VHDL Simulation: A Flexible Approach to Verification and Performance Analysis of Communication Protocols

Mario Baldi Alberto Macii Enrico Macii Massimo Poncino

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

Keywords

VHDL, Gate-Level Simulation, Communication Protocol

Contact Person

Enrico Macii

Politecnico di Torino
Dip. di Automatica e Informatica
Corso Duca degli Abruzzi 24
Torino, ITALY 10129

Phone: +39-11-564.7074
Fax: +39-11-564.7099
E-mail: enrico@athena.polito.it

1 Introduction

Verification and performance evaluation are key steps in the process of designing a communication protocol. Usually, these tasks are accomplished by way of computer simulation: A formal description of the protocol is first constructed starting from the functional specification or from an existing implementation; such description is then fed to a simulator together with a trace of typical input stimuli (i.e., input conditions that emulate the behavior of the external world) with the purpose of verifying the correctness of the specification (*spec verification*) or the implementation (*impl verification*). By properly defining the set of input vectors provided to the simulator (for example, to model different load and traffic conditions), some performance analysis can be carried out as well.

Formal description languages that enable an easy modeling of the behavior of a communication protocol (e.g., Lotos [1], Estelle [2]) and sophisticated simulation tools that are able to process descriptions of this kind have been developed in the recent 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 the available protocol 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, that is widely employed in another field of electrical and computer engineering: The design of digital integrated circuits. Here, currently available simulation tools are able to efficiently handle extremely large systems; circuit analysis tools usually work on an implementation of the design, i.e., the topological description in terms of gates and/or transistors.

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 size (hundreds of states at most); as a consequence, a complete protocol stack can be modeled by properly interconnecting the FSM networks representing protocols of different levels. It follows that verification and simulation tools developed for digital circuit analysis could be successfully applied to the evaluation of protocols and, more in general, of complete communication networks whose behavior is specified by the interconnection of several FSM networks, each of which models a separate protocol stack.

The main 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 [4, 5] 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.

The remainder of this manuscript is organized as follows. Sections 2 and 3 briefly outline how a protocol and a complete communication network must be manipulated so as to obtain models which are suitable to be simulated using hardware simulation tools. Section 4 provides details on how the simulation of the models is actually carried out. In Section 5 we provide an example of modeling of a real communication network based on the IEEE 802.3 protocol, and in Section 6 we report some simulation results concerning the above example. Finally, Section 7 is devoted to concluding remarks.

2 Protocol Modeling

As already 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 for our experiments is

`DesignCompiler` by Synopsys.

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.

In the context of this work, we have chosen VHDL [6, 7] as the hardware description language, and `VSS` by Synopsys as the simulation framework. A fully *behavioral* description of the protocol under study is produced starting from the FSM specifying the protocol; being the VHDL description at a high level of abstraction, it is easy to focus on the aspects more strictly related to the protocol and neglect some issues concerning its implementation as a logic circuit.

3 Network Modeling

The VHDL implementation of the FSM describing a protocol is a circuit with inputs and outputs; in the context of a layered protocol stack, the inputs (outputs) are the service primitives and the associated parameters that the protocol provides to (is provided by) adjacent protocol layers.

Once a VHDL description of each layer of the protocol stack under evaluation is available, the circuit modeling a complete communication network can be constructed by properly interconnecting the circuits representing adjacent protocol layers through their inputs and outputs. Thus, a protocol stack is modeled by a single digital circuit having one interface (i.e., a set of input and output pins) towards the applications, and one towards the physical network. At first glance, this operation may seem particularly straight-forward; 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. Similarly, 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 do not interact directly, but by using the services of the lower layer protocol they rely upon. The inter-operation of two neighboring 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 communication network. The I/O signals 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 communication 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 network, the delays experienced by the data, their variations, and the amount of data lost during the transmission. The proposed approach is highly modular and scalable, 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 networks exploiting various protocol suites can be easily modeled and studied through hardware simulation tools. In fact, tracing some connections inside the circuit modeling the network enables the identification of congestion points in the communication system.

5 A Case Study: IEEE 802.3

Figure 1 shows the state diagram of the controller of the IEEE 802.3 protocol [8].

