A Comparison of MDG and HOL for Hardware Verification

Sofiène Tahar[§] and Paul Curzon[‡]

[§]University of Montreal, IRO Department, Canada.
[‡] University of Cambridge, Computer Laboratory, UK.

Abstract. Interactive formal proof and automated verification based on decision graphs are two contrasting formal hardware verification techniques. In this paper, we compare these two approaches. In particular we consider HOL and MDG. The former is an interactive theorem proving system based on higher-order logic, while the latter is an automatic system based on Multiway Decision Graphs. As the basis for our comparison we have used both systems to independently verify a fabricated ATM communications chip: the Fairisle 4 by 4 switch fabric.

1 Introduction

Formal hardware verification techniques are starting to attract widespread interest due to their potential to give very strong results about the correctness of designs. Two very different forms of formal verification have arisen: interactive proof and automated decision graph techniques. The aim of this paper is to compare and contrast these two approaches.

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

In the automated decision graph approach the circuit is represented as a decision diagram, and techniques such as reachability analysis are used to automatically verify given properties of the circuit or verify machine equivalence. We consider the MDG system. It uses a new class of decision graphs called Multiway Decision Graphs [3]. They subsume the class of Bryant's Reduced Ordered Binary Decision Diagrams [1] while accommodating abstract sorts and uninterpreted function symbols.

As the basis of our comparison of HOL and MDG, we have used both to independently verify the Fairisle [10] 4 by 4 switch fabric[1]. This is a fabricated chip which forms the heart of an ATM communication switch. It does the actual switching of data cells from input ports to output ports within the switch,

[1] See URL http://www.cl.cam.ac.uk/Research/HVG/atmproof/ for more details of Fairisle, the 4 by 4 fabric design and both the MDG and HOL verification projects.

arbitrating clashes and sending acknowledgments. It was not designed for the verification case study. Indeed it was fabricated and in use, carrying real user data, prior to any formal verification attempt.

There has been a vast amount of work on formal hardware verification. We mention here only that which is directly related to our study on verifying network hardware components.

J. Herbert [8] used HOL to formally verify the ECL chip: a local area network interface which formed part of the Cambridge Fast Ring. This is of roughly similar complexity to the circuit we considered, though our HOL proof took less time, demonstrating the increased maturity of the system.

B. Chen *et. al* at Fujitsu Digital Technology Ltd. [2] verified an ATM circuit that makes high-speed switching operations at 156 MHz and consists of about 111K gates. When the circuit was manufactured it showed an abnormal behavior under certain circumstances. Using the SMV tool [11], the authors identified the design error by checking some properties expressed in Computational Tree Logic [11]. Due to the restriction of the Boolean computation used by SMV and in order to avoid a state space explosion, they had to abstract the data width of addresses from 8 bits to 1 bit, and the number of addresses in the Write Address FIFO from 168 to 5. Although the design error was diagnosed, there is no proof showing that the abstracted circuit was itself correct.

K. Schneider *et. al* [12] formally verified the Fairisle 4 by 4 switch fabric using a verification system based on the HOL theorem prover: MEPHISTO. They described the structure of each of the modules used in the hardware design hierarchically down to the gate level and provided their behavioral specifications using hardware formulas. Although they automated the verification of lower-level hardware modules which implement the top-level block units, they have not accomplished the complete verification of the intended overall behavior of the switch fabric against its implementation.

The outline of the paper is as follows. In Section 2 we give a brief overview of the particular hardware considered: the Fairisle 4 by 4 switch fabric. We describe its verification using HOL in Section 3 and using MDG in Section 4. For each we overview the verification method, tools and our experiences on this case study. Finally, in Section 5 we draw conclusions. Since we have considered only a single case study it should be noted that such conclusions cannot be definitive.

