The Impact of Design Changes on Verification Using MDGs

M. Hasan Zobair, Sofiène Tahar and Paul Curzon[§]

ECE Dept., Concordia University, Montreal, Canada
Email: {mh_zobai, tahar}@ ece.concordia.ca

[§]School of Computing Science, Middlesex University, London, U.K.
Email: p.curzon@mdx.ac.uk

Abstract

In this paper, we investigate the impact of design changes on formal verification using the MDG (Multiway Decision Graphs) tools. In particular, we would like to determine whether the design changes that make verification by interactive theorem proving simpler, also make verification by automated decision diagram approach simpler as well. The design we consider is the Fairisle 4 by 4 switch fabric which has been used for real applications in the Cambridge ATM Fairisle network. A major consideration was that design change decisions should not compromise other design goals such as performance and functionality. The specification and verification obtained in MDG demonstrated the expected positive impact of these design changes.

1. Introduction

As communication networks become all pervasive, the consequences of errors in the design or implementation of network components become increasingly important. The validation of network components is at best difficult. Simulation cannot uncover all errors in an implementation because only a small fraction of all possible cases can be considered. Formal verification is a different technique that can alleviate this problem, because the correctness of a formally verified design implicitly involves all cases regardless of the input values [6].

In this paper, we investigate whether the formal verification of an ATM design can be simplified by making design changes: that is whether a notion of "Design for Verifiability", similar to that for testability, is of practical interest. Curzon [3][4] introduced this idea in the context of interactive proof. By using the HOL theorem prover [5], he suggested that the cost of verification in terms of time can be reduced by making appropriate design changes. Here, we investigate whether the same design changes also reduce the verification cost while using the MDG tools [1].

Our investigation involved the verification of an existing hardware design which was designed at the University of Cambridge. The component we considered is the Fairisle 4 by 4 switching fabric which performs the actual switching of data cells and forms the heart of the ATM Fairisle communication network [7]. The Cambridge Fairisle switch fabric verification had been done by Curzon [2] using the HOL theorem prover. Tahar *et al.* [9] verified the same fabric in an automatic fashion using the MDG (Multiway Decision Graphs) tools. While verifying the original description of the switch fabric which we refer to as the *Original* switch fabric, Curzon *et al.* [4] noted the factors that were increasing the verification cost in terms of time. It became obvious that particular aspects of the behavioral specification were lengthening the verification time by significant amounts. Moreover, by changing the behavior of the switch fabric, which is controlled by the environment of the switch fabric, i.e., port controllers, the problems would have been removed. While changing the actual design, Curzon *et al.* [4] were concerned that such changes should not affect the performance or functionality of the device. We will refer to the modified design which includes the suggested design changes during the interactive proof, as the *Cleaned* version.

The outline of this paper is as follows: In Section 2, we describe the *Original* version of the Fairisle switch fabric in terms of behavioral and structural description. In Section 3, we describe the changes to the fabric that were suggested by the verification attempt using a theorem prover. In Section 4, we describe the verification of the *Cleaned* version in MDG. In Section 5, we compare and contrast different aspects of the *Cleaned* and *Original* versions of the switch fabric and Section 6 concludes the paper.

2. The Fairisle ATM Switch

The Fairisle ATM switch consists of three types of components: *input port controllers*, *output port controllers* and a *switch fabric* (Figure 1). It switches ATM cells from the input ports to the output ports. A cell consists of a *header* (a one-byte tag containing routing information as shown in Figure 2) and a fixed number of data bytes. The port con-

0-7803-5957-7/00/$10.00

trollers synchronize incoming data cells, append headers to the front of the cells, and send them to the fabric. The fabric waits for cells to arrive, strips off the tags, arbitrates between cells destined to the same output port, sends successful cells to the appropriate output port controllers, and passes acknowledgments from the output port controllers to the input port controllers. If different port controllers inject cells destined for the same output port controller into the fabric at the same time, then only one will succeed and the others must retry later. The header also includes a *priority* bit that is used by the fabric for arbitration which takes place in two stages.

