Protected IP-Core Test Generation

Alessandro Fin Franco Fummi

Dipartimento di Informatica
UniversitàÁdi Verona
ITALY

ABSTRACT

Design simplification is becoming necessary to respect the target time-to-market of SoCs, and this goal can be obtained by using predesigned IP-cores. However, their correct integration in a design implies more complex verification problems. IPs are usually provided with their own test patterns that can be used only by applying design for testability techniques onto the chip. Whenever physical faults must be detected, this approach is reasonable, even if it implies circuit performance degradation. However, it is completely useless at the design level, when the correct integration of the IPs into the global design must be investigated. At this level, proprietary test sequences must be generated in relation to the actual use of the IPs into the design. In this paper, the SystemC language is exploited to define a design verification framework for integration test of IP-cores. Intellectual properties of cores are guaranteed by adopting a client/server simulation architecture and by allowing functional test generation on faulty IP-core models without disclosing their internal structure. The methodology can be applied to mixed descriptions based on VHDL and SystemC, since an abstraction layer has been defined allowing clients and/or servers to be indifferently described in VHDL or SystemC. Finally, remote simulation can be also performed locally to avoid bandwidth bottleneck and the test generation process can be indifferently applied at lower abstraction levels such as RT and gate.

1. INTRODUCTION

The current trend of systems on silicon is leading to a complexity that can be reduced only by integrating already produced and optimized parts (IP-cores). A new design is usually developed by assembling *ex novo* IP-cores, previously designed IPs, externally provided IPs and some user-defined modules to connect all the included components.

The identification of the most suited core for a design is one of the most time consuming aspect of design management [1] and all core selection implications, i.e. functionalities, testability, performances, have to be evaluated before including it into the final design. Previously defined techniques [2, 6, 7, 9] present alternative methodologies for allowing a design team to evaluate different IPs integrations into the final design without violating the IPs of the core provider. The goal of the technique presented in this paper is to integrate these functionality evaluation methodologies with a testability evaluation framework. Providing this kind of framework, the core vendor allows more users to have free chance to evaluate the produced cores. The opportunity to evaluate the final testability of a design including the selected core can have high impact on the profitability of the developed design decreasing the testability issues, which can arise during the design phase. In market with so many competitors, such provided features can make the difference and that core vendor can increase his visibility and his business volumes.

The main issue for developing such a feature is how to provide a framework capable to evaluate the testability properties of the selected core and simultaneously to protect the core IP. The presented solution is based on a client-server architecture, which allows to test and to simulate a core without disclosing the core IP. This goal is obtained by adding to the core model, which can be either remotely or locally simulated, additional methods to query the core simulation server about testability information. This additional information can be useful to evaluate the testability of the selected core merged into the final design and/or to drive the automatic test pattern generator, which is going to be adopted to test the design. The core vendor has full control on the information released by the query methods by enabling/disabling each single query method, depending on the business relation with the customer. This can be achieved by a customized activation key file provided to the core user to activate the core client simulation. All such features can be reached by exploiting the new system-level description language: SystemC.

The rest of the paper is organized as follows. The methodology for distributing IP-cores is presented in Section 2. Section 3 discusses all features and problems related to the simulation of SystemC descriptions across the Internet. Section 4 shows the implementation of a remote test generator based on the query methods framework. An application example is described in Section 5, while Section 6 is devoted to future works and concluding remarks.

GLSVLSI'02, April 18-19, 2002, New York, New York, USA.

2. IP-CORE DISTRIBUTION

The core vendor and the core user are the two cooperating players of the proposed methodologies for IP-Core testing and distribution.

- The goal of the **core vendor** is to allow the core user to simulate and test the cores without disclosing their internal descriptions.
- The **core user** usually analyzes at first the general characteristics of the core reported in the Web site of the core vendor. Results of this rough analysis must then be refined by performing some simulation sessions, eventually remotely, to explore the effective integration of the core in the design. The increasing relevance and cost of the test phase in the design workflow requires to evaluate *a priori* also the core integration from the testability point of view.

Testing cores and protecting IPs are activities with opposite requirements. In fact, a deep internal knowledge of the core is required to efficiently test it and part of that knowledge is what the IP protection techniques hide to the core user. This paper proposes a technique to efficiently evaluate the integration of a core in the RTL design under development from the functional and testability points of view. These evaluations are provided without disclosing IP information, thus matching the requirements of both cooperating parts.

