Three Approaches to Hardware Verification: HOL, MDG and VIS Compared

Sofiène Tahar[1], Paul Curzon[2], and Jianping Lu[1]

[1] ECE Department, Concordia University, Montreal, Canada.
{tahar, jianping}@ece.concordia.ca
[2] School of Computing Science, Middlesex University, London, UK
p.curzon@mdx.ac.uk

Abstract. There exist a wide range of hardware verification tools, some based on interactive theorem proving and other more automated tools based on decision diagrams. In this paper, we compare three different verification systems covering the spectrum of today's verification technology. In particular, we consider HOL, MDG and VIS. HOL is an interactive theorem proving system based on higher-order logic. VIS is an automatic system based on ROBDDs and integrating verification with simulation and synthesis. The MDG system is an intermediate approach based on Multiway Decision Graphs providing automation while accommodating abstract data sorts, uninterpreted functions and rewriting. As the basis for our comparison we used all three systems to independently verify a fabricated ATM communications chip: the Fairisle 4×4 switch fabric.

1 Introduction

Formal hardware verification techniques have established themselves as a complementary means to simulation for the validation of digital systems due to their potential to give very strong results about the correctness of designs. Many academic and commercial verification tools have emerged in recent years, which can be broadly classified into two contrasting formal verification techniques: interactive formal proof and automated decision graph based verification. This paper compares and contrast such tools using an Asynchronous Transfer Mode (ATM) switch fabric as a case study.

In the interactive proof approach, the circuit and its behavioral specification are represented in the logic of a general purpose theorem prover. The user interactively constructs a formal proof to prove a theorem stating the correctness of the circuit. Many different proof systems with a variety of interaction approaches have been used. In this paper we consider one such system: HOL [7], an LCF style proof system based on higher-order logic.

In the automated decision diagram approach the circuit is represented as a state machine. Techniques such as reachability analysis are used to automatically verify given properties of the circuit or verify machine equivalence. We consider the MDG [3] and VIS [2] tools. The VIS tool is based on a multi-valued extension of pure ROBDDs (Reduced Ordered Binary Decision Diagrams [1]).

The MDG system uses Multiway Decision Graphs [3] which subsume ROBDDs while accommodating abstract sorts and uninterpreted function symbols.

As the basis of our comparison, we used HOL, MDG and VIS to independently verify the Fairisle 4×4 switch fabric [9]. This is a fabricated chip which forms the heart of an ATM communication switch. The device, designed at the University of Cambridge is used for real applications in the Cambridge Fairisle network. It switches data cells from input ports to output ports within the ATM switch, arbitrating clashes and sending acknowledgments. It was not designed for the verification case study. Indeed, it was already fabricated and in use, carrying real user data, prior to any formal verification attempt.

Other groups have also used the 4×4 fabric as a case study. Schneider *et. al* [12] used a verification system based on the HOL theorem prover, MEPHISTO, to automate the verification of lower-level hardware modules against top-level block units of the fabric. Jakubiec and Coupet-Grimal are using the fabric in their work based on the Coq proof system [8]. Garcez also verified some properties of the 4×4 fabric using the HSIS model checking tool [6].

2 The Fairisle 4 by 4 Switch Fabric

The Fairisle switch forms the heart of the Fairisle network. It consists of a series of port controllers connected to a central switch fabric. In this paper, we are concerned with the verification of the switch fabric which is the core of the Fairisle ATM switch. The port controllers provide the interface between the transmission lines and the switch fabric, and synchronize incoming and outgoing data cells, appending control information to the front of the cells in a routing byte.

A cell consists of a fixed number of data bytes which arrive one at a time. The fabric switches cells from the input ports to the output ports according to the routing byte (header) which is stripped off before the cell reaches the output stage of the fabric. If different port controllers inject cells destined for the same output port controller (indicated by *route* bits in the routing byte) into the fabric at the same time, only one will succeed—the others must retry later. The routing byte also includes a priority bit (*priority*) that is used by the fabric during round-robin arbitration which gives preference to cells with the priority bit set. The fabric sends a negative acknowledgment to the unsuccessful input ports, and passes the acknowledgment from the requested output port to the successful one.

The port controllers and switch fabric all use the same clock, hence bytes are received synchronously on all links. They also use a higher-level cell frame clock—the *frame start* signal, which ensures the port controllers inject data cells into the fabric so routing bytes arrive together. The fabric does not know when this will happen. Instead, it monitors the *active* bit of the routing bytes: when any goes high the cells have arrived. If no input port raises the active bit throughout the frame then the frame is inactive—no cells are processed; otherwise it is active.

