V-HOLT Verifier - An Automatic Formal Verification Tool for Combinational Circuits

Nirmal Saeed, Ayesha Inam, Aisha Khan and Osman Hasan
School of Electrical Engineering and Computer Sciences
National University of Sciences and Technology
Islamabad, Pakistan
{08beenirmals,08beeainam,08beeaishak,osman.hasan}@seecs.nust.edu.pk

Abstract—**Formal verification using theorem proving ascertains 100% accuracy of digital circuit verification and is thus far more useful than simulation. However, most of the theorem proving based formal verification tools do not accept commonly used HDLs, like VHDL or Verilog, and require their users to manually conduct the verification, which is a step that involves rigorous mathematical analysis. It is due to these limitations that theorem proving based verification tools are not commonly used in the industry despite their ultimate accuracy guarantee. As a first step to overcome these limitations, we present an automatic verification tool, V-HOLT Verifier, for the verification of combinational circuits described in VHDL format. Besides the VHDL description, the user of the tool provides the property that needs to be verified using a user friendly JAVA based interface. The verification of the property is done using the HOL4 theorem prover, which is a widely used theorem proving tool based on higher-order logic. The translation to HOL4 compatible code and the generation of HOL4 verification script is automatically done and thus the user is not involved with these details, which makes V-HOLT Verifier quite user friendly. The final outcome is in the form of 'Goal Proved', if the property is verified, or an 'Error Trace' in case the failure. For illustration purposes, we tested some commonly combinational circuits including 4 x 1 MUX and a full adder.**

Keywords- Formal Verification, HOL, VHDL, Verification Tool

I. INTRODUCTION

Nowadays, digital designs are being increasingly used in safety-critical domains, such as medicine or transportation. Thus, faulty systems cannot only cause massive financial losses but in severe cases, precious lives may be lost as well. Examples of various disasters throughout history include the software bug in the cancer therapy machine Therac that led to three deaths and three severe cases of injuries between 1985 and 87. Similarly, the famous Pentium bug in 1994 resulted in the financial loss of about 500 million US$ to Intel due to system recalls. Another example can be the Mars Polar Lander. Its engines shut down prior to landing causing a damage of about 370 million US$ dollars in the year 1999. Keeping this problem in view, digital designs need to be thoroughly tested and verified before implementation, essentially because reliability of system during designing and integrating is vital to maintain.

The state-of-the-art digital circuit verification approach is simulation. Simulation generally works by analyzing the behavior of the given circuit under some test cases to deduce whether the system exhibits the required properties. Unfortunately, despite its obvious advantage of being a lot easier to use, simulation has a few major flaws. The simulation based analysis is prone to be inaccurate because it is practically impossible to test for all possible combinations of input scenarios given the input size and complexity of present-age digital circuits. For instance, in a 64-bit floating point division routine, there are 2^{128} combinations. Exhaustively simulating this number at the rate of 1 test/ms would take about 10^{25} years, which is obviously unrealistic and is thus not done in practice. As a result the bugs that might remain undetected can lead to design flaws which in turn may lead to major disasters. The amount of time involved in testing is another reason that could lead to inaccuracy in generated results.

Formal methods [1] have the capability to overcome the above mentioned limitation of simulation. They are essentially computer based mathematical analysis tools. Since the verification is done using mathematical principles and logical reasoning so the results are guaranteed to be correct just like a mathematical theorem. The use of computers and thus the associated bookkeeping facilitates to handle complex systems, which cannot be managed using paper-and-pencil proof methods. Model checking [2] is an automatic formal verification technique that allows us to prove temporal, or sequential, properties of hardware(See e.g. [3, 4, 5, 6, 7]). But Model checking has a severe limitation in terms of state-space explosion, i.e., as the design complexity grows the technique cannot handle it due to the limited computational and memory resources. In order to overcome the state-space explosion problem of Model checking, an alternative is to represent the given hardware system behavior in mathematical logic and establish its correctness as a logical proof. To perform such analysis, there are several automated proof checkers for first and higher-order logics available, e.g., HOL [8], PVS [9], and ACL2 [10]. Even though, theorem proving overcomes the state-space issues of model checking, it has its own limitations. Translating hardware system behavior from hardware description languages (HDL) like VHDL or Verilog to formal specifications is not automatic and verification engineers have to perform these translations manually. Similarly, proving the correctness of a contemporary hardware system is a rigorous process and requires a lot of user effort. Due to the above mentioned difficulties in formal verification usage and because of the fact that it needs expressive human guidance in almost all cases, it is unsuitable for application in industry.

