Hardware Simulation: A Flexible Approach to Verification and Performance Evaluation of Communication Protocols

Mario Baldi Enrico Macii Massimo Poncino

Politecnico di Torino
Dip. di Automatica e Informatica
Torino, ITALY 10129

Abstract

Communication protocols can be modeled as finite state machines, a formalism commonly used in digital circuit design. Sophisticated and efficient simulation techniques have been developed to help integrated circuit designers. In this paper, we propose an approach to the verification and performance evaluation of communication protocols and, in general, of entire computer networks based on such techniques. Preliminary results seem to indicate the feasibility of the method.

1 Introduction

Verification and performance evaluation are essential phases in the design process of communication protocols, and computer simulation is definitely the most used technique to perform these tasks. For this purpose, formal description languages, such as Lotos [1] and Estelle [2], and sophisticated simulation tools have been developed in the past. Because of the relative simplicity of the existing protocols, such tools were able to provide accurate analysis results in reasonable time. However, with the advent of modern hierarchical design techniques (e.g., the ISO/OSI layered model [3]), the global complexity of the protocols has sensibly increased, making the use of available simulation programs more difficult. Furthermore, existing simulators usually target specific protocol stacks. Therefore, efficient and flexible approaches need to be developed to deal with the complexity of today's protocols.

In essence, a communication protocol is a finite state system whose behavior may be conveniently described using a formalism, the finite state machine (FSM) model, widely employed in another field of computer engineering: The design of digital integrated circuits. Here, currently available simulation tools are able to efficiently handle extremely large systems. In this paper, we propose to apply techniques commonly employed in the design of digital circuits to the evaluation of communication protocols. Our approach is based on the following observation: Sequential circuits of interest are commonly modeled as finite state machines with billions of states. The specification of a communication protocol, on the other hand, is usually given as a network of interacting finite state machines whose components have very limited sizes (hundred states at most); as a consequence, a complete protocol stack can be modeled by properly interconnecting FSM networks representing protocols of different levels. It follows that verification and simulation tools developed for digital circuit manipulation could be successfully applied to the evaluation of protocols and, more in general, of complete computer networks whose behavior is specified by the interconnection of several FSM networks, each of which modeling a separate protocol stack. The only drawback of the approach above is that circuit analysis tools usually work on an implementation of the design, i.e., the topological description in terms of gates and/or transistors. On the other hand, communication protocols are, in general, realized as software procedures, and no hardware implementation of their specifications is available. To overcome this problem, we propose to use logic synthesis techniques to produce a gate-level (or switch-level) description of the communication protocol under evaluation starting from its FSM specification. Once this description is available, several identical instances can be created and properly interconnected to form a hardware model of a complete communication network ready to be simulated. Besides the possibility of employing very efficient hardware simulation tools, an interesting feature of the approach we are proposing is that, once the basic circuitry modeling the interconnection of several TX/RX units (i.e., the physical layer) has been synthesized, analyzing protocols at different OSI levels becomes extremely simple and inexpensive. In fact, it is sufficient to synthesize the protocol entity of each OSI layer, and then connect the obtained circuit to the current I/O interface of the circuit modeling the protocol at the physical layer. Following this paradigm, it is then possible to evaluate the relative performance of protocols at different levels of the OSI reference model, and to compare the operation of a protocol entity over different lower layer protocol entities. These two tasks are not easy to accomplish with traditional protocol analysis and simulation methodologies.

