Ver2Smv - A Tool for Automatic Verilog to SMV Translation for Verifying Digital Circuits

Mishal Minhas and Osman Hasan
School of Electrical Engineering and Computer Science
National University of Sciences and Technology (NUST)
Islamabad, Pakistan
{mminhas.msee15seecs,osman.hasan}@seecs.nust.edu.pk

Kashif Saghar
Software Quality Assurance Department
Centers of Excellence in Sciences and Applied Technologies, Islamabad, Pakistan
Kashif.saghar@gmail.com

***Abstract*—Verification of today's complex digital circuit designs is of critical concern for hardware designers. A tremendous amount of effort and computing resources are spent to assure if the developed design is correct or not. Simulation is by far the most extensively used method for this purpose. However, it does not provide 100% coverage of the possible test patterns. Formal verification tends to overcome these limitations. However, digital circuits are usually described in Verilog/VHDL and these descriptions need to be manually expressed in a formal language for their formal verification. This is a cumbersome task and thus limits the usage of formal verification in the industrial setting. This paper presents a step towards overcoming these problems by presenting a tool, Ver2Smv, for automatically translating RTL Verilog to the SMV language, i.e., a language supported by the NuXmv model checker. Besides the Verilog description, the user provides a VCD format file (random stimulus data) through a user friendly Python interface. This file helps in the automatic specification generation as assertions and relieves the user from writing them manually. These assertions are verified using the nuXmv model checker for the corresponding SMV model.. The workflow of Ver2Smv tool is illustrated and tested on some commonly used sequential circuit designs.**

Keywords—Formal Verification, Model Checking, NuXmv, SMV, Verilog.

I. Introduction

Digital designs are being extensively used in many safety-critical domains, making their verification an utmost concern. Nowadays, circuits are composed of millions of gates, which make their simulation more and more difficult due to the limited computational resources. The chances of undetected bugs leading to flawed designs are quite high due to the incompleteness of simulation based verification. However, formal methods [1] have become a highly acknowledged part of the digital design process due to their effectiveness and completeness as compared to simulation. Primarily, formal methods are computer-based mathematical methods that allow verification using mathematical analysis and logical reasoning. Therefore, the results produced are assured to be correct like that of a mathematical theorem. Apart from ensuring safety, formal verification reduces the overall cost of the design process, saving the resources and overcoming the limitations of simulation techniques. Theorem proving and model checking are the two well-established methods for formal verification [1]. Theorem provers verify the hardware system behavior represented in mathematical logic and the correctness is established as a logical proof. To perform this analysis, several proof checkers for higher- order logic, such as HOL [2] and ACL2 [3], are used. However, translating the system behavior and implementation from HDL to formal mathematical model is not automatic and takes a lot of human effort making it unsuitable for industrial applicability. In contrast, model checking [4] is an automatic formal verification method, which proves the correctness of a system exhaustively using FSMs and allows verifying temporal/sequential and invariant properties.

Digital circuits are mostly described in Verilog/VHDL containing the implementation/structural information along with the properties to be checked. These properties tell us how the design should behave. For an efficient verification, specifications must be well written. Assertions [5] are gradually becoming the way of specifying properties. Instead of the input-output based simulation, assertions are capable of expressing the behavior of a design more appropriately. Writing quality assertions is still a manual task, which requires a lot of effort and time. This paper explores a solution by generating Verilog RTL assertions automatically.

Among the various levels of Verilog abstractions, verification is commonly done on Gate (Boolean)-Level because majority of industrial hardware verification tools employ bit-level techniques [6]. However, digital circuit designs often have registers and memory structures that cannot be expressed at the bit-level, whereas, the RTL level contains word-level information [6]. In order to avoid the loss of structural information, formal verification needs to be done directly on the RTL level. Recently, several approaches have been developed to verify RTL level designs using model checking tools that show the advantages of using word- level information for property checking. The goal of this paper is to apply verification at the RTL level so that the memory information is retained.

