A Web-CAD Methodology for IP-Core Analysis and Simulation

Alessandro Fin Franco Fummi
DST Informatica, Università di Verona
Verona, ITALY
fin,fummi@sci.univr.it

ABSTRACT

An effective selection of the more suited IP-core, available for a particular design, should be based on some simulation sessions. However, simulation models cannot be close enough to the real models of the core to protect the intellectual property. This paper proposes a Web-CAD methodology for IP-core analysis based on a client/server simulation architecture. The core vendor can make available to the public even the core models used for core synthesis without disclosing IP information. On the other side, the core user can simulate the remote core in the local simulation environment in the same way a local library component is simulated. To achieve this result, some problems concerning the non equivalence of the event driven semantic and the message driven semantic have been analyzed and solved.

1. INTRODUCTION

The use of intellectual property cores can sensibly reduce the development time of a design, thus allowing to satisfy the strategic goal of the *time-to-market*. However, if design complexity can be reduced by integrating already produced and optimized parts, design management becomes more and more critical, thus absorbing a considerable part of the entire design effort [1]. A critical aspect of design management concerns the identification of the more suited core to be integrated in the design. An extensive trade-off analysis between cost and performance should be performed to maximize the advantages of using cores.

The majority (perhaps the entirety) of the core vendors have sophisticated Web sites, which provide some information on the cores they are selling (e.g., pins, functionality, power dissipation, etc.). However, such an information is usually not enough when a core must be evaluated combined with the rest of the design and some simulation sessions should be performed. To complete this task, some simulatable views of the core should be available, possibly at the different abstraction levels (behavioral, RT, gate, switch). An hardware description language such as VHDL [2] is able to efficiently cover all such views, but the distribution of such descriptions cannot be safely performed if the intellectual property must be preserved. Cryptographic techniques for delivering simulation models have been proposed [3, 4, 5], but they do not seem to completely solve the problem, since they are extremely simulator dependent and they require a considerable effort for the core vendors to maintain the coherence of the different versions of the cores and to upgrade the delivered files.

This paper proposes to concentrate the source of simulations directly in the Web server of the core vendor, by proposing a typical client-server architecture, where the core users perform distributed simulations by connecting their simulation environment to the simulation environment of the core vendor. This idea has been already analyzed in the literature [7, 8, 9], by proposing the use of ad-hoc languages, such as Java, to model the core functionality and allowing cooperative working. The main disadvantage of such approaches is the need of a remodeling phase, by the core designer, which does not usually use Java-based tools to design cores. Even disregarding the necessary extra work, there is a high possibility of introducing discrepancies and differences between the Java models and the design models.

On the contrary, the proposed Web-based simulation methodology directly uses design languages, such as Verilog or VHDL, thus giving to the core provider the possibility of safely making available the real design models used to develop the core. In fact, the core user has only the view of the core interface (by using VHDL this is the `entity`) and the answers from the server simulator in relation to each input vector submitted. No internal model information is discovered, thus preserving the intellectual property. The simulation methodology is based on standard techniques and tools, that is, on a VHDL/Verilog simulator, on the Internet protocols and on a socket-based [11] interface. In this way:

- original models used to design the core can be made available on the core-vendor Web server, without writing *ad-hoc* simulation models;
- new versions of the core can be made available by the core vendor, by simply updating the server instead of upgrading delivered files;
- the core user simulates the core embedded in the architecture, by using the same simulation environment used for the rest of the design;
- different abstraction levels can be investigated by the core user, by simply changing the server socket port and without changing other aspects of the simulation environment.

The rest of the paper is organized as follows. The Web-based simulation methodology is presented in Section 2. Section 3 discusses all features and problems related to the distribution of a VHDL/Verilog simulation across the Internet. An application example is described in Section 4 to underline advantages and drawbacks of the proposed methodology. Finally, Section 5 is devoted to feature works and concluding remarks.

2. IP-CORE ANALYSIS METHODOLOGY

The proposed methodology for IP-Core analysis and simulation can be described by considering the point of view of the two cooperating parts: the core vendor and the core user.

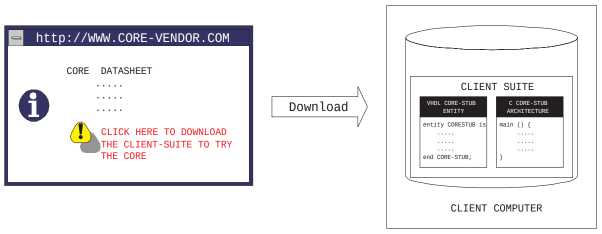


Figure 1: Client-suite download process.