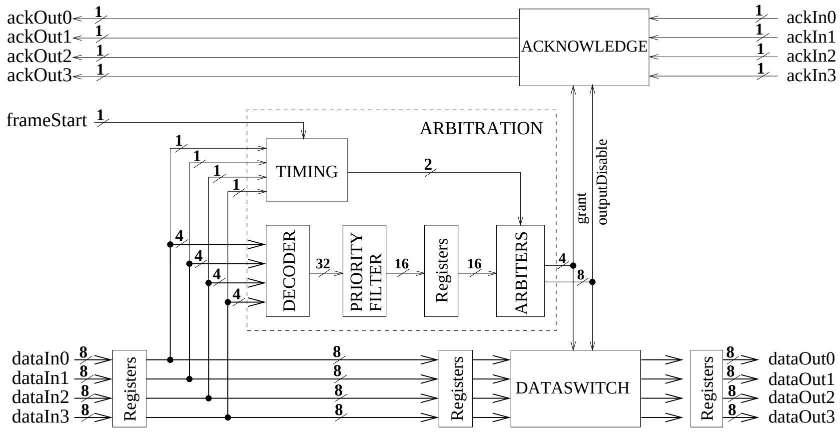

Fig. 1. The Fairisle ATM Switch Fabric

Figure 1 shows a block diagram of the 4×4 switch fabric. It is composed of an arbitration unit (timing, decode, priority filter and arbiters), an acknowledgment unit and a dataswitch unit. The timing block controls the timing of the arbitration decision based on the frame start signal and the time the routing bytes arrive. The decoder reads the routing bytes of the cells and decodes the port requests and priorities. The priority filter discards requests with low priority which are competing with high priority requests. It then passes the resulting request situation for each output port to the arbiters. The arbiters (in total four—one for each port) make arbitration decisions for each output port and pass the result to the other units with the grant signal. The arbiters indicate to the other units when a new arbitration decision has been made using the output disable signals. The dataswitch unit performs the switching of data from input port to output port according to the latest arbitration decision. The acknowledgment unit passes acknowledgment signals to the input ports. Negative acknowledgments are sent until a decision is made.

Each unit is repeatedly subdivided down to the logic gate level, providing a hierarchy of modules. The design has a total of 441 basic components including 162 1-bit flip flops. It is built on a 4200 gate equivalent Xilinx programmable gate array. The switching element can be clocked at 20 MHz and frame start pulses occur every 64 clock cycles.

3 The HOL Verification

In the first study, the Fabric was verified using the HOL90 Theorem Proving System [7]. This is a general purpose interactive proof system. It provides a range of proof commands of varying sophistication, including decision procedures. It is also fully user programmable, allowing user-defined, application-specific proof tools to be developed. The interface to the system is an ML interpreter. Proofs are input to the system as calls to ML functions. The system represents theorems by an ML abstract type. The only way a theorem can be created is by applying a small set of functions that correspond to the primitive rules of higher-order logic. More complex inference rules and tactics must ultimately call a series of primitive rules to do the work. User programming errors cannot cause a non-theorem to be erroneously proved: the user can have a great deal of confidence in the results of the system.

The verification was structured hierarchically following the implementation's module structure. The hierarchical, modular nature of the proof facilitated the management of its complexity. The structural and behavioral specifications of each module were given as relations in higher-order logic. This meant that a correctness statement could be stated using logical implication for "implements". I/O signals are represented by universally quantified variables holding functions over time. Internal signals are represented by existentially quantified variables. The overall correctness statement for the switch fabric has the (simplified) form:

```
∀aIn aOut dOut d frameStart.
  ENVIRONMENT frameStart d ⊃
   FABRIC4B4 ((d,frameStart,aIn), (dOut, aOut)) ⊃
    ∃last. FABRIC4B4_SPEC last ((d,frameStart,aIn), (dOut, aOut))
```

The correct operation of the fabric relies on an assumption about the environment. Cells must not arrive at certain times around a frame start. The relation `ENVIRONMENT` above, specifies this condition in a general way.

A correctness theorem of the above form was proved for each module stating that its implementation down to the logic gate level satisfied the specification.

In conducting the overall proof, the human verifier needs a very clear understanding of why the design is correct, since a proof is essentially a statement of this. Thus performing a formal proof involves a deep investigation of the design. It also provides a means to help achieve that understanding. Having to write formal specifications for each module helps in this way. Having to formulate the reasons why the implementation has that behavior gives much greater insight. In addition to uncovering errors, this can serve to highlight anomalies in the design and suggest improvements, simplifications or alternatives [4].

3.1 The Specifications

The structural specification of a design describes its implementation: the components it consists of and how they are wired together. The original designers of the fabric used the relatively simple Qudos HDL [5], to give structural descriptions of the hardware. This description was used to simulate the design prior to fabrication. The Xilinx netlist was also generated from this description. The descriptions used in the HOL verification were hand-derived from the Qudos descriptions. Qudos structural descriptions can be mimicked very closely in HOL up to surface syntax. However, the extra expressibility of HOL was used to simplify and generalize the description.

In HOL words of words are supported. Therefore, a signal carrying 4 bytes can be represented as a word of 4 8-bit words, rather than as one 32-bit signal. This allows more flexible indexing of bits, so that the module duplication operator `FOR` can be used. Arithmetic can also be used to specify which bit of a word is connected to an input or output of a component. For example, we can specify that for all i, the $2i$-th bit of an output is connected to the i-th bit of a subcomponent. This, again, meant that we could avoid writing essentially identical pieces of code several times, as was necessary in the Qudos specifications. When an additional module, used in several places is introduced, the verification task is reduced as that module need only be verified once.

It should be stressed that while the descriptions of the implementation were modified in the ways outlined above, no simplification was made to the implementation itself to facilitate the verification. The netlists of the structural specifications used corresponds to that actually implemented.

The behavioral specification against which the structural specification was verified describes the actual un-simplified behavior of the switch fabric. It is presented at a similar level of abstraction to that used by the designers, describing the behavior over a frame in terms of interval temporal operators (i.e. timing diagrams). The behavior of each output is specified as a series of interval specifications. The start and end times are specified in terms of the frame times given in an assumption. The values output are functions of the inputs and state at earlier times.