The proposed technique is abstraction level independent and it can be applied at each workflow design level, i.e. functional, behavioral, RT and gate. However, the most significant results can be obtained applying it at the RT level, where most of the core requirements are already defined (i.e. performance, timing) and they can be checked during the core selection process.

The prototype of the presented technique has been developed in SystemC: a C++ library for system design. Given that all implementations are in the C++ language and it has been very easy to integrate the core description with the socket library, required to allow a simulation base on a client-server architecture. Many VHDL and Verilog simulators provide APIs to communicate via socket with other processes, thus the presented testing framework allows to test and cosimulate cores developed in different HDL languages.

The core vendor can allow two different ways to simulate and test the cores by providing the following two different packages:

- **SystemC package for remote simulation.**
  It is composed of a core interface (a SystemC component declaration) and the socket interface (a SystemC component definition). The SystemC component definition is released as a C++ source file and the core interface as a C++ header file. SystemC designers can simply compile and link this suite to their design in order to remotely simulate the core. Moreover, VHDL designers can also use this suite to verify the integration of the core in their preliminary SystemC-based prototype. When the design will be represented in VHDL, the same suite or the VHDL-C suite will be used to refine the simulation.

- **SystemC suite for local simulation.**
  It is oriented to the same users and can be used for the same targets. It is composed of a core interface (a SystemC component declaration) and its implementation (a SystemC component definition). The SystemC component definition is released as a C++ object file, compiled for the different supported computer architectures, and the core interface as a C++ header file. An alternative for local simulation is releasing the executable file containing the core server simulator, which can be locally stimulated and tested by using a socket interface.

The core vendors can publish these packages on their web sites or provide them directly to the customers. The advantage of the package for the local simulation is the higher overall simulation speed for the core user, while the remote simulation package allows the core vendors to provide to their customers the functionalities of the newest version of the core. Moreover, the upgrade or the bug fix of the core is completely transparent to the core users by using the remote simulation package. The package for local simulation should be provided for mature cores, which do not require fixes at all, while for recently designed cores the package for remote simulation could be more effective. A remote simulation approach could be useful as web-CAD technique for cooperative designs with some IPs disclosure issues.

As part of the core specifications, the core vendor provides a list of the query methods available for the selected core. These methods are implemented via a specific type of packet exchanged by core client and server.

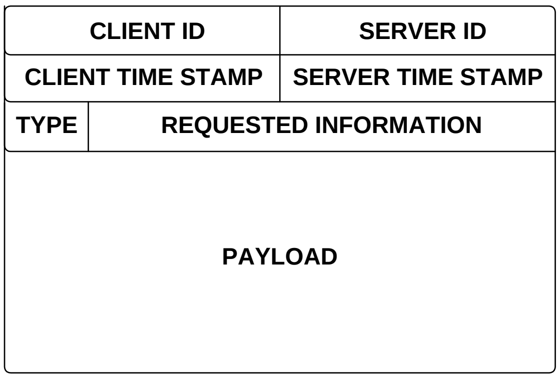

Figure 1: Packet structure.

Figure 1 shows the structure of a packet: six header fields and payload. ID client and ID server allow both client and server simulators to identify each other and check the source of the incoming packets. Unrecognized packets are dropped by the receiving peer. Time stamp fields allow to easily check if the simulations are synchronized. The time stamp contains the simulation time of the local simulation. If a received packet has a time stamp newer than the last one, then the simulation is aborted. The last two fields of the header determine if the packet is containing either simulation or testing query. The packet length is constant (1024 byte). This allows a faster packet elaboration at a cost of potential bandwidth waste.

3. IP-CORE SIMULATION

Remote and local simulation configurations have been investigated. The main characteristics are presented in the following subsections.

3.1 Remote Simulation

Remote testing and simulation requires a client-server architecture composed by two SystemC simulators connected via socket. This solution allows the core vendor to have the highest control on the distribuited cores. The vendor can define the access policy to the core servers. For the actual version of the software, a security technique based on password and IP address check has been developed. The security level can be increased applying further control access techniques, for example by defining an authentication procedure based on *one time passwords*. The core vendor can limit the number of the remote simulations for each core user. The remote simulation guarantees the core user to simulate and test always the last version of the core. In fact, keeping the same interface the core vendor can change the implementation of the offered functionalities without forcing the core user either to download a new package or to modify the previous design.