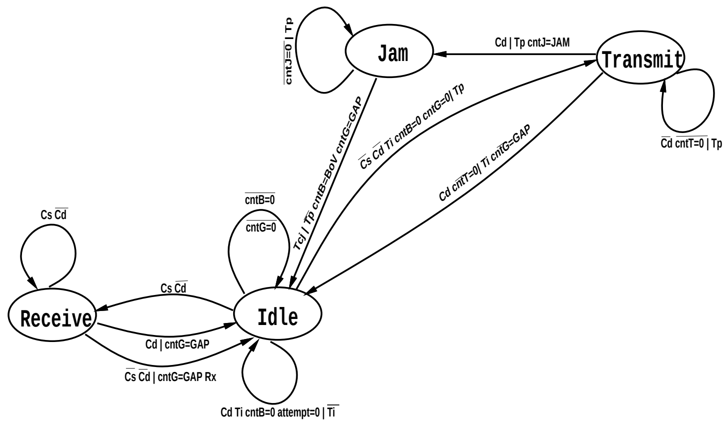
Jam
Transmit
Receive
Idle
$\overline{\text{cntJ=0}}$ | Tp
Cd | Tp cntJ=JAM
$\overline{\text{Cd}}$ $\overline{\text{cntT=0}}$ | Tp
Tcj | $\overline{\text{Tp}}$ cntB=BoV cntG=GAP
$\overline{\text{Cs}}$ $\overline{\text{Cd}}$ Ti cntB=0 cntG=0| Tp
Cd $\overline{\text{cntT=0}}$| Ti $\overline{\text{cntG=GAP}}$
$\overline{\text{cntB=0}}$
$\overline{\text{cntG=0}}$
Cs $\overline{\text{Cd}}$
Cs $\overline{\text{Cd}}$
Cd | cntG=GAP
$\overline{\text{Cs}}$ $\overline{\text{Cd}}$ | cntG=GAP Rx
Cd Ti cntB=0 attempt=0 | $\overline{\text{Ti}}$

Figure 1: State Diagram of the Controller of the IEEE 802.3 Protocol.

Starting from the state diagram, we have described in VHDL a circuit modeling the controller of the IEEE 802.3 protocol. An excerpt of such description is shown in Figure 2; in particular, we report the lines coding the state named `Idle`.

Such controller is used as building block of the complete IEEE 802.3 protocol, whose hardware implementation is shown in Figure 3.

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`; the number of time units to be used in the back-off algorithm is derived from the content of `SHRn`. Counter `CNTt` is used to count the number of bits that have already been 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.

We can connect instances of the circuit that implements the protocol so that they interact as IEEE 802.3 protocol entities would do in a computer network; as an example, Figure 4 depicts a network with three IEEE 802.3 nodes.

When the user of the layer wants to send a packet, it should provide the binary code of the packet length on inputs `PktLen`, the destination address on `DestAdd` and raise the `Tx` signal. If no other station 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, respectively. `AddrOut` and `AddrIn` are connected to a bus on which the address of the actual destination is present during each valid transmission. Output `Rx` indicates to the upper layer the reception of a packet.

```
ENTITY Station IS
  PORT (Tx:  IN BIT; PktLen:  IN INTEGER;
    Ti:  INOUT BIT;
    Tp:  OUT BIT; AddrOut:  OUT INTEGER;
    .....
  );
END Station;

ARCHITECTURE Stat_Behavior OF Station IS
  TYPE Status IS (Idle, Transmit, Receive, Jam);
BEGIN
  PROCESS
    ......
    BEGIN
    IF clock'event AND clock='1' THEN
      ......
      CASE Status IS
        WHEN Idle =>
          IF Ti = '1' AND Cs = '1' AND CntB = 0 AND CntG = 0 THEN
            -- trying to transmit when the bus is busy
            Attempt := Attempt - 1;
            cntB := ......
          END IF;
          IF Attempt = 0 THEN
            CntB := 0;
            Ti <= '0';
          END IF;
          IF Cs = '1' THEN
            Status := Receive;
          END IF;
          IF Ti = '1' AND Cs = '0' AND CntB = 0 AND CntG = 0 THEN
            CntT := PktLen;
            Tp <= '1';
            ......
            Status := Transmit;
          END IF;
        WHEN Transmit =>
          ......
      END CASE;
    END IF;
  END PROCESS;
END Stat_Behavior;
```

Figure 2: VHDL Description of the IEEE 802.3 Protocol.