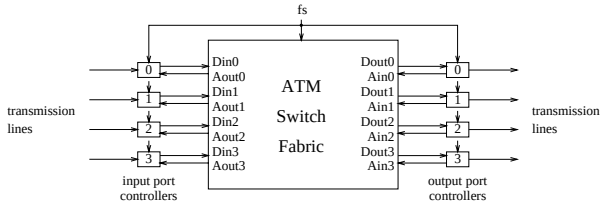

Figure 1. The structure of the Fairisle ATM switch

High priority cells are given precedence. The choice within both priorities is made on a round-robin basis. The input controllers are informed of whether their cells were successful using acknowledgment signals. The fabric sends a negative acknowledgment to the unsuccessful input ports, but passes the acknowledgment from the requested output port controllers to the successful input port. The port controllers and the 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* (f_s) signal (Figure 1). It ensures that the port controllers inject data cells into the fabric so that the headers arrive together. Here we are concerned with the verification of the *switch fabric*.

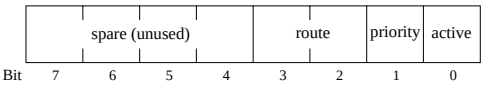

Figure 2. The routing tag of a Fairisle ATM cell

3. MDG Modeling of the *Cleaned* Fabric

The new design of the Fairisle switch fabric incorporates the following changes without any significant loss of functionality:

- The header arrives at least 5 cycles after the frame start signal.
- The header and frame start must not occur together.
- Internal delays were added to the datapaths so that no extra cell byte is lost.
- Minor changes to the internal timing of the data switch so it reads two grant signals at a more sensible time.

Inspired by [3][4] and the verification of the *Original* design of the Fairisle switch fabric using the MDG tools [9], we derived an MDG description of the *Cleaned* version of the switch fabric. The behavioral specification of the switch fabric is represented in the form of an Abstract State Machine (ASM). We investigated the modified behavior of the switch fabric under the control of the environment.

3.1. Environment for the port controllers

The timing-diagrams in Figure 3 represent the expected behavior of the *Cleaned* version of the switch fabric during an active frame. Based on this and similar sets of timing-diagrams we derived our environment state machine which controls the changed input-output behavior of the switch fabric. After the *frame start* (at time t_s), the switch waits for the headers to appear on the input lines *Din*. After the arrival of the headers (at time t_h), an arbitration between the inputs clashing for the same output is done in at most 2 cycles. The successful cells (bytes that follow the headers on *Din*) are transferred to the corresponding output port (*Dout*) with a delay of 5 cycles while acknowledgment (*Ain*) starting at time t_h+3 traverse in the opposite direction without any synchronous delay. Note that the last 5 cycles t_e-1 to t_e-5 of a frame do not accept any data.

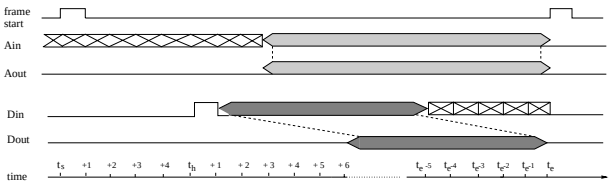

Figure 3. Timing behavior during an active frame

We modified the original environment state machine as given in the verification design documentation of the Fairisle switch [9] to comply with the modifications suggested by Curzon *et al*. [4]. Figure 4 shows the modified environment state machine which reflects the above timing diagram for a 64 clock cycles frame. In [10], we describe in details the four modified assumptions about the environment of the switch fabric and the reasons of the modifications from [9].

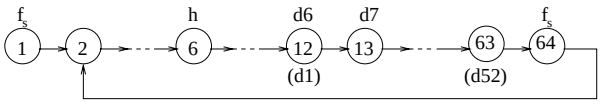

Figure 4. The new environment state machine

In Figure 4, there are 64 states enumerated by integers (using MDG-HDL, state variables can be described as a concrete variable of sort [1, 2, .., 64]). State transitions are denoted by arrows. In analogy to the environment state machine of [9], we used f_s, h and d_i to denote the *frame start* signal, the header of an active cell and the data processing in that state, respectively. The notation (d_i) in Figure 4 indicates that data is switched to the output port in that state.