The most complex issue to address connecting two simulators is maintaining the temporal coherency of the two simulations. The client and the server simulations have both two independent threads for receiving and transmitting data packets. The temporal alignment of the two simulators is managed exploiting SystemC dynamic event generation, one of the new features introduced since the 1.2 version of the library. After sending a data packet, the simulation waits for an event, which will be notified when the receiving thread will receive the updated packet from the peer. For each sent packet, another packet will be received, even if the received packet does not contain any update signal. This solution, which increases the communication overhead, is required to guarantee the correctness of the remote simulation and its equivalence to the local simulation. The data packet have to be sent at the end of the elaboration phase, after all signals have reached a stable value. This is required to avoid simulation with no definitive signal values. In order to solve this problem, an additional internal bit signal is added to the sensitivity list of the sending thread.

This signal is inverted during the same clock cycle $wait_{max}$ times to allow the core input signals to stabilize. This guarantees to send only one data packet per clock cycle, containing the definitive signal values. The core user has to fix the value of $wait_{max}$: underestimating this parameter causes to send more than a data packet per clock cycle. This condition can be checked by both client and server comparing the simulation stamp of the last two either sent or received data packets. Overestimating this parameter implies the schedule of useless events, which slows down the simulation performance. The simulation is stopped if this condition is violated and the core user has to increase the $wait_{max}$ value to run again the simulation. The remote simulation allows a larger control on the IP simulation for the core vendor: core users, simulated cores, executed services, etc. There are less advantages for the core users, who have longer simulation time and an Internet connection is required during the testing procedure. Adopting this simulation configuration, core users do not have to check for new core simulators updates. They simply run the the remote simulation and the remote core simulation will provide the new services and functionalities.

3.2 Local Simulation

The core local simulation is achieved by incorporating the downloaded SystemC compiled core description into the design. The core user knows the core interface considering the header file coming with the core object file. Avoiding the communication of the remote client-server simulation, a faster overall simulation time can be achieved. The core vendor can alternatively allow the local simulation by providing the executable file of the core server. The server simulation can be run by the core user either on the same machine of the core client or on the intranet. This simulation configuration presents more advantages for the core user with respect of the core vendor. The simulation/testing time decreases significantly and the user testing procedures are shorter. The core vendor loses some of the control on the core simulation offered by the remote configuration. If this factor is critical for the core vendor, then one can provide resources for remote simulation only.

4. PROTECTED TEST GENERATION

Testing efficiently a design requires a deep knowledge of the design structure. This requirement can imply the violation of core IP. This problem can be avoided by defining the core public interface, composed of a set of methods, which can be executed by the TPG to collect information about the efficiency of the applied test sequences. The core vendor defines the set of these querying methods, thus the disclosure level of the information provided to the core user. The core vendor could apply a different level of disclosure for each core user by checking the client ID field of the packet header. The proposed framework for remote testing is sketched in Figure 3. An error free and an erroneous instance of the core are included on the server side of the proposed architecture. The error free is the RT core description which is provided to the core user when the core is bought. While the erroneous description is generated by the core vendor to simulate the presence of stack-at errors in the RT core description and to allow the core user to evaluate the impact of the testing procedure on the core under selection. The error presence is simulated adding extra C++ code into the SystemC design description to stuck-at either 0 or 1 the values of core internal signals and ports [4]. The adopted error coverage is the bit-coverage [10]. It allows to simulate, under the single error assumption:

- **Bit failures**. Each variable, signal or port is considered as a vector of bits. Each bit can be stuck-at zero or one.
- **Condition failures**. Each condition can be stuck-at true or stuck-at false, thus removing some execution paths in the erroneous representation.

The error model excludes explicitly the incorrect behavior of the elementary operators (e.g., +,-,*,...). Only single bit input or output errors are considered, therefore including all operator's equivalent errors.

It has been proved in [11] that this error model covers all statement, branch and condition errors, moreover it covers an important part of all path errors.

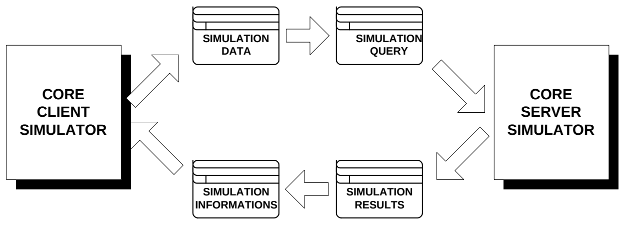

Figure 2: Packet traffic.

