Formal Verification of an ATM Switch Fabric using Multiway Decision Graphs

Sofiène Tahar, Zijian Zhou, Xiaoyu Song, Eduard Cerny and Michel Langevin[§]
University of Montreal, IRO Dept., Canada. E-mail: {tahar,zhouz,song,cerny}@iro.umontreal.ca
[§] GMD-SET, Germany. E-mail: langevin@borneo.gmd.de

Abstract

In this paper we present our results on formally verifying the implementation of an Asynchronous Transfer Mode (ATM) network switching fabric using a new class of decision graphs, called Multiway Decision Graphs (MDG). The design we consider is in use for real applications in the Cambridge Fairisle network. We produced the description of the hardware implementation at different levels of abstraction. We then performed the verification of an abstract description model against the description of the gate-level implementation. Using this abstract model, we accomplished the verification of specific properties that reflect the behavior of the Fairisle ATM switch fabric.

1. Introduction

As communication networks are becoming pervasive, the consequence of errors in the design or implementation of network components becomes increasingly important. Simulation and testing have traditionally been used for checking the correctness of those systems. However, it is practically impossible to run an exhaustive test or simulation for such large and complex systems. The use of formal verification for determining the correctness of digital systems is thus gaining interest, as the correctness of a formally verified design implicitly involves all cases regardless the input values.

ATM (Asynchronous Transfer Mode) has been conceived as an appropriate network technology to address the variety of needs for new high-speed, high-bandwidth applications. It is being hailed as the most important communication mechanism of the foreseeable future. However, there is currently little experience on the application of formal verification to ATM network hardware.

In this paper, we present our results on formally verifying an ATM network component using a new class of decision graphs, called Multiway Decision Graphs (MDG) [3]. These decision graphs subsume the class of Bryant's Reduced Ordered Binary Decision Diagrams (ROBDD) [1] while accommodating abstract sorts and uninterpreted function symbols. The device we investigated is part of a network which carries real user data: the Fairisle ATM network [8], designed and in use at the Computer Laboratory of the University of Cambridge. It provides a realistic formal verification case study. The component we consider is the Fairisle 4 by 4 switching fabric. It performs the actual switching of data cells from input ports to output ports and arbitrates cell clashes and forms the heart of the ATM Fairisle communications network switches.

We produced the description of the switch fabric at two different levels of abstraction. We then performed the verification of an abstract description model against the original gate-level implementation. Using the former model, we then verified specific safety properties which reflect the behavior of the Fairisle ATM switch. In addition, we checked the correctness of several faulty implementations in which the introduced errors were successfully identified by generating adequate counterexamples. Using the applications provided by the MDG software package, all verification tasks were achieved automatically in a reasonable amount of time.

The organization of this paper is as follows: Section 2 outlines some related work in the formal verification of ATM hardware. In Section 3, we give a brief description of Multiway Decision Graphs (MDGs) and the existing MDG-related verification techniques. In Section 4, we overview the Fairisle ATM switch. The descriptions of the hardware implementation at the gate and abstract (word) levels are sketched out in Section 5. In Section 6, we explore the different verifications we accomplished using the models. Experimental results are presented in Section 7 and Section 8 finally concludes the paper.

2. Related works

There exists in the literature only few work which addressed the formal verification of ATM related circuits.

P. Curzon [4] formally verified the 4 by 4 fabric of the Fairisle switch using the HOL theorem prover [6]. He verified each of the modules used in the design of the switching element separately by describing the behavioral and structural specifications down to the gate level, and then proving the related correctness theorems in HOL. The separate proofs were then combined to give a result about the verification of the whole switch fabric.

Another approach of formal verification of an ATM circuit was made by B. Chen et al. at Fujitsu Digital Technology Ltd. [2]. The authors identified a design error in an ATM circuit using the tool SMV (Symbolic Model Verifier) [7] by verifying some properties expressed in CTL (Computational Tree Logic) [7]. To avoid the state explosion problem, the authors abstracted the datapath from 8 bits to 1 bit. However, since the datapath was only 1 bit in the state model, in certain blocks the property could not be checked because of this reduction. In these cases, a more detailed datapath model was built to pinpoint the source of the error. If the error was not identified in that block, then the more abstract model is used for the remaining verification.