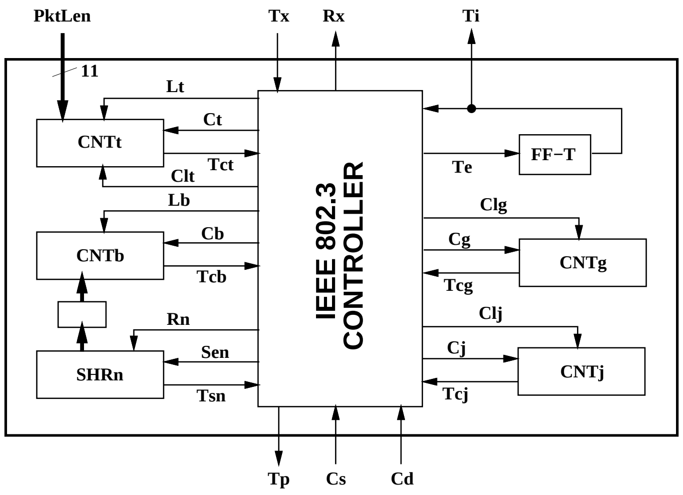

Figure 3: Hardware Implementation of the IEEE 802.3 Protocol.

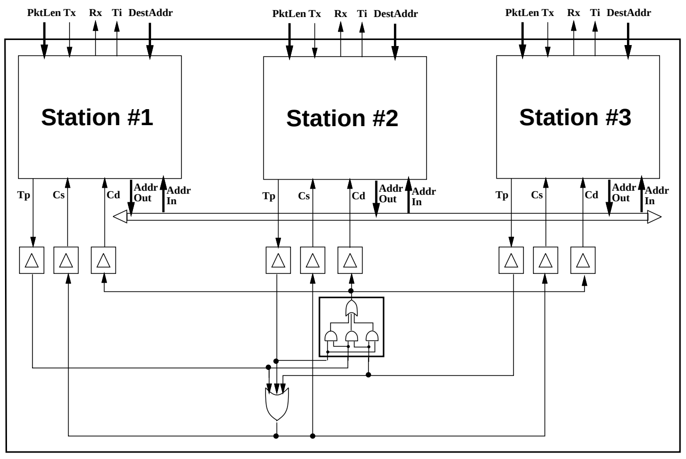

Figure 4: Circuit Modeling an IEEE 802.3 Communication Network.

The IEEE 802.3 bus itself is modeled as a separate VHDL entity; its interface (i.e., the entity declaration) obviously depends on the number of connected stations. The interface description for the example case of three stations depicted in Figure 4 is reported in Figure 5.

```
ENTITY bus8023 IS
  PORT (clock:  IN BIT; tp1,tp2,tp3:  IN BIT;
    aout1,aout2,aout3:  IN INTEGER;
    cs, cd:  OUT BIT;
    ain:  OUT INTEGER
  );
END bus8023;
```

Figure 5: VHDL Description of the IEEE 802.3 Bus Interface.

The buses corresponding to the `AddrOut` and `AddrIn` signals are modeled as integers, whereas the `clock` input is used only for synchronization purposes, and works as a discretization mechanism to be used in the simulation phase.

By simulating the behavior of the circuit that models the entire network, we can assess the performance of the protocol. Input patterns are chosen so as to model the load imposed by each user to the network, and applied to the inputs (of the overall circuit); the outputs signaling the data delivery are eventually observed. 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

As mentioned in Section 5, we have run several simulations to measure the throughput of networks with a different number of stations by varying the overall load imposed by the IEEE 802.3 users.

The experiments consist of building two networks with different number of stations that transmit at increasing rates in the range from 0.5Mb/s to 12Mb/s. The transmission rate is the same for each station.

The input waveforms are specified in a proper VHDL test bench file, which is connected to the overall system representing the network. Timing properties and constraints are specified in terms of units of the simulation time provided by the VHDL simulator.

Figures 6 and 7 show the results for the cases of 3 and 5 stations, respectively. As expected, there exists a progressive degradation of the network throughput when the speed of the stations increases; for higher speed, the throughput degrades and tends to saturate around 6Mb/s. Clearly, increasing the number of stations causes an earlier saturation of the throughput.

The CPU time of the simulations is independent of the network load, and is around 2500 and 4000 seconds (of a MS-DOS 486/DX-100 PC with 16MB of memory) for the cases of 3 and 5 stations, respectively. The results are referred to simulations of approximately 3 seconds of network operation.