To activate the target error, an extra port is added to the original interface of the core for testing purpose only. This port allows to select the injected error. Each error is identified by an unique ID, but the correlation between ID and selected error is not known by the core user to further increase the IP undisclosure. To simulate a target error the core user has to include the error ID in the payload of the simulation packets. If no error ID is transmitted then no error will be simulated and the returned simulation data packet will contain error free results. The communication protocol has been extended to manage an extra protocol packet: the query packet. The simulation is executed by the core user, by sending these packets to query the server simulation about the effectiveness of the last data packet sent containing the last test pattern.

To experiment the presented technique we adopted a local genetic TPG. To generate test patterns it requires some information to evaluate the fitness function for each generated test sequence:

- **Local Observability.** The fitness function of a sequence is higher if the error can be propagated to the primary outputs of the core, which are connected to internal signals of the main design. In fact, this is a *condicio sine qua non* to detect the injected error on the primary outputs of the complete design. The internal signal values can be obtained via simulator interface for both SystemC and VHDL cores.
- **Hamming Distance.** Not all sequences can propagate the error to the core primary outputs. The fitness of such sequences is proportional to the Hamming distance between the internal signal values of the erroneous core and the error-free core (see Figure 3). By applying the genetic operators, such as mutation and crossover, the TPG tries to modify sequences with high Hamming distance in order to propagate the injected error to the core primary outputs.

The actual version of the remote ATPG defines two query packets, these packets can query the core server for the two information required by the genetic ATPG. The core server replies to these packets by sending back packets containing numeric values. The set of the available query packets can be enriched by the core vendor with packets allowing the core user to obtain more information about the core server simulation. The core vendor can make available a query packet to export each core server simulation information, which does not violate any core IP.

Figure 2 sketches the packet traffic generated by the core client and the core server for remotely testing and simulating the core.

The presented methodology can provide also a testability evaluation of the overall architecture. Developers are able to consider the impact of each alternative IP-core on the architecture under development from the functional and the testability points of view. Due to the large amount of resources required by the testing phases of a new design, an early testability evaluation of the final design can imply a resource saving and a better product quality.

5. APPLICATION EXAMPLE

The proposed methodology for the verification and integration test of IP cores is applied in this section to an example. SystemC 1.2 has been used for modeling and simulating the SystemC descriptions. All experiments have been performed on a SUN Ultra5 333 MHz with 256MByte Ram.

5.1 Core Selection

We consider the problem to implement a 3-tap digital filter characterized by the following equation:

$$y = c_0 z + c_1 z^{-1} + c_2 z^{-2}$$

The hardware realization of the filter uses, as embedded core, a public domain load/store CPU available at the RT and logic levels deeply integrated in an embedded design composed of some other components as a memory and the MMU. The SystemC definition of the core is provided on the Web site of the core vendor with the *core stub* and the corresponding implementation . The SystemC simulation of the core is performed to verify the effectiveness of the selected core. Its correct integration in the design is checked by generating functional test patterns as reported in the next paragraph.

The example reproduces the core selection process of a design team, which has to select the more suitable CPU core to complete the designed embedded system to evaluate the digital filter. This selection process will end as soon as a CPU core is found compliant with the required functional and testability features. Four types of simulation have been performed to measure the applicability of the proposed simulation methodology:

- **Local** simulation of the core embedded into the global architecture. It corresponds to the simulation of a core, which is directly provided in SystemC (VHDL) source code.
- **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.

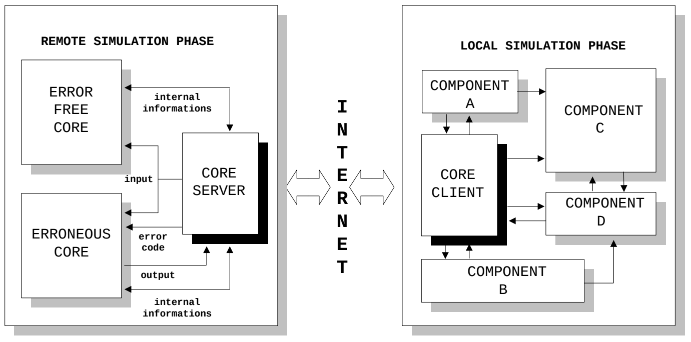

Figure 3: Testing architecture.

- **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. We simulate a traffic condition close to have a client in Europe and a server in north America, or vice versa.