In this paper, an automatic tool is proposed that is suitable for industrial verification of digital circuits. The proposed tool is based on model checking in order to minimize the overall user involvement.

A software tool Ver2Smv (Verilog to SMV Translator) is developed that translates the HDL code into SMV format supported by the NuXmv model checker. Moreover, besides translation, this tool also provides automatic property generation allowing the users to verify them without having to develop any formal specifications manually. Ver2Smv provides a user-friendly Python based application that allows automatic generation of properties and the HDL to SMV translation. The properties are generated as Verilog RTL assertions using the GoldMine engine [7]. The translation is done using a converter verilog2smv tool [8] built on Yosys [9]. Once the two inputs i.e. Verilog file and VCD file are provided by the user, the assertions are generated and merged with the input Verilog file which then gets automatically translated into a SMV model and invariant specifications. Finally, the SMV file is verified using nuXmv, which provides the final verification results.

The paper is organized as: Section II presents a review of existing translators and related tools. The proposed methodology of Ver2Smv is described in Section III. The implementation details with a sequential circuit example are presented in Section IV. Finally, Section V concludes the paper.

II. RELATED WORK

Simulation allows us to test complex concurrent systems, for instance, a large sequential design, automatically. However, it is not exhaustive, which means it largely depends on the technical skills of the person writing the test cases. Thus, it is a lengthy process and does not usually achieve 100% coverage of the system behavior under consideration.

This paper aims at using assertions for property specification. The assertion based verification technique uses assertions to express the required behavior of the design. Compared to the traditional simulation, this assertion based verification approach is quite fast and effective in debugging and verifying a design. However, the assertions specifying the behavior must be chosen such that they expose as many bugs as possible. Writing such quality assertions requires manual guidance and a lot of effort and time without any guarantee of complete behavior coverage. Vasudevan et al. [7] present an automatic assertion generator engine, GoldMine that solves this issue by applying data mining and static analysis to generate optimized Verilog RTL assertions with a wide scope of behavior.

Formal methods are known to perform a rigorous analysis and thus can overcome the limitations of simulation. Model checking is a frequently used formal method for hardware verification [18]. The main idea is to capture the behavior of the given circuit as a finite state machine. It comprises of the state variables of the model, and from the combinational logic, we can deduce the rules that control transitions between states. After creating such state machine model, we can verify properties. The automatic and exhaustive verification of temporal properties for finite state machines is the main strength of model checking.

There are many tools developed for converting Verilog designs to formal models. For example, V3 [10] is a framework for research on hardware verification and debugging. Verilog designs are read as input to V3 and word-level BTOR designs are produced using a Verilog frontend QuteRTL [8]. ABC (Berkeley ABC) [11] is an academic tool for logic synthesis and verification for both combinational and sequential circuits. It converts the Verilog designs into gate-level designs for verification with its Verilog frontend [8]. However, it cannot generate word level model checking problems and does not focus on advanced combinational and sequential circuits as its basic data structures with limited synthesis and verification options. Reveal [12] is also a formal verification tool for Verilog designs. The main principle behind this tool is to perform checks of safety properties on hardware designs. It reads the high-level specification and a detailed implementation of the Verilog designs and checks their functional equivalence. Cadence-SMV [13] uses the Verilog hardware description language to express the system model. It uses the Verilog design as input to generate Boolean-level SMV design. However, it does not provide word-level information in the SMV design. EBMC [14] takes Verilog designs with assertions and returns the Boolean-level model-checking (MC) problem in SMV format using BMC and/or k-induction. It can also output Boolean-level MC problem in SMV format. Properties in EBMC can be given in Linear Temporal Logic (LTL) or a fragment of System Verilog Assertions. Among all other options, EBMC seems the one that can handle memories without abstracting them.

Yosys [15] is an Open-Source Verilog synthesis tool that converts Verilog designs with simple assertions to BTOR format. BTOR format has certain limitations, e.g., it initializes registers to a zero value and the memories stay uninitialized. Similarly, it does not have support for multi- dimensional memories. Verilog2smv [16] is an open-source tool for Verilog-to-SMV conversion. The main goal of this tool is to formally verify directly RTL models using MC techniques provided by the nuXmv model checker. It handles memories efficiently and has been found to be better than EBMC [8].

