Formal Hardware Verification by Integrating HOL and MDG

V.K. Pisini[1], S. Tahar[1], P. Curzon[2], O. Ait-Mohamed[3] and X. Song[4]

Abstract

In order to overcome the limitations of automated tools and the cumbersome proof process of interactive theorem proving, we adopt a hybrid approach for formal hardware verification which uses the strengths of theorem proving (HOL) with powerful mathematical tools such as induction and abstraction, and the advantages of automated tools (MDG) which support equivalence checking and model checking. The MDG system is a decision diagram based verification tool, primarily designed for hardware verification. HOL is a theorem prover built on higher-order logic.

1 INTRODUCTION

With the ever increasing complexity of the design of digital systems and the size of the circuits in VLSI technology, the role of design verification has gained a lot of importance. Simulation, which is the state-o-the-art is often used as the main approach for verification, and despite the major simulation efforts, serious design errors often remain undetected which resulted in the evolution of applications such as formal methods in verifying the hardware design. There are several approaches to formal hardware verification: theorem-proving, model checking, equivalence checking, symbolic simulation to name a few [1]. Each of them has its own strengths and weaknesses. In this paper we present a methodology with an example as to how equivalence checking of the automated MDG system [2] supports the proof process of the HOL theorem prover [3]. HOL is an interactive system that is built on higher-order logic and developed at the University of Cambridge, U.K. Theorem proving can handle very large circuits for verification but it is a cumbersome and time-consuming process and needs expertise in using it. We believe that the present VLSI industry, however needs the automation of the verification process as much as possible without suffering the under-capability of the automated tools when it comes to handling large circuits. The integration of interactive and automated tools eases the verification complexity to a great extent.

The remaining sections of this paper are organized as follows. Section 2 contains related work in the area. Section 3 describes the HOL and MDG systems. In Section 4 we present the methodology of our hybrid approach and how MDG is embedded inside the logic of an interactive theorem-prover. In Section 5 we present an example we considered, the Timing block of the Fairisle ATM switch fabric, through which we illustrate the advantages of our approach. Section 6 finally concludes the paper.

[1]ECE Dept., Concordia University, Montreal, Canada
[2]School of Computing Science, Middlesex University, London, UK
[3]Nortel Networks, Ottawa, Canada
[4]IRO Dept., Université de Montreal, Montreal, Canada

2 RELATED WORK

There exist a number of hybrid approaches such as combining theorem proving with model checking [4], [5] and combining theorem proving and symbolic trajectory evaluation [6]. For instance, Rajan *et al* [5] described an approach where a BDD-based model checker for the propositional mu-calculus has been used as a decision procedure within the framework of the PVS proof checker. Joyce and Seger [6] described an approach by means of an interface between the Voss system and HOL. They have implemented a tactic VOSS_TAC which calls the Voss system to do a part of the verification using symbolic trajectory evaluation to decide whether an assertion is true which in turn can be transformed into a HOL theorem and this theorem is used by the HOL system to proceed with further verification procedure. Schneider *et al* [4] proposed an approach of invoking model checking from within HOL where properties are translated from HOL to temporal logic. More recently, HOL98 has been integrated with the BuDDy BDD package [7].

The motivation to take up this work originated while looking into ways to integrate VHDL and formal verification. The work described in this paper is part of a larger project to link VHDL, HOL and MDG as shown in Figure 1. Here, the VHDL model is analyzed to get a data structure (Directed Acyclic Graph—DAG) of the model which is passed through an HOL Generator to get the HOL model. Within HOL, we use the functions, MDG_COMB_TAC and MDG_SEQ_TAC, to generate the required files for the MDG system to complete the verification for combinational and sequential verifications respectively. In the case of property verification, an LTL property description (L_MDG) [8] is transformed into an equivalent VHDL or MDG-HDL circuit description that will either be fed into the Analyzer or directly to the MDG system, respectively.