5.2 Core Simulation

Table 1 shows the simulation times for 4 different configurations of core client and core server. The real, user and system times are the times provided by Unix command `time` to run 1000 times the program evaluating the 3-tap digital filter. The results confirm that the network traffic overhead has the biggest impact on the overall performance, as confirmed by normalized results in Table 2. In fact, the simulation through Internet is one order of magnitude slower than the same simulation run over a intranet.

Sim. type	Real Time	User Time	System Time
Local	9.94	7.80	0.94
Local+socket	24.95	10.06	5.10
Intranet	27.44	9.85	4.08
Internet	781.73	16.05	4.62

Table 1: Simulation times.

Sim. type	Norm. Real	Norm. User	Norm. System
Local	1	1	1
Local+socket	2.51	1.29	5.43
Intranet	2.76	1.26	4.34
Internet	78.613	2.06	4.02

Table 2: Normalized simulation times.

Sim. Type	VHDL	SystemC	Speed up
Local	105.43	9.94	10.6
Local+socket	456.02	25.95	18.2
Intranet	478.58	27.44	17.4
Internet	1518.07	781.73	1.9

Table 3: VHDL-SystemC speedup

The simulation architecture has been developed in VHDL too. The results in Table 3 show the speedup of SystemC simulations. The SystemC performance is one order of magnitude faster than VHDL for the three of the simulations. The quality of both VHDL and SystemC descriptions is comparable. Thus, the main reason of such performance speedup is that SystemC descriptions are compiled, whereas the VHDL ones are interpreted, thus slower.

The speedup for the Internet simulation is affected by the traffic overhead again, smoothing the performance differences between SystemC and VHDL.

5.3 Core Testing

The genetic TPG based on the exchange of query packets has been compared with a random TPG. The results are presented in Table 4. The best result has been obtained by the genetic RTP. The higher test generation time of the GA based TPG depends on the more complex test vector generation and the higher number of packet exchanged by the core client and the core server. Two different kinds of query packets have been defined by the core vendor to support the user core evaluation:

- `isObservable`, this packet returns to the core client 0 if there are no differences between the primary outputs of the erroneous and the error free core server simulations, otherwise a 1 is returned.
- `getHammingDistance`, the returned float value in the range $[0, 1]$ expresses the ratio of the core internal signals, which differs between the erroneous and the error free server core simulations.

	Random TPG	GA TPG
Num. of generation	-	100
Num. of sequence per error	500	25
Max sequence length	15	15
Num. of generated vectors	538339	519806
Num. of generated sequence	4725	37253
Test set length (sequences)	24	27
Simulation time	749.21	1246.38
Trasmited packets	538339	1039612
Coverage	88.6%	99.0

Table 4: Core testing results

No more query packets have been implemented to show how the overall coverage can increased just providing few IP unviolating information extracted by the defined query packets.

6. CONCLUDING REMARKS

A methodology for IP-Core simulation and testing has been presented . It allows the core vendor to make available very detailed core models without disclosing IP information. Moreover, it allows the core user to perform a validation test to verify the correct integration of the selected IP core into the core-based design under development. This is performed by generating test patterns for both the core and the surrounding logic. This main idea has been exploited by using two different modeling languages: VHDL and SystemC. Experimental results showed that, the use of SystemC produces better results in terms of simulation time and SystemC models can be more efficiently linked to interface libraries. Performance degradation of the remote test generator is dominated by the remote simulation process, thus it seems to be feasible whenever remote simulation is acceptable. However, waiting for an improvement of Internet transmission capabilities, the proposed technique can be directly used for local simulation, by providing detailed descriptions of the cores, even with faults, without disclosing IP information.

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

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

[4] A. Fin, F. Fummi, and G. Pravadelli. AMLETO: A Multi-language Environment for Functional Test Generation. Proc. IEEE ITC, pp 821-829, 2001.

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

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

[7] A. Fin and F. Fummi. A Web-CAD Methodology for IP-Core Analysis and Simulation. Proc. ACM/IEEE DAC, pages 128–133, 2000.

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

[9] M. Dalpasso, A. Bogliolo, L. Benini and M. Favalli. Virtual Fault Simulation of Distributed IP-based Designs, Proc. DATE, pages 99–103, 2000.

[10] F. Ferrandi, A. Fin, F. Fummi, and D. Sciuto. Functional Test Generation for Behaviorally Sequential Models. Proc. IEEE DATE, pp 403–410, 2001.

[11] F. Ferrandi, F. Fummi, and D. Sciuto. Design Verification of VHDL Specifications through Functional Testing. Internal Report 3–99, Universitá di Verona, 1999.