An Effective System Development Environment based on VHDL Prototyping

Serafín Olcoz, Luis Entrena, Luis Berrojo
Design Technology Department. TGI S.A.
Velázquez 134-bis. 28006 Madrid (Spain)

Abstract

This paper presents the use of VHDL prototyping as an effective basis for developing electronic (hardware and software) systems. VHDL simulation is the platform on which a distributed environment for debugging the hardware and the software, that is running on the VHDL prototype, is built. To do it, the natural monitoring and observing facilities provided by a commercial VHDL simulator have been enhanced. The new environment supports the development of the complete system from the different points of view corresponding to the involved domains. To ease the task of creating a virtual prototype of the hardware, an enhancement of the natural configuration capabilities of VHDL, is also provided by a complementary tool that helps to build the prototype.
The use of this new environment implies a new ESDA methodology.

1. Introduction

The main feature of VHDL, [1], is that it allows to describe the behavior of a system by means of simulating its response to certain set of stimuli during a predetermined window time. 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) or Stimuli Generator (SG), is needed, see figure 1. The tests provided to the UUT use to be an abstraction of its environment. Once the UUT has been tested, the SG has no more use in the design process of the electronic system.
However, in the VHDL Virtual Prototyping, [2-4], approach there is a complete working model of all hardware components, see figure 2. Because in system design it is not just the design, but also the environment, that counts, the specific environment of the system is also part of the prototype.

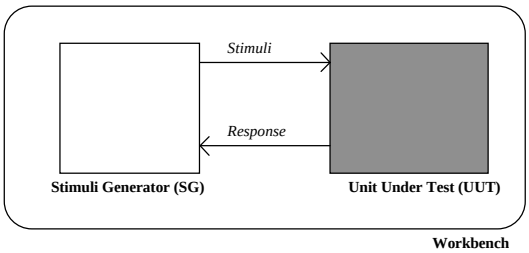

Figure 1.- VHDL Workbench.

In the virtual prototype the environment of the UUT is a specific functional description instead of an abstract one. The TB is fuzzed among parts of it. With this approach, the part of the VHDL description that corresponds to the UUT only depends on the part of the system on which the designer wants to be focused on.

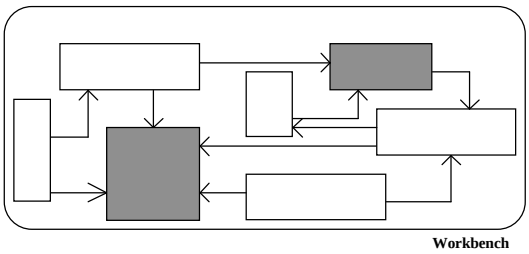

Figure 2.- VHDL Workbench.

The designer can identify the part of the model under study (dark units in figures 1 and 2). 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 way, VHDL is used as a language for Virtual Prototyping.
The essential difference between the verification environment, figure 1, and the virtual prototyping, figure 2, is that in a virtual

prototype there is no question of an attempt at completeness of the working model.
Completeness also means that the software part of the system, if any, is also included in the VHDL Prototype. The part of the SG that is not included in the hardware models of the system, figure 2, is kept as pure stimuli. These stimuli can be interpreted as the machine code and then, from the point of view of the system, the simulation of the VHDL Virtual Prototype corresponds to the "execution" of the software on the VHDL model of the hardware. This way, VHDL Virtual Prototyping can be considered as an effective technique to system prototyping.
In practice, this approach is supported by a special VHDL component that reads a VHDL file containing the machine code coming from the previous compilation/linking of the software.
This approach addresses the necessity to tackle on-chip system development in the same way on-board prototyping was classically used. Next step of this approach is to include an interface with the environment in which the electronic system is embedded (e.g., a mechanical system, [5]).
The paper is organized as follows. Section 2 presents an effective methodology for systems development based on VHDL Virtual Prototyping. Section 3 presents the system development environment that supports this new methodology. Finally, section 4 presents the conclusions.

