Reinvented Prototyping on VHDL
Serafin Olcoz, Luis Entrena, Luis Berrojo, Juan Goicolea
TGI
Velazquez 134-bis. 28006 Madrid

Abstract
Prototyping is an old technique commonly used to develop systems in general and electronic systems in particular. However, the use of VHDL for developing the hardware part of these systems allows the introduction of a new aspect of prototyping based on extending the common use of a VHDL simulator. This approach allows to use a virtual hardware model able to interact with the software of the system and even with other parts of it, such as the mechanical ones, emerging what can be considered a new technique that we named VHDL Virtual prototyping.

The paper presents the evolution of this technique throughout its application in several TGI projects: IDEAS\textsuperscript{G}, ECU\textsuperscript{O} and SIMAID\textsuperscript{C}.

1. Introduction
Since the early beginning it has been attempted to solve problems in development in the design life cycle with the technique of prototyping. Prototyping aids the discovery of missing and faulty requirements before design and implementation. The word "prototype" stems from the Greek and means primeval type, idol or model. In industry, it uses to mean the first model (e.g. vehicle or aeroplane) which was created and tested before the production of a series. In the literature, prototyping is separated into three types, [FLOY 84], according to its purpose:

- Explorative prototyping;
- Experimental prototyping; and
- Evolutionary prototyping.

This classification can be applied to any kind of prototyping. However, we are interested in presenting these concepts around the use of VHDL for prototyping. Such a technique is increasingly needed to satisfy the demands of embedded system design. We will show how the needs for VHDL prototyping and the solutions taken have been evolving throughout several TGI's projects.

The paper is organized as follows. Section 2 introduces the new ideas of VHDL Virtual Prototyping within the classical taxonomy of prototyping. Section 3 presents the evolution of VHDL Virtual Prototyping at TGI throughout the projects mentioned above. Finally, section 4 presents the conclusions of this work.

2. VHDL Virtual Prototyping
The main feature of VHDL, [IEEE 87] and [IEEE 92], is that it allows to describe the behavior of a system (in principle, a digital system) by means of simulating its response to certain set of stimuli during a predetermined window time. The simulation model of VHDL is defined by the discrete event simulation semantics of the language, [OLCO 94].

In order to perform a simulation, a description of the Unit Under Test (UUT) and the unit that generates the tests, the Test Bench (TB), is needed, see figure 1. The tests provided to the UUT use to be an abstraction of its environment. The idea is that these tests must help the designer on the UUT behavior verification.

A VHDL simulator creates a simulation model from the top level VHDL design unit of any description provided by the designer. So, in...
order to introduce the verification ideas presented above, both kind of units must be considered as part of a workbench (the VHDL design unit located on the top of the design hierarchy). see figure 2a. This approach allows to verify the behavior of the VHDL description of the UUT under the stimuli provided by the the VHDL description of the TB.

![Test Bench](image)

Figure 1.- Validation environment.

A complementary solution can be found if the VHDL simulation concepts presented above and the underlying modeling techniques are slightly modified. The idea is to consider VHDL as a language for prototyping.

![VHDL Workbench](image)

Figure 2.- VHDL Workbench.

It is possible to consider that the workbench represents a hardware board composed by several units, see figure 2b. The board-level simulation of this workbench presents the same results as the previous one, but now, there is not a clear distinction between which part of the workbench corresponds to the UUT and which part corresponds to the TB. With this approach, the part of the description that corresponds to the UUT only depends on the part of the system on which the designer wants to be focused on. The designer can identify the part of the model under study (dark units in figures 2a and 2b). Obviously, the rest of the description corresponds to the TB. So, when the interest of the designer changes, it is possible to consider a different part of the model as the UUT without modifying the description of the complete system. This means a different way of modeling with VHDL.

This approach allows to tackle board-level simulation with VHDL. Next step of this extension is to include an interface with the system in which the board is embedded (e.g., a mechanical system).

The system model can be considered as a prototype of the complete system. In fact, it is a virtual prototype of the system under study in the sense that it is just a model that is being executed by a computer. This simulated execution allows the designer to observe and modify the status of the system at any time without making special hardware (described by its corresponding VHDL units) for this purpose. The work is done through the simulator user and procedural interfaces.