2 The Fairisle 4 by 4 Switch Fabric

The Fairisle switch forms the heart of the Fairisle network. It consists of three types of components: input port controllers, output port controllers and a switch fabric. Each port controller is connected to a transmission line and to the switch fabric. The port controllers synchronize incoming and outgoing data cells, appending control information to the front of the cells in a routing byte (header). This byte is stripped off before the cell reaches the output stage of the fabric. 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

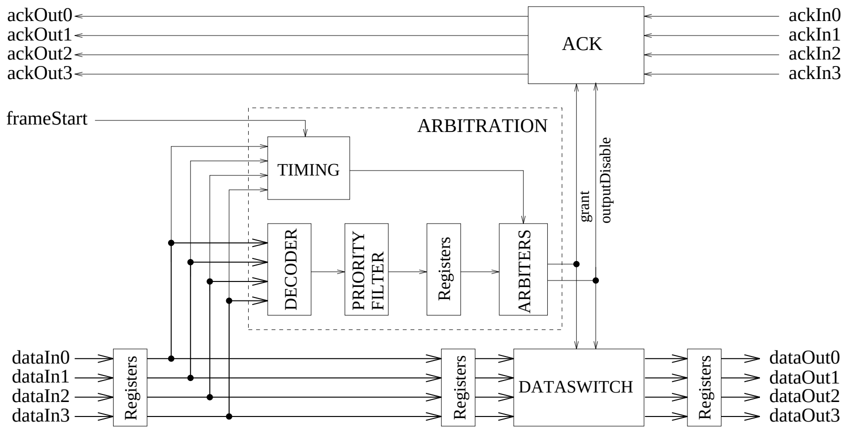

Fig. 1. The Fairisle Switch Fabric

byte. 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, then only one will succeed. The others must retry later. The routing byte also includes a priority bit that is used by the fabric during arbitration. It takes place in two stages. First, high priority cells are given precedence, and for the remaining cells the choice is made on a round-robin basis. The input controllers are informed of whether their cell was successful using acknowledgment lines. The fabric sends a negative acknowledgment to the unsuccessful input ports, but passes the acknowledgment from the requested output port to the successful input port. 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. It ensures that the port controllers inject data cells into the fabric synchronously so that the routing bytes arrive at the same time. In this paper, we are concerned with the verification of the switch fabric which is the core of the Fairisle ATM switch.

The behavior of the switch fabric is cyclic. In each cycle or frame, it waits for cells to arrive, reads them in, processes them, sends successful ones to the appropriate output ports, and sends acknowledgments. It then waits for the arrival of the next round of cells. The cells from all the input ports start when the *active* bit of any one of their routing bytes goes high. The fabric does not know when this will happen. However, all the input port controllers must start sending cells at the same time within the frame. If no input port raises the active bit throughout the frame then the frame is inactive–no cells are processed. Otherwise it is active.

Figure 1 shows a block diagram of a 4 by 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 decision with respect to the frame start signal and the time the routing byte arrives.

The decoder reads the routing bytes of the cells and decodes the port requests and priorities. The priority filter discards requests with low priority and those from inactive inputs. It then passes the actual 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. Using the output disable signals, the arbiters indicate to the other units when a new arbitration decision has been made. The dataswitch unit performs the actual switching of data from input port to output port according to the latest arbitration decision (the grant signals). The acknowledgment unit passes appropriate acknowledgment signals to the input ports. Negative acknowledgments are sent until a decision is made.

Each of these units is repeatedly subdivided down to the logic gate level, providing a hierarchy of modules. The design has a total of 441 basic components (a multiple input logic gate or single bit flip flop). It is built on a 4200 gate equivalent Xilinx programmable gate array. The switching element can be clocked at 20 MHz and currently frame start pulses occur every 64 clock cycles. The hardware was originally described in the Qudos HDL hardware description language which was used for generating the Xilinx netlist. The Qudos simulator was used to perform the original (non-formal) validation.

3 The HOL Verification

The HOL90 theorem proving system is an LCF style theorem prover for higher-order logic [7]. The original HOL system was intended as a tool for hardware verification. However, it is actually a general purpose proof system that has subsequently been used in a wide variety of application areas. Proofs are input to the system as calls to Standard ML functions. Because of the use of an abstract type to represent theorems, the user can have a great deal of confidence in the results of the system. Programming errors cannot cause a non-theorem to be erroneously proved unless they are in a few simple functions corresponding to the primitive inference rules of the system.

The verification of the 4 by 4 switch fabric used standard techniques [6]. We give only a brief overview. Structural and behavioral specifications of each module were given in higher-order logic. A correctness theorem was then independently proved for each module that its implementation satisfied (implied) the specification. Finally, the correctness theorems for the separate modules were used to prove a correctness theorem for the whole design. The verification was conducted down to the level of the basic logic gates used by the simulator. As in the simulator they were described behaviorally rather than structurally. The modular nature of the proof facilitates the management of the complexity of large designs.

In conducting the proof, the 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, but 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 [5].