2. An Effective Methodology for System Development

Figure 3 shows the classical development cycle followed by system engineers. After the hardware and software partitioning, the system designer disappears from this development cycle until all the components have been completely developed. The system designer appears later on to perform the integration and validation of these components against the system requirements.

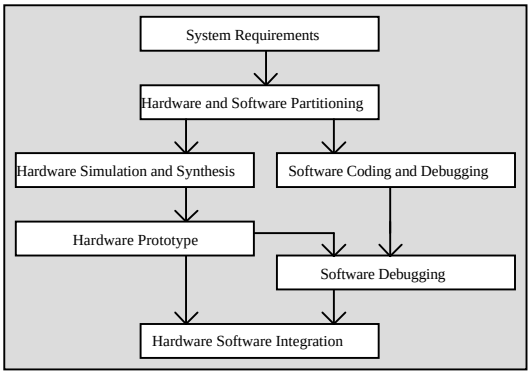

Figure 3.- Classical System Development Cycle.

This cycle originated from the existing gap between the hardware and software development cycles. This gap causes the serialization of the system debugging: the software can be debugged only when the hardware (free of bugs) is ready.
The VHDL Virtual Prototyping enables a more effective systems development cycle, figure 4. In this cycle, hardware and software can be concurrently debugged. In fact, both parts of the system are debugged on the same platform: the VHDL prototype. This improved cycle implies a new methodology for systems development. In this methodology, the hardware and software debugging can be done on the Virtual Prototype, closing the gap between both worlds.

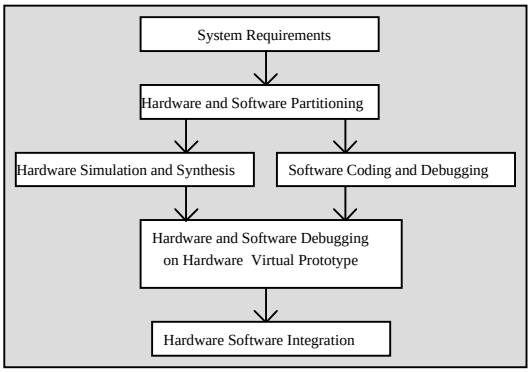

Figure 4.- Improved System Development Cycle.

This methodology relies upon two pillars: The availability of system components libraries and the extension of the existing tools in each domain to create the needed ESDA (Electronic Systems Design Automation) tools.

2.1. Library-Based Approach

The VHDL Virtual Prototyping enables a library-based systems-development approach. Libraries allow system engineers to rapidly build a simulatable prototype of their system.

Maximum productivity is achieved when the majority of the system building blocks can be selected from a library and quickly composed in a complete and working prototype.

To support this new methodology a set of VHDL models (SPARC®, [6],integer and floating-point units, memory management unit, cache memories, interrupt controllers, timers and counters, USARTs, IOs, and so on) are under development at TGI in several industrial projects: IDeAS[G],SMILE[S], ECU[O] and SIMAID[C].

The result of the IDeAS project was the silicon implementation, fabricated by ES2 in 0.7 μm CMOS TML technology, of a SPARC Integer Unit from a VHDL description, [7]. The chip and the VHDL Virtual Prototypes used to develop it, were certified as SCD by SPARC Intl (January 1994) using the same test suites, [8].

This library-approach promotes effective design reuse of the different parts of the system. These parts, corresponding to different domains of the system (i.e., the ASIC, the software, and so on), are developed in specialized environments. The availability of models in each domain is crucial in order to explore the system design space quickly and efficiently.

Finally, all parts must be integrated to compose the complete system and for this task new tools and a new ESDA development environment are needed.

2.2. ESDA Tools

It is not necessary to start these tools from scratch. The already existing tools for each specific domain (hardware, software, mechanics, etc.) can be extended in order to cover the system integration arena. The first needed extension is to allow the integration of other domains into every one of them. The second extension is to enable the communication between the domain-specific environments. These communication will provide the distributed environment for developing systems.