This modeling approach allows to extend the domain of VHDL to describe not only digital circuits but also hardware and software parts of systems and even the interaction with mechanical parts of a more general system, see figure 3, as we will show in the following sections of this work.

![System Level Simulation](image)

Figure 3.- System Level Simulation.

The use of VHDL for system design represents a new expansion over its original goals, which is today possible due to the market acceptance of the IEEE standard, the enhanced performance of VHDL based tools (simulators, synthesizers, etc.) and intrinsic description capabilities of the language (abstract data types, support for different description styles at different abstraction levels, design library management and maintenance, design parametrization, etc.).
The purpose of the VHDL Virtual Prototyping can fit in more than one of the types of prototyping introduced above. Next subsections present the relation between the TGF's VHDL Virtual Prototyping and them.

2.1. Explorative prototyping

The aim of explorative prototyping is to create, if possible, a complete system specification to enable the developers to gain insight into the application area, to clarify different approaches to solutions with the user, and to make sure the planned electronic system in a given organizational environment can be realized. These are the objectives that a VHDL virtual prototype tries to cover.

However, the main disadvantage of VHDL Virtual Prototyping considered as an explorative prototyping technique can be the low performance presented by the simulator. The simulated execution of VHDL virtual prototype is never going to present the same performance as a real prototype of the same system.

With the use of the VHDL virtual prototype, real examples of the application can be run and the desired functionality is tried. In this context is not the quality of the construction of the prototype which is important, but the functionality and ease of modification and development within a short time scale.

The VHDL virtual prototype can be created jointly by developers (hardware designers) and users (system designers). It is made available in the life-cycle phases of requirement analysis and system specification.

The explorative prototyping aspects of VHDL Virtual Prototyping have following advantages:

- Requirement and specification problem/errors are recognized and dealt with very early (and consequently at lower cost). The functionality is more widely specified and tried by the user.

- The dynamics of the electronic system are also modeled and tested by means of simulation.

- Language and communication problems between developer and user are minimized, and resulting potential error sources are eliminated.

- The user validates his wants and needs of the planned electronic system by use of a working system model based on VHDL. This is a useful extension of static reviews of the written specifications of the requirements.

- The developer can define sensible steps for the electronic system together with the user, and can in this way avoid maintenance problems by use of a release step concept.

One disadvantage lies in the danger of generating permanent changes to the prototype and thus destablising project planning. If insufficient explanation is given (e.g., the VHDL description does not correspond with a path to silicon), the user could also assume that the prototype is very close to the finished product. However, this problem can be solved using VHDL Virtual Prototyping from the experimental perspective.

2.2. Experimental prototyping

The aim in experimental prototyping is a practical design specification which forms the basis for implementation. This means checking technical goals for practicalities. This is to prove the suitability and quality of the electronic system architecture, or inherent solution ideas and of the partitioning into system components (modules) as an experiment. An extension of this approach should consider hardware-software partitioning issues.

The explorative prototyping performed with the VHDL Virtual Prototyping can be considered as a horizontal use of the model. This way, when using the VHDL Virtual Prototyping as an experimental technique in top-down methodology, the approach can be considered as a vertical one.

This technique implies to refining the VHDL virtual prototype, or a part of it, from an abstract level of description to a level closer to the implementation. Considering VHDL virtual prototyping as an experimental technique is part of the current practise of VHDL for synthesis purposes.

On the basis of the first ideas about partitioning of the electronic system, a VHDL prototype is developed which allows the checking of the interaction between interfaces of the individual (hardware-software) components and of the flexibility of the system partitioning with
regard to extensions in the experiment. This can be considered the first issue to be covered by the experimental prototyping. Sometimes, it can be considered as the following step of applying VHDL Virtual prototyping as an explorative technique. However, this is not always feasible.

Within the framework of partial implementation of individual components, design and implementation requirements (e.g., performance, concurrency) can also be examined as well as their effects on system partitioning.

Refining the prototype mainly involves the hardware designer. The result of the experimental prototyping can be considered a part of a components library, supporting in this way the system and component design.

