Integrating and Verifying Intellectual Property Blocks using Platform Express and ModelSim

Mardav Wala
Electrical & Computer Engineering,
University of Tennessee
Knoxville, TN, 37996-2100
mwala@tennessee.edu

Don Bouldin
Electrical & Computer Engineering,
University of Tennessee
Knoxville, TN, 37996-2100
dbouldin@tennessee.edu

Abstract
Platform-based design is a proven method for minimizing the time and risks involved in designing and verifying a system-on-chip (SoC). Our experiences in using Mentor Graphics' Platform Express (PX) tool to design an SoC platform will be described in this paper. The tool allows Intellectual Property (IP) developers to add their IP cores and enables system designers to explore different system architectures rapidly by integrating pre-installed IP components. Thus, the designers work directly at the component level instead of the usual register-transfer description level. We have successfully added an open IP core to the PX component library and have used it in our system design. Also, since we intend to add only freely-available IP cores—or the ones that are internally generated—to expand our component library, it can be shared with anyone using Platform Express. This paper discusses integrating IP components to build a baseline platform for future, derivative SoC designs as well as verification for correctness using ModelSim. The complete IP integration and system design flow is outlined along with a workaround for designs involving IPs specified using both, VHDL and Verilog. Pointers are also provided as recommendations for first-time users to avoid possible errors.

Keywords
Platform Express, Platform-based Design, System-on-Chip,

I. INTRODUCTION
It is not uncommon for an SoC to contain tens of millions of gates consisting of processor cores, on-chip interconnects, specialized DSP units and analog components. Hence, it is a challenging task for the chip design team to design all the components completely from scratch. Moreover, a product launch deadline must be met. Circumstances such as these have resulted in a trend towards increased intellectual property (IP) reuse, which requires little or no modification of the reusable IP blocks. A major benefit of this approach is that properly defined IP blocks can be reused across multiple designs. Figure 1 illustrates the role of IP reuse methodology in closing the design productivity gap.

Many educational institutions have been able to acquire comprehensive electronic design automation tools via generous university programs supported by Synopsys, Mentor

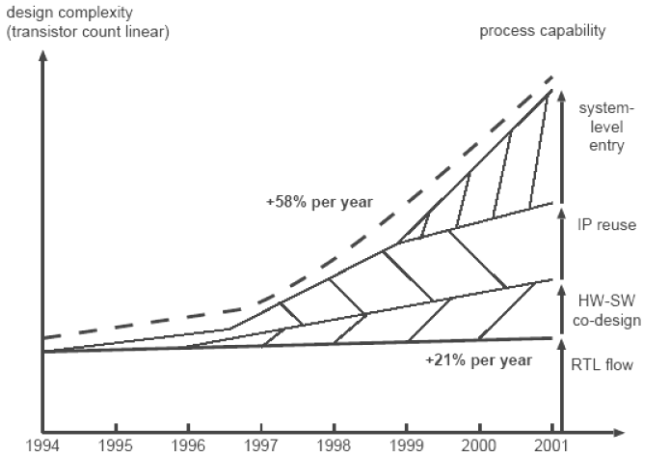

Figure 1: Bridging the design productivity gap. [1]

Graphics, Cadence Design Systems, etc. However, what is generally missing is the availability of affordable IP blocks, which are needed to build a large SoC and its derivatives using those tools. The graduate program in the Electrical and Computer Engineering department at the University of Tennessee [2] spanning four semesters has addressed this issue by offering courses intended to equip individual students with the understanding of design *for* reuse and a team of students with the understanding of design *with* reuse.

In the spring of 2003, the graduate class consisting of sixteen students was split into groups of twos and fours and each group was assigned the task to simulate, synthesize and test a single IP core—either internally generated or obtained for free. The intention was to verify each IP block for functionality before integrating it with the open core Volunteer SoC platform [3].

When the SoC platform was completed in August 2004, the next step in the design process was to raise the level of abstraction through which the platform designers integrated the IP blocks. This way the designers could work directly at the component level rather than at the VHDL-entry level and they could also rapidly identify, select and integrate (or remove) the required IP block into (or from) their design.

The idea thus conceived, was the major motivational factor for taking the Volunteer SoC project to the next level and selecting Platform Express™ (*PX*) [4], an EDA tool by Mentor Graphics® for this purpose. Working from two

different design perspectives, we have been successful, both as an IP integrator and as a System Designer in using *PX* for designing SoC platforms [5].

In this paper, we describe the complete IP integration and platform development process using Platform Express. In section II we present an overview of the *PX* environment. Section III illustrates our complete design flow and explains IP integration using *PX*. In section IV we perform the role of a System Designer to design a system platform using the integrated IP. We conclude in section V.

