Formal Verification of an Asynchronous MAC Layer Protocol in VIS

Hong Peng and Sofiène Tahar

Technical Report

Dept. of Electrical & Computer Engineering, Concordia University
1455 de Maisonneuve W., Montreal, Quebec, H3G 1M8 Canada
E-mail: {pengh,tahar}@ece.concordia.ca

May 1999

Abstract

Model checking techniques have established themselves as significant means in early detection of design errors. When a higher layer protocol is implemented in hardware and we want to verify its implementation satisfying the specification, it cannot be handled by software (protocol) model checking tools as most of these tools are based on an interleaving model unlike the hardware implementations, which are based on a synchronous concurrent model. In this paper, we describe our attempt in modeling and formally verifying the hardware implementation of an ATM ring media access (MAC) protocol using a hardware verification model checking tool, VIS. We succeeded verifying the original asynchronous hardware model of the protocol as well as a synchronous version of it. We also report comparative results of both approaches using VIS and SPIN independently and discuss some abstraction techniques we adopted in the verification.

Contents

1 Introduction

The protocol design of finite-state concurrent systems is today present in many fields, in particular in the design of digital and communication systems. Late detection of design errors in the design phase are very costly in any system development. These errors can delay the product deployment and even cause the failure of the product. The traditionally used simulation techniques cannot cover all design errors when the state space of the system is big. During the past years, model checking techniques have established themselves as significant means for early detection of design errors due to their ability of validation and conformance checking.

While lower parts of ISO protocols are implemented in hardware, higher layer protocols are traditionally implemented in software. These latter ones are usually verified using some specific protocol verification tools, like SPIN [5], which are based on interleaving models. In the interleaving model, only one component of the system state is allowed to change at a time. Today, in some high speed data communication architectures, like ATM (Asynchronous Transfer Mode), even some higher layer protocols are implemented in hardware in order to match the required performance. In contrast to software, hardware implementations are based on a synchronous model, where any number of components of the system state can change state at a time. A number of difficulties are however involved in the verification of such higher layer protocols using a hardware verification tool. For instance, in addition to some abstraction and reduction techniques used to avoid state space explosion, we might need to simulate an asynchronous protocol in the synchronous hardware verification tool environment.

The aim of this paper is to describe the formal verification in VIS (Verification Interacting with Synthesis) [2] of the asynchronous and synchronous implementations of a single ATM ring media access protocol (ATMR) [4] as a case study. VIS, developed by University of California at Berkeley, is a tool that integrates the verification, simulation, and synthesis of finite-state hardware systems. It uses a Verilog front-end and supports fair CTL model checking, language emptiness checking, combinational and sequential equivalence checking, cycle-based simulation, and hierarchical synthesis.

This paper was inspired by Charpentier and Padiou [3], who presented a pencil-and-paper verification of the ATMR protocol. The authors first described the protocol in the UNITY language [3], and then specified the correctness requirements of the protocol by means of the UNITY logic. Their validation abstracts away from any implementations, be it in a synchronous or in an asynchronous environment. The verification presented in the current paper has several advantages over this earlier work. For instance, our proof is fully mechanically checked, using a symbolic model checking tool rather than by manual proof or automated proof assistants. On the other hand, we verified an RTL (Verilog) implementation of the protocol that can be synthesized automatically into real gates and latches.

A number of other related work on the formal verification of protocols and ATM can be found in the open literature. These can be classified in three major categories. (1) formal verification of high layer software protocols, e.g., [6]; (2) formal verification of synchronous hardware protocol, e.g., verification of cache coherence protocols in [8]; and (3) formal verification of ATM hardware devices (not protocols), e.g., verification of ATM switch in [7] [10]. Our work distinguishes itself from these related publications by the fact that we verify the hardware implementation of an asynchronous high layer protocol (ATMR). We present some techniques on how to simulate the asynchronous ATMR protocol in a synchronous environment and also propose some abstraction and reduction approaches for the aimed verification. Furthermore, we report comparative verification results of the asynchronous and synchronous ATMR protocol using both VIS and SPIN independently, and also analyze the source of the complexity in the protocol checking.

The rest of the paper is structured as follows. The next section introduces the ATMR protocol to be verified. Section 3 discusses the modeling techniques we used for the asynchronous protocol. This includes the simulation of an asynchronous protocol in a synchronous environment and module abstraction. Section 4 describes the verification of the asynchronous protocol against a set of CTL consistency properties. Section 5 exposes a comparative verification of asynchronous and synchronous implementations of the ATMR protocol in VIS as well as in SPIN. Section 6 finally concludes the paper.