3 HOL AND MDG

3.1 HOL System

The HOL System is a theorem prover based on higher-order logic [3] which was originally intended for use in

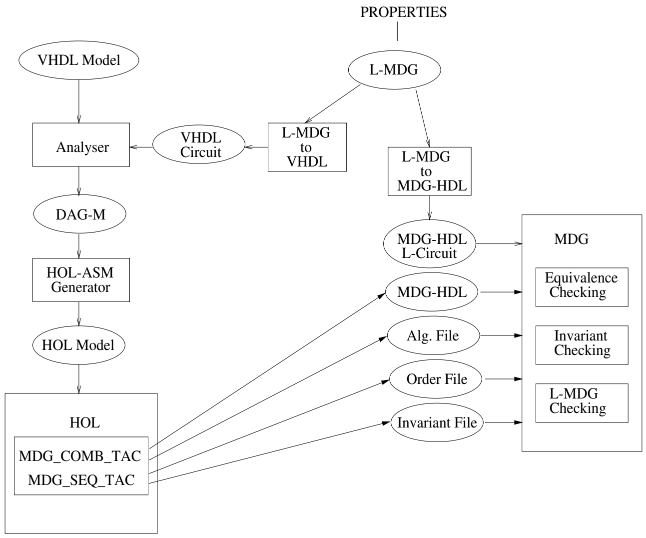

Fig. 1. Intended VHDL-HOL-MDG Project Skeleton

hardware verification but now used in a variety of application areas since it is a general purpose proof system. In theorem proving approach to verification, a system and its properties are described by means of logical formulae and the system is shown by means of a logical proof to entail the desired properties. The basic interface to the system is a Standard ML (SML) interpreter. SML is both the implementation language of the system and the meta-language in which proofs are written. Proofs are input to the system as calls to SML functions. Higher-order logic is very flexible and has a well-defined and well-understood semantics. It supports forward and backward proof by creating theorems and applying inference rules to the already created theorems. In the backward proof, the user sets the desired theorem as a goal. Tactics are applied to the goal to create sub-goals and inference rules are applied to prove the sub-goals which in turn proves the main goal. By applying a set of primitive inference rules, a theorem can be created. Complex inference rules call for the simpler inference rules to do the work. The results are strong and the user can have great confidence since the most primitive rules are used to prove a theorem.

HOL also allows us to use hierarchical verification methodology wherein the modules are divided into submodules and even the sub-modules are divided until the lowest level (gate level) is reached. The behavioral and structural specifications of each module are expressed in higher-order logic and each module is verified by proving a theorem stating that the implementation implies the specification. Each sub-module is verfied, and its result is used to verify the other sub-modules as needed. To complete a verification, however, a very deep understanding of the internal structure of the design is required, as it is a white-box approach. Modeling and verifying a system is very time-consuming.

3.2 MDG System

The MDG system is a decision diagram based verification tool, primarily designed for hardware verification which allows equivalence checking and model checking. The MDG verification approach is a black-box approach. During the verification the user does not need to understand the internal structure of the design being verified. The strength of MDG is its speed and ease of use. The MDG hardware verification system has been used in the verification of significant hardware examples [2].

Multiway Decision Graphs (MDGs) have been proposed [2] as a solution to the data width problem of ROBDD based verification tools. The MDG tool combines the advantages of representing a circuit at higher abstract levels as is possible in a theorem prover, and of the automation offered by ROBDD based tools. An MDG is a finite, directed acyclic graph (DAG). MDGs essentially represent relations rather than functions. MDGs can also represent sets of states. They are much more compact than ROBDDs for designs containing a datapath. Furthermore, sequential circuits can be verified independently of the width of the datapath. The MDG tools package the basic MDG operators and verification procedures [9]. The verification procedures are combinational and sequential verification. The combinational verification provides the equivalence checking of two combinational circuits. The sequential verification provides invariant checking and equivalence checking of two state machines. The MDG operators and verification procedures are implemented in Quintus Prolog [9].