2.1 Core Vendor

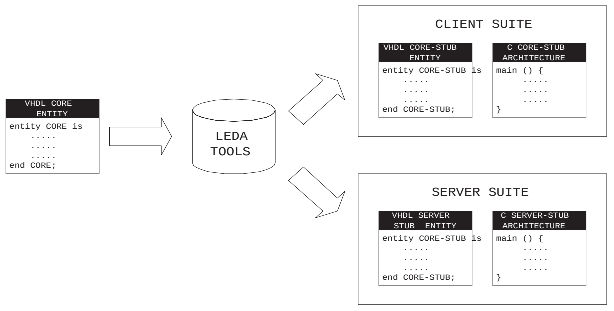
The task of the core vendor is the setup of a simulation server, which allows each core user to remotely simulate the cores without discovering their internal descriptions. We propose to make a simulation server for each core based on the following components:

- **The VHDL/Verilog simulator of the core.**
  It simulates the VHDL/Verilog core models that have been used to design the core. The simulator must be able to integrate VHDL/Verilog models with C procedures. This is a feature, for instance, of the Modeltechnology simulation environment [10].
- **The socket-based interface.**
  It accepts remote simulation sessions and transmits (receives) signal values to (from) the VHDL/Verilog simulator. This part is automatically built as described belove.
- **Models retrieval.**
  The Web server of the core vendor allows the core user to retrieve the simulation suite for each core (see Figure 1). This suite is composed of the core interface (e.g., a VHDL entity) and the socket interface (e.g., a C-language architecture).
- **Simulation server activation.**
  The simulation server can be always active waiting for connections on different ports, or the core user can explicitly ask the server to activate the particular simulator related to the core under analysis.

The socket-based interface is the unique component that must be built for each core. This operation can be automatically performed starting from the VHDL entity of the core (see Figure 2).

2.2 Core User

The core user must at first verify the general characteristics of the core, by consulting the textual information reported in the Web site of the core vendor. Whenever, this first analysis has been positively terminated, the core user must verify more accurately the core characteristics by running some simulation sessions.



```
entity CORE is
    .....
    .....
    .....
end CORE;
```

Figure 2: Socket-based interface generation.

Note that, the delivery of the *core stub* as a C source does not release IP information, since it is composed only of a set of procedures implementing the socket-based communication. The VHDL entity of the core is the unique IP information released, but this is performed whenever the set of input/output ports of the core is made available.

3. IP-CORE REMOTE SIMULATION

The remote simulation is guaranteed by the client/server architecture, which has been introduced in the previous section. All involved parts, from the client and the server side, are reported in Figure 3.

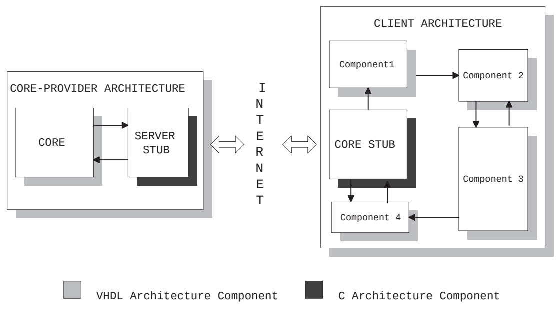


Figure 3: Client/server architecture for remote simulation.

This client/server architecture is based on two stubs, written in C, which allow the conversion of signal values into Internet packets and vice versa. This implementation requires the use of a VHDL simulator able to link C objects. At simulation time, the simulator links all compiled VHDL modules and all compiled C modules and simulate them concurrently.

Let us now analyze the main components of the client/server architecture.

3.1 Server stub

The VHDL entity of this component has an input port for each output port of the core and an output port for each input port of the core. This component is connected to the core design entity (see Figure 3), which is simulated on the server. The architecture of the core can describe it at different abstraction levels. The main task of this component is the communication with the *core stub* instantiated in the design of the core user. The communication is based on messages exchanged on the Internet. Each message includes the values of the signals to be assigned and some temporal information as described belove.

3.2 Core stub

This component has the same VHDL entity of the core and an architecture written in C. The core user includes the *core* *stub* in the project as it would be the real core. The *core stub* behaves exactly as the original core, since it communicates with the *server stub* via messages exchanged on the Internet. Every time there is an event on an input port of the *core stub*, the corresponding signal value is transmitted to the *server stub* and produces an event in the server simulator. The real core is accordingly simulated and if there is at least a modification of an output port of the real core, this value is sent back from the *server stub* to the *core stub*, which assigns the value to its corresponding output port. This assignment produces an event on the client simulator, which reacts in the correct way.

