation process of 'EDIFtoBLIF-MV' consists of three-step as blocked in Fig.2. It first reads an EDIF file and precedes the three-step and then saves a BLIF-MV file. First, it parses the inputted EDIF file text by text and then constructs the internal data structure. Second, it proceeds pre-processing, which delete unnecessary information from the internal data structures. Finally, it translates the constructed data structure into the BLIF-MV with translation rules. The three-step is follow:

- Step 1. (Text Parsing) The 'EDIFtoBLIF-MV' parses the inputted EDIF text by text and then constructs the internal data structure. The inputted file format should follow the EDIF international industry standard version 2.0.0 [12] and not include any other syntactic errors.
- Step 2. (Pre-Processing) Before translating the internal data structure into the EDIF, it processes pre-processing, which delete unnecessary information. For example, all ports of *clk* are ignored in BLIF-MV, and we need to delete these information. *VCC* cell and *GND* cell have a fixed value such as 1 and 0, respectively. Thus, all ports should have default value 0 and 1, respectively.
- Step 3. (Translation) The translation rules from EDIF to BLIF-MV were researched by our previous work [27]. While we translate EDIF to BLIF-MV with previous research, however, we found some of rules are not proper. Thus, we fixed and improved them to apply 'EDIFtoBLIF-MV.' We are now planning to define the translation rules more formally and generally on the basis of [27] in order to deal with all categories of combinational cells, which the 'Actel Libero IDE' used.

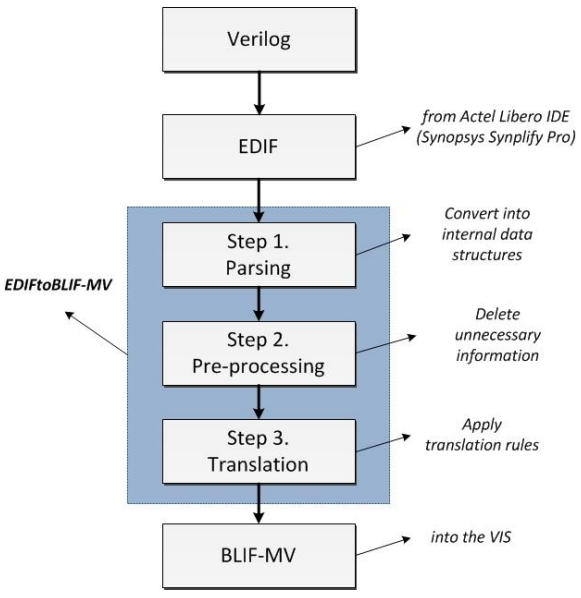


Fig. 2. The Overall Process of Translation from EDIF to BLIF-MV

The 'EDIFtoBLIF-MV' depicted in Fig.3 implemented the translation process from EDIF to BLIF-MV, described in Fig.2. As other translators and compilers, it has simple GUI to read an input file - EDIF and to store the translated output file - BLIF-MV. The console at the bottom shows the translation process in steps, i.e., EDIF open → Parsing → Pre-processing → Translation → BLIF-MV Saving.

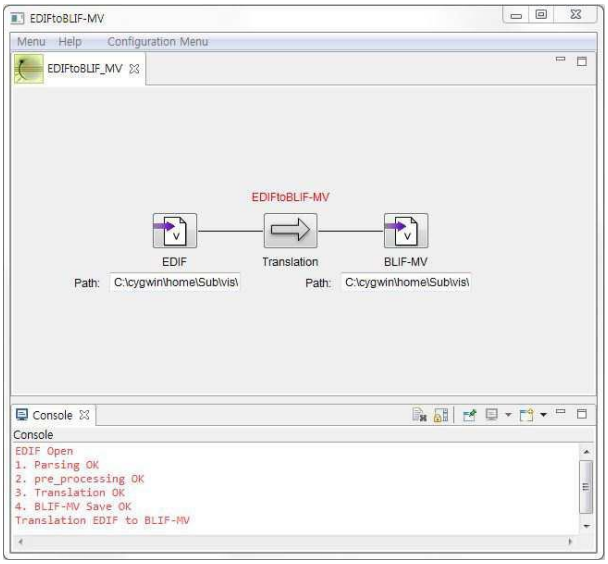


Fig. 3. A screen-dump of 'EDIFtoBLIF-MV'

## 4. Case Study

This paper performed a case study with an example of the KNICS APR-1400 RPS BP (Bistable Processor) [15] in Korea. The BP reads 18 sensor values from a nuclear reactor and decides to generate trip/pre-trip signals out to shutdown the reactor immediately, if any value is out of safe range. This case study used two examples of the 18 logics, such as 'FIX-RISING' and 'FIX-FALLING.' The case study is aiming for demonstrating correctness of the 3rd-party synthesis tools at least for the two logics.