The SMV model checker uses Symbolic Model Checking: An approach to solve the state explosion problem by using boolean formulas to represent sets and relations between states and avoid ever having to construct the entire state graph [18]. The inherent state-space explosion problem of model checking is dealt using the bounded model checking technique [19]. NuXmv [20] is an extension of NuSMV that expresses transition systems using all the finite data types allowed by NuSMV (Booleans, enumerative, bounded integers), and also includes the support for bit-vectors, reals, integers, and (finite and infinite) arrays, with no restriction on the specification of the initial values of the variables. Therefore, the proposed methodology uses nuXmv, the word-level model checker for the formal verification of sequential circuit designs.

III. PROPOSED METHODOLOGY

The proposed methodology of Ver2Smv tool is shown in Fig.1. It consists of two main modules, i.e., Automatic Property Generation and Verilog to SMV translation. The first module uses the GoldMine engine. The verilog2smv convertor tool is used in the second module. Ver2Smv, developed in Python, integrates the two modules producing a nuXmv file that is analyzed and verified using the nuXmv model checker.

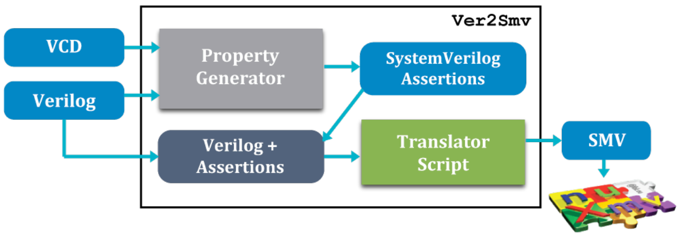


Fig. 1. Proposed Methodology

The modules of our methodology are explained below:

A. Automatic Property Generation

Ver2Smv takes two inputs, i.e., Verilog code and VCD file for random simulation data. The properties are specified using simple SystemVerilog assertions. The simulation data of VCD file is required for data mining done by GoldMine for generating assertions. It also uses a static analyzer composed of a Verilog parser and elaborator. A control-data flow graph (CDFG) is constructed by the parser from the input Verilog design. The top-level module, variable, clock, reset, and other important information is extracted from the CDFG by the elaborator. The data generator consists of a test bench generator, simulator, and data compressor [7]. The test bench generates an unconstrained random test bench for the top module in the Verilog design. The top module is simulated using the test bench by the simulator. Then, the data compressor removes duplicate examples from the data set. The data generated by the static analyzer and data generator is then used by the data miner to generate a set of assertions. The output produced by this module is a gmvh format file consisting of the Verilog RTL assertions. This module thus avoids the user intervention required for writing assertions. Fig.2 shows the internal working of this module.

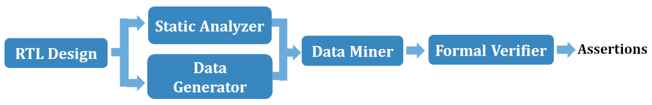


Fig. 2. Assertion Generation Steps

B. Verilog-to-SMV Translation

The second module is based on Yosys, which is a verilog2smv translator built on a Verilog synthesizer. It generates a MC problem at RTL in two formats, i.e, BTOR and nuXmv. It handles sequential circuits without losing any memory information.