The approach of P. Curzon [4] provided a successful case study of applying HOL theorem prover to the verification of an ATM switch. However, the use of HOL is interactive and requires lots of expertise to guide the verification process. The work done at Fujitsu Ltd. showed the application of SMV for checking some important properties related to the circuit implementation. Although the verification is automatic, the adopted data abstraction (e.g., using 1 bit to represent 8-bit data width) for avoiding the state explosion problem was not quite adequate.

3. Multiway decision graphs

Although ROBDDs have proved to be a powerful tool for automated hardware verification, they require a binary representation of the circuit. Hence, the size of an ROBDD grows, sometimes exponentially, with the number of variables. ROBDD-based verification thus cannot be applied to circuits with complex datapaths. Recently, *Multiway Decision Graphs* (MDG) have been proposed to represent circuits at a more abstract level [3]. It is based on a subset of a many-sorted first-order logic augmented with a distinction between *abstract sorts* 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. For circuits with large datapaths, MDGs are thus much more compact than ROBDDs, and hence greatly increase the range of circuits that can be verified since the verification is independent of the width of the datapath. For details about MDGs refer to [3].

Using abstract sorts, we are able to represent circuits at the RT-level (Register Transfer). We have developed a reachability analysis algorithm (based on a technique called *abstract implicit enumeration* [3]) where we use MDGs to represent sets of abstract states as well as the transition and output relations of sequential RTL designs. We also have developed applications for hardware verification such as combinational circuits verification, safety property checking and equivalence checking of two state machines. In safety property checking, we verify that a certain condition holds in all reachable states of a state machine. In equivalence checking, we form a product machine of the specification and the implementation and check that the equivalence of corresponding outputs is an invariant. When an invariant is not satisfied during the verification process, a counterexample is provided to help with identifying the source of the error.

4. The Fairisle ATM switch

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* (Figure 1). Each port controller is connected to a transmission line and to the switch fabric. The port controllers synchronize and process incoming and outgoing data cells, appending control information in the front of the cells in a *routing tag* (Figure 2). This tag 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

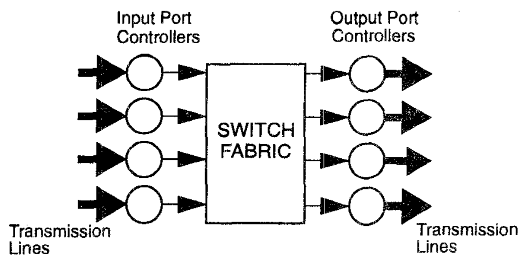

Figure 1. The Fairisle ATM switch

switches cells from the input ports to the output ports according to the routing tag. If different port controllers inject cells destined for the same output port controller (as indicated by the *route* bits) into the fabric at the same time, then only one will succeed. The others must retry later. The routing tag also includes priority information (*priority* bit) that is used by the fabric for arbitration which 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 cells were 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 use all the same clock, hence bytes are received synchronously on all links. They also use 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 tags (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.

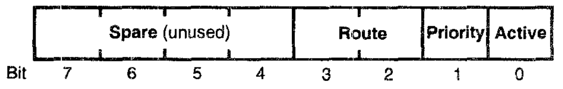

Figure 2. The routing tag of a Fairisle ATM cell

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 a particular bit (the *active* bit) of any one of them 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 3 shows a block diagram of a 4 by 4 switch fabric. The inputs of the fabric consists of the data lines which carry the cells, the acknowledgments that pass in the reverse direction, and the frame start signal which is the only external control signal. The outputs consist of the switched data and the switched and modified acknowledgment signals. The switch fabric is composed of an arbitration unit, an acknowledgment unit and a dataswitch unit (Figure 3). The arbitration unit reads the routing tags, makes arbitration decisions when two or more cells are destined for the same output port, passes the result to the other modules with the grant signals and governs the timing of the other units using the output disable signals. The dataswitch performs the actual switching of data from an input port to an output port according to the most recent arbitration decision. The acknowledgment unit passes appropriate acknowledgment signals to the input ports. Negative acknowledgments are sent until arbitration is completed.