3.3 Concurrent processes

The architectures of both the *server stub* and *core stub* are composed of two concurrent processes: one for message sending and one for message receiving. Two processes are necessary since the two activities must be independent to guarantee the correct interface of server and client. In the *core stub*, the sending process is activated from the simulator every time an event happened on an input port of the core. This is guaranteed by the simulator since all input ports of the core have been inserted in the sensitivity list of this process.

The receiving process should be always active to monitor the socket interface, waiting for a message arriving from the *server stub*. The proposed solution is based on the insertion of the signal `selfin` in the sensitivity list of this process. This signal is connected to the signal `selfout`, which is asserted by the process itself by using the following instruction:

```
selfout <= NOT selfin AFTER WAKEUP ns;
```

This allows the client simulator to wake up the process every WAKEUP time unit of simulation. The same problem does not affect the two processes of the *server stub*. In fact, the sending process is activated by the server simulator every time an event happened on the output ports of the core.

3.4 Simulators synchronization

The synchronization of the client and server seems to be a non relevant problem, since it is solved by the socket. On the contrary, there are some hidden problems originating from the differences between the semantic of the VHDL simulator (event driven) and the semantic of the socket interface (message driven). In the proposed client/server architecture, at every VHDL event corresponds a message sending (for both the *core stub* and *server stub*), while the opposite is not always true (a message arrives to the *server stub*, but the core simulation does not produce an event on the core output ports). This is the reason of the addition of the `selfin` and `selfout` signals, which guarantee to always generate events on the client simulator even if a message is not sent by the server simulator.

Moreover, the simulation time of the server simulator and the client simulator must evolve coherently to guarantee the correct synchronization of both simulators. To achieve this result, exchanged messages are composed of the following fields:

- *signalID*. It identifies the signal transmitted by using a code, which has been defined during the automatic generation of the *server stub* and the *core stub*.
- *signal value*. Value of the transmitted signal.
- *time-stamp*. Simulation time in the server simulator, or in the client simulator, of the event of the transmitted signal.
- *type*. It differentiates the `INERTIAL` delay model (default) from the `TRANSPORT` delay model.

The receiving processes of both the *server stub* and the *core stub* are responsible of the correct synchronization of the two simulators. Every time a message is received, the *time-stamp* is compared to the *current-time* of the simulator and one of the following two actions is executed:

- If *current-time* $>$ *time-stamp* the simulation is aborted since there is no longer synchronization between the two simulators.
- If *current-time* $\leq$ *time-stamp* a signal assignment (`INERTIAL` or `TRANSPORT`) of *signal value* is performed on the signal identified by the *signalID* with a delay equal to *current-time - time-stamp* time units.

Let us analyze first case corresponding to an error. The activation and execution of the sending processes does not modify the simulation time of both server and client since these processes do not assign signals. On the contrary, the receiving processes allow the increasing of simulation time since they assign signals to a scheduled time. All processes consume real time to implement the transmission. Moreover, the receiving process of the *core stub* works in a polling way to check if messages are available on the socket interface. For this process the following temporal constraint has to be checked:

$$2T_t + T_{core} \leq T_{client}$$

where T_t is the real time used by the message to traverse the network and to reach the other side of the client/server architecture, T_{core} is the real time used by the core to eventually produce an event on the output ports of the core and T_{client} measures the real time between the sending of two messages of the client. This constraint certifies that the client simulator receives all messages from the server simulation in time for scheduling events in the client server. If this constraint is not satisfied, the correct relation between inputs and outputs would be lost and the values on the output lines of the *core stub* would be propagated with a wrong delay respect to the input values that have generated them.

The importance of the WAKEUP time derives from this problem. In fact, by accordingly setting this parameter it is possible to satisfy the above constraint. Reducing the WAKEUP time increases T_{client}, since more time is spent by the client simulator to activate the receiving process of the *core stub*. Therefore, it is possible to have a correct simulation even in presence of a slow connection. However, a low WAKEUP time increases the real time wasted by the polling cycle of the receiving process of the client simulator. To minimize the real simulation time, maintaining the correctness of it, it has to minimize the difference between $2T_t + T_{core}$ and T_{client}. The simulation starts with a low WAKEUP time and then it is dynamically increased during the simulation, in relation to the connection speed, but preserving the correctness of the simulation.

4. APPLICATION EXAMPLE

The proposed methodology for the analysis and remote simulation of IP cores is applied in this section to a a public domain load/store CPU available at the RT and logic levels [6]. The Modeltecnology VHDL simulation environment [10] has

Simulation Type	Real Time	User Time	System Time	NReal Time	NUser Time	NSystem Time
Local	21.2	8.9	1.1	1	1	1
Local with socket	1411.9	68.4	8.2	66.6	7.7	7.5
Intranet	1422.4	72.1	8.8	67.1	7.9	8.0
Internet	1518.0	77.8	10.8	71.6	8.7	9.8

Table 1: Simulation times and normalized times for the different kinds of simulation.

been used, since it allows the mixed simulation of VHDL and C modules.