2.2.1. The Functional Extensions. The functional extensions to the existing design tools will allow to deploy the different design activities (simulation, debugging, profiling and so on) through the different domains (VHDL, Hardware and Software) of the system, figure 5.

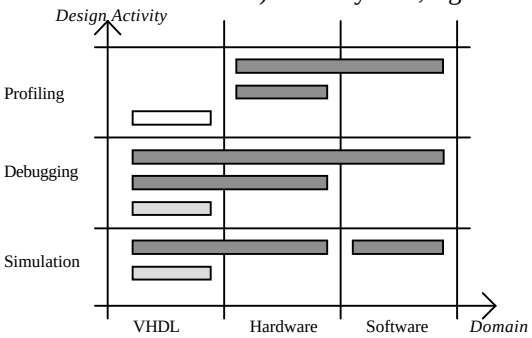

Figure 5.- Design Activity Deployment.

The VHDL simulator can be extended in order to be considered as a hardware simulator in which the complexity of the information coming from the VHDL description is filtered in order to keep just the relevant information to the hardware designer, not to the VHDL designer. Following this approach the execution on the simulated hardware can also be interpreted in software terms, extending this way the simulation to the software domain. These extensions can be performed by extending the monitoring facilities currently provided by a VHDL simulator to allow the hardware and software debugging.

The relationship between the hardware and software profiling can be also considered under this point of view. However, the extension of the profiling activity in order to cover the range from software to the VHDL models is not considered in this approach. This profiling activity is not currently applied to the VHDL models and it is not very interesting from the systems designer point of view.

2.2.2. The Communications in a Distributed Environment. To allow the design activity deployment through the different environments corresponding to the involved domains a communication mechanism must be defined. This mechanism must interconnect a distributed environment in a pipeline-like way that allows the translation of information between the

---

[G] IDeAS project was an internal project of TGI partially supported by GAME program (Special action of ESPRIT III to promote the microelectronics in Spain).

[S] SMILE is an OMI (Open Microprocessor Systems Inititiave) ESPRIT III Project coordinated by MATRA MHS.

[O] ECU is an OMI (Open Microprocessor Systems Inititiave) ESPRIT III Project coordinated by TGI.

[C] SIMAID is a (Computer_Integrated manufacturing) ESPRIT III Project coordinated by SIEMENS.

software, the hardware and the VHDL descriptions, figure 6.
However, this pipeline must not be straight to avoid the excess of information exchange through the network. To obtain better performance, the hardware-VHDL part of the pipeline has two ports to directly connect the VHDL and the hardware and also to allow the transparent communication between the software and the VHDL, figure 6.

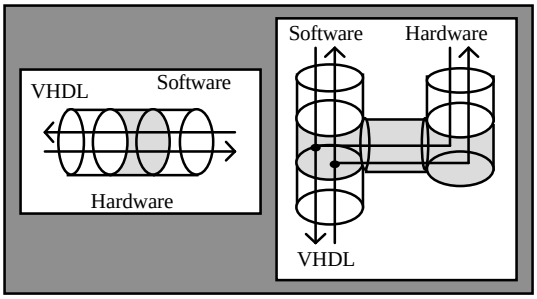

Figure 6.- Communicating a distributed toolset.

Since the software domain is on top of the hardware domain a debugging/profiling action in the software domain is transformed into a set of debugging actions in the hardware domain, which in turn are transformed into a set of VHDL debugging actions over the underlying VHDL models.
Next section shows how these communication mechanisms and the needed functional extensions has been developed in an industrial environment.

3. The System Development Environment

Figure 7 shows the main components of the ECU Development System (EDS) that is under development in the ECU (Embedded Control Unit) Project, [9]. In the EDS, the hardware is first configured with the help of a Hardware Configuration Builder (HCB) tool and the software is generated with the Software Development System (SDS). The software is loaded into the hardware to create the ECU Virtual Prototype (EVP). Then the dynamic behavior of the EVP can be executed, debugged and profiled from both the software and hardware points of view. To this purpose we can use software development tools (SDTs) and hardware debug and profile tools (HDPTs).