Advantages of VHDL Virtual Prototyping as experimental prototyping are:

• Early testing of the design and implementation requirements.

• Improved maintainability through the verification of modules with regard to the characteristics of extendibility and adaptability.

• Consistent and complete interface specification of system components.

• Extension to static reviews.

A disadvantage is the additional effort and cost involved, which can delay the project. The use of suitable tools reduces this drawback. There is a strong dependency with the synthesis tools.

2.3. Evolutionary prototyping

Evolutionary prototyping differs from the two types of approaches described above, in as much as we are dealing with an incremental electronic system development, i.e. a gradual development strategy. The VHDL virtual prototype is constantly modified to include new and discovered requirements.

The development (i.e., the VHDL synthesis) is not an isolated project, but runs parallel to the utilization of the electronic system in the application. There is no partition between prototype and product. In this kind of prototyping the prototype is, as a rule, not discarded but enhanced until the path to silicon is clearly reached. In contrast to throw-away and incomplete prototypes, we are dealing here with a complete prototype which contains all the salient functions of an evolutionary step.

The advantages of VHDL Virtual Prototyping as evolutionary prototyping are as follows:

• The specification problem of the classic life cycle (always working with an incomplete specification) can be simplified in this way. The VHDL prototype is a manageable part of the electronic system specification.

• Maintenance problems (difficulty in altering, and diverging and outdated documentation) are less severe because maintenance requirements are taken into account by the VHDL designer from the beginning.

• Quick validation of (hardware-software) requirements by the user is immediately possible on a working electronic system.

The disadvantages of applying this technique can be really dangerous:

• The design structure can be inadequate (patchwork). The problem of the complete electronic system design is not solved by this approach.

• Developing software and hardware parts of the system becomes a permanent alteration process which is no longer target oriented.

• Project management becomes much more costly.

Considering VHDL as an evolutionary prototyping technique is not suitable for those applications which are by nature subject to fast-changing requirements. Specially for those systems in which maintainability is an important aspect.

3. The evolution of VHDL Virtual Prototyping at TGI

The VHDL modeling techniques and the surrounding tools under development at TGI have evolved a lot since its beginning, in 1991. We have developed a design methodology based on VHDL Virtual Prototyping [OLCO 93]. This work was mainly done in the IDEAS project that finished in January 1994. This project provided us with one of the key factors for an effective system development: the first
VHDL models with different timing accuracy (instruction set, cycle, nano-second accurate). This project was performed using only commercial tools (Synopsis) and just a few and small C programs were developed for helping us to load the application software into the VHDL virtual prototypes.

In order to enhance the system design capabilities of the VHDL virtual prototypes we started last year the ECU and SIMAID projects. The first one is going to provide us with a new version of the VHDL library components based on the SPARC architecture, by the end of this year.

However, the main result of this project is the development of an environment for hardware/software development: the EDS (ECU Development System). A first demo of this environment was finished by November 1994.

Finally, an extension for mechatronics of the co-simulation capabilities offered by the EDS is under development in the SIMAID project.

Next subsections present in more detail the results of these projects.

3.1. A case of study: IDeAS Project

IDeAS project was devoted to develop and implement a SPARC® Integer Unit (IU), [SPAR 91]. This architecture was designed by Sun Microsystems Inc. and currently is controlled by SPARC International, independent and non-profit corporation. Moreover SPARC has been proposed as an IEEE standard (PAR. 1754). The possibility of licensing this RISC architecture was one of the main points that helped us to choose it.

The starting point of this project was the textual SPARC Instruction Set definition, [SPAR 91]. The result was a CYPRESS CY601, [SPAR 90], pin-compatible IU fabricated by ES2 in CMOS 0.7 μm technology. The SPARC compliance of this implementation was certified by SPARC Intl. (September 1994).

3.1.1. A SPARC µProcessor from VHDL to silicon

The SPARC Architecture specifies the structure of a system as composed by the following design entities, see fig. 4: a CPU (consisting of an Integer Unit, a Floating Point Unit (FPU), and a Storage Unit (composed of a Memory Management Unit (MMU) and a Memory Unit).