II. THE PLATFORM EXPRESS ENVIRONMENT

Platform Express is an electronic design automation tool, which allows the system designer to build a system design quickly using the components that the IP integrators have created from their own hardware designs. The *PX* interface (see Figure 2) presents users with a graphical interface and allows them to enter designs as block diagrams by selecting processors, memories and peripherals from various libraries of pre-installed IP components. *PX* offers automatic bus decoding and automatic bus and interrupt-bridging.

The tool also allows creating and implementing user-defined libraries and provides a built-in IP meta-data generation interface—PxEdit—to realize that objective. The IP meta-data describes the characteristics of the IP components; this includes information about invoking simulation and verification environment that the component requires, and also allows setting up and logging of design configuration. Platform Express uses the open source Extensible Markup Language (XML) as the meta-data language to describe the IP components for integration with the *PX* component libraries. The XML meta-data, in association with the *PX* tool also initiates other code written in Java, VHDL and Verilog that allow components to function in a design.

The *PX* tool speeds up design creation by presenting the significant design elements in detail. Once the design is created, *PX* provides tools for automating the build process. The resulting build files also include ones that could be used for validation with *Seamless Co-verification Environment (CVE)*.

III. INTEGRATING IP COMPONENTS

Starting with the compiled HDL model of the IP component, the IP integrator can install ready-to-use modules by following the steps outlined in Figure 3. Note that the two HDL flows in the illustration are interchangeable. The raw IP can be described using either VHDL or Verilog. RAMs are added if the IP needs memory for storing pre- and post-processed data. A bus-compliant top-level wrapper is defined for connecting components together. The *PxEdit* tool [6] supplied with the *PX* software is used to generate the IP's XML meta-data file before installing it into the component library. The tool allows the user to enter data for standard elements of the component and then generates a valid XML file with the recorded information. The generated XML file can be modified according to needs outside of *PxEdit* using a simple text editor.

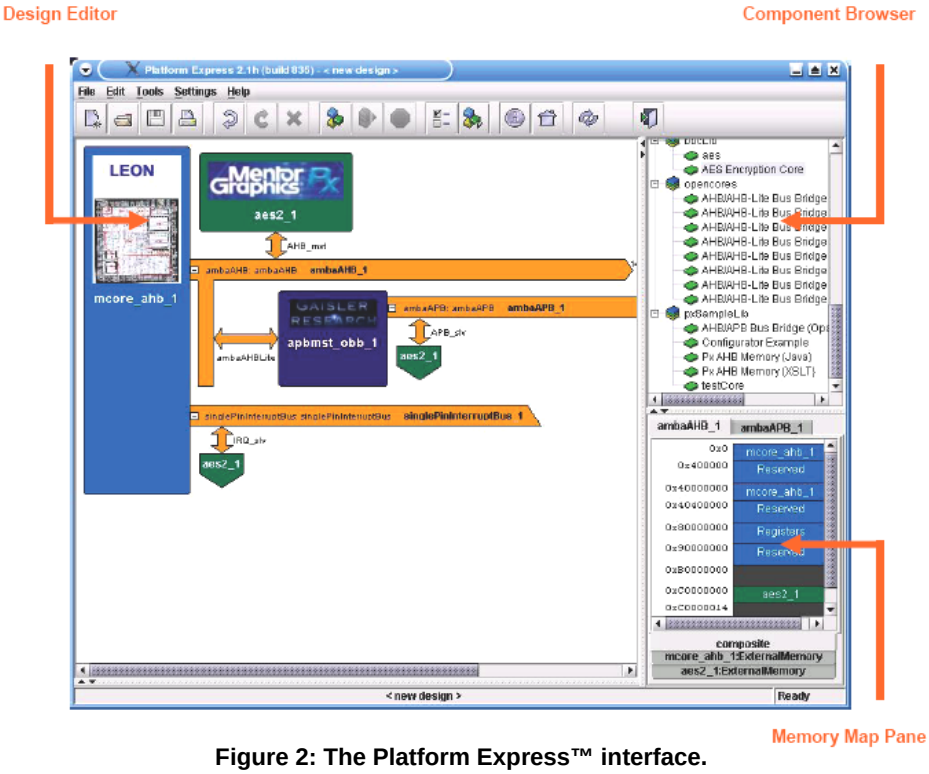

Figure 2: The Platform Express™ interface.