3.2 Time Taken

The module specifications (both behavioral and structural) were written prior to any proof. This took between one and two person-months. No breakdown of this time has been kept. Much of the time was spent in understanding the design. The behavioral specifications were more difficult. The specifier had no previous knowledge of the design. There was a good English overview of the intended function of the switch fabric. This also outlined the function of the major components. However, it was not sufficient to construct an unambiguous behavioral specification of all the modules. The behavioral specifications were instead constructed by analyzing the HDL. This was very time-consuming.

Approximately two person-months were spent performing the verification. Of this, one week was spent proving theorems of general use. Approximately three weeks were spent verifying the upper modules of the arbitration unit, and a further week was spent on the top two modules of the switch. 3–4 days were spent combining the correctness theorems of the 43 modules to give a single correctness theorem for the whole circuit. The remaining time of just over two weeks was spent proving the correctness theorems for the 36 lower level units. The proofs of the upper-level modules were generally more time-consuming for several reasons: there were more intervals to consider; they gave the behavior of several outputs; and those behaviors were defined in terms of more complex notions. They also contained more errors which severely hampered progress. Apart from standard libraries, the work did not build on previous theories.

It takes several hours of machine time on a Sparc 10 to completely rebuild the proofs from scratch by re-running the scripts in batch mode. Single theories representing individual modules generally take minutes to rebuild. A large proportion of the time is actually spent restarting HOL and loading in appropriate parent theories and libraries for each theory. In the initial development of the proof the machine time is generally not critical, as the human time is so much greater. However, since the proof process consists of a certain amount of replay of old proofs (e.g. when mistakes are made), a speed up would be desirable.

If changes are made to the design, it is important that the new verification can be done quickly. Since theorem proofs are very time consuming, this is especially important. This problem is attacked in several ways in the HOL approach: the proofs can be made generic; their modular nature means that only affected modules need to be reverified; and proofs of modules which have changed can often be replayed with only minor changes. After the original verification had been completed, several real variations on the design were also verified. Each took only a matter of hours or days.

One of the biggest disadvantages of the HOL system is that its learning curve is very steep. Furthermore, interactive proof is a time-consuming activity even for an expert. Much time is spent dealing with trivial details of a proof. Recent advances in the system such as new simplifiers and decision procedures (not used in this study) may alleviate these problems.

3.3 Errors

No errors were discovered in the fabricated hardware. Errors that had inadvertently been introduced in the structural specifications (and could just as easily have been in the implementation) were discovered. The original versions of the behavioral specifications of many modules contained errors.

A strong indication of the source of detected errors was obtained. Because each module was verified independently, the source of an error was immediately narrowed down to being in the current module, or in the specification of one of its submodules. Furthermore, because performing the proof involves understanding why the design is correct, the exact location of the error was normally obvious from the way the proof failed. For example, in one of the modules, two wires were inadvertently swapped. This was discovered because the subgoal `([T, F] = [F, T])` was generated in the proof attempt. One side of this equality originated from the behavioral specification and one from the structural specification. It was clear from the context of the subgoal in the proof attempt that two wires were crossed. It was also clear which signals were involved. It was not immediately clear, however, which specification (structural or behavioral) was wrong.

A further example of a discovered error concerned the time the grant signal (Figure 1) was read by the dataswitch. It was specified that the two bits of each grant signal were read on a single cycle. However, the implementation read them on consecutive cycles. This resulted in a subgoal of the form $grant\ t = grant\ (t + 1)$. No information was available in the goal to allow this to be proven, suggesting an error. On this occasion it was in the specification.

3.4 Scalability

In theory, the HOL proof approach is scalable to large designs. Because the approach is modular and hierarchical, increasing the size of the design does not necessarily increase the complexity of the proof. However, in practice the modules higher in the hierarchy generally (though not always) take longer to verify. This is demonstrated by the fact that two of the upper most modules took approximately half of the total verification time—a matter of weeks.

The extra time arises in part because there are more cases to consider. The situation is made worse if the interfaces between modules are left containing lots of low level detail. For example, for the switch fabric, low level modules required assumptions to be made about their inputs. These assumptions had to be dealt with in the proofs of higher level modules adding extra proof work manipulating and discharging them. If the proof is to be tractable for large designs, it is important that the interfaces between modules are as clean as possible. The interfaces of the Fairisle fabric could have been much simpler. We demonstrated this by redesigning the fabric with cleaner interfaces [4].

4 The MDG Verification

In the second study, the same circuit was verified using a decision graph approach. A new class of decision diagrams called *multiway decision graphs* (MDGs) was used to represent sets of states and the transition and output relations [3]. Based on a technique called *abstract implicit enumeration* [3], hardware verification tools have been developed which perform combinational circuit verification, safety property checking and equivalence checking of two sequential machines [3].

The formal system underlying MDGs is many-sorted first-order logic augmented with a distinction between abstract and concrete sorts. Concrete sorts have enumerations, while abstract sorts do not. A data value can be represented by a single variable of abstract sort, rather than by concrete boolean variables, and a data operation can be represented by an uninterpreted function (cross-operator) symbol. MDGs permit the description of the output and next state relations of a state machine in a similar way to the way ROBDDs do for FSMs. We call the model an *abstract state machine* (ASM) since it may represent an unbounded class of FSMs, depending on the interpretation of the abstract sorts and operators. For circuits with large datapaths, MDGs are thus much more compact than ROBDDs. As the verification is independent of the width of the datapath, the range of circuits that can be verified is greatly increased. Because of the use of uninterpreted functions, reachability analysis on MDGs may not terminate in some cases when circuits include some specific cyclic behavior [3]. We did not encounter this problem in the current study.