To have a better understanding of the SPARC architecture definition and to explore the design space we started by developing an explorative VHDL virtual prototype of a SPARC system including the above SPARC components and some Auxiliary Units: an Initialization Unit (Reset and Software loader) that sets the initial state of the different units and loads software programs in Memory unit, a Clock generator Unit, and an Interrupt Unit that models the asynchronous interaction with the external world, see figure 4. This prototype was instruction set accurate.

This prototype allows the hardware/software integration. The VHDL model is able to run programs written by the software programmer, and previously compiled by a commercial C compiler (i.e., CC or GCC). The machine code corresponding to the software is stored in a text file that is read by the VHDL Software Loader unit. In particular, this capability was used to run the Test Suites, [USSE 93] provided by SPARC Intl., in order to verify the SPARC Compliance Definition (SCD) of the VHDL descriptions.

![Figure 4.- Block Diagram of the SPARC Architecture modeled in VHDL.](image)

The result of this explorative prototype was a complete behavioral description of the whole SPARC-based system including the software connection. The next step was to refine the VHDL description corresponding to the IU, following an experimental prototyping approach. The message passing interfaces of the first VHDL prototype were refined to obtain a synchronous prototype (CY601 pin-compatible). This prototype was cycle accurate.

From here it was not possible to follow a pure top-down methodology. In fact we had to replace the IU description in the prototype by an RTL description designed totally from
scratch. We applied an evolutionary technique to refine this description. This stage was totally dependent on the synthesis tool available at TGI (Synopsys).

Finally, part of this evolutionary approach had to be done in Verilog because the VHDL sign-off was not supported by ES2 (January 1994). The synthesized prototypes were nano-second accurate.

The goal of this project, to implement a SPARC IU, was successfully reached. Besides we obtained a library corresponding to the different virtual prototypes as a by-product of the IU development. This library allows us to reuse the different components to create new prototypes and products. The VHDL virtual prototypes were also certified as SCD by SPARC Intl (January 1994).

### 3.1.2. Performance

Table 1 shows the experimental results obtained from executing the IDEAS virtual prototypes with different VHDL simulators available at TGI, [SYNO 94], [VANT 94], [LEAP 93].

<table>
<thead>
<tr>
<th></th>
<th>LEAPFROG</th>
<th>SYNOPSYS</th>
<th>VANTAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Behavioral Model</td>
<td>400 cps</td>
<td>240 cps (1)</td>
<td>120 cps</td>
</tr>
<tr>
<td>RT Level Model</td>
<td>(2)</td>
<td>50 cps</td>
<td>(2)</td>
</tr>
</tbody>
</table>

Table 1.- Simulation performance results of IDEAS project with several VHDL simulators. Note: (1) Interpreted Simulation. (2) Not Aavailable.

Performance is measured in cycles per second for behavioral model and RTL model in a Sun10 workstation. The performance obtained is sufficient to allow the integration of software and hardware for light software tasks. These figures are expected to rise dramatically in the future due to increased workstation performance and VHDL simulation advancements (multithreaded and hardware accelerated), allowing heavier software tasks.

### 3.2. A platform for Hardware/Software Development: ECU Project

A bottleneck in the design of a complex system, including hardware and software, is the lack of a common design environment and the difficulties to exchange the design information between both worlds, [GOIC 94]. In an attempt to solve this problem, we have integrated hardware and software debugging environments with a VHDL commercial simulator, see figure 5.

In ECU project we have extended the interface capabilities offered by a VHDL simulator in order to improve the monitoring and observing facilities and allow the communication with other tools. In particular the GNU software development toolkit [GNU 92] has been integrated in our VHDL Virtual Prototyping environment.

#### 3.2.1. The EDS Architecture

The EDS is based on a layered architecture, see figure 6. Each layer provides services to the layer above it by using the services provided by the layer below it.

