ICCD'98
IEEE International Conference on Computer Design
Austin, Texas, USA, October 1998
IEEE Computer Society Press, pp. 195-198

Model Checking of a Real ATM Switch

Jianping Lu, Sofiène Tahar
Concordia University, ECE Dept.
Montreal, Quebec, H3G 1M8 Canada
{jianping, tahar}@ece.concordia.ca

Dan Voicu , Xiaoyu Song
University of Montreal, IRO Dept.
Montreal, Quebec, H3C 3J7 Canada
{voicu, song}@iro.umontreal.ca

Abstract

In this paper we present our experience on model checking of an Asynchronous Transfer Mode (ATM) switch using the Verification Interacting with Synthesis (VIS) tool. The switch we considered is in use for real applications in the Cambridge Fairisle network. It is composed of four input/output port controllers and a switch fabric, and contains around 1MB memory, 2KB FIFO buffer and 800 registers (latches). To overcome state space explosion, we adopted several abstraction and reduction techniques to reduce the model, and applied compositional reasoning combined with a novel property division approach. Using the above techniques, we succeeded in verifying the entire switch at different hierarchy levels within reasonable CPU time.

1. Introduction

Although ATM is hailed as the most important communication mechanism in the foreseeable future, there is currently little experience on the application of formal verification to ATM network hardware. For instance, Curzon [5] formally verified the 4 by 4 fabric of the Cambridge Fairisle switch using the HOL theorem prover [6]. Tahar *et al*. [13][7] verified the same switch fabric in an automatic fashion using MDGs (Multiway Decision Graphs) [4] by property checking and equivalence checking. Another case of formal verification of an ATM circuit was done by Chen *et al*. [3] at Fujitsu Digital Technology Ltd., where the authors identified a design error in an ATM circuit using the tool SMV (Symbolic Model Verifier) [11] by verifying some properties expressed in CTL (Computational Tree Logic) [11]. Recently, Rajan *et al*. [12] used a combination of theorem proving and model checking to discover bugs in a high-level ATM model that was presumed correct using simulation.

In this paper, we present our methods and experimental results on the model checking of a complete ATM switch using VIS [1]. The switch we investigated is part of the Fairisle ATM network [8], designed and in use at the Computer Laboratory of the University of Cambridge.

The switch contains around 1MB memory, 2KB FIFO buffer and 800 latches. This is too large to be directly verified by model checking. Therefore, we first wrote Verilog RTL descriptions of the port controllers and the switch fabric at different levels of abstraction using model reduction. Based on the reduced port controllers and fabric models, we performed the model checking of the entire ATM switch. Due to state space explosion, we adopted compositional reasoning [9] associated with two novel property division techniques which manage the proof in simpler consecutive steps. We found no errors in the ATM switch design. We succeeded, however, in detecting several injected design errors in both the port controllers and the fabric models.

The rest of the paper is structured as follows: In Section 2, we describe briefly the structure and behavior of the Fairisle ATM switch and introduce the modeling methods we adopted. In Section 3 and 4, we illustrate the model checking on the port controllers and the fabric models, respectively. In Section 5, we elaborate the property division techniques we developed. In Section 6, we describe the verification of the complete ATM switch model. We conclude the paper in Section 7.

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). The input port controllers receive ATM cells from transmission lines, store them in queues, dequeue them, append *fabric header* and *output header*, and then transfer the cells into the switch fabric. An ATM cell consists of 48 data bytes plus a 4 bytes header. The input port controllers also receive acknowledgment signals from the fabric and decide whether to send new data, to retransmit previous cell, or to stop sending data. The switch fabric transfers data cells from input port controllers to the output port controllers and passes the acknowledgment signals from the output port controllers to the input port controllers according to the fabric header. If cells on common destination clash, the fabric arbitrates using round-robin allocation. The output port controllers decide whether to transfer data to transmission lines or loop back data to the input port controllers according to the output header. They also detect errors in received cells, and send the acknowledgment signals to the switch fabric. The port controllers and switch fabric all use the same clock, and they also use a higher-level cell frame clock: the *frameStart* signal (*fs*). It ensures that the port controllers inject data cells into the fabric synchronously so that the fabric headers arrive at the same time.

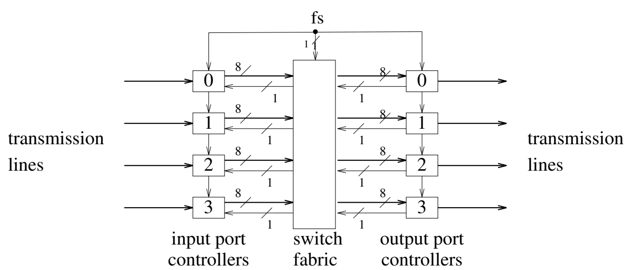

Figure 1. Structure of the Fairisle ATM switch

The port controllers and the fabric are both too large to directly perform model checking on any of them due to state space explosion. Therefore, we introduced a number of abstractions and reductions for both the port controllers and fabric.