We used two Verilog files translated from FBDs (Function Block Diagram) with 'FBDtoVerilog,' which we intend to support the platform change from PLC-based (FBD) to FPGA-based (Verilog). It will offer the possibility for designer to change of platform more seamlessly. First, we obtained Netlist from the Verilog using 'Symplify Pro' automatic synthesis tool which included in 'Actel Libero IDE.' Next, we obtained 1st BLIF-MV from the same Verilog using 'vl2mv,' which is in-house tool provided by VIS system. Lastly, we obtained 2nd BLIF-MV from Netlist (EDIF format), which synthesize by 3rd-party synthesis tool, using 'EDIFtoBLIF-MV.'

We now can perform the VIS equivalence checking between the Verilog program and the EDIF program which has just been transformed from the Verilog. If the verification succeeds, we can assure that the synthesis process and tool (i.e., 'Actel Libero IDE') from Verilog into EDIF worked correctly for the Verilog program. If it fails, it means that the synthesis tool has some problems to be analyzed in depth. Of course, we assume that the proposed technique was thoroughly refined and stabilized.

Fig.4 shows the results of the VIS verification. After a sequence of VIS commands, the VIS produced a successful result - "Networks are sequentially equivalent." for the two Verilog programs. Therefore, we can assure that the FPGA synthesis tool - 'Actel Libero IDE' works correctly for those Verilog programs of 'Fixed set-point rising trip logic' and 'Fixed set-point falling trip logic.'

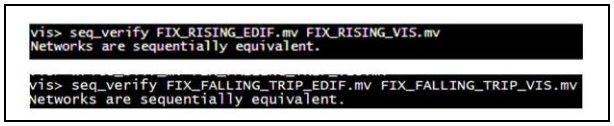


Fig. 4. Result of the VIS verification (excerpted)

Fig.5 shows the hypothetical failure case. We also performed a case study with simple seeded errors (like toggling 1 and 0) in order to check the correct functioning of the implemented 'EDIFtoBLIF-MV.' The VIS resulted in a failure and also provided a sequence of input variables, resulting in the failed result (i.e., producing not equivalent outputs), called a counterexample. Counterexamples of the VIS verification can help analysts confirm whether the synthesis works well or not.

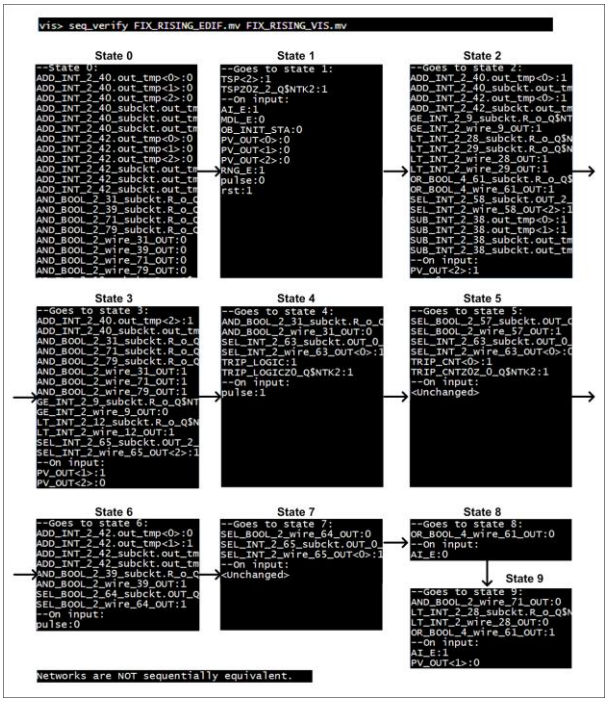


Fig. 5. Results of the VIS verification (excerpted and edited)

## 5. Conclusion and Future Work

This paper proposes a formal verification technique which can contribute to the correctness demonstration of commercial FPGA synthesis processes and tools in part. It formally checks the behavioral equivalence between Verilog and subsequently synthesized Netlist with the VIS verification system. If the formal verification succeeds, then we can assure that the synthesis process from Verilog into Netlist worked correctly at least for the Verilog program.

In order to support the formal verification, we developed the mechanical translator 'EDIFtoBLIF-MV,' which translate EDIF into BLIF-MV, while preserving their behavior equivalence. The translation from EDIF into BLIF-MV consists of three steps – Parsing, Pro-processing and Translation. We performed the case study with Verilog programs designed for a digital I&C system in Korea. It shows that the verification technique can be used positively as a means of demonstrating the correctness of the FPGA synthesis tools of 3rd-party developers.