![Figure 6.- EDS Layered Architecture.](image)

The topmost layer provides a user interface for debugging commands. The user commands are transformed internally into specific debugging actions at the tool layer. These debugging actions are independent of the type of debugging target. In the next layer, namely the target layer, the debugging actions are further decomposed into a set of target-dependent service requests. Finally, the communication interface encodes the target-dependent service requests and sends them to the VHDL simulator over the communication channel.
On the other end of the communication channel, the communication interface decodes the service requests and transfers them to the next layer. The Monitoring & Observing layer performs the monitoring and observing of the virtual prototype, independently of the VHDL simulator used. The next layer is specific of each VHDL simulator. It supports the requests of the monitoring & observing layer by using the C language interface provided by the VHDL simulator.

This architecture can be extended to connect the VHDL simulator to other tools over the same communication channel. Section 3.3 introduces the connection of a mechanical simulator.

3.2.2. Communication interfaces

Several communication mechanisms are available for the exchange of data between UNIX processes. [SUN 90]. Transport-level programming services are provided by socket-based interprocess communication (IPC) or stream-based Transport Library Interface (TLI). Higher level services are provided by the RPC/XDR (Remote Procedure Call/External Data Representation) interface. This latter interface hides the details of transport-level mechanisms from the user by providing a logical client-server communications structure.

In order to choose an underlying communication protocol for the development of the communication of the various processes involved in the Hardware/Software development platform, several considerations must be taken into account. First of all, we target a complete co-simulation environment, which involves several tools connected to a VHDL simulator. In its current state, only GDB is connected on-line to the VHDL simulator, but many other tools can be connected to it and we intend to do so. The connection approach must be kept as general as possible in order to allow future enhancements.

Secondly, some of the tools are not proprietary. In this case, they must be connected through commercially available procedural interfaces. These interfaces may provide only limited connection capabilities for our purposes, as we will show in section 3.2.4.2. Because of the limited controllability and observability capabilities provided by these interfaces, we were forced to create a special VHDL unit which communicates through its VHDL ports.

This unit is executed by the VHDL simulator. Because it cannot work as a server of a external tool, it prevented using remote procedure calls to implement the communication mechanisms. Finally, GDB itself provides a remote protocol. This protocol is very basic, as it is designed to allow communication with real hardware which may not support powerful communication mechanisms. Taking advantage of this remote protocol is straightforward, because GDB provides a reference remote communication stub for the GDB side and the debugging target side of the communication. Thus, the communication with the VHDL simulator is obtained by porting the target side stub to the VHDL simulator interface.

Because of the above reasons, the communication mechanism has been developed at the transport level using IPC. Stream sockets were preferred because the application required a reliable connection. RPC would have been a better option if we have had full, direct access to the VHDL simulator. However, it must be noted that this option is expected to exhibit lower performance.

3.2.3. The GNU interface

The GNU ToolKit is a complete set of software development tools. In the EDS we have developed an interface to the software debugger (GDB) and a link to GPROF to obtain profiles of the virtual prototype execution. It must be noted that the interface is not specific of these tools and can be ported to other software development tools. The following subsections describe the connection of GDB and GPROF with the VHDL simulator.

3.2.3.1. The GDB Target Layer

In a real target, the application software and the debugging-support software must be run on the same platform. Since the platform is shared, the execution of the application software must be interrupted to attend debugging actions. This mechanism produces a big overhead for some debug operations, such as variable tracing. Moreover, it may not allow to debug programs that involve specific interrupt control.

Software debugging in a virtual prototype target introduces several differences with respect to normal debugging in a real target. In a virtual prototype, the simulator takes care of debugging actions. To support debugging
actions in this way, we have to modify the GDB target layer. We have implemented a specific target backend for debugging with a virtual prototype. The main features of this backend are related to the following services.

- Breakpoint services
- Watchpoint services
- Saving the execution state

Breakpoint services are supported on the Monitoring & Observing layer without the need to modify the machine code of the program. The Monitoring & Observing layer will manage the list of breakpoints and will check if a breakpoint is reached every time the contents of the PC register (or any other signal specified for this purpose) change. This mechanism is external to the software being debugged and allows the system to attend all breakpoints, independently of the particular trap masking conditions that are set by the user or the system software.

Watchpoint services are usually supported through the breakpoint services. As in the case of breakpoints, simple watchpoints (watching for a change in a simple value) are supported directly at the Monitoring & Observing layer. The GDB is notified when an event has occurred that may trigger the watchpoint.