2 ATMR Model

The ATMR protocol [4] is a new ISO standard based on a high speed shared medium bus connecting a number of access nodes by channels in a ring form. Figure 1 gives an example ring with five nodes connected via a channel transferring cells between the nodes. For controlling access to this type of shared medium, the ring is first initialized

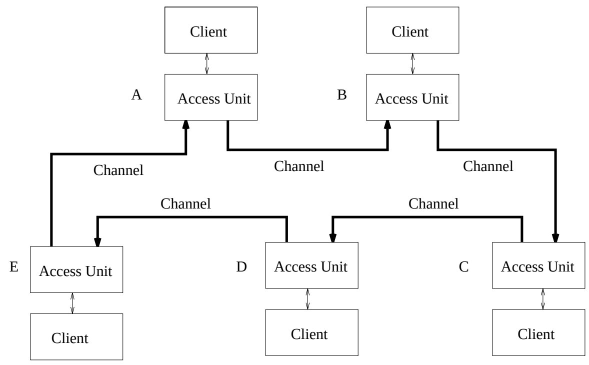

Figure 1: ATMR structure with 5 nodes

with a fixed number of ATM cells continuously circulating around the channel from one node to another. Within each access node there is an access unit which performs both the physical layer convergence function and the access control function. Access to the ring is requested by the client and controlled by a combination of a window mechanism and a reset procedure. The client can issue a sending request to the access unit and receive a data cell. The window mechanism limits the number of cells a node can transmit at a time, called the "credits" of this node. The reset procedure reinitializes the window in all access units to a predefine credit value. The format of an ATMR cell is shown in Figure 2. It contains an access control field (ACF), which

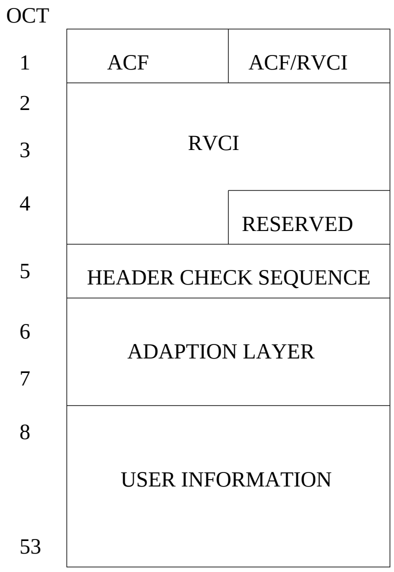

Figure 2: Format of an ATMR cell

includes a reset bit, a monitor bit and a busy address. When an access node releases an empty cell, it will fill its own address in the busy address field. The ATM cell is routed by using a ring virtual channel ID (RVCI) in the cell header.

The protocol entity of an access unit begins from an $IDLE$ state (see Figure 3). When the access unit has cells queued for transmission, it enters a $SEND$ state