In the port controller model: (1) we abstracted away the memory by using a cell memory interface while keeping pointers in separate queues represented by functionally equivalent models. (2) FIFOs were reduced to primary inputs where only the socket bits are consulted to distinguish the beginning of a cell from a cell body. (3) The queue length was abstracted for certain safety properties by using queue models with no content but only status bits that non-deterministically indicate whether the queue is full or empty, or neither full nor empty.

For the switch fabric, we wrote behavioral Verilog RTL descriptions of the switch fabric submodules following the same design hierarchy as the netlist description. We then reduced the datapath of the *dataswitch* from 8 bits to 1 bit in the RTL description of the fabric because the behavior and the structure of 1-bit datapath are exactly the same as that of the other 7 bits.

After establishing the port controllers and switch fabric models, we first checked a number of *local* properties on each model. These are reported in the coming Sections 3 and 4, respectively. Later in Section 6, we present the verification of *global* properties on the complete ATM switch.

In following sections, we use " ! ", "*", "+" and "−>" to denote logical *"not", "and*", "or" and *"imply"*, respectively. We use *i, j, x, y, z* to represent a port number which could be 0 (00), 1 (01), 2 (10) or 3 (11), where *i, x, y, z* are disjoint values (for different input port numbers).

3. Model Checking of the Port Controllers

We verified in total 58 properties for the port controllers. All properties were expressed in CTL. We also imposed several fairness constraints with the liveness properties, to avoid infinite idle input sequences. In the following, we show one example property. More examples of model checking of the port controllers are reported in [10].

Property 1: A negative acknowledge implies that the port controller stops sending data, and a positive acknowledge will make the port controller transmit data continually, i.e., in CTL:

```
AG(ackOut.i = 0 * (transmit_state.i = send) ->
AX(transmit_state = stop));
AG(ackOut.i= 1 * (transmit_state.i = send) ->
AX(transmit_state = send));
```

In above CTL expressions, *transmit_state.i = send* and *transmit_state.i = stop* mean that the input port controller *i* is in the *send* and the *stop* state, respectively. *ackOut.i= 0* and *ackOut.i=1* indicate the negative and positive acknowledgment, respectively. Since each port controller model includes around 50 latches, the model checking of the port controllers consumed a reasonable amount of CPU time. For example, the model checking of the above property took around 21 seconds of CPU time (elapse time). All the properties of the ATM switch were checked on a Sun Sparc 20 (55MHz/256MB). We found no errors in our model. However, we introduced some errors on purpose and they were all successfully identified [10].

4. Model Checking of the Switch Fabric

The switch fabric forms the heart of the ATM switch. It receives the input signals from both the input and output port controllers, so the switch fabric is not a "closed" system. We have hence establish an environment state machine to imitate the associated behavior of input and output port controllers in a similar way to the concept of the compositional reasoning theory described in [9].

The switch fabric's interface with the port controllers consists of the signal *frameStart*, 32-bit data inputs, 32-bit data outputs, 4-bit acknowledgment inputs and 4-bit acknowledgment outputs (Figure 1). We modeled the switch fabric environment (port controllers) as a finite state machine (Figure 2) representing the behavior of the port controllers. The frame start (t_s), header arrival (t_h) and frame end (t_e) correspond with states S0, S2 and S6, respectively. States S1 to S7 represent the cyclic behavior of the fabric, where one cycle corresponds to one frame.

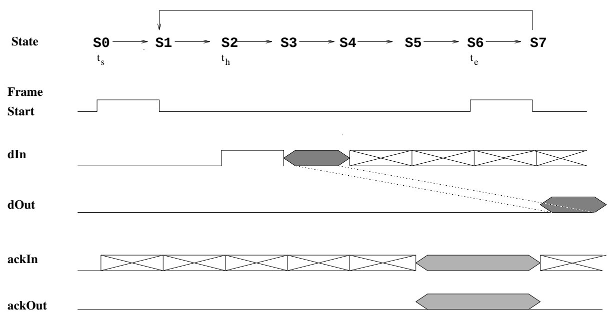

Figure 2. Abstracted environment state machine with related timing diagrams

After establishing the environment state machine, we considered a number of properties of the fabric including liveness and safety properties. For illustration purposes, we give in following one example property. More example properties are reported in [10].