The MDG operators and verification procedures are packaged as MDG tools implemented in Prolog [3]. The ATM circuit we investigate here is an order of magnitude larger than any other circuit verified using MDGs.

We described the actual hardware implementation of the switch fabric at two levels of abstraction. We gave a description of the original Qudos gate-level implementation and a more abstract RTL description which holds for an arbitrary word width n. Using the MDG equivalence checking, we verified the gate-level implementation against the abstract (RTL) hardware model. Here the n-bit words of abstract sort of the latter were instantiated to 8 bits using uninterpreted functions which encode and decode abstract data to boolean data and vice-versa [13]. Besides, we used a few rewriting rules to map 8-bit constants of concrete sort to generic ones of abstract sort.

Starting from timing-diagrams describing the expected behavior of the switch fabric, we derived a complete high-level behavioral specification in the form of a state machine. This specification was developed independently of the actual hardware design and includes no restrictions with respect to the frame size, cell length and word width. Using implicit reachability analysis, we checked its equivalence against the RTL hardware model when both seen as abstract state machines. That is, we ensured that the two machines produce the same observable behavior by feeding them with the same inputs and checking that an invariant stating the equivalence of their outputs holds in all reachable states.

By combining the above two verification steps, we hierarchically obtain a complete verification of the switch fabric from a high-level behavior down to the gate-level implementation. Prior to the full verification, we also checked both behavioral and RTL structural specifications against several specific safety properties of the switch. Here, we combined an environment state machine with each switch fabric specification yielding a composed machine which represented the required platform for checking if the invariant properties hold in all reachable states of the specification [13]. Although the properties we verified do not represent the complete behavior of the switch fabric, we were able to detect several injected design errors in the structural model.

4.1 The Specifications

As with the HOL study, we translated the Qudos HDL gate-level description into a suitable HDL description; here a Prolog-style HDL, called MDG-HDL. As in the HOL study, extra modularity was added over the Qudos descriptions, while leaving the underlying implementation unchanged. A structural description is usually a (hierarchical) network of components (modules) connected by signals. The MDG-HDL comes with a large library of predefined, commonly used, basic components (such as logic gates, multiplexors, registers, bus drivers, ROMs, etc.). Multiplexors, registers and drivers can be modeled at the Boolean or the abstract level using abstract terms as inputs and outputs.

Hardware descriptions in MDG are very similar up to syntax to HOL. The data sorts of the interface and internal signals must always be specified. MDG does not provide a module replication facility, so repeated elements must be explicitly written out multiple times, nor an ability to structure words, so this description cannot be abstracted as in HOL.

Besides the gate-level description, we also provided a more abstract (RTL) description of the implementation which holds for arbitrary word width n. Here, the data-in and data-out lines are modeled using an abstract sort *wordn*. The *ac-*

tive, *priority* and *route* fields are accessed through corresponding cross-operators (functions). In addition to the generic words and functions, the RTL specification also abstracts the behavior of the dataswitch unit by modeling it using abstract data multiplexors instead of logic gates. We thus obtain a simpler implementation model of the dataswitch which reflects the switching behavior in a more natural way and is implemented with fewer components and signals. For more details about the abstraction techniques used, refer to [13].

MDG-HDL is also used for behavioral descriptions. A behavioral description is given by high-level constructs as ITE (If-Then-Else) formulas, CASE formulas or tabular representations. The tabular constructor is similar to a truth table but allows first-order terms in rows. It can be used to define arbitrary logic relations. In the MDG study, we gave the behavioral specification of the switch fabric in two different forms: 1) as a complete high-level behavioral state machine and 2) as a set of *safety* properties which reflect the essential behavior of the switch fabric as it is used in its environment.

The main behavioral description of the switch fabric was as an abstract state machine (ASM) which reflects its complete behavior under the assumption that the environment maintains certain timing constraints on the arrival of the frame start signal and headers. This ASM reproduces the exact behavior of the switch fabric during the initialization phase, the arrival of a frame start, the arrival of the routing bytes, and the end of the frame. The generation of the acknowledgment and data output signals is described by case analysis on the result of the round-robin arbitration. This is done in MDG-HDL using ITE and tabular constructs.

Although this ASM specification describes the complete behavior of the switch fabric, we also validated (in an early stage of the project) the fabric implementation by property checking. This is useful as it gives a quick verification result at low cost. We verified that the structural specification satisfies its requirements when the ATM switch fabric works under the control of its operating environment, i.e., the port controllers. We provided for this purpose a set of safety properties which reflect the essential behavior of the switch fabric, e.g., for checking of correct priority computation, circuit reset or data routing. We first modeled the environment as a state machine with one state variable s of enumerated (concrete) sort [1..68]. This allowed us to map the time points for initialization, frame start, header arrival and frame end to specific states. We then described the properties as invariants which should hold in all reachable states of the fabric model. Examples of properties are described in [13].

4.2 Time Taken

The translation of the Qudos design description to the MDG-HDL gate-level structural model was straightforward and took about one person-week. The description of the RTL structural specification including modeling required about one person-week. The time spent for understanding the expected behavior and writing the behavioral specification was about one person-week. The time taken for the verification of the gate-level description against the RTL model, including the adoption of abstraction mechanisms and correction of description errors, was about two person-weeks. The verification of the RTL structural specification against the behavioral model required about one person-week of work. The user time required to set up four properties, build the environment state machine, conduct the property checking on the structural specification and interpret the results was about one person-week. Checking of these same properties on the behavioral specification took about one hour. The average time for the introduction and verification of a design error was less than an hour. The experimental results are shown in Table 1. The CPU time given is for a SPARC station 10.