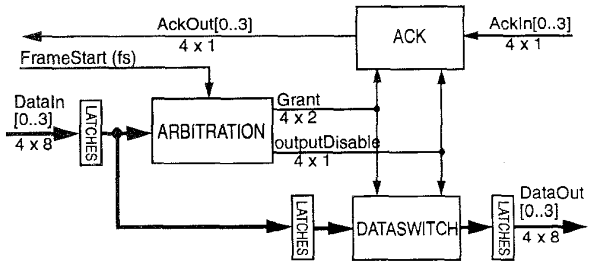

Figure 3. The Fairisle switch fabric

All the units are repeatedly subdivided until eventually the logic gate level is reached, providing a hierarchy of units. The design has a total of 441 basic components where a basic component is a logic gate with two or multiple inputs or a one-bit flip flop. The switch fabric is built on a 4200 gate equivalent Xilinx programmable gate array.

5. Description of the switch fabric hardware

The Fairisle switch fabric was designed using the Qudos HDL [5] hardware description language. To formally verify the fabric using MDGs, we translated these descriptions into very similar descriptions using a prolog-style HDL—MDG-HDL which is supported by the MDG software package.

5.1. Gate-level description

Many of the modules in the original Qudos description were large and logically unrelated while preserving the mapping into a Xilinx gate array. Similarly to the description done by Curzon in HOL [4], we organized the hardware description in several levels of hierarchy, making use of modularity within MDG-HDL that is lacking in Qudos HDL, thus facilitating both the specification and the verification.

The Fairisle switch fabric is composed of the acknowledgment, arbitration and dataswitch units (Figure 3). Each unit is further defined as a module which is further subdivided until the same gate-level implementation is reached as in the original Qudos HDL design. Both the Qudos HDL and the MDG-HDL descriptions are in terms of the same collection of logic gates. To illustrate the similarities and differences of the two descriptions, we consider the Qudos HDL and MDG-HDL structural descriptions of a mutiplexing component of the dataswitch—DMUX4T2. We first give the Qudos HDL definition as:

```
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;
```

where the first statement is a dummy declaration providing information about the way the component design should be mapped into a Xilinx gate array, and the AO components are AND-OR logic gates. Using MDG-HDL this same module is described as:

```
module(DMUX4T2
   port(inputs((d0,bool),(d1,bool),
               (d2,bool),(d3,bool)),(x,bool)),
       outputs((dout0,bool),(dout1,bool))),
   structure(
      signals(xBar,bool),
      component(InvX,NOT(input(x),output(xBar))),
      component(AO_0,AO(input(d0,xBar,d1,x),
                        output(dout0))),
      component(AO_1,AO(input(d2,xBar,d3,x),
                        output(dout1))))).
```

Here, the components NOT and AO are atomic modules provided by the MDG software package. Note also that the data sorts in the interface and the internal signals must always be specified.

5.2. Abstract (word-level) description

The data inputs and outputs of the switch fabric consist of 4 byte-wide lines. In Qudos HDL there is no facility for describing high-level words. Thus the data-in and data-out lines are modelled as 32 individual lines. This could be easily modelled in MDG-HDL using concrete sorts, say an enumeration sort word8 for words of size 8. However, they are better described as words of size *n* using abstract sorts, e.g., an abstract sort wordn. Such high-level words are of arbitrary size, thus making descriptions generic where the word sizes do not always have to be specified. The verification of the switch element is therefore directly applicable to switch elements of different word size.