Property 2: An acknowledge signal associated with a cell with priority could be transferred from the output port of the fabric to the input port of the fabric according to the fabric header, i.e., in CTL:

```
AG ( Din.head.i [3:0] = j11 * Din.head.x [1] = 0
* Din.head.y[1] = 0 * Din.head.z[1] = 0 * ackIn.j
= 0 * ( state = S5 + state = S6) -> ackOut.i = 0);
AG ( Din.head.i [3:0] = j11 * Din.head.x [1] = 0
* Din.head.y[1] = 0 * Din.head.z[1] = 0 * ackIn.j
= 1 * (state = S5 + state = S6) -> ackOut.i = 1);
```

where *ackIn.j* and *ackOut.i* signals denote the acknowledgment signal into the output port *j* of the fabric and the acknowledgment signal out of the input port *i* of the fabric, respectively. *S5* and *S6* are the states where acknowledgment signals are transferred. The above property includes 32 combinational CTL expressions.

Based on the fabric model, the property passes the model checking. However, since the abstracted fabric model contains over 100 latches, checking all 32 combinational CTL expressions consumed 13248 seconds of CPU time (elapse time). We also found that most of the properties checked on the abstracted fabric model were verified with such unreasonably long CPU time (days of machine usage). Therefore, we applied a new technique based on compositional reasoning and property division to enhance the model checking. This will be elaborated in next section.

No errors were discovered in the model checking of the original design of the switch fabric. For experimental purposes, we injected several design errors into the implementation. All errors were detected by model checking and diagnosed using the counterexamples generated by the VIS tool. Injected errors and further experimental results are reported in [10].

5. Property Division Techniques

In order to capture the behavior of the whole switch including both the port controllers and the switch fabric, we established a number of *global* properties for the entire ATM switch. However, because of state space explosion, we did not succeed in directly verifying any of these properties on the whole switch model. To verify the entire ATM switch and in order to enhance the model checking of the switch fabric and port controllers, we propose a new property division technique. In this approach, we try to subdivide a property into several sub-properties. Similar to compositional reasoning [9], which exploits the natural decomposition of a system into communicating parallel processes to infer properties for a complete system from individual components, we check each sub-property in its corresponding submodule and environment. If all the sub-properties pass the model checking, their verification is still valid in the whole module. Then the property can be deduced from the sub-properties. We call this *Property Division*. We propose two approaches to divide a property. One is *Cascade Property Division*, and the other is *Parallel Property Division*. Cascade property division is to divide a property into several sequentially related sub-properties. Parallel property division splits a property into several parallel sub-properties without introducing any intermediate variable, and checks every sub-property on a corresponding reduced submodule that is extracted from the design. For illustration purposes, we use the first CTL expression of *Property 2* (see Section 4) to explain the cascade property division approach.

The original CTL expression is:

```
AG ( Din.head.0 [3:0] = 0011 * Din.head.1 [1] = 0
* Din.head.2[1] = 0 * Din.head.3[1] = 0 * ackIn.0
= 0 * ( state = S5 + state = S6) -> ackOut.0 = 0);
```

To divide this expression, we introduced the intermediate signals (variables) *xGrant[0], yGrant[0]* and *outputDisable[0]*, yielding the following two sub-expressions:

```
AG (Din.head.0 [3:0] = 0011 * Din.head.1 [1] = 0
* Din.head.2[1] = 0 * Din.head.3[1] = 0 * ( state
= S5 + state = S6)-> xGrant[0] = 0 * yGrant[0] =
0 * outputDisable[0] = 0);
AG ((state = S5 + state = S6) * ackIn.0 = 0 *
xGrant[0] = 0 * yGrant[0]= 0 * outputDisable[0] = 0
-> ackOut.0 = 0);
```

which are verified in the arbitration and the acknowledgment submodules of the fabric with their corresponding environments, respectively. Since the two sub-expressions are also valid in the whole ATM model, we can infer through substitution the original CTL expression of *Property 2* [9]. Using cascade property division, the CPU time for checking *Property 2* is enhanced by 41 times. Through the use of the cascade and parallel property division approach, we have been able to enhance the CPU time for checking all the properties of the switch fabric and port controllers [10].

6. Model Checking of the entire ATM Switch

In the entire ATM switch model, we verified a number of *global* properties which capture the behavior of the whole switch including both the port controllers and the switch fabric. For illustration purposes, in following we present one example global property and explain how we applied the property division. More examples are reported in [10].