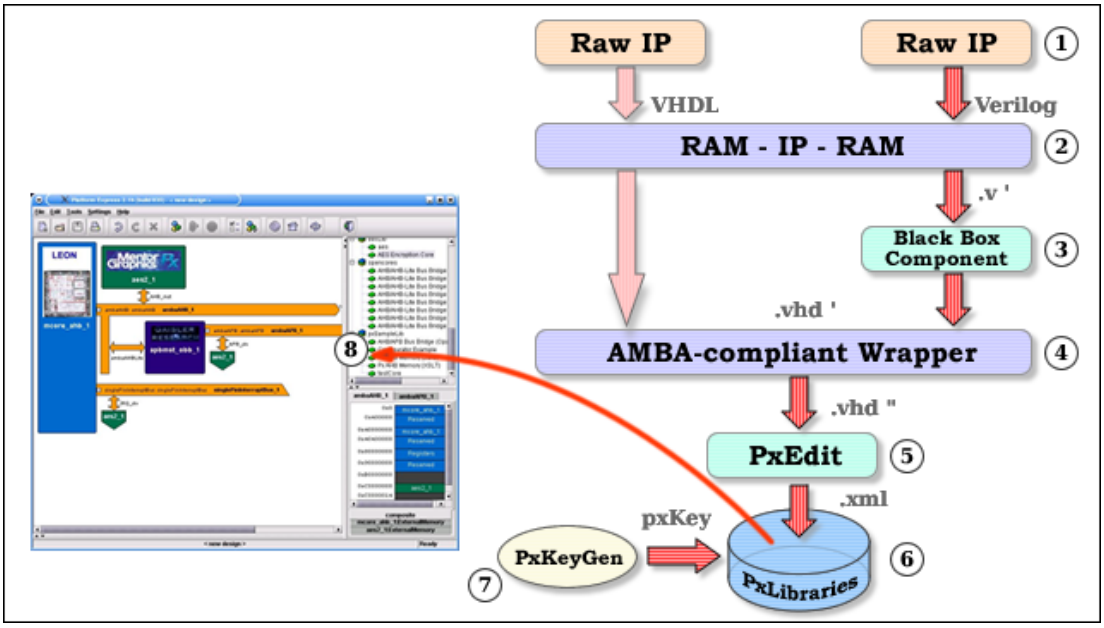

Figure 3: IP integration & system design using Platform Express.

A key is generated to protect the IP from modifications and also so that it appears in the component browser of the *PX* tool. The meta-data file contains all the information: component hierarchy, associated HDL file sets, bus configuration and choice of verification environment.

A black box component (step 3) is required when the top-level wrapper is written in an HDL other than the one used to define the raw IP. For example, in a case where an in-house bus-compliant module is written in VHDL for a Verilog IP module obtained from somewhere else. When a component is described as a black box, only the bus interface signals are imported into the XML file. The information about the lower-level sub-modules, specified using a different HDL, is kept hidden from the *PX* tool. This workaround is useful to avoid errors generally encountered while compiling designs involving components specified in multiple HDLs. Figure 4 shows the component design browser with an AES Rijndael IP component [7] installed as a black box under the *bbcLib* component library and its unmodified version under the *VOLIPository* component library.

IV. DESIGNING IP-BASED SoC PLATFORMS

Designing SoC platforms using *PX* is relatively easy once the required IP is installed within the *PX* component libraries. System developers can choose any of the visible components in the component browser for their designs. However, not all component libraries will be visible when starting a new design. This is because initially, *PX* shows components that can be dragged onto the design editor pane, i.e., CPU cores only. *PX* updates the component browser once a CPU core is selected and only shows the compatible bus architectures and IP cores. When IP cores are added into the design, *PX* also checks for inserting bus bridges wherever necessary, thus allowing the designer to focus on product enhancement and differentiation.

The process of compiling the design is known as 'building' in *PX* terminology. This is performed using a hardware simulator such as *ModelSim* or *NCSim* and has to be specified in the IP meta-data during component integration. Building a design from *PX* invokes *ModelSim* (or *NCSim*) in the background and the compilation messages are displayed on the *PX* tool's output pane as well as on the command line terminal.

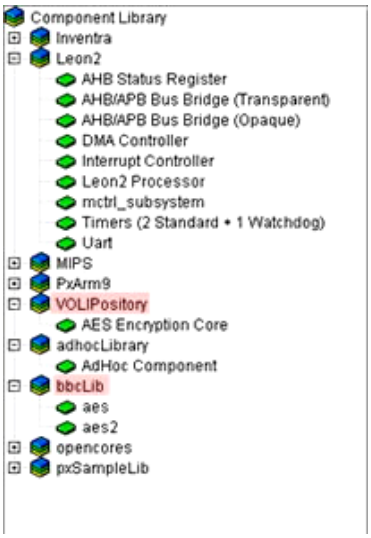

Figure 4: Updated component browser

This process also generates a *build.xml* file containing the command line equivalent instructions for compiling HDL scripts, which is used to invoke *Seamless CVE*. This is known as 'executing the build'.