2 Protocol Modeling

As mentioned in the previous section, because of their finite state nature, communication protocols can be specified through networks of symbolic finite state machines. Each component of such networks usually has a very limited state space; therefore, common high-level and logic synthesis algorithms can be applied in order to construct a sequential circuit modeling exactly the behavior of the protocol. In general, digital systems consist of two distinct parts: The control logic and the data processing elements. For the realization of the control circuitry, the use of ad-hoc design techniques is required. Modern logic synthesis tools provide integrated frameworks for the full development of a gate-level design, starting from the state machine specification. The one we have used is SIS, the sequential synthesis system developed at UC Berkeley [4]. Concerning the implementation of the data paths belonging to the protocol (i.e., counters, arithmetic circuits, etc.), on the other hand, the use of high-level synthesis tools is preferable. This is because quite rich libraries of complex modules are commonly available, providing the synthesis tool with a large number of opportunities to generate reasonably well-designed circuits at very low cost. Once the architectural choices have been made, the final step of the modeling process consists of connecting the control logic to the data paths to get a complete description of the circuit implementing the protocol. To conduct our preliminary experiments, we have chosen Mozart [5] as the hardware description language to specify the data portion of the protocols. The reason for our choice is that Mozart gives us an easy way to interconnect gate-level descriptions (i.e., the control logic) to register-transfer-level descriptions (i.e., the data paths); furthermore, Mozart provides us with a sophisticated behavioral and logic simulator which can be used for performance analysis.

3 Network Modeling

Once a hardware description of each layer of the protocol stack under evaluation is available, the circuit modeling a full communication network can be constructed by appropriately interconnecting several instances of the protocol stack. At first glance, this operation may seem particularly trivial; however, several issues arise while building the connections between the physical layer entities (i.e., the equivalent of wiring the network). The main one has to do with delay modeling. Except for very special kinds of devices (for example, the ones made using sub-micron technology), in common VLSI circuits wire delays are much smaller than gate delays; therefore, they can be ignored (at least partially) in the definition of the circuit model without affecting the final simulation results. In the case of communication networks, wire delays definitely play a fundamental role. Assuming zero wire delay while interconnecting two or more TX/RX units may result in an unacceptable loss of accuracy in the model. For this reason, appropriate delay blocks need to be inserted into the circuit representing the complete communication network in order to obtain a realistic network model.

4 Simulation

Using circuit simulation, we are able to investigate protocol and communication network performance at different levels of abstraction. First of all, the simulation of the hardware implementation of a protocol can be exploited to study the features and the behavior of the protocol. Given a circuit that models a protocol layer entity, the service primitives provided to the upper layers are represented by the inputs of the circuit, while the responses and indications provided by the protocol are represented by the outputs of the circuit. In a similar way, the service primitives invoked by the protocol to the lower layers are modeled as outputs, while the responses and indications sent by the lower layer are modeled as inputs of the circuit. The simulation of the protocol consists of exciting the inputs with patterns that model legal requests (responses and indications) issued by the upper (lower) layer, and tracing the outputs that model the corresponding requests (responses and indications) issued to the lower (upper) layer. If the results of the simulation match the results expected according to the informal description of the protocol behavior, the correctness of the protocol specification (and its implementation) is proved. In a similar fashion, the interaction of peer protocol layers can be simulated to assess the performance of the protocol in many different situations. According to the OSI model, peer protocols (except for the ones modeling the physical layer) do not interact directly, but by using the services of the lower layer protocol they rely upon. The inter-operation of two neighboring protocol layers, in terms of the interaction of their hardware implementations, is simply obtained by properly connecting the pins of the circuits, according to the services or indications modeled by the pins. Once a protocol stack is implemented as a set of interconnected circuits, several instances of these circuits are created and connected to form the model of an entire computer network. The inputs and outputs of the overall circuit represent the services provided to and the indications issued by the top layer protocol of each of the stacks. The performance of the protocol stack under study can be assessed under various network load conditions by simulating the overall circuit. The inputs of the circuit modeling the computer network are excited with patterns. These patterns represent the service requests to the top layer protocol on each machine and produce the traffic that loads the network. The trace of the outputs shows the indications of data received by the user of the top protocol layer. This trace allows one to measure the throughput of the computer network, the delays experienced by the data, their variations, and the amount of data lost during the transmission. The proposed approach has high modularity and scalability, and can be easily extended to several protocol stacks. More important, hardware implementations of protocols belonging to interacting layers can be interconnected with little effort. Different coupling of neighboring protocols can be assessed and similar protocol stacks compared through simulation. Finally, complex computer networks which exploit various protocol suites can be easily modeled and studied by exploiting hardware simulation tools. In fact, tracing some connections inside the circuit representing a network allows us to identify and monitor congestion points in the communication network.