The Structural Specifications No simplification was made to the implementation to facilitate the verification. While some simplification was made to the surface description (such as grouping components into extra modules), the netlists of the structural specifications used corresponded to that actually implemented. The basic building blocks used were logic gates and single bit registers. These corresponded to the basic units of the simulator used by the designers. 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. For example, 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 4 separate signals or as one 32-bit signal. This allows more flexible indexing of bits, so that the module duplication operator `FOR` can be used. To illustrate the expressibility of HOL, we consider the Qudos HDL description of the following multiplexing component of the dataswitch–DMUX4T2:

```
DEF DMUX4T2(d[0..3],x:IN;dOut[0..1]:IO); xBar:IO;
BEGIN
      Clb:=XiCLBMAP5i20(d[0..1],x,d[2..3],dOut[0..1]);
      InvX:= XiINV(x,xBar);
      B[0]:= AO(d[0],xBar,d[1],x,dOut[0]);
      B[1]:= AO(d[2],xBar,d[3],x,dOut[1]);
END;
```

The `Clb` statement is a dummy declaration providing information about the way the component design should be mapped into a Xilinx gate array. `XiINV` is an inverter and the `AO` components are AND-OR logic gates. Using HOL, this module can be expressed as follows with only a single occurrence of `AO` rather than two as in the Qudos version.

```
DMUX4T2((d,x),dOut) = LOCAL xBar.
      XiINV(x,xBar) ∧
      FOR i :: TO 2 .
       AO((SBIT 0 (SBIT i d),xBar,SBIT 1 (SBIT i d), x), SBIT i dOut)
```

In HOL, 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 a single module could be used instead of needing to write essentially identical pieces of code several times.

The Behavioral Specifications 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 timing diagrams represented as interval temporal operators. Within the interval, the values output are functions of the input values and state at earlier times.

As an example, consider the specification for the acknowledgment signal on a frame where cell headers arrive at time t_h. The predicate AFRAME specifies that we are dealing with intervals corresponding to such active frames. The *ackOut* signal must be zeroed until time $t_h + 3$. Thereafter, its value depends on the arbitration decision made. This depends on the value of the data injected into the fabric at time t_h (the header), the value of the last arbitration decision, and the value of the acknowledgments coming in from the output ports. This behavior is specified by a function argument to the interval operator DURING. We omit the details here for the purposes of exposition.

$$
\begin{array}{l}
(\texttt{AFRAME}\ t_s\ t_h\ t_e\ fs\ \ldots) \supset \\
\qquad \texttt{STABLE}\ (t_s+1)\ (t_h+3)\ ackOut\ (\texttt{ZEROW}\ \ldots) \wedge \\
\qquad \texttt{DURING}\ (t_h+3)\ (t_e+1)\ ackOut \\
\qquad\ (\lambda t.\ \ldots\ (d\ t_h)\ \ldots\ (last\ (t_h+2))\ \ldots\ (ackIn\ t)\ \ldots)
\end{array}
$$

The correct operation of the fabric relies on an assumption that the environment maintains the frame structure of repeated frame start signals and that cells will not arrive at certain times within a few clock cycles of the frame start. The cycles on which the cells cannot arrive was specified and verified precisely.

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 structural specifications were adapted directly from the Qudos HDL. 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. While it gave a good introduction, 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 3 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. The verifier had not previously performed a hardware verification, though was a competent HOL user. Apart from standard libraries, the work did not build directly on previous theories.

The machine time taken to completely rebuild the proofs from scratch by rerunning the scripts in batch mode is several hours on a Sparc 10. Single theories representing individual modules generally take minutes to rebuild. 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, 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 proof is very time consuming this is especially important. This 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. While the 4 by 4 switch fabric took several months to specify and verify, modified versions took only a matter of hours or days [4]. Generic proofs were not used to as great an extent as was possible in this study as it was generally easier to reason about specific values than general ones. Furthermore, there were many different ways that the design and its submodules could be made generic. It was not clear which if any of these might be utilized in subsequent designs. It thus seemed sensible in the first instance to stick closely to the actual design. Indeed the limited ways that the proofs were made generic turned out not to cover design changes incorporated into later designs.

One of the biggest disadvantages of the HOL system is that its learning curve is very steep. Furthermore, interactive proof is generally a time-consuming activity even for the expert. Much time is spent dealing with trivial details of a proof. Recent advances in the system such as new simplifiers and decision procedures may alleviate these problems. However, more work is needed to bring the level of interaction with the system closer to that of an informal proof.