The *CVE* session brings up the *ModelSim* tool and its *Waveform Viewer* interface. Users can now examine internal signals and check the functional operation of the design. Additionally, It also starts the *XRAY Debugger* (if a Processor Support Package (PSP) exists for the selected CPU core) to enable users to monitor the execution results of CPU instructions and the changes in contents of the internal registers.

Building a design involving black boxes, compiles without any errors. However, with no substantial information regarding the underlying sub-modules in the black box component, it is not possible to use a hardware simulator for checking functional correctness of the component. This problem can be overcome by adding a few intermediate steps between 'performing a build' and 'executing the build'. The *build.xml* file generated after building the design is modified using a text editor by adding the proper HDL compile arguments. Then, the Java equivalent of *make* command – *ant* [8], which uses XML-based configuration files to execute tasks – is executed from the command line terminal to compile the entire mixed-HDL design using the modified *build.xml* script. Hence, in this exercise we are building the design from *PX*, then executing *ant* from the command line (outside of *PX*) and finally invoking the *Seamless CVE*, again from *PX*. It should be noted that *ant* is completely independent of the *PX* tool.

V. CONCLUSIONS

In this paper, we discussed using Platform Express in conjunction with *ModelSim* for IP integration and platform design. We addressed the topic from the IP integrator's as well as a system designer's viewpoint. We identified issues regarding mixed-HDL compilation and validation and provided workarounds to overcome them. A sample checklist is illustrated in Figure 5 to help first-time *PX* users in their design tasks. In our experiences, *Platform Express* is an excellent tool to rapidly explore different system architectures and to rapidly evaluate various IP components before incorporating them into a design. Our findings may enable academic institutions to investigate platform-based SoC design opportunities and may allow sharing of internally-developed IPs.

VI. ACKNOWLEDGMENTS

The authors thank Mentor Graphics Corporation for the support of the laboratory with its design tools. Thanks also to Mr. Tomas Thoresen and the entire technical staff at Mentor Graphics for providing technical support for Platform Express.

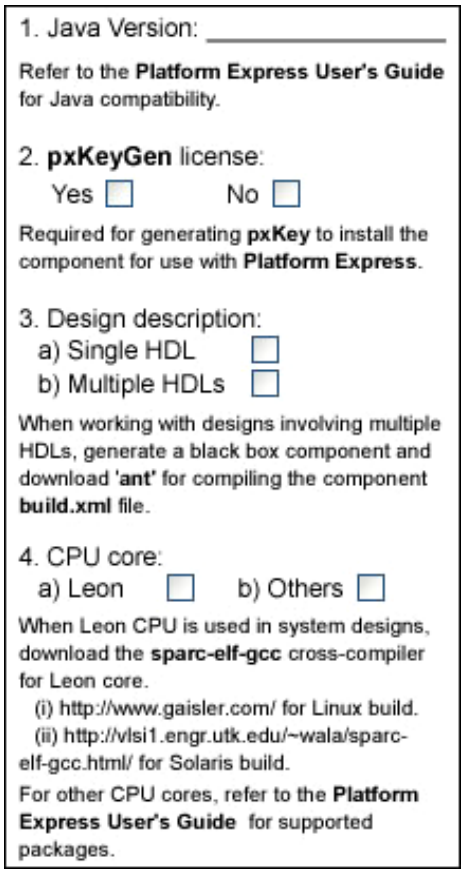

Figure 5: Platform Express checklist.

VII. REFERENCES

[1] Borel, J., "Design Automation in MEDEA: Present and Future," IEEE Micro, Vol. 19, No. 5, pp. 71-79 (1999)

[2] Bouldin, D., Microelectronic Systems Courses, University of Tennessee. Available at <http://vlsi1.engr.utk.edu/ece/bouldin_courses/>

[3] Bouldin, D., and R. Srivastava, "An Open System-on-Chip Platform for Education," Proceedings of 2004 European Workshop on Microelectronics Education (EWME), Lausanne, Switzerland (2004).

[4] Platform Express Client, Mentor Graphics Corporation. Available at <http://www.mentor.com/platform_ex/>

[5] Wala, M., "Using Platform Express for System-on-Chip Design," Masters' Thesis, University of Tennessee, Knoxville, TN (2005).

[6] Mentor Graphics Corporation, "Platform Express Integrator's Guide." Available at <http://www.mentor.com/products/embedded_software/platform_baseddesign/platform_express/upload/pxCompIntGuide.pdf/>

[7] OpenCores. Available at <http://www.opencores.org/>

[8] Apache Ant. Available at <http://ant.apache.org/>