Like ROBDDs, the MDGs require a fixed node ordering. The variable ordering plays an important role as it determines the canonical attribute of the graphs and the size of the graphs which greatly affects its efficiency. A bad ordering easily leads to a state space explosion as occurred after an early ordering attempt. In contrast to VIS which provides heuristics for several node ordering techniques including dynamic ordering, node ordering in MDG has to be given by the user explicitly. This takes much of the verification time. On the other hand, unlike ROBDDs where all variables are Boolean, time must be spent assigning to every variable used an appropriate sort and type definitions must be provided for all functions. In some cases, rewrite rules may need to be provided to partially interpret the otherwise uninterpreted function symbols.

Because the verification is essentially automatic, the work re-running a verification for a new design is minimal compared to the initial effort since the latter includes all the modeling aspects. Much of the effort is spent on re-determining a suitable variable ordering. Depending on the kind of design changes adopted, the original variable ordering may need major changes for a modified design.

The MDG gate-level specification is a concrete description of the implementation. In contrast, the RTL structural and ASM behavioral specifications are generic. They abstract away from frame, cell and word sizes, provided the environment timing assumptions are kept. Design changes at the gate-level that still satisfy the RTL model behavior would hence not affect the verification against the ASM specification. For property checking, specific assumptions about the operating environment were made, (e.g. that the frame interval is 64 cycles). This is sound since the switch fabric will be used under the behest of its operating environment (the port controllers) which ensure this. While this reduces the verification cost, a disadvantage is that the verification must be completely redone if the operating environment changes, though only a few parameters have to be changed in the description of the (simple) environment state machine [13].

4.3 Errors

As with the HOL study, no errors were discovered in the implementation. For experimental purposes, we injected several errors into the structural specifications and checked them using either the set of properties or the behavioral model. Errors were automatically detected and automatically generated counter-examples were used to identify them. The injected errors included the main errors introduced accidently in the HOL study and in addition following three examples:

Table 1. Experimental Results for the MDG and VIS (Model checking) Verifications

	MDG			VIS		
Verification	CPU Time (s)	Memory (MB)	MDG Nodes	CPU Time (s)	Memory (MB)	BDD Nodes
Gate-Level to RTL	183	22	183,300			
RTL to Beh. Model	2920	150	320,556			
P1: Data Output Reset	202	15	30,295	3593	32	93,073,140
P2: Ack. Output Reset	183	15	30,356	833	5	28,560,871
P3: Data Routing	143	14	27,995	3680	41	79,687,78
P4: Ack. Output	201	15	33,001	415	5	4,180,124
Error (i)	20	1	2,462	83	4	1,408,477
Error (ii)	1300	120	150,904	49	2	250,666
Error (iii)	1000	105	147,339	15	1	85,238

(i) We exchanged the JK flip-flop inputs that produce the output disable signal. This prevented the circuit from resetting. (ii) We used, at one point, the priority bit of input port 0 instead of input port 2. (iii) We used an AND gate instead of an OR gate. Experimental results for these errors, when checked by verifying the RTL model against the behavioral specification, are given in Table 1.

While checking properties on the hardware structural description, we also discovered errors that we mistakenly introduced in the structural specifications. However, we were able to easily identify and correct them using the counter-example facility of the MDG tools. Also, during the verification of the gate-level model, we found a few errors in the description that were introduced during the translation from Qudos HDL to MDG-HDL. These were easily removed by comparing both descriptions, as they included the same collection of gates. Finally, many trivial typing errors were found at an early stage by the error messages output after each compilation of the specification's components.

4.4 Scalability

Like all FSM-based verification, MDG proof is not directly scalable to large designs due to the state space explosion. Unlike other approaches, MDGs can cope with datapath complexity as they use data of abstract sort and uninterpreted functions. Still, a direct verification of the gate-level model against the behavioral model or even against the set of properties was practically impossible. We overcame this by providing an abstract RTL structural specification which we instantiated for the verification against the gate-level model. To handle large designs, major effort is required to set up the model abstraction levels.

5 The VIS Verification

We also verified the fabric using VIS [2], another decision diagram based tool. It integrates the verification, simulation and synthesis of finite-state hardware systems. It uses a Verilog front-end and supports fair CTL model checking, language emptiness checking, combinational and sequential equivalence checking, cycle-based simulation, and hierarchical synthesis. Its fundamental data structure is a multi-level network of latches and combinational gates. The variables of a network are multi-valued, and logic functions over these variables are represented by an extension of BDDs: multi-valued decision diagrams.

VIS operates on the intermediate format BLIF-MV. It includes a compiler from Verilog to BLIF-MV. It extracts a set of interacting FSMs that preserves the behavior of the Verilog program defined in terms of simulated results. Through the interacting FSMs, VIS performs fair CTL model checking under Buchi fairness constraints. The language of a design is given by sequences over the set of reachable states that do not violate the fairness constraint. Also VIS can check the combinational and sequential equivalence of two designs. Sequential verification involves building the product FSM, and checking whether a state where the values of corresponding outputs differ can be reached from the set of initial states of the product machine. If model checking or equivalence checking fails, VIS reports the failure with a counter-example.