Saving the execution state at user request allows the user to continue debugging later from the saved state. This feature avoids repeating long simulation runs.

3.2.3.2. The GProf Link

Obtaining accurate execution profiles is one of the most important functions needed in Hardware/Software development. The GNU tools provide a version of a typical execution profile tool, GPROF.

A virtual prototype provides a more accurate and flexible profile mechanism than real hardware. First, the profile regions can be set dynamically, by inserting profile marks at the GDB front-end. Second, the execution of the application software on the virtual prototype need not be interrupted to attend profiling actions.

The profile-support software runs on the host and do not interfere with the flow of application software execution. Therefore, the profile information obtained can be as accurate as the virtual prototype is.

In the EDS, profile services are supported on the Monitoring & Observing layer. The Monitoring & Observing layer maintains a list of profile counters and is sensitive to profile marks. Whenever a profile mark is reached, the corresponding profile counter is incremented with the elapsed time from the last profile mark. The resulting profile information is later postprocessed by an enhanced version of GPROF that is able to deal with dynamic profile marks.

3.2.4. The VHDL simulator interface

The VHDL simulator interface is designed to support the monitoring and observing capabilities of the internal state of the virtual prototype that are needed for debugging purposes.

3.2.4.1. The Monitoring and Observing Layer

The Monitoring & Observing layer supports the control and the observation of the virtual prototype execution. It provides a conceptual model of the virtual prototype which abstracts the user from the implementation details of the VHDL model of the hardware.

The Monitoring & Observing layer provides the following services:

- Access to the values of hardware signals, register contents, memory contents, etc...
  These are mapped at this layer into the values of the VHDL objects of the VHDL implementation of the virtual prototype

- Access to specific services of the virtual prototype modules. These services are commonly used to set a module to a particular state. They are implemented by VHDL or C routines attached to the virtual prototype modules.

- Hardware and software breakpoint services.
  As explained before, breakpoint data is maintained at this layer. Breakpoint conditions may be a combination of hardware and software conditions.

- Collection of profile information
3.2.4.2. The Simulator Access Layer

The interaction with the virtual prototype is performed by accessing to the VHDL simulator. Most commercial simulators provide a procedural interface to which C programs can be linked. This interface is usually designed to allow mixed simulation of VHDL models and C models. However, these interfaces are not suited enough for allowing the communication with other tools.

The simulator access layer extends the raw communication capabilities of the VHDL simulators and provides a simulator independent interface to the monitoring and observing layer.

The kind of interaction needed to support the monitoring & observing services can be summarized in the following features:

1. Access to values of variables for both reading and writing.

2. Access to current values of signals for both reading and writing. Preferably, the access should be immediate, with no delta cycle between the access request and the moment the access to the signal is effective.

3. Access to the virtual/simulation time.

4. External calls to VHDL functions and procedures.

5. Allow C routines to be called based on several simulation conditions (signal events, the advancement of simulation time, the execution of simulation cycles, etc...). The capability of setting the sensitivity to these conditions dynamically is particularly desirable.

We have evaluated three VHDL simulators for the kind of support they provide for these features. The results can be summarized as follows (see Table 2):

1. VSS C Language Interface (CLI) supports 1, 2 and 3. Access to variables and signals require at least a delta delay to become effective. In order to have feature 5, a foreign architecture must be created. The sensitivity of a foreign architecture to signal activity or signal events can be declared statically. Also, timeout can be emulated.

2. Vantage Styx does not support feature 1. Feature 2 is supported for reading, but writing is only possible through emulation of a VHDL signal assignment statement. Feature 3 is supported. For feature 5, foreign models can be called at the end of a simulation cycle or on signal events.

3. Cadence Leapfrog C Interface supports 1, 2, 3 and 5. Access to variables and signals can be immediate. C routines can be called based on signal events, the advancement of simulation time or the execution of simulation cycles. All callbacks can be enabled, disabled, re-enabled and removed dynamically.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Synopsis</th>
<th>Vantage</th>
<th>Cadence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Signals</td>
<td>Delta-delayed writing</td>
<td>Emulated assignment</td>
<td>Immediate</td>
</tr>
<tr>
<td>Current time</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>External calls</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Foreign models</td>
<td>Architecture</td>
<td>Architecture, package, subprogram</td>
<td>Architecture, process, subprogram</td>
</tr>
<tr>
<td>Simulation sensitivity</td>
<td>Only emulated timeout</td>
<td>Yes</td>
<td>Yes. Dynamic sensitivity</td>
</tr>
</tbody>
</table>

Table 2.- Comparison of VHDL simulator interfaces.

