Hardware Modeling and Verification of an ATM Ring MAC Protocol

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

Abstract— **In this paper, we describe the modeling and verification of the register transfer level (RTL) design of an ATM ring (ATMR) media access control (MAC) protocol using a hardware verification model checking tool, VIS. We succeeded verifying a synchronous as well as an asynchronous design alternative of this MAC. Throughout the verification, we report the performance of hardware protocol verification in model checking, and discuss some modeling techniques we adopted in the verification.**

I. Introduction

The VLSI design of finite-state concurrent hardware 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 [7] have established themselves as significant means for early detection of hardware design errors due to their ability of validation and conformance checking.

Generally, when a protocol is implemented in VLSI, it is difficult to be handled by a software (protocol) verification tool. The latter ones are based on an interleaving model and cannot reflect the synchronous concurrency aspects of a hardware implementation.

The aim of this paper is to describe the modeling and formal verification in VIS (Verification Interacting with Synthesis) [1] of the RTL hardware implementation of a single ATM ring (ATMR) MAC protocol [3]. A number of related work can be found in the open literature. These can be classified in three major categories: (1) formal verification of high layer software protocols [5]; (2) formal verification of synchronous hardware protocol [8]; and (3) formal verification of ATM hardware devices [6]. Our work distinguishes itself from these related publications by the fact that we verify the Verilog RTL hardware implementation of an asynchronous MAC. We present some techniques on how to simulate the asynchronous ATMR MAC design in a synchronous environment and also propose some abstraction and reduction approaches for the aimed verification. Furthermore, we analyze the performance of the source of the complexity in the verification with repsect to the CPU time, memory usage and state space.

The rest of the paper is structured as follows. The next section introduces the ATMR MAC to be verified. Section 3 discusses the modeling techniques we used for the asynchronous MAC. Section 4 describes the verification of the asynchronous and synchronous MAC hardware implementation against a set of CTL consistency properties. Section 5 finally concludes the paper.

II. A TMR Media Access Control Protocol

The ATMR MAC [3] 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

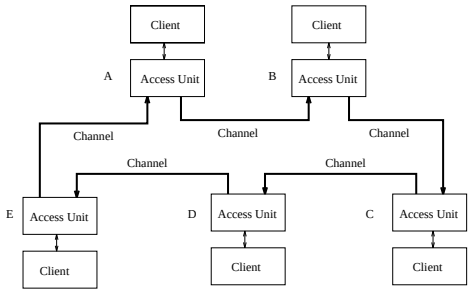

Fig. 1. A TMR structure with 5 nodes

to this type of shared medium, the ring is first initialized 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

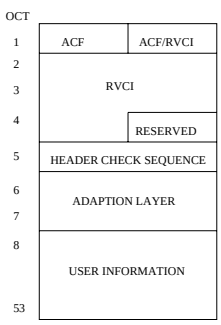

Fig. 2. Format of an ATMR cell

field (ACF), which 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.

This protocol was first checked by Chartier and Padiou [2] who used UNITY to conduct a pencil-and-paper verification. Their validation abstracts away from any implementations, be it in software or in hardware. In [9], the verification of a software implementation of this ATMR protocol using SPIN [4] is presented. Here, specific LTL properties as well as general behaviors such as deadlock and live-lock have been verified.

III. Establishment of the Verification Model

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 MAC can have n nodes and p channels [3], 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) [10].

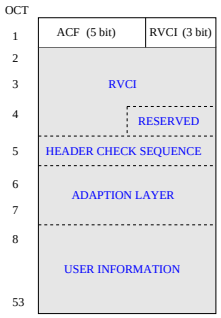

Fig. 3. Simplified cell format

A. Model Reduction

Due to state space explosion, we did not succeed in verifying the whole data path of the model. For a protocol verification, however, we can use the data independent model reduction technique as described in [11]. A data independent system is a system where the data path values do not affect the course of the computation. For example, a protocol whose only function is to move data from a sender to a receiver (without payload error checking, etc.) is typically data independent. The verification for such systems can be done using only the control structure; the data can be abstracted away entirely. In our case, since we want to check the ring access mechanism, 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, and this model reduction is sound. The reduced cell format on which we based our verification is shown in Figure 3, where 5 bits ACF and 3 bits RVCI will be used in our verification (non-shaded boxes in Figure 3). 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.