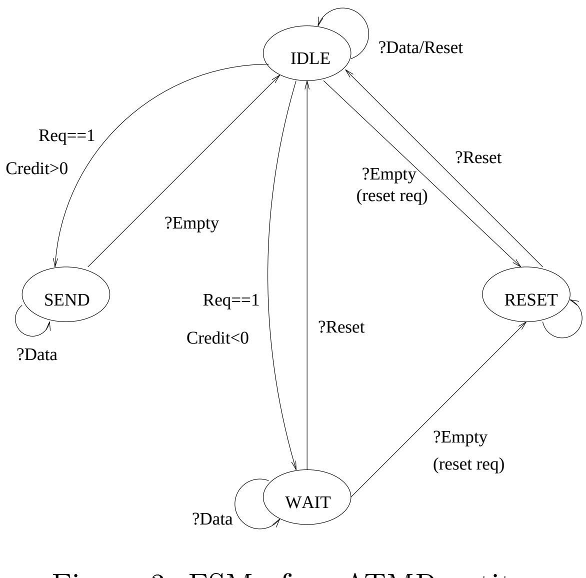

Figure 3: FSM of an ATMR entity

and sends them in empty slots received at the ring interface with the address of the destination in the RVCI field of the cell header. The RVCI field in the header of all cells received at the ring interface of each node is checked and, if the cell is addressed to this node, the cell contents are copied and passed to the appropriate convergence sublayer. The RVCI field is then set to zero, which indicates an empty cell, and the cell is relayed to the next node in the ring. If a match is not found, then, this cell remains unchanged. Transmissions on the ring occur in cycles during which each access unit is allocated a fixed window size credit. This credit indicates the number of cells the access unit can transmit in this cycle before issuing or receiving a reset cell from the ring interface. A window credit counter is maintained by each access unit. Whenever this value is less than zero, the protocol entity enters a $WAIT$ state to wait for a new credit. This value is initialized to the window size credit each time the ring is reset, namely the protocol entity is in a $RESET$ state and the credit is decremented by one each time the access unit transmits a cell from its transmission queue. This mechanism is followed by all access units in the ring and hence eventually all units become inactive and the flow of cells around the ring ceases.

To reinitialize the transmission of the cells, an active access unit always overwrites its own address in the busy address field in the head of all cells passing the ring interface. This way, if an active access node receives a cell with its own address in the busy address field, it concludes that other nodes are inactive. Then after completely sending any remaining queued data from the higher layer, it creates a reset cell by setting the reset bit in the header of the next cell passing the ring interface. The reset cell circulates around the ring and causes all other access units to reinitialize their window credit counters. Once reinitialized, any access unit having data queued for transmission regains the active state and restarts sending cells.

3 Modeling of the Asynchronous ATMR Protocol

In this section, we will illustrate the modeling approaches we adopted to perform the verification of the asynchronous ATMR protocol.

Since a given property is often true under certain environments, we need to build an environment of the component to make the verification model a closed system. As an ATMR protocol can have n nodes and p channels [4], We restrict our verification to the model shown in Figure 1 including five ATMR nodes and a channel size of ten cells. The channel length between two neighboring nodes is two cells. We realized through experimentation that this is the maximum model size that can be actually verified within the capability of VIS and the memory available in the machine we used (Sun Ultra 2 with 768 MB memory).

3.1 Model Reduction

Due to state space explosion, we did not succeed in verifying the whole data path of the ATMR model. In order to simplify this latter, we abstract away all the information which will not affect the behavior of the ring accessing scheme, namely the HCS field, the adaption layer field and the user information field. The reduced cell format on which we based our verification is shown in Figure 4, where 5 bits ACF and 3 bits RVCI will be used in our verification (non-shaded boxes in Figure 4). Because we kept all the access control information in the head format, namely the ACF and RVCI fields, the control behavior of ATMR with simplified cell format is exactly the same as that of the original one.

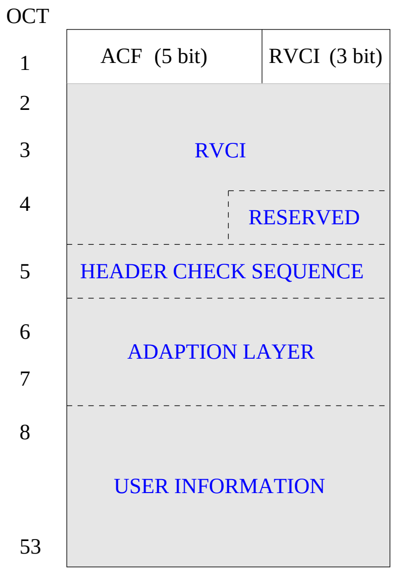

Figure 4: Simplified Cell Format

3.2 Simulation of the Asynchronous Model

Since VIS is built upon synchronous models, it is impossible to directly describe the original asynchronous ATMR in VIS, e.g., the description of the cell transmission between two access units using synchronous Verilog. We hence need to simulate the ATMR protocol in the synchronous VIS environment. There are many methods to simulate an asynchronous system in a synchronous environment [1]. Here, because we only request that cell transmission be asynchronous and the module itself be synchronous, we simply add a module *channel* in the Verilog specification. This channel model will play the role of a queue between two ATMR nodes (see Figure 5). All the cells sent or received by the access unit will hence be queued in the channel module. When the access unit wants to read a cell from the channel, it actually reads the cell from the head of the queue. If the destination is the current node, the cell will be processed in this access unit. Otherwise, the cell will be forwarded to the next node via the channel module. This way, the sending and the receiving processes within the ring can remain asynchronous.