Beside abstracting the data lines from a bundle of bits to a compact word of abstract sort, we have abstracted the behavior of the dataswitch unit by modelling it using simple data multiplexors instead of collections of logic gates. For example, a set of 8 DMUX4T2 modules (see Section 5.1) is modelled using a single mutiplexor component (which is part of a larger module) as follows:

```
signal(dw0,wordn).
signal(dw1,wordn).
signal(dwout,wordn).
signal(x,bool).
component(DMUX4T2_w,MUX(sel(x),
                        inputs([(0,dw0),(1,dw1)]),
                        output(dwout))).
```

We now obtain a much simpler abstract implementation of the dataswitch which reflects the switching behavior in a more natural way and is implemented with a smaller network of components and signals. Figure 4 shows the abstraction of the switch fabric used, where *w* is a word of abstract sort wordn. Note that the arbitration block is now provided with data inputs of sort wordn. It is no more fed with single bits of the routing tag and its description involves uninterpreted functions which extract (decode) specific bits from the input tags (words), e.g., the active bit is obtained through the use of the uninterpreted function bit0 of type [wordn $\rightarrow$ bool] which selects the first bit of the routing tag. This ability to use uninterpreted functions for bit manipulation is used in the verification of the abstract description against the gate-level one in Section 6.1.

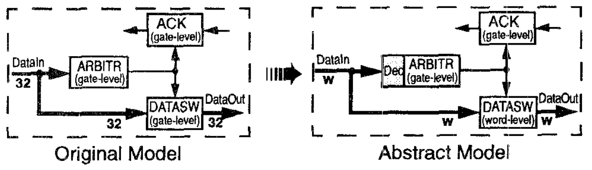

Figure 4. Model abstraction for the switch fabric

6. Formal verification

6.1. Verification of the abstract model

If we wish to use the abstract (word-level) model of the fabric for further experimentation, we should ensure the equivalence of the gate-level implementation against the abstract model. The correctness of equivalent behavior is established if the two machines produce the same data outputs for all input sequences. This, however, cannot be done for an arbitrary word size *n* since the gate-level description is not generic. We should somehow "instantiate" the data signals of the abstract model to be 8 bit wide. This can be realized within the MDG environment using *uninterpreted functions* which encode and decode abstract data to Boolean data and vice-versa. For instance, *decoding* can be realized using 8 uninterpreted functions bit^i (*i*: 0..7) of type [wordn $\rightarrow$ bool], which extract the *i*th bit of an *n*-bit data and hence encode the 4 *n*-bit data lines to a 32-bit bundle. *Encoding* is done using one uninterpreted function concat8 of type [(bool x bool x bool x bool x bool x bool x bool x bool) $\rightarrow$ wordn] which concatenates any 8 Boolean signals to a single word and thus encodes a bundle of 32 Boolean data signals to 4 signals of sort wordn.

Based on this technique, four symmetric configurations are possible for performing the verification. This is illustrated in Figure 5 where only the data inputs and outputs of the fabric are considered, since the abstraction in fact only affects the dataswitch block (Figure 4). In all of these cases, we ensure that we feed the two machines with the same inputs and check the equivalence of their outputs which should also be the same. The encoding and decoding blocks (as represented in Figure 5) are not functional blocks that

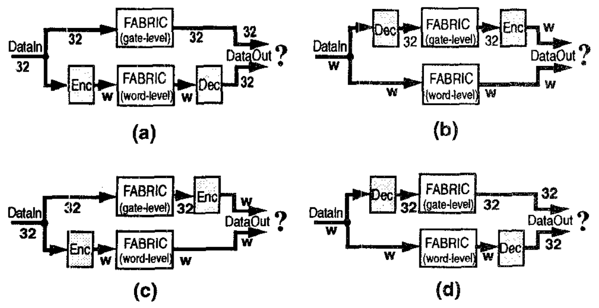

Figure 5. Different configurations for the abstract model verification

we add to the implementation description but illustrate the uninterpreted functions that we insert into the invariant during equivalence verification. Theoretically any one of these configurations could be used in the verification. However, we noticed that case (d) in Figure 5 is the least expensive. This is because it avoids the use of the uninterpreted function concat8 which requires a full encoding of the 8-bit Boolean vector at the abstract level, i.e., for each Boolean vector which could be represented by the 8 bits, we have to provide a fixed correspondence to a specific constant of abstract sort, e.g., (0,0,0,0,0,0,0,0) $\leftrightarrow$ zero, (0,0,0,0,0,0,1,0) $\leftrightarrow$ two, where zero and two are of abstract sort wordn.