As a first step to overcome the existing limitations of formal hardware verification, we propose an automatic formal verification tool for combinational circuits. The proposed solution is based on the theorem proving principles but does not involve user intervention in the verification process. This is done by developing a software tool, (VHDL to HOL Translator) V-HOLT Verifier, that will translate the digital design description provided in the VHDL language format to the syntax that is supported by HOL - a system designed to support interactive theorem proving in Higher-order Logic. Moreover, V-HOLT Verifier allows the user to enter the properties that need to be checked via a user-friendly JAVA based application and upon clicking a button these properties are automatically formally verified against the design given VHDL code. The VHDL to HOL translation is done using a VHDL to XML converter tool VSYML. The XML format is then into HOL format. The property writer is a part of the main User Interface that prompts the user to input the theorem or property in such a manner that it automatically becomes HOL compliant without the user even having to know the proper HOL syntax. Finally, once these two inputs are ready then they are merged together to form a HOL goal that is automatically given to HOL besides a set of HOL tactics and theorems that are guaranteed to prove the Boolean logic goal automatically. The final output of the HOL verification is shown to the user in the form of a 'Goal Proved', in case the property is verified, or 'Error trace', if the property fails so that the user can debug the system.

The rest of the paper is organized as follows: In Section II, we present a review of existing HDL to formal languages translators. The implementation details of V-HOLT Verifier are described in Section III. We present the verification example of a full adder using V-HOLT Verifier in Section IV. Finally, Section V concludes the paper.

II. Related Work

Automatic translation of HDLs to formal languages has been investigated by many researchers. In this section, we review some of the key contributions in this direction.

Formal Verification at Centaur Technology is done by translating RTL Verilog source codes to EMOD, which is a HDL with formal semantics [11]. Thereafter, equivalence checking and theorem proving techniques are used for the purposes of proving the functionality of the circuit using the ACL2 theorem prover. Since ACL2 is a first-order-logic theorem prover so this approach is limited to a small set of digital circuits. For example, the tool may not be used to verify floating point multiplication and division circuits. Another translator for translating a synthesizable subset of VHDL to the input format of ACL2 (a theorem prover) is proposed in[12]. No verification support is provided after the translation in this work. Moreover, again due to the limited expressiveness of ACL2, the domain of this approach is not very wide. In this paper, we propose a VHDL to HOL theorem prover supported code translator. HOL is a higher-order-logic theorem prover and thus is more expressive than ACL2 and this way the proposed work tends to overcome the limitations of the above mentioned ACL2 based approaches.

Drusinsky developed a specification based verification tool called Temporal Rover [13] for applications written in C, C++, Java, Verilog and VHDL. This tool is used to verify complex protocols and reactive systems where behavior is time dependent. It utilizes a Linear-Time Temporal Logic (LTL) and Metric Temporal Logic (MTL), with conventional Simulation or execution based testing. The tool is primarily based on model checking technique and thus cannot be used for functional verification of combinational circuits, which is context of this paper.

VHDL Symbolic Simulator in Caml or VSYML [14] is a VHDL to XML translator that was deigned to evaluate the robustness of systems that were supposed to be fault-tolerant and intrusion resistant. The simulation results are used for evaluation purposes on the basis of theorem proving techniques. The main advantage of the XML format is that it provides a very organized, strictly tagged and hierarchical, version of the VHDL description, which facilitates translating to other formats. Due to this unique feature of XML descriptions, our proposed V-HOLT Verifier tool integrates the VSYML tool and converts the XML description to the desired HOL input format.