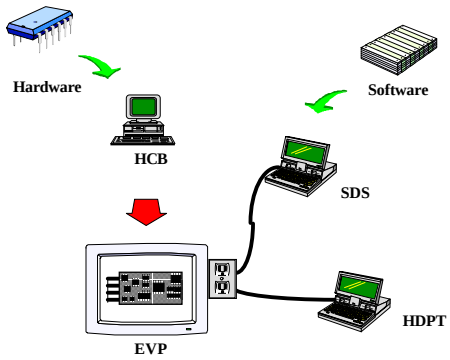

Figure 7.- The ECU Development System (EDS).

The following subsections presents a more detailed description of the EDS toolset, paying more attention to the HCB and HDPT description because the SDS, specially the SDT, was already presented in [4].

3.1. Hardware Configuration

In order to exploit the advantages provided by this systems design approach, VHDL Virtual Prototypes shall be generated easily and in a user-friendly manner. This is the task assigned to the *Hardware Configuration Builder* (HCB). This tool has two main features, figure 8: Creating a hardware configuration and allowing the rest of the EDS to analyze the hardware configuration chosen by the systems designer. The HCB can be considered as a layer on top of a Hardware Configuration Library (HCL).

3.1.1. Hardware Configuration Library. In hardware systems configuration, the hardware models are selected and instantiated taking into account the role they play in the system. Hardware components are previously classified according to their functionality. This functionality has to be recognized at the time of configuring a system. To configure the hardware, some additional information about the modules is required that is not usually provided within their VHDL descriptions. This information is stored in the HCL, which can be viewed as an extension of a VHDL library, see figure 8.
The HCL contains a catalog of all components, their corresponding views together with the path to the actual data files. The HCL has a HCL Browser that allows to navigate through the stored configurations or inside one of them.

The information regarding to the components is provided by the HCL user/supplier. Each HCL object is defined by a name and a set views:

- A VHDL view, that defines the VHDL model of the object. This view is the core view to which the others views have been added. Some times, a component can have associated more than one VHDL view (i.e., behavioral, synthesizable, gate level, etc.).
- A hierarchical hardware view, that defines the subelements that integrate this component and provides the meaning of the object from the hardware point of view, such as registers, bus sizes and so on. This view can be considered the result of filtering the VHDL information.
- A Testing view, that defines the test environment of the object. Provides the VHDL description corresponding to the test bench of the component.
- A schematic view, that defines the schematic symbol associated to the object. This view can be used by a graphical entry and is also linked to a structural VHDL description of the component.
- A connection view, that defines the connectivity properties of the object.
- An implementation view, that defines implementation properties of the object, such as manufacturing technology, frequency range, area and so on. This view could also be decompose into several other views.

All hardware components have at least a VHDL view. This view is needed to provide a link with the rest of the EDS tools, see figure 8.

In the HCL, the components are organized according to their configuration properties with an object-oriented approach. A hardware component view can be a file belonging to a commercial database.

3.1.2. Hardware Configuration Creation. Hardware alternatives are configured by creating configuration templates and instantiating the modules in the HCL. The process to create a new configuration or a new module may involve several steps:

1. Creation of a configuration template. A configuration template is an incompletely specified configuration, that can be further specified in several ways to achieve different versions with different properties.
2. Instantiation and configuration of the modules in a configuration template.
3. Connection of the modules.

3.1.3. Hardware Configuration Analysis. When required, the definition of a hardware component or a complete hardware configuration can be extracted from the HCL to generate a VHDL description that is downloaded into the VHDL Library. The corresponding VHDL configuration will constitute the EVP that can be analyzed by means of the other tools of the EDS. As result of this analysis, the configuration can be validated or sent back to the HCB in order to modify it in an iterative way.