Property 3: An acknowledge signal associated with a cell with priority will be transferred from an output port of the fabric to the destined input port according to the fabric header; once the corresponding input port controller receives positive acknowledge signal, it will transmit data continually, otherwise it will stop sending data. We express this in CTL as:

```
Din.head.i [3:0] = j11 * Din.head.x[1] = 0 *
Din.head.y[1] = 0 * Din.head.z[1] = 0 * (state =
S5 + state = S6) * ackIn.j = 0 * transmit_state.i
= send -> AX (transmit_state.i = stop));

Din.head.i [3:0] = j11 * Din.head.x[1] = 0 *
Din.head.y[1] = 0 * Din.head.z[1] = 0 * (state =
S5 + state = S6) * ackIn.j = 1* transmit_state.i
= send -> AX (transmit_state.i = send ));
```

This property could be subdivided into following two sub-properties:

Sub-prop. 3-1:
```
AG ( Din.head.i [3:0] = j11 *
Din.head.x[1] = 0 * Din.head.y[1] = 0 *
Din.head.z[1] = 0 * ackIn.j = 0 * (state
= S5 + state = S6) -> ackOut.i = 0);

AG ( Din.head.i [3:0] = j11 *
Din.head.x[1] = 0 * Din.head.y[1] = 0 *
Din.head.z[1] = 0 * ackIn.j = 1 * (state
= S5 + state = S6) -> ackOut.i = 1);
```

Sub-prop. 3-2:
```
AG (ackOut.i = 0 * (transmit_state.i =
send) -> AX(transmit_state= stop));

AG (ackOut.i= 1 * (transmit_state.i =
send) -> AX(transmit_state= send));
```

which are equivalent to *Property 2* and *1*, respectively. According to the compositional reasoning theory [9], *Property 2* and *1* are verified on their corresponding submodules and ATM environments, so they are also valid in the whole switch model. If we use the assumption of *Property 2* to substitute the assumption of *Property 1*, we will get *Property 3*, hence *Property 3* is valid.

7. Conclusions

In this study, we explored the model checking of a real ATM switch and detected several injected design errors in it. The main contributions of this work are (1) the establishment of abstracted and reduced models of the switch at different levels of the design hierarchy, (2) the definition of a set of typical properties to the port controller, the fabric and the entire switch, and (3) the development of a new property division technique to avoid state space explosion.

Although the current application of VIS is limited to circuits of moderate size, we demonstrated that VIS can partially verify large circuits by using abstraction, reduction and property division. In this paper, we provided some practical ideas for ATM switch modeling and properties specification. The modeling techniques and properties are not restricted to the Fairisle ATM switch but are also generally applicable to other switches. In addition, we adopted the approach of an environment state machine for the fabric to restrict the possible inputs according to the behavior. This can be used as a general method for model checking. Moreover, we applied property division to enhance the model checking of the fabric, and further used property division to verify the whole ATM switch model. Although we applied our methods using VIS, our techniques can be adopted by any other automatic model checking tools.

References

[1] R. Brayton et. al. VIS: A System for Verification and Synthesis. Technical Report UCB/ERL M95, Electronics Research Laboratory, University of California, Berkeley, December 1995.

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

[3] B. Chen, M. Yamazaki, and M. Fujita. 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.

[4] 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, No. 1, 1997, pp. 7-46.

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

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

[7] M. Langevin, S. Tahar, Z. Zhou, X. Song, and E. Cerny. Behavior Verification of an ATM Switch Fabric using Implicit Abstract State Enumeration. *Proc. IEEE International Conference on Computer Design* (ICCD'96), Austin, Texas, USA, October 1996, pp. 20 - 26.

[8] I. Leslie and D. McAuley. Fairisle: An ATM Network for the Local Area. *ACM Communication Review,* Vol. 19, No. 4, September 1991, pp. 327-336.

[9] D. Long. Model Checking, Abstraction and Compositional Verification. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, July 1993.

[10] J. Lu, D. Voicu, S. Tahar, and X. Song. Model Checking of the Fairisle ATM Switch. Technical Report, Concordia University, June 1998.

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

[12] S. Rajan, M. Fujita, K. Yuan, and M. Lee. High-Level Design and Validation of ATM Switch. *Proc. IEEE International High Level Design Validation and Test Workshop* (HLDVT'97), Oakland, California, USA, November 1997.

[13] S. Tahar, Z. Zhou, X. Song, E. Cerny, and M. Langevin. Formal Verification of an ATM Switch Fabric using Multiway Decision Graphs. *Proc. IEEE Sixth Great Lakes Symposium on VLSI* (GLS-VLSI'96), Ames, Iowa, USA, March 1996, pp. 106-111.