III. Proposed Approach

The proposed idea behind the V-HOLT Verifier is depicted in Figure 1. Besides the file formats, the main modules of our proposed consist of three translators, i.e., T1, T2 and T3. Translators T1 and T2 are developed in C++ and deal with the VHDL to HOL translation where translator T3, developed in JAVA, deals with the user property input and its conversion to the HOL syntax. We now explain these three on by one:

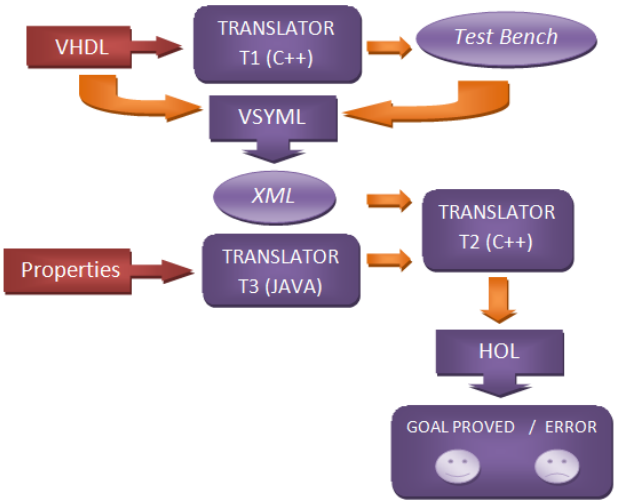

Figure 1: Proposed Approach

A. *Test Bench Generator:*

VSYML requires two inputs, i.e., the VHDL code and its corresponding test bench. In our case, we do not a test bench since our analysis approach is not simulation but for the sake of using VSYML, we had to provide a test bench. For this purpose, we developed an automatic test-bench generator (Translator T1) to save the user the trouble of having to generate a test-bench for any given VHDL code. The output of

T1 is just a test bench that declares the input and output variables with some dummy assignments as the assignments are not important at all in our context. This generator is developed in C++ and its output along with the VHDL code are given as the inputs for the VSYML tool to extract the XML format for the given VHDL code.

B. XML to HOL Translator:

The XML to HOL Translator T2, basically uses the technique of string matching and is also developed in C++. It reads the .xml file, corresponding to the given VHDL file, and generates two text files as its output. One of these files contains all the input and output port names which are utilized by the Property Generator (Translator T3). The second file contains the functionality of all the outputs of the VHDL file specified in HOL acceptable syntax. Thus, this file will have a line corresponding to each one of the outputs in the VHDL file. As indicated in Figure 1, one of the inputs to T2 is also coming from T3 which is the property to be verified for the given digital circuit. There is a property corresponding to each output. Translator T2 accepts these properties and patches them with the corresponding output functionality line and thus creates an input file for the HOL theorem prover with a goal, whose right-hand-side is the desired property and the left-hand-side, is the functionality that is extracted from the VHDL. This way the verification of this goal guarantees that the VHDL code exhibits the desired characteristics. In order to make the verification automatic, translator T2 also appends a generic HOL proof script with the goal. The theorems, tactics and simplifiers in the proof script are chosen in such a way that any goal related to basic Boolean logic or integer arithmetic can be proven automatically. Translator T2 can handle all basic combinational circuits like adders, decoders and multiplexers with single bit inputs.

C. Property Generating Interface:

The third Translator T3 is actually the User Interface Module through which the user actually interacts with V-HOLT Verifier. The Interface is designed to facilitate the user as much as possible so the entire process of writing a property and verifying it takes a total of six simple steps that are depicted in the Figure 2.