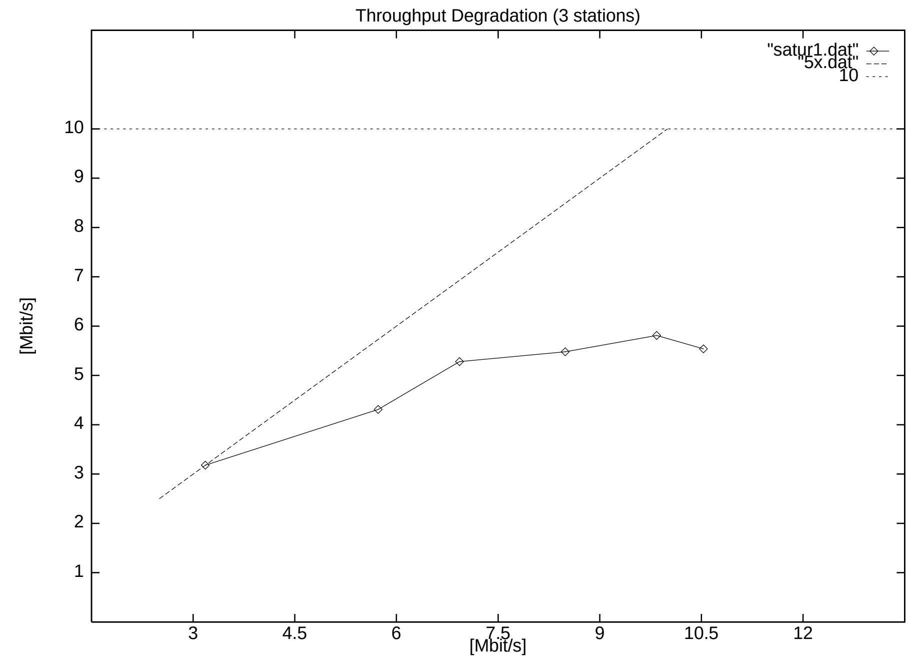

Figure 6: Performance Degradation with Increasing Transmission Speed (3 Stations).

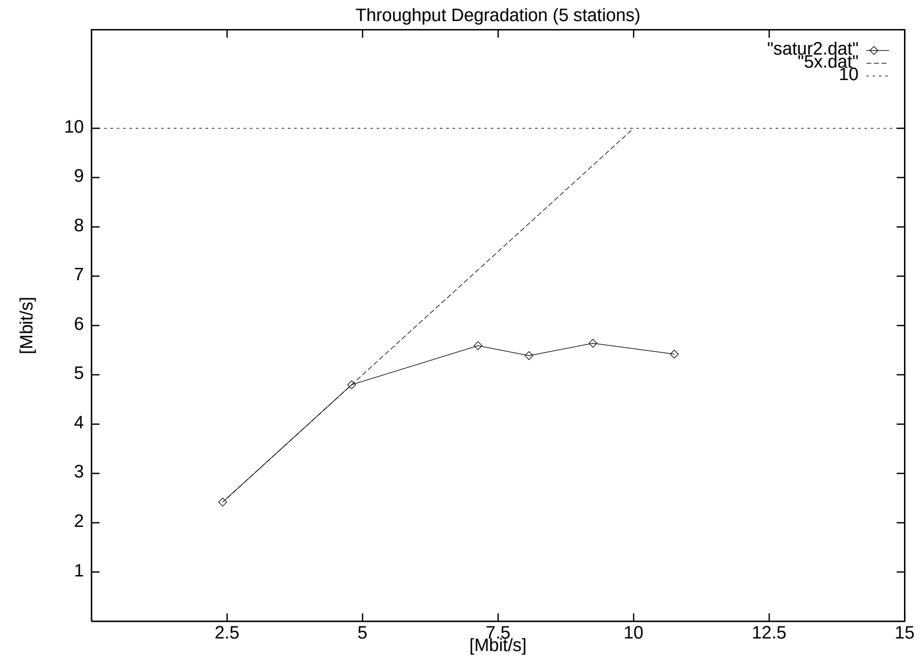

Figure 7: Performance Degradation with Increasing Transmission Speed (5 Stations).

7 Conclusions

With the increase, in size, of modern communication systems, evaluating the performance of heterogeneous networks, that is, networks whose nodes do not run the same protocol suite has become of primary importance.

In this paper we have proposed a method for verifying and evaluating the performance of communication protocols based on hardware simulation. Our approach has high modularity and scalability, and can be easily extended to the case of 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.

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] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.

[5] G. D. Hachtel, F. Somenzi, *Algorithms for Logic Synthesis and Verification*, Kluwer Academic Publishers, 1996.

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

[7] D. L. Perry, *VHDL*, McGraw-Hill, 1993.

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