5 A Case Study: IEEE 802.3

Figure 1 shows the state diagram and the hardware implementation of the IEEE 802.3 protocol [6]. The controller coordinates three counters which are responsible for managing the inter-packet gap (`CNTg`), the length of the jamming sequence (`CNTj`), and the back-off algorithm (`CNTb`). Retransmissions are controlled by the shift register `SHRn`; from the content of `SHRn` is derived the number of time units to be used in the back-off algorithm. Counter `CNTt` is used to count the number of bits that have already been

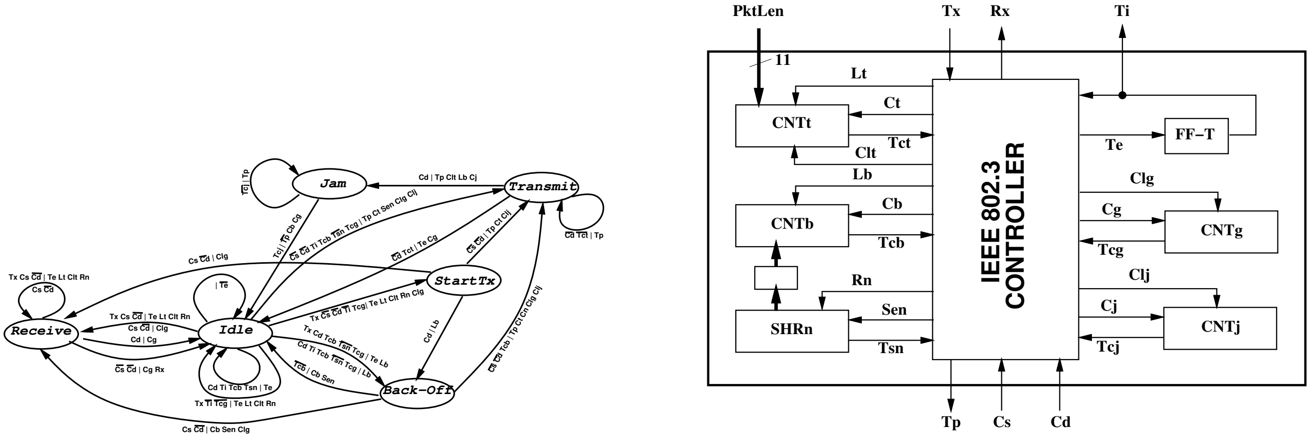

Figure 1: State Diagram and Hardware Implementation of the IEEE 802.3 Protocol.

transmitted from a packet, while flip-flop `FF-T` is used to store the information of whether the node is in transmitting mode or not.

When the user of the layer wants to send a packet, it should provide the binary code of the packet length on inputs `PktLen` and raise the `Tx` signal. If nobody is transmitting on the bus (both `Cd` and `Cs` are low), outputs `Tp` and `Ti` are raised to model the transmission and to indicate that the layer is transmitting. Output `Rx` indicates to the upper layer the reception of a packet.

Hardware implementation of IEEE 802.3 allows the protocol specification to be validated by performing reachability analysis on its FSM. Moreover, we can connect instances of the circuit that implement the protocol so that they interact as IEEE 802.3 protocol layers would do in a computer network; as an example, Figure 2 depicts a network with three IEEE 802.3 nodes. The blocks denoted with "Δ" represent delay elements that are used to model the propagation delay between two stations. We have assumed a star connection, so that the delays are assumed to be the same for each station pair.