The VHDL simulation of the overall architecture is performed to verify the effectiveness of the selected core and its correct integration in the design. We performed four kinds of simulation to measure the applicability of the proposed simulation methodology:

- **Local** simulation with the VHDL description of the core embedded into the global architecture. It corresponds to the simulation of a core, which is directly provided in VHDL.
- **Local with socket** simulation. Both client and server are running on the same machine and they are interfaced via socket. This simulation measure the overhead of the socket interface disregarding network problems.
- **Intranet** remote simulation. Both client and server belong to the same internet domain and they are connected trough a 100Mbit Ethernet connection. This simulation is related to the use and distribution of a core in the same company, where a design group can only use the results of another design group without disclosing the intellectual property.
- **Internet** remote simulation. This is the more general case, where a local client is connected to a remote server located in any part of the Internet. In this case, client and server have been located in different laboratories of north of Italy and the measurement has been performed in the rush hours. Due to the low bandwidth of the Internet connections in the north of Italy, this situation is not faraway from the localization, for instance, of a client in Europe and a server in north America, or vice versa.

A very detailed gate-level core description has been selected for the analysis. Table 3.4 shows the time necessary to run the simulation of the entire architecture with the embedded core. Times are expressed in seconds. In the last three columns there are the normalized simulation times with respect to the times of the local simulation.

The local simulation without socket interface is obviously the fastest solution, and the socket interface increases of 60-70 times the real time, while only of one order or magnitude the CPU time.

There is a small difference between the *Local with socket*, *Intranet* and *Internet* simulation methods, thus showing the real possibility of performing remote simulations across Internet. In fact, the remote simulation mechanism is limited by the conversion of simulation events into data packets and it is only partially limited by their transmission. Thus, a direct integration of this conversion mechanism into the simulation environment will probably reduce this overhead, thus making effective the remote simulation of a core.

5. CONCLUDING REMARKS

A Web-CAD methodology for IP-Core analysis and simulation has been presented in the paper. It allows the core vendor to make available very detailed core models without disclosing IP information. This is possible since the proposed client/server architecture allows the core user to integrate in the project a design entity, representing the core, which interfaces the client simulator to the server simulator. In this way, the core user simulates the core by using very detailed models in the same way the simulation of a local library component is performed.

The proposed client/server architecture is based on two VHDL (or Verilog) simulators (one working as server and one as client) and on the socket interface to connect and synchronize the two simulators. Problems concerning the differences between the event-driven semantic of the simulator and the message-driven semantic of the socket interface have been analyzed and solved, thus guaranteeing the correct remote simulation of the core in the local environment. Moreover, the efficiency of the proposed client/server architecture is acceptable as shown on an application example.

6. REFERENCES

[1] J. Notbauer, T. Albrecht, G. Niedrist and S. Rohringer. Verification and management of a multimillion-gate embedded core design. *Proc. ACM/IEEE DAC*, pages 425–428, 1999.

[2] IEEE standard VHDL language reference manual. *IEEE Sid 1076-1993, The Institute of Electrical and Electronic Enginnerings, Inc., New York, NY*, 1993.

[3] M.J. Silva and R.H. Katz. The case for design using the World Wide Web. *Proc. ACM/IEEE DAC*, pages 579–585, 1995.

[4] LEDA VHDL*Verilog System user's manual. *VHDL Compiler Version 4.1*, 1993.

[5] S. Hauck and S. Knoll. Data security for Web-based CAD. *Proc. ACM/IEEE DAC*, pages 788–793, 1998.

[6] *SPeedCHART Project Designer User's Manual*, Version 3.2.0 Speed S.A., 1996.

[7] R. HelaiHl and K. Olukotun. Java as a Specification Language for Hardware-Software Systems. *Proc. IEEE ICCAD*, pages 690–697, 1997.

[8] M. Dalpasso, A. Bogliolo and L. Benini. VSpecification and Validation of distributed IP-based designs with JavaCAD. *Proc. IEEE DATE*, pages 684–688, 1999.

[9] M. Dalpasso, A. Bogliolo and L. Benini. Virtual Simulation od Distributed IP-based Design. *Proc. ACM/IEEE DAC*, pages 50–55, 1999.

[10] ModelSim user's manual. *Model Technology*, 1998.

[11] L. Peterson and B. Davie. Computer Networks: A System Approach. *Morgan Kaufmann*, 1996.

[12] C. Patchett and M. Wright. CGI Cookbook. *John Wiley and Sons*, 1997.

[13] D.L. Perry. VHDL. *McGraw-Hill, Inc.*, 1990.