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 dataswitch 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 proof attempt that two wires had been swapped and also which signals they were from the context of the subgoal. It was not immediately clear in which specification they had been swapped.

A further example of an error that was discovered concerned the time the grant signal was read by the dataswitch. It was specified that the two bits of the grant signal from each arbiter 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. In this case it was in the specification.

Occasionally false alarms occurred: an unprovable goal was obtained, suggesting an error. However, on closer inspection it was found that the problem was that information had been lost in the course of the proof. For example, if $t_1 < t_2$ is turned into $t_1 \leq t_2$ during the proof, the information that the two times are not equal is lost. Such a false alarm could lead to an unnecessary change in the implementation being made.

Many trivial typing errors were caught at an early stage by type-checking. However, many other trivial mistakes were made over the size of words and signals. For example, words of size 4 by 2 were inadvertently specified as 2 by 4 words. These errors were found during the proof process. It would have been much better if they had been picked up earlier. This would have been possible if dependent typing had been available.

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 do take longer to verify, partly because there are more cases to consider. This is made worse if the interfaces between modules are left containing lots of low level detail. For example, in the proof of 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. 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. However, it should be noted that the very top module which simply added various delays to various inputs and outputs of the main module, only took a day to verify.

4 The MDG Verification

In the second study, the same circuit was verified using a decision graph approach. A new technique called abstract implicit enumeration has been developed where decision graphs are used to represent sets of states as well as the transition and output relations [3]. Based on this technique hardware verification tools have been developed which perform combinational circuit verification, safety property checking and equivalence checking of two state machines.

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 symbol (cross-operator). 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.

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. Using the MDG tools, we verified the gate-level implementation against the abstract (RTL) hardware model. 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].

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 [9].

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. 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 description.

When an invariant is not satisfied during the verification process, a counterexample is provided to help with identifying the source of the error. Like ROBDDs, the MDGs require a fixed node ordering. Currently, the node ordering has to be given by the user explicitly. Unlike ROBDDs where all variables are Boolean, every variable used in the MDGs needs to be assigned an appropriate sort and type definitions must be provided for all functions. Rewrite rules may need to be provided to partially interpret the otherwise uninterpreted function symbols.

The Structural Specification 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 and registers can be modeled at the Boolean or the abstract level using abstract terms as inputs and outputs. A translator from a subset of VHDL into MDG-HDL is under development.

As an example, the following is an MDG-HDL description of the `DMUX4T2` module given in Section 3:

$$
\begin{array}{l}
\textbf{module}(\texttt{DMUX4T2} \\
\quad \textbf{port}(\textbf{inputs}((d0, bool), (d1, bool), (d2, bool), (d3, bool)), (x, bool)), \\
\qquad \textbf{outputs}((dOut0, bool), (dOut1, bool))), \\
\quad \textbf{structure}(\\
\qquad \textbf{signals}(xBar, bool), \\
\qquad \textbf{component}(\texttt{InvX}, \textbf{NOT}(\textbf{input}(x), \textbf{output}(xBar))), \\
\qquad \textbf{component}(\texttt{AO\_0}, \textbf{AO}(\textbf{input}(d0, xBar, d1, x), \textbf{output}(dOut0))), \\
\qquad \textbf{component}(\texttt{AO\_1}, \textbf{AO}(\textbf{input}(d2, xBar, d3, x), \textbf{output}(dOut1))))).
\end{array}
$$

Here, the components **NOT** and **AO** are basic components provided by the MDG-HDL library. Note also that the data sorts of the interface and internal signals must always be specified.

Besides the gate-level description, we also provided a more abstract (RTL) description of the implementation which holds for arbitrary word width. Here, the data-in and data-out lines are modeled using an abstract sort *wordn*. The active, 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 example, a set of four `DMUX4T2` modules is modeled using a single multiplexor component. For more details about the abstraction techniques used refer to [13].