We translated the original Qudos HDL gate-level description of the switch fabric into Verilog HDL. We also derived a complete high-level behavioral specification in the form of a finite state machine according to the timing diagrams describing the expected behavior of the switch fabric. This specification was developed independently of the actual hardware design and uses a different design hierarchy to the structural one. Using these Verilog specifications, we attempted to obtain a complete verification of the switch fabric from a high-level behavioral specification down to the gate-level implementation through equivalence checking. This verification was similar to that in the MDG case. However, it did not succeed in VIS due to state space explosion. We therefore attempted to separately verify the submodules of the fabric based on the same design hierarchy as the structural one. This is similar to the HOL study, and involved writing separate Verilog RTL behavioral specifications for each submodule. We succeeded in verifying the equivalence of the behavioral specification of each submodule and its corresponding structural specification by VIS sequential equivalence checking. Through this verification, we checked that the implementation of each submodule satisfies its specification. Unlike the HOL verification, we could not verify the correctness of the connections among the submodules of the switch fabric. For real designs, this step would be useful to verify if the logic synthesis is correct.

As an alternative to equivalence checking, we attempted model checking of the switch fabric. Unlike MDG, model checking is the main verification approach in VIS. As for the MDG approach an environment state machine was needed [11]. To ease the model checking we compressed the 68 states into 7 states. Again we failed to verify the whole switch fabric due to the state space explosion. We succeeded in model checking a simplified fabric with its datapath and control path reduced from 8 bits to the minimum 1 bit and 4 bits, respectively.

Both behavioral and structural specification were written in Verilog, so we were able to perform their simulation in Verilog-XL directly. It was very useful for detecting some syntax and semantic errors of the descriptions before performing equivalence or model checking. In addition, we extracted some safety properties from the generalization of simulation vectors. These safety properties were further used in model checking, enabling the detection of design errors that were omitted by simulation. The Verilog-XL graphical interface also eased the analysis of counter-examples which were generated by model and equivalence checking. Furthermore, as the RTL behavioral specification was written in Verilog, we were able to synthesize the structural specification with some timing constraints directly using the Synopsys Design Compiler. We performed equivalence checking between the submodules of the RTL behavioral specification and the submodules of the synthesized structural one to ensure the correctness of the synthesis.

5.1 The Specifications

The Verilog structural specification of the fabric is very similar to the other descriptions. A big advantage of the VIS Verilog front-end is the ease of importing existing (industrial) designs with no extra overhead of manual translation. Moreover, it allows the direct interaction of VIS with other commercial tools for simulation and synthesis. However, the fabric structure had to be reduced to 4 bits and the datapath further to one bit to enable the model checking procedure to terminate. The control path could not be reduced below 4 bits as the data includes the header control information. For more details about the abstraction and reduction techniques adopted refer to [11].

We gave the behavioral specification of the fabric in two forms: an RTL description as a state machine of the whole fabric and a set of liveness and safety properties covering its essential behavior. In addition, as with HOL, behavioral specifications of submodules of the design hierarchy were developed. VIS-Verilog HDL is used for behavioral specification. It contains two new features over standard Verilog: a nondeterministic construct, $ND, to specify non-determinism on wire variables; and symbolic variables which use an enumerated type mechanism similar to the one available in the MDG system.

Unlike in MDG, we extensively used property checking to verify the fabric in VIS as it is optimized for model checking. Moreover, thanks to the expressiveness of CTL, properties can be defined more easily in VIS. With MDG a property (invariant) is described in MDG-HDL using ITE and tabular constructs. Before using model checking to verify the overall behavior of the switch fabric, we set up an environment state machine and developed a set of properties. The nondeterministic construct ($ND) of VIS-Verilog HDL eases the establishment of an environment state machine. We used it to express the inputs of the switch fabric. CTL can represent both safety and liveness properties. The latter can be used to detect deadlock or livelock which is difficult using simulation. 58 CTL properties were verified. We first verified a number of safety properties including all those used in the MDG study. In addition, we verified many CTL liveness properties. Example properties that we checked on the fabric model can be found in [11].

Due to the state space explosion, we succeeded in checking only a few properties on the abstracted fabric directly. Instead we adopted several techniques that divide a property into sequentially or parallelly related sub-properties in a similar manner to the compositional reasoning proposed in [10]. Details about the specific property division techniques we used are reported in [11].

5.2 Time Taken

The translation of the Qudos structural description to Verilog was straightforward taking about three person-days. The time spent for understanding the expected behavior and writing the behavioral specification was around ten person-days. The time taken for the simulation of both RTL behavioral and structural specification in Verilog-XL, including the development of test-bench files, was about three person-days. The verification of the RTL behavioral specification against the structural specification was done automatically, and took around one person-day. The user time required to set up 58 CTL properties, build the related environment state machine, construct the appropriate property division and conduct the model checking took approximately three weeks. The injection and verification of an error took less than one hour.

The experimental results of model checking, which were obtained on a SPARC station 20, are shown in Table 1. VIS generates comparatively more BDD nodes than the MDG system does. This is due to the data abstraction within MDG that is absent in VIS. The equivalence checking of the whole switch fabric ran for three days before running out of memory. The same problem occurred with the dataswitch module. Equivalence checking of the arbitration module was successful but it took two days of machine time. The lower level modules such as the timing unit were verified in seconds. We also failed to verify the properties on the original switch fabric after two days of machine time. Finally we reduced the datapath of the switch fabric from 8 to 1 bit. The successful model checking results in Table 1 are based on this reduced model.