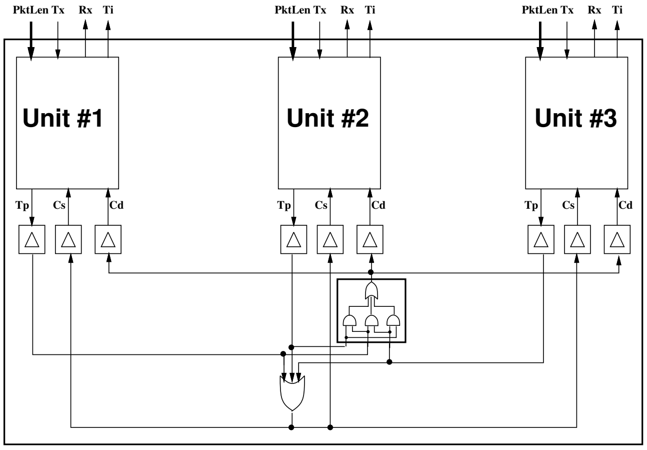

Figure 2: Circuit Modeling an IEEE 802.3 Communication Network.

We choose input patterns to model the load imposed by each user to the network, we apply them to the inputs (of the overall circuit), and we observe the outputs signalling the data delivery. This enables us to collect statistics about the network throughput, the delay experienced by data frames, and the variation of such a delay. By modifying the input patterns, we can collect statistics under various traffic scenarios and discover protocol limitations.

6 Experimental Data

The experimental results, though preliminary, have shown the feasibility of our approach. As already mentioned in the previous section, we simulated service requests by applying patterns to the circuit modeling the network. The input patterns are collected in a waveform file which is fed to the simulator. Each pattern is tagged with its corresponding application times; this allows to set the desired rate at which each station requires a transmission.

The experiments consisted of building two networks with different numbers of stations that try to transmit at increasing rates in the range from 0.5Mb/s to 4Mb/s. The transmission rate is the same for each station.

Figures 3 and 4 show the results for the cases of 3 and 5 stations, respectively. As expected, we have observed a progressive degradation of the network throughput; while for the lowest speeds (i.e., around 1Mb/s) the network throughput equals the overall requested transmission rate, for higher speeds the throughput degrades and tends to saturate to 6Mb/s. Clearly, increasing the number of stations causes an earlier saturation of the throughput.

The simulations showed that there are cases in which increasing the transmission rate of the stations results in a lower overall throughput (e.g., the point for 3.5Mb/s in Figure 3). This case occurs when two or more stations try to transmit at the same time thus generating collisions on the bus; this situation is more likely to happen as the request rate increases.

The CPU times of the simulations are independent of the network load, and are 1720 seconds and 3380 seconds (of a VAX 7000/610) for the cases of 3 and 5 stations, respectively. The results are referred to simulations of approximately 1 second of network operation.

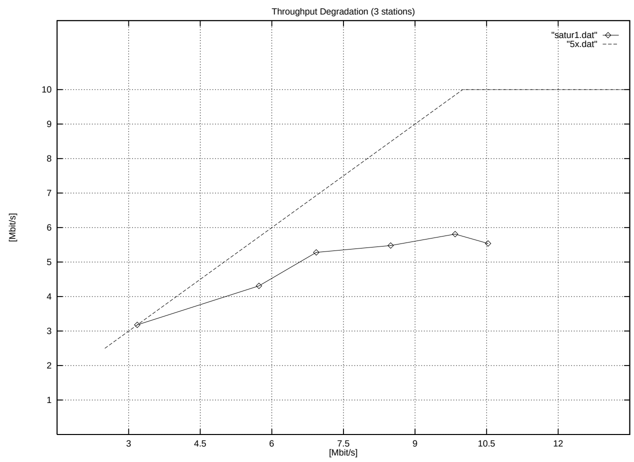

Figure 3: Performance Degradation with Increasing Transmission Speed (3 Stations).