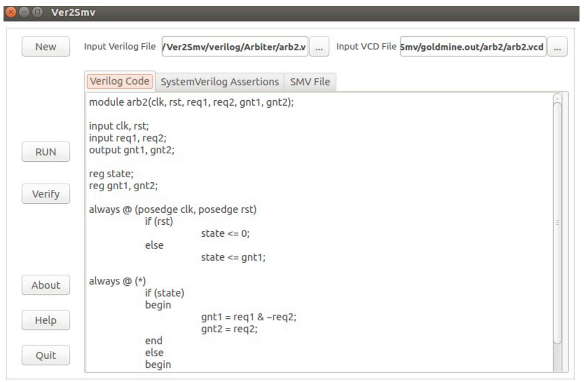
Fig. 3 shows the shell script of Yosys. It takes three parameters, the first parameter is the input design, the second parameter is the file name of BTOR output, and the third parameter is the name of top module in the design. BTOR format can only handle one-dimensional arrays and bit-vectors whereas nuXmv format handles finite and infinite arrays with no restriction on initial value specification of the variables. Therefore, we choose the nuXmv format as the target language for verification.

```
verilog2btor.sh fsm.v fsm.btor test
```

Fig. 3. Yosys Shell Script

IV. IMPLEMENTATION

As described in the previous section, the proposed method-ology works without any user involvement. The modules run just by a click of a button by the user and the required output is automatically generated. Ver2Smv contains all the required packages to run this tool smoothly so that the user does not have to download them separately. The GUI, shown in Fig. 4, opens by a simple python command run from the terminal.



```
module arb2(clk, rst, req1, req2, gnt1, gnt2);

input clk, rst;
input req1, req2;
output gnt1, gnt2;

reg state;
reg gnt1, gnt2;

always @ (posedge clk, posedge rst)
        if (rst)
                state <= 0;
        else
                state <= gnt1;

always @ (*)
        if (state)
        begin
                gnt1 = req1 & ~req2;
                gnt2 = req2;
        end
        else
        begin
```

Fig. 4. Ver2Smv User Interface

The next step is to open the Verilog and VCD files. The RUN button then initiates the process and the output file isbe displayed on the screen including the assertions and translated SMV file. For verification purposes, the SMV file is used in the nuXmv model checker and properties are verified automatically. The workflow is illustrated by using a simple example of an arbiter circuit. The Verilog code for the circuit has two outputs *g*nt1 and *g*nt2, has one reg *s*tate and four inputs *c*lk, *r*eset, *r*eq1, *r*eq2. The input-output relationship is shown in Fig.5. Once the VCD file is provided, the first module uses the simulation traces contained in it. The static analyzer uses behavioral analysis techniques to extract domain-specific information about the Verilog design and passes to the data miner as shown in Fig.2. A decision tree is formed by the data miner and produces a set of some candidate assertions (i) *((req1 == 0)* |-> *(gnt1 == 0))* (ii) *((req1 == 1 && req2 == 0)* |-> *(gnt1 == 1))* based on the data in the decision tree. For automatic property specification, Ver2Smv appends the generated assertions within the Verilog model, shown in Fig.5.

```
always @ (posedge clk, posedge rst)
  if (rst)
        state <= 0;
        state <= gnt1;  else
always @ (*)
  if (state)  begin
        gnt1 = req1 & ~req2;
        gnt2 = req2;   end
  else
  begin
        gnt1 = req1;
        gnt2 = req2 & ~req1;   end
  assert property ( !( req1 == 0 ) || ( gnt1 == 0 ) );
  assert property ( !( req1 == 1 && req2 == 0 ) || ( gnt1 == 1 ));
  assert property ( !( req2 == 0 ) || ( gnt2 == 0 ) );
  assert property ( !( req1 == 0 && req2 == 1 ) || ( gnt2 == 1 ));
endmodule
```

Fig. 5. Properties Specification in Verilog Design

Fig.6 shows the SMV file produced by the second module. The initial blocks of Verilog design are converted into INIT constraints. Memory and register assignments are translated into TRANS constraints. The memory information is retained as array declaration. INVARSPEC corresponds to the assertions in the Verilog design where all the *a*ssert commands are translated into INVARSPEC. This nuXmv file is analyzed using the model checking algorithms to get verification results.

```
MODULE main
IVAR
  "clk" : boolean;
  "req1" : boolean;
  "req2" : boolean;
  "rst" : boolean;
VAR
  "state" : boolean;
DEFINE
  __expr0 := word1("req1");
       .
       .
       .
  __expr85 := bool(0ub1_1);
  __expr86 := (__expr85 -> __expr84);
TRANS next("state") = __expr8
INVARSPEC  __expr22;
INVARSPEC  __expr44;
INVARSPEC  __expr64;
INVARSPEC  __expr86;
```