3.2. The Extended VHDL Simulation

In order to support system development, the VHDL simulator has to be extended to interface with the various domains involved in a system, and to communicate with other tools in a distributed environment.

In the EDS, three main domains are handled, figure 9: VHDL domain, hardware domain and software domain. In addition, other domains considered, such as the mechanical domain, require special adapters to the VHDL domain.

In each domain, the user needs tools to analyze the behavior of the entire system. This function is typically assigned to debuggers and profilers. Thus, we need a software debugger, a hardware debugger and of course, a VHDL debugger.

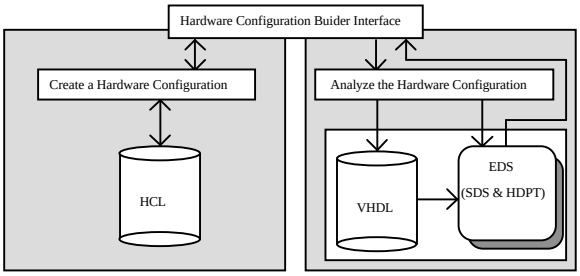

Figure 8.- The Configuration Creation and Analysis Flow.

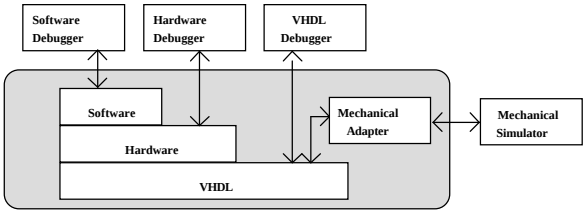

Figure 9.-The Functional Model Mapping

The internal structure of the extension of the VHDL simulator (EDS Core) is shown in figure 10. It consists of two basic modules: the Extended Simulation Module, consisting of the Model Mapping (MM) and the Monitoring & Observing, and the Communication Modules, consisting of the Communication Server and the Communication interface. he Communication Module provides the simulator with an IPC mechanism. This extension includes a communication server that is able to manage multiple requests from different tools (e.g., GDB, Mechanical simulator).

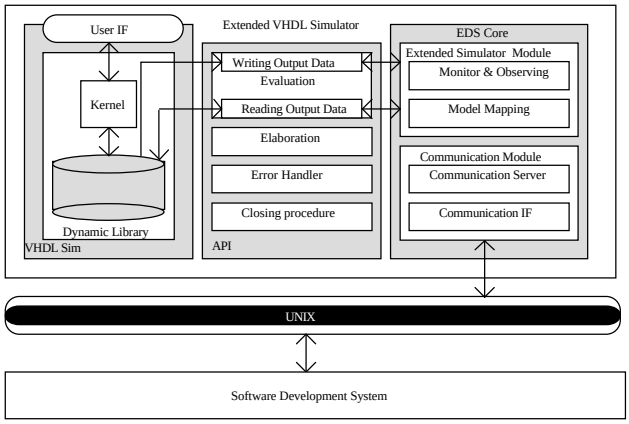

Figure 10.-The VHDL Extended Simulator

The MM layer performs the mapping between the hardware/software worlds and the elements in the underlying VHDL models, figure 10. When a request involving hardware items arrives to the MM, the request is transformed into a request (or a set of requests) to the MO. This request(s) are in terms of the underlying VHDL model.

The Monitoring & Observing supports the control and the observation of the virtual prototype execution. At this layer, the virtual prototype is seen from the VHDL point of view. The services provided by this layer are similar to those at the MM, but at a lower level of abstraction: Access to the values of VHDL objects; Access to C or VHDL routines that implement specific unit services; VHDL breakpoint services; Collection of profile information.

The EDS Core interfaces with the simulator by means of the application procedural interface (API). Most commercial simulators provide an API to link C programs to the VHDL models. Although this interface is usually designed to allow mixed simulation of VHDL models and C models, we have decided to use it for a different purpose: to extend the functionality of the simulator.