The design of the VHDL simulator interface has been based on the basic common subset provided by these simulators, in order to increase the portability of the approach. To provide feature 5 in all cases, a new VHDL unit, called the Monitoring & Observing Unit (MOU), was designed and attached to the VHDL model. This unit is sensitive to the system clock signal. This way, the MOU is activated with the events in the system clock signal. The MOU then may access the simulator according to the user requests.

The lack of support for features 1 and 2 in the case of Styx difficulties the integration. This problem has been solved partially by modifying the code of the VHDL models in the following sense. A new signal is included in the model for each variable that needed to be observed. For software debugging, only the VHDL objects that represent memory contents
are required to be accessed. These objects can be declared using the \texttt{STYX\_MEMORY} package.

### 3.2.5. Performance

The impact of the GDB connection in the performance of the virtual prototype should be negligible for burst software execution. However, some VHDL objects are accessed every simulation cycle to check for breakpoint conditions. In the case of the Synopsys simulator, access to these VHDL objects makes the simulation about 30% slower.

For heavy debugging tasks that require intensive user interaction, the global performance of the EDS is comparable to that obtained with a real debugging target. This is due to the following:

- Hardware boards usually come with simple communication interfaces for remote debugging. Thus, communications are the debugging bottleneck and dominate over software execution.

- Some user level debug actions need to interrupt the execution of the application software as often as at every instruction. In this case, the overhead introduced by the debug-support software is very big.

### 3.3. An enhancement for mechatronics: SIMAID Project

The aim of the SIMAID project is to develop advanced 3D general purpose computer simulation tools, able to be used in design and production environments in the Mechanical Engineering field. An interface with commercial CAD and CAE in general and ESDA in particular is under developing. The ESDA interface is based on an extension of the communication capabilities offered by a VHDL commercial simulator, following the guidelines presented in the ECU project.

The objective is to be able to describe and simulate a mechatronic system by communicating existing simulators for the electronic and mechanical subsystems. The electronic control of the whole system is described in VHDL. To interface with the simulator of the mechanical part of the system, actuation and sensing channels are being defined which insure an adequate synchronization between both simulators. First results will appear by mid of 1995.

### 4. Conclusions

VHDL is becoming a good support for developing not only electronic systems but also ESDA tools in order to help system designers. These tools are based on the capability of VHDL for executing/simulating a virtual prototype of a system with several objectives (explorative, experimental and evolutionary prototyping).

The use of VHDL has helped us to close a semantical gap that exists today between system and circuit design and can significantly enhance the introduction of ASIC design in embedded systems.

The connection of software and hardware debugging environments with the VHDL simulator allows us to exploit all features offered by the different prototyping techniques. The communication between different simulators will allow us to tackle systems covering several domains (e.g., mechatronics).

Several types of models can be used to trade off between accuracy and speed. The capabilities of virtual prototyping are expected to increase quickly with the advancements in general computing and simulation.

In the future, we plan to continue extending the VHDL interface by means of a virtual prototype visualization tool on top of the monitoring and observing layer. This tool will provide an interface to the virtual prototype based on the abstraction provided by the monitoring and observing layer.

### 5. Acknowledgements

The authors would like to thank Dr. Denis Navarro and Prof. Armando Roy of University of Zaragoza for their cooperation in the IDEAS project. We would like also mention TGI's people also involved in this project: Ms. María-Angeles Salas, and Ms. Fabiola Bermejo.

### 6. References


[OLCO 94] Serafín Olcoz, José Manuel Colom, "The Discrete Event Simulation Semantics of VHDL", Intl. Conf. on Simulation and HDLs, pp. 128-134, January 1994, Tempe (Arizona), U.S.A.