3.2. Behavioral Specification

Inspired by the constraints from the above environment state machine which represents the port controller behavior, we describe in the following the overall behavior of the switch fabric. It can be expressed in the form of a finite state machine (ASM) having 12 states (Figure 5). To simplify the presentation, the symbols *s* and *h* denote a *frame start* (f_s=1) and the arrival of headers (active bit set in at least one *Din*), respectively; "~" denotes negation, and the symbols *a*, *d* or *r* inside a state represent the processing of the acknowledgment output (*Aout*), the data output (*Dout*) or round-robin arbitration, respectively.

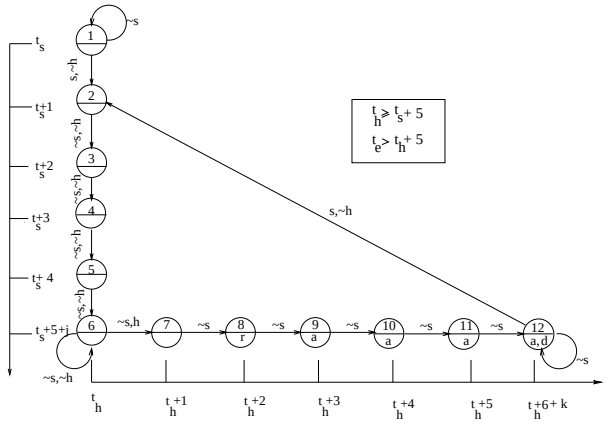

Figure 5. ASM of the *Cleaned* switch fabric

Two time axes illustrate the time units of a frame to which the transitions correspond. The symbols t_s and t_h represent the arrival time of a *frame start* signal and the arrival time of a header, respectively. The end time (t_e) of a frame is not given, since it is the same as t_s of the next frame. State 1 is the initial state from which a frame may begin without any delay. This complies with the first constraint on the environment of the switch. After a waiting loop for the first *frame start* in state 1, states 2 to 6 describe the behavior of the fabric after the arrival of a *frame start*, with at least a five-cycle delay before the arrival of the headers. This delay represents the second constraint on the environment. The waiting loop for the arrival of the headers in state 6 is shown by a natural number *j*. States 7 to 12 describe the behavior of the fabric after the arrival of the headers. When the headers arrive, the *frame start* signal must not arrive before at least 6 cycles to comply with the third constraint on the environment. The arrival of a *frame start* in state 12 complies with the last constraint of the environment which requires that the next frame does not arrive before 11 cycles from the current *frame start*. After arbitration (state 9), the switch fabric transfers the acknowledgments in each cycle of a frame and switches data, delayed by three cycles. This delay is represented using the sequence of transitions from state 9 to 12. The loop in state 12 represents the transmission of data and acknowledgments in the remaining cycles of the cell (indicated by a natural number *k*). The arrival of a *frame start* in state 12 marks the beginning of another frame. Here, a new sequence of state transitions along the t_s axis progresses similarly as in states 2, 3, 4, 5 and 6 described above.

The *Original* ASM of the switch fabric used in the MDG verification of the Fabric is given in [9]. To model the computation in MDG of the acknowledgments, the data outputs and the round-robin arbitration, we use the techniques described in [9].

3.3. Structural Implementation

Figure 6 shows a block diagram of the switch fabric implementation. It consists of an arbitration unit, an acknowledgment unit and a dataswitch unit. The arbitration unit is composed of a Timing unit, a Decoder, a Priority Filter and a set of Arbiters. For more details about the implementation refer to [2].

To reflect the modifications suggested in [4], minor changes were made to the Timing unit, Arbiters, control path between the Arbitration and Dataswitch units and datapath to the Dataswitch unit of the original implementation. The modified Timing module ensures that the header and frame start signals must not occur together. The *frame start* signal just gets there 5 cycles later as required to make it trigger 5 cycles later. The shaded boxes in Figure 6 represent the modified modules in the *Cleaned* implementation.