Figure 2: User Interface Steps

The users have no need to concern themselves with the previously mentioned translators T1 and T2 as they will be called in the background of this interface. The user only needs to know how to write a property on the interface and the rest of the work, including the property verification will be done by the interface itself. The 6 main steps are explained below:

1. The user is first asked to select the path of the directory containing all the executable files (Translator T1, T2, VHDL codes etc). In order for the Interface to function smoothly the user needs to make sure that all the required files are in the same directory. This is also the directory where all the generated text files will also be stored.
2. The next step is file compilation where the interface calls Translator T1, VSYML and Translator T2 to create the text files with the input and output port names of the VHDL file.
3. The I/O port files are then read by the Interface and the input ports are displayed as useable variables for the user and the output ports are shown as the output variables for the user on the GUI. This way the user can keep track of the port for which the user is going to be writing the property for.
4. The GUI of T3 provides buttons and check boxes for the user to write properties. These options correspond to Boolean operators AND, OR, NOT, equality and implication, arithmetic operators +, –, MOD and DIV and the if-else conditional statement. Using these primitives, any sort of combination circuit property can be specified. The user selects the input variable names and the operators to construct the property using the user-friendly interface. Once the property is finalized, the user presses a button through which the property is automatically translated to the corresponding HOL syntax compliant code.
5. The property file generation is the second last step which calls the Translator T2 to write the final HOL goal as described in the last section.
6. The last step is where HOL is invoked to verify the goal that has been generated. The HOL window provides the messages for the verified or failed goal. In case of failure, the exact problem can also be seen, which is a very useful debugging feature.

IV. FULL ADDER EXAMPLE

In order to illustrate the working of V-HOLT Verifier, we explain the complete verification flow using a simple example of the full adder circuit. The VHDL code that we used for the full adder circuit used two output ports S and Cout for sum and carry out, respectively, and has three inputs. The input-output relationship was defined as follows:

```
S <=A xor B xor C;
Cout <= (A and B) or ((A or B) and C);
```

Once the VHDL file for the code has been added in the same directory that contains the Translators and VSYML, HOL User Interface is opened and step 1 to step 3 are performed. The Interface will display the three inputs A B C as the input variables besides the checkboxes and the output bit S will show up as the output variable text box in the GUI. The next step is to write the property corresponding to the S output. We as a user of the tool provided the following property ((BV A + BV B + BV C) MOD 2), which ensures the correct behavior of the full-adder's sum bit. Next, the GUI displays Cout in the output variable text box and we entered its property (BV A + BV B +

BV C) DIV 2. The next step generates the HOL file, which is given below:

```
1.app load ["HolKernel", "Parse", "boolLib",
"bossLib", "boolTheory", "arithmeticTheory"];
2.open HolKernel Parse boolLib bossLib boolTheory
arithmeticTheory;

3. val BV = Define`BV b = (if b then SUC 0 else 0)`;

4.g`! (A:bool) (B:bool) (C:bool). BV(((A /\ B)  \/
((A  \/  B) /\ C)))=
((BV A + BV B + BV C) DIV 2)`;
5. e (RW_TAC std_ss [BV]);

6.g`! (A:bool) (B:bool) (C:bool).
BV(~ (~ (A = B) = C))=
((BV A + BV B + BV C) MOD 2)`;
7.e(RW_TAC std_ss [BV]);
```

The first two lines of the code indicate the loading of in-built HOL libraries that our required for the given verification. The next defines a HOL function that maps a bit to its corresponding integer value. The statements on lines 4 and 6 starting with a g represent the goals that need to be verified corresponding to the outputs Cout and S, respectively. The symbols /\, \/, and ~ represent HOL symbols for logical operations AND, OR and NOT, respectively. Finally, the command on lines 5 and 7 is the HOL tactic that would verify the combinational circuit related properties automatically. In the last step the above file is given to the HOL theorem prover and it automatically verifies the properties and a snapshot of the HOL output is given in Figure 3.

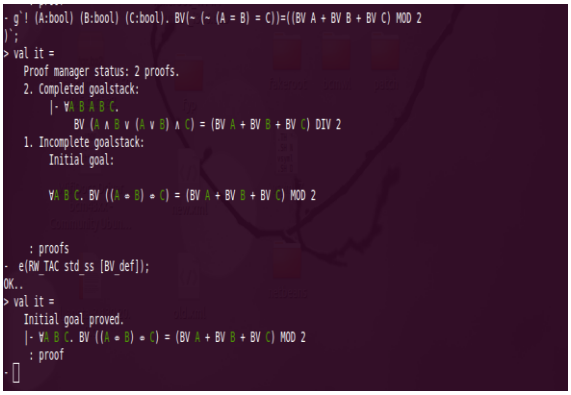

Figure 3: HOL output for the Full Adder Circuit

V. CONCLUSIONS

In this paper, we presented an automatic formal verification tool V-HOLT Verifier that is capable of verifying any combinational circuit. The distinguishing feature of the tool is its automatic and user friendly nature when compared to the other existing formal verification tools. A user with no prior knowhow about formal verification or the HOL theorem prover can easily use our tool. It can be seen from the HOL code given in the previous section that writing such code manually is not a straightforward job, which clearly indicates the usefulness of the proposed too.