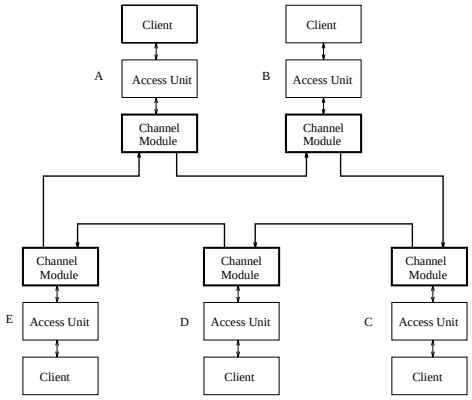

Fig. 4. Modified ATMR ring structure

B. 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 MAC in the synchronous VIS environment. 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 4). 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.

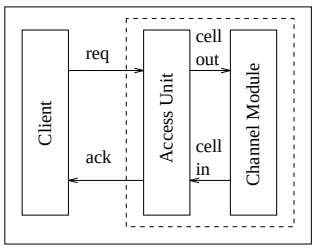

Fig. 5. Modified ATMR node as a closed system

C. 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 5). 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 6. "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.

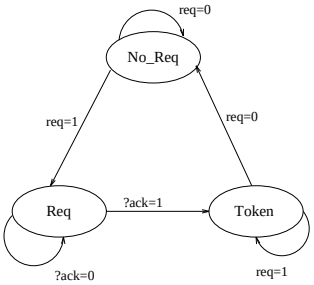

Fig. 6. Abstracted client behavior

IV. Model Checking of A TMR MAC

Once the abstract ATMR MAC Verilog RTL 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. "*AG*" and "*AF*" are temporal logic quantifiers meaning "always" and "eventually", respectively.

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

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

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 eventuallyexit the *reset* state and enter the *sending* state.

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

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

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

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

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 I, including CPU time in seconds, memory usage in MB and the number of BDD nodes generated by VIS.

Property	CPU Time	Memory	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 I
Verification results of the asynchronous ATMR

From Table I, we can see that the state space of the verification model is large due to the parallel composition of the system components. However, all properties could be checked thanks to the powerful BDD manufacture techniques in the tool. During the verification, we used the advanced ordering option [1] to reduce the BDD/MDD size.

The actual state space during the verification, however, varies depending on different properties. For instance, Figure 7 shows the growth of the state space with respect to the number of nodes for Property 2. 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 I, 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 data-

path in the ring will increase according to the number of nodes (Figure 7). 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.

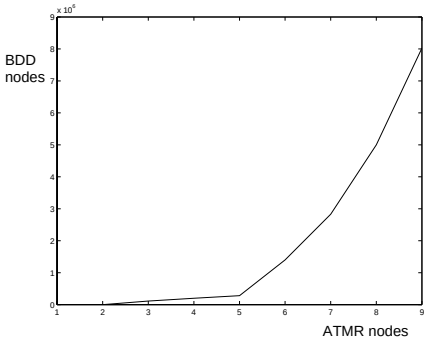

Fig. 7. BDD size vs. number of ring nodes

The verified 5-node model has 220 latches, which is beyond the capability of VIS. 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.

For the purpose of comparison, we built a synchronous model of the ATMR MAC. This implementation is hypothetical and does not comply with the intended ATMR MAC 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 II.

Compared to the results of the asynchronous verification (Table I), 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.

Property	CPU Time	Memory	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 II
VERIFICATION RESULTS OF THE SYNCHRONOUS ATMR

Compared to other protocol verification tools like SPIN, VIS cannot directly report deadlock, live-lock and unreachable code [9]. 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.

V. CONCLUSIONS

In this paper, we described the modeling and formal verification of the RTL hardware implementation of an ATMR MAC using a hardware verification tool, VIS. Most of the 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 MAC 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 data-path and nondeterministic environment abstraction.

ACKNOWLEDGMENTS

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

REFERENCES

[1] 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.

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

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

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

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

[6] 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.

[7] K. L. McMillan. *Symbolic model checking*. Kluwer, 1993.

[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] H. Peng and S. Tahar. Formal verification of an asynchronous MAC layer protocol in VIS. Technical report, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada, May 1999.

[11] P. Wolper. Expressing interesting properties of programs in propositional temporal logic. In *the 13th annual ACM symposium on principles of programming languages*, January 1986.