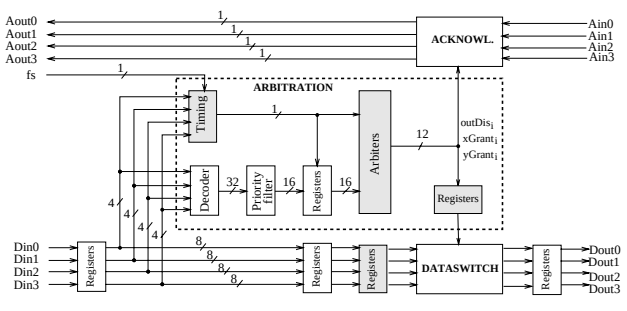

Figure 6. *Cleaned* fabric implementation

The Arbiters generate the *output disable* and *grant* signals. The *Original* Arbiters disable the outputs one cycle earlier than is desirable. On each cycle, to determine which output the current byte should be sent to, the Dataswitch consults the two bits of the grant control signal produced by the Arbiter. One of those bits is sampled on the cycle before it is used, but the other is sampled on the same cycle. This ultimately means that the grant signal for the last cycle cannot be used as its value changes between the bits being sampled. The problem is removed by adding extra delays across the path to the Dataswitch. The *Cleaned* Arbiters disable the outputs one cycle later than the *Original* one, so that the last bytes of a cell are not ignored.

The Dataswitch module chooses a word to be output to each of the output ports. It delays the data long enough for an arbitration decision to be made. To comply with the extra delay in the arbitration unit, minor changes had been made to its internal timing so it can read the two grant lines at a more sensible time. To do so, an extra register is added across the datapath to the Dataswitch unit.

4. MDG Verification of the *Cleaned* Fabric

The *Cleaned* version was formally verified, based on the modifications described in the previous section. In the following two sub-sections we describe property checking and sequential equivalence checking of this modified switch fabric.

4.1. Property Checking

We applied property checking to ascertain that both implementation and specification of the switch fabric satisfy some specific requirements while working under the control of the new environment, i.e., port controllers. Sample properties are correct circuit-reset and correct data-routing. Using the time points t_s, t_h and t_e, as introduced in Section 3.1, we described several properties which reflect the modified behavior of the switch fabric. The verification of the *Cleaned* fabric was done using four properties similar to those described in [9].

Properties 1 and *2* deal with the reset behavior of the circuit, while *Property 3* and *4* describe specific behaviors of the switching of cells.

- *Property 1*: From t_s+5 to t_h+5, the default value (*zero*) appears on the data output port $Dout_i$, where *zero* is a generic constant and i = 0,...,3.
- *Property 2*: From t_s+1 to t_h+2, the default (0) appears on the acknowledgment output port $Aout_i$ where i = 0,...,3.
- *Property 3*: From t_h+6 to t_e-1 (i.e., 1 cycle before the next t_s), if input port i, i ={0,..,3}, chooses output port j, j ={0,..,3}, with the priority bit set in the header, and no other input ports have their priority bits set, the value on $Dout_j$ will be equal to that of Din_i 5 clock cycles earlier.
- *Property 4*: From t_h+3 to t_e-1 if input port i chooses output port j with the priority bit set in the header, and no other input ports have the priority bit set, the value on $Aout_j$ will be that of Ain_i.

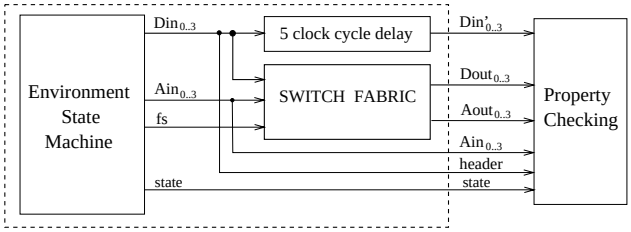

Figure 7. Composed state machine for property checking

To verify these safety properties, we composed the fabric (specification or implementation) with the environment state machine as shown in Figure 7. This verification approach is inspired by the technique described in [9]. By using the property checking facility of the MDG tools, we checked in each reachable state if the outputs satisfy the logic expression of the property which should be true over all reachable states. The experimental results from the verification of all the properties stated above for both implementation and specification, are given in Table 1. All experimental results were obtained on a Sun Ultra SPARC 2 workstation (296MHz / 768 MB) and include CPU time in sec., memory usage in MB and the number of MDG nodes generated.