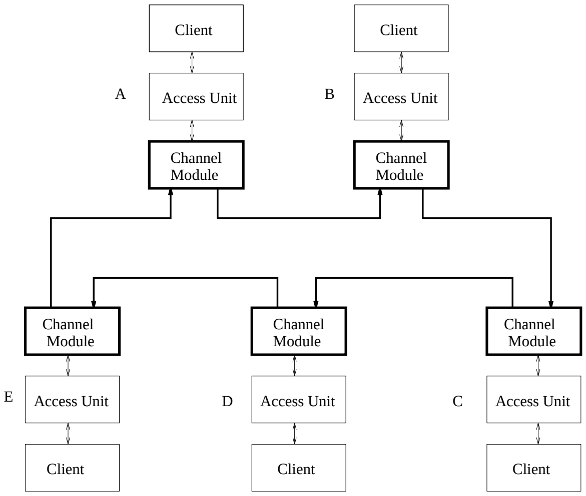

Figure 5: Modified ATMR ring structure

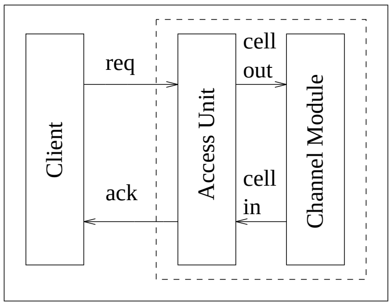

Figure 6: Modified ATMR node as a closed system

3.3 Environment Abstraction

Since it is often the case that a property is only true under certain conditions, we need to be able to make assumptions about the environment of the protocol during the verification process. In the ATMR case, we propose to verify the behavior of the access units and the channel modules assuming the client modules as environment (see Figure 6). In order to avoid state space explosion, we further construct an abstracted client environment module that characterizes just the client behavior which is visible to the access unit module via the interfaces connecting them. The abstracted client environment module hides internal details of the client like the generation of request signals.

In the ATMR model, we do expect that the client module will have an output signal *req* after it receives a positive acknowledgment signal *ack* from the access unit. The *req* may be 0 or 1 nondeterministically. We can implement this specification using the nondeterministic statement provided by the VIS tool. The FSM of the abstracted client is shown in Figure 7. "No_Req" is the initial state of the abstracted client. This state machine can nondeterministically set the *req* signal to 0 or 1 and receive the *ack* signal from the access unit.

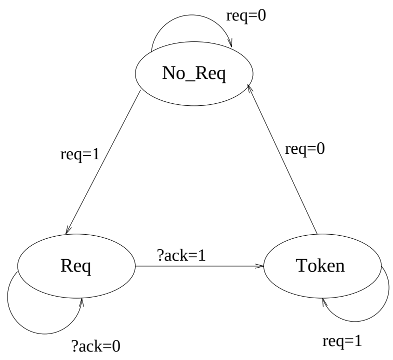

Figure 7: Abstracted client behavior

4 Model Checking of the ATMR Protocol

Once the abstract ATMR model is established, we validated it against a set of basic consistency properties. For illustration purposes, we present here four example CTL properties. In the following descriptions, "=", "→", " ∗" means "logical equality", "implication" and "logical and", respectively.

Property 1: Once an access unit exhausts its window size credits, the credits will eventually be renewed.

$$AG(((acc\_unit.crdt[2] = 1) * (acc\_unit.crdt[1] = 1) * (acc\_unit.crdt[0] = 0)) \rightarrow AF((acc\_unit.crdt[0] = 1)));$$

acc_unit.crdt stands for the number of credits which is being held by an access unit. *acc_unit.crdt* is composed of three bits: *acc_unit.crdt*[2], *acc_unit.crdt*[1] and *acc_unit.crdt*[0].

Property 2: A client's request will eventually be acknowledged.

$$AG((client\_req = 1) \rightarrow AF(acc\_unt\_ack = 1));$$

client_req is a cell sending request signal from a client to an access unit. If the requested cell has been sent out, the access unit will return an *acc_unitack* signal to the client.

Property 3: An access unit will eventually exit the RESET state and enter the SEND state (see Figure 4).

$$AG((acc\_unit.state = RESET) \rightarrow AF(acc\_unit.state = SEND));$$

acc_unit.state stands for the current state of an access unit.