Fig. 6. NuXmv File

V. CONCLUSIONS

In this paper, we have presented an automatic tool Ver2Smv that is capable of translating sequential circuits written in a high-level description language Verilog to a model checker language SMV. The distinguishing feature of this tool is the automatic property generation and a user-friendly interface as compared to other existing tools. A user with no prior knowledge of assertions and formal model can easily use this tool, which otherwise writing manually is not very simple.

The paper describes the functioning of the tool using a simple example. We are experimenting with many other case studies as well including several benchmark circuits with high number of gates. It has been observed that the proposed Ver2Smv tool greatly facilitates the verification process by avoiding manual guidance required by the user.

REFERENCES

[1] Clarke, M. Edmund, M. W. Jeannette, *Formal Methods: State-of-the-art and future directions* ACM Computing Surveys, 28(4):626-643, 1996.

[2] Gordon, JC. Michael, T. F. Melham, *Introduction to HOL A theorem proving environment for higher order logic,* Cambridge University Press, 1993.

[3] Kaufmann, Matt, P. Manolios, J. S. Moore, *Computer-aided reasoning: ACL2 case studies. Vol. 4* Springer Science and Business Media, 2013.

[4] Clarke, M. Edmund, O. Grumberg, D. Peled, *Model checking,* MIT press, 1999.

[5] Dahan, Anat, et al *Combining system level modeling with assertion based verification* Quality of Electronic Design, Quality of Electronic Design, IEEE, 2005.

[6] R. Drechsler, *Using word-level information in formal hardware verification,* Automation and Remote Control, 65(6):963-977, 2004.

[7] S. Vasudevan, D. Sheridan, S. Patel, D. Tcheng, B. Tuohy, D. Johnson *Goldmine: Automatic assertion generation using data mining and static analysis,* Design, Automation and Test in Europe, European Design and Automation Association, pp. 626-629, 2010.

[8] A. Irfan, A. Cimatti, A. Griggio, M. Roveri, R. Sebastiani *Verilog2SMV: A tool for word-level verification,* Design, Automation & Test in Europe, IEEE, pp: 1156-1159, 2016.

[9] A. Irfan, C. Wolf, *Yosys Application Note 012: Converting Verilog to BTOR,* 2015.

[10] V3, https://github.com/chengyinwu/V3, 2017.

[11] R. Brayton, A. Mishchenko *ABC: An academic industrial-strength verification tool,* Computer Aided Verification, Springer, LNCS 6174, pp. 24-40, 2010.

[12] S.A. Zaher, M.H. Liffiton, K. A. Sakallah *Reveal: A formal verification tool for verilog designs,* Logic for Programming Artificial Intelligence and Reasoning, Springer, LNCS 5330, pp. 343-352. 2008.

[13] M.A. Abbas, S. Balakrishnan, and S. Tahar, *Modeling and verification of embedded systems using cadence SMV,* Electrical and Computer Engineering, Vol. 1. IEEE, 2000.

[14] http://www.cprover.org/ebmc/, 2017.

[15] http://www.clifford.at/yosys/, 2017.

[16] https://es-static.fbk.eu/tools/verilog2smv/, 2017.

[17] S.P. Martin, *An environment for the automatic verification of digital circuits*, MS thesis, Universitat Politecnica De Catalunya, Barcelonatech, 2011.

[18] Cimatti, Alessandro, Edmund, M, Clarke, Fausto Giunchiglia and Marco Rovera, *NuSMV: A new symbolic model checker,* Software Tools for Technology Transfer, 2(4): 410-425, Springer, 2000.

[19] Cimatti, Alessandro, Edmund, M, Clarke, Fausto Giunchiglia and Marco Rovera, *NuSMV: A symbolic model checker,* Computer Aided Verification. Springer, pp. 495-499, 2014.