Automating the Verification of Parameterized Hardware using a Hybrid Tool

Paul Curzon[1] and Sofiene Tahar[2]

[1]School of Computing Science, Middlesex University, UK.
[2]Dept of Electrical and Computer Engineering, Concordia University, Canada.
Email: p.curzon@mdx.ac.uk, tahar@ece.concordia.ca

Abstract— **We outline how a hybrid formal hardware verification tool that links an interactive theorem prover and an automated hardware verification tool, can verify parameterized circuits containing replicated components. We show that the approach integrates well with the hierarchical proof approach embodied in the hybrid tool.**

I. Introduction

There are many techniques available to the hardware designer to verify a given design meets its specification. Traditionally, simulation and testing have been used. More recently formal approaches to hardware verification have become significant [6]. In these approaches mathematical techniques are used to prove facts about the correctness of the design. Such work is based on mathematical descriptions of the design and the specification it should meet or properties that should be true of it. Formal reasoning about the design, specification and properties can then be performed. Rather than just verifying that the design meets the specification for a set of values tested, a proof can be conducted that the design meets the specification for all input values.

There are two broad approaches to formal verification: automated decision diagram techniques and interactive deductive proof techniques [6]. In the former, decision diagrams are used to represent the design and specification. By utilizing the fact that the decision diagrams have a canonical form, the equivalence of two circuit descriptions can be determined. State-space exploration techniques can also be used to automatically enumerate all possible behaviors, and so check, for example, that particular properties always hold. In contrast, in the deductive proof approach descriptions of the design and specification are given in some logic such as higher-order logic. A mathematical proof within that logic is then constructed of a theorem that the design implements the specification. A theorem prover is used both to construct the proof and to provide automation of proof steps. This approach is well-suited to performing hierarchical verification where each submodule in the design is verified independently, with the resulting theorems combined to give a correctness theorem for the whole design. This allows the approach to scale to arbitrarily large circuits, at least in theory.

Automated decision diagram based formal hardware verification is fast and convenient, but does not scale well [6]. A particular problem is that only concrete circuits can be verified: all details of the design must be fully specified before a verification can be performed. Parameterized circuits, which contain n replicated components where the value of n is unspecified, cannot be verified directly within the tool. A new verification would need to be performed for each possible value of n, or informal reasoning performed outside the tool. In contrast, parameterized verification is easily performed using interactive proof systems. Here a theorem that the circuit is correct for all n can be proved. However, much more user intervention is required to do the verification. Rather than minutes of automated proof, hours might be spent constructing an interactive proof. Ideally the advantages of both approaches would be obtained. The bulk of the low-level verification would be performed quickly and automatically, but parameterized aspects of the proof still need to be done.

The contribution of this paper is not to introduce the idea of verifying parameterized circuits which is well known within the theorem proving community (see for example [1]). Rather we show how such techniques can be used to extend the capabilities of a particular automated decision diagram based verification system that is combined in a hybrid tool [7] with a theorem prover. In particular, we show how such techniques can be integrated seamlessly with a hybrid tool designed to explicitly support hierarchical verification. In this paper we first overview the hybrid tool, we then describe how a proof of a parameterized circuit can be performed semi-interactively using the hybrid tool.

II. The Hybrid Tool

Work to combine the advantages of automated and interactive tools falls generally into two areas: hybrid tools in which two existing, stand-alone verification systems are linked such as the Voss-ThmTac System [2] and the SMV-HOL linkage [10] and systems where external proof packages are tightly integrated as decision procedures for some subset of the logic by an interactive system [9]. One focus of work is on proof management tools [1][8] used to break down and recombine subgoals in the interactive prover to give to the hybrid tool. Such tools have not previously provided direct support for hierarchical proof.

In this paper we discuss how a hybrid tool linking HOL [5], a higher-order logic theorem prover, and MDG [4], a decision diagram based system, can be used to verify parameterized circuits. We have previously demonstrated that this tool can be used to verify concrete circuits much

This work was funded by EPSRC Grant GR/M45221 and NSERC Grant OGP0194302.