4.2. Equivalence Checking

The original design of the switch fabric was described at the gate-level in Qudos HDL. The authors in [9] translated the Qudos HDL description into MDG-HDL using the same collection of gates. By abstracting the data lines from a bundle of bits to a compact word of abstract sort, we also obtain an abstract RTL model of the switch fabric. This RTL model will be equivalent to the original gate-level description if it produces the same output as the original gate-level for all input sequences. We adopted the modifications mentioned in Section 3.3 to both the gate-level and RTL models of the switch fabric to reflect the design changes. We then proceeded with the equivalence checking between gate-level and RTL description using the sequential equivalence checking of MDG. This was achieved after instantiation of the RTL model to 8-bits [9].

Using the sequential equivalence checking facility of the MDG tool, we also verified that the abstract RTL implementation of the switch fabric complied with the new specification of the behavioral model described in Section 3.2. We obtained a complete verification of the switch fabric from a behavioral specification down to the gate-level implementation. The experimental results are given in Table 1.

Table 1. Verification of the *Cleaned* version

Verification Phases	CPU time (s)	Memory (MB)	No. of Nodes
Reach. Analy.			
Specification	180.40	32	73157
Implementation	219.22	34	90208
Prop. Checking			
Specification			
Prop. 1	186.65	27	74948
Prop. 2	199.67	23	75287
Prop. 3	195.23	23	73020
Prop. 4	160.43	28	72843
Implementation			
Prop. 1	173.81	32	88085
Prop. 2	151.53	31	89738
Prop. 3	166.21	30	90554
Prop. 4	164.76	32	90933
Equv. Checking			
RTL vs. Beh. +	1934.56	148	230798
RTL vs. Gate	30.85	13	13899

5. Summary of Results

The motivation of this work was to compare the formal verification, in terms of time, of the *Cleaned* version of the Fairisle ATM switch fabric with the *Original* version using MDG tool. Timing aspect of formal verification is an important issue to the industrial community. We compare these two versions with respect to *machine-time* and *human-time*.

Human time spent on the verification of the *Original* version was longer than that of the *Cleaned* version. The amount of work in re-running a verification of a modified design is minimal compared to the initial effort since the latter includes all the modeling aspect. In the verification by MDG tools, manual intervention is needed for variables ordering which has an impact on the verification time.

In the verification of the *Original* version, much of the time was spent on determining a suitable variable ordering. As there were no major changes to the *Original* version, we did not spend much time on redetermining a suitable variable ordering. The translation of the original Qudos HDL design description to the MDG-HDL gate-level structural model took about one person-week as described by Tahar *et al.* [8]. The time spent on the modification of the structural description of the design for the *Cleaned* version was four person-days. Because the verifier needs to understand the design thoroughly, the time spent for understanding and writing the behavioral specification of the *Original* version was about four person-weeks. On the other hand, for the *Cleaned* version it took two person-weeks. In the verification of the *Original* version, the time required to setup four properties, to build the environment state machine, to conduct the property checking both on the implementation and the specification and to interpret the results was about three person-weeks. For the *Cleaned* version, building a new environment state machine and conducting the property checking on both the implementation and the specification took about two person-weeks. The equivalence checking of the RTL implementation with its behavioral specification and the RTL model against the gate-level model of the *Original* version required about two person-weeks due to the adoption of abstraction mechanisms and correction of description errors for the RTL implementation. On the other hand for the *Cleaned* version, it took about one person-week. The summary of the differences between the *Original* version and the *Cleaned* version, in terms of *human-time* taken during the verification phase, is given in Table 2.

Table 2. Summary of human-time taken

Verification Phase	Cleaned version	Original version
Behavioral spec. modeling	Two person-weeks	Four person-weeks
Gate and RTL impl. modifications	Four person-days	One person-week
Env. state machine and property checking	Two person-weeks	Three person-weeks
Equiv. Checking: RTL vs. Beh. Spec. + RTL vs. Gate-level	One person-week	Two person-weeks

To demonstrate the reduced verification time we compare the *machine-time* taken to complete the *Cleaned* version verification with that for the *Original* version. The machine-time taken by the *Cleaned* version for both the Property checking and the Equivalence checking was reduced by a significant amount of CPU time to that of the *Original* ver-

sion. The differences between the verification of the *Cleaned* and the *Original* versions are illustrated with respect to CPU time taken, memory usages and MDG nodes generated (compare Tables 1 and 3).