Using the sequential equivalence checking facility of the MDG tools, we verified that the abstract machine is equivalent to the original gate-level one for a word size equal to 8. Since the data abstraction affects only the dataswitch block (refer to Figure 4), the verification needed to deal with the equivalence of the dataswitch blocks at the two levels. The verification run time for the (best) variant (d) in Figure 5 is given in Section 7.

6.2. Verification of safety properties

Property checking is used for verifying that a design satisfies some specific requirements. The ATM switch fabric works under the control of its environment, i.e., the port controllers. It transfers the data at exact clock cycles. Thus checking safety properties is sufficient to verify its correct behavior. For designs containing symmetrical portions, like the ATM switch in question, we can even trace the error to the parts of the circuit. In this section, we describe our techniques for the safety property verification using the abstract model of the switch fabric. Examples of properties are checking of correct circuit reset and checking of correct data routing. In difference to the previous verification of the switch fabric in HOL [4], we perform the verification under the conditions of its operating environment, which is sound and can greatly reduce the cost of verification.

We consider the behavior of the fabric in the real Fairisle switch. The environment of the switch fabric generates the frame start signal (Figure 3), denoted as *fs*, at every 64th rising clock cycle. It goes low in the next clock cycle. Initially, it should wait at least 2 clock cycles to let the fabric reset before it can generate the first *fs* signal. The first byte of the cell (the routing tag, or the header), denoted as *h*, must be generated at the 8th rising clock edge after the *fs is* reset to low, i.e., 9 clock cycles after *fs* is set. When the active bit in this routing tag is set, the cell is called active. Otherwise it is inactive (an empty cell). To simplify our presentation, we consider the case for active cells only.

In analogy to the specification of P. Curzon [4], we use the time points t_s, t_a and t_e to denote the start of a frame, the start of an active cell and the end of a frame (which is the start of the next frame), respectively. Using these time points, we can state several properties which reflect the behavior of the switch fabric. These properties are indeed inspired by the top-level behavioral specification of the switch fabric as given in [4] and the other documentation about the switch element design. In the rest of this section, we illustrate our verification technique by the following representative properties:

- *Property 1*: From t_s+3 to t_a+4, the default value (zero) will be put on the data output port DataOut[0] where zero is a generic constant.
- *Property 2*: From t_s+1 to t_a+2, the default value (0) will be put on the acknowledgment output port AckOut[0].
- *Property 3*: From t_a+5 to t_e+2, if input port 0 chooses output port 0 with the priority bit set in the routing tag, and no other input ports have their priority bits set, the value on DataOut[0] will be the input of DataIn[0] of 4 clock cycles earlier.
- *Property 4*: From t_a+3 to t_e, if input port 0 chooses output port 0 with the priority bit set in the routing tag, and no other input ports have their priority bits set, the value on AckOut[0] will be the input of AckIn[0].

Property 1 and *2* deal with the reset behavior of the circuit, while *Property 3* and *4* state specific behaviors of the switching of cells. Although the (informal) description of the above properties explicitly involves the notion of time, we can verify them using only safety property checking based on a state machine model inspired by [9].

First, we simulate the environment as a state machine with one state variable *s* as shown in Figure 6 where we assume that the first frame start signal is generated after two clock cycles after power on.