Since VIS is based on ROBDDs, the node ordering has a dramatic influence on the speed of both equivalence checking and model checking. Unlike MDG, VIS provides dynamic ordering facilities to reduce the cost of manual variable ordering.

The experimental results given in Table 1 were obtained using VIS dynamic ordering. It is to be noted that in some cases a manually optimized ordering, e.g., an interleaved order of the bits of the data words, would have enhanced the VIS verification.

We also applied cascade and parallel property divisions (practical approaches to compositional reasoning). Using these techniques, we enhanced the model checking by up to 200 times. However, we had to establish environment state machines and abstract the circuit first. The derivation of reduced models from the original structure and the division of properties was very time consuming. For a cascade property division, we built a new partial environment state machine for each target sub-circuit. For parallel property division, we disassembled a circuit at different symmetric locations and later composed the properties.

5.3 Errors

As in the HOL and MDG studies, no errors were discovered in the switch fabric implementation. We injected the same errors as for MDG into the implementation and checked them using either model checking or equivalence checking. Experimental results are reported in Table 1. Like MDG, VIS provides a counter-example generation facility to help identify the source of design errors. Injected errors were hence automatically detected and further viewed graphically with Verilog-XL. Through checking the equivalence between the RTL behavioral and the structural specifications of the submodules, we discovered errors that we mistakenly introduced in the structural specification. Also, during model checking, we found connection errors that were mistakenly introduced in the RTL behavioral and structural specifications. We easily identified and corrected these errors from the counter-examples.

5.4 Scalability

The VIS proof approach is not directly scalable to large designs due to state space explosion. To solve this problem the datapath complexity must be decreased by abstraction and reduction. In a large design like the switch fabric, we also had to apply compositional reasoning [10]. The environment state machine must imitate the behavior of the models which are associated with the target model. It must also have fewer components than the original models. Consequently, the environment state machine is especially hard to develop when the concurrent interaction between the target model and its associated models is complex.

6 Conclusions

The structural descriptions are very similar. HOL provides significantly more expressibility allowing more natural specifications. Some generic features were included in the MDG description that were not in the HOL description. This could have been done with minimal effort, however. Due to its Verilog front-end, (commercial) designs can be imported into VIS with no extra overhead of a manual translation, which is one reason for its popularity. This also allows direct interaction with commercial tools for simulation and synthesis.

The behavioral descriptions are totally different. The MDG and VIS specifications are based on a state machine model while HOL's is based on interval operators explicitly describing the timing behavior in terms of frames corresponding to whole ATM cells arriving. In the MDG and VIS specifications the frame abstraction is not used: the description is firmly at the byte level. Verilog allows direct testing of the specifications using commercial simulation facilities, however. Unlike Verilog descriptions, HOL's higher-order logic and MDG-HDL descriptions are not directly executable. All describe the behavior in a clear and comprehensive form. Writing the behavioral specifications took longer in HOL and VIS, as separate specifications were needed for each module. In MDG this was not necessary as the whole design was verified in one go.

An advantage of MDG and VIS is that a property specification is easy to set up and verify. For both systems it was necessary to introduce an environment state machine in order to restrict the possible inputs to the switch fabric. It is verified that the specification satisfies its requirements under specific working conditions. It can greatly reduce the full verification cost by catching errors at an early stage. In this respect VIS, with its very efficient CTL based model checking, outperforms its MDG counterpart. Properties are easier to describe in CTL than are invariants in the MDG system. Currently, MDG tools do not provide CTL property specification or liveness property verification. Work on the integration of a recently developed MDG model checking algorithm based on a restricted first-order temporal logic (Abstract CTL–ACTL) is ongoing.

The HOL verification was much slower, taking several months. This time includes the verification of each of the modules and of their combination. Much of the time was spent on the connection of the highest level modules (which VIS failed on). Using HOL, many lemmas had to be proved and much effort was required to interactively create the proof scripts. For example, the time spent verifying the dataswitch was about three days. The proof script was over 500 lines long (17 KB). The MDG and VIS verifications were achieved automatically without the need of a proof script. For MDG, however, careful management of the MDG node ordering was needed (which currently has to be done manually). This could take hours or a few days of work. In contrast, VIS provides several options for variable ordering heuristics which eliminate the ordering overhead. However, major effort was spent here developing abstract models of the switch fabric units to manage the state explosion of the boolean representation. Furthermore, the HOL and MDG verifications succeeded in verifying the whole switch fabric but VIS failed to verify even the smallest 1-bit datapath version of the fabric using equivalence checking. Additional time was spent hierarchically verifying submodules as with HOL but their combination could not be verified.

In all the approaches, the work needed to verify a modified design is greatly reduced once the original has been verified. MDG and HOL allow generic verification to be performed (e.g. word sizes are unspecified), though HOL is more flexible. No generic verification is possible in VIS. Because MDG and VIS are automated and fast, re-verification times are largely the time taken to modify the specifications and, for MDG, to find a new variable ordering. With HOL the behavioral specifications of many modules and the proof scripts themselves may need to be modified. For model checking in VIS, new environment machines, and model abstraction and reduction techniques may be required.