MDG-HDL which is the input language for MDG, supports structural descriptions, behavioral ASM descriptions or a mixture of both. A structural description is usually a netlist of components connected by signals, and a behavioral description is given by a tabular representation of the transition/output relation or truth table. The MDG-HDL comes with a large library of predefined, commonly used, basic components (such as logic gates, multiplexers, registers, bus drivers, ROMs, etc.). A circuit description includes the definition of signals, components and the circuit outputs. Signals are declared along with their sorts. Components are declared by the instantiation of the input/output ports of a predefined component module. Among predefined modules we have a special module called a table. Tables can be used to describe a functional block in the implementation, as well as in the specification. A table is essentially a series of lists, together with a single final default value. The first list contains variables and cross-terms. The last element of the list must be a variable (either concrete or abstract). For example, a 2-input AND gate can be described as a table as:

```
table([[x1,x2,y], [0,*,0], [1,0,0] | 1])
```

The necessary files for verification in MDG are: a behavioral specification file, a circuit description file, an algebraic file, a symbol order file, and an invariant file [9]. The behavioral specification file declares signals and specifies the behavior of the circuit using tables as described above. The circuit description file declares signals and their sort assignments and describes the circuit network. The algebraic specification file defines sorts, function types and generic constants. The symbol order file provides the user-defined symbol order for all the variables and cross operators which would appear in MDGs. The invariant file takes the corresponding outputs from both behavioral specification and circuit description for equivalence checking using MDGs.

4 LINKING APPROACH

4.1 Hierarchical Verification

In our hybrid approach, we follow a hierarchical hardware verification methodology. Generally, when we use HOL to verify a design, the design is modeled as a hierarchy structure with modules divided into sub-modules as shown in Fig. 2. The sub-modules are repeatedly subdivided until eventually the logic gate level is reached.

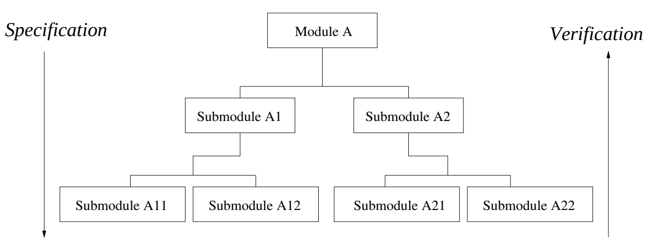

Fig. 2. Hierarchical Verification

By proving a theorem saying that the implementation (structure) implements the specification (behavior), we accomplish the verification of each module. That is:

$$\vdash Implementation\_A \Longrightarrow Specification\_A \quad (4.1)$$

The verification starts in HOL with a goal to be proved. The correctness theorem for each module states that its implementation down to the logic gate level satisfies the specification. The correctness theorem for each module can be established using the correctness theorems of its sub-modules. When the module is subdivided, then we can write the theorem about the structural description as

$$\vdash Implementation\_A = Imp\_A1 \land Imp\_A2 \quad (4.2)$$

Now (4.1) can be written as

$$\vdash Imp\_A1 \land Imp\_A2 \Longrightarrow Specification\_A \quad (4.3)$$

The correctness statements of the sub-modules $A1$ and $A2$ can be used to prove the correctness theorem for the module A. Likewise we can prove independently for each sub-module that

$$\vdash Imp\_A1 \Longrightarrow Spec\_A1 \quad (4.4)$$

$$\vdash Imp\_A2 \Longrightarrow Spec\_A2 \quad (4.5)$$

Since these are implications, to prove (4.1), it is enough to prove that

$$\vdash Spec\_A1 \land Spec\_A2 \Longrightarrow Specification\_A \quad (4.6)$$

Similarly, $A1$ is verified from its sub-modules $A11$ and $A12$, and $A2$ is verified from its sub-modules $A21$ and $A22$. Hence, we verify module A by independently verifying its sub-modules $A1$ and $A2$. Using this top-down approach, the main objective of our work is to identify and prove the correctness of certain sub-modules in an automatic fashion using the MDG system. In MDG, for each sub-module it will be proved by automatic verification that implementation is equivalent to specification and the result is imported into HOL. In our hybrid system, the sub-module is treated as a black-box.

4.2 Translation of HOL Description to MDG

In HOL, the specification and implementation are expressed in higher-order logic. The MDG system uses MDG-HDL to describe the implementation and the specification, the latter is written in the table form [10]. The sub-goals from the main goal are generated by HOL. The user decides if the sub-goal can be proved in MDG and its description is written in MDG acceptable form using the description predicates. In case a sub-goal is not expressed in the MDG acceptable form or the MDG verification fails, then the regular HOL proof procedure is followed. Once all the sub-goals are proved, it implies that the main goal is proved and hence the circuit is formally verified. As in the block diagram of the hybrid system shown in Fig. 3, the interface converts the HOL descriptions to equivalent MDG files and all required files for the MDG verification as specified in the following. It is a lot different from the mere translation of the output of one tool to the input of the other tool.