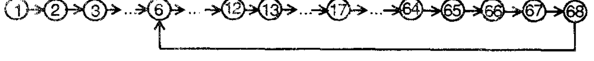

Figure 6. Environment state machine

In Figure 6, there are 68 states enumerated by integers (Using MDG-HDL, *s* can be described as a concrete variable of sort [1..68]). Arrows denote state transitions. *fs*, *h* and *d* above the states mean that the frame start signal, the routing tag of an active cell and the data are generated in that state, respectively. States 1 to 5 are related to the initialization of the fabric. States 6 to 68 represent the cyclic behavior of the fabric, where one cycle corresponds to one frame. With this diagram, we can map the time points t_s, t_a and t_e to states, as $t_s = 3$ or $t_s = 66$; $t_a = 12$; and $t_e = 66$. Then, e.g., t_a+5 to t_e+2 are essentially the states between 17 and 68 when the remaining 52 bytes of the cell following the routing tag are switched to the output port. Consequently, we can express our properties in terms of states rather than time points.

The environment state machine is composed with the switch fabric, as shown in Figure 7. As there is a 4-clock-cycle delay for the cells to reach the output ports, it is necessary to use a delay circuit to remember the input values that are to be compared with the outputs (remember that the fabric is modelled at an abstract (word) level, i.e., DataIn[0..3] and DataOut[0..3] are all *n*-bit words). We also need to memorize the grant signal (which grants an output port to an input port) of the previous frame for the round-robin arbitration (Section 4). Combining these machines, we obtain the required platform for checking the safety properties. This composed machine is represented inside the dashed frame in Figure 7.

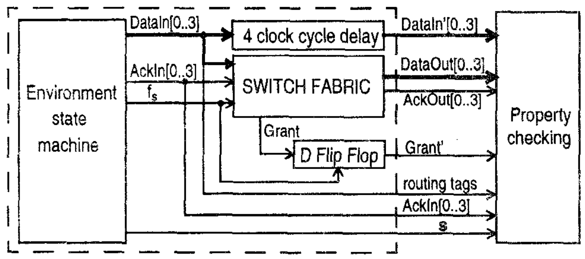

Figure 7. Checking safety properties

We re-state now the previous properties as invariants using ITE (If-Then-Else) formulas.

- *Property 1'*: **If** (s $\in N$) **then** DataOut[0] = zero, where N = [6,...,16] and zero is the default data value (a generic constant of sort wordn).
- *Property 2'*: **If** (s $\in N$) **then** AckOut[0] = 0, where N = [4,...,14, 67, 68] and 0 is the default acknowledgment value.
- *Property 3'*: **If** (s $\in N$) **and** priority[0..3] = [1,0,0,0] **and** route[0] = 0 **then** DataOut[0] = DataIn'[0], where N = [17,...,68], priority[0..3] are the priority bits for all the input ports and route[0] are the routing bits for input port 0 (refer to Figure 2).
- *Property 4'*: **If** (s $\in N$) **and** priority[0..3] = [1,0,0,0] **and** route[0] = 0 **then** AckOut[0] = AckIn[0], where N = [15,...,66], priority[0..3] and route[0] are the same as explained in the above property.

These invariants can be easily represented using MDGs. By exploring the state space of the combined machine, at each reachable state, we check if the outputs satisfy the invariant. We have verified the above properties for one output port of the fabric. In addition, we were able to discover several introduced design errors by checking those properties.

7. Experimental results

We verified the abstract model against the gate-level one, and the properties described in Section 6.2 which reflect the essential behavior of the switch. Using these properties, we also checked the correctness of three erroneous implementations. The experimental results are shown in Table 1. They were done on a SPARC station 20 with 128 MB of memory. The CPU time is in seconds and the memory usage is given in megabytes.

The verification of the abstract model against the gate-level implementation was reduced to the comparison of the dataswitch outputs of both machines. Therefore, we were able to verify their equivalence after 2 transitions since we have a 2-clock-cycle delay within the dataswitch. Note that the state variables of the two machines had to be initialized with corresponding values, i.e., constants zero of sort wordn, and 0's of sort Boolean for the abstract and gate-level models, respectively. In this case, we used several rewriting rules stating that all bits of zero are equal to the Boolean 0, i.e., $\text{bit}^i(\text{zero}) = 0$, i=0..7.