Property 4: An access unit will eventually exhaust its window size credit.

$$AG(((acc\_unit.crdt[2] = 1) * (acc\_unit.crdt[1] = 1) * (acc\_unit.crdt[0] = 0)) \rightarrow$$
$$AF((acc\_unit.crdt[2] = 0) * (acc\_unit.crdt[1] = 0) * (acc\_unit.crdt[0] = 0)));$$

In this property, we expect that all the credits will be consumed during the sending procedure.

The verification of these properties was performed on a 296MHz Sun Ultra-2 workstation with 768MB of memory. (All subsequent experimental results reported in this paper were performed on the same machine.) The result of the of model checking of above properties are summarized in Table 1, including CPU time in seconds, memory usage in MB and the number of BDD nodes generated by VIS. The growth of BDD size is shown in Figure 8.

Property	CPU Time (s)	Memory (MB)	BDD Nodes
P1	49,616.9	623	1,376,130,560
P2	509.1	28	4,423,598
P3	27,400.5	371	1,928,041,758
P4	44,093.6	579	1,193,682,251

Table 1: Verification of the asynchronous ATMR

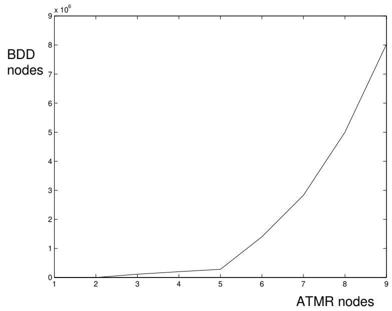

Figure 8: BDD size vs. number of ring nodes

Generally, the verified 5-node model has 220 latches, which is beyond the capability of VIS. See Figure 8 for the growth of the state space with respect to Property 2. However, the actual state space during the verification varies depending on different properties. For example, the BDD size of Property 2 is 4,423,598, while that of Property 3 is 1,928,041,758. Regarding Property 3, an ATMR node will enter the reset state only after all the remaining cell in the ring are sent out. It is supposed to be the most difficult property to be verified with respect to the BDD size. From Table 1, we can see that it takes 27,400.5 seconds and 371 MB memory to verify this property.

There are several reasons that the BDD size may blow up. First, the datapath in the ring will increase according to the number of nodes. Second, the variables are circular dependent due to the ring access. Third, the performance of the model checking itself is not steady in various applications.

As we see from these results, even though we used a reduced cell format and an abstract environment, the CPU time and memory usage are large, e.g., up to 20 hours of machine time and 623 MB memory for Property 1. In next section, we compare these results with the verification of a synchronous model of the ATMR and another verification of similar models done using SPIN.

5 Comparative Verifications of ATMR Protocol

For the purpose of comparison, we built a synchronous model of the ATMR protocol. This implementation is hypothetical and does not comply with the intended ATMR MAC protocol specification which is supposed to be asynchronous. Its structural model is similar to the asynchronous version, with five nodes in the ring and each node having a client and an access unit. In difference, however, no channel module is needed. The cell transmission between two neighboring nodes is achieved directly from the output/input port of one node to the input/output port of the next one. The communication is synchronized by the system clock. Besides this feature, the behavior of the access unit as well as the cell format are exactly the same as the asynchronous case. Given this synchronous model, we checked all the above correctness properties. The experimental results of model checking obtained with VIS are given in Table 2, including CPU time, memory usage and number of BDD nodes generated.

Property	CPU Time (s)	Memory (MB)	BDD Nodes
P1	5.6	12	614,833
P2	5.4	12	308,419
P3	5.7	12	921,727
P4	6.0	12	1,230,117

Table 2: Verification of the synchronous ATMR

Compared to the results of the asynchronous verification (Table 1), we see the tremendous cost of the simulation of the asynchronous model. This is mainly due to the fact that the simulated asynchronous model contains 220 latches, while the synchronous model has only 125 latches. The introduction of the channel modules creates a much larger state space. If the channel can be a built-in feature of the tool, like SPIN [5], the verification can be done in an easier way.

In [9], we used SPIN to verify this same protocol based on the same reduced model. The experiments were conducted on the same machine as before and are reported in Table 3, which includes CPU time in seconds, memory usage in MB and the number of states visited.