A discussion about the kind of support provided for the these features by several commercial VHDL simulators can be found in [2-3].

3.3. The Software Development System

The SDS consists of a complete set of software development tools (compilers for various languages, assemblers and a linkers/loaders, debuggers, and profiling tools, [10]) integrated into the EDS. In the ECU project we are extending these tools to take advantage of virtual prototyping. The extensions are of two types: (a) improved support of existing functionality; (b) new functionality that is possible when the software is executed in a virtual prototype.

SDTs need the addition of specific software to control and observe the execution of the application software on the target (local/remote) hardware. This specific software is executed by the target along with the application software. Since the platform is shared, the execution of the software must be interrupted to attend debugging actions and to collect information to be postprocessed later on. This is the typical intrusive software debugging approach.

In a virtual target, the development-support software is part of the extended simulator (EDS Core) instead of an extension of the application software. This approach allows to control and observe the execution of the software on the virtual prototype in a non-intrusive manner, [4].

3.4. Hardware Debug & Profile

The Hardware Debug & Profile (HDPT) provides a hardware model of the virtual prototype which abstracts the designer from the details of the VHDL description. The hardware model is defined upon configuration by the HCB.

The HDPT, through the extended VHDL simulator, 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 description 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 hardware state (e.g., reset state). They are implemented by C routines attached to the virtual prototype components.
- Control of the execution by setting breakpoints based on hardware conditions.

The HDPT is the debugger/profiler interface for the hardware domain. The hardware designer is allowed to control and observe the execution of the Virtual Prototype from the point of view of the hardware of the system.

4. Conclusions

The use of VHDL for prototyping represents a expansion over its original goals, that allows an effective system development methodology. This methodology 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 parameterization, etc.).

The execution performance obtained, [2], is sufficient to allow the integration of software and hardware for light software tasks.

This approach is based on the availability of system components (VHDL, hardware and software) libraries and the extension of the existing tools in each domain to create the needed ESDA (Electronic Systems Design Automation) tools.

With this approach, an effective environment for developing systems based on VHDL Virtual Prototyping has been presented.

References

[1] IEEE Standard VHDL Language Reference Manual, IEEE Standard 1076-1993, The IEEE, New York, NY,1993.

[2] S. Olcoz, L. Entrena, L. Berrojo, J. Goicolea, "Reinvented Prototyping on VHDL". VIUF Spring Conference. San Diego. April 1995.

[3] S. Olcoz, L. Entrena, L. Berrojo, J. Goicolea. "Prototyping: the Bottom Line of VHDL System Simulation". Proc. Workshop on Libraries, Component Modelling and Quality Assurance. Nantes (France). April, 1995.

[4] S. Olcoz, L. Entrena, L. Berrojo. "VHDL Virtual Prototyping". 6th IEEE Int´l Workshop on Rapid System Prototyping. Chapel Hill, NC, June 1995.

[5] S. Olcoz, L. Entrena, L. Berrojo. "A VHDL Virtual Prototyping Technique for Mechatronic Systems Design". International Conference in Recent Advances in Mechatronics. Istambul, August 1995.

[6] "SPARC Architecture". SPARC International Inc, Prentice Hall, 1991.

[7] S. Olcoz, J. Goicolea, "VHDL modeling *SPARC Architecture*", EuroSPARC-92, Madrid, October, 1992.

[8] R. Usselmann. "MicroSPARC Instruction Set Architecture Test Suite Manual". SPARC International, Inc. Rev. 2.0. March, 1993.

[9] J. Goicolea, R. Guzmán, M. A. Salas, S. Olcoz, D. Navarro, A. Roy, "*System Designers Approach to the Development of Embedded Systems based on VHDL*", Proc. of the APCHDL-94, pp. 135-138, Toyohashi, Japan, October, 1994.

[10]GNU User's Guide. Cygnus Support 1992.