An advantage of the HOL approach over the others is the confidence in the tool the LCF approach offers. Although the VIS (and to a certain extent the MDG) software package has been successfully tested on several benchmarks and has been considerably improved, they cannot guarantee the same level of proof security as HOL. Compared to MDG, VIS is a more mature tool. It is implemented in a well-engineered fashion in C as compared to the prototype implementation of MDG in Prolog. Moreover, VIS is very widely used in both academia and industry, giving confidence in its correctness.

Table 2. Summary of the Comparison

Area	Feature	HOL	MDG	VIS
Specification	Behavioral Specification - Time Taken		++	+
	Structural Specification - Time Taken	+	+	++
	Behavioral Specification - Expressibility	++	+	
	Structural Specification - Expressibility	++		+
Verification	Full Verification Completed	++	+	
	Machine Time Taken		+	+
	Total Verification Time Taken		++	+
	Verification Time for Design Modifications	+	+	+
	Property Checking		+	++
	Equivalence Checking		++	+
	Scalability	++		
	Confidence in Tool	++		+
Errors	Detect Errors	++	++	++
	Locating Errors	+	++	++
	Avoid Error Introduction	+	+	+
	False Error Reports		++	+
Design	Impart Understanding of Design	++	+	+
	Suggesting Design Improvements	++	+	+
	Commercial Front End			++

All the approaches highlight errors, and help determine their location. However, the way this information manifests itself differs. VIS and MDG are more straightforward, outputting a trace of the input sequence that leads to the erroneous behavior. Errors are detected automatically and can be diagnosed with the help of the counter-example facility. In addition, due to its front-end, VIS counter-examples can be analyzed using commercial tools such as XL-Verilog. In HOL, errors manifest themselves as unprovable goals. The form of the goal, the context of the proof and the verifier's understanding of the proof are combined to track down the location, and understand its cause.

With the MDG (and to a certain extent VIS) verification approach the verifier does not need be concerned with the internal structure of the design being verified. This means that no understanding of the internals is obtained by doing the verification. In contrast, with HOL, a very detailed understanding of the internal structure is needed. The verifier must know why the design works the way it does. The process of doing the verification helps the verifier achieve this understanding. This means that internal idiosyncrasies in the implementation are likely to be spotted, as are other potential improvements.

A summary of the main comparison points is given in Table 2. Each system is given a rough rating of either "++", "+" or nothing to indicate how favorably the system comes out with respect to that feature. In conclusion, the major advantages of HOL are: the expressibility of the specification language; the confidence afforded in its results; the potential for scalability and the insight into the design that is obtained. The strength of MDG and VIS is in their speed; their relative ease of use and their error detection capabilities. MDG has the advantage of using abstract data types and uninterpreted functions with a rewriting facility, hence allowing larger circuits to be verified—but with the drawback that an MDG verification may not terminate in some cases. VIS is a very efficient model checker supporting the CTL expressiveness for both liveness and safety properties. Moreover, VIS outperforms MDG due to its maturity in the use of efficient graph manipulation techniques. The VIS Verilog front-end and mature C implementation make VIS very attractive to industry.

Acknowledgments

We are grateful for the help of X. Song and E. Cerny at the Univ. of Montreal, I. Leslie and M. Gordon at Cambridge, R. Brayton at Berkeley, F. Somenzi at Colorado, H. Thimbleby at Middlesex, Z. Zhou at Texas Instruments and M. Langevin at Nortel.

References

1. R. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Trans. on Computers*, C-35(8):677–691, 1986.
2. R. Brayton et. al. VIS: A System for Verification and Synthesis In R. Alur and T. Henzinger, eds, *Computer Aided Verification*, LNCS 1102, 428–432, Springer-Verlag, 1996.
3. 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.
4. P. Curzon and I.M. Leslie. Improving Hardware Designs whilst Simplifying their Proof. *Designing Correct Circuits*, Workshops in Comp., Springer-Verlag, 1996.
5. K. Edgcombe. *The Qudos Quick Chip User Guide*. Qudos Limited.
6. E. Garcez and W. Rosenstiel. The Verification of an ATM Switching Fabric using the HSIS Tool. In *IX Brazilian Symp. on the Design of Integrated Circuits*, 1996.
7. M.J.C. Gordon and T.F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher-order Logic*. Cambridge University Press, 1993.
8. L. Jakubiec, S. Coupet-Grimal, and P. Curzon. A Comparison of the Coq and HOL Proof Systems for Specifying Hardware. In E. Gunter and A. Felty, eds, *Theorem Proving in Higher Order Logics: Short Presentations*, 63–78, 1997.
9. I.M. Leslie and D.R. McAuley. Fairisle: An ATM Network for the Local Area. *ACM Communication Review*, 19(4):327–336, 1991.
10. D.E. Long. Model Checking, Abstraction and Compositional Verification. *Ph.D thesis*, Carnegie Mellon University, July 1993.
11. J. Lu and S. Tahar. Practical Approaches to the Automatic Verification of an ATM Switch Fabric using VIS. In *Proc. IEEE Great Lakes Symp. on VLSI*, 368–373, 1998.
12. K. Schneider and T. Kropf. Verifying Hardware Correctness by Combining Theorem Proving and Model Checking. In J. Alves-Foss, editor, *Higher Order Logic Theorem Proving and Its Applications: Short Presentations*, 89–104, 1995.
13. S. Tahar, Z. Zhou, X. Song, E. Cerny, and M. Langevin. Formal Verification of an ATM Switch Fabric using Multiway Decision Graphs. In *Proc. IEEE Great Lakes Symp. on VLSI*, 106–111, 1996.