Property	Asynchronous			Synchronous		
	CPU Time	Memory	States	CPU Time	Memory	States
P1	6,033	360.9	49,485,700	40	172	427,995
P2	3,777	141.4	49,358,900	40	173	327,887
P3	–	-	–	46	208	426,044
P4	6,827	166.3	55,348,800	58	145	682,231

Table 3: ATMR verification with SPIN

SPIN uses an on-the-fly model cheking algorithm. It has advantages when verifying local properties. However, here the properties can be verified only after the whole behavior model has been constructed. Due to SPIN's explicit state counting nature, it is not strange that the verification of Property 3 could not be completed.

From Table 3, we first notice that the verification of the synchronous model is faster in VIS than in SPIN. This was expected as VIS targets synchronous concurrent processes. Second, SPIN runs in general faster for the asynchronous model verification than VIS. This is due to the fact that SPIN is based on an interleaving model, where each state update is a simpler operation restricted to one component only. Moreover, the interleaving semantics support a partial order reduction algorithm [5], which is not practical in hardware verification tools. In some cases, however, the efficient manipulation of BDDs and state encoding in VIS allows a larger state space to be explored as shown by the fact that unlike in VIS, the verification of Property 3 cannot be completed in SPIN due to an "out of memory" error.

Compared to protocol verification tools like SPIN, VIS cannot directly report deadlock, livelock and unreachable code. One has to explicitly express these properties using temporal formulas. For example, a dead-lock is expressed as: "Sender is not in send state and receiver is not in receiving state and there is at least one cell in the channel." Generally, this property is difficult to specify in CTL.

6 Conclusions

In this paper, we described the modeling and formal verification of an ATMR protocol using a hardware verification tool, VIS. Generally, when a higher layer protocol is implemented in hardware, it cannot be handled by a software (protocol) verification tool. The latter ones are based on an interleaving model and cannot reflect the concurrency aspects of a hardware implementation.

Most of the higher layer protocols are asynchronous protocols. VIS is a model checking tool targeting synchronous hardware systems. In this work, we showed how to simulate the asynchronous ATMR protocol in the synchronous VIS environment. This in turn created a larger state space, however. A verification attempt of the original model led to state space explosion. In order to complete the verification, we used a number of abstraction and reduction techniques based on datapath and nondeterministic environment abstraction.

The protocol verification in VIS is not directly scalable to large designs due to state space explosion. Thus, it is important to find techniques that can be used in conjunction with model checking tools like VIS to extend the size of the systems that can be verified. We are currently working on compositional approaches to verify large asynchronous protocol with existing model checking tools.

Acknowledgments

We would like to thank Ferhat Khendek at Concordia University, for initiation of this work and valuable suggestions. This work is partially funded by a Concordia graduate student scholarship and an NSERC research grant no. OGP0194302.

References

[1] R. Alur and T. Henzinger. Reactive modules. In *Proc. 11th IEEE Symposium on Logic in Computer Science*, pages 207–218. IEEE Computer Society Press, 1996.

[2] R. K. Brayton et al. VIS: A system for verification and synthesis. In T. Henzinger and R. Alur, editors, *Eigth Conference on Computer Aided Verification (CAV'96)*, pages 428–432. Springer-Verlag, Rutgers University, 1996. LNCS 1102.

[3] M. Charpentier and G. Padiou. Specification and verification of the ATMR protocol using UNITY. In *International Workshop on Formal Methods for Parallel Programming*, pages 26–36, 1997.

[4] F.Halsall. *Data Communication, Computer Networks and Open systems*, pages 618–623. Addision-wesley, 1996.

[5] G. J. Holzmann. *Design and validation of computer protocols.* Prentice hall, 1991.

[6] A. Joesang. Security protocol verification using spin. In *Proc. 5th SPIN workshop*, INRS-Télécommunications, Montréal, Quebec, October 1995.

[7] J. Lu, S. Tahar, D. Voicu, and X. Song. Model checking of a real ATM switch. In *Proc. IEEE International Conference on Computer Design (ICCD'98)*, pages 195–198, Austin, Texas, USA, October 1998.

[8] K. L. McMillan and J. Schwalbe. Formal verification of the encore gigamax cache consistency protocols. In *Proc. Int. Symp. on Shared Memory Multiprocessors*, pages 242–51, Tokyo, Japan, April 1991.

[9] H. Peng and F. Khendek. Formal verification of an atmr protocol. Technical report, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada, November 1997.

[10] S. Rajan, M. Fujita, K. Yuan, and M. Lee. High-level design and validation of atm switch. In *IEEE International High Level Design Validation and Test Workshop (HLDVT'97)*, Oakland, California, USA, November 1997.