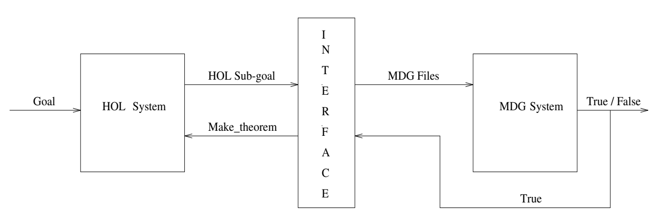

Fig. 3. Block Diagram of the Hybrid System

The sub-goal specification and implementation which are in two different files are given as input to the interface which is built in SML. The two HOL files contain the inputs, outputs, intermediate outputs and their signal types. If there are new user defined types, they are defined earlier in the theory. From the given two HOL files, corresponding MDG circuit description, specification, algebraic, order and invariant files are created automatically. These files are used for equivalence checking verification by the MDG system. In the case where the equivalence checking has succeeded, MDG returns "true", this result is imported into HOL in the form of a theorem (using the *make_theorem* in HOL) and the main proof procedure continues in HOL with the next sub-goal to be proved. Xiong *et al* [11] showed how the results of MDG can be imported into HOL. As part of the build-up of the mathematical interface between the two tools, the total MDG library was specified in HOL as predicates and Curzon *et al* [10] formally verified the MDG component library in HOL. They also showed how the MDG tables can be expressed in HOL.

Within HOL, the tactics MDG_COMB_TAC (for combinational verification) or MDG_SEQ_TAC (for sequential verification) starts the translation of the files and the verification in MDG and analyses the result to eventually generate a theorem. The tasks of these tactics are shown as a flow-diagram in Fig. 4.

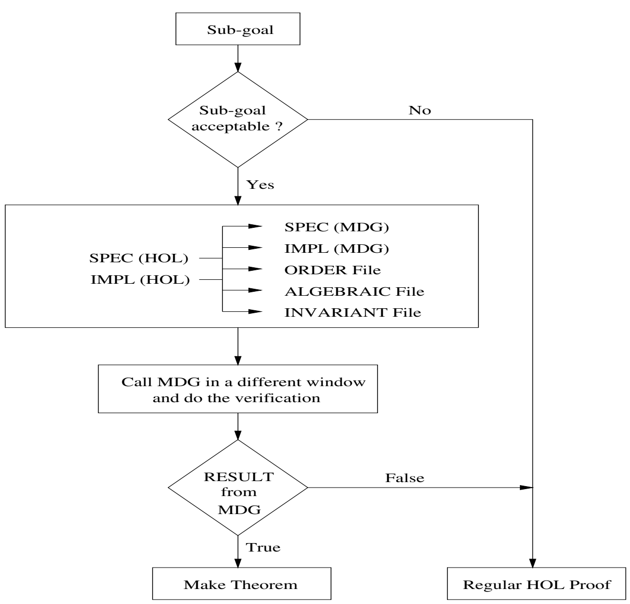

Fig. 4. Task of MDG_COMB_TAC/MDG_SEQ_TAC

5 HYBRID VERIFICATION METHODOLOGY

For illustration purposes, we show the verification of a sub-module of the Fairisle ATM switch fabric [12] (see Fig. 5). Curzon [13] formally verified this ATM switching element using the theorem-prover HOL. The Fairisle switch fabric is a real switch fabric designed and in use at University of Cambridge for multimedia applications. The Fairisle switch forms the heart of the Fairisle network. Considering the fabric as the main module to be verified, it can be split into 3 sub-modules, namely Acknowledgement, Arbitration and Data Switch. Further dividing the Arbitration sub-module, we have Timing, Decoder, Priority Filter and Arbiters as sub-sub-modules. In our example, we have taken the Timing block to be a sub-sub-module (one of the sub-goals) and used our hybrid tool to achieve the desired verification result.

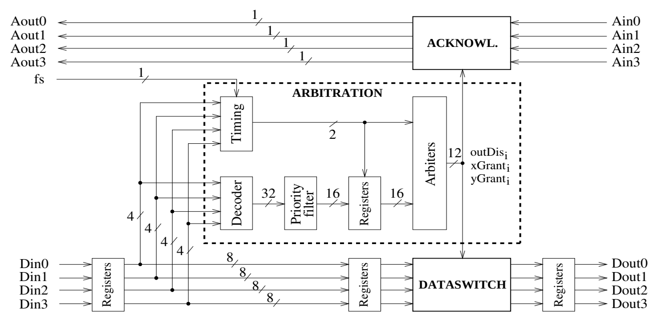

Fig. 5. Fairisle ATM Switch Fabric

5.1 Proof Structure of the ATM Fabric

The verification of the Fairisle switch fabric is arranged according to the division of the fabric in a hierarchical fashion as shown in Fig.6. The goal is to prove that

$$\vdash Fabric\_Imp \Longrightarrow Fabric\_Spec \tag{5.1}$$

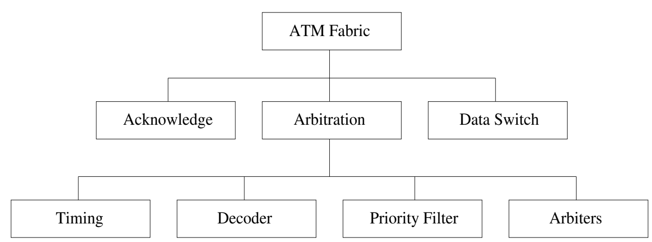

Fig. 6. Hierarchical Verification of 4x4 Switch Fabric

From Fig.6 and the equations in Section 4.1, we have

$$\vdash Fabric\_Imp = Ack\_Imp \wedge Arb\_Imp \wedge DataSW\_Imp \tag{5.2}$$

as in (4.4) and (4.5), we can prove that

$$\vdash Ack\_Imp \Longrightarrow Ack\_Spec \tag{5.3}$$
$$\vdash Arb\_Imp \Longrightarrow Arb\_Spec \tag{5.4}$$
$$\vdash DataSW\_Imp \Longrightarrow DataSW\_Spec \tag{5.5}$$

Now it is enough to prove that

$$\vdash Ack\_Spec \wedge Arb\_Spec \wedge DataSW\_Spec \Longrightarrow Fabric\_Spec \tag{5.6}$$

Likewise, at the next lower level the Arbitration block is proved in the same fashion. In this Arbitration block, one of the sub-modules or sub-goal is the Timing block. Instead of proving the implication in HOL, it can be proved using equivalence in MDG which we illustrate in the following section.

5.2 Timing Block Verification

The Timing block controls the timing of the arbitration decision based on the frame start signal and the time the routing bytes arrive. The implementation of the timing is shown in Fig. 7 and the FSM representation is shown in Fig. 8.

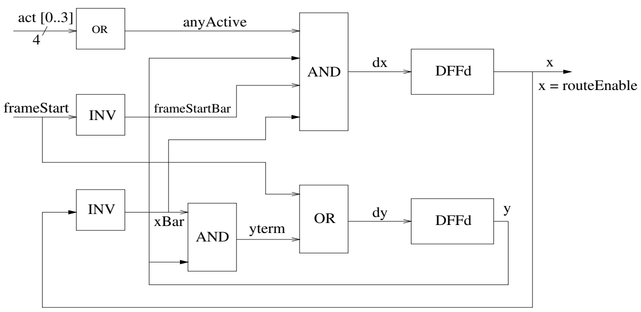

Fig. 7. Implementation of the Timing Block

We wrote the high level specification and implentation and used our hybrid tool to do the verification using equivalence checking of MDG. The implementation of the Timing block shown in Fig.7 described in HOL is:

```
⊢ ∀ frameStart act0 act1 act2 act3
           routeEnable.
 TIMING_IMP((frameStart act0 act1 act2 act3))
           ((routeEnable)) =
∃ anyActive frameStartBar
   x xBar y yterm dx dy .
 (or4 act0 act1 act2 act3 anyActive) ∧
 (not frameStart frameStartBar) ∧
 (not x xBar) ∧
 (and xBar y yterm) ∧
 (and4 anyActive y frameStartBar xBar dx) ∧
 (or frameStart yterm dy) ∧
 (reg dx x) ∧
 (reg dy y) ∧
 (fork x routeEnable)
```

The resulting MDG-HDL implementation of the Timing block equivalent to that of HOL, generated by MDG_SEQ_TAC is:

```
component(anyActive_impl,or4(input
 (act0,act1,act2,act3),output(anyActive))).
component(frameStartBar_impl,not(input
 (frameStart),output(frameStartBar))).
component(xBar_impl,not(input(x),
 output(xBar))).
component(yterm_impl,and(input(y,xBar),
 output(yterm))).
component(dx_impl,and4(input(anyActive,
 y,frameStartBar,xBar),output(dx))).
component(dy_impl,or(input
 (frameStart,yterm),output(dy))).
component(x_impl,reg(input(dx),output(x))).
component(y_impl,reg(input(dy),output(y))).
component(fork_for_routeEnable_impl,
 fork(input(x),output(routeEnable))).
```

The specifications of the Timing block in HOL and MDG are shown below. The HOL specification of the Timing FSM is described using a state transition function and an output function. The HOL definition of the state transitions of FSM of Fig. 8, written in terms of the table specification is given as:

```
TABLE [anyActive;frameStart;timing_state]
    (n_timing_state o NEXT)
 [[DONT_CARE;TABLE_VAL(TRANS T);TABLE_VAL
    (STATE RUN)];
 [DONT_CARE;TABLE_VAL(TRANS F);TABLE_VAL
    (STATE RUN)];
 [TABLE_VAL(TRANS T);TABLE_VAL(TRANS F);
    TABLE_VAL(STATE WAIT)];
 [DONT_CARE;TABLE_VAL(TRANS F);TABLE_VAL
    (STATE ROUTE)];
 [DONT_CARE;TABLE_VAL(TRANS T);TABLE_VAL
    (STATE ROUTE)]]
 [WAITSIG;RUNSIG;ROUTESIG;RUNSIG;WAITSIG]
    WAITSIG
```

The equivalent MDG table specification of the Timing FSM state transition is generated using MDG_SEQ_TAC as:

```
[[anyActive, frameStart, timing_state,
    n_timing_state],
 [*,1,run, wait],
 [*,0,run, run],
 [1,0,wait, route],
 [*,0,route, run],
 [*,1,route, wait] | wait]
```

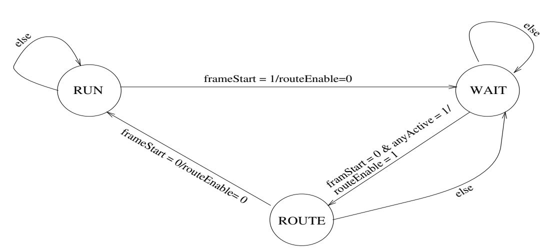

Fig. 8. State Transitions of the Timing Block

Once the specification and implementation in HOL are translated, MDG_SEQ_TAC generates the required order file, algebraic specification file and invariant file and calls MDG for equivalence checking. The succeeded result from MDG is imported into HOL as a theorem. And hence the verification of the Timing block is done.

5.3 Verification Results

We have shown that:

$$Timing\_Imp \equiv Timing\_Spec \text{ (equivalence)} \tag{6.1}$$

We got the above result from MDG and it is imported into HOL [11] as:

$$\vdash Timing\_Imp \Longrightarrow Timing\_Spec \tag{6.2}$$

Using similar MDG proofs for the other sub-modules of the arbitration block, we get:

$$\vdash Timing\_Spec \wedge Decoder\_Spec \wedge PFilter\_Spec \wedge Arbiters\_Spec \Longrightarrow Arbitration\_Spec \tag{6.3}$$

Using our hybrid tool, the procedure is faster than proving in HOL that the implementation implies the high-level specification. Curzon [13] took several hours to do the proof of the Timing block whereas the verification is done in less than a second in MDG (see Table I). The verification results obtained by means of equivalence checking can be formally related to higher levels of abstraction. Also, *equivalence* is a stronger result than *implication*.

MDG Nodes	CPU Time (sec.)	Memory (MB)
227	0.41	0.161

TABLE I

MDG Equivalence Checking Results for Timing Block

We showed using MDG that the structural description (i.e. implementation) is equivalent to the high-level specification, described in terms of tables. Writing the high-level specification using tables in MDG is far easy compared to writing it down in HOL. In HOL, the proof is interactive and is time-consuming [13].

6 CONCLUSIONS

To summarize our work, we have built a linkage tool between HOL and MDG. The tool uses the HOL specification and implementation files and generates all the required MDG files automatically. It then calls the MDG equivalence checking procedure and generates an appropriate theorem in HOL in the positive case. We thus yield a significant reduction of specification and verification time avoiding cumbersome proof process of HOL as shown by our example.

This hybrid approach is more effective in hierarchical verification. If the main module can be divided into smaller sub-modules, then certainly the use of this hybrid approach proves to be effective since there are less chances of state-explosion problem since MDG can effectively handle smaller circuits.

Acknowledgements

This work was partially supported by Micronet research grant and a GRIAO student scholarship. Thanks are due to E. Cerny and A. Dekdouk at IRO, University of Montreal and to I. Kort and M.H. Zobair at ECE Dept., Concordia University who helped us out with HOL and MDG proofs.

References

[1] C. Seger, "An Introduction to Formal Hardware Verification," Tech. Rep. 92-13, Dept. of Computer Science, University of British Columbia, Vancouver, B.C., Canada, June 1992.

[2] F. Corella, Z. Zhou, X. Song, M. Langevin, and E. Cerny, "Multiway Decision Graphs for Automated Hardware Verification," *Formal Methods in System Design*, vol. 10, no. 1, pp. 7–46, 1997.

[3] M. Gordon and T. Melham, *Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic.* Cambridge University Press, Cambridge, U.K., 1993.

[4] K. Schneider and T. Kropf, "Verifying Hardware Correctness by Combining Theorem Proving and Model Checking," Tech. Rep. SFB358-C2-5/95, University of Karlsruhe, Karlsruhe, Germany, December 1995.

[5] S. Rajan, N. Shankar, and M. Srivas, "An Integration of Model-checking with Automated Proof Checking," in *Computer Aided Verification* (P. Wolper, ed.), Lecture Notes in Computer Science 939, pp. 84–97, Springer Verlag, 1995.

[6] J. Joyce and C. Seger, "Linking BDD-based Symbolic Evaluation to Interactive Theorem Proving," in *Proceedings of the 30th Design Automation Conference*, 1993.

[7] M. Gordon, "Combining Deductive Theorem Proving with Symbolic State Enumeration." 21 Years of Hardware Verification, December 1998. Royal Society Workshop to mark 21 years of BCS FACS.

[8] Y. Xu, E. Cerny, X. Song, F. Corella, and O. At-Mohamed, "Model Checking for a First-Order Temporal Logic using Multiway Decision Graphs," in *Computer Aided Verification* (A. Hu and M. Vardi, eds.), Lecture Notes in Computer Science 1427, pp. 219–231, Springer Verlag, 1998.

[9] Z. Zhou and N. Boulerice, *MDG Tools (V1.0) User's Manual.* Dept. of Computer Science, University of Montreal, Montreal, Canada, June 1996.

[10] P. Curzon, S. Tahar, and O. Ait-Mohamed, "Verification of the MDG Components Library in HOL," in *Theorem Proving in Higher Order Logics: Emerging Trends* (J. Grundy and M. Newey, eds.), (Australian National University, Canberra, Australia), pp. 31–45, September 1998.

[11] H. Xiong, P. Curzon, and S. Tahar, "Importing MDG Results into HOL," in *Theorem Proving in Higher Order Logics* (Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Theryvon, eds.), Lecture Notes in Computer Science 1690, pp. 293–310, Springer Verlag, 1999.

[12] I. Leslie and D. McAuley, "Fairisle: An ATM Network for the Local Area," *ACM Communication Review*, vol. 19(4), pp. 327–336, 1991.

[13] P. Curzon, "The Formal Verification of the Fairisle ATM Switching Element," Techical Report 329, Computer Laboratory, University of Cambridge, U.K., March 1994.