V-HOLT is smart enough to recognize errors in the submitted VHDL file and appropriate error messages are generated. Moreover, the automatic syntax conversion to HOL ensures that the property is never written in the wrong syntax even if the user has no idea of what the HOL syntax is like.

The paper describes the functioning of the tool using a simple full-adder circuit example for illustration purposes. We have been able to successfully verify many other classical combinational circuits using the V-HOLT Verifier including different sized Muxes and Decoders. We are experimenting with many other case studies as well including several benchmark combinational circuits with high number of gates. Furthermore, extending V-HOLT Verifier to tackle vectored input circuits is also under investigation.

ACKNOWLEDGMENT

This work was supported by the National Research Program for Universities grant (number 1543) of Higher Education Commission (HEC), Pakistan.

REFERENCES

[1] T. Kropf, Introduction to Formal Hardware Verification, Springer, 1999.

[2] Clarke, E.M., Emerson, E.A.: Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic. Proc. of Workshop Logic of Programs, LNCS 131, 1981.

[3] O'Leary, J., Zhao, X., Gerth, R., Seger, C.H.: Formally Verifying IEEE Compliance of Floating-Point Hardware. Intel Technical Journal, Q1:147-190, 1999.

[4] K. L. McMillan: A methodology for hardware verification using compositional model checking, Science of Computer Programming, Volume 37, Issues 1-3, May 2000, Pages 279-309.

[5] Bjesse, P., Leonard, T., Mokkedem, A.: Finding Bugs in an Alpha Microprocessor Using Satisfiability Solvers. In Proc. Computer Aided Verification. LNCS 2102, pp.454-464, 2001.

[6] Sun, X. Xie, F. Wu, J. Song, X.: Verification of a network ASIC Component using Bounded Model Checking, Int. J. Electronics, 2007, 94(2):183-196.

[7] Franco Raimondi, Alessio Lomuscio: Automatic verification of multi-agent systems by model checking via ordered binary decision diagrams, Journal of Applied Logic, 2007, 5(2): 235-251.

[8] Gordon, M.J.C., Melham, T.F.: Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic. Cambridge University Press,1993.

[9] Owre, S., Rushby, J.M., Shankar, N.: PVS: A Prototype Verification System. Proc. of International Conference on Automated Deduction, Lecture Notes in Artificial Intelligence 607, pp. 748-752, 1992.

[10] Kaufmann, M., Manolios, P., Moore, J.S.: Computer-Aided Reasoning. An Approach. Kluwer Academic Publishers, 2000. http://www.cs.utexas.edu/users/moore/acl2.

[11] Warren A. Hunt Jr., Sol Swords, Jared Davis, and Anna Slobadova. Use of Formal Verification at Centaur Technology. In Design and Verification of Microprocessor Systems for High Assurance Applications. 2010. Springer. Pages 65-88

[12] Vanderlei Moraes Rodrigues: Dominique Borrione, Philippe Georgelin Laboratoire TIMA, UJF at Grenoble, France, An ACL2 Model of VHDL for Symbolic Simulation and Formal Verification,Integrated Circuits and Systems Design, 2000. pp. 269-274, 2000.

[13] K. Havelund, J. Penix, W. Visser, The Temporal Rover and the ATG Rover, SPIN 2000, LNCS 1885, pp. 323-330, 2000.

[14] Florent Ouchet, Dominique Borrione, Katell Morin-Allory, Laurence Pierre, High-level symbolic simulation for automatic model extraction, DDECS, pp. 218-221, 2009.