We are currently focusing on stabilizing the transformation process including the translation rules and the translator. We are also planning to perform several full-scale case studies and compare the correctness verification results and efficiency with the commercial solution.

## Acknowledgements

This research was supported, in part, by a grant from the Korea Ministry of Science, ICT and Future Planning, under the development of the integrated framework of I&C dependability assessment, monitoring, and response for nuclear facilities. It was also supported, in part, by a grant from the Korea Atomic Energy Research Institute, under the development of the core software technologies of the integrated development environment for FPGA-based controllers.

## REFERENCES

[1] J. Yoo, J.-H. Lee and J.-S. Lee, A RESEARCH ON SEAMLESS PLATFORM CHANGE OF REACTOR PROTECTION SYSTEM FROM PLC TO FPGA, Nuclear Engineering and Technology, no.4, p.477-488, 2013.
[2] D.-H. Lee, E.-S. Kim, J. Yoo, J.-S. Lee, and J.-G. Choi, FBDtoVerilog 2.0: An automatic translation of FBD into Verilog to develop FPGA, International Conference on Information Science & Applications 2014 (ICISA2014), pp. 447–450, 2014.
[3] Synopsys, Synplify Pro, http://www.synopsys.com/Tools/Implementation/FPGAImplementation/FPGASynthesis/Pages/SynplifyPro.aspx
[4] Cadence Encounter RTL Compiler http://www.cadence.com/products/ld/rtl_compiler/Pages/default.aspx
[5] IEC, IEC 61508: Functional safety of electrical, electronic and programmable electronic (E/E/PE) safety-related systems, International Standard, Second Edition, International Electrotechnical Commission, Geneva, vol. 1, 2003.
[6] IEC, IEC 60880: Nuclear power plants - Instrumentation and control systems - important to safety Software aspects for computer-based systems performing category A functions, 2006.
[7] Mentor Graphics, FormalPro, http://www.mentor.com/products/fv/formalpro/
[8] Cadence, Conformal, http://www.cadence.com/products/ld/equivalence_checker/pages/default.aspx
[9] Synopsys, Formality, http://www.synopsys.com/Tools/Verification/FormalEquivalence/Pages/Formality.aspx
[10] S.-Y. Huang and K.-T. Cheng, Fromal Equivalence Checking and Debugging, Kliwer Academic Publishers, 1998.
[11] R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S.-T. Cheng, S. A. Edwards, S. P. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. K. Ranjan, S. Sarwary, T. R. Shiple, G. Swamy, and T. Villa, VIS : A system for verification and synthesis, in the Eighth International Conference on Computer Aided Verification, CAV '96, pp. 428–432, 1996.
[12] Electronic Industries Association, Electronic design interchange format (EDIF), EIA-548, Version 2.0.0, 1998.
[13] R. K. Brayton, BLIF-MV: An interchange format for design verification and synthesis, University of California, Tech. Rep., 1991.
[14] S. T. Cheng, G. York, R. K. Brayton, VL2MV: A Compiler from Verilog to BLIF-MV, HSIS Distribution, 1993.
[15] Korea Atomic Energy Research Institute (KAERI), Software Design Specification for Reactor Protection System, KNICS-RPS-SD231 Rev.02, 2006.
[16] Actel, Actel Libero IDE, http://www.actel.com/products/software/
[17] S. Brown and J. Rose, FPGA and CPLD architectures: A tutorial, IEEE Software, vol. 13, no. 2, pp. 42–57, 1996.
[18] Mentor Graphics Corporation, HDL Designer SeriesTM User Manual, Tech. Rep., 2008.
[19] Xilinx, Xilinx ise design suite, http://www.xilinx.com/products/
[20] Altera, Altera quartus ii, http://www.altera.com/products/software/
[21] S. Palnitkar, Verilog HDL: A Guide to Digital Design and Synthesis. McGraw-Hill Inc., 1997.
[22] Z. Navabi, VHDL: Analysis and Modeling of Digital Systems. McGraw-Hill Inc., 1997.
[23] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "NuSMV: A new symbolic model verifier," in 11th International Conference on Computer Aided Verification (CAV '99), pp. 495–499, 1999.
[24] E. Clarke and D. Kroening, Hardware verification using ansi-c programs as a reference, in 2003 Asia and South Pacific Design Automation Conference, pp. 308–311, 2003.
[25] 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.
[26] E. M. Clarke, O. Grumberg, and D. A. Peled, Model Checking, MIT Press, 1999.
[27] J.-H. Lee, Automatic translation for equivalence checking between verilog and edif netlist with vis, Master's thesis, Konkuk University, 2013.