Table 3. Verifications of the *Original* version

Verification Phases	CPU time (s)	Memory (MB)	No. of Nodes
Reach. Analy.			
Specification	188.59	36	74130
Implementation	232.74	35	90319
Prop. Checking			
Specification			
Prop.1	251.53	25	74554
Prop. 2	279.02	26	76265
Prop.3	257.52	25	74636
Prop.4	236.42	25	74441
Implementation			
Prop.1	235.34	34	92229
Prop.2	233.49	35	93882
Prop.3	205.04	33	92052
Prop.4	268.75	84	225486
Equv. Checking			
RTL vs. Beh +	2210.22	162	245707
RTL vs. Gate	30.85	13	13899

6. Conclusions

In this paper, we have demonstrated that design for verifiability can have a significant effect on the speed of verification using automated decision diagram based technique. The same result was obtained by using interactive proof with the HOL theorem prover for the same design verification. The difference in nature of these two verification methodologies suggests design for verifiability can be widely applicable. One of the motivations of this work was to show that designers can ease the verification task without compromising other design considerations. Our investigation suggests that one way this can be done is by ensuring that the operating assumptions of modules are as few and as simple as possible. The designer may have to work a little harder to ease the verifier's task. However, the result is a much cleaner design. It thus can be done early in the design cycle. The development of design constraints for formal verification would be useful. This is vital for safety-critical systems where formal verification techniques are most likely to be used.

The implementation we considered for this investigation is the Fairisle 4 by 4 switch fabric which forms the heart of the ATM Fairisle communication network. We made some changes to the timing constraints of the fabric which is controlled by the environment of the fabric, i.e., port controllers. By changing these timing constraints we made the operating assumptions of the fabric simpler and cleaner. We also changed the design of the Arbiters, the Timing unit, the control path between the Arbitration and Dataswitch unit and datapath to the Dataswitch unit without loss of any significant functionality. The verification time taken by both *human* and *machine* for the modified design was much less than that of the original design as demonstrated in the previous section. Based on the above statistics we can conclude that the verification time can be saved if a notion of "design for verifiability" is integrated into the design process itself.

References

[1] F. Corella, Z. Zhou, X. Song, M. Langevin and E. Cerny, "Multiway Decision Graphs for Automated Hardware Verification", *Formal Methods in System Design*, Vol. 10, pp. 7-46, February 1997.

[2] P. Curzon, "The Formal Verification of the Fairisle ATM Switching Element", *Technical Reports 328 & 329*, University of Cambridge, Computer Lab., March 1994.

[3] P. Curzon, "Tracking Design Changes with Formal Machine-checked Proof", *The Computer Journal*, Vol. 38, No. 2, pp. 91-100, July 1995.

[4] P. Curzon and I. Leslie, "Improving Hardware Designs whilst Simplifying their Proof", *Designing Correct Circuits*, Workshops in Computing, Springer-Verlag, 1996.

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

[6] C. Kern and M. Greenstreet, "Formal Verification in Hardware Design: A Survey", *ACM Transactions on Design Automation of E. Systems*, Vol. 4, pp. 123-193, April 1999.

[7] I. Leslie and D. McAuley, "Fairisle: An ATM Network for Local Area", *ACM Communication Review*, Vol. 19, pp. 237-336, September 1991.

[8] S. Tahar and P. Curzon, "Comparing HOL and MDG: A case study on the Verification of an ATM Switch Fabric", *Nordic Journal of Computing*, Vol. 6, pp. 372-402, 1999.

[9] S. Tahar, X. Song, E. Cerny, Z. Zhou, M. Langevin and O. Ait-Mohamed, "Modeling and Verification of the Fairisle ATM Switch Fabric using MDGs", *IEEE Transactions on CAD of Integrated Circuits and Systems*, Vol. 18, No. 7, pp. 956-972, July 1999.

[10] M. Zobair, S. Tahar and P. Curzon, "The Impact of Design Changes on Verification Using MDGs", *Tech. Report*, Dept. of ECE, Concordia University, October 1999.