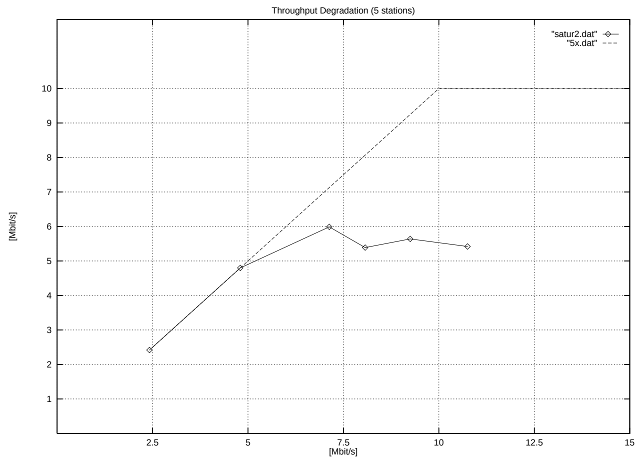

Figure 4: Performance Degradation with Increasing Transmission Speed (5 Stations).

7 Future Applications

With the increase, in size, of modern computer networks, evaluating the performance of heterogeneous networks , that is, networks whose nodes do not run the same protocol suite (see Figure 5), has become of primary importance. Results concerning verification and performance evaluation of communication protocols based on hardware simulation, though preliminary, seem to suggest the use of the techniques outlined in this paper to accomplish this task. To do so, we need to increase the expressiveness and the flexibility of the language we use to describe the hardware realization of each protocol layer and, in general, of protocol stacks and complete networks. Furthermore, we need to increase the degree of optimization in producing protocol hardware models and in carrying out the simulation. This is the reason why we are planning to replace the Mozart hardware description language and simulator with a VHDL-based design tool [7], where both the modeling and simulation activities are part of a self-contained support environment.

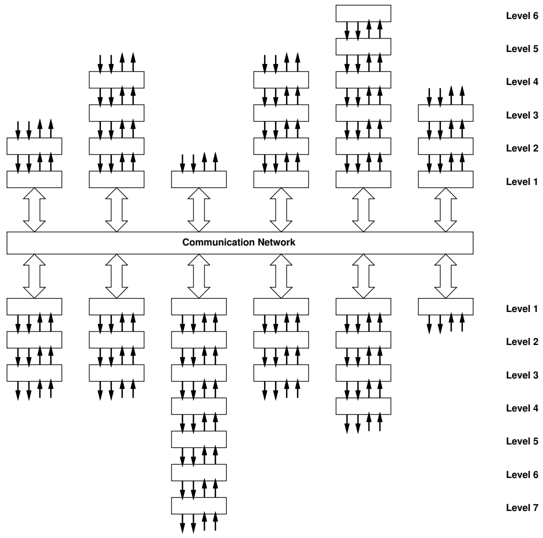

Figure 5: An Heterogeneous Communication Network.

References

[1] ISO, *IS 8807: Information Processing Systems, Open Systems Interconnection, LOTOS: A Formal Description Technique Based on the Temporal Ordering of Observational Behavior*, International Standards Organization, Geneva, Switzerland, 1987.

[2] ISO, *IS 9074: Information Processing Systems, Open Systems Interconnection, Estelle: A Formal Description Technique Based on an Extended Finite State Transition Model*, International Standards Organization, Geneva, Switzerland, 1989.

[3] ISO, *IS 7498: Information Processing Systems, Open Systems Interconnection, Basic Reference Model*, International Standards Organization, Geneva, Switzerland, 1984.

[4] E. M. Sentovich, K. J. Singh, C. W. Moon, H. Savoj, R. K. Brayton, A. Sangiovanni-Vincentelli, "Sequential Circuits Design Using Synthesis and Optimization," *ICCD-92: IEEE International Conference on Computer Design*, pp. 328-333, Cambridge, MA, October 1992.

[5] S. Gai, P. L. Montessoro, F. Somenzi, "MOZART: a Concurrent Multi-Level Simulator," *IEEE Transactions on Computer Aided Design*, Vol. CAD-7, No. 9, pp. 1005-1016, September 1988.

[6] IEEE, *Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method*, IEEE Standard 802.3, 1985.

[7] IEEE, *The VHDL Language Reference Manual*, IEEE Standard 1076-1987, 1988.