The Behavioral Specification 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 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

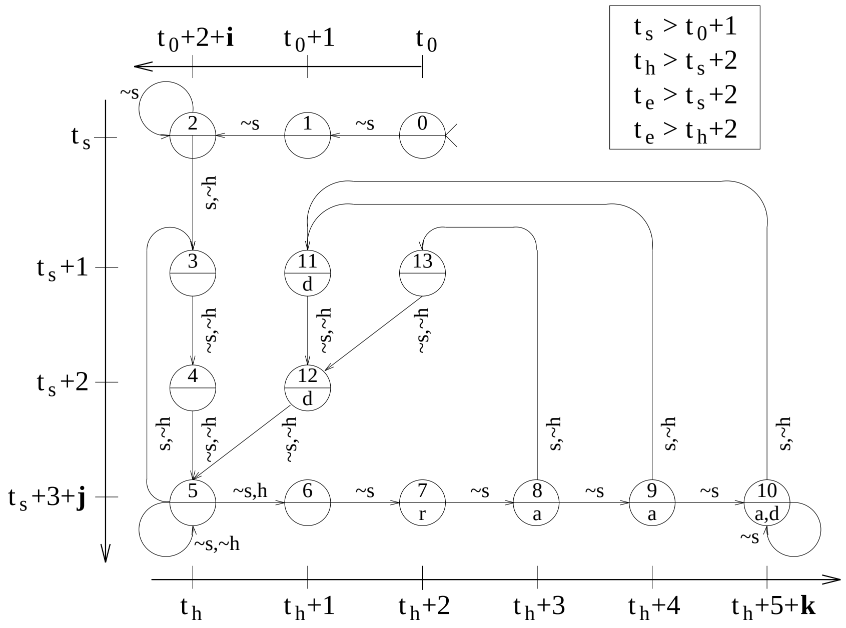

Fig. 2. ASM Behavioral Specification

the environment maintains certain timing constraints on the arrival of the frame start signal and headers. A schematic representation of the ASM specification of the 4 by 4 switch fabric is shown in Figure 2. The symbols t_0, t_s, t_h and t_e in the figure represent the initial time, the time of arrival of the frame start signal, the time of arrival of the routing bytes and the time of the end of a frame, respectively. There are 14 conceptual states: States 0, 1 and 2 along the time axis t_0 describe the initial behavior of the switch fabric. States 2, 3, 4 and 5 along the time axis t_s describe the behavior of the switch on the arrival of a frame start signal. States 6 to 13 along the time axis t_h describe the behavior of the switch fabric after the arrival of the headers. The waiting loops in states 2, 5 and 10 are illustrated in the figure by the non-zero natural numbers i, j and k, respectively. Figure 2 also includes many meta symbols used to keep the presentation simple. For instance, the symbols s and h denote a frame start and the arrival of a routing tag (header), respectively, and the symbol "$\sim$" denotes negation. The symbols a, d and r inside a conceptual state represent the computation of the acknowledgment output, the data output and the round-robin arbitration, respectively. The absence of an acknowledgment or a data symbol means that no computation takes place and the default value is output.

To formally describe this ASM using MDGs, we first introduced some basic sorts, constants and functions (cross-operators), e.g. a concrete sort $port = \{0, .., 3\}$, an abstract sort $wordn$, a constant $zero$ of sort $wordn$ and a cross-operator rou of type $[wordn \rightarrow port]$ representing the route field in a header.

Further, 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-constructs. For example, the acknowledgment output is described by four formulas determining the value of $ackOut_i$, $i \in \{0, .., 3\}$:

$$\begin{array}{l} \textbf{if } ((co_0 = 1) \textbf{ and } (ip_0 = i)) \textbf{ then } (ackOut_i = ackIn_0) \\ \textbf{ef } ((co_1 = 1) \textbf{ and } (ip_1 = i)) \textbf{ then } (ackOut_i = ackIn_1) \\ \textbf{ef } ((co_2 = 1) \textbf{ and } (ip_2 = i)) \textbf{ then } (ackOut_i = ackIn_2) \\ \textbf{ef } ((co_3 = 1) \textbf{ and } (ip_3 = i)) \textbf{ then } (ackOut_i = ackIn_3) \\ \qquad\qquad\qquad\qquad\qquad\qquad \textbf{else } (ackOut_i = 0) \end{array}$$

where co_i $(i \in \{0, .., 3\})$ of sort *bool* and ip_i $(i \in \{0, .., 3\})$ of sort *port* are state variables generated by the round-robin computation and corresponding to the output disable and grant signals, respectively (Figure 1).

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 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 simulated 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 t_0, t_s, t_h and t_e to specific states. We then described the properties as invariants which should hold in all reachable states of the specification model. The following is an example of a property which checks for correct routing to port 0. It is expressed in MDG-HDL using an ITE construct.

$$\begin{array}{l} \textbf{if } (s \in \{17, .., 68\}) \textbf{ and } priority[0..3] = [1, 0, 0, 0] \textbf{ and } route[0] = 0 \\ \textbf{then } dataOut[0] = dataIn'[0] \end{array}$$

Here $priority[0..3]$ indicates the priority bits for all input ports, $route[0]$ represents the routing bits for input port 0 and $dataIn'[0]$ is the data input on port 0 delayed by 4 clock cycles. Further examples of properties are described in [13].

Time Taken The user time required for the specification and verification is hard to determine since it included the improvement of the MDG package, writing documentation, etc. 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

Verification	CPU Time (s)	Memory (MB)	MDG Nodes Generated
Gate-Level to RTL	183	22	183300
RTL to Beh. Model	2920	150	320556
P1: Data Output Reset	202	15	30295
P2: Ack. Output Reset	183	15	30356
P3: Data Routing	143	14	27995
P4: Ack. Output	201	15	33001
Error (i)	20	1	2462
Error (ii)	1300	120	150904
Error (iii)	1000	105	147339

Table 1. Experimental Results for the MDG Verification

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 injection and verification of an introduced design error was less than one hour. The experimental results in machine time are shown in Table 1 including CPU time (on a SPARC station 10), memory usage and number of MDG nodes generated.

A disadvantage of MDGs is that much verification time is spent finding an optimal variable ordering. This is crucial since a bad ordering easily leads to a state space explosion. This occurred after an early ordering attempt. For more information about the variable ordering problem, which is common to all ROBDD-based systems, see [1].

Because the verification is essentially automatic, the amount of work rerunning 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 determining a suitable variable ordering. Depending on the kind of design changes adopted, it is not obvious if the original variable ordering could still be used on a modified design without major changes.

The MDG gate-level specification is a concrete description of the fabricated 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 implementation 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 in fact be used under the behest of its operating environment, i.e. the port controllers. However, while this reduces the verification cost, it has the disadvantage that the verification must be completely redone if the operating environment changes. Still, the work required is minor as only a few parameters have to be changed in the description of the environment state machine (which is a simple machine [13]).

Errors As with the HOL study, no errors were discovered in the implementation. For experimental purposes, however, we injected several errors into the implementation and checked them using either the set of properties or the behavioral model. Errors were automatically detected and identified using the counterexample facility. The injected errors included the main errors introduced in the HOL study, discussed in Section 3. We summarize here three further examples. (i) We exchanged the inputs to the JK Flip-Flop that produces the output disable signal. This prevented the circuit from resetting. (ii) We used, at one point, the priority information of input port 0 instead of input port 2. (iii) We used an AND gate instead of an OR gate within the acknowledgment unit, thus producing a faulty $ackOut[0]$ signal. Experimental results for these three errors, which have been checked by verifying the RTL model against the behavioral specification, are reported in Table 1.

While checking properties on the hardware structural description, we also discovered some errors that we mistakenly introduced in the structural specifications. However, we were able to easily identify and correct these errors using the counterexample 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, since they included the same collection of gates. Finally, many trivial typing errors were highlighted at an early stage of the description process by the error messages output after each compilation of the specification's components.

Scalability Like any FSM-based verification system, the MDG proof approach is not directly scalable to large designs. This is due to the possible state space explosion that results from large designs. Unlike other ROBDD-based approaches, however, MDGs do not need to cope with the datapath complexity since 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 is practically impossible. We overcame this problem by providing an abstract RTL structural specification which we instantiated for the verification of the gate-level model. In order to handle large designs, major efforts are in general required to set up the appropriate model abstraction levels.

5 Conclusions

The MDG and HOL structural descriptions are very similar, both to each other and to the original designer's description. 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 only minimal additional effort, however.

The behavioral descriptions of the two approaches are totally different. The MDG specification is based on a state machine model while the HOL one is based on interval temporal logic operators, explicitly describing the timing behavior using a set of formulas that include different scenarios of the switch fabric behavior, e.g. active or inactive frames. Both describe the behavior in a clear and comprehensive form. Which of these is preferred is perhaps a matter of taste.

An advantage of MDG is that a property specification is easy to set up and verify. Expected operating conditions can be used to simplify this, even if the full specification is more general. This is useful for verifying that a specification satisfies its requirements. It can greatly reduce the full verification cost by catching errors at an early stage.

Writing the behavioral specifications was far slower in HOL, as separate specifications were needed for each module. In MDG this was not necessary because the whole design was verified in one go, rather than a module at a time. This also reduced the MDG verification time because fewer mistakes were made.

Both approaches successfully highlight errors, and help determine their location. However, the way this information manifests itself differs. MDG is more straightforward, outputting a trace of the input sequence that leads to the erroneous behavior. 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.

The HOL verification was much slower, taking a matter of months. This time includes the verification of each of the modules and the verification of their combination. Using HOL, a large number of lemmas had to be proved and much effort was required to interactively create the proof scripts. For example, the time spent for the verification of the dataswitch unit was about 3 days. Here the proof script was about 530 lines long (17 KB). The MDG verification was achieved automatically without the need of a proof script. All that was required was the careful management of the MDG node ordering (as with ROBDDs). However, this is a matter of hours or at most a few days of work.

In both the HOL and MDG approaches, the amount of work necessary to verify a modified design, once the original has been verified, is greatly reduced. Both allow generic verification to be performed, though HOL has the potential to be more flexible. Because MDG is automated and fast, the re-verification times would largely be just the time taken to modify the specifications and to find a new variables ordering. In the HOL approach, the behavioral specifications of many modules and the proof scripts themselves may need to be modified.

An advantage of the HOL approach in contrast to the MDG method is the confidence in the tool the LCF approach offers. Although the MDG software package has been successfully tested on several benchmarks and has been considerably improved, it is not yet a mature tool. It cannot guarantee the same level of proof security as HOL. The main advantage of the MDG approach is that it is much quicker and is automatic. On the other hand the theorem proving approach is potentially scalable and involves a comprehensive investigation of why the design works correctly. However, these advantages are only likely to be realized in practice if the level of proofs which must be provided to the system can be raised closer to the level of informal proofs.

Acknowledgements We are grateful to Zijian Zhou, Xiaoyu Song and Eduard Cerny at the University of Montreal, Canada and Michel Langevin at GMD-SET, Germany for initiating and advocating this study. Ian Leslie and Mike Gordon at Cambridge University were also of great help. This work was partially funded by EPSRC research agreements GR/J11133 and GR/K10294.

References

1. R. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.
2. B. Chen, M. Yamazaki and M. Fujita. Bug Identification of a Real Chip Design by Symbolic Model Checking. In *Proc. of the Int. Conf. on Circuits And Systems*, pages 132–136, June 1994.
3. F. Corella, Z. Zhou, X. Song, M. Langevin and E. Cerny. Multiway Decision Graphs for Automated Hardware Verification. *Formal Methods in System Design*, To appear. Available as IBM research report RC19676(87224), July 1994.
4. P. Curzon. Tracking Design Changes with Formal Machine-checked Proof. *The Computer Journal*, 38(2):91–100, July 1995.
5. P. Curzon and I.M. Leslie. A Case Study on Design for Provability. In *Proc. of the Int. Conf. on Engineering of Complex Computer Systems*, pages 59–62, IEEE Computer Society Press, November 1995.
6. M.J.C. Gordon. HOL: A Proof Generating System for Higher-order Logic. In G. Birtwistle and P.A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, pages 73–128. Kluwer Academic Publishers, 1988.
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. J.M.J. Herbert. Case Study of the Cambridge Fast Ring ECL Chip using HOL. Technical Report 123, University of Cambridge, Computer Laboratory, February 1988.
9. M. Langevin, S. Tahar, Z. Zhou, X. Song and E. Cerny. Behavioral Verification of an ATM Switch Fabric using Implicit Abstract State Enumeration. In *Proc. of the Int. Conf. on Computer Design*, IEEE Computer Society Press, October 1996.
10. I.M. Leslie and D.R. McAuley. Fairisle: An ATM Network for the Local Area. *ACM Communication Review*, 19(4):327–336, September 1991.
11. K.L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.
12. K. Schneider and T. Kropf. Verifying Hardware Correctness by Combining Theorem Proving and Model Checking. In J. Alves-Foss, editor, *International Workshop on Higher Order Logic Theorem Proving and Its Applications: B-Track: Short Presentations*, pages 89–104, August 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. of the Great Lakes Symp. on VLSI*, pages 106–111, IEEE Computer Society Press, March 1996.