We checked Properties 1' to 4' for one output port of the switch fabric. The composed machine (Figure 7) for one port has 157 components, 161 signals and 53 state variables (27 abstract variables and 26 concrete variables which are equivalent to 32 Boolean variables).

We also introduced several errors into the implementation. We describe here three examples. First, we exchanged the inputs to the JK Flip-Flop that produces the output disable signal. As the output disable signal is wrong, the circuit cannot be correctly reset. This error was identified by property 1' after 6 transitions and by property 2' after 4 transitions. Both experiments generated counterexamples which indicated the wrong behavior of the output disable signal. Second, we used the priority signal of input port 0 as the priority signal of input port 2. This error was discovered by property 3' after 17 transitions. Third, we used an AND gate instead of an OR gate which produces the AckOut[0] signal. This error was found by property 4' after 15 transitions.

Table 1. Experimental results

Verifications	CPU time (in sec)	Memory (in MB)	Transitions	Nodes generated
Abstract model	183	22	2	183300
Property 1'	202	15	68	30295
Property 2'	183	15	68	30356
Property 3'	143	14	68	27995
Property 4'	201	15	68	33001
Error 1 using property 1'	49	8	6	16119
Error 2 using property 3'	77	11	17	24001
Error 3 using property 4'	82	11	15	24274

8. Conclusions

In this paper, we have shown the feasibility of formal verification techniques based on a new class of decision graphs—Multiway Decision Graphs (MDG), when applied to a real circuit. In this sense, we have demonstrated that formal verification of a real piece of communication hardware can be conducted automatically using the MDG tools. This ATM is much larger than any other circuit we verified before using MDGs. Descriptions and verifications of the fabric have been done at different levels of abstraction. We verified the equivalence of the original gate-level implementation of the switch fabric against an abstract description model in which the generic words of abstract sort were aligned with 8-bit words. Based on the abstract model, we then verified some specific safety properties that reflect the behavior of the fabric when used in the Fairisle ATM switching network. We also accomplished the verification of several faulty implementations where the introduced errors were successfully identified. These different achievements illustrate the practicability of such a complete formal verification down to the gate level using tools based on Multiway Decision Graphs.

In [4] it is reported that the time spent testing would have been in the order of several weeks. However, errors were discovered after the testing process was completed when the first version of the fabric was in use. Had formal verification been applied to the ancestors of the actual design, the formal verification could possibly have discovered the errors and then validated the corrections.

References

[1] Bryant, R.: Graph-Based Algorithms for Boolean Function Manipulation; *IEEE Transactions on Computers*, Vol. C-35, No. 8, August 1986, pp. 677-691.

[2] Chen, B.; Yamazaki, M.; Fujita, M.: Bug Identification of a Real Chip Design by Symbolic Model Checking; *Proc. International Conference on Circuits And Systems* (ISCAS'94), London, UK, June 1994, pp. 132-136.

[3] Corella, F.; Zhou, Z.; Song, X.; Langevin, M.; Cerny, E.: Multiway Decision Graphs for Automated Hardware verification. To appear in the journal of *Formal Methods in System Design*. Available as IBM research report RC19676(87224), July 1994.

[4] Curzon, P.: The Formal Verification of the Fairisle ATM Switching Element; Technical Reports No. 328 & No. 329, University of Cambridge, Computer Laboratory, March 1994.

[5] Edgcombe, K.: *The Qudos Quick Chip User Guide*; Qudos Limited.

[6] Gordon, M.; Melham, T.: *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*; Cambridge, University Press, 1993.

[7] McMillan, M.: *Symbolic Model Checking*; Kluwer Academic Publishers, Boston, Massachusetts, 1993.

[8] Leslie, I.; McAuley, D.: Fairisle: An ATM Network for Local Area; *ACM Communication Review*, Vol. 19, No. 4, September 1991, pp. 237-336.

[9] Thuau, G.; Berkane B.: A Unified Framework for Describing and Verifying Hardware Synchronous Sequential Systems; Journal of *Formal Methods in System Design*, Vol. 2, 1993, pp 259-276.