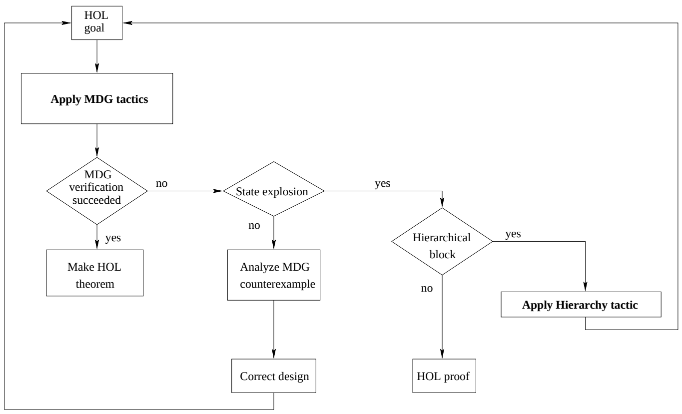

Fig. 1. Using the hybrid tool

faster than in either tool alone [7].

HOL is a general purpose theorem prover that has been used in a wide variety of application areas, not just hardware verification. Specifications and goals are written in higher-order logic. The user works interactively with the system calling proof functions, called *tactics*, that implement the inference rules of the logic to apply proof steps and so create theorems. A subgoal manager keeps track of so far unproven subgoals required to prove the original goal. Results produced by external tools can be imported into HOL.

The MDG hardware verification system provides verification procedures for equivalence and property checking. The former provides the verification of two combinational circuits or of two state machines. The latter allows verification through invariant checking or model checking. The strength of the MDG system is its automation and ease of use. It has been used in the verification of significant hardware examples (e.g., [11], [3]). The MDG system is a decision diagram based verification tool using Multiway Decision Graphs (MDGs) [4]. An MDG is a finite, directed acyclic graph (DAG). The MDG tools combine some of the advantages of representing a circuit at more abstract levels with the automation offered by other decision-diagram based tools. The input language for MDG, MDG-HDL, supports structural descriptions, behavioral 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 of the component.

The hybrid tool links these systems in a way that directly supports hierarchical verification. Figure 1 outlines the use of the tool. HOL is used as the proof manager. To perform a verification, a goal that the implementation implies the specification is set using the HOL subgoal manager. The MDG automated verification tools can then be called to prove the goal. Such use of MDG is essentially seamless in the sense that MDG is treated like any other tactic within the HOL system as far as the user is concerned. If MDG fails to prove the goal due to state-space explosion, then a specially written HOL hierarchical verification tactic can be called. This uses the hierarchy explicit in the implementation to break down the original goal to goals about the components of the circuit. A correctness subgoal is automatically generated for each immediate submodule of the circuit. These subgoals can be verified in the same way using the MDG tactics or further hierarchical verification. The HOL subgoal manager automatically keeps track of unproved goals. Once the subgoals are proved, the hierarchy tactic combines the resulting theorem into the correctness theorem of the original goal. Thus HOL proof is used to manage the hierarchical aspects of a proof, whereas MDG is called to automatically deal with low level proof. In this way much larger verification can be performed than in either tool on its own.

The hybrid tool must be supplied with a behavioral specification written in higher-order logic for each block in the design that is verified independently. Structural specifications are similarly written in higher-order logic. However, the structural specification of a block differs from a behavioral specification in that its body consists of a network of components. These components (registers, logic gates, etc.) are predefined in the logic.

III. Parameterized Circuit Descriptions

A parameterized circuit description describes a family of similar circuits. For example it might contain some replicated part where the number of times that part is replicated is specified as a variable. By providing different values for the variable, different circuits within the family are obtained. We will examine two simple classes of parameterized circuit to show how such circuits can be verified using our hybrid tool.

A. Parallel Composition

A very simple form of parameterized circuit consists of a component that is replicated in parallel. Each component takes similar input and produces similar output, but there is no interconnection between the replicated parts (see Figure 2). In essence, this kind of replication alters the datapath width. A simple example of its use would be in specifying a register constructed from 1-bit delays. However, it also occurs where much larger components are replicated. A common form of simplification used when using automated decision diagram systems (needed to make a verification tractable) is to verify such circuits as though they consisted only of a single component—simplifying the datapath from say 32-bits to 1-bit. It is then deduced informally that the full circuit is correct. MDG provides an alternative way of dealing with this simple situation using abstract variables. However, the current implementation of the hybrid tool does not make such variables available. We therefore look at a more general approach.

Using the hybrid tool, specifications are written in HOL and then exported to MDG. Submodules of the circuit that are to be verified by calls to MDG must therefore be in a standard syntax. However high level modules that are not going to be verified in MDG do not need to stick to this limited form. They can use the full power of higher-order logic specification. This does not preclude their submodules (if

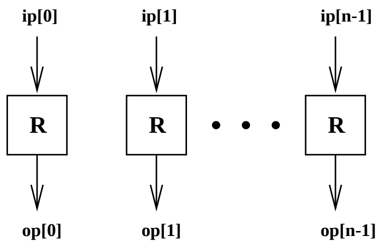

Fig. 2. The Parallel Circuit Pattern

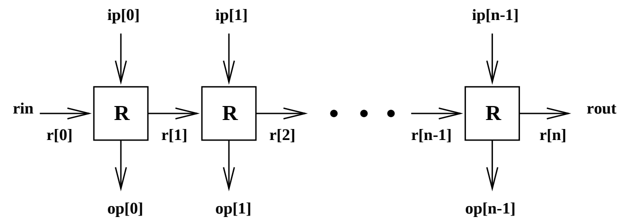

Fig. 3. The Ripple Circuit Pattern

in the appropriate form) from being verified using MDG. In particular, we can give parameterized specifications to the hybrid tool, provided we only export non-parameterized modules to MDG to verify.

The following is a general description of the circuit of Figure 2.

PARALLEL_I (n, ip, op) =
  ∀k (0 ≤ k < n). I (ip[k], op[k])

This defines a parameterized circuit implementation called PARALLEL_I that is parameterized by n. It has inputs, *ip*, and outputs, *op*. The implementation is then specified as consisting of a series of instances of hardware component I: one for each value of k between 0 and $n-1$. Each instance takes as input the kth bit of *ip* and generates as output the kth bit of *op*. We assume here that I is some component, defined elsewhere, that can be verified using MDG.

If the specification for the component I is S, then the specification of the whole circuit is similar to the above.

PARALLEL_S (n, ip, op) =
  ∀k (0 ≤ k < n). S (ip[k], op[k])

The correctness theorem we wish to prove has the form:

∀n ip op.
  PARALLEL_I (n, ip, op) ⇒ PARALLEL_S (n, ip, op)

That is we wish to prove that the implementation of our parallel circuit implements the specification as defined above. Using the hybrid tool, the first steps in this proof can be done interactively in HOL. By rewriting with the definitions of the parallel components, resolving the assumptions and generalizing we are left to prove a similar correctness subgoal to that we started with, but about the submodule rather than the whole circuit.

We have used HOL proof to eliminate references to the parameter. We now have a concrete correctness goal that can thus be proved automatically using the hybrid tool by calling MDG, or by using the hybrid tool's hierarchical verification facilities. On proving it, the HOL subgoal manager will automatically use the theorem to prove a theorem about the original parameterized circuit. Thus by performing some interactive proof in HOL before switching to using the hybrid tools facilities for hierarchical verification we can verify parameterized circuits.

Since this parameterized pattern is common, it is worth writing a template definition and write a tactic that processes it automatically. The definition PARALLEL below embodies the pattern. It now takes the replicated component, R, (a specification or implementation) as an additional argument. Such hardware patterns have been suggested before, for example in [1].

PARALLEL R n ip op =
  ∀k (0 ≤ k < n). R (ip[k], op[k])

The original correctness theorem could now be stated as follows.

∀n ip op.
  PARALLEL (I, n, ip, op) ⇒ PARALLEL (S, n, ip, op)

The hybrid tool could be extended with a HOL proof function that proves this automatically.

Note that in MDG, inputs to components must be specified as individual bits. There is no concrete word abstraction, though an abstract variable could be used for this. However, in a parameterized circuit, the number of inputs and outputs can vary with the parameter. Thus a list or word abstraction is needed. This can be done in HOL. The inputs and outputs to PARALLEL are words of length n rather than a series of independent bit signals. However, the goals that are sent to MDG are about components that have single bit inputs and outputs only. More generally such word abstractions can appear in other, non-parameterized specifications. HOL rewriting can be used to rewrite concrete versions of the specifications using word inputs and outputs to versions using bit inputs and outputs that can be processed by MDG.

B. Ripple Carry Composition

A second common pattern for parameterized hardware is Rippling. This is the pattern embodied in the Ripple-carry Adder (see Figure 3). Each component still takes some input and produces some output. However part of that output (the carry for a ripple-carry adder) is input to the next component. This pattern can be specified using the following HOL definition where *ip* and *op* are inputs and outputs as before, *rin* is the input that starts the ripple (eg a carry-in) and *rout* the output that is the final ripple output (eg a carry-out) and r is a word that represents the series of values that are the ripple-in/out for each component:

RIPPLE R (n, (rin, ip), (op, rout), r) =
  ∀k (0 ≤ k < n). R (r[k], ip[k], op[k], r[k+1]) ∧
  (rin = r[0]) ∧ (r[n] = rout)

This definition is very similar to that given for the parallel circuits. The main difference is in the interconnection

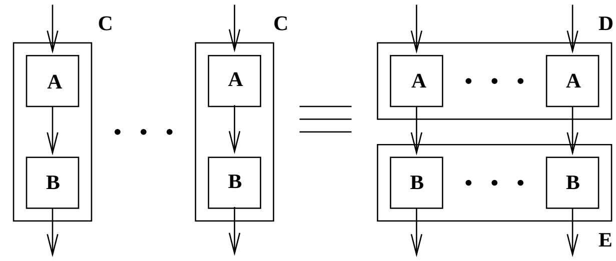

Fig. 4. Transforming a Circuit

pattern used. Now, the component being replicated, R has two inputs and two outputs. The kth instance, uses the kth bits of the input and r for its input and the kth bit of op for its output. However, it uses the k+1th bit of r for its ripple output. The last line of the definition just state that the ripple input rin is wired to bit 0 of r and bit n of r is wired to the ripple output, $rout$.

A correctness goal for a circuit consisting of such a component would have a similar form to that for the PARALLEL circuit:

$$\begin{array}{l} \forall \text{n rin rout ip r op.} \\ \quad \text{RIPPLE I (n, (rin, ip), (r, op, rout))} \Rightarrow \\ \qquad\qquad \text{RIPPLE S (n, (rin, ip), (r, op, rout))} \end{array}$$

As with the parallel circuit, HOL proof can strip it down to a correctness subgoal about the component which may then be verified automatically using the hybrid tool by calling MDG or by hierarchical proof.

A variety of other patterns, such as sequential connections, trees, etc., could be treated in a similar way. Each of the HOL proofs for such patterns have a very similar form. A HOL tactic could therefore be written to cope with each. Also if the pattern definitions such as RIPPLE are used it would be simple for the tactic to detect which pattern was being used and do the appropriate proof. This could be integrated into the current hierarchical verification tactic so that one tactic would be used whether the circuit was parameterized or not, to break it down to the next level of subgoals.

C. Transforming Circuits

A parameterized circuit description may not be given in a form that would make best use of the hybrid tool. Ideally, MDG should be called on the largest (so non-parameterized) circuit that it can verify automatically in reasonable time. However a parameterized circuit could be described in two ways (see Figure 4). One slice of the whole circuit could be described concretely and then it could be replicated at the top level. Alternatively, the most primitive components could be replicated directly, resulting in all submodules being parameterized. For example, with the circuit description on the left of Figure 4 high-level module C is replicated so it could be verified by MDG and the replication dealt with by HOL. With the equivalent description on the right, modules D and E are both parameterized so cannot be verified using MDG. Pure HOL must be used to verify the whole circuit. With the hybrid tool this could be overcome, however. HOL could be used to transform the description on the right to that on the left allowing MDG to be used in the verification. A correctness theorem about the original circuit would be obtained. This could be automated relatively easily.

IV. Conclusions

We have described how a HOL-MDG hybrid system can be used to verify parameterized circuits by using HOL to deal with the parameterized aspects of the circuit, leaving a concrete circuit that can be verified automatically by calling MDG, or if necessary hierarchically. By identifying particular patterns of generic circuit, such proofs could be conducted automatically, in a similar way to the way that the hierarchy tactic currently deals with submodules. The approach thus integrates with the tool's current approach to hierarchical verification. We illustrated the approach by considering simple parameterized patterns.

References

[1] M.D. Aagaard, M. Leeser, and P. Windley. Toward a super duper hardware tactic. In J.J. Joyce and C.H. Seger, editors, *Higher Order Logic Theorem Proving and Its Applications*, Lecture Notes in Computer Science 780, pages 400–413. Springer-Verlag, 1993.

[2] M.D. Aagaard, R.B. Jones, and C-J.H. Seger. Lifted-FL:A Pragmatic Implementation of Combined Model Checking and Theorem Proving. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Thery, editors, *Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science 1690, pages 323–340. Springer-Verlag, 1999.

[3] S. Balakrishnan and S. Tahar. A Hierarchical Approach to the Formal Verification of Embedded Systems Using MDGs. In *Proceedings IEEE 9th Great Lakes Symposium on VLSI*, Ann Arbor, Michigan, USA, March 1999, pages 284–287, IEEE Computer Society Press.

[4] F. Corella, Z. Zhou, X. Song, M. Langevin, and E. Cerny. Multiway Decision Graphs for Automated Hardware Verification. *Formal Methods in System Design*, 10(1):7–46, 1997.

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

[6] C. Kern and M. Greenstreet. Formal Verification in Hardware Design: A Survey. *ACM Transactions on Design Automation of Electronic Systems*, 4:123–193, 1999.

[7] S. Kort, S. Tahar and P. Curzon. Hierarchical Verification using an MDG-HOL Hybrid Tool. In *Proceedings of the IFIP Conference on Correct Hardware Design and Verification Methods (CHARME'2001)*, Lecture Notes in Computer Science, Springer Verlag, September 2001.

[8] R. Kumar, K. Schneider and T. Kropf. Structuring and Automating Hardware Proofs in a Higher-Order Theorem-Proving Environment. *Formal Methods in System Design*, 2:165–223, 1993.

[9] S. Rajan, N. Shankar, and M.K. Srivas. An Integration of Model-checking with Automated Proof Checking. In Pierre Wolper, editor, *Computer Aided Verification*, Lecture Notes in Computer Science 939, pages 84–97. Springer Verlag, 1995.

[10] K. Schneider and D.W. Hoffmann. A HOL Conversion for Translating Linear Time Temporal Logic to ω-Automata. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Thery, editors, *Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science 1690, pages 255–272. Springer Verlag, 1999.

[11] S. Tahar, X. Song, E. Cerny, Z. Zhou, M. Langevin and O. Ait-Mohamed. Modeling and Verification of the Fairisle ATM Switch Fabric using MDGs